

스마트 야드 구축을 위한 빅데이터 분석 교육

- 머신러닝 -

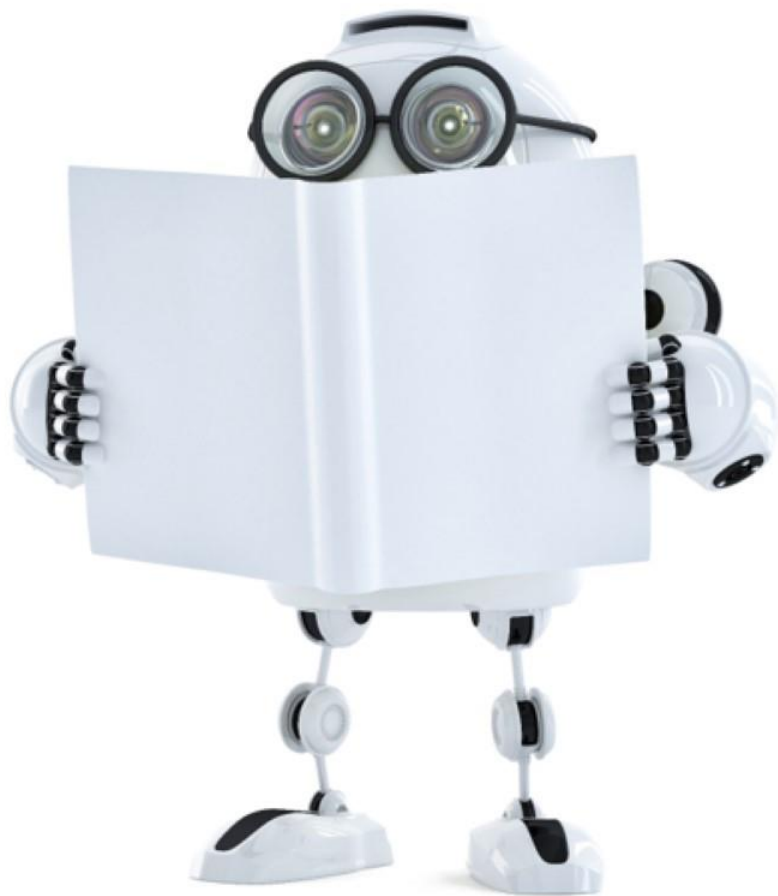
Sungchul Choi



What ML is?

Machine Learning Overview

**Director of TEAMLAB
Sungchul Choi**



머신 러닝

Machine Learning

**컴퓨터에게
뭔가를 가르쳐서
잘하게 한다.**



Daum 루빅스

2,710만 명

2016년 11월

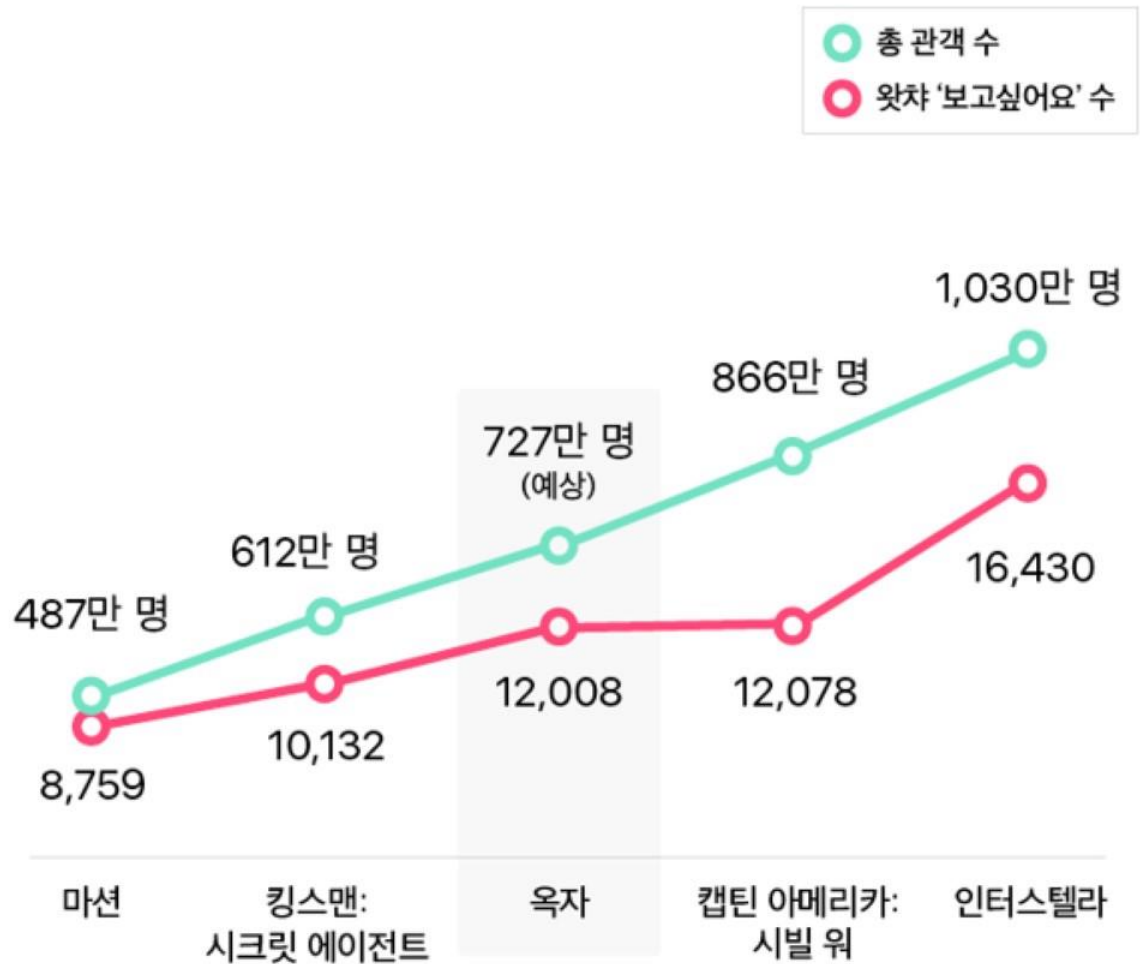
1,910만 명

2015년 11월

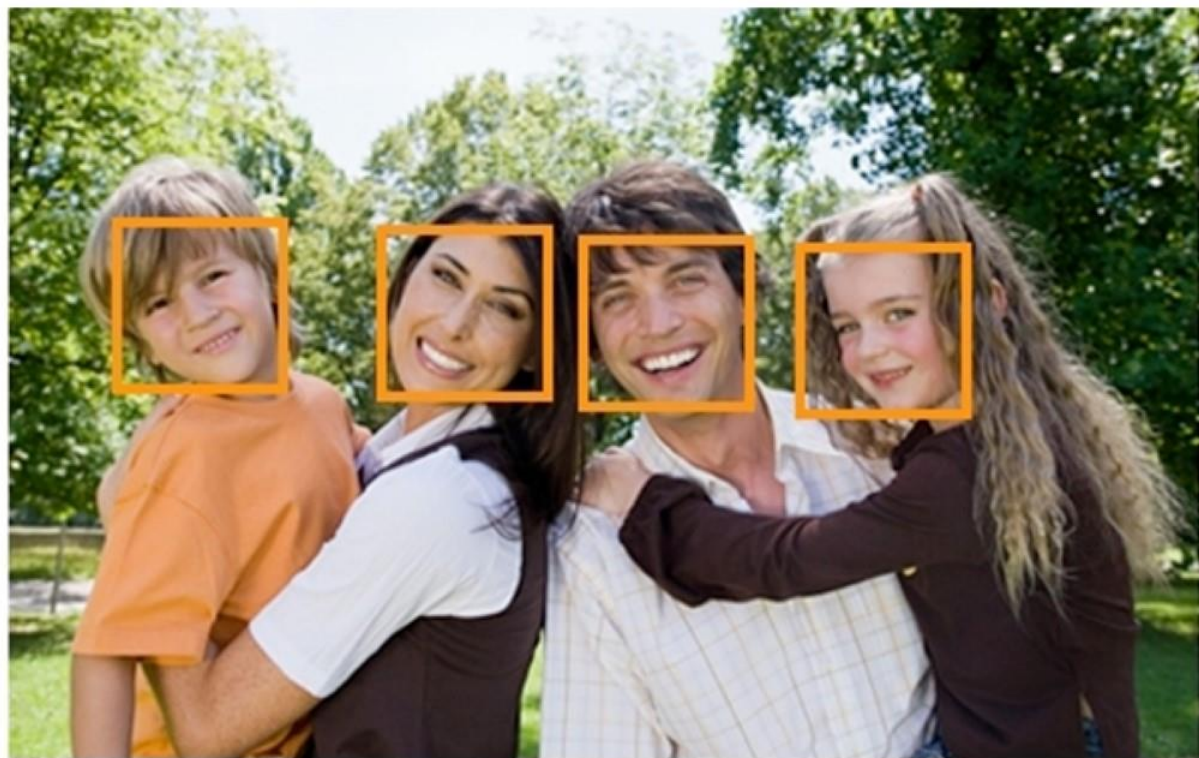




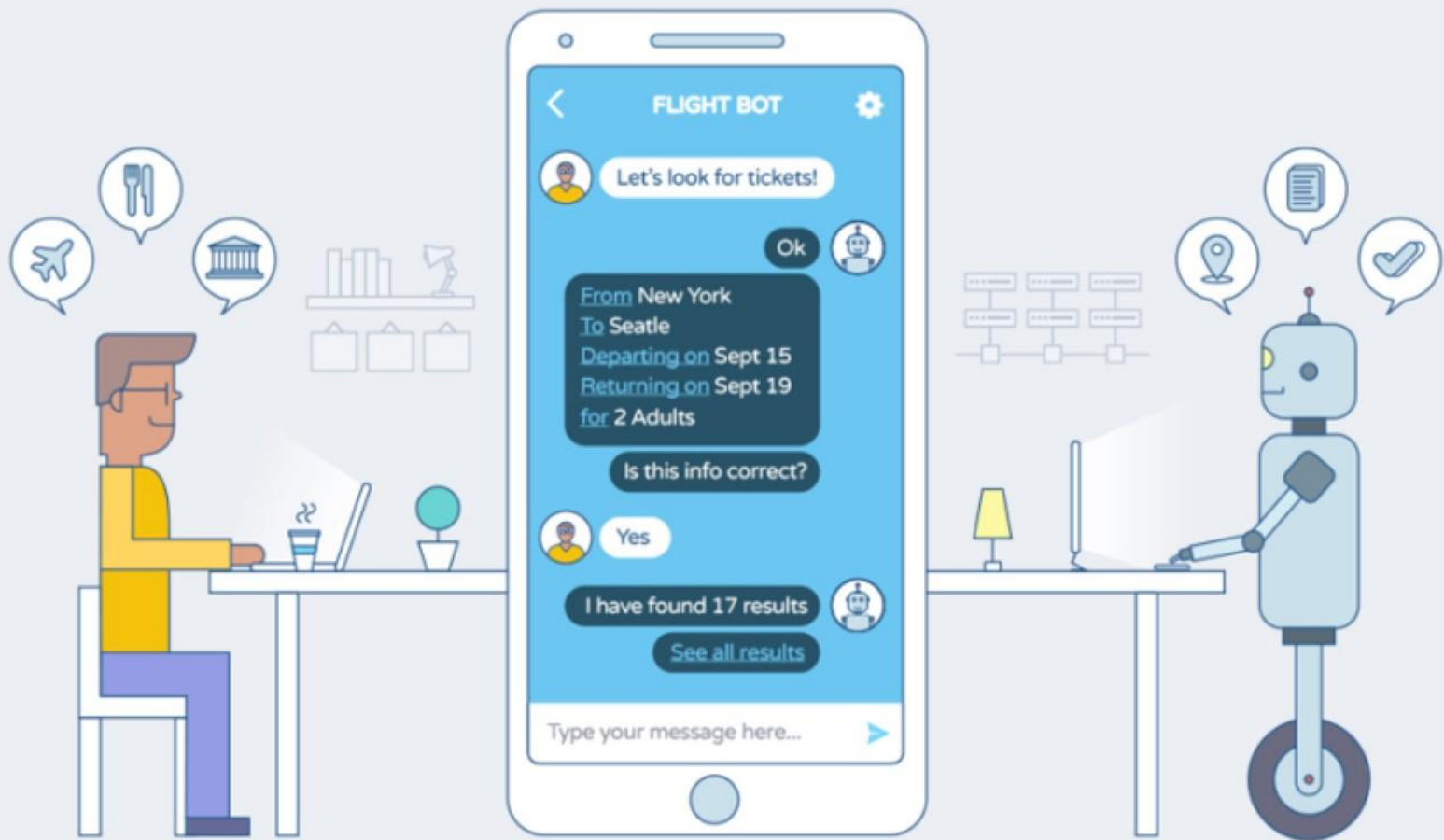
학기 평량평균이 2.0 미만이거나,
그 학기에 수강신청을 전혀 하지
않으면, 성적경고를 받는다.



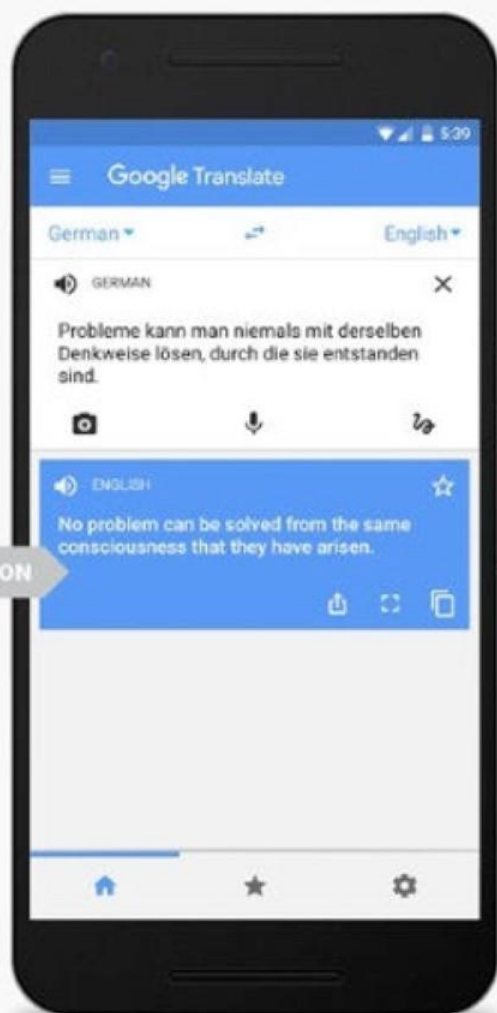




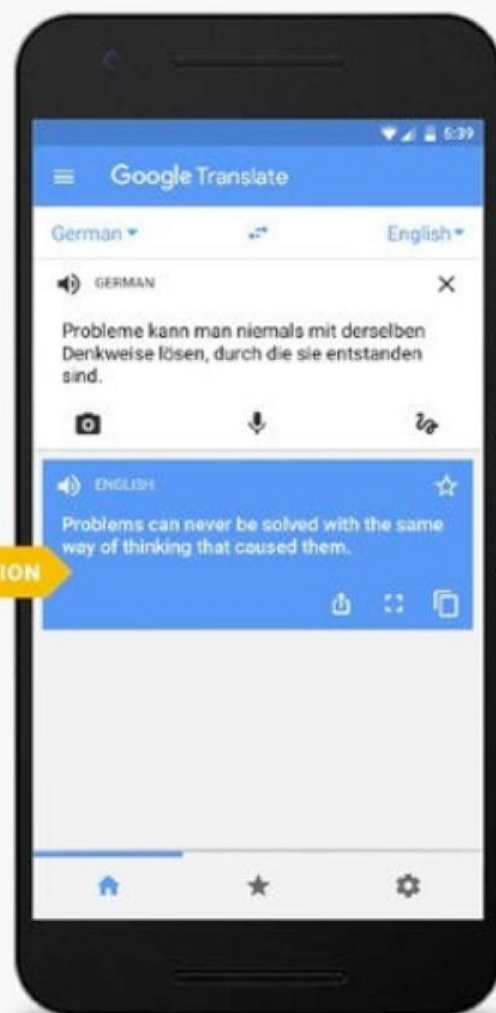




OLD TRANSLATION



NEW TRANSLATION



**이런일을 기존엔
어떻게 했을까?**

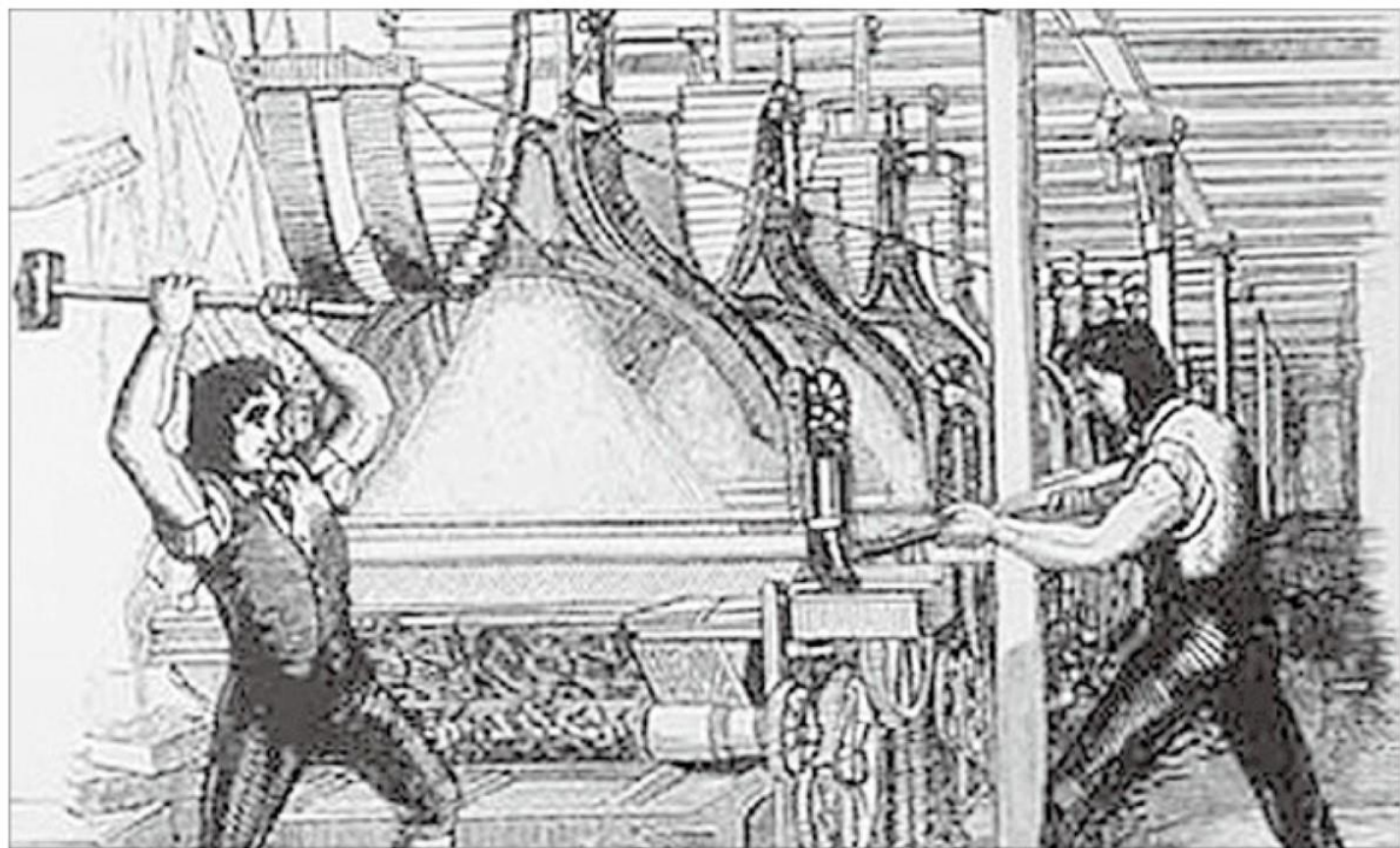
사람 □□ 기계

4차 산업혁명

머신러닝

=

사람의 지적업무를 대신함





산업혁명

인간의노동 Ⅱ 증기기관

머신러닝

인간의 지식 ⇔ 알고리즘



Machine Learning

산업혁명 - **물건** 생산을 위해 **육체노동**을 기계로 자동화하자

머신러닝 - **정보** 추출을 위해 **정신노동**을 기계로 자동화하자

(특히 deep learning)

모든 분야의 데이터 분석

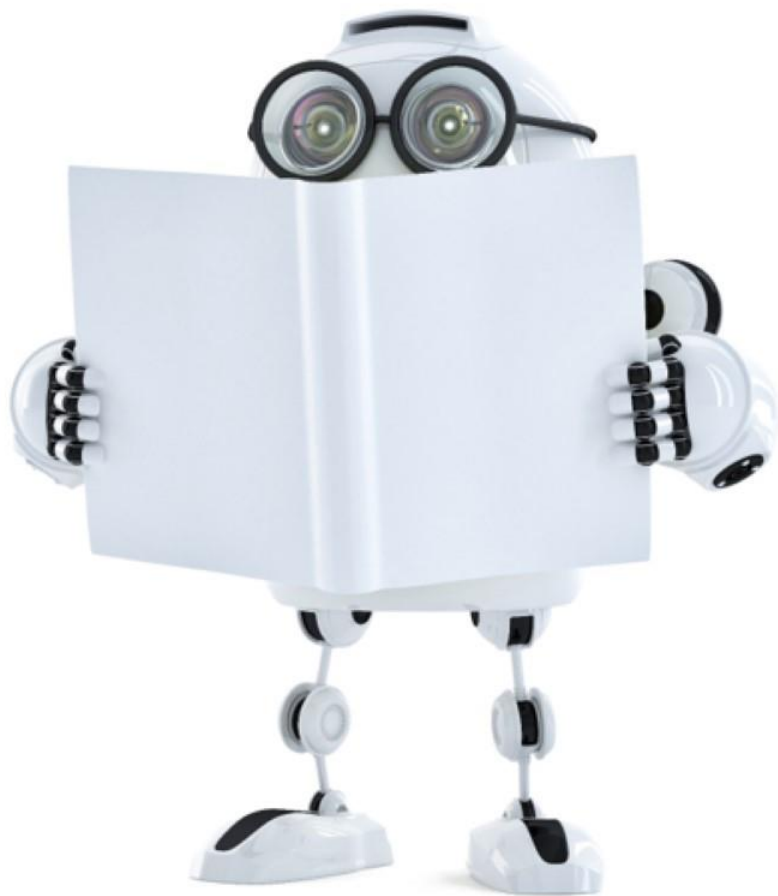
TEAMLAB

Human knowledge belongs to the world.

An Understanding of the Data Keywords

Machine Learning Overview

**Director of TEAMLAB
Sungchul Choi**



머신 러닝

Machine Learning

빅 데이터

Big Data

데이터 과학

Data Science

데이터 마이닝

Data Mining

Key concepts

- Big Data
- 정해진 시간내에 **한 대의 머신**으로 처리가 어려운 크기의 데이터
 - 고성능 머신 또는 **다수의 머신**을 사용해 **목표**대로 데이터를 처리할 수 있음
 - ex) 실시간 대용량 SQL 처리

Key concepts

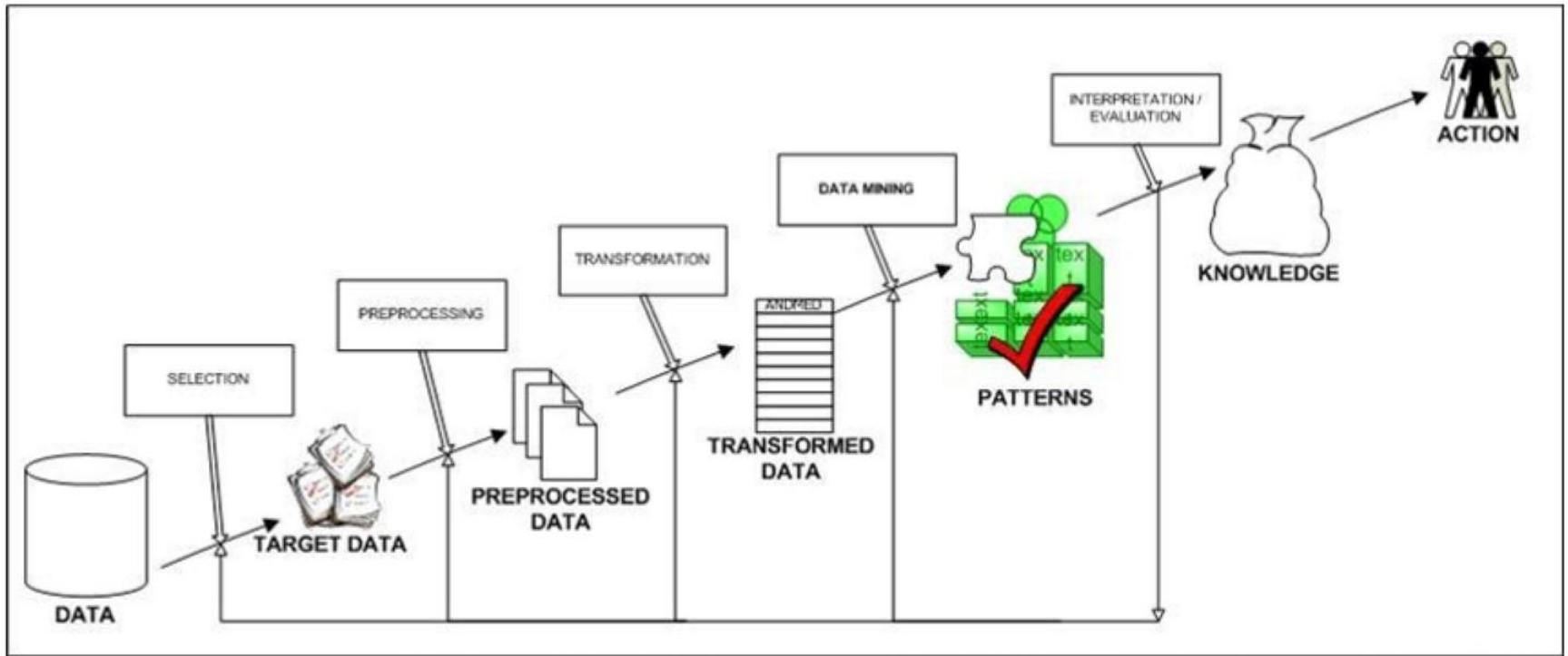
- Machine Learning
- 데이터로 부터 패턴을 학습하는 기법
 - 알고리즘, 모델의 묶음? 상위개념
 - 데이터의 크기는 중요,
그러나 작다고 불가능한 것은 아님
 - ex) 컴퓨터에서 손 글씨 이해시키기

Key concepts

Data Mining

- 사실상 머신러닝과 같은 의미
 - 데이터에서 의미있는 규칙 찾기
- 주요적용 사례가 비즈니스 분야
- 머신러닝 - 알고리즘
데이터마이닝 - 어플리케이션
- ex) 스팸 필터, 상품 추천

Key concepts



Source - <http://dataminingwarehousing.blogspot.kr/2008/10/data-mining-steps-of-data-mining.html>

Key concepts

데이터 처리

Big Data



Data
Engineering

데이터 분석

Machine
Learning



Data
Analysis

Key concepts

Big Data

활용

Data Mining

분산처리
데이터베이스

기술개념

Machine
Learning

Hadoop
NoSQL Spark
BigTable

알고리즘
구현체

Navie Bayes
SVM Logistics
Regression

딥 러닝

Deep Learning

Key concepts

Deep Learning

- Neural Net 기반 머신러닝 알고리즘
 - 최근 높은 성능에 대세로 떠오름
- 사진내 객체 인식, 번역 등 탁월한 성능
- 머신러닝과 구분하여, 새로운 분야로 주장하는 사람도 있음

ImageNet Challenge

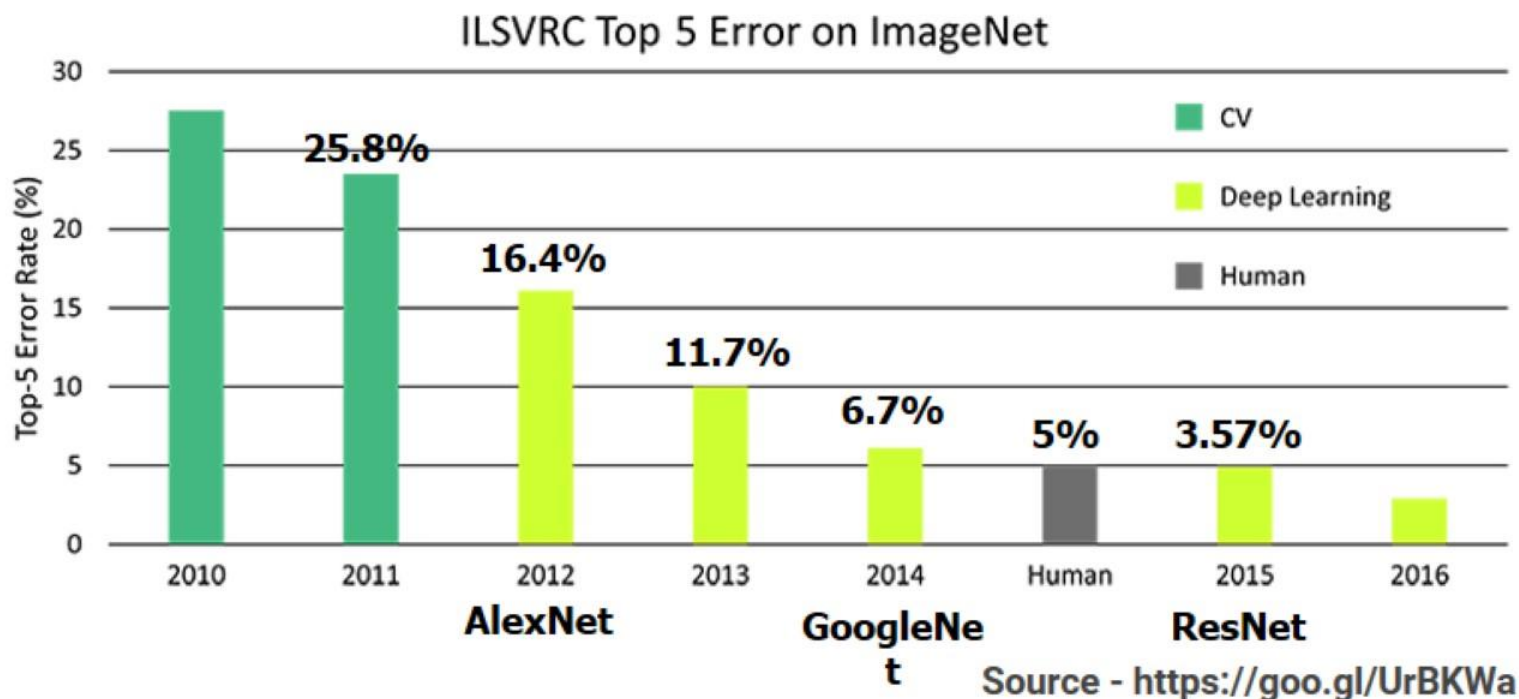
IMAGENET

- 1,000 object classes (categories).
- Images:
 - 1.2 M train
 - 100k test.



Source - <https://goo.gl/pCX3cD>

인간보다 낮아진 이미지 오인식률



ARTIFICIAL INTELLIGENCE

Early artificial intelligence stirs excitement.



MACHINE LEARNING

Machine learning begins to flourish.



DEEP LEARNING

Deep learning breakthroughs drive AI boom.



1950's

1960's

1970's

1980's

1990's

2000's

2010's

Source: <https://goo.gl/gz7EZT>

● Big Data
검색어

● Machine Learning
검색어

● Deep Learning
검색어

● Hadoop
검색어

+

전 세계 ▼

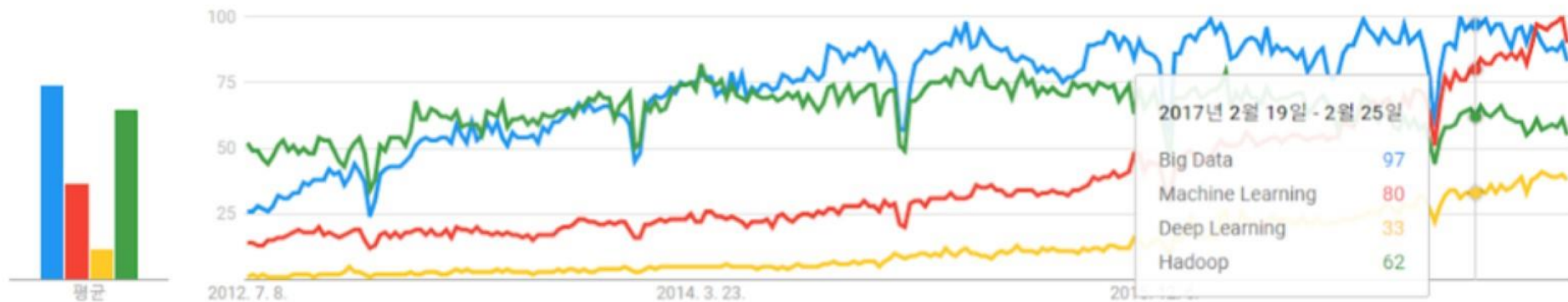
지난 5년 ▼

모든 카테고리 ▼

웹 검색 ▼

Google Trends

시간 흐름에 따른 관심도 변화 ?



빅데이터 ➡ 머신러닝 ➡ 딥러닝

**딥 러닝은
마법의 탄환이 아니다**

머신러닝 부터
차근 차근

Data Science

빅 데이터 처리

분산 처리
(병렬처리)

Database

Hadoop / Spark / NoSQL

MapReduce

HDFS

분산 파일저장

대용량 데이터 처리
Data Engineering

대형 데이터 처리 지원

인공지능

Data Mining

Computer
Vision

NLP

가능하게 함

Machine
Learning

=

Pattern
Recognition

Neural Net

SVM

Decision
Tree

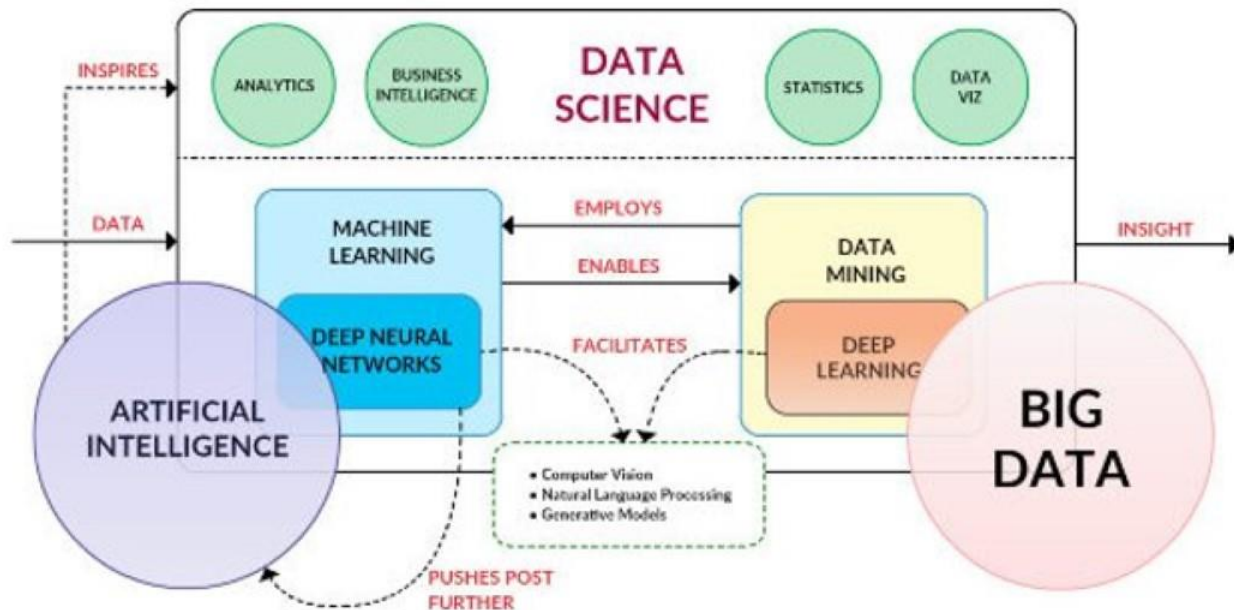
Statics
Probability

Linear
Algebra

Information
Theory

데이터 분석
Data Analysis

정의는 조금씩 다릅니다



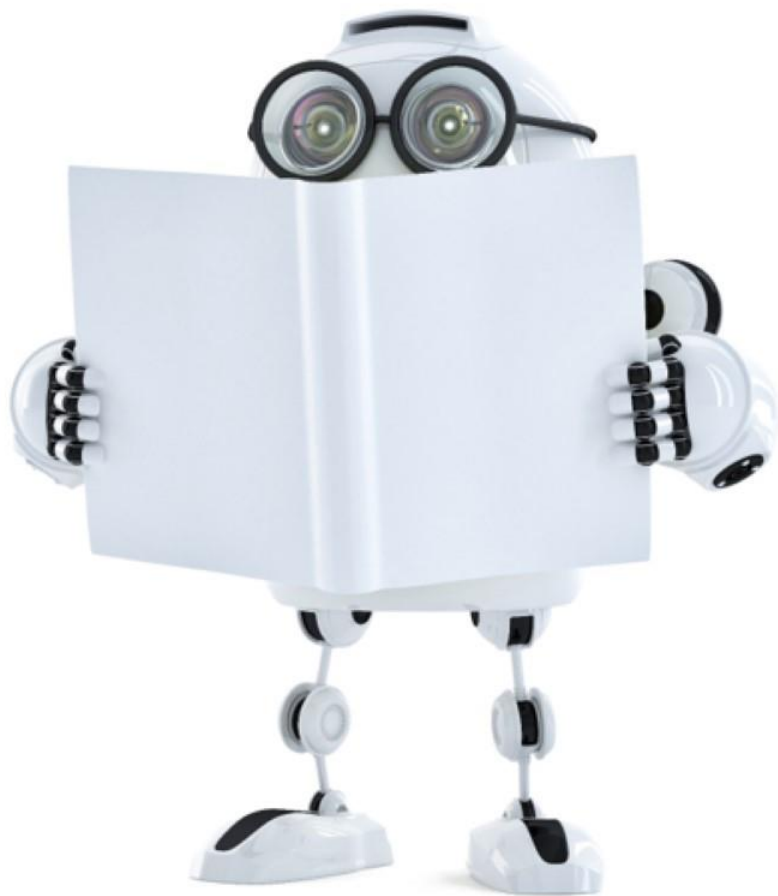
TEAMLAB

Human knowledge belongs to the world.

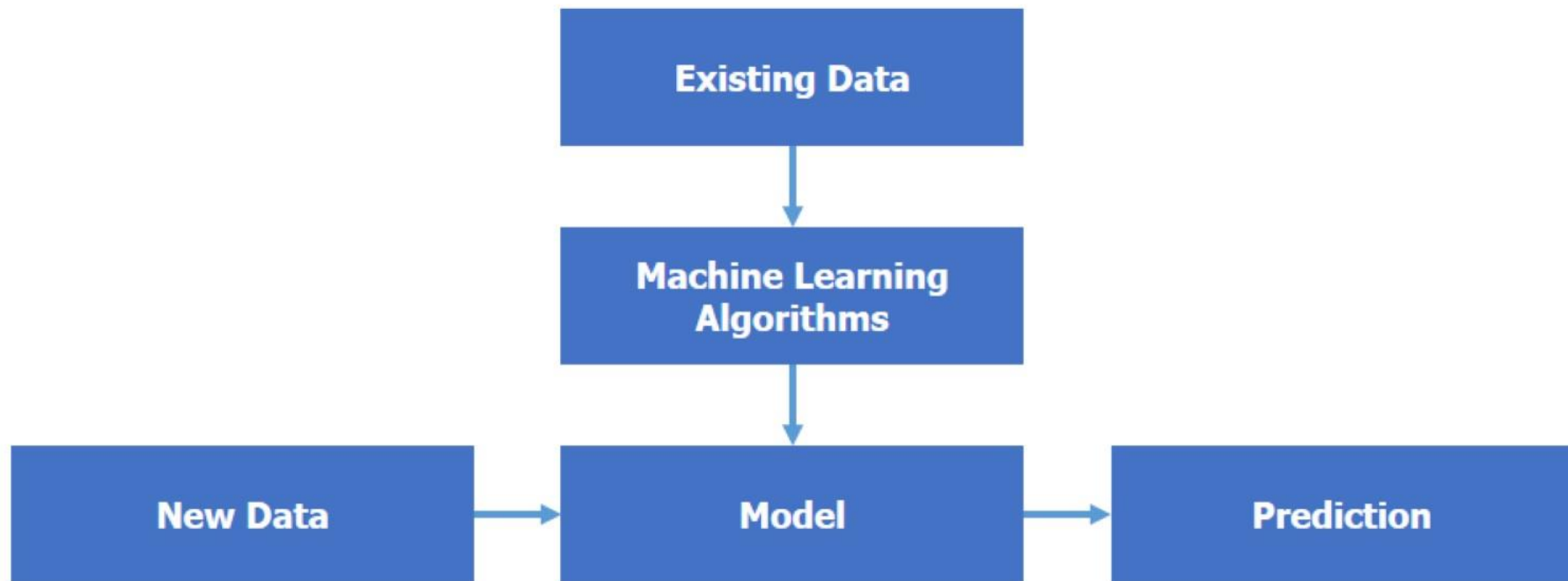
How to learn machine learning

Machine Learning Overview

**Director of TEAMLAB
Sungchul Choi**

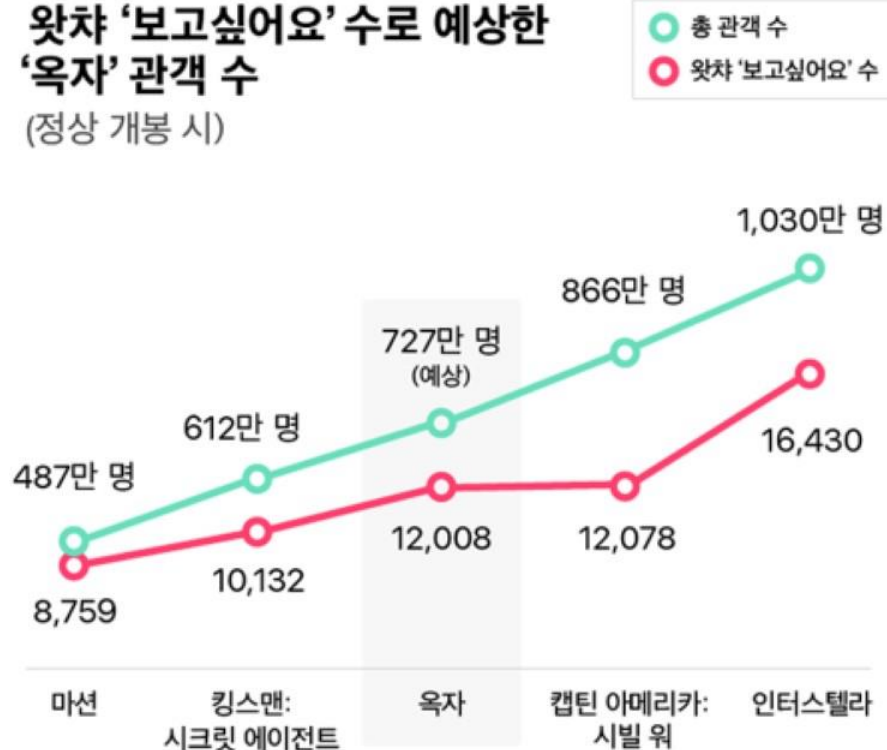


Machine Learning Process



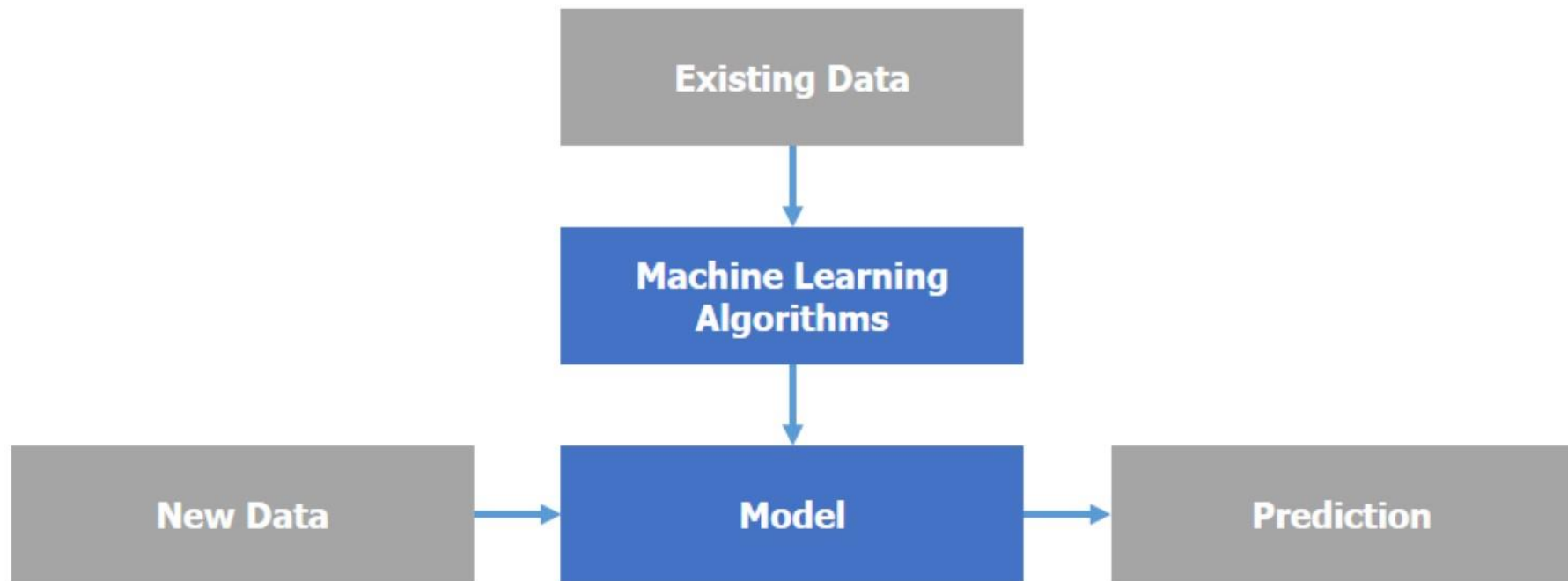
왓차 '보고싶어요' 수로 예상한 '옥자' 관객 수

(정상 개봉 시)



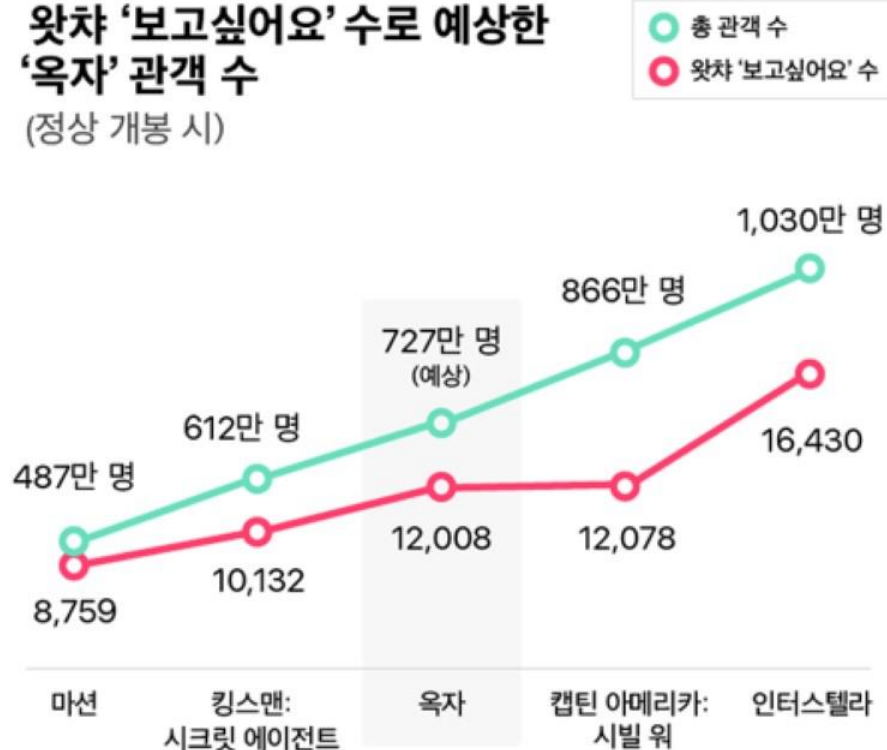
Source: <http://platum.kr/archives/83757>

Machine Learning Process



왓차 '보고싶어요' 수로 예상한 '옥자' 관객 수

(정상 개봉 시)



Source: <http://platum.kr/archives/83757>

Key concepts

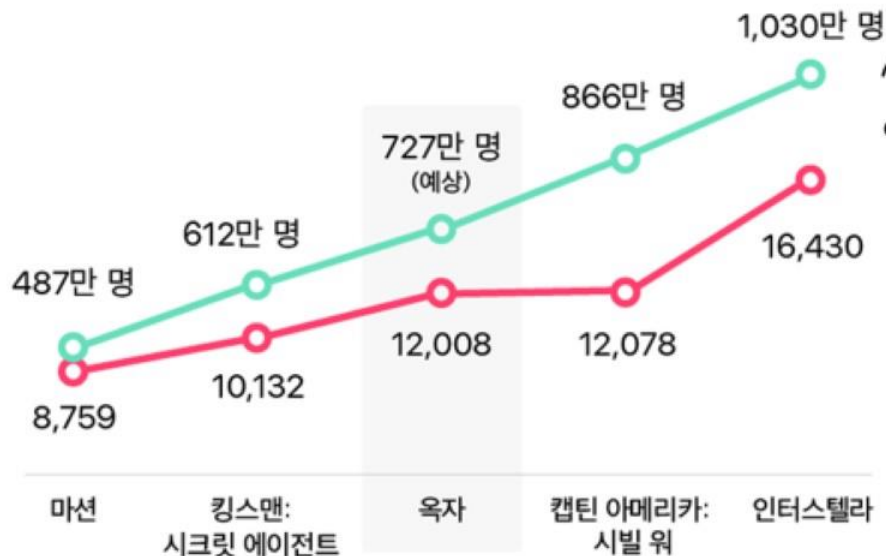
Model - 예측을 위한 수학 공식, 함수
1차 방정식, 확률분포, condition rule

Algorithms - 어떠한 문제를 풀기 위한 과정
Model을 생성하기 위한 (훈련) 과정

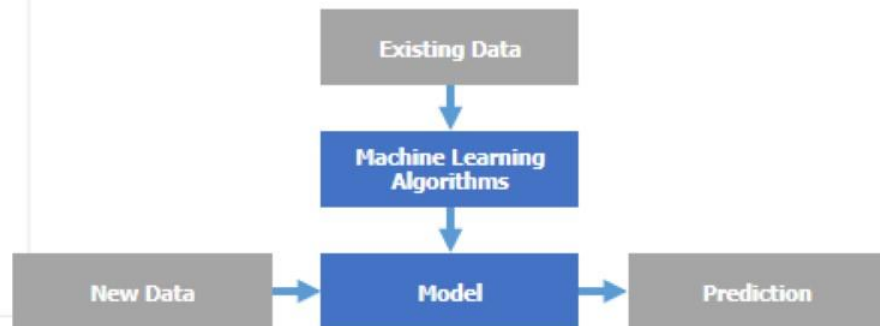
왓차 '보고싶어요' 수로 예상한 '옥자' 관객 수

(정상 개봉 시)

- 총 관객 수
- 왓차 '보고싶어요' 수



$$y = ax + b$$



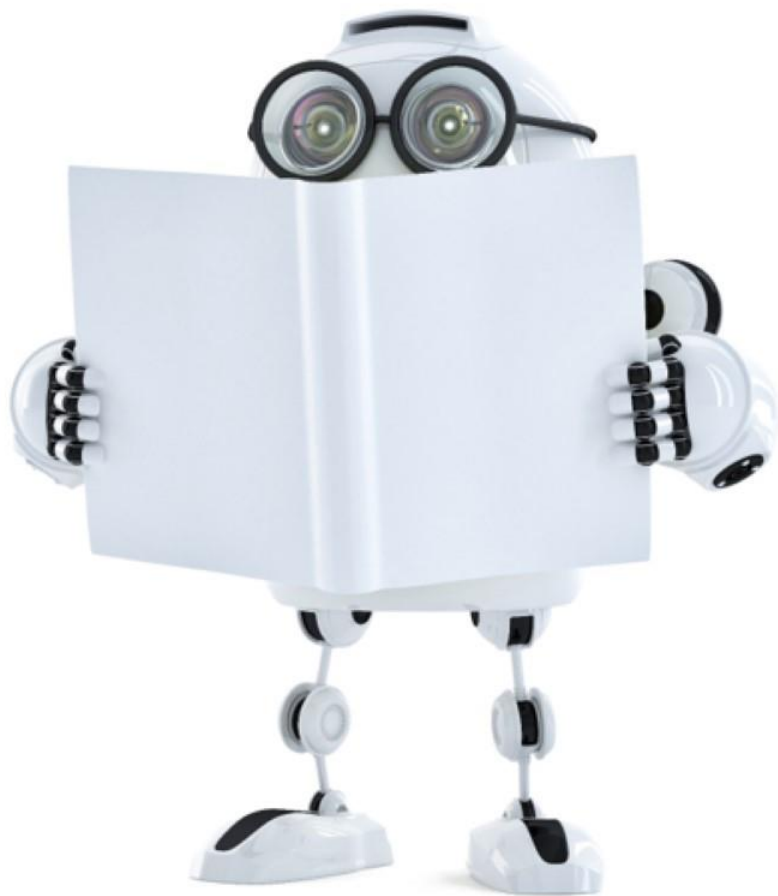
TEAMLAB

Human knowledge belongs to the world.

Types of Machine Learning

Machine Learning Overview

**Director of TEAMLAB
Sungchul Choi**



**내가 생각하는
머신러닝**

선을 찾는 것
선을 긋는 것

꼭 맞는 얘기는 아니에요...

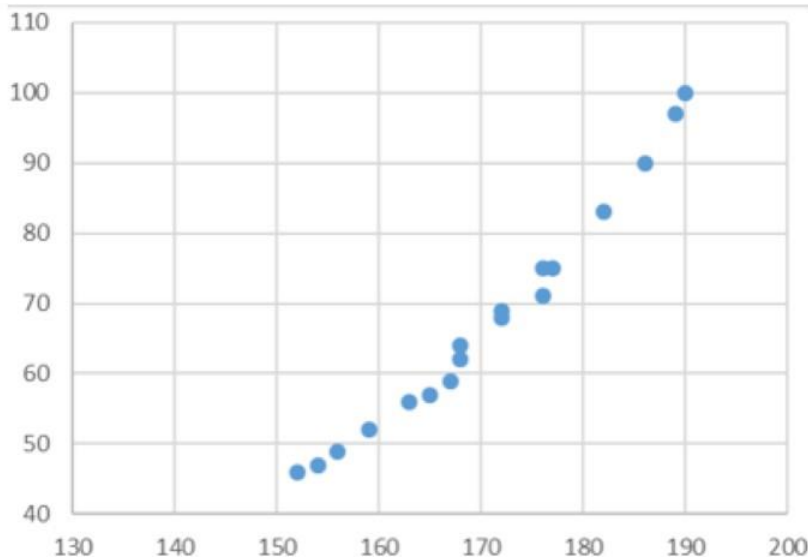
**어떤 종류의
선을 긋는가?**

Regression

키를 가지고 몸무게를 예측할 수 있을까?

Height	Weight	Height	Weight
152	46	172	69
154	47	172	68
156	49	176	71
159	52	176	75
163	56	177	75
165	57	182	83
167	59	186	90
168	64	189	97
168	62	190	100

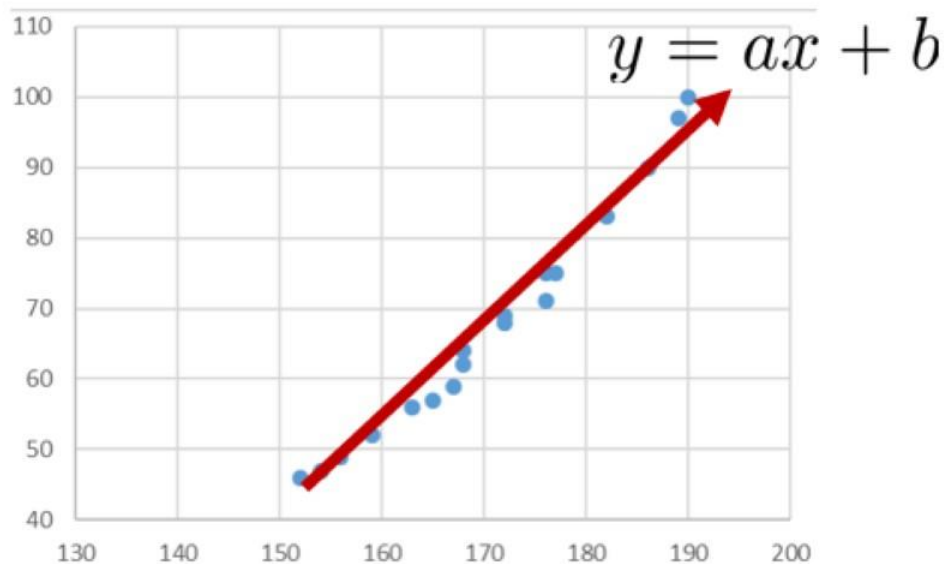
Data source: <http://goo.gl/gDscUQ>



키를 가지고 몸무게를 예측할 수 있을까?

Height	Weight	Height	Weight
152	46	172	69
154	47	172	68
156	49	176	71
159	52	176	75
163	56	177	75
165	57	182	83
167	59	186	90
168	64	189	97
168	62	190	100

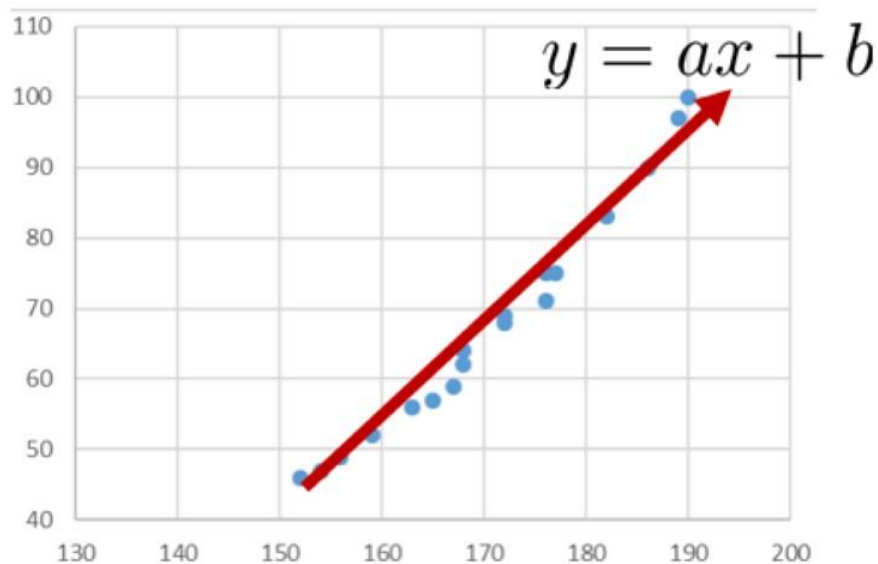
Data source: <http://goo.gl/gDscUQ>



Regression - 회귀, 추세선을 긋는 것

Height	Weight	Height	Weight
152	46	172	69
154	47	172	68
156	49	176	71
159	52	176	75
163	56	177	75
165	57	182	83
167	59	186	90
168	64	189	97
168	62	190	100

Data source: <http://goo.gl/gDscUQ>

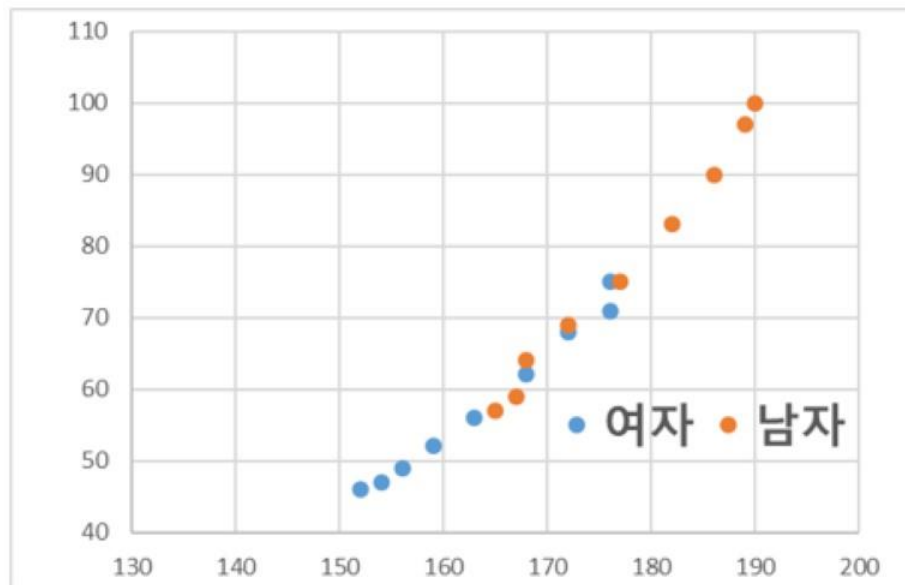


Classification

키와 몸무게를 가지고 남/녀를 분류할 수 있을까?

H 186, W 82 남자? 여자?

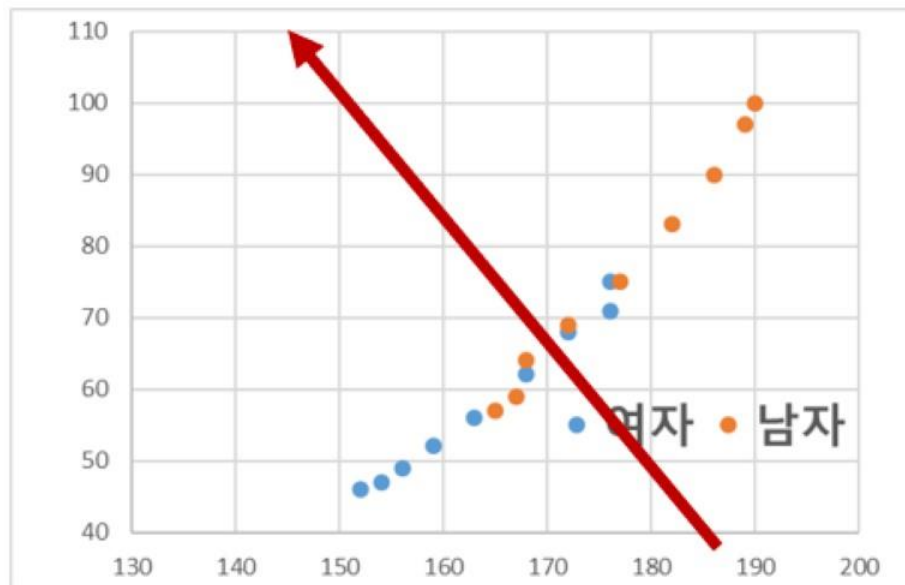
Sex	Height	Weight	Sex	Height	Weight
여자	152	46	남자	172	69
여자	154	47	여자	172	68
여자	156	49	여자	176	71
여자	159	52	여자	176	75
여자	163	56	남자	177	75
남자	165	57	남자	182	83
남자	167	59	남자	186	90
남자	168	64	남자	189	97
여자	168	62	남자	190	100

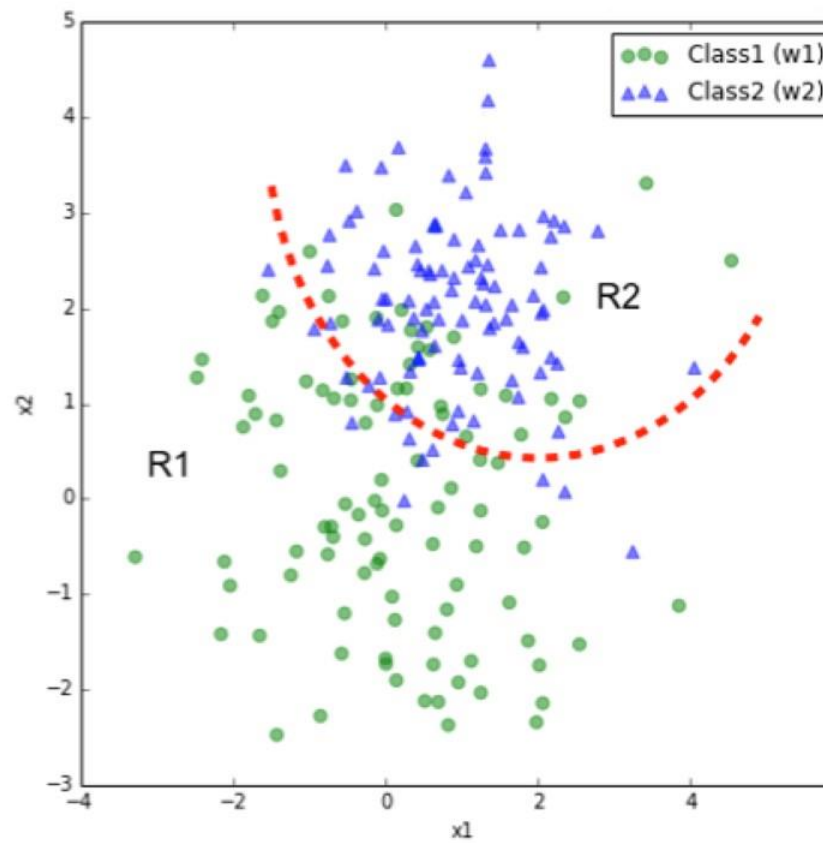
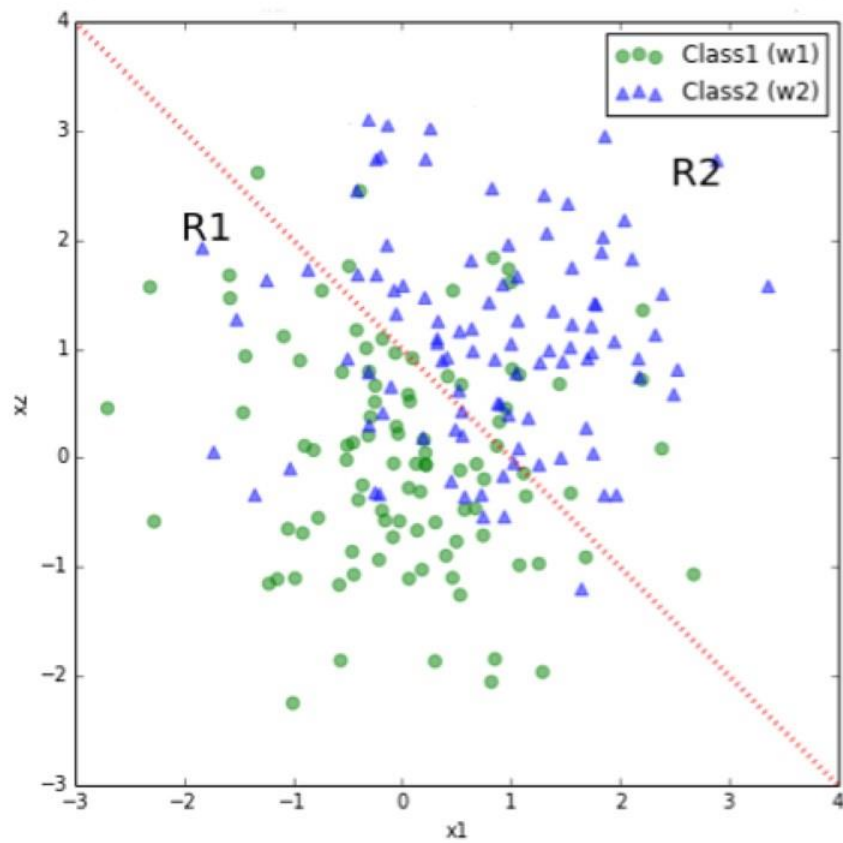


Classification - 분류, 데이터의 유형을 나누는 것

기존 데이터를 바탕으로 데이터 유형을 나눠 보기

Sex	Height	Weight	Sex	Height	Weight
여자	152	46	남자	172	69
여자	154	47	여자	172	68
여자	156	49	여자	176	71
여자	159	52	여자	176	75
여자	163	56	남자	177	75
남자	165	57	남자	182	83
남자	167	59	남자	186	90
남자	168	64	남자	189	97
여자	168	62	남자	190	100





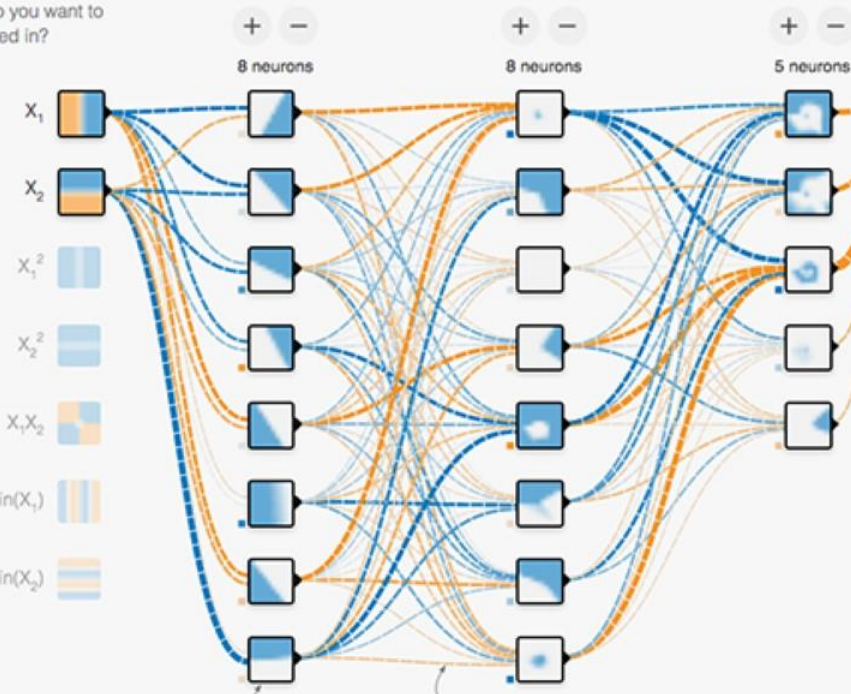
<https://goo.gl/z5rEu6>

FEATURES

Which properties
do you want to
feed in?

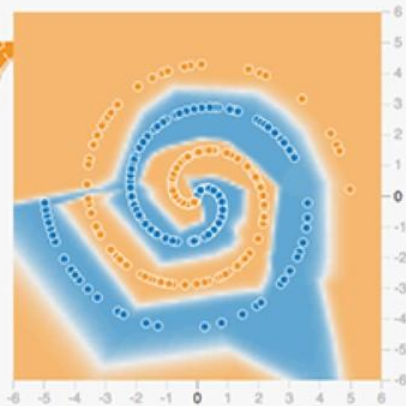


3 HIDDEN LAYERS



OUTPUT

Training loss 0.001



Colors shows
data, neuron and
weight values.

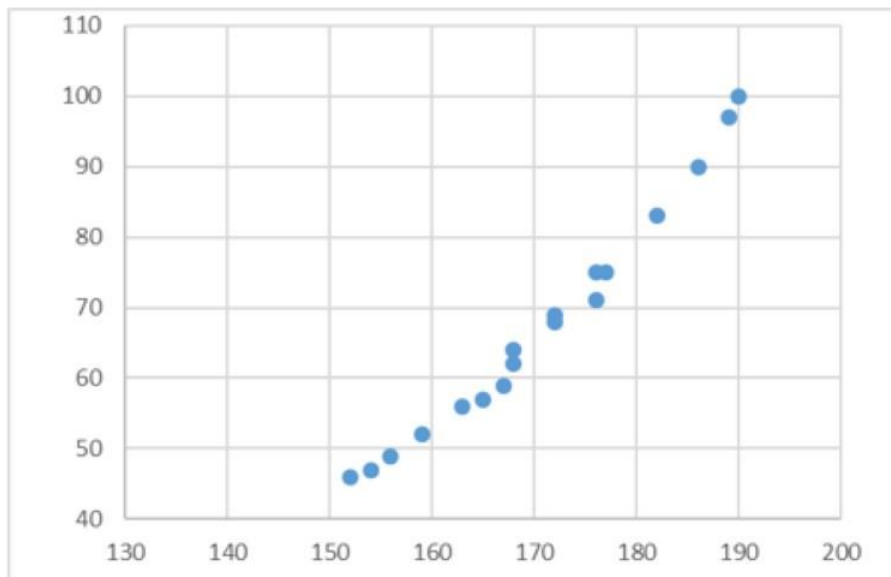


Clustering

농구팀을 만든다면 몇 팀으로 만들까?

어떻게 사람을 묶으면 될까?

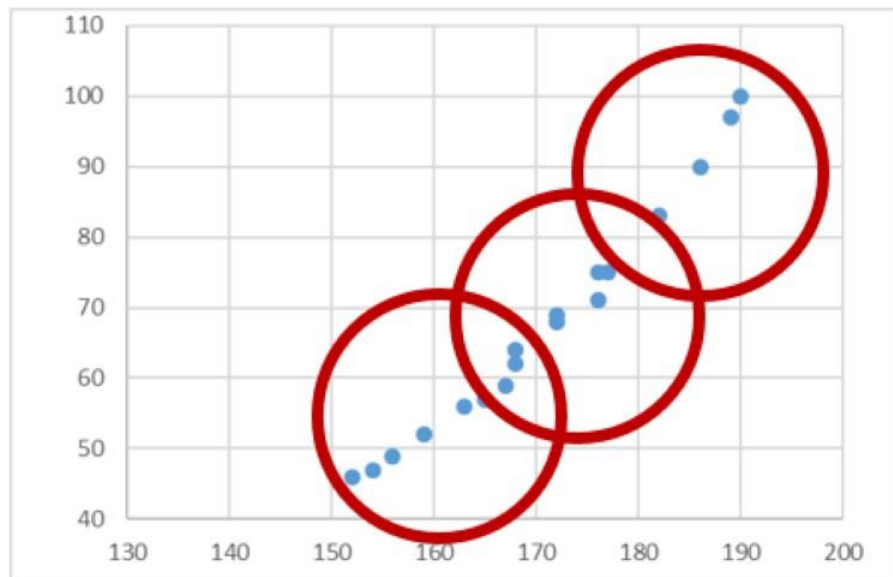
Height	Weight	Height	Weight
152	46	172	69
154	47	172	68
156	49	176	71
159	52	176	75
163	56	177	75
165	57	182	83
167	59	186	90
168	64	189	97
168	62	190	100



Clustering - 군집, 데이터를 모으는 것

아무런 사전정보 없이 데이터 유형을 나눠보기

Sex	Height	Weight	Sex	Height	Weight
여자	152	46	남자	172	69
여자	154	47	여자	172	68
여자	156	49	여자	176	71
여자	159	52	여자	176	75
여자	163	56	남자	177	75
남자	165	57	남자	182	83
남자	167	59	남자	186	90
남자	168	64	남자	189	97
여자	168	62	남자	190	100



정리

Types of Machine Learning

회귀 - 데이터(X, Y)를 줄께, X' 의 숫자 Y' 를 예측해봐(Continuous Y)
ex) 주가 예측, 경제 성장률 예측, 영화 관람객 예측

분류 - 데이터(X, Y)를 줄께, X' 의 Y' 를 분류해봐(Discrete Y)
ex) 스팸메일 여부, 게임 어뷰저 여부, 신문기사 분류

군집 - 데이터(X)를 줄께, X 를 n 개로 나눠봐
ex) 고객 집단 나누기

Types of Machine Learning

Supervised

Y 데이터가 있는 것

Regression

Classification

Unsupervised

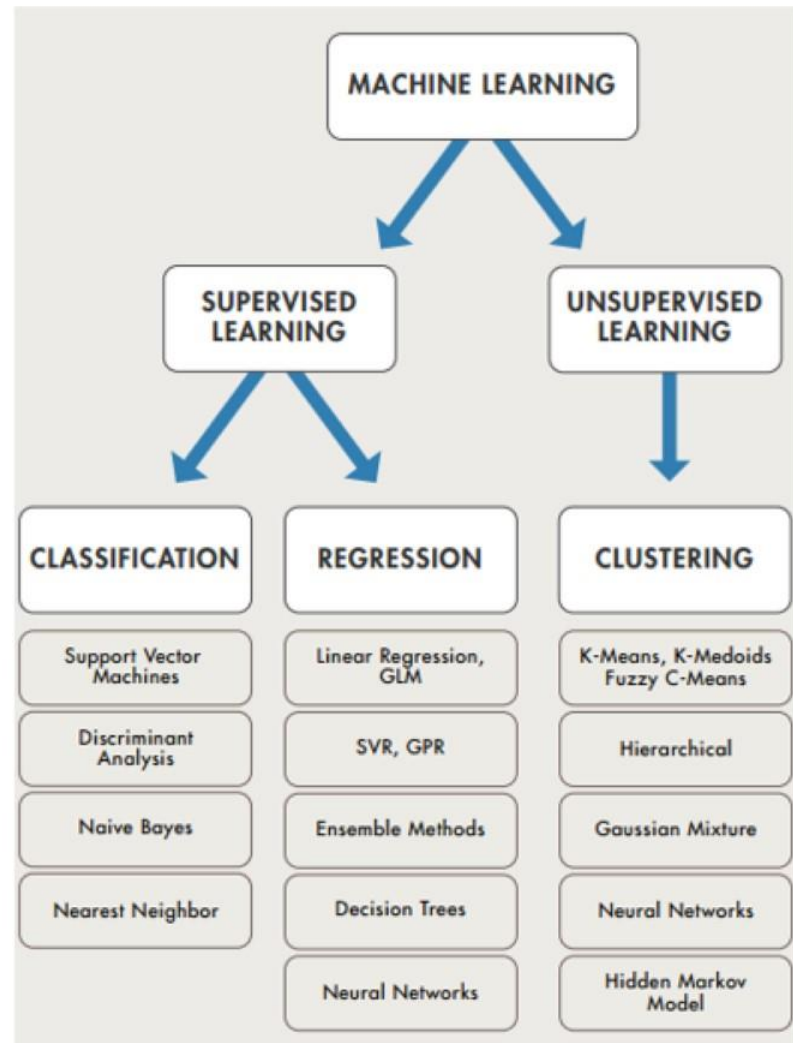
Y 데이터가 없는 것

Clustering

Reinforcement Learning, Recommendation을 추가하기 포함

Types of Machine Learning

<https://goo.gl/UaGGrk>



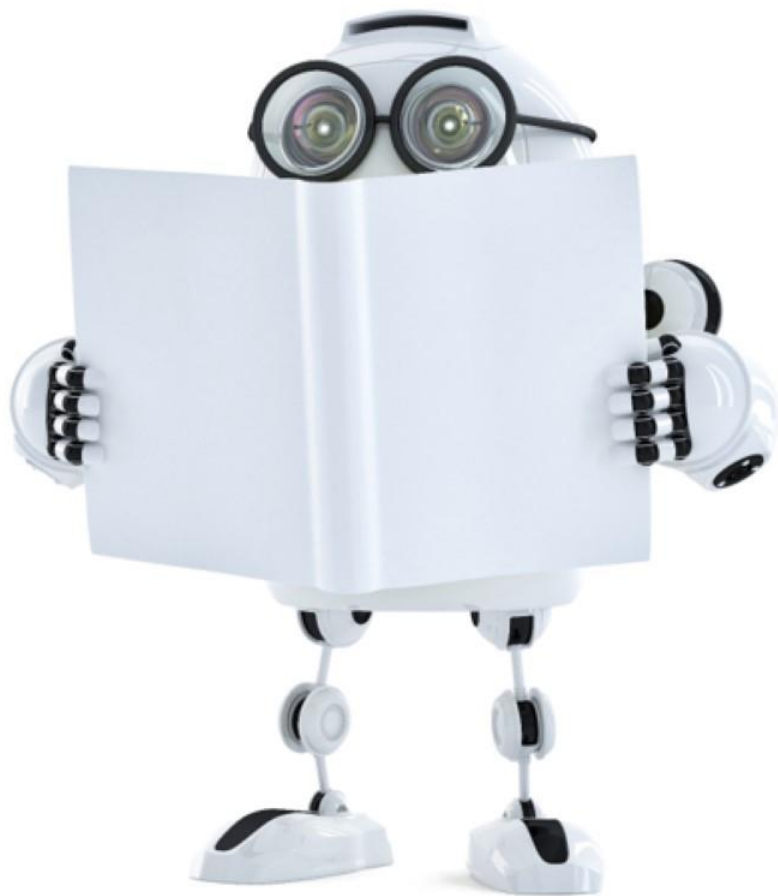
TEAMLAB

Human knowledge belongs to the world.

Python ecosystem for ML

Machine Learning Overview

**Director of TEAMLAB
Sungchul Choi**





과학 계산용 Python 통합 배포판



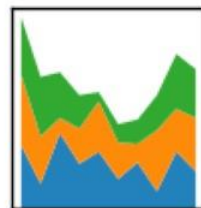
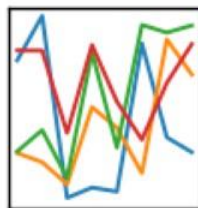
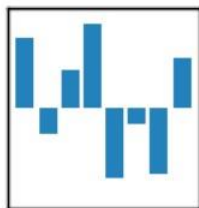
파이썬 가상 환경 관리 패키지



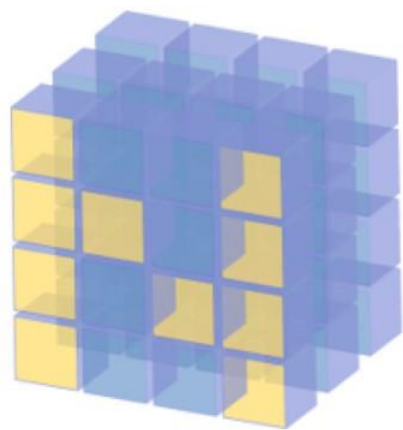
데이터 분석을 위한 파이썬 IDE

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Python 데이터 분석 라이브러리

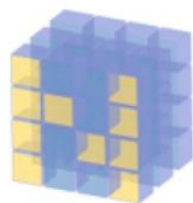


NumPy

고성능 Array 처리 라이브러리



데이터 시각화를 위한 파이썬 패키지

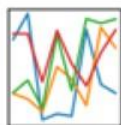


NumPy

Vector, matrix 처리

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



데이터 전처리 EDA

matplotlib

EDA

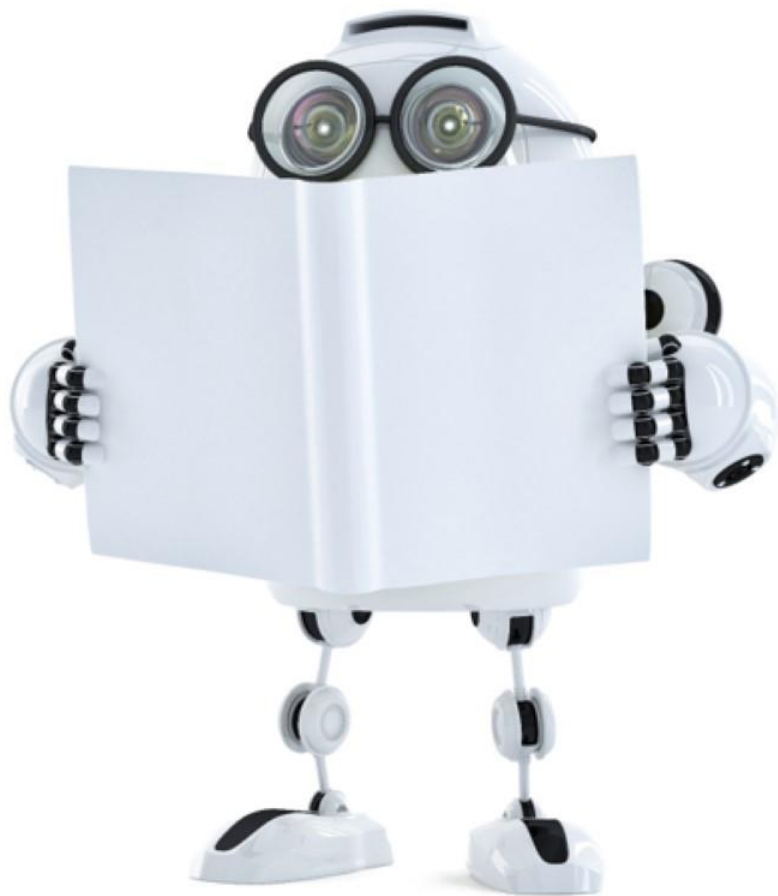
TEAMLAB

Human knowledge belongs to the world.

Python ecosystem for ML

Machine Learning Overview

**Director of TEAMLAB
Sungchul Choi**





데이터 분석을 위한 파이썬 IDE

Jupyter

- IPython(<http://ipython.org>) 커널을 기반으로 한 대화형 파이썬 셸
- 일반적인 터미널 셸 + 웹 기반 데이터 분석 Notebook 제공
- 미디어, 텍스트, 코드, 수식 등을 하나의 문서로 표현 가능
- 사실상의 데이터 분석 Interactive Shell의 표준
- Julia + Python + R

Jupyter

- 실행방법 실행시키고자 하는 디렉토리로 이동함

jupyter notebook

- <http://localhost:8888/tree> 주소로 jupyter 가 실행됨

Jupyter

- 터미널 창의 token 정보를 사용해야할 때가 있음

```
[I 21:35:00.179 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

```
[C 21:35:00.180 NotebookApp]
```

Copy/paste this URL into your browser when you connect for the first time, to login with a token:

<http://localhost:8888/?token=7c8e9d6a158470b699bc77c989efa7d3838d6ecd7323552a>

```
[I 21:35:00.342 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

```
[W 21:35:01.471 NotebookApp] 404 GET /static/components/moment/locale/ko.js?v=20171216213459 (:1) 6.93ms referer=http://localhost:8888/tree
```

```
pygmentize -f rtf FILE | pbcopy
```

Files

Running

Clusters

Select items to perform actions on them.

Upload

New ▾



0 ▾



Name ▾

Last Modified

The notebook list is empty.

Upload

New ▾



Notebook:

Python 3

Other:

Text File

Folder

Terminal



```
In [1]:
cell
```

```
a = 3
b = 3
a + b
```

```
Out[1]: 6
```

- cell 단위로 실행 □□ 실행 시점에 해당 코드가 memory에 올라감
- 실행 명령어 ctrl + enter , shift + enter, alt + enter

File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3











Code



단축키 존재

jupyter-notebook command group	
change cell to code	(command mode) Y
change cell to heading 1	(command mode) 1
change cell to heading 2	(command mode) 2
change cell to heading 3	(command mode) 3
change cell to heading 4	(command mode) 4
change cell to heading 5	(command mode) 5
change cell to heading 6	(command mode) 6
change cell to markdown	(command mode) M
change cell to raw	(command mode) R
clear all cells output	

주요 단축키 (command mode - cell 밖 상태)

- 아래 셀이랑 합치기 : shift + M
- 셀 내려두기 : x 셀 copy : c 셀 붙여넣기 : v or shift + v
- 셀 지우기 : d, d 셀 지우기 취소 : z
- Markdown 변환 : m, m Code로 변환 : y, y

주요 단축키 (edit mode - cell 안 상태)

- 툴팁 표시하기 : Shift + Tab
- 들여쓰기 사용하기 : ctrl +] or ctrl + [
- 셀 나누기 : ctrl + shift + -

문서로서의 Notebook

- Jupyter Notebook은 Notebook (ipynb) 자체가 하나의 보고서
- 문서로서의 기능을 위해 markdown 과 latex 기반 수식 표현 제공

<https://jupyter.brynmawr.edu/services/public/dblank/Jupyter%20Notebook%20Users%20Manual.ipynb>

<https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>

<https://daringfireball.net/projects/markdown/>

<https://github.com/rasbt/python-machine-learning-book/blob/master/code/ch02/ch02.ipynb>

TEAMLAB

Human knowledge belongs to the world.

The concept of a feature

An understanding of data

**Director of TEAMLAB
Sungchul Choi**



**모델을 학습할 때
영향을 주는 것들**

주어진 Y값: 종속변수

주어진 X값: 독립변수

$$y = ax + b$$

알고리즘을 통해
최적값을 찾음

**Y값에 영향을 주는
X값은 하나인가?**

Y값에 대한 정보를 주는
X값은 하필인가?

Boston House Price Dataset

- 머신 러닝 등 데이터 분석을 처음 배울 때,
가장 대표적으로 사용하는 **Example Dataset**
- 1978년에 발표된 데이터로, 미국 인구통계 조사 결과
미국 보스턴 지역의 주택 가격에 영향 요소들을 정리함

<http://lib.stat.cmu.edu/datasets/boston>

Boston House Price Dataset

X 변수
13개

Y 변수

[01]	CRIM	자치시(town) 별 1인당 범죄율
[02]	ZN	25,000 평방피트를 초과하는 거주지역의 비율
[03]	INDUS	비소매상업지역이 점유하고 있는 토지의 비율
[04]	CHAS	찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)
[05]	NOX	10ppm 당 농축 일산화질소
[06]	RM	주택 1가구당 평균 방의 개수
[07]	AGE	1940년 이전에 건축된 소유주택의 비율
[08]	DIS	5개의 보스턴 직업센터까지의 접근성 지수
[09]	RAD	방사형 도로까지의 접근성 지수
[10]	TAX	10,000 달러 당 재산세율
[11]	PTRATIO	자치시(town)별 학생/교사 비율
[12]	B	$1000(Bk - 0.63)^2$, 여기서 Bk는 자치시별 흑인의 비율을 말함.
[13]	LSTAT	모집단의 하위계층의 비율(%)
[14]	MEDV	본인 소유의 주택가격(중앙값) (단위: \$1,000)

<http://www.dator.co.kr/ctg258/textyle/1721307>

<http://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html>

$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 \\ + \beta_6 x_6 + \beta_7 x_7 + \cdots \beta_{13} x_{13} + \beta_0 \cdot 1$$

13개의 x변수, 1개의 y변수

X변수의 실제 데이터는 특징(feature)을 나타냄

Feature

- 머신러닝에서 데이터의 특징을 나타내는 변수
- feature, 독립변수, input 변수 등은 동의의미로 사용
- 일반적으로 Table 상에 Data를 표현할 때, Column을 의미
- 하나의 data instance (실제 데이터)는 feature vector로 표현

Feature

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	CAT. MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24	0
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6	0
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	1
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	1
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2	1
0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7	0
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9	0
0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	27.1	0
0.21124	12.5	7.87	0	0.524	5.631	100	6.0821	5	311	15.2	386.63	29.93	16.5	0
0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.1	18.9	0
0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15	0
0.11747	12.5	7.87	0	0.524	6.009	82.9	6.2267	5	311	15.2	396.9	13.27	18.9	0
0.09378	12.5	7.87	0	0.524	5.889	39	5.4509	5	311	15.2	390.5	15.71	21.7	0
0.62976	0	8.14	0	0.538	5.949	61.8	4.7075	4	307	21	396.9	8.26	20.4	0
0.63796	0	8.14	0	0.538	6.096	84.5	4.4619	4	307	21	380.02	10.26	18.2	0

Feature

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	CAT. MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24	0

$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 + w_7x_7 + \cdots w_{13}x_{13} + w_0 \cdot 1$$

Feature vector

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	CAT. MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24	0

$$\mathbf{x}^{(1)} = \begin{bmatrix} 1 \\ 0.00632 \\ 18 \\ 2.31 \\ 0.538 \\ \vdots \\ 24 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_{13} \end{bmatrix}$$

※ Scalar는 이탤릭체, vector는 소문자 볼드, matrix는 대문자 볼드

Feature

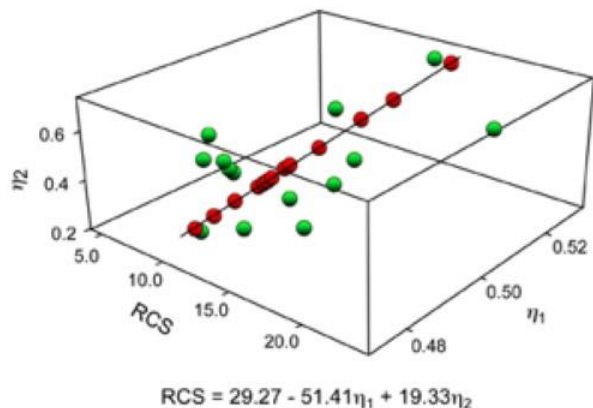
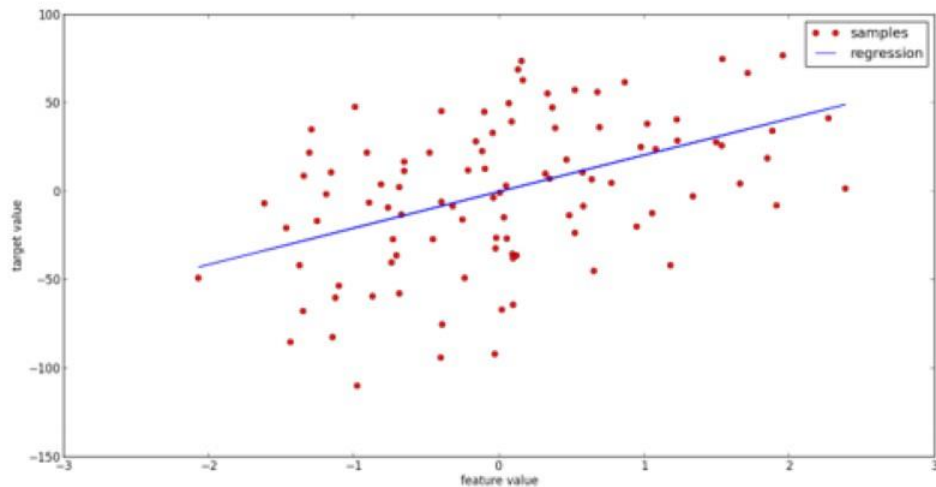
CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	CAT. MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24	0

$$y = w_1x_1 + w_2x_2 \dots w_{13}x_{13} + w_0x_0$$

$$= \sum_{i=0}^{13} w_i x_i = \mathbf{w}^T \mathbf{x}$$

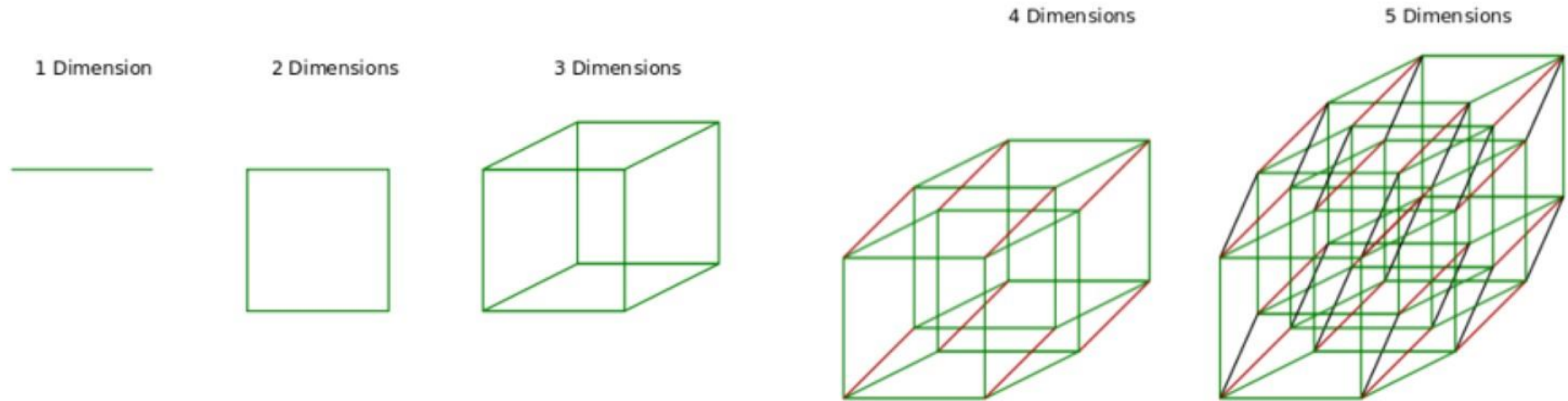
Feature의 개수?

Feature가 1개 일 때, Feature가 2개 일 때



<https://goo.gl/d1zRGq>

Feature가 n개 일 때?



Curse of dimensionality

<https://goo.gl/mCg5nu>

차원의 저주(curse of dimensionality)

- 데이터의 차원이 증가할 수록 (= feature가 증가할 수록)

데이터를 표현하는 공간이 증가하기 때문에

- 1) 희박한 벡터가 증가 (값이 없는 feature가 늘어남)
- 2) 샘플데이터가 급속도로 늘어남

- 데이터 분포나 모델 추정의 어려움이 생김

TEAMLAB

Human knowledge belongs to the world.

Data attributes

An understanding of data

**Director of TEAMLAB
Sungchul Choi**



CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	CAT. MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24	0

$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 + w_7x_7 + \cdots w_{13}x_{13} + w_0 \cdot 1$$

x_i 에는 어떤 종류의
값들이 들어갈까?

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTATIO	B	LSTAT	MEDV	CAT. MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24	0

$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 \\ + w_6x_6 + w_7x_7 + \cdots w_{13}x_{13} + w_0 \cdot 1$$

Feature별로 Data의 유형이 다름

DB를 알면 굳이 몰라도 되는 내용...

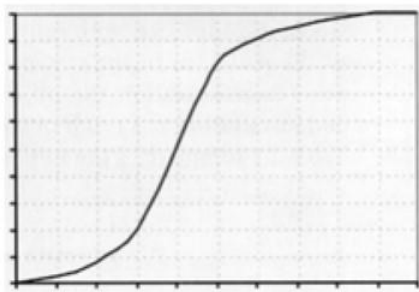
연속형 값 vs 이산형 값

continuous

값이 끊어지지 않고 연결됨

온도, 시험평균 점수, 속도

일반적으로 실수 값들



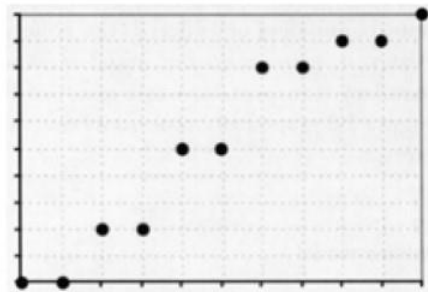
<https://goo.gl/1sSRS>
V

discrete

값이 연속적이지 않음

성별, 우편주소, 등수

Label로 구분되는 값들



Numeric Types

- 정량적으로 측정 가능한 data type
- 일반적으로 정수(integer) 또는 실수(real-number)로 표현
- 온도, 자동차 속도, 날짜의 차이(year or day)
- 단위(scale)이 있는 Interval-scaled type
- 비율이 있는 Ratio-scaled type

A diagram showing a set notation $\{x \in \mathbb{R} \mid x \geq 3\}$ enclosed in yellow curly braces. The expression is split by a vertical red line. Below the first part, $x \in \mathbb{R}$, is the text "type of number" with a blue arrow pointing to it. Below the second part, $x \geq 3$, is the text "conditions" with a blue arrow pointing to it. Between the two parts, the text "such that" is written in red.

<https://goo.gl/S7W69>

t

Nominal Types

- 범주(category)로 분류가 가능한 data type
- 명목 척도라는 표현으로 사용되기도 함
- 색깔, 학교명, ID, 전공명 등
- 두 개의 Category만 분류할 때는 Binary Type으로 구별

Ordinal Types

- 범주(category)로 분류가 가능하나 범주간의 순서가 있음
- 명목 척도라는 표현으로 사용되기도 함
- 음료수 병의 크기, 학점, 5점 척도 설문조사
- 측정되는 Scale 또는 Unit이 사람마다 다를 수 있음
- 순서가 있는 것 **≠** 배수로 증가하는 개념은 다름

실제 값을 넣어보면...

$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 + w_7x_7 + \cdots w_{13}x_{13} + w_0 \cdot 1$$

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	CAT. MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24	0
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6	0
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	1
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	1
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2	1
0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7	0
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9	0
0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	27.1	0
0.21124	12.5	7.87	0	0.524	5.631	100	6.0821	5	311	15.2	386.63	29.93	16.5	0
0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.1	18.9	0
0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15	0
0.11747	12.5	7.87	0	0.524	6.009	82.9	6.2267	5	311	15.2	396.9	13.27	18.9	0
0.09378	12.5	7.87	0	0.524	5.889	39	5.4509	5	311	15.2	390.5	15.71	21.7	0
0.62976	0	8.14	0	0.538	5.949	61.8	4.7075	4	307	21	396.9	8.26	20.4	0
0.63796	0	8.14	0	0.538	6.096	84.5	4.4619	4	307	21	380.02	10.26	18.2	0

생길 수 있는 문제점들

- 데이터의 최대/최수가 다름 $\Rightarrow$ Scale에 따른 y값에 영향
- Ordinary 또는 Nominal 한 값 들의 표현은 어떻게?
- 잘 못 기입된 값들에 대한 처리
- 값이 없을 경우는 어떻게?
- 극단적으로 큰 값 또는 작은 값들은 그대로 놔둬야 하는가?

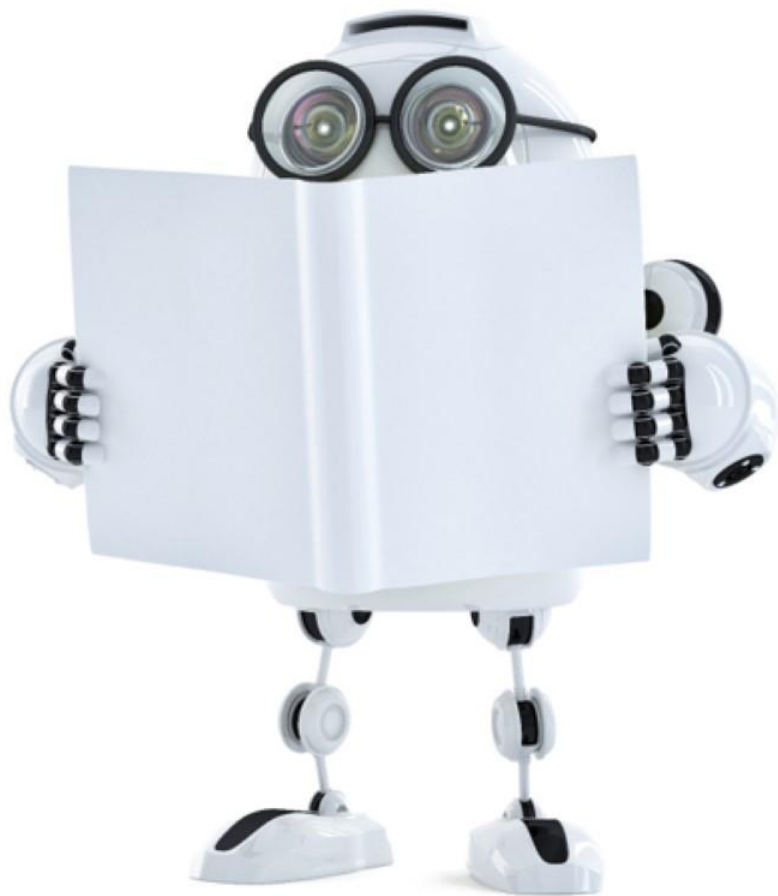
TEAMLAB

Human knowledge belongs to the world.

Loading data with pandas

An understanding of data

**Director of TEAMLAB
Sungchul Choi**



**우리의 데이터는
누가 처리한다?**

컴퓨터...

그러려면 먼저 불러오기 부터

전에 먼저...

Data table, Sample

attribute, field, feature, column

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	CAT. MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24	0
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6	0
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	1
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	1
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2	1
0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7	0
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9	0
0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	27.1	0
0.21124	12.5	7.87	0	0.524	5.631	100	6.0821	5	311	15.2	386.63	29.93	16.5	0
0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.1	18.9	0
0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15	0
0.11747	12.5	7.87	0	0.524	6.009	82.9	6.2267	5	311	15.2	396.9	13.27	18.9	0
0.09378	12.5	7.87	0	0.524	5.889	39	5.4509	5	311	15.2	390.5	15.71	21.7	0
0.62976	0	8.14	0	0.538	5.949	61.8	4.7075	4	307	21	396.9	8.26	20.4	0
0.63796	0	8.14	0	0.538	6.096	84.5	4.4619	4	307	21	380.02	10.26	18.2	0

instance, tuple, row

**Feature
vector**

data

데이터의 형식

- 일반적으로 데이터분석시 사용하는 Raw data는 Binary가 아닌 text 형태의 데이터
- 주로 사용되는 데이터 포맷은 csv, json, xml 등
- pandas를 사용하여 데이터를 호출함

Padas

엑셀처럼 데이터 사용

- 구조화된 데이터의 처리를 지원하는 Python 라이브러리
- 고성능 Array 계산 라이브러리인 Numpy와 통합하여,
강력한 “스프레드시트” 처리 기능을 제공
- 인덱싱, 연산용 함수, 전처리 함수 등을 제공함

Pandas 설치

```
conda create -n ml_scratch python=3.6 # 가상환경생성
activate ml_scratch # 가상환경실행
conda install pandas # pandas 설치
```

```
jupyter notebook # 주피터 실행하기
```

데이터 로딩

```
In [1]: import pandas as pd #라이브러리 호출
```

```
In [2]: data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data' #Data URL  
df_data = pd.read_csv(data_url, sep='#s+', header = None) #csv 타임 데이터 로드, separate는 빈공간으로 지정하고, Column은 없음
```

```
In [3]: df_data.head() #처음 다섯줄 출력
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

Column 지정

```
In [4]: df_data.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']  
# Column Header 이름 지정  
df_data.head()
```

Out [4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	CAT. MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24	0
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6	0
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	1
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	1
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2	1
0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7	0
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9	0
0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	27.1	0
0.21124	12.5	7.87	0	0.524	5.631	100	6.0821	5	311	15.2	386.63	29.93	16.5	0
0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.1	18.9	0
0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15	0
0.11747	12.5	7.87	0	0.524	6.009	82.9	6.2267	5	311	15.2	396.9	13.27	18.9	0
0.09378	12.5	7.87	0	0.524	5.889	39	5.4509	5	311	15.2	390.5	15.71	21.7	0
0.62976	0	8.14	0	0.538	5.949	61.8	4.7075	4	307	21	396.9	8.26	20.4	0
0.63796	0	8.14	0	0.538	6.096	84.5	4.4619	4	307	21	380.02	10.26	18.2	0

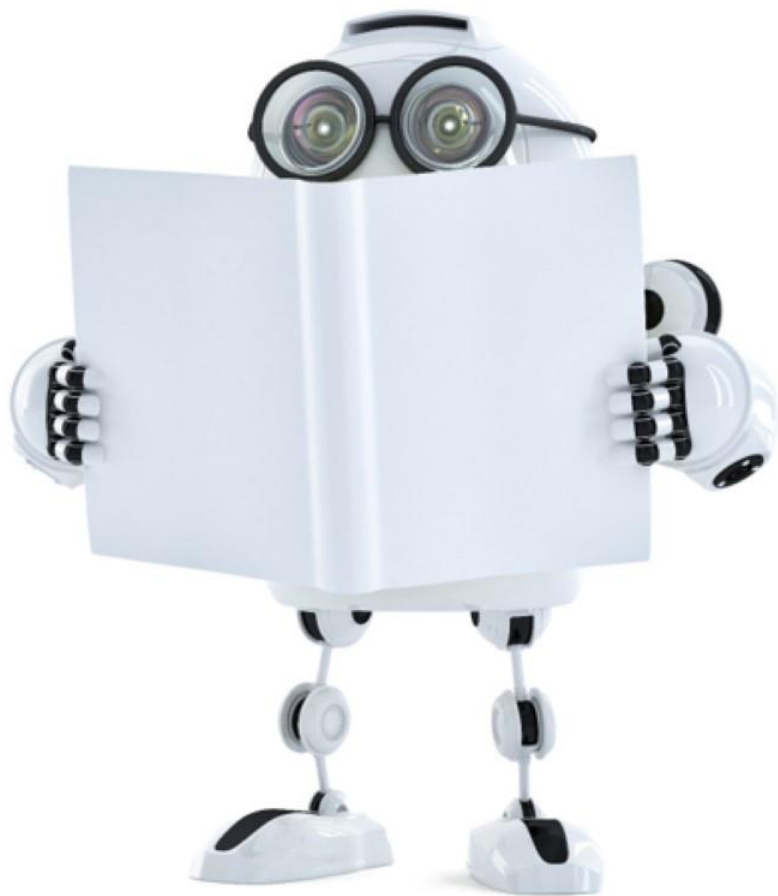
TEAMLAB

Human knowledge belongs to the world.

Representing a model

An understanding of data

**Director of TEAMLAB
Sungchul Choi**



**데이터는 불렀다
그 다음은?**

$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 \\ + \beta_6 x_6 + \beta_7 x_7 + \cdots \beta_{13} x_{13} + \beta_0 \cdot 1$$

이 식을 표현하고 싶다

컴퓨터로 저 식을 다 표현할 수 있을까?

<https://goo.gl/3PoYSx>

<https://goo.gl/1DgF83>

<https://goo.gl/6uDPX4>

있다

sympy

그러나 더 쉽게
표현하고 싶다.

비밀은 Vector+Matrix

$$y = w_1x_1 + w_2x_2 + \cdots + w_{13}x_{13} + w_0x_0$$

$$= \sum_{j=0}^{13} w_j x_j = \mathbf{w}^T \mathbf{x}$$

벡터를 Array로 표현하기

$$\mathbf{w}^T \mathbf{x}$$

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_{13} \end{bmatrix}$$

$$\mathbf{x}^{(1)} = \begin{bmatrix} 1 \\ 0.00632 \\ 18 \\ 2.31 \\ 0.538 \\ \vdots \\ 24 \end{bmatrix}$$

numpy

벡터연산 최적 라이브러리

Numpy

- Numerical Python
- 파이썬의 고성능 과학 계산을 위한 기초 패키지
- Matrix와 Vector와 같은 Array 연산의 사실상의 표준
- 한글로 넘파이로 주로 통칭, 넘피/눔파이라고 부르기도 함
- 일반 List에 비해 빠르고, 메모리 효율적
- 반복문 없이 데이터 배열에 대한 처리를 지원함
- 선형대수와 관련된 다양한 기능을 제공함
- C, C++, 포트란등의 언어와 통합 가능

Numpy 설치

```
activate ml_scratch      # 가상환경실행  
conda install numpy      # numpy 설치  
jupyter notebook # 주피터 실행하기
```


Numpy 계산해보기

$$\mathbf{w}^T \mathbf{x} \quad \mathbf{w} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{x}^{(1)} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}$$

```
In [1]: import numpy as np # Numpy 라이브러리 호출
```

```
In [2]: weight_vector = np.array([[1], [1], [1]]) # Weight Vector  
x_vector = np.array([[3], [4], [5]])
```

```
In [3]: weight_vector.T.dot(x_vector) # 1 * 3 + 1 * 4 + 1 * 5 = 12
```

```
Out[3]: array([[12]])
```

그럼 이건???

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	CAT. MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24	0
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6	0
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	1
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	1
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2	1
0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7	0
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9	0
0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	27.1	0
0.21124	12.5	7.87	0	0.524	5.631	100	6.0821	5	311	15.2	386.63	29.93	16.5	0
0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.1	18.9	0
0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15	0
0.11747	12.5	7.87	0	0.524	6.009	82.9	6.2267	5	311	15.2	396.9	13.27	18.9	0
0.09378	12.5	7.87	0	0.524	5.889	39	5.4509	5	311	15.2	390.5	15.71	21.7	0
0.62976	0	8.14	0	0.538	5.949	61.8	4.7075	4	307	21	396.9	8.26	20.4	0
0.63796	0	8.14	0	0.538	6.096	84.5	4.4619	4	307	21	380.02	10.26	18.2	0

Pandas + Numpy

$$\text{for each } i \sum_{j=0}^{13} w_j x_j^{(i)} = \underline{X} \cdot \underline{\mathbf{w}}$$

$x_0 = 1$

데이터 Sample **Weight number** **Data table**

Pandas 데이터 로딩

```
In [4]: import pandas as pd
```

```
In [5]: data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data' #Data URL
df_data = pd.read_csv(data_url, sep='#s+', header = None) #csv 타임 데이터 로드, separate는 빈공간으로 지정하고, Column은 없음
df_data.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
df_data.head()
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

```
In [6]: df_data['weight_0'] = 1
df_data= df_data.drop("MEDV", axis=1)
df_data.head()
```

Out[6]:

X0 추가, Y 값 없애기

```
In [6]: df_data['weight_0'] = 1 # weight 0 값 추가  
df_data= df_data.drop("MEDV", axis=1) #Y값 제거  
df_data.head()
```

Out[6]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	weight_0
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	1
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	1
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	1
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	1
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	1

Dataframe ↔ numpy, 곱셈연산

```
In [9]: df_matrix = df_data.as_matrix() # Matrix Data로 변환하기  
weight_vector = np.random.random_sample((14, 1))
```

```
In [10]: df_matrix.dot(weight_vector)
```

```
Out[10]: array([[ 433.52000019],  
[ 394.83907923],  
[ 390.44525364],  
[ 377.71929915],  
[ 380.42922667],  
[ 378.55996125],  
[ 442.05358228],  
[ 446.92169217],  
[ 442.16560852],  
[ 439.29722575],  
[ 444.48909317],  
[ 444.65609699],  
[ 436.55549191],  
[ 429.5979006 ],  
[ 421.84837601],  
[ 428.20934742],  
[ 420.27744009],  
[ 426.35315551],  
[ 360.92936403],  
[ 426.96198486].
```

for each i $\sum_{j=0}^{13} w_j x_j^{(i)} = X \cdot \mathbf{w}$

TEAMLAB

Human knowledge belongs to the world.

Numerical Python - numpy

Data handling section

**Director of TEAMLAB
Sungchul Choi**



어떻게 수식을
코드로 표현할 것인
가?

코드로 방정식 표현하기

$$\begin{bmatrix} 2 & 2 & 1 & 9 \\ 2 & -1 & 2 & 6 \\ 1 & -1 & 2 & 5 \end{bmatrix}$$

```
coefficient_matrix = [[2, 2, 1], [2, -1, 2], [1, -1, 2]]  
constant_vector = [9, 6, 5]
```

코드로 방정식 표현하기

```
coefficient_matrix = [[2, 2, 1], [2, -1, 2], [1, -1, 2]]  
constant_vector = [9, 6, 5]
```

- 다양한 Matrix 계산을 어떻게 만들 것 인가?
- 굉장히 큰 Matrix에 대한 표현
- 처리 속도 문제 – 파이썬은 Interpreter 언어

적절한 패키지를 활용하는 건 좋은 방법

파이썬 과학 처리 패키지

Numpy

Numpy

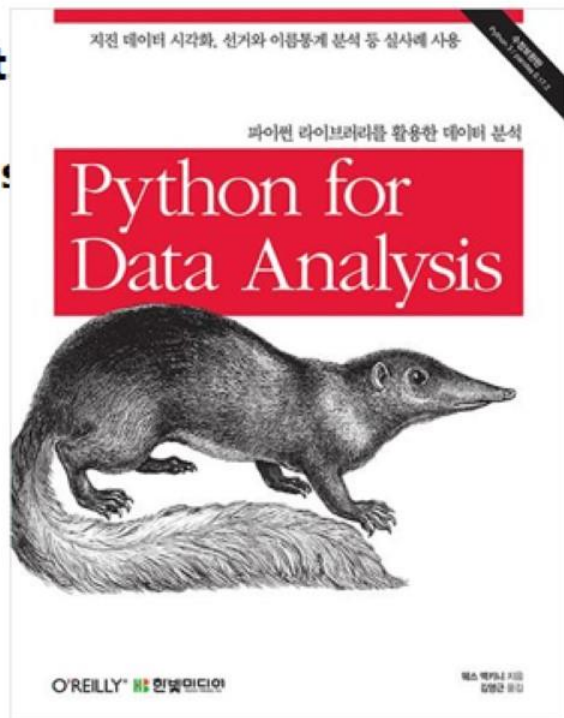
- Numerical Python
- 파이썬의 고성능 과학 계산용 패키지
- Matrix와 Vector와 같은 Array 연산의 사실상의 표준
- 한글로 넘파이로 주로 통칭, 넘피/눔파이라고 부르기도 함

Numpy 특징

- 일반 List에 비해 빠르고, 메모리 효율적
- 반복문 없이 데이터 배열에 대한 처리를 지원함
- 선형대수와 관련된 다양한 기능을 제공함
- C, C++, 포트란 등의 언어와 통합 가능

References

- cs231 - <http://cs231n.github.io/python-numpy-tutorial/#numpy>
- <https://docs.scipy.org/doc/numpy-dev/user/quickstart>
- 데이터 사이언스 스쿨 (파이썬 버전) - <https://goo.gl/3hs>
- Numpy - <https://goo.gl/7Nwjvv>
- 파이썬 라이브러리를 활용한 데이터 분석



Numpy Install

```
activate ml_scratch  
conda install numpy
```

- Windows 환경에선 conda로 패키지 관리 필요
(C 패키지 핸들링 등)
- jupyter 등을 설치한 상태에서는 추가 설치 필요 없음

ndarray

import

```
import numpy as np
```

- numpy의 호출 방법
- 일반적으로 numpy는 np라는 alias(별칭) 이용해서 호출함
- 특별한 이유는 없음 세계적인 약속 같은 것

Array creation

```
test_array = np.array([1, 4, 5, 8], float)
```

```
print(test_array)
```

```
type(test_array[3])
```

- numpy는 np.array 함수를 활용하여 배열을 생성함 □ ndarray
- numpy는 하나의 데이터 type만 배열에 넣을 수 있음
- List와 가장 큰 차이점, Dynamic typing not supported
- C의 Array를 사용하여 배열을 생성함

Array creation

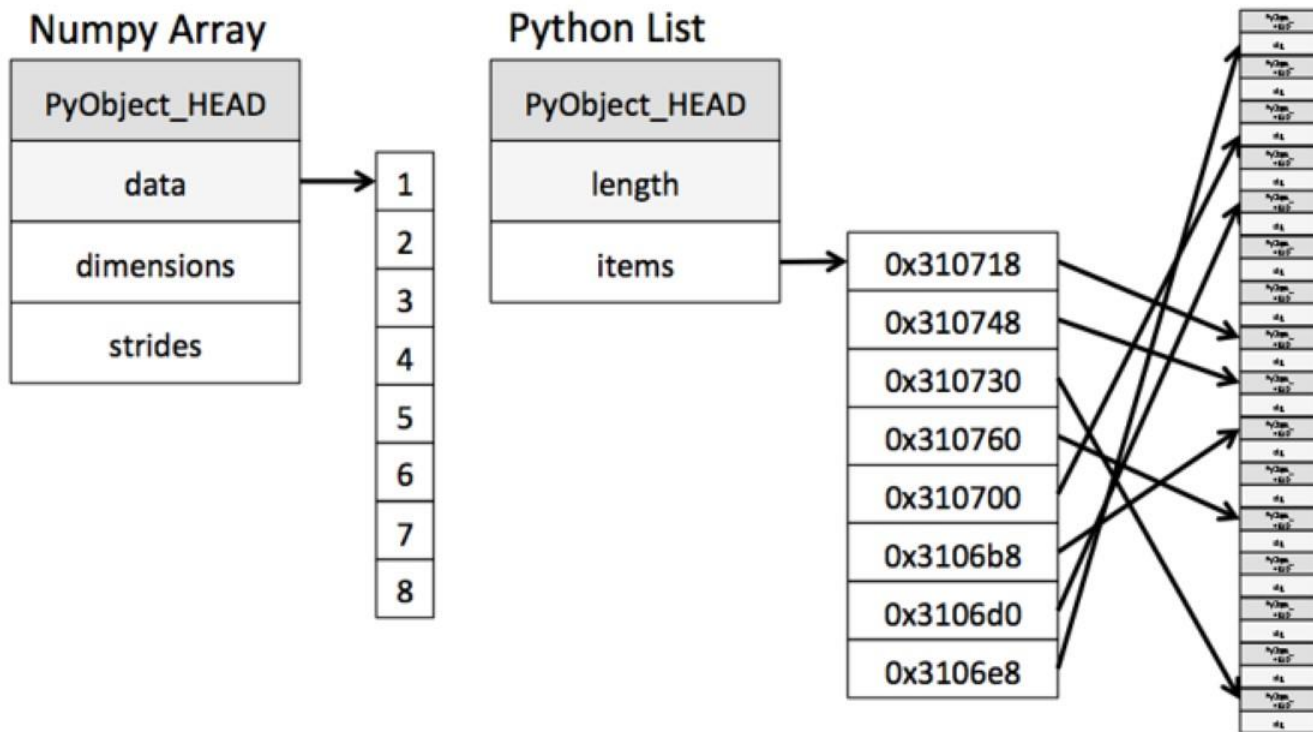
```
test_array = np.array([1, 4, 5, 8], float)
test_array
```

```
array([ 1.,  4.,  5.,  8.])
```

```
type(test_array[3])
```

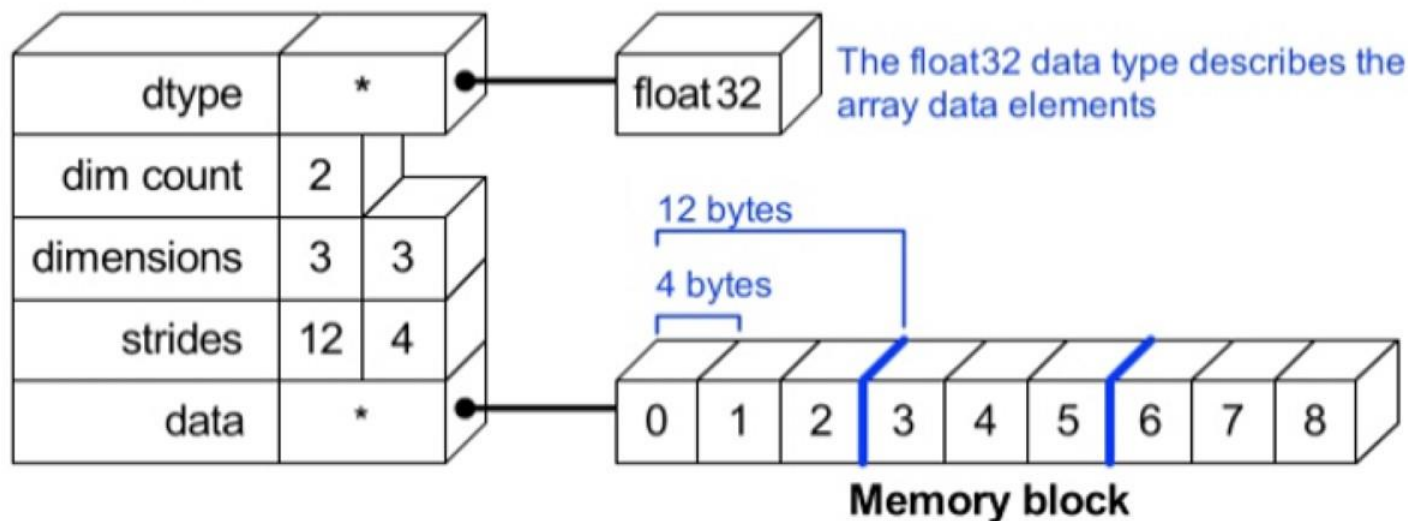
```
numpy.float64
```

Array creation



Array creation

NDArray Data Structure



Array creation

```
test_array = np.array([1, 4, 5, "8"], float)    # String Type의 데이터를 입력해도
print(test_array)
print(type(test_array[3]))                     # Float Type으로 자동 형변환을 실시
print(test_array.dtype)                       # Array(배열) 전체의 데이터 Type을 반환함
print(test_array.shape)                       # Array(배열)의 shape을 반환함
```

- **shape** : numpy array의 object의 dimension 구성을 반환함
- **dtype** : numpy array의 데이터 type을 반환함

Array creation

```
test_array = np.array([1, 4, 5, "8"], float) # String Type의 데이터를 입력해도  
test_array
```

```
array([ 1.,  4.,  5.,  8.])
```

```
type(test_array[3]) # Float Type으로 자동 형변환을 실시
```

```
numpy.float64
```

```
test_array.dtype # Array(배열) 전체의 데이터 Type을 반환함
```

```
dtype('float64')
```

```
test_array.shape # Array(배열) 의 shape을 반환함
```

```
(4,)
```


Array shape (vector)

- Array (vector, matrix, tensor)의 크기, 형태 등에 대한 정보

```
test_array = np.array([1, 4, 5, "8"], float) # String Type의 데이터를 입력해도  
test_array
```

```
array([ 1.,  4.,  5.,  8.])
```

1	4	5	8
---	---	---	---



(4,)

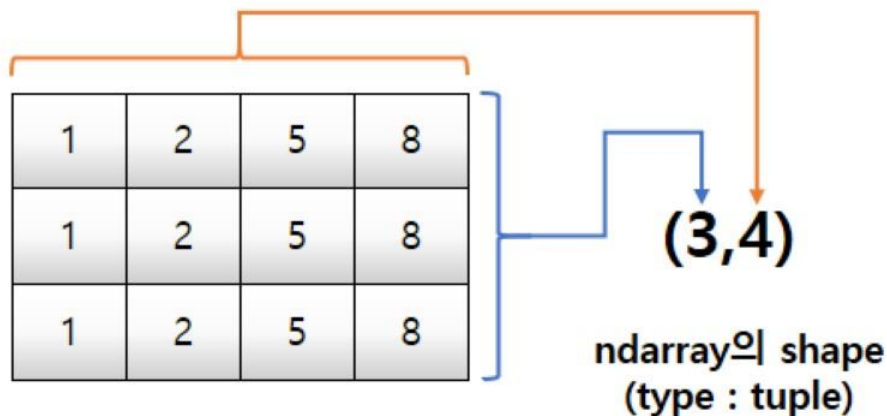
ndarray의 구성

ndarray의 shape
(type : tuple)

Array shape (matrix)

```
matrix = [[1,2,5,8],[1,2,5,8],[1,2,5,8]]  
np.array(matrix, int).shape
```

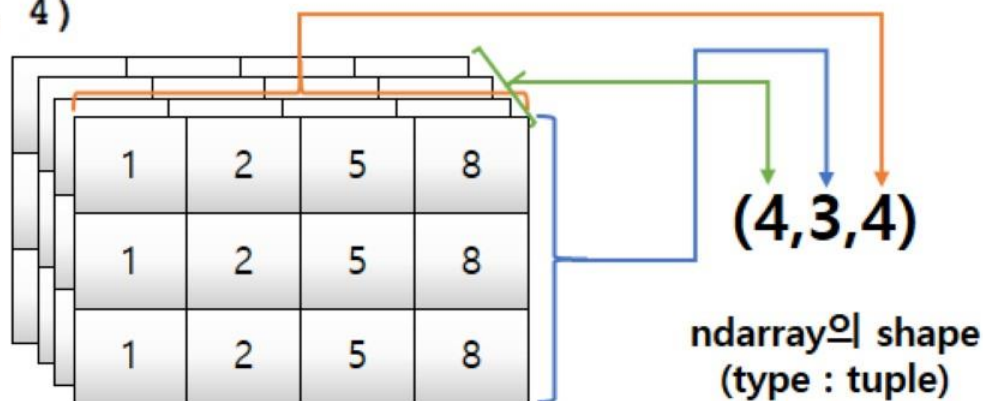
(3, 4)



Array shape (3rd order tensor)

```
tensor = [[[1,2,5,8],[1,2,5,8],[1,2,5,8]],  
          [[1,2,5,8],[1,2,5,8],[1,2,5,8]],  
          [[1,2,5,8],[1,2,5,8],[1,2,5,8]],  
          [[1,2,5,8],[1,2,5,8],[1,2,5,8]]]  
np.array(tensor, int).shape
```

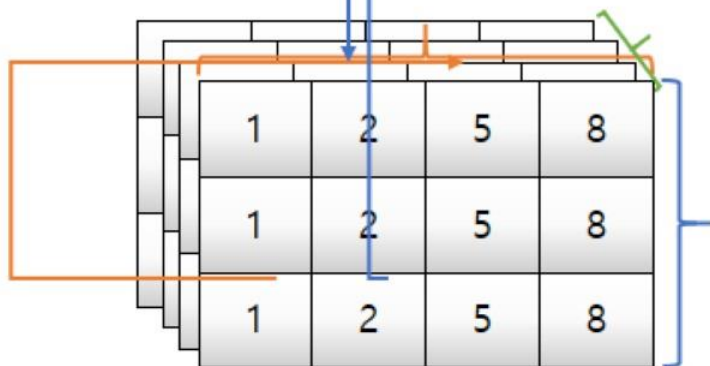
(4, 3, 4)



Array shape - ndim & size

- ndim - number of dimension
- size - data의 개수

```
tensor = [[[1,2,5,8],[1,2,5,8],[1,2,5,8]],  
          [[1,2,5,8],[1,2,5,8],[1,2,5,8]],  
          [[1,2,5,8],[1,2,5,8],[1,2,5,8]],  
          [[1,2,5,8],[1,2,5,8],[1,2,5,8]]]
```



```
np.array(tensor, int).ndim
```

3

```
np.array(tensor, int).size
```

48

(4,3,4)

ndarray의 shape
(type : tuple)

Array dtype

- Narray의 single element가 가지는 data type
- 각 element가 차지하는 memory의 크기가 결정됨

```
np.array([[1, 2, 3], [4.5, 5, 6]], dtype=int)
```

```
array([[1, 2, 3],  
       [4, 5, 6]])
```

Data type을 integer로 선언

```
np.array([[1, 2, 3], [4.5, "5", "6"]], dtype=np.float32)
```

```
array([[ 1. ,  2. ,  3. ],  
       [ 4.5,  5. ,  6. ]], dtype=float32)
```

Data type을 float로 선언

Array dtype

- C의 data type과 compatible

Basic Type	Available NumPy types
Boolean	bool
Integer	int8, int16, int32, int64, int128, int
Unsigned Integer	uint8, uint16, uint32, uint64, uint128, uint
Float	float32, float64, float, longfloat,
Complex	complex64, complex128, complex
Strings	str, unicode
Object	object
Records	void

Data Types	Size in Bytes	Can contain:
boolean	1	true (1) or false (0)
char	1	ASCII character or signed value between -128 and 127
unsigned char, byte, uint8_t	1	ASCII character or unsigned value between 0 and 255
int, short	2	signed value between -32,768 and 32,767
unsigned int, word, uint16_t	2	unsigned value between 0 and 65,535
long	4	signed value between -2,147,483,648 and 2,147,483,647
unsigned long, uint32_t	4	unsigned value between 0 and 4,294,967,295
float, double	4	floating point value between -3.4028235E+38 and 3.4028235E+38 (Note that double is the same as a float on this platform.)

Array dtype

- nbytes – ndarray object의 메모리 크기를 반환함

```
np.array([[1, 2, 3], [4.5, "5", "6"]], dtype=np.float32).nbytes
```

24

32bits = 4bytes □□ 6 * 4bytes

```
np.array([[1, 2, 3], [4.5, "5", "6"]], dtype=np.int8).nbytes
```

6

8bits = 1bytes □□ 6 * 1bytes

```
np.array([[1, 2, 3], [4.5, "5", "6"]], dtype=np.float64).nbytes
```

48

64bits = 8bytes □□ 6 * 8bytes

Handling shape

reshape

- Array의 shape의 크기를 변경함 (element의 갯수는 동일)

1	2	5	8
1	2	5	8

(2,4)



1	2	5	8	1	2	5	8
---	---	---	---	---	---	---	---

(8,)

reshape

- Array의 shape의 크기를 변경함 (element의 갯수는 동일)

```
test_matrix = [[1,2,3,4], [1,2,5,8]]  
np.array(test_matrix).shape
```

(2, 4)

```
np.array(test_matrix).reshape(8,)
```

array([1, 2, 3, 4, 1, 2, 5, 8])

```
np.array(test_matrix).reshape(8,).shape
```

(8,)

reshape

- Array의 size만 같다면 다차원으로 자유로이 변형가능

```
np.array(test_matrix).reshape(2,4).shape
```

(2, 4)

```
np.array(test_matrix).reshape(-1,2).shape
```

(4, 2)

-1: size를 기반으로 row 개수 선정

```
np.array(test_matrix).reshape(2,2,2)
```

```
array([[[1, 2],  
        [3, 4]],  
       [[1, 2],  
        [5, 8]]])
```

```
np.array(test_matrix).reshape(2,2,2).shape
```

(2, 2, 2)

flatten

- 다차원 array를 1차원 array로 변환

1	2	5	8
1	2	5	8

(2,2,4)



1	2	5	8	1	2	5	8	1	2	5	8	1	2	5	8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

(16,)

flatten

- 다차원 array를 1차원 array로 변환

```
test_matrix = [[[1,2,3,4], [1,2,5,8]], [[1,2,3,4], [1,2,5,8]]]  
np.array(test_matrix).flatten()
```

```
array([1, 2, 3, 4, 1, 2, 5, 8, 1, 2, 3, 4, 1, 2, 5, 8])
```

Indexing & slicing

indexing

```
a = np.array([[1, 2, 3], [4.5, 5, 6]], int)
print(a)
print(a[0,0]) # Two dimensional array representation #1
print(a[0][0]) # Two dimensional array representation #2

a[0,0] = 12 # Matrix 0,0 에 12 할당
print(a)
a[0][0] = 5 # Matrix 0,0 에 12 할당
print(a)
```

- List와 달리 이차원 배열에서 [0,0] 과 같은 표기법을 제공함
- Matrix 일경우 앞은 row 뒤는 column을 의미함

indexing

```
test_exmaple = np.array([[1, 2, 3], [4.5, 5, 6]], int)
test_exmaple
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
test_exmaple[0][0]
```

1

```
test_exmaple[0,0]
```

1

indexing

```
test_exmaple[0,0] = 12 # Matrix 0,0 에 12 할당  
test_exmaple
```

```
array([[12,  2,  3],  
       [ 4,  5,  6]])
```

```
test_exmaple[0][0] = 5 # Matrix 0,0 에 12 할당  
test_exmaple[0,0]
```

slicing

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

```
a[:,2:] # 전체 Row의 2열 이상
```

```
a[1,1:3] # 1 Row의 1열 ~ 2열
```

```
a[1:3] # 1 Row ~ 2Row의 전체
```

- List와 달리 행과 열 부분을 나눠서 slicing이 가능함
- Matrix의 부분 집합을 추출할 때 유용함

slicing

1	2	5	8
1	2	5	8
1	2	5	8
1	2	5	8



1	2	5	8
1	2	5	8
1	2	5	8
1	2	5	8

`[:2,:]`

Row - 0~1 까지

column - 전체

slicing

```
test_exmaple = np.array([  
    [1, 2, 5, 8], [1, 2, 5, 8], [1, 2, 5, 8], [1, 2, 5, 8]], int)  
test_exmaple[:2,:]
```

```
array([[1, 2, 5, 8],  
       [1, 2, 5, 8]])
```

```
test_exmaple[:,1:3]  
test_exmaple[1,:2]
```

1	2	5	8
1	2	5	8
1	2	5	8
1	2	5	8

slicing

```
test_exmaple = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]], int)
test_exmaple[:,2:] # 전체 Row의 2열 이상
```

```
array([[ 3,  4,  5],
       [ 8,  9, 10]])
```

```
test_exmaple[1,1:3] # 1 Row의 1열 ~ 2열
```

```
array([7, 8])
```

```
test_exmaple[1:3] # 1 Row ~ 2Row의 전체
```

```
array([[ 6,  7,  8,  9, 10]])
```


creation function

arange

- array의 범위를 지정하여, 값의 list를 생성하는 명령어

```
np.arange(30) # range: List의 range와 같은 효과, integer로 0부터 29까지 배열추출
```

```
array([ 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])
```

(시작, 끝, step)

```
np.arange(0, 5, 0.5) # floating point도 표시가능함
```

```
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5])
```

```
np.arange(30).reshape(5,6)
```

```
array([[ 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]])
```


ones, zeros and empty

- zeros - 0으로 가득찬 ndarray 생성

`np.zeros(shape, dtype, order)`

```
np.zeros(shape=(10,), dtype=np.int8) # 10 - zero vector 생성
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int8)
```

```
np.zeros((2,5)) # 2 by 5 - zero matrix 생성
```

```
array([[ 0.,  0.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  0.,  0.]])
```

ones, zeros and empty

- ones – 1로 가득찬 ndarray 생성

`np.ones(shape, dtype, order)`

```
np.ones(shape=(10,), dtype=np.int8)
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int8)
```

```
np.ones((2,5))
```

```
array([[ 1.,  1.,  1.,  1.,  1.],  
       [ 1.,  1.,  1.,  1.,  1.]])
```

ones, zeros and empty

- empty – shape만 주어지고 비어있는 ndarray 생성 (memory initialization 이 되지 않음)

```
np.empty(shape=(10,), dtype=np.int8)
```

```
array([ 0,  0,  0,  0,  0,  0,  0, 64, -74, 105], dtype=int8)
```

```
np.empty((3,5))
```

```
array([[ 2.00000000e+000,  2.00000000e+000,  6.42285340e-323,
         0.00000000e+000,  0.00000000e+000],
       [ 0.00000000e+000,  0.00000000e+000,  0.00000000e+000,
         0.00000000e+000,  0.00000000e+000],
       [ 0.00000000e+000,  2.12199579e-314,  2.00000000e+000,
         2.00000000e+000,  3.45845952e-323]])
```

something_like

- 기존 ndarray의 shape 크기 만큼 1, 0 또는 empty array를 반환

```
test_matrix = np.arange(30).reshape(5,6)
np.ones_like(test_matrix)
```

```
array([[1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1]])
```

identity

- 단위 행렬(i 행렬)을 생성함

n $\square$ number of

rows

```
np.identity(n=3, dtype=np.int8)
```

```
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]], dtype=int8)
```

```
np.identity(5)
```

```
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1.]])
```

eye

- 대각선인 1인 행렬, k값의 시작 index의 변경이 가능

```
np.eye(N=3, M=5, dtype=np.int8)
```

```
array([[1, 0, 0, 0, 0],  
       [0, 1, 0, 0, 0],  
       [0, 0, 1, 0, 0]], dtype=int8)
```

```
np.eye(3)
```

```
array([[ 1.,  0.,  0.],  
       [ 0.,  1.,  0.],  
       [ 0.,  0.,  1.]])
```

```
np.eye(3, 5, k=2)      k □□ start index
```

```
array([[ 0.,  0.,  1.,  0.,  0.],  
       [ 0.,  0.,  0.,  1.,  0.],  
       [ 0.,  0.,  0.,  0.,  1.]])
```

diag

- 대각 행렬의 값을 추출함

0	1	2
3	4	5
6	7	8

```
matrix = np.arange(9).reshape(3,3)  
np.diag(matrix)
```

```
array([0, 4, 8])
```

0	1	2
3	4	5
6	7	8

```
np.diag(matrix, k=1)
```

k start index

```
array([1, 5])
```


random sampling

- 데이터 분포에 따른 sampling으로 array를 생성

```
np.random.uniform(0,1,10).reshape(2,5)    균등분포
```

```
array([[ 0.67406593,  0.71072857,  0.06963986,  0.09194939,  0.47293574],  
       [ 0.13840676,  0.97410297,  0.60703044,  0.04002073,  0.08057727]])
```

```
np.random.normal(0,1,10).reshape(2,5)    정규분포
```

```
array([[ 1.02694847,  0.39354215,  0.63411928, -1.03639086, -1.76669162],  
       [ 0.50628853, -1.42496802,  1.23288754,  1.26424168,  0.53718751]])
```


operation functions

sum

- ndarray의 element들 간의 합을 구함, list의 sum 기능과 동일

```
test_array = np.arange(1,11)  
test_array
```

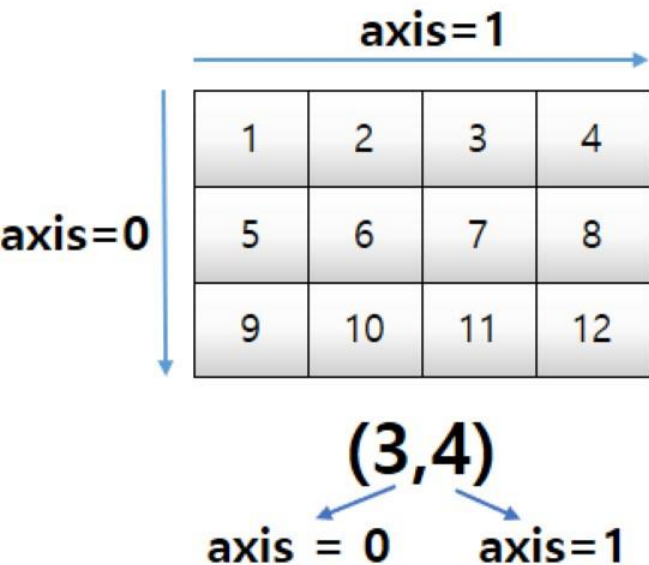
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
test_array.sum(dtype=np.float)
```

```
55.0
```

axis

- 모든 operation function을 실행할 때, 기준이 되는 dimension 축



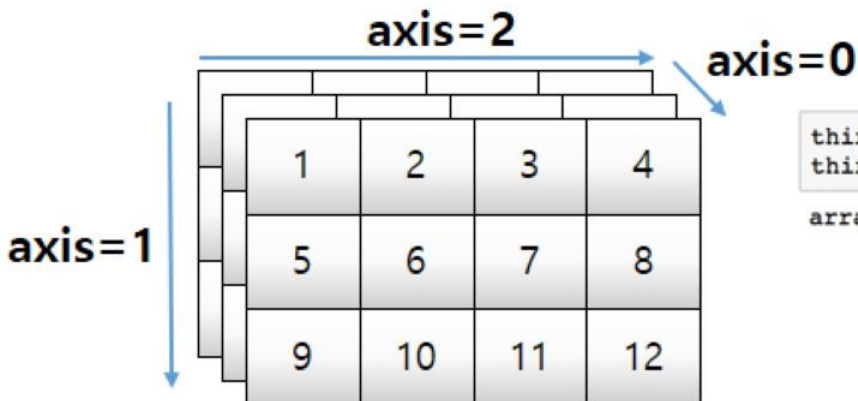
```
test_array = np.arange(1,13).reshape(3,4)
test_array
```

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

```
test_array.sum(axis=1), test_array.sum(axis=0)
(array([10, 26, 42]), array([15, 18, 21, 24]))
```

axis

- 모든 operation function을 실행할 때, 기준이 되는 dimension 축



(3,3,4)

axis = 0 axis=1 axis=2

```
third_order_tensor = np.array([test_array, test_array, test_array])  
third_order_tensor
```

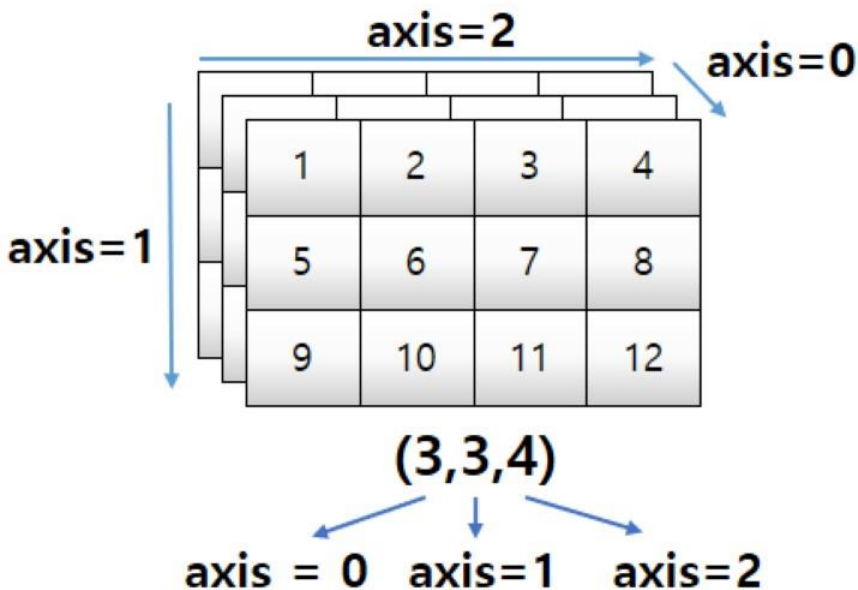
```
array([[[ 1,  2,  3,  4],  
        [ 5,  6,  7,  8],  
        [ 9, 10, 11, 12]],
```

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

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

axis

- 모든 operation function을 실행할 때, 기준이 되는 dimension 축



```
third_order_tensor.sum(axis=2)
```

```
array([[10, 26, 42],  
       [10, 26, 42],  
       [10, 26, 42]])
```

```
third_order_tensor.sum(axis=1)
```

```
array([[15, 18, 21, 24],  
       [15, 18, 21, 24],  
       [15, 18, 21, 24]])
```

```
third_order_tensor.sum(axis=0)
```

```
array([[ 3,  6,  9, 12],  
       [15, 18, 21, 24],  
       [27, 30, 33, 36]])
```

mean & std

- ndarray의 element들 간의 평균 또는 표준 편차를 반환

```
test_array = np.arange(1,13).reshape(3,4)
test_array
```

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

```
test_array.mean(), test_array.mean(axis=0)
(6.5, array([ 5.,  6.,  7.,  8.]))
```

```
test_array.std(), test_array.std(axis=0)
(3.4520525295346629,
 array([ 3.26598632,  3.26598632,  3.26598632,  3.26598632]))
```

Mathematical functions

- 그 외에도 다양한 수학 연산자를 제공함

exponential: `exp`, `expm1`, `exp2`, `log`, `log10`, `log1p`, `log2`,
`power`, `sqrt`

trigonometric: `sin`, `cos`, `tan`, `acsin`, `arccos`, `atctan`

hyperbolic: `sinh`, `cosh`, `tanh`, `acsinh`, `arccosh`, `atctanh`

Mathematical functions

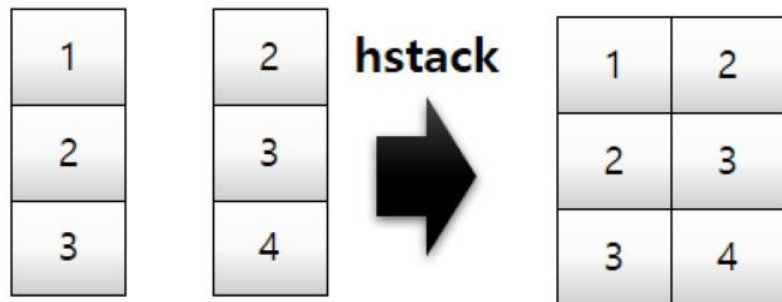
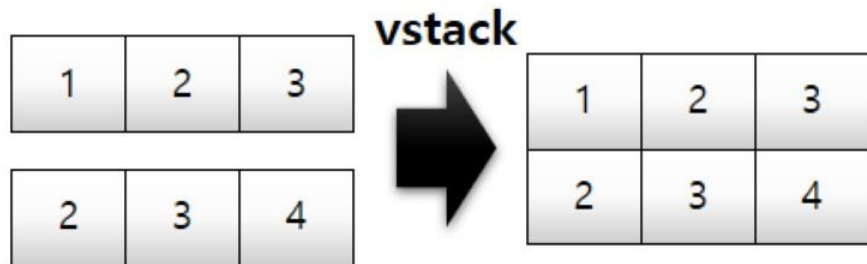
- 그 외에도 다양한 수학 연산자를 제공함 (np.something 호출)

```
np.exp(test_array), np.sqrt(test_array)
```

```
(array([[ 2.71828183e+00,  7.38905610e+00,  2.00855369e+01,
         5.45981500e+01],
       [ 1.48413159e+02,  4.03428793e+02,  1.09663316e+03,
         2.98095799e+03],
       [ 8.10308393e+03,  2.20264658e+04,  5.98741417e+04,
        1.62754791e+05]]),
array([[ 1.          ,  1.41421356,  1.73205081,  2.          ],
       [ 2.23606798,  2.44948974,  2.64575131,  2.82842712],
       [ 3.          ,  3.16227766,  3.31662479,  3.46410162]]))
```


concatenate

- Numpy array를 합치는 함수



```
a = np.array([1, 2, 3])  
b = np.array([2, 3, 4])  
np.vstack((a,b))
```

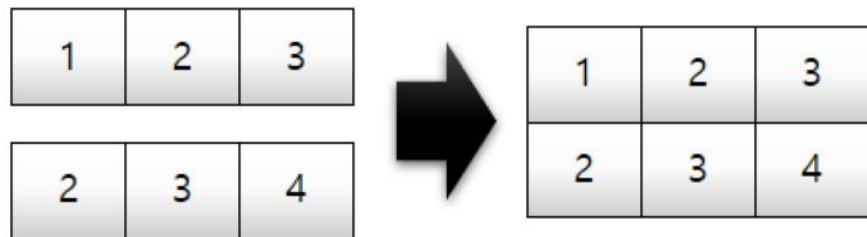
```
array([[1, 2, 3],  
       [2, 3, 4]])
```

```
a = np.array([ [1], [2], [3] ])  
b = np.array([ [2], [3], [4] ])  
np.hstack((a,b))
```

```
array([[1, 2],  
       [2, 3],  
       [3, 4]])
```

concatenate

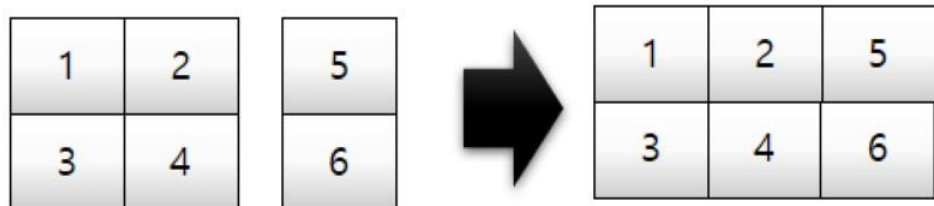
concatenate / axis=0



```
a = np.array([[1, 2, 3]])  
b = np.array([[2, 3, 4]])  
np.concatenate( (a,b) ,axis=0)
```

```
array([[1, 2, 3],  
       [2, 3, 4]])
```

concatenate / axis=1



```
a = np.array([[1, 2], [3, 4]])  
b = np.array([[5, 6]])
```

```
np.concatenate( (a,b.T) ,axis=1)
```

```
array([[1, 2, 5],  
       [3, 4, 6]])
```

array operations

Operations b/t arrays

- Numpy는 array간의 기본적인 사칙 연산을 지원함

```
test_a = np.array([[1,2,3],[4,5,6]], float)
```

```
test_a + test_a # Matrix + Matrix 연산
```

```
array([[ 2.,  4.,  6.],  
       [ 8., 10., 12.]])
```

```
test_a - test_a # Matrix - Matrix 연산
```

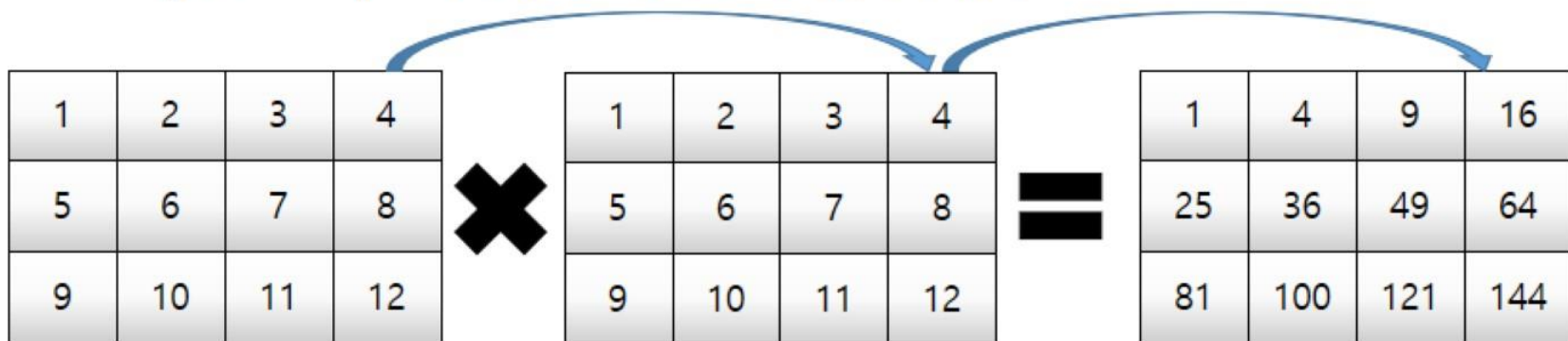
```
array([[ 0.,  0.,  0.],  
       [ 0.,  0.,  0.]])
```

```
test_a * test_a # Matrix내 element들 간 같은 위치에 있는 값들끼리 연산
```

```
array([[ 1.,  4.,  9.],  
       [16., 25., 36.]])
```

Element-wise operations

- Array간 shape이 같을 때 일어나는 연산



```
matrix_a = np.arange(1,13).reshape(3,4)  
matrix_a * matrix_a
```

```
array([[ 1,  4,  9, 16],  
       [25, 36, 49, 64],  
       [81, 100, 121, 144]])
```

Dot product

- Matrix의 기본 연산
- dot 함수 사용

"Dot Product"

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & \\ & \end{bmatrix}$$

```
test_a = np.arange(1,7).reshape(2,3)
test_b = np.arange(7,13).reshape(3,2)
```

```
test_a.dot(test_b)
```

```
array([[ 58,  64],
       [139, 154]])
```

transpose

- transpose 또는 T attribute 사용

```
test_a = np.arange(1,7).reshape(2,3)  
test_a
```

```
array([[1, 2, 3],  
       [4, 5, 6]])
```

```
test_a.T.dot(test_a) # Matrix 간 곱셈
```

```
array([[ 17.,  22.,  27.],  
       [ 22.,  29.,  36.],  
       [ 27.,  36.,  45.]])
```

```
test_a.transpose()
```

```
array([[1, 4],  
       [2, 5],  
       [3, 6]])
```

```
test_a.T
```

```
array([[1, 4],  
       [2, 5],  
       [3, 6]])
```

broadcasting

- Shape이 다른 배열 간 연산을 지원하는 기능

1	2	3
4	5	6

+

3

=

4	5	6
7	8	9

```
test_matrix = np.array([[1,2,3],[4,5,6]], float)
scalar = 3
```

```
test_matrix + scalar # Matrix - Scalar 덧셈
```

```
array([[ 4.,  5.,  6.],
       [ 7.,  8.,  9.]])
```


broadcasting

```
test_matrix - scalar # Matrix - Scalar 뺄셈
```

```
array([[ -4.,  -3.,  -2.],  
       [ -1.,   0.,   1.]])
```

```
test_matrix * 5 # Matrix - Scalar 곱셈
```

```
array([[ 5., 10., 15.],  
       [20., 25., 30.]])
```

```
test_matrix / 5 # Matrix - Scalar 나눗셈
```

```
array([[ 0.2,  0.4,  0.6],  
       [ 0.8,  1. ,  1.2]])
```

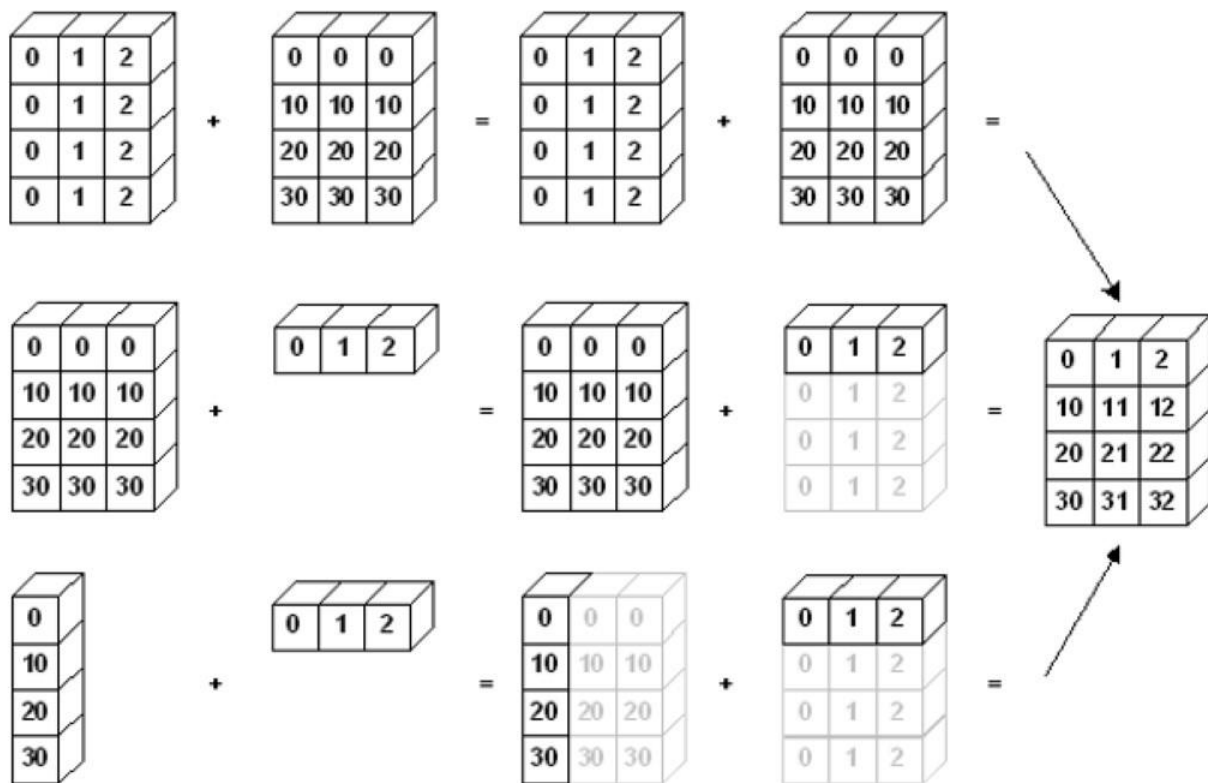
```
test_matrix // 0.2 # Matrix - Scalar 몫
```

```
array([[ 4.,  9., 14.],  
       [19., 24., 29.]])
```

```
test_matrix ** 2 # Matrix - Scalar 제곱
```

```
array([[ 1.,  4.,  9.],  
       [16., 25., 36.]])
```

- Scalar -
vector 외에도
vector -
matrix 간의 연
산도 지원



broadcasting

1	2	3
4	5	6
7	8	9
10	11	12



10	20	30
----	----	----



11	22	33
14	25	36
17	28	39
20	31	42

```
test_matrix = np.arange(1,13).reshape(4,3)
test_vector = np.arange(10,40,10)
test_matrix + test_vector
```

```
array([[11, 22, 33],
       [14, 25, 36],
       [17, 28, 39],
       [20, 31, 42]])
```

Numpy performance #1

```
def sclar_vector_product(scalar, vector):  
    result = []  
    for value in vector:  
        result.append(scalar * value)  
    return result
```

```
iteration_max = 100000000
```

```
vector = list(range(iteration_max))  
scalar = 2
```

```
%timeit sclar_vector_product(scalar, vector) # for loop을 이용한 성능
```

```
%timeit [scalar * value for value in range(iteration_max)] # list comprehension을 이용한 성능
```

```
%timeit np.arange(iteration_max) * scalar # numpy를 이용한 성능
```

- **timeit: jupyter 환경에서 코드의 퍼포먼스를 체크하는 함수**

Numpy performance #2

- 일반적으로 속도는 아래 순
for loop < list comprehension < numpy
- 100,000,000 번의 loop이 돌 때 약 약 4배 이상의 성능 차이를 보임
- Numpy는 C로 구현되어 있어, 성능을 확보하는 대신
- 파이썬의 가장 큰 특징인 dynamic typing을 포기함
- 대용량 계산에서는 가장 흔히 사용됨
- Concatenate 처럼 계산이 아닌, 할당에서는 연산 속도의 이점이 없음

comparisons

All & Any

- Array의 데이터 전부(and) 또는 일부(or)가 조건에 만족 여부 반환

```
a = np.arange(10)
a
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
np.any(a>5), np.any(a<0)    any □□ 하나라도 조건에 만족한다면 true
(True, False)
```

```
np.all(a>5) , np.all(a < 10)  all □□ 모두가 조건에 만족한다면 true
(False, True)
```


Comparison operation #1

- Numpy는 배열의 크기가 동일 할 때
element간 비교의 결과를 Boolean type으로 반환하여 돌려줌

```
test_a = np.array([1, 3, 0], float)
test_b = np.array([5, 2, 1], float)
test_a > test_b
```

```
array([False,  True, False], dtype=bool)
```

```
test_a == test_b
```

```
array([False, False, False], dtype=bool)
```

```
(test_a > test_b).any()
```

True

any □□ 하나라도 true라면 true

Comparison operation #2

```
a = np.array([1, 3, 0], float)
np.logical_and(a > 0, a < 3) # and 조건의 condition

array([ True, False, False], dtype=bool)
```

```
b = np.array([True, False, True], bool)
np.logical_not(b) # NOT 조건의 condition

array([False,  True, False], dtype=bool)
```

```
c = np.array([False, True, False], bool)
np.logical_or(b, c) # OR 조건의 condition

array([ True,  True,  True], dtype=bool)
```

np.where

```
np.where(a > 0, 3, 2) # where(condition, TRUE, FALSE)
```

```
array([3, 3, 2])
```

```
a = np.arange(10)  
np.where(a>5)
```

Index 값 반환

```
(array([6, 7, 8, 9]),)
```

```
a = np.array([1, np.NaN, np.Inf], float)  
np.isnan(a)
```

Not a Number

```
array([False,  True, False], dtype=bool)
```

```
np.isfinite(a) is finite number
```

```
array([ True, False, False], dtype=bool)
```

argmax & argmin

- array내 최대값 또는 최소값의 index를 반환함

```
a = np.array([1,2,4,5,8,78,23,3])  
np.argmax(a) , np.argmin(a)
```

(5, 0)

- axis 기반의 반환

```
a=np.array([[1,2,4,7],[9,88,6,45],[9,76,3,4]])  
np.argmax(a, axis=1) , np.argmin(a, axis=0)
```

(array([3, 1, 1]), array([0, 0, 2, 2]))

1	2	4	7
9	88	6	45
9	76	3	4

boolean & fancy
index

boolean index

- numpy는 배열은 특정 조건에 따른 값을 배열 형태로 추출 할 수 있음
- Comparison operation 함수들도 모두 사용가능

```
test_array = np.array([1, 4, 0, 2, 3, 8, 9, 7], float)
test_array > 3
```

```
array([False,  True, False, False, False,  True,  True,  True], dtype=bool)
```

```
test_array[test_array > 3]    조건이 True인 index의 element만 추출
```

```
array([ 4.,  8.,  9.,  7.])
```

```
condition = test_array < 3
test_array[condition]
```

```
array([ 1.,  0.,  2.])
```

boolean index

```
A = np.array([
[12, 13, 14, 12, 16, 14, 11, 10, 9],
[11, 14, 12, 15, 15, 16, 10, 12, 11],
[10, 12, 12, 15, 14, 16, 10, 12, 12],
[9, 11, 16, 15, 14, 16, 15, 12, 10],
[12, 11, 16, 14, 10, 12, 16, 12, 13],
[10, 15, 16, 14, 14, 14, 16, 15, 12],
[13, 17, 14, 10, 14, 11, 14, 15, 10],
[10, 16, 12, 14, 11, 12, 14, 18, 11],
[10, 19, 12, 14, 11, 12, 14, 18, 10],
[14, 22, 17, 19, 16, 17, 18, 17, 13],
[10, 16, 12, 14, 11, 12, 14, 18, 11],
[10, 16, 12, 14, 11, 12, 14, 18, 11],
[10, 19, 12, 14, 11, 12, 14, 18, 10],
[14, 22, 12, 14, 11, 12, 14, 17, 13],
[10, 16, 12, 14, 11, 12, 14, 18, 11]])
B = A < 15
B.astype(np.int)
```

```
array([[1, 1, 1, 1, 0, 1, 1, 1, 1],
       [1, 1, 1, 0, 0, 0, 1, 1, 1],
       [1, 1, 1, 0, 1, 0, 1, 1, 1],
       [1, 1, 0, 0, 1, 0, 0, 1, 1],
       [1, 1, 0, 1, 1, 1, 0, 1, 1],
       [1, 0, 0, 1, 1, 1, 0, 0, 1],
       [1, 0, 1, 1, 1, 1, 1, 0, 1],
       [1, 0, 1, 1, 1, 1, 1, 0, 1],
       [1, 0, 1, 1, 1, 1, 1, 0, 1],
       [1, 0, 0, 0, 0, 0, 0, 0, 1],
       [1, 0, 1, 1, 1, 1, 1, 0, 1],
       [1, 0, 1, 1, 1, 1, 1, 0, 1],
       [1, 0, 1, 1, 1, 1, 1, 0, 1],
       [1, 0, 1, 1, 1, 1, 1, 0, 1],
       [1, 0, 1, 1, 1, 1, 1, 0, 1]])
```


fancy index

- numpy는 array를 index value로 사용해서 값을 추출하는 방법

```
a = np.array([2, 4, 6, 8], float)
b = np.array([0, 0, 1, 3, 2, 1], int) # 반드시 integer로 선언
a[b] #bracket index, b 배열의 값을 index로 하여 a의 값을 추출함
```

```
array([ 2.,  2.,  4.,  8.,  6.,  4.])
```

0	1	2	3
---	---	---	---

```
a.take(b) #take 함수: bracket index와 같은 효과
```

2	4	6	8
---	---	---	---

```
array([ 2.,  2.,  4.,  8.,  6.,  4.])
```

fancy index

- Matrix 형태의 데이터도 가능

```
a = np.array([[1, 4], [9, 16]], float)
b = np.array([0, 0, 1, 1, 0], int)
c = np.array([0, 1, 1, 1, 1], int)
a[b,c] # b를 row index, c를 column index로 변환하여 표시함
```

array([1., 4., 16., 16., 4.])

	0	1
0	1	4
1	9	16

numpy data i/o

loadtxt & savetxt

- Text type의 데이터를 읽고, 저장하는 기능

Int type 변환

```
a = np.loadtxt("./populations.txt")  
a[:10]
```

파일 호출

```
array([[ 1900.,  30000.,  4000.,  48300.],  
       [ 1901.,  47200.,  6100.,  48200.],  
       [ 1902.,  70200.,  9800.,  41500.],  
       [ 1903.,  77400.,  35200.,  38200.],  
       [ 1904.,  36300.,  59400.,  40600.],  
       [ 1905.,  20600.,  41700.,  39800.],  
       [ 1906.,  18100.,  19000.,  38600.],  
       [ 1907.,  21400.,  13000.,  42300.],  
       [ 1908.,  22000.,   8300.,  44500.],  
       [ 1909.,  25400.,   9100.,  42100.]])
```

```
a_int = a.astype(int)  
a_int[:3]
```

```
array([[ 1900, 30000,  4000, 48300],  
       [ 1901, 47200,  6100, 48200],  
       [ 1902, 70200,  9800, 41500]])
```

```
np.savetxt('int_data.csv', a_int, delimiter=",")
```

int_data.csv 로 저장

1.9000000000000000e+03,3.0000000000000000e+04,4.0000000000000000e+03,4.8300000000000000e+04
1.9010000000000000e+03,4.7200000000000000e+04,6.1000000000000000e+03,4.8200000000000000e+04
1.9020000000000000e+03,7.0200000000000000e+04,9.8000000000000000e+03,4.1500000000000000e+04
1.9030000000000000e+03,7.7400000000000000e+04,3.5200000000000000e+04,3.8200000000000000e+04
1.9040000000000000e+03,3.6300000000000000e+04,5.9400000000000000e+04,4.0600000000000000e+04
1.9050000000000000e+03,2.0600000000000000e+04,4.1700000000000000e+04,3.9800000000000000e+04
1.9060000000000000e+03,1.8100000000000000e+04,1.9000000000000000e+04,3.8600000000000000e+04
1.9070000000000000e+03,2.1400000000000000e+04,1.3000000000000000e+04,4.2300000000000000e+04
1.9080000000000000e+03,2.2000000000000000e+04,8.3000000000000000e+03,4.4500000000000000e+04
1.9090000000000000e+03,2.5400000000000000e+04,9.1000000000000000e+03,4.2100000000000000e+04
1.9100000000000000e+03,2.7100000000000000e+04,7.4000000000000000e+03,4.6000000000000000e+04
1.9110000000000000e+03,4.0300000000000000e+04,8.0000000000000000e+03,4.6800000000000000e+04
1.9120000000000000e+03,5.7000000000000000e+04,1.2300000000000000e+04,4.3800000000000000e+04
1.9130000000000000e+03,7.6600000000000000e+04,1.9500000000000000e+04,4.0900000000000000e+04
1.9140000000000000e+03,5.2300000000000000e+04,4.5700000000000000e+04,3.9400000000000000e+04
1.9150000000000000e+03,1.9500000000000000e+04,5.1100000000000000e+04,3.9000000000000000e+04
1.9160000000000000e+03,1.1200000000000000e+04,2.9700000000000000e+04,3.6700000000000000e+04
1.9170000000000000e+03,7.6000000000000000e+03,1.5800000000000000e+04,4.1800000000000000e+04
1.9180000000000000e+03,1.4600000000000000e+04,9.7000000000000000e+03,4.3300000000000000e+04
1.9190000000000000e+03,1.6200000000000000e+04,1.0100000000000000e+04,4.1300000000000000e+04
1.9200000000000000e+03,2.4700000000000000e+04,8.6000000000000000e+03,4.7300000000000000e+04

numpy object - npy

- Numpy object (pickle) 형태로 데이터를 저장하고 불러옴
- Binary 파일 형태로 저장함

```
np.save("np_test", arr=a_int)
```

Last executed 2017-09-26 11:36:56 in 4ms

```
np_array = np.load(file="np_test.npy")  
np_array[:3]
```

Last executed 2017-09-26 11:37:07 in 5ms

```
array([[ 1900, 30000,  4000, 48300],  
       [ 1901, 47200,  6100, 48200],  
       [ 1902, 70200,  9800, 41500]])
```

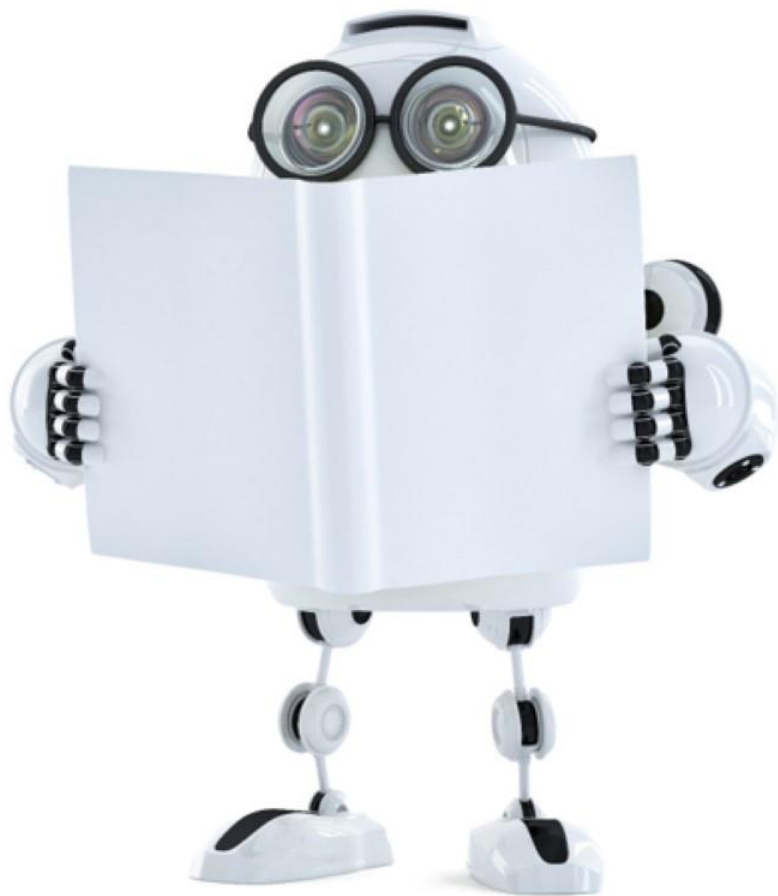
TEAMLAB

Human knowledge belongs to the world.

Pandas #1

Data handling

**Director of TEAMLAB
Sungchul Choi**



Pandas

Pandas

구조화된 데이터의 처리를 지원하는 Python 라이브러리

Python계의 엑셀!

Pandas

- 구조화된 데이터의 처리를 지원하는 Python 라이브러리
- 고성능 Array 계산 라이브러리인 Numpy와 통합하여, 강력한 “스프레드시트” 처리 기능을 제공
- 인덱싱, 연산용 함수, 전처리 함수 등을 제공함

데이터 로딩

```
In [1]: import pandas as pd #라이브러리 호출
```

```
In [2]: data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data' #Data URL  
df_data = pd.read_csv(data_url, sep='#s+', header = None) #csv 타임 데이터 로드, separate는 빈공간으로 지정하고, Column은 없음
```

```
In [3]: df_data.head() #처음 다섯줄 출력
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

Series

Pandas의 구성

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	weight_0
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	1
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	1
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	1
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	1
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	1

Series

DataFrame 중 하나의 Column에 해당
하는
데이터의 모음 Object

DataFrame

Data Table 전체를 포함하는
Object

일반적인 padas의 활용

```
In [1]: import pandas as pd #라이브러리 호출
```

```
In [2]: data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data' #Data URL  
df_data = pd.read_csv(data_url, sep='#s+', header = None) #csv 타임 데이터 로드, separate는 빈공간으로 지정하고, Column은 없음
```

```
In [3]: df_data.head() #처음 다섯줄 출력
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

기존 데이터를 불러
와서
DataFrame을 생성

**기본적인 것 부터
해보자**

Series

Column Vector를 표현하는 object

```
In [1]: from pandas import Series, DataFrame  
import pandas as pd
```

```
In [ ]: example_obj = Series()
```

Init signature: Series(data=None, index=None, dtype=None, name=None, copy=False, fastpath=False)^{^ + x}

Docstring:

One-dimensional ndarray with axis labels (including time series).

Shift + TAB

Series

```
In [2]: list_data = [1,2,3,4,5]
example_obj = Series(data = list_data)
example_obj
```

```
Out [2]: 0    1
         1    2
         2    3
         3    4
         4    5
dtype: int64
```

Index

data

Data type

Series

index		values
A	→	5
B	→	6
C	→	12
D	→	-5
E	→	6.7

- Subclass of `numpy.ndarray`
- Data: any type
- Index labels need not be ordered
- Duplicates are possible (but result in reduced functionality)

Series

```
In [3]: list_data = [1,2,3,4,5]
list_name = ["a","b","c","d","e"]
example_obj = Series(data = list_data, index=list_name)
example_obj
```

index 이름을
지정

```
Out[3]: a    1
b    2
c    3
d    4
e    5
dtype: int64
```

Series

Data와 index 이름을 지정

```
In [4]: dict_data = {"a":1, "b":2, "c":3, "d":4, "e":5}
example_obj = Series(dict_data, dtype=np.float32, name="example_data")
example_obj
```

data type
설정

series 이름
설정

```
Out[4]: a    1.0
        b    2.0
        c    3.0
        d    4.0
        e    5.0
        Name: example_data, dtype: float32
```

Series

data index에 접근

```
In [5]: example_obj["a"]
```

하기

```
Out[5]: 1.0
```

data index에 값 할당

```
In [6]: example_obj["a"] = 3.2
example_obj
```

하기

```
Out[6]: a    3.2
       b    2.0
       c    3.0
       d    4.0
       e    5.0
```

```
Name: example_data, dtype: float32
```

Series

example_obj.values

값 리스트만

```
array([ 3.20000005,  2.          ,  3.          ,  4.          ,  5.          ],  
      dtype=float32)
```

example_obj.index

Index 리스트

Index(['a', 'b', 'c', 'd', 'e'], dtype='object')

```
example_obj.name = "number"  
example_obj.index.name = "alphabet"  
example_obj
```

Data에 대한 정보를 저장

```
alphabet  
a      3.2  
b      2.0  
c      3.0  
d      4.0  
e      5.0  
Name: number, dtype: float32
```

Series

```
dict_data_1 = {"a":1, "b":2, "c":3, "d":4, "e":5}
indexes = ["a", "b", "c", "d", "e", "f", "g", "h"]
series_obj_1 = Series(dict_data_1, index=indexes)
series_obj_1
```

index 값을 기준으로 series 생성

```
a    1.0
b    2.0
c    3.0
d    4.0
e    5.0
f    NaN
g    NaN
h    NaN
dtype: float64
```

Dataframe Overview

Pandas의 구성

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	weight_0
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	1
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	1
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	1
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	1
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	1

Series

DataFrame 중 하나의 Column에 해당
하는
데이터의 모음 Object

DataFrame

Data Table 전체를 포함하는
Object

Dataframe

columns		foo	bar	baz	qux
index					
A	→	0	x	2.7	True
B	→	4	y	6	True
C	→	8	z	10	False
D	→	-12	w	NA	False
E	→	16	a	18	False

- NumPy array-like
- Each column can have a different type
- Row and column index
- Size mutable: insert and delete columns

DataFrame

Series를 모아서 만든 Data Table = 기본 2차원

```
In [4]: from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

```
In [ ]: DataFrame()
```

Shift + TAB

```
In [3]: Init signature: DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)
Docstring:
Two-dimensional size-mutable, potentially heterogeneous tabular data
structure with labeled axes (rows and columns). Arithmetic operations
```

DataFrame

```
In [1]: from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

column_name : data

```
In [2]: # Example from - https://chrisalbon.com/python/pandas\_map\_values\_to\_values.html
raw_data = {'first_name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
            'last_name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze'],
            'age': [42, 52, 36, 24, 73],
            'city': ['San Francisco', 'Baltimore', 'Miami', 'Douglas', 'Boston']}
df = pd.DataFrame(raw_data, columns = ['first_name', 'last_name', 'age', 'city'])
df
```

Out[2]:

	first_name	last_name	age	city
0	Jason	Miller	42	San Francisco
1	Molly	Jacobson	52	Baltimore
2	Tina	Ali	36	Miami
3	Jake	Milner	24	Douglas
4	Amy	Cooze	73	Boston

DataFrame

```
In [3]: DataFrame(raw_data, columns = ["age", "city"])
```

Out [3]:

	age	city
0	42	San Francisco
1	52	Baltimore
2	36	Miami
3	24	Douglas
4	73	Boston

column 선택

```
In [4]: DataFrame(raw_data, columns = ["first_name", "last_name", "age", "city", "debt"])
```

Out [4]:

	first_name	last_name	age	city	debt
0	Jason	Miller	42	San Francisco	NaN
1	Molly	Jacobson	52	Baltimore	NaN
2	Tina	Ali	36	Miami	NaN
3	Jake	Milner	24	Douglas	NaN
4	Amy	Gooze	73	Boston	NaN

새로운 column 추가

DataFrame

```
df = DataFrame(raw_data, columns = ["first_name", "last_name", "age", "city", "debt"])  
df.first_name
```

```
0    Jason  
1    Molly  
2     Tina  
3     Jake  
4     Amy  
Name: first_name, dtype: object
```

column 선택 – series 추출

```
df["first_name"]
```

```
0    Jason  
1    Molly  
2     Tina  
3     Jake  
4     Amy  
Name: first_name, dtype: object
```

column 선택 – series 추출

DataFrame

```
df
```

	first_name	last_name	age	city	debt
0	Jason	Miller	42	San Francisco	NaN
1	Molly	Jacobson	52	Baltimore	NaN
2	Tina	Ali	36	Miami	NaN
3	Jake	Milner	24	Douglas	NaN
4	Amy	Gooze	73	Boston	NaN

```
df.loc[1]
```

loc – index location

```
first_name    Molly
last_name     Jacobson
age           52
city          Baltimore
debt          NaN
Name: 1, dtype: object
```

```
df["age"].iloc[1:]
```

iloc – index position

```
1    52
2    36
3    24
4    73
Name: age, dtype: int64
```

DataFrame

loc은 index 이름, iloc은 index number

```
# Example from - https://stackoverflow.com/questions/31593201/pandas-iloc-vs-ix-vs-loc-explanation  
s = pd.Series(np.nan, index=[49,48,47,46,45, 1, 2, 3, 4, 5])  
s.loc[:3]
```

```
49    NaN  
48    NaN  
47    NaN  
46    NaN  
45    NaN  
1     NaN  
2     NaN  
3     NaN  
dtype: float64
```

```
s.iloc[:3]
```

```
49    NaN  
48    NaN  
47    NaN  
dtype: float64
```

DataFrame

Column에 새로운 데이터 할당

```
df.debt = df.age > 40  
df
```

	first_name	last_name	age	city	debt
0	Jason	Miller	42	San Francisco	True
1	Molly	Jacobson	52	Baltimore	True
2	Tina	Ali	36	Miami	False
3	Jake	Milner	24	Douglas	False
4	Amy	Gooze	73	Boston	True

```
df.T
```


DataFrame

```
In [13]: df.T
```

```
Out [13]:
```

	0	1	2	3	4
first_name	Jason	Molly	Tina	Jake	Amy
last_name	Miller	Jacobson	Ali	Milner	Cooze
age	42	52	36	24	73
city	San Francisco	Baltimore	Miami	Douglas	Boston
debt	True	True	False	False	True

transpose

```
In [14]: df.values
```

```
Out [14]: array([[ 'Jason', 'Miller', 42, 'San Francisco', True],  
                [ 'Molly', 'Jacobson', 52, 'Baltimore', True],  
                [ 'Tina', 'Ali', 36, 'Miami', False],  
                [ 'Jake', 'Milner', 24, 'Douglas', False],  
                [ 'Amy', 'Cooze', 73, 'Boston', True]], dtype=object)
```

값 출력

```
In [20]: df.to_csv()
```

```
Out [20]: ',first_name,last_name,age,city,debt\n0,Jason,Miller,42, San Francisco,True\n1,Molly,Jacobson,52,Baltimore,True\n2,Tina,Ali,36,Miami,False\n3,Jake,Milner,24,Douglas,False\n4,Amy,Cooze,73,Boston,True\n'
```

csv 변환

DataFrame

Column을 삭제함

```
del df["debt"]
```

```
df
```

	first_name	last_name	age	city
0	Jason	Miller	42	San Francisco
1	Molly	Jacobson	52	Baltimore
2	Tina	Ali	36	Miami
3	Jake	Milner	24	Douglas
4	Amy	Cooze	73	Boston

DataFrame

```
# Example from Python for data analysis
```

```
|
```

Nested dict에서는

```
pop = {'Nevada': {2001: 2.4, 2002: 2.9},  
       'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
```

Column 값

Index 값

```
DataFrame(pop)
```

	Nevada	Ohio
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

Selection & Drop

Selection with column names

```
df["account"].head(3)
```

한개의 column 선택시

```
0    211829
```

```
1    320563
```

```
2    648336
```

```
Name: account, dtype: int64
```

1개 이상의 column 선택

```
df[["account", "street", "state"]].head(3)
```

	account	street	state
0	211829	34456 Sean Highway	Texas
1	320563	1311 Alvis Tunnel	NorthCarolina
2	648336	62184 Schamberger Underpass Apt. 231	Iowa

Selection with index number

`df[:3]` column 이름 없이 사용하는 index number는 row 기준 표시

	account	name	street	city	state	postal-code	Jan	Feb	Mar
0	211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000
1	320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365	95000	45000	35000
2	648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	91000	120000	35000

`df["account"][:3]` column 이름과 함께 row index 사용시, 해당 column만

```
0    211829
1    320563
2    648336
Name: account, dtype: int64
```

Series selection

```
account_series = df["account"]  
account_series[:3]
```

```
0    211829  
1    320563  
2    648336
```

Name: account, dtype: int64

```
account_series[[0,1,2]]
```

```
0    211829  
1    320563  
2    648336
```

1개 이상의
index

Name: account, dtype: int64

```
account_series[account_series<250000]
```

```
0    211829  
3    109996  
4    121213  
5    132971  
6    145068  
7    205217  
8    209744  
9    212303  
10   214098  
11   231907  
12   242368
```

Boolean index

Name: account, dtype: int64

Index 변경

```
df.index = df["account"]  
del df["account"]  
df.head()
```

	name	street	city	state	postal-code	Jan	Feb	Mar
account								
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000
320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365	95000	45000	35000
648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	91000	120000	35000
109996	D'Amore, Gleichner and Bode	155 Fadel Crescent Apt. 144	Hyattburgh	Maine	46021	45000	120000	10000
121213	Bauch-Goldner	7274 Marissa Common	Shanahanchester	California	49681	162000	120000	35000

Basic, loc, iloc selection

Column 과
index number

```
df[["name", "street"]][:2]
```

	name	street
account		
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway
320563	Walter-Trantow	1311 Alvis Tunnel

Column number와
index number

```
df.iloc[:2,:2]
```

	name	street
account		
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway
320563	Walter-Trantow	1311 Alvis Tunnel

```
df.loc[[211829, 320563], ["name", "street"]]
```

Column 과
index name

	name	street
account		
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway
320563	Walter-Trantow	1311 Alvis Tunnel

index 재설정

```
df.index = list(range(0,15))  
df.head()
```

	name	street	city	state	postal-code
0	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752
1	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365
2	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517

Data drop

```
df.drop(1)    Index number로 drop
```

	name	street	city	s
0	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	T
2	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	lc

Data drop

`df.drop([0,1,2,3])` **한개 이상의 Index number로 drop**

	account	name	street	city	state
4	121213	Bauch-Goldner	7274 Marissa Common	Shanahanchester	California
5	132971	Williamson, Schumm and Hettinger	89403 Casimer Spring	Jeremieburgh	Arkansas
6	145068	Casper LLC	340 Consuela Bridge Apt. 400	Lake Gabriellaton	Mississippi

Data drop

axis 지정으로 축을 기준으로 drop □□ column 중에 "city"

```
df.drop("city",axis=1) # df.drop(["city", "state"],axis=1)
```

	name	street	state	postal-code
0	Kerluke, Koepp and Hilpert	34456 Sean Highway	Texas	28752
1	Walter-Trantow	1311 Alvis Tunnel	NorthCarolina	38365
2	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	Iowa	76517

Dataframe Operations

Series operation

```
s1 = Series(  
    range(1,6), index=list("abced"))
```

s1

```
a    1  
b    2  
c    3  
e    4  
d    5  
dtype: int64
```

```
s2 = Series(  
    range(5,11), index=list("bcedef"))
```

s2

```
b    5  
c    6  
e    7  
d    8  
e    9  
f   10  
dtype: int64
```

s1.add(s2)

```
a    NaN  
b    7.0  
c    9.0  
d   13.0  
e   11.0  
e   13.0  
f    NaN  
dtype: float64
```

s1 + s2

```
a    NaN  
b    7.0  
c    9.0  
d   13.0  
e   11.0  
e   13.0  
f    NaN  
dtype: float64
```

index 으로 기준으로 연산수행

겹치는 index가 없을 경우
NaN값으로 반환

Dataframe operation

```
df1 = DataFrame(  
    np.arange(9).reshape(3,3),  
    columns=list("abc"))  
df1
```

	a	b	c
0	0	1	2
1	3	4	5
2	6	7	8

```
df2 = DataFrame(  
    np.arange(16).reshape(4,4),  
    columns=list("abcd"))  
df2
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

df1 + df2

	a	b	c	d
0	0.0	2.0	4.0	NaN
1	7.0	9.0	11.0	NaN
2	14.0	16.0	18.0	NaN
3	NaN	NaN	NaN	NaN

df1.add(df2, fill_value=0)

	a	b	c	d
0	0.0	2.0	4.0	3.0
1	7.0	9.0	11.0	7.0
2	14.0	16.0	18.0	11.0
3	12.0	13.0	14.0	15.0

df는 column과 index를 모두 고려

add operation을 쓰면 NaN값 0으로 변환
Operation types: add, sub, div, mul

Series + Dataframe

```
df = DataFrame(  
    np.arange(16).reshape(4,4),  
    columns=list("abcd"))  
df
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15



10 11 12 13
10 11 12 13
10 11 12 13
10 11 12 13

```
s = Series(  
    np.arange(10,14),  
    index=list("abcd"))  
s
```

a 10
b 11
c 12
d 13
dtype: int64



df + s

	a	b	c	d
0	10	12	14	16
1	14	16	18	20
2	18	20	22	24
3	22	24	26	28

column을 기준으로
broadcasting이 발생함

Series + Dataframe

```
df = DataFrame(  
    np.arange(16).reshape(4,4),  
    columns=list("abcd"))  
df
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

```
s2 = Series(np.arange(10,14))  
s2
```

```
0    10  
1    11  
2    12  
3    13  
dtype: int64
```

```
df + s2
```

	a	b	c	d	0	1	2	3
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
df.add(s2, axis=0)
```

	a	b	c	d
0	10	11	12	13
1	15	16	17	18
2	20	21	22	23
3	25	26	27	28

**axis를 기준으로
row broadcasting 실행**

lambda, map, apply

Lambda 함수

- 한 줄로 함수를 표현하는 익명 함수 기법
- Lisp 언어에서 시작된 기법으로 오늘날 현대언어에 많이 사용

lambda argument : expression

```
lambda x,y: x + y
```

```
def f(x , y):  
    return x + y
```

```
f = lambda x,y: x + y  
f(1,4)
```

5

Lambda 함수

```
f = lambda x: x / 2  
f(3)
```

1.5

하나의 argument만 처리하는
lambda 함수

```
f = lambda x: x ** 2  
f(3)
```

9

```
(lambda x: x + 1)(5)
```

6

이름을 할당하지 않는 lambda 함수

map 함수

- 함수와 sequence형 데이터를 인자로 받아
- 각 element마다 입력받은 함수를 적용하여 list로 반환
- 일반적으로 함수를 lambda형태로 표현함

`map(function, sequence)`

```
ex = [1,2,3,4,5]  
f = lambda x: x ** 2  
list(map(f, ex))
```

`[1, 4, 9, 16, 25]`

map 함수

```
f = lambda x, y: x + y  
list(map(f, ex, ex))
```

[2, 4, 6, 8, 10]

두 개이상의 argument가 있을 때는
두 개의 sequence형을 써야함

익명 함수 그대로 사용할 수 있음

```
list(map(lambda x: x+x, ex))  
#python 3에서는 list를 꼭 붙여줘야함
```

[2, 4, 6, 8, 10]

map for series

- Pandas의 series type의 데이터에도 map 함수 사용가능
- function 대신 dict, sequence형 자료등으로 대체 가능

```
s1 = Series(np.arange(10))  
s1.head(5)
```

```
0    0  
1    1  
2    2  
3    3  
4    4  
dtype: int64
```

```
s1.map(lambda x: x**2).head(5)
```

```
0    0  
1    1  
2    4  
3    9  
4   16  
dtype: int64
```


map for series

```
z = {1: 'A', 2: 'B', 3: 'C'}  
s1.map(z).head(5)
```

0	NaN
1	A
2	B
3	C
4	NaN

dtype: object

dict type으로
데이터 교체

없는 값은 NaN

```
s2 = Series(np.arange(10,20))  
s1.map(s2).head(5)
```

0	10
1	11
2	12
3	13
4	14

dtype: int64

같은 위치의 데이터를
s2로 전환

Example - map for series

```
df = pd.read_csv("wages.csv")  
df.head()
```

	earn	height	sex	race	ed	age
0	79571.299011	73.89	male	white	16	49
1	96396.988643	66.23	female	white	16	62
2	48710.666947	63.77	female	white	16	33
3	80478.096153	63.22	female	other	16	95
4	82089.345498	63.08	female	white	17	43

Example - map for series

```
df.sex.unique()
```

```
array(['male', 'female'], dtype=object)
```

```
df["sex_code"] = df.sex.map({"male":0, "female":1})  
df.head(5)
```

성별 str → 성별 code

	earn	height	sex	race	ed	age	sex_code
0	79571.299011	73.89	male	white	16	49	0
1	96396.988643	66.23	female	white	16	62	1
2	48710.666947	63.77	female	white	16	33	1
3	80478.096153	63.22	female	other	16	95	1
4	82089.345498	63.08	female	white	17	43	1

Replace function

- Map 함수의 기능중 데이터 변환 기능만 담당
- 데이터 변환시 많이 사용하는 함수

```
df.sex.replace(  
    {"male":0, "female":1}  
).head()
```

```
0    0  
1    1  
2    1  
3    1  
4    1
```

Name: sex, dtype: int64

dict type 적용

```
df.sex.replace(  
    ["male", "female"],  
    [0,1], inplace=True)  
df.head(5)
```

	earn	height	sex	race	ed	age	sex_code
0	79571.299011	73.89	0	white	16	49	0
1	96396.988643	66.23	1	white	16	62	1
2	48710.666947	63.77	1	white	16	33	1

inplace □□ 데이터 변환결과를 적용

Target list

Conversion list

apply for dataframe

- map과 달리, series 전체(column)에 해당 함수를 적용
- 입력값이 series 데이터로 입력받아 handling 가능

```
df_info = df[["earn", "height", "age"]]  
df_info.head()
```

	earn	height	age
0	79571.299011	73.89	49
1	96396.988643	66.23	62
2	48710.666947	63.77	33
3	80478.096153	63.22	95
4	82089.345498	63.08	43

```
f = lambda x : x.max() - x.min()  
df_info.apply(f)
```

```
earn      318047.708444  
height    19.870000  
age       73.000000  
dtype: float64
```

각 column 별로 결과값 반환

apply for dataframe

- 내장 연산 함수를 사용할 때도 똑같은 효과를 거둘 수 있음
- mean, std 등 사용가능

```
df_info.sum()
```

```
earn      4.474344e+07  
height    9.183125e+04  
age       6.250800e+04  
dtype: float64
```

```
df_info.apply(sum)
```

```
earn      4.474344e+07  
height    9.183125e+04  
age       6.250800e+04  
dtype: float64
```

apply for dataframe

- scalar 값 이외에 series값의 반환도 가능함

```
def f(x):  
    return Series([x.min(), x.max()], index=["min", "max"])  
df_info.apply(f)
```

	earn	height	age
min	-98.580489	57.34	22
max	317949.127955	77.21	95

applymap for dataframe

- series 단위가 아닌 element 단위로 함수를 적용함
- series 단위에 apply를 적용시킬 때와 같은 효과

```
f = lambda x : -x  
df_info.applymap(f).head(5)
```

	earn	height	age
0	-79571.299011	-73.89	-49
1	-96396.988643	-66.23	-62
2	-48710.666947	-63.77	-33
3	-80478.096153	-63.22	-95
4	-82089.345498	-63.08	-43

```
f = lambda x : -x  
df_info["earn"].apply(f).head(5)
```

```
0    -79571.299011  
1    -96396.988643  
2    -48710.666947  
3    -80478.096153  
4    -82089.345498  
Name: earn, dtype: float64
```


Pandas Built-in functions

describe

- Numeric type 데이터의 요약 정보를 보여줌

```
df = pd.read_csv("wages.csv")  
df.head()
```

	earn	height	sex	race	ed	age
0	79571.299011	73.89	male	white	16	49
1	96396.988643	66.23	female	white	16	62
2	48710.666947	63.77	female	white	16	33
3	80478.096153	63.22	female	other	16	95
4	82089.345498	63.08	female	white	17	43

```
df.describe()
```

	earn	height	ed	age
count	1379.000000	1379.000000	1379.000000	1379.000000
mean	32446.292622	66.592640	13.354605	45.328499
std	31257.070006	3.818108	2.438741	15.789715
min	-98.580489	57.340000	3.000000	22.000000
25%	10538.790721	63.720000	12.000000	33.000000
50%	26877.870178	66.050000	13.000000	42.000000
75%	44506.215336	69.315000	15.000000	55.000000
max	317949.127955	77.210000	18.000000	95.000000

unique

- series data의 유일한 값을 list를 반환함

```
df.race.unique()
```

유일한 인종의 값 list

```
array(['white', 'other', 'hispanic', 'black'], dtype=object)
```

```
np.array(dict(enumerate(df["race"].unique())))
```

dict type으로 index

```
array({0: 'white', 1: 'other', 2: 'hispanic', 3: 'black'}, dtype=object)
```

```
value = list(map(int, np.array(list(enumerate(df["race"].unique()))[:, 0].tolist()))  
key = np.array(list(enumerate(df["race"].unique())), dtype=str)[:, 1].tolist()
```

```
value, key
```

label index 값과 label 값 각각 추출

```
([0, 1, 2, 3], ['white', 'other', 'hispanic', 'black'])
```

unique

label str → index 값으로 변환

```
df["race"].replace(to_replace=key, value=value, inplace=True)
```

```
value = list(map(int, np.array(list(enumerate(df["sex"].unique()))[:, 0].tolist())))  
key = np.array(list(enumerate(df["sex"].unique())), dtype=str)[:, 1].tolist()
```

value, key

성별에 대해서도 동일하게 적용

```
([0, 1], ['male', 'female'])
```

```
df["sex"].replace(to_replace=key, value=value, inplace=True)  
df.head(5)
```

	earn	height	sex	race	ed	age
0	79571.299011	73.89	0	0	16	49
1	96396.988643	66.23	1	0	16	62

“sex”와 “race” column의
index labelling

sum

- 기본적인 column 또는 row 값의 연산을 지원
- sub, mean, min, max, count, median, mad, var 등

`df.sum(axis=0)` | column 별

earn	4.474344e+07
height	9.183125e+04
sex	8.590000e+02
race	5.610000e+02
ed	1.841600e+04
age	6.250800e+04
dtype:	float64

`df.sum(axis=1)` row 별

0	79710.189011
1	96542.218643
2	48824.436947
3	80654.316153
4	82213.425498
5	15423.882901
6	47231.711821

isnull

- column 또는 row 값의 NaN (null) 값의 index를 반환함

```
df.isnull()
```

	earn	height	sex	race	ed	age
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False

```
df.isnull().sum()
```

```
earn      0
height    0
sex        0
race       0
ed         0
age        0
dtype: int64
```

Null인 값의 합

sort_values

- column 값을 기준으로 데이터를 sorting

```
df.sort_values(["age", "earn"], ascending=True).head(10)
```

	earn	height	sex	race	ed	age
1038	-56.321979	67.81	0	2	10	22
800	-27.876819	72.29	0	0	12	22
963	-25.655260	68.90	0	0	12	22
1105	988.565070	64.71	1	0	12	22
801	1000.221504	64.09	1	0	12	22

ascending □□ 오름차순

Correlation & Covariance

- 상관계수와 공분산을 구하는 함수
- `corr`, `cov`, `corrwith`

```
: df.age.corr(df.earn)
```

```
: 0.074003491778360575
```

```
: df.age.cov(df.earn)
```

```
: 36523.6992104089
```

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

```
df.corrwith(df.earn)
```

```
earn      1.000000  
height    0.291600  
sex       -0.337328  
race      -0.063977  
ed         0.350374  
age        0.074003  
dtype: float64
```


Correlation & Covariance

```
df.corr()
```

	earn	height	sex	race	ed	age
earn	1.000000	0.291600	-0.337328	-0.063977	0.350374	0.074003
height	0.291600	1.000000	-0.703672	-0.045974	0.114047	-0.133727
sex	-0.337328	-0.703672	1.000000	0.000858	-0.061747	0.070036
race	-0.063977	-0.045974	0.000858	1.000000	-0.049487	-0.056879
ed	0.350374	0.114047	-0.061747	-0.049487	1.000000	-0.129802
age	0.074003	-0.133727	0.070036	-0.056879	-0.129802	1.000000

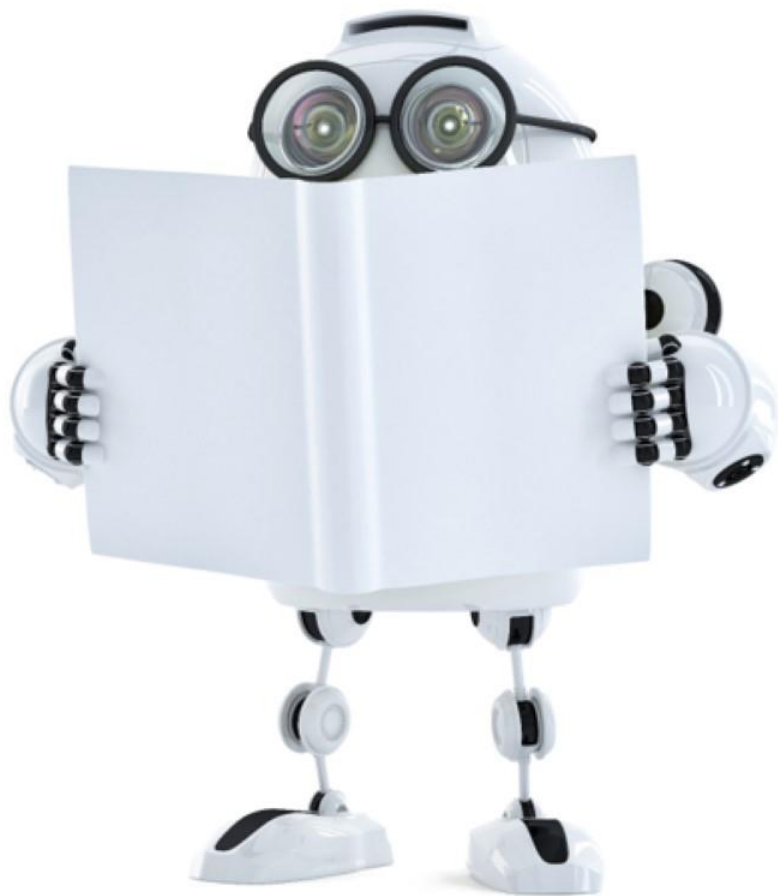
TEAMLAB

Human knowledge belongs to the world.

Pandas #2

Data handling

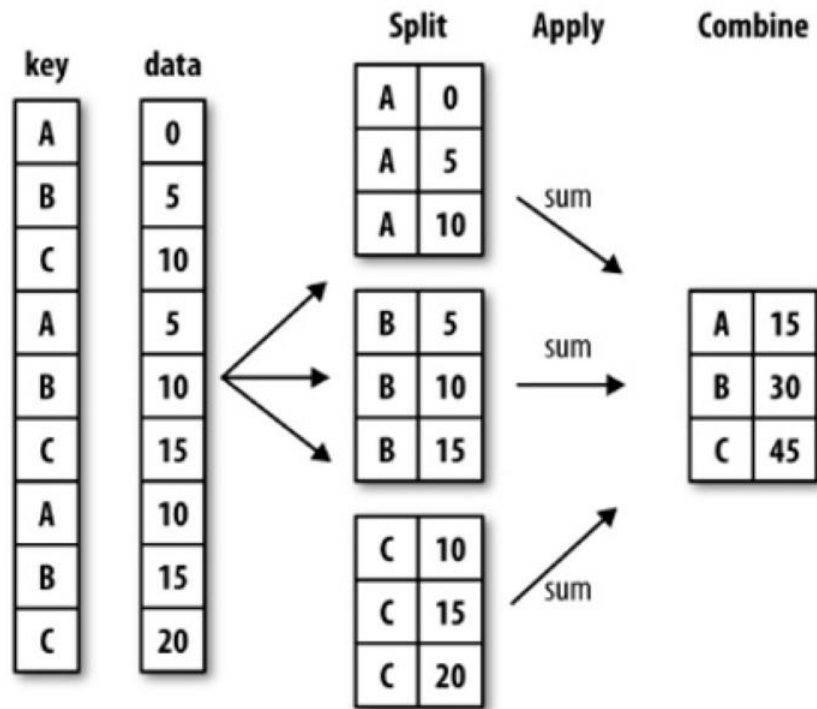
**Director of TEAMLAB
Sungchul Choi**



Groupby I

Groupby

- SQL groupby 명령어와 같음
- split → apply → combine
- 과정을 거쳐 연산함



Groupby

```
df.groupby("Team")["Points"].sum()
```

적용받는 연산

묶음의 기준이 되는 컬럼

적용받는 컬럼

	Points	Rank	Team	Year
0	876	1	Riders	2014
1	789	2	Riders	2015
2	863	2	Devils	2014
3	673	3	Devils	2015
4	741	3	Kings	2014

Team

Devils 1536

Kings 2285

Riders 3049

Royals 1505

kings 812

Name: Points, dtype: int64

결과

TEAM을 기준으로
Points을 Sum

Groupby

- 한 개이상의 column을 묶을 수 있음

```
df.groupby([ "Team", "Year" ])[ "Points" ].sum()
```

Team	Year	
Devils	2014	863
	2015	673
Kings	2014	741
	2016	756
	2017	788
Riders	2014	876
	2015	789
	2016	694

	Points	Rank	Team	Year
0	876	1	Riders	2014
1	789	2	Riders	2015
2	863	2	Devils	2014
3	673	3	Devils	2015
4	741	3	Kings	2014

Hierarchical index

- Groupby 명령의 결과물도 결국은 dataframe
- 두 개의 column으로 groupby를 할 경우, index가 두개 생성

```
h_index.index
```

```
MultiIndex(levels=[['Devils', 'Kings', 'Riders', 'Royals', 'kings'], [2014, 2015, 2016, 2017]],  
            labels=[[0, 0, 1, 1, 1, 2, 2, 2, 2, 3, 3, 4], [0, 1, 0, 2, 3, 0, 1, 2, 3, 0, 1,  
1]],  
            names=['Team', 'Year'])
```

```
h_index["Devils": "Kings"]
```

Team	Year	
Devils	2014	863
	2015	673
Kings	2014	741
	2016	756
	2017	788

Name: Points, dtype: int64

Hierarchical index – unstack()

- Group으로 묶여진 데이터를 matrix 형태로 전환해줌

Team	Year	
Devils	2014	863
	2015	673
Kings	2014	741
	2016	756
	2017	788
Riders	2014	876
	2015	789
	2016	694
	2017	690
Royals	2014	701
	2015	804
kings	2015	812



```
h_index.unstack()
```

Year	2014	2015	2016	2017
Team				
Devils	863.0	673.0	NaN	NaN
Kings	741.0	NaN	756.0	788.0
Riders	876.0	789.0	694.0	690.0
Royals	701.0	804.0	NaN	NaN
kings	NaN	812.0	NaN	NaN

Hierarchical index – swaplevel

- Index level을 변경할 수 있음

```
h_index.swaplevel()
```

Year	Team	
2014	Devils	863
2015	Devils	673
2014	Kings	741
2016	Kings	756
2017	Kings	788
2014	Riders	876
2015	Riders	789
2016	Riders	694
2017	Riders	690
2014	Royals	701
2015	Royals	804
	kings	812

Name: Points, dtype: int64

```
h_index.swaplevel().sortlevel(0)
```

Year	Team	
2014	Devils	863
	Kings	741
	Riders	876
	Royals	701
2015	Devils	673
	Riders	789
	Royals	804
	kings	812
2016	Kings	756
	Riders	694
2017	Kings	788
	Riders	690

Name: Points, dtype: int64

Hierarchical index – operations

- Index level을 기준으로 기본 연산 수행 가능

```
h_index.sum(level=0)
```

Team	
Devils	1536
Kings	2285
Riders	3049
Royals	1505
kings	812

Name: Points, dtype: int64

```
h_index.sum(level=1)
```

Year	
2014	3181
2015	3078
2016	1450
2017	1478

Name: Points, dtype: int64

Groupby II

Groupby – gropued

- Groupby에 의해 Split된 상태를 추출 가능함

```
grouped = df.groupby("Team")  
  
for name, group in grouped:  
    print (name)  
    print (group)
```

Tuple 형태로 그룹의 key 값
Value값이 추출됨

Devils				
	Points	Rank	Team	Year
2	863	2	Devils	2014
3	673	3	Devils	2015
Kings				
	Points	Rank	Team	Year
4	741	3	Kings	2014
6	756	1	Kings	2016
7	788	1	Kings	2017

Groupby – gropued

- 특정 key값을 가진 그룹의 정보만 추출 가능

```
grouped.get_group("Devils")
```

	Points	Rank	Team	Year
2	863	2	Devils	2014
3	673	3	Devils	2015

Groupby – gropued

- 추출된 group 정보에는 세 가지 유형의 apply가 가능함
- Aggregation: 요약된 통계정보를 추출해 줌
- Transformation: 해당 정보를 변환해줌
- Filtration: 특정 정보를 제거 하여 보여주는 필터링 기능

Groupby – aggregation

```
grouped.agg(sum)
```

	Points	Rank	Year
Team			
Devils	1536	5	4029
Kings	2285	5	6047
Riders	3049	7	8062
Royals	1505	5	4029
kings	812	4	2015

```
import numpy as np  
grouped.agg(np.mean)
```

	Points	Rank	Year
Team			
Devils	768.000000	2.500000	2014.500000
Kings	761.666667	1.666667	2015.666667
Riders	762.250000	1.750000	2015.500000
Royals	752.500000	2.500000	2014.500000
kings	812.000000	4.000000	2015.000000

Groupby – aggregation

```
grouped[ 'Points' ].agg([np.sum, np.mean, np.std])
```

	sum	mean	std
Team			
Devils	1536	768.000000	134.350288
Kings	2285	761.666667	24.006943
Riders	3049	762.250000	88.567771
Royals	1505	752.500000	72.831998
kings	812	812.000000	NaN

특정 컬럼에
여러개의 function을
Apply 할 수 도 있음

Groupby – transformation

- Aggregation과 달리 key값 별로 요약된 정보가 아님
- 개별 데이터의 변환을 지원함

df

	Points	Rank	Team	Year
0	876	1	Riders	2014
1	789	2	Riders	2015
2	863	2	Devils	2014
3	673	3	Devils	2015
4	741	3	Kings	2014
5	812	4	kings	2015
6	756	1	Kings	2016
7	788	1	Kings	2017
8	694	2	Riders	2016
9	701	4	Royals	2014

```
score = lambda x: (x)
grouped.transform(score)
```

	Points	Rank	Year
0	876	1	2014
1	789	2	2015
2	863	2	2014
3	673	3	2015
4	741	3	2014
5	812	4	2015
6	756	1	2016
7	788	1	2017
8	694	2	2016
9	701	4	2014

df

	Points	Rank	Team	Year
0	876	1	Riders	2014
1	789	2	Riders	2015
2	863	2	Devils	2014
3	673	3	Devils	2015
4	741	3	Kings	2014
5	812	4	kings	2015
6	756	1	Kings	2016
7	788	1	Kings	2017
8	694	2	Riders	2016
9	701	4	Royals	2014

```
score = lambda x: (x.max())  
grouped.transform(score)
```

	Points	Rank	Year
0	876	2	2017
1	876	2	2017
2	863	3	2015
3	863	3	2015
4	788	3	2017
5	812	4	2015
6	788	3	2017
7	788	3	2017
8	876	2	2017
9	804	4	2015

단 max나 min 처럼
Series 데이터에 적용되
는 데이터 들은
Key값을 기준으로
Grouped된 데이터 기준

df

	Points	Rank	Team	Year
0	876	1	Riders	2014
1	789	2	Riders	2015
2	863	2	Devils	2014
3	673	3	Devils	2015
4	741	3	Kings	2014
5	812	4	kings	2015
6	756	1	Kings	2016
7	788	1	Kings	2017
8	694	2	Riders	2016
9	701	4	Royals	2014

```
score = lambda x: (x - x.mean()) / x.std()
grouped.transform(score)
```

	Points	Rank	Year
0	1.284327	-1.500000	-1.161895
1	0.302029	0.500000	-0.387298
2	0.707107	-0.707107	-0.707107
3	-0.707107	0.707107	0.707107
4	-0.860862	1.154701	-1.091089
5	NaN	NaN	NaN
6	-0.236043	-0.577350	0.218218
7	1.096905	-0.577350	0.872872
8	-0.770596	0.500000	0.387298
9	-0.707107	0.707107	-0.707107

$$z_i = \frac{x_i - \mu}{\sigma}$$

Groupby – filter

- 특정 조건으로 데이터를 검색할 때 사용

```
df.groupby('Team').filter(lambda x: len(x) >= 3)
```

	Points	Rank	Team	Year
0	876	1	Riders	2014
1	789	2	Riders	2015
4	741	3	Kings	2014
6	756	1	Kings	2016
7	788	1	Kings	2017
8	694	2	Riders	2016
11	690	2	Riders	2017

- filter안에는 boolean 조건이 존재해야함
- len(x)는 grouped된 dataframe 개수

```
df.groupby('Team').filter(lambda x: x["Rank"].sum() > 2)
```

```
df.groupby('Team').filter(lambda x: x["Points"].sum() > 1000)
```

```
df.groupby('Team').filter(lambda x: x["Rank"].mean() > 1)
```

Case study

Data

- 시간과 데이터 종류가 정리된 통화량 데이터

```
import dateutil
```

```
df_phone = pd.read_csv("phone_data.csv")  
df_phone['date'] = df_phone['date'].apply(dateutil.parser.parse, dayfirst=True)  
df_phone.head()
```

	index	date	duration	item	month	network	network_type
0	0	2014-10-15 06:58:00	34.429	data	2014-11	data	data
1	1	2014-10-15 06:58:00	13.000	call	2014-11	Vodafone	mobile
2	2	2014-10-15 14:46:00	23.000	call	2014-11	Meteor	mobile
3	3	2014-10-15 14:48:00	4.000	call	2014-11	Tesco	mobile
4	4	2014-10-15 17:27:00	4.000	call	2014-11	Tesco	mobile

https://www.shanelynn.ie/wp-content/uploads/2015/06/phone_data.csv


```
df_phone.groupby('month')['duration'].sum()
```

```
month
2014-11    26639.441
2014-12    14641.870
2015-01    18223.299
2015-02    15522.299
2015-03    22750.441
Name: duration, dtype: float64
```

```
df_phone[df_phone['item'] == 'call'].groupby('network')['duration'].sum()
```

```
network
Meteor      7200.0
Tesco       13828.0
Three       36464.0
Vodafone    14621.0
landline    18433.0
voicemail   1775.0
Name: duration, dtype: float64
```

```
df_phone.groupby(['month', 'item'])['date'].count()
```

month	item	
2014-11	call	107
	data	29
	sms	94
2014-12	call	79
	data	30
	sms	48
2015-01	call	88
	data	31
	sms	86
2015-02	call	67
	data	31
	sms	39
2015-03	call	47
	data	29
	sms	25

Name: date, dtype: int64

```
df_phone.groupby(['month', 'item'])['date'].count().unstack()
```

item	call	data	sms
month			
2014-11	107	29	94
2014-12	79	30	48
2015-01	88	31	86
2015-02	67	31	39
2015-03	47	29	25

```
df_phone.groupby('month', as_index=False).agg({"duration": "sum"})
```

	month	duration
0	2014-11	26639.441
1	2014-12	14641.870
2	2015-01	18223.299
3	2015-02	15522.299
4	2015-03	22750.441

```
df_phone.groupby(['month', 'item']).agg({'duration':sum,
                                         'network_type': "count",
                                         'date': 'first'})
```

		network_type	date	duration
month	item			
2014-11	call	107	2014-10-15 06:58:00	25547.000
	data	29	2014-10-15 06:58:00	998.441
	sms	94	2014-10-16 22:18:00	94.000
2014-12	call	79	2014-11-14 17:24:00	13561.000
	data	30	2014-11-13 06:58:00	1032.870
	sms	48	2014-11-14 17:28:00	48.000
2015-01	call	88	2014-12-15 20:03:00	17070.000
	data	31	2014-12-13 06:58:00	1067.299
	sms	22	2014-12-15 12:52:00	22.000

```
df_phone.groupby(['month', 'item']).agg({'duration': [min, max, sum],          # find the min
                                         'network_type': "count", # find the number of network types
                                         'date': [min, 'first', 'nunique']}) # get the min date
```

		network_type	date			duration		
		count	min	first	nunique	min	max	sum
month	item							
2014-11	call	107	2014-10-15 06:58:00	2014-10-15 06:58:00	104	1.000	1940.000	25547.000
	data	29	2014-10-15 06:58:00	2014-10-15 06:58:00	29	34.429	34.429	998.441
	sms	94	2014-10-16 22:18:00	2014-10-16 22:18:00	79	1.000	1.000	94.000
2014-12	call	79	2014-11-14 17:24:00	2014-11-14 17:24:00	76	2.000	2120.000	13561.000
	data	30	2014-11-13 06:58:00	2014-11-13 06:58:00	30	34.429	34.429	1032.870
	sms	48	2014-11-14 17:28:00	2014-11-14 17:28:00	41	1.000	1.000	48.000

```
grouped = df_phone.groupby('month').agg( {"duration" : [min, max, np.mean]})

grouped.columns = grouped.columns.droplevel(level=0)
grouped.rename(columns={"min": "min_duration", "max": "max_duration", "mean": "mean_duration"})
```

	min_duration	max_duration	mean_duration
month			
2014-11	1.0	1940.0	115.823657
2014-12	1.0	2120.0	93.260318
2015-01	1.0	1859.0	88.894141
2015-02	1.0	1863.0	113.301453
2015-03	1.0	10528.0	225.251891

Pivot table

Crosstab

Pivot Table

- 우리가 Excel에서 보던 그 것!
- Index 축은 groupby와 동일함
- Column에 추가로 labelling 값을 추가하여,
- Value에 numeric type 값을 aggregation 하는 형태

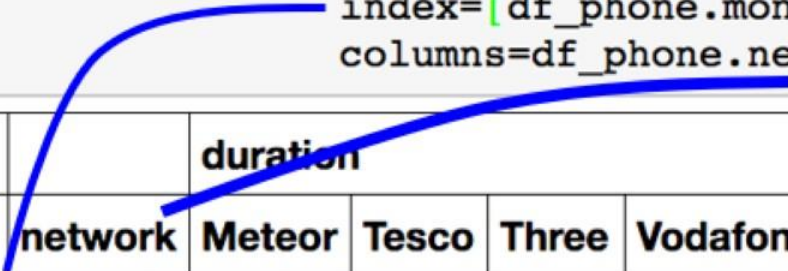
Pivot Table

```
df_phone = pd.read_csv("phone_data.csv")
df_phone['date'] = df_phone['date'].apply(dateutil.parser.parse, dayfirst=True)
df_phone.head()
```

	index	date	duration	item	month	network	network_type
0	0	2014-10-15 06:58:00	34.429	data	2014-11	data	data
1	1	2014-10-15 06:58:00	13.000	call	2014-11	Vodafone	mobile
2	2	2014-10-15 14:46:00	23.000	call	2014-11	Meteor	mobile
3	3	2014-10-15 14:48:00	4.000	call	2014-11	Tesco	mobile
4	4	2014-10-15 17:27:00	4.000	call	2014-11	Tesco	mobile

값 가로축 세로축

```
df_phone.pivot_table(["duration"],  
                      index=[df_phone.month,df_phone.item],  
                      columns=df_phone.network, aggfunc="sum", fill_value=0)
```



		duration								
	network	Meteor	Tesco	Three	Vodafone	data	landline	special	voicemail	world
month	item									
2014-11	call	1521	4045	12458	4316	0.000	2906	0	301	0
	data	0	0	0	0	998.441	0	0	0	0
	sms	10	3	25	55	0.000	0	1	0	0
2014-12	call	2010	1819	6316	1302	0.000	1424	0	690	0
	data	0	0	0	0	1032.870	0	0	0	0
	sms	12	1	13	18	0.000	0	0	0	4

Crosstab

- 특히 두 칼럼에 교차 빈도, 비율, 덧셈 등을 구할 때 사용
- Pivot table의 특수한 형태
- User-Item Rating Matrix 등을 만들 때 사용가능함

Crosstab

```
df_movie = pd.read_csv("./movie_rating.csv")  
df_movie.head()
```

	critic 세로축	title 가로축	rating 값
0	Jack Matthews	Lady in the Water	3.0
1	Jack Matthews	Snakes on a Plane	4.0
2	Jack Matthews	You Me and Dupree	3.5
3	Jack Matthews	Superman Returns	5.0
4	Jack Matthews	The Night Listener	3.0

```
pd.crosstab(index=df_movie.critic,columns=df_movie.title,values=df_movie.rating,  
            aggfunc="first").fillna(0)
```

title	Just My Luck	Lady in the Water	Snakes on a Plane	Superman Returns	The Night Listener	You Me and Dupree
critic						
Claudia Puig	3.0	0.0	3.5	4.0	4.5	2.5
Gene Seymour	1.5	3.0	3.5	5.0	3.0	3.5
Jack Matthews	0.0	3.0	4.0	5.0	3.0	3.5
Lisa Rose	3.0	2.5	3.5	3.5	3.0	2.5
Mick LaSalle	2.0	3.0	4.0	3.0	3.0	2.0
Toby	0.0	0.0	4.5	4.0	0.0	1.0

```
df_movie.pivot_table(["rating"], index=df_movie.critic, columns=df_movie.title,
                      aggfunc="sum", fill_value=0)
```

	rating					
title	Just My Luck	Lady in the Water	Snakes on a Plane	Superman Returns	The Night Listener	You Me and Dupree
critic						
Claudia Puig	3.0	0.0	3.5	4.0	4.5	2.5
Gene Seymour	1.5	3.0	3.5	5.0	3.0	3.5
Jack Matthews	0.0	3.0	4.0	5.0	3.0	3.5
Lisa Rose	3.0	2.5	3.5	3.5	3.0	2.5
Mick LaSalle	2.0	3.0	4.0	3.0	3.0	2.0
Toby	0.0	0.0	4.5	4.0	0.0	1.0

Merge & Concat

Merge

- SQL에서 많이 사용하는 Merge와 같은 기능
- 두 개의 데이터를 하나로 합침

	subject_id	test_score
0	1	51
1	2	15
2	3	15
3	4	61
4	5	16
5	7	14

	subject_id	first_name	last_name
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

Merge

subject_id 기준으로 merge

```
pd.merge(df_a, df_b, on='subject_id')
```

	subject_id	test_score
0	1	51
1	2	15
2	3	15
3	4	61
4	5	16
5	7	14

	subject_id	test_id	first_name	last_name
0	4	61	Billy	Bonder
1	5	16	Brian	Black
2	7	14	Bryce	Brice
3	8	15	Betty	Btisan

	subject_id	first_name	last_name
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

Merge

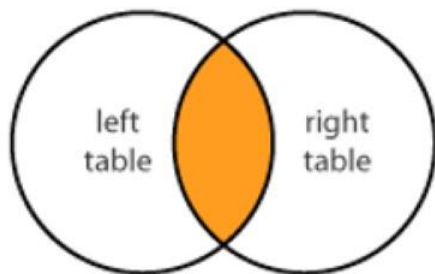
두 dataframe이 column이름이 다를 때

```
pd.merge(df_a, df_b, left_on='subject_id', right_on='subject_id')
```

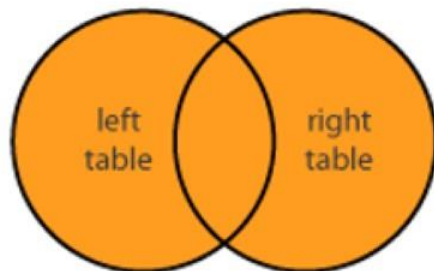
	subject_id	test_score	first_name	last_name
0	4	61	Billy	Bonder
1	5	16	Brian	Black
2	7	14	Bryce	Brice
3	8	15	Betty	Btisan

Join method

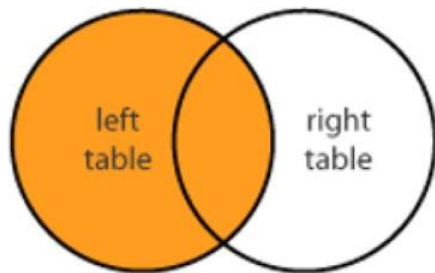
INNER JOIN



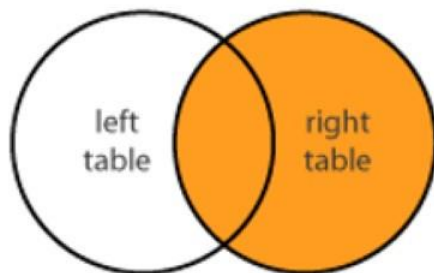
FULL JOIN



LEFT JOIN



RIGHT JOIN



Data

	subject_id	first_name	last_name
0	1	Alex	Anderson
1	2	Amy	Ackerman
2	3	Allen	Ali
3	4	Alice	Aoni
4	5	Ayoung	Atiches

	subject_id	first_name	last_name
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

Left join

```
pd.merge(df_a, df_b, on='subject_id', how='left')
```

	click to scroll output; double click to hide subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	1	Alex	Anderson	NaN	NaN
1	2	Amy	Ackerman	NaN	NaN
2	3	Allen	Ali	NaN	NaN
3	4	Alice	Aoni	Billy	Bonder
4	5	Ayoung	Atiches	Brian	Black

Right join

```
pd.merge(df_a, df_b, on='subject_id', how='right')
```

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	4	Alice	Aoni	Billy	Bonder
1	5	Ayoung	Atiches	Brian	Black
2	6	NaN	NaN	Bran	Balwner
3	7	NaN	NaN	Bryce	Brice
4	8	NaN	NaN	Betty	Btisan

Full(outer) join

```
pd.merge(df_a, df_b, on='subject_id', how='outer')
```

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	1	Alex	Anderson	NaN	NaN
1	2	Amy	Ackerman	NaN	NaN
2	3	Allen	Ali	NaN	NaN
3	4	Alice	Aoni	Billy	Bonder
4	5	Ayoung	Atiches	Brian	Black
5	6	NaN	NaN	Bran	Balwner
6	7	NaN	NaN	Bryce	Brice
7	8	NaN	NaN	Betty	Btisan

Inner join

```
pd.merge(df_a, df_b, on='subject_id', how='inner')
```

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	4	Alice	Aoni	Billy	Bonder
1	5	Ayoung	Atiches	Brian	Black

Index based join

```
pd.merge(df_a, df_b, right_index=True, left_index=True)
```

	subject_id_x	first_name_x	last_name_x	subject_id_y	first_name_y	last_name_y
0	1	Alex	Anderson	4	Billy	Bonder
1	2	Amy	Ackerman	5	Brian	Black
2	3	Allen	Ali	6	Bran	Balwner
3	4	Alice	Aoni	7	Bryce	Brice
4	5	Ayoung	Atiches	8	Betty	Btisan

Concat

- 같은 형태의 데이터를 붙이는 연산작업

df1					Result				
	A	B	C	D		A	B	C	D
0	A0	B0	C0	D0	0	A0	B0	C0	D0
1	A1	B1	C1	D1	1	A1	B1	C1	D1
2	A2	B2	C2	D2	2	A2	B2	C2	D2
3	A3	B3	C3	D3	3	A3	B3	C3	D3

df2						A	B	C	D
	A	B	C	D		A	B	C	D
4	A4	B4	C4	D4	4	A4	B4	C4	D4
5	A5	B5	C5	D5	5	A5	B5	C5	D5
6	A6	B6	C6	D6	6	A6	B6	C6	D6
7	A7	B7	C7	D7	7	A7	B7	C7	D7

df3						A	B	C	D
	A	B	C	D		A	B	C	D
8	A8	B8	C8	D8	8	A8	B8	C8	D8
9	A9	B9	C9	D9	9	A9	B9	C9	D9
10	A10	B10	C10	D10	10	A10	B10	C10	D10
11	A11	B11	C11	D11	11	A11	B11	C11	D11

df1					df4				Result							
									A	B	C	D	B	D	F	
0	A0	B0	C0	D0	2	B2	D2	F2	0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	3	B3	D3	F3	1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	6	B6	D6	F6	2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	7	B7	D7	F7	3	A3	B3	C3	D3	B3	D3	F3
									6	NaN	NaN	NaN	NaN	B6	D6	F6
									7	NaN	NaN	NaN	NaN	B7	D7	F7

Concat

```
df_new = pd.concat([df_a, df_b])  
df_new.reset_index()
```

	index	subject_id	first_name	last_name
0	0	1	Alex	Anderson
1	1	2	Amy	Ackerman
2	2	3	Allen	Ali
3	3	4	Alice	Aoni
4	4	5	Ayoung	Atiches
5	0	4	Billy	Bonder
6	1	5	Brian	Black
7	2	6	Bran	Balwner

```
df_a.append(df_b)
```

	subject_id	first_name	last_name
0	1	Alex	Anderson
1	2	Amy	Ackerman
2	3	Allen	Ali
3	4	Alice	Aoni
4	5	Ayoung	Atiches
0	4	Billy	Bonder
1	5	Brian	Black

Concat

```
df_new = pd.concat([df_a, df_b], axis=1)  
df_new.reset_index()
```

	index	subject_id	first_name	last_name	subject_id	first_name	last_name
0	0	1	Alex	Anderson	4	Billy	Bonder
1	1	2	Amy	Ackerman	5	Brian	Black
2	2	3	Allen	Ali	6	Bran	Balwner

DB

Persistence

Database connection

- Data loading시 db connection 기능을 제공함

```
import sqlite3
```

Database 연결 코드

```
conn = sqlite3.connect("./data/flights.db")  
cur = conn.cursor()  
cur.execute("select * from airlines limit 5;")  
results = cur.fetchall()  
results
```

db 연결 conn을 사용하여 dataframe 생성

```
df_airplines = pd.read_sql_query("select * from airlines;", conn)  
df_airports = pd.read_sql_query("select * from airports;", conn)  
df_routes = pd.read_sql_query("select * from routes;", conn)
```


XLS persistence

- Dataframe의 엑셀 추출 코드
- Xls 엔진으로 openpyxls 또는 XlsxWrite 사용

```
writer = pd.ExcelWriter('./data/df_routes.xlsx', engine='xlsxwriter')  
df_routes.to_excel(writer, sheet_name='Sheet1')
```


Pickle persistence

- 가장 일반적인 python 파일 persistence
- `to_pickle`, `read_pickle` 함수 사용

```
df_routes.to_pickle("./data/df_routes.pickle")
```

```
df_routes_pickle = pd.read_pickle("./data/df_routes.pickle")  
df_routes_pickle.head()
```

	index	airline	airline_id	source	source_id	dest	dest_id	codeshare	stops	equipment
0	0	2B	410	AER	2965	KZN	2990	None	0	CR2
1	1	2B	410	ASF	2966	KZN	2990	None	0	CR2
2	2	2B	410	ASF	2966	MRV	2962	None	0	CR2

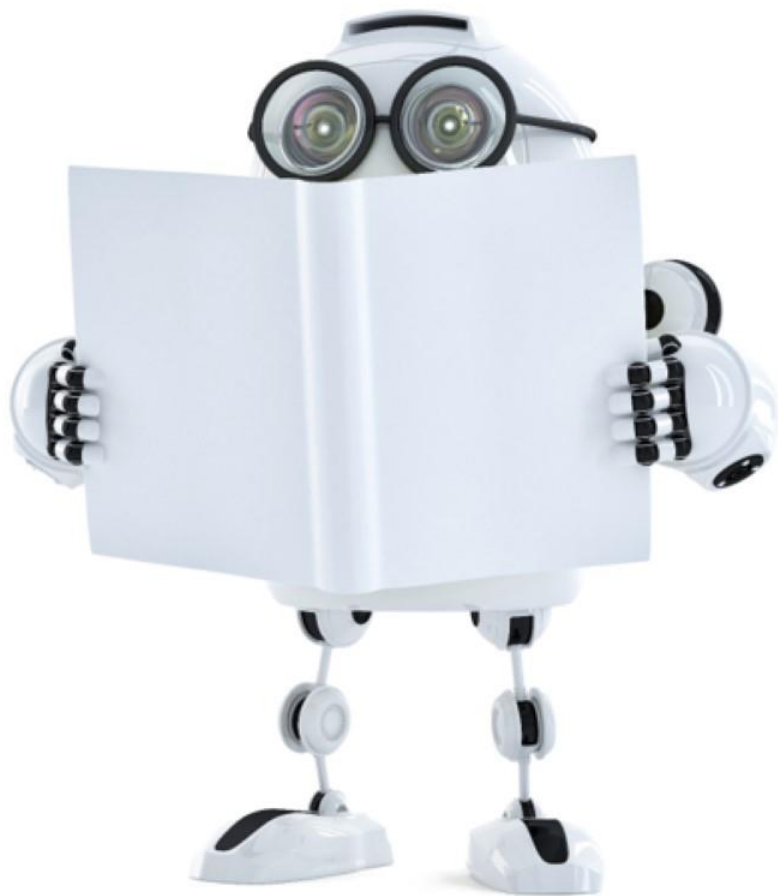
TEAMLAB

Human knowledge belongs to the world.

matplotlib

Visaulization

**Director of TEAMLAB
Sungchul Choi**



matplotlib

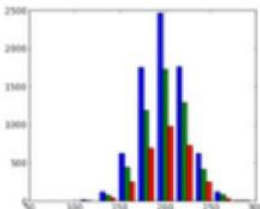
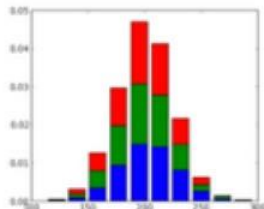
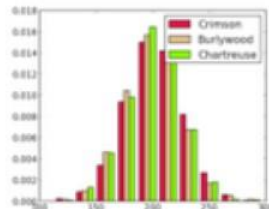
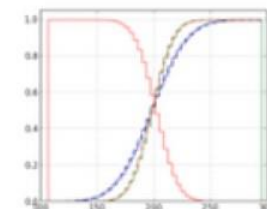
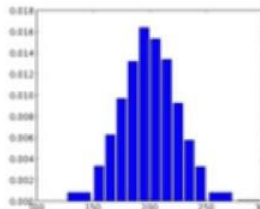
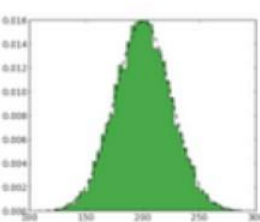
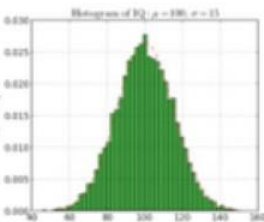
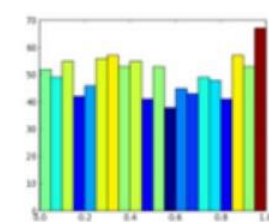
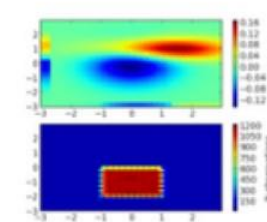
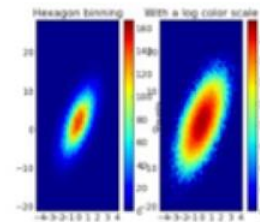
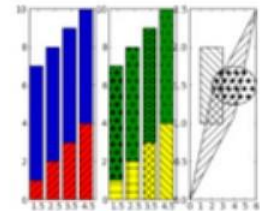
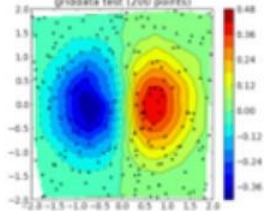
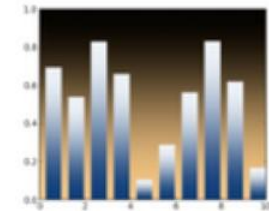
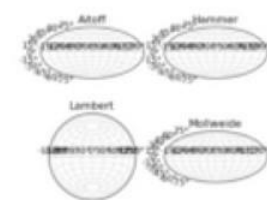
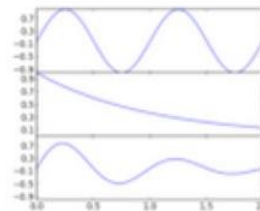
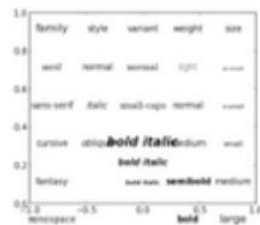
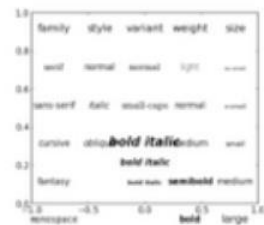
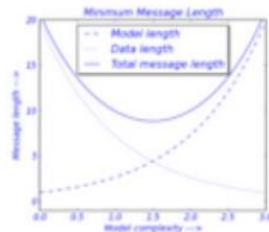
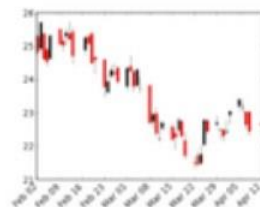
**우리의 데이터는
어떻게 생겼을까?**

Visualization

데이터 시각화

파이썬의 대표적인 시각화 도구

matplotlib



**다양한 graph 지원
Pandas 연동!**

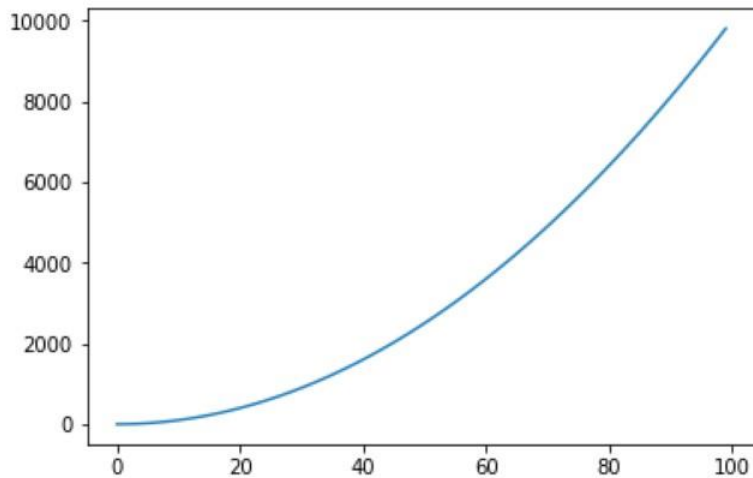
기본 사용법

matplotlib

- pyplot 객체를 사용하여 데이터를 표시
- Pyplot 객체에 그래프들을 쌓은 다음 show로 flush

```
import matplotlib.pyplot as plt

X = range(100)
Y = [value**2 for value in X]
plt.plot(X, Y)
plt.show()
```



matplotlib

- 최대 단점 argument를 kwargs 받음,
- 고정된 argument가 없어서 alt+tab으로 확인이 어려움

Signature: `plt.plot(*args, **kwargs)`

Docstring:

Plot lines and/or markers to the

:class:`~matplotlib.axes.Axes`. `*args*` is a variable length argument, allowing for multiple `*x*`, `*y*` pairs with an optional format string. For example, each of the following is legal::

```
plot(x, y)           # plot x and y using default line style and color
plot(x, y, 'bo')      # plot x and y using blue circle markers
plot(y)              # plot y using x as index array 0..N-1
plot(y, 'r+')         # ditto, but with red plusses
```

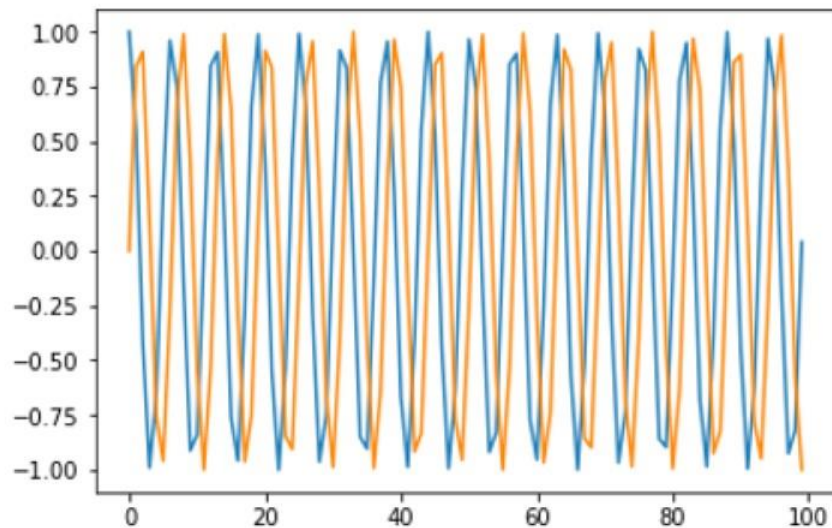
matplotlib

- Graph는 원래 figure 객체에 생성됨
- pyplot 객체 사용시, 기본 figure에 그래프가 그려짐

```
X_1 = range(100)
Y_1 = [np.cos(value) for value in X]

X_2 = range(100)
Y_2 = [np.sin(value) for value in X]

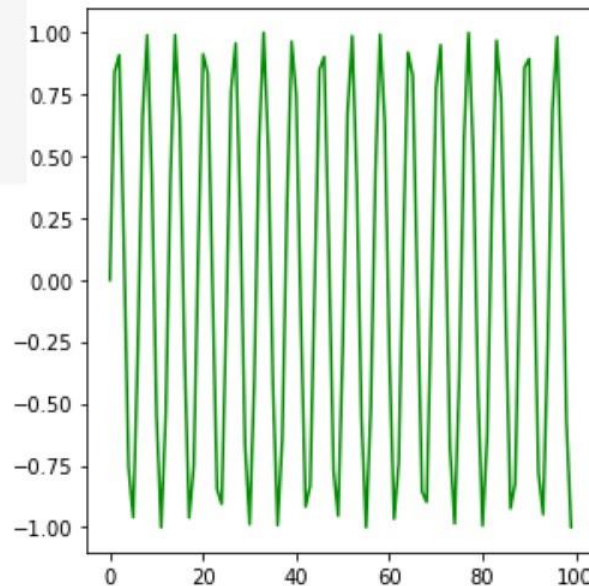
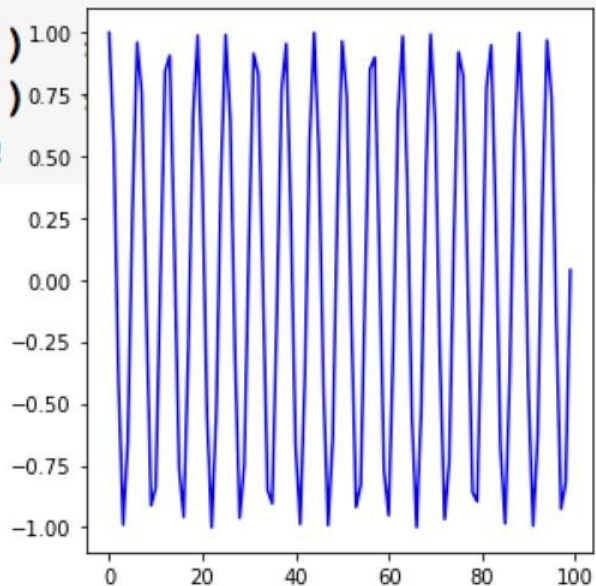
plt.plot(X_1, Y_1)
plt.plot(X_2, Y_2)
plt.show()
```



matplotlib

```
fig = plt.figure() # figure 반환  
fig.set_size_inches(10,5) # 크기지정  
ax_1 = fig.add_subplot(1,2,1) # 두개의 plot 생성  
ax_2 = fig.add_subplot(1,2,2) # 두개의 plot 생성
```

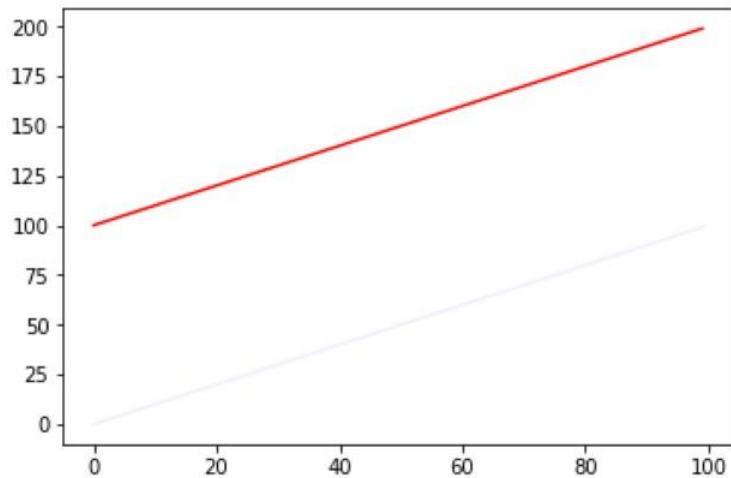
```
ax_1.plot(X_1, Y_1, c="b")  
ax_2.plot(X_2, Y_2, c="g")  
plt.show() # show & flush
```



Set color

- Color 속성을 사용
- Float □□ 흑백, rgb color, predefined color 사용

```
plt.plot(X_1, Y_1, color="#eeefff")  
plt.plot(X_2, Y_2, color="r")  
  
plt.show()
```

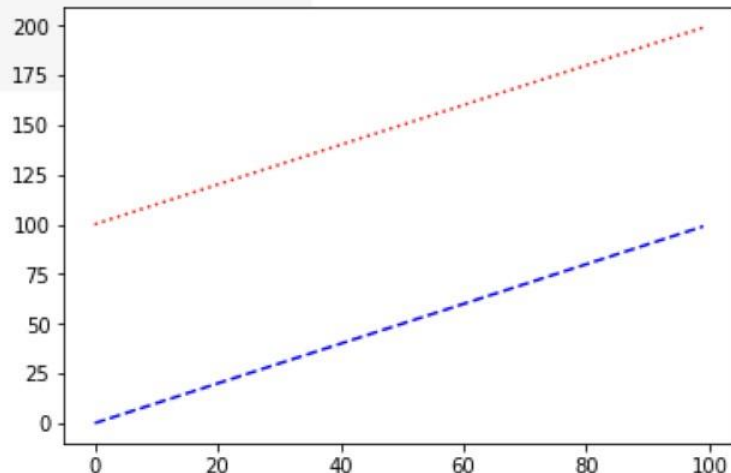


Set linestyle

- ls 또는 linestyle 속성 사용

```
plt.plot(X_1, Y_1, c="b", linestyle="dashed")  
plt.plot(X_2, Y_2, c="r", ls="dotted")
```

```
plt.show()
```

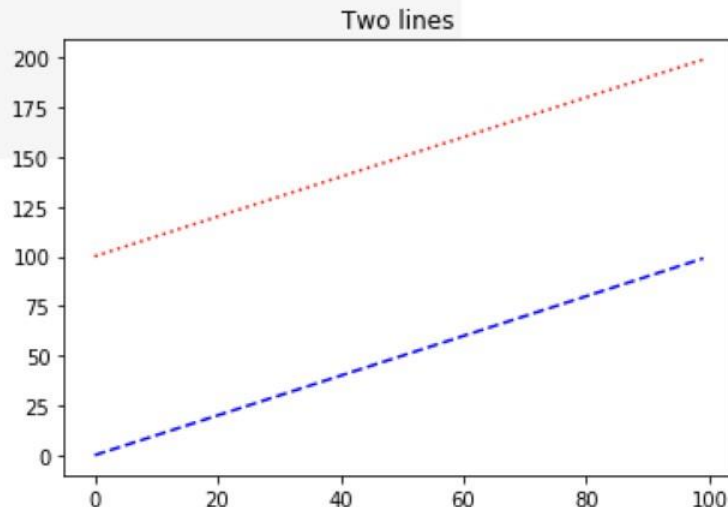


Set title

- Pyplot에 title함수 사용, figure의 subplot별 입력 가능

```
plt.plot(X_1, Y_1, color="b", linestyle="dashed")  
plt.plot(X_2, Y_2, color="r", linestyle="dotted")
```

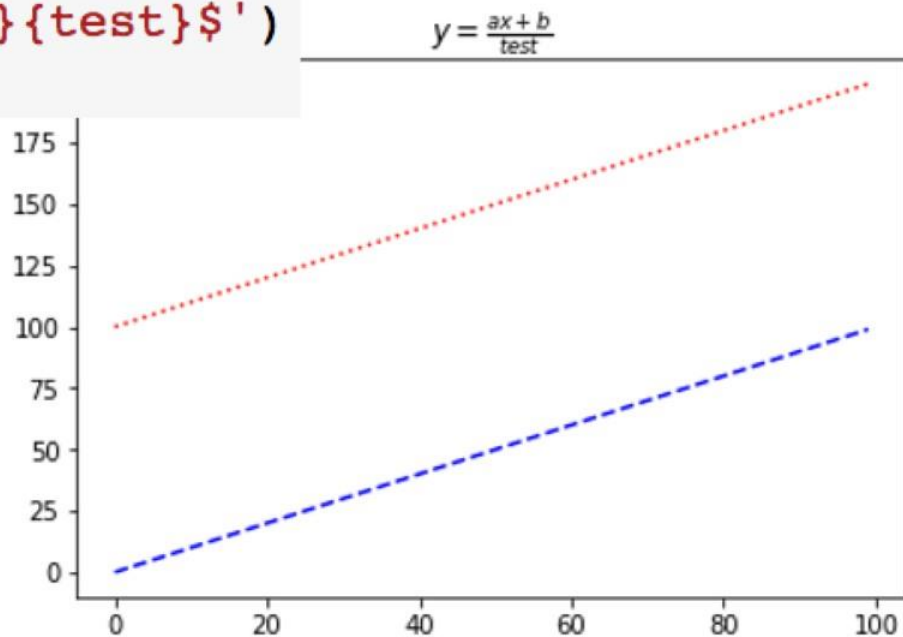
```
plt.title("Two lines")  
plt.show()
```



Set title

- Latex 타입의 표현도 가능 (수식 표현 가능)

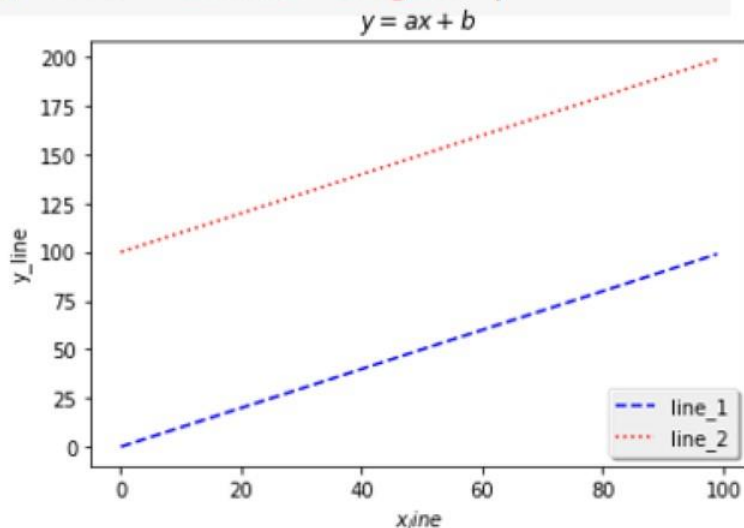
```
plt.title('$y = \frac{ax + b}{test}$')  
plt.show()
```



Set legend

- Legend 함수로 범례를 표시함, loc 위치등 속성 지정

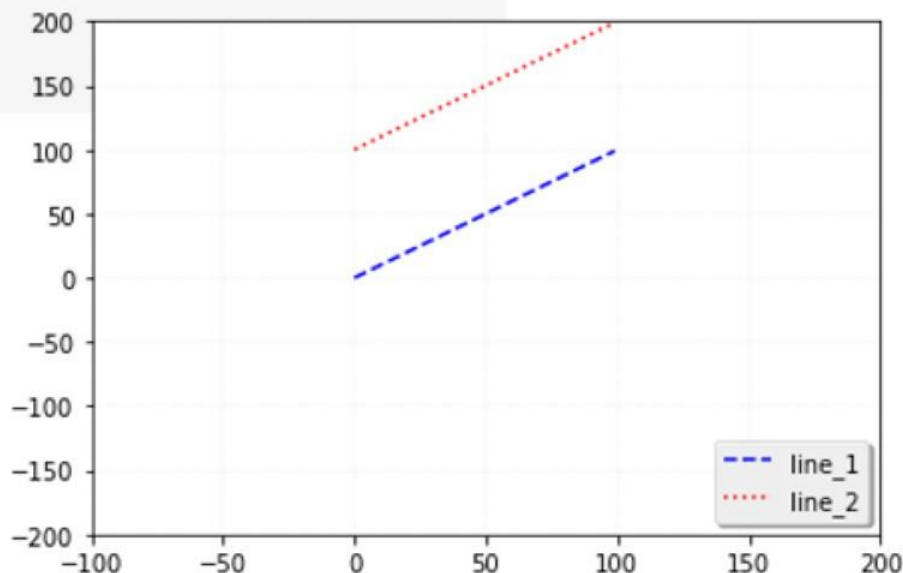
```
plt.plot(X_1, Y_1, color="b", linestyle="dashed", label='line_1')  
plt.plot(X_2, Y_2, color="r", linestyle="dotted", label='line_2')  
plt.legend(shadow=True, fancybox=True, loc="lower right")
```



Set grid & xlim

- Graph 보조선을 긋는 grid와 xy축 범위 한계를 지정

```
plt.grid(True, lw=0.4, ls="--", c=".90")  
plt.xlim(-100, 200)  
plt.ylim(-200, 200)
```



Matplotlib Graph

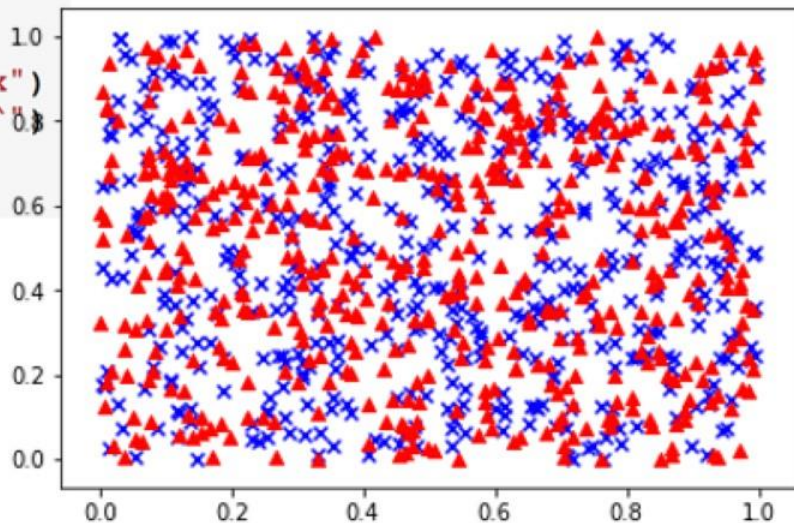
Scatter

- scatter 함수 사용, marker: scatter 모양지정

```
data_1 = np.random.rand(512, 2)
data_2 = np.random.rand(512, 2)

plt.scatter(data_1[:,0], data_1[:,1], c="b", marker="x")
plt.scatter(data_2[:,0], data_2[:,1], c="r", marker="^")

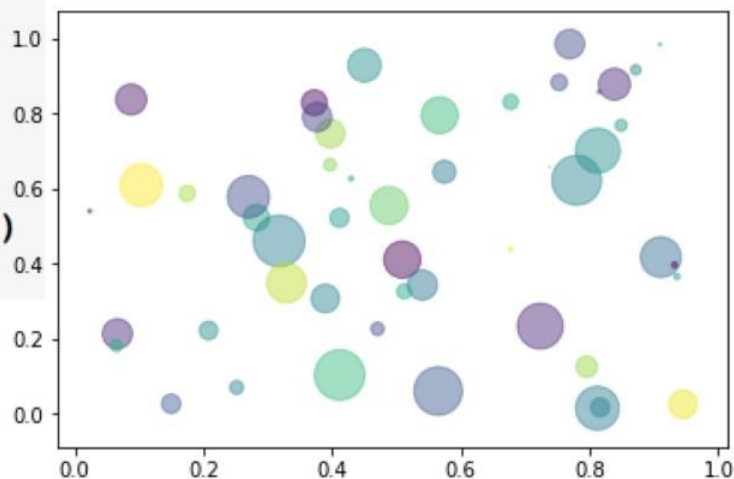
plt.show()
```



Scatter

- **s**: 데이터의 크기를 지정, 데이터의 크기비교가능

```
N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = np.pi * (15 * np.random.rand(N))**2
plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```

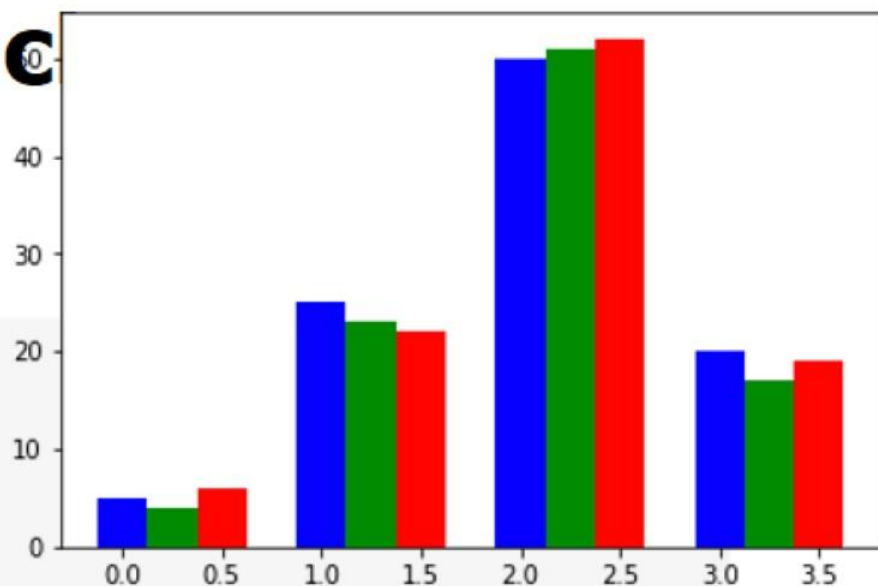


Bar c

- Bar 함수 사용

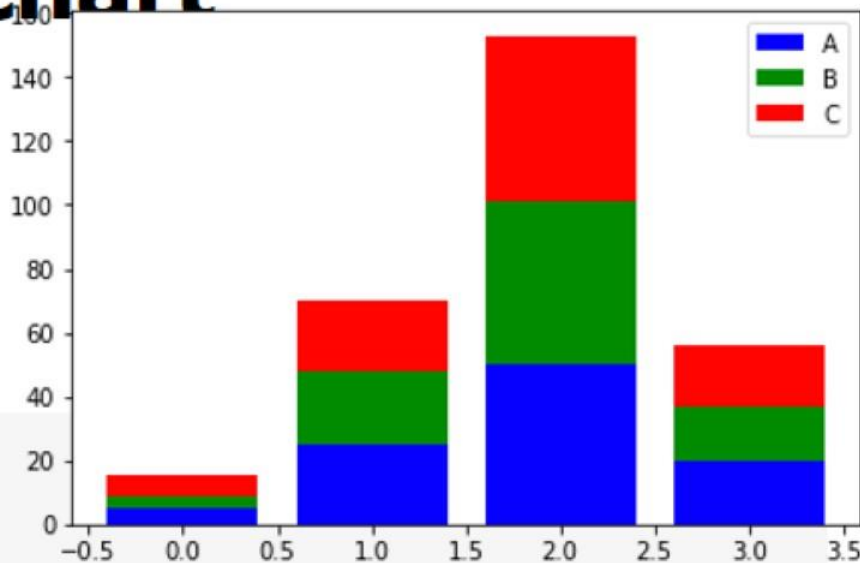
```
data = [[5., 25., 50., 20.],  
        [4., 23., 51., 17],  
        [6., 22., 52., 19]]
```

```
X = np.arange(4)  
plt.bar(X + 0.00, data[0], color = 'b', width = 0.25)  
plt.bar(X + 0.25, data[1], color = 'g', width = 0.25)  
plt.bar(X + 0.50, data[2], color = 'r', width = 0.25)  
plt.xticks(X+0.25, ("A", "B", "C", "D"))  
plt.show()
```



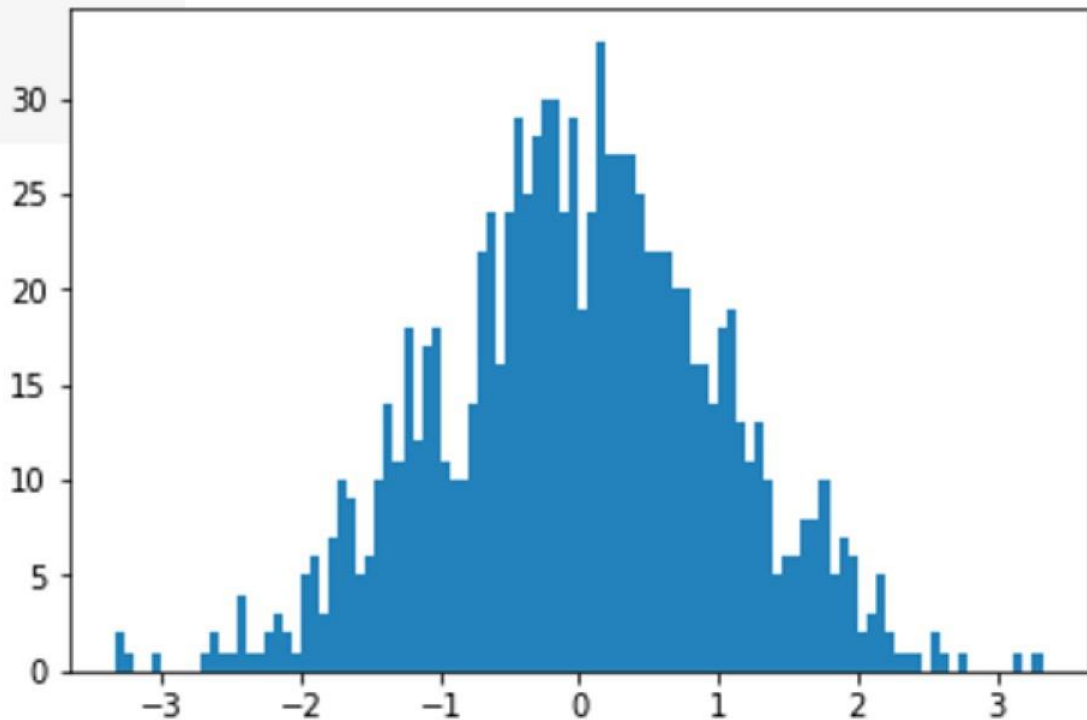
Bar chart

```
color_list = ['b', 'g', 'r']
data_label = ["A", "B", "C"]
X = np.arange(data.shape[1])
for i in range(data.shape[0]):
    plt.bar(X, data[i], bottom = np.sum(data[:i], axis=0),
            color = color_list[i], label=data_label[i])
plt.legend()
plt.show()
```



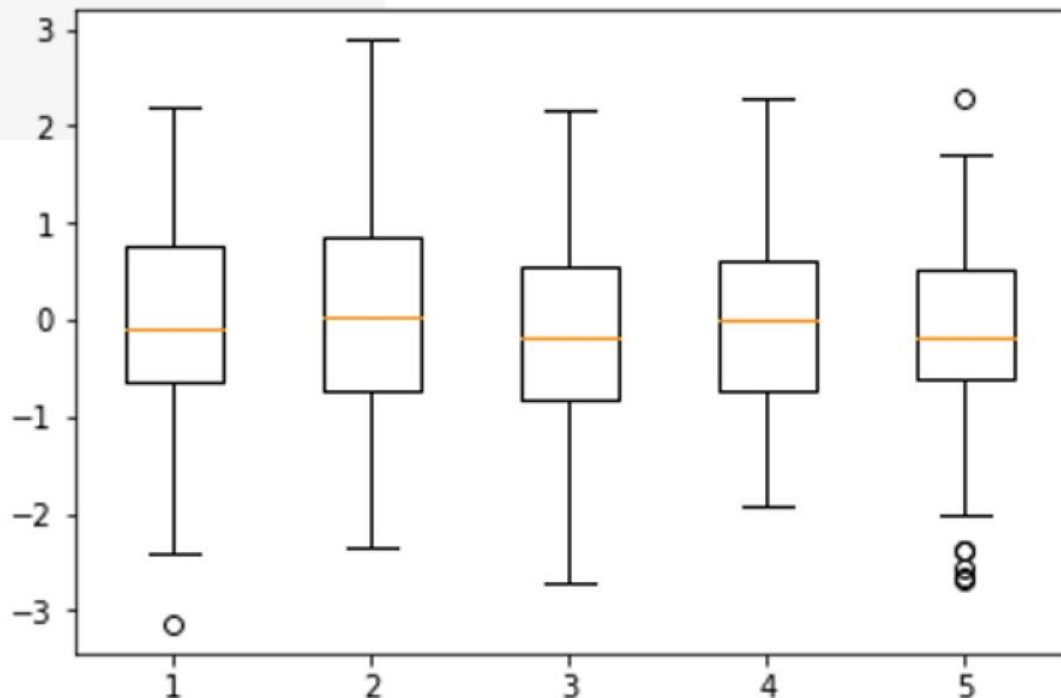
histogram

```
X = np.random.randn(1000)  
plt.hist(X,bins=100)  
plt.show()
```



boxplot

```
data = np.random.randn(100,5)  
plt.boxplot(data)  
plt.show()
```



matplotlib with pandas

pandas matplotlib

- Pandas 0.7 버전이상 부터 matplotlib를 사용한 그래프 지원
- Dataframe, series별로 그래프 작성 가능

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

Boston House Price Dataset

- 머신 러닝 등 데이터 분석을 처음 배울 때,
가장 대표적으로 사용하는 **Example Dataset**
- 1978년에 발표된 데이터로, 미국 인구통계 조사 결과
미국 보스턴 지역의 주택 가격에 영향 요소들을 정리함

<https://lib.stat.cmu.edu/datasets/boston>

Boston House Price Dataset

X 변수
13개

Y 변수

[01]	CRIM	자치시(town) 별 1인당 범죄율
[02]	ZN	25,000 평방피트를 초과하는 거주지역의 비율
[03]	INDUS	비소매상업지역이 점유하고 있는 토지의 비율
[04]	CHAS	찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)
[05]	NOX	10ppm 당 농축 일산화질소
[06]	RM	주택 1가구당 평균 방의 개수
[07]	AGE	1940년 이전에 건축된 소유주택의 비율
[08]	DIS	5개의 보스턴 직업센터까지의 접근성 지수
[09]	RAD	방사형 도로까지의 접근성 지수
[10]	TAX	10,000 달러 당 재산세율
[11]	PTRATIO	자치시(town)별 학생/교사 비율
[12]	B	$1000(Bk - 0.63)^2$, 여기서 Bk는 자치시별 흑인의 비율을 말함.
[13]	LSTAT	모집단의 하위계층의 비율(%)
[14]	MEDV	본인 소유의 주택가격(중앙값) (단위: \$1,000)

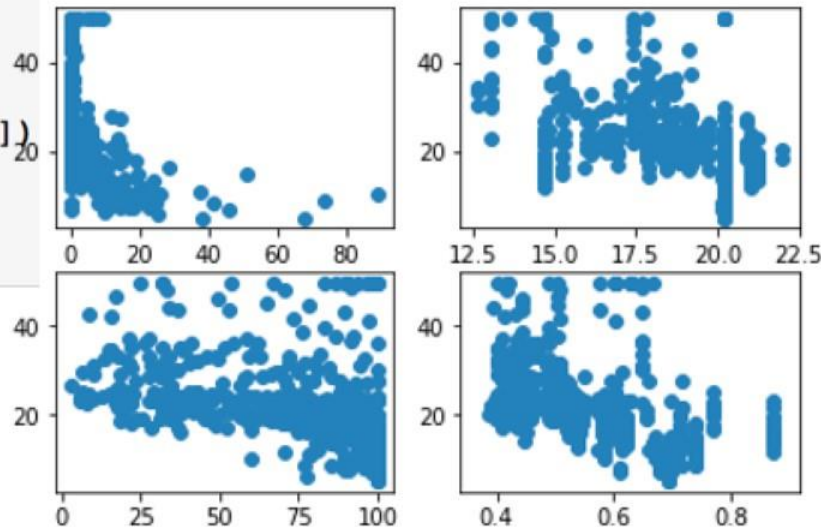
<http://www.dator.co.kr/ctg258/textyle/17213>

<http://www.cs.toronto.edu/~dave/data/boston/bostonDetail.html>

pandas matplotlib

- 데이터간의 상관관계를 볼 때 scatter graph 사용 가능

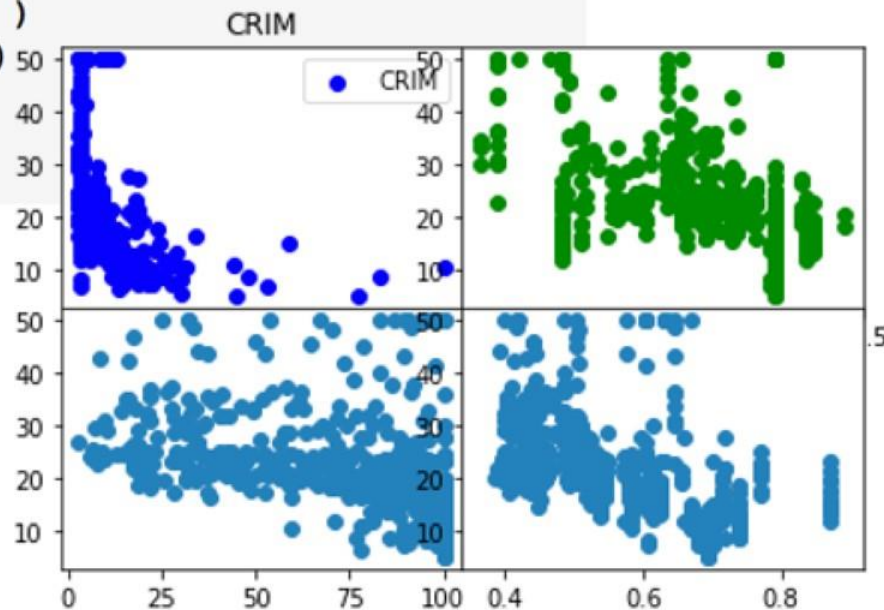
```
fig = plt.figure()
ax = []
for i in range(1,5):
    ax.append(fig.add_subplot(2,2,i))
ax[0].scatter(df_data["CRIM"], df_data["MEDV"])
ax[1].scatter(df_data["PTRATIO"], df_data["MEDV"])
ax[2].scatter(df_data["AGE"], df_data["MEDV"])
ax[3].scatter(df_data["NOX"], df_data["MEDV"])
plt.show()
```



pandas matplotlib

- matplotlib의 꾸미기 함수 그대로 사용함

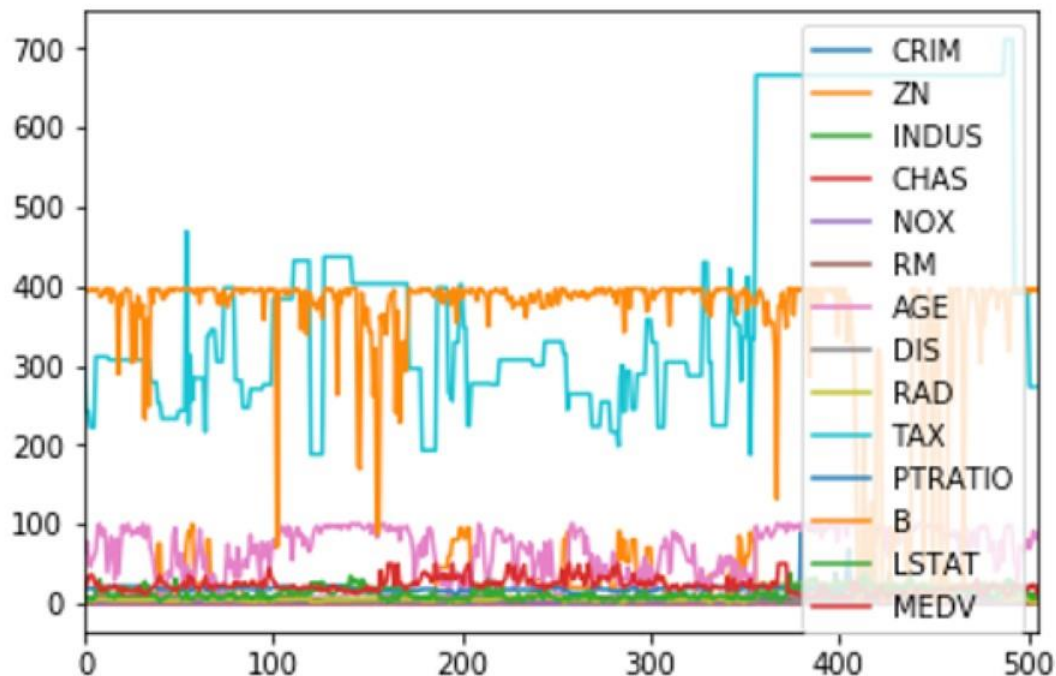
```
ax[0].scatter(df_data["CRIM"], df_data["MEDV"], color="b", label="CRIM")
ax[1].scatter(df_data["PTRATIO"], df_data["MEDV"], color="g" )
ax[2].scatter(df_data["AGE"], df_data["MEDV"] )
ax[3].scatter(df_data["NOX"], df_data["MEDV"])
plt.subplots_adjust(wspace=0, hspace=0)
ax[0].legend()
ax[0].set_title("CRIM")
```



pandas matplotlib

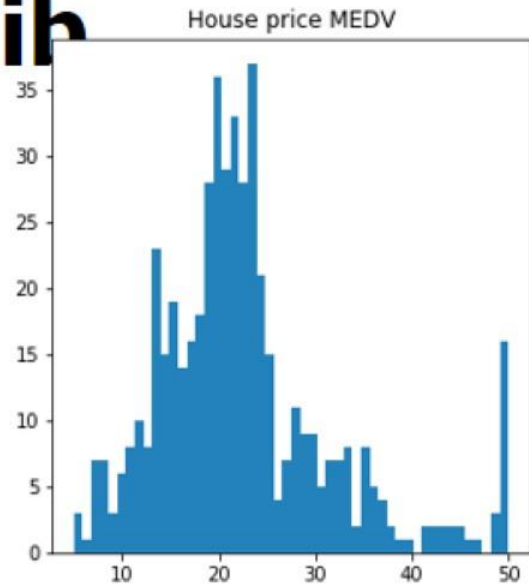
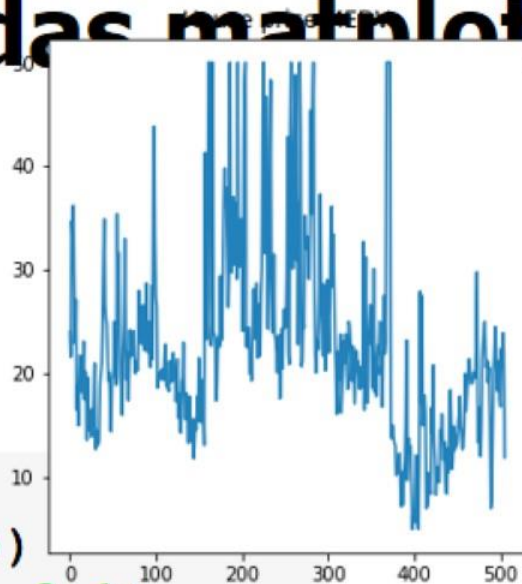
- plot 함수를 사용하면 전체 데이터의 graph를 그림

```
df_data.plot()
```



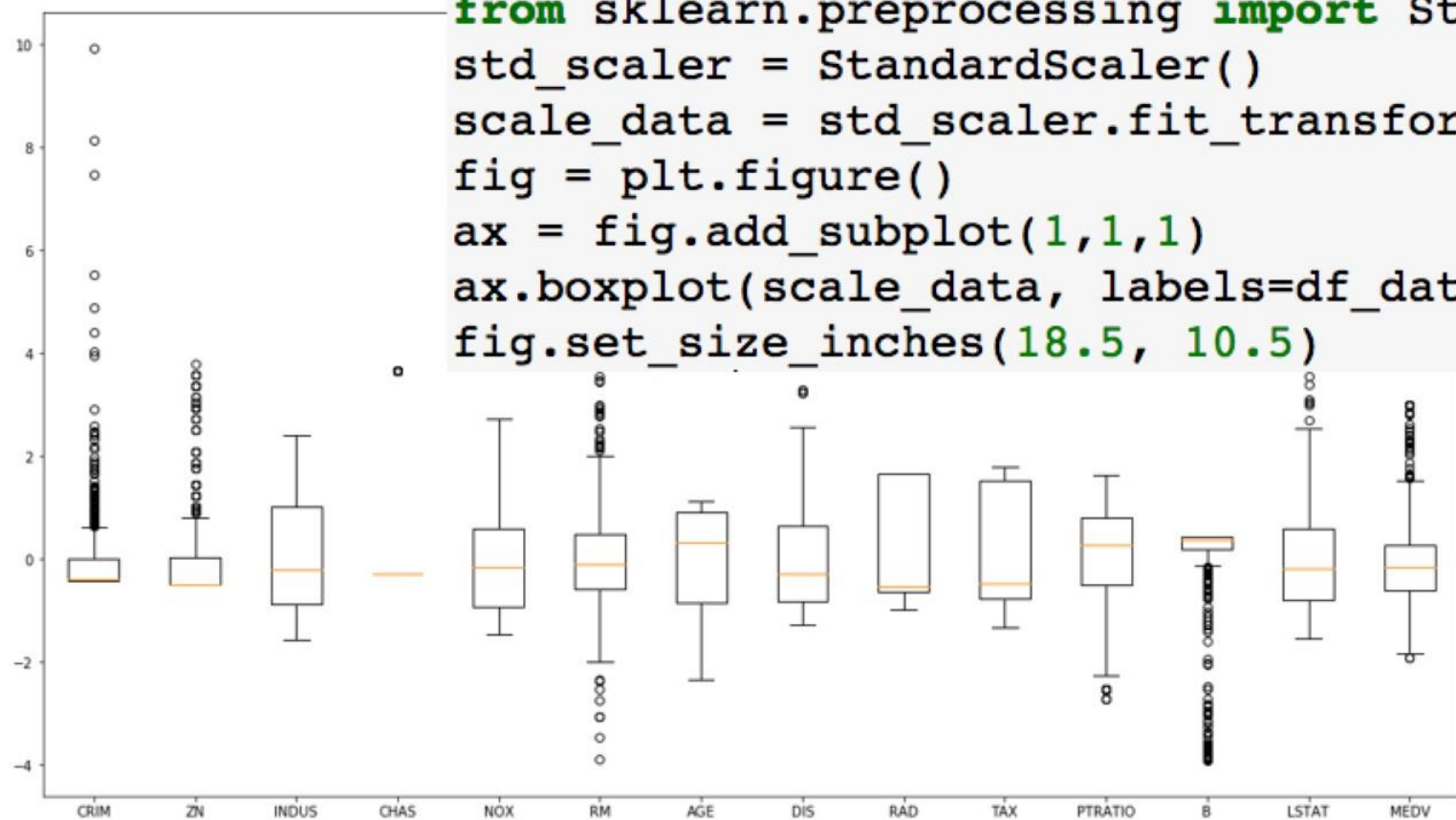
pandas matplotlib

```
fig = plt.figure()
fig.set_size_inches(10,5)
ax_1 = fig.add_subplot(1,2,1)
ax_2 = fig.add_subplot(1,2,2)
ax_1.plot(df_data["MEDV"])
ax_2.hist(df_data["MEDV"], bins=50)
ax_1.set_title("House price MEDV")
ax_2.set_title("House price MEDV")
```



Scaled boxplot

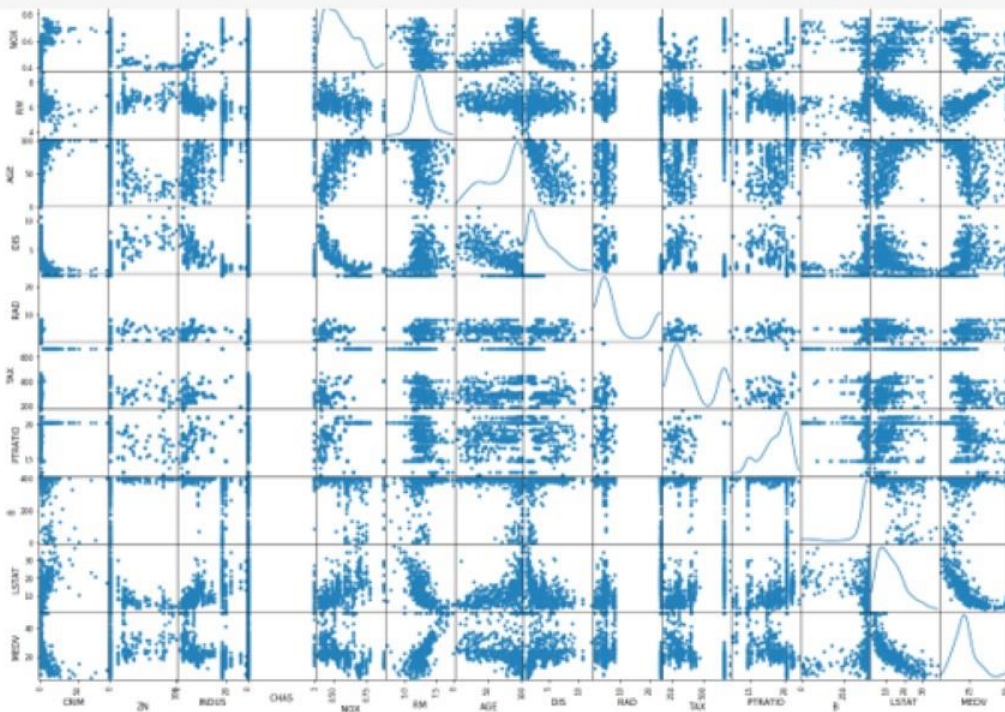
```
from sklearn.preprocessing import StandardScaler
std_scaler = StandardScaler()
scale_data = std_scaler.fit_transform(df_data)
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.boxplot(scale_data, labels=df_data.columns)
fig.set_size_inches(18.5, 10.5)
```



Scatter



```
pd.scatter_matrix(df_data, diagonal="kde", alpha=1, figsize=(20, 20))  
plt.show()
```



Scatter matrix

```
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(corr_data, vmin=-1, vmax=1, interpolation='nearest')
fig.colorbar(cax)
fig.set_size_inches(10,10)
ticks= np.arange(0,14,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)

ax.set_xticklabels(df_data.columns)
ax.set_yticklabels(df_data.columns)
```

