14장 데이터 분석을 위한 패키지

14.1배열 데이터를 효과적으로 다루는 NumPy

■ NumPy는 파이썬으로 과학 연산을 쉽고 빠르게 할 수 있게 만든 패키지이다.
- https://www.numpy.org

■ NumPy를 이용하면 파이썬의 기본 데이터 형식과 내장 함수를 이용하는 것보다 다차원 배열 데이터를 효과적으로 처리할 수 있다.

14.1.1 배열 생성하기

■ NumPy는 파이썬의 내장 모듈이 아니므로 NumPy를 이용하려면 별도로 설치해야 한다. 하지만 아니콘다 배포판에는 NumPy가 포함돼 있어서 아나콘다를 설치할 때 NumPy도 설치된다.

(1) 시퀀스 데이터로부터 배열 생성

■ 시퀀스 데이터(seq_data)로 NumPy의 배열을 생성한다.
- 형식: arr_obj = np.array(seq_data)
- 08~11: NumPy에서 인자로 정수와 실수가 혼합돼 있을 때 모두 실수로 변환한다.
- 13: array()에 리스트 데이터를 직접 넣어서 배열 객체를 생성할 수 있다.
- 15: 다차원 배열도 생성할 수 있다.

[ch14_numpy/ex01_array.py]

```
01  import numpy as np
02
03  data1 = [0, 1, 2, 3, 4, 5]
04  a1 = np.array(data1)
05  print(a1)   # [0 1 2 3 4 5]
06  print(a1.dtype)  # int32
07
08  data2 = [0.1, 5, 4, 12, 0.5]
09  a2 = np.array(data2)
10  print(a2)   # [ 0.1  5.   4.  12.   0.5]
11  print(a2.dtype)  # float64
12
13  print(np.array([0.5, 2, 0.01, 8]))  # [0.5  2.   0.01 8.  ]
14
15  print(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]))
16  '''
17  [[1 2 3]
18   [4 5 6]
19   [7 8 9]]
20  '''
```

(2) 범위를 지정해 배열 생성

■ NumPy의 arrange()를 이용해 NumPy 배열을 생성한다.
- 형식: arr_obj = np.arange([start,] stop[, step])
- 04: step이 1인 경우에는 생략할 수 있다.

- 05: start가 0인 경우에는 start도 생략할 수 있다.
- 07: 12개의 숫자를 생성한 후 4x3 행렬을 만든다.
- 17: np.linespace(start, stop[, num])는 start부터 stop까지 num개의 NumPy 배열을 생성한다.
- 20: 0부터 π까지 동일한 간격으로 나눈 20개의 데이터를 생성한다.

[ch14_numpy/ex02_arange.py]

```
01  import numpy as np
02
03  print(np.arange(0, 10, 2))  # [0 2 4 6 8]
04  print(np.arange(1, 10))     # [1 2 3 4 5 6 7 8 9]
05  print(np.arange(5))         # [0 1 2 3 4]
06
07  b1 = np.arange(12).reshape(4, 3)
08  print(b1)
09  '''
10  [[ 0  1  2]
11   [ 3  4  5]
12   [ 6  7  8]
13   [ 9 10 11]]
14  '''
15  print(b1.shape) # (4, 3)
16
17  print(np.linspace(1, 10, 10))
18  # [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
19
20  print(np.linspace(0, np.pi, 20))
21  '''
22  [0.         0.16534698 0.33069396 0.49604095 0.66138793 0.82673491
23   0.99208189 1.15742887 1.32277585 1.48812284 1.65346982 1.8188168
24   1.98416378 2.14951076 2.31485774 2.48020473 2.64555171 2.81089869
25   2.97624567 3.14159265]
26   '''
```

(3) 특별한 형태의 배열 생성

■ 다음은 특별한 형태의 배열을 생성하는 방법이다.
- 03: zero() 함수로 모든 원소가 0이고 개수가 10개인 1차원 배열을 생성한다.
- 06: 3x4의 2차원 배열을 생성한다.
- 14,16: ones()는 모든 원소가 1인 다차원 배열을 생성한다.
- 24: 3x3 단위행렬(Identity matrix)를 생성한다.

[ch14_numpy/ex03_zeros.py]

```
01  import numpy as np
02
03  print(np.zeros(10))
04  # [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
05
06  print(np.zeros((3, 4)))
07  '''
08  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
09  [[0. 0. 0. 0.]
10   [0. 0. 0. 0.]
11   [0. 0. 0. 0.]]
12  '''
13
```

```
print(np.ones(5))   # [1. 1. 1. 1. 1.]

print(np.ones((3, 5)))
'''
[1. 1. 1. 1. 1.]
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
'''

print(np.eye(3))
'''
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
'''
```

(4) 배열의 데이터 타입 변환

■ NumPy 데이터의 형식

기호	의미
'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) objets
'S' 혹은 'a'	바이트 문자열, (byte) string
'U'	유니코드, Unicode

■ NumPy의 배열은 숫자뿐만 아니라 문자열도 원소로 가질 수 있다.
- 03: 문자열이 원소인 NumPy 배열을 생성한다.
- 07: NumPy 배열의 형 변환은 astype()으로 할 수 있다. 문자열을 원소로 갖는 NumPy 배열을 실수 타입으로 변환한다.
- 10: 'ndarray'으로 NumPy 배열의 데이터 타입을 확인한다.

[ch14_numpy/ex04_astype.py]

```
import numpy as np

print(np.array(['1.5', '0.62', '2', '3.14', '3.141592']))
# ['1.5' '0.62' '2' '3.14' '3.141592']

str_a1 = np.array(['1.567', '0.123', '5.123', '9', '8'])
num_a1 = str_a1.astype(float)
print(num_a1)
# [1.567 0.123 5.123 9.    8.   ]
print(str_a1.dtype)     # <U5
print(num_a1.dtype)     # float64

str_a2 = np.array(['1', '3', '5', '7', '9'])
num_a2 = str_a2.astype(int)
print(num_a2)           # [1 3 5 7 9]
print(str_a2.dtype)     # <U1
print(num_a2.dtype)     # int32

num_f1 = np.array([10, 21, 0.549, 4.75, 5.98])
```

```
num_i1 = num_f1.astype(int)
print(num_i1)          # [10 21  0  4  5]
print(num_f1.dtype)    # float64
print(num_i1.dtype)    # int32
```

(5) 난수 배열의 생성

■ random 모듈을 이용해 임의의 숫자인 난수(random number)를 생성한다.
- 03: rand() 함수는 0,1 사이의 실수로 난수 배열을 생성한다.
- 04: 2x3의 2차원 배열의 실수 난수를 생성한다.
- 21: randint() 함수는 지정한 범위에 해당하는 정수로 난수 배열을 생성한다.

[ch14_numpy/ex05_random.py]

```
import numpy as np

print(np.random.rand())     # 0.5686832723320677
print(np.random.rand(2, 3))
'''
[[0.70907696 0.52048631 0.22789206]
 [0.89651869 0.09436117 0.99870432]]
'''

print(np.random.rand(2, 3, 4))
'''
[[[0.03924296 0.54690974 0.3277407  0.11096887]
  [0.07716176 0.47734592 0.77689285 0.09760376]
  [0.21371426 0.45822489 0.88555773 0.27958194]]

 [[0.28183662 0.34660443 0.96422014 0.61493368]
  [0.84405469 0.70484702 0.06221535 0.82058264]
  [0.74525313 0.50875136 0.44468405 0.98379758]]]
'''

print(np.random.randint(10, size=(3, 4)))
'''
[[9 3 1 7]
 [6 3 8 4]
 [5 6 5 3]]
'''
```

14.1.2 배열의 연산

(1) 기본 연산

■ 배열의 형태(shape)가 같다면 덧셈과 뺄셈, 곱셈과 나눗셈 연산을 할 수 있다.

[ch14_numpy/ex06_operation.py]

```
import numpy as np

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

print(arr1 + arr2)      # [11 22 33 44]
print(arr1 - arr2)      # [ 9 18 27 36]
```

```
08  print(arr2 * 2)         # [2 4 6 8]
09  print(arr1 * arr2)      # [ 10  40  90 160]
10  print(arr1 / arr2)      # [10. 10. 10. 10.]
11  print(arr1 / (arr2 ** 2))
12  # [10.          5.          3.33333333  2.5       ]
13  print(arr1 > 20)        # [False False  True  True]
```

(2) 통계를 위한 연산

■ 배열의 합, 평균, ㅍ준 편차, 분산, 최솟값, 최댓값, 누적 합과 누적 곱 등 주로 통계에서 많이 사용하는 메소드가 있다.

[ch14_numpy/ex07_stat.py]

```
01  import numpy as np
02
03  arr3 = np.arange(5)
04  print(arr3)         # [0 1 2 3 4]
05  print([arr3.sum(), arr3.mean()])    # [10, 2.0]
06  print([arr3.std(), arr3.var()])
07  # [1.4142135623730951, 2.0]
08
09  print([arr3.min(), arr3.max()])     # [0, 4]
10  print([arr3.sum(), arr3.mean()])    # [10, 2.0]
11
12  arr4 = np.arange(1, 5)
13  print(arr4)             # [1 2 3 4]
14  print(arr4.cumsum())    # [ 1  3  6 10]
15  print(arr4.cumprod())   # [ 1  2  6 24]
```

(3) 행렬 연산

■ 참고
- 머신러닝 기초수학: https://mml-book.github.io/book/mml-book.pdf

■ NumPy는 배열의 단순 연산뿐만 아니라 선형 대수(Linear algebra)를 위한 행렬(2차원 배열) 연산도 지원한다.
- 03,10: 2x2 행렬을 만든다.
- 17,22: 행렬 A와 B 를 이용한 행렬 곱을 구한다.
- 28,33: 행렬 A의 전치 행렬(행렬을 주대각선을 기준으로 뒤집어 얻음)을 구한다.
- 39: 행렬 A의 역행렬을 구한다.
- 45: 행렬 A의 행렬식을 구한다.

[ch14_numpy/ex08_matrix.py]

```
01  import numpy as np
02
03  A = np.array([0, 1, 2, 3]).reshape(2, 2)
04  print(A)
05  '''
06  [[0 1]
07   [2 3]]
08  '''
09
10  B = np.array([3, 2, 0, 1]).reshape(2, 2)
```

```
print(B)
'''
[[3 2]
 [0 1]]
'''

print(A.dot(B))
'''
[[0 1]
 [6 7]]
'''
print(np.dot(A, B))
'''
[[0 1]
 [6 7]]
'''

print(np.transpose(A))
'''
[[0 2]
 [1 3]]
'''
print(A.transpose())
'''
[[0 2]
 [1 3]]
'''

print(np.linalg.inv(A))
'''
[[-1.5  0.5]
 [ 1.   0. ]]
'''

print(np.linalg.det(A))     # -2.0
```

14.1.3 배열의 인덱싱과 슬라이싱

(1) 배열의 인덱싱

■ 배열의 위치나 조건을 지정해 배열의 원소를 선택하는 것을 인덱싱(Indexing)이라 한다.

- 03: 1차원 배열을 생성한다.
- 05: 배열(a1)에서 위치 0의 원소를 선택한다.
- 38: 2차원 배열 a2에서 (0,0) 위치의 원소 10과 (2,1) 위치의 원소 80을 선택해 가져온다.

[ch14_numpy/ex09_indexing.py]

```
import numpy as np

a1 = np.array([0, 10, 20, 30, 40, 50])
print(a1)
print(a1[0])    # 0
print(a1[4])    # 40

a1[5] = 70
print(a1)           # [ 0 10 20 30 40 70]
print(a1[[1, 3, 4]])  # [10 30 40]

```

```
a2 = np.arange(10, 100, 10).reshape(3, 3)
print(a2)
'''
[[10 20 30]
 [40 50 60]
 [70 80 90]]
'''
print(a2[0, 2])     # 30

a2[2, 2] = 95
print(a2)
'''
[[10 20 30]
 [40 50 60]
 [70 80 95]]
'''

print(a2[1])    # [40 50 60]
a2[1] = np.array([45, 55, 65])
print(a2)
'''
[[10 20 30]
 [45 55 65]
 [70 80 95]]
'''

print(a2[[0, 2], [0, 1]])   # [10 80]

a = np.array([1, 2, 3, 4, 5, 6])
print(a[a > 3])         # [4 5 6]
print(a[(a % 2) == 0])  # [2 4 6]
```

(2) 배열의 슬라이싱

■ 배열의 범위를 지정해 배열의 원소를 선택하는 것을 슬라이싱(Slicing)이라고 한다.

