



파이썬 고급 기능과 데이터 탐색, 전처리, 분석, 시각화 과정

2021. 3. 22~24 (3일 과정)

정준수 Ph.D

과정 목표

1. 파이썬 함수의 고급 기능들을 습득하고 이를 기반으로 고급 모듈 및 패키지를 개발할 수 있는 능력 향상
2. N차원 배열을 다루고 고급 인덱싱과 선형 대수학을 이용한 데이터 탐색 및 분석방법에 대해서 습득
3. 데이터프레임 저장, 탐색, 병합, 연결, 집계, 구조변경 하는 방법을 습득
4. 탐색 데이터를 matplotlib과 seaborn을 이용하여 2차원으로 플롯하는 방법을 습득
5. 실무 데이터를 활용한 데이터 전처리 프로젝트를 통해 현업의 데이터 탐색 및 전처리가 어떻게 이루어지는지 과정 습득

<실습 프로그램>

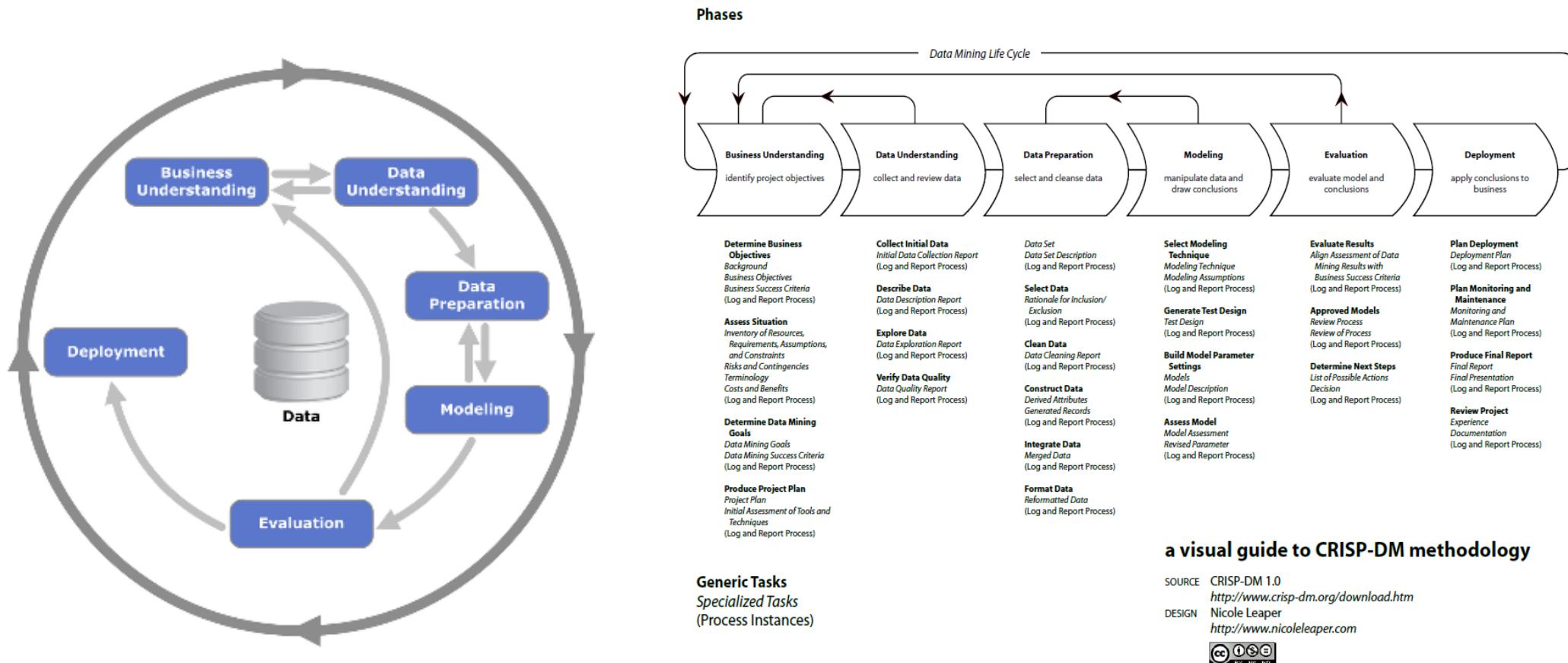
<https://github.com/JSJeong-me/KOSA-Python-Advance>

Deep learning is not like pure mathematics. It is a heavily experimental field, so it's important to be a strong practitioner, not just a theoretician.

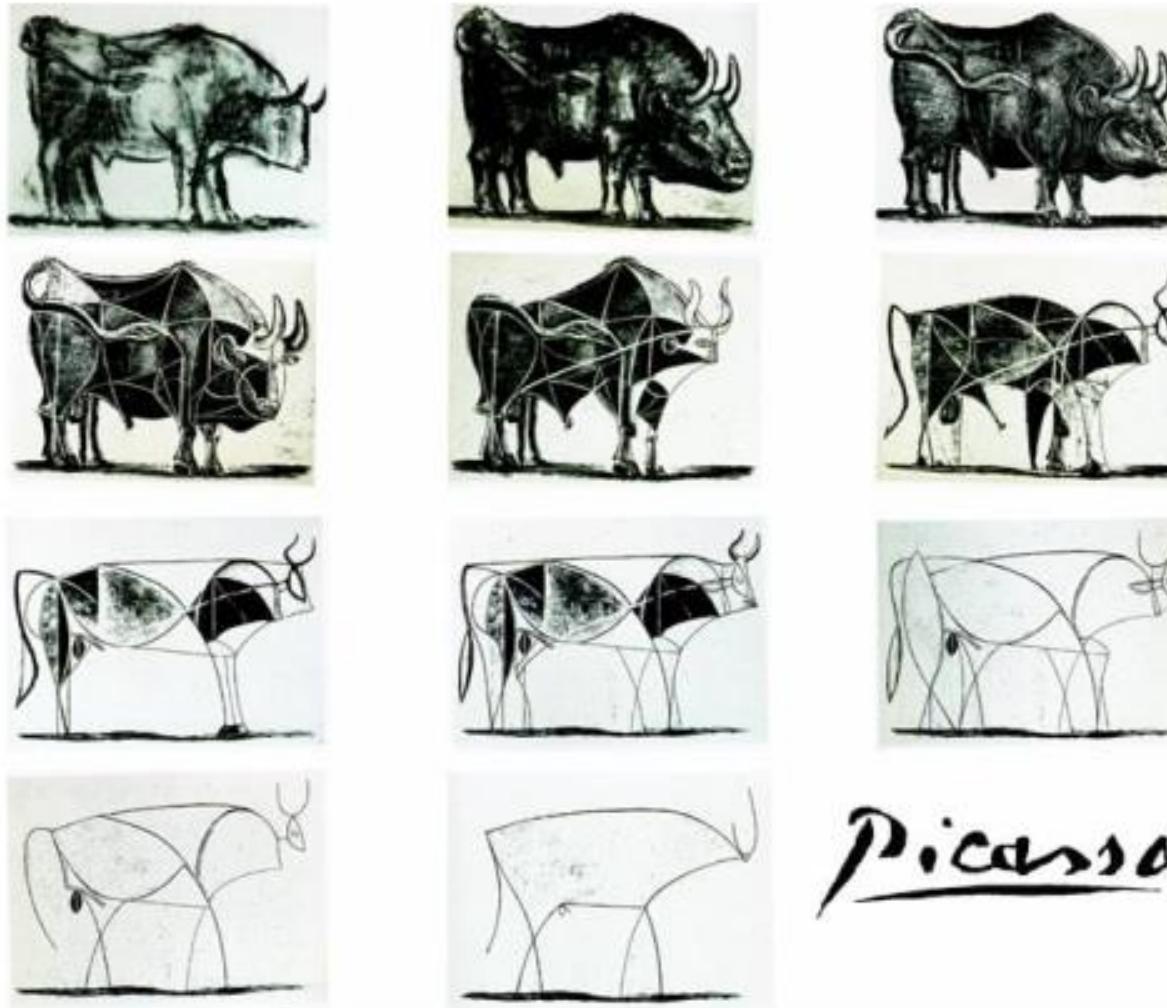
딥 러닝은 순수한 수학과는 다릅니다. 매우 실험적인 분야이기 때문에 이론가가 아닌 실무자가 되는 것이 중요합니다.

CRISP-DM (Cross Industry Standard Process for Data Mining)

CRISP-DM(Cross Industry Standard Process for Data Mining)은 데이터 마이닝 전문가가 사용하는 일반적인 접근 방식을 설명한 가장 널리 사용되는 공개 표준 분석 모델입니다.



◆ 추상화 (Abstract)



Pablo Picasso, Bull (plates I - XI) 1945

일반적 Data Preparation 과정 정리

1. 데이터 준비 과정의 중요성
2. 결측치의 처리방법
3. 특징 추출 (Recursive Feature Elimination)
4. 데이터 정규화
5. 원 핫 인코딩으로 범주 변환 (One Hot Encoding)
6. 숫자 변수의 범주형 변수로 변환
7. PCA를 통한 차원 축소

1. 데이터 준비 과정의 중요성

데이터 준비 과정은 원시 데이터를 모델링에 적합한 형식으로 변환하는 작업이 중심 내용이며, 데이터를 준비하는 것은 예측 모델 생성 프로젝트에서 가장 중요한 부분이며 가장 많은 시간이 소요된다. 모든 데이터 준비 과정은 기계 학습 알고리즘에 맞추어져 진행 되며, 실제 데이터와 매개 변수 등을 활용한다. 실질적인 데이터 준비 과정에는 데이터 정리, 특징 추출, 데이터 변환, 차원 축소 등에 대한 지식이 필요하다.

예측 모델링 프로젝트에는 데이터로부터 학습이 진행됩니다. 데이터는 해결하려는 문제를 특징 짓는 도메인으로부터 경험 할 수 있는 사례를 제공하지만, 분류 또는 회귀와 같은 예측 모델링 프로젝트에서 원시 데이터는 일반적으로 직접 사용할 수 없습니다. 이것이 사실인 네 가지 주요 이유가 있습니다.

원시 데이터는 기계 학습 모델을 맞추고 평가하는 데 사용되기 전에 사전 처리 되어야 합니다. 기계 학습 프로젝트의 데이터 준비 단계에서 사용하거나 탐색 할 수 있는 일반 또는 표준 작업이 있습니다.

- 데이터 정리 : 데이터의 오류 또는 오류를 식별하고 수정
- 특징 선택 : 작업과 가장 관련된 입력 변수 식별
- 데이터 변환 : 변수의 척도 또는 분포 파악
- 특징 엔지니어링 : 사용 가능한 데이터에서 새로운 변수 도출
- 차원 감소 : 데이터의 간결한 예측 생성

2. 결측치의 처리방법

실제 데이터에는 종종 결 측값이 있습니다. 데이터에는 기록되지 않은 관찰 및 데이터 손상과 같은 여러 가지 이유로 인해 누락 된 값이 있을 수 있습니다. 누락 된 데이터 처리가 중요합니다. 많은 기계 학습 알고리즘이 결 측값이 있는 데이터를 지원하지 않기 때문입니다. 누락 된 값을 데이터로 채우는 것을 데이터 대치라고 하며 데이터 대치에 대한 일반적인 접근 방식은 각 열 (예 : 평균)에 대한 통계 값을 계산하고 해당 열의 모든 누락 된 값을 통계로 바꾸는 것입니다.

(결측치 처리 예제 – 단순 평균값 입력)

SimpleImputer() 클래스를 사용하여 NaN 값으로 표시된 모든 누락 된 값을 열의 평균으로 변환 할 수 있습니다.

<실습 프로그램>

<https://gist.github.com/JSJeong-me/fdbba476a9cff9400ba32064a92f54e8>

3. 특징 추출 (Recursive Feature Elimination)

특징 선택은 예측 모델을 개발할 때 입력 변수의 수를 줄이는 프로세스입니다. 모델링의 계산 비용을 줄이고 경우에 따라 모델의 성능을 향상시키기 위해 입력 변수의 수를 줄이는 것이 바람직합니다. 간단히 RFE는 인기있는 기능 선택 알고리즘입니다.

RFE는 구성 및 사용이 쉽고 대상 변수를 예측하는 데 더 많거나 가장 관련성이 높은 학습 데이터 세트에서 이러한 기능 (열)을 선택하는 데 효과적이기 때문에 널리 사용됩니다.

(특징 추출 예제 – scikit-learn 사용)

scikit-learn Python 기계 학습 라이브러리는 기계 학습을 위한 RFE 구현을 제공합니다. RFE 변환을 사용하려면 먼저 추정치 인수를 통해 지정된 선택한 알고리즘과 인수를 선택하기 위해 n 개의 기능을 통해 선택할 기능수로 클래스를 구성합니다. 다음의 예는 5 개의 중복 입력 기능이 있는 합성 분류 데이터 세트를 정의합니다. 그런 다음 RFE를 사용하여 의사 결정 트리 알고리즘을 사용하여 5 개의 기능을 선택합니다.

<실습 프로그램>

<https://gist.github.com/JSeong-me/7d6a3f852eb0e9eb451e4c153af6cc6f>

4. 데이터 정규화

기계 학습을 위해 숫자 데이터를 확장하는 방법을 알아 봅니다. 많은 기계 학습 알고리즘은 숫자 입력 변수가 표준 범위로 조정될 때 더 잘 수행됩니다. 여기에는 선형 회귀와 같은 입력의 가중 합계를 사용하는 알고리즘과 k- 최근 접 이웃과 같은 거리 측정을 사용하는 알고리즘이 포함됩니다.

모델링 전에 수치 데이터를 스케일링하는 가장 널리 사용되는 기술 중 하나는 정규화입니다. 정규화는 각 입력 변수를 0-1 범위로 개별적으로 조정합니다. 이는 가장 정밀도가 높은 부동 소수점 값의 범위입니다. 각 변수에 대한 최소 및 최대 관찰 가능 값을 알고 있거나 정확하게 추정 할 수 있어야합니다.

(데이터 정규화 예제 – MinMaxScaler 사용)

scikit-learn 객체 MinMaxScaler를 사용하여 데이터 세트를 정규화 할 수 있습니다. 아래 예제는 합성 분류 데이터 세트를 정의한 다음 MinMaxScaler를 사용하여 입력 변수를 정규화합니다.

<실습 프로그램>

<https://gist.github.com/JSJeong-me/fa8e38c0d0960f520731de45f7e1b6eb>

5. 원 핫 인코딩으로 범주 변환 (One Hot Encoding)

범주 형 입력 변수를 숫자로 인코딩하는 방법을 알아 봅니다. 기계 학습 모델에서는 모든 입력 및 출력 변수가 숫자여야 합니다. 즉, 데이터에 범주형 데이터가 포함된 경우 모델을 적합하고 평가하기 전에 이를 숫자로 인코딩 해야합니다. 범주 형 변수를 숫자로 변환하는 가장 널리 사용되는 기술 중 하나는 원 핫 인코딩입니다. 범주 형 데이터는 숫자 값이 아닌 레이블 값을 포함하는 변수입니다.

범주 형 변수에 대한 각 레이블은 서수 인코딩이라고하는 고유 한 정수에 매핑 될 수 있습니다. 그런 다음 서수 표현에 원 핫 인코딩을 적용 할 수 있습니다. 여기서 변수의 고유 한 정수 값 각각에 대해 하나의 새 이진 변수가 데이터 세트에 추가되고 원래 범주 형 변수가 데이터 세트에서 제거됩니다.

(데이터 Encoding 예제 – One Hot Encoding 사용)

이 핫 인코딩 변환은 OneHotEncoder 클래스를 통해 scikit-learn Python 기계 학습 라이브러리에서 사용할 수 있습니다. 유방암 데이터 세트에는 범주형 입력 변수만 포함됩니다. 아래 예제는 데이터 세트를 로드하고 하나의 핫 인코딩은 각 범주형 입력 변수를 인코딩합니다.

<실습 프로그램>

<https://gist.github.com/JSeong-me/664af116682e32f0e1fb0141c280d879>

6. 숫자 변수의 범주형 변수로 변환

숫자 변수를 범주형 변수로 변환하는 방법을 알아 봅니다. 일부 기계 학습 알고리즘은 일부 의사 결정 트리 및 규칙 기반 알고리즘과 같은 범주 형 또는 순서 형 입력 변수를 선호하거나 요구할 수 있습니다. 이것은 데이터, 다중 입력 데이터 분포, 고도의 지수 분포 등. 많은 기계 학습 알고리즘은 비표준 분포를 가진 숫자 입력 변수가 새로운 분포 또는 완전히 새로운 데이터 유형을 갖도록 변환 될 때 더 나은 성능을 제공합니다.

한 가지 접근 방식은 숫자 변수의 변환을 사용하여 각 숫자 값에 레이블이 할당되고 레이블에 순서가 지정된 (순서적) 관계가 있는 이산 확률 분포를 갖는 것입니다. 이를 이산화 변환이라고 하며 숫자 입력 변수의 확률 분포를 이산 적으로 만들어 데이터 세트에 대한 일부 기계 학습 모델의 성능을 향상시킬 수 있습니다.

(숫자 변수의 범주형 변수 변환 예제 – KBinsDiscretizer 사용)

이산화 변환은 KBinsDiscretizer 클래스를 통해 scikit-learn Python 기계 학습 라이브러리에서 사용할 수 있습니다. 생성 할 이산 구간의 수 (n 구간), 변환 결과가 서수인지 아니면 하나의 핫 인코딩 (인코딩)인지, 변수 값을 나누는 데 사용되는 분포 (전략)를 지정할 수 있습니다. 아래 예제에서는 10 개의 숫자 입력 변수가 있는 합성 입력 변수를 생성 한 다음 각각을 서수 인코딩을 사용하여 10 개의 개별 Bin(바구니)으로 인코딩합니다.

<실습 프로그램>

<https://gist.github.com/JSJeong-me/34721c717cf4a4f9e5eaee4f33124c2>

7. PCA를 통한 차원 축소

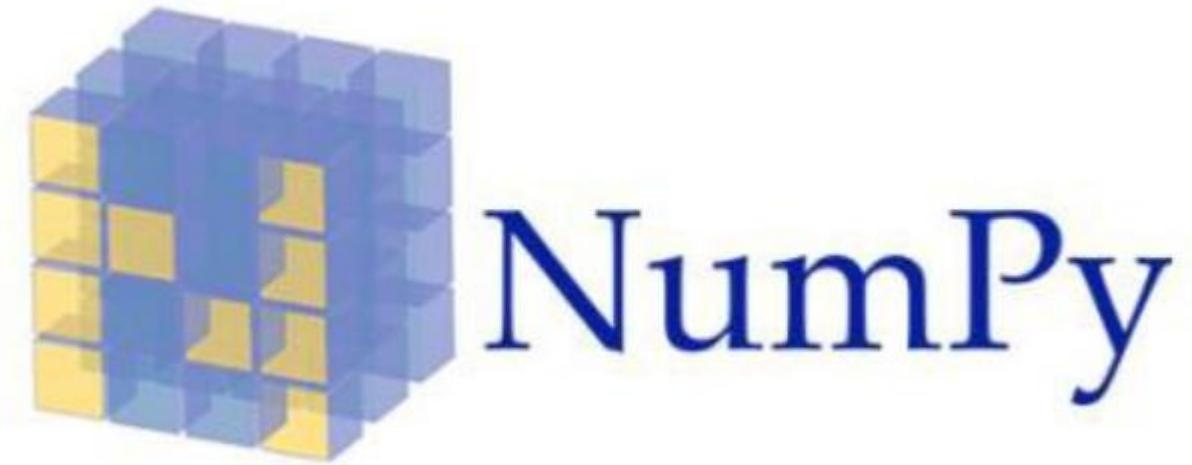
차원 감소를 사용하여 데이터 세트의 입력 변수를 줄이는 방법을 알아 봅니다. 데이터 세트에 대한 입력 변수 또는 기능의 수를 차원이라고 합니다. 차원 감소는 Features 수를 줄이는 기술을 의미합니다. 데이터 세트의 입력 변수. 더 많은 입력 기능은 종종 예측 모델링 작업을 모델링하기 더 어렵게 만들고, 일반적으로 차원의 저주라고 합니다. 고차원 통계에서는 차원 감소 기술이 데이터 시각화에 자주 사용되지만, 이러한 기술은 예측 모델에 더 적합하도록 분류 또는 회귀 데이터 세트를 단순화하기 위해 적용된 기계 학습에서 사용할 수 있습니다. 기계 학습에서 차원 감소를 위한 가장 인기있는 기술은 주성분 분석 (줄여서 PCA) 일 것입니다.

(PCA를 통한 차원 축소 예제 – PCA)

scikit-learn 라이브러리는 데이터 세트에 적합하고 향후 학습 데이터 세트 및 추가 데이터 세트를 변환하는 데 사용할 수 있는 PCA 클래스를 제공합니다. 아래 예제는 10 개의 입력 변수가 있는 합성 이진 분류 데이터 세트를 만든 다음 PCA를 사용하여 데이터 세트의 차원을 가장 중요한 세 가지 구성 요소로 줄입니다.

<실습 프로그램>

<https://gist.github.com/JSJeong-me/ec0826fc9da3f7e1bac19bcf95aef47f>



NumPy

numpy

- 대용량 데이터 배열을 효율적으로 다룰 수 있도록 설계됨
 - numpy는 데이터를 다른 내장 파이썬 객체와 구분되는 연속된 메모리 블록에 저장
 - numpy의 각종 알고리즘은 모두 C로 작성되어 타입 검사나 다른 오버헤드 없이 메모리를 직접 조작할 수 있음
 - numpy 배열은 내장 파이썬의 연속된 자료형들보다 훨씬 더 적은 메모리를 사용함
 - numpy 연산은 파이썬 반복문을 사용하지 않고 전체 배열에 대한 복잡한 계산을 수행함

numpy

- 파이썬 리스트 자료형의 대안으로 선형대수, 행렬, 이미지 연산 등 지원

```
mass = [10, 20, 30, 40, 50]
accel = [1, 3, 7, 9, 11]

force = mass * accel
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-4-9ca3a37b620b> in <module>
      2 accel = [1, 3, 7, 9, 11]
      3
----> 4 force = mass * accel

TypeError: can't multiply sequence by non-int of type 'list'
```

```
force = []
for i in range(len(mass)):
    force.append(mass[i] * accel[i])

force
```

```
[10, 60, 210, 360, 550]
```

```
import numpy as np

mass = np.array([10, 20, 30, 40, 50])
accel = np.array([1, 3, 7, 9, 11])

force = mass * accel
force

array([ 10,   60,  210,  360,  550])
```

numpy 배열 vs. Python 리스트

- 성능 비교

```
1 import numpy as np  
2  
3 a_array = np.arange(1_000_000) # numpy ndarray  
4 a_list = list(range(1_000_000)) # python List
```

```
1 %time for _ in range(10): b_array = a_array * 2
```

Wall time: 21 ms

```
1 %time for _ in range(10): b_list = [x*2 for x in a_list]
```

Wall time: 899 ms

```
1 round(899/21, 1) # 넘파이가 약 43배 빠름
```

42.8

NumPy ndarray : 다차원 배열 객체

- ndarray
 - 같은 종류의 데이터를 담을 수 있는 포괄적인 다차원 배열
 - 단, 리스트와 달리 모든 원소는 같은 자료형 이어야 함

```
1 # Generate some random number  
2 data = np.random.randn(2, 3)
```

```
1 data
```

```
array([[-0.14039762,  0.41357354, -0.54189105],  
      [-0.1666982 ,  0.0870287 ,  0.73809299]])
```

```
1 data * 10
```

```
array([[-1.40397618,  4.13573537, -5.41891054],  
      [-1.66698199,  0.87028696,  7.38092994]])
```

```
1 data + data
```

```
array([[-0.28079524,  0.82714707, -1.08378211],  
      [-0.3333964 ,  0.17405739,  1.47618599]])
```

NumPy ndarray : 다차원 배열 객체

- `shape`
 - 각 차원의 크기를 알려줌
- `dtype`
 - 튜플과 배열에 저장된 자료형을 알려줌

```
1 data.shape
```

(2, 3)

```
1 data.dtype
```

`dtype('float64')`

NumPy ndarray : ndarray 생성하기

- array(배열) 함수
 - 배열 혹은 순차적인 객체를 전달 받아 새로운 numpy 배열 생성

```
1 data_01 = [5, 6, 7.5, 8, 9.9, 10]
2 array_01 = np.array(data_01)
3 array_01 # 1차원 배열
```

```
array([ 5. ,  6. ,  7.5,  8. ,  9.9, 10. ])
```

```
1 data_02 = [[1,2,3,4,5], [6,7,8,9,10]]
2 array_02 = np.array(data_02)
3 array_02 # 2차원 배열
```

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

NumPy ndarray : ndarray 생성하기

- ndim
 - 차원 확인

```
1 array_01.ndim
```

```
1
```

```
1 array_01.shape # tuple
```

```
(6,)
```

```
1 array_01.dtype
```

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

```
1 array_02.ndim
```

```
2
```

```
1 array_02.shape # tuple
```

```
(2, 5)
```

```
1 array_02.dtype
```

```
dtype('int32')
```

NumPy ndarray : ndarray 생성하기

- zeros, ones 함수
 - 주어진 길이나 모양에 각각 0과 1이 들어 있는 배열 생성
- empty 함수
 - 초기화되지 않은 배열 생성
- 생성 시 원하는 형태의 튜플을 함수로 전달

```
1 np.zeros(10)
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

1 np.ones((2, 10))
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]])
```

```
1 np.empty((2, 5, 3)) # 5 x 3 배열 2개인 3차원 배열
array([[[2.29175545e-312, 2.10077583e-312, 2.07955588e-312],
        [2.05833592e-312, 2.27053550e-312, 2.29175545e-312],
        [2.10077583e-312, 2.07955588e-312, 2.05833592e-312],
        [2.27053550e-312, 2.29175545e-312, 2.10077583e-312],
        [2.07955588e-312, 2.05833592e-312, 2.27053550e-312]],

       [[2.29175545e-312, 2.10077583e-312, 2.07955588e-312],
        [2.05833592e-312, 2.27053550e-312, 2.29175545e-312],
        [2.10077583e-312, 2.07955588e-312, 2.05833592e-312],
        [2.27053550e-312, 2.29175545e-312, 2.10077583e-312],
        [2.07955588e-312, 2.05833592e-312, 2.27053550e-312]])]
```

NumPy ndarray : ndarray 생성하기

- arange 함수
 - 파이썬 range 함수의 배열 버전

```
1 np.arange(15) # np.arange(0, 15, 1)
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

```
1 np.linspace(1, 5, 10) # 1부터 10까지 수에서 10개로 쪼개는 range
```

```
array([1.          , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.          ])
```

```
1
```

NumPy ndarray : ndarray의 dtype

- dtype
 - ndarray가 메모리에 있는 특정 데이터를 해석하기 위해 필요한 정보(또는 메타데이터)를 담고 있는 특수한 객체
 - C 언어 등으로 작성된 코드와 쉽게 연동 가능

```
1 array_01 = np.array([1,2,3,4,5], dtype=np.float64)
2 array_02 = np.array([1,2,3,4,5], dtype=np.int32)
```

```
1 array_01.dtype
```

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

```
1 array_02.dtype
```

```
dtype('int32')
```

- numpy data type →

Type	Type Code	Description
int8, uint8	i1, u1	Signed and unsigned 8-bit (1 byte) integer types
int16, uint16	i2, u2	Signed and unsigned 16-bit integer types
int32, uint32	i4, u4	Signed and unsigned 32-bit integer types
int64, uint64	i8, u8	Signed and unsigned 32-bit integer types
float16	f2	Half-precision floating point
float32	f4 or f	Standard single-precision floating point. Compatible with C float
float64, float128	f8 or d	Standard double-precision floating point. Compatible with C double and Python floatobject
float128	f16 or g	Extended-precision floating point
complex64, complex128, complex256	c8, c16, c32	Complex numbers represented by two 32, 64, or 128 floats, respectively
bool	?	Boolean type storing True and False values
object	O	Python object type
string_	S	Fixed-length string type (1 byte per character). For example, to create a string dtype with length 10, use 'S10'.
unicode_	U	Fixed-length unicode type (number of bytes platform specific). Same specification semantics as string_ (e.g. 'U10').

NumPy ndarray : ndarray의 dtype

- astype 메소드

- 명시적 타입 변환 (캐스팅)
- 정수형 → 부동 소수점형

```
1 int_array = np.array([1,2,3,4,5])
2 int_array.dtype
```

```
dtype('int32')
```

```
1 float_array = int_array.astype(np.float64)
2 float_array.dtype
```

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

- 부동소수점형 → 정수형 (소수점 아래 자리 버짐)

```
1 float_array = np.array([1.2, 2.5, 3.4, 5.6, 7.8, 9.0])
2 float_array
```

```
array([1.2, 2.5, 3. , 4. , 5. , 6. , 7. , 8. , 9. ])
```

```
1 float_array.astype(np.int32)
```

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

NumPy ndarray : ndarray의 dtype

- 숫자 형식 문자열 → 숫자

※ 주의: numpy에서 문자열 데이터는 고정 크기를 가지며 별다른 경고를 출력하지 않고 입력을 임의로 잘라낼 수 있음

```
1 numeric_string = np.array(['1.25', '-9.6', '42'], dtype=np.string_)
2 numeric_string.astype(np.float64) # astype(float)
```

```
array([ 1.25, -9.6 , 42. ])
```

- 다른 배열의 dtype 속성 이용

※ 주의: astype을 호출하면 새로운 dtype이 이전 dtype과 동일해도 항상 새로운 배열을 생성(데이터를 복사)

```
1 int_array = np.arange(10)
2 int_array
```

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

```
1 calibers = np.array([.22, .270, .356, .380, .44, .50], dtype=np.float64)
2 int_array.astype(calibers.dtype)
```

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

- 축약 코드 사용 가능

```
1 empty_unit32 = np.empty(10, dtype='u4')
2 empty_unit32
```

```
array([ 375165360,          339,        3801156,       7143516,      7209057,
       6750305,      3014757,    7929968, 2058878976, 2059041465],
      dtype=uint32)
```

NumPy ndarray : 배열의 산술 연산

- 벡터화 : for 문을 사용하지 않고 데이터를 일괄 처리 가능
 - 같은 크기의 배열 간의 산술 연산은 배열의 각 원소 단위로 적용
 - 크기가 다른 배열 간의 연산 : 브로드캐스팅(broadcasting)

```
1 a_array = np.array([[1., 2., 3.], [4., 5., 6.]])  
2 a_array # 2 by 3 matrix
```

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

```
1 a_array * a_array
```

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

```
1 a_array - a_array
```

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

```
1 1/ a_array
```

```
array([[1.          , 0.5          , 0.33333333],  
       [0.25         , 0.2          , 0.16666667]])
```

```
1 a_array ** 0.5
```

```
array([[1.          , 1.41421356, 1.73205081],  
       [2.          , 2.23606798, 2.44948974]])
```

```
1 b_array = np.array([[0., 4., 1.], [7., 2., 12.]])  
2 b_array > a_array
```

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

NumPy ndarray : 색인 및 슬라이싱

- 조작 시 데이터 복사가 아니라 원본 배열에 그대로 반영됨
- 대용량 데이터 처리 염두 (복사 시에는 copy 함수 사용)

```
1 array_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
2 array_2d # 3 by 3 matrix
```

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

```
1 array_2d[2] # 3rd index
```

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

```
1 array_2d[0][2] # 0th row 2nd col
```

```
3
```

```
1 array_2d[0, 2] # 0th row 2nd col
```

```
3
```

```
1 array_2d[ :2] # from 0th to 1st
```

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

```
1 array_2d[ :2, 1: ] # 0,1 row, 1st, 2nd col
```

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

```
1 array_2d[1, :2] # 1st row, 0,1 col
```

```
array([4, 5])
```

```
1 array_2d[ :2, 2] # 0,1 row, 2nd col
```

```
array([3, 6])
```

```
1 array_2d[:, :1] # all row, 0th col
```

```
array([[1],  
       [4],  
       [7]])
```

```
1 array_2d[:2, 1:] # 0,1 row, 1, 2 col
```

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

```
1 array_2d[:2, 1:] = 0
```

```
array([[1, 0, 0],  
       [4, 0, 0],  
       [7, 8, 9]])
```

NumPy ndarray : Boolean Indexing

- Boolean 배열을 배열의 색인으로 사용

```
1 name_list = ['Bob', 'Joe', 'Will', 'Bob', 'Will', "joe", "Joe"]
2 names = np.array(name_list)
3 data = np.random.randn(7,4)
```

1 names # 7 elements

joe is not Joe

```
array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'joe', 'Joe'], dtype='<U4')
```

1 data # 7 by 4 matrix

```
array([[ 0.24225625,  2.28339039,  0.83670996,  0.12247309],
       [ 0.80877744, -0.72551652,  1.65148444, -1.24723941],
       [-0.08816675,  0.4060429 ,  1.27812113, -0.01765826],
       [ 0.71457754, -0.50468375,  0.60981897, -0.22058279],
       [-0.08531094,  0.22222095,  0.65656547,  0.12164069],
       [ 2.39122459,  0.35761168,  2.25078093,  0.52452568],
       [-1.25806627, -1.57186624, -0.57212614,  0.73847539]])
```

1 names == 'Bob' # 2 matches in the names array

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

1 data[names == 'Bob'] # 2 rows extract(0th and 3rd row)

```
array([[ 0.24225625,  2.28339039,  0.83670996,  0.12247309],
       [ 0.71457754, -0.50468375,  0.60981897, -0.22058279]])
```

1 data[names == 'Bob', 2:] # 0,3 rows and from 2 to last col

```
array([[ 0.83670996,  0.12247309],
       [ 0.60981897, -0.22058279]])
```

NumPy ndarray : Boolean Indexing

```
1 data[names == 'Bob', 3] # 0,3 rows and 3rd col only
```

```
array([ 0.12247309, -0.22058279])
```

```
1 data[~(names == 'Bob')] # 0,3 row 제외한 모든 rows
```

```
array([[ 0.80877744, -0.72551652,  1.65148444, -1.24723941],
       [-0.08816675,  0.4060429 ,  1.27812113, -0.01765826],
       [-0.08531094,  0.22222095,  0.65656547,  0.12164069],
       [ 2.39122459,  0.35761168,  2.25078093,  0.52452568],
       [-1.25806627, -1.57186624, -0.57212614,  0.73847539]])
```

```
1 data[data < 0] = 0 # if data < 0, then data = 0
2 data
```

```
array([[0.24225625, 2.28339039, 0.83670996, 0.12247309],
       [0.80877744, 0.          , 1.65148444, 0.          ],
       [0.          , 0.4060429 , 1.27812113, 0.          ],
       [0.71457754, 0.          , 0.60981897, 0.          ],
       [0.          , 0.22222095, 0.65656547, 0.12164069],
       [2.39122459, 0.35761168, 2.25078093, 0.52452568],
       [0.          , 0.          , 0.          , 0.73847539]])
```

```
1 data[names != 'Joe'] = 7 # if data != 'Joe', then data = 7
2 data
```

```
array([[7.          , 7.          , 7.          , 7.          , 7.          ],
       [0.80877744, 0.          , 1.65148444, 0.          , 0.          ],
       [7.          , 7.          , 7.          , 7.          , 7.          ],
       [7.          , 7.          , 7.          , 7.          , 7.          ],
       [7.          , 7.          , 7.          , 7.          , 7.          ],
       [7.          , 7.          , 7.          , 7.          , 7.          ],
       [0.          , 0.          , 0.          , 0.          , 0.73847539]])
```

NumPy ndarray : Fancy Indexing

- 정수 배열을 사용한 indexing

```
1 array_x = np.empty((8, 4)) # 8 by 4 matrix
2 for x in range(8): array_x[x] = x
3 array_x
```

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

```
1 array_x[[4, 3, 0, 6]] # 4,3,0,6 rows extract
```

```
array([[4., 4., 4., 4.],
       [3., 3., 3., 3.],
       [0., 0., 0., 0.],
       [6., 6., 6., 6.]])
```

```
1 array_x[[-4, -5, -8, -2]] # 4,3,0,6 rows extract
```

```
array([[4., 4., 4., 4.],
       [3., 3., 3., 3.],
       [0., 0., 0., 0.],
       [6., 6., 6., 6.]])
```

NumPy ndarray : Fancy Indexing

- 다차원 배열의 fancy indexing 결과는 항상 1차원

```
1 array_f = np.arange(32).reshape(8, 4) # 8 by 4 matrix
2 array_f
```

```
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, 30, 31]])
```

```
1 array_f[[1, 3, 5, 7], [0, 1, 2, 3]] # row, col diagnostic
array([ 4, 13, 22, 31])
```

```
1 array_f[[1, 3, 5, 7]] [:, [0, 1, 2]] # 1,3,5,7 rows, 0,1,2 cols
array([[ 4,  5,  6],
       [12, 13, 14],
       [20, 21, 22],
       [28, 29, 30]])
```

```
1 array_f[[1, 3, 5, 7]] # 1,3,5,6 rows, all cols extract
array([[ 4,  5,  6,  7],
       [12, 13, 14, 15],
       [20, 21, 22, 23],
       [28, 29, 30, 31]])
```

- 행렬의 low와 column에 대응하는 값 선택되길 기대
- fancy indexing은 slicing과 달리 선택된 데이터를 새로운 배열로 복사

NumPy ndarray : 배열 전치와 축 바꾸기

- ◆ **reshape** : 내부 데이터 보존한 채 형태 변경
 - 숫자는 -1 사용 가능. 계산되어 사용

```
1 array_f[[1, 3, 5, 7]] # 1,3,5,6 rows, all cols extract
```

```
array([[ 4,  5,  6,  7],  
       [12, 13, 14, 15],  
       [20, 21, 22, 23],  
       [28, 29, 30, 31]])
```

```
1 array_r = np.arange(15).reshape(3, 5)  
2 array_r
```

```
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14]])
```

```
1 array_r.reshape(5, -1) # -1 means 알아서 계산하라
```

```
array([[ 0,  1,  2],  
       [ 3,  4,  5],  
       [ 6,  7,  8],  
       [ 9, 10, 11],  
       [12, 13, 14]])
```

```
1 array_r.reshape(4, -1) # 4 by x에 맞는 정수가 없음
```

```
-----  
ValueError
```

```
Traceback (most recent call last)
```

```
<ipython-input-223-bd70aa29438e> in <module>
```

```
----> 1 array_r.reshape(4, -1) # 4 by x에 맞는 정수가 없음
```

```
ValueError: cannot reshape array of size 15 into shape (4,newaxis)
```

NumPy ndarray : 배열 전치와 축 바꾸기

- T: 축을 뒤바꾸는 간단한 전치

```
1 array_r # 3 by 4 matrix
```

```
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14]])
```

```
1 array_r.T # 4 by 3 transpose
```

```
array([[ 0,  5, 10],  
       [ 1,  6, 11],  
       [ 2,  7, 12],  
       [ 3,  8, 13],  
       [ 4,  9, 14]])
```

NumPy ndarray : Universal 함수

Function	Description
abs, fabs	Compute the absolute value element-wise for integer, floating-point, or complex values
sqrt	Compute the square root of each element (equivalent to <code>arr ** 0.5</code>)
square	Compute the square of each element (equivalent to <code>arr ** 2</code>)
exp	Compute the exponent e^x of each element
log, log10, log2, log1p	Natural logarithm (base e), log base 10, log base 2, and $\log(1 + x)$, respectively
sign	Compute the sign of each element: 1 (positive), 0 (zero), or -1 (negative)
ceil	Compute the ceiling of each element (i.e., the smallest integer greater than or equal to that number)
floor	Compute the floor of each element (i.e., the largest integer less than or equal to each element)
rint	Round elements to the nearest integer, preserving the <code>dtype</code>
modf	Return fractional and integral parts of array as a separate array
isnan	Return boolean array indicating whether each value is NaN (Not a Number)
isfinite, isinf	Return boolean array indicating whether each element is finite (<code>non-inf</code> , non-NaN) or infinite, respectively
cos, cosh, sin, sinh, tan, tanh	Regular and hyperbolic trigonometric functions
arccos, arccosh, arcsin, arcsinh, arctan, arctanh	Inverse trigonometric functions
logical_not	Compute truth value of <code>not x</code> element-wise (equivalent to <code>~arr</code>).

NumPy ndarray : Universal 함수

Function	Description
<code>add</code>	Add corresponding elements in arrays
<code>subtract</code>	Subtract elements in second array from first array
<code>multiply</code>	Multiply array elements
<code>divide, floor_divide</code>	Divide or floor divide (truncating the remainder)
<code>power</code>	Raise elements in first array to powers indicated in second array
<code>maximum, fmax</code>	Element-wise maximum; <code>fmax</code> ignores NaN
<code>minimum, fmin</code>	Element-wise minimum; <code>fmin</code> ignores NaN
<code>mod</code>	Element-wise modulus (remainder of division)
<code>copysign</code>	Copy sign of values in second argument to values in first argument

- `dir(np)` : numpy의 속성(변수)과 함수 알아보기

Pandas (Python Data Analysis Library)

- 데이터분석 라이브러리
- 특징
 - 데이터의 수정/가공 및 분석이 용이
 - 데이터 가공을 위한 수많은 함수 지원
 - numpy 기반으로 빠른 데이터 처리
 - 행과 열로 이루어진 데이터 객체



Pandas: Series

- 1차원 자료구조
- 한 개의 열(column)을 나타내는 자료
- python 리스트를 간직한 오브젝트
- 리스트를 적용하여 바로 시리즈 생성

```
In [1]: 1 import pandas as pd  
2 series_1 = pd.Series([1, 3, 5, 7, 9])  
3 series_1
```



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

Pandas: Index에 이름 부여한 Series

- pd.Series ([리스트 value], index = [인덱스 리스트])

```
In [2]: 1 import pandas as pd  
2 series = pd.Series( [ 58, 99, 68, 92, 100 ],  
3                      index=['exam1','exam2','exam3','exam4','exam5'])  
4 series
```

```
Out[2]: exam1      58  
exam2      99  
exam3      68  
exam4      92  
exam5     100  
dtype: int64
```

Pandas: NumPy를 활용한 Series 생성

In [3]:

```
1 import pandas as pd
2 import numpy as np
3 my_arr = np.array([1,3,5,7,9])
4 series = pd.Series(my_arr)
5 series
```

Out [3]:

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

Pandas: DataFrame

- 2차원 배열/구조
- 엑셀 시트와 같은 테이블 형태의 자료
- DataFrame은 1개 이상의 Series로 구성된 자료 구조

```
In [2]: 1 import pandas as pd  
2 series_1 = pd.Series ([ 1, 3, 5, 7, 9 ] )  
3 series_2 = pd.Series ([ 11, 13, 15, 17, 19] )  
4 pd.DataFrame ( data = dict ( single = series_1, tens = series_2 ) )
```

Out[2]:

	single	tens
0	1	11
1	3	13
2	5	15
3	7	17
4	9	19

Pandas: 딕셔너리 활용 DataFrame 생성

- 딕셔너리 자료형을 활용하여 DataFrame 값 설정
 - 딕셔너리의 Key 값을 column명으로 사용

```
In [3]: 1 import pandas as pd
2 temp_seoul = [20, 25, 23, 24, 28]
3 temp_daejeon = [22, 27, 25, 23, 29]
4 temp_kwangju = [22, 23, 24, 25, 28]
5 temp_pusan = [24, 257, 26, 26, 27]
6 temperatures = {'Seoul' : temp_seoul,
7                  'Daejeon' : temp_daejeon,
8                  'Kwangju' : temp_kwangju,
9                  'Pusan' : temp_pusan}
10 pd.DataFrame(temperatures)
```

Out [3]:

	Seoul	Daejeon	Kwangju	Pusan
0	20	22	22	24
1	25	27	23	257
2	23	25	24	26
3	24	23	25	26
4	28	29	28	27

Pandas: 파일 활용 DataFrame 생성 (1)

- read_csv() 함수로 파일을 DataFrame으로 작성

```
1 import pandas as pd  
2 df = pd.read_csv('c:/Users/SSEN-HAN/PaperWriting/Data_Frame.csv')  
3 df
```

	FIELD	CASES	CIVIL	CRIMINAL
0	economics	2435	2142	293
1	tax	5243	3899	1344
2	work	3214	3101	113
3	social	4112	2019	2093
4	education	2131	1993	138
5	environment	4192	4081	111

	A	B	C	D
1	FIELD	CASES	CIVIL	CRIMINAL
2	economics	2435	2142	293
3	tax	5243	3899	1344
4	work	3214	3101	113
5	social	4112	2019	2093
6	education	2131	1993	138
7	environment	4192	4081	111

Data_Frame.csv

Pandas: 파일 활용 DataFrame 생성 (2)

- 콤마로 구분된 text 파일로 DataFrame으로 작성

```
1 import pandas as pd  
2 df = pd.read_csv('c:/Users/SSEN-HAN/PaperWriting/Data_Frame.txt')  
3 df
```

	FIELD	CASES	CIVIL	CRIMINAL
0	economics	2435	2142	293
1	tax	5243	3899	1344
2	work	3214	3101	113
3	social	4112	2019	2093
4	education	2131	1993	138
5	environment	4192	4081	111

Data_Frame.txt



Data_Frame - 메모장	
파일(F)	편집(E)
서식(O)	보기(V)
도움말	
FIELD, CASES, CIVIL, CRIMINAL	
economics, 2435, 2142, 293	
tax, 5243, 3899, 1344	
work, 3214, 3101, 113	
social, 4112, 2019, 2093	
education, 2131, 1993, 138	
environment, 4192, 4081, 111	

Pandas: 파일 활용 DataFrame 생성 (3)

- csv 파일에 header가 없는 경우

```
1 import pandas as pd  
2 df = pd.read_csv('c:/Users/SSEN-HAN/PaperWriting/Data_Frame_NoHeader.csv',  
3                   header = None)  
4 df.head()  
  
          0      1      2      3  
0 economics  2435  2142  293  
1 tax        5243  3899  1344  
2 work       3214  3101  113  
3 social     4112  2019  2093  
4 education   2131  1993  138
```

	A	B	C	D
1	conomics	2435	2142	293
2	tax	5243	3899	1344
3	work	3214	3101	113
4	social	4112	2019	2093
5	education	2131	1993	138
6	environme	4192	4081	111

Data_Frame_NoHeader.csv

Pandas: DataFrame 컬럼 Header 작성법

- df.columns 활용

```
1 df.columns = ['법 분야', '판례', '민사', '형사']
2 df.head()
```

법 분야 판례 민사 형사

0	economics	2435	2142	293
1	tax	5243	3899	1344
2	work	3214	3101	113
3	social	4112	2019	2093
4	education	2131	1993	138

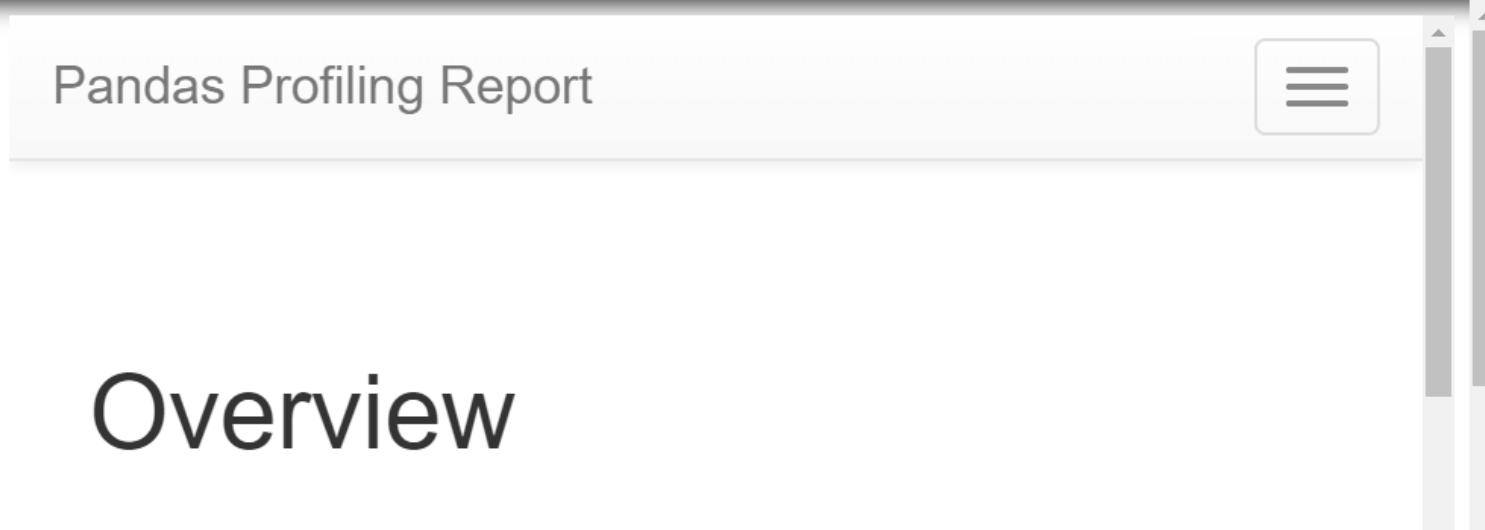
Pandas: Profiling (1)

- DataFrame 자료에 대한 분석

```
In [24]: 1 import pandas as pd  
2 from pandas_profiling import ProfileReport  
3 df = pd.read_csv('c:/Users/SSEN-HAN/PaperWriting/Data_Frame.csv')  
4 pr = ProfileReport(df)  
5 pr
```

Pandas Profiling Report

Overview



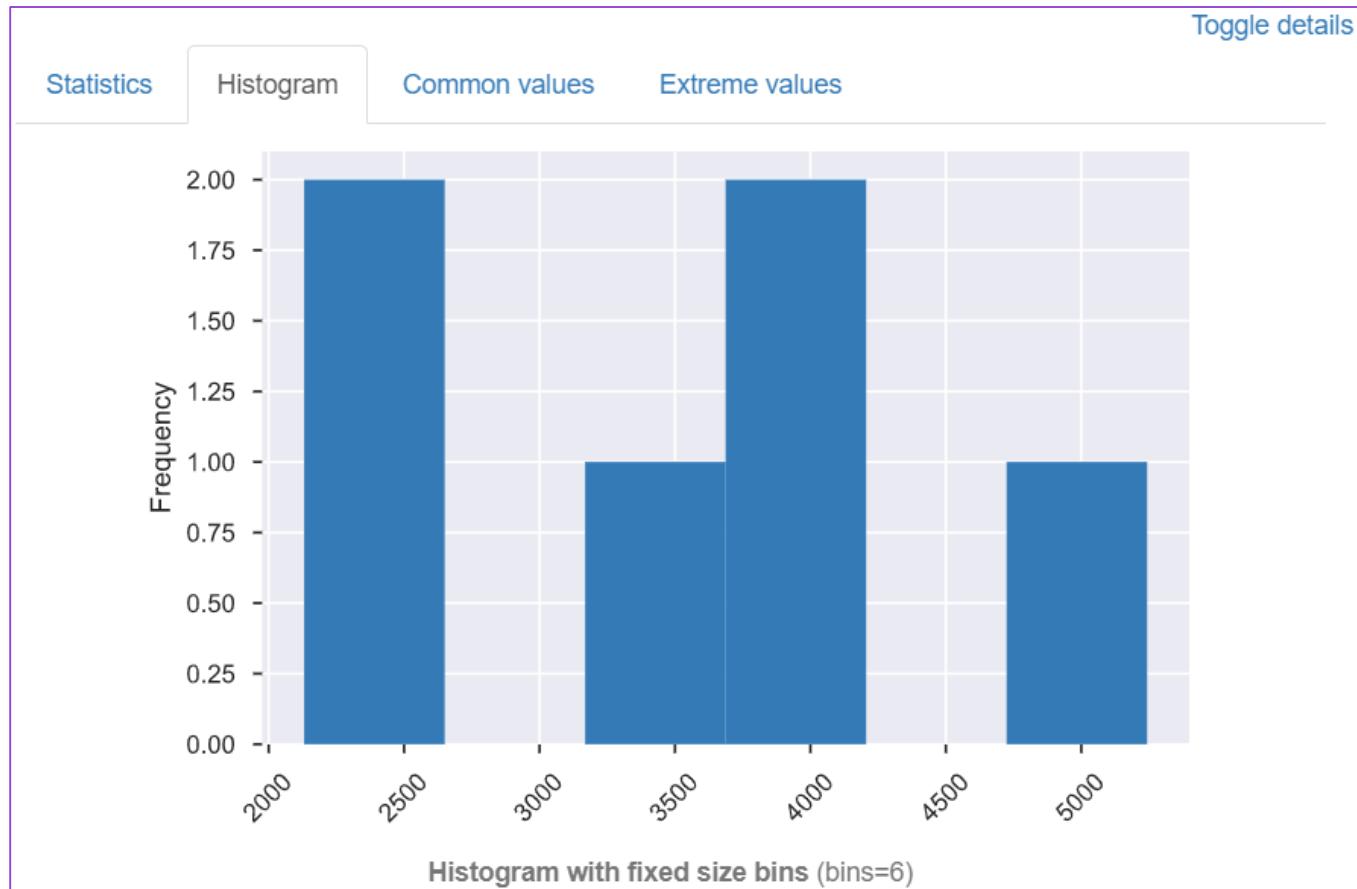
Pandas: Profiling (2)

- 통계자료 Profiling



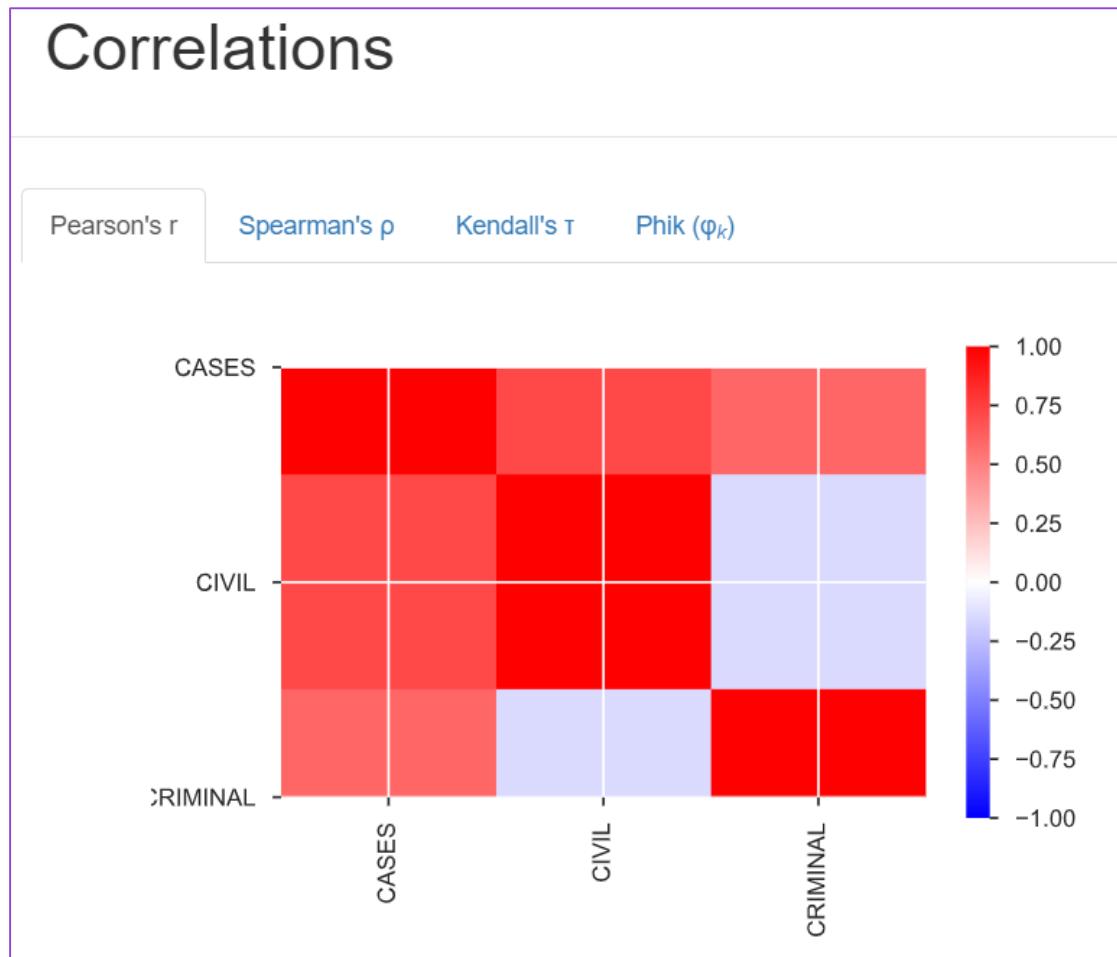
Pandas: Profiling (3)

- Histogram



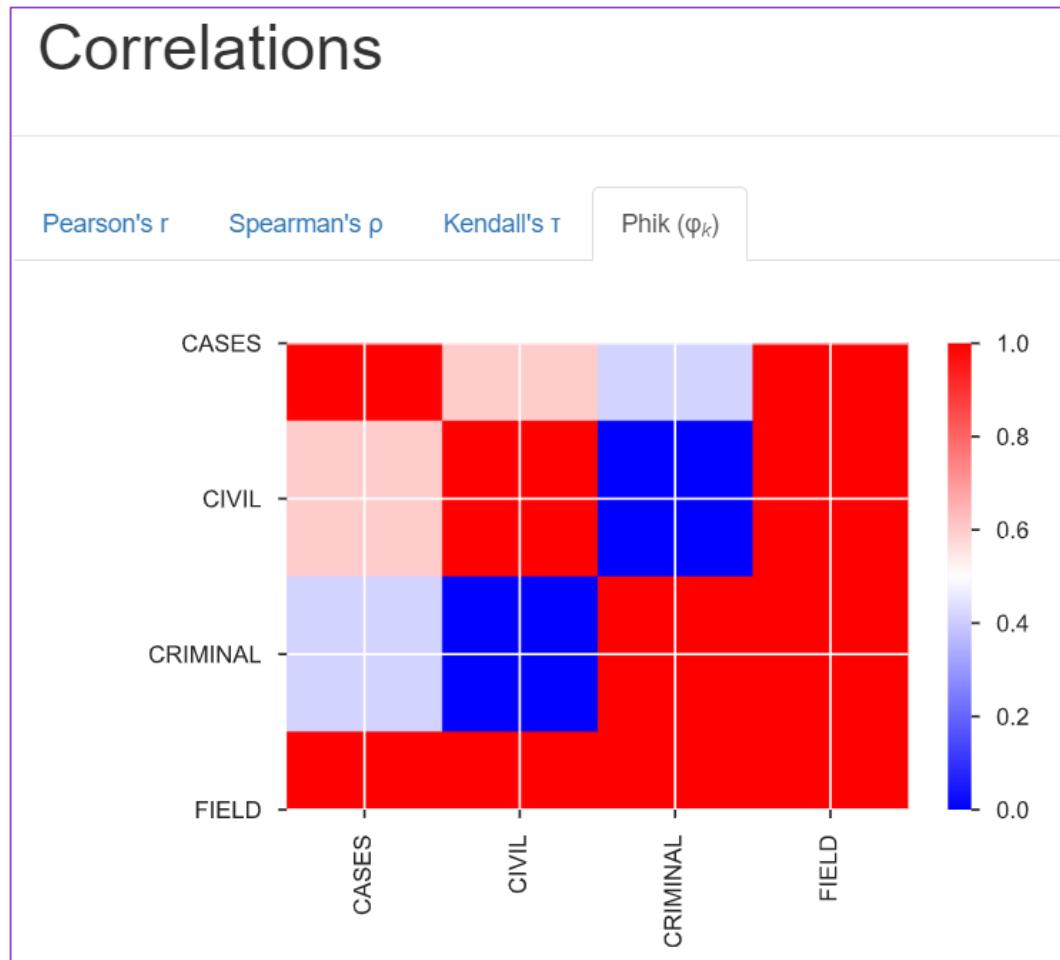
Pandas: Profiling (4)

- Pearson 상관관계 분석



Pandas: Profiling (5)

- Phik 상관관계 분석



Pandas: 슬라이싱 활용 row 자료 선택

```
In [25]: 1 import pandas as pd  
2 df = pd.read_csv('c:/Users/SSEN-HAN/PaperWriting/Data_Frame.csv')  
3 df
```

Out [25] :

	FIELD	CASES	CIVIL	CRIMINAL
0	economics	2435	2142	293
1	tax	5243	3899	1344
2	work	3214	3101	113
3	social	4112	2019	2093
4	education	2131	1993	138
5	environment	4192	4081	111

```
In [26]: 1 df[1:3]
```

Out [26] :

	FIELD	CASES	CIVIL	CRIMINAL
1	tax	5243	3899	1344
2	work	3214	3101	113

Pandas: 순차적이지 않은 row 선택

- df.loc [[원하는 row index]]

```
In [28]: 1 df.loc[[1,3,5]]
```

```
Out[28]:
```

	FIELD	CASES	CIVIL	CRIMINAL
1	tax	5243	3899	1344
3	social	4112	2019	2093
5	environment	4192	4081	111

Pandas: column 값을 기준으로 row 선택

- df [df.column 조건식]

```
In [30]: 1 df_filtered = df[df.CRIMINAL < 120]  
         2 df_filtered
```

Out [30]:

	FIELD	CASES	CIVIL	CRIMINAL
2	work	3214	3101	113
5	environment	4192	4081	111

Pandas: query 사용 row 선택

- df.query ('조건식')

```
In [31]: 1 df_filtered = df.query('CIVIL > 3000')  
2 df_filtered
```

Out[31]:

	FIELD	CASES	CIVIL	CRIMINAL
1	tax	5243	3899	1344
2	work	3214	3101	113
5	environment	4192	4081	111

Pandas: 논리 연산자 활용 row 선택

- df [관계_연산식1 & 관계_연산식2]

```
In [32]: 1 df_filtered = df[(df.CASES > 3000) & (df.CRIMINAL < 120)]  
2 df_filtered
```

Out [32]:

	FIELD	CASES	CIVIL	CRIMINAL
2	work	3214	3101	113
5	environment	4192	4081	111

Pandas: 행(row) 추가 연산

```
1 df2 = pd.DataFrame([[80, 70, 90, 80]],  
2                 columns = ['Test1','Test2','Project','Average'],  
3                 index=['Kang'])  
4 print("">>>> DataFrame 1\n", df1)  
5 print("">>>> DataFrame 2\n", df2)  
6 data = df1.append(df2)  
7 print ("">>>> 새로운 행(row) 추가 결과\n", data)
```

```
>>> DataFrame 1  
    Test1  Test2  Project  Average  
Han    89.0     77      90    85.33  
Kim    55.0     82      83    73.33  
Lee     NaN      65      67      NaN  
Park    70.0     56      87    71.00
```

```
>>> DataFrame 2  
    Test1  Test2  Project  Average  
Kang    80      70      90      80
```

```
>>> 새로운 행(row) 추가 결과  
    Test1  Test2  Project  Average  
Han    89.0     77      90    85.33  
Kim    55.0     82      83    73.33  
Lee     NaN      65      67      NaN  
Park    70.0     56      87    71.00  
Kang    80.0     70      90    80.00
```

Pandas: 행(row) 삭제 연산

- df.drop()

```
In [65]: 1 print(data)  
2 data.drop('Lee')
```

	Test1	Test2	Project	Average
Han	89.0	77	90	85.33
Kim	55.0	82	83	73.33
Lee	NaN	65	67	NaN
Park	70.0	56	87	71.00
Kang	80.0	70	90	80.00

Out [65]:

	Test1	Test2	Project	Average
Han	89.0	77	90	85.33
Kim	55.0	82	83	73.33
Park	70.0	56	87	71.00
Kang	80.0	70	90	80.00

Pandas: 슬라이싱 활용 column 자료 선택

- df.iloc [:, 시작열:끝열+1] # 모든 row 선택

```
In [34]: 1 df . i loc[ :, 0:2]
```

Out [34]:

	FIELD	CASES
0	economics	2435
1	tax	5243
2	work	3214
3	social	4112
4	education	2131
5	environment	4192

Pandas: 선택적 column 자료 선택

- df.iloc [:, [선택1, 선택열2]] # 모든 row 선택

```
In [35]: 1 df . i l o c [ : , [ 0 , 2 ] ]
```

```
Out [35]:
```

	FIELD	CIVIL
0	economics	2142
1	tax	3899
2	work	3101
3	social	2019
4	education	1993
5	environment	4081

Pandas: 컬럼 이름으로 필터링

- df [[원하는 column명 리스트]]

```
In [36]: 1 df_filtered = df[['FIELD', 'CRIMINAL']]  
2 df_filtered
```

Out [36]:

	FIELD	CRIMINAL
0	economics	293
1	tax	1344
2	work	113
3	social	2093
4	education	138
5	environment	111

Pandas: filter() 함수 활용

- df.filter (items = [원하는 column명 리스트])

```
In [38]: 1 df.filter(items=['CASES', 'CIVIL'])
```

Out [38]:

CASES CIVIL

0 2435 2142

1 5243 3899

2 3214 3101

3 4112 2019

4 2131 1993

5 4192 4081

Pandas: 원하는 글자를 포함한 column

- df.filter (like = '원하는 글자', axis=1)

```
In [39]: 1 df.filter(like='C',axis=1)
```

Out [39]:

	CASES	CIVIL	CRIMINAL
0	2435	2142	293
1	5243	3899	1344
2	3214	3101	113
3	4112	2019	2093
4	2131	1993	138
5	4192	4081	111

Pandas: 정규식 활용 column 선택

- df.filter (regex = '원하는 글자\$', axis = 1) #원하는 글자로 끝나는 column명 선택

```
In [42]: 1 df.filter(regex='L$',axis=1)
```

Out [42]:

	CIVIL	CRIMINAL
0	2142	293
1	3899	1344
2	3101	113
3	2019	2093
4	1993	138
5	4081	111

Pandas: 연산을 통한 column 추가

```
1 import pandas as pd
2 data = { 'Test1' : pd.Series([70, 55, 89],
3                             index=['Park', 'Kim', 'Han']),
4         'Test2' : pd.Series([56, 82, 77, 65],
5                             index=['Park', 'Kim', 'Han', 'Lee'])}
6 df1 = pd.DataFrame(data)
7 print (df1)
8 df1['Project'] = pd.Series([90, 83, 67, 87],
9                            index=['Han','Kim','Lee', 'Park'])
10 print ("="*20)
11 df1['Average'] = round((df1['Test1']+df1['Test2']+df1['Project'])/3, 2)
13 print (df1)
```

	Test1	Test2		
Han	89.0	77		
Kim	55.0	82		
Lee	NaN	65		
Park	70.0	56		
=====				
	Test1	Test2	Project	Average
Han	89.0	77	90	85.33
Kim	55.0	82	83	73.33
Lee	NaN	65	67	NaN
Park	70.0	56	87	71.00

Pandas: Column 삭제 (1)

- del 사용

```
1 df2 = df1  
2 del df2['Test2']  
3 print(df2)
```

	Test1	Project	Average
Han	89.0	90	85.33
Kim	55.0	83	73.33
Lee	NaN	67	NaN
Park	70.0	87	71.00

Pandas: Column 삭제 (2)

- pop() 사용

```
1 df2 = df1  
2 df2.pop("Project")  
3 print(df2)
```

	Test1	Average
Han	89.0	85.33
Kim	55.0	73.33
Lee	NaN	NaN
Park	70.0	71.00

```
1 print(df1)
```

	Test1	Average
Han	89.0	85.33
Kim	55.0	73.33
Lee	NaN	NaN
Park	70.0	71.00

이해가 안되요...

- 지켜주지 못한 df1...
- 해결 방법은 df.copy()

```
1 df2 = df1.copy() # copy df1 into df2 using copy()
2 print(df2)
3 #delete columns using del and pop methods
4 del df2['Test2']
5 df2.pop('Project')
6 print("== 열 삭제된 결과>>> \n", df2)
7 print("== 원래의 DataFrame>>> \n", df1)
```

	Test1	Test2	Project	Average
Han	89.0	77	90	85.33
Kim	55.0	82	83	73.33
Lee	NaN	65	67	NaN
Park	70.0	56	87	71.00

== 열 삭제된 결과>>>

	Test1	Average
Han	89.0	85.33
Kim	55.0	73.33
Lee	NaN	NaN
Park	70.0	71.00

== 원래의 DataFrame>>>

	Test1	Test2	Project	Average
Han	89.0	77	90	85.33
Kim	55.0	82	83	73.33
Lee	NaN	65	67	NaN
Park	70.0	56	87	71.00

DataFrame 분석

In [67]:

```
1 print(data)  
2 data.describe()
```

	Test1	Test2	Project	Average
Han	89.0	77	90	85.33
Kim	55.0	82	83	73.33
Lee	NaN	65	67	NaN
Park	70.0	56	87	71.00
Kang	80.0	70	90	80.00

Out [67]:

	Test1	Test2	Project	Average
count	4.000000	5.000000	5.000000	4.000000
mean	73.500000	70.000000	83.400000	77.415000
std	14.571662	10.173495	9.607289	6.510732
min	55.000000	56.000000	67.000000	71.000000
25%	66.250000	65.000000	83.000000	72.747500
50%	75.000000	70.000000	87.000000	76.665000
75%	82.250000	77.000000	90.000000	81.332500
max	89.000000	82.000000	90.000000	85.330000

Pandas: 특정 자료 분석

- Project 열(column) 자료를 확인하고 싶다면

```
In [68]: 1 data.Project.describe()  
Out[68]: count      5.000000  
          mean      83.400000  
          std       9.607289  
          min      67.000000  
          25%     83.000000  
          50%     87.000000  
          75%     90.000000  
          max      90.000000  
          Name: Project, dtype: float64
```

Pandas: 선택적 분석 – 수치 자료

In [74]:

```
1 import numpy as np  
2 print(data)  
3 data.describe(include=[np.number])
```

	Test1	Test2	Project	Average	Major
Han	89.0	77	90	85.33	CS
Kim	55.0	82	83	73.33	ECON
Lee	NaN	65	67	NaN	MATH
Park	70.0	56	87	71.00	CS
Kang	80.0	70	90	80.00	NaN

Out [74]:

	Test1	Test2	Project	Average
count	4.000000	5.000000	5.000000	4.000000
mean	73.500000	70.000000	83.400000	77.415000
std	14.571662	10.173495	9.607289	6.510732
min	55.000000	56.000000	67.000000	71.000000
25%	66.250000	65.000000	83.000000	72.747500
50%	75.000000	70.000000	87.000000	76.665000
75%	82.250000	77.000000	90.000000	81.332500
max	89.000000	82.000000	90.000000	85.330000

Pandas: 선택적 분석 – 문자 자료

In [75]:

```
1 print(data)
2 data.describe(include=[np.object])
```

	Test1	Test2	Project	Average	Major
Han	89.0	77	90	85.33	CS
Kim	55.0	82	83	73.33	ECON
Lee	NaN	65	67	NaN	MATH
Park	70.0	56	87	71.00	CS
Kang	80.0	70	90	80.00	NaN

Out [75]:

Major

count 4

unique 3

top CS

freq 2

Pandas: 조건 확인

In [80]:

```
1 print(data)
2 B_grade = data['Average'] >= 80
3 B_grade
```

	Test1	Test2	Project	Average	Major
Han	89.0	77	90	85.33	CS
Kim	55.0	82	83	73.33	ECON
Lee	NaN	65	67	NaN	MATH
Park	70.0	56	87	71.00	CS
Kang	80.0	70	90	80.00	NaN

Out [80]:

Han	True
Kim	False
Lee	False
Park	False
Kang	True

Name: Average, dtype: bool

Pandas: 열(column) 기준 정렬

In [83]:

```
1 print(data)
2 data.sort_values(by='Average', ascending=False)
```

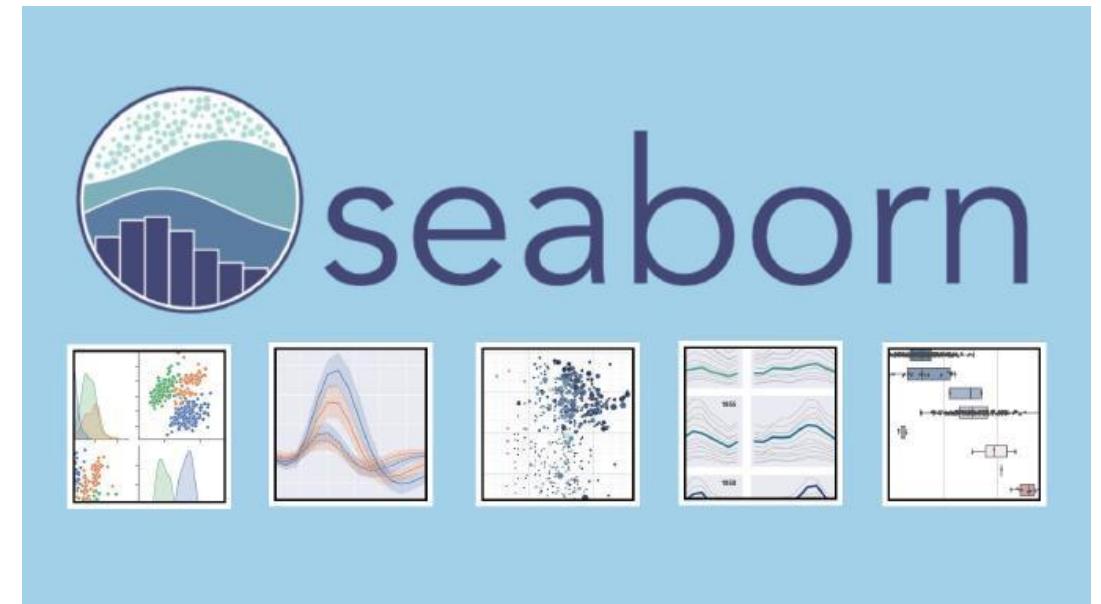
	Test1	Test2	Project	Average	Major
Han	89.0	77	90	85.33	CS
Kim	55.0	82	83	73.33	ECON
Lee	NaN	65	67	NaN	MATH
Park	70.0	56	87	71.00	CS
Kang	80.0	70	90	80.00	NaN

Out [83]:

	Test1	Test2	Project	Average	Major
Han	89.0	77	90	85.33	CS
Kang	80.0	70	90	80.00	NaN
Kim	55.0	82	83	73.33	ECON
Park	70.0	56	87	71.00	CS
Lee	NaN	65	67	NaN	MATH



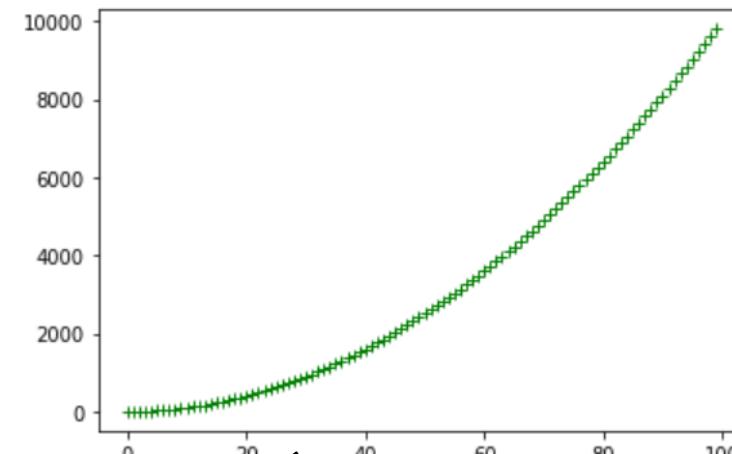
Version 3.2.1



데이터 시각화 - 간단한 그래프 그리기

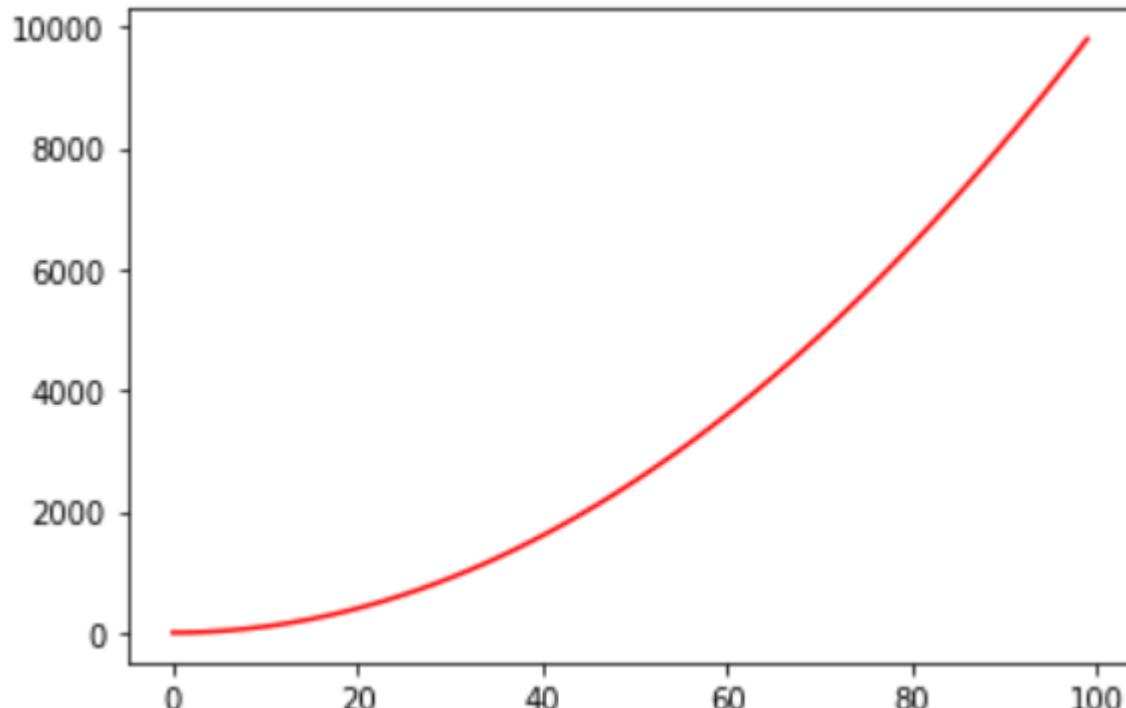
- matplotlib.pyplot 모듈
- plot() 함수 : 값을 서로 연결하여 라인 형태의 그래프 그림
 - 기본 포맷 : 파란색 선
 - 'r' : 빨간색 선, plt.plot(x, y, 'r')
 - 'ro' : 빨간색 원 마커 모양, plt.plot(x, y, 'ro')
 - 'r--' : 빨간색 대쉬라인, plt.plot(x, y, 'r--')
 - 'bs' : 파란색 사각형, plt.plot(x, y, 'bs')
 - 'g+' : 녹색 십자모양, plt.plot(x, y, 'g+')

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3
4 x = [i for i in range(100)]
5 y = [i**2 for i in x]
6
7 plt.plot(x, y, 'g+')
```

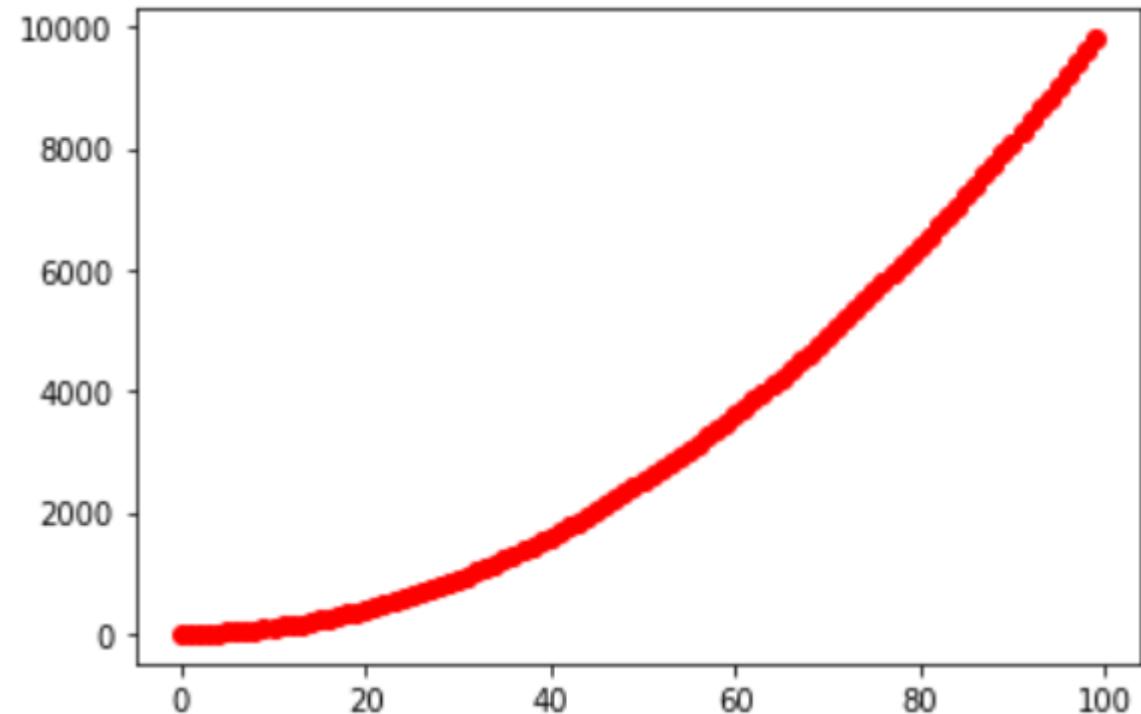


데이터 시각화 - 간단한 그래프 그리기

```
1 plt.plot(x, y, 'r')
```



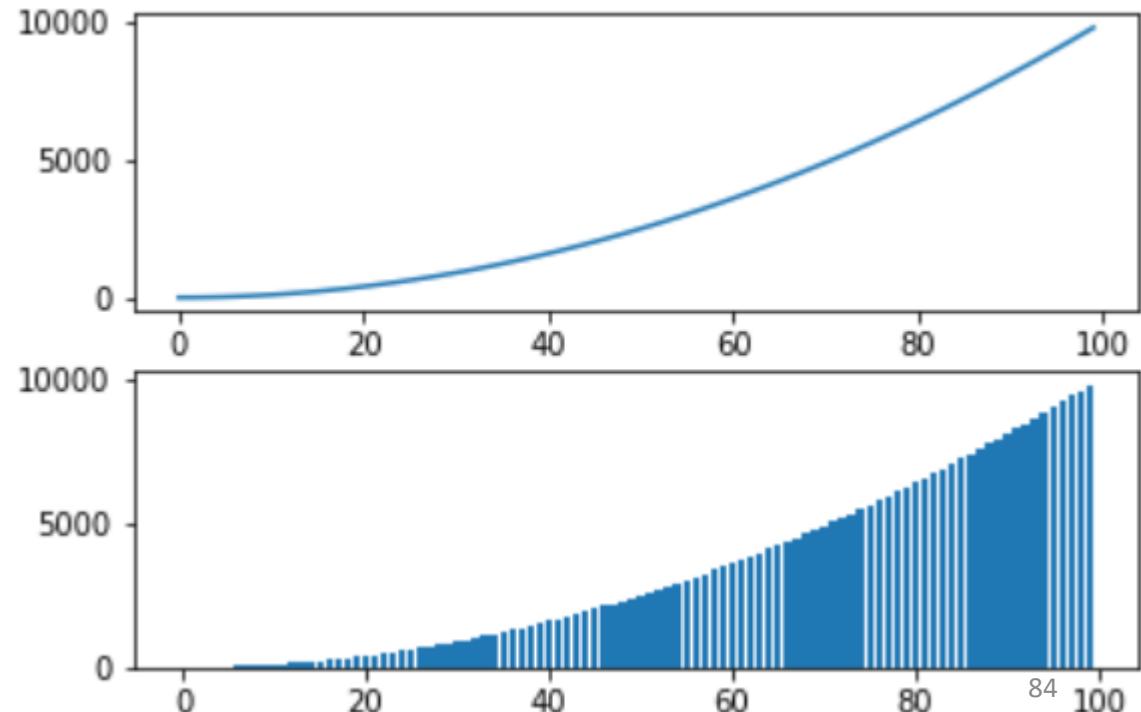
```
1 plt.plot(x, y, 'ro')
```



데이터 시각화 - 한 화면에 여러 개 그래프

- figure() : figure 객체 생성
- add_subplot(위치 및 개수) : 생성된 figure 객체에 subplot 추가
 - add_subplot(2, 1, 1) : 2x1 형태의 subplot, 두 개의 subplot 중 첫 번째
 - add_subplot(2, 1, 2) : 2x1 형태의 subplot, 두 개의 subplot 중 두 번째
- add_subplot() 함수 호출 시 AxesSubplot 객체 생성됨
- 생성된 subplot 객체를 그래프 drawing 함수와 연결

```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3
4 x = [i for i in range(100)]
5 y = [i**2 for i in x]
6
7 fig = plt.figure() # instance a figure object
8 axe_01 = fig.add_subplot(2,1,1) # 1st subplot in 2x1
9 axe_02 = fig.add_subplot(2,1,2) # 2nd subplot in 2x2
10
11 axe_01.plot(x, y)           # plot a line graph
12 axe_02.bar(x, y)           # plot a bar graph
```

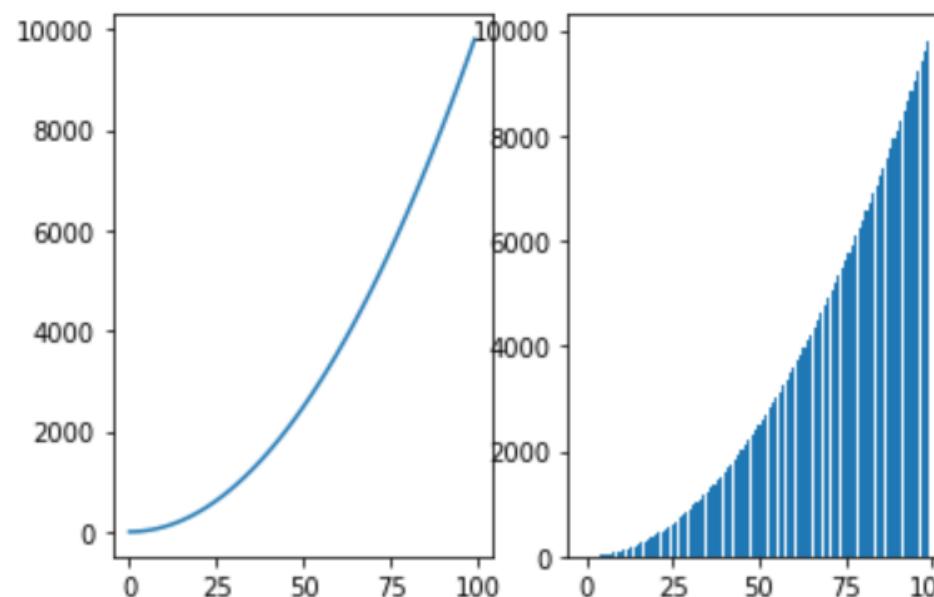


데이터 시각화 - 한 화면에 여러 개 그래프

- 1x2 형태 그래프들
 - add_subplot(1, 2, 1) : 1x2 형태의 subplot, 두 개의 subplot 중 첫 번째
 - add_subplot(1, 2, 2) : 1x2 형태의 subplot, 두 개의 subplot 중 두 번째

```
1 fig = plt.figure() # instance a figure object
2 axe_01 = fig.add_subplot(1,2,1) # 1st subplot in 2x1
3 axe_02 = fig.add_subplot(1,2,2) # 2nd subplot in 2x2
4
5 axe_01.plot(x, y)           # plot a line graph
6 axe_02.bar(x, y)           # plot a bar graph
```

