数据分析 9 数据分析案例

来自Bitly的USA.gov数据

2011年，URL缩短服务Bitly跟美国政府网站USA.gov合作，提供了一份从生成.gov或.mil短链接的用户那里收集来的匿名数据。

以每小时快照为例，文件中各行的格式为JSON

```
In [5]: path = 'datasets/bitly_usagov/example.txt'

In [6]: open(path).readline()
Out[6]: '{ "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'
```

使用json模块及其loads函数逐行加载已经下载好的数据文件

```
import json
path = 'datasets/bitly_usagov/example.txt'
records = [json.loads(line) for line in open(path)]
```

用纯Python代码对时区进行计数

求该数据集中最常出现的是哪个时区（即tz字段）

```
In [12]: time_zones = [rec['tz'] for rec in records]
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-12-db4fbd348da9> in <module>()
----> 1 time_zones = [rec['tz'] for rec in records]
<ipython-input-12-db4fbd348da9> in <listcomp>(.0)
```

```
----> 1 time_zones = [rec['tz'] for rec in records]
KeyError: 'tz'
```

因为并不是所有记录都有时区字段

```
In [13]: time_zones = [rec['tz'] for rec in records if 'tz' in rec]

In [14]: time_zones[:10]
Out[14]:
['America/New_York',
 'America/Denver',
 'America/New_York',
 'America/Sao_Paulo',
 'America/New_York',
 'America/New_York',
 'Europe/Warsaw',
 '',
 '',
 '']
```

对时区进行计数，这里介绍两个办法：一个较难（只使用标准Python库），另一个较简单（使用pandas）

计数的办法之一是在遍历时区的过程中将计数值保存在字典中

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

```
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
```

```
In [17]: counts = get_counts(time_zones)

In [18]: counts['America/New_York']
Out[18]: 1251

In [19]: len(time_zones)
Out[19]: 3440
```

要得到前10位的时区及其计数值

```
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:]
```

```
In [21]: top_counts(counts)
Out[21]:
[(33, 'America/Sao_Paulo'),
 (35, 'Europe/Madrid'),
(36, 'Pacific/Honolulu'),
 (37, 'Asia/Tokyo'),
 (74, 'Europe/London'),
 (191, 'America/Denver'),
 (382, 'America/Los_Angeles'),
 (400, 'America/Chicago'),
 (521, ''),
 (1251, 'America/New_York')]
```

collections.Counter类，它可以使这项工作更简单

```
In [22]: from collections import Counter

In [23]: counts = Counter(time_zones)

In [24]: counts.most_common(10)
Out[24]:
[('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)]
```

用pandas对时区进行计数

```
In [25]: import pandas as pd

In [26]: frame = pd.DataFrame(records)

In [27]: frame.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3560 entries, 0 to 3559
Data columns (total 18 columns):
_heartbeat_    120 non-null float64
a              3440 non-null object
al             3094 non-null object
c              2919 non-null object
cy             2919 non-null object
g              3440 non-null object
```

```
gr              2919 non-null object
h               3440 non-null object
hc              3440 non-null float64
hh              3440 non-null object
kw              93 non-null object
l               3440 non-null object
ll              2919 non-null object
nk              3440 non-null float64
r               3440 non-null object
t               3440 non-null float64
tz              3440 non-null object
u               3440 non-null object
dtypes: float64(4), object(14)
memory usage: 500.7+ KB

In [28]: frame['tz'][:10]
Out[28]:
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, dtype: object
```

对Series使用value_counts方法

```
In [29]: tz_counts = frame['tz'].value_counts()

In [30]: tz_counts[:10]
Out[30]:
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
Name: tz, dtype: int64
```

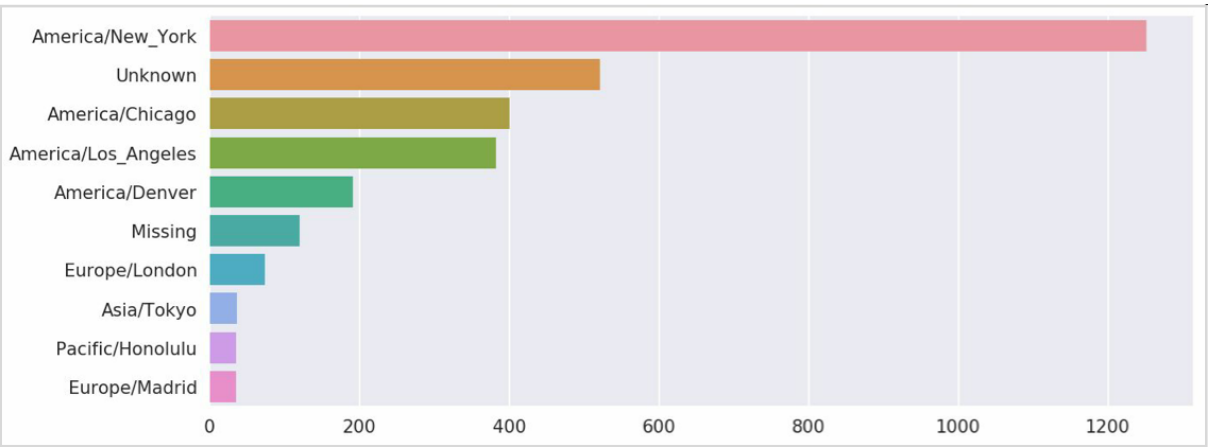
可以用matplotlib可视化这个数据。为此，我们先给记录中未知或缺失的时区填上一个替代值。fillna函数可以替换缺失值（NA），而未知值（空字符串）则可以通过布尔型数组索引加以替换

```
In [31]: clean_tz = frame['tz'].fillna('Missing')

In [32]: clean_tz[clean_tz == ''] = 'Unknown'

In [33]: tz_counts = clean_tz.value_counts()

In [34]: tz_counts[:10]
Out[34]:
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
Name: tz, dtype: int64
```

用seaborn包创建水平柱状图

```
In [36]: import seaborn as sns
```

```
In [37]: subset = tz_counts[:10]

In [38]: sns.barplot(y=subset.index, x=subset.values)
```



a字段含有执行URL短缩操作的浏览器、设备、应用程序的相关信息

```
In [39]: frame['a'][1]
Out[39]: 'GoogleMaps/RochesterNY'

In [40]: frame['a'][50]
Out[40]: 'Mozilla/5.0 (Windows NT 5.1; rv:10.0.2)
Gecko/20100101 Firefox/10.0.2'

In [41]: frame['a'][51][:50]  # long line
Out[41]: 'Mozilla/5.0 (Linux; U; Android 2.2.2; en-us; LG-P9'
```

将这些”agent”字符串中的所有信息都解析出来是一件挺郁闷的工作。一种策略是将这种字符串的第一节（与浏览器大致对应）分离出来并得到另外一份用户行为摘要

```
In [42]: results = pd.Series([x.split()[0] for x in frame.a.dropna()])

In [43]: results[:5]
Out[43]:
0               Mozilla/5.0
1    GoogleMaps/RochesterNY
2               Mozilla/4.0
3               Mozilla/5.0
```

```
4               Mozilla/5.0
dtype: object

In [44]: results.value_counts()[:8]
Out[44]:
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
dtype: int64
```

假设想按Windows和非Windows用户对时区统计信息进行分解。为了简单起见，我们假定只要agent字符串中含有"Windows"就认为该用户为Windows用户。由于有的agent缺失，所以首先将它们从数据中移除

```
In [45]: cframe = frame[frame.a.notnull()]
```

然后计算出各行是否含有Windows的值

```
In [47]: cframe['os'] = np.where(cframe['a'].str.contains('Windows'),
   ....:                         'Windows', 'Not Windows')

In [48]: cframe['os'][:5]
Out[48]:
0        Windows
1    Not Windows
2        Windows
3    Not Windows
4        Windows
Name: os, dtype: object
```

根据时区和新得到的操作系统列表对数据进行分组

```
In [49]: by_tz_os = cframe.groupby(['tz', 'os'])
```

分组计数，类似于value_counts函数，可以用size来计算。并利用unstack对计数结果进行重塑

```
In [50]: agg_counts = by_tz_os.size().unstack().fillna(0)

In [51]: agg_counts[:10]
Out[51]:
os                               Not Windows  Windows
tz
                                       245.0    276.0
Africa/Cairo                             0.0      3.0
Africa/Casablanca                        0.0      1.0
Africa/Ceuta                             0.0      2.0
Africa/Johannesburg                      0.0      1.0
Africa/Lusaka                            0.0      1.0
America/Anchorage                        4.0      1.0
America/Argentina/Buenos_Aires           1.0      0.0
America/Argentina/Cordoba                0.0      1.0
America/Argentina/Mendoza                0.0      1.0
```

根据agg_counts中的行数构造了一个间接索引数组

```
# Use to sort in ascending order
In [52]: indexer = agg_counts.sum(1).argsort()

In [53]: indexer[:10]
Out[53]:
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
dtype: int64
```

通过take按照这个顺序截取了最后10行最大值

```
In [54]: count_subset = agg_counts.take(indexer[-10:])

In [55]: count_subset
Out[55]:
os                   Not Windows  Windows
tz
America/Sao_Paulo           13.0     20.0
Europe/Madrid               16.0     19.0
Pacific/Honolulu             0.0     36.0
Asia/Tokyo                   2.0     35.0
Europe/London               43.0     31.0
America/Denver             132.0     59.0
America/Los_Angeles        130.0    252.0
America/Chicago            115.0    285.0
                           245.0    276.0
America/New_York           339.0    912.0
```

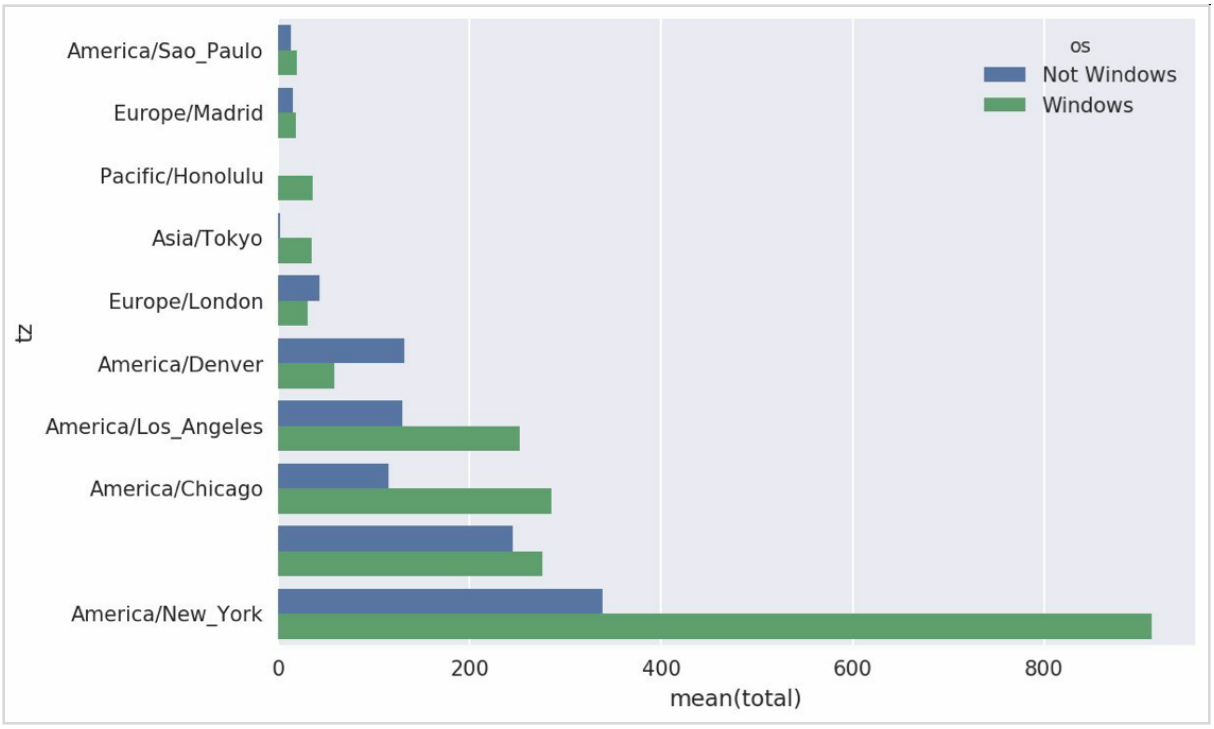
pandas有一个简便方法nlargest，可以做同样的工作

```
In [56]: agg_counts.sum(1).nlargest(10)
Out[56]:
tz
America/New_York       1251.0
                        521.0
America/Chicago         400.0
America/Los_Angeles     382.0
America/Denver          191.0
Europe/London            74.0
Asia/Tokyo               37.0
```

```
Pacific/Honolulu         36.0
Europe/Madrid            35.0
America/Sao_Paulo        33.0
dtype: float64
```

传递一个额外参数到seaborn的barpolt函数，来画一个堆积条形图

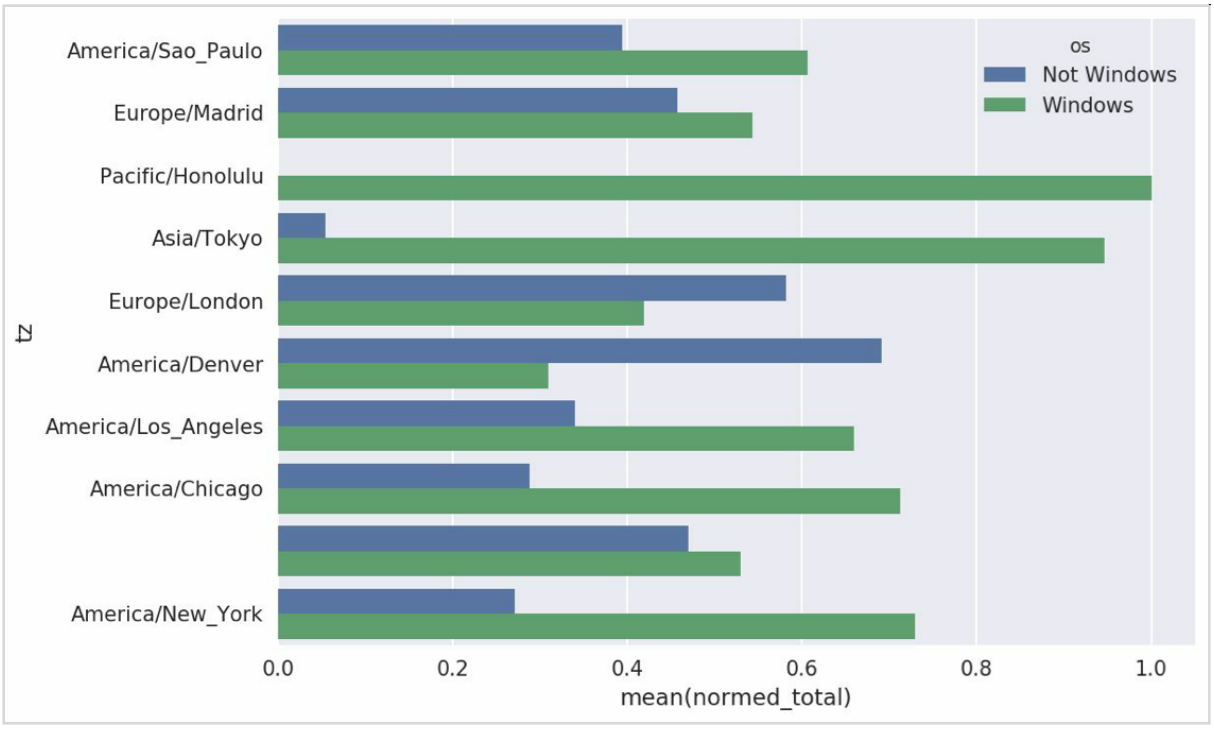
```
# Rearrange the data for plotting
In [58]: count_subset = count_subset.stack()

In [59]: count_subset.name = 'total'

In [60]: count_subset = count_subset.reset_index()

In [61]: count_subset[:10]
Out[61]:
                  tz           os  total
0  America/Sao_Paulo  Not Windows   13.0
1  America/Sao_Paulo      Windows   20.0
2      Europe/Madrid  Not Windows   16.0
3      Europe/Madrid      Windows   19.0
4   Pacific/Honolulu  Not Windows    0.0
5   Pacific/Honolulu      Windows   36.0
6         Asia/Tokyo  Not Windows    2.0
7         Asia/Tokyo      Windows   35.0
8      Europe/London  Not Windows   43.0
9      Europe/London      Windows   31.0

In [62]: sns.barplot(x='total', y='tz', hue='os',  data=count_subset)
```



这张图不容易看出Windows用户在小分组中的相对比例，因此标准化分组百分比之和为1

```
def norm_total(group):
    group['normed_total'] = group.total / group.total.sum()
    return group

results = count_subset.groupby('tz').apply(norm_total)
```

```
In [65]: sns.barplot(x='normed_total', y='tz', hue='os',  data=results)
```



还可以用groupby的transform方法，更高效的计算标准化的和

```
In [66]: g = count_subset.groupby('tz')

In [67]: results2 = count_subset.total / g.total.transform('sum')
```

MovieLens 1M数据集

MovieLens 1M数据集含有来自6000名用户对4000部电影的100万条评分数据。它分为三个表：评分、用户信息和电影信息。

```
import pandas as pd

# Make display smaller
pd.options.display.max_rows = 10

unames = ['user_id', 'gender', 'age', 'occupation', 'zip']
users = pd.read_table('datasets/movielens/users.dat', sep='::',
                      header=None, names=unames, engine='python')

rnames = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_table('datasets/movielens/ratings.dat', sep='::',
```

```
                       header=None, names=rnames, engine='python')

mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table('datasets/movielens/movies.dat', sep='::',
                       header=None, names=mnames, engine='python')
```

查看每个DataFrame的前几行即可验证数据加载工作是否一切顺利

```
In [69]: users[:5]
Out[69]:
   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 [70]: ratings[:5]
Out[70]:
   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 [71]: movies[:5]
Out[71]:
   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 [72]: ratings
Out[72]:
         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
...          ...       ...     ...        ...
1000204     6040      1091       1  956716541
1000205     6040      1094       5  956704887
1000206     6040       562       5  956704746
1000207     6040      1096       4  956715648
1000208     6040      1097       4  956715569
[1000209 rows x 4 columns]
```

想要根据性别和年龄计算某部电影的平均得分，如果将所有数据都合并到一个表中的话问题就简单多了。我们先用pandas的merge函数将ratings跟users合并到一起，然后再将movies也合并进去。pandas会根据列名的重叠情况推断出哪些列是合并（或连接）键

```
In [73]: data = pd.merge(pd.merge(ratings, users), movies)

In [74]: data
Out[74]:
         user_id  movie_id  rating  timestamp gender  age  occupation    zip  \
0              1      1193       5  978300760      F    1          10  48067
1              2      1193       5  978298413      M   56          16  70072
2             12      1193       4  978220179      M   25          12  32793
3             15      1193       4  978199279      M   25           7  22903
4             17      1193       5  978158471      M   50           1  95350
...          ...       ...     ...        ...    ...  ...         ...    ...
1000204     5949      2198       5  958846401      M   18          17  47901
1000205     5675      2703       3  976029116      M   35          14  30030
1000206     5780      2845       1  958153068      M   18          17  92886
1000207     5851      3607       5  957756608      F   18          20  55410
1000208     5938      2909       4  957273353      M   25           1  35401
                                            title                genres
0            One Flew Over the Cuckoo's Nest (1975)                 Drama
1            One Flew Over the Cuckoo's Nest (1975)                 Drama
2            One Flew Over the Cuckoo's Nest (1975)                 Drama
3            One Flew Over the Cuckoo's Nest (1975)                 Drama
```

```
4             One Flew Over the Cuckoo's Nest (1975)                  Drama
...                                                ...                   ...
1000204                          Modulations (1998)           Documentary
1000205                        Broken Vessels (1998)                 Drama
1000206                           White Boys (1999)                 Drama
1000207                    One Little Indian (1973)  Comedy|Drama|Western
1000208  Five Wives, Three Secretaries and Me (1998)           Documentary
[1000209 rows x 10 columns]

In [75]: data.iloc[0]
Out[75]:
user_id                                             1
movie_id                                         1193
rating                                              5
timestamp                                   978300760
gender                                              F
age                                                 1
occupation                                         10
zip                                             48067
title         One Flew Over the Cuckoo's Nest (1975)
genres                                          Drama
Name: 0, dtype: object
```

按性别计算每部电影的平均得分，我们可以使用pivot_table方法

```
In [76]: mean_ratings = data.pivot_table('rating', index='title',
   .....:                                 columns='gender', aggfunc='mean')

In [77]: mean_ratings[:5]
Out[77]:
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
```

过滤掉评分数据不够250条的电影（随便选的一个数字）。为了达到这个目的，先对title进行分组，然后利用size()得到一个含有各电影分组大小的Series对象

```
In [78]: ratings_by_title = data.groupby('title').size()

In [79]: ratings_by_title[:10]
Out[79]:
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
dtype: int64

In [80]: active_titles = ratings_by_title.index[ratings_by_title >= 250]

In [81]: active_titles
Out[81]:
Index(['''burbs, The (1989)', '10 Things I Hate About You (1999)',
       '101 Dalmatians (1961)', '101 Dalmatians (1996)', '12 Angry Men (1957)',
       '13th Warrior, The (1999)', '2 Days in the Valley (1996)',
       '20,000 Leagues Under the Sea (1954)', '2001: A Space Odyssey (1968)',
       '2010 (1984)',
       ...
'X-Men (2000)', 'Year of Living Dangerously (1982)',
       'Yellow Submarine (1968)', 'You've Got Mail (1998)',
       'Young Frankenstein (1974)', 'Young Guns (1988)',
       'Young Guns II (1990)', 'Young Sherlock Holmes (1985)',
       'Zero Effect (1998)', 'eXistenZ (1999)'],
      dtype='object', name='title', length=1216)
```

标题索引中含有评分数据大于250条的电影名称，然后我们就可以据此从前面的mean_ratings中选取所需的行了

```
# Select rows on the index
In [82]: mean_ratings = mean_ratings.loc[active_titles]

In [83]: mean_ratings
Out[83]:
gender                                   F         M
title
'burbs, The (1989)                 2.793478  2.962085
10 Things I Hate About You (1999)  3.646552  3.311966
101 Dalmatians (1961)              3.791444  3.500000
101 Dalmatians (1996)              3.240000  2.911215
12 Angry Men (1957)                4.184397  4.328421
...                                     ...       ...
Young Guns (1988)                  3.371795  3.425620
Young Guns II (1990)               2.934783  2.904025
Young Sherlock Holmes (1985)       3.514706  3.363344
Zero Effect (1998)                 3.864407  3.723140
eXistenZ (1999)                    3.098592  3.289086
[1216 rows x 2 columns]
```

为了了解女性观众最喜欢的电影，我们可以对F列降序排列

```
In [85]: top_female_ratings = mean_ratings.sort_values(by='F', ascending=False)

In [86]: top_female_ratings[:10]
Out[86]:
gender                                                     F         M
title
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...   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
```

计算评分分歧

要找出男性和女性观众分歧最大的电影。一个办法是给mean_ratings加上一个用于存放平均得分之差的列，并对其进行排序

```
In [87]: mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']
```

按“diff”排序即可得到分歧最大且女性观众更喜欢的电影

```
In [88]: sorted_by_diff = mean_ratings.sort_values(by='diff')

In [89]: sorted_by_diff[:10]
Out[89]:
gender                                     F         M      diff
title
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
```

对排序结果反序并取出前10行，得到的则是男性观众更喜欢的电影

```
# Reverse order of rows, take first 10 rows
In [90]: sorted_by_diff[::-1][:10]
Out[90]:
gender                                       F         M      diff
```

```
title
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
```

如果只是想要找出分歧最大的电影（不考虑性别因素），则可以计算得分数据的方差或标准差，
在统计描述中，方差用来计算每一个变量（观察值）与总体均数之间的差异。

```
# Standard deviation of rating grouped by title
In [91]: rating_std_by_title = data.groupby('title')['rating'].std()

# Filter down to active_titles
In [92]: rating_std_by_title = rating_std_by_title.loc[active_titles]

# Order Series by value in descending order
In [93]: rating_std_by_title.sort_values(ascending=False)[:10]
Out[93]:
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, dtype: float64
```

1880-2010年间全美婴儿姓名

用UNIX的head命令查看了其中一个文件的前10行

```
In [94]: !head -n 10 datasets/babynames/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
```

```
In [95]: import pandas as pd

In [96]: names1880 = pd.read_csv('datasets/babynames/yob1880.txt',
   ....:                         names=['name', 'sex', 'births'])

In [97]: names1880
Out[97]:
           name sex  births
0          Mary   F    7065
1          Anna   F    2604
2          Emma   F    2003
3     Elizabeth   F    1939
4        Minnie   F    1746
...         ...  ..     ...
1995     Woodie   M       5
1996     Worthy   M       5
1997     Wright   M       5
1998       York   M       5
1999  Zachariah   M       5
[2000 rows x 3 columns]
```

我们可以用births列的sex分组小计表示该年度的births总计

```
In [98]: names1880.groupby('sex').births.sum()
Out[98]:
sex
F     90993
M    110493
Name: births, dtype: int64
```

由于该数据集按年度被分隔成了多个文件，所以第一件事情就是要将所有数据都组装到一个DataFrame里面，并加上一个year字段。使用pandas.concat即可达到这个目的

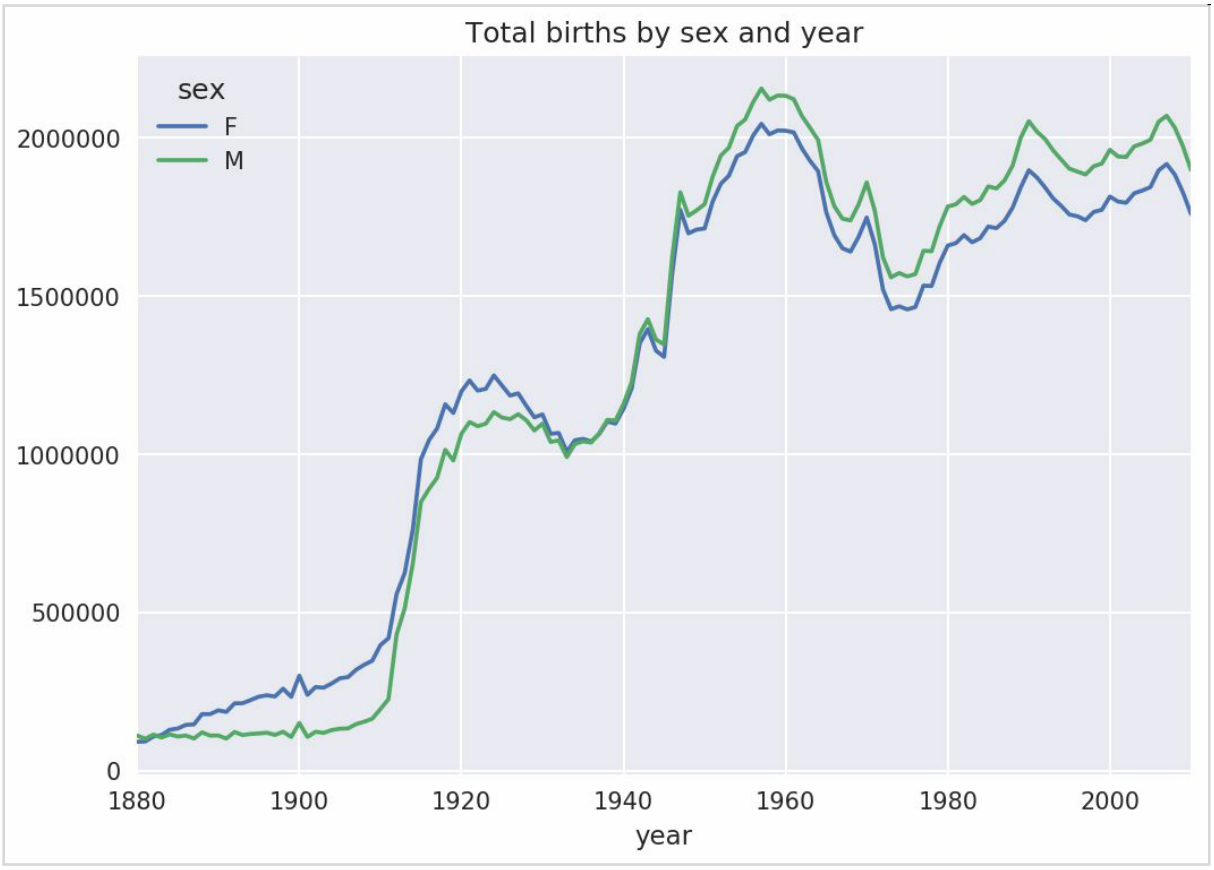
```
years = range(1880, 2011)

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

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

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

# Concatenate everything into a single DataFrame
names = pd.concat(pieces, ignore_index=True)
```

这里需要注意几件事情。第一，concat默认是按行将多个DataFrame组合到一起的；第二，必须指定ignore_index=True，因为我们不希望保留read_csv所返回的原始行号。

利用groupby或pivot_table在year和sex级别上对其进行聚合了

```
In [101]: total_births = names.pivot_table('births', index='year',
   .....:                                  columns='sex', aggfunc=sum)
```

```
In [102]: total_births.tail()
Out[102]:
sex         F        M
year
2006  1896468  2050234
2007  1916888  2069242
2008  1883645  2032310
2009  1827643  1973359
2010  1759010  1898382

In [103]: total_births.plot(title='Total births by sex and year')
```



插入一个prop列，用于存放指定名字的婴儿数相对于总出生数的比例

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

```
In [105]: names
Out[105]:
              name sex  births  year      prop
0             Mary   F    7065  1880  0.077643
1             Anna   F    2604  1880  0.028618
2             Emma   F    2003  1880  0.022013
3        Elizabeth   F    1939  1880  0.021309
4           Minnie   F    1746  1880  0.019188
...            ...  ..     ...   ...       ...
1690779    Zymaire   M       5  2010  0.000003
1690780     Zyonne   M       5  2010  0.000003
1690781  Zyquarius   M       5  2010  0.000003
1690782      Zyran   M       5  2010  0.000003
1690783      Zzyzx   M       5  2010  0.000003
[1690784 rows x 5 columns]
```

在执行这样的分组处理时，一般都应该做一些有效性检查，比如验证所有分组的prop的总和是否为1

```
In [106]: names.groupby(['year', 'sex']).prop.sum()
Out[106]:
year  sex
1880  F      1.0
      M      1.0
1881  F      1.0
      M      1.0
1882  F      1.0
            ...
2008  M      1.0
2009  F      1.0
      M      1.0
2010  F      1.0
      M      1.0
Name: prop, Length: 262, dtype: float64
```

工作完成。为了便于实现更进一步的分析，我需要取出该数据的一个子集：每对sex/year组合的前1000个名字。这又是一个分组操作

```
def get_top1000(group):
    return group.sort_values(by='births', ascending=False)[:1000]
grouped = names.groupby(['year', 'sex'])
top1000 = grouped.apply(get_top1000)
# Drop the group index, not needed
top1000.reset_index(inplace=True, drop=True)
```

接下来的数据分析工作就针对这个top1000数据集

分析命名趋势

首先将前1000个名字分为男女两个部分

```
In [109]: boys = top1000[top1000.sex == 'M']

In [110]: girls = top1000[top1000.sex == 'F']
```

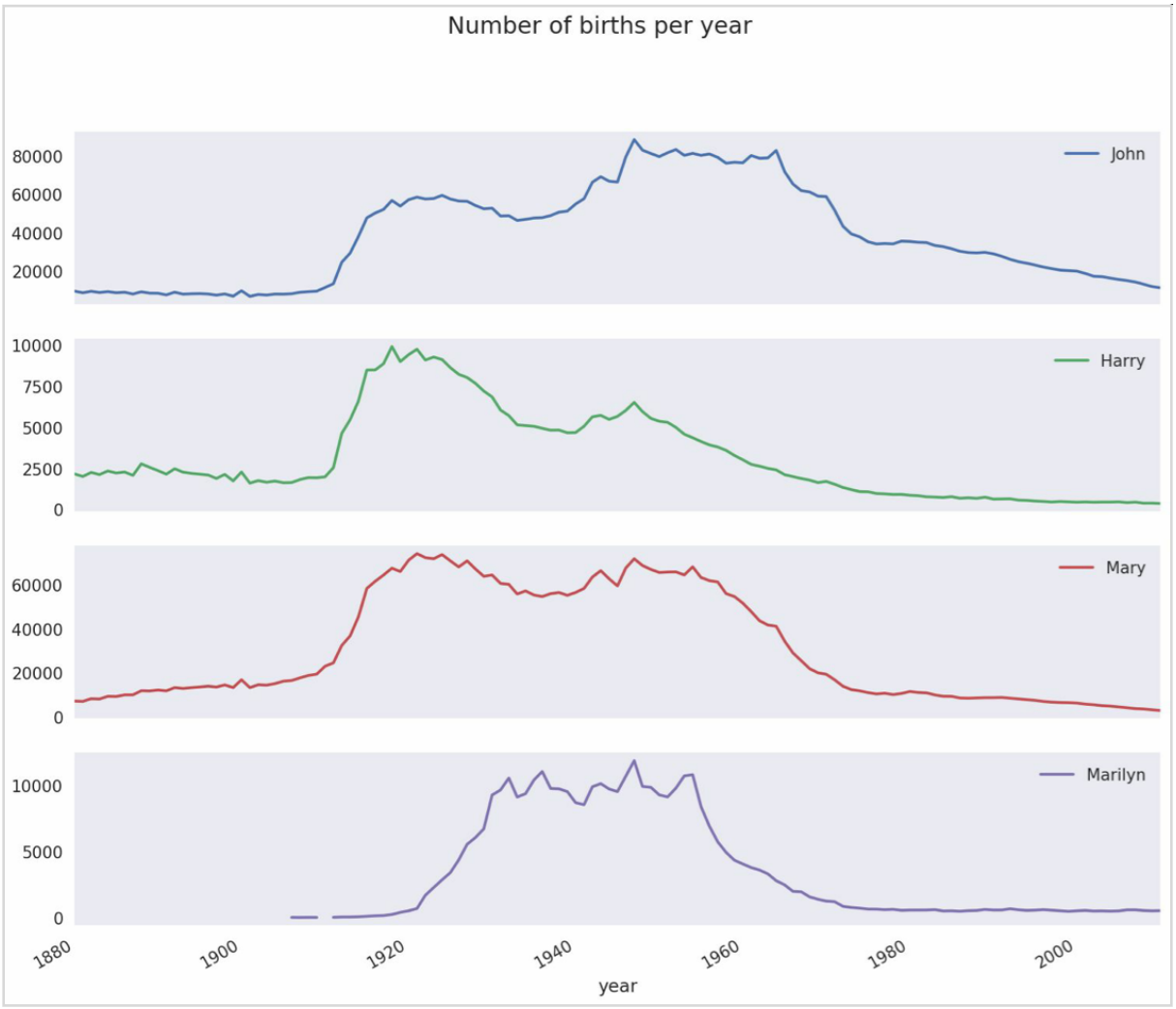
生成一张按year和name统计的总出生数透视表

```
In [111]: total_births = top1000.pivot_table('births', index='year',
   .....:                                     columns='name',
   .....:                                     aggfunc=sum)
```

我们用DataFrame的plot方法绘制几个名字的曲线图

```
In [112]: total_births.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 131 entries, 1880 to 2010
Columns: 6868 entries, Aaden to Zuri
dtypes: float64(6868)
memory usage: 6.9 MB

In [113]: subset = total_births[['John', 'Harry', 'Mary', 'Marilyn']]
```

```
In [114]: subset.plot(subplots=True, figsize=(12, 10), grid=False,
   .....:             title="Number of births per year")
```



评估命名多样性的增长

计算最流行的1000个名字所占的比例，按year和sex进行聚合并绘图

```
In [116]: table = top1000.pivot_table('prop', index='year',
   .....:                             columns='sex', aggfunc=sum)

In [117]: table.plot(title='Sum of table1000.prop by year and sex',
   .....:            yticks=np.linspace(0, 1.2, 13), xticks=range(1880, 2020,
10))
```

从图中可以看出，名字的多样性确实出现了增长（前1000项的比例降低）
另一个办法是计算占总出生人数前50%的不同名字的数量，这个数字不太好计算。我们只考虑2010年男孩的名字：

```
In [118]: df = boys[boys.year == 2010]

In [119]: df
Out[119]:
           name sex  births  year      prop
260877    Jacob   M   21875  2010  0.011523
260878    Ethan   M   17866  2010  0.009411
260879  Michael   M   17133  2010  0.009025
260880   Jayden   M   17030  2010  0.008971
260881  William   M   16870  2010  0.008887
...         ...  ..     ...   ...       ...
261872   Camilo   M     194  2010  0.000102
261873   Destin   M     194  2010  0.000102
261874   Jaquan   M     194  2010  0.000102
261875   Jaydan   M     194  2010  0.000102
261876   Maxton   M     193  2010  0.000102
[1000 rows x 5 columns]
```

在对prop降序排列之后，我们想知道前面多少个名字的人数加起来才够50%。虽然编写一个for循环确实也能达到目的，但NumPy有一种更聪明的矢量方式。先计算prop的累计和cumsum，然后再通过searchsorted方法找出0.5应该被插入在哪个位置才能保证不破坏顺序

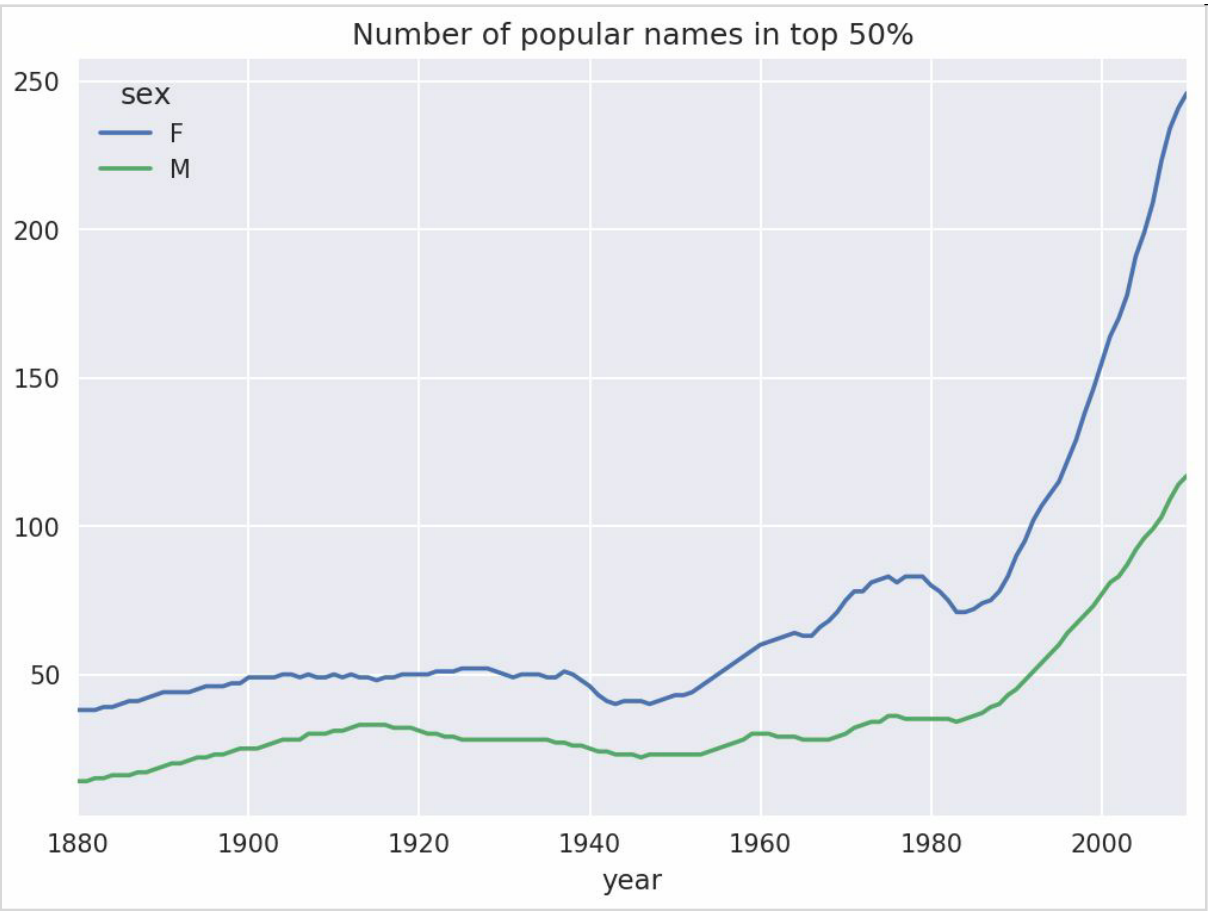
```
In [120]: prop_cumsum = df.sort_values(by='prop',
ascending=False).prop.cumsum()

In [121]: prop_cumsum[:10]
Out[121]:
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
260886    0.089621
Name: prop, dtype: float64

In [122]: prop_cumsum.values.searchsorted(0.5)
Out[122]: 116
```

由于数组索引是从0开始的，因此我们要给这个结果加1，即最终结果为117

```
def get_quantile_count(group, q=0.5):
    group = group.sort_values(by='prop', ascending=False)
    return group.prop.cumsum().values.searchsorted(q) + 1

diversity = top1000.groupby(['year', 'sex']).apply(get_quantile_count)
diversity = diversity.unstack('sex')
```

diversity.plot()
从图中可以看出，女孩名字的多样性总是比男孩的高，而且还在变得越来越高



“最后一个字母”的变革

2007年，一名婴儿姓名研究人员Laura Wattenberg在她自己的网站上指出（ http://www.babynamewizard.com ）：近百年来，男孩名字在最后一个字母上的分布发生了显著的变化

首先将全部出生数据在年度、性别以及末字母上进行了聚合

```
# extract last letter from name column
get_last_letter = lambda x: x[-1]
last_letters = names.name.map(get_last_letter)
last_letters.name = 'last_letter'

table = names.pivot_table('births', index=last_letters,
                          columns=['sex', 'year'], aggfunc=sum)
```

选出具有一定代表性的三年，并输出前面几行

```
In [131]: subtable = table.reindex(columns=[1910, 1960, 2010], level='year')
```

```
In [132]: subtable.head()
Out[132]:
sex                 F                            M
year             1910      1960      2010     1910      1960      2010
last_letter
a            108376.0  691247.0  670605.0    977.0    5204.0   28438.0
b                 NaN     694.0     450.0    411.0    3912.0   38859.0
c                 5.0      49.0     946.0    482.0   15476.0   23125.0
d              6750.0    3729.0    2607.0  22111.0  262112.0   44398.0
e            133569.0  435013.0  313833.0  28655.0  178823.0  129012.0
```

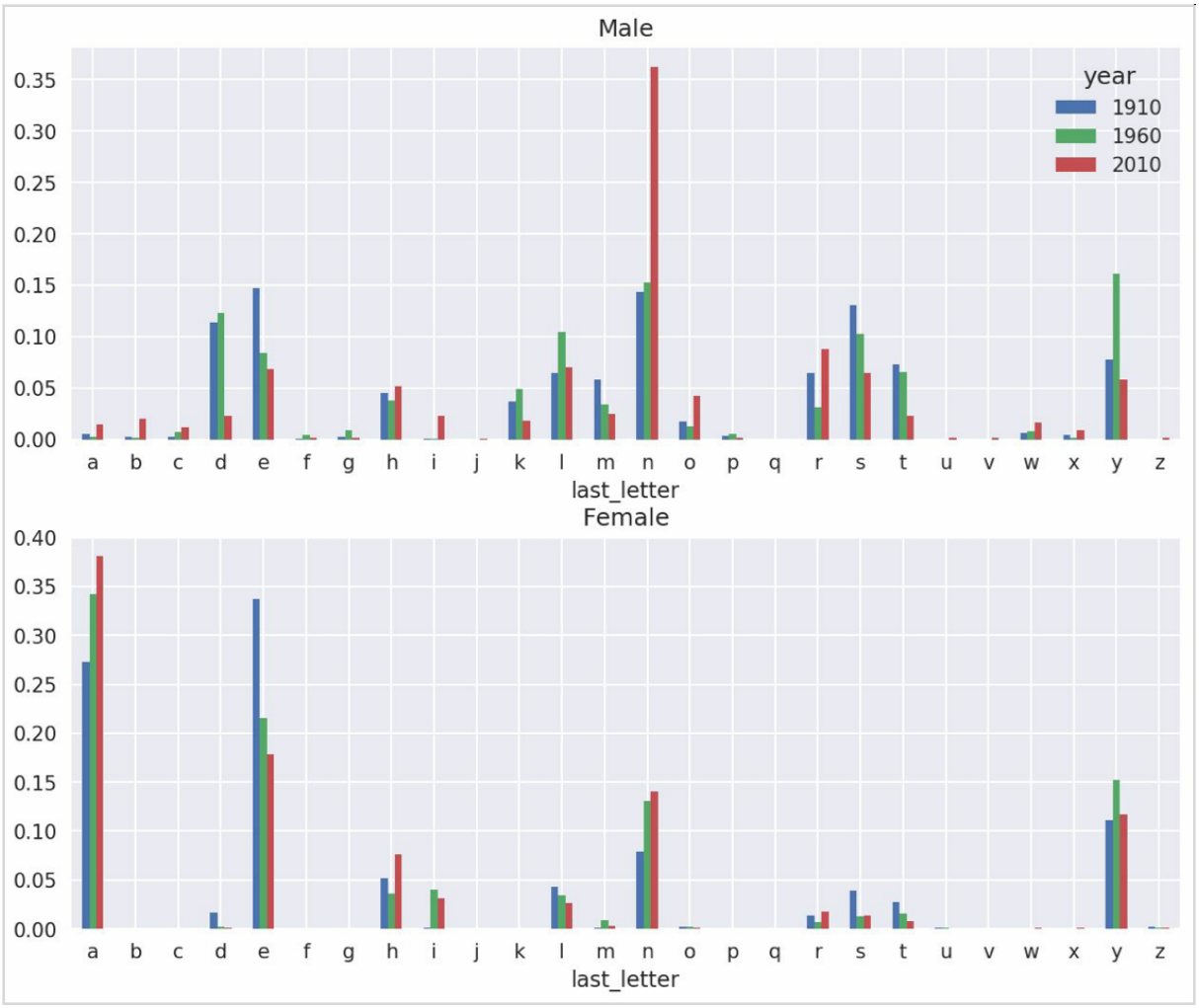
按总出生数对该表进行规范化处理，以便计算出各性别各末字母占总出生人数的比例

```
In [133]: subtable.sum()
Out[133]:
sex  year
F    1910     396416.0
     1960    2022062.0
     2010    1759010.0
M    1910     194198.0
     1960    2132588.0
2010    1898382.0
dtype: float64

In [134]: letter_prop = subtable / subtable.sum()

In [135]: letter_prop
Out[135]:
sex                 F                              M
year             1910      1960      2010      1910      1960      2010
last_letter
a            0.273390  0.341853  0.381240  0.005031  0.002440  0.014980
b                 NaN  0.000343  0.000256  0.002116  0.001834  0.020470
c            0.000013  0.000024  0.000538  0.002482  0.007257  0.012181
d            0.017028  0.001844  0.001482  0.113858  0.122908  0.023387
e            0.336941  0.215133  0.178415  0.147556  0.083853  0.067959
...               ...       ...       ...       ...       ...       ...
```

```
v                NaN  0.000060  0.000117  0.000113
0.000037  0.001434
w           0.000020  0.000031  0.001182  0.006329  0.007711  0.016148
x           0.000015  0.000037  0.000727  0.003965  0.001851  0.008614
y           0.110972  0.152569  0.116828  0.077349  0.160987  0.058168
z           0.002439  0.000659  0.000704  0.000170  0.000184  0.001831
[26 rows x 6 columns]
```

生成一张各年度各性别的条形图

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 1, figsize=(10, 8))
letter_prop['M'].plot(kind='bar', rot=0, ax=axes[0], title='Male')
letter_prop['F'].plot(kind='bar', rot=0, ax=axes[1], title='Female',
                      legend=False)
```



可以看出，从20世纪60年代开始，以字母"n"结尾的男孩名字出现了显著的增长

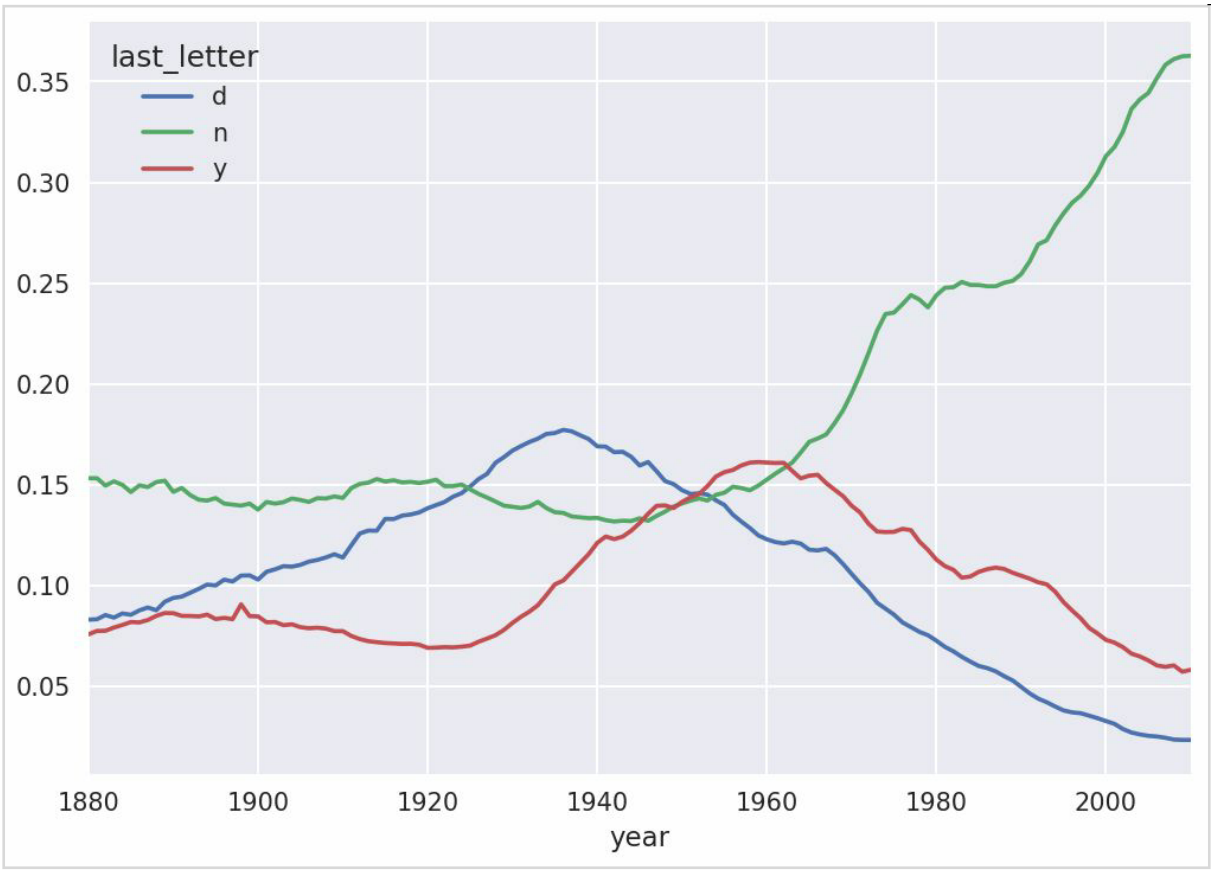
回到之前创建的那个完整表，按年度和性别对其进行规范化处理，并在男孩名字中选取几个字母，最后进行转置以便将各个列做成一个时间序列

```
In [138]: letter_prop = table / table.sum()

In [139]: dny_ts = letter_prop.loc[['d', 'n', 'y'], 'M'].T

In [140]: dny_ts.head()
Out[140]:
last_letter         d         n         y
year
1880         0.083055  0.153213  0.075760
1881         0.083247  0.153214  0.077451
1882         0.085340  0.149560  0.077537
1883         0.084066  0.151646  0.079144
```

```
1884         0.086120  0.149915  0.080405
```

通过其plot方法绘制出一张趋势图

```
In [143]: dny_ts.plot()
```



变成女孩名字的男孩名字

另一个有趣的趋势是，早年流行于男孩的名字近年来“变性了”，例如Lesley或Leslie。回到top1000数据集，找出其中以”lesl”开头的一组名字

```
In [144]: all_names = pd.Series(top1000.name.unique())

In [145]: lesley_like = all_names[all_names.str.lower().str.contains('lesl')]

In [146]: lesley_like
Out[146]:
632     Leslie
2294    Lesley
```

```
4262    Leslee
4728     Lesli
6103     Lesly
dtype: object
```

利用这个结果过滤其他的名字，并按名字分组计算出生数以查看相对频率

```
In [147]: filtered = top1000[top1000.name.isin(lesley_like)]

In [148]: filtered.groupby('name').births.sum()
Out[148]:
name
Leslee      1082
Lesley     35022
Lesli        929
Leslie    370429
Lesly      10067
Name: births, dtype: int64
```

按性别和年度进行聚合，并按年度进行规范化处理

```
In [149]: table = filtered.pivot_table('births', index='year',
   .....:                              columns='sex', aggfunc='sum')

In [150]: table = table.div(table.sum(1), axis=0)

In [151]: table.tail()
Out[151]:
sex     F   M
year
2006  1.0 NaN
2007  1.0 NaN
2008  1.0 NaN
2009  1.0 NaN
2010  1.0 NaN
```

绘制一张分性别的年度曲线图了

```
In [153]: table.plot(style={'M': 'k-', 'F': 'k--'})
```

USDA食品数据库

美国农业部（USDA）制作了一份有关食物营养信息的数据库。

```
{
  "id": 21441,
  "description": "KENTUCKY FRIED CHICKEN, Fried Chicken, EXTRA CRISPY,
Wing, meat and skin with breading",
  "tags": ["KFC"],
  "manufacturer": "Kentucky Fried Chicken",
"group": "Fast Foods",
  "portions": [
    {
      "amount": 1,
      "unit": "wing, with skin",
      "grams": 68.0
    },

    ...
  ],
  "nutrients": [
    {
      "value": 20.8,
      "units": "g",
      "description": "Protein",
      "group": "Composition"
    },

    ...
  ]
}
```

每种食物都带有若干标识性属性以及两个有关营养成分和分量的列表

```
In [154]: import json

In [155]: db = json.load(open('datasets/usda_food/database.json'))

In [156]: len(db)
Out[156]: 6636
```

b中的每个条目都是一个含有某种食物全部数据的字典。nutrients字段是一个字典列表，其中的每个字典对应一种营养成分

```
In [157]: db[0].keys()
Out[157]: dict_keys(['id', 'description', 'tags', 'manufacturer', 'group',
'porti
ons', 'nutrients'])

In [158]: db[0]['nutrients'][0]
Out[158]:
{'description': 'Protein',
 'group': 'Composition',
 'units': 'g',
 'value': 25.18}

In [159]: nutrients = pd.DataFrame(db[0]['nutrients'])

In [160]: nutrients[:7]
Out[160]:
                    description        group units    value
0                       Protein  Composition     g    25.18
1             Total lipid (fat)  Composition     g    29.20
2  Carbohydrate, by difference  Composition     g     3.06
3                           Ash        Other     g     3.28
4                        Energy       Energy  kcal   376.00
5                         Water  Composition     g    39.28
6                        Energy       Energy    kJ  1573.00
```

```
In [161]: info_keys = ['description', 'group', 'id', 'manufacturer']

In [162]: info = pd.DataFrame(db, columns=info_keys)

In [163]: info[:5]
Out[163]:
                          description                   group    id  \
0                    Cheese, caraway  Dairy and Egg Products  1008
1                    Cheese, cheddar  Dairy and Egg Products  1009
2                       Cheese, edam  Dairy and Egg Products  1018
3                       Cheese, feta  Dairy and Egg Products  1019
4  Cheese, mozzarella, part skim milk  Dairy and Egg Products  1028
  manufacturer
0
1
2
3
4

In [164]: info.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6636 entries, 0 to 6635
Data columns (total 4 columns):
description     6636 non-null object
group           6636 non-null object
id              6636 non-null int64
manufacturer    5195 non-null object
dtypes: int64(1), object(3)
memory usage: 207.5+ KB
```

通过value_counts，你可以查看食物类别的分布情况

```
In [165]: pd.value_counts(info.group)[:10]
Out[165]:
Vegetables and Vegetable Products    812
Beef Products                        618
Baked Products                       496
Breakfast Cereals                    403
```

```
Fast Foods                            365
Legumes and Legume Products           365
Lamb, Veal, and Game Products         345
Sweets                                341
Pork Products                         328
Fruits and Fruit Juices               328
Name: group, dtype: int64
```

首先，将各食物的营养成分列表转换为一个DataFrame，并添加一个表示编号的列，然后将该DataFrame添加到一个列表中。最后通过concat将这些东西连接起来就可以了

由于两个DataFrame对象中都有"group"和"description", 需要对它们进行重命名

```
In [170]: col_mapping = {'description' : 'food',
   .....:                'group'       : 'fgroup'}

In [171]: info = info.rename(columns=col_mapping, copy=False)

In [172]: info.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6636 entries, 0 to 6635
Data columns (total 4 columns):
food            6636 non-null object
fgroup          6636 non-null object
id              6636 non-null int64
manufacturer    5195 non-null object
dtypes: int64(1), object(3)
memory usage: 207.5+ KB

nutrients = []

for rec in db:
  fnuts = pd.DataFrame(rec['nutrients'])
  fnuts['id'] = rec['id']
  nutrients.append(fnuts)
nutrients = pd.concat(nutrients,ignore_index = True)

In [173]: col_mapping = {'description' : 'nutrient',
```

```
   .....:                'group' : 'nutgroup'}
In [174]: nutrients = nutrients.rename(columns=col_mapping, copy=False)

In [175]: nutrients
Out[175]:
                                  nutrient     nutgroup units    value     id
0                                  Protein  Composition     g   25.180   1008
1                        Total lipid (fat)  Composition     g   29.200   1008
2              Carbohydrate, by difference  Composition     g    3.060   1008
3                                      Ash        Other     g    3.280   1008
4                                   Energy       Energy  kcal  376.000   1008
...                                    ...          ...   ...      ...    ...
389350                Vitamin B-12, added     Vitamins   mcg    0.000  43546
389351                         Cholesterol        Other    mg    0.000  43546
389352        Fatty acids, total saturated        Other     g    0.072  43546
389353  Fatty acids, total monounsaturated        Other     g    0.028  43546
389354  Fatty acids, total polyunsaturated        Other     g    0.041  43546
[375176 rows x 5 columns]
```

将info跟nutrients合并起来

```
In [176]: ndata = pd.merge(nutrients, info, on='id', how='outer')

In [177]: ndata.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 375176 entries, 0 to 375175
Data columns (total 8 columns):
nutrient        375176 non-null object
nutgroup        375176 non-null object
units           375176 non-null object
value           375176 non-null float64
id              375176 non-null int64
food            375176 non-null object
fgroup          375176 non-null object
manufacturer    293054 non-null object
dtypes: float64(1), int64(1), object(6)
```

```
memory usage: 25.8+ MB


In [178]: ndata.iloc[30000]
Out[178]:
nutrient                                        Glycine
nutgroup                                   Amino Acids
units                                                g
value                                             0.04
id                                                6158
food            Soup, tomato bisque, canned, condensed
fgroup                       Soups, Sauces, and Gravies
manufacturer
Name: 30000, dtype: object
```

根据食物分类和营养类型画出一张中位值图

```
In [180]: result = ndata.groupby(['nutrient', 'fgroup'])['value'].quantile(0.5)


In [181]: result['Zinc, Zn'].sort_values().plot(kind='barh')
```

2012联邦选举委员会数据库

美国联邦选举委员会发布了有关政治竞选赞助方面的数据。其中包括赞助者的姓名、职业、雇主、地址以及出资额等信息

```
In [184]: fec = pd.read_csv('datasets/fec/P00000001-ALL.csv')


In [185]: fec.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1001731 entries, 0 to 1001730
Data columns (total 16 columns):
cmte_id             1001731 non-null object
cand_id             1001731 non-null object
cand_nm             1001731 non-null object
contbr_nm           1001731 non-null object
contbr_city         1001712 non-null object
contbr_st           1001727 non-null object
```

```
contbr_zip           1001620 non-null object
contbr_employer      988002 non-null object
contbr_occupation    993301 non-null object
contb_receipt_amt    1001731 non-null float64
contb_receipt_dt     1001731 non-null object
receipt_desc         14166 non-null object
memo_cd              92482 non-null object
memo_text            97770 non-null object
form_tp              1001731 non-null object
file_num             1001731 non-null int64
dtypes: float64(1), int64(1), object(14)
memory usage: 122.3+ MB
```

```
In [186]: fec.iloc[123456]
Out[186]:
cmte_id             C00431445
cand_id             P80003338
cand_nm         Obama, Barack
contbr_nm         ELLMAN, IRA
contbr_city             TEMPE
                      ...
receipt_desc              NaN
memo_cd                   NaN
memo_text                 NaN
form_tp                 SA17A
file_num               772372
Name: 123456, Length: 16, dtype: object
```

通过unique，你可以获取全部的候选人名单

```
In [187]: unique_cands = fec.cand_nm.unique()

In [188]: unique_cands
Out[188]:
array(['Bachmann, Michelle', 'Romney, Mitt', 'Obama, Barack',
       "Roemer, Charles E. 'Buddy' III", 'Pawlenty, Timothy',
```

```
       'Johnson, Gary Earl', 'Paul, Ron', 'Santorum, Rick', 'Cain, Herman',
       'Gingrich, Newt', 'McCotter, Thaddeus G', 'Huntsman, Jon',
       'Perry, Rick'], dtype=object)

In [189]: unique_cands[2]
Out[189]: 'Obama, Barack'
```

指明党派信息的方法之一是使用字典

```
parties = {'Bachmann, Michelle': 'Republican',
           'Cain, Herman': 'Republican',
           'Gingrich, Newt': 'Republican',
           'Huntsman, Jon': 'Republican',
           'Johnson, Gary Earl': 'Republican',
           'McCotter, Thaddeus G': 'Republican',
           'Obama, Barack': 'Democrat',
           'Paul, Ron': 'Republican',
           'Pawlenty, Timothy': 'Republican',
           'Perry, Rick': 'Republican',
           "Roemer, Charles E. 'Buddy' III": 'Republican',
           'Romney, Mitt': 'Republican',
           'Santorum, Rick': 'Republican'}
```

根据候选人姓名得到一组党派信息

```
In [191]: fec.cand_nm[123456:123461]
Out[191]:
123456    Obama, Barack
123457    Obama, Barack
123458    Obama, Barack
123459    Obama, Barack
123460    Obama, Barack
Name: cand_nm, dtype: object

In [192]: fec.cand_nm[123456:123461].map(parties)
Out[192]:
123456    Democrat
```

```
123457    Democrat
123458    Democrat
123459    Democrat
123460    Democrat
Name: cand_nm, dtype: object

# Add it as a column
In [193]: fec['party'] = fec.cand_nm.map(parties)

In [194]: fec['party'].value_counts()
Out[194]:
Democrat      593746
Republican    407985
Name: party, dtype: int64
```

```
In [195]: (fec.contb_receipt_amt > 0).value_counts()
Out[195]:
True     991475
False     10256
Name: contb_receipt_amt, dtype: int64
```

只包含Barack Obama和Mitt Romney竞选活动的赞助信息

```
In [197]: fec_mrbo = fec[fec.cand_nm.isin(['Obama, Barack','Romney, Mitt'])]
```

根据职业和雇主统计赞助信息

基于职业的赞助信息统计是另一种经常被研究的统计任务

```
In [198]: fec.contbr_occupation.value_counts()[:10]
Out[198]:
RETIRED                                   233990
INFORMATION REQUESTED                      35107
ATTORNEY                                   34286
HOMEMAKER                                  29931
```

```
PHYSICIAN                                  23432
INFORMATION REQUESTED PER BEST EFFORTS     21138
ENGINEER                                   14334
TEACHER                                    13990
CONSULTANT                                 13273
PROFESSOR                                  12555
Name: contbr_occupation, dtype: int64
```

对职业信息进行清理

```
occ_mapping = {
   'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
   'INFORMATION REQUESTED' : 'NOT PROVIDED',
   'INFORMATION REQUESTED (BEST EFFORTS)' : 'NOT PROVIDED',
   'C.E.O.': 'CEO'
}

# If no mapping provided, return x
f = lambda x: occ_mapping.get(x, x)
fec.contbr_occupation = fec.contbr_occupation.map(f)
```

对雇主信息进行清理

```
emp_mapping = {
   'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
   'INFORMATION REQUESTED' : 'NOT PROVIDED',
   'SELF' : 'SELF-EMPLOYED',
   'SELF EMPLOYED' : 'SELF-EMPLOYED',
}

# If no mapping provided, return x
f = lambda x: emp_mapping.get(x, x)
fec.contbr_employer = fec.contbr_employer.map(f)
```

通过pivot_table根据党派和职业对数据进行聚合，然后过滤掉总出资额不足200万美元的数据

```
```

```
In [201]: by_occupation = fec.pivot_table('contb_receipt_amt',
   .....:                                 index='contbr_occupation',
   .....:                                 columns='party', aggfunc='sum')

In [202]: over_2mm = by_occupation[by_occupation.sum(1) > 2000000]

In [203]: over_2mm
Out[203]:
party                 Democrat    Republican
contbr_occupation
ATTORNEY           11141982.97  7.477194e+06
CEO                 2074974.79  4.211041e+06
CONSULTANT          2459912.71  2.544725e+06
ENGINEER             951525.55  1.818374e+06
EXECUTIVE           1355161.05  4.138850e+06
...                        ...           ...
PRESIDENT           1878509.95  4.720924e+06
PROFESSOR           2165071.08  2.967027e+05
REAL ESTATE          528902.09  1.625902e+06
RETIRED            25305116.38  2.356124e+07
SELF-EMPLOYED        672393.40  1.640253e+06
[17 rows x 2 columns]
```

```
In [205]: over_2mm.plot(kind='barh')
```

对Obama和Romney总出资额最高的职业和企业

```
def get_top_amounts(group, key, n=5):
    totals = group.groupby(key)['contb_receipt_amt'].sum()
    return totals.nlargest(n)
```

根据职业和雇主进行聚合

```
In [207]: grouped = fec_mrbo.groupby('cand_nm')
```

```
In [208]: grouped.apply(get_top_amounts, 'contbr_occupation', n=7)
Out[208]:
cand_nm        contbr_occupation
Obama, Barack  RETIRED                   25305116.38
               ATTORNEY                  11141982.97
               INFORMATION REQUESTED      4866973.96
               HOMEMAKER                  4248875.80
               PHYSICIAN                  3735124.94
                                            ...
Romney, Mitt   HOMEMAKER                  8147446.22
               ATTORNEY                   5364718.82
               PRESIDENT                  2491244.89
               EXECUTIVE                  2300947.03
               C.E.O.                     1968386.11
Name: contb_receipt_amt, Length: 14, dtype: float64

In [209]: grouped.apply(get_top_amounts, 'contbr_employer', n=10)
Out[209]:
cand_nm        contbr_employer
Obama, Barack  RETIRED                   22694358.85
               SELF-EMPLOYED             17080985.96
               NOT EMPLOYED               8586308.70
               INFORMATION REQUESTED      5053480.37
               HOMEMAKER                  2605408.54
                                            ...
Romney, Mitt   CREDIT SUISSE               281150.00
               MORGAN STANLEY              267266.00
               GOLDMAN SACH & CO.          238250.00
               BARCLAYS CAPITAL            162750.00
               H.I.G. CAPITAL              139500.00
Name: contb_receipt_amt, Length: 20, dtype: float64
```

对出资额分组

利用cut函数根据出资额的大小将数据离散化到多个面元中

```
In [210]: bins = np.array([0, 1, 10, 100, 1000, 10000,
   .....:                  100000, 1000000, 10000000])
```

```
In [211]: labels = pd.cut(fec_mrbo.contb_receipt_amt, bins)

In [212]: labels
Out[212]:
411         (10, 100]
412       (100, 1000]
413       (100, 1000]
414         (10, 100]
415         (10, 100]
             ...
701381      (10, 100]
701382    (100, 1000]
701383        (1, 10]
701384      (10, 100]
701385    (100, 1000]
Name: contb_receipt_amt, Length: 694282, dtype: category
Categories (8, interval[int64]): [(0, 1] < (1, 10] < (10, 100] < (100, 1000] <
(1
000, 10000] <
                                  (10000, 100000] < (100000, 1000000] <
(1000000,
 10000000]]
```

根据候选人姓名以及面元标签对奥巴马和罗姆尼数据进行分组，以得到一个柱状图

```
In [213]: grouped = fec_mrbo.groupby(['cand_nm', labels])

In [214]: grouped.size().unstack(0)
Out[214]:
cand_nm              Obama, Barack  Romney, Mitt
contb_receipt_amt
(0, 1]                       493.0          77.0
(1, 10]                    40070.0        3681.0
(10, 100]                 372280.0       31853.0
(100, 1000]               153991.0       43357.0
(1000, 10000]              22284.0       26186.0
(10000, 100000]                2.0           1.0
```

```
(100000, 1000000]              3.0           NaN
(1000000, 10000000]            4.0           NaN
```

从这个数据中可以看出，在小额赞助方面，Obama获得的数量比Romney多得多, 还可以对出资额求和并在面元内规格化，以便图形化显示两位候选人各种赞助额度的比例

```
In [216]: bucket_sums = grouped.contb_receipt_amt.sum().unstack(0)

In [217]: normed_sums = bucket_sums.div(bucket_sums.sum(axis=1), axis=0)

In [218]: normed_sums
Out[218]:
cand_nm              Obama, Barack  Romney, Mitt
contb_receipt_amt
(0, 1]                    0.805182      0.194818
(1, 10]                   0.918767      0.081233
(10, 100]                 0.910769      0.089231
(100, 1000]               0.710176      0.289824
(1000, 10000]             0.447326      0.552674
(10000, 100000]           0.823120      0.176880
(100000, 1000000]         1.000000           NaN
(1000000, 10000000]       1.000000           NaN

In [219]: normed_sums[:-2].plot(kind='barh')
```