Korea Aerospace University

2AI4203: Machine Learning

Lec08: Machine Learning in Practice

Jiyoon Shin

KAU College of AI Convergence

Oct. 28, 2026

Korea Aerospace University

Part 1. Data Preprocessing

전처리

Real-World Data

- Real-world datasets are often incomplete, inconsistent, or noisy

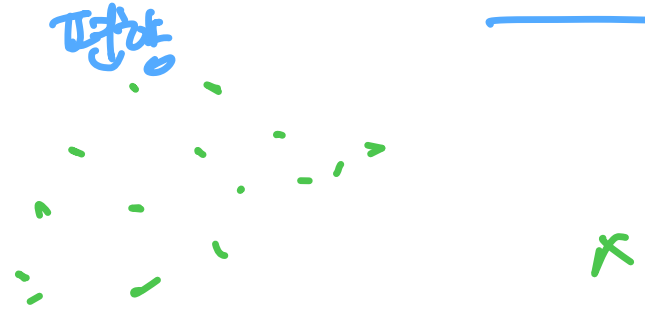
- Common data challenges
  - Missing values: unavailable feature values
  - Categorical features: non-numerical categories
  - Outliers: unusually large or small observations
  - Skewed distributions: asymmetric feature distributions

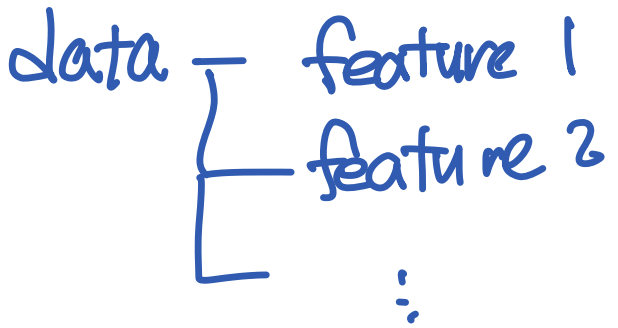
Categorical feature

Missing values

ID	Age	Income ($)	City
1	25	45,000	Seoul
2	32	?	Busan
3	?	60,000	Seoul
4	29	500,000	Incheon
5	41	52,000	Busan

Potential outlier





Missing Values

- Missing values occur when feature values are unavailable or not recorded
  - Causes: sensor failures, incomplete records, unanswered questions
- Deletion
  - Remove incomplete samples or features
  - May discard useful information
- Simple imputation
  - Numerical: mean or median
  - Categorical: most frequent value
- KNN Imputation
  - Estimate missing values from similar samples

K Means
: clustering



KNN (K-nearest neighbors)

Mean vs. Median Imputation

- Mean Imputation
  - Replaces missing values with the mean of observed values
  - Sensitive to extreme values
- Median Imputation
  - Replaces missing values with the median of observed values
  - Less sensitive to outliers
- Example: income (in $1,000s)

$$[30, 35, 40, 45, 300, ?]$$

$$\text{Mean} = \frac{30 + 35 + 40 + 45 + 300}{5} = 90$$

$$\text{Median} = 40$$

2.

Categorical Features

- Categorical features represent distinct groups or categories



- Nominal Features
  - Categories without an inherent order
  - Examples: city, color, blood type
- Ordinal Features
  - Categories with a meaningful order
  - Examples: low / medium / high
- Many ML algorithms require numerical encoding of categorical features

Categorical Encoding 인코딩

- Ordinal Encoding
  - Maps categories to integer values
  - Suitable for features with meaningful ordering

dog → 0 0 1
cat → 0 1 0
human → 1 0 0

Low → 0, Medium → 1, High → 2

- One-Hot Encoding
  - Creates a binary feature for each category
  - Avoids imposing an artificial ordering
  - May increase feature dimensionality

Color
Red
Blue
Green

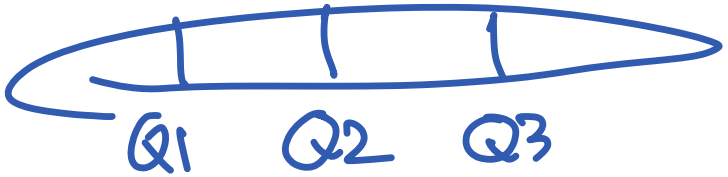


Color	Red	Blue	Green
Red	1	0	0
Blue	0	1	0
Green	0	0	1

- Integer encoding may impose artificial ordering on nominal features

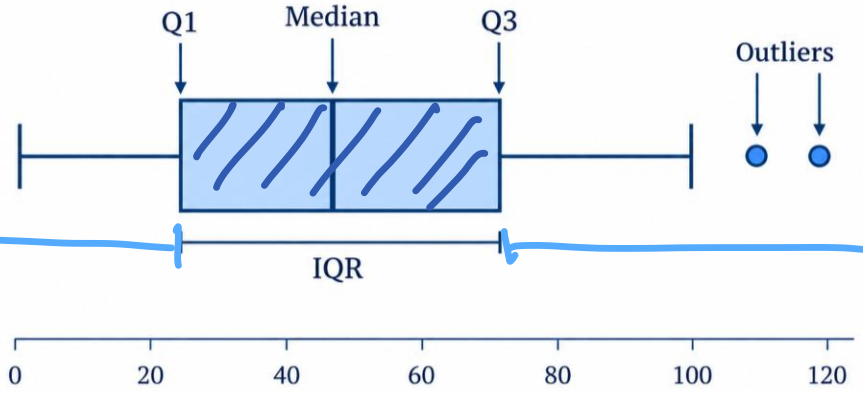
Outlier Detection

- An outlier differs substantially from other observations
- Possible causes
  - Measurement or data-entry errors
  - Rare but valid observations



- Interquartile Range (IQR)
  - Measures the spread of the middle 50% of the data

$$\text{IQR} = Q_3 - Q_1$$

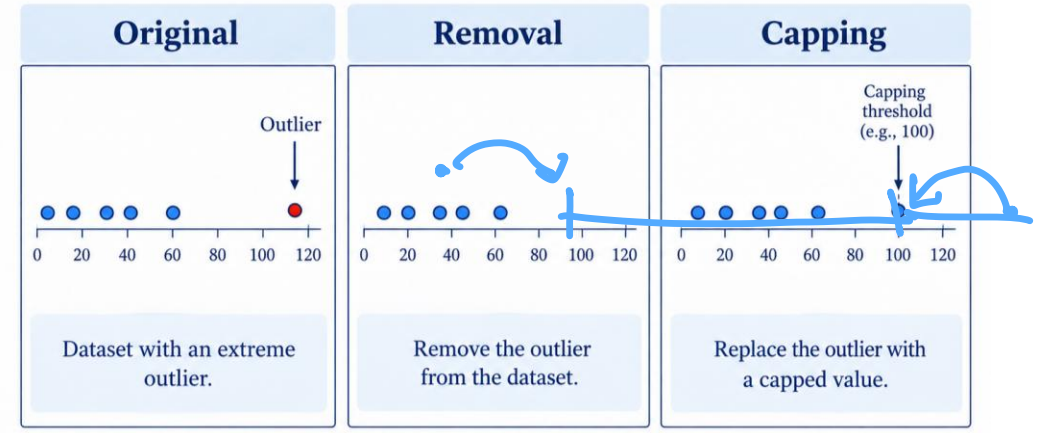


- IQR-based outlier rule

$$x < Q_1 - 1.5\text{IQR} \quad \text{or} \quad x > Q_3 + 1.5\text{IQR}$$

Outlier Handling

- Removal
  - Remove erroneous or unsuitable observations
  - May discard valid information
- Capping (Winsorization)
  - Replace extreme values with predefined limits
- Retaining Outliers
  - Keep rare but meaningful observations



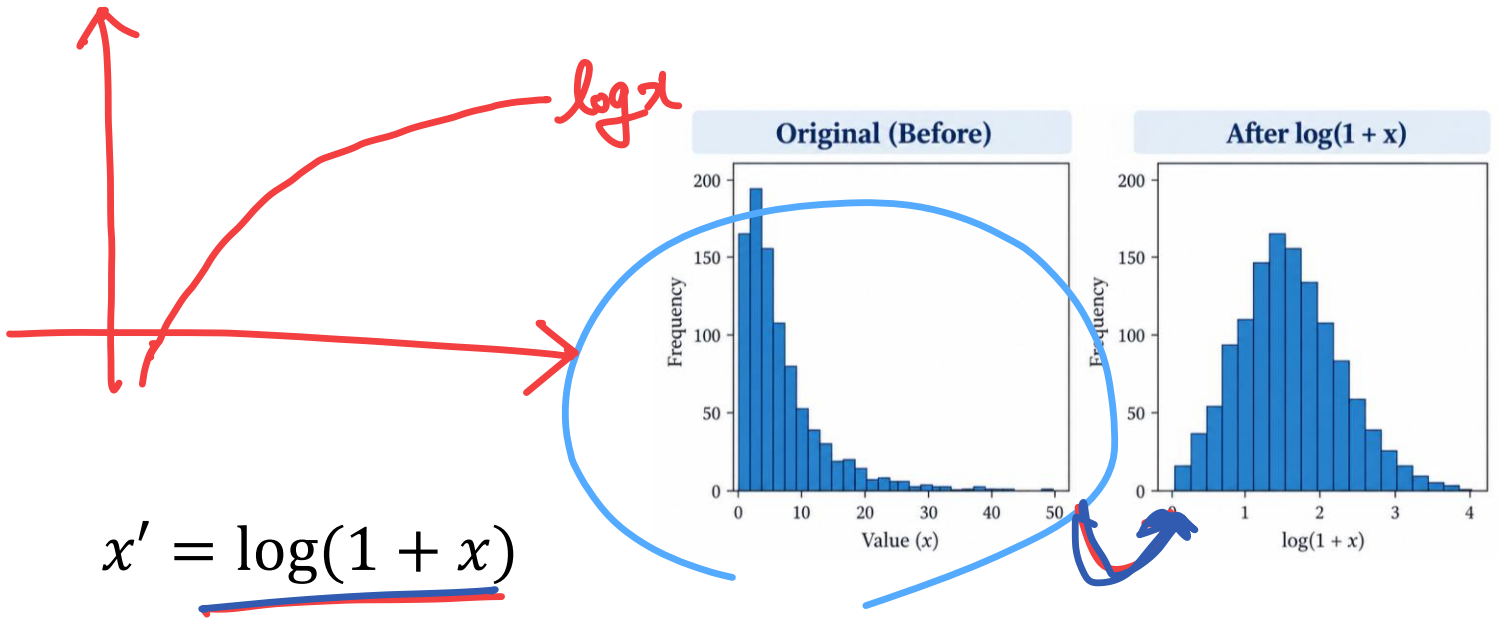
- The appropriate strategy depends on the data and application

4. Skewed dis

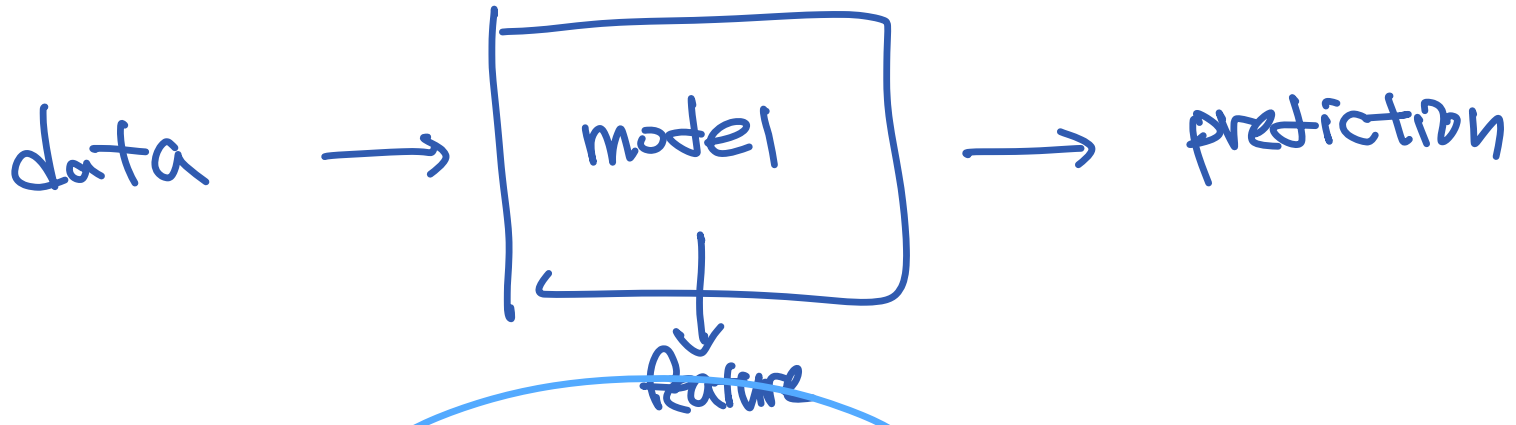
Log Transformation

- Numerical features may have highly skewed distributions
  - Examples: income, transaction amounts, house prices
- Log Transformation
  - Compresses large values
  - Can reduce right-skewness
- For non-negative features:

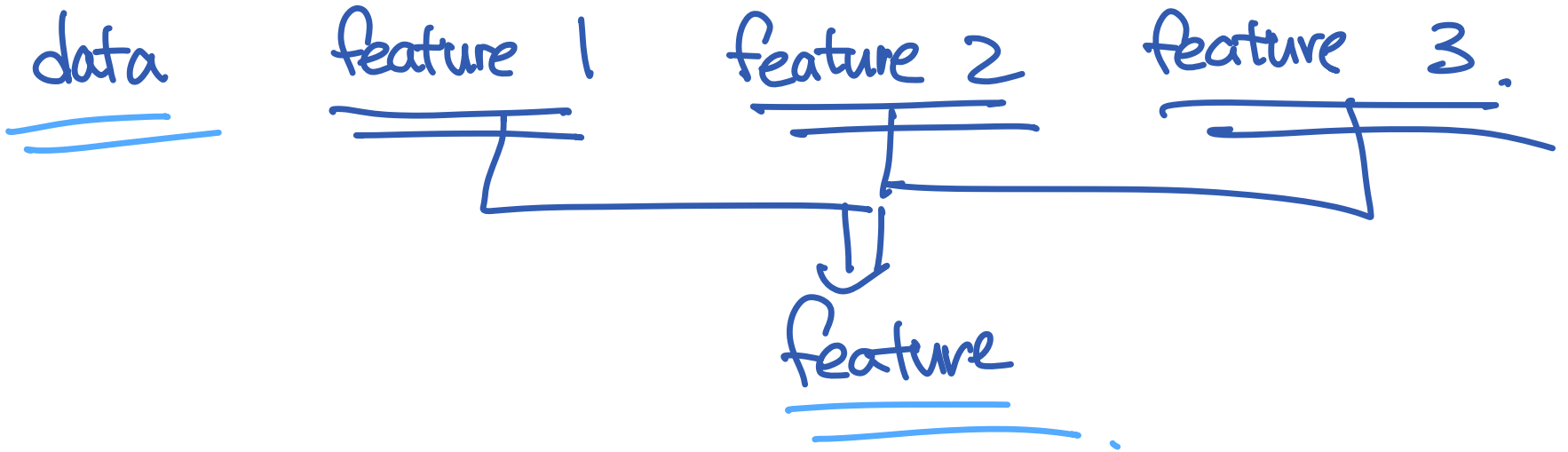
$$x' = \log(1 + x)$$



- Transformations do not always improve model performance ?



Part 2. Feature Engineering



Feature Engineering

- Feature engineering creates or selects useful features for machine learning

- Two main approaches
  - Feature construction: create new features from existing data
  - Feature selection: retain relevant features
- Goals
  - Improve predictive performance
  - Reduce irrelevant or redundant information

- Feature engineering often requires domain knowledge

Feature Construction

- Feature construction creates new features from existing data

- Common approaches
  1. Ratios: express relative relationships
  2. Aggregations: summarize multiple observations
  3. Date-based features: extract information from dates

Feature Construction

- Examples

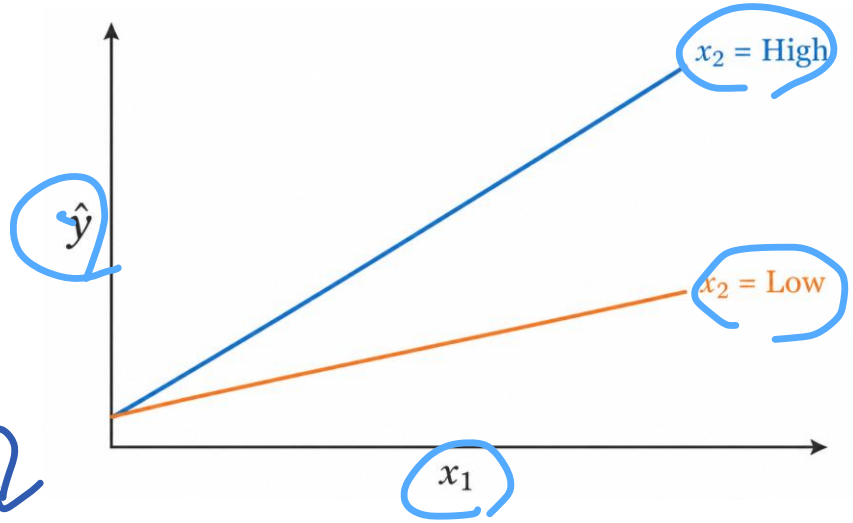
Raw Features	Constructed Feature
Debt, Income	Debt-to-Income Ratio
Total Spending, Number of Orders	Average Purchase Amount
Transaction Date	Day of Week
Birth Date, Reference Date	Age

Feature Interactions

- Feature interactions capture relationships between multiple input features
  - The effect of one feature may depend on another feature

- Interaction Features

$$x_{\text{interaction}} = x_1 x_2$$



- Example: linear model with an interaction term

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2$$

  - Interaction terms allow one feature's effect to depend on another
  - They enable linear models to capture more complex relationships

2.

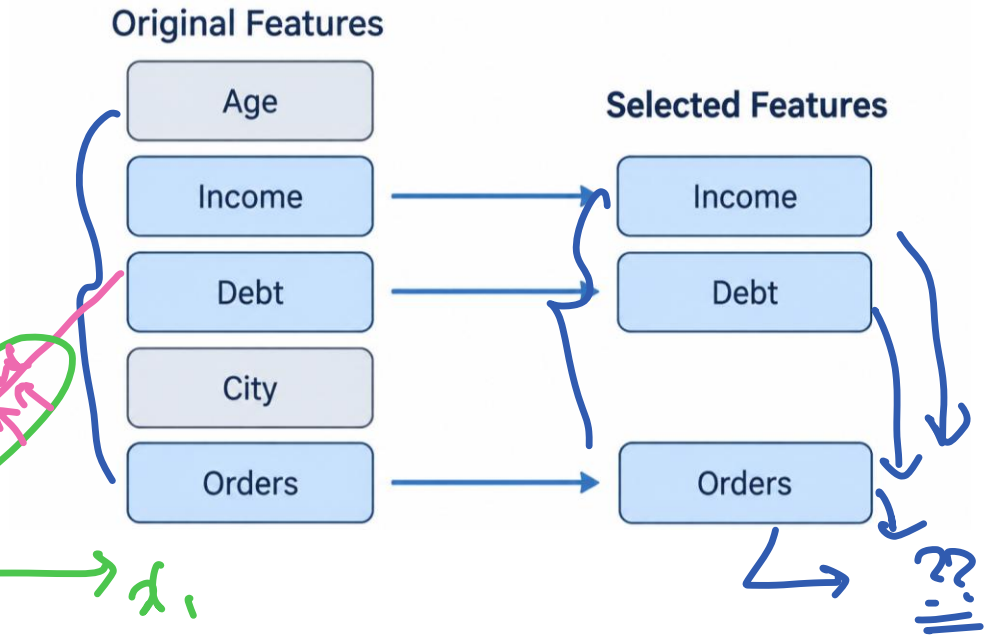
Feature Selection

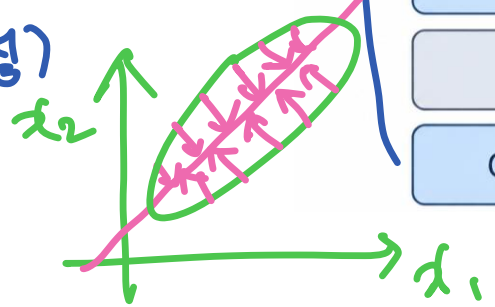
data → model → output?

- Feature selection retains a subset of relevant input features

"black-box model"

- Why select features?
  - Remove irrelevant or redundant features
  - Reduce computational cost
  - Improve interpretability (설명 가능성)
  - Potentially reduce overfitting

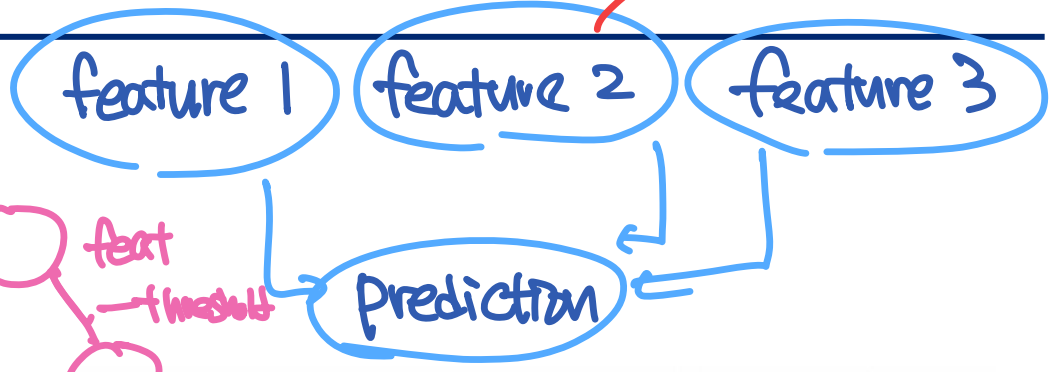


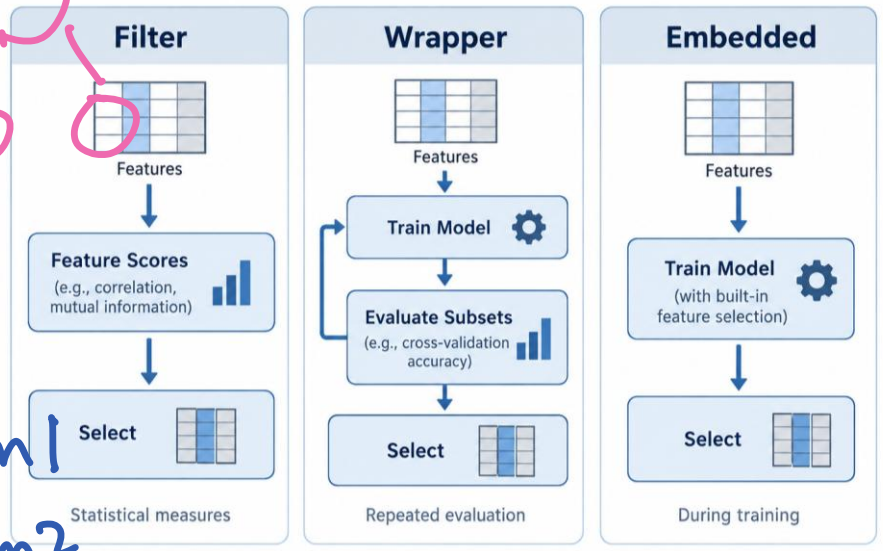


- Unlike feature extraction (e.g., PCA), feature selection retains original features

Feature Selection Methods

- Filter Methods
  - Select features using statistical measures
  - Examples: correlation, mutual information
- Wrapper Methods
  - Evaluate feature subsets using a predictive model
  - Example: recursive feature elimination (RFE)
- Embedded Methods
  - Select features during model training
  - Examples: L1 regularization, tree-based importance





Korea Aerospace University

Part 3. Data Leakage

Data Leakage

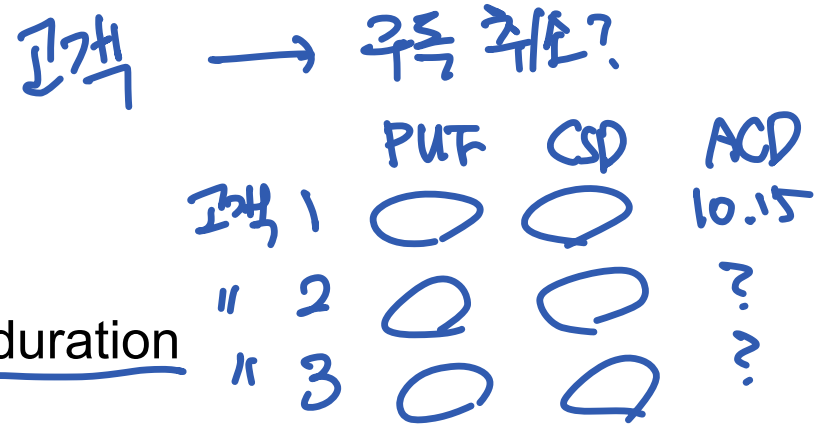
- Data leakage occurs when unavailable information influences model training or evaluation

- Common types
  1. Target leakage: using information unavailable at prediction time
  2. Data contamination: using evaluation data during model development
- Consequences
  - Overly optimistic evaluation results
  - Poor generalization to new data
  - Unreliable model comparisons

train/val/test split
↳ data leakage

Target Leakage

- Target leakage uses information unavailable at prediction time
- Example: customer churn prediction
  - Target: whether a customer will cancel a subscription
  - Valid features: past usage frequency, current subscription duration
  - Leaky feature: actual cancellation date
- Key question: would this feature be available when the prediction is made?

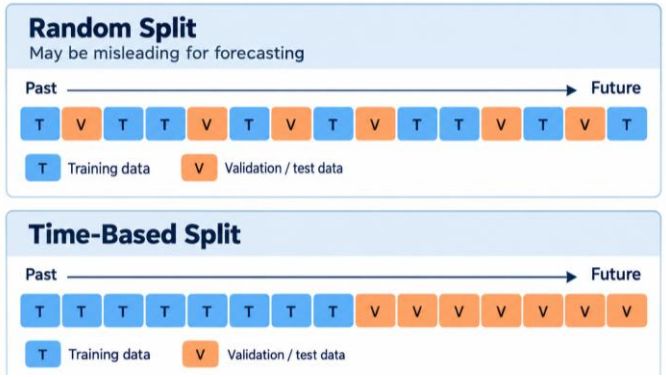


Feature Selection Leakage

- Feature selection causes leakage when evaluation data influences feature selection
- Incorrect Approach
  - Select features using the entire dataset and target labels
  - Split data into training and test sets afterward
- Correct Approach
  - Split data before feature selection
  - Select features using training data only
  - Apply the selected features to evaluation data
- Perform feature selection within each training fold during cross-validation

Temporal Leakage

- Temporal leakage involves using future information for predictions or evaluation

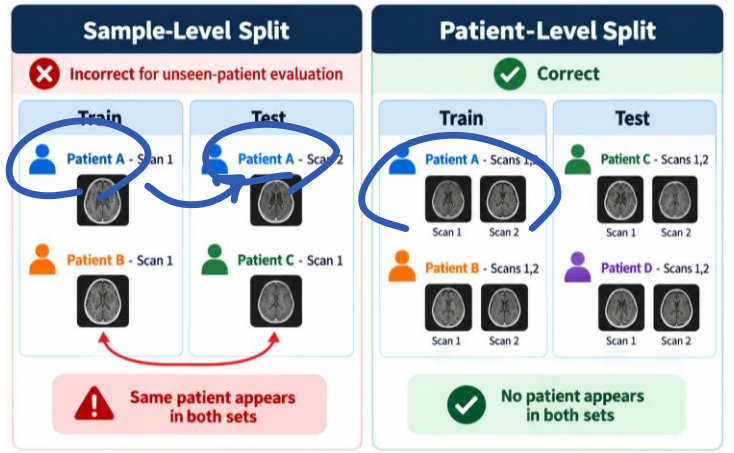


- Example: sales forecasting
  - Target: next week's sales
  - Valid features: historical sales, previous week's average
  - Leaky features: future sales, statistics computed using future observations
- Time-Based Split
  - Train on past observations
  - Evaluate on future observations
- Random splitting may produce misleading results for forecasting tasks

Group Leakage

- Group leakage occurs when related samples overlap across training and evaluation

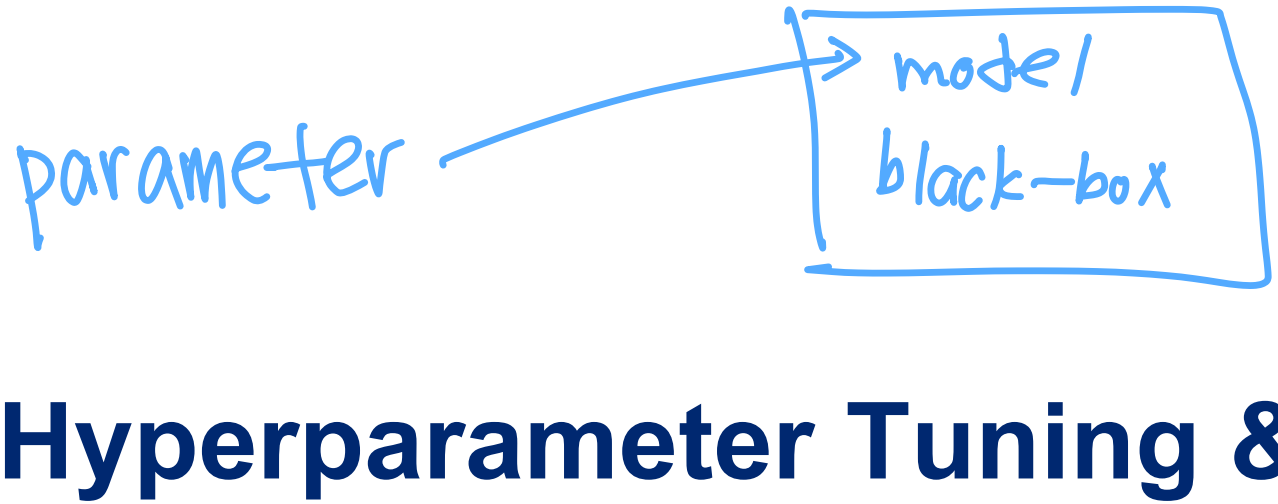
- Example: medical image classification
  - Multiple MRI scans may come from the same patient
  - Random splitting may place the same patient in both sets
  - The model may exploit patient-specific characteristics



MRI pat 1 → scan 1, scan 2, ...
pat 2 → "
pat 3 → "

- Group-Based Split
  - Keep all samples from the same group in one split
  - Example: patients, users, recording sessions

- Group-aware evaluation helps assess generalization to unseen groups



Part 4. Hyperparameter Tuning & Pipelines

Grid Search

- Grid Search evaluates every combination of predefined hyperparameter values
- Example: RBF-SVM

$$C \in \{0.1, 1, 10\}, \quad \gamma \in \{0.01, 0.1, 1\}$$

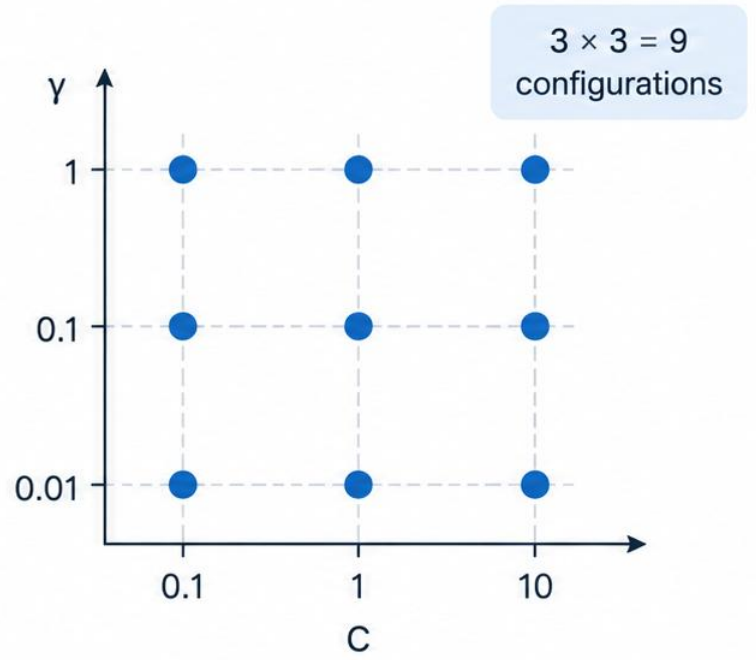
- Number of configurations

$$N = 3 \times 3 = 9$$

- With 5-fold cross-validation:

$$N_{\text{CVfits}} = 9 \times 5 = 45$$

- Advantages
  - Simple and exhaustive within the predefined grid
- Limitation
  - Computational cost grows rapidly with search space size

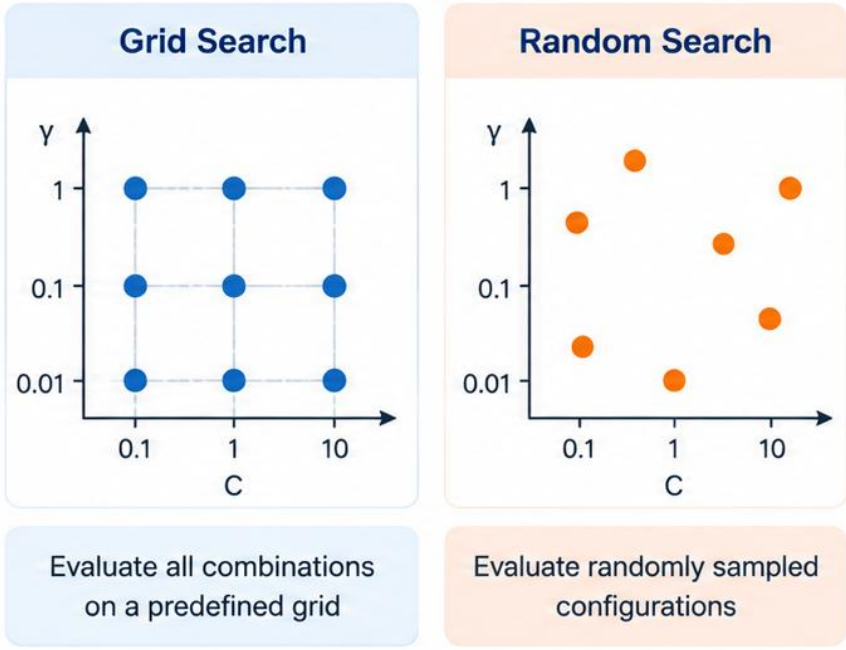


Random Search

- Random Search evaluates randomly sampled hyperparameter configurations
- Procedure
  - Define search distributions
  - Sample candidate configurations
  - Evaluate using cross-validation
- Advantages
  - Flexible computational budget
  - Efficient exploration of large search spaces
- Limitation
  - May miss promising configurations
  - Results depend on random sampling

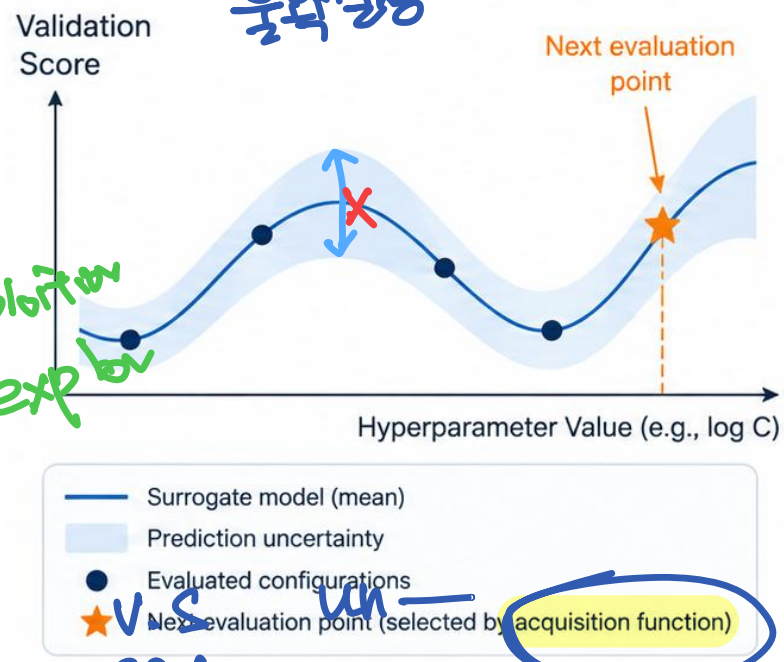
$0.1 \le C \le 10$

$0.01 \le \gamma \le 1$



Bayesian Optimization

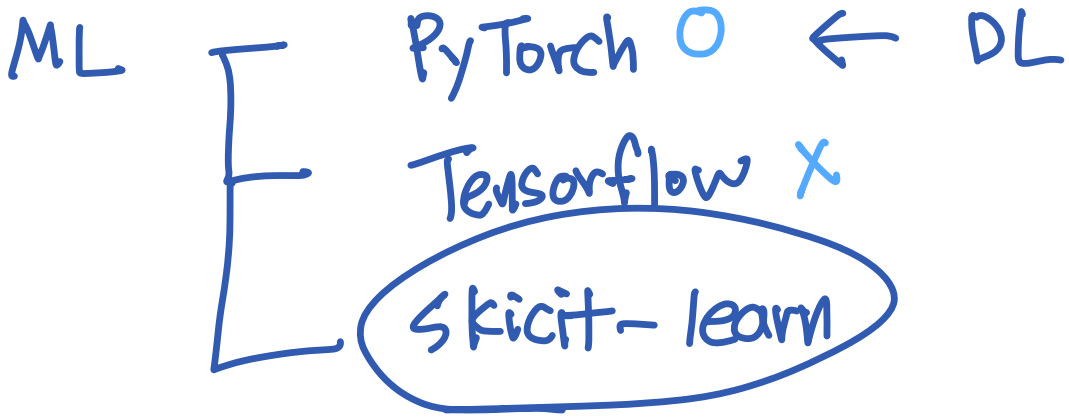
- Bayesian Optimization uses previous evaluation results to select promising hyperparameters
- Key idea
  - Build a surrogate model from previous evaluations
  - Select promising configurations
  - Evaluate and update the surrogate model
- Exploration vs. Exploitation
  - Exploration: investigate uncertain regions
  - Exploitation: focus on promising regions
- May reduce expensive evaluations



GridSearchCV

- `GridSearchCV` automates hyperparameter tuning with cross-validation

- Main arguments
  - `estimator`: model to optimize
  - `param_grid`: candidate values
  - `cv`: cross-validation strategy
  - `scoring`: evaluation metric



