

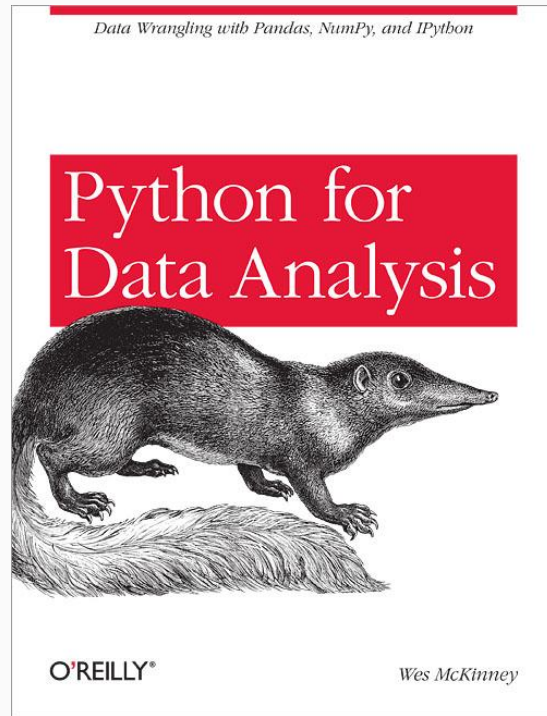
Pandas - not just for data scientists

Uzi Halaby Senerman | Chief Architect @ BlueVine



This talk is not...

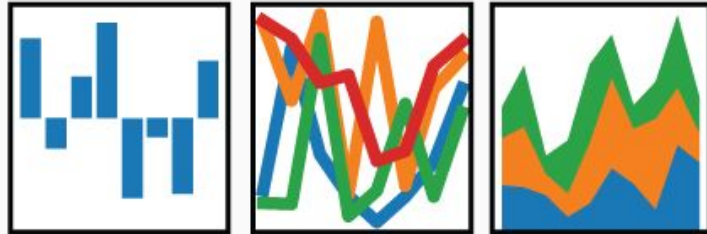
- for data scientists (but you're welcome to stay :-)
- a tutorial
 - Pandas tutorial by Brandon Rhodes from PyCon 2015: <https://www.youtube.com/watch?v=5JnMutdy6Fw>
 - Python for Data Analysis by Wes McKinney



This talk...

- is for Python developers
- will expose you to a very powerful tool that can be very useful from research phase to production


pandas
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



About me



- FinTech - Flexible business lines of credit and invoice factoring
- Reliable and fast risk assessment for potential customers
- Data science:
 - pandas as a major tool
 - Machine learning models
 - Starting to cope with “Big Data” problems



The screenshot displays the BlueVine dashboard. At the top, there are four summary cards: 'Total Available' (\$50,000.00), 'Total Used' (\$7,800.00), 'Total Available' (\$2,000.00), and 'Total Used' (\$2,200.00). Below these is a navigation menu with options like 'Home', 'Customers', 'Invoices', 'Reports', and 'Profile'. The main content area is titled 'Unpaid Invoices (5)' and contains a table with columns for Customer, Amount, Invoice, Invoice Date, Due Date, and Action. The table lists three invoices: 'Hobby Shop and Supplies' for \$2,200, 'Knapen' for \$2,200, and 'Target' for \$2,200. Each row has a 'Get paid now' button. Below the table, there is a section for 'Customer info' and 'Billing address' for the selected invoice.