[ch14_numpy/ex10_slicing.py]

```
import numpy as np

b1 = np.array([0, 10, 20, 30, 40, 50])
print(b1[1:4])      # [10 20 30]
print(b1[:3])       # [ 0 10 20]
print(b1[2:])       # [20 30 40 50]

b1[2:5] = np.array([25, 35, 45])
print(b1)           # [ 0 10 25 35 45 50]

b1[3:6] = 60
print(b1)           # [ 0 10 25 60 60 60]

b2 = np.arange(10, 100, 10).reshape(3,3)
print(b2)
'''
[[10 20 30]
 [40 50 60]
 [70 80 90]]
'''

print(b2[1:3, 1:3])
'''
```

```
[[50 60]
 [80 90]]
'''

print(b2[:3, 1:])
'''
[[20 30]
 [50 60]
 [80 90]]
'''

print(b2[1][0:2])       # [40 50]

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

14.2 구조적 데이터 표시와 처리에 강한 pandas

- 파이썬에서 데이터 분석과 처리를 쉽게 할 수 있게 도와주는 것이 바로 pandas 라이브러리이다.
  - https://pandas.pydata.org

- pandas는 NumPy를 기반으로 만들어졌지만 좀 더 복잡한 데이터 분석에 특화돼 있다.

14.2.1 구조적 데이터 생성하기

(1) Series를 활용한 데이터 생성

- NumPy처럼 pandas도 사용하려면 pandas를 불러와야 한다.
  - 01: pandas를 이용할 때 pandas 대신 pd를 이용할 수 있다.
  - 03: Series()를 이용하면 Series 형식의 구조적 데이터(라벨을 갖는 1차원 데이터)를 생성할 수 있다. Series()의 인자로는 스퀀스 데이터(리스트, 튜블, 딕셔너리)가 들어간다.
  - 51: 입력 인자로 딕셔너리 데이터를 입력하면 딕셔너리 데이터의 키(keys)와 값(values)이 각각 Series 데이터의 index와 values로 들어간다.

[ch14_numpy/ex11_series.py]

```
import pandas as pd

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

```
13
14  s1.index
15  print(s1.index)     # RangeIndex(start=0, stop=5, step=1)
16  print(s1.values)    # [10 20 30 40 50]
17
18  s2 = pd.Series(['a', 'b', 'c', 1, 2, 3])
19  print(s2)
20  '''
21  0    a
22  1    b
23  2    c
24  3    1
25  4    2
26  5    3
27  dtype: object
28  '''
29
30
31  s3 = pd.Series([np.nan, 10, 30])
32  print(s3)
33  '''
34  0     NaN
35  1    10.0
36  2    30.0
37  dtype: float64
38  '''
39
40  index_date = ['2018-10-07', '2018-10-08', '2018-10-09', '2018-10-10']
41  s4 = pd.Series([200, 195, np.nan, 205], index=index_date)
42  print(s4)
43  '''
44  2018-10-07    200.0
45  2018-10-08    195.0
46  2018-10-09      NaN
47  2018-10-10    205.0
48  dtype: float64
49  '''
50
51  s5 = pd.Series({'국어': 100, '영어': 95, '수학': 90})
52  print(s5)
53  '''
54  국어    100
55  영어     95
56  수학     90
57  dtype: int64
58  '''
59
```

(2) 날짜 자동 생성: date_range

- pandas에서 제공하는 date_range() 함수는 몇 가지 설정만 하면 원하는 날짜를 자동으로 생성하므로 날짜 데이터를 입력할 때 편리하다.

[ch14_numpy/ex12_date_range.py]

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

```
'''

print(pd.date_range(start='2019/01/01',end='2019.01.07'))
'''
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')
'''

print(pd.date_range(start='01-01-2019',end='01/07/2019'))
'''
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')
'''

print(pd.date_range(start='2019-01-01', periods = 7))
'''
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')
'''

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

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

print(pd.date_range(start='2019-01-01', periods = 12, freq = '2BM'))
'''
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')
'''

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

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

print(pd.date_range(start = '2019-01-01 08:00', periods = 10, freq='H'))
'''
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')
'''

print(pd.date_range(start = '2019-01-01 08:00', periods = 10, freq='BH'))
```

```
'''
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')
'''

print(pd.date_range(start = '2019-01-01 10:00', periods = 4, freq='30min'))
'''
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')
'''

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

(3) DataFrame을 활용한 데이터 생성

■ Series는 1차원 데이터를 생성한다. pandas에서는 표(table)와 같은 2차원 데이터 처리를 위해 DataFrame을 제공한다.

[ch14_numpy/ex13_dataframe.py]

```
import numpy as np
import pandas as pd

print(pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]]))
'''
   0  1  2
0  1  2  3
1  4  5  6
2  7  8  9
'''

data_list = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])
print(pd.DataFrame(data_list))
'''
    0   1   2
0  10  20  30
1  40  50  60
2  70  80  90
'''

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']
print(pd.DataFrame(data, index=index_date, columns=columns_list))
'''
2  70  80  90
             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
'''

table_data = {'연도': [2015, 2016, 2016, 2017, 2017],
              '지사': ['한국', '한국', '미국', '한국', '미국'],
              '고객 수': [200, 250, 450, 300, 500]}
print(table_data)
'''
{'연도': [2015, 2016, 2016, 2017, 2017], '지사': ['한국', '한국', '미국', '한
국', '미국'], '고객 수': [200, 250, 450, 300, 500]}
'''
print(pd.DataFrame(table_data))
'''
     연도  지사  고객 수
0  2015  한국   200
1  2016  한국   250
2  2016  미국   450
3  2017  한국   300
4  2017  미국   500
'''

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

print(df.index)     # RangeIndex(start=0, stop=5, step=1)
print(df.columns)   # Index(['연도', '지사', '고객 수'], dtype='object')
print(df.values)
'''
[[2015 '한국' 200]
 [2016 '한국' 250]
 [2016 '미국' 450]
 [2017 '한국' 300]
 [2017 '미국' 500]]
'''
```

14.2.2 데이터 연산

■ pandas의 Series()와 DataFrame()으로 생성한 데이터끼리는 사칙 연산을 할 수 있다.

[ch14_numpy/ex14_pandasOperation.py]

```
import pandas as pd

s1 = pd.Series([1, 2, 3, 4, 5])
s2 = pd.Series([10, 20, 30, 40, 50])
print(s1 + s2)
'''
0    11
1    22
2    33
```

```
3    44
4    55
dtype: int64
'''

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

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)
print(df1)
'''
   A   B    C
0  1  10  100
1  2  20  200
2  3  30  300
3  4  40  400
4  5  50  500
'''

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

print(df1 + df2)
'''
      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
'''

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)
print(df3)
'''
          봄     여름     가을     겨울
2012  256.5  770.6  363.5  139.3
```

```
  75  2013  264.3  567.5  231.2   59.9
  76  2014  215.9  599.8  293.1   76.9
  77  2015  223.2  387.1  247.7  109.1
  78  2016  312.8  446.2  381.6  108.1
  79  '''
  80
  81  print(df3.mean())
  82  '''
  83  봄     254.54
  84  여름    554.24
  85  가을    303.42
  86  겨울     98.66
  87  dtype: float64
  88  '''
  89
  90  print(df3.mean(axis=1))
  91  '''
  92  2012    382.475
  93  2013    280.725
  94  2014    296.425
  95  2015    241.775
  96  2016    312.175
  97  dtype: float64
  98  '''
  99
 100  print(df3.describe())
 101  '''
 102                  봄          여름          가을          겨울
 103  count    5.000000    5.000000    5.000000    5.000000
 104  mean   254.540000  554.240000  303.420000   98.660000
 105  std     38.628267  148.888895   67.358496   30.925523
 106  min    215.900000  387.100000  231.200000   59.900000
 107  25%    223.200000  446.200000  247.700000   76.900000
 108  50%    256.500000  567.500000  293.100000  108.100000
 109  75%    264.300000  599.800000  363.500000  109.100000
 110  max    312.800000  770.600000  381.600000  139.300000
 111  '''
```

14.2.3 데이터를 원하는 대로 선택하기

[ch14_numpy/ex15_selection.py]

```
  01  import pandas as pd
  02  import numpy as np
  03
  04  KTX_data = {'경부선 KTX': [39060, 39896, 42005, 43621, 41702, 41266, 32427],
  05              '호남선 KTX': [7313, 6967, 6873, 6626, 8675, 10622, 9228],
  06              '경전선 KTX': [3627, 4168, 4088, 4424, 4606, 4984, 5570],
  07              '전라선 KTX': [309, 1771, 1954, 2244, 3146, 3945, 5766],
  08              '동해선 KTX': [np.nan, np.nan, np.nan, np.nan, 2395, 3786, 6667]}
  09  col_list = ['경부선 KTX', '호남선 KTX', '경전선 KTX', '전라선 KTX', '동해선 KTX']
  10  index_list = ['2011', '2012', '2013', '2014', '2015', '2016', '2017']
  11
  12  df_KTX = pd.DataFrame(KTX_data, columns=col_list, index=index_list)
  13  print(df_KTX)
  14  '''
  15        경부선 KTX  호남선 KTX  경전선 KTX  전라선 KTX  동해선 KTX
  16  2011    39060     7313     3627      309      NaN
  17  2012    39896     6967     4168     1771      NaN
  18  2013    42005     6873     4088     1954      NaN
  19  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
'''

print(df_KTX.index)
'''
Index(['2011', '2012', '2013', '2014', '2015', '2016', '2017'], dtype='object')
'''
print(df_KTX.columns)
'''
Index(['경부선 KTX', '호남선 KTX', '경전선 KTX', '전라선 KTX', '동해선 KTX'],
dtype='object')
'''
print(df_KTX.values)
'''
[[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.]]
'''

print(df_KTX.head())
'''
      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
'''

print(df_KTX.tail())
'''
      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
'''

print(df_KTX.head(3))
'''
      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
'''

print(df_KTX.tail(2))
'''
      경부선 KTX  호남선 KTX  경전선 KTX  전라선 KTX  동해선 KTX
2016    41266    10622     4984     3945   3786.0
2017    32427     9228     5570     5766   6667.0
'''

print(df_KTX[1:2])
'''
      경부선 KTX  호남선 KTX  경전선 KTX  전라선 KTX  동해선 KTX
2012    39896     6967     4168     1771      NaN
'''
```

```
print(df_KTX[2:5])
'''
      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
'''

print(df_KTX.loc['2011'])
'''
경부선 KTX    39060.0
호남선 KTX     7313.0
경전선 KTX     3627.0
전라선 KTX      309.0
동해선 KTX        NaN
Name: 2011, dtype: float64
'''

print(df_KTX.loc['2013':'2016'])
'''
      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
'''

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

print(df_KTX['경부선 KTX']['2012':'2014'])
'''
2012    39896
2013    42005
2014    43621
Name: 경부선 KTX, dtype: int64
'''

print(df_KTX['경부선 KTX'][2:5])
'''
2013    42005
2014    43621
2015    41702
Name: 경부선 KTX, dtype: int64
'''

print(df_KTX.T)
'''
             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
'''

```

```
print(df_KTX)
'''
      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
'''

print(df_KTX[['동해선 KTX', '전라선 KTX', '경전선 KTX', '호남선 KTX', '경부선 KTX']])
'''
      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
'''
```

14.2.4 데이터 통합하기

(1) 세로 방향으로 통합하기

[ch14_numpy/ex16_append.py]

```
import pandas as pd
import numpy as np

df1 = pd.DataFrame({'Class1': [95, 92, 98, 100],
                    'Class2': [91, 93, 97, 99]})
print(df1)
'''
   Class1  Class2
0      95      91
1      92      93
2      98      97
3     100      99
'''

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

print(df1.append(df2))
'''
   Class1  Class2
0      95      91
1      92      93
2      98      97
3     100      99
0      87      85
```

```
32  1      89      90
33  '''
34
35  print(df1.append(df2, ignore_index=True))
36  '''
37     Class1  Class2
38  0      95      91
39  1      92      93
40  2      98      97
41  3     100      99
42  4      87      85
43  5      89      90
44  '''
45
46  df3 = pd.DataFrame({'Class1': [96, 83]})
47  print(df3)
48  '''
49     Class1
50  0      96
51  1      83
52  '''
53
54  print(df2.append(df3, ignore_index=True))
55  '''
56     Class1  Class2
57  0      87    85.0
58  1      89    90.0
59  2      96     NaN
60  3      83     NaN
61  '''
```

(2) 가로 방향으로 통합하기

[ch14_numpy/ex17_join.py]

