# pandas: a python data analysis library

Wes McKinney[1]

[1]AQR

New York Financial Python Users Group 12/15/2009

# Outline

1. Motivation
   - Technology for quantitative finance
2. pandas
   - Origins
   - Data structures
   - Applications

# Some common financial research tasks

- Data manipulation
  - Raw data series are transformed into asset scores
  - Handle missing observations, time series of different frequencies, other sources of heterogeneity
- Portfolio construction, backtesting
  - Transform scores into tradable portfolios
  - Analyzing historical strategy performance
- Statistical estimation
  - Econometric analysis: linear regression and other more advanced models
  - Modeling risk: forecasting portfolio volatility

# Widely used research technology

- Commercial: MATLAB, Stata, eViews, etc.
- Open-source: R, others
- Frequently little code reuse (with exceptions, of course, e.g. CRAN)
- Typical workflow: research in one of the above, implement for real in C++, Java, etc.

# How does Python compare?

- NumPy provides a comparable (and often superior) array object and wonderfully extensible API
- Ability to use low-level code (C, Fortran, Cython, SWIG) can bridge performance gaps
- Python as a language is great for building larger systems
- But existing statistical modeling and econometrics libraries are relatively weak
- Pythonistas are often left creating their own tools, or using Python to prepare data sets for use in the other languages

# My goal

- Help Python become a compelling environment for finance, economics research and other statistical applications
- Implement convenient statistical estimation routines
- Provide tools for interfacing with other libraries / languages

# pandas origins

- Open-sourced by AQR in 2009
- Idea: data structures which understand labeled data, are lightweight and easy-to-visualize
- Link identifiers (dates, tickers, data name) to numerical data
- Works well with both time-series and cross-sectional data
- Prevent common errors associated with heterogeneous data
- Etymology: **pan**el **da**ta system

# Basic building blocks: overview

- 1-dimensional: Series, TimeSeries
  - NumPy array subclass with item label vector (Index)
  - Both ndarray and dict-like
- 2-dimensional: DataFrame, DataMatrix
  - Represents a dict of Series objects
  - Conforms Series to a common Index
- 3-dimensional: WidePanel, LongPanel
  - Behave as a dict of DataMatrix objects
  - Three indices: `items`, `major_axis`, `minor_axis`

# Series functionality

- ndarray subclass with various conveniences
- Combining Series matches Index values
- Many ndarray functions overridden to respect the Index and to exclude missing values (represented as NaN)
- Operates in essence as an ordered, fixed-length dict

# Series example

```
>>> s              >>> s2             >>> s + s2
AAPL     3.4       AAPL     3.4       AAPL     6.8
GOOG     9.4       GOOG     9.4       GOOG     18.8
IBM      7.2       IBM      7.2       IBM      14.4
MSFT     1.5                          MSFT     nan
```

```
>>> s + s2.reindex(s.index).fill(0)
AAPL     6.8
GOOG     18.8
IBM      14.4
MSFT     1.5
```

# DataMatrix

- A 2D container for Series objects which enforces data alignment
- Arithmetic operations between DataFrames match on both row and column labels
- Operations between DataFrames and Series broadcast depending on context
- Aggregation, group by, reindexing methods
- "Frame" versus "Matrix": different implementations of the same fundamental data structure (with slightly different performance characteristics)

