Иерархическая индексация

https://dfedorov.spb.ru

Иерархическая индексация

До сих пор мы рассматривали главным образом одномерные и двумерные данные, находящиеся в объектах Series и DataFrame библиотеки Pandas. Часто бывает удобно выйти за пределы двух измерений и хранить многомерные данные, то есть данные, индексированные по более чем двум ключам. На практике используется **иерархическая индексация** (hierarchical indexing), или **мультииндексация** (multi-indexing), для включения в один индекс нескольких уровней.

Пускай нам требуется проанализировать данные о штатах за два разных года. Может показаться соблазнительным, воспользовавшись утилитами библиотеки Pandas, применить в качестве ключей кортежи языка Python:

```
>>> index = [('California', 2000), ('California', 2010),
             ('New York', 2000), ('New York', 2010),
             ('Texas', 2000), ('Texas', 2010)]

>>> populations = [33871648, 37253956, 18976457, 19378102, 20851820, 25145561]
>>> pop = pd.Series(populations, index=index)
>>> pop
```

```
(California, 2000)    33871648
(California, 2010)    37253956
(New York, 2000)      18976457
(New York, 2010)      19378102
(Texas, 2000)         20851820
(Texas, 2010)         25145561
dtype: int64
```

```
>>> pop[('California', 2010):('Texas', 2000)]
```

```
(California, 2010)    37253956
(New York, 2000)      18976457
(New York, 2010)      19378102
(Texas, 2000)         20851820
dtype: int64
```

Например, при необходимости **выбрать все значения из 2010 года** придется проделать громоздкую (и потенциально медленную) очистку данных:

```
>>> pop[[i for i in pop.index if i[1] == 2010]]

(California, 2010)    37253956
(New York, 2010)      19378102
(Texas, 2010)         25145561
dtype: int64
```

В библиотеке Pandas есть лучший способ выполнения таких операций. Наша индексация, основанная на кортежах, по сути, является примитивным мультииндексом (**MultiIndex**).

```
>>> index = pd.MultiIndex.from_tuples(index)
>>> index

MultiIndex([('California', 2000),
            ('California', 2010),
            (  'New York', 2000),
            (  'New York', 2010),
            (     'Texas', 2000),
            (     'Texas', 2010)],
           )
```

Обратите внимание, что MultiIndex содержит несколько уровней (**levels**) индексации. В данном случае названия штатов и годы, а также несколько кодирующих эти уровни меток (**labels**) для каждой точки данных.

```
>>> pop = pop.reindex(index)
>>> pop

California  2000    33871648
            2010    37253956
New York    2000    18976457
            2010    19378102
Texas       2000    20851820
            2010    25145561
dtype: int64
```

Иерархическое представление данных. В этом мультииндексном представлении все пропущенные элементы означают то же значение, что и строкой выше.

```
>>> pop[:, 2010]

California    37253956
New York      19378102
Texas         25145561
dtype: int64
```

Теперь для выбора всех данных, второй индекс которых равен 2010, можно просто воспользоваться синтаксисом срезов библиотеки Pandas.

Мультииндекс как дополнительное измерение

Мы могли с легкостью хранить те же самые данные с помощью простого объекта DataFrame с индексом и метками столбцов. На самом деле библиотека Pandas создана с учетом этой равнозначности.

```
>>> pop_df = pop.unstack()
>>> pop_df
```

	2000	2010
California	33871648	37253956
New York	18976457	19378102
Texas	20851820	25145561

Метод **unstack()** может быстро преобразовать мультииндексный объект Series в индексированный обычным образом объект DataFrame.

```
>>> pop_df.stack()
```

```
California  2000    33871648
            2010    37253956
New York    2000    18976457
            2010    19378102
Texas       2000    20851820
            2010    25145561
dtype: int64
```

Почему вообще имеет смысл возиться с иерархической индексацией?
Причина проста: аналогично тому, как мы использовали мультиндексацию для представления двумерных данных в одномерном объекте Series, можно использовать ее для представления данных с тремя или более измерениями в объектах Series или DataFrame.

Каждый новый уровень в мультиндексе представляет дополнительное измерение данных. Благодаря использованию этого свойства мы получаем намного больше свободы в представлении типов данных.

Например, нам может понадобиться добавить в демографические данные по каждому штату за каждый год еще один столбец (допустим, количество населения младше 18 лет). Благодаря типу MultiIndex это сводится к добавлению еще одного столбца в объект DataFrame.

```
>>> pop_df = pd.DataFrame({'total': pop,
                           'under18': [9267089, 9284094,
                                       4687374, 4318033,
                                       5906301, 6879014]})
>>> pop_df
```

		total	under18
California	**2000**	33871648	9267089
	2010	37253956	9284094
New York	**2000**	18976457	4687374
	2010	19378102	4318033
Texas	**2000**	20851820	5906301
	2010	25145561	6879014

Все универсальные функции и остальная функциональность, Pandas также прекрасно работают с иерархическими индексами.

Вычисляем по годам долю населения младше 18 лет на основе вышеприведенных данных.

```
>>> f_u18 = pop_df['under18'] / pop_df['total']
>>> f_u18.unstack()
```

	2000	2010
California	0.273594	0.249211
New York	0.247010	0.222831
Texas	0.283251	0.273568

Методы создания мультииндексов

Наиболее простой метод создания мультииндексированного объекта Series или DataFrame — передать в конструктор список из двух или более индексных массивов.

```
>>> df = pd.DataFrame(np.random.rand(4, 2),
                      index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],
                      columns=['data1', 'data2'])
>>> df
```

		data1	data2
a	1	0.453804	0.670941
	2	0.891329	0.927398
b	1	0.301705	0.686508
	2	0.267348	0.589724

Если передать словарь с соответствующими кортежами в качестве ключей, Pandas распознает такой синтаксис и будет по умолчанию использовать мультииндекс:

```
>>> data = {('California', 2000): 33871648,
            ('California', 2010): 37253956,
            ('Texas', 2000): 20851820,
            ('Texas', 2010): 25145561,
            ('New York', 2000): 18976457,
            ('New York', 2010): 19378102}

>>> pd.Series(data)
```

```
California  2000    33871648
            2010    37253956
Texas       2000    20851820
            2010    25145561
New York    2000    18976457
            2010    19378102
dtype: int64
```

Явные конструкторы для объектов MultiIndex

При формировании индекса для большей гибкости можно воспользоваться имеющимися в классе **pd.MultiIndex** конструкторами-методами класса.

Например, можно сформировать объект MultiIndex из простого списка массивов, задающих значения индекса в каждом из уровней.

```
>>> pd.MultiIndex.from_arrays([['a', 'a', 'b', 'b'], [1, 2, 1, 2]])

MultiIndex([('a', 1),
            ('a', 2),
            ('b', 1),
            ('b', 2)],
           )
```

Или из списка кортежей, задающих все значения индекса в каждой из точек:

```
>>> pd.MultiIndex.from_tuples([('a', 1), ('a', 2), ('b', 1), ('b', 2)])

MultiIndex([('a', 1),
            ('a', 2),
            ('b', 1),
            ('b', 2)],
           )
```

Или из декартова произведения обычных индексов:

```
>>> pd.MultiIndex.from_product([['a', 'b'], [1, 2]])

MultiIndex([('a', 1),
            ('a', 2),
            ('b', 1),
            ('b', 2)],
           )
```

Названия уровней мультииндексов

Иногда бывает удобно задать названия для уровней объекта MultiIndex.

Сделать это можно, передав аргумент **names** любому из вышеперечисленных конструкторов класса MultiIndex или задав значения атрибута **names** постфактум:

```
>>> pop.index.names = ['state', 'year']
>>> pop
```

```
state       year
California  2000    33871648
            2010    37253956
New York    2000    18976457
            2010    19378102
Texas       2000    20851820
            2010    25145561
dtype: int64
```

Мультииндекс для столбцов

В объектах DataFrame строки и столбцы полностью симметричны, и у столбцов, точно так же, как и у строк, может быть несколько уровней индексов. Рассмотрим следующий пример, представляющий собой имитацию неких (в чем-то достаточно реалистичных) медицинских данных:

```
>>> index = pd.MultiIndex.from_product([[2013, 2014], [1, 2]],
                                       names=['year', 'visit'])
>>> index

MultiIndex([(2013, 1),
            (2013, 2),
            (2014, 1),
            (2014, 2)],
           names=['year', 'visit'])

>>> columns = pd.MultiIndex.from_product([['Bob', 'Guido', 'Sue'],
                                          ['HR', 'Temp']],
                                          names=['subject', 'type'])
>>> columns

MultiIndex([(  'Bob',   'HR'),
            (  'Bob', 'Temp'),
            ('Guido',   'HR'),
            ('Guido', 'Temp'),
            (  'Sue',   'HR'),
            (  'Sue', 'Temp')],
           names=['subject', 'type'])
```

```
>>> data = np.round(np.random.randn(4, 6), 1)
>>> data[:, ::2] *= 10
>>> data += 37

>>> health_data = pd.DataFrame(data,
                               index=index,
                               columns=columns)

>>> health_data
```

subject		Bob		Guido		Sue	
type		HR	Temp	HR	Temp	HR	Temp
year	visit						
2013	1	40.0	37.7	30.0	39.0	52.0	35.7
	2	30.0	35.4	48.0	37.0	33.0	37.0
2014	1	46.0	38.1	49.0	38.5	43.0	36.0
	2	49.0	37.4	20.0	37.1	37.0	37.5

Пульс (HR, от англ. heart rate — «частота сердцебиений»)

Как видим, мультииндексация как строк, так и столбцов может оказаться чрезвычайно удобной. По сути дела, это четырехмерные данные со следующими измерениями: субъект, измеряемый параметр (пульс и температура), год и номер посещения. При наличии этого мы можем, например, индексировать столбец верхнего уровня по имени человека и получить объект DataFrame, содержащий информацию только об этом человеке:

```
>>> health_data['Guido']
```

	type	HR	Temp
year	visit		
2013	1	30.0	39.0
	2	48.0	37.0
2014	1	49.0	38.5
	2	20.0	37.1

Индексация и срезы по мультииндексу

Рассмотрим мультииндексированный объект Series, содержащий количество населения по штатам:

```
>>> pop
```

```
state       year
California  2000    33871648
            2010    37253956
New York    2000    18976457
            2010    19378102
Texas       2000    20851820
            2010    25145561
dtype: int64
```

```
>>> pop['California', 2000]
```

```
33871648
```

Объект MultiIndex поддерживает также частичную индексацию (partial indexing), то есть индексацию только по одному из уровней индекса. Результат — тоже объект Series , с более низкоуровневыми индексами:

```
>>> pop['California']
```

```
year
2000    33871648
2010    37253956
dtype: int64
```

Возможно также выполнение частичных срезов, **если мультииндекс отсортирован**:

```
>>> pop.loc['California':'New York']
```

```
state       year
California  2000    33871648
            2010    37253956
New York    2000    18976457
            2010    19378102
dtype: int64
```

С помощью отсортированных индексов можно выполнять частичную индексацию по нижним уровням, указав пустой срез в первом индексе:

```
>>> pop[:, 2000]

state
California    33871648
New York      18976457
Texas         20851820
dtype: int64
```

Выборка данных на основе булевой маски:

```
>>> pop[pop > 22000000]

state       year
California  2000    33871648
            2010    37253956
Texas       2010    25145561
dtype: int64
```

Выборка на основе «прихотливой» индексации:

```
>>> pop[['California', 'Texas']]

state       year
California  2000    33871648
            2010    37253956
Texas       2000    20851820
            2010    25145561
dtype: int64
```

Мультииндексация объектов DataFrame

```
>>> health_data
```

	subject	Bob		Guido		Sue	
	type	HR	Temp	HR	Temp	HR	Temp
year	visit						
2013	1	40.0	37.7	30.0	39.0	52.0	35.7
	2	30.0	35.4	48.0	37.0	33.0	37.0
2014	1	46.0	38.1	49.0	38.5	43.0	36.0
	2	49.0	37.4	20.0	37.1	37.0	37.5

```
>>> health_data['Guido', 'HR']

year  visit
2013  1        30.0
      2        48.0
2014  1        49.0
      2        20.0
Name: (Guido, HR), dtype: float64
```

```
>>> health_data.iloc[:2, :2]
```

	subject	Bob	
	type	HR	Temp
year	visit		
2013	1	40.0	37.7
	2	30.0	35.4

Отсортированные и неотсортированные индексы

Большинство операций срезов с мультииндексами завершится ошибкой, если индекс не отсортирован. Рассмотрим этот вопрос.
Начнем с создания простых мультииндексированных данных, индексы в которых не отсортированы лексикографически:

```
>>> index = pd.MultiIndex.from_product([['a', 'c', 'b'], [1, 2]])
>>> data = pd.Series(np.random.rand(6), index=index)
>>> data.index.names = ['char', 'int']
>>> data
```

```
char  int
a     1      0.191575
      2      0.443920
c     1      0.397574
      2      0.297181
b     1      0.970645
      2      0.941924
dtype: float64
```

Если мы попытаемся выполнить частичный срез этого индекса, то получим ошибку:

```
try:
   data['a':'b']
except KeyError as e:
   print(type(e))
   print(e)
```

```
<class 'pandas.errors.UnsortedIndexError'>
'Key length (1) was greater than MultiIndex
lexsort depth (0)'
```

Хотя из сообщения об ошибке это не вполне понятно, ошибка генерируется, потому что объект MultiIndex не отсортирован. По различным причинам частичные срезы и другие подобные операции требуют, чтобы уровни мультииндекса были отсортированы (лексикографически упорядочены).

Библиотека Pandas предоставляет множество удобных инструментов для выполнения подобной сортировки.
В качестве примеров можем указать методы **sort_index()** и **sortlevel()** объекта DataFrame.

```
>>> data = data.sort_index()
>>> data

char  int
a     1      0.191575
      2      0.443920
b     1      0.970645
      2      0.941924
c     1      0.397574
      2      0.297181
dtype: float64
```

Убывание:
`ascending=False`

После подобной сортировки индекса частичный срез будет выполняться как положено:

```
>>> data['a':'b']

char  int
a     1      0.191575
      2      0.443920
b     1      0.970645
      2      0.941924
dtype: float64
```

Выполнение над индексами операций stack и unstack

Существует возможность преобразовывать набор данных из вертикального мультииндексированного в простое двумерное представление, при необходимости указывая требуемый уровень:

```
>>> pop.unstack(level=0)
```

state	California	New York	Texas
year			
2000	33871648	18976457	20851820
2010	37253956	19378102	25145561

```
>>> pop.unstack(level=1)
```

year	2000	2010
state		
California	33871648	37253956
New York	18976457	19378102
Texas	20851820	25145561

Методу unstack() противоположен по действию метод stack() , которым можно воспользоваться, чтобы получить обратно исходный ряд данных:

```
>>> pop.unstack().stack()
```

```
state       year
California  2000    33871648
            2010    37253956
New York    2000    18976457
            2010    19378102
Texas       2000    20851820
            2010    25145561
dtype: int64
```

Создание и перестройка индексов

Еще один способ перегруппировки иерархических данных — преобразовать метки индекса в столбцы с помощью метода **reset_index** . Результатом вызова этого метода для нашего ассоциативного словаря населения будет объект DataFrame со столбцами state (штат) и year (год), содержащими информацию, ранее находившуюся в индексе. Для большей ясности можно при желании задать название для представленных в виде столбцов данных:

```
>>> pop_flat = pop.reset_index(name='population')
>>> pop_flat
```

	state	year	population
0	California	2000	33871648
1	California	2010	37253956
2	New York	2000	18976457
3	New York	2010	19378102
4	Texas	2000	20851820
5	Texas	2010	25145561

При работе с реальными данными исходные входные данные очень часто выглядят подобным образом, поэтому удобно создать объект **MultiIndex** из значений столбцов. Это можно сделать с помощью метода **set_index** объекта **DataFrame** , возвращающего мультииндексированный объект **DataFrame** :

```
>>> pop_flat.set_index(['state', 'year'])
```

		population
state	**year**	
California	**2000**	33871648
	2010	37253956
New York	**2000**	18976457
	2010	19378102
Texas	**2000**	20851820
	2010	25145561

Агрегирование по мультииндексам

В Pandas имеются встроенные методы для агрегирования данных, например mean() , sum() и max() . В случае иерархически индексированных данных им можно передать параметр **level** для указания подмножества данных, на котором будет вычисляться сводный показатель.

```
>>> health_data
```

	subject	Bob		Guido		Sue	
	type	HR	Temp	HR	Temp	HR	Temp
year	visit						
2013	1	40.0	37.7	30.0	39.0	52.0	35.7
	2	30.0	35.4	48.0	37.0	33.0	37.0
2014	1	46.0	38.1	49.0	38.5	43.0	36.0
	2	49.0	37.4	20.0	37.1	37.0	37.5

Допустим, нужно усреднить результаты измерений показателей по двум визитам в течение года. Сделать это можно путем указания уровня индекса, который мы хотели бы исследовать, в данном случае года (year):

```
>>> data_mean = health_data.mean(level='year')
>>> data_mean
```

subject	Bob		Guido		Sue	
type	HR	Temp	HR	Temp	HR	Temp
year						
2013	35.0	36.55	39.0	38.0	42.5	36.35
2014	47.5	37.75	34.5	37.8	40.0	36.75

Далее, воспользовавшись ключевым словом **axis** , можно получить и среднее значение по уровням по столбцам:

```
>>> data_mean.mean(axis=1, level='type')
```

type	HR	Temp
year		
2013	38.833333	36.966667
2014	40.666667	37.433333

Так, всего двумя строками кода мы смогли найти средний пульс и температуру по всем субъектам и визитам за каждый год.