You don't need N dimensions when you have **pandas**

Pietro Battiston
Department of Economics, Management and Statistics
University of Milan-Bicocca
me@pietrobattiston.it

Pycon X
May 5, 2019 – Florence

Who am I?

Who am I?

- Researcher in Economics (discovered Python as a Math student)

Who am I?

- Researcher in Economics (discovered Python as a Math student)
- Working on social and economic networks (book in fall: "**Network responsibility** *How contagion shapes our societies, how our societies can shape contagion*")

Who am I?

- Researcher in Economics (discovered Python as a Math student)
- Working on social and economic networks (book in fall: "**Network responsibility** *How contagion shapes our societies, how our societies can shape contagion*")
- (Mostly) doing research with Open-Source software

Who am I?

- Researcher in Economics (discovered Python as a Math student)
- Working on social and economic networks (book in fall: "**Network responsibility** *How contagion shapes our societies, how our societies can shape contagion*")
- (Mostly) doing research with Open-Source software
- Only Python user **I** know of in my department

Who am I?

- Researcher in Economics (discovered Python as a Math student)
- Working on social and economic networks
  (book in fall: "**Network responsibility** *How contagion shapes our societies, how our societies can shape contagion*")
- (Mostly) doing research with Open-Source software
- Only Python user I know of in my department
  ...until next Tuesday!

Who am I?

- Researcher in Economics (discovered Python as a Math student)
- Working on social and economic networks (book in fall: "**Network responsibility** *How contagion shapes our societies, how our societies can shape contagion*")
- (Mostly) doing research with Open-Source software
- Only Python user I know of in my department
  ...until next Tuesday!
- **pandas** core dev

Disclaimer

[. . . speaker bores audience about personal issues. . .]

What is **pandas**?

- *The* Python library for data manipulation and analysis

What is **pandas**?

- *The* Python library for data manipulation and analysis
- *The* way for pythonists who do data analysis not to feel inferior to “**R**ists”, with their dataframes (well on the contrary!)

What is **pandas**?

- *The* Python library for data manipulation and analysis
- *The* way for pythonists who do data analysis not to feel inferior to “**R**ists”, with their dataframes (well on the contrary!)
- In pratice, “just”:
  - data structures that (heavily) extend `numpy`’s `arrays`

What is **pandas**?

- *The* Python library for data manipulation and analysis
- *The* way for pythonists who do data analysis not to feel inferior to “**R**ists”, with their dataframes (well on the contrary!)
- In pratice, “just”:
  - data structures that (heavily) extend `numpy`’s `arrays`
  - (a lot of) additional utilities (IO, datetime. . .)

pandas in one equation

numpy : `list` = **pandas** : `dict`

pandas in one equation

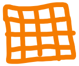
$$\textbf{numpy} : \texttt{list} = \textbf{pandas} : \texttt{dict}$$

$$(\textbf{numpy} : \text{nested } \texttt{list} = \textbf{pandas} : \text{nested } \texttt{dict})$$

Climbing up dimensions

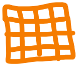
1		
		```
		In :
		np.array(1, ndmin=1)
		Out:
		array([1])
		```
			```
			In :
			pd.Series([1])
			Out:
			0      1
			dtype: int64
			```

Climbing up dimensions

1		`In : np.array(1, ndmin=1) Out: array([1])`	`In : pd.Series([1]) Out: 0    1 dtype: int64`
2		`In : np.array(1, ndmin=2) Out: array([[1]])`	`In : pd.DataFrame([[1]]) Out:    0 0  1`

Climbing up dimensions

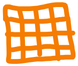
1		`In  :`
		`np.array(1, ndmin=1)`
		`Out:`
		`array([1])`
			`In  :`
			`pd.Series([1])`
			`Out:`
			`0    1`
			`dtype: int64`

2		`In  :`
		`np.array(1, ndmin=2)`
		`Out:`
		`array([[1]])`
			`In  :`
			`pd.DataFrame([[1]])`
			`Out:`
			`   0`
			`0  1`

3		`In  :`
		`np.array(1, ndmin=3)`
		`Out:`
		`array([[[1]]])`
			`In  :`
			`pd.Panel([[[1]]])`
			`Out:`
			`FutureWarning: Panel is deprecated[...]`

Climbing up dimensions

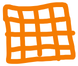
1		In :
		np.array(1, ndmin=1)
		Out:
		array([1])
			In :
			pd.Series([1])
			Out:
			0 1
			dtype: int64

2		In :
		np.array(1, ndmin=2)
		Out:
		array([[1]])
			In :
			pd.DataFrame([[1]])
			Out:
			   0
			0 1

3		In :
		np.array(1, ndmin=3)
		Out:
		array([[[1]]])
			In :
			pd.Panel([[1]])
			Out:
			FutureWarning: Panel is deprecated [...]

