



# Aichemist Session

CHAP 01 파이썬 기반의 머신러닝과 생태계 이해

## 머신러닝의 분류

- 데이터를 기반으로 패턴을 학습하고 결과를 예측하는 알고리즘 기법
- 소프트웨어 코드만으로는 해결하기 어려운 복잡한 문제들에 활용
- 머신러닝은 문제를 관통하는 일정한 패턴을 찾기 어려운 경우에도, 데이터를 기반으로 **패턴**을 인지해 문제 해결

# 머신러닝의 개념



# 지도학습 vs 비지도학습

1. 지도학습 : 레이블 데이터를 활용한 학습

- 분류
- 회귀
- 추천 시스템
- 시각/음성 감지/인지
- 텍스트 분석, NLP

2. 비지도학습 : Unlabeled 데이터를 비슷한 특징끼리 모아 새로운 데이터에 대한 결과 예측

- 클러스터링
- 차원 축소

+ ) 강화학습 : 데이터 없이 그냥 해보면서 그에 따른 보상을 받으며 학습하는 것

# 데이터 전쟁

“garbage in, garbage out”

머신러닝 알고리즘도 중요하지만, 좋은 데이터가 있어야 좋은 결과도 나오는 것.

*최적의 머신러닝 알고리즘을 구축하는 능력도 중요하지만,*

*데이터를 이해하고 효율적으로 가공, 처리, 추출해*

*최적의 데이터를 기반으로 알고리즘을 구동할 수 있도록 준비하는 능력이 더 중요할 수도 있다.*

# 넘파이

- 파이썬에서 선형대수 기반의 프로그램을 쉽게 만들 수 있도록 지원하는 대표적인 패키지
- 넘파이 모듈 불러오기  
`import numpy as np`
- 알고리즘의 입력 데이터와 출력 데이터를 넘파이 배열 타입으로 사용하므로  
넘파이를 이해하는 것은 파이썬 기반의 머신러닝에서 매우 중요!

# 판다스

## 기본 이해

- Pandas 핵심 개체 : DataFrame
- Series와 DataFrame
- **Series** : 칼럼이 하나뿐인 데이터 구조체
- **DataFrame**: 칼럼이 여러 개인 데이터 구조체 (여러 개의 Series)

[\\*numpy&pandas에 관한 자세한 사항은 백과사전 pdf를 참고 !](#)



# Aichemist Session

CHAP 02 사이킷런으로 시작하는 머신러닝



# CONTENTS

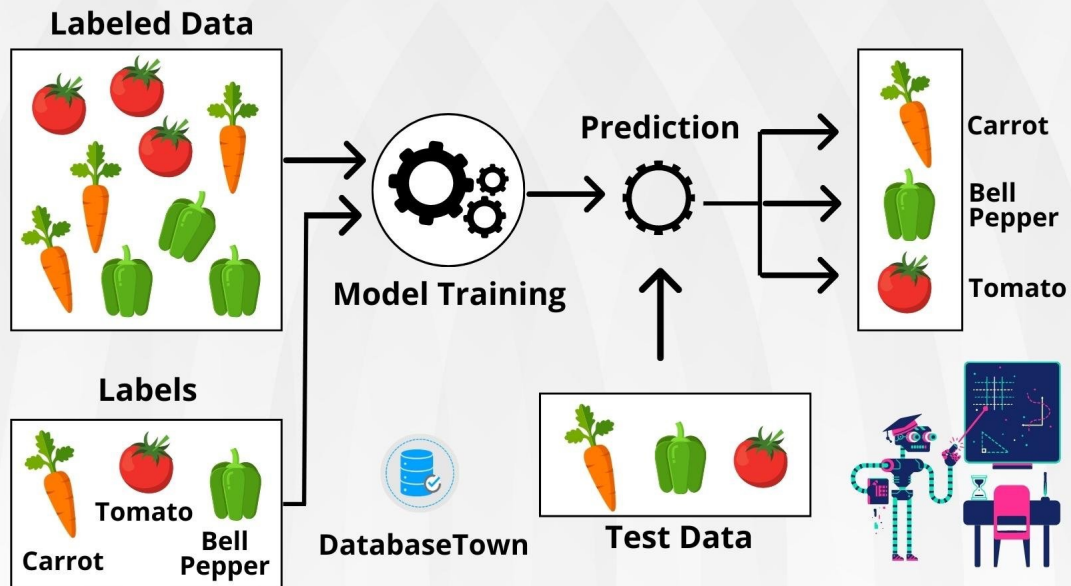
## 사이킷런으로 시작하는 머신러닝

1. 머신러닝의 분류
2. 붓꽃 품종 예측 머신러닝 모델
3. 사이킷런 기반 프레임워크 익히기
4. Model Selction 모듈 소개
5. 데이터 전처리
6. 데이터 분석의 프로세스

