



```
1 bream_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]
2 bream_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,
```

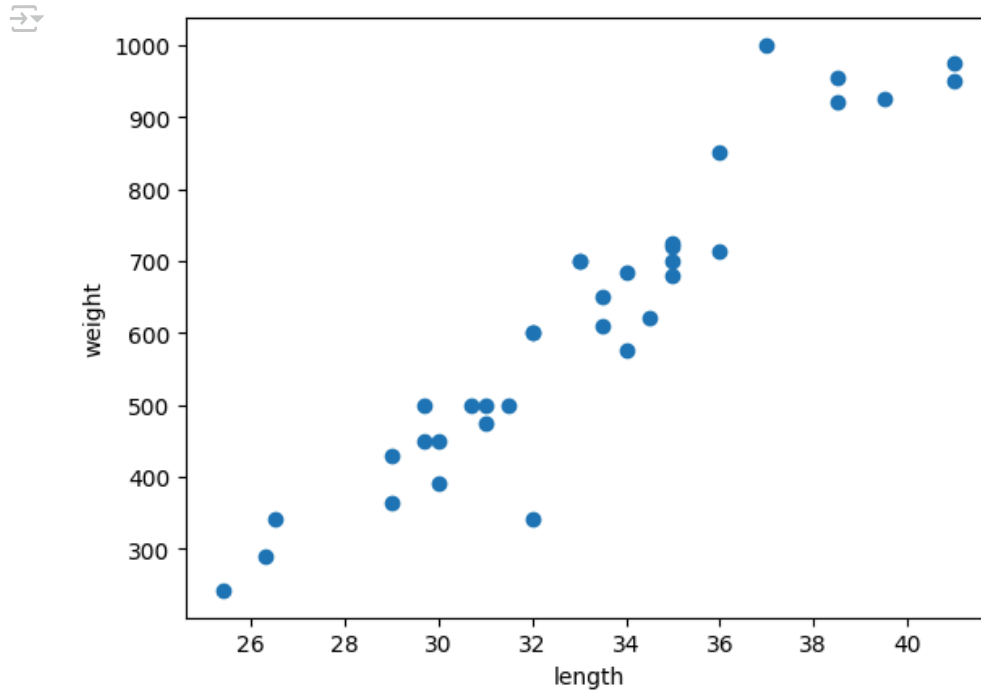
→ 35
35

<https://colab.research.google.com/drive/1e79foxmTCjH1-519BJLd1HfT8Xdmwdlt>

```

3
4 import matplotlib.pyplot as plt
5 plt.scatter(bream_length, bream_weight)
6 plt.xlabel('length')
7 plt.ylabel('weight')
8 plt.show()

```



✓ 빙어 데이터 준비하기(14마리)

```

1 smelt_length = [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]
2 smelt_weight = [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]

```

```

1 print(len(smelt_length))
2 print(len(smelt_weight))

```

```

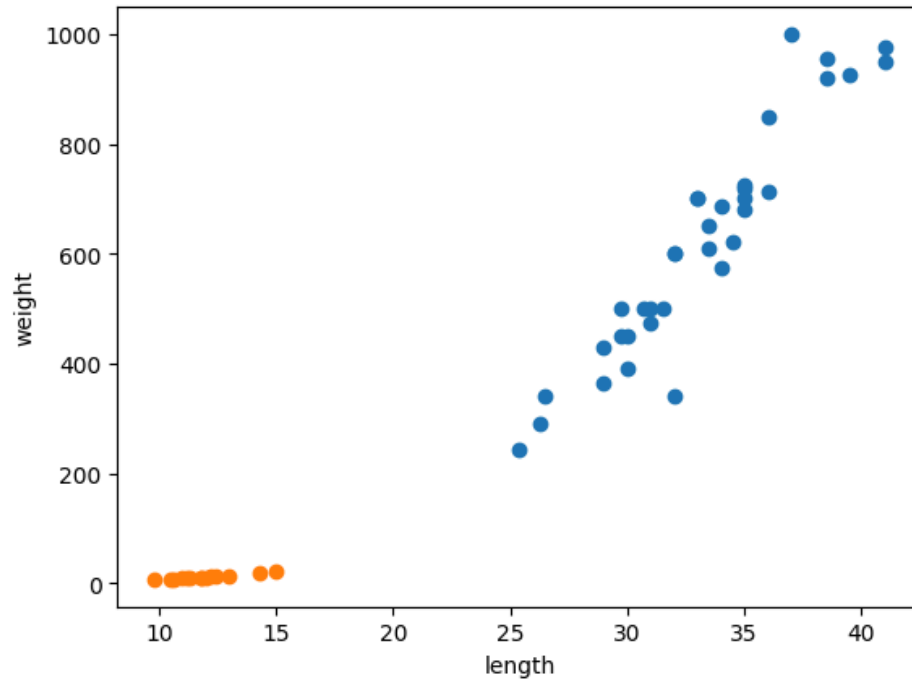
14
14

```

```

1 plt.scatter(bream_length, bream_weight)
2 plt.scatter(smelt_length, smelt_weight)
3 plt.xlabel('length')
4 plt.ylabel('weight')
5 plt.show()

```



▼ 첫 번째 머신러닝 프로그램

```

1 #생선을 분류하기 위해 k-최근접 이웃 알고리즘을 사용하려면 도미와 빙어 데이터를 하나의 데이터로 합침
2 #사이킷런 머신러닝 패키지를 사용하기 위해서는 샘플 하나의 데이터가 하나의 리스트에 담기도록 변경해야 함
3
4 length = bream_length + smelt_length
5 weight = bream_weight + smelt_weight
6
7 #zip() 함수와 리스트 내포 구문을 사용해 각 특성의 리스트를 세로 방향으로 늘어뜨린 2차원 리스트로 만들어 줌
8 fish_data = [[l, w] for l, w in zip(length, weight)]
9 print(fish_data)