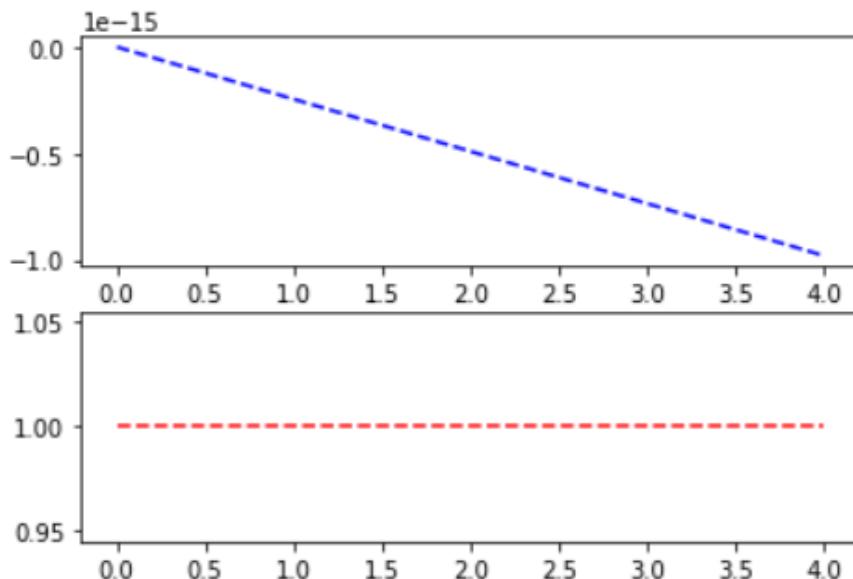
<BarContainer object of 100 artists>



데이터 시각화 - sine, cosine 그래프

```
1 import math  
2  
3 x = [i for i in range(5)]  
4 sin_y = [math.sin(2*math.pi*i) for i in x]  
5 cos_y = [math.cos(2*math.pi*i) for i in x]  
6  
7 fig = plt.figure() # instance a figure object  
8 axe_01 = fig.add_subplot(2,1,1) # 1st subplot in 2x1  
9 axe_02 = fig.add_subplot(2,1,2) # 2nd subplot in 2x2  
10  
11 axe_01.plot(x, sin_y, 'b--') # plot a sin graph  
12 axe_02.plot(x, cos_y, 'r--') # plot a cos graph
```

```
[<matplotlib.lines.Line2D at 0x1531b92aa88>]
```



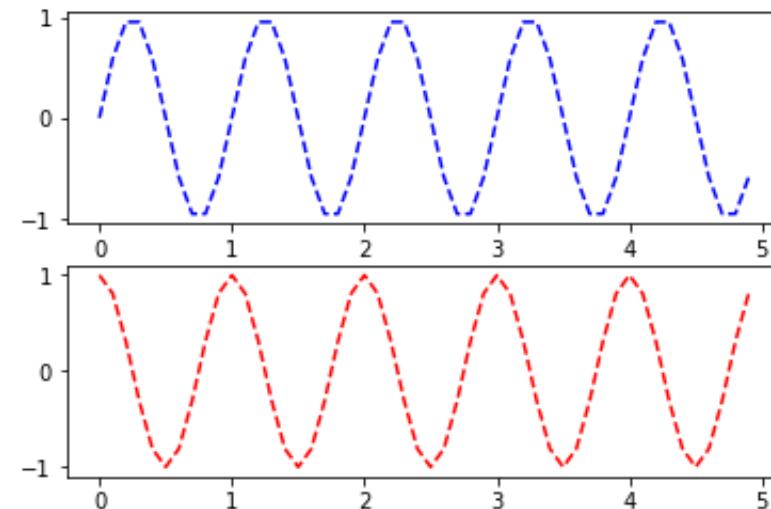
- 파이썬의 기본 math 모듈 사용

정수 0~5 사이 범위의 값에서 range(0,5)로는 시간의 값을 촘촘하게 만들 수 없음

데이터 시각화 - numpy 모듈

- numpy : Numerical Python의 약자. 행렬 연산이나 수치 계산에 자주 사용
- numpy.arange() 함수 : 실수 단위로도 step 값 가능
 - arange(0.0, 5.0, 0.1) : 0.0 ~ 5.0 사이에서 0.1 간격으로 값 생성
- numpy.pi : π 값

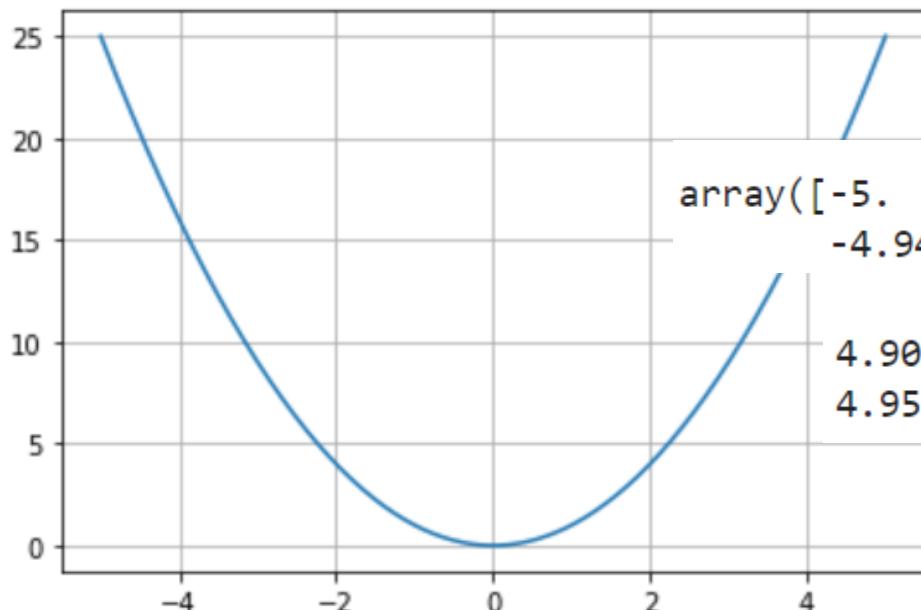
```
1 import numpy as np
2
3 x = np.arange(0., 5., 0.1)
4 sin_y = [np.sin(2*np.pi*i) for i in x]
5 cos_y = [np.cos(2*np.pi*i) for i in x]
6
7 fig = plt.figure() # instance a figure object
8 axe_01 = fig.add_subplot(2,1,1) # 1st subplot in 2x1
9 axe_02 = fig.add_subplot(2,1,2) # 2nd subplot in 2x2
10
11 axe_01.plot(x, sin_y, 'b--')    # plot a sin graph
12 axe_02.plot(x, cos_y, 'r--')    # plot a cos graph
```



데이터 시각화 - numpy 기반 값 생성

- `numpy.linspace(start, end, num-points)` : 시작점과 끝점을 균일 간격으로 나눈 값 생성
- `numpy.power(x, y)` : x의 y제곱 값

```
1 x = np.linspace(-5, 5, 1_000) # ndarray
2 y = np.power(x, 2)             # ndarray
3
4 plt.plot(x, y)   # plot a line with x, y
5 plt.grid()       # show a grid
6 plt.show()        # show a whole graph
```



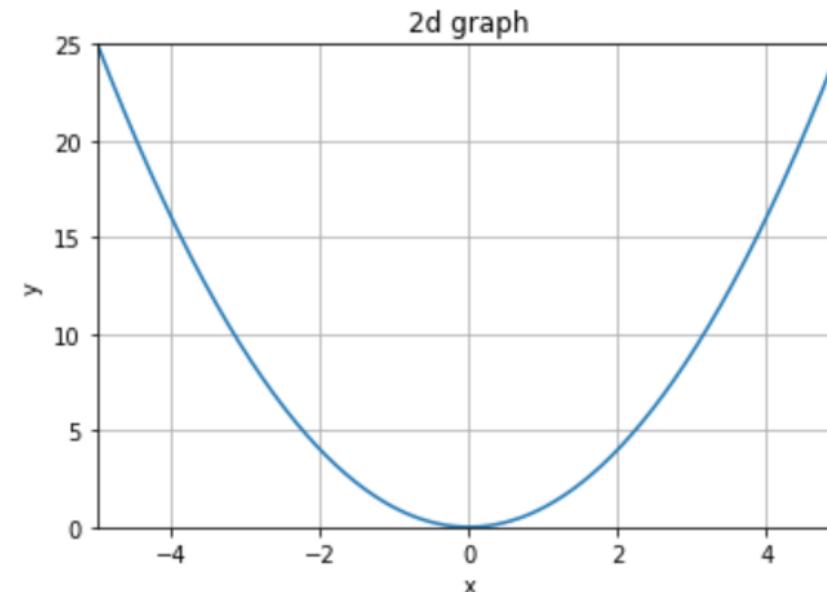
array([-5.0, -4.98998999, -4.97997998, -4.96996997, -4.95995996,
 -4.94994995, -4.93993994, -4.92992993, -4.91991992, -4.90990991,
 4.90990991, 4.91991992, 4.92992993, 4.93993994, 4.94994995,
 4.95995996, 4.96996997, 4.97997998, 4.98998999, 5.0])



데이터 시각화 - 그래프 라벨 및 범례 표시

- title(문자열) : 그래프 제목
- xlabel(문자열) : x축 라벨
- ylabel(문자열) : y축 라벨
- xlim((시작값, 끝값)) : x축 값 범위
- ylim((시작값, 끝값)) : y축 값 범위

```
1 x = np.linspace(-5, 5, 1_000) # ndarray
2 y = np.power(x, 2)             # ndarray
3
4 plt.plot(x, y)    # plot a Line with x, y
5 plt.title("2d graph")
6 plt.xlabel("x")   # x axe Label
7 plt.ylabel('y')   # y axe Label
8 plt.xlim((-5,5)) # range tuple
9 plt.ylim((0,25)) # range tuple
10 plt.grid()       # show a grid
11 plt.show()       # show a whole graph
```

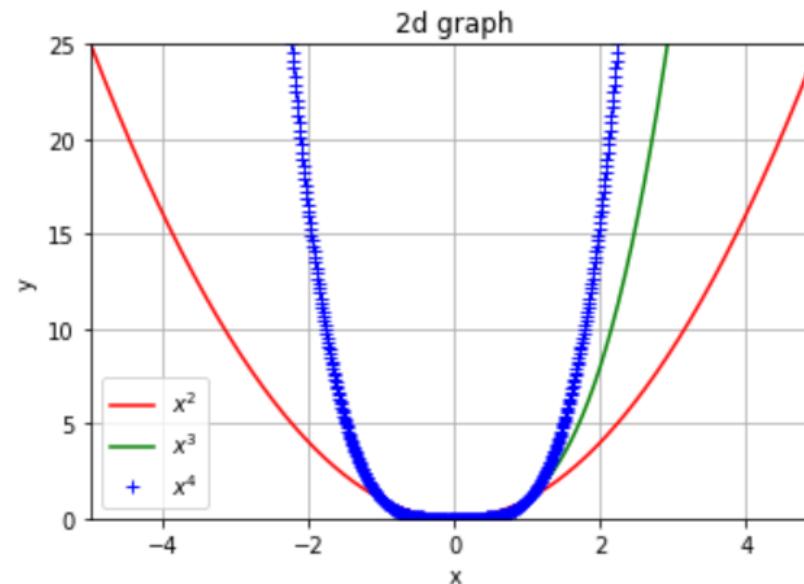


데이터 시각화 - multiple lines

- legend() : 범례 추가, loc 파라미터 = 표시 위치

```
1 x = np.linspace(-5, 5, 1_000) # ndarray
2
3 plt.plot(x, np.power(x, 2), "r-", label="$x^2$")
4 plt.plot(x, np.power(x, 3), "g-", label="$x^3$")
5 plt.plot(x, np.power(x, 4), "b+", label="$x^4$")
6
7 plt.title("2d graph")
8 plt.xlabel("x") # x axe label
9 plt.ylabel('y') # y axe label
10 plt.xlim((-5,5)) # range tuple
11 plt.ylim((0,25)) # range tuple
12 plt.grid() # show a grid
13 plt.legend(loc="best")
14 plt.show() # show a whole graph
```

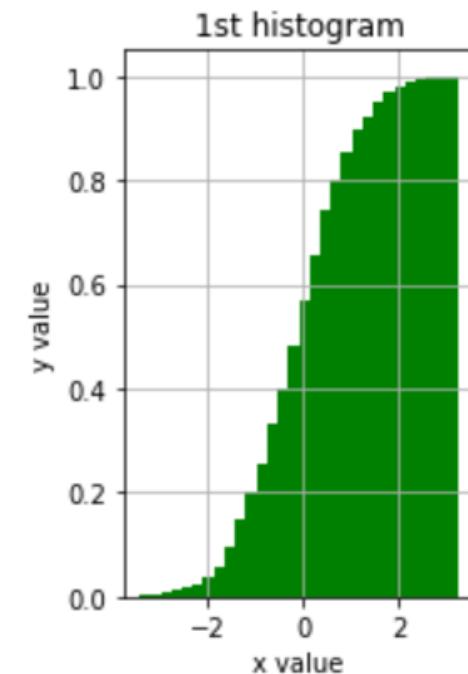
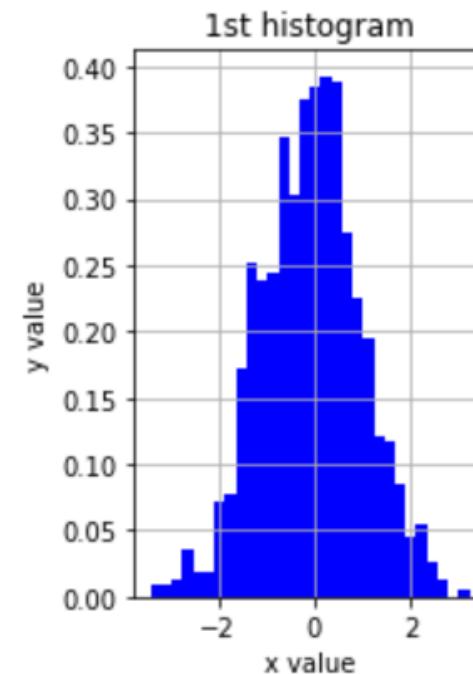
Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9



데이터 시각화 - histogram

- numpy.random 모듈 randn() 함수 : 임의의 표준정규분포 데이터 생성
- matplotlib 모듈 hist() 함수 : 히스토그램 그리기
 - bin=나누는 구간
 - cumulative=누적 옵션
- grid(True) : 그리드 배경
- savefig("파일명") 파일로 저장

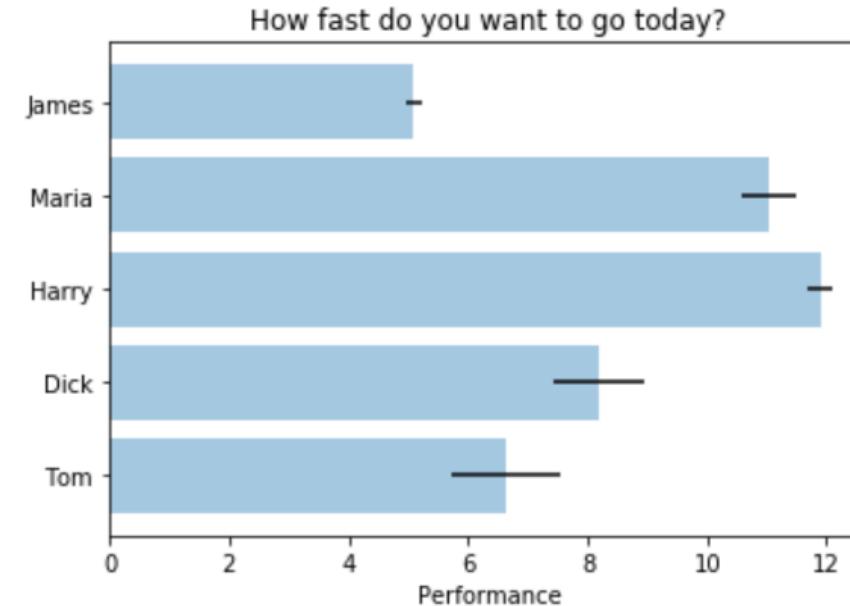
```
1 data = np.random.randn(1_000)
2
3 fig    = plt.figure()          # instance a figure object
4 axe_01 = fig.add_subplot(121) # 1st subplot in 2x1
5 axe_02 = fig.add_subplot(122) # 2nd subplot in 2x2
6
7 axe_01.set_title("1st histogram")
8 axe_01.set_xlabel("x value")
9 axe_01.set_ylabel("y value")
10 axe_01.grid(True)
11 axe_01.hist(data, bins=30, density=True, color='b')
12
13 axe_02.set_title("1st histogram")
14 axe_02.set_xlabel("x value")
15 axe_02.set_ylabel("y value")
16 axe_02.grid(True)
17 axe_02.hist(data, bins=30, density=True, color='g', cumulative=True)
18
19 plt.subplots_adjust(wspace=.4)
20 plt.show()
```



