Data Wrangling with Pandas, NumPy, and IPython

Python for Data Analysis

O'REILLY®

Wes McKinney

Python for Data Analysis

Wes McKinney

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Special Upgrade Offer

If you purchased this ebook directly from oreilly.com, you have the following benefits:

- DRM-free ebooks—use your ebooks across devices without restrictions or limitations
- Multiple formats—use on your laptop, tablet, or phone
- Lifetime access, with free updates
- Dropbox syncing—your files, anywhere

If you purchased this ebook from another retailer, you can upgrade your ebook to take advantage of a
these benefits for just $4.99. Click here to access your ebook upgrade.

Please note that upgrade offers are not available from sample content.

Preface

The scientific Python ecosystem of open source libraries has grown substantially over the last 10 years. By late 2011, I had long felt that the lack of centralized learning resources for data analysis an statistical applications was a stumbling block for new Python programmers engaged in such work. Key projects for data analysis (especially NumPy, IPython, matplotlib, and pandas) had also matured enough that a book written about them would likely not go out-of-date very quickly. Thus, I mustered the nerve to embark on this writing project. This is the book that I wish existed when I started using Python for data analysis in 2007. I hope you find it useful and are able to apply these tools productively in your work.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`

Shows commands or other text that should be typed literally by the user.

`Constant width italic`

Shows text that should be replaced with user-supplied values or by values determined by context.

> **TIP**
>
> This icon signifies a tip, suggestion, or general note.

> **CAUTION**
>
> This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducin a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: “*Python for Data Analysis* by William Wesley McKinney (O’Reilly). Copyright 2012 William McKinney, 978-1-449-31979-3.”

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

Safari® Books Online

NOTE

Safari Books Online (www.safaribooksonline.com) is an on-demand digital library that delivers expert content in both book and video form from the world’s leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos and prepublication manuscripts in one fully searchable database from publishers like O’Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O’Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-998-9938 (in the United States or Canada)

707-829-0515 (international or local)

707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at http://oreil.ly/python_for_data_analysis.

To comment or ask technical questions about this book, send email to bookquestions@oreilly.com.

For more information about our books, courses, conferences, and news, see our website at http://www.oreilly.com.

Find us on Facebook: http://facebook.com/oreilly

Follow us on Twitter: http://twitter.com/oreillymedia

Watch us on YouTube: http://www.youtube.com/oreillymedia

Chapter 1. Preliminaries

What Is This Book About?

This book is concerned with the nuts and bolts of manipulating, processing, cleaning, and crunching data in Python. It is also a practical, modern introduction to scientific computing in Python, tailored for data-intensive applications. This is a book about the parts of the Python language and libraries you'll need to effectively solve a broad set of data analysis problems. This book is *not* an exposition on analytical methods using Python as the implementation language.

When I say "data", what am I referring to exactly? The primary focus is on *structured data*, a deliberately vague term that encompasses many different common forms of data, such as

- Multidimensional arrays (matrices)
- Tabular or spreadsheet-like data in which each column may be a different type (string, numeric, date, or otherwise). This includes most kinds of data commonly stored in relational databases or tab- or comma-delimited text files
- Multiple tables of data interrelated by key columns (what would be primary or foreign keys for a SQL user)
- Evenly or unevenly spaced time series

This is by no means a complete list. Even though it may not always be obvious, a large percentage of data sets can be transformed into a structured form that is more suitable for analysis and modeling. I not, it may be possible to extract features from a data set into a structured form. As an example, a collection of news articles could be processed into a word frequency table which could then be used t perform sentiment analysis.

Most users of spreadsheet programs like Microsoft Excel, perhaps the most widely used data analysis tool in the world, will not be strangers to these kinds of data.

Why Python for Data Analysis?

For many people (myself among them), the Python language is easy to fall in love with. Since its firs appearance in 1991, Python has become one of the most popular dynamic, programming languages, along with Perl, Ruby, and others. Python and Ruby have become especially popular in recent years for building websites using their numerous web frameworks, like Rails (Ruby) and Django (Python). Such languages are often called *scripting* languages as they can be used to write quick-and-dirty sma programs, or *scripts*. I don't like the term "scripting language" as it carries a connotation that they cannot be used for building mission-critical software. Among interpreted languages Python is distinguished by its large and active *scientific computing* community. Adoption of Python for scientific computing in both industry applications and academic research has increased significantly since the early 2000s.

For data analysis and interactive, exploratory computing and data visualization, Python will inevitabl draw comparisons with the many other domain-specific open source and commercial programming languages and tools in wide use, such as R, MATLAB, SAS, Stata, and others. In recent years, Python's improved library support (primarily pandas) has made it a strong alternative for data manipulation tasks. Combined with Python's strength in general purpose programming, it is an excellent choice as a single language for building data-centric applications.

Python as Glue

Part of Python's success as a scientific computing platform is the ease of integrating C, C++, and FORTRAN code. Most modern computing environments share a similar set of legacy FORTRAN and C libraries for doing linear algebra, optimization, integration, fast fourier transforms, and other such algorithms. The same story has held true for many companies and national labs that have used Python to glue together 30 years' worth of legacy software.

