

Data Manipulation with Pandas

In [Part 2 \(02.00-Introduction-to-NumPy.ipynb\)](#), we dove into detail on NumPy and its `ndarray` object, which enables efficient storage and manipulation of dense typed arrays in Python. Here we'll build on this knowledge by looking in depth at the data structures provided by the Pandas library. Pandas is a newer package built on top of NumPy that provides an efficient implementation of a `DataFrame`. `DataFrame`s are essentially multidimensional arrays with attached row and column labels, 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've seen, 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 (e.g., attaching labels to data, working with missing data, etc.) and when attempting operations that do not map well to element-wise broadcasting (e.g., 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 part of the book, 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

Installation of Pandas on your system requires NumPy to be installed, and if you're building the library from source, you will need the appropriate tools to compile the C and Cython sources on which Pandas is built. Details on the installation process can be found in the [Pandas documentation \(http://pandas.pydata.org/\)](#). If you followed the advice outlined in the [Preface \(00.00-Preface.ipynb\)](#) and used the Anaconda stack, you already have Pandas installed.

Once Pandas is installed, you can import it and check the version; here is the version used by this book:

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

```
Out[1]: '1.3.5'
```

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 part of the book, 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 \(01.01-Help-And-Documentation.ipynb\)](#) if you need a refresher on this.

For example, to display all the contents of the Pandas namespace, you can type:

```
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/> (<http://pandas.pydata.org/>).