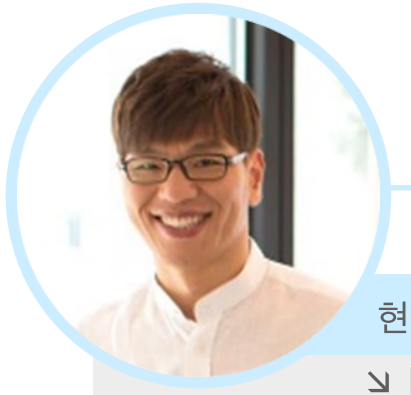
BIGDATA & AI ANALYTICS
EXPERT COMPANY

Python 분석 기초
NumPy

강사 소개



고 우 주 josh.ko@datakbuwa.com

현 데이터쿱와 대표분석 컨설턴트

- Business School Lausanne 박사 과정
- aSSIST, Big Data MBA 석사
- Aalto University, Executive MBA 석사

활동

- 명지대학교 ICT융합 겸임교수
- 한국데이터산업진흥원 빅데이터 전문 강사
- 한국여성과학기술인지원센터(WISET) 빅데이터 전문강사
- 인텔코리아 AI 분석 파트너 교육 강사
- 서울지역 인적자원개발위원회 4차산업분과위 자문위원

강의

- 한국데이터산업진흥원 금강대학교 빅데이터 인력양성 교육
- 한국데이터산업진흥원 빅데이터 청년인재 교육
- 한국데이터산업진흥원 정보보안 빅데이터 전문가교육
- WISET 빅데이터 AI 교육콘텐츠지도사 양성교육
- WISET 주관 덕성여대 빅데이터 교육
- 건국대학교 데이터사이언스 & MBA AI 강의

Contents

데이터의 기본구조

- 차원(벡터)와 배열
- 2차원 테이블
- 3차원 데이터의 종류

NumPy

- 배열(Array)의 개요
- 배열의 타입 및 연산
- 배열의 활용

Pandas

- 시리즈, 데이터프레임 개요
- 인덱스
- 데이터프레임 활용 및 응용

데이터 시각화

- Matplotlib
- Seaborn
- Pandas 시각화

튜토리얼 실습

빅데이터 분석 절차

- 데이터 정제
- 결측치 /이상치 처리
- 극단치 분석

- 데이터 스케일 맞추기(Feature Scaling)
- 더미 변수화 (Dummification)
- 차원 축소 (Dimensionality Reduction)

Python 데이터 분석 기본 패키지

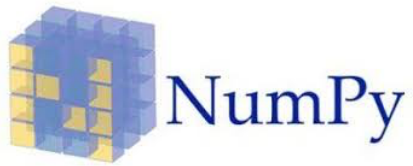
Python package

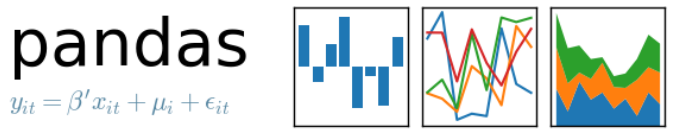
1. NumPy

2 pandas

3 matplotlib

4 Seaborn





NumPy

데이터는 소중하다

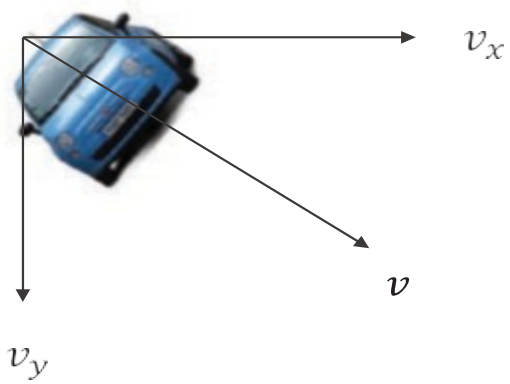
모방하고 반복하기

협업 프로젝트

BIGDATA & AI ANALYTICS
EXPERT COMPANY

데이터의 기본 구조

Vector, Scalar

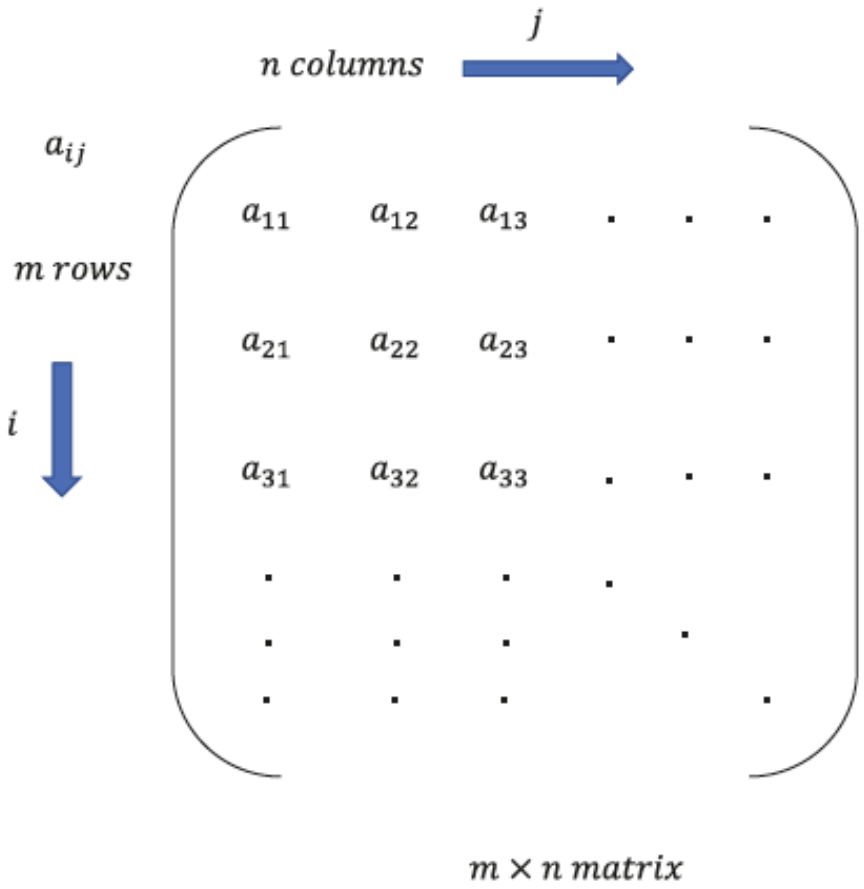


- **Vector :** 연속 또는 이산 숫자배열
- **Vector Space :** 벡터로 구성된 공간

- x와 y 방향으로 평면에서 움직이는 각각 속도 V_x, V_y

- **Scalar :** 1차원 벡터로 크기만 있고 방향성이 없는 양

NumPy의 배열



행렬 (Matrix)

