INSA ROUEN NORMANDIE

Machine Learning

Methodology, Evaluation and Validation

ITI 4 - Benoit Gaüzère

Learning objectives

By the end of this lesson, you should be able to:

- design a complete **learning and evaluation protocol**;
- assign distinct roles to training, validation and test data;
- select a cross-validation strategy that matches the data structure;
- identify common forms of **data leakage**;
- select metrics consistent with the scientific or operational objective;
- quantify uncertainty and assess the practical relevance of score differences.

Warm-up — which model wins?

Model	Validation accuracy	Test accuracy
A	0.91	0.88
B	0.89	0.90

Before answering, list every piece of information that is missing.

> Is a two-point difference large enough to matter?

Warm-up — the table is not enough

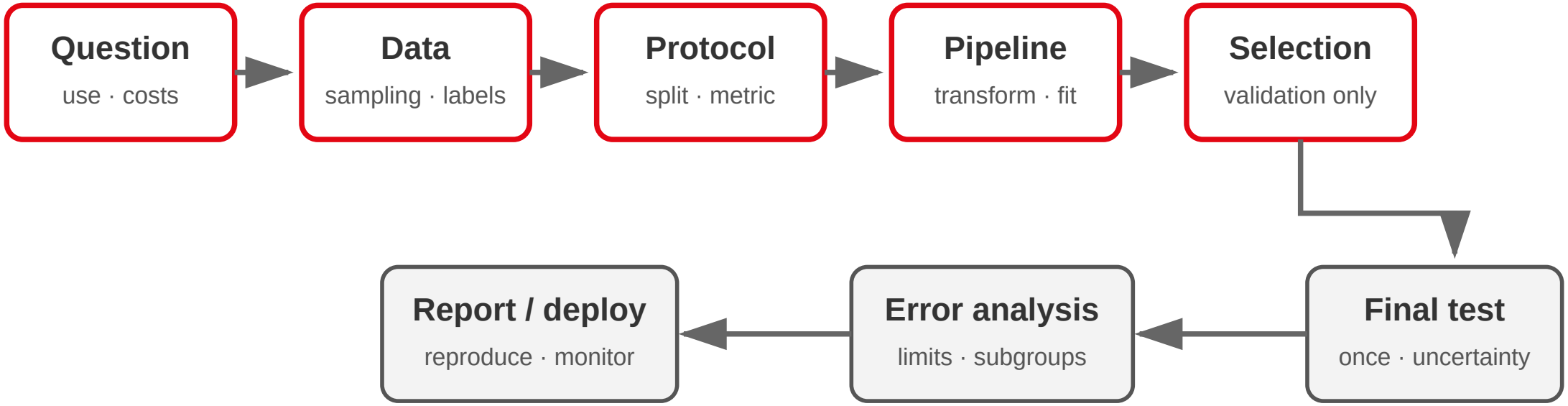
We need at least:

- the task and the cost of each kind of error;
- the dataset size, class balance and sampling process;
- how the partitions were constructed;
- whether the test set influenced any decision;
- variability across samples or splits;
- a baseline and the operational value of the difference.

A score has meaning only inside a documented protocol.

1. A complete machine-learning workflow

The workflow is the experiment



The model family is only one decision among many.

Start with the decision, not the algorithm

Before opening a notebook, specify:

1. **Generalisation:** what future cases should the model handle?
2. **Prediction:** what information is genuinely available at prediction time?
3. **Output:** class, probability, numerical value or ranking?
4. **Cost:** which errors matter?
5. **Success criterion:** what metric and minimum useful improvement?

These choices define the experimental conditions against which the model will be judged.

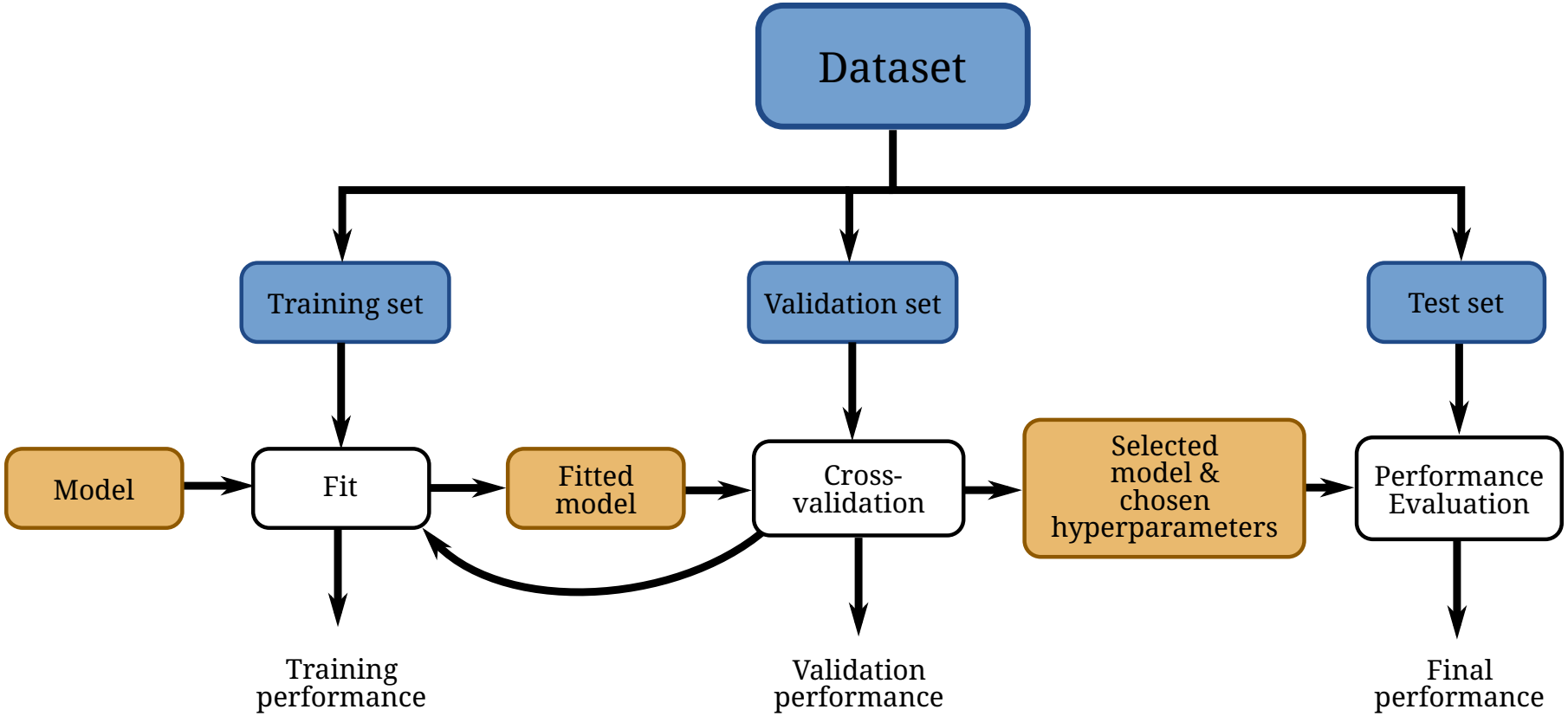
Translate the question into a predictive model

Suppose we want to identify defective parts before shipping.

Question	Explicit choice
Unit of prediction	one manufactured part
Available inputs	measurements collected before inspection
Target	defect confirmed by inspection
Intended population	parts produced next month
Main risk	shipping a defective part
Primary metric	recall at an acceptable false-positive rate

A vague objective such as “maximize accuracy” is not yet a protocol.

The learning procedure



During cross-validation, training and validation are **roles that rotate** within the development data.

Parameters and hyperparameters

Parameters	Hyperparameters
Learned by the fitting algorithm	Chosen around the fitting algorithm
Coefficients, tree splits…	Regularization, depth, number of neighbours…
Estimated from training data	Selected using validation data

Hyperparameter selection is itself a learning process. Its performance estimate is therefore optimistic on the data used for selection.

The test set is a firewall

Use training and validation information to choose:

- representations and preprocessing;
- model family and hyperparameters;
- threshold and decision rule;
- debugging actions.

Use the test set **once**, after the protocol is frozen, to estimate final performance.

Repeatedly checking the test result turns the test set into validation data.

After the final score

A responsible study does not stop at one number:

- inspect errors and performance across meaningful subgroups;
- document limitations and the intended population;
- check compute, latency and maintainability constraints;
- define monitoring for distribution and performance changes.

Error analysis may motivate a **new experiment**, with a new untouched test set.

2. Partitions and cross-validation

Three partitions, three roles

Partition	Used to fit parameters?	Used to choose the model?	Used for final reporting?
Training	✓	indirectly	no
Validation	no	✓	no
Test	no	no	✓

These partition names describe **roles**.

A simple holdout protocol

```
available labelled data
├── development set
│   ├── training set      → fit parameters
│   └── validation set    → select choices
└── test set              → final estimate
```

After selection, we may refit the chosen model on the full development set before evaluating it on the test set.

Can we reuse the test set?

A team evaluates version 1 on the test set. It then:

- adds a feature after inspecting test errors;
- modifies class weights;
- changes the prediction threshold;
- reports version 2 on the same test set.

Is the version-2 score still a valid final estimate? What could the team do now?

The test set has entered the feedback loop

Version 2 was indirectly selected using test information. Its reported score is optimistically biased.

Possible responses:

- label or collect a new independent test sample;
- use the old test set as part of development from now on;
- evaluate only after choices and success criteria are frozen;
- document the exploratory history honestly.

There is no statistical trick that makes forgotten information disappear.

One split can be unstable

The result depends on which observations happen to be in validation:

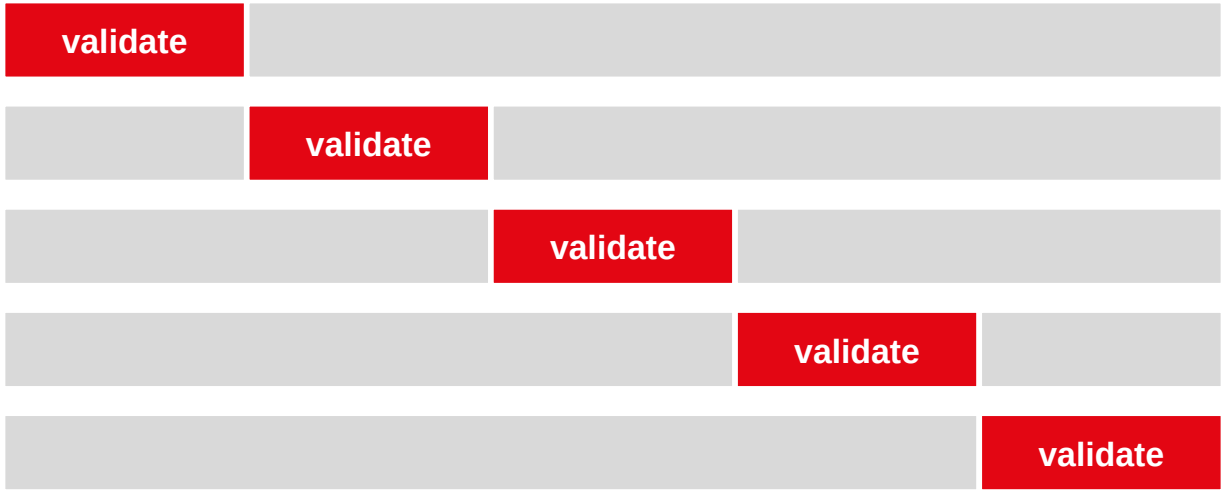
- small datasets produce noisy estimates;
- rare classes may be poorly represented;
- easy or hard groups may be unevenly distributed;
- a single seed hides sensitivity to the split.

Cross-validation rotates the validation role across several subsets.

k-fold cross-validation

Outer holdout

5-fold CV inside development data



Each sample:

- trains 4 times
- validates once

Mean and spread describe this validation procedure — not five independent experiments.

Every observation is used for validation once and for training $k - 1$ times.

Cross-validation algorithm

For folds $V_1, \ldots, V_k$:

1. fit the complete pipeline on all folds except V_j;
2. evaluate it on V_j;
3. repeat for $j = 1, \ldots, k$;
4. aggregate the fold scores.

$$\bar{s} = \frac{1}{k} \sum_{j=1}^{k} s_j$$

Common choices are $k = 5$ or $k = 10$: larger k costs more and is not automatically better.

What the fold standard deviation means

Reporting $\bar{s} \pm \mathrm{sd}(s_1, \ldots, s_k)$ describes variation among these fold scores.

It is **not automatically**:

- a confidence interval for future performance;
- the standard error of the mean divided by $\sqrt{k}$;
- evidence that two models differ significantly.

Stratified cross-validation

`StratifiedKFold` approximately preserves class proportions in every fold.

Useful when:

- the target is categorical;
- a minority class could disappear from a fold;
- the selected metric requires both classes.

Stratification stabilizes the engineering of folds; it does not fix an unrepresentative dataset.

scikit-learn: cross-validation strategies

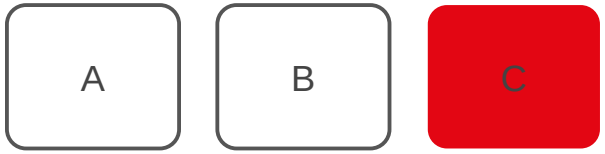
The split must preserve independence

Random / stratified

i.i.d. samples

preserve class ratios if needed

Grouped



patients · machines · molecules

keep each group intact

Temporal

forecasting · drift

never train on the future

training validation / test

The correct unit of independence may be a patient, device, site or time period, not a row.

Grouped data

Examples:

- several scans from one patient;
- several windows from one sensor recording;
- repeated measurements of one chemical compound;
- customers belonging to the same household.

Random row splitting lets the model encounter the same entity during training and evaluation.

Use group-aware splitting such as `GroupKFold` or `GroupShuffleSplit`.

Stratification with grouped data

Two constraints may apply at the same time:

- observations from one group must remain in the same fold.
- each fold should contain enough examples of every class.

`StratifiedGroupKFold` attempts to preserve class proportions while keeping groups disjoint.

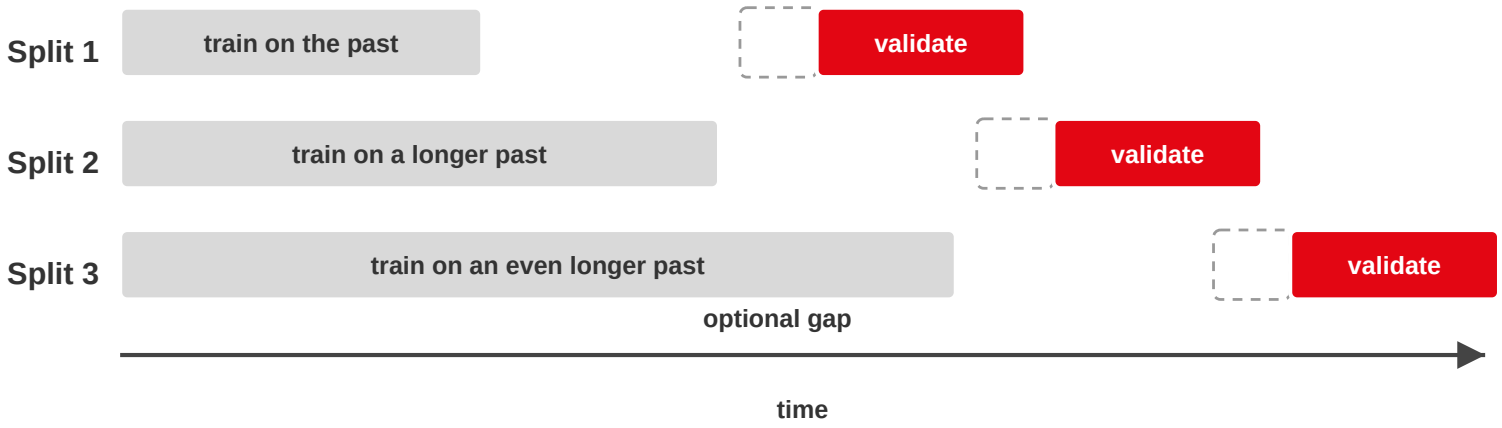
```
from sklearn.model_selection import StratifiedGroupKFold

cv = StratifiedGroupKFold(
    n_splits=5, shuffle=True, random_state=42
)
search.fit(X_dev, y_dev, groups=groups_dev)
```

`GroupShuffleSplit` preserves groups, but it does not stratify using `y`. Always inspect class prevalence after a grouped holdout split.

Time-ordered data

When predicting the future, training must precede validation:



Random splitting can transfer seasonality, future preprocessing statistics or near-duplicate time windows backward in time.

Use a chronological holdout or `TimeSeriesSplit`, possibly with a gap.

Augmentation and duplicates

Split **before** generating correlated observations.

If an original image is rotated into ten variants and those variants are randomly split, nearly identical content can appear in both train and test.

The same risk occurs with:

- overlapping signal windows;
- replicated records;
- multiple crops of one image;
- repeated experimental runs from one specimen.

Keep all descendants of one source in the same partition.

Choosing a splitter

Data-generating structure	Appropriate starting point
Independent regression rows	`KFold(shuffle=True)`
Independent classification rows	`StratifiedKFold(shuffle=True)`
Repeated entities	`GroupKFold`
Grouped classification	`StratifiedGroupKFold`
Prediction into the future	chronological holdout / `TimeSeriesSplit`
Deployment at unseen sites	hold out sites, possibly group CV

Ask: **what must be unseen for this evaluation to imitate deployment?**

Splitters in scikit-learn

```
from sklearn.model_selection import (
    StratifiedKFold, GroupKFold, TimeSeriesSplit
)

cv_iid = StratifiedKFold(
    n_splits=5, shuffle=True, random_state=42
)
cv_groups = GroupKFold(n_splits=5)
cv_time = TimeSeriesSplit(n_splits=5, gap=24)
```

Pass the splitter explicitly to make the experimental design visible.

Select the validation strategy

1. Predict cancer from several images per patient.
2. Predict tomorrow's electricity consumption.
3. Classify 500 independent samples with a 5% minority class.
4. Predict performance for a factory not represented in training.

For each case, what must be kept together or ordered?

Matching split to deployment

1. **Patient groups:** all images from one patient stay together.
2. **Time order:** train on the past, validate on later periods.
3. **Stratification:** preserve enough minority examples per fold.
4. **Factory groups:** hold out complete factories; ideally reserve target-like sites for final evaluation.

The split is part of the problem formulation, not a generic utility call.

3. Data leakage and shortcuts

What is data leakage?

Leakage occurs when information not legitimately available during training or at prediction time influences model fitting or selection.

It creates an evaluation protocol that is easier than the real task.

Two broad mechanisms:

- information crosses a partition boundary;
- a feature contains information from the future, the target or the decision process.

High scores can be a symptom, but leakage can also be subtle.

Leakage through preprocessing

Incorrect order:

```
all data → imputation / scaling / PCA / selection → split → evaluate
```

Correct order:

```
split → fit complete pipeline on training → transform validation/test → evaluate
```

Even unsupervised transformations use distributional information from X.

Leakage through target-guided selection

```
# Incorrect: selects features using every target value
X_selected = SelectKBest(k=20).fit_transform(X, y)
scores = cross_val_score(model, X_selected, y, cv=5)
```

Temporal and post-event leakage

Suppose the prediction is made when a patient arrives.

Invalid inputs include:

- a treatment administered after the diagnosis;
- the discharge code;
- statistics computed at the end of the hospital stay;
- a database field updated after clinical review.

Every feature needs an **availability timestamp**, not merely a semantic description.

