14. 데이터 분석 예제

실제 데이터셋을 살펴본다. 여기서 설명하는 기술은 데이터셋을 포함하여 모든 데이터셋에 적용할 수 있을 것이다.

14.1 Bit.ly의 1.USA.gov 데이터

매 시간별 스냅샷 파일의 각 로우는 웹 데이터 형식으로 흔히 사용되는 JSON이다. 스냅샷 파일의 첫 줄을 열어보면 다음과 비슷한 내용을 확인할 수 있다.

```
In [226]: path = 'datasets/bitly_usagov/example.txt'

In [227]: open(path).readline()
Out[227]: '{ "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'
```

파이썬에는 JSON 문자열을 파이썬 사전 객체로 바꿔주는 다양한 내장 모듈과 서드파티 모듈이 있다. 여기서는 json 모듈의 loads 함수를 이용해서 내려받은 샘플 파일을 한 줄씩 읽는다.

```
In [240]: import json

In [241]: path = 'C:/Users/김대현/Desktop/data/파라데/pydata-book-2nd-e
     ...: dition/datasets/bitly_usagov/example_ansi.txt'

In [242]: records = [json.loads(line) for line in open(path)]

In [243]: records[0]
Out[243]:
{'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]}
```

14.1.1 순수 파이썬으로 표준시간대 세어보기

이 데이터에서 가장 빈도가 높은 표준시간대(tz 필드)를 구한다고 가정하자. 다양한 방법이 있지만 먼저 리스트 표기법을 사용해서 표준시간대의 목록을 가져오자.

```
In [244]: time_zones = [rec['tz'] for rec in records]
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-244-f3fbbc37f129> in <module>
----> 1 time_zones = [rec['tz'] for rec in records]

<ipython-input-244-f3fbbc37f129> in <listcomp>(.0)
----> 1 time_zones = [rec['tz'] for rec in records]

KeyError: 'tz'
```

하지만 records의 아이템이 모드 표준시간대 필드가 가지고 있는 건 아니라는 게 드러났다. 이 문제는 if 'tz' in rec을 리스트 표기법 뒤에 추가해서 tz 필드가 있는지 검사하면 쉽게 해결 할 수 있다.

```
In [245]: time_zones = [rec['tz'] for rec in records if 'tz' in rec]

In [246]: time_zones[:10]
Out[246]:
['America/New_York',
 'America/Denver',
 'America/New_York',
 'America/Sao_Paulo',
 'America/New_York',
 'America/New_York',
 'Europe/Warsaw',
 '',
 '',
 '']
```

상위 10개의 표준시간대를 보면 그중 몇 개는 비어있어서 뭔지 알 수 없다. 비어 있는 필드를 제거할 수도 있지만 일단은 그냥 두고 표준시간대를 세어보자.

```
In [247]: def get_counts(sequence):
     ...:     counts = {}
     ...:     for x in sequence:
     ...:         if x in counts:
     ...:             counts[x] += 1
     ...:         else:
     ...:             counts[x] = 1
     ...:     return counts
     ...:
```

파이썬 표준 라이브러리에 익숙하다면 다음처럼 좀 더 간단하게 작성할 수도 있다.

```
In [248]: from collections import defaultdict

In [249]: def get_counts2(sequence):
     ...:     counts = defaultdict(int) #값이 0으로 초기화 된다.
     ...:     for x in sequence:
     ...:         counts[x] += 1
     ...:     return counts
```

재사용이 쉽도록 이 로직을 함수로 만들고 이 함수에 time_zone 리스트를 넘겨서 사용하자.

```
In [250]: counts = get_counts(time_zones)

In [251]: counts['America/New_York']
Out[251]: 1251

In [252]: len(time_zones)
Out[252]: 3440
```

가장 많이 등장하는 상위 10개의 표준시간대를 알고 싶다면 좀 더 세련된 방법으로 사전을 사용하면 된다.

```
In [253]: def top_counts(count_dict, n=10):
     ...:     value_key_pairs = [(count,tz) for tz, count in count_dict
     ...: .items()]
     ...:     return value_key_pairs[-n:]
     ...:
```

이제 상위 10개의 표준시간대를 구했다.

```
In [256]: top_counts(counts)
Out[256]:
[(1, 'America/St_Kitts'),
 (11, 'Pacific/Auckland'),
 (1, 'America/Santo_Domingo'),
 (1, 'America/Argentina/Cordoba'),
 (1, 'Asia/Kuching'),
 (1, 'Europe/Volgograd'),
 (1, 'America/La_Paz'),
 (1, 'Africa/Casablanca'),
 (3, 'Asia/Jakarta'),
 (1, 'America/Tegucigalpa')]
```

파이썬 표준 라이브러리의 collections.Counter 클래스를 이용하면 지금까지 했던 작업을 훨씬 쉽게 할 수 있다.

```
In [257]: from collections import Counter

In [258]: counts = Counter(time_zones)

In [259]: counts.most_common(10)
Out[259]:
[('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)]
```

14.1.2 pandas로 표준시간대 세어보기

records를 가지고 DataFrame을 만드는 방법은 아주 쉽다. 그냥 레코드가 담긴 리스트를 pandas.DataFrame으로 넘기면 된다.

```
In [261]: frame = pd.DataFrame(records)

In [262]: frame
Out[262]:
                                                      a  ...   kw
0     Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKi...  ...  NaN
1                                GoogleMaps/RochesterNY  ...  NaN
2     Mozilla/4.0 (compatible; MSIE 8.0; Windows NT ...  ...  NaN
3     Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8)...  ...  NaN
4     Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKi...  ...  NaN
...                                                 ...  ...  ...
3555  Mozilla/4.0 (compatible; MSIE 9.0; Windows NT ...  ...  NaN
3556  Mozilla/5.0 (Windows NT 5.1) AppleWebKit/535.1...  ...  NaN
3557                             GoogleMaps/RochesterNY  ...  NaN
3558                                     GoogleProducer  ...  NaN
3559  Mozilla/4.0 (compatible; MSIE 8.0; Windows NT ...  ...  NaN

[3560 rows x 18 columns]

In [263]: frame.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3560 entries, 0 to 3559
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   a            3440 non-null   object
 1   c            2919 non-null   object
 2   nk           3440 non-null   float64
 3   tz           3440 non-null   object
 4   gr           2919 non-null   object
 5   g            3440 non-null   object
 6   h            3440 non-null   object
 7   l            3440 non-null   object
 8   al           3094 non-null   object
 9   hh           3440 non-null   object
 10  r            3440 non-null   object
 11  u            3440 non-null   object
 12  t            3440 non-null   float64
 13  hc           3440 non-null   float64
 14  cy           2919 non-null   object
 15  ll           2919 non-null   object
 16  _heartbeat_  120 non-null    float64
 17  kw           93 non-null     object
dtypes: float64(4), object(14)
```

```
memory usage: 500.8+ KB

In [270]: frame.columns
Out[270]:
Index(['a', 'c', 'nk', 'tz', 'gr', 'g', 'h', 'l', 'al', 'hh', 'r', 'u', 't',
       'hc', 'cy', 'll', '_heartbeat_', 'kw'],
      dtype='object')


In [271]: frame['tz'][:10]
Out[271]:
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
```

frame의 출력 결과는 거대한 DataFrame 객체의 요약 정보다. frame['tz'] 에서 반환되는 Series 객체에는 value_counts 메서드를 이용해서 시간대를 세어볼 수 있다.

```
In [272]: tz_counts = frame['tz'].value_counts()

In [273]: tz_counts[:10]
Out[273]:
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
```

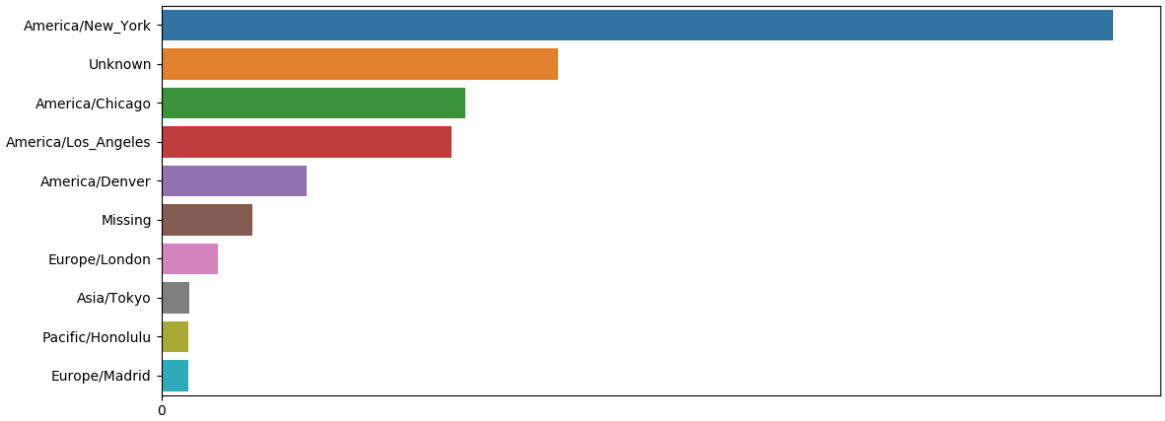
matplotlib 라이브러리로 이 데이터를 그래프로 그릴 수 있다. 그전에 records 에서 비어 있는 표준시간대를 다른 이름으로 바꿔보자. fillna 함수로 빠진 값을 대체하고, 불리언 배열 색인을 이용해서 비어 있는 값을 대체할 수 있다.

```
In [274]: clean_tz = frame['tz'].fillna('Missing')

In [275]: clean_tz[clean_tz == ''] = 'Unknown'

In [276]: tz_counts = clean_tz.value_counts()

In [277]: tz_counts[:10]
Out[277]:
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
```

여기서는 seaborn 패키지를 이용해서 수평막대그래프를 그려보자.

```
In [278]: import seaborn as sns

In [279]: subset = tz_counts[:10]

In [280]: sns.barplot(y=subset.index, x=subset.values)
Out[280]: <matplotlib.axes._subplots.AxesSubplot at 0x1e935d1ef88>

In [295]: import matplotlib.pyplot as plt

In [296]: sns.barplot(y=subset.index, x=subset.values)
Out[296]: <matplotlib.axes._subplots.AxesSubplot at 0x1e935d1ef88>

In [297]: plt.show()
```



a 필에는 URL 단축을 실행하는 브라우저, 단말기, 애플리케이션에 대한 정보(User Agent 문자열)가 들어 있다.

```
In [298]: frame['a'][1]
Out[298]: 'GoogleMaps/RochesterNY'

In [299]: frame['a'][50]
Out[299]: 'Mozilla/5.0 (Windows NT 5.1; rv:10.0.2) Gecko/20100101 Firefox/10.0.2'

In [300]: frame['a'][51][:50] #긴 문자열
Out[300]: 'Mozilla/5.0 (Linux; U; Android 2.2.2; en-us; LG-P9'

In [301]: frame['a'][51]
Out[301]: '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/533.1'
```

'agent' 라고 하는 흥미로운 문자열 정보를 분석하는 일이 어려워 보일 수도 있다. 한 가지 가능한 전략은 문자열에서 첫 번째 토큰(브라우저의 종류를 어느 정도 알 수 있을 만큼)을 잘라내서 사용자 행동에 대한 또 다른 개요를 만드는 것이다.

```
In [302]: results = pd.Series([x.split()[0] for x in frame.a.dropna()])
     ...:

In [303]: results[:5]
Out[303]:
0               Mozilla/5.0
1    GoogleMaps/RochesterNY
2               Mozilla/4.0
3               Mozilla/5.0
4               Mozilla/5.0
dtype: object

In [305]: results.value_counts()[:8]
Out[305]:
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
```

이제 표준시간대 순위표를 윈도우 사용자와 비윈도우 사용자 그룹으로 나눠보자. 문제를 단순화해서 agent 문자열이 'Windows'를 포함하면 윈도우 사용자라고 가정하고 agent 값이 없는 데이터는 다음과 같이 제외한다.

```
In [306]: cframe = frame[frame.a.notnull()]

In [307]: cframe
Out[307]:
                                                      a  ...   kw
0     Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKi...  ...  NaN
1                                GoogleMaps/RochesterNY  ...  NaN
2     Mozilla/4.0 (compatible; MSIE 8.0; Windows NT ...  ...  NaN
3     Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8)...  ...  NaN
4     Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKi...  ...  NaN
...                                                 ...  ...  ...
3555  Mozilla/4.0 (compatible; MSIE 9.0; Windows NT ...  ...  NaN
3556  Mozilla/5.0 (Windows NT 5.1) AppleWebKit/535.1...  ...  NaN
3557                             GoogleMaps/RochesterNY  ...  NaN
3558                                     GoogleProducer  ...  NaN
3559  Mozilla/4.0 (compatible; MSIE 8.0; Windows NT ...  ...  NaN

[3440 rows x 18 columns]
```

그리고 이제 각 로우가 윈도우인지 아닌지 검사한다.

```
In [308]: cframe['os'] = np.where(cframe['a'].str.contains('Windows'),'
     ...: Windows','Not Windows')
C:\ProgramData\Anaconda3\Scripts\ipython:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
In [309]: cframe['os'][:5]
Out[309]:
0        Windows
1    Not Windows
2        Windows
3    Not Windows
4        Windows
Name: os, dtype: object
```

그런 다음 표준시간대와 운영체제를 기준으로 데이터를 그룹으로 묶는다.

```
In [310]: by_tz_os = cframe.groupby(['tz','os'])
```

앞에서 살펴본 value_count 함수처럼 그룹별 합계는 size 함수로 계산할 수 없다. 결과는 unstack 함수를 이용해서 표로 재배치한다.

```
In [311]: agg_counts = by_tz_os.size().unstack().fillna(0)

In [312]: agg_counts[:10]
Out[312]:
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
```

마지막으로 전체 표준시간대의 순위를 모아보자. 먼저 agg_counts를 보자.

```
#오름차순
In [313]: indexer = agg_counts.sum(1).argsort()

In [314]: indexer[:10]
Out[314]:
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
```

agg_counts에 take를 사용해서 로우를 정렬된 순서 그대로 선택하고 마지막 10개 로우(가장 큰 값) 만 잘라낸다.

```
In [317]: counts_subset = agg_counts.take(indexer[-10:])

In [318]: counts_subset
Out[318]:
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
```

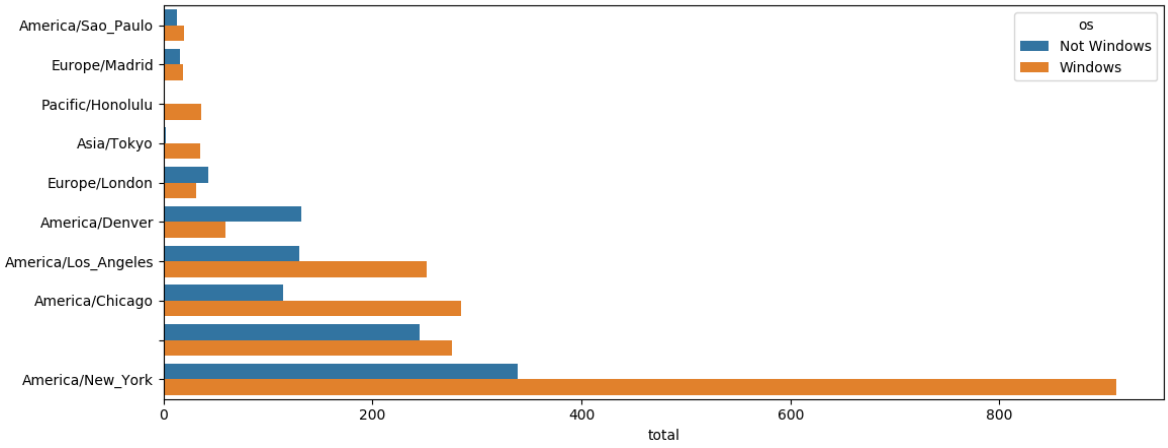
pandas에는 이와 똑같은 동작을 하는 nlargest 라는 편리한 메서드가 존재한다.

```
In [319]: agg_counts.sum(1).nlargest(10)
Out[319]:
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
```

그런 다음 앞에서 해본 것처럼 plot 함수에 stacked = True를 넘겨주면 데이터를 중첩막대그래프로 만들 수 있다.

```
In [328]: count_subset = counts_subset.stack()

In [329]: count_subset.name = 'total'


In [331]: count_subset = count_subset.reset_index()
```

```
In [332]: count_subset[:10]
Out[332]:
                  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 [333]: sns.barplot(x='total',y='tz',hue='os',data = count_subset)
Out[333]: <matplotlib.axes._subplots.AxesSubplot at 0x1e93ad1f848>
```

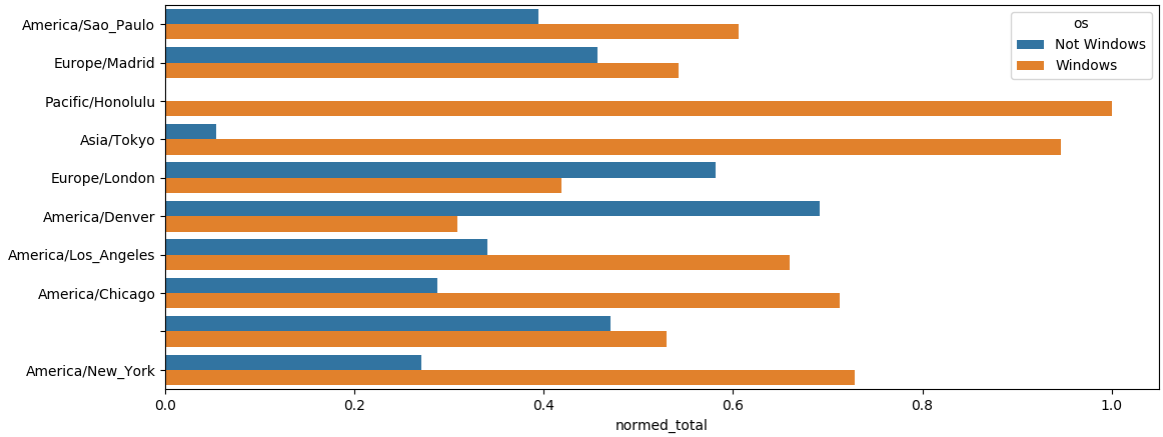


위 그래프로는 작은 그룹에서 윈도우 사용자의 상대 비율을 확인하기 어렵다. 하지만 각 로우에서 총합을 1로 정규화한 뒤 그래프를 그리면 쉽게 확인할 수 있다.

```
In [335]: def norm_total(group):
     ...:     group['normed_total'] = group.total / group.total.sum()
     ...:     return group
     ...:

In [336]: results = count_subset.groupby('tz').apply(norm_total)
```

정규화한 데이터를 그래프로 그려보자.

```
In [346]: sns.barplot(x='normed_total',y='tz',hue='os',data = results)
Out[346]: <matplotlib.axes._subplots.AxesSubplot at 0x1e93c03d7c8>
```



groupby와 transform 메서드를 이용해서 정규합 계산을 더 효율적으로 할 수도 있다.

```
In [348]: g = count_subset.groupby('tz')

In [349]: results2 = count_subset.total/ g.total.transform('sum')
```

14.2 MovieLens의 영화 평점 데이터

MovieLens LM(백만 개) 데이터셋은 약 6,000여 명의 사용자로부터 수집한 4,000여 편의 영화에 대한 백만 개의 영화 평점을 담고 있다. 이 데이터셋은 평점, 사용자 정보, 영화 정보의 3가지 테이블로 나뉘어 있는데, zip 파일의 압축을 풀고 각 테이블을 pandas.read_table 함수를 사용하여 DateFrame 객체로 불러오자.

```
In [1]: import pandas as pd

In [2]: pd.options.display.max_rows = 10

In [3]: unames = ['user_id', 'gender', 'age', 'occupation', 'zip']

In [4]: cd C:\Users\김대현\Desktop\data\밑시딥_1\deep-learning-from-scr
   ...: atch-master\deep-learning-from-scratch-master
C:\Users\김대현\Desktop\data\밑시딥_1\deep-learning-from-scratch-master
    \deep-learning-from-scratch-master

In [9]: users = pd.read_table('datasets/movielens/users.dat',sep='::',h
   ...: eader = None, names = unames)
C:\ProgramData\Anaconda3\Scripts\ipython:1: ParserWarning:
                Falling back to the 'python' engine because the
                'c' engine does not support regex separators
                (separators > 1 char and different from '\s+'
                 are interpreted as regex);
                you can avoid this warning by specifying engine='python'.
In [12]: mnames = [ 'movie_id', 'title', 'genres']

In [13]: movies = pd.read_table('datasets/movielens/movies.dat',sep='::
    ...: ',header = None, names = mnames)
C:\ProgramData\Anaconda3\Scripts\ipython:1: ParserWarning:
Falling back to the 'python' engine because the 'c' engine does not support
regex separators (separators > 1 char and different from '\s+' are
interpreted as regex); you can avoid this warning by specifying
                               engine='python'.
```

DataFrame 객체에 데이터가 제대로 들어갔는지 확인하기 위해 파이썬의 리스트 분할 문법을 사용해서 첫 5개 로우를 출력해보자.

```
In [15]: users[:5]
Out[15]:
   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 [16]: ratings[:5]
Out[16]:
   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 [17]: movies[:5]
Out[17]:
   movie_id  ...                        genres
0         1  ...   Animation|Children's|Comedy
1         2  ...  Adventure|Children's|Fantasy
2         3  ...                Comedy|Romance
3         4  ...                  Comedy|Drama
4         5  ...                        Comedy

[5 rows x 3 columns]
```

나이와 직업은 실제값이 아닌 그룹을 가리키는 코드 번호이며 데이터셋에 있는 README 파일에 해당 코드에 대한 설명이 들어있다. 세 종류의 테이블에 걸쳐 있는 데이터를 분석하는 일은 단순한 작업이 아니다. 나이와 성별에 따른 어떤 영화의 평균 평점을 계산한다고 해보자. 모든 데이터를 하나의 테이블로 병합하여 계산하면 편리하겠다.

pandas의 merge 함수를 이용해서 ratings 테이블과 users 테이블을 병합하고 그 결과를 다시 movies 테이블과 병합한다. pandas는 병합하려는 두 테이블에서 중복되는 컬럼의 이름을 키로 사용한다.

```
In [19]: data = pd.merge(pd.merge(ratings,users), movies)

In [24]: data
Out[24]:
         user_id  movie_id  rating  timestamp gender  age  occupation    zip
title                genres
0              1      1193       5  978300760      F    1          10  48067
One Flew Over the Cuckoo's Nest (1975)                 Drama
1              2      1193       5  978298413      M   56          16  70072
One Flew Over the Cuckoo's Nest (1975)                 Drama
2             12      1193       4  978220179      M   25          12  32793
One Flew Over the Cuckoo's Nest (1975)                 Drama
3             15      1193       4  978199279      M   25           7  22903
One Flew Over the Cuckoo's Nest (1975)                 Drama
4             17      1193       5  978158471      M   50           1  95350
One Flew Over the Cuckoo's Nest (1975)                 Drama
```

```
...          ...       ...     ...        ...    ...  ...          ...    ...
...                   ...
1000204     5949      2198       5  958846401      M   18          17  47901
Modulations (1998)           Documentary
1000205     5675      2703       3  976029116      M   35          14  30030
Broken Vessels (1998)                 Drama
1000206     5780      2845       1  958153068      M   18          17  92886
White Boys (1999)                  Drama
1000207     5851      3607       5  957756608      F   18          20  55410
One Little Indian (1973)  Comedy|Drama|Western
1000208     5938      2909       4  957273353      M   25           1  35401
Five Wives, Three Secretaries and Me (1998)           Documentary

[1000209 rows x 10 columns]

In [25]: data.iloc[0]
Out[25]:
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
```

성별에 따른 각 영화의 평균 평점을 구하려면 pivot_table 메서드를 사용하면 된다.

```
In [26]: mean_ratings = data.pivot_table('rating', index ='title',
    ...:                                  columns = 'gender', aggfunc =
    ...: 'mean')

In [34]: mean_ratings[:5]
Out[34]:
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
```

이렇게 하면 매 로우마다 성별에 따른 평균 영화 평점 정보를 담고 있는 DataFrame 객체가 만들어 진다. 먼저 250건 이상의 평점 정보가 있는 영화만 추려보자. 데이터를 영화 제목으로 그룹화하고 size() 함수를 사용해서 제목별 평점 정보 건수를 Series 객체로 얻어낸다.

```
In [35]: ratings_by_title = data.groupby('title').size()

In [36]: ratings_by_title[:10]
Out[36]:
title
$1,000,000 Duck (1971)               37
'Night Mother (1986)                 70
```

```
'Til There Was You (1997)              52
'burbs, The (1989)                    303
...And Justice for All (1979)         199
1-900 (1994)                            2
10 Things I Hate About You (1999)     700
101 Dalmatians (1961)                 565
101 Dalmatians (1996)                 364
12 Angry Men (1957)                   616

In [37]: active_titles = ratings_by_title.index[ratings_by_title>=250]

In [38]: active_titles
Out[38]:
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)
```

250건 이상의 평점 정보가 있는 영화에 대한 색인은 mean_ratings에서 항목을 선택하기 위해 사용할 수 있다.

```
In [39]: mean_ratings = mean_ratings.loc[active_titles]

In [40]: mean_ratings
Out[40]:
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]
```

여성에게 높은 평점을 받은 영화 목록을 확인하기 위해 F 컬럼을 내림차순으로 정렬한다.

```
In [41]: top_female_ratings = mean_ratings.sort_values(by='F', ascendin
    ...: g = False)

In [42]: top_female_ratings
Out[42]:
```

```
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
...                                                        ...       ...
Avengers, The (1998)                                  1.915254  2.017467
Speed 2: Cruise Control (1997)                        1.906667  1.863014
Rocky V (1990)                                        1.878788  2.132780
Barb Wire (1996)                                      1.585366  2.100386
Battlefield Earth (2000)                              1.574468  1.616949

[1216 rows x 2 columns]
```

14.2.1 평점 차이 구하기

이번엔 남녀 간의 호불호가 갈리는 영화를 찾아보자. mean_ratings에 평균 평점의 차이를 담을 수 있는 컬럼을 하나 추가하고, 그 컬럼을 기준으로 정렬하자.

```
In [48]: mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']

In [49]: mean_ratings
Out[49]:
gender                                     F         M      diff
title
'burbs, The (1989)                  2.793478  2.962085  0.168607
10 Things I Hate About You (1999)   3.646552  3.311966 -0.334586
101 Dalmatians (1961)               3.791444  3.500000 -0.291444
101 Dalmatians (1996)               3.240000  2.911215 -0.328785
12 Angry Men (1957)                 4.184397  4.328421  0.144024
...                                      ...       ...       ...
Young Guns (1988)                   3.371795  3.425620  0.053825
Young Guns II (1990)                2.934783  2.904025 -0.030758
Young Sherlock Holmes (1985)        3.514706  3.363344 -0.151362
Zero Effect (1998)                  3.864407  3.723140 -0.141266
eXistenZ (1999)                     3.098592  3.289086  0.190494

[1216 rows x 3 columns]

In [50]: sorted_by_diff = mean_ratings.sort_values(by = 'diff')

In [51]: sorted_by_diff
Out[51]:
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
...                                       ...       ...       ...
Cable Guy, The (1996)                2.250000  2.863787  0.613787
Longest Day, The (1962)              3.411765  4.031447  0.619682
Dumb & Dumber (1994)                 2.697987  3.336595  0.638608
Kentucky Fried Movie, The (1977)     2.878788  3.555147  0.676359
```

```
Good, The Bad and The Ugly, The (1966)  3.494949  4.221300  0.726351

[1216 rows x 3 columns]
```

역순으로 정렬한 다음 상위 10개 로우를 잘라내면 남성의 선호도 순으로 확인할 수 있다.

```
In [52]: sorted_by_diff[::-1][:10]
Out[52]:
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
```

성별에 관계없이 영화에 대한 호불호가 극명하게 나뉘는 영화를 찾아보자. 호불호는 평점의 분산이나 표준편차로 측정할 수 있다.

```
In [53]: rating_std_by_title = data.groupby('title')['rating'].std()

In [54]: rating_std_by_title = rating_std_by_title.loc[active_titles]

In [55]: rating_std_by_title.sort_values(ascending = False)[:10]
Out[55]:
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
```

14.3 신생아 이름

신생아 이름으로 부터 여러 분석을 할 수 있다.

- **시대별로 특정 이름이 차지하는 비율을 구해 얼마나 흔한 이름인지 알아보기**
- **이름의 상대 순위 알아보기**
- **각 연도별로 가장 인기 있는 이름, 가장 많이 증가하거나 감소한 이름 알아보기**
- **모음, 자음, 길이, 전체 다양성, 철자 변화, 첫 글자와 마지막 글자 등 이름 유행 분석하기**
- **성서에 등장하는 이름, 유명인, 인구통계학적 변화 등 외부 자료를 통한 유행 분석**

```
In [75]: names1880 = pd.read_csv('datasets/babynames/yob1880.txt',names
```

```
    ...:  = ['names', 'sex', 'births'])

In [76]: names1880
Out[76]:
          names 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]
```

이 데이터는 각 연도별로 최소 5명이상 중복되는 이름만 포함하고 있다.

```
In [77]: names1880.groupby('sex').sum()
Out[77]:
     births
sex
F     90993
M    110493
```

자료가 연도별 파일로 나뉘어져 있으니 먼저 모든 데이터를 DataFrame 하나로 모든 다음 year 항목을 추가한다. pandas.concat을 이용하면 이 작업을 쉽게 할 수 있다.

```
In [78]: years = range(1880,2011)

In [79]: pieces = []

In [80]: columns = ['name', 'sex', 'births']

In [81]: for year in years:
    ...:     path = 'datasets/babynames/yob%d.txt' % year
    ...:     frame = pd.read_csv(path, names = columns)
    ...:
    ...:     frame['year'] = year
    ...:     pieces.append(frame)
    ...:
        #모두 하나의 DataFrame으로 합치기
    ...: names = pd.concat(pieces, ignore_index = True)
```

여기서 두 가지 언급해야 할 내용이 있다. 첫째, concat 메서드는 DataFrame 객체를 합쳐준다.

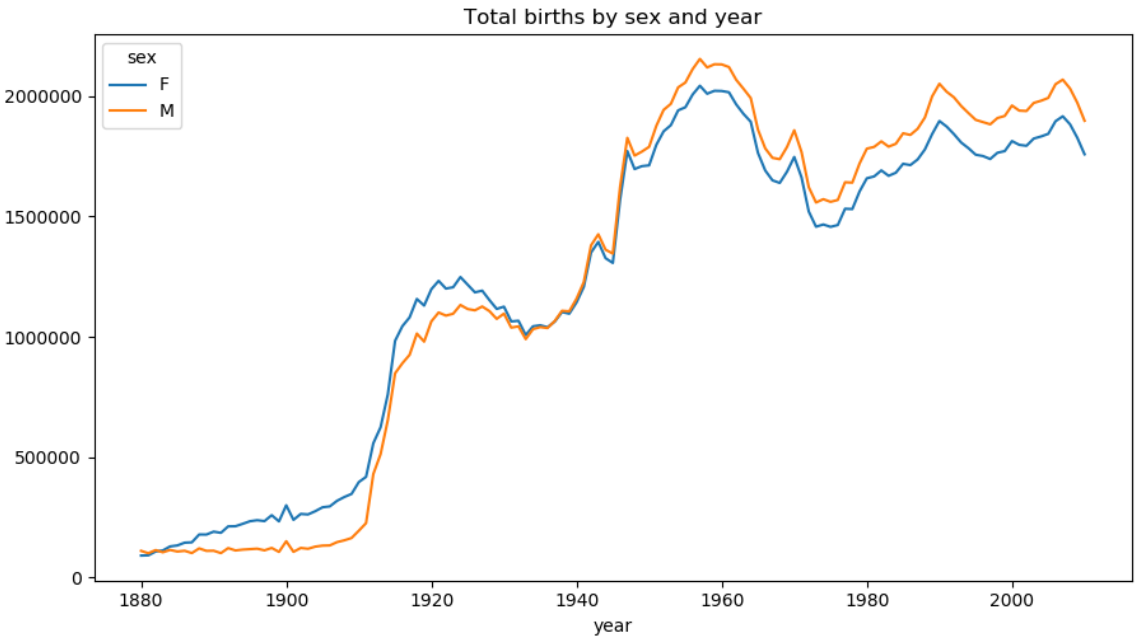
둘째, read_csv로 읽어온 원래 로우 순서는 몰라도 되니 concat 메서드에 ignore_index = True를 인자로 전달해야 한다. 이렇게 해서 전체 이름 데이터를 담고 있는 거대한 DataFrame 객체를 만들었다.

```
In [82]: names
Out[82]:
             name sex  births  year
0            Mary   F    7065  1880
```

```
1            Anna   F    2604  1880
2            Emma   F    2003  1880
3       Elizabeth   F    1939  1880
4          Minnie   F    1746  1880
...           ...  ..     ...   ...
1690779   Zymaire   M       5  2010
1690780    Zyonne   M       5  2010
1690781  Zyquarius  M       5  2010
1690782     Zyran   M       5  2010
1690783     Zzyzx   M       5  2010

[1690784 rows x 4 columns]
```

이제 이 데이터에 groupby나 pivot_table을 이용해서 연도나 성별에 따른 데이터를 수집할 수 있다.

```
In [83]: total_births = names.pivot_table('births', index = 'year',
    ...:                                   columns='sex', aggfunc=sum)

In [84]: total_births.tail()
Out[84]:
sex         F        M
year
2006  1896468  2050234
2007  1916888  2069242
2008  1883645  2032310
2009  1827643  1973359
2010  1759010  1898382

In [85]: total_births.plot(title = 'Total births by sex and year')
Out[85]: <matplotlib.axes._subplots.AxesSubplot at 0x27b07d95208>
```



다음은 prop 컬럼을 추가해서 각 이름이 전체 출생수에서 차지하는 비율을 계산하자. prop 값이 0.02 라면 100명의 아기 중 2명의 이름이 같다는 뜻이다. 데이터를 연도와 성별로 그룹화 하고 각 그룹에 새 칼럼을 추가하자.

```
In [16]: def add_prop(group):
    ...:     group['prop'] = group.births/group.births.sum()
```

```
    ...:     return group
    ...:

In [17]: names = names.groupby(['year','sex']).apply(add_prop)

In [18]: names
Out[18]:
              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]
```

그룹 관련 연산을 수행할 때는 모든 그룹에서 prop 컬럼의 합이 1이 맞는지 확인하는 새너티 테스트를 하는 게 좋다.

```
In [19]: names.groupby(['year','sex']).prop.sum()
Out[19]:
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
```

이제 모든 준비가 끝났고, 분석에 사용할 각 연도별/성별에 따른 선호하는 이름 1,000개를 추출하자.

```
In [20]: def get_top1000(group):
    ...:     return group.sort_values(by = 'births', ascending = False)
    ...: [:1000]
    ...:

In [21]: grouped = names.groupby(['year','sex'])

In [22]: top1000 = grouped.apply(get_top1000)
    #그룹 색인은 필요없으므로 삭제
In [23]: top1000.reset_index(inplace= True, drop = True)
```

함수를 정의하지 않고직접 추출하고 싶다면 다음처럼 할 수도 있다.

```
In [24]: pieces =  []

In [27]: for year, group in names.groupby(['year', 'sex']):
    ...:     pieces.append(group.sort_values(by='births',ascending=Fals
    ...: e)[:1000])

In [28]: top1000 = pd.concat(pieces, ignore_index=True)
```

이렇게 추출한 상위 1,000개의 이름 데이터는 이어지는 분석에서 사용하도록 하자.

```
In [29]: top1000
Out[29]:
             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
...           ...  ..     ...   ...       ...
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

[261877 rows x 5 columns]
```

14.3.1 이름 유행 분석

```
In [30]: boys = top1000[top1000.sex == 'M']

In [31]: girls = top1000[top1000.sex == 'F']
```

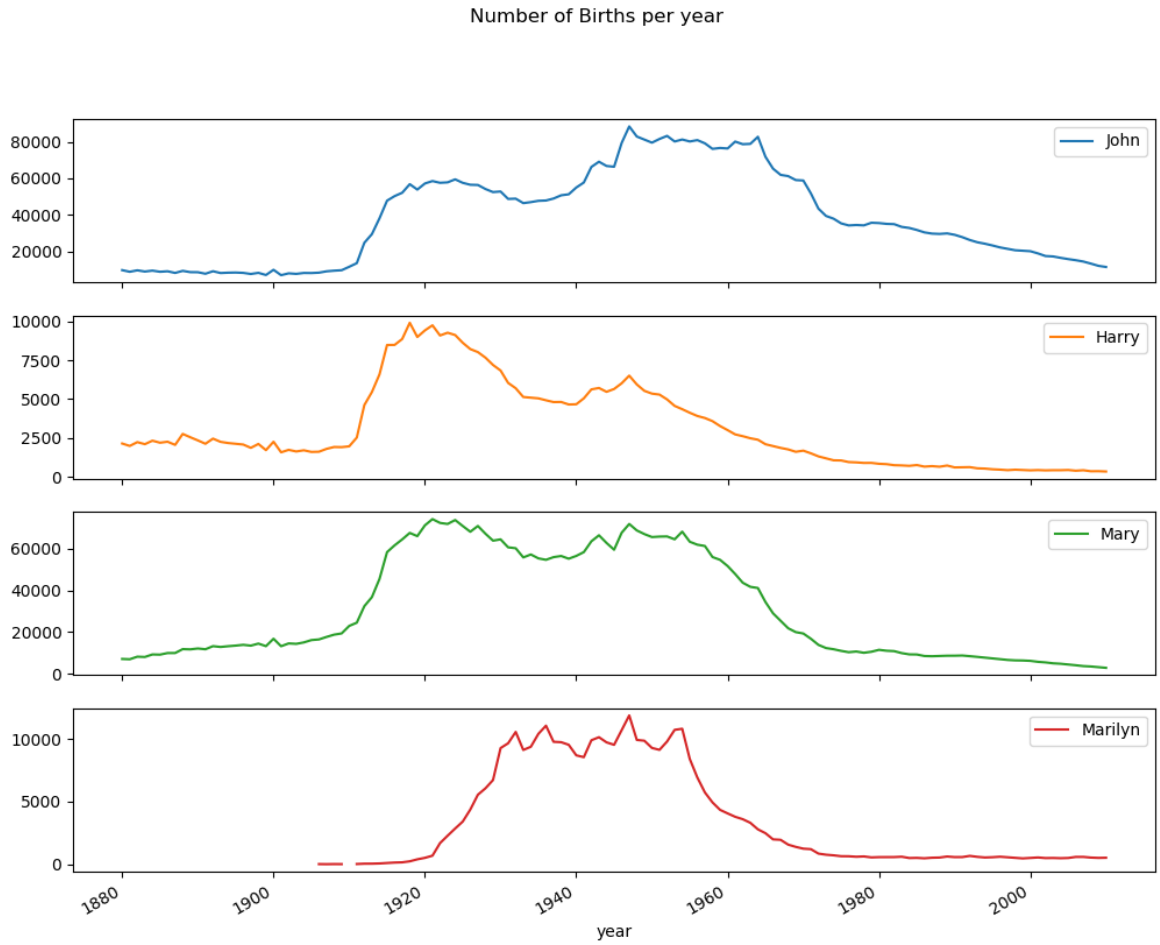
연도별로 John이나 Mary라는 이름의 추이를 간단하게 그래프로 그릴 수 있는데, 그전에 데이터를 살짝 변경할 필요가 있다.

```
In [32]: total_births = top1000.pivot_table('births', index = 'year',
    ...:                                     columns='name', aggfunc=su
    ...: m)
```

DataFrame의 plot 메서드를 사용해서 몇몇 이름의 추이를 그래프로 그려보자.

```
In [33]: 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 [34]: total_births
Out[34]:
name   Aaden  Aaliyah  Aarav   Aaron  ...  Zona  Zora  Zula   Zuri
year                                  ...
1880     NaN      NaN    NaN   102.0  ...   8.0  28.0  27.0    NaN
```

```
1881     NaN      NaN    NaN    94.0  ...    9.0  21.0  27.0    NaN
1882     NaN      NaN    NaN    85.0  ...   17.0  32.0  21.0    NaN
1883     NaN      NaN    NaN   105.0  ...   11.0  35.0  25.0    NaN
1884     NaN      NaN    NaN    97.0  ...    8.0  58.0  27.0    NaN
...      ...      ...    ...     ...  ...    ...   ...   ...    ...
2006     NaN   3737.0    NaN  8279.0  ...    NaN   NaN   NaN    NaN
2007     NaN   3941.0    NaN  8914.0  ...    NaN   NaN   NaN    NaN
2008   955.0   4028.0  219.0  8511.0  ...    NaN   NaN   NaN    NaN
2009  1265.0   4352.0  270.0  7936.0  ...    NaN   NaN   NaN    NaN
2010   448.0   4628.0  438.0  7374.0  ...    NaN   NaN   NaN  258.0

[131 rows x 6868 columns]

In [35]: subset = total_births[['John', 'Harry', 'Mary', 'Marilyn']]

In [36]: subset
Out[36]:
name     John   Harry    Mary  Marilyn
year
1880   9701.0  2158.0  7092.0      NaN
1881   8795.0  2002.0  6948.0      NaN
1882   9597.0  2246.0  8179.0      NaN
1883   8934.0  2116.0  8044.0      NaN
1884   9427.0  2338.0  9253.0      NaN
...       ...     ...     ...      ...
2006  15140.0   414.0  4073.0    596.0
2007  14405.0   443.0  3665.0    597.0
2008  13273.0   379.0  3478.0    543.0
2009  12048.0   383.0  3132.0    519.0
2010  11424.0   363.0  2826.0    531.0


In [37]: subset.plot(subplots = True, figsize = (12,10), grid = False,t
    ...: itle = 'Number of Births per year')
Out[37]:
array([<matplotlib.axes._subplots.AxesSubplot object at 0x0000029186EDA748>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000002918CC00288>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000002918C9BF188>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000002918CC67848>],
      dtype=object)

In [38]: plt.show()
```

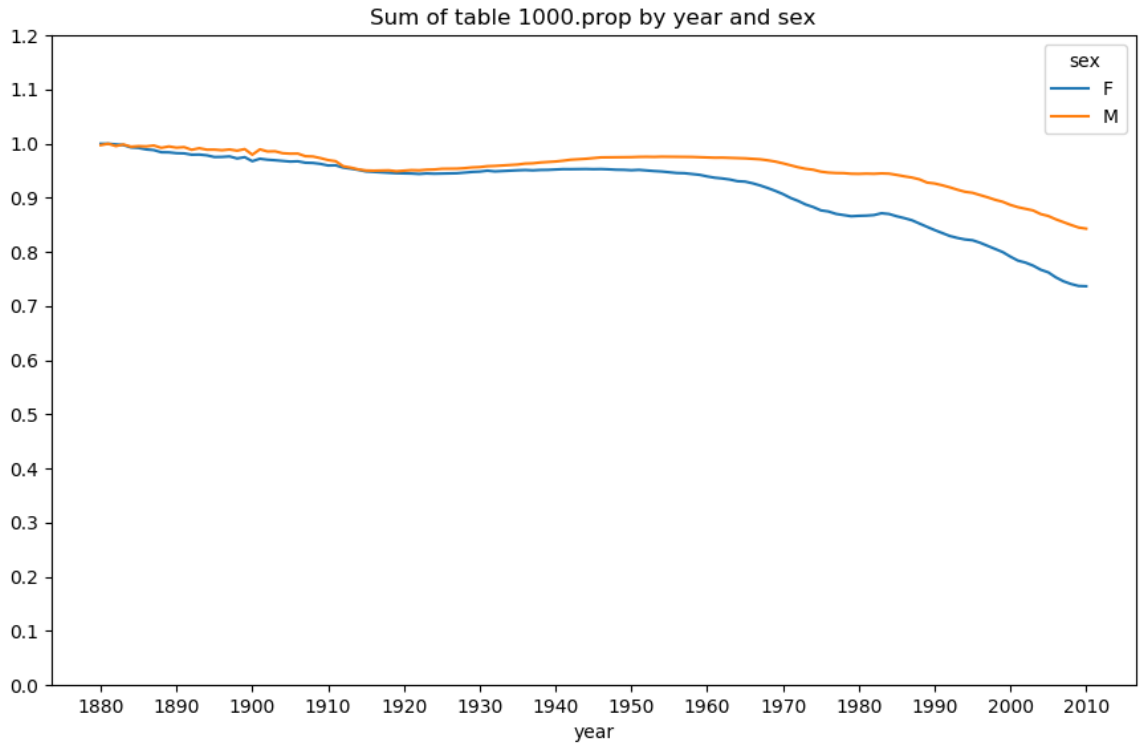


다양한 이름을 사용하는 경향 측정하기

위에서 확인한 그래프의 감소 추세는 부모가 아이의 이름을 지을 때 흔한 이름은 기피한다고 해석할 수 있다. 이 가설은 데이터에서 살펴볼 수 있으며 확인 가능하다. 좀 더 자세히 알아보기 위해 인기 있는 이름 1,000개가 전체 출생수에서 차지하는 비율을 연도별/성별 그래프로 그려보자.

```
In [41]: table = top1000.pivot_table('prop', index='year',columns='sex'
    ...: , aggfunc=sum)

In [42]: table
Out[42]:
sex          F         M
year
1880  1.000000  0.997375
1881  1.000000  1.000000
1882  0.998702  0.995646
1883  0.997596  0.998566
1884  0.993156  0.994539
...        ...       ...
2006  0.753153  0.860368
2007  0.745959  0.855159
2008  0.740933  0.850003
2009  0.737290  0.845256
2010  0.736780  0.843156

[131 rows x 2 columns]
```

```
In [45]: table.plot(title = 'Sum of table 1000.prop by year and sex', y
    ...: ticks = np.linspace(0,1.2,13), xticks = range(1880,2020,10))
Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x29187e374c8>
```



그림에서 확인할 수 있듯이 실제로 이름의 다양성이 증가하고 있음을 보여준다(상위 1,000개의 이름에서 비율의 총합이 감소하고 있다). 또한 인기있는 이름순으로 정렬했을 때, 전체 출생수의 50%를 차지하기까지 등장하는 이름 수도 흥미롭다.

```
In [47]: df = boys[boys.year == 2010]

In [48]: df
Out[48]:
           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]
```

prop을 내림차순으로 정렬하고 나서 전체의 50%가 되기까지 얼마나 많은 이름이 등장하는지 알아보자. for문을 사용해서 구현할 수도 있지만, 벡터화된 NumPy를 사용하는 편이 조금 더 편하다. prop의 누계를 cumsum에 저장하고 searchsorted 메서드를 호출해서 정렬된 상태에서 누계가 0.5되는 위치를 구한다.

```
In [49]: prop_cumsum = df.sort_values(by = 'prop', ascending=False).pro
    ...: p.cumsum()
```

```
In [50]: prop_cumsum[:10]
Out[50]:
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 [51]: prop_cumsum.values.searchsorted(0.5)
Out[51]: 116
```

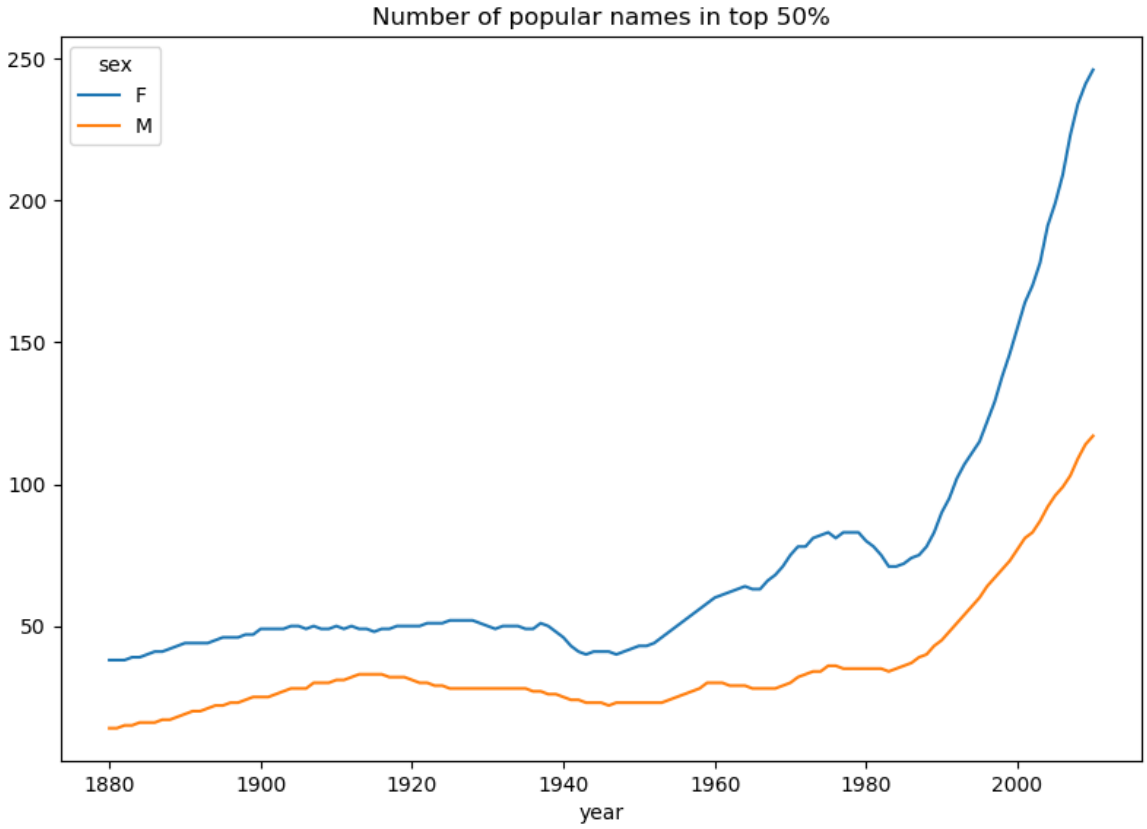
배열의 색인은 0부터 시작하기에 결과에 1을 더해주면 117이 나온다. 1900년에는 이보다 더 낮았다.

```
In [52]: df = boys[boys.year == 1900]

In [53]: in1900 = df.sort_values(by = 'prop', ascending=False).prop.cum
    ...: sum()

In [54]: in1900.values.searchsorted(0.5) + 1
Out[54]: 25
```

이제 이 연산을 각 연도별/성별 조합에 적용할 수 있다. 연도와 성을 groupby로 묶고 각 그룹에 apply를 사용해서 이 연산을 적용하면 된다.

```
In [55]: def get_quantile_count(group, q=0.5):
    ...:     group = group.sort_values(by='prop', ascending = False)
    ...:     return group.prop.cumsum().values.searchsorted(q)+1
    ...:

In [56]: diversity = top1000.groupby(['year', 'sex']).apply(get_quantil
    ...: e_count)

In [57]: diversity
Out[57]:
year  sex
1880  F       38
      M       14
1881  F       38
      M       14
1882  F       38
            ...
2008  M      109
2009  F      241
      M      114
2010  F      246
      M      117
Length: 262, dtype: int64

In [58]: diversity = diversity.unstack('sex')
```

```
In [59]: diversity
Out[59]:
sex     F    M
year
1880   38   14
1881   38   14
1882   38   15
1883   39   15
1884   39   16
...   ...  ...
2006  209   99
2007  223  103
2008  234  109
2009  241  114
2010  246  117

[131 rows x 2 columns]

In [60]: diversity.plot(title = 'Number of popular names in top 50%')
Out[60]: <matplotlib.axes._subplots.AxesSubplot at 0x2918e082908>

In [61]: plt.show()
```



여자아이의 이름은 항상 남자아이의 이름보다 더 다양하며, 시간이 흐를수록 더욱 다양지고 있다. 대체되는 철자의 증가 같은 다양성을 높이는 요인에 대한 자세한 분석은 알아서 해라.

마지막 글자의 변화

아이 이름을 연구하는 자가 지난 100년 동안 남자아이 이름의 마지막 글자의 분포에 중요한 변화가 있었다고 주장했다. 전체 자료에서 연도, 성별, 이름의 마지막 글자를 수집해서 이를 확인해보자.

```
In [33]: get_last_letter = lambda x:x[-1]

In [34]: last_letters = names.name.map(get_last_letter)

In [35]: last_letters.name = 'last_letter'

In [38]: table = names.pivot_table('births', index = last_let
    ...: ters, columns=['sex', 'year'], aggfunc=sum)
```

이제 전체 기간 중 세 지점을 골라 이름의 마지막 글자 몇 개를 출력해보자.

1910,1960,2010을 골랐다.

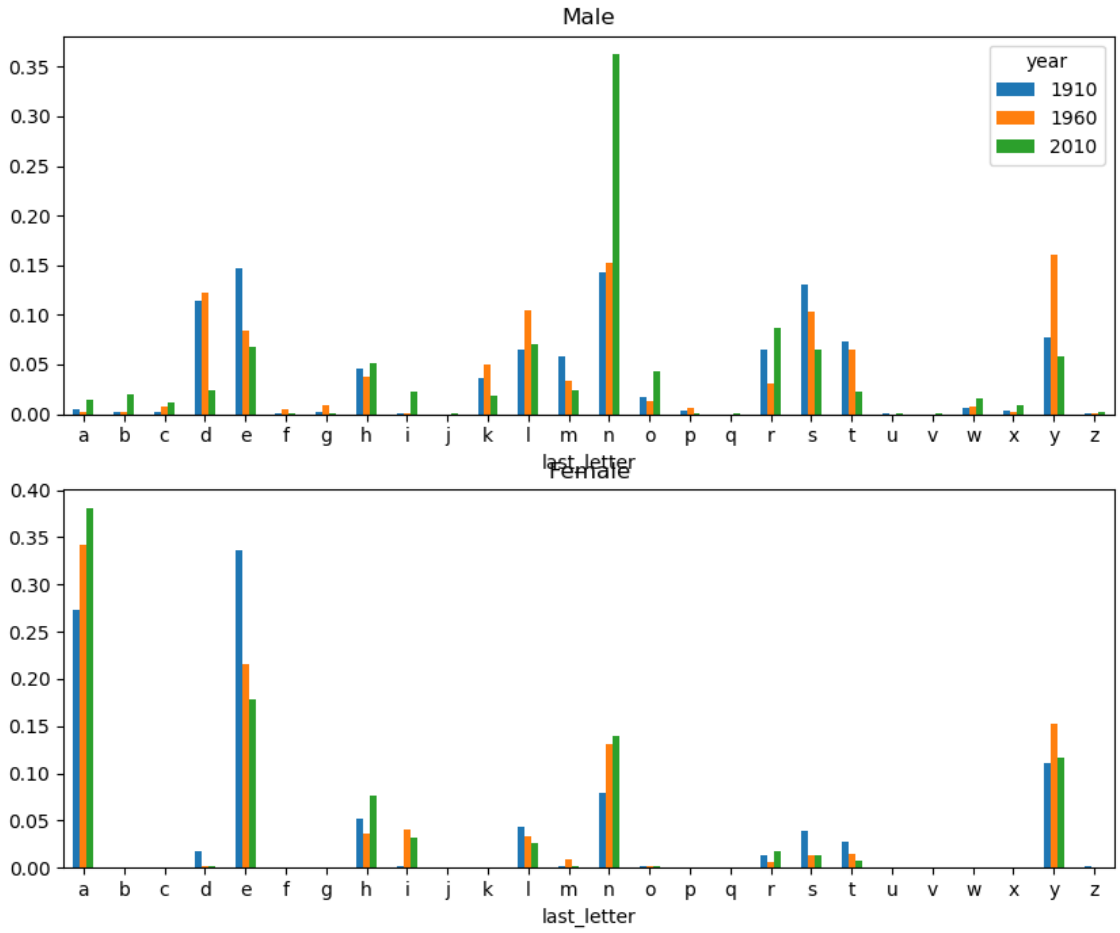
```
In [40]: subtable = table.reindex(columns=[1910,1960,2010], l
    ...: evel='year')

In [42]: subtable.head()
Out[42]:
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
```

그 다음에는 전체 출생수에서 성별별로 각각의 마지막 글자가 차지하는 비율을 계산하기 위해 전체 출생수로 정규화하자.

```
In [43]: subtable.sum()
Out[43]:
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 [44]: letter_prop = subtable/subtable.sum()

In [47]: letter_prop.head()
Out[47]:
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
```

이렇게 구한 이름의 마지막 글자 비율로 성별과 출생 연도에 대한 막대그래프를 그려보자.

```
In [48]: import matplotlib.pyplot as plt

In [50]: fig, axes = plt.subplots(2,1,figsize = (10,8))

In [52]: letter_prop['M'].plot(kind = 'bar', rot =0, ax = axe
    ...: s[0], title = 'Male')
Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x1b499cbe608>

In [53]: letter_prop['F'].plot(kind = 'bar', rot =0, ax = axe
    ...: s[1], title = 'Female', legend = False)
Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x1b499a02f48>
```

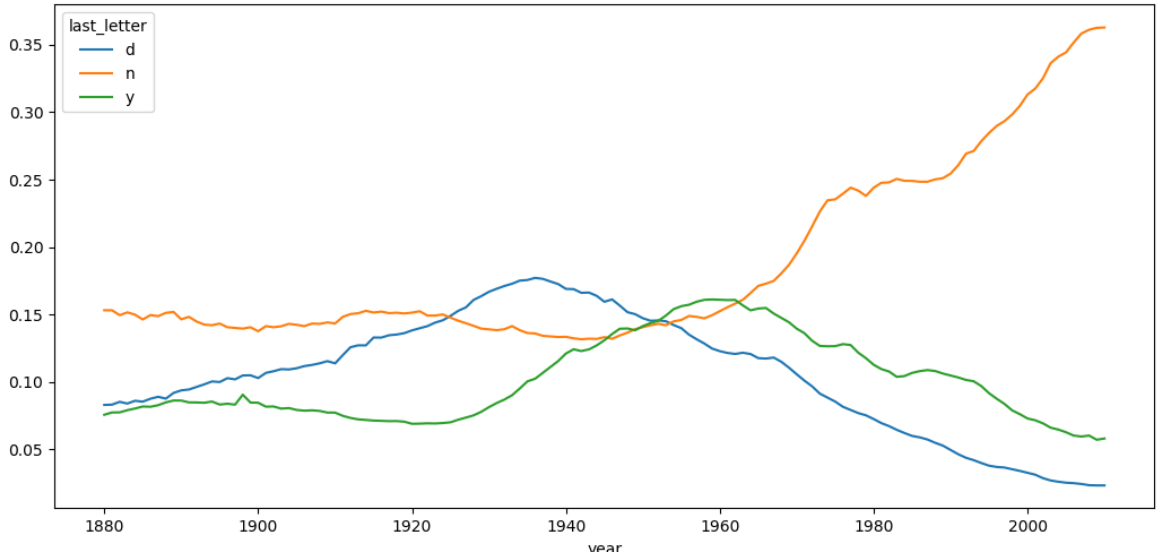


그래프에서 확인할 수 있듯이 'n'으로 끝나는 남자아이 이름의 빈도가 1960년대 이후에 급격하게 증가했다. 이제 세 지점이 아닌 전체 자료에 대해 출생연도와 성별, 남자아이 이름에서 몇가지 글자로 정규화하고 시계열 데이터로 변환하자.

```
In [55]: letter_prop = table/table.sum()

In [56]: dny_ts = letter_prop.loc[['d','n','y'], 'M'].T

In [57]: dny_ts.head()
Out[57]:
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
```

이 시계열 데이터를 plot 메서드를 이용해서 연도별 그래프로 만들어 보자.

```
In [58]: dny_ts.plot()
Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x1b499928fc8>
```



남자 이름과 여자 이름이 바뀐 경우

또 다른 재미있는 경향은 예전에 남자이름으로 선호되다가 현재는 여자이름으로 선호되는 경우다. Lesley와 Leslie 라는 이름이 그렇다. top1000 데이터를 이용해서 'lesl'로 시작하는 이름을 포함하는 목록을 만들어 보자.

```
In [60]: all_names = pd.Series(top1000.name.unique())

In [61]: all_names
Out[61]:
0             Mary
1             Anna
2             Emma
3        Elizabeth
4           Minnie
           ...
6863         Masen
6864         Rowen
6865        Yousef
6866        Joziah
6867        Maxton
```

```
Length: 6868, dtype: object

In [62]: lesley_like = all_names[all_names.str.lower().str.co
    ...: ntains('lesl')]

In [63]: lesley_like
Out[63]:
632     Leslie
2294    Lesley
4262    Leslee
4728     Lesli
6103     Lesly
dtype: object
```

이제 이 이름들만 걸러내서 이름별로 출생수를 구하고 상대도수를 확인해보자.

```
In [64]: filtered = top1000[top1000.name.isin(lesley_like)]

In [65]: filtered.groupby('name').births.sum()
Out[65]:
name
Leslee      1082
Lesley     35022
Lesli        929
Leslie    370429
Lesly      10067
Name: births, dtype: int64
```

그리고 성별과 연도별로 모은 다음 출생연도로 정규화한다.

```
In [67]: table = filtered.pivot_table('births', index = 'year
    ...: ', columns='sex', aggfunc='sum')

In [68]: table = table.div(table.sum(1), axis=0)

In [69]: table
Out[69]:
sex          F         M
year
1880  0.091954  0.908046
1881  0.106796  0.893204
1882  0.065693  0.934307
1883  0.053030  0.946970
1884  0.107143  0.892857
...        ...       ...
2006  1.000000       NaN
2007  1.000000       NaN
2008  1.000000       NaN
2009  1.000000       NaN
2010  1.000000       NaN
```

마지막으로 시대별로 성별에 따른 명세를 그래프로 그려보자.


```
In [70]: table.plot(style = {'M':'k-', 'F':'k--'})
Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x1b491e43d88>
```

14.4 미국농무부 영양소 정보

웹사이트에서 데이터를 내려받은 다음 압축을 풀고, 선호하는 JSON 라이브러리를 사용해서 파이썬에서 읽어오자.

```
In [72]: import json

In [73]: db = json.load(open('datasets/usda_food/database.jso
    ...: n'))

In [74]: len(db)
Out[74]: 6636
```

db에 있는 각 엔트리는 한 가지 음식에 대한 모든 정보를 담고 있는 사전형이다. 'nutrients'필드는 사전의 리스트이며 각 항목은 한 가지 영양소에 대한 정보를 담고 있다.

```
In [77]: db[0].keys()
Out[77]: dict_keys(['id', 'description', 'tags', 'manufacturer',
        'group', 'portions', 'nutrients'])

In [78]: db[0]['nutrients'][0]
Out[78]:
{'value': 25.18,
 'units': 'g',
 'description': 'Protein',
 'group': 'Composition'}

 In [84]: nutrients = pd.DataFrame(db[0]['nutrients'])

 In [86]: nutrients [:7]
Out[86]:
     value units                 description        group
0    25.18     g                     Protein  Composition
1    29.20     g           Total lipid (fat)  Composition
```

```
2     3.06     g  Carbohydrate, by difference  Composition
3     3.28     g                          Ash        Other
4   376.00  kcal                       Energy       Energy
5    39.28     g                        Water  Composition
6  1573.00    kJ                       Energy       Energy
```

사전의 리스트를 DataFrame으로 바꿀 때 추출할 필드 목록을 지정해줄 수 있다. 우리는 음식의 이름과 그룹, id 그리고 제조사를 추출하겠다.

```
In [93]: info_keys = ['description', 'group', 'id', 'manufacturer']

In [94]: info = pd.DataFrame(db, columns=info_keys)

In [96]: info[:5]
Out[96]:
                         description                   group    id manufacturer
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

In [97]: info.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6636 entries, 0 to 6635
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   description   6636 non-null   object
 1   group         6636 non-null   object
 2   id            6636 non-null   int64
 3   manufacturer  5195 non-null   object
dtypes: int64(1), object(3)
memory usage: 207.5+ KB
```

value_counts 메서드를 이용해서 음식 그룹의 분포를 확인할 수 있다.

```
In [108]: pd.value_counts(info.group)[:10]
Out[108]:
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
Fruits and Fruit Juices              328
Pork Products                        328
Name: group, dtype: int64
```

모든 영양소 정보를 분석해보자. 좀 더 쉬운 분석을 위해 각 음식의 영양소 정보를 거대한 테이블 하나에 담아보자. 그러려면 사전에 몇 가지 과정을 거쳐야 한다. 먼저 음식의 영양소 리스트를 하나의 DataFrame으로 변환하고, 음식의 id를 더한 컬럼을 하나 추가한다.

또한 이 DataFrame을 리스트에 추가한다. 그리고 마지막으로 이 리스트를 concat 메서드를 사용해서 하나로 합친다.

```
In [119]: terms = range(0,len(db))

In [120]: pieces_n = []

In [122]: for term in terms:
     ...:     frame = pd.DataFrame(db[term]['nutrients'])
     ...:     frame['id'] = id
     ...:     pieces_n.append(frame)

In [123]: nutrients_nnn = pd.concat(pieces_n, ignore_index=True)

In [124]: nutrients_nnn
Out[124]:
          value units                           description        group
id
0        25.180     g                               Protein  Composition  <built-in
function id>
1        29.200     g                     Total lipid (fat)  Composition  <built-in
function id>
2         3.060     g           Carbohydrate, by difference  Composition  <built-in
function id>
3         3.280     g                                   Ash        Other  <built-in
function id>
4       376.000  kcal                                Energy       Energy  <built-in
function id>
...         ...   ...                                   ...          ...
...
389350    0.000   mcg                   Vitamin B-12, added     Vitamins  <built-in
function id>
389351    0.000    mg                           Cholesterol        Other  <built-in
function id>
389352    0.072     g          Fatty acids, total saturated        Other  <built-in
function id>
389353    0.028     g  Fatty acids, total monounsaturated        Other  <built-in
function id>
389354    0.041     g  Fatty acids, total polyunsaturated        Other  <built-in
function id>

[389355 rows x 5 columns]
```

이유야 어쨌든 이 DataFrame에는 중복된 데이터가 있으므로 제거하도록 한다.

```
In [140]: nutrients_nnn.duplicated().sum() #중복 확인
Out[140]: 311192

In [144]: nutrients_nnn = nutrients_nnn.drop_duplicates()

In [145]: nutrients_nnn
Out[145]:
          value units                           description        group
id
0        25.180     g                               Protein  Composition  <built-in
function id>
```

```
1        29.200     g                  Total lipid (fat)  Composition  <built-in
function id>
2         3.060     g          Carbohydrate, by difference  Composition  <built-in
function id>
3         3.280     g                                  Ash        Other  <built-in
function id>
4       376.000  kcal                              Energy       Energy  <built-in
function id>
...         ...   ...                                 ...          ...
...
389202    9.719     g  Fatty acids, total monounsaturated        Other  <built-in
function id>
389214    7.280     g                      Sugars, total  Composition  <built-in
function id>
389236   33.800    mg      Vitamin C, total ascorbic acid     Vitamins  <built-in
function id>
389256   17.400     g          Carbohydrate, by difference  Composition  <built-in
function id>
389315   11.360     g                      Sugars, total  Composition  <built-in
function id>

[78163 rows x 5 columns]
```

'group'과 'description'은 모두 DataFrame 객체이므로 뭔가 뭔지 쉽게 알아볼 수 없기에 이름을 바꿔주자.

14.5 2012년 연방선거관리위원회 데이터베이스

```
In [163]: fec = pd.read_csv('datasets/fec/P00000001-ALL.csv')
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3063:
DtypeWarning: Columns (6) have mixed types.Specify dtype option on import or set
low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)

In [164]: fec.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1001731 entries, 0 to 1001730
Data columns (total 16 columns):
 #   Column             Non-Null Count    Dtype
---  ------             --------------    -----
 0   cmte_id            1001731 non-null  object
 1   cand_id            1001731 non-null  object
 2   cand_nm            1001731 non-null  object
 3   contbr_nm          1001731 non-null  object
 4   contbr_city        1001712 non-null  object
 5   contbr_st          1001727 non-null  object
 6   contbr_zip         1001620 non-null  object
 7   contbr_employer    988002 non-null   object
 8   contbr_occupation  993301 non-null   object
 9   contb_receipt_amt  1001731 non-null  float64
 10  contb_receipt_dt   1001731 non-null  object
 11  receipt_desc       14166 non-null    object
 12  memo_cd            92482 non-null    object
 13  memo_text          97770 non-null    object
 14  form_tp            1001731 non-null  object
```

```
 15  file_num          1001731 non-null  int64
dtypes: float64(1), int64(1), object(14)
memory usage: 122.3+ MB
```

DataFrame에는 다음과 같은 형태로 저장되어 있다.

```
In [165]: fec.iloc[123456]
Out[165]:
cmte_id                            C00431445
cand_id                            P80003338
cand_nm                         Obama, Barack
contbr_nm                         ELLMAN, IRA
contbr_city                             TEMPE
contbr_st                                  AZ
contbr_zip                          852816719
contbr_employer      ARIZONA STATE UNIVERSITY
contbr_occupation                   PROFESSOR
contb_receipt_amt                          50
contb_receipt_dt                    01-DEC-11
receipt_desc                              NaN
memo_cd                                   NaN
memo_text                                 NaN
form_tp                                 SA17A
file_num                               772372
Name: 123456, dtype: object
```

여기에는 정당 가입 여부에 대한 데이터가 없으므로 추가해주는 것이 유용하다. unique 메서드를 이용해서 모든 정당의 후보 목록을 얻자.

```
In [205]: unique_cands = fec.cand_nm.unique()

In [206]: unique_cands
Out[206]:
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 [207]: unique_cands[2]
Out[207]: 'Obama, Barack'
```

소속 정당은 dict을 사용해서 표시할 수 있다.

```
In [208]: parties = {'Bachmann, Michelle' : 'Republican',
     ...:            'Cain, Herman' : 'Republican',
     ...:            'Romney, Mitt' : 'Republican',
     ...:            "Roemer, Charles E. 'Buddy' III" : 'Republican',
     ...:            'Pawlenty, Timothy' : 'Republican',
     ...:            'Johnson, Gary Earl' : 'Republican',
     ...:            'Paul, Ron' : 'Republican',
     ...:            'Santorum, Rick' : 'Republican',
     ...:            'Gingrich, Newt' : 'Republican',
     ...:            'McCotter, Thaddeus G' : 'Republican',
     ...:            'Huntsman, Jon' : 'Republican',
```

```
...:             'Perry, Rick' : 'Republican',
...:             'Obama, Barack' : 'Democrat'
...:              }
```

이제 이 사전 정보와 Series 객체의 map 메서드를 사용해 후보 이름으로부터 정당 배열을 계산해낼 수 있다.

```
In [210]: fec.cand_nm[123456:123461]
Out[210]:
123456    Obama, Barack
123457    Obama, Barack
123458    Obama, Barack
123459    Obama, Barack
123460    Obama, Barack
Name: cand_nm, dtype: object

In [213]: fec.cand_nm[123456:123461].map(parties)
Out[213]:
123456    Democrat
123457    Democrat
123458    Democrat
123459    Democrat
123460    Democrat
Name: cand_nm, dtype: object

#party 컬럼으로 추가
In [214]: fec['party'] = fec.cand_nm.map(parties)

In [215]: fec['party'].value_counts()
Out[215]:
Democrat      593746
Republican    407985
Name: party, dtype: int64
```

분석을 하기 전에 데이터를 다듬어야 한다. 이 데이터에는 기부금액과 환급금액(기부금액이 마이너스인 경우)이 함께 포함되어 있다.

```
In [216]: (fec.contb_receipt_amt>0).value_counts()
Out[216]:
True     991475
False     10256
Name: contb_receipt_amt, dtype: int64
```

분석을 단순화 하기 위해 기부금액이 양수인 데이터만 골라내겠다.

```
In [217]: fec = fec[fec.contb_receipt_amt>0]

In [218]: fec
Out[218]:
          cmte_id    cand_id             cand_nm  ... form_tp file_num
party
0        C00410118  P20002978  Bachmann, Michelle  ...   SA17A   736166
Republican
1        C00410118  P20002978  Bachmann, Michelle  ...   SA17A   736166
Republican
```

```
2        C00410118  P20002978  Bachmann, Michelle  ...   SA17A   749073
Republican
3        C00410118  P20002978  Bachmann, Michelle  ...   SA17A   749073
Republican
4        C00410118  P20002978  Bachmann, Michelle  ...   SA17A   736166
Republican
...            ...        ...                 ...  ...     ...      ...
...
1001726  C00500587  P20003281         Perry, Rick  ...   SA17A   751678
Republican
1001727  C00500587  P20003281         Perry, Rick  ...   SA17A   751678
Republican
1001728  C00500587  P20003281         Perry, Rick  ...   SA17A   751678
Republican
1001729  C00500587  P20003281         Perry, Rick  ...   SA17A   751678
Republican
1001730  C00500587  P20003281         Perry, Rick  ...   SA17A   751678
Republican

[991475 rows x 17 columns]
```

버락 오바마와 미트 롬니가 양대 후보이므로 이 두 후보의 기부금액 정보만 따로 추려내겠다.

```
In [175]: fec_mrbo = fec[fec.cand_nm.isin(['Obama, Barack','Romney, Mitt'])]
```

14.5.1 직업 및 고용주에 따른 기부 통계

직업에 따른 기부 통계는 흔한 조사방법이다. 예를 들어 변호사는 민주당에 더 많은 돈을, 기업 임원은 공화당에 더 많은 돈을 기부하는 경향이 있다. 이를 데이터로 확인해보자.

```
In [175]: fec_mrbo = fec[fec.cand_nm.isin(['Obama, Barack','Romney, Mitt'])]

In [176]: fec.contbr_occupation.value_counts()[:10]
Out[176]:
RETIRED                                   234829
INFORMATION REQUESTED                      35176
ATTORNEY                                   34409
HOMEMAKER                                  30199
PHYSICIAN                                  23530
INFORMATION REQUESTED PER BEST EFFORTS     21364
ENGINEER                                   14372
TEACHER                                    13998
CONSULTANT                                 13335
PROFESSOR                                  12565
Name: contbr_occupation, dtype: int64
```

내용을 보면 일반적인 직업 유형이거나 같은 유형이지만 다른 이름으로 많은 결과가 포함되어 있음을 알 수 있다. 아래 코드를 이용해서 하나의 직업을 다른 직업으로 매핑함으로써 이런 몇몇 문제를 제거하자. dict.get을 사용하는 '꼼수'를 써서 매핑 정보가 없는 직업은 그대로 사용한다.

```
In [180]: occ_mapping = {
     ...:     'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
     ...:     'INFORMATION REQUESTED' : 'NOT PROVIDED',
     ...:     'INFORMATION REQUESTED (BEST EFFORTS)': 'NOT PROVIDED',
     ...:     'C.E.O':'CEO'}

#mapping이 없다면 x를 반환한다.
In [182]: f = lambda x: occ_mapping.get(x,x)

In [183]: fec.contbr_occupation = fec.contbr_occupation.map(f)
```

고용주에 대해서도 마찬가지로 처리하자.

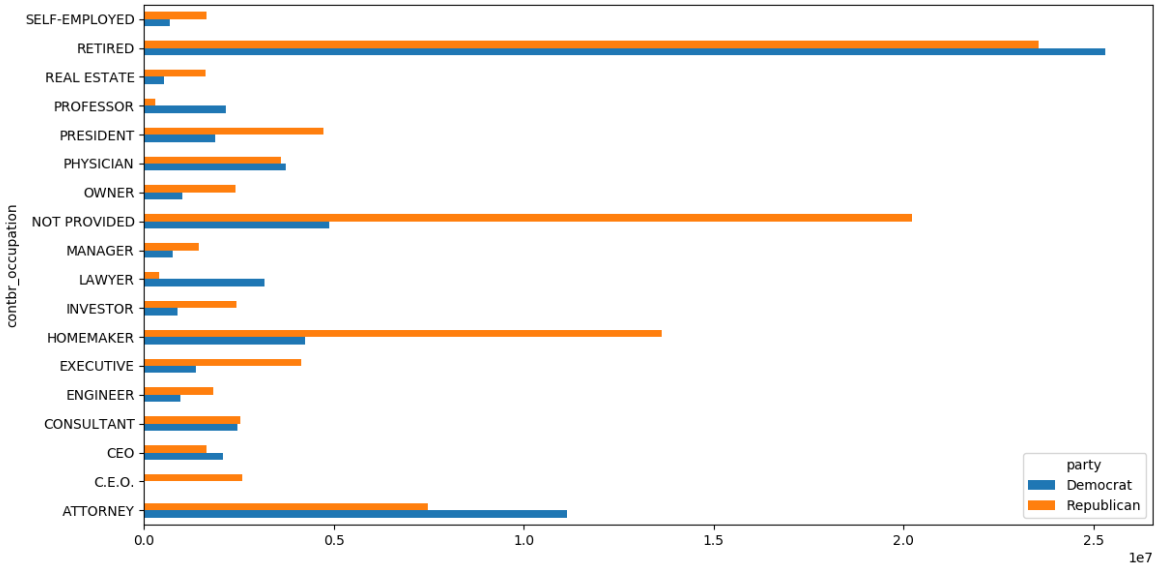
```
In [186]: emp_mapping = {
     ...:     'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
     ...:     'INFORMATION REQUESTED' : 'NOT PROVIDED',
     ...:     'SELF': 'SELF-EMPLOYED',
     ...:     'SELF EMPLOYED':'SELF-EMPLOYED'}

In [187]: f = lambda x: emp_mapping.get(x,x)

In [189]: fec.contbr_employer = fec.contbr_employer.map(f)
```

이제 pivot_table을 사용해서 정당과 직업별로 데이터를 집계한 다음 최소 2백만불 이상 기부한 직업만 골라내자.

```
In [222]: by_occupation = fec.pivot_table('contb_receipt_amt',
     ...:                                  index = 'contbr_occupation',
     ...:                                  columns = 'party', aggfunc='sum')

In [223]: over_2mm = by_occupation[by_occupation.sum(1)>2000000]

In [224]: over_2mm
Out[224]:
party                 Democrat    Republican
contbr_occupation
ATTORNEY           11141982.97  7.477194e+06
C.E.O.                1690.00  2.592983e+06
CEO                2074284.79  1.640758e+06
CONSULTANT         2459912.71  2.544725e+06
ENGINEER            951525.55  1.818374e+06
EXECUTIVE          1355161.05  4.138850e+06
HOMEMAKER          4248875.80  1.363428e+07
INVESTOR            884133.00  2.431769e+06
LAWYER             3160478.87  3.912243e+05
MANAGER             762883.22  1.444532e+06
NOT PROVIDED       4866973.96  2.023715e+07
OWNER              1001567.36  2.408287e+06
PHYSICIAN          3735124.94  3.594320e+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
```

이런 종류의 데이터는 막대그래프('barh'는 수평막대그래프를 의미한다.)로 시각화하는 편이 보기 좋다.

```
In [225]: over_2mm.plot(kind = 'barh')
```



오바마 후보와 롬니 후보별로 가장 많은 금액을 기부한 직군을 알아보자. 이 통계를 구하려면 후보 이름으로 그룹을 묶고 이 장의 앞에 사용했던 변형된 top 메서드를 사용하면 된다.

```
In [228]: def get_top_amounts(group,key, n=5):
     ...:     totals = group.groupby(key)['contb_receipt_amt'].sum()
     ...:     return totals.nlargest(n)
```

그리고 직업과 고용주에 따라 집계하면 된다.

```
In [229]: grouped = fec_mrbo.groupby('cand_nm')

In [230]: grouped.apply(get_top_amounts, 'contbr_occupation', n=7)
Out[230]:
cand_nm         contbr_occupation
Obama, Barack   RETIRED                   25305116.38
                ATTORNEY                  11141982.97
                NOT PROVIDED               4866973.96
                HOMEMAKER                  4248875.80
                PHYSICIAN                  3735124.94
                LAWYER                     3160478.87
                CONSULTANT                 2459912.71
Romney, Mitt    RETIRED                   11508473.59
                NOT PROVIDED              11396894.84
                HOMEMAKER                  8147446.22
                ATTORNEY                   5364718.82
                PRESIDENT                  2491244.89
                EXECUTIVE                  2300947.03
                C.E.O.                     1968386.11
Name: contb_receipt_amt, dtype: float64

In [20]: grouped.sum()
Out[20]:
                contb_receipt_amt      file_num
cand_nm
Obama, Barack       1.358774e+08  456512611214
```

```
Romney, Mitt        8.833591e+07   81366834342

In [21]: grouped.apply(get_top_amounts, 'contbr_employer', n=10)
Out[21]:
cand_nm        contbr_employer
Obama, Barack  RETIRED                                   22694358.85
               SELF-EMPLOYED                             17080985.96
               NOT EMPLOYED                               8586308.70
               INFORMATION REQUESTED                      5053480.37
               HOMEMAKER                                  2605408.54
               SELF                                       1076531.20
               SELF EMPLOYED                               469290.00
               STUDENT                                     318831.45
               VOLUNTEER                                   257104.00
               MICROSOFT                                   215585.36
Romney, Mitt   INFORMATION REQUESTED PER BEST EFFORTS     12059527.24
               RETIRED                                   11506225.71
               HOMEMAKER                                  8147196.22
               SELF-EMPLOYED                              7409860.98
               STUDENT                                     496490.94
               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, dtype: float64
```

14.5.2 기부금액

이 데이터를 효과적으로 분석하는 방법은 cut 함수를 사용해서 기부 규모별로 버킷을 만들어 기부자 수를 분할하는 것이다.

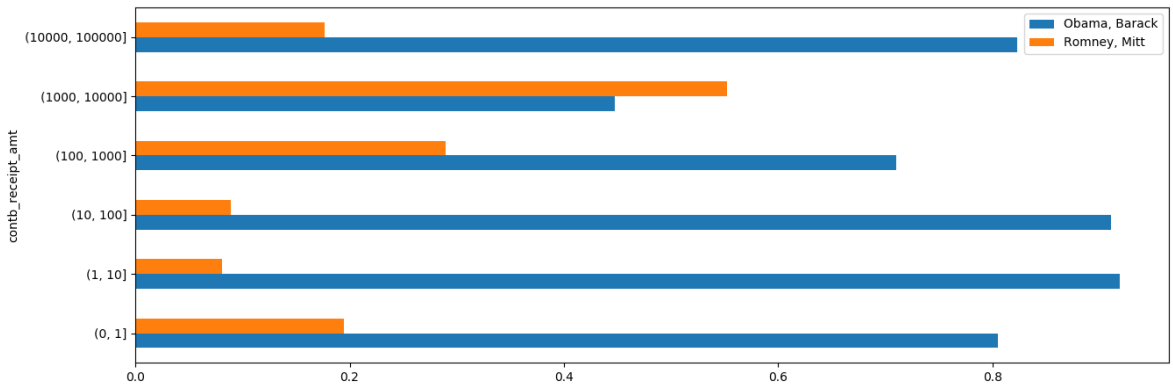
```
In [86]: bins = np.array([0, 1,10,100,1000,10000,100000, 1000000, 10000
    ...: 000])

In [87]: labels = pd.cut(fec_mrbo.contb_receipt_amt, bins)

In [89]: labels
Out[89]:
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] <
(1000, 10000] <
                                  (10000, 100000] < (100000, 1000000] < (1000000,
10000000]]
```

이제 이 데이터를 이름과 버킷 이름으로 그룹지어 기부금액 규모에 따른 히스토그램 그릴 수 있다.

```
In [90]: grouped = fec_mrbo.groupby(['cand_nm', labels])

In [91]: grouped.size().unstack(0)
Out[91]:
cand_nm              Obama, Barack  Romney, Mitt
contb_receipt_amt
(0, 1]                         493            77
(1, 10]                      40070          3681
(10, 100]                   372280         31853
(100, 1000]                 153991         43357
(1000, 10000]                22284         26186
(10000, 100000]                  2             1
(100000, 1000000]                3             0
(1000000, 10000000]              4             0
```

이 데이터를 보면 오바마는 롬니보다 적은 금액의 기부를 훨씬 많이 받았다. 기부 금액을 모두 더한 후 버킷 별로 정규화해서 후보별 전체 기부금액 대비 비율을 시각화할 수 있다.

```
In [98]: bucket_sums = grouped.contb_receipt_amt.sum().unstack(0)

In [99]: normed_sums = bucket_sums.div(bucket_sums.sum(axis =1), axis =
    ...: 0)

In [100]: normed_sums
Out[100]:
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 [106]: normed_sums[:-2].plot(kind = 'barh')
Out[106]: <matplotlib.axes._subplots.AxesSubplot at 0x26441100b88>

In [107]: plt.legend()
Out[107]: <matplotlib.legend.Legend at 0x264417ab6c8>
```



기부금액 순에서 가장 큰 2개의 버킷은 개인 후원이 아니므로 그래프에서 제외시켰다.

14.5.3 주별 기부 통계

데이터를 후보자와 주별로 집계하는 것은 흔한 일이다.

```
In [109]: grouped = fec_mrbo.groupby(['cand_nm','contbr_st'])

In [110]: totals = grouped.contb_receipt_amt.sum().unstack(0).fillna(0)
     ...:

In [114]: totals = totals[totals.sum(1)>100000]

In [115]: totals[:10]
Out[115]:
cand_nm    Obama, Barack  Romney, Mitt
contbr_st
AK              281840.15      86204.24
AL              543123.48     527303.51
AR              359247.28     105556.00
AZ             1506476.98    1888436.23
CA            23824984.24   11237636.60
CO             2132429.49    1506714.12
CT             2068291.26    3499475.45
DC             4373538.80    1025137.50
DE              336669.14      82712.00
FL             7318178.58    8338458.81
```

각 로우를 전체 기부금액으로 나누면 각 후보에 대한 주별 전체 기부금액의 상대적인 비율을 얻을 수 있다.

```
In [116]: percent = totals.div(totals.sum(1), axis=0)

In [117]: percent[:10]
Out[117]:
cand_nm    Obama, Barack  Romney, Mitt
contbr_st
AK              0.765778      0.234222
AL              0.507390      0.492610
AR              0.772902      0.227098
AZ              0.443745      0.556255
CA              0.679498      0.320502
CO              0.585970      0.414030
CT              0.371476      0.628524
DC              0.810113      0.189887
DE              0.802776      0.197224
FL              0.467417      0.532583
```