Processing tables with Python

Marcelo Leomil Zoccoler and Martin Schätz

With materials from

Till Korten, Robert Haase, PoL – TU Dresden

August 2023

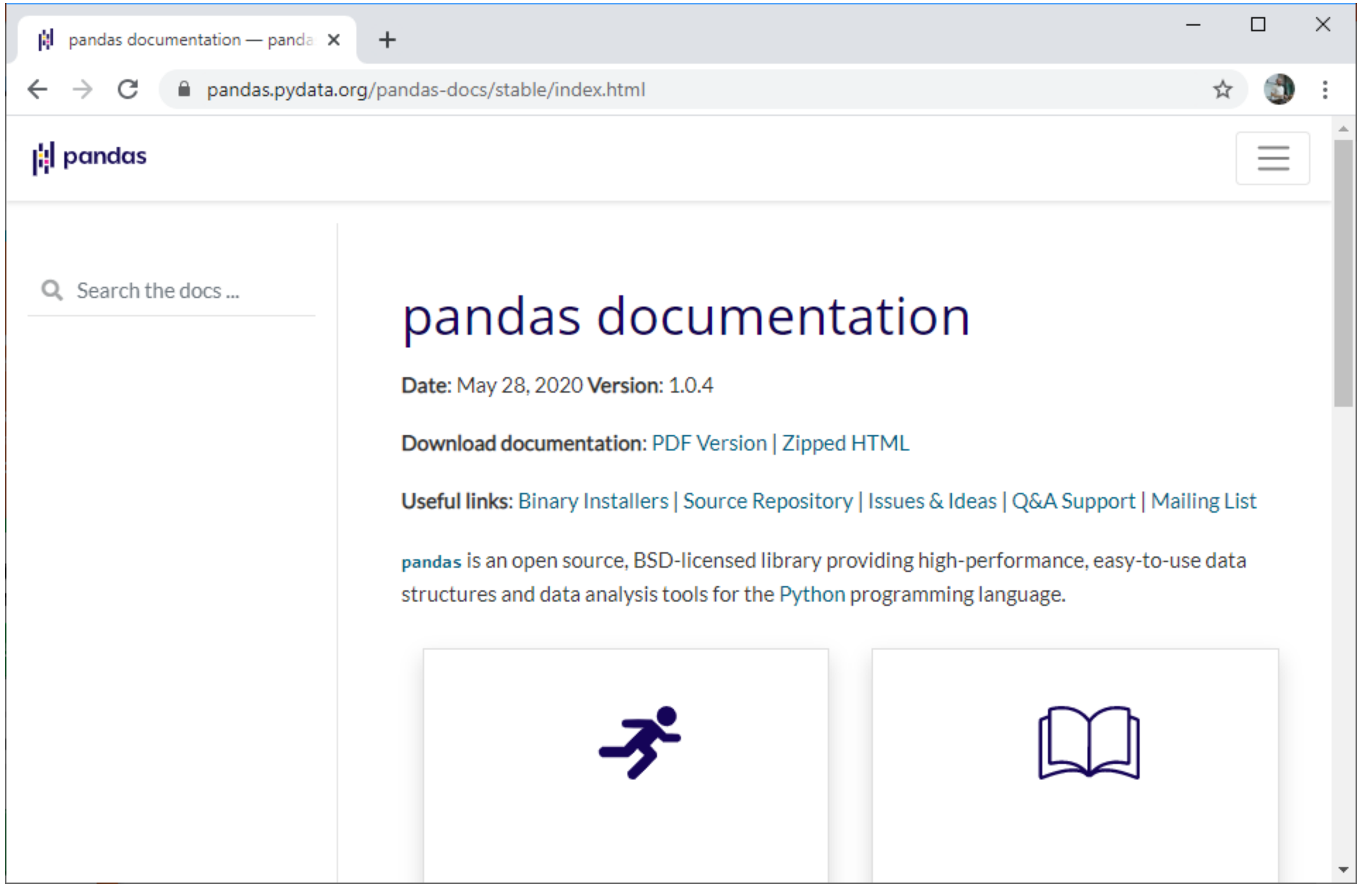
@zoccolermarcelo
@SchatzCz

Pandas is very useful for processing 2D tables

- Typical use-case:
  - Data from a colleague (i.e. an excel file)
  - Output from a software that was saved to disk (i.e. a csv file)

  - Use pandas

```
conda install pandas
```



Loading a pandas table from a csv file


```
import pandas as pd

df_csv = pd.read_csv('../../data/blobs_statistics.csv')
df_csv
```

		Area	Mean	Circ.	AR	Round	Solidity
0	1	2610	96.920	0.773	1.289	0.776	1.0
1	2	2100	90.114	0.660	2.333	0.429	1.0
2	3	27	110.222	0.108	27.000	0.037	1.0