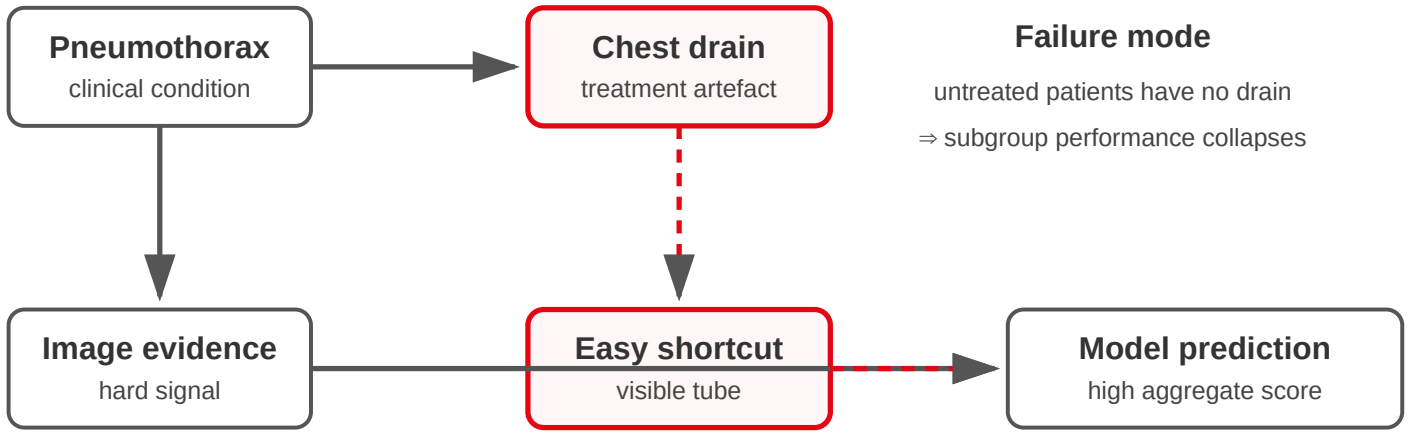
Leakage through identity

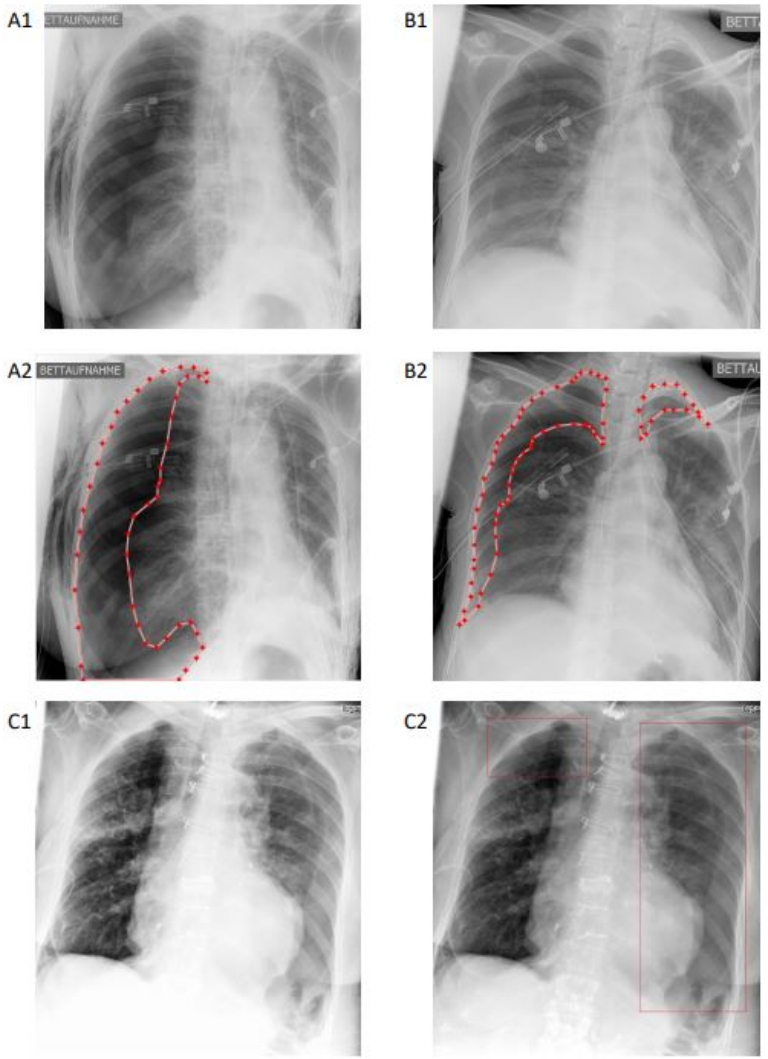
A model may identify entities instead of learning the intended relationship:

- hospital-specific image markers;
- device serial numbers;
- filenames containing class labels;
- background or acquisition protocol;
- duplicate patients in different partitions.

Random-split performance can look excellent while deployment to new entities fails.

A real shortcut: chest drains





The visible drain is an effect of clinical action, not evidence available for untreated cases.

Rueckel J. et al., “Pneumothorax detection in chest radiographs: optimizing artificial intelligence system for accuracy and confounding bias reduction using in-image annotations in algorithm training”, 2021.

Hidden stratification in medical imaging

Chest X-ray classifiers for pneumothorax were found to rely strongly on **intercostal drains**, which often appear after diagnosis and treatment.

The aggregate metric concealed poor performance on the clinically important subgroup: pneumothorax **without** a drain.

This illustrates both a shortcut and hidden subgroup structure.

Oakden-Rayner L. et al., “Hidden stratification causes clinically meaningful failures in machine learning for medical imaging”, 2020.

Leakage audit checklist

For every feature and transformation, ask:

- Was it available at the declared prediction time?
- Was any statistic estimated outside the training subset?
- Can the target be reconstructed directly or indirectly?
- Are related or duplicated observations split apart?
- Does annotation, treatment or acquisition reveal the label?
- Does validation reproduce the target population, site and time?

Then inspect errors by subgroup—not only the aggregate score.

Leakage or legitimate information?

A company predicts whether an invoice will be paid late.

Candidate features:

1. customer sector;
2. amount invoiced;
3. number of reminder emails sent;
4. historical late-payment rate computed last year;
5. current late-payment rate computed from the full database.

Assume prediction occurs when the invoice is issued. Audit each feature.

Leakage audit — timing decides

1. **Sector:** legitimate if recorded at issue time.
2. **Amount:** legitimate if finalized at issue time.
3. **Reminder emails:** post-event information; unavailable at issue time.
4. **Historical rate:** legitimate if frozen before the prediction period.
5. **Current full-database rate:** likely leaks future invoice outcomes.

Names are not sufficient: the computation date and data lineage determine validity.

Session 1 checkpoint

You should now be able to explain:

- why the test set is not a development dashboard;
- why every learned preprocessing step belongs in a pipeline;
- how groups and time change the split strategy;
- why leakage is defined relative to prediction time;
- how a model can exploit a shortcut without solving the intended task.

> Next: what exactly should we measure, and how certain are we?

4. Metrics must match the objective

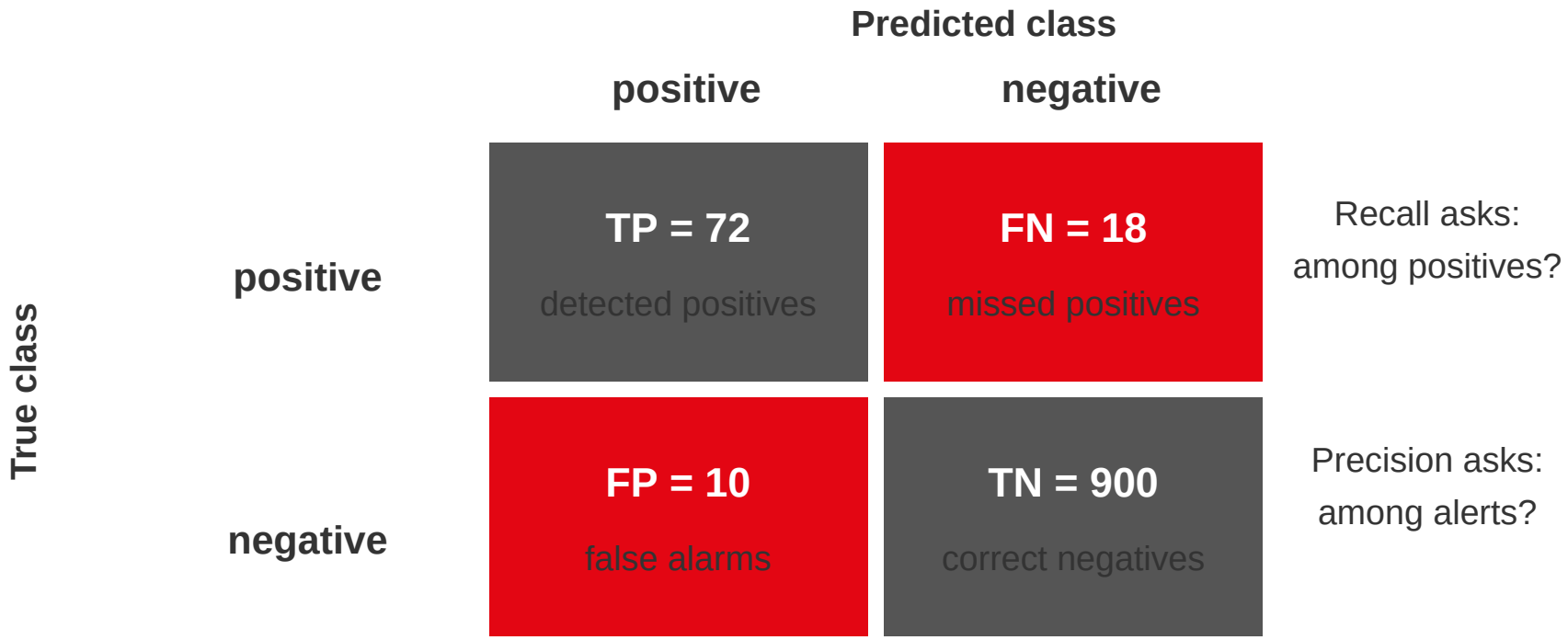
A metric encodes preferences

A metric decides how predictions become a number:

- Are all mistakes equally costly?
- Do we need one class decision or a ranking across thresholds?
- Is performance on each class important?
- Are large regression errors especially harmful?
- At what decision threshold will the model operate?

Choose a **primary metric before comparing models**. Add diagnostic metrics to explain it.

Confusion matrix



Rows and columns vary across software: always read the axis labels.

Core binary-classification metrics

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{precision} = \frac{TP}{TP + FP}, \qquad \text{recall} = \frac{TP}{TP + FN}$$

$$\text{specificity} = \frac{TN}{TN + FP}, \qquad F_1 = 2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Balanced accuracy averages recall over the two classes:

$$\text{balanced accuracy} = \frac{\text{recall} + \text{specificity}}{2}$$

Accuracy can hide the relevant failure

For 1,000 predictions:

	Predicted positive	Predicted negative
Actually positive	72	18
Actually negative	10	900

- Accuracy: **97.2%**
- Precision: **87.8%**
- Recall: **80.0%**
- Specificity: **98.9%**
- Balanced accuracy: **89.5%**

The apparently excellent accuracy coexists with 18 missed positive cases.

Precision or recall?

Prioritize recall when missing a positive is costly:

- disease screening;
- safety defects;
- fraud candidate generation.

Prioritize precision when acting on a positive is costly:

- intrusive follow-up;
- limited manual review;
- automatic account blocking.

Usually neither should be optimized without a constraint on the other.

Further reading: Powers D. M. W., “Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation”, 2020.

Choose a primary metric

1. Detect a rare but dangerous manufacturing defect.
2. Rank search results for later human inspection.
3. Trigger an expensive laboratory confirmation test.
4. Classify four equally important but imbalanced species.

Propose a metric and state the operational assumption behind it.

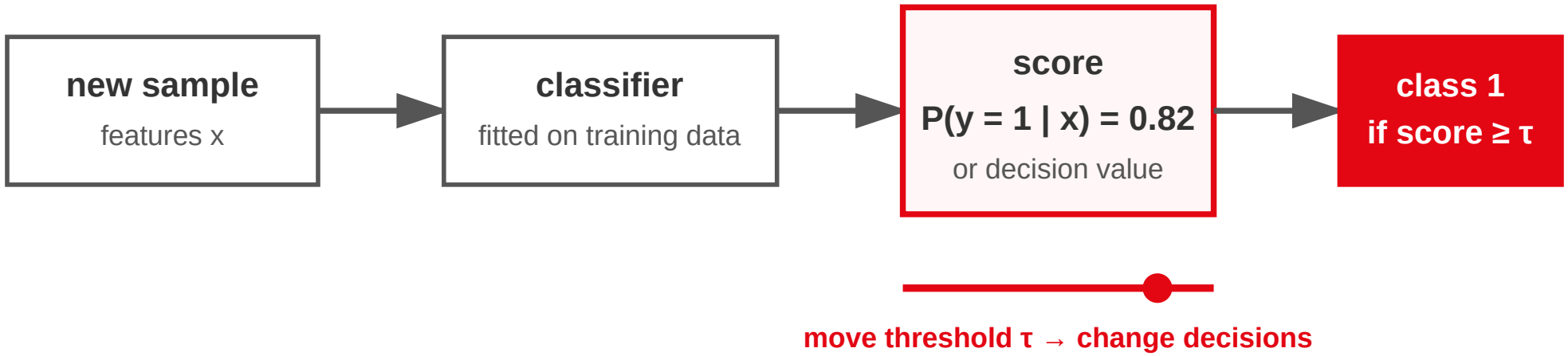
Metrics follow costs

Possible choices:

1. **Recall** at a maximum false-positive rate; missed defects dominate.
2. A **ranking metric**, such as average precision; ordering matters more than one threshold.
3. **Precision** at a required recall; confirmations are expensive but missed cases still matter.
4. **Macro-averaged F1** or balanced accuracy; each species should contribute similarly.

Different costs could justify different answers. The assumption must accompany the metric.

A classifier first produces a score



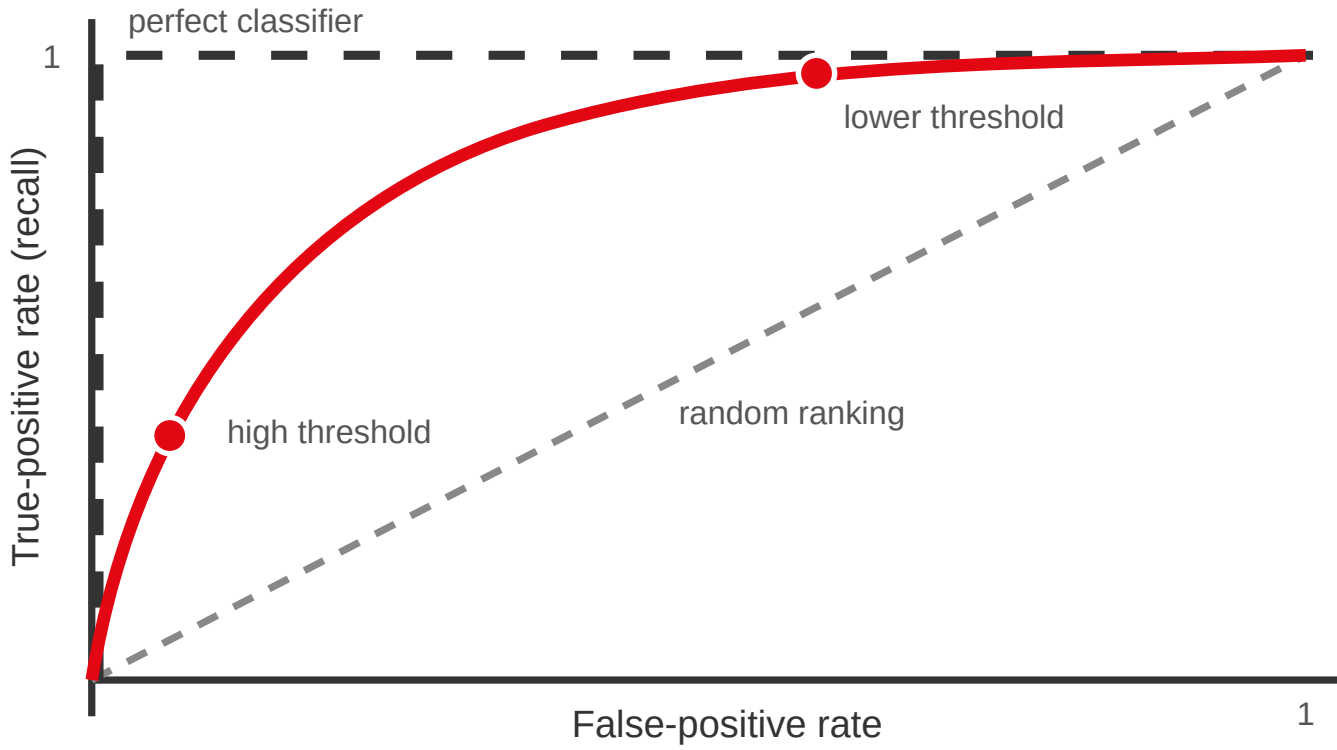
The usual class prediction hides this intermediate step. Moving the threshold τ changes the predicted classes **without refitting the model**.

Thresholds trace the ROC curve

Each threshold gives one point:

$$(\mathrm{FPR}, \mathrm{TPR}) = \left(\frac{FP}{FP + TN}, \frac{TP}{TP + FN} \right)$$

- A high threshold predicts few positives.
- Lowering it finds more positives, but usually creates more false positives.
- Aim toward the **top-left corner**.



ROC AUC summarizes ranking over all thresholds

- $\mathrm{AUC} = 1$: every positive is ranked above every negative.
- A random ranking has expected $\mathrm{AUC} = 0.5$.
- Example: $\mathrm{AUC} = 0.80$ means that a randomly chosen positive receives a higher score than a randomly chosen negative about 80% of the time.

ROC AUC summarizes all thresholds, but it does **not** choose the operating threshold used for the final decision.

Bradley A. P., “The use of the area under the ROC curve in the evaluation of machine learning algorithms”, *Pattern Recognition*, 30(7), 1145–1159, 1997.

Choosing an operating point on the ROC curve

The ROC curve displays possible trade-offs. The application determines the required operating point.

Define an operational constraint on development data, for example:

- recall must reach at least 80%.
- among eligible thresholds, choose the lowest false-positive rate.

The selected threshold becomes part of the learned procedure. Freeze it with the preprocessing and hyperparameters before evaluating the test set.

Choosing a point after inspecting the test ROC curve would use the test set for validation.

Threshold selection from out-of-fold scores

```
import numpy as np
from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict

scores_oof = cross_val_predict(
    search.best_estimator_, X_dev, y_dev,
    cv=cv, groups=groups_dev,
    method="predict_proba",
)[:, 1]

fpr, recall, thresholds = roc_curve(y_dev, scores_oof)
eligible = np.flatnonzero(recall >= 0.80)
index = eligible[np.argmin(fpr[eligible])]
decision_threshold = thresholds[index]
```

Out-of-fold scores avoid evaluating each observation with a model fitted on that observation.

Multiclass averaging

Compute a per-class metric, then choose how to aggregate:

- **macro:** unweighted mean across classes;
- **weighted:** mean weighted by class support;
- **micro:** pool all individual decisions before computing the metric.

Macro averaging makes rare classes visible. Weighted and micro averages can be dominated by frequent classes.

Always report the averaging convention.

Regression metrics

For residuals $e_i = y_i - \hat{y}_i$:

Metric	Definition	Main property
MAE	$\frac{1}{n}\sum_i \lvert e_i \rvert$	same unit as target; less sensitive to large errors than MSE
MSE	$\frac{1}{n}\sum_i e_i^2$	strongly penalizes large errors
RMSE	$\sqrt{\mathrm{MSE}}$	same unit as target; sensitive to large errors
R^2	$1 - \frac{\sum_i e_i^2}{\sum_i (y_i - \bar{y})^2}$	relative to mean predictor

“Best” depends on the cost structure, not on convention.

Interpreting R^2

- $R^2 = 1$: perfect predictions on the evaluated sample.
- $R^2 = 0$: the same squared error as predicting the target mean.
- $R^2 < 0$: worse than that constant baseline.

R^2 is unitless, but its value depends on target variability in the evaluated population.

There is no universal threshold for a “good” R^2: it does not tell us whether an error of 3 units is operationally acceptable.

Residual analysis

One aggregate error can hide structure. Plot residuals against:

- predicted value;
- important input variables;
- time and acquisition site;
- meaningful subgroups.

Look for bias, heteroscedasticity, nonlinear structure and extreme failures.

A useful model should fail in understood, documented ways.

Relative error needs care

Metrics based on $|y_i - \hat{y}_i|/|y_i|$ are tempting across scales, but:

- they explode when y_i is near zero;
- they are asymmetric for over- and under-prediction;
- arbitrary offsets change their interpretation.

State the denominator and check its scientific meaning. A normalized MAE or domain-specific tolerance may be safer.

Use a metric panel

A defensible evaluation often includes:

1. one predeclared **primary metric**;
2. metrics for important error types;
3. subgroup and residual diagnostics;
4. compute or latency constraints;
5. uncertainty around the main comparison.

