

Pandas

Alexandr Pak

Основные объекты в Pandas

1. Серия (1D)

```
s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
```

```
s
a    1.321250
b    0.365307
c    0.709577
d    0.542710
e   -0.212721
dtype: float64
```

индекс						
серия						

Похоже на словарь:

```
print s['b']
0.365307109524
```

```
print s.get('z', 'error')
error
```

Автоматическое выравнивание по индексу

```
print s + s[1:]
```

```
a         NaN
b    0.730614
c    1.419154
d    1.085419
e   -0.425441
```

Основные объекты в Pandas

2. ДатаФрейм (2D)

```
df = pd.DataFrame(np.random.randn(8, 3),  
index=pd.date_range('1/1/2000', periods=8),  
columns=['A', 'B', 'C'])
```

df

	A	B	C
2000-01-01	0.684918	0.240427	-0.030283
2000-01-02	0.533952	-0.573713	-1.602537
2000-01-03	-1.291314	-0.650594	1.771561
2000-01-04	2.813297	-1.093390	-0.209462
2000-01-05	0.894795	-0.574468	0.765031
2000-01-06	1.513772	0.618505	-1.402341
2000-01-07	-0.435267	-1.199286	0.990490
2000-01-08	-0.541890	0.590653	-0.530153

КОЛОНКИ				
индекс	серия	серия	серия	серия

Основные объекты в Pandas

3. Панель (3D)

```
wp = pd.Panel(np.random.randn(2, 5, 4), items=['Item1', 'Item2'],  
major_axis=pd.date_range('1/1/2000', periods=5),  
minor_axis=['A', 'B', 'C', 'D'])  
wp
```

```
<class 'pandas.core.panel.Panel'>  
Dimensions: 2 (items) x 5 (major_axis) x 4 (minor_axis)  
Items axis: Item1 to Item2  
Major_axis axis: 2000-01-01 00:00:00 to 2000-01-05 00:00:00  
Minor_axis axis: A to D
```

Подключение пакетов

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

Загрузка данных

```
# Excel
data2 = pd.read_excel('D:\\filename.xlsx', sheetname='1')
# csv-файл
data = pd.read_csv('D:\\filename.csv', sep=';', decimal=',')
data.to_csv('foo.csv') # сохранение
# HDF5
pd.read_hdf('foo.h5', 'df').to_hdf('foo.h5', 'df') #
сохранение
```

Важно: не забывать сепараторы

После загрузки – 1. Смотрим на данные

```
datatrain = pd.read_csv('D:\\Competitions\\Rossman\\train.csv')  
datatrain[:3]
```

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	5	2015-07-31	5263	555	1	1	0	1
1	2	5	2015-07-31	6064	625	1	1	0	1
2	3	5	2015-07-31	8314	821	1	1	0	1

В ноутбуке `print datatrain[:3]` смотрится хуже

2. Приводим данные к нужным типам

```
datatrain.Date = pd.to_datetime(datatrain.Date)
```


Создание ДатаФрейма

первый способ

```
data = pd.DataFrame({ 'A' : [1., 4., 2., 1.],  
  'B' : pd.Timestamp('20130102'),  
  'C' : pd.Series(1,index=list(range(4)),dtype='float32'),  
  'D' : np.array([3] * 4,dtype='int32'),  
  'E' : pd.Categorical(["test","train","test","train"]),  
  'F' : 'foo' }, index=pd.period_range('Jan-2000', periods=4,  
freq='M'))  
print data
```

	A	B	C	D	E	F
2000-01	1	2013-01-02	NaN	3	test	foo
2000-02	4	2013-01-02	NaN	3	train	foo
2000-03	2	2013-01-02	NaN	3	test	foo
2000-04	1	2013-01-02	NaN	3	train	foo

Создание ДатаФрейма

второй способ

```
tmp = dict([('A', [1., np.nan, 2., 1.]), ('B', [2.2, np.nan, np.nan,  
0.0])]) # ещё один способ
```

```
data2 = pd.DataFrame(tmp)
```

```
print data2
```

	A	B
0	1	2.2
1	NaN	NaN
2	2	NaN
3	1	0.0

Простейшие операции

```
# простейшие операции
```

```
# столбцы
```

```
print data.columns
```

```
Index([u'A', u'B', u'C', u'D', u'E', u'F'], dtype='object')
```

```
# строки - но тут временная индексация
```

```
print data.index
```

```
<class 'pandas.tseries.period.PeriodIndex'>
```

```
[2000-01, ..., 2000-04]
```

```
# сортировка
```

```
print data.sort(columns='A')
```

	A	B	C	D	E	F
2000-01	1	2013-01-02	NaN	3	test	foo
2000-04	1	2013-01-02	NaN	3	train	foo
2000-03	2	2013-01-02	NaN	3	test	foo
2000-02	4	2013-01-02	NaN	3	train	foo

Простейшие операции

```
# превращение в пр-матрицу  
print data.values # без скобок
```

```
array([[1.0, Timestamp('2013-01-02 00:00:00'), nan, 3, 'test', 'foo'],  
       [4.0, Timestamp('2013-01-02 00:00:00'), nan, 3, 'train', 'foo'],  
       [2.0, Timestamp('2013-01-02 00:00:00'), nan, 3, 'test', 'foo'],  
       [1.0, Timestamp('2013-01-02 00:00:00'), nan, 3, 'train', 'foo']],  
dtype=object)
```

Статистика по признакам

типы

```
print data.dtypes
```

```
A          float64
B    datetime64[ns]
C          float32
D          int32
E          object
F          object
dtype: object
```