Most programs consist of small portions of code where most of the time is spent, with large amounts of "glue code" that doesn't run often. In many cases, the execution time of the glue code is insignificant; effort is most fruitfully invested in optimizing the computational bottlenecks, sometimes by moving the code to a lower-level language like C.

In the last few years, the Cython project (http://cython.org) has become one of the preferred ways of both creating fast compiled extensions for Python and also interfacing with C and C++ code.

Solving the "Two-Language" Problem

In many organizations, it is common to research, prototype, and test new ideas using a more domain-specific computing language like MATLAB or R then later port those ideas to be part of a larger production system written in, say, Java, C#, or C++. What people are increasingly finding is that Python is a suitable language not only for doing research and prototyping but also building the production systems, too. I believe that more and more companies will go down this path as there are often significant organizational benefits to having both scientists and technologists using the same se of programmatic tools.

Why Not Python?

While Python is an excellent environment for building computationally-intensive scientific applications and building most kinds of general purpose systems, there are a number of uses for whic Python may be less suitable.

As Python is an interpreted programming language, in general most Python code will run substantiall slower than code written in a compiled language like Java or C++. As *programmer time* is typically more valuable than *CPU time*, many are happy to make this tradeoff. However, in an application with very low latency requirements (for example, a high frequency trading system), the time spent programming in a lower-level, lower-productivity language like C++ to achieve the maximum possible performance might be time well spent.

Python is not an ideal language for highly concurrent, multithreaded applications, particularly applications with many CPU-bound threads. The reason for this is that it has what is known as the *global interpreter lock* (GIL), a mechanism which prevents the interpreter from executing more than one Python bytecode instruction at a time. The technical reasons for why the GIL exists are beyond th scope of this book, but as of this writing it does not seem likely that the GIL will disappear anytime soon. While it is true that in many big data processing applications, a cluster of computers may be required to process a data set in a reasonable amount of time, there are still situations where a single-process, multithreaded system is desirable.

This is not to say that Python cannot execute truly multithreaded, parallel code; that code just cannot be executed in a single Python process. As an example, the Cython project features easy integration with OpenMP, a C framework for parallel computing, in order to to parallelize loops and thus significantly speed up numerical algorithms.

Essential Python Libraries

For those who are less familiar with the scientific Python ecosystem and the libraries used throughou the book, I present the following overview of each library.

NumPy

NumPy, short for Numerical Python, is the foundational package for scientific computing in Python. The majority of this book will be based on NumPy and libraries built on top of NumPy. It provides, among other things:

- A fast and efficient multidimensional array object *ndarray*
- Functions for performing element-wise computations with arrays or mathematical operations between arrays
- Tools for reading and writing array-based data sets to disk
- Linear algebra operations, Fourier transform, and random number generation
- Tools for integrating C, C++, and Fortran code to Python

Beyond the fast array-processing capabilities that NumPy adds to Python, one of its primary purpose with regards to data analysis is as the primary container for data to be passed between algorithms. Fo numerical data, NumPy arrays are a much more efficient way of storing and manipulating data than the other built-in Python data structures. Also, libraries written in a lower-level language, such as C o Fortran, can operate on the data stored in a NumPy array without copying any data.

pandas

pandas provides rich data structures and functions designed to make working with structured data fas easy, and expressive. It is, as you will see, one of the critical ingredients enabling Python to be a powerful and productive data analysis environment. The primary object in pandas that will be used in this book is the `DataFrame`, a two-dimensional tabular, column-oriented data structure with both row and column labels:

```
>>> frame
 total_bill  tip   sex     smoker  day  time    size
1   16.99       1.01  Female  No      Sun  Dinner  2
2   10.34       1.66  Male    No      Sun  Dinner  3
3   21.01       3.5   Male    No      Sun  Dinner  3
4   23.68       3.31  Male    No      Sun  Dinner  2
5   24.59       3.61  Female  No      Sun  Dinner  4
6   25.29       4.71  Male    No      Sun  Dinner  4
7   8.77        2     Male    No      Sun  Dinner  2
8   26.88       3.12  Male    No      Sun  Dinner  4
9   15.04       1.96  Male    No      Sun  Dinner  2
10  14.78       3.23  Male    No      Sun  Dinner  2
```

pandas combines the high performance array-computing features of NumPy with the flexible data manipulation capabilities of spreadsheets and relational databases (such as SQL). It provides sophisticated indexing functionality to make it easy to reshape, slice and dice, perform aggregations, and select subsets of data. pandas is the primary tool that we will use in this book.

For financial users, pandas features rich, high-performance time series functionality and tools well-suited for working with financial data. In fact, I initially designed pandas as an ideal tool for financia

data analysis applications.

For users of the R language for statistical computing, the DataFrame name will be familiar, as the object was named after the similar R `data.frame` object. They are not the same, however; the functionality provided by `data.frame` in R is essentially a strict subset of that provided by the panda `DataFrame`. While this is a book about Python, I will occasionally draw comparisons with R as it is one of the most widely-used open source data analysis environments and will be familiar to many readers.