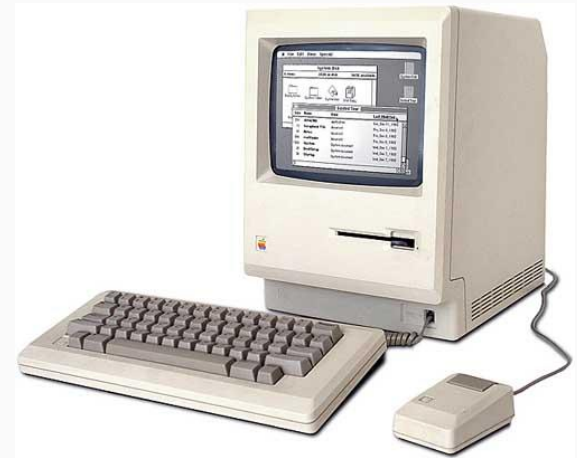
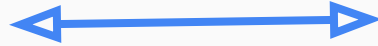
Customer	Amount	Invoice	Invoice Date	Due Date	Action
Hobby Shop and Supplies	\$2,200	1083	11/24/2015	1/26/2016	Get paid now
Knapen	\$2,200	1084	11/11/2015	1/16/2016	Get paid now
Target	\$2,200	1027	11/11/2014		Get paid now

Python - greatness that comes with a price

An interface between the human developer and the machine.



Probably the best general purpose programming language :-)



Not always the best option
(greatness comes with a price)

Specialized Python feature

For/list comprehensions

```
list_all = range(100000000)
filtered_list = []
for x in list_all:
    if x > 50000000:
        filtered_list.append(x)
```

CPU times: user 13.2 s, sys: 2.1 s
Wall time: 15.6 s

```
list_all = range(100000000)
filtered_list = [x for x in list_all if x > 50000000]
```

CPU times: user 7.74 s, sys: 3.26 s, total: 11 s
Wall time: 11.6 s

This is idiomatic Python and you should always prefer list comprehension when it's applicable

Leverage the advantages of C (with the greatness of Python)

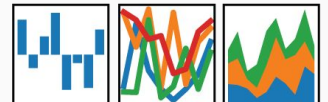
- Implement performance-critical parts of the code in C (with static typing)
 - “If you want python fast, fix **dynamic** not interpreted”
Brandon Rhodes: The Day of the EXE Is Upon Us - PyCon 2014
- "Python as a glue language"
- Many libraries, including some of the standard libraries in CPython
- Including NumPy & pandas...

NumPy & pandas

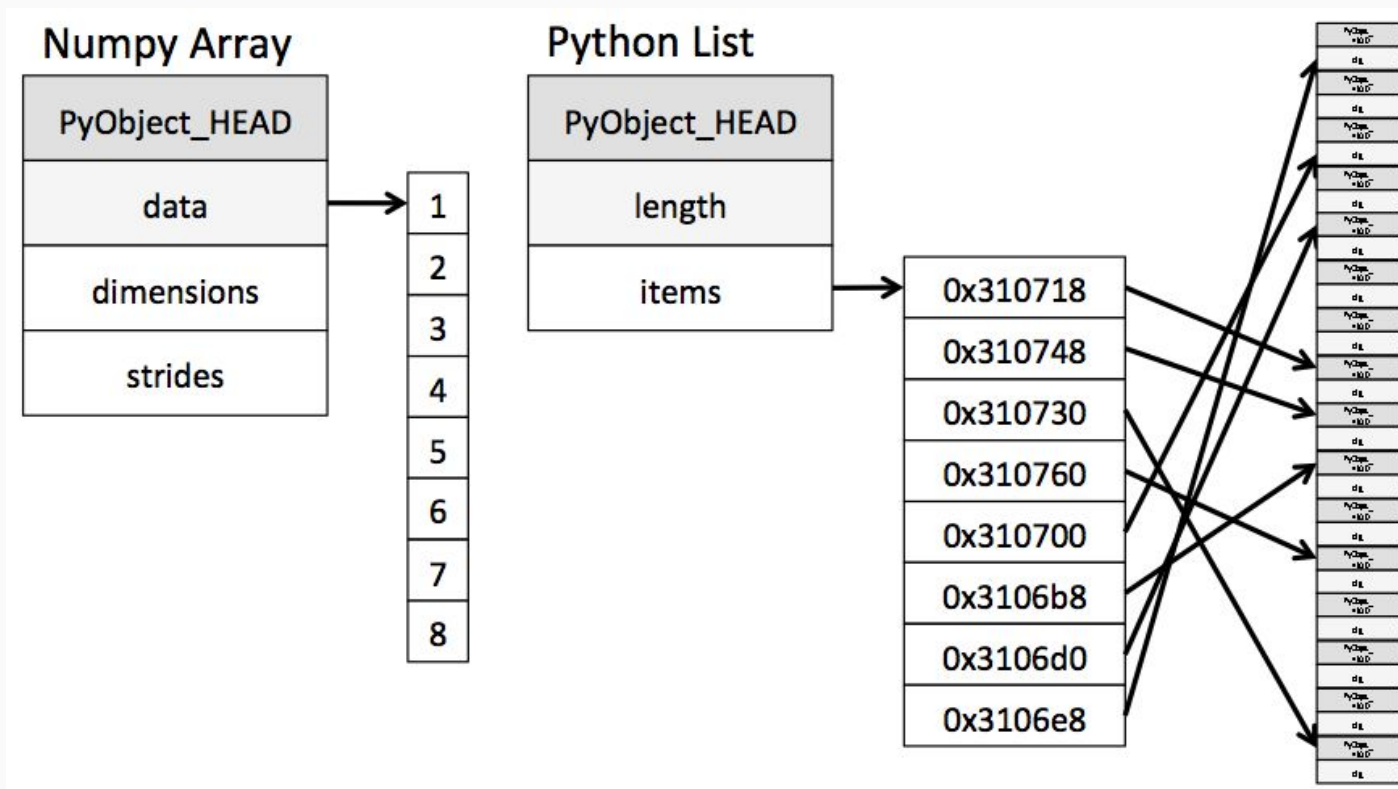
- pandas is highly optimized for performance, with critical code paths written in Cython or C
- NumPy array / pandas Series and DataFrame
 - Fixed size at creation
 - Elements are the same data type
 - ufuncs - vectorized version of many useful operations
- Highly flexible and powerful - everything you can do with a DB, Excel or R Data Frames



