CHAPTER 3

Data Manipulation with Pandas

In the previous chapter, we dove into detail on NumPy and its `ndarray` object, which provides efficient storage and manipulation of dense typed arrays in Python. Here we'll build on this knowledge by looking in detail at the data structures provided by the Pandas library. Pandas is a newer package built on top of NumPy, and provides an efficient implementation of a `DataFrame`. `DataFrame`s are essentially multidimensional arrays with attached row and column labels, and often with heterogeneous types and/or missing data. As well as offering a convenient storage interface for labeled data, Pandas implements a number of powerful data operations familiar to users of both database frameworks and spreadsheet programs.

As we saw, NumPy's `ndarray` data structure provides essential features for the type of clean, well-organized data typically seen in numerical computing tasks. While it serves this purpose very well, its limitations become clear when we need more flexibility (attaching labels to data, working with missing data, etc.) and when attempting operations that do not map well to element-wise broadcasting (groupings, pivots, etc.), each of which is an important piece of analyzing the less structured data available in many forms in the world around us. Pandas, and in particular its `Series` and `DataFrame` objects, builds on the NumPy array structure and provides efficient access to these sorts of "data munging" tasks that occupy much of a data scientist's time.

In this chapter, we will focus on the mechanics of using `Series`, `DataFrame`, and related structures effectively. We will use examples drawn from real datasets where appropriate, but these examples are not necessarily the focus.

Installing and Using Pandas

Installing Pandas on your system requires NumPy to be installed, and if you're building the library from source, requires the appropriate tools to compile the C and

Cython sources on which Pandas is built. Details on this installation can be found in the Pandas documentation. If you followed the advice outlined in the preface and used the Anaconda stack, you already have Pandas installed.

Once Pandas is installed, you can import it and check the version:

```
In[1]: import pandas
       pandas.__version__

Out[1]: '0.18.1'
```

Just as we generally import NumPy under the alias `np`, we will import Pandas under the alias `pd`:

```
In[2]: import pandas as pd
```

This import convention will be used throughout the remainder of this book.

Reminder About Built-In Documentation

As you read through this chapter, don't forget that IPython gives you the ability to quickly explore the contents of a package (by using the tab-completion feature) as well as the documentation of various functions (using the `?` character). (Refer back to "Help and Documentation in IPython" on page 3 if you need a refresher on this.)

For example, to display all the contents of the `pandas` namespace, you can type this:

```
In [3]: pd.<TAB>
```

And to display the built-in Pandas documentation, you can use this:

```
In [4]: pd?
```

More detailed documentation, along with tutorials and other resources, can be found at *http://pandas.pydata.org/*.

Introducing Pandas Objects

At the very basic level, Pandas objects can be thought of as enhanced versions of NumPy structured arrays in which the rows and columns are identified with labels rather than simple integer indices. As we will see during the course of this chapter, Pandas provides a host of useful tools, methods, and functionality on top of the basic data structures, but nearly everything that follows will require an understanding of what these structures are. Thus, before we go any further, let's introduce these three fundamental Pandas data structures: the `Series`, `DataFrame`, and `Index`.

We will start our code sessions with the standard NumPy and Pandas imports:

```
In[1]: import numpy as np
       import pandas as pd
```