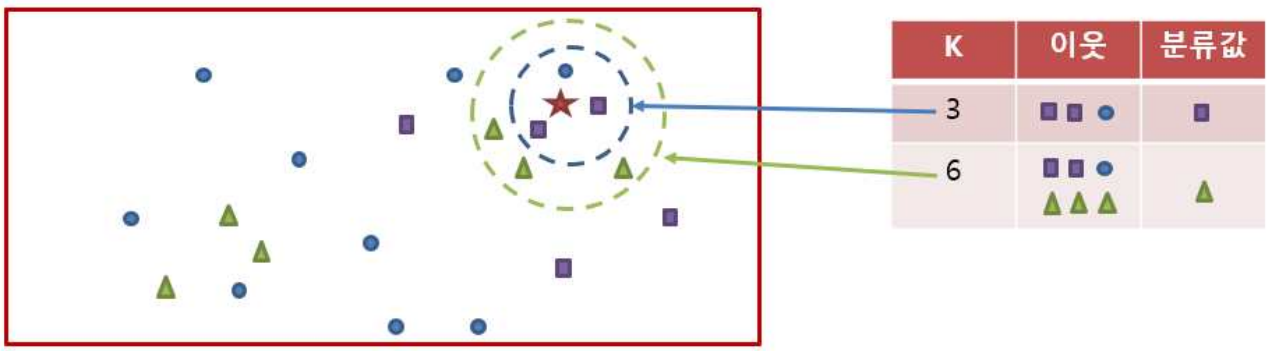
11. KNN

1. k 근방 분류기법의 이해

○ KNN

- k 근방 분류기법 (k-Nearest-Neighbors, KNN)은 주변 k개의 자료의 클래스 중 가장 많은 클래스로 특정 자료의 클래스를 지정하는 방법임.
- 이를 투표방식(Voting)이라고 함.
- k 근방 분류기법은 새로운 관측값이 주어지면 기존 데이터 중에서 가장 속성이 비슷한 k개의 이웃을 먼저 찾음.

- 간단한 예를 통해 k 근방 분류기법의 알고리즘을 알아봅시다.
- 두 개의 특성변수 (X_1과 X_2)를 가지고 있는 학습 데이터에 존재하는 3개 클래스들의 분포를 보여주는 그림임.



- 원 안의 ☆로 표시된 새로운 자료를 위치시킨 후 가장 가까운 $k=3$의 자료를 구함.
- ○ 클래스가 1개, □ 클래스가 2개이므로 원안의 ☆는 □ 클래스로 분류함.
- 그리고 원 안의 ☆로 표시된 새로운 자료를 위치시킨 후 가장 가까운 $k=6$의 자료를 구함.
- ○ 클래스가 1개, □ 클래스가 2개, △ 클래스가 3개이므로 원안의 ☆는 △ 클래스로 분류함.

- k 근방 분류기법에서는 k의 값에 따라 그 결과가 달라지게 됨.
- 즉, k값에 따라 예측의 정확도가 달라지므로 적절한 k값을 찾는 것이 매우 중요함.

- 그러므로 k 근방 분류기법에서는 k를 결정하고 거리를 측정하는 측도만 결정하면 됨.

- k 근방 분류기법에서는 i번째 관측치와 j번째 관측치의 거리로 민콥스키 거리를 이용하여 k개의 근접치를 구함.

$$d(x_i,x_j) = (\sum_{i=1}^{p}|x_i - x_j|^p)^{1/p}$$

- 어떤 추정방법도 없으며 새로운 자료의 클래스 분류를 위해 어떤 통계적 모형이나 자료변환도 하지 않는, 학습데이터 자체만 있으면 되므로 이를 게으른 학습 또는 사례중심학습이라고도 함.
- k 근방 분류기법은 매우 간단하지만 다른 분류기법과 비교해서 성능이 떨어지지 않음.
- 하지만 입력변수의 개수가 증가하면 소위 차원의 저주 문제가 발생하게 됨.

2. sklearn을 활용한 k 근방 분류기법

- k 근방 분류기법을 실시하기 위해 먼저 sklearn 모듈을 활용한 k 근방 분류기법 방법에 대해서 살펴보도록 하겠음.

