iris 데이터를 이용한 머신러닝

필요한 모듈 import

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris

iris = load_iris()
type(iris) # sklearn.utils.Bunch
print(iris)
print(iris.keys()) # col_name을 얻어내는 함수
# dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'f
```

데이터 확인

```
print(iris.target_names) # 컬럼명으로 해당 컬럼의 종류 확인 가능
# 품종은 크게 3가지로 나뉜것 확인 가능 : ['setosa' 'versicolor' 'virginica']

print(iris.feature_names) # 각 품종에 대한 특성명들 출력(컬럼명으로 해당 값들 출력
# ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal wid
```

feature들을 통해 (iris.feature_names)
이 데이터 셋에 없는 새로운 데이터가 들어왔을 때,
그 품종을 예측하는 머신러닝 모델을 만드는 것이 목적이다.

```
iris.data[:10]

#array([[5.1, 3.5, 1.4, 0.2],
#       [4.9, 3. , 1.4, 0.2],
#       [4.7, 3.2, 1.3, 0.2],
#       [4.6, 3.1, 1.5, 0.2],
#       [5. , 3.6, 1.4, 0.2],
#       [5.4, 3.9, 1.7, 0.4],
#       [4.6, 3.4, 1.4, 0.3],
#       [5. , 3.4, 1.5, 0.2],
```

```
#       [4.4, 2.9, 1.4, 0.2],
#       [4.9, 3.1, 1.5, 0.1]])
```

> 데이터의 1행을 보면 5.1 , 3.5, 1.4, 0.2라고 되어있는데
> 이는 각각 sepal length, sepal width, petal length, petal width 를 나타낸다.

생산지표시를 하려면 그 상품에 대한 품종을 알고, 품종을 통해 할 수 있는 것 :
마트에 가서 참외를 찍게되면 선이라던지, 꼭지의 특성을 이미지로 분석해서
기존 품종과 비교해서 어느지역의 참외인지 알 수 있다.
우리가 예측하고자 하는 붓꽃의 품종은 다음과 같다 0은 setosa, 1 은 versicolor, 2 는 virginica 이다.

```
iris.target # 라벨링 작업된 것 확인(어떤 품종은 어떤 값, 어떤 품종은 어떤 값)
#array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
#       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
#       0, 0, 0, 0, 0, 0, 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, 1, 1, 1,
#       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
#       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
#       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

지도학습

머신러닝 모델을 만들 때는 데이터를 인풋으로 넣고 학습시키는 것도 중요하지만,
그렇게 학습시킨 모델의 성능을 평가하는 것도 중요하다.
모델의 ★성능을 평가하지 않으면,
모델이 ★과적합(Overfitting)되었는지 일반화가 되지 않았는지를 판단할 수 없다.

가령, 데이터를 전부 학습하는 데에만 사용한다면 모델의 정확도는 높을 수 있으나
새로운 데이터가 주어졌을 때 예측도가 현저히 떨어지는,
과적합이 발생할 수 있다. (즉, 일반화에 실패할 수 있다.)

머신러닝의 목적은 성능을 높이고 일반화하는 데에 있다.

이를 위해 주어진 데이터를
학습데이터(training data)와

테스트데이터(test data, hold-out set)
으로 나누어야 한다.

```
# 지도학습(정답이 반드시 필요한 학습)
from sklearn.model_selection import train_test_split
```

> 훈련셋과 테스트셋으로 분할하는 데에 용이한 train_test_split 이란 함수를 제공한다

```
X_train, X_test, Y_train, Y_test = train_test_split(iris.data,   # iris 데이
                                                    iris.target,
                                                    # 학습시킬 값 (0, 1, 2로

                                                    test_size=0.3,
                                                    # 테스트사이즈를 30%로 선

                                                    random_state = 2019)
#훈련용2가지, 학습용2가지로 값이 반환되기 때문에 4가지의 변수를 작성해야 한다.
print(X_train)
print(X_test)
print(Y_train)
print(Y_test)
```

> X_train에는 전체 데이터 셋의 70%를, X_test 에는 나머지 30%의 데이터를 가진다.
> 원래 머신러닝 모델에 데이터를 학습시킬 때는 feature engineering 을 거쳐야 하는 데
> iris 셋에선 큰 필요가 없을 것 같아 그냥 있는 그대로의 데이터를 넣고 학습을 시킨다

KNN모델의 fit 메서드를 통해 입력(X_train)과 정답(Y_train)을 넣고 학습시킨다.

```
from sklearn.neighbors import KNeighborsClassifier # 1. 클래스를 가져오고
knn = KNeighborsClassifier(n_neighbors=3) # 2. 클래스를 이용해서 객체를 만들어줘
                                          # (어떤 값보다 적으면 맞다, 크면 틀리
                                          # 만들어진다.)
                                          # 근접계수에 대한 값을 써줘야 한다.
                                          # 그 값이 n_neighbors=3 이다.
knn.fit(X_train, Y_train) # 해당 객체가 가지고있는 fit 함수를 이용해서 학습 시킨다
```

학습을 시켰으므로 모델의 성능을 평가한다.

```
print('accuracy : {:.2f}'.format(knn.score(X_test,Y_test)))
# accuracy : 1.00 (문제와 정답을 넣어서 평가)
```

> 무작위데이터가 아닌 순서대로 들어있는 데이터를 사용해서 정확도가 100% 가 나오게 된다.

전체 데이터셋에서 30% 데이터를 분할하여 test 데이터로 만들었었다.
그 test 데이터에는 {입력:정답}을 모두 갖춘 데이터 셋이므로 학습시킨 모델을 평가하기에 적절하다.

knn.score(X_test,Y_test)는 X_test 데이터를 아까 학습시킨 knn모델에 넣고 Y_test 와 비교하여 정확도를 출력한다.
100%의 정확도를 얻었으며, 이는 테스트 세트에 있는 샘플 데이터들의 100%를 정확히 맞췄다는 뜻이다