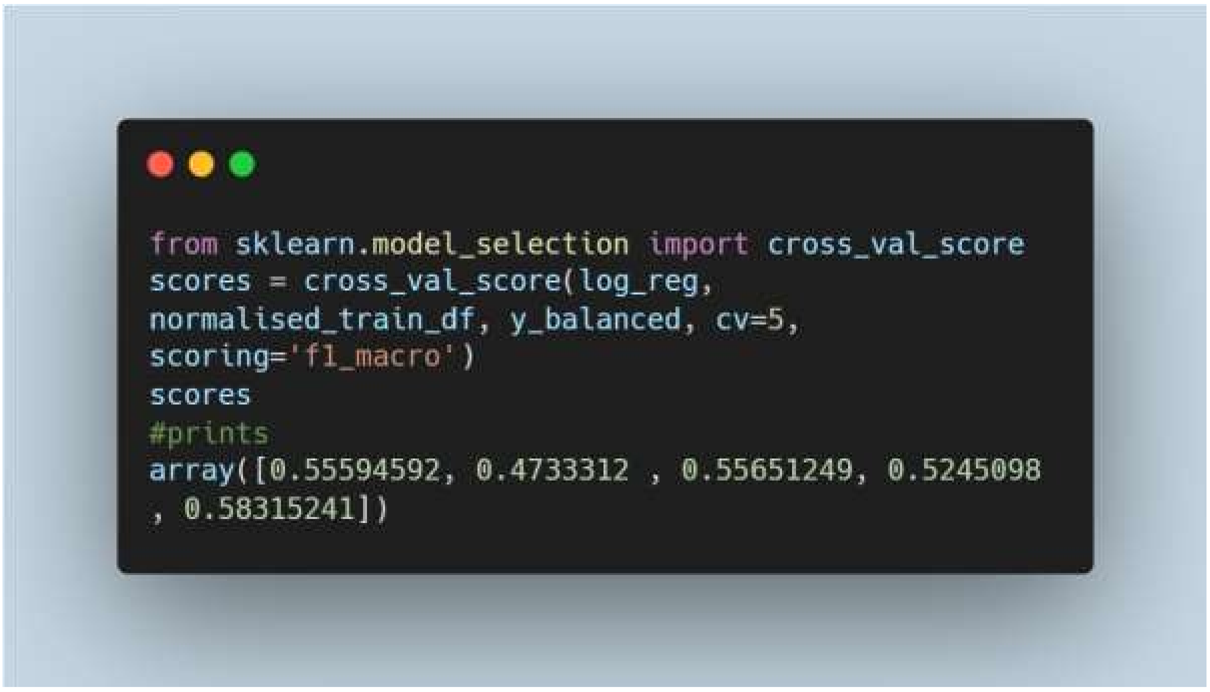
Measuring Classification Performance

Cross-validation and accuracy

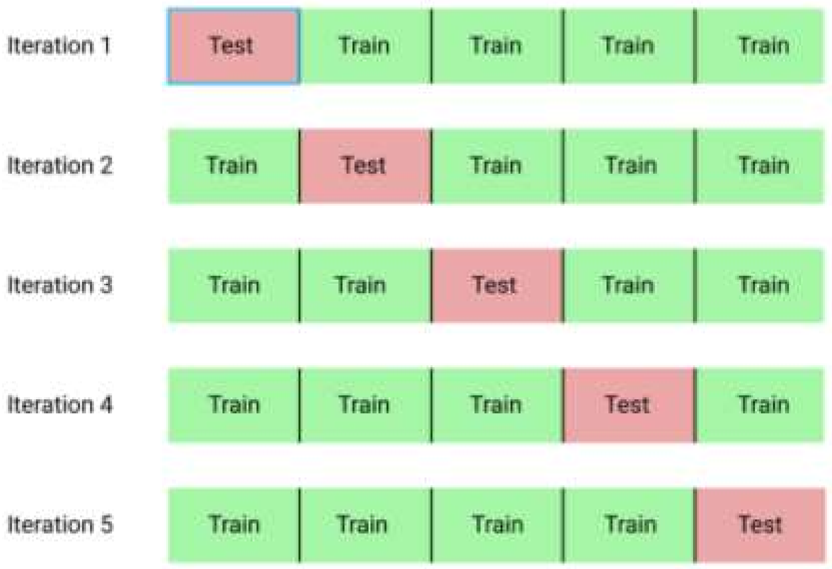
From the previous module, we now understand why data scientists and machine learning engineers avoid having models that overfit or underfit. Cross Validation (CV) is a well known and trusted method applied to avoid overfitting and enable generalization. Although there are different techniques used in performing cross validation, the fundamental concept involves partitioning the dataset into a number of subsets, holding out a set for evaluation then training the model on the other sets. This gives a more reliable estimate of how the model performs across different training sets because it provides an average score across different training samples used. The only drawback with cross validation is that it takes more time and computational resources however, the gain obtained in having a better model is very well worth this cost. K-Fold cross validation, Stratified K-Fold cross validation and Leave One Out Cross Validation (LOOCV) are some cross validation techniques.

