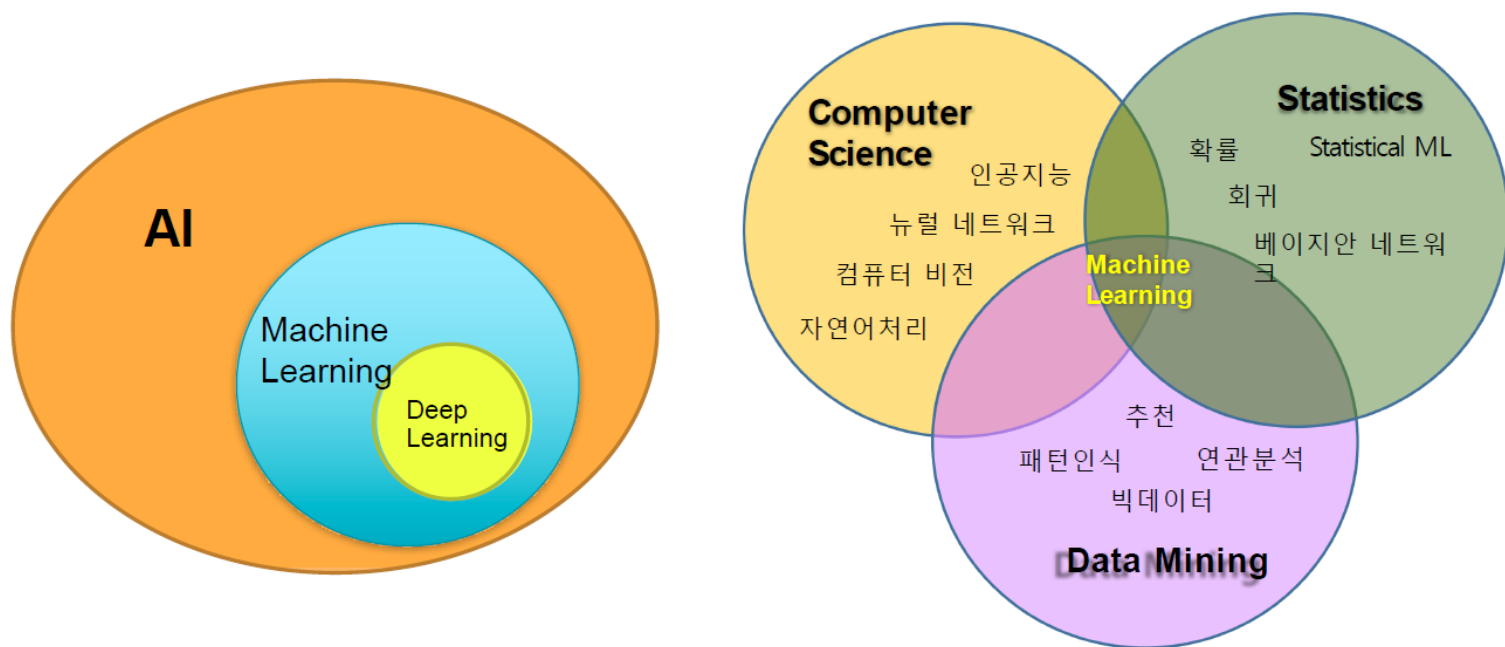


# 머신 러닝 (Machine Learning)

# 머신러닝의 개념

인공지능 > 머신러닝 > 딥러닝



# 머신러닝의 개념

---

- 어플리케이션을 수정하지 않고도, 데이터를 기반으로 패턴을 학습하고 결과를 예측하는 알고리즘 기법을 통칭.
  - 데이터를 기반으로 통계적인 신뢰도를 강화하고, 예측 오류를 최소화하기 위한 다양한 수학적 기법을 적용해 데이터 내의 패턴을 스스로 인지하고, 신뢰도 있는 예측 결과를 도출.
  - 데이터 분석 영역은 재빠르게 머신러닝 기반의 예측분석(Predictive Analysis)으로 재편되고 있음.
  - 많은 데이터 분석가와 데이터 과학자가 머신러닝 알고리즘 기반의 새로운 예측 모델을 이용해 더욱 정확한 예측 및 의사 결정을 도출하고 있으며, 데이터에 감춰진 새로운 의미와 통찰력(insight)를 발굴해 놀랄 만한 이익으로 연결시키고 있음.
  - 귀납적 학습 : 데이터만 주어지더라도 구조를 추론하려고 시도하기 때문.
-

# 머신러닝의 분류

---

## ➤ 지도학습(Supervised Learning)

- 회귀(Regression) : 단순 선형 회귀 / 다항 회귀
- 분류(Classification) : 이진 분류(로지스틱 회귀) / 다중 분류
- 추천 시스템
- 시각/음성 감지/인지
- 텍스트 분석, NLP

## ➤ 비지도학습(Un-supervised Learning)

- 군집화(clustering) : ex) k 평균 알고리즘
  - 차원 축소 : ex) 주성분 분석
-

# 머신러닝 모델 개선

---

- 최적의 머신러닝 알고리즘과 모델 파라미터를 구축하는 능력도 중요하지만,
  - 데이터를 이해하고 효율적으로 가공, 처리, 추출해서 최적의 데이터를 기반으로 알고리즘을 구동할 수 있도록 준비하는 능력이 무엇보다 더 중요.
  - 다양하고 광대한 데이터를 기반으로 만들어진 머신러닝 모델은 더 좋은 품질을 약속.
  - 앞으로 많은 회사의 경쟁력은 어떠한 품질의 데이터로 만든 머신러닝 모델이냐에 따라 결정.
-

# 파이썬이 R에 비해 뛰어난 점

---

- 쉽고 뛰어난 개발 생산성으로 전 세계 개발자들이 선호. 특히 구글, 페이스북 등 유수의 IT 업계에서도 파이썬의 높은 생산성으로 인해 활용도가 매우 높음.
  - 오픈 소스 계열의 전폭적인 지원을 받고 있으며 놀라울 정도의 많은 라이브러리로 인해 개발 시 높은 생산성을 보장.
  - 인터프리터 언어(Interpreter Language)의 특성상 속도는 느리지만, 대신에 뛰어난 확장성, 유연성, 호환성으로 인해 서버, 네트워크, 시스템, IOT, 심지어 데스크톱까지 다양한 영역에서 사용되고 있음.
  - 머신러닝 어플리케이션과 결합한 다양한 어플리케이션 개발이 가능.
  - 엔터프라이즈 아키텍처로의 확장 및 마이크로 서비스 기반의 실시간 연계 등 다양한 기업 환경으로의 확산이 가능.
  - 딥러닝 프레임워크인 텐서플로(TensorFlow), 케라스(Keras), 파이토치(PyTorch) 등에서 우선 정책으로 파이썬을 지원.
-

# 파이썬 언어를 이용한 데이터 분석 패키지

---

## ➤ 머신러닝 패키지

- Scikit-Learn : 데이터 마이닝 기반의 머신러닝.

## ➤ 행렬 / 선형대수 / 통계 패키지

- NumPy : 행렬과 선형대수를 다루는 패키지.
- SciPy : 자연과학과 통계를 위한 패키지.

## ➤ 데이터 핸들링

- Pandas : 데이터 처리 패키지, 2차원 데이터 처리에 특화.

## ➤ 시각화

- Matplotlib : 파이썬의 대표적인 시각화 패키지.
- Seaborn : matplotlib 기반으로 만들었지만, 판다스와의 쉬운 연동, 더 함축적인 API, 분석을 위한 다양한 유형의 그래프/차트 제공.

## ➤ 서드파티 라이브러리

## ➤ 아이파이썬(IPython, Interactive Python) 툴인 주피터 노트북(Jupyter Notebook)

# 넘파이(NumPy, Numerical Python)

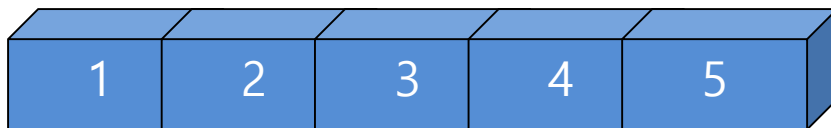
---

- 파이썬에서 선형대수 기반의 프로그램을 쉽게 만들 수 있도록 지원하는 대표적인 패키지.
  - 대량 데이터의 배열 연산을 가능하게 하므로 빠른 배열 연산 속도를 보장.
  - C/C++과 같은 저수준 언어 기반의 호환 API를 제공.
  - 파이썬 언어 자체가 가지는 수행 성능의 제약이 있으므로 수행 성능이 매우 중요한 부분은 C/C++ 기반의 코드로 작성하고 이를 넘파이에서 호출하는 방식으로 쉽게 통합 가능.
  - 다양한 데이터 핸들링 기능도 제공.
  - 넘파이 모듈의 임포트 방법
    - `import numpy as np`
    - 넘파이 기반 데이터 타입 : `ndarray`
  - 넘파이 `array()` 함수는 파이썬의 리스트와 같은 다양한 인자를 입력 받아서 `ndarray`로 변환하는 기능을 수행.
-

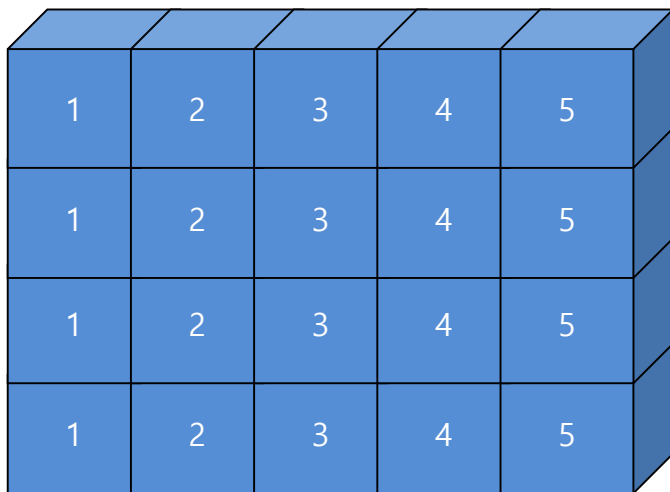


# 넘파이 ndarray 배열의 차원들

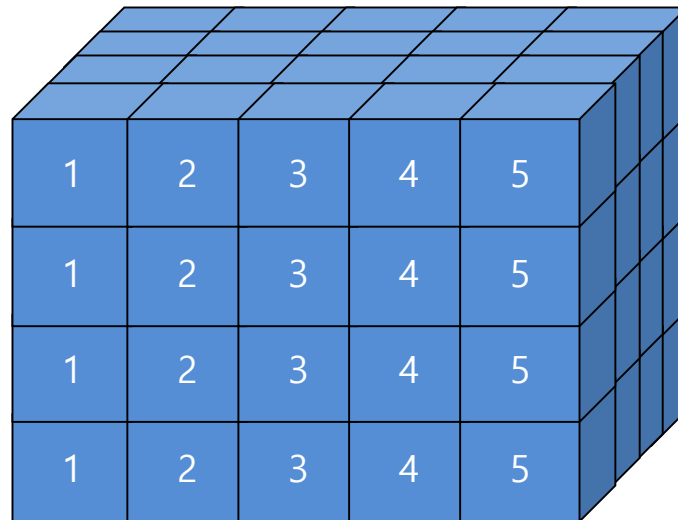
---



1차원 배열



2차원 배열



3차원 배열

# ndarray 배열의 shape 변수

---

- ndarray의 크기, 즉 행과 열의 수를 튜플 형태로 제공, 차원 확인까지 가능.

```
array1 = np.array([1,2,3])  
print('array1 type:', type(array1))  
print('array1 array 형태:', array1.shape)
```

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

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

---

# ndarray 배열

---

- 차원 확인 : ndarray.ndim

```
print('array1: {:0}차원, array2: {:1}차원, array3: {:2}차원'  
      .format(array1.ndim,array2.ndim,array3.ndim))
```

- 데이터 타입 확인 : ndarray.dtype

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

```
array1 = np.array(list1)  
print(type(array1))  
print(array1, array1.dtype)
```

# ndarray 배열

---

- 서로 다른 데이터 타입의 ndarray에서의 처리

```
list2 = [1, 2, 'test']  
array2 = np.array(list2)  
print(array2, array2.dtype)
```

```
list3 = [1, 2, 3.0]  
array3 = np.array(list3)  
print(array3, array3.dtype)
```

- ndarray 내 데이터 값의 타입 변경 : astype()

```
array_int = np.array([1, 2, 3])  
array_float = array_int.astype('float64')  
print(array_float, array_float.dtype)
```

```
array_int1 = array_float.astype('int32')  
print(array_int1, array_int1.dtype)
```

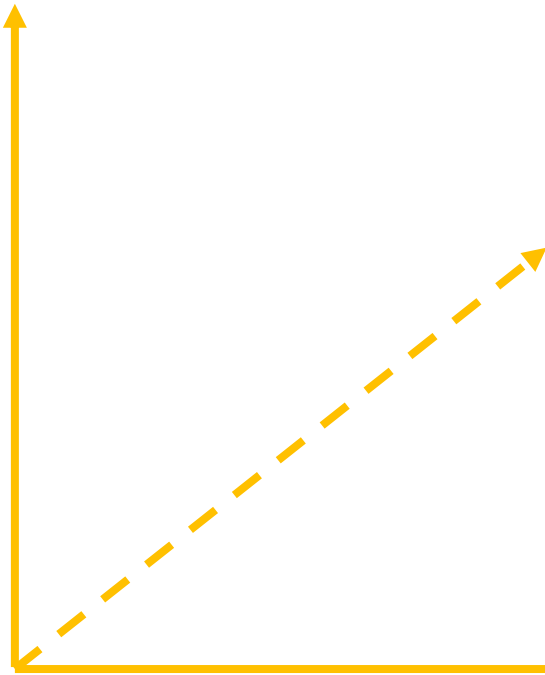
```
array_float1 = np.array([1.1, 2.1, 3.1])  
array_int2 = array_float1.astype('int32')  
print(array_int2, array_int2.dtype)
```

---

# 3차원 배열의 축(axis)

---

Y축(axis=2) : 열(column)



Z축(axis=0) : depth(면)

X축(axis=1) : 행(row)

3차원 배열에서 axis(축)의 의미

---

# axis 이해

---

## ➤ numpy의 sum 함수 사용 예

- 다음 코드는 3차원 배열을 만들고, 3차원 배열의 합을 구하는 코드

```
>>> arr = np.arange(0, 2*3*4)
```

```
>>> len(arr)
```

```
24
```

```
>>> arr
```

```
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])
```

```
>>> v = arr.reshape([2,3,4]) # 차원 변환 [2, 3, 4]; depth: 2, row: 3, column: 4
```

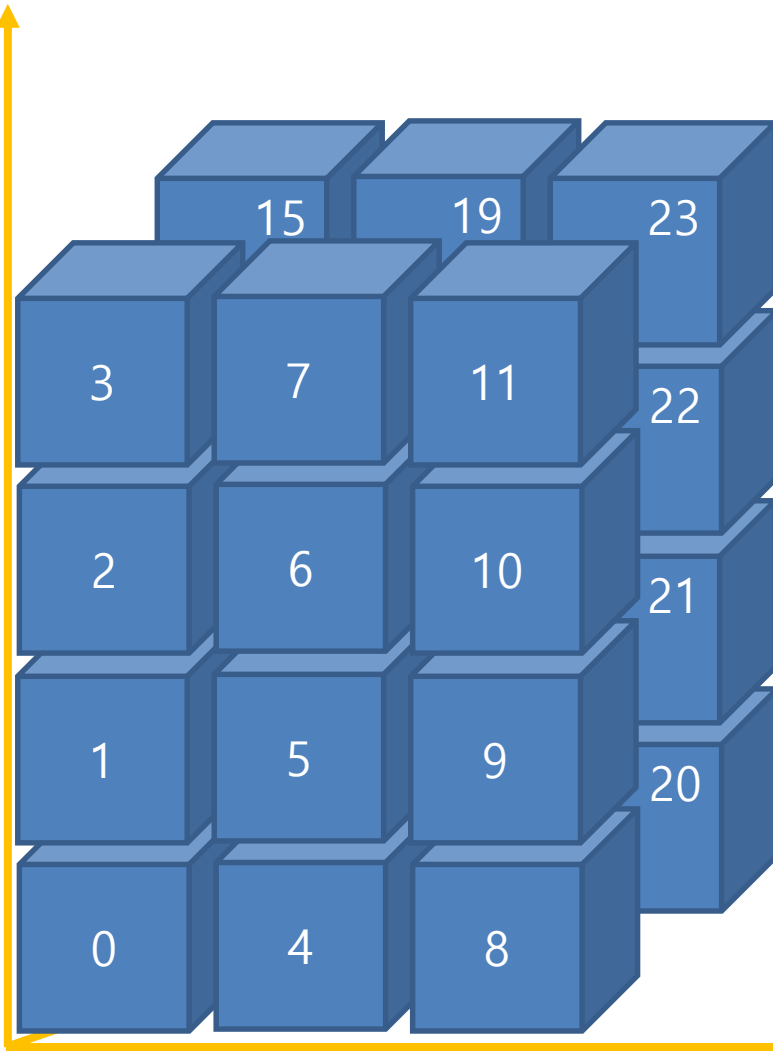
```
>>> v
```

```
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]]])
```

