NumPy

배열 데이터를 효과적으로 다루는 NumPy

- 파이썬으로 과학 연산을 쉽고 빠르게 할 수 있게 만든 패키지
- NumPy 홈페이지: http://www.numpy.org
- 아나콘다 배포판에는 NumPy가 포함돼 있음

배열 데이터를 효과적으로 다루는 NumPy

- 배열 생성하기
  - 배열(Array)이란 순서가 있는 같은 종류의 데이터가 저장된 집합
  - NumPy 임포트

```
In:import numpy as np
```

  - 시퀀스 데이터로부터 배열 생성

```
arr_obj = np.array(seq_data)
```

  - 리스트로부터 NumPy의 1차원 배열을 생성하는 예

```
In: import numpy as np
    data1 = [0, 1, 2, 3, 4, 5]
    a1 = np.array(data1)
    a1
Out: array([0, 1, 2, 3, 4, 5])
```

```
In: data2 = [0.1, 5, 4, 12, 0.5]
    a2 = np.array(data2)
    a2
Out: array([ 0.1,  5. ,  4. , 12. ,  0.5])
```

배열 데이터를 효과적으로 다루는 NumPy

– 배열 객체의 타입 확인

```
In: a1.dtype
Out: dtype('int32')
```

```
In: a2.dtype
Out: dtype('float64')
```

– 다차원 배열 생성

```
In: np.array([[1,2,3], [4,5,6], [7,8,9]])
Out: array([[1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]])
```

– 범위를 지정해 배열 생성

```
arr_obj = np.arange([start,] stop[, step])
```

```
In: np.arange(0, 10, 2)
Out: array([0, 2, 4, 6, 8])
```

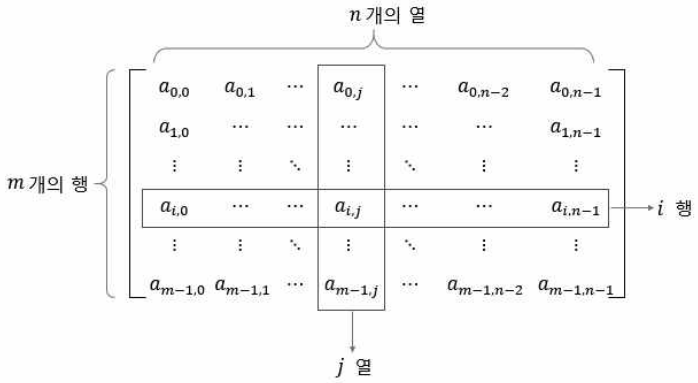
배열 데이터를 효과적으로 다루는 NumPy

- 범위를 지정해 배열 생성

```
In: np.arange(1, 10)
Out: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In: np.arange(5)
Out: array([0, 1, 2, 3, 4])
```

- m x n 행렬: .reshape(m, n) 이용




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

배열 데이터를 효과적으로 다루는 NumPy

- m x n 행렬의 형태 확인

```
In: b1 = np.arange(12).reshape(4,3)
    b1.shape
Out: (4, 3)
```

```
In: b2 = np.arange(5)
    b2.shape
Out: (5,)
```

- 범위의 시작과 끝, 데이터의 개수를 지정해 배열 생성

```
arr_obj = np.linspace(start, stop[, num])
```

```
In: np.linspace(1, 10, 10)
Out: array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

```
In: np.linspace(0, np.pi, 20 )
Out: array([0.        , 0.16534698, 0.33069396, 0.49604095, 0.66138793,
                    0.82673491, 0.99208189, 1.15742887, 1.32277585, 1.48812284,
                    1.65346982, 1.8188168 , 1.98416378, 2.14951076, 2.31485774,
                    2.48020473, 2.64555171, 2.81089869, 2.97624567, 3.14159265])
```

배열 데이터를 효과적으로 다루는 NumPy

- 특별한 형태의 배열 생성

```
arr_zero_n = np.zeros(n)
arr_zero_mxn = np.zeros((m,n))
arr_one_n = np.ones(n)
arr_one_mxn = np.ones((m,n))
```

```
In: np.zeros(10)
Out: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
In: np.zeros((3,4))
Out: array([[0., 0., 0., 0.],
            [0., 0., 0., 0.],
            [0., 0., 0., 0.]])
```

```
In: np.ones(5)
Out: array([ 1.,  1.,  1.,  1.,  1.])
```

```
In: np.ones((3,5))
Out: array([[ 1.,  1.,  1.,  1.,  1.],
            [ 1.,  1.,  1.,  1.,  1.],
            [ 1.,  1.,  1.,  1.,  1.]])
```

배열 데이터를 효과적으로 다루는 NumPy

- 단위 행렬 생성

```
arr_I = np.eye(n)
```

```
In: np.eye(3)
Out: array([[1., 0., 0.],
            [0., 1., 0.],
            [0., 0., 1.]])
```

배열 데이터를 효과적으로 다루는 NumPy

- 배열의 데이터 타입 변환

```
In: np.array(['1.5', '0.62', '2', '3.14', '3.141592'])
Out: array(['1.5', '0.62', '2', '3.14', '3.141592'], dtype='<U8')
```

  - NumPy 데이터의 형식

기호	설명
'b'	불, bool
'i'	기호가 있는 정수, (signed) integer
'u'	기호가 없는 정수, unsigned integer
'f'	실수, floating-point
'c'	복소수, complex-floating point
'M'	날짜, datetime
'O'	파이썬 객체, (Python) objects
'S' 혹은 'a'	바이트 문자열, (byte) string
'U'	유니코드, Unicode

  - NumPy 배열의 형 변환

```
num_arr = str_arr.astype(dtype)
```

배열 데이터를 효과적으로 다루는 NumPy

– NumPy 배열의 형 변환

```
In: str_a1 = np.array(['1.567', '0.123', '5.123', '9', '8'])
    num_a1 = str_a1.astype(float)
    num_a1
Out: array([1.567, 0.123, 5.123, 9.   , 8.   ])
```

```
In: str_a1.dtype
Out: dtype('<U5')
```

```
In: num_a1.dtype
Out: dtype('float64')
```

배열 데이터를 효과적으로 다루는 NumPy

- 난수 배열의 생성

```
rand_num = np.random.rand([d0, d1, ..., dn])
rand_num = np.random.randint([low,] high [,size])
```

```
In: np.random.rand(2,3)
Out: array([[0.65311939, 0.89752463, 0.63411962],
            [0.1345534 , 0.27230463, 0.02711115]])
```

```
In: np.random.rand()
Out: 0.8324172369983784
```

```
In: np.random.rand(2,3,4)
Out: array([[[ 0.06256587,  0.48831201,  0.57252114,  0.78417988],
             [ 0.62835321,  0.13173961,  0.46895454,  0.00443031],
             [ 0.76377121,  0.71765738,  0.0828908 ,  0.57340376]],
            [[ 0.97789304,  0.94486134,  0.86353152,  0.2843577 ],
             [ 0.1634681 ,  0.39515681,  0.21691386,  0.19066458],
             [ 0.38078663,  0.35489043,  0.60452622,  0.91283752]]])
```

배열 데이터를 효과적으로 다루는 NumPy

- 난수 배열의 생성

```
In: np.random.randint(10, size=(3, 4))
Out: array([[4, 1, 8, 7],
            [2, 9, 3, 2],
            [2, 9, 3, 8]])
```

```
In: np.random.randint(1, 30)
Out: 12
```

배열 데이터를 효과적으로 다루는 NumPy

- 배열의 연산
  - 기본 연산

```
In: arr1 = np.array([10, 20, 30, 40])
    arr2 = np.array([1, 2, 3, 4])
```

```
In: arr1 + arr2
Out: array([11, 22, 33, 44])
```

```
In: arr1 - arr2
Out: array([ 9, 18, 27, 36])
```

```
In: arr2 * 2
Out: array([2, 4, 6, 8])
```

```
In: arr2 ** 2
Out: array([ 1,  4,  9, 16], dtype=int32)
```

```
In: arr1 * arr2
Out: array([ 10,  40,  90, 160])
```

```
In: arr1 / arr2
Out: array([ 10.,  10.,  10.,  10.])
```

배열 데이터를 효과적으로 다루는 NumPy

- 기본 연산

```
In: arr1 / (arr2 ** 2)
Out: array([10.        ,  5.        ,  3.33333333,  2.5       ])
```

```
In: arr1 > 20
Out: array([False, False,  True,  True])
```

- 통계를 위한 연산
  - sum(), mean(), std(), var(), min(), max(), cumsum(), cumprod() 등

```
In: arr3 = np.arange(5)
    arr3
Out: array([0, 1, 2, 3, 4])
```

```
In: [arr3.sum(), arr3.mean()]
Out: [10, 2.0]
```

```
In: [arr3.std(), arr3.var()]
Out: [1.4142135623730951, 2.0]
```

```
In: [arr3.min(), arr3.max()]
Out: [0, 4]
```

배열 데이터를 효과적으로 다루는 NumPy

– 통계를 위한 연산

```
In: arr4 = np.arange(1,5)
    arr4
Out: array([1, 2, 3, 4])
```

```
In: arr4.cumsum()
Out: array([ 1,  3,  6, 10], dtype=int32)
```

```
In: arr4.cumprod()
Out: array([ 1,  2,  6, 24], dtype=int32)
```

– 행렬 연산

행렬 연산	사용 예
행렬곱(matrix product)	A.dot(B), 혹은 np.dot(A,B)
전치행렬(transpose matrix)	A.transpose(), 혹은 np.transpose(A)
역행렬(inverse matrix)	np.linalg.inv(A)
행렬식(determinant)	np.linalg.det(A)

배열 데이터를 효과적으로 다루는 NumPy

– 행렬 연산

```
In: A = np.array([0, 1, 2, 3]).reshape(2,2)
    A
Out: array([[0, 1],
            [2, 3]])
```

```
In: B = np.array([3, 2, 0, 1]).reshape(2,2)
    B
Out: array([[3, 2],
            [0, 1]])
```

```
In: A.dot(B)
Out: array([[0, 1],
            [6, 7]])
```

```
In: np.dot(A,B)
Out: array([[0, 1],
            [6, 7]])
```

배열 데이터를 효과적으로 다루는 NumPy

– 행렬 연산

```
In: np.transpose(A)
Out: array([[0, 2],
            [1, 3]])
```

```
In: A.transpose()
Out: array([[0, 2],
            [1, 3]])
```

```
In: np.linalg.inv(A)
Out: array([[-1.5,  0.5],
            [ 1. ,  0. ]])
```

```
In: np.linalg.det(A)
Out: -2.0
```

배열 데이터를 효과적으로 다루는 NumPy

- 배열의 인덱싱과 슬라이싱
  - 배열에서 선택된 원소는 값을 가져오거나 변경할 수 있음
  - 인덱싱(Indexing): 배열의 위치나 조건을 지정해 배열의 원소를 선택
  - 슬라이싱(Slicing):범위를 지정해 배열의 원소를 선택
  - 배열의 인덱싱

```
배열명[위치]
```

```
In: a1 = np.array([0, 10, 20, 30, 40, 50])
    a1
Out: array([ 0, 10, 20, 30, 40, 50])
```

```
In: a1[0]
Out: 0
```

```
In: a1[4]
Out: 40
```

```
In: a1[5] = 70
    a1
Out: array([ 0, 10, 20, 30, 40, 70])
```

배열 데이터를 효과적으로 다루는 NumPy

- 배열에서 여러 개의 원소를 선택

```
배열명[[위치1, 위치2, ..., 위치n]]
```

```
In: a1[[1,3,4]]
Out: array([10, 30, 40])
```

- 2차원 배열에서 특정 위치의 원소를 선택

```
배열명[행_위치, 열_위치]
```

```
In: a2 = np.arange(10, 100, 10).reshape(3,3)
    a2
Out: array([[10, 20, 30],
            [40, 50, 60],
            [70, 80, 90]])
In: a2[0, 2]
Out: 30
```

배열 데이터를 효과적으로 다루는 NumPy

– 2차원 배열에서 특정 위치의 원소를 선택

```
In: a2[2, 2] = 95
    a2
Out: array([[10, 20, 30],
            [40, 50, 60],
            [70, 80, 95]])
```

```
In: a2[1]
Out: array([40, 50, 60])
```

```
In: a2[1] = np.array([45, 55, 65])
    a2
Out: array([[10, 20, 30],
            [45, 55, 65],
            [70, 80, 95]])
```

```
In: a2[1] = [47, 57, 67]
    a2
Out: array([[10, 20, 30],
            [47, 57, 67],
            [70, 80, 95]])
```

배열 데이터를 효과적으로 다루는 NumPy

- 2차원 배열에서 여러 원소를 선택

```
배열명[[행_위치1, 행_위치2, ..., 행_위치n], [열_위치1, 열_위치2, ..., 열_위치n]]
```

```
In: a2[[0, 2], [0, 1]]
Out: array([10, 80])
```

- 배열에 조건을 지정해 조건에 맞는 배열을 선택

```
배열명[조건]
```

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

```
In: a[(a % 2) == 0]
Out: array([2, 4, 6])
```

배열 데이터를 효과적으로 다루는 NumPy

- 배열의 슬라이싱

```
배열[시작_위치:끝_위치]
```

```
In: b1 = np.array([0, 10, 20, 30, 40, 50])
    b1[1:4]
Out: array([10, 20, 30])
```

```
In: b1[:3]
Out: array([ 0, 10, 20])
```

```
In: b1[2:]
Out: array([20, 30, 40, 50])
```

```
In: b1[2:5] = np.array([25, 35, 45])
    b1
Out: array([ 0, 10, 25, 35, 45, 50])
```

```
In: b1[3:6] = 60
    b1
Out: array([ 0, 10, 25, 60, 60, 60])
```

배열 데이터를 효과적으로 다루는 NumPy

– 2차원 배열의 슬라이싱

```
배열[행_시작_위치:행_끝_위치, 열_시작_위치:열_끝_위치]
```

– 특정 행을 선택한 후 열을 슬라이싱

```
배열[행_위치][열_시작_위치:열_끝_위치]
```

– 슬라이싱 예

```
In: b2 = np.arange(10, 100, 10).reshape(3,3)
    b2
Out: array([[10, 20, 30],
            [40, 50, 60],
            [70, 80, 90]])
```

```
In: b2[1:3, 1:3]
Out: array([[50, 60],
            [80, 90]])
```

```
In: b2[:3, 1:]
Out: array([[20, 30],
            [50, 60],
            [80, 90]])
```

배열 데이터를 효과적으로 다루는 NumPy

- 슬라이싱 예

```
In: b2[1][0:2]
Out: array([40, 50])
```

```
In: b2[0:2, 1:3] = np.array([[25, 35], [55, 65]])
    b2
Out: array([[10, 25, 35],
            [40, 55, 65],
            [70, 80, 90]])
```

Pandas

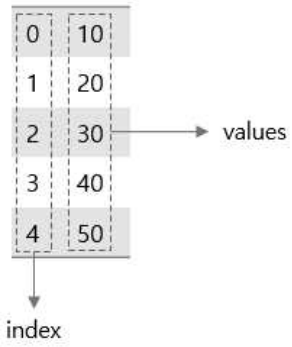
구조적 데이터 표시와 처리에 강한 pandas

- 데이터 분석과 처리를 쉽게 할 수 있게 도와주는 라이브러리
- NumPy를 기반으로 하지만 좀 더 복잡한 데이터 분석에 특화
- 아나콘다 배포판에 포함돼 있음
- pandas 홈페이지: http://pandas.pydata.org

구조적 데이터 표시와 처리에 강한 pandas

- 구조적 데이터 생성하기
  - Series를 활용한 데이터 생성

```
In: import pandas as pd
s = pd.Series(seq_data)
```




```
In: s1 = pd.Series([10, 20, 30, 40, 50])
    s1
Out: 0    10
     1    20
     2    30
     3    40
     4    50
     dtype: int64
```

구조적 데이터 표시와 처리에 강한 pandas

– Series를 활용한 데이터 생성

```
In: s1.index
    print(s1.index)
Out: RangeIndex(start=0, stop=5, step=1)
```

```
In: s1.values
Out: array([10, 20, 30, 40, 50], dtype=int64)
```

```
In: s2 = pd.Series(['a', 'b', 'c', 1, 2, 3])
    s2
Out: 0    a
    1    b
    2    c
    3    1
    4    2
    5    3
    dtype: object
```

구조적 데이터 표시와 처리에 강한 pandas

- Series를 활용한 데이터 생성

```
In: import numpy as np
    s3 = pd.Series([np.nan,10,30])
    s3
Out: 0    NaN
     1   10.0
     2   30.0
     dtype: float64
```

```
s = pd.Series(seq_data, index = index_seq)
```

```
In: index_date = ['2018-10-07','2018-10-08','2018-10-09','2018-10-10']
    s4 = pd.Series([200, 195, np.nan, 205], index = index_date)
    s4
Out: 2018-10-07   200.0
     2018-10-08   195.0
     2018-10-09     NaN
     2018-10-10   205.0
     dtype: float64
```

구조적 데이터 표시와 처리에 강한 pandas

– Series를 활용한 데이터 생성

```
s = pd.Series(dict_data)
```

```
In: s5 = pd.Series({'국어': 100, '영어': 95, '수학': 90})
    s5
Out: 국어   100
     수학   90
     영어   95
     dtype: int64
```

– 날짜 자동 생성: date_range

```
pd.date_range(start=None, end=None, periods=None, freq='D')
```

```
In: import pandas as pd
    pd.date_range(start='2019-01-01',end='2019-01-07')
Out: DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',
                    '2019-01-05', '2019-01-06', '2019-01-07'],
                    dtype='datetime64[ns]', freq='D')
```

구조적 데이터 표시와 처리에 강한 pandas

- 날짜 자동 생성: date_range

```
In: pd.date_range(start='2019/01/01',end='2019.01.07')
Out: DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',
                    '2019-01-05', '2019-01-06', '2019-01-07'],
                    dtype='datetime64[ns]', freq='D')
```

```
In: pd.date_range(start='01-01-2019',end='01/07/2019')
Out: DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',
                    '2019-01-05', '2019-01-06', '2019-01-07'],
                    dtype='datetime64[ns]', freq='D')
```

```
In: pd.date_range(start='2019-01-01',end='01.07.2019')
Out: DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',
                    '2019-01-05', '2019-01-06', '2019-01-07'],
                    dtype='datetime64[ns]', freq='D')
```

```
In: pd.date_range(start='2019-01-01', periods = 7)
Out: DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',
                    '2019-01-05', '2019-01-06', '2019-01-07'],
                    dtype='datetime64[ns]', freq='D')
```

구조적 데이터 표시와 처리에 강한 pandas

- 날짜 자동 생성: date_range
  - date_rage() 함수의 freq 옵션

약어	설명	부가 설명 및 사용 예
D	달력 날짜 기준 하루 주기	하루 주기: freq = 'D', 이틀 주기: freq = '2D'
B	업무 날짜 기준 하루 주기	업무일(월요일 ~ 금요일) 기준으로 생성. freq = 'B', freq = '3B'
W	일요일 시작 기준 일주일 주기	월요일: W-MON, 화요일: W-TUE. freq = 'W', freq = 'W-MON'
M	월말 날짜 기준 주기	한 달 주기: freq = 'M', 네 달 주기: freq = '4M'
BM	업무 월말 날짜 기준 주기	freq = 'BM', freq = '2BM'
MS	월초 날짜 기준 주기	freq = 'MS', freq = '3MS'
BMS	업무 월초 날짜 기준 주기	freq = 'BMS', freq = '3BMS'
Q	분기 끝 날짜 기준 주기	freq = 'Q', freq = '2Q'
BQ	업무 분기 끝 날짜 기준 주기	freq = 'BQ', freq = '2BQ'
QS	분기 시작 날짜 기준 주기	freq = 'QS', freq = '2QS'
BQS	업무 분기 시작 날짜 기준 주기	freq = 'BQS', freq = '2BQS'
A	일년 끝 날짜 기준 주기	freq = 'A', freq = '5A'
BA	업무 일년 끝 날짜 기준 주기	freq = 'BA', freq = '3BA'
AS	일년 시작 날짜 기준 주기	freq = 'AS', freq = '2AS'
BAS	업무 일년 시작 날짜 기준 주기	freq = 'BAS', freq = '2BAS'
H	시간 기준 주기	1시간 주기: freq = 'H', 2시간 주기: freq = '2H'
BH	업무 시간 기준 주기	업무 시간 (09:00 ~ 17:00) 기준으로 생성
T, min	분 주기	10분 주기: freq = '10T', 30분 주기: freq = '30min'
S	초 주기	1초 주기: freq = 'S', 10초 주기: freq = '10S'

구조적 데이터 표시와 처리에 강한 pandas

- 날짜 자동 생성: date_range

```
In: pd.date_range(start='2019-01-01', periods = 4, freq = '2D')
Out: DatetimeIndex(['2019-01-01', '2019-01-03', '2019-01-05', '2019-01-07'],
    dtype='datetime64[ns]', freq='2D')
```

```
In: pd.date_range(start='2019-01-01', periods = 4, freq = 'W')
Out: DatetimeIndex(['2019-01-06', '2019-01-13', '2019-01-20', '2019-01-27'],
    dtype='datetime64[ns]', freq='W-SUN')
```

```
In: pd.date_range(start='2019-01-01', periods = 12, freq = '2BM')
Out: DatetimeIndex(['2019-01-31', '2019-03-29', '2019-05-31', '2019-07-31',
                    '2019-09-30', '2019-11-29', '2020-01-31', '2020-03-31',
                    '2020-05-29', '2020-07-31', '2020-09-30', '2020-11-30'],
                    dtype='datetime64[ns]', freq='2BM')
```

```
In: pd.date_range(start='2019-01-01', periods = 4, freq = 'QS')
Out: DatetimeIndex(['2019-01-01', '2019-04-01', '2019-07-01', '2019-10-01'],
    dtype='datetime64[ns]', freq='QS-JAN')
```

```
In: pd.date_range(start='2019-01-01', periods = 3, freq = 'AS')
Out: DatetimeIndex(['2019-01-01', '2020-01-01', '2021-01-01'], dtype='datetime64[ns]',
    freq='AS-JAN')
```

구조적 데이터 표시와 처리에 강한 pandas

– 날짜 자동 생성: date_range

```
In: pd.date_range(start = '2019-01-01 08:00', periods = 10, freq='H')
Out: DatetimeIndex(['2019-01-01 08:00:00', '2019-01-01 09:00:00',
                    '2019-01-01 10:00:00', '2019-01-01 11:00:00',
                    '2019-01-01 12:00:00', '2019-01-01 13:00:00',
                    '2019-01-01 14:00:00', '2019-01-01 15:00:00',
                    '2019-01-01 16:00:00', '2019-01-01 17:00:00'],
                    dtype='datetime64[ns]', freq='H')
```

```
In: pd.date_range(start = '2019-01-01 08:00', periods = 10, freq='BH')
Out: DatetimeIndex(['2019-01-01 09:00:00', '2019-01-01 10:00:00',
                     '2019-01-01 11:00:00', '2019-01-01 12:00:00',
                     '2019-01-01 13:00:00', '2019-01-01 14:00:00',
                     '2019-01-01 15:00:00', '2019-01-01 16:00:00',
                     '2019-01-02 09:00:00', '2019-01-02 10:00:00'],
                     dtype='datetime64[ns]', freq='BH')
```

```
In: pd.date_range(start = '2019-01-01 10:00', periods = 4, freq='30min')
Out: DatetimeIndex(['2019-01-01 10:00:00', '2019-01-01 10:30:00',
                     '2019-01-01 11:00:00', '2019-01-01 11:30:00'],
                     dtype='datetime64[ns]', freq='30T')
```

구조적 데이터 표시와 처리에 강한 pandas

- 날짜 자동 생성: date_range

```
In: pd.date_range(start = '2019-01-01 10:00', periods = 4, freq='30T')
Out: DatetimeIndex(['2019-01-01 10:00:00', '2019-01-01 10:30:00',
                    '2019-01-01 11:00:00', '2019-01-01 11:30:00'],
                    dtype='datetime64[ns]', freq='30T')
```

```
In: pd.date_range(start = '2019-01-01 10:00:00', periods = 4, freq='10S')
Out: DatetimeIndex(['2019-01-01 10:00:00', '2019-01-01 10:00:10',
                    '2019-01-01 10:00:20', '2019-01-01 10:00:30'],
                    dtype='datetime64[ns]', freq='10S')
```

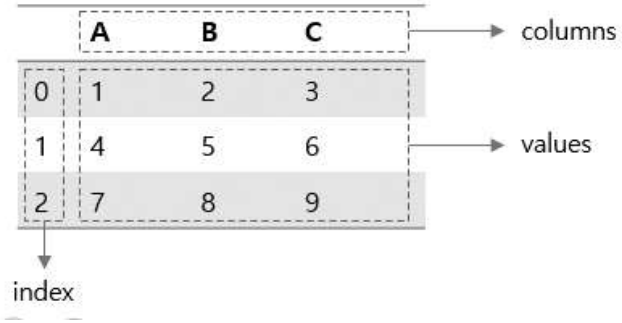
```
In: index_date = pd.date_range(start = '2019-03-01', periods = 5, freq='D')
    pd.Series([51, 62, 55, 49, 58], index = index_date )
Out: 2019-03-01    51
     2019-03-02    62
     2019-03-03    55
     2019-03-04    49
     2019-03-05    58
     Freq: D, dtype: int64
```

구조적 데이터 표시와 처리에 강한 pandas

- DataFrame을 활용한 데이터 생성
  - DataFrame: 표(Table)와 같은 2차원 데이터 처리를 위한 형식

```
df = pd.DataFrame(data [, index = index_data, columns = columns_data])
```

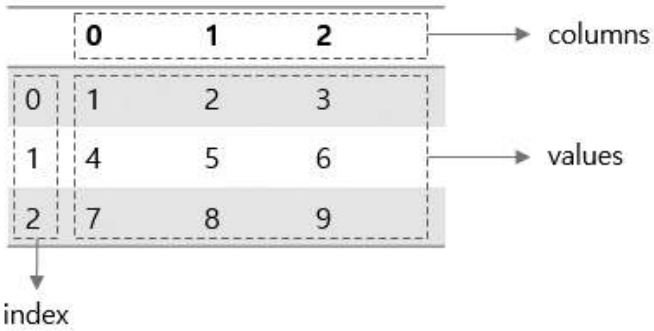
  - DataFrame의 구조




```
In: import pandas as pd
    pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
Out:
       0  1  2
    0  1  2  3
    1  4  5  6
    2  7  8  9
```

구조적 데이터 표시와 처리에 강한 pandas

– 자동으로 생성된 index와 columns를 갖는 DataFrame 데이터




```
In: import numpy as np
    import pandas as pd
    data_list = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])
    pd.DataFrame(data_list)
Out:
        0   1   2
    0  10  20  30
    1  40  50  60
    2  70  80  90
```

구조적 데이터 표시와 처리에 강한 pandas

- DataFrame 데이터 예제

```
In: import numpy as np
    import pandas as pd

    data = np.array([[1, 2, 3], [4, 5, 6], [7, 8 ,9], [10, 11, 12]])
    index_date = pd.date_range('2019-09-01', periods=4)
    columns_list = ['A', 'B', 'C']
    pd.DataFrame(data, index=index_date, columns=columns_list)
Out:
                    A     B     C
        2019-09-01      1     2     3
        2019-09-02      4     5     6
        2019-09-03      7     8     9
        2019-09-04      10    11    12
```

구조적 데이터 표시와 처리에 강한 pandas

– DataFrame 데이터 예제

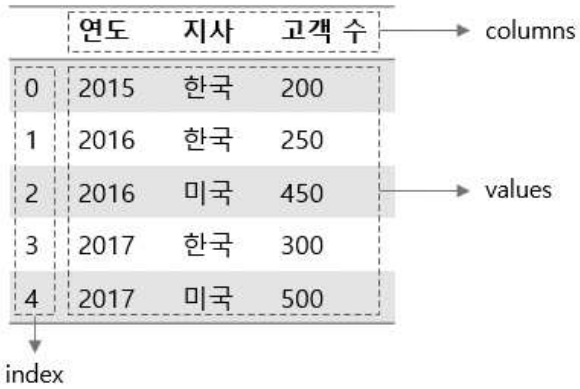
```
In: table_data = {'연도': [2015, 2016, 2016, 2017, 2017],
                  '지사': ['한국', '한국', '미국', '한국','미국'],
                  '고객 수': [200, 250, 450, 300, 500]}
    table_data
Out: {'고객 수': [200, 250, 450, 300, 500],
     '연도': [2015, 2016, 2016, 2017, 2017],
     '지사': ['한국', '한국', '미국', '한국', '미국']}
```

```
In: pd.DataFrame(table_data)
Out:
        고객 수      연도     지사
    0   200       2015    한국
    1   250       2016    한국
    2   450       2016    미국
    3   300       2017    한국
    4   500       2017    미국
```

구조적 데이터 표시와 처리에 강한 pandas

– DataFrame 데이터 예제

```
In: df = pd.DataFrame(table_data, columns=['연도', '지사', '고객 수'])
    df
Out:
        연도    지사    고객 수
    0   2015    한국    200
    1   2016    한국    250
    2   2016    미국    450
    3   2017    한국    300
    4   2017    미국    500
```



구조적 데이터 표시와 처리에 강한 pandas

- DataFrame 데이터 예제

```
In: df.index
Out: RangeIndex(start=0, stop=5, step=1)
```

```
In: df.columns
Out: Index(['연도', '지사', '고객 수'], dtype='object')
```

```
In: df.values
Out: array([[2015, '한국', 200],
            [2016, '한국', 250],
            [2016, '미국', 450],
            [2017, '한국', 300],
            [2017, '미국', 500]], dtype=object)
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터 연산

```
In: s1 = pd.Series([1, 2, 3, 4, 5])
    s2 = pd.Series([10, 20, 30, 40, 50])
    s1 + s2
Out: 0    11
    1    22
    2    33
    3    44
    4    55
    dtype: int64
```

```
In: s2 - s1
Out: 0    9
    1    18
    2    27
    3    36
    4    45
    dtype: int64
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터 연산

```
In: s1 * s2
Out: 0    10
     1    40
     2    90
     3   160
     4   250
     dtype: int64
```

```
In: s2 / s1
Out: 0    10.0
     1    10.0
     2    10.0
     3    10.0
     4    10.0
     dtype: float64
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터 연산
  - 데이터 크기가 다른 경우

```
In: s3 = pd.Series([1, 2, 3, 4])
    s4 = pd.Series([10, 20, 30, 40, 50])
    s3 + s4
Out: 0    11.0
     1    22.0
     2    33.0
     3    44.0
     4     NaN
     dtype: float64
```

```
In: s4 - s3
Out: 0     9.0
     1    18.0
     2    27.0
     3    36.0
     4     NaN
     dtype: float64
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터 크기가 다른 경우

```
In: s3 * s4
Out: 0    10.0
     1    40.0
     2    90.0
     3   160.0
     4     NaN
     dtype: float64
```

```
In: s4/s3
Out: 0    10.0
     1    10.0
     2    10.0
     3    10.0
     4     NaN
     dtype: float64
```

구조적 데이터 표시와 처리에 강한 pandas

- DataFrame 간의 사칙 연산

```
In: table_data1 = {'A': [1, 2, 3, 4, 5],
                   'B': [10, 20, 30, 40, 50],
                   'C': [100, 200, 300, 400, 500]}
    df1 = pd.DataFrame(table_data1)
    df1
Out:
```

	A	B	C
0	1	10	100
1	2	20	200
2	3	30	300
3	4	40	400
4	5	50	500

구조적 데이터 표시와 처리에 강한 pandas

- DataFrame 간의 사칙 연산

```
In: table_data2 = {'A': [6, 7, 8],
                   'B': [60, 70, 80],
                   'C': [600, 700, 800]}
    df2 = pd.DataFrame(table_data2)
    df2
Out:
        A    B     C
    0   6    60    600
    1   7    70    700
    2   8    80    800
```

```
In: df1 + df2
Out:
        A       B        C
    0   7.0     70.0     700.0
    1   9.0     90.0     900.0
    2   11.0    110.0    1100.0
    3   NaN     NaN      NaN
    4   NaN     NaN      NaN
```

구조적 데이터 표시와 처리에 강한 pandas

- 통계 분석을 위한 메서드

연도	봄	여름	가을	겨울
2012	256.5	770.6	363.5	139.3
2013	264.3	567.5	231.2	59.9
2014	215.9	599.8	293.1	76.9
2015	223.2	387.1	247.7	109.1
2016	312.8	446.2	381.6	108.1

2012년부터 2016년까지 우리나라의 계절별 강수량

```
In: table_data3 = {'봄':  [256.5, 264.3, 215.9, 223.2, 312.8],
                   '여름': [770.6, 567.5, 599.8, 387.1, 446.2],
                   '가을': [363.5, 231.2, 293.1, 247.7, 381.6],
                   '겨울': [139.3, 59.9, 76.9, 109.1, 108.1]}
    columns_list = ['봄', '여름', '가을', '겨울']
    index_list = ['2012', '2013', '2014', '2015', '2016']
    df3 = pd.DataFrame(table_data3, columns = columns_list, index = index_list)
    df3
Out:
            봄      여름      가을      겨울
    2012    256.5   770.6     363.5     139.3
    2013    264.3   567.5     231.2     59.9
    2014    215.9   599.8     293.1     76.9
    2015    223.2   387.1     247.7     109.1
    2016    312.8   446.2     381.6     108.1
```

구조적 데이터 표시와 처리에 강한 pandas

- 통계 분석을 위한 메서드

```
In: df3.mean()
Out: 봄      254.54
     여름    554.24
     가을    303.42
     겨울     98.66
     dtype: float64
```

```
In: df3.std()
Out: 봄      38.628267
     여름   148.888895
     가을    67.358496
     겨울    30.925523
     dtype: float64
```

구조적 데이터 표시와 처리에 강한 pandas

- 통계 분석을 위한 메서드

```
In: df3.mean(axis=1)
Out: 2012    382.475
     2013    280.725
     2014    296.425
     2015    241.775
     2016    312.175
     dtype: float64
```

```
In: df3.std(axis=1)
Out: 2012    274.472128
     2013    211.128782
     2014    221.150739
     2015    114.166760
     2016    146.548658
     dtype: float64
```

구조적 데이터 표시와 처리에 강한 pandas

- 통계 분석을 위한 메서드

```
In: df3.describe()
Out:
              봄          여름          가을          겨울
count    5.000000     5.000000     5.000000     5.000000
mean   254.540000   554.240000   303.420000    98.660000
std     38.628267   148.888895    67.358496    30.925523
min    215.900000   387.100000   231.200000    59.900000
25%    223.200000   446.200000   247.700000    76.900000
50%    256.500000   567.500000   293.100000   108.100000
75%    264.300000   599.800000   363.500000   109.100000
max    312.800000   770.600000   381.600000   139.300000
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터를 원하는 대로 선택하기
  - 2010년부터 2017년까지 노선별 KTX 이용자 수

연도	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2011	39060	7313	3627	309	-
2012	39896	6967	4168	1771	-
2013	42005	6873	4088	1954	-
2014	43621	6626	4424	2244	-
2015	41702	8675	4606	3146	2395
2016	41266	10622	4984	3945	3786
2017	32427	9228	5570	5766	6667

구조적 데이터 표시와 처리에 강한 pandas

- 데이터를 원하는 대로 선택하기

```
In: import pandas as pd
    import numpy as np

    KTX_data = {'경부선 KTX': [39060, 39896, 42005, 43621, 41702, 41266, 32427],
                '호남선 KTX': [7313, 6967, 6873, 6626, 8675, 10622, 9228],
                '경전선 KTX': [3627, 4168, 4088, 4424, 4606, 4984, 5570],
                '전라선 KTX': [309, 1771, 1954, 2244, 3146, 3945, 5766],
                '동해선 KTX': [np.nan,np.nan, np.nan, np.nan, 2395, 3786, 6667]}
    col_list = ['경부선 KTX','호남선 KTX','경전선 KTX','전라선 KTX','동해선 KTX']
    index_list = ['2011', '2012', '2013', '2014', '2015', '2016', '2017']
    df_KTX = pd.DataFrame(KTX_data, columns = col_list, index = index_list)
    df_KTX
Out:
```

	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2011	39060	7313	3627	309	NaN
2012	39896	6967	4168	1771	NaN
2013	42005	6873	4088	1954	NaN
2014	43621	6626	4424	2244	NaN
2015	41702	8675	4606	3146	2395.0
2016	41266	10622	4984	3945	3786.0
2017	32427	9228	5570	5766	6667.0

구조적 데이터 표시와 처리에 강한 pandas

- 데이터를 원하는 대로 선택하기

```
In: df_KTX.index
Out: Index(['2011', '2012', '2013', '2014', '2015', '2016', '2017'], dtype='object')
```

```
In: df_KTX.columns
Out: Index(['경부선 KTX', '호남선 KTX', '경전선 KTX', '전라선 KTX', '동해선 KTX'], dtype='object')
```

```
In: df_KTX.values
Out: array([[39060.,  7313.,  3627.,   309.,    nan],
            [39896.,  6967.,  4168.,  1771.,    nan],
            [42005.,  6873.,  4088.,  1954.,    nan],
            [43621.,  6626.,  4424.,  2244.,    nan],
            [41702.,  8675.,  4606.,  3146.,  2395.],
            [41266., 10622.,  4984.,  3945.,  3786.],
            [32427.,  9228.,  5570.,  5766.,  6667.]])
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터를 원하는 대로 선택하기
  - 처음 일부분과 끝 일부분만 선택

```
DataFrame_data.head([n])
DataFrame_data.tail([n])
```

```
In: df_KTX.head()
Out:
```

	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2011	39060	7313	3627	309	NaN
2012	39896	6967	4168	1771	NaN
2013	42005	6873	4088	1954	NaN
2014	43621	6626	4424	2244	NaN
2015	41702	8675	4606	3146	2395.0

```
In: df_KTX.tail()
Out:
```

	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2013	42005	6873	4088	1954	NaN
2014	43621	6626	4424	2244	NaN
2015	41702	8675	4606	3146	2395.0
2016	41266	10622	4984	3945	3786.0
2017	32427	9228	5570	5766	6667.0

구조적 데이터 표시와 처리에 강한 pandas

– 처음 일부분과 끝 일부분만 선택

```
In: df_KTX.head(3)
Out:
```

	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2011	39060	7313	3627	309	NaN
2012	39896	6967	4168	1771	NaN
2013	42005	6873	4088	1954	NaN

```
In: df_KTX.tail(2)
Out:
```

	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2016	41266	10622	4984	3945	3786.0
2017	32427	9228	5570	5766	6667.0

구조적 데이터 표시와 처리에 강한 pandas

– 연속된 구간의 행 데이터 선택

DataFrame_data[행_시작_위치:행_끝_위치]

In: df_KTX[1:2]

Out:

	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2012	39896	6967	4168	1771	NaN

In: df_KTX[2:5]

Out:

	경부선 KTX	호남선 KTX	경전선 KTX	전라선 KTX	동해선 KTX
2013	42005	6873	4088	1954	NaN
2014	43621	6626	4424	2244	NaN
2015	41702	8675	4606	3146	2395.0

구조적 데이터 표시와 처리에 강한 pandas

- index 항목 이름을 지정해 행을 선택

```
DataFrame_data.loc[index_name]
```

```
In: df_KTX.loc['2011']
Out: 경부선 KTX    39060.0
   호남선 KTX     7313.0
   경전선 KTX     3627.0
   전라선 KTX      309.0
   동해선 KTX        NaN
   Name: 2011, dtype: float64
```

- index 항목 이름으로 구간을 지정해 연속된 구간의 행을 선택

```
DataFrame_data.loc[start_index_name:end_index_name]
```

```
In: df_KTX.loc['2013':'2016']
Out:
          경부선 KTX    호남선 KTX    경전선 KTX    전라선 KTX    동해선 KTX
    2013     42005        6873         4088         1954          NaN
    2014     43621        6626         4424         2244          NaN
    2015     41702        8675         4606         3146         2395.0
    2016     41266       10622          4984         3945          3786.0
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터에서 하나의 열만 선택

```
DataFrame_data[column_name]
```

```
In: df_KTX['경부선 KTX']
Out: 2011    39060
     2012    39896
     2013    42005
     2014    43621
     2015    41702
     2016    41266
     2017    32427
     Name: 경부선 KTX, dtype: int64
```

구조적 데이터 표시와 처리에 강한 pandas

- 하나의 열을 선택한 후 index의 범위를 지정해 선택

```
DataFrame_data[column_name][start_index_name:end_index_name]
DataFrame_data[column_name][start_index_pos:end_index_pos]
```

```
In: df_KTX['경부선 KTX']['2012':'2014']
Out: 2012    39896
     2013    42005
     2014    43621
     Name: 경부선 KTX, dtype: int64
```

```
In: df_KTX['경부선 KTX'][2:5]
Out: 2013    42005
     2014    43621
     2015    41702
     Name: 경부선 KTX, dtype: int64
```

구조적 데이터 표시와 처리에 강한 pandas

– 하나의 원소만 선택

```
DataFrame_data.loc[index_name][column_name]
DataFrame_data.loc[index_name, column_name]
DataFrame_data[column_name][index_name]
DataFrame_data[column_name][index_pos]
DataFrame_data[column_name].loc[index_name]
```

```
In: df_KTX.loc['2016']['호남선 KTX']
Out: 10622.0
```

```
In: df_KTX.loc['2016','호남선 KTX']
Out: 10622
```

```
In: df_KTX['호남선 KTX']['2016']
Out: 10622
```

```
In: df_KTX['호남선 KTX'][5]
Out: 10622
```

```
In: df_KTX['호남선 KTX'].loc['2016']
Out: 10622
```

구조적 데이터 표시와 처리에 강한 pandas

- DataFrame의 행과 열을 바꾸는 방법(전치)

```
DataFrame_data.T
```

```
In: df_KTX.T
Out:
              2011     2012     2013     2014     2015     2016     2017
경부선 KTX   39060.0  39896.0  42005.0  43621.0  41702.0  41266.0  32427.0
호남선 KTX    7313.0   6967.0   6873.0   6626.0   8675.0  10622.0   9228.0
경전선 KTX    3627.0   4168.0   4088.0   4424.0   4606.0   4984.0   5570.0
전라선 KTX     309.0   1771.0   1954.0   2244.0   3146.0   3945.0   5766.0
동해선 KTX       NaN      NaN      NaN      NaN   2395.0   3786.0   6667.0
```

```
In: df_KTX
Out:
      경부선 KTX  호남선 KTX  경전선 KTX  전라선 KTX  동해선 KTX
2011       39060        7313        3627         309         NaN
2012       39896        6967        4168        1771         NaN
2013       42005        6873        4088        1954         NaN
2014       43621        6626        4424        2244         NaN
2015       41702        8675        4606        3146      2395.0
2016       41266       10622        4984        3945      3786.0
2017       32427        9228        5570        5766      6667.0
```

구조적 데이터 표시와 처리에 강한 pandas

- 열의 항목을 지정해 열의 순서를 지정

```
In: df_KTX[['동해선 KTX', '전라선 KTX', '경전선 KTX', '호남선 KTX', '경부선 KTX']]
Out:
```

	동해선 KTX	전라선 KTX	경전선 KTX	호남선 KTX	경부선 KTX
2011	NaN	309	3627	7313	39060
2012	NaN	1771	4168	6967	39896
2013	NaN	1954	4088	6873	42005
2014	NaN	2244	4424	6626	43621
2015	2395.0	3146	4606	8675	41702
2016	3786.0	3945	4984	10622	41266
2017	6667.0	5766	5570	9228	32427

구조적 데이터 표시와 처리에 강한 pandas

- 데이터 통합하기
  - 세로로 증가하는 방향으로 통합하기
  - 가로로 증가하는 방향으로 통합하기
  - 특정 열을 기준으로 통합하기
- 세로 방향으로 통합하기

```
DataFrame_data1.append(DataFrame_data2 [,ignore_index=True])
```

```
In: import pandas as pd
    import numpy as np
    df1 = pd.DataFrame({'Class1': [95, 92, 98, 100],
                        'Class2': [91, 93, 97, 99]})
    df1
Out:
          Class1    Class2
       0  95        91
       1  92        93
       2  98        97
       3  100       99
```

구조적 데이터 표시와 처리에 강한 pandas

- 세로 방향으로 통합하기

```
In: df2 = pd.DataFrame({'Class1': [87, 89],
                        'Class2': [85, 90]})
    df2
Out:
        Class1    Class2
    0   87        85
    1   89        90
```

  - append()로 데이터 추가

```
In: df1.append(df2)
Out:
        Class1    Class2
    0   95        91
    1   92        93
    2   98        97
    3   100       99
    0   87        85
    1   89        90
```

구조적 데이터 표시와 처리에 강한 pandas

- 세로 방향으로 통합하기

```
In: df1.append(df2, ignore_index=True)
Out:
      Class1   Class2
   0  95       91
   1  92       93
   2  98       97
   3  100      99
   4  87       85
   5  89       90
```

  - 열이 하나만 있는 DataFrame 생성

```
In: df3 = pd.DataFrame({'Class1': [96, 83]})
   df3
Out:
      Class1
   0  96
   1  83
```

구조적 데이터 표시와 처리에 강한 pandas

– 열이 두 개인 데이터(df2)에 열이 하나인 DataFrame 데이터(df3)를 추가

```
In: df2.append(df3, ignore_index=True)
Out:
      Class1   Class2
   0  87       85.0
   1  89       90.0
   2  96       NaN
   3  83       NaN
```

- 가로 방향으로 통합하기

```
DataFrame_data1.join(DataFrame_data2)
```

```
In: df4 = pd.DataFrame({'Class3': [93, 91, 95, 98]})
   df4
Out:
      Class3
   0  93
   1  91
   2  95
   3  98
```

구조적 데이터 표시와 처리에 강한 pandas

- 가로 방향으로 통합하기

```
In: df1.join(df4)
Out:
        Class1   Class2   Class3
    0   95       91       93
    1   92       93       91
    2   98       97       95
    3   100      99       98
```

  - index 라벨을 지정한 경우

```
In: index_label = ['a','b','c','d']
    df1a = pd.DataFrame({'Class1': [95, 92, 98, 100],
                         'Class2': [91, 93, 97, 99]}, index= index_label)
    df4a = pd.DataFrame({'Class3': [93, 91, 95, 98]}, index=index_label)
    df1a.join(df4a)
Out:
        Class1   Class2   Class3
    a   95       91       93
    b   92       93       91
    c   98       97       95
    d   100      99       98
```

구조적 데이터 표시와 처리에 강한 pandas

- index의 크기가 다른 경우

```
In: df5 = pd.DataFrame({'Class4': [82, 92]})
    df5
Out:
          Class4
       0    82
       1    92
```

```
In: df1.join(df5)
Out:
          Class1    Class2    Class4
       0  95        91        82.0
       1  92        93        92.0
       2  98        97        NaN
       3  100       99        NaN
```

구조적 데이터 표시와 처리에 강한 pandas

- 특정 열을 기준으로 통합하기

```
DataFrame_left_data.merge(DataFrame_right_data)
```

```
In: df_A_B = pd.DataFrame({'판매월': ['1월', '2월', '3월', '4월'],
                           '제품A': [100, 150, 200, 130],
                           '제품B': [90, 110, 140, 170]})
    df_A_B
Out:
        제품A    제품B    판매월
    0   100      90       1월
    1   150      110      2월
    2   200      140      3월
    3   130      170      4월
```

```
In: df_C_D = pd.DataFrame({'판매월': ['1월', '2월', '3월', '4월'],
                           '제품C': [112, 141, 203, 134],
                           '제품D': [90, 110, 140, 170]})
    df_C_D
Out:
        제품C    제품D    판매월
    0   112      90       1월
    1   141      110      2월
    2   203      140      3월
    3   134      170      4월
```

구조적 데이터 표시와 처리에 강한 pandas

- 특정 열을 기준으로 통합하기

```
In: df_A_B.merge(df_C_D)
Out:
      제품A    제품B    판매월    제품C    제품D
  0   100      90       1월       112      90
  1   150      110      2월       141      110
  2   200      140      3월       203      140
  3   130      170      4월       134      170
```

  - 두 개의 DataFrame이 특정 열을 기준으로 일부만 공통된 값을 갖는 경우

```
DataFrame_left_data.merge(DataFrame_right_data, how=merge_method, on=key_label)
```

    - merge() 함수의 how 선택 인자에 따른 통합 방법

how 선택 인자	설명
left	왼쪽 데이터는 모두 선택하고 지정된 열(key)에 값이 있는 오른쪽 데이터를 선택
right	오른쪽 데이터는 모두 선택하고 지정된 열(key)에 값이 있는 왼쪽 데이터를 선택
outer	지정된 열(key)을 기준으로 왼쪽과 오른쪽 데이터를 모두 선택
inner	지정된 열(key)을 기준으로 왼쪽과 오른쪽 데이터 중 공통 항목만 선택(기본값)

구조적 데이터 표시와 처리에 강한 pandas

– 두 개의 DataFrame이 특정 열을 기준으로 일부만 공통된 값을 갖는 경우

```
In: df_left = pd.DataFrame({'key':['A','B','C'], 'left': [1, 2, 3]})
    df_left
Out:
        key    left
    0   A      1
    1   B      2
    2   C      3
```

```
In: df_right = pd.DataFrame({'key':['A','B','D'], 'right': [4, 5, 6]})
    df_right
Out:
        key    right
    0   A      4
    1   B      5
    2   D      6
```

```
In: df_left.merge(df_right, how='left', on = 'key')
Out:
          key    left   right
    0     A      1      4.0
    1     B      2      5.0
    2     C      3      NaN
```

구조적 데이터 표시와 처리에 강한 pandas

– 두 개의 DataFrame이 특정 열을 기준으로 일부만 공통된 값을 갖는 경우

```
In: df_left.merge(df_right, how='right', on = 'key')
Out:
      key   left   right
0     A     1.0    4
1     B     2.0    5
2     D     NaN    6
```

```
In: df_left.merge(df_right, how='outer', on = 'key')
Out:
      key   left   right
0     A     1.0    4.0
1     B     2.0    5.0
2     C     3.0    NaN
3     D     NaN    6.0
```

```
In: df_left.merge(df_right, how='inner', on = 'key')
Out:
      key   left   right
0     A     1      4
1     B     2      5
```

구조적 데이터 표시와 처리에 강한 pandas

- 데이터 파일을 읽고 쓰기
  - 표 형식의 데이터 파일을 읽기

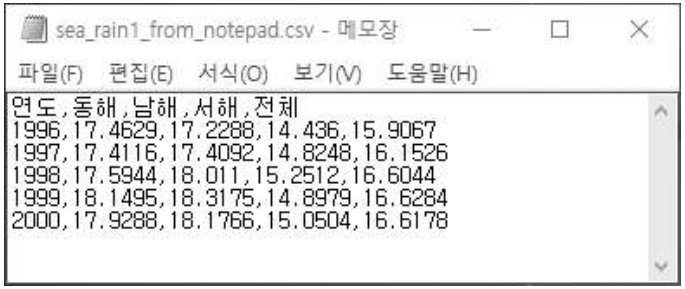
```
DataFrame_data = pd.read_csv(file_name [, options])
```

```
In: %%writefile C:\myPyCode\data\sea_rain1.csv
    연도,동해,남해,서해,전체
    1996,17.4629,17.2288,14.436,15.9067
    1997,17.4116,17.4092,14.8248,16.1526
    1998,17.5944,18.011,15.2512,16.6044
    1999,18.1495,18.3175,14.8979,16.6284
    2000,17.9288,18.1766,15.0504,16.6178
Out: Writing C:\myPyCode\data\sea_rain1.csv
```

```
In: import pandas as pd
    pd.read_csv('C:/myPyCode/data/sea_rain1.csv')
Out:
        연도     동해      남해      서해      전체
    0  1996  17.4629  17.2288  14.4360  15.9067
    1  1997  17.4116  17.4092  14.8248  16.1526
    2  1998  17.5944  18.0110  15.2512  16.6044
    3  1999  18.1495  18.3175  14.8979  16.6284
    4  2000  17.9288  18.1766  15.0504  16.6178
```

구조적 데이터 표시와 처리에 강한 pandas

– 표 형식의 데이터 파일을 읽기




```
In: pd.read_csv('C:/myPyCode/data/sea_rain1_from_notepad.csv', encoding = "cp949")
Out:
```

	연도	동해	남해	서해	전체
0	1996	17.4629	17.2288	14.4360	15.9067
1	1997	17.4116	17.4092	14.8248	16.1526
2	1998	17.5944	18.0110	15.2512	16.6044
3	1999	18.1495	18.3175	14.8979	16.6284
4	2000	17.9288	18.1766	15.0504	16.6178

```
In: %%writefile C:\myPyCode\data\sea_rain1_space.txt
    연도 동해 남해 서해 전체
    1996 17.4629 17.2288 14.436 15.9067
    1997 17.4116 17.4092 14.8248 16.1526
    1998 17.5944 18.011 15.2512 16.6044
    1999 18.1495 18.3175 14.8979 16.6284
    2000 17.9288 18.1766 15.0504 16.6178
Out: Writing C:\myPyCode\data\sea_rain1_space.txt
```

구조적 데이터 표시와 처리에 강한 pandas

– 표 형식의 데이터 파일을 읽기

```
In: pd.read_csv('C:/myPyCode/data/sea_rain1_space.txt', sep=" ")
Out:
     연도    동해      남해      서해      전체
0  1996  17.4629  17.2288  14.4360  15.9067
1  1997  17.4116  17.4092  14.8248  16.1526
2  1998  17.5944  18.0110  15.2512  16.6044
3  1999  18.1495  18.3175  14.8979  16.6284
4  2000  17.9288  18.1766  15.0504  16.6178
```

```
In: pd.read_csv('C:/myPyCode/data/sea_rain1.csv', index_col="연도" )
Out:
        동해      남해      서해      전체
연도
1996  17.4629  17.2288  14.4360  15.9067
1997  17.4116  17.4092  14.8248  16.1526
1998  17.5944  18.0110  15.2512  16.6044
1999  18.1495  18.3175  14.8979  16.6284
2000  17.9288  18.1766  15.0504  16.6178
```

구조적 데이터 표시와 처리에 강한 pandas

– 표 형식의 데이터를 파일로 쓰기

```
DataFrame_data = pd.to_csv(file_name [, options])
```

```
In: df_WH = pd.DataFrame({'Weight':[62, 67, 55, 74],
                          'Height':[165, 177, 160, 180]},
                          index=['ID_1', 'ID_2', 'ID_3', 'ID_4'])
    df_WH.index.name = 'User'
    df_WH
Out:
            Height     Weight
      User
      ID_1    165        62
      ID_2    177        67
      ID_3    160        55
      ID_4    180        74
```

```
In: bmi = df_WH['Weight']/(df_WH['Height']/100)**2
    bmi
Out: User
     ID_1    22.773186
     ID_2    21.385936
     ID_3    21.484375
     ID_4    22.839506
     dtype: float64
```

구조적 데이터 표시와 처리에 강한 pandas

– 표 형식의 데이터를 파일로 쓰기

```
In: df_WH['BMI'] = bmi
    df_WH
Out:
             Height     Weight       BMI
      User
      ID_1    165        62         22.773186
      ID_2    177        67         21.385936
      ID_3    160         55        21.484375
      ID_4    180        74         22.839506
```

```
In: df_WH.to_csv('C:/myPyCode/data/save_DataFrame.csv')
```

```
In: !type C:\myPyCode\data\save_DataFrame.csv
Out: User,Height,Weight,BMI
    ID_1,165,62,22.77318640955005
    ID_2,177,67,21.38593635289987
    ID_3,160,55,21.484374999999996
    ID_4,180,74,22.839506172839506
```

구조적 데이터 표시와 처리에 강한 pandas

– 표 형식의 데이터를 파일로 쓰기

```
In: df_pr = pd.DataFrame({'판매가격':[2000, 3000, 5000, 10000],
                          '판매량':[32, 53, 40, 25]},
                          index=['P1001', 'P1002', 'P1003', 'P1004'])
    df_pr.index.name = '제품번호'
    df_pr
Out:
                판매가격    판매량
        제품번호
        P1001       2000        32
        P1002       3000        53
        P1003       5000        40
        P1004       10000        25
```

```
In: file_name = 'C:/myPyCode/data/save_DataFrame_cp949.txt'
    df_pr.to_csv(file_name, sep=" ", encoding = "cp949")
```

```
In: !type C:\myPyCode\data\save_DataFrame_cp949.txt
Out: 제품번호 판매가격 판매량
     P1001 2000 32
     P1002 3000 53
     P1003 5000 40
     P1004 10000 25
```

구조적 데이터 표시와 처리에 강한 pandas

– 표 형식의 데이터를 파일로 쓰기

```
In: df_pr = pd.DataFrame({'판매가격':[2000, 3000, 5000, 10000],
                          '판매량':[32, 53, 40, 25]},
                          index=['P1001', 'P1002', 'P1003', 'P1004'])
    df_pr.index.name = '제품번호'
    df_pr
Out:
                판매가격    판매량
        제품번호
        P1001       2000        32
        P1002       3000        53
        P1003       5000        40
        P1004       10000        25
```

```
In: file_name = 'C:/myPyCode/data/save_DataFrame_cp949.txt'
    df_pr.to_csv(file_name, sep=" ", encoding = "cp949")
```

```
In: !type C:\myPyCode\data\save_DataFrame_cp949.txt
Out: 제품번호 판매가격 판매량
     P1001 2000 32
     P1002 3000 53
     P1003 5000 40
     P1004 10000 25
```