# DataMatrix example

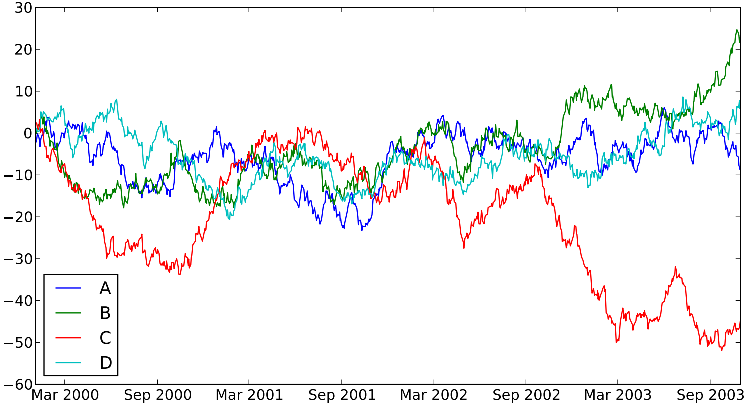
```
>>> dm
                         A            B            C            D
2000-01-03 00:00:00    -0.0415269   0.88057      -1.7492      0.548433
2000-01-04 00:00:00    0.166356     -0.955948    0.0679696    -0.26994
2000-01-05 00:00:00    0.954089     -0.0122938   -0.906397    -1.29401
2000-01-06 00:00:00    0.026427     0.643754     -0.167905    -0.35435
2000-01-07 00:00:00    1.786        -0.399267    0.858169     0.306875
2000-01-10 00:00:00    -0.850005    0.963422     -0.228602    1.12401
2000-01-11 00:00:00    -0.309066    1.28251      1.38328      -0.114106
2000-01-12 00:00:00    -0.0533492   0.328646     -1.52632     1.67064
2000-01-13 00:00:00    0.192041     0.79258      -1.0988      0.533673
2000-01-14 00:00:00    0.182775     0.325071     -0.288246    -1.05236
```

```
>>> dm.sum(axis=0)
A   2.05374198744
B   3.84904518163
C   -3.65605643845
D   1.09887239512
```

```
>>> dm.sum(axis=1)
2000-01-03 -0.361724
2000-01-04 -0.991562
2000-01-05 -1.258608
...
```

```
>>> dm['A']
2000-01-03   -0.0415269
2000-01-04   0.166356
2000-01-05   0.954089
...
```

- `dm.plot(); legend()`



# WidePanel

- Panel data stored in 3D ndarray ("wide" format)
- Slicing and aggregation produce DataMatrix objects
- Any axis can be reindexed
- Fast conversion to "long" (stacked) format for output to CSV or for regression analysis
- Recent addition to pandas, still a work in progress

# WidePanel example

```
>>> wp
<class 'pandas.core.panel.WidePanel'>
Dimensions: 2 (items) x 30 (major) x 4 (minor)
Items: Item1 to Item2
Major axis: 2000-01-03 00:00:00 to 2000-02-11 00:00:00
Minor axis: A to D
```

```
>>> wp.mean(axis='major')
     Item1          Item2
A    -0.079891      -0.175715
B    0.101753       0.102943
C    0.202509       0.133795
D    0.418179       -0.0465659
```

```
>>> wp.getMajorXS(wp.major_axis[0])
     Item1          Item2
A    -0.154756      -0.587252
B    -2.45935       0.666575
C    0.300465       2.51455
D    -1.14056       -1.33442
```

# LongPanel

- 2D “stacked” representation of panel data (balanced or unbalanced)
- Supports much of WidePanel functionality
- Built-in support for constructing dummy variables for regression modeling
- Can be lexicographically sorted

# LongPanel example

| Major | Minor | Item1 | Item2 |
|---|---|---|---|
| 2000-01-03 00:00:00 | A | -0.154756 | -0.587252 |
| 2000-01-03 00:00:00 | B | -2.45935 | 0.666575 |
| 2000-01-03 00:00:00 | C | 0.300465 | 2.51455 |
| 2000-01-03 00:00:00 | D | -1.14056 | -1.33442 |
| 2000-01-04 00:00:00 | A | -0.600915 | -0.156101 |
| 2000-01-04 00:00:00 | B | 1.37085 | 1.80986 |
| 2000-01-04 00:00:00 | C | -0.531156 | -1.93413 |
| 2000-01-04 00:00:00 | D | 0.713402 | 0.0890342 |
| 2000-01-05 00:00:00 | A | 0.470185 | -0.584578 |
| 2000-01-05 00:00:00 | B | 0.602647 | 0.510549 |
| 2000-01-05 00:00:00 | C | -0.0772073 | -2.62006 |
| 2000-01-05 00:00:00 | D | 0.157917 | -0.383053 |
| 2000-01-06 00:00:00 | A | -0.155252 | 0.50461 |
| 2000-01-06 00:00:00 | B | -0.0498417 | -0.616731 |
| 2000-01-06 00:00:00 | C | -0.44834 | -2.08052 |
| 2000-01-06 00:00:00 | D | -1.97119 | 0.353407 |
| 2000-01-07 00:00:00 | A | 0.995084 | -1.19059 |
| 2000-01-07 00:00:00 | B | -0.935756 | 2.31528 |

