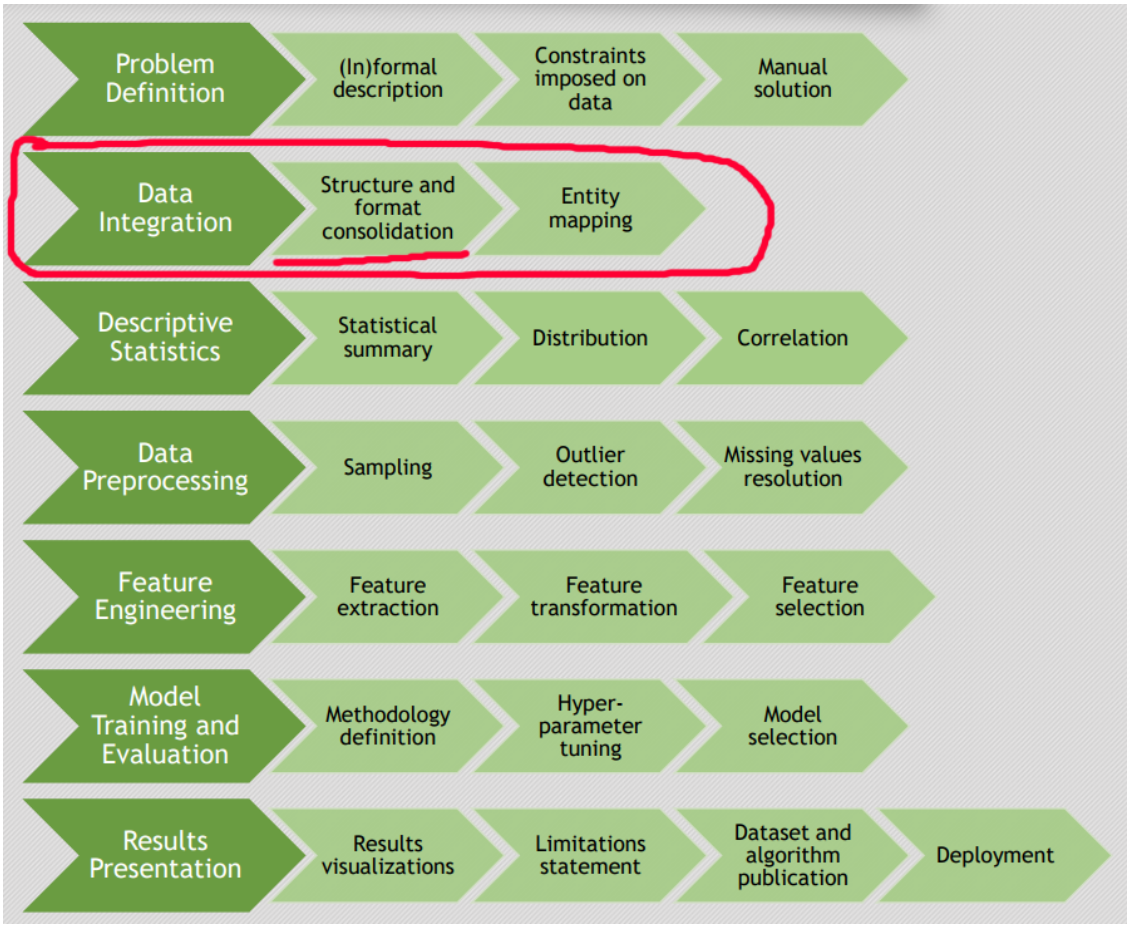
Integracia_dat

October 30, 2017

1 Integracia dat

```
In [1]: from IPython.display import Image
        Image('ML_Workflow.PNG')
```

Out[1]:



2 O com nejdem hovorit

- Nejdem opisovat vsetky mozne record linkage a entity mapping metody (to je minimalne na samostatnu prednasku)
- Nejdem opisovat komplexne ETL nastroje a postupy na spajanie tabuliek a roznych databaz (na to tu mame dokonca samostatny predmet)

3 Obsah dnesnej prezentacie

- Intro do pouzivania kniznic Pandas, Matplotlib a Numpy
- Ako pouzit tieto kniznice na zakladne upravovanie formy dat (data cleaning, reshaping, wrangling)
- Velmi lahke zaklady explorativnej analyzy a prace s chybajucimi hodnotami

```
In [2]: %matplotlib inline
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn
```

3.0.1 Na co nam je Pandas?

- importovanie dat zo standardnych formatov
- vycistit
- pozriet sa do dat (statistiky, sampling, zakladne grafy)
- posunut data na analyzu / trenovanie modelov

3.0.2 Co je Pandas?

- Python komunita sa inspirovala a ukradla to dobre z `data.frame` struktury v R alebo obdobnych struktur v Matlabe alebo Octave
- Umoznuje zakladne operacie s datami, sampling, group by, merge, ...
- Ako zaklad je pouzite NumPy pole

3.0.3 Zakladne ulohy

- Spracovanie chybajucich udajov (.dropna(), pd.isnull())
- Merge, join (concat, join)
- Group
- Zmena tvaru dat (pivotovanie) (stack, pivot)
- Praca s casovymi radmi (resampling, timezones, ..)
- Kreslenie

3.1 Nieco k Numpy

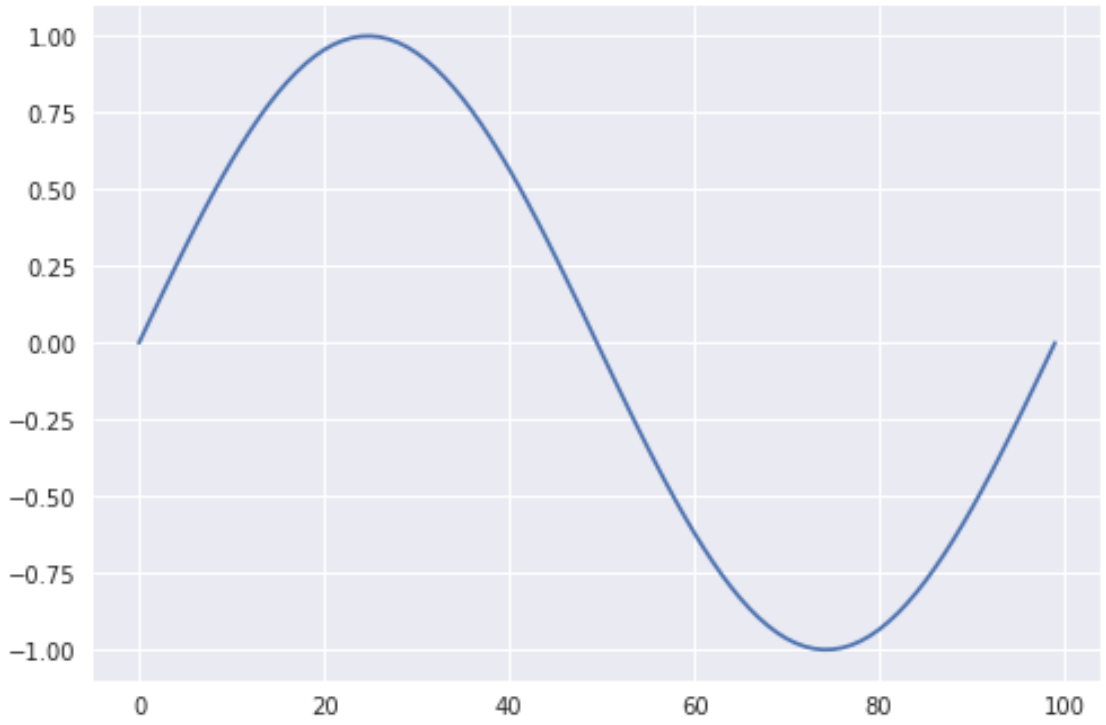
```
In [3]: pole = [1,2,3]
        pole * 3

Out[3]: [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
In [4]: np_pole = np.array([1,2,3])
        np_pole * 3

Out[4]: array([3, 6, 9])

In [12]: x = np.arange(20).reshape(4, 5) # skusit viacere dimenzie
         x

Out[12]: array([[ 0,  1,  2,  3,  4],
                [ 5,  6,  7,  8,  9],
                [10, 11, 12, 13, 14],
                [15, 16, 17, 18, 19]])

In [13]: x.shape

Out[13]: (4, 5)

In [14]: x.ndim

Out[14]: 2

In [17]: x.sum(axis=0)

Out[17]: array([30, 34, 38, 42, 46])
```

3.2 Viacero typov cisel

```
In [18]: x.dtype

Out[18]: dtype('int64')

In [19]: a = np.array([.1,.2])
         print(a)
         a.dtype

[ 0.1  0.2]


Out[19]: dtype('float64')

In [20]: c = np.array( [ [1,2], [3,4] ], dtype=complex )
         print(c)
         c.dtype

[[ 1.+0.j  2.+0.j]
 [ 3.+0.j  4.+0.j]]


Out[20]: dtype('complex128')

In [21]: np.zeros((3,4))
```

```
Out[21]: array([[ 0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.]])

In [22]: np.ones((2,5))

Out[22]: array([[ 1.,  1.,  1.,  1.,  1.],
                [ 1.,  1.,  1.,  1.,  1.]])

In [23]: np.repeat(3, 10).reshape([2,5])

Out[23]: array([[3, 3, 3, 3, 3],
                [3, 3, 3, 3, 3]])

In [24]: np.linspace(0, 2, 9)

Out[24]: array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,  2.  ])

In [25]: x = np.linspace( 0, 2*np.pi, 100 )
         f = np.sin(x)

In [26]: plt.plot(f)

Out[26]: [<matplotlib.lines.Line2D at 0x7f9dc50ede80>]

/usr/local/lib/python3.5/dist-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: F
  (prop.get_family(), self.defaultFamily[fontext]))
```



