

[TA 3-2]

NumPy & Pandas Primer

빅데이터프로그래밍기초

박진수 교수

Big Data Institute,
Seoul National University

NumPy



NumPy란?

· 계산과학(Computation science) 분야에 자주 활용되는 Python 라이브러리

- 고성능의 다차원 배열(array) 객체와 이를 다룰 수 있는 다양한 도구를 지원

- 문법이나 구동 방식이 MATLAB과 비슷함

- NumPy 불러오기

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

```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]  
array([ 1, 12, 23, 34, 45])
```

```
>>> a[3:,[0, 2, 5]]  
array([[30, 32, 35],  
       [40, 42, 45]],  
      [50, 52, 55])
```

```
>>> mask = array([1,0,1,0,0,1],  
                  dtype=bool)
```

```
>>> a[mask,2]  
array([2,22,52])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

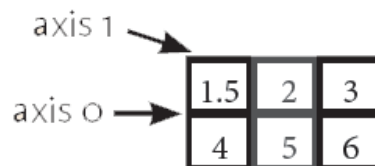
배열(Array)

- 배열: NumPy의 핵심적인 객체로 “동일한 자료형을 가지는 값들이 격자판 형태로 있는 것”
 - 파이썬의 리스트(list)와 거의 비슷하나 배열은 동일한 자료형만 들어가야 한다는 차이를 갖는다
 - 사용방법도 리스트와 유사하나 특성과 활용 가능한 함수 등이 다르다
 - 다차원(n -dimensional) 배열로 확장 가능

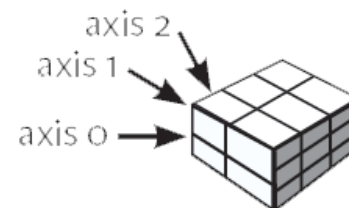
1D array



2D array



3D array



배열(Array)

- 배열의 생성

- 일반적으로 리스트를 배열로 변환하여 생성한다

```
In [2]: arr1 = np.array([1, 2, 3])  
arr1
```

```
Out [2]: array([1, 2, 3])
```

```
In [5]: print(type(arr1))  
  
<class 'numpy.ndarray'>
```

- 다차원 배열도 같은 방법으로 생성한다

```
In [4]: arr2 = np.array([[1,2,3],[4,5,6]])  
arr2
```

```
Out [4]: array([[1, 2, 3],  
               [4, 5, 6]])
```

배열(Array)

· 배열의 생성

- NumPy에 내장된 다양한 함수로 배열을 만들 수 있다

```
In [5]: arr1 = np.zeros((2,2)) # 모든 값이 0인 배열 생성  
arr1
```

```
Out [5]: array([[ 0.,  0.],  
               [ 0.,  0.]])
```

```
In [4]: arr2 = np.ones((1,2)) # 모든 값이 1인 배열 생성  
arr2
```

```
Out [4]: array([[ 1.,  1.]])
```

```
In [6]: arr3 = np.full((1,2), 10) # 모든 값이 특정 상수의 배열 생성  
arr3
```

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\numeric.py:301: FutureWarning: i  
n the future, full((1, 2), 10) will return an array of dtype('int32')  
format(shape, fill_value, array(fill_value).dtype), FutureWarning)
```

```
Out [6]: array([[ 10.,  10.]])
```

```
In [8]: arr4 = np.eye(3) # 단위행렬 생성  
arr4
```

```
Out [8]: array([[ 1.,  0.,  0.],  
               [ 0.,  1.,  0.],  
               [ 0.,  0.,  1.]])
```

```
In [9]: arr5 = np.random.random((2,2)) # 랜덤한 값으로 채워진 배열 생성  
arr5
```

```
Out [9]: array([[ 0.43185899,  0.10813613],  
               [ 0.85616989,  0.78237002]])
```

배열(Array)

- 배열의 rank와 shape

- rank(ndim): 배열이 몇 차원인지를 의미한다

```
In [8]: arr1 = np.array([1, 2, 3])
        print(arr1.ndim)
        arr2 = np.array([[1,2,3], [4,5,6]])
        print(arr2.ndim)
        arr3 = np.array([[[1,2,3], [4,5,6]], [[7,8,9], [10,11,12]]])
        print(arr3.ndim)

1
2
3
```

- shape: 각 차원의 크기를 알려주는 튜플

```
In [9]: arr1 = np.array([1, 2, 3])
        print(arr1.shape)
        arr2 = np.array([[1,2,3], [4,5,6]])
        print(arr2.shape)
        arr3 = np.array([[[1,2,3], [4,5,6]], [[7,8,9], [10,11,12]]])
        print(arr3.shape)

(3,)
(2, 3)
(2, 2, 3)
```

배열(Array)

· 배열 인덱싱

- Python의 리스트와 같은 방법으로([]) 리스트를 slicing할 수 있다

```
In [10]: arr1 = np.array([1, 2, 3])  
arr1[0]
```

```
Out[10]: 1
```

```
In [11]: arr2 = np.array([[1,2,3],[4,5,6]])  
arr2[1]
```

```
Out[11]: array([4, 5, 6])
```

```
In [12]: arr3 = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]])  
arr3[0][1]
```

```
Out[12]: array([4, 5, 6])
```

```
In [13]: arr3[0,0,1]
```

```
Out[13]: 2
```

```
In [14]: arr4 = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])  
arr5 = arr4[:,2, :]  
arr5
```

```
Out[14]: array([[1, 2, 3, 4],  
                [5, 6, 7, 8]])
```


배열(Array)

· 배열 인덱싱

- 불리언 배열 인덱싱을 통해 특정 조건을 만족하는 요소는 True로, 만족하지 않는 요소는 False로 반환할 수 있다
- 그리고 이를 통해 특정 조건을 만족하는 원소들만 취사선택해 배열로 저장할 수 있다

```
In [15]: arr1 = np.array([[1,2], [3, 4], [5, 6]])  
         idx = (arr1 > 3)  
         idx
```

```
Out [15]: array([[False, False],  
                [False,  True],  
                [ True,  True]], dtype=bool)
```

```
In [16]: arr1[idx]
```

```
Out [16]: array([4, 5, 6])
```

배열(Array)

- 배열 자료형

- 흔히 쓰이는 배열의 자료형은 int64(정수형)와 float64(실수형)이다

- 배열을 생성할 때 명시적으로 특정 자료형을 지정할 수 있다

```
In [18]: arr1 = np.array([1])  
arr1.dtype
```

```
Out [18]: dtype('int32')
```

```
In [19]: arr2 = np.array([1.0])  
arr2.dtype
```

```
Out [19]: dtype('float64')
```

```
In [20]: arr3 = np.array([1,2], dtype = np.float64)  
arr3.dtype
```

```
Out [20]: dtype('float64')
```

배열(Array)

· 배열 연산

- 기초연산(사칙연산)

```
In [21]: arr1 = np.array([[1,2], [3,4]], dtype = np.float64)
arr2 = np.array([[5,6], [7,8]], dtype = np.float64)
```

```
In [22]: arr1 + arr2
```

```
Out [22]: array([[ 6.,  8.],
                [10., 12.]])
```

```
In [23]: arr1 - arr2
```

```
Out [23]: array([[ -4., -4.],
                [-4., -4.]])
```

```
In [24]: # 행렬의 곱연산이 아님
arr1 * arr2
```

```
Out [24]: array([[ 5., 12.],
                [21., 32.]])
```

```
In [25]: arr1 / arr2
```

```
Out [25]: array([[ 0.2      ,  0.33333333],
                [ 0.42857143,  0.5      ]])
```

```
In [26]: np.sqrt(arr1)
```

```
Out [26]: array([[ 1.      ,  1.41421356],
                [ 1.73205081,  2.      ]])
```

배열(Array)

- 배열 연산

- 내적 및 행렬의 곱연산

```
In [27]: # 벡터의 내적
vector1 = np.array([1,2])
vector2 = np.array([3,4])
np.dot(vector1, vector2)
```

```
Out [27]: 11
```

```
In [28]: # 행렬의 곱연산
arr1 = np.array([[1,2], [3,4]], dtype = np.float64)
arr2 = np.array([[5,6], [7,8]], dtype = np.float64)
np.dot(arr1, arr2)
```

```
Out [28]: array([[ 19.,  22.],
                 [ 43.,  50.]])
```

배열(Array)

- 배열 연산

- sum 연산

```
In [29]: arr1 = np.array([[1,2], [3,4]], dtype = np.float64)  
         np.sum(arr1)
```

```
Out[29]: 10.0
```

```
In [31]: np.sum(arr1, axis = 0)
```

```
Out[31]: array([ 4.,  6.])
```

```
In [32]: np.sum(arr1, axis = 1)
```

```
Out[32]: array([ 3.,  7.])
```

배열(Array)

- 배열 연산

- 전치(행렬) 연산

전치행렬

```
In [33]: arr1 = np.array([[1,2], [3,4]], dtype = np.float64)
arr1
```

```
Out [33]: array([[ 1.,  2.],
                [ 3.,  4.]])
```

```
In [34]: arr1.T # 행과 열을 바꿈
```

```
Out [34]: array([[ 1.,  3.],
                [ 2.,  4.]])
```

데이터 불러오기

· 수치형 데이터 불러오기

- loadtxt() 함수를 이용해 수치형 데이터(정수 혹은 실수형)를 불러올 수 있다
- 결과로 NumPy 배열이 생성된다

```
In [2]: data = np.loadtxt('test-data.csv', delimiter = ',')
```

```
In [3]: data
```

```
Out[3]: array([[ 73.,  80.,  75., 152.],
 [ 93.,  88.,  93., 185.],
 [ 89.,  91.,  90., 180.],
 [ 96.,  98., 100., 196.],
 [ 73.,  66.,  70., 142.],
 [ 53.,  46.,  55., 101.],
 [ 69.,  74.,  77., 149.],
 [ 47.,  56.,  60., 115.],
 [ 87.,  79.,  90., 175.],
 [ 79.,  70.,  88., 164.],
 [ 69.,  70.,  73., 141.],
 [ 70.,  65.,  74., 141.],
 [ 93.,  95.,  91., 184.],
 [ 79.,  80.,  73., 152.],
 [ 70.,  73.,  78., 148.],
 [ 93.,  89.,  96., 192.],
 [ 78.,  75.,  68., 147.],
 [ 81.,  90.,  93., 183.],
 [ 88.,  92.,  86., 177.],
 [ 78.,  83.,  77., 159.],
 [ 82.,  86.,  90., 177.],
 [ 86.,  82.,  89., 175.],
 [ 78.,  83.,  85., 175.],
 [ 76.,  83.,  71., 149.],
 [ 96.,  93.,  95., 192.]])
```

데이터 불러오기

· 수치형 및 문자열 데이터 불러오기

- 문자열이 포함된 데이터를 불러오거나 결측치(missing value)가 있을 경우 `genfromtxt()` 함수를 활용한다
- 자료형(data type)으로는 `None`를 설정한다

```
In [8]: data = np.genfromtxt('building.csv', dtype = None, delimiter = ',')
```

```
In [9]: data
```

```
Out[9]: array([[b'building', b'city', b'height'],  
               [b'Empire State Building', b'New York', b'381'],  
               [b'Eiffel Tower', b'Paris', b'300'],  
               [b'63 building', b'Seoul', b'250']],  
            dtype='<S21')
```


실습 3-2-1. 랜덤 벡터 생성과 정렬

· 길이 10인 랜덤 벡터(1차원 배열)을 생성하고 이를 정렬한다

- 오름차순과 내림차순으로 한 번씩 정렬해 본다

- 수행 예시

```
In [7]: arr # 오름차순 정렬
```

```
Out [7]: array([ 0.030492 ,  0.12043689,  0.32015067,  0.33430422,  0.72236358,  
                0.83430805,  0.88724013,  0.8975324 ,  0.91277164,  0.98039898])
```

```
In [9]: arr # 내림차순 정렬
```

```
Out [9]: array([ 0.98039898,  0.91277164,  0.8975324 ,  0.88724013,  0.83430805,  
                0.72236358,  0.33430422,  0.32015067,  0.12043689,  0.030492  ])
```

실습 3-2-2. 벡터를 행렬로 변환하기

- 길이 9의 벡터(1차원 배열)를 생성하고 이를 3X3 크기의 행렬로 변환한다
 - NumPy의 `arange()` 함수를 이용해 0부터 8까지의 정수를 담은 1차원 배열을 생성한다
 - 이를 `reshape()` 함수로 2차원 배열로 변환한다
 - 수행 예시

```
In [17]: arr
```

```
Out[17]: array([[0, 1, 2],  
               [3, 4, 5],  
               [6, 7, 8]])
```

실습 3-2-3. 대각행렬(diagonal matrix)

- 실습 3-2-2에서 생성한 행렬을 대각행렬로 변환해 본다
 - 우선 행렬의 주대각 원소(대각선상에 있는 원소)만 1차원 배열로 출력해 본다
 - 3X3 대각행렬(정사각행렬의 주대각 성분 이외의 모든 성분이 0인 행렬)로 변환해 본다
 - 수행 예시

```
In [21]: diag
```

```
Out[21]: array([0, 4, 8])
```

```
In [23]: diag_mat
```

```
Out[23]: array([[0, 0, 0],  
               [0, 4, 0],  
               [0, 0, 8]])
```

실습 3-2-4. 행렬 정규화(normalization)

· 정수로 이루어진 5X5 행렬을 만들고 이를 정규화해본다

- randint() 함수를 이용해 1부터 100 사이의 정수로 이루어진 행렬을 생성한다

- 행렬 내의 값들을 정규화한다

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

- 수행 예시

```
In [33]: arr
```

```
Out [33]: array([[ 2, 37,  7, 27, 14],
 [83, 64, 96, 95,  4],
 [26, 12, 44, 78, 33],
 [95, 92, 69, 90, 85],
 [34, 49, 10, 96, 82]])
```

```
In [37]: arr_normalized
```

```
Out [37]: array([[ 0.          ,  0.37234043,  0.05319149,  0.26595745,  0.12765957],
 [ 0.86170213,  0.65957447,  1.          ,  0.9893617 ,  0.0212766 ],
 [ 0.25531915,  0.10638298,  0.44680851,  0.80851064,  0.32978723],
 [ 0.9893617 ,  0.95744681,  0.71276596,  0.93617021,  0.88297872],
 [ 0.34042553,  0.5          ,  0.08510638,  1.          ,  0.85106383]])
```

실습 3-2-5. 행렬 곱(matrix multiplication)

· 아래의 행렬 곱 연산을 수행한다

- np.dot() 함수를 활용한다

$$A * B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

- 수행 예시

```
In [44]: result
```

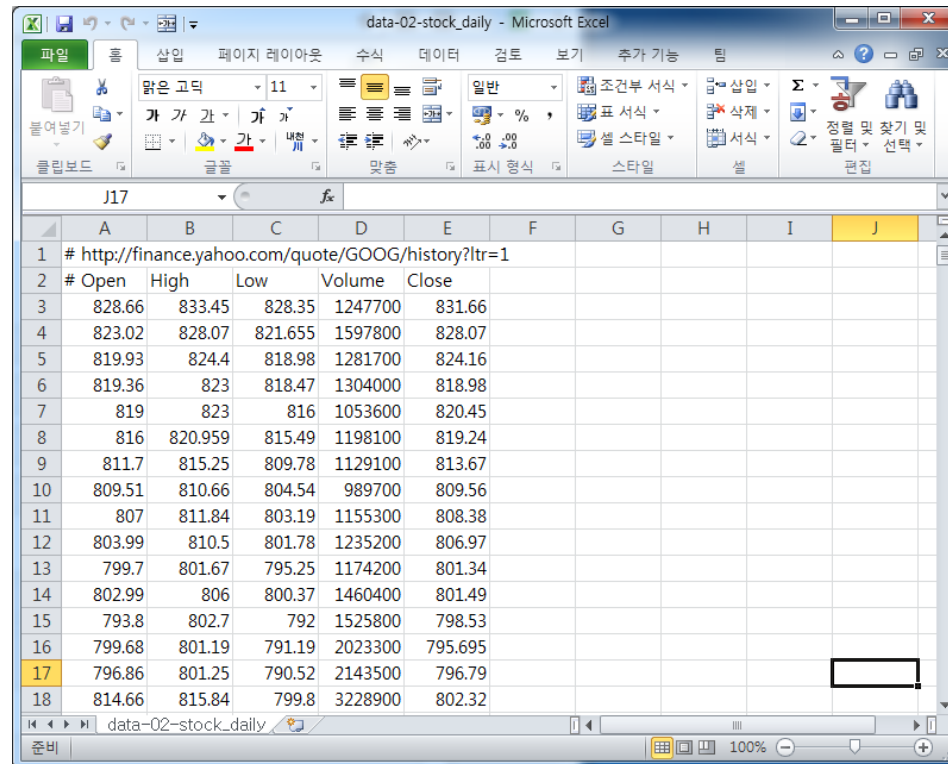
```
Out[44]: array([[22, 28],  
               [49, 64]])
```

실습 3-2-6. 주식 데이터 불러오기

· stock-data.csv의 데이터를 불러온다

- 1열부터 4열('Open', 'High', 'Low', 'Volume')까지의 데이터는 x_data로 저장하고 5열('Close')의 데이터는 y_data로 저장한다

- 즉, x_data의 shape은 (732,4), y_data의 shape은 (732,) 혹은 (732,1)이 되어야 한다



	A	B	C	D	E	F	G	H	I	J
1	#	http://finance.yahoo.com/quote/GOOG/history?ltr=1								
2	#	Open	High	Low	Volume	Close				
3		828.66	833.45	828.35	1247700	831.66				
4		823.02	828.07	821.655	1597800	828.07				
5		819.93	824.4	818.98	1281700	824.16				
6		819.36	823	818.47	1304000	818.98				
7		819	823	816	1053600	820.45				
8		816	820.959	815.49	1198100	819.24				
9		811.7	815.25	809.78	1129100	813.67				
10		809.51	810.66	804.54	989700	809.56				
11		807	811.84	803.19	1155300	808.38				
12		803.99	810.5	801.78	1235200	806.97				
13		799.7	801.67	795.25	1174200	801.34				
14		802.99	806	800.37	1460400	801.49				
15		793.8	802.7	792	1525800	798.53				
16		799.68	801.19	791.19	2023300	795.695				
17		796.86	801.25	790.52	2143500	796.79				
18		814.66	815.84	799.8	3228900	802.32				

실습 3-2-6. 주식 데이터 불러오기

· stock-data.csv의 데이터를 불러온다

- 수행 예시

```
In [8]: x_data.shape
```

```
Out [8]: (732, 4)
```

```
In [9]: x_data
```

```
Out [9]: array([[ 8.28659973e+02,  8.33450012e+02,  8.28349976e+02,
                  1.24770000e+06],
                [ 8.23020020e+02,  8.28070007e+02,  8.21655029e+02,
                  1.59780000e+06],
                [ 8.19929993e+02,  8.24400024e+02,  8.18979980e+02,
                  1.28170000e+06],
                ...,
                [ 5.66892592e+02,  5.67002574e+02,  5.56932537e+02,
                  1.08000000e+04],
                [ 5.61202549e+02,  5.66432590e+02,  5.58672539e+02,
                  4.12000000e+04],
                [ 5.68002570e+02,  5.68002570e+02,  5.52922516e+02,
                  1.31000000e+04]])
```

```
In [10]: y_data.shape
```

```
Out [10]: (732,)
```

```
In [11]: y_data
```

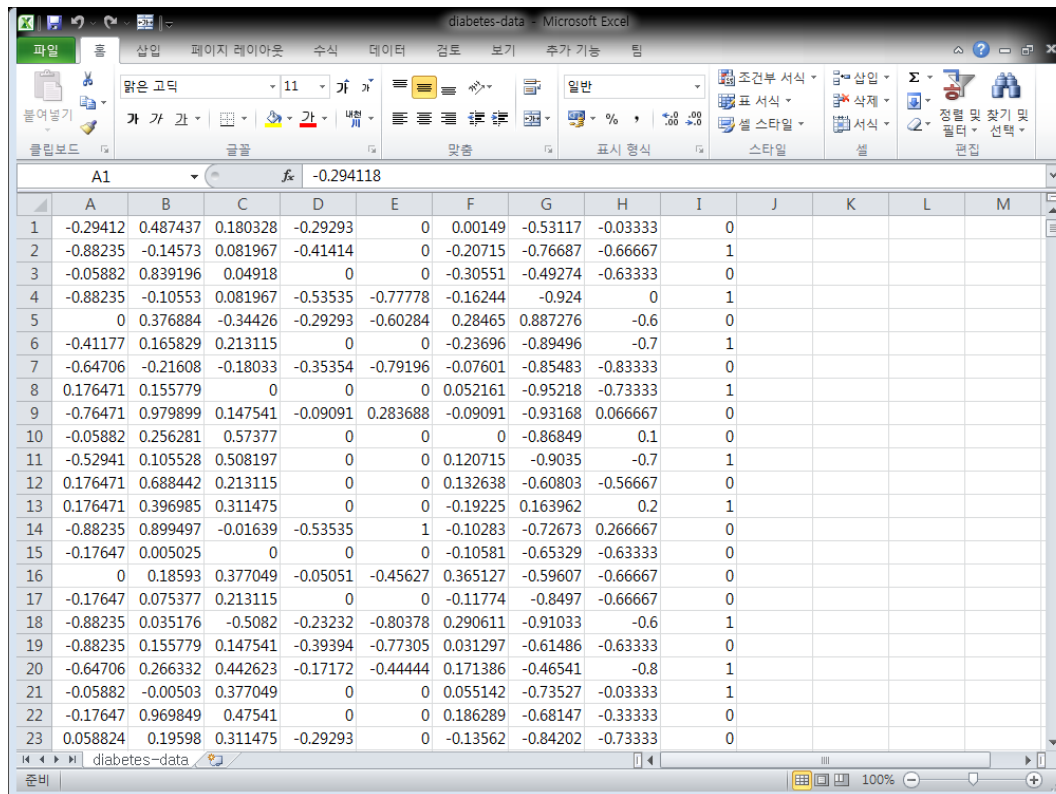
```
Out [11]: array([ 831.659973,  828.070007,  824.159973,  818.97998 ,  820.450012,
                  819.23999 ,  813.669983,  809.559998,  808.380005,  806.969971,
                  801.340027,  801.48999 ,  798.530029,  795.695007,  796.789978,
                  802.320007,  823.309998,  832.150024,  835.669983,  823.869995,
                  819.309998,  805.02002 ,  802.174988,  806.070007,  804.609985,
                  807.880005,  806.359985,  807.909973,  804.789978,  806.650024,
                  806.150024,  794.02002 ,  786.900024,  786.140015,  771.820007,
                  782.789978,  785.049988,  791.549988,  789.909973,  791.26001 ,
                  794.559998,  796.419983,  794.200012,  790.799988,  797.849976,
                  797.070007,  796.099976,  789.27002 ,  789.289978,  776.419983,
                  771.190002,  759.109985,  762.52002 ,  750.5 ,  747.919983,
                  758.039978,  770.840027,  768.23999 ,  761.679993,  760.98999 ,
                  768.27002 ,  769.200012,  760.539978,  771.22998 ,  764.47998 ,
                  758.48999 ,  736.080017,  754.02002 ,  762.559998,  785.309998,
                  790.51001 ,  782.52002 ,  762.02002 ,  762.130005,  768.700012,
                  783.609985,  784.539978,  795.369995,  795.349976,  799.070007,
                  807.669983,  813.109985,  799.369995,  796.969971,  801.5 ,
                  795.26001 ,  779.960022,  778.530029,  778.190002,  786.140015,
                  783.070007,  785.940002,  775.080017,  776.859985,  776.469971,
                  776.420002,  772.550002,  772.200029,  775.01001 ,  791.550002])
```

실습 3-2-7. 당뇨 데이터 불러오기

· diabetes-data.csv의 데이터를 불러온다

- 1열부터 8열까지의 데이터는 x_data로 저장하고 9열의 데이터는 y_data로 저장한다

- 즉, x_data의 shape은 (759, 8), y_data의 shape은 (759,) 혹은 (759,1)이 되어야 한다



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	-0.29412	0.487437	0.180328	-0.29293	0	0.00149	-0.53117	-0.03333	0				
2	-0.88235	-0.14573	0.081967	-0.41414	0	-0.20715	-0.76687	-0.66667	1				
3	-0.05882	0.839196	0.04918	0	0	-0.30551	-0.49274	-0.63333	0				
4	-0.88235	-0.10553	0.081967	-0.53535	-0.77778	-0.16244	-0.924	0	1				
5	0	0.376884	-0.34426	-0.29293	-0.60284	0.28465	0.887276	-0.6	0				
6	-0.41177	0.165829	0.213115	0	0	-0.23696	-0.89496	-0.7	1				
7	-0.64706	-0.21608	-0.18033	-0.35354	-0.79196	-0.07601	-0.85483	-0.83333	0				
8	0.176471	0.155779	0	0	0	0.052161	-0.95218	-0.73333	1				
9	-0.76471	0.979899	0.147541	-0.09091	0.283688	-0.09091	-0.93168	0.066667	0				
10	-0.05882	0.256281	0.57377	0	0	0	-0.86849	0.1	0				
11	-0.52941	0.105528	0.508197	0	0	0.120715	-0.9035	-0.7	1				
12	0.176471	0.688442	0.213115	0	0	0.132638	-0.60803	-0.56667	0				
13	0.176471	0.396985	0.311475	0	0	-0.19225	0.163962	0.2	1				
14	-0.88235	0.899497	-0.01639	-0.53535	1	-0.10283	-0.72673	0.266667	0				
15	-0.17647	0.005025	0	0	0	-0.10581	-0.65329	-0.63333	0				
16	0	0.18593	0.377049	-0.05051	-0.45627	0.365127	-0.59607	-0.66667	0				
17	-0.17647	0.075377	0.213115	0	0	-0.11774	-0.8497	-0.66667	0				
18	-0.88235	0.035176	-0.5082	-0.23232	-0.80378	0.290611	-0.91033	-0.6	1				
19	-0.88235	0.155779	0.147541	-0.39394	-0.77305	0.031297	-0.61486	-0.63333	0				
20	-0.64706	0.266332	0.442623	-0.17172	-0.44444	0.171386	-0.46541	-0.8	1				
21	-0.05882	-0.00503	0.377049	0	0	0.055142	-0.73527	-0.03333	1				
22	-0.17647	0.969849	0.47541	0	0	0.186289	-0.68147	-0.33333	0				
23	0.058824	0.19598	0.311475	-0.29293	0	-0.13562	-0.84202	-0.73333	0				

실습 3-2-7. 당뇨 데이터 불러오기

· diabetes-data.csv의 데이터를 불러온다

- 수행 예시

```
In [14]: x_data.shape
```

```
Out[14]: (759, 8)
```

```
In [15]: x_data
```

```
Out[15]: array([[ -0.294118 ,  0.487437 ,  0.180328 , ...,  0.00149028,
                  -0.53117  , -0.0333333 ],
                 [-0.882353 , -0.145729 ,  0.0819672 , ..., -0.207153 ,
                  -0.766866 , -0.666667 ],
                 [-0.0588235 ,  0.839196 ,  0.0491803 , ..., -0.305514 ,
                  -0.492741 , -0.633333 ],
                 ...,
                 [-0.411765 ,  0.21608  ,  0.180328 , ..., -0.219076 ,
                  -0.857387 , -0.7    ],
                 [-0.882353 ,  0.266332 , -0.0163934 , ..., -0.102832 ,
                  -0.768574 , -0.133333 ],
                 [-0.882353 , -0.0653266 ,  0.147541 , ..., -0.0938897 ,
                  -0.797609 , -0.933333 ]])
```

```
In [16]: y_data.shape
```

```
Out[16]: (759,)
```

```
In [17]: y_data
```

```
Out[17]: array([[ 0.,  1.,  0.,  1.,  0.,  1.,  0.,  1.,  0.,  0.,  1.,  0.,  1.,
                  0.,  0.,  0.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  1.,
                  1.,  1.,  1.,  0.,  0.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,  1.,
                  1.,  1.,  0.,  1.,  0.,  1.,  1.,  0.,  1.,  1.,  1.,  1.,  0.,
                  1.,  1.,  1.,  0.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,  1.,
                  1.,  1.,  1.,  1.,  0.,  1.,  1.,  1.,  0.,  1.,  1.,  1.,
                  0.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,  1.,  1.,  1.,  1.,  1.,
                  1.,  1.,  1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  1.,  1.,
                  1.,  0.,  1.,  1.,  1.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,
                  0.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,  1.,
                  1.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,  1.,  0.,  0.,  1.,  1.,
                  1.,  0.,  1.,  1.,  1.,  0.,  1.,  0.,  1.,  1.,  1.,  1.,
                  1.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  1.,  0.,  1.,
                  0.,  0.,  0.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,  1.,  0.,
                  1.,  1.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  1.,
```

Pandas



Pandas란?

- 데이터 전처리와 준비를 효과적으로 할 수 있도록 도와주는 Python 라이브러리
 - 엑셀의 시트와 같은 '데이터프레임(Dataframe)'을 활용하여 데이터를 쉽게 핸들링할 수 있다
 - Pandas 불러오기

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

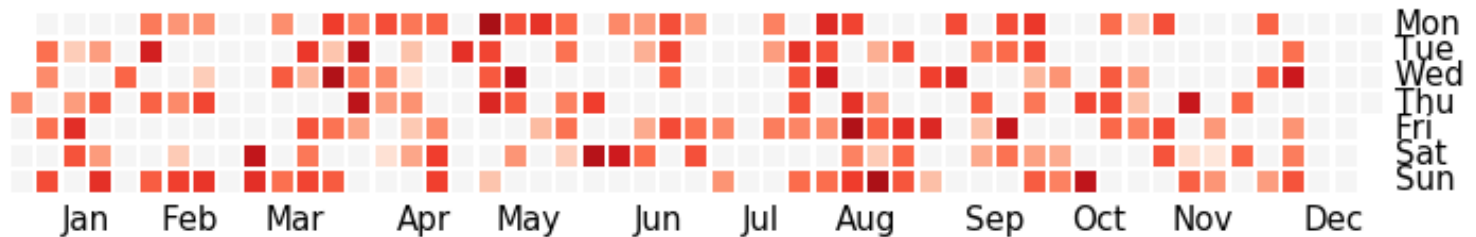
	A	B	C	D
1				
2	Cookie Sales by Region			
3	SalesRep	Region	# Orders	Total Sales
4	Bryan	West	217	\$35,000
5	Hilary	West	268	\$24,547
6	Henry	North	224	\$21,253
7	Jane	North	286	\$12,485
8	Joseph	South	226	\$22,463
9	Mary	East	228	\$32,368
0	Molly	West	234	\$57,639
1	Raymond	East	267	\$42,463
2	Sussy	East	279	\$53,467
3	Thomas	South	261	\$63,456
4				

```
In [3]: DataFrame(data, columns = ['year', 'state', 'pop'])
```

Out [3]:		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

시리즈(Series)

- Pandas의 시리즈(Series)는 1차원 리스트, 혹은 NumPy의 1차원 배열과 유사하다
 - 엑셀의 시트와 같은 '데이터프레임(Dataframe)'을 활용하여 데이터를 쉽게 핸들링할 수 있다
 - NumPy 배열과는 다르게 서로 다른 자료형도 시리즈 내에 담을 수 있다



시리즈(Series)

- 시리즈 생성하기

- 일반적으로 파이썬 리스트를 시리즈로 변환하여 생성한다

```
In [3]: s1 = pd.Series([1, 4, 7, 8, 9, 10, 'a'])  
s1
```

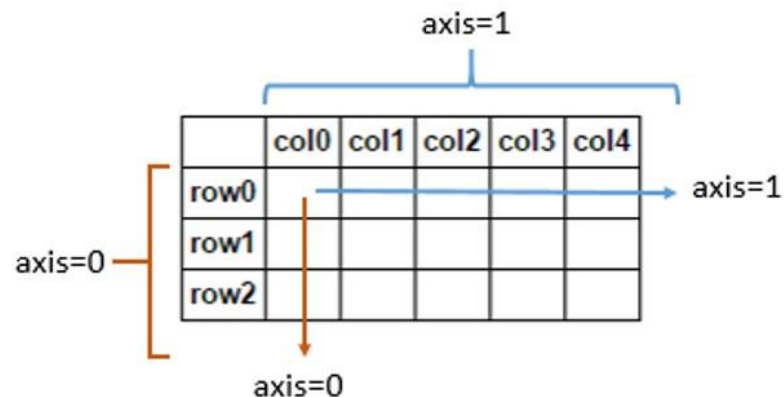
```
Out[3]: 0    1  
        1    4  
        2    7  
        3    8  
        4    9  
        5   10  
        6    a  
        dtype: object
```

```
In [6]: s2 = pd.Series([1, 2, 3, np.nan, 4, 5])  
s2
```

```
Out[6]: 0    1.0  
        1    2.0  
        2    3.0  
        3    NaN  
        4    4.0  
        5    5.0  
        dtype: float64
```

데이터프레임(Dataframe)

- Pandas의 데이터프레임(Dataframe)은 2차원 리스트 혹은 배열과 유사한 자료구조이다
 - 행렬(matrix)과 같이 행(row)과 열(column)로 이루어져 있으며, 이를 인덱스를 이용해 접근할 수 있다
 - 두 개의 축(axis)가 있어 행과 열을 관리할 수 있다
 - 데이터프레임에는 시리즈와 같이 서로 다른 자료형을 담을 수 있다

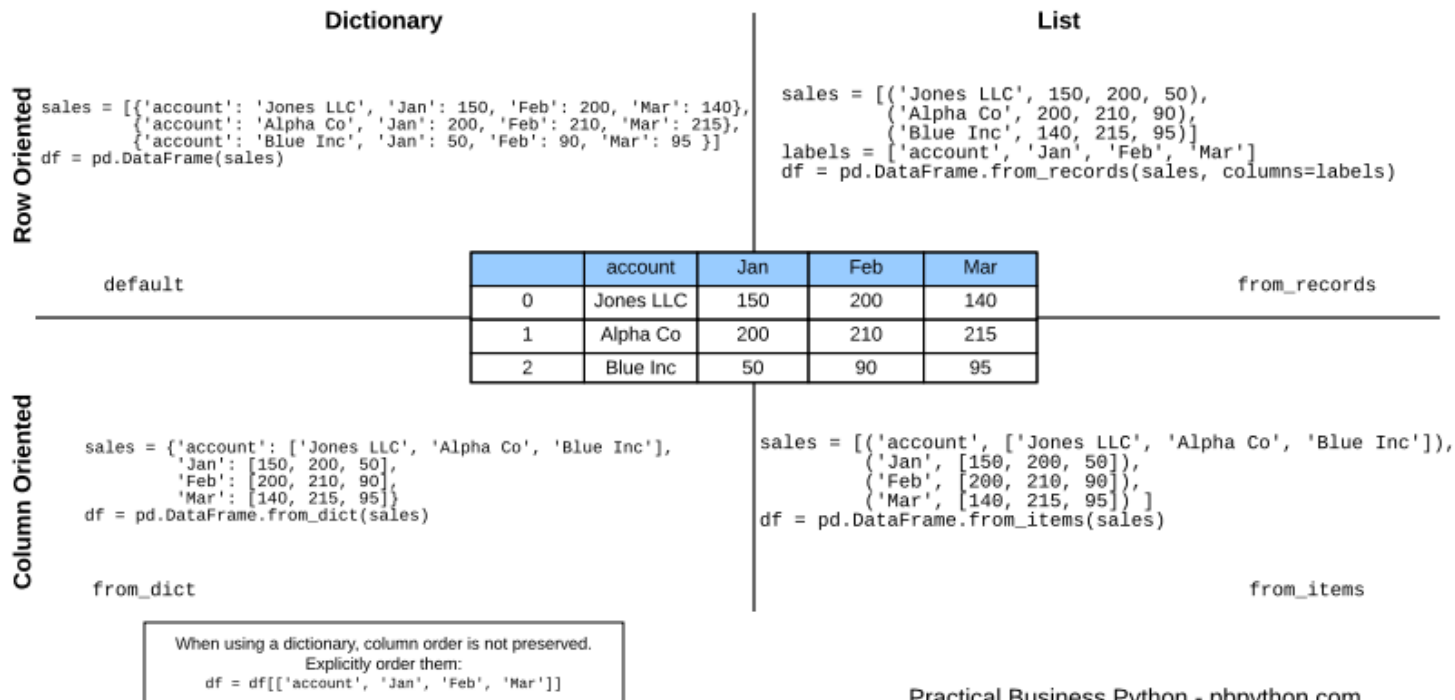


데이터프레임(Dataframe)

· 데이터프레임 생성하기

- 데이터프레임은 리스트, 배열 혹은 딕셔너리로 생성할 수 있다

Creating Pandas DataFrames from Python Lists and Dictionaries



데이터프레임(Dataframe)

- 데이터프레임 생성하기

- 배열 혹은 리스트로 데이터프레임 생성하기

```
In [8]: df1 = pd.DataFrame([[1,2,3],[4,5,6]])  
df1
```

```
Out[8]:
```

	0	1	2
0	1	2	3
1	4	5	6

```
In [12]: df2 = pd.DataFrame(np.zeros((2,3)))  
df2
```

```
Out[12]:
```

	0	1	2
0	0.0	0.0	0.0
1	0.0	0.0	0.0

■ 생성하면서 축(axis)을 잘 고려한다

데이터프레임(Dataframe)

- 데이터프레임 생성하기

- 딕셔너리로 데이터프레임 생성하기

```
In [14]: names = ['John', 'Tim', 'Bruce']  
grades = ['A+', 'B-', 'F']
```

```
In [15]: df3 = pd.DataFrame({'name': names, 'grade': grades})  
df3
```

```
Out [15]:
```

	grade	name
0	A+	John
1	B-	Tim
2	F	Bruce

■ 생성하면서 축(axis)을 잘 고려한다

데이터프레임(Dataframe)

- 데이터프레임 생성하기

- 컬럼 이름과 인덱스를 인자값으로 설정할 수 있다

```
In [17]: df4 = pd.DataFrame(np.random.randn(10, 3), index = range(1, 11), columns = list('abc'))  
df4
```

Out [17]:

	a	b	c
1	-0.865203	-0.041992	0.426080
2	-0.430009	0.070265	-0.499093
3	2.522299	0.632737	0.903667
4	0.075403	1.592519	1.263607
5	0.774317	-0.783115	0.229059
6	-0.264901	-0.385646	1.540035
7	0.440429	-1.067525	-0.458921
8	0.930104	0.551395	-0.557046
9	0.959832	1.307272	0.134029
10	0.302500	-1.073217	0.946988

■ 따로 설정하지 않을 경우 디폴트(0부터 시작하는 정수)로 설정된다

데이터프레임(Dataframe)

- 데이터 열람하기

- head()와 tail() 함수로 데이터의 일부를 열람할 수 있다

- 인자값을 주지 않을 경우 상위, 혹은 하위 5개 데이터만 보여준다

```
In [21]: df4.head()
```

```
Out [21]:
```

	a	b	c
1	0.298188	0.199613	-0.866343
2	-1.082578	1.835957	0.292041
3	-0.617539	0.817866	0.370413
4	0.514701	-0.601944	0.910449
5	0.831008	-0.241993	0.746269

```
In [22]: df4.tail(3)
```

```
Out [22]:
```

	a	b	c
8	-2.489759	0.221283	0.557908
9	-0.443511	0.913075	0.931968
10	0.724700	1.848305	1.972213

데이터프레임(Dataframe)

- 데이터 열람하기

- 인덱스(행)과 컬럼(열)의 데이터만 열람할 수 있다

```
In [23]: df4.index
```

```
Out [23]: RangeIndex(start=1, stop=11, step=1)
```

```
In [24]: df4.columns
```

```
Out [24]: Index(['a', 'b', 'c'], dtype='object')
```

- 데이터프레임 내의 값들만 추출해 볼 수 있다

```
In [25]: df4.values
```

```
Out [25]: array([[ 0.29818847,  0.19961256, -0.86634334],
                 [-1.08257833,  1.83595738,  0.29204105],
                 [-0.61753931,  0.81786649,  0.37041304],
                 [ 0.51470107, -0.6019438 ,  0.91044903],
                 [ 0.83100765, -0.24199324,  0.74626873],
                 [-1.24664972,  0.44143236, -0.58160246],
                 [-1.63216173,  0.41169074, -1.02701095],
                 [-2.4897595 ,  0.22128316,  0.55790792],
                 [-0.44351133,  0.91307501,  0.93196769],
                 [ 0.72470046,  1.84830507,  1.97221313]])
```

데이터프레임(Dataframe)

- 데이터 열람하기

- describe() 함수로 데이터의 요약(수치형인 경우 기초 통계량)을 열람할 수 있다

In [26]: `df4.describe()`

Out [26]:

	a	b	c
count	10.000000	10.000000	10.000000
mean	-0.514360	0.584529	0.330630
std	1.110103	0.799090	0.926653
min	-2.489759	-0.601944	-1.027011
25%	-1.205632	0.205030	-0.363192
50%	-0.530525	0.426562	0.464160
75%	0.460573	0.889273	0.869404
max	0.831008	1.848305	1.972213

데이터프레임(Dataframe)

· 데이터 변경하기

- 전치(transpose): 데이터프레임의 열과 행을 바꾼다

In [28]:

df4

Out [28]:

	a	b	c
1	0.298188	0.199613	-0.866343
2	-1.082578	1.835957	0.292041
3	-0.617539	0.817866	0.370413
4	0.514701	-0.601944	0.910449
5	0.831008	-0.241993	0.746269
6	-1.246650	0.441432	-0.581602
7	-1.632162	0.411691	-1.027011
8	-2.489759	0.221283	0.557908
9	-0.443511	0.913075	0.931968
10	0.724700	1.848305	1.972213

In [27]:

df4.T

Out [27]:

	1	2	3	4	5	6	7	8	9	10
a	0.298188	-1.082578	-0.617539	0.514701	0.831008	-1.246650	-1.632162	-2.489759	-0.443511	0.724700
b	0.199613	1.835957	0.817866	-0.601944	-0.241993	0.441432	0.411691	0.221283	0.913075	1.848305
c	-0.866343	0.292041	0.370413	0.910449	0.746269	-0.581602	-1.027011	0.557908	0.931968	1.972213

데이터프레임(Dataframe)

- 데이터 변경하기

- 정렬(sorting): 특정 컬럼의 값을 기준으로 정렬하기

```
In [29]: df4.sort_values(by = 'b')
```

```
Out [29]:
```

	a	b	c
4	0.514701	-0.601944	0.910449
5	0.831008	-0.241993	0.746269
1	0.298188	0.199613	-0.866343
8	-2.489759	0.221283	0.557908
7	-1.632162	0.411691	-1.027011
6	-1.246650	0.441432	-0.581602
3	-0.617539	0.817866	0.370413
9	-0.443511	0.913075	0.931968
2	-1.082578	1.835957	0.292041
10	0.724700	1.848305	1.972213

데이터프레임(Dataframe)

· 데이터 변경하기

- 삽입(insertion): 새로운 컬럼을 삽입할 수 있다

```
In [30]: df4['d'] = np.random.random(10)
```

```
In [31]: df4
```

```
Out [31]:
```

	a	b	c	d
1	0.298188	0.199613	-0.866343	0.632383
2	-1.082578	1.835957	0.292041	0.676948
3	-0.617539	0.817866	0.370413	0.000727
4	0.514701	-0.601944	0.910449	0.463687
5	0.831008	-0.241993	0.746269	0.423285
6	-1.246650	0.441432	-0.581602	0.081380
7	-1.632162	0.411691	-1.027011	0.181873
8	-2.489759	0.221283	0.557908	0.819547
9	-0.443511	0.913075	0.931968	0.273757
10	0.724700	1.848305	1.972213	0.558798

데이터프레임(Dataframe)

- 데이터 선택하기

- 컬럼의 이름을 넣어 특정 컬럼의 데이터를 선택할 수 있다

■ 시리즈가 결과로 나옴

```
In [33]: a = df4['a']  
a
```

```
Out [33]: 1    0.298188  
2   -1.082578  
3   -0.617539  
4    0.514701  
5    0.831008  
6   -1.246650  
7   -1.632162  
8   -2.489759  
9   -0.443511  
10   0.724700  
Name: a, dtype: float64
```

```
In [34]: type(a)
```

```
Out [34]: pandas.core.series.Series
```

데이터프레임(Dataframe)

- 데이터 선택하기

- 슬라이싱을 통해 행을 기준으로 데이터를 자를 수 있다

```
In [36]: df4[0:5]
```

```
Out [36]:
```

	a	b	c	d
1	0.298188	0.199613	-0.866343	0.632383
2	-1.082578	1.835957	0.292041	0.676948
3	-0.617539	0.817866	0.370413	0.000727
4	0.514701	-0.601944	0.910449	0.463687
5	0.831008	-0.241993	0.746269	0.423285

```
In [38]: df4[5:]
```

```
Out [38]:
```

	a	b	c	d
6	-1.246650	0.441432	-0.581602	0.081380
7	-1.632162	0.411691	-1.027011	0.181873
8	-2.489759	0.221283	0.557908	0.819547
9	-0.443511	0.913075	0.931968	0.273757
10	0.724700	1.848305	1.972213	0.558798

데이터프레임(Dataframe)

· 데이터 합치기

- concat(): 따로 축을 설정하지 않을 경우 데이터를 컬럼을 기준으로 병합한다

```
In [40]: df1 = pd.DataFrame([[1,2,3], [4,5,6]])  
df1
```

Out [40]:

	0	1	2
0	1	2	3
1	4	5	6

```
In [42]: df2 = pd.DataFrame([[7,8,9], [10,11,12]])  
df2
```

Out [42]:

	0	1	2
0	7	8	9
1	10	11	12

```
In [43]: pd.concat([df1, df2])
```

Out [43]:

	0	1	2
0	1	2	3
1	4	5	6
0	7	8	9
1	10	11	12

df1					Result				
	A	B	C	D		A	B	C	D
0	A0	B0	C0	D0	0	A0	B0	C0	D0
1	A1	B1	C1	D1	1	A1	B1	C1	D1
2	A2	B2	C2	D2	2	A2	B2	C2	D2
3	A3	B3	C3	D3	3	A3	B3	C3	D3
df2					4	A4	B4	C4	D4
	A	B	C	D	5	A5	B5	C5	D5
4	A4	B4	C4	D4	6	A6	B6	C6	D6
5	A5	B5	C5	D5	7	A7	B7	C7	D7
6	A6	B6	C6	D6	8	A8	B8	C8	D8
7	A7	B7	C7	D7	9	A9	B9	C9	D9
df3					10	A10	B10	C10	D10
	A	B	C	D	11	A11	B11	C11	D11
8	A8	B8	C8	D8					
9	A9	B9	C9	D9					
10	A10	B10	C10	D10					
11	A11	B11	C11	D11					

데이터프레임(Dataframe)

· 데이터 합치기

- concat(): 축을 1로 설정할 경우 행을 기준으로 병합한다

```
In [44]: pd.concat([df1, df2], axis = 1)
```

Out [44]:

	0	1	2	0	1	2
0	1	2	3	7	8	9
1	4	5	6	10	11	12

df1					df4				Result							
										A	B	C	D	B	D	F
0	A				2	B			0	A0	B0	C0	D0	NaN	NaN	NaN
	B					D			1	A1	B1	C1	D1	NaN	NaN	NaN
	C					F			2	A2	B2	C2	D2	B2	D2	F2
	D								3	A3	B3	C3	D3	B3	D3	F3
							6	NaN	NaN	NaN	NaN	B6	D6	F6		
							7	NaN	NaN	NaN	NaN	B7	D7	F7		

데이터프레임(Dataframe)

- 데이터 합치기

- 더욱 많은 데이터 병합 함수와 예제는 아래를 참고한다

■ LINK: <https://pandas.pydata.org/pandas-docs/stable/merging.html#merging>

데이터 불러오기

- pandas에서도 numpy와 유사하게 csv와 text 데이터를 불러올 수 있다

- read_csv() 함수를 활용한다

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

```
In [2]: df = pd.read_csv('csv_data.csv')
```

```
In [3]: df
```

Out [3]:

	ID	LAST_NAME	AGE
0	1	KIM	30
1	2	CHOI	25
2	3	LEE	41
3	4	PARK	19
4	5	LIM	36

■ read_csv() 함수를 통해 데이터를 불러올 경우 저절로 데이터프레임으로 저장된다

데이터 불러오기

- pandas에서는 excel 형태(.xlsx/.xls)의 데이터도 쉽게 불러올 수 있다

- read_excel() 함수를 활용한다

```
In [6]: df = pd.read_excel('excel_data.xlsx')  
df
```

Out [6]:

	ID	LAST_NAME	AGE
0	1	KIM	30
1	2	CHOI	25
2	3	LEE	41
3	4	PARK	19
4	5	LIM	36

■ read_excel() 함수를 통해 데이터를 불러올 경우에도 저절로 데이터프레임으로 저장된다

데이터 불러오기

- pandas를 활용하면 csv 혹은 excel 형태의 데이터로 저장하는 것도 용이하다

- to_csv() 혹은 to_excel() 함수를 활용한다

```
In [7]: df.to_csv('data.csv', sep = ',')
```

```
In [8]: df.to_excel('data.xlsx')
```


실습 3-2-8. 데이터프레임 생성하기

· 아래 내용의 데이터를 담은 데이터프레임을 생성해 본다

- 1열부터 8열까지의 데이터는 x_data로 저장하고 9열의 데이터는 y_data로 저장한다

Index	Title	Artist	Length	Year
0	Skyfall	Adele	4:46	2012
1	Lose Yourself	Eminem	5:26	2002
2	Another Brick in the Wall	Pink Floyd	3:59	1979
3	Use Somebody	Kings of Leon	3:51	2008
4	Treasure	Bruno Mars	2:58	2013

- Title, Artist, Length는 String 타입으로, Year는 int 타입으로 저장한다

실습 3-2-8. 데이터프레임 생성하기

· 아래 내용의 데이터를 담은 데이터프레임을 생성해 본다

- 수행 예시

In [3]: data

Out [3]:

	Artist	Length	Title	Year
0	Adele	4:46	Skyfall	2012
1	Eminem	5:26	Lose Yourself	2002
2	Pink Floyd	3:59	Another Brick in the Wall	1979
3	Kings of Leon	3:51	Use Somebody	2008
4	Bruno Mars	2:58	Treasure	2013

실습 3-2-9. 데이터프레임 나누기

- 실습 3-2-8에서 생성된 데이터프레임에서 'Title' 컬럼의 데이터만 추출한다
 - 추출한 데이터는 모두 소문자(lowercase)로 변환한 뒤 title_lowercase 이름의 리스트에 저장한다
 - 수행 예시

```
In [5]: title_lowercase
```

```
Out [5]: ['skyfall',  
          'lose yourself',  
          'another brick in the wall',  
          'use somebody',  
          'treasure']
```

실습 3-2-10. 엑셀 데이터 불러오기

- animals.xlsx 파일을 불러와 데이터프레임을 생성한다
 - 엑셀 파일의 첫 세 컬럼(name, hair, feathers)의 데이터만 포함하는 데이터프레임 animals를 생성한다
 - 수행 예시

```
In [11]: animals
```

```
Out [11]:
```

	name	hair	feathers
0	aardvark	1	0
1	antelope	1	0
2	bass	0	0
3	bear	1	0
4	boar	1	0
5	buffalo	1	0
6	calf	1	0
7	carp	0	0
8	catfish	0	0
9	cavy	1	0

■ 참고: .loc[] 혹은 .iloc[] 를 통해 슬라이싱한다 (refer to: <https://stackoverflow.com/questions/10665889/how-to-take-column-slices-of-dataframe-in-pandas>)

실습 3-2-11. 엑셀 파일로 저장하기

- 실습 3-2-10에서 생성된 animals 데이터프레임의 데이터를 엑셀 파일로 저장한다
 - animals_sub.xlsx 이름으로 저장한다
 - 수행 예시

animals_sub - Microsoft Excel

파일 홈 삽입 페이지 레이아웃 수식 데이터 검토 보기 추가 기능 탭

읽은 고딕 11 가 > 일반 조건부 서식 삽입 삭제 정렬 및 필터
블여넣기 가 가 과 표 서식 셀 스타일 서식 섹션 찾기 및 선택
클립보드 글꼴 맞춤 표시 형식 스타일 편집

	A	B	C	D	E	F	G	H	I	J	K	L	M
		name	hair	feathers									
1													
2	0	aardvark	1	0									
3	1	antelope	1	0									
4	2	bass	0	0									
5	3	bear	1	0									
6	4	boar	1	0									
7	5	buffalo	1	0									
8	6	calf	1	0									
9	7	carp	0	0									
10	8	catfish	0	0									
11	9	cavy	1	0									
12	10	cheetah	1	0									
13	11	chicken	0	1									
14	12	chub	0	0									
15	13	clam	0	0									
16	14	crab	0	0									
17	15	crayfish	0	0									
18	16	crow	0	1									
19	17	deer	1	0									
20	18	dogfish	0	0									
21	19	dolphin	0	0									
22	20	dove	0	1									
23	21	duck	0	1									

준비 | Sheet1

100%