

머신러닝 데이터 전처리 과정 (Total 180분)

6강. 숫자의 언어로 번역하기

: 스케일링과 인코딩

머신러닝 알고리즘이 이해할 수 있도록
'문자'를 숫자로 바꾸고, '숫자의 체급'을 맞추는 핵심 전처리 단계



1교시 (50분) 인코딩 (Encoding)

컴퓨터가 이해할 수 없는 '문자열'을
수학적 연산이 가능한 '숫자'로 변환



2교시 (50분) 스케일링 (Scaling)

단위가 다른 변수들을 공정한 기준선 상에
놓아 모델의 학습 효율 극대화



3교시 (60분) Leakage & 실습

Data Leakage(정보 누수) 방지 원칙과
주택 가격 예측 파이프라인 구축 실습

학습 목표 & 로드맵

오늘 우리는 데이터가 머신러닝 모델에 들어
가기 전, 반드시 거쳐야 할 '번역'과 '체급 조
절' 과정을 마스터합니다.

Why Encoding?

머신러닝 모델은 수학 공식입니다. 문자를 숫자로 바꾸고, 숫자 크기의 중요성을 이해합니다.

How to Transform

Encoding: Label vs One-Hot

Scaling: Standard vs MinMax vs Robust

Critical Warning

가장 빈번한 실수 '**Data Leakage**'

Test 데이터에는 절대 fit() 하지 않는 원칙 학습.

Real-world Practice

주택 가격 데이터(Housing Price)로 전처리부터 예측까지
완전한 파이프라인을 직접 구축합니다.

① 왜 문자를 숫자로 바꿔야 할까?

머신러닝 모델(회귀식, 신경망)은 본질적으로 수학 공식입니다.

$$\text{"빨강"} \text{ String} + \text{"파랑"} \text{ String} = ? \text{ Error}$$

◆ 범주형 데이터(Categorical Data)의 두 얼굴

명목형 (Nominal)

순서나 랭킹이 없음

혈액형 MBTI 도시



순서형 (Ordinal)

명확한 순서가 존재함

학점(A/B/C) 사이즈(S/M/L)

SCIKIT-LEARN CODE PREVIEW

encoding_example.py

```
from sklearn.preprocessing import OneHotEncoder

# 인코더 객체 생성
enc = OneHotEncoder(
    handle_unknown='ignore',
    sparse_output=False
)

# 데이터를 학습하고 변환 (fit & transform)
X_encoded = enc.fit_transform(X_train)

print(X_encoded)
# Output: array([[1.0, 0.0], [0.0, 1.0], ...])
```

💡 컴퓨터는 이제 "빨강"을 [1, 0]과 같은 숫자 벡터로 인식하여 계산할 수 있게 됩니다.



범주형 데이터 유형 정리

Nominal vs Ordinal: 데이터의 성격에 따라 처리 방법이 달라집니다.

KEY CONCEPT

명목형

순서가 없는 카테고리

Nominal

MATHEMATICAL RELATION

$$A \neq B \neq C$$

"서로 다르다는 것 외에 비교 불가능"

EXAMPLES



혈액형

A형, B형, O형, AB형



지역 / 도시

서울, 부산, 뉴욕, 런던



성별

남성, 여성

순서형

명확한 순서나 등급 존재

Ordinal

MATHEMATICAL RELATION

$$S > M > L$$

"크기나 중요도의 대소 관계 성립"

EXAMPLES



학점 / 등급

A+, B0, C, F (성적순)



의류 사이즈

XS < S < M < L < XL



만족도

매우 불만족 ~ 매우 만족



RECOMMENDED ENCODING

One-Hot Encoding



RECOMMENDED ENCODING

Label / Ordinal Encoding

↳ 개념: 줄 세우기 (Mapping)

고유한 문자열 값(Category)을 알파벳 순서나 등장 순서에 따라 **0부터 시작하는 정수로** 변환합니다.



