Объединение наборов данных

https://dfedorov.spb.ru

Объединение наборов данных: конкатенация и добавление в конец

Некоторые наиболее интересные исследования выполняются благодаря объединению различных источников данных. Эти операции могут включать в себя что угодно, начиная с простейшей конкатенации двух различных наборов данных до более сложных соединений и слияний в стиле баз данных, корректно обрабатывающих все возможные частичные совпадения наборов.

```
def make_df(cols, ind):
    """Быстро создаем объект DataFrame"""
    data = {c: [str(c) + str(i) for i in ind] for c in cols}
    return pd.DataFrame(data, ind)
```

```
>>> make_df('ABC', range(3))
```

Напоминание: конкатенация массивов NumPy

Конкатенация объектов **Series** и **DataFrame** очень похожа на конкатенацию массивов библиотеки NumPy, которую можно осуществить посредством функции np.concatenate:

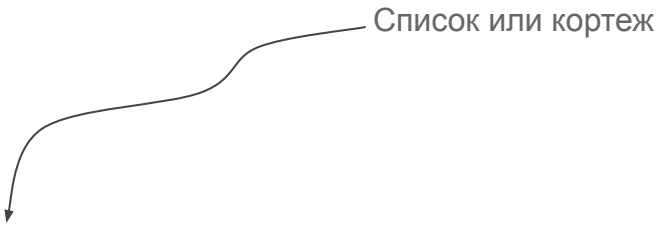
```
>>> import pandas as pd
>>> import numpy as np

>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> z = [7, 8, 9]
>>> np.concatenate([x, y, z])

 array([1, 2, 3, 4, 5, 6, 7, 8, 9])


>>> x = [[1, 2], [3, 4]]
>>> np.concatenate([x, x], axis=1)

 array([[1, 2, 1, 2],
        [3, 4, 3, 4]])
```



Простая конкатенация с помощью метода pd.concat

В Pandas имеется функция, pd.concat(), синтаксис которой аналогичен функции np.concatenate:

```
>>> ser1 = pd.Series(['A', 'B', 'C'], index=[1, 2, 3])
>>> ser2 = pd.Series(['D', 'E', 'F'], index=[4, 5, 6])
>>> pd.concat([ser1, ser2])

1    A
2    B
3    C
4    D
5    E
6    F
dtype: object
```

```
>>> df1 = make_df('AB', [1, 2])
>>> df1
```

	A	B
1	A1	B1
2	A2	B2

```
>>> df2 = make_df('AB', [3, 4])
>>> df2
```

	A	B
3	A3	B3
4	A4	B4

```
>>> pd.concat([df1, df2])
```

	A	B
1	A1	B1
2	A2	B2
3	A3	B3
4	A4	B4

```
>>> df3 = make_df('AB', [0, 1])
>>> df3
```

	A	B
0	A0	B0
1	A1	B1

```
>>> df4 = make_df('CD', [0, 1])
>>> df4
```

	C	D
0	C0	D0
1	C1	D1

```
>>> pd.concat([df3, df4], axis=1)
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1

Дублирование индексов

Важное различие между функциями **np.concatenate()** и **pd.concat()** состоит в том, что конкатенация из Pandas сохраняет индексы, даже если в результате некоторые индексы будут дублироваться.

```
>>> x = make_df('AB', [0, 1])
>>> x
```

	A	B
0	A0	B0
1	A1	B1

```
>>> y = make_df('AB', [2, 3])
>>> y
```

	A	B
2	A2	B2
3	A3	B3

```
>>> y.index = x.index # Дублируем индексы!
```

	A	B
0	A0	B0
1	A1	B1

	A	B
0	A2	B2
1	A3	B3

```
>>> pd.concat([x, y])
```

	A	B
0	A0	B0
1	A1	B1
0	A2	B2
1	A3	B3

Проверка на наличие перекрывающихся индексов

```
try:
    pd.concat([x, y], verify_integrity=True)
except ValueError as e:
    print("ValueError:", e)
```

```
ValueError: Indexes have overlapping values:
Int64Index([0, 1], dtype='int64')
```

Игнорирование индекса

Иногда индекс сам по себе не имеет значения и лучше его просто проигнорировать. Для этого достаточно установить флаг **ignore_index**. В случае равного **True** значения этого флага конкатенация приведет к созданию нового целочисленного индекса для итогового объекта.

```
>>> pd.concat([x, y], ignore_index=True)
```

	A	B
0	A0	B0
1	A1	B1
2	A2	B2
3	A3	B3

Добавление ключей мультииндекса

Параметр **keys** позволяет задать метки для источников данных.
Результатом будут иерархически индексированные ряды.

```
>>> pd.concat([x, y], keys=['x', 'y'])
```

		A	B
x	**0**	A0	B0
	1	A1	B1
y	**0**	A2	B2
	1	A3	B3

Конкатенация с использованием соединений

На практике у данных из разных источников могут быть различные наборы имен столбцов.
На этот случай у функции **pd.concat()** имеется несколько опций.
Рассмотрим объединение следующих двух объектов DataFrame , у которых столбцы (но не все!) называются одинаково:

```
>>> df5 = make_df('ABC', [1, 2])
>>> df5
```

	A	B	C
1	A1	B1	C1
2	A2	B2	C2

```
>>> df6 = make_df('BCD', [3, 4])
>>> df6
```

	B	C	D
3	B3	C3	D3
4	B4	C4	D4

```
>>> pd.concat([df5, df6])
```

join='outer'

	A	B	C	D
1	A1	B1	C1	NaN
2	A2	B2	C2	NaN
3	NaN	B3	C3	D3
4	NaN	B4	C4	D4

По умолчанию элементы, данные для которых отсутствуют, заполняются NA-значениями. Чтобы поменять это поведение, можно указать одну из нескольких опций для параметров **join** и **join_axes** функции конкатенации.

По умолчанию соединение — объединение входных столбцов (**join='outer'**), но есть возможность поменять это поведение на пересечение столбцов с помощью опции **join='inner'** :

```
>>> df5
```

	A	B	C
1	A1	B1	C1
2	A2	B2	C2

```
>>> df6
```

	B	C	D
3	B3	C3	D3
4	B4	C4	D4

```
>>> pd.concat([df5, df6], join='inner')
```

	B	C
1	B1	C1
2	B2	C2
3	B3	C3
4	B4	C4

Метод append()

Непосредственная конкатенация массивов настолько распространена, что в объекты **Series** и **DataFrame** был включен метод append() , позволяющий выполнить то же самое с меньшими усилиями. Например, вместо вызова **pd.concat([df1, df2])** можно вызвать **df1.append(df2)** :

```
>>> df1
```

	A	B
1	A1	B1
2	A2	B2

```
>>> df2
```

	A	B
3	A3	B3
4	A4	B4

```
>>> df1.append(df2)
```

	A	B
1	A1	B1
2	A2	B2
3	A3	B3
4	A4	B4

Не забывайте, что, в отличие от методов append() и extend() списков языка Python, метод append() в библиотеке Pandas не изменяет исходный объект. Вместо этого он создает новый объект с объединенными данными, что делает этот метод не слишком эффективным, поскольку означает создание нового индекса и буфера данных.

Следовательно, если вам необходимо выполнить несколько операций append , лучше создать список объектов DataFrame и передать их все сразу функции concat() .

Конкатенация данных

```
# создаем два объекта Series для конкатенации
>>> s1 = pd.Series(np.arange(0, 3))
>>> s1
0    0
1    1
2    2
dtype: int64
>>> s2 = pd.Series(np.arange(5, 8))
>>> s2
0    5
1    6
2    7
dtype: int64
# конкатенируем их
>>> pd.concat([s1, s2])
0    0
1    1
2    2
0    5
1    6
2    7
dtype: int64
```

```
# создаем два объекта DataFrame для конкатенации,
# используя те же самые индексные метки и имена столбцов,
# но другие значения
>>> df1 = pd.DataFrame(np.arange(9).reshape(3, 3),
                       columns=['a', 'b', 'c'])
>>> df1
```

	a	b	c
0	0	1	2
1	3	4	5
2	6	7	8

```
# df2 имеет значения 9 .. 18
>>> df2 = pd.DataFrame(np.arange(9, 18).reshape(3, 3),
                       columns=['a', 'b', 'c'])
>>> df2
```

	a	b	c
0	9	10	11
1	12	13	14
2	15	16	17

```
# выполняем конкатенацию
>>> pd.concat([df1, df2])
```

	a	b	c
0	0	1	2
1	3	4	5
2	6	7	8
0	9	10	11
1	12	13	14
2	15	16	17

```
# демонстрируем конкатенацию двух объектов DataFrame
# с разными столбцами
>>> df1 = pd.DataFrame(np.arange(9).reshape(3, 3),
                       columns=['a', 'b', 'c'])
>>> df1
```

	a	b	c
0	0	1	2
1	3	4	5
2	6	7	8

```
>>> df2 = pd.DataFrame(np.arange(9, 18).reshape(3, 3),
                       columns=['a', 'c', 'd'])
>>> df2
```

	a	c	d
0	9	10	11
1	12	13	14
2	15	16	17

```
# выполняем конкатенацию, пропусками будут заполнены
# значения столбца d в датафрейме df1 и
# значения столбца b в датафрейме df2
>>> pd.concat([df1, df2], sort=True)
```

	a	b	c	d
0	0	1.0	2	NaN
1	3	4.0	5	NaN
2	6	7.0	8	NaN
0	9	NaN	10	11.0
1	12	NaN	13	14.0
2	15	NaN	16	17.0

```
# выполняем конкатенацию двух объектов,
# но при этом создаем индекс с помощью
# заданных ключей
>>> c = pd.concat([df1, df2], keys=['df1', 'df2'], sort=True)
# обратите внимание на метки строк в выводе
>>> c
```

		a	b	c	d
df1	0	0	1.0	2	NaN
	1	3	4.0	5	NaN
	2	6	7.0	8	NaN
df2	0	9	NaN	10	11.0
	1	12	NaN	13	14.0
	2	15	NaN	16	17.0

```
# мы можем извлечь данные, относящиеся к первому
# или второму исходному датафрейму
>>> c.loc['df2']
```

	a	b	c	d
0	9	NaN	10	11.0
1	12	NaN	13	14.0
2	15	NaN	16	17.0

```
# конкатенируем датафреймы df1 и df2 по оси столбцов
# выравниваем по меткам строк,
# получаем дублирующиеся столбцы
>>> pd.concat([df1, df2], axis=1)
```

	a	b	c	a	c	d
0	0	1	2	9	10	11
1	3	4	5	12	13	14
2	6	7	8	15	16	17

```
# создаем новый датафрейм d3, чтобы конкатенировать его
# с датафреймом df1
# датафрейм df3 имеет общую с датафреймом df1
# метку 2 и общий столбец (a)
>>> df3 = pd.DataFrame(np.arange(20, 26).reshape(3, 2),
                       columns=['a', 'd'],
                       index=[2, 3, 4])
>>> df3
```

	a	d
2	20	21
3	22	23
4	24	25

```
# конкатерируем их по оси столбцов. Происходит выравнивание по меткам строк,
# осуществляется заполнение значений столбцов df1, а затем
# столбцов df3, получаем дублирующиеся столбцы
>>> pd.concat([df1, df3], axis=1)
```

	a	b	c	a	d
0	0.0	1.0	2.0	NaN	NaN
1	3.0	4.0	5.0	NaN	NaN
2	6.0	7.0	8.0	20.0	21.0
3	NaN	NaN	NaN	22.0	23.0
4	NaN	NaN	NaN	24.0	25.0

```
# выполняем внутреннее соединение (пересечение) вместо внешнего
# результат представлен в виде одной строки
>>> pd.concat([df1, df3], axis=1, join='inner')
```

	a	b	c	a	d
2	6	7	8	20	21

```
# добавляем ключи к столбцам
>>> df = pd.concat([df1, df2],
                   axis=1,
                   keys=['df1', 'df2'])
>>> df
```

	df1			df2		
	a	b	c	a	c	d
0	0	1	2	9	10	11
1	3	4	5	12	13	14
2	6	7	8	15	16	17

```
# извлекаем данные из датафрейма
# с помощью ключа 'df2'
>>> df.loc[:, 'df2']
```

	a	c	d
0	9	10	11
1	12	13	14
2	15	16	17

```
# метод .append() выполняет конкатенацию по оси строк (axis=0),
# в результате получаем дублирующиеся индексные метки строк
>>> df1.append(df2, sort=True)
```

	a	b	c	d
0	0	1.0	2	NaN
1	3	4.0	5	NaN
2	6	7.0	8	NaN
0	9	NaN	10	11.0
1	12	NaN	13	14.0
2	15	NaN	16	17.0

```
# избавляемся от дублирования меток в итоговом
индексе,
# игнорируя индексные метки в датафреймах-источниках
>>> df1.append(df2, ignore_index=True, sort=True)
```

	a	b	c	d
0	0	1.0	2	NaN
1	3	4.0	5	NaN
2	6	7.0	8	NaN
3	9	NaN	10	11.0
4	12	NaN	13	14.0
5	15	NaN	16	17.0