- k 근방 분류기법은 앞서 설명한데로 분류를 위한 분석으로, k 근방 분류기법을 통해 출력변수의 값을 분류하여 예측함.
- 따라서 지난 시간 사용하였던 분석방법과 마찬가지로 주어진 데이터를 100% 활용하여 k 근방 분류기법을 실시한 후 새롭게 데이터를 모형에 적용하였을 때 예측 성능이 떨어지는 경우가 많음.
- 이러한 현상을 해소하기 위하여 k 근방 분류기법 역시 모형을 만드는 Train 데이터와 검정을 하는 Test 데이터를 일정 비율로 나누어 Test 데이터로 모형을 평가하는 데 이를 교차 검증을 이용함.
- 주어진 데이터를 Train 데이터와 Test 데이터로 분할은 지난 선형회귀분석과 로지스틱회귀분석과 마찬가지로 sklearn.model_selection 모듈의 train_test_split() 함수를 이용하여 실시할 수 있음.

```
from sklearn.model_selection import train_test_split
train 객체명, test 객체명 = train_test_split(데이터셋, test_size=비율,
random_state=번호, stratify = 출력변수)
```

- k 근방 분류기법은 로지스틱회귀분석과 마찬가지로 분류 방법이기 때문에 stratify 옵션을 이용하여 예측의 대상이 되는 출력변수에 대하여 층화추출을 통해 Train 데이터와 Test 데이터를 분류함.
- 그 외 train_test_split() 함수의 사용방법은 지난 로지스틱회귀분석을 참고 바람.

- 그럼 분석에 사용되는 Train 데이터를 활용하여 k 근방 분류기법을 실시하는 방법을 알아보도록 하겠음.
- 분석에 사용되는 Train 데이터를 활용하여 k 근방 분류기법 실시는 sklearn.neighbors 모듈의 KNeighborsClassifier() 함수를 이용하여 실시할 수 있음.

```
from sklearn.neighbors import KNeighborsClassifier
KNeighborsClassifier(n_neighbors= k값, p=민콥스키 모수).fit(입력변수, 출력변수)
```

- KNeighborsClassifier().fit() 함수에 분석에 사용되는 Train 데이터의 입력변수와 출력변수를 입력함.
- 그리고 n_neighbors 옵션을 통해 Voting을 위한 인접한 데이터의 수인 k값을 지정할 수 있음.
- 또한, 데이터간 거리를 측정하는데 사용되는 민콥스키의 모수를 p 옵션을 통해 지정할 수 있음. 만약 1의 값을 입력하면 맨하탄거리를 이용하여 거리를 측정하고, 2의 값을 입력하면 유클리디안 거리를 토해 거리를 측정하게 됨.
- 위 명령어를 사용하면 선형회귀분석과 로지스틱회귀분석과 마찬가지로 아무런 결과가 출력되지 않고, 분석이 실시되었다는 의미로 KNeighborsClassifier()이라는 메시지만 출력됨.
- 이는 R과 같은 다른 프로그램들과 마찬가지로 분석이 수행된 결과를 다른 객체에 할당한 후 이를 다양한 함수에 활용하여 결과를 확인할 수 있음.

- 따라서 KNeighborsClassifier().fit() 함수를 실행시킨 결과를 객체에 할당하여 다음과 같은 함수를 활용하여 k 근방 분류기법의 예측값을 확인할 수 있음.

예측값: knn 객체.**predict**(입력변수)

- k 근방 분류기법에 의해 예측된 출력변수의 값을 확인하기 위해서는 k 근방 분류기법의 결과를 할당한 객체를 활용하여 predict() 함수를 사용하여 확인할 수 있음. 이 때 predict() 함수에 출력변수의 예측값을 계산하기 위한 입력변수를 입력하면 출력변수의 예측값을 확인할 수 있음. 만약 Test 데이터의 입력변수를 입력하면 생성된 선형회귀모형에 Test 데이터를 적용하였을 때의 출력변수의 예측값을 확인할 수 있음.

