A2.52 Logistic Regression With Class Weight Balancing

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

    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import classification_report, confusion_matrix
    import seaborn as sns
    import matplotlib.pyplot as plt

    # 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 Logistic Regression model with class weight balancing
    reg_weighted = LogisticRegression(random_state=0, class_weight=class_weight)

    # 3. Train the Logistic Regression model with class weight balancing
    reg_weighted.fit(X_train, y_train)

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

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

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

This code provides an approach to implementing logistic regression with class weight balancing in order to tackle class imbalance issues in the dataset:

1. **Import Necessary Libraries**:
   - **`sklearn.linear_model`**: Contains the Logistic Regression model.
   - **`sklearn.metrics`**: Provides functions to compute metrics like the classification report and confusion matrix.
   - **`seaborn` and `matplotlib`**: Visualization libraries for plotting.

2. **Class Weight Balancing**:
   - This step calculates weights for classes to help the model pay more attention to the class with fewer samples. The logic behind calculating the weight for a class is to take the total number of samples and divide it by twice the number of samples in that class. As a result, classes with fewer samples will get higher weights.

3. **Initialize Logistic Regression Model**:
   - `LogisticRegression(random_state=0, class_weight=class_weight)`: Initializes a logistic regression model with a fixed random state for reproducibility and the calculated class weights.

4. **Training the Model**:
   - `reg_weighted.fit(X_train, y_train)`: Trains the logistic regression model using the training data.

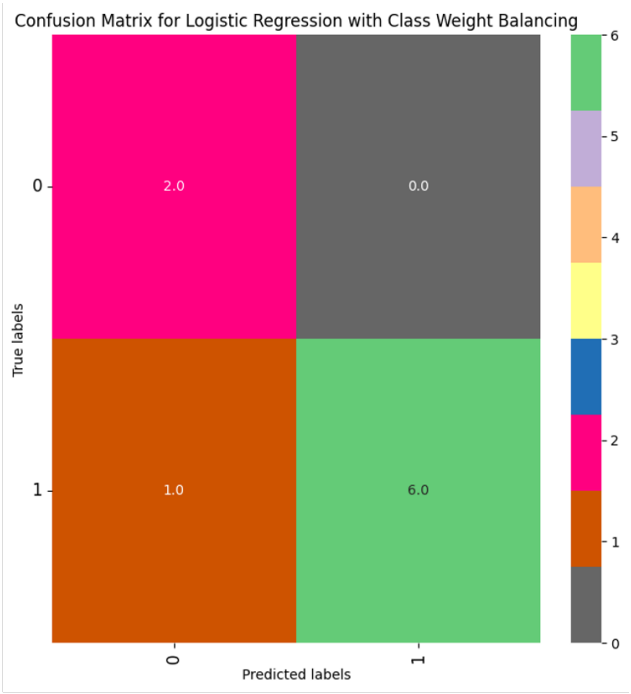
5. **Evaluate Model on Test Set**:
   - After training, the model's performance on the test set is evaluated and printed. The metrics include precision, recall, f1-score, and support for each class.

6. **Evaluate Model on Training Set**:
   - The performance of the model on the training data is also printed. This is useful to check for overfitting or underfitting. If a model has a significantly better performance on the training set compared to the test set, it might be overfitting.

7. **Plotting the Confusion Matrix**:
   - A confusion matrix visually represents the performance of a classification model.
   - **`confusion_matrix(y_test, reg_weighted.predict(X_test))`**: Computes the confusion matrix using true labels and predicted labels.
   - **`plt.subplots(figsize=(8, 8))`**: Sets the figure size for the confusion matrix plot.
   - **`sns.heatmap()`**: Uses Seaborn to create a heatmap representation of the confusion matrix.
   - The remaining commands in this section are for aesthetic purposes, setting the labels, rotation angles, font sizes, and titles to make the plot readable and understandable.

The objective of this code is to give a more balanced approach to classifying instances when there's an imbalance in the number of samples between classes.

```
Logistic Regression with Class Weight Balancing - Classification Report (Test Set):
              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

Logistic Regression 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
```



The provided results are the classification reports for a logistic regression model with class weight balancing. Let's break down the results for a better understanding.

Test Set Classification Report:

1. **Class 0**:
   - **Precision**: Out of all the samples predicted as class 0, 67% were correctly classified (True Positives). This means that 33% were wrongly classified as class 0.
   - **Recall**: Out of all the actual samples of class 0, 100% were correctly predicted by the model.
   - **F1-Score**: This is the harmonic mean of precision and recall. It provides a balance between the two. For class 0, it's 0.80, indicating a decent performance for this class on the test set.
   - **Support**: There are 2 actual instances of class 0 in the test set.

2. **Class 1**:
   - **Precision**: 100% of the samples predicted as class 1 were correctly classified.
   - **Recall**: 86% of the actual class 1 samples were correctly predicted by the model. This means 14% were missed.
   - **F1-Score**: It's 0.92, indicating an excellent performance for this class on the test set.
   - **Support**: There are 7 actual instances of class 1 in the test set.

3. **Overall Metrics**:
   - **Accuracy**: 89% of all predictions on the test set were correct.
   - **Macro Average**: Averages the metrics without considering class support. Here, it's 0.83 for precision, 0.93 for recall, and 0.86 for the F1-score.
   - **Weighted Average**: Takes into account the support (number of true instances) for each class. It's 0.93 for precision, 0.89 for accuracy, and 0.90 for the F1-score.

Training Set Classification Report:

For the training set, every metric for both classes is at 100%, indicating a perfect performance.

1. **Class 0** has 4 samples, and **Class 1** has 16 samples.
2. The overall **accuracy** is 100%.
3. Both the **macro and weighted averages** are also perfect with scores of 1.00 across precision, recall, and F1-score.

Output Interpretation:
1. The model performs flawlessly on the training set, which is expected since models tend to perform well on the data they were trained on.
2. For the test set, the model also performs relatively well, especially for class 1