

Week1_머신러닝, 넘파이, 판다스

머신러닝(Machine Learning)

: 애플리케이션(프로그램)을 수정하지 않고도 데이터를 기반으로 패턴을 학습하고 결과를 예측하는 알고리즘 기법의 통칭

- 전통적 개발 방식 (코드 수정): 데이터의 양이 방대해짐 → 모든 케이스를 커버하는 코드를 짜기 어려워짐 (정확도↓)
- 가비지 인, 가비지 아웃: 머신러닝의 수행 결과는 데이터의 품질에 의존
- 머신러닝 모델 구축: 머신러닝 알고리즘 + 모델 파라미터 + 데이터 전처리

머신러닝의 분류

지도학습 (Supervised Learning)	비지도학습 (Un-supervised Learning)
분류, 회귀, 추천 시스템, 시각/음성 감지/인지, 텍스트 분석, NLP (자연어 처리)	클러스터링, 차원 축소, 강화학습

파이썬 머신러닝 주요 패키지 (+툴)

- 머신러닝 패키지: 사이킷런(Scikit-Learn) - 넘파이 기반 / 텐서플로(TensorFlow) / 케라스(Keras)
- 행렬/선형대수/통계 패키지: 넘파이(NumPy) - 행렬, 선형대수 / 사이파이(SciPy) - 자연과학, 통계
- 데이터 핸들링: 판다스(Pandas) - 2차원 데이터
- 시각화: 맷플롯립(Matplotlib), 시본(Seaborn)

** Jupyter Notebook: 아이파이썬(Interactive Python) 툴

*** **사이킷런**의 머신러닝 알고리즘에 입력하기 위한 데이터 처리 과정 → **Numpy**, **Pandas**의 프레임워크 이해 필요

넘파이

Numpy = Numerical Python

: 파이썬에서 선형대수 기반의 프로그램을 쉽게 만들 수 있도록 지원하는 라이브러리

- 빠른 배열 연산 속도
- C/C++ 등 저수준 언어 기반의 호환 API 제공 (*파이썬의 성능 제약 보완*)

넘파이 ndarray

넘파이 모듈 import한 뒤, np라는 약어로 모듈을 표현해준다.

```
import numpy as np
```

넘파이의 기반 데이터 타입 = **ndarray**

머신러닝 알고리즘과 데이터 세트 간 입출력과 변환을 수행할 때, 특정 차원의 데이터를 요구하는 경우가 있기 때문에 ndarray의 차원을 구분하는 것은 중요하다!



넘파이의 array() 함수로 다양한 인자를 ndarray로 변환하기

생성된 ndarray 배열의 shape 변수: ndarray의 (행, 열) 튜플 반환

```
[ ] array1 = np.array([1,2,3])
    print('array1 type:', type(array1))
    print('array1 array 형태:', array1.shape)
    # 1차원 array, 3개의 데이터
```

```
⇒ array1 type: <class 'numpy.ndarray'>
   array1 array 형태: (3,)
```

```
[ ] array2 = np.array([[1,2,3], [2,3,4]])
    print('array2 type:', type(array2))
    print('array2 array 형태:', array2.shape)
    # 2차원 array, 2*3 데이터
```

```
⇒ array2 type: <class 'numpy.ndarray'>
   array2 array 형태: (2, 3)
```

```
[ ] array3 = np.array([[1,2,3]])
    print('array3 type:', type(array3))
    print('array3 array 형태:', array3.shape)
    # 2차원 array, 1*3 데이터
```

```
⇒ array3 type: <class 'numpy.ndarray'>
   array3 array 형태: (1, 3)
```

ndarray.ndim으로 array의 차원 확인하기

```
[ ] print('array1: {0}차원, array2: {1}차원, array3: {2}차원'.format(array1.ndim, array2.ndim, array3.ndim))
```

```
⇒ array1: 1차원, array2: 2차원, array3: 2차원
```

ndarray의 데이터 타입

ndarray내의 데이터 값은 complex 타입을 포함한 모든 데이터 타입이 가능하다.

단, 하나의 ndarray 내의 데이터 타입은 같은 것만 가능하며, ndarray내의 데이터 타입은 **dtype** 속성으로 확인 가능하다.

```
list1 = [1,2,3]
print(type(list1))
array1 = np.array(list1)
print(type(array1))
print(array1, array1.dtype)
```

```
<class 'list'>
<class 'int'>
<class 'numpy.ndarray'>
[1 2 3] int64
```

여러 데이터 유형이 섞여 있는 리스트를 ndarray로 변환하면, 데이터 크기가 더 큰 데이터 타입으로 형 변환된다.

```
[ ] list2 = [1,2,'test']
array2 = np.array(list2)
print(array2, array2.dtype)
# int 데이터 -> unicode 문자열로 변환됨

list3 = [1,2, 3.0]
array3 = np.array(list3)
print(array3, array3.dtype)
# int 데이터 -> float64 형으로 변환됨
```

```
↔ ['1' '2' 'test'] <U21
[1. 2. 3.] float64
```

astype()을 통해 ndarray 내 데이터 값의 타입 변경하기

대용량 데이터의 ndarray를 만들 때, 메모리를 절약하기 위해 사용하는 경우가 있다.

```

▶ array_int = np.array([1,2,3])
array_float = array_int.astype('float64')
print(array_float, array_float.dtype)

array_int1 = array_float.astype('int64')
print(array_int1, array_int1.dtype)

array_float1 = np.array([1.1, 2.1, 3.1])
array_int2 = array_float1.astype('int64')
print(array_int2, array_int2.dtype)

```

```

↔ [1. 2. 3.] float64
   [1 2 3] int64
   [1 2 3] int64

```

**** float → int 변형하는 경우, 소수점 이하는 그냥 사라짐**

ndarray 생성 - arange, zeros, ones

특정 크기와 차원을 가진 ndarray를 연속값이나 0, 1로 초기화해 쉽게 생성해야 할 필요가 있는 경우

- `arange()`: array를 `range()`로 표현하는 것
(default 인자: `stop(==10)` - 0~9까지의 연속 숫자 값으로 된 1차원 ndarray 반환)

```

[ ] sequence_array = np.arange(10)
print(sequence_array)
print(sequence_array.dtype, sequence_array.shape)

```

```

↔ [0 1 2 3 4 5 6 7 8 9]
   int64 (10,)

```

- `zeros()`: shape 튜플 입력 -> 모든 값 0인 ndarray 반환(default type: float64)
- `ones()`: → 모든 값 1인 ndarray 반환 (default type: float64)

```
[ ] zero_array = np.zeros((3,2), dtype='int64')
    print(zero_array)
    print(zero_array.dtype, zero_array.shape)

    one_array = np.ones((3,2))
    print(one_array)
    print(one_array.dtype, one_array.shape)
```

```
⇒ [[0 0]
   [0 0]
   [0 0]]
   int64 (3, 2)
   [[1. 1.]
    [1. 1.]
    [1. 1.]]
   float64 (3, 2)
```

ndarray 차원, 크기 변경 - reshape()

reshape() 메소드는 ndarray를 특정 차원 및 크기로 변환한다.

1차원 ndarray → 2차원 ndarray 변환하기

```
[ ] array1 = np.arange(10)
    print('array1: \n', array1)
```

```
⇒ array1:
   [0 1 2 3 4 5 6 7 8 9]
```

```
[ ] array2 = array1.reshape(2,5)
    print('array2: \n', array2)
```

```
⇒ array2:
   [[0 1 2 3 4]
    [5 6 7 8 9]]
```

```
[ ] array3 = array1.reshape(5,2)
    print('array3: \n', array3)
```

```
⇒ array3:
   [[0 1]
    [2 3]
    [4 5]
    [6 7]
    [8 9]]
```

**** 변경 불가능한 사이즈를 지정하면 오류가 발생함**

```
array1.reshape(4,3)
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-7-a40469ec5825> in <cell line: 0>()
----> 1 array1.reshape(4,3)
```

```
ValueError: cannot reshape array of size 10 into shape (4,3)
```

reshape()의 인자를 -1로 적용하여 원래 ndarray와 호환되는 새로운 shape를 자동으로 지정받기

```
▶ array1 = np.arange(10)
print(array1)
array2 = array1.reshape(-1,5)
print('array2 shape:', array2.shape)
array3 = array1.reshape(5, -1)
print('array3 shape:', array3.shape)

# 확인용
print(array2)
print(array3)
```

```
⇄ [0 1 2 3 4 5 6 7 8 9]
array2 shape: (2, 5)
array3 shape: (5, 2)
[[0 1 2 3 4]
 [5 6 7 8 9]]
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
```

reshape()를 이용해 차원 변경하기


```

▶ array1 = np.arange(8)
array3d = array1.reshape((2,2,2))
print('array3d: \n', array3d.tolist())
print()

# 3차원 ndarray를 2차원 ndarray로 변환
array5 = array3d.reshape(-1,1)
print('array5: \n', array5.tolist())
print('array5 shape:', array5.shape)

print()

# 1차원 ndarray를 2차원 ndarray로 변환
array6 = array1.reshape(-1, 1)
print('array6: \n', array6.tolist())
print('array6 shape:', array6.shape)

```

```

⇒ array3d:
[[[0, 1], [2, 3]], [[4, 5], [6, 7]]]

array5:
[[0], [1], [2], [3], [4], [5], [6], [7]]
array5 shape: (8, 1)

array6:
[[0], [1], [2], [3], [4], [5], [6], [7]]
array6 shape: (8, 1)

```

ndarray의 데이터 세트 선택 - 인덱싱

: ndarray 내의 일부 데이터 세트나 특정 데이터만을 선택하기


1. 특정 데이터만 추출: 원하는 위치의 인덱스 값 지정

```

▶ # 1~9 1차원 ndarray 생성
array1 = np.arange(start=1, stop=10)
print('array:', array1)
|
# 인덱스 값 지정
value = array1[2]
print('value:', value)
print(type(value)) # ndarray 내의 데이터 값

# 마이너스 인덱스 값 지정
print('맨 뒤의 값:', array1[-1], ', 맨 뒤에서 두 번째 값:', array1[-2])

```


 array: [1 2 3 4 5 6 7 8 9]
 value: 3
 <class 'numpy.int64'>
 맨 뒤의 값: [9 1] , 맨 뒤에서 두 번째 값: 8

단일 인덱스를 이용해 데이터 값을 수정할 수 있다.

```

[ ] array1[0] = 9
    array1[8] = 0
    print('array1:', array1)

```

 array1: [9 2 3 4 5 6 7 8 0]

다차원 ndarray에서 단일 값 추출: (row, column) 인덱스로 접근

```

▶ array1d = np.arange(start=1, stop=10)
  array2d = array1d.reshape(3,3)
  print(array2d)

  print('(row=0, col=0) index 가리키는 값:', array2d[0,0])
  print('(row=0, col=1) index 가리키는 값:', array2d[0,1])
  print('(row=1, col=0) index 가리키는 값:', array2d[1,0])
  print('(row=2, col=2) index 가리키는 값:', array2d[2,2])

```

```

⇒ [[1 2 3]
   [4 5 6]
   [7 8 9]]
(row=0, col=0) index 가리키는 값: 1
(row=0, col=1) index 가리키는 값: 2
(row=1, col=0) index 가리키는 값: 4
(row=2, col=2) index 가리키는 값: 9

```

[+ 코드](#)

[+ 텍스트](#)

** ndarray에서는 axis 값을 이용해 행, 열 방향을 지정한다. row→axis=0, column→axis=1

2. 슬라이싱: ':' 기호를 이용해 연속한 데이터를 슬라이싱해서 추출할 수 있다.

```

[ ] array1 = np.arange(start=1, stop=10)
    array3 = array1[0:3]
    print(array3)
    print(type(array3))

```

```

⇒ [1 2 3]
   <class 'numpy.ndarray'>

```

```

[ ] array4 = array1[:3]
    print(array4)
    array5 = array1[3:]
    print(array5)
    array6 = array1[:]
    print(array6)

```

```

⇒ [1 2 3]
   [4 5 6 7 8 9]
   [1 2 3 4 5 6 7 8 9]

```

2차원 ndarray의 슬라이싱

```
array1d = np.arange(start=1, stop=10)
array2d = array1d.reshape(3, 3)
print('array2d:\n', array2d)

print('array2d[0:2, 0:2]\n', array2d[0:2,0:2])
print('array2d[1:3, 0:3]\n', array2d[1:3, 0:3])
print('array2d[1:3, :]\n', array2d[1:3, :])
print('array2d[:, :]\n', array2d[:, :])
print('array2d[:2, 1:]\n', array2d[:2,1:])
print('array2d[:2, 0]\n', array2d[:2,0])
```

```
array2d:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
array2d[0:2, 0:2]
[[1 2]
 [4 5]]
array2d[1:3, 0:3]
[[4 5 6]
 [7 8 9]]
array2d[1:3, :]
[[4 5 6]
 [7 8 9]]
array2d[:, :]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
array2d[:2, 1:]
[[2 3]
 [5 6]]
array2d[:2, 0]
[1 4]
```

2차원 ndarray에서 뒤에 오는 인덱스를 없애면 1차원 ndarray를 반환한다.

```
[ ] print(array2d[0]) # 1차원 row 0
    print(array2d[1]) # 1차원 row 1
    print('array2d[0] shape:', array2d[0].shape, 'array2d[1] shape:', array2d[1].shape)
```

```
array2d[0] shape: (3,) array2d[1] shape: (3,)
```

3. Fancy Indexing: 일정한 인덱싱 집합을 리스트 또는 ndarray 형태로 지정해 해당 위치의 ndarray 반환

```
[ ] array1d = np.arange(start=1, stop=10)
    array2d = array1d.reshape(3, 3)

    # 팬시 인덱싱 [0,1] + 단일 인덱싱 2
    # -> (0,2), (1,2) 적용
    array3 = array2d[[0,1], 2]
    print('array2d[[0,1], 2] =>', array3.tolist())

    # 팬시 인덱싱 [0,1] + 슬라이싱 0:2
    # -> ((0,0), (0,1)), ((1,0), (1,1)) 적용
    array4 = array2d[[0,1], 0:2]
    print('array2d[[0,1], 0:2] =>', array4.tolist())

    # 팬시 인덱싱 [0,1]
    # -> ((0,:), (1,:)) 적용
    array5 = array2d[[0,1]]
    print('array2d[[0,1]] =>', array5.tolist())
```

⇒ array2d[[0,1], 2] => [3, 6]
 array2d[[0,1], 0:2] => [[1, 2], [4, 5]]
 array2d[[0,1]] => [[1, 2, 3], [4, 5, 6]]

4. 불린 인덱싱: 특정 조건에 대한 불리언 값을 기반으로 **True**에 해당하는 인덱스 위치의 ndarray 반환

→ 조건 필터링 + 검색(인덱싱)

```
[ ] array1d = np.arange(start=1, stop=10)
    # [ ] 안에 array1d > 5 불린 인덱싱을 적용
    array3 = array1d[array1d > 5]
    print('array1d > 5 불린 인덱싱 결과 값 :', array3)
```

⇒ array1d > 5 불린 인덱싱 결과 값 : [6 7 8 9]

• 불리언 인덱싱의 과정

ndarray 객체에 조건식 할당

-> False, True로 이루어진 ndarray 객체 반환

-> Boolean ndarray 객체로 인덱싱하면 값에 해당하는 인덱스 위치의 데이터만 반환

```
[ ] array1d > 5
```

```
⇒ array([False, False, False, False, False,  True,  True,  True,  True])
```

```
[ ] # 원리
boolean_indexes = np.array([False, False, False, False, False,  True,  True,  True,  True])
array3 = array1d[boolean_indexes]
print('불린 인덱스로 필터링 결과 :', array3)
```

```
⇒ 불린 인덱스로 필터링 결과 : [6 7 8 9]
```

```
[ ] indexes = np.array([5,6,7,8]) # True값을 가진 인덱스를 저장
array4 = array1d[indexes] # array1d[[5,6,7,8]] 반환
print('일반 인덱스로 필터링 결과 :', array4)
```

```
⇒ 일반 인덱스로 필터링 결과 : [6 7 8 9]
```

행렬의 정렬 - np.sort(), ndarray.sort(), argsort()

행렬 정렬 - sort()

np.sort()의 경우 원 행렬은 그대로 유지한 채 원 행렬의 정렬된 행렬을 반환하며, ndarray.sort()는 원 행렬 자체를 정렬한 형태로 변환하며 반환 값은 None이다.

```
[ ] org_array = np.array([3,1,9,5])
print('원본 행렬:', org_array)

# np.sort()로 정렬 - 원본 행렬 유지, 정렬 행렬 반환
sort_array1 = np.sort(org_array)
print('np.sort() 호출 후 반환된 정렬 행렬:', sort_array1)
print('np.sort() 호출 후 원본 행렬:', org_array)

# ndarray.sort()로 정렬 - 원본 행렬 변환
sort_array2 = org_array.sort()
print('np.sort() 호출 후 반환된 정렬 행렬:', sort_array2)
print('np.sort() 호출 후 원본 행렬:', org_array)
```

```
⇒ 원본 행렬: [3 1 9 5]
np.sort() 호출 후 반환된 정렬 행렬: [1 3 5 9]
np.sort() 호출 후 원본 행렬: [3 1 9 5]
np.sort() 호출 후 반환된 정렬 행렬: None
np.sort() 호출 후 원본 행렬: [1 3 5 9]
```

내림차순으로 정렬하기

```
[ ] # 내림차순 정렬
sort_array1_desc = np.sort(org_array)[::-1]
print('내림차순으로 정렬:', sort_array1_desc)
```

➡ 내림차순으로 정렬: [9 5 3 1]

2차원 이상 행렬을 axis 축 값을 설정하여 정렬하기

```
▶ # 2차원 이상 행렬 정렬 - axis 축 값 설정
array2d = np.array([[8, 12], [7, 1]])

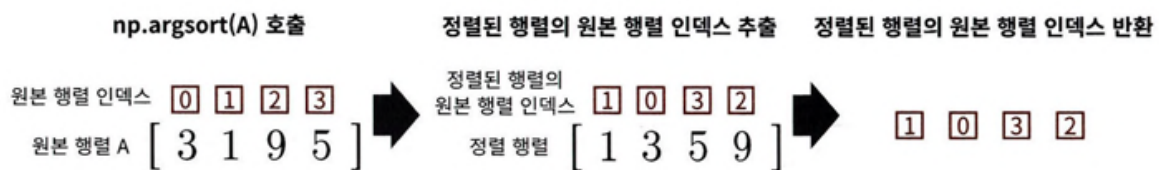
sort_array2d_axis0 = np.sort(array2d, axis=0)
print('로우 방향으로 정렬:\n', sort_array2d_axis0)

sort_array2d_axis1 = np.sort(array2d, axis=1)
print('칼럼 방향으로 정렬:\n', sort_array2d_axis1)
```

➡ 로우 방향으로 정렬:
[[7 1]
[8 12]]
칼럼 방향으로 정렬:
[[8 12]
[1 7]]

정렬된 행렬의 인덱스 반환 - argsort()

정렬 행렬의 원본 행렬 인덱스를 ndarray 형으로 반환



```
[ ] org_array = np.array([3,1,9,5])
sort_indices = np.argsort(org_array)
print(type(sort_indices))
print('행렬 정렬 시 원본 행렬의 인덱스:', sort_indices)
```

➡ <class 'numpy.ndarray'>
행렬 정렬 시 원본 행렬의 인덱스: [1 0 3 2]

np.argsort()로 내림차순 정렬하기

```
# 내림차순 정렬
sort_indices_desc = np.argsort(org_array[::-1])
print('행렬 내림차순 정렬 시 원본 행렬의 인덱스:', sort_indices_desc)
```

⇒ 행렬 내림차순 정렬 시 원본 행렬의 인덱스: [2 3 0 1]

넘파이의 메타 데이터 추출에 argsort()가 활용된다.

```
name_array = np.array(['John', 'Mike', 'Sarah', 'Kate', 'Samuel'])
score_array = np.array([78, 95, 84, 98, 88])

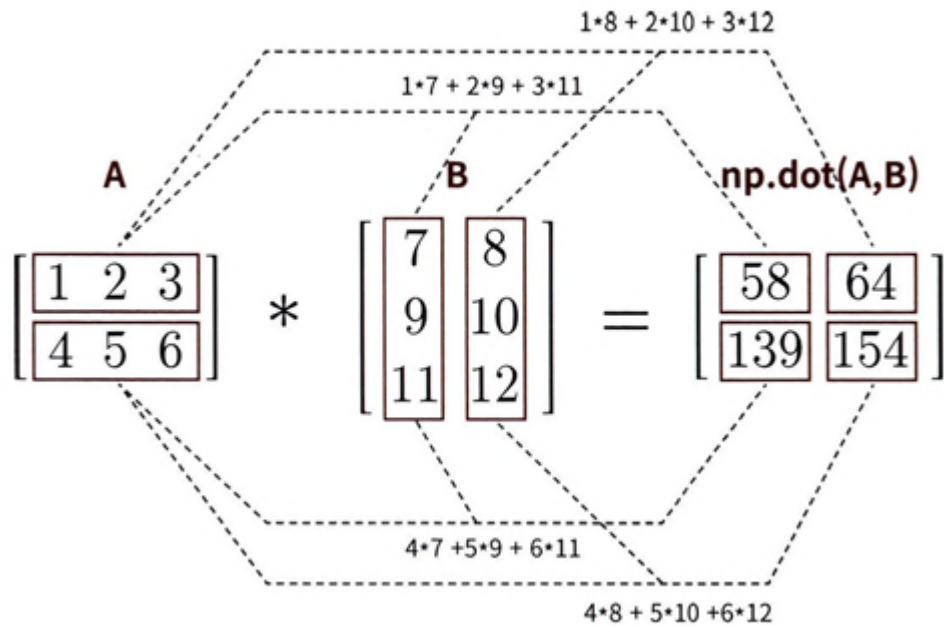
sort_indices_asc = np.argsort(score_array)
print('성적 오름차순 정렬 시 score_array의 인덱스:', sort_indices_asc)
print('성적 오름차순으로 name_array의 이름 출력:', name_array[sort_indices_asc])
```

⇒ 성적 오름차순 정렬 시 score_array의 인덱스: [0 2 4 1 3]
성적 오름차순으로 name_array의 이름 출력: ['John' 'Sarah' 'Samuel' 'Mike' 'Kate']

선형대수 연산 - 행렬 내적(dot())과 전치 행렬(transpose()) 구하기

행렬 내적(행렬 곱) - np.dot()

행렬 내적 = 왼쪽 행렬의 로우와 오른쪽 행렬의 칼럼의 원소들을 순차적으로 곱한 뒤 그 결과를 모두 더한 값



두 행렬의 행렬 내적을 dot()으로 구하기

```
[ ] A = np.array([[1,2,3], [4,5,6]])
    B = np.array([[7,8], [9,10], [11,12]])
    dot_product = np.dot(A, B)
    print('행렬 내적 결과:\n', dot_product)
```

⇒ 행렬 내적 결과:
[[58 64]
[139 154]]

전치 행렬 - np.transpose()

전치 행렬: 원 행렬에서 행과 열 위치를 교환한 원소로 구성된 행렬

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

transpose()로 전치 행렬 구하기

```
▶ A = np.array([[1,2], [3,4]])  
  transpose_mat = np.transpose(A)  
  print('A의 전치 행렬:\n', transpose_mat)
```

```
⇒ A의 전치 행렬:  
  [[1 3]  
   [2 4]]
```

판다스

Pandas

: (행x열)로 이루어진 2차원 데이터를 효율적으로 가공 및 처리할 수 있는 다양한 기능을 제공하는 라이브러리

- 고수준 API 제공
- DataFrame: 2차원 데이터를 담은 데이터 구조체 (판다스의 핵심 객체)
 - Index: 개별 데이터를 고유하게 식별하는 Key 값
 - Series: 칼럼이 하나인 데이터 구조체, Index를 key 값으로 가짐, DataFrame은 여러 Series로 이루어짐
- 여러 분리 문자(csv, tab 등)로 칼럼을 분리한 파일 → DataFrame으로 로딩할 수 있게 해 줌

파일을 DataFrame으로 로딩 - read_csv(), read_table(), read_fwf()

- read_csv(): 필드 구분 문자(Delimiter)가 ','
read_csv('경로+파일명', 'sep='\t') 가능
- read_table(): Delimiter가 '\t'
- read_fwf(): Fixed Width, 고정 길이 기반 칼럼 포맷을 로딩

pd.read_csv()로 파일명 인자로 들어온 파일을 로딩해 DataFrame 객체로 반환하기

```
[3] titanic_df=pd.read_csv(r'/content/drive/MyDrive/Euron Homework/titanic_train.csv')
print('titanic 변수 type:', type(titanic_df))
titanic_df # DataFrame의 모든 데이터 출력
```

↗ titanic 변수 type: <class 'pandas.core.frame.DataFrame'>

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

DataFrame 객체의 shape 변수

```
[ ] print('DataFrame 크기:', titanic_df.shape) #행x열
```

↗ DataFrame 크기: (891, 12)

DataFrame 객체의 메타 데이터 조회 - info(), describe()

```
[ ] titanic_df.info()
```

```
# DataFrame 타입  
# RangeIndex = DataFrame index의 범위 = 전체 row 개수 = 총 데이터 건수  
# Column 개수  
# Column별 데이터 타입 (object: string와 비슷)  
# Column별 non-null 값 개수  
# 전체 데이터의 타입 요약
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 891 entries, 0 to 890  
Data columns (total 12 columns):  
#   Column      Non-Null Count  Dtype  
---  -  
0   PassengerId  891 non-null    int64  
1   Survived     891 non-null    int64  
2   Pclass       891 non-null    int64  
3   Name         891 non-null    object  
4   Sex          891 non-null    object  
5   Age          714 non-null    float64  
6   SibSp        891 non-null    int64  
7   Parch        891 non-null    int64  
8   Ticket       891 non-null    object  
9   Fare         891 non-null    float64  
10  Cabin        204 non-null    object  
11  Embarked     889 non-null    object  
dtypes: float64(2), int64(5), object(5)  
memory usage: 83.7+ KB
```

describe(): column별 숫자형 데이터값의 n-percentile 분포도, 평균값, 최댓값, 최솟값 등
object 타입 Column은 출력에서 제외

```
[ ] titanic_df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Series = Index + 하나의 칼럼

dropna 인자: Null 값을 포함하여 개별 데이터 값의 건수를 계산할지 판단 - default=True

```
[ ] print('titanic_df 데이터 건수:', titanic_df.shape[0]) # shape[0] = row 개수 = 총 데이터 개수
    print('기본 설정인 dropna=True로 value_counts()')
    # value_counts(dropna=True)와 동일
    print(titanic_df['Embarked'].value_counts())
    print()
    # Null 값을 따로 빼서 value_counts() 적용
    print(titanic_df['Embarked'].value_counts(dropna=False))
```

```
⇒ titanic_df 데이터 건수: 891
   기본 설정인 dropna=True로 value_counts()
Embarked
S      644
C      168
Q       77
Name: count, dtype: int64

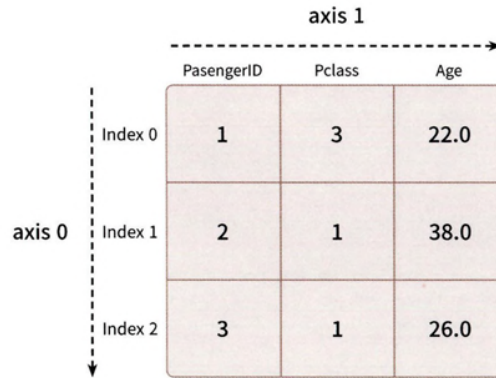
Embarked
S      644
C      168
Q       77
NaN       2
Name: count, dtype: int64
```

```
# 기존 칼럼 Series의 데이터를 이용해 새로운 칼럼 Series 만들기
titanic_df['Age_by_10'] = titanic_df['Age'] * 10
titanic_df['Family_No'] = titanic_df['SibSp'] + titanic_df['Parch'] + 1
titanic_df.head(3)
```

DataFrame 데이터 삭제 - drop()

DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise')

axis: 값에 따라 특정 column(axis=1) 또는 row(axis=0)를 드롭



〈 DataFrame에서의 axis 구분 〉

Titanic DataFrame에 추가한 'Age_0' 칼럼 삭제하기

```
titanic_drop_df = titanic_df.drop('Age_0', axis=1)
titanic_drop_df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Age_0	Age_by_10	Family_No
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	0	320.0	2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	C85	C	0	480.0	2
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	0	360.0	1

titanic_df.drop('Age_0', axis=1)을 수행한 결과 → titanic_drop_df 변수로 반환

axis=0으로 설정해 index 0,1,2 row 삭제하기

```
# DF 디스플레이 설정
pd.set_option('display.width', 1000)
pd.set_option('display.max_colwidth', 15)

print('#### before axis 0 drop ####')
print(titanic_df.head(3))

titanic_df.drop([0,1,2], axis=0, inplace=True)

print('#### after axis 0 drop ####')
print(titanic_df.head(3))
```

```
#### before axis 0 drop ####
  PassengerId  Survived  Pclass    Name     Sex  Age  SibSp  Parch    Ticket   Fare Cabin Embarked
0            1         0       3  Braund, Mr. ... male  22.0    1     0      A/5 21171   7.2500   NaN        S
1            2         1       1  Cumings, Mr. ... female 38.0    1     0      PC 17599  71.2833   C85        C
2            3         1       3  Heikkinen, ... female 26.0    0     0  STON/O2. 31...   7.9250   NaN        S

#### after axis 0 drop ####
  PassengerId  Survived  Pclass    Name     Sex  Age  SibSp  Parch  Ticket   Fare Cabin Embarked
3            4         1       1  Futrelle, M. ... female 35.0    1     0  113803  53.1000  C123        S
4            5         0       3  Allen, Mr. ... male 35.0    0     0  373450   8.0500   NaN        S
5            6         0       3  Moran, Mr. ... male  NaN    0     0  330877   8.4583   NaN        Q
```

→ 삭제 후 head(3)로 확인한 맨 앞 3개 데이터의 인덱스: 3, 4, 5

inplace: DataFrame의 데이터도 삭제할지 여부. default=False이므로 값을 지정하지 않으면 원본 DataFrame은 유지됨.

```
drop_result = titanic_df.drop(['Age_0', 'Age_by_10', 'Family_No'], axis=1, inplace=True)
print('inplace=True로 drop 후 반환된 값:', drop_result)
titanic_df.head(3)
```

inplace=True로 설정하면 None이 반환됨. 원본 DataFrame 변수에 반환 값을 할당받지 않도록 주의할 것!

Embarked	Age_0	Age_by_10
S	0	320.0

C	0	480.0
---	---	-------

S	0	360.0
---	---	-------

inplace=False

→ 원본

titanic_df:

'Age_0' 칼럼이
존재함

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

inplace=True → 원본 titanic_df: 칼럼이 삭제됨

Index 객체

: DataFrame, Series의 레코드를 고유하게 식별하는 객체
index 속성을 통해 Index 객체만 추출할 수 있다.

titanic_df DataFrame에서 Index 객체 추출하기

```
# 원본 파일 재로딩
titanic_df = pd.read_csv(r'/content/drive/MyDrive/Euron Homework/titanic_train.csv')
# Index 객체 추출
indexes = titanic_df.index
print(indexes)
# Index 객체를 실제 값 array로 반환
print('Index 객체 array 값:\n', indexes.values)
```

```
RangeIndex(start=0, stop=891, step=1)
Index 객체 array 값:
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163
164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182
183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201
202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220
221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239
240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258
259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277
278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296
297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315
316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334
335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353
354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372
373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391
392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410
411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429
430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448
449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467
468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486
487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505
506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524
525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543
544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562
563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581
582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600
601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619
620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638
639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657
658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676
677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695
696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714
715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733
734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752
753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771
772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790
791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809
810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828
829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847
848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866
867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885
886 887 888 889 890 891]
```

Index 객체의 values 속성으로 실제 값을 1차원 ndarray로 확인 가능
단일 값 반환 및 슬라이싱이 가능하다.

```
print(type(indexes.values))
print(indexes.values.shape)
print(indexes[:5].values)
print(indexes.values[:5])
print(indexes[6])
```

```
<class 'numpy.ndarray'>
(891,)
[0 1 2 3 4]
[0 1 2 3 4]
6
```

그러나 한 번 만들어진 DataFrame 및 Series의 Index 객체는 함부로 변경할 수 없다.

```
indexes[0]=5
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-33-33f4b187a6bd> in <cell line: 0>()
----> 1 indexes[0]=5

/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in __setitem__(self, key, value)
   5369     @final
   5370     def __setitem__(self, key, value) -> None:
-> 5371         raise TypeError("Index does not support mutable operations")
   5372
   5373     def __getitem__(self, key):

TypeError: Index does not support mutable operations
```


Series = Index + 실제 값 이지만 Index는 연산에서 제외된다.

```
series_fair=titanic_df['Fare']
print('Fair Series max 값:', series_fair.max())
print('Fair Series sum 값:', series_fair.sum())
print('sum() Fair Series:', sum(series_fair))
print('Fair Series + 3:¶n', (series_fair+3).head(3)) # Series 일괄 연산
```

```
Fair Series max 값: 512.3292
Fair Series sum 값: 28693.9493
sum() Fair Series: 28693.949299999967
Fair Series + 3:
0    10.2500
1    74.2833
2    10.9250
Name: Fare, dtype: float64
```

reset_index(): 인덱스를 새롭게 연속 숫자 형으로 할당, 기존 인덱스는 'index'라는 새로운 칼럼명으로 추가됨.

```
titanic_reset_df = titanic_df.reset_index(inplace=False)
titanic_reset_df.head(3)
```

	index	PassengerId	Survived	Pclass	Name	
0	0	1	0	3	Braund, Mr....	n
1	1	2	1	1	Cumings, Mr...	fer
2	2	3	1	3	Heikkinen, ...	fer

Series에 reset_index() 적용 → 기존 인덱스가 칼럼으로 추가되므로 DataFrame 객체가 반환된다.

```
print('### before reset_index ###')
value_counts = titanic_df['Pclass'].value_counts() # Pclass 데이터 값이 인덱스로 할당됨
print(value_counts)
print('value_counts 객체 변수 타입:', type(value_counts))

new_value_counts=value_counts.reset_index(inplace=False) # 인덱스 재할당
print('### After reset_index ###')
print(new_value_counts)
print('new_value_counts 객체 변수 타입:', type(new_value_counts))
```

```

#### before reset_index ####
Pclass
3    491
1    216
2    184
Name: count, dtype: int64
value_counts 객체 변수 타입: <class 'pandas.core.series.Series'>
#### After reset_index ####
   Pclass  count
0        3    491
1        1    216
2        2    184
new_value_counts 객체 변수 타입: <class 'pandas.core.frame.DataFrame'>

```

`reset_index(drop=True)` → 기존 인덱스가 새로운 칼럼으로 추가되지 않고 삭제되고, 새로운 칼럼이 추가되지 않으므로 타입 Series가 유지됨.

데이터 셀렉션 및 필터링

- numpy의 데이터 추출 및 필터링: '['] 연산자 내 단일 값 추출, 슬라이싱, 팬시 인덱싱, 불린 인덱싱
- pandas의 데이터 추출 및 필터링: `iloc[]`, `loc[]` 연산자

DataFrame의 '['] 연산자

DataFrame의 '['] 안에 들어갈 수 있는 것: 칼럼명 문자 및 리스트 객체, 인덱스로 변환 가능한 표현식

즉, 특정 값이 아닌 칼럼만 지정할 수 있는

'칼럼 지정 연산자'

- DataFrame 바로 뒤의 '['] 연산자는 numpy, Series의 '[']와 다르다
- DataFrame 바로 뒤의 '['] 내 입력값은 칼럼명 또는 리스트를 지정해 칼럼 지정 연산에 사용하거나 불린 인덱스 용도로 사용
- 슬라이싱 연산은 가능하지만, 사용하지 않는 게 좋음

titanic_df에서 칼럼명을 지정해 칼럼 데이터의 일부 추출하기

```
print('단일 칼럼 데이터 추출:\n', titanic_df['Pclass'].head(3))
print('\n여러 칼럼의 데이터 추출:\n', titanic_df[['Survived', 'Pclass']].head(3))
```

단일 칼럼 데이터 추출 :

```
0    3
1    1
2    3
Name: Pclass, dtype: int64
```

여러 칼럼의 데이터 추출 :

```
Survived  Pclass
0         0      3
1         1      1
2         1      3
```

```
print('[] 안에 숫자 index는 KeyError 오류 발생:\n', titanic_df[0])
```

```
-----
KeyError                                Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
    3804         try:
```

DataFrame의 [] 내에 숫자 값을 입력하면 오류가 발생하지만, 판다스의 Index 형태로 변환 가능한 표현식은 [] 내에 입력 가능 ex) 슬라이싱

슬라이싱을 이용해 원하는 결과 반환하기

```
titanic_df[0:2]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C

불린 인덱싱을 이용해 데이터 추출하기

```
titanic_df[titanic_df['Pclass'] == 3].head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250	NaN	S
2	3	1	3	Heikinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925	NaN	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.050	NaN	S

Pclass의 값이 3과 같은 데이터만 추출

- 불린 인덱싱: [] 연산자 내부에 조건식 삽입 → 반환된 데이터 중 True인 인덱스로 DataFrame의 데이터 추출

DataFrame.iloc[] 연산자

판다스에서 DataFrame의 로우나 칼럼을 지정하여 데이터를 선택할 수 있는 인덱싱 방식

- `iloc[]`: 위치 기반 기반 인덱싱 방식 → 행과 열 위치를 0을 출발점으로 하는 세로축, 가로축 좌표 정숫값으로 지정하는 방식

`iloc[]`로 데이터 추출하기 - 행과 열의 좌표 위치 값 입력

: 정숫값, 정수형의 슬라이싱, 팬시 리스트 값 입력 가능

```
data = {'Name': ['Chulmin', 'Eunkyung', 'Jinwoong', 'Soobeom'], 'Year': [2011, 2016, 2015, 2015], 'Gender': ['Male', 'Female', 'Male', 'Male']}
data_df = pd.DataFrame(data, index=['one', 'two', 'three', 'four'])
data_df
```

	Name	Year	Gender
one	Chulmin	2011	Male
two	Eunkyung	2016	Female
three	Jinwoong	2015	Male
four	Soobeom	2015	Male

문자열 타입의 인덱스를 가지고 있는 DataFrame 'data_df'

→ data_df의 1행, 1열의 데이터를 `iloc[행, 열]`를 이용해 추출하기

```
data_df.iloc[0, 0]
```

```
'Chulmin'
```

** `iloc[]`을 사용할 때 DataFrame의 인덱스 값이나 칼럼명을 입력하면 오류 발생

▼ [*인덱스*, *칼럼명*] 을 입력하는 경우

```
# 오류 발생 코드
data_df.iloc[0, 'Name']
```

```
ValueError                                Traceback (most recent call last)
~/usr/local/lib/python3.11/dist-packages/pandas/core/indexing.py in _validate_tuple_indexer(self, key)
    965         try:
--> 966             self._validate_key(k, i)
    967         except ValueError as err:
```

```
ValueError: Can only index by location with a [integer, integer slice (START point is INCLUDED, END point is EXCLUDED), listlike of integers, boolean array]
```

▼ [*칼럼명*, *인덱스*] 을 입력하는 경우

```
# 오류 발생 코드 2
data_df.iloc['one', 0]

-----
ValueError                                Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexing.py in _validate_tuple_indexer(self, key)
    965         try:
--> 966             self._validate_key(k, i)
    967         except ValueError as err:

-----
ValueError: Can only index by location with a [integer, integer slice (START point is INCLUDED, END point is EXCLUDED), listlike of integers, boolean array]
```

`iloc[:, -1]` : 맨 마지막 칼럼의 값 추출

`iloc[:, :-1]` : 처음부터 맨 마지막 칼럼을 제외한 모든 칼럼의 값 추출

```
print("\n 맨 마지막 칼럼 데이터[:, -1] \n", data_df.iloc[:, -1])
print("\n 맨 마지막 칼럼을 제외한 모든 데이터[:, :-1] \n", data_df.iloc[:, :-1])
```

```
맨 마지막 칼럼 데이터[:, -1]
one      Male1
two      Female
three    Male
four     Male
Name: Gender, dtype: object
```

```
맨 마지막 칼럼을 제외한 모든 데이터[:, :-1]
      Name  Year
one  Chulmin  2011
two  Eunkyung 2016
three Jinwoong 2015
four  Soobeom  2015
```

`iloc`는 명확한 위치 기반 인덱싱이 사용되어야 하므로 불린 인덱싱은 제공하지 않음

DataFrame loc[] 연산자

- `loc[]` : 명칭 기반 인덱싱 방식 → 행=인덱스 값, 열=칼럼 명칭 으로 지정

loc[인덱스값, 칼럼명] 형식으로 데이터 추출하기

```
data_df.loc['one', 'Name']
```

```
'Chulmin'
```

**** DataFrame마다 고유의 인덱스 타입이 있을 수 있기 때문에, 무심코 정수형 인덱스를 사용하지 않도록 주의해야 함**

```
data_df.loc[0, 'Name']
```

```
-----
KeyError                                Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
    3804         try:
-> 3805             return self._engine.get_loc(casted_key)
    3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 0
```

The above exception was the direct cause of the following exception:

**** loc[]는 명칭 기반 인덱싱이므로, [시작값:종료값]에서 종료값까지 포함한 데이터를 추출함**

```
print('위치기반 iloc slicing\n', data_df.iloc[0:1, 0], '\n')
print('명칭기반 loc slicing\n', data_df.loc['one':'two', 'Name'])
```

```
위치기반 iloc slicing
one    Chulmin
Name: Name, dtype: object
```

```
명칭기반 loc slicing
one    Chulmin
two    Eunkyung
Name: Name, dtype: object
```

명칭 기반 인덱싱과 위치 기반 인덱싱에서 슬라이싱의 차이

**** loc[]는 불린 인덱싱이 가능함**

```
data_df.loc[data_df.Year >= 2014]
```

	Name	Year	Gender
two	Eunkyung	2016	Female
three	Jinwoong	2015	Male
four	Soobeom	2015	Male

불린 인덱싱

불린 인덱싱은 [], loc[]에서 지원한다. iloc[]는 정수형 값만 지원하기 때문에, 불린 인덱싱이 불가능하다.

불린 인덱싱으로 승객 중 나이가 60세 이상인 데이터를 추출하기

```
titanic_df = pd.read_csv('/content/drive/MyDrive/Euron Homework/titanic_train.csv')
titanic_boolean = titanic_df[titanic_df['Age'] > 60]
print(type(titanic_boolean))
titanic_boolean
```

<class 'pandas.core.frame.DataFrame'>

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
33	34	0	2	Wheadon, Mr. Edward H	male	66.0	0	0
54	55	0	1	Ostby, Mr. Engelhart Cornelius	male	65.0	0	1
96	97	0	1	Goldschmidt, Mr. George B	male	71.0	0	0

- 60세 이상인 승객의 나이와 이름 추출

```
# 추출하고자 하는 칼럼이 두 개 이상이므로 [[]] 사용
titanic_df[titanic_df['Age']>60][['Name', 'Age']].head(3)
```

	Name	Age
33	Wheadon, Mr. Edward H	66.0
54	Ostby, Mr. Engelhart Cornelius	65.0
96	Goldschmidt, Mr. George B	71.0

- loc[]를 이용한 불린 인덱싱

```
titanic_df.loc[titanic_df['Age']>60, ['Name', 'Age']].head(3)
```

	Name	Age
33	Wheadon, Mr. Edward H	66.0
54	Ostby, Mr. Engelhart Cornelius	65.0
96	Goldschmidt, Mr. George B	71.0

- 나이가 60세 이상이고, 선실 등급이 1등급이며, 성별이 여성인 승객 데이터 추출하기
: 조건 연산자 사용

```
titanic_df[(titanic_df['Age']>60)&(titanic_df['Pclass']==1)&(titanic_df['Sex']=='female')]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
275	276	1	1	Andrews, Miss. Kornelia Theodosia	female	63.0	1	0	13502	77.9583	D7	S
829	830	1	1	Stone, Mrs. George Nelson (Martha Evelyn)	female	62.0	0	0	113572	80.0000	B28	NaN

: 개별 조건을 변수에 할당한 뒤, 변수를 결합하여 불린 인덱싱 수행

```
cond1 = titanic_df['Age']>60
cond2 = titanic_df['Pclass']==1
cond3 = titanic_df['Sex']=='female'
titanic_df[cond1&cond2&cond3]
```

정렬, Aggregation 함수, GroupBy 적용

DataFrame, Series의 정렬 - sort_values()

- 주요 입력 파라미터
 - by='칼럼명' 또는 by=[칼럼명 리스트] : 해당 칼럼(들)으로 정렬 수행
 - ascending=True : 오름차순 정렬 / ascending=False : 내림차순 정렬
 - inplace=True : 호출한 DataFrame의 정렬 결과 그대로 적용 (default: False)

titanic_df를 Name 칼럼으로 오름차순 정렬해 반환하기

```
titanic_sorted = titanic_df.sort_values(by='Name')
titanic_sorted.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
845	846	0	3	Abbing, Mr. Anthony	male	42.0	0	0	C.A. 5547	7.55	NaN	S
746	747	0	3	Abbott, Mr. Rossmore Edward	male	16.0	1	1	C.A. 2673	20.25	NaN	S
279	280	1	3	Abbott, Mrs. Stanton (Rosa Hunt)	female	35.0	1	1	C.A. 2673	20.25	NaN	S

titanic_df를 Pclass와 Name을 내림차순으로 정렬하기

```
titanic_sorted = titanic_df.sort_values(by=['Pclass', 'Name'], ascending=False)
titanic_sorted.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
868	869	0	3	van Melkebeke, Mr. Philemon	male	NaN	0	0	345777	9.5	NaN	S
153	154	0	3	van Billiard, Mr. Austin Blyler	male	40.5	0	2	A/5. 851	14.5	NaN	S
282	283	0	3	de Pelsmaeker, Mr. Alfons	male	16.0	0	0	345778	9.5	NaN	S

Aggregation 함수 적용

min(), max(), sum(), count() 등을 적용하기

DataFrame의 모든 칼럼에 count() 함수 적용하기

: DataFrame에서 바로 aggregation을 호출하면 모든 칼럼에 해당 aggregation을 적용함

** count(): Null 값은 제외하고 세아림

```
titanic_df.count()
```

0

PassengerId	891
Survived	891
Pclass	891
Name	891
Sex	891
Age	714
SibSp	891
Parch	891
Ticket	891
Fare	891
Cabin	204
Embarked	889

dtype: int64

특정 칼럼에 aggregation 함수 적용하기

DataFrame에 대상 칼럼들만 추출해 aggregation 적용

```
titanic_df[['Age', 'Fare']].mean()
```

0

Age	29.699118
Fare	32.204208

dtype: float64

groupby() 적용

- groupby(by='칼럼명') : 해당 칼럼을 기준으로 GroupBy된 DataFrame(GroupBy) 객체 반환

```
titanic_groupby = titanic_df.groupby(by='Pclass')
print(type(titanic_groupby))
```

```
<class 'pandas.core.groupby.generic.DataFrameGroupBy'>
```

이 상태에서 titanic_groupby를 프린트하면 객체의 주소값이 출력됨

Pclass를 제외한 모든 칼럼에 aggregation 함수 적용하기

```
titanic_groupby = titanic_df.groupby(by='Pclass').count()
titanic_groupby
```

	PassengerId	Survived	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
Pclass											
1	216	216	216	216	186	216	216	216	216	176	214
2	184	184	184	184	173	184	184	184	184	16	184
3	491	491	491	491	355	491	491	491	491	12	491

DataFrameGroupBy 객체에 특정 칼럼을 필터링한 뒤 aggregation 함수 적용하기

```
titanic_groupby = titanic_df.groupby('Pclass')[['PassengerId', 'Survived']].count()
titanic_groupby
```

```
<class 'pandas.core.frame.DataFrame'>
```

	PassengerId	Survived
Pclass		
1	216	216
2	184	184
3	491	491

agg()를 적용하여 특정 칼럼에 대한 여러 aggregation 함수를 적용하기

```
titanic_df.groupby('Pclass')['Age'].agg([max, min])
```

<ipython-input-31-b719d79acb8d>:1: FutureWarning: The pr
titanic_df.groupby('Pclass')['Age'].agg([max, min])
<ipython-input-31-b719d79acb8d>:1: FutureWarning: The pr
titanic_df.groupby('Pclass')['Age'].agg([max, min])

	max	min
Pclass		
1	80.0	0.92
2	70.0	0.67
3	74.0	0.42

교재에서 소개하는 코드

```
titanic_df.groupby('Pclass')['Age'].agg(['max', 'min'])
```

	max	min
Pclass		
1	80.0	0.92
2	70.0	0.67
3	74.0	0.42

최신 버전의 판다스에서는 aggregation 함수명을 문자열로 입력하는 것을 권장함

칼럼 별로 서로 다른 aggregation 함수를 적용하기

agg() 내에 딕셔너리 형태의 {'칼럼명': 'aggregation 함수명'} 입력

```
agg_format={'Age': 'max', 'SibSp': 'sum', 'Fare': 'mean'}
titanic_df.groupby('Pclass').agg(agg_format)
```

	Age	SibSp	Fare
Pclass			
1	80.0	90	84.154687
2	70.0	74	20.662183
3	74.0	302	13.675550

결손 데이터 처리하기

- 결손 데이터: 칼럼 값이 NULL인 경우 → NaN으로 표시. 머신러닝 알고리즘은 기본적으로 NaN 값을 처리하지 않으며, 각종 함수 연산 시 제외된다.
 - ex) 100개 데이터 중 10개가 NaN 값 → mean 값: 나머지 90개 데이터에 대한 평균

isna()로 결손 데이터 여부 확인하기

: 데이터가 NaN인지의 여부를 불리언 값으로 반환

```
titanic_df.isna().head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False

결손 데이터의 개수를 sum() 함수로 구하기

True, False는 내부적으로 숫자 1, 0으로 변환되므로 sum()을 호출해 결손 데이터의 개수를 구할 수 있다.

```
titanic_df.isna().sum()
```

	0
PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

dtype: int64

fillna()로 결손 데이터 대체하기

‘Cabin’ 칼럼의 NaN 값을 ‘C000’으로 대체하기

```
titanic_df['Cabin'] = titanic_df['Cabin'].fillna('C000')
titanic_df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	C000	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	C000	S

'Age' 칼럼의 NaN→평균 나이, 'Embarked' 칼럼의 NaN→'S'로 대체해 결손 데이터 처리하기

- fillna()를 호출한 뒤 실제 데이터 세트 값까지 변경하기
 1. fillna()를 이용해 반환 값을 다시 받기
 2. inplace=True 파라미터를 fillna()에 추가하기

```
titanic_df['Age']=titanic_df['Age'].fillna(titanic_df['Age'].mean())
titanic_df['Embarked']=titanic_df['Embarked'].fillna('S')
titanic_df.isna().sum()
```

```

0
PassengerId  0
Survived     0
Pclass       0
Name         0
Sex          0
Age          0
SibSp        0
Parch        0
Ticket       0
Fare         0
Cabin        0
Embarked     0

dtype: int64
```

apply lambda 식으로 데이터 가공

판다스는 apply 함수에 lambda 식을 결합해 DataFrame이나 Series의 레코드별로 데이터를 가공하는 기능을 제공한다. 칼럼에 일괄적으로 데이터 가공을 하는 것이 속도 면에서 더 빠르지만, 복잡한 데이터 가공이 필요할 경우 apply lambda를 이용함.

▼ 파이썬의 lambda 식

```
# 입력값의 제곱 값을 구해서 반환하는 함수
def get_square(a):
    return a**2

print('3의 제곱은:', get_square(3)) # 3의 제곱은: 9

# lambda → 함수의 선언과 함수 내의 처리를 한 줄의 식으로 변환하기
# lambda '입력 인자':'반환값'
lambda_square = lambda x : x**2
print('3의 제곱은:', lambda_square(3)) # 3의 제곱은: 9

# 여러 개의 값을 입력 인자로 사용하는 경우
a=[1,2,3]
squares=map(lambda x:x**2, a)
list(squares) # [1, 4, 9]
```

‘Name’ 칼럼의 문자열 개수를 별도의 칼럼인 ‘Name_len’에 생성하기

```
titanic_df['Name_len']=titanic_df['Name'].apply(lambda x:len(x))
titanic_df[['Name', 'Name_len']].head(3)
```

	Name	Name_len
0	Braund, Mr. Owen Harris	23
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	51
2	Heikkinen, Miss. Laina	22

if else 절을 사용해 나이가 15세 미만이면 ‘Child’, 그렇지 않으면 ‘Adult’로 구분하는 새로운 칼럼 ‘Child_Adult’ 생성하기

lambda 식 ‘:’ 기호의 오른쪽에 반환 값이 있어야 하기 때문에 if 조건식보다 반환값이 우선함. else는 else 식에 먼저 나오고 반환 값이 나중에 위치한다. if, else if, else 등...은 지원

하지 않음.

```
titanic_df['Child_Adult']=titanic_df['Age'].apply(lambda x:'Child' if x<=15 else 'Adult')
titanic_df[['Age', 'Child_Adult']].head(8)
```

	Age	Child_Adult
0	22.000000	Adult
1	38.000000	Adult
2	26.000000	Adult
3	35.000000	Adult
4	35.000000	Adult
5	29.699118	Adult
6	54.000000	Adult
7	2.000000	Child

나이가 15세 이하이면 Child, 15~60세 사이는 Adult, 61세 이상은 Elderly로 분류하는 'Age_Cat' 칼럼 생성하기

여러 조건식을 적용해야 하므로, else 다음의 ()안에 또다른 조건식을 포함함.

```
titanic_df['Age_cat']=titanic_df['Age'].apply(lambda x:'Child' if x<=15 else ('Adult' if x<=60 else 'Elderly'))
titanic_df['Age_cat'].value_counts()
```

	count
Age_cat	
Adult	786
Child	83
Elderly	22

dtype: int64

조건식이 많은 경우 함수를 통해 lambda 식 적용하기

나이에 따라 세분화된 분류를 수행하는 함수

```
def get_category(age):  
    cat = ''  
    if age <= 5: cat = 'Baby'  
    elif age <= 12: cat = 'Child'  
    elif age <= 18: cat = 'Teenager'  
    elif age <= 25: cat = 'Student'  
    elif age <= 35: cat = 'Young Adult'  
    elif age <= 60: cat = 'Adult'  
    else : cat='Elderly'  
    return cat
```

lambda 식에 위에서 생성한 get_category() 함수를 반환값으로 지정

```
# get_category(x)는 입력값으로 'Age' 칼럼 값을 받아서 해당하는 카테고리 반환  
titanic_df['Age_cat']=titanic_df['Age'].apply(lambda x:get_category(x))  
titanic_df[['Age', 'Age_cat']].head()
```

	Age	Age_cat
0	22.0	Student
1	38.0	Adult
2	26.0	Young Adult
3	35.0	Young Adult
4	35.0	Young Adult