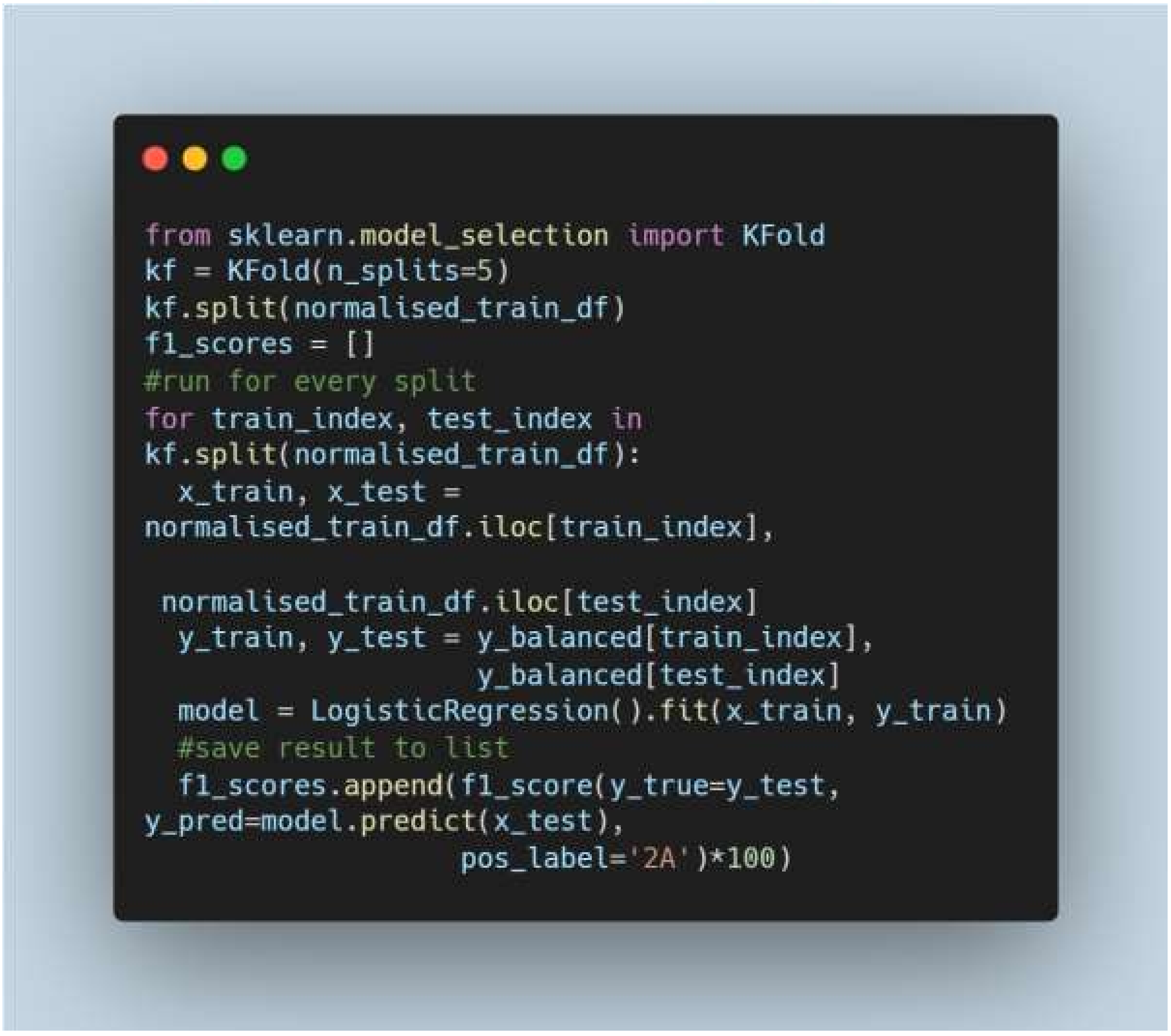


```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(log_reg,
normalised_train_df, y_balanced, cv=5,
scoring='f1_macro')
scores
#prints
array([0.55594592, 0.4733312 , 0.55651249, 0.5245098
, 0.58315241])
```

