**Глава 5**

**Первое знакомство с pandas**

Библиотека pandas будет основным предметом изучения в последующих главах. Она содержит высокоуровневые структуры данных и средства манипуляции ими, спроектированные так, чтобы обеспечить простоту и высокую скорость анализа данных на Python. Эта библиотека построена поверх NumPy, поэтому cii легко пользоваться в приложениях, ориентированных на NumPy.

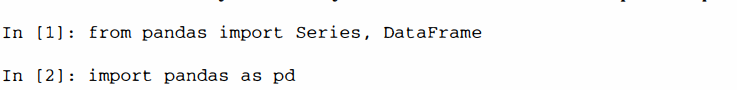
Для сведения - я начал разрабатывать pandas в начале 2008 года, когда работал в компании AQR, занимающейся количественным анализом инвестиционных рисков. Тогда я столкнулся с требованиями, которые не мог полностью удовлетворить ни один из имевшихся в моем распоряжении инструментов:

* Структуры данных с помеченными осями, поддерживающие автоматическое или явное выравнивание. Это помогает избежать типичных ошибок, связанных с невыровненностью данных и работой с данными, поступивши­ ми из разных источников и по-разному проиндексированных.
* Встроенная функциональность для работы с временными рядами.
* Одни и те же структуры должны быть пригодны для обработки как временных рядов, так и данных иного характера.
* Арифметические операции и операции редуцирования (например, суммирование вдоль оси) должны пробрасывать метаданные (метки осей).
* Гибкая обработка отсутствующих данных.
* Объединение и другие реляционные операции, имеющиеся в популярных базах данных (например, на основе SQL).

Я хотел, чтобы все это было сосредоточено в одном месте и предпочтительно написано на языке, подходящем для разработки программ общего назначения. Python казался неплохим кандидатом, но в то время в нем не было встроенных структур данных и инструментов, поддерживающих нужную мне функциональность.

За прошедшие четыре года paшlas превратилась в довольно обширную библиотеку, способную решать куда более широкий круг задач обработки данных, чем я планировал первоначально, но расширялась она, не принося в жертву простоту и удобство использования, к которым я стремился с самого начала. Надеюсь, что, прочитав эту книгу, вы так же, как и я, станете считать ее незаменимым инструментом.

В этой книге используются следующие соглашения об импорте для pandas:



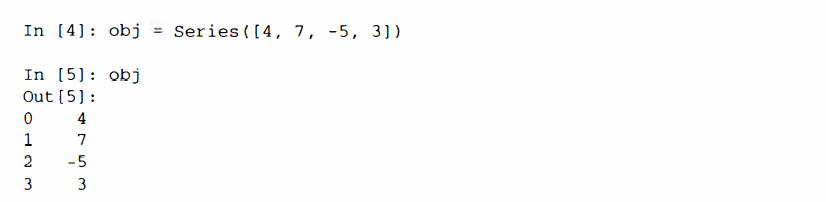
Таким образом, увидев в коде строку pd., знайте, что это ссылка на pandas. Объекты Series и DataFrame используются так часто, что я счел полезным импортировать их в локальное пространство имен.

Введение в структуры данных pandas

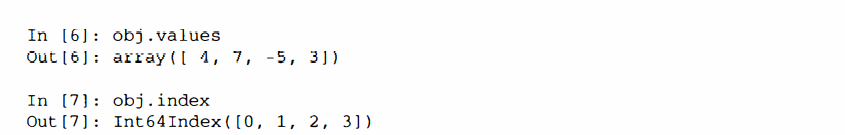
Чтобы начать работу с pandas, вы должны освоить две основные структуры данных: *Saies* и *DataFmme.* Они, конечно, не являются универсальным решением любой задачи, но все же образуют солидную и простую для использования основу большинства приложений.

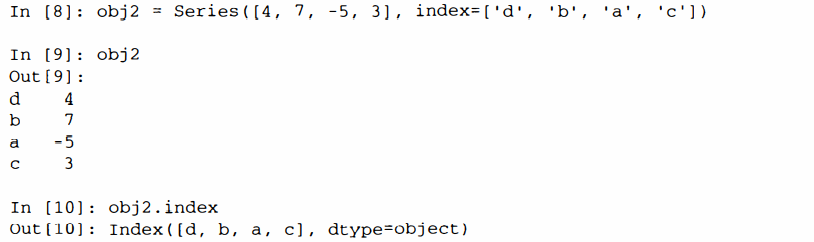
Объект Series

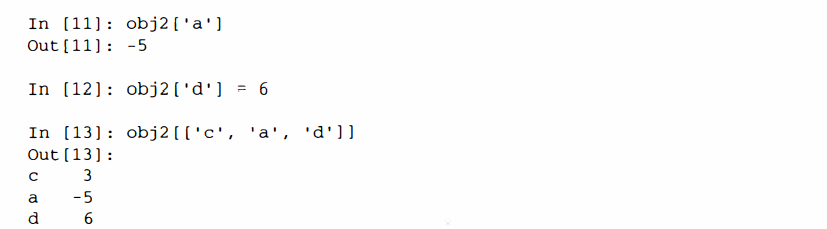
Series - одномерный похожий на массив объект, содержащий массив данных (любого типа, поддерживаемого NumPy) и ассоциированный с ним массив меток, который называется *индексо.м.* Простейший объект Series состоит только из мас­ сива данных:

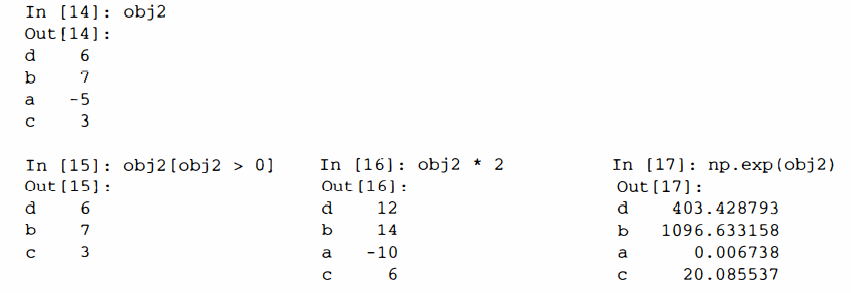


П строковом представлении Series, отображаемом в интерактивном режиме, индекс находится слева, а значения справа. Поскольку мы не задали индекс для данных, то по умолчанию создается индекс, состоящий из целых чисел от О до *N* - 1 (где *N* - длина массива данных). Имея объект Series, получить представление самого массива и его индекса можно с помощью атрибутов values и index соответственно:

Часто желательно создать объект Series с индексом, идентифицирующим каждый элемент данных:

В отличие от обычного массива NumPy, для выборки одного или нескольких элементов из объекта Series можно использовать значения индекса:

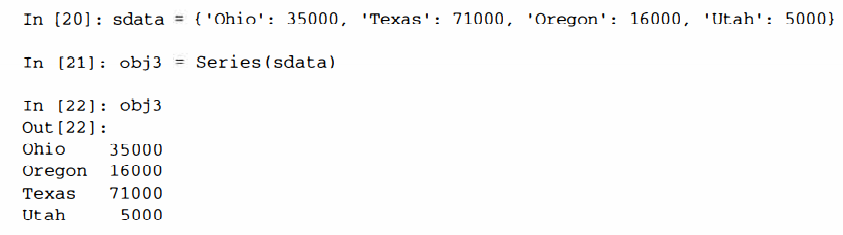
Операции с массивом NuшPy, например фильтрация с помощью булева массива, скалярное умножение или применение математических функций, сохраняют связь между индексом и значением:



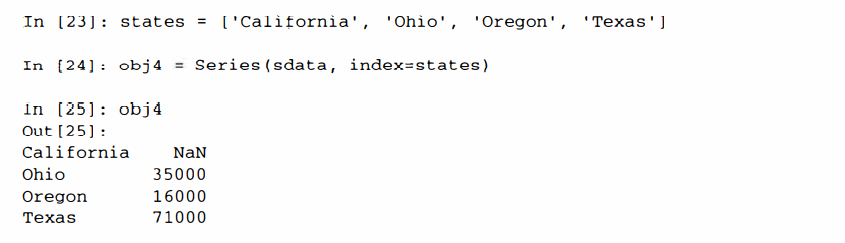
Объект Series можно также представлять себе как упорядоченный словарь фиксированной длины, поскольку он отображает индекс на данные. Его можно передавать многим функциям, ожидающим получить словарь:



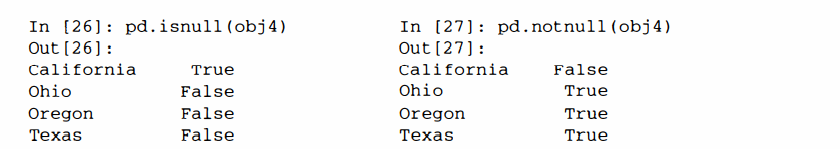
Если имеется словарь Python, содержащий данные, то из него можно создать объект Series:



Если передается только словарь, то в получившемся объекте Series ключи будут храниться в индексе по порядку:



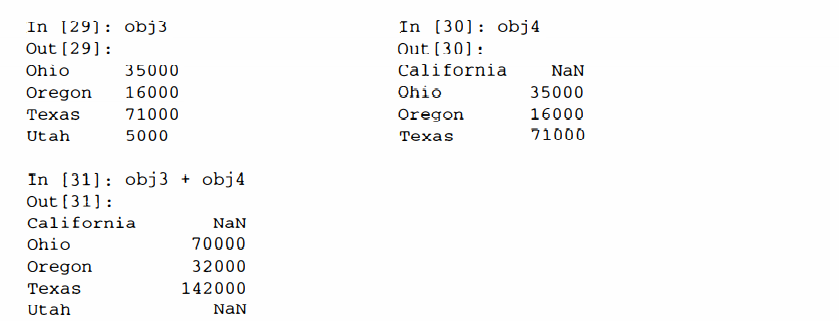
В данном случае 3 значения, найденные в sdata, помещены в соответствующие им позиции, а для метки «California» никакого значения не нашлось, поэтому ей соответствует признак NaN (не число), которым в pandas обозначаются отсутствующие значения. Иногда, говоря об отсутствующих данных, я буду употреблять термин «NA». Для распознавания отсутствующих данных в pandas следует использовать функции isnull и notnull:



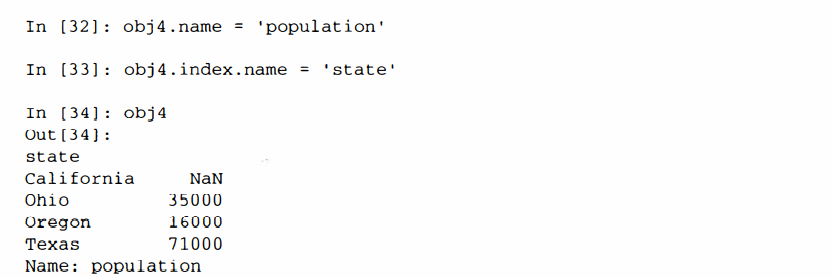
У объекта Series есть также методы экземпляра:



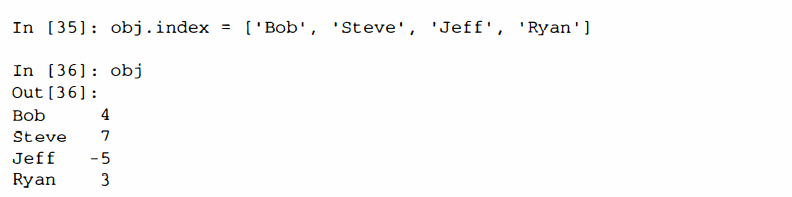
Более подробно работа с отсутствующими данными будет обсуждаться ниже в этой главе. Для многих приложений особенно важно, что при выполнении арифметических операций объект Series автоматически выравнивает данные, которые проиндексированы в разном порядке:



Вопрос о выравнивании данных будет рассмотрен отдельно. И у самого объекта Series, и у его индекса имеется атрибут name, тесно связанный с другими частями функциональности pandas:



Индекс объекта Series можно изменить на месте с помощью присваивания:

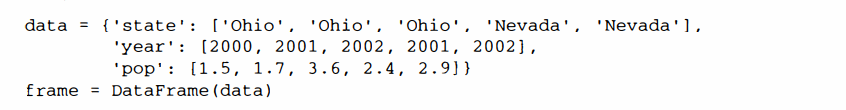


**Объект DataFrame**

Объект DataFrame представляет табличную структуру данных, состоящую из упорядоченной коллекции столбцов, причем типы значений (числовой, строковый, булев и т. д.) в разных столбцах могут различаться. В объекте DataFгame хранятся два индекса: по строкам и по столбцам. Можно считать, что это словар1, объектов Series. По сравнению с другими похожими на DataFrame структурами, которые вам могли встречаться раньше (например, data. frame в языке R), операции со строками и столбцами в DataFrame в первом приближении симметричны. Внутри объекта данные хранятся в виле одного или нескольких двумерных блоков, а не в виде списка, словаря или еще какой-нибудь коллекции одномерных массивов. Технические детали внутреннего устройства DataFrame выходят за рамки этой книги.

Хотя в DataFrame данные хранятся в двумерном формате, в виде таблицы, нетрудно представить и данные более высокой размерности, если воспользоваться иерархическим индексированием. Эту тему мы обсудим в следующем разделе, она лежит в основе многих продвинутых механизмов обработки данных в pandas.

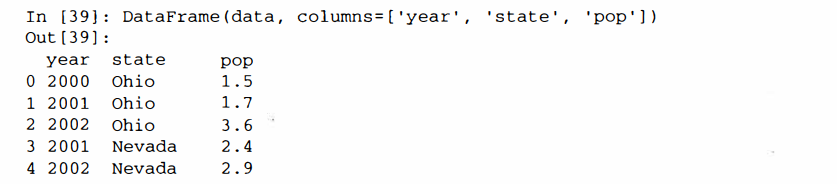
Есть много способов сконструировать объект DataFrame, один из самых распространенных - па основе словаря списков одинаковой длины или массивов NumPy:



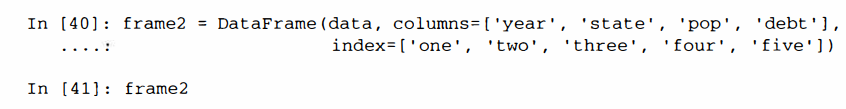
Для получившегося DataFrame автоматически будет построен индекс, как и в случае Series, и столбцы расположатся по порядку:

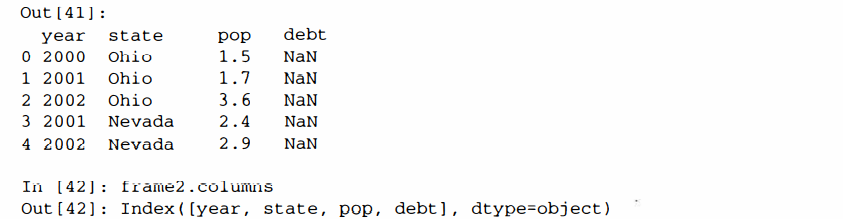


Если задать последовательность столбцов, то столбцы DataFrame расположатся строго в указанном порядке

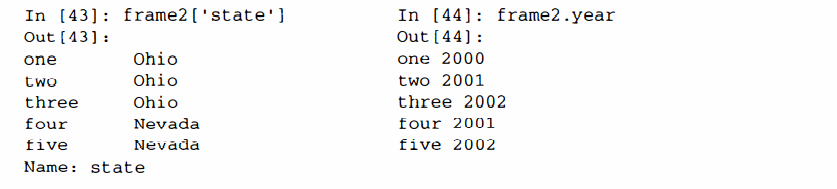


Как и в случае Series, если запросить столбец, которого нет в data, то он будет заполнен значениями NaN:





Столбец DataFrame можно извлечь как объект Series, воспользовавшись нотацией словарей, или с помощью атрибута:

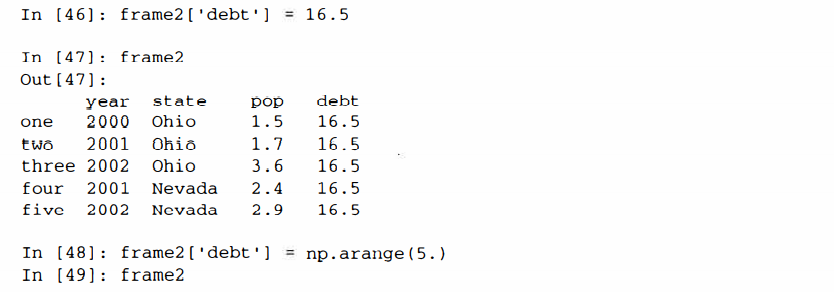


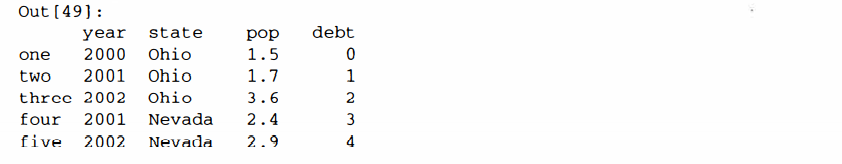
Отметим, что возвращенный объект Series имеет тот же индекс, что и DataFrame, а его атрибут name установлен соответствующим образом.

Строки также можно извлечь по позиции или по имени, для чего есть два метода, один из них - ix с указанием индексного поля (подробнее об этом ниже):

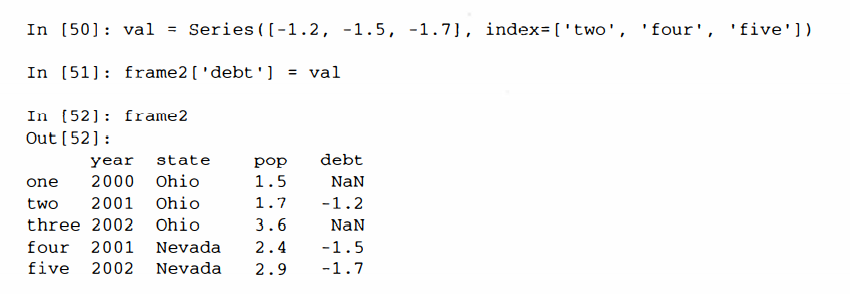


Столбцы можно модифицировать путем присваивания. Например, пустому столбцу «debt» можно было бы присвоить скалярное значение или массив значений:

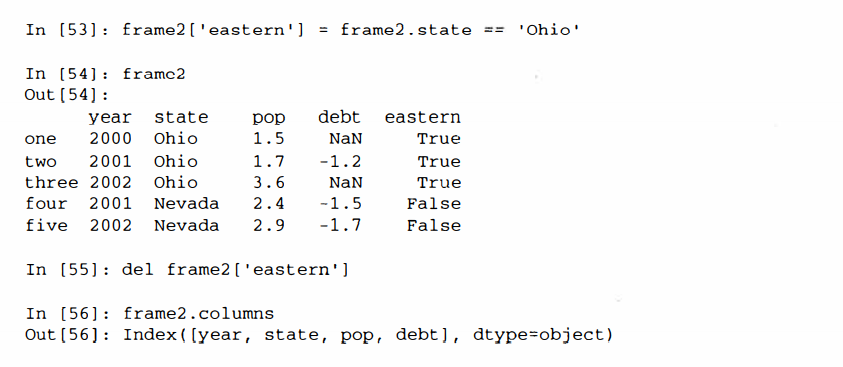


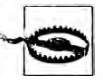


Когда столбцу присваивается список или массив, длина значения должна совпадать с длиной DataFrame. Если же присваивается объект Series, то он будет точно согласован с индексом DataFrame, а в «дырки» будут вставлены значения NA:

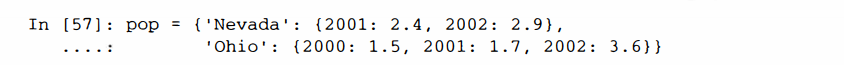


Присваивание несуществующему столбцу приводит к созданию нового столбца. Для удаления столбцов служит ключевое слово del, как и в обычном словаре:

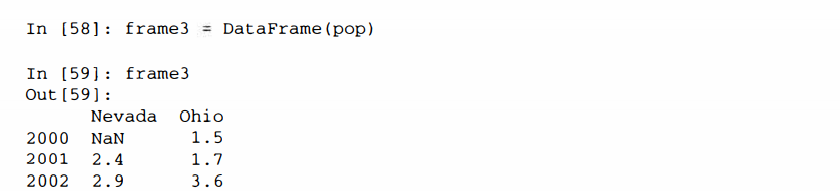


Столбец, возвращенный в ответ на запрос к DataFrame по индексу, является представлением, а не копией данных. Следовательно, любые модификации этого объекта Series, найдут отражение в DataFrame. Чтобы скопировать столбец, нужно вызвать метод сору объекта Serie.

Еще одна распространенная форма данных - словарь словарей:



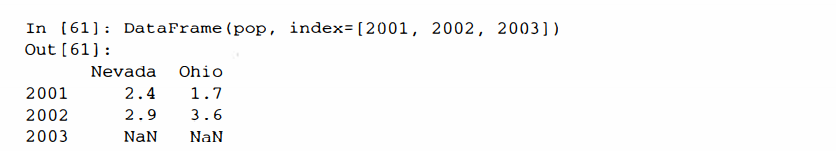
Если передать его конструктору DataFrarne, то ключи внешнего словаря будут интерпретированы как столбцы, а ключи внутреннего словаря - как индексы строк:



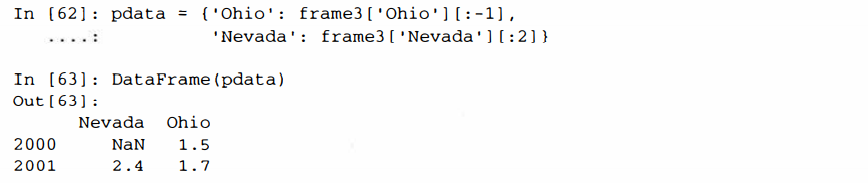
Разумеется, результат можно транспонировать:



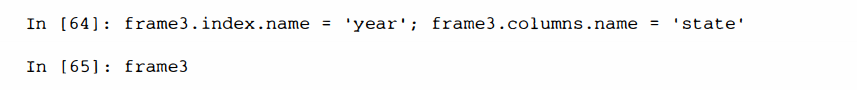
Ключи внутренних словарей объединяются и сортируются для образования индекса результата. Однако этого не происходит, если индекс задан явно:

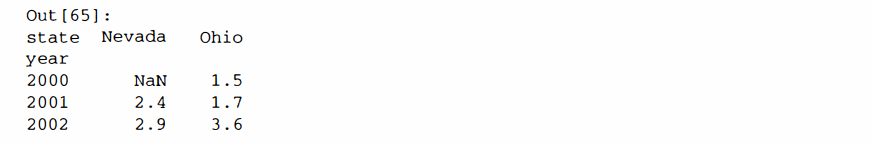


Словари объектов Series интерпретируются очень похоже:



Полный перечень возможных аргументов конструктора DataFrarne приведен в табл. 5.1. Если у объектов, возвращаемых при обращении к атрибутам index и colurnns объекта DataFraшe, установлен атрибут name, то он также выводится:

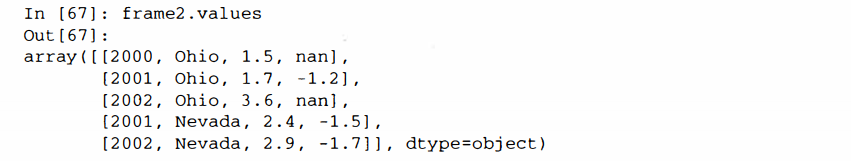




Как и в случае Series, атрибут values возвращает данные, хранящиеся в DataFraшe, в виде двумерного массива ndarray:



Если у столбцов DataFramc разные типы данных, то dtype массива values будет выбран так, чтобы охватить все столбцы:

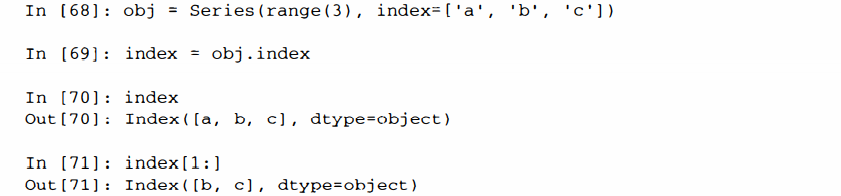


**Таблица 5.1.** Аргументы конструктора DataFrame

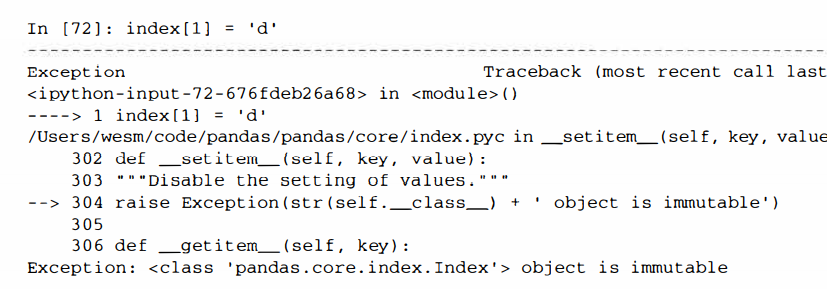
|  |  |
| --- | --- |
| **Тип** | **Примечания** |
| Двумерный ndarray | Матрица данных, дополнительно можно передать метки строк и столбцов |
| Словарь массивов, списков или кортежей | Каждая последовательность становится столбцом объекта DataFrame. Все последовательности должны быть одинаковой длины |
| Структурный массив NumPy | Интерпретируется так же, как «словарь массивов» |
| Словарь объектов Series | Каждое значение становится столбцом. Если индекс явно не задан, то индексы объектов Series объединяются и образуют индекс строк результата |
| Словарь словарей | Каждый внутренний словарь становится столбцом. Ключи объединяются и образуют индекс строк, как в случае «словаря объектов Series» |
| Список словарей или объектов Series | Каждый элемент списка становится строкой объекта DataFrame. Объединение ключей словаря или индексов объектов Series становится множеством меток столбцов DataFrame |
| Список списков или кортежей | Интерпретируется так же, как «двумерный ndarray» |
| Другой объект DataFrame | Используются индексы DataFrame, если явно не заданы другие индексы |
| Объект NumPy MaskedArray | Как «двумерный ndarray» с тем отличием, что замаскированные значения становятся отсутствующими в результирующем объекте DataFrame |

**Индексные объекты**

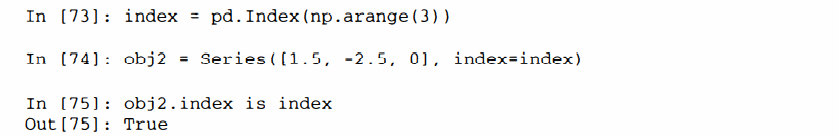
В индексных объектах pandas хранятся метки вдоль осей и прочие метаданные (например, имена осей). Любой массив или иная последовательность меток, указанная при конструировании Series или DataFrame, преобразуется в объект Index:



Индексные объекты неизменяемы, т. е. пользователь не может их модифицировать:



Неизменяемость важна для того, чтобы несколько структур данных могли совместно использовать один и тот же индексный объект, не опасаясь его повредить:



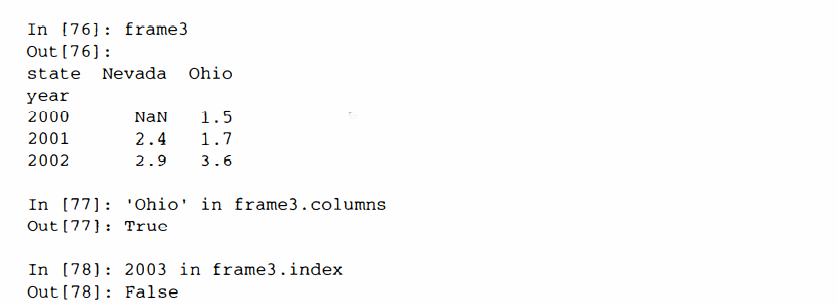
В табл. 5.2 перечислены включенные в библиотеку индексные классы. При некотором усилии можно даже создать подклассы класса Index, если требуется реализовать специализированную функциональность индексирования вдоль оси.

Многим пользователям подробная информация об индексных объектах не нужна, но они, тем не менее, являются важной частью модели данных pandas.

**Таблица 5.2**. Основные индексные объекты в pandas

|  |  |
| --- | --- |
| **Тип** | **Примечания** |
| Index | Наиболее общий индексный объект, представляющий оси в массиве NumPy, состоящем из объектов Python |
| Int64Index | Специализированный индекс для целых значений |
| Multiindex | «Иерархический» индекс, представляющий несколько уровней индексирования по одной оси. Можно считать аналогом массива кортежей |
| Datetimeindex | Хранит временные метки с наносекундной точностью (представлены типом данных NumPy datetime64) |
| Periodindex | Специализированный индекс для данных о периодах (временных промежутках) |

Индексный объект не только похож на массив, но и ведет себя как множество фиксированного размера:



У любого объекта Index есть ряд свойств и методов для ответа на типичные вопросы о хранящихся в нем данных. Они перечислены в табл. 5.3.

**Таблица 5.3**. Методы и свойства объекта lndex

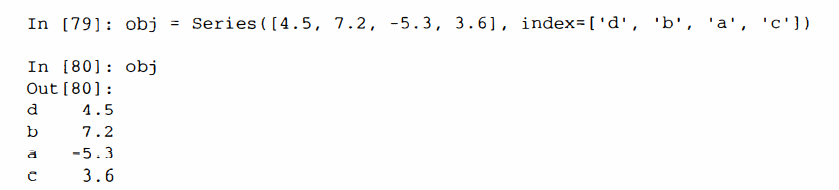
|  |  |
| --- | --- |
| **Метод** | **Описание** |
| append | Конкатенирует с дополнительными индексными объектами, порождая новый объект lndex |
| diff | Вычисляет теоретико-множественную разность, представляя ее в виде индексного объекта |
| intersection | Вычисляет теоретико-множественное пересечение |
| union | Вычисляет теоретико-множественное объединение |
| isin | Вычисляет булев массив, показывающий, содержится ли каждое значение индекса в переданной коллекции |
| delete | Вычисляет новый индексный объект, получающийся после удаления элемента с индексом i |
| drop | Вычисляет новый индексный объект, получающийся после удаления переданных значений |
| insert | Вычисляет новый индексный объект, получающийся после вставки элемента в позицию с индексом i |
| is\_rnonotonic | Возвращает True, если каждый элемент больше или равен предыдущему |
| is\_unique | Возвращает True, если в индексе нет повторяющихся значений |
| unique | Вычисляет массив уникальных значений в индексе |

**Базовая функциональность**

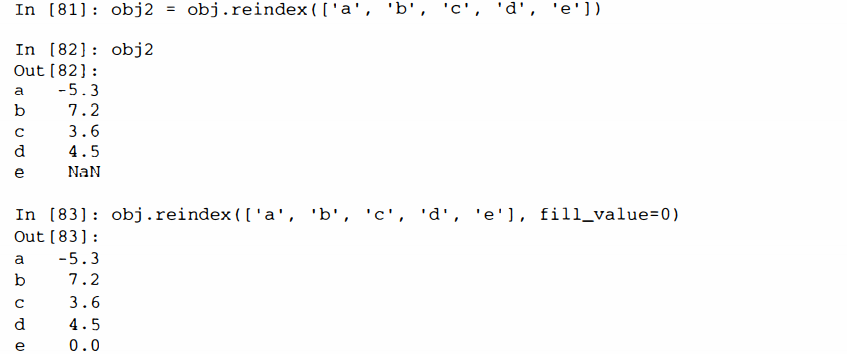
В этом разделе мы рассмотрим фундаментальные основы взаимодействия с данными, хранящимися в объектах Series и DataFraшe. В последующих главах мы более детально обсудим вопросы анализа и манипуляции данными с применением pandas. Эта книга не задумывалась как исчерпывающая документация по библиотеке pandas, я хотел лишь акцентировать внимание на наиболее важных чертах, оставив не столь употребительные (если не сказать эзотерические) вещи для самостоятельного изучения читателю.

**Переиндексация**

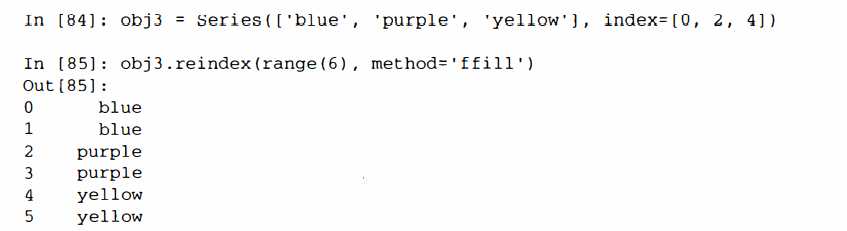
Для объектов pandas критически важен метод reindex, т. е. возможность создания нового объекта, данные в котором согласуются с новым индексом. Рассмотрим простой пример:



Если вызвать reindex для этого объекта Seгies, то данные будут реорганизованы в соответствии с новым индексом, а если каких-то из имеющихся в этом индексе значений раньше не было, то вместо них будут подставлены отсутствующие значения:



Для упорядоченных данных, например временных рядов, иногда желательно произвести интерполяцию, или восполнение отсутствующих значений в процессе переиндексации. Это позволяет сделать параметр method; так, если задать для него значение ffill, то будет произведено прямое восполнение значений:

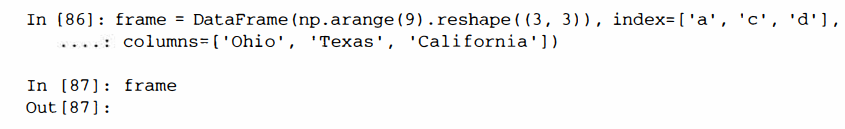


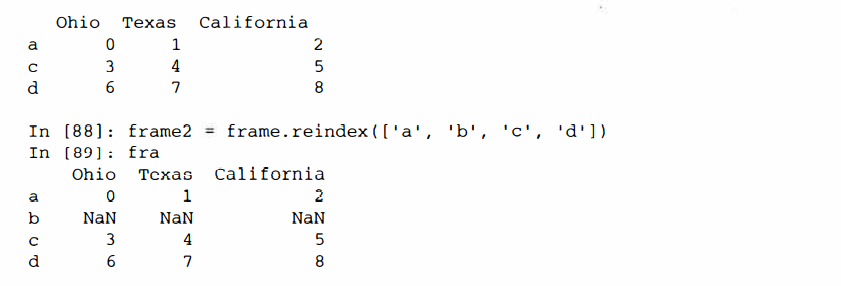
В табл. 5.4 перечислены возможные значения параметра method. В настоящее время интерполяцию, более сложную, чем прямое и обратное восполнение, нужно производить постфактум.

Таблица 5.4. Значение параметра method функции reindex (интерполяция)

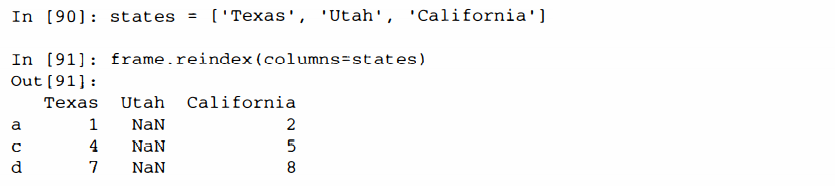
|  |  |
| --- | --- |
| **Значение** | **Описание** |
| ffill или pad | Восполнить (или перенести) значения в прямом направлении |
| Ьfill или backfill | Восполнить (или перенести) значения в обратном направлении |

В случае объекта DataFrame функция reindex может изменять строки, столбцы или то и другое. Если ей передать просто последовательность, то в результирующем объекте переиндексируются строки:

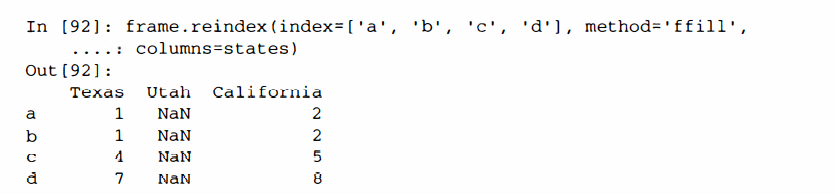




Столбцы можно переиндексировать, задав ключевое слово columns:



Строки и столбцы можно переиндексировать за одну операцию, хотя интерполяция будет применена только к строкам (к оси О):

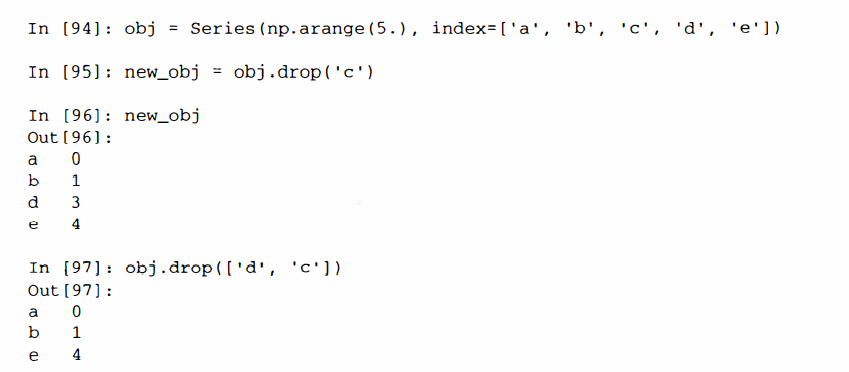


**Таблица 5.5**. Аргументы функции reindex

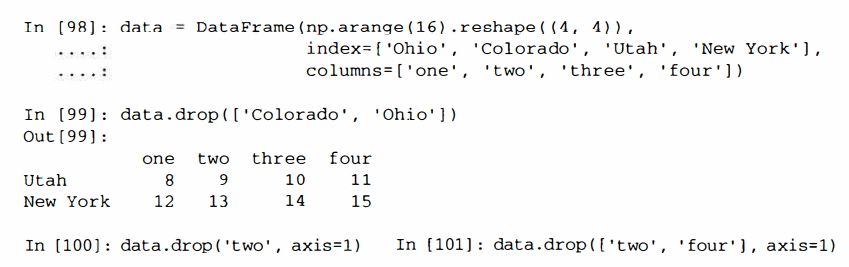
|  |  |
| --- | --- |
| **Аргумент** | **Описание** |
| index | Последовательность, которая должна стать новым индексом. Может быть экземпляром Index или любой другой структурой данных Python, похожей на последовательность. Экземпляр Index будет использован «как есть», без копирования |
| method | Метод интерполяции (восполнения), возможные значения приведены в табл. 5.4 |
| fill\_value | Значение, которой должно подставляться вместо отсутствующих значений, появляющихся в результате переиндексации |
| limit | При прямом или обратном восполнении максимальная длина восполняемой лакуны |
| level | Сопоставить с простым объектом Index на указанном уровне Multiindex, иначе выбрать подмножество |
| сору | Не копировать данные, если новый индекс эквивалентен старому. По умолчанию True (т. е. всегда копировать данные) |

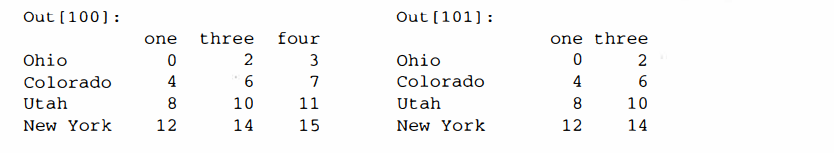
**Удаление элементов из оси**

Удалить один или несколько элементов из оси просто, если имеется индексный массив или список, не содержащий этих значений. Метод drop возвращает новый объект, в котором указанные значения удалены из оси:



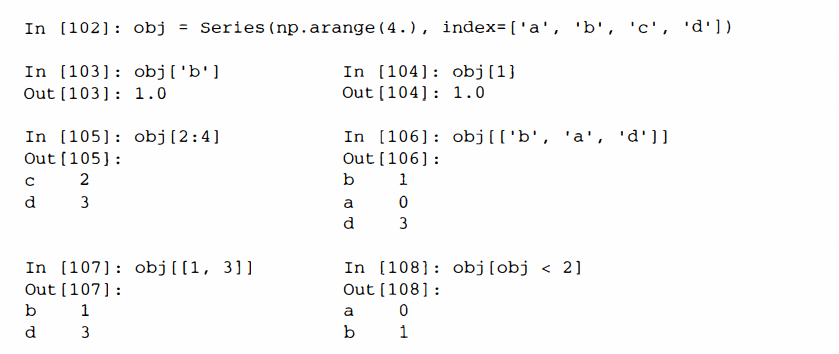
В случае DataFrame указанные в индексе значения можно удалить из любой оси:





**Доступ по индексу, выборка и фильтрация**

Доступ по индексу к объекту Series ( obj [ ... J) работает так же, как для массивов NшnPy с тем отличием, что индексами могут быть не только целые, но любые значения из индекса объекта Series. Вот несколько примеров:



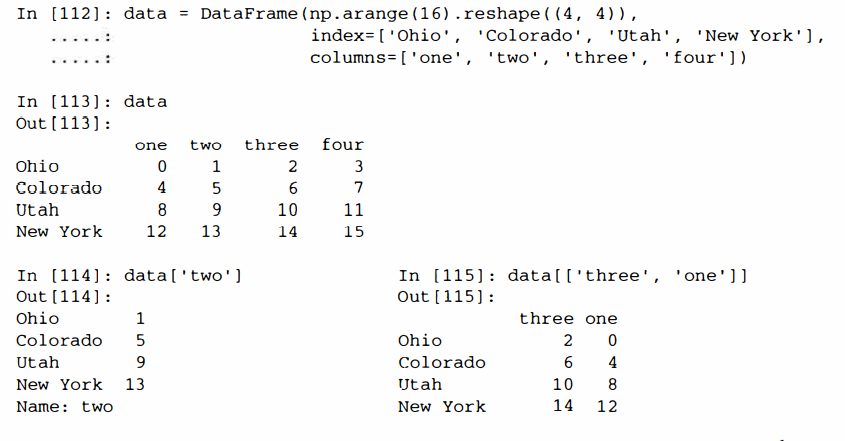
Вырезание с помощью меток отличается от обычного вырезания в Python тем, что конечная точка включается:



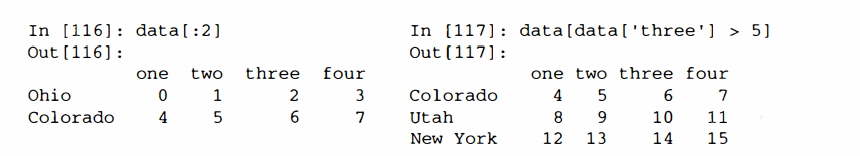
Установка с помощью этих методов работает ожидаемым образом:



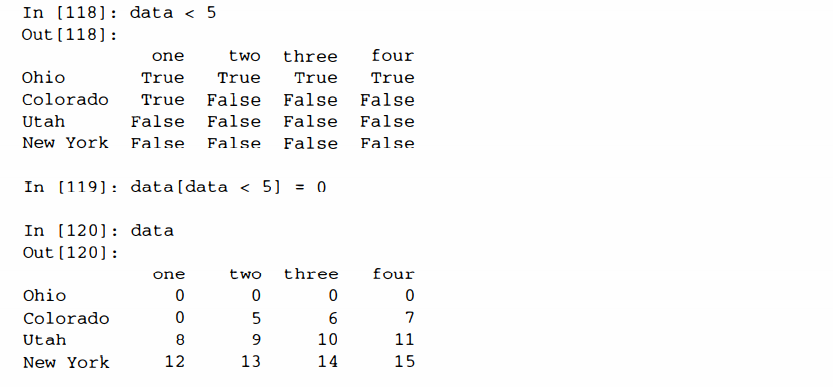
Как мы уже видели, доступ по индексу к DataFrame применяется для извлечения одного или нескольких столбцов путем задания единственного значения или последовательности:



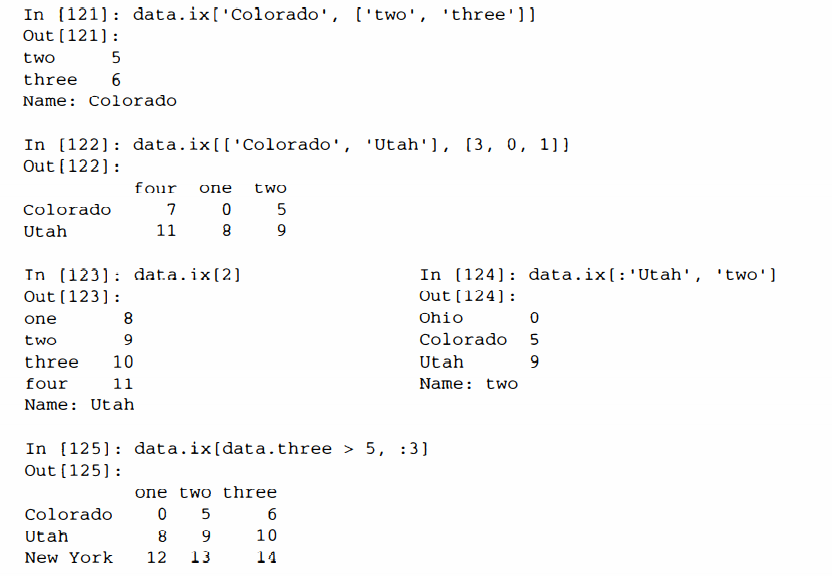
У доступа по индексу есть несколько частных случаев. Во-первых, выборка строк с помощью вырезания или булева массива:



Некоторым читателям такой синтаксис может показаться непоследовательным, по он был выбран исключительно из практических соображений. Еще одна возможность - доступ по индексу с помощью булева DataFramc, например порожденного в результате скалярного сравнения:



Идея в том, чтобы сделать DataFraшe синтаксически более похожим на ndarray в данном частном случае. Для доступа к строкам по индексу с помощью меток я ввел специальное индексное поле ix. Оно позволяет выбрать подмножество строк и столбцов DataFrame с применение нотации NumPy, дополненной метками осей. Как я уже говорил, это еще и более лаконичный способ выполнить переиндексацию:



Таким образом, существует много способов выборки и реорганизации данных, содержащихся в объекте pandas. Для DataFrame краткая сводка многих их них приведена в табл. 5.6. Позже мы увидим, что при работе с иерархическими индексами есть ряд дополнительных возможностей

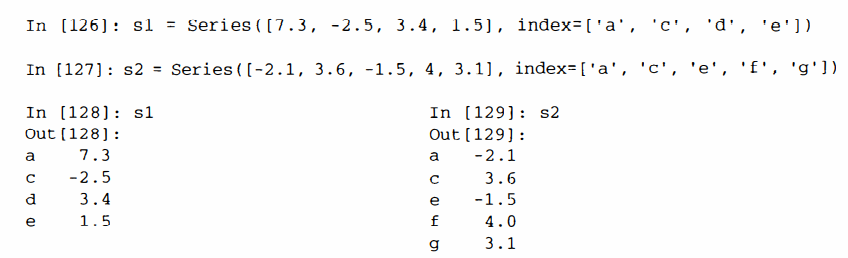
**Таблица 5.6**. Варианты доступа по индексу для объекта DataFrame

|  |  |
| --- | --- |
| **Вариант** | **Примечание** |
| obj [val] | Выбрать один столбец или последовательность столбцов из DataFrame. Частные случаи: булев массив (фильтрация строк), срез (вырезание строк) или булев DataFrame (установка значений в позициях, удовлетворяющих некоторому критерию) |
| obj.ix (val] | Выбрать одну строку или подмножество строк из DataFrame |
| obj.ix[:, val] | Выбрать один столбец или подмножество столбцов |
| obj.ix[vall, val2] | Выбрать строки и столбцы |
| метод reindex | Привести одну или несколько осей в соответствие с новыми индексами |
| Метод хs | Выбрать одну строку или столбец по метке и вернуть объект Series |
| методы icol, irow | Выбрать одну строку или столбец соответственно по целочисленному номеру и вернуть объект Series |
| методы get\_value, set\_value | Выбрать одно значение по меткам строки и столбца |

Проектируя pandas, я подспудно ощущал, что нотация frarile [: , col] для выборки столбца слишком громоздкая (и провоцирующая ошибки), поскольку выборка столбца - одна из самых часто встречающихся операций. Поэтому я пошел на компромисс и перенес все более выразительные операции доступа с помощью индексов-меток в ix.

**Арифметические операции и выравнивание данных**

Одна из самых важных черт pandas - поведение арифметических операций для объектов с разными индексами. Если при сложении двух объектов обнаруживаются различные пары индексов, то результирующий индекс будет объединением индексов. Рассмотрим простой пример:

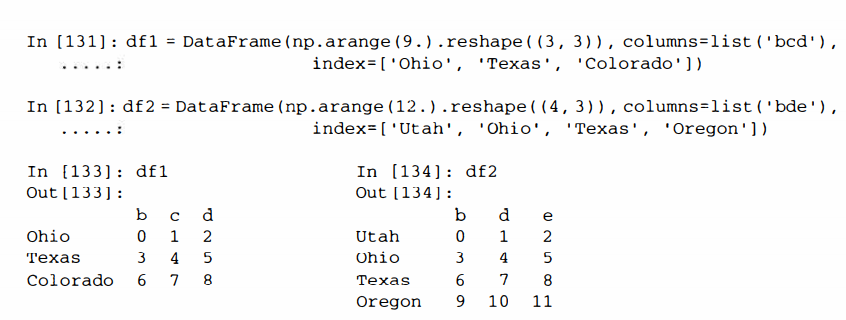


Сложение этих объектов дает:

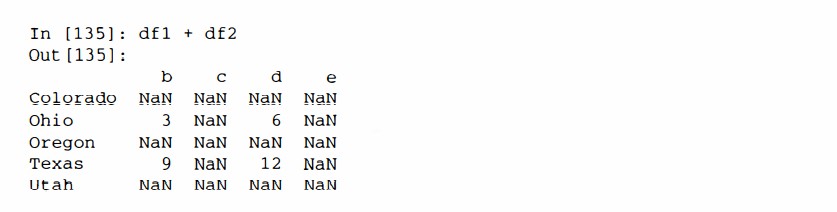


Вследствие внутреннего выравнивания данных образуются отсутствующие значения в позициях, для которых не нашлось соответственной пары. Отсутствующие значения распространяются на последующие операции.

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

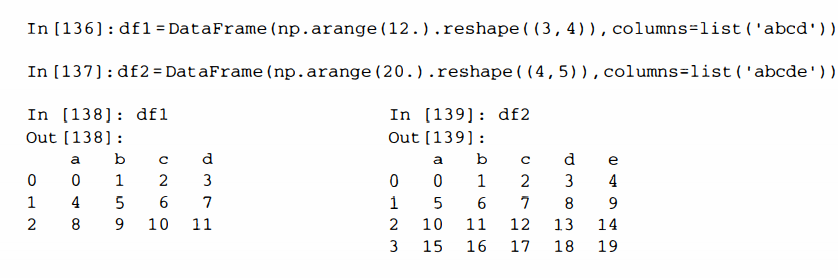


При сложении этих объектов получается DataFrame, индекс и столбцы которого являются объединениями индексов и столбцов слагаемых:



**Восполнение значений в арифметических методах**

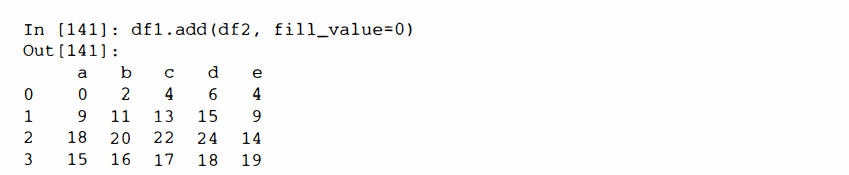
При выполнении арифметических операций с объектами, проиндексированными по-разному, иногда желательно поместить специальное значение, например 0, в позиции операнда, которым в другом операнде соответствует отсутствующая позиция:



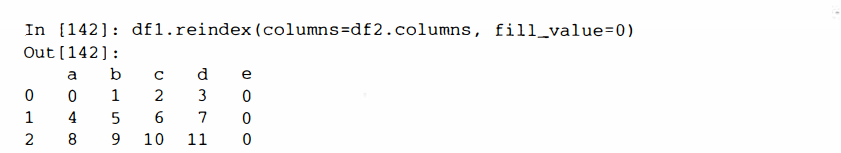
Сложение этих объектов порождает отсутствующие значения в позициях, которые имеются не в обоих операндах:



Теперь я вызову метод add объекта dfl и передам ему объект df2 и значение параметра fill\_value:



Точно так же, выполняя переиндексацию объекта Serics или DataFrame, можно указать восполняемое значение:

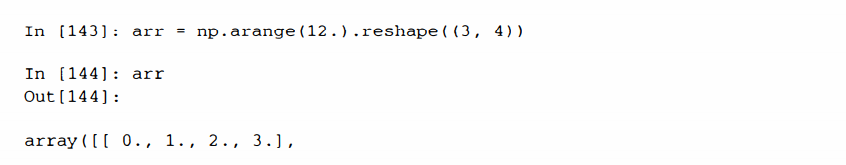


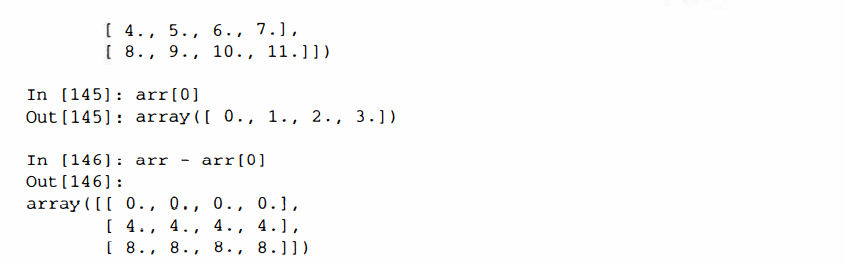
**Таблица 5.7.** Гибкие арифметические методы

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| add | Сложение (+) |
| sub | Вычитание (-) |
| div | Деление (/) |
| mul | Умножение (\*) |

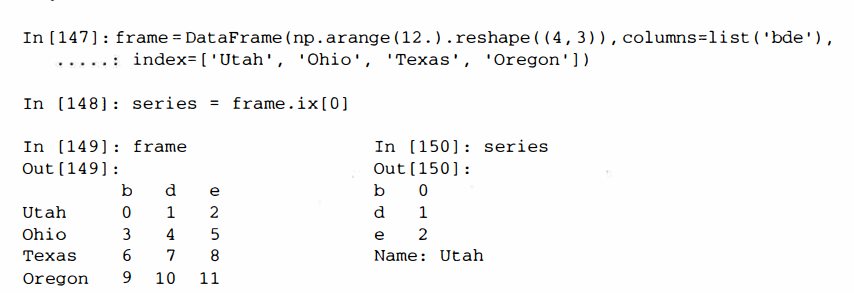
**Операции между DataFrame и Series**

Как и в случае массивов NumPy, арифметические операции между DataFrame и Series корректно определены. В качестве пояснительного примера рассмотрим вычисление разности между двумерным массивом и одной из его строк:





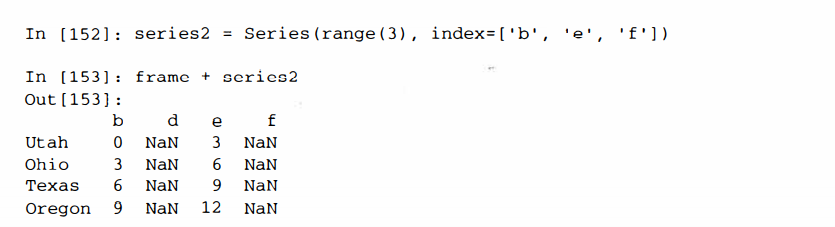
Это называется укладыванием и подробно объясняется в главе 12. Операции между DataFrarne и Series аналогичны:



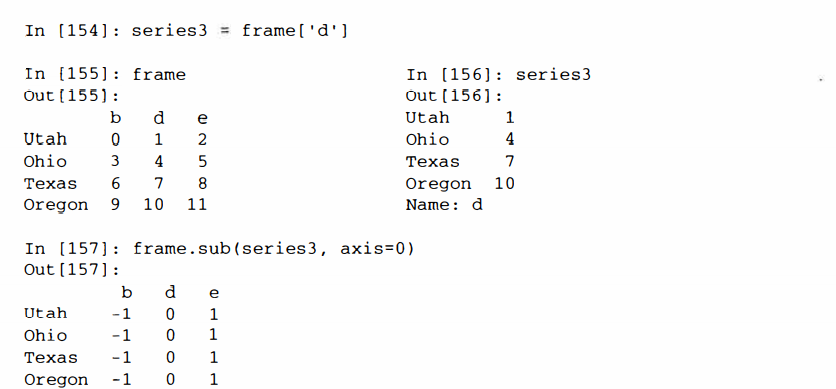
По умолчанию при выполнении арифметических операций между DataFrarne и Series индекс Series сопоставляется со столбцами DataFrarne, а укладываются строки:



Если какой-нибудь индекс не найден либо в столбцах DataFraшe, либо в индексе Series, то объекты переиндексируются для образования объединения:



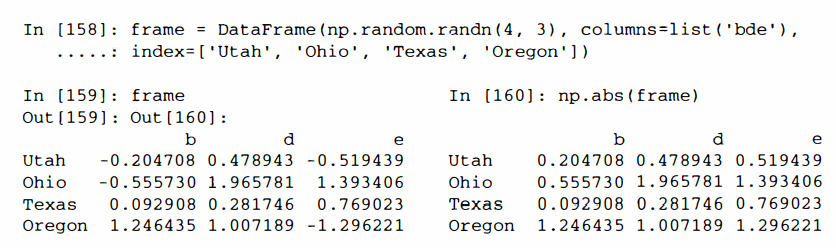
Если вы хотите вместо этого сопоставлять строки, а укладывать столбцы, то должны будете воспользоваться каким-нибудь арифметическим методом. Например:



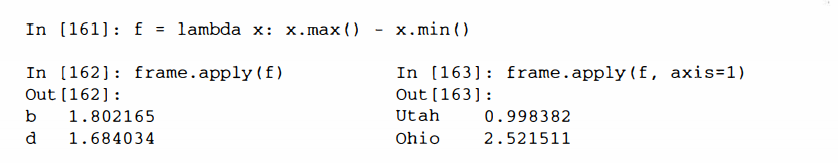
Передаваемый номер оси — это ось, вдоль которой производится сопоставление. В данном случае мы хотим сопоставлять с индексом строк DataFrame и укладывать поперек.

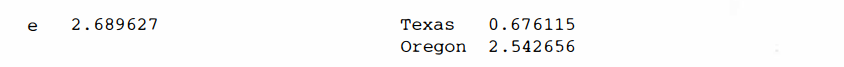
**Применение функций и отображение**

Универсальные функции NumPy (поэлементные методы массивов) отлично работают и с объектами pandas:

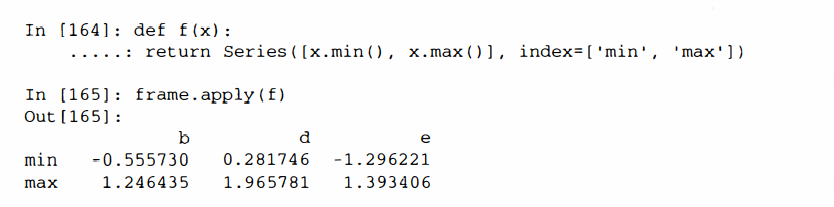


Еще одна часто встречающаяся операция - применение функции, определенной для одномерных массивов, к каждому столбцу или строке. Именно это и делает метод apply объекта DataFrame:

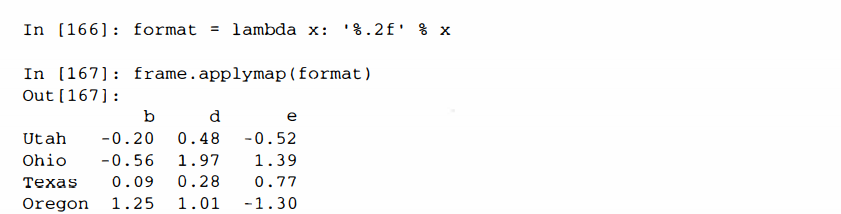




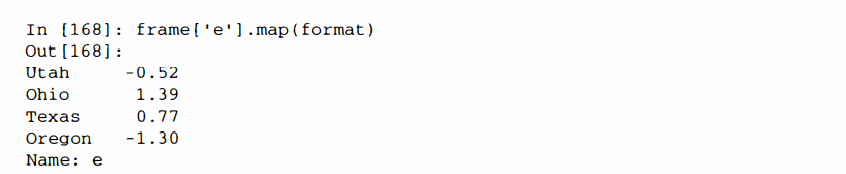
Многие из наиболее распространенных статистик массивов (например, surn и rnean) - методы DataFrame, поэтому применять apply в этом случае необязательно. Функция, передаваемая методу apply, не обязана возвращать скалярное значение, она может вернуть и объект Series, содержащий несколько значений:



Можно использовать и поэлементные функции Python. Допустим, требуется вычислить форматированную строку для каждого элемента frame с плавающей точкой. Это позволяет сделать метод applymap:

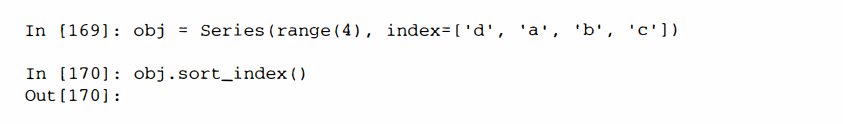


Этот метод называется applymap, потому что в классе Se1·ies есть метод map для применения функции к каждому элементу:



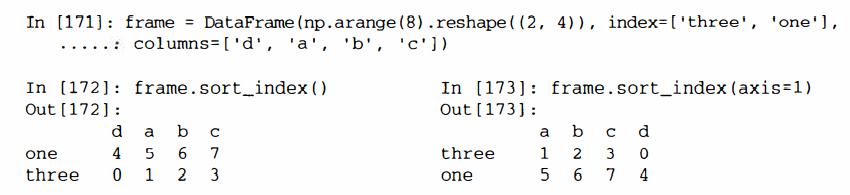
**Сортировка и ранжирование**

Сортировка набора данных по некоторому критерию - еще одна важная встроенная операция. Для лексикографической сортировки по индексу служит метод sort\_index, который возвращает новый отсортированный объект:

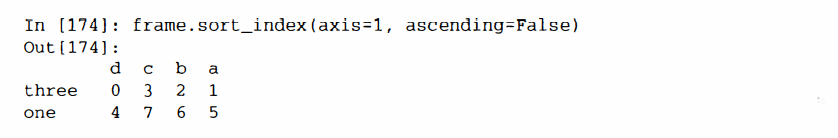




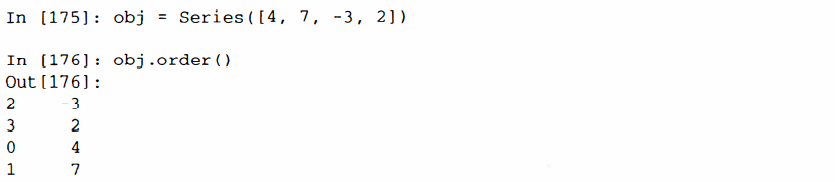
В случае DataFrame можно сортировать по индексу, ассоциированному с любой осью:



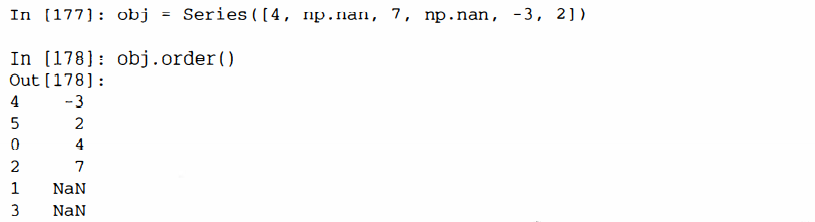
По умолчанию данные сортируются в порядке возрастания, но можно отсортировать их и в порядке убывания:



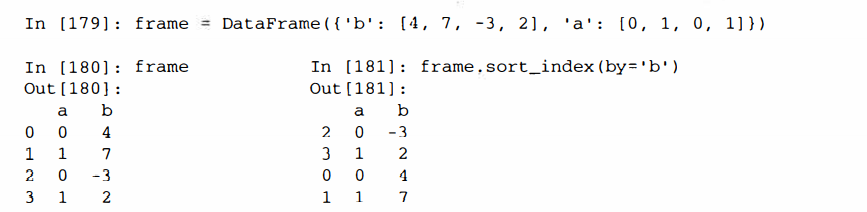
Для сортировки Series по значениям служит метод order:



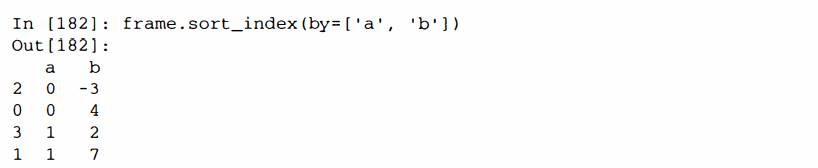
Отсутствующие значения по умолчанию оказываются в конце Series:



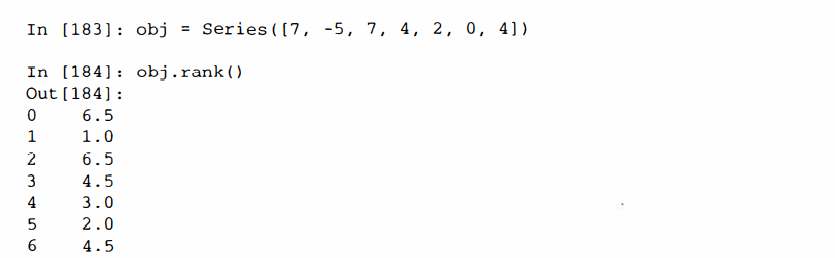
Объект Dataframe можно сортировать по значениям в одном или нескольких столбцах. Для этого имена столбцов следует передать в качестве значения параметра Bу:



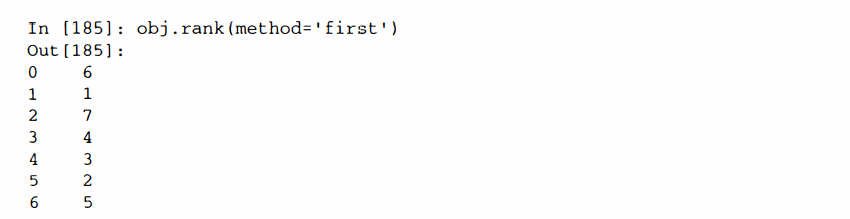
Для сортировки по нескольким столбцам следует передать список имен:



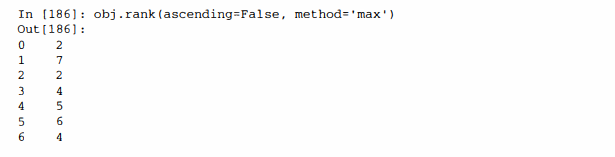
Ранжирование тесно связано с сортировкой, заключается оно в присваивании рангов - от единицы до числа присутствующих в массиве элементов. Это аналогично косвенным индексам, порождаемым методом numpy. argsort, с тем отличием, что существует правило обработки связанных рангов. Для ранжирования применяется метод rank объектов Series и DataFrame; по умолчанию rank обрабатывает связанные ранги, присваивая каждой группе средний ранг:



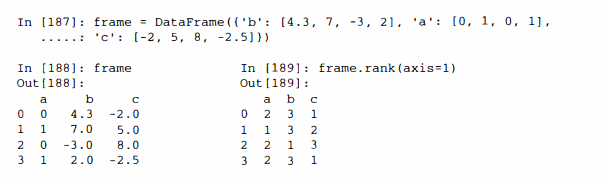
Ранги можно также присваивать u соответствии с порядком появления в данных:



Естественно, можно ранжировать и в порядке убывания:



В табл. 5.8 приведен перечень способов обработки связанных рангов. DataFrame умеет вычислять ранги как по строкам, так и по столбцам:

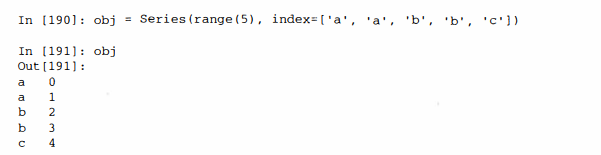


**Таблица 5.8.** Способы обработки связанных рангов

|  |  |
| --- | --- |
| Способ | Описание |
| 'average' | По умолчанию: одинаковым значениям присвоить средний ран |
| 'min' | По умолчанию: одинаковым значениям присвоить средний ран |
| 'max' | Всем элементам группы присвоить максимальный ранг |
| 'first' | Присваивать ранги в порядке появления значений в наборе данных |

**Индексы по осям с повторяющимися значениями**

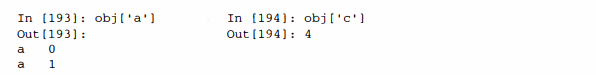
Во всех рассмотренных до сих пор примерах метки на осях (значения индекса) были уникальны. Хотя для многих функций pandas (например, reindex) требуется уникальность меток, в общем случае это необязательно. Рассмотрим небольшой объект Series с повторяющимися значениями в индексе:



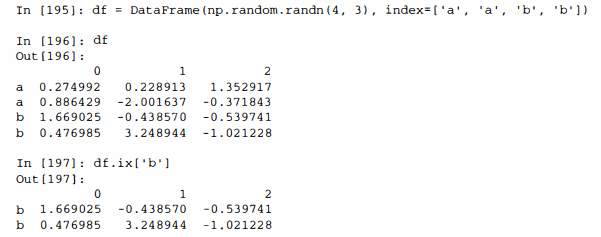
О том, являются значения уникальными или нет, можно узнать, опросив свойство is\_unique:



Выборка данных - одна из основных операций, поведение которых меняется в зависимости от наличия или отсутствия дубликатов. При доступе по индексу, встречающемуся несколько раз, возвращается объект Series; если же индекс одиночный, то скалярное значение:

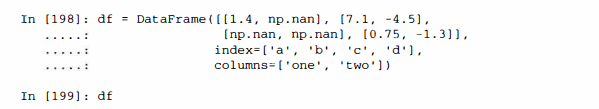


Такое же правило действует и для доступа к строкам в DataFrame:



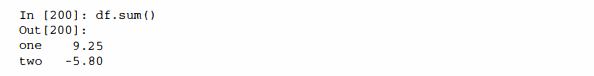
**Редукция и вычисление описательных статистик**

Объекты pandas оснащены набором стандартных математических и статистических методов. Большая их часть попадает в категорию редукций, или сводных статистик - методов, которые вычисляют единственное значение (например, сумму или среднее) для Series или объект Series - для строк либо столбцов DataFrame. По сравнению с эквивалентными методами массивов NumPy, все они игнорируют отсутствующие значения. Рассмотрим небольшой объект DataFrame:





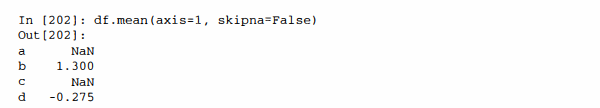
Метод sum объекта DataFrame возвращает Series, содержащий суммы по столбцам:



Если передать параметр axis= l, то суммирование будет производиться по строкам:



Отсутствующие значения исключаются, если только не является отсутствующим весь срез (в данном случае строка или столбец). Это можно подавить, задав параметр skipna:



Перечень часто употребляемых параметров методов редукции приведен в табл. 5.9.

**Таблица 5.9.** Параметры методов редукции

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| axis | Ось, по которой производится редуцирование. В случае DataFrame О означает строки, 1 - столбцы. |
| skipna | Исключать отсутствующие значения. По умолчанию True |
| level | Редуцировать с группировкой по уровням, если индекс по оси иерархический (Multilndex) |

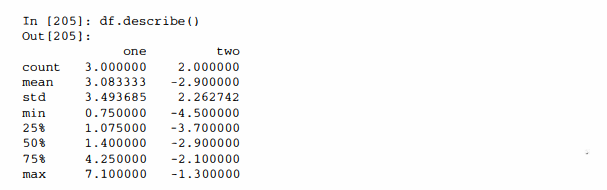
Некоторые методы, например idxmin и idxmax, возвращают косвенные статистики, скажем, индекс, при котором достигается минимум или максимум:



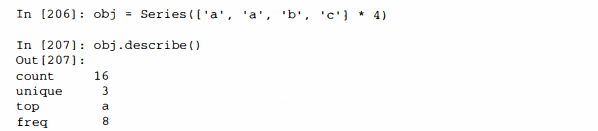
Есть также аккумулирующие методы:



Наконец, существуют методы, не относящиеся ни к редуцирующим, ни к аккумулирующим. Примером может служить метод describe, который возвращает несколько сводных статистик за одно обращение:



Б случае нечисловых данных describe возвращает другие сводные статистики:



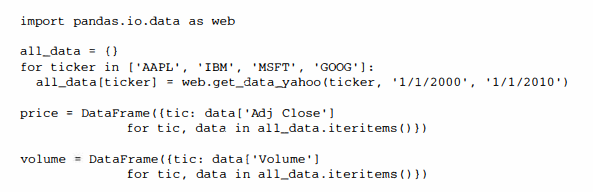
Полный список сводных статистик и родственных методов приведен в табл. 5.10.

**Таблица 5.10.** Описательные и сводные статистики

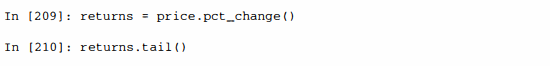
|  |  |
| --- | --- |
| **Метод** | **Описание** |
| count | Количество значений, исключая отсутствующие |
| describe | Вычисляет набор сводных статистик для Series или для каждого столбца DataFrame |
| min, max | Вычисляет минимальное или максимальное значение |
| argmin, argmax | Вычисляет позицию в индексе (целые числа), при котором достигается минимальное или максимальное значение соответственно |
| idxmin, idxmax | Вычисляет значение индекса, при котором достигается минимальное или максимальное значение соответственно |
| quantile | Вычисляет выборочный квантиль в диапазоне от 0 до 1 |
| sum | Сумма значений |
| mean | Среднее значение |
| median | Медиана (50%-ый квантиль) |
| mad | Среднее абсолютное отклонение от среднего |
| var | Выборочная дисперсия |
| std | Выборочное стандартное отклонение |
| skew | Асимметрия (третий момент) |
| kurt | Куртозис (четвертый момент) |
| cumsum | Нарастающая сумма |
| cummin, cummax | Нарастающий минимум или максимум соответственно |
| cummin, cummax | Нарастающее произведение |
| diff | Первая арифметическая разность (полезно для временных рядов) |
| pct\_change | Первая арифметическая разность (полезно для временных рядов) |

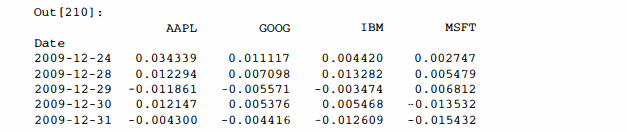
**Корреляция и ковариация**

Некоторые сводные статистики, например корреляция и ковариация, вычисляются по парам аргументов. Рассмотрим объекты DataFrame, содержащие цены акций и объемы биржевых сделок, взятые с сайта Уаhoo! Finance:

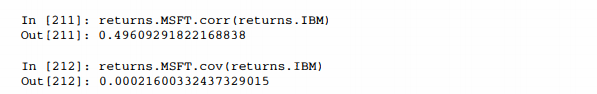


Теперь вычислим процентные изменения цен:

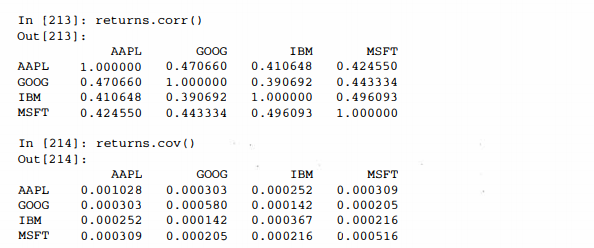




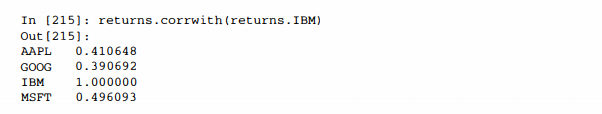
Метод corr объекта Series вычисляет корреляцию перекрывающихся, отличных от NA, выровненных по индексу значений в двух объектах Series. Соответственно, метод cov вычисляет ковариацию:



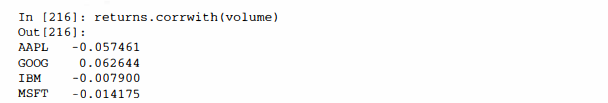
С другой стороны, методы corr и cov объекта DataFrame возвращают соответственно полную корреляционную или ковариационную матрицу в виде DataFame:



С помощью метода corrwith объекта DataFrame можно вычислить попарные корреляции между столбцами или строками DataFrame и другим объектом Series или DataFrame. Если передать ему объект Series, то будет возвращен Series, содержащий значения корреляции, вычисленные для каждого столбца:



Если передать объект DataFrame, то будут вычислены корреляции столбцов с соответственными именами. Ниже я вычисляю корреляции процентных изменений с объемом сделок:



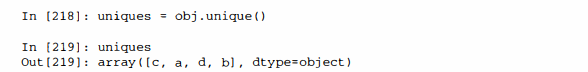
Если передать axis= 1, то будут вычислены корреляции строк. Во всех случаях перед началом вычислений данные выравниваются по меткам.

**Уникальные значения, счетчики значений и членство**

Еще один класс методов служит для извлечения информации о значениях, хранящихся в одномерном объекте Series. Для иллюстрации рассмотрим пример



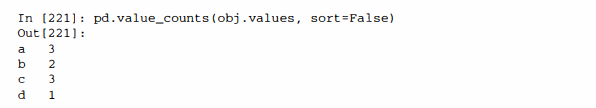
Метод unique возвращает массив уникальных значений в Series:



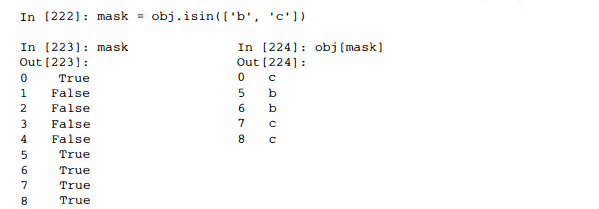
Уникальные значения необязательно возвращаются в отсортированном порядке, но могут быть отсортированы впоследствии, если это необходимо (uniques.sort () ). Метод value\_counts вычисляет объект Series, содержащий частоты встречаемости значений:



Для удобства этот объект отсортирован по значениям в порядке убывания. Функция value\_counts может быть также вызвана как метод pandas верхнего уровня и в таком случае применима к любому массиву или последовательности:



Наконец, метод isin вычисляет булев вектор членства в множестве и может быть полезен для фильтрации набора данных относительно подмножества значений в объекте Series или столбце DataFrame

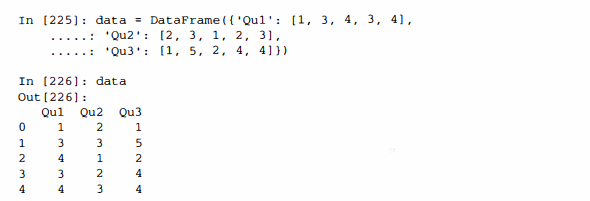


Справочная информация по этим методам приведена в табл. 5.11.

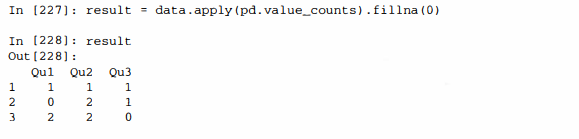
**Таблица 5. 11.** Уникальные значения, счетчики значений и методы «раскладывания»

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| isin | Вычисляет булев массив, показывающий, содержится ли каждое принадлежащее Series значение в переданной последовательности |
| unique | Вычисляет массив уникальных значений в Series и возвращает их в порядке появления |
| value\_counts | Возвращает объект Series, который содержит уникальное значение в качестве индекса и его частоту в качестве соответствующего значения. Отсортирован в порядке убывания частот |

Иногда требуется вычислить гистограмму нескольких взаимосвязанных столбцов в DataFrame. Приведем пример:



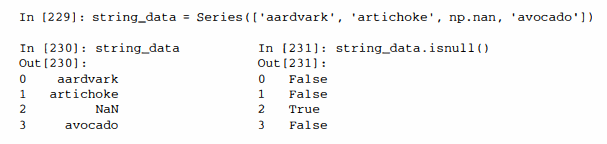
Передача pandas.value\_counts методу apply этого объекта DataFrame дает:

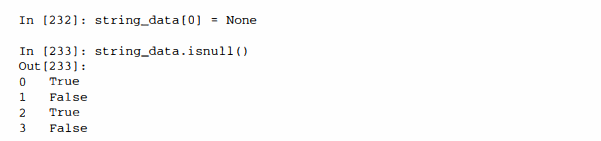
**Обработка отсутствующих данных**

Отсутствующие данные - типичное явление в большинстве аналитических приложений. При проектировании pandas в качестве одной из целей ставилась задача сделать работу с отсутствующими данными как можно менее болезненной. Например, выше мы видели, что при вычислении всех описательных статистик для объектов pandas отсутствующие данные не учитываются.

В pandas для представления отсутствующих данных в любых массивах - как чисел с плавающей точкой, так и иных - используется значение с плавающей точкой NaN (не число). Это просто признак, который легко распознать:



Встроенное в Python значение None также рассматривается как отсутствующее в массивах объектов:



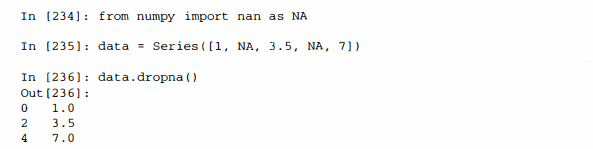
Я не утверждаю, что представление отсутствующих значений в pandas оптимально, но оно простое и в разумной степени последовательное. Принимая во внимание характеристики производительности и простой API, это наилучшее решение, которое я смог придумать в отсутствие истинного типа данных NA или выделенной комбинации бит среди типов данных NumPy. Поскольку разработка NumPy продолжается, в будущем ситуация может измениться.

**Таблица 5.12.** Методы обработки отсутствующих данных

|  |  |
| --- | --- |
| **Аргумент** | **Описание** |
| dropna | Фильтрует метки оси в зависимости от того, существуют ли для метки отсутствующие данные, причем есть возможность указать различные пороги, определяющие, какое количество отсутствующих данных считать допустимым |
| fillna | Восполняет отсутствующие данные указанным значением или использует какой-нибудь метод интерполяции, например ‘ffill’ или ‘bfill’ |
| isnull | Возвращает объект, содержащий булевы значения, которые показывают, какие значения отсутствуют |
| notnull | Логическое отрицание isnull |

**Фильтрация отсутствующих данных**

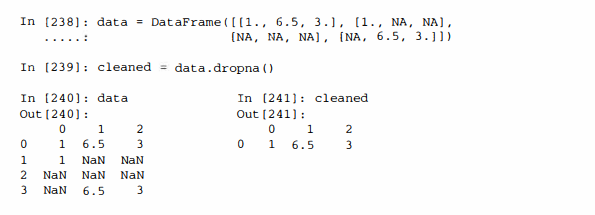
Существует ряд средств для фильтрации отсутствующих данных. Конечно, можно сделать это и вручную, но часто бывает полезен метод dropna. Для Series он возвращает другой объект Se1·ies, содержащий только данные и значения индекса, отличные от NA:



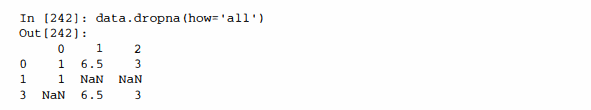
Естественно, это можно было бы вычислить и самостоятельно с помощью булевой индексации:



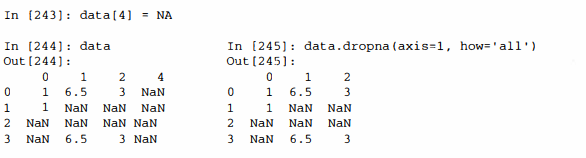
В случае объектов DataFrame все немного сложнее. Можно отбрасывать строки или столбцы, если они содержат только NА-значения или хотя бы одно NАзначение. По умолчанию метод dropna отбрасывает все строки, содержащие хотя бы одно отсутствующее значение:



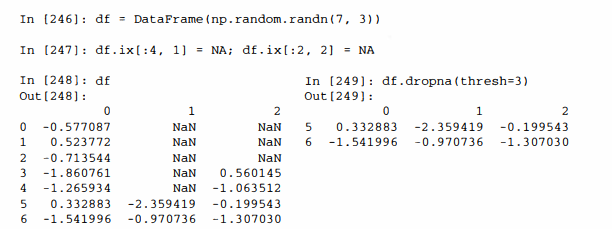
Если передать параметр how= ‘all’ ·, то будут отброшены строки, которые целиком состоят из отсутствующих значений:



Для отбрасывания столбцов достаточно передать параметр axis = 1:



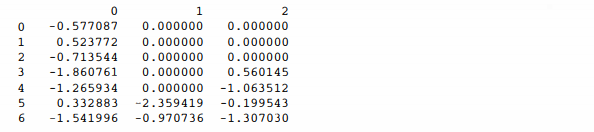
Родственный способ фильтрации строк DataFrame в основном применяется к временным рядам. Допустим, требуется оставить только строки, содержащие определенное количество наблюдений. Этот порог можно задать с помощью аргумента thresh:



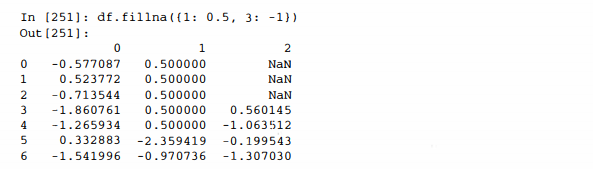
**Восполнение отсутствующих данных**

Иногда отсутствующие данные желательно не отфильтровывать (и потенциально вместе с ними отбрасывать полезные данные), а каким-то способом заполнить «дыры». В большинстве случаев для этой цели можно использовать метод fillna. Ему передается константа, подставляемая вместо отсутствующих значений:

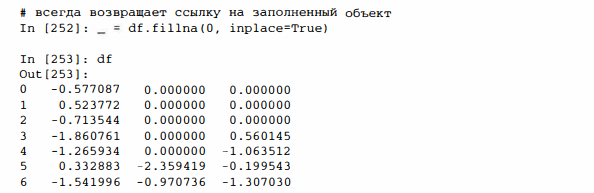




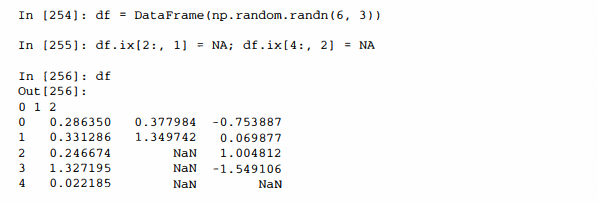
Если передать методу fillna словарь, то можно будет подставлять вместо отсутствующих данных значение, зависящее от столбца:

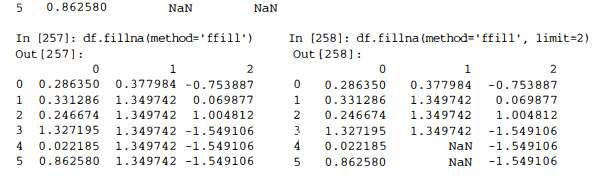


Метод fillna возвращает новый объект, но можно также модифицировать существующий объект на месте:

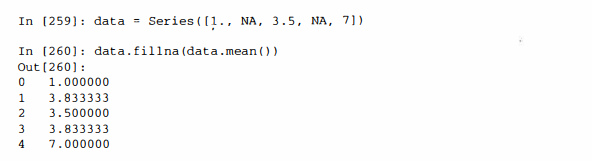


Те же методы интерполяции, что применяются для переиндексации, годятся и для fillna:





При некоторой изобретательности можно использовать fillna и другими способами. Например, можно передать среднее или медиану объекта Series:



Справочная информация о методе fillna приведена в табл. 5.13.

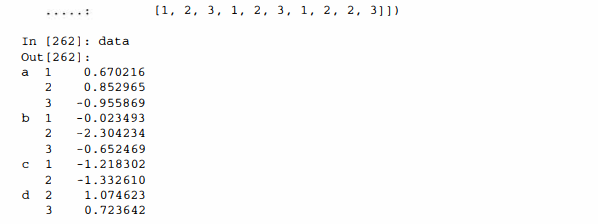
**Таблица 5.13.** Аргументы метода fillna

|  |  |
| --- | --- |
| **Аргумент** | **Описание** |
| value | Скалярное значение или похожий на словарь объект для восполнения отсутствующих значений |
| method | Метод интерполяции. По умолчанию, если не задано других аргументов, предполагается метод ‘ffill’ |
| axis | Ось, по которой производится восполнение. По умолчанию axis= 0 |
| inplace | Модифицировать исходный объект, не создавая копию |
| limit | Для прямого и обратного восполнения максимальное количество непрерывных заполняемых промежутков |

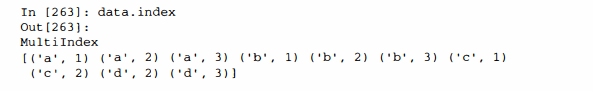
**Иерархическое индексирование**

Иерархическое индексирование - важная особенность pandas, позволяющая организовать несколько (два и более) уровней индексирования по одной оси. Говоря абстрактно, это способ работать с многомерными данными, представив их в форме с меньшей размерностью. Начнем с простого примера - создадим объект Series с индексом в виде списка списков или массивов:

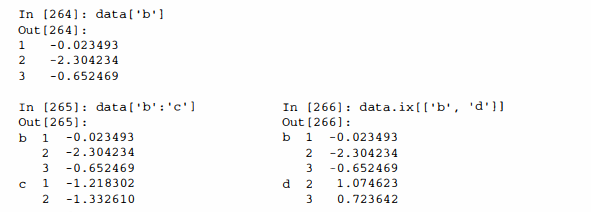




Здесь мы видим отформатированное представление Series с мультииндексом (мultiIndex). «Разрывы» в представлении индекса означают «взять значение вышестоящей метки».



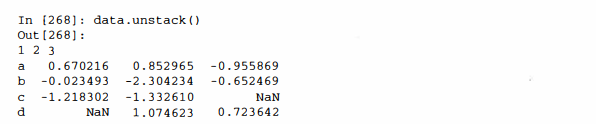
Для иерархически индексированного объекта возможен доступ по так называемому частичному индексу, что позволяет лаконично записывать выборку подмножества данных:



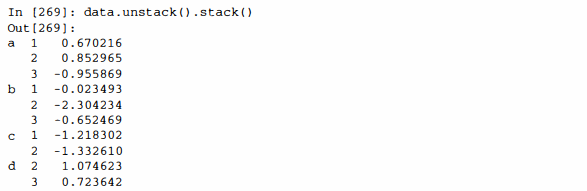
В некоторых случаях возможна даже выборка с «внутреннего» уровня:



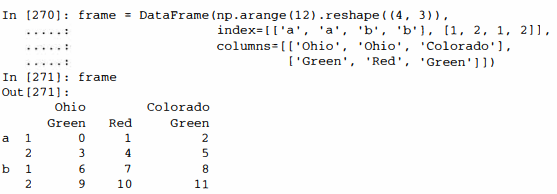
Иерархическое индексирование играет важнейшую роль в изменении формы данных и групповых операциях, в том числе построении сводных таблиц. Например, эти данные можно было бы преобразовать в DataFrame с помощью метода unstack:



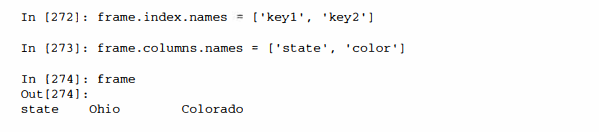
Обратной к unstack операцией является stack:

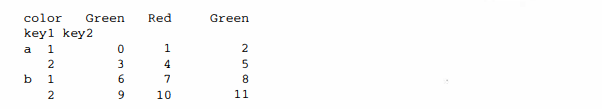


Методы stack и unstack будут подробно рассмотрены в главе 7. В случае DataFrame иерархический индекс можно построить по любой оси:



Уровни иерархии могут иметь имена (как строки или любые объекты Python). В таком случае они будут показаны при выводе на консоль (не путайте имена индексов с метками на осях!):





Доступ по частичному индексу, как и раньше, позволяет выбирать группы столбцов:

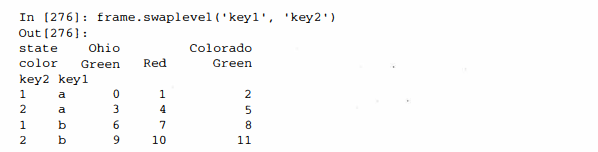


Мультииндекс можно создать отдельно, а затем использовать повторно; в показанном выше объекте DataFrame столбцы с именами уровней можно было бы создать так:

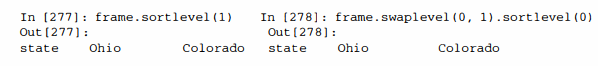


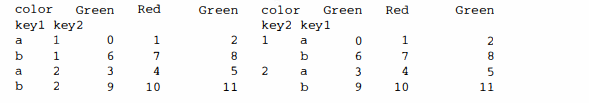
**Уровни переупорядочения и сортировки**

Иногда требуется изменить порядок уровней на оси или отсортировать данные по значениям на одной уровне. Метод swaplevel принимает номера или имена двух уровней и возвращает новый объект, в котором эти уровни переставлены (но во всех остальных отношениях данные не изменяются):



С другой стороны, метод sortlevel выполняет устойчивую сортировку данных, используя только значения на одном уровне. После перестановки уровней обычно вызывают также sortlevel, чтобы лексикографически отсортировать результат:

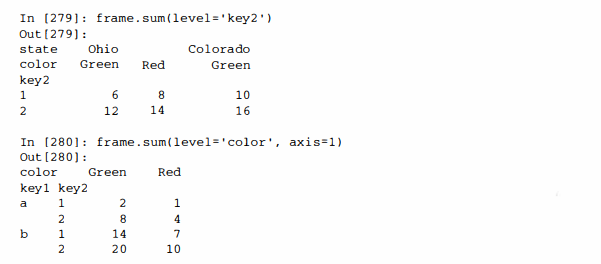




Производительность выборки данных из иерархически индексированных объектов будет гораздо выше, если индекс отсортирован лексикографически, начиная с самого внешнего уровня, т. е. в результате вызова sortleve(0) или sort\_index () .

**Сводная статистика по уровню**

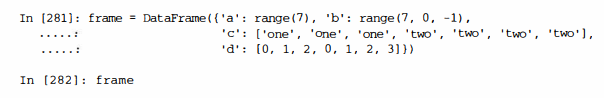
У многих методов объектов DataFrame и Series, вычисляющих сводные и описательные статистики, имеется параметр level для задания уровня, на котором требуется производить суммирование по конкретной оси. Рассмотрим тот же объект DataFrame, что и выше; мы можем суммировать по уровню для строк или для столбцов:

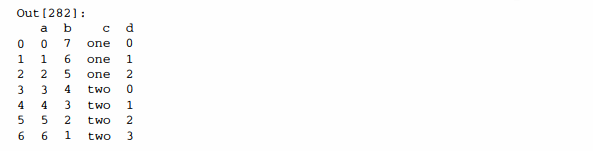


Реализовано это с помощью имеющегося в pandas механизма groupby, который мы подробно рассмотрим позже.

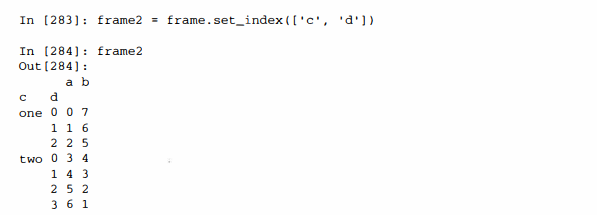
**Работа со столбцами DataFrame**

Не так уж редко возникает необходимость использовать один или несколько столбцов DataFrame в качестве индекса строк; альтернативно можно переместить индекс строк в столбцы DataFrame. Рассмотрим пример:

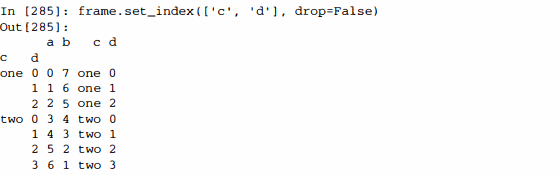




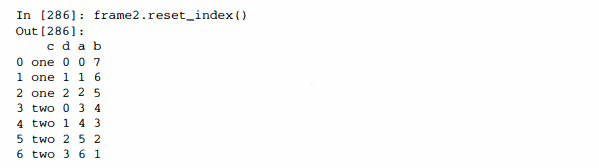
Метод set\_index объекта DataFrame создает новый DataFrame, используя в качестве индекса один или несколько столбцов:



По умолчанию столбцы удаляются из DataFrame, хотя их можно и оставить:



Есть также метод reset\_index, который делает прямо противоположное set\_ index; уровни иерархического индекса перемещаются в столбцы:



**Другие возможности pandas**

Ниже перечислено еще несколько возможностей, которые могут пригодиться вам в экспериментах с данными.

**Доступ по целочисленному индексу**

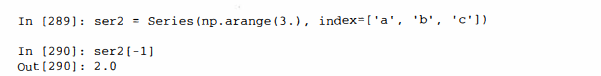
При работе с объектами pandas, проиндексированными целыми числами, начинающие часто попадают в ловушку из-за некоторых различий с семантикой доступа по индексу к встроенным в Python структурам данных, в частности, спискам и кортежам. Например, вряд ли вы ожидаете ошибки в таком коде:



данном случае pandas могла бы прибегнуть к целочисленному индексированию, но я не знаю никакого общего и безопасного способа сделать это, не внося тонких ошибок. Здесь мы имеем индекс, содержащий значения 0, 1, 2, но автоматически попять, чего хочет пользователь (индекс по метке или по позиции) трудно:



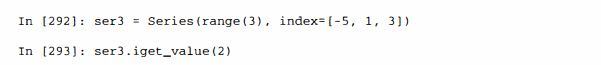
С другой стороны, когда индекс не является целым числом, неоднозначности не возникает:

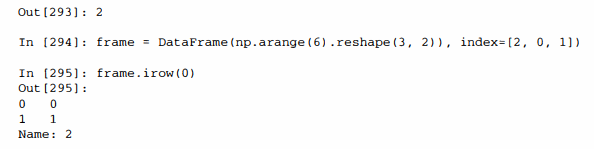


Чтобы не оставлять места разноречивым интерпретациям, принято решение: если индекс по оси содержит индексаторы, то доступ к данным по целочисленному индексу всегда трактуется как доступ по метке. Это относится и к полю ix:



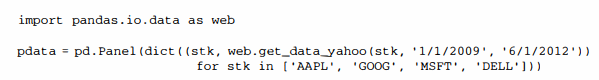
В случае, когда требуется надежный доступ по номеру позиции вне зависимости от типа индекса, можно использовать метод iget\_value объекта Series или методы irow и icol объекта DataFrame:



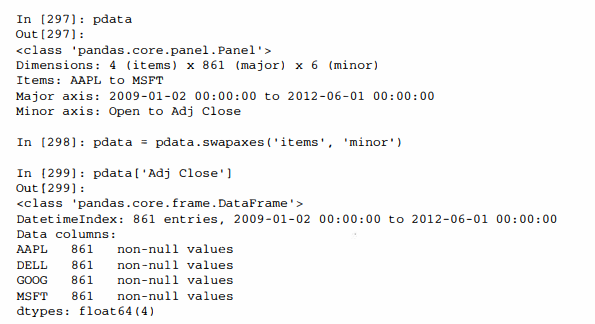


**Структура данных Рапеl**

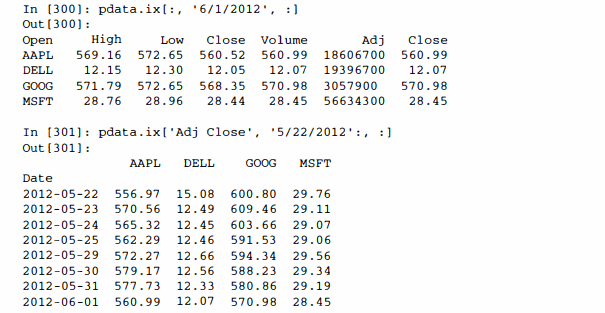
Хотя структура данных Panel и не является основной темой этой книги, она существует в pandas и может рассматриваться как трехмерный аналог DataFraшe. При разработке pandas основное внимание уделялось манипуляциям с табличными данными, поскольку о них проще рассуждать, а наличие иерархических индексов в большинстве случаев позволяет обойтись без настоящих N-мерных массивов. Для создания Panel используется словарь объектов DataFrame или трехмерный массив ndarray:



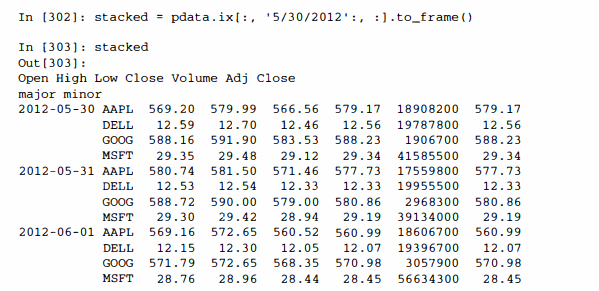
Каждый элемент Panel (аналог столбца в DataFraшe) является объектом DataFrame:



Основанное на поле ix индексирование по меткам обобщается на три измерения, поэтому мы можем выбрать все данные за конкретную дату или диапазон дат следующим образом:



Альтернативный способ представления панельных данных, особенно удобный для подгонки статистических моделей, - в виде «стопки» объектов DataFrame:



У объекта DataFrame есть метод to\_panel - обращение to\_frame:

