



# 데이터 시각화 이해와 실습

## Lecture 08. pandas 라이브러리를 활용한 프로젝트

동덕여자대학교  
데이터사이언스 전공  
권 범

# 목차

- ❖ 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리
- ❖ 02. 위키피디아 데이터 엑셀로 저장하기
- ❖ 03. pandas로 인구 구조 분석하기

## 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

02. 위키피디아 데이터 엑셀로 저장하기

03. pandas로 인구 구조 분석하기

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ pandas 라이브러리 (1/9)

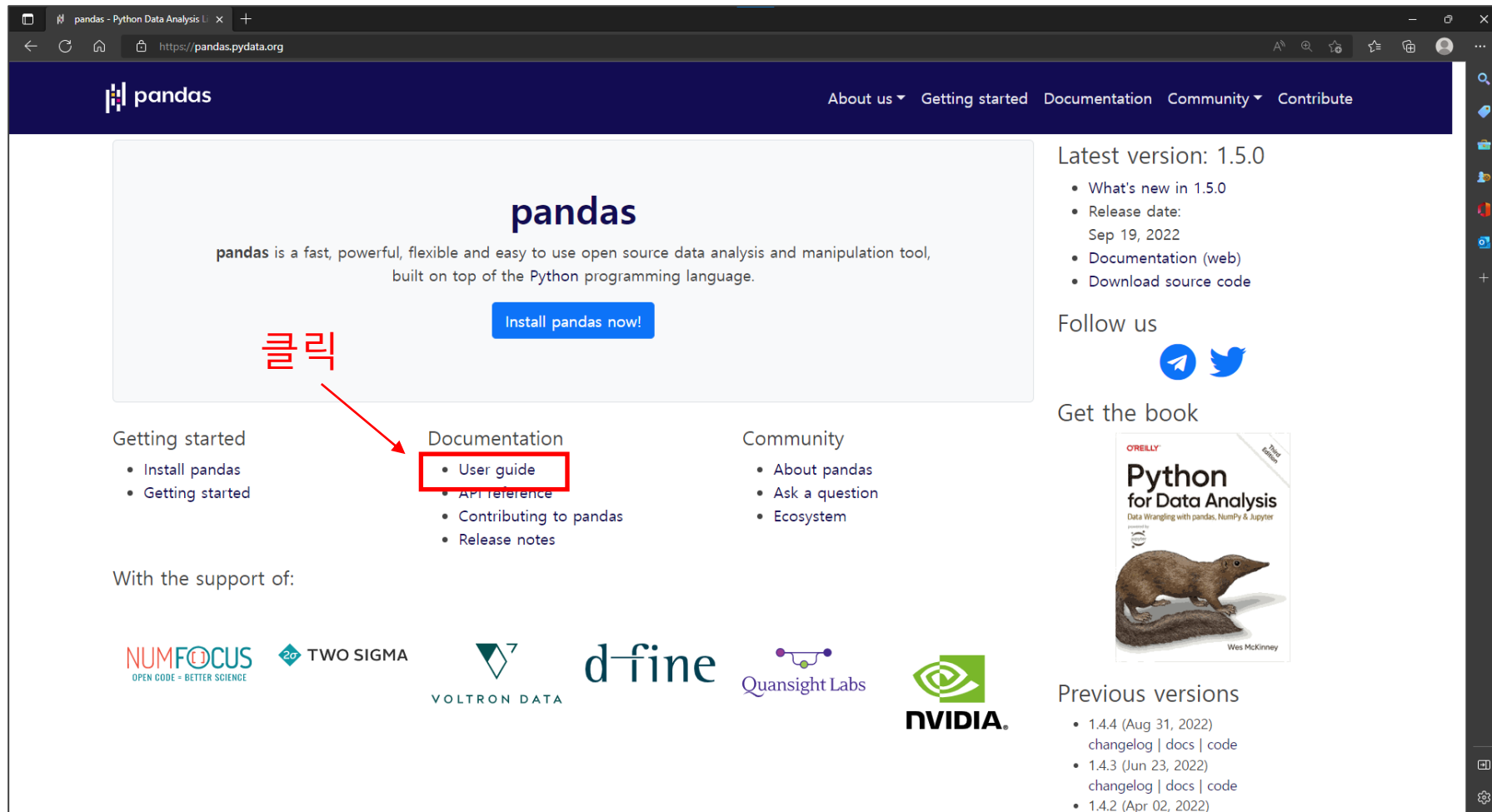
- pandas의 약자로 파이썬을 활용한 데이터 분석에서 많이 활용되고 있음
- numpy를 기반으로 만들어졌으며 데이터 분석을 위한 효율적인 데이터 구조를 제공하고 있음
- [URL] <https://pandas.pydata.org/>



# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ pandas 라이브러리 (2/9)

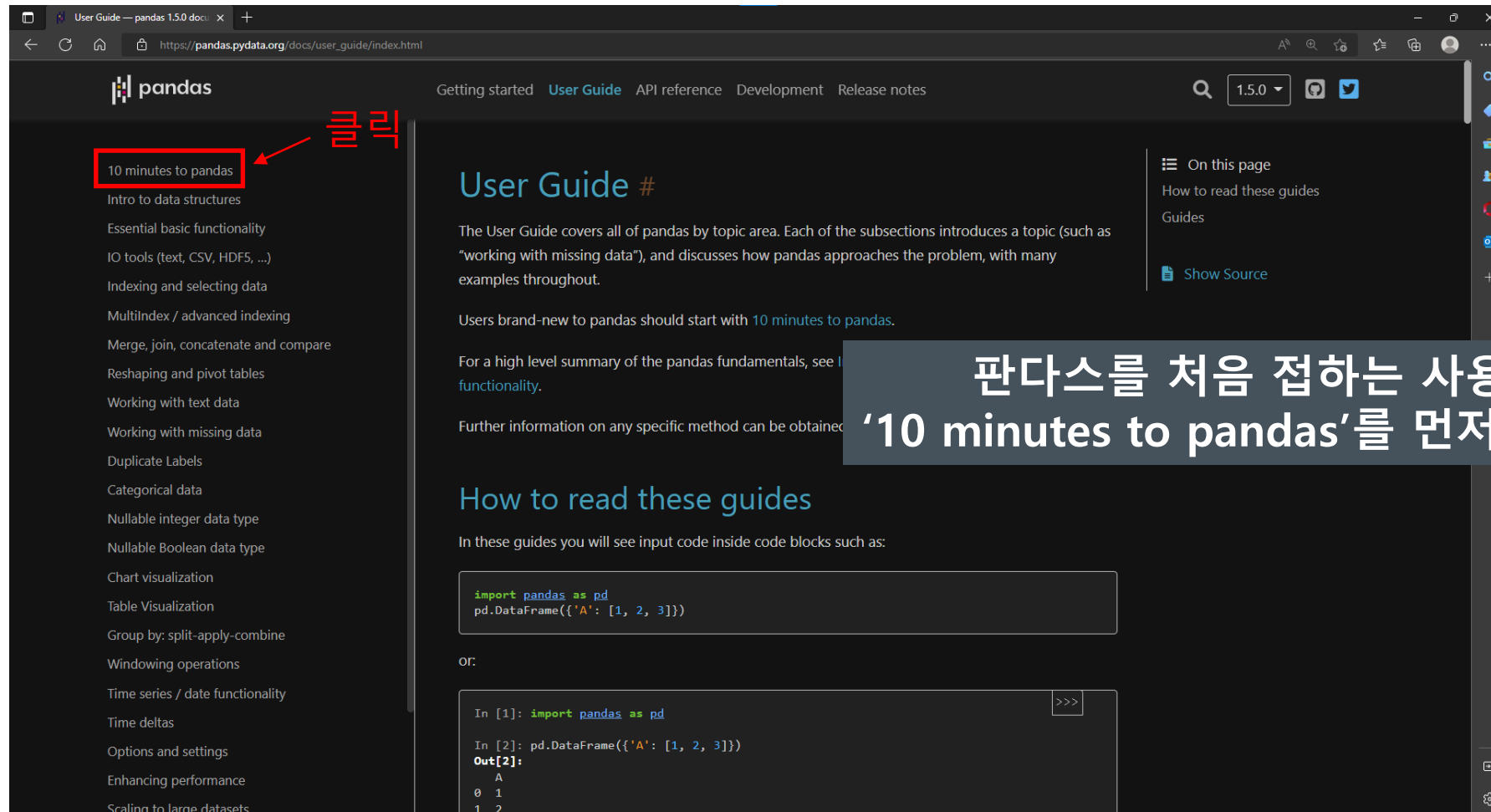
- 사용자 가이드(User guide)를 클릭



# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ pandas 라이브러리 (3/9)

- pandas 라이브러리 사용자를 위한 설명을 확인할 수 있음



# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ pandas 라이브러리 (4/9)

- pandas 라이브러리에 대한 간략한 소개 내용을 살펴볼 수 있음

The screenshot shows the pandas documentation page titled "10 minutes to pandas". The page is part of the pandas user guide, specifically the "10min.html" document. The left sidebar contains a table of contents with links to various topics, including "10 minutes to pandas", "Intro to data structures", "Essential basic functionality", "IO tools (text, CSV, HDF5, ...)", "Indexing and selecting data", "Multiindex / advanced indexing", "Merge, join, concatenate and compare", "Reshaping and pivot tables", "Working with text data", "Working with missing data", "Duplicate Labels", "Categorical data", "Nullable integer data type", "Nullable Boolean data type", "Chart visualization", "Table Visualization", "Group by: split-apply-combine", "Windowing operations", "Time series / date functionality", "Time deltas", "Options and settings", "Enhancing performance", and "Scaling to large datasets". The main content area is titled "10 minutes to pandas" and contains a short introduction to pandas, geared mainly for new users. It mentions that users can see more complex recipes in the Cookbook. The text states: "Customarily, we import as follows:" followed by a code block showing the import statements: 

```
In [1]: import numpy as np
In [2]: import pandas as pd
```

 Below this, the section "Object creation" is introduced, with a link to the "Intro to data structures" section. It then describes creating a `Series` by passing a list of values, letting pandas create a default integer index: 

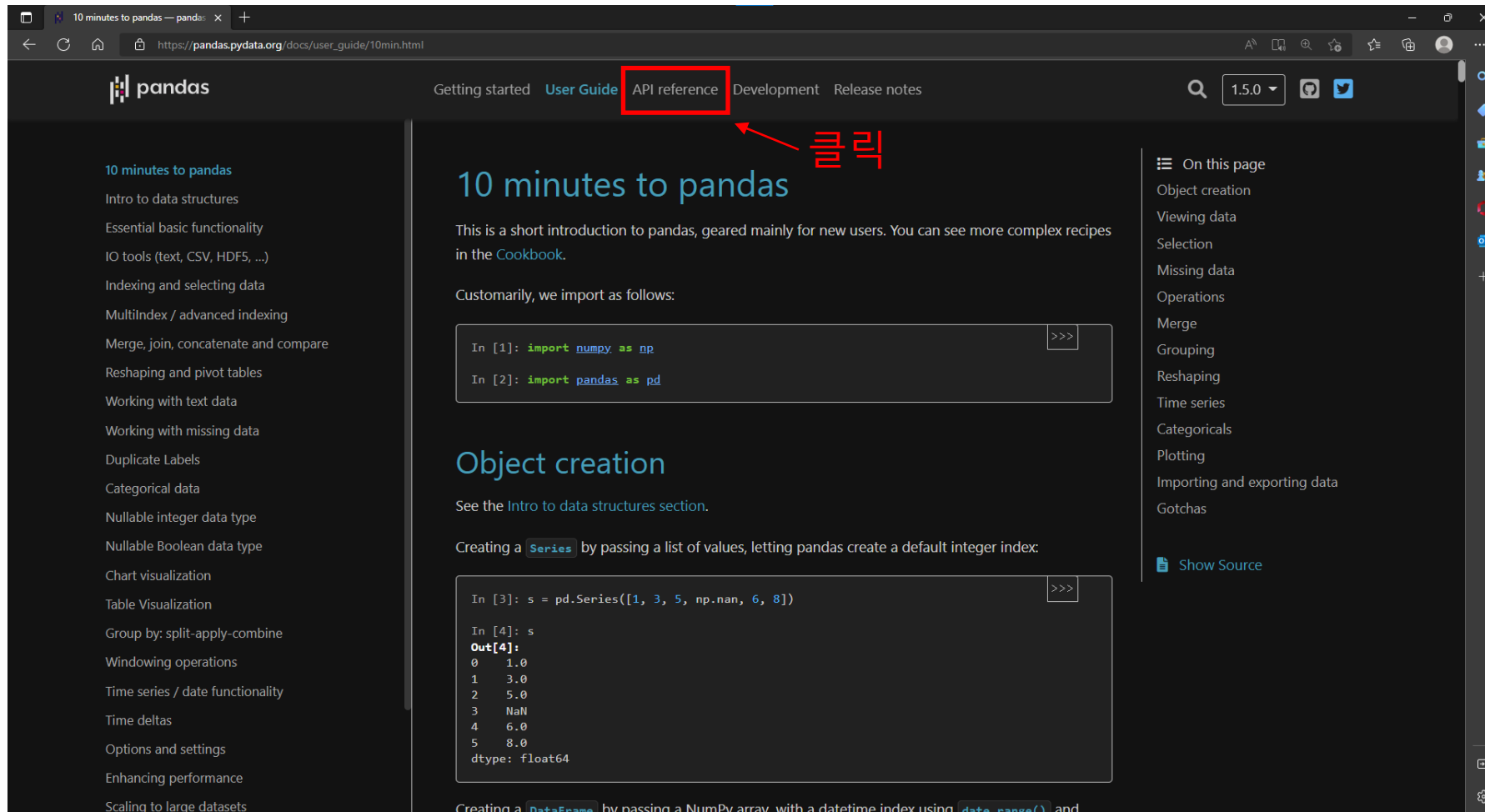
```
In [3]: s = pd.Series([1, 3, 5, np.nan, 6, 8])
Out[4]:
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

 The page also includes a "Show Source" link and a "On this page" sidebar with links to "Object creation", "Viewing data", "Selection", "Missing data", "Operations", "Merge", "Grouping", "Reshaping", "Time series", "Categoricals", "Plotting", "Importing and exporting data", and "Gotchas".

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ pandas 라이브러리 (5/9)

- 상단에 위치한 메뉴 중에서 [API reference]를 클릭

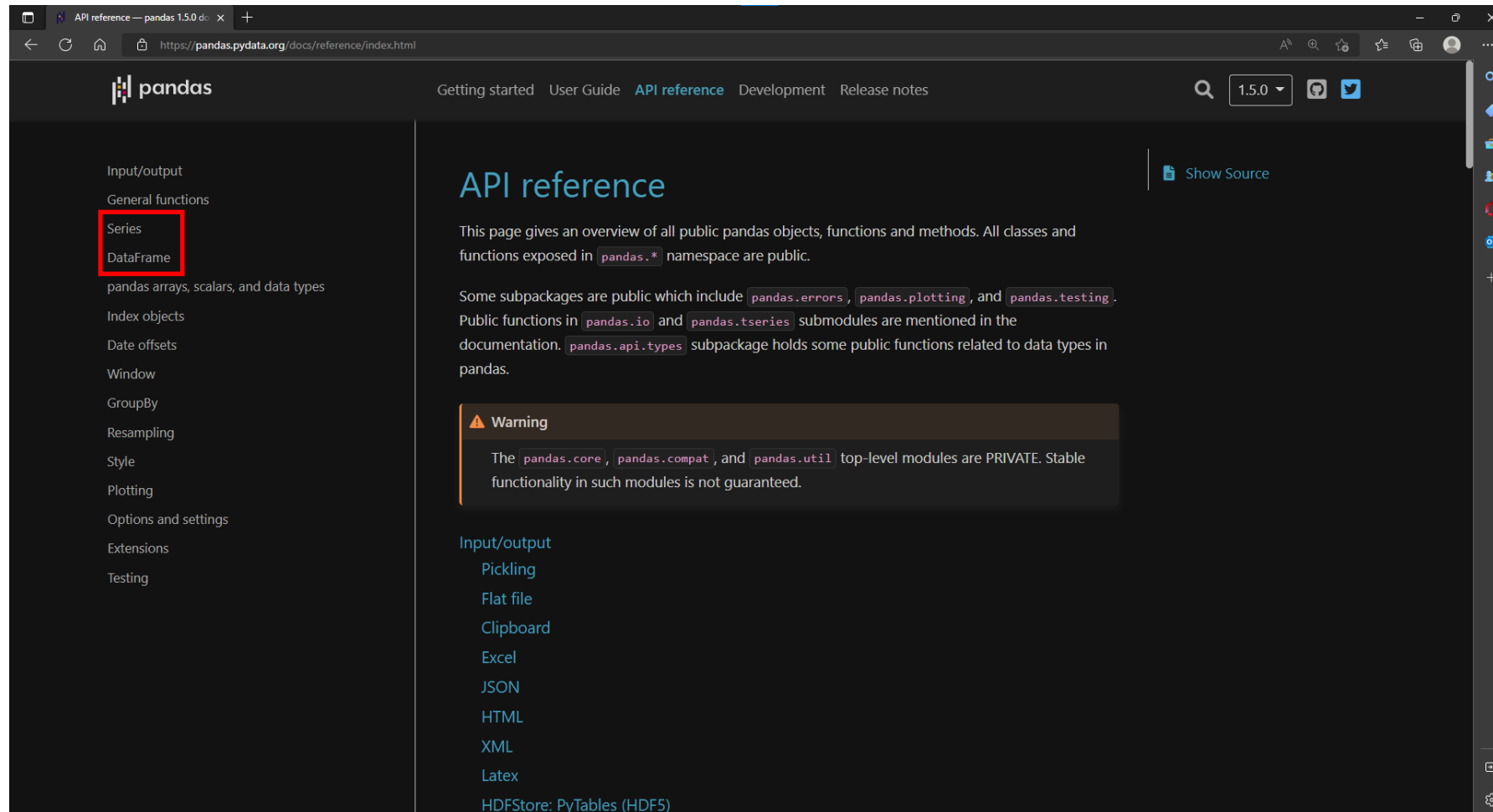




# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ pandas 라이브러리 (6/9)

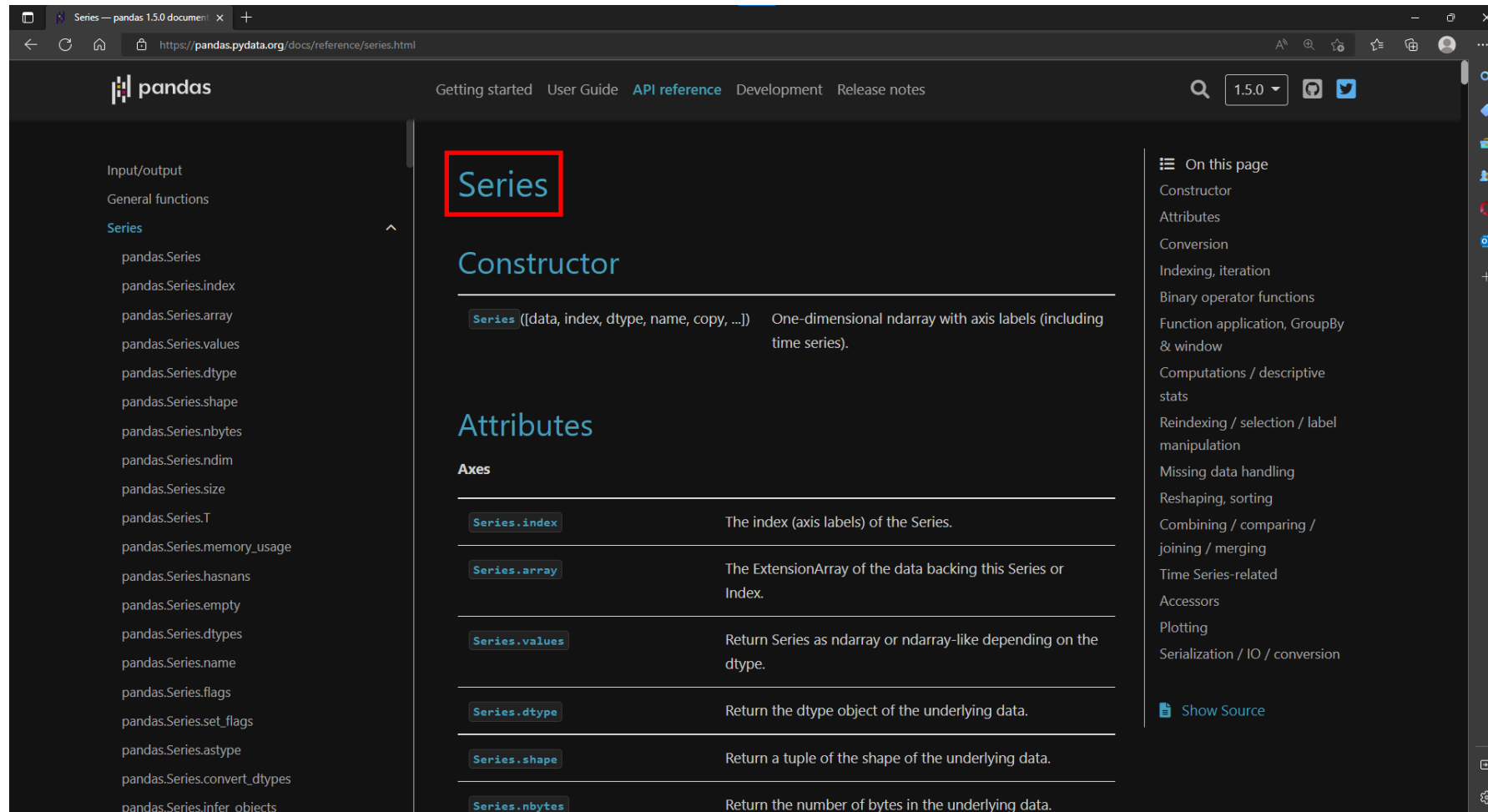
- 판다스에서 지원하는 데이터 구조인 Series(시리즈), DataFrame(데이터 프레임)을 확인할 수 있음



# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ pandas 라이브러리 (7/9)

- Series(시리즈)에 대한 내용을 확인할 수 있음



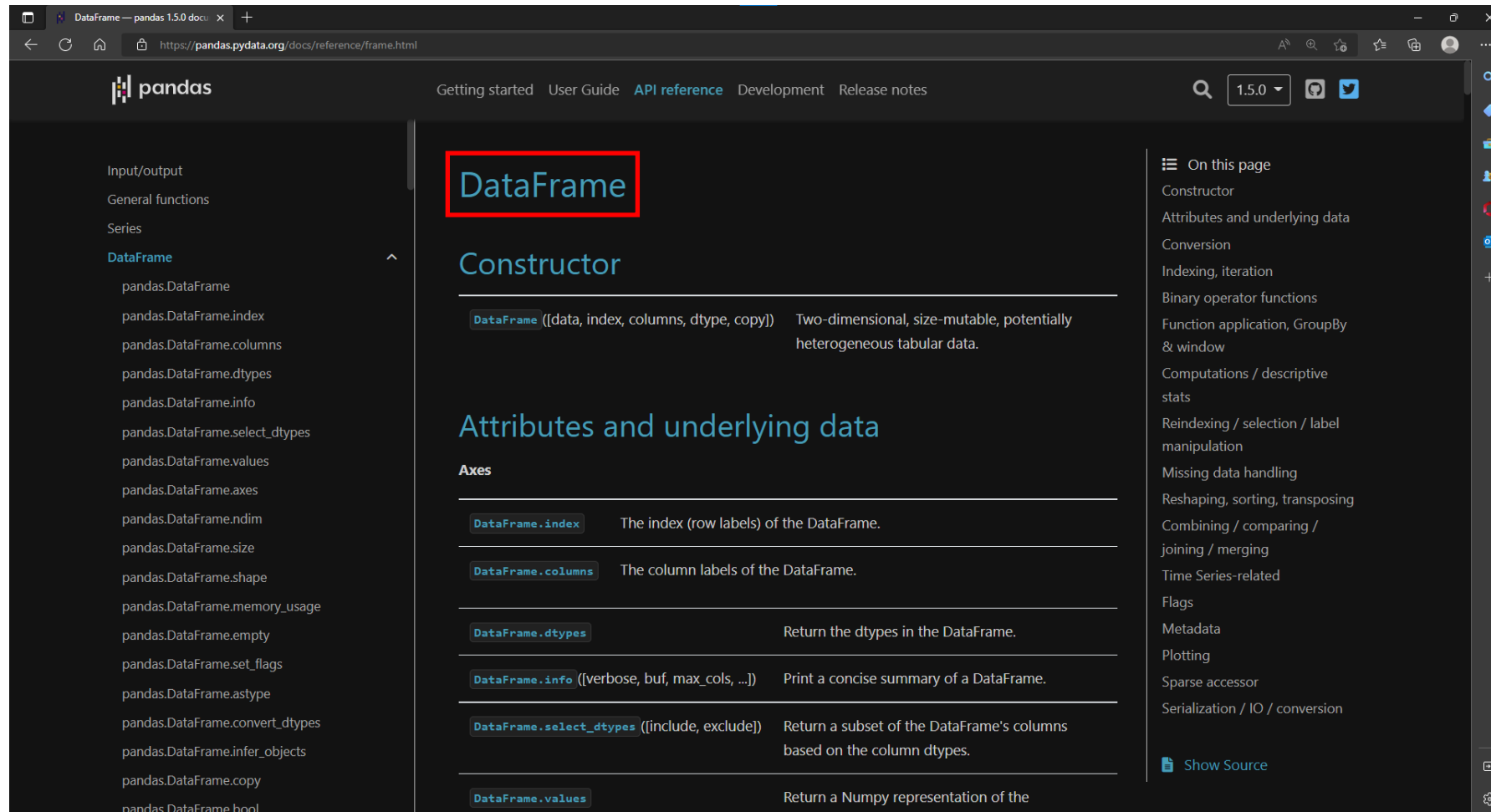
The screenshot shows the pandas 1.5.0 documentation page for the Series object. The page title is "Series" and it is highlighted with a red box. The page content includes sections for "Constructor", "Attributes", and "Axes". The "Constructor" section describes Series as a "One-dimensional ndarray with axis labels (including time series)". The "Attributes" section lists various attributes like .index, .array, .values, .dtype, .shape, and .nbytes. The "Axes" section lists the .index attribute. The left sidebar shows a navigation menu with "Series" selected. The right sidebar shows a table of contents for the page.

Series	Description
<code>Series</code>	One-dimensional ndarray with axis labels (including time series).
<code>Series.index</code>	The index (axis labels) of the Series.
<code>Series.array</code>	The ExtensionArray of the data backing this Series or Index.
<code>Series.values</code>	Return Series as ndarray or ndarray-like depending on the dtype.
<code>Series.dtype</code>	Return the dtype object of the underlying data.
<code>Series.shape</code>	Return a tuple of the shape of the underlying data.
<code>Series.nbytes</code>	Return the number of bytes in the underlying data.

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ pandas 라이브러리 (8/9)

- DataFrame(데이터 프레임)에 대한 내용을 확인할 수 있음



# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ pandas 라이브러리 (9/9)

- Series(시리즈)

- ◆ 1차원 배열 형태의 데이터 구조

--	--	--	--	--	--	--	--	--	--

- DataFrame(데이터 프레임)

- ◆ 2차원 배열 형태의 데이터 구조


# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ DataFrame (1/2)

- 행(Row)과 열(Column)로 구성
- 행을 구분해주는 인덱스, 열을 구분해주는 컬럼이 있음
- 인덱스는 별도로 지정을 하지 않으면 정수로 지정됨
- 인덱스 만들기 예제
  - period  
(명사) 기간, 시기  
(명사) 시대

```
1 import pandas as pd
2
3 index = pd.date_range("2000-1-1", periods=8)
4 print(index)
```

### 실행결과

```
DatetimeIndex(['2000-01-01', '2000-01-02', '2000-01-03', '2000-01-04',
               '2000-01-05', '2000-01-06', '2000-01-07', '2000-01-08'],
              dtype='datetime64[ns]', freq='D')
```

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ DataFrame (2/2)

```
1 import pandas as pd
2
3 # index = pd.date_range("2000-1-1", periods=8)
4 index = pd.date_range(start="2000-1-1", end="2000-1-8")
5 print(index)
```

### 실행결과

```
DatetimeIndex(['2000-01-01', '2000-01-02', '2000-01-03', '2000-01-04',
               '2000-01-05', '2000-01-06', '2000-01-07', '2000-01-08'],
              dtype='datetime64[ns]', freq='D')
```

**periods 매개변수에 생성할 기간의 수를 지정하는 것 대신에  
end 매개변수에 종료 날짜를 지정해, 생성하는 것도 가능!**

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ DataFrame 생성 예제

```
1 import pandas as pd
2 import numpy as np
3
4 index = pd.date_range("2000-1-1", periods=8)
5
6 df = pd.DataFrame(np.random.rand(8, 3), index=index, columns=['A', 'B', 'C'])
7 df
```

### 실행결과

	A	B	C
2000-01-01	0.962444	0.137672	0.321605
2000-01-02	0.349221	0.747925	0.435609
2000-01-03	0.281906	0.258660	0.291959
2000-01-04	0.698653	0.271535	0.656180
2000-01-05	0.159718	0.694678	0.346911
2000-01-06	0.528567	0.067998	0.642864
2000-01-07	0.734958	0.811348	0.209149
2000-01-08	0.142863	0.418683	0.381006

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ DataFrame의 특정 열에 접근하기 (1/3)

```
1 import pandas as pd
2 import numpy as np
3
4 index = pd.date_range("2000-1-1", periods=8)
5
6 df = pd.DataFrame(np.random.rand(8, 3), index=index, columns=['A', 'B', 'C'])
7 df['B']
```

### 실행결과

2000-01-01	0.137672
2000-01-02	0.747925
2000-01-03	0.258660
2000-01-04	0.271535
2000-01-05	0.694678
2000-01-06	0.067998
2000-01-07	0.811348
2000-01-08	0.418683

Freq: D, Name: B, dtype: float64

**'B'라는 이름(컬럼 이름)을 통해서  
DataFrame에 저장된 특정 열에 접근할 수 있음**



# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ DataFrame의 특정 열에 접근하기 (2/3)

```
1 import pandas as pd
2 import numpy as np
3
4 index = pd.date_range("2000-1-1", periods=8)
5
6 df = pd.DataFrame(np.random.rand(8, 3), index=index, columns=['A', 'B', 'C'])
7 df['B'][0]
```

### 실행결과

0.13767169912968746

FutureWarning: Series.\_\_getitem\_\_ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use 'ser.iloc[pos]'

df['B'][0]

Series에서 [ ] 안에 인덱스를 지정하면, 해당 원소 값을 갖고 올 수 있으나, (DataFrame 동작과 일치시키기 위해) 해당 기능은 추후 사라질 예정!

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ DataFrame의 특정 열에 접근하기 (3/3)

```
1 import pandas as pd
2 import numpy as np
3
4 index = pd.date_range("2000-1-1", periods=8)
5
6 df = pd.DataFrame(np.random.rand(8, 3), index=index, columns=['A', 'B', 'C'])
7 df['B'].iloc[0]
```

### 실행결과

0.13767169912968746

앞으로는 Series.iloc[pos]를 사용해  
원소 값을 갖고 오도록 하자!

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ 특정 열에 마스크(Mask) 생성하기

```
1 import pandas as pd
2 import numpy as np
3
4 index = pd.date_range("2000-1-1", periods=8)
5
6 df = pd.DataFrame(np.random.rand(8, 3), index=index, columns=['A', 'B', 'C'])
7 print(df['B'] > 0.4)
```

### 실행결과

2000-01-01	False
2000-01-02	True
2000-01-03	False
2000-01-04	False
2000-01-05	True
2000-01-06	False
2000-01-07	True
2000-01-08	True

Freq: D, Name: B, dtype: bool

마스크는 특정한 조건을 만족하는지에 따라,  
참(True) 또는 거짓(False)을 반환하여  
원하는 데이터를 골라내는데 유용하게 사용할 수 있음

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ 마스크가 적용된 결과를 데이터 프레임으로 저장하기

```
1 import pandas as pd
2 import numpy as np
3
4 index = pd.date_range("2000-1-1", periods=8)
5
6 df = pd.DataFrame(np.random.rand(8, 3), index=index, columns=['A', 'B', 'C'])
7 df
```

```
1 df2 = df[df['B'] > 0.4]
2 df2
```

### 실행결과

	A	B	C
2000-01-01	0.962444	0.137672	0.321605
2000-01-02	0.349221	0.747925	0.435609
2000-01-03	0.281906	0.258660	0.291959
2000-01-04	0.698653	0.271535	0.656180
2000-01-05	0.159718	0.694678	0.346911
2000-01-06	0.528567	0.067998	0.642864
2000-01-07	0.734958	0.811348	0.209149
2000-01-08	0.142863	0.418683	0.381006



	A	B	C
2000-01-02	0.349221	0.747925	0.435609
2000-01-05	0.159718	0.694678	0.346911
2000-01-07	0.734958	0.811348	0.209149
2000-01-08	0.142863	0.418683	0.381006

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ 행과 열 바꾸기

```
1 df2.T
```

### 실행결과

	A	B	C
2000-01-02	0.349221	0.747925	0.435609
2000-01-05	0.159718	0.694678	0.346911
2000-01-07	0.734958	0.811348	0.209149
2000-01-08	0.142863	0.418683	0.381006

	2000-01-02	2000-01-05	2000-01-07	2000-01-08
A	0.349221	0.159718	0.734958	0.142863
B	0.747925	0.694678	0.811348	0.418683
C	0.435609	0.346911	0.209149	0.381006

T는 행과 열을 바꾼다는 의미의 단어인 Transpose를 의미함

데이터 프레임 뒤에 .T만 붙여주면  
행과 열을 쉽게 바꿀 수 있음

- transpose  
(동사) 뒤바꾸다  
(동사) 바꾸다

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ A열의 값을 B열의 값으로 나눈 결과를 D열로 추가하기

```
1 import pandas as pd
2 import numpy as np
3
4 index = pd.date_range("2000-1-1", periods=8)
5
6 df = pd.DataFrame(np.random.rand(8, 3), index=index, columns=['A', 'B', 'C'])
7 df
```

```
1 df['D'] = df['A'] / df['B']
2 df
```

### 실행결과

	A	B	C		A	B	C	D	
2000-01-01	0.962444	0.137672	0.321605		2000-01-01	0.962444	0.137672	0.321605	6.990861
2000-01-02	0.349221	0.747925	0.435609		2000-01-02	0.349221	0.747925	0.435609	0.466919
2000-01-03	0.281906	0.258660	0.291959		2000-01-03	0.281906	0.258660	0.291959	1.089869
2000-01-04	0.698653	0.271535	0.656180		2000-01-04	0.698653	0.271535	0.656180	2.572981
2000-01-05	0.159718	0.694678	0.346911		2000-01-05	0.159718	0.694678	0.346911	0.229917
2000-01-06	0.528567	0.067998	0.642864		2000-01-06	0.528567	0.067998	0.642864	7.773241
2000-01-07	0.734958	0.811348	0.209149		2000-01-07	0.734958	0.811348	0.209149	0.905848
2000-01-08	0.142863	0.418683	0.381006		2000-01-08	0.142863	0.418683	0.381006	0.341219

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

❖ 같은 행에 있는 데이터의 합을 구하고, 결과를 E열에 추가하기