The pandas name itself is derived from *panel data*, an econometrics term for multidimensional structured data sets, and *Python data analysis* itself.

matplotlib

matplotlib is the most popular Python library for producing plots and other 2D data visualizations. It was originally created by John D. Hunter (JDH) and is now maintained by a large team of developers It is well-suited for creating plots suitable for publication. It integrates well with IPython (see below) thus providing a comfortable interactive environment for plotting and exploring data. The plots are also *interactive*; you can zoom in on a section of the plot and pan around the plot using the toolbar in the plot window.

IPython

IPython is the component in the standard scientific Python toolset that ties everything together. It provides a robust and productive environment for interactive and exploratory computing. It is an enhanced Python shell designed to accelerate the writing, testing, and debugging of Python code. It is particularly useful for interactively working with data and visualizing data with matplotlib. IPython is usually involved with the majority of my Python work, including running, debugging, and testing code.

Aside from the standard terminal-based IPython shell, the project also provides

- A Mathematica-like HTML notebook for connecting to IPython through a web browser (more on this later).
- A Qt framework-based GUI console with inline plotting, multiline editing, and syntax highlightin
- An infrastructure for interactive parallel and distributed computing

I will devote a chapter to IPython and how to get the most out of its features. I strongly recommend using it while working through this book.

SciPy

SciPy is a collection of packages addressing a number of different standard problem domains in scientific computing. Here is a sampling of the packages included:

- `scipy.integrate`: numerical integration routines and differential equation solvers
- `scipy.linalg`: linear algebra routines and matrix decompositions extending beyond those provided in `numpy.linalg`.
- `scipy.optimize`: function optimizers (minimizers) and root finding algorithms
- `scipy.signal`: signal processing tools

- `scipy.sparse`: sparse matrices and sparse linear system solvers
- `scipy.special`: wrapper around SPECFUN, a Fortran library implementing many common mathematical functions, such as the gamma function
- `scipy.stats`: standard continuous and discrete probability distributions (density functions, samplers, continuous distribution functions), various statistical tests, and more descriptive statistics
- `scipy.weave`: tool for using inline C++ code to accelerate array computations

Together NumPy and SciPy form a reasonably complete computational replacement for much of MATLAB along with some of its add-on toolboxes.

Installation and Setup

Since everyone uses Python for different applications, there is no single solution for setting up Pytho and required add-on packages. Many readers will not have a complete scientific Python environment suitable for following along with this book, so here I will give detailed instructions to get set up on each operating system. I recommend using one of the following base Python distributions:

- Enthought Python Distribution: a scientific-oriented Python distribution from Enthought (http://www.enthought.com). This includes EPDFree, a free base scientific distribution (with NumPy, SciPy, matplotlib, Chaco, and IPython) and EPD Full, a comprehensive suite of more tha 100 scientific packages across many domains. EPD Full is free for academic use but has an annua subscription for non-academic users.
- Python(x,y) (http://pythonxy.googlecode.com): A free scientific-oriented Python distribution for Windows.

I will be using EPDFree for the installation guides, though you are welcome to take another approach depending on your needs. At the time of this writing, EPD includes Python 2.7, though this might change at some point in the future. After installing, you will have the following packages installed ar importable:

- Scientific Python base: NumPy, SciPy, matplotlib, and IPython. These are all included in EPDFre
- IPython Notebook dependencies: tornado and pyzmq. These are included in EPDFree.
- pandas (version 0.8.2 or higher).

At some point while reading you may wish to install one or more of the following packages: statsmodels, PyTables, PyQt (or equivalently, PySide), xlrd, lxml, basemap, pymongo, and requests. These are used in various examples. Installing these optional libraries is not necessary, and I would would suggest waiting until you need them. For example, installing PyQt or PyTables from source or OS X or Linux can be rather arduous. For now, it's most important to get up and running with the bar minimum: EPDFree and pandas.

For information on each Python package and links to binary installers or other help, see the Python Package Index (PyPI, http://pypi.python.org). This is also an excellent resource for finding new Python packages.

NOTE

To avoid confusion and to keep things simple, I am avoiding discussion of more complex environment management tools like pip and virtualenv. There are many excellent guides available for these tools on the Internet.

CAUTION

Some users may be interested in alternate Python implementations, such as IronPython, Jython, or PyPy. To make use of the tools presented in this book, it is (currently) necessary to use the standard C-based Python interpreter, known as CPython.

Windows

To get started on Windows, download the EPDFree installer from http://www.enthought.com, which should be an MSI installer named like `epd_free-7.3-1-win-x86.msi`. Run the installer and accept the default installation location `C:\Python27`. If you had previously installed Python in this location, you may want to delete it manually first (or using Add/Remove Programs).