장점 (Pros)



- ✓ 메모리 사용량이 적음
- ✓ 구현이 매우 간단함
- ✓ 타겟 변수(y) 변환에 유용

치명적 단점 (Cons)



"순서의 왜곡"
모델이 숫자의 크기($0 < 1 < 2$)를 데이터의 중요도나 순위로 오해할 수 있음.

SCIKIT-LEARN CODE

label_encoding.py

```

from sklearn.preprocessing import LabelEncoder

# 1. 데이터 준비 (문자열)
cities = ["파리", "도쿄", "파리", "뉴욕"]

# 2. 인코더 생성
le = LabelEncoder()

# 3. 변환 (알파벳/가나다 순으로 번호 부여)
encoded = le.fit_transform(cities)

print(encoded)
# Output: [2, 0, 2, 1]
# 도쿄(0), 뉴욕(1), 파리(2)
# "ㄴ" < "ㄷ" < "ㅍ" 순서 반영
  
```



트리 모델에서는 괜찮아요!

랜덤 포레스트나 XGBoost 같은 트리 기반 모델은 데이터의 대소 관계보다는 분기(Split)를 중요시하므로, Label Encoding을 사용해도 성능 저하가 적습니다.



WARNING

CRITICAL PITFALL

"숫자의 크기를
순서로 오해합니다"

모델은 단순해서 0보다 1이 크고, 1보다 2가 크다
고 생각합니다. 의도치 않은 랭킹(Rank)이 만들어
집니다.

Label Encoding의 치명적 단점

명목형(Nominal) 데이터에 적용할 때 발생하는 문제

BAD CASE

Raw Data

서울

Encoded: 0

0

Raw Data

부산

Encoded: 1

1

Raw Data

대구

Encoded: 2

2



Model's View: 서울(0) < 부산(1) < 대구(2)

가짜 순서 발생!

✓ 해결책 (Solution)

순서가 없는 명목형 데이터는
One-Hot Encoding을 사용하세요.

ⓘ 예외 (Exception)

트리 모델(Tree-based)은
대소 관계 영향이 적어 Label Encoding 가능.



● 오직 하나만 Hot(1), 나머지는 Cold(0)

해당 카테고리에만 1을 부여하고, 나머지에는 0을 부여하는 벡터로 변환합니다.

Apple

[1 , 0 , 0]

Banana

[0 , 1 , 0]

Cherry

[0 , 0 , 1]

▣ 장점과 단점 (Pros & Cons)

장점 (Good)



- ✓ 순서 왜곡이 없음 (공정한 거리 계산)
- ✓ 명목형 변수(도시, 성별 등)에 최적

단점 (Bad)



- ✗ 차원의 저주 (컬럼 수가 급격히 증가)
- ✗ 메모리 사용량 증가

PANDAS VS SCIKIT-LEARN

one_hot_encoding.py

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# 1. Pandas: 직관적이고 빠름 (데이터 분석용)
# drop_first=True로 다중공선성(Dummy Trap) 방지
df_encoded = pd.get_dummies(
    df,
    columns=['city'],
    drop_first=True
)
```

2. Scikit-learn: 파이프라인 연동 (머신러닝용)