```
1 df['E'] = np.sum(df, axis=1)
2 df
```

axis 매개변수에 값을 지정하지 않거나,  
axis=0일 경우 E열에는 NaN 값으로 채워짐

## 실행결과

	A	B	C	D	E
2000-01-01	0.962444	0.137672	0.321605	6.990861	8.412582
2000-01-02	0.349221	0.747925	0.435609	0.466919	1.999674
2000-01-03	0.281906	0.258660	0.291959	1.089869	1.922393
2000-01-04	0.698653	0.271535	0.656180	2.572981	4.199350
2000-01-05	0.159718	0.694678	0.346911	0.229917	1.431225
2000-01-06	0.528567	0.067998	0.642864	7.773241	9.012670
2000-01-07	0.734958	0.811348	0.209149	0.905848	2.661303
2000-01-08	0.142863	0.418683	0.381006	0.341219	1.283771

- NA: Not Available
- NaN: Not a Number

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ 새 열 추가하기

```
1 df['F'] = np.ones(8)
2 df.head(3)      # head(n)은 처음 n개의 데이터만 보여줌
```

### 실행결과

	A	B	C	D	E	F
2000-01-01	0.962444	0.137672	0.321605	6.990861	8.412582	1.0
2000-01-02	0.349221	0.747925	0.435609	0.466919	1.999674	1.0
2000-01-03	0.281906	0.258660	0.291959	1.089869	1.922393	1.0



# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ 특정 열 제거하기 (1/2)

```
1 df1 = df.drop(['E', 'F'], axis="columns")           # axis=1이라고 적어도 됨
2 df1.head()           # 소괄호 안에 숫자를 지정하지 않으면, 처음 5개의 데이터만 보여줌
```

### 실행결과

	A	B	C	D	E	F		A	B	C	D	
2000-01-01	0.962444	0.137672	0.321605	6.990861	8.412582	1.0		2000-01-01	0.962444	0.137672	0.321605	6.990861
2000-01-02	0.349221	0.747925	0.435609	0.466919	1.999674	1.0		2000-01-02	0.349221	0.747925	0.435609	0.466919
2000-01-03	0.281906	0.258660	0.291959	1.089869	1.922393	1.0		2000-01-03	0.281906	0.258660	0.291959	1.089869
2000-01-04	0.698653	0.271535	0.656180	2.572981	4.199350	1.0		2000-01-04	0.698653	0.271535	0.656180	2.572981
2000-01-05	0.159718	0.694678	0.346911	0.229917	1.431225	1.0		2000-01-05	0.159718	0.694678	0.346911	0.229917

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ 특정 열 제거하기 (2/2)

```
1 del df1['D']           # del로는 한 번에 하나의 열만 삭제할 수 있음
2 df1.head()
```

### 실행결과

	A	B	C	D		A	B	C
2000-01-01	0.962444	0.137672	0.321605	6.990861	2000-01-01	0.962444	0.137672	0.321605
2000-01-02	0.349221	0.747925	0.435609	0.466919	2000-01-02	0.349221	0.747925	0.435609
2000-01-03	0.281906	0.258660	0.291959	1.089869	2000-01-03	0.281906	0.258660	0.291959
2000-01-04	0.698653	0.271535	0.656180	2.572981	2000-01-04	0.698653	0.271535	0.656180
2000-01-05	0.159718	0.694678	0.346911	0.229917	2000-01-05	0.159718	0.694678	0.346911

df.pop 메소드를 이용해서,  
열을 제거할 수도 있음

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ 전체 열 이름 변경하기

```
1 df1.columns = ['a', 'b', 'c']  
2 df1
```

### 실행결과

	A	B	C
2000-01-01	0.962444	0.137672	0.321605
2000-01-02	0.349221	0.747925	0.435609
2000-01-03	0.281906	0.258660	0.291959
2000-01-04	0.698653	0.271535	0.656180
2000-01-05	0.159718	0.694678	0.346911
2000-01-06	0.528567	0.067998	0.642864
2000-01-07	0.734958	0.811348	0.209149
2000-01-08	0.142863	0.418683	0.381006

	a	b	c
2000-01-01	0.962444	0.137672	0.321605
2000-01-02	0.349221	0.747925	0.435609
2000-01-03	0.281906	0.258660	0.291959
2000-01-04	0.698653	0.271535	0.656180
2000-01-05	0.159718	0.694678	0.346911
2000-01-06	0.528567	0.067998	0.642864
2000-01-07	0.734958	0.811348	0.209149
2000-01-08	0.142863	0.418683	0.381006

columns를 이용할 경우, 열 개수를 정확하게 일치시켜 주어야 함.  
따라서 열의 개수가 많을 경우에는 불편할 수 있음

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ 특정 열 이름 변경하기 (1/2)

```
1 df1.rename(columns={'a': "2000", 'b': "3000"})
2 df1.tail()           # 전체 데이터 중, 뒤에서 5개의 데이터를 보여줌
```

### 실행결과

	a	b	c
2000-01-04	0.698653	0.271535	0.656180
2000-01-05	0.159718	0.694678	0.346911
2000-01-06	0.528567	0.067998	0.642864
2000-01-07	0.734958	0.811348	0.209149
2000-01-08	0.142863	0.418683	0.381006

**inplace** 옵션을 지정하지 않는 경우,  
연산 결과를 별도 변수에 저장해야 함

inplace=True를 적은 경우와  
적지 않은 경우를 비교해 보자

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ 특정 열 이름 변경하기 (2/2)

```
1 df1.rename(columns={'a':'2000', 'b':'3000'}, inplace=True)
2 df1.tail()
```

### 실행결과

	a	b	c
2000-01-04	0.698653	0.271535	0.656180
2000-01-05	0.159718	0.694678	0.346911
2000-01-06	0.528567	0.067998	0.642864
2000-01-07	0.734958	0.811348	0.209149
2000-01-08	0.142863	0.418683	0.381006



	2000	3000	c
2000-01-04	0.698653	0.271535	0.656180
2000-01-05	0.159718	0.694678	0.346911
2000-01-06	0.528567	0.067998	0.642864
2000-01-07	0.734958	0.811348	0.209149
2000-01-08	0.142863	0.418683	0.381006

**inplace=True로 지정하면,  
연산 결과를 기존 데이터 프레임에 저장함**

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ 전체 데이터에 대해서 '2000'열의 값을 빼기

```
1 df1 = df1.sub(df1["2000"], axis="index")           # axis=0이라고 적어도 됨
2 df1.tail()
```

### 실행결과

	2000	3000	c
<b>2000-01-04</b>	0.698653	0.271535	0.656180
<b>2000-01-05</b>	0.159718	0.694678	0.346911
<b>2000-01-06</b>	0.528567	0.067998	0.642864
<b>2000-01-07</b>	0.734958	0.811348	0.209149
<b>2000-01-08</b>	0.142863	0.418683	0.381006



	2000	3000	c
<b>2000-01-04</b>	0.0	-0.427119	-0.042473
<b>2000-01-05</b>	0.0	0.534960	0.187193
<b>2000-01-06</b>	0.0	-0.460569	0.114297
<b>2000-01-07</b>	0.0	0.076390	-0.525809
<b>2000-01-08</b>	0.0	0.275821	0.238143

# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ 전체 데이터에 대해서 'c'열의 값으로 나누기

```
1 df1 = df1.div(df1['c'], axis="index")  
2 df1.tail()
```

### 실행결과

	2000	3000	c
<b>2000-01-04</b>	0.0	-0.427119	-0.042473
<b>2000-01-05</b>	0.0	0.534960	0.187193
<b>2000-01-06</b>	0.0	-0.460569	0.114297
<b>2000-01-07</b>	0.0	0.076390	-0.525809
<b>2000-01-08</b>	0.0	0.275821	0.238143



	2000	3000	c
<b>2000-01-04</b>	-0.0	10.056193	1.0
<b>2000-01-05</b>	0.0	2.857805	1.0
<b>2000-01-06</b>	0.0	-4.029583	1.0
<b>2000-01-07</b>	-0.0	-0.145281	1.0
<b>2000-01-08</b>	0.0	1.158214	1.0


# 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리

## ❖ 데이터 프레임을 CSV 파일로 저장하기

```
1 df1.to_csv("test.csv")
2 df1
```

### 실행결과

	2000	3000	c
2000-01-01	-0.0	1.287020	1.0
2000-01-02	0.0	4.615233	1.0
2000-01-03	0.0	-2.312256	1.0
2000-01-04	-0.0	10.056193	1.0
2000-01-05	0.0	2.857805	1.0
2000-01-06	0.0	-4.029583	1.0
2000-01-07	-0.0	-0.145281	1.0
2000-01-08	0.0	1.158214	1.0

test.csv ×				...
1 to 8 of 8 entries				Filter 
	2000	3000	c	
2000-01-01	-0.0	1.2870196819436	1.0	
2000-01-02	0.0	4.615233317613973	1.0	
2000-01-03	0.0	-2.312256208377905	1.0	
2000-01-04	-0.0	10.05619294559536	1.0	
2000-01-05	0.0	2.857805300632913	1.0	
2000-01-06	0.0	-4.029582911786968	1.0	
2000-01-07	-0.0	-0.14528097596325892	1.0	
2000-01-08	0.0	1.158214163062987	1.0	
Show 10 per page				



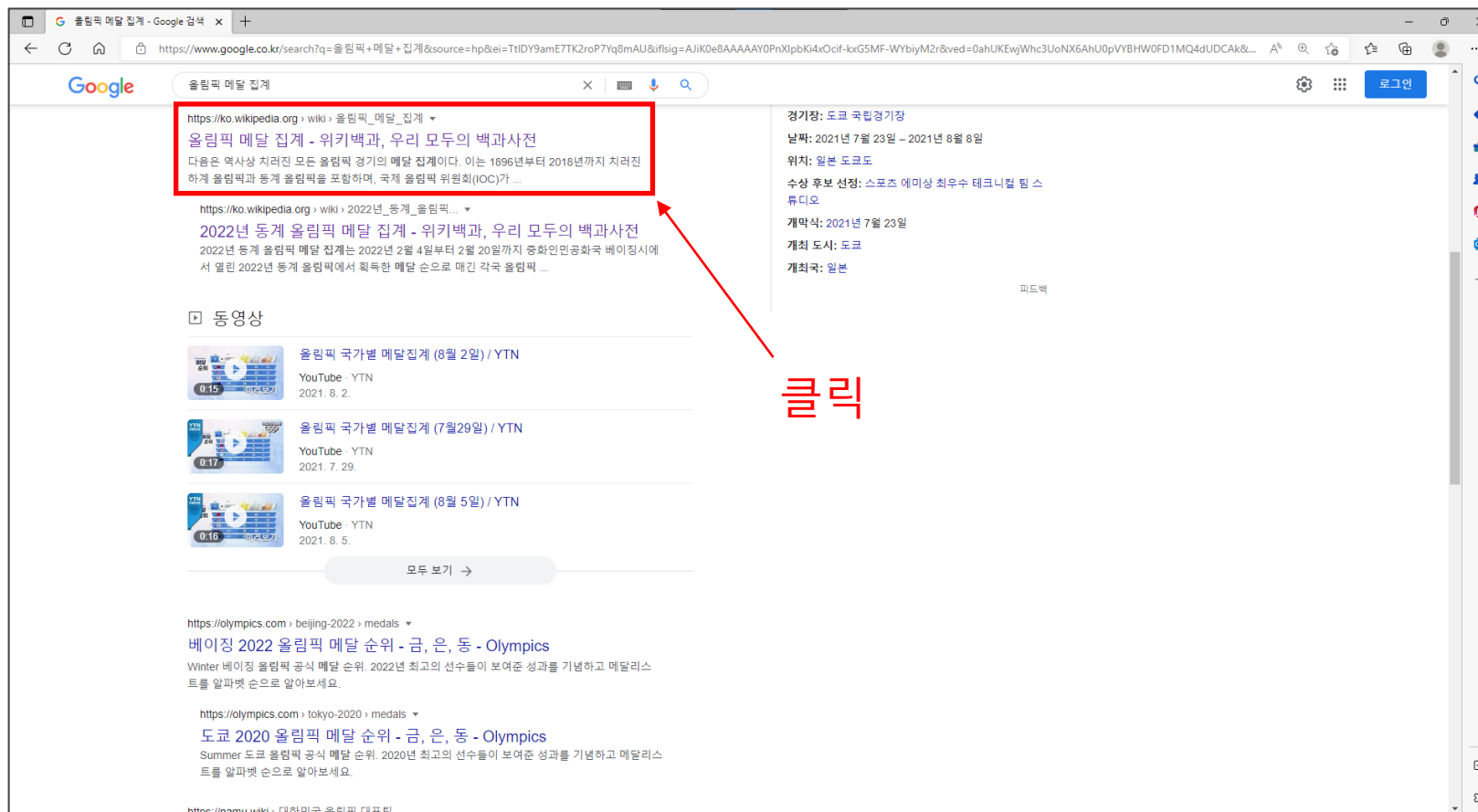
## 02. 위키피디아 데이터 엑셀로 저장하기

- 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리
- 03. pandas로 인구 구조 분석하기

## 02. 위키피디아 데이터 엑셀로 저장하기

### ❖ 올림픽 메달 기록 데이터 (1/2)

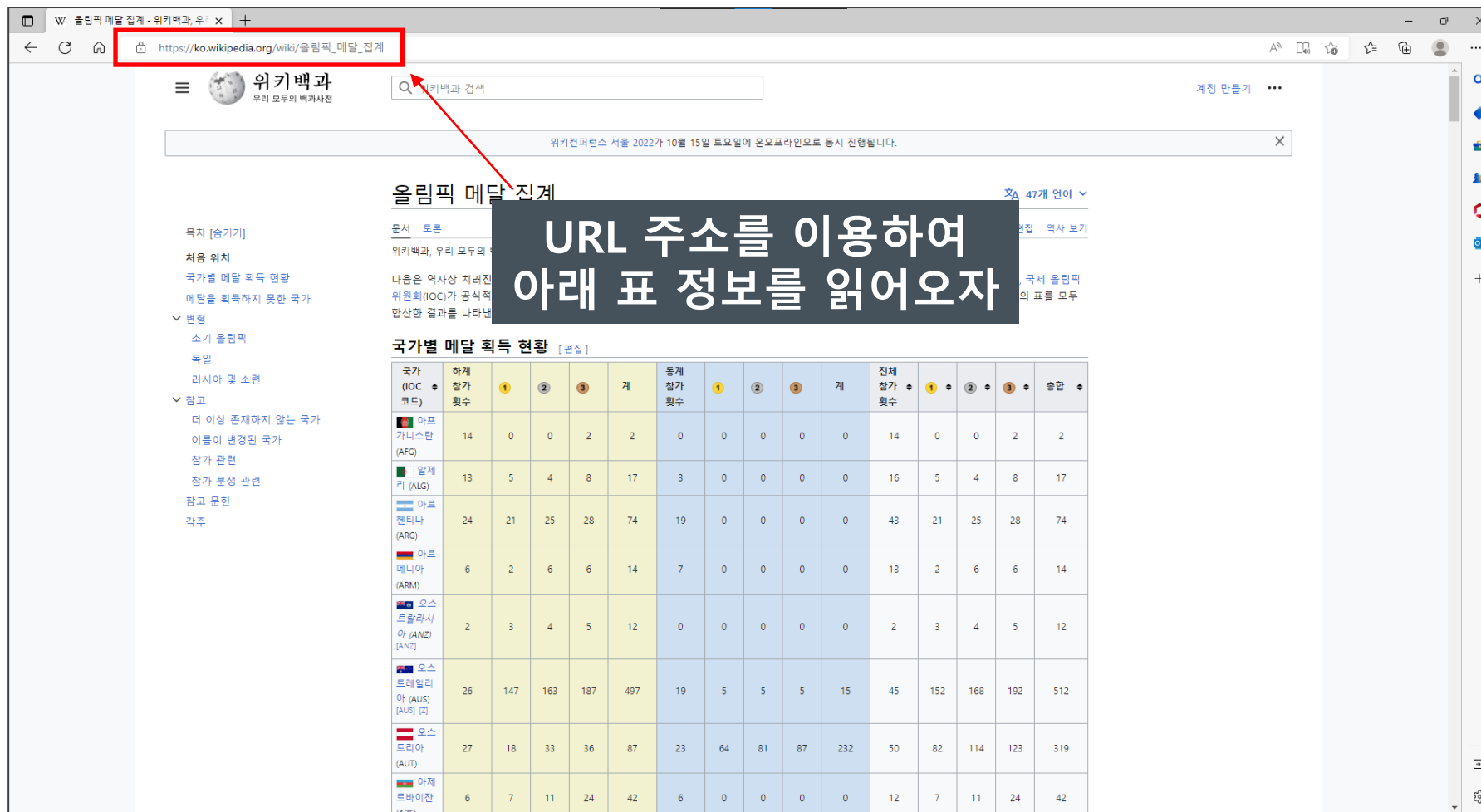
- pandas 라이브러리 실습을 위해 올림픽 메달 기록 데이터를 활용해 보자
- Google에서 "올림픽 메달 집계"라고 검색 → "올림픽 메달 집계 - 위키백과, 우리 모두의 백과사전" 검색 결과 클릭



## 02. 위키피디아 데이터 엑셀로 저장하기

### ❖ 올림픽 메달 기록 데이터 (2/2)

- 위키백과(Wikipedia)에서 제공하는 하계 및 동계 올림픽 메달 획득 결과 표



URL 주소를 이용하여  
아래 표 정보를 읽어오자

국가 (IOC 코드)	하계 참가 횟수	1	2	3	계	동계 참가 횟수	1	2	3	계	전제 참가 횟수	1	2	3	총합
아프가니스탄 (AFG)	14	0	0	2	2	0	0	0	0	0	14	0	0	2	2
알제리 (ALG)	13	5	4	8	17	3	0	0	0	0	16	5	4	8	17
아르헨티나 (ARG)	24	21	25	28	74	19	0	0	0	0	43	21	25	28	74
아르메니아 (ARM)	6	2	6	6	14	7	0	0	0	0	13	2	6	6	14
오스트레일리아 (AUS) [ANZ] [ANZ]	2	3	4	5	12	0	0	0	0	0	2	3	4	5	12
오스트리아 (AUT)	26	147	163	187	497	19	5	5	5	15	45	152	168	192	512
아제르바이잔 (AZE)	27	18	33	36	87	23	64	81	87	232	50	82	114	123	319
아제르바이잔 (AZE)	6	7	11	24	42	6	0	0	0	0	12	7	11	24	42

## 02. 위키피디아 데이터 엑셀로 저장하기

### ❖ ① pandas 라이브러리를 이용하여 URL로부터 데이터 읽어 오기

```
1 import pandas as pd
2
3 df = pd.read_html("https://ko.wikipedia.org/wiki/올림픽_메달_집계")
4 print(df)
```

#### 실행결과

```
[ 국가 (IOC 코드) 하계 참가 횟수 Unnamed: 2 Unnamed: 3 Unnamed: 4 계 \
0 아프가니스탄 (AFG) 14 0 0 2 2
1 알제리 (ALG) 13 5 4 8 17
2 아르헨티나 (ARG) 24 21 25 28 74
3 아르메니아 (ARM) 6 2 6 6 14
4 오스트랄라시아 (ANZ) [ANZ] 2 3 4 5 12
... ..
148 독립 (IOA) [IOA] 3 1 0 1 2
149 독립 참가 (IOP) [IOP] 1 0 1 2 3
150 러시아 출신 올림픽 선수 (OAR) 0 0 0 0 0
151 혼성 (ZZX) [ZZX] 3 8 5 4 17
152 총합 28 5116 5082 5490 15688
... (중략) ...
```

URL 뒷부분을 한글로 변경하지 말고,  
복사한 그대로 값을 붙여넣기 해야 오류 발생하지 않음!

대괄호 [ ]가 있음! 즉, 자료형이 리스트라는 뜻!  
df의 구성을 살펴 보자!

## 02. 위키피디아 데이터 엑셀로 저장하기

### ❖ ② 데이터 프레임 df[0] 살펴보기

```
1 import pandas as pd
2
3 df = pd.read_html("https://ko.wikipedia.org/wiki/올림픽_메달_집계")
4 df[0]
```

#### 실행결과

df[0]에 우리가 원하는 데이터가 담겨있음!

	국가 (IOC 코드)	하계 참가 횟수	Unnamed: 2	Unnamed: 3	Unnamed: 4	계	동계 참가 횟수	Unnamed: 7	Unnamed: 8	Unnamed: 9	계.1	전체 참가 횟수	Unnamed: 12	Unnamed: 13	Unnamed: 14	총합
0	아프가니스탄 (AFG)	14	0	0	2	2	0	0	0	0	0	14	0	0	2	2
1	알제리 (ALG)	13	5	4	8	17	3	0	0	0	0	16	5	4	8	17
2	아르헨티나 (ARG)	24	21	25	28	74	19	0	0	0	0	43	21	25	28	74
3	아르메니아 (ARM)	6	2	6	6	14	7	0	0	0	0	13	2	6	6	14
4	오스트랄라시아 (ANZ) [ANZ]	2	3	4	5	12	0	0	0	0	0	2	3	4	5	12
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
148	독립 (IOA) [IOA]	3	1	0	1	2	0	0	0	0	0	3	1	0	1	2
149	독립 참가 (IOP) [IOP]	1	0	1	2	3	0	0	0	0	0	1	0	1	2	3
150	러시아 출신 올림픽 선수 (OAR)	0	0	0	0	0	1	2	6	9	17	1	2	6	9	17
151	혼성 (ZZX) [ZZX]	3	8	5	4	17	0	0	0	0	0	3	8	5	4	17
152	총합	28	5116	5082	5490	15688	23	1062	1058	1050	3170	51	6178	6140	6540	18858

153 rows × 16 columns

# 02. 위키피디아 데이터 엑셀로 저장하기

## ❖ ③ 인덱스(Index)와 열 이름(Column Name) 확인

	국가 (IOC 코드)	하계 참가 횟수	Unnamed: 2	Unnamed: 3	Unnamed: 4	계	동계 참가 횟수	Unnamed: 7	Unnamed: 8	Unnamed: 9	계.1	전체 참가 횟수	Unnamed: 12	Unnamed: 13	Unnamed: 14	총합
0	아프가니스탄 (AFG)	14	0	0	2	2	0	0	0	0	0	14	0	0	2	2
1	알제리 (ALG)	13	5	4	17	3	0	0	0	0	0	16	5	4	8	17
2	아르헨티나 (ARG)	24	21	25	28	74	19	0	0	0	0	43	21	25	28	74
3	아르메니아 (ARM)	6	2	6	6	14	7	0	0	0	0	13	2	6	6	14
4	오스트랄라시아 (ANZ) [ANZ]	2	3	4	5	12	0	0	0	0	0	2	3	4	5	12
...			...	...	...	...	...	...	...	...	...	...	...	...	...	...
148	독립 (IOA) [IOA]	3	1	0	1	2	0	0	0	0	0	3	1	0	1	2
149	독립 참가 (IOP) [IOP]	1	0	1	2	3	0	0	0	0	0	1	0	1	2	3
150	러시아 출신 올림픽 선수 (OAR)	0	0	0	0	0	1	2	6	9	17	1	2	6	9	17
151	혼성 (ZZX) [ZZX]	3	8	5	4	17	0	0	0	0	0	3	8	5	4	17
152	총합	28	5116	5082	5490	15688	23	1062	1058	1050	3170	51	6178	6140	6540	18858

153 rows × 16 columns

# 02. 위키피디아 데이터 엑셀로 저장하기

## ❖ ④ 인덱스를 '국가 (IOC 코드)'로 변경하기

```
1 df2 = df[0].set_index("국가 (IOC 코드)")
2 df2
```

실행결과

	하계 참가 횟수	Unnamed: 2	Unnamed: 3	Unnamed: 4	계	동계 참가 횟수	Unnamed: 7	Unnamed: 8	Unnamed: 9	계.1	전체 참가 횟수	Unnamed: 12	Unnamed: 13	Unnamed: 14	총합
국가 (IOC 코드)															
아프가니스탄 (AFG)	14	0	0	2	2	0	0	0	0	0	14	0	0	2	2
알제리 (ALG)	13	5	4	8	17	3	0	0	0	0	16	5	4	8	17
아르헨티나 (ARG)	24	21	25	28	74	10	0	0	0	0	43	21	25	28	74
아르메니아 (ARM)	6										13	2	6	6	14
오스트랄라시아 (ANZ) [ANZ]	2	3	4	5	12	0	0	0	0	0	2	3	4	5	12
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
독립 (IOA) [IOA]	3	1	0	1	2	0	0	0	0	0	3	1	0	1	2
독립 참가 (IOP) [IOP]	1	0	1	2	3	0	0	0	0	0	1	0	1	2	3
러시아 출신 올림픽 선수 (OAR)	0	0	0	0	0	1	2	6	9	17	1	2	6	9	17
혼성 (ZZX) [ZZX]	3	8	5	4	17	0	0	0	0	0	3	8	5	4	17
총합	28	5116	5082	5490	15688	23	1062	1058	1050	3170	51	6178	6140	6540	18858

153 rows × 15 columns

## 02. 위키피디아 데이터 엑셀로 저장하기

### ❖ ⑤ 하계 정보만 추출하기

```
1 summer = df2.iloc[:, :5]
2 summer
```

iloc은 integer location의 약어로, 데이터 프레임의 행이나 열의 순서를 나타내는 정수로 특정 값을 추출함

#### 실행결과

	하계 참가 횟수	Unnamed: 2	Unnamed: 3	Unnamed: 4	계
국가 (IOC 코드)					
아프가니스탄 (AFG)	14	0	0	2	2
알제리 (ALG)	13	5	4	8	17
아르헨티나 (ARG)	24	21	25	28	74
아르메니아 (ARM)	6	2	6	6	14
오스트랄라시아 (ANZ) [ANZ]	2	3	4	5	12
...	...	...	...	...	...
독립 (IOA) [IOA]	3	1	0	1	2
독립 참가 (IOP) [IOP]	1	0	1	2	3
러시아 출신 올림픽 선수 (OAR)	0	0	0	0	0
혼성 (ZZX) [ZZX]	3	8	5	4	17
총합	28	5116	5082	5490	15688

153 rows × 5 columns

**iloc를 활용하여,  
하계 정보만 추출함!**



## 02. 위키피디아 데이터 엑셀로 저장하기

### ❖ ⑥ 컬럼 이름 설정하기

```
1 summer.columns = ["경기수", "금", "은", "동", "합계"]
2 summer
```

#### 실행결과

	경기수	금	은	동	합계
국가 (IOC 코드)					
아프가니스탄 (AFG)	14	0	0	2	2
알제리 (ALG)	13	5	4	8	17
아르헨티나 (ARG)	24	21	25	28	74
아르메니아 (ARM)	6	2	6	6	14
오스트랄라시아 (ANZ) [ANZ]	2	3	4	5	12
...	...	...	...	...	...
독립 (IOA) [IOA]	3	1	0	1	2
독립 참가 (IOP) [IOP]	1	0	1	2	3
러시아 출신 올림픽 선수 (OAR)	0	0	0	0	0
혼성 (ZZX) [ZZX]	3	8	5	4	17
총합	28	5116	5082	5490	15688

153 rows × 5 columns

데이터 프레임의 columns에  
컬럼 이름을 설정함

## 02. 위키피디아 데이터 엑셀로 저장하기

### ❖ ⑦ 내림차순으로 정렬하기

```
1 summer = summer.sort_values("금", ascending=False)      # 내림차순으로 정렬
2 summer
```

#### 실행결과

	경기수	금	은	동	합계
국가 (IOC 코드)					
총합	28	5116	5082	5490	15688
미국 (USA) [P] [Q] [R] [Z] [F]	27	1022	795	706	2523
소련 (URS) [URS]	9	395	319	296	1010
영국 (GBR) [GBR] [Z]	28	263	295	293	851
중화인민공화국 (CHN) [CHN]	10	224	167	155	546
...	...	...	...	...	...
리히텐슈타인 (LIE)	17	0	0	0	0
쿠웨이트 (KUW)	12	0	0	2	2
북마케도니아 (MKD)	6	0	0	1	1
말레이시아 (MAS) [MAS]	13	0	7	4	11
아프가니스탄 (AFG)	14	0	0	2	2

153 rows × 5 columns

**sort\_values( )** 함수를 이용하면  
원하는 열을 기준으로 데이터 순서를 정렬할 수 있음

## 02. 위키피디아 데이터 엑셀로 저장하기

### ❖ ⑧ 엑셀 파일로 저장하기

1 `summer.to_excel("하계올림픽메달.xlsx")`

`to_excel()` 함수를 이용하여 작업했던 'summer' 데이터 프레임을 엑셀 파일로 저장함

#### 실행결과



하계올림픽메달.xlsx

하계올림픽메달 - Excel												
국가 (IOC 코드)												
A	B	C	D	E	F	G	H	I	J	K	L	M
1	국가 (IOC 코드)	경수기	금	은	동	합계						
2	총합	28	5116	5082	5490	15688						
3	미국 (USA) [P] [Q] [R] [Z] [F]	27	1022	795	706	2523						
4	소련 (URS) [URS]	9	395	319	296	1010						
5	영국 (GBR) [GBR] [Z]	28	263	295	293	851						
6	중화인민공화국 (CHN) [CHN]	10	224	167	155	546						
7	프랑스 (FRA) [O] [P] [Z]	28	212	241	263	716						
8	이탈리아 (ITA) [M] [S]	27	206	178	193	577						
9	독일 (GER) [GER] [Z]	16	191	194	230	615						
10	헝가리 (HUN)	26	175	147	169	491						
11	동독 (GDR) [GDR]	5	153	129	127	409						
12	러시아 (RUS) [RUS]	6	148	125	153	426						
13	오스트레일리아 (AUS) [AUS] [Z]	26	147	163	187	497						
14	스웨덴 (SWE) [Z]	27	145	170	179	494						
15	일본 (JPN)	22	142	136	161	439						
16	핀란드 (FIN)	25	101	85	117	303						
17	대한민국 (KOR)	17	90	87	90	267						
18	루마니아 (ROU)	21	89	95	122	306						
19	네덜란드 (NED) [Z]	26	85	92	108	285						
20	쿠바 (CUB) [Z]	20	78	68	80	226						
21	폴란드 (POL)	21	68	83	133	284						
22	캐나다 (CAN)	26	64	102	136	302						
23	노르웨이 (NOR) [Q]	25	56	49	47	152						
24	서독 (FRG) [FRG]	5	56	67	81	204						
25	불가리아 (BUL) [H]	20	51	87	80	218						
26	스위스 (SUI)	28	50	75	67	192						

## 03. pandas로 인구 구조 분석하기

- 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리
- 02. 위키피디아 데이터 엑셀로 저장하기

## 03. pandas로 인구 구조 분석하기

### ❖ 알고리즘(Algorithm) 설계하기

- Step 1) 데이터를 읽음
  - ① 전체 데이터를 총 인구수로 나누어 비율로 변환함
  - ② 총 인구수와 연령 구간 인구수를 삭제함
- Step 2) 궁금한 지역(A)의 이름을 입력 받음
- Step 3) 궁금한 지역의 인구 구조를 저장함
- Step 4) 궁금한 지역의 인구 구조와 가장 비슷한 인구 구조를 가진 지역을 찾음
  - ① 전국의 모든 지역 중 한 곳(B)를 선택함
  - ② 궁금한 지역의 이름을 입력 받음
  - ③ ②를 100세 이상 인구수에 해당하는 값까지 반복한 후 차이의 제곱을 모두 더함
  - ④ 전국의 모든 지역에 대해 반복하며, 그 차이가 가장 작은 지역을 찾음
- Step 5) 가장 비슷한 곳의 인구 구조와 궁금한 지역의 인구 구조를 시각화함

# 03. pandas로 인구 구조 분석하기

## ❖ Step 1) 데이터를 읽어 오기 (1/6)

```
1 import pandas as pd
2 df = pd.read_csv("age.csv", encoding="cp949", index_col=0)
3 df.head()
```

### 실행결과

	2022년08 월_계_총 인구수	2022년08 월_계_연 령구간인 구수	2022년 08월_계_0세	2022년 08월_계_1세	2022년 08월_계_2세	2022년 08월_계_3세	2022년 08월_계_4세	2022년 08월_계_5세	2022년 08월_계_6세	2022년 08월_계_7세	...	2022 년08 월_계_91세	2022 년08 월_계_92세	2022 년08 월_계_93세	2022 년08 월_계_94세	2022 년08 월_계_95세	2022 년08 월_계_96세	2022 년08 월_계_97세	2022 년08 월_계_98세	2022 년08 월_계_99세	2022 년08 월_계_100세 이상
행정구역																					
서울특별시 (1100000000)	9,488,454	9,488,454	40,931	44,731	45,511	49,245	52,278	57,054	64,993	67,389	...	7,820	6,636	5,631	4,113	2,990	1,896	1,333	1,013	950	1,706
서울특별시 종로구 (1111000000)	143,499	143,499	444	484	530	545	580	656	821	781	...	178	141	122	90	75	47	32	22	23	43
서울특별시 종로구 청운효자동 (1111051500)	11,766	11,766	48	46	48	52	52	80													
서울특별시 종로구 사직동 (1111053000)	9,278	9,278	34	30	36	43	45	63													
서울특별시 종로구 삼청동 (1111054000)	2,384	2,384	4	7	5	5	13	8	12	13	...	3	7	1	3	2	1	1	1	0	4

5 rows × 103 columns

데이터를 확인해 보면 숫자 사이에 쉼표(,)가 있음.  
자료형을 확인해 보자

# 03. pandas로 인구 구조 분석하기

## ❖ Step 1) 데이터를 읽어 오기 (2/6)

```
1 df.dtypes
```

### 실행결과

```
2022년08월_계_총인구수
2022년08월_계_연령구간인구수
2022년08월_계_0세
2022년08월_계_1세
2022년08월_계_2세
...
2022년08월_계_96세
2022년08월_계_97세
2022년08월_계_98세
2022년08월_계_99세
2022년08월_계_100세 이상
Length: 103, dtype: object
```

```
object
object
object
object
object
...
object
object
object
object
object
```

정수형 자료형이 아니라 객체(object)임.  
정수형 자료형으로 변환하기 전에,  
숫자 사이에 있는 쉼표(,)부터 제거하자

# 03. pandas로 인구 구조 분석하기

## ❖ Step 1) 데이터를 읽어 오기 (3/6)

```
1 df = df.replace(',', '', regex=True)
2 df.head(3)
```

### 실행결과

	2022년08 월_계_총 인구수	2022년08 월_계_연 령구간인 구수	2022년 08월_계_0세	2022년 08월_계_1세	2022년 08월_계_2세	2022년 08월_계_3세	2022년 08월_계_4세	2022년 08월_계_5세	2022년 08월_계_6세	2022년 08월_계_7세	...	2022 년08 월_계_91세	2022 년08 월_계_92세	2022 년08 월_계_93세	2022 년08 월_계_94세	2022 년08 월_계_95세	2022 년08 월_계_96세	2022 년08 월_계_97세	2022 년08 월_계_98세	2022 년08 월_계_99세	2022 년08 월_계_100세 이상
행정구역																					
서울특별시 (1100000000)	9488454	9488454	40931	44731	45511	49245	52278	57054	64993	67389	...	7820	6636	5631	4113	2990	1896	1333	1013	950	1706
서울특별시 중 로구 (1111000000)	143499	143499	444	484	530	545	580	656	821	781	...	178	141	122	90	75	47	32	22	23	43
서울특별시 중 로구 청운효자 동 (1111051500)	11766	11766	48	46	48	52	52	80	86	88	...	15	7	7	5	6	4	3	3	1	3

3 rows × 103 columns

이제 정수형 자료형으로 변환해 보자

- ✓ replace( ) 함수를 이용하면 전체 데이터 프레임에서 쉼표(,)를 한 번에 "으로 바꿀 수가 있음
- ✓ regex 옵션은 regular expression의 약자로, 정규 표현식으로 문자열이 완전히 일치하지 않더라도 문자열의 일부분만 치환하고 싶을 경우 True로 설정함



## 03. pandas로 인구 구조 분석하기

### ❖ Step 1) 데이터를 읽어 오기 (4/6)

