**ГЛABA2.**

## Первые примеры

## В этой книге рассказывается об инструментах, позволяющих продуктивно рабо­ тать 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 .миро.м

Чтение и запись данных, хранящихся в файлах различных форматов и в базах данных.

* Подготовка

Очистка, переформатирование, комбинирование, нормализация, измене­ ние формы, формирование продольных и поперечных срезов, преобразова­ ние данных для анализа.

* Преобразование

Применение математических и статистических операций к группам на­ боров данных для порождения новых наборов. Например, агрегирование большой таблицы по групповым переменным.

* Моделирование и расчет

Соединение данных со статистическими моделями, алгоритмами машин­ ного обучения и другими вычислительными средствами.

* Презентация

Создание интерактивных или статических графических представлений или текстовых сводок.

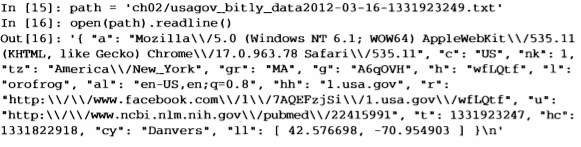
В этой главе я продемонстрирую несколько наборов данных и что с ними можно делать. Примеры преследуют только одну цель - возбудить у вас интерес, поэтому объяснения будут весьма общими. Не расстраивайтесь, если у вас пока нет опыта работа с описываемыми инструментами; они будут подробно рассматриваться 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-aдpecoв bit.ly заключила партнерское со­ глашение с сайтом правительства США usa.gov о синхронном предоставлении

анонимных данных о пользователях, которые сокращают ссылки, заканчивающиеся на .gov или .mi 1. На момент написания этой книги помимо синхронной ленты, каждый час формируются мгновенные снимки, доступные в виде текстовых файлов1.

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

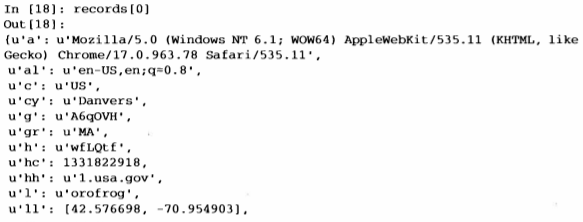


Для Python имеется много встроенных и сторонних модулей, позволяющих преобразовать JSON-cтpoкy в объект словаря Python. Ниже я воспользовался мо­ дулем json; принадлежащая ему функция loads вызывается для каждой строки скачанного мной файла:

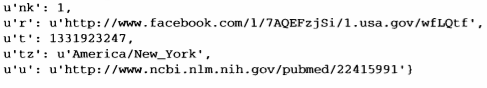
*import json*

*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:



**Глава 2. Первые примеры**



Отметим, что в Python индексы начинаются с О, 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[О] ['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[O]['tz'] America/New\_York*

***Подсчет часовых поясов на чистом Python***

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

*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',*

*и' ' '*

*и' , '*

*и"]*

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 х in sequence:*

*if х in counts:*

*counts[x] += 1 else:*

*counts[x] = 1 return counts*

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

*from collections import defaultdict def get\_counts2(sequence):*

*counts = defaultdict(int) # values will initialize to О*

*for х 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 t ime\_zones:

*In (31]: counts = get\_counts(time\_zones) In [32]: counts['America/Neи\_York']*

*Out[32]: 1251*

*In (33]: len(time\_zones)*

*Out[33]: 34 4 0*

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

*def top\_counts(count\_dict, n=lO):*

*value\_key\_pairs = ((count, tz) for tz, count in count\_dict.items()] value\_key\_pairs.sort()*

*return value\_key\_pairs[-n:J*

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{lO) Out[Sl]:*

*[(u'America/New\_York', 1251),*

*(и'', 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, О to 3559 Data columns:*

*-heartbeat 120 non-null values*

*а 3440 non-null values*

*al 3094 non-null values*

*с 2919 non-null values*

*су 2919 non-null values*

1. *3440 non-null valu*

*gr 2919 non-null values*

1. *3440 non-null values*

*hc 3440 non-null values*

*hh 3440 non-null values*

*kw 93 non-null values*

*l 3440 non- null values*

1. *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]: frarne[ 'tz'][: 10]*

*Out [293]:*

о

*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 пока­ зывается для больших объектов DataFraшe. Объект Series, возвращаемый в ответ на запрос frarne [ tz ], имеет метод value\_counts, кoтopый дает как раз то, что нам нужно:

*In [294]: tz\_counts = frarne['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*

35

*Europe/Madrid Arnerica/Sao Paulo*

После этого можно с помощью библиотеки matplotlib построить график этих данных. Возможно, придется слегка подправить их, подставив какое-нибудь зна­ чение вместо неизвестных и отсутствующих часовых поясов. Заменить отсутству­ ющие (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.valt1e\_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 а содержит информацию о браузере, устройстве или приложении, выполнившем сокращение URL:

*In [302]: frame[ ‘а’] [1)*

*Out[302]: u'GoogleMaps/RochesterNY'*

*In [303]: frame[‘а' J [50)*

*Out[303]: u'Mozilla/5.0 (Windows NТ 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*

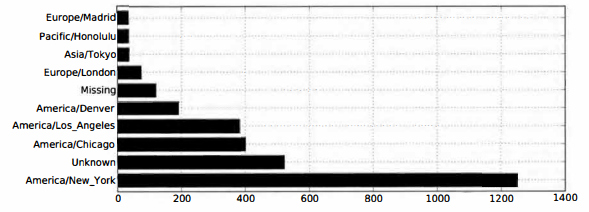


Рис. 2.1. Первые 10 часовых поясов из набора данных 1.usa.go

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 предста­ вить поведение пользователя в другом разрезе:

*In [305]: results = Series([x.split() (О] for х in frame.a.dropna()]) In [306]: results[:5]*

*Out[306]:*

*О Mozilla/5.О*

*l GoogleMaps/RochesterNY*

1. *Mozilla/4.0*
2. *Mozilla/5.О*

*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 пользовате­ лю Windows или нет:

*In (309]: operating\_systern np.where(cframe['a'] .str.contains('Windows'),*

*'Windows', 'Not Windows')*

*In (310]: operating\_systern[:5]*

*Out (310] :*

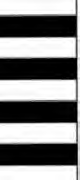
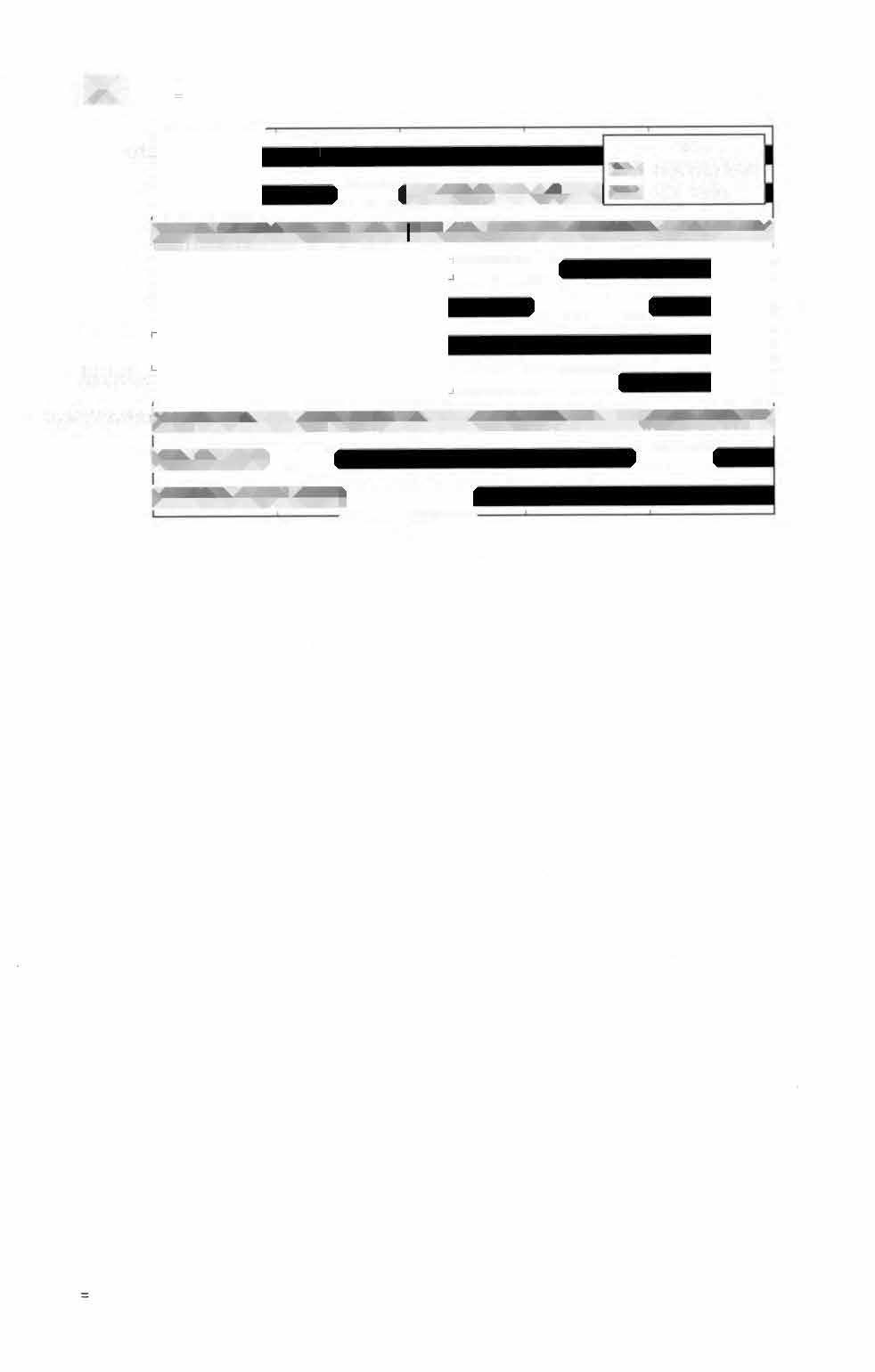
о