```
enc = OneHotEncoder(
    handle_unknown='ignore'
)
df_enc = enc.fit_transform(df[['city']])
```

⚠ Dummy Trap 주의: 카테고리가 N개일 때, N-1개의 컬럼만 있어도 정보 표현이 가능합니다. (예: 서울, 부산, 대구 중 서울=0, 부산=0이면 자동 대구)



Label Encoding

범주를 0부터 시작하는 정수로 매핑

Integer

DATA TRANSFORMATION

서울



0

부산



1

대구



2

VS

⚠ Rank Problem

$0 < 1 < 2$

(서울 < 부산 < 대구?)

✓ 메모리 효율적 (1개 컬럼 유지)

✗ 숫자의 크기를 중요도로 오해



BEST FOR
트리 모델 (XGBoost, RF)

One-Hot Encoding

해당 값만 1(Hot), 나머지는 0(Cold)

Binary Vector

DATA TRANSFORMATION

입력

서울

부산

대구

서울	1	0	0
부산	0	1	0
대구	0	0	1

✓ Solution

모든 항목 간 거리 동일

(순서 왜곡 없음)

✓ 순서(Rank) 문제 완벽 해결

✗ 차원의 저주 (컬럼 수 급증)



BEST FOR
회귀, 신경망, 거리기반 모델



Pop Quiz

데이터 특성에 맞는 인코딩 방법을 찾아보세요.

Check Your Knowledge

Q1 티셔츠 사이즈



Q2 거주 지역



DATA PREVIEW

Small Medium Large

"S, M, L 사이즈는 서로 순서(Order)가 있나요?"

✓ 정답: Ordinal Encoding

S < M < L 순서가 중요하므로, 0, 1, 2와 같이 크기가 있는 숫자로 변환합니다.

DATA PREVIEW

서울 경기 부산

"서울이 부산보다 더 큰 값인가요?"

✓ 정답: One-Hot Encoding

순서가 없는 명목형 데이터이므로, 순서 왜곡을 막기 위해 0과 1의 벡터로 변환합니다.



Bonus Tip

타겟 변수(Target, y)가 문자열('Cat', 'Dog')이라면?

Use `LabelEncoder()` → 타겟은 모델 학습에 직접적인 대조 대상이 아님

SECTION 02

⌚ 50 min

데이터의 체급 맞추기

서로 다른 단위(Unit)와 범위(Range)를 가진 변수들을
공정한 기준선 상에 놓는 과정입니다.

KEY METHODS

StandardScaler

MinMaxScaler

RobustScaler



▲ 단위가 다르면 모델이 착각합니다

거리 기반 알고리즘(KNN, K-Means)은 값이 큰 변수가 학습을 지배해버립니다.

압도적 영향력

VS

연봉

단위: 50,000,000원

나이

단위: 30세

☰ 스케일링이 필수적인 모델들



거리 기반

KNN, K-Means, SVM



최적화 기반

선형회귀, 로지스틱, 신경망(Neural Net)

STANDARD SCALER USAGE

scaling_pattern.py

✓ Best Practice

```
from sklearn.preprocessing import StandardScaler

# 1. 스케일러 객체 생성
scaler = StandardScaler()

# 2. Train 데이터: 기준 학습(fit) 후 변환
X_train_scaled = scaler.fit_transform(X_train)

# 3. Test 데이터: 학습된 기준으로 변환만(transform)
# 주의: 절대 fit()을 다시 호출하지 않음!
X_test_scaled = scaler.transform(X_test)
```

⚠ Fit vs Transform:

Fit은 데이터의 분포(평균, 분산)를 배우는 과정이고, Transform은 실제로 값을 바꾸는 과정입니다.

어떻게 변환하는가?

공식 (Z-Score):

$$z = \frac{x - \mu}{\sigma}$$

- x : 원본 데이터
- μ : 평균 (Mean)
- σ : 표준편차 (Std Dev)

특징 및 활용

언제 쓰는가?



- 정규분포를 가정하는 모델
- 선형회귀, 로지스틱 회귀
- SVM (Support Vector Machine)
- 경사하강법 최적화 속도 향상

주의할 점



- 이상치(Outlier)에 민감함
- 이상치가 평균과 분산을 왜곡시킴
- 데이터가 정규분포가 아니어도 쓸 수 있지만, 효과가 떨어질 수 있음

SCIKIT-LEARN CODE

