Python for data science

Tal Yarkoni

Why do data science in Python?

- Easy to learn
- Readable, explicit syntax
- Most packages are well documented
  - E.g., scikit-learn's documentation is widely held up as a model
- A huge number of tutorials, guides, and other educational materials

Comprehensive standard library

- The Python standard library contains many high-quality modules
- Always check it before you write your own tools!
- Examples:
  - **os**: operating system tools (now partly superseded by pathlib)
  - **re**: regular expressions
  - **collections**: useful data structures
  - **multiprocessing**: simple parallelization tools
  - **pickle**: serialization
  - **json**: reading and writing JSON
  - **itertools**: tools for constructing/applying iterators
  - **functools**: support for higher-order functions

Exceptional external libraries

- Python has very good (often best-in-class) third-party packages for many applications
- Particularly important for data science, which draws on a broad toolkit
- Package management is (relatively) easy—conda and pip
- Examples:
  - Back-end web development: flask, Django
  - Database ORMs: SQLAlchemy, Django ORM
  - Scraping/parsing text/markup: beautifulsoup, scrapy
  - Natural language processing (NLP): nltk, gensim, textblob
  - Numerical computation and data analysis: numpy, scipy, pandas, xarray
  - Machine learning: scikit-learn, Tensorflow, PyTorch, keras
  - Image processing: pillow, scikit-image, OpenCV
  - Visualization/plotting: matplotlib, seaborn, altair, ggplot, Bokeh
  - Testing: py.test

(Relatively) good performance

- Python is a high-level dynamic language--this comes at a performance cost
- For many (not all!) data scientists, performance is irrelevant most of the time
- In general, the less Python code you write yourself, the better your performance will be
  - Much of the standard library consists of Python interfaces to C functions
  - Numpy, scikit-learn, TensorFlow, etc. all rely heavily on C or C++

[48]:
```
# Create a list of 100,000 integers
my_list = list(range(100000, ))
```

Built-in sum()

[49]:
```
# Python's built-in sum() function is pretty fast
%timeit sum(my_list)
```

```
442 µs ± 20.2 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

What if we do it ourselves, naively?

[50]:
```
def ill_write_my_own_sum_thank_you_very_much(l):
    s = 0
    for elem in my_list:
        s += elem
    return s

%timeit ill_write_my_own_sum_thank_you_very_much(my_list)
```

```
4.26 ms ± 86.7 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

The built-in version is about 10 times faster.

The community

- Python is one of the ~3 most popular programming languages, so the community is *huge*
- Incredible range and number of educational resources
- Almost every common problem has been answered multiple times on Stack Overflow
- Community is slanted heavily towards software developers and data science (as opposed to statistics, as in R)
  - Trade-off: poorer support for stats, but quality of software is higher

If you need more speed…

- Write code in a way that plays to Python's strengths
  - E.g., for array operations vectorize in numpy, don't use loops on lists!
- Cython (a superset of Python) allows C type declarations and function calls
- Rapid progress on just-in-time compilers that optimize certain subsets of Python code very effectively
- See Ariel Rokem's *High-performance Python* talk

Python vs. other DS languages

- Python competes for data science mind share with many languages
  - Most notably, R
  - To a lesser extent, Matlab, Julia, Mathematica, SAS, Java, Scala, etc…
- Let's briefly review a couple of these
  - I'll *try* to be objective, but I might fail
  - In a few years, Julia might be a serious part of the conversation

R

- Dominant in traditional statistics and some fields of science
  - Has attracted many SAS, SPSS, and Stata users
- Exceptional statistics support; hundreds of best-in-class libraries
- Designed to make data analysis and visualization as easy as possible
- Often slow (but, as always “it depends”)
- Language quirks drive many experienced software developers crazy
- Less support for most things non-data-related

Matlab

- A proprietary numerical computing language still widely used in engineering and some scientific disciplines
- Good performance and very active development, but expensive
- Closed ecosystem, relatively few third-party libraries
  - There is an open-source port (Octave)
- Not suitable for use as a general-purpose language
- Feature-wise, clearly playing catch-up to open source languages at this point

So, why Python?

- Why choose Python over other languages?
- Arguably none of these offers the same combination of readability, flexibility, libraries, and performance
- Python is sometimes described as "the second best language for everything"
- But: this doesn't mean you should always use Python!
  - Depends on your needs, community, etc.
  - A very rough heuristic: if you're mainly doing traditional statistical inference (e.g., mixed LMMs, causal inference, SEM, etc.), stick with R. Otherwise, Python.

You can have your cake and eat it

- Many languages—particularly R—now interface almost seamlessly with Python
  - E.g., RPy2
- You can work primarily in Python, fall back on R when you need it (or vice versa)
- The best of all possible worlds?

The DS stack: a look ahead

- The Python ecosystem contains tens of thousands of packages
- Several are very widely used in data science applications:
  - Jupyter: interactive notebooks
  - Numpy: numerical computing in Python
  - Scipy: scientific Python tools
  - Matplotlib: plotting in Python
  - pandas: data structures for Python
  - scikit-learn: machine learning in Python
- We've already covered Jupyter
- Other tutorials will go into much greater detail on the rest