훈련 세트와 테스트 세트

지도 학습과 비지도 학습

- 지도 학습: 입력과 타겟이 모두 있음 -> 정답을 맞히는 것을 학습
- 비지도 학습: 입력은 있지만 타깃은 없음 -> 데이터를 잘 파악하거나 변형하는 데 도움을 줌

훈련 세트와 테스트 세트

- 테스트 세트: 평가에 사용하는 데이터
- 훈련 세트: 훈련에 사용되는 데이터
- 샘플: 하나의 데이터

In [1]:
```
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 [2]:
```
fish_data=[[l,w] for l,w in zip(fish_length,fish_weight)]
fish_target=[1]*35+[0]*14
```

In [3]:
```
from sklearn.neighbors import KNeighborsClassifier
kn=KNeighborsClassifier()
```

In [4]:
```
# 훈련 세트로 입력값 중 0부터 34번째 인덱스까지 사용
train_input=fish_data[:35]

# 훈련 세트로 타깃값 중 0부터 34번째 인덱스까지 사용
train_target=fish_target[:35]

# 테스트 세트로 입력값 중 35번째부터 마지막 인덱스까지 사용
test_input=fish_data[35:]

# 테스트 세트로 타깃값 중 35번째부터 마지막 인덱스까지 사용
test_target=fish_target[35:]
```

In [5]:
```
kn=kn.fit(train_input,train_target)
kn.score(test_input,test_target)
```

Out[5]:
```
0.0
```

샘플링 편향

- 샘플링 편향: 일반적으로 훈련 세트와 테스트 세트에 샘플이 골고루 섞여 있지 않으면 샘플링 한쪽으로 치우쳤다는 의미로 사용됨

넘파이

- 넘파이: 파이썬의 대표적인 배열 라이브러리
- 배열 인덱싱은 []가 아니라 [[]] 사용. []은 슬라이싱임

In [6]:
```
import numpy as np

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

print(input_arr) # 넘파이는 배열의 차원을 구분하기 쉽도록 행과 열을 가지런히 출력함
```

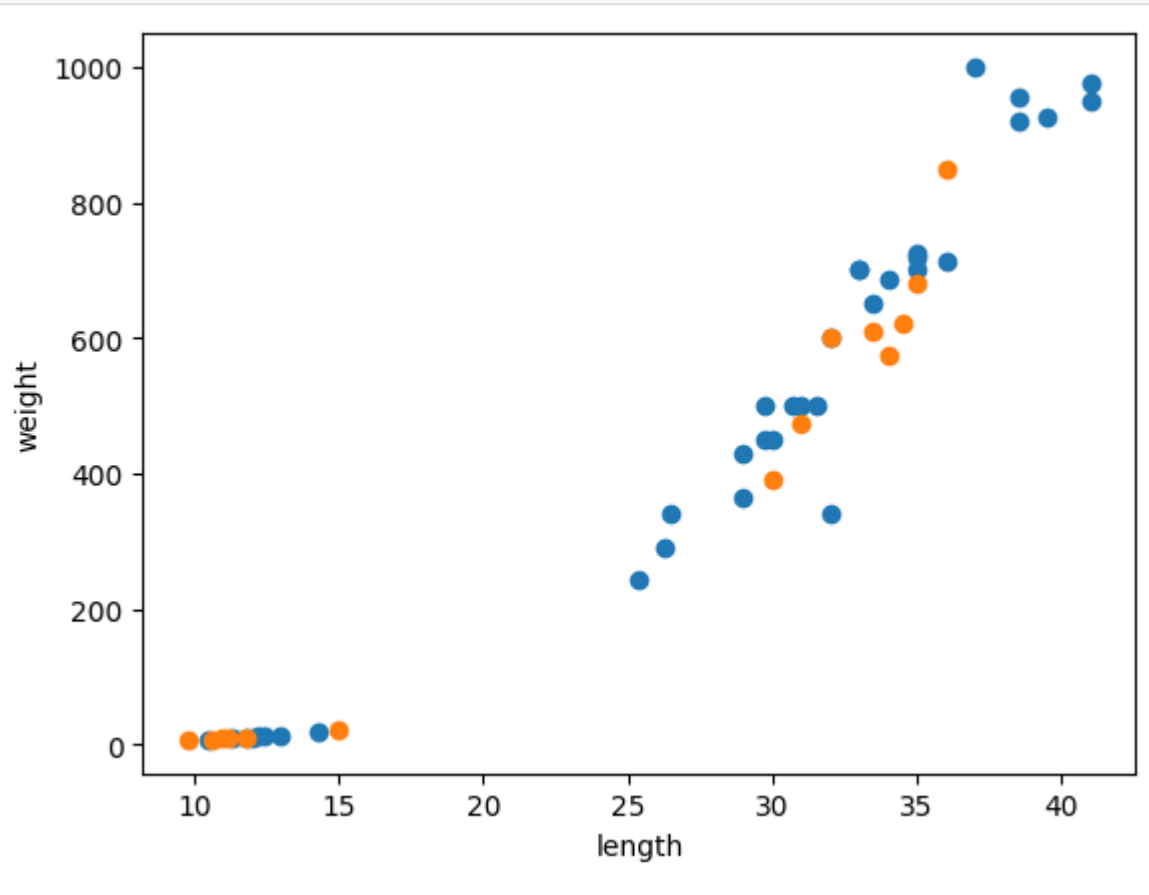
```
[[  25.4  242. ]
 [  26.3  290. ]
 [  26.5  340. ]
 [  29.   363. ]
 [  29.   430. ]
 [  29.7  450. ]
 [  29.7  500. ]
 [  30.   390. ]
 [  30.   450. ]
 [  30.7  500. ]
 [  31.   475. ]
 [  31.   500. ]
 [  31.5  500. ]
 [  32.   340. ]
 [  32.   600. ]
 [  32.   600. ]
 [  33.   700. ]
 [  33.   700. ]
 [  33.5  610. ]
 [  33.5  650. ]
 [  34.   575. ]
 [  34.   685. ]
 [  34.5  620. ]
 [  35.   680. ]
 [  35.   700. ]
 [  35.   725. ]
 [  35.   720. ]
 [  36.   714. ]
 [  36.   850. ]
 [  37.  1000. ]
 [  38.5  920. ]
 [  38.5  955. ]
 [  39.5  925. ]
 [  41.   975. ]
 [  41.   950. ]
 [   9.8    6.7]
 [  10.5    7.5]
 [  10.6    7. ]
 [  11.     9.7]
 [  11.2    9.8]
 [  11.3    8.7]
 [  11.8   10. ]
 [  11.8    9.9]
 [  12.     9.8]
 [  12.2   12.2]
 [  12.4   13.4]
 [  13.    12.2]
 [  14.3   19.7]
 [  15.    19.9]]
```

In [7]:
```
np.random.seed(42) # 넘파이의 랜덤 시드를 42로 설정, 난수를 생성하기 위한 정수 초깃값 지정
index=np.arange(49)
np.random.shuffle(index)

print(index)
```

```
[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]
```

In [8]:
```
train_input=input_arr[index[:35]]
train_target=target_arr[index[:35]]

test_input=input_arr[index[35:]]
test_target=target_arr[index[35:]]
```

In [9]:
```
import matplotlib.pyplot as plt
plt.scatter(train_input[:,0],train_input[:,1])
plt.scatter(test_input[:,0],test_input[:,1])
plt.xlabel("length")
plt.ylabel("weight")
plt.show()
```



두 번째 머신러닝 프로그램

In [10]:
```
kn=kn.fit(train_input,train_target)
kn.score(test_input,test_target)
```

Out[10]:
```
1.0
```

In [11]:
```
kn.predict(test_input) # 넘파이 배열로 출력
```

Out[11]:
```
array([0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0])
```

In [12]:
```
test_target
```

Out[12]:
```
array([0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0])
```

데이터 전처리

넘파이로 데이터 준비하기

- np.column_stack(): 전달받은 리스트를 일렬로 세운 다음 차례대로 나란히 연결
- np.concatenate(): 첫 번째 차원을 따라 배열을 연결

In [13]:
```
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 [14]:
```
import numpy as np

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

Out[14]:
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [15]:
```
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. ]]
```

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

사이킷런으로 훈련세트와 테스트 세트 나누기

- train_test_split(): 전달되는 리스트나 배열을 비율에 맞게 섞어서 훈련 세트와 테스트 세트로 나누어줌. 기본적으로 75%

In [17]:
```
from sklearn.model_selection import train_test_split

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

In [18]:
```
print(train_input.shape,test_input.shape)
```

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

In [19]:
```
print(test_target) # 샘플링 편향이 보임
```

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

In [20]:
```
# stratify 매개 변수는 클래스 비율에 맞게 데이터를 나눠줌
train_input, test_input, train_target, test_target = train_test_split(fish_data, fish_target,
                                                                      random_state=42, stratify=fish_target)
```

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

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

수상한 도미 한 마리

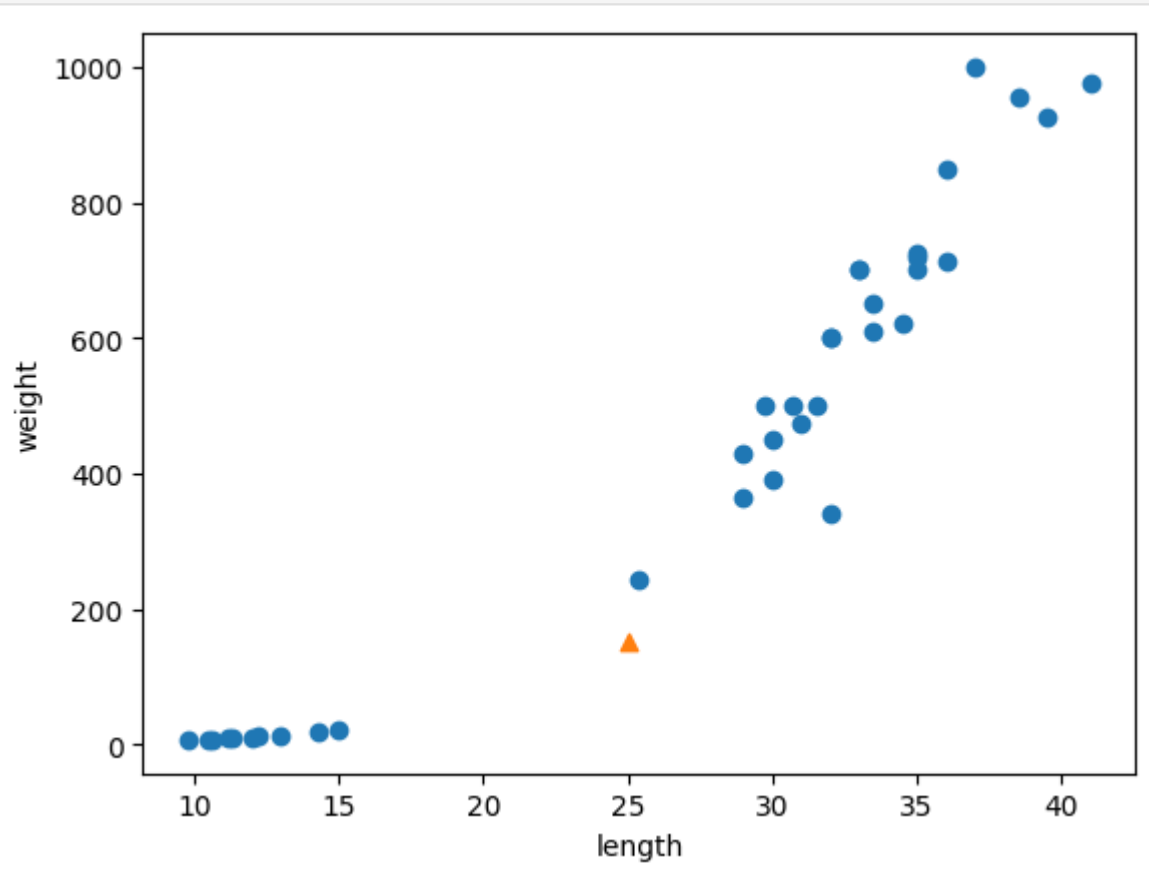
In [22]:
```
from sklearn.neighbors import KNeighborsClassifier
kn= KNeighborsClassifier()
kn.fit(train_input,train_target)
kn.score(test_input,test_target)
```

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

In [23]:
```
print(kn.predict([[25,150]]))
```

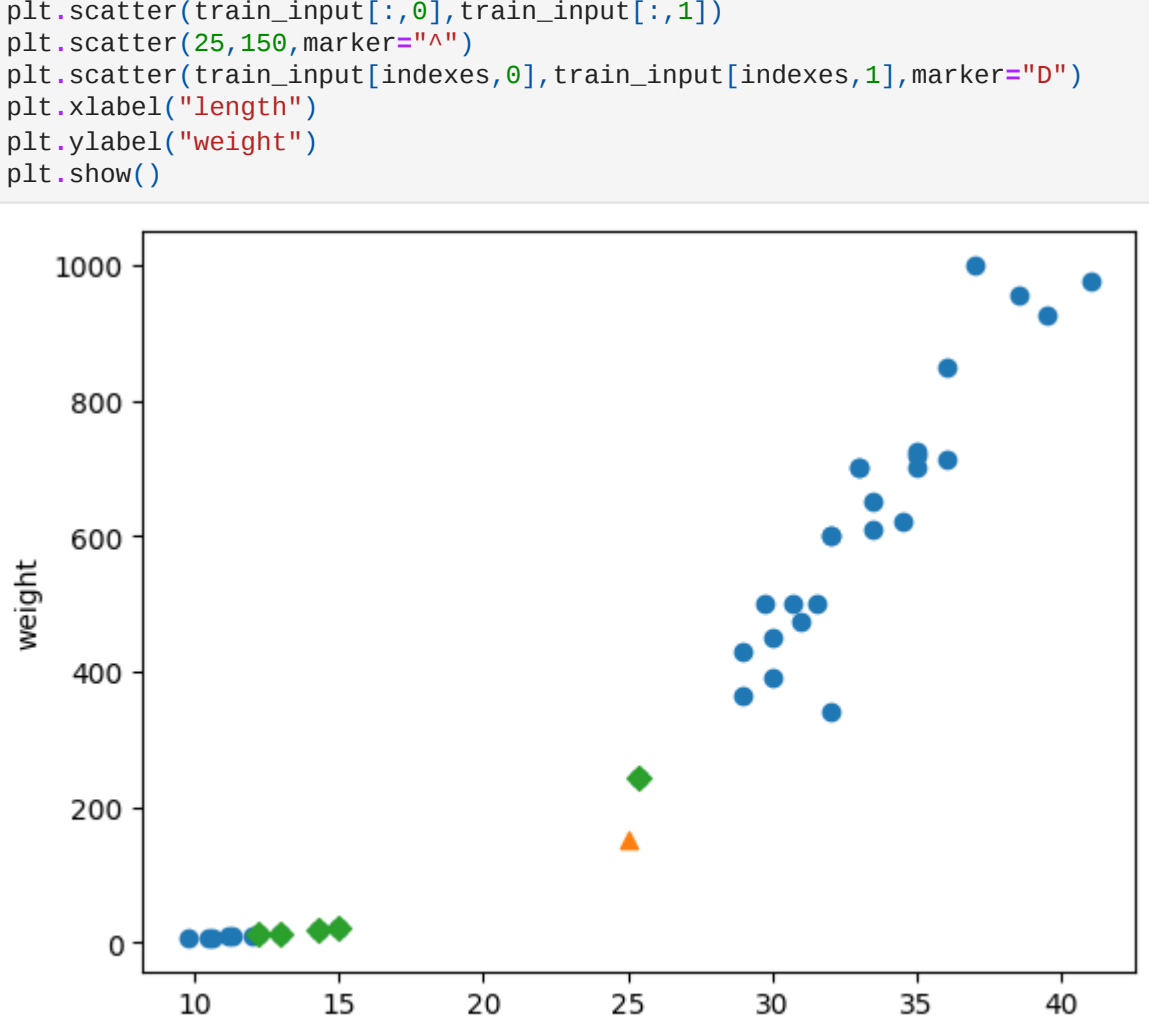
```
[0.]
```

In [24]:
```
import matplotlib.pyplot as plt
plt.scatter(train_input[:,0],train_input[:,1])
plt.scatter(25,150,marker="^") # marker 매개변수는 모양을 지정함
plt.xlabel("length")
plt.ylabel("weight")
plt.show()
```



In [25]:
```
distances, indexes = kn.kneighbors([[25,150]]) # kneighbors() 메서드는 샘플에서 가장 가까운 이웃의 거리와 인덱스를 알려줌
```

In [26]:
```
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.xlabel("length")
plt.ylabel("weight")
plt.show()
```



In [27]:
```
print(distances)
```

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

기준을 맞춰라

- 두 특성의 값이 놓인 범위가 매우 다를 수 있음(스케일이 다를 수 있음)
- 표준 점수(z 점수)를 이용할 수 있음
- 표준 점수: 각 특성값이 평균에서 표준편차의 몇 배 만큼 떨어져 있는지 나타냄