```
from sklearn.preprocessing import StandardScaler

# 1. 스케일러 객체 생성
std = StandardScaler()

# 2. Train 데이터: 학습(fit) + 변환(transform)
# 평균과 표준편차를 계산하고 저장함
X_train_scaled = std.fit_transform(X_train)

# 3. Test 데이터: 변환(transform)만!
# Train에서 구한 평균/분산을 그대로 적용
X_test_scaled = std.transform(X_test)

print(X_train_scaled.mean())
# Output: 0.0 (근사값)
print(X_train_scaled.std())
# Output: 1.0
```

핵심 포인트

데이터의 중심이 0으로 이동하고, 퍼진 정도가 1로 맞춰집니다. 서로 단위가 다른(예: 키와 몸무게) 변수들을 공정하게 비교할 수 있게 됩니다.



MinMaxScaler: 0~1 정규화

데이터의 분포 모양은 유지하면서 크기만 줄이기

Normalization

※ 가장 작은 값을 0, 가장 큰 값을 1로

데이터의 원래 분포(모양)를 그대로 유지하면서, 모든 값을 0과 1 사이의 공간으로 압축합니다.

$$x_{\text{new}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

❖ 특징 및 주의사항

Best for...

이미지 데이터 & 딥러닝

픽셀값(0~255)을 0~1로 변환하거나, 활성화 함수 범위에 맞출 때 필수적입니다.



Warning!



이상치(Outlier)에 매우 취약

엄청 큰 이상치가 있으면, 나머지 정상 데이터들이 0 근처 한 점으로 찌그러집니다.

SCIKIT-LEARN CODE

minmax_scaling.py

```
from sklearn.preprocessing import MinMaxScaler

# 0 ~ 1 사이로 압축하는 스케일러 생성
mm = MinMaxScaler()

# Train 데이터로 최대/최소 학습 후 변환
X_train_scaled = mm.fit_transform(X_train)

# Test 데이터는 학습된 기준(Train의 min/max) 적용
X_test_scaled = mm.transform(X_test)

print(X_train_scaled.min(), X_train_scaled.max())
# Output: 0.0 1.0
```

💡 정보의 손실 없이 **비율**만 줄이고 싶을 때 사용하세요. 단, 이상치가 있다면 먼저 제거하거나 RobustScaler를 고려해야 합니다.



⚡ 왜 RobustScaler인가?

평균(Mean)과 분산(Variance)은 **이상치**에 매우 민감합니다. 엄청 큰 값이 하나만 있어도 전체 기준이 흔들립니다.

대신 **중앙값(Median)**과 **사분위수(IQR)**를 사용해 중심을 잡습니다.

MATHEMATICAL FORMULA

$$\text{New } x = \frac{x - \text{Median (Q2)}}{\text{IQR (Q}_3 - \text{Q}_1)}$$

* IQR (Interquartile Range): 데이터의 하위 25% ~ 상위 25% 사이의 범위

★ 핵심 특징



이상치 저항성

극단적인 값(Outlier)이 있어도 나머지 데이터들이 뭉개지지 않음



중앙값 중심

데이터의 중앙값이 0이 되도록 이동하지만, 범위가 1로 고정되진 않음

SCIKIT-LEARN CODE

robust_scaling.py