Next, you need to verify that Python has been successfully added to the system path and that there are no conflicts with any prior-installed Python versions. First, open a command prompt by going to the Start Menu and starting the Command Prompt application, also known as `cmd.exe`. Try starting the Python interpreter by typing `python`. You should see a message that matches the version of EPDFree you installed:

```
C:\Users\Wes>python
Python 2.7.3 |EPD_free 7.3-1 (32-bit)| (default, Apr 12 2012, 14:30:37) on win32
Type "credits", "demo" or "enthought" for more information.
>>>
```

If you see a message for a different version of EPD or it doesn't work at all, you will need to clean up your Windows environment variables. On Windows 7 you can start typing "environment variables" in the programs search field and select `Edit environment variables for your account`. On Windows XP, you will have to go to `Control Panel > System > Advanced > Environment Variables`. On the window that pops up, you are looking for the `Path` variable. It needs to contain the following two directory paths, separated by semicolons:

```
C:\Python27;C:\Python27\Scripts
```

If you installed other versions of Python, be sure to delete any other Python-related directories from both the system and user `Path` variables. After making a path alternation, you have to restart the command prompt for the changes to take effect.

Once you can launch Python successfully from the command prompt, you need to install pandas. The easiest way is to download the appropriate binary installer from http://pypi.python.org/pypi/pandas. For EPDFree, this should be `pandas-0.9.0.win32-py2.7.exe`. After you run this, let's launch IPython and check that things are installed correctly by importing pandas and making a simple matplotlib plot:

```
C:\Users\Wes>ipython --pylab
Python 2.7.3 |EPD_free 7.3-1 (32-bit)|
Type "copyright", "credits" or "license" for more information.

IPython 0.12.1 -- An enhanced Interactive Python.
```

```
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

Welcome to pylab, a matplotlib-based Python environment [backend: WXAgg].
For more information, type 'help(pylab)'.

In [1]: import pandas

In [2]: plot(arange(10))
```

If successful, there should be no error messages and a plot window will appear. You can also check that the IPython HTML notebook can be successfully run by typing:

```
$ ipython notebook --pylab=inline
```

CAUTION

If you use the IPython notebook application on Windows and normally use Internet Explorer, you will likely need to install and run Mozilla Firefox or Google Chrome instead.

EPDFree on Windows contains only 32-bit executables. If you want or need a 64-bit setup on Windows, using EPD Full is the most painless way to accomplish that. If you would rather install from scratch and not pay for an EPD subscription, Christoph Gohlke at the University of California, Irvine, publishes unofficial binary installers for all of the book's necessary packages (http://www.lfd.uci.edu/~gohlke/pythonlibs/) for 32- and 64-bit Windows.

Apple OS X

To get started on OS X, you must first install Xcode, which includes Apple's suite of software development tools. The necessary component for our purposes is the gcc C and C++ compiler suite. The Xcode installer can be found on the OS X install DVD that came with your computer or downloaded from Apple directly.

Once you've installed Xcode, launch the terminal (Terminal.app) by navigating to `Applications > Utilities`. Type `gcc` and press enter. You should hopefully see something like:

```
$ gcc
i686-apple-darwin10-gcc-4.2.1: no input files
```

Now you need to install EPDFree. Download the installer which should be a disk image named something like `epd_free-7.3-1-macosx-i386.dmg`. Double-click the `.dmg` file to mount it, then double-click the `.mpkg` file inside to run the installer.

When the installer runs, it automatically appends the EPDFree executable path to your `.bash_profile` file. This is located at `/Users/your_uname/.bash_profile`:

```
# Setting PATH for EPD_free-7.3-1
PATH="/Library/Frameworks/Python.framework/Versions/Current/bin:${PATH}"
export PATH
```

Should you encounter any problems in the following steps, you'll want to inspect your `.bash_profile` and potentially add the above directory to your path.

Now, it's time to install pandas. Execute this command in the terminal:

```
$ sudo easy_install pandas
Searching for pandas
Reading http://pypi.python.org/simple/pandas/
Reading http://pandas.pydata.org
Reading http://pandas.sourceforge.net
Best match: pandas 0.9.0
Downloading http://pypi.python.org/packages/source/p/pandas/pandas-0.9.0.zip
Processing pandas-0.9.0.zip
Writing /tmp/easy_install-H5mIX6/pandas-0.9.0/setup.cfg
Running pandas-0.9.0/setup.py -q bdist_egg --dist-dir /tmp/easy_install-H5mIX6/
pandas-0.9.0/egg-dist-tmp-RhLG0z
Adding pandas 0.9.0 to easy-install.pth file

Installed /Library/Frameworks/Python.framework/Versions/7.3/lib/python2.7/
site-packages/pandas-0.9.0-py2.7-macosx-10.5-i386.egg
Processing dependencies for pandas
Finished processing dependencies for pandas
```

To verify everything is working, launch IPython in Pylab mode and test importing pandas then makin a plot interactively:

```
$ ipython --pylab
22:29 ~/VirtualBox VMs/WindowsXP $ ipython
Python 2.7.3 |EPD_free 7.3-1 (32-bit)| (default, Apr 12 2012, 11:28:34)
Type "copyright", "credits" or "license" for more information.

IPython 0.12.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

Welcome to pylab, a matplotlib-based Python environment [backend: WXAgg].
For more information, type 'help(pylab)'.

In [1]: import pandas

In [2]: plot(arange(10))
```

If this succeeds, a plot window with a straight line should pop up.

