Pandas

Николай Матхеев
nikolay@stego.su

Москва
07.10.2016

Немного лирики

- Изначально Python не обладал библиотеками для анализа данных – все использовали R
- Wes McKinney решил исправить положение и начал работу над Pandas в 2007 году
- Работа продолжается до сих пор, многое еще надо реализовать
- Название pandas не связано с пандами.
- Акроним от “panel data” – термин для многомерных массивов данных

Великодушный пожизненный диктатор Pandas

Основные объекты в Pandas

- Series (1D)

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

```
a   -2.091564
b    0.135039
c   -0.065467
d   -1.169783
e   -1.609569
dtype: float64
```

Выравнивание по индексу:

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

```
a         NaN
b    0.270077
c   -0.130934
d   -2.339566
e   -3.219137
dtype: float64
```

Основные объекты в Pandas

- DataFrame(2D)

```
df = pd.DataFrame(np.random.randn(8,3),
                  index=pd.date_range('1/1/2011', periods=8),
                  columns=['A','B','C'])
```

```
df
```

	A	B	C
2011-01-01	0.352593	0.223594	-0.854430
2011-01-02	-0.922488	-0.348662	0.117223
2011-01-03	-1.148719	-0.348253	1.159168
2011-01-04	-0.873848	-0.006430	-0.436925
2011-01-05	0.296080	0.097156	0.560831
2011-01-06	0.494053	0.711460	-0.233050
2011-01-07	-0.641131	-1.570160	0.288397
2011-01-08	-0.387903	-1.297635	-0.179522

Основные объекты в Pandas

- Panel(3D)

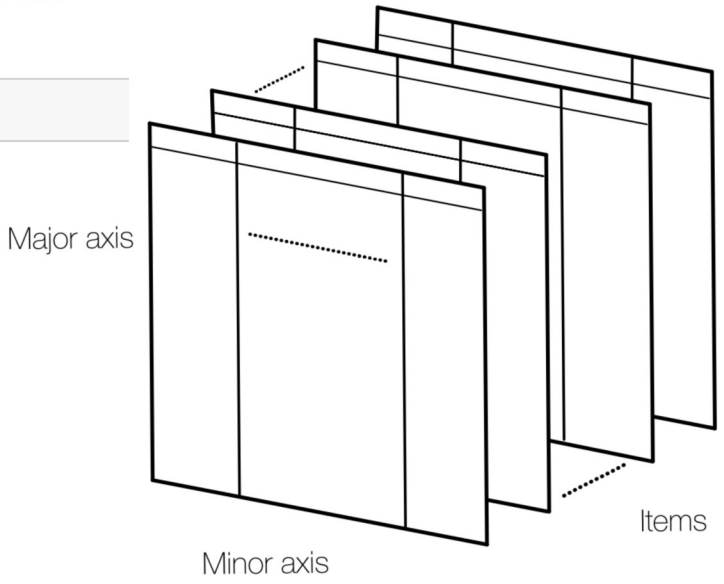
```
wp = pd.Panel(np.random.randn(2,5,4),
              items=['Item1', 'Item2'],
              major_axis=pd.date_range('1/1/2000', periods=5),
              minor_axis=['A','B','C','D']
             )
```

```
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
```

```
wp.ix[0].head()
```

	A	B	C	D
2000-01-01	0.192770	2.452199	1.255622	-0.415262
2000-01-02	0.001139	-0.694615	1.296104	1.136564
2000-01-03	1.333120	-0.972151	-1.431222	1.231503
2000-01-04	-1.346358	-0.002258	1.324954	-0.064672
2000-01-05	-2.439119	-0.866038	0.837556	-0.071488



Индекс и мультииндекс

0
1
2
3
4
5
6
7
8
9
10
11

x	0
x	1
x	2
x	3
y	4
y	5
y	6
y	7
z	8
z	9
z	10
z	11

Работа с внешними данными

Семейство функций: pandas.read_xxx

Семейство методов класса: DataFrame.to_xxx.

xxx – формат данных. Работают на 2D-уровне

- CSV

```
df2 = pd.read_csv('ДЗ_01/выгрузка.csv', sep=';')
df2.to_csv('data.csv')
```

- Excel

```
data2 = pd.read_excel(filename, sheetname='1')
```

- HDF5
- таблицы в СУБД
- JSON
- Буфер обмена
- HTML/XML

Жизнь после загрузки данных

Посмотрим на данные:

- head(), tail(), info()

```
df2.head(1)
```

	EVENT_TIME	USER_HASH
0	2015-12-15 01:09:38	7215be4441716d2f96d932ecf20e324145933912

1 rows × 31 columns

```
df2.tail(1)
```

	EVENT_TIME	USER_HASH
29303	2016-01-12 23:05:34	f1c4e08219c20f829b9ff07656fb39a0186727f

Приведение данных к типам – семейство функций to_xxx.

xxx-тип данных()

```
df2.X2 = pd.to_datetime(df2.X2)
```

Посмотреть столбцы:

```
df2.columns
```

```
Index(['EVENT_TIME', 'USER_HASH', 'EVENT_TYPE', 'EVENT_TYPE_EX', 'AMOUNT',
       'X2', 'X3', 'X4', 'X5', 'X6', 'X9', 'X10', 'X11', 'X12', 'X13', 'X14',
       'X15', 'X16', 'X17', 'X18', 'X22', 'X23', 'X24', 'X26', 'COOKIE',
       'p1_Fraud', 'p2_Fraud', 'p3_Fraud', 'p4_Fraud', 'p5_Fraud', 'CLASS'],
      dtype='object')
```

Индекс:

```
df2.index
```

```
RangeIndex(start=0, stop=29304, step=1)
```

Жизнь после загрузки данных

Узнать типы данных столбцов

```
df2.dtypes
```

```
EVENT_TIME          object
USER_HASH           object
EVENT_TYPE          object
```

Сортировка данных по столбцу

```
df2.sort_values(by='AMOUNT')
```

Транформирование в NumPy 2d-массив

```
type(df2.values)
```

```
numpy.ndarray
```

Статистика
по датафрейму:

```
df2.describe().T
```

```
/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-packages/numpy
eWarning: Invalid value encountered in percentile
  RuntimeWarning)
```

	count	mean	std	min	25%	50%	75%	max
X9	29304.0	7.643216e+01	3.184550e+02	-1440.0000	0.0000	0.000	0.000000	1.585000e+03
X13	29304.0	2.308613e+02	4.951268e+02	-502.0000	0.0000	0.000	0.000000	1.991000e+03

Число
уникальных
значений

```
for col in df2.columns:
    print(col, ' have {0} unique values'.format(df2[col].nunique()))
```

```
EVENT_TIME  have 28884 unique values
USER_HASH  have 7608 unique values
EVENT_TYPE  have 2 unique values
```

Переименование и удаление колонок

```
df2.rename(columns={'EVENT_TIME':'MEGAEVENT'}, inplace=False)
```

	MEGAEVENT	USER_HASH	EVENT_TYPE

```
df2.drop(1,axis=0, inplace=True)
```

```
del df['AMOUNT']
df.drop('AMOUNT', axis=0)
```

axis=0 – ось “строк”
axis=1 – ось “столбцов”

Выборка данных и индексация

.loc

- Выборка по меткам:
  - одна метка
  - список меток
  - диапазон меток
  - callable (какая-нибудь функция – доступно с 0.18.1)

```
In [41]: df1.loc['20130102':'20130104']
Out[41]:
                   A         B         C         D
2013-01-02  0.357021 -0.674600 -1.776904 -0.968914
```

```
In [88]: df1.iloc[:, lambda df: [0, 1]]
Out[88]:
          A         B
a -0.023688  2.410179
b -0.251905 -2.213588
```

.iloc

- Выборка по позиции:
  - целое число
  - список int'ов
  - срез (1:7 к примеру)
  - callable (какая-нибудь функция – доступно с 0.18.1)

.ix

- Смешанный способ (полезен при наличии мультииндекса)

Выборка данных и индексация

.query(numexpr, *args)

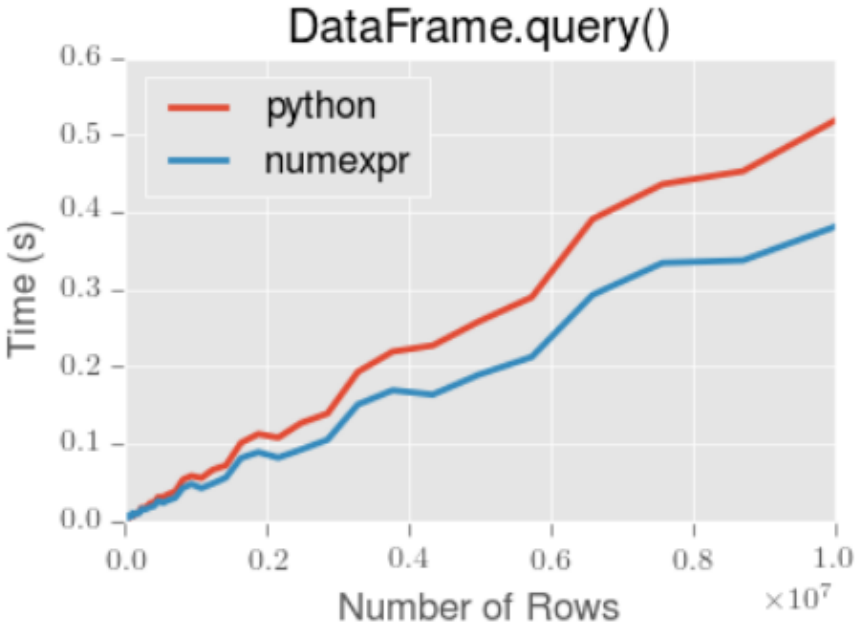
```
dff = pd.DataFrame(np.random.rand(10, 3), columns=list('abc'))
```

```
dff.query('(a < b) & (b < c)')
```

	a	b	c
0	0.195293	0.369608	0.529688

Есть целесообразность использования:
1) выразительность условия выбоорки
2) на больших объемах **query** производительней (за счет распараллеливания)

Но нужно иметь установленный numexpr (входит в anaconda)



Выборка данных и индексация

Переиндексация

```
sss = pd.Series([10,20,60], index=[1,2,6])
```

```
sss.reindex(index=[1,2,3,4,5,6], method='ffill')
```

```
1    10
2    20
3    20
4    20
5    20
6    60
```

Итерация:

Семейство методов iterXXX:

- iterrows (по строкам)
- itertuples (по строкам)
- iteritems (по столбцам)

```
df
```

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

```
for index, rows in df.iteritems():
    print(index, rows)
```

```
x a    1
b    2
c    1
d    2
Name: x, dtype: int64
y a    1
b    2
c    3
d    3
```

Сравнения

```
df1 > df2
```

	x	y
0	False	False
1	True	False
2	False	False

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

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

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

```
x    True
y    True
dtype: bool
```

Результат сравнения – датафрейм с булевыми значениями

NaN (not a number)

```
df1
```

	A	B
0	1.0	2.0
1	NaN	NaN
2	2.0	1.0

```
df1.isnull()
```

	A	B
0	False	False
1	True	True
2	False	False

```
df1.mean()
```

```
A    1.5
B    1.5
dtype: float64
```

Некоторые функции
игнорируют NaN-значения

```
df1.apply(np.cumsum)
```

	A	B
0	1.0	2.0
1	NaN	NaN
2	3.0	3.0

NaN (not a number)

Что же делать с NaN?

- dropna() – удаление строк датафрейма, в которых есть NaN-значения
- fillna() – заполнить NaN-ячейки указанным параметром
- ffill() – заполнение соседними значениями (доступны разные стратегии заполнения)

```
df1.dropna()
```

	A	B
0	1.0	2.0
2	2.0	1.0

```
df1.fillna(0)
```

	A	B
0	1.0	2.0
1	0.0	0.0
2	2.0	1.0

```
df1.fillna(method='ffill')
```

	A	B
0	1.0	2.0
1	1.0	2.0
2	2.0	1.0

NaN (not a number)

Как бороться с ним:

Интерполяция – interpolate(method, *args)

```
In [60]: df = DataFrame({'A': [1, 2.1, np.nan, 4.7, 5.6, 6.8],
   ....:                 'B': [.25, np.nan, np.nan, 4, 12.2, 14.4]})
   ....:

In [61]: df
Out[61]:
     A      B
0  1.0   0.25
1  2.1    NaN
2  NaN    NaN
3  4.7   4.00
4  5.6  12.20
5  6.8  14.40

In [62]: df.interpolate()
Out[62]:
     A      B
0  1.0   0.25
1  2.1   1.50
2  3.4   2.75
3  4.7   4.00
4  5.6  12.20
5  6.8  14.40
```

apply и applymap:

- .apply(func, *args)

Применяет func по всем столбцам/строкам датафрейма (axis)

- .applymap(func, *args)

Применяет func по всем элементам датафрейма (axis)

	b	d	e
Utah	-0.222829	1.431772	-1.667800
Ohio	0.337446	-1.205942	0.894948
Texas	-1.248979	0.350620	1.099477
Oregon	0.510153	-0.137684	1.015742

```
frame = pd.DataFrame(np.random.randn(4, 3),
                     columns=list('bde'),
index=['Utah', 'Ohio', 'Texas', 'Oregon'])
```

```
frame.apply(lambda x: x.max() - x.min(),
            axis=1,)
```

```
Utah      3.099572
Ohio      2.100890
Texas     2.348457
Oregon    1.153425
```

Склейка датафреймов:

Конкатенация

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                   index=[0, 1, 2, 3])


df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                     index=[4, 5, 6, 7])


df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
                   index=[8, 9, 10, 11])


frames = [df1, df2, df3]

result = pd.concat(frames)
```

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

df3

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Result

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Группировка - .groupby()

План действий

- Разделяем данные на группы (по некоторому критерию)
- Применяем к каждой группе некую *функцию*
- Получаем результат

Функция:

- Агрегация (статистика по группе)
- Трансформация (изменение/формирование значений по группе)
- Фильтрация (удаление некоторых групп)

Группировка

Для каждого уникального A найти минимальный B:

```
d.sort_values('B').\
groupby('A', as_index=False).\
first()
```

```
d.sort_values('B').\
groupby('A').groups
```

```
{1: [0, 3], 2: [1, 2], 3: [5, 4]}
```

	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
5	3	1
1	2	2
4	3	2
2	2	3
3	1	3

=>

	A	B
0	1	1
1	2	2
2	3	1

Агрегация

.aggregate(*func, *args)

```
d.groupby('A').aggregate(np.sum)
```

	B
A	
1	4
2	5
3	3

```
d.groupby('A').aggregate([np.sum, np.mean, np.std])
```

	B		
	sum	**mean**	**std**
A			
1	4	2.0	1.414214
2	5	2.5	0.707107
3	3	1.5	0.707107

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

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

	A	B	C
0	NaN	NaN	NaN
1	2.0	4.0	5.0
2	2.0	3.0	5.0
3	NaN	NaN	NaN
4	NaN	NaN	NaN
5	2.0	3.0	6.0
6	2.0	4.0	6.0

Строки

Функции доступны через атрибут датафрейма **str**

```
df
```

	mail	name
0	bunny666@mail.ru	Даша
1	swaglover@ya.ru	Саша
2	dontscrew@sts.su	Маша

```
import re
df.mail.str.match('([\w\d]+)@(\w+)\\.(\w+)')
```

```
/Library/Frameworks/Python.framework/Versions/
 In future versions of pandas, match will cha
  from ipykernel import kernelapp as app
```

```
0    (bunny666, mail, ru)
1     (swaglover, ya, ru)
2    (dontscrew, sts, su)
Name: mail, dtype: object
```

```
df.mail.str.replace('bunny666@mail.ru', 'nomail')
```

```
0               nomail
1     swaglover@ya.ru
2    dontscrew@sts.su
Name: mail, dtype: object
```

```
pd.Series(['a1', 'b2', 'c3']).\
str.extract('(?P<letter>[ab])(?P<digit>\d)', expand=False)
```

	letter	digit
0	a	1
1	b	2
2	NaN	NaN

Работа на дом:

- Пролистать туториалы:

http://pandas.pydata.org/pandas-docs/version/0.18.1/tutorials.html