K-Fold Cross Validation

This technique is called K-Fold because the data is split into K equal groups. If k=5,a 5-fold cross validation can be performed such that the data is split into k1, k2, k3, k4 and k5. The model is trained on k2 - k5 and evaluated on k1 then repeated k times until every group is used to train and test the model.



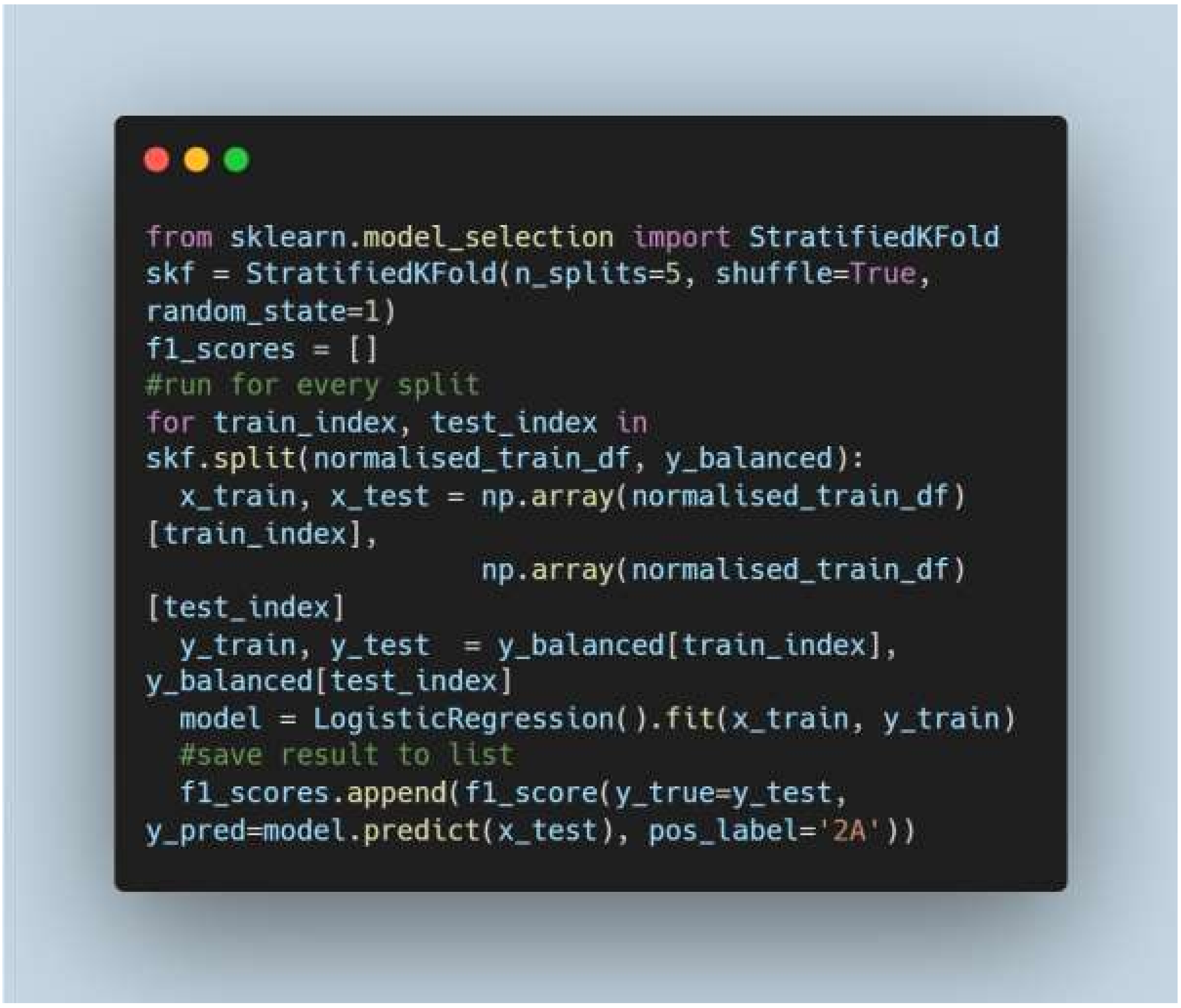


```
from sklearn.model_selection import KFold
kf = KFold(n_splits=5)
kf.split(normalised_train_df)
f1_scores = []
#run for every split
for train_index, test_index in
kf.split(normalised_train_df):
  x_train, x_test =
normalised_train_df.iloc[train_index],

 normalised_train_df.iloc[test_index]
  y_train, y_test = y_balanced[train_index],
                    y_balanced[test_index]
  model = LogisticRegression().fit(x_train, y_train)
  #save result to list
  f1_scores.append(f1_score(y_true=y_test,
y_pred=model.predict(x_test),
                   pos_label='2A')*100)
```