Изменение типа: .astype()

статистика + транспонирование

```
print data.describe().T # транспонирование часто удобно!
```

	count	mean	std	min	25%	50%	75%	max
A	4	2	1.414214	1	1	1.5	2.5	4
C	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
D	4	3	0.000000	3	3	3.0	3.0	3

```
print data.describe(include=['object']) # категориальные признаки
```

Совет: смотрите на число уникальных элементов .nunique()

```
# число уникальных элементов (можно через describe)
for i in data.columns: # можно просто data
    print str(i) + ':' + str(data[i].nunique())
```

A:3

B:1

C:0

D:1

E:2

F:1

.columns – это список, по нему можно бегать в цикле

**Кстати, нельзя data.i.nunique() нельзя пропустить str
«+» – конкатенация строк**

Переименование колонок

```
df2 = df.rename(columns={'int_col' : 'some_other_name'})  
  
# изменение текущего датафрейма  
df2.rename(columns={'some_other_name' : 'int_col'}, inplace = True)
```

Во втором случае не нужно присваивание!

Удаления

```
df = pd.DataFrame({'x': [1, 3, 2], 'y': [2, 4, 1]})  
# удаление строки  
df.drop(1, axis=0, inplace=True)  
# удаление столбца  
del df['x'] # df.drop('x', axis=1)
```

Индексация

```
data.at['2000-01','A'] = 10. # по названию
data.iat[0,1] = pd.Timestamp('19990101') # по номеру
# просто = '1999/01/01' не работает

data.loc['2000-01':'2000-02',['D','B','A']] # по названию
data.iloc[0:2,1:3] # по номеру

data.ix[0:2,1:3] # по номеру и по названию

# выбор с проверкой на вхождение
data[data['E'].isin(['test','valid'])] # полезно: isin
```

первая строка (точнее срез датафрейма) – data[:1]
последняя строка – data[-1:]
нельзя – data[1], data[1,2]
можно – data[data.columns[0]][2]

Индексация

Выбор нескольких случайных строк

```
print df.take(np.random.permutation(len(df))[:2])
```

Изменить порядок записи в датафрейме

```
data.reindex(index=data.index[::-1])  
# или data = data.iloc[::-1]
```

Умная переиндексация

```
s = pd.DataFrame({'x':[1,2,3,4], 'y':[10,20,30,40]}, index=['a','b','c','d'])  
s.reindex(index=['d','b','x'], columns=['y','z'])
```

	y	z
d	40	NaN
b	20	NaN
x	NaN	NaN

Индексация

Переиндексация

```
s = pd.Series([10, 20, 60], index=[1, 2, 6])  
s.reindex(index=[2, 3, 4, 5, 6, 7], method='ffill')
```

2	20
3	20
4	20
5	20
6	60
7	60

Для вставки колонок в любое место

```
.insert() # если df['new'] = ..., то вставляется в конец
```

Итерации

```
df = pd.DataFrame({'x': [1, 2, 1, 2], 'y': [1, 2, 3, 3], 'z': [0, 0, 0, 0]},  
index=['a', 'b', 'c', 'd'])
```

	x	y	z
a	1	1	0
b	2	2	0
c	1	3	0
d	2	3	0

```
for col in df: # не обязательно писать df.columns  
    print col
```

x

y

z

Итерации

```
for index, row in df.iterrows():  
    print index, row
```

a		b		c		d	
x	1	x	2	x	1	x	2
y	1	y	2	y	3	y	3
z	0	z	0	z	0	z	0

```
for t in df.itertuples(): # так быстрее;
```

```
    print t
```

```
('a', 1, 1, 0)
```

```
('b', 2, 2, 0)
```

```
('c', 1, 3, 0)
```

```
('d', 2, 3, 0)
```

Не модифицировать внутри итераций то, по чему итерируетесь

Сравнения

```
df1 = pd.DataFrame({'x': [1, 3, 2], 'y': [2, 4, 1]})  
df2 = pd.DataFrame({'x': [3, 1, 2], 'y': [0, 2, 2]})
```

```
print df1>=df2
```

	x	y
0	False	True
1	True	True
2	True	False

```
print (df1>=df2).any(axis=1)
```

0	True
1	True
2	True

```
print (df1>=df2).all()
```

x	False
y	False

NaN

	A	B
0	1	2.2
1	NaN	NaN
2	2	NaN
3	1	0.0

```
print data2.isnull() # маска Нанов
```

```
      A      B
0  False  False
1   True   True
2  False   True
3  False  False
```

```
print data2.mean() # nan автоматически не учитываются
```

```
A      1.333333
B      1.100000
```

```
print data2.apply(np.cumsum) # тоже обходятся nan
```

```
      A      B
0     1     2.2
1  NaN    NaN
2     3     NaN
3     4     2.2
```

не забывать `data3 = data2.apply(np.cumsum)`

NaN

	A	B
0	1	2.2
1	NaN	NaN
2	2	NaN
3	1	0.0

```
print data2.dropna() # удаление Нанов
```

```
   A    B  
0  1  2.2  
3  1  0.0
```

```
print data2.fillna(value=5.5) # заполнение Нанов
```

```
   A    B  
0  1.0  2.2  
1  5.5  5.5  
2  2.0  5.5  
3  1.0  0.0
```

```
print data2.ffill() # заполнение соседними значениями
```

```
dtype: float64
```

```
   A    B  
0  1  2.2  
1  1  2.2  
2  2  2.2  
3  1  0.0
```