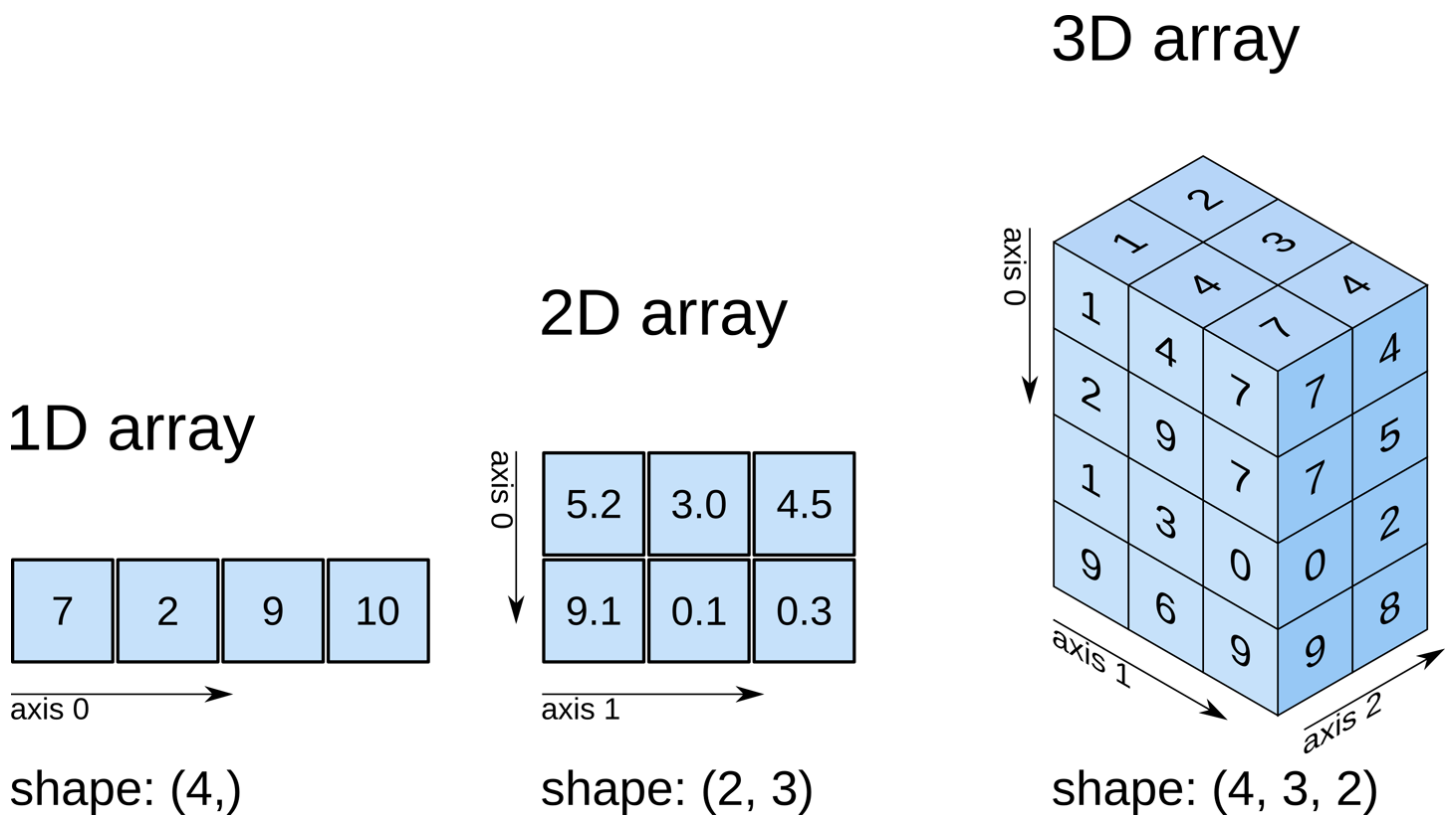
- 행렬은 행과 열로 배열된 숫자의 2차원 배열(Arrary)
- 행렬 A가 m × x 원소를 각는 객체이며 $A_{m\times n}$ 로 표시

$$A_{m\times n} \in \mathbb{R}^{m\times n}$$

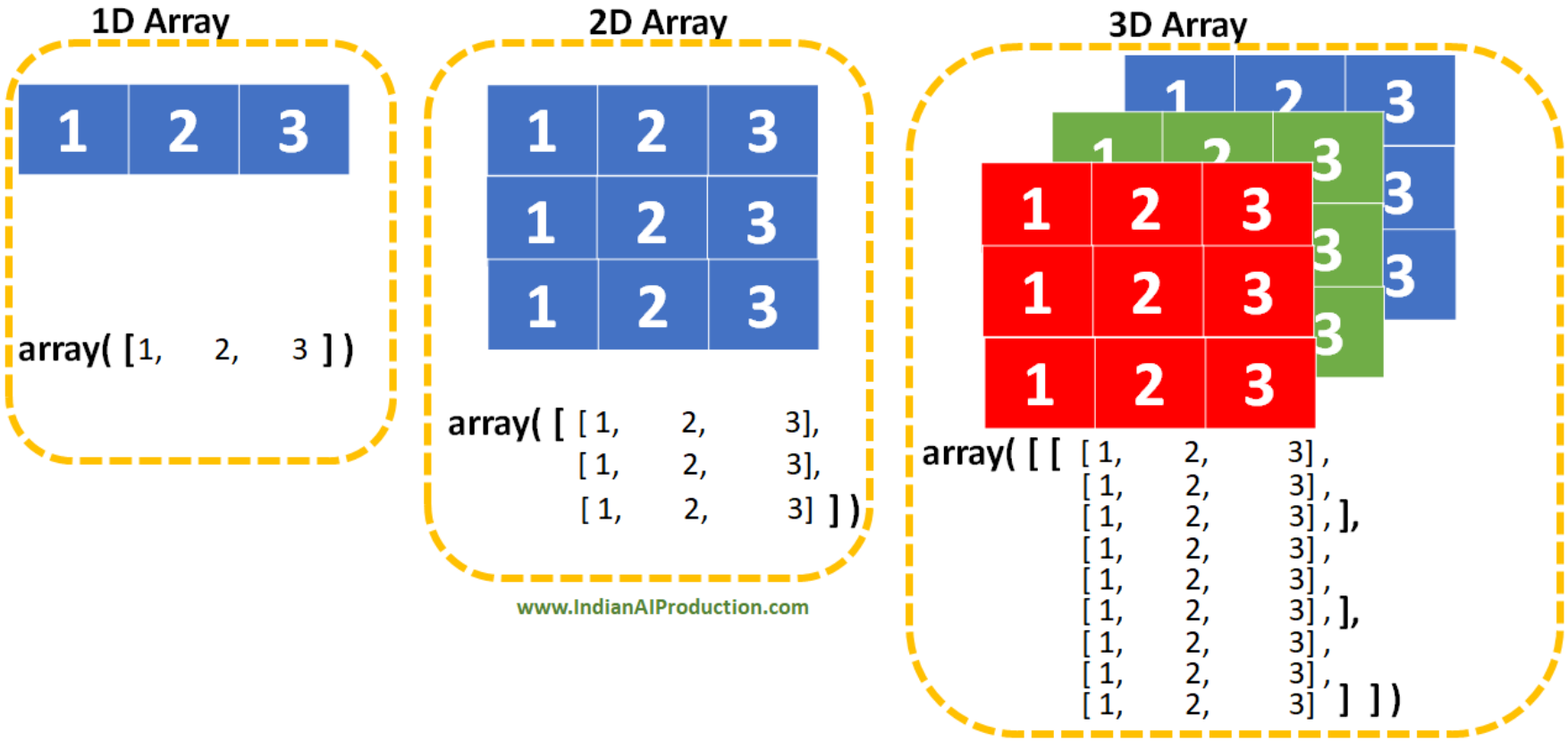
배열 (Arrary)

- 숫자의 다차원 배열로 벡터와 행렬은 NumPy의 1D, 2D 배열이다.
- 파이썬 기계학습에서 배열은 주로 1, 2, 3, 4차원이다.

NumPy의 배열


1D array
7
2
9
10
axis 0
shape: (4,)
2D array
axis 0
5.2
3.0
4.5
9.1
0.1
0.3
axis 1
shape: (2, 3)
3D array
axis 0
axis 1
axis 2
shape: (4, 3, 2)

NumPy의 다차원 배열



2차원 테이블 데이터

Excel 온라인 | 편집 ⌵ ⓘ 데이터 새로 고침이 지원되지 않음

	A	B	C	D	E	F	G
1	CustomerKey	GeographyKey	FirstName	LastName	BirthDate	MaritalStatus	Gender
2	11000	26	Jon	Yang	1971-10-06	M	M
3	11001	37	Eugene	Huang	1976-05-10	S	M
4	11002	31	Ruben	Torres	1971-02-09	M	M
5	11003	11	Christy	Zhu	1973-08-14	S	F
6	11004	19	Elizabeth	Johnson	1979-08-05	S	F
7	11005	22	Julio	Ruiz	1976-08-01	S	M
8	11006	8	Janet	Alvarez	1976-12-02	S	F
9	11007	40	Marco	Mehta	1969-11-06	M	M
10	11008	32	Rob	Verhoff	1975-07-04	S	F
11	11009	25	Shannon	Carlson	1969-09-29	S	M
12	11010	22	Jacquelyn	Suarez	1969-08-05	S	F
13	11011	22	Curtis	Lu	1969-05-03	M	M
14	11017	39	Shannon	Wang	1949-12-24	S	F
15	11018	32	Clarence	Rai	1955-10-06	S	M
16	11025	24	Alejandro	Beck	1951-06-22	M	M
17	11026	4	Harold	Sai	1951-10-01	S	M
18	11027	40	Jessie	Zhao	1952-06-05	M	M
19	11028	17	Jill	Jimenez	1951-10-09	M	F
20	11029	32	Jimmy	Moreno	1952-06-19	M	M

테이블 데이터의 개요

2차원의 행과 열로 구성

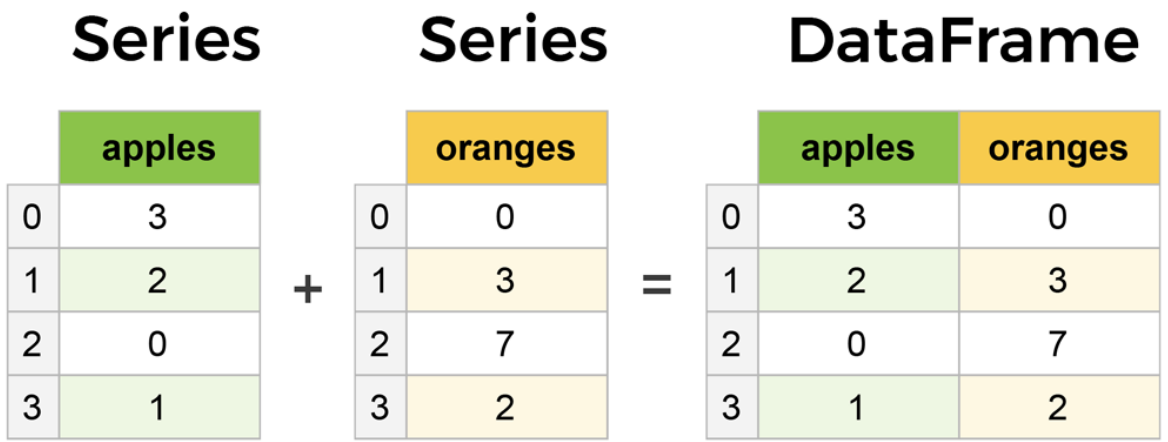
- 가로방향의 행(row)
- 세로방향의 열(column)

pandas 데이터프레임과 시리즈

시리즈 클래스 (series class)
시리즈 = 값(value) + 인덱스(index)

데이터프레임(DataFrame) 개요

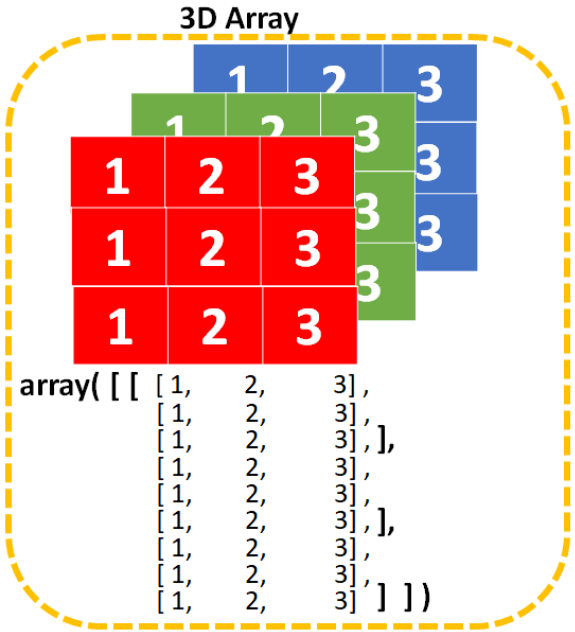
- 시리즈가 1차원 벡터 데이터에 행 방향 인덱스(row index)이라면,
- 데이터프레임(data-frame) 클래스는 2차원 행렬 데이터에 합친 것으로
- 행 인덱스(row index)와 열 인덱스(column index)를 지정

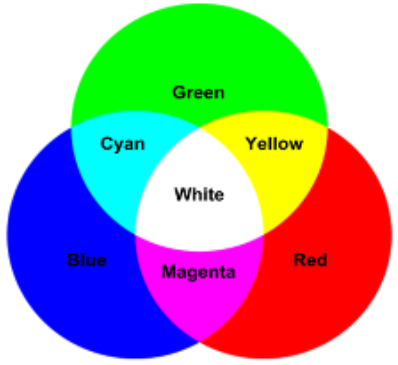


3차원 이미지 데이터

높이(Height)

너비(Width)



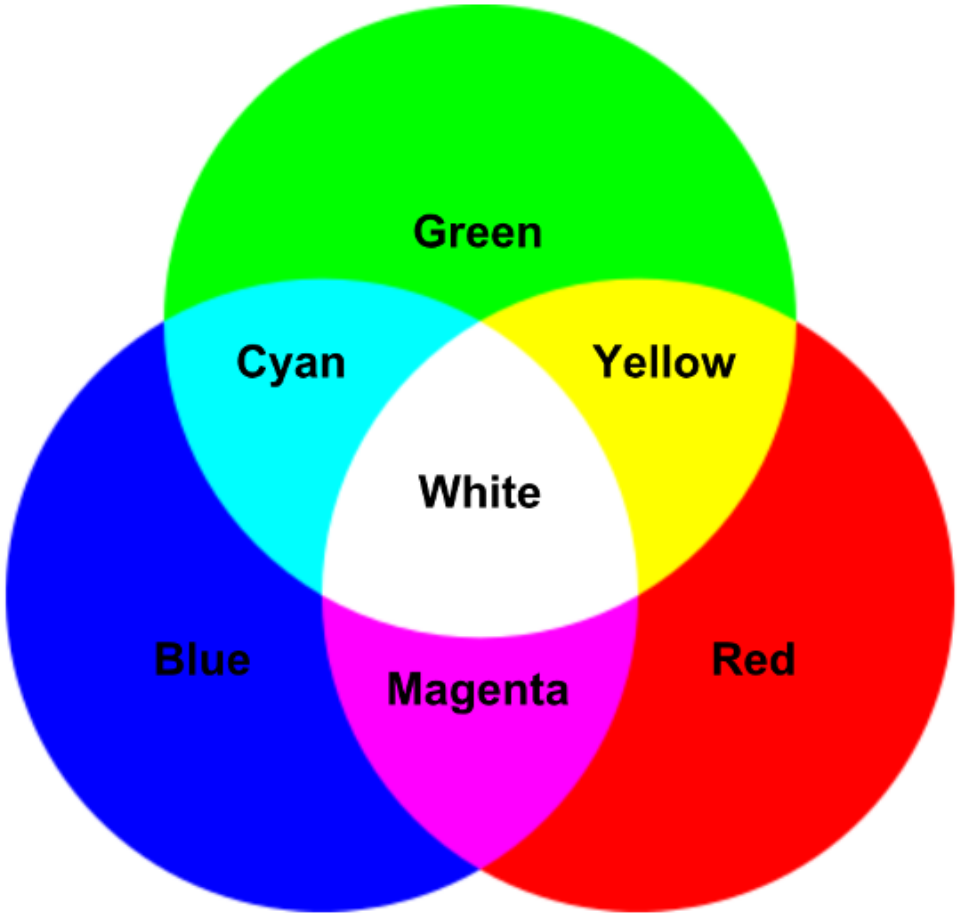


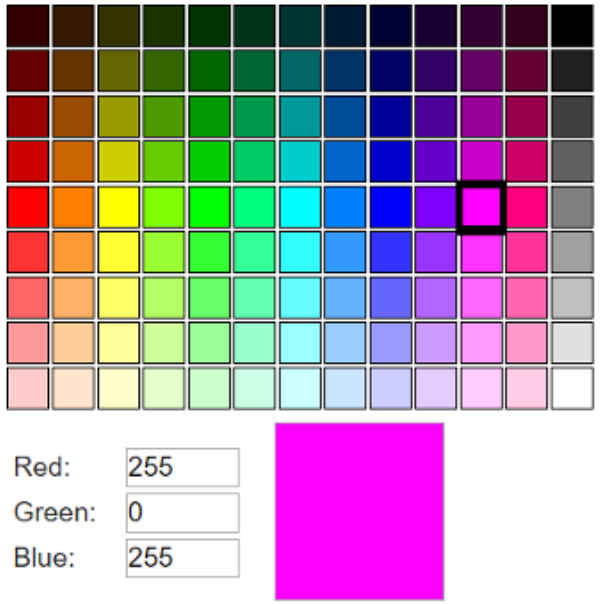
색상(Channel)

이미지 데이터의. 개요

3차원의
가로 방향의 너비(Width),
세로 방향의 높이(Height)와
색상 채널(Channel) 로 구성

이미지 컨셉



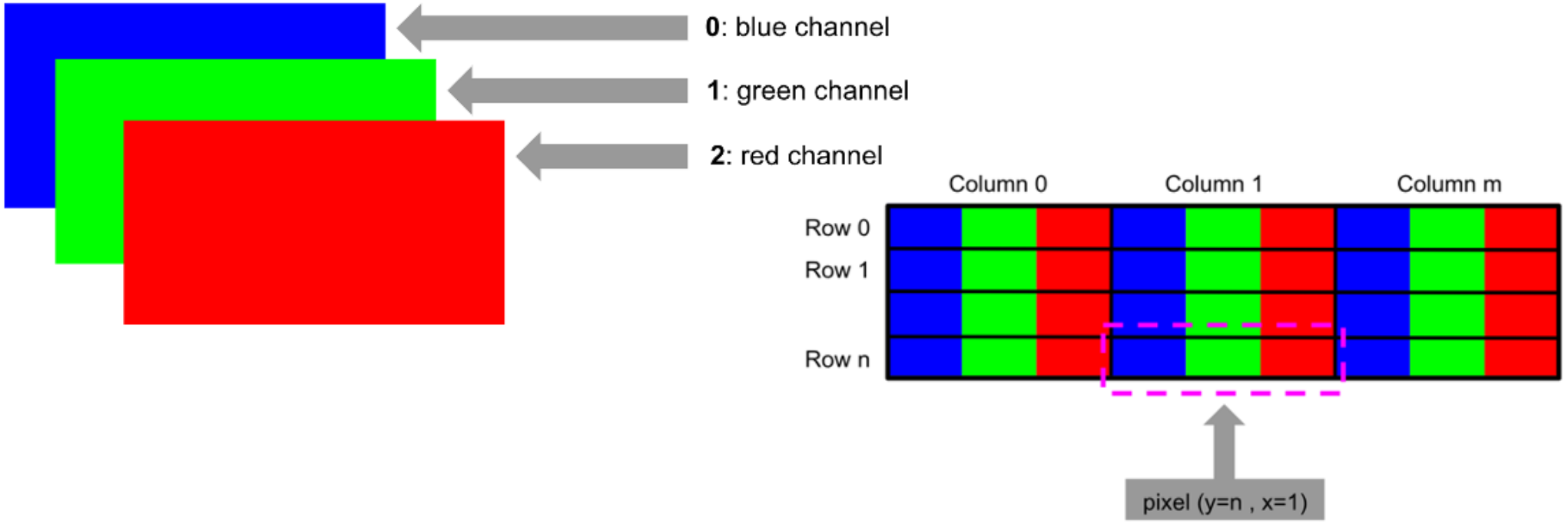


R, G, B
[0, 255] range.

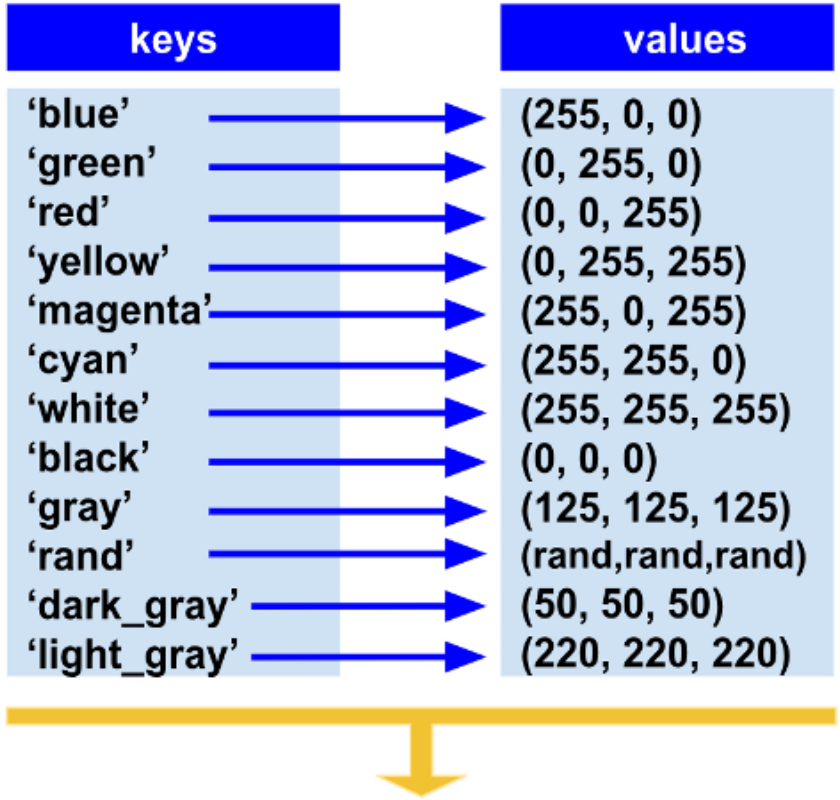
2^8 = 256
24-bit color depth:

RGB, BGR

- Python은 기본적으로 RGB
- OpenCV는 RGB 대신에 BGR 컬러 포맷 사용



Colors key & values



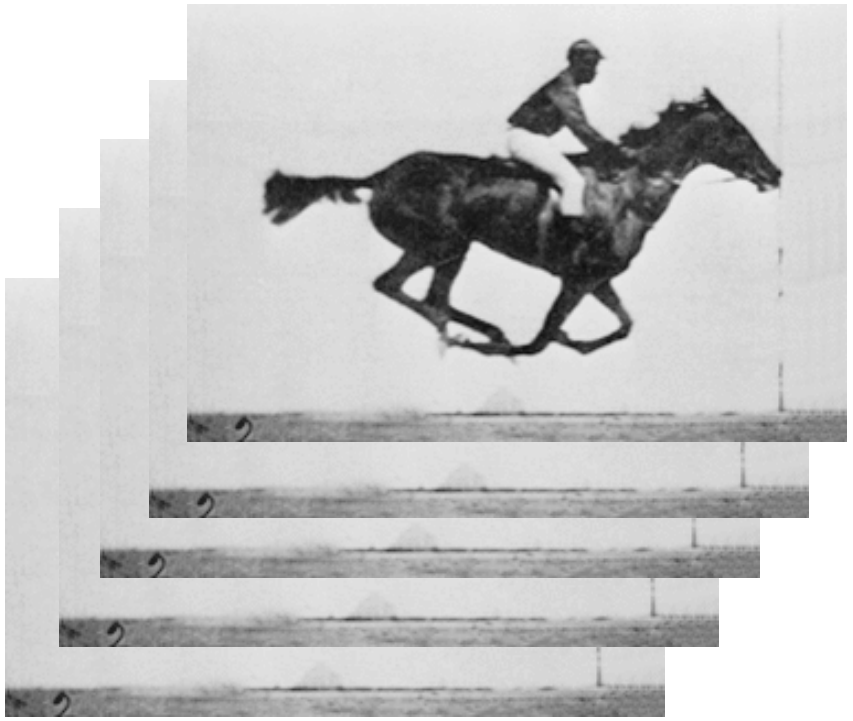
BGR 컬러 크기

BLUE = (255, 0, 0)
GREEN = (0, 255, 0)
RED = (0, 0, 255)
YELLOW = (0, 255, 255)
MAGENTA = (255, 0, 255)
CYAN = (255, 255, 0)
DARK_GRAY = (50, 50, 50)
...

```
get one color:
colors['light_gray']
```

```
colors = {'blue': (255, 0, 0), 'green': (0, 255, 0), 'red': (0, 0, 255), 'yellow': (0, 255,
255), 'magenta': (255, 0, 255), 'cyan': (255, 255, 0), 'white':  (255, 255, 255), 'black':
(0, 0, 0), 'gray': (125, 125, 125), 'rand': np.random.randint(0,high=256,size=(3,)).tolist(),
'dark_gray': (50, 50, 50), 'light_gray': (220, 220, 220)}
```

4차원 영상 데이터



FPS(Frame per Second)

영상 데이터의 개요

3차원의
세로 방향의 너비(Width),
가로방향의 높이(Height)와
색상 채널(Channel) 로 구성

+ 1차원의 프레임 Frame(fps)

(500, 500, 3, 100)

Text Tokenization

Tokenization 문장을 단어로 분리

one-hot-encoding 으로 카테고리 범주화

“The cat in the hat.”

```
['the','cat','in','the','hat','<EOS>']
```

Text Tokenization

tokens으로 단어 만들기

단어를 하나씩 one-to-one으로 딕셔너리 형태로 숫자 부여

리스트와 딕셔너리 사용으로 NumPy 배열 형식으로 변경

token ➤ word

```
[
  ‘<EOS>’,
  ‘the’,
  ‘cat’,
  ‘in’,
  ‘hat’,
  ‘.’
]
```

index ➤ word

```
{
  ‘<EOS>’: 0,
  ‘the’:    1,
  ‘cat’:    2,
  ‘in’:     3,
  ‘hat’:    4,
  ‘.’:   5
}
```

BIGDATA & AI ANALYTICS
EXPERT COMPANY

1. NumPy 개요

DATAKUBWA

NumPy 개요

1.1 NumPy 개요

- NumPy는 수치해석용 파이썬 패키지로 Numerical Python의 줄임말
- 다차원의 배열 자료구조 클래스인 ndarray 클래스를 지원
- 벡터와 행렬을 사용하는 선형대수 계산 사용

1.2 NumPy 특징

- NumPy의 배열 연산은 c로 구현된 내부 반복문을 사용
- 파이썬 반복문에 비해 빠른 속도
- 벡터화 연산(Vectorized Operation)을 이용
- 간단한 코드로도 복잡한 선형 대수 연산을 수행
- 배열 인덱싱(Array Indexing)을 사용한 질의(Query) 기능

DATAKUBWA

NumPy 개요

1.3 데이터 분석에서 빠른 연산을 위해 자주 사용하는 기능

- 배열에서 데이터 변경, 정제, 부분 집합, 필터링의 빠른 수행
- 정렬, 유일 원소 찾기, 집합 연산
- 통계 표현과 데이터의 수집/요약
- 여러 데이터의 병합, 데이터 정렬과 데이터 조작

1.4 NumPy 패키지 설치

- cmd, 터미널에서 설치시

```
pip install numpy
```

- Jupyter Notebook 설치 시

```
!pip install numpy
```

DATAKUBWA

NumPy 개요

1.5 NumPy 패키지 import

- 배열을 사용하기 위해 numpy 패키지를 임포트
- numpy는 np라는 축약어 사용이 관례

```
import numpy as np
```

NumPy

DATAKUBWA

NumPy 개요

1.6 스칼라와 벡터

- 스칼라 - 하나의 숫자만으로 이루어진 데이터로 아래와 같이 표기

$$x \in \mathbf{R}$$

- 벡터 - 벡터는 여러 개의 숫자가 특정한 순서대로 모여 있는 것으로

n개의 숫자가 모여 있으면 N-차원 벡터(n-dimensional vector)로 아래와 같이 표기

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

$$x \in \mathbf{R}^N$$

BIGDATA & AI ANALYTICS
EXPERT COMPANY

2. NumPy 배열

DATAKUBWA

1차원 배열

numpy의 array라는 함수에 리스트[]를 넣으면 배열로 변환

1차원 배열

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

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

타입 확인

```
type(arr)
```

> numpy.ndarray

DATAKUBWA

1차원 배열

numpy의 array라는 함수에 리스트를 넣으면 배열로 변환

1차원 배열 확인

```
# array의 형태(크기)를 확인
arr.shape
```

> (10,)

데이터 타입 확인

```
# array의 자료형을 확인
arr.dtype
```

> dtype('int64')

NumPy

DATAKUBWA

벡터화 연산

벡터화 연산(vectorized operation)

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

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

```
2 * x
```

> array([0, 2, 4, 6, 8, 10, 12, 14, 16, 18])

벡터화 연산

arrary 끼리 연산

```
a = np.array([1, 2, 3])
b = np.array([10, 20, 30])
```

```
2 * a + b
```

> array([12, 24, 36])

```
b - a
```

> array([9, 18, 27])

DATAKUBWA

조건연산

조건연산 (True, False)

```
a == 2
```

> array([False, True, False])

```
b > 10
```

> array([False, True, True])

```
(a == 2) & (b > 10)
```

> array([False, True, False])

DATAKUBWA

2차원 배열

- 다차원 배열 자료 구조 ex) 1차원, 2차원, 3차원 배열
- 2차원 배열은 행렬(matrix)로 행(row)과 열(column)로 구성

```
c = np.array([[0, 1, 2], [3, 4, 5]])
```

```
> array([[0, 1, 2],
         [3, 4, 5]])
```

2.8 행과 열의 갯수를 len() 함수로 확인

```
len(c)
```

```
> 2
```

```
len(c[0])
```

```
> 3
```

NumPy

DATAKUBWA

3차원 배열

배열을 행, 열, 깊이로 표시

2.9 3차원 배열 생성 (2 x 3 x 4 array)

```
d = np.array([[[1, 2, 3, 4],
               [5, 6, 7, 8],
               [9, 10, 11, 12]],
               [[11, 12, 13, 14],
               [15, 16, 17, 18],
               [19, 20, 21, 22]]])
d
```

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

        [11, 12, 13, 14],
        [15, 16, 17, 18],
        [19, 20, 21, 22]]])
```

3차원 배열

3차원 배열의 행, 열, 깊이를 len() 함수로 확인

```
len(d)
```

> 2

```
len(d[0])
```

> 3

```
len(d[0][0])
```

> 4

배열의 차원과 크기

ndim 속성은 배열의 차원, **shape** 속성은 배열의 크기를 반환

```
ab = np.array([1, 2, 3])
```

```
print(ab.ndim)
print(ab.shape)
```

> 1
(3,)

```
abc = np.array([[0, 1, 2], [3, 4, 5]])
```

```
print(abc.ndim)
print(abc.shape)
```

> 2
(2, 3)

BIGDATA & AI ANALYTICS
EXPERT COMPANY

3. 배열의 인덱싱

인덱싱(Indexing)

1차원 배열의 인덱싱

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

```
a[2]
```

> 2

```
a[-1]
```

> 4

NumPy

DATAKUBWA

인덱싱(Indexing)

다차원 배열의 인덱싱

- 배열 객체로 구현한 다차원 배열의 원소 중 하나의 개체를 선택
- 콤마로 구분된 차원을 축(axis)이라 하며, 그래프의 x, y축과 동일