Stratified K-Fold Cross Validation

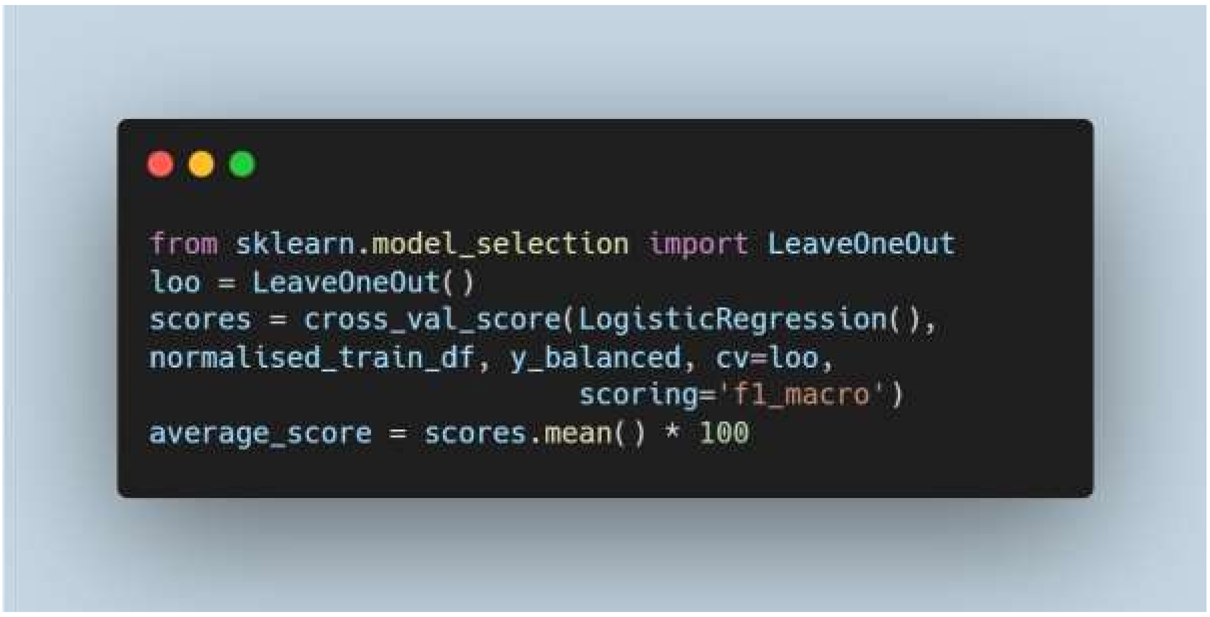
Although similar to the technique described above, Stratified K-Fold cross validation ensures that in every fold, there is an equal proportion of each target class to obtain a good representation of the data and avoid imbalance and biased results. For example, if there are two target classes t1 and t2 with equal distribution in the data, it is best to ensure that the folds also have the same distribution.



```
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5, shuffle=True,
random_state=1)
f1_scores = []
#run for every split
for train_index, test_index in
skf.split(normalised_train_df, y_balanced):
  x_train, x_test = np.array(normalised_train_df)
[train_index],
                    np.array(normalised_train_df)
[test_index]
  y_train, y_test  = y_balanced[train_index],
y_balanced[test_index]
  model = LogisticRegression().fit(x_train, y_train)
  #save result to list
  f1_scores.append(f1_score(y_true=y_test,
y_pred=model.predict(x_test), pos_label='2A'))
```