GNU/Linux

> **NOTE**
>
> Some, but not all, Linux distributions include sufficiently up-to-date versions of all the required Python packages and can be installed using the built-in package management tool like `apt`. I detail setup using EPDFree as it's easily reproducible across distributions.

Linux details will vary a bit depending on your Linux flavor, but here I give details for Debian-based GNU/Linux systems like Ubuntu and Mint. Setup is similar to OS X with the exception of how EPDFree is installed. The installer is a shell script that must be executed in the terminal. Depending on whether you have a 32-bit or 64-bit system, you will either need to install the `x86` (32-bit) or `x86_64` (64-bit) installer. You will then have a file named something similar to `epd_free-7.3-1-`

rh5-x86_64.sh. To install it, execute this script with bash:

```
$ bash epd_free-7.3-1-rh5-x86_64.sh
```

After accepting the license, you will be presented with a choice of where to put the EPDFree files. I recommend installing the files in your home directory, say /home/wesm/epd (substituting your own username for wesm).

Once the installer has finished, you need to add EPDFree's bin directory to your $PATH variable. If you are using the bash shell (the default in Ubuntu, for example), this means adding the following pa addition in your .bashrc:

```
export PATH=/home/wesm/epd/bin:$PATH
```

Obviously, substitute the installation directory you used for /home/wesm/epd/. After doing this you can either start a new terminal process or execute your .bashrc again with source ~/.bashrc.

You need a C compiler such as gcc to move forward; many Linux distributions include gcc, but other may not. On Debian systems, you can install gcc by executing:

```
sudo apt-get install gcc
```

If you type gcc on the command line it should say something like:

```
$ gcc
gcc: no input files
```

Now, time to install pandas:

```
$ easy_install pandas
```

If you installed EPDFree as root, you may need to add sudo to the command and enter the sudo or ro password. To verify things are working, perform the same checks as in the OS X section.

Python 2 and Python 3

The Python community is currently undergoing a drawn-out transition from the Python 2 series of interpreters to the Python 3 series. Until the appearance of Python 3.0, all Python code was backward compatible. The community decided that in order to move the language forward, certain backwards incompatible changes were necessary.

I am writing this book with Python 2.7 as its basis, as the majority of the scientific Python communit has not yet transitioned to Python 3. The good news is that, with a few exceptions, you should have n trouble following along with the book if you happen to be using Python 3.2.

Integrated Development Environments (IDEs)

When asked about my standard development environment, I almost always say "IPython plus a text editor". I typically write a program and iteratively test and debug each piece of it in IPython. It is als useful to be able to play around with data interactively and visually verify that a particular set of data manipulations are doing the right thing. Libraries like pandas and NumPy are designed to be easy-to-use in the shell.

However, some will still prefer to work in an IDE instead of a text editor. They do provide many nice "code intelligence" features like completion or quickly pulling up the documentation associated with functions and classes. Here are some that you can explore:

- Eclipse with PyDev Plugin
- Python Tools for Visual Studio (for Windows users)
- PyCharm
- Spyder
- Komodo IDE

Community and Conferences

Outside of an Internet search, the scientific Python mailing lists are generally helpful and responsive to questions. Some ones to take a look at are:

- pydata: a Google Group list for questions related to Python for data analysis and pandas
- pystatsmodels: for statsmodels or pandas-related questions
- numpy-discussion: for NumPy-related questions
- scipy-user: for general SciPy or scientific Python questions

I deliberately did not post URLs for these in case they change. They can be easily located via Interne search.

Each year many conferences are held all over the world for Python programmers. PyCon and EuroPython are the two main general Python conferences in the United States and Europe, respectively. SciPy and EuroSciPy are scientific-oriented Python conferences where you will likely find many "birds of a feather" if you become more involved with using Python for data analysis after reading this book.

Navigating This Book

If you have never programmed in Python before, you may actually want to start at the *end* of the boo where I have placed a condensed tutorial on Python syntax, language features, and built-in data structures like tuples, lists, and dicts. These things are considered prerequisite knowledge for the remainder of the book.

The book starts by introducing you to the IPython environment. Next, I give a short introduction to th key features of NumPy, leaving more advanced NumPy use for another chapter at the end of the book Then, I introduce pandas and devote the rest of the book to data analysis topics applying pandas, NumPy, and matplotlib (for visualization). I have structured the material in the most incremental wa possible, though there is occasionally some minor cross-over between chapters.

Data files and related material for each chapter are hosted as a git repository on GitHub:

```
http://github.com/pydata/pydata-book
```

I encourage you to download the data and use it to replicate the book's code examples and experimen with the tools presented in each chapter. I will happily accept contributions, scripts, IPython notebooks, or any other materials you wish to contribute to the book's repository for all to enjoy.

