PoL
Physics of Life
TU Dresden

CENTER FOR
SYSTEMS BIOLOGY
DRESDEN

Processing tables with Python

Till Korten

With materials from

Marcelo L. Zoccoler, Robert Haase, PoL – TU Dresden

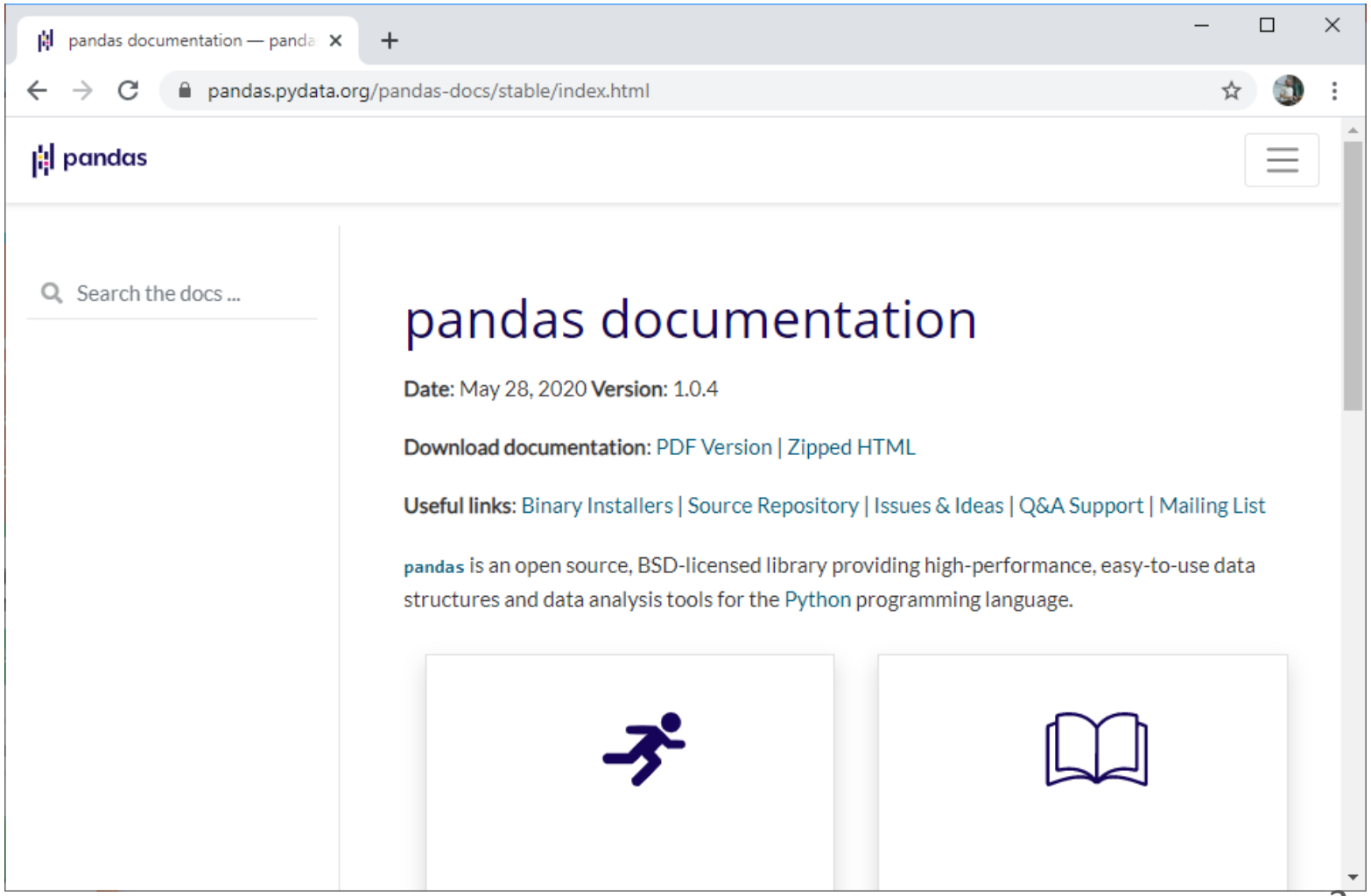
December 2022

@TillKorten

Pandas

- pandas is a library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

```
conda install pandas
```



Pandas is very useful for processing 2D tables

PoL Physics of Life TU Dresden

- Typical use-case:
  - You get data from a colleague in form of a table
  - You get a table as output of a function and save it to disk
  - Using pandas, you can analyze it in python.
- Loading a table in python using pandas:

```
import pandas as pd

data_frame = pd.read_csv("Measurements_ImageJ.csv", delimiter=',')
data_frame
```

		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 5 first rows of a table:

```
data_frame.head(5)
```

Display just the 5 last rows of a table:

```
data_frame.tail(5)
```

Creating pandas tables from Python data

PoL Physics of Life TU Dresden

- from a list of lists

```
row_header = ['labels', 'area', 'minor_axis', 'major_axis']

data = [
    [1, 2, 3],      # first row
    [45, 23, 68],   # second row
    [2, 4, 4],      # third row
    [3, 4, 5],      # fourth row
]

# convert the data and header arrays in a pandas data frame
data_frame = pd.DataFrame(data, row_header)

# show it
data_frame
```

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

- from a nupy array

```
import numpy as np

data = np.random.random((4,3))

pd.DataFrame(data, row_header)
```

✓ 0.2s

	0	1	2
labels	0.564022	0.271416	0.539888
area	0.686078	0.836249	0.784269
minor_axis	0.907028	0.773920	0.097580
major_axis	0.186222	0.228141	0.289165

- from a dictionary

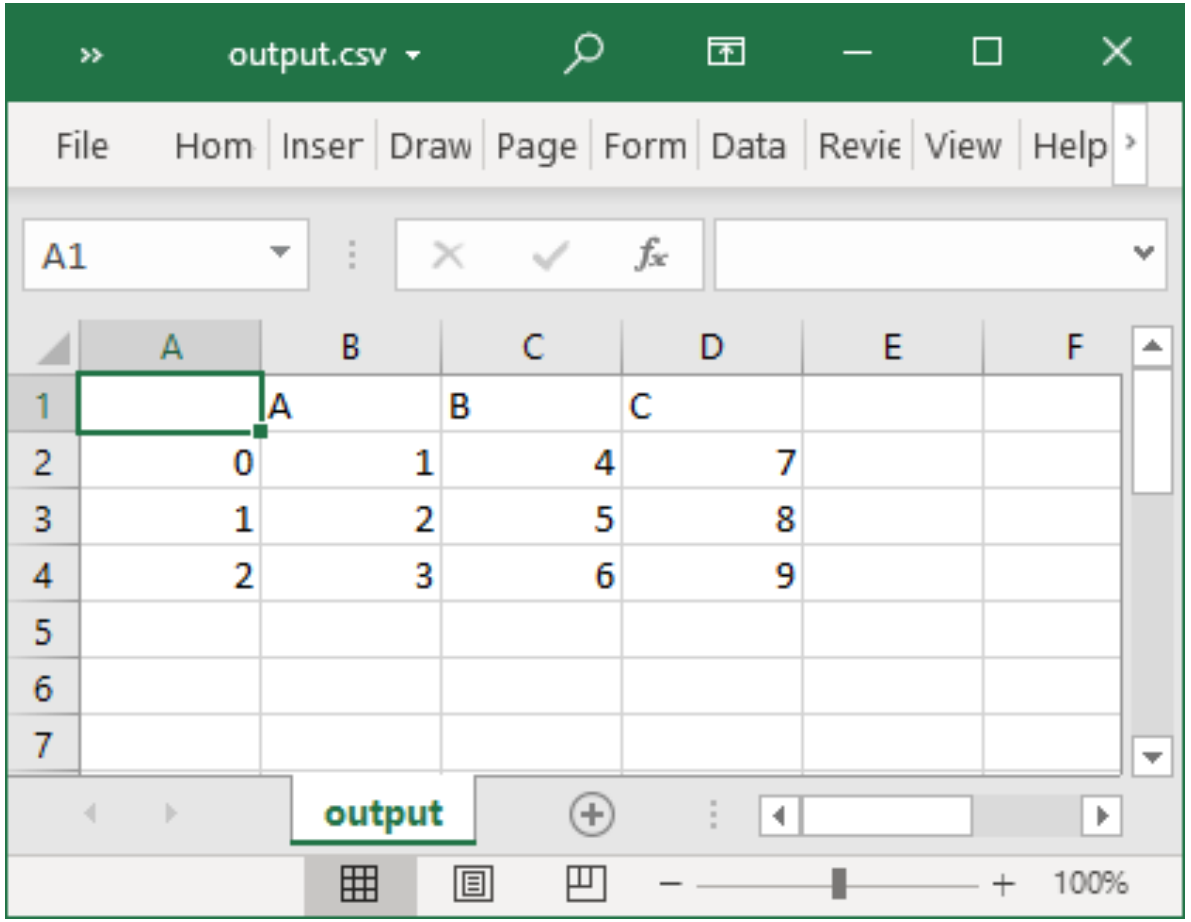
```
measurements = {
    "labels":      [1, 2, 3],
    "area":        [45, 23, 68],
    "minor_axis": [2, 4, 4],
    "major_axis": [3, 4, 5],
}

pd.DataFrame(measurements)
```

✓ 0.2s

	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

```
# save a dataframe to a CSV
data_frame.to_csv("output.csv")
```



Selecting pandas table columns

		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

- How to select the column “Area”?

```
data_frame['Area']
```

```
data_frame[:][0]
```

```
data_frame.iloc[:,1]
```

```
data_frame.loc[:,1]
```

