Wizualizacja danych - wykład 6

dr Piotr Jastrzębski

Biblioteka Pandas

Bibliotea Pandas - cd.

Import:

```
import numpy as np
import pandas as pd
```

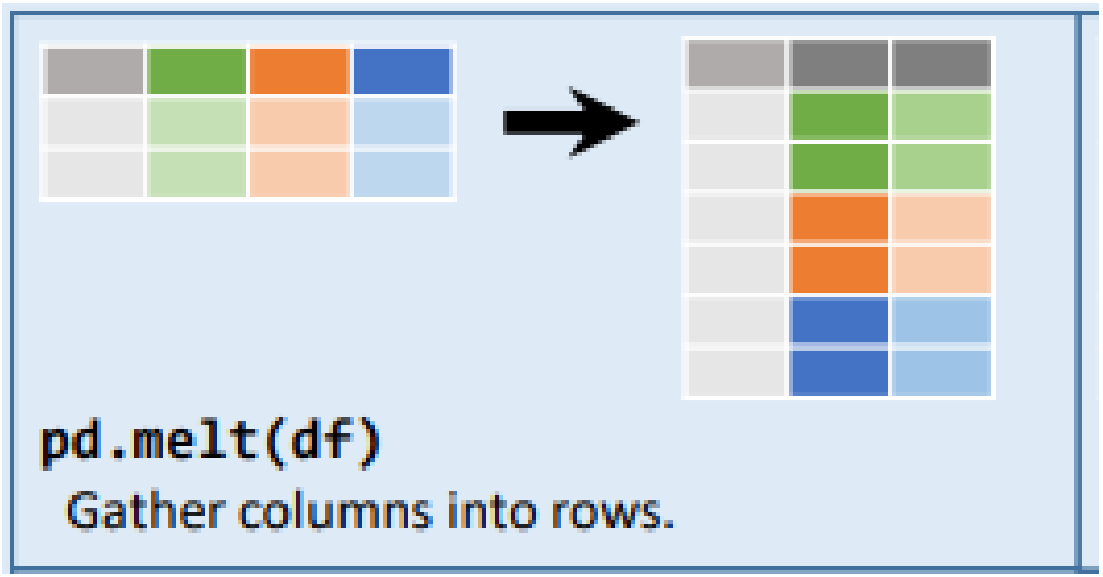
Obsługa plików csv

Funkcja `pandas.read_csv`

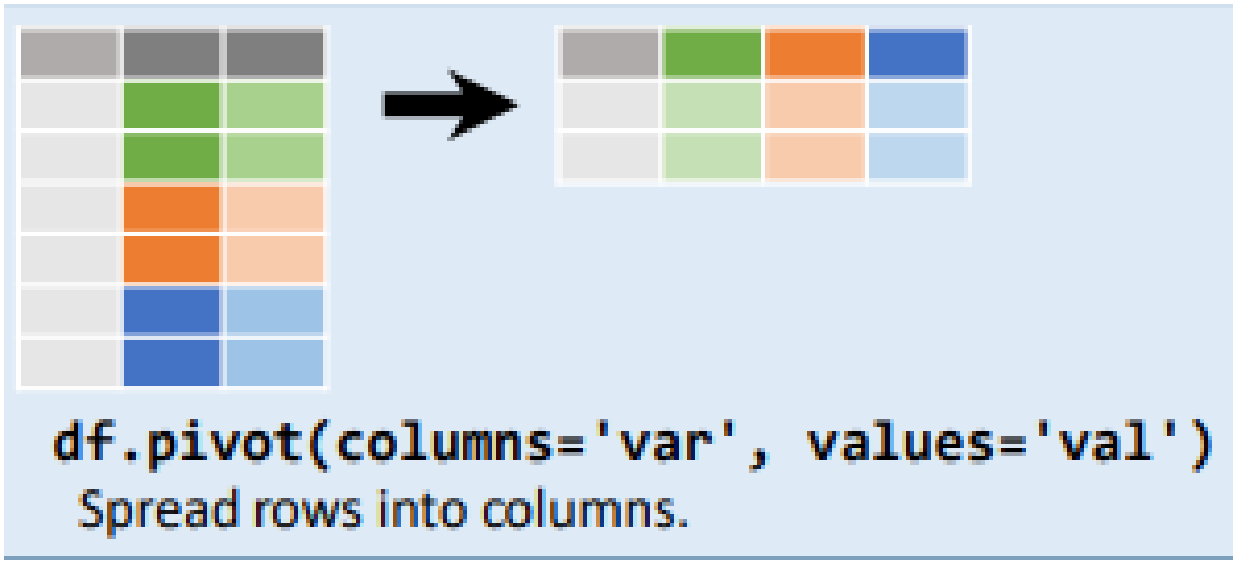
Dokumentacja: link

Zapis `pandas.DataFrame.to_csv`

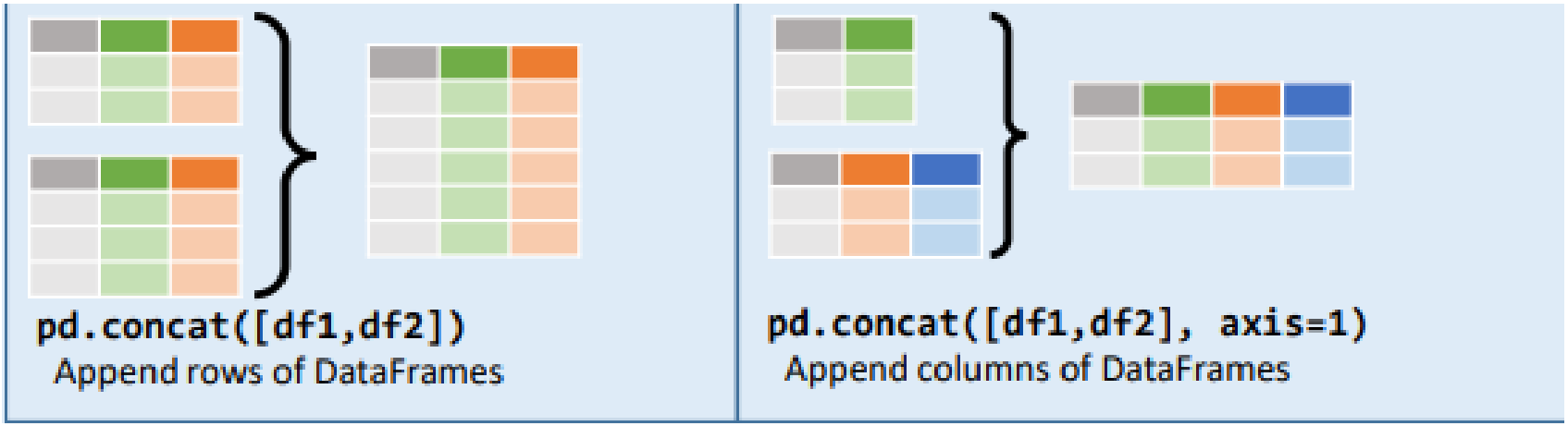
Dokumentacja: link



Rysunek 1



Rysunek 2



Rysunek 3

https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html

"Czyszczenie danych"

"Tidy data"

Imię	Wiek	Wzrost	Kolor oczu
Adam	26	167	Brązowe
Sylwia	34	164	Piwne
Tomasz	42	183	Niebieskie

- jedna obserwacja (jednostka statystyczna) = jeden wiersz w tabeli/macierzy/ramce danych
- wartosci danej cechy znajduja sie w kolumnach
- jeden typ/rodzaj obserwacji w jednej tabeli/macierzy/ramce danych

