



# 데이터 처리/분석-4

엄진영

# 그룹화된 산점도 그래프 작성하기

- anime\_master.csv파일을 이용하여 애니메이션 작품의 배급 종별로 그룹화한 산점도 그래프

```
df = pd.read_csv('C:/Users/jyeon/anime_master.csv', encoding="utf-8", index_col = 'anime_id')
df.head()
```

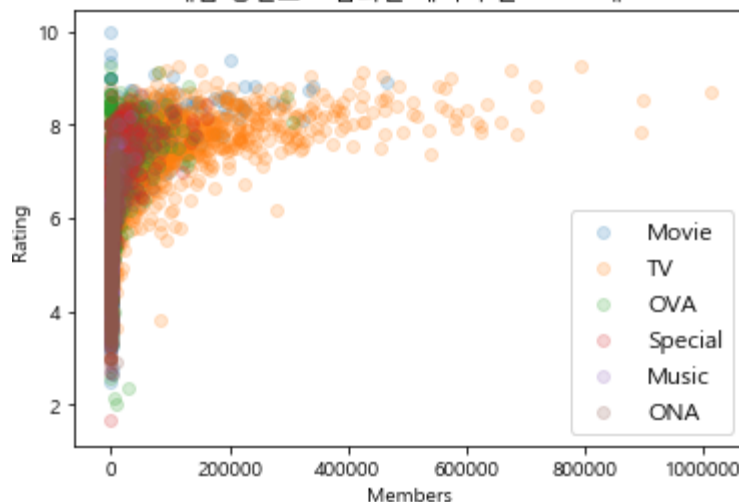
| anime_id | name                             | genre   | type  | episodes | rating | members |
|----------|----------------------------------|---|-------|----------|--------|---------|
| 32281    | Kimi no Na wa.                   | Drama, Romance, School, Supernatural              | Movie | 1        | 9.37   | 200630  |
| 5114     | Fullmetal Alchemist: Brotherhood | Action, Adventure, Drama, Fantasy, Magic, Mili... | TV    | 64       | 9.26   | 793665  |
| 28977    | Gintama°                         | Action, Comedy, Historical, Parody, Samurai, S... | TV    | 51       | 9.25   | 114262  |
| 9253     | Steins;Gate                      | Sci-Fi, Thriller                                  | TV    | 24       | 9.17   | 673572  |
| 9969     | Gintama'                         | Action, Comedy, Historical, Parody, Samurai, S... | TV    | 51       | 9.16   | 151266  |

```
plt.rcParams['font.family']='Malgun Gothic'
plt.rcParams['axes.unicode_minus'] = False

for t in types:
    x = df.loc[df['type'] == t, 'members']
    y = df.loc[df['type'] == t, 'rating']
    plt.scatter(x, y, alpha=0.2, label=t)

plt.title('배급 종별로 그룹화된 데이터 산포도 그래프')
plt.xlabel('Members')
plt.ylabel('Rating')
plt.legend(loc='lower right', fontsize = 12)
plt.show()
```

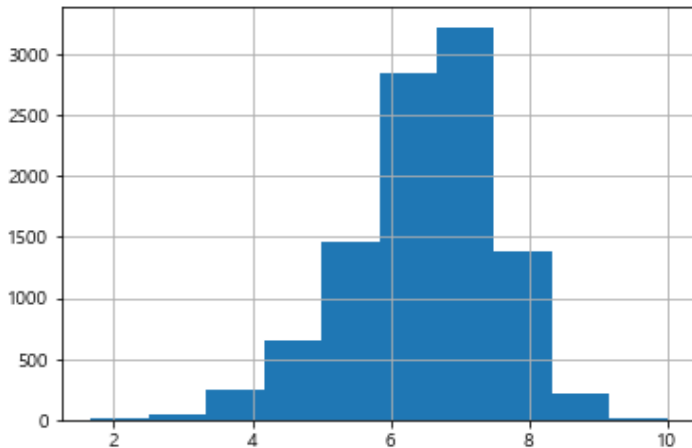
배급 종별로 그룹화된 데이터 산포도 그래프



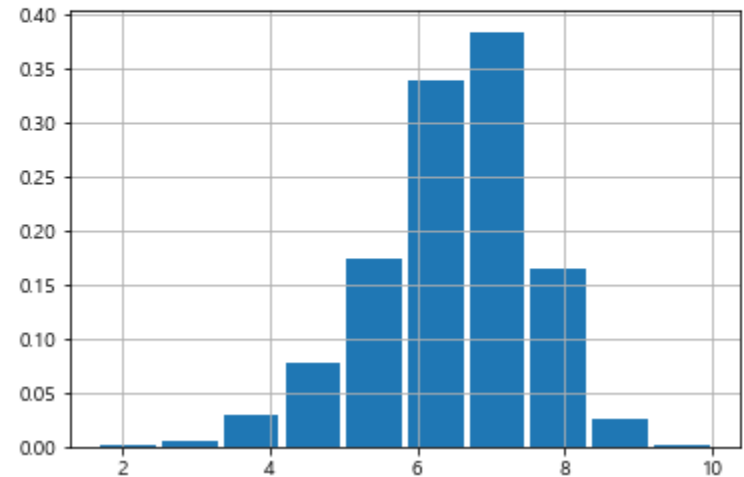
# 상대도수 히스토그램

- 데이터 수가 다른 그룹의 히스토그램을 비교하는 경우에는 상대도수를 이용해서 히스토그램화하면 비교가 용이

```
df = pd.read_csv('C:/Users/jyeon/anime_master.csv', encoding="utf-8")  
  
plt.hist(df['rating'])  
plt.grid()  
plt.show()
```



```
plt.hist(df['rating'], density=True, rwidth = 0.9 )  
plt.grid()  
plt.show()
```



# 여러 그룹의 막대그래프 활용

- anime\_genre\_top10\_pivoted.csv 파일을 이용하여 배급 종별, 장르별 막대그래프

```
df = pd.read_csv('C:/Users/jyeon/anime_genre_top10_pivoted.csv', encoding="utf-8")  
df.head()
```

|   | genre        | Movie                      | Music                    | ONA                      | OVA                       | Special                   | TV                         |
|---|--------------|----------------------------|--------------------------|--------------------------|---------------------------|---------------------------|----------------------------|
| 0 | Comedy       | 7293127.0                  | 20860.0                  | 1477266.0                | 5614758.0                 | 6659293.0                 | 65420862.0                 |
| 1 | Action       | <a href="#">10224960.0</a> | 77054.0                  | <a href="#">524907.0</a> | 5793680.0                 | 3412689.0                 | <a href="#">63364032.0</a> |
| 2 | Drama        | 9034099.0                  | <a href="#">100734.0</a> | <a href="#">188427.0</a> | 3043374.0                 | <a href="#">1915578.0</a> | <a href="#">41011557.0</a> |
| 3 | Romance      | <a href="#">5245386.0</a>  | 42811.0                  | <a href="#">411331.0</a> | <a href="#">3143167.0</a> | <a href="#">2015820.0</a> | 40703388.0                 |
| 4 | Supernatural | <a href="#">5452779.0</a>  | 9189.0                   | <a href="#">192989.0</a> | <a href="#">2696715.0</a> | <a href="#">2336723.0</a> | 38956520.0                 |

```
df = pd.read_csv('C:/Users/jyeon/anime_genre_top10_pivoted.csv', encoding="utf-8", index_col = 'genre')  
df.head()
```