```
from sklearn.preprocessing import RobustScaler

# 1. 스케일러 객체 생성
rb = RobustScaler()

# 2. 학습 및 변환 (Median, IQR 계산)
X_robust = rb.fit_transform(X_train)

# 테스트 데이터는 transform만!
X_test_robust = rb.transform(X_test)

print(X_robust.mean())
# 평균은 0이 아닐 수 있습니다 (중앙값이 0)
```



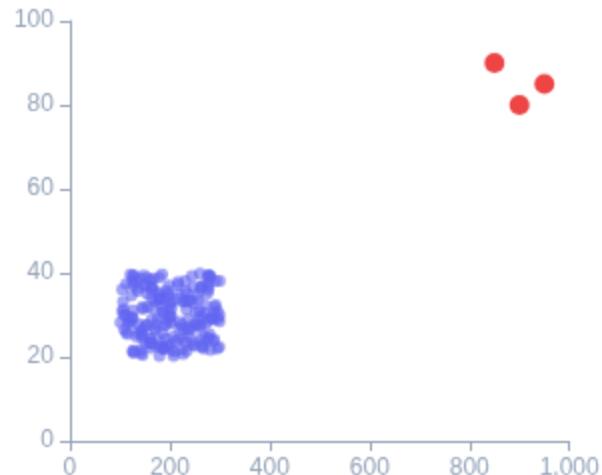
언제 사용할까요?

데이터에 이상치(Outlier)가 많다고 의심될 때, StandardScaler보다 먼저 시도해보세요.

이상치(Outlier)가 포함된 데이터에 스케일링을 적용했을 때 분포의 변화를 비교해봅시다.

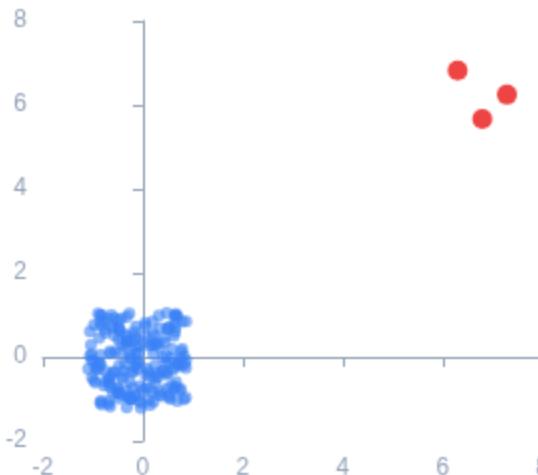
1. Original Data

Raw



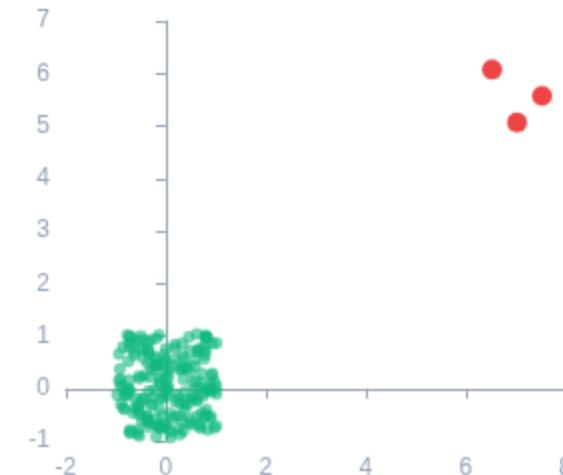
2. StandardScaler

Z-Score



3. RobustScaler

IQR



CHARACTERISTICS

데이터 단위가 서로 다름
(X: 0~1000, Y: 0~50)
이상치가 존재하여 분포가 치우침

StandardScaler()

평균 0, 분산 1로 이동
이상치 때문에 정상 데이터가
좁은 구간에 압축되어 보임

RobustScaler()

중앙값 0, IQR 기준 스케일링
이상치는 멀리 그대로 두고
중앙 분포의 모양을 잘 보존함

● Normal Data

● Outlier (이상치)

💡 이상치가 많다면 RobustScaler가 더 효과적일 수 있습니다.

스케일러 선택 핵심 가이드

데이터의 특성(이상치 유무)과 사용하는 모델 알고리즘에 따라 최적의 스케일러는 달라집니다.

상황별 맞춤 전략을 한 눈에 정리해봅시다.



이상치(Outlier)가 많다면?

RobustScaler 추천

평균과 분산 대신 중앙값(Median)과 사분위수(IQR)를 사용하여 이상치의 영향을 최소화합니다.



일반적인 경우 (회귀, SVM)

StandardScaler 추천

데이터가 정규분포를 따른다고 가정하는 선형회귀, 로지스틱 회귀, SVM 모델에 필수입니다.



이미지 / 딥러닝 데이터

MinMaxScaler 추천

데이터의 분포 모양을 유지하면서 0~1 사이로 값을 압축합니다. 신경망 입력에 주로 사용됩니다.



트리 기반 모델 (Random Forest, XGB)

스케일링 불필요

트리 모델은 데이터의 절대적인 값보다는 대소 관계(Rank)를 보고 분기하므로 스케일링 영향이 적습니다.

SECTION 03

Data Leakage 방지 & 종합 실습

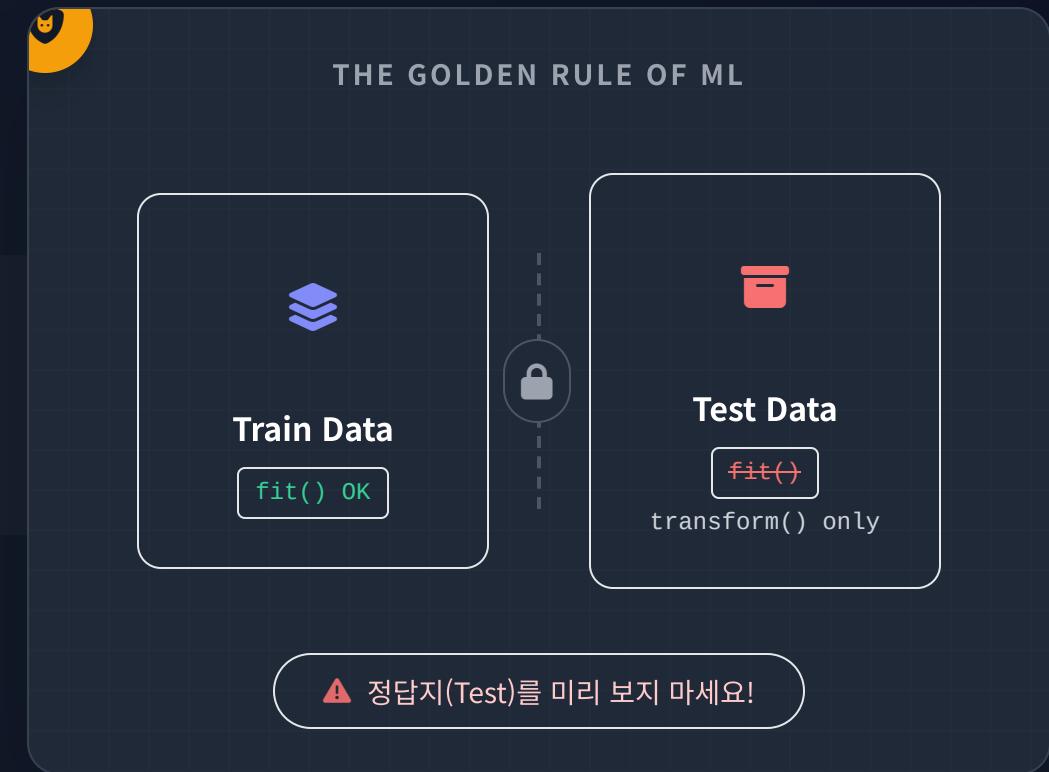
전처리 과정에서 미래의 정보(Test Data)가 모델에 유출되는 것을 막는 철칙을 배웁니다.

KEY CONCEPTS

Fit vs Transform

Scikit-learn Pipeline

Info Leakage





WARNING

DATA LEAKAGE ALERT

"정답지를 미리
훔쳐보지 마세요!"

테스트 데이터(Test)의 정보가 학습 과정에 섞이면,
수능 시험지를 미리 보고 공부하는 것과 같습니다.

Fit vs Transform: 킬러 원칙

스케일링 적용 시 가장 빈번하게 발생하는 '정보 누수' 실수

BAD CASE (실수)

Test Data에 Fit을 하면?

```
scaler.fit_transform(X_test)
```

결과 (Consequence)

✖ Test 데이터만의 평균과 분산이 새로 계산되어 모델에 반영됨.

⚠ 현상: 평가 점수(Score)는 엄청 높게 나오지만, 실제 서비스(Production)에서는 성능이 폭락합니다. (Overfitting)

BEST PRACTICE (정석)

1 Train Data

```
scaler.fit(X_train)  
scaler.transform(X_train)
```

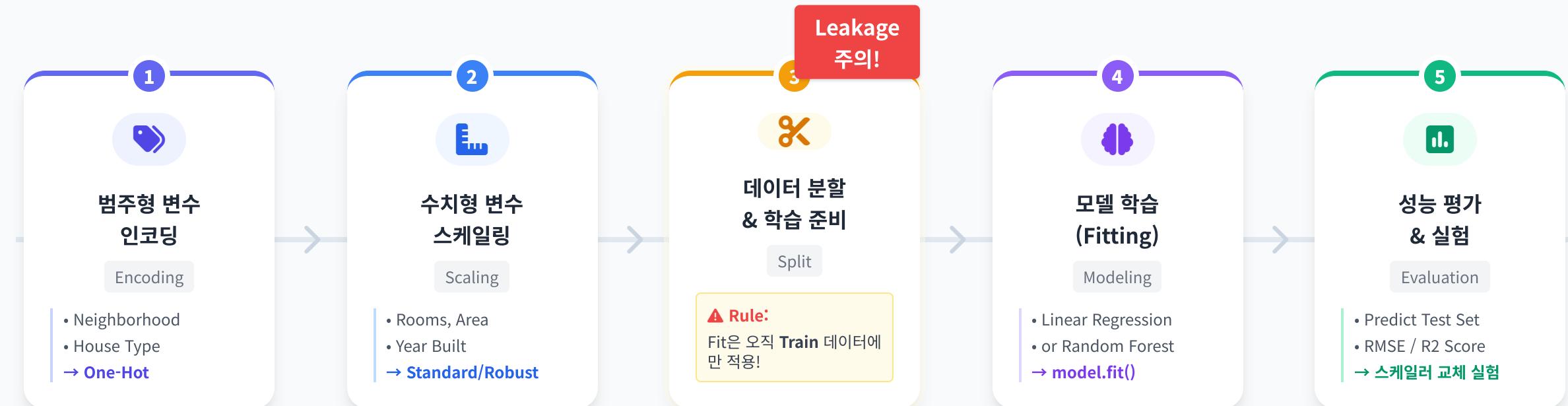
✓ 기준(평균, 분산) 마련

2 Test Data

```
scaler.transform(X_test)  
// 절대 fit 금지!
```

✓ Train 기준을 그대로 적용

💡 Pipeline을 사용하면 이 실수를 자동으로 방지할 수 있습니다.



</>

```
pipe = Pipeline([
    ('preprocessor', ColumnTransformer([ ... ])),
    ('model', LinearRegression())
])
```

Next: View Code →



Final Practice Code

Scikit-learn Pipeline을 활용한 완벽한 전처리

Python 3.x



```
1  from sklearn.compose import ColumnTransformer
2  from sklearn.pipeline import Pipeline
3  from sklearn.preprocessing import StandardScaler, OneHotEncoder
4
5  # 1. 컬럼 정의
6  cat_cols = ['Neighborhood', 'Type']
7  num_cols = ['Rooms', 'Area', 'YearBuilt']
8
9  # 2. 전처리기 정의 (Transformer)
10 processor = ColumnTransformer([
11     ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols),
12     ('num', StandardScaler(), num_cols)
13 ])
14
15 # 3. 파이프라인 구축 (전처리 + 모델)
16 pipe = Pipeline([
17     ('prep', processor),
18     ('model', LinearRegression())
19 ])
20
21 # 4. 학습 (Train에만 fit!)
pipe.fit(X_train, y_train)
```

01 ColumnTransformer

범주형 변수와 수치형 변수를 나누어서 처리합니다.

병렬적(Parallel)으로 각각 인코딩과 스케일링을 수행하고 결과를 합칩니다.

02 Pipeline의 마법

전처리기(`prep`)와 모델(`model`)을 하나로 묶습니다.

데이터가 파이프를 통과하듯 **순차적(Sequential)**으로 흐릅니다.

03 Leakage 방지 효과

`pipe.fit(X_train)` 만 호출하면 끝!

`predict(X_test)` 할 때, 파이프라인이 알아서 Train의 정보(평균, 분산)로 Test를 변환합니다. (실수 방지)

EXPERIMENT TIP

이상치가 많다면? 위 코드의 `StandardScaler()` 를
`RobustScaler()` 로 한 단어만 바꿔보세요!