```
1 df = df.apply(pd.to_numeric)
2 df.dtypes
```

#### 실행결과

```
2022년08월_계_총인구수      int64
2022년08월_계_연령구간인구수  int64
2022년08월_계_0세          int64
2022년08월_계_1세          int64
2022년08월_계_2세          int64
...
2022년08월_계_96세          int64
2022년08월_계_97세          int64
2022년08월_계_98세          int64
2022년08월_계_99세          int64
2022년08월_계_100세 이상    int64
Length: 103, dtype: object
```

객체(object)에서  
정수형 자료형 'int64'로 변경됨

# 03. pandas로 인구 구조 분석하기

## ❖ Step 1) 데이터를 읽어 오기 (5/6)

```
1 df = df.div(df[“2022년08월_계_총인구수”], axis=“index”)
2 df.head(3)
```

① 전체 데이터를 총 인구수로 나누어 비율로 변환함

### 실행결과

	2022 년08 월_계_ 총인 구수	2022 년08 월_계_ 연령 구간 인구 수	2022년 08월_계_ _0세	2022년 08월_계_ _1세	2022년 08월_계_ _2세	2022년 08월_계_ _3세	2022년 08월_계_ _4세	2022년 08월_계_ _5세	2022년 08월_계_ _6세	2022년 08월_계_ _7세	...	2022년 08월_계_ _91세	2022년 08월_계_ _92세	2022년 08월_계_ _93세	2022년 08월_계_ _94세	2022년 08월_계_ _95세	2022년 08월_계_ _96세	2022년 08월_계_ _97세	...
행정구역																			
서울특별시 (1100000000)	1.0	1.0	0.004314	0.004714	0.004796	0.005190	0.005510	0.006013	0.006850	0.007102	...	0.000824	0.000699	0.000593	0.000433	0.000315	0.000200	0.000140	...
서울특별시 중 로구 (1111000000)	1.0	1.0	0.003094	0.003373	0.003693	0.003798	0.004042	0.004571	0.005721	0.005443	...	0.001240	0.000983	0.000850	0.000627	0.000523	0.000328	0.000223	...
서울특별시 중 로구 청운효자 동 (1111051500)	1.0	1.0	0.004080	0.003910	0.004080	0.004420	0.004420	0.006799	0.007309	0.007479	...	0.001275	0.000595	0.000595	0.000425	0.000510	0.000340	0.000255	...

3 rows × 103 columns

# 03. pandas로 인구 구조 분석하기

## ❖ Step 1) 데이터를 읽어 오기 (6/6)

```
1 del df["2022년08월_계_총인구수"], df["2022년08월_계_연령구간인구수"]
2 df.head(3)
```

② 총 인구수와 연령 구간 인구수를 삭제함

### 실행결과

	2022년 08월_계 _0세	2022년 08월_계 _1세	2022년 08월_계 _2세	2022년 08월_계 _3세	2022년 08월_계 _4세	2022년 08월_계 _5세	2022년 08월_계 _6세	2022년 08월_계 _7세	2022년 08월_계 _8세	2022년 08월_계 _9세	...	2022년 08월_계 _91세	2022년 08월_계 _92세	2022년 08월_계 _93세	2022년 08월_계 _94세	2022년 08월_계 _95세	2022년 08월_계 _96세	...
행정구역																		
서울특별시 (1100000000)	0.004314	0.004714	0.004796	0.005190	0.005510	0.006013	0.006850	0.007102	0.007004	0.007349	...	0.000824	0.000699	0.000593	0.000433	0.000315	0.000200	...
서울특별시 중 로구 (1111000000)	0.003094	0.003373	0.003693	0.003798	0.004042	0.004571	0.005721	0.005443	0.005993	0.006209	...	0.001240	0.000983	0.000850	0.000627	0.000523	0.000328	...
서울특별시 중 로구 청운효자 동 (1111051500)	0.004080	0.003910	0.004080	0.004420	0.004420	0.006799	0.007309	0.007479	0.008414	0.008074	...	0.001275	0.000595	0.000595	0.000425	0.000510	0.000340	

3 rows × 101 columns

# 03. pandas로 인구 구조 분석하기

## ❖ Steps 2~3) 궁금한 지역의 이름을 입력 받고 해당 지역의 인구 구조를 저장하기 (1/3)

```
1 name = input("원하는 지역의 이름을 입력하세요: ")
2
3 a = df.index.str.contains(name)
4
5 df2 = df[a]
6 df2
```

**df.index.str.contains( ) 함수는**  
데이터 프레임의 인덱스 문자열로부터,  
원하는 문자열이 포함된 행을 찾아냄

### 실행결과

원하는 지역의 이름을 입력하세요: **신도림**

2022년	2022년	2022년	2022년	2022년	2022년	2022년	2022년	2022년	2022년	2022년	...	2022년	2022년	2022년	2022년	2022년	2022년	...
08월_계	08월_계	08월_계	08월_계	08월_계	08월_계	08월_계	08월_계	08월_계	08월_계	08월_계	...	08월_계	08월_계	08월_계	08월_계	08월_계	08월_계	...
_0세	_1세	_2세	_3세	_4세	_5세	_6세	_7세	_8세	_9세	_9세	...	_91세	_92세	_93세	_94세	_95세	_96세	...

행정구역

서울특별시 구																		
로구 신도림동	0.007039	0.007511	0.007484	0.00868	0.008096	0.008708	0.01085	0.011462	0.010739	0.010266	...	0.000612	0.000167	0.000417	0.000362	0.000111	0.000223	...
(1153051000)																		

1 rows × 101 columns

## 03. pandas로 인구 구조 분석하기

❖ Steps 2~3) 궁금한 지역의 이름을 입력 받고 해당 지역의 인구 구조를 저장하기 (2/3)

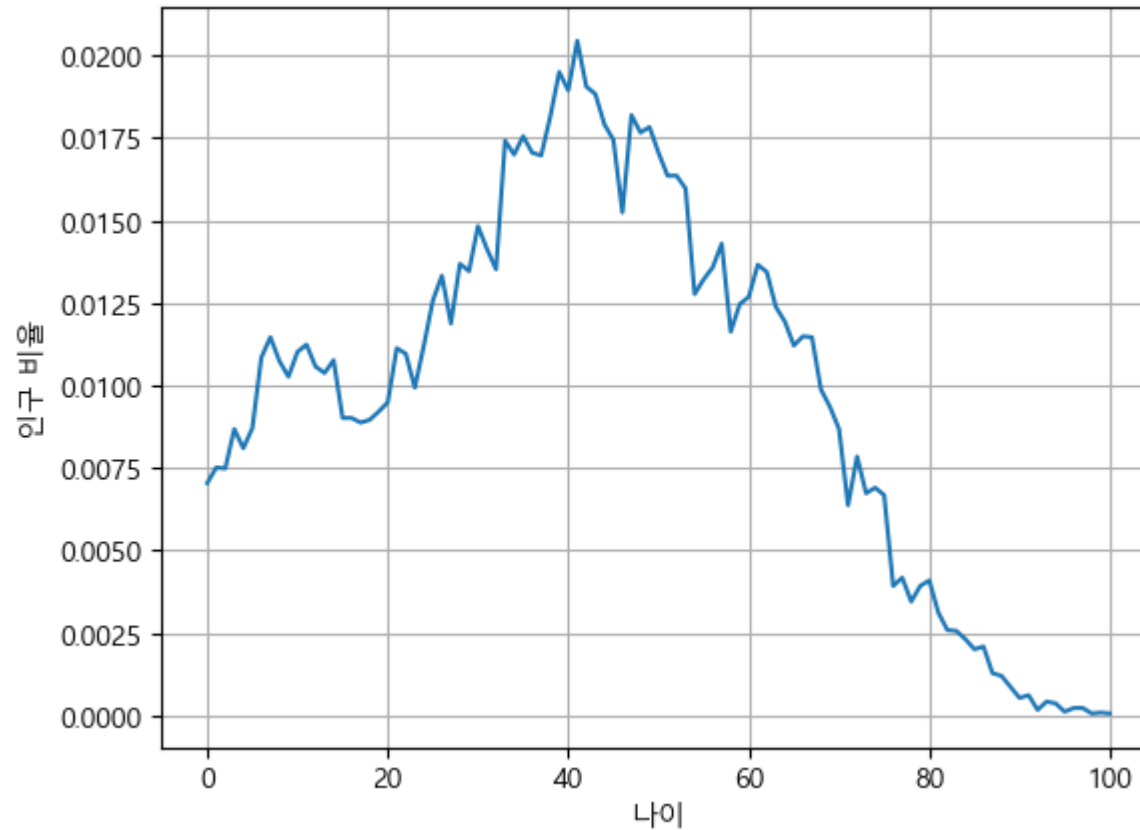
```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 df = pd.read_csv("age.csv", encoding="cp949", index_col=0)
6 df = df.replace(',', '', regex=True)
7 df = df.apply(pd.to_numeric)
8 df = df.div(df["2022년08월_계_총인구수"], axis="index")
9 del df["2022년08월_계_총인구수"], df["2022년08월_계_연령구간인구수"]
10
11 name = input("원하는 지역의 이름을 입력하세요: ")
12 a = df.index.str.contains(name)
13 df2 = df[a]
14
15 plt.rc("font", family="Malgun Gothic")
16 plt.plot(np.arange(0, 101, 1), df2.iloc[0])
17 plt.xlabel("나이")
18 plt.ylabel("인구 비율")
19 plt.grid()
20 plt.show()
```

## 03. pandas로 인구 구조 분석하기

❖ Steps 2~3) 궁금한 지역의 이름을 입력 받고 해당 지역의 인구 구조를 저장하기 (3/3)

### 실행결과

원하는 지역의 이름을 입력하세요: **신도림**



# 03. pandas로 인구 구조 분석하기

❖ Steps 4~5) 궁금한 지역의 인구 구조와 가장 비슷한 인구 구조를 가진 지역의 인구 구조를 시각화하기 (1/6)

```
1 # A의 인구 비율에서 B의 인구 비율을 뺀
2 x = df.sub(df2.iloc[0], axis="columns") # axis=1이라고 적어도 됨
3 x.head(3)
```

## 실행결과

	2022년08 월_계_0 세	2022년08 월_계_1 세	2022년08 월_계_2 세	2022년08 월_계_3 세	2022년08 월_계_4 세	2022년08 월_계_5 세	2022년08 월_계_6 세	2022년08 월_계_7 세	2022년08 월_계_8 세	2022년08 월_계_9 세	...	2022년 08월_계 _91세	2022년 08월_계 _92세	2022년 08월_계 _93세	2022년 08월_계 _94세	2022년 08월_계 _95세	...
행정구역																	
서울특별시 (1100000000)	-0.002725	-0.002797	-0.002687	-0.003490	-0.002586	-0.002695	-0.004000	-0.004360	-0.003735	-0.002917	...	0.000212	0.000532	0.000176	0.000072	0.000204	...
서울특별시 중 로구 (1111000000)	-0.003944	-0.004139	-0.003790	-0.004882	-0.004054	-0.004136	-0.005129	-0.006019	-0.004746	-0.004057	...	0.000628	0.000816	0.000433	0.000266	0.000411	...
서울특별시 중 로구 청운효자 동 (1111051500)	-0.002959	-0.003602	-0.003404	-0.004260	-0.003676	-0.001908	-0.003541	-0.003983	-0.002325	-0.002192	...	0.000663	0.000428	0.000178	0.000063	0.000399	

3 rows × 101 columns

## 03. pandas로 인구 구조 분석하기

❖ Steps 4~5) 궁금한 지역의 인구 구조와 가장 비슷한 인구 구조를 가진 지역의 인구 구조를 시각화하기 (2/6)

