Введение в библиотеку pandas

С.В. Лемешевский (sergey.lemeshevsky@gmail.com)

Институт математики НАН Беларуси

Apr 1, 2020

Библиотека `pandas` содержит структуры данных и инструменты управления данными, предназначенные для очистки данных и быстрого и простого анализа данных в Python. Библиотека `pandas` часто используется в тандеме с инструментами для численных расчетов, такими как NumPy и SciPy, библиотеками для анализа данных, такими как `statmodels` и `scikit-learn`, и библиотеками для визуализации, такими как `matplotlib`.

Ниже будем использовать следующее соглашение для импорта библиотеки `pandas`:

```
import pandas as pd
```

Содержание

1. Структуры данных в pandas

Чтобы начать работать с `pandas`, рассмотрим две основные структуры: `Series` и `DataFrame`. Они не являются универсальными решениями любых задач, однако эти структуры предоставляют прочный легкий в использовании фундамент для большинства приложений.

1.1. Класс `Series`

`Series` (*ряд*) — объект, типа одномерного массива, содержащий последовательность значений (типов, аналогичных типам NumPy) и связанный с ним массив меток данных, называемых *индексами*. Создадим простейший объект типа `Series` только из массива данных:

```
In [2]: obj = pd.Series([4, 7, -5, 3])

In [3]: obj
Out[3]:
0    4
1    7
2   -5
3    3
dtype: int64
```

Строковое представление объекта `Series` в интерактивном режиме отображает индексы слева, а данные справа. Так как мы не определили индексы, то по умолчанию индексы содержат целые числа от `0` до `N-1` (где `N` — длина массива данных). Можно получить представление в виде массива и индексы ряда с помощью атрибутов `values` и `index`:

```
In [4]: obj.values
Out[4]: array([ 4,  7, -5,  3])

In [5]: obj.index     # как range(4)
Out[5]: RangeIndex(start=0, stop=4, step=1)
```

Часто желательно создать ряд с индексами, идентифицирующими каждую точку данный с меткой:

```
In [6]: obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])

In [7]: obj2
Out[7]:
d    4
b    7
a   -5
c    3
dtype: int64

In [8]: obj2.index
Out[8]: Index(['d', 'b', 'a', 'c'], dtype='object')
```

В отличие от массивов NumPy, можно использовать метки при индексации при выборе отдельных значений или набора значений:

```
In [9]: obj2['a']
Out[9]: -5

In [10]: obj2['d'] = 6

In [11]: obj2[['c', 'a', 'd']]
Out[11]:
c    3
a   -5
d    6
dtype: int64
```

Использование функций NumPy или операций подобных NumPy, таких как фильтрация с помощью булевых массивов, умножение на скаляр или вычисление математических функций, сохраняет значения индексов:

```
In [12]: obj2[obj2 > 0]
Out[12]:
d    6
b    7
c    3
dtype: int64

In [13]: obj2 * 2
Out[13]:
d    12
b    14
a   -10
c     6
dtype: int64

In [14]: import numpy as np

In [15]: np.exp(obj2)
Out[15]:
d     403.428793
b    1096.633158
```

```
a       0.006738
c      20.085537
dtype: float64
```

Ряды можно рассматривать как словари фиксированной длины:

```
In [16]: 'b' in obj2
Out[16]: True

In [17]: 'e' in obj2
Out[17]: False
```

Если имеются данные, содержащиеся в словаре, можно создать ряд из него:

```
In [18]: sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}

In [19]: obj3 = pd.Series(sdata)

In [20]: obj3
Out[20]:
Ohio      35000
Texas     71000
Oregon    16000
Utah       5000
dtype: int64
```

Если передается только словарь, то индексами ряда будут ключи словаря в том порядке, в котором были при создании словаря. Можно изменить порядок индекса передавая ключи словаря в порядке, который нужен:

```
In [21]: states = ['California', 'Ohio', 'Oregon', 'Texas']

In [22]: obj4 = pd.Series(sdata, index=states)

In [23]: obj4
Out[23]:
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

Здесь три значения, найденные в `sdata`, были размещены в соответствующих местах, но так как не было найдено значение для `'California'`, оно отображается как `NaN` (не число), которое в `pandas` используется для обозначение пропущенных значений или значений «NA» (*not available*). Поскольку `'Юта'` не была включена в `states`, этот элемент исключается из результирующего объекта. Функции `isnull` и `notnull` в `pandas` используются для обнаружения отсутствующих данных:

```
In [24]: pd.isnull(obj4)
Out[24]:
California     True
Ohio          False
Oregon        False
Texas         False
dtype: bool

In [25]: pd.notnull(obj4)
Out[25]:
California    False
Ohio           True
Oregon         True
Texas          True
dtype: bool
```

Класс `Series` также имеет эти методы:

```
In [26]: obj4.isnull()
Out[26]:
California     True
Ohio          False
Oregon        False
Texas         False
dtype: bool
```

Полезное свойство `Series` заключается в том, что она автоматически происходит выравнивание по индексам в арифметических операциях:

```
In [27]: obj3
Out[27]:
Ohio      35000
Texas     71000
Oregon    16000
Utah       5000
dtype: int64

In [28]: obj4
Out[28]:
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64

In [29]: obj3 + obj4
Out[29]:
California         NaN
Ohio           70000.0
Oregon         32000.0
Texas         142000.0
Utah               NaN
dtype: float64
```

Как объекты `Series`, так и из индексы имеют атрибут `name`:

```
In [30]: obj4.name = 'population'

In [31]: obj4.index.name = 'state'

In [32]: obj4
Out[32]:
state
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
Name: population, dtype: float64
```

Можно изменять индексы рядов присваиванием:

```
In [33]: obj
Out[33]:
0    4
1    7
2   -5
3    3
dtype: int64

In [34]: obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']

In [35]: obj
Out[35]:
Bob      4
Steve    7
Jeff    -5
Ryan     3
dtype: int64
```

1.2. Класс `DataFrame`

`DataFrame` представляет собой прямоугольную таблицу данных и содержит упорядоченную коллекцию столбцов, каждый из которых может иметь различный тип значения (числовой, строковый, логический и т.д.). `DataFrame` имеет индексы столбцов и строк.

Есть много способов создания объекта `DataFrame`, хотя один из наиболее распространенных — это использование списков, словарей или массивов NumPy:

```
In [36]: data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
    ...: 'year': [2000, 2001, 2002, 2001, 2002, 2003],
    ...: 'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}

In [37]: frame = pd.DataFrame(data)
```

Полученный в результате `DataFrame` получит автоматически индексацию для строк (как в `Series`), а индексом столбцов будут ключи словаря:

```
In [38]: frame
Out[38]:
    state  year  pop
0    Ohio  2000  1.5
1    Ohio  2001  1.7
2    Ohio  2002  3.6
3  Nevada  2001  2.4
4  Nevada  2002  2.9
5  Nevada  2003  3.2
```

Если используется блокнот Jupyter, объекты `DataFrame` будут отображаться в виде более удобной для просмотра HTML-таблицы.

Для больших DataFrames метод head выбирает только первые пять строк:

```
In [39]: frame.head()
Out[39]:
    state  year  pop
0    Ohio  2000  1.5
1    Ohio  2001  1.7
2    Ohio  2002  3.6
3  Nevada  2001  2.4
4  Nevada  2002  2.9
```

Можно задавать другой порядок столбцов:

```
In [40]: pd.DataFrame(data, columns=['year', 'state', 'pop'])
Out[40]:
   year   state  pop
0  2000    Ohio  1.5
1  2001    Ohio  1.7
2  2002    Ohio  3.6
3  2001  Nevada  2.4
4  2002  Nevada  2.9
5  2003  Nevada  3.2
```

Если передать столбец, который не содержится в словаре, то в результате будет столбец с отсутствующими значениями:

```
In [41]: frame2 = pd.DataFrame(data, columns=['year', 'state', 'pop', 'debt'],
    ...: index=['one', 'two', 'three', 'four',
    ...: 'five', 'six'])

In [42]: frame2
Out[42]:
       year   state  pop debt
one    2000    Ohio  1.5  NaN
two    2001    Ohio  1.7  NaN
three  2002    Ohio  3.6  NaN
four   2001  Nevada  2.4  NaN
five   2002  Nevada  2.9  NaN
six    2003  Nevada  3.2  NaN
```

```
In [43]: frame2.columns
Out[43]: Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

К столбцу `DataFrame` можно получить доступ как к ряду с помощью нотацией подобной словарю или через атрибут:

```
In [44]: frame2['state']
Out[44]:
one        Ohio
two        Ohio
three      Ohio
four     Nevada
five     Nevada
six      Nevada
Name: state, dtype: object

In [45]: frame2.year
Out[45]:
one      2000
two      2001
three    2002
four     2001
five     2002
six      2003
Name: year, dtype: int64
```

Замечание
IPython предоставлет доступ по атрибуту (например, `frame2.year`) по автодополнению с помощью клавиши <TAB>.

Вариант `frame2[column]` работает для любых имен столбцов, в то время как `frame2.column` работает только если имя столбца является допустимым в Python именем переменной.

К строкам можно получить доступ по позиции или с помощью специального атрибута `loc`:

```
In [46]: frame2.loc['three']
Out[46]:
year     2002
state    Ohio
pop       3.6
debt      NaN
Name: three, dtype: object
```

Можно менять значения столбцов. Например, пустой столбцу `debt` можно присвоить скалярное значение или массив:

```
In [47]: frame2['debt'] = 16.5
```

```
In [48]: frame2
Out[48]:
       year   state  pop  debt
one    2000    Ohio  1.5  16.5
two    2001    Ohio  1.7  16.5
three  2002    Ohio  3.6  16.5
four   2001  Nevada  2.4  16.5
five   2002  Nevada  2.9  16.5
six    2003  Nevada  3.2  16.5

In [49]: frame2['debt'] = np.arange(6.)

In [50]: frame2
Out[50]:
       year   state  pop  debt
one    2000    Ohio  1.5   0.0
two    2001    Ohio  1.7   1.0
three  2002    Ohio  3.6   2.0
four   2001  Nevada  2.4   3.0
five   2002  Nevada  2.9   4.0
six    2003  Nevada  3.2   5.0
```

При присваивании столбцу списка или массива, их длина должна быть той же, что и длина `DataFrame`. Если присваивать объект `Series`, то его метки будут выровнены по индексу `DataFrame`, при этом будут вставляться отсутствующие значения для любых «дыр»:

```
In [51]: val = pd.Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])

In [52]: frame2['debt'] = val

In [53]: frame2
Out[53]:
       year   state  pop  debt
one    2000    Ohio  1.5   NaN
two    2001    Ohio  1.7  -1.2
three  2002    Ohio  3.6   NaN
four   2001  Nevada  2.4  -1.5
five   2002  Nevada  2.9  -1.7
six    2003  Nevada  3.2   NaN
```

При присваивании отсутствующего столбца в объекте `DataFrame` добавится новый столбец. Ключевое слово `del` удаляет столбец, как и для словарей:

```
In [54]: frame2['eastern'] = frame2.state == 'Ohio'

In [55]: frame2
Out[55]:
       year   state  pop  debt  eastern
one    2000    Ohio  1.5   NaN     True
two    2001    Ohio  1.7  -1.2     True
three  2002    Ohio  3.6   NaN     True
four   2001  Nevada  2.4  -1.5    False
five   2002  Nevada  2.9  -1.7    False
six    2003  Nevada  3.2   NaN    False
```

Предупреждение
Новый столбец не может быть добавлен с помощью синтаксиса `frame2.eastern`.

```
In [56]: del frame2['eastern']

In [57]: frame2.columns
Out[57]: Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

Предупреждение
Столбец, возвращаемый при индексации `DataFrame`, является представлением данных, а не копией. Таким образом, любые изменения в объекте `Series` будут отражены в объекте `DataFrame`. Столбец можно явно скопировать с помощью метода `Series.copy`.

Другой распространенной формой представления данных является вложенный словарь словарей:

```
In [58]:
```

Если вложенный словарь передать в конструктор `DataFrame`, `pandas` интерпретирует ключи внешнего словаря как столбцы, а внутренние ключи — как индексы:

```
In [59]: frame3 = pd.DataFrame(pop)

In [60]: frame3
Out[60]:
      Nevada  Ohio
2001     2.4   1.7
2002     2.9   3.6
2000     NaN   1.5
```

Можно транспонировать `DataFrame`:

```
In [61]: frame3.T
```

Можно задать порядок индексов:

```
In [62]: pd.DataFrame(pop, index=[2001, 2002, 2003])
Out[62]:
      Nevada  Ohio
2001     2.4   1.7
2002     2.9   3.6
2003     NaN   NaN
```

Словари рядов обрабатываются практически также:

```
In [63]: pdata = {'Ohio': frame3['Ohio'][:-1], 'Nevada': frame3['Nevada'][:2]}

In [64]: pd.DataFrame(pdata)
Out[64]:
      Nevada   Ohio
2000   NaN     1.5
2001   2.4     1.7
```

Полный список параметров, которые можно передавать в конструктор `DataFrame`, можно найти в таблице 1.

Если для индекса и столбцов DataFrame установлены атрибуты `name`, они также будут отображены:

```
In [65]: frame3.index.name = 'year'; frame3.columns.name = 'state'

In [66]: frame3
Out[66]:
state  Nevada  Ohio
year
2001      2.4   1.7
2002      2.9   3.6
2000      NaN   1.5
```

Как и в случае `Series`, атрибут `values` возвращает данные, содержащиеся в `DataFrame`, в виде двумерного массива:

```
In [67]: frame3.values
Out[67]:
array([[2.4, 1.7],
       [2.9, 3.6],
       [nan, 1.5]])
```

Таблица 1: Возможные входные данные для конструктора `DataFrame`

Тип	Примечания
Двумерный `ndarray`	Матрица данных, передающаяся с необязательными метками строк и столбцов
`dict` массивов, `list`, `tuple`	Каждая последовательность становится столбцом в `DataFrame`. Все последовательности должны быть одинаковой длины
Структурированный массив (или массив записей) NumPy	Обрабатывается как предыдущий случай

`dict` объектов типа `Series`	Каждое значение становится столбцом. Индексы из каждой серии объединяются вместе, чтобы сформировать индекс строки результата, если не передан явный индекс
`dict` объектов типа `dict`	Каждый внутренний словарь становится столбцом. Ключи объединяются для формирования индекса строки, как в предыдущем случае
`list` объектов `dict` или `Series`	Каждый элемент становится строкой в `DataFrame`. Объединение ключей `dict` или индексов `Series` становится метками столбцов `DataFrame`
`list` объектов `list` или `tuple`	Обрабатывается как случай двумерного массива
`DataFrame`	Используются индексы `DataFrame` , если не переданы другие
маскированный массив NumPy	Как случай двумерного массива, за исключением того, что маскированные значения становятся пропущенными (NA) значениями в итоговом `DataFrame`

1.3. Объекты типа `Index`

Объекты типа `Index` в `pandas` отвечают за хранение меток осей и других метаданных (таких как имя или имя оси).

Любой массив или другая последовательность меток, которые используются при создании `Series` или `DataFrame`, преобразуется в `Index`:

```
In [68]: obj = pd.Series(range(3), index=['a', 'b', 'c'])

In [69]: index = obj.index

In [70]: index
Out[70]: Index(['a', 'b', 'c'], dtype='object')

In [71]: index[1:]
Out[71]: Index(['b', 'c'], dtype='object')
```

Объекты `Index` — неизменяемый тип и не может изменяться пользователем:

```
In [72]: index[1] = 'd'
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-75-a452e55ce13b> in <module>
----> 1 index[1] = 'd'

/usr/lib/python3.8/site-packages/pandas/core/indexes/base.py in __setitem__(self, key, value)
   3907
   3908     def __setitem__(self, key, value):
-> 3909         raise TypeError("Index does not support mutable operations")
   3910
```

```
   3911     def __getitem__(self, key):

TypeError: Index does not support mutable operations
```

Неизменяемость делает более безопасным совместное использование объектов `Index`:

```
In [73]: labels = pd.Index(np.arange(3))

In [74]: lables
Out[74]: Int64Index([0, 1, 2], dtype='int64')

In [75]: obj2 = pd.Series([1.5, -2.5, 0], index=lables)

In [76]: obj2
Out[76]:
0    1.5
1   -2.5
2    0.0
dtype: float64

In [77]: obj2.index is lables
Out[77]: True
```

С объектами `Index` можно работать как с массивами фиксированного размера:

```
In [78]: frame3
Out[78]:
state  Nevada  Ohio
year
2001      2.4   1.7
2002      2.9   3.6
2000      NaN   1.5

In [79]: frame3.columns
Out[79]: Index(['Nevada', 'Ohio'], dtype='object', name='state')

In [80]: 'Ohio' in frame3.columns
Out[80]: True

In [81]: 2003 in frame3.index
Out[81]: False
```

В отличие от множеств Python объекты `Index` могут содержать повторяющиеся метки:

```
In [18]: sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}

In [19]: obj3 = pd.Series(sdata)

In [20]: obj3
Out[20]:
Ohio      35000
Texas     71000
Oregon    16000
```

```
Utah       5000
dtype: int64
```

Каждый объект `Index` имеет ряд методов и свойств. Некоторые полезные из них приведены в таблице 2.

Таблица 2: Некоторые методы и свойства `Index`

Метод	Описание
`append`	Добавляет дополнительные объекты `Index`, создавая новый объект `Index`
`difference`	Возвращает разность множеств как `Index`
`intersection`	Возвращает пересечение множеств
`union`	Возвращает объединение множеств
`isin`	Возвращает логический массив, указывающий, содержится ли каждое значение в переданной коллекции
`delete`	Возвращает новый объект `Index` с удаленным элементом по индексу `i`
`drop`	Возвращает новый объект `Index`, удаляя переданные значения
`insert`	Возвращает новый объект `Index`, вставляя по индексу `i` элемент
`is_monotonic`	Возвращает `True`, если каждый элемент больше либо равен предыдущего
`is_unique`	Возвращает `True`, если объект `Index` не содержит дупликатов
`unique`	Возвращает массив уникальных занчений в объекте `Index`

2. Основная функциональность

Приведем основные подходы к работе с данными, содержащимися в `Series` и `DataFrame`.

2.1. Переиндексация

Важный метод в объектах `pandas` — это `reindex`, который создает новый объект с данными, согласованными с новым индексом. Рассмотрим пример:

```
In [84]: obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])

In [85]: obj
Out[85]:
d    4.5
b    7.2
a   -5.3
```

```
c    3.6
dtype: float64
```

Вызов `reindex` в объекте `Series` переупорядочивает данные в соответствии с новым индексом, вводя пропущенные значения, если какие-либо значения индекса еще не присутствовали:

```
In [86]: obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])

In [87]: obj2
Out[87]:
a   -5.3
b    7.2
c    3.6
d    4.5
e    NaN
dtype: float64
```

*л*я упорядоченных данных, таких как временные ряды, может быть желательно выполнить некоторую интерполяцию или заполнение значений при переиндексации. Аргумент `method` позволяет нам сделать это, используя метод такой как `ffill` (forward-fill), который заполняет «вперед» значениями ряд:

```
In [88]: obj3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])

In [89]: obj3
Out[89]:
0      blue
2    purple
4    yellow
dtype: object

In [90]: obj3.reindex(range(6), method='ffill')
Out[90]:
0      blue
1      blue
2    purple
3    purple
4    yellow
5    yellow
dtype: object
```

В объектах `DataFrame` метод `reindex` может изменять *л*ибо индекс (строки), столбцы, *л*ибо и то и то. Когда передается только одна последовательность, то переиндексируются строки:

```
In [91]: frame = pd.DataFrame(np.arange(9).reshape((3, 3)),
    ...: index=['a', 'c', 'd'],
    ...: columns=['Ohio', 'Texas', 'California'])

In [92]: frame
```

```
Out[92]:
   Ohio  Texas  California
a     0      1           2
c     3      4           5
d     6      7           8

In [93]: frame2 = frame.reindex(['a', 'b', 'c', 'd'])

In [94]: frame2
Out[94]:
   Ohio  Texas  California
a   0.0    1.0         2.0
b   NaN    NaN         NaN
c   3.0    4.0         5.0
d   6.0    7.0         8.0
```

Столбцы переиндексируются с помощью аргумента `columns`:

```
In [95]: states = ['Texas', 'Utah', 'California']

In [96]: frame.reindex(columns=states)
Out[96]:
   Texas  Utah  California
a      1   NaN           2
c      4   NaN           5
d      7   NaN           8
```

В таблице 3 представлены аргументы функции `reindex`.

Таблица 3: Аргументы функции `reindex`

Аргумент	Описание
`index`	Новая последовательность для использования в качестве индекса. Может быть экземпляром `Index` или любой последовательности Python
`method`	Метод интерполяции (заполнения): `ffil` (forward-fill) — прямое заполнение, `bfill` (backward-fill) — обратное заполнение
`fill_value`	Подставляется это значения при заполнении пропущенных данных, которые появляются при переиндексации
`limit`	При заполнении задает максимальный размер шага (по количеству элементов) заполнения
`tolerance`	При заполнении задает максимальный размер шага (в абсолютном числовом расстоянии) для заполнения неточных совпадений
`level`	Сопоставляет простой `Index` на уровне `MultiIndex`; в противном случае выбирает подмножество
`copy`	Если `True`, всегда копирует данные, даже если новый индекс эквивалентен старому; если `False` не копирует данные, если индексы эквивалентны

2.2. Удаление записей с оси

Удалить одну или несколько записей легко, если имеется массив или список индексов, которые не содержат эти записи. Поскольку это может потребовать некоторых операций над множествами, метод `drop` возвращает новый объект с указанными значениями, удаленными с оси:

```
In [97]: obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])

In [98]: obj
Out[98]:
a    0.0
b    1.0
c    2.0
d    3.0
e    4.0
dtype: float64

In [99]: new_obj = obj.drop('c')

In [100]: new_obj
Out[100]:
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64

In [101]: obj.drop(['d', 'c'])
Out[101]:
a    0.0
b    1.0
e    4.0
dtype: float64
```

В `DataFrame` значения индекса могут быть удалены с любой оси:

```
In [102]: data = pd.DataFrame(np.arange(16).reshape((4, 4)),
     ...: index=['Ohio', 'Colorado', 'Utah', 'New York'],
     ...: columns=['one', 'two', 'three', 'four'])

In [103]: data
Out[103]:
          one  two  three  four
Ohio        0    1      2     3
Colorado    4    5      6     7
Utah        8    9     10    11
New York   12   13     14    15
```

Вызов `drop` с последовательностью меток удаляет значения из меток строк (ось `0`):

```
In [104]: data.drop(['Colorado', 'Ohio'])
Out[104]:
          one  two  three  four
Utah        8    9     10    11
New York   12   13     14    15
```

Удалить значения в столбцах можно передавая параметр `axis=1` или `axis=columns`:

```
In [105]: data.drop('two', axis=1)
Out[105]:
          one  three  four
Ohio        0      2     3
Colorado    4      6     7
Utah        8     10    11
New York   12     14    15

In [106]: data.drop(['two', 'four'], axis='columns')
Out[106]:
          one  three
Ohio        0      2
Colorado    4      6
Utah        8     10
New York   12     14
```

Многие функции, такие как `drop`, которые изменяют размер или форму `Series` или `DataFrame`, могут изменять сам объект (*in-place*) без создания нового объекта:

```
In [107]: obj.drop('c', inplace=True)

In [108]: obj
Out[108]:
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64
```

Предупреждение
Будьте осторожны с параметром `inplace`, так как происходит **удаление** данных.

2.3. Арифметические операции и выравнивание данных

Важной особенностью `pandas` для некоторых приложений является поведение арифметических операций между объектами с разными индексами. При сложении объекты в случае, когда любые пары индексов отличаются, соответствующий индекс в результате является объединением исходных индексов:

```
In [109]: s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])

In [110]: s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1], index=['a', 'c', 'e', 'f', 'g'])

In [111]: s1
Out[111]:
a    7.3
c   -2.5
d    3.4
e    1.5
dtype: float64

In [112]: s2
Out[112]:
a   -2.1
c    3.6
e   -1.5
f    4.0
g    3.1
dtype: float64

In [113]: s1 + s2
Out[113]:
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
dtype: float64
```

Выравнивание данных вводит пропущенные значения в местах меток, которые не пересекаются. Пропущенные значения будут распространяться в дальнейших арифметических вычислениях.

В случае `DataFrame` выравнивание осуществляется как для строк, так и для столбцов:

```
In [114]: df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)), columns=list('bcd'),
     ...: index=['Ohio', 'Texas', 'Colorado'])

In [115]: df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)), columns=list('bde'),
     ...: index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [116]: df1
Out[116]:
            b    c    d
Ohio      0.0  1.0  2.0
Texas     3.0  4.0  5.0
Colorado  6.0  7.0  8.0

In [117]: df2
Out[117]:
          b     d     e
Utah    0.0   1.0   2.0
Ohio    3.0   4.0   5.0
```

```
Texas   6.0   7.0   8.0
Oregon  9.0  10.0  11.0
```

Сумма введенных объектов вернет новый объект `DataFrame`, чьи индексы и столбцы являются объединениями индексов и столбцов двух складываемых объектов:

```
In [118]: df1 + df2
Out[118]:
            b   c     d   e
Colorado  NaN NaN   NaN NaN
Ohio      3.0 NaN   6.0 NaN
Oregon    NaN NaN   NaN NaN
Texas     9.0 NaN  12.0 NaN
Utah      NaN NaN   NaN NaN
```

Так как столбцы `'c'` и `'e'` не находятся одновременно в обоих объектах `DataFrame`, в результате они содержат отсутствующие значения. Такое же происходит и со строками.

Если сложить объекты `DataFrame` без общих меток столбцов или строк, результат будет содержать все отсутствующие значения:

```
In [119]: df1 = pd.DataFrame({'A': [1, 2]})

In [120]: df2 = pd.DataFrame({'B': [3, 4]})

In [121]: df1
Out[121]:
   A
0  1
1  2

In [122]: df2
Out[122]:
   B
0  3
1  4

In [123]: df1 - df2
Out[123]:
    A   B
0 NaN NaN
1 NaN NaN
```

Арифметические методы с заполнением значений.

```
In [124]: df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),
     ...: columns=list('abcd'))

In [125]: df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)),
     ...: columns=list('abcde'))

In [126]: df2.loc[1, 'b'] = np.nan
```

```
In [127]: df1
Out[127]:
     a    b     c     d
0  0.0  1.0   2.0   3.0
1  4.0  5.0   6.0   7.0
2  8.0  9.0  10.0  11.0

In [128]: df2
Out[128]:
      a     b     c     d     e
0   0.0   1.0   2.0   3.0   4.0
1   5.0   NaN   7.0   8.0   9.0
2  10.0  11.0  12.0  13.0  14.0
3  15.0  16.0  17.0  18.0  19.0
```

Сложение этих объектов приводит к значениям NA в местах, которые не перекрываются:

```
In [129]: df1 + df2
Out[129]:
      a     b     c     d   e
0   0.0   2.0   4.0   6.0 NaN
1   9.0   NaN  13.0  15.0 NaN
2  18.0  20.0  22.0  24.0 NaN
3   NaN   NaN   NaN   NaN NaN
```

Для заполнения отсутствующих значений можно воспользоваться функцией `add` с дополнительным аргументом `fill_value`:

```
In [130]: df1.add(df2, fill_value=0)
Out[130]:
      a     b     c     d     e
0   0.0   2.0   4.0   6.0   4.0
1   9.0   5.0  13.0  15.0   9.0
2  18.0  20.0  22.0  24.0  14.0
3  15.0  16.0  17.0  18.0  19.0
```

В таблице 4 представлены методы для арифметических операций. У каждого из них есть аналог, начинающийся с буквы `r`, у которого переставлены аргументы. Приведенные ниже примеры эквивалентны:

```
In [131]: 1 / df1
Out[131]:
       a         b         c         d
0    inf  1.000000  0.500000  0.333333
1  0.250  0.200000  0.166667  0.142857
2  0.125  0.111111  0.100000  0.090909

In [132]: df1.rdiv(1)
Out[132]:
       a         b         c         d
0    inf  1.000000  0.500000  0.333333
```

```
1  0.250  0.200000  0.166667  0.142857
2  0.125  0.111111  0.100000  0.090909
```

Соответственно, при переиндексации `Series` или `DataFrame` вы также можете указать другое значение заполнения:

```
In [133]: df1.reindex(columns=df2.columns, fill_value=0)
Out[133]:
     a    b     c     d  e
0  0.0  1.0   2.0   3.0  0
1  4.0  5.0   6.0   7.0  0
2  8.0  9.0  10.0  11.0  0
```

Таблица 4: Гибкие арифметические методы

Метод	Описание
`add`, `radd`	Сложение (+)
`sub`, `rsub`	Вычитание (-)
`div`, `rdiv`	Деление (/)
`floordiv`, `rfloordiv`	Целочисленное деление (//)
`mul`, `rmul`	Умножение (*)
`pow`, `rpow`	Возведение в степень (**)

2.4. Операции между объектами `DataFrame` и `Series`

Как и для массивов NumPy разной размерности, существуют арифметические операции между объектами `DataFrame` и `Series`. В качестве примера рассмотрим разность между двумерным массивом и одной из его строк:

```
In [134]: arr = np.arange(12.).reshape((3, 4))

In [135]: arr
Out[135]:
array([[ 0.,  1.,  2.,  3.],
       [ 4.,  5.,  6.,  7.],
       [ 8.,  9., 10., 11.]])

In [136]: arr[0]
Out[136]: array([0., 1., 2., 3.])

In [137]: arr - arr[0]
Out[137]:
array([[0., 0., 0., 0.],
       [4., 4., 4., 4.],
       [8., 8., 8., 8.]])
```

При вычитании `arr[0]` из `arr`, операция осуществляется для каждой строки. Операции между `DataFrame` и `Series` производятся аналогично.

```
In [138]: frame = pd.DataFrame(np.arange(12.).reshape((4, 3)),
     ...: columns=list('bde'),
     ...: index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [139]: series = frame.iloc[0]

In [140]: frame
Out[140]:
          b     d     e
Utah    0.0   1.0   2.0
Ohio    3.0   4.0   5.0
Texas   6.0   7.0   8.0
Oregon  9.0  10.0  11.0

In [141]: series
Out[141]:
b    0.0
d    1.0
e    2.0
Name: Utah, dtype: float64
```

По умолчанию арифметические операции между `DataFrame` и `Series` приводят индексы объекта `Series` к столбцам объекта `DataFrame`, распространяя операцию по строкам:

```
In [142]: frame - series
Out[142]:
          b    d    e
Utah    0.0  0.0  0.0
Ohio    3.0  3.0  3.0
Texas   6.0  6.0  6.0
Oregon  9.0  9.0  9.0
```

Если значение индекса не найдено ни в столбцах `DataFrame`, ни в индексе `Series`, объекты будут переиндексированы для формирования объединения:

```
In [143]: series2 = pd.Series(range(3), index=['b', 'e', 'f'])

In [144]: frame + series2
Out[144]:
          b   d     e   f
Utah    0.0 NaN   3.0 NaN
Ohio    3.0 NaN   6.0 NaN
Texas   6.0 NaN   9.0 NaN
Oregon  9.0 NaN  12.0 NaN
```

Если вместо согласования по столбцам нужно согласовывать операцию по строкам, нужно использовать арифметический метод:

```
In [145]: series3 = frame['d']

In [146]: frame
Out[146]:
          b     d     e
Utah    0.0   1.0   2.0
Ohio    3.0   4.0   5.0
Texas   6.0   7.0   8.0
Oregon  9.0  10.0  11.0

In [147]: series3
Out[147]:
Utah       1.0
Ohio       4.0
Texas      7.0
Oregon    10.0
Name: d, dtype: float64

In [148]: frame.sub(series3, axis='index')
Out[148]:
          b    d    e
Utah   -1.0  0.0  1.0
Ohio   -1.0  0.0  1.0
Texas  -1.0  0.0  1.0
Oregon -1.0  0.0  1.0
```

2.5. Применение функций и отображение

Универсальные функции NumPy также работают с объектами `pandas`:

```
In [149]: frame = pd.DataFrame(np.random.randn(4, 3), columns=list('bde'),
     ...: index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [150]: frame
Out[150]:
                b         d         e
Utah   -0.373512 -0.408758 -0.865501
Ohio    0.644210 -1.974533  0.827712
Texas  -0.002276 -0.452517  0.904933
Oregon -0.458469  0.526754 -1.517036

In [151]: np.abs(frame)
Out[151]:
                b         d         e
Utah    0.373512  0.408758  0.865501
Ohio    0.644210  1.974533  0.827712
Texas   0.002276  0.452517  0.904933
Oregon  0.458469  0.526754  1.517036
```

Другой частой операцией является применение функции к одномерным массивам для каждого столбца или строки. Метод `apply` объекта `DataFrame` выполняет это:

```
In [152]: f = lambda x: x.max() - x.min()

In [153]: frame.apply(f)
Out[153]:
b    1.102679
d    2.501287
e    2.421969
dtype: float64
```

В результате мы получили объект `Series`, у которого индекс совпадает со столбцами объекта `DataFrame`.

Если задать параметр `axis = 'columns'` в функции `apply`, функция будет применяться к строкам:

```
In [154]: frame.apply(f, axis='columns')
Out[154]:
Utah      0.491989
Ohio      2.802245
Texas     1.357449
Oregon    2.043790
dtype: float64
```

Многие из наиболее распространенных статистических методов (например, `sum` и `mean`) являются методами `DataFrame`, поэтому использование `apply` не обязательно.

Функция, передаваемая в `apply`, не обязана возвращать скалярное значение, она может также возвращать объект `Series`:

```
In [155]: frame
Out[155]:
               b         d         e
Utah    0.485727  0.656970  0.799615
Ohio    1.296945 -0.736579  0.243545
Texas  -0.953977 -1.293465  0.154692
Oregon -0.270480  0.684838  1.046487

In [156]: def f(x):
   ...:     return pd.Series([x.min(), x.max()], index=['min', 'max'])

In [157]: frame.apply(f)
Out[157]:
            b         d         e
min -0.953977 -1.293465  0.154692
max  1.296945  0.684838  1.046487
```

Также можно использовать поэлементные функции. Предположим, нужно получить форматированную строку для каждого значения в объекте `frame`. Это можно реализовать с помощью функции `applymap`:

```
In [158]: format = lambda x: '%.2f' % x
```

```
In [159]: frame.applymap(format)
Out[159]:
             b      d     e
Utah      0.49   0.66  0.80
Ohio      1.30  -0.74  0.24
Texas    -0.95  -1.29  0.15
Oregon   -0.27   0.68  1.05
```

В функции `applymap` используется метод `map` класса `Series`:

```
In [160]: frame['e'].map(format)
Out[160]:
Utah      0.80
Ohio      0.24
Texas     0.15
Oregon    1.05
Name: e, dtype: object
```

2.6. Сортировка и ранжирование

Одна из важных встроенных операций — это сортировка данных. Для того, чтобы выполнить лексикографическую сортировку по индексам строк или столбцов, можно использовать функцию `sort_index`, которая возвращает новый отсортированный объект:

```
In [161]: obj = pd.Series(range(4), index=['d', 'a', 'b', 'c'])

In [162]: obj.sort_index()
Out[162]:
a    1
b    2
c    3
d    0
dtype: int64
```

Объект `DataFrame` можно сортировать по индексам на любой оси:

```
In [163]: frame = pd.DataFrame(np.arange(8).reshape((2, 4)),
     ...: index=['three', 'one'],
     ...: columns=['d', 'a', 'b', 'c'])

In [164]: frame.sort_index()
Out[164]:
       d  a  b  c
one    4  5  6  7
three  0  1  2  3

In [165]: frame.sort_index(axis=1)
Out[165]:
       a  b  c  d
three  1  2  3  0
one    5  6  7  4
```

Данные сортируются по возрастанию по умолчанию, но могут быть отсортированы также по убыванию:

```
In [166]: frame.sort_index(axis=1, ascending=False)
Out[166]:
       d  c  b  a
three  0  3  2  1
one    4  7  6  5
```

Для сортировки объекта `Series` по значениям используется метод `sort_values`:

```
In [167]: obj = pd.Series([4, 7, -3, 2])

In [168]: obj.sort_values()
Out[168]:
2   -3
3    2
0    4
1    7
dtype: int64
```

Все пропущенные значения по умолчанию сортируются в конец объекта `Series`:

```
In [169]: obj = pd.Series([4, np.nan, 7, np.nan, -3, 2])

In [170]: obj.sort_values()
Out[170]:
4   -3.0
5    2.0
0    4.0
2    7.0
1    NaN
3    NaN
dtype: float64
```

При сортировке объекта `DataFrame` можно использовать данные в одном или нескольких столбцах в качестве ключей для сортировки. Чтобы выполнить это, необходимо передать имя одного или нескольких столбцов параметру `by` метода `sort_vlues`:

```
In [171]: frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})

In [172]: frame
Out[172]:
   b  a
0  4  0
1  7  1
2 -3  0
3  2  1

In [173]: frame.sort_values(by='b')
Out[173]:
```

```
   b  a
2 -3  0
3  2  1
0  4  0
1  7  1
```

Для сортировки по нескольким столбцам, необходимо передать список имен столбцов:

```
In [174]: frame.sort_values(by=['a', 'b'])
Out[174]:
   b  a
2 -3  0
0  4  0
3  2  1
1  7  1
```

Ранжирование заключается в присвоении *ранга* от единицы до числа значений в массиве. Объекты `Series` и `DataFrame` имеют метод `rank`, который по умолчанию разрывает связи, присваивая каждой группе среднее значение ранга:

```
In [175]: obj = pd.Series([7, -5, 7, 4, 2, 0, 4])

In [176]: obj.rank()
Out[176]:
0    6.5
1    1.0
2    6.5
3    4.5
4    3.0
5    2.0
6    4.5
dtype: float64
```

Ранги также могут быть назначены в соответствии с порядком, в котором они наблюдаются в данных:

```
In [177]: obj.rank(method='first')
Out[177]:
0    6.0
1    1.0
2    7.0
3    4.0
4    3.0
5    2.0
6    5.0
dtype: float64
```

Здесь вместо использования среднего ранга 6.5 для записей с индексами `0` и `2` они вместо этого были установлены на 6 и 7, потому что метка 0 предшествует метке 2 в данных.

В таблице 5 представлен перечень методов построения ранга.

Объект `DataFrame` может вычислять ранги по строкам или по столбцам:

```
In [178]: frame = pd.DataFrame({'b': [4.3, 7, -3, 2], 'a': [0, 1, 0, 1],
    ...: 'c': [-2, 5, 8, -2.5]})

In [179]: frame
Out[179]:
     b  a    c
0  4.3  0 -2.0
1  7.0  1  5.0
2 -3.0  0  8.0
3  2.0  1 -2.5

In [180]: frame.rank(axis='columns')
Out[180]:
     b    a    c
0  3.0  2.0  1.0
1  3.0  1.0  2.0
2  1.0  2.0  3.0
3  3.0  2.0  1.0
```

Таблица 5: Методы ранжирования

Метод	Описание
`average`	Используется по умолчанию. Присваивает среднее значение ранга каждому значению в группе
`min`	Использует минимальный ранг для всей группы
`max`	Использует максимальный ранг для всей группы
`first`	Присваивает ранги в порядке появления значений в данных
`dense`	Как `method = 'min'`, но ранги между группами всегда увеличиваются на 1, а не на количество равных элементов в группе

2.7. Индексация с повторяющимися метками

В рассматриваемых выше примерах индексы имели единственные значения, без повторений. Хотя многие функции библиотеки `pandas` (например, `reindex`) требуют, чтобы метки были уникальными, это не обязательно. Рассмотрим ряд с повторяющимися индексами:

```
In [181]: obj = pd.Series(range(5), index=['a', 'a', 'b', 'b', 'c'])

In [182]: obj
Out[182]:
a    0
a    1
b    2
b    3
c    4
dtype: int64
```

У объекта `Index` есть атрибут `is_unique`, который дает информацию являются ли метки индекса уникальными:

```
In [183]: obj.index.is_unique
Out[183]: False
```

В случае, когда несколько данных имеют одинаковые метки, обращение по этому индексу вернет объект `Series`, в то время как для меток без дублирования возвращается скалярное значение:

```
In [184]: obj['a']
Out[184]:
a    0
a    1
dtype: int64

In [185]: obj['c']
Out[185]: 4
```

Та же логика распространяется и на индексирование строк в `DataFrame`:

```
In [186]: df = pd.DataFrame(np.random.randn(4, 3), index=['a', 'a', 'b', 'b'])

In [187]: df
Out[187]:
          0         1         2
a -0.131446  1.837088 -0.618983
a  0.505283  1.940973  0.619603
b  0.209986  0.225563  0.507683
b -0.763314  0.380339  1.837083

In [188]: df.loc['b']
Out[188]:
          0         1         2
b  0.209986  0.225563  0.507683
b -0.763314  0.380339  1.837083
```

3. Описательная и сводная статистика

Объекты `pandas` оснащены набором общих математических и статистических методов. Большинство из них попадают в категорию сводной статистики. В отличие от соответствующих методов массивов NumPy методы объектов `pandas` имеют встроенную обработку пропущенных значений. Рассмотрим небольшой объект `DataFrame`:

```
In [189]: df = pd.DataFrame([[1.4, np.nan], [7.1, -4.5],
     ...: [np.nan, np.nan], [0.75, -1.3]],
     ...: index=['a', 'b', 'c', 'd'],
     ...: columns=['one', 'two'])
```

```
In [190]: df
Out[190]:
    one  two
a  1.40  NaN
b  7.10 -4.5
c   NaN  NaN
d  0.75 -1.3
```

Вызов метода `sum` возвращает суммы значений по столбцам:

```
In [191]: df.sum()
Out[191]:
one    9.25
two   -5.80
dtype: float64
```

Чтобы получить суммы значений по строкам нужно передать параметр `axis='columns'` или `axis=1`:

```
In [192]: df.sum(axis='columns')
Out[192]:
a    1.40
b    2.60
c    0.00
d   -0.55
dtype: float64
```

Значения NA исключаются, если только весь срез (в данном случае строка или столбец) не равен NA. Это поведение можно изменить с помощью параметра `skipna`:

```
In [193]: df.mean(axis='columns', skipna=False)
Out[193]:
a      NaN
b    1.300
c      NaN
d   -0.275
dtype: float64
```

Некоторые методы, такие как `idxmin` и `idxmax`, возвращают косвенную статистику, такую как значение индекса, где достигаются минимальные или максимальные значения:

```
In [194]: df.idxmax()
Out[194]:
one    b
two    d
dtype: object
```

Есть методы являются аккумулирующими:

```
In [195]: df.cumsum()
Out[195]:
    one  two
a  1.40  NaN
b  8.50 -4.5
c   NaN  NaN
d  9.25 -5.8
```

Метод `describe` возвращает множественную суммарную статистику:

```
In [196]: df.describe()
Out[196]:
            one       two
count  3.000000  2.000000
mean   3.083333 -2.900000
std    3.493685  2.262742
min    0.750000 -4.500000
25%    1.075000 -3.700000
50%    1.400000 -2.900000
75%    4.250000 -2.100000
max    7.100000 -1.300000
```

На нечисловых данных метод `describe` возвращает следующую информацию:

```
In [197]: obj = pd.Series(['a', 'a', 'b', 'c'] * 4)

In [198]: obj.describe()
Out[198]:
count     16
unique     3
top        a
freq       8
dtype: object
```

В таблице 6 представлен полный список методов сводной статистики и связанных с этим методов:

Таблица 6: Описательная и сводная статистика

Метод	Описание
`count`	Количество нечисловых значений
`describe`	Вычисляет сводную статистику для ряда или для каждого столбца объекта `DataFrame`
`min`, `max`	Вычисляет минимальное и максимальное значение
`argmin`, `argmax`	Возвращают индекс (целое число), где расположено минимальное или максимальное значение
`idxmin`, `idxmax`	Возвращают метку индекса, где расположено минимальное или максимальное значение
`quantile`	Вычисляет квантиль выборки от 0 до 1

`sum`	Сумма значений
`mean`	Среднее значение
`median`	Медиана (50-процентная квантиль) значений
`mad`	Среднее абсолютное отклонение от среднего значения
`prod`	Произведение значений
`var`	Дисперсия множества выборки значений
`std`	Стандартное отклонение выборки значений
`skew`	Асимметрия (третий момент) выборки значений
`kurt`	Эксцесс (четвертый момент) выборки значений
`cumsum`	Накопленная сумма значений
`cummin`, `cummax`	Совокупный минимум и максимум
`cumprod`	Накопленное произведение значений
`diff`	Вычисляет первую арифметическую разность (полезно для временных рядов)
`pct_change`	Вычисляет процентные изменения

4. Чтение и запись данных

В библиотеке `pandas` реализованы функции чтения табличных данных в объект `DataFrame`. В таблице 7 представлены некоторые из таких функций.

Таблица 7: Функции чтения данных

Функция	**Описание**
`read_csv`	Загружает разделенные значения из файла или файлоподобного объекта; в качестве разделителя по умолчанию используется запятая
`read_table`	Загружает разделенные значения из файла или файлоподобного объекта; в качестве разделителя по умолчанию используется табуляция (`'\t'`)
`read_fwf`	Читает данные в формате со столбцами фиксированной длины (без разделителей)
`read_clipboard`	Версия функции `read_table`, которая читает данные из буфера обмена
`read_excel`	Читает данные из файлов формата `.xls` или `.xlsx`
`read_hdf`	Читает файлы вормата HDF5, записанные с помощью библиотеки `pandas`
`read_html`	Читает все таблицы из заданного документа HTML
`read_json`	Читает данные из JSON (JavsScript Object Notation)
`read_msgpack`	Читает данные `pandas` закодированные с помощью двоичного формата MessagePack
`read_pickle`	Читает любой объект, сохраненный в формате Python pickle
`read_sas`	Читает набор данных SAS, хранящийся в одном из пользовательских форматов хранения системы SAS

`read_sql`	Читает результат запроса SQL (используя SQLAlchemy) как объект `DataFrame`

Библиотека `pandas` поддерживает нативную работу со многими реляционными БД.

Можно не только загружать данные из локальных файлов, но и из Интернета — достаточно вместо адреса на локальном компьютере указать прямю ссылку на файл.

Также существует дополнительный пакет, который называется `pandas_datareader`. Если он не установлен, его можно установить через `conda` или `pip`. Он загружает данные из некоторых источников. Загрузим с помощью `pandas_datareader` некоторые данные для некоторых биржевых тикеров:

```
import pandas_datareader.data as web

all_data = {ticker: web.get_data_yahoo(ticker)
            for ticker in ['AAPL', 'IBM', 'MSFT', 'GOOG']}
price = pd.DataFrame({ticker: data['Adj Close']
                      for ticker, data in all_data.items()})
volume = pd.DataFrame({ticker: data['Volume']
                       for ticker, data in all_data.items()})
```

Вычислим изменение процентные изменения цен:

```
In [199]: returns = price.pct_change()

In [200]: returns.tail()
Out[200]:
                AAPL       IBM      MSFT      GOOG
Date
2020-03-23 -0.021244 -0.006500 -0.009975 -0.014641
2020-03-24  0.100325  0.113011  0.090896  0.073669
2020-03-25 -0.005509  0.003508 -0.009573 -0.028181
2020-03-26  0.052623  0.066509  0.062551  0.053751
2020-03-27 -0.041402 -0.043051 -0.041061 -0.043934
```

Метод `corr` объекта `Series` вычисляет корреляцию перекрывающихся, выровненных по индексу значений в двух объектах `Series`. Соответственно `cov` вычисляет ковариацию:

```
In [202]: returns['MSFT'].corr(returns['IBM'])
Out[202]: 0.5840961357368786

In [203]: returns['MSFT'].cov(returns['IBM'])
Out[203]: 0.00015025941925307093
```

Поскольку `MSFT` является допустимым атрибутом Python, мы также можем выбрать эти столбцы, используя более краткий синтаксис

```
In [204]: returns.MSFT.corr(returns.IBM)
Out[204]: 0.5840961357368786
```

Методы `corr` и `cov` объекта `DataFrame` возвращают полные матрицы корреляции или ковариации:

```
In [205]: returns.corr()
Out[205]:
          AAPL       IBM      MSFT      GOOG
AAPL  1.000000  0.520795  0.688470  0.627134
IBM   0.520795  1.000000  0.584096  0.513872
MSFT  0.688470  0.584096  1.000000  0.738942
GOOG  0.627134  0.513872  0.738942  1.000000

In [206]: returns.cov()
Out[206]:
          AAPL       IBM      MSFT      GOOG
AAPL  0.000315  0.000140  0.000208  0.000185
IBM   0.000140  0.000229  0.000150  0.000129
MSFT  0.000208  0.000150  0.000290  0.000209
GOOG  0.000185  0.000129  0.000209  0.000275
```

Используя метод `corrwith` объекта `DataFrame`, можно вычислять попарные корреляции между столбцами или строками `DataFrame` с другими объектами `Series` или `DataFrame`. Передача в качестве аргумента ряда возвращает ряд со значением корреляции, вычисленным для каждого столбца:

```
In [207]: returns.corrwith(returns.IBM)
Out[207]:
AAPL    0.520795
IBM     1.000000
MSFT    0.584096
GOOG    0.513872
dtype: float64
```

5. Задания

Предлагается поработать с набором данных о преступности в Лос-Анджелесе.

Результат работы должен быть в виде блокнота Jupyter. Все задачи в одном блокноте, например, `pandas-da-ans.ipynb`.

Для построения графических данных смотрите главу Графическая визуализация данных

5.1. Быстрый анализ данных

- Загрузите случайную выборку из этого набора.
- Сколько строк и столбцов в таблице?
- Каковы названия столбцов?
- Какие типы данных у столбцов?
- Сколько в каждом из них уникальных значений?

- Сколько пропущенных значений?
- Постройте распределения числовых переменных?

5.2. Жертвы

В наборе данных имеется информация о Возрасте, Поле, и Происхождении каждой жертвы. Есть ли связь между этими признаками?

- Верно ли, что женщины чаще оказываются жертвами по сравнению с мужчинами?

5.3. Преступления, пол и возраст

- Изучите распределение количества преступлений по возрасту. Какова тенденция? Люди какого возраста чаще всего подвергаются преступлениям? Есть ли локальные минимумы? Используйте типы графиков `hist` и `density`.
- Как различается вероятность женщин и мужчин стать жертвой в зависимости от возраста? Постройте визуализацию. В каком возрастном промежутке мужчины чаще становятся жетрвами преступлений?
- Определите 10 самых распространенных преступлений в Лос-Анджелесе. Постройте график.
- От каких преступлений чаще старадют женщины, а от каких мужчины?

5.4. Происхождение

Символ	Происхождение
'A'	Other Asian
'B'	Black
'C'	Chinese
'D'	Cambodian
'F'	Filipino
'G'	Guamanian
'H'	Hispanic/Latin/Mexican
'I'	American Indian/Alaskan Native
'J'	Japanesea
'K'	Korean
'L'	Laotian

'O'	Other
'P'	Pacific Islander
'S'	Samoan
'U'	Hawaiian
'V'	Vietnamese
'W'	White
'X'	Unknown
'Z'	Asian Indian

- *Люди какого происхождения чаще всего подвергаются преступлениям?*

5.5. Место проишествия

- Отсортируйте районы, по количеству преступлений. Постройте график, показывающий самые безопасный и опасный районы.
- *Люди какого происхождения чаще всего страдают от преступлений в каждом из районов? Не забудьте нормировать на общее количество жертв.*