Wizualizacja Danych - Biblioteka Pandas

Ostatnia aktualizacja: 2022-04-23 17:03:47

Pandas

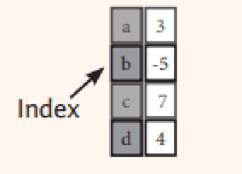
Pandas jest biblioteką Pythona służącą do analizy i manipulowania danymi

Import:

```
import pandas as pd
```

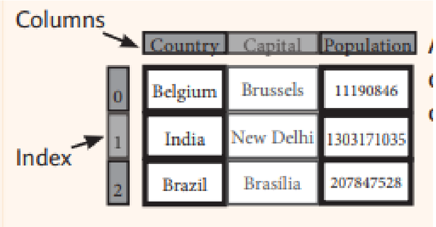
Podstawowe byty

Seria - Series



a	3
b	-5
c	7
d	4

Ramka danych - DataFrame



	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847528

```
import pandas as pd
import numpy as np

s = pd.Series([3, -5, 7, 4])
print(s)
print(s.values)
print(type(s.values))
t = np.sort(s.values)
print(t)
print(s.index)
print(type(s.index))
```

```
## 0    3
## 1   -5
## 2    7
## 3    4
## dtype: int64
## [ 3 -5  7  4]
## <class 'numpy.ndarray'>
## [-5  3  4  7]
## RangeIndex(start=0, stop=4, step=1)
## <class 'pandas.core.indexes.range.RangeIndex'>
```

```
import pandas as pd
import numpy as np

s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
print(s)
print(s['b'])
s['b'] = 8
print(s)
print(s[s > 5])
print(s * 2)
print(np.sin(s))
```

```
## a    3
## b   -5
## c    7
## d    4
## dtype: int64
## -5
## a    3
## b    8
## c    7
## d    4
## dtype: int64
## b    8
## c    7
## dtype: int64
## a     6
## b    16
## c    14
## d     8
## dtype: int64
## a    0.141120
## b    0.989358
## c    0.656987
## d   -0.756802
## dtype: float64
```

```
import pandas as pd

d = {'key1': 350, 'key2': 700, 'key3': 70}
s = pd.Series(d)
print(s)
```

```
## key1    350
## key2    700
## key3     70
## dtype: int64
```

```
import pandas as pd

d = {'key1': 350, 'key2': 700, 'key3': 70}
k = ['key0', 'key2', 'key3', 'key1']
s = pd.Series(d, index=k)
print(s)
pd.isnull(s)
pd.notnull(s)
s.isnull()
s.notnull()
s.name = "Wartosc"
s.index.name = "Klucz"
print(s)
```

```
## key0      NaN
## key2    700.0
## key3     70.0
## key1    350.0
## dtype: float64
## key0     True
## key2    False
## key3    False
## key1    False
## dtype: bool
## key0    False
## key2     True
## key3     True
## key1     True
## dtype: bool
## key0     True
## key2    False
## key3    False
## key1    False
## dtype: bool
## key0    False
## key2     True
## key3     True
## key1     True
## dtype: bool
## Klucz
## key0      NaN
## key2    700.0
## key3     70.0
## key1    350.0
## Name: Wartosc, dtype: float64
```

```
import pandas as pd

data = {'Country': ['Belgium', 'India', 'Brazil'],
        'Capital': ['Brussels', 'New Delhi', 'Brasília'],
        'Population': [11190846, 1303171035, 207847528]}
frame = pd.DataFrame(data)
print(frame)
df = pd.DataFrame(data, columns=['Country', 'Capital',
                                 'Population'])
print(df)
print(df.iloc[[0], [0]])
print(df.loc[[0], ['Country']])
print(df.loc[2])
print(df.loc[:, 'Capital'])
print(df.loc[1, 'Capital'])
print(df[df['Population'] > 1200000000])
print(df.drop('Country', axis=1))
print(df.shape)
print(df.index)
print(df.columns)
print(df.info())
print(df.count())
```

```
##    Country    Capital  Population
## 0  Belgium   Brussels    11190846
## 1    India  New Delhi  1303171035
## 2   Brazil   Brasília   207847528
##    Country    Capital  Population
## 0  Belgium   Brussels    11190846
## 1    India  New Delhi  1303171035
## 2   Brazil   Brasília   207847528
##    Country
## 0  Belgium
##    Country
## 0  Belgium
## Country          Brazil
## Capital        Brasília
## Population    207847528
## Name: 2, dtype: object
## 0     Brussels
## 1    New Delhi
## 2     Brasília
## Name: Capital, dtype: object
## New Delhi
##   Country    Capital  Population
## 1   India  New Delhi  1303171035
##     Capital  Population
## 0   Brussels    11190846
## 1  New Delhi  1303171035
## 2   Brasília   207847528
## (3, 3)
## RangeIndex(start=0, stop=3, step=1)
## Index(['Country', 'Capital', 'Population'], dtype='object')
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 3 entries, 0 to 2
## Data columns (total 3 columns):
##  #   Column      Non-Null Count  Dtype 
## ---  ------      --------------  ----- 
##  0   Country     3 non-null      object
##  1   Capital     3 non-null      object
##  2   Population  3 non-null      int64 
## dtypes: int64(1), object(2)
## memory usage: 200.0+ bytes
## None
## Country       3
## Capital       3
## Population    3
## dtype: int64
```