# 3차원 배열의 축(axis) - 실습 데이터

Y축(axis=2)



실습 데이터 다차원 배열의  
shape: (2, 3, 4) 데이터 구조

z축(axis=0)

X축(axis=1)

# axis 이해

```
>>> v.shape # v의 형상
```

```
(2, 3, 4)
```

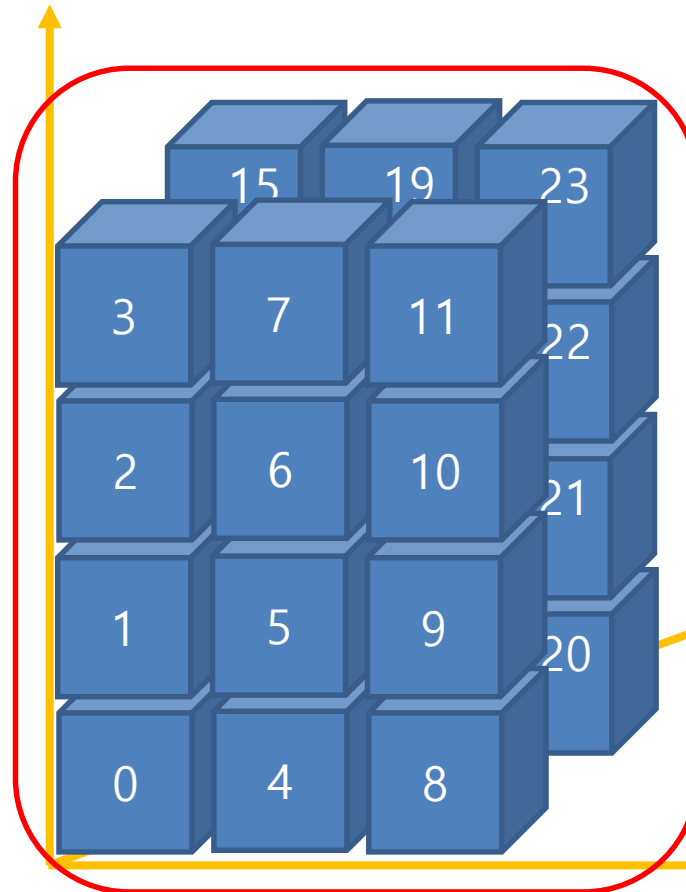
```
>>> v.ndim # v의 차원
```

```
3
```

```
>>> v.sum() # axis=None 기준 합계
```

```
276
```

Y축(axis=2)



z축(axis=0)

X축(axis=1)



