Episode 5: ML Lifecycle Fundamentals

A High-Level Guide to Building Machine Learning Pipelines

AI & ML: From Theory to Practice

Matthew Care 2026

What You'll Learn

By the end of this episode, you will understand:

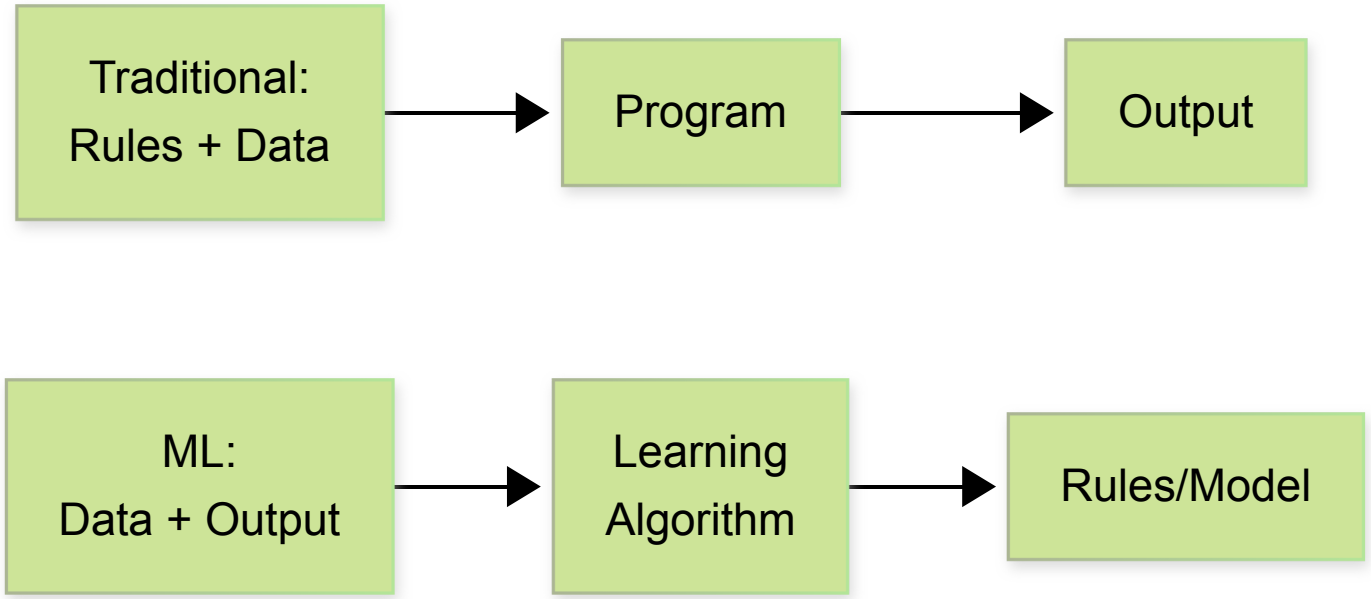
- What machine learning is and the types of problems it solves
- The **end-to-end ML lifecycle** — the steps from raw data to predictions
- Why **data preparation** (cleaning, scaling, encoding) is critical
- How we **train, test, and evaluate** models fairly
- The most common **pitfalls** that undermine ML projects

What is Machine Learning?

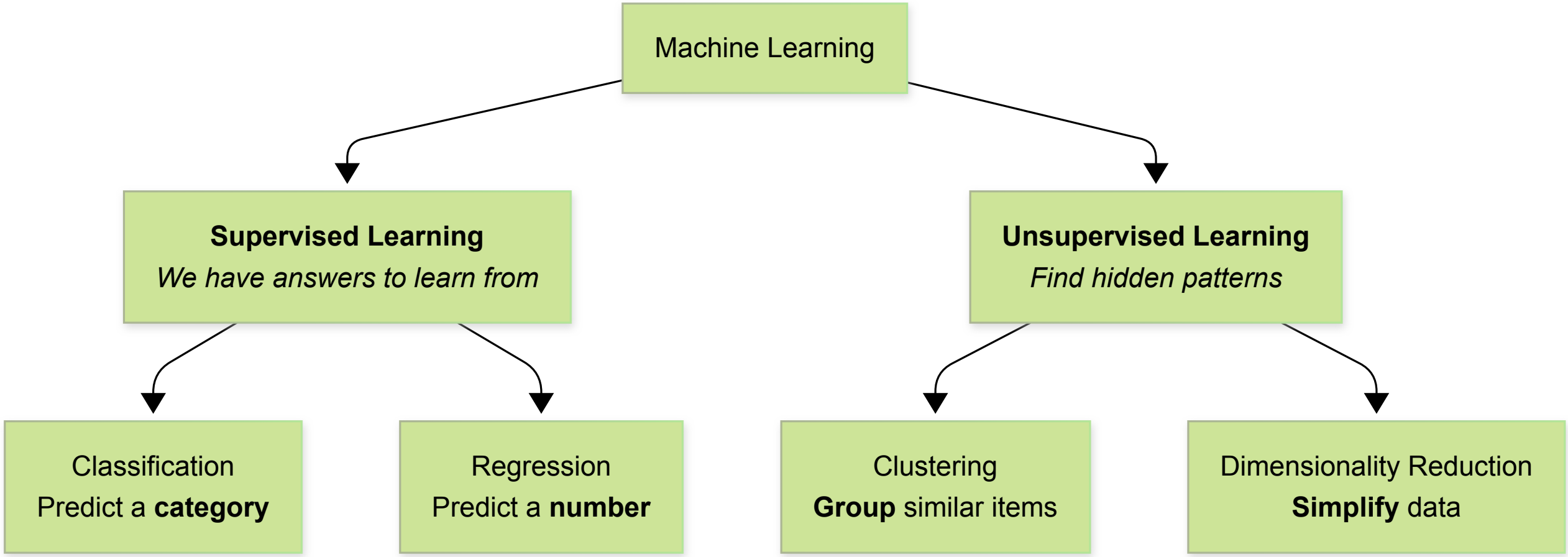
Traditional programming: You write explicit rules

Machine learning: The computer learns rules from examples

> *Think of it like teaching someone to cook. Traditional programming is giving them a precise recipe. Machine learning is showing them 1,000 meals and letting them figure out the patterns.*



Types of Machine Learning



Classification vs Regression

Classification

Predict a **category or class**

- Is this email spam? *(Yes/No)*
- What disease subtype is this?
- Which species does this belong to?

> *Like sorting items into labelled boxes*

Regression

Predict a **continuous number**

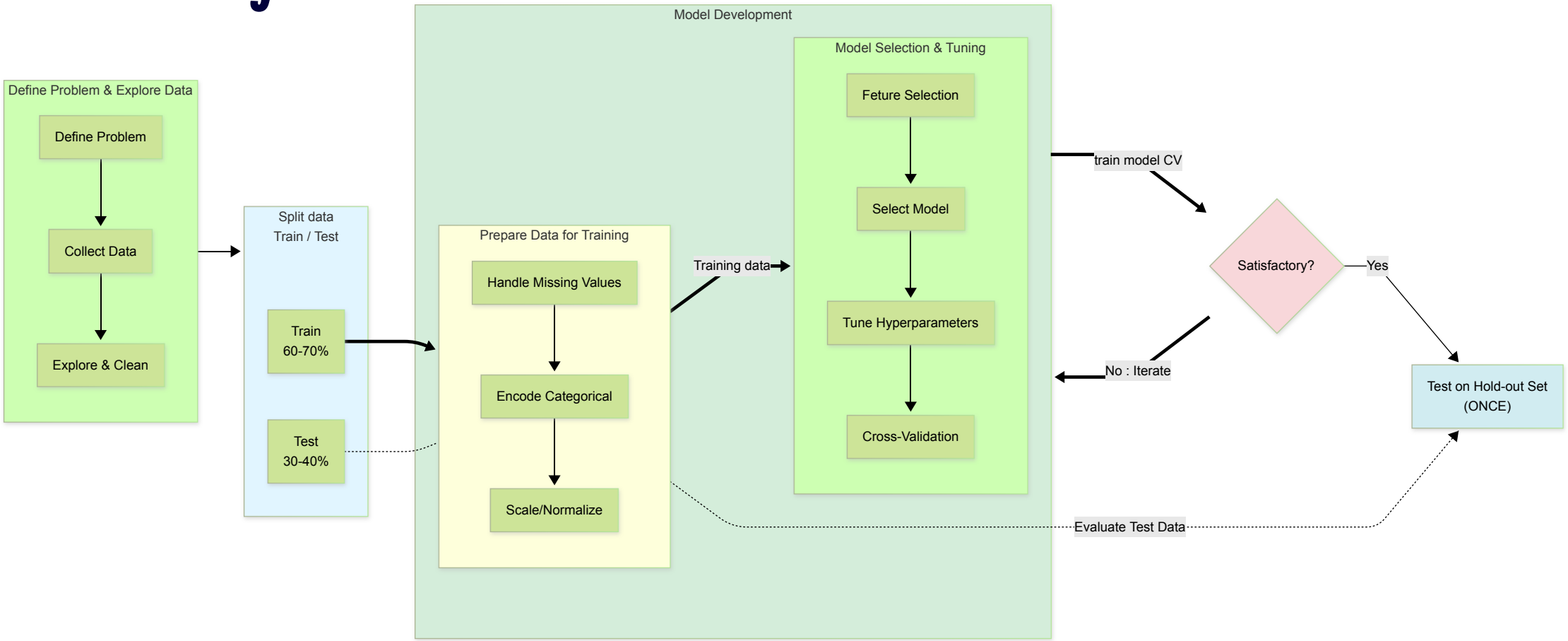
- What will the house price be?
- How many sales next month?
- What temperature tomorrow?

> *Like estimating a value on a scale*

The ML Lifecycle

The Journey from Raw Data to Predictions

The ML Lifecycle: Overview



> *Building an ML model is not a straight line — it's a cycle. You'll often revisit earlier steps as you learn more about your data and problem.*

The ML Lifecycle Unpacked

Understand

1. **Define the problem** — What question are we answering?
2. **Collect data** — Gather relevant, representative examples
3. **Explore** — Understand distributions, quality, patterns

Build

4. **Split** — Separate training from testing data
5. **Prepare** — Clean, scale, encode
6. **Train** — Fit model(s) to training data
7. **Evaluate** — Measure performance honestly (on unseen data!)

> *Each step has pitfalls. The rest of this episode covers what can go wrong and how to do each step properly.*

The Toolbox

Choosing the Right Tools for the Job

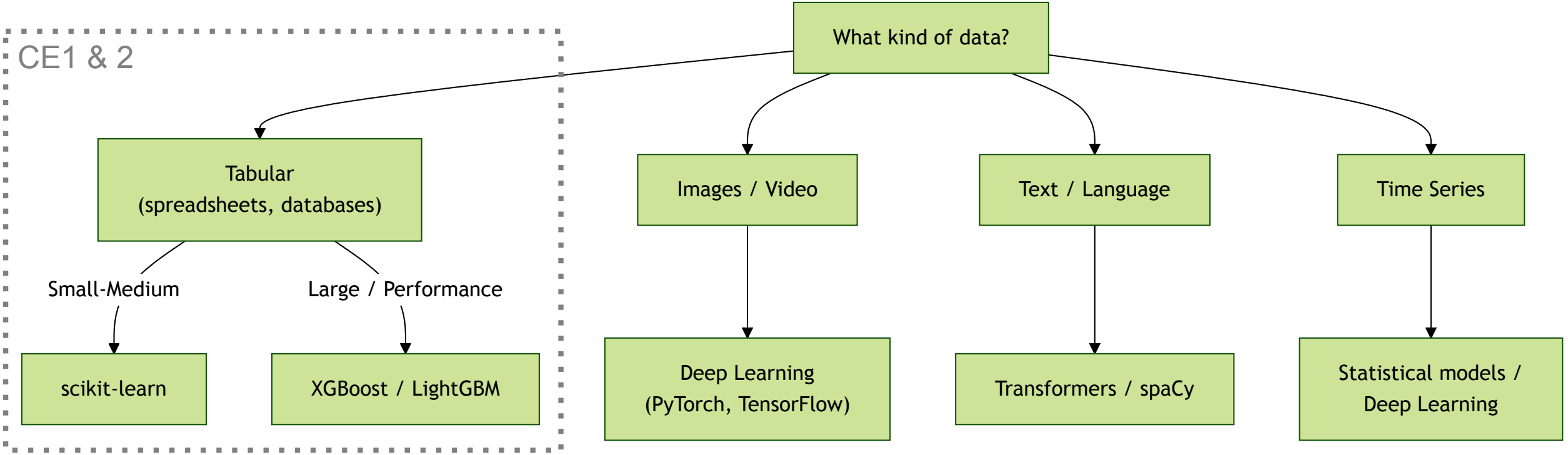
The ML Ecosystem

The ML landscape can seem daunting, but most projects only need a **few key tools**:

Tool Category	What It Does	Example Tools
Data handling	Load, clean, manipulate data	pandas, NumPy
Classical ML	Train traditional ML models	Scikit-learn, XGBoost
Deep Learning	Neural networks for complex data	PyTorch, TensorFlow
NLP	Process text/language	Transformers, spaCy
Visualisation	Plot and explore data	matplotlib, seaborn

> *For tabular/structured data (spreadsheets, databases), classical ML tools are often all you need — and often outperform deep learning.*

When to Use What?



> *Start simple. Only move to more complex tools when simpler ones clearly aren't enough.*
> Episode 5/6 focus on **scikit-learn** - good framework for learning ML concepts.

Understanding Your Data

Before You Model, You Must Explore

Anatomy of a Tabular Dataset

> *A dataset is like a spreadsheet — but with specific ML terminology.*

Feature 1	Feature 2	Feature 3	…	Target
value	value	value	…	label
value	value	value	…	label
…	…	…	…	…

- **Rows** = Individual observations *(e.g., one patient, one transaction)*
- **Columns** = Measured characteristics (**features**) *(e.g., age, income, gene expression)*
- **Target** = What we're trying to predict *(e.g., disease subtype, income level)*

Feature Types

Numerical

Values you can measure and order

- **Continuous:** Height, weight, temperature
- **Discrete:** Number of children, age in years

> *These can be used directly by models (but should be scaled!)*

Categorical

Labels or groups

- **Nominal:** Colour, country, blood type *(no order)*
- **Ordinal:** Education level, pain scale *(ordered)*

> *These must be converted to numbers first*

Why does this matter? Different feature types need different preparation steps before a model can use them.

Exploratory Data Analysis (EDA)

Know your data before modelling!

Before building any model, always check:

Check	Why It Matters
Size — How many rows and columns?	Too few samples can limit model quality
Balance — Are categories equally represented?	Imbalance can mislead evaluation
Missing values — Are there gaps?	Most models can't handle blanks
Data types — Numerical vs categorical?	Determines preprocessing steps
Distributions — Normal? Skewed? Outliers?	Affects scaling and model choice

> *Skipping this step is like building a house without checking the foundation.*

The Class Imbalance Problem: A Preview

Example: Predicting income level from census data

Income Group	Percentage
≤ $50K/year	**76%**
> $50K/year	**24%**

Problem: A model that *always* predicts "≤$50K" is correct 76% of the time — but it's completely useless! It never identifies high earners.

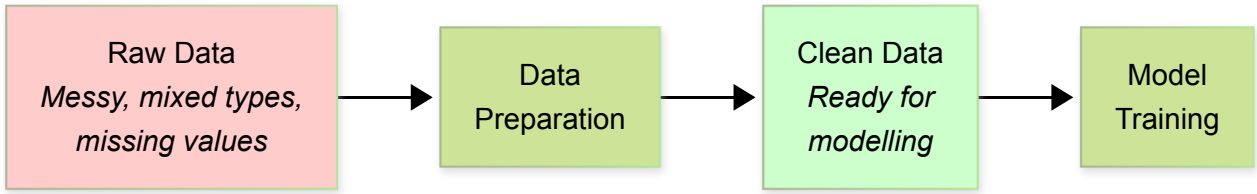
> *Imagine a fire alarm that never goes off — it's "correct" 99.99% of the time, but worthless when there's actually a fire.*