*Windows*

1. *Not Windows*
2. *Windows*
3. *Not Windows*
4. *Windows*

*Narne: а*

**Глава 2. Первые примеры**



Затем мы можем сгруппировать данные по часовому поясу и только что сформировавшемуся 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 ре­ зультат в таблицу с помощью unstack:

*In (312]: agg\_counts = by\_tz\_os.size() .unstack() .fillna(0) In [313]: agg\_counts[:10]*

*Out[313]:*

*а tz*

*Africa/Cairo Africa/Casablanca Africa/Ceuta Africa/Johannesburg*

*Africa/Lusaka America/Anchorage*

*Not Windows*

*2450*

*о о о о*

*4*

*Windows*

*276*

*3*

*1*

*2*

*1*

*1*

*1*

*America/Argentina/Buenos\_Aires 1 о*

о

*America/Argentina/Cordoba 1*

о

*America/Argentina/Mendoza 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(l) .argsort()*

*In [315): indexer(:10] Out[315):*

*tz*

*24*

*Africa/Cairo 20*

*Africa/CasaЬlanca 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 ин­ дексом, и оставлю только последние 1 О:

*In (316]: count\_subset = agg\_counts.take(indexer) (-10:) In (317): count subset*

*Out[317]*

*Not Windows Windows*

*America/SanPaulo*

*13 20*

*Europe/Madrid 16 19*

о

*Pacific/Honolulu 36*

*Asia/Tokyo 2 35*

*Europe/London 43 31*

*America/Denver 132 59*

*America/Los\_Angeles 130 252*

*America/Chicago 115 285*

*245 276*

339

*America/New\_York 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):

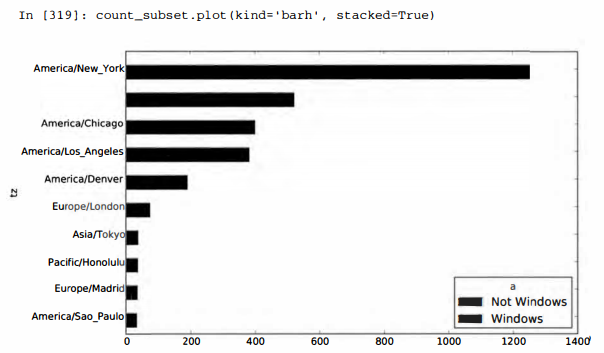
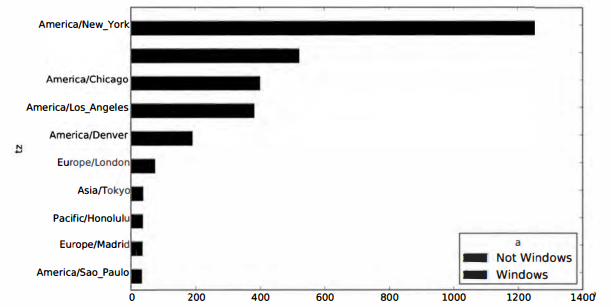


Рис. 2.2. Первые 10 часовых поясов с выделением пользователей Windows и прочих

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(l), axis=0) In [322): normed\_subset.plot(kind='barh', stacked=True)*

Все использованные нами методы будут подробно рассмотрены в последующих главах.

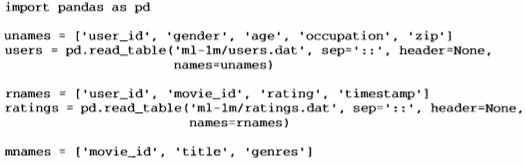


**Рис. 2.2.** Первые 10 часовых поясов с выделением пользователей Windows и прочих

###### Набор данных MovieLens 1 М

Исследовательская группа GroupLens Research (*http.j/www.gouplens.org/node/73)* предлагает несколько наборов данных о рейтингах фильмов, проставленных поль­ зователями сайта MovieLeнs 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 ма­ шинного обучения не рассматриваются, я все же покажу, как формировать про­ дольные и поперечные срезы таких наборов данных с целью привести их к нуж­ ному виду.