# 1. 지도학습 vs 비지도 학습

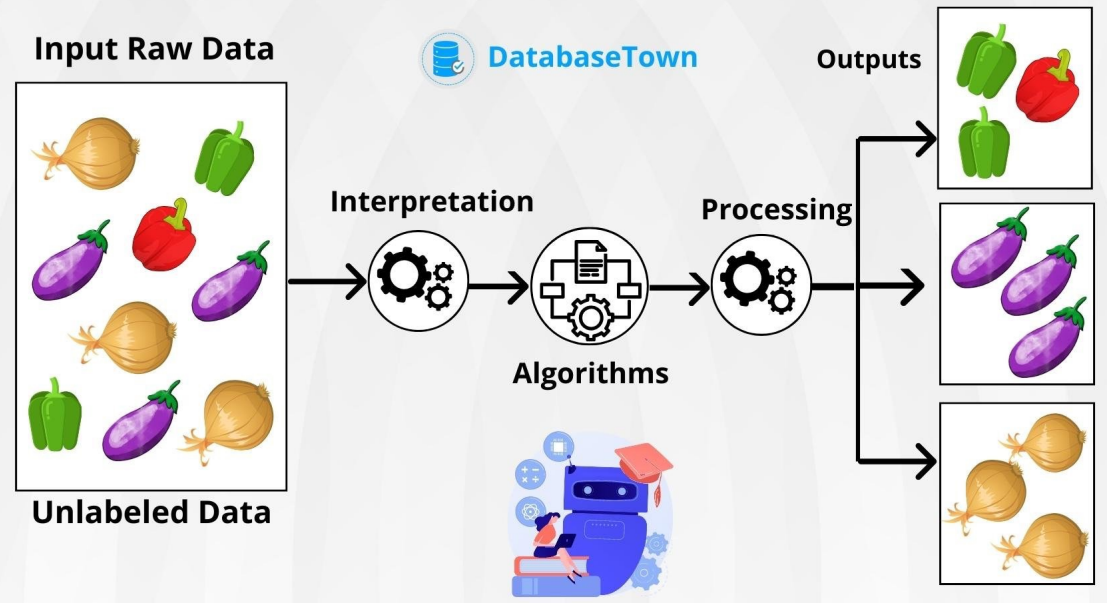
## SUPERVISED LEARNING

Supervised machine learning is a branch of artificial intelligence that focuses on training models to make predictions or decisions based on labeled training data.



## UNSUPERVISED LEARNING

Unsupervised learning is a type of machine learning where the algorithm learns from unlabeled data without any predefined outputs or target variables.



---

# 02.

첫 번째 머신러닝 만들어보기 – 붓꽃 품종 예측

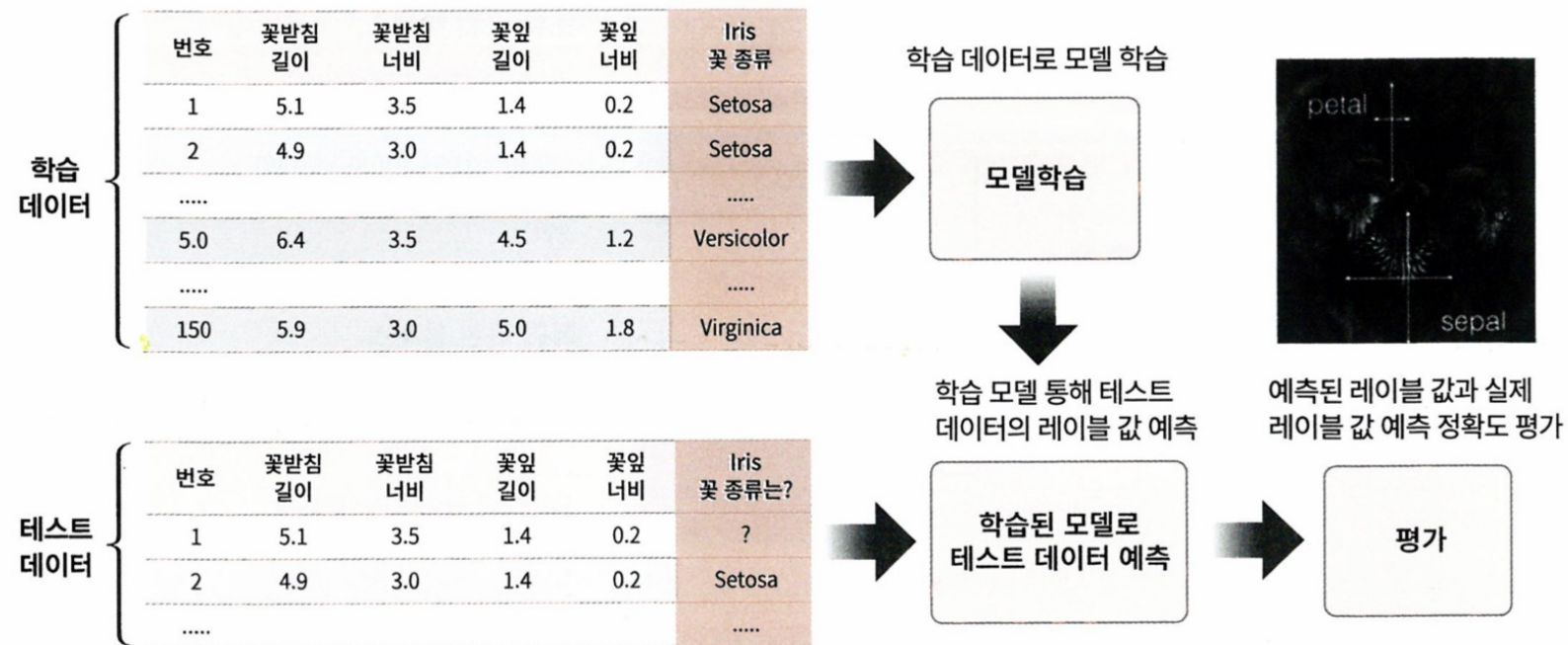
# 머신러닝 프로세스

1. 데이터 세트 분리: `train_test_split()`
2. 모델 학습 : `fit()`
3. 예측 수행 : `predict()`
4. 평가 : `accuracy_score()`

\*예측 성능 평가를 위한 별도 데이터

1. 데이터를 학습용 데이터 와 테스트용 데이터 로 분리
2. 학습 데이터 기반으로 ML 알고리즘 적용해 모델 학습
3. 학습된 ML 모델을 이용해 테스트 데이터 예측
4. 예측 결과값과 테스트 데이터의 실제 결과값 비교해 모델 성능 평가

# 붓꽃 데이터 세트 기반의 머신러닝 분류 예측 수행 프로세스



〈붓꽃 데이터 세트 기반의 ML 분류 예측 수행 프로세스〉



# 03.

## 사이킷런의 기반 프레임워크 익히기

# Estimator 이해 및 fit( ), predict( ) 메서드

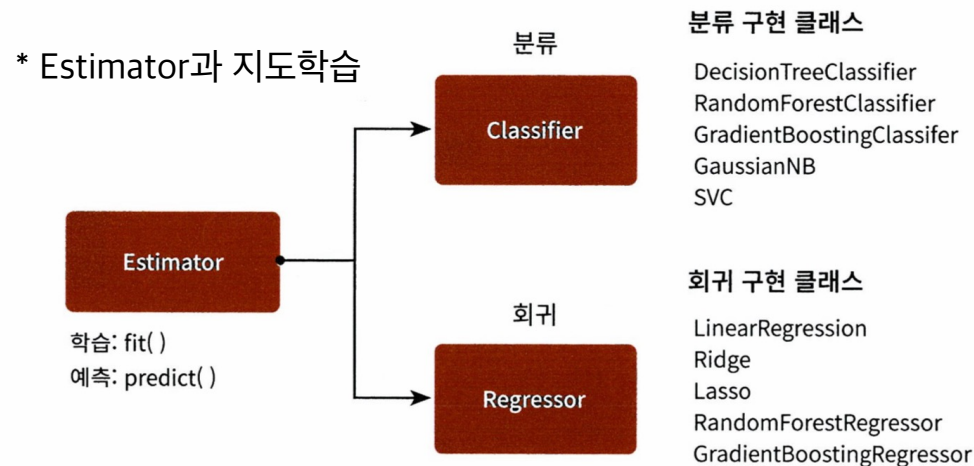
## Estimator

- 데이터셋을 기반으로 머신러닝 결과를 예측하는 class 객체
- 지도학습의 모든 알고리즘을 구현한 클래스

### 1) 지도학습

- 분류(Classification)과 회귀(Regression)로 구성 -> Estimator 클래스
- 모델 학습을 위한 `fit()` 과 모델 예측을 위한 `predict()` 도 Estimator 내부에서 구현
- 평가 함수, 하이퍼 파라미터 튜닝 클래스의 경우 Estimator 를 인자로 받아 Estimator 내의 학습, 예측 함수를 호출해 진행

# Estimator 이해 및 fit( ), predict( ) 메서드



## 2) 비지도학습

- 차원 축소, 클러스터링, 피쳐 추출(Feature Extraction) 등의 클래스
- fit( )과 transform( ) 내부 구현 (하나로 결합한 fit\_transform( ) 제공)
  - fit(): 입력 데이터의 형태에 맞춰 데이터를 변환하기 위한 사전 구조 맞춤
  - transform(): 이후 입력 데이터의 차원 변환, 클러스터링 등 실제 작업 수행



## 사이킷런의 주요 모듈

- 피처 처리

<code>Sklearn.preprocessing</code>	데이터 전처리에 필요한 기능
<code>Sklearn.feature_selection</code>	알고리즘에 큰 영향을 미치는 feature를 우선순위대로 선택 작업 수행
<code>Sklearn.feature_extraction</code>	텍스트 데이터와 이미지 데이터의 벡터화된 feature 추출

## 사이킷런의 주요 모듈

- 피처 처리 & 차원 축소

<code>Sklearn.decomposition</code>	차원 축소 관련된 알고리즘	PCA, NMF, Truncated SVD
------------------------------------	----------------	-------------------------

- 데이터 분리, 검증 & 파라미터 튜닝

<code>Sklearn.model_selection</code>	교차 검증을 위한 학습/테스트 분리, 최적 파라미터 추출	Train_test_split, GridSearchCV
--------------------------------------	------------------------------------	-----------------------------------

## 사이킷런의 주요 모듈

- 평가

<a href="#">Sklearn.metrics</a>	성능 측정 방법 제공	Accuracy, Precision, Recall, ROC-AUC, RMSE 등
---------------------------------	-------------	---

## 사이킷런의 주요 모듈

- ML 알고리즘

<code>Sklearn.ensemble</code>	앙상블 알고리즘 제공	RandomForest, Adaboost, Gradient boosting 등
<code>Sklearn.linear_model</code>	회귀 알고리즘 제공	선형회귀, 릿지, 라쏘, 로지스틱 등
<code>Sklearn.naïve_bayes</code>	나이브 베이즈 알고리즘 제공	가우시안 NB, 다항분포 NB 등

## 사이킷런의 주요 모듈

- ML 알고리즘

<code>Sklearn.neighbors</code>	최근접 이웃 알고리즘 제공	KNN
<code>Sklearn.svm</code>	서포트 벡터 머신 알고리즘 제공	
<code>Sklearn.tree</code>	의사 결정 트리 알고리즘 제공	Decision tree 등
<code>Sklearn.cluster</code>	비지도 클러스터링 알고리즘 제공	K-평균, 계층형, DBSCAN



# 04.

## Model Selection 모듈 소개

# 학습/테스트 데이터 세트 분리

## 1. `train_test_split()`

```
from sklearn.model_selection import train_test_split
```

- 학습/테스트 데이터 세트 분리
- 반환값: train\_X, test\_X, train\_y, test\_y 튜플 형태
- 주요 파라미터

`test_size` , `test_size`

`Shuffle`

`Random_state`

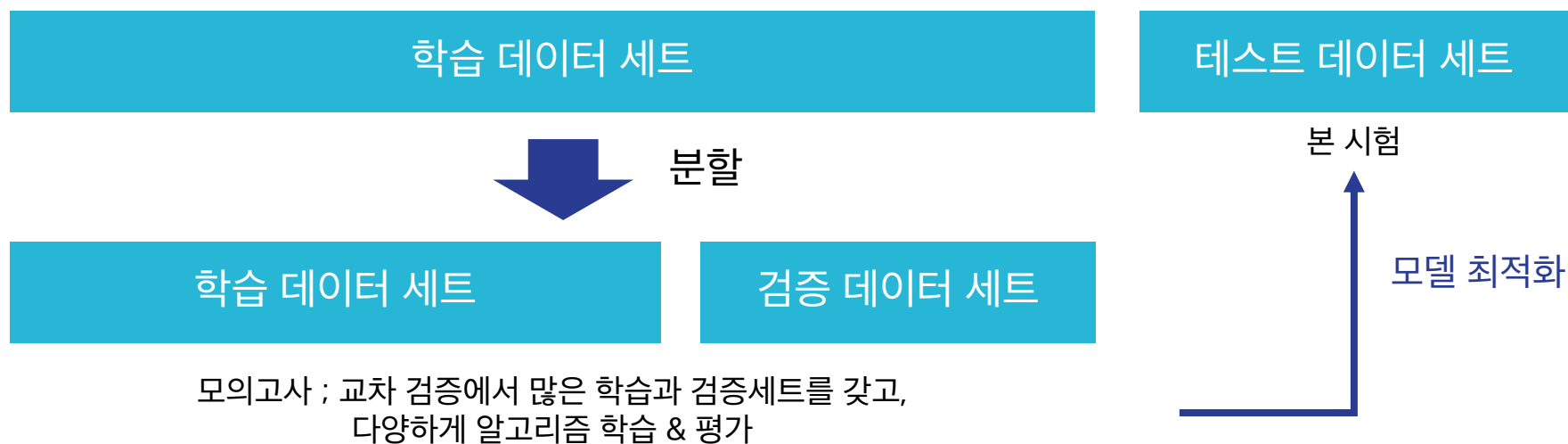
# 교차 검증

- 과적합 방지 \* 모델이 학습 데이터에만 과도하게 최적화되어, 실제 예측을 다른 데이터로 수행하면 성능이 떨어지는 현상

- 데이터 편종을 막기 위해 별도의 여러 세트로 구성된 학습/테스트 세트에서 학습과 평가 수행

*\*쉽게 말해, 모의고사를 여러 번 보고, 본 시험을 보는 것!*

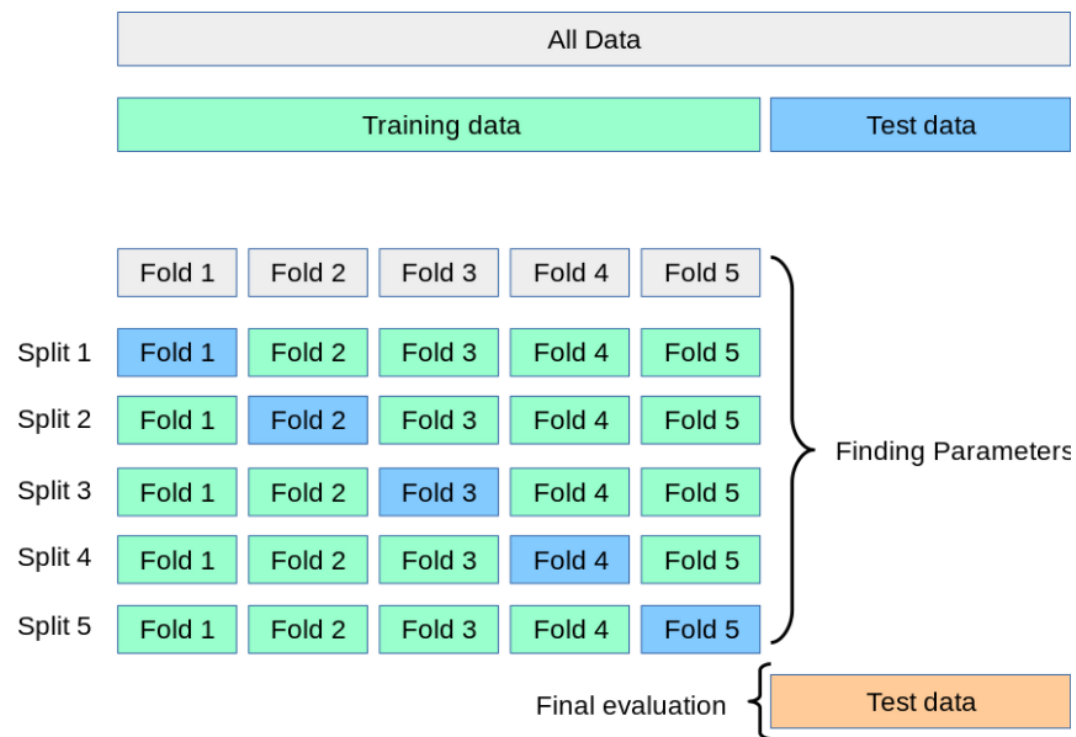
*각 세트(=모의고사)의 결과를 기반으로 하이퍼 파라미터 튜닝 등의 모델 최적화 진행*





# K 폴드 교차 검증

- K개의 데이터 폴드 세트를 만들어 k번 만큼 각 폴드 세트에 학습과 검증 평가를 반복적으로 수행
- 가장 보편적인 교차 검증 기법
  - `From sklearn.model_selection import Kfold`
- `n_splits` 파라미터를 이용해 데이터 세트를 지정
  - `kf = KFold(n_splits = 5)`  
5개의 폴드 세트로 분리하는 kf 객체 생성
  - `kf.split(ar)`  
폴드별 학습&검증용 테스트의 인덱스를 array 형태로 반환



# Stratified K 폴드

\* 특정 값이 많거나 적어, 값의 분포가 치우치고 왜곡된 것

- 불균형한 분포도를 가진 Y 데이터 집합을 위한 k 폴드 방식

\* 가질 수 있는 값이 정수이며, 연속되지 않고 뚝뚝 끊어진 값

- 배열 형태의 레이블(Y)을 가진 데이터 한해 적용 - 분류, 로지스틱 회귀 등

- `From sklearn.model_selection import StratifiedKFold`

- 원본 데이터의 Y 분포를 먼저 고려한 뒤 이 분포와 동일하게 학습/검증 데이터 분배

- `Skfold = StrtifiedKFold(n_splits=3)`

# Stratified K 폴드

Stratified K 폴드 교차 검증 과정

1. 폴드 세트 설정
2. for 루프에서 반복적으로 학습/검증용 인덱스 추출
3. 반복적으로 학습&예측 수행해 예측 성능 반환

How to make it simpler ?

## cross\_val\_score( )

- 교차 검증 과정을 한꺼번에 수행해주는 API
  - `From sklearn.model_selection import cross_val_score, cross_validate`
- 배열 형태의 지정된 성능 지표 측정값 반환
- 주요 파라미터
  - Estimator: Classifier 또는 Regressor
  - X: 피처 데이터 세트 (X에 해당하는 데이터)
  - y: 레이블 데이터 세트 (target 값)
  - scoring: 예측 성능 평가 지표
  - cv: 교차 검증 폴드 수 (default : 5)

## cross\_validate()

- cross\_val\_score() 과 유사하지만, 여러 개의 평가 지표 반환 가능
  - + 학습 데이터 성능 평가 지표와 수행시간도 함께 제공
    - `From sklearn.model_selection import cross_val_score, cross_validate`
- dict 형태의 각 폴드별 test\_score, fit\_time 등 반환
- 주요 파라미터
  - Estimator: Classifier 또는 Regressor
  - X: 피쳐 데이터 세트(X에 해당하는 데이터)
  - y: 레이블 데이터 세트 (target 값)
  - Scoring: 예측 성능 평가 지표 - ['accuracy', 'roc-auc']처럼 리스트 형으로 여러 개 지정 가능
  - Cv: 교차 검증 폴드 수 (default : 3)
  - Return\_train\_score: 훈련 폴드에 대한 점수(train\_score)와 score\_time(default: True)

# GridSearchCV - 교차검증과 최적 하이퍼 파라미터 튜닝을 한 번에

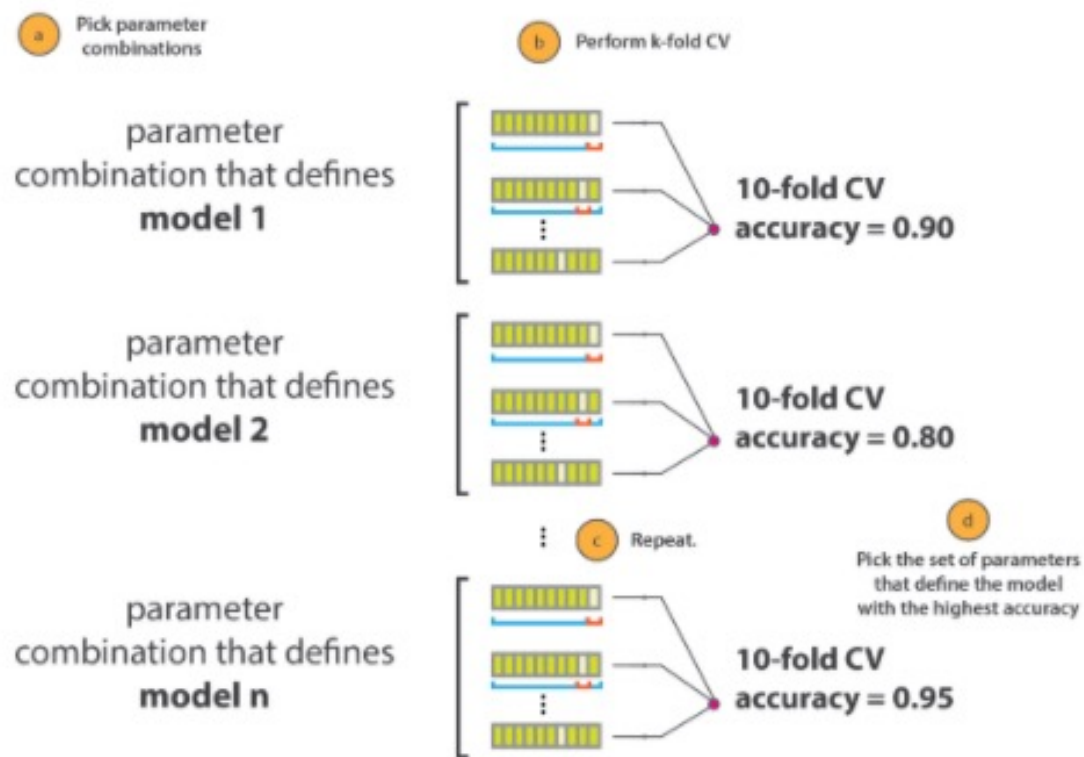
## 1. 기본 정보

- 교차 검증을 기반으로 하이퍼 파라미터의 최적 값을 찾는 api
- 하이퍼 파라미터를 순차적으로 변경하며 최고 성능 파라미터 조합을 찾아준다
- 주요 파라미터
  - Estimator: Classifier 또는 Regressor, pipeline
  - param\_grid: (dict 형태) estimator 튜닝을 위해 파라미터명과 여러 파라미터 값들 지정
  - scoring: 예측 성능 평가 지표
  - cv: 교차 검증을 위해 분할되는 학습/테스트 세트의 개수
  - refit : 가장 최적의 하이퍼 파라미터를 찾은 뒤 estimator 객체에 재학습 (default: True)

# GridSearchCV - 교차검증과 최적 하이퍼 파라미터 튜닝을 한 번에

## 2. 과정

1. 파라미터 조합들 중 하나를 선택해 모델 생성
2. 입력받은 cv 만큼 교차검증해 성능 평가
3. 모든 파라미터 조합들에 대해 이 과정을 반복
4. 가장 성능이 좋은 파라미터 조합을 채택



# GridSearchCV - 교차검증과 최적 하이퍼 파라미터 튜닝을 한 번에

## 3. attributes

<code>cv_results_</code>	(dict 형태) 파라미터별 score	<code>grid.cv_results_</code>
<code>best_estimators_</code>	(estimator) 파라미터가 최적화된 모델	<code>grid.best_estimators_</code>
<code>best_score</code>	(float) 가장 좋은 모델의 성능 평균값	<code>grid.best_score_</code>
<code>best_params_</code>	(dict) 최적의 파라미터	<code>grid.best_params_</code>





# 05.

## 데이터 전처리

## 결손값 (Null 값) 처리

### 1. 피처들 중 Null 값이 적은 경우

해당 피처의 평균값 등으로 대체

이때 피처의 중요도에 따라 대체값 선정에 유의

*\*중요한 피처를 뺏다 평균으로 대체해버리면, 예측 결과가 왜곡됨 !*

### 2. 피처들 중 Null 값 많은 경우

해당 피처를 드롭 하는 게 일반적

# 데이터 인코딩

## 1. Label Encoding

- 문자열 값을 숫자형 카테고리 값으로 변환
  - From `sklearn.preprocessing import LabelEncoder`
- 일괄적 숫자값으로 변환되면서 특정 알고리즘에서는 예측 성능 저하 문제 발생

\* 선형 회귀 등

Species (범주형 변수)	인코더 값
Setosa	1
versicolor	2
virginica	3



- 숫자의 크고 작음에 대한 특성이 작용되기 때문.

왼쪽 예시의 1, 2, 3은 단순한 “카테고리 값”이지  
1이 작은, 3은 큰 중요도를 가지는 것은 아님. 즉, 숫자 그 자체의 의미가 아니다 !

- 트리 계열의 ML 알고리즘에서 사용 추천

\* 선형 회귀 등과는 다르게 숫자의 특성을 반영하지 않는 머신러닝 알고리즘이기 때문.