# Linear regression

- Idea: provide convenient high-level interface to commonly used stats routines
- Standard ordinary least squares: OLS
  - $y_t = \alpha + \overrightarrow{\beta} X_t + \varepsilon_t$
- Time-pooled cross-sectional regression: PanelOLS
  - $y_{it} = \alpha + \overrightarrow{\beta} X_{it} + \varepsilon_{it}$
  - With entity, time fixed effects: $y_{it} = \overrightarrow{\beta} X_{it} + \mu_i + \tau_t + \varepsilon_{it}$
- Corresponding with scikits.statsmodels developers, who are implementing other econometric models

# Regression output

```
>>> model = ols(y=Y, x=data)
Formula: Y ~ <A> + <B> + <C> + <intercept>

Number of Observations:         700
Number of Degrees of Freedom:   4

R-squared:         0.0026
Adj R-squared:    -0.0017

Rmse:              0.9800

F-stat (3, 696):     0.6013, p-value:     0.6143

Degrees of Freedom: model 3, resid 696

-----------------------Summary of Estimated Coefficients------------------------
      Variable       Coef    Std Err     t-stat    p-value    CI 2.5%   CI 97.5%
--------------------------------------------------------------------------------
             A     0.0252     0.0377       0.67     0.5031    -0.0486     0.0991
             B    -0.0154     0.0392      -0.39     0.6945    -0.0922     0.0614
             C     0.0403     0.0368       1.10     0.2731    -0.0317     0.1124
     intercept     0.0526     0.0371       1.42     0.1564    -0.0201     0.1252
```

# Additional regression functionality

- Supports rolling and expanding time series regressions (for forecasting)
- A few common adjustments for heteroskedasticity and autocorrelation (Newey-West correction, clustering)
- Does not try to reinvent every wheel available in R, Stata, etc., but rather to reduce how often you need to use another language

# Comparison with other Python libraries

- scikits.statsmodels
  - Implements statistical models using only NumPy and SciPy
  - pandas can hopefully be a companion library with high-level interfaces to scikits.statsmodels classes
- scikits.timeseries
  - Supports many kinds of time series-specific manipulations, built on numpy.MaskedArray
  - pandas provides support for generic date offsets and creating fixed frequency dates using the DateRange class. Could likely be enhanced by using scikits.timeseries functionality
  - pandas makes no intention of competing: rather it tries to be simple and easy-to-use for the majority of applications

# Comparison with other Python libraries

- tabular
  - A new library released recently providing spreadsheet-like functionality for 2D data
  - pandas is mostly intended for manipulating numerical data and building models with it
- Others?

# Ideas for future

- Expand existing functionality to address other applications
- Implement more statistical models / wrap scikits.statsmodels classes
- Develop seamless **rpy** interface to leverage CRAN wealth
- Better / more efficient IO functions for getting data into pandas

# Summary

- pandas provides a good starting place for working with time series and cross-sectional data sets
  - Let me know if you find it useful, or have suggestions
- Python use is growing in finance: with some more work we can further overcome the C++ / Java / MATLAB monoculture

# Thank you

- Cython and NumPy developers
- Jonathan Taylor and the scikits.statsmodels guys
- IPython devs: F. Perez et al
- Enthought, for inviting me

- Contact: wesmckinn@gmail.com
- Website: pandas.googlecode.com
- Official release on PyPI forthcoming

# AQR Disclaimer

The views and opinions expressed herein are those of the author and do not necessarily reflect the views of AQR Capital Management, LLC its affiliates, or its employees. The information set forth herein has been obtained or derived from sources believed by author to be reliable. However, the author does not make any representation or warranty, express or implied, as to the information's accuracy or completeness, nor does the author recommend that the attached information serve as the basis of any investment decision. This document has been provided to you solely for information purposes and does not constitute an offer or solicitation of an offer, or any advice or recommendation, to purchase any securities or other financial instruments, and may not be construed as such. This document is intended exclusively for the use of the person to whom it has been delivered by the author, and it is not to be reproduced or redistributed to any other person.