3.3 Maticove operacie

```
In [27]: A = np.array( [[1,1], [0,1]] )
         B = np.array( [[2,0], [3,4]] )

In [28]: A

Out[28]: array([[1, 1],
                [0, 1]])

In [29]: B

Out[29]: array([[2, 0],
                [3, 4]])

In [30]: np.transpose(B)

Out[30]: array([[2, 3],
                [0, 4]])

In [31]: A*B

Out[31]: array([[2, 0],
                [0, 4]])

In [32]: A.dot(B) # np.dot(A, B)

Out[32]: array([[5, 4],
                [3, 4]])
```

3.4 Vyberanie prvkov

```
In [33]: a = np.arange(10)**3
         a

Out[33]: array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])

In [34]: a[2]

Out[34]: 8

In [35]: a[2:5]

Out[35]: array([ 8, 27, 64])

In [36]: a[2:6:2]

Out[36]: array([ 8, 64])

In [37]: a[:6:2] = -1000
         a

Out[37]: array([-1000,     1, -1000,    27, -1000,   125,   216,   343,   512,   729])

In [38]: a[ : :-1]

Out[38]: array([  729,   512,   343,   216,   125, -1000,    27, -1000,     1, -1000])
```

3.5 Vyberanie prvkov z viacrozmerneho pola

```
In [39]: b = np.arange(20).reshape(4,5)
         b

Out[39]: array([[ 0,  1,  2,  3,  4],
                [ 5,  6,  7,  8,  9],
                [10, 11, 12, 13, 14],
                [15, 16, 17, 18, 19]])

In [40]: b[2,3]

Out[40]: 13

In [41]: b[2,]

Out[41]: array([10, 11, 12, 13, 14])

In [42]: b[1:3,2:4]

Out[42]: array([[ 7,  8],
                [12, 13]])

In [43]: b[:,2:4]

Out[43]: array([[ 2,  3],
                [ 7,  8],
                [12, 13],
                [17, 18]])
```

Dalsie operacie si pozrite * tu https://docs.scipy.org/doc/numpy-dev/user/quickstart.html * a tu https://docs.scipy.org/doc/numpy-dev/reference/index.html

3.6 Nejake ukazky k Pandas

Pandas pouziva Numpy pole a nad nim si postavili typ `Series` a `DataFrame`

```
In [44]: s = pd.Series([0,1,2,3,4])
         s

Out[44]: 0    0
         1    1
         2    2
         3    3
         4    4
         dtype: int64

In [45]: # k numpy polu je pridany explicitny index
         s.index

Out[45]: RangeIndex(start=0, stop=5, step=1)
```

```
In [46]: s.values

Out[46]: array([0, 1, 2, 3, 4])

In [47]: s[0]

Out[47]: 0

In [48]: # na rozdiel od numpy vsak index moze byt aj nieco ine ako cislo
         s2 = pd.Series(np.arange(4), index=['a', 'b', 'c', 'd'])
         s2

Out[48]: a    0
         b    1
         c    2
         d    3
         dtype: int64

In [49]: s2['c']

Out[49]: 2

In [50]: s2[2]

Out[50]: 2

In [51]: s2.c

Out[51]: 2

In [52]: # na vytvorenie Series objektu sa da pouzit aj asociatyvne pole
         population = pd.Series({'Germany': 81.3, 'Belgium': 11.3, 'France': 64.3, 'United Kingd
         population

Out[52]: Belgium           11.3
         France            64.3
         Germany           81.3
         Netherlands       16.9
         United Kingdom    64.9
         dtype: float64

In [53]: population['France']

Out[53]: 64.299999999999997

In [54]: # kedze je to postavene na Numpy, tak vieme robit vsetky zaujimave operacie
         population * 1000

Out[54]: Belgium           11300.0
         France            64300.0
         Germany           81300.0
         Netherlands       16900.0
         United Kingdom    64900.0
         dtype: float64
```

```
In [55]: # index ma implicitne dane poradie, takze sa da robit rozsah
         population['Belgium':'Netherlands']

Out[55]: Belgium        11.3
         France         64.3
         Germany        81.3
         Netherlands    16.9
         dtype: float64

In [56]: population.mean()

Out[56]: 47.739999999999995
```

Da sa pristupovat k prvkom tak, ako sme na to zvyknuti z R

```
In [57]: population[['France', 'Netherlands']]

Out[57]: France         64.3
         Netherlands    16.9
         dtype: float64

In [ ]: population[population > 20]
```

No a `DataFrame` je vlastne multidimenzionalny `Series`

```
In [67]: data = {'country': ['Belgium', 'France', 'Germany', 'Netherlands', 'United Kingdom'],
                 'population': [11.3, 64.3, 81.3, 16.9, 64.9],
                 'area': [30510, 671308, 357050, 41526, 244820],
                 'capital': ['Brussels', 'Paris', 'Berlin', 'Amsterdam', 'London']}
         countries = pd.DataFrame(data)
         countries

Out[67]:      area    capital         country  population
         0   30510   Brussels         Belgium        11.3
         1  671308      Paris          France        64.3
         2  357050     Berlin         Germany        81.3
         3   41526  Amsterdam     Netherlands        16.9
         4  244820     London  United Kingdom        64.9

In [68]: countries.index

Out[68]: RangeIndex(start=0, stop=5, step=1)

In [69]: countries.columns

Out[69]: Index(['area', 'capital', 'country', 'population'], dtype='object')

In [70]: countries.values
```

```
Out[70]: array([[30510, 'Brussels', 'Belgium', 11.3],
                [671308, 'Paris', 'France', 64.3],
                [357050, 'Berlin', 'Germany', 81.3],
                [41526, 'Amsterdam', 'Netherlands', 16.9],
                [244820, 'London', 'United Kingdom', 64.9]], dtype=object)

In [71]: countries.dtypes

Out[71]: area            int64
         capital        object
         country        object
         population    float64
         dtype: object

In [72]: countries.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
area          5 non-null int64
capital       5 non-null object
country       5 non-null object
population    5 non-null float64
dtypes: float64(1), int64(1), object(2)
memory usage: 240.0+ bytes


In [73]: countries.describe()

Out[73]:                 area  population
         count       5.000000    5.000000
         mean   269042.800000   47.740000
         std    264012.827994   31.519645
         min     30510.000000   11.300000
         25%     41526.000000   16.900000
         50%    244820.000000   64.300000
         75%    357050.000000   64.900000
         max    671308.000000   81.300000

In [74]: countries = countries.set_index('country')
         countries

Out[74]:                   area    capital  population
         country
         Belgium          30510   Brussels        11.3
         France          671308      Paris        64.3
         Germany         357050     Berlin        81.3
         Netherlands      41526  Amsterdam        16.9
         United Kingdom  244820     London        64.9
```

a vieme teraz velmi jednoducho pristupovat k jednotlivym stlpcom

```
In [75]: countries.area # countries['area']
```

```
Out[75]: country
         Belgium           30510
         France           671308
         Germany          357050
         Netherlands       41526
         United Kingdom   244820
         Name: area, dtype: int64
```

```
In [76]: countries['population']*1000000 / countries['area'] # hustota zaludnenia
```

```
Out[76]: country
         Belgium          370.370370
         France            95.783158
         Germany          227.699202
         Netherlands      406.973944
         United Kingdom   265.092721
         dtype: float64
```

```
In [77]: # vieme si jednoducho vyrobit novy stlpec
         countries['density'] = countries['population']*1000000 / countries['area']
         countries
```

```
Out[77]:                  area    capital  population     density
         country
         Belgium          30510   Brussels        11.3  370.370370
         France          671308      Paris        64.3   95.783158
         Germany         357050     Berlin        81.3  227.699202
         Netherlands      41526  Amsterdam        16.9  406.973944
         United Kingdom  244820     London        64.9  265.092721
```

```
In [78]: # a na zaklade neho napriklad vyberat riadky
         countries[countries['density'] > 300]
```

```
Out[78]:              area    capital  population     density
         country
         Belgium      30510   Brussels        11.3  370.370370
         Netherlands  41526  Amsterdam        16.9  406.973944
```