4		In :
		np.array(1, ndmin=4)
		Out:
		array([[[[1]]]])
			In :
			pd.Panel4D([[1]])
			Out:
			AttributeError: [...]

Climbing up dimensions

1		`In : np.array(1, ndmin=1) Out: array([1])`	`In : pd.Series([1]) Out: 0    1 dtype: int64`
2		`In : np.array(1, ndmin=2) Out: array([[1]])`	`In : pd.DataFrame([[1]]) Out:    0 0  1`
3		`In : np.array(1, ndmin=3) Out: array([[[1]]])`	`In : pd.Panel([[1]]) Out: FutureWarning: Panel is deprecated [...]`
4		`In : np.array(1, ndmin=4) Out: array([[[[1]]]])`	`In : pd.Panel4D([[1]]) Out: AttributeError: [...]`
...	...	...	...
N		`In : np.array(1, ndmin=n) Out: array([...[1]...])`	?!

Are two dimensions enough?

Are two dimensions enough?

No

So what, are **pandas** devs crazy?!

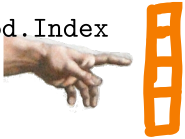
So what, are **pandas** devs crazy?!

No

The structure of **pandas** data structures

The structure of **pandas** data structures

`pd.Index`

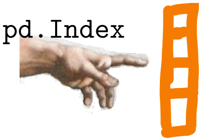


`pd.Series`

`s.loc["a"]`

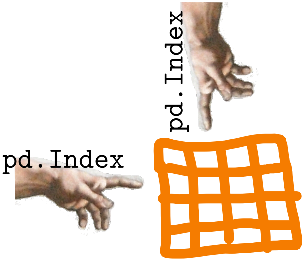
The structure of **pandas** data structures

pd.Series



s.loc["a"]

pd.DataFrame



df.loc["a", "b"]

→1.1 pandas for dummies

There are many types of indexes

- `pd.Int64Index`, `pd.RangeIndex`, `pd.FloatIndex`

There are many types of indexes

- `pd.Int64Index`, `pd.RangeIndex`, `pd.FloatIndex`

- `pd.DatetimeIndex`, `pd.PeriodIndex`

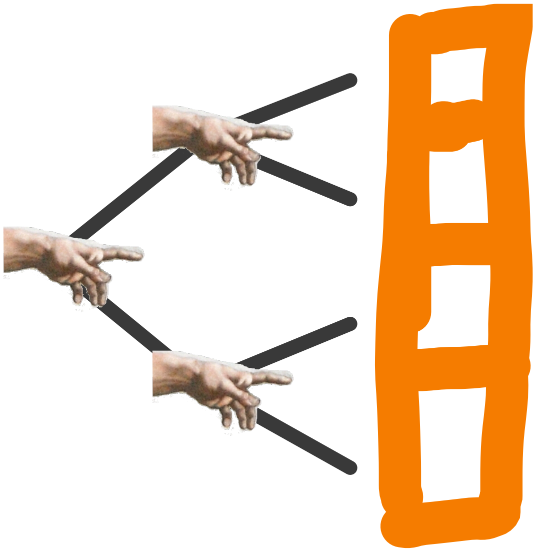
There are many types of indexes

- `pd.Int64Index`, `pd.RangeIndex`, `pd.FloatIndex`
- `pd.DatetimeIndex`, `pd.PeriodIndex`
- `pd.IntervalIndex` ...

There are many types of indexes

- `pd.Int64Index`, `pd.RangeIndex`, `pd.FloatIndex`
- `pd.DatetimeIndex`, `pd.PeriodIndex`
- `pd.IntervalIndex` . . .
- The star tonight: `pd.MultiIndex`.

