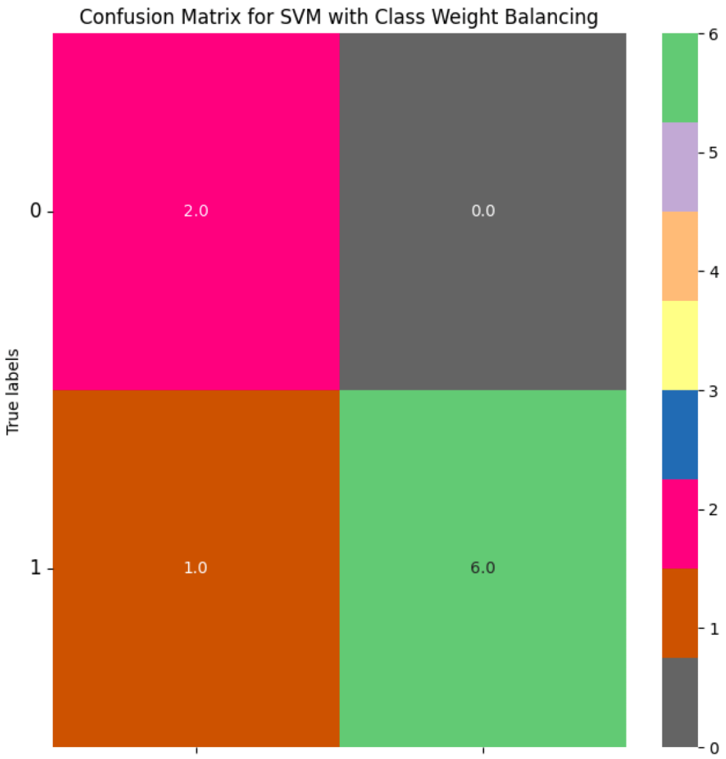
A2.32 SVM With Weight Balance

```
[ ] # Applyig Weight Balancing to address the class imbalance in the dataset
    from sklearn.svm import SVC
    from sklearn.metrics import classification_report, confusion_matrix

    class_weight = {0: len(y_train) / (2 * sum(y_train == 0)), 1: len(y_train) / (2 * sum(y_train == 1))}

    # 3. Create the SVM model with class weight balancing
    svm_model_weighted = SVC(kernel='linear', class_weight=class_weight)

    # 4. Train the SVM model with class weight balancing
    svm_model_weighted.fit(X_train, y_train)

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

    print("SVM with Class Weight Balancing - Classification Report (Train Set Results):")
    print(classification_report(y_train, svm_model_weighted.predict(X_train)))

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

```
# Applying Weight Balancing to address the class imbalance in the dataset
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix


class_weight = {0: len(y_train) / (2 * sum(y_train == 0)), 1: len(y_train) / (2 *
sum(y_train == 1))}

# Create the SVM model with class weight balancing
svm_model_weighted = SVC(kernel='linear', class_weight=class_weight)
# Train the SVM model with class weight balancing
svm_model_weighted.fit(X_train, y_train)
# Evaluate the SVM model on the test set
print("SVM with Class Weight Balancing - Classification Report (Test Set Results):")
print(classification_report(y_test, svm_model_weighted.predict(X_test)))

print("SVM with Class Weight Balancing - Classification Report (Train Set Results):")
print(classification_report(y_train, svm_model_weighted.predict(X_train)))

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

```
SVM with Class Weight Balancing - Classification Report (Test Set Results):
              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

SVM with Class Weight Balancing - Classification Report (Train Set Results):
              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 classification report is for a SVM with class weight balancing.

Metrics:

1. **Precision**: It quantifies the number of correct positive predictions made. It's the ratio of correctly predicted positive observations to the total predicted positives.

   \[\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \]

2. **Recall (or Sensitivity or True Positive Rate)**: It quantifies the number of correct positive predictions made out of all actual positives. It's the ratio of correctly predicted positive observations to all the observations in the actual class.

   \[\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \]

3. **F1-Score**: It's the harmonic mean of Precision and Recall and provides a better measure when there are uneven class distributions. It's a good metric when the false positives and false negatives have similar costs.

   \[\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \]

4. **Support**: It's the number of actual occurrences of the class in the specified dataset. For our dataset, there are 2 occurrences of the class labeled 0 and 7 occurrences of the class labeled 1.

Rows in the Report:

- **0**: Metrics for the class labeled as '0'.
   - **Precision**: 0.67
   - **Recall**: 1.00
   - **F1-Score**: 0.80
   - **Support**: 2

- **1**: Metrics for the class labeled as '1'.
   - **Precision**: 1.00
   - **Recall**: 0.86
   - **F1-Score**: 0.92
   - **Support**: 7

- **Accuracy**: It's the ratio of correctly predicted observation to the total observations. For our report, it's 0.89, meaning the model was accurate for 89% of all predictions on the provided dataset.

- **Macro Avg**: This computes the metric independently for each class and then takes the average, hence treating all classes equally.