Code Examples

Most of the code examples in the book are shown with input and output as it would appear executed i the IPython shell.

```
In [5]: code
Out[5]: output
```

At times, for clarity, multiple code examples will be shown side by side. These should be read left to right and executed separately.

```
In [5]: code          In [6]: code2
Out[5]: output        Out[6]: output2
```

Data for Examples

Data sets for the examples in each chapter are hosted in a repository on GitHub: http://github.com/pydata/pydata-book. You can download this data either by using the git revision control command-line program or by downloading a zip file of the repository from the website.

I have made every effort to ensure that it contains everything necessary to reproduce the examples, b I may have made some mistakes or omissions. If so, please send me an e-mail: `wesmckinn@gmail.com`.

Import Conventions

The Python community has adopted a number of naming conventions for commonly-used modules:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

This means that when you see `np.arange`, this is a reference to the `arange` function in NumPy. This done as it's considered bad practice in Python software development to import everything (`from numpy import *`) from a large package like NumPy.

Jargon

I'll use some terms common both to programming and data science that you may not be familiar wit Thus, here are some brief definitions:

Munge/Munging/Wrangling

: Describes the overall process of manipulating unstructured and/or messy data into a structured or clean form. The word has snuck its way into the jargon of many modern day data hackers. Munge rhymes with "lunge".

Pseudocode

: A description of an algorithm or process that takes a code-like form while likely not being actual valid source code.

Syntactic sugar

: Programming syntax which does not add new features, but makes something more convenient or easier to type.

Acknowledgements

It would have been difficult for me to write this book without the support of a large number of peopl

On the O'Reilly staff, I'm very grateful for my editors Meghan Blanchette and Julie Steele who

guided me through the process. Mike Loukides also worked with me in the proposal stages and helpe make the book a reality.

I received a wealth of technical review from a large cast of characters. In particular, Martin Blais and Hugh Brown were incredibly helpful in improving the book's examples, clarity, and organization fro cover to cover. James Long, Drew Conway, Fernando Pérez, Brian Granger, Thomas Kluyver, Adam Klein, Josh Klein, Chang She, and Stéfan van der Walt each reviewed one or more chapters, providin pointed feedback from many different perspectives.

I got many great ideas for examples and data sets from friends and colleagues in the data community among them: Mike Dewar, Jeff Hammerbacher, James Johndrow, Kristian Lum, Adam Klein, Hilary Mason, Chang She, and Ashley Williams.

I am of course indebted to the many leaders in the open source scientific Python community who've built the foundation for my development work and gave encouragement while I was writing this book the IPython core team (Fernando Pérez, Brian Granger, Min Ragan-Kelly, Thomas Kluyver, and others), John Hunter, Skipper Seabold, Travis Oliphant, Peter Wang, Eric Jones, Robert Kern, Josef Perktold, Francesc Alted, Chris Fonnesbeck, and too many others to mention. Several other people provided a great deal of support, ideas, and encouragement along the way: Drew Conway, Sean Taylo Giuseppe Paleologo, Jared Lander, David Epstein, John Krowas, Joshua Bloom, Den Pilsworth, John Myles-White, and many others I've forgotten.

I'd also like to thank a number of people from my formative years. First, my former AQR colleagues who've cheered me on in my pandas work over the years: Alex Reyfman, Michael Wong, Tim Sarge Oktay Kurbanov, Matthew Tschantz, Roni Israelov, Michael Katz, Chris Uga, Prasad Ramanan, Ted Square, and Hoon Kim. Lastly, my academic advisors Haynes Miller (MIT) and Mike West (Duke).

On the personal side, Casey Dinkin provided invaluable day-to-day support during the writing proces tolerating my highs and lows as I hacked together the final draft on top of an already overcommitted schedule. Lastly, my parents, Bill and Kim, taught me to always follow my dreams and to never settl for less.

Chapter 2. Introductory Examples

This book teaches you the Python tools to work productively with data. While readers may have man different end goals for their work, the tasks required generally fall into a number of different broad groups:

Interacting with the outside world

: Reading and writing with a variety of file formats and databases.

Preparation

: Cleaning, munging, combining, normalizing, reshaping, slicing and dicing, and transforming data for analysis.

Transformation

: Applying mathematical and statistical operations to groups of data sets to derive new data sets. F example, aggregating a large table by group variables.

Modeling and computation

: Connecting your data to statistical models, machine learning algorithms, or other computational tools

Presentation

: Creating interactive or static graphical visualizations or textual summaries

In this chapter I will show you a few data sets and some things we can do with them. These examples are just intended to pique your interest and thus will only be explained at a high level. Don't worry if you have no experience with any of these tools; they will be discussed in great detail throughout the rest of the book. In the code examples you'll see input and output prompts like `In [15]:`; these are from the IPython shell.

> **NOTE**
>
> To follow along with these examples, you should run IPython in Pylab mode by running `ipython --pylab` at the command prompt.

