Finance 5350 - Fall 2016

Computational Financial Modeling

These notes are based off the material by Jake VanderPlas on his GitHub repository. A big thanks goes to Dr. VanderPlas for making these notebooks available under the "No Rights Reserved" CC0 license!

These notes form essentially the first chapter of his forthcoming book, which looks fantastic:

> A Whirlwind Tour of Python by Jake VanderPlas (O'Reilly). Copyright 2016 O'Reilly Media, Inc., 978-1-491-96465-1.

1. Introduction

Conceived in the late 1980s as a teaching and scripting language, Python has since become an essential tool for many programmers, engineers, researchers, and data scientists across academia and industry. As an astronomer focused on building and promoting the free open tools for data-intensive science, I've found Python to be a near-perfect fit for the types of problems I face day to day, whether it's extracting meaning from large astronomical datasets, scraping and munging data sources from the Web, or automating day-to-day research tasks.

The appeal of Python is in its simplicity and beauty, as well as the convenience of the large ecosystem of domain-specific tools that have been built on top of it. For example, most of the Python code in scientific computing and data science is built around a group of mature and useful packages:

- NumPy provides efficient storage and computation for multi-dimensional data arrays.
- SciPy contains a wide array of numerical tools such as numerical integration and interpolation.
- Pandas provides a DataFrame object along with a powerful set of methods to manipulate, filter, group, and transform data.
- Matplotlib provides a useful interface for creation of publication-quality plots and figures.
- Scikit-Learn provides a uniform toolkit for applying common machine learning algorithms to data.
- IPython/Jupyter provides an enhanced terminal and an interactive notebook environment that is useful for exploratory analysis, as well as creation of interactive, executable documents. For example, the manuscript for this report was composed entirely in Jupyter notebooks.

No less important are the numerous other tools and packages which accompany these: if there is a scientific or data analysis task you want to perform, chances are someone has written a package that will do it for you.

To tap into the power of this data science ecosystem, however, first requires familiarity with the Python language itself. I often encounter students and colleagues who have (sometimes extensive) backgrounds in computing in some language – MATLAB, IDL, R, Java, C++, etc. – and are looking for a brief but comprehensive tour of the Python language that respects their level of knowledge rather than starting from ground zero. This report seeks to fill that niche.

As such, this report in no way aims to be a comprehensive introduction to programming, or a full introduction to the Python language itself; if that is what you are looking for, you might check out one of the recommended references listed in Resources for Learning. Instead, this will provide a whirlwind tour of some of Python's essential syntax and semantics, built-in data types and structures, function definitions, control flow statements, and other aspects of the language. My aim is that readers will walk away with a solid foundation from which to explore the data science stack just outlined.

Using Code Examples

Installation and Practical Considerations

Installing Python and the suite of libraries that enable scientific computing is straightforward whether you use Windows, Linux, or Mac OS X. This section will outline some of the considerations when setting up your computer.

Python 2 vs Python 3

This report uses the syntax of Python 3, which contains language enhancements that are not compatible with the *2.x* series of Python. Though Python 3.0 was first released in 2008, adoption has been relatively slow, particularly in the scientific and web development communities. This is primarily because it took some time for many of the essential packages and toolkits to be made compatible with the new language internals. Since early 2014, however, stable releases of the most important tools in the data science ecosystem have been fully-compatible with both Python 2 and 3, and so this book will use the newer Python 3 syntax. Even though that is the case, the vast majority of code snippets in this book will also work without modification in Python 2: in cases where a Py2-incompatible syntax is used, I will make every effort to note it explicitly.

Installation with conda

Though there are various ways to install Python, the one I would suggest – particularly if you wish to eventually use the data science tools mentioned above – is via the cross-platform Anaconda distribution. There are two flavors of the Anaconda distribution:

- Miniconda gives you Python interpreter itself, along with a command-line tool called `conda` which operates as a cross-platform package manager geared toward Python packages, similar in spirit to the `apt` or `yum` tools that Linux users might be familiar with.
- Anaconda includes both Python and `conda`, and additionally bundles a suite of other pre-installed packages geared toward scientific computing.

Any of the packages included with Anaconda can also be installed manually on top of Miniconda; for this reason I suggest starting with Miniconda.

To get started, download and install the Miniconda package – make sure to choose a version with Python 3 – and then install the IPython notebook package:

```
[~]$ conda install ipython-notebook
```

For more information on `conda`, including information about creating and using conda environments, refer to the Miniconda package documentation linked at the above page.

The Zen of Python

Python aficionados are often quick to point out how "intuitive", "beautiful", or "fun" Python is. While I tend to agree, I also recognize that beauty, intuition, and fun often go hand in hand with familiarity, and so for those familiar with other languages such florid sentiments can come across as a bit smug. Nevertheless, I hope that if you give Python a chance, you'll see where such impressions might come from. And if you *really* want to dig into the programming philosophy that drives much of the coding practice of Python power-users, a nice little Easter egg exists in the Python interpreter: simply close your eyes, meditate for a few minutes, and `import this`:

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

With that, let's start our tour of the Python language.