Then the following questions open up:

- How do I measure if my fraud prediction is actually working?
- Do I have the right data to evaluate an algorithm?
- If I am successful, what will be the business impact of my solution?

As we discussed in Chapter 5, it is best if you can measure the performance of your algorithm directly using a business metric, like increased profit or decreased losses. This is often hard to do, though. A question that can be easier to answer is "What if I built the perfect model?" If perfectly detecting any fraud will save your company $100 a month, these possible savings will probably not be enough to warrant the effort of you even starting to develop an algorithm. On the other hand, if the model might save your company tens of thousands of dollars every month, the problem might be worth exploring.

Say you've defined the problem to solve, you know a solution might have a significant impact for your project, and you've ensured that you have the right information to evaluate success. The next steps are usually acquiring the data and building a working prototype. In this book we have talked about many models you can employ, and how to properly evaluate and tune these models. While trying out models, though, keep in mind that this is only a small part of a larger data science workflow, and model building is often part of a feedback circle of collecting new data, cleaning data, building models, and analyzing the models. Analyzing the mistakes a model makes can often be informative about what is missing in the data, what additional data could be collected, or how the task could be reformulated to make machine learning more effective. Collecting more or different data or changing the task formulation slightly might provide a much higher payoff than running endless grid searches to tune parameters.

Humans in the Loop

You should also consider if and how you should have humans in the loop. Some processes (like pedestrian detection in a self-driving car) need to make immediate decisions. Others might not need immediate responses, and so it can be possible to have humans confirm uncertain decisions. Medical applications, for example, might need very high levels of precision that possibly cannot be achieved by a machine learning algorithm alone. But if an algorithm can make 90 percent, 50 percent, or maybe even just 10 percent of decisions automatically, that might already increase response time or reduce cost. Many applications are dominated by "simple cases," for which an algorithm can make a decision, with relatively few "complicated cases," which can be rerouted to a human.

From Prototype to Production

The tools we've discussed in this book are great for many machine learning applications, and allow very quick analysis and prototyping. Python and `scikit-learn` are also used in production systems in many organizations—even very large ones like international banks and global social media companies. However, many companies have complex infrastructure, and it is not always easy to include Python in these systems. That is not necessarily a problem. In many companies, the data analytics teams work with languages like Python and R that allow the quick testing of ideas, while production teams work with languages like Go, Scala, C++, and Java to build robust, scalable systems. Data analysis has different requirements from building live services, and so using different languages for these tasks makes sense. A relatively common solution is to reimplement the solution that was found by the analytics team inside the larger framework, using a high-performance language. This can be easier than embedding a whole library or programming language and converting from and to the different data formats.

Regardless of whether you can use `scikit-learn` in a production system or not, it is important to keep in mind that production systems have different requirements from one-off analysis scripts. If an algorithm is deployed into a larger system, software engineering aspects like reliability, predictability, runtime, and memory requirements gain relevance. Simplicity is key in providing machine learning systems that perform well in these areas. Critically inspect each part of your data processing and prediction pipeline and ask yourself how much complexity each step creates, how robust each component is to changes in the data or compute infrastructure, and if the benefit of each component warrants the complexity. If you are building involved machine learning systems, we highly recommend reading the paper "Machine Learning: The High Interest Credit Card of Technical Debt", published by researchers in Google's machine learning team. The paper highlights the trade-off in creating and maintaining machine learning software in production at a large scale. While the issue of technical debt *is* particularly pressing in large-scale and long-term projects, the lessons learned can help us build better software even for short-lived and smaller systems.

Testing Production Systems

In this book, we covered how to evaluate algorithmic predictions based on a test set that we collected beforehand. This is known as *offline evaluation*. If your machine learning system is user-facing, this is only the first step in evaluating an algorithm, though. The next step is usually *online testing* or *live testing*, where the consequences of employing the algorithm in the overall system are evaluated. Changing the recommendations or search results users are shown by a website can drastically change their behavior and lead to unexpected consequences. To protect against these surprises, most user-facing services employ *A/B testing*, a form of blind user study. In