Leave One Out Cross Validation (LOOCV)

In this method, one instance is left out and used as the test set while the model is trained on N-1data points where N is the number of data points. This means that the number of instances and folds are equal.



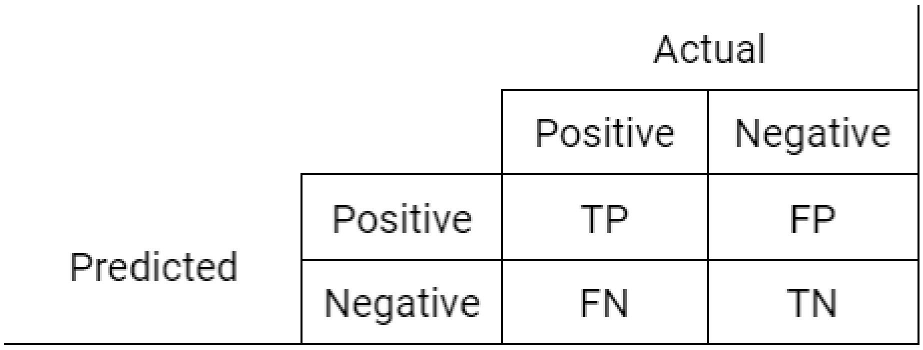
```
from sklearn.model_selection import LeaveOneOut
loo = LeaveOneOut()
scores = cross_val_score(LogisticRegression(),
normalised_train_df, y_balanced, cv=loo,
                            scoring='f1_macro')
average_score = scores.mean() * 100
```

Confusion Matrix, Precision-Recall, ROC curve and the F1-score

Accuracy, precision, recall, F1-score and many others are evaluation metrics used in measuring the performance of classification models. In this section, we discuss these metrics.

Confusion Matrix

It is an N x N matrix that gives a summary of the correct and incorrect predicted classification results for the *N*target classes. The values in the diagonal of the matrix represent the number of correctly predicted classes while every other cell in the matrix indicates the misclassified classes. This means that the more predicted values that fall in the diagonal, the better the model. True positive, false positive, true negative and false negative are terms used when interpreting a confusion matrix.