# 데이터 인코딩

## 2. One-hot Encoding

- 일괄적 숫자 형태로만 변환하는 레이블 인코딩의 문제점을 해결한 인코딩 방식
- 행 형태의 피쳐 고유 값을 열 형태로 차원 변환
  - `from sklearn.preprocessing import OneHotEncoder`
- 고유 값에 해당하는 칼럼에만 1, 나머지는 0 표시

color			
red			
green			
blue			
red			

one-hot  
encoding →

color_red	color_green	color_blue
1	0	0
0	1	0
0	0	1
1	0	0

## 각각 어떤 인코딩 방식을 적용한걸까?

ID	과일
1	사과
2	바나나
3	체리

원-핫 인코딩

\* 고유 값에 해당하는 칼럼에만 1

ID	사과	바나나	체리
1	1	0	0
2	0	1	0
3	0	0	1

레이블 인코딩

\* 일괄 숫자 카테고리형으로 변환

ID	과일
1	0
2	1
3	2

# 데이터 인코딩

사이킷런의 인코더는 문자를 숫자형으로 변환 후, 인코딩

`get_dummies` `()`

- 원 핫 인코딩을 더 쉽게 해주는 판다스 API
- 사이킷런과 달리 숫자형 변환 필요없이 바로 인코딩 가능
- 데이터 프레임을 인수로 받음

# 피처 스케일링

## 1. 표준화

- 각 피처 값을 가우시안 정규 분포 (평균=0, 분산 =1)를 가진 값으로 변환
- 일부 ML 알고리즘이 데이터를 가우시안 분포 형태로 가정하고 구현했기 때문에 표준화를 거치는 것이 예측 성능 향상에 도움이 된다.

## 2. 정규화

- 서로 다른 피처의 크기를 0과 1 사이 값으로 통일 (음수가 있을 경우 -1과 1 사이)
- 피처들의 크기를 통일하기 위해 크기를 변환. 동일한 크기단위로 비교하기 위함

\* 키가 140이고 몸무게가 80일 때,  $140 > 80$ 이라고 해서 몸무게가 적고, 키가 큰 것이 아님.  
두 값을 비교하기 위해선 크기를 맞추어야 함.

