데이터 전처리


구글 코랩에서 실행하기

넘파이로 데이터 준비하기

In [2]:
```
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]
```

In [3]:
```
import numpy as np
```

In [9]:
```
result=np.column_stack(([1,2,3], [4,5,6])) # The elements of each lists form a row
print(result.shape)
a = [1,2,3]
b = [4,5,6]
result1 = np.array([(i,j) for i,j in zip(a,b)])
print(result==result1)
```

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

In [10]:
```
fish_data = np.column_stack((fish_length, fish_weight))
```

In [11]:
```
print(fish_data[:5])
```

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

In [14]:
```
print(np.ones(5))
```

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

In [15]:
```
fish_target = np.concatenate((np.ones(35), np.zeros(14))) # default value of dim is 0
```

In [16]:
```
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. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0.]
```

사이킷런으로 훈련 세트와 테스트 세트 나누기

In [17]:
```
from sklearn.model_selection import train_test_split
# 비율에 맞게 train, test 세트로 나눠줌
```

In [18]:
```
train_input, test_input, train_target, test_target = train_test_split(
    fish_data, fish_target, random_state=42) # 3대 1, test 비율 : 25%
```

In [19]:
```
print(train_input.shape, test_input.shape) # 형태 출력
```

```
(36, 2) (13, 2)
```

In [ ]:
```
print(train_target.shape, test_target.shape) # print label's shape
```

```
(36,) (13,)
```

In [ ]:
```
print(test_target) # sampling bias
```

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

In [20]:
```
train_input, test_input, train_target, test_target = train_test_split(
    fish_data, fish_target, stratify=fish_target, random_state=42) # so using stratify : 도미와 빙어의 비율을 유지한
```

In [21]:
```
print(test_target)
```

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

수상한 도미 한마리

In [22]:
```
from sklearn.neighbors import KNeighborsClassifier

kn = KNeighborsClassifier()
kn.fit(train_input, train_target)
kn.score(test_input, test_target)
```

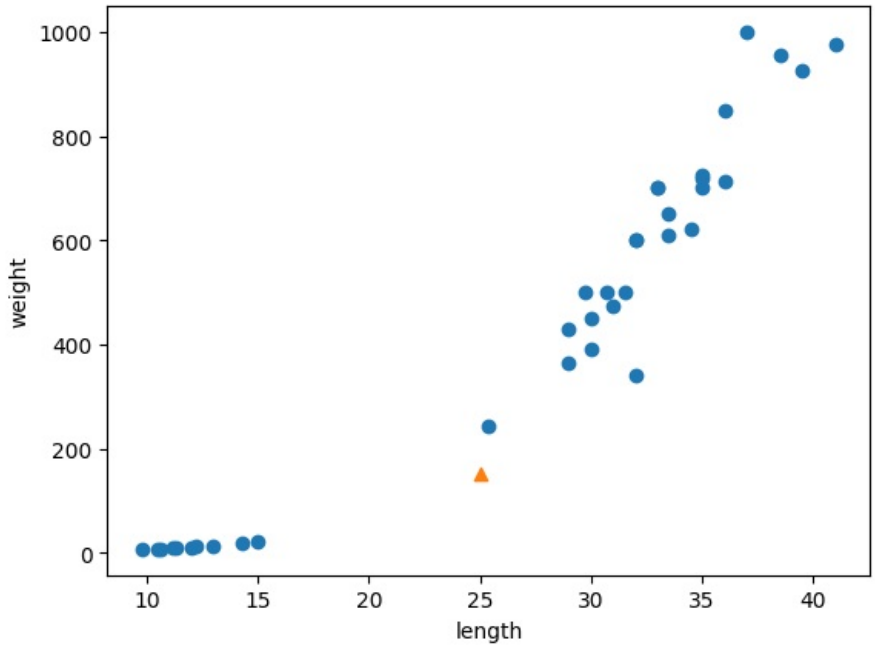
Out[22]:
```
1.0
```

In [23]:
```
print(kn.predict([[25, 150]])) # 도미 데이터를 넣고 결과 확인
```

```
[0.]
```

In [24]:
```
import matplotlib.pyplot as plt
```

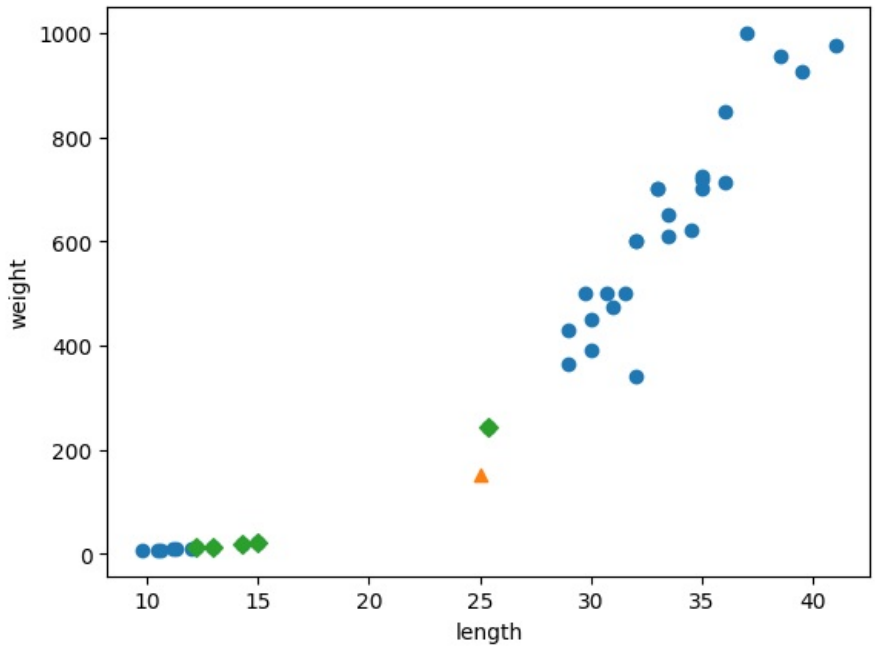
In [25]:
```
plt.scatter(train_input[:,0], train_input[:,1])
plt.scatter(25, 150, marker='^') # 이상한 도미
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



In [27]:
```
distances, indexes = kn.kneighbors([[25, 150]])
# KNeighborsClassifier 클래스는 주어진 샘플에서 가장 가까운 이웃을 찾아주는 Kneighbors() 메서드르 제공.
# 이 메서드는 이웃까지의 거리와 이웃 샘플의 인덱스를 반환
print(distances,indexes)
```

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

In [29]:
```
plt.scatter(train_input[:,0], train_input[:,1])
plt.scatter(25, 150, marker='^')
plt.scatter(train_input[indexes,0], train_input[indexes,1], marker='D') # 이상한 도미와 가까운 5개 데이터
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



In [ ]:
```
print(train_input[indexes])
```

```
[[[ 25.4 242. ]
  [ 15.   19.9]
  [ 14.3  19.7]
  [ 13.   12.2]
  [ 12.2  12.2]]]
```

In [ ]:
```
print(train_target[indexes])
```

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

In [ ]:
```
print(distances)
```

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

In [30]:
```
print(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]
```

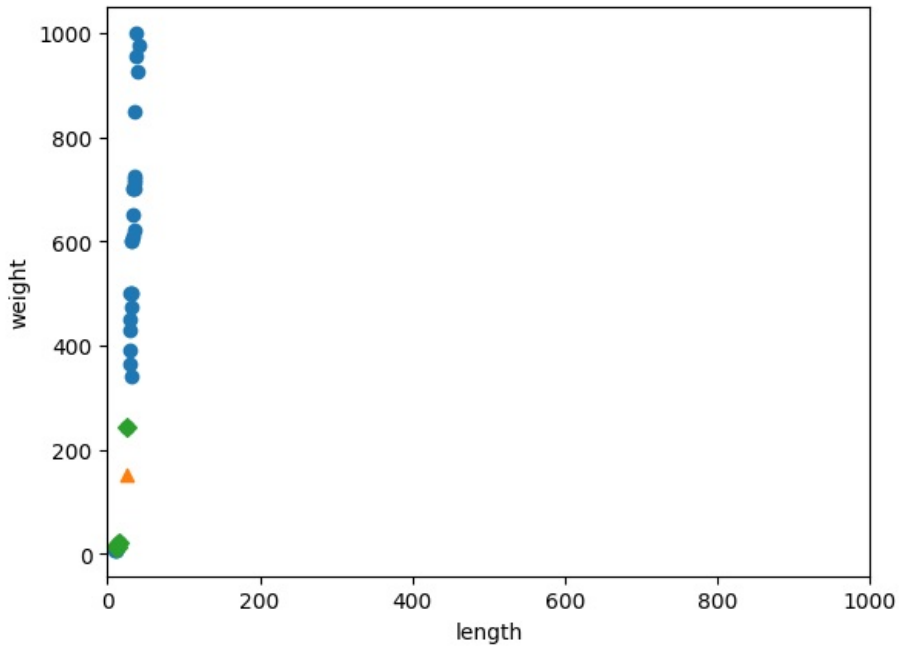
기준을 맞춰라

In [35]:
```
plt.scatter(train_input[:,0], train_input[:,1])
plt.scatter(25, 150, marker='^')
plt.scatter(train_input[indexes,0], train_input[indexes,1], marker='D')
plt.xlim((0, 1000))
plt.xlabel('length')
plt.ylabel('weight')
plt.show()

"""

x축과 y축의 범위를 동일하게 맞추니, 데이터가 수직으로 늘어선 형태를 보인
이는 생선의 길이(x축)은 가장 가까운 이웃을 찾는데 크게 영향을 못 줌
오로지 무게(y축)만을 고려 --> 특성값을 일정한 기준으로 맞추는 데이터 전처리가 필요
전처리 방법 중 하나인 표준 점수 사용

"""
```



Out[35]: '\n\nx축과 y축의 범위를 동일하게 맞추니, 데이터가 수직으로 늘어선 형태를 보인\n이는 생선의 길이(x축)은 가장 가까운 이웃을 찾는데 크게 영향을 못 줌\n오로지 무게(y축)만을 고려 --> 특성값을 일정한 기준으로 맞추는 데이터 전처리가 필요\n전처리 방법 중 하나인 표준 점수 사용\n\n'

In [41]:
```
mean = np.mean(train_input, axis=0)
std = np.std(train_input, axis=0)
print(train_input.shape)
```

```
(36, 2)
```

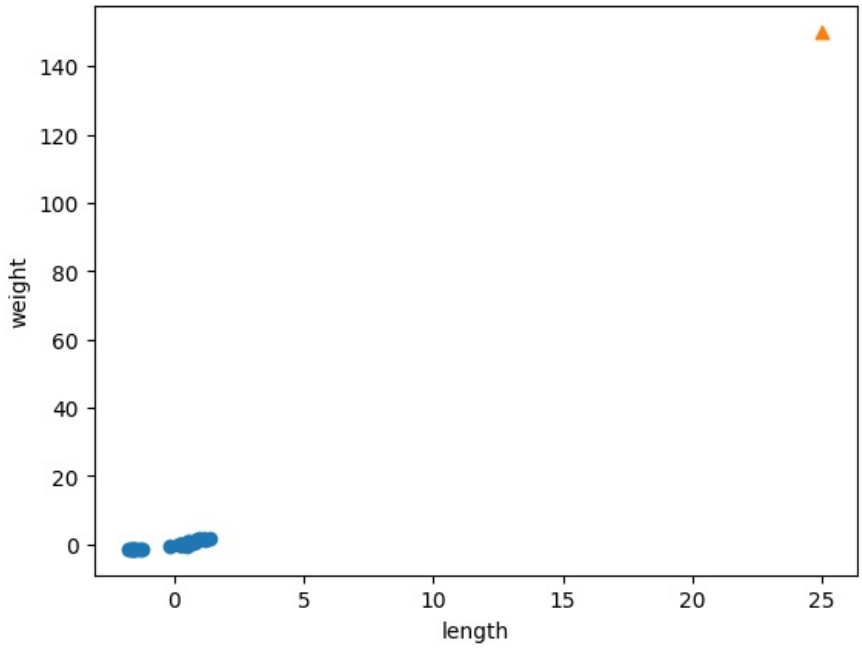
In [42]:
```
print(mean, std)
```

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

In [44]:
```
train_scaled = (train_input - mean) / std
```

전처리 데이터로 모델 훈련하기

In [45]:
```
plt.scatter(train_scaled[:,0], train_scaled[:,1])
plt.scatter(25, 150, marker='^')
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```

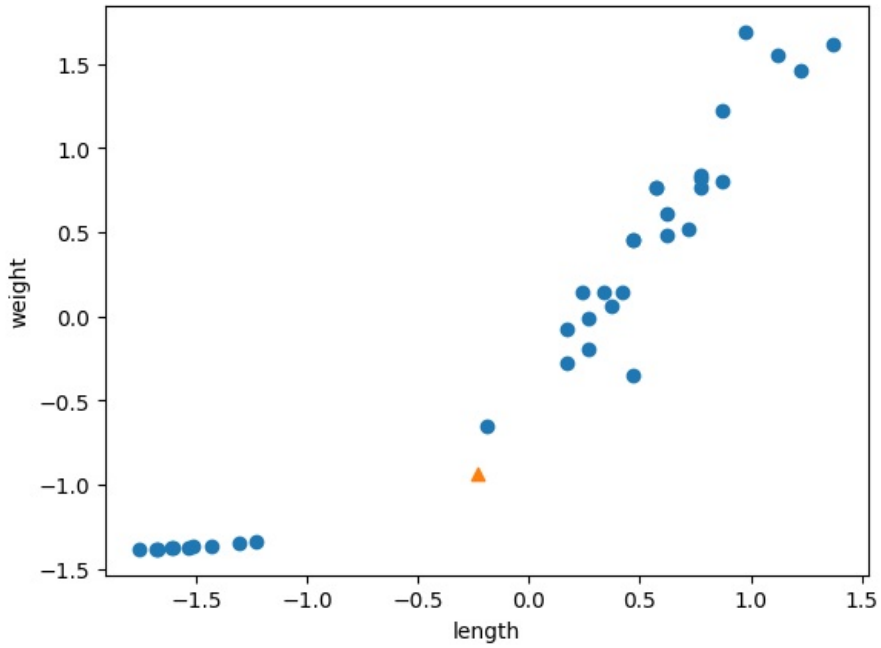


In [47]:
```
new = ([25, 150] - mean) / std
print(new)
```

```
[-0.23012627 -0.94060693]
```

In [48]:
```
plt.scatter(train_scaled[:,0], train_scaled[:,1])
plt.scatter(new[0], new[1], marker='^')
plt.xlabel('length')
```

```
plt.ylabel('weight')
plt.show()
```



In [49]:
```
kn.fit(train_scaled, train_target)
```

Out[49]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [50]:
```
test_scaled = (test_input - mean) / std # 테스트 데이터도 변환을 해야 함, KNN은 거리 기반이므로
```

In [51]:
```
kn.score(test_scaled, test_target)
```

Out[51]:
```
1.0
```

In [52]:
```
print(kn.predict([new]))
```

```
[1.]
```

In [54]:
```
distances, indexes = kn.kneighbors([new])
print(distances,indexes) # 변환되고나서 다시 확인
```

```
[[0.2873737  0.7711188  0.89552179 0.91493515 0.95427626]] [[21 14 34 32  5]]
```

In [55]:
```
plt.scatter(train_scaled[:,0], train_scaled[:,1])
plt.scatter(new[0], new[1], marker='^')
plt.scatter(train_scaled[indexes,0], train_scaled[indexes,1], marker='D')
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```