```
1 # 차이의 제곱 값을 모두 더함
2 y = np.power(x, 2)
3 z = y.sum(axis=1)
4 z
```

### 실행결과

```
행정구역
서울특별시 (1100000000) 0.000582
서울특별시 종로구 (1111000000) 0.001165
서울특별시 종로구 청운효자동(1111051500) 0.000540
서울특별시 종로구 사직동(1111053000) 0.000796
서울특별시 종로구 삼청동(1111054000) 0.001783
...
제주특별자치도 서귀포시 서홍동(5013058000) 0.000639
제주특별자치도 서귀포시 대륜동(5013059000) 0.000287
제주특별자치도 서귀포시 대천동(5013060000) 0.000311
제주특별자치도 서귀포시 중문동(5013061000) 0.000328
제주특별자치도 서귀포시 예래동(5013062000) 0.002025
Length: 3866, dtype: float64
```



# 03. pandas로 인구 구조 분석하기

❖ Steps 4~5) 궁금한 지역의 인구 구조와 가장 비슷한 인구 구조를 가진 지역의 인구 구조를 시각화하기 (3/6)

```
1 z = z[z[:, ] != 0]
```

age.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
826	대구광역시 달성군 가창면(2771031000)	7,784	7,784	24	25	27	29	33	49	44	42	48	68	53	59	36
827	대구광역시 달성군 하빈면(2771033000)	3,606	3,606	7	12	7	6	3	5	10	10	14	16	10	18	7
828	대구광역시 달성군 구지면(2771038000)	18,766	18,766	257	288	268	305	302	278	273	270	221	238	199	201	138
829	인천광역시 (2800000000)	2,946,319	2,946,319	14,611	16,087	18,070	19,316	21,083	24,083	25,953	25,278	25,591	28,122	28,000	27,245	26,069
830	인천광역시 중구 (2811000000)	143,052	143,052	801	828	1,036	1,081	1,171	1,260	1,391	1,306	1,349	1,460	1,427	1,441	1,260
831	인천광역시 중구 연안동(2811052000)	6,141	6,141	15	11	18	18	14	28	23	29	33	32	30	29	28
832	인천광역시 중구 신포동(2811053000)	5,094	5,094	17	14	16	15	20	26	21	31	27	43	42	26	28
833	인천광역시 중구 신흥동(2811054000)	13,202	13,202	53	66	86	63	79	88	77	88	97	105	93	119	95
834	인천광역시 중구 도원동(2811056000)	3,919	3,919	6	6	4	11	14	17	12	13	13	21	18	24	23
835	인천광역시 중구 율목동(2811057000)	3,193	3,193	1	3	8	5	10	5	15	8	11	14	15	23	14
836	인천광역시 중구 동인천동(2811058500)	5,825	5,825	15	15	16	17	13	17	26	16	21	32	24	26	24
837	인천광역시 중구 개항동(2811061500)	7,040	7,040	14	10	19	22	29	24	28	30	36	35	32	38	34
838	인천광역시 중구 영종동(2811062000)	19,016	19,016	79	110	107	132	119	125	118	121	114	164	148	150	148
839	인천광역시 중구 영종1동(2811062200)	43,784	43,784	464	469	614	653	687	760	858	762	771	786	743	725	564
840	인천광역시 중구 운서동(2811062800)	31,886	31,886	131	118	144	140	180	158	207	200	216	213	272	271	293
841	인천광역시 중구 용유동(2811063000)	3,952	3,952	6	6	4	5	6	12	6	8	10	15	10	10	9
842	인천광역시 중구영종출장소 (2811400000)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
843	인천광역시 중구용유출장소 (2811800000)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
844	인천광역시 동구 (2814000000)	61,716	61,716	200	233	291	310	307	405	444	420	391	506	469	498	459
845	인천광역시 동구 만석동(2814051000)	6,936	6,936	32	36	35	35	36	45	48	59	35	65	61	61	53
846	인천광역시 동구 화수1.화평동(2814052500)	5,945	5,945	11	11	12	13	21	22	24	25	44	36	40	35	37
847	인천광역시 동구 화수2동(2814053000)	7,613	7,613	17	18	12	22	22	34	49	40	35	60	66	62	54

age.csv 파일 내 일부 행정구역의 연령별 인구수 정보가 전부 0인 경우가 있음.  
이 행정구역들의 경우 z 값이 0으로 계산되기 때문에 제외시킴

## 03. pandas로 인구 구조 분석하기

❖ Steps 4~5) 궁금한 지역의 인구 구조와 가장 비슷한 인구 구조를 가진 지역의 인구 구조를 시각화하기 (4/6)

```
1 idx = z.sort_values().index[:5]
2 idx
```

### 실행결과

```
Index(['경기도 하남시 (4145000000)', '경기도 남양주시 별내동(4136057000)',
      '서울특별시 영등포구 문래동(1156060500)', '충청남도 아산시 (4420000000)',
      '서울특별시 영등포구 신길제7동(1156069000)'],
      dtype='object', name='행정구역')
```

**sort\_values( )** 함수를 활용하여 오름차순으로 정렬하고,  
**index[:5]**를 활용하여 차이가 가장 작은 지역 5곳을 찾음

## 03. pandas로 인구 구조 분석하기

❖ Steps 4~5) 궁금한 지역의 인구 구조와 가장 비슷한 인구 구조를 가진 지역의 인구 구조를 시각화하기 (5/6)

```
1 plt.rc("font", family="Malgun Gothic")
2 plt.rcParams["axes.unicode_minus"] = False
3 plt.figure(figsize=(10, 6))
4 plt.title(name + " 지역과 가장 비슷한 인구 구조를 가진 지역")
5 for j in range(0, 5, 1):
6     plt.plot(np.arange(0, 101, 1), df.loc[idx[j]], label=idx[j])
7
8 plt.xlabel("나이")
9 plt.ylabel("인구 비율")
10 plt.legend()
11 plt.grid()
12 plt.show()
```

- ✓ loc은 location의 약자로 인덱스를 기준으로 행 데이터를 읽기 위해서 사용됨
- ✓ [참고] iloc은 행 번호를 기준으로 행 데이터를 읽기 위해서 사용됨

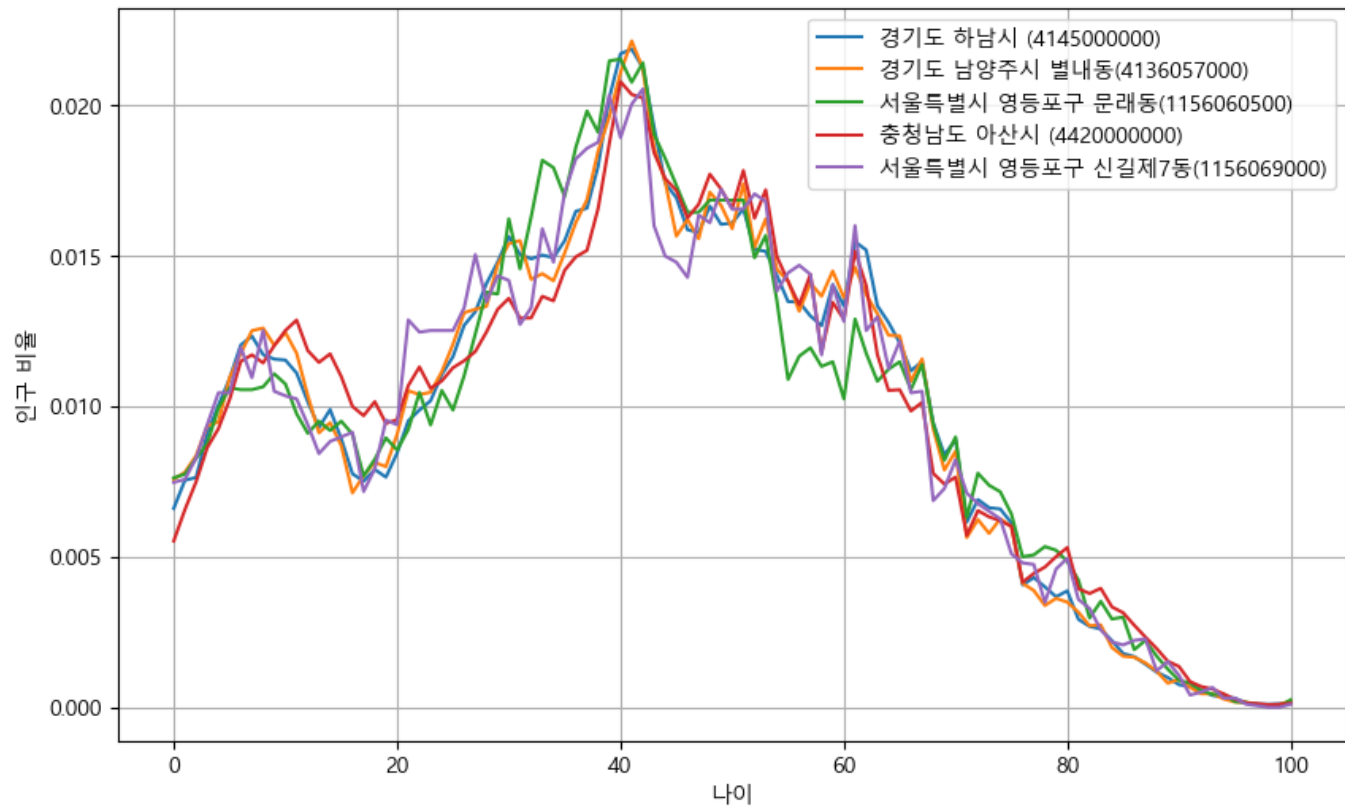
# 03. pandas로 인구 구조 분석하기

❖ Steps 4~5) 궁금한 지역의 인구 구조와 가장 비슷한 인구 구조를 가진 지역의 인구 구조를 시각화하기 (6/6)

## 실행결과

원하는 지역의 이름을 입력하세요: **신도림**

신도림 지역과 가장 비슷한 인구 구조를 가진 지역



## 03. pandas로 인구 구조 분석하기

### ❖ 전체 코드 (1/2)

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 df = pd.read_csv("age.csv", encoding="cp949", index_col=0)
6 df = df.replace(',', '', regex=True)
7 df = df.apply(pd.to_numeric)
8 df = df.div(df["2022년08월_계_총인구수"], axis="index")
9 del df["2022년08월_계_총인구수"], df["2022년08월_계_연령구간인구수"]
10
11 name = input("원하는 지역의 이름을 입력하세요: ")
12 a = df.index.str.contains(name)
13 df2 = df[a]      # 궁금한 지역(A)의 인구 구조 정보가 저장돼 있음
14
15 # A의 인구 비율에서 B의 인구 비율을 뺀
16 x = df.sub(df2.iloc[0], axis="columns") # axis=1이라고 적어도 됨
17
18 # 차이의 제곱 값을 모두 더함
19 y = np.power(x, 2)
20 z = y.sum(axis=1)
```

## 03. pandas로 인구 구조 분석하기

### ❖ 전체 코드 (2/2)

```
21 z = z[z[:] != 0]
22
23 idx = z.sort_values().index[:5]
24
25 plt.rc("font", family="Malgun Gothic")
26 plt.rcParams["axes.unicode_minus"] = False
27 plt.figure(figsize=(10, 6))
28 plt.title(name + " 지역과 가장 비슷한 인구 구조를 가진 지역")
29 for j in range(0, 5, 1):
30     plt.plot(np.arange(0, 101, 1), df.loc[idx[j]], label=idx[j])
31
32 plt.xlabel("나이")
33 plt.ylabel("인구 비율")
34 plt.legend()
35 plt.grid()
36 plt.show()
```

- ❖ 01. 테이블 형태의 데이터를 쉽게 다루도록 돕는 pandas 라이브러리
- ❖ 02. 위키피디아 데이터 엑셀로 저장하기
- ❖ 03. pandas로 인구 구조 분석하기

# THANK YOU!

## Q & A

- Name: 권범
- Office: 동덕여자대학교 인문관 B821호
- Phone: 02-940-4752
- E-mail: [bkwon@dongduk.ac.kr](mailto:bkwon@dongduk.ac.kr)