- 그리고 k 근방 분류기법 모형 평가에 사용되는 분류표, 정분류율, 평가지표를 확인하기 위해 sklearn.metrics 모듈의 confusion_matrix() 함수, accuracy_score() 함수, classification_report() 함수를 이용하여 실시할 수 있음.

from **sklearn.metrics** import **confusion_matrix, accuracy_score, classification_report** 분류표: **confusion_matrix**(출력변수 예측값, 출력변수) 정분류율: **accuracy_score**(출력변수, 출력변수 예측값) 분류 예측력 평가 지표: **classification_report**(출력변수, 출력변수 예측값)

- k 근방 분류기법을 평가하기 위해 실제 출력변수의 값과 예측값을 이원분할표의 형태로 나타내기 위해 confusion_matrix() 함수를 사용할 수 있음. 이 때, 이원분할표의 행 위치와 열 위치를 차례로 입력하여 분류표의 행과 열에 위치할 값을 지정할 수 있음.
- confusion_matrix() 함수에 의해 나타난 분류표 상의 정분류율을 계산하기 위해 accuracy_score() 함수를 사용할 수 있음.이 때, 출력변수와 출력변수 예측값을 차례로 입력함.
- k 근방 분류기법을 평가하기 위한 지표인 presicion, recall, f1_score를 계산하기 위해 classification_report() 함수를 사용할 수 있음. 이 때, 출력변수와 출력변수 예측값을 차례로 입력함.

- 또한, k 근방 분류기법은 거리를 측정하여 데이터간 인접성을 판단하므로, 데이터의 단위가 매우 중요함. 만약 단위의 크기가 매우 상이하다면 좋은 분석결과가 나타나지 않을 수 있음.
- 입력변수의 표준화를 위해 sklearn.preprocessing 모듈의 StandardScaler().fit() 함수를 이용할 수 있음.

```
from sklearn.preprocessing import StandardScaler
객체명 = StandardScaler().fit(Train 데이터 입력변수)
Train 데이터 표준화: 객체명.transform(Train 데이터 입력변수)
Test 데이터 표준화: 객체명.transform(Test 데이터 입력변수)
```

- StandardScaler().fit() 함수에 분석에 사용되는 Train 데이터의 입력변수를 입력함.
- 이를 통해 각 입력변수별 표준화를 위한 평균과 표준편차를 구하게 됨. 이를 transform() 함수를 이용하여 자료를 변환함.
- Train 데이터를 표준화 하려 한다면, transform() 함수에 Train 데이터의 입력변수를 입력하고, Test 데이터를 표준화 하려 한다면 Test 데이터의 입력변수를 입력함.
- 만약 표준화를 통해 k 근방 분류기법을 적용하였을 때의 결과가 표준화를 하지 않았을 때의 결과가 크게 차이가 나지 않는다면, 모형 해석의 용이성 때문에 표준화 하지 않은 결과를 사용함.

- seaborn 모듈의 iris 데이터에서, species를 예측하기 위해 sepal_length, sepal_width, petal_length, petal_width를 사용한 k 근방 분류기법을 실시하고자 함.
- info() 함수를 활용하여 자료를 확인한 결과 분석에 사용되는 모든 변수에서 결측자료가 없는 것을 확인할 수 있음.

```
In [1]: import seaborn as sns

In [2]: df = sns.load_dataset('iris')

In [3]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

- 다음으로 모형을 생성하기 위한 Train 데이터와 검증을 위한 Test 데이터를 구분하기 위해 train_test_split() 함수를 사용하였음.

```
In [4]: from sklearn.model_selection import train_test_split

In [5]: train, test = train_test_split(df, test_size=0.3, random_state=1, stratify=df['species'])

In [6]: y_train=train['species']
   ...: X_train=train[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
   ...: y_test=test['species']
   ...: X_test=test[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
```

- 이 때 Train 데이터는 train 객체에 할당하고, Test 데이터는 test 객체에 할당하였음. 그런 후 X_train과 X_test 객체에 train 객체와 test 객체의 입력변수를 선택하여 각각 할당하였고, y_train과 y_test 객체에 train 객체와 test 객체의 출력변수를 선택하여 각각 할당하였음.

- Train 데이터의 입력변수 자료인 X_train과 Train 데이터의 출력변수 자료인 y_train을 각각 KNeighborsClassifier().fit() 함수에 대입하여 k 근방 분류기법을 실시하였음.

```
In [7]: from sklearn.neighbors import KNeighborsClassifier

In [8]: knn=KNeighborsClassifier(n_neighbors=5, p=2).fit(X_train, y_train)
```

- 이 때 k의 값을 5로 지정하였고, 거리의 측정방법을 p=2로 지정하여 유클리디안거리 방법으로 측정하였음.
- KNeighborsClassifier().fit() 함수를 실행시킨 결과는 출력되지 않으므로 이를 knn이라는 객체에 할당하였고, 이를 활용하여 출력변수의 예측값을 계산하였음.

```
In [9]: y_train_pred = knn.predict(X_train)
```

- predict() 함수에 Train 데이터의 입력변수 데이터인 X_train을 대입하여 Train 데이터의 출력변수 예측값을 y_train_pred 객체에 할당하여 k 근방 분류기법을 평가하는 지표인 정분류율, Precision, Recall, F1-score 등을 계산하였음.

```
In [9]: y_train_pred = knn.predict(X_train)

In [10]: from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

In [11]: confusion_matrix(y_train_pred, y_train)
Out[11]:
array([[35,  0,  0],
       [ 0, 33,  0],
       [ 0,  2, 35]], dtype=int64)

In [12]: accuracy_score(y_train_pred, y_train)
Out[12]: 0.9809523809523809

In [13]: knn_report = classification_report(y_train, y_train_pred)

In [14]: print(knn_report)
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        35
  versicolor       1.00      0.94      0.97        35
   virginica       0.95      1.00      0.97        35

    accuracy                           0.98       105
   macro avg       0.98      0.98      0.98       105
weighted avg       0.98      0.98      0.98       105
```

- 분류표, 정분류율, precision, recall, f1 score 등을 계산하기 위하여 sklearn.metrics 모듈의 confusion_matrix, accuracy_score, classification_report() 함수를 이용하였음. 그 결과 모형 생성에 사용된 Train 데이터에서는 species가 setosa, versicolor, virginica를 각각 35개, 33개, 35개 데이터에 대해서 제대로 분류하여 정분류율는 0.98095로 나타났음. 그리고 species가 setosa인 경우 precision, recall, f1-score는 각각 1.00, 1.00, 1.00으로 나타났고, species가 versicolor인 경우 precision, recall, f1-score는 각각 1.00, 0.94, 0.97로 나타났고, species가 virginica인 경우 precision, recall, f1-score는 각각 0.95, 1.00, 0.97로 나타났음.

- 다음으로 predict() 함수에 Test 데이터의 독립변수 데이터인 X_test를 대입하여 Test 데이터의 종속변수 예측값을 y_test_pred 객체에 할당하여 k 근방 분류기법을 평가하는 지표인 정분류율, precision, recall, f1 score 등을 계산하였음.

```
In [15]: y_test_pred = knn.predict(X_test)

In [16]: from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

In [17]: confusion_matrix(y_test_pred, y_test)
Out[17]:
array([[15,  0,  0],
       [ 0, 15,  1],
       [ 0,  0, 14]], dtype=int64)

In [18]: accuracy_score(y_test_pred, y_test)
Out[18]: 0.9777777777777777

In [19]: knn_report1 = classification_report(y_test, y_test_pred)

In [20]: print(knn_report1)
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.94      1.00      0.97        15
   virginica       1.00      0.93      0.97        15

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

- 그 결과 모형 생성에 사용된 Test 데이터에서는 species가 setosa, versicolor, virginica를 각각 15개, 15개, 14개 데이터에 대해서 제대로 분류하여 정분류율는 0.97777로 나타났음. 그리고 species가 setosa인 경우 precision, recall, f1-score는 각각 1.00, 1.00, 1.00으로 나타났고, species가 versicolor인 경우 precision, recall, f1-score는 각각 0.94, 1.00, 0.97로 나타났고, species가 virginica인 경우 precision, recall, f1-score는 각각 1.00, 0.93, 0.97로 나타났음.
- 이는 Train 데이터를 활용하여 나타난 결과와 큰 차이가 존재하지 않는다는 것을 확인할 수 있음.

- 다음으로 입력변수의 표준화를 통하여 거리를 측정한 후 k 근방 분류기법을 실시하기 위해 먼저 입력변수들을 표준화 하기 위해 StandardScaler() 함수를 사용하였음.

```
In [21]: from sklearn.preprocessing import StandardScaler

In [22]: sc=StandardScaler().fit(X_train)

In [23]: X_train_std=sc.transform(X_train)

In [24]: X_test_std=sc.transform(X_test)
```

- Train 데이터의 입력변수 X_train 객체를 StandardScaler().fit()에 대입하여 표준화를 위한 평균과 표준편차를 계산한 결과를 sc 객체에 할당하였음.
- 이 sc 객체에 transform() 함수를 적용하여 각각 X_train 객체와 X_test 객체를 표준화한 X_train_std와 X_test_std 객체를 생성하였음.

- 표준화한 X_train_std와 Train 데이터의 출력변수 자료인 y_train을 각각 KNeighborsClassifier().fit() 함수에 대입하여 k 근방 분류기법을 실시하였음.

```
In [25]: from sklearn.neighbors import KNeighborsClassifier

In [26]: knn1=KNeighborsClassifier(n_neighbors=5, p=2).fit(X_train_std, y_train)
```

- 표준화 하기 전 k 근방 분류기법과 마찬가지로, k의 값을 5로 지정하였고, 거리의 측정방법을 p=2로 지정하여 유클리디안거리 방법으로 측정하였음.
- KNeighborsClassifier().fit() 함수를 실행시킨 결과는 출력되지 않으므로 이를 knn이라는 객체에 할당하였고, 이를 활용하여 출력변수의 예측값을 계산하였음.

```
In [27]: y_train_std_pred = knn1.predict(X_train_std)
```

- predict() 함수에 Train 데이터의 입력변수 데이터인 X_train_std를 대입하여 Train 데이터의 출력변수 예측값을 y_train_std_pred 객체에 할당하여 k 근방 분류기법을 평가하는 지표인 정분류율, Precision, Recall, F1-score 등을 계산하였음.

```
In [28]: from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

In [29]: confusion_matrix(y_train_std_pred, y_train)
Out[29]:
array([[35,  0,  0],
       [ 0, 33,  2],
       [ 0,  2, 33]], dtype=int64)

In [30]: accuracy_score(y_train_std_pred, y_train)
Out[30]: 0.9619047619047619

In [31]: knn1_report = classification_report(y_train, y_train_std_pred)

In [32]: print(knn1_report)
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        35
  versicolor       0.94      0.94      0.94        35
   virginica       0.94      0.94      0.94        35

    accuracy                           0.96       105
   macro avg       0.96      0.96      0.96       105
weighted avg       0.96      0.96      0.96       105
```

- 분류표, 정분류율, precision, recall, f1 score 등을 계산하기 위하여 sklearn.metrics 모듈의 confusion_matrix, accuracy_score, classification_report() 함수를 이용하였음. 그 결과 모형 생성에 사용된 Train 데이터에서는 species가 setosa, versicolor, virginica를 각각 35개, 33개, 33개 데이터에 대해서 제대로 분류하여 정분류율는 0.9619로

나타났음. 그리고 species가 setosa인 경우 precision, recall, f1-score는 각각 1.00, 1.00, 1.00으로 나타났고, species가 versicolor인 경우 precision, recall, f1-score는 각각 0.94, 0.94, 0.94로 나타났고, species가 virginica인 경우 precision, recall, f1-score는 각각 0.94, 0.94, 0.94로 나타났음.

- 다음으로 predict() 함수에 Test 데이터의 독립변수 데이터인 X_test_std를 대입하여 Test 데이터의 종속변수 예측값을 y_test_std_pred 객체에 할당하여 k 근방 분류기법을 평가하는 지표인 정분류율, precision, recall, f1 score 등을 계산하였음.

```
In [33]: y_test_std_pred = knn1.predict(X_test_std)

In [34]: from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

In [35]: confusion_matrix(y_test_std_pred, y_test)
Out[35]:
array([[15,  0,  0],
       [ 0, 13,  1],
       [ 0,  2, 14]], dtype=int64)

In [36]: accuracy_score(y_test_std_pred, y_test)
Out[36]: 0.9333333333333333

In [37]: knn1_report1 = classification_report(y_test, y_test_std_pred)

In [38]: print(knn1_report1)
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.93      0.87      0.90        15
   virginica       0.88      0.93      0.90        15

    accuracy                           0.93        45
   macro avg       0.93      0.93      0.93        45
weighted avg       0.93      0.93      0.93        45
```

