머신러닝 2장
(데이터 다루기)

혼자 공부하는 머신러닝+딥러닝

목차

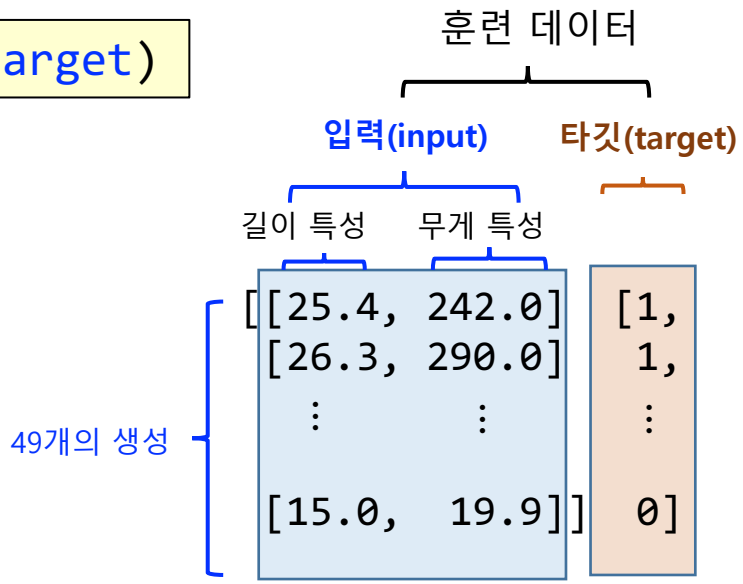
- 지도 학습, 비지도학습, 강화 학습 개요
- 훈련 세트와 테스트 세트
- 데이터 전처리
- 내용 정리

지도 학습과 비지도 학습

- **지도 학습 (Supervised Learning)**
  - 훈련을 위한 데이터(training data)와 정답이 필요

    훈련 데이터 = 입력(input) + 정답(target)

  - 알고리즘이 정답을 맞추는지 학습
  - 분류, 회귀



- **비지도 학습 (Unsupervised Learning)**
  - 정답(target)이 없고, 입력 데이터만 사용
  - 비슷한 특징끼리 군집화
  - 새로운 데이터에 대한 결과 예측
  - 클러스터링(Clustering)

- **강화 학습 (Reinforcement Learning)**
  - 현재 상태에서 어떤 행동을 취하는 것이 최선인지를 학습
  - {입력값-출력값}의 쌍이 정해지지 않음
  - 보상(reward)을 최대화하는 방법으로 학습
    - 바둑에서 승리, 주식 거래의 비용, 최종적으로 벌어들인 돈 등

훈련 세트와 테스트 세트

- 1장 모델의 문제점
  - 모든 데이터를 가지고 있음: 정답을 알고 있음
    - 정확도: 100%

- 훈련 세트와 테스트 세트 분리
  - 알고리즘의 정확한 평가
    - 훈련 데이터와 평가할 데이터가 달라야 됨

  - 정확한 평가를 위한 데이터 세트 준비 방법
    - 새로운 데이터를 준비
    - 이미 준비된 데이터의 일부분을 사용

  - 훈련 세트 (train set)
    - 훈련에 사용되는 데이터

  - 테스트 세트 (test set)
    - 평가에 사용되는 데이터

훈련 세트와 테스트 세트 만들기 #1

- 도미와 빙어 데이터를 2차원 리스트로 변경
  - 총 49마리의 데이터: 도미 35마리, 빙어 14마리
  - zip() 함수 사용

- 데이터 생성
  - 길이와 무게를 합친 2차원 리스트 생성: fish_data
  - 결과 확인을 위한 리스트 생성: fish_target

```
fish_length = [25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7, 30.0, 30.0, 30.7,
               31.0, 31.0, 31.5, 32.0, 32.0, 32.0, 33.0, 33.0, 33.5, 33.5,
                34.0, 34.0, 34.5, 35.0, 35.0, 35.0, 35.0, 36.0, 36.0, 37.0,
                38.5, 38.5, 39.5, 41.0, 41.0, 9.8,  10.5, 10.6, 11.0, 11.2,
                11.3, 11.8, 11.8, 12.0, 12.2, 12.4, 13.0, 14.3, 15.0]
fish_weight = [242.0, 290.0, 340.0, 363.0, 430.0, 450.0, 500.0, 390.0, 450.0, 500.0,
                475.0, 500.0, 500.0, 340.0, 600.0, 600.0, 700.0, 700.0, 610.0, 650.0,
                575.0, 685.0, 620.0, 680.0, 700.0, 725.0, 720.0, 714.0, 850.0, 1000.0,
                920.0, 955.0, 925.0, 975.0, 950.0, 6.7,  7.5, 7.0, 9.7, 9.8,
                8.7, 10.0, 9.9, 9.8, 12.2, 13.4, 12.2, 19.7, 19.9]
```

```
fish_data = [[l, w] for l, w in zip(fish_length, fish_weight)] # 2차원 리스트[길이, 무게]
fish_target = [1] * 35 + [0]*14 # target 값 생성
```

훈련 세트와 테스트 세트 만들기 #2

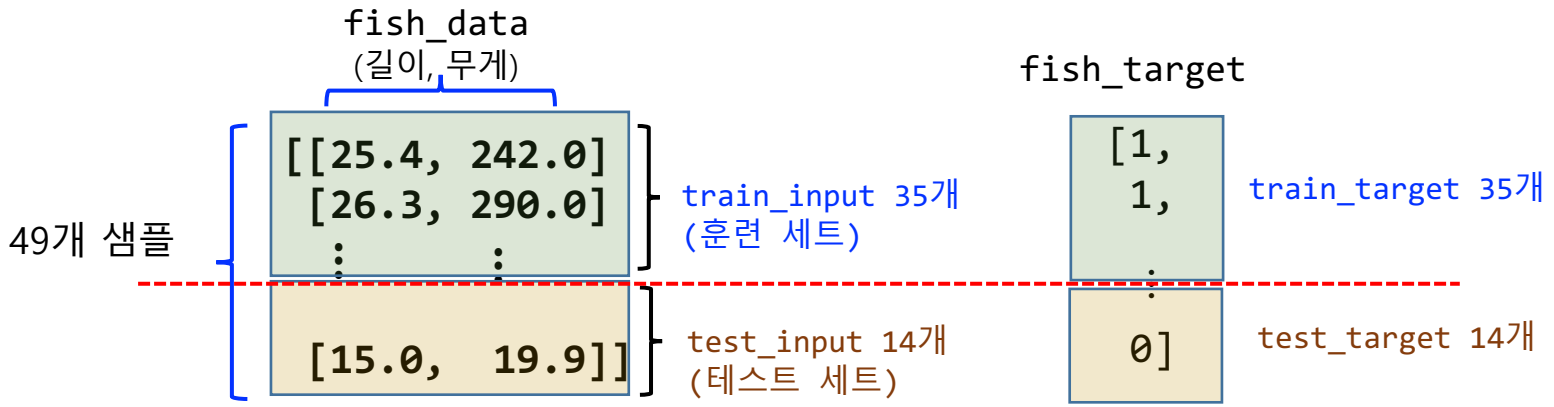
- 훈련 세트와 테스트 세트 만들기
  - fish_data, fish_target 리스트를 훈련 세트와 테스트 세트로 분리
    - 훈련 세트: 35개, 테스트 세트: 14개로 분리

- 훈련 세트 구성
  - train_input: 훈련용 입력 데이터
  - train_target: 훈련용 정답 데이터

```
# 훈련 세트 생성(35개)
train_input = fish_data[:35]
train_target = fish_target[:35]
```

- 테스트 세트 구성
  - test_input: 테스트 입력 데이터
  - test_target: 테스트 정답 데이터

```
# 테스트 세트 생성(14개)
test_input = fish_data[35:]
test_target = fish_target[35:]
```



테스트 세트로 평가하기

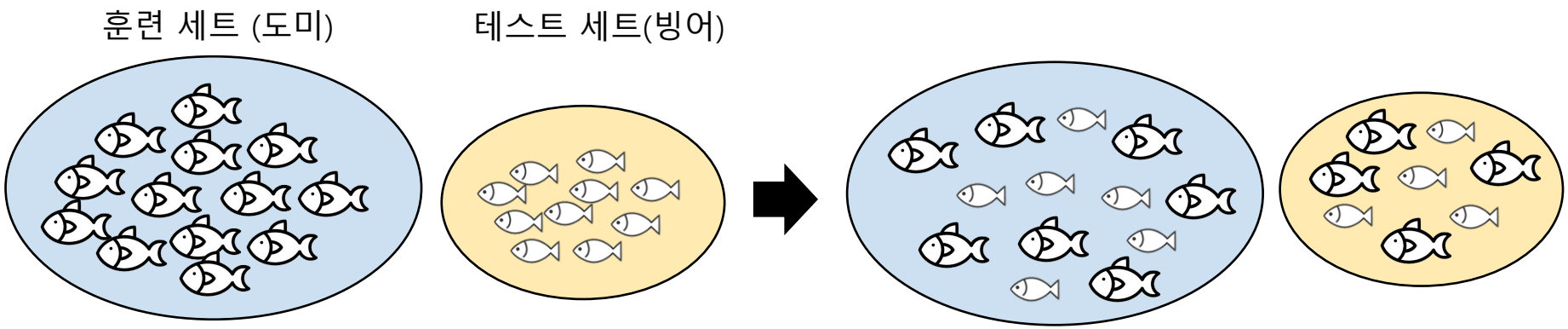
- k-최근접 알고리즘 모델 생성 및 평가하기
  - 훈련 세트로 모델을 훈련
  - 테스트 세트로 정확도 평가

```
from sklearn.neighbors import KNeighborsClassifier

kn = KNeighborsClassifier() # 객체 생성
kn = kn.fit(train_input, train_target) # 모델 훈련
kn.score(test_input, test_target) # 테스트 세트로 평가 (정확도 출력)
```

```
0.0
```

  - 잘못된 훈련 데이터 사용 결과
    - 정확도 0.0



Numpy를 활용한 배열 생성

- Numpy 사용
  - 다차원 배열을 쉽게 조작할 수 있음

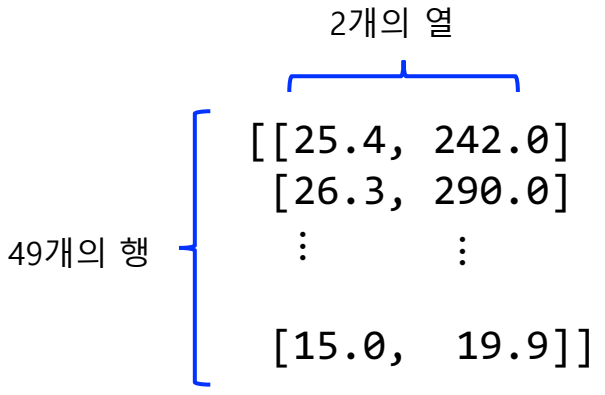
- 기존 fish_data와 fish_target를 Numpy 배열로 변경
  - Numpy.array(list): 파이썬의 list를 Numpy의 array로 변경

```
import numpy as np

input_arr = np.array(fish_data)
target_arr = np.array(fish_target)

print(input_arr)
print(input_arr.shape) # shape: 배열의 크기 리턴
```

```
[[  25.4  242. ]
 [  26.3  290. ]
 [  26.5  340. ]
 [  29.   363. ]
       . . .
 [  12.4   13.4]
 [  13.    12.2]
 [  14.3   19.7]
 [  15.    19.9]]
(49, 2)
```



Numpy를 이용하여 데이터 섞기

- Numpy 사용
  - random.seed(seed) : 랜덤값 생성을 위한 초기값 지정
    - 초기값(seed)값이 같으면 동일한 랜덤값을 뽑을 수 있음
  - Numpy.arange(stop): stop-1까지 1의 간격으로 배열 생성
    - arange: array range
  - random.shuffle(x): 배열 x(ndarray 타입)를 무작위로 섞음

```
import numpy as np

np.random.seed(42)
index_list = np.arange(49) # 0~48까지 값을 가지는 배열 생성
print(index_list)
```

```
[ 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 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48]
```

```
np.random.shuffle(index_list) # 배열 내용을 무작위로 섞음
print(index_list)
```

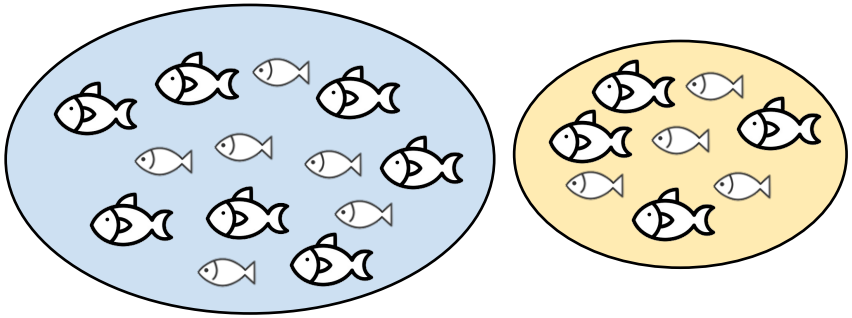
```
[13 45 47 44 17 27 26 25 31 19 12  4 34  8  3  6 40 41 46 15  9 16 24 33
 30  0 43 32  5 29 11 36  1 21  2 37 35 23 39 10 22 18 48 20  7 42 14 28
 38]
```

랜덤 배열로 훈련 세트와 테스트 세트 만들기

- Numpy의 배열 인덱싱 이용
  - 여러 개의 인덱스로 한 번에 여러 개의 배열 원소를 선택할 수 있음

```
# 훈련 세트 생성
train_input = input_arr[index_list[:35]]
train_target = target_arr[index_list[:35]]
```

```
# 테스트 세트 생성
test_input = input_arr[index_list[35:]]
test_target = target_arr[index_list[35:]]
```



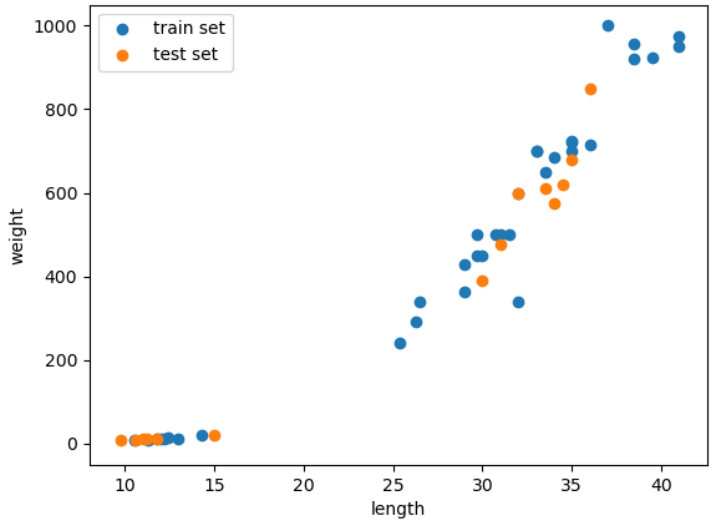
올바른 훈련 데이터

무작위로 섞인 데이터 확인하기

- 훈련 데이터와 테스트 데이터 다시 확인

```
import matplotlib.pyplot as plt
# [:, 0]: 모든 행에서 0번째 length, [:, 1]: 모든 행에서 1번째 weight
plt.scatter(train_input[:, 0], train_input[:, 1], label='train set')
plt.scatter(test_input[:, 0], test_input[:, 1], label='test set’)

plt.xlabel('length')
plt.ylabel('weight')
plt.legend()
plt.show()
```




```
[:, 0]   [:, 1]
[[25.4,  242.0]
 [26.3,  290.0]
   ⋮       ⋮
 [15.0,   19.9]]
```

length weight

두 번째 머신러닝 프로그램

- 무작위로 섞인 훈련 세트와 테스트 세트를 이용하여 k-최근접 이웃 모델 생성 및 훈련

  - fit() -> score() -> predict() 호출

```
kn = kn.fit(train_input, train_target)
print("score: ", kn.score(test_input, test_target))

print("predict:     ", kn.predict(test_input))
print("test_target: ", test_target) # predict 결과값과 비교를 위해 출력
```

```
score:  1.0
predict:      [0 0 1 0 1 1 1 0 1 1 0 1 1 0]
test_target:  [0 0 1 0 1 1 1 0 1 1 0 1 1 0]
```

100% 정확도

[0, ..0]: Numpy 배열
scikit-learn 모델의 입출력 값은 모두 Numpy의 배열

전체 소스 코드(chap02-1-1.py) #1

```
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

fish_length = [25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7, 30.0, 30.0, 30.7, 31.0, 31.0,
               31.5, 32.0, 32.0, 32.0, 33.0, 33.0, 33.5, 33.5, 34.0, 34.0, 34.5, 35.0,
               35.0, 35.0, 35.0, 36.0, 36.0, 37.0, 38.5, 38.5, 39.5, 41.0, 41.0, 9.8,
               10.5, 10.6, 11.0, 11.2, 11.3, 11.8, 11.8, 12.0, 12.2, 12.4, 13.0, 14.3, 15.0]
fish_weight = [242.0, 290.0, 340.0, 363.0, 430.0, 450.0, 500.0, 390.0, 450.0, 500.0,
               475.0, 500.0, 500.0, 340.0, 600.0, 600.0, 700.0, 700.0, 610.0, 650.0,
               575.0, 685.0, 620.0, 680.0, 700.0, 725.0, 720.0, 714.0, 850.0, 1000.0,
               920.0, 955.0, 925.0, 975.0, 950.0, 6.7,  7.5, 7.0, 9.7, 9.8,
               8.7, 10.0, 9.9, 9.8, 12.2, 13.4, 12.2, 19.7, 19.9]

# 길이와 무게를 합쳐 2차원 리스트 생성
fish_data = [[l, w] for l, w in zip(fish_length, fish_weight)]
fish_target = [1] * 35 + [0]*14 # target 값 생성

'''
Numpy 활용
'''
input_arr = np.array(fish_data)
target_arr = np.array(fish_target)

np.random.seed(42)
index_list = np.arange(49) # 0~48까지 배열 생성
np.random.shuffle(index_list) # 무작위로 섞음
```

전체 소스 코드(chap02-1.py) #2

```
'''
 랜덤하게 섞인 index_list의 값을 이용 input_arr에서 훈련용 데이터를 가져옴
 (올바른 훈련 데이터를 얻기 위함)
'''
# 훈련 세트 생성
train_input = input_arr[index_list[:35]]
train_target = target_arr[index_list[:35]]

# 테스트 세트 생성
test_input = input_arr[index_list[35:]]
test_target = target_arr[index_list[35:]]

import matplotlib.pyplot as plt
# [:, 0]: 모든 행에서 0번째 length, [:, 1]: 모든 행에서 1번째 weight
plt.scatter(train_input[:, 0], train_input[:, 1], label='train set')
plt.scatter(test_input[:, 0], test_input[:, 1], label='test set')
plt.xlabel('length')
plt.ylabel('weight')
plt.legend()
plt.show()

# KNeighborClassifier 객체 생성 및 모델 훈현
kn = KNeighborsClassifier()
kn = kn.fit(train_input, train_target)
print("score: ", kn.score(test_input, test_target))
print("predict:     ", kn.predict(test_input))
print("test_target: ", test_target) # predict 결과값과 비교를 위해 출력
```

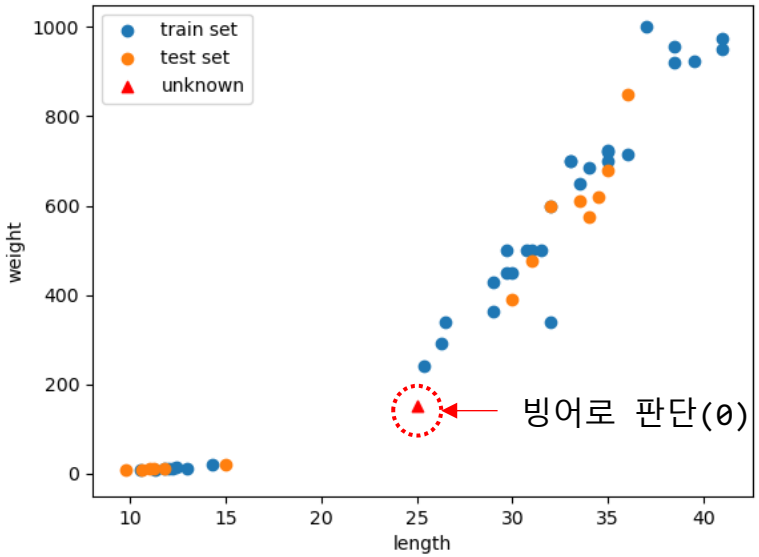
문제점 발생

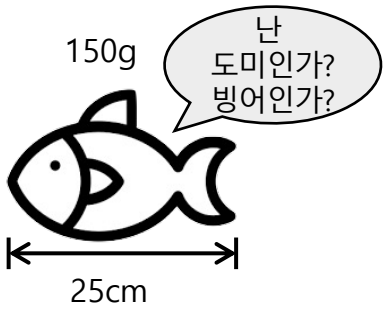
- 2-1 머신러닝 프로그램의 문제점 발생
  - 길이 25cm, 무게 150g인 도미를 빙어로 판단

```
print("predict(길이 25cm, 무게 150g): ", kn.predict([[25, 150.0]]))
```

```
predict(길이 25cm, 무게 150g):  [0]
```

  - 샘플 데이터의 두 특성(길이, 무게)의 스케일이 다르기 때문
    - 스케일을 조정해야 됨: 표준점수로 변환해야 됨
  - 표준 점수: 평균과의 거리





Numpy를 활용한 데이터 전처리 #1

- column_stack()
  - Numpy 제공 함수: 리스트를 일렬로 세운 다음 차례대로 연결
  - Python에서 제공하는 zip()함수와 동일한 기능

```
# column_stack((list1, list2)): 튜플 형태로 전달
fish_data = np.column_stack((fish_length, fish_weight))
print(fish_data[:5])
```

```
[[ 25.4 242. ]
 [ 26.3 290. ]
 [ 26.5 340. ]
 [ 29.  363. ]
 [ 29.  430. ]]
```

- ones(n), zeros(n)
  - 개수(n)만큼 각각 1과 0으로 채운 배열을 생성함

```
print(np.ones(35))
```

```
[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. 1. 1. 1. 1.]
```

```
print(np.zeros(14))
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Numpy를 활용한 데이터 전처리 #2

- concatenate((a1, a2, ...))
  - a1, a2 배열을 서로 연결함 (a1 배열 다음에 a2 배열 연결)
  - 연결할 배열을 튜플로 전달

```
fish_target = np.concatenate((np.ones(35), np.zeros(14)))
print(fish_target)
```

```
[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. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0.]
```

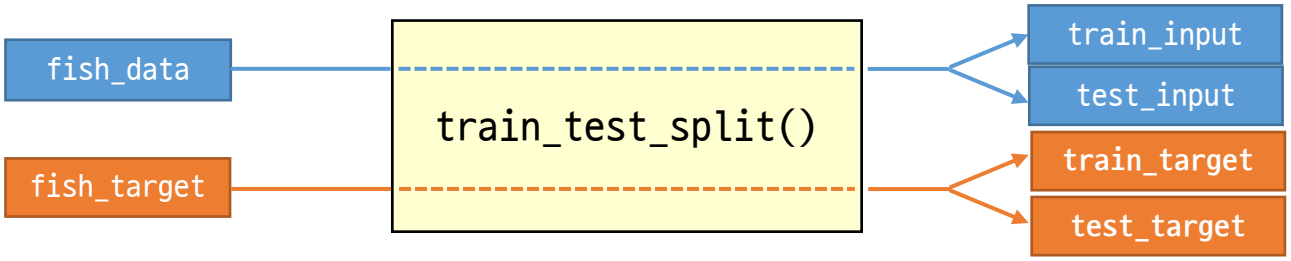
사이킷런으로 훈련 세트와 테스트 세트 나누기 #1

- 사이킷런으로 훈련 세트와 테스트 세트 나누기
  - train_test_split() 함수 원형

```
sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)
```

    - 전달되는 리스트나 배열을 비율에 맞게 훈련 세트와 테스트 세트로 나눔
    - 총 4개의 배열이 반환됨

  - 함수 파라미터 내용
    - **test_size**: 0.0 ~ 1.0 사이
      - None: 자동으로 25% 비율로 테스트 세트 생성
    - **train_size**: 0.0 ~ 1.0 사이
      - None: 자동으로 75% 비율로 훈련 세트 생성
    - **random_state**: 랜덤 시드 설정
    - **stratify**: target 데이터를 전달하면, 클래스 비율(0, 1)에 맞게 데이터를 나눔



사이킷런으로 훈련 세트와 테스트 세트 나누기 #1

- 사이킷런으로 훈련 세트와 테스트 세트 나누기
  - 4개의 데이터 셋을 리턴

```
from sklearn.model_selection import train_test_split

train_input, test_input, train_target, test_target = train_test_split(
    fish_data, fish_target, stratify=fish_target, random_state=42)

print('train_input.shape:', train_input.shape,
      'test_input.shape', test_input.shape)
```

```
train_input.shape: (36, 2) test_input.shape (13, 2)
```

```
print('train_target: ', train_target)
```

```
train_target: [1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 0. 1.
1. 0. 1. 0. 1. 0. 1. 1. 1. 0. 1. 1. 0. 1. 1.]
```

```
print('test_target: ', test_target)
```

```
train_target: [1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 0. 1.
1. 0. 1. 0. 1. 0. 1. 1. 1. 0. 1. 1. 0. 1. 1.]
test_target:  [0. 0. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1.]
```

가장 가까운 이웃 찾기 #1

- k-최근접 이웃 모델 적용
  - train_test_split()함수로 나눈 데이터 사용

```
from sklearn.neighbors import KNeighborsClassifier

kn = KNeighborsClassifier()
kn.fit(train_input, train_target)

print("score: ", kn.score(test_input, test_target))
```

```
score:  1.0
```

```
# 25cm, 150g의 생선 분류 (수상한 도미)
print(kn.predict([[25, 150]]))
```

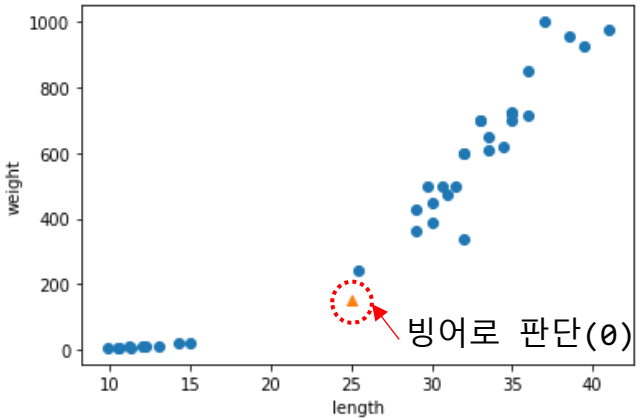
```
[0.] # 빙어로 분류
```

가장 가까운 이웃 찾기 #2

- 길이 25cm, 무게 150g의 생선을 산점도로 표시

```
import matplotlib.pyplot as plt

plt.scatter(train_input[:,0], train_input[:,1])
plt.scatter(25, 150, marker='^')
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



- 가장 가까운 이웃과의 거리 계산
  - kneighbors(): n_neighbors(default값 5)개의 이웃을 찾음

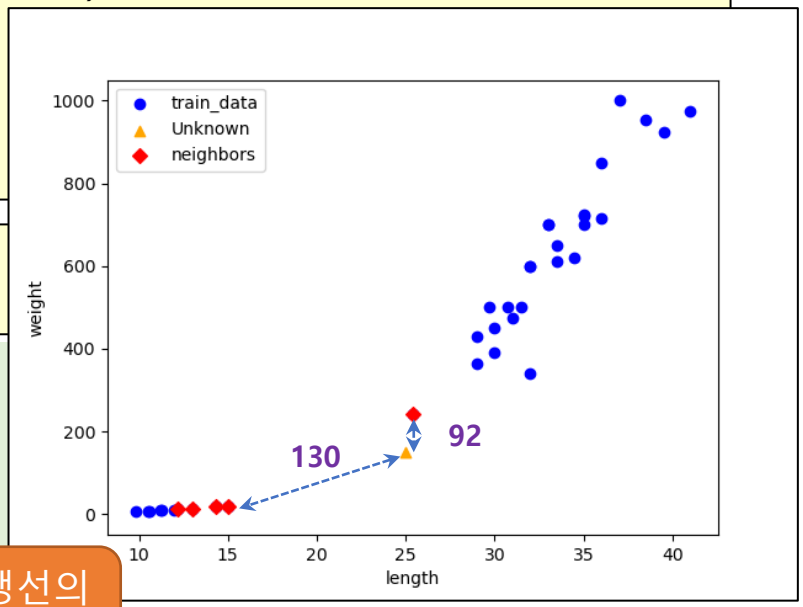
```
distances, indexes = kn.kneighbors([[25, 150]])
print(distances)
print(indexes)
```

```
[[ 92.00086956 130.48375378 130.73859415 138.32150953 138.39320793]]
[[21 33 19 30  1]]
```

가장 가까운 이웃 찾기 #2

- 가장 가까운 이웃 5개를 산점도로 표시

```
# 가장 가까운 이웃들을 산점도로 표시
import matplotlib.pyplot as plt
plt.scatter(train_input[:, 0], train_input[:, 1], label='train_data',
            color='b')
plt.scatter(25, 150, marker='^', color='orange', label='Unknown')
plt.scatter(train_input[indexes, 0], train_input[indexes, 1],
            marker='D', label='neighbors', color='r')
plt.xlabel('length')
plt.ylabel('weight')
plt.legend()
plt.show()
```




```
print("neighbors: ", train_input[indexes])
print("target: ", train_target[indexes])
```

```
neighbors:  [[[ 25.4 242. ]
  [ 15.   19.9]
  [ 14.3  19.7]
  [ 13.   12.2]
  [ 12.2  12.2]]]
target:  [[1. 0. 0. 0. 0.]]
```

(25cm, 150g) 생선의 이웃 4개는 빙어

```
print(distances)
```

```
[[ 92.00086956 130.48375378 130.73859415 138.32150953 138.39320793]]
```

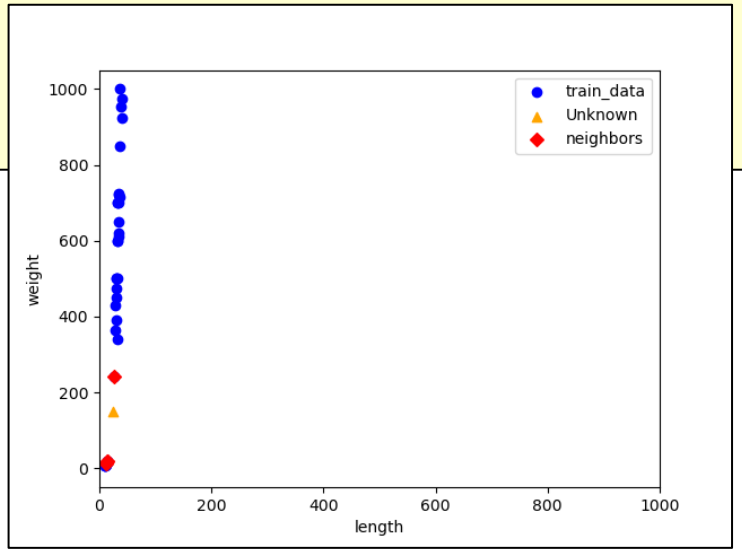
Scale을 동일하게 수정

- x축과 y축의 범위를 동일하게 수정
  - 두 특성(길이, 무게)의 scale이 다름
  - xlim(), ylim()함수: x, y축의 범위를 지정

```
plt.scatter(train_input[:, 0], train_input[:, 1],
            label='train_data', color='b')
plt.scatter(25, 150, marker='^', color='orange', label='Unknown')
plt.scatter(train_input[indexes, 0], train_input[indexes, 1],
            marker='D', label='neighbors', color='r')

plt.xlim((0, 1000)) # x축의 눈금 간격을 y축과 동일하게 변경
plt.xlabel('length')
plt.ylabel('weight')
plt.legend()
plt.show()
```

생선의 무게에 따라
분류에 큰 영향을 미침



데이터 전처리 (Data Preprocessing) #1

- 데이터 전처리
  - 특성값을 일정한 기준으로 맞춤
  - 표준 점수 (Standard Score, Z 점수)
    - 각 특성(길이, 무게)의 값이 0에서 표준편차의 몇 배만큼 떨어져 있는지를 나타냄

$$Z = \frac{x - \mu}{\sigma} \quad (x: \text{특성값}, \mu: \text{평균}, \sigma: \text{표준편차})$$

- 평균 및 표준편차 계산
  - 브로드 캐스팅
    - 행(row)이나 열(column)의 모든 데이터에 대해 한 번에 연산을 수행

```
mean = np.mean(train_input, axis=0) # axes=0: 각 컬럼의 평균 계산
std = np.std(train_input, axis=0)
print(mean, std)
```

```
[ 27.29722222 454.09722222] [  9.98244253 323.29893931]
```

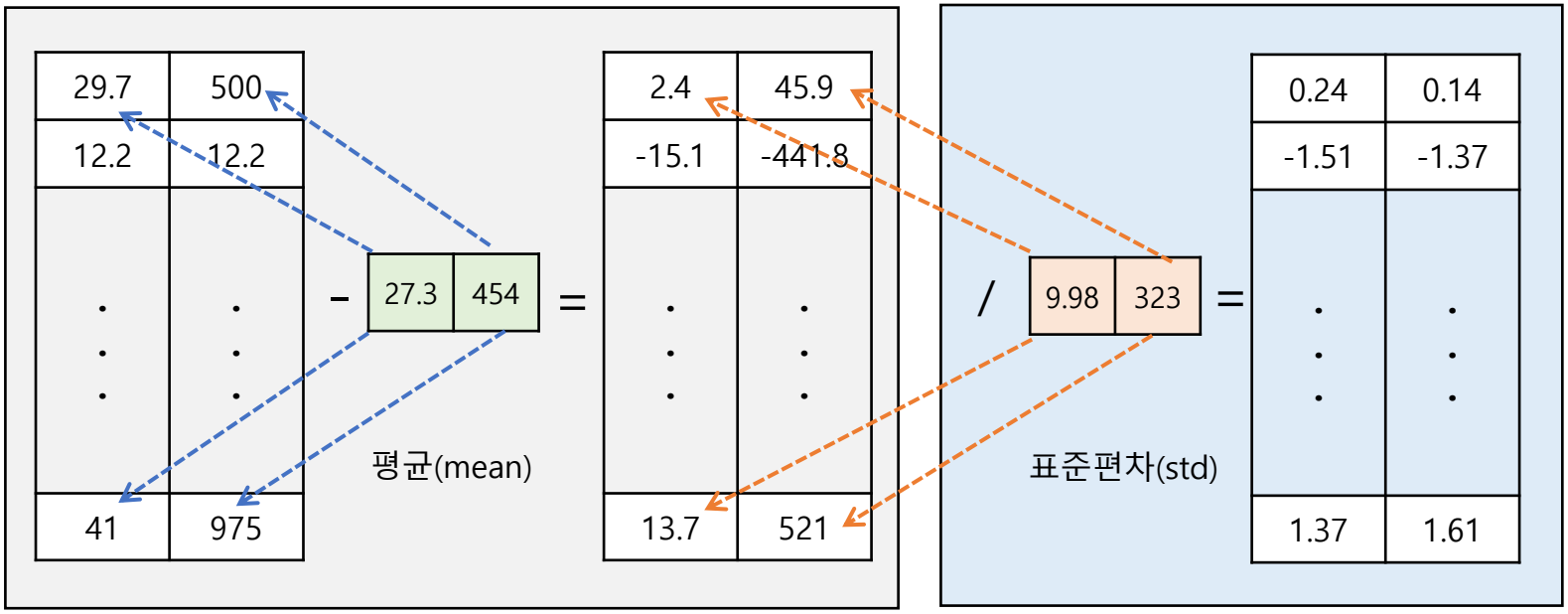
length 평균 | weight 평균 | length 표준 편차 | weight 표준 편차

데이터 전처리 (Data Preprocessing) #2

- 표준 점수(Z값) 계산 $Z = \frac{x - \mu}{\sigma}$ (x: 특성값, μ: 평균, σ: 표준편차)

```
train_scaled = (train_input - mean) / std  # 표준 점수로 변환한 데이터
print(train_scaled[:3])
```

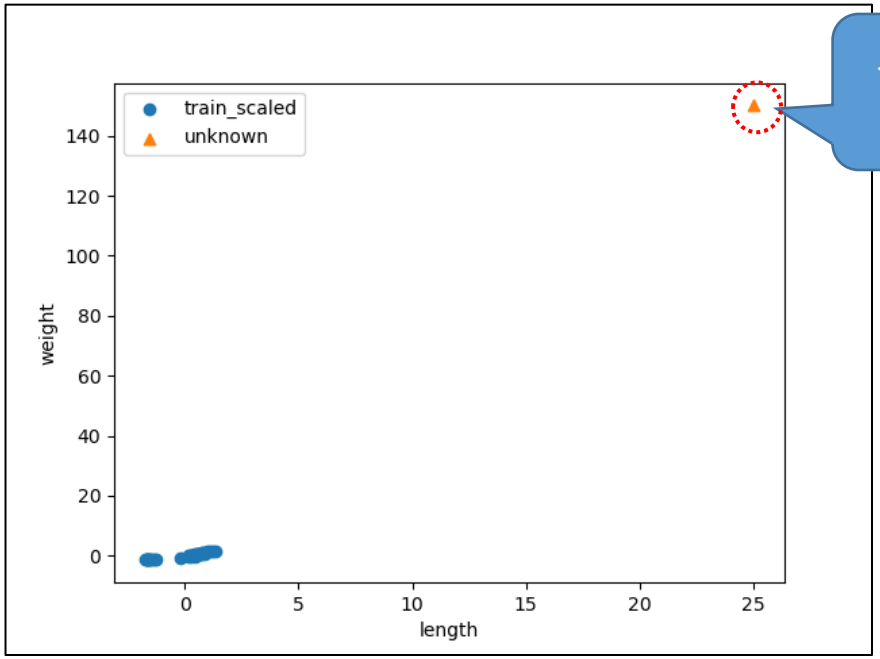
```
[[ 0.24070039  0.14198246]
 [-1.51237757 -1.36683783]
 [ 0.5712808   0.76060496]]
```



전처리 데이터로 모델 훈련 #1

- 표준 점수로 변환한 훈련 데이터 분포 확인
  - train_scaled 데이터와 샘플 데이터(25, 150) 비교

```
plt.scatter(train_scaled[:, 0], train_scaled[:, 1], label='train_scaled')
plt.scatter(25, 150, marker='^', label='unknown')
plt.xlabel('length')
plt.ylabel('weight')
plt.legend()
plt.show()
```



샘플 데이터(25, 150)은
표준 점수가 아님

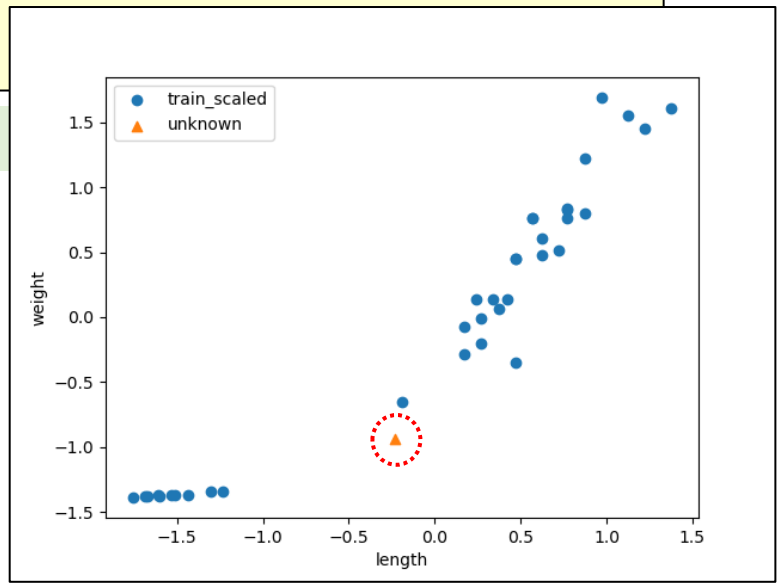
전처리 데이터로 모델 훈련 #2

- 샘플 데이터를 표준 점수로 변환

```
# 샘플 데이터를 표준 점수로 변환
new = ([25, 150] - mean) / std
print("표준 점수 sample", new)

plt.scatter(train_scaled[:, 0], train_scaled[:, 1], label='train_scaled')
plt.scatter(new[0], new[1], marker='^', label='unknown')
plt.xlabel('length')
plt.ylabel('weight')
plt.legend()
plt.show()
```

```
표준 점수 sample [-0.23012627 -0.94060693]
```



전처리 데이터로 모델 훈련 #3

- 표준 점수로 변환한 훈련 데이터로 k-최근접 이웃 모델 훈련

```
# 표준점수로 변환한 훈련 데이터로 k-NN 훈련
kn.fit(train_scaled, train_target)

# 테스트 세트를 표준 점수로 변환
test_scaled = (test_input - mean) / std
print(kn.score(test_scaled, test_target))

# 표준 점수로 변환된 샘플 데이터를 가지고 예측
print(kn.predict([new]))
```

```
1.0
[1.]
```

- 결과
  - 길이가 25cm, 무게가 150g인 생선을 도미로 인식함

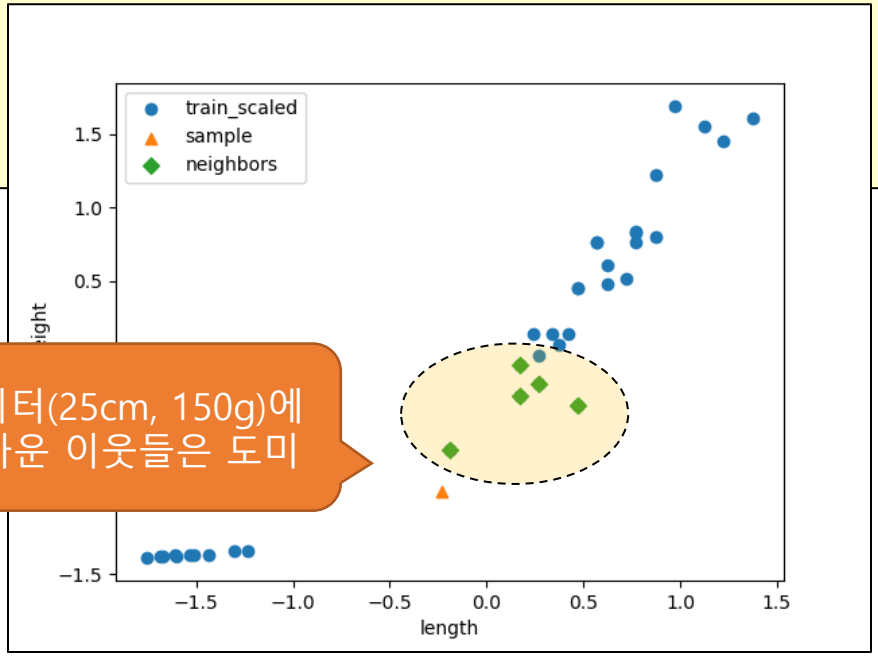
최종 이웃 찾기 및 산점도

```
# 표준 점수로 변경된 샘플 데이터의 이웃 찾기 및 산점도
distances, indexes = kn.kneighbors([new])
print(distances)

plt.scatter(train_scaled[:, 0], train_scaled[:, 1], label='train_scaled')
plt.scatter(new[0], new[1], marker='^', label='sample')
plt.scatter(train_scaled[indexes, 0], train_scaled[indexes, 1],
            marker='D', label='neighbors')

plt.xlabel('length')
plt.ylabel('weight')
plt.legend()
plt.show()
```

```
[[0.2873737 0.7711188 0.89552179
0.91493515 0.95427626]]
```



샘플 데이터(25cm, 150g)에
가장 가까운 이웃들은 도미

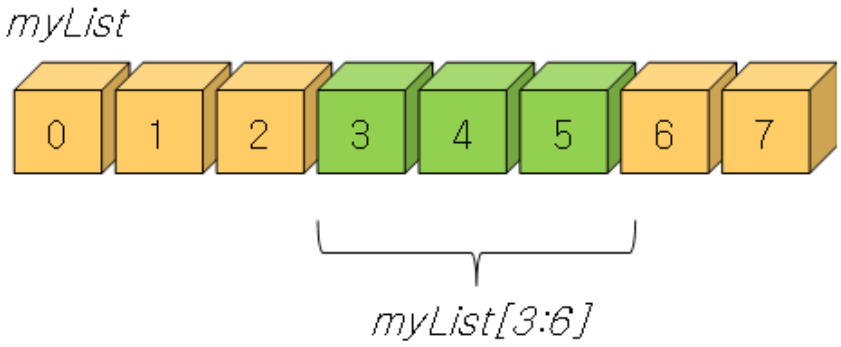
마무리 정리

- 특성의 스케일이 다른 경우
  - 길이보다 무게에 따라 예측값이 변경되었음
- 스케일이 다른 특성 처리
  - 훈련 세트 및 테스트 세트를 표준 점수로 변환
- 데이터 전처리
  - 머신러닝 모델에 훈련 데이터를 입력하기 전에 가공하는 단계
- scikit-learn 함수 정리
  - `train_test_split()`
    - 입력 데이터를 훈련 세트와 테스트 세트로 자동으로 나눔
    - 테스트 세트의 기본 크기: 25%
  - `kneighbors()`
    - k-최근접 이웃 모델에서 입력한 샘플 데이터와 가장 가까운 이웃을 찾음
    - 거리와 인덱스 반환

Python 기본 문법

슬라이싱

- 슬라이싱(slicing)
  - 리스트 안에서 범위를 지정하여 원하는 요소들을 선택하는 연산
- 슬라이싱 사용법
  - myList[start : end]
  - myList[start]요소부터 **myList[end-1]** 요소까지 선택됨




```
squares = [0, 1, 4, 9, 16, 25, 36, 49]
print(squares[3:6])  # 슬라이싱은 새로운 리스트를 반환한다. (3, 4, 5)
[9, 16, 25]
```

Indexing/Slicing: 1차원 배열

- 범위 지정:
  - data[$from_{index}$: to_{index}]: to_{index}는 범위에 포함되지 않음
  - $from_{index}$는 생략 가능 (생략한 경우, 0으로 간주함)
  - to_{index}: 생략 가능 (생략한 경우, 마지막 인덱스로 설정)
- 1차원 배열
  - [:] : 인덱스 없이 콜론(:) 만 사용하면 전체 데이터를 선택

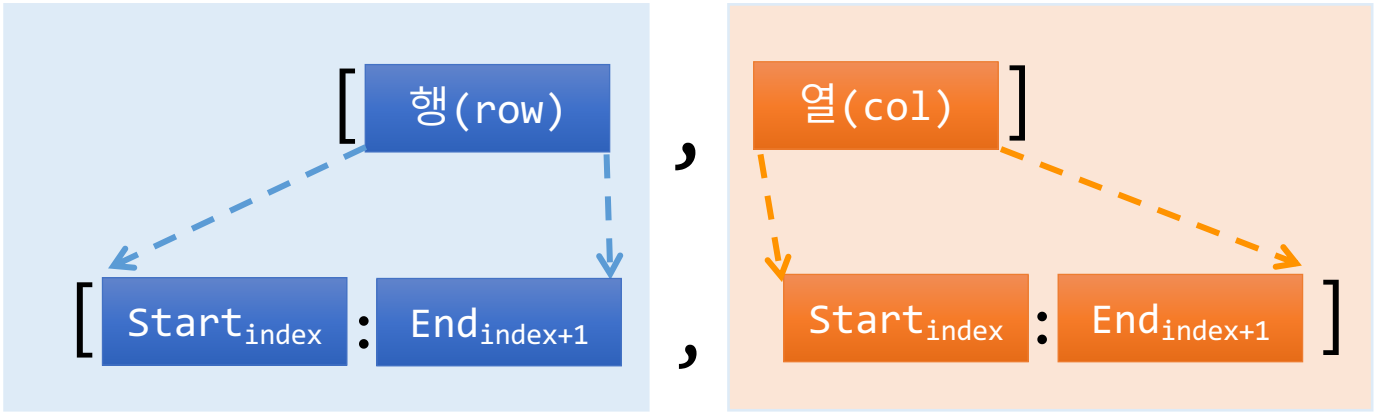
```
import numpy as np
data = np.array([1, 2, 3, 4, 5])

print(data[:]) # 전체 데이터 선택
print(data[:3]) # from 생략: index 0 ~ 2까지
print(data[0:1]) # index 0 에서 1까지 (1은 포함 안됨)
print(data[-2:]) # to 생략: index -2에서 끝까지
```

```
[1 2 3 4 5]
[1 2 3]
[1]
[4 5]
```

Indexing/Slicing: 2차원 배열 #1

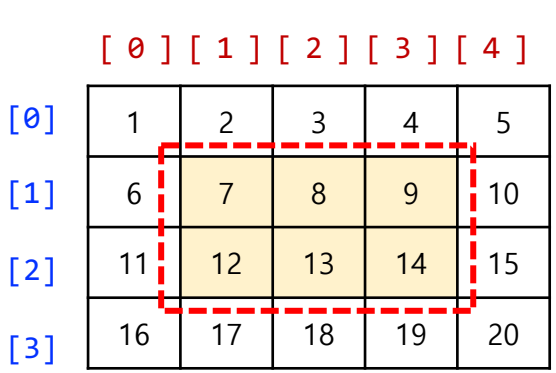
- Indexing/Slicing
  - 콤마를 기준으로 2개의 인자를 받음: [행(row), 열(col)]
  - 콜론(:)을 이용하여, 콜론 좌측은 시작 index, 우측은 마지막 index+1의 값이 대입
  - [row_start_{index}: $row\_end_{index+1}$, col_start_{index}: $col\_end_{index+1}$]
    - 예) [1:3, 1:4]

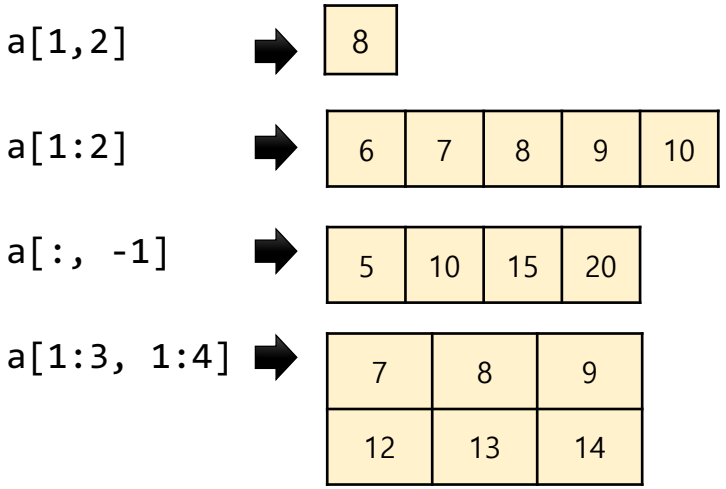


Indexing/Slicing: 2차원 배열 #2

- [row start$_{index}$: row end$_{index+1}$, col start$_{index}$: col end$_{index+1}$]
  - 예) [1:3, 1:4]

```
a = np.array([[ 1,  2,  3,  4,  5],
              [ 6,  7,  8,  9, 10],
              [11, 12, 13, 14, 15],
              [16, 17, 18, 19, 20]])
print(a[1, 2])        # 1행 2열의 원소 출력 (8)
print(a[1:2])         # 1행의 원소 출력 [6, 7, 8, 9, 10]
print(a[:, -1])       # 모든 행의 마지막 열 출력 [5, 10, 15, 20]
print(a[1:3, 1:4])    # 1~2행, 1~3열 출력
```