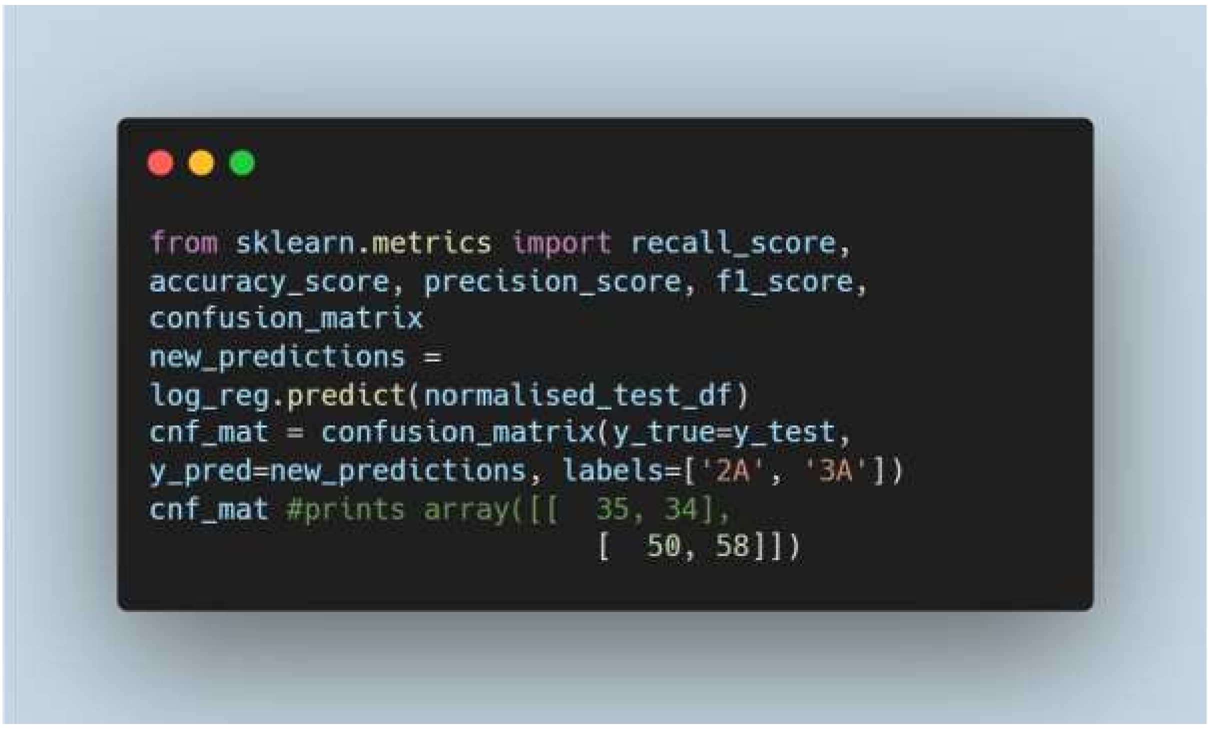
		Actual	
		Positive	Negative
Predicted	Positive	TP	FP
	Negative	FN	TN

- True Positive (TP): This is a correct classification where the predicted value is the same as the actual value. Using the table above, this means that actual value was positive and the predicted value was also positive.

- True Negative (TN): The predicted value also matches the actual value. In this case, it is for the negative class. The actual value is negative and the predicted value is negative.

- False Positive (FP): Also called a Type I error, this is a misclassification such that the model predicted a positive class while the actual class is negative. Telling a man that he is pregnant is definitely a false positive.

- False Negative (FN): Also another misclassification where the predicted value is negative and the actual value is positive. Another example will be telling a pregnant woman that she is not pregnant. FN is known as a Type II error.

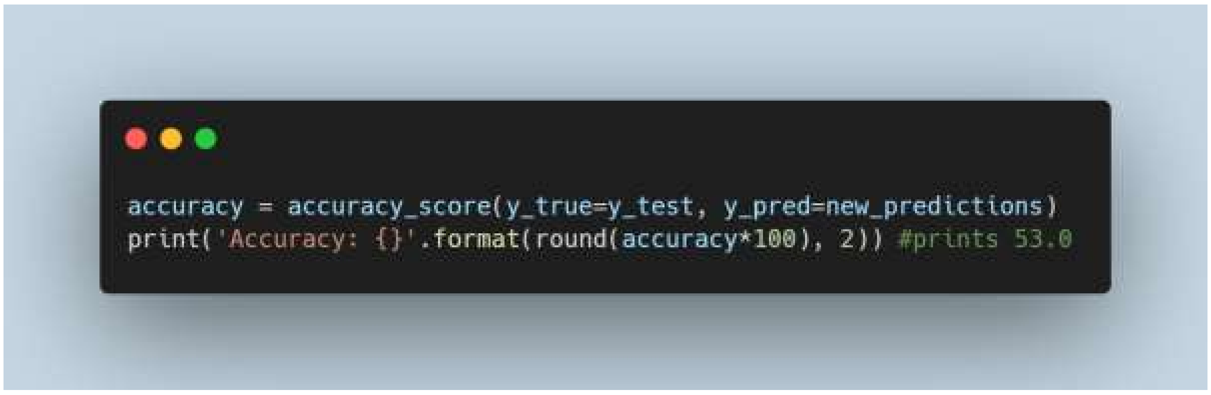


```
from sklearn.metrics import recall_score,
accuracy_score, precision_score, f1_score,
confusion_matrix
new_predictions =
log_reg.predict(normalised_test_df)
cnf_mat = confusion_matrix(y_true=y_test,
y_pred=new_predictions, labels=['2A', '3A'])
cnf_mat #prints array([[  35, 34],
                       [  50, 58]])
```

