

파이썬 라이브러리를 활용한 데이터 분석

5장: 판다스 시작하기

판다스

• 개요

- 표 형식의 데이터나 다양한 형태의 테이블을 처리하기 위한 라이브러리
 - 2010년부터, 약 800여명의 기여자가 활동
- 주 자료 구조
 - 시리지와 데이터프레임

Pandas 에서 사용되는 대표적인 데이터 오브젝트

시리즈 (Series)

Series 는 1차원 배열의 형태를 갖는다.
인덱스(노란색)라는 한 가지 기준에
의하여 데이터가 저장된다.

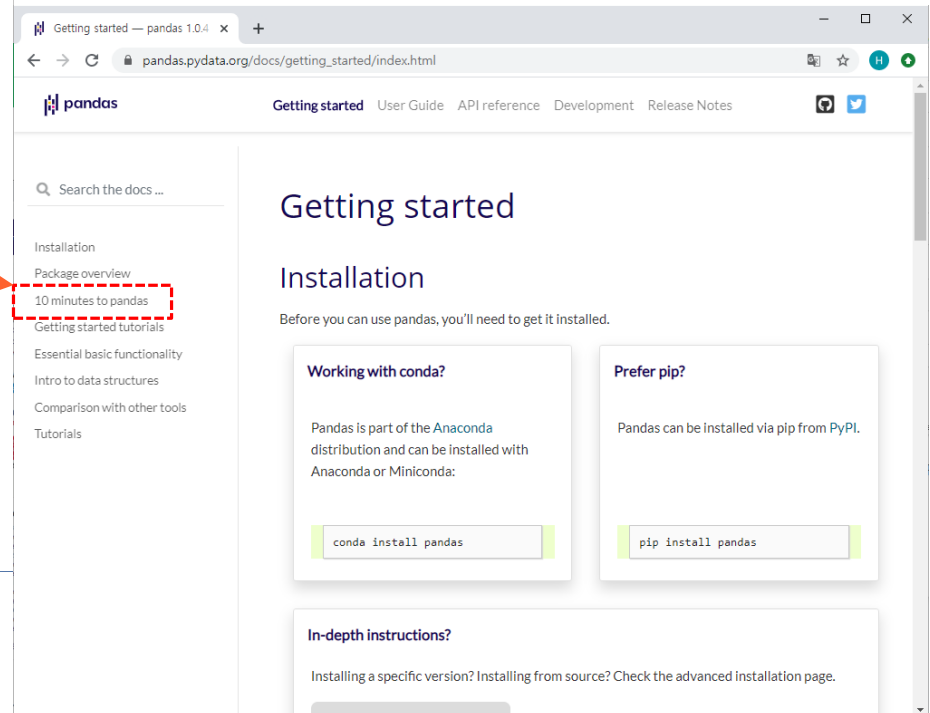
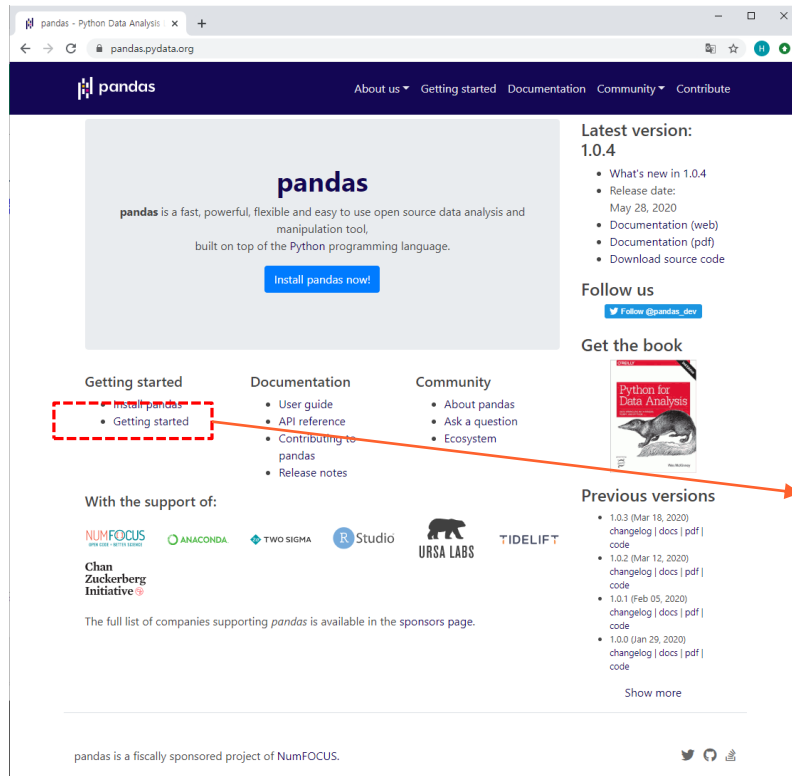
데이터프레임 (DataFrame)

DataFrame 은 2차원 배열의 형태를 갖는다.
인덱스(노란색)와 컬럼(파란색)이라는 두 가지
기준에 의하여 표 형태처럼 데이터가 저장된다.

dandyrilla.github.io

판다스 홈페이지

- **pandas.pydata.org**



참고 사이트

- **교재 깃허브**

- <https://github.com/wesm/pydata-book>

- **한글**

- <https://dandyrilla.github.io/2017-08-12/pandas-10min/>
- <https://dataitgirls2.github.io/10minutes2pandas/>
- <https://datascienceschool.net/view-notebook/ee0a5679dd574b94b55193690992f850/>

- **영어**

- <https://www.geeksforgeeks.org/python-pandas-dataframe/>
- <https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/>
- <https://www.kaggle.com/residentmario/creating-reading-and-writing>
- <https://www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python>
- <https://data36.com/pandas-tutorial-1-basics-reading-data-files-dataframes-data-selection/>
- <https://towardsdatascience.com/my-python-pandas-cheat-sheet-746b11e44368>

파일 my-ch05-study.ipynb

Introduction to pandas Data Structures

Series

```
In [6]: obj = pd.Series([4, 7, -5, 3])  
obj
```

```
Out [6]: 0    4  
         1    7  
         2   -5  
         3    3  
         dtype: int64
```

```
In [7]: obj.values  
obj.index # like range(4)
```

```
Out [7]: RangeIndex(start=0, stop=4, step=1)
```

```
In [8]: obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])  
obj2
```

```
Out [8]: d    4  
         b    7  
         a   -5  
         c    3  
         dtype: int64
```

```
In [9]: obj2.index
```

```
Out [9]: Index(['d', 'b', 'a', 'c'], dtype='object')
```

Series 개요

• pd.Series

- NumPy에서 제공하는 1차원 배열과 비슷
- 각 데이터의 의미를 표시하는, 레이블(label)이라 하는 인덱스(index)를 붙일 수 있고
- 데이터 자체는 값(value)

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

```
In [2]: x = pd.Series([1, 2, 2, np.nan], index=['p', 'q', 'r', 's'])
x
```

```
Out [2]: p    1.0
         q    2.0
         r    2.0
         s    NaN
         dtype: float64
```

pd.Series([1, 2, 2, np.nan], index=['p', 'q', 'r', 's'])

		Data
Index	p	1.0
	q	2.0
	r	2.0
	s	NaN

dtype: float64

Series

© w3resource.com

```
In [3]: y = pd.Series([2, np.nan, 1, 1], index=['p', 'q', 's', 't'])
y
```

```
Out [3]: p    2.0
         q    NaN
         s    1.0
         t    1.0
         dtype: float64
```

pd.Series([2, np.nan, 1, 1], index=['p', 'q', 's', 't'])

		Data
Index	p	2.0
	q	NaN
	s	1.0
	t	1.0

dtype: float64

Series

© w3resource.com

DataFrame 개요

- **pd.DataFrame**

- R의 dataframe 데이터 타입을 참고하여 만든 것이 바로 pandas DataFrame
- 테이블 형태의 자료
 - 행과 열을 인덱스(index)와 칼럼(columns)으로 구분

	A	B	C	D	E	F	G	H	I
1	Order Date	OrderID	Salesperson	UK Units	UK Order Amt	USA Units	USA Order Amt	Total Units	Total Order Amt
2	1/01/2011	10392	Fuller			13	1440	13	1440
3	2/01/2011	10397	Gloucester	17	716.72			17	716.72
4	2/01/2011	10771	Bromley	18	344			18	344
5	3/01/2011	10393	Finchley			16	2556.95	16	2556.95
6	3/01/2011	10394	Finchley			10	442	10	442
7	3/01/2011	10395	Gillingham	9	2122.92			9	2122.92
8	6/01/2011	10396	Finchley			7	1903.8	7	1903.8
9	8/01/2011	10399	Callahan			17	1765.6	17	1765.6
10	8/01/2011	10404	Fuller			7	1591.25	7	1591.25
11	9/01/2011	10398	Fuller			11	2505.6	11	2505.6
12	9/01/2011	10403	Coghill	18	855.01			18	855.01
13	10/01/2011	10401	Finchley			7	3868.6	7	3868.6

DataFrame 이해

- 여러 시리즈의 모임

Series			Series			DataFrame		
	apples			oranges			apples	oranges
0	3		0	0		0	3	0
1	2	+	1	3	=	1	2	3
2	0		2	7		2	0	7
3	1		3	2		3	1	2

- 인덱스와 칼럼

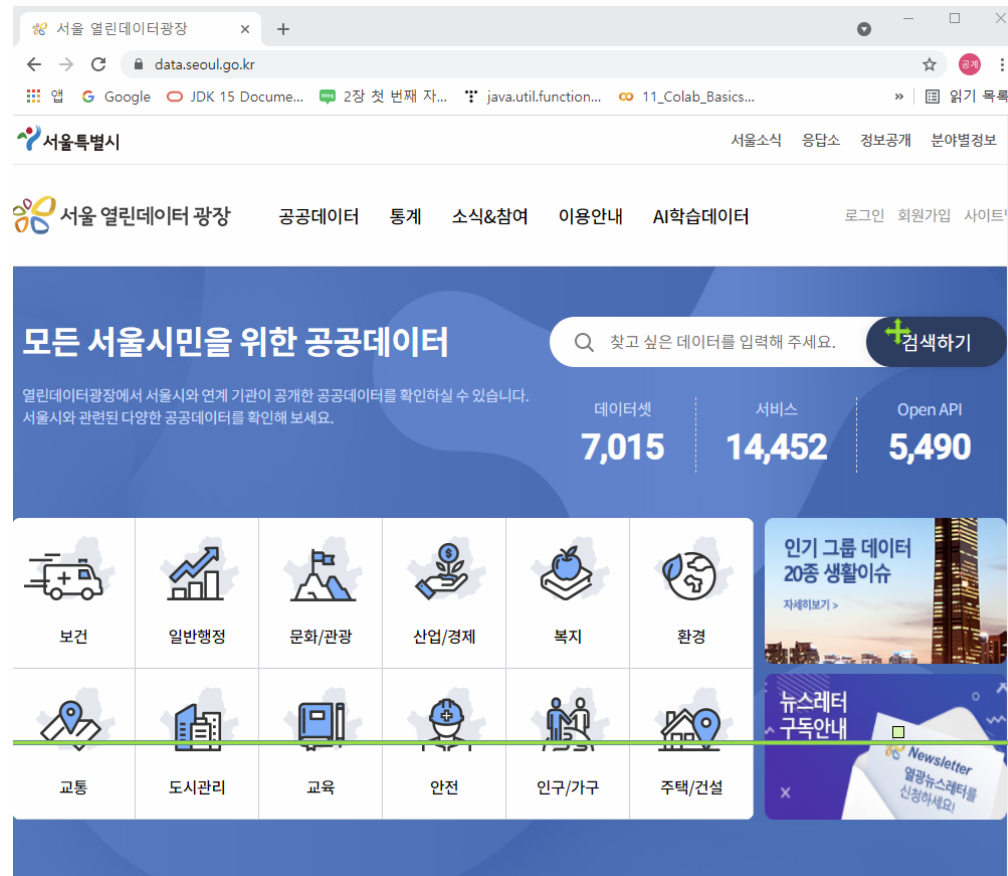
Columns					
	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Rows index

Data

실제 공공 데이터

- 서울 열린데이터 광장
 - <https://data.seoul.go.kr/>



지하철 시간대별 승하차 인원

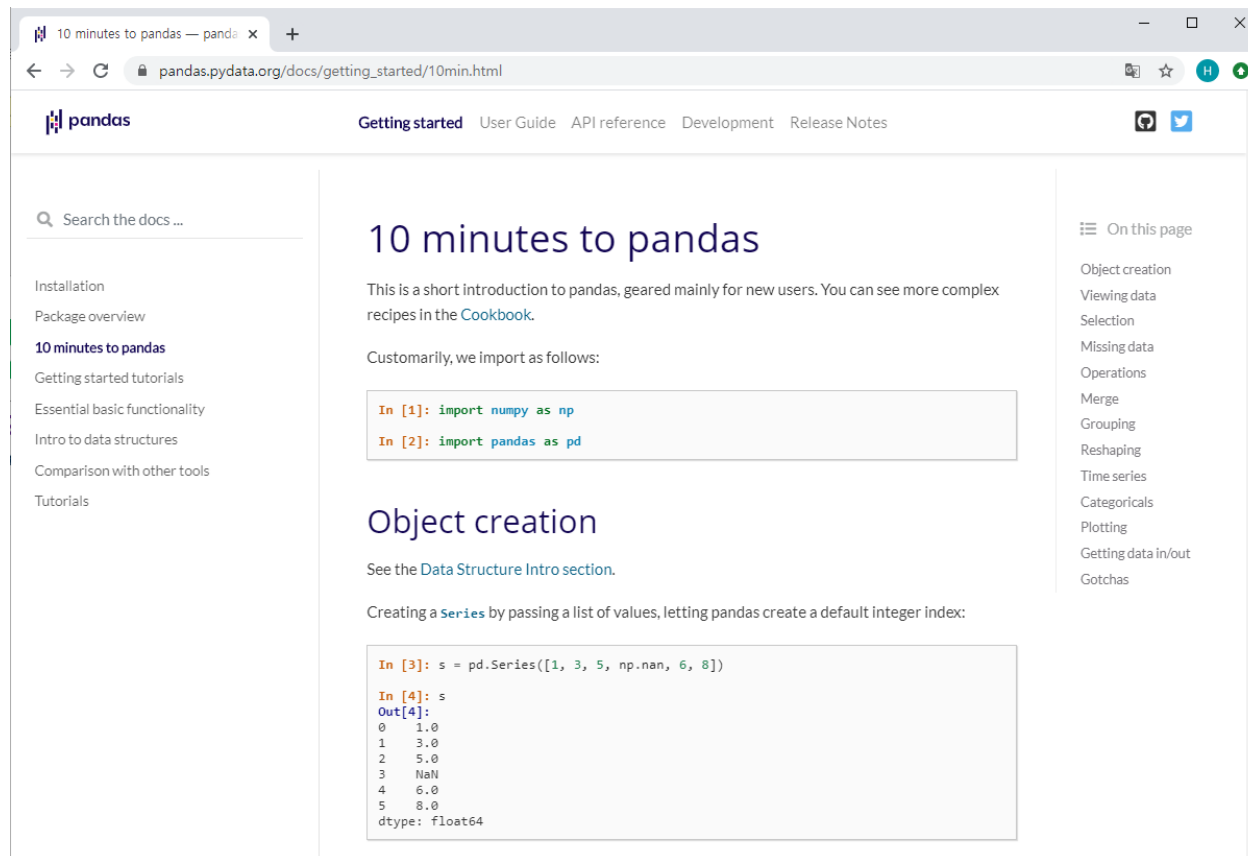
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
	USE_MON	LINE_NUM	SUB_STA	FOUR_RID	FOUR_ALI	FIVE_RIDE	FIVE_ALI	SIX_RIDE	SIX_ALI	SEVEN_RID	SEVEN_AL	EIGHT_RID	EIGHT_ALI	NINE_RIDE	NINE_ALI	TEN_RIDE
0	202103	1호선	서울역	668	19	9048	5684	11314	32628	35812	85372	56518	181087	41700	117196	40292
1	202103	1호선	동묘앞	73	1	2888	815	3453	4739	5946	9283	9005	25504	7258	16394	8616
2	202103	1호선	시청	20	0	1722	3784	2707	18444	6099	60071	7574	162609	7807	73107	8733
3	202103	1호선	종각	147	1	2181	4199	2947	21373	5269	93962	8253	210534	9686	126119	13850
4	202103	1호선	종로3가	147	29	2834	3189	2790	10922	4327	24475	7531	68562	10178	70576	17593
5	202103	1호선	종로5가	106	1	1780	3491	2397	15045	4574	38776	7282	106368	10177	62287	17191
6	202103	1호선	동대문	607	10	12572	1965	8945	6697	14117	11879	18264	24742	16256	21527	14922
7	202103	1호선	신설동	361	13	7927	2898	7826	9755	17350	25202	23741	62892	15735	32839	14025
8	202103	1호선	제기동	373	6	4650	2053	7789	7776	21328	18137	29637	41168	20567	34379	23281
9	202103	1호선	청량리(서)	1015	15	9682	4140	14053	19319	36259	16663	44413	35386	30533	31168	29734
10	202103	2호선	시청	8	0	864	1386	1860	13764	4734	68044	6668	203885	8296	80047	10938

10 minutes to pandas

- 실제로는 약 3시간 정도 소요

- https://pandas.pydata.org/docs/getting_started/10min.html

- 약 150 여개의 문장

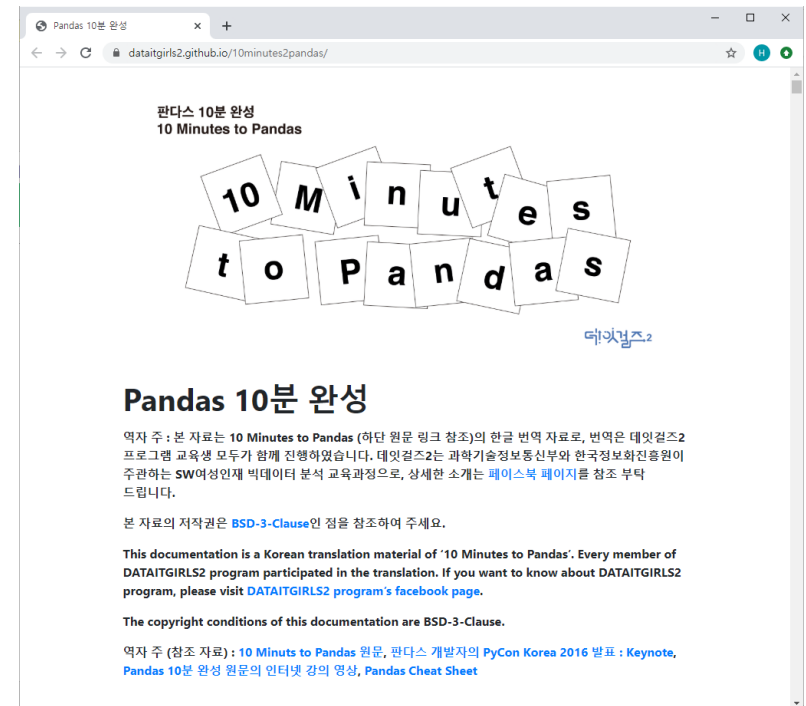
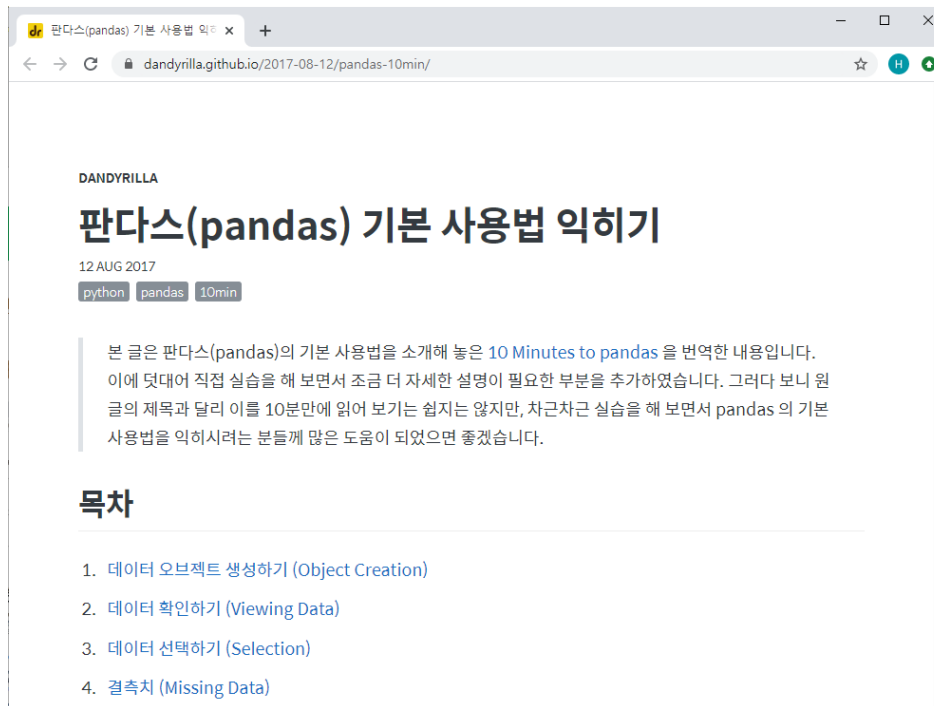


판다스 10분 노트북 파일

- '10 minutes to pandas ipynb'로 검색
 - <https://github.com/takenory/10-minutes-to-pandas/blob/master/notebooks/10-minutes-to-pandas.ipynb>
- Colab 접속 URL
 - <https://colab.research.google.com>
 - 다음으로 계속 연결
 - [github/사용자명/저장소/파일명](#)
- 다음으로 colab에서 실행
 - <https://colab.research.google.com/github/takenory/10-minutes-to-pandas/blob/master/notebooks/10-minutes-to-pandas.ipynb>

한글 판다스 10분

- <https://dandyrilla.github.io/2017-08-12/pandas-10min/>
 — 설명까지 자세히
- <https://dataitgirls2.github.io/10minutes2pandas/>



5. 판다스 시작하기

- 판다스 특징

- 표 형식의 데이터, 다양한 데이터를 처리하는 것에 초점
- Numpy: 단일 산술 배열 데이터 처리에 특화
- 800여명이 함께 개발하는 공개 SW

시리즈 생성

- 일련의 객체를 담은 1차원 배열 구조
 - 색인(index): 배열의 데이터와 연관된 이름
 - 지정하지 않으면 [0, ... N-1]로 자동 지정
 - `pd.Series([4, 7, -5, 3])`
 - 속성: values, index, array

```
> s = pd.Series([3, 20, 21],
                 index=['Bei Bei', 'Mei Xiang', 'Tian Tian'],
                 name='Age')
```

```
Bei Bei      3
Mei Xiang    20
Tian Tian    21
Name: Age, dtype: int64
```

Age

Bei Bei	3
Mei Xiang	20
Tian Tian	21

```
> s.array # ``a thin (no copy) wrapper around numpy.ndarray``
<PandasArray>
[3, 20, 21]
Length: 3, dtype: int64
```

시리즈 참조와 값 대입

- 여러 값 선택
 - 색인 리스트 사용

```
In [23]: obj2
Out[23]: d    6
          b    7
          a   -5
          c    3
          dtype: int64

In [24]: obj2[obj2 > 0]
Out[24]: d    6
          b    7
          c    3
          dtype: int64

In [25]: obj2 * 2
Out[25]: d    12
          b    14
          a   -10
          c     6
          dtype: int64

In [26]: np.exp(obj2)
Out[26]: d    403.428793
          b   1096.633158
          a     0.006738
          c   20.085537
          dtype: float64

In [18]: import math
          math.exp(6)
Out[18]: 403.4287934927351

In [19]: 'b' in obj2
Out[19]: True

In [20]: 'e' in obj2
Out[20]: False
```

```
In [10]: obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
          obj2
Out[10]: d    4
          b    7
          a   -5
          c    3
          dtype: int64

In [11]: obj2.index
Out[11]: Index(['d', 'b', 'a', 'c'], dtype='object')

In [12]: obj2['a']
Out[12]: -5

In [13]: obj2['d'] = 6
          obj2
Out[13]: d    6
          b    7
          a   -5
          c    3
          dtype: int64

In [14]: obj2[['c', 'a', 'd']]
Out[14]: c    3
          a   -5
          d    6
          dtype: int64

In [15]: obj2
Out[15]: d    6
          b    7
          a   -5
          c    3
          dtype: int64
```


데이터, 색인, NaN

- **pd.Series(사전)**
 - 키는 인덱스
- **pd.Series(데이터, index=리스트)**
 - 기존의 데이터라면
지정한 인덱스를 기
반으로 생성
 - 'Utah'는 지정하
는 index에 없으
므로 빠짐
 - 기존의 데이터에
는 index인
california가 없
으므로 NaN
 - NaN(Not a
Number)
 - np.nan
 - pd.isnull()
 - pd.notnull()

```
In [27]: sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
obj3 = pd.Series(sdata)
obj3
```

```
Out[27]: Ohio      35000
Texas      71000
Oregon     16000
Utah        5000
dtype: int64
```

```
In [28]: states = ['California', 'Ohio', 'Oregon', 'Texas']
obj4 = pd.Series(sdata, index=states)
obj4
```

```
Out[28]: California      NaN
Ohio      35000.0
Oregon     16000.0
Texas      71000.0
dtype: float64
```

여기에 지정하는 index에 utah
가 없으므로 데이터에서 빠짐

기존의 데이터에 는 index인
california가 없으므로 NaN

```
In [29]: states = ['California', 'Ohio', 'Oregon', 'Texas']
obj4 = pd.Series(sdata, index=states, name='usadata')
obj4
```

```
Out[29]: California      NaN
Ohio      35000.0
Oregon     16000.0
Texas      71000.0
Name: usadata, dtype: float64
```

Quiz

- 1. 다음 pandas 프로그램에서 실행 결과가 다른 하나는? (2)
 - `s = pd.Series([2, 3, 5, 7], index=[1, 5, 6, 7])`
 - # 시리즈의 in은 값이 아닌 index의 검사임
 - `print(1 in s)`
 - `print(2 in s)`
 - `print(5 in s)`
 - `print(7 in s)`

산술연산과 속성

- 산술 연산
 - 색인과 라벨로 자동 정렬
 - 피연산자가 해당 색인이 있어야 계산
- 속성
 - name
 - index.name

```
In [35]: obj3
```

```
Out[35]: Ohio      35000
         Texas      71000
         Oregon     16000
         Utah        5000
         dtype: int64
```

```
In [36]: obj4
```

```
Out[36]: California      NaN
         Ohio             35000.0
         Oregon           16000.0
         Texas             71000.0
         Name: usadata, dtype: float64
```

```
In [37]: obj3 + obj4
```

```
Out[37]: California      NaN
         Ohio             70000.0
         Oregon           32000.0
         Texas            142000.0
         Utah             NaN
         dtype: float64
```

```
In [38]: obj4.name = 'population'
         obj4.index.name = 'state'
         obj4
```

```
Out[38]: state
         California      NaN
         Ohio             35000.0
         Oregon           16000.0
         Texas             71000.0
         Name: population, dtype: float64
```

DataFrame 다양한 참조 방법과 삭제



Python pandas Series, DataFrame 행과 열 선택

pd.DataFrame

index	C_1	C_2	C_3
0	11	21	31
1	12	22	32
2	13	23	33
3	14	24	34
4	15	25	35
5	16	26	36

Indexing & Slicing

DataFrame[['C_1', 'C_2']][1:3]

12	22
13	23

DataFrame.iloc[5]

5	16	26	36
---	----	----	----

Select : DataFrame['C_3']

Delete : DataFrame.drop(['C_3'], 1)

축 axis=1로 삭제할
열을 지정할 때 사용,
기본은 행 삭제

<http://rfriend.tistory.com>

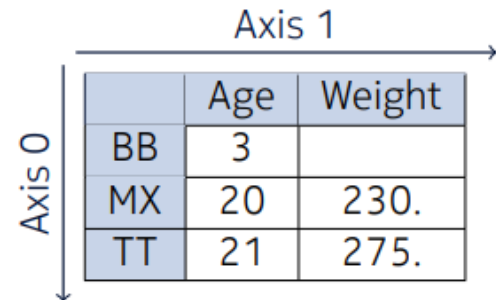
DataFrame 생성

- 열은 서로 다른 종류의 값도 가능

```
> df = pd.DataFrame({'Age': [3, 20, 21], 'Weight': [np.nan, 230., 275.]},
                      index=['Bei Bei', 'Mei Xiang', 'Tian Tian'])
```

```
      Age  Weight
Bei Bei    3    NaN
Mei Xiang  20  230.0
Tian Tian  21  275.0
```

```
> df.dtypes # returns a Series
Age          int64
Weight      float64
dtype: object
```



Axis 1		
	Age	Weight
Axis 0	BB	3
	MX	20
	TT	21

In general: list \approx rows, dictionary \approx columns.

열명 지정

- 옵션 **columns=**
 - 새로 만들 때
 - 기존의 것을 사용시
 - 순서 이동

```
In [39]: data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
                'year': [2000, 2001, 2002, 2001, 2002, 2003],
                'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
frame = pd.DataFrame(data)
frame
```

Out[39]:

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

```
In [35]: frame.head()
```

Out[35]:

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9

```
In [36]: pd.DataFrame(data, columns=['year', 'state', 'pop'])
```

Out[36]:

	year	state	pop
0	2000	Ohio	1.5
1	2001	Ohio	1.7
2	2002	Ohio	3.6
3	2001	Nevada	2.4
4	2002	Nevada	2.9
5	2003	Nevada	3.2

주요 속성

- 결측치는 NaN
- 속성
 - columns
 - 열 색인 값
 - df.칼럼명
 - df['칼럼명']
 - 열 조회

```
In [40]: frame2 = pd.DataFrame(data, columns=['year', 'state', 'pop', 'debt'],
                                index=['one', 'two', 'three', 'four', 'five', 'six'])
frame2
```

```
Out[40]:
```

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN
six	2003	Nevada	3.2	NaN

```
In [41]: frame2.columns
```

```
Out[41]: Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

```
In [42]: frame2['state']
```

```
Out[42]: one      Ohio
two      Ohio
three    Ohio
four     Nevada
five     Nevada
six      Nevada
Name: state, dtype: object
```

```
In [43]: frame2.year
```

```
Out[43]: one      2000
two      2001
three    2002
four     2001
five     2002
six      2003
Name: year, dtype: int64
```

행과 열 참조

- `df.loc['행명']`
- `df['열명']`
 - 없는 열에 대입하면
 - 새로운 열 생성

```
In [41]: frame2.loc['three']
```

```
Out[41]: year      2002
state      Ohio
pop        3.6
debt       NaN
Name: three, dtype: object
```

```
In [42]: frame2['debt'] = 16.5
frame2
```

```
Out[42]:
```

	year	state	pop	debt
one	2000	Ohio	1.5	16.5
two	2001	Ohio	1.7	16.5
three	2002	Ohio	3.6	16.5
four	2001	Nevada	2.4	16.5
five	2002	Nevada	2.9	16.5
six	2003	Nevada	3.2	16.5

```
In [43]: frame2['debt'] = np.arange(6.)
frame2
```

```
Out[43]:
```

	year	state	pop	debt
one	2000	Ohio	1.5	0.0
two	2001	Ohio	1.7	1.0
three	2002	Ohio	3.6	2.0
four	2001	Nevada	2.4	3.0
five	2002	Nevada	2.9	4.0
six	2003	Nevada	3.2	5.0

열 추가와 삭제

- `df['새로운_열명'] = 값`
 - 새로운 열 추가
- `df['기존_열명'] = 값`
 - 값 수정
- `del df['기존_열명']`
 - 기존_열명의 열 삭제

```
In [47]: val = pd.Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])
         frame2['debt'] = val
         frame2
```

Out[47]:

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	-1.2
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	-1.5
five	2002	Nevada	2.9	-1.7
six	2003	Nevada	3.2	NaN

```
In [49]: frame2['eastern'] = frame2.state == 'Ohio'
         frame2
```

Out[49]:

	year	state	pop	debt	eastern
one	2000	Ohio	1.5	NaN	True
two	2001	Ohio	1.7	-1.2	True
three	2002	Ohio	3.6	NaN	True
four	2001	Nevada	2.4	-1.5	False
five	2002	Nevada	2.9	-1.7	False
six	2003	Nevada	3.2	NaN	False

```
In [50]: del frame2['eastern']
         frame2.columns
```

Out[50]: Index(['year', 'state', 'pop', 'debt'], dtype='object')

사전 데이터

- 사전의 키
 - 열명
 - 값의 내부에도 사전이 있다면
 - 키는 인덱스

```
In [51]: pop = {'Nevada': {2001: 2.4, 2002: 2.9},
               'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
```

```
Out[51]: {'Nevada': {2001: 2.4, 2002: 2.9}, 'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
```

```
In [52]: frame3 = pd.DataFrame(pop)
         frame3
```

```
Out[52]:
```

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

```
In [53]: frame3.T
```

```
Out[53]:
```

	2001	2002	2000
Nevada	2.4	2.9	NaN
Ohio	1.7	3.6	1.5

```
In [54]: pd.DataFrame(pop, index=[2001, 2002, 2003])
```

```
Out[54]:
```

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2003	NaN	NaN

행 2001, 2002는 원래 자료로 없던 2003은 모두 NaN로 저장

```
In [55]: frame3
```

```
Out[55]:
```

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

주요 속성

- `index.name`
- `columns.name`
- `values`

```
In [55]: frame3
```

```
Out[55]:
```

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

```
In [52]: pdata = {'Ohio': frame3['Ohio'][:-1],
                  'Nevada': frame3['Nevada'][:2]}
          pd.DataFrame(pdata)
```

```
Out[52]:
```

	Ohio	Nevada
2001	1.7	2.4
2002	3.6	2.9

```
In [56]: frame3.index.name = 'year';
          frame3.columns.name = 'state'
          frame3
```

```
Out[56]:
```

state	Nevada	Ohio
year		
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

```
In [57]: frame3.values
```

```
Out[57]: array([[2.4, 1.7],
                 [2.9, 3.6],
                 [nan, 1.5]])
```

색인

- 색인
 - 중복을 허락

```
In [65]: frame3
```

```
Out[65]:
```

state	Nevada	Ohio
year		
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

```
In [66]: frame3.columns
```

```
Out[66]: Index(['Nevada', 'Ohio'], dtype='object', name='state')
```

```
In [67]: 'Ohio' in frame3.columns
```

```
Out[67]: True
```

```
In [68]: 2003 in frame3.index
```

```
Out[68]: False
```

```
In [69]: dup_labels = pd.Index(['foo', 'foo', 'bar', 'bar'])
dup_labels
```

```
Out[69]: Index(['foo', 'foo', 'bar', 'bar'], dtype='object')
```

Quiz

- 2. 다음 pandas 프로그램에서 실행 결과가 다른 하나는? (4)
 - `df = pd.DataFrame({`
 - `"이름": ['이적', '허승', '강소라', '강적'],`
 - `"남여구분": ['남', '남', '여', '남'],`
 - `"학점": [3.2, 2.3, 4.3, 2.9]`
 - `})`

 - `print(type(df.loc[3]))`
 - `print(type(df.이름))`
 - `print(type(df['학점']))`
 - `print(type(df.index))`