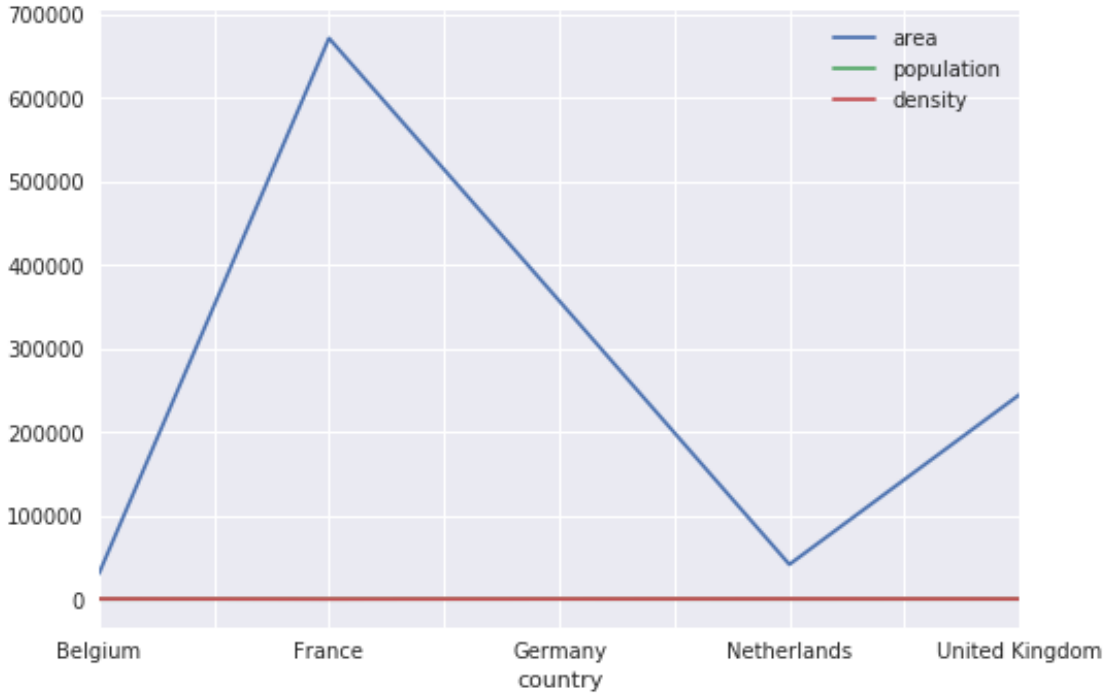
```
In [79]: # vieme potom napriklad usporiadavat
         countries.sort_values(by='density', ascending=False)
```

```
Out[79]:                  area    capital  population     density
         country
         Netherlands      41526  Amsterdam        16.9  406.973944
         Belgium          30510   Brussels        11.3  370.370370
         United Kingdom  244820     London        64.9  265.092721
         Germany         357050     Berlin        81.3  227.699202
         France          671308      Paris        64.3   95.783158
```

In [82]:
```
# velmi silna vlastnost je priamociare vykreslovanie
# countries.density.plot()
# countries.density.plot(kind='bar')
countries.plot()
```

Out[82]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9dc4f57c88>

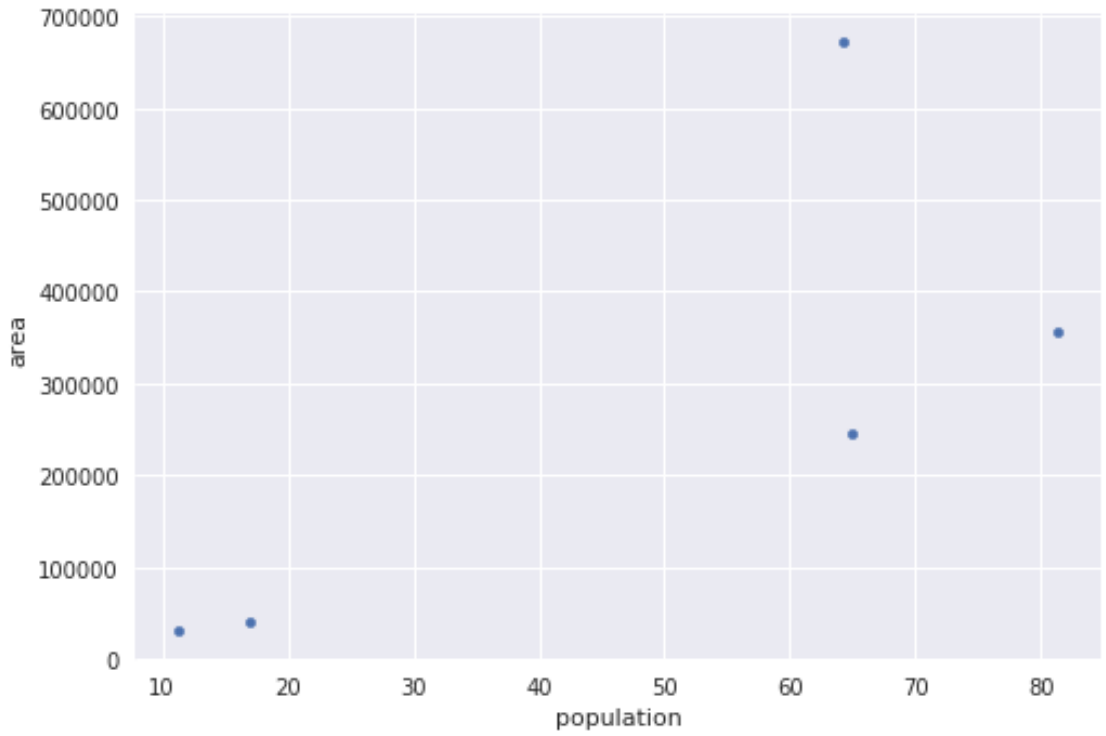
```
/usr/local/lib/python3.5/dist-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: F
  (prop.get_family(), self.defaultFamily[fontext]))
```



In [83]:
```
countries.plot(kind='scatter', x='population', y='area')
```

Out[83]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9dc4ed6550>

```
/usr/local/lib/python3.5/dist-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: F
  (prop.get_family(), self.defaultFamily[fontext]))
```



Kedze nam v `DataFrame` pribudla moznost vyberat stlpce podla nazvu, tak sa nam trochu skomplikovalo vyberanie prvkov oproti Numpy. Musime rozoznavat * vyberanie podla nazvu a * podla pozicie.

```
In [84]: countries['area']

Out[84]: country
         Belgium            30510
         France            671308
         Germany           357050
         Netherlands        41526
         United Kingdom    244820
         Name: area, dtype: int64

In [85]: countries[['area', 'density']]

Out[85]:                    area     density
         country
         Belgium           30510  370.370370
         France           671308   95.783158
         Germany          357050  227.699202
         Netherlands       41526  406.973944
         United Kingdom   244820  265.092721

In [86]: # ked ale chceme rozsah, tak nam to pristupuje k riadkom
         countries['France':'Netherlands']
```

```
Out[86]:                area    capital  population     density
         country
         France       671308      Paris        64.3   95.783158
         Germany      357050     Berlin        81.3  227.699202
         Netherlands   41526  Amsterdam        16.9  406.973944
```

Na pokrocilejsie vyberanie z tabulky pouzivame: * `loc` a * `iloc`

```
In [87]: # pristup ku konkretnej bunke pomocou riadka a stlpca
         countries.loc['Germany', 'area']

Out[87]: 357050

In [88]: # tu sa daju pouzit aj rozsahy na oboch rozmeroch
         countries.loc['France':'Germany', :]

Out[88]:            area capital  population     density
         country
         France   671308   Paris        64.3   95.783158
         Germany  357050  Berlin        81.3  227.699202

In [89]: # ale aj vymenovanie
         countries.loc[countries['density']>300, ['capital', 'population']]

Out[89]:                capital  population
         country
         Belgium       Brussels        11.3
         Netherlands  Amsterdam        16.9

In [ ]: # iloc vybera podla poradia. Toto je podobne pristupovaniu k prvkom ako v Numpy
        countries.iloc[0:2,1:3]

In [90]: # samozrejem, ze sa stale daju priradovat hodnoty
         countries.loc['Belgium':'Germany', 'population'] = 10
         countries

Out[90]:                  area    capital  population     density
         country
         Belgium          30510   Brussels        10.0  370.370370
         France          671308      Paris        10.0   95.783158
         Germany         357050     Berlin        10.0  227.699202
         Netherlands      41526  Amsterdam        16.9  406.973944
         United Kingdom  244820     London        64.9  265.092721
```

3.7 Zmena tvaru dat pomocou Pandas

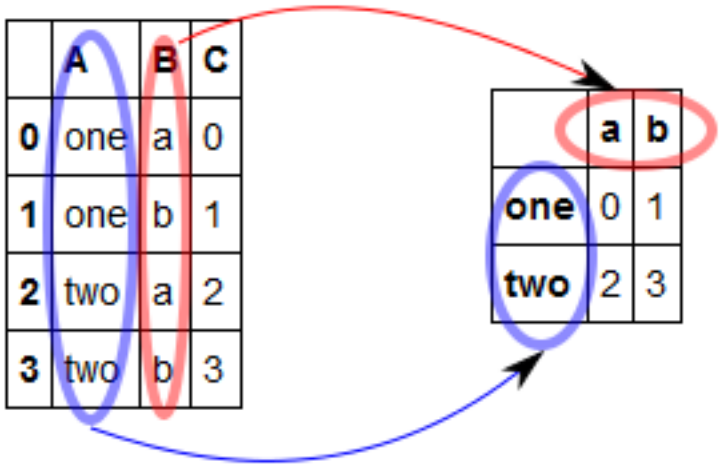
```
In [91]: df = pd.DataFrame({'A':['one', 'one', 'two', 'two'], 'B':['a', 'b', 'a', 'b'], 'C':rang
         # df = pd.DataFrame({'A':['one', 'one', 'two', 'two'], 'B':['a', 'b', 'a', 'b'], 'C':ra
         df
```

Out[91]:
```
     A  B  C
0  one  a  0
1  one  b  1
2  two  a  2
3  two  b  3
```

`unstack` presuva hodnoty v nejakom stlpci a vytvori z nich nazvy stlpcov
casto sa nam to hodi ak mame data, ktore su v trochu unej forme ako by sme potrebovali

In [92]:
```
Image("img/stack.png")
```

Out[92]:



In [93]:
```
df = df.set_index(['A', 'B']) # najskor si vyberieme stlpec, ktory pouzijeme ako index.
# Ten druhy bude dodavat hodnoty do nazvov novych stlpcov
df
```

Out[93]:
```
        C
A   B
one a   0
    b   1
two a   2
    b   3
```

In [94]:
```
# teraz si povieme v ktorom stlpci su hodnoty a nechame to preskupit
result = df['C'].unstack()
result
```

Out[94]:
```
B    a  b
A
one  0  1
two  2  3
```

```
In [95]: # opacna transformacia je stack. zoberie nazvy stlpcov a spravi z nich hodnoty
         df = result.stack().reset_index(name='C')
         df

Out[95]:      A  B  C
         0  one  a  0
         1  one  b  1
         2  two  a  2
         3  two  b  3

In [96]: # pivot je velmi podobny ako unstack, ale necha nastavit mena stlpcov a moze ich byt vi
         df = pd.DataFrame({'A':['one', 'one', 'two', 'two'], 'B':['a', 'b', 'a', 'b'], 'C':rang
         df

Out[96]:      A  B  C
         0  one  a  0
         1  one  b  1
         2  two  a  2
         3  two  b  3

In [97]: df.pivot(index='A', columns='B', values='C')

Out[97]: B    a  b
         A
         one  0  1
         two  2  3

In [98]: # pivot_table je podobne ako pivot, ale dokaze pracovat s duplicitnymi stlpcami a necha
         df = pd.DataFrame({'A':['one', 'one', 'two', 'two', 'one', 'two'], 'B':['a', 'b', 'a',
         df

Out[98]:      A  B  C
         0  one  a  0
         1  one  b  1
         2  two  a  2
         3  two  b  3
         4  one  a  4
         5  two  b  5

In [99]: df.pivot_table(index='A', columns='B', values='C', aggfunc=np.sum) #aggfunct je default

Out[99]: B    a  b
         A
         one  4  1
         two  2  8

In [ ]:
```

3.8 Ok, skusme sa konecne pohrat s nejakymi datami

In [100]:
```
data = pd.read_csv("data/BETR8010000800100hour.1-1-1990.31-12-2012", sep='\t')
data.head()
# Data su tvorene meraniami nejakej veliciny v jednotlivych hodinach dna.
# Co den, to riadok. Kazda hodina ma zvlast stlpec + je tu stlpec pre nejaky flag, kto
# su tam nejak divne hodnoty, ktore by tam asi nemali byt -999 a -9999
# datum bude asi index
# v subore nieje hlavicka
```

Out[100]:

	1990-01-01	-999.000	0	-999.000.1	0.1	-999.000.2	0.2	-999.000.3	0.3	\
0	1990-01-02	-999.0	0	-999.0	0	-999.0	0	-999.0	0	
1	1990-01-03	51.0	1	50.0	1	47.0	1	48.0	1	
2	1990-01-04	-999.0	0	-999.0	0	-999.0	0	-999.0	0	
3	1990-01-05	51.0	1	51.0	1	48.0	1	50.0	1	
4	1990-01-06	-999.0	0	-999.0	0	-999.0	0	-999.0	0	

	-999.000.4	...	-999.000.19	0.19	-999.000.20	0.20	-999.000.21	0.21	\
0	-999.0	...	57.0	1	58.0	1	54.0	1	
1	51.0	...	84.0	1	75.0	1	-999.0	0	
2	-999.0	...	69.0	1	65.0	1	64.0	1	
3	51.0	...	-999.0	0	-999.0	0	-999.0	0	
4	-999.0	...	-999.0	0	-999.0	0	-999.0	0	

	-999.000.22	0.22	-999.000.23	0.23
0	49.0	1	48.0	1
1	-999.0	0	-999.0	0
2	60.0	1	59.0	1
3	-999.0	0	-999.0	0
4	-999.0	0	-999.0	0

[5 rows x 49 columns]

In [101]:
```
filename = "data/BETR8010000800100hour.1-1-1990.31-12-2012"

data = pd.read_csv(filename, sep='\t', header=None,
                   na_values=[-999, -9999], index_col=0)
# vela upratovania dat vieme spravit uz pri nacitani
data.head()
```

Out[101]:

0	1	2	3	4	5	6	7	8	9	10	...	39	40	\
											...			
1990-01-01	NaN	0	NaN	0	NaN	0	NaN	0	NaN	0	...	NaN	0	
1990-01-02	NaN	0	NaN	0	NaN	0	NaN	0	NaN	0	...	57.0	1	
1990-01-03	51.0	1	50.0	1	47.0	1	48.0	1	51.0	1	...	84.0	1	
1990-01-04	NaN	0	NaN	0	NaN	0	NaN	0	NaN	0	...	69.0	1	
1990-01-05	51.0	1	51.0	1	48.0	1	50.0	1	51.0	1	...	NaN	0	

0	41	42	43	44	45	46	47	48

```
0
1990-01-01   NaN   0   NaN   0   NaN   0   NaN   0
1990-01-02  58.0   1  54.0   1  49.0   1  48.0   1
1990-01-03  75.0   1   NaN   0   NaN   0   NaN   0
1990-01-04  65.0   1  64.0   1  60.0   1  59.0   1
1990-01-05   NaN   0   NaN   0   NaN   0   NaN   0

[5 rows x 48 columns]
```

In [102]:
```
# skusime povyhadzovat tie flagy, ktore nas nezaujimaju. Zhodou okolnosti je to kazdy
data = data.drop(data.columns[1::2], axis=1)
data.head()
```

Out[102]:
```
              1     3     5     7     9     11    13    15    17    19  ...   \
0                                                                       ...
1990-01-01   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  ...
1990-01-02   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  48.0  ...
1990-01-03  51.0  50.0  47.0  48.0  51.0  52.0  58.0  57.0   NaN   NaN  ...
1990-01-04   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN  ...
1990-01-05  51.0  51.0  48.0  50.0  51.0  58.0  65.0  66.0  69.0  74.0  ...

              29    31    33    35    37    39    41    43    45    47
0
1990-01-01   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
1990-01-02  55.0  59.0  58.0  59.0  58.0  57.0  58.0  54.0  49.0  48.0
1990-01-03  69.0  74.0   NaN   NaN  103.0  84.0  75.0   NaN   NaN   NaN
1990-01-04   NaN  71.0  74.0  70.0  70.0  69.0  65.0  64.0  60.0  59.0
1990-01-05   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN

[5 rows x 24 columns]
```

In [103]:
```
["{:02d}".format(i) for i in range(len(data.columns))]
```

Out[103]:
```
['00',
 '01',
 '02',
 '03',
 '04',
 '05',
 '06',
 '07',
 '08',
 '09',
 '10',
 '11',
 '12',
 '13',
 '14',
 '15',
```

```
 '16',
 '17',
 '18',
 '19',
 '20',
 '21',
 '22',
 '23']
```

In [104]:
```
# mam nejako rozsypane nazvy stlpcov
data.columns = ["{:02d}".format(i) for i in range(len(data.columns))]
data.head()
```

Out[104]:

	00	01	02	03	04	05	06	07	08	09	...	\
0											...	
1990-01-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	
1990-01-02	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	48.0	...	
1990-01-03	51.0	50.0	47.0	48.0	51.0	52.0	58.0	57.0	NaN	NaN	...	
1990-01-04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	
1990-01-05	51.0	51.0	48.0	50.0	51.0	58.0	65.0	66.0	69.0	74.0	...	