# 피처 스케일링

## 1. StandardScaler

- 표준화 지원 클래스
- 특히 SVM, 선형 회귀, 로지스틱 회귀 전 표준화 적용

위의 모델들은 데이터를 가우시안 분포 형태로 가정하고 구현했기 때문에, 사전 표준화가 예측 성능을 향상시킴

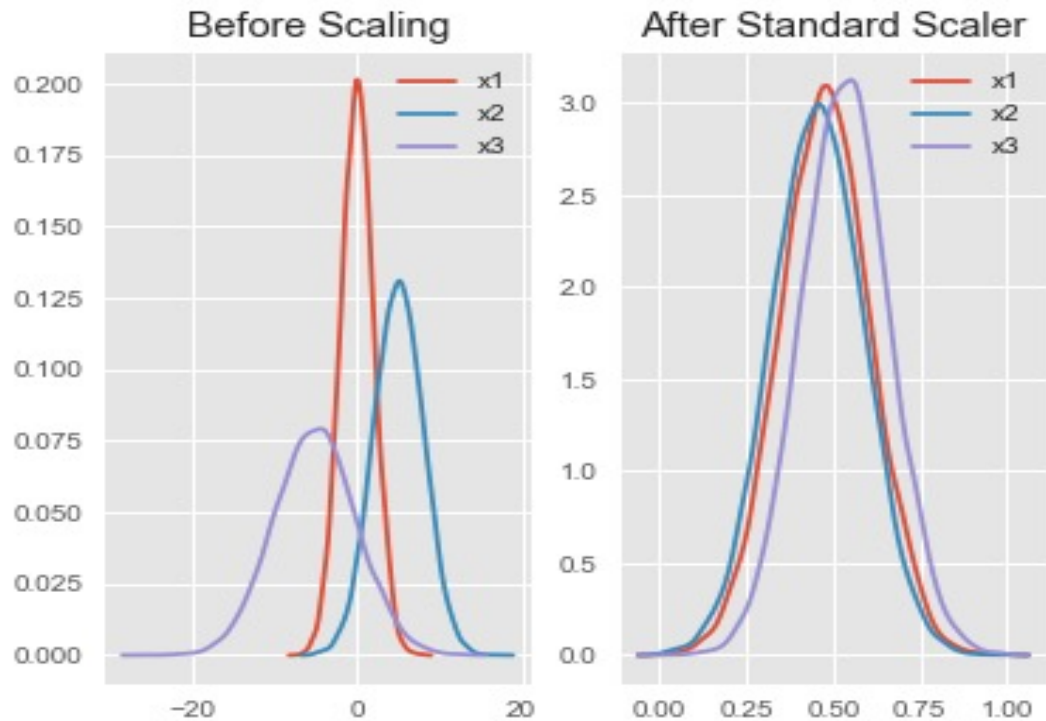
## 2. MinMaxScaler 이용

- 데이터값을 0과 1 사이의 범위로 변환 (음수 존재 시, -1~1)
- 데이터 분포가 가우시안 분포가 아닐 경우 적용해볼 수 있음

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

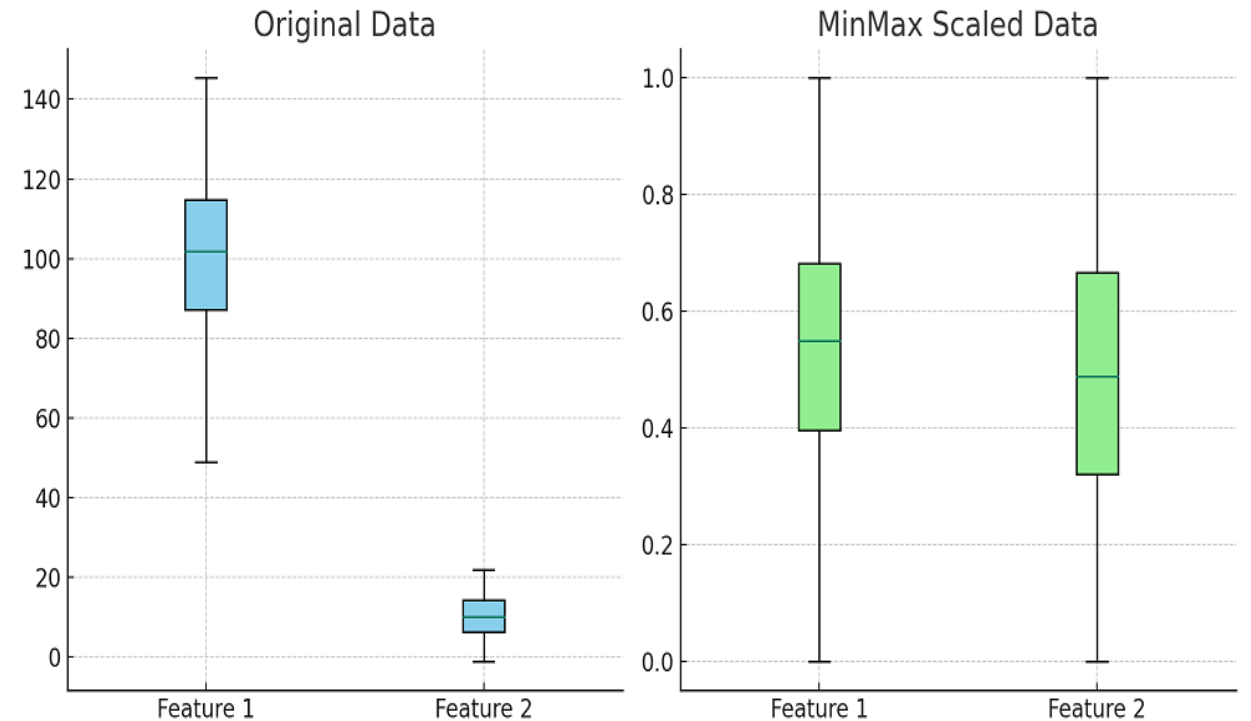


# 피처 스케일링



**StandardScaler**

표준 정규 분포 형태로 스케일링



**MinMaxScaler**

피처 크기가 0 ~ 1 사이로 맞춰짐

# 학습&테스트 데이터의 스케일링 변환 시 유의점

- 데이터 스케일링 시 사용하는 메서드
  - fit() : 데이터 변환을 위한 최대 최솟값 등의 기준 정보 설정을 적용
  - transform() : 설정된 정보로 데이터를 변환
  - fit\_transform() : 한 번에 적용

이 때, 학습 데이터와 테스트 데이터 각각에 fit()을 적용한다면, 오류가 발생한다.

가령, 두 데이터 집단이 각각의 fit()에 따른 서로 다른 기준 정보를 가져서, 스케일링의 기준이 달라지므로,

학습 데이터에서는 10이 1로, 테스트 데이터에서는 5가 1로 스케일링 되는 등의 오류 발생 !

```
# MinMaxScaler() scaler 객체 생성.  
scaler = MinMaxScaler()  
# 학습 데이터에 대해서 fit(), transform() 수행.  
scaler.fit(X_train)  
scaled_X_train = scaler.transform(X_train)
```

\* 애초에 학습과 테스트로 데이터를 분리하기 전,  
전체 데이터 세트에 스케일링을 적용하는 것이 더욱 바람직하다

```
# 테스트 데이터에서는 다시 fit()이나 fit_transform()을 수행하지 않고 transform만 수행.  
scaled_X_test = scaler.transform(X_test)
```

# 머신러닝 데이터 분석 프로세스 정리

## 0. 문제 정의 및 데이터 수집

### 1. 데이터 탐색

데이터가 어떻게 생겼는지, 어떤 특성을 가졌는지, 그래서 어떤 모델을 쓸 지 고민하는 과정

### 2. 데이터 전처리

Null 값 처리, scaling, 인코딩 등 => sklearn.preprocessing 모듈

### 3. 학습/테스트 데이터 세트 분리

=> train\_test\_split()

### 4. 모델 학습 (학습 데이터 이용)

데이터 특성별 모델 선택

교차 검증 => sklearn.model\_selection 모듈

### 5. 모델 성능 평가 (검증 데이터 이용)

평가 및 최적화 => sklearn.model\_selection 모듈



수고하셨습니다