TWO SIGMA

www.twosigma.com

Python Data Wrangling: Preparing for the Future

Wes McKinney @wesmckinn

DataEngConf NYC 2016

October 26, 2016

Disclaimer

Important Legal Information

The information presented here is offered for informational purposes only and should not be used for any other purpose (including, without limitation, the making of investment decisions). Examples provided herein are for illustrative purposes only and are not necessarily based on actual data. Nothing herein constitutes: an offer to sell or the solicitation of any offer to buy any security or other interest; tax advice; or investment advice. This presentation shall remain the property of Two Sigma Investments, LP ("Two Sigma") and Two Sigma reserves the right to require the return of this presentation at any time.

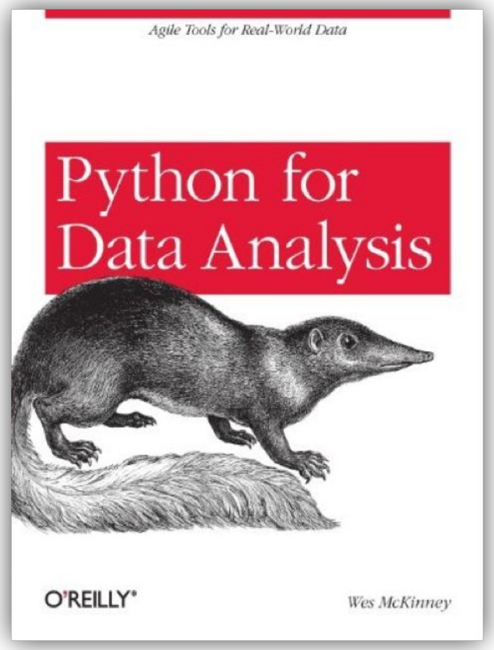
Me

- Creator of Python pandas project (ca. 2008)
- PMC member for Apache Arrow and Parquet
- Currently: Data technology at Two Sigma
- Past
  - Cloudera: Python on Hadoop/Spark + Ibis project
  - DataPad: Co-founder/CEO

Python for Data Analysis, 2nd Edition

Coming in 2017

- Updated for pandas 1.0
- Remove deprecated / out-of-date functionality
- New content: Advanced pandas, intro to statsmodels, scikit-learn



The Python data community

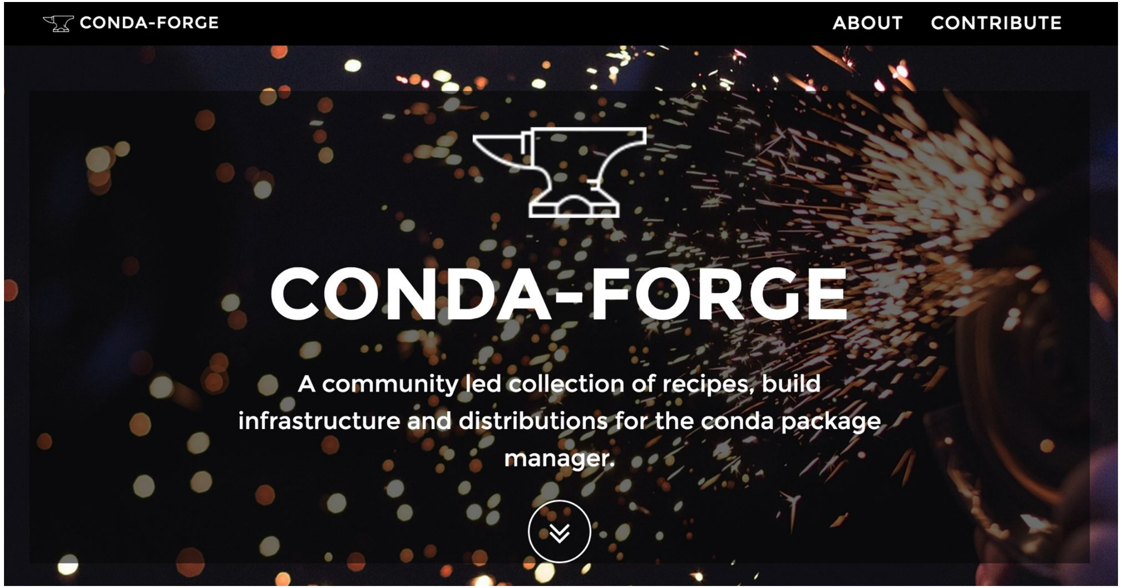
- Python has grown from a niche scientific computing language in 2011 to a mainstream data science language now in 2016
- A language of choice for latest-gen ML: Keras, Tensorflow, Theano
- Worldwide ecosystem of conferences and meetups: PyData, SciPy, etc.

How / why did the community grow?

- Confluence of important projects
  - **Array computing, linear algebra:** NumPy, Cython, SciPy
  - **Data manipulation tools:** pandas
  - **Statistics / Machine Learning**: statsmodels, scikit-learn
  - **Programming interfaces**: Jupyter notebooks
  - **Packaging**: {Ana, mini}conda, portable wheels (.whl) for pip

conda-forge: community-led conda packaging

```
conda config --add channels conda-forge
```



NumFOCUS: Not-for-profit open source sponsorship

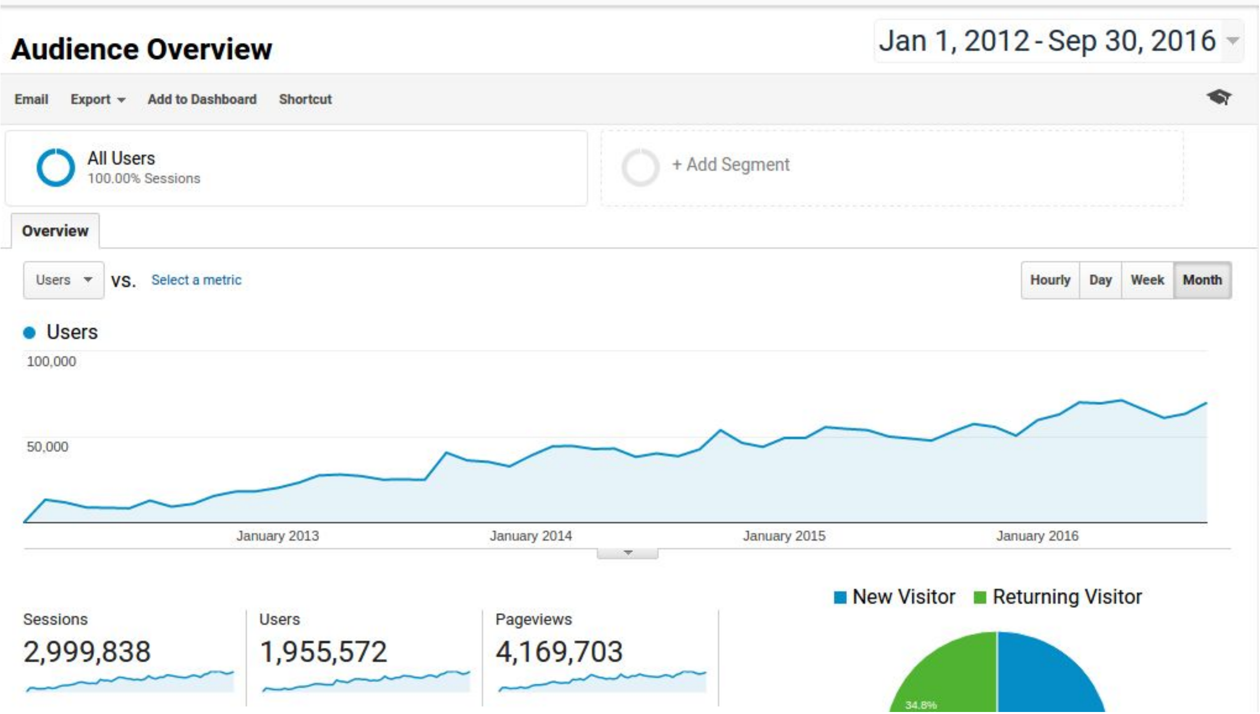
NUMFOCUS

OPEN CODE = BETTER SCIENCE

pandas, the project

- https://github.com/pandas-dev/pandas
- Latest version: 0.19.0, released October 3, 2016
- Code base turned 8 years old in April 2016
- Over 600 distinct contributors, 14,000+ commits
- > 5300 pull requests, > 7200 issues closed on GitHub
- Only 11 developers with 100 or more commits

Estimating the pandas user base size



pandas.pydata.org

50-70K unique visitors per month

Sources of pandas's popularity

- Responsive (volunteer) core developers: code reviewed and bugs fixed quickly
- Easy to use, good performance
- One of the only mature data preparation tools available. Most data is not clean
- Strong suite of time series / event analytics
- Library has been battle tested by tens of thousands of users
- Good learning resources (multiple books, etc.)

pandas development, where to from here?

- pandas has accumulated much technical debt, problems stemming from early software architecture decisions
- pandas being used increasingly as a building block in distributed systems like Spark and dask
- Sprawling codebase: over 200K lines of code
- In works: **pandas 1.0**
    - Instead of pandas v0.{n + 1}
    - API stable release, focusing only on stability, bug fixes, performance improvements

Some known pandas problems

- Missing data inconsistencies
- Excess / unpredictable memory use
- Mutability issues, defensive memory copying
- Difficult to add new data types
- Mostly single-threaded, GIL-contention issues
- Degrading micro-performance from complex pure Python internals
- Costly interoperability with file formats, other systems (Dask, Spark)

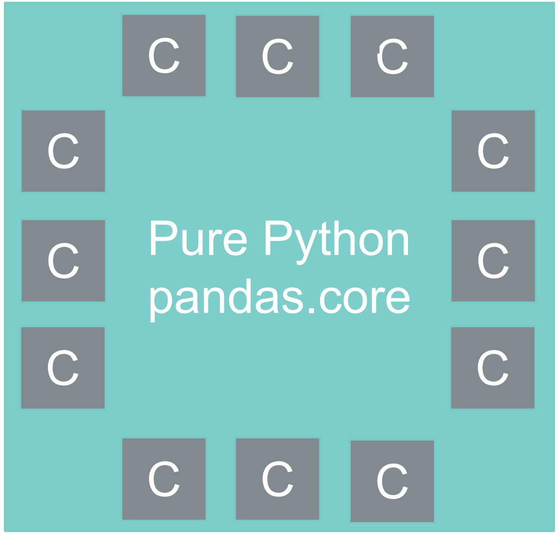
See also: 2013 talk "10 Things I Hate About pandas"

pandas 2.0

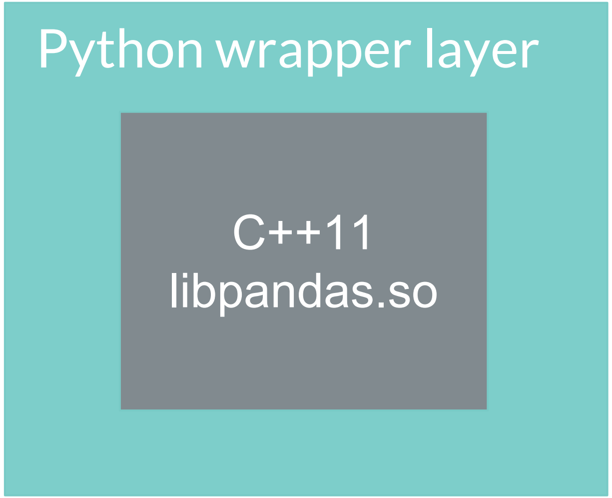
- Re-architecting of pandas Series / DataFrame internals
- Backwards compatible for 90+% of user code
- Goals
  - Improved performance
  - Lower and more predictable memory requirements
  - Better utilize hardware with many cores and large amounts of RAM
  - Tighter integration with external data

libpandas: a modern C++ core "engine"

pandas 0.x/1.x



pandas 2.x



pandas 2.0: some libpandas benefits

- Encapsulate memory management and low-level details of Series and DataFrame
- Isolate user from low-level data issues (e.g. missing data in integers, booleans)
- Better performance for “cheap” operations (e.g. indexing / selection)
- Easier access to multithreading primitives, hardware optimizations

- **Note:** Similar to the architecture used by Tensorflow and other modern scientific tools targeting Python

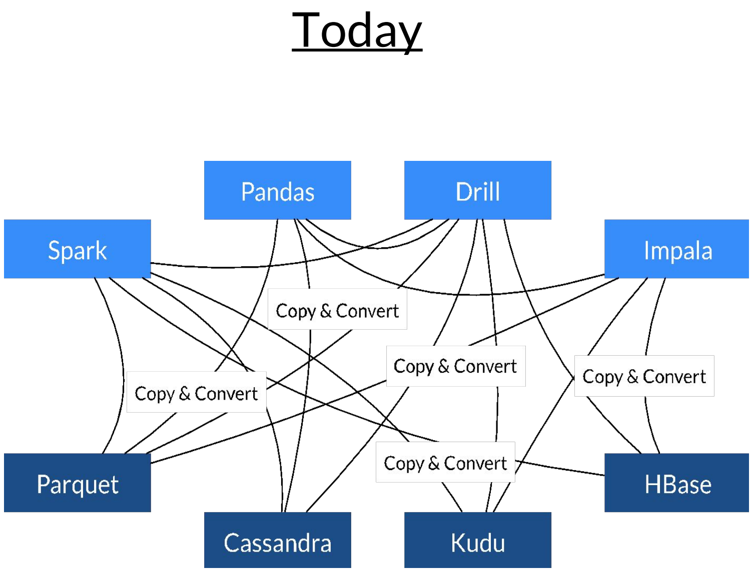
pandas 2.0: memory mapping and fast serialization

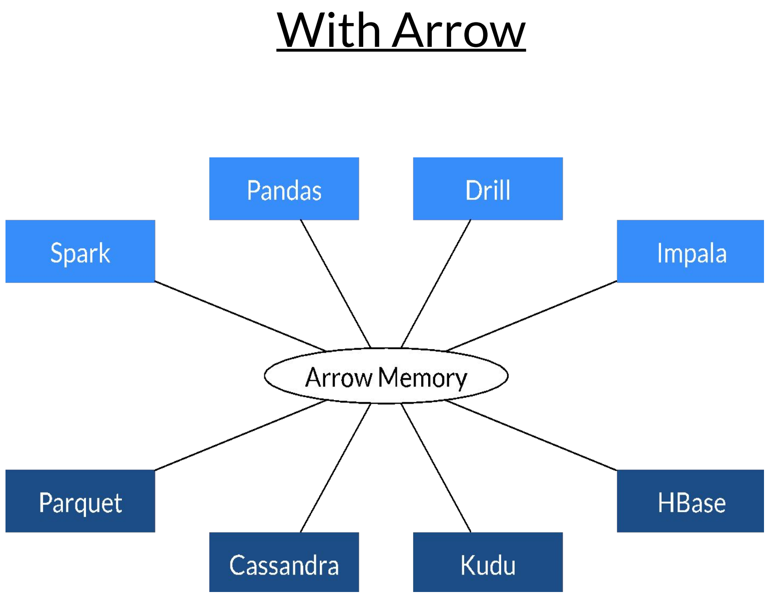
- Provide for memory-mapped DataFrames or “lazy-loading” of data from disk
- Improved performance with fast disk formats
  - HDF5, feather, bcolz, zarr, etc.
- Faster data movement in distributed applications
  - Dask, Spark

Apache Arrow: Process and Move Data Fast

- New Top-level Apache project as of February 2016
- Collaboration amongst broad set of OSS projects around shared needs
- Language-independent columnar data structures
- Metadata for describing schemas / chunks of data
- Protocol for moving data between processes with minimal serialization overhead

High performance data interchange





Source: Apache Arrow

File formats and memory interoperability

- Improving IO throughput essential to good in-memory pandas performance
- Output data quickly to other tools/systems (R, Spark, Dask, etc.)

Arrow in action: Feather file format

- https://github.com/wesm/feather
- Cross-language binary DataFrame format
  - Write from Python, read in R, and vice versa
- Collaboration with Hadley Wickham from the R community

Arrow in action: Parquet files in Python

- Parquet C++ project https://github.com/apache/parquet-cpp
- Parquet files read into Arrow arrays
- Array arrays written back to Parquet files

```
import pyarrow.parquet as pq
table = pq.read_table(path, **options)
```

Some serialization benchmarks

- Benchmark setup
  - 1 GB DataFrame of floating point data, 100 columns
  - Pickle, Parquet, bcolz, Feather format, Arrow IPC format

Benchmark results

Tool	Absolute time	Bandwidth
Arrow IPC	148 ms	6.77 GB/s
pandas.HDFStore	178 ms	5.62 GB/s
pickle	296 ms	3.38 GB/s
bcolz	498 ms	2.01 GB/s

Wall clock RAM-to-RAM conversion time from source to 1 GB pandas.DataFrame with 100 float64 columns (no missing data). HDF5 using tmpfs

Benchmark results

Tool	Absolute time	Bandwidth
Arrow IPC	148 ms	6.77 GB/s
pandas.HDFStore	178 ms	5.62 GB/s
pickle	296 ms	3.38 GB/s
bcolz	498 ms	2.01 GB/s
pandas.read_csv	21247 ms	0.047 GB /s

Wall clock RAM-to-RAM conversion time from source to 1 GB pandas.DataFrame with 100 float64 columns (no missing data). HDF5 using tmpfs

Benefits of standard columnar formats

- Readable for other systems (even JVM-based), easier interoperability
- On-disk filtering / subsetting possible
- Benefit from development work by other communities

Thank you

- Conference organizers
- pandas developer community
- Apache Software Foundation

- Note: Opinions are my own