Building ML models from tabular data

Marco Visentini-Scarzanella

Sr. Manager, Applied Science

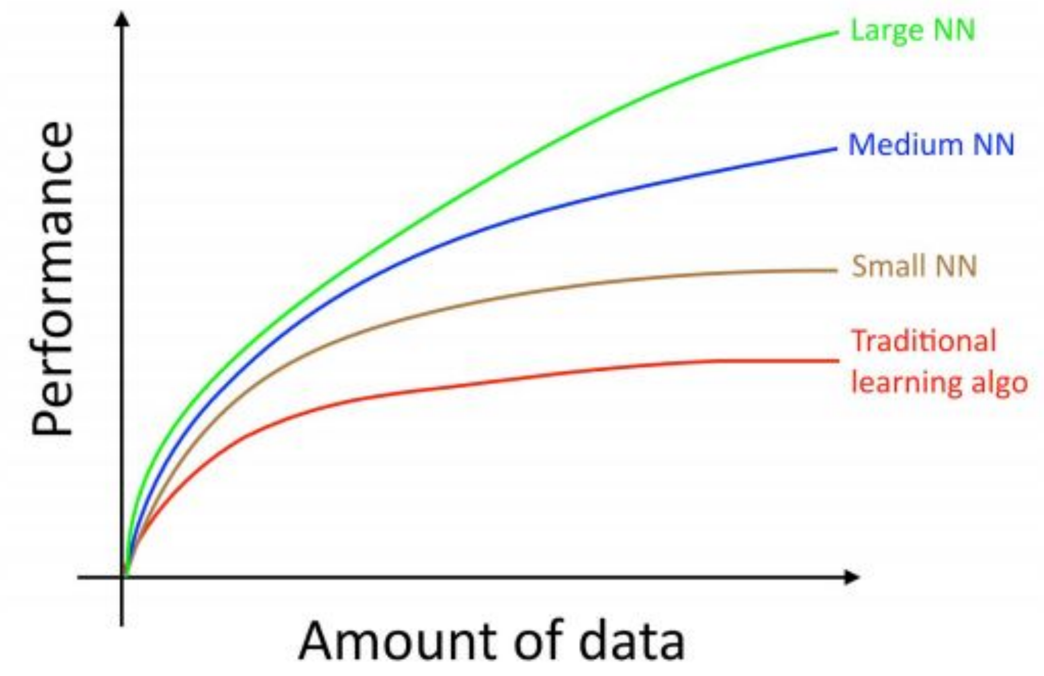
Amazon Japan

A formal definition of ML

"A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*."

Mitchell, 1997

- Now substitute…
  - T with a problem definition
  - P with a cost function
  - E with training data



No performance measure -> it's **analytics**

No learning from experience -> it's **optimization**

1. What is tabular data?