```

◀ ▶

[illegible]

```
1 #scikit-learn은 knn 분류기를 KNeighborsClassifier 클래스로 제공
2 from sklearn.neighbors import KNeighborsClassifier
3
4 #클래스 객체를 만들고 fish_data와 fish_target을 전달해 도미를 찾기 위한 기준을 학습시키는 훈련을 함
5 #이를 위해 fit() 메소드를 사용해 fish_data와 fish_target을 순서대로 전달하고 주어진 데이터로 알고리즘을 훈련
6 kn = KNeighborsClassifier()
7 kn.fit(fish_data, fish_target)
8
9 #객체 또는 모델인 kn이 얼마나 잘 훈련되었는지 평가하기 위해 score() 메소드를 사용해 모델을 평가
10 #0에서 1사이의 값을 반환하며 1은 모든 데이터를 정확히 맞췄다는 것을 의미하며 이 값을 정확도라고 함
11 kn.score(fish_data, fish_target)
```

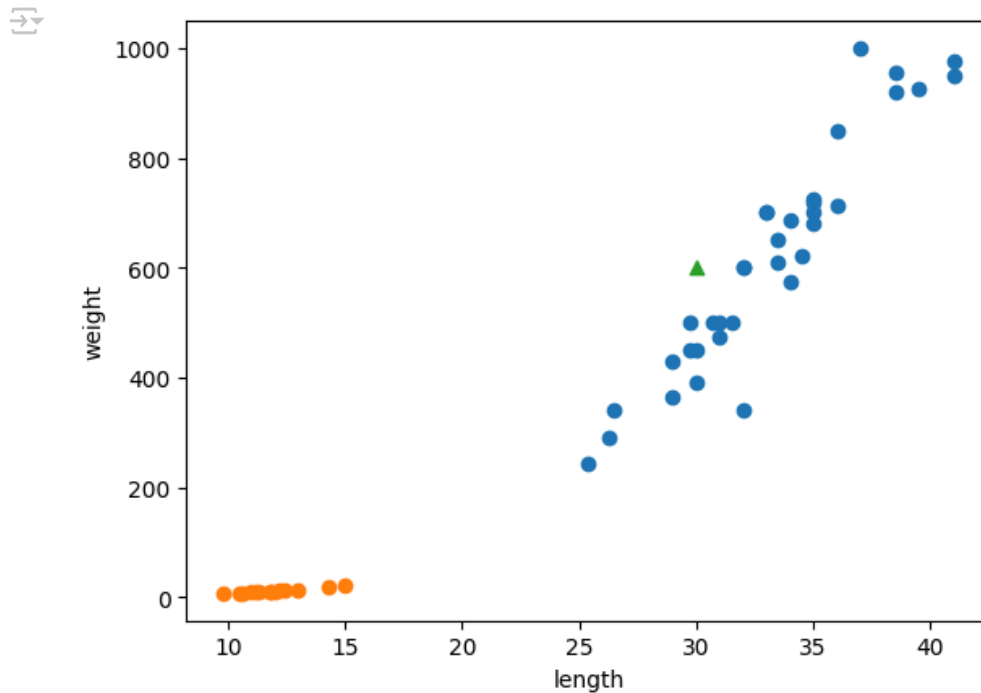
1.0

1 #새로운 데이터가 있을 때 직관적으로 이 삼각형은 주변에 다른 도미 데이터가 많기 때문에
2 #도미라고 판단할 것이며 k-최근접 이웃 알고리즘도 삼각형 주위에 도미 데이터가 많으므로 삼각형을 도미라고 판단할 것임

```

3
4 plt.scatter(bream_length, bream_weight)
5 plt.scatter(smelt_length, smelt_weight)
6
7 plt.scatter(30, 600, marker='^') #파이썬 마커 - https://wikidocs.net/92083
8
9 plt.xlabel('length')
10 plt.ylabel('weight')
11 plt.show()

```



```

1 #predict() 메소드를 사용해 리스트의 리스트를 전달해 새로운 데이터의 정답을 예측
2 #반환되는 값이 1이면 도미이며, 0이면 빙어이므로 새로 들어온 데이터인 삼각형은 도미임
3
4 kn.predict([[30, 600]])

```

```
array([1])
```

```

1 #속성 -X에 전달한 fish_data를 가지고 있음
2


```

```
3 print(kn_fit_X)
```

```
[[ 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]]
```

[illegible]

 0.7142857142857143

 0.7142857142857143

```
1 #n_neighbors의 기본값인 5부터 49까지 바꾸어 가며 점수가 1.0 아래로 내려가기 시작하는 이웃의 개수를 찾아보자!!!
2 #k-최근접 이웃 알고리즘의 훈련은 데이터를 저장하는 것이 전부이기 때문에
3 #객체를 매번 새로 만들거나 fit() 메소드를 통해 훈련할 필요가 없음
4
5 kn = KNeighborsClassifier()
6 kn.fit(fish_data, fish_target)
7
8 for n in range(5, 50):
```

```
9    kn.n_neighbors = n #최근접 이웃 개수 설정
10    score = kn.score(fish_data, fish_target) # 점수 계산
11
12    if score < 1: # 100% 정확도에 미치지 못하는 이웃 개수 출력
13        print(n, score)
14        break
```

↩ 18 0.9795918367346939

✓ <<<참조자료 사이트>>>
