

## \* 머신러닝의 분류

지도학습 – 분류, 회귀, 추천 시스템, 시각/음성 감지/인지, 텍스트 분석, NLP

비지도학습 – 클러스터링, 차원 축소, 강화학습

## \* 주요 패키지

Scikit-Learn : 데이터 마이닝 기반의 머신러닝

Numpy : 행렬/선형대수/통계 패키지

맷플롯립 : 시각화

## \* 넘파이

: 파이썬에서 선형대수 기반의 프로그램을 쉽게 만들 수 있도록 지원하는 대표적인 패키지

: 루프 사용X로 대량 데이터의 배열 연산을 가능하게 해 빠른 배열 연산 속도 보장

: C/C++과 같은 저수준 언어 기반의 호환 API 제공

→ 기존 C/C++ 기반의 타 프로그램과 데이터를 주고받거나 API를 호출해 쉽게 통합 가능

### 1) 넘파이 ndarray 개요

넘파이 기반 데이터 타입 : ndarray

array() : 파이썬의 리스트와 같은 다양한 인자를 입력 받아 ndarray로 변환

ndarray 배열의 shape 변수 : ndarray의 크기(행, 열의 수)를 튜플 형태로 가지고 있음

→ ndarray 배열의 차원 알 수 있음

```
import numpy as np
```

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

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

## 2) ndarray의 데이터 타입

ndarray내의 데이터값 : 숫자, 문자열, 불 등 모두 가능

ndarray내의 데이터 타입은 그 연산의 특성상 **같은 데이터 타입**만 가능

ndarray내의 데이터 타입 : dtype 속성으로 확인

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

```
<class 'list'>
<class 'numpy.ndarray'>
[1 2 3] int32
```

만약 다른 데이터 유형이 섞여 있는 리스트를 ndarray로 변경하면 **데이터 크기가 더 큰 데이터 타입**으로 형 변환 일괄 적용

```
list2 = [1, 2, 'test']
array2 = np.array(list2)
print(array2, array2.dtype)

list3 = [1, 2, 3.0]
array3 = np.array(list3)
print(array3, array3.dtype)
```

```
['1' '2' 'test'] <U11
[1. 2. 3.] float64
```

array2는 숫자형이 모두 문자형으로, array3은 int형이 모두 float형으로

ndarray내 데이터값의 타입 변경 : astype()

→ 보통 메모리 절약할 때 이용

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

array_int1 = array_float.astype('int32')
print(array_int1, array_int1.dtype)

array_float1 = np.array([1.1, 2.1, 3.1])
array_int2 = array_float1.astype('int32')
print(array_int2, array_int2.dtype)
```

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

### 3) ndarray를 편리하게 생성하기 – arange, zeros, ones

테스트용 데이터 만들 때, 대규모 데이터 일괄적으로 초기화할 때

arange() : array를 range()로 표현

```
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]
int32 (10,)
```

default 함수 인자는 stop 값

0부터 stop 값인 10에서 -1을 더한 9까지의 연속 숫자 값으로 구성된 1차원 ndarray 만들어줌

range와 유사하게 start 값도 부여해 0이 아닌 다른 값부터 시작한 연속 값 부여 가능

zeros() : 함수 인자로 튜플 형태의 shape 값 입력하면 모든 값을 0으로 채운 ndarray 반환

ones() : 유사하게 모든 값을 1로 채운 ndarray 반환

함수 인자로 dtype을 정해주지 않으면 default로 float64 형의 데이터로 ndarray 채움

```
zero_array = np.zeros((3, 2), dtype='int32')
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]]
int32 (3, 2)
[[1. 1.]
 [1. 1.]
 [1. 1.]]
float64 (3, 2)
```

### 4) ndarray의 차원과 크기를 변경하는 reshape()

reshape() : ndarray를 특정 차원 및 크기로 변환

변환을 원하는 크기를 함수 인자로 부여

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

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

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

```
array1:
[0 1 2 3 4 5 6 7 8 9]
array2:
[[0 1 2 3 4]
 [5 6 7 8 9]]
array3:
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
```

reshape()는 지정된 사이즈로 변경이 불가능하면 오류 발생!

-1을 인자로 사용하면 원래 ndarray와 호환되는 새로운 shape로 변환

```
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)
```

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

(-1, 5)의 의미 : array1과 호환될 수 있는 2차원 ndarray로 변환하되, 고정된 5개의 칼럼에 맞는 로우를 자동으로 새롭게 생성해 변환하라

(-1, 1) 자주 사용 : 여러 개의 로우를 갖되 반드시 1개의 칼럼 보장

tolist() : 리스트 자료형으로 변환

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

array5 = array3d.reshape(-1, 1)
print('array5:\n', array5.tolist())
print('array5.shape:', array5.shape)

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)
```

3차원을 2차원으로

1차원을 2차원으로

## 5) 넘파이의 ndarray의 데이터 세트 선택하기 – 인덱싱(Indexing)

### 가. 단일 값 추출

ndarray 객체에 해당하는 위치의 인덱스 값을 []안에 입력

```
array1 = np.arange(1, 10)
print('array1: ', array1)
value = array1[2]
print('value: ', value)
print(type(value))

array1: [1 2 3 4 5 6 7 8 9]
value: 3
<class 'numpy.int32'>
```

인덱스는 0부터 시작하므로 array1[2]는 3번째 인덱스 위치 값

인덱스에 마이너스 기호를 이용하면 맨 뒤에서부터 데이터 추출

```
print('맨 뒤의 값:', array1[-1], ', 맨 뒤에서 두 번째 값:', array1[-2])

맨 뒤의 값: 9 , 맨 뒤에서 두 번째 값: 8
```

단일 인덱스를 이용해 ndarray 내의 데이터값 간단히 수정 가능

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

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

### 2차원에서 단일 값 추출

```
array1d = np.arange(1, 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
```

row와 col은 이해를 돕기 위한 형식일 뿐 ndarray에서는 axis 사용

axis 0 : row, axis 1 : column

axis 생략 시 axis 0 의미

## 나. 슬라이싱

‘:’ 기호를 이용해 연속한 데이터를 슬라이싱해 추출 가능

‘:’ 사이에 시작, 종료 인덱스 표시하면 시작부터 종료-1의 위치에 있는 데이터의 ndarray 반환

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

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

시작 또는 종료 인덱스 생략 가능

```
array1 = np.arange(1, 10)
array4 = array1[:3]
print(array4)

array5 = array1[3:]
print(array5)

array6 = array1[:]
print(array6)
```

시작 인덱스를 맨 처음 인덱스인 0으로 간주

종료 인덱스를 맨 마지막 인덱스로 간주

맨 처음/맨 마지막 인덱스로 간주

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

2차원 ndarray에서도 1차원과 유사, 단지 콤마로 행과 열 인덱스 지칭

```
array1d = np.arange(1, 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, :]\n', array2d[1:3, :])
print('array2d[0:2, 0:2]\n', array2d[:2, 1:])
```

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

2차원 ndarray에서 뒤에 오는 인덱스를 없애면 1차원 ndarray 반환

### 다. 팬시 인덱싱

리스트나 ndarray로 인덱스 집합을 지정하면 해당 위치의 인덱스에 해당하는 ndarray를 반환하는 인덱싱 방식

```
array1d = np.arange(1, 10)
array2d = array1d.reshape(3, 3)

array3 = array2d[[0, 1], 2]
print(array3.tolist())

array4 = array2d[[0, 1], 0:2]
print(array4.tolist())

array5 = array2d[[0, 1]]
print(array5.tolist())
```

[3, 6]  
[[1, 2], [4, 5]]  
[[1, 2, 3], [4, 5, 6]]

[[0, 1], 2] → (0, 2), (1, 2)

[[0, 1], 0:2] → ((0, 0), (0, 1)), ((1, 0), (1, 1))

[[0, 1]] → ((0, :), (1, :)) → [[1, 2, 3], [4, 5, 6]]

### 라. 불린 인덱싱

조건 필터링과 검색 동시에 가능

for loop/if else 문보다 훨씬 간단하게 구현

```
array1d = np.arange(1, 10)
array3 = array1d[array1d > 5]
print(array3)
```

[6 7 8 9]

array1d>5를 적용하면 5보다 큰 데이터가 있는 위치는 True, 그렇지 않는 경우는 False로 반환

True 값에 해당하는 index 값만 저장

## 6) 행렬의 정렬 – sort()와 argsort()

### 가. 행렬 정렬

np.sort() 같이 넘파이에서 sort()를 호출하는 방식 : 원 행렬 그대로 유지한 채 정렬된 행렬 반환

ndarray.sort() 같이 행렬 자체에서 sort()를 호출하는 방식 : 원 행렬 자체를 정렬한 형태로 반환

```
org_array = np.array([3, 1, 9, 5])
print('원본:', org_array)
sort_array1 = np.sort(org_array)
print('np.sort() 후 반환된 정렬 행렬:', sort_array1)
print('np.sort() 후 원본 행렬:', org_array)

sort_array2 = org_array.sort()
print('org_array.sort() 후 반환된 정렬 행렬:', sort_array2)
print('org_array.sort() 후 원본 행렬:', org_array)
```

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

둘다 기본적으로 오름차순으로 행렬 내 원소 정렬

내림차순으로 정렬하기 위해서는[::-1] 적용

```
sort_array1_desc = np.sort(org_array)[::-1]
print(sort_array1_desc)

[9 5 3 1]
```

행렬이 2차원 이상일 경우 axis 축 값 설정을 통해 행 방향/열 방향으로 정렬 수행 가능

```
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]]
```

## 나. 정렬된 행렬의 인덱스를 반환하기

np.argsort() : 정렬 행렬의 원본 행렬 인덱스를 ndarray형으로 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]
```



내림차순으로 정렬 시 원본 행렬의 인덱스를 구할 때에도 `[::-1]` 사용하면 됨

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

행렬 내림차순 정렬 시 원본 행렬의 인덱스: [2 3 0 1]

## 7) 선형대수 연산 – 행렬 내적과 전치 행렬구하기

### 가. 행렬 내적(행렬 곱)

`np.dot()` 를 이용해 내적 계산 가능

```
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(dot_product)
```

```
[[ 58  64]
 [139 154]]
```

### 나. 전치 행렬

`transpose()` 이용

```
A=np.array([[1, 2], [3, 4]])
transpose_mat = np.transpose(A)
print(transpose_mat)
```

```
[[1 3]
 [2 4]]
```

## \* 판다스

핵심 객체 : dataframe – 여러 개의 행과 열로 이루어진 2차원 데이터를 담은 데이터 구조체

Index : RDBMS의 PK처럼 개별 데이터를 고유하게 식별하는 Key 값

Series : 칼럼이 하나뿐인 데이터 구조체

### 1) 판다스 시작

주피터 노트북과 데이터 파일이 같은 디렉터리에 있을 때 `read.csv('파일이름')`으로 로딩 가능

```
import pandas as pd
```

```
titanic_df = pd.read_csv('train.csv')
print(type(titanic_df))
titanic_df
```

```
<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
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

```
891 rows x 12 columns
```

pd.read\_csv() : 파일명 인자로 들어온 파일을 로딩해 dataframe 객체로 반환

read\_csv() : 별다른 파라미터 지정이 없으면 파일의 맨 처음 로우를 칼럼명으로

맨 왼쪽 0, 1, 2, ... : 판다스의 Index 객체 값

dataframe.head() : 맨 앞에 있는 N개의 로우 반환, default 5

```
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	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

dataframe.shape : 행과 열의 크기

```
print(titanic_df.shape)
```

```
(891, 12)
```

```
titanic_df.info()
```

```
<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
```

info() : 칼럼의 타입, null 데이터 개수, 데이터 분포도 등

전체 데이터 891개 row, 12개 column

object – 문자열 타입

714 non-null : 891개 중 714개가 null이 아님

칼럼 2개 float64, 5개 int64, 5개 object 타입

```
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

describe() : 칼럼별 숫자형 데이터값의 n-percentile 분포도, 평균값, 최댓값, 최솟값

오직 숫자형 칼럼의 분포도만 조사

숫자 칼럼이 숫자형 카테고리 칼럼인지 판단할 수 있게 도와줌

PassengerID – 승객 ID 식별하는 칼럼이므로 1~891까지 숫자 할당 → 의미X

Survived – 0과 1로 이루어진 숫자형 카테고리 칼럼

Pclass – 1, 2, 3으로 이루어진 숫자형 카테고리 칼럼

```
value_counts = titanic_df['Pclass'].value_counts()
print(value_counts)
```

```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

dataframe의 []연산자 내부에 칼럼명을 입력하면 series 형태로 특정 칼럼 데이터 세트 반환

→ 반환된 series 객체에 value\_counts() 메서드 호출하면 해당 칼럼값의 유형, 건수 확인 가능

Pclass 값 3이 491개, 1이 216개, 2가 184개

Series – Index + 하나의 칼럼으로 구성된 데이터

value\_counts() 메서드 : Series 객체에 정의, dataframe X

## 2) DataFrame과 리스트, 딕셔너리, 넘파이 ndarray 상호 변환

DataFrame ↔ 리스트, 딕셔너리, 넘파이 ndarray

가. 넘파이 ndarray, 리스트, 딕셔너리를 DataFrame으로 변환하기

2차원 이하의 데이터들만 DataFrame으로 변환 가능

```
col_name1=['col1']
list1=[1, 2, 3]
array1=np.array(list1)
print(array1.shape)
df_list1=pd.DataFrame(list1, columns=col_name1)
print(df_list1)
df_array1 = pd.DataFrame(array1, columns=col_name1)
print(df_array1)
```

```
(3,)
  col1
0     1
1     2
2     3
  col1
0     1
1     2
2     3
```

1차원 리스트로 만든 DataFrame

1차원 ndarray로 만든 DataFrame

```
col_name2=['col1', 'col2', 'col3']
list2=[[1, 2, 3], [11, 12, 13]]
array2=np.array(list2)
print(array2.shape)
df_list2=pd.DataFrame(list2, columns=col_name2)
print(df_list2)
df_array2 = pd.DataFrame(array2, columns=col_name2)
print(df_array2)
```

```
(2, 3)
  col1  col2  col3
0     1     2     3
1    11    12    13
  col1  col2  col3
0     1     2     3
1    11    12    13
```

2차원 리스트로 만든 DataFrame

2차원 ndarray로 만든 DataFrame

```
dict={'col1':[1, 11], 'col2':[2, 22], 'col3':[3, 33]}
df_dict = pd.DataFrame(dict)
print(df_dict)
```

```
   col1  col2  col3
0      1     2     3
1     11    22    33
```

딕셔너리를 DataFrame으로

Key – 컬럼명, Value – 컬럼 데이터

## 나. DataFrame을 넘파이 ndarray, 리스트, 딕셔너리로 변환하기

DataFrame 객체의 values를 이용

```
array3 = df_dict.values
print(type(array3), array3.shape)
print(array3)
```

DataFrame을 ndarray로

```
<class 'numpy.ndarray'> (2, 3)
[[ 1  2  3]
 [11 22 33]]
```

```
list3 = df_dict.values.tolist()
print(type(list3))
print(list3)
```

DataFrame을 리스트로

ndarray에 tolist() 호출

```
<class 'list'>
[[1, 2, 3], [11, 22, 33]]
```

```
dict3 = df_dict.to_dict('list')
print(type(dict3))
print(dict3)
```

```
<class 'dict'>
{'col1': [1, 11], 'col2': [2, 22], 'col3': [3, 33]}
```

DataFrame을 딕셔너리로

인자로 list 입력 시 딕셔너리 값이 리스트형으로 반환

### 3) DataFrame의 칼럼 데이터 세트 생성과 수정

```
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

Series에 상숫값 할당하면 Series의 모든 데이터 세트에 일괄 적용

```
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

기존 칼럼 Series의 데이터 이용해 새로운 칼럼 Series 만들기

```
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

기존 칼럼 값 일괄 업데이트

### 4) DataFrame 데이터 삭제

drop() 메서드에서 중요한 파라미터

axis : axis=1 입력 시 칼럼 축 방향으로 드롭, axis=0 입력 시 로우 축 방향으로 드롭

labels : 인덱스

```
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

'Age\_0' 칼럼 삭제됨

inplace : 디폴트 False, False면 자기 자신의 DataFrame의 데이터 삭제X, 삭제된 DataFrame 반환  
True면 자신의 DataFrame 데이터 삭제, 반환 값 None

여러 개의 칼럼 삭제 – 리스트 형태로 삭제하고자 하는 칼럼 명 labels 파라미터로

```
drop_result = titanic_df.drop(['Age_0', 'Age_by_10', 'Family_No'], axis=1, inplace=True)
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	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

## 5) Index 객체

DataFrame.index, Series.index 속성 을 통해 Index 객체 추출 가능

```
titanic_df = pd.read_csv('train.csv')
indexes = titanic_df.index
print(indexes)
print(indexes.values)
```

```
RangeIndex(start=0, stop=891, step=1)
[ 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]
```

반환된 Index 객체의 실제 값은 넘파이 1차원 ndarray로 볼 수 있음

단일 값 반환, 슬라이싱 가능

0~890

Series 객체에 연산 함수 적용할 때 Index는 연산에서 제외됨, 오직 식별용으로

reset\_index() : 새롭게 인덱스를 연속 숫자 형으로 할당, 기존 인덱스는 'index'라는 새로운 칼럼 명으로 추가

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

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

인덱스가 연속된 int 숫자형 데이터가 아닐 경우 이를 연속 int 숫자형 데이터로 만들 때 사용

```
value_counts = titanic_df['Pclass'].value_counts()
print(value_counts)

new_value_counts = value_counts.reset_index(inplace=False)
print(new_value_counts)
```

```
3    491
1    216
2    184
Name: Pclass, dtype: int64
   index  Pclass
0      3     491
1      1     216
2      2     184
```

reset\_inde()의 파라미터 중 drop=True로 설정하면 기존 인덱스는 새로운 칼럼으로 추가되지 않고 삭제됨

## 6) 데이터 선택 및 필터링

### 가. DataFrame의 [] 연산자

넘파이 [] : 행의 위치, 열의 위치, 슬라이싱 범위 등 지정해 데이터 가져오기

DataFrame [] : 칼럼 명 문자(칼럼 명 리스트), 인덱스로 변환 가능한 표현식

DataFrame [] 내에 숫자 값 입력 시 오류 발생, 그러나 슬라이싱, 불린 인덱싱 가능

```
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

```
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	Heikkinen, 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

### 나. DataFrame ix[] 연산자

명칭과 위치 기반 인덱싱 모두를 허용해 행 위치에 적용되는 인덱스값과 위치 기반 인덱싱이

integer형일 때 코드 작성에 혼선을 초래할 우려가 있어 사라질 예정, 따라서 새롭게 명칭 기반 인덱싱 연산자인 `loc[]`와 위치 기반 인덱싱인 `iloc[]` 연산자 도입

### 다. DataFrame `iloc[]` 연산자

위치 기반 인덱싱 – integer, integer형의 슬라이싱, 팬시 리스트 값

명칭 입력 시, 문자열 인덱스 입력 시 오류

```
titanic_df.iloc[0, 4]
'male'
```

### 라. DataFrame `loc[]` 연산자

명칭 기반 인덱싱 – 행 위치에 DataFrame index 값, 열 위치에 칼럼 명

```
titanic_df.loc[0, 'Name']
'Braund, Mr. Owen Harris'
```

슬라이싱 적용 시 종료 값까지 포함 – 숫자형이 아닐 수 있기 때문에 -1 할 수 X

```
titanic_df.loc[0:2, 'Name']
0      Braund, Mr. Owen Harris
1  Cumings, Mrs. John Bradley (Florence Briggs Th...
2  Heikkinen, Miss. Laina
Name: Name, dtype: object
```

### 마. 불린 인덱싱

`[], ix[], loc[]` 공통 지원

`iloc[]`은 정수형 값이 아닌 불린 값에 대해서 지원X

```
titanic_df = pd.read_csv('train.csv')
titanic_boolean = titanic_df[titanic_df['Age']>60]
titanic_boolean
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
33	34	0	2	Wheaton, Mr. Edward H	male	66.0	0	0	C.A. 24579	10.5000	NaN	S
54	55	0	1	Ostby, Mr. Engelhart Cornelius	male	65.0	0	1	113509	61.9792	B30	C
96	97	0	1	Goldschmidt, Mr. George B	male	71.0	0	0	PC 17754	34.6542	A5	C
116	117	0	3	Connors, Mr. Patrick	male	70.5	0	0	370369	7.7500	NaN	Q
170	171	0	1	Van der hoef, Mr. Wilcoff	male	61.0	0	0	111240	22.5000	B10	S



```
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

원하는 칼럼 명만 별도로 추출

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

	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

여러 개의 조건을 복합 조건 연산자를 이용해 표현 가능

## 7) 정렬, Aggregation 함수, GroupBy 적용

### 가. DataFrame, Series의 정렬 – sort\_values()

sort\_values()의 주요 파라미터

by : 해당 칼럼으로 정렬 수행

ascending = True : 오름차순 정렬(디폴트)

inplace = False : 호출한 DataFrame은 그대로 유지, 정렬된 DataFrame 결과로 반환

```
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 Pelsmaecker, Mr. Alfons	male	16.0	0	0	345778	9.5	NaN	S

### 나. Aggregation 함수 적용

min(), max(), sum(), count() 같은 함수

DataFrame에서 바로 호출할 경우 모든 칼럼에 해당 aggregation 적용

```
titanic_df.count()
```

```
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
```

```
titanic_df[['Age', 'Fare']].mean()
Age      29.699118
Fare     32.204208
dtype: float64
```

## 다. groupby() 적용

대상 칼럼으로 groupby

DataFrameGroupBy라는 또 다른 형태의 DataFrame 반환

반환된 결과에 aggregation 함수 호출 시 groupby() 대상 칼럼 제외한 모든 칼럼에 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

DataFrame의 groupby()에 특정 칼럼만 aggregation 함수 적용하려면 groupby()로 반환된 DataFrameGroupBy 객체에 해당 칼럼을 필터링한 뒤 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



결손 데이터 개수 : `isna() + sum()`, `True==1`, `False==0`

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

```
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
```

### 나. `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	Heikinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	C000	S

`fillna()`를 이용해 반환 값을 받거나 `inplace=True` 파라미터를 `fillna()`에 추가해야 실제 데이터 세트 값이 변경됨

‘Age’의 NaN을 평균 나이로, ‘Embarked’의 NaN을 ‘S’로

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

```
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
```

모든 결손데이터 처리 완료

## 9) apply lambda 식으로 데이터 가공

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

```
lambda_square = lambda x : x**2
print(lambda_square(3))
```

9

‘:’ 왼쪽 - 입력 인자, 오른쪽 - 입력 인자의 계산식(반환 값)

```
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

if 절의 경우 if 식보다 반환 값 먼저 기술! → ‘:’ 오른쪽에 반환 값이 있어야 하기 때문

else if 지원X

else if 이용 시 else 절을 ()로 내포해 () 내에서 다시 if, else 적용해 사용