	14	15	16	17	18	19	20	21	22	23
0										
1990-01-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1990-01-02	55.0	59.0	58.0	59.0	58.0	57.0	58.0	54.0	49.0	48.0
1990-01-03	69.0	74.0	NaN	NaN	103.0	84.0	75.0	NaN	NaN	NaN
1990-01-04	NaN	71.0	74.0	70.0	70.0	69.0	65.0	64.0	60.0	59.0
1990-01-05	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

[5 rows x 24 columns]

In [105]:
```
data = data.stack()
data.head()
```

Out[105]:
```
1990-01-02  09    48.0
            12    48.0
            13    50.0
            14    55.0
            15    59.0
dtype: float64
```

In [106]:
```
type(data) # vysledok preusporiadania je viacdimenzionaly Series objekt a nie DataFram
```

Out[106]:
```
pandas.core.series.Series
```

In [107]:
```
# mohli by sme nejak normalne poemnovat stlpec
import os
_, fname = os.path.split(filename)
station = fname[:7]
print(filename)
print(station)
```

```
data/BETR8010000800100hour.1-1-1990.31-12-2012
BETR801
```

```
In [108]: data = data.reset_index(name=station) #reset index mi z toho sprav data frame
          # data = data.reset_index() #reset index mi z toho sprav data frame
          print(type(data))
          data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Out[108]:             0 level_1  BETR801
          0  1990-01-02      09     48.0
          1  1990-01-02      12     48.0
          2  1990-01-02      13     50.0
          3  1990-01-02      14     55.0
          4  1990-01-02      15     59.0
```

```
In [109]: data = data.rename(columns = {0:'date', 'level_1':'hour'})
          data.head()
```

```
Out[109]:          date hour  BETR801
          0  1990-01-02   09     48.0
          1  1990-01-02   12     48.0
          2  1990-01-02   13     50.0
          3  1990-01-02   14     55.0
          4  1990-01-02   15     59.0
```

```
In [110]: # teraz tomu vyrobime novy index z datumu a hodiny
          data.index = pd.to_datetime(data['date'] + ' ' + data['hour'])
          data.head()
```

```
Out[110]:                            date hour  BETR801
          1990-01-02 09:00:00  1990-01-02   09     48.0
          1990-01-02 12:00:00  1990-01-02   12     48.0
          1990-01-02 13:00:00  1990-01-02   13     50.0
          1990-01-02 14:00:00  1990-01-02   14     55.0
          1990-01-02 15:00:00  1990-01-02   15     59.0
```

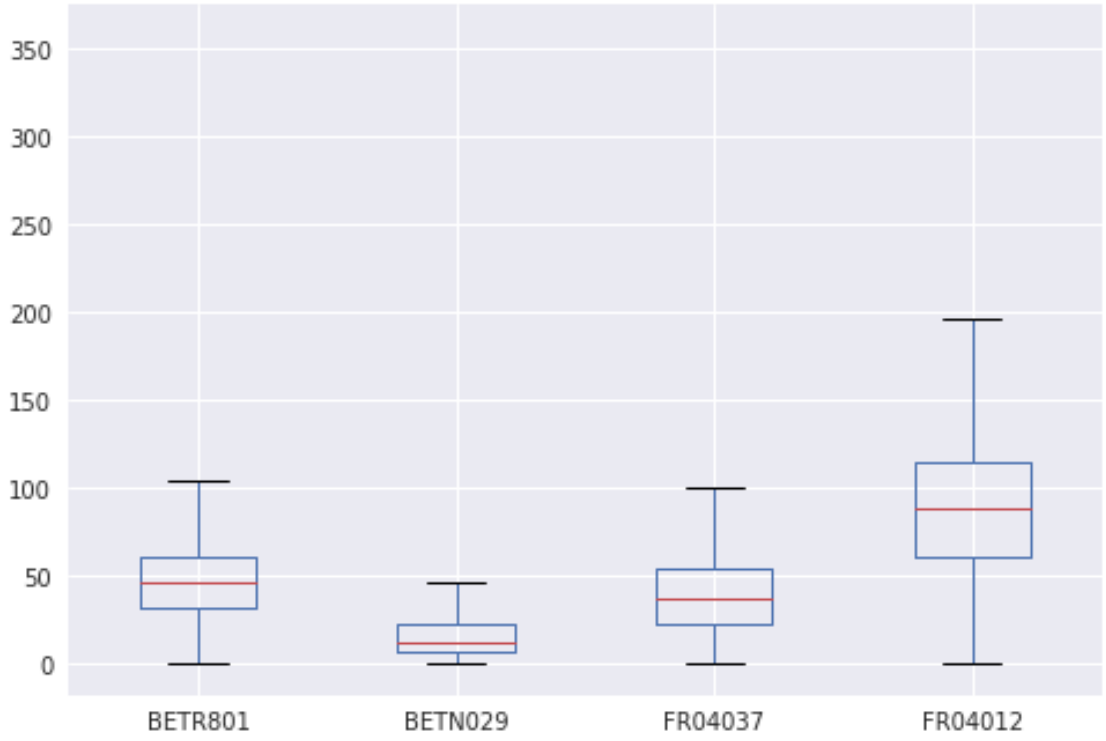
```
In [111]: # a zmazeme nepotrebne stlpce
          data = data.drop(['date', 'hour'], axis=1)
          data.head()
          # Teraz uz mame data, s ktorymi sa uz da nieco robit
```

```
Out[111]:                      BETR801
          1990-01-02 09:00:00     48.0
          1990-01-02 12:00:00     48.0
          1990-01-02 13:00:00     50.0
          1990-01-02 14:00:00     55.0
          1990-01-02 15:00:00     59.0
```

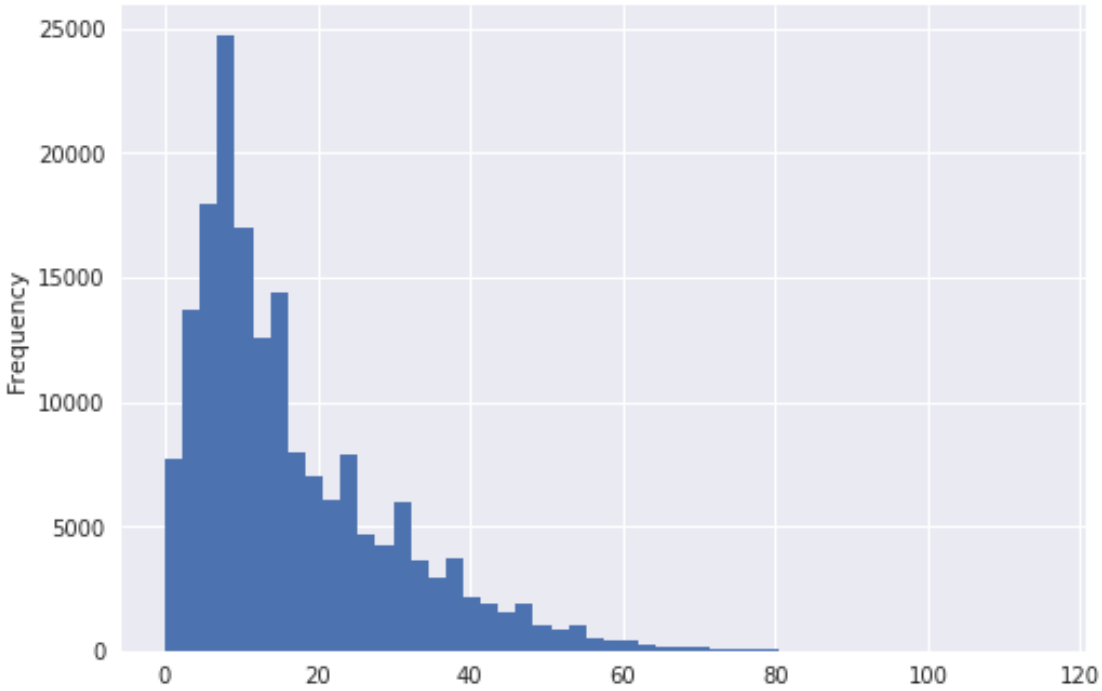
Ja mam tych suborov viac. Kazdy obsahuje data z inej meracej stanice. Aby som zjednodusil prezentaciu, tak predchadzajuci kod som dal do cyklu a vlozil do skriptu

```
In [112]: import airbase
          no2 = airbase.load_data()

In [113]: no2.head(3)

Out[113]:                      BETR801  BETN029  FR04037  FR04012
          1990-01-01 00:00:00      NaN     16.0      NaN      NaN
          1990-01-01 01:00:00      NaN     18.0      NaN      NaN
          1990-01-01 02:00:00      NaN     21.0      NaN      NaN

In [114]: no2.tail()

Out[114]:                      BETR801  BETN029  FR04037  FR04012
          2012-12-31 19:00:00     21.0      2.5     28.0     67.0
          2012-12-31 20:00:00     16.5      2.0     16.0     47.0
          2012-12-31 21:00:00     14.5      2.5     13.0     43.0
          2012-12-31 22:00:00     16.5      3.5     14.0     42.0
          2012-12-31 23:00:00     15.0      3.0     13.0     49.0

In [115]: no2.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 198895 entries, 1990-01-01 00:00:00 to 2012-12-31 23:00:00
Data columns (total 4 columns):
BETR801    170794 non-null float64
BETN029    174807 non-null float64
FR04037    120384 non-null float64
FR04012    119448 non-null float64
dtypes: float64(4)
memory usage: 7.6 MB


In [116]: no2.describe()

Out[116]:              BETR801        BETN029        FR04037        FR04012
          count  170794.000000  174807.000000  120384.000000  119448.000000
          mean       47.914561      16.687756      40.040005      87.993261
          std        22.230921      13.106549      23.024347      41.317684
          min         0.000000       0.000000       0.000000       0.000000
          25%        32.000000       7.000000      23.000000      61.000000
          50%        46.000000      12.000000      37.000000      88.000000
          75%        61.000000      23.000000      54.000000     115.000000
          max       339.000000     115.000000     256.000000     358.000000

In [117]: no2.plot(kind='box')

Out[117]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9dc44260f0>
```