Alternatywne koncepcje

- 'as is'
- messy data

Obsługa brakującacyh danych

```
string_data = pd.Series(['aardvark', 'artichoke', np.nan,
print(string_data)
```

```
## 0     aardvark
## 1    artichoke
## 2          NaN
## 3      avocado
## dtype: object
```

```
print(string_data.isnull())
```

```
## 0    False
## 1    False
## 2     True
## 3    False
## dtype: bool
```

```
print(string_data.dropna())
```

```
## 0     aardvark
## 1    artichoke
## 3      avocado
## dtype: object
```

```
from numpy import nan as NA
data = pd.DataFrame([[1., 6.5, 3.], [1., NA, NA],
                     [NA, NA, NA], [NA, 6.5, 3.]])
cleaned = data.dropna()
print(cleaned)
```

```
##      0    1    2
## 0  1.0  6.5  3.0
```

```
print(data.dropna(how='all'))
```

```
##      0    1    2
## 0  1.0  6.5  3.0
## 1  1.0  NaN  NaN
## 3  NaN  6.5  3.0
```

```
data[4] = NA
print(data.dropna(how='all', axis=1))
```

```
##      0    1    2
## 0  1.0  6.5  3.0
## 1  1.0  NaN  NaN
## 2  NaN  NaN  NaN
## 3  NaN  6.5  3.0
```

Uzupełnienie braków

```
print(data)
```

```
##      0    1    2   4
## 0  1.0  6.5  3.0 NaN
## 1  1.0  NaN  NaN NaN
## 2  NaN  NaN  NaN NaN
## 3  NaN  6.5  3.0 NaN
```

```
print(data.fillna(0))
```

```
##      0    1    2    4
## 0  1.0  6.5  3.0  0.0
## 1  1.0  0.0  0.0  0.0
## 2  0.0  0.0  0.0  0.0
## 3  0.0  6.5  3.0  0.0
```

```
print(data.fillna({1: 0.5, 2: 0}))
```

```
##      0    1    2   4
## 0  1.0  6.5  3.0 NaN
## 1  1.0  0.5  0.0 NaN
## 2  NaN  0.5  0.0 NaN
## 3  NaN  6.5  3.0 NaN
```

Usuwanie duplikatów

```
data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],
                     'k2': [1, 1, 2, 3, 3, 4, 4]})
print(data)
```

```
##     k1  k2
## 0  one   1
## 1  two   1
## 2  one   2
## 3  two   3
## 4  one   3
## 5  two   4
## 6  two   4
```

```
print(data.duplicated())
```

```
## 0    False
## 1    False
## 2    False
## 3    False
## 4    False
## 5    False
## 6     True
## dtype: bool
```

```
print(data.drop_duplicates())
```

```
##     k1  k2
## 0  one   1
## 1  two   1
## 2  one   2
## 3  two   3
## 4  one   3
## 5  two   4
```

Zastępowanie wartościami

```
data = pd.Series([1., -999., 2., -999., -1000., 3.])
print(data)
```

```
## 0       1.0
## 1    -999.0
## 2       2.0
## 3    -999.0
## 4   -1000.0
## 5       3.0
## dtype: float64
```

```
print(data.replace(-999, np.nan))
```

```
## 0       1.0
## 1       NaN
## 2       2.0
## 3       NaN
## 4   -1000.0
## 5       3.0
## dtype: float64
```

```
print(data.replace([-999, -1000], np.nan))
```

```
## 0    1.0
## 1    NaN
## 2    2.0
## 3    NaN
## 4    NaN
## 5    3.0
## dtype: float64
```

```
print(data.replace([-999, -1000], [np.nan, 0]))
```

```
## 0    1.0
## 1    NaN
## 2    2.0
## 3    NaN
## 4    0.0
## 5    3.0
## dtype: float64
```

```
print(data.replace({-999: np.nan, -1000: 0}))
```

```
## 0    1.0
## 1    NaN
## 2    2.0
## 3    NaN
## 4    0.0
## 5    3.0
## dtype: float64
```