- 그 결과 모형 생성에 사용된 Test 데이터에서는 species가 setosa, versicolor, virginica를 각각 15개, 13개, 14개 데이터에 대해서 제대로 분류하여 정분류율는 0.93333으로 나타났음. 그리고 species가 setosa인 경우 precision, recall, f1-score는 각각 1.00, 1.00, 1.00으로 나타났고, species가 versicolor인 경우 precision, recall, f1-score는 각각 0.93, 0.87, 0.90으로 나타났고, species가 virginica인 경우 precision, recall, f1-score는 각각 0.88, 0.93, 0.90으로 나타났음.
- 표준화로 인해 정분류율이 떨어지고 있고, 그 외 지표인 precision, recall, f1-score 모두 같거나 떨어지는 경향을 보이고 있음. 따라서 iris 자료의 경우 표준화를 실시하여 k 근방 분류기법을 사용하기 보다 row 데이터를 그대로 사용하여 분석하는 것이 더 효율적이라고 판단

할 수 있음.

3. k 근방 분류기법의 실습

※ seaborn 모듈의 penguins 데이터에서, species를 예측하기 위해 bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g을 사용한 k 근방 분류기법을 실시하세요.

- info() 함수를 활용하여 자료를 확인한 결과 bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g에서 결측자료가 존재하는 것을 확인할 수 있음. 따라서 dropna() 함수를 이용하여 결측값을 제거할 필요가 있음.

```
In [1]: import seaborn as sns

In [2]: df = sns.load_dataset('penguins')

In [3]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB

In [4]: df = df.dropna(subset=['species', 'bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g'],
   ...:                how='any', axis=0)
```

- 다음으로 모형을 생성하기 위한 Train 데이터와 검증을 위한 Test 데이터를 구분하기 위해 train_test_split() 함수를 사용하였음.

```
In [5]: from sklearn.model_selection import train_test_split

In [6]: train, test = train_test_split(df, test_size=0.3, random_state=1, stratify=df['species'])

In [7]: y_train=train['species']
   ...: X_train=train[['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']]
   ...: y_test=test['species']
   ...: X_test=test[['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']]
```

- 이 때 Train 데이터는 train 객체에 할당하고, Test 데이터는 test 객체에 할당하였음. 그런 후 X_train과 X_test 객체에 train 객체와 test 객체의 입력변수를 선택하여 각각 할당하였고, y_train과 y_test 객체에 train 객체와 test 객체의 출력변수를 선택하여 각각 할당하였음.

- Train 데이터의 입력변수 자료인 X_train과 Train 데이터의 출력변수 자료인 y_train을 각각 KNeighborsClassifier().fit() 함수에 대입하여 k 근방

분류기법을 실시하였음.

```
In [8]: from sklearn.neighbors import KNeighborsClassifier

In [9]: knn=KNeighborsClassifier(n_neighbors=5, p=2).fit(X_train, y_train)
```

- 이 때 k의 값을 5로 지정하였고, 거리의 측정방법을 p=2로 지정하여 유클리디안거리 방법으로 측정하였음.
- KNeighborsClassifier().fit() 함수를 실행시킨 결과는 출력되지 않으므로 이를 knn이라는 객체에 할당하였고, 이를 활용하여 출력변수의 예측값을 계산하였음.

```
In [10]: y_train_pred = knn.predict(X_train)
```

- predict() 함수에 Train 데이터의 입력변수 데이터인 X_train을 대입하여 Train 데이터의 출력변수 예측값을 y_train_pred 객체에 할당하여 k 근방 분류기법을 평가하는 지표인 정분류율, Precision, Recall, F1-score 등을 계산하였음.

```
In [11]: from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

In [12]: confusion_matrix(y_train_pred, y_train)
Out[12]:
array([[95, 20,  5],
       [ 4, 25,  1],
       [ 7,  2, 80]], dtype=int64)

In [13]: accuracy_score(y_train_pred, y_train)
Out[13]: 0.8368200836820083

In [14]: knn_report = classification_report(y_train, y_train_pred)

In [15]: print(knn_report)
              precision    recall  f1-score   support

      Adelie       0.79      0.90      0.84       106
   Chinstrap       0.83      0.53      0.65        47
      Gentoo       0.90      0.93      0.91        86

    accuracy                           0.84       239
   macro avg       0.84      0.79      0.80       239
weighted avg       0.84      0.84      0.83       239
```

- 분류표, 정분류율, precision, recall, f1 score 등을 계산하기 위하여 sklearn.metrics 모듈의 confusion_matrix, accuracy_score, classification_report() 함수를 이용하였음. 그 결과 모형 생성에 사용된 Train 데이터에서는 species가 Adelie, Chinstrap, Gentoo를 각각 95개, 25개, 80개 데이터에 대해서 제대로 분류하여 정분류율는 0.83682로 나타났음. 그리고 species가 Adelie인 경우 precision, recall, f1-score는 각각 0.79, 0.90, 0.84로 나타났고, species가 Chinstrap인 경우 precision, recall, f1-score는 각각 0.83, 0.53, 0.65로 나타났고, species

가 Gentoo인 경우 precision, recall, f1-score는 각각 0.90,0.93, 0.91로 나타났음.

- 다음으로 predict() 함수에 Test 데이터의 독립변수 데이터인 X_test를 대입하여 Test 데이터의 종속변수 예측값을 y_test_pred 객체에 할당하여 k 근방 분류기법을 평가하는 지표인 정분류율, precision, recall, f1 score 등을 계산하였음.

```
In [16]: y_test_pred = knn.predict(X_test)

In [17]: from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

In [18]: confusion_matrix(y_test_pred, y_test)
Out[18]:
array([[42, 16,  4],
       [ 1,  4,  0],
       [ 2,  1, 33]], dtype=int64)

In [19]: accuracy_score(y_test_pred, y_test)
Out[19]: 0.7669902912621359

In [20]: knn_report1 = classification_report(y_test, y_test_pred)

In [21]: print(knn_report1)
              precision    recall  f1-score   support

      Adelie       0.68      0.93      0.79        45
   Chinstrap       0.80      0.19      0.31        21
      Gentoo       0.92      0.89      0.90        37

    accuracy                           0.77       103
   macro avg       0.80      0.67      0.67       103
weighted avg       0.79      0.77      0.73       103
```

- 그 결과 모형 생성에 사용된 Test 데이터에서는 species가 Adelie, Chinstrap, Gentoo를 각각 42개, 4개, 33개 데이터에 대해서 제대로 분류하여 정분류율는 0.76699로 나타났음. 그리고 species가 Adelie인 경우 precision, recall, f1-score는 각각 0.68, 0.93, 0.79로 나타났고, species가 Chinstrap인 경우 precision, recall, f1-score는 각각 0.80, 0.19, 0.31로 나타났고, species가 Gentoo인 경우 precision, recall, f1-score는 각각 0.92, 0.89, 0.90으로 나타났음.
- 이는 species가 Chinstrap인 경우 Train 데이터를 활용하여 나타난 결과와 차이가 존재하는 것을 확인할 수 있음.

- 따라서 입력변수의 표준화를 통하여 거리를 측정한 후 k 근방 분류기법을 실시하기 위해 먼저 입력변수들을 표준화 하기 위해 StandardScaler() 함수를 사용하였음.

```
In [22]: from sklearn.preprocessing import StandardScaler

In [23]: sc=StandardScaler().fit(X_train)

In [24]: X_train_std=sc.transform(X_train)

In [25]: X_test_std=sc.transform(X_test)
```

- Train 데이터의 입력변수 X_train 객체를 StandardScaler().fit()에 대입하여 표준화를 위한 평균과 표준편차를 계산한 결과를 sc 객체에 할당하였음.
- 이 sc 객체에 transform() 함수를 적용하여 각각 X_train 객체와 X_test 객체를 표준화한 X_train_std와 X_test_std 객체를 생성하였음.

- 표준화한 X_train_std와 Train 데이터의 출력변수 자료인 y_train을 각각 KNeighborsClassifier().fit() 함수에 대입하여 k 근방 분류기법을 실시하였음.

```
In [26]: from sklearn.neighbors import KNeighborsClassifier

In [27]: knn1=KNeighborsClassifier(n_neighbors=5, p=2).fit(X_train_std, y_train)
```

- 표준화 하기 전 k 근방 분류기법과 마찬가지로, k의 값을 5로 지정하였고, 거리의 측정방법을 p=2로 지정하여 유클리디안거리 방법으로 측정하였음.
- KNeighborsClassifier().fit() 함수를 실행시킨 결과는 출력되지 않으므로 이를 knn이라는 객체에 할당하였고, 이를 활용하여 출력변수의 예측값을 계산하였음.

```
In [28]: y_train_std_pred = knn1.predict(X_train_std)
```

- predict() 함수에 Train 데이터의 입력변수 데이터인 X_train_std를 대입하여 Train 데이터의 출력변수 예측값을 y_train_std_pred 객체에 할당하여 k 근방 분류기법을 평가하는 지표인 정분류율, Precision, Recall, F1-score 등을 계산하였음.

```
In [29]: from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

In [30]: confusion_matrix(y_train_std_pred, y_train)
Out[30]:
array([[106,   3,   0],
       [  0,  44,   0],
       [  0,   0,  86]], dtype=int64)

In [31]: accuracy_score(y_train_std_pred, y_train)
Out[31]: 0.9874476987447699

In [32]: knn1_report = classification_report(y_train, y_train_std_pred)

In [33]: print(knn1_report)
              precision    recall  f1-score   support

      Adelie       0.97      1.00      0.99       106
   Chinstrap       1.00      0.94      0.97        47
      Gentoo       1.00      1.00      1.00        86

    accuracy                           0.99       239
   macro avg       0.99      0.98      0.98       239
weighted avg       0.99      0.99      0.99       239
```

- 분류표, 정분류율, precision, recall, f1 score 등을 계산하기 위하여 sklearn.metrics 모듈의 confusion_matrix, accuracy_score, classification_report() 함수를 이용하였음. 그 결과 모형 생성에 사용된 Train 데이터에서는 species가 Adelie, Chinstrap, Gentoo를 각각 95개, 25개, 80개 데이터에 대해서 제대로 분류하여 정분류율는 0.987447로 나타났음. 그리고 species가 Adelie인 경우 precision, recall, f1-score는 각각 0.967, 1.00, 0.99로 나타났고, species가 Chinstrap인 경우 precision, recall, f1-score는 각각 1.00, 0.94, 0.97로 나타났고, species가 Gentoo인 경우 precision, recall, f1-score는 각각 1.00, 1.00, 1.00으로 나타났음.

- 다음으로 predict() 함수에 Test 데이터의 독립변수 데이터인 X_test_std를 대입하여 Test 데이터의 종속변수 예측값을 y_test_std_pred 객체에 할당하여 k 근방 분류기법을 평가하는 지표인 정분류율, precision, recall, f1 score 등을 계산하였음.

```
In [34]: y_test_std_pred = knn1.predict(X_test_std)

In [35]: from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

In [36]: confusion_matrix(y_test_std_pred, y_test)
Out[36]:
array([[44,  3,  0],
       [ 1, 18,  0],
       [ 0,  0, 37]], dtype=int64)

In [37]: accuracy_score(y_test_std_pred, y_test)
Out[37]: 0.9611650485436893

In [38]: knn1_report1 = classification_report(y_test, y_test_std_pred)

In [39]: print(knn1_report1)
              precision    recall  f1-score   support

      Adelie       0.94      0.98      0.96        45
   Chinstrap       0.95      0.86      0.90        21
      Gentoo       1.00      1.00      1.00        37

    accuracy                           0.96       103
   macro avg       0.96      0.94      0.95       103
weighted avg       0.96      0.96      0.96       103
```

- 그 결과 모형 생성에 사용된 Test 데이터에서는 species가 Adelie, Chinstrap, Gentoo를 각각 44개, 18개, 37개 데이터에 대해서 제대로 분류하여 정분류율는 0.961165로 나타났음. 그리고 species가 Adelie인 경우 precision, recall, f1-score는 각각 0.94, 0.98, 0.96으로 나타났고, species가 Chinstrap인 경우 precision, recall, f1-score는 각각 0.95, 0.86, 0.90으로 나타났고, species가 Gentoo인 경우 precision, recall, f1-score는 각각 1.00, 1.00, 1.00으로 나타났음.
- 표준화로 인해 정분류율이 상승하고 있고, 그 외 지표인 precision, recall, f1-score 모두 상승하는 경향을 보이고 있음. 따라서 penguins 자료의 경우 표준화를 실시하여 k 근방 분류기법을 사용하는 것이 더 효율적이라고 판단할 수 있음.