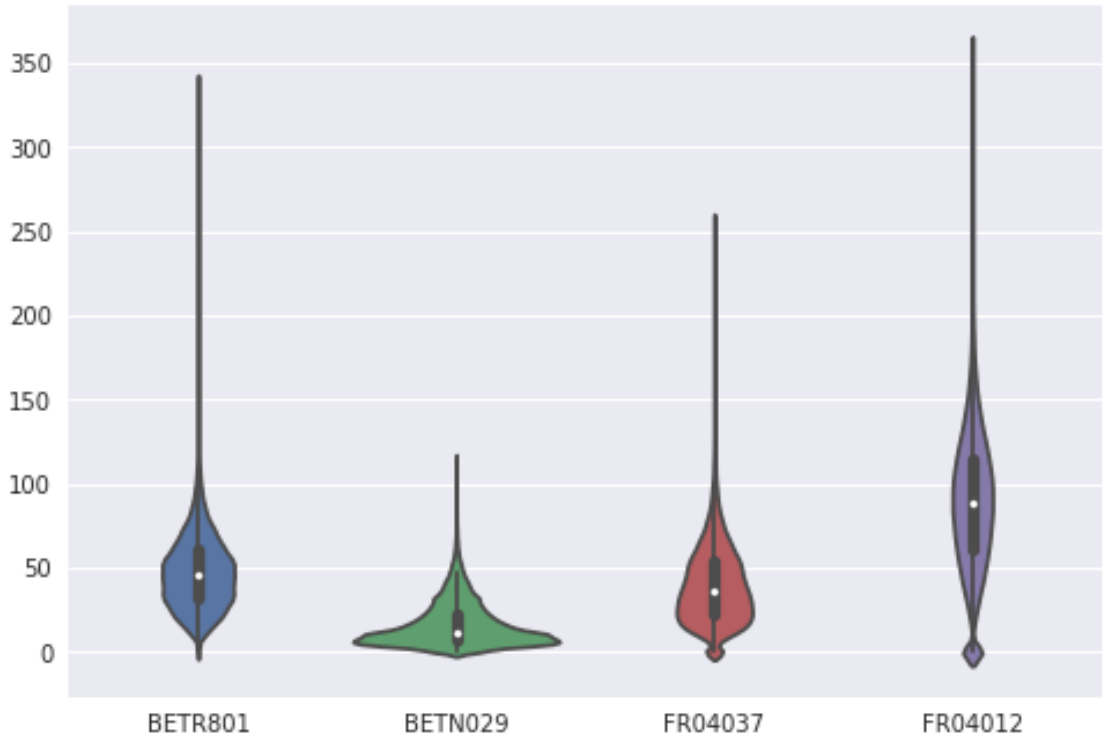
```
/usr/local/lib/python3.5/dist-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: F
  (prop.get_family(), self.defaultFamily[fontext]))
```




```
In [118]: no2['BETN029'].plot(kind='hist', bins=50)

Out[118]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9dc32d52b0>

/usr/local/lib/python3.5/dist-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: F
  (prop.get_family(), self.defaultFamily[fontext]))
```



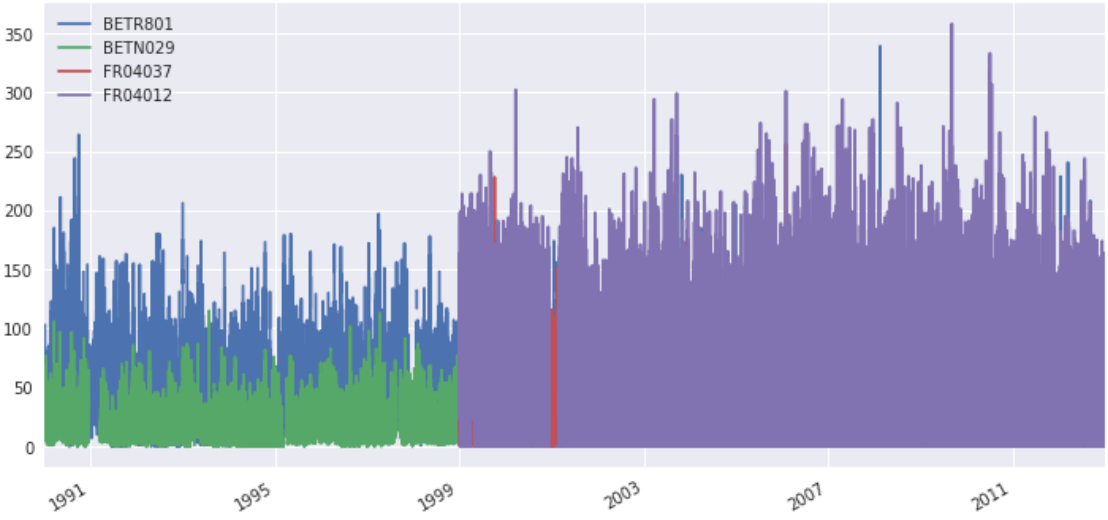

```
In [119]: import seaborn

In [120]: seaborn.violinplot(no2)

/usr/local/lib/python3.5/dist-packages/seaborn/categorical.py:2342: UserWarning: The violinplot
  warnings.warn(msg, UserWarning)


Out[120]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9dc4b714e0>

/usr/local/lib/python3.5/dist-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: F
  (prop.get_family(), self.defaultFamily[fontext]))
```



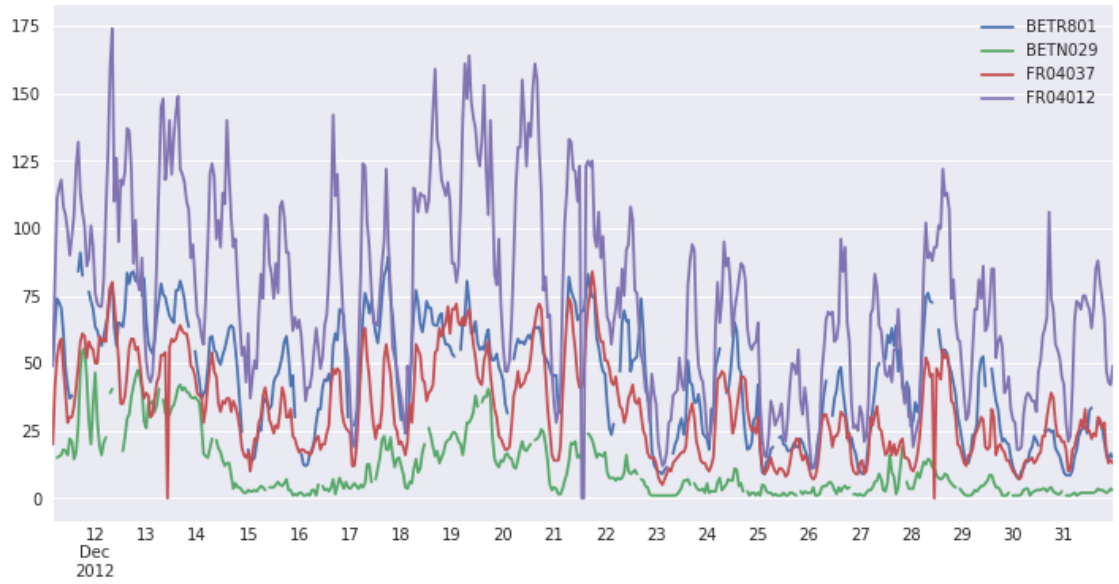

```
In [121]: no2.plot(figsize=(12,6))
          # mozem si vyplotovat surove data, ale je otazne, co mi to povie
```

```
Out[121]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9dc4b42048>
```

```
/usr/local/lib/python3.5/dist-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: F
  (prop.get_family(), self.defaultFamily[fontext]))
```



```
In [122]: # mozem si povedat, ze chcem len nejaku mensiu cast
          no2[-500:].plot(figsize=(12,6))

Out[122]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9dc264da20>

/usr/local/lib/python3.5/dist-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: F
  (prop.get_family(), self.defaultFamily[fontext]))
```



alebo pouzijem zaujimavejsie operacie s casovymi radmi

```
In [123]: no2.index # kedze index su casy, tak viem robit s nimi zaujimave veci

Out[123]: DatetimeIndex(['1990-01-01 00:00:00', '1990-01-01 01:00:00',
                         '1990-01-01 02:00:00', '1990-01-01 03:00:00',
                         '1990-01-01 04:00:00', '1990-01-01 05:00:00',
                         '1990-01-01 06:00:00', '1990-01-01 07:00:00',
                         '1990-01-01 08:00:00', '1990-01-01 09:00:00',
                         ...
                         '2012-12-31 14:00:00', '2012-12-31 15:00:00',
                         '2012-12-31 16:00:00', '2012-12-31 17:00:00',
                         '2012-12-31 18:00:00', '2012-12-31 19:00:00',
                         '2012-12-31 20:00:00', '2012-12-31 21:00:00',
                         '2012-12-31 22:00:00', '2012-12-31 23:00:00'],
                        dtype='datetime64[ns]', length=198895, freq=None)

In [124]: no2["2010-01-01 09:00": "2010-01-01 12:00"] # napriklad definovat rozsahy pomocou stri
```

Out[124]:

	BETR801	BETN029	FR04037	FR04012
2010-01-01 09:00:00	17.0	7.0	19.0	41.0
2010-01-01 10:00:00	18.0	5.0	21.0	48.0
2010-01-01 11:00:00	17.0	4.0	23.0	63.0
2010-01-01 12:00:00	18.0	4.0	22.0	57.0

In [125]:
```
no2['2012'] # alebo takto vybrat vsetky data z jedneho konkretneho roku
# no2['2012'].head()
# no2['2012/03'] # alebo len data z marca
```

Out[125]:

	BETR801	BETN029	FR04037	FR04012
2012-01-01 00:00:00	21.0	1.0	17.0	56.0
2012-01-01 01:00:00	18.0	1.0	16.0	50.0
2012-01-01 02:00:00	20.0	1.0	14.0	46.0
2012-01-01 03:00:00	16.0	1.0	17.0	47.0
2012-01-01 04:00:00	13.0	1.0	18.0	45.0
2012-01-01 05:00:00	17.0	1.0	15.0	36.0
2012-01-01 06:00:00	15.0	1.0	11.0	31.0
2012-01-01 07:00:00	13.0	1.0	12.0	30.0
2012-01-01 08:00:00	15.0	1.0	10.0	28.0
2012-01-01 09:00:00	15.0	1.0	12.0	42.0
2012-01-01 10:00:00	19.0	NaN	14.0	44.0
2012-01-01 11:00:00	28.0	1.0	18.0	54.0
2012-01-01 12:00:00	25.0	4.0	17.0	39.0
2012-01-01 13:00:00	29.5	4.0	15.0	45.0
2012-01-01 14:00:00	31.5	1.0	17.0	54.0
2012-01-01 15:00:00	33.5	1.0	18.0	69.0
2012-01-01 16:00:00	32.5	1.0	19.0	66.0
2012-01-01 17:00:00	30.0	1.0	19.0	69.0
2012-01-01 18:00:00	25.0	2.5	17.0	57.0
2012-01-01 19:00:00	20.0	1.0	15.0	47.0
2012-01-01 20:00:00	14.0	4.0	12.0	43.0
2012-01-01 21:00:00	13.0	1.0	11.0	35.0
2012-01-01 22:00:00	15.0	2.5	10.0	26.0
2012-01-01 23:00:00	14.0	1.0	11.0	39.0
2012-01-02 00:00:00	NaN	1.0	10.0	29.0
2012-01-02 01:00:00	NaN	1.0	9.0	22.0
2012-01-02 02:00:00	10.5	1.0	9.0	15.0
2012-01-02 03:00:00	12.0	1.0	7.0	12.0
2012-01-02 04:00:00	12.0	1.0	11.0	27.0
2012-01-02 05:00:00	39.0	1.0	26.0	44.0
...	...	...	...	...
2012-12-30 18:00:00	24.5	4.0	39.0	74.0
2012-12-30 19:00:00	25.0	2.5	37.0	70.0
2012-12-30 20:00:00	18.5	2.0	28.0	57.0
2012-12-30 21:00:00	17.0	1.5	23.0	55.0
2012-12-30 22:00:00	15.5	1.5	23.0	52.0
2012-12-30 23:00:00	12.5	2.5	21.0	45.0

```
2012-12-31 00:00:00      9.5      NaN     21.0     42.0
2012-12-31 01:00:00      8.5      1.0     18.0     28.0
2012-12-31 02:00:00      8.5      1.0     10.0     21.0
2012-12-31 03:00:00      8.5      1.0     11.0     23.0
2012-12-31 04:00:00     10.5      1.5     18.0     41.0
2012-12-31 05:00:00     15.5      2.0     19.0     66.0
2012-12-31 06:00:00     18.0      1.0     23.0     73.0
2012-12-31 07:00:00     23.0      1.5     25.0     72.0
2012-12-31 08:00:00     25.0      2.0     29.0     70.0
2012-12-31 09:00:00     26.0      2.0     26.0     75.0
2012-12-31 10:00:00     26.5      2.0     33.0     75.0
2012-12-31 11:00:00     24.0      2.0     25.0     72.0
2012-12-31 12:00:00     32.5      2.0     25.0     70.0
2012-12-31 13:00:00     33.5      2.0     22.0     63.0
2012-12-31 14:00:00      NaN      2.0     24.0     71.0
2012-12-31 15:00:00      NaN      2.5     23.0     85.0
2012-12-31 16:00:00     28.0      3.5     30.0     88.0
2012-12-31 17:00:00     27.5      3.0     29.0     80.0
2012-12-31 18:00:00     26.0      3.0     26.0     74.0
2012-12-31 19:00:00     21.0      2.5     28.0     67.0
2012-12-31 20:00:00     16.5      2.0     16.0     47.0
2012-12-31 21:00:00     14.5      2.5     13.0     43.0
2012-12-31 22:00:00     16.5      3.5     14.0     42.0
2012-12-31 23:00:00     15.0      3.0     13.0     49.0

[8784 rows x 4 columns]
```

In [126]:
```
# komponenty datumu su pristupne z indexu
# no2.index.hour
no2.index.year
```

Out[126]:
```
array([1990, 1990, 1990, ..., 2012, 2012, 2012], dtype=int32)
```

In [127]:
```
# a co je zaujimavejsie viem zmenit vzorkovaciu frekvenciu
no2.resample('D').mean().head()
```

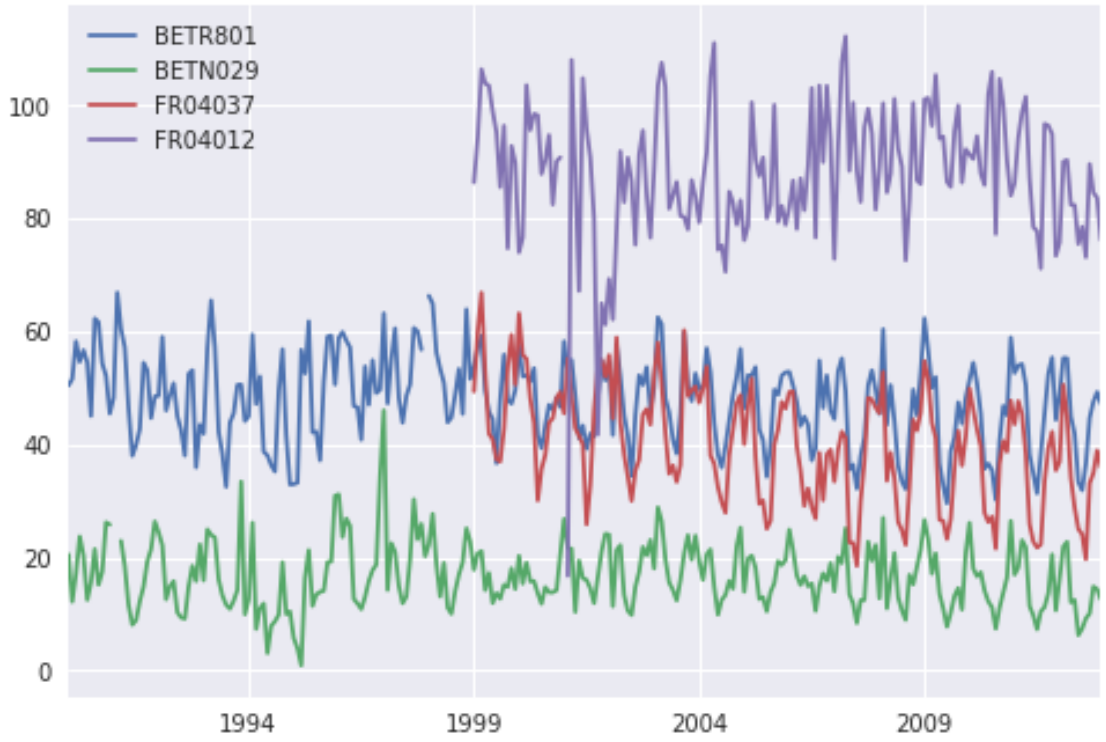
Out[127]:

	BETR801	BETN029	FR04037	FR04012
1990-01-01	NaN	21.500000	NaN	NaN
1990-01-02	53.923077	35.000000	NaN	NaN
1990-01-03	63.000000	29.136364	NaN	NaN
1990-01-04	65.250000	42.681818	NaN	NaN
1990-01-05	63.846154	40.136364	NaN	NaN

In [128]:
```
no2.resample('M').mean().plot()
# toto sa zda, ze povie uz trochu viac. Napriklad, ze je tu asi nejaka sezonnost
```

Out[128]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9dc2cff4a8>
```

```
/usr/local/lib/python3.5/dist-packages/matplotlib/font_manager.py:1297: UserWarning: findfont: F
  (prop.get_family(), self.defaultFamily[fontext]))
```




```
In [1]: no2.resample('A').mean().plot()
        # a mozno aj nejaky dlhodoby trend
```

```
        ---------------------------------------------------------------------------
        NameError                                 Traceback (most recent call last)

        <ipython-input-1-aaacf2f84af1> in <module>()
    ----> 1 no2.resample('A').mean().plot()
          2 # a mozno aj nejaky dlhodoby trend


        NameError: name 'no2' is not defined
```

```
In [ ]: no2['2012-3':'2012-4'].resample('D').mean().plot()
        # mozno je tam aj nejaka tyzdenna sezonnost
```

```
In [ ]: # mozem pouzit aj viacero agregacnych funkcii a porovnat si ich
        no2.loc['2009':, 'FR04037'].resample('M').agg(['mean', 'median']).plot()
        # no2.loc['2009':, 'FR04037'].resample('M').agg(['mean', 'std']).plot()
```

3.9 Dalsia casta operacia je groupby

urcite poznate z SQL

```
In [ ]: df = pd.DataFrame({'key':['A','B','C','A','B','C','A','B','C'],
                           'data': [0, 5, 10, 5, 10, 15, 10, 15, 20]})
        df
```

```
In [ ]: df.groupby('key').aggregate('sum') # df.groupby('key').sum()
```

```
In [ ]: no2['month'] = no2.index.month
        no2.head()
```

```
In [ ]: no2.groupby('month').mean()
```

```
In [ ]: no2.groupby('month').mean().plot()
```

4 Priklad analyzy s pouzitim ineho datasetu

tentokrat to nebudu casove rady, ale klasicky dataset na predvadzanie kalsifikacie Iris

```
In [ ]: iris_data = pd.read_csv('data/iris-data.csv')
        iris_data.head()
        # toto je trochu spotvoreny dataset kvetiniek
```

```
In [ ]: iris_data.info()
```

```
In [ ]: iris_data.describe()
```

```
In [ ]: seaborn.pairplot(iris_data.dropna(), hue='class')
```

```
In [ ]: iris_data.loc[iris_data['class'] == 'versicolor', 'class'] = 'Iris-versicolor'
        iris_data.loc[iris_data['class'] == 'Iris-setossa', 'class'] = 'Iris-setosa'

        iris_data['class'].unique()
```

```
In [ ]: seaborn.pairplot(iris_data.dropna(), hue='class')
```

```
In [ ]: iris_data.loc[iris_data['class'] == 'Iris-versicolor', 'sepal_length_cm'].hist()
```

```
In [ ]: plt.rc("lines", markeredgewidth=0.5)
        iris_data.loc[iris_data['class'] == 'Iris-versicolor', 'sepal_length_cm'].plot(kind='box
```

```
In [ ]: iris_data.loc[(iris_data['class'] == 'Iris-versicolor') & (iris_data['sepal_length_cm']
```

```
In [ ]: iris_data.loc[(iris_data['class'] == 'Iris-versicolor') & (iris_data['sepal_length_cm']
```

```
In [ ]: mask = (iris_data['class'] == 'Iris-versicolor') & (iris_data['sepal_length_cm'] < 1 )

        iris_data.loc[mask, 'sepal_length_cm'] = iris_data.loc[mask, 'sepal_length_cm'] * 100
```

```
In [ ]: iris_data.loc[mask, 'sepal_length_cm']
```

```
In [ ]: seaborn.pairplot(iris_data.dropna(), hue='class')
```

4.1 Skusme sa pozriet este na tie chybajuce hodnoty

```
In [ ]: iris_data.loc[(iris_data['sepal_length_cm'].isnull()) |
                     (iris_data['sepal_width_cm'].isnull()) |
                     (iris_data['petal_length_cm'].isnull()) |
                     (iris_data['petal_width_cm'].isnull())]
```

```
In [ ]: iris_data.loc[iris_data['class'] == 'Iris-setosa', 'petal_width_cm'].hist()
```

```
In [ ]: average_petal_width = iris_data.loc[iris_data['class'] == 'Iris-setosa', 'petal_width_cm

        iris_data.loc[(iris_data['class'] == 'Iris-setosa') &
                     (iris_data['petal_width_cm'].isnull()),
                     'petal_width_cm'] = average_petal_width
```

```
In [ ]: seaborn.pairplot(iris_data, hue='class')
```

5 Sumar co si zobrat z tejto explorativnej analyzy

- Uisite sa, ze data su kodovane spravne (najcastejsie sa treba pozriet manualne do dat)
- Uistite sa, ze data spadaju do ocakavaneho rozsahu a vsetky maju ocakavany tvar (napriklad format casu)
- Porieste chybajuce data napriklad vyhodenim alebo nahradenim priemerom (priemer musi byt s ohladom na triedu)
- Nikdy nesahajte do dat manualne. Vzdy pouzivajte kod, ktory si odlozite a pouzijete vzdy ked budete opakovat experiment. Chceme aby bola analyza reprodukovatelna
- Spravte si grafy vsetkeho, co sa len da, aby ste si vizualne potvrdili, ze nieco je tak ako by malo byt

5.1 SQL v Pandas

```
In [ ]: from pandasql import sqldf
```

```
In [ ]: from pandasql import load_meat, load_births

        meat = load_meat()
        births = load_births()
```

```
In [ ]: type(meat)
```

```
In [ ]: meat.head()
```

```
In [ ]: births.head()
```

```
In [ ]: data = {'meat': meat}
```

```
In [ ]: sqldf('select * from meat limit 10', data)
```

```
In [ ]: data2 = {'meat2': meat}
```

```
In [ ]: sqldf('select * from meat2 limit 10', data2)

In [ ]: sqldf('select * from meat limit 10', locals())

In [ ]: sqldf('select * from births limit 10', locals())

In [ ]: q = """
            SELECT
                m.date
                , b.births
                , m.beef
            FROM
                meat m
            INNER JOIN
                births b
                    on m.date = b.date
            ORDER BY
                m.date
            LIMIT 100;
            """

        joined = sqldf(q, locals())
        print(joined.head())
```

Pandasql bezi na SQLite3, takze vsetky klasicke opercaie v SQL viete robit aj tu. Funguju podmienky, vnorene dopyty, joiny, union, funkcie, ...

6 Zopar dalsich uzitocnych veci pri praci s Pandas DataFrame

```
In [ ]: df = pd.read_csv('https://raw.githubusercontent.com/rasbt/python_reference/master/Data/s
        df.head()

In [ ]: # premenovanie vybranych stlpcov
        df = df.rename(columns={'P': 'points',
                                'GP': 'games',
                                'SOT': 'shots_on_target',
                                'G': 'goals',
                                'PPG': 'points_per_game',
                                'A': 'assists',})
        df.head()
```

6.1 transformacia hodnot v stlpci

```
In [ ]: df['SALARY'] = df['SALARY'].apply(lambda x: x.strip('$m'))
        df.head()
```

6.2 Pridanie stlpcu

```
In [ ]: df['team'] = pd.Series('', index=df.index)
        df['position'] = pd.Series('', index=df.index)
        df.head()
```

6.3 Transformacia ineho stlpca a naplnenie dalsich

```
In [ ]: def process_player_col(text):
            name, rest = text.split('\n')
            position, team = [x.strip() for x in rest.split('  ')]
            return pd.Series([name, team, position])

        df[['PLAYER', 'team', 'position']] = df.PLAYER.apply(process_player_col)
        df.head()
```

6.4 Zistenie, kolko stlpcov ma prazdne hodnoty

```
In [ ]: df.shape[0] - df.dropna().shape[0]
```

6.5 Vyber riadkov, kde su prazdne hodnoty

```
In [ ]: df[df['assists'].isnull()]
```

6.6 Vyber plnych riadkov

```
In [ ]: df[df['assists'].notnull()]
        # df[~df['assists'].isnull()]
```

6.7 Nahradzanie prazdnych hodnot

```
In [ ]: # predtym sme to robili manulane.
        # iris_data.loc[(iris_data['class'] == 'Iris-setosa') & (iris_data['petal_width_cm'].isn

        # Da sa na to pouzit takato pekna funkcia
        df.fillna(value=0, inplace=True)
        df
```

6.8 Existuje vsak este elegantnejsi sposob

```
In [ ]: df = pd.read_csv('https://raw.githubusercontent.com/rasbt/python_reference/master/Data/s
        df = df.rename(columns={'P': 'points',
                                'GP': 'games',
                                'SOT': 'shots_on_target',
                                'G': 'goals',
                                'PPG': 'points_per_game',
                                'A': 'assists',})
        df['SALARY'] = df['SALARY'].apply(lambda x: x.strip('$m'))
        df[['PLAYER', 'team', 'position']] = df.PLAYER.apply(process_player_col)
```

```
        df.head()
```

```
In [ ]: from sklearn.preprocessing import Imputer
        imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
        df[['games', 'assists']] = imp.fit_transform(df[['games', 'assists']].values)
        df.head()
```

Pozor, toto doplnanie neberie do uvahy triedu

```
In [ ]: df.games.mean()
```

```
In [ ]: df[df.position == 'Forward'].games.mean()
```

6.9 Spajanie podmienok

```
In [ ]: df[ (df['team'] == 'Arsenal') | (df['team'] == 'Chelsea') ]
```

```
In [ ]: df[ (df['team'] == 'Arsenal') & (df['position'] == 'Forward') ]
```

7 Nejake zdroje na studium

- http://nbviewer.jupyter.org/format/slides/github/jorisvandenbossche/2015-PyDataParis/blob/master/pandas_introduction.ipynb
- http://nbviewer.jupyter.org/github/rasbt/python_reference/blob/master/tutorials/things_in_pandas.ip
- Pandas Cheat Sheet, nejaky komentar k tomu

8 Nejake dalsie nastroje

- OpenRefine - standalone nastroj na cistenie a pozeranie sa do dat
- Trifacta