Display just the first 3 rows of a table:

```
df_csv.head(3)
```

Display just the last 3 rows of a table:

```
df_csv.tail(3)
```

Creating pandas tables from Python data

- from a nupy array

```python
import numpy as np

data = np.random.random((4,3))
column_header = ['area',
'minor_axis', 'major_axis']

pd.DataFrame(data,
columns=column_header)
```

	area	minor_axis	major_axis
0	0.425681	0.135821	0.017084
1	0.036739	0.120840	0.925127
2	0.506095	0.453657	0.690560
3	0.748323	0.174359	0.603710

- from a dictionary

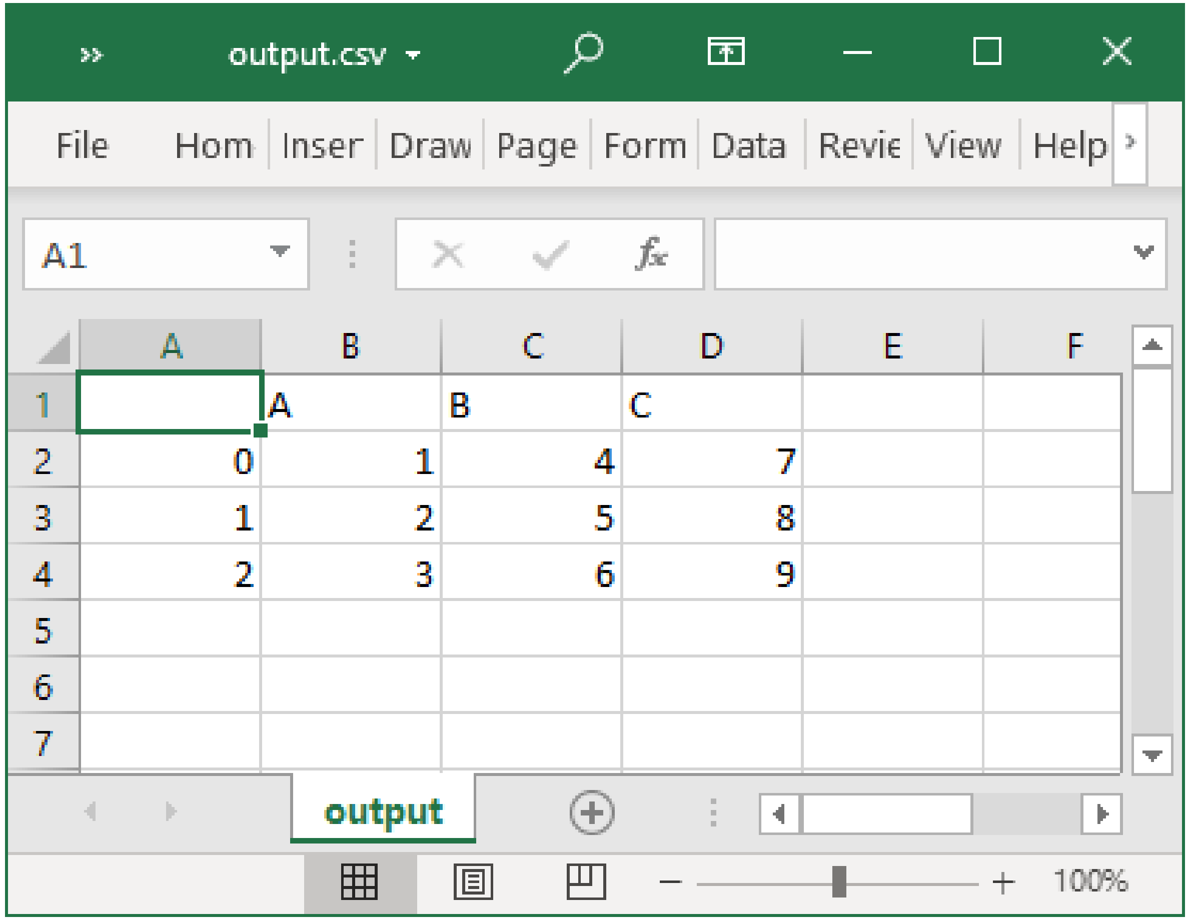
```python
measurements = {
"labels": [1, 2, 3],
"area": [45, 23, 68],
"minor_axis": [2, 4, 4],
"major_axis": [3, 4, 5],
}

pd.DataFrame(measurements)
```

	labels	area	minor_axis	major_axis
0	1	45	2	3
1	2	23	4	4
2	3	68	4	5

Saving pandas tables to disk

```python
df.to_csv(”output.csv")
```



	A	B	C	D	E	F
1		A	B	C		
2	0	1	4	7		
3	1	2	5	8		
4	2	3	6	9		
5						
6						
7						

Select a table column similar to an element of a `dict`

CzechBIAS

```
cities['City']
```

	City	Country	Population	Area_km2
0	Tokyo	Japan	13515271	2191
1	Delhi	India	16753235	1484
2	Shanghai	China	24183000	6341
3	Sao Paulo	Brazil	12252023	1521
4	Mexico City	Mexico	9209944	1485