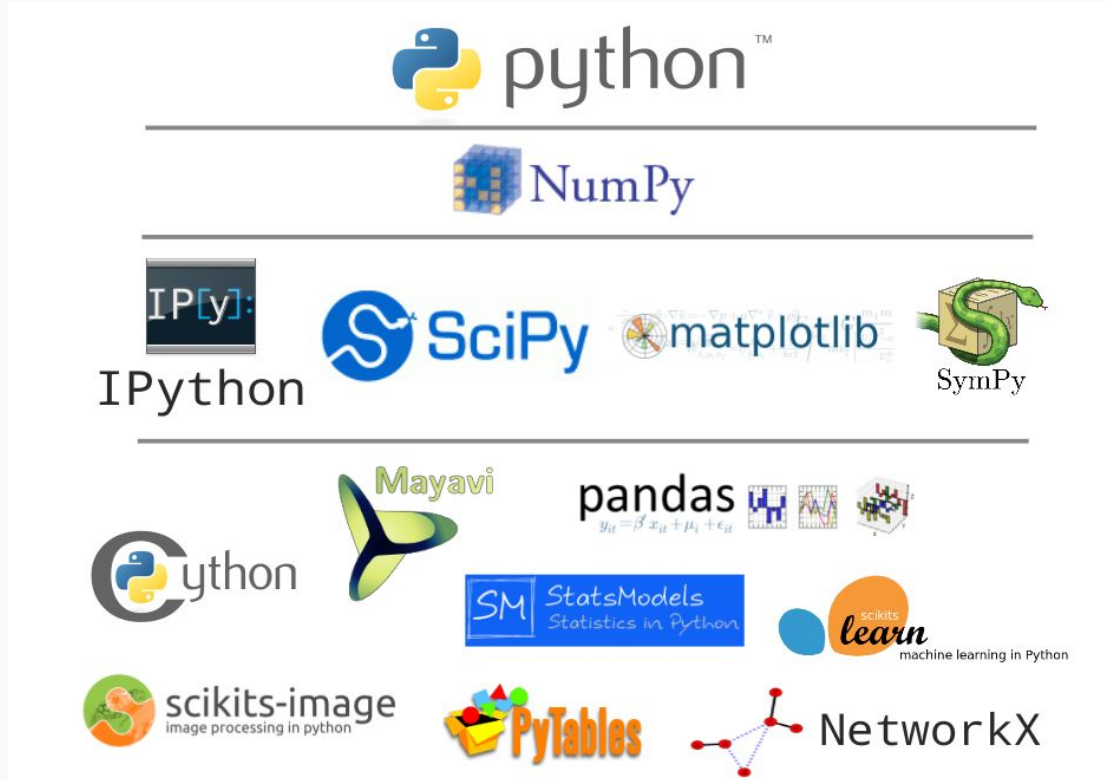
pandas
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