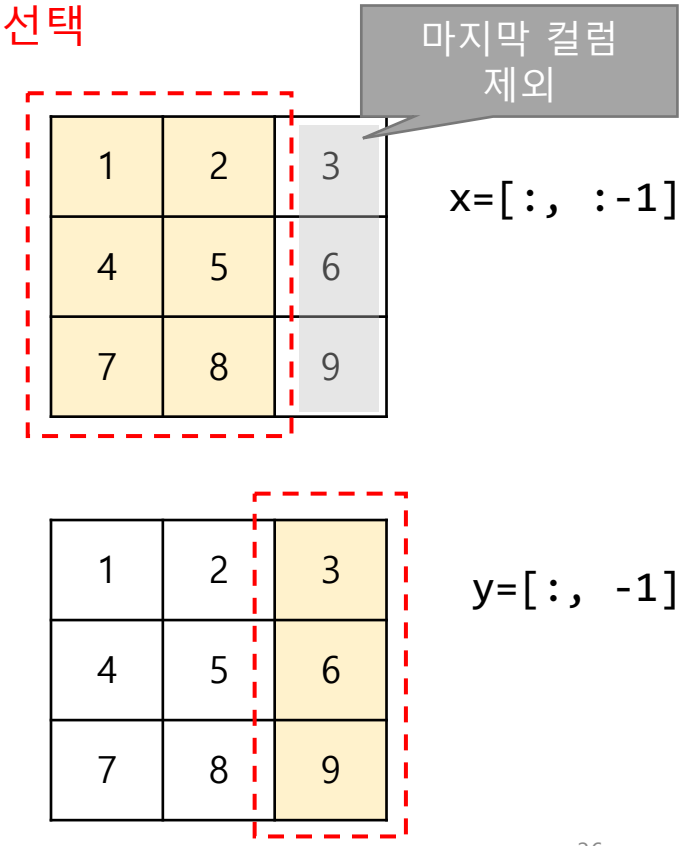
Indexing/Slicing: 2차원 배열 #3

- 2차원 배열 분리
  - `X = [: , :-1] or X = [:, :2]`
    - 2차원 배열에서 모든 행 선택, 마지막 컬럼을 제외한 모든 컬럼(컬럼 0, 1) 선택
  - `Y = [: , -1] or Y = [:, 2]`
    - 2차원 배열에서 모든 행 선택, 마지막 col만 선택

```
import numpy as np
data = np.array([[1, 2, 3],
                 [4, 5, 6],
                 [7, 8, 9]])
x, y = data[:, :-1], data[:, -1]
print(x)
print(y)
```

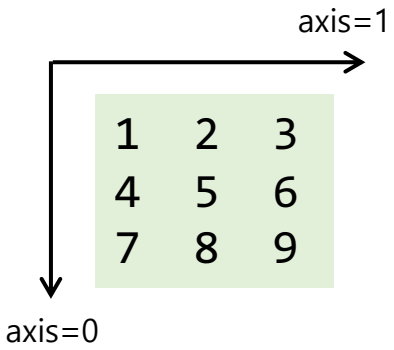
```
[[1 2]
 [4 5]
 [7 8]]

[3 6 9]
```



Numpy axis 사용

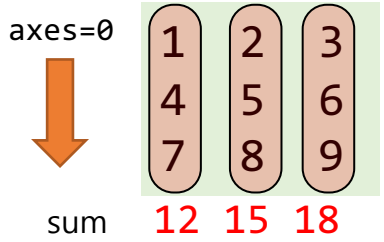
- axis(축) 구분
  - axis(축)을 따라 전체 데이터를 연산
  - axis=None: 전체 데이터
  - axis=0 (down)
    - 열(column) 단위 연산
  - axis=1 (across)
    - 행(row) 단위 연산

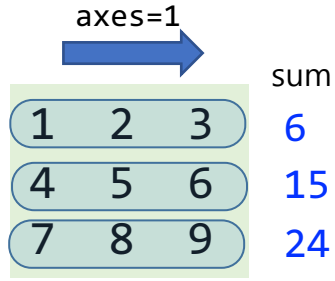



```
import numpy as np
array1 = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])
sum_col = np.sum(array1, axis=0) # axis=0, column(열) 단위 연산
print("column_sum:", sum_col)

sum_row = np.sum(array1, axis=1) # axis=1, row(행) 단위 연산
print("row sum:", sum_row)
```

```
column_sum: [12 15 18]
row sum:    [ 6 15 24]
```



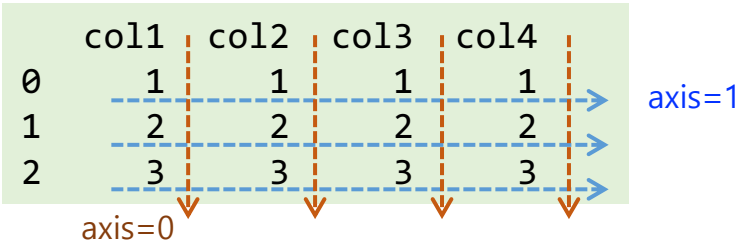


Pandas axis 연산 #1

- axis(축) 구분
  - axis(축)을 따라 전체 데이터를 연산
  - `axis=0` (down)
  - `axis=1` (across)

```
import pandas as pd

df = pd.DataFrame([[1, 1, 1, 1],
                   [2, 2, 2, 2],
                   [3, 3, 3, 3]], columns=['col1', 'col2', 'col3', 'col4’])
print(df)
```




```
print(df.mean(axis=1))
```

```
0    1.0
1    2.0
2    3.0
```

```
print(df.mean(axis=0))
```

```
col1    2.0
col2    2.0
col3    2.0
col4    2.0
```

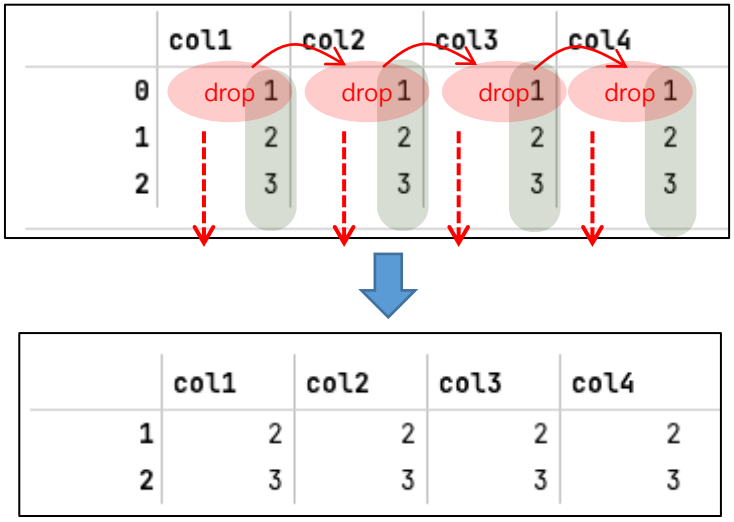
Pandas axis 연산 #2

```
import pandas as pd

df = pd.DataFrame([[1, 1, 1, 1],
                   [2, 2, 2, 2],
                   [3, 3, 3, 3]], columns=['col1', 'col2', 'col3', 'col4’])
print(df)
```

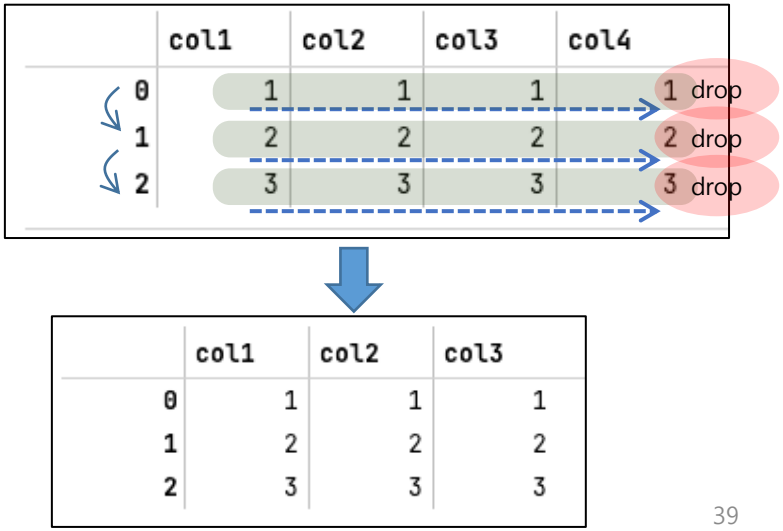
	col1	col2	col3	col4
0	1	1	1	1
1	2	2	2	2
2	3	3	3	3

```
df.drop(index=0, axis=0)
```



	col1	col2	col3	col4
1	2	2	2	2
2	3	3	3	3

```
df.drop('col4', axis=1)
```



	col1	col2	col3
0	1	1	1
1	2	2	2
2	3	3	3

Questions?