Uzupełnianie braków

```
import pandas as pd

s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
s2 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
print(s + s2)
print(s.add(s2, fill_value=0))
print(s.mul(s2, fill_value=2))
```

```
## a    10.0
## b     NaN
## c     5.0
## d     7.0
## dtype: float64
## a    10.0
## b    -5.0
## c     5.0
## d     7.0
## dtype: float64
## a    21.0
## b   -10.0
## c   -14.0
## d    12.0
## dtype: float64
```

Obsługa plików csv

Funkcja `pandas.read_csv`

Dokumentacja: link

Zapis `pandas.DataFrame.to_csv`

Dokumentacja: link

Obsługa plików z Excela

Funkcja `pandas.read_excel`

https://pandas.pydata.org/docs/reference/api/pandas.read_excel.html

** Ważne: trzeba zainstalować bibliotekę `openpyxl` do importu .xlsx oraz `xlrd` do importu .xls (nie trzeba ich importować w kodzie jawnie w większości wypadków)

Operacje manipulacyjne

- `merge`

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html?highlight=merge#pandas.DataFrame.merge

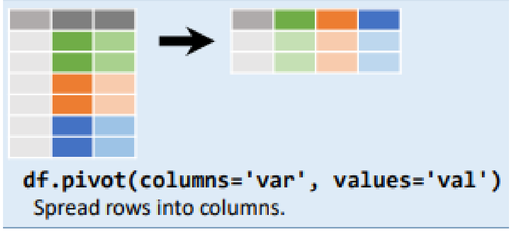
- `join`

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.join.html?highlight=join#pandas.DataFrame.join

- `concat`

https://pandas.pydata.org/docs/reference/api/pandas.concat.html?highlight=concat#pandas.concat

- `pivot`

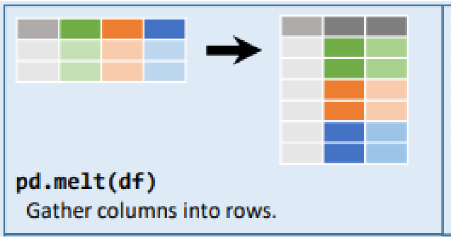


```
df.pivot(columns='var', values='val')
```

Spread rows into columns.

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.pivot.html?highlight=pivot#pandas.DataFrame.pivot

- `melt`



https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.melt.html?highlight=melt#pandas.DataFrame.melt

“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

Obsługa brakujących danych

```
import numpy as np
import pandas as pd

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

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

```
from numpy import nan as NA
import pandas as pd

data = pd.DataFrame([[1., 6.5, 3.], [1., NA, NA],
                     [NA, NA, NA], [NA, 6.5, 3.]])
cleaned = data.dropna()
print(cleaned)
print(data.dropna(how='all'))
data[4] = NA
print(data.dropna(how='all', axis=1))
print(data)
print(data.fillna(0))
print(data.fillna({1: 0.5, 2: 0}))
```

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

```
import pandas as pd

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

```
##     k1  k2
## 0  one   1
## 1  two   1
## 2  one   2
## 3  two   3
## 4  one   3
## 5  two   4
## 6  two   4
## 0     False
## 1     False
## 2     False
## 3     False
## 4     False
## 5     False
## 6      True
## dtype: bool
##     k1  k2
## 0  one   1
## 1  two   1
## 2  one   2
## 3  two   3
## 4  one   3
## 5  two   4
```