Accuracy

This is the ratio of the number of correctly predicted instances to the total number of instances. It is a commonly used metric suitable when the target classes are not imbalanced. A high accuracy does not necessarily mean that the model has high predicting power. Hence, depending on the task, it is important to not use only the accuracy metric because it does not provide enough information about the model.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

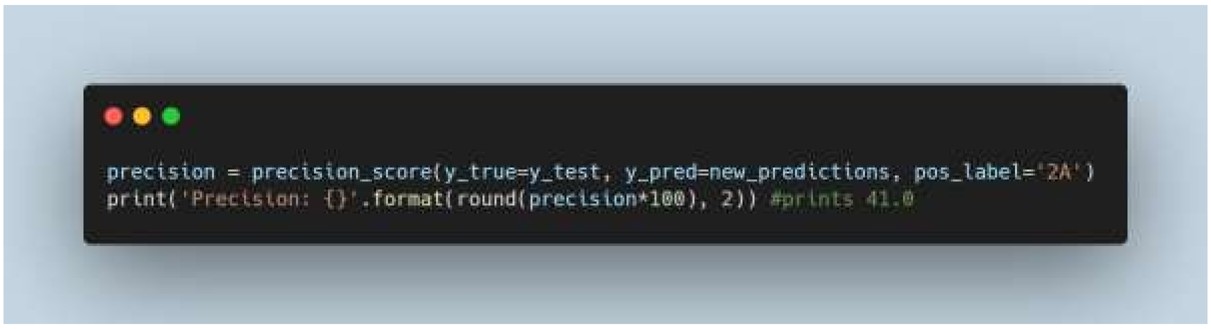


```
accuracy = accuracy_score(y_true=y_test, y_pred=new_predictions)
print('Accuracy: {}'.format(round(accuracy*100), 2)) #prints 53.0
```

Precision

The ratio of correctly predicted instances of a class to the total number of items predicted by the model to be in that class is referred to as precision (known as Positive Predicted Value - PPV). This translates to the total percentage of the results obtained that are relevant. For the positive class, it is the ratio of true positives to the sum of true positives and false positives

$$Precision = \frac{TP}{TP + FP}$$



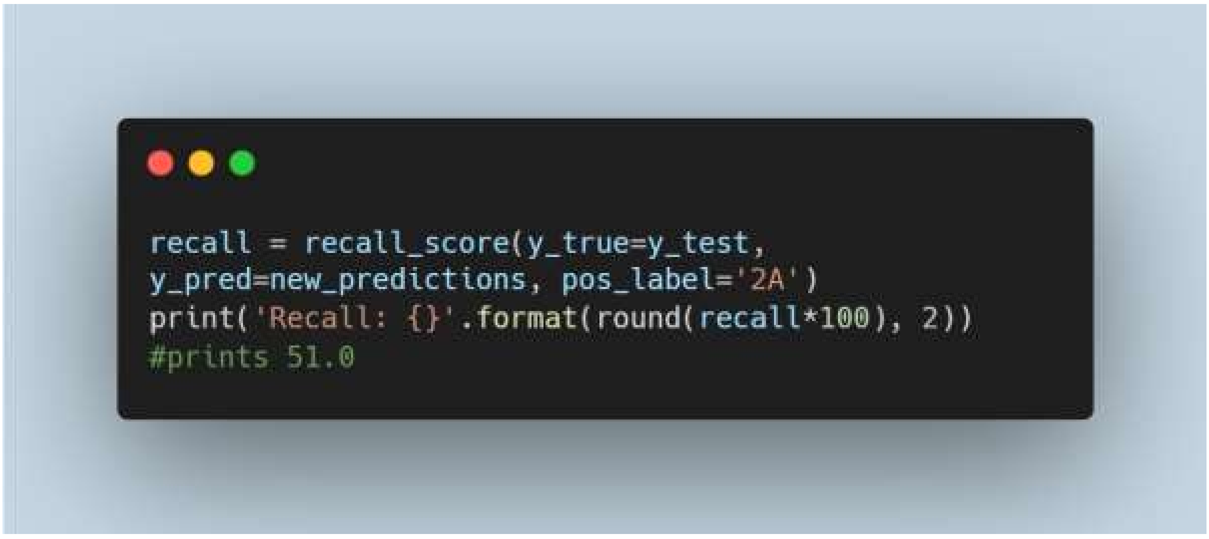
```
precision = precision_score(y_true=y_test, y_pred=new_predictions, pos_label='2A')
print('Precision: {}'.format(round(precision*100), 2)) #prints 41.0
```