Dyskretyzacja i podział na koszyki

```
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]
cats = pd.cut(ages, bins)
print(cats)
```

```
## [(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ...,
## Length: 12
## Categories (4, interval[int64]): [(18, 25] < (25, 35] <
```

```
print(cats.codes)
```

```
## [0 0 0 1 0 0 2 1 3 2 2 1]
```

```
print(cats.categories)
```

```
## IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]],
##               closed='right',
##               dtype='interval[int64]')
```

```
print(pd.value_counts(cats))
```

```
## (18, 25]     5
## (35, 60]     3
## (25, 35]     3
## (60, 100]    1
## dtype: int64
```

```
cats2 = pd.cut(ages, [18, 26, 36, 61, 100], right=False)
print(cats2)
```

```
## [[18, 26), [18, 26), [18, 26), [26, 36), [18, 26), ...,
## Length: 12
## Categories (4, interval[int64]): [[18, 26) < [26, 36) <
```

```
group_names = ['Youth', 'YoungAdult', 'MiddleAged', 'Senio
print(pd.cut(ages, bins, labels=group_names))
```

```
## [Youth, Youth, Youth, YoungAdult, Youth, ..., YoungAdul
## Length: 12
## Categories (4, object): [Youth < YoungAdult < MiddleAge
```

```
data = np.random.rand(20)
print(pd.cut(data, 4, precision=2))
```

```
## [(0.64, 0.84], (0.43, 0.64], (0.43, 0.64], (0.43, 0.64],
## Length: 20
## Categories (4, interval[float64]): [(0.018, 0.22] < (0.2
```

```
data = np.random.randn(1000)
cats = pd.qcut(data, 4)
print(cats)
```

```
## [(-0.0735, 0.668], (-0.714, -0.0735], (-0.714, -0.0735],
## Length: 1000
## Categories (4, interval[float64]): [(-3.153, -0.714] < (
##                                     (0.668, 2.941]]
```

```
print(pd.value_counts(cats))
```

```
## (0.668, 2.941]       250
## (-0.0735, 0.668]     250
## (-0.714, -0.0735]    250
## (-3.153, -0.714]     250
## dtype: int64
```

Wykrywanie i filtrowanie elementów odstających

```
data = pd.DataFrame(np.random.randn(1000, 4))
print(data.describe())
```

```
##                 0            1            2
## count  1000.000000  1000.000000  1000.000000  1000.0000
## mean      0.028726    -0.003608    -0.020062     0.0134
## std       1.009187     0.952352     0.984592     0.9967
## min      -3.738678    -3.568974    -3.937396    -3.3289
## 25%      -0.636437    -0.649406    -0.649706    -0.6895
## 50%       0.021729     0.033320    -0.007595     0.0124
## 75%       0.721714     0.619660     0.665374     0.6960
## max       3.327107     3.541407     3.125360     3.3238
```

```
col = data[2]
print(col[np.abs(col) > 3])
```

```
## 106    3.125360
## 736   -3.553834
## 992   -3.937396
## Name: 2, dtype: float64
```

```
print(data[(np.abs(data) > 3).any(1)])
```

```
##              0         1         2         3
## 2    -0.409243 -0.849343  0.219955 -3.328998
## 106  -1.008608 -0.942840  3.125360  1.149931
## 156   0.094160  3.061381  0.439956  0.886473
## 304   0.784909 -3.145077 -0.769122 -0.105634
## 364  -3.738678  0.006943 -0.381620 -0.192870
## 394   3.327107 -0.421259 -1.353155  0.017185
## 401  -3.264390  1.000468  0.840316 -0.031463
## 416   0.957949 -3.568974 -0.026893  2.205284
## 417  -0.881073  3.541407 -0.886750 -0.771266
## 436   3.271522 -1.519330 -0.168125 -1.254539
## 449   3.103106 -0.625239  1.442857  0.289026
## 736   0.770504 -1.413431 -3.553834  0.260638
## 760  -3.103907 -0.147791 -0.116939  0.487697
## 779   0.453248 -0.162306 -1.213866  3.323834
```

Bibliografia

- https://s3.amazonaws.com/assets.datacamp.com/blog_assets/PandasPythonForDataScience.pdf, dostęp online 5.4.2019.
- https://www.marsja.se/pandas-read-csv-tutorial-to-csv/, dostęp online 20.04.2019.
- https://www.geeksforgeeks.org/python-pandas-melt/
- https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf