

강원지역혁신플랫폼

기계학습

Machine Learning

K-최근점 이웃 분석 실습(2)



▼ 학습목표

- K-최근접 이웃 모델로 회귀와 분류의 문제에 적용할 수 있습니다.





01 | 실습

❖ (권장) 아래와 같은 경로에 실행 소스가 존재하면 환경 구축 완료

- ◆ 구글 드라이브 “PyWork > ML” 폴더로 이동함
 - 아래의 [ch12] 폴더를 클릭하면 됨

내 드라이브 > PyWork > ML

유형 ▾ 사람 ▾ 수정 날짜 ▾

이름	↑
ch09	
ch10	
ch11	
ch12	
ch13	
ch14	
HelloWorld	



01 | 실습

- ◆ “ML > ch12 >” 폴더를 클릭함
 - › 아래의 [ch12_02_K-최근접이웃분석실습.ipynb] 스크립트를 클릭함

... > ML > ch12 ▾

유형 ▾ 사람 ▾ 수정 날짜 ▾

이름 ↑
basketball_test.csv
basketball_train.csv
ch12_01_K-최근접이웃분석실습.ipynb
ch12_02_K-최근접이웃분석실습.ipynb



02 | K-최근접 이웃(K-NN) 분석 실습 (1)



K-최근접 이웃(K-NN) 분석 실습 (1)

- ❖ 다음은 K-NN 모델로 주택 가격 예측을 해 보자.
 - ◆ 임의로 주택 가격 예측을 위한 데이터 셋을 생성함
 - › 독립 변수 1개와 종속 변수 1개로 구성함
 - › 관측치는 100개임



02 | K-최근접 이웃(K-NN) 분석 실습 (1)

❖ 다음은 임의로 주택 가격 예측을 위한 데이터셋을 생성하는 코드이다.

- ◆ 독립 변수 1개와 종속 변수 1개로 구성함
- ◆ 관측치는 100개임

```
np.random.seed(42)
X = np.random.rand(100, 1) * 10 # 0에서 10 사이의 랜덤 숫자 100개
y = 2.5 * X.flatten() + np.random.randn(100) * 2 # y = 2.5x + 노이즈

print(X.shape) # (100, 1)
print(y.shape) # (100,)
```



02 | K-최근접 이웃(K-NN) 분석 실습 (1)

❖ 다음은 데이터 셋을 **훈련 데이터**(80%)와 **테스트 데이터**(20%)로 **분리**하는 코드이다.

◆ 아래와 같이 8:2 비율로 잘 분리된 것을 볼 수 있음

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(80, 1)
(20, 1)
(80,)
(20,)
```



02 | K-최근접 이웃(K-NN) 분석 실습 (1)

❖ 다음은 훈련 데이터와 테스트 데이터의 **데이터 단위를 표준화**하는 코드이다.

◆ 아래와 같이 **표준정규분포**로 표준화하여 각 특성의 **평균**을 0, **분산**을 1로 변경함

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
print(X_train.mean())
print(X_train.var())
print(X_test.mean())
print(X_test.var())
```

```
-2.2204460492503132e-17
1.0000000000000002
0.03574762444559236
1.1639184016520738
```



02 | K-최근접 이웃(K-NN) 분석 실습 (1)

- ❖ 다음은 **K값**을 3으로 설정하여 **K-NN 모델**을 **생성**하는 코드이다.
 - ◆ K-NN 알고리즘을 사용하여 **수치**(연속형 변수)를 **예측하는 모델**을 만듦
 - > 이 경우 **K-NN 회귀 모델**을 **사용**하게 됨

```
k = 3      # k 값을 설정  
model_knn = KNeighborsRegressor(n_neighbors=k)
```



02 | K-최근점 이웃(K-NN) 분석 실습 (1)

❖ 다음은 훈련 데이터 셋으로 K-NN 모델을 학습하는 코드이다.

◆ 아래와 같이 K-NN 회귀 모델에 훈련 데이터로 학습함

```
model_knn.fit(X_train, y_train)
```



02 | K-최근점 이웃(K-NN) 분석 실습 (1)

❖ 다음은 테스트 데이터 셋으로 학습된 K-NN 모델 예측을 수행하는 코드이다.

◆ 아래와 같이 예측 결과를 볼 수 있음

```
y_pred = model_knn.predict(X_test)  
y_pred
```

```
array([ 2.97760023, 23.84264292, 18.08990858, 15.8495983 , 5.99052799,  
       10.85759084, 6.01131037, 22.35680206, 1.83871816, 8.79602854,  
       10.85759084, 13.43025843, 20.97552345, 22.38662587, 2.93373605,  
       2.97722812, 18.08990858, 1.53754247, 20.97552345, 6.75481822])
```



02 | K-최근접 이웃(K-NN) 분석 실습 (1)

❖ 다음은 평균 제곱 오차(MSE), 결정 계수(R^2)로 학습된 K-NN 모델의 성능을 평가하는 코드이다.

◆ 아래와 같이 MSE는 3.38, R^2 는 0.94인 것을 볼 수 있음

```
mse = mean_squared_error(y_test, y_pred) # MSE
r2 = r2_score(y_test, y_pred)           # R^2
print(f"K-NN 회귀 모델의 MSE: {mse:.2f}")
print(f"K-NN 회귀 모델의 R^2: {r2:.2f}")
```

K-NN 회귀 모델의 MSE: 3.38
K-NN 회귀 모델의 R^2: 0.94

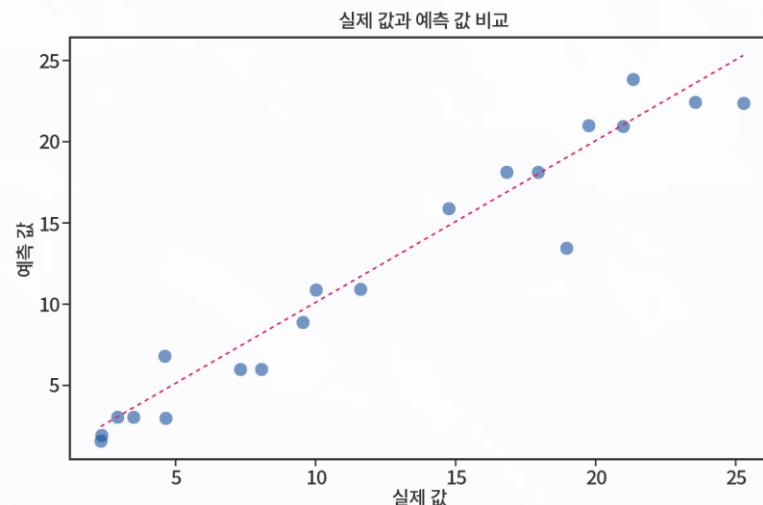


02 | K-최근점 이웃(K-NN) 분석 실습 (1)

❖ 다음은 K-NN 모델의 실제 값과 예측 값을 시각화하는 코드이다.

◆ 아래와 같이 모델이 비슷하게 예측은 하지만, 어떤 값은 차이가 큰 것을 볼 수 있음

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, edgecolor='k', alpha=0.7, s=100)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--', lw=2)
plt.xlabel('실제 값')
plt.ylabel('예측 값')
plt.title('실제 값과 예측 값 비교')
plt.show()
```





02 | K-최근접 이웃(K-NN) 분석 실습 (1)

- ❖ 다음은 선형 회귀 모델을 생성 및 학습하는 코드이다.
 - ◆ 아래와 같이 선형 회귀 모델을 생성하고, 훈련 데이터로 학습하는 것을 볼 수 있음

```
model_lr = LinearRegression().fit(X_train, y_train)
```



02 | K-최근접 이웃(K-NN) 분석 실습 (1)

❖ 다음은 테스트 데이터 셋으로 학습된 선형 회귀 모델로 예측하는 코드이다.

- ◆ 아래와 같이 잘 예측된 것을 볼 수 있음

```
y_pred = model_lr.predict(X_test)  
y_pred
```

```
array([ 1.82376677, 21.93822768, 18.97206085, 16.31707355, 6.5476021 ,  
       10.93632621, 7.35493677, 21.17059453, 0.78391571, 9.34868425,  
      10.73772752, 14.98676842, 20.01778708, 23.24628955, 3.17968384,  
     4.06105565, 18.94848246, 2.07750689, 20.42868648, 4.4120491 ])
```



02 | K-최근접 이웃(K-NN) 분석 실습 (1)

❖ 다음은 평균 제곱 오차(MSE), 결정 계수(R^2)로 학습된 선형 회귀 모델의 성능을 평가하는 코드이다.

◆ 아래와 같이 MSE는 2.61, R^2 는 0.95인 것을 볼 수 있음

```
lr_mse = mean_squared_error(y_test, y_pred)
lr_r2 = r2_score(y_test, y_pred)
print(f"선형 회귀 모델의 MSE: {lr_mse:.2f} ")
Print(f"선형 회귀 모델의 R^2: {lr_r2:.2f}")
```

선형 회귀 모델의 MSE: 2.61

선형 회귀 모델의 R^2: 0.95

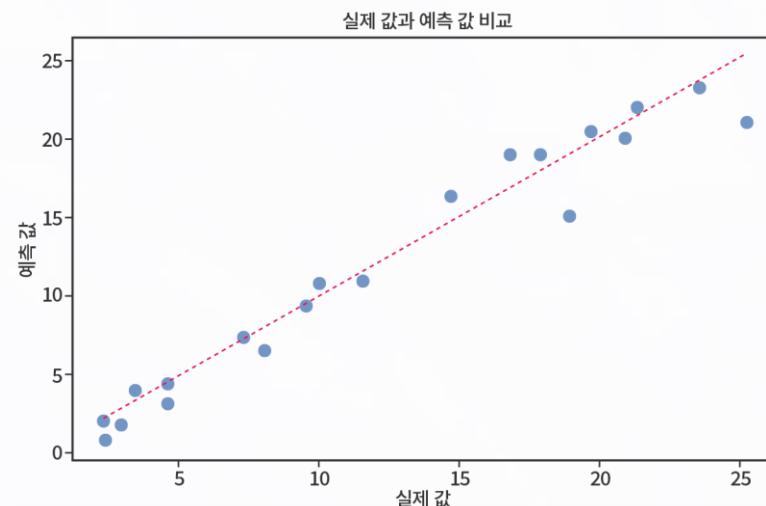


02 | K-최근접 이웃(K-NN) 분석 실습 (1)

❖ 다음은 선형 회귀 모델의 실제 값과 예측 값을 시각화하는 코드이다.

◆ 아래와 같이 모델이 비슷하게 예측은 하지만, 어떤 값은 차이가 큰 것을 볼 수 있음

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, edgecolor='k', alpha=0.7, s=100)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--', lw=2)
plt.xlabel('실제 값')
plt.ylabel('예측 값')
plt.title('실제 값과 예측 값 비교')
plt.show()
```



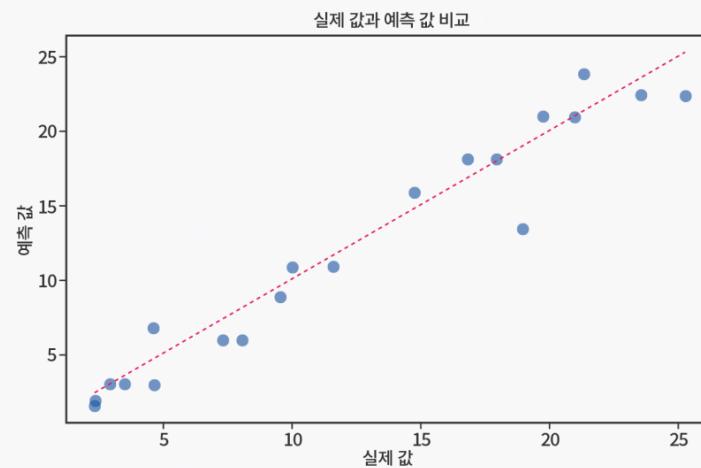


02 | K-최근점 이웃(K-NN) 분석 실습 (1)

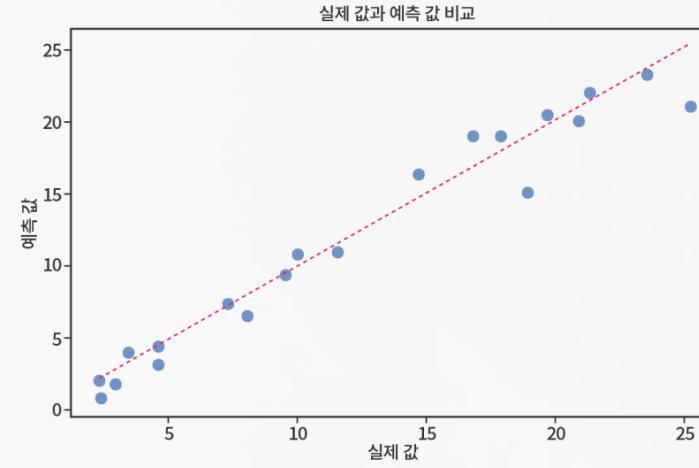
❖ 다음은 평균 제곱 오차(MSE), 결정 계수(R^2)로 학습된 K-NN 모델과 선형 회귀 모델의 성능을 비교한 결과이다.

◆ 아래와 같이 선형 회귀 모델의 MSE 값이 작아 조금 더 우수한 것을 볼 수 있음

K-NN 회귀 모델의 MSE: 3.38
K-NN 회귀 모델의 R^2 : 0.94



선형 회귀 모델의 MSE: 2.61
선형 회귀 모델의 R^2 : 0.95





02 | K-최근접 이웃(K-NN) 분석 실습 (2)

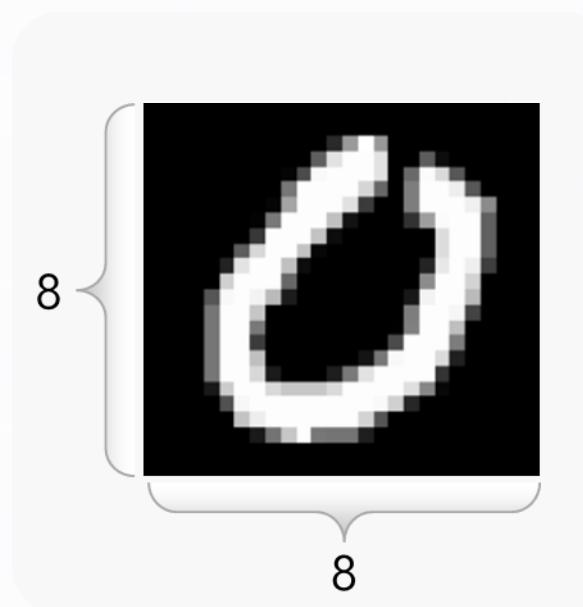


K-최근접 이웃(K-NN) 분석 실습 (2)

△ 다음은 K-NN 모델로 **8x8** 크기의 **손글씨 숫자 이미지**를 예측해 보자.

◆ **손글씨 숫자 이미지는 MNIST 데이터 셋을 사용함**

› 손글씨 숫자 이미지는 아래와 같이 **너비 8** 픽셀, **높이 8** 픽셀임





02 | K-최근접 이웃(K-NN) 분석 실습 (2)

❖ 다음은 MNIST 손글씨 숫자 이미지 데이터 셋을 읽어오는 코드이다.

- ◆ 아래와 같이 독립 변수의 형상은 (1797, 64)이고, 종속 변수의 형상은 (1797,) 인 것을 볼 수 있음
 - > 독립 변수의 각 이미지는 64개의 숫자 값으로 구성된 것을 볼 수 있음
 - > 종속 변수는 각 정답 레이블 값으로 구성된 것을 볼 수 있음

```
digits = load_digits()  
X = digits.data  
y = digits.target  
print(X.shape)  
print(y.shape)  
print(X[:2])  
print(y[:2])
```

```
(1797, 64)  
(1797,)  
[[ 0.  0.  5. 13.  9.  1.  0.  0.  0.  0. 13. 15. 10. 15.  5.  0.  0.  3.  
 15.  2.  0. 11.  8.  0.  0.  4. 12.  0.  0.  8.  8.  0.  0.  5.  8.  0.  
 0.  9.  8.  0.  0.  4. 11.  0.  1. 12.  7.  0.  0.  2. 14.  5. 10. 12.  
 0.  0.  0.  6. 13. 10.  0.  0.  0.]  
[ 0.  0.  0. 12. 13.  5.  0.  0.  0.  0. 11. 16.  9.  0.  0.  0.  0.  
 3. 15. 16.  6.  0.  0.  0.  7. 15. 16. 16.  2.  0.  0.  0.  0.  1. 16.  
 16.  3.  0.  0.  0.  0.  1. 16. 16.  6.  0.  0.  0.  0.  1. 16. 16.  6.  
 0.  0.  0.  0. 11. 16. 10.  0.  0. ]]  
[0 1]
```



02 | K-최근접 이웃(K-NN) 분석 실습 (2)

❖ 다음은 데이터 셋을 **훈련 데이터(80%)**와 **테스트 데이터(20%)**로 **분리**하는 코드이다.

◆ 아래와 같이 **8:2 비율**로 **잘 분리된 것**을 볼 수 있음

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1437, 64)
(360, 64)
(1437,)
(360,)
```



02 | K-최근접 이웃(K-NN) 분석 실습 (2)

❖ 다음은 훈련 데이터와 테스트 데이터의 **데이터 단위를 표준화**하는 코드이다.

◆ 아래와 같이 **표준정규분포**로 표준화하여 각 특성의 **평균**을 0, **분산**을 1로 변경함

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
print(X_train.mean())
print(X_train.var())
print(X_test.mean())
print(X_test.var())
```

```
-1.1102230246251566e-17
1.0000000000000002
0.03574762444559236
1.1639184016520738
```



02 | K-최근접 이웃(K-NN) 분석 실습 (2)

❖ 다음은 최적의 K를 찾기 위한 10-겹 교차검증 수행하는 코드이다.

◆ 다만 K값은 3부터 $\sqrt{\text{관측치의 개수}} \text{ 범위 이내}$ 이면서 2씩 증가시킴

```
max_k_range = int(math.sqrt(X_train.shape[0]))
print(max_k_range)
# max_k_range = train.shape[0] // 2

k_list = []
for i in range(3, max_k_range, 2):
    k_list.append(i)
print(k_list)

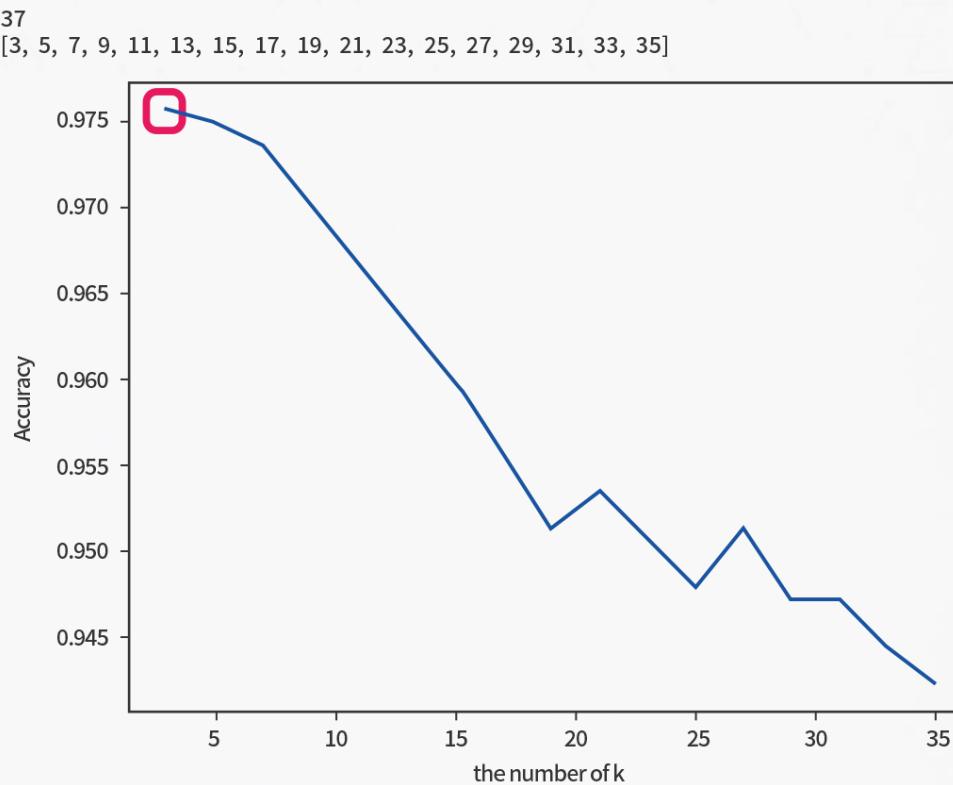
cross_validation_scores = []
# 10-fold cross validation
for k in k_list:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    cross_validation_scores.append(scores.mean())
cross_validation_scores

# visualize accuracy according to k
plt.plot(k_list, cross_validation_scores)
plt.xlabel('the number of k')
plt.ylabel('Accuracy')
plt.show()
```



02 | K-최근점 이웃(K-NN) 분석 실습 (2)

- ◆ 실행결과는 아래 그림과 같음
 - > 아래결과에서 K값은 3 ~ 35 범위 중에서 K = 3인 경우 정확도가 가장 높은 것을 알 수 있음





02 | K-최근접 이웃(K-NN) 분석 실습 (2)

- ❖ 다음은 최적의 K값을 확인하는 코드이다.
 - ◆ 실행결과는 최적의 K값은 3인 것을 알 수 있음

```
cvs = cross_validation_scores  
k = k_list[cvs.index(max(cross_validation_scores))]  
print("The best number of k : " + str(k)) # The best number of k : 3
```



02 | K-최근접 이웃(K-NN) 분석 실습 (2)

- ❖ 다음은 **K값을 3으로 설정하여 K-NN 모델을 생성하는 코드**이다.
 - ◆ K-NN 알고리즘을 사용하여 **분류(범주형)**를 **예측하는 모델**을 만듦
 - > 이 경우 **K-NN 분류 모델**을 사용하게 됨

```
k = 3      # k 값을 설정  
knn = KNeighborsClassifier(n_neighbors=k)
```



02 | K-최근접 이웃(K-NN) 분석 실습 (2)

- ❖ 다음은 훈련 데이터 셋으로 K-NN 모델을 학습하는 코드이다.
 - ◆ 아래와 같이 K-NN 분류 모델에 훈련 데이터로 학습함

```
model_knn.fit(X_train, y_train)
```



02 | K-최근접 이웃(K-NN) 분석 실습 (2)

❖ 다음은 테스트 데이터 셋으로 학습된 K-NN 모델 예측을 수행하는 코드이다.

◆ 아래와 같이 예측 결과를 볼 수 있음

```
y_pred = model_knn.predict(X_test)  
y_pred
```

```
array([6, 9, 3, 7, 2, 1, 5, 2, 5, 2, 1, 9, 4, 0, 4, 2, 3, 7, 8, 8, 4, 3,  
    9, 7, 5, 6, 3, 5, 6, 3, 4, 9, 1, 4, 4, 6, 9, 4, 7, 6, 6, 9, 1, 3,  
    6, 1, 3, 0, 6, 5, 5, 1, 3, 5, 6, 0, 9, 0, 0, 1, 0, 4, 5, 2, 4, 5,  
    7, 0, 7, 5, 9, 9, 5, 4, 7, 0, 4, 5, 5, 9, 9, 0, 2, 3, 8, 0, 6, 4,  
    4, 9, 1, 2, 8, 3, 5, 2, 9, 0, 4, 4, 4, 3, 5, 3, 1, 3, 5, 9, 4, 2,  
    7, 7, 4, 4, 1, 9, 2, 7, 8, 7, 2, 6, 9, 4, 0, 7, 2, 7, 5, 8, 7, 5,  
    7, 9, 0, 6, 6, 4, 2, 8, 0, 9, 4, 6, 8, 9, 6, 9, 0, 3, 5, 6, 6, 0,  
    6, 4, 2, 9, 3, 4, 7, 2, 9, 0, 4, 5, 3, 6, 5, 9, 9, 8, 4, 2, 1, 3,  
    7, 7, 2, 2, 3, 9, 8, 0, 3, 2, 1, 5, 6, 9, 9, 4, 1, 5, 4, 2, 3, 6,  
    4, 8, 5, 9, 5, 7, 1, 9, 4, 8, 1, 5, 4, 4, 9, 6, 1, 8, 6, 0, 4, 5,  
    2, 7, 4, 6, 4, 5, 6, 0, 3, 2, 3, 6, 7, 1, 5, 1, 4, 7, 6, 8, 8, 5,  
    5, 1, 6, 2, 8, 8, 9, 5, 7, 6, 2, 2, 3, 4, 8, 8, 3, 6, 0, 9, 7,  
    7, 0, 1, 0, 4, 5, 1, 5, 3, 6, 0, 4, 1, 0, 0, 3, 6, 5, 9, 7, 3, 5,  
    5, 9, 9, 8, 5, 3, 3, 2, 0, 5, 8, 3, 4, 0, 2, 4, 6, 4, 3, 4, 5, 0,  
    5, 2, 1, 3, 1, 4, 1, 1, 7, 0, 1, 5, 2, 1, 2, 8, 7, 0, 6, 4, 8, 8,  
    5, 1, 2, 4, 5, 8, 7, 9, 8, 6, 0, 6, 2, 0, 7, 9, 8, 9, 5, 2, 7, 7,  
    1, 8, 7, 4, 3, 8, 3, 5])
```



02 | K-최근점 이웃(K-NN) 분석 실습 (2)

❖ 다음은 K-NN 모델의 정확도를 확인하는 코드이다.

◆ 아래와 같이 정확도는 97%인 것을 볼 수 있음

```
accuracy = accuracy_score(y_test, y_pred)  
print(f"k-NN 모델의 정확도: {accuracy:.2f}")
```

```
k-NN 모델의 정확도: 0.97
```



02 | K-최근점 이웃(K-NN) 분석 실습 (2)

❖ 다음은 K-NN 모델의 실제 값과 예측 값을 시각화하는 코드이다.

◆ 아래와 같이 모델이 정확하게 예측한 것도 있지만, 어떤 값은 차이가 큰 것을 볼 수 있음

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, edgecolor='k', alpha=0.7, s=100)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--', lw=2)
plt.xlabel('실제 값')
plt.ylabel('예측 값')
plt.title('실제 값과 예측 값 비교')
plt.show()
```