click to hide

|              | Movie                      | Music                    | ONA                      | OVA                       | Special                   | TV                         |
|--------------|----------------------------|--------------------------|--------------------------|---------------------------|---------------------------|----------------------------|
| genre        |                            |                          |                          |                           |                           |                            |
| Comedy       | 7293127.0                  | 20860.0                  | 1477266.0                | 5614758.0                 | 6659293.0                 | 65420862.0                 |
| Action       | <a href="#">10224960.0</a> | 77054.0                  | <a href="#">524907.0</a> | 5793680.0                 | 3412689.0                 | <a href="#">63364032.0</a> |
| Drama        | 9034099.0                  | <a href="#">100734.0</a> | <a href="#">188427.0</a> | 3043374.0                 | <a href="#">1915578.0</a> | <a href="#">41011557.0</a> |
| Romance      | <a href="#">5245386.0</a>  | 42811.0                  | <a href="#">411331.0</a> | <a href="#">3143167.0</a> | <a href="#">2015820.0</a> | 40703388.0                 |
| Supernatural | <a href="#">5452779.0</a>  | 9189.0                   | <a href="#">192989.0</a> | <a href="#">2696715.0</a> | <a href="#">2336723.0</a> | 38956520.0                 |

배급종별

장르별

# 여러 그룹의 막대그래프 활용

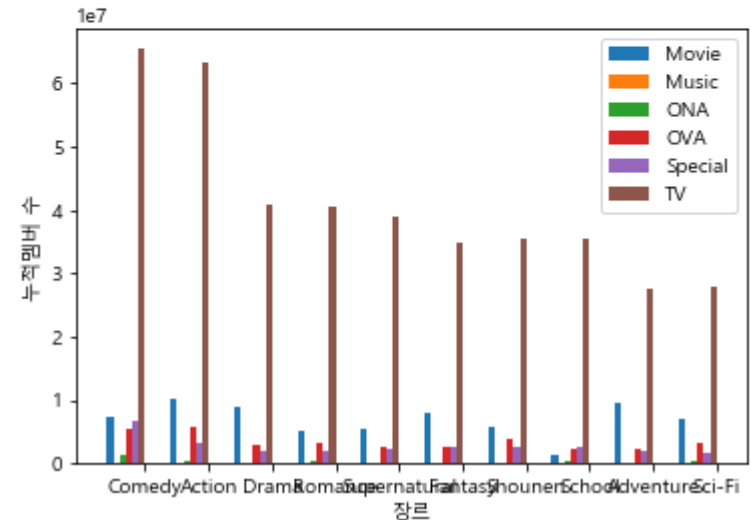
df

|              | Movie                      | Music                    | ONA                      | OVA                       | Special                   | TV                         |
|--------------|----------------------------|--------------------------|--------------------------|---------------------------|---------------------------|----------------------------|
| genre        |                            |                          |                          |                           |                           |                            |
| Comedy       | 7293127.0                  | 20860.0                  | 1477266.0                | 5614758.0                 | 6659293.0                 | 65420862.0                 |
| Action       | <a href="#">10224960.0</a> | 77054.0                  | <a href="#">524907.0</a> | 5793680.0                 | 3412689.0                 | <a href="#">63364032.0</a> |
| Drama        | 9034099.0                  | <a href="#">100734.0</a> | <a href="#">188427.0</a> | 3043374.0                 | <a href="#">1915578.0</a> | <a href="#">41011557.0</a> |
| Romance      | <a href="#">5245386.0</a>  | 42811.0                  | <a href="#">411331.0</a> | <a href="#">3143167.0</a> | <a href="#">2015820.0</a> | 40703388.0                 |
| Supernatural | <a href="#">5452779.0</a>  | 9189.0                   | <a href="#">192989.0</a> | <a href="#">2696715.0</a> | <a href="#">2336723.0</a> | 38956520.0                 |
| Fantasy      | 8019406.0                  | 43962.0                  | <a href="#">188937.0</a> | <a href="#">2754224.0</a> | <a href="#">2504131.0</a> | 34932563.0                 |
| Shounen      | 5698808.0                  | NaN                      | 97833.0                  | 3861296.0                 | <a href="#">2591988.0</a> | 35532847.0                 |
| School       | 1512533.0                  | 5496.0                   | <a href="#">523223.0</a> | <a href="#">2417660.0</a> | <a href="#">2661425.0</a> | 35489099.0                 |
| Adventure    | 9485223.0                  | 42829.0                  | 70431.0                  | <a href="#">2373765.0</a> | <a href="#">2052024.0</a> | <a href="#">27529975.0</a> |
| Sci-Fi       | 6967146.0                  | 154801.0                 | <a href="#">415311.0</a> | <a href="#">3358525.0</a> | <a href="#">1616450.0</a> | <a href="#">28072322.0</a> |

```
wt = np.array(range(len(df)))
wt
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

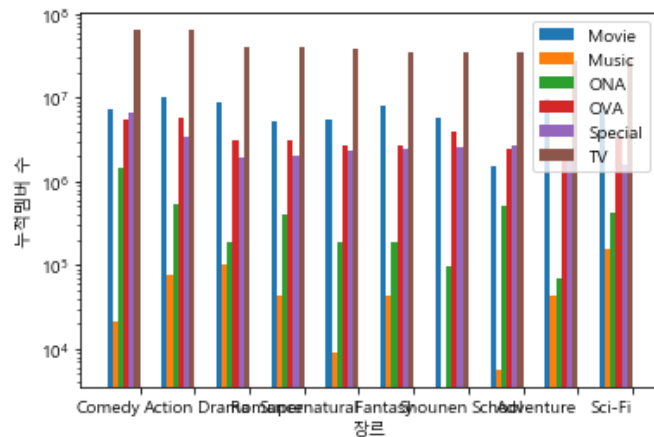
```
w = 0.1
for i in df.columns:
    plt.bar(wt, df[i], align='edge', width=w, label=i)
    wt = wt + w
plt.xticks(wt, df.index)
plt.xlabel('장르')
plt.ylabel('누적멤버 수')
plt.legend()
plt.show()
```



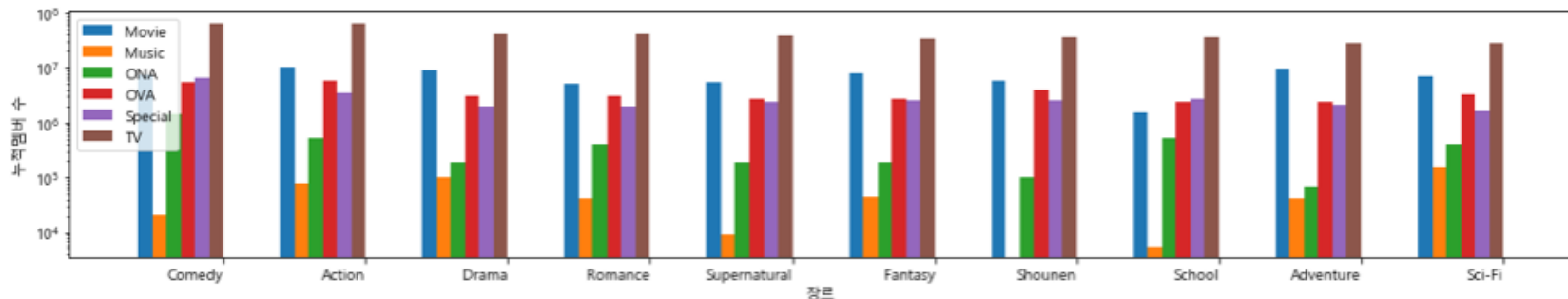
TV 멤버수가 돌출되어 있고 다음에  
Movie 멤버수가 많은 것을 확인  
Music이나 ONA값이 상대적으로 작아  
눈으로 확인하는 것이 어려움 → y축을  
로그축으로 설정하면 가독성이 좋아짐

# 여러 그룹의 막대그래프 활용

```
# y축을 로그축으로
wt = np.arange(len(df))
w = 0.1
for i in df.columns:
    plt.bar(wt, df[i], align='edge', width=w, label=i)
    wt = wt + w
plt.xticks(wt, df.index, ha='right')
plt.xlabel('장르')
plt.ylabel('누적멤버 수')
plt.yscale('log')
plt.legend()
plt.show()
```



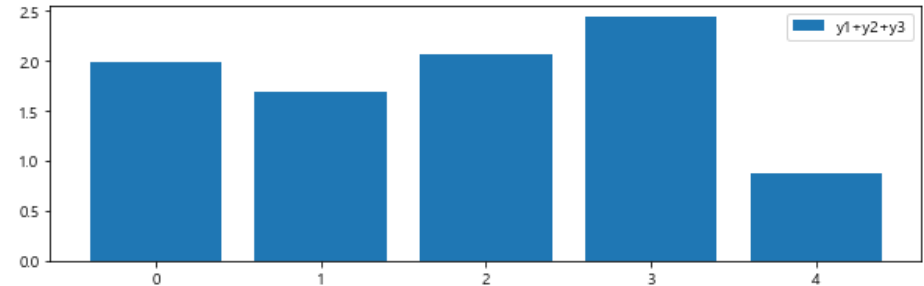
```
#그림 크기 조절
fig = plt.figure(figsize=(18,3))
wt = np.arange(len(df))
w = 0.1
for i in df.columns:
    plt.bar(wt, df[i], align='edge', width=w, label=i)
    wt = wt + w
plt.xticks(wt, df.index, ha='right')
plt.xlabel('장르')
plt.ylabel('누적멤버 수')
plt.yscale('log')
plt.legend()
plt.show()
```



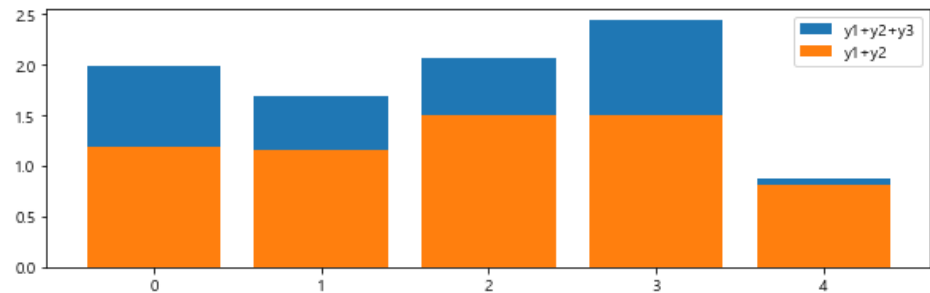
# 누적 막대그래프 생성

- 누적 막대 그래프를 그릴 때에도 여러 그룹의 막대 그래프와 같이 작성시 요령이 필요
- $y_1, y_2, y_3$ 이라는 세개의 값을 누적한 경우의 그림을 그리려면?

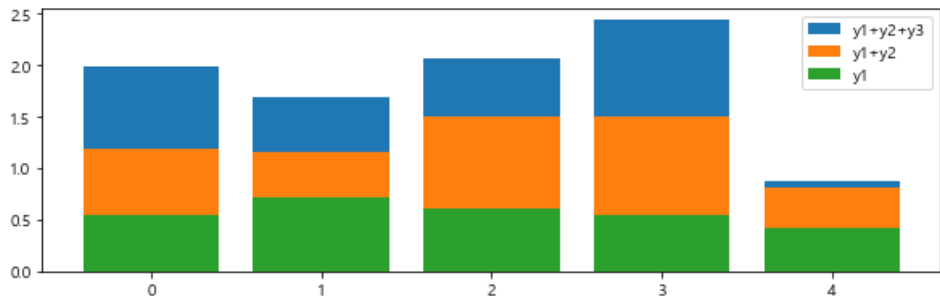
1.  $y_1$ 과  $y_2$ 와  $y_3$ 의 합을 그림



2. 1에  $y_2$ 와  $y_3$ 의 합을 겹쳐서 그림



3. 2에  $y_1$ 의 겹쳐서 그림

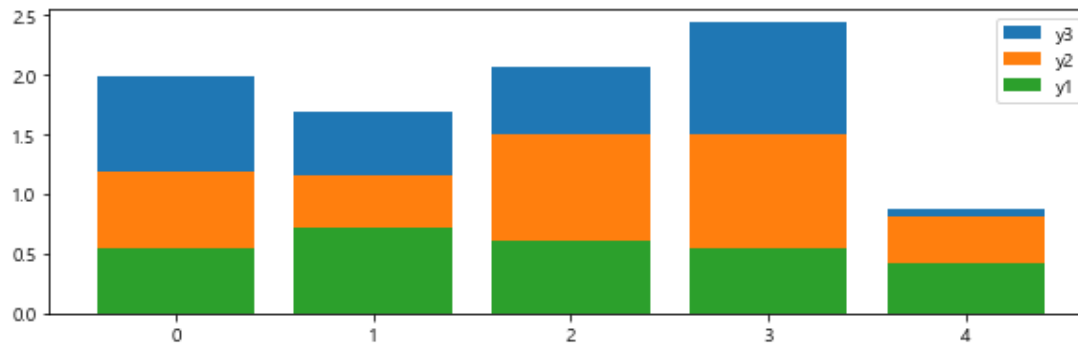


# 누적 막대그래프 생성

```
x = np.arange(5)
np.random.seed(0)
y = np.random.rand(15).reshape((3,5))
y1, y2, y3 = y
```

```
y1b = np.array(y1) # 1차원 배열로 데이터 타입을 수정
y2b = y1b + np.array(y2) # 데이터의 요소끼리 가산하기 위해서
y3b = y2b + np.array(y3)
```

```
fig = plt.figure(figsize=(10,3))
plt.bar(x, y3b, label='y3')
plt.bar(x, y2b, label='y2')
plt.bar(x, y1b, label='y1')
plt.legend()
plt.show()
```



```
x = np.array(data)
x
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [7]:

```
2 * x
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

```
np.random.seed(0); y = np.random.rand(4)
y
```

```
array([0.5488135 , 0.71518937, 0.60276338, 0.54488318])
```

```
np.random.seed(0); y = np.random.rand(4)
y
```

```
array([0.5488135 , 0.71518937, 0.60276338, 0.54488318])
```

```
y = np.random.rand(4)
y
```

```
array([0.96366276, 0.38344152, 0.79172504, 0.52889492])
```

```
y = np.random.rand(4)
y
```

```
array([0.56804456, 0.92559664, 0.07103606, 0.0871293 ])
```

```
L = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(2 * L)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```



- anime\_genre\_top10\_pivoted.csv 파일을 이용하여 배급 종별, 장르별 막대그래프를 누적 막대그래프로 그림

## Hint: enumerate

- ✓반복문 사용 시 몇 번째 반복문인지 확인이 필요할 때 사용
- ✓인덱스와 컬렉션의 원소를 tuple 형태로 반환



```
>>> t = [1, 5, 7, 33, 39, 52]
>>> for p in enumerate(t):
...     print(p)
...
(0, 1)
(1, 5)
(2, 7)
(3, 33)
(4, 39)
(5, 52)
```

```
>>> for i, v in enumerate(t):
...     print("index : {}, value: {}".format(i,v))
...
index : 0, value: 1
index : 1, value: 5
index : 2, value: 7
index : 3, value: 33
index : 4, value: 39
index : 5, value: 52
```

# Dask 시작하기

- Pandas는 데이터베이스나 csv 파일의 데이터를 모두 메모리로 읽은 후에 메모리 위에서 데이터를 처리
- pandas의 단점
  - 데이터의 양이 많은 경우에는 메모리의 제한으로 데이터프레임을 만들 수 없는 경우가 생김
  - 데이터프레임의 크기가 너무 크면 질의나 그룹 연산을 할 때 하나의 CPU 코어로 처리하기에는 시간이 너무 많이 걸릴 수도 있음
- Python 나무위키

빅 데이터, 통계학 라이브러리.

- 판다스(pandas): 데이터 처리와 분석을 위한 라이브러리이다. 테이블 형태의 데이터를 다루기 위한 데이터프레임(DataFrame) 자료형을 제공한다. R의 data.frame을 본떠서 설계한 DataFrame이라는 데이터 구조를 기반으로 만들어졌다.  [Visualizing Pandas' Pivoting and Reshaping Functions](#) 참조.
- 다스크(Dask):  홈페이지 판다스의 병렬&분산처리 버전. 판다스랑 사용법이 거의 같으면서도 드라마틱하게 빨라진다. 다만 아직 갈 길이 먼 것이 흠.
- StatsModels: 통계 및 회귀 분석, 시계열 분석을 위한 라이브러리이다.

# Dask 시작하기

- **Dask는 분석 컴퓨팅을 위한 python 병렬 컴퓨팅 라이브러리**

- Dask는 Python 작업을 가져다가 여러 시스템에서 효율적으로 일정을 조율
- Dask의 가장 유용한 기능은 Dask 작업을 실행하기 위해 사용하는 구문이 실제로 파이썬에서 다른 작업을 위해 사용하는 구문과 같기 때문에 기존의 코드를 거의 수정하지 않고도 유용하게 활용할 수 있다는 점이다.
- DataFrame객체는 pandas의 라이브러리의 것과 같은 방식으로 동작하여 코드의 몇줄만 변경하여 기존의 코드를 신속하게 병렬화 가능
- Dask를 활용하여 순수 파이썬으로 작성한 작업을 병렬화할 수 있으며 이런 유형의 작업을 최적화 하는데 적합한 객체유형(Bag 등)이 있다.

# Dask 시작하기

- Dask 패키지는 Pandas 데이터프레임 형식으로 빅데이터를 처리하기 위한 파이썬 패키지로 다음과 같은 두 가지 기능을 가짐
  1. 가상 데이터프레임
  2. 병렬처리를 위한 작업 스케줄러

# 가상 데이터프레임

- Dask 패키지를 사용하면 가상 데이터프레임을 만들 수 있음

- 가상 데이터프레임은 Pandas 데이터프레임과 비슷한 기능을 제공하지만 실제로 모든 데이터가 메모리 상에 로드되어 있는 것이 아니라 하나 이상의 파일 혹은 데이터베이스에 존재하는 채로 처리할 수 있는 기능
- 따라서 메모리 크기와 관계 없이 엄청나게 큰 CSV 파일을 가상 데이터프레임으로 로드하거나 같은 형식의 데이터를 가진 여러개의 CSV 파일을 하나의 가상 데이터프레임에 로드할 수 있음

```
In [3]: import dask.dataframe as dd # Dask 패키지의 dataframe 서브 패키지를 dd라는 이름으로
df = dd.read_csv("anime_master.csv") # read_csv 명령으로 anime_master.csv에 대한 가상데이터 프레임 df을 만들
```

Out [3]: Dask DataFrame Structure:

|               | anime_id | name   | genre  | type   | episodes | rating  | members |
|---------------|----------|--------|--------|--------|----------|---------|---------|
| npartitions=1 |          |        |        |        |          |         |         |
|               | int64    | object | object | object | int64    | float64 | int64   |
|               | ...      | ...    | ...    | ...    | ...      | ...     | ...     |

df는 데이터프레임과 유사하지만 실제로 데이터를 메모리에 읽지 않았기때문에 값은 표시 되지 않음

Dask Name: from-delayed, 3 tasks

In [4]: df.head() head, tail 명령을 내리면 일부 데이터를 읽어서 표시

Out [4]:

|   | anime_id | name                             | genre   | type  | episodes | rating | members |
|---|----------|----------------------------------|---|-------|----------|--------|---------|
| 0 | 32281    | Kimi no Na wa.                   | Drama, Romance, School, Supernatural              | Movie | 1        | 9.37   | 200630  |
| 1 | 5114     | Fullmetal Alchemist: Brotherhood | Action, Adventure, Drama, Fantasy, Magic, Mili... | TV    | 64       | 9.26   | 793665  |
| 2 | 28977    | Gintama*                         | Action, Comedy, Historical, Parody, Samurai, S... | TV    | 51       | 9.25   | 114262  |
| 3 | 9253     | Steins;Gate                      | Sci-Fi, Thriller                                  | TV    | 24       | 9.17   | 673572  |
| 4 | 9969     | Gintama'                         | Action, Comedy, Historical, Parody, Samurai, S... | TV    | 51       | 9.16   | 151266  |

# 가상 데이터프레임

## members의 평균을 구해보자.

```
In [5]: df.members.mean()
```

```
Out [5]: dd.Scalar<series-..., dtype=float64>
```

```
In [6]: df.members.mean().compute()
```

```
Out [6]: 18924.95076923077
```

```
In [8]: df=df.assign(members_mean=df.members.mean())  
df.head()
```

Out [8]:

|   | anime_id | name                             | genre   | type  | episodes | rating | members | members_mean                 |
|---|----------|----------------------------------|---|-------|----------|--------|---------|------------------------------|
| 0 | 32281    | Kimi no Na wa.                   | Drama, Romance, School, Supernatural              | Movie | 1        | 9.37   | 200630  | <a href="#">18924.950769</a> |
| 1 | 5114     | Fullmetal Alchemist: Brotherhood | Action, Adventure, Drama, Fantasy, Magic, Mili... | TV    | 64       | 9.26   | 793665  | <a href="#">18924.950769</a> |
| 2 | 28977    | Gintama*                         | Action, Comedy, Historical, Parody, Samurai, S... | TV    | 51       | 9.25   | 114262  | <a href="#">18924.950769</a> |
| 3 | 9253     | Steins;Gate                      | Sci-Fi, Thriller                                  | TV    | 24       | 9.17   | 673572  | <a href="#">18924.950769</a> |
| 4 | 9969     | Gintama'                         | Action, Comedy, Historical, Parody, Samurai, S... | TV    | 51       | 9.16   | 151266  | <a href="#">18924.950769</a> |

```
In [10]: df=df.assign(members_mean=df.members_mean.astype(str)+"명")  
df.head()
```

Out [10]:

|   | anime_id | name                             | genre   | type  | episodes | rating | members | members_mean       |
|---|----------|----------------------------------|---|-------|----------|--------|---------|--------------------|
| 0 | 32281    | Kimi no Na wa.                   | Drama, Romance, School, Supernatural              | Movie | 1        | 9.37   | 200630  | 18924.95076923077명 |
| 1 | 5114     | Fullmetal Alchemist: Brotherhood | Action, Adventure, Drama, Fantasy, Magic, Mili... | TV    | 64       | 9.26   | 793665  | 18924.95076923077명 |
| 2 | 28977    | Gintama*                         | Action, Comedy, Historical, Parody, Samurai, S... | TV    | 51       | 9.25   | 114262  | 18924.95076923077명 |
| 3 | 9253     | Steins;Gate                      | Sci-Fi, Thriller                                  | TV    | 24       | 9.17   | 673572  | 18924.95076923077명 |
| 4 | 9969     | Gintama'                         | Action, Comedy, Historical, Parody, Samurai, S... | TV    | 51       | 9.16   | 151266  | 18924.95076923077명 |

데이터 프레임과 달리 바로 결과가 나오지 않음. 그 이유는 연산 반환값이 결과가 아닌 작업(task)이기 때문이다. 구체적으로 어떤 작업을 보려면 visualize 메서드를 사용하여 작업 그래프(graph)를 볼 수 있다. 작업 그래프란 이 계산을 하기 위해 실제로 CPU가 해야 할 일들의 순서도라고 볼 수 있다. 이 작업의 값을 실제로 구하려면 결과로 받은 작업 객체의 compute 메서드를 호출해야 한다.

이 값으로 members\_mean이라는 새로운 열을 추가해본다. 이때는 pandas의 문법을 쓰지 못하고, assign 메서드를 사용 (assign 메서드를 사용할때는 compute 메서드를 사용할 필요 X)

자료형을 변환하거나 기존의 열 갱신도 가능

# 가상 데이터프레임

- 복수 데이터에 대한 가상 데이터프레임

```
%%writefile data1.csv
time,temperature,humidity
0,22,58
1,21,57
2,25,57
3,26,55
4,22,53
5,23,59
```

```
%%writefile data2.csv
time,temperature,humidity
0,22,58
1,21,57
2,25,57
3,26,55
4,22,53
5,23,59
```

```
%%writefile data3.csv
time,temperature,humidity
0,22,58
1,21,57
2,25,57
3,26,55
4,22,53
5,23,59
```

```
In [12]: df = dd.read_csv('data*.csv')
df.count().compute()
```

한번에 하나의 데이터프레임으로 읽어 들임  
복수파일은 와일드카드(\*)기호를 이용하여 읽음

```
Out [12]: time          18
temperature  18
humidity     18
dtype: int64
```

```
In [13]: df.temperature.describe().compute()
```

```
Out [13]: count      18.000000
mean       23.166667
std        1.823055
min        21.000000
25%        22.000000
50%        22.500000
75%        24.500000
max        26.000000
dtype: float64
```

# 대용량 데이터의 병렬 처리

- 샘플로 쓸 데이터는 미국 정부가 발표하는 공개 정보 중 하나로 시카고의 범죄 관련 데이터

<https://catalog.data.gov/dataset/crimes-2001-to-present-398a4>

- csv 파일의 크기가 1.69GB므로 다운로드에 10분이상 걸릴 수도 있음

```
In [18]: df = dd.read_csv("Crimes_-_2001_to_present.csv", dtype=str,  
                        error_bad_lines=False, warn_bad_lines=False)  
df
```

error\_bad\_lines 옵션과 warn\_bad\_lines을 False로 해서 오류가 나는 데이터는 생략

Out [18]: Dask DataFrame Structure:

| ID             | Case Number | Date   | Block  | IUCR   | Primary Type | Description | Location Description | Arrest | Domestic | Beat   | District | Ward   | Community Area | FBI Code | Coordina |
|----------------|-------------|--------|--------|--------|--------------|-------------|----------------------|--------|----------|--------|----------|--------|----------------|----------|----------|
| npartitions=29 |             |        |        |        |              |             |                      |        |          |        |          |        |                |          |          |
| object         | object      | object | object | object | object       | object      | object               | object | object   | object | object   | object | object         | object   | object   |
| ...            | ...         | ...    | ...    | ...    | ...          | ...         | ...                  | ...    | ...      | ...    | ...      | ...    | ...            | ...      | ...      |
| ...            | ...         | ...    | ...    | ...    | ...          | ...         | ...                  | ...    | ...      | ...    | ...      | ...    | ...            | ...      | ...      |
| ...            | ...         | ...    | ...    | ...    | ...          | ...         | ...                  | ...    | ...      | ...    | ...      | ...    | ...            | ...      | ...      |
| ...            | ...         | ...    | ...    | ...    | ...          | ...         | ...                  | ...    | ...      | ...    | ...      | ...    | ...            | ...      | ...      |

Dask Name: from-delayed, 87 tasks

< >



# 대용량 데이터의 병렬 처리

```
In [19]: df.tail()
```

Out [19]:

|        | ID      | Case Number | Date                         | Block                      | IUCR | Primary Type          | Description        | Location Description | Arrest | Domestic | ... | Longitude     | Loca                                     |
|--------|---------|-------------|------------------------------|----------------------------|------|-----------------------|--------------------|----------------------|--------|----------|-----|---------------|--|
| 117968 | 1374123 | G083329     | 01/01/2001<br>12:00:00<br>AM | 045XX S<br>LAPORTE<br>AV   | 0820 | THEFT                 | \$500 AND<br>UNDER | STREET               | false  | false    | ... | -87.747061323 | <a href="#">(41.810679)</a><br>87.747061 |
| 117969 | 1535712 | G287406     | 01/01/2001<br>12:00:00<br>AM | 100XX W<br>OHARE ST        | 1140 | DECEPTIVE<br>PRACTICE | EMBEZZLEMENT       | AIRPORT/AIRCRAFT     | true   | false    | ... | -87.905312411 | <a href="#">(41.976200)</a><br>87.905312 |
| 117970 | 1310854 | G001433     | 01/01/2001<br>12:00:00<br>AM | 022XX W<br>LUNT AV         | 1320 | CRIMINAL<br>DAMAGE    | TO VEHICLE         | OTHER                | false  | false    | ... | -87.686868567 | <a href="#">(42.008635)</a><br>87.686868 |
| 117971 | 1322501 | G016827     | 01/01/2001<br>12:00:00<br>AM | 010XX N<br>KEDVALE<br>AV   | 0820 | THEFT                 | \$500 AND<br>UNDER | STREET               | false  | false    | ... | -87.730036643 | <a href="#">(41.899046)</a><br>87.730036 |
| 117972 | 1448953 | G177802     | 01/01/2001<br>12:00:00<br>AM | 010XX W<br>GRANVILLE<br>AV | 0560 | ASSAULT               | SIMPLE             | APARTMENT            | false  | true     | ... | -87.657186625 | <a href="#">(41.994699)</a><br>87.657186 |