Do not search many metrics and report only the one that tells the best story.

5. Baselines and hyperparameter selection

A baseline is mandatory

A model is useful only relative to a credible alternative.

- Classification: most frequent class, stratified random prediction, simple rule.
- Regression: mean or median predictor, last observed value for time series.
- Scientific task: current standard method or domain heuristic.

`DummyClassifier` and `DummyRegressor` provide reproducible data-independent baselines.

Hyperparameter selection loop

For each candidate configuration:

1. fit the **whole pipeline** on training folds;
2. evaluate the predeclared metric on validation folds;
3. aggregate validation results;
4. select according to the decision rule;
5. refit the chosen configuration on all development data;
6. evaluate once on the held-out test set.

The test score is not another hyperparameter-selection criterion.

Grid search

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

search = GridSearchCV(
    estimator=KNeighborsClassifier(),
    param_grid={"n_neighbors": [2, 5, 10, 15, 20]},
    scoring="balanced_accuracy",
    cv=cv_iid,
    n_jobs=-1,
)
search.fit(X_dev, y_dev)
```

But what if the learning procedure includes preprocessing?

Two kinds of learned objects

Transformer	Predictor
Defines how to transform X	Learns how to map X to a prediction
Scaling, imputation, encoding	k-NN, logistic regression, SVM…
`fit` + `transform`	`fit` + `predict`

Feature selection and dimensionality reduction are also **learned transformations**.

> Anything that calls `fit` must be trained without seeing validation or test information, including during grid search.

A pipeline defines one estimator

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

model = Pipeline([
    ("scaler", StandardScaler()),
    ("knn", KNeighborsClassifier()),
])
model.fit(X_train, y_train)
```

Explicit names make the nested hyperparameters easy to address.

Why pipelines matter

During each call to `fit`:

1. the scaler estimates means and standard deviations on the supplied training subset;
2. the predictor learns from the transformed subset;
3. `predict` reuses the fitted transformation.

This makes the workflow easier to reproduce and protects cross-validation from preprocessing leakage.

scikit-learn: pipelines and composite estimators

Tune the complete pipeline

```
search = GridSearchCV(
    estimator=model,
    param_grid={
        "knn__n_neighbors": [2, 5, 10, 15, 20]
    },
    scoring="balanced_accuracy",
    cv=cv_iid,
    n_jobs=-1,
)
search.fit(X_dev, y_dev)
```

The `step__parameter` syntax addresses a parameter inside a pipeline.

Audit this code

```
from sklearn.neighbors import KNeighborsClassifier

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)
classifier = KNeighborsClassifier()
classifier.fit(X_train, y_train)
print(classifier.score(X_test, y_test))
```

What information crossed the train/test boundary? How would you fix it?

Audit — preprocessing saw the test set

The global means and standard deviations depend on **all observations**, including the future test set.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

model = Pipeline([
    ("scaler", StandardScaler()),
    ("knn", KNeighborsClassifier()),
])
model.fit(X_train, y_train)
print(model.score(X_test, y_test))
```

The same principle applies to imputation, feature selection, PCA and target encoding.

Search is an experimental budget

Grid search evaluates a predefined Cartesian product.

Randomized search samples configurations from distributions and is often more efficient when only a few dimensions matter.

Adaptive optimization tools such as Optuna use previous trials to propose later ones.

All three can overfit validation data when the search is extensive. More trials do not create more information.

Selection creates optimism

Imagine 100 equally good configurations evaluated with noisy validation estimates.

The selected one is likely to be the candidate that benefited most from random variation.

Consequences:

- report the search space and number of trials;
- keep the final test separate;
- avoid “tuning” after seeing test performance;
- consider nested CV when data are limited and unbiased comparison is important.

Nested cross-validation

```
outer training set
└── inner CV: select hyperparameters

outer validation fold
└── evaluate the selected procedure
```

Repeat over outer folds. The outer scores estimate the performance of the **whole selection procedure**, not one configuration fitted once.

Nested CV is computationally expensive but separates model selection from model assessment.

Final refit and final test

After all choices are frozen:

```
best_pipeline = search.best_estimator_
test_predictions = best_pipeline.predict(X_test)
```

With the default `refit=True`, `GridSearchCV.fit` has already refitted the best pipeline on all development data.

Report:

- the prespecified primary metric and uncertainty;
- diagnostic metrics and subgroup results;
- the baseline under the same protocol;
- the complete selection procedure.

Audit this model-comparison claim

> “We tried six classifiers, several scalers, 200 hyperparameter settings and three metrics. Random forest obtained the best test F1, 0.84 instead of 0.82, so it is significantly better.”

List the unsupported steps in this conclusion.

Audit — several claims are missing evidence

- The test set appears to have selected the model.
- Trying three metrics invites post-hoc metric selection.
- No baseline or operational effect size is given.
- No sample size or uncertainty accompanies the 0.02 difference.
- “Significantly” requires a defined estimand and valid paired procedure.
- Group, time and duplication structure are unspecified.
- F1 depends on the selected threshold and positive-class definition.

A longer leaderboard does not repair an invalid protocol.

6. Variability, uncertainty and significance

A score is a statistic

The observed score depends on the particular sample used for evaluation.

Two complementary questions are useful:

Question	Output
Which values of the score are plausible?	confidence interval
Is the observed difference compatible with “no difference”?	hypothesis test

Neither question tells us whether the difference is useful in practice.

Reminder — the classical testing workflow

1. Define H_0, H_1 and the significance level α.
2. Choose a test statistic T before looking at the result.
3. Determine the distribution of T under H_0.
4. Compute the observed statistic t_{obs}.
5. Obtain the **p-value** from that reference distribution.
6. Compare it with the chosen significance level α.

The test and its assumptions must match the data-generating process.

What does the p-value mean?

$$p = P_{H_0}(\text{a result at least as extreme as } t_{obs})$$

- If $p < \alpha$, reject H_0 at level α.
- If $p \geq \alpha$, do **not** reject H_0.

The p-value is **not** the probability that H_0 is true.

Failure to reject H_0 is not proof that the models are equivalent.

McNemar's test for paired predictions

Intuition: if A and B are equally accurate, neither model should win much more often on the cases where they disagree.

Evaluate classifiers A and B on the **same independent test examples**:

	B correct	B wrong
A correct	both correct	n_{10}
A wrong	n_{01}	both wrong

$$H_0 : \; P(\text{A correct, B wrong}) = P(\text{A wrong, B correct})$$

Under H_0, the two types of disagreement should occur equally often.

McNemar's statistic

With the continuity correction:

$$\chi^2_{obs} = \frac{(|n_{10} - n_{01}| - 1)^2}{n_{10} + n_{01}}$$

Under H_0, this statistic is approximately distributed as a χ^2 with **one degree of freedom**, provided there are enough discordant cases.

For few disagreements, use the exact binomial version instead.

The test applies to paired **correct / incorrect outcomes**, not directly to AUC or F1.

McNemar example — from statistic to p-value

On the same 300 examples, suppose $n_{10} = 28$ and $n_{01} = 12$.

$$\chi^2_{obs} = \frac{(|28 - 12| - 1)^2}{28 + 12} = 5.625$$

From the χ^2_1 table:

Tail probability	0.05	0.025	0.01
Critical value	3.84	5.02	6.63

Thus $0.01 < p < 0.025$ ($p \approx 0.018$). At $\alpha = 5\%$, we reject H_0: A wins significantly more disagreements than B on this test sample.

What have we established?

The asymmetry between the two types of errors is unlikely under H_0.

We have **not** established that:

- A is sufficiently better for the application;
- the test population represents deployment;
- the result will reproduce on another sample;
- the experimental protocol is free of leakage.

Report the score difference and its practical scale alongside the test.

Confidence intervals — one compact option

A bootstrap interval can describe sampling uncertainty:

1. resample the independent test units with replacement;
2. recompute the score or the paired score difference;
3. repeat many times;
4. use the empirical distribution to form an interval.

Resample **patients**, not rows, when several rows belong to one patient.

SciPy: `bootstrap`

Why not a naive t-test on CV folds?

Fold scores are not independent because their training sets overlap.

Treating k fold differences as k independent observations underestimates uncertainty and violates the usual t-test assumptions.

For two classifiers evaluated on one untouched test set:

- McNemar tests paired correct / incorrect outcomes;
- a paired bootstrap can handle other score differences.

Statistical vs practical significance

Statistical conclusion	Practical question
Is the result surprising under H_0?	Is the effect large enough to matter?
Depends on effect and sample size	Depends on costs and constraints
Summarized by a p-value	Summarized by an effect size or utility

A 0.2% accuracy gain may be valuable at massive scale—or useless if it doubles latency.

A reporting template

> On an untouched test set of **N independent units**, model A obtained **metric = value**, 95% CI **[lower, upper]**. Against baseline B on the same cases, the paired difference was **Δ**, 95% CI **[lower, upper]**. This [does / does not] exceed the predeclared practically relevant threshold of **δ**.

Also report:

- data population and split strategy;
- metric definition and threshold;
- subgroup results;
- search space and seeds;
- known limitations.

Reproducibility checklist

- Fix and record random seeds where supported.
- Save the exact data partitions or group identifiers.
- Version data, code and dependencies.
- Keep preprocessing and predictor in one pipeline.
- Record metric definitions, positive class and averaging.
- Store all candidate results, not only the winner.
- Separate exploratory analysis from confirmatory evaluation.
- Make the final report regenerable from raw results.

Reproducibility makes mistakes auditable; it does not guarantee validity.

Design a credible comparison

You have 2,000 labelled samples from 120 patients and want to compare logistic regression with an SVM.

In pairs, specify:

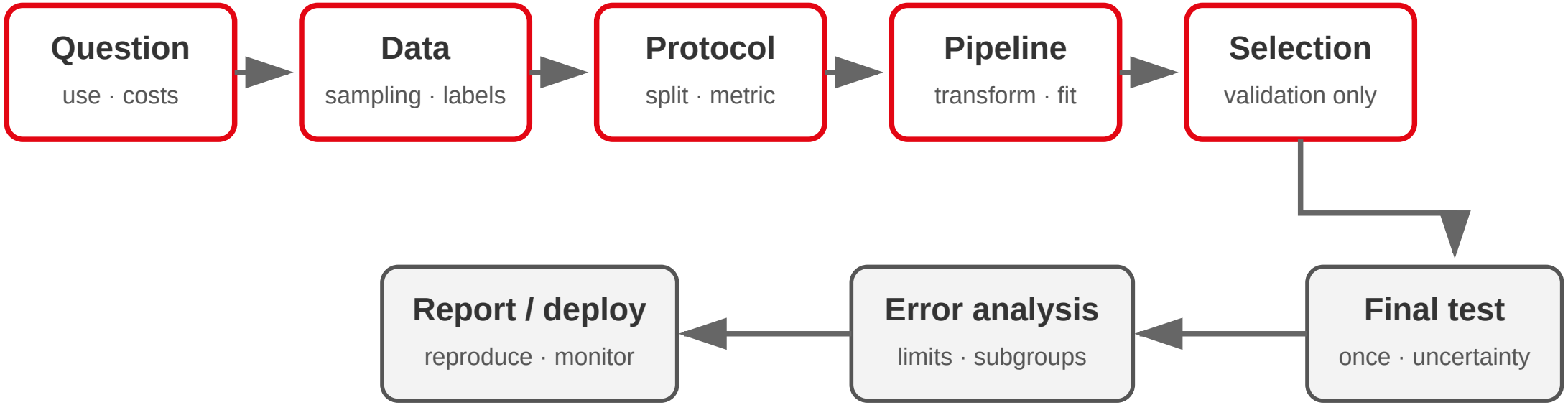
1. the outer test design;
2. the cross-validation splitter;
3. one primary metric;
4. a baseline;
5. the hyperparameter-selection protocol;
6. how you would quantify the final difference.

One defensible design

- Hold out complete patients as a final test set.
- Use group-aware CV on development patients.
- Choose the metric from the use case; for imbalance, perhaps balanced accuracy or recall at constrained specificity.
- Include a dummy and a simple clinical rule if available.
- Tune each complete pipeline only inside development CV, with the same search budget.
- Refit on development data and predict the untouched patient-level test set once.
- Bootstrap **patients** jointly for a confidence interval on the paired metric difference.

Other answers are valid if their assumptions match deployment.

The complete protocol



Every arrow represents a decision that must be justified and documented.

What to remember

1. Define the population, prediction time and metric before model comparison.
2. Keep the test set outside every development decision.
3. Fit all learned transformations inside a pipeline.
4. Split by the true independent unit and respect time.
5. Audit features for leakage and shortcuts.
6. Compare against a baseline under the same protocol.
7. Report effect sizes, uncertainty and practical relevance.

Practical sessions

Evaluation audit

- detect leakage and partitioning errors;
- repair the workflow with pipelines and suitable splitters;
- compare misleading and task-relevant metrics.

First complete learning study

- establish a baseline;
- select hyperparameters using cross-validation;
- perform one final evaluation with uncertainty;
- produce a concise, reproducible report.

Further reading

- Raschka, Liu & Mirjalili, *Machine Learning with PyTorch and Scikit-Learn*, Chapter 6, 2022 — local course resource.
- scikit-learn — Cross-validation: evaluating estimator performance
- scikit-learn — Model selection and evaluation
- scikit-learn — Metrics and scoring
- Oakden-Rayner et al. — Hidden stratification in medical imaging
- Demšar — Statistical Comparisons of Classifiers over Multiple Data Sets

Exit ticket

Write one sentence for each question:

1. What is the most important rule for the test set?
2. Which leakage risk had you not considered before?
3. What must be stated next to any reported score?
4. Which point should we revisit during the practical sessions?

Questions?

Next: applying the evaluation protocol in scikit-learn