A2.42 Random Forest Random Forest Without Class Weight Balancing

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

    from sklearn.ensemble import RandomForestClassifier
    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 Random Forest model with class weight balancing
    rf_weighted = RandomForestClassifier(max_depth=2, random_state=0, class_weight=class_weight)

    # 3. Train the Random Forest model with class weight balancing
    rf_weighted.fit(X_train, y_train)

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

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

    # 6. Plot the confusion matrix for the Random Forest model with class weight balancing
    confmat_weighted = confusion_matrix(y_test, rf_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 Random Forest with Class Weight Balancing")
    plt.show()
```

The given code addresses the problem of class imbalance in a dataset by applying weight balancing when training a random forest classifier. This is essential in scenarios where the classes in a dataset are unevenly distributed, as models can often become biased towards the majority class. This section computes and then visualises the confusion matrix for the test data. A confusion matrix provides a clear view of the model's predictions against the true labels, making it easy to spot misclassifications. In conclusion, the code provides a method for addressing class imbalance in a dataset by training a weighted random forest classifier. The evaluations and visualisations further assist in understanding the effectiveness of this approach.

1. **Introduction Comment**:
   ```python
   # Applying Weight Balancing to address the class imbalance in the dataset
   ```
   A comment explaining that the code will address class imbalance by applying weight balancing.

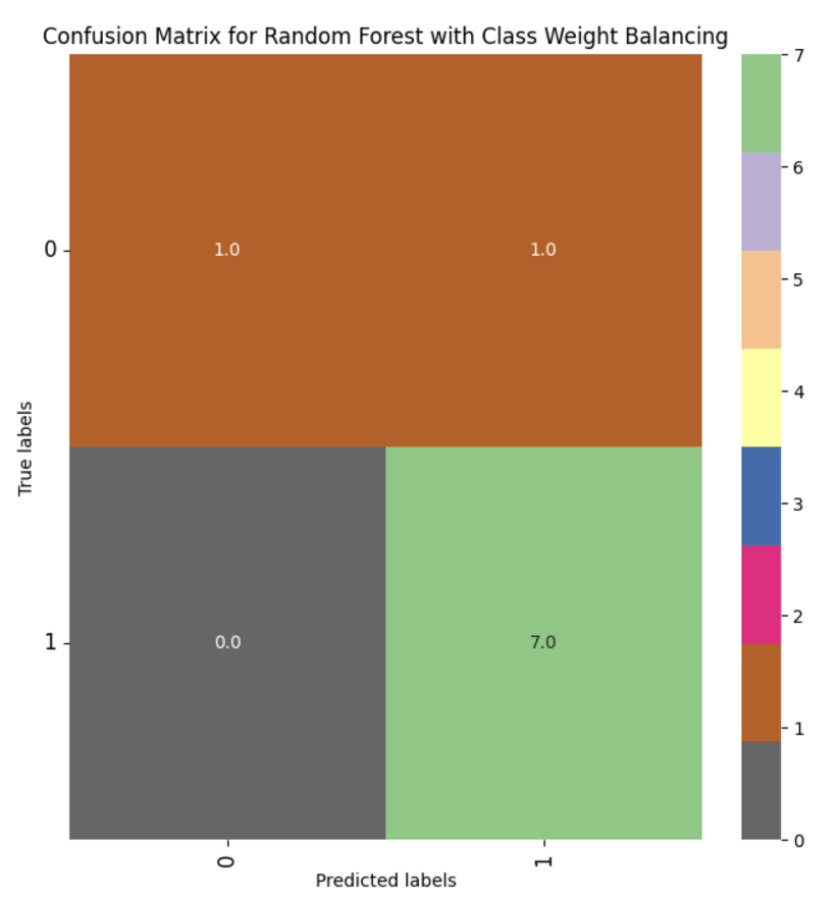
2. **Imports**:
   ```python
   from sklearn.ensemble import RandomForestClassifier
   from sklearn.metrics import classification_report, confusion_matrix
   import seaborn as sns
   ```
   The necessary libraries and functions are imported. This includes the random forest classifier, functions for evaluation metrics, and seaborn for visualisation.

3. **Class Weight Balancing Calculation**:
   ```python
   # 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))}
   ```
   Here, class weights are calculated based on the frequency of each class in the training dataset. It computes the inverse of the class proportions. These weights will be used to tell the random forest algorithm to "pay more attention" to the minority class.

4. **Creating and Training the Random Forest Model with Class Weights**:
   ```python
   rf_weighted = RandomForestClassifier(max_depth=2, random_state=0, class_weight=class_weight)
   rf_weighted.fit(X_train, y_train)
   ```
   A random forest classifier is initialised with the class weights provided. It is then trained on the training data.

5. **Evaluation on Test Data**:

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

           0       1.00      0.50      0.67         2
           1       0.88      1.00      0.93         7

    accuracy                           0.89         9
   macro avg       0.94      0.75      0.80         9
weighted avg       0.90      0.89      0.87         9

Random Forest 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
```



Interpretation of the Random Forest model with class weight balancing, both for the test set and the training set:

Test Set:

Metrics for Class `0`:

- **Precision**: 1.00 — When the model predicts an instance as class '0', it is correct 100% of the time.
- **Recall**: 0.50 — The model correctly identified 50% of the actual instances of class '0'.
- **F1-Score**: 0.67 — The harmonic mean of precision and recall for class '0'.
- **Support**: 2 — There were 2 instances of class '0' in the test set.

Metrics for Class `1`:

- **Precision**: 0.88 — When the model predicts an instance as class '1', it is correct 88% of the time.
- **Recall**: 1.00 — The model correctly identified all of the actual instances of class '1'.
- **F1-Score**: 0.93 — The harmonic mean of precision and recall for class '1'.
- **Support**: 7 — There were 7 instances of class '1' in the test set.

Overall Metrics:

- **Accuracy**: 0.89 — Out of all predictions made by the model on the test set, 89% were correct.
- **Macro Avg**: 0.94 (precision), 0.75 (recall), 0.80 (F1-Score) — Average scores for both classes, treating them equally.
- **Weighted Avg**: 0.90 (precision), 0.89 (recall), 0.87 (F1-Score) — Average scores weighted by the number of true instances for each label.

Training Set:

The classification report for the training set indicates a perfect performance, with all metrics at 100%. This suggests the model has learned the training data completely.

Overall Metrics:

- **Accuracy**: 1.00 — The model correctly predicted every instance in the training set.
- **Macro Avg & Weighted Avg**: Both are 1.00 for all metrics (precision, recall, and F1-score).

Output Interpretation:

1. The model performs exceptionally well on the training data, achieving perfect