# axis 이해

```
>>> res01=v.sum(axis=0) # axis=0 기준 합계
```

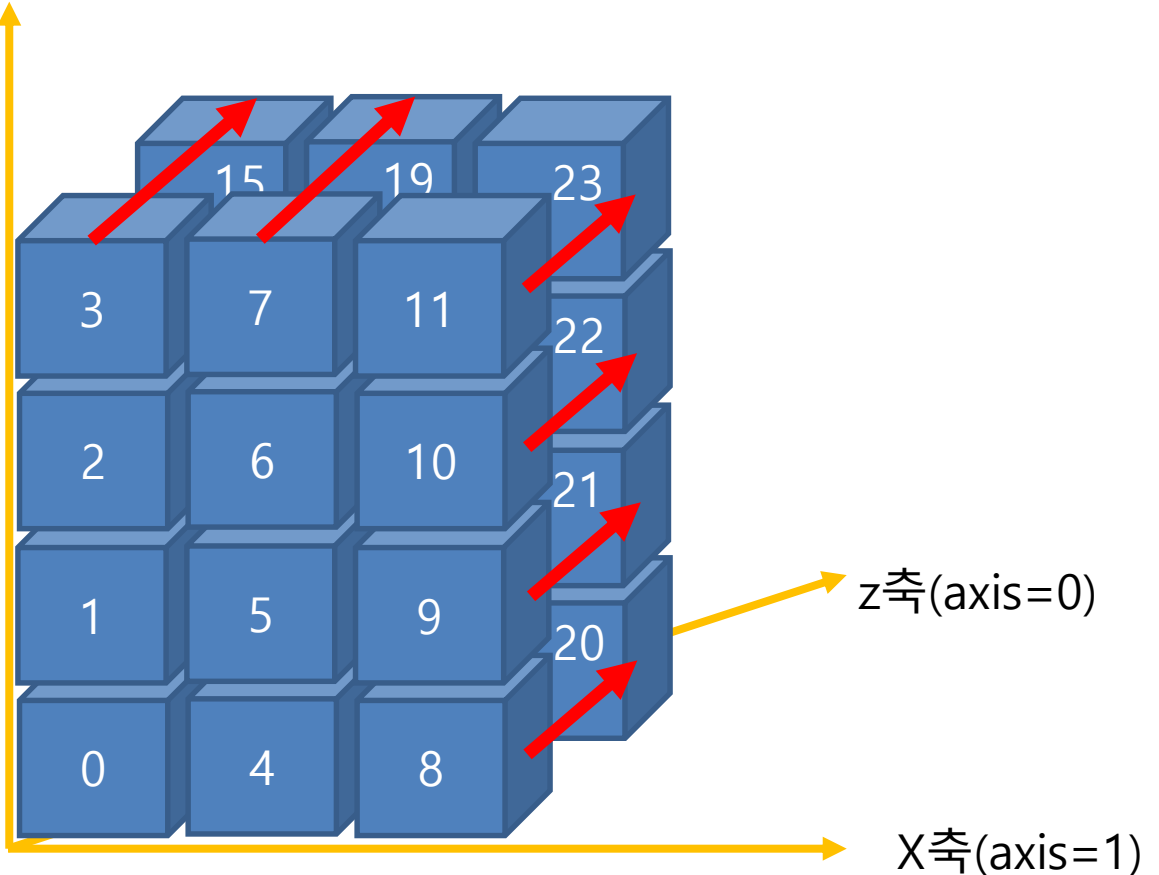
```
>>> res01.shape
```

```
(3, 4)
```

```
>>> res01
```

```
array([[12, 14, 16, 18],  
       [20, 22, 24, 26],  
       [28, 30, 32, 34]])
```

Y축(axis=2)



# axis 이해

```
>>> res02=v.sum(axis=1) # axis=1 기준 합계
```

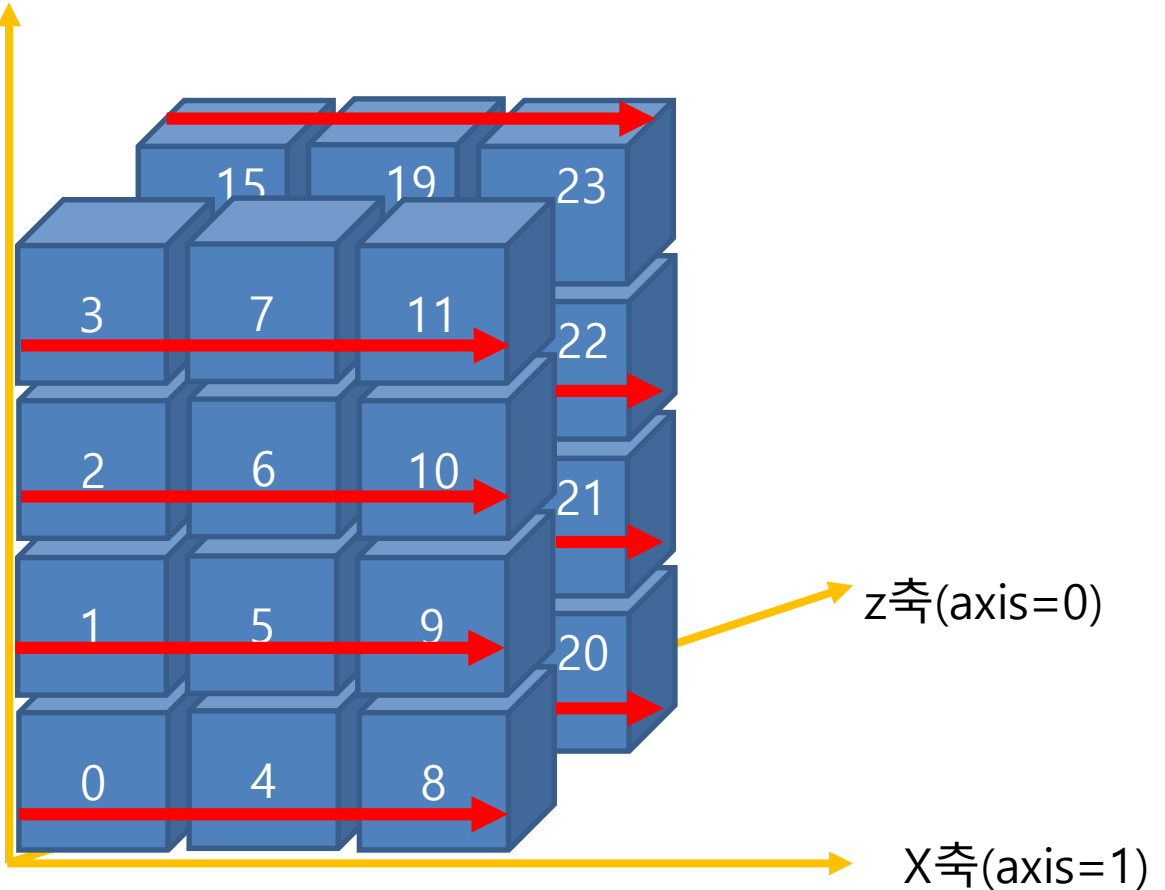
```
>>> res02.shape
```

```
(2, 4)
```

```
>>> res02
```

```
array([[ 12, 15, 18, 21],  
       [48, 51, 54, 57]])
```

Y축(axis=2)



# axis 이해

```
>>> res03=v.sum(axis=2) # axis=2 기준 합계
```

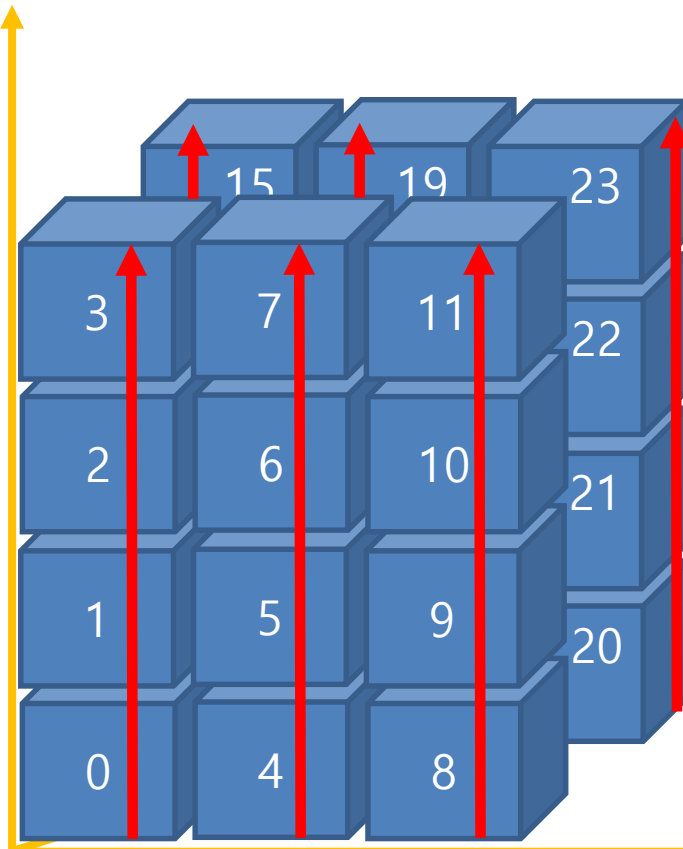
```
>>> res03.shape
```

```
(2, 3)
```

```
>>> res03
```

```
array([[ 6, 22, 38],  
       [54, 70, 86]])
```

Y축(axis=2)



Z축(axis=0)

X축(axis=1)

# 머신러닝 모델의 성능 평가 - 혼동행렬(Confusion Matrix)

- 혼동행렬이란 특정 분류 모델의 성능을 평가하는 지표로, 실제값과 모델이 예측한 예측값을 한 눈에 알아볼 수 있게 배열한 행렬.

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

## ➤ 혼동행렬의 구성 요소

- True Positive(TP): 실제 값이 참이고 모델의 예측 값도 참인 경우.
- True Negative(TN): 실제 값도 거짓이고 모델의 예측 값도 거짓인 경우.
- False Positive (FP): 실제 값은 거짓이나 모델의 예측 값이 참인 경우.
- False Negative(FN): 실제 값은 참이나 모델의 예측 값이 거짓인 경우.

# 머신러닝 모델의 성능 평가 - 혼동행렬(Confusion Matrix)

---

## ➤ 혼동행렬에 기반한 분류 모델 평가 지표

1. **정확도(Accuracy)**: 모델이 전체 중에서 예측을 올바르게 한 비율

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}}$$

2. **재현율(Recall)**: 실제 참인 값 중 모델이 참으로 예측한 값의 비율. 재현율이 낮을 경우 암인 사람에게 암이 아니라고 하였으니 심각한 결과를 초래할 것.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

3. **정밀도(Precision)**: 모델이 참으로 예측한 값 중 실제 참인 값의 비율. 정밀도가 낮을 경우 암이 아닌 사람에게 암이라고 했으니 불필요한 치료가 발생할 수 있음.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

4. **F1 점수(F1-Score)**: 정밀도와 재현율의 조화 평균

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

---

# sklearn의 train\_test\_split() 사용법

---

## ➤ 개요

머신러닝 모델을 학습하고 그 결과를 검증하기 위해서는 원래의 데이터를 Training, Validation, Testing 의 용도로 나누어 다뤄야 한다. 그렇지 않고 Training에 사용한 데이터를 검증용으로 사용하면 시험문제를 알고 있는 상태에서 공부를 하고 그 지식을 바탕으로 시험을 치르는 꼴이므로 제대로 된 검증이 이루어지지 않기 때문이다.

딥러닝을 제외하고도 다양한 기계학습과 데이터 분석 툴을 제공하는 scikit-learn 패키지 중 model\_selection에는 데이터 분할을 위한 train\_test\_split 함수가 들어있다.

# sklearn의 train\_test\_split() 사용법

---

## ➤ Parameter & Return

```
from sklearn.model_selection import train_test_split  
train_test_split(arrays, test_size, train_size, random_state, shuffle, stratify)
```

### (1) Parameter

- **arrays** : 분할시킬 데이터를 입력 (*Python list, Numpy array, Pandas dataframe 등..*)
- **test\_size** : 테스트 데이터셋의 비율(float)이나 갯수(int) (*default = 0.25*)
- **train\_size** : 학습 데이터셋의 비율(float)이나 갯수(int) (*default = test\_size의 나머지*)
- **random\_state** : 데이터 분할시 셔플이 이루어지는데 이를 위한 시드값 (*int나 RandomState로 입력*)
- **shuffle** : 셔플 여부 설정 (*default = True*)
- **stratify** : 지정한 Data의 비율을 유지한다. 예를 들어, Label Set인 Y가 25%의 0과 75%의 1로 이루어진 Binary Set일 때, stratify=Y로 설정하면 나누어진 데이터셋들도 0과 1을 각각 25%, 75%로 유지한 채 분할된다.

### (2) Return

- **X\_train, X\_test, Y\_train, Y\_test** : arrays에 데이터와 레이블을 둘 다 넣었을 경우의 반환이며, 데이터와 레이블의 순서쌍은 유지된다.
  - **X\_train, X\_test** : arrays에 레이블 없이 데이터만 넣었을 경우의 반환
-

# sklearn의 train\_test\_split() 사용법

---

## ➤ Example

```
import numpy as np from sklearn.model_selection import train_test_split
```

```
X = [[0,1],[2,3],[4,5],[6,7],[8,9]]
```

```
Y = [0,1,2,3,4]
```

```
# 데이터(X)만 넣었을 경우
```

```
X_train, X_test = train_test_split(X, test_size=0.2, random_state=123)
```

```
# X_train : [[0,1],[6,7],[8,9],[2,3]]
```

```
# X_test : [[4,5]]
```

```
# 데이터(X)와 레이블(Y)을 넣었을 경우
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=321)
```

```
# X_train : [[4,5],[0,1],[6,7]]
```

```
# Y_train : [2,0,3]
```

```
# X_test : [[2,3],[8,9]]
```

```
# Y_test : [1,4]
```

---