



# 파이썬 기반의 머신러닝과 생태계 이해

1조 – 김예나, 유연, 장인서

# 목차

---

**#01 NumPy**

**#02 Pandas Part 1.**

**#03 Pandas Part 2.**



# #1. 넘파이



# #1.1 넘파이 ndarray 개요

ndarray: 넘파이 기반의 데이터 타입  
→ 다차원 배열 생성, 연산 수행 가능

이름	설명	코드 예시	출력 예시
np.array	인자를 ndarray로 변환	array1 = np.array([1, 2, 3]) array2 = np.array([[1, 2, 3] , [4, 5, 6]])	[1, 2, 3] [[1, 2, 3], [2, 3, 4]]
.shape	ndarray의 크기, ndarray 배열의 차원	array1.shape array2.shape	(3, ) (2, 3)
.ndim	각 array의 차원	array1.ndim array2.ndim	1 2

# # 1.2 ndarray의 데이터 타입

ndarray 내 데이터값: 숫자, 문자열, 불

→ 숫자 : int, unsigned int, float, complex

## " ndarray 내 같은 데이터 타입만 가능! "

더 크거나 정밀한 데이터 타입으로 자동 바뀜

```
list1 = [1, 2, 3.0]
array = np.array(list1)
print(array, array.dtype)
```

```
[1. 2. 3.] float64
```

# # 1.2 ndarray의 데이터 타입

ndarray 내 데이터값: 숫자, 문자열, 불

→ 숫자 : int, unsigned int, float, complex

## " ndarray 내 같은 데이터 타입만 가능! "

더 크거나 정밀한 데이터 타입으로 자동 바뀜

```
list1 = [1, 2, 3.0]
array = np.array(list1)
print(array, array.dtype)
```

```
[1. 2. 3.] float64
```

**astype()** - 강제 변환

```
array1 = np.array([1, 2, 3.0])
print(array1, array1.dtype)
array_int = array1.astype(int)
print(array_int, array_int.dtype)
```

```
[1. 2. 3.] float64
[1 2 3] int32
```

## # 1.3 ndarray 편리하게 생성하기 - arange, zeros, ones

### arange()

- array를 range()로 표현
- 값을 순차적으로 ndarray의 데이터값으로 변환

```
sequence_array = np.arange(10)
print(sequence_array)
```

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

### zeros()

- 모든 값을 0으로 채운 shape을 가진 ndarray 반환

```
zero_array = np.zeros((3, 2), dtype='int32')
print(zero_array)
print(zero_array.dtype, zero_array.shape)
```

```
[[0 0]
 [0 0]
 [0 0]]
int32 (3, 2)
```

### ones()

- 모든 값을 1로 채운 shape을 가진 ndarray 반환

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

```
[[1. 1.]
 [1. 1.]
 [1. 1.]]
float64 (3, 2)
```

# # 1.4 ndarray 의 차원과 크기를 변경하는 reshape()

reshape() - ndarray를 특정 차원/크기로 변환

이름	설명	코드 예시	출력 예시
reshape()	ndarray를 원하는 행(row)과 열(column) 크기로 변환 가능	<pre>array1 = np.range(10) array2 = array1.reshape(2, 5)</pre>	<pre>[[0 1 2 3 4]  [5 6 7 8 9]]</pre>
reshape 오류	배열 크기가 일치하지 않음 → ValueError	<pre>array1.reshape(4, 3)</pre>	<pre>ValueError: cannot reshape array of size 10 into shape (4,3)</pre>



# # 1.4 ndarray 의 차원과 크기를 변경하는 reshape()

reshape() 인자로 -1을 적용

→ 원래 ndarray와 호환되는 shape으로 변환

이름	설명	코드 예시	출력 예시
reshape(-1, a) reshape(a, -1)	자동으로 호환 가능한 크기로 변환	array1 = np.range(10) array2 = array1.reshape(-1, 5)	[[0 1 2 3 4] [5 6 7 8 9]]
reshape(-1, a) reshape(a, -1) 오류	배열 크기가 일치하지 않음 → ValueError	array4 = array1.reshape(-1, 4)	ValueError: cannot reshape array of size 10 into shape (4)

## # 1.4 ndarray 의 차원과 크기를 변경하는 reshape()

### reshape(-1,1) 형태

→ 원본의 ndarray가 어떤 형태라도 2차원/여러 개의 행/하나의 열을 가진 ndarray로 변환됨 보장

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

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

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

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

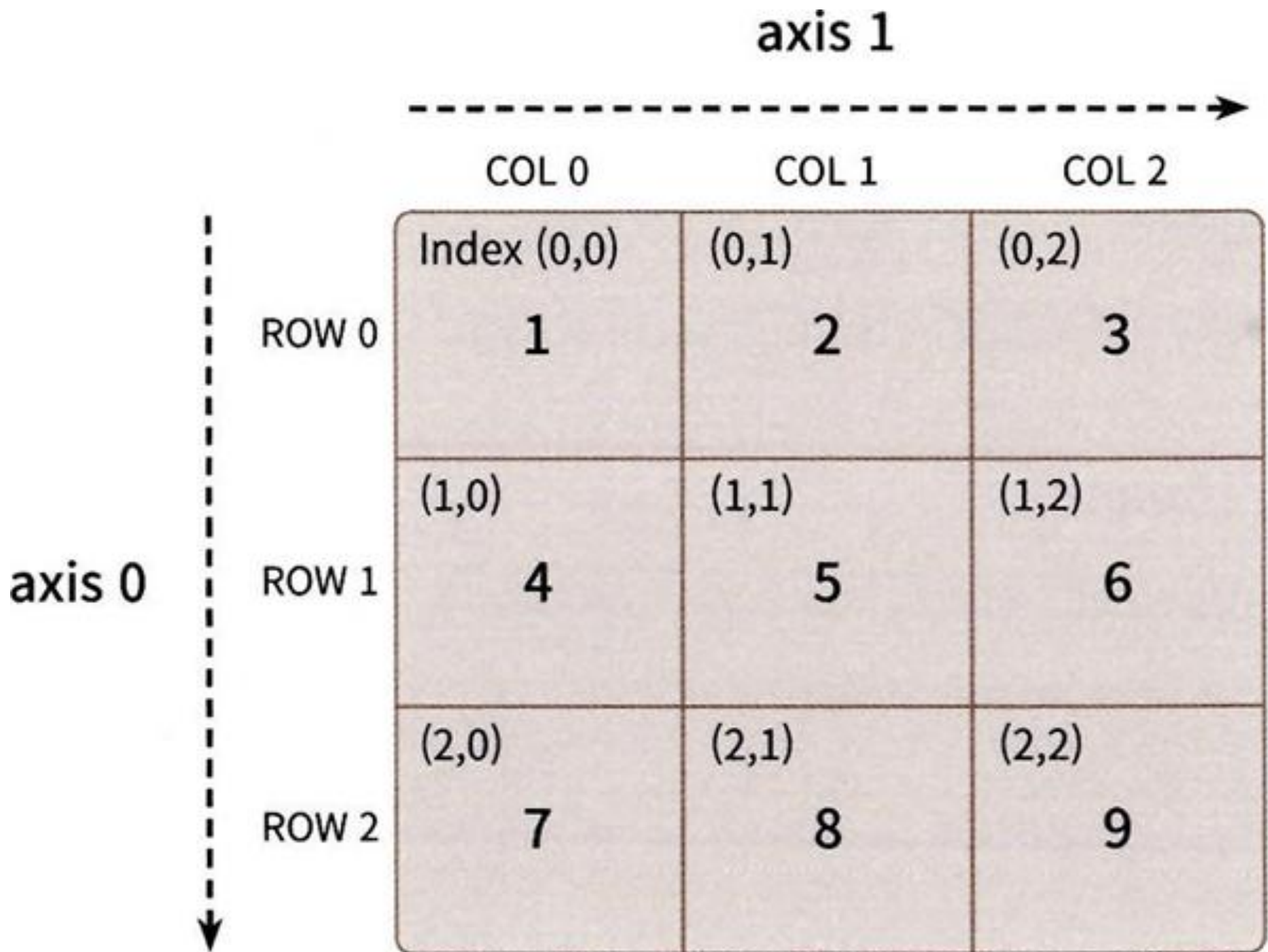
# #1.5 넘파이의 ndarray의 데이터 세트 선택하기 - 인덱싱(indexing)

array[index] - 1차원 배열에서 단일값 선택하기

인덱싱 방식	설명	코드 예시	출력 예시
양수 인덱스	n 번째 원소 선택	<pre>array1 = np.arange(start=1, stop=10) print(array1[2])</pre>	3
음수 인덱스	뒤에서 n번째 원소 선택	<pre>print(array1[-1])</pre>	9
값 변경	특정 원소 값 수정	<pre>array1[0] = 9 print(array1)</pre>	[9 2 3 4 5 6 7 8 9]

# # 1.5 넘파이의 ndarray의 데이터 세트 선택하기 - 인덱싱(indexing)

array[row, column] - 2차원 배열에서 단일 값 선택하기



row 방향 축 - axis 0  
column 방향 축 - axis 1

# # 1.5 넘파이의 ndarray의 데이터 세트 선택하기 - 인덱싱(indexing)

slicing - ‘:’ 사용하여 특정 범위 데이터 선택

차원	슬라이싱 방식	코드 예시	출력 예시
1차원 배열	[start:end] → start ~ end-1 생략 시 맨 처음 or 맨 끝	array1 = np.arange(1, 10)  print(array[0:3]) → 1, 2, 3 print(array[:3]) → 1, 2, 3 print(array[3:]) → 4, 5, 6, 7, 8, 9 print(array[:]) → 1, 2, 3, 4, 5, 6, 7, 8, 9	1, 2, 3  1, 2, 3  4, 5, 6, 7, 8, 9  1, 2, 3, 4, 5, 6, 7, 8, 9
2차원 배열	array[row_start:row_end, col_start:col_end]	array1 = np.arange(1, 10) array2 = array1.reshape(3, 3) print(array2[0:2, 0:2]) print(array2[1:3, 0:3])	[[1 2] [4 5]] [[4 5 6] [7 8 9]]

※ 2차원 배열에서 특정 행만 선택하면 1차원 배열 반환

## # 1.5 넘파이의 ndarray의 데이터 세트 선택하기 - 인덱싱(indexing)

### fancy indexing:

- 리스트나 배열을 이용해 원하는 인덱스 위치 지정하여 데이터 추출
- 여러 개의 행/열 한 번에 선택 가능

array2d[ [0,1], 2 ]

1	2	3
4	5	6
7	8	9

array2d[ [0, 1], 0:2 ]

1	2	3
4	5	6
7	8	9

array2d[ [0, 1] ]

1	2	3
4	5	6
7	8	9



## # 1.5 넘파이의 ndarray의 데이터 세트 선택하기 - 인덱싱(indexing)

Boolean indexing - 특정 [조건]을 만족하는 값들만 선택

→ for문 없이 한 줄 코드로 필터링 가능

```
# 1부터 9까지의 1차원 배열 생성
array1d = np.arange(start=1, stop=10)

# 조건: 5보다 큰 값만 선택
array3 = array1d[array1d > 5]
print("array1d > 5 불린 인덱싱 결과:", array3)
```

```
array1d > 5 불린 인덱싱 결과: [6 7 8 9]
```

## # 1.6 행렬의 정렬 — sort()와 argsort()

**np.sort() - 원본 유지, 정렬된 복사본 반환**

**ndarray.sort() - 원본 배열 자체를 정렬, 반환값 없음**

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

# np.sort() 사용 (원본 유지)
sorted_array1 = np.sort(org_array)
print("np.sort() 호출 후 반환된 정렬 배열:", sorted_array1)
print("np.sort() 호출 후 원본 배열:", org_array) # 변경 없음

# ndarray.sort() 사용 (원본 변경)
org_array.sort()
print("ndarray.sort() 호출 후 원본 배열:", org_array) # 원본이 정렬됨
```

```
원본 배열: [3 1 9 5]
np.sort() 호출 후 반환된 정렬 배열: [1 3 5 9]
np.sort() 호출 후 원본 배열: [3 1 9 5]
ndarray.sort() 호출 후 원본 배열: [1 3 5 9]
```

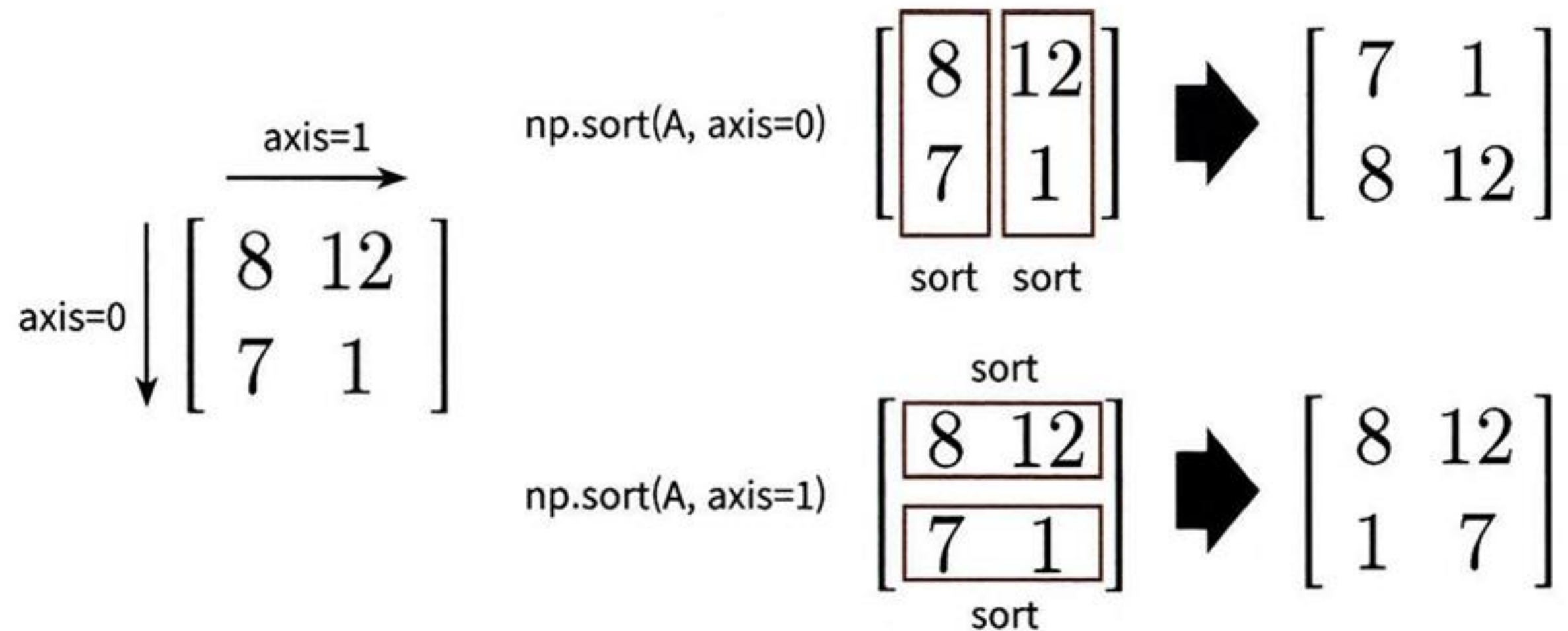


## # 1.6 행렬의 정렬 — `sort()`와 `argsort()`

(2차원 배열)

`axis=0` → 행(로우) 기준 정렬 (위에서 아래로 정렬)

`axis=1` → 열(칼럼) 기준 정렬 (왼쪽에서 오른쪽으로 정렬)



## # 1.6 행렬의 정렬 — sort()와 argsort()

**argsort()** - 정렬된 값의 원래 인덱스를 반환

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

# 성적 기준 정렬된 인덱스
sorted_indices_asc = np.argsort(score_array)

# 정렬된 성적순으로 이름 출력
sorted_names = name_array[sorted_indices_asc]
print("성적 오름차순 정렬 시 학생 이름:", sorted_names)
```

성적 오름차순 정렬 시 학생 이름: ['John' 'Sarah' 'Samuel' 'Mike' 'Kate']

오름차순(np.sort())을 내림차순으로 변경하려면 슬라이싱[::-1] 사용

# # 1.7 선형대수 연산 - 행렬 내적과 전치 행렬 구하기

A

123

456

\*

B

78

910

1112

=

np.dot(A,B)

5864

139154

※ 행렬 내적을 수행하려면 A의 "열 개수"와 B의 "행 개수"가 같아야 함

연산	코드	설명
행렬 내적 (Matrix Dot Product)	np.dot(A, B)	A의 행 × B의 열을 곱한 후 합산한 행렬 반환
전치 행렬 (Transpose Matrix)	np.transpose(A)	A의 행과 열을 서로 바꿈

# 판다스 Part 1.



# #판다스 시작 – 파일을 DataFrame으로 로딩, 기본 API

## #1 판다스 모듈을 임포트

: pandas를 pd로 alias해서 임포트한다.

```
import pandas as pd
```

## #2 데이터 파일을 DataFrame으로 로딩

: 판다스의 API를 이용해 다양한 포맷으로 된 파일을 DataFrame으로 로딩

### read\_csv()

- CSV 파일 포맷 변환을 위한 API
- sep에 해당 구분 문자를 입력, sep 인자 생략 시 자동으로 콤마로 할당(sep=',')
- filepath에 데이터 파일의 경로를 포함한 파일명을 입력

### read\_table()

- Read\_csv()와 비슷
- 필드 구분 문자(Delimiter)가 탭 문자("\t")

### read\_fwf

- Fixed Width : 고정 길이 기반의 칼럼 포맷을 DataFrame으로 로딩하기 위한 API

# #판다스 시작 – 파일을 DataFrame으로 로딩, 기본 API

## #3 DataFrame 정보 확인하기

- **DataFrame.head()**

: DataFrame의 맨 앞에 있는 N개의 로우를 반환 (디폴트 : 5개)

- **DataFrame.shape()**

: DataFrame의 행과 열을 튜플 형태로 반환 -> 행과 열 크기를 알 수 있음

- **DataFrame.info()**

: 총 데이터 건수, 데이터 타입, Null 건수를 알 수 있음

- **DataFrame.describe()**

: 칼럼별 숫자형 데이터값의 n-percentile 분포도, 평균값, 최댓값, 최솟값을 알 수 있음, 숫자형 칼럼의 분포도만 조사

- **DataFrame[ '칼럼명' ].value\_counts()**

: 지정된 칼럼의 데이터값 건수를 반환, 데이터 분포도 확인 가능

# #DataFrame과 리스트, 딕셔너리, 넘파이 ndarray 상호 변환

## # 넘파이 ndarray, 리스트, 딕셔너리를 DataFrame으로 변환하기

: 판다스 DataFrame 객체의 생성 인자 data는 리스트나 딕셔너리 또는 넘파이 ndarray를 입력받고, 생성 인자 columns는 컬럼명 리스트를 입력받아서 쉽게 DataFrame을 생성할 수 있음

DataFrame은 기본적으로 행과 열을 가지는 2차원 데이터 -> 2차원 이하의 데이터들만 DataFrame으로 변환 가능

# #DataFrame과 리스트, 딕셔너리, 넘파이 ndarray 상호 변환

## 1) 1차원 형태의 리스트, 넘파이 ndarray

- 1차원 데이터 -> 칼럼 1개 : 칼럼명 'col1'
- 칼럼명이 1개만 필요하다는 것!

```
import pandas as pd
import numpy as np

col_name1=['col1']
list1 = [1,2,3]
array1 = np.array(list1)
print('array1 shape:', array1.shape)
#리스트를 이용해 DataFrame 생성
df_list1 = pd.DataFrame(list1, columns=col_name1)
print('1차원 리스트로 만든 DataFrame:\n', df_list1)
#넘파이 ndarray를 이용해 DataFrame 생성
df_array1 = pd.DataFrame(array1, columns=col_name1)
print('1차원 ndarray로 만든 DataFrame:\n', df_array1)
```



```
array1 shape: (3,)
1차원 리스트로 만든 DataFrame:
   col1
0      1
1      2
2      3
1차원 ndarray로 만든 DataFrame:
   col1
0      1
1      2
2      3
```



# #판다스 시작 – 파일을 DataFrame으로 로딩, 기본 API

## 2) 2차원 형태의 데이터

- 2행 3열의 형태의 리스트와 ndarray -> 컬럼명 3개 필요

```
# 3개의 컬럼명이 필요함.  
col_name2=['col1','col2','col3']  
# 2행x3열 형태의 리스트와 ndarray 생성한 뒤 이를 DataFrame으로 변환.  
list2 = [[1, 2, 3],  
          [11, 12, 13]]  
array2 = np.array(list2)  
print('array2 shape:', array2.shape)  
df_list2 = pd.DataFrame(list2, columns=col_name2)  
print('2차원 리스트로 만든 DataFrame : \n', df_list2)  
df_array2 = pd.DataFrame(array2, columns=col_name2)  
print('2차원 ndarray로 만든 DataFrame : \n', df_array2)
```



```
array2 shape: (2, 3)  
2차원 리스트로 만든 DataFrame :  
   col1  col2  col3  
0     1     2     3  
1    11    12    13  
2차원 ndarray로 만든 DataFrame :  
   col1  col2  col3  
0     1     2     3  
1    11    12    13
```

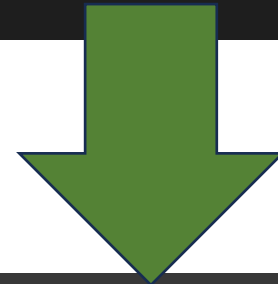
# #판다스 시작 – 파일을 DataFrame으로 로딩, 기본 API

## 3) 딕셔너리

: 딕셔너리의 키(key)는 컬럼명, 딕셔너리의 값(value)은 키에 해당하는 컬럼 데이터로 변환됨

-> 키의 경우-문자열, 값의 경우-리스트(or ndarray) 형태로 딕셔너리를 구성함

```
#Key는 문자열 컬럼명으로 매핑, Value는 리스트 형(또는 ndarray) 컬럼 데이터로 매핑
dict = {'col1':[1, 11], 'col2':[2, 22], 'col3':[3, 33]}
df_dict = pd.DataFrame(dict)
print('딕셔너리로 만든 DataFrame : \n', df_dict)
```



딕셔너리로 만든 DataFrame :

	col1	col2	col3
0	1	2	3
1	11	22	33

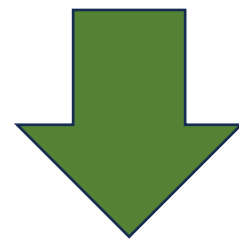
# #DataFrame과 리스트, 딕셔너리, 넘파이 ndarray 상호 변환

# DataFrame을 넘파이 ndarray, 리스트, 딕셔너리로 변환하기

## 1) DataFrame -> ndarray

: DataFrame 객체의 values를 이용

```
#DataFrame을 ndarray로 변환
array3 = df_dict.values
print('df_dict.values 타입:', type(array3), 'df_dict.values shape:', array3.shape)
print(array3)
```



```
df_dict.values 타입: <class 'numpy.ndarray'> df_dict.values shape: (2, 3)
[[ 1  2  3]
 [11 22 33]]
```

# #DataFrame과 리스트, 딕셔너리, 넘파이 ndarray 상호 변환

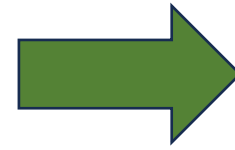
## # DataFrame을 넘파이 ndarray, 리스트, 딕셔너리로 변환하기

### 2) DataFrame -> 리스트, 딕셔너리

- 리스트로의 변환 : values로 얻은 ndarray에 tolist() 호출
- 딕셔너리로의 변환 : DataFrame 객체의 to\_dict() 메서드 호출  
-> 인자로 'list' 입력 시 딕셔너리의 값이 리스트형으로 반환됨

```
# DataFrame을 리스트로 변환
list3 = df_dict.values.tolist()
print('df_dict.values.tolist() 타입:', type(list3))
print(list3)
```

```
# DataFrame을 딕셔너리로 변환
dict3 = df_dict.to_dict('list')
print('\n df_dict.to_dict() 타입:', type(dict3))
print(dict3)
```



```
df_dict.values.tolist() 타입: <class 'list'>
[[1, 2, 3], [11, 22, 33]]

df_dict.to_dict() 타입: <class 'dict'>
{'col1': [1, 11], 'col2': [2, 22], 'col3': [3, 33]}
```

# #DataFrame의 칼럼 데이터 세트 생성과 수정

## - 칼럼 데이터 세트 생성과 수정

: [ ] 연산자 이용 -> DataFrame [ ] 내에 새로운 칼럼명을 입력하고 값을 할당

```
titanic_df['Age_0']=0  
titanic_df.head(3)
```



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

-> 새로운 칼럼명 'Age\_0' 으로 모든 데이터값이 0으로 할당된 Series가 기존 DataFrame에 추가됨

# #DataFrame의 칼럼 데이터 세트 생성과 수정

## - 칼럼 데이터 세트 생성과 수정

: [ ] 연산자 이용 -> DataFrame [ ] 내에 새로운 칼럼명을 입력하고 값을 할당

```
titanic_df['Age_by_10'] = titanic_df['Age']*10
titanic_df['Family_No'] = titanic_df['SibSp'] + titanic_df['Parch']+1
titanic_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	220.0	2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	0	380.0	2
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	0	260.0	1



업데이트를 원하는 칼럼 Series를 DataFrame [ ] 내에 칼럼명으로 입력한 뒤에 값을 할당

```
titanic_df['Age_by_10'] = titanic_df['Age_by_10'] + 100
titanic_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

# #DataFrame 데이터 삭제

## -Drop()

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

- axis 값에 따라서 특정 칼럼 or 특정 행을 드롭(drop)
  - 2차원 ndarray 구성 시의 axis 0, axis 1 -> axis 0은 row 방향 축, axis 1은 column 방향 축
- ⇒ drop() 메서드에 axis=1을 입력하면 column 축 방향으로 드롭 / labels에 오는 값 - 인덱스
- ⇒ drop() 메서드에 axis=0을 입력하면 row 축 방향으로 드롭

axis 1			
	PasengerID	Pclass	Age
Index 0	1	3	22.0
Index 1	2	1	38.0
Index 2	3	1	26.0

〈 DataFrame에서의 axis 구분 〉

# #DataFrame 데이터 삭제

## Drop()

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

- axis 값에 따라서 특정 칼럼 or 특정 행을 드롭(drop)
  - 2차원 ndarray 구성 시의 axis 0, axis 1 -> axis 0은 row 방향 축, axis 1은 column 방향 축
- ⇒ drop() 메서드에 axis=1을 입력하면 column 축 방향으로 드롭 / labels에 오는 값 - 인덱스
- ⇒ drop() 메서드에 axis=0을 입력하면 row 축 방향으로 드롭

```
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_by_10	Family_No
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	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	480.0	2
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	360.0	1



# #DataFrame 데이터 삭제

## Drop()

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

- Inplace = False -> 자기 자신의 DataFrame의 데이터는 삭제 X, 삭제된 결과 DataFrame 반환 (디폴트값이 False)
- Inplace = True -> 자기 자신의 DataFrame의 데이터를 삭제

여러 개의 칼럼 삭제 : 리스트 형태로 삭제하고자 하는 칼럼명을 입력해 labels 파라미터로 입력

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

inplace=True 로 drop 후 반환된 값: None

	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

titanic\_df의 'Age\_0',  
'Age\_by\_10',  
'Family\_No' 칼럼을 모두  
삭제

drop() 시 inplace=True로 설정하면 반환 값이 None(아무 값도 아님)이 됨

# #Index 객체

- RDBMS의 PK(Primary Key)와 유사하게 DataFrame, Series의 레코드를 고유하게 식별하는 객체
- DataFrame, Series에서 Index 객체만 추출하려면 `DataFrame.index` 또는 `Series.index` 속성을 통해 가능

```
# Index 객체 추출
indexes = titanic_df.index
print(indexes)
# Index 객체를 실제 값 array로 변환
print('Index 객체 array값:\n', indexes.values)
```

RangeIndex(start=0, stop=891, step=1)  
Index 객체 array값:  
[ 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

# #Index 객체

- 식별성 데이터를 1차원 array로 가짐
- 한번 만들어진 DataFrame 및 Series의 Index 객체는 함부로 변경 불가

## reset\_index()

- 새롭게 인덱스를 연속 숫자 형으로 할당, 기존 인덱스는 'index' 라는 새로운 칼럼 명으로 추가
- 연속된 int 숫자형 데이터가 아닐 경우에 다시 이를 연속 int 숫자형 데이터로 만들 때 주로 사용

# 판다스 Part 2.



# # 데이터 셀렉션 및 필터링

## # 넘파이 [ ] 연산자

- 단일 값 추출, 슬라이싱, 팬시 인덱싱, 불린 인덱싱을 통해 데이터 추출
- 행과 열의 위치, 슬라이싱 범위 지정



## # 판다스 [ ] 연산자

- `iloc[ ]`, `loc[ ]` 연산자를 통해 동일한 작업 수행
  - `DataFrame[ ]`: 칼럼명 문자 또는 인덱스로 변환 가능한 표현식
- \* 칼럼만 지정할 수 있는 '칼럼 지정 연산자'로 이해하기!

# # 데이터 선택 및 필터링

- 리스트 객체를 이용하여 여러 칼럼의 데이터 추출

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

여러 칼럼의 데이터 추출:

	Survived	Pclass
0	0	3
1	1	1
2	1	3

오류) 칼럼명이 아닌 숫자 값을 입력할 경우

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

File "<ipython-input-72-ddd1c2341f67>", line 1

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

SyntaxError: incomplete input

- **불린 인덱싱**

```
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....	male	22.0	1	0	A/5 21171	7.250	NaN	S
2	3	1	3	Heikkinen, ...	female	26.0	0	0	STON/O2. 31...	7.925	NaN	S
4	5	0	3	Allen, Mr. ...	male	35.0	0	0	373450	8.050	NaN	S

인덱스 형태로 변환 가능한 슬라이싱 사용

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

	PassengerId	Survived	Pclass	Name	Sex
0	1	0	3	Braund, Mr....	male
1	2	1	1	Cumings, Mr...	female

\* 위치 기반 or 라벨 기반 혼동 가능성 0

# # 데이터 셀렉션 및 필터링

## - `iloc[ ]` 연산자

- 위치(location) 기반 인덱싱
- 정숫값 또는 정수형의 슬라이싱, 팬시 리스트 값 입력
- 불린 인덱싱 제공 X <- 명확한 위치 기반 인덱싱이 사용되어야 하기 때문!!

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

'Chulmin'

오류)

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

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

행과 열의 좌표 위치에 해당하는 값 입력

열 위치에 위치 **정숫값이 아닌** 칼럼 명칭 입력

행 위치에 위치 **정숫값이 아닌** 인덱스 값 입력

\* **위치 기반 인덱싱**이란?

행과 열 위치를 세로축과 가로축 좌표 정숫값으로 지정하는 방식

		0	1	2
		Name	Year	Gender
0	<b>one</b>	Chulmin	2011	Male
1	<b>two</b>	Eunkyoung	2016	Female
2	<b>three</b>	Jinwoong	2015	Male
3	<b>four</b>	Soobeom	2015	Male

# # 데이터 셀렉션 및 필터링

- 정수 슬라이싱

		0 Name	1 Year	2 Gender
0	one	Chulmin	2011	Male
1	two	Eunkyoung	2016	Female
2	three	Jinwoong	2015	Male
3	four	Soobeom	2015	Male

(1) `data_df.iloc[0:2, [0,1]]`

	Name	Year
one	Chulmin	2011
two	Eunkyoung	2016

(2) `data_df.iloc[0:2,0:3]`

	Name	Year	Gender
one	Chulmin	2011	Male
two	Eunkyoung	2016	Female
three	Jinwoong	2015	Male

(3) `data_df.iloc[:]`

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



# # 데이터 셀렉션 및 필터링

- 열 위치에 -1을 입력하여 가장 마지막 데이터 추출

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

## (1) 맨 마지막 칼럼 데이터

```
data_df.iloc[:, -1]
```

	Gender
one	Male
two	Female
three	Male
four	Male

## (2) 맨 마지막 칼럼을 제외한 모든 데이터

```
data_df.iloc[:, :-1]
```

	Name	Year
one	Chulmin	2011
two	Eunkyoung	2016
three	Jinwoong	2015
four	Soobeom	2015

# # 데이터 셀렉션 및 필터링

## - loc 연산자

- 명칭(Label) 기반 인덱싱
- 행 위치: DataFrame 인덱스
- 열 위치: 칼럼명 지정
- 숫자형이 아닐 수 있기 때문에 -1 사용 X

loc[인덱스값, 칼럼명]

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


'Chulmin'

loc[시작점:종료점] : 시작점 ~ 종료점

```
data_df.loc['one':'two', ['Name', 'Year']]
```

	Name	Year
one	Chulmin	2011
two	Eunkyung	2016

오류)

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

0을 인덱스 값으로 가지지 않음

# # 데이터 셀렉션 및 필터링

- 불린 인덱싱
- loc[조건]: 자동으로 원하는 값 필터링

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

60세 이상인 승객의 나이와 이름만 추출

	Name	Age
33	Wheadon, Mr...	66.0
54	Ostby, Mr. ...	65.0
96	Goldschmidt...	71.0

- 복합 조건도 가능: ( and: & ), ( or: | ), ( Not: ~ )

```
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, Mi...	female	63.0	1	0	13502	77.9583	D7	S
829	830	1	1	Stone, Mrs....	female	62.0	0	0	113572	80.0000	B28	NaN

# # 정렬, Aggregation 함수, GroupBy 적용

- `sort_values()`

1) `by`: 해당 칼럼으로 정렬 수행

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

	PassengerId	Survived	Pclass	Name	Sex	Age
845	846	0	3	Abbing, Mr....	male	42.0
746	747	0	3	Abbott, Mr....	male	16.0
279	280	1	3	Abbott, Mrs...	female	35.0

2) `ascending`

True(default) – 오름차순  
False - 내림차순

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

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
868	869	0	3	van Melkebe...	male	NaN	0	0	345777
153	154	0	3	van Billiar...	male	40.5	0	2	A/5. 851
282	283	0	3	de Pelsmaek...	male	16.0	0	0	345778

```
titanic_sorted=titanic_df.sort_values(by='Age', ascending=True)
titanic_sorted.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
803	804	1	3	Thomas, Mas...	male	0.42	0	1	2625
755	756	1	2	Hamalainen,...	male	0.67	1	1	250649
644	645	1	3	Baclini, Mi...	female	0.75	2	1	2666

# # 정렬, Aggregation 함수, GroupBy 적용

- sort\_values()

## 3) inplace

False(default) - 원본 유지 & 정렬 결과 반환

```
sorted_df=titanic_df.sort_values(by=['Age'],inplace=False)
sorted_df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp
803	804	1	3	Thomas, Mas...	male	0.42	0
755	756	1	2	Hamalainen,...	male	0.67	1
644	645	1	3	Baclini, Mi...	female	0.75	2

True – 원본 변경 & 반환 값 없음

```
titanic_df.sort_values(by=['Age'],inplace=True)
titanic_df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp
803	804	1	3	Thomas, Mas...	male	0.42	0
755	756	1	2	Hamalainen,...	male	0.67	1
644	645	1	3	Baclini, Mi...	female	0.75	2

# # 정렬, Aggregation 함수, GroupBy 적용

- min( ), max( ), sum( ), count( )

count( ): Null 값을 반영하지 않음

```
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

특정 칼럼 추출

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

	0
Age	29.699118
Fare	32.204208

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

	0
Age	80.0000
Fare	512.3292

# # 정렬, Aggregation 함수, GroupBy 적용

- 입력 파라미터 by에 칼럼 입력 시, 대상 칼럼으로 GroupBy됨

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

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

by= 'Pclass' 호출 -> Pclass 칼럼 기준으로  
Groupby된 DataFrameGroupby 객체 변환

- 반환된 결과에 aggregation 함수 호출 -> 대상 칼럼 제외 모든 칼럼에 aggregation 함수 적용

```
titanic_groupby=titanic_df.groupby('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



# # 정렬, Aggregation 함수, GroupBy 적용

- 특정 칼럼만 aggregation 함수 적용시,  
반환된 DataFrameGroupBy 객체에 해당 칼럼 필터링 후 aggregation 함수 적용

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

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

1. Pclass 칼럼을 기준으로 그룹화  
→ Pclass 값이 같은 행끼리 하나의 그룹이 됨
2. 필터링: 특정 칼럼 선택
3. count( ) 수행



# # 정렬, Aggregation 함수, GroupBy 적용

- 여러 개의 aggregation 함수 명을 객체의 agg( ) 내에 인자로 입력해서 사용

```
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
---	------	------

- 여러 개의 칼럼을 서로 다른 aggregation 함수로 호출 시, 딕셔너리 형태로 처리

```
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 NaN  
: 머신러닝은 NaN 값을 처리하지 않기 때문에 다른 값으로 대체해야함
- `isna()`: 결손 데이터 여부 확인  
  
`sum()`: True -> 1, False -> 0 변환되므로 결손 데이터의 개수를 구할 수 있음

```
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	True	False
1		False	False	False	False	False	False	False	False	False	False	False
2		False	False	False	False	False	False	False	False	False	True	False

# # 결손 데이터 처리하기

- fillna(): 결손 데이터 대체

```
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....	male	22.0	1	0	A/5 21171	7.2500	C000	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	C000	S

- 실제 데이터 세트 값 변경
- 1. 반환 값을 다시 받기
- 2. inplace=True 파라미터 추가

# # apply lambda 식으로 데이터 가공

- **lambda 식**: 함수의 선언과 함수 내의 처리를 한 줄의 식으로 변환하는 식

```
def get_square(a):  
    return a**2  
  
print('3의 제곱은:', get_square(3))
```



```
lambda_square = lambda x: x**2  
print('3의 제곱은:', lambda_square(3))
```

3의 제곱은: 9

3의 제곱은: 9

입력 인자

lambda **x**: **x \*\* 2**

입력 인자를 기반으로 한 계산식이며  
호출 시 계산 결과가 반환됨

- **map()**: 여러 개의 값을 입력 인자로 사용할 경우

```
a = [1, 2, 3]  
squares = map(lambda x: x**2, a)  
list(squares)
```

[1, 4, 9]

# # apply lambda 식으로 데이터 가공

- if else 절
  - : 오른쪽에 반환 값이 있어야 하기 때문에,  
if 식보다 반환 값을 먼저 기술
- 나이가 15세 미만이면 'Child', 그렇지 않으면 'Adult' 로 구분하는 새로운 칼럼 생성

lambda x : if x<=15 'Child' else 'Adult' ❌

lambda x : 'Child' if x <=15 else 'Adult' ○

- else if 지원 X -> else 절을 ( ) 로 내포해 ( ) 내에서 다시 if else 적용

```
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

# THANK YOU