1.usa.gov data from bit.ly

In 2011, URL shortening service bit.ly partnered with the United States government website `usa.gov` to provide a feed of anonymous data gathered from users who shorten links ending with `.gov` or `.mil` As of this writing, in addition to providing a live feed, hourly snapshots are available as downloadabl text files.[1]

In the case of the hourly snapshots, each line in each file contains a common form of web data known as JSON, which stands for JavaScript Object Notation. For example, if we read just the first line of a file you may see something like

```
In [15]: path = 'ch02/usagov_bitly_data2012-03-16-1331923249.txt'

In [16]: open(path).readline()
```

```
Out[16]: '{ "a": "Mozilla\\/5.0 (Windows NT 6.1; WOW64) AppleWebKit\\/535.11
(KHTML, like Gecko) Chrome\\/17.0.963.78 Safari\\/535.11", "c": "US", "nk": 1,
"tz": "America\\/New_York", "gr": "MA", "g": "A6qOVH", "h": "wfLQtf", "l":
"orofrog", "al": "en-US,en;q=0.8", "hh": "1.usa.gov", "r":
"http:\\/\\/www.facebook.com\\/l\\/7AQEFzjSi\\/1.usa.gov\\/wfLQtf", "u":
"http:\\/\\/www.ncbi.nlm.nih.gov\\/pubmed\\/22415991", "t": 1331923247, "hc":
1331822918, "cy": "Danvers", "ll": [ 42.576698, -70.954903 ] }\n'
```

Python has numerous built-in and 3rd party modules for converting a JSON string into a Python dictionary object. Here I'll use the `json` module and its `loads` function invoked on each line in the sample file I downloaded:

```
import json
path = 'ch02/usagov_bitly_data2012-03-16-1331923249.txt'
records = [json.loads(line) for line in open(path)]
```

If you've never programmed in Python before, the last expression here is called a *list comprehension* which is a concise way of applying an operation (like `json.loads`) to a collection of strings or other objects. Conveniently, iterating over an open file handle gives you a sequence of its lines. The resulting object `records` is now a list of Python dicts:

```
In [18]: records[0]
Out[18]:
{u'a': u'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.11 (KHTML, like
Gecko) Chrome/17.0.963.78 Safari/535.11',
 u'al': u'en-US,en;q=0.8',
 u'c': u'US',
 u'cy': u'Danvers',
 u'g': u'A6qOVH',
 u'gr': u'MA',
 u'h': u'wfLQtf',
 u'hc': 1331822918,
 u'hh': u'1.usa.gov',
 u'l': u'orofrog',
 u'll': [42.576698, -70.954903],
 u'nk': 1,
 u'r': u'http://www.facebook.com/l/7AQEFzjSi/1.usa.gov/wfLQtf',
 u't': 1331923247,
 u'tz': u'America/New_York',
 u'u': u'http://www.ncbi.nlm.nih.gov/pubmed/22415991'}
```

Note that Python indices start at 0 and not 1 like some other languages (like R). It's now easy to access individual values within records by passing a string for the key you wish to access:

```
In [19]: records[0]['tz']
Out[19]: u'America/New_York'
```

The `u` here in front of the quotation stands for *unicode*, a standard form of string encoding. Note that IPython shows the time zone string object *representation* here rather than its print equivalent:

```
In [20]: print records[0]['tz']
America/New_York
```

Counting Time Zones in Pure Python

Suppose we were interested in the most often-occurring time zones in the data set (the `tz` field). Ther are many ways we could do this. First, let's extract a list of time zones again using a list

comprehension:

```
In [25]: time_zones = [rec['tz'] for rec in records]
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/home/wesm/book_scripts/whetting/<ipython> in <module>()
----> 1 time_zones = [rec['tz'] for rec in records]

KeyError: 'tz'
```

Oops! Turns out that not all of the records have a time zone field. This is easy to handle as we can add the check `if 'tz' in rec` at the end of the list comprehension:

```
In [26]: time_zones = [rec['tz'] for rec in records if 'tz' in rec]

In [27]: time_zones[:10]
Out[27]:
[u'America/New_York',
 u'America/Denver',
 u'America/New_York',
 u'America/Sao_Paulo',
 u'America/New_York',
 u'America/New_York',
 u'Europe/Warsaw',
 u'',
 u'',
 u'']
```

Just looking at the first 10 time zones we see that some of them are unknown (empty). You can filter these out also but I'll leave them in for now. Now, to produce counts by time zone I'll show two approaches: the harder way (using just the Python standard library) and the easier way (using pandas). One way to do the counting is to use a dict to store counts while we iterate through the time zones:

```
def get_counts(sequence):
    counts = {}
    for x in sequence:
        if x in counts:
            counts[x] += 1
        else:
            counts[x] = 1
    return counts
```

If you know a bit more about the Python standard library, you might prefer to write the same thing more briefly:

```
from collections import defaultdict

def get_counts2(sequence):
    counts = defaultdict(int) # values will initialize to 0
    for x in sequence:
     counts[x] += 1
    return counts
```

