Chapter 01 : 파이썬 기반의 머신러닝과 생태계 이해

01-01 머신러닝의 개념

- **[머신러닝(Machine Learning)]** : 애플리케이션을 수정하지 않고도 데이터를 기반으로 패턴을 학습하고 결과를 예측하는 알고리즘 기법
    - 기존 소프트웨어 코드만으로 해결하기 어려웠던 문제도 해결 가능
    - 소프트웨어 코드로 로직을 구성하여 이들을 관통하는 일정한 패턴을 찾기 어려운 경우 사용
    - 데이터를 기반으로 패턴을 인지해 문제를 해결
        - **머신러닝 알고리즘**
            - 데이터를 기반으로 통계적인 신뢰도를 강화
            - 예측 오류를 최소화하기 위한 다양한 수학적 기법 적용
            - 데이터 내 패턴을 스스로 인지, 신뢰도 있는 예측 결과를 도출
    - 데이터를 관통하는 패턴을 학습, 이에 기반한 예측을 수행하며 데이터 분석 영역에 새로운 혁신을 가져옴
    - 데이터마이닝, 영상 인식, 음성 인식, 자연어 처리 등 다양한 분야에서 두각을 보임

데이터 분석 영역은 머신러닝 기반의 예측 분석(Predictive Analysis)으로 재편 중.

- **[머신러닝의 분류]**
    - **지도학습(Supervised Learing)**
        - **분류(Classification)**
        - **회귀(Regression)**
        - 추천 시스템
        - 시각/음성 감지/인지
        - 텍스트 분석, NLP
    - **비지도학습(Un-supervised Learning)**
        - 클러스터링
        - 차원 축소
        - 강화학습

<<마스터 알고리즘>> 에서는 머신러닝 알고리즘을 기호주의, 연결주의, 베이지안 통계, 유추주의 등의 유형으로 나눔. 알고리즘의 특징을 잘 반영한 부류에 해당.

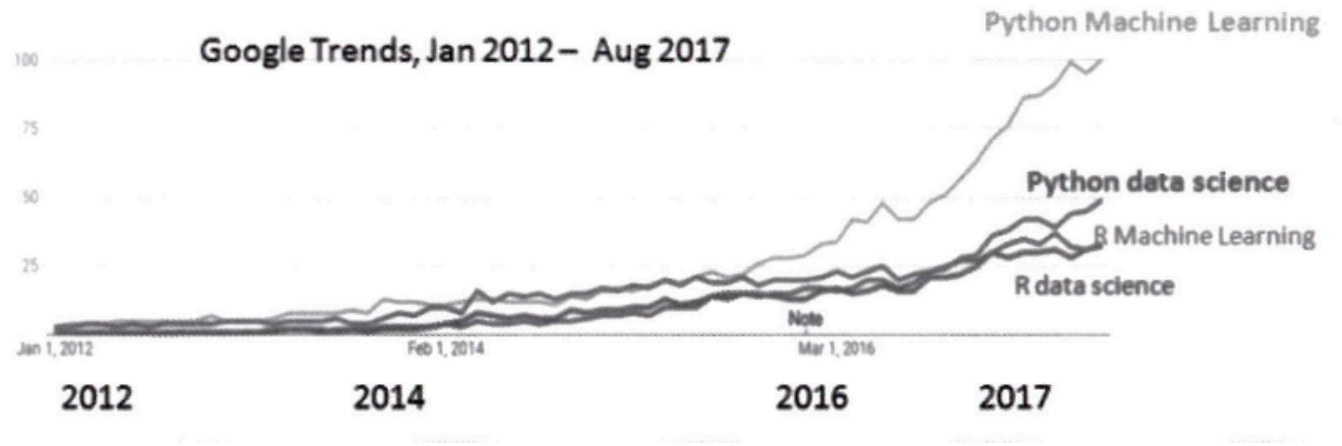
- **[데이터 전쟁]** 데이터와 머신러닝 알고리즘 모두 중요한 요소. 머신러닝이 본격적으로 보편화될 시에는 데이터의 중요성이 무엇보다 커짐.
    - 머신러닝의 가장 큰 단점 : 데이터에 매우 의존적이다.

        - 가비지 인(Garbage In), 가비지 아웃(Garbage Out) = 좋은 품질의 데이터를 갖추지 못한다면 머신러닝의 수행 결과도 좋을 수 없다.
        - 머신러닝을 이용해 데이터만 집어넣으면 자동으로 최적화된 결과를 도출할 것이라는 믿음은 환상에 불과!(특정 경우에는 개발자가 직접 만든 코드보다 정확도가 더 떨어질 수도.)

데이터를 이해하고 효율적으로 가공, 처리, 추출해 퍼적의 데이터를 기반으로 알고리즘을 구동할 수 있도록 준비하는 능력 중요. + 최적의 머신러닝 알고리즘, 모델 파라미터를 구축하는 능력

- **[파이썬과 R 기반의 머신러닝 비교]**
- 머신러닝 프로그램을 작성할 수 있는 대표적인 오픈 소스 프로그램 언어 : 파이썬, R
    - **R** : 통계 전용 프로그램 언어
        - 다양하고 전통적인 통계 패키지 보유
    - **파이썬** : 다양한 영역에서 사용되는 개발 전문 프로그램 언어
        - 직관적인 문법, 객체지향과 함수형 프로그래밍 모두를 포괄 - 유연한 프로그램 아키텍쳐
        - 다양한 라이브러리

(C/C++, JAVA도 가능은 하나, 개발 생산성이 떨어지고 지원 패키지와 생태계가 활발하지 않음 - 컴파일러 기반, 임베디드 영역 등에 활용)



- 파이썬 선호 현황 및 특징
    - 쉽고 뛰어난 개발 생산성
    - 오픈 소스 계열의 전폭적인 지원, 많은 라이브러리 보유
        - 딥러닝 프레임워크(텐서플로(TensorFlow), 케라스(Keras), 파이토치(PyTorch))에서 파이썬 우선 정책으로 파이썬을 지원 중.
    - 인터프리터 언어(Interpreter Language) 특성 상 속도는 느리지만, 뛰어난 확장성, 유연성, 호환성으로 다양한 영역에서 사용되고 있음
    - 머신러닝 애플리케이션과 결합한 다양한 애플리케이션 개발이 가능함
    - 다양한 기업 환경으로의 확산이 가능함.

01-02 파이썬 머신러닝 생태계를 구성하는 주요 패키지

- 파이썬 기반의 머신러닝을 익히기 위해 필요한 패키지
    - 머신러닝 패키지 : **Scikit-Learn**, 텐서플로, 케라스
    - 행렬/선형대수/통계 패키지 : **넘파이(NumPy**; 행렬과 선형대수를 다루는 패키지**)**, 사이파이(SciPy; 자연과학과 통계를 위한 다양한 패키지)
    - 데이터 핸들링 : 판다스(2차원 데이터 처리), 맷플롯립(Matplotlib; 데이터 시각화)
    - 시각화 : 맷플롯립, 시본(Seaborn)
    - 아이파이썬(iPython, Interactive Python; 대화형 파이썬 툴, 코드 영역별 개별 수행 지원, 영역별 코드 이해의 명확한 설명 가능), 주피터 노트북(Jupyter Notebook; 중요 코드 단위로 설명 필기, 코드 역할의 직관적인 이해 가능)
- **사이킷런(Scikit-Learn)**
    - **넘파이 & 판다스**
        - 사이킷런의 머신러닝 알고리즘에 입력하기 위한 데이터의 추출/가공/변형, 원하는 차원 배열로의 변환을 포함해 머신러닝 알고리즘 처리 결과에 대한 다양한 가공 등으로 구성 -> 넘파이와 파이썬의 묶

01-03 넘파이

넘파이(NumPy) = Numerical Python

- 선형대수 기반의 프로그램을 쉽게 만들 수 있도록 지원하는 대표적인 패키지
- 루프를 사용하지 않고 대량 데이터의 배열 연산을 가능하게 하므로, 빠른 배열 연산 속도를 보장함.
- C/C++과 같은 저수준 언어 기반의 호환 API 제공. (수행 성능이 중요한 부분은 해당 언어 사용하여 작성 후 넘파이에서 호출하는 방식)
- 배열 기반 연산 기능 보유
- 다양한 데이터 핸들링 기능 보유

[넘파이 narray 개요]

```
1 import numpy as np
```

```
array1 = np.array([1, 2, 3])
print('array1 type: ', type(array1))
print('array1 array 형태: ', array1.shape)
# shape 변수: ndarray의 크기(행과 열의 수)를 튜플 형태로 가짐
#             ndarray 배열의 차원까지 알 수 있음

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

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

```
array1 type:  <class 'numpy.ndarray'>
array1 array 형태:  (3,)
array2 type:  <class 'numpy.ndarray'>
array2 array 형태:  (2, 3)
array3 type:  <class 'numpy.ndarray'>
array3 array 형태:  (1, 3)
```

```
# 각 array의 차원 확인하기 - ndarray.ndim
print('array1: {:0}차원, array2: {:1}차원, array3: {:2}차원'.format(array1.ndim

# 리스트 []: 1차원, 리스트의 리스트 [[]]: 2차원
```

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

[ndarray의 데이터 타입]

숫자 값, 문자열 값, 불 값 등 모두 가능.

- 숫자형
  - int형(8 bit, 16 bit, 32 bit)
  - unsigned int형(8 bit, 16 bit, 32 bit)
  - float형(16 bit, 32 bit, 64 bit, 128 bit)
  - complex 타입(* 더 큰 숫자값이나 정밀도를 위함)

연산 특성상 **같은 데이터 타입만** 가능함. (= 한 개의 ndarray 객체에 int와 float가 함께 있을 수 없다.)

ndarray 내의 데이터 타입은 **dtype** 속성으로 확인 가능

```
list1 = [1, 2, 3]
print(type(list1)) # class, int32형

array1 = np.array(list1)
print(type(array1)) # class, int32형
print(array1, array1.dtype) # 데이터 타입
```

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

```
list2 = [1, 2, 'test'] # + string형
array2 = np.array(list2)
print(array2, array2.dtype)
# 숫자형 값이 문자형 값 '1', '2'로 변환됨 - 모두 같은 데이터 타입이어야 하므로

list3 = [1, 2, 3.0] # + float형
array3 = np.array(list3)
print(array3, array3.dtype)
# int형이 float(64)형으로 변환됨
```

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

ndarray 내 데이터값의 타입 변경은 **astype()** 메서드를 이용해 가능함

- 메모리를 더 절약해야 할 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
```

[ndarray를 편리하게 생성하기 - arange, zeros, ones]

테스트용으로 데이터를 만들거나 대규모의 데이터를 일괄적으로 초기화해야 할 경우 사용

- **arange() 함수** : array를 range()로 표현하는 것
  - 0부터 함수 인자 값 -1까지의 값을 순차적으로 ndarray의 데이터값으로 변환해줌

```
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() 함수** : 함수 인자로 튜플 형태의 shape 값을 입력하면 모든 값을 0으로 채운 해당 shape를 가진 ndarray를 반환함
- **ones() 함수** : 함수 인자로 튜르 형태의 shape 값을 입력하면 모든 값을 1로 채운 해당 shape를 가진 ndarray를 반환함
    - 함수 인자로 dtype를 지정하지 않으면 default로 float64형의 데이터로 ndarray를 채움

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

# ones() 함수
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)
```

[ndarray의 차원과 크기를 변경하는 reshape()] : ndarray를 특정 차원 및 크기로 변환함.

- 변환을 원하는 크기를 함수 인자로 부여함.

```
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]]
```

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call
last)
<ipython-input-12-28a7a100c135> in <cell line: 1>()
----> 1 array1.reshape(4, 3)

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

```
1 array1 = np.arange(10)
2 print(array1)
3
4 array2 = array1.reshape(-1, 5)
5 print('array2 shape: ', array2.shape)
6
7 array3 = array1.reshape(5, -1)
8 print('array3 shape: ', array3.shape)
```

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

```
1 array1 = np.arange(10)
2
3 array4 = array1.reshape(-1, 4)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call
last)
<ipython-input-14-177d495d8b4b> in <cell line: 3>()
      1 array1 = np.arange(10)
      2
----> 3 array4 = array1.reshape(-1, 4)

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

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

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

# 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)
```

[넘파이의 ndarray의 데이터 세트 선택하기 - 인덱싱(indexing)]

- *특정한 데이터만 추출* : 원하는 위치의 인덱스 값을 지정하면 해당 위치의 데이터가 반환됨
- *슬라이싱(Slicing)* : 연속된 인덱스상의 ndarray를 추출하는 방식
- *팬시 인덱싱(Fancy Indexing)* : 일정한 인덱싱 집합을 리스트 또는 ndarray 형태로 지정해 해당 위치에 있는 데이터의 ndarray를 반환
- *불린 인덱싱(Boolean Indexing)* : 특정 조건에 해당하는지 여부인 True/False 값 인덱싱 집합을 기반으로 True에 해당하는 인덱스 위치에 있는 데이터의 ndarray를 반환

[단일 값 추출]

```
# 1부터 9까지의 1차원 ndarray 생성
array1 = np.arange(start = 1, stop = 10)
print('array1: ', array1)

# index는 0부터 시작하므로 array1[2]는 3번째 index 위치의 데이터값을 의미
value = array1[2]
print('value: ', value)
print(type(value))
```

```
array1:  [1 2 3 4 5 6 7 8 9]
value:  3
<class 'numpy.int64'>
```

```
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 활용한 ndarray 내의 데이터 값 수정
array1[0] = 9
array1[8] = 0

print('array1: ', array1)
```

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

```
# 1차원 ndarray -> 2차원 3x3 ndarray -> 데이터 추출
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
```

[슬라이싱] : ':' 기호 이용해 연속한 데이터 추출 가능

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

print(array3)
print(type(array3))
```

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

```
array1 = np.arange(start = 1, stop = 10)

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]
```

```
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]
```

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

```
[1 2 3]
[4 5 6]
array2d[0] shape: (3,) array2d[1] shape: (3,)
```

[팬시 인덱싱(Fancy Indexing)] : 리스트나 ndarray로 인덱스 집합을 지정하면 해당 위치의 인덱스에 해당하는 ndarray를 반환하는 인덱싱 방식

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

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

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

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]]
```

[불린 인덱싱(Boolean Indexing)] : 조건 필터링과 검색을 동시에 할 수 있기 때문에 매우 자주 사용되는 인덱싱 방식.

- ndarray의 인덱싱을 지정하는 [ ] 내에 조건문을 그대로 기재하기만 하면 됨.

```
array1d = np.arange(start = 1, stop = 10)

# [ ] 안에 array1d > 5 Boolean indexing을 적용
array3 = array1d[array1d > 5]

print('array1d > 5 불린 인덱싱 결과 값: ', array3)
```

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

```
array1d > 5
```

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

```
boolean_indexes = np.array([False, False, False, False, False, True, True, T
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])
array4 = array1d[indexes]
print('일반 인덱스로 필터링 결과: ' , array4)
```

```
일반 인덱스로 필터링 결과:  [6 7 8 9]
```

[행렬의 정렬 - sort()와 argsort()]

- **sort()** : 넘파이 행렬 정렬

- **argsort()** : 정렬된 행렬의 인덱스 반환

[행렬 정렬]

- np.sort() : 원 행렬은 그대로 유지, 원 행렬의 정렬된 행렬을 반환
- ndarray.sort() : 원 행렬 자체를 정렬한 형태로 변환, 반환값은 None

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

# np.sort( )로 정렬
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( )로정렬
sort_array2 = org_array.sort()
print('org_array.sort( ) 호출 후 반환된 행렬: ', sort_array2)
print('org_array.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]
org_array.sort( ) 호출 후 반환된 행렬:  None
org_array.sort( ) 호출 후 원본 행렬:  [1 3 5 9]
```

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

```
내림차순으로 정렬: [9 5 3 1]
```

[정렬된 행렬의 인덱스를 반환하기]

```
# argsort() 활용
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]
```

```
org_array = np.array([3, 1, 9, 5])
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]
```

```
# 시험 성적 순으로 학생 이름 출력하기
import numpy as np

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']
```

[선형 대수 연산 - 행렬 내적과 전치 행렬 구하기]

[행렬 내적(행렬 곱)]

np.dot() 이용

```
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() 이용

```
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]]
```

01-04 데이터 핸들링 - 판다스

판다스 : 행과 열로 이루어진 2차원 데이터를 효율적으로 가공/처리할 수 있는 다양한 기능 제공

- 고수준 API 제공

- 리스트, 컬렉션, 넘파이 등의 내부 데이터
- CSV 등의 파일을 쉽게 데이터 프레임으로 변경
- 데이터의 가공/분석을 편리하게 수행할 수 있게 만들어줌
- 대표적인 데이터 핸들링 프레임워크
- csv, tab 같은 다양한 유형의 분리 문자로 칼럼을 분리한 파일을 손쉽게 DataFrame으로 로딩할 수 있게 해줌

핵심 객체: Data Frame - 여러 개의 행과 열로 이루어진 2차원 데이터를 담는 데이터 구조체

- **Index** : 개별 데이터를 고유하게 식별하는 Key값
- **Series** : 칼럼이 하나 뿐인 데이터 구조체 (DataFrame: 칼럼이 여러 개인 데이터 구조체)
    - DataFrame은 여러 개의 Series로 이루어짐

[판다스 시작 - 파일을 DataFrame으로 로딩, 기본 API]

```
1 import pandas as pd
```

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
1 titanic_df = pd.read_csv('/content/train.csv')
2 titanic_df.head(5)
```

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

Next steps: View recommended plots

```
1 print('DataFrame 크기: ', titanic_df.shape)
```

```
DataFrame 크기:  (891, 12)
```

```
1 # 총 데이터 건수와 데이터 타입, Null 건수를 알아보기
2 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
```

```
# 칼럼별 숫자형 데이터값의 n-percentile 분포도, 평균값, 최댓값, 최솟값 확인하기
# describe()는 오직 숫자형(int, float) 칼럼의 분포도만 조사함. (object 타입의 칼럼은 출력
titanic_df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	89
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	3
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	4
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	1
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	3
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	5

```
# Pclass 칼럼 값의 분포 및 구성 살펴보기
# value_counts() : 해당 칼럼값의 유형과 건수 확인 가능, 지정된 칼럼의 데이터값 건수를 반환
value_counts = titanic_df['Pclass'].value_counts()
print(value_counts)
```

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

```
# 특정 칼럼의 Series 객체 반환
titanic_pclass = titanic_df['Pclass']
print(type(titanic_pclass))
```

```
<class 'pandas.core.series.Series'>
```

```
titanic_pclass.head() # 상위 다섯 개 값만 추출. 괄호 안에 숫자 미지정 시 자동으로 다섯 줄만
```

```
0    3
1    1
2    3
3    1
4    3
Name: Pclass, dtype: int64
```

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

# Series 객체 – 보통 왼쪽이 인덱스값, 오른쪽이 데이터값
# 고유성이 보장된다면 의미 있는 데이터 값 할당도 가능함.
# value_counts()는 칼럼별 데이터 건수를 반환하므로 고유 칼럼값을 식별자로 사용할 수 있음.
# 인덱스는 숫자형뿐만 아니라 문자열도 가능함. (고유성이 보장되어야 함.)
```

```
<class 'pandas.core.series.Series'>
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

[DataFrame과 리스트, 딕셔너리, 넘파이 ndarray 상호 변환]

- **넘파이 ndarray, 리스트, 딕셔너리를 DataFrame으로 변환하기**

```
# 1차원 형태의 리스트와 넘파이 ndarray를 DataFrame으로 변환하기
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 Dataf

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
```

```
# 2차원 형태의 데이터를 기반으로 DataFrame 생성하기
# 3개의 칼럼명이 필요함.
col_name2 = ['col1', 'col2', 'col3']

# 2×3 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
```

```
# Key는 문자열 칼럼명으로 매핑, Value는 리스트형(또는 ndarray) 칼럼 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
```

- **DataFrame을 넘파이 ndarray, 리스트, 딕셔너리로 변환하기**

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

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

```
# 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**

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

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

Next steps: View recommended plots

```
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
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282

Next steps: View recommended plots

```
1 titanic_df['Age_by_10'] = titanic_df['Age_by_10'] +10
2 titanic_df.head(3)
```

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

Next steps: View recommended plots

- **DataFrame 데이터 삭제**

drop() 메서드 사용

```
1 # DataFrame.drop(labels = None, axis = 0, index = None, columns = None, leve
2
3 titanic_drop_df = titanic_df.drop('Age_0', axis = 1)
4 titanic_drop_df.head(3)
```

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

Next steps: View recommended plots

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

```
inplace = True 로 drop 후 반환된 값 :  None
```

```
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  Parc
0            1         0       3  Braund, Mr....    male  22.0      1
1            2         1       1  Cumings, Mr...  female  38.0      1
2            3         1       3  Heikkinen, ...  female  26.0      0
#### after axis 0 drop ####
   PassengerId  Survived  Pclass            Name     Sex   Age  SibSp  Parc
3            4         1       1  Futrelle, M...  female  35.0      1
4            5         0       3  Allen, Mr. ...    male  35.0      0
5            6         0       3  Moran, Mr. ...    male   NaN      0
```

- **Index 객체** : DataFrame, Series의 레코드를 고유하게 식별하는 객체

```
# 원본 파일 다시 로딩
titanic_df = pd.read_csv('train.csv')

# Index 객체 추출
indexes = titanic_df.index
print(indexes)

# Index 객체를 실제 값 array로 변환
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]
```

```
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
```

```
# index는 오직 식별용으로만 사용됨
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))
```

```
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
```

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

	index	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	T
0	0	1	0	3	Braund, Mr....	male	22.0	1	0	A
1	1	2	1	1	Cumings, Mr...	female	38.0	1	0	F
2	2	3	1	3	Heikkinen, ...	female	26.0	0	0	S

Next steps: View recommended plots

```
print('### before reset index ###')
value_counts = titanic_df['Pclass'].value_counts()
print(value_counts)
print('value_counts 객체변수 타입: ', type(value_counts))
new_value_counts = value_counts.reset_index(inplace = False)

print('### After reset_index ###')
print(new_value_counts)
print('new_value_counts 객체변수 타입: ',type(new_value_counts))
```

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

[데이터 셀렉션 및 필터링]

```
print('단일 칼럼 데이터 추출:\n', titanic_df['Pclass'].head(3))
print('\n 여러 칼럼의 데이터 추출: \n', titanic_df[['Survived', 'Pclass']].head(3
print('[]안에 숫자 index는 KeyError 오류 발생: \n', titanic_df[0])
```

```
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
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call
last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in
get_loc(self, key, method, tolerance)
   3801             try:
-> 3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:

 4 frames
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:

KeyError                                  Traceback (most recent call
last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in
get_loc(self, key, method, tolerance)
   3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:
-> 3804                 raise KeyError(key) from err
   3805             except TypeError:
   3806                 # If we have a listlike key, _check_indexing_error
will raise

KeyError: 0
```

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

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr....	male	22.0	1	0	A/5 21171	7
1	2	1	1	Cumings, Mr...	female	38.0	1	0	PC 17599	7

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

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr....	male	22.0	1	0	A/5 21171
2	3	1	3	Heikkinen, ...	female	26.0	0	0	STON/O2. 31...
4	5	0	3	Allen, Mr. ...	male	35.0	0	0	373450

DataFrame ix[ ] 연산자

: 행 위치 지정으로 인덱스값 0, 열 위치 지정으로 칼럼 명인 'Pclass'를 입력해 원하는 위치의 데이터를 추출할 수 있다.

- 칼럼 **명칭** 기반 인덱싱 - ix[ ] 연산자 처리 방식
  - **loc[ ]**: 칼럼 명칭 기반 인덱싱 연산자
- 칼럼 **위치** 기반 인덱싱
  - **iloc[ ]**: 칼럼 위치 기반 인덱싱 연산자

```
1 print('칼럼 위치 기반 인덱싱 데이터 추출:', titanic_df.ix[0, 2])
2 print('칼럼 명 기반 인덱싱 데이터 추출:', titanic_df.ix[0, 'Pclass'])
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-67-fa472f267976> in <cell line: 1>()
----> 1 print('칼럼 위치 기반 인덱싱 데이터 추출:', titanic_df.ix[0, 2])
      2 print('칼럼 명 기반 인덱싱 데이터 추출:', titanic_df.ix[0, 'Pclass'])

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in __getattr__(self, name)
   5900         ):
   5901             return self[name]
-> 5902         return object.__getattribute__(self, name)
   5903
   5904     def __setattr__(self, name: str, value) -> None:

AttributeError: 'DataFrame' object has no attribute 'ix'
```

```
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

Next steps: View recommended plots

명칭 기반 인덱싱과 위치 기반 인덱싱의 구분

- **명칭(Label) 기반 인덱싱** : 칼럼의 명칭을 기반으로 위치를 지정하는 방식
- **위치(Position) 기반 인덱싱** : 0을 출발점으로 하는 가로축, 세로축 좌표 기반의 행과 열 위치를 기반으로 데이터를 지정함.
    - 행, 열 값으로 정수가 입력됨

```
# data_df를 reset_index()로 새로운 숫자형 인덱스를 생성
data_df_reset = data_df.reset_index()
data_df_reset = data_df_reset.rename(columns={'index':'old_index'})

# 인덱스값에 1을 더해서 1부터 시작하는 새로운 인덱스값 생성
data_df_reset.index = data_df_reset.index + 1
data_df_reset
```

	old_index	Name	Year	Gender
1	one	Chulmin	2011	Male
2	two	Eunkyung	2016	Female
3	three	Jinwoong	2015	Male
4	four	Soobeom	2015	Male

Next steps: View recommended plots

```
data_df_reset.ix[1, 1]
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-70-be01c2f05827> in <cell line: 1>()
----> 1 data_df_reset.ix[1, 1]

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in __getattr__(self, name)
   5900         ):
   5901             return self[name]
-> 5902         return object.__getattribute__(self, name)
   5903
   5904     def __setattr__(self, name: str, value) -> None:

AttributeError: 'DataFrame' object has no attribute 'ix'
```

DataFrame iloc[ ] 연산자

: iloc[ ]은 위치 기반 인덱싱만 허용.

- 행과 열 값으로 integer / integer형의 슬라이싱, 팬시 리스트 값을 입력해줘야 함.

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

```
'Chulmin'
```

```
1 # 다음 코드는 오류를 발생시킵니다.
2 data_df.iloc[0, 'Name']
```

```
---------------------------------------------------------------------------
_
ValueError                                Traceback (most recent call
last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexing.py in
_validate_tuple_indexer(self, key)
    872             try:
--> 873                 self._validate_key(k, i)
    874             except ValueError as err:

 4 frames

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

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

ValueError                                Traceback (most recent call
last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexing.py in
_validate_tuple_indexer(self, key)
    873                 self._validate_key(k, i)
    874             except ValueError as err:
--> 875                 raise ValueError(
    876                     "Location based indexing can only have "
    877                     f"[{self._valid_types}] types"

ValueError: Location based indexing can only have [integer, integer slice
(START point is INCLUDED, END point is EXCLUDED), listlike of integers,
boolean array] types
```

```
1 # 다음 코드는 오류를 발생시킵니다.
2 data_df.iloc['one', 0]
```

```
---------------------------------------------------------------------------
_
ValueError                                Traceback (most recent call
last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexing.py in
_validate_tuple_indexer(self, key)
    872             try:
--> 873                 self._validate_key(k, i)
    874             except ValueError as err:

                    4 frames

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

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

ValueError                                Traceback (most recent call
last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexing.py in
_validate_tuple_indexer(self, key)
    873                 self._validate_key(k, i)
    874             except ValueError as err:
--> 875                 raise ValueError(
    876                     "Location based indexing can only have "
    877                     f"[{self._valid_types}] types"

ValueError: Location based indexing can only have [integer, integer slice
(START point is INCLUDED, END point is EXCLUDED), listlike of integers,
boolean array] types
```

```
1 data_df_reset.iloc[0, 1]
```

```
'Chulmin'
```

DataFrame loc[ ] 연산자

: loc[ ]은 명칭 기반으로 데이터를 추출함.

- 행 위치에는 DataFrame index값을, 열 위치에는 칼럼 명을 입력해야 함.

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

```
'Chulmin'
```

```
1 data_df_reset.loc[1, 'Name']
```

```
'Chulmin'
```

```
# 다음 코드는 오류를 발생합니다.
data_df_reset.loc[0, 'Name']
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/range.py in get_loc(self, key, method, tolerance)
    390                 try:
--> 391                     return self._range.index(new_key)
    392                 except ValueError as err:

ValueError: 0 is not in range

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

KeyError                                  Traceback (most recent call last)
                                  3 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/range.py in get_loc(self, key, method, tolerance)
    391                     return self._range.index(new_key)
    392                 except ValueError as err:
--> 393                     raise KeyError(key) from err
    394             self._check_indexing_error(key)
    395             raise KeyError(key)

KeyError: 0
```

```
# 명칭 기반 인덱싱과 위치 기반 인덱싱에서 슬라이싱을 적용할 때의 차이점

# print('명칭 기반 ix slicing \n', data_df.ix['one': 'two', 'Name'], '\n')
print('위치 기반 iloc slicing \n', data_df.iloc[0:1, 0], '\n')
print('명칭 기반 loc slicing \n', data_df.loc['one': 'two', 'Name'])
```

```
위치 기반 iloc slicing
 one    Chulmin
Name: Name, dtype: object

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

```
print(data_df_reset.loc[1:2, 'Name'])
```

```
1     Chulmin
2    Eunkyung
Name: Name, dtype: object
```

```
print(data_df.ix[1:2, 'Name'])
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-83-b9b0ccdeb773> in <cell line: 1>()
----> 1 print(data_df.ix[1:2, 'Name'])

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in __getattr__(self, name)
   5900         ):
   5901             return self[name]
-> 5902         return object.__getattribute__(self, name)
   5903
   5904     def __setattr__(self, name: str, value) -> None:

AttributeError: 'DataFrame' object has no attribute 'ix'
```

ix, iloc, loc의 문제점 및 주의할 점

- ix[ ]는 명칭 기반 인덱싱과 위치 기반 인덱싱 **모두** 적용 가능
  - DataFrame의 인덱스가 숫자형일 경우 행 위치에 오는 숫자는 **명칭 기반** 인덱싱의 DataFrame 인덱스임.
- iloc[ ]은 **위치 기반** 인덱싱만 가능.
  - 행과 열 위치 값으로 정수형 값을 지정해 원하는 데이터를 반환.
- loc[ ]은 **명칭 기반** 인덱싱만 가능.
  - 행 위치에 DataFrame 인덱스가 오며, 열 위치에는 칼럼 명을 지정해 원하는 데이터를 반환함.
- 명칭 기반 인덱싱에서 슬라이싱을 '시작점:종료점'으로 지정할 때, 시작점에서 종료점을 포함한 위치에 있는 데이터를 반환.

불린 인덱싱 : 데이터 필터링 방식

- 처음부터 가져올 값을 조건으로 ix[ ] 내에 입력하면 자동으로 원하는 값을 필터링함.
- iloc[ ] 정수형 값이 아닌 불린 값에 대해서는 지원하지 않기에 불린 인덱싱이 지원되지 않음.

```
titanic_df = pd.read_csv('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	Tic
33	34	0	2	Wheadon, Mr...	male	66.0	0	0	24
54	55	0	1	Ostby, Mr. ...	male	65.0	0	1	113
96	97	0	1	Goldschmidt...	male	71.0	0	0	17
116	117	0	3	Connors, Mr...	male	70.5	0	0	370
170	171	0	1	Van der hoe...	male	61.0	0	0	111
252	253	0	1	Stead, Mr. ...	male	62.0	0	0	113
275	276	1	1	Andrews, Mi...	female	63.0	1	0	13
280	281	0	3	Duane, Mr. ...	male	65.0	0	0	336
326	327	0	3	Nysveen, Mr...	male	61.0	0	0	345
438	439	0	1	Fortune, Mr...	male	64.0	1	4	19
456	457	0	1	Millet, Mr....	male	65.0	0	0	13
483	484	1	3	Turkula, Mr...	female	63.0	0	0	4
493	494	0	1	Artagaveyti...	male	71.0	0	0	17
545	546	0	1	Nicholson, ...	male	64.0	0	0	
555	556	0	1	Wright, Mr....	male	62.0	0	0	113
570	571	1	2	Harris, Mr....	male	62.0	0	0	S.W
625	626	0	1	Sutton, Mr....	male	61.0	0	0	36
630	631	1	1	Barkworth, ...	male	80.0	0	0	27
672	673	0	2	Mitchell, M...	male	70.0	0	0	24
745	746	0	1	Crosby, Cap...	male	70.0	1	1	W 5
829	830	1	1	Stone, Mrs....	female	62.0	0	0	113
851	852	0	3	Svensson, M...	male	74.0	0	0	347

Next steps: View recommended plots

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

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

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

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

복합 조건 결합 이용

1. and 조건일 때 - **&**
2. or 조건일 때 - |
3. not 조건일 때 - **~**

```
1 # 나이가 60세 이상, 선실 등급이 1등급, 성별이 여성인 승객 추출
2 # 개별 조건은 ()로 묶고, 복합 조건 연산자 사용
3
4 titanic_df[(titanic_df['Age'] > 60) & (titanic_df['Pclass'] == 1) & (titanic
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
275	276	1	1	Andrews, Mi...	female	63.0	1	0	13502
829	830	1	1	Stone, Mrs....	female	62.0	0	0	113572

```
1 # 개별 조건을 변수에 할당하고, 결합한 불린 인덱싱 수행
2 cond1 = titanic_df['Age'] > 60
3 cond2 = titanic_df['Pclass'] == 1
4 cond3 = titanic_df['Sex'] == 'female'
5 titanic_df[cond1 & cond2 & cond3]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
275	276	1	1	Andrews, Mi...	female	63.0	1	0	13502
829	830	1	1	Stone, Mrs....	female	62.0	0	0	113572

[정렬, Aggregation 함수, GroupBy 적용]

DataFrame, Series의 정렬 - **sort_values( )**

- 주요 입력 파라미터
  - **by** : 특정 칼럼을 입력하면 해당 칼럼으로 정렬 수행
  - **ascending**
    - ascending = True로 설정 : 오름차순으로 정렬
    - ascending = False로 설정 : 내림차순으로 정렬
  - **inplace**
    - inplace = True로 설정 : 호출한 DataFrame의 정렬 결과를 그대로 적용
    - inplace = **False**로 설정(기본값) : sort_values()를 호출한 DataFrame은 그대로 유지하며 정렬된 DataFrame을 결과로 반환.

```
1 # 오름차순 정렬, 반환
2 titanic_sorted = titanic_df.sort_values(by=['Name'])
3 titanic_sorted.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
845	846	0	3	Abbing, Mr....	male	42.0	0	0	C.A. 5547	
746	747	0	3	Abbott, Mr....	male	16.0	1	1	C.A. 2673	2
279	280	1	3	Abbott, Mrs...	female	35.0	1	1	C.A. 2673	2

Next steps: View recommended plots

```
1 # 내림차순 정렬, 반환
2 titanic_sorted = titanic_df.sort_values(by=['Pclass', 'Name'], ascending = F
3 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

Next steps: View recommended plots

Aggregation 함수 적용

DataFrame의 경우 바로 aggregation을 호출할 경우 모든 칼럼에 해당 aggregation을 적용함.

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

```
1 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
```

```
1 titanic_df[['Age', 'Fare']].mean() # 평균 계산
```

```
Age     29.699118
Fare    32.204208
dtype: float64
```

groupby( ) 적용

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

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

```
1 titanic_groupby = titanic_df.groupby('Pclass').count()
2 titanic_groupby
```

Pclass	PassengerId	Survived	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cal
1	216	216	216	216	186	216	216	216	216	
2	184	184	184	184	173	184	184	184	184	
3	491	491	491	491	355	491	491	491	491	

Next steps: View recommended plots

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

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

Next steps: View recommended plots

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

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

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

Pclass	Age	SibSp	Fare
1	80.0	90	84.154687

[결손 데이터 처리하기]

- 결손 데이터 = NULL 인 경우
- 넘파이의 NaN으로 표시
  - 머신러닝 알고리즘은 NaN값을 처리하지 않으므로 이 값을 다른 값으로 대체해야 함.
  - 평균, 총합 등의 함수 연산 시 제외됨.
- **isna( )** : NaN 여부를 확인
- **fillna( )** : NaN 값을 다른 값으로 대체.

isna( )로 결손 데이터 여부 확인

: 데이터가 NaN인지 아닌지를 알려줌. (True/False)

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

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

```
1 titanic_df.isna().sum()
2 # True = 1, False = 0
```