A/B testing, without their knowledge a selected portion of users will be provided with a website or service using algorithm A, while the rest of the users will be provided with algorithm B. For both groups, relevant success metrics will be recorded for a set period of time. Then, the metrics of algorithm A and algorithm B will be compared, and a selection between the two approaches will be made according to these metrics. Using A/B testing enables us to evaluate the algorithms "in the wild," which might help us to discover unexpected consequences when users are interacting with our model. Often A is a new model, while B is the established system. There are more elaborate mechanisms for online testing that go beyond A/B testing, such as *bandit algorithms*. A great introduction to this subject can be found in the book *Bandit Algorithms for Website Optimization* by John Myles White (O'Reilly).

Building Your Own Estimator

This book has covered a variety of tools and algorithms implemented in `scikit-learn` that can be used on a wide range of tasks. However, often there will be some particular processing you need to do for your data that is not implemented in `scikit-learn`. It may be enough to just preprocess your data before passing it to your `scikit-learn` model or pipeline. However, if your preprocessing is data dependent, and you want to apply a grid search or cross-validation, things become trickier.

In Chapter 6 we discussed the importance of putting all data-dependent processing inside the cross-validation loop. So how can you use your own processing together with the `scikit-learn` tools? There is a simple solution: build your own estimator! Implementing an estimator that is compatible with the `scikit-learn` interface, so that it can be used with `Pipeline`, `GridSearchCV`, and `cross_val_score`, is quite easy. You can find detailed instructions in the `scikit-learn` documentation, but here is the gist. The simplest way to implement a transformer class is by inheriting from `BaseEstimator` and `TransformerMixin`, and then implementing the `__init__`, `fit`, and `predict` functions like this:

In[1]:

```
from sklearn.base import BaseEstimator, TransformerMixin

class MyTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, first_parameter=1, second_parameter=2):
        # All parameters must be specified in the __init__ function
        self.first_parameter = 1
        self.second_parameter = 2

    def fit(self, X, y=None):
        # fit should only take X and y as parameters
        # Even if your model is unsupervised, you need to accept a y argument!

        # Model fitting code goes here
        print("fitting the model right here")
        # fit returns self
        return self

    def transform(self, X):
        # transform takes as parameter only X

        # Apply some transformation to X
        X_transformed = X + 1
        return X_transformed
```

Implementing a classifier or regressor works similarly, only instead of `TransformerMixin` you need to inherit from `ClassifierMixin` or `RegressorMixin`. Also, instead of implementing `transform`, you would implement `predict`.

As you can see from the example given here, implementing your own estimator requires very little code, and most `scikit-learn` users build up a collection of custom models over time.

Where to Go from Here

This book provides an introduction to machine learning and will make you an effective practitioner. However, if you want to further your machine learning skills, here are some suggestions of books and more specialized resources to investigate to dive deeper.

Theory

In this book, we tried to provide an intuition of how the most common machine learning algorithms work, without requiring a strong foundation in mathematics or computer science. However, many of the models we discussed use principles from probability theory, linear algebra, and optimization. While it is not necessary to understand all the details of how these algorithms are implemented, we think that