데이터 시각화 - Horizontal bar

- barh() 함수 : 수평 차트 그리기
- yticks() 함수 : ticker 위치별 각 위치에서의 라벨 설정

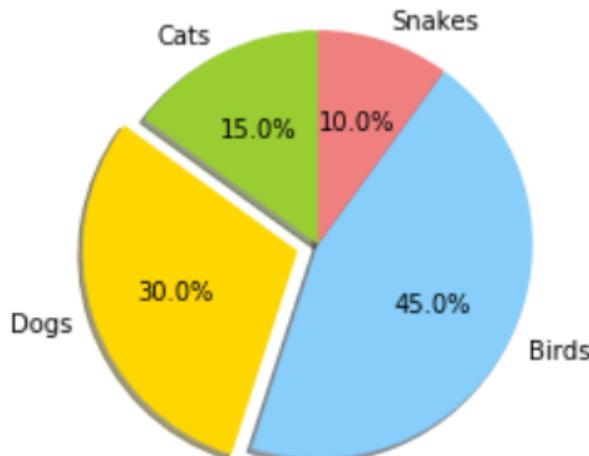
```
1 people = ["Tom", "Dick", "Harry", "Maria", "James"]
2
3 y_position = np.arange(len(people))
4 performance = 3 + 10 * np.random.rand(len(people))
5 error = np.random.rand(len(people))
6
7 plt.barh(y_position, performance,
8           xerr=error, align='center', alpha=0.4)
9 plt.yticks(y_position, people)
10 plt.xlabel("Performance")
11 plt.title("How fast do you want to go today?")
12
13 plt.show()
```



데이터 시각화 - 원 그래프

- pie () 함수 : 원 그래프 그리기

```
1 # The slices will be ordered and plotted counter-clockwise.
2 labels = ["Cats", "Dogs", "Birds", "Snakes"]
3 sizes = [15, 30, 45, 10]
4 colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']
5 explode = [0, 0.1, 0, 0] # only explode : 2nd slice(i.e. Dogs)
6
7 plt.pie(sizes, explode=explode, labels=labels, colors=colors,
8         autopct="%3.1f%%", shadow=True, startangle=90)
9
10 plt.show()
```



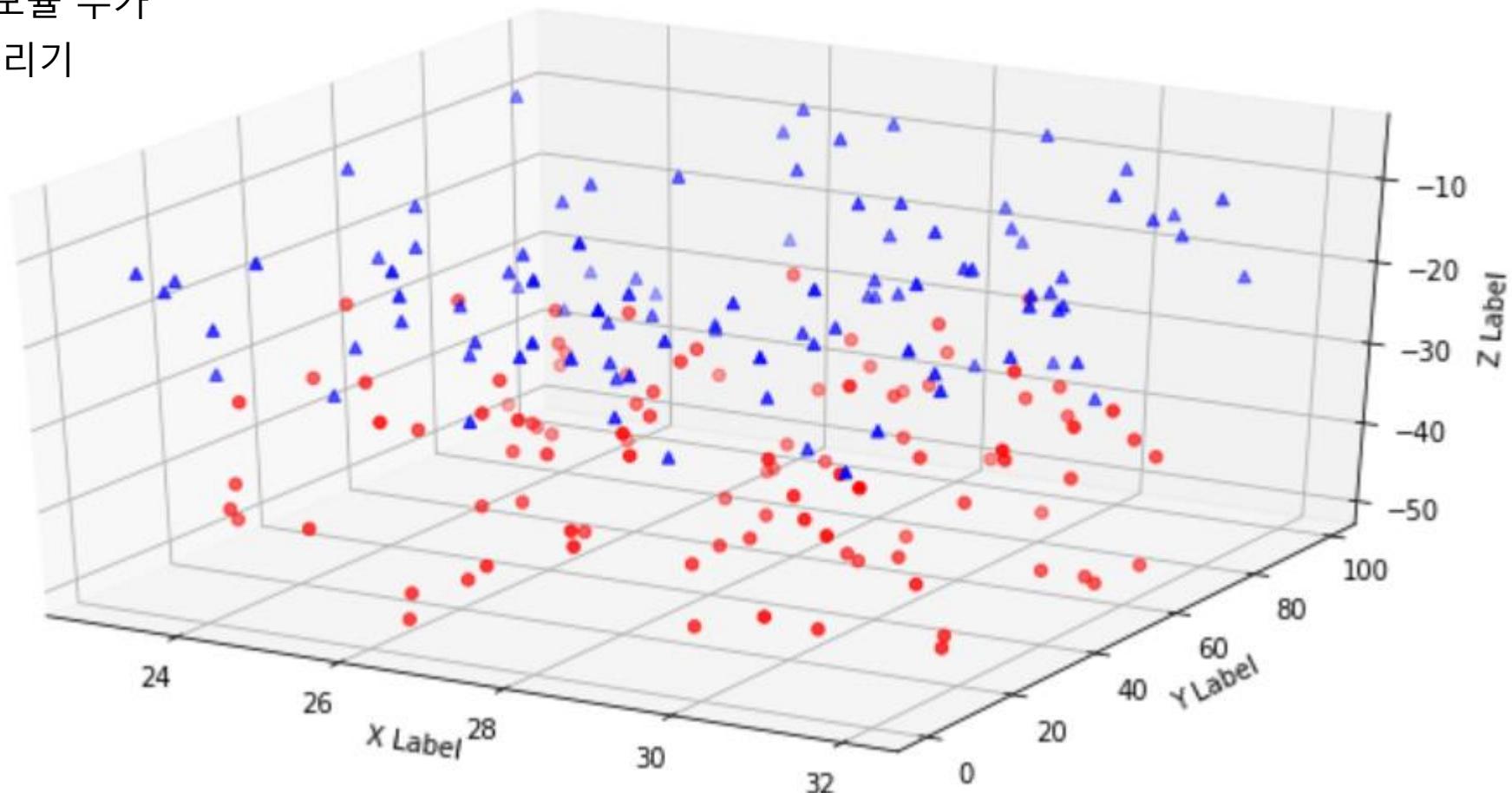
데이터 시각화 - 3D 그래프(2/1)

- mpl_toolkits.mplot3d 모듈 추가
- scatter() : 점 그래프 그리기

```
1 from mpl_toolkits.mplot3d import Axes3D
2
3 def randrange(n, min, max):
4     return (max-min)*np.random.rand(n)+min
5
6 n = 100
7 fig = plt.figure()
8 axe = fig.add_subplot(111, projection='3d')
9
10 for c, m, zl, zh in [('r', 'o', -50, -25), ('b', '^', -30, -5)]:
11     xs = randrange(n, 23, 32)
12     ys = randrange(n, 0, 100)
13     zs = randrange(n, zl, zh)
14     axe.scatter(xs, ys, zs, c=c, marker=m)
15
16 axe.set_xlabel('X Label')
17 axe.set_ylabel('Y Label')
18 axe.set_zlabel('Z Label')
19
20 plt.show()
```

데이터 시각화 - 3D 그래프(2/2)

- ◆ mpl_toolkits.mplot3d 모듈 추가
- ◆ scatter() : 점 그래프 그리기



머신러닝이란?

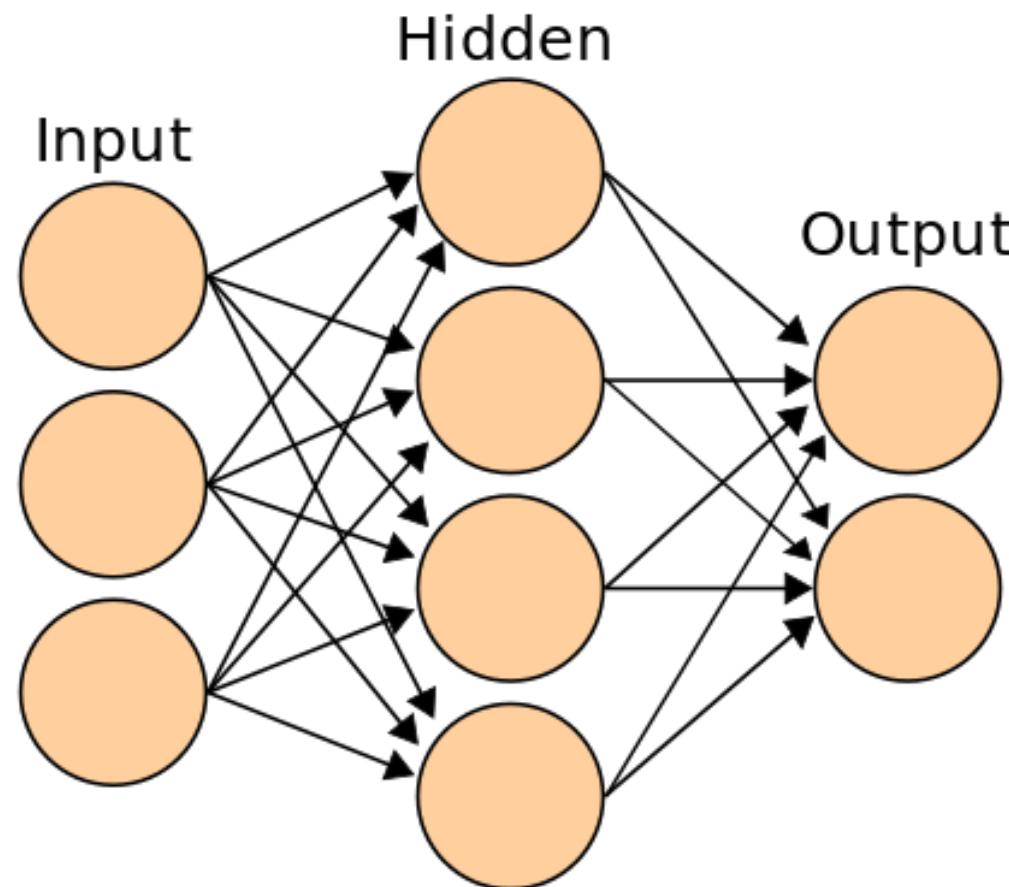
인간이 다양한 경험과 시행착오를 통해 지식을 배우는 것처럼, 컴퓨터에게 충분히 많은 데이터를 주고, 거기에서 일반적인 패턴을 찾아내게 하는 방법을 말합니다. (머신러닝의 대표적인 알고리즘이 딥러닝입니다.)

딥러닝이란?

머신러닝의 대표적인 학습법이 다음장 그림처럼 여러 층을 거쳐 점점 추상화 단계로 접어드는 알고리즘 형태인 '딥 러닝(Deep Learning)'입니다.

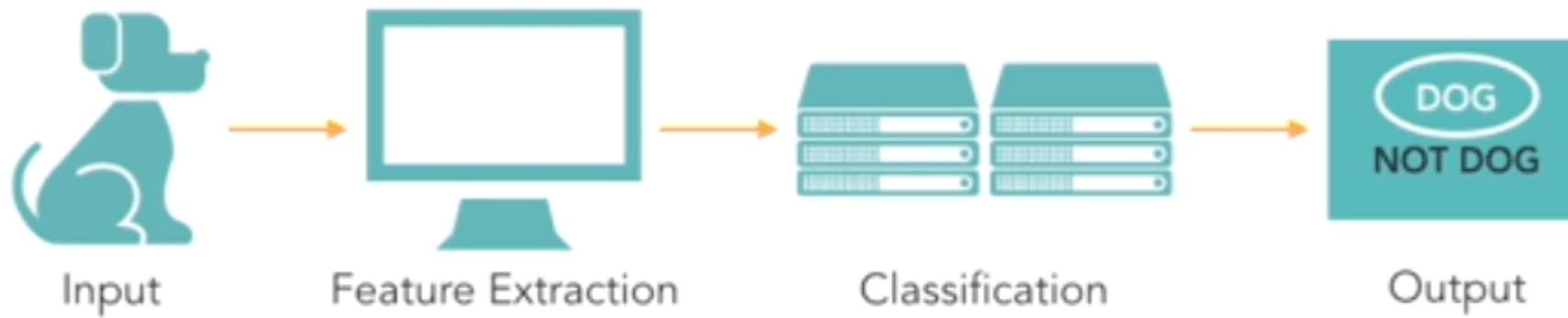
패턴을 찾기 위해선, 수많은 데이터가 있어야 되겠지요. 패턴을 견고하게 만드는 일종의 학습용 훈련데이터 말입니다.

딥러닝 레이어



Machine Learning vs Deep Learning

TRADITIONAL MACHINE LEARNING



DEEP LEARNING



보·이·는·대·로·믿·지·마·라!

대/반/전/ 음악추리쇼

나의 목소리가 보여

E



Mnet tvN 공동 방송

10월 22일 목요일 | 밤 9시 40분

6명 중에는 음치도 있고 노래를 잘하는 실력자(정상)도 있다.

번호 [1, 2, 3, 4, 5, 6]

정답 : [음치, 음치, 음치, 음치, 정상, 정상] 가 있다.

누가 음치인지 겉모습만 보고 맞춰야 한다.

감으로 예측을 한다.

예측 : [음치, 음치, 정상, 정상, 정상, 정상]

Predicted Values

Actual Values

1

0

TRUE POSITIVE

FALSE NEGATIVE

You're pregnant

FALSE POSITIVE

TYPE 1 ERROR

You're pregnant

TRUE NEGATIVE

TYPE 2 ERROR

You're not pregnant

You're not pregnant

강사 소개



정 준 수 / Ph.D (heinem@naver.com)

- 前) 삼성전자 연구원
- 前) 삼성의료원 (삼성생명과학연구소)
- 前) 삼성SDS (정보기술연구소)
- 現) (사)한국인공지능협회, AI, 머신러닝 강의
- 現) 한국소프트웨어산업협회, AI, 머신러닝 강의
- 現) 서울디지털재단, AI 자문위원
- 現) 한성대학교 교수(겸)
- 전문분야: 시각 모델링, 머신러닝(ML), RPA
- <https://github.com/jsjeong-me>