Python
и анализ данных

Уэс Маккинни

ДМК ПРЕСС издательство

O'REILLY®

Уэс Маккинли

Python
и анализ данных

Python for Data Analysis

Wes McKinney

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Python и анализ данных

Уэс Маккинли

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО
Москва, 2015

УДК 004.438Python:004.6
ББК 32.973.22
М15

М15 Уэс Маккинли

Python и анализ данных / Пер. с англ. Слинкин А. А. – М.: ДМК Пресс, 2015. – 482 с.: ил.

ISBN 978-5-97060-315-4

Книгу можно рассматривать как современное практическое введение в разработку научных приложений на Python, ориентированных на обработку данных. Описаны те части языка Python и библиотеки для него, которые необходимы для эффективного решения широкого круга аналитических задач: интерактивная оболочка IPython, библиотеки NumPy и pandas, библиотека для визуализации данных matplotlib и др.

Издание идеально подойдет как аналитикам, только начинающим осваивать обработку данных, так и опытным программистам на Python, еще не знакомым с научными приложениями.

УДК 004.438Python:004.6
ББК 32.973.22

ISBN 978-1-449-31979-3 (англ.)
ISBN 978-5-97060-315-4 (рус.)

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

За последние 10 лет вокруг языка Python образовалась и активно развивается целая экосистема библиотек с открытым исходным кодом. К началу 2011 года у меня сложилось стойкое ощущение, что нехватка централизованных источников учебных материалов по анализу данных и математической статистике становится камнем преткновения на пути молодых программистов на Python, которым такие приложения нужны по работе. Основные проекты, связанные с анализом данных (в особенности NumPy, IPython, matplotlib и pandas), к тому времени стали уже достаточно зрелыми, чтобы про них можно было написать книгу, которая не устареет сразу после выхода. Поэтому я набрался смелости заняться этим делом. Я был бы очень рад, если бы такая книга существовала в 2007 году, когда я приступал к использованию Python для анализа данных. Надеюсь, вам она окажется полезной, и вы сумеете с успехом воспользоваться описываемыми инструментами в собственной работе.

Графические выделения

В книге применяются следующие графические выделения:

Курсив

Новые термины, URL-адреса, адреса электронной почты, имена и расширения имен файлов.

`Моноширинный`

Листинги программ, а также элементы кода в основном тексте: имена переменных и функций, базы данных, типы данных, переменные окружения, предложения и ключевые слова языка.

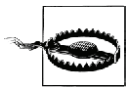
`Моноширинный полужирный`

Команды или иной текст, который должен быть введен пользователем буквально.

`Моноширинный курсив`

Текст, вместо которого следует подставить значения, заданные пользователем или определяемые контекстом.

Так обозначается совет, рекомендация или замечание общего характера.



Так обозначается предупреждение или предостережение.

ГЛАВА 1.
Предварительные сведения

О чем эта книга?

Эта книга посвящена вопросам преобразования, обработки, очистки данных и вычислениям на языке Python. Кроме того, она представляет собой современное практическое введение в научные и инженерные расчеты на Python, ориентированное на приложения для обработки больших объемов данных. Это книга о тех частях языка Python и написанных для него библиотек, которые необходимы для эффективного решения широкого круга задач анализа данных. Но в ней вы *не* найдете объяснений аналитических методов с привлечением Python в качестве языка реализации.

Говоря «данные», я имею в виду, прежде всего, *структурированные данные*; это намеренно расплывчатый термин, охватывающий различные часто встречающиеся виды данных, как то:

- многомерные списки (матрицы);
- табличные данные, когда данные в разных столбцах могут иметь разный тип (строки, числа, даты или еще что-то). Сюда относятся данные, которые обычно хранятся в реляционных базах или в файлах с запятой в качестве разделителя;
- данные, представленные в виде нескольких таблиц, связанных между собой по ключевым столбцам (то, что в SQL называется первичными и внешними ключами);
- равноотстоящие и неравноотстоящие временные ряды.

Этот список далеко не полный. Значительную часть наборов данных можно преобразовать к структурированному виду, более подходящему для анализа и моделирования, хотя сразу не всегда очевидно, как это сделать. В тех случаях, когда это не удается, иногда есть возможность извлечь из набора данных структурированное множество признаков. Например, подборку новостных статей можно преобразовать в таблицу частот слов, к которой затем применить анализ эмоциональной окраски.

Большинству пользователей электронных таблиц типа Microsoft Excel, пожалуй, самого широко распространенного средства анализа данных, такие виды данных хорошо знакомы.

Почему именно Python?

Для многих людей (и меня в том числе) Python — язык, в который нельзя не влюбиться. С момента своего появления в 1991 году Python стал одним из самых популярных динамических языков программирования наряду с Perl, Ruby и другими. Относительно недавно Python и Ruby приобрели особую популярность как средства создания веб-сайтов в многочисленных каркасах, например Rails (Ruby) и Django (Python). Такие языки часто называют *скриптовыми*, потому что они используются для быстрого написания небольших программ – *скриптов*. Лично мне термин «скриптовый язык» не нравится, потому что он наводит на мысль, будто для создания ответственного программного обеспечения язык не годится. Из всех интерпретируемых языков Python выделяется большим и активным сообществом *научных расчетов*. Применение Python для этой цели в промышленных и академических кругах значительно расширилось с начала 2000-х годов.

В области анализа данных и интерактивных научно-исследовательских расчетов с визуализацией результатов Python неизбежно приходится сравнивать со многими предметно-ориентированными языками программирования и инструментами – с открытым исходным кодом и коммерческими – такими, как R, MATLAB, SAS, Stata и другими. Сравнительно недавнее появление улучшенных библиотек для Python (прежде всего, pandas) сделало его серьезным конкурентом в решении задач манипулирования данными. В сочетании с достоинствами Python как универсального языка программирования это делает его отличным выбором для создания приложений обработки данных.

Python как клей

Своим успехом в качестве платформы для научных расчетов Python отчасти обязан простоте интеграции с кодом на C, C++ и FORTRAN. Во многих современных вычислительных средах применяется общий набор унаследованных библиотек, написанных на FORTRAN и C, содержащих реализации алгоритмов линейной алгебры, оптимизации, интегрирования, быстрого преобразования Фурье и других. Поэтому многочисленные компании и национальные лаборатории используют Python как «клей» для объединения написанных за 30 лет программ.

Многие программы содержат небольшие участки кода, на выполнение которых уходит большая часть времени, и большие куски «склеивающего кода», который выполняется нечасто. Во многих случаях время выполнения склеивающего кода несущественно, реальную отдачу дает оптимизация узких мест, которые иногда имеет смысл переписать на низкоуровневом языке типа C.

За последние несколько лет на одно из первых мест в области создания быстрых компилируемых расширений Python и организации интерфейса с кодом на C и C++ вышел проект Cython (*http://cython.org*).

Решение проблемы «двух языков»

Во многих организациях принято для научных исследований, создания опытных образцов и проверки новых идей использовать предметно-ориентированные языки типа MATLAB или R, а затем переносить удачные разработки в производственную систему, написанную на Java, C# или C++. Но все чаще люди приходят к выводу, что Python подходит не только для стадий исследования и создания прототипа, но и для построения самих производственных систем. Я полагаю, что компании все чаще будут выбирать этот путь, потому что использование одного и того же набора программных средств учеными и технологами несет несомненные выгоды организации.

Недостатки Python

Python – великолепная среда для создания приложений для научных расчетов и большинства систем общего назначения, но тем не менее существуют задачи, для которых Python не очень подходит.

Поскольку Python – интерпретируемый язык программирования, в общем случае написанный на нем код работает значительно медленнее, чем эквивалентный код на компилируемом языке типа Java или C++. Но поскольку *время программиста* обычно стоит гораздо дороже *времени процессора*, многих такой компромисс устраивает. Однако в приложениях, где задержка должна быть очень мала (например, в торговых системах с большим количеством транзакций), время, потраченное на программирование на низкоуровневом и не обеспечивающем максимальную продуктивность языке типа C++, во имя достижения максимальной производительности, будет потрачено не зря.

Python – не идеальный язык для программирования многопоточных приложений с высокой степенью параллелизма, особенно при наличии многих потоков, активно использующих процессор. Проблема связана с наличием *глобальной блокировки интерпретатора* (GIL) – механизма, который не дает интерпретатору исполнять более одной команды байт-кода Python в каждый момент времени. Объяснение технических причин существования GIL выходит за рамки этой книги, но на данный момент представляется, что GIL вряд ли скоро исчезнет. И хотя во многих приложениях обработки больших объектов данных для обеспечения приемлемого времени приходится организовывать кластер машин, встречаются все же ситуации, когда более желательна однопроцессная многопоточная система.

Я не хочу сказать, что Python вообще непригоден для исполнения многопоточного параллельного кода; просто такой код нельзя выполнять в одном процессе Python. Например, в проекте Cython реализована простая интеграция с OpenMP, написанной на C библиотеке параллельных вычислений, позволяющая распараллеливать циклы и тем самым значительно ускорять работу численных алгоритмов.

Необходимые библиотеки для Python

Для читателей, плохо знакомых с экосистемой Python и используемыми в книге библиотеками, я приведу краткий обзор библиотек.

NumPy

NumPy, сокращение от «Numerical Python», – основной пакет для выполнения научных расчетов на Python. Большая часть этой книги базируется на NumPy и построенных поверх него библиотек. В числе прочего он предоставляет:

- быстрый и эффективный объект многомерного массива *ndarray*;
- функции для выполнения вычислений над элементами одного массива или математических операций с несколькими массивами;
- средства для чтения и записи на диски наборов данных, представленных в виде массивов;
- операции линейной алгебры, преобразование Фурье и генератор случайных чисел;
- средства для интеграции с кодом, написанным на C, C++ или Fortran.

Помимо быстрых средств работы с массивами, одной из основных целей NumPy в части анализа данных является организация контейнера для передачи данных между алгоритмами. Как средство хранения и манипуляции данными массивы NumPy куда эффективнее встроенных в Python структур данных. Кроме того, библиотеки, написанные на низкоуровневом языке типа C или Fortran, могут работать с данными, хранящимися в массиве NumPy, вообще без копирования.

pandas

Библиотека pandas предоставляет структуры данных и функции, призванные сделать работу со структурированными данными простым, быстрым и выразительным делом. Как вы вскоре убедитесь, это один из основных компонентов, превращающих Python в мощный инструмент продуктивного анализа данных. Основной объект pandas, используемый в этой книге, – `DataFrame` – двумерная таблица, в которой строки и столбцы имеют метки:

```
>>> frame
     total_bill   tip     sex      smoker  day     time          size
1    16.99        1.01    Female   No      Sun     Dinner        2
2    10.34        1.66    Male     No      Sun     Dinner        3
3    21.01        3.5     Male     No      Sun     Dinner        3
4    23.68        3.31    Male     No      Sun     Dinner        2
5    24.59        3.61    Female   No      Sun     Dinner        4
6    25.29        4.71    Male     No      Sun     Dinner        4
7    8.77         2       Male     No      Sun     Dinner        2
8    26.88        3.12    Male     No      Sun     Dinner        4
9    15.04        1.96    Male     No      Sun     Dinner        2
10   14.78        3.23    Male     No      Sun     Dinner        2
```

Библиотека pandas сочетает высокую производительность средств работы с массивами, присущую NumPy, с гибкими возможностями манипулирования дан-

ными, свойственными электронным таблицам и реляционным базам данных (например, на основе SQL). Она предоставляет развитые средства индексирования, позволяющие без труда изменять форму наборов данных, формировать продольные и поперечные срезы, выполнять агрегирование и выбирать подмножества. В этой книге библиотека pandas будет нашим основным инструментом.

Для разработки финансовых приложений pandas предлагает богатый набор высокопроизводительных средств анализа временных рядов, специально ориентированных на финансовые данные. На самом деле, я изначально проектировал pandas как удобный инструмент анализа именно финансовых данных.

Пользователям языка статистических расчетов R название DataFrame покажется знакомым, потому что оно выбрано по аналогии с объектом `data.frame` в R. Однако они не идентичны: функциональность `data.frame` является собственным подмножеством той, что предлагает `DataFrame`. Хотя эта книга посвящена Python, я время от времени будут проводить сравнения с R, потому что это одна из самых распространенных сред анализа данных с открытым исходным кодом, знакомая многим читателям.

Само название pandas образовано как от *panel data* (панельные данные), применяемого в эконометрике термина для обозначения многомерных структурированных наборов данных, так и от фразы *Python data analysis*.

matplotlib

Библиотека matplotlib – самый популярный в Python инструмент для создания графиков и других способов визуализации двумерных данных. Первоначально она была написана Джоном Д. Хантером (John D. Hunter, JDH), а теперь сопровождается большой группой разработчиков. Она отлично подходит для создания графиков, пригодных для публикации. Интегрирована с IPython (см. ниже), что позволяет организовать удобное интерактивное окружение для визуализации и исследования данных. Графики *интерактивны* – можно увеличить масштаб какого-то участка графика и выполнять панорамирование с помощью панели инструментов в окне графика.

IPython

IPython – компонент стандартного набора инструментов научных расчетов на Python, который связывает все воедино. Он обеспечивает надежную высокопродуктивную среду для интерактивных и исследовательских расчетов. Это оболочка Python с дополнительными возможностями, имеющая целью ускорить написание, тестирование и отладку кода на Python. Особенно она полезна для работы с данными и их визуализации с помощью matplotlib. Я почти всегда использую IPython в собственной работе для прогона, отладки и тестирования кода.

Помимо стандартных средств консольной оболочки, IPython предоставляет:

- HTML-блокнот в духе программы Mathematica для подключения к IPython с помощью веб-браузера (подробнее об этом ниже);

- консоль с графическим интерфейсом пользователя на базе библиотеки Qt, включающую средства построения графиков, многострочный редактор и подсветку синтаксиса;
- инфраструктуру для интерактивных параллельных и распределенных вычислений.

Я посвятил отдельную главу оболочке IPython и способам оптимальной работы с ней. Настоятельно рекомендую использовать ее при чтении этой книги.

SciPy

SciPy – собрание пакетов, предназначенных для решения различных стандартных вычислительных задач. Вот несколько из них:

- `scipy.integrate`: подпрограммы численного интегрирования и решения дифференциальных уравнений;
- `scipy.linalg`: подпрограммы линейной алгебры и разложения матриц, дополняющие те, что включены в `numpy.linalg`;
- `scipy.optimize`: алгоритмы оптимизации функций (нахождения минимумов) и поиска корней;
- `scipy.signal`: средства обработки сигналов;
- `scipy.sparse`: алгоритмы работы с разреженными матрицами и решения разреженных систем линейных уравнений;
- `scipy.special`: обертка вокруг SPECFUN, написанной на Fortran библиотеке, содержащей реализации многих стандартных математических функций, в том числе гамма-функции;
- `scipy.stats`: стандартные непрерывные и дискретные распределения вероятностей (функции плотности вероятности, формирования выборки, функции непрерывного распределения вероятности), различные статистические критерии и дополнительные описательные статистики;
- `scipy.weave`: средство для встраивания кода на C++ с целью ускорения вычислений с массивами.

Совместно NumPy и SciPy образуют достаточно полную замену значительной части системы MATLAB и многочисленных дополнений к ней.

Установка и настройка

Поскольку Python используется в самых разных приложениях, не существует единственно верной процедуры установки Python и необходимых дополнительных пакетов. У многих читателей, скорее всего, нет среды, подходящей для научных применений Python и проработки этой книги, поэтому я приведу подробные инструкции для разных операционных систем. Я рекомендую использовать следующие базовые дистрибутивы Python:

Enthought Python Distribution: ориентированный на научные применения дистрибутив от компании Enthought (*http://www.enthought.com*). Включает EPDFree – бесплатный дистрибутив (содержит NumPy, SciPy, matplotlib, Chaco и IPython), и

EPD Full — полный комплект, содержащий более 100 научных пакетов для разных предметных областей. EPD Full бесплатно поставляется для академического использования, а прочим пользователям предлагается платная годовая подписка.

Python(x,y) (*http://pythonxy.googlecode.com*): бесплатный ориентированный на научные применения дистрибутив для Windows.

В инструкциях ниже подразумевается EPDFree, но вы можете выбрать любой другой подход, все зависит от ваших потребностей. На момент написания этой книги EPD включает версию Python 2.7, хотя в будущем это может измениться. После установки на вашей машине появятся следующие готовые к импорту пакеты:

- базовые пакеты для научных расчетов: NumPy, SciPy, matplotlib и IPython (входят в EPDFree);
- зависимости для IPython Notebook: tornado и pyzmq (также входят в EPDFree);
- pandas (версии 0.8.2 или выше).

По ходу чтения книги вам могут понадобиться также следующие пакеты: statsmodels, PyTables, PyQt (или эквивалентный ему PySide), xlrd, lxml, basemap, pymongo и requests. Они используются в разных примерах. Устанавливать их необязательно, и я рекомендую не торопиться с этим до момента, когда они действительно понадобятся. Например, сборка PyQt или PyTables из исходных кодов в OS X или Linux – довольно муторное дело. А пока нам важно получить минимальную работоспособную среду: EPDFree и pandas.

Сведения обо всех Python-пакетах, ссылки на установщики двоичных версий и другую справочную информацию можно найти в указателе пакетов Python (Python Package Index – PyPI, *http://pypi.python.org*). Заодно это отличный источник для поиска новых Python-пакетов.

Во избежание путаницы я не стану обсуждать более сложные средства управления окружением такие, как pip и virtualenv. В Интернете можно найти немало отличных руководств по ним.

Некоторым пользователям могут быть интересны альтернативные реализации Python, например IronPython, Jython или PyPy. Но для работы с инструментами, представленными в этой книге, в настоящее время необходим стандартный написанный на C интерпретатор Python, известный под названием CPython.

Windows

Для начала установки в Windows скачайте установщик EPDFree с сайта *http://www.enthought.com*; это файл с именем `epd_free-7.3-1-winx86.msi`. Запустите его и согласитесь с предлагаемым по умолчанию установочным каталогом `C:\Python27`. Если в этом каталоге уже был установлен Python, то рекомендую

предварительно удалить его вручную (или с помощью средства «Установка и удаление программ»).

Затем нужно убедиться, что Python прописался в системной переменной PATH и что не возникло конфликтов с ранее установленными версиями Python. Откройте консоль (выберите из меню «Пуск» пункт «Выполнить...» и наберите `cmd.exe`). Попробуйте запустить интерпретатор Python, введя команду `python`. Должно появиться сообщение, в котором указана установленная версия EPDFree:

```
C:\Users\Wes>python
Python 2.7.3 |EPD_free 7.3-1 (32-bit)| (default, Apr 12 2012, 14:30:37) on win32
Type "credits", "demo" or "enthought" for more information.
>>>
```

Если в сообщении указана другая версия EPD или вообще ничего не запускается, то нужно привести в порядок переменные среды Windows. В Windows 7 начните вводить фразу «environment variables» в поле поиска программ и выберите раздел `Edit environment variables for your account`. В Windows XP нужно перейти в `Панель управления > Система > Дополнительно > Переменные среды`. В появляющемся окне найдите переменную `Path`. В ней должны присутствовать следующие два каталога, разделенные точкой с запятой:

```
C:\Python27;C:\Python27\Scripts
```

Если вы ранее устанавливали другие версии Python, удалите относящиеся в Python каталоги из системы и из переменных `Path`. После манипуляций с путями консоль необходимо перезапустить, чтобы изменения вступили в силу.

После того как Python удалось успешно запустить из консоли, необходимо установить pandas. Проще всего для этой цели загрузить подходящий двоичный установщик с сайта *http://pypi.python.org/pypi/pandas*. Для EPDFree это будет файл `pandas-0.9.0.win32-py2.7.exe`. После того как установщик отработает, запустим IPython и проверим, что все установилось правильно, для этого импортируем pandas и построим простой график с помощью matplotlib:

```
C:\Users\Wes>ipython --pylab
Python 2.7.3 |EPD_free 7.3-1 (32-bit)|
Type "copyright", "credits" or "license" for more information.

IPython 0.12.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??" for extra details.

Welcome to pylab, a matplotlib-based Python environment [backend: WXAgg].
For more information, type 'help(pylab)'.

In [1]: import pandas

In [2]: plot(arange(10))
```

Если все нормально, то не будет никаких сообщений об ошибках и появится окно с графиком. Можно также проверить, что HTML-блокнот IPython HTML работает правильно:

```
$ ipython notebook --pylab=inline
```

Если в Windows вы обычно используете Internet Explorer, то для работы блокнота IPython, скорее всего, придется установить Mozilla Firefox или Google Chrome.

Дистрибутив EPDFree для Windows содержит только 32-разрядные исполняемые файлы. Если вам необходим 64-разрядный дистрибутив, то проще всего взять EPD Full. Если же вы предпочитаете устанавливать все с нуля и не платить EPD за подписку, то Кристоф Гольке (Christoph Gohlke) из Калифорнийского университета в Ирвайне опубликовал неофициальные двоичные 32- и 64-разрядные установщики для всех используемых в книге пакетов (*http://www.lfd.uci.edu/~gohlke/pythonlibs/*).

Apple OS X

Перед тем как приступать к установке в OS X, необходимо установить Xcode – комплект средств разработки ПО от Apple. Нам понадобится из него gcc – комплект компиляторов для C и C++. Установщик Xcode можно найти на установочном DVD, поставляемом вместе с компьютером, или скачать непосредственно с сайта Apple.

После установки Xcode запустите терминал (Terminal.app), перейдя в меню `Applications > Utilities`. Введите `gcc` и нажмите клавишу **Enter**. Должно появиться такое сообщение:

```
$ gcc
i686-apple-darwin10-gcc-4.2.1: no input files
```

Теперь необходимо установить EPDFree. Скачайте установщик, который должен представлять собой образ диска с именем вида `epd_free-7.3-1-macosx-i386.dmg`. Дважды щелкните мышью по файлу `dmg`-файлу, чтобы смонтировать его, а затем дважды щелкните по `mpkg`-файлу, чтобы запустить установщик.

Установщик автоматически добавит путь к исполняемому файлу EPDFree в ваш файл `.bash_profile`, его полный путь `/Users/ваше_имя/.bash_profile`:

```
# Setting PATH for EPD_free-7.3-1
PATH="/Library/Frameworks/Python.framework/Versions/Current/bin:${PATH}"
export PATH
```

Если на последующих шагах возникнут проблемы, проверьте файл `.bash_profile` – быть может, придется добавить указанный каталог в переменную PATH вручную.

Пришло время установить pandas. Выполните в терминале такую команду:

```
$ sudo easy_install pandas
Searching for pandas
Reading http://pypi.python.org/simple/pandas/
Reading http://pandas.pydata.org
Reading http://pandas.sourceforge.net
Best match: pandas 0.9.0
Downloading http://pypi.python.org/packages/source/p/pandas/pandas-0.9.0.zip
Processing pandas-0.9.0.zip
Writing /tmp/easy_install-H5mIX6/pandas-0.9.0/setup.cfg
Running pandas-0.9.0/setup.py -q bdist_egg --dist-dir /tmp/easy_install-H5mIX6/
pandas-0.9.0/egg-dist-tmp-RhLG0z

Adding pandas 0.9.0 to easy-install.pth file
Installed /Library/Frameworks/Python.framework/Versions/7.3/lib/python2.7/
site-packages/pandas-0.9.0-py2.7-macosx-10.5-i386.egg
Processing dependencies for pandas
Finished processing dependencies for pandas
```

Чтобы убедиться в работоспособности, запустите IPython в режиме Pylab и проверьте импорт pandas и интерактивное построение графика:

```
$ ipython --pylab
22:29 ~/VirtualBox VMs/WindowsXP $ ipython
Python 2.7.3 |EPD_free 7.3-1 (32-bit)| (default, Apr 12 2012, 11:28:34)
Type "copyright", "credits" or "license" for more information.

IPython 0.12.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

Welcome to pylab, a matplotlib-based Python environment [backend: WXAgg].
For more information, type 'help(pylab)'.

In [1]: import pandas

In [2]: plot(arange(10))
```

Если все нормально, появится окно графика, содержащее прямую линию.

GNU/Linux

В некоторые, но не во все дистрибутивы Linux включены достаточно актуальные версии всех необходимых Python-пакетов, и их можно установить с помощью встроенного средства управления пакетами, например `apt`. Я продемонстрирую установку на примере EPDFree, потому что она типична для разных дистрибутивов.

Детали варьируются в зависимости от дистрибутива Linux, я буду ориентироваться на дистрибутив Debian, на базе которого построены такие системы, как

Ubuntu и Mint. Установка в основных чертах производится так же, как для OS X, отличается только порядок установки EPDFree. Установщик представляет собой скрипт оболочки, запускаемый из терминала. В зависимости от разрядности системы нужно выбрать установщик типа `x86` (32-разрядный) или `x86_64` (64-разрядный). Имя соответствующего файла имеет вид `epd_free-7.3-1-rh5-x86_64.sh`. Для установки нужно выполнить такую команду:

```
$ bash epd_free-7.3-1-rh5-x86_64.sh
```

После подтверждения согласия с лицензией вам будет предложено указать место установки файлов EPDFree. Я рекомендую устанавливать их в свой домашний каталог, например `/home/wesm/epd` (вместо `wesm` подставьте свое имя пользователя).

После того как установщик отработает, добавьте в свою переменную `$PATH` подкаталог `bin` EPDFree. Если вы работаете с оболочкой bash (в Ubuntu она подразумевается по умолчанию), то нужно будет добавить такую строку в свой файл `.bashrc`:

```
export PATH=/home/wesm/epd/bin:$PATH
```

Естественно, вместо `/home/wesm/epd/` следует подставить свой установочный каталог. Затем запустите новый процесс терминала или повторно выполните файл `.bashrc` командой `source ~/.bashrc`.

Далее понадобится компилятор C, например gcc; во многие дистрибутивы Linux gcc включен, но это необязательно. В системах на базе Debian установка gcc производится командой:

```
sudo apt-get install gcc
```

Если набрать в командной строке слово `gcc`, то должно быть напечатано сообщение:

```
$ gcc
gcc: no input files
```

Теперь можно устанавливать pandas:

```
$ easy_install pandas
```

Если вы устанавливали EPDFree от имени пользователя root, то, возможно, придется добавить в начало команды слово `sudo` и ввести пароль. Проверка работоспособности производится так же, как в случае OS X.

Python 2 и Python 3

Сообщество Python в настоящее время переживает затянувшийся переход от семейства версий Python 2 к семейству Python 3. До появления Python 3.0 весь код на Python был обратно совместимым. Сообщество решило, что ради прогресса

языка необходимо внести некоторые изменения, которые лишат код этого свойства.

При написании этой книги я взял за основу Python 2.7, потому что большая часть научного сообщества Python еще не перешла на Python 3. Впрочем, если не считать немногих исключений, у вас не возникнет трудностей с исполнением приведенного в книге кода, даже если работать с Python 3.2.

Интегрированные среды разработки (IDE)

Когда меня спрашивают о том, какой средой разработки я пользуюсь, я почти всегда отвечаю: «IPython плюс текстовый редактор». Обычно я пишу программу и итеративно тестирую и отлаживаю ее по частям в IPython. Полезно также иметь возможность интерактивно экспериментировать с данными и визуально проверять, что в результате определенных манипуляций получается ожидаемый результат. Библиотеки pandas и NumPy спроектированы с учетом простоты использования в оболочке.

Но кто-то по-прежнему предпочитает работать в IDE, а не в текстовом редакторе. Интегрированные среды действительно предлагают много полезных «фенечек» типа автоматического завершения или быстрого доступа к документации по функциям и классам. Вот некоторые доступные варианты:

- Eclipse с подключаемым модулем PyDev;
- Python Tools для Visual Studio (для работающих в Windows);
- PyCharm;
- Spyder;
- Komodo IDE.

Сообщество и конференции

Помимо поиска в Интернете, существуют полезные списки рассылки, посвященные использованию Python в научных расчетах. Их участники быстро отвечают на вопросы. Вот некоторые из таких ресурсов:

- pydata: группа Google по вопросам, относящимся к использованию Python для анализа данных и pandas;
- pystatsmodels: вопросы, касающиеся statsmodels и pandas;
- numpy-discussion: вопросы, касающиеся NumPy;
- scipy-user: общие вопросы использования SciPy и Python для научных расчетов.

Я сознательно не публикую URL-адреса, потому что они часто меняются. Поиск в Интернете вам в помощь.

Ежегодно в разных странах проводят конференции для программистов на Python. PyCon и EuroPython – две самых крупных, проходящие соответственно в США и в Европе. SciPy и EuroSciPy – конференции, ориентированные на научные применения Python, где вы найдете немало «собратьев», если, прочитав эту книгу, захотите более плотно заняться анализом данных с помощью Python.