Selecting multiple columns

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

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

Accessing pandas table rows

		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

- How to select the first row?

```
data_frame[0,:]
```

```
data_frame[:][0]
```

```
data_frame.iloc[0,:]
```

```
data_frame.loc[0,:]
```

Selecting individual cells in pandas tables

		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

```
data_frame["Mean"][0]
```

```
data_frame.loc[0, "Mean"]
```

```
data_frame.iloc[0, 2]
```

Selecting individual cells in pandas tables

		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

```
data_frame["Mean"][0]
```

```
1.2890000000000001
```

```
data_frame.loc[0, "Mean"]
```

```
data_frame.iloc[0, 2]
```

- Accessing a column

```
data_frame["Mean"]
```

```
0     96.920
1     90.114
2    110.222
Name: Mean, dtype: float64
```

- Determining mean of a column

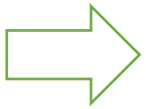
```
import numpy as np
np.mean(data_frame["Mean"])
```

```
99.08533333333332
```

Selecting rows that fulfill criteria

	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

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



```
cities['Area_km2'] > 2000
```

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

```
cities[cities['Area_km2'] > 2000]
```

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

Combining similar tables

- The big art in data science is the ability of combining information from multiple sources to gain new insights.
- If tables have the same columns

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

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

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

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

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

```
countries1
```

	Country	Population
0	Japan	127202192
1	India	1352642280
2	China	1427647786

```
countries2
```

	Country	Population
0	Brazil	209489323
1	Mexico	126190788

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

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

Combining different tables

- The big art in data science is the ability of combining information from multiple sources to gain new insights.

	Country	Population
0	Japan	127202192
1	India	1352642280
2	China	1427647786
3	Brazil	209469323
4	Mexico	126190788

	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

```
combined = countries.merge(cities, on='Country', suffixes=['_country', '_city'])
combined
```

	Country	Population_country	City	Population_city	Area_km2
0	Japan	127202192	Tokyo	13515271	2191
1	India	1352642280	Delhi	16753235	1484
2	China	1427647786	Shanghai	24183000	6341
3	Brazil	209469323	Sao Paulo	12252023	1521
4	Mexico	126190788	Mexico City	9209944	1485

```
# compute ratio
combined['City_Country_population_ratio'] = combined['Population_city'] / combined['Population_country']

# only show selected columns
combined[['City', 'City_Country_population_ratio']]
```

	City	City_Country_population_ratio
0	Tokyo	0.106250
1	Delhi	0.012386
2	Shanghai	0.016939
3	Sao Paulo	0.058491
4	Mexico City	0.072984

Handling NaNs

- Sometimes tables may contains NaNs (Not a Number). These values may come from missing experimental data or from missing data when merging tables.

- They can introduce errors to calculations with tables.

- The easiest way to drop them is to use the ".dropna" method. This will drop any rows that contain NaN.

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

- But be careful, do not drop NaNs carelessly. Try to investigate first why they are there. Also, you may accidentally discard useful data from other columns.

Always work with tidy-data

- Tidy data frames follow the rules:
  - Each variable is a column.
  - Each observation is a row.
  - Each type of observation has its own separate data frame.

```
df['intensity_mean'] > 200
```

Which of these data is 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

```
df['intensity_mean'] > 200
```

	time	label	intensity_mean	area
0	0	1	233.5	20
1	0	2	403.0	40
2	0	3	255.3	30
3	1	1	244.5	20
4	1	2	402.0	40
5	1	3	256.7	30
6	2	1	278.9	20
7	2	2	401.2	40
8	2	3	255.1	30

Tidy-Data

- Tidy data frames follow the rules:
  - Each variable is a column.
  - Each observation is a row.
  - Each type of observation has its own separate data frame.

Using pd.melt may help tidying data

	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

```
df.melt()
```

	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

Common workflow: Split-Apply-Combine

	area	intensity_mean	major_axis_length	minor_axis_length	aspect_ratio	file_name	round
0	139	96.546763	17.504104	10.292770	1.700621	20P1_POS0010_D_1UL	False
1	360	86.613889	35.746808	14.983124	2.385805	20P1_POS0010_D_1UL	False
2	43	91.488372	12.967884	4.351573	2.980045	20P1_POS0010_D_1UL	False
3	140	73.742857	18.940508	10.314404	1.836316	20P1_POS0010_D_1UL	False
4	144	89.375000	13.639308	13.458532	1.013432	20P1_POS0010_D_1UL	True

- compute the median "intensity_mean“
- of round objects

```
grouped = df.groupby('round')
```

Apply (calculate median):

```
df_median = grouped.median()
```

```
df_median.reset_index()
```

	round	area	intensity_mean	major_axis_length	minor_axis_length	aspect_ratio
0	False	270.0	92.788345	21.459495	15.858324	1.412849
1	True	291.0	100.256000	20.155547	18.352287	1.101700