- Selects the best configuration based on cross-validation performance

GridSearchCV

- Code Example

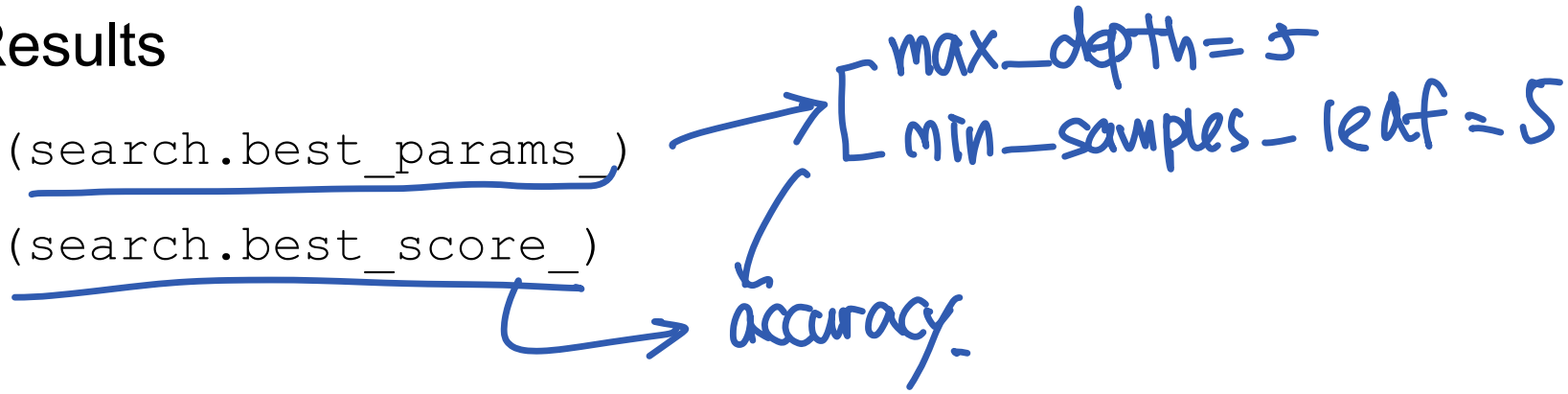
```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier


param_grid = {
    "max_depth": [3, 5, 10],
    "min_samples_leaf": [1, 5, 10]
}
search = GridSearchCV(
    DecisionTreeClassifier(random_state=42),
    param_grid, cv=5, scoring="accuracy"
)
search.fit(X_train, y_train)
```

GridSearchCV

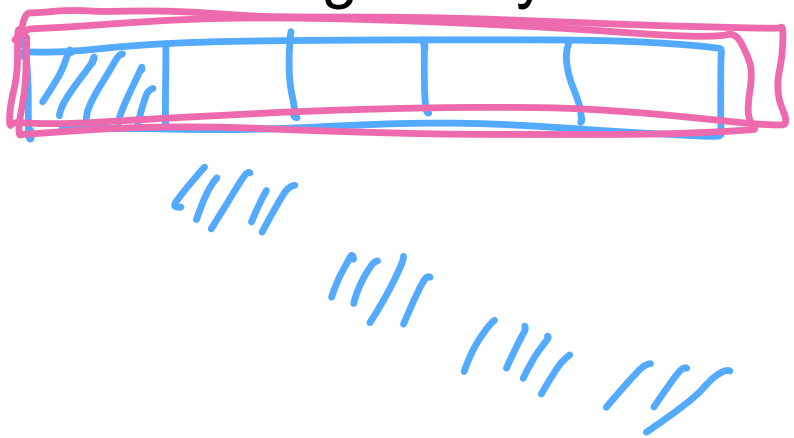
- Search Results

```
print(search.best_params_)
print(search.best_score_)
```



- `best_params_`: best hyperparameter configuration
- `best_score_`: best mean cross-validation score
- `best_estimator_`: best model refitted on the full training set by default

search.best_estimator_

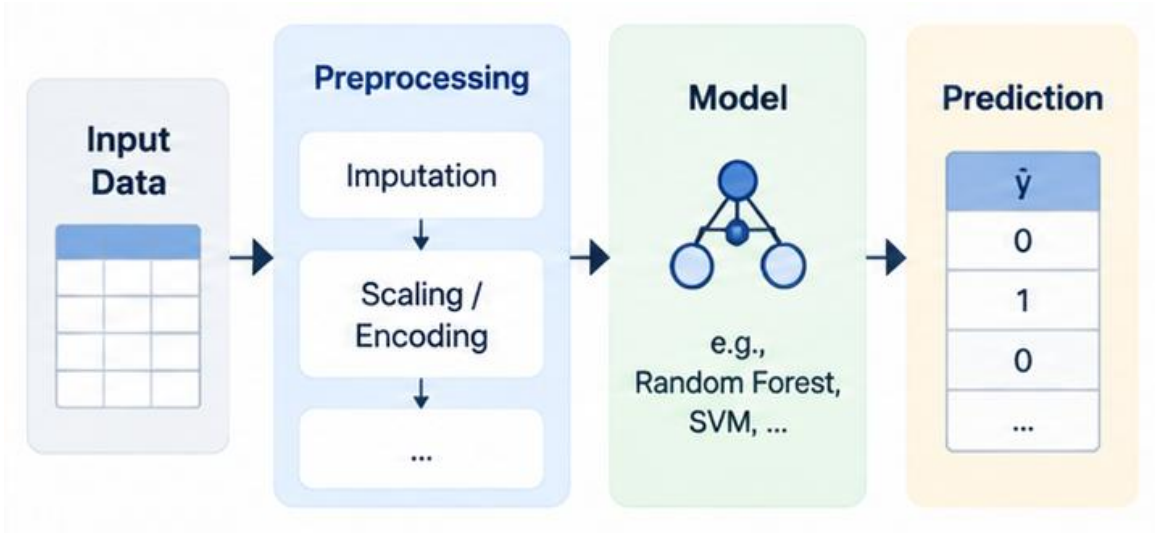


ML Pipelines

- A pipeline combines preprocessing and model training into a single workflow

- Typical steps
  - Imputation → Scaling / Encoding → Model

- Benefits
  - Consistent preprocessing
  - Reduced repetitive code
  - Lower risk of preprocessing leakage
  - Easier tuning and evaluation



ML Pipelines

- Code Example

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC


pipe = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler()),
    ("model", SVC(kernel="rbf"))
])
pipe.fit(X_train_num, y_train)
y_pred = pipe.predict(X_test_num)
```

SVM (support vector machine)

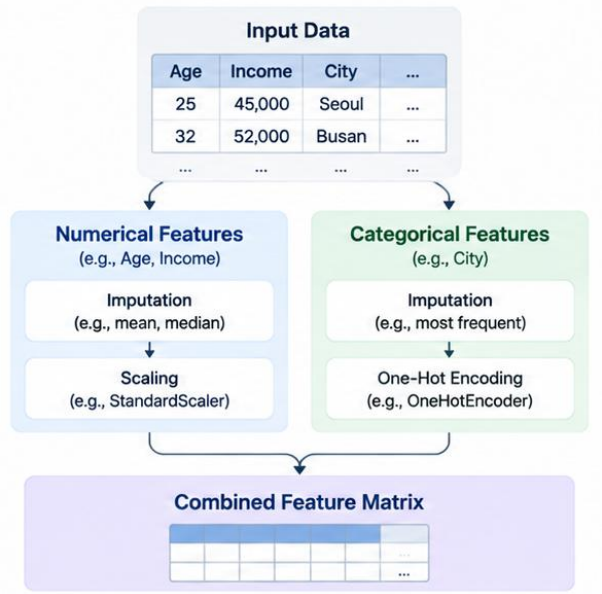
SVC (~ classif —)

SVR (~ regression —)

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

ColumnTransformer

- Real-world datasets often contain multiple feature types



- Different feature types require different preprocessing
  - Numerical: imputation and scaling
  - Categorical: imputation and encoding

- `ColumnTransformer` applies different transformations to specified columns
- Transformed features are combined into a single feature matrix

ColumnTransformer

- Code Example

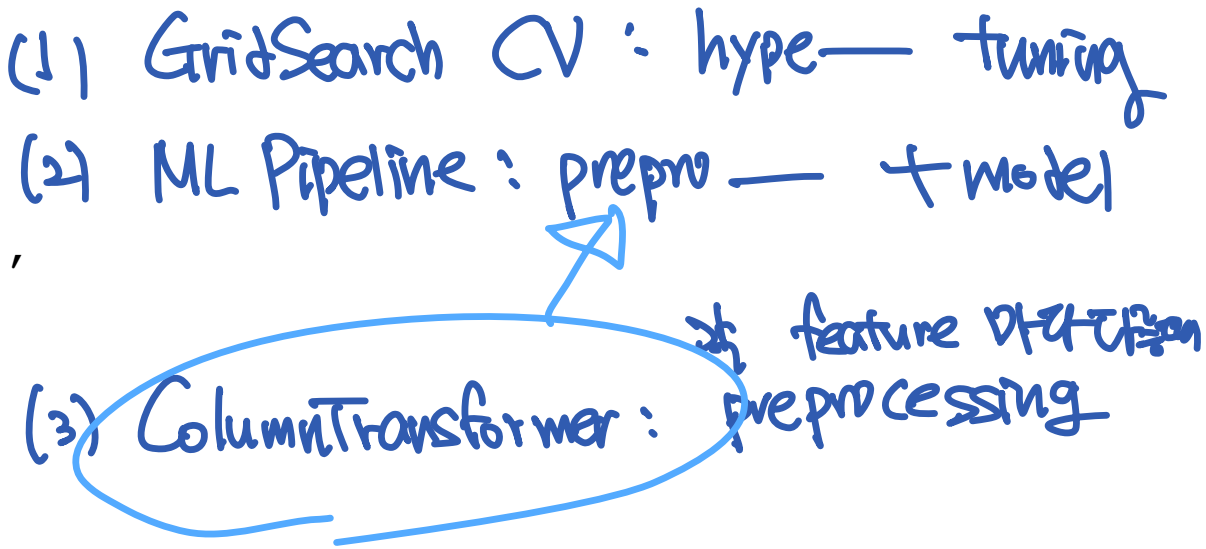
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder


num_pipe = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])
cat_pipe = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])
```

ColumnTransformer

- Code Example

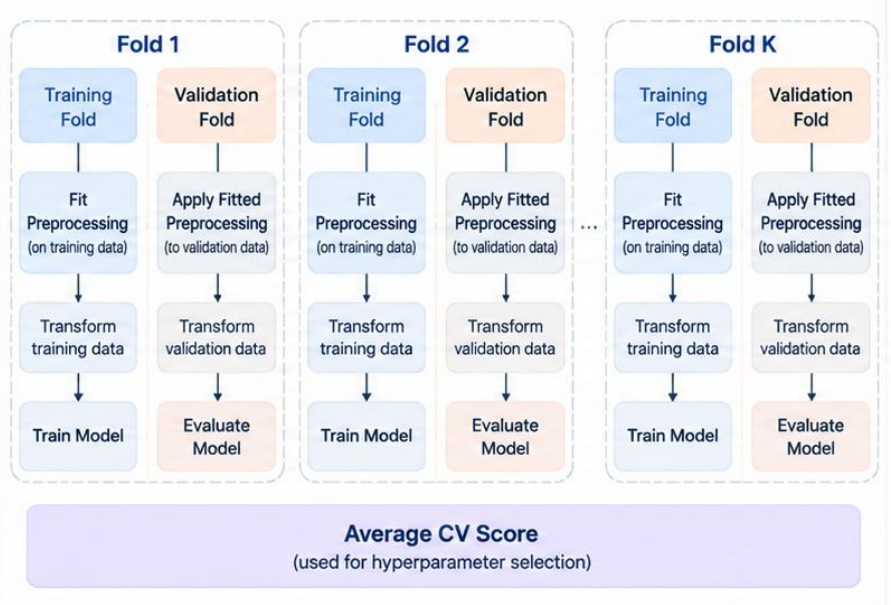
```
preprocessor = ColumnTransformer([
    ("num", num_pipe, ["age", "income"]),
    ("cat", cat_pipe, ["city"])
])
```



- Applies different preprocessing to numerical and categorical features
- Combines their outputs into a single feature matrix

Pipeline + GridSearchCV

- Pipelines can be optimized using GridSearchCV



- During each cross-validation fold:
  - Fit preprocessing steps on the training fold
  - Train the model using transformed training data
  - Evaluate on the validation fold

- Use `step__parameter` to tune pipeline components

Pipeline + GridSearchCV

- Code Example

```
pipe = Pipeline([
    ("preprocess", preprocessor),
    ("model", SVC(kernel="rbf"))
])
param_grid = {
    "model__C": [0.1, 1, 10],
    "model__gamma": [0.01, 0.1, 1]
}
search = GridSearchCV(
    pipe, param_grid, cv=5, scoring="accuracy"
)
search.fit(X_train, y_train)
```



Reproducibility

- Reproducibility means obtaining consistent results when experiments are repeated under the same conditions
- Important practices
  - Random Seeds: control randomness
  - Data Versioning: track dataset versions
  - Configuration Tracking: record model settings
  - Environment Tracking: record software versions
- Experiment tracking tools
  - MLflow
  - Weights & Biases
- Fixing random seeds alone does not guarantee complete reproducibility

Next Lecture

- Lecture 9: Neural Networks & Representation Learning
  - From Linear Models to Neural Networks
  - Multi-Layer Perceptrons (MLPs)
  - Activation Functions
  - Gradient-Based Learning
  - Representation Learning
  - From Neural Networks to Deep Learning

Korea Aerospace University

Questions?

jiyoonshin@kau.ac.kr