I put this logic in a function just to make it more reusable. To use it on the time zones, just pass the `time_zones` list:

```
In [31]: counts = get_counts(time_zones)
```

```
In [32]: counts['America/New_York']
Out[32]: 1251

In [33]: len(time_zones)
Out[33]: 3440
```

If we wanted the top 10 time zones and their counts, we have to do a little bit of dictionary acrobatics

```
def top_counts(count_dict, n=10):
    value_key_pairs = [(count, tz) for tz, count in count_dict.items()]
    value_key_pairs.sort()
    return value_key_pairs[-n:]
```

We have then:

```
In [35]: top_counts(counts)
Out[35]:
[(33, u'America/Sao_Paulo'),
 (35, u'Europe/Madrid'),
 (36, u'Pacific/Honolulu'),
 (37, u'Asia/Tokyo'),
 (74, u'Europe/London'),
 (191, u'America/Denver'),
 (382, u'America/Los_Angeles'),
 (400, u'America/Chicago'),
 (521, u''),
 (1251, u'America/New_York')]
```

If you search the Python standard library, you may find the `collections.Counter` class, which mak this task a lot easier:

```
In [49]: from collections import Counter

In [50]: counts = Counter(time_zones)

In [51]: counts.most_common(10)
Out[51]:
[(u'America/New_York', 1251),
 (u'', 521),
 (u'America/Chicago', 400),
 (u'America/Los_Angeles', 382),
 (u'America/Denver', 191),
 (u'Europe/London', 74),
 (u'Asia/Tokyo', 37),
 (u'Pacific/Honolulu', 36),
 (u'Europe/Madrid', 35),
 (u'America/Sao_Paulo', 33)]
```

Counting Time Zones with pandas

The main pandas data structure is the *DataFrame*, which you can think of as representing a table or spreadsheet of data. Creating a DataFrame from the original set of records is simple:

```
In [289]: from pandas import DataFrame, Series

In [290]: import pandas as pd; import numpy as np

In [291]: frame = DataFrame(records)
```

```
In [292]: frame
Out[292]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3560 entries, 0 to 3559
Data columns:
_heartbeat_    120  non-null values
a              3440  non-null values
al             3094  non-null values
c              2919  non-null values
cy             2919  non-null values
g              3440  non-null values
gr             2919  non-null values
h              3440  non-null values
hc             3440  non-null values
hh             3440  non-null values
kw             93  non-null values
l              3440  non-null values
ll             2919  non-null values
nk             3440  non-null values
r              3440  non-null values
t              3440  non-null values
tz             3440  non-null values
u              3440  non-null values
dtypes: float64(4), object(14)

In [293]: frame['tz'][:10]
Out[293]:
0     America/New_York
1       America/Denver
2     America/New_York
3    America/Sao_Paulo
4     America/New_York
5     America/New_York
6        Europe/Warsaw
7
8
9
Name: tz
```

The output shown for the `frame` is the *summary view*, shown for large DataFrame objects. The Series object returned by `frame['tz']` has a method `value_counts` that gives us what we're looking for:

```
In [294]: tz_counts = frame['tz'].value_counts()

In [295]: tz_counts[:10]
Out[295]:
America/New_York       1251
                        521
America/Chicago         400
America/Los_Angeles     382
America/Denver          191
Europe/London            74
Asia/Tokyo               37
Pacific/Honolulu         36
Europe/Madrid            35
America/Sao_Paulo        33
```

Then, we might want to make a plot of this data using plotting library, matplotlib. You can do a bit o munging to fill in a substitute value for unknown and missing time zone data in the records. The `fillna` function can replace missing (NA) values and unknown (empty strings) values can be replace by boolean array indexing:

```
In [296]: clean_tz = frame['tz'].fillna('Missing')

In [297]: clean_tz[clean_tz == ''] = 'Unknown'

In [298]: tz_counts = clean_tz.value_counts()

In [299]: tz_counts[:10]
Out[299]:
America/New_York       1251
Unknown                 521
America/Chicago         400
America/Los_Angeles     382
America/Denver          191
Missing                 120
Europe/London            74
Asia/Tokyo               37
Pacific/Honolulu         36
Europe/Madrid            35
```

Making a horizontal bar plot can be accomplished using the `plot` method on the `counts` objects:

```
In [301]: tz_counts[:10].plot(kind='barh', rot=0)
```

See Figure 2-1 for the resulting figure. We'll explore more tools for working with this kind of data. For example, the `a` field contains information about the browser, device, or application used to perform the URL shortening:

```
In [302]: frame['a'][1]
Out[302]: u'GoogleMaps/RochesterNY'

In [303]: frame['a'][50]
Out[303]: u'Mozilla/5.0 (Windows NT 5.1; rv:10.0.2) Gecko/20100101 Firefox/10.0.2'

In [304]: frame['a'][51]
Out[304]: u'Mozilla/5.0 (Linux; U; Android 2.2.2; en-us; LG-P925/V10e Build/FRG83G)
AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1'
```