Структура книги

Если вы раньше никогда не программировали на Python, то имеет смысл начать с *конца* книги, где я поместил очень краткое руководство по синтаксису Python, языковым средствам и встроенным структурам данных: кортежам, спискам и словарям. Эти знания необходимы для чтения книги.

Книга начинается знакомством со средой IPython. Затем следует краткое введение в основные возможности NumPy, а более продвинутые рассматриваются в другой главе ближе к концу книги. Далее я знакомлю читателей с библиотекой pandas, а все остальные главы посвящены анализу данных с помощью pandas, NumPy и matplotlib (для визуализации). Я старался располагать материал последовательно, но иногда главы все же немного перекрываются.

Файлы с данными и материалы, относящиеся к каждой главе, размещаются в репозитории git на сайте GitHub:

```
http://github.com/pydata/pydata-book
```

Призываю вас скачать данные и пользоваться ими при воспроизведении примеров кода и экспериментах с описанными в книге инструментами. Я буду благодарен за любые добавления, скрипты, блокноты IPython и прочие материалы, которые вы захотите поместить в репозиторий книги для всеобщей пользы.

Примеры кода

Примеры кода в большинстве случаев показаны так, как выглядят в оболочке IPython: ввод и вывод.

```
In [5]: код
Out[5]: результат
```

Иногда для большей ясности несколько примеров кода показаны рядом. Их следует читать слева направо и исполнять по отдельности.

```
In [5]: код                 In [6]: код2
Out[5]: результат           Out[6]: результат2
```

Данные для примеров

Наборы данных для примеров из каждой главы находятся в репозитории на сайте GitHub: *http://github.com/pydata/pydata-book*. Вы можете получить их либо с помощью командной утилиты системы управления версиями git, либо скачав zip-файл репозитория с сайта.

Я стремился, чтобы в репозиторий попало все необходимое для воспроизведения примеров, но мог где-то ошибиться или что-то пропустить. В таком случае пишите мне на адрес `wesmckinn@gmail.com`.

Соглашения об импорте

В сообществе Python принят ряд соглашений об именовании наиболее употребительных модулей:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Это означает, что `np.arange` – ссылка на функцию `arange` в пакете NumPy. Так делается, потому что импорт всех имен из большого пакета, каким является NumPy (`from numpy import *`), считается среди разработчиков на Python дурным тоном.

Жаргон

Я употребляю некоторые термины, встречающиеся как в программировании, так и в науке о данных, с которыми вы, возможно, незнакомы. Поэтому приведу краткие определения.

- *Переформатирование (Munge/Munging/Wrangling)*
  Процесс приведения неструктурированных и (или) замусоренных данных к структурированной или чистой форме. Слово вошло в лексикон многих современных специалистов по анализу данных.
- *Псевдокод*
  Описание алгоритма или процесса в форме, напоминающей код, хотя фактически это не есть корректный исходный код на каком-то языке программирования.
- *Синтаксическая глазурь (syntactic sugar)*
  Синтаксическая конструкция, которая не добавляет какую-то новую функциональность, а лишь вносит дополнительное удобство или позволяет сделать код короче.

Благодарности

Я не смог бы написать эту книгу без помощи со стороны многих людей.

Из сотрудников издательства O'Reilly я крайне признателен своим редакторам Меган Бланшетт (Meghan Blanchette) и Джулии Стил (Julie Steele), которые направляли меня на протяжении всей работы. Майк Лукидес (Mike Loukides) работал со мной на этапе подготовки предложения и помог превратить замысел в реальность.

Техническое рецензирование осуществляла большая группа людей. В частности, Мартин Блез (Martin Blais) и Хью Уайт (Hugh White) оказали неоценимую помощь в подборе примеров, повышении ясности изложения и улучшении структуры книги в целом. Джеймс Лонг (James Long), Дрю Конвей (Drew Conway), Фернандо Перес (Fernando Pérez), Брайан Грейнджер (Brian Granger), Томас Клюйвер (Thomas Kluyver), Адам Клейн (Adam Klein), Джош Клейн (Josh Klein), Чань Ши (Chang She) и Стефан ван дер Вальт (Stéfan van der Walt) просмотрели по одной или по несколько глав под разными углами зрения.

Много идей по поводу примеров и наборов данных мне предложили друзья и коллеги по сообществу анализа данных, в том числе: Майк Дьюар (Mike Dewar),

Джефф Хаммербахер (Jeff Hammerbacher), Джеймс Джондроу (James Johndrow), Кристиан Лам (Kristian Lum), Адам Клейн, Хилари Мейсон (Hilary Mason), Чань Ши и Эшли Уильямс (Ashley Williams).

Разумеется, я в долгу перед многими лидерами сообщества «научного Python», которые создали открытый исходный код, легший в основу моих исследований, и оказывали мне поддержку на протяжении всей работы над книгой: группа разработки ядра IPython (Фернандо Перес, Брайан Грейнджер, Мин Рэган-Келли (Min Ragan-Kelly), Томас Клюйвер и другие), Джон Хантер (John Hunter), Скиппер Сиболд (Skipper Seabold), Трэвис Олифант (Travis Oliphant), Питер Уонг (Peter Wang), Эрик Джоунз (Eric Jones), Роберт Керн (Robert Kern), Джозеф Перктольд (Josef Perktold), Франческ Олтед (Francesc Alted), Крис Фоннесбек (Chris Fonnesbeck) и многие, многие другие, перечислять которых здесь нет никакой возможности. Меня также поддерживали, подбадривали и делились идеями Дрю Конвей, Шон Тэйлор (Sean Taylor), Джузеппе Палеолого (Giuseppe Paleologo), Джаред Ландер (Jared Lander), Дэвид Эпштейн (David Epstein), Джог Кровас (John Krowas), Джошуа Блум (Joshua Bloom), Дэн Пилсфорт (Den Pilsworth), Джон Майлз-Уайт (John Myles-White) и многие другие, имена которых я сейчас не могу вспомнить.

Я также благодарен многим, кто оказал влияние на мое становление как ученого. В первую очередь, это мои бывшие коллеги по компании AQR, которые поддерживали мою работу над pandas в течение многих лет: Алекс Рейфман (Alex Reyfman), Майкл Вонг (Michael Wong), Тим Сарджен (Tim Sargen), Октай Курбанов (Oktay Kurbanov), Мэтью Щанц (Matthew Tschantz), Рони Израэлов (Roni Israelov), Майкл Кац (Michael Katz), Крис Уга (Chris Uga), Прасад Раманан (Prasad Ramanan), Тэд Сквэр (Ted Square) и Хун Ким (Hoon Kim). И наконец, благодарю моих университетских наставников Хэйнса Миллера (МТИ) и Майка Уэста (университет Дьюк).

Если говорить о личной жизни, то я благодарен Кэйси Динкин (Casey Dinkin), чью каждодневную поддержку невозможно переоценить, ту, которая терпела перепады моего настроения, когда я пытался собрать окончательный вариант рукописи в дополнение к своему и так уже перегруженному графику. А также моим родителям, Биллу и Ким, которые учили меня никогда не отступать от мечты и не соглашаться на меньшее.

ГЛАВА 2.
Первые примеры

В этой книге рассказывается об инструментах, позволяющих продуктивно работать с данными в программах на языке Python. Хотя конкретные цели читателей могут быть различны, почти любую задачу можно отнести к одной из нескольких широких групп:

- *Взаимодействие с внешним миром*
  Чтение и запись данных, хранящихся в файлах различных форматов и в базах данных.
- *Подготовка*
  Очистка, переформатирование, комбинирование, нормализация, изменение формы, формирование продольных и поперечных срезов, преобразование данных для анализа.
- *Преобразование*
  Применение математических и статистических операций к группам наборов данных для порождения новых наборов. Например, агрегирование большой таблицы по групповым переменным.
- *Моделирование и расчет*
  Соединение данных со статистическими моделями, алгоритмами машинного обучения и другими вычислительными средствами.
- *Презентация*
  Создание интерактивных или статических графических представлений или текстовых сводок.

В этой главе я продемонстрирую несколько наборов данных и что с ними можно делать. Примеры преследуют только одну цель — возбудить у вас интерес, поэтому объяснения будут весьма общими. Не расстраивайтесь, если у вас пока нет опыта работа с описываемыми инструментами; они будут подробно рассматриваться на протяжении всей книги. В примерах кода вы встретите строки вида `In [15]:`, они взяты напрямую из оболочки IPython.

Набор данных 1.usa.gov с сайта bit.ly

В 2011 году служба сокращения URL-адресов bit.ly заключила партнерское соглашение с сайтом правительства США `usa.gov` о синхронном предоставлении

анонимных данных о пользователях, которые сокращают ссылки, заканчивающиеся на `.gov` или `.mil`. На момент написания этой книги помимо синхронной ленты, каждый час формируются мгновенные снимки, доступные в виде текстовых файлов[1].

В мгновенном снимке каждая строка представлена в формате JSON (JavaScript Object Notation), широко распространенном в веб. Например, первая строка файла выглядит примерно так:

```
In [15]: path = 'ch02/usagov_bitly_data2012-03-16-1331923249.txt'
In [16]: open(path).readline()
Out[16]: '{ "a": "Mozilla\\/5.0 (Windows NT 6.1; WOW64) AppleWebKit\\/535.11
(KHTML, like Gecko) Chrome\\/17.0.963.78 Safari\\/535.11", "c": "US", "nk": 1,
"tz": "America\\/New_York", "gr": "MA", "g": "A6qOVH", "h": "wfLQtf", "l":
"orofrog", "al": "en-US,en;q=0.8", "hh": "1.usa.gov", "r":
"http:\\/\\/www.facebook.com\\/l\\/7AQEFzjSi\\/1.usa.gov\\/wfLQtf", "u":
"http:\\/\\/www.ncbi.nlm.nih.gov\\/pubmed\\/22415991", "t": 1331923247, "hc":
1331822918, "cy": "Danvers", "ll": [ 42.576698, -70.954903 ] }\n'
```

Для Python имеется много встроенных и сторонних модулей, позволяющих преобразовать JSON-строку в объект словаря Python. Ниже я воспользовался модулем `json`; принадлежащая ему функция `loads` вызывается для каждой строки скачанного мной файла:

```
import json
path = 'ch02/usagov_bitly_data2012-03-16-1331923249.txt'
records = [json.loads(line) for line in open(path)]
```

Для тех, кто никогда не программировал на Python, скажу, что выражение в последней строке называется *списковым включением*; это краткий способ применить некую операцию (в данном случае `json.loads`) к коллекции строк или других объектов. Очень удобно – в случае, когда итерация применяется к описателю открытого файла, мы получаем последовательность прочитанных из него строк. Получившийся в результате объект `records` представляет собой список словарей Python:

```
In [18]: records[0]
Out[18]:
{u'a': u'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.11 (KHTML, like
Gecko) Chrome/17.0.963.78 Safari/535.11',
 u'al': u'en-US,en;q=0.8',
 u'c': u'US',
 u'cy': u'Danvers',
 u'g': u'A6qOVH',
 u'gr': u'MA',
 u'h': u'wfLQtf',
 u'hc': 1331822918,
 u'hh': u'1.usa.gov',
 u'l': u'orofrog',
 u'll': [42.576698, -70.954903],
```

[1] http://www.usa.gov/About/developer-resources/1usagov.shtml

```
 u'nk': 1,
 u'r': u'http://www.facebook.com/l/7AQEFzjSi/1.usa.gov/wfLQtf',
 u't': 1331923247,
 u'tz': u'America/New_York',
 u'u': u'http://www.ncbi.nlm.nih.gov/pubmed/22415991'}
```

Отметим, что в Python индексы начинаются с 0, а не с 1, как в некоторых других языках (например, R). Теперь нетрудно выделить интересующие значения из каждой записи, передав строку, содержащую ключ:

```
In [19]: records[0]['tz']
Out[19]: u'America/New_York'
```

Буква `u` перед знаком кавычки означает *unicode* – стандартную кодировку строк. Отметим, что IPython показывает *представление* объекта строки часового пояса, а не его печатный эквивалент:

```
In [20]: print records[0]['tz']
America/New_York
```

Подсчет часовых поясов на чистом Python

Допустим, что нас интересуют часовые пояса, чаще всего встречающиеся в наборе данных (поле `tz`). Решить эту задачу можно разными способами. Во-первых, можно извлечь список часовых поясов, снова воспользовавшись списковым включением:

```
In [25]: time_zones = [rec['tz'] for rec in records]
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/home/wesm/book_scripts/whetting/<ipython> in <module>()
----> 1 time_zones = [rec['tz'] for rec in records]
KeyError: 'tz'
```

Вот те раз! Оказывается, что не во всех записях есть поле часового пояса. Это легко поправить, добавив проверку `if 'tz' in rec` в конец спискового включения:

```
In [26]: time_zones = [rec['tz'] for rec in records if 'tz' in rec]
In [27]: time_zones[:10]
Out[27]:
[u'America/New_York',
 u'America/Denver',
 u'America/New_York',
 u'America/Sao_Paulo',
 u'America/New_York',
 u'America/New_York',
 u'Europe/Warsaw',
 u'',
 u'',
 u'']
```

Мы видим, что уже среди первых 10 часовых поясов встречаются неизвестные (пустые). Их можно было бы тоже отфильтровать, но я пока оставлю. Я покажу два способа подсчитать количество часовых поясов: трудный (в котором используется только стандартная библиотека Python) и легкий (с помощью pandas). Для подсчета можно завести словарь для хранения счетчиков и обойти весь список часовых поясов:

```
def get_counts(sequence):
 counts = {}
 for x in sequence:
  if x in counts:
   counts[x] += 1
  else:
   counts[x] = 1
  return counts
```

Зная стандартную библиотеку Python немного получше, можно было бы записать то же самое короче:

```
from collections import defaultdict

def get_counts2(sequence):
 counts = defaultdict(int) # values will initialize to 0
 for x in sequence:
  counts[x] += 1
 return counts
```

Чтобы можно было повторно воспользоваться этим кодом, я поместил его в функцию. Чтобы применить его к часовым поясам, достаточно передать этой функции список `time_zones`:

```
In [31]: counts = get_counts(time_zones)

In [32]: counts['America/New_York']
Out[32]: 1251

In [33]: len(time_zones)
Out[33]: 3440
```

Чтобы получить только первые 10 часовых поясов со счетчиками, придется поколдовать над словарем:

```
def top_counts(count_dict, n=10):
 value_key_pairs = [(count, tz) for tz, count in count_dict.items()]
 value_key_pairs.sort()
 return value_key_pairs[-n:]
```

В результате получим:

```
In [35]: top_counts(counts)
Out[35]:
[(33, u'America/Sao_Paulo'),
```

```
 (35, u'Europe/Madrid'),
 (36, u'Pacific/Honolulu'),
 (37, u'Asia/Tokyo'),
 (74, u'Europe/London'),
 (191, u'America/Denver'),
 (382, u'America/Los_Angeles'),
 (400, u'America/Chicago'),
 (521, u''),
 (1251, u'America/New_York')]
```

Пошарив в стандартной библиотеке Python, можно найти класс `collections.Counter`, который позволяет решить задачу гораздо проще:

```
In [49]: from collections import Counter

In [50]: counts = Counter(time_zones)

In [51]: counts.most_common(10)
Out[51]:
[(u'America/New_York', 1251),
 (u'', 521),
 (u'America/Chicago', 400),
 (u'America/Los_Angeles', 382),
 (u'America/Denver', 191),
 (u'Europe/London', 74),
 (u'Asia/Tokyo', 37),
 (u'Pacific/Honolulu', 36),
 (u'Europe/Madrid', 35),
 (u'America/Sao_Paulo', 33)]
```

Подсчет часовых поясов с помощью pandas

Основной в библиотеке pandas является структура данных *DataFrame*, которую можно представлять себе как таблицу. Создать экземпляр DataFrame из исходного набора записей просто:

```
In [289]: from pandas import DataFrame, Series

In [290]: import pandas as pd

In [291]: frame = DataFrame(records)

In [292]: frame

Out[292]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3560 entries, 0 to 3559
Data columns:
_heartbeat_   120 non-null values
a            3440 non-null values
al           3094 non-null values
c            2919 non-null values
cy           2919 non-null values
g            3440 non-null values
```

```
gr           2919 non-null values
h            3440 non-null values
hc           3440 non-null values
hh           3440 non-null values
kw             93 non-null values
l            3440 non-null values
ll           2919 non-null values
nk           3440 non-null values
r            3440 non-null values
t            3440 non-null values
tz           3440 non-null values
u            3440 non-null values
dtypes: float64(4), object(14)

In [293]: frame['tz'][:10]
Out[293]:
0     America/New_York
1       America/Denver
2     America/New_York
3    America/Sao_Paulo
4     America/New_York
5     America/New_York
6        Europe/Warsaw
7
8
9
Name: tz
```

На выходе по запросу `frame` мы видим сводное представление, которое показывается для больших объектов DataFrame. Объект Series, возвращаемый в ответ на запрос `frame['tz']`, имеет метод `value_counts`, который дает как раз то, что нам нужно:

```
In [294]: tz_counts = frame['tz'].value_counts()

In [295]: tz_counts[:10]
Out[295]:

America/New_York    1251
                     521
America/Chicago     400
America/Los_Angeles 382
America/Denver      191
Europe/London        74
Asia/Tokyo           37
Pacific/Honolulu     36
Europe/Madrid        35
America/Sao_Paulo    33
```

После этого можно с помощью библиотеки matplotlib построить график этих данных. Возможно, придется слегка подправить их, подставив какое-нибудь значение вместо неизвестных и отсутствующих часовых поясов. Заменить отсутствующие (NA) значения позволяет функция `fillna`, а неизвестные значения (пустые строки) можно заменить с помощью булевой индексации массива:

```
In [296]: clean_tz = frame['tz'].fillna('Missing')

In [297]: clean_tz[clean_tz == ''] = 'Unknown'

In [298]: tz_counts = clean_tz.value_counts()

In [299]: tz_counts[:10]
Out[299]:
America/New_York    1251
Unknown              521
America/Chicago      400
America/Los_Angeles  382
America/Denver       191
Missing              120
Europe/London         74
Asia/Tokyo            37
Pacific/Honolulu      36
Europe/Madrid         35
```

Для построения горизонтальной столбчатой диаграммы можно применить метод `plot` к объектам `counts`:

```
In [301]: tz_counts[:10].plot(kind='barh', rot=0)
```

Результат показан на рис. 2.1. Ниже мы рассмотрим и другие инструменты для работы с такими данными. Например, поле `a` содержит информацию о браузере, устройстве или приложении, выполнившем сокращение URL:

```
In [302]: frame['a'][1]
Out[302]: u'GoogleMaps/RochesterNY'

In [303]: frame['a'][50]
Out[303]: u'Mozilla/5.0 (Windows NT 5.1; rv:10.0.2) Gecko/20100101 Firefox/10.0.2'

In [304]: frame['a'][51]
Out[304]: u'Mozilla/5.0 (Linux; U; Android 2.2.2; en-us; LG-P925/V10e Build/
FRG83G) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533
```

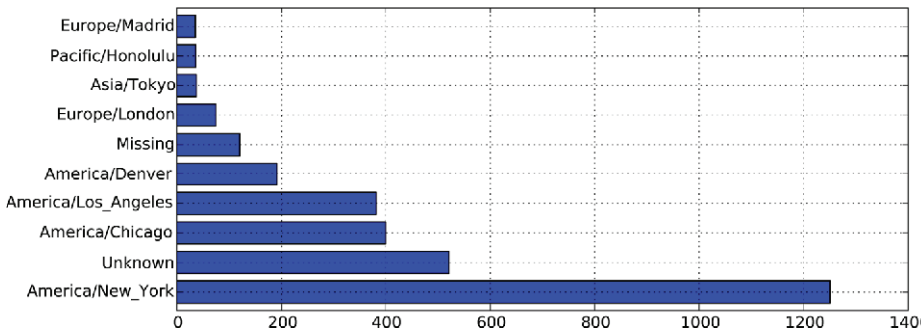


Рис. 2.1. Первые 10 часовых поясов из набора данных 1.usa.gov

Выделение всей интересной информации из таких строк «пользовательских агентов» поначалу может показаться пугающей задачей. По счастью, на деле все не так плохо – нужно только освоить встроенные в Python средства для работы со строками и регулярными выражениями. Например, вот как можно вырезать из строки первую лексему (грубо описывающую возможности браузера) и представить поведение пользователя в другом разрезе:

```
In [305]: results = Series([x.split()[0] for x in frame.a.dropna()])

In [306]: results[:5]
Out[306]:
0    Mozilla/5.0
1    GoogleMaps/RochesterNY
2    Mozilla/4.0
3    Mozilla/5.0
4    Mozilla/5.0

In [307]: results.value_counts()[:8]
Out[307]:
Mozilla/5.0                 2594
Mozilla/4.0                  601
GoogleMaps/RochesterNY       121
Opera/9.80                    34
TEST_INTERNET_AGENT           24
GoogleProducer                21
Mozilla/6.0                    5
BlackBerry8520/5.0.0.681       4
```

Предположим теперь, что требуется разделить пользователей в первых 10 часовых поясах на работающих в Windows и всех прочих. Упростим задачу, предположив, что пользователь работает в Windows, если строка агента содержит подстроку `'Windows'`. Но строка агента не всегда присутствует, поэтому записи, в которых ее нет, я исключу:

```
In [308]: cframe = frame[frame.a.notnull()]
```

Мы хотим вычислить значение, показывающее, относится строка к пользователю Windows или нет:

```
In [309]: operating_system = np.where(cframe['a'].str.contains('Windows'),
  .....:                              'Windows', 'Not Windows')

In [310]: operating_system[:5]
Out[310]:
0        Windows
1    Not Windows
2        Windows
3    Not Windows
4        Windows
Name: a
```

Затем мы можем сгруппировать данные по часовому поясу и только что сформированному столбцу с типом операционной системы:

```
In [311]: by_tz_os = cframe.groupby(['tz', operating_system])
```

Групповые счетчики по аналогии с рассмотренной выше функцией `value_counts` можно вычислить с помощью функции `size`. А затем преобразовать результат в таблицу с помощью `unstack`:

```
In [312]: agg_counts = by_tz_os.size().unstack().fillna(0)

In [313]: agg_counts[:10]
Out[313]:
a                      Not Windows    Windows
tz
                                245        276
Africa/Cairo                      0          3
Africa/Casablanca                 0          1
Africa/Ceuta                      0          2
Africa/Johannesburg               0          1
Africa/Lusaka                     0          1
America/Anchorage                 4          1
America/Argentina/Buenos_Aires    1          0
America/Argentina/Cordoba         0          1
America/Argentina/Mendoza         0          1
```

Наконец, выберем из полученной таблицы первые 10 часовых поясов. Для этого я построю массив косвенных индексов `agg_counts` по счетчикам строк:

```
# Нужен для сортировки в порядке возрастания
In [314]: indexer = agg_counts.sum(1).argsort()

In [315]: indexer[:10]
Out[315]:
tz
                                  24
Africa/Cairo                      20
Africa/Casablanca                 21
Africa/Ceuta                      92
Africa/Johannesburg               87
Africa/Lusaka                     53
America/Anchorage                 54
America/Argentina/Buenos_Aires    57
America/Argentina/Cordoba         26
America/Argentina/Mendoza         55
```

А затем с помощью `take` расположу строки в порядке, определяемом этим индексом, и оставлю только последние 10:

```
In [316]: count_subset = agg_counts.take(indexer)[-10:]

In [317]: count_subset
Out[317]:
```

```
a                   Not Windows       Windows
tz
America/Sao_Paulo             13            20
Europe/Madrid                 16            19
Pacific/Honolulu               0            36
Asia/Tokyo                     2            35
Europe/London                 43            31
America/Denver               132            59
America/Los_Angeles          130           252
America/Chicago              115           285
                             245           276
America/New_York             339           912
```

Теперь можно построить столбчатую диаграмму, как и в предыдущем примере. Только на этот раз я сделаю ее штабельной, передав параметр `stacked=True` (см. рис. 2.2):

```
In [319]: count_subset.plot(kind='barh', stacked=True)
```

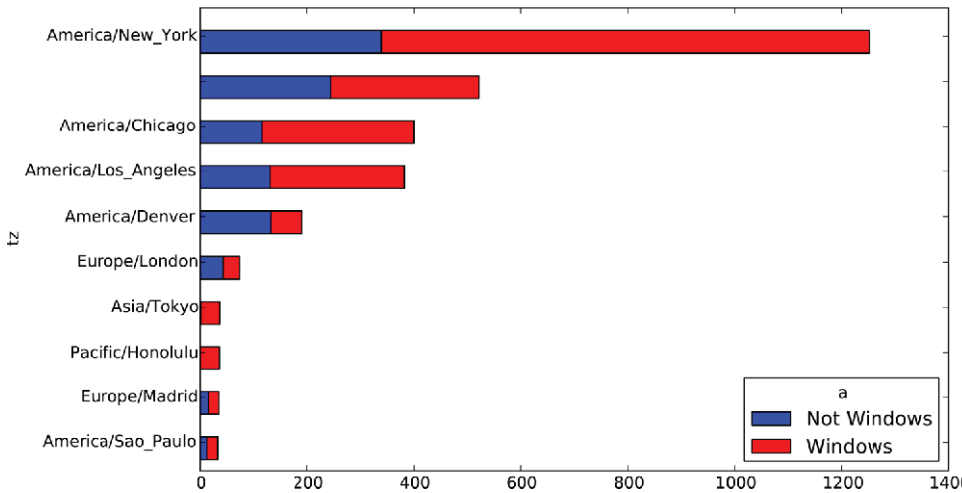


Рис. 2.2. Первые 10 часовых поясов с выделением пользователей Windows и прочих

Из этой диаграммы трудно наглядно представить, какова процентная доля пользователей Windows в каждой группе, но строки легко можно нормировать, так чтобы в сумме получилась 1, а затем построить диаграмму еще раз (рис. 2.3):

```
In [321]: normed_subset = count_subset.div(count_subset.sum(1), axis=0)
In [322]: normed_subset.plot(kind='barh', stacked=True)
```

Все использованные нами методы будут подробно рассмотрены в последующих главах.

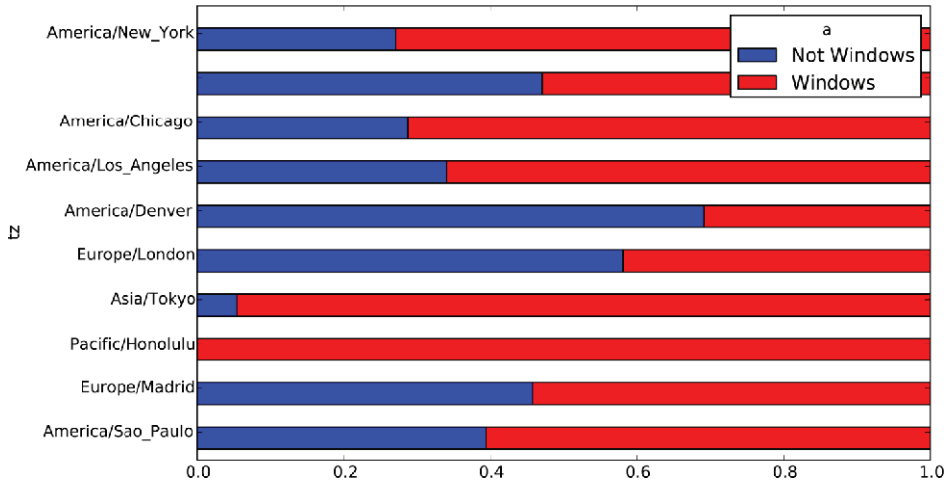


Рис. 2.3. Процентная доля пользователей Windows и прочих в первых 10 часовых поясах

Набор данных MovieLens 1M

Исследовательская группа GroupLens Research (*http://www.grouplens.org/node/73*) предлагает несколько наборов данных о рейтингах фильмов, проставленных пользователями сайта MovieLens в конце 1990-х – начале 2000-х. Наборы содержат рейтинги фильмов, метаданные о фильмах (жанр и год выхода) и демографические данные о пользователях (возраст, почтовый индекс, пол и род занятий). Такие данные часто представляют интерес для разработки систем рекомендования, основанных на алгоритмах машинного обучения. И хотя в этой книге методы машинного обучения не рассматриваются, я все же покажу, как формировать продольные и поперечные срезы таких наборов данных с целью привести их к нужному виду.