*We'll **address this properly** in Episode 6 with better evaluation metrics.*

Preparing Your Data

Cleaning, Scaling, and Encoding

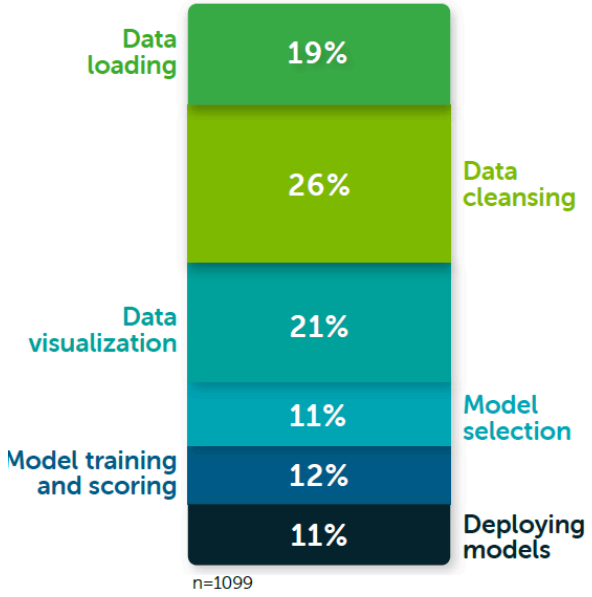
Why Data Preparation Matters



> ***"Garbage in, garbage out"*** *— even the best algorithm can't learn from poorly prepared data. Data scientists typically spend > 45% of their time on data preparation.*

The three main preparation steps:

1. **Handle missing values** — fill gaps or remove incomplete records
2. **Scale numerical features** — put features on comparable scales
3. **Encode categorical features** — convert text categories to numbers



Handling Missing Data

Real-world data is **messy**. Missing values are common and must be dealt with.

Types of Missingness

Type	Meaning	Example
MCAR	Missing completely at random	Random sensor failure
MAR	Missingness depends on other observed data	Older respondents skip online questions
MNAR	Missingness depends on the missing value itself	High earners don't report income

> *Understanding **why** data is missing helps you choose the right strategy.*

See https://stefvanbuuren.name/fimd/sec-MCAR.html#sec:MCAR

Strategies for Missing Data

Deletion

Remove incomplete records

- Simple and fast
- Works well if < 5% is missing
- Can lose valuable information
- May introduce bias

Imputation

Fill in estimated values

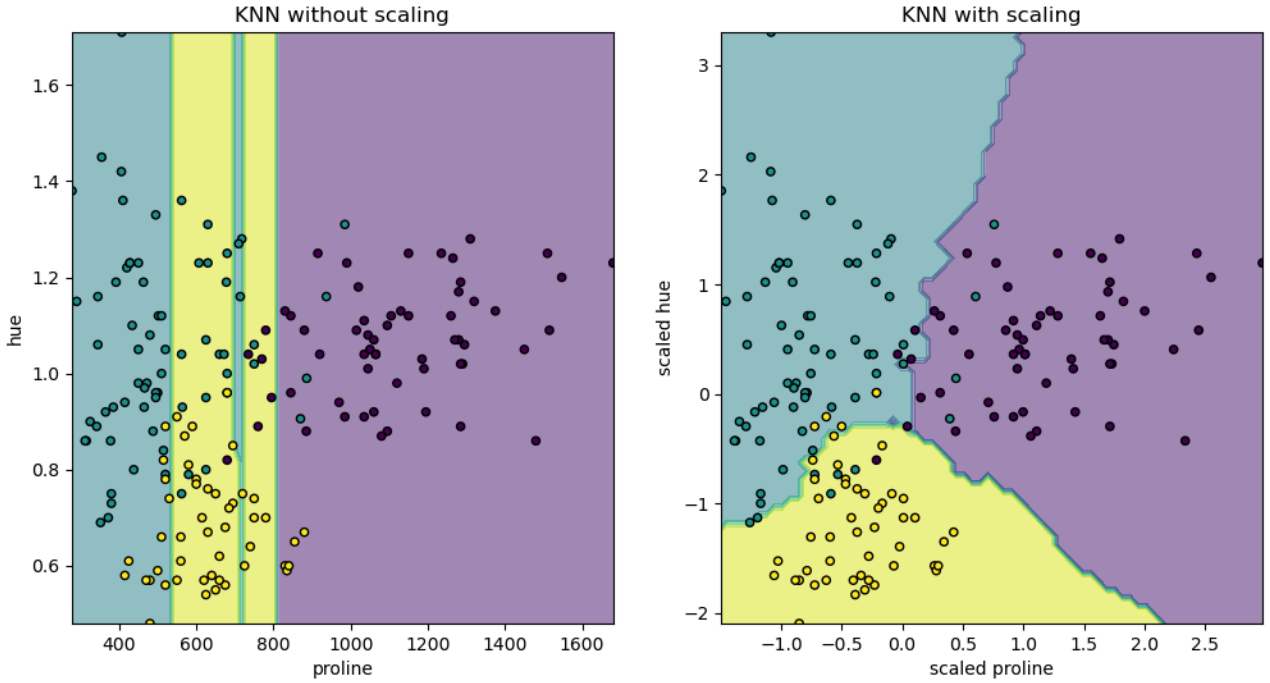
- Preserves sample size
- **Median** for numerical (robust to outliers)
- **Most common** value for categorical
- Advanced: use relationships between features

Rule of thumb: If only a small fraction of data is missing, deletion is fine. If missingness is substantial, imputation preserves information but introduces assumptions.

Why Scaling Matters

Problem: Features measured on differer scales can distort model learning.

Feature	Typical Range
Age	17 – 90
Annual income	0 – 500,000
Hours per week	1 – 99



> *Without scaling, a feature with **larger numbers dominates** simply because of its scale, not because it's more important. It's like comparing distances in miles vs millimetres.*

Many algorithms (but not all) are sensitive to feature scales:

- **Affected:** Distance-based methods: neural networks, linear models, SVM, Kmeans etc...
- **Not affected:** Tree-based methods: (Random Forest, Gradient Boosting)

Normalization Methods Overview

Method	Formula	Output Range	Use When
StandardScaler	$\frac{x-\mu}{\sigma}$	Unbounded	Normally distributed data
MinMaxScaler	$\frac{x-min}{max-min}$	[0, 1]	Bounded range needed
RobustScaler	$\frac{x-median}{IQR}$	Unbounded	Outliers present
QuantileTransformer	Rank → distribution	[0, 1] or Normal	Non-normal data

*All methods share the same goal: put features on a **level playing field** so the model treats them fairly.*

Encoding Categorical Features

ML algorithms need numbers. Categories must be converted.

The Wrong Way: Simple Numbering

Category	Number
Private sector	0
Government	1
Self-employed	2

Problem: This implies Government (1) is "between" Private (0) and Self-employed (2). **The model may treat these as ordered when they're not!**

One-Hot Encoding: The Right Approach

Create a separate yes/no column for each category:

Private?	Government?	Self-employed?
1	0	0
0	**1**	0
0	0	**1**

- No **implied** ordering between categories
- Each category is equally different from the others
- Trade-off: **more columns** (one per category)

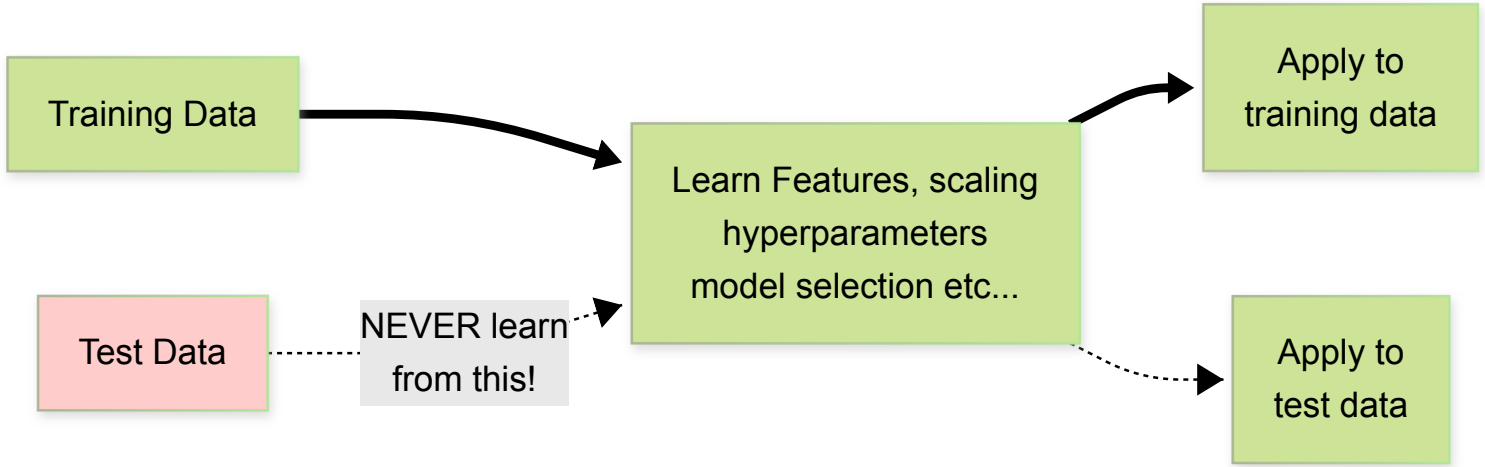
> *Think of it like ticking boxes on a form rather than assigning ranking numbers.*

Use numbering only when categories have a genuine order (e.g., education level: school < bachelor's < master's < doctorate)

The Golden Rule of Data Preparation

Always prepare data using ONLY the training set

Never let information from your **test data influence** your preparation steps.



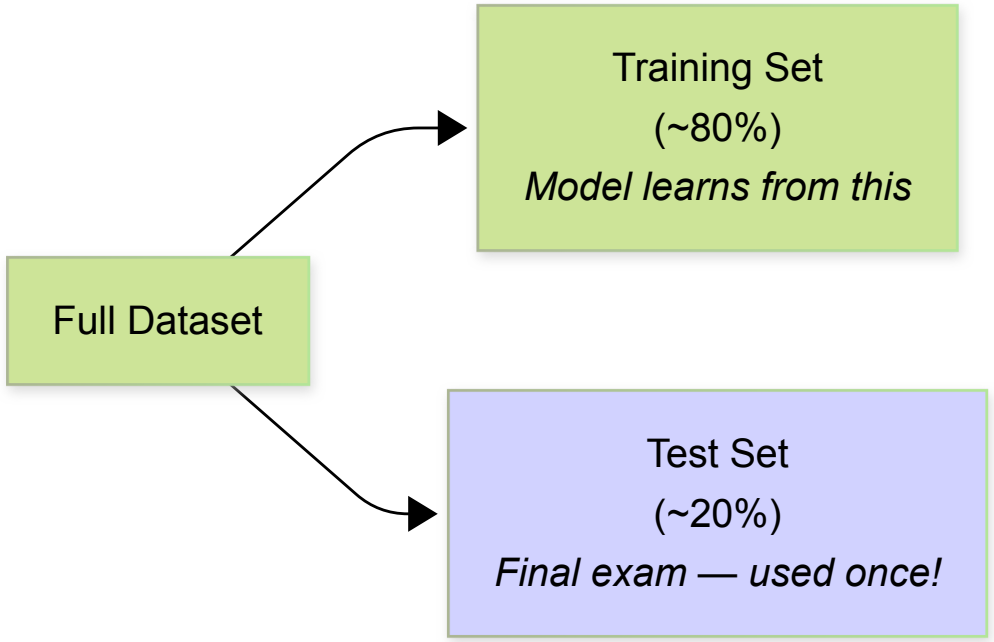
> *This is like taking an exam — your preparation shouldn't include seeing the exam paper. If it does, your score won't reflect real-world performance...*

*This concept (**data leakage**) is the single most important pitfall in ML (very easy to do without knowing) → We'll revisit it throughout the course.*

Training & Evaluation

How We Build and Test Models Fairly

Splitting Your Data



- **Training set:** The model learns patterns from this data.
- **Test set:** Held back to evaluate how well the model generalises to *unseen* data

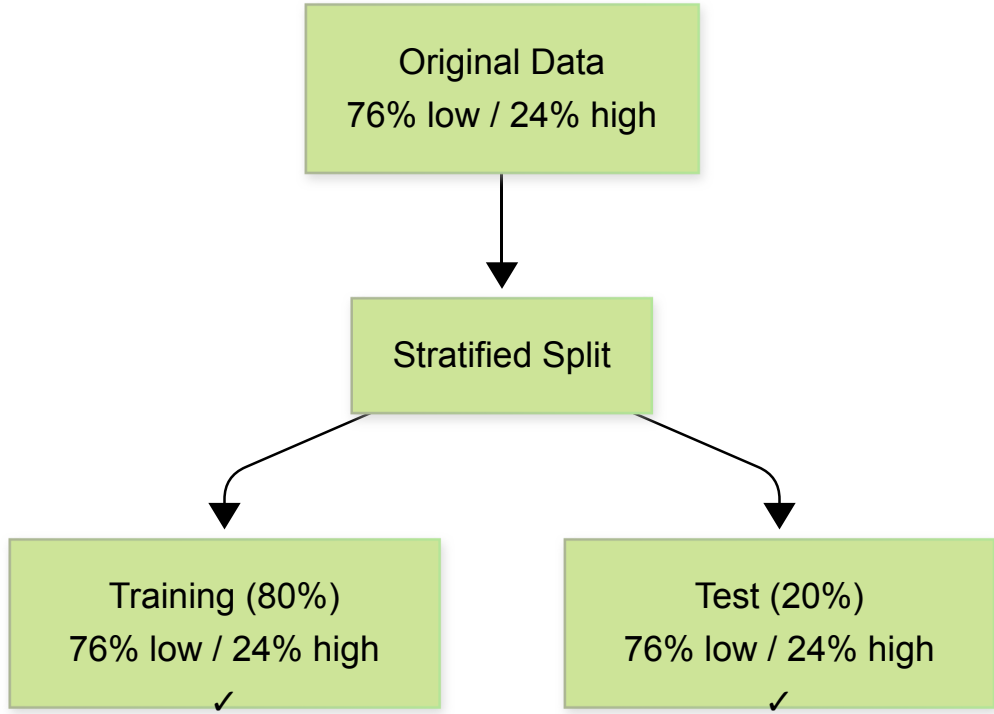
> *If you test a student with the same questions they practised on, you measure memorisation, not understanding. The test set must be separate and used only once.*

Stratification: Preserving Balance

When splitting, ensure each subset reflects the original class proportions:

Without stratification, random chance could produce very different proportions in train vs test — especially with small or imbalanced datasets.

> It's like making sure a clinical trial includes the right proportions of age groups, sexes, or risk categories — so results reflect the real patient population instead of overrepresenting one group



K-Fold Cross-Validation

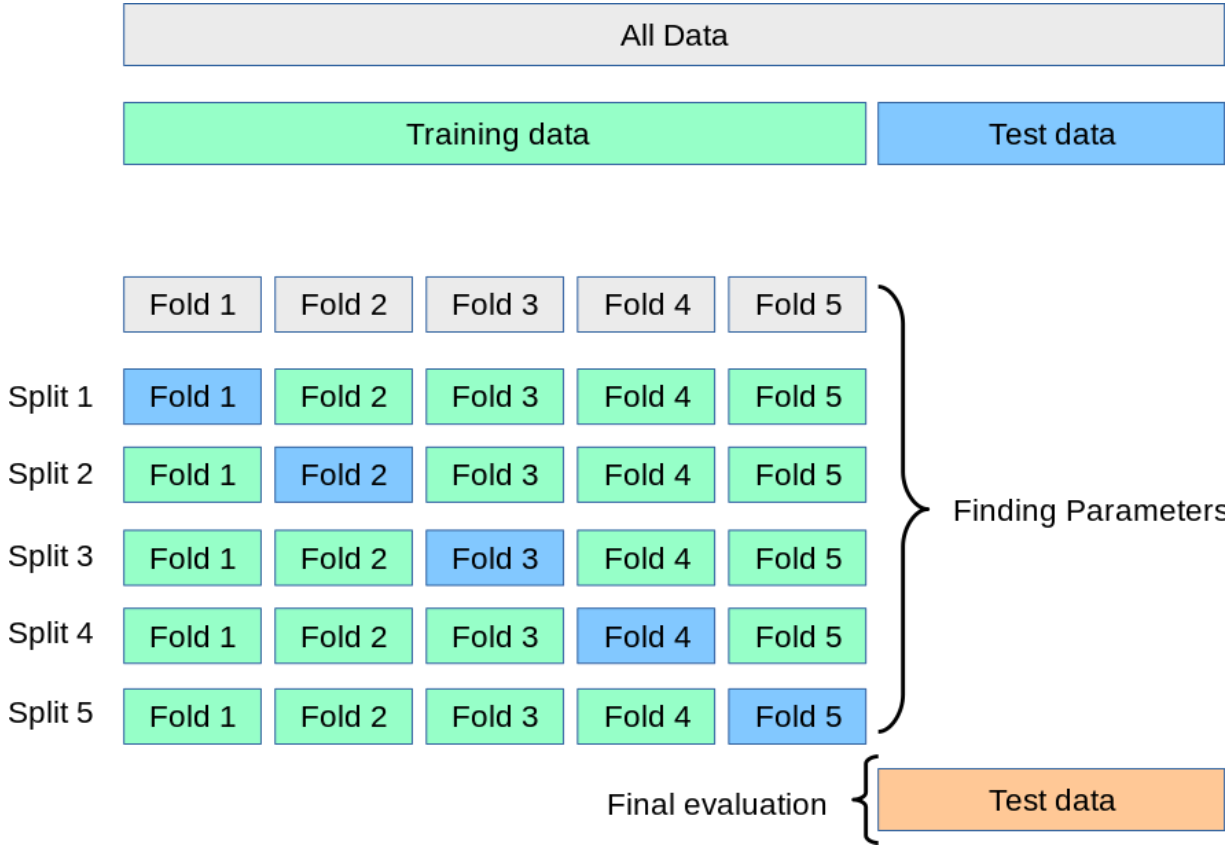
Problem: A single train/test split depends on *which* data ends up in each set.

Solution: Test on **multiple different splits** and average the results.

Divide data into K parts, train on K-1, test on 1, rotate:

Every sample is tested exactly once! The average gives a more reliable performance estimate.

see: sklearn docs



Why Cross-Validation Matters

Single Split	5-Fold Cross-Validation
One performance estimate	K estimates, averaged (k=5 or 10)
Result depends on which data lands where	Much more stable estimate
Fast	Slower (trains K models)
May overestimate or underestimate	Reliable average with uncertainty

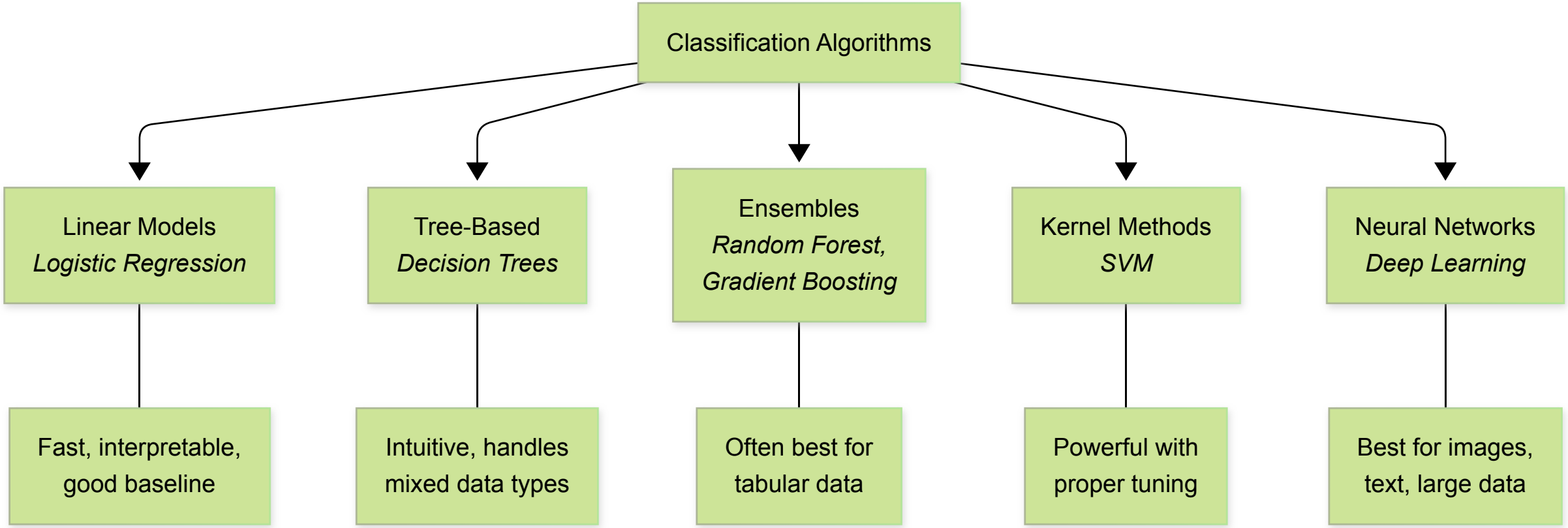
> *A single exam score might not reflect a student's true ability — they might have had a good or bad day. Multiple exams across different topics give a fairer picture.*

Always use stratified cross-validation for classification to maintain class proportions in each fold.

Choosing a Model

Which Algorithm Should You Use?

Algorithm Families



Practical Model Selection

Situation	Consider
First attempt / baseline	Simple linear model
Need to explain decisions	Decision trees, linear models
Tabular data, best accuracy	Gradient Boosting, Random Forest
Many features, risk of overfitting	Random Forest
Images, text, audio	Deep Learning
Very large dataset	Neural Networks, linear models

Best practice: Try several models and compare (via CV) — don't assume one will be best. Modern tools make this easy.

> *Choosing a model is like choosing a vehicle — the best choice depends on the terrain (data), the journey (problem), and the cargo (constraints).*

Common Pitfalls

Mistakes That Undermine ML Projects

Pitfall 1: Data Leakage

Data leakage occurs when information from outside the training set influences the model building process.

Common causes:

- Preparing data (scaling, encoding) before splitting train/test (as in CE1!)
- Using future information to predict past events
- Using all data for feature selection ← *common mistake*
- Not properly isolating test data during cross-validation

> *It's like a student who accidentally sees the exam paper before the test. They'll score well, but it doesn't reflect genuine understanding — and they'll fail on any new questions.*

Data leakage can make completely random data appear highly predictive. This is the #1 most dangerous pitfall in ML.

Pitfall 2: Trusting Accuracy Alone

Model	Accuracy	Reality
Always predicts majority class	76%	**Useless!** Never identifies the minority
Actual trained model	82%	Only **6% "better"** than doing nothing?

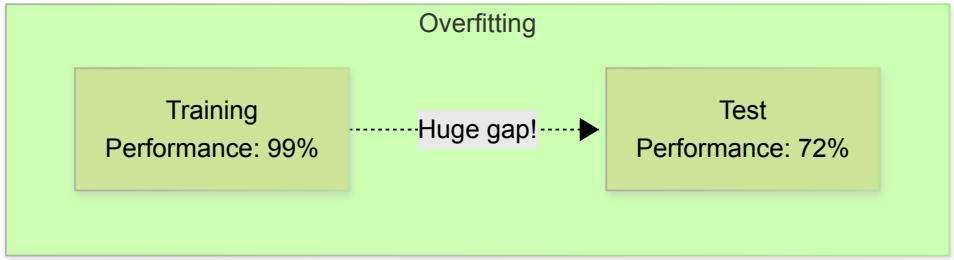
Accuracy doesn't tell you whether your model is genuinely finding patterns or just echoing the class distribution.

Solution: Use metrics designed for imbalanced data *(covered in Episode 6 / CE2)*

> *A weather forecaster in LA who always predicts "sunny" is right 75% of the time — but adds no value.*

Pitfall 3: Overfitting

The model memorises the training data instead of learning general patterns.



Signs: *Large* gap between training and test performance

Causes: Model too complex for the amount of data

Solutions:

- Get more training data
- Use a simpler model - apply **regularisation** (penalty for complexity)
- Use cross-validation for early detection

> *An overfitted model is like a student who memorises exam answers word-for-word but can't apply the concepts to new questions.*

Pitfall 4: Not Stratifying Splits

Random splits can accidentally produce unbalanced subsets:

Without Stratification

- Train: 80% low / 20% high
- Test: 65% low / 35% high

Mismatched distributions!

With Stratification

- Train: 76% low / 24% high
- Test: 76% low / 24% high

Proportions preserved ✓

Always stratify when splitting data for classification. It's simple, costs nothing, and prevents problems.

Pitfall Checklist

Before trusting your model's results, verify:

- [ ] Data preparation was fitted on **training data only**
- [ ] **Stratified splitting** was used for classification
- [ ] **Cross-validation** was used for performance estimates
- [ ] **Appropriate metrics** were chosen for the class distribution
- [ ] The gap between train and test performance is **reasonable**
- [ ] The test set was used **only once** for final evaluation

> *Think of this as a pre-flight checklist. Skipping items may seem fine most of the time, but when something goes wrong, the consequences are serious (or embarrassing → publishing incorrect results).*

Key Takeaways

1. **ML learns from examples** rather than following explicit rules
2. **Data preparation** (gathering, cleaning, scaling, encoding) is 60–80% of the work
3. **Different feature types** need different treatment
4. **Train/test splits** must be stratified and kept separate
5. **Cross-validation** gives more reliable performance than a single split
6. **No single "best" algorithm** — try several and compare
7. **Data leakage** is the most dangerous pitfall — never let test data influence training
8. **Accuracy alone is misleading** for imbalanced data *(see Episode 6)*
9. **Overfitting** means memorisation, not learning

Questions Before CE1?

Up Next: Coding Exercise 1

You'll apply everything from this episode to build your first ML workflow!

- Work through at your own pace
- Run each cell in turn and **understand** the output
- Experiment with different parameters
- Compare your results to expected outputs

Remember: Making mistakes is learning. If something doesn't work, debug it!

Feel free to ask questions as you proceed.

What's after CE1?

Episode 6: Metrics, Pipelines & Hyperparameter Tuning

Now you understand the ML lifecycle. Next, we'll learn:

- Why accuracy fails and what metrics to use instead
- How **Pipelines** automate data preparation and prevent leakage
- How **hyperparameter tuning** finds optimal model configurations
- How to combine models for better predictions

> *Episode 5 taught you the basics. Episode 6 teaches you how to do it properly.*

Resources and Documentation

Official Documentation:

- scikit-learn User Guide
- scikit-learn API Reference

Tutorials:

- sklearn Getting Started
- pandas User Guide

Cheat Sheets:

- sklearn Algorithm Cheat Sheet