AI활용 표현과 문제해결

[Pandas & Pytorch]

허세훈

충남대학교
지능소프트웨어연구실

2021.03.15

지난 주 Review

개발환경 구축

Pycharm, Anaconda, Jupyter Notebook 등을 설치

Anaconda 가상환경

Anaconda 가상환경을 사용하는 이유, 가상환경 사용법 등에 대해서 공부

Numpy 설치 및 사용법

Numpy 설치 후, Numpy 배열을 다루는 방법, Numpy에서 지원하는 다양한 함수들 공부

지난 주 | 가상환경 관련 명령어 오류 정정

1주차 수업자료 P.21

Anaconda | 가상환경 관련 명령어들

	명령어 형태	예시
가상환경 생성	conda create –n {가상환경이름} python={python 버전}	conda create –n AI python=3.8
가상환경 목록 확인	conda env list	conda env list
가상환경 활성화	conda activate {가상환경 이름}	conda activate AI
가상환경 비활성화	conda deactivate ~~{가상환경이름}~~	conda deactivate ~~AI~~

가상환경 비활성화를 위해서는
“conda deactivate”만 입력하면 됩니다.

지난 주 | Numpy 관련 오류 정정

1주차 수업자료 P.29

```
import numpy as np

target_arr = np.array([11, 22, 33, 44, 55, 66, 77, 88, 99])
idx = np.array([0, 2, 4, 6, 8]) # 선택하고 싶은 Index들을 np.array 형태에 저장합니다.
print(target_arr[idx]) # [11, 33, 55, 77, 99]
```

~~[주의] Index를 위한 배열은 Numpy Array 형태이어야 함~~

리스트로도 Indexing을 할 수 있지만 추천하지는 않습니다.

Warning: Augmented assignment - the operators like "+=" - works, but it does not necessarily do what you would expect. In particular, repeated indices do not result in the value getting added twice:

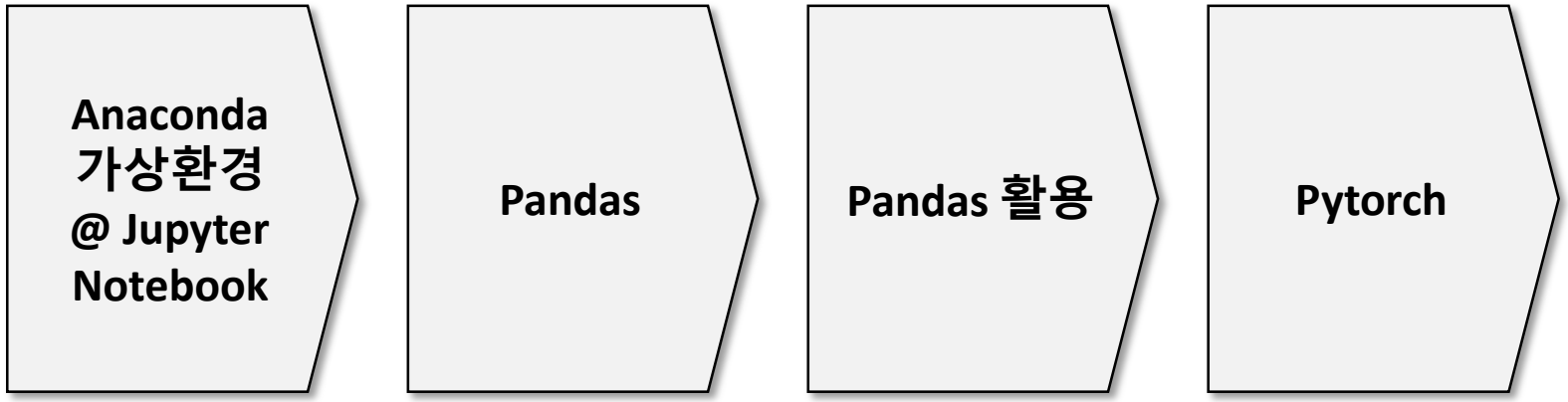
np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])

In [ ]:

```
>>> T = A.copy()
>>> T[ [0,1,2,3,3,3] ] += 10
>>> T
array([10, 11, 12, 13,  4,  5,  6,  7,  8,  9])
```

출처 : https://scipy-cookbook.readthedocs.io/items/Indexing.html

이번 주 실습 내용



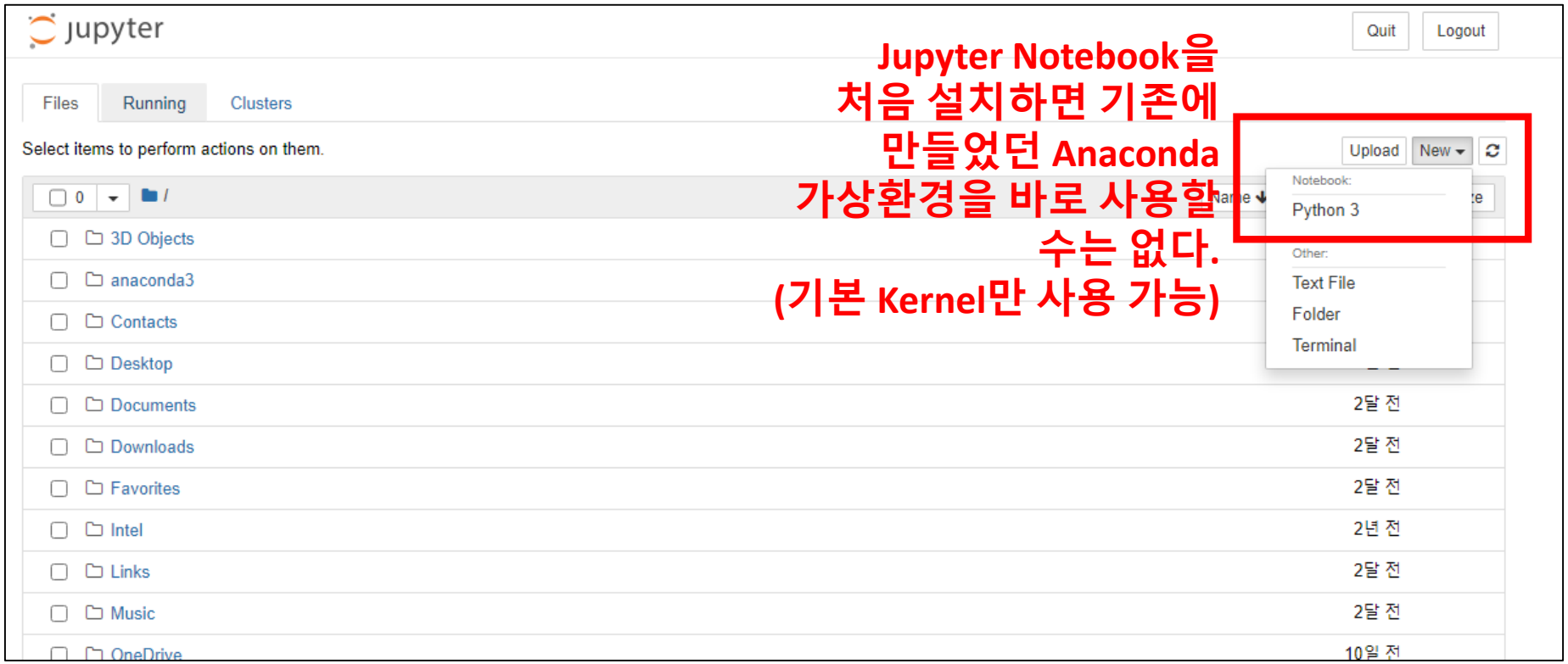
- Jupyter Notebook에서 Anaconda Env 사용하기

- 소개 및 설치
- Series와 DataFrame
- 데이터 확인 및 선택하기

- 결측치 처리
- 데이터 분석

- 소개 및 설치
- 텐서의 종류와 차원
- 텐서 다루기

Anaconda 가상환경 @ Jupyter Notebook



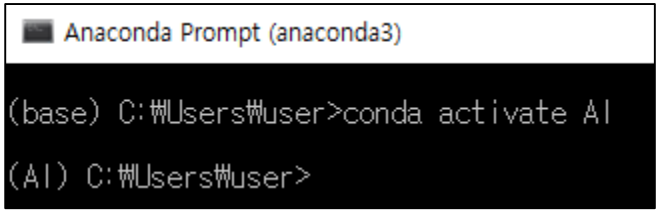
Jupyter Notebook을 처음 설치하면 기존에 만들었던 Anaconda 가상환경을 바로 사용할 수는 없다. (기본 Kernel만 사용 가능)

Jupyter Notebook과 Anaconda 가상환경간의 호환을 위해,
가상환경마다 Jupyter Notebook에서의 Kernel을 생성해주어야 함

Anaconda 가상환경 추가 방법 @ Jupyter Notebook

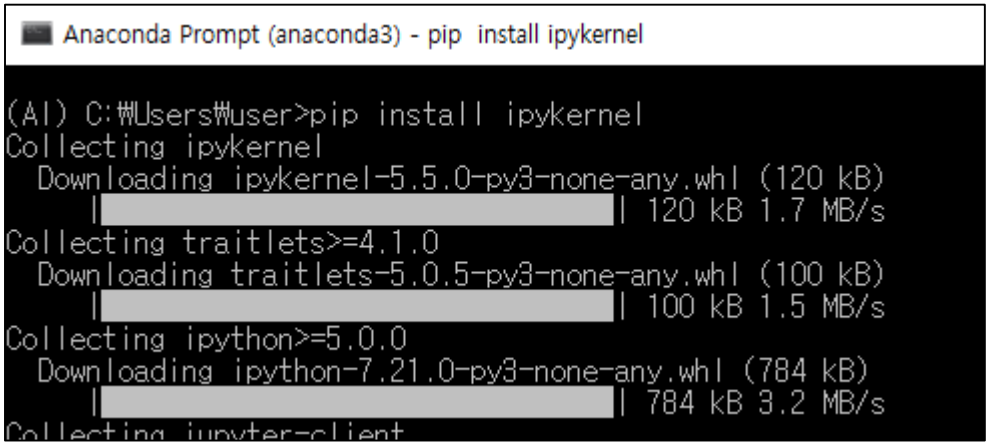
1. Anaconda 가상환경 활성화

conda activate {가상환경 이름}



2. ipykernel 설치 → 반드시 pip를 통해서 설치해주세요

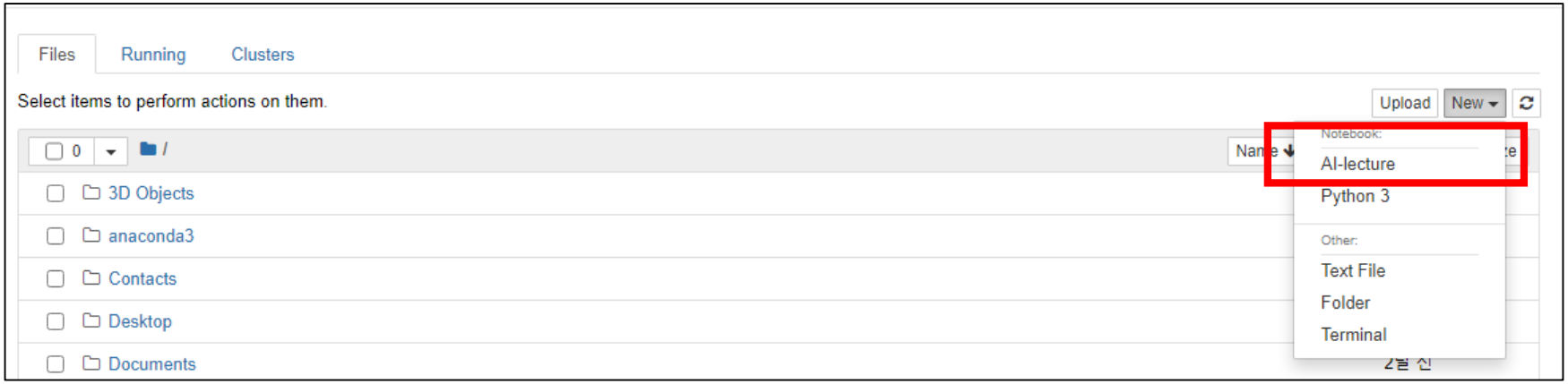
pip install ipykernel



Anaconda 가상환경 추가 방법 @ Jupyter Notebook

3. ipykernel을 통해 Anaconda 가상환경을 Jupyter Notebook에 연결하기

python -m ipykernel install --user --name {가상환경 이름}
--display-name “jupyter에서 보여질 이름”



Pandas | 소개

대용량 데이터 처리 가능

Pandas를 이용하면 GB 단위 이상의 대용량 데이터를 다룰 수 있음

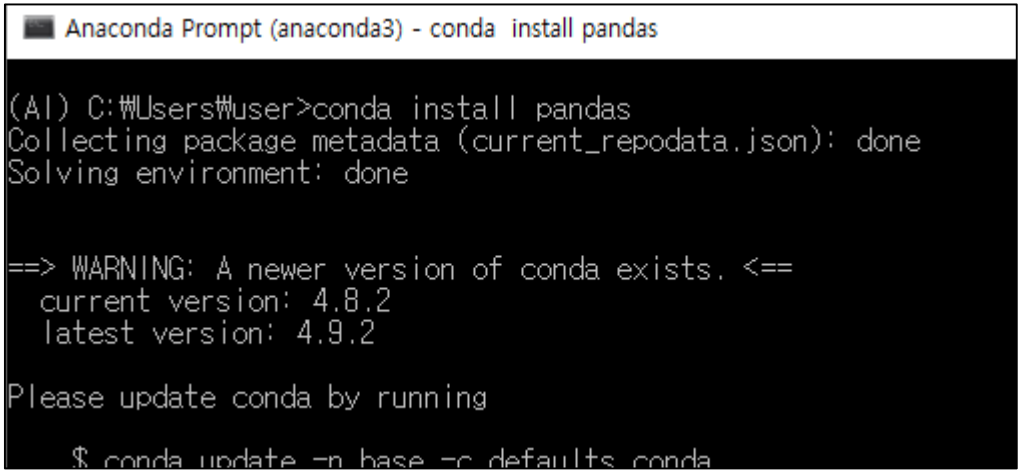
복잡한 데이터 처리 작업들을 비교적 쉽게 해줌

데이터를 합치고 관계 연산을 수행하는 기능들을 제공 + 누락 데이터 등을 처리할 수 있는 기능들을 제공

Pandas | 설치

1. Anaconda Prompt 창에서 설치 명령어 입력

```
conda install pandas
```



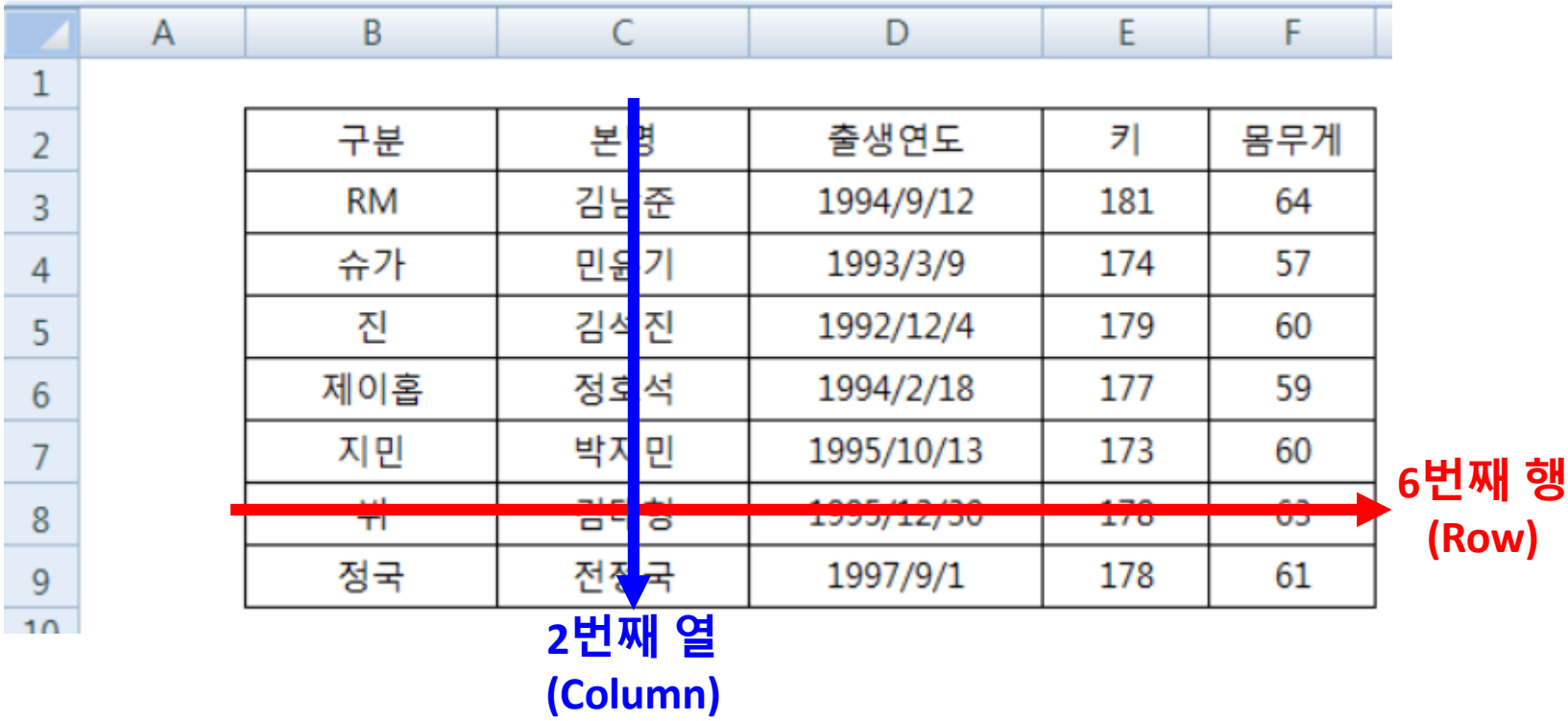
2. Import를 통해 사용

```
import pandas as pd # pandas를 pd라는 이름으로 불러옴

ser = pd.Series([1, 2, 3, 4])
print(ser)
```

Pandas | 제공하는 자료구조

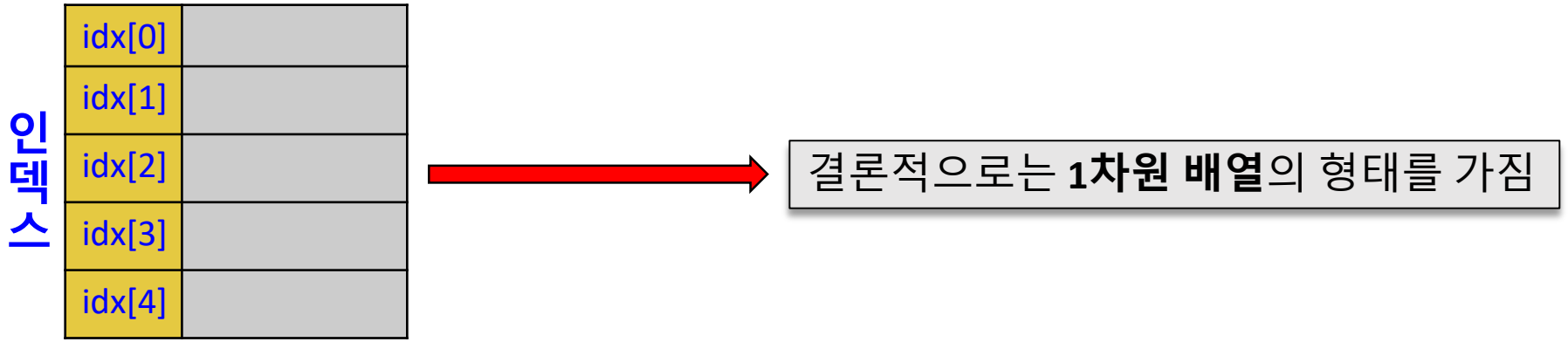
Pandas는 기본적으로 “엑셀”과 비슷한 형태의 자료구조들을 지원



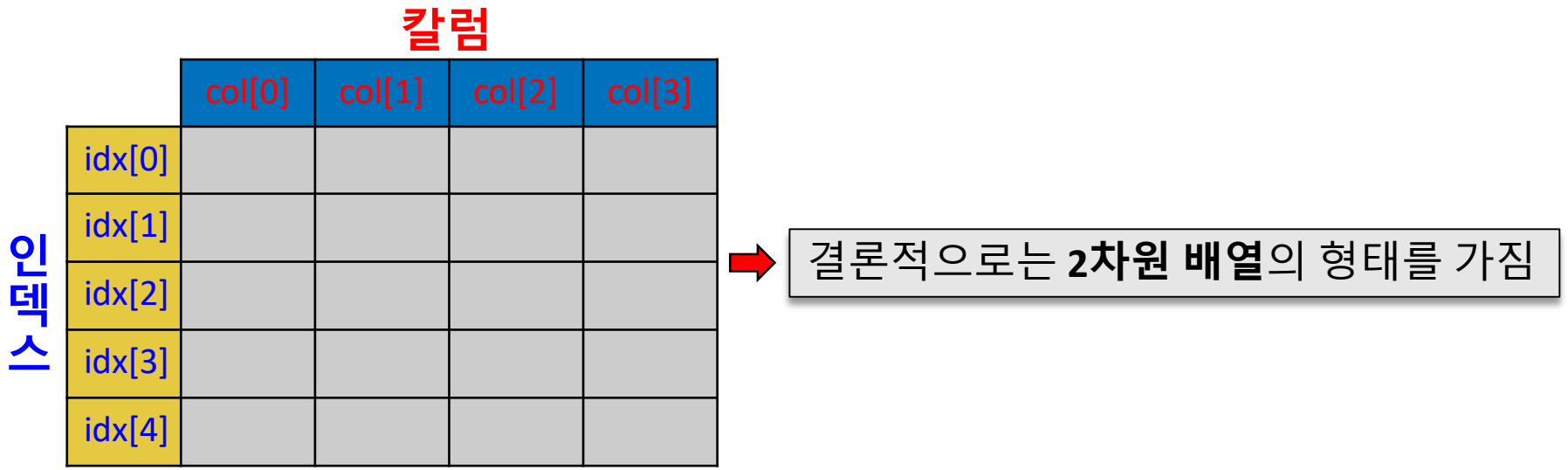
Pandas에서는 **“Series”와 “DataFrame”이라는 자료구조를 제공**함으로써 데이터 분석을 도와준다.

Pandas | Series와 DataFrame의 개념

시리즈(Series) : **인덱스**에 의해 데이터가 저장되고 검색된다.



데이터프레임(DataFrame) : **인덱스**와 **칼럼**에 의해 데이터가 저장되고 검색된다.



Pandas | Series 생성 방법

1. Python의 리스트를 통해 생성하기

Index가 자동으로 생성됨

```
s = pd.Series([1, 3, 5, 7, 9])
s
```

```
0    1
1    3
2    5
3    7
4    9
dtype: int64
```

2. Numpy의 Array를 통해 생성하기

Index가 자동으로 생성됨

```
s3 = pd.Series(np.array([1, 3, 5, 7, 9]))
s3
```

```
0    1
1    3
2    5
3    7
4    9
dtype: int32
```

Pandas | Series 생성 방법

3. Python의 Dictionary를 통해 생성하기

Dictionary의 키 값이 Series의 Index가 됨

```
s2 = pd.Series({'a' : 1, 'b': 2, 'c' : 3})
s2
```

```
a    1
b    2
c    3
dtype: int64
```

참고) Numpy Array와 Python List를 통해 생성할 때도 Index 지정 가능

```
s4 = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
s4
```

```
a    1
b    2
c    3
dtype: int64
```

```
s5 = pd.Series(np.array([1, 2, 3]), index=['a', 'b', 'c'])
s5
```

```
a    1
b    2
c    3
dtype: int32
```

```
s6 = pd.Series(np.array([1, 2, 3]), index=np.array(['a', 'b', 'c']))
s6
```

```
a    1
b    2
c    3
dtype: int32
```

Pandas | DataFrame 생성 방법

1. Python의 리스트를 통해 생성하기

```
data = [1, 3, 5, 7, 9]

d = pd.DataFrame(data)
d
```

	0
0	1
1	3
2	5
3	7
4	9

Series와 다르게
Column이 생성됨

```
data = [[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]

d1 = pd.DataFrame(data)
d1
```

	0	1	2	3	4
0	1	3	5	7	9
1	2	4	6	8	10

2. Numpy의 Array를 통해 생성하기

```
data = np.array([[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]])

d2 = pd.DataFrame(data)
d2
```

	0	1	2	3	4
0	1	3	5	7	9
1	2	4	6	8	10

Pandas | DataFrame 생성 방법

3. Python의 Dictionary를 통해 생성하기

```
data = {
    "name" : ["Jung", "Kim", "Park", "Hu", "Jung"],
    "year" : [2013, 2011, 2012, 2015, 2016],
    "points" : [1.5, 2.3, 1.6, 3.2, 2.8]
}

d3 = pd.DataFrame(data)
d3
```

	name	year	points
0	Jung	2013	1.5
1	Kim	2011	2.3
2	Park	2012	1.6
3	Hu	2015	3.2
4	Jung	2016	2.8

Dictionary의 키 값이
DataFrame의 Column이 됨

Pandas | 데이터 확인하기

```
index = ["가", "나", "다", "라", "마", "바"]
columns = list('ABCD')

df = pd.DataFrame(np.random.randn(6,4), index=index, columns=columns)
df
```

	A	B	C	D
가	1.003856	0.372504	1.277486	0.298482
나	1.118750	-1.658703	-0.577452	-0.549368
다	1.875092	0.319789	-0.607596	-1.411642
라	0.602278	-0.763088	-0.938394	-1.889965
마	0.170081	0.503280	-0.958260	-0.462446
바	0.245095	-1.108389	-1.343866	0.434252

(1) 데이터의 속성 확인

```
df.index # DataFrame의 Index 확인
```

```
Index(['가', '나', '다', '라', '마', '바'], dtype='object')
```

```
df.columns # DataFrame의 Column 확인
```

```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

```
df.values # DataFrame의 값 확인
```

```
array([[ 1.00385566,  0.37250356,  1.27748568,  0.29848225],
       [ 1.11874957, -1.65870294, -0.57745177, -0.54936771],
       [ 1.87509202,  0.31978931, -0.60759588, -1.41164165],
       [ 0.60227827, -0.76308786, -0.93839408, -1.88996523],
       [ 0.17008115,  0.50328046, -0.95825977, -0.46244589],
       [ 0.24509499, -1.10838852, -1.34386598,  0.43425223]])
```

Pandas | 데이터 확인하기

(2) 맨 앞의 데이터 또는 맨 뒤의 데이터 확인하기

```
df.head()
```
첫 5개 행의 데이터를 보여줌

	A	B	C	D
가	1.003856	0.372504	1.277486	0.298482
나	1.118750	-1.658703	-0.577452	-0.549368
다	1.875092	0.319789	-0.607596	-1.411642
라	0.602278	-0.763088	-0.938394	-1.889965
마	0.170081	0.503280	-0.958260	-0.462446

```
df.tail()
```
마지막 5개 행의 데이터를 보여줌

	A	B	C	D
나	1.118750	-1.658703	-0.577452	-0.549368
다	1.875092	0.319789	-0.607596	-1.411642
라	0.602278	-0.763088	-0.938394	-1.889965
마	0.170081	0.503280	-0.958260	-0.462446
바	0.245095	-1.108389	-1.343866	0.434252

맨 앞의 3개만 보고 싶다면?

```
df.head(3)
```

	A	B	C	D
가	1.003856	0.372504	1.277486	0.298482
나	1.118750	-1.658703	-0.577452	-0.549368
다	1.875092	0.319789	-0.607596	-1.411642

Pandas | 데이터의 통계량 확인하기

```
df.describe()
```

각 Column에 대하여

	A	B	C	D	
count	6.000000	6.000000	6.000000	6.000000	개수
mean	0.835859	-0.389101	-0.524680	-0.596781	평균값
std	0.637978	0.910828	0.926032	0.918819	표준편차
min	0.170081	-1.658703	-1.343866	-1.889965	최소값
25%	0.334391	-1.022063	-0.953293	-1.196073	사분위수
50%	0.803067	-0.221649	-0.772995	-0.505907	
75%	1.090026	0.359325	-0.584988	0.108250	
max	1.875092	0.503280	1.277486	0.434252	최대값

Pandas | 데이터 선택하기

(1) Column 명을 이용하여 데이터 선택하기

```
df['A']
```

```
가    1.003856
나    1.118750
다    1.875092
라    0.602278
마    0.170081
바    0.245095
Name: A, dtype: float64
```

```
df.A
```

```
가    1.003856
나    1.118750
다    1.875092
라    0.602278
마    0.170081
바    0.245095
Name: A, dtype: float64
```

(2) Row의 범위를 이용하여 데이터 선택하기

```
df[0:3] # df[시작 위치:(마지막위치+1)]
# df["가":"다"] # df["시작 index 이름":"마지막 index 이름"]
```

	A	B	C	D
가	1.138224	-0.645965	0.931917	1.223539
나	1.630241	-0.194415	-1.041876	-0.686106
다	-0.253868	0.776370	0.848227	-0.208126

- df[시작 위치:마지막위치+1]
- df[“시작 index이름” : “마지막 index이름”]

주의) '가'와 같이 인덱스 이름을 통해 데이터를 가져오기 위해서 df['가']라고 하면 KeyError가 발생
→ DataFrame은 Column을 Key 값으로 갖기 때문
→ 따라서, df[칼럼명]의 형태로만 사용 가능

Pandas | 데이터 선택하기

(3) Index의 이름(Row의 이름)으로 데이터 선택하기

```
df.loc['가']
```

```
A    1.003856
B    0.372504
C    1.277486
D    0.298482
Name: 가, dtype: float64
```

```
df.loc['가', 'B']
```

```
0.3725035594331696
```

특정 Index, 특정 열의 이름으로 데이터 선택

```
df.loc[:, ['A', 'B']]
```

	A	B
가	1.003856	0.372504
나	1.118750	-1.658703
다	1.875092	0.319789
라	0.602278	-0.763088
마	0.170081	0.503280
바	0.245095	-1.108389

여러 개의 칼럼명을 이용하여 데이터를 선택

Pandas | 데이터 선택하기

(4) 위치를 기준으로 데이터 선택하기

```
df.iloc[3] # 4번째 행의 데이터들 선택
```

```
A    0.602278
B   -0.763088
C   -0.938394
D   -1.889965
Name: 라, dtype: float64
```

```
df.iloc[0:2, 3:4] # 0~1번째 행과 3번째 열의 데이터들 선택
```

	D
가	0.298482
나	-0.549368

```
df.iloc[:, 2] # 3번째 열의 데이터들 선택
```

```
가    1.277486
나   -0.577452
다   -0.607596
라   -0.938394
마   -0.958260
바   -1.343866
Name: C, dtype: float64
```

Pandas | 데이터 선택하기

(5) 조건을 이용하여 데이터 선택하기

```
df[df['B'] > 0] # B라는 열에 들어있는 값이 양수인 행들만 선택
```

	A	B	C	D
가	1.003856	0.372504	1.277486	0.298482
다	1.875092	0.319789	-0.607596	-1.411642
마	0.170081	0.503280	-0.958260	-0.462446

```
df[df > 0] # 값이 양수인 경우 원래의 값들이 보여지고 나머지 값들은 NaN으로 보임
```

	A	B	C	D
가	1.003856	0.372504	1.277486	0.298482
나	1.118750	NaN	NaN	NaN
다	1.875092	0.319789	NaN	NaN
라	0.602278	NaN	NaN	NaN
마	0.170081	0.503280	NaN	NaN
바	0.245095	NaN	NaN	0.434252

Pandas | 데이터 선택하기

```
df['E'] = ['A', 'I', '활', '용', '표', '현']
df
```

	A	B	C	D	E
가	1.003856	0.372504	1.277486	0.298482	A
나	1.118750	-1.658703	-0.577452	-0.549368	I
다	1.875092	0.319789	-0.607596	-1.411642	활
라	0.602278	-0.763088	-0.938394	-1.889965	용
마	0.170081	0.503280	-0.958260	-0.462446	표
바	0.245095	-1.108389	-1.343866	0.434252	현

```
df[df['E'].isin(['표', '현'])] # isin() 함수를 이요해서 '표', '현'이 들어간 행들의 데이터들 선택
```

	A	B	C	D	E
마	0.170081	0.503280	-0.958260	-0.462446	표
바	0.245095	-1.108389	-1.343866	0.434252	현

Pandas 활용 | 결측치 처리

결측치(Missing Value)란?

데이터의 수집 과정에서 발생한 오류 등으로 인해 **누락된 값** 또는 **비어 있는 값**을 의미
→ 보통의 경우, **NaN 값으로 표기**함
→ 데이터를 분석할 때, 걸림돌이 되므로 처리해주어야 함

	성별	나이	학점
Kim	여	23	3.8
Jung	남	32	2.7
Hu	남	25	
Park		21	3.9
Lee	여	22	4.1

Pandas 활용 | 결측치 처리

```
index = ["가", "나", "다", "라", "마", "바"]
columns = list('ABCD')

missing_df = pd.DataFrame(np.random.randn(6,4), index=index, columns=columns)
missing_df['E'] = [1, np.nan, 3.5, 6.1, np.nan, 7.0]

missing_df
```

	A	B	C	D	E
가	0.481304	-2.232060	-0.963896	0.683263	1.0
나	0.449869	-0.818211	-0.844325	1.122499	NaN
다	-0.047973	-1.051871	-0.810267	0.352061	3.5
라	-1.020223	-0.490100	-0.915464	0.787651	6.1
마	0.186024	1.560437	-1.145343	-0.587345	NaN
바	0.049458	0.139444	0.507654	0.851807	7.0

Pandas 활용 | 결측치 처리

방법 1. 결측치가 발생한 행을 제거

```
missing_df.dropna(how='any')
# 행의 값 중 하나라도 NaN 값이 있을 경우, 그 행 자체를 제거
```

	A	B	C	D	E
가	0.481304	-2.232060	-0.963896	0.683263	1.0
다	-0.047973	-1.051871	-0.810267	0.352061	3.5
라	-1.020223	-0.490100	-0.915464	0.787651	6.1
바	0.049458	0.139444	0.507654	0.851807	7.0

```
missing_df.dropna(how='all')
# 행의 값이 모두 NaN 값인 경우, 그 행을 제거
```

	A	B	C	D	E
가	0.481304	-2.232060	-0.963896	0.683263	1.0
나	0.449869	-0.818211	-0.844325	1.122499	NaN
다	-0.047973	-1.051871	-0.810267	0.352061	3.5
라	-1.020223	-0.490100	-0.915464	0.787651	6.1
마	0.186024	1.560437	-1.145343	-0.587345	NaN
바	0.049458	0.139444	0.507654	0.851807	7.0

Pandas 활용 | 결측치 처리

방법 2. 결측치에 특정 값 채워넣기

```
missing_df.fillna(value=0.5)
# 결측치가 발생한 곳에 0.5를 넣는다
```

	A	B	C	D	E
가	0.481304	-2.232060	-0.963896	0.683263	1.0
나	0.449869	-0.818211	-0.844325	1.122499	0.5
다	-0.047973	-1.051871	-0.810267	0.352061	3.5
라	-1.020223	-0.490100	-0.915464	0.787651	6.1
마	0.186024	1.560437	-1.145343	-0.587345	0.5
바	0.049458	0.139444	0.507654	0.851807	7.0

주로, 평균 값과 같이 통계량으로 NaN 값을 채운다.

```
missing_df.fillna(value=missing_df.mean())
```

	A	B	C	D	E
가	0.481304	-2.232060	-0.963896	0.683263	1.0
나	0.449869	-0.818211	-0.844325	1.122499	4.4
다	-0.047973	-1.051871	-0.810267	0.352061	3.5
라	-1.020223	-0.490100	-0.915464	0.787651	6.1
마	0.186024	1.560437	-1.145343	-0.587345	4.4
바	0.049458	0.139444	0.507654	0.851807	7.0

Pandas 활용 | 데이터 분석

(1) 행방향으로의 또는 열방향으로의 합 구하기

	A	B	C	D
가	0.934872	-0.282359	-1.064603	0.794630
나	1.427002	-1.763495	-0.681890	1.774112
다	-1.014097	1.215559	0.274770	-0.142746
라	1.354790	1.907308	1.007737	0.275456
마	0.094142	0.707011	-0.947351	0.414702
바	1.057494	-0.955807	0.164436	-3.258582

열 방향(axis=1)

```
target_df.sum(axis=1)
# 열방향으로의 합 (각 행의 합)
```

```
가    0.382540
나    0.755729
다    0.333486
라    4.545292
마    0.268503
바   -2.992459
dtype: float64
```

행 방향(axis=0)

```
target_df.sum(axis=0)
# 행방향으로의 합 (각 열의 합)
```

```
A    3.854201
B    0.828217
C   -1.246901
D   -0.142428
dtype: float64
```

Pandas 활용 | 데이터 분석

(2) 이외의 데이터 분석 관련 함수들

함수	설명	예시
count()	데이터의 개수 계산	df.count()
min() max()	최소값 계산 최대값 계산	df.min() df.max()
argmin() argmax()	최소값의 위치 반환 최대값의 위치 반환	df.argmin() df.argmax()
idxmin() idxmax()	인덱스 중 최소값 반환 인덱스 중 최대값 반환	df.idxmin() df.idxmax()
quantile()	사분위수 값 반환	df.quantile()
mean()	평균 계산	df.mean()
median()	중간값 계산	df.median()
std() var()	표준편차 계산 분산 계산	df.std() df.var()
cov() corr()	공분산 계산 상관계수 계산	df['A'].cov(df['B']) df['A'].corr(df['B'])

PyTorch | 소개

딥러닝 관련 연산 자동화
및 GPU 연산 지원

역전파 알고리즘을 위한 미분 기울기 계산 자동화
+ GPU를 활용한 병렬처리 연산 기능 제공

객체 지향적인 방식을
통한 심층 신경망 구현

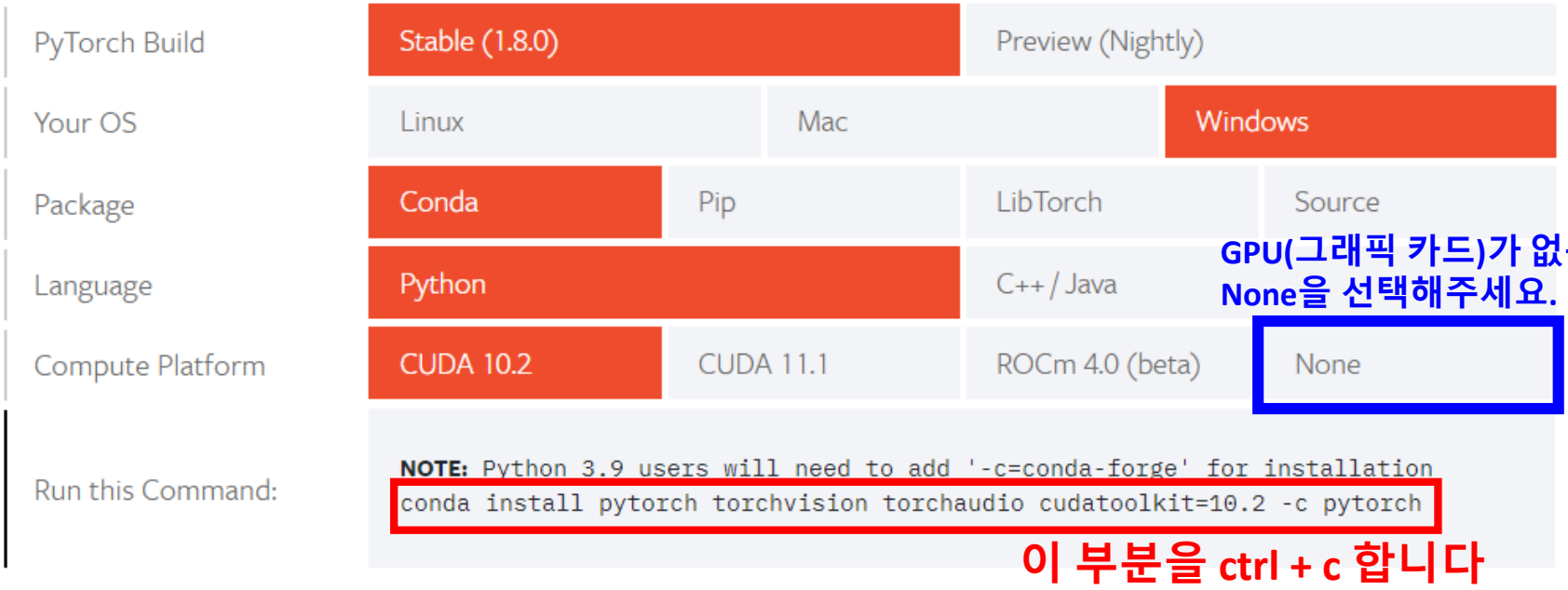
복잡한 심층 신경망 코드의 Abstraction /
Reusability / Readability 등을 높이기 위해
객체지향적인 코드 스타일을 적용

PyTorch | 설치

1. PyTorch 홈페이지의 Get Started 메뉴로 이동합니다.

https://pytorch.org/get-started/locally/

2. 자신의 운영체제 및 환경에 맞는 명령어를 복사해옵니다.



GPU(그래픽 카드)가 없는 경우, None을 선택해주세요.

이 부분을 ctrl + c 합니다

PyTorch | 설치

3. Anaconda Prompt에서 명령어를 Ctrl + v한 다음, 실행합니다.

```
(AI) C:₩Users₩tpgns>conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.8.3
  latest version: 4.9.2

Please update conda by running

    $ conda update -n base -c defaults conda
```

4. PyTorch를 import하여 사용합니다.

```
import torch # PyTorch를 Import합니다.

x = torch.Tensor([23, 22, 11, 33])
x
```

```
tensor([23., 22., 11., 33.])
```

PyTorch | 텐서(Tensor)

텐서(Tensor)란?

다차원 배열로 Numpy Array와 유사하다.
하지만 Tensor는 Numpy Array와는 달리, 연산 속도를 극대화시키기 위해서 GPU 연산을 지원한다는 점에서 차이가 있다.

텐서의 종류 – 1차원 텐서

```
x = torch.FloatTensor([23, 24, 25.5, 26, 27.2])
x
```

```
tensor([23.0000, 24.0000, 25.5000, 26.0000, 27.2000])
```

1차원 배열과 같은 형태

텐서의 종류 – 2차원 텐서

```
x = torch.zeros(5, 3)
print(x)
print(x.shape)
```

```
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]])
torch.Size([5, 3])
```

2차원 배열과 같은 형태

PyTorch | 텐서(Tensor)

텐서의 종류 – 3차원 텐서

```
import torch
from PIL import Image

# PIL 라이브러리를 활용해서 디스크에서 이미지를 읽고 Numpy 배열로 변환
image = np.array(Image.open('a.jpeg'))

tensor = torch.from_numpy(image) # Numpy를 통해 Tensor 생성
print(tensor.shape)
```

```
torch.Size([600, 1200, 3])
```

3차원 배열과 같은 형태

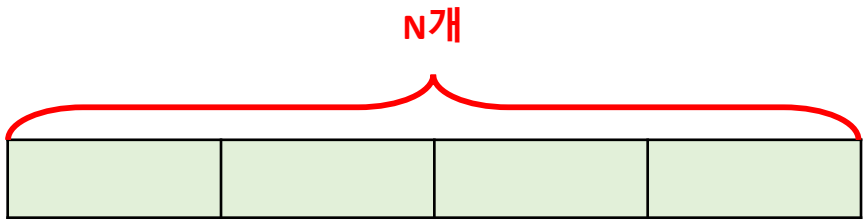
텐서의 종류 – N차원 텐서

torch.Size([a, b, c, d]) ⟶ 4차원 배열과 같은 형태

torch.Size([a, b, c, d, e]) ⟶ 5차원 배열과 같은 형태

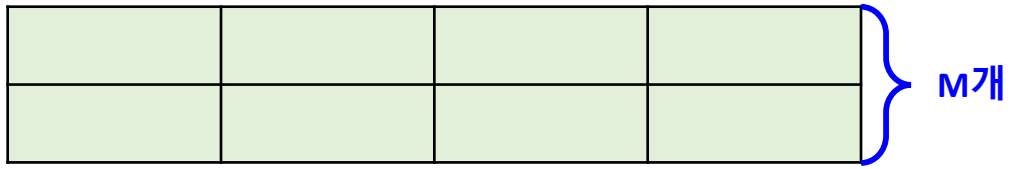
⋮ ⋮

PyTorch | 차원



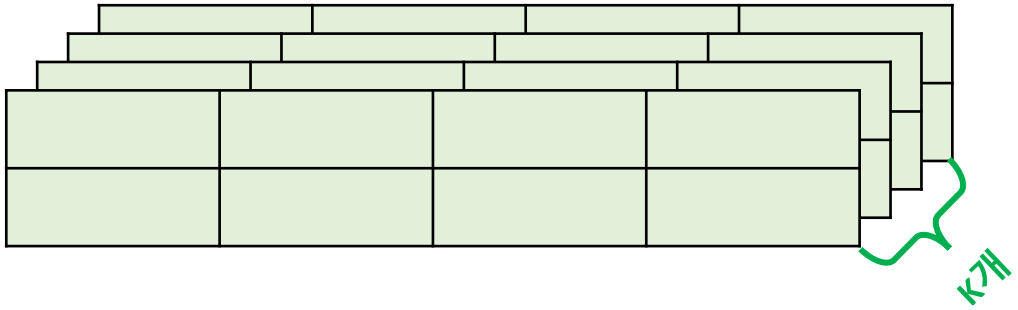
torch.Tensor([1, 2, 3, ...]) … 1차원

⇒ torch.Size([N])



torch.Tensor([[1, 2, 3, ...], … 2차원
[1, 2, 3, ...]])

⇒ torch.Size([N, M])



torch.Tensor([[[1, 2, 3, ...], … 3차원
[1, 2, 3, ...]],
[[1, 2, 3, ...],
[1, 2, 3, ...]]])

⇒ torch.Size([N, M, K])

⋮

⋮

PyTorch | 텐서(Tensor)

텐서의 종류 – GPU지원 텐서

```
a = torch.rand(1000, 100000)
b = torch.rand(1000, 100000)

# GPU가 사용가능한지 확인
if torch.cuda.is_available():
    # cuda() 함수를 통해 gpu tensor로 변환
    a = a.cuda()
    b = b.cuda()

print(a)
print(b)
```

```
tensor([[0.8591, 0.3886, 0.0135,  ..., 0.1025, 0.1788, 0.8606],
        [0.9453, 0.2467, 0.2187,  ..., 0.8200, 0.9189, 0.2654],
        [0.5102, 0.0638, 0.9669,  ..., 0.7215, 0.7844, 0.9244],
        ...,
        [0.8385, 0.8985, 0.4396,  ..., 0.0512, 0.3224, 0.1456],
        [0.4675, 0.4841, 0.4577,  ..., 0.3895, 0.2722, 0.9531],
        [0.1666, 0.5874, 0.0897,  ..., 0.7657, 0.7469, 0.4735]],
       device='cuda:0')
tensor([[0.4275, 0.6830, 0.0993,  ..., 0.9489, 0.9780, 0.6562],
        [0.5240, 0.4844, 0.1695,  ..., 0.0167, 0.7248, 0.1540],
        [0.7723, 0.5445, 0.6640,  ..., 0.7222, 0.7901, 0.3146],
        ...,
        [0.1375, 0.0850, 0.2293,  ..., 0.2454, 0.8424, 0.8072],
        [0.5370, 0.5438, 0.7350,  ..., 0.7349, 0.2352, 0.0141],
        [0.3022, 0.9106, 0.8692,  ..., 0.4461, 0.3143, 0.5680]],
       device='cuda:0')
```

0번 GPU가 할당 되어 있음을 확인 할 수 있다.

PyTorch | 텐서 다루기

```
tensor = torch.FloatTensor([[1., 2., 3.],
                            [4, 5, 6],
                            [7, 8, 9],
                            [10, 11, 12]
                            ])
tensor
```

```
tensor([[ 1.,  2.,  3.],
        [ 4.,  5.,  6.],
        [ 7.,  8.,  9.],
        [10., 11., 12.]])
```

(1) 차원, Shape 확인하기

```
tensor.dim() # tensor의 차원 확인
```

```
2
```

```
tensor.size() # tensor의 shape 확인
```

```
torch.Size([4, 3])
```

(2) 슬라이싱

```
tensor[:, 1] # 2번째 열만 취함
```

```
tensor([ 2.,  5.,  8., 11.])
```

```
tensor[:, 1].size() # shape 확인
```

```
torch.Size([4])
```

PyTorch | 텐서 다루기

(3) 브로드캐스팅 : 서로 다른 크기를 갖는 텐서끼리 연산 가능

```
t1 = torch.Tensor([[3, 3]])
t2 = torch.Tensor([[2, 1]])

t1 + t2
```

```
tensor([[5., 4.]])
```

→ 문제 없음

```
# Vector + scalar
t1 = torch.Tensor([[1, 2]])
t2 = torch.Tensor([3]) # [3] -> [3, 3]

t1 + t2
```

```
tensor([[4., 5.]])
```

→ 스칼라를
벡터의 크기만큼 늘림

```
# 1 x 2 Vector + 2 x 1 Vector --> 수학적으로는 연산 불가능
t1 = torch.Tensor([[1, 2]]) # (1 x 2) -> (2 x 2)
t2 = torch.Tensor([[3], [4]]) # (2 x 1) -> (2 x 2)

t1 + t2
```

```
tensor([[4., 5.],
        [5., 6.]])
```

→ PyTorch에서는
연산이 가능하도록 변형함

PyTorch | 텐서 다루기

(4) 행렬 곱셈과 원소 별 곱셈

```
t1 = torch.FloatTensor([[1, 2], [3, 4]])
t2 = torch.FloatTensor([[1], [2]])

print(t1)
print(t2)
```

```
tensor([[1., 2.],
        [3., 4.]])
tensor([[1.],
        [2.]])
```

```
t1.matmul(t2) # 행렬 곱셈
```

```
tensor([[ 5.],
        [11.]])
```

```
t1 * t2 # 행렬의 원소별 곱셈
```

```
tensor([[1., 2.],
        [6., 8.]])
```

```
t1.mul(t2) # 행렬의 원소별 곱셈
```

```
tensor([[1., 2.],
        [6., 8.]])
```

브로드캐스팅 발생

PyTorch | 텐서 다루기

(5) 합, 평균 구하기

```
t = torch.FloatTensor([[1, 2], [3, 4]])
t
```

```
tensor([[1., 2.],
        [3., 4.]])
```

```
print(t.sum()) # 단순히 원소 전체의 덧셈을 수행
print(t.sum(dim=0)) # 행을 제거하면서 뎃셈 수행
print(t.sum(dim=1)) # 열을 제거하면서 덧셈 수행
print(t.sum(dim=-1)) # 마지막 차원을 제거하면서 덧셈 수행 --> 열 제거
```

```
tensor(10.)
tensor([4., 6.])
tensor([3., 7.])
tensor([3., 7.])
```

```
print(t.mean()) # 단순히 원소 전체의 평균을 계산
print(t.mean(dim=0)) # 행을 제거하면서 평균 계산
print(t.mean(dim=1)) # 열을 제거하면서평균 계산
print(t.mean(dim=-1)) # 마지막 차원을 제거하면서 평균 계산 --> 열 제거
```

```
tensor(2.5000)
tensor([2., 3.])
tensor([1.5000, 3.5000])
tensor([1.5000, 3.5000])
```

PyTorch | 텐서 다루기

(6) 최대/최소 구하기

```
t = torch.FloatTensor([[1, 2], [3, 4]])
t
```

```
tensor([[1., 2.],
        [3., 4.]])
```

```
print(t.max()) # 단순히 원소 전체의 최대값 반환
print(t.max(dim=0)) # 행을 제거하면서 최대값 반환
print(t.max(dim=1)) # 열을 제거하면서 최대값 반환
print(t.max(dim=-1)) # 마지막 차원을 제거하면서 최대값 반환 --> 열 제거
```

```
tensor(4.)
torch.return_types.max(
values=tensor([3., 4.]),
indices=tensor([1, 1]))
torch.return_types.max(
values=tensor([2., 4.]),
indices=tensor([1, 1]))
torch.return_types.max(
values=tensor([2., 4.]),
indices=tensor([1, 1]))
```

```
print(t.argmax()) # 단순히 원소 전체의 최대값의 인덱스 반환
print(t.argmax(dim=0)) # 행을 제거하면서 최대값의 인덱스 반환
print(t.argmax(dim=1)) # 열을 제거하면서 최대값의 인덱스 반환
print(t.argmax(dim=-1)) # 마지막 차원을 제거하면서 최대값의 인덱스 반환 --> 열 제거
```

```
tensor(3)
tensor([1, 1])
tensor([1, 1])
tensor([1, 1])
```

실습 Jupyter Notebook 파일 링크

https://drive.google.com/drive/folders/1CGX4mzfa3XwmnTOl3HxtE6bKWjU1fs1f?usp=sharing

.ipynb 파일을 다운로드 받아서 Jupyter Notebook으로 열어주시면 됩니다.