Zastępowanie wartościami

```
import pandas as pd
import numpy as np

data = pd.Series([1., -999., 2., -999., -1000., 3.])
print(data)
print(data.replace(-999, np.nan))
print(data.replace([-999, -1000], np.nan))
print(data.replace([-999, -1000], [np.nan, 0]))
print(data.replace({-999: np.nan, -1000: 0}))
```

```
## 0       1.0
## 1    -999.0
## 2       2.0
## 3    -999.0
## 4   -1000.0
## 5       3.0
## dtype: float64
## 0       1.0
## 1       NaN
## 2       2.0
## 3        NaN
## 4    -1000.0
## 5        3.0
## dtype: float64
## 0    1.0
## 1    NaN
## 2    2.0
## 3    NaN
## 4    NaN
## 5    3.0
## dtype: float64
## 0    1.0
## 1    NaN
## 2    2.0
## 3    NaN
## 4    0.0
## 5    3.0
## dtype: float64
## 0    1.0
## 1    NaN
## 2    2.0
## 3    NaN
## 4    0.0
## 5    3.0
## dtype: float64
```

Dyskretyzacja i podział na koszyki

```
import pandas as pd

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

```
## [(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100],
## Length: 12
## Categories (4, interval[int64, right]): [(18, 25] < (25, 35] < (35, 60] < (60
## [0 0 0 1 0 0 2 1 3 2 2 1]
## IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]], dtype='interval[int6
## (18, 25]     5
## (25, 35]     3
## (35, 60]     3
## (60, 100]    1
## dtype: int64
```

```
import pandas as pd

ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]
cats2 = pd.cut(ages, [18, 26, 36, 61, 100], right=False)
print(cats2)
group_names = ['Youth', 'YoungAdult',
               'MiddleAged', 'Senior']
print(pd.cut(ages, bins, labels=group_names))
```

```
## [[18, 26), [18, 26), [18, 26), [26, 36), [18, 26), ..., [26, 36), [61, 100),
## Length: 12
## Categories (4, interval[int64, left]): [[18, 26) < [26, 36) < [36, 61) < [61,

## ['Youth', 'Youth', 'Youth', 'YoungAdult', 'Youth', ..., 'YoungAdult', 'Senior
## Length: 12
## Categories (4, object): ['Youth' < 'YoungAdult' < 'MiddleAged' < 'Senior']
```

```
import pandas as pd
import numpy as np

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

```
## [(0.081, 0.29], (0.29, 0.5], (0.29, 0.5], (0.29, 0.5], (0.081, 0.29], ..., (0
## Length: 20
## Categories (4, interval[float64, right]): [(0.081, 0.29] < (0.29, 0.5] < (0.5
```

```
import pandas as pd
import numpy as np

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

```
## [(-0.606, -0.0364], (0.601, 2.906], (0.601, 2.906], (-0.606, -0.0364], (0.601
## Length: 1000
## Categories (4, interval[float64, right]): [(-3.3489999999999998, -0.606] < (-
##                                             (0.601, 2.906]]
## (-3.3489999999999998, -0.606]    250
## (-0.606, -0.0364]                250
## (-0.0364, 0.601]                 250
## (0.601, 2.906]                   250
## dtype: int64
```

Wykrywanie i filtrowanie elementów odstających

```
import pandas as pd
import numpy as np

data = pd.DataFrame(np.random.randn(1000, 4))
print(data.describe())
col = data[2]
print(col[np.abs(col) > 3])
print(data[(np.abs(data) > 3).any(1)])
```

```
##                  0            1            2            3
## count  1000.000000  1000.000000  1000.000000  1000.000000
## mean     -0.057109     0.012727     0.045559    -0.064776
## std       0.985574     1.006186     0.992972     1.008053
## min      -2.871763    -3.826926    -3.477464    -3.300333
## 25%      -0.728968    -0.661943    -0.597260    -0.727630
## 50%      -0.054641     0.040948     0.054566    -0.045455
## 75%       0.617440     0.633324     0.706807     0.587376
## max       3.007700     2.994164     3.011782     2.917422
## 127    3.011782
## 229   -3.368881
## 963   -3.218372
## 997   -3.477464
## Name: 2, dtype: float64
##             0         1         2         3
## 127 -0.175062  1.782662  3.011782 -2.371048
## 229 -0.253300 -0.120071 -3.368881 -1.770955
## 502  3.006265 -0.696815  0.010320  0.239408
## 669  0.816250 -3.826926 -0.177378 -0.525864
## 811  1.048825  0.556837  1.602003 -3.300333
## 877  3.007700  0.343596 -1.207129  0.541541
## 963  1.575457 -0.156850 -3.218372  0.252540
## 997  0.384228 -0.597243 -3.477464 -0.037356
```

Bibliografia:

- Dokumentacja biblioteki, https://pandas.pydata.org/, dostęp online 5.03.2021.
- Hannah Stepanek, Thinking in Pandas, How to Use the Python Data Analysis Library the Right Way, Apress, 2020.