How can it improve performance



Entire Eco System



The logo consists of two orange curved lines forming a circle, with two small grey dots positioned at the top and bottom of the circle.

jupyter

How much faster is it?

```
l = list(range(100000000))  
a = np.array(l)
```

Without pandas

```
sum(l)
```

CPU times: user 1.32 s

```
filtered=[x for x in l if x>0.5]
```

CPU times: user 13.5 s, sys: 4.57 s

```
[x*999 for x in l]
```

CPU times: user 8.17 s

With pandas

```
a.sum()
```

CPU times: user 98.6 ms

```
filtered = a[a > 0.5]
```

CPU times: user 467 ms

```
a*999
```

CPU times: user 274 ms

Results in production - great performance boost

- Sync process that runs every several minutes
- Comparing hundreds of thousands of values
- External API vs. Django ORM



- X15 faster when moving to pandas
- Cleaner code

Results in production - WOW

- Calculating summaries for aggregated data
- Very complicated business logic



- X1900 faster when moving to pandas
- Much cleaner code
- Optimization for the non-pandas code is doable (it will probably won't be as good as with pandas), but the price would be MUCH more complicated code

The pandas way

- Work with pandas the way it was designed to be used
- ufunc (e.g. `sum()`) are better than `apply()`
- `apply()` is better than iterating over a Series/DataFrame
- (Iterating over a Series/DataFrame is better than iterating over a Python list/dict)
- And don't always follow the most intuitive way...

Twisting your mind

Date	Category
2015-01-02	A
2015-02-02	B
2015-01-12	A
2015-02-22	B
2015-03-08	?
2015-02-22	
2015-01-19	
2015-01-17	

50,000 rows

13 categories

From Date	To Date	Category to Assign
2015-01-02	2015-01-21	A
2015-01-22	2015-02-27	B
2015-02-28	2015-03-15	C
2015-03-15	2015-04-01	D

Twisting your mind

- Straight forward approach:
`df["category"] = df.apply(get_category)`
- The efficient approach:
`for from_date, to_date, category in periods:
 df.loc[(df['date'] >= from_date) &
 (df['date'] < to_date), 'category'] = category`
- **X2340** faster (26.1ms vs. 61 seconds)!!!

Data Exploration with Jupyter & pandas

- Very powerful tools to explore the data
- Run the same notebook in multiple environments (production, staging)
- Run the same notebook in different times
- Share notebook with other team members
- Or share only the results (HTML, PDF)
- Use the notebook as starting point for your production code

Learn pandas (and start using Jupyter)!

- Explore your data more effectively
- Optimize your code (and make it cleaner):
 - Data analysis
 - Sync processes
 - Reports / Exports
- And when you use pandas - remember that changing your point of view can lead you to more efficient implementation

Thank you!

uzi@bluevine.com

Extras

Specialized Python feature

Slots (you shouldn't use this in your code)

```
class Test:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c
```

```
for i in range(10000000):
    Test(i,i,i)
```

CPU times: user 6.57 s, sys: 40.1 ms
Wall time: 6.68 s

```
class Test:
    __slots__ = ['a', 'b', 'c']

    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c
```

```
for i in range(10000000):
    Test(i,i,i)
```

CPU times: user 5.41 s, sys: 33.8 ms
Wall time: 5.49 s