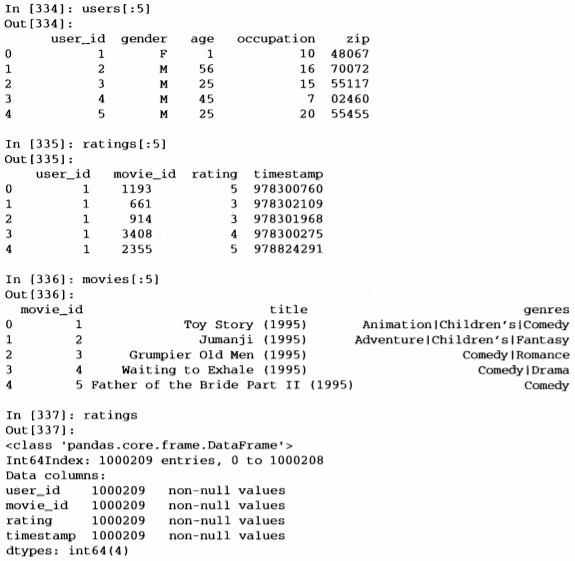
Набор MovieLens 1 М содержит 1 миллион рейтингов 4000 фильмов, простав­ ленных 6000 пользователей. Данные распределены по трем таблицам: рейтинги, информация о пользователях и информация о фильмах. После распаковки ziр­ файла каждую таблицу можно загрузить в отдельный объект DataFrame с помо­ щью метода pandas. read\_table:



**Набор данных Movielens 1 М**

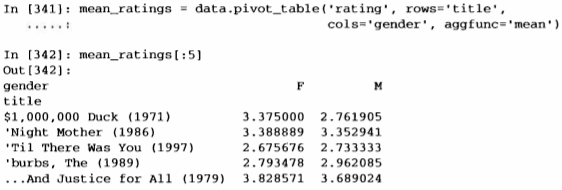


Проверить, все ли прошло удачно, можно, посмотрев на первые несколько строк каждого DataFrame с помощью встроенного в Python синтаксиса вырезания:

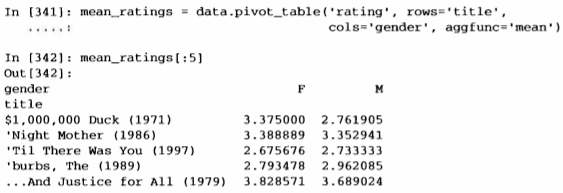


Отметим, что возраст и род занятий кодируются целыми числами, а расшиф­ ровка приведена в прилагаемом к набору данных файлу README. Анализ данных, хранящихся в трех таблицах, - непростая задача. Пусть, например, требуется вы­ числить средние рейтинги для конкретного фильма в разрезе пола и возраста.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

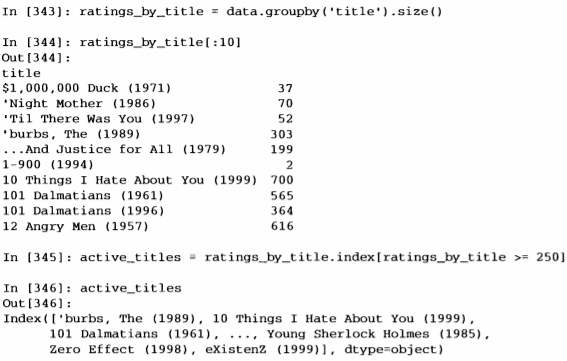
В таком виде агрегирование рейтингов, сгруппированных по одному или не­ скольким атрибутам пользователя или фильма, для человека, хоть немного знако­ мого с pandas, не представляет никаких трудностей. Чтобы получить средние рейтинги фильмов при группировке по полу, воспользуемся методом pivot\_table:



В результате получается еще один объект DataFrame, содержащий средние рей тинги, в котором метками строк являются общее количество оценок фильма метками столбцов - обозначения полов. Сначала я оставлю только фильмы, получившие не менее 250 оценок (число выбрано совершенно произвольно); для это­ го сгруппирую данные по названию и с помощью метода size () получу объект Scries, содержащий размеры групп для каждого наименования:



Затем для отбора строк из приведенного выше объекта mean\_ratings восполь­ зуемся индексом фильмов, получивших не менее 250 оценок:



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

*Sunset Blvd. (a.k.a. Sunset Boulevard) (1950) 4.572650 4.464589*

*Wallace & Grornit: The Best of Aardrnan Anirnation (1996) 4.563107 4.385075*

*Schindler's Listn (1993)*

*4.562602 4.491415*

*Shawshank Rederption, The (1994) Grand Day Out, А (1992)*

*То Kill а Mockingbird (1962) Creature Cornforts (1990) Usual Suspects, The (1995)*

**Измерение несогласия в оценках**

*4.539075 4.560625*

*4.537879 4.293255*

*4.536667 4.372611*

*4.513889 4.272277*

*4.513317 4.518248*

Допустим, мы хотим найти фильмы, по которым мужчины и женщины сильнее всего разошлись в оценках. Для этого можно добавить столбец mean\_ratings, со­ держащий разность средних, а затем отсортировать по нему:

*In [352]: mean\_ratings['diff'] = mean\_ratings['M'] - mean\_ratings['F']*

Сортировка по столбцу · dif f • дает фильмы с наибольшей разностью оценок, которые больше нравятся женщинам:

*In (353]: sorted\_by\_diff = rnean\_ratings.sort\_index(by='diff')*

*In [354]: sorted\_by\_diff(:15]*

*Out(354]:*

*gender*

*Dirty Dancing (1987) Jurnpin' Jack Flash (1986) Grease (1978)*

*Little Women (1994)*

*Steel Magnolias (1989)*

*Anastasia (1997)*

*Rocky Horror Picture Show, The (1975) Color Purple, The (1985)*

*Age of Innocence, The (1993) Free Willy (1993)*

*French Kiss (1995)*

*F 3.790378*

*3.254717*

*3.975265*

*3.870588*

*3.901734*

*3.800000*

*3.673016*

*4.158192*

*3.827068*

*2.921348*

*3.535714*

*м*

*2.959596*

*2.578358*

*3.367041*

*3.321739*

*3.365957*

*3.281609*

*3.160131*

*3.659341*

*3.339506*

*2.438776*

*3.056962*

*diff*

*-0.830782*

*-0.676359*

*-0.608224*

*-0.548849*

*-0.535777*

-о.

*518391*

*-0.512885*

*-0.498851*

*-0.487561*

*-0.482573*

*-0.478752*

*Little Shop of Horrors, The (1960) Guys and Dolls (1955)*

*Mary Poppins (1964)*

*Patch Adarns (1998)*

*3.650000 3.179688 -0.470312*

*4.051724 3.583333 -0.468391*

*4.197740 3.730594 -0.467147*

*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*

*Good, The Bad and The Ugly, The (1966) 3.494949 Kentucky Fried Movie, The (1977) 2.878788*

*Durnb & Dumber (1994) 2.697987*

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

*Longest Day, The (1962) 3.411765 4.031447 0.619682*

'

*СаЫе Guy, The (1996) 2.250000 2.863787 0.613787*

*Evil Dead II (Dead Ву 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 а 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 раз­ ногласие 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*

*Evitа ( 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 (1). 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(lO)*

*Out [4]:*

*name sex births year Mary F 7065 1880*

о0

*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 (совпадаю­ щее 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*](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 . z ip, вы получите каталог, содержащий файлы с именами вида уоЫ880. txt. С помощью команды UNIX head я могу вывести первые 10 строк каждого файла (в Windows можно воспользоваться командой mоrе или открыть файл в текстовом редакторе):

*In [367]: !head -n 10 names/yoЫ880.txt Mary,F,7065*

*Anna,F,2604 Emma,F,2003*

*Elizabeth,F,1939 Minnie,F,1746 Margaret,F,1578*

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

*Ida,F,1472*

*Alice,F,1414 Bertha,F,1320 Sarah,F,1288*

Поскольку поля разделены запятыми, файл можно загрузитьв объект DataFrame методом pandas. read\_csv:

*In (368]: import pandas as pd*

*In (369]: names1880 = pd.read\_csv('names/yoЬ1880. txt', names= ['name', 'sex', 'births']) In (370]: nameslBBO*

*Out (370]:*

*<class 'pandas.core.frame.DataFrame'> Int64Index: 2000 entries, О 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 мла­ денцев в году, поэтому для простоты сумму значений в столбце sex можно считать общим числом родившихся в данном году младенцев:

*In (371]: names1000.groupby('sex').births.sum()*

*Out (371] :*

*sex*

*F 90993*

*М 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 = []*

*colurnns = [ 'name', 'sex', 'births' ]*

*for year in years:*

*path = 'narnes/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 умол­ чанию объединяет объекты DataFrame построчно. Во-вторых, следует задать па­ раметр ignore\_index= True, потому что нам неинтересно сохранять исходные но­ мера строк, прочитанных методом read\_csv. Таким образом, мы получили очень большой DataFrame, содержащий данные обо всех именах.

Выглядит объект names следующим образом:

*In [373]: narnes Out [373 J :*

*<class 'pandas.core.frarne.DataFrame'> Int64Index: 1690784 entries, О to 1690783 Data colurnns:*

*Name 1690784 non-null values sex 1690784 non-null values births 1690784 noп-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 pi vot\_tablee (см. рис. 2.4):

*In [374): total\_births narnes.pivot\_tablee('births', rows= 'year',*

*cols= 'sex’, aggfunc=surn)*

*In [375): total\_births.tail() Out[375]:*

*sex F м*

*year*

*2006 1896468 2050234*

*2007 1916888 2069242*

*2008 1883645 2032310*

*2009 1827643 1973359*

*2010 1759010 1898382*

*In [376]: total\_births.plot(title= 'Tota1 births bу sex and year')*

Далее вставим столбец prop, содержащий долю младенцев, получивших данное имя, относительно общего числа родившихся. Значение prop, равное о. 02, озна­ чает, что данное имя получили 2 из 100 млменцев. Затем сгруппируем данные по году и полу и добавим в каждую группу новый столбец:

def add\_prop(group):

