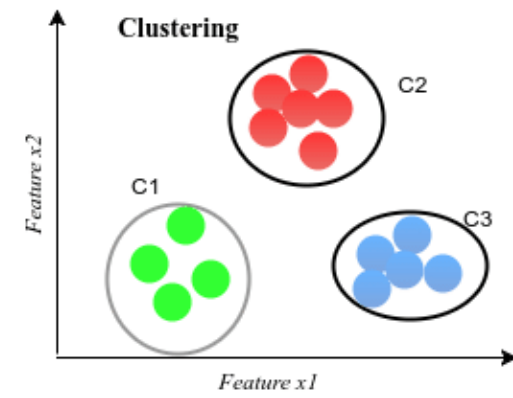
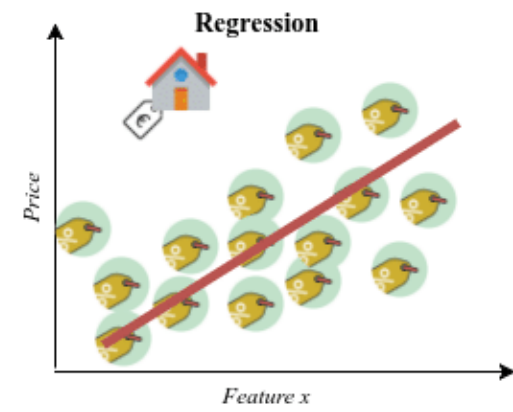
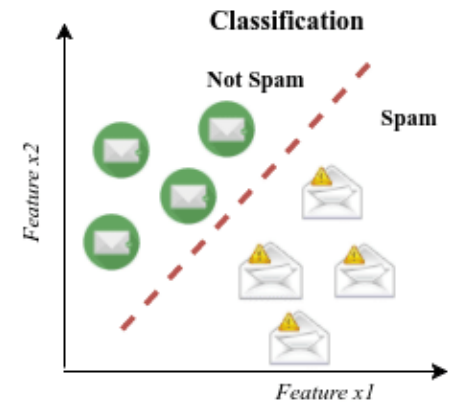
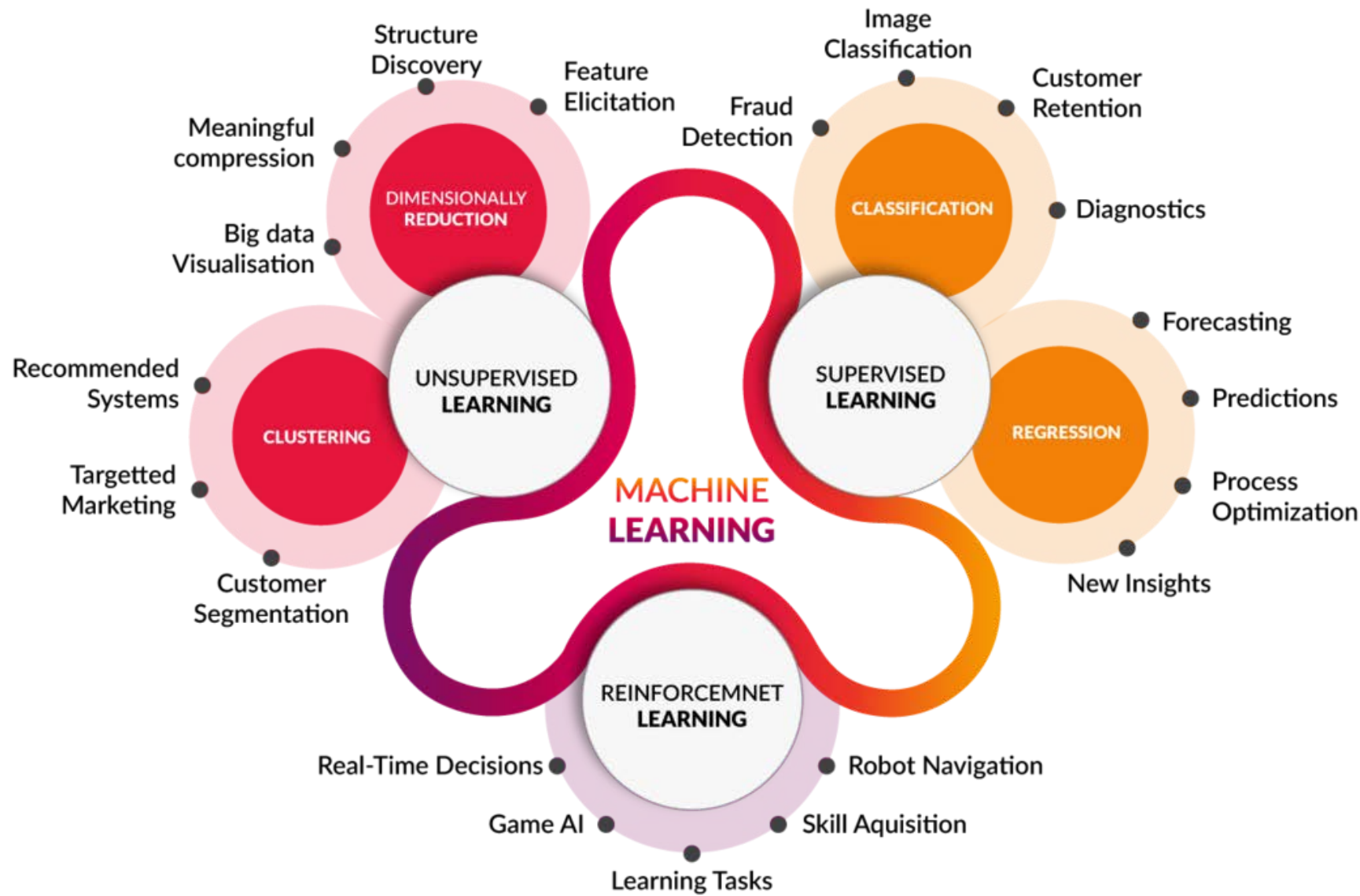


# 사이킷런으로 시작하는 머신러닝

## Chapter 02



※ 출처: <https://www.linkedin.com/pulse/machine-learning-business-intelligence-ibrahim-devops-engineer/>

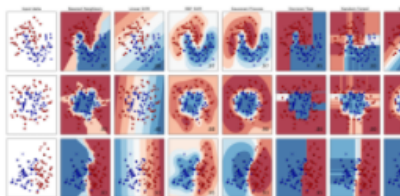
※ 출처: [https://www.researchgate.net/figure/Examples-of-real-life-problems-in-the-context-of-supervised-and-unsupervised-learning\\_fig8\\_319093376](https://www.researchgate.net/figure/Examples-of-real-life-problems-in-the-context-of-supervised-and-unsupervised-learning_fig8_319093376)

## Classification

Identifying which category an object belongs to.

**Applications:** Spam detection, image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, and more...



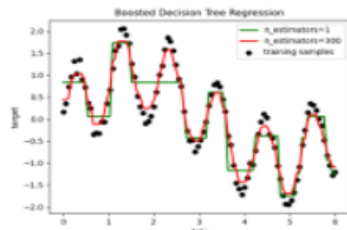
Examples

## Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, nearest neighbors, random forest, and more...



Examples

## Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, and more...



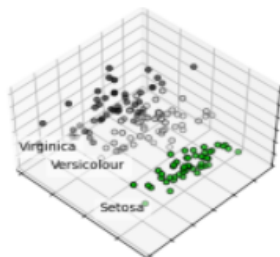
Examples

## Dimensionality reduction

Reducing the number of random variables to consider.

**Applications:** Visualization, Increased efficiency

**Algorithms:** k-Means, feature selection, non-negative matrix factorization, and more...



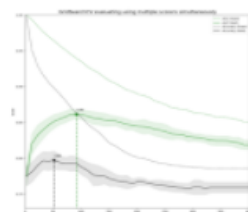
Examples

## Model selection

Comparing, validating and choosing parameters and models.

**Applications:** Improved accuracy via parameter tuning

**Algorithms:** grid search, cross validation, metrics, and more...



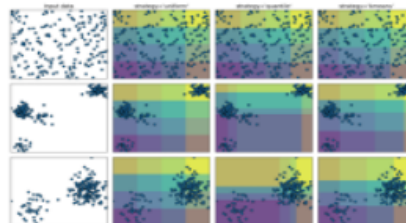
Examples

## Preprocessing

Feature extraction and normalization.

**Applications:** Transforming input data such as text for use with machine learning algorithms.

**Algorithms:** preprocessing, feature extraction, and more...



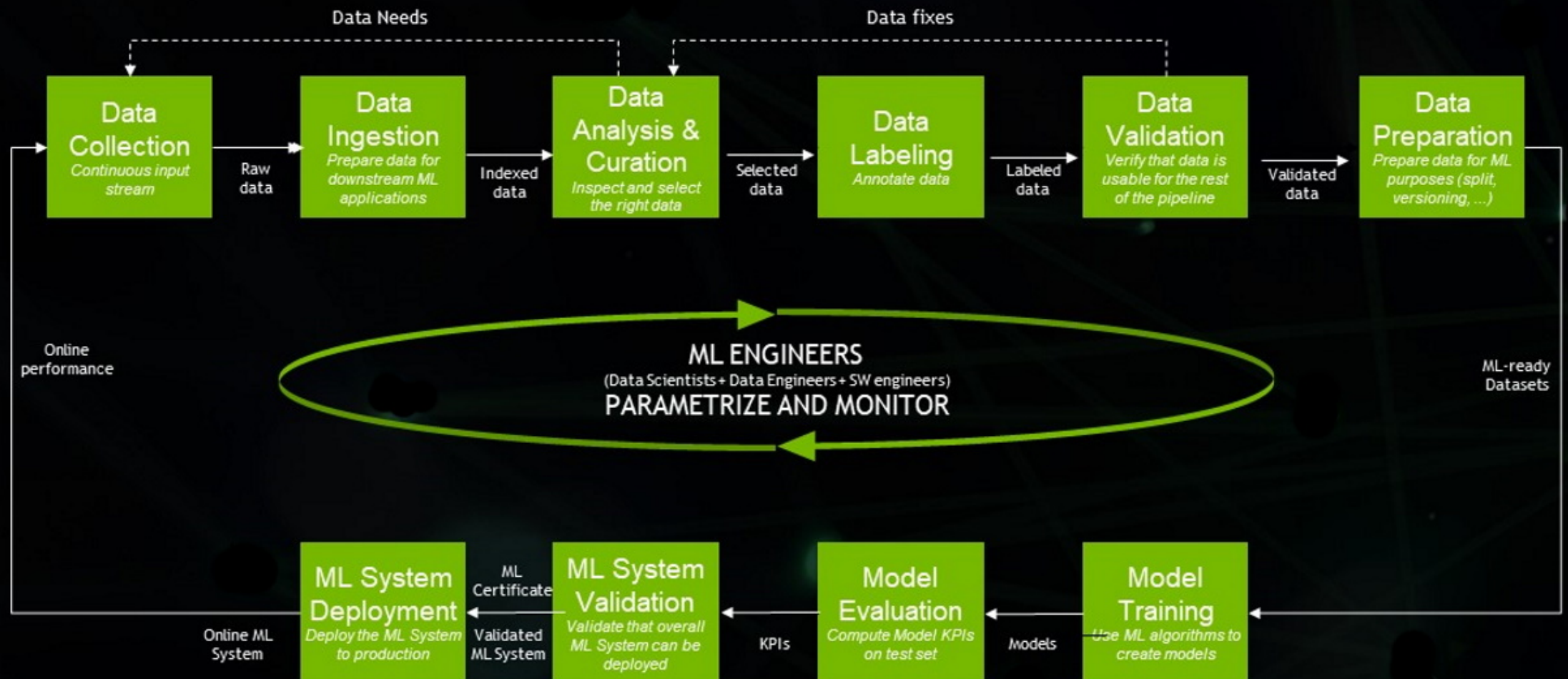
Examples

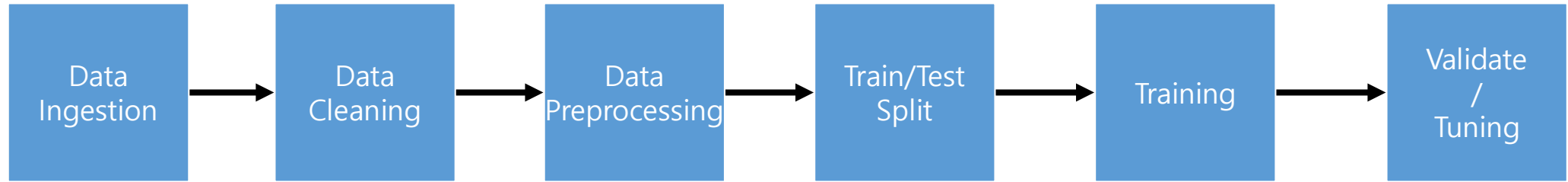
<https://scikit-learn.org/>

# scikit-learn 주요 모듈

분류	모듈명	설명
예제 데이터	sklearn.datasets	사이킷런에 내장되어 예제로 제공하는 데이터 세트
피처처리	sklearn.preprocessing	데이터 전처리에 필요한 다양한 가공 기능 제공(문자열을 숫자형 코드 값으로 인코딩, 정규화, 스케일링 등)
	sklearn.feature_selection	알고리즘에 큰 영향을 미치는 피처를 우선순위로 선택 작업을 수행하는 다양한 기능 제공
	sklearn.feature_extraction	텍스트 데이터나 이미지 데이터의 벡터화된 피처를 추출하는데 사용됨. 예를 들어 텍스트 데이터에서 Count Vectorizer나 Tf-Idf Vectorizer 등을 생성하는 기능 제공. 텍스트 데이터의 피처 추출은 sklearn.feature_extraction.text 모듈에, 이미지 데이터의 피처 추출은 sklearn.feature_extraction.image 모듈에 지원 API가 있음
피처 처리 & 차원 축소	sklearn.decomposition	차원 축소와 관련한 알고리즘을 지원하는 모듈이다. PCA, NMF, Truncated SVD 등을 통해 차원 축소 기능을 수행할 수 있다.
데이터 분리, 검증 & 파라미터 튜닝	sklearn.model_selection	교차 검증을 위한 학습용/테스트용 분리, 그리드 서치(Grid Search)로 최적 파라미터 추출 등의 API 제공
평가	sklearn.metrics	분류, 회귀, 클러스터링, 페어와이즈(Pairwise)에 대한 다양한 성능 측정 방법 제공. Accuracy, Precision, Recall, ROC-AUC, RMSE 등 제공
ML 알고리즘	sklearn.ensemble	앙상블 알고리즘 제공. 랜덤 포레스트, 에이다 부스팅, 그래디언트 부스팅 등을 제공
	sklearn.linear_model	주로 선형 회귀, 릿지(Ridge), 라쏘(Lasso) 및 로지스틱 회귀 등 회귀 관련 알고리즘을 지원. 또한 SGD(Stochastic Gradient Descent) 관련 알고리즘도 제공
	sklearn.naïve_bayes	나이브 베이즈 알고리즘 제공. 가우시안 NB, 다항 분포 NB 등
	sklearn.neighbors	최근접 이웃 알고리즘 제공. K-NN(K-Nearest Neighborhood) 등
	sklearn.svm	서포트 벡터 머신 알고리즘 제공
	sklearn.tree	의사 결정 트리 알고리즘 제공
	sklearn.cluster	비지도 클러스터링 알고리즘 제공 (K-평균, 계층형, DBSCAN 등)
유틸리티	sklearn.pipeline	피처 처리 등의 변환과 ML 알고리즘 학습, 예측 등을 함께 묶어서 실행할 수 있는 유틸리티 제공

# MLOPS: THE AI LIFECYCLE FOR IT PRODUCTION





**sklearn**

datasets

preprocessing

preprocessing

model\_selection

tree

mode\_selection

LabelEncoder

StandardScaler

train\_test\_split

DecisionTreeClassifier

cross\_val\_score

OneHotEncoder

MinMaxScaler

KFold

KNeighborsClassifier

GridSearchCV

StratifiedKFold

**pandas**

reader

get\_dummies

describe

plot

# Data Ingestion

# sklearn : datasets

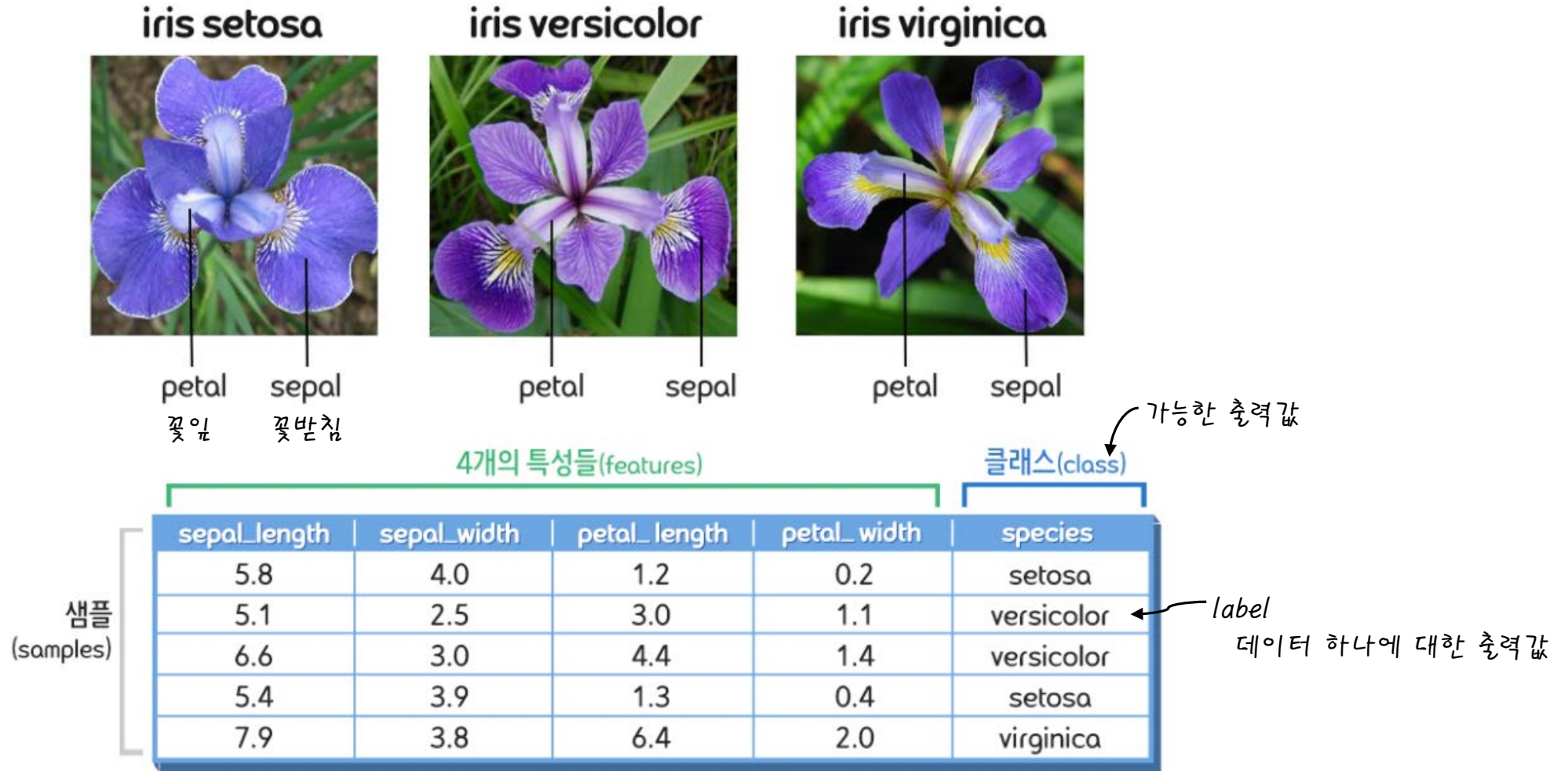
구분	API	Type	Comments
내장 예제 데이터 세트	datasets.load_boston()	Regression	미국 보스턴 집 특성 및 가격
	datasets.load_breast_cancer()	Classification	위스콘신 유방암 특성 및 악성/음성 레이블 포함
	datasets.load_diabetes()	Regression	당뇨
	datasets.load_digits()	Classification	0~9 숫자 이미지 픽셀
	datasets.load_iris()	Classification	붓꽃 특성 및 분류 정보 포함
원격 다운로드 데이터 세트	datasets.fetch_covtype()	Regression	토지 조사 내용
	datasets.fetch_20newsgroups()		뉴스 그룹 텍스트
	datasets.fetch_olivetti_faces()		얼굴 이미지
	datasets.fetch_lfw_people()		얼굴 이미지
	datasets.fetch_lfw_pairs()		얼굴 이미지
	datasets.fetch_rcv1()		로이터 뉴스 말뭉치
	datasets.fetch_mldata()		ML 웹사이트에서 다운로드
표본 데이터 생성	datasets.make_classifications()	Classification	분류를 위한 데이터 세트 생성
	datasets.make_blobs()	Clustering	클러스터링을 위한 데이터 세트 생성



# pandas : reader (I/O API)

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

# iris data (붓꽃 데이터)



# 실습



```
from sklearn.datasets import load_iris
```

```
iris = load_iris()
```

```
print(iris.DESCR)
```

```
.. _iris_dataset:
```

```
Iris plants dataset
```

```
-----
```

```
**Data Set Characteristics:**
```

```
:Number of Instances: 150 (50 in each of three classes)
```

```
:Number of Attributes: 4 numeric, predictive attributes and the class
```

```
:Attribute Information:
```

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
  - Iris-Setosa
  - Iris-Versicolour
  - Iris-Virginica

```
:Summary Statistics:
```

	Min	Max	Mean	SD	Class Correlation
sepal length:	4.3	7.9	5.84	0.83	0.7826
sepal width:	2.0	4.4	3.05	0.43	-0.4194
petal length:	1.0	6.9	3.76	1.76	0.9490 (high!)
petal width:	0.1	2.5	1.20	0.76	0.9565 (high!)

```
:Missing Attribute Values: None
```

```
:Class Distribution: 33.3% for each of 3 classes.
```

```
:Creator: R.A. Fisher
```

```
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
```

```
:Date: July, 1988
```

# 실습

py-ml-ch02-sklearn.ipynb ☆

파일 수정 보기 삽입 런타임 도구 도움말

RAM 디스크 수정 가능

content

- sample\_data
  - train.csv

```
[5] import pandas as pd

titanic_train = pd.read_csv('/content/sample_data/train.csv')
titanic_train.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

여기에 업로드하고 불러오도록 했다.  
(Colab 재시작 하면 사라진다)

sklearn은 문자열 입력을 허용하지 않는다.

결손값(결측치/NaN/Null)도 허용하지 않는다.

# **Data Cleaning**

# sklearn : preprocessing - LabelEncoder

[4] `import pandas as pd`

```
titanic_train = pd.read_csv('/content/sample_data/train.csv')
titanic_train.head(3)
```

PassengerId	Survived	Pclass	Name	Sex	Age	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0

입력은 숫자로 해줘야 하기에 문자열을 숫자로 변경해야 한다.

transformer 계열들은 “fit() → transform()” 으로 구현된다.

숫자로 변형된 값의 원본을 확인하고 싶으면 “inverse\_transform” 사용하면 된다.

`encoder.inverse_transform(titanic_train['Sex_encoded'])`

그런데, 사실 *scikit-learn*에서 문자열을 넣어줘도 알아서 해준다. (해보면 된다)

`titanic_train.head(3)`

PassengerId	Survived	Pclass	Name	Sex_encoded	
0	1	0	3	Braund, Mr. Owen Harris	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0
2	3	1	3	Heikkinen, Miss. Laina	0

[5] `print(titanic_train['Sex'])`

```
0      male
1     female
2     female
3     female
4      male
...
886     male
887    female
888    female
889     male
890     male
Name: Sex, Length: 891, dtype: object
```

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
encoder.fit(titanic_train['Sex'])
titanic_train['Sex_encoded'] = encoder.transform(titanic_train['Sex'])

print(titanic_train['Sex_encoded'])
```

```
0      1
1      0
2      0
3      0
4      1
...
886     1
887     0
888     0
889     1
890     1
Name: Sex_encoded, Length: 891, dtype: int64
```

# sklearn : preprocessing - OneHotEncoder

titanic\_train.head(5)

	PassengerId	Survived	Pclass	Name	Sex
0	1	0	3	Braund, Mr. Owen Harris	male
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female
2	3	1	3	Heikkinen, Miss. Laina	female
3	4	1	1	Futelle, Mrs. Jacques Heath (Lily May Peel)	female
4	5	0	3	Allen, Mr. William Henry	male

labels\_raw

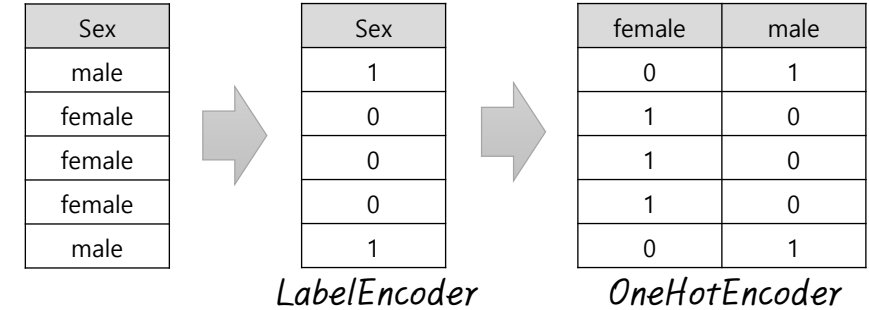
```
[1 0 0 0 1 1 1 1 0 0 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 1 1 0 0 1 1 1 1
 1 0 0 0 0 1 0 0 1 1 0 1 0 1 1 0 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 1 1
 ... 0 1 1]
```

labels\_reshape

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

labels\_onehot

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



```
[13] from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
encoder.fit(titanic_train['Sex'])
labels_raw = encoder.transform(titanic_train['Sex'])

labels_reshape = labels.reshape(-1, 1)
```

```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder()
encoder.fit(labels_reshape)
labels_onehot = encoder.transform(labels_reshape)

print(labels_onehot.toarray())
```

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

# pandas : get\_dummies

```
[18] import pandas as pd

titanic_train = pd.read_csv('/content/sample_data/train.csv')

titanic_train.Sex
```

0	male
1	female
2	female
3	female
4	male
...	...
886	male
887	female
888	female
889	male
890	male

Name: Sex, Length: 891, dtype: object

```
Sex_OneHot = pd.get_dummies(titanic_train.Sex)
```

Sex\_OneHot

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1
...	...	...
886	0	1
887	1	0
888	1	0
889	0	1
890	0	1

891 rows x 2 columns

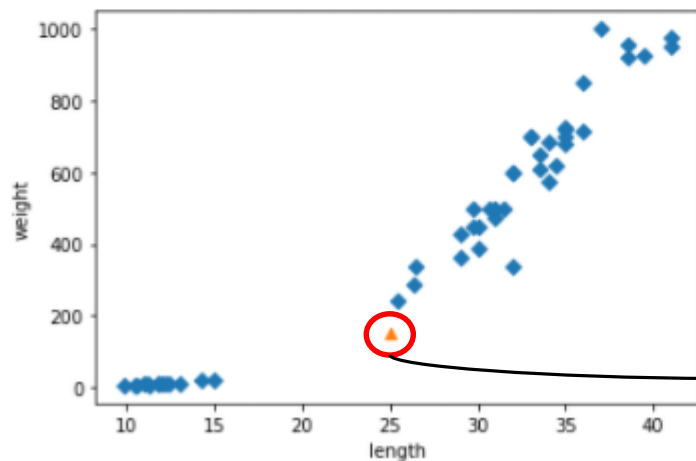
*pandas 만세 !!*



# Data Preprocessing

# Why Scaler ???

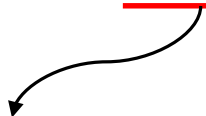
```
fish_length = [25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7, 30.0, 30.0, 30.7, 31.0, 31.0,  
              31.5, 32.0, 32.0, 32.0, 33.0, 33.0, 33.5, 33.5, 34.0, 34.0, 34.5, 35.0,  
              35.0, 35.0, 35.0, 36.0, 36.0, 37.0, 38.5, 38.5, 39.5, 41.0, 41.0, 9.8,  
              10.5, 10.6, 11.0, 11.2, 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]  
  
import matplotlib.pyplot as plt  
  
plt.scatter(fish_length, fish_weight, marker='D')  
plt.scatter(25, 150, marker='^')  
plt.xlabel('length')  
plt.ylabel('weight')  
plt.show()
```



왼쪽에 있는 아이들과 상당히 떨어져 있는 것으로 보이지만,  
값 간격을 보면 *length*보다 *weight*의 차이가 훨씬 크다.

# Scaler

스케일링을 통해 다차원의 값들을 비교 분석하기 쉽게 만들어주며, 자료의 오버 플로우(overflow)나 언더 플로우(underflow)를 방지 하고, 독립 변수의 공분산 행렬의 조건수(condition number)를 감소시켜 최적화 과정에서의 안정성 및 수렴 속도를 향상 시킨다.

 argument에서의 작은 변화의 비율에 대해 함수가 얼마나 변화할 수 있는지 에 대한 *argument measure*이다.

조건수가 크면 약간의 오차만 있어도 해가 전혀 다른 값을 가진다. 따라서 조건수가 크면 회귀분석을 사용한 예측 값도 오차가 커지게 된다.

종류		설명	
1	StandardScaler	기본 스케일. 평균과 표준편차 사용	아웃라이어의 영향 큼
2	MinMaxScaler	최대/최소값이 각각 1, 0이 되도록 스케일링	
3	MaxAbsScaler	최대절대값과 0이 각각 1, 0이 되도록 스케일링	
4	RobustScaler	중앙값(median)과 IQR(interquartile range) 사용. 아웃라이어의 영향을 최소화	

# pandas : describe & matplotlib

```
[27] from sklearn.datasets import load_iris
import pandas as pd

iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

iris_df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows x 4 columns

```
iris_df.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

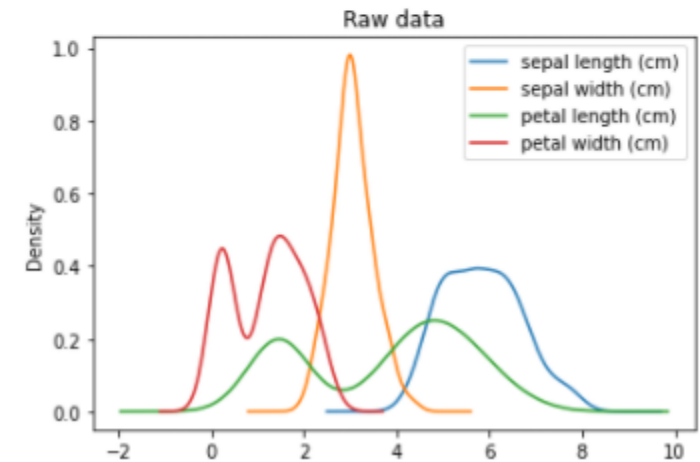
```
[29] iris_df.var()
```

```
sepal length (cm)    0.685694
sepal width (cm)     0.189979
petal length (cm)    3.116278
petal width (cm)     0.581006
dtype: float64
```

```
import matplotlib.pyplot as plt

iris_df.plot(kind='kde', title='Raw data')

plt.show()
```



# preprocessing : StandardScaler

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

iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
```

```
import matplotlib.pyplot as plt
```

```
fig, ax = plt.subplots(1, 2, figsize=(12, 4))
```

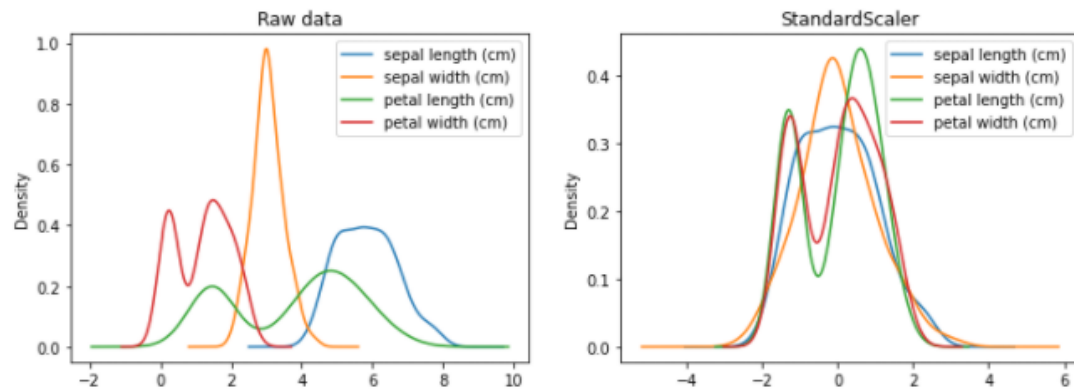
```
iris_df.plot(kind='kde', title='Raw data', ax=ax[0])
```

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)

iris_df_std = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
iris_df_std.plot(kind='kde', title='StandardScaler', ax=ax[1])
```

```
plt.show()
```

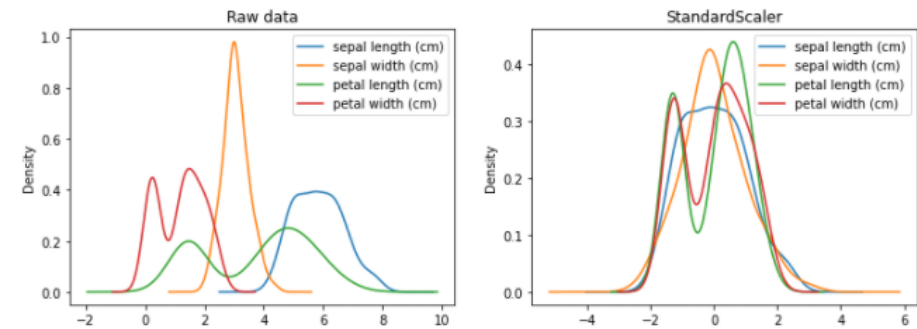


```
fig, ax = plt.subplots(1, 2, figsize=(12, 4))
iris_df.plot(kind='kde', title='Raw data', ax=ax[0])

scaler = StandardScaler()
scaler.fit_transform(iris_df)

iris_df_std = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
iris_df_std.plot(kind='kde', title='StandardScaler', ax=ax[1])

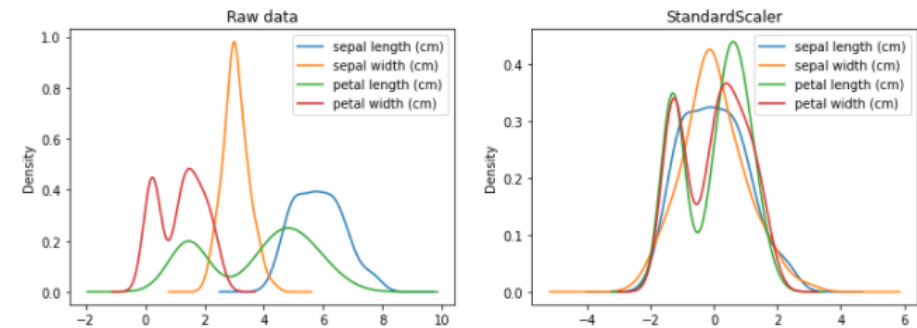
plt.show()
```



```
[50] fig, ax = plt.subplots(1, 2, figsize=(12, 4))
iris_df.plot(kind='kde', title='Raw data', ax=ax[0])
```

```
iris_df_std = pd.DataFrame(data=StandardScaler().fit_transform(iris_df), columns=iris.feature_names)
iris_df_std.plot(kind='kde', title='StandardScaler', ax=ax[1])
```

```
plt.show()
```



# preprocessing : MinMaxScaler

```

from sklearn.datasets import load_iris
import pandas as pd

iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 2, figsize=(12, 4))

iris_df.plot(kind='kde', title='Raw data', ax=ax[0])

```

```

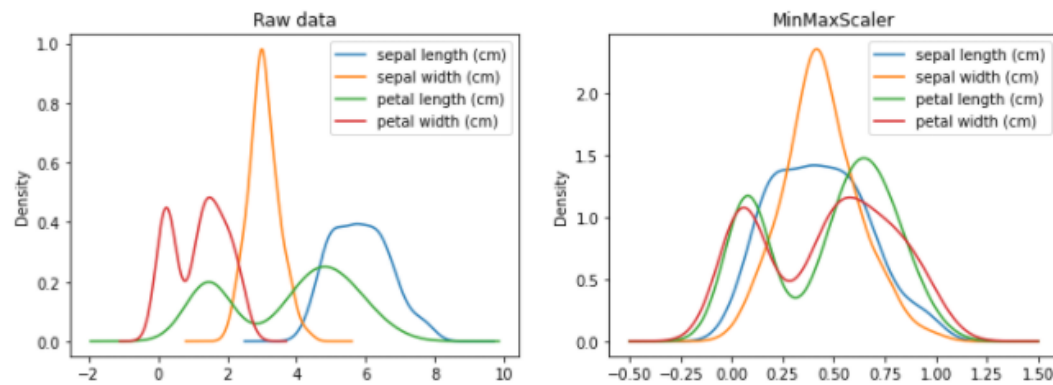
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)

iris_df_minmax = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
iris_df_minmax.plot(kind='kde', title='MinMaxScaler', ax=ax[1])

plt.show()

```



```

[54] fig, ax = plt.subplots(1, 2, figsize=(12, 4))
iris_df.plot(kind='kde', title='Raw data', ax=ax[0])

```

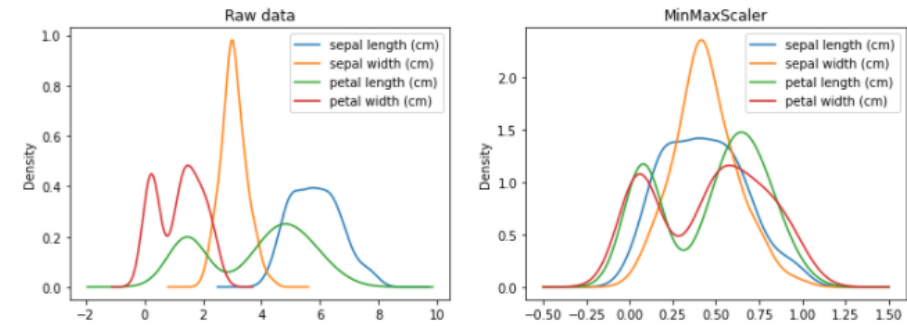
```

scaler = MinMaxScaler()
scaler.fit_transform(iris_df)

iris_df_minmax = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
iris_df_minmax.plot(kind='kde', title='MinMaxScaler', ax=ax[1])

plt.show()

```



```

fig, ax = plt.subplots(1, 2, figsize=(12, 4))
iris_df.plot(kind='kde', title='Raw data', ax=ax[0])

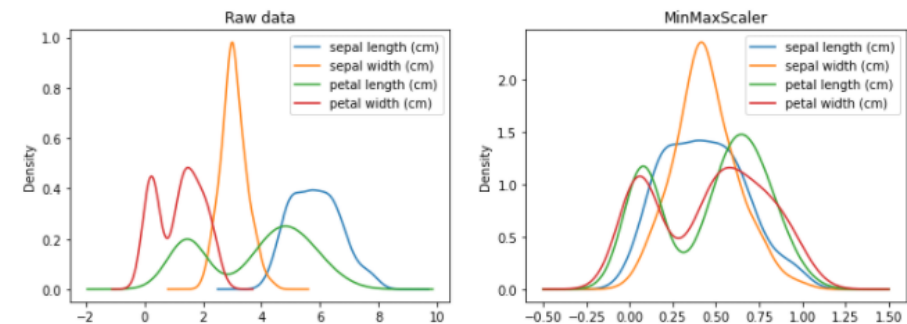
```

```

iris_df_minmax = pd.DataFrame(data=MinMaxScaler().fit_transform(iris_df), columns=iris.feature_names)
iris_df_minmax.plot(kind='kde', title='MinMaxScaler', ax=ax[1])

plt.show()

```



# preprocessing : Standard vs MinMaxScaler

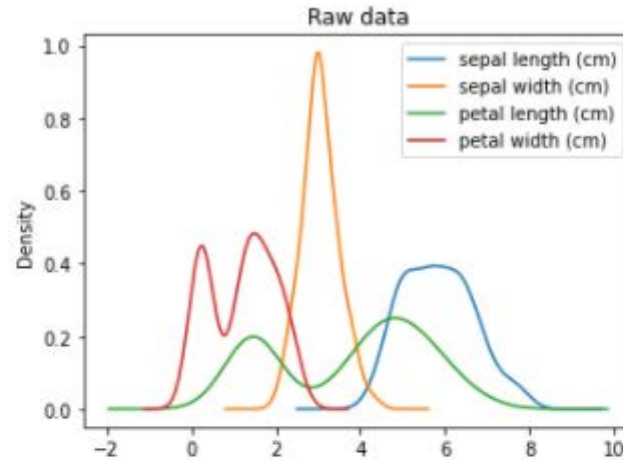
표준화 (Standardization)

- 가우시안 정규 분포로 변환 (평균 0, 분산 1)

$$x_{i\_new} = \frac{x_i - \text{mean}(x)}{\text{stdev}(x)}$$

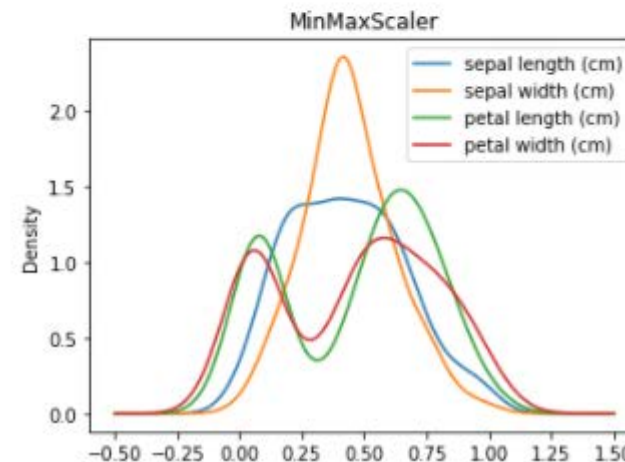
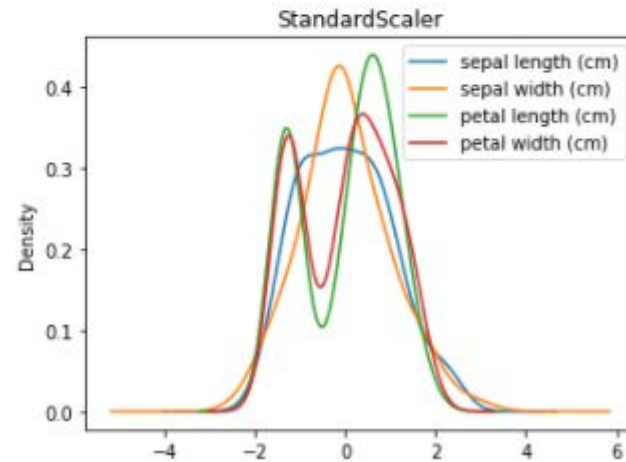
$\text{mean}(x)$  : 피쳐  $x$ 의 평균

$\text{stdev}(x)$  : 피쳐  $x$ 의 표준편차



정규화 (Normalization)

- 각 피쳐의 크기를 통일하기 위해 변환



$$x_{i\_new} = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

$$x_{i\_new} = \frac{x_i}{\sqrt{x_i^2 + y_i^2 + z_i^2}}$$

sklearn의 normalization은

선형대수의 normalization을 적용

# **Train/Test Split**



# model\_selection : train\_test\_split()

```
[65] from sklearn.datasets import load_iris

iris = load_iris()

iris.data.shape

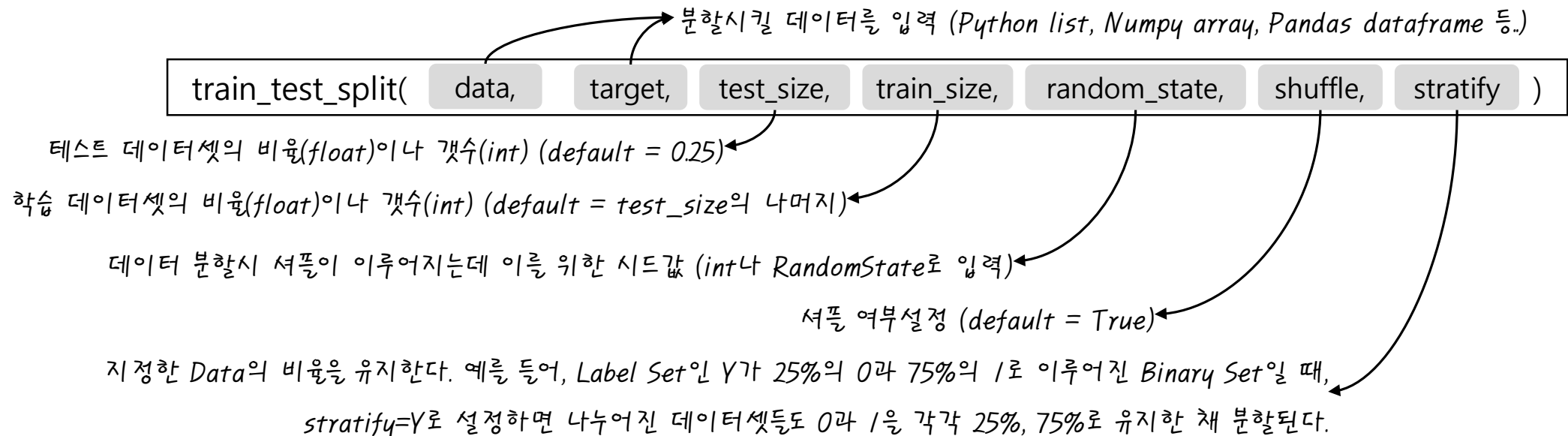
(150, 4)

from sklearn.model_selection import train_test_split

train_data, test_data, train_target, test_target = train_test_split(iris.data, iris.target, test_size=0.3, random_state=121)

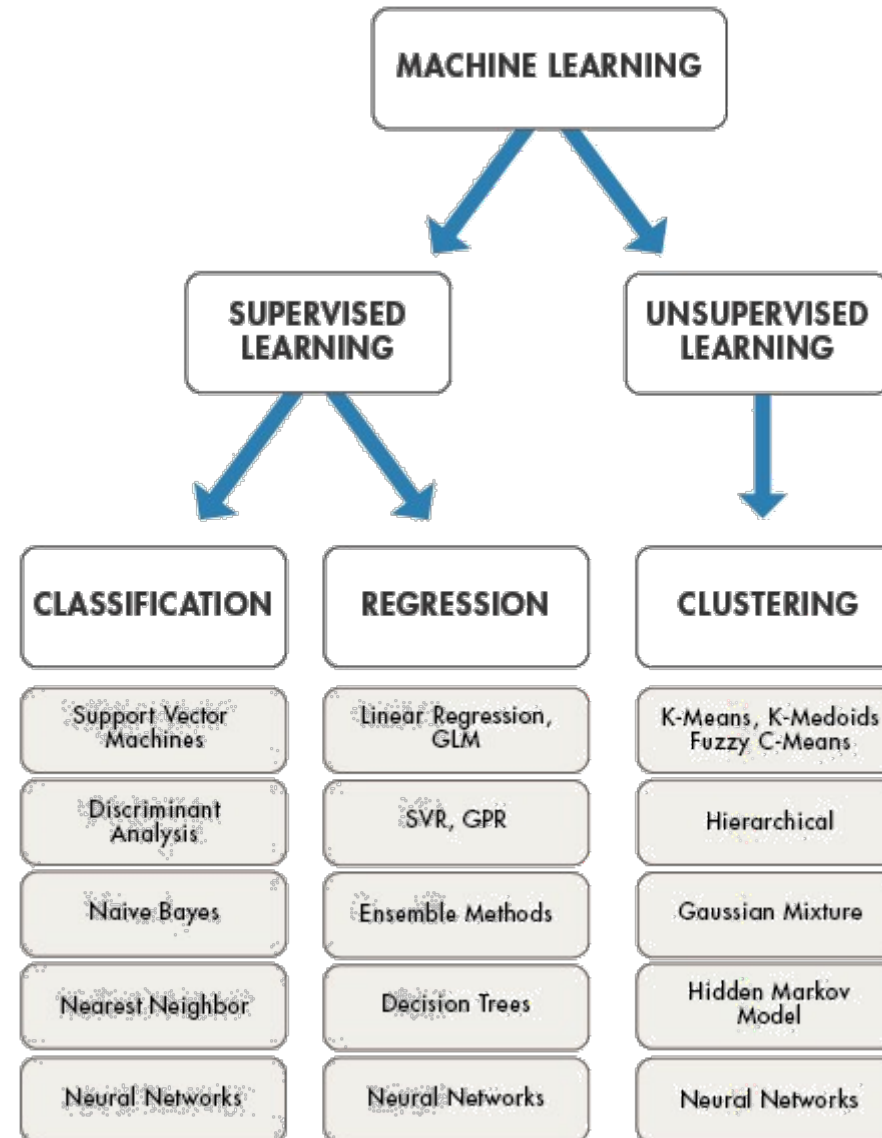
test_data.shape

(45, 4)
```

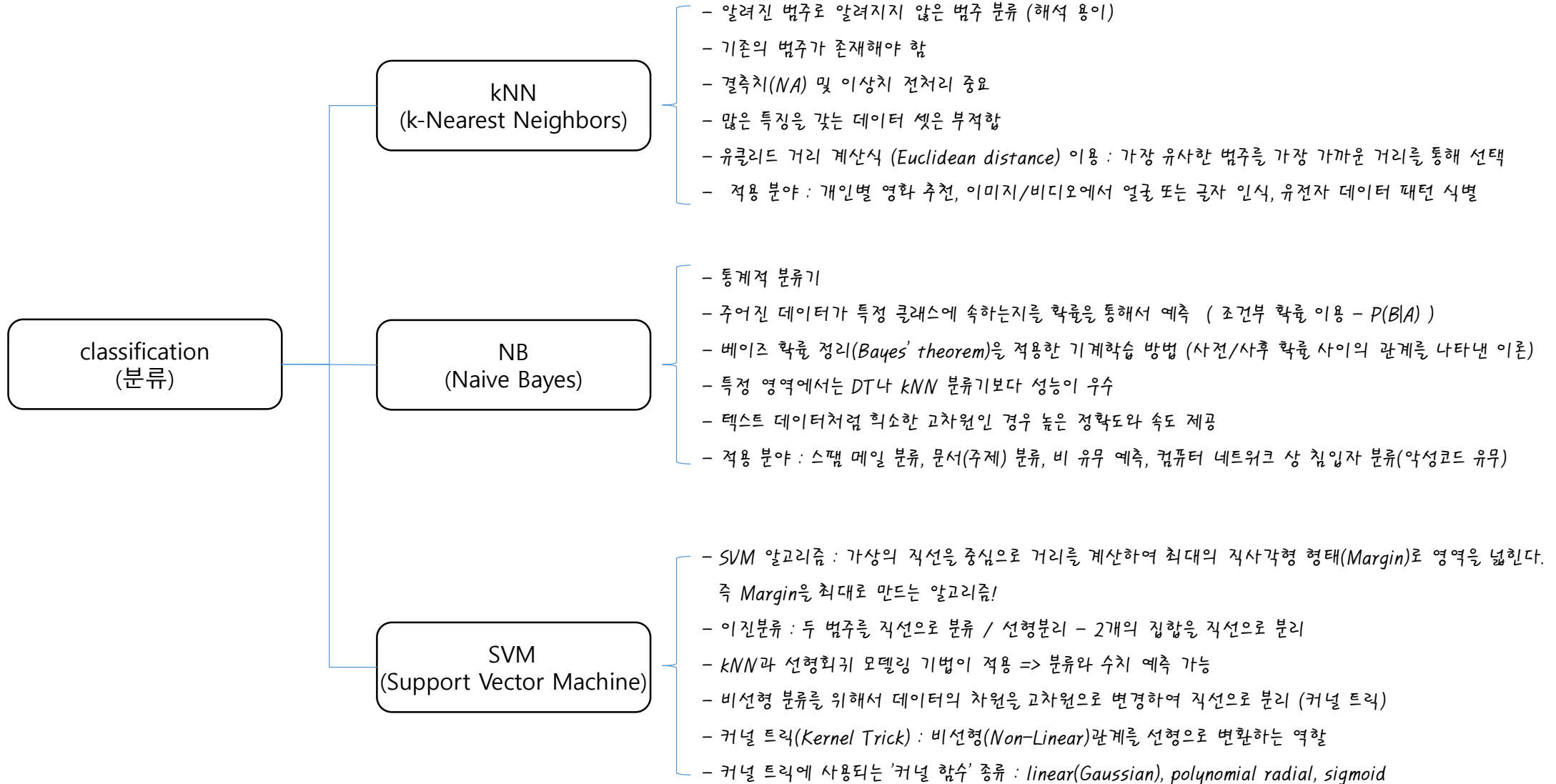


# Training

# Machine Learning Algorithm



# Classification



# Decision Tree Learning

어떤 항목에 대한 관측값과 목표값을 연결시켜주는 예측 모델

트리 모델 중 목표 변수가 유한한 수의 값을 가지는 것을 분류 트리라 한다.

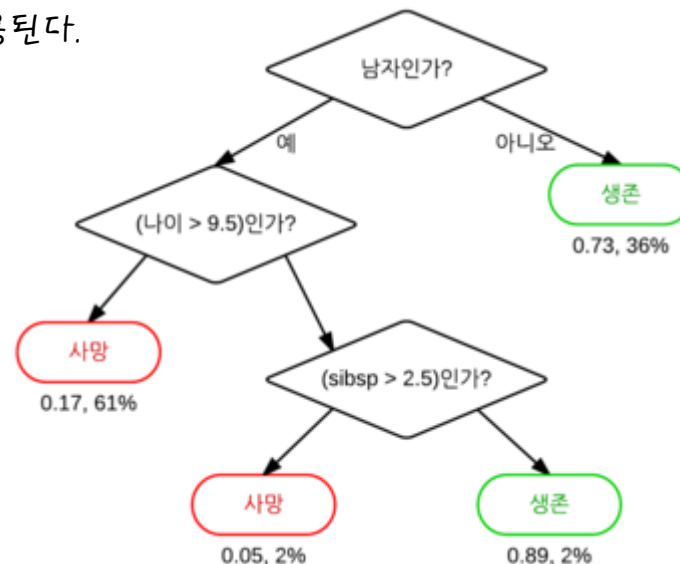
이 트리 구조에서 잎(리프 노드)은 클래스 라벨을 나타내고 가지는 클래스 라벨과 관련 있는 특징들의 논리곱을 나타낸다.

결정 트리 중 목표 변수가 연속하는 값, 일반적으로 실수를 가지는 것은 회귀 트리라 한다.

의사 결정 분석에서 결정 트리는 시각적이고 명시적인 방법으로 의사 결정 과정과 결정된 의사를 보여주는데 사용된다.

데이터 마이닝 분야에서 결정 트리는 결정된 의사보다는 자료 자체를 표현하는데 사용된다.

다만, 데이터 마이닝의 결과로서의 분류 트리는 의사 결정 분석의 입력 값으로 사용될 수 있다.



# Decision Tree Learning

```
▶ from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

iris = load_iris()
train_data, test_data, train_target, test_target = train_test_split(iris.data, iris.target, test_size=0.3, random_state=121)

dt_clf = DecisionTreeClassifier()
dt_clf.fit(train_data, train_target)

pred = dt_clf.predict(test_data)
print('예측 정확도: {0:.4f}'.format(accuracy_score(test_target, pred)))
```

예측 정확도: 0.9556

```
▶ from sklearn.datasets import load_iris
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

scaler = MinMaxScaler()
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)

train_data, test_data, train_target, test_target = train_test_split(iris_scaled, iris.target, test_size=0.3, random_state=121)

dt_clf = DecisionTreeClassifier()
dt_clf.fit(train_data, train_target)

pred = dt_clf.predict(test_data)
print('예측 정확도: {0:.4f}'.format(accuracy_score(test_target, pred)))
```

예측 정확도: 0.9556

# KNeighborsClassifier

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

iris = load_iris()
train_data, test_data, train_target, test_target = train_test_split(iris.data, iris.target, test_size=0.3, random_state=121)

dt_clf = KNeighborsClassifier()
dt_clf.fit(train_data, train_target)

pred = dt_clf.predict(test_data)
print('예측 정확도: {0:.4f}'.format(accuracy_score(test_target, pred)))
```

예측 정확도: 0.9556

```
from sklearn.datasets import load_iris
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

scaler = MinMaxScaler()
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)

train_data, test_data, train_target, test_target = train_test_split(iris_scaled, iris.target, test_size=0.3, random_state=121)

dt_clf = KNeighborsClassifier()
dt_clf.fit(train_data, train_target)

pred = dt_clf.predict(test_data)
print('예측 정확도: {0:.4f}'.format(accuracy_score(test_target, pred)))
```

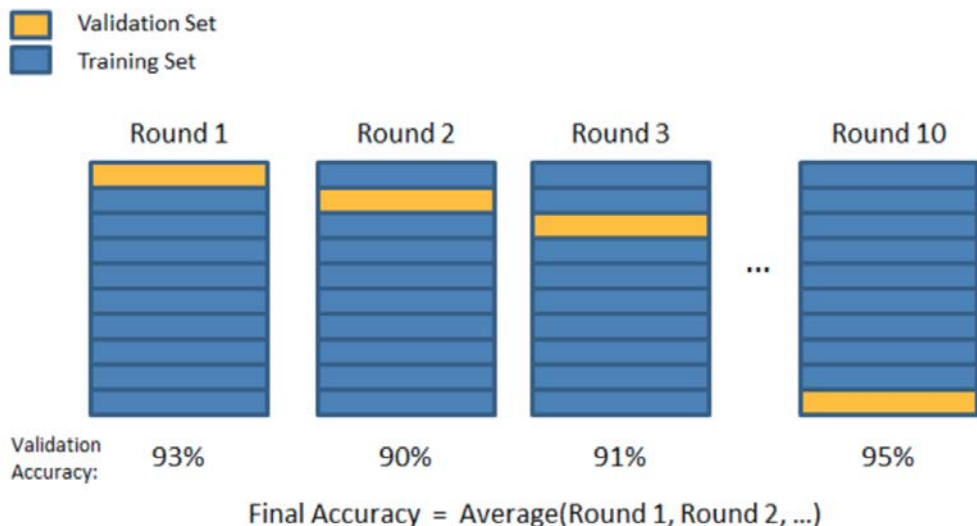
예측 정확도: 0.9778

**Validate / Tuning**



# model\_selection : KFold()

교차검증



index 값에서 보이는 것과 같이,

KFold는 그냥 순서대로 잘라내서 분할해준다.

→ 불균형한 데이터 분포인 경우 문제가 있을 수 있다!

```
from sklearn.datasets import load_iris
from sklearn.model_selection import KFold

iris = load_iris()

print("Raw Size : %s" % iris.data.shape[0])

kfold = KFold(n_splits=5)

for n_iter, (train_index, test_index) in enumerate(kfold.split(iris.data), start=1):

    train_data, test_data = iris.data[train_index], iris.data[test_index]
    train_target, test_target = iris.target[train_index], iris.target[test_index]

    print("n_iter=%s, train size=%s, test size=%s" % (n_iter, train_data.shape[0], test_data.shape[0]))
    print("  index : %s" % test_index)
```

Raw Size : 150  
n\_iter=1, train size=120, test size=30  
index : [ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23  
24 25 26 27 28 29]  
n\_iter=2, train size=120, test size=30  
index : [30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53  
54 55 56 57 58 59]  
n\_iter=3, train size=120, test size=30  
index : [60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83  
84 85 86 87 88 89]  
n\_iter=4, train size=120, test size=30  
index : [ 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107  
108 109 110 111 112 113 114 115 116 117 118 119]  
n\_iter=5, train size=120, test size=30  
index : [120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137  
138 139 140 141 142 143 144 145 146 147 148 149]

# model\_selection : StratifiedKFold()

```
from sklearn.datasets import load_iris
from sklearn.model_selection import StratifiedKFold

iris = load_iris()

print("Raw Size : %s" % iris.data.shape[0])

skfold = StratifiedKFold(n_splits=5)

for n_iter, (train_index, test_index) in enumerate(skfold.split(iris.data, iris.target), start=1):

    train_data, test_data = iris.data[train_index], iris.data[test_index]
    train_target, test_target = iris.target[train_index], iris.target[test_index]

    print("n_iter=%s, train size=%s, test size=%s" % (n_iter, train_data.shape[0], test_data.shape[0]))
    print("    index : %s" % test_index)
```

```
Raw Size : 150
n_iter=1, train size=120, test size=30
index : [ 0  1  2  3  4  5  6  7  8  9 50 51 52 53 54 55 56 57
58 59 100 101 102 103 104 105 106 107 108 109]
n_iter=2, train size=120, test size=30
index : [ 10 11 12 13 14 15 16 17 18 19 60 61 62 63 64 65 66 67
68 69 110 111 112 113 114 115 116 117 118 119]
n_iter=3, train size=120, test size=30
index : [ 20 21 22 23 24 25 26 27 28 29 70 71 72 73 74 75 76 77
78 79 120 121 122 123 124 125 126 127 128 129]
n_iter=4, train size=120, test size=30
index : [ 30 31 32 33 34 35 36 37 38 39 80 81 82 83 84 85 86 87
88 89 130 131 132 133 134 135 136 137 138 139]
n_iter=5, train size=120, test size=30
index : [ 40 41 42 43 44 45 46 47 48 49 90 91 92 93 94 95 96 97
98 99 140 141 142 143 144 145 146 147 148 149]
```

→ target 분포에 맞춰서 데이터를 분류해준다.

# model\_selection : cross\_val\_score()

```

▶ from sklearn.datasets import load_iris
  from sklearn.tree import DecisionTreeClassifier
  from sklearn.model_selection import train_test_split, cross_val_score, cross_validate
  import numpy as np

  iris = load_iris()

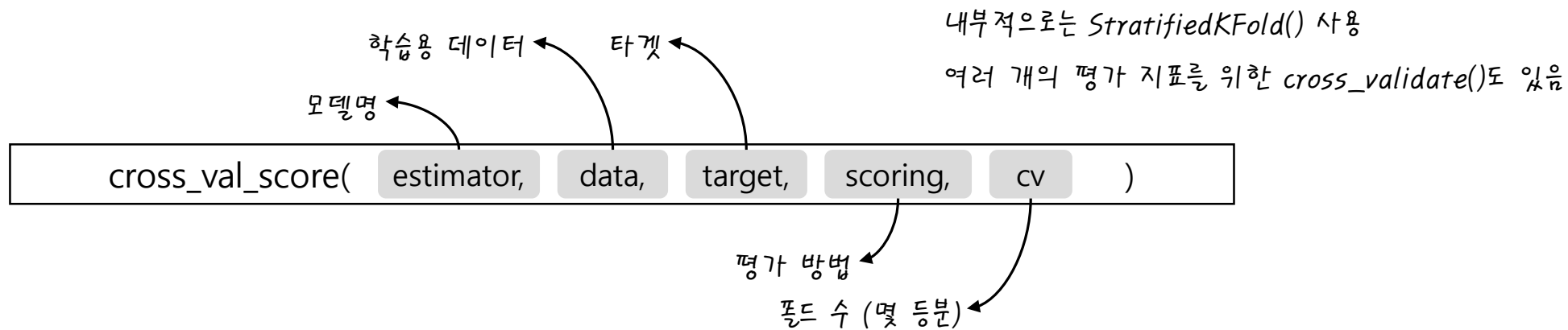
  train_data, test_data, train_target, test_target = train_test_split(iris.data, iris.target, test_size=0.3, random_state=121)

  dt_clf = DecisionTreeClassifier()

  scores = cross_val_score(dt_clf, train_data, train_target, scoring='accuracy', cv=3)
  print('교차 검증별 정확도 :', np.round(scores, 4))
  print('평균 검증 정확도 :', np.round(np.mean(scores), 4))

  교차 검증별 정확도 : [0.9429 0.9143 0.9429]
  평균 검증 정확도 : 0.9333

```



# model\_selection : GridSearchCV() – 1/2

```

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
import pandas as pd

iris = load_iris()
train_data, test_data, train_target, test_target = train_test_split(iris.data, iris.target, test_size=0.3, random_state=121)

dt_clf = DecisionTreeClassifier()

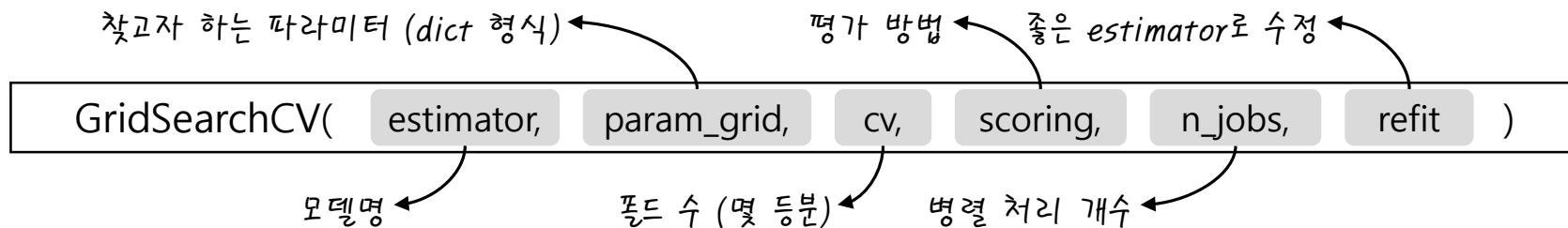
parameters = {'max_depth' : [1,2,3], 'min_samples_split' : [2,3]}

dt_grid = GridSearchCV(dt_clf, param_grid=parameters, cv=3, refit=True)
dt_grid.fit(train_data, train_target)

scores_df = pd.DataFrame(dt_grid.cv_results_)
scores_df[['params', 'mean_test_score', 'rank_test_score', 'split0_test_score', 'split1_test_score', 'split2_test_score']]

```

	params	mean_test_score	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	{'max_depth': 1, 'min_samples_split': 2}	0.657143	5	0.657143	0.657143	0.657143
1	{'max_depth': 1, 'min_samples_split': 3}	0.657143	5	0.657143	0.657143	0.657143
2	{'max_depth': 2, 'min_samples_split': 2}	0.933333	3	0.942857	0.914286	0.942857
3	{'max_depth': 2, 'min_samples_split': 3}	0.933333	3	0.942857	0.914286	0.942857
4	{'max_depth': 3, 'min_samples_split': 2}	0.942857	1	0.971429	0.914286	0.942857
5	{'max_depth': 3, 'min_samples_split': 3}	0.942857	1	0.971429	0.914286	0.942857



# model\_selection : GridSearchCV() – 2/2

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score

iris = load_iris()
train_data, test_data, train_target, test_target = train_test_split(iris.data, iris.target, test_size=0.3, random_state=121)

dt_clf = DecisionTreeClassifier()

parameters = {'max_depth' : [1,2,3], 'min_samples_split' : [2,3]}

dt_grid = GridSearchCV(dt_clf, param_grid=parameters, cv=3, refit=True)
dt_grid.fit(train_data, train_target)

model = dt_grid.best_estimator_
pred = model.predict(test_data)
print('예측 정확도: {0:.4f}'.format(accuracy_score(test_target, pred)))

예측 정확도: 0.9556
```

<https://kahoot.it>