In [28]:
```
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()
```

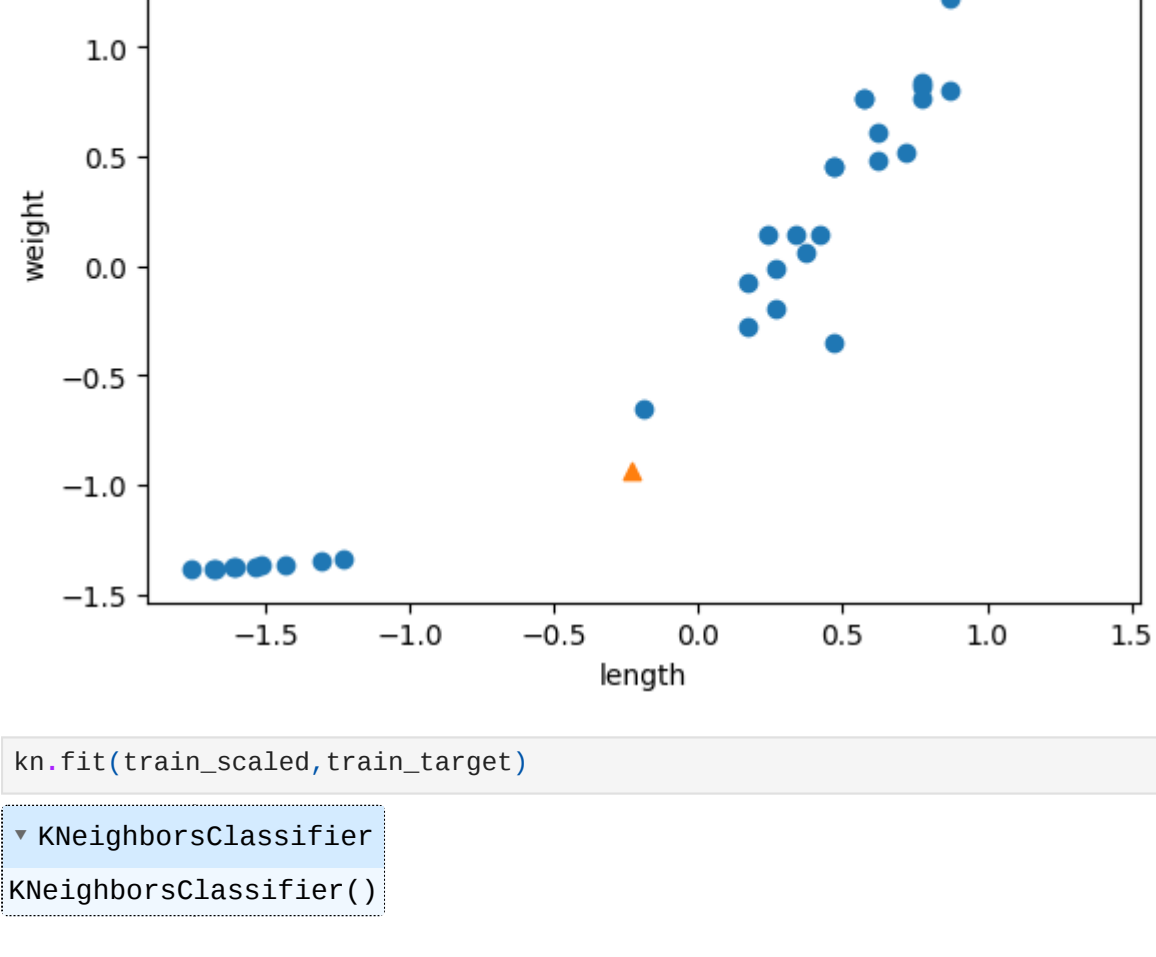
In [29]:
```
mean=np.mean(train_input,axis=0) # axis=0은 세로 축을 따라 계산하겠다는 뜻
std=np.std(train_input,axis=0)

print(mean,std)
```

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

In [30]:
```
train_scaled=(train_input-mean)/std
```

전처리 데이터로 모델 훈련하기

In [31]:
```
new=([25,150]-mean)/std
plt.scatter(train_scaled[:,0],train_scaled[:,1])
plt.scatter(new[0],new[1],marker="^")
plt.xlabel("length")
plt.ylabel("weight")
plt.show()
```



In [32]:
```
kn.fit(train_scaled,train_target)
```

Out[32]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [33]:
```
test_scaled=(test_input-mean)/std
kn.score(test_scaled,test_target)
```

Out[33]:
```
1.0
```

In [34]:
```
print(kn.predict([new]))
```

```
[1.]
```

In [35]:
```
distances, indexes=kn.kneighbors([new])
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()
```