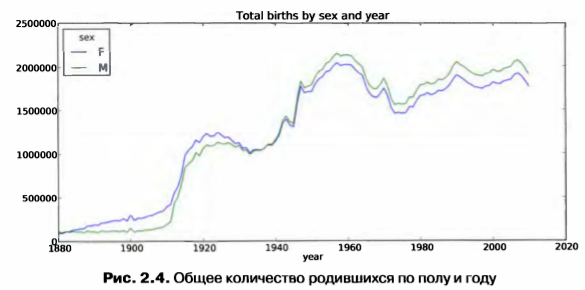
*# При целочисленном делении производится округление с недостатком births = group.births.astype(float)*

*group['prop'] = births / births.surn() 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!) .



Получившийся в результате упрощенный набор данных состоит из таких столбцов:

*In (378]: names*

*Out (378]:*

*<class 'pandas.core.frame.DataFrame'> Int64Index: 1690784 entries, О 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\_toplOOO(group):*

*return group.sort\_index(by='births', ascending=False) (:1000] grouped names. groupby(['year', 'sex' J)*

*toplOOO = 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 births*

*261877*

*261877*

*non-null values 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***

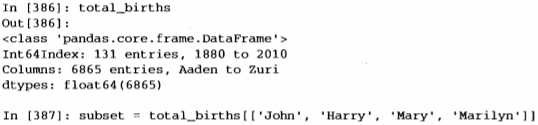
Имея полный набор данных и первые 1000 записей, мы можем приступить к анализу различных интересных тенденций. Для начала решим простую задачу: разобьем набор Тор 1000 на части, относящиеся к мальчикам и девочкам.

*In [383]: boys = top1000 [top1000.sex = 'М'] In [384]: girls = top1000 [top1000.sex =='F']*

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



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



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

*In (388]: subset.plot(subplots=True, figsize= (l2, 10), grid= False,*

*. ... . . title= "Number of births per year")*

Результат показан на рис. 2.5. Глядя на него, можно сделать вывод, что эти име­ на в Америке вышли из моды. Но на самом деле картина несколько сложнее, как станет ясно в следующем разделе.

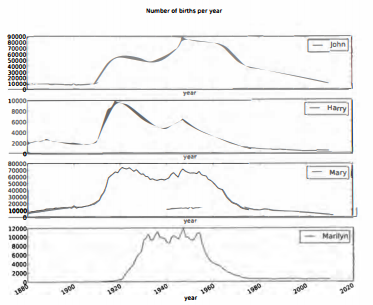


Рис. 2.5. Распределение нескольких имен мальчиков и девочек по годам

**Измерение роста разнообразия имен**

Убывание кривых на рисунках выше можно объяснить тем, что меньше роди­ телей стали выбирать такие распространенные имена. Эту гипотезу можно про­ верить и подтвердить имеющимися данными. Один из возможных показателей - доля родившихся в наборе 1000 самых популярных имен, который я агрегирую по году и полу:





Результат показан на рис. 2.6. Действительно, похоже, что разнообразие имен растет (доля в первой тысяче падает). Другой интересный показатель - количе­ ство различных имен среди первых 50 % родившихся, упорядоченное по популяр­ ности в порядке убывания. Вычислить его несколько сложнее. Рассмотрим только имена мальчиков, родившихся в 2010 году:

*In [392]: df = boys[boys.year == 2010] In [393): df*

*Out [393]:*

*<class 'pandas.core.frame.DataFrame'> Int64Index: 1000 entries, 260877 to 261876 Data columns:*

*name 1000 non-null values sex 1000 non-null values births 1000 non-null values year 1000 non-null values prop 1000 non-null values*

*dtypes: float64(1), int64(2), object(2)*

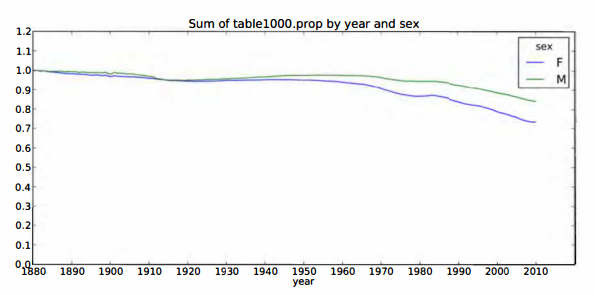


Рис. 2.6. Доля родившихся мальчиков и девочек, представленных в первой тысяче имен

После сортировки prop в порядке убывания мы хотим узнать, сколько популяр­ ных имен нужно взять, чтобы достичь 50 %. Можно написать для этого цикл for, но NurnPy предлагает более хитроумный векторный подход. Если вычислить на­ копительные суммы cumsum массива prop, а затем вызвать метод searchsorted, то будет возвращена позиция в массиве накопительных сумм, в которую нужно было бы вставить о . 5, чтобы не нарушить порядок сортировки:

*In [394): prop\_cumsum = df.sort\_index(by='prop', ascending=False).prop. cumsum ()*

*In [395): prop\_cumsum(:10)*

*Out [395]:*

*260877 0.011523*

*260878 0.020934*

*260879 0.029959*

*260880 0.038930*

*260881 0.047817*

*260882 0.056579*

*260883 0.065155*

*260884 0.073414*

*260885 0.081528*

**Имена, которые давали детям в США за период с 1880 по 201О rод**

*260886 0.089621*

*In [396): prop\_curnsum.searchsorted(0.5)*

*Out[396]: 116*

Поскольку индексация массивов начинается с нуля, то нужно прибавить к ре­ зультату 1 - получится 117. Заметим, что в 1900 году этот показатель был гораздо меньше:

*In [397]: df = boys[boys.year == 1900)*

*In [398): in1900 = df.sort\_index(by='prop', ascending=False).prop.cumsum() In [399]: in1900.searchsorted(0.5) + 1*

*Out[399]: 2 5*

Теперь нетрудно применить эту операцию к каждой комбинации года и пола; произведем группировку по этим полям с помощью метода groupby, а затем с по­ мощью метода apply применим функцию, возвращающую нужный показатель для каждой группы:

*def get\_quantile\_count(group, q=0.5):*

*group = group.sort\_index(by='prop', ascending=False) return group.prop.curnsum() .searchsorted(q) + 1*

*diversity diversity*

*top1000.groupby(['year•, 'sex']).apply(get\_quantile\_count) diversity.unstack('sex')*

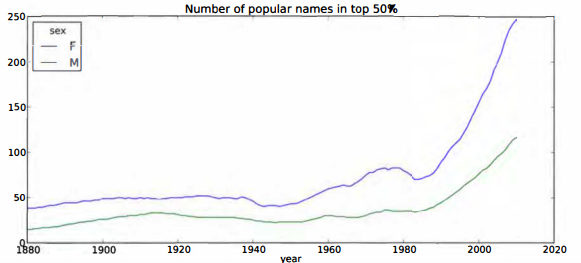
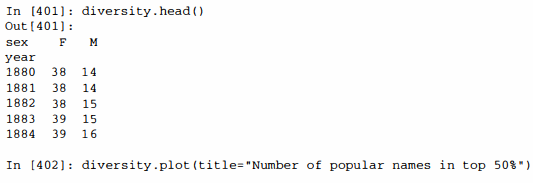


Рис. 2. 7. График зависимости разнообразия от года

В получившемся объекте DataFrame с именем di versi ty хранится два времен­ ных ряда, по одному для каждого поля, индексированные по году. Его можно ис­ следовать в IPython и, как и раньше, нанести на график (рис. 2.7).



Как видим, девочкам всегда давали более разнообразные имена, чем мальчикам, и со временем эта тенденция проявляется все ярче. Анализ того, что именно явля­ ется причиной разнообразия, например рост числа вариантов написания одного и того же имени, оставляю читателю.

**Революция «последней буквы»**

В 2007 году исследователышца детских имен Лаура Уоттенберг (Laura Wattenberg) отметила на своем сайте *(http.j/ww&.1.babynamewizard.com),* что распределение имен мальчиков по последней букве за последние 100 лет существенно измени­ лось. Чтобы убедиться в этом, я сначала агрегирую данные полного набора обо всех родившихся по году, полу и последней букве:

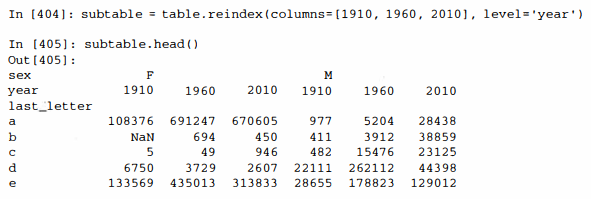
*# извлекаем последнюю букву имени в столбце name get\_last\_letter = lamЬda х: х[-1]*

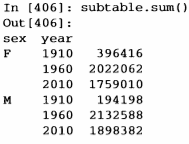
*last\_letters = names.name.map(get\_last\_letter) last letters.name = 'last letter'*

*tablе = names.pivot\_table('births', rows=last\_letters,*

*cols=['sex', 'year'], aggfunc=sum)*

Затем выберу из всего периода три репрезентативных года и напечатаю первые несколько строк:

 Далее я пронормирую эту таблицу на общее число родившихся, чтобы вычис­ лить новую таблицу, содержащую долю от общего родившихся для каждого пола и каждой последней буквы:



*In (407]: letter\_prop = subtable / subtable.sum().astype(float)*

Зная доли букв, я теперь могу нарисовать столбчатые диаграммы для каждого пола, разбив их по годам. Результат показан на рис. 2.8.