NaN

Отличие от numpy

```
df = pd.DataFrame({'x': [1, np.nan], 'y': [1, 2]})
```

```
print df.mean()
```

```
x    1.0  
y    1.5
```

```
print np.mean(df)
```

```
x    1.0  
y    1.5
```

```
print np.mean(df.values)
```

```
nan
```

Комбинирование

```
df1 = pd.DataFrame({'x': [1, np.nan, 2], 'y': [2, 4, np.nan], 'z': [1, 2, 3]})  
df2 = pd.DataFrame({'x': [20, 40, np.nan], 'y': [2, 4, 20]})
```

```
print df1  
print df2
```

	x	y	z
0	1	2	1
1	NaN	4	2
2	2	NaN	3

	x	y
0	20	2
1	40	4
2	NaN	20

```
print df1.combine_first(df2)
```

	x	y	z
0	1	2	1
1	40	4	2
2	2	20	3

```
print df1.combineAdd(df2)
```

	x	y	z
0	21	4	1
1	40	8	2
2	2	20	3

Пытаемся грамотно объединить: учитывая, что одинаковые строки могут быть частично описаны в разных ДатаФреймах

Комбинирование

используем свой комбайнер

```
combiner = lambda x, y: np.where(pd.isnull(x), y, 100*x) # свой комбайнер
```

```
print df1  
print df2
```

	x	y	z
0	1	2	1
1	NaN	4	2
2	2	NaN	3

	x	y
0	20	2
1	40	4
2	NaN	20

```
print df1.combine(df2, combiner)
```

	x	y	z
0	100	200	100
1	40	400	200
2	200	20	300

Объединение ДатаФреймов

```
# объединение дата-фреймов
left = pd.DataFrame({'key': [1,2,1], 'l': [1, 2, 3]})
right = pd.DataFrame({'key': [1,2,3], 'r': [4, 5, 6]})
print left
print right
pd.merge(left, right, on='key')
```

Вертикальная конкатенация ДатаФреймов

```
a = pd.DataFrame(dict([('A',[1., 3., 2., 1.]), ('B',[2.2, 1.1, 3.3, 0.0]), ('C', 1)]))
b = pd.DataFrame(dict([('A',[0., 2.]), ('B',4)]))
```

Первый способ

`a.append(b)`

Второй способ

`pd.concat([a, b])`

	A	B	C
0	1	2.2	1
1	3	1.1	1
2	2	3.3	1
3	1	0.0	1
0	0	4.0	NaN
1	2	4.0	NaN

можно и вертикально
и с ключами

`pd.concat([a, b],
keys=['a', 'b'], axis=1)`

	a			b	
	A	B	C	A	B
0	1	2.2	1	0	4
1	3	1.1	1	2	4
2	2	3.3	1	NaN	NaN
3	1	0.0	1	NaN	NaN

Передача аргументов через список типична

Выравнивание

```
s1 = pd.Series([10,20,30,40], index = [1,2,3,4])  
s2 = pd.Series([20,30,50], index = [2,3,5])  
  
s1.align(s2)
```

1	10	1	NaN
2	20	2	20
3	30	3	30
4	40	4	NaN
5	NaN	5	50

Это такой вывод: tuple из двух серий.

```
s1.align(s2, join='inner')
```

2	20	2	20
3	30	3	30

Группировка

Функция `.groupby()` :

- 1. Разделение данных на группы (по некоторому критерию)**
- 2. Применение к каждой группе функции**
- 3. Получение результата**

Функция

- 2.1. Агрегация (статистика по группе)**
- 2.2. Трансформация (изменение/формирование значений по группе)**
- 2.3. Фильтрация (удаление некоторых групп)**

Группировка

Для каждого уникального значения A найти минимальный B

```
d = pd.DataFrame({'A': [1,2,2,1,3,3], 'B': [1,2,3,3,2,1]})  
print d
```

```
# первый способ
```

```
print d.loc[d.groupby('A')['B'].idxmin()]
```

```
# второй способ
```

```
print d.sort('B').groupby('A', as_index=False).first()
```

	A	B
0	1	1
1	2	2
2	2	3
3	1	3
4	3	2
5	3	1

	A	B
0	1	1
1	2	2
5	3	1

	A	B
0	1	1
1	2	2
5	3	1

Группировка

	A	B	C
0	1	3	5
1	2	4	5
2	2	3	5
3	1	4	6
4	1	3	6
5	2	3	6
6	2	4	6

```
print a.groupby(['A', 'B']).groups # индексы элементов групп
{(1L, 3L): [0L, 4L], (2L, 3L): [2L, 5L], (2L, 4L): [1L, 6L], (1L,
4L): [3L]}
# вывод групп
for x, y in a.groupby(['A', 'B']): # можно for (x1, x2), y in ...
    print x
    print y
(1, 3)
  A  B  C
0  1  3  5
4  1  3  6
(1, 4)
  A  B  C
3  1  4  6
(2, 3)
  A  B  C
2  2  3  5
5  2  3  6
(2, 4)
  A  B  C
1  2  4  5
6  2  4  6
```

`.groupby(, sort=True)` – сортировка результата

n

Группировка

	A	B	C
0	1	3	5
1	2	4	5
2	2	3	5
3	1	4	6
4	1	3	6
5	2	3	6
6	2	4	6

```
print a.groupby(['A','B']).first() # первые элементы
```

```
      C
A B
1 3  5
   4  6
2 3  5
   4  5
```

```
print a.groupby(['A','B'])['C'].mean() # средние по группам
```

```
      A  B
1  3  5.5
   4  6.0
2  3  5.5
   4  5.5
```

```
print a.groupby(['A','B']).get_group((1,3)) # выбор конкретной группы
```

```
      A  B  C
0  1  3  5
4  1  3  6
```

`.cumcount()` – номер в группе,

Можно группировать по столбцам...