```
01  import pandas as pd
02  import numpy as np
03
04  df1 = pd.DataFrame({'Class1': [95, 92, 98, 100],
05                      'Class2': [91, 93, 97, 99]})
06  print(df1)
07  '''
08     Class1  Class2
09  0      95      91
10  1      92      93
11  2      98      97
12  3     100      99
13  '''
14
15  df4 = pd.DataFrame({'Class3': [93, 91, 95, 98]})
16  print(df4)
17  '''
18     Class3
19  0      93
20  1      91
21  2      95
22  3      98
23  '''
24
25  print(df1.join(df4))
26  '''
27     Class1  Class2  Class3
```

```
0      95      91      93
1      92      93      91
2      98      97      95
3     100      99      98
'''

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)

print(df1a.join(df4a))
'''
   Class1  Class2  Class3
a      95      91      93
b      92      93      91
c      98      97      95
d     100      99      98
'''

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

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

(3) 특정 열을 기준으로 통합하기

■ 두 개의 DataFrame 데이터를 특정 열(key)을 기준으로 통합한다.

[ch14_numpy/ex18_merge.py]

```
import pandas as pd

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

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

```
'''
  판매월  제품C  제품D
0  1월  112   90
1  2월  141  110
2  3월  203  140
3  4월  134  170
'''

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

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

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

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

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

print(df_left.merge(df_right, how='outer', on='key'))
'''
  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
'''

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

```
84  '''
```

14.2.5 데이터 파일을 읽고 쓰기

- pandas는 표 형식의 데이터 파일을 DataFrame 형식의 데이터로 읽어오는 방법과 DataFrame 형식의 데이터를 표 형식으로 파일로 저장하는 편리한 방법을 제공한다.

(1) 표 형식의 데이터 파일을 읽기

- read_csv()는 기본적으로 각 데이터 필드가 콤마(,)로 구분된 CSV(comma-separated values) 파일을 읽는데 이용한다.

[ch14_numpy/ex19_read_csv.py]

```
01  import pandas as pd
02
03  print(pd.read_csv('D:\dev\workspace\python\ch14_numpy\sea_rain1.csv'))
04  '''
05       연도       동해       남해       서해       전체
06  0  1996  17.4629  17.2288  14.4360  15.9067
07  1  1997  17.4116  17.4092  14.8248  16.1526
08  2  1998  17.5944  18.0110  15.2512  16.6044
09  3  1999  18.1495  18.3175  14.8979  16.6284
10  4  2000  17.9288  18.1766  15.0504  16.6178
11  '''
12
13  print(pd.read_csv('D:\dev\workspace\python\ch14_numpy\sea_rain1.csv', index_col="연도"))
14  '''
15            동해       남해       서해       전체
16  연도
17  1996  17.4629  17.2288  14.4360  15.9067
18  1997  17.4116  17.4092  14.8248  16.1526
19  1998  17.5944  18.0110  15.2512  16.6044
20  1999  18.1495  18.3175  14.8979  16.6284
21  2000  17.9288  18.1766  15.0504  16.6178
22  '''
```

(2) 표 형식의 데이터를 파일로 쓰기

- pandas에서 제공하는 to_csv()를 이용해 DataFrame 형식의 데이터를 텍스트 파일로 저장한다.

[ch14_numpy/ex20_to_csv.py]

```
01  import pandas as pd
02
03  df_WH = pd.DataFrame({'Weight': [62, 67, 55, 74],
04                        'Height': [165, 177, 160, 180]},
05                       index=['ID_1', 'ID_2', 'ID_3', 'ID_4'])
06  df_WH.index.name = 'User'
07  print(df_WH)
08  '''
09        Weight  Height
10  User
```

```
ID_1      62     165
ID_2      67     177
ID_3      55     160
ID_4      74     180
'''

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

df_WH['BMI'] = bmi
print(df_WH)
'''
      Weight  Height        BMI
User
ID_1      62     165  22.773186
ID_2      67     177  21.385936
ID_3      55     160  21.484375
ID_4      74     180  22.839506
'''

df_WH.to_csv('D:\dev\workspace\python\ch14_numpy\save_DataFrame.csv')

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

file_name = 'D:\dev\workspace\python\ch14_numpy\save_DataFrame_utf-8.txt'
df_pr.to_csv(file_name, sep=" ", encoding="utf-8")
```