```
b = np.array([[0, 1, 2], [3, 4, 5]])
b
```

```
> array([[0, 1, 2],
         [3, 4, 5]])
```

```
b[0, 0]
```

```
> 0
```

```
b[0, 1]
```

```
> 1
```

```
b[-1, -1]
```

```
> 5
```

DATAKUBWA

불리언 인덱싱

배열의 불리언 인덱싱 (Boolean Indexing)

- 불리안 배열 인덱싱 방식은 인덱스 배열의 원소가 True, False 두 값으로만 구성
- 인덱스 배열의 크기가 원래 ndarray 객체의 크기와 동일한 조건

예를 들어 1차원 ndarray에서 짝수인 원소만 골라내려면 짝수인 원소에 대응하는 인덱스 값이 true, 홀수인 원소에 대응하는 인덱스 값이 false인 인덱스 배열을 리스트로 넣어주면 된다.

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

idx = np.array([True, False, True, False, True,
                False, True, False, True, False])
a[idx]
```

> array([0, 2, 4, 6, 8])

NumPy

DATAKUBWA

불리언 인덱싱

```
a % 2
```

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

```
a % 2 == 0
```

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

```
a[a % 2 == 0]
```

> array([0, 2, 4, 6, 8])

DATAKUBWA

슬라이싱(Slicing)

배열의 슬라이싱

- 배열 객체로 구현한 다차원 배열의 원소 중 복수 개를 선택
- 일반적인 파이썬의 슬라이싱(slicing)과 comma(,)를 함께 사용

Slicing 사용 예

[:] 배열 전체
[0:n] 0번째부터 n-1번째까지, 즉 n번 항목은 포함하지 않는다.
[:5] 0번째부터 4번째까지,5번은 포함하지 않는다.
[2:] 2번째부터 끝까지
[-1] 제일 끝에 있는 배열값 반환
[-2] 제일 끝에서 두번째 값 반환

NumPy

DATAKUBWA

슬라이싱

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

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

```
a[0, :]  # 첫번째 행 전체
```

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

```
a[:, 1]  # 두번째 열 전체
```

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

```
a[:2, :2] # 첫번째 행 2열, 두번째 행 2열
```

```
> array([[0, 1],
         [4, 5]])
```

BIGDATA & AI ANALYTICS
EXPERT COMPANY

4. 데이터 타입

NumPy

DATAKUBWA

NumPy 데이터 타입

구분		type	type code
숫자형 (numeric)	bool형 (booleans)	bool	-
	정수형 (integers)	int8 int16 int32 **int64**	i1 i2 i4 i8
	부호없는 (양수) 정수형 (unsigned integers)	uint8 uint16 uint32 uint64	u1 u2 u4 u8
	부동소수형 (floating points)	float16 float32 **float64**	f2 f4 f8
	복수수형(실수 + 허수) (complex)	complex64 complex128	c8 c16
문자형 (character)	문자형 (string)	**string_**	u

데이터 타입 확인

- ndarray클래스는 데이터가 같은 자료형
- array 명령으로 배열을 만들 때 자료형 지정은 dtype 사용

```
a = np.array([1, 2, 3])
a.dtype
```

> dtype('int64')

```
b = np.array([1.0, 2.0, 3.0])
b.dtype
```

dtype('float64')

```
c = np.array([1, 2, 3.0])
c.dtype
```

> dtype('float64')

```
d = np.array([1, 2, 3], dtype="f")
d.dtype
```

> dtype('float32')

BIGDATA & AI ANALYTICS
EXPERT COMPANY

5. 배열 생성

DATAKUBWA

배열 생성

NumPy는 간단한 배열을 생성하는 명령을 제공

- zeros, ones
- zeros_like, ones_like
- empty
- arange
- linspace, logspace
- rand, randn

np.zeros

'0'으로 초기화된 배열을 생성

```
a = np.zeros(5)
a
```

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

```
b = np.zeros((2, 3))
b
```

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

```
c = np.zeros((5, 2), dtype="i")
c
```

```
array([[0, 0,
 [0, 0],
 [0, 0],
 [0, 0],
 [0, 0], dtype=int32)
```

np.ones

'1'로 초기화된 배열을 생성

```
d = np.ones((2, 3, 4), dtype="i8")
d
```

```
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]]])
```

```
d.dtype
```

```
> dtype('int64')
```

DATAKUBWA

np.arange

np.arange

Python 기본 명령어 range와 같은 특정한 규칙에 따라 증가하는 수열을 생성

```
np.arange(10)
```

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

```
# 시작, 끝(포함하지 않음), 단계
np.arange(1, 21, 2)
```

> array([1, 3, 5, 7, 9, 11, 13, 15, 17, 19])

NumPy

DATAKUBWA

전치연산

배열의 전치

행과 열을 바꾸는 전치(transpose) 연산으로 **t** 속성 사용

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

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

```
# 2x3 행렬을 3x2으로 전치
a.T
```

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

NumPy

DATAKUBWA

배열의 크기 변환

배열의 변형

만들어진 배열의 내부 데이터는 보존한 채로 형태만 **reshape** 함수로 변형

```
a = np.arange(12)
a
```

> array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])

```
# 1차원 배열을 3x4 행렬로 변형
b = a.reshape(3, 4)
b
```

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

BIGDATA & AI ANALYTICS
EXPERT COMPANY

6. 배열의 연산

NumPy

DATAKUBWA

벡터화 연산

벡터화 연산(Vectorized Operation)

x 벡터와 y 벡터의 연산 시

z = x + y

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ 10000 \end{bmatrix}, \quad y = \begin{bmatrix} 10001 \\ 10002 \\ 10003 \\ \vdots \\ 20000 \end{bmatrix},$$

$$z = x + y$$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ 10000 \end{bmatrix} + \begin{bmatrix} 10001 \\ 10002 \\ 10003 \\ \vdots \\ 20000 \end{bmatrix} = \begin{bmatrix} 1 + 10001 \\ 2 + 10002 \\ 3 + 10003 \\ \vdots \\ 10000 + 20000 \end{bmatrix} = \begin{bmatrix} 10002 \\ 10004 \\ 10006 \\ \vdots \\ 30000 \end{bmatrix}$$

NumPy

DATAKUBWA

벡터화 연산

4.2 numpy의 벡터 연산

```
x = np.arange(1, 10001)
y = np.arange(10001, 20001)
x, y
```

(array([1, 2, 3, ..., 9998, 9999, 10000]),
array([10001, 10002, 10003, ..., 19998, 19999, 20000]))

```
z = x + y
z
```

> array([10002, 10004, 10006, ..., 29996, 29998, 30000])

NumPy

DATAKUBWA

벡터 연산 - 지수,제곱, 로그함수

```
a = np.arange(5)
a
```

> array([0, 1, 2, 3, 4])

```
np.exp(a)
```

> array([1. , 2.71828183, 7.3890561 , 20.08553692, 54.59815003])

```
10 ** a
```

> array([1, 10, 100, 1000, 10000])

```
np.log(a + 1)
```

> array([0. , 0.69314718, 1.09861229, 1.38629436, 1.60943791])

NumPy

DATAKUBWA

스칼라와 벡터 연산

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

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

```
100 * x
```

> array([0, 100, 200, 300, 400, 500, 600, 700, 800, 900])

```
x = np.arange(12).reshape(3, 4)
x
```

> ararray([[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]])

```
100 * x
```

> array([[0, 100, 200, 300], [400, 500, 600, 700], [800, 900, 1000, 1100]])

NumPy

DATAKUBWA

브로드캐스팅

브로드캐스팅(Broadcasting)

- 벡터 연산 시에 두 벡터의 크기가 동일한 조건 요구,
- 브로드캐스팅(broadcasting)은 서로 다른 크기를 가진 두 배열의 사칙 연산에서,
- 크기가 작은 배열을 자동으로 반복 확장하여 크기가 큰 배열에 맞춰준다.

$$x = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \quad x + 1 = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + 1 = ?$$

브로드캐스팅은 다음과 같이 스칼라를 벡터와 같은 크기로 확장시켜서 덧셈 계산한다.

$$\begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + 1 = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

NumPy

DATAKUBWA

브로드캐스팅

```
x = np.array([[0, 1, 2], [1, 2, 3], [2, 3, 4], [4, 5, 6]])
x
```

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

```
y = np.arange(3)
y
```

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

```
x + y
```

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

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix}$$

BIGDATA & AI ANALYTICS
EXPERT COMPANY

7. 차원축소 연산

DATAKUBWA

1차원 배열의 차원축소 연산

차원 축소(Dimension Reduction) 연산은 행렬의 하나의 행에 있는 원소들을 하나의 데이터 집합으로 보고 그 집합의 평균을 구하면 1차원 벡터가 반환

NumPy 차원 축소 연산 메서드

- 최대/최소: min, max, argmin, argmax
- 통계: sum, mean, median, std, var
- 불리언: all, any

NumPy

DATAKUBWA

1차원 배열의 차원축소 연산

x = [1, 2, 3, 4] 일 때, 합(sum) 값을 구한다면

```
x = np.array([1, 2, 3, 4])
x
```

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

```
np.sum(x)
```

>10

```
x.sum()
```

> 10

DATAKUBWA

1차원 배열의 차원축소 연산

x = [1, 2, 3, 4] 일 때,
최소(min), 최대(max)값 및 그 수의 위치를 구한다면,

```
x = np.array([1, 2, 3, 4])
x
```

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

```
x.min()
```

> 1

```
x.max()
```

> 4

```
x.argmin()
```

> 0

```
x.argmax()
```

> 3

NumPy

DATAKUBWA

1차원 배열의 차원축소 연산

x = [1, 2, 3, 1] 일 때,
평균(mean), 중간(median)값을 구한다면,

```
y = np.array([1, 2, 3, 1])
y
```

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

```
y.mean()
```

> 1.75

```
np.median(y)
```

> 1.5

DATAKUBWA

2차원 배열의 차원축소 연산

- 연산의 대상이 2차원 이상인 경우에는 어느 차원으로 계산을 할 지를 axis 인수를 사용
- axis=0인 경우는 열 연산, axis=1인 경우는 행 연산
- 디폴트 값은 axis=0

```
x = np.array([[1, 3], [2, 4]])
x
```

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

```
x.sum()
```

```
> 10
```

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

```
> array([3, 7])
```

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

```
> array([4, 6])
```

DATAKUBWA

NumPy 통계 함수

4.9 NumPy를 이용한 기술 통계(descriptive statistics)

메서드	설 명
count	배열 전체의 요소 수를 계산한다.
sum	배열 전체 혹은 특정 축에 대한 모든 원소의 합을 계산
mean	산술평균을 구한다. 크기가 0인 배열에 대한 mean 결과는 nan
std, var	각각 표준편차와 분산을 구한다.
min, max	최소값, 최대값
argmin, argmax	최소 원소의 색인 값, 최대 원소의 색인 값
cumsum	각 원소의 누적 합
cumprod	각 원소의 누적 곱

DATAKUBWA

NumPy 통계 함수

샘플 x의 집합이 아래와 같다면,
x = {18, 5, 10, 23, 19, -5, 10, 0, 0, 5, 2, 126, 8, 2, 5, 5, 15, -3, 4, -1, -20 ,8, 9, -4, 25, -12}

```
x = np.array([18, 5, 10, 23, 19, -5, 10, 0, 0, 5, 2, 126, 8, 2, 5, 5, 15, -3, 4, -1, -20 ,8, 9, -4, 25, -12])
x
```

> array([18, 5, 10, 23, 19, -5, 10, 0, 0, 5, 2, 126, 8, 2, 5, 5, 15, -3, 4, -1, -20, 8, 9, -4, 25, -12])

```
len(x)  # 데이터 개수
```

> 26

```
np.mean(x) # 평균
```

> 9.76923076923077

```
np.var(x) # 분산
```

> 637.9467455621302

NumPy

DATAKUBWA

NumPy 통계 함수

```
np.std(x) # 표준편차
```

> 25.257607676938253

```
np.max(x)  # 최대값
```

> 126

```
np.min(x) # 최소값
```

> -20

```
np.median(x) # 중앙값
```

> 5.0

DATAKUBWA

NumPy 난수생성

데이터를 무작위로 섞거나 임의의 수 즉, 난수(random number)를 발생시키는 numpy의 random 서브패키지 사용 명령어는

- rand: 0부터 1사이의 균일 분포
- randn: 가우시안 표준 정규 분포
- randint: 균일 분포의 정수 난수

```
np.random.seed(0)
np.random.rand(10)
```

```
array([0.95894927, 0.65279032, 0.63505887, 0.99529957, 0.58185033,
0.41436859, 0.4746975 , 0.6235101 , 0.33800761, 0.67475232])
```

```
np.random.rand(3, 2)
```

```
array([[0.31720174, 0.77834548, 0.94957105],
 [0.6228461 , 0.67365963, 0.971945],
 [0.05571469, 0.45115921, 0.01998767]])
```

DATAKUBWA

NumPy 난수생성

4.11 randint 명령

numpy.random.randint(low, high=none, size=none)
(만약 high를 입력하지 않으면 0과 low사이의 숫자를, high를 입력하면 low와 high는 사이의 숫자를 출력하고, size는 난수의 숫자)

```
np.random.seed(0)
np.random.randint(10, 20, size=10)
```

> array([10, 19, 16, 15, 13, 11, 18, 10, 14, 19])

```
np.random.randint(10, 20, size=(3, 5))
```

```
array([[16, 15, 17, 18, 18],
 [19, 12, 18, 16, 16],
 [19, 11, 16, 18, 18]])
```