Набор MovieLens 1M содержит 1 миллион рейтингов 4000 фильмов, проставленных 6000 пользователей. Данные распределены по трем таблицам: рейтинги, информация о пользователях и информация о фильмах. После распаковки zip-файла каждую таблицу можно загрузить в отдельный объект DataFrame с помощью метода `pandas.read_table`:

```
import pandas as pd

unames = ['user_id', 'gender', 'age', 'occupation', 'zip']
users = pd.read_table('ml-1m/users.dat', sep='::', header=None,
                      names=unames)

rnames = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_table('ml-1m/ratings.dat', sep='::', header=None,
                        names=rnames)

mnames = ['movie_id', 'title', 'genres']
```

```
movies = pd.read_table('ml-1m/movies.dat', sep='::', header=None,
                       names=mnames)
```

Проверить, все ли прошло удачно, можно, посмотрев на первые несколько строк каждого DataFrame с помощью встроенного в Python синтаксиса вырезания:

```
In [334]: users[:5]
Out[334]:
   user_id gender  age  occupation    zip
0        1      F    1          10  48067
1        2      M   56          16  70072
2        3      M   25          15  55117
3        4      M   45           7  02460
4        5      M   25          20  55455

In [335]: ratings[:5]
Out[335]:
   user_id  movie_id  rating  timestamp
0        1      1193       5  978300760
1        1       661       3  978302109
2        1       914       3  978301968
3        1      3408       4  978300275
4        1      2355       5  978824291

In [336]: movies[:5]
Out[336]:
  movie_id                               title                        genres
0        1                    Toy Story (1995)   Animation|Children's|Comedy
1        2                      Jumanji (1995)  Adventure|Children's|Fantasy
2        3             Grumpier Old Men (1995)                Comedy|Romance
3        4            Waiting to Exhale (1995)                  Comedy|Drama
4        5  Father of the Bride Part II (1995)                        Comedy

In [337]: ratings
Out[337]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns:
user_id    1000209   non-null values
movie_id   1000209   non-null values
rating     1000209   non-null values
timestamp  1000209   non-null values
dtypes: int64(4)
```

Отметим, что возраст и род занятий кодируются целыми числами, а расшифровка приведена в прилагаемом к набору данных файлу `README`. Анализ данных, хранящихся в трех таблицах, – непростая задача. Пусть, например, требуется вычислить средние рейтинги для конкретного фильма в разрезе пола и возраста. Как мы увидим, это гораздо легче сделать, если предварительно объединить все данные в одну таблицу. Применяя функцию `merge` из библиотеки pandas, мы сначала объединим `ratings` с `users`, а затем результат объединим с `movies`. Pandas определяем, по каким столбцам объединять (или *соединять*), ориентируясь на совпадение имен:

```
In [338]: data = pd.merge(pd.merge(ratings, users), movies)
In [339]: data

Out[339]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns:
user_id    1000209  non-null values
movie_id   1000209  non-null values
rating     1000209  non-null values
timestamp  1000209  non-null values
gender     1000209  non-null values
age        1000209  non-null values
occupation 1000209  non-null values
zip        1000209  non-null values
title      1000209  non-null values
genres     1000209  non-null values
dtypes: int64(6), object(4)

In [340]: data.ix[0]
Out[340]:
user_id              1
movie_id             1
rating               5
timestamp    978824268
gender               F
age                  1
occupation          10
zip              48067
title       Toy Story (1995)
genres      Animation|Children's|Comedy
Name: 0
```

В таком виде агрегирование рейтингов, сгруппированных по одному или нескольким атрибутам пользователя или фильма, для человека, хоть немного знакомого с pandas, не представляет никаких трудностей. Чтобы получить средние рейтинги фильмов при группировке по полу, воспользуемся методом `pivot_table`:

```
In [341]: mean_ratings = data.pivot_table('rating', rows='title',
   .....:                                  cols='gender', aggfunc='mean')

In [342]: mean_ratings[:5]
Out[342]:
gender                                F         M
title
$1,000,000 Duck (1971)         3.375000  2.761905
'Night Mother (1986)           3.388889  3.352941
'Til There Was You (1997)      2.675676  2.733333
'burbs, The (1989)             2.793478  2.962085
...And Justice for All (1979)  3.828571  3.689024
```

В результате получается еще один объект DataFrame, содержащий средние рейтинги, в котором метками строк являются общее количество оценок фильма, а

метками столбцов – обозначения полов. Сначала я оставлю только фильмы, получившие не менее 250 оценок (число выбрано совершенно произвольно); для этого сгруппирую данные по названию и с помощью метода `size()` получу объект Series, содержащий размеры групп для каждого наименования:

```
In [343]: ratings_by_title = data.groupby('title').size()

In [344]: ratings_by_title[:10]
Out[344]:
title
$1,000,000 Duck (1971)              37
'Night Mother (1986)                70
'Til There Was You (1997)           52
'burbs, The (1989)                 303
...And Justice for All (1979)      199
1-900 (1994)                         2
10 Things I Hate About You (1999)  700
101 Dalmatians (1961)              565
101 Dalmatians (1996)              364
12 Angry Men (1957)                616

In [345]: active_titles = ratings_by_title.index[ratings_by_title >= 250]

In [346]: active_titles
Out[346]:
Index(['burbs, The (1989), 10 Things I Hate About You (1999),
       101 Dalmatians (1961), ..., Young Sherlock Holmes (1985),
       Zero Effect (1998), eXistenZ (1999)], dtype=object)
```

Затем для отбора строк из приведенного выше объекта `mean_ratings` воспользуемся индексом фильмов, получивших не менее 250 оценок:

```
In [347]: mean_ratings = mean_ratings.ix[active_titles]

In [348]: mean_ratings
Out[348]:
<class 'pandas.core.frame.DataFrame'>
Index: 1216 entries, 'burbs, The (1989) to eXistenZ (1999)
Data columns:
F     1216  non-null values
M     1216  non-null values
dtypes: float64(2)
```

Чтобы найти фильмы, оказавшиеся на первом месте у зрителей-женщин, мы можем отсортировать результат по столбцу `F` в порядке убывания:

```
In [350]: top_female_ratings = mean_ratings.sort_index(by='F', ascending=False)

In [351]: top_female_ratings[:10]
Out[351]:
gender                                                F         M
Close Shave, A (1995)                          4.644444  4.473795
Wrong Trousers, The (1993)                     4.588235  4.478261
```

```
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)            4.572650  4.464589
Wallace & Gromit: The Best of Aardman Animation (1996)   4.563107  4.385075
Schindler's List (1993)                                  4.562602  4.491415
Shawshank Redemption, The (1994)                         4.539075  4.560625
Grand Day Out, A (1992)                                  4.537879  4.293255
To Kill a Mockingbird (1962)                             4.536667  4.372611
Creature Comforts (1990)                                 4.513889  4.272277
Usual Suspects, The (1995)                               4.513317  4.518248
```

Измерение несогласия в оценках

Допустим, мы хотим найти фильмы, по которым мужчины и женщины сильнее всего разошлись в оценках. Для этого можно добавить столбец `mean_ratings`, содержащий разность средних, а затем отсортировать по нему:

```
In [352]: mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']
```

Сортировка по столбцу `'diff'` дает фильмы с наибольшей разностью оценок, которые больше нравятся женщинам:

```
In [353]: sorted_by_diff = mean_ratings.sort_index(by='diff')

In [354]: sorted_by_diff[:15]
Out[354]:
gender                                     F         M       diff
Dirty Dancing (1987)                3.790378  2.959596  -0.830782
Jumpin' Jack Flash (1986)           3.254717  2.578358  -0.676359
Grease (1978)                       3.975265  3.367041  -0.608224
Little Women (1994)                 3.870588  3.321739  -0.548849
Steel Magnolias (1989)              3.901734  3.365957  -0.535777
Anastasia (1997)                    3.800000  3.281609  -0.518391
Rocky Horror Picture Show, The (1975)  3.673016  3.160131  -0.512885
Color Purple, The (1985)            4.158192  3.659341  -0.498851
Age of Innocence, The (1993)        3.827068  3.339506  -0.487561
Free Willy (1993)                   2.921348  2.438776  -0.482573
French Kiss (1995)                  3.535714  3.056962  -0.478752
Little Shop of Horrors, The (1960)  3.650000  3.179688  -0.470312
Guys and Dolls (1955)               4.051724  3.583333  -0.468391
Mary Poppins (1964)                 4.197740  3.730594  -0.467147
Patch Adams (1998)                  3.473282  3.008746  -0.464536
```

Изменив порядок строк на противоположный и снова отобрав первые 15 строк, мы получим фильмы, которым мужчины поставили высокие, а женщины – низкие оценки:

```
# Изменяем порядок строк на противоположный и отбираем первые 15 строк
In [355]: sorted_by_diff[::-1][:15]
Out[355]:

gender                                     F         M      diff
Good, The Bad and The Ugly, The (1966) 3.494949  4.221300  0.726351
Kentucky Fried Movie, The (1977)       2.878788  3.555147  0.676359
Dumb & Dumber (1994)                   2.697987  3.336595  0.638608
```

```
Longest Day, The (1962)                3.411765  4.031447  0.619682
Cable Guy, The (1996)                  2.250000  2.863787  0.613787
Evil Dead II (Dead By Dawn) (1987)     3.297297  3.909283  0.611985
Hidden, The (1987)                     3.137931  3.745098  0.607167
Rocky III (1982)                       2.361702  2.943503  0.581801
Caddyshack (1980)                      3.396135  3.969737  0.573602
For a Few Dollars More (1965)          3.409091  3.953795  0.544704
Porky's (1981)                         2.296875  2.836364  0.539489
Animal House (1978)                    3.628906  4.167192  0.538286
Exorcist, The (1973)                   3.537634  4.067239  0.529605
Fright Night (1985)                    2.973684  3.500000  0.526316
Barb Wire (1996)                       1.585366  2.100386  0.515020
```

А теперь допустим, что нас интересуют фильмы, вызвавшие наибольшее разногласие у зрителей независимо от пола. Разногласие можно изменить с помощью дисперсии или стандартного отклонения оценок:

```
# Стандартное отклонение оценок, сгруппированных по названию
In [356]: rating_std_by_title = data.groupby('title')['rating'].std()

# Оставляем только active_titles
In [357]: rating_std_by_title = rating_std_by_title.ix[active_titles]

# Упорядочиваем Series по значению в порядке убывания
In [358]: rating_std_by_title.order(ascending=False)[:10]
Out[358]:
title
Dumb & Dumber (1994)                   1.321333
Blair Witch Project, The (1999)        1.316368
Natural Born Killers (1994)            1.307198
Tank Girl (1995)                       1.277695
Rocky Horror Picture Show, The (1975)  1.260177
Eyes Wide Shut (1999)                  1.259624
Evita (1996) 1.253631
Billy Madison (1995)                   1.249970
Fear and Loathing in Las Vegas (1998)  1.246408
Bicentennial Man (1999)                1.245533
Name: rating
```

Вы, наверное, обратили внимание, что жанры фильма разделяются вертикальной чертой (|). Чтобы провести анализ по жанрам, прошлось бы проделать дополнительную работу по преобразованию данных в более удобную форму. Ниже я еще вернусь к этому набору данных и покажу, как это сделать.