Recall

Known as the sensitivity of the model, recall gives a percentage of total relevant results correctly predicted by the model. It is the ratio of the true positives to the actual number of positives (true positives and false negatives).

$$Recall = \frac{TP}{TP + FN}$$

Like in the previous module where we discussed the bias-variance trade-off, there is also a trade-off between precision and recall. It is impossible to maximise both metrics simultaneously because an increase in recall decreases precision. Identify which metric is important based on your task and optimise.

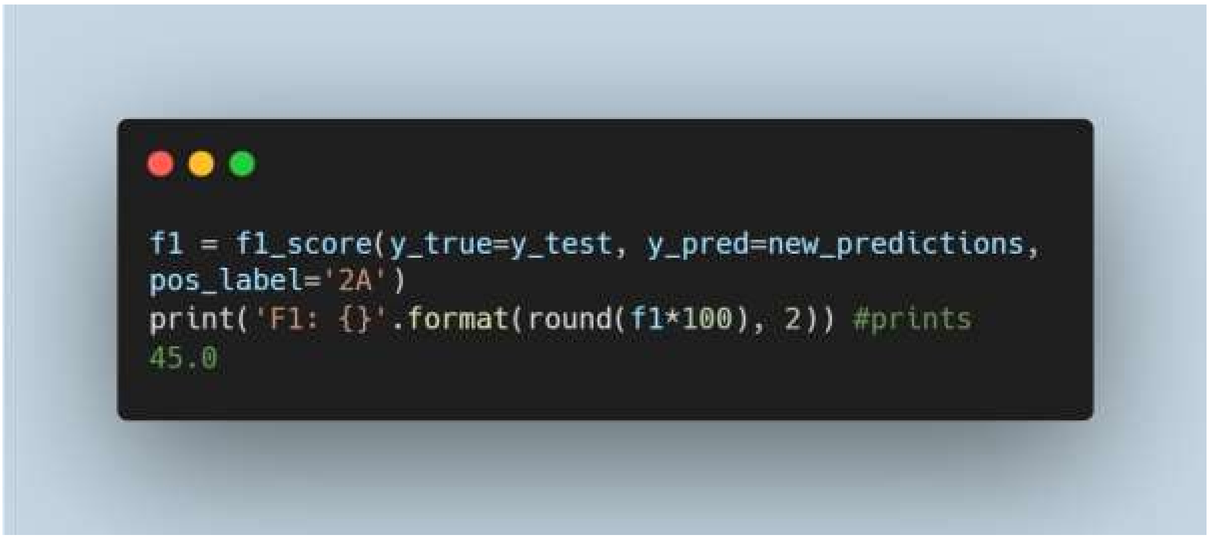


```
recall = recall_score(y_true=y_test,
y_pred=new_predictions, pos_label='2A')
print('Recall: {}'.format(round(recall*100), 2))
#prints 51.0
```

F1-Score

This metric is the harmonic mean of precision and recall that aims to have an optimal balance of both. The F1-Score is quite easy to use and can be focused on to maximize as opposed to maximizing precision and recall.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$



```
f1 = f1_score(y_true=y_test, y_pred=new_predictions,
pos_label='2A')
print('F1: {}'.format(round(f1*100), 2)) #prints
45.0
```

ROC Curve

The Receiver Operating Characteristics (ROC) curve is a probability curve that measures the performance of a classification model at different set thresholds. Recall also known as the True Positive Rate (TPR) is plotted on the y-axis against the False Positive Rate (FPR) on the x-axis.

The code examples above are not the optimal results that can be obtained with the model. Hyperparameter tuning can be performed to improve the model.

https://colab.research.google.com/notebooks/mlcc/logistic_regression.ipynb?hl=en