5 rows × 30 columns

```
In [20]: from dask.diagnostics import ProgressBar  
  
pbar = ProgressBar()  
pbar.register()
```

데이터의 크기가 큰 만큼 시간이 오래 걸리기 때문에  
Dask는 작업 진행도를 알 수 있는 ProgressBar라는 걸  
제공  
progressBar를 만들고 등록

```
In [*]: df.count().compute()
[#####] | 40% Completed | 24.4s
```

```
In [*]: df.count().compute(scheduler='processes', num_workers=4)
[#####] | 84% Completed | 34.1s
```

```
In [21]: df.count().compute()
[#####] | 100% Completed | 48.1s
```

```
Out [21]: ID 6925640
Case Number 6925636
Date 6925640
Block 6925640
IUCR 6925640
Primary Type 6925640
Description 6925640
Location Description 6920334
Arrest 6925640
Domestic 6925640
Beat 6925640
District 6925593
Ward 6310815
Community Area 6312145
FBI Code 6925640
X Coordinate 6859959
Y Coordinate 6859959
Year 6925640
Updated On 6925640
Latitude 6859959
Longitude 6859959
Location 6859959
Historical Wards 2003-2015 6840014
Zip Codes 6859959
Community Areas 6842753
Census Tracts 6844900
Wards 6842865
Boundaries - ZIP Codes 6842800
Police Districts 6843830
Police Beats 6843853
dtype: int64
```

```
In [22]: df.count().compute(scheduler='processes', num_workers=4)
[#####] | 100% Completed | 34.9s
```

```
Out [22]: ID 6925640
```

병렬 처리를 위해서는 어떤 병렬처리 방식을 사용할지 ,  
작업 프로세스의 개수는 어떻게 할지 등은 **compute** 명령에서 인수로  
설정  
멀티프로세싱을 하고, 4개의 cpu코어를 동시에 사용하도록 설정  
(이코드가 실행되는 컴퓨터가 실제로 4개 이상의 코어를 가지고 있어야  
성능이 개선)

```
Domestic 6925640
Beat 6925640
District 6925593
Ward 6310815
Community Area 6312145
FBI Code 6925640
X Coordinate 6859959
Y Coordinate 6859959
Year 6925640
Updated On 6925640
Latitude 6859959
Longitude 6859959
Location 6859959
Historical Wards 2003-2015 6840014
Zip Codes 6859959
Community Areas 6842753
Census Tracts 6844900
Wards 6842865
Boundaries - ZIP Codes 6842800
Police Districts 6843830
Police Beats 6843853
dtype: int64
```

각 열을 세는데 48초가 걸렸다. Dask는 이러한 대량 데이터의 분석작업을 돕기 위한 작업 스케줄러(task scheduler)를 제공

- **dask.get** : 단일 쓰레드
- **dask.threaded.get** : 멀티쓰레드 풀
- **dask.multiprocessing.get**: 멀티프로세스 풀
- **dask.distributed.Client.get**: 여러대의 컴퓨터에서 분산처리

# Numba(눔바) 시작하기

- Python으로 직접 작성된 고성능 함수로 응용 프로그램의 속도를 높여 줌
- 몇가지 주석을 사용하면 배열 지향 및 수학 중심 python 코드가 언어 또는 python 해석기를 전환할 필요 없이 C, C++ 및 Fortran과 유사한 성능의 네이티브 머신 명령어로 적시에 컴파일 될 수 있음
- Numba는 가져오기 시간(import time), 런타임(runtime) 또는 정적으로 (포함된 pycc 도구를 사용하여) LLVM 컴파일러 인프라를 사용하여 최적화 된 기계코드를 생성
- Numba는 cpu 또는 GPU 하드웨어에서 실행되는 python 컴파일을 지원하며 python 과학 소프트웨어 스택과 통합되도록 설계

NationalNames.csv 파일을 이용하여 분석한다.

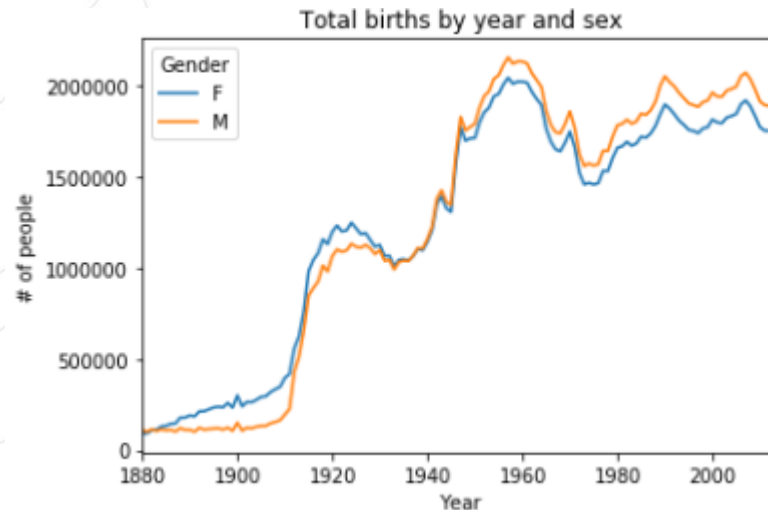
데이터분석 1: 각 연도별 성별의 총 출생횟수 분석

1. 연도('Year')와 성별('Gender')로 그룹하고, 출생횟수('Count')의 합을 구함

>> 실행결과

```
Year  Gender  Count
1880   F      90893
      M     110491
1881   F      91954
      M     100745
1882   F     107850
Name: Count, dtype: int64
```

2. 위에서 얻어진 결과로 아래의 선 그래프를 그려봄



## 데이터분석 2 : 각 연도별 남녀의 그룹별 / 각 이름의 출생횟수가 전체에서 차지하는 비중 계산

1. 연도('Year')와 성별('Gender') 의 기준열을 가지고 group하여 딕셔너리 형태로 키값 출력
2. 샘플로 1990년생 남자('M')에 해당하는 데이터를 추출

>> 실행결과

|         | Id      | Name        | Year | Gender | Count |
|---------|---------|-------------|------|--------|-------|
| 1084683 | 1084684 | Michael     | 1990 | M      | 65274 |
| 1084684 | 1084685 | Christopher | 1990 | M      | 52323 |
| 1084685 | 1084686 | Matthew     | 1990 | M      | 44794 |
| 1084686 | 1084687 | Joshua      | 1990 | M      | 43214 |
| 1084687 | 1084688 | Daniel      | 1990 | M      | 33809 |

3. 연도('Year')와 성별('Gender') 의 기준열을 가지고 group한 결과물 데이터프레임에 출생비율을 계산하여 새로운열로 추가해줄 사용자 정의함수를 정의하고, 이를 이용하여 비율 계산 후 새로운 'prop'열에 저장

출생비율('prop') = [Count 열 변수 / Count 열 변수.sum() 총합]

>> 실행결과

|   | Id | Name      | Year | Gender | Count | prop     |
|---|----|-----------|------|--------|-------|----------|
| 0 | 1  | Mary      | 1880 | F      | 7065  | 0.077643 |
| 1 | 2  | Anna      | 1880 | F      | 2604  | 0.028618 |
| 2 | 3  | Emma      | 1880 | F      | 2003  | 0.022013 |
| 3 | 4  | Elizabeth | 1880 | F      | 1939  | 0.021309 |
| 4 | 5  | Minnie    | 1880 | F      | 1746  | 0.019188 |

## 데이터분석 3: 각 연도별, 남녀의 그룹별 각 이름의 출생횟수 TOP 1000 추출

1. 데이터 분석 2의 결과 테이블로 연도('Year')와 성별('Gender')의 기준열을 가지고 group하여 딕셔너리 형태로 키값 출력
2. 1990년생 남자('M')의 'Count'가 높은 상위 1000개의 데이터를 추출

>> 실행결과

|         | Id      | Name        | Year | Gender | Count | prop     |
|---------|---------|-------------|------|--------|-------|----------|
| 1084683 | 1084684 | Michael     | 1990 | M      | 65274 | 0.031802 |
| 1084684 | 1084685 | Christopher | 1990 | M      | 52323 | 0.025492 |
| 1084685 | 1084686 | Matthew     | 1990 | M      | 44794 | 0.021824 |
| 1084686 | 1084687 | Joshua      | 1990 | M      | 43214 | 0.021054 |
| 1084687 | 1084688 | Daniel      | 1990 | M      | 33809 | 0.016472 |

3. 2번과 3번의 결과물에 착안해서 연도('Year')와 성별('Gender')의 기준열을 가지고 group한 결과물 데이터프레임에 출생횟수('Count')로 내림차순 정렬하여 1000번째 행까지만 저장하는 사용자 정의함수를 정의하고, 이를 이용하여 'Count' 열 기준 상위 1000개의 데이터를 추출

>> 실행결과

|      | Id | Name   | Year | Gender    | Count | prop            |
|------|----|--------|------|-----------|-------|-----------------|
| Year |    | Gender |      |           |       |                 |
| 1880 | F  | 0      | 1    | Mary      | 1880  | F 7065 0.077643 |
|      |    | 1      | 2    | Anna      | 1880  | F 2604 0.028618 |
|      |    | 2      | 3    | Emma      | 1880  | F 2003 0.022013 |
|      |    | 3      | 4    | Elizabeth | 1880  | F 1939 0.021309 |
|      |    | 4      | 5    | Minnie    | 1880  | F 1746 0.019188 |

## 데이터분석 4: 각 연도에 따른 성별의 전체 출생횟수 대비 top 1000이름들의 출생횟수 비중의 합 산출

1. 데이터분석 3의 결과 테이블에서 멀티 인덱스를 제거하기 위해 `DataFrameName.droplevel([0,1])`을 실행하여 다른 변수에 저장
2. 연도('Year')와 성별('Gender')의 기준열을 가지고 group하여 'prop'의 합 출력한 결과 확인 후 unstack하여 데이터프레임의 형태로 변환

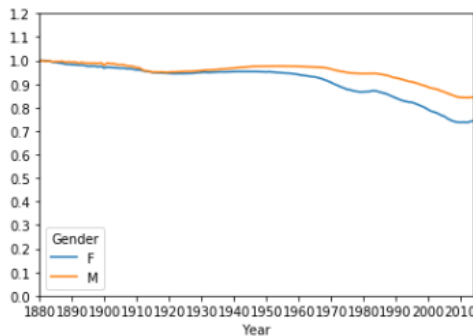
>> 결과 화면

| Gender | 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 |

3. 각 연도별 / 성별별 / 전체 출생횟수 대비, 상위 1000개의 출생횟수비중의 합을 선 그래프를 그림

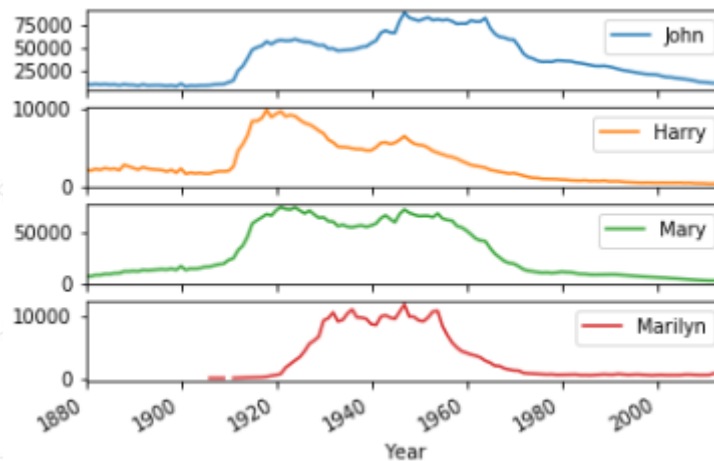
- ✓ x축은 1880년부터 2020년전까지 10의 간격으로, y축은 0부터 1.3전까지 0.1 간격으로 수정

>> 결과 화면



## 데이터분석 5 : 각 연도에 따른 특정 이름들의 출생횟수 변화 추이 분석하기

1. 각 연도별/성별별/탑1000개의 이름데이터 들이 나와있는 데이터분석 3의 결과 테이블을 이용하여 각 연도에 따른 (기준열 1), 특정이름들의 (기준열 2), 출생횟수(추출열), 변화(통계함수 sum으로 이름별 출생횟수의 합으로 변화를 살핍)
2. 1번의 결과인 7031가지 이름 중에 열 인덱싱으로 특별한 이름 4개('John', 'Harry', 'Mary', 'Marilyn')만 가져오기
3. 2번의 결과 데이터프레임으로 선그래프를 그림
4. 2번의 결과 데이터프레임으로 선그래프를 그릴때 옵션으로 `subplots=True`를 사용





## 데이터분석 6 : 남아의 이름 마지막글자에 따른 출생횟수가 연도에 따라 어떻게 변화하였는지 분석

저명한 연구자에 따르면 지난 100년동안 남아 이름의 마지막글자의 분포가 급격하게 변화하였다고 알려졌다. 그것을 조사해보자.