	City
0	Tokyo
1	Delhi
2	Shanghai
3	Sao Paulo
4	Mexico City

Select multiple columns with a list of column names

CzechBIAS

```
cities[ ['City', 'Country'] ]
```

Note the double brackets

	City	Country	Population	Area_km2
0	Tokyo	Japan	13515271	2191
1	Delhi	India	16753235	1484
2	Shanghai	China	24183000	6341
3	Sao Paulo	Brazil	12252023	1521
4	Mexico City	Mexico	9209944	1485

	City	Country
0	Tokyo	Japan
1	Delhi	India
2	Shanghai	China
3	Sao Paulo	Brazil
4	Mexico City	Mexico

Select table rows through the `loc` object

CzechBIAS

```
data_frame.loc[ 0, ['City', 'Country']]
```

	City	Country	Population	Area_km2
0	Tokyo	Japan	13515271	2191
1	Delhi	India	16753235	1484
2	Shanghai	China	24183000	6341
3	Sao Paulo	Brazil	12252023	1521
4	Mexico City	Mexico	9209944	1485

	City	Country
0	Tokyo	Japan

Select individual cells

CzechBIAS

	City	Country	Population	Area_km2
0	Tokyo	Japan	13515271	2191
1	Delhi	India	16753235	1484
2	Shanghai	China	24183000	6341
3	Sao Paulo	Brazil	12252023	1521
4	Mexico City	Mexico	9209944	1485

```
data_frame[‘City'][0]
```

```
‘Tokyo’
```

Selecting rows that fulfill criteria

PoL
Physics of Life
TU Dresden
CzechBIAS

- Select cities with an area of more than 2000 km^2

	City	Country	Population	Area_km2
0	Tokyo	Japan	13515271	2191
1	Delhi	India	16753235	1484
2	Shanghai	China	24183000	6341
3	Sao Paulo	Brazil	12252023	1521
4	Mexico City	Mexico	9209944	1485

```
cities["area"] > 2000
```

```
0     True
1    False
2     True
3    False
4    False
Name: Area_km2, dtype: bool
```

```
cities[ cities["area"] > 2000 ]
```

	City	Country	Population	Area_km2
0	Tokyo	Japan	13515271	2191
2	Shanghai	China	24183000	6341

@zoccolermarcelo
@SchatzCz

Combining similar tables

CzechBIAS

- If tables have the same columns

```python
pd.concat([countries1, countries2])
```

countries1

	Country	Population
0	Japan	127202192
1	India	1352642280
2	China	1427647786

+

countries2

	Country	Population
0	Brazil	209489323
1	Mexico	126190788

=

	Country	Population
0	Japan	127202192
1	India	1352642280
2	China	1427647786
0	Brazil	209489323
1	Mexico	126190788

@zoccolermarcelo
@SchatzCz

Keep information about the data source

- Add a column to each table before concatenating them

```
countries1['Survey ID'] = 26
```

	Country	Population	Survey ID
0	Japan	127202192	26
1	India	1352642280	26
2	China	1427647786	26

```
countries2['Survey ID'] = 73
```

	Country	Population	Survey ID
0	Brazil	209489323	73
1	Mexico	126190788	73

```
pd.concat([countries1, countries2])
```

	Country	Population	Survey ID
0	Japan	127202192	26
1	India	1352642280	26
2	China	1427647786	26
0	Brazil	209489323	73
1	Mexico	126190788	73

Handling NaN values

CzechBIAS

- Usually indicate missing data
- Can cause errors when handling the data

- The easiest is to drop them using the “.dropna” method
- Drops any row containing a NaN value

```
data_no_nan = data.dropna(how="any")
```

Work with tidy-data when processing tables

|CzechBIAS

- Each variable is a column.
- Each observation is a row.
- Each type of observation has its own separate data frame.

```
data_frame.melt()
```

Not tidy:

	Before		After	
	channel_1	channel_2	channel_1	channel_2
0	13.250000	21.000000	15.137984	42.022776
1	44.954545	24.318182	43.328836	48.661610
2	13.590909	18.772727	11.685995	37.926184
3	85.032258	19.741935	86.031461	40.396353

Tidy:

	variable_0	variable_1	value
0	Before	channel_1	13.250000
1	Before	channel_1	44.954545
2	Before	channel_1	13.590909
3	Before	channel_1	85.032258
4	Before	channel_1	10.731707
...	...	...	...
99	After	channel_2	73.286439
100	After	channel_2	145.900739