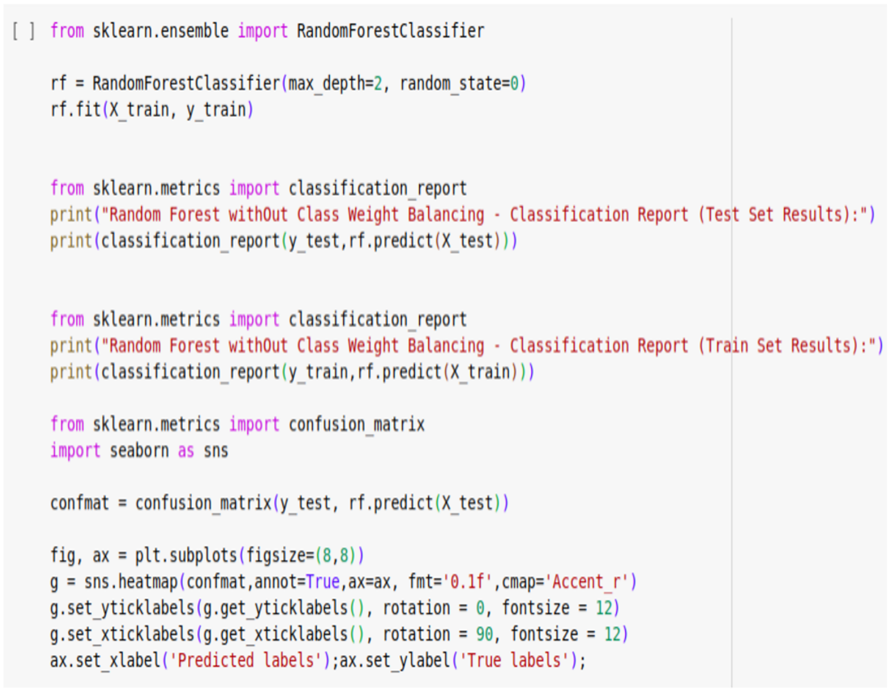
A2.41 Random Forest Random Forest Without Class Weight Balancing



```
[ ] from sklearn.ensemble import RandomForestClassifier

    rf = RandomForestClassifier(max_depth=2, random_state=0)
    rf.fit(X_train, y_train)


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


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

    from sklearn.metrics import confusion_matrix
    import seaborn as sns

    confmat = confusion_matrix(y_test, rf.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');
```

This code is implementing a random forest classifier, training it, and then evaluating its performance on both the training and test data. Let's break it down step-by-step:

1. **Random Forest Classifier Initialization**:
```python
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(max_depth=2, random_state=0)
```
Here, a random forest classifier from the Scikit-learn library is imported and initialized. The classifier is set to have a maximum depth of 2 for each decision tree in the forest. The `random_state` parameter ensures that the results remain consistent across different runs.

2. **Training the Classifier**:
```python
rf.fit(X_train, y_train)
```
The random forest classifier `rf` is trained using the training data `X_train` and the corresponding labels `y_train`.

3. **Evaluating the Classifier on Test Data**:
```python
from sklearn.metrics import classification_report
print(classification_report(y_test, rf.predict(X_test)))
```
The model's predictions on the test dataset are compared to the actual labels (`y_test`). The `classification_report` then prints metrics such as precision, recall, and F1-score for each class.
4. **Evaluating the Classifier on Training Data**:
```python
print(classification_report(y_train, rf.predict(X_train)))
```
This is similar to the previous step, but here the model's predictions on the training dataset are evaluated. This helps in understanding if the model is overfitting (i.e., performing exceptionally well on the training data but not as well on unseen/test data).