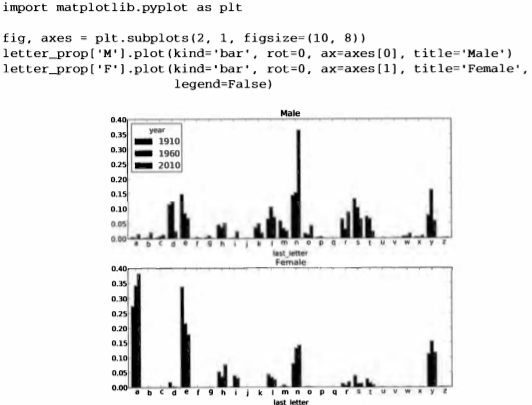
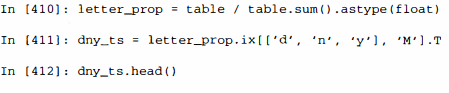
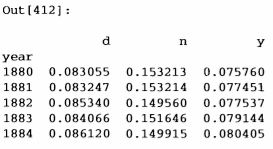


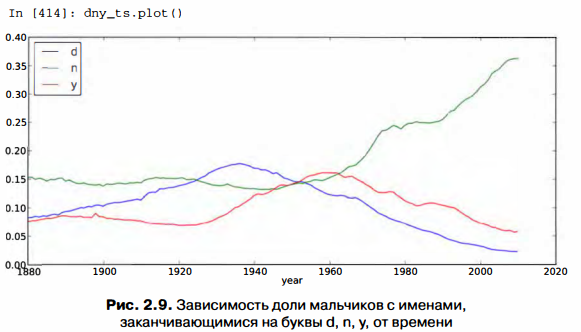
Рис. 2.8. Доли имен мальчиков и девочек, заканчивающихся на каждую букву

Как видим, с 1960-х годов доля имен мальчиков, заканчивающихся буквой <n>,значительно возросла. Снова вернусь к созданной ранее полной таблице, пронор­ мирую ее по году и полу, выберу некое подмножество букв для имен мальчиков и транспонирую, чтобы превратить каждый столбец во временной ряд:





Имея этот объект DataFrame, содержащие временные ряды, я могу все тем же методом plot построить график изменения тенденций в зависимости от времени (рис. 2.9):



**Мужские имена, ставшие женскими, и наоборот**

Еще одно интересное упражнение - изучить имена, которые раньше часто давали мальчикам, а затем «сменили пол». Возьмем, к примеру, имя Leslcey или Leslie. По набору top1000 вычисляю список имен, начинающихся с ‘lesl ‘:

*In (415]: all\_names = top1000.name.unique()*

*In [416]: mask = np.array(['lesl' in x.lower() for х in all\_names]) In [417]: lesley\_like = all\_names[mask]*

*In [418): lesley\_like*

*Out[418]: array([Leslie, Lesley, Leslee, Lesli, Lesly], dtype=object)*

Далее можно оставить только эти имена и просуммировать количество родив­ шихся, сгруппировав по имени, чтобы найти относительные частоты:

**Имена, которые довали детям в США эо пернод с 1880 по 2010 год**

*In [419): filtered = top1000[top1000.name.isin(lesley\_like)] In [420): filtered.groupby('name').births.sum()*

*Out [420):*

*name Leslee Lesley Lesli Leslie Lesly*

*1082*

*35022*

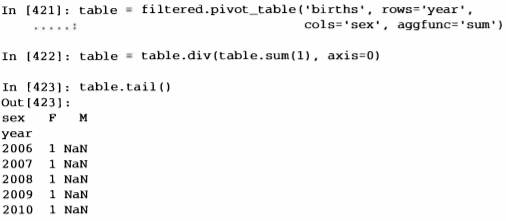
*929*

*370429*

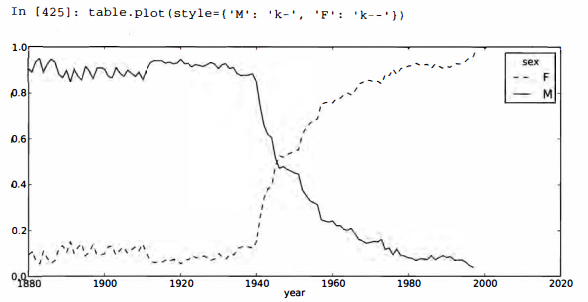
*10067*

*Name: births*

Затем агрегируем по полу и году и нормируем в пределах каждого года:



Наконец, нетрудно построить график распределения по полу в зависимости от времени (рис. 2.10).



Выводы и перспективы

Приведенные в этой главе примеры довольно просты, но они позволяют составить представление о том, чего ожидать от последующих глав. Эта книга посвящена, прежде всего, инструментам, а не демонстрации более сложных аналитических методов. Овладев описываемыми приемами, вы сможете без труда реализовать собственные методы анализа ( если, конечно, знаете, чего хотите)