Агрегирование и группировка

https://dfedorov.spb.ru

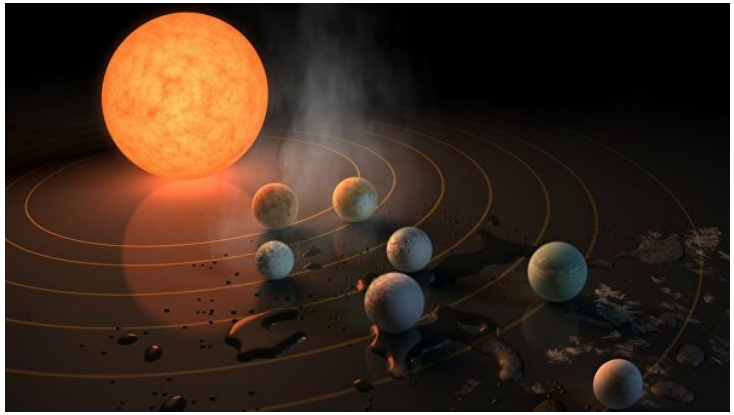
Воспользуемся набором данных «Планеты» (Planets), доступным через пакет Seaborn. Он включает информацию об открытых астрономами планетах, вращающихся вокруг других звезд, известных под названием внесолнечных планет или экзопланет (exoplanets). Скачать его можно с помощью команды пакета Seaborn:

```
>>> import seaborn as sns
>>> planets = sns.load_dataset('planets')
>>> planets.shape
```

```
(1035, 6)
```

```
>>> planets.head()
```

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009



Этот набор данных содержит определенную информацию о более чем 1000 экзопланет, открытых до 2014 года.

Доплеровская спектроскопия

Параметры

Простое агрегирование в библиотеке Pandas

Как и в случае одномерных массивов библиотеки NumPy, для объектов Series библиотеки Pandas агрегирующие функции возвращают скалярное значение.

В случае объекта DataFrame по умолчанию агрегирующие функции возвращают сводные показатели по каждому столбцу.

Можно вместо этого агрегировать и по строкам, задав аргумент axis.

Объекты Series и DataFrame библиотеки Pandas содержат методы, соответствующие всем распространенным агрегирующим функциям. В них есть удобный метод describe() , вычисляющий сразу несколько самых распространенных сводных показателей для каждого столбца и возвращающий результат.
Опробуем его на наборе данных «Планеты», пока удалив строки с отсутствующими значениями:

```
>>> planets.dropna().describe()
```

	number	orbital_period	mass	distance	year
count	498.00000	498.000000	498.000000	498.000000	498.000000
mean	1.73494	835.778671	2.509320	52.068213	2007.377510
std	1.17572	1469.128259	3.636274	46.596041	4.167284
min	1.00000	1.328300	0.003600	1.350000	1989.000000
25%	1.00000	38.272250	0.212500	24.497500	2005.000000
50%	1.00000	357.000000	1.245000	39.940000	2009.000000
75%	2.00000	999.600000	2.867500	59.332500	2011.000000
max	6.00000	17337.500000	25.000000	354.000000	2014.000000

Эта возможность очень удобна для первоначального знакомства с общими характеристиками набора данных. Например, мы видим в столбце **year** , что, хотя первая экзопланета была открыта еще в 1989 году, половина всех известных экзопланет открыта не ранее 2010 года.

В значительной степени мы обязаны этим миссии «Кеплер», представляющей собой космический телескоп, специально разработанный для поиска затмений от планет, вращающихся вокруг других звезд.

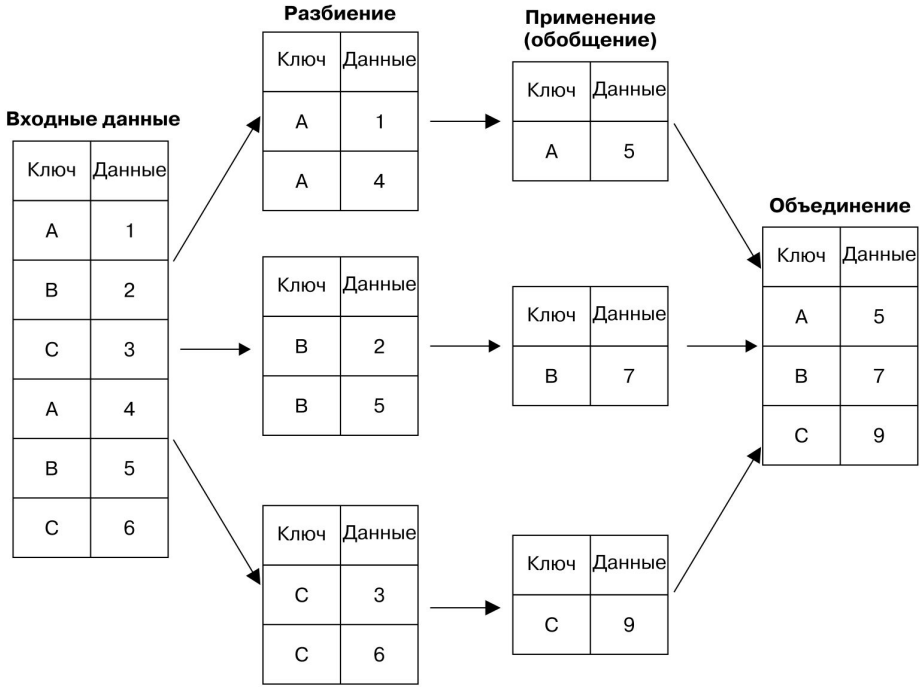
Агрегирующая функция	**Описание**
count()	Общее количество элементов
first(), last()	Первый и последний элементы
mean(), median()	Среднее значение и медиана
min(), max()	Минимум и максимум
std(), var()	Стандартное отклонение и дисперсия
mad()	Среднее абсолютное отклонение
prod()	Произведение всех элементов
sum()	Сумма всех элементов

Это все методы объектов DataFrame и Series .

GroupBy: разбиение, применение, объединение

Простые агрегирующие функции дают возможность «прочувствовать» набор данных, но зачастую бывает нужно выполнить условное агрегирование по какой-либо метке или индексу. Это действие реализовано в так называемой операции **GroupBy**.
Название group by («сгруппировать по») ведет начало от одноименной команды в языке SQL баз данных, но, возможно, будет понятнее говорить о ней в терминах, придуманных Хэдли Викхэмом, более известным своими библиотеками для языка R: **разбиение**, **применение** и **объединение**.

- **Шаг разбиения** включает разделение на части и группировку объекта DataFrame на основе значений заданного ключа.
- **Шаг применения** включает вычисление какой-либо функции, обычно агрегирующей, преобразование или фильтрацию в пределах отдельных групп.
- На **шаге объединения** выполняется слияние результатов этих операций в выходной массив.



Хотя мы, конечно, могли бы сделать это вручную с помощью какого-либо сочетания описанных выше команд маскирования, агрегирования и слияния, важно понимать, что не обязательно создавать объекты для промежуточных разбиений. Операция GroupBy может проделать все это за один проход по данным, вычисляя сумму, среднее значение, количество, минимум и другие сводные показатели для каждой группы. Мощь операции GroupBy состоит в абстрагировании этих шагов: пользователю не нужно заботиться о том, как фактически выполняются вычисления, а можно вместо этого думать об операции в целом.

```
>>> df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],
                       'data': range(6)}, columns=['key', 'data'])
>>> df
```

	key	data
0	A	0
1	B	1
2	C	2
3	A	3
4	B	4
5	C	5

Имя ключевого столбца

```
>>> df.groupby('key')
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fef3c70b8b0>
```

```
>>> df.groupby('key').sum()
```

	data
key	
A	3
B	5
C	7

Метод sum() — лишь один из возможных вариантов. Здесь можно использовать практически любую распространенную агрегирующую функцию библиотек Pandas или NumPy, равно как и практически любую корректную операцию объекта DataFrame .

Обратите внимание, что возвращаемое — не набор объектов DataFrame , а объект **DataFrameGroupBy**. Этот объект особенный, его можно рассматривать как специальное представление объекта DataFrame , готовое к группировке, но не выполняющее никаких фактических вычислений до этапа применения агрегирования. Подобный метод «отложенного вычисления» означает возможность очень эффективной реализации распространенных агрегирующих функций, причем практически прозрачным для пользователя образом.

Объект GroupBy

Объект GroupBy — очень гибкая абстракция. Во многом с ним можно обращаться как с коллекцией объектов DataFrame, и вся сложность будет скрыта от пользователя. Рассмотрим примеры на основе набора данных «Планеты».

Индексация по столбцам

Объект GroupBy поддерживает индексацию по столбцам аналогично объекту DataFrame , с возвратом модифицированного объекта GroupBy .

```
>>> planets.groupby('method')
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fef0f3207c0>

>>> planets.groupby('method')['orbital_period']
<pandas.core.groupby.generic.SeriesGroupBy object at 0x7fef0f2fc0a0>
```

Здесь мы выбрали конкретную группу Series из исходной группы DataFrame, сославшись на соответствующее имя столбца.
Как и в случае с объектом GroupBy, никаких вычислений не происходит до вызова для этого объекта какого-нибудь агрегирующего метода.

```
>>> planets.groupby('method')['orbital_period'].median()
method
Astrometry                         631.180000
Eclipse Timing Variations         4343.500000
Imaging                          27500.000000
Microlensing                      3300.000000
Orbital Brightness Modulation        0.342887
Pulsar Timing                       66.541900
Pulsation Timing Variations       1170.000000
Radial Velocity                    360.200000
Transit                              5.714932
Transit Timing Variations           57.011000
Name: orbital_period, dtype: float64
```

Результат дает нам общее представление о масштабе чувствительности каждого из методов к периодам обращения (в днях).

Цикл по группам

Объект GroupBy поддерживает непосредственное выполнение циклов по группам с возвратом каждой группы в виде объекта Series или DataFrame:

```
>>> for (method, group) in planets.groupby('method'):
        print("{0:30s} shape={1}".format(method, group.shape))
```

```
Astrometry                     shape=(2, 6)
Eclipse Timing Variations      shape=(9, 6)
Imaging                        shape=(38, 6)
Microlensing                   shape=(23, 6)
Orbital Brightness Modulation  shape=(3, 6)
Pulsar Timing                  shape=(5, 6)
Pulsation Timing Variations    shape=(1, 6)
Radial Velocity                shape=(553, 6)
Transit                        shape=(397, 6)
Transit Timing Variations      shape=(4, 6)
```

Методы диспетчеризации

Все методы, не реализованные явным образом объектом GroupBy , будут передаваться далее и выполняться для групп, вне зависимости от того, являются ли они объектами Series или DataFrame . Например, можно использовать метод describe() объекта DataFrame для вычисления набора сводных показателей, описывающих каждую группу в данных:

```
>>> planets.groupby('method')['year'].describe()
```

method	count	mean	std	min	25%	50%	75%	max
Astrometry	2.0	2011.500000	2.121320	2010.0	2010.75	2011.5	2012.25	2013.0
Eclipse Timing Variations	9.0	2010.000000	1.414214	2008.0	2009.00	2010.0	2011.00	2012.0
Imaging	38.0	2009.131579	2.781901	2004.0	2008.00	2009.0	2011.00	2013.0
Microlensing	23.0	2009.782609	2.859697	2004.0	2008.00	2010.0	2012.00	2013.0
Orbital Brightness Modulation	3.0	2011.666667	1.154701	2011.0	2011.00	2011.0	2012.00	2013.0
Pulsar Timing	5.0	1998.400000	8.384510	1992.0	1992.00	1994.0	2003.00	2011.0
Pulsation Timing Variations	1.0	2007.000000	NaN	2007.0	2007.00	2007.0	2007.00	2007.0
Radial Velocity	553.0	2007.518987	4.249052	1989.0	2005.00	2009.0	2011.00	2014.0
Transit	397.0	2011.236776	2.077867	2002.0	2010.00	2012.0	2013.00	2014.0
Transit Timing Variations	4.0	2012.500000	1.290994	2011.0	2011.75	2012.5	2013.25	2014.0

Эта таблица позволяет получить лучшее представление о наших данных. Например, большинство планет было открыто методом измерения лучевой скорости (radial velocity method) и транзитным методом (transit method), хотя последний стал распространенным благодаря новым более точным телескопам только в последнее десятилетие. Похоже, что новейшими методами являются метод вариации времени транзитов (transit timing variation method) и метод модуляции орбитальной яркости (orbital brightness modulation method), которые до 2011 года не использовались для открытия новых планет.

Это всего лишь один пример полезности методов диспетчеризации. Обратите внимание, что они применяются к каждой отдельной группе, после чего результаты объединяются в объект GroupBy и возвращаются. Можно использовать для соответствующего объекта GroupBy любой допустимый метод объектов Series / DataFrame , что позволяет выполнять многие весьма гибкие и мощные операции!

Агрегирование, фильтрация, преобразование, применение

В частности, у объектов GroupBy имеются методы **aggregate**() , **filter**() , **transform**() и **apply**() , эффективно выполняющие множество полезных операций до объединения сгруппированных данных.

```
>>> rng = np.random.RandomState(0)
>>> df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],
                       'data1': range(6),
                       'data2': rng.randint(0, 10, 6)},
>>> columns = ['key', 'data1', 'data2'])
>>> df
```

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

Агрегирование

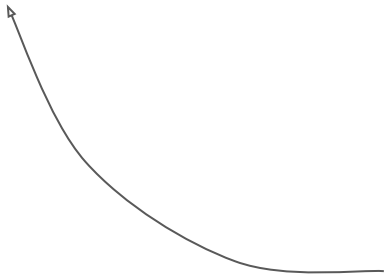
Метод aggregate() обеспечивает еще большую гибкость.
Он может принимать на входе строку, функцию или список и вычислять все сводные показатели сразу. Вот пример, включающий все вышеупомянутое:

```
>>> df.groupby('key').aggregate(['min', np.median, max])
```

	data1			data2		
	min	median	max	min	median	max
key						
A	0	1.5	3	3	4.0	5
B	1	2.5	4	0	3.5	7
C	2	3.5	5	3	6.0	9

```
>>> df.groupby('key').aggregate({'data1': 'min', 'data2': 'max'})
```

	data1	data2
key		
A	0	5
B	1	7
C	2	9



Еще один удобный паттерн — передача словаря, связывающего имена столбцов с операциями, которые должны быть применены к этим столбцам.

Фильтрация

Операция фильтрации дает возможность опускать данные в зависимости от свойств группы.
Например, нам может понадобиться оставить в результате все группы, в которых стандартное отклонение превышает какое-либо критическое значение:

```
>>> df
```

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

```
>>> df.groupby('key').std()
```

key	data1	data2
A	2.12132	1.414214
B	2.12132	4.949747
C	2.12132	4.242641

```
>>> df.groupby('key').filter(lambda x: x['data2'].std() > 4)
```

	key	data1	data2
1	B	1	0
2	C	2	3
4	B	4	7
5	C	5	9

Функция **filter()** возвращает булево значение, определяющее, прошла ли группа фильтрацию. В данном случае, поскольку стандартное отклонение группы A не превышает 4, она удаляется из итогового результата.

Преобразование

В то время как агрегирующая функция должна возвращать сокращенную версию данных, преобразование может вернуть версию полного набора данных, преобразованную ради дальнейшей их перекомпоновки. При подобном преобразовании форма выходных данных совпадает с формой входных. Распространенный пример — центрирование данных путем вычитания среднего значения по группам:

```
>>> df
```

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

```
>>> df.groupby('key').transform(lambda x: x - x.mean())
```

	data1	data2
0	-1.5	1.0
1	-1.5	-3.5
2	-1.5	-3.0
3	1.5	-1.0
4	1.5	3.5
5	1.5	3.0

Метод apply()

Метод apply() позволяет применять произвольную функцию к результатам группировки.

```
>>> def norm_by_data2(x):
        # x – объект DataFrame сгруппированных значений
        x['data1'] = x['data1'] / x['data2'].sum()
        return x

>>> df.groupby('key').apply(norm_by_data2)
```

	key	data1	data2
0	A	0.000000	5
1	B	0.142857	0
2	C	0.166667	3
3	A	0.375000	3
4	B	0.571429	7
5	C	0.416667	9

```
>>> df
```

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

Функция apply() в GroupBy достаточно гибка. Единственное требование, чтобы она принимала на входе объект DataFrame и возвращала объект библиотеки Pandas или скалярное значение; что вы делаете внутри, остается на ваше усмотрение!

Задание ключа разбиения

Список, массив, объект Series и индекс как ключи группировки

```
>>> df
```

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

```
>>> L = [0, 1, 0, 1, 2, 0]
```

```
>>> df.groupby(L).sum()
```

	data1	data2
0	7	17
1	4	3
2	4	7

```
>>> df.groupby(df['key']).sum()
```

	data1	data2
key		
A	3	8
B	5	7
C	7	12

Есть еще один, несколько более длинный способ выполнить вышеприведенную операцию df.groupby('key')

Словарь или объект Series, связывающий индекс и группу

Еще один метод - указать словарь, задающий соответствие значений индекса и ключей группировки:

```
>>> df
```

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

```
>>> df2 = df.set_index('key')
>>> df2
```

key	data1	data2
A	0	5
B	1	0
C	2	3
A	3	3
B	4	7
C	5	9

```
>>> mapping = {'A': 'vowel', 'B': 'consonant', 'C': 'consonant'}

>>> df2.groupby(mapping).sum()
```

	data1	data2
consonant	12	19
vowel	3	8

Любая функция языка Python

Аналогично заданию соответствия можно передать функции groupby любую функцию, принимающую на входе значение индекса и возвращающую группу:

```
>>> df2
```

key	data1	data2
A	0	5
B	1	0
C	2	3
A	3	3
B	4	7
C	5	9

```
>>> df2.groupby(str.lower).mean()
```

	data1	data2
a	1.5	4.0
b	2.5	3.5
c	3.5	6.0

Список допустимых ключей

Можно комбинировать любые из предыдущих вариантов ключей для группировки по мультииндексу:

```
>>> mapping = {'A': 'vowel', 'B': 'consonant', 'C': 'consonant'}
>>> df2.groupby([str.lower, mapping]).mean()
```

		data1	data2
a	vowel	1.5	4.0
b	consonant	2.5	3.5
c	consonant	3.5	6.0