파이썬 라이브러리를 활용한 데이터 분석

14장 데이터 분석 예제

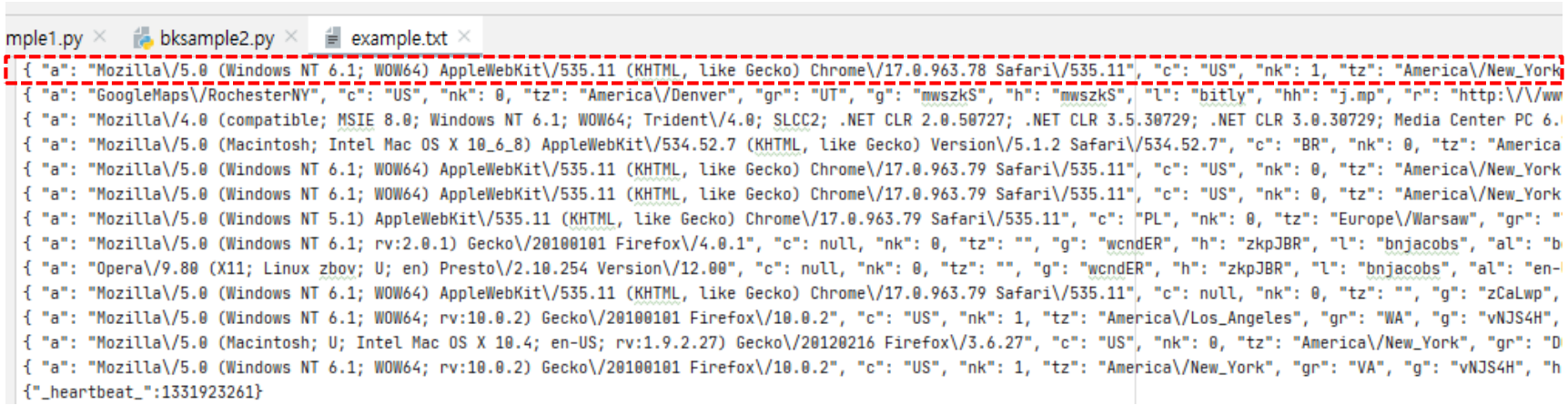
14장 데이터 분석 예제

URL 축약 서비스 정보 분석

2h

Bit.ly

- URL 축약 서비스 업체
- 파일 내용
  - 미국 정보 웹사이트 USA.gov와 제휴하여
    - **.gov와 .mil을 축약한 사용자 들의 익명 정보 제공**
  - 서비스를 사용한 정보를 행마다 저장한 파일
- 파이참에서 확인 가능



JSON(JavaScript Object Notation) 형식

- **특징**
  - JSON은 경량(Lightweight)의 DATA-교환 형식
  - Javascript에서 객체를 만들 때 사용하는 표현식을 의미
  - 사람과 기계 모두 이해하기 쉬우며 용량이 작음
    - **최근에는 JSON이 XML을 대체해서 데이터 전송 등에 많이 사용**
  - 특정 언어에 종속되지 않으며, 대부분의 프로그래밍 언어에서 JSON 포맷의 데이터를 핸들링 할 수 있는 라이브러리를 제공

- **파이썬의 사전 형식**

```
{
  "firstName": "Kwon",
  "lastName": "YoungJae",
  "email": "kyoje11@gmail.com",
  "hobby": ["puzzles","swimming"]
}
```

파일 읽기

- **스냅 샷 파일의 행**
  - JSON
  - 한 줄 검사
- **records 객체**
  - 파이썬 사전 리스트
  - 총 3560개

In [5]:
```
import json
path = 'datasets/bitly_usagov/example.txt'
open(path).readline()
```

Out[5]:
```
'{ "a": "Mozilla₩₩/5.0 (Windows NT 6.1; WOW64) AppleWebKit₩₩/535.11 (KHTML, like Ge
cko) Chrome₩₩/17.0.963.78 Safari₩₩/535.11", "c": "US", "nk": 1, "tz": "America₩₩/Ne
w_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.us
a.gov₩₩/wfLQtf", "u": "http:₩₩/₩₩/www.ncbi.nlm.nih.gov₩₩/pubmed₩₩/22415991", "t": 1
331923247, "hc": 1331822918, "cy": "Danvers", "ll": [ 42.576698, -70.954903 ] }₩n'
```

In [10]:
```
records = [json.loads(line) for line in open(path, encoding="utf-8")]
records[0]
```

Out[10]:
```
{'a': 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.11 (KHTML, like Gecko) C
hrome/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]}
```

표준 시간대 파악

- **표준 시간대**
  - tz 필드
  - 모든 행이 키 'tz'가 있는 건 아님
    - **오류 발생**

```
In [15]: records[0]['tz']
Out[15]: 'America/New_York'

In [13]: time_zones = [rec['tz'] for rec in records]
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-13-f3fbbc37f129> in <module>
----> 1 time_zones = [rec['tz'] for rec in records]

<ipython-input-13-f3fbbc37f129> in <listcomp>(.0)
----> 1 time_zones = [rec['tz'] for rec in records]

KeyError: 'tz'

In [25]: records[:10]
  'hh': '1.usa.gov',
  'r': 'http://www.facebook.com/l.php?u=http%3A%2F%2F1.usa.gov%2FzkpJBR&h=fAQG5ntSG
AQHqKPIWzuJKUA9LYeckHZCUxvjQipJDd7Rmmw',
  'u': 'http://www.nasa.gov/mission_pages/nustar/main/index.html',
  't': 1331923254,
  'hc': 1331922854},
 {'a': 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.11 (KHTML, like Gecko)
Chrome/17.0.963.79 Safari/535.11',
  'c': None,
  'nk': 0,
  'tz': '',
  'g': 'zCaLwp',
  'h': 'zUtuOu',
  'l': 'alelex88',
  'al': 'pt-BR,pt;q=0.8,en-US;q=0.6,en;q=0.4',
  'hh': '1.usa.gov',
  'r': 'http://t.co/o1PdOWeV',
  'u': 'http://apod.nasa.gov/apod/ap120312.html',
  't': 1331923255,
  'hc': 1331923068}]
```

파이썬으로 표준시간대 세기(1)

- **키 tz 없는 행 처리**
  - 키가 있는 것만
    - **time_zones에 추가**
  - 키 'tz'의 값으로
    - **''도 많음**
  - 총 3560개
    - **키 'tz'가 있는 것은**
      - **3440 개**
    - **키 'tz'가 없는 것**
      - **120개**

```
In [29]: time_zones = [rec['tz'] for rec in records if 'tz' in rec]
         time_zones[:10]

Out[29]: ['America/New_York',
          'America/Denver',
          'America/New_York',
          'America/Sao_Paulo',
          'America/New_York',
          'America/New_York',
          'Europe/Warsaw',
          '',
          '',
          '']

In [30]: len(time_zones)

Out[30]: 3440

In [31]: time_zones2 = [rec.get('tz', None) for rec in records]
         len(time_zones2)

Out[31]: 3560
```

타임존이 있는 갯수

타임존이 없는 None도 추가

defaultdict와 counter

- **표준 패키지 collections**
  - defaultdict
    - **딕셔너리(dictionary)와 거의 비슷하지만 키 값이 없을 경우 미리 지정해 놓은 초기(default) 값을 지정하는 dictionary**
  - Counter
    - **컨테이너에 동일한 값의 자료가 몇개인지를 파악하는데 사용하는 객체**
      - **A Counter is a dict subclass for counting hashable objects.**
    - **collections.Counter()의 결과값(return)은 딕셔너리 형태**
    - **메소드 most_common(n)**
      - **빈도 수 내림차순으로**

```
In [9]:
from collections import defaultdict
s = ['a', 'b', 'c', 'b', 'a', 'b', 'c']
d = defaultdict(int)

for k in s:
    d[k] += 1
d

Out[9]: defaultdict(int, {'a': 2, 'b': 3, 'c': 2})
```

```
In [20]:
c = Counter(s)
c

Out[20]: Counter({'a': 2, 'b': 3, 'c': 2})
```

```
In [13]:
from collections import Counter
lst = ['aa', 'cc', 'dd', 'aa', 'bb', 'ee']
print(Counter(lst))

Counter({'aa': 2, 'cc': 1, 'dd': 1, 'bb': 1, 'ee': 1})
```

```
In [17]: Counter({'가': 3, '나': 2, '다': 4})

Out[17]: Counter({'가': 3, '나': 2, '다': 4})
```

파이썬으로 표준시간대 세기(2)

- **사전 counts에는 표준 시간대 수가 저장**

```
In [32]:
def get_counts(sequence):
    counts = {}
    for x in sequence:
        if x in counts:
            counts[x] += 1
        else:
            counts[x] = 1
    return counts
```

```
In [33]:
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 [22]:
counts = get_counts(time_zones)
counts
```

```
Out[22]:
{'America/New_York': 1251,
 'America/Denver': 191,
 'America/Sao_Paulo': 33,
 'Europe/Warsaw': 16,
 '': 521,
 'America/Los_Angeles': 382,
 'Asia/Hong_Kong': 10,
 'Europe/Rome': 27,
```

```
In [34]:
counts['America/New_York']
```

```
Out[34]:
1251
```

```
In [35]:
len(time_zones)
```

```
Out[35]:
3440
```

딕셔너리(dictionary)와 거의 비슷하지만 키 값이 없을 경우 미리 지정해 놓은 초기(default) 값을 지정하는 dictionary

파이썬으로 표준시간대 세기(3)

- 상위 10 개의 표준시간대

```
In [38]: 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 [39]: top_counts(counts)
```

```
Out[39]: [(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')]
```

- 표준 라이브러리 collections.Counter 사용

```
In [40]: from collections import Counter
         counts = Counter(time_zones)
         counts.most_common(10)
```

```
Out[40]: [('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)]
```

데이터프레임으로 보기

- **메소드 info()**
  - 데이터프레임의 요약 정보
    - **행 3560, 열 18 개**

In [45]:
```
import pandas as pd
frame = pd.DataFrame(records)
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 [48]:
```
frame
```

Out[48]:

	a	c	nk	tz	gr	g	h	l	al	hh	
0	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKi...	US	1.0	America/New_York	MA	A6qOVH	wfLQtf	orofrog	en-US,en;q=0.8	1.usa.gov	htt
1	GoogleMaps/RochesterNY	US	0.0	America/Denver	UT	mwszkS	mwszkS	bitly	NaN	j.mp	
2	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT ...	US	1.0	America/New_York	DC	xxr3Qb	xxr3Qb	bitly	en-US	1.usa.gov	
3	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8)...	BR	0.0	America/Sao_Paulo	27	zCaLwp	zUtuOu	alelex88	pt-br	1.usa.gov	
4	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKi...	US	0.0	America/New_York	MA	9b6kNl	9b6kNl	bitly	en-US,en;q=0.8	bit.ly	
...	...	...	...	...	...	...	...	...	...	...	
3555	Mozilla/4.0 (compatible; MSIE 9.0; Windows NT ...	US	1.0	America/New_York	NJ	e5SvKE	fqPSr9	tweetdeckapi	en	1.usa.gov	
3556	Mozilla/5.0 (Windows NT 5.1) AppleWebKit/535.1...	US	0.0	America/Chicago	OK	jQLtP4	jQLtP4	bitly	en-US,en;q=0.8	1.usa.gov	
3557	GoogleMaps/RochesterNY	US	0.0	America/Denver	UT	mwszkS	mwszkS	bitly	NaN	j.mp	
3558	GoogleProducer	US	0.0	America/Los_Angeles	CA	zjtl4X	zjtl4X	bitly	NaN	1.usa.gov	
3559	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT ...	US	0.0	America/New_York	VA	qxKrTK	qxKrTK	bitly	en-US	1.usa.gov	

3560 rows × 18 columns

In [46]:
```
frame['tz'][:10]
```

Out[46]:
```
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
```

판다스로 표준시간대 세기

- **간단히 처리**
  - 필드 tz 아예 빠진 것은
    - **Missing으로 넣고**
  - 필드 tz가 ''인 것은
    - **시간대 이름을 unknown으로**

```
In [49]: tz_counts = frame['tz'].value_counts()
         tz_counts[:10]

Out[49]: 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
```

```
In [54]: clean_tz = frame['tz'].fillna('Missing')
         clean_tz[clean_tz == ''] = 'Unknown'
         tz_counts = clean_tz.value_counts()
         tz_counts[:10]

Out[54]: 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
```

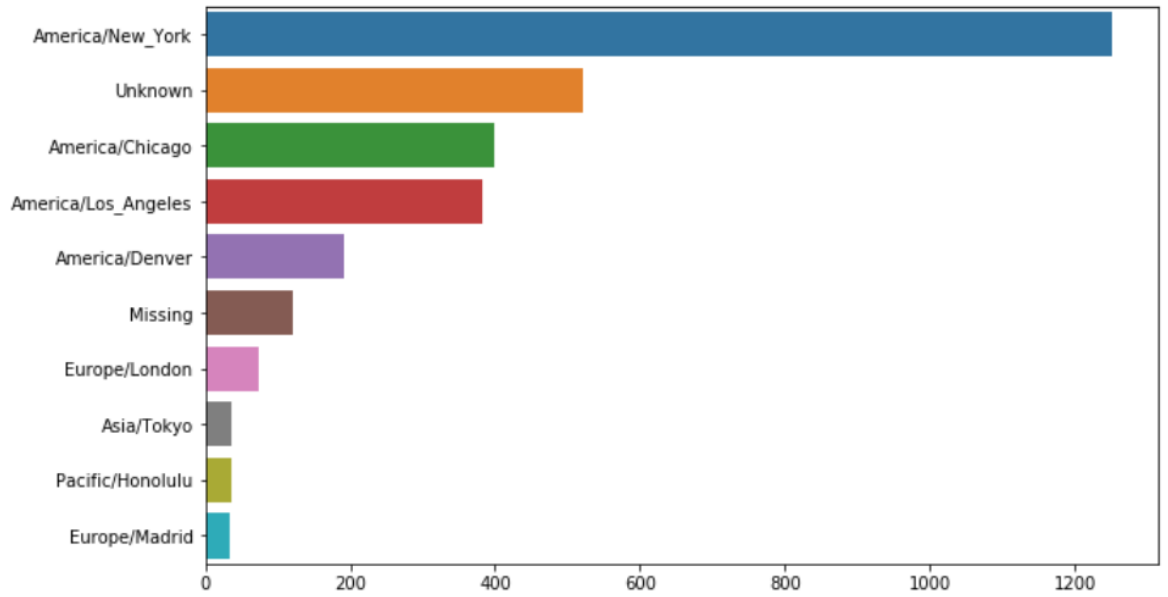
수평 막대 그리기

- **가장 많이 나타난 시간대 10개**

```
In [55]: plt.figure(figsize=(10, 4))
Out[55]: <Figure size 720x288 with 0 Axes>

         <Figure size 720x288 with 0 Axes>

In [56]: import seaborn as sns
         subset = tz_counts[:10]
         sns.barplot(y=subset.index, x=subset.values)
Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x2515b4c3f88>
```



필드 a 분석

- **URL 단축을 실행하는 정보**
  - 브라우저
  - 단말기
  - 애플리케이션
- **브라우저의 종류와 수 알기**
  - 첫 토큰(문자열)

열 'a' 자료를 분리하여 첫 번째 내용(브라우저 종류)만을 저장한 시리즈 생성

```
In [57]: frame['a'][0:2]
Out[57]: 0    Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKi...
         1                          GoogleMaps/RochesterNY
         Name: a, dtype: object

In [58]: frame['a'][1]
Out[58]: 'GoogleMaps/RochesterNY'

In [59]: frame['a'][50]
Out[59]: 'Mozilla/5.0 (Windows NT 5.1; rv:10.0.2) Gecko/20100101 Firefox/10.0.2'

In [60]: frame['a'][51][:50]  # long line
Out[60]: 'Mozilla/5.0 (Linux; U; Android 2.2.2; en-us; LG-P9'

In [61]: results = pd.Series([x.split()[0] for x in frame.a.dropna()])
         results[:5]
Out[61]: 0               Mozilla/5.0
         1    GoogleMaps/RochesterNY
         2               Mozilla/4.0
         3               Mozilla/5.0
         4               Mozilla/5.0
         dtype: object

In [63]: results.value_counts()[:8]
Out[63]: 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
```

시간대와 윈도 사용자

- **표준 시간대를**
  - 윈도/비윈도 사용자로 비교
    - **필드 a(agent 문자열)에 Windows 포함 여부에 따라**

```
In [65]: cframe = frame[frame.a.notnull()]

In [66]: cframe = cframe.copy()

In [67]: cframe['os'] = np.where(cframe['a'].str.contains('Windows'),
                                 'Windows', 'Not Windows')
         cframe['os'][:5]

Out[67]: 0        Windows
         1    Not Windows
         2        Windows
         3    Not Windows
         4        Windows
         Name: os, dtype: object
```

In [80]: cframe

Out[80]:

	a	c	nk
0	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKi...	US	1.0
1	GoogleMaps/RochesterNY	US	0.0
2	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT ...	US	1.0
3	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8)...	BR	0.0
4	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKi...	US	0.0
...	...	...	...
3555	Mozilla/4.0 (compatible; MSIE 9.0; Windows NT ...	US	1.0
3556	Mozilla/5.0 (Windows NT 5.1) AppleWebKit/535.1...	US	0.0
3557	GoogleMaps/RochesterNY	US	0.0
3558	GoogleProducer	US	0.0
3559	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT ...	US	0.0

3440 rows × 18 columns

```
In [85]: by_tz_os = cframe.groupby(['tz', 'os'])
         by_tz_os.size()

Out[85]: tz                          os
                                     Not Windows    245
                                     Windows        276
         Africa/Cairo                Windows          3
         Africa/Casablanca           Windows          1
         Africa/Ceuta                Windows          2
                                                    ...
         Europe/Warsaw               Windows         15
         Europe/Zurich               Not Windows      4
         Pacific/Auckland            Not Windows      3
                                     Windows          8
         Pacific/Honolulu            Windows         36
         Length: 149, dtype: int64
```

```
In [68]: by_tz_os = cframe.groupby(['tz', 'os'])

In [69]: agg_counts = by_tz_os.size().unstack().fillna(0)
         agg_counts[:10]

Out[69]:
```

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

''으로 지정된 시간대의 수

데이터프레임 정렬

- **메소드 sort_values(by='열명', ascending=False)**

```
In [152]:
data = {
    "도시": ["서울", "서울", "서울", "부산", "부산", "부산", "인천", "인천"],
    "연도": ["2015", "2010", "2005", "2015", "2010", "2005", "2015", "2010"],
    "인구": [9904312, 9631482, 9762546, 3448737, 3393191, 3512547, 2890451, 263203],
    "지역": ["수도권", "수도권", "수도권", "경상권", "경상권", "경상권", "수도권", "
}
columns = ["도시", "연도", "인구", "지역"]
df1 = pd.DataFrame(data, columns=columns)
df1
```

Out[152]:

	도시	연도	인구	지역
0	서울	2015	9904312	수도권
1	서울	2010	9631482	수도권
2	서울	2005	9762546	수도권
3	부산	2015	3448737	경상권
4	부산	2010	3393191	경상권
5	부산	2005	3512547	경상권
6	인천	2015	2890451	수도권
7	인천	2010	263203	수도권

```
In [161]: df1.sort_values(by=['연도', '인구'], ascending=False)
```

Out[161]:

	도시	연도	인구	지역
0	서울	2015	9904312	수도권
3	부산	2015	3448737	경상권
6	인천	2015	2890451	수도권
1	서울	2010	9631482	수도권
4	부산	2010	3393191	경상권
7	인천	2010	263203	수도권
2	서울	2005	9762546	수도권
5	부산	2005	3512547	경상권

```
In [159]: df1.인구.sort_values()
```

```
Out[159]: 7     263203
          6    2890451
          4    3393191
          3    3448737
          5    3512547
          1    9631482
          2    9762546
          0    9904312
          Name: 인구, dtype: int64
```

```
)]: df1.sort_values(by='인구', ascending=False)
```

	도시	연도	인구	지역
0	서울	2015	9904312	수도권
2	서울	2005	9762546	수도권
1	서울	2010	9631482	수도권
5	부산	2005	3512547	경상권
3	부산	2015	3448737	경상권
4	부산	2010	3393191	경상권
6	인천	2015	2890451	수도권
7	인천	2010	263203	수도권


```
In [159]: df1.인구.sort_values()
```

```
Out[159]: 7     263203
          6    2890451
          4    3393191
          3    3448737
          5    3512547
          1    9631482
          2    9762546
          0    9904312
          Name: 인구, dtype: int64
```

데이터프레임 정렬 요약

[Python pandas DataFrame]

Sort by value → Select Top N per Group

index	group	value
0	A	1
1	A	3
2	A	4
3	A	5
4	A	2
5	B	7
6	B	10
7	B	8
8	B	9
9	B	6

```
df2 = df.sort_values(by="val",
                     ascending=False).₩
      groupby("grp").head(3)
```

```
df2.sort_values(by=["grp", "val"],
                ascending=[True, False])
```

http://rfriend.tistory.com

index	group	value
3	A	5
2	A	4
1	A	3
6	B	10
8	B	9
7	B	8

메소드 argsort()

- **값을 (기본) 올림차순으로 정렬한 인덱스 시리즈를 반환**
  - 첫 번째 행은 순위 7

	도시	연도	인구	지역
0	서울	2015	9904312	수도권
1	서울	2010	9631482	수도권
2	서울	2005	9762546	수도권
3	부산	2015	3448737	경상권
4	부산	2010	3393191	경상권
5	부산	2005	3512547	경상권
6	인천	2015	2890451	수도권
7	인천	2010	263203	수도권

```
In [157]: indexer = df1.인구.argsort()
          indexer

Out[157]: 0    7
          1    6
          2    4
          3    3
          4    5
          5    1
          6    2
          7    0
          Name: 인구, dtype: int64

In [158]: df1.take(indexer)

Out[158]:
```

	도시	연도	인구	지역
7	인천	2010	263203	수도권
6	인천	2015	2890451	수도권
4	부산	2010	3393191	경상권
3	부산	2015	3448737	경상권
5	부산	2005	3512547	경상권
1	서울	2010	9631482	수도권
2	서울	2005	9762546	수도권
0	서울	2015	9904312	수도권

전체 표준시간대 순위

- **먼저 표준시간대 합 구하고**
  - 순위의 arg 구하기
  - 순위 첨자인 indexer로 자료를 take
    - **마지막 10개 가장 큰 값**

```
In [129]: count_subset = agg_counts.take(indexer[-10:])
          count_subset
```

Out[129]:

os / tz	Not Windows	Windows
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

```
In [91]: agg_counts
```

Out[91]:

os / tz	Not Windows	Windows
	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
...	...	...
Europe/Volgograd	0.0	1.0
Europe/Warsaw	1.0	15.0
Europe/Zurich	4.0	0.0
Pacific/Auckland	3.0	8.0
Pacific/Honolulu	0.0	36.0

97 rows × 2 columns

```
In [72]: # Use to sort in ascending order
         indexer = agg_counts.sum(1).argsort()
         indexer[:10]
```

Out[72]:

```
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
```

시간대의 수로 정렬한 첨자를 준비, 가장 작은 10개 출력

전체 표준시간대 순위 간단히

- **판다스의 nlargest()**
  - 다음으로 간단히 처리

```
In [130]: agg_counts.sum(1).nlargest(10)

Out[130]: 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
```

```
In [101]: agg_counts.take(agg_counts.sum(1).argsort()[:-11:-1])

Out[101]:
```

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

중첩 막대 그래프

```
In [103]: count_subset
```

Out[103]:

tz	os	Not Windows	Windows
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

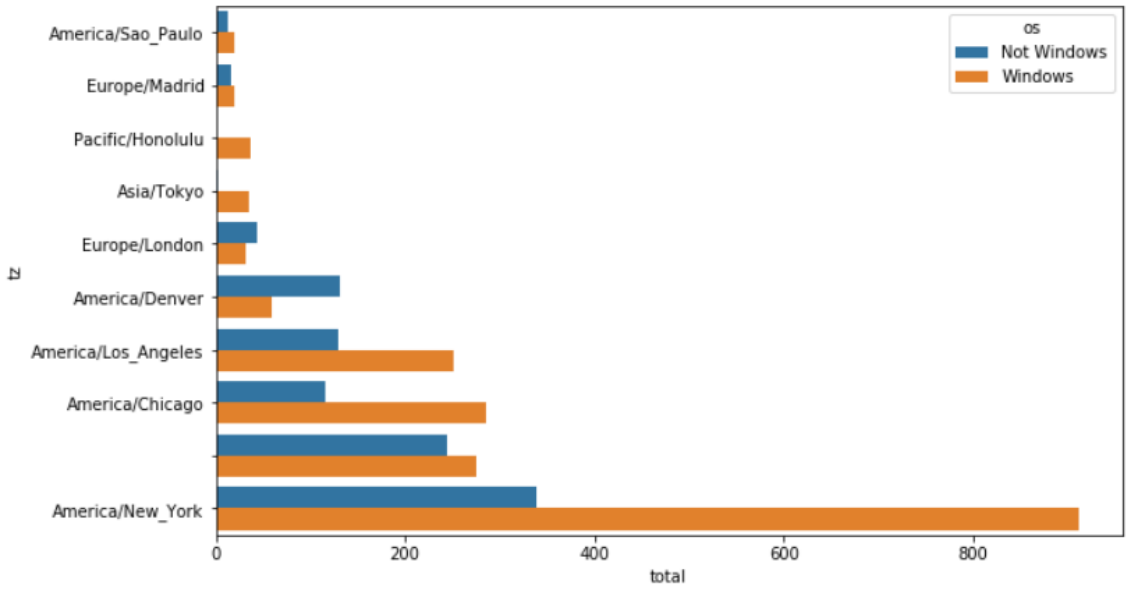
```
In [165]: # Rearrange the data for plotting
count_subset = count_subset.stack()
count_subset.name = 'total'
count_subset = count_subset.reset_index()
count_subset[:10]
```

Out[165]:

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

```
Out[166]: <matplotlib.axes._subplots.AxesSubplot at 0x25157a3bf08>
```



시간대를 모두 정규화시킨 그래프

- **시간대 사용자 총합을 1로 한 정규화된 그래프**

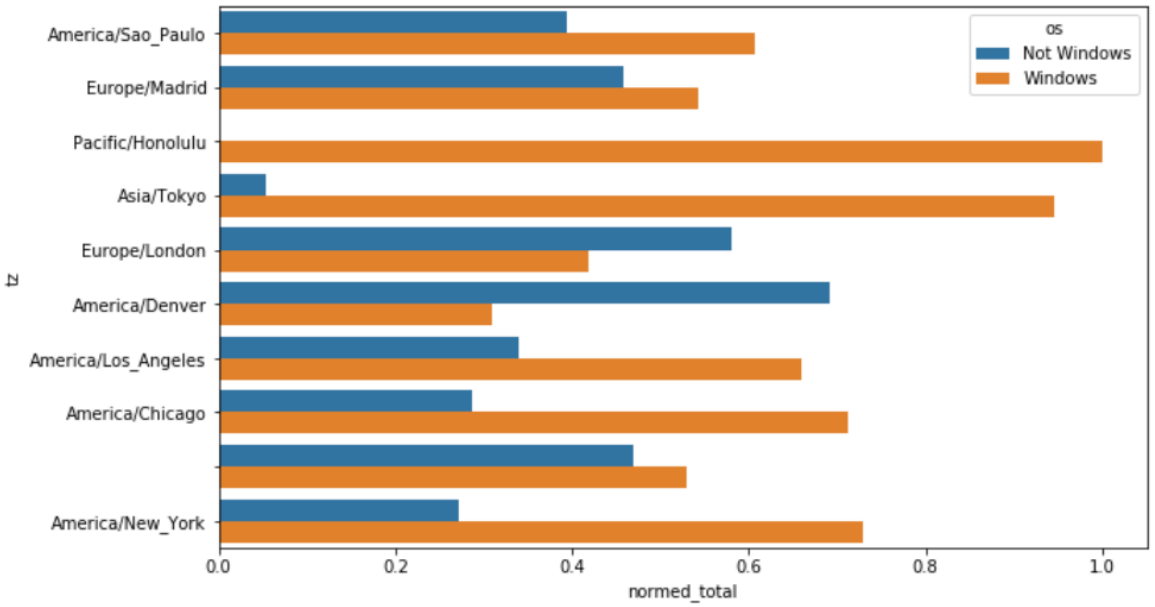
```
In [77]: def norm_total(group):
             group['normed_total'] = group.total / group.total.sum()
             return group

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

In [78]: plt.figure()
Out[78]: <Figure size 720x432 with 0 Axes>

         <Figure size 720x432 with 0 Axes>

In [79]: sns.barplot(x='normed_total', y='tz', hue='os',  data=results)
Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x2515b9366c8>
```



정규화 계산 효율화

- 메소드 groupby와 transform 사용

```
In [124]: count_subset
```

Out[124]:

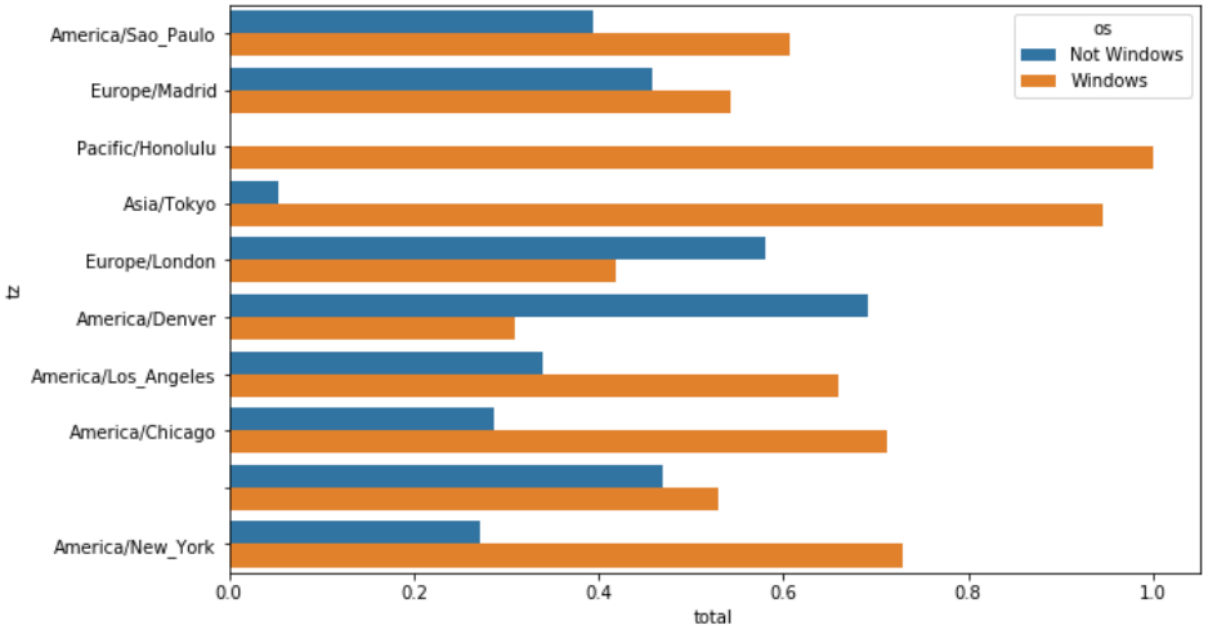
	tz	os	total
0	America/Sao_Paulo	Not Windows	13.0
1	America/Sao_Paulo	Windows	20.0
2	Europe/Madrid	Not Windows	16.0

```
In [89]: g = count_subset.groupby('tz')
         results2 = count_subset.total / g.total.transform('sum')
         results2
```

```
Out[89]: 0     0.393939
         1     0.606061
         2     0.457143
         3     0.542857
         4     0.000000
         5     1.000000
         6     0.054054
         7     0.945946
         8     0.581081
         9     0.418919
         10    0.691099
         11    0.308901
         12    0.340314
         13    0.659686
         14    0.287500
         15    0.712500
         16    0.470250
         17    0.529750
         18    0.270983
         19    0.729017
         Name: total, dtype: float64
```

```
In [90]: sns.barplot(x=results2, y='tz', hue='os',  data=results)
```

```
Out[90]: <matplotlib.axes._subplots.AxesSubplot at 0x2515bbb1308>
```



메소드 transform()

- 그룹별 대표 값을 만드는 것이 아니라 그룹별 계산을 통해 데이터프레임 자체를 변화
- 만들어진 데이터프레임의 크기는 원래 데이터프레임과 같음

```
In [132]: df = pd.DataFrame({'Year': [1997, 1997, 1997, 1998, 1999],
                             'Japan': [100, 100, 300, 200, 100],
                             'USA': [200, 100, 300, 400, 500],
                             'Canada': [400, 300, 200, 100, 400]},
                            index=['1', '2', '3', '4', '5'])
          df
```

Out[132]:

	Year	Japan	USA	Canada
1	1997	100	200	400
2	1997	100	100	300
3	1997	300	300	200
4	1998	200	400	100
5	1999	100	500	400

```
In [131]: df.groupby('Year').transform(np.sum)
```

Out[131]:

	Japan	USA	Canada
1	500	600	900
2	500	600	900
3	500	600	900
4	200	400	100
5	100	500	400

```
In [133]: df.groupby('Year').sum()
```

Out[133]:

Year	Japan	USA	Canada
1997	500	600	900
1998	200	400	100
1999	100	500	400

14장 데이터 분석 예제

MovieLens의 영화 평점 데이터 처리

1h

GroupLens 연구소의 영화 평점 데이터

- **1990년대 말부터 2000년 초**
  - 약 6천 여명으로부터 4천 여 편의 영화에 대한 백만 개의 영화 평점
  - 사용자, 영화, 평점 정보의 3개의 파일 제공
    - **datasets/movielens/users.dat**
    - **datasets/movielens/ratings.dat**
    - **datasets/movielens/movies.dat**

사용자 정보

- **열**
  - 사용자: user_id
  - 성별: gender
  - 나이: age
  - 직업: occupation
  - 우편번호: zip

In [158]:
```
unames = ['user_id', 'gender', 'age', 'occupation', 'zip']
users = pd.read_table('datasets/movielens/users.dat', sep='::',
                      header=None, names=unames, skiprows=1, engine='python')
users
```

Out[158]:

	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
...	...	...	...	...	...
6035	6036	F	25	15	32603
6036	6037	F	45	1	76006
6037	6038	F	56	1	14706
6038	6039	F	45	0	01060
6039	6040	M	25	6	11106

6040 rows × 5 columns

In [159]:
```
users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   user_id     6040 non-null   int64 
 1   gender      6040 non-null   object
 2   age         6040 non-null   int64 
 3   occupation  6040 non-null   int64 
 4   zip         6040 non-null   object
dtypes: int64(3), object(2)
memory usage: 236.1+ KB
```

평점 정보

- **열 정보**
  - 사용자: user_id
  - 영화ID: movie_id
  - 평점: rating
  - 시간: timestamp

In [160]:
```
rnames = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_table('datasets/movielens/ratings.dat', sep='::',
                        header=None, names=rnames, skiprows=1, engine='python')
ratings
```

Out[160]:

	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 × 4 columns

In [161]:
```
ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 4 columns):
 #   Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0   user_id    1000209 non-null  int64
 1   movie_id   1000209 non-null  int64
 2   rating     1000209 non-null  int64
 3   timestamp  1000209 non-null  int64
dtypes: int64(4)
memory usage: 30.5 MB
```

영화 정보

- **열 정보**
  - 영화ID: movie_id
  - 제목: title
  - 장르: genres

In [162]:
```
mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table('datasets/movielens/movies.dat', sep='::',
                       header=None, names=mnames, skiprows=1, engine='python')
movies
```

Out[162]:

	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
...	...	...	...
3878	3948	Meet the Parents (2000)	Comedy
3879	3949	Requiem for a Dream (2000)	Drama
3880	3950	Tigerland (2000)	Drama
3881	3951	Two Family House (2000)	Drama
3882	3952	Contender, The (2000)	Drama|Thriller

3883 rows × 3 columns

In [163]:
```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3883 entries, 0 to 3882
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   movie_id  3883 non-null   int64
 1   title     3883 non-null   object
 2   genres    3883 non-null   object
dtypes: int64(1), object(2)
memory usage: 91.1+ KB
```

3개의 DataFrame을 병합

- **공통된 열로 병합: 중복되는 열 이름을 키로 조인**
  - 먼저 ratings, users를 합병 후, 다시 결과와 movies를 합병
    - **총 10개의 열이 생김**

```
In [170]: data = pd.merge(pd.merge(ratings, users), movies)
          data.head()
```

Out[170]:

	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

```
In [171]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns (total 10 columns):
 #   Column      Non-Null Count    Dtype
---  ------      --------------    -----
 0   user_id     1000209 non-null  int64
 1   movie_id    1000209 non-null  int64
 2   rating      1000209 non-null  int64
 3   timestamp   1000209 non-null  int64
 4   gender      1000209 non-null  object
 5   age         1000209 non-null  int64
 6   occupation  1000209 non-null  int64
 7   zip         1000209 non-null  object
 8   title       1000209 non-null  object
 9   genres      1000209 non-null  object
dtypes: int64(6), object(4)
memory usage: 83.9+ MB
```

여러 정보 분석

- 성별에 따른 평균 평점

```
In [173]: mean_ratings = data.pivot_table('rating', index='title',
                                          columns='gender', aggfunc='mean')
          mean_ratings[:5]
```

Out[173]:

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

- 영화 제목에 따른 평점 건수

```
In [176]: ratings_by_title = data.groupby('title').size()
          ratings_by_title[:5]
```

Out[176]:

```
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
dtype: int64
```

- 평점 건수가 250 개 이상인 영화
  - 인덱스만 저장

```
In [177]: active_titles = ratings_by_title.index[ratings_by_title >= 250]
          active_titles[:5]
```

Out[177]:

```
Index(['''burbs, The (1989)', '10 Things I Hate About You (1999)',
       '101 Dalmatians (1961)', '101 Dalmatians (1996)',
       '12 Angry Men (1957)'],
      dtype='object', name='title')
```

주요 영화 중, 여성에게 높은 평점을 받은 영화 목록

- **평점 건수가 250 개 이상인 영화 제목에 따른 성별 평점 평균**
  - 목록 active_titles을 인덱스로 사용
- **여성에게 높은 평점을 받은 영화 목록**
  - 열 F를 내림차순으로 정렬

```
In [177]: active_titles = ratings_by_title.index[ratings_by_title >= 250]
          active_titles[:5]

Out[177]: Index(['''burbs, The (1989)', '10 Things I Hate About You (1999)',
                 '101 Dalmatians (1961)', '101 Dalmatians (1996)',
                 '12 Angry Men (1957)'],
                dtype='object', name='title')
```

```
In [178]: # Select rows on the index
          mean_ratings = mean_ratings.loc[active_titles]
          mean_ratings[:5]
```

Out[178]:

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

```
In [179]: top_female_ratings = mean_ratings.sort_values(by='F', ascending=False)
          top_female_ratings[:10]
```

Out[179]:

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 (1996)	4.563107	4.385075
Schindler's List (1993)	4.562602	4.491415
Shawshank Redemption, The (1994)	4.539075	4.560625
Grand Day Out, A (1992)	4.537879	4.293255
To Kill a Mockingbird (1962)	4.536667	4.372611
Creature Comforts (1990)	4.513889	4.272277
Usual Suspects, The (1995)	4.513317	4.518248

남녀 간의 호불호가 갈리는 영화

- **열 'diff'**
  - 평균 평점 차를 저장하는 칼럼 추가

In [180]:
```
mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']
mean_ratings.head()
```

Out[180]:

title	F	M	diff
'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

- **성별 선호, 상위 5개**
  - 여자가 선호
    - **열 diff로 정렬**
  - 남자가 선호
    - **열 diff로 역정렬**

In [182]:
```
# 여자가 선호하는 영화
sorted_by_diff = mean_ratings.sort_values(by='diff')
sorted_by_diff.head()
```

title	F	M	diff
Dirty Dancing (1987)	3.790378	2.959596	-0.830782
Jumpin' Jack Flash (1986)	3.254717	2.578358	-0.676359
Grease (1978)	3.975265	3.367041	-0.608224
Little Women (1994)	3.870588	3.321739	-0.548849
Steel Magnolias (1989)	3.901734	3.365957	-0.535777

In [184]:
```
# Reverse order of rows, take first 10 rows
sorted_by_diff[::-1][:5]
```

Out[184]:

title	F	M	diff
Good, The Bad and The Ugly, The (1966)	3.494949	4.221300	0.726351
Kentucky Fried Movie, The (1977)	2.878788	3.555147	0.676359
Dumb & Dumber (1994)	2.697987	3.336595	0.638608
Longest Day, The (1962)	3.411765	4.031447	0.619682
Cable Guy, The (1996)	2.250000	2.863787	0.613787

성별에 관계 없이 극명한 호불호가 있는 영화

- **호불호 측정**
  - 표준편차인 std() 함수로 계산

In [185]:
```
# Standard deviation of rating grouped by title
rating_std_by_title = data.groupby('title')['rating'].std()
rating_std_by_title.head()
```

Out[185]:
```
title
$1,000,000 Duck (1971)              1.092563
'Night Mother (1986)                1.118636
'Til There Was You (1997)           1.020159
'burbs, The (1989)                  1.107760
...And Justice for All (1979)       0.878110
Name: rating, dtype: float64
```

In [187]:
```
# Filter down to active_titles
rating_std_by_title = rating_std_by_title.loc[active_titles]
rating_std_by_title.head()
```

Out[187]:
```
title
'burbs, The (1989)                      1.107760
10 Things I Hate About You (1999)       0.989815
101 Dalmatians (1961)                   0.982103
101 Dalmatians (1996)                   1.098717
12 Angry Men (1957)                     0.812731
Name: rating, dtype: float64
```

In [188]:
```
# Order Series by value in descending order
rating_std_by_title.sort_values(ascending=False)[:10]
```

Out[188]:
```
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
```