**Глава 6. Чтение и запись данных, форматы файлов**

Описанные в этой книге инструменты были бы бесполезны, если бы в программе на Python не было возможности легко импортировать и экспортировать данные Я буду рассматривать в основном ввод и вывод с помощью объектов pandas, хотя, разумеется, в других библиотеках нет недостатка в соответствующих средствах. Например, в NumPy имеются низкоуровневые, но очень быстрые функции для загрузки и сохранения данных, включая 11 поддержку файлов, спроецированных на память. Подробнее об этом см. главу 12.

Обычно средства ввода-вывода относят к нескольким категориям: чтение файлов в текстовом или каком-то более эффективном двоичном формате, загрузка из баз данных и взаимодействие с сетевым источником, например API доступа к веб.

**Чтение и запись данных в текстовом формате**

Python превратился n излюбленный язык манипулирования текстом и файлами благодаря простому синтаксису взаимодействия с файлами, интуитивно понятным структурам данных и таким удобным средствам, как упаковка и распаковка кортежей.

В библиотеке pandas имеется ряд функций для чтения табличных данных, представленных в виде объекта Dataframe. Все они перечислены в табл. 6.1, хотя чаще всего вы будете иметь дело с функциями read\_csv и read\_table.

**Таблица 6.1.** Функции чтения в pandas

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| read\_csv | Загружает данные с разделителями из файла, URL-aдpeca или похожего на файл объекта. По умолчанию разделителем является запятая |
| read\_csv | Загружает данные с разделителями из файла, URL-aдpeca или похожего на файл объекта. По умолчанию разделителем является символ табуляции ('\t') |
| read\_fwf | Читает данные в формате с фиксированной шириной столбцов (без разделителей) |
| read\_clipboard | Вариант read\_taЬle, который читает данные из буфера обмена. Полезно для преобразования в таблицу данных на веб-странице |

Я дам краткий обзор этих функций, которые служат для преобразования текстовых данных в объект DataFrame. Их параметры можно отнести к нескольким категориям:

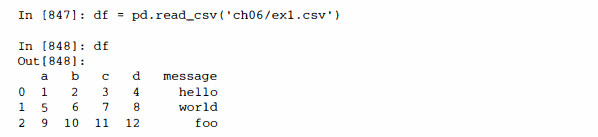
индексирование: какие столбцы рассматривать как индекс возвращаемого DataFrame и откуда брать имена столбцов: из файла, от пользователя или вообще ниоткуда;

* выведение типа и преобразование данных. включает определенные пользователем преобразования значений и список маркеров отсутствующих данных;
* разбор даты и времени: включает средства комбинирования, в том числе сбор данных о дате и времени из нескольких исходных столбцов в один результирующий;
* итерирование: поддержка обхода очень больших файлов;
* проблемы «грязных» данных пропуск заголовка или концевика, комментариев и другие мелочи, например обработка числовых данных, в которых тройки разрядов разделены запятыми.

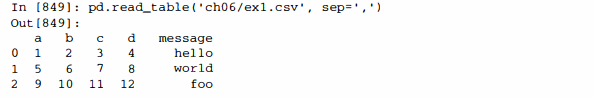
Выведение типа - одна из самых важных черт этих функций; это означает, что пользователю необязательно явно задавать, содержат столбцы данные с плавающей точкой, целочисленные, булевы или строковые. Правда, для обработки дат и нестандартных типов требуется больше усилий. Начнем с текстового файла, содержащего короткий список данных через запятую (формат CSV):



Поскольку данные разделены запятыми, мы можем прочитать их в DataFrame с помощью функции read\_csv:

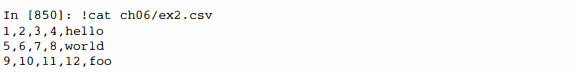


Можно было бы также воспользоваться функцией read\_taЫe, указав разделитель:

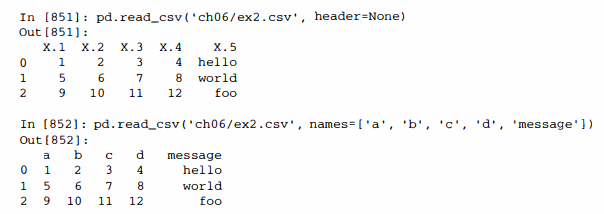


Я здесь пользуюсь командой Unix cat, которая печатает содержимое файла на экране без какого-либо форматирования. Если вы работаете в Windows, можете с тем же успехом использовать команду type.

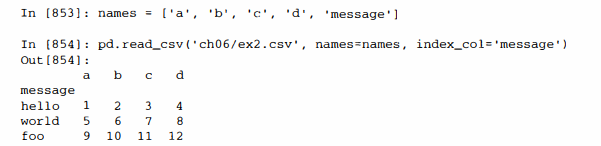
В файле не всегда есть строка-заголовок. Рассмотрим такой файл:



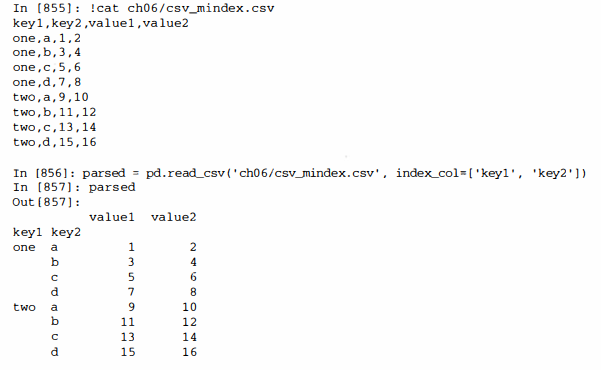
Прочитать его можно двумя способами. Можно поручить pandas выбор имен столбцов по умолчанию, а можно задать их самостоятельно:



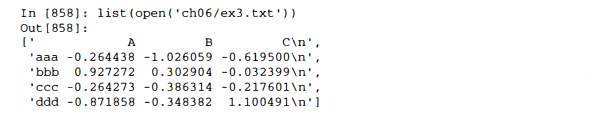
Допустим, мы хотим, чтобы столбец message стал индексом возвращаемого объекта DataFrame. Этого можно добиться, задав аргумент index\_col, в котором указать, что индексом будет столбец с номером 4 или с именем ‘message’:



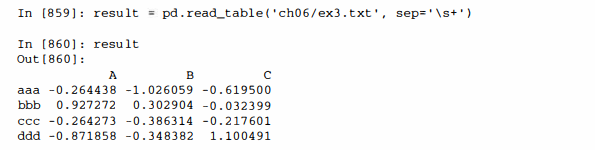
Если вы хотите сформировать иерархический индекс из нескольких столбцов, то просто передайте список их номеров или имен:



Иногда в таблице нет фиксированного разделителя, а для разделения полей используются пробелы или еще какой-то символ. В таком случае можно передать функции read\_table регулярное выражение вместо разделителя. Рассмотрим такой текстовый файл:

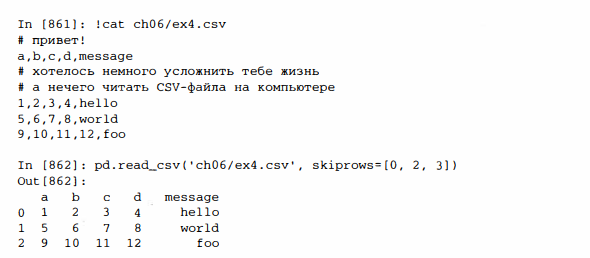


В данном случае поля разделены переменным числом пробелов и, хотя можно было бы переформатировать данные вручную, проще передать регулярное выражение \s+:

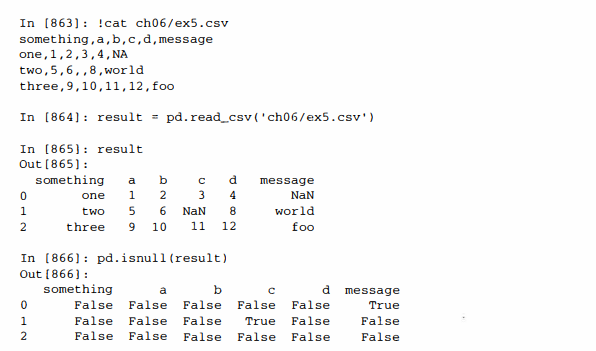


Поскольку имен столбцов на одно меньше, чем число строк, read\_ table делает вывод, что в данном частном случае первый столбец должен быть индексом DataFrame.

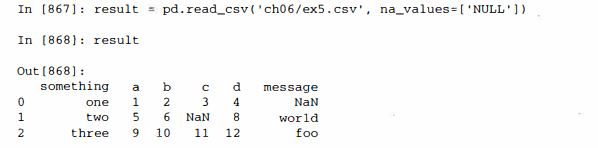
У функций разбора много дополнительных аргументов, которые помогают справиться с широким разнообразием файловых форматов (см. табл. 6.2). Например, параметр skiprows позволяет пропустить первую, третью и четвертую строку файла:



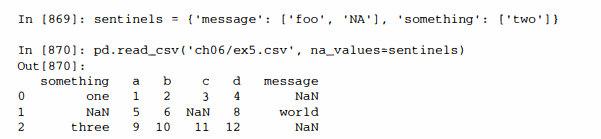
Обработка отсутствующих значений - важная и зачастую сопровождаемая тонкими нюансами часть разбора файла. Отсутствующие значения обычно либо вообще опущены (пустые строки), либо представлены специальными маркера­aaми. По умолчанию в pandas используется набор общеупотребительных маркеров: NA, -1. #IND И NULL:



Параметр na\_values может принимать список или множество строк, рассматриваемых как маркеры отсутствующих значений:



Если n разных столбцах применяются разные маркеры, то их можно задать с помощью словаря:

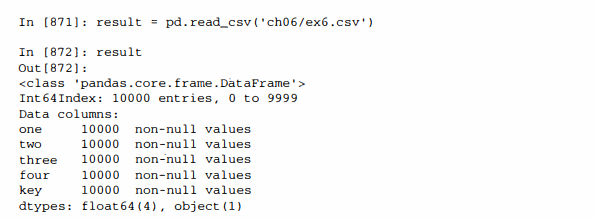


**Таблица 6.2.** Аргументы функций read\_csv и read\_table

|  |  |
| --- | --- |
| **Аргумент** | **Описание** |
| path | Строка, обозначающая путь в файловой системе, URL-aдpec или похожий на файл объект |
| sep или delimiter | Последовательность символов или регулярное выражение, служащее для разделения полей в строке |
| header | Номер строки, содержащей имена столбцов. По умолчанию равен О (первая строка). Если строки-заголовка нет, должен быть равен None |
| index\_col | Номера или имена столбцов, трактуемых как индекс строк в результирующем объекте. Может быть задан один номер (имя) или список номеров (имен), определяющий иерархический индекс |
| names | Список имен столбцов результирующего объекта, задается, если header= None |
| skiprows | Количество игнорируемых начальных строк или список номеров игнорируемых строк (нумерация начинается с 0) |
| na\_values | Последовательность значений, интерпретируемых как маркеры отсутствующих данных |
| comment | Один или несколько символов, начинающих комментарий, который продолжается до конца строки |
| parse\_dates | Пытаться разобрать данные как дату и время; по умолчанию False. Если равен True, то производится попытка разобрать все столбцы. Можно также задать список столбцов, которые следует объединить перед разбором (если, например, время и даты заданы в разных столбцах) |
| keep\_date\_col | В случае, когда для разбора данных столбцы объединяются, следует ли отбрасывать объединенные столбцы. По умолчанию тrue |
| converters | Словарь, содержащий отображение номеров или имен столбцов на функции. Например, , { ‘ foo ‘: f}означает, что нужно применить функцию f ко всем значением в столбце foo |
| dayfirst | При разборе потенциально неоднозначных дат предполагать международный формат (т. е. 7 /6/2012 означает «7 июня 2012»). По умолчанию False |
| date\_parser | Функция, применяемая для разбора дат |
| nrows | Количество читаемых строк от начала файла |
| iterator | Возвращает объект 'ГextParser для чтения файла порциями |
| chunksize | Размер порции при итерировании |
| skip\_footer | Сколько строк в конце файла игнорировать |
| verbose | Печатать разного рода информацию о ходе разбора, например, количество отсутствующих значений, помещенных в нечисловые столбцы |
| encoding | Кодировка текста в случае Unicode. Например, ‘utf-8’ означает, что текст представлен в кодировке UTF-8 |
| squeeze | Если в результате разбора данных оказалось, что имеется только один столбец, вернуть объект Series |
| thousands | Разделитель тысяч, например, · , · или · . · |

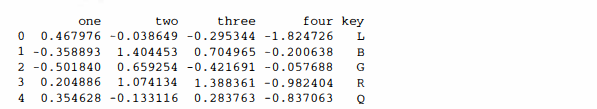
**Чтение текстовых файлов порциями**

Для обработки очень больших файлов или для того, чтобы определить правильный набор аргументов, необходимых для обработки большого файла, иногда требуется прочитать небольшой фрагмент файла или последовательно читать файл небольшими порциями

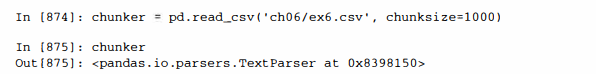


Чтобы прочитать только небольшое число строк (а не весь файл), нужно задать это число в параметре nrows:

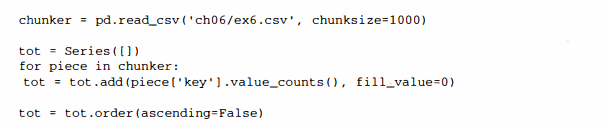




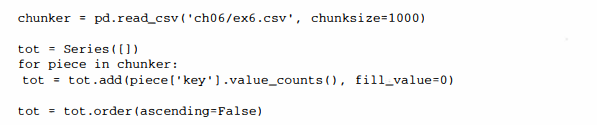
Для чтения файла порциями задайте с помощью параметра chunksize размер порции в строках:



Объект TextParser, возвращаемый функцией read\_csv, позволяет читать файл порциями размера chunksize. Например, можно таким образом итеративно читать файл ехб.csv, агрегируя счетчики значений в столбце ‘key’:



Объект TextParser, возвращаемый функцией read\_csv, позволяет читать файл порциями размера chunksize. Например, можно таким образом итеративно читать файл ехб. csv, агрегируя счетчики значений в столбце ‘key’:



Имеем:

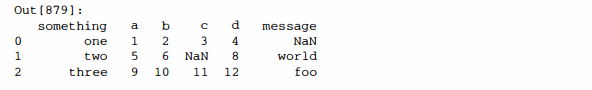


У объекта TextParser имеется также метод get\_chunk, который позволяет читать куски произвольного размера.

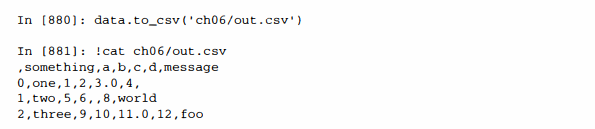
**Вывод данных в текстовом формате**

Данные можно экспортировать в формате с разделителями. Рассмотрим одну из приведенных выше операций чтения СSV-файла:

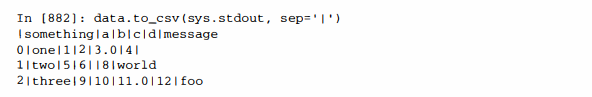




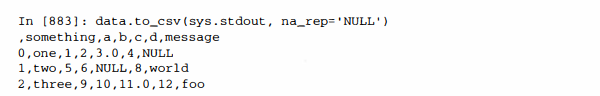
С помощью метода to\_csv объекта DataFrame мы можем вывести данные в файл через запятую:



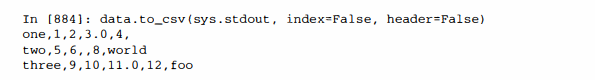
Конечно, допустимы и другие разделители (при выводе в sys.stdout результат отправляется на стандартный вывод, обычно на экран):



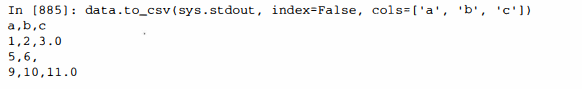
Отсутствующие значения представлены пустыми строками. Но можно вместо этого указать какой-нибудь маркер:



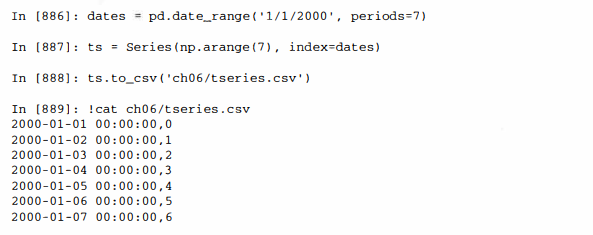
Если не указано противное, выводятся метки строк и столбцов. Но и те, и другие можно подавить:



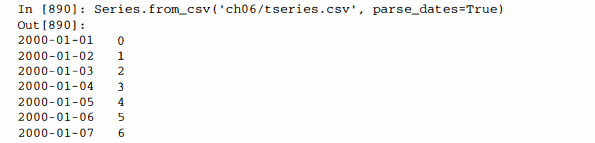
Можно также вывести лишь подмножество столбцов, задав их порядок:



У объекта Series также имеется метод to\_csv:



Подходящим образом задав параметры (заголовок отсутствует, первый столбец считается индексом), СSV-представление объекта Series можно прочитать методом read\_csv, но существует вспомогательный метод from\_csv, который позволяет сделать это немного проще:



Дополнительные сведения о методах to\_csv и from\_csv смотрите в строках документации в IPython.

**Ручная обработка данных в формате с разделителями**

Как правило, табличные данные можно загрузить с диска с помощью функции pandas.read\_taЫe и родственных ей. Но иногда требуется ручная обработка. Не так уж необычно встретить файл, в котором одна или несколько строк сформированы неправильно, что сбивает read\_taЫe. Для иллюстрации базовых средств рассмотрим небольшой СSV-файл:



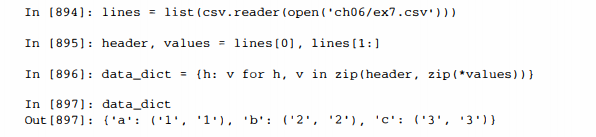
Для любого файла с односимвольным разделителем можно воспользоваться стандартным модулем Python csv. Для этого передайте открытый файл или объект, похожий на файл, методу csv. reader:



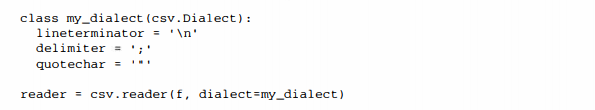
Итерирование файла с помощью объекта reader дает кортежи значений в каждой строке после удаления кавычек:



Далее можно произвести любые манипуляции, необходимые для преобразования данных к нужному виду. Например:



Встречаются различные вариации СSV-файлов. Для определения нового формата со своим разделителем, соглашением об употреблении кавычек и способе завершения строк необходимо определить простой подкласс класса csv.Dialect:



Параметры диалекта CSV можно задать также в виде именованных параметров csv. reader, не определяя подкласса:



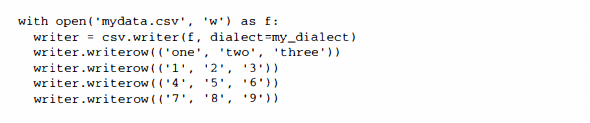
Возможные атрибуты csv. Dialect вместе с назначением каждого описаны в табл. 6.3.

**Таблица 6.3**. Параметры диалекта CSV

|  |  |
| --- | --- |
| **Аргумент** | **Описание** |
| delimiter | Односимвольная строка, определяющая разделитель полей. По умолчанию ’,’ |
| lineterminator | Завершитель стррк при выводе, по умолчанию ‘\r\n'. Объект reader игнорирует этот параметр, используя вместо него платформенное соглашение о концах строк |
| quotechar | Символ закавычивания для полей, содержащих специальные символы (например, разделитель). По умолчанию ' " ‘ |
| quoting | Соглашение об употреблении кавычек. Допустимые значения: csv.QUOTE\_ALL (заключать в кавычки все поля), csv. QUOTE\_MINIMAL (только поля, содержащие специальные символы, например разделитель), csv. QUOTE\_NONNUMERIC и csv. QUOTE\_NON (не заключать в кавычки). Полное описание см. в документации. По умолчанию QUOTE\_MINIMAL |
| skipinitialspace | Игнорировать пробелы после каждого разделителя. По умолчанию False |
| douЫequote | Как обрабатывать символ кавычки внутри поля. Если True, добавляется второй символ кавычки. Полное описание поведения см. в документации. |
| escapechar | Строка для экранирования разделителя в случае, когда quoting равно csv.QUOTE\_NONE. По умолчанию экранирование выключено. |

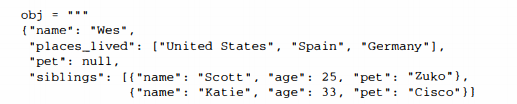
Если в файле употребляются более сложные или фиксированные многосимвольные разделители, то воспользоваться модулем csv не удастся. В таких случаях придется разбивать строку на части и производить другие действия по очистке данных, применяя метод строки split или метод регулярного выражения re.split.

Для записи файлов с разделителями вручную можно использовать метод csv.writer. Он принимает объект, который представляет открытый, допускающий за1шсь файл, и те же параметры диалекта и форматирования, что csv. reader:



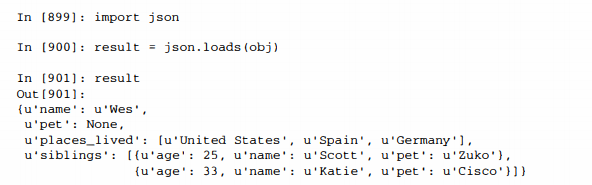
**Данные в формате JSON**

Формат JSON (JavaScript Object Notation) стал очень популярен для обмена данными по протоколу НТТР между веб-сервером и браузером или другим клиентским приложением. Этот формат обладает куда большей гибкостью, чем табличный текстовый формат типа CSV. Например:





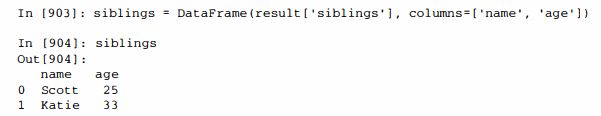
Данные в формате JSON очень напоминают код на Python с тем отличием, что отсутствующее значение обозначается null, и еще некоторыми нюансами (например, запрещается ставить запятую после последнего элемента списка). Базовыми типами являются объекты (словари), массивы (списки), строки, числа, булевы значения и null. Ключ любого объекта должен быть строкой. На Python существует несколько библиотек для чтения ц записи JSОN-данных. Здесь я воспользуюсь модулем json, потому что он входит в стандартную библиотеку Python. Для преобразования JSОN-строки в объект Python служит метод json.loads:



Напротив, метод json.durnps преобразует объект Python в формат JSON:



Как именно преобразовывать объект JSON или список таких объектов в DataFrame или еще какую-то структуру данных для анализа, решать вам. Для удобства предлагается возможность передать список объектов JSON конструктору DataFrame и выбрать подмножество полей данных:



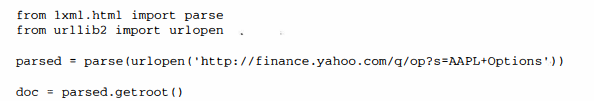
Более полный пример чтения и манипулирования данными в формате JSON (включая и вложенные записи) приведен при рассмотрении базы данных о продуктах питания USDA в следующей главе.

Сейчас идет работа по добавлению в pandas быстрых написанных на С средств для экспорта в формате JSON (to\_json) и декодирования данных в этом формате (from\_json). На момент написания этой книги они еще не были готовы.

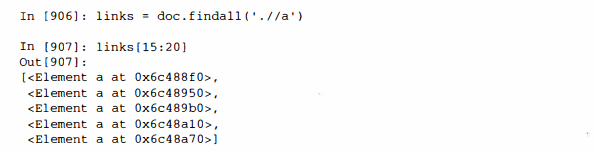
**XML и HTML: разбор веб-страниц**

На Python написано много библиотек для чтения и записи данных в вездесущих форматах HTML и XML. В частности, библиотека lxml (http.//lxml.de) известна высокой производительностью при разборе очень больших файлов. Для lxml имеется несколько программных интерфейсов; сначала я продемонстрирую интерфейс lxml. html для работы с HTML, а затем разберу ХМL-документ с помощью lxml.objectify. Многие сайты показывают данные в виде НТМL-таблиц, удобных для просмотра в браузере, по не предлагают их в таких машиночитаемых форматах, как JSON или XML. Так, например, обстоит дело с данными об биржевых опционах на сайте Yahoo! Finance. Для тех, кто не в курсе, скажу, что опцион — это производный финансовый инструмент (дериватив), который дает право покупать (опцион на покупку, или колл-опцион) или продавать (опцион на продажу, или пут-опцию) акции компании по некоторой цене (цене исполнения) в промежутке времени между текущим моментом и некоторым фиксированным моментом в будущем (конечной датой). Колл- и пут-опционы торгуются с разными ценами исполнения и конечными датами; эти данные можно найти в таблицах на сайте Yahoo! Finance.

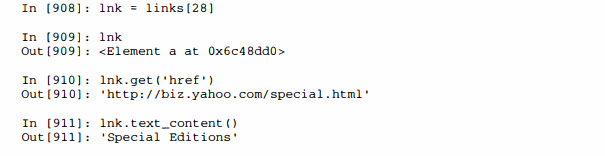
Для начала решите, с какого URL-aдpeca вы хотите загружать данные, затем откройте его с помощью средств из библиотеки urllib2 и разберите поток, пользуясь lxml:



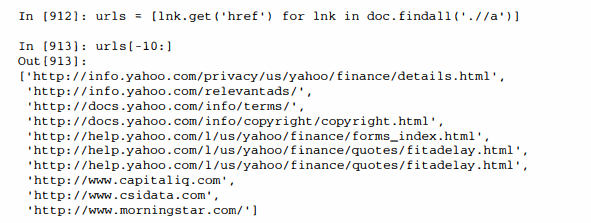
Имея этот объект, мы можем выбрать вес НТМL-теги указанного типа, например, теги tablе, внутри которых находятся интересующие нас данные. Для примера получим список всех гиперссылок в документе, они представляются в HTML тегом а. Вызовем метод findall корневого элемента документа, передав ему выражение XPath (это язык, на котором записываются «запросы» к документу):



Но это объекты, представляющие НТМL-элементы; чтобы получить URL и текст ссылки, нам нужно воспользоваться методом get элемента (для получения URL) или методом text\_content (для получения текста):



Таким образом, получение всех гиперссылок в документе сводится к списковому включению:



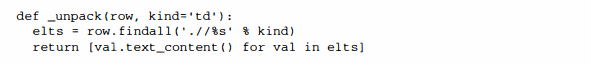
Что касается отыскания нужных таблиц в документе, то это делается методом проб и ошибок; на некоторых сайтах решение этой задачи упрощается, потому что таблица имеет атрибут id. Я нашел, какие таблицы содержат данные о колл- и пут-опционах:



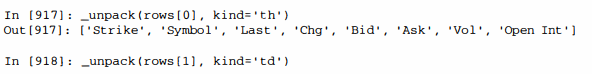
В каждой таблице имеется строка-заголовок, а за ней идут строки с данными:



Для всех строк, включая заголовок, мы хотим извлечь текст из каждой ячейки; в случае заголовка ячейками являются элементы th, а для строк данных - элементы td:

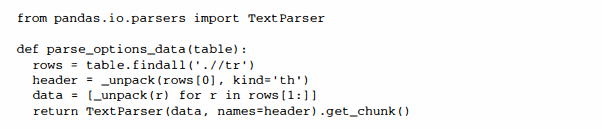


Таким образом, получаем:

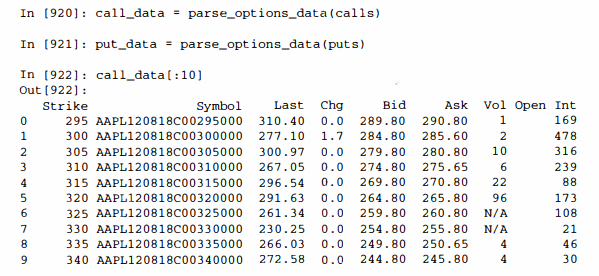




Теперь для преобразования данных в объект DataFrame осталось объединить все описанные шаги вместе. Поскольку числовые данные по-прежнему записаны в виде строк, возможно, потребуется преобразовать некоторые, но не все столбцы в формат с плавающей точкой. Это можно сделать и вручную, но, по счастью, в библиотеке pandas есть класс TextParser, который используется функцией read\_ csv и другими функциями разбора для автоматического преобразования типов:



Наконец, вызываем эту функцию разбора для табличных объектов lxml и получаем результат в виде DataFrame:

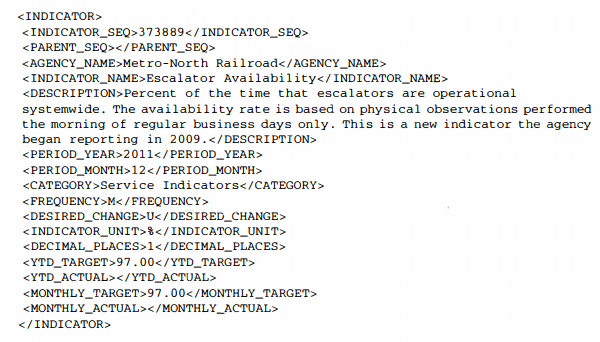


**Разбор XML с помощью lxml.objectify**

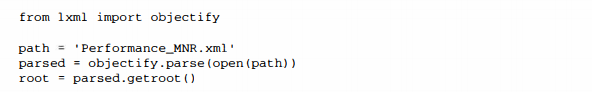
XML (расширяемый язык разметки)- еще один популярный формат представления структурированных данных, поддерживающий иерархически вложенные данные, снабженные метаданными. Текст этой книги на самом деле представляет собой набор больших ХМL-документов.

Выше я продемонстрировал применение библиотеки lxml и ее интерфейса lxml. html. А сейчас покажу альтернативный интерфейс, удобный для работы с ХМL-данными, - lxml.objectify.

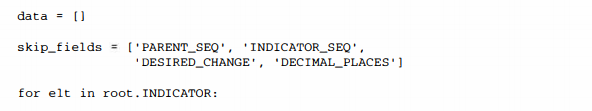
Управление городского транспорта Нью-Йорка (МТА) публикует временные ряды с данными о работе автобусов и электричек (http://www.mta.info/developers/ download.html). Мы сейчас рассмотрим данные о качестве обслуживания, хранящиеся в виде ХМL-файлов. Для каждой автобусной и железнодорожной компании существует свой файл (например, Performance\_MNR.xml для компании MetroNorth Railroad), содержащий данные за один месяц в виде последовательности таких ХМL-записей:

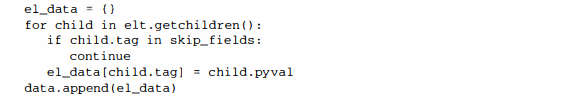


Используя lxml.objectify, мы разбираем файл и получаем ссылку на корневой узел ХМL-документа от метода getroot:

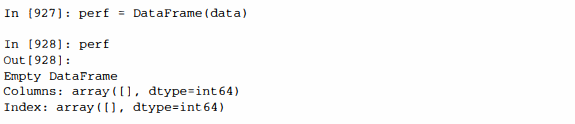


Свойство root.INDICATOR возвращает генератор, последовательно отдающий все элементы. Для каждой записи мы заполняем словарь имен тегов (например, YTD\_ACTUAL) значениями данных (некоторые теги пропускаются):





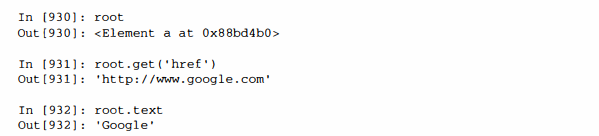
Наконец, преобразуем этот список словарей в объект DataFrame:



ХМL-документы могут быть гораздо сложнее, чем в этом примере. В частности, в каждом элементе могут быть метаданные. Рассмотрим тег гиперссылки в формате HTML, который является частным случаем XML:

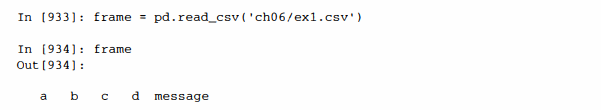


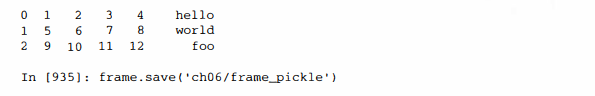
Теперь мы можем обратиться к любому атрибуту тега (например, href) или к тексту ссылки:



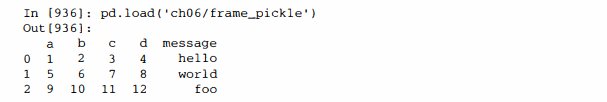
**Двоичные форматы данных**

Один их самых простых способов эффективного хранения данных в двоичном формате - воспользоваться встроенным в Python методом сериализации pickle. Поэтому у всех объектов pandas есть метод save, который сохраняет данные на диске в виде рiсklе-файла:





Прочитать данные с диска позволяет метод pandas.load, также упрощающий интерфейс с pickle:



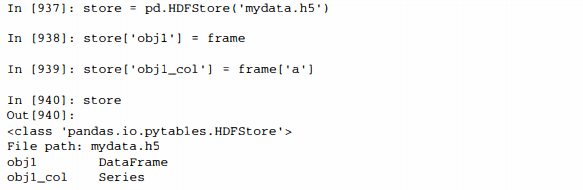
pickle рекомендуется использовать только для краткосрочного хранения. Проблема в том, что невозможно гарантировать неизменность формата: сегодня вы сериализовали объект в формате pickle, а следующая версия библиотеки не сможет его десериализовать. Я приложил все усилия к тому, чтобы в pandas такое не случалось, но, возможно, наступит момент. когда придется «поломать» формат pickle.

**ФорматНDF5**

Существует немало инструментов, предназначенных для эффективного чтения и записи больших объемов научных данных в двоичном формате. Популярна, в частности, библиотека промышленного качества HDFS, написанная па С и имеющая интерфейсы ко многим языкам, в том числе Java, Python и MATIAB. Акроним «HDF» в ее названии означает hierarchical data format (иерархический формат данных). Каждый НDFS-файл содержит внутри себя структуру узлов. напоминающую файловую систему, которая позволяет хранить несколько наборов данных вместе с относящимися к ним метаданными. В отличие от более простых форматов, HDFS поддерживает сжатие на лету с помощью различных алгоритмов сжатия, что позволяет более эффективно хранить повторяющиеся комбинации данных. Для очень больших наборов данных, которые не помещаются в память, HDFS - отличный выбор, потому что даст возможность эффективно читать и записывать небольшие участки гораздо больших массивов.

К библиотеке HDFS существует целых два интерфейса из Python: РуТаblеs и hSpy, в которых приняты совершенно различные подходы. hSpy - прямой, хотя и высокоуровневый интерфейс к HDFS API, тогда как РуТаblеs абстрагирует многие детали HDFS с целью предоставления нескольких гибких контейнеров данных, средств индексирования таблиц, средств запроса и поддержки некоторых вычислений, отсутствующих в исходной библиотеке.

В pandas имеется минимальный похожий на слоnар1, класс HDFStore, в котором для сохранения объектов pandas используется интерфейс РуТаblеs:



Объекты из НDFS-файла можно извлекать, как из словаря:



Если вы собираетесь работать с очень большими объемами данных, то я рекомендую изучить РуТаblеs и hSpy и посмотреть, в какой мере они отвечают вашим потребностям. Поскольку многие задачи анализа данных ограничены, прежде всего, скоростью ввода-вывода (а не быстродействием процессора), использование средства типа HDFS способно существенно ускорить работу приложения.

HDF5 не является базой данных. Лучше всего она приспособлена для работы с наборами данных, коtорые записываются один раз, а читаются многократно. Данные можно добавлять в файл в любой момент, но если это делают одновременно несколько клиентов, то файл можно повредить.

**Чтение файлов Microsoft Excel**

В pandas имеется также поддержка для чтения табличных данных в формате Excel 2003 (и более поздних версией) с помощью класса ExcelFile. На внутреннем уровне ExcelFile пользуется пакетами xlrd и openpyxl, поэтому их нужно предварительно установить. Для работы с ExcelFile создайте его экземпляр, передав конструктору путь к файлу с расширением xls или xlsx:

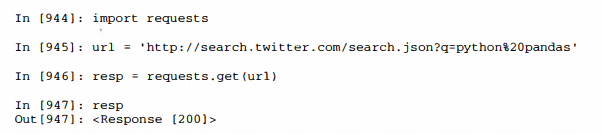
xls\_file = pd.ExcelFile('data.xls')

Прочитать данные из рабочего листа в объект DataFrame позволяет метод parse

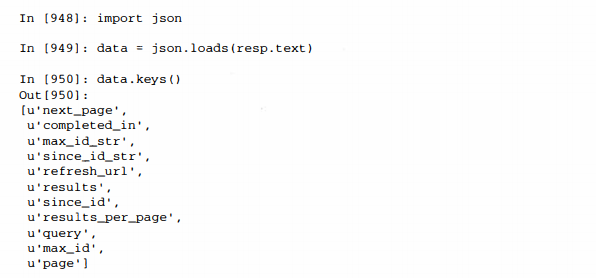
tаЫе = xls\_file.parse('Sheetl')

Взаимодействие с HTML и Web АРI

Многие сайты предоставляют открытый API для получения данных в формате JSON или каком-то другом. Получить доступ к таким API из Python можно разны Многие сайты предоставляют открытый API для получения данных в формате JSON или каком-то другом. Получить доступ к таким API из Python можно разными способами; я рекомендую простой пакет requests (http://docs.python-requests. 01g). Для поиска по словам «python pandas» в Твиттере мы можем отправить такой HTTP-запрос GET:



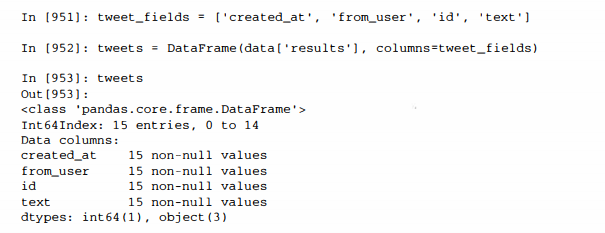
У объекта Response имеет атрибут text, в котором хранится содержимое ответа на запрос GET. Многие API в веб возвращают JSON-cтpoкy, которую следует загрузить в объект Python:



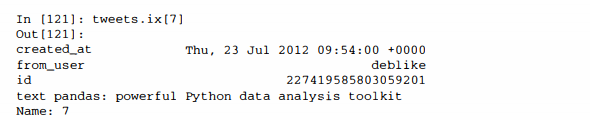
Поле ответа results содержит список твитов, каждый из которых представлен таким словарем Python:



Далее мы можем построить список интересующих пас полей твита aи передать его конструктору DataFrame:



Теперь в каждой строке DataFrame находятся данные, извлеченные из одного твита:



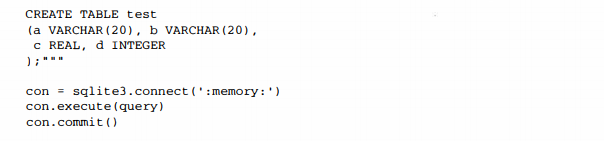
Приложив толику усилий, вы сможете создать высокоуровневые интерфейсы к популярным в веб API, которые будут возвращать объекты DataFrame, легко поддающиеся анализу.

**Взаимодействие с базами данных**

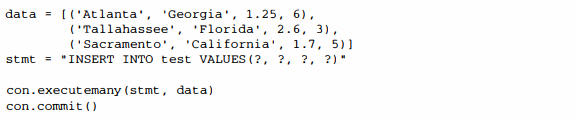
Во многие приложения данные поступают не из файлов, потому что для хранения больших объемов данных текстовые файлы неэффективны. Широко используются реляционные базы данных на основе SQL (например, SQL Server, PostgreSQL и MySQL), а равно так называемые базы данных NoSQL, быстро набирающие популярность. Выбор базы данных обычно диктуется производительностью, необходимостью поддержания целостности данных и потребностями приложения в масштабируемости.

Загрузка данных из реляционной базы в DataFrame производится довольно прямолинейно, и в pandas есть несколько функций для упрощения этой процедуры. В качестве примера я возьму базу данных SQLite, целиком размещающуюся в памяти, и драйвер sqliteЗ, включенный в стандартную библиотеку Python:

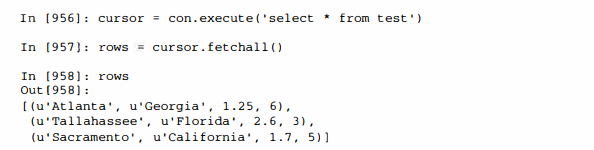




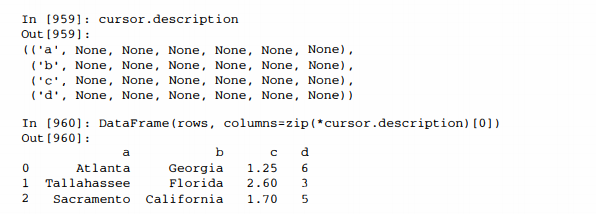
Затем вставлю несколько строк в таблицу:



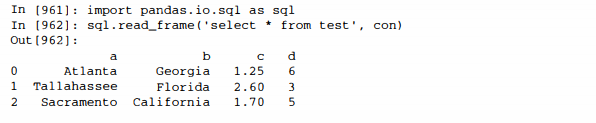
Большинство драйверов SQL, имеющихся в Python (PyODBC, psycopg2, MySQLdb, pyшssql и т. д. ), при выборе данных из таблицы возвращают список кортежей:



Этот список кортежей можно передать конструктору DataFrame, но необходимы еще имена столбцов, содержащиеся в атрибуте курсора description:



Такое переформатирование не хочется выполнять при каждом запросе к базе данных. В pandas, точнее в модуле paпdas. io. sql, имеется функция read\_frame, которая упрощает эту процедуру. Нужно просто передать команду select и объект соединения:



**Чтение и сохранение данных в MongoDB**

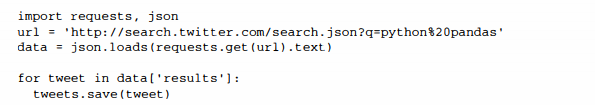
Базы данных NoSQL весьма многообразны. Есть простые хранилища ключей и значений, напоминающие словарь, например BerkeleyDВ или Tokyo Cabinet, а есть и документо-ориентированные базы, в которых похожий на словарь объект является основной единицей хранения. Я решил взять в качестве примера MongoDB (http://mongodb.org). Я запустил локальный экземпляр MongoDB на своей машине и подключился к ее порту по умолчанию с помощью pymongo, официального драйвера для MongoDB:



В MongoDB документы хранятся в коллекциях в базе данных. Каждый экземпляр сервера MongoDB может обслуживать несколько баз данных, а в каждой базе может быть несколько коллекций. Допустим, я хочу сохранить данные, полученные ранее из Твиттера. Прежде всего, получаю доступ к коллекции твитов (пока пустой):



Затем запрашиваю список твитов и записываю каждый из них в коллекцию методом tweets. save (который сохраняет словарь Python в базе MongoDB):



Если затем я захочу получить все мои твиты из коллекции, то должен буду опросить ее, как показано ниже:



Возвращенный курсор — это итератор, который отдает каждый документ в виде словаря. Как и раньше, я могу преобразовать этот документ в DataFrame, возможно, ограничившись некоторым подмножеством полей данных:

