Let's see an example of this using the k-nearest neighbors (kNN) classification algorithm. When initializing **KFold**, it is standard practice to shuffle the data before splitting.

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier

# load dataset
data = datasets.load_iris()

# separate features and target
X = data.data
y = data.target

# initialize KFold - with shuffle = True, shuffle the data before splitting
kfold = KFold(n_splits=3, shuffle=True)

# create the model
knn_clf = KNeighborsClassifier(n_neighbors=3)

# fit the model using cross validation
cv_result = cross_val_score(knn_clf, X, y, cv=kfold)

# evaluate the model performance using accuracy metric
print("Accuracy: %.3f%% (%.3f%%)" % (cv_result.mean()*100.0, cv_result.
std()*100.0))
'Output':
Accuracy: 93.333% (2.494%)
```

Leave-One-Out Cross-Validation (LOOCV)

In LOOCV just one example is assigned to the test set, and the model is trained on the remainder of the dataset. This process is repeated for all the examples in the dataset. This process is repeated until all the examples in the dataset have been used for evaluating the model.

```
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
```

```
# load dataset
data = datasets.load_iris()

# separate features and target
X = data.data
y = data.target

# initialize LOOCV
loocv = LeaveOneOut()

# create the model
knn_clf = KNeighborsClassifier(n_neighbors=3)

# fit the model using cross validation
cv_result = cross_val_score(knn_clf, X, y, cv=loocv)

# evaluate the model performance using accuracy metric
print("Accuracy: %.3f%% (%.3f%%)" % (cv_result.mean()*100.0, cv_result.
std()*100.0))
'Output':
Accuracy: 96.000% (19.596%)
```

Model Evaluation

This chapter has already used a couple of evaluation metrics for assessing the quality of the fitted models. In this section, we survey a couple of other metrics for regression and classification use cases and how to implement them using Scikit-learn. For each metric, we show how to use them as stand-alone implementations, as well as together with cross-validation using the **cross_val_score** method.

What we'll cover here includes

Regression evaluation metrics

- Mean squared error (MSE): The average sum of squared difference between the predicted label, $\hat{y}$, and the true label, y. A score of 0 indicates a perfect prediction without errors.
- Mean absolute error (MAE): The average absolute difference between the predicted label, $\hat{y}$, and the true label, y. A score of 0 indicates a perfect prediction without errors.