What a `MultiIndex` looks like



→1.1 Basic example

What do we gain/loose

Pros

- simpler implementation

What do we gain/loose

Pros

- simpler implementation
- extreme flexibility

What do we gain/loose

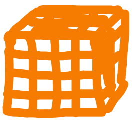
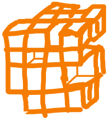
Pros

- simpler implementation
- extreme flexibility
- intuitive operations (later)

What do we gain/loose

Pros

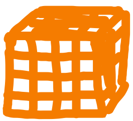
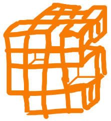
- simpler implementation
- extreme flexibility
- intuitive operations (later)
- efficient use of space for unbalanced data


vs.

$\rightarrow$`1.2 - Unbalanced data`

What do we gain/loose

Pros

- simpler implementation
- extreme flexibility
- intuitive operations (later)
- efficient use of space for unbalanced data


vs.


$\rightarrow$`1.2 - Unbalanced data`

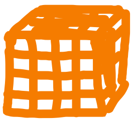
Cons

- comparatively inefficient for balanced data

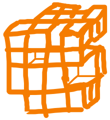
What do we gain/loose

Pros

- simpler implementation
- extreme flexibility
- intuitive operations (later)
- efficient use of space for unbalanced data



vs.



→`1.2 - Unbalanced data`

Cons

- comparatively inefficient for balanced data
- more complex semantics for `DataFrame`

→`1.2 - Mi DF`

But then, isn't `DataFrame` superfluous?!

No

But then, isn't `DataFrame` superfluous?!

No

- Remember: we don't want to feel inferior to "**R**ists"

But then, isn't `DataFrame` superfluous?!

No

- Remember: we don't want to feel inferior to "**R**ists"
- More importantly, people are just *too used* to having data in tables

But then, isn't `DataFrame` superfluous?!

No

- Remember: we don't want to feel inferior to "**R**ists"
- More importantly, people are just *too used* to having data in tables
- Even more importantly, we want to switch data between columns and index level

$\rightarrow$`1.2 - Unbalanced data`

But then, isn't `DataFrame` superfluous?!

No

- Remember: we don't want to feel inferior to "**R**ists"
- More importantly, people are just *too used* to having data in tables
- Even more importantly, we want to switch data between columns and index level

  $\rightarrow$`1.2 - Unbalanced data`
- Most importantly, we want to "split" dimensions in two groups when doing operations

  $\rightarrow$`1.2 - Reshape`

If you *really* need *n*-dimensional, indexed structures

```
In [4]: xr.DataArray(np.random.randn(2, 3))
Out[4]:
<xarray.DataArray (dim_0: 2, dim_1: 3)>
array([[ 1.643563, -1.469388,  0.357021],
       [-0.6746  , -1.776904, -0.968914]])
Dimensions without coordinates: dim_0, dim_1

In [5]: data = xr.DataArray(np.random.randn(2, 3), coords={'x': ['a', 'b']}, dims=('x',
'y'))

In [6]: data
Out[6]:
<xarray.DataArray (x: 2, y: 3)>
array([[-1.294524,  0.413738,  0.276662],
       [-0.472035, -0.01396 , -0.362543]])
Coordinates:
  * x        (x) <U1 'a' 'b'
Dimensions without coordinates: y
```

“groupby” is lavels aware!

```
df.groupby(level=...)
```

→1.2 Groupby

Thanks

to

- you, for your patience

Thanks

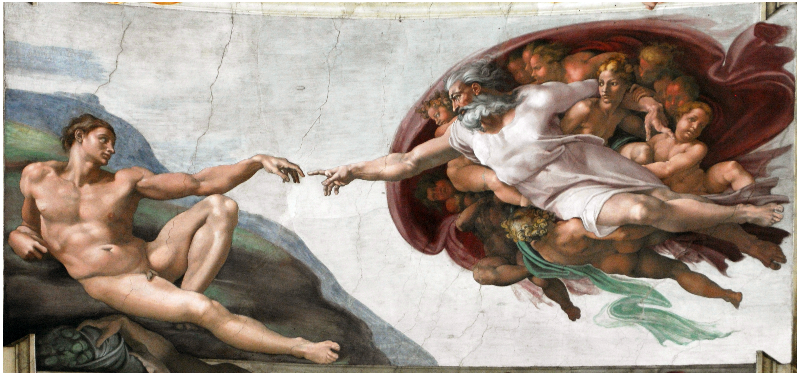
to

- you, for your patience
- the organizers, for their faith in my succinctness

Thanks

to

- you, for your patience
- the organizers, for their faith in my succinctness
- Michelangelo, for his pioneering use of indexes



To contact me: me@pietrobattiston.it