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

confmat = confusion_matrix(y_test, rf.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)
```

A2.41 Random Forest Random Forest Without Class Weight Balancing - 30/05/2024,

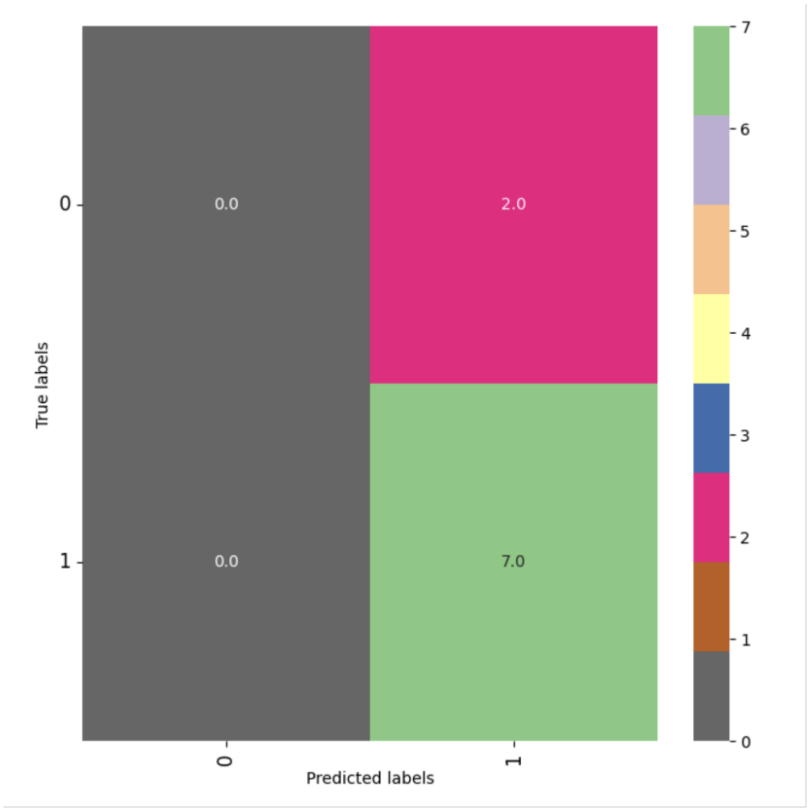
```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.78      1.00      0.88         7

    accuracy                           0.78         9
   macro avg       0.39      0.50      0.44         9
weighted avg       0.60      0.78      0.68         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
```



A2.41 Random Forest Random Forest Without Class Weight Balancing - 30/05/2024,

This output provides a classification report for a model, specifically for a Random Forest classifier, across two different sets (likely training and testing sets). The report contains metrics such as precision, recall, f1-score, and support for each class, as well as some aggregated metrics.:

First Output (test set):

1. For Class `0`:
   - **Precision**: 0.00 - Out of all the samples predicted as class `0`, none were actually class `0`.
   - **Recall**: 0.00 - Out of all the actual class `0` samples, none were correctly predicted.
   - **F1-Score**: 0.00 - Harmonic mean of precision and recall, indicating a poor performance for class `0`.
   - **Support**: 2 - The actual number of occurrences of class `0` in the test set.

2. For Class `1`:
   - **Precision**: 0.78 - Out of all the samples predicted as class `1`, 78% were actually class `1`.
   - **Recall**: 1.00 - All the actual class `1` samples were correctly predicted.
   - **F1-Score**: 0.88 - Indicates a relatively good performance for class `1`.
   - **Support**: 7 - The actual number of occurrences of class `1` in the test set.

3. **Accuracy**: 0.78 - Overall, 78% of all predictions were correct out of the 9 samples.

4. **Macro Avg**: Averages the unweighted mean per label.
   - **Precision**: 0.39
   - **Recall**: 0.50
   - **F1-Score**: 0.44
   - **Support**: 9

5. **Weighted Avg**: Averages the support-weighted mean per label.
   - **Precision**: 0.60
   - **Recall**: 0.78
   - **F1-Score**: 0.68
   - **Support**: 9

Second Output (train set):

1. For Class `0` and Class `1`, all metrics (Precision, Recall, and F1-Score) are 1.00, indicating perfect predictions.

2. **Accuracy**: 1.00 - All predictions were correct for the 20 samples.

3. **Macro Avg** and **Weighted Avg** are also perfect (1.00 for all metrics), further underscoring the model's exemplary performance on this set.

A2.41 Random Forest Random Forest Without Class Weight Balancing - 30/05/2024,