Агрегация

	A	B	C
0	1	3	5
1	2	4	5
2	2	3	5
3	1	4	6
4	1	3	6
5	2	3	6
6	2	4	6

```
print a.groupby(['A', 'B']).aggregate(np.sum) # пример агрегации
print a.groupby(['A', 'B']).sum() # эквивалентная запись
```

C

A B

1 3 11

4 6

2 3 11

4 11

```
print a.groupby(['A', 'B']).sum().reset_index() # без индекс-и
```

A B C

0 1 3 11

1 1 4 6

2 2 3 11

3 2 4 11

```
print a.groupby(['A', 'B']).agg([np.sum, np.mean, np.std]) # ещё
```

C

sum mean std

A B

1 3 11 5.5 0.707107

4 6 6.0 NaN

2 3 11 5.5 0.707107

4 11 5.5 0.707107

Агрегация

	A	B	C
0	1	3	5
1	2	4	5
2	2	3	5
3	1	4	6
4	1	3	6
5	2	3	6
6	2	4	6

агрегация по одному столбцу

```
print a.groupby(['A', 'B'])['C'].agg({'sum': np.sum,  
    'mean': np.mean})
```

```
      sum  mean  
A  B  
1  3     11   5.5  
   4      6   6.0  
2  3     11   5.5  
   4     11   5.5
```

агрегация по разным столбцам

```
print a.groupby('A').agg({'B': np.sum, 'C': np.mean})
```

```
      C      B  
A  
1  5.666667   10  
2  5.500000   14
```

Замечание: aggregate = agg

Трансформация

	A	B	C
0	1	3	5
1	2	4	5
2	2	3	5
3	1	4	6
4	1	3	6
5	2	3	6
6	2	4	6

```
mmean = lambda x: (x-np.mean(x))  
print a.groupby('A').transform(mmean)
```

	B	C
0	-0.333333	-0.666667
1	0.500000	-0.500000
2	-0.500000	-0.500000
3	0.666667	0.333333
4	-0.333333	0.333333
5	-0.500000	0.500000
6	0.500000	0.500000

Фильтрация

	A	B	C
0	1	3	5
1	2	4	5
2	2	3	5
3	1	4	6
4	1	3	6
5	2	3	6
6	2	4	6

```
a.groupby('A').filter(lambda x: x['B'].sum()>10, dropna=False)
```

	A	B	C
0	NaN	NaN	NaN
1	2	4	5
2	2	3	5
3	NaN	NaN	NaN
4	NaN	NaN	NaN
5	2	3	6
6	2	4	6

Применение функций

`pipe()` – к ДатаФреймам
`apply()` – к строкам/столбцам
`applymap()` – поэлементно

Pipe

```
f(g(h(df), arg1=1), arg2=2, arg3=3)
```

эквивалентно

```
(df.pipe(h)
     .pipe(g, arg1=1)
     .pipe(f, arg2=2, arg3=3)
)
```

Apply

	A	B	C
0	1	3	5
1	2	4	5
2	2	3	5
3	1	4	6
4	1	3	6
5	2	3	6
6	2	4	6

```
def f(x):  
    return pd.DataFrame({'x': x, 'x-mean': x - x.mean()})  
a.groupby('A')['B'].apply(f)
```

	x	x-mean
0	3	-0.333333
1	4	0.500000
2	3	-0.500000
3	4	0.666667
4	3	-0.333333
5	3	-0.500000
6	4	0.500000

Apply

Пример нормировки

```
a.apply(lambda x: x/sum(x))  
# по столбцам
```

	A	B	C
0	0.090909	0.125000	0.128205
1	0.181818	0.166667	0.128205
2	0.181818	0.125000	0.128205
3	0.090909	0.166667	0.153846
4	0.090909	0.125000	0.153846
5	0.181818	0.125000	0.153846
6	0.181818	0.166667	0.153846

```
a.apply(lambda x: x/sum(x), axis=1)  
# по строкам
```

	A	B	C
0	0.111111	0.333333	0.555556
1	0.181818	0.363636	0.454545
2	0.200000	0.300000	0.500000
3	0.090909	0.363636	0.545455
4	0.100000	0.300000	0.600000
5	0.181818	0.272727	0.545455
6	0.166667	0.333333	0.500000

Applymap

applymap - поэлементно

```
a = pd.DataFrame({'A': [1,2,2], 'B': ['a','b','a']})
```

```
def some_fn(x):  
    if type(x) is str:  
        return 'applymap_' + x  
    else:  
        return (10*x)
```

```
a.applymap(some_fn)
```

	A	B
0	10	applymap_a
1	20	applymap_b
2	20	applymap_a

Map

все строки, в которых столбец начинается с определённой буквы

```
d = pd.DataFrame({'A': [1,2,2,1,2,3,2,1,3],  
                  'B': ['as', 'bs', 'e', 'qq', 'aaa', 'a', 'e', 'qwr', 'www']})
```

```
d[d['B'].map(lambda x: x.startswith('a'))]
```

	A	B
0	1	as
4	2	aaa
5	3	a

```
df = pd.DataFrame({'name': [u'Маша', u'Саша', u'Рудольф'],  
                  'marks': [[2,3,3,5], [4,5,5], [2,3]]})  
print df[df['marks'].map(lambda x: 3 in x)]
```

	marks	name
0	[2, 3, 3, 5]	Маша
2	[2, 3]	Рудольф

Map – основное применение

```
df = pd.DataFrame({'CITY': [u'London', u'Moscow', u'Paris'], 'Stats': [0,2,1]})

d = {u'London':u'GB', u'Moscow':u'RUS', u'Paris':u'FR'}
df['country'] = df['CITY'].map(d)
df.columns = map(str.lower, df.columns)
df
```

	city	stats	country
0	London	0	GB
1	Moscow	2	RUS
2	Paris	1	FR

Иногда есть другие средства – замена значений

```
df.replace(u'Moscow', u'Ufa') # замена значения
```

Иерархическая (многоуровневая) индексация

```
tuples = list(zip(*[['bar', 'bar', 'baz', 'baz', 'foo', 'foo', 'qux', 'qux'],
                    ['one', 'two', 'one', 'two', 'one', 'two', 'one', 'two'])))
print tuples
[('bar', 'one'), ('bar', 'two'), ('baz', 'one'), ('baz', 'two'),
 ('foo', 'one'), ('foo', 'two'), ('qux', 'one'), ('qux', 'two')]

index = pd.MultiIndex.from_tuples(tuples, names=['first', 'second'])
df = pd.DataFrame(np.random.randn(8, 2), index=index, columns=['A', 'B'])
print df
```

		A	B
first	second		
bar	one	-0.240469	-0.533312
	two	-0.847305	0.845316
baz	one	0.274592	0.473476
	two	1.433575	-0.977992
foo	one	0.957252	-1.246396
	two	-2.821039	-0.625924
qux	one	0.086683	-0.450850
	two	-1.236494	0.706156

Иерархическая (многоуровневая) индексация

```
print df.stack() # обратная операция unstack()
```

```
first second  
bar one A -0.240469  
      B -0.533312  
     two A -0.847305  
        B  0.845316  
baz one A  0.274592  
      B  0.473476  
     two A  1.433575  
        B -0.977992  
foo one A  0.957252  
      B -1.246396  
     two A -2.821039  
        B -0.625924  
qux one A  0.086683  
      B -0.450850  
     two A -1.236494  
        B  0.706156  
dtype: float64
```

Pivot tables

```
df = pd.DataFrame({'ind1': [1, 1, 1, 2, 2, 2, 2], 'ind2': [1, 1, 2, 2, 3, 3, 2],  
                  'x': [1, 2, 3, 4, 5, 6, 7], 'y': [1, 1, 1, 1, 1, 1, 2]})  
  
print df  
print df.pivot(index='x', columns='ind2', values='y')
```

	ind1	ind2	x	y
0	1	1	1	1
1	1	1	2	1
2	1	2	3	1
3	2	2	4	1
4	2	3	5	1
5	2	3	6	1
6	2	2	7	2

	ind2	1	2	3
	x			
1	1	NaN	NaN	
2	1	NaN	NaN	
3	NaN	1	NaN	
4	NaN	1	NaN	
5	NaN	NaN	1	
6	NaN	NaN	1	
7	NaN	2	NaN	

Pivot tables

```
dfp = df.pivot_table(index=['ind1', 'ind2'], aggfunc='sum')
dfp
```

		x	y
ind1	ind2		
1	1	3	2
	2	3	1
2	2	11	3
	3	11	2

```
# отличается от
df.set_index(['ind1', 'ind2'], drop=False)
```

```
ind1 ind2
1      1      1      1      1      1
      1      1      2      1
      2      1      2      3      1
2      2      2      2      4      1
      3      2      3      5      1
      3      2      3      6      1
      2      2      2      7      2
```


Pivot tables

```
print dfp
```

```
print dfp.sum(level='ind2')
```

```
dfp.swaplevel('ind1','ind2')
```

		x	y
ind1	ind2		
1	1	3	2
	2	3	1
2	2	11	3
	3	11	2

	x	y
ind2		
1	3	2
2	14	4
3	11	2

		x	y
ind2	ind1		
1	1	3	2
	2	11	3
2	1	3	1
	2	11	2
3	2	11	2

Melt

Операция, в некотором смысле обратная Pivot

```
cheese = pd.DataFrame({'first' : ['John', 'Mary'], 'last' :  
['Doe', 'Bo'], 'height' : [5.5, 6.0], 'weight' : [130, 150]})  
  
pd.melt(cheese, id_vars=['first', 'last'])
```

	first	height	last	weight
0	John	5.5	Doe	130
1	Mary	6.0	Bo	150

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130.0
3	Mary	Bo	weight	150.0

Делаем из «широкой» таблицы «высокую».

Категориальные признаки

создание категориального признака = интервалы попаданий

```
x = np.random.randn(10000)
```

```
y = pd.cut(x, 10)
```

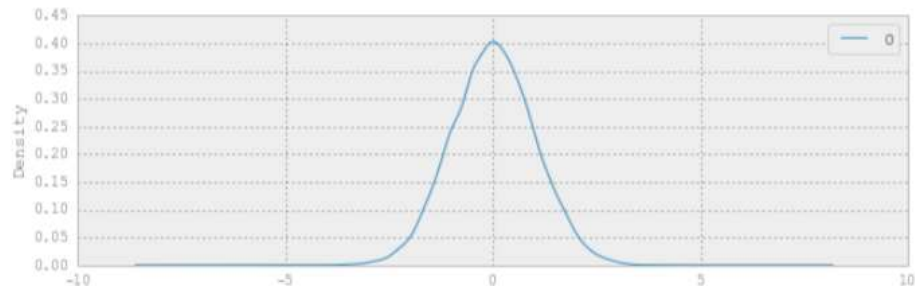
```
z = pd.value_counts(y)
```

```
z.plot(figsize=(20, 3))
```

```
pd.DataFrame(x).plot(kind='kde')
```

```
pd.DataFrame(z).T
```

	(-0.211, 0.629]	(-1.0502, -0.211]	(0.629, 1.468]	(-1.89, -1.0502]	(1.468, 2.308]	(-2.729, -1.89]	(2.308, 3.147]	(-3.569, -2.729]	(3.147, 3.987]	(-4.417, -3.569]
0	3247	2638	1921	1192	634	226	102	32	6	2



Несколько колонок как функция одной

```
a = pd.DataFrame({'a': [1,2,1,2], 'b': [3,3,3,4]})
```

```
def two_three_strings(x):  
    return x*2, x*3
```

```
a['twice'], a['thrice'] = zip(*a['a'].map(two_three_strings))  
a
```

	a	b	twice	thrice
0	1	3	2	3
1	2	3	4	6
2	1	3	2	3
3	2	4	4	6

```
0      (2, 3)  
1      (4, 6)  
2      (2, 3)  
3      (4, 6)
```

Одна колонка как функция нескольких

Из имени и фамилии делаем полное имя

```
df = pd.DataFrame({'name': [u'Маша', u'Саша', u'Рудольф'],  
                  'surname': [u'Петрова', u'Сидоров', u'Кац']})
```

первый способ

```
lst = []  
for n, s in zip(df.name, df.surname):  
    lst.append(n + ' ' + s)  
df['fullname'] = lst
```

второй способ

```
df['fullname2'] = df[['name', 'surname']].apply(lambda x: x[0] + ' ' +  
x[1], axis=1)
```

самый простой способ

```
df['fullname3'] = df['name'] + ' ' + df['surname']
```

	name	surname	fullname	fullname2	fullname3
0	Маша	Петрова	Маша Петрова	Маша Петрова	Маша Петрова
1	Саша	Сидоров	Саша Сидоров	Саша Сидоров	Саша Сидоров
2	Рудольф	Кац	Рудольф Кац	Рудольф Кац	Рудольф Кац

Временные ряды

#временные ряды

```
data = {'date': ['2014-05-01 18:47:05.069722', '2014-05-01 18:47:05.119994', '2014-05-02  
18:47:05.178768', '2014-05-02 18:47:05.230071', '2014-05-02 18:47:05.230071', '2014-05-02  
18:47:05.280592', '2014-05-03 18:47:05.332662', '2014-05-03 18:47:05.385109', '2014-05-04  
18:47:05.436523', '2014-05-04 18:47:05.486877'],  
'battle_deaths': [34, 25, 26, 15, 15, 14, 26, 25, 62, 41]}
```

```
df = pd.DataFrame(data)  
df.index = pd.to_datetime(df['date'])  
del df['date']  
df['05-2014']
```

	battle_deaths
date	
2014-05-01 18:47:05.069722	34
2014-05-01 18:47:05.119994	25
2014-05-02 18:47:05.178768	26
2014-05-02 18:47:05.230071	15
2014-05-02 18:47:05.230071	15
2014-05-02 18:47:05.280592	14
2014-05-03 18:47:05.332662	26
2014-05-03 18:47:05.385109	25
2014-05-04 18:47:05.436523	62
2014-05-04 18:47:05.486877	41

Временные ряды

Индексация

```
print df['5/1/2014'] # df['2014-05-01']
                        battle_deaths
```

date

2014-05-01	18:47:05.069722	34
2014-05-01	18:47:05.119994	25

```
print df['2014-05-03':'2014-05-04']
                        battle_deaths
```

date

2014-05-03	18:47:05.332662	26
2014-05-03	18:47:05.385109	25
2014-05-04	18:47:05.436523	62
2014-05-04	18:47:05.486877	41

Временные ряды

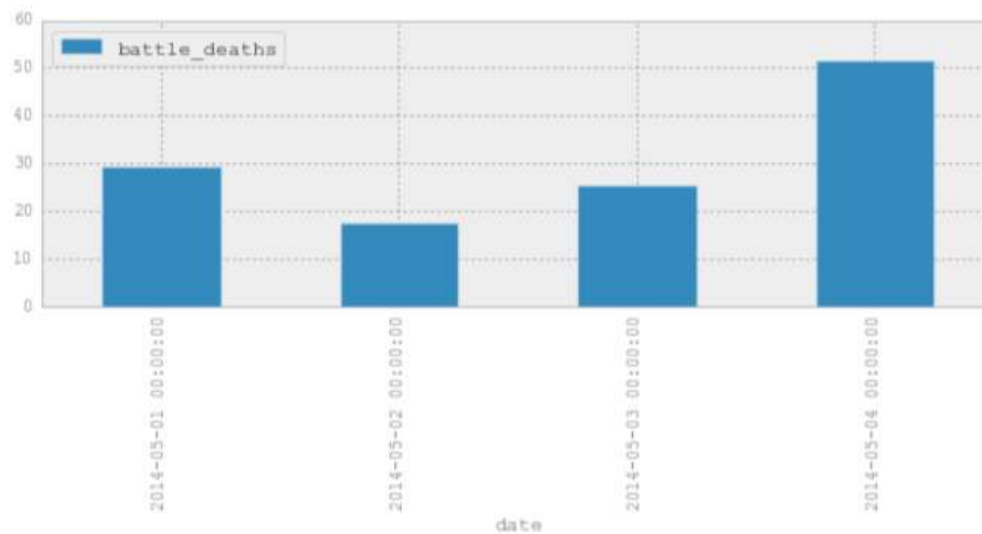
переход к дням и визуализация

```
print df.resample('D', how='mean').plot(kind='bar')
```

пересортировка df

```
df.sort_index(by = 'battle_deaths', inplace=True)
```

df



	battle_deaths
date	
2014-05-02 18:47:05.280592	14
2014-05-02 18:47:05.230071	15
2014-05-02 18:47:05.230071	15
2014-05-01 18:47:05.119994	25
2014-05-03 18:47:05.385109	25
2014-05-02 18:47:05.178768	26
2014-05-03 18:47:05.332662	26
2014-05-01 18:47:05.069722	34
2014-05-04 18:47:05.486877	41
2014-05-04 18:47:05.436523	62

Строки

```
s = pd.Series(['AbA', 'Sasha', 'DataMining'])
s.str.lower()
```

```
0      aba
1     sasha
2  datamining
```

```
df = pd.DataFrame({'name': [u'Маша', u'Саша', u'Рудольф'],
                    'mail': ['1@mail.ru', 'Amail@vk.ru', '12_Wq@ru.ru']})
print df.mail.str.contains('mail')
```

```
0      True
1      True
2     False
```

```
pattern = '([A-Z0-9._%+-]+)@([A-Z0-9.-]+\.[A-Z]{2,4})'
import re as re
df['mail'].str.match(pattern, flags=re.IGNORECASE)
0      (1, mail, ru)
1      (Amail, vk, ru)
2      (12 Wq, ru, ru)
```

Строки

Пример возможного извлечения признаков

```
lst = ['mark 10 12-10-2015', 'also 7 10-10-2014', 'take 2 01-05-2015']  
df = pd.DataFrame({'x':lst})  
df['num'] = df.x.str.extract('(\d+)')  
df['date'] = df.x.str.extract('(..-..-....)')  
df['word'] = df.x.str.extract('([a-z]\w{0,})')  
df
```

	x	num	date	word
0	mark 10 12-10-2015	10	12-10-2015	mark
1	also 7 10-10-2014	7	10-10-2014	also
2	take 2 01-05-2015	2	01-05-2015	take

Как часто встречаются пары значений

очень полезная штука!

```
d = pd.DataFrame({'A': [1,2,2,1,2,3,2,1,3],  
                  'B': [1,2,3,4,1,2,3,3,4]})
```

```
pd.crosstab(d['A'], d['B'])
```

B	1	2	3	4
A				
1	1	0	1	1
2	1	1	2	0
3	0	1	0	1

Другие возможности

Скользящее среднее

```
a = pd.DataFrame({'x': [1, 2, 3, 1, 2, 3, 1, 2, 3], 'y': [2, 2, 10, 2, 2, 2, 2, 2, 2]})
```

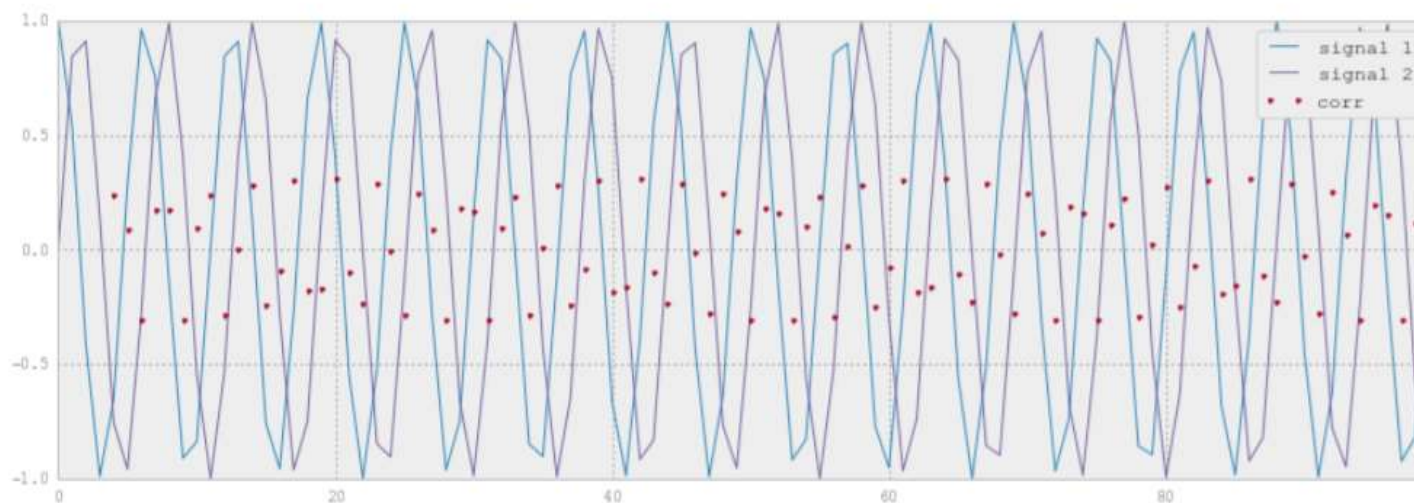
```
pd.rolling_mean(a, 2) # можно сумму, медиану и любую функцию!
```

	x	y
0	NaN	NaN
1	1.5	2
2	2.5	6
3	2.0	6
4	1.5	2
5	2.5	2
6	2.0	2
7	1.5	2
8	2.5	2

Другие возможности

Корреляция

```
s1 = pd.Series(np.cos(np.arange(100)))  
s2 = pd.Series(np.sin(np.arange(100)))  
f = pd.DataFrame({'s1':s1, 's2':s2}).plot()  
pd.rolling_corr(s1, s2, window=5).plot(style='.')  
f.legend(['signal 1', 'signal 2', 'corr'])
```

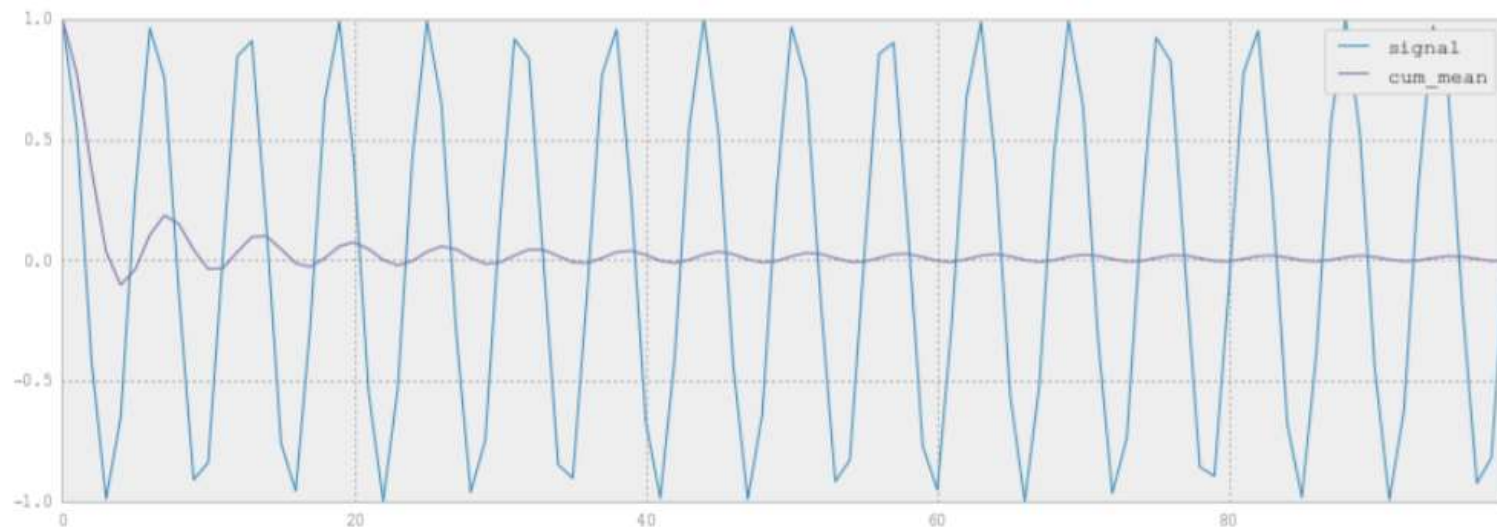


.corr() – корреляция (колонок)

Другие возможности

Кумулятивные функции

```
# кумулятивное усреднение  
s1 = pd.Series(np.cos(np.arange(100)))  
s2 = pd.expanding_mean(s1)  
f = pd.DataFrame({'s1':s1, 's2':s2}).plot()  
f.legend(['signal', 'cum_mean'])
```



Другие возможности

Удаление дубликатов

```
df = pd.DataFrame({'name': ['Al', 'Max', 'Al'],  
                  'surname': [u'Run', u'Crone', u'Run']})  
print df.duplicated()
```

```
df.drop_duplicates(['name'], take_last=True)  
# df.drop_duplicates()
```

```
0    False  
1    False  
2     True
```

	name	surname
1	Max	Crone
2	Al	Run

Другие возможности

Максимальные элементы

```
df = pd.DataFrame({'x':[12, 10, 54, 10], 'y':[2, 4, 1, 4]})  
print df.rank(method='average') # номера по возрастанию  
print df.idxmax() # индексы максимальных элементов
```

	x	y
0	3.0	2.0
1	1.5	3.5
2	4.0	1.0
3	1.5	3.5

x 2
y 1

n наименьших

```
s = pd.Series([3, 2, 6, 5, 1, 4])  
s.nsmallest(3)
```

4	1
1	2
0	3

Другие возможности

сколько каждого значения

```
print pd.value_counts([3,2,2,2,5,3], sort=False)
```

```
2    3
```

```
3    2
```

```
5    1
```

мода

```
print pd.DataFrame({'x': [3,2,2,2,5,3,3]}).mode()
```

```
   x
```

```
0  2
```

```
1  3
```

dummy-кодирование для категориальных признаков

```
pd.get_dummies([1,2,1,2,3])
```

	1	2	3
0	1	0	0
1	0	1	0
2	1	0	0
3	0	1	0
4	0	0	1

И так можно!

```
pd.Series(['one,two', 'two,three', 'one!']).str.get_dummies(sep=',')
```

	one	one!	three	two
0	1	0	0	1
1	0	0	1	1
2	0	1	0	0

Кодирование категориальных признаков по порядку

```
pd.factorize([20,10,np.nan,10,np.nan,30,20])  
  
(array([ 0,  1, -1,  1, -1,  2,  0]), array([ 20.,  10.,  30.]))
```

Exercises

- Describe Obama and McCain
- Merge Obama and McCain
- Group by city Obama or McCain and describe()