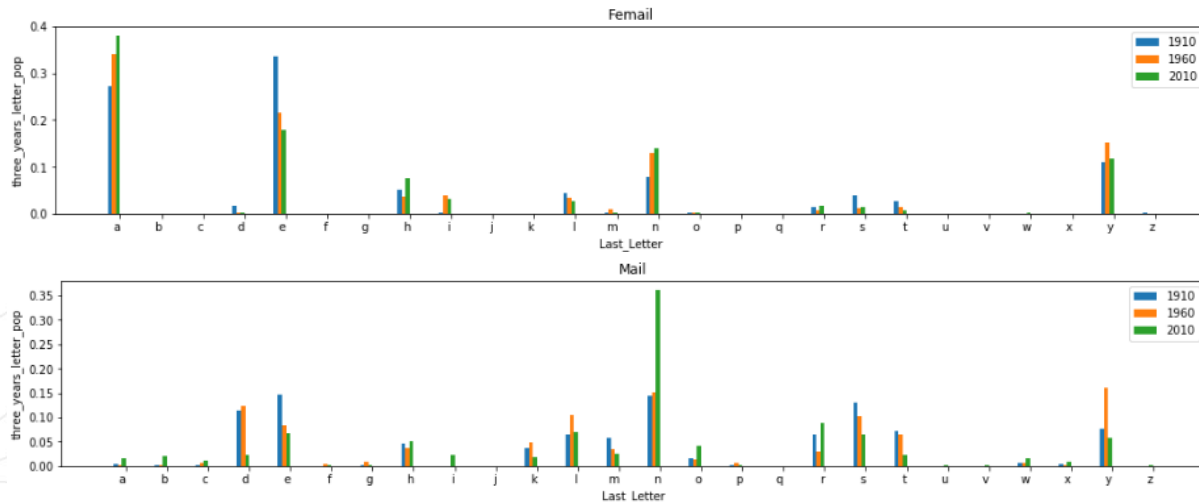
1. 처음 불러온 NationalNames.csv 파일이 저장되어 있는 데이터프레임을 이용하여 'Name'열의 마지막 글자를 추출하여 반환하는 함수를 생성하고 이를 이용하여 마지막 글자를 'Last\_Letter'열에 저장
2. 남아의 마지막 글자에 따른 출생횟수가 연도에 따라 어떻게 변화하는지 분석이므로 추출열 = 출생횟수('Count'), 기준열 1 = 마지막 글자열('Last\_Letter'), 기준열 2 = 남아(성별 'Gender') 로 group하여 데이터프레임 생성
3. 연도가 너무 많으므로 reindex()함수를 사용하여 특정 컬럼(1910, 1960, 2010)만 골라내어 다른 데이터프레임이름으로 저장

| Gender      | F                        |                          |                          | M       |                          |          |
|-------------|--------------------------|--------------------------|--------------------------|---------|--------------------------|----------|
|             | 1910                     | 1960                     | 2010                     | 1910    | 1960                     | 2010     |
| Last_Letter |                          |                          |                          |         |                          |          |
| a           | <a href="#">108397.0</a> | 691245.0                 | 675901.0                 | 977.0   | 5214.0                   | 28814.0  |
| b           | NaN                      | 694.0                    | 454.0                    | 411.0   | 3912.0                   | 39208.0  |
| c           | 5.0                      | 49.0                     | 953.0                    | 482.0   | 15466.0                  | 23307.0  |
| d           | 6751.0                   | 3728.0                   | 2635.0                   | 22113.0 | <a href="#">262143.0</a> | 44758.0  |
| e           | 133601.0                 | <a href="#">435048.0</a> | <a href="#">316288.0</a> | 28665.0 | <a href="#">178810.0</a> | 130073.0 |

4. 전체에 대한 비중을 구하는 식을 이용하여 각 성별/연도에 따른 마지막 글자들 a~z를 가진 아이의 출생횟수의 상대적 비중을 산출한 결과로 데이터프레임을 생성 후 저장

| Gender      | F                        |                          |                          | M                        |          |          |
|-------------|--------------------------|--------------------------|--------------------------|--------------------------|----------|----------|
| Year        | 1910                     | 1960                     | 2010                     | 1910                     | 1960     | 2010     |
| Last_Letter |                          |                          |                          |                          |          |          |
| a           | <a href="#">0.273384</a> | 0.341846                 | 0.381275                 | 0.005030                 | 0.002445 | 0.015056 |
| b           | NaN                      | 0.000343                 | 0.000256                 | 0.002116                 | 0.001834 | 0.020486 |
| c           | 0.000013                 | 0.000024                 | 0.000538                 | 0.002482                 | 0.007252 | 0.012178 |
| d           | 0.017026                 | 0.001844                 | 0.001486                 | <a href="#">0.113857</a> | 0.122915 | 0.023386 |
| e           | <a href="#">0.336950</a> | <a href="#">0.215147</a> | <a href="#">0.178418</a> | 0.147592                 | 0.083841 | 0.067964 |

5. 4번의 결과로 성별에 따른 선그래프를 그려봄
6. 4번의 결과로 성별에 따른 막대 그래프를 그려봄



## 데이터분석 7 : 특정 몇개의 글자에 대해 연도에 따른 남아 출생횟수

1. 데이터분석 6의 2번의 결과 데이터프레임을 이용하여 성별별 연도별로 합을 구해 비율을 계산하기 위해 'Gender'를 level 0의 행으로 바꾸고, 데이터분석 6의 4번과 같이 전체에 대한 비중을 구하여 데이터

| Gender      | F        |          |                          |          |                          |          |                          |                          |                          |                          | ... | M        |          |          |
|-------------|----------|----------|--------------------------|----------|--------------------------|----------|--------------------------|--------------------------|--------------------------|--------------------------|-----|----------|----------|----------|
| Year        | 1880     | 1881     | 1882                     | 1883     | 1884                     | 1885     | 1886                     | 1887                     | 1888                     | 1889                     | ... | 2005     | 2006     | 2007     |
| Last_Letter |          |          |                          |          |                          |          |                          |                          |                          |                          |     |          |          |          |
| a           | 0.345587 | 0.343443 | <a href="#">0.338767</a> | 0.341254 | <a href="#">0.338547</a> | 0.341272 | <a href="#">0.339710</a> | <a href="#">0.335261</a> | <a href="#">0.332766</a> | <a href="#">0.328717</a> | ... | 0.018486 | 0.017643 | 0.016757 |
| b           | NaN      | NaN      | NaN                      | NaN      | NaN                      | NaN      | NaN                      | NaN                      | NaN                      | NaN                      | ... | 0.021645 | 0.020778 | 0.020361 |
| c           | NaN      | NaN      | 0.000046                 | 0.000045 | NaN                      | NaN      | NaN                      | NaN                      | NaN                      | NaN                      | ... | 0.013085 | 0.012990 | 0.012978 |
| d           | 0.006693 | 0.006601 | 0.006806                 | 0.007211 | 0.007100                 | 0.006479 | 0.006967                 | 0.007035                 | 0.007267                 | 0.007703                 | ... | 0.025431 | 0.025080 | 0.024461 |
| e           | 0.366841 | 0.370620 | 0.374585                 | 0.373154 | 0.372719                 | 0.372898 | 0.372809                 | 0.372327                 | 0.373684                 | 0.373732                 | ... | 0.070801 | 0.069737 | 0.069452 |

2. 마지막 글자가 'd', 'n', 'y'이고, 성별이 'M'인 데이터만 추출하여 저장 후 전치행렬로 바꿈
3. 2번의 결과로 선그래프를 그림