Имена, которые давали детям в США за период с 1880 по 2010 год

Управление социального обеспечения США выложило в сеть данные о частоте встречаемости детских имен за период с 1880 года по настоящее время. Хэдли Уикхэм (Hadley Wickham), автор нескольких популярных пакетов для R, часто использует этот пример для иллюстрации манипуляций с данными в R.

```
In [4]: names.head(10)
Out[4]:
          name  sex  births  year
0        Mary    F    7065  1880
1        Anna    F    2604  1880
2        Emma    F    2003  1880
3   Elizabeth    F    1939  1880
4      Minnie    F    1746  1880
5    Margaret    F    1578  1880
6         Ida    F    1472  1880
7       Alice    F    1414  1880
8      Bertha    F    1320  1880
9       Sarah    F    1288  1880
```

С этим набором можно проделать много интересного.

- Наглядно представить долю младенцев, получавших данное имя (совпадающее с вашим или какое-нибудь другое) за весь период времени.
- Определить относительный ранг имени.
- Найти самые популярные в каждом году имена или имена, для которых фиксировалось наибольшее увеличение или уменьшение частоты.
- Проанализировать тенденции выбора имен: количество гласных и согласных, длину, общее разнообразие, изменение в написании, первые и последние буквы.
- Проанализировать внешние источники тенденций: библейские имена, имена знаменитостей, демографические изменения.

С помощью уже рассмотренных инструментов большая часть этих задач решается очень просто, и я это кратко продемонстрирую. Призываю вас скачать и исследовать этот набор данных самостоятельно. Если вы обнаружите интересную закономерность, буду рад узнать про нее.

На момент написания этой книги Управление социального обеспечения США представило данные в виде набора файлов, по одному на каждый год, в которых указано общее число родившихся младенцев для каждой пары пол/имя. Архив этих файлов находится по адресу

```
http://www.ssa.gov/oact/babynames/limits.html
```

Если со временем адрес этой страницы поменяется, найти ее, скорее всего, можно будет с помощью поисковой системы. Загрузив и распаковав файл `names.zip`, вы получите каталог, содержащий файлы с именами вида `yob1880.txt`. С помощью команды UNIX `head` я могу вывести первые 10 строк каждого файла (в Windows можно воспользоваться командой `more` или открыть файл в текстовом редакторе):

```
In [367]: !head -n 10 names/yob1880.txt
Mary,F,7065
Anna,F,2604
Emma,F,2003
Elizabeth,F,1939
Minnie,F,1746
Margaret,F,1578
```

```
Ida,F,1472
Alice,F,1414
Bertha,F,1320
Sarah,F,1288
```

Поскольку поля разделены запятыми, файл можно загрузить в объект DataFrame методом `pandas.read_csv`:

```
In [368]: import pandas as pd

In [369]: names1880 = pd.read_csv('names/yob1880.txt', names=['name', 'sex', 'births'])

In [370]: names1880
Out[370]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2000 entries, 0 to 1999
Data columns:
name       2000  non-null values
sex        2000  non-null values
births     2000  non-null values
dtypes: int64(1), object(2)
```

В эти файлы включены только имена, которыми были названы не менее 5 младенцев в году, поэтому для простоты сумму значений в столбце sex можно считать общим числом родившихся в данном году младенцев:

```
In [371]: names1880.groupby('sex').births.sum()
Out[371]:
sex
F        90993
M       110493
Name: births
```

Поскольку в каждом файле находятся данные только за один год, то первое, что нужно сделать, – собрать все данные в единый объект DataFrame и добавить поле `year`. Это легко сделать методом `pandas.concat`:

```
# На данный момент 2010 - последний доступный год
years = range(1880, 2011)

pieces = []
columns = ['name', 'sex', 'births']

for year in years:
 path = 'names/yob%d.txt' % year
 frame = pd.read_csv(path, names=columns)

 frame['year'] = year
 pieces.append(frame)

 # Собрать все данные в один объект DataFrame
 names = pd.concat(pieces, ignore_index=True)
```

Обратим внимание на два момента. Во-первых, напомним, что `concat` по умолчанию объединяет объекты DataFrame построчно. Во-вторых, следует задать параметр `ignore_index=True`, потому что нам неинтересно сохранять исходные номера строк, прочитанных методом `read_csv`. Таким образом, мы получили очень большой DataFrame, содержащий данные обо всех именах.

Выглядит объект `names` следующим образом:

```
In [373]: names
Out[373]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1690784 entries, 0 to 1690783
Data columns:
name      1690784  non-null values
sex       1690784  non-null values
births    1690784  non-null values
year      1690784  non-null values
dtypes: int64(2), object(2)
```

Имея эти данные, мы уже можем приступить к агрегированию на уровне года и пола, используя метод `groupby` или `pivot_table` (см. рис. 2.4):

```
In [374]: total_births = names.pivot_table('births', rows='year',
   .....:                                  cols='sex', aggfunc=sum)

In [375]: total_births.tail()
Out[375]:
sex         F        M
year
2006  1896468  2050234
2007  1916888  2069242
2008  1883645  2032310
2009  1827643  1973359
2010  1759010  1898382
In [376]: total_births.plot(title='Total births by sex and year')
```

Далее вставим столбец `prop`, содержащий долю младенцев, получивших данное имя, относительно общего числа родившихся. Значение `prop`, равное `0.02`, означает, что данное имя получили 2 из 100 младенцев. Затем сгруппируем данные по году и полу и добавим в каждую группу новый столбец:

```
def add_prop(group):
 # При целочисленном делении производится округление с недостатком
 births = group.births.astype(float)

 group['prop'] = births / births.sum()
 return group
names = names.groupby(['year', 'sex']).apply(add_prop)
```

Напомним, что поскольку тип поля `births` – целое, для вычисления дробного числа необходимо привести числитель или знаменатель к типу с плавающей точкой (если только вы не работаете с Python 3!).

Рис. 2.4. Общее количество родившихся по полу и году

Получившийся в результате пополненный набор данных состоит из таких столбцов:

```
In [378]: names
Out[378]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1690784 entries, 0 to 1690783
Data columns:
name    1690784  non-null values
sex     1690784  non-null values
births  1690784  non-null values
year    1690784  non-null values
prop    1690784  non-null values
dtypes: float64(1), int64(2), object(2)
```

При выполнении такой операции группировки часто бывает полезно произвести проверку разумности результата, например, удостовериться, что сумма значений в столбце `prop` по всем группам равна 1. Поскольку это данные с плавающей точкой, воспользуемся методом `np.allclose`, который проверяет, что сумма по группам достаточно близка к 1 (хотя может и не быть равна в точности).

```
In [379]: np.allclose(names.groupby(['year', 'sex']).prop.sum(), 1)
Out[379]: True
```

Далее я извлеку подмножество данных, чтобы упростить последующий анализ: первые 1000 имен для каждой комбинации пола и года. Это еще одна групповая операция:

```
def get_top1000(group):
 return group.sort_index(by='births', ascending=False)[:1000]
grouped = names.groupby(['year', 'sex'])
top1000 = grouped.apply(get_top1000)
```

Если вы предпочитаете все делать самостоятельно, то можно поступить и так:

```
pieces = []
for year, group in names.groupby(['year', 'sex']):
  pieces.append(group.sort_index(by='births', ascending=False)[:1000])
top1000 = pd.concat(pieces, ignore_index=True)
```

Теперь результирующий набор стал заметно меньше:

```
In [382]: top1000
Out[382]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 261877 entries, 0 to 261876
Data columns:
name    261877  non-null values
sex     261877  non-null values
births  261877  non-null values
year    261877  non-null values
prop    261877  non-null values
dtypes: float64(1), int64(2), object(2)
```

Этот набор, содержащий первые 1000 записей, мы и будем использовать для исследования данных в дальнейшем.

Анализ тенденций в выборе имен

Имея полный набор данных и первые 1000 записей, мы можем приступить к анализу различных интересных тенденций. Для начала решим простую задачу: разобьем набор Top 1000 на части, относящиеся к мальчикам и девочкам.

```
In [383]: boys = top1000[top1000.sex == 'M']

In [384]: girls = top1000[top1000.sex == 'F']
```

Можно нанести на график простые временные ряды, например количество Джонов и Мэри в каждом году, но для этого потребуется предварительное переформатирование. Сформируем сводную таблицу, в которой представлено общее число родившихся по годам и по именам:

```
In [385]: total_births = top1000.pivot_table('births', rows='year', cols='name',
    .....:                                    aggfunc=sum)
```

Теперь можно нанести на график несколько имен, воспользовавшись методом `plot` объекта DataFrame:

```
In [386]: total_births
Out[386]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 131 entries, 1880 to 2010
Columns: 6865 entries, Aaden to Zuri
dtypes: float64(6865)

In [387]: subset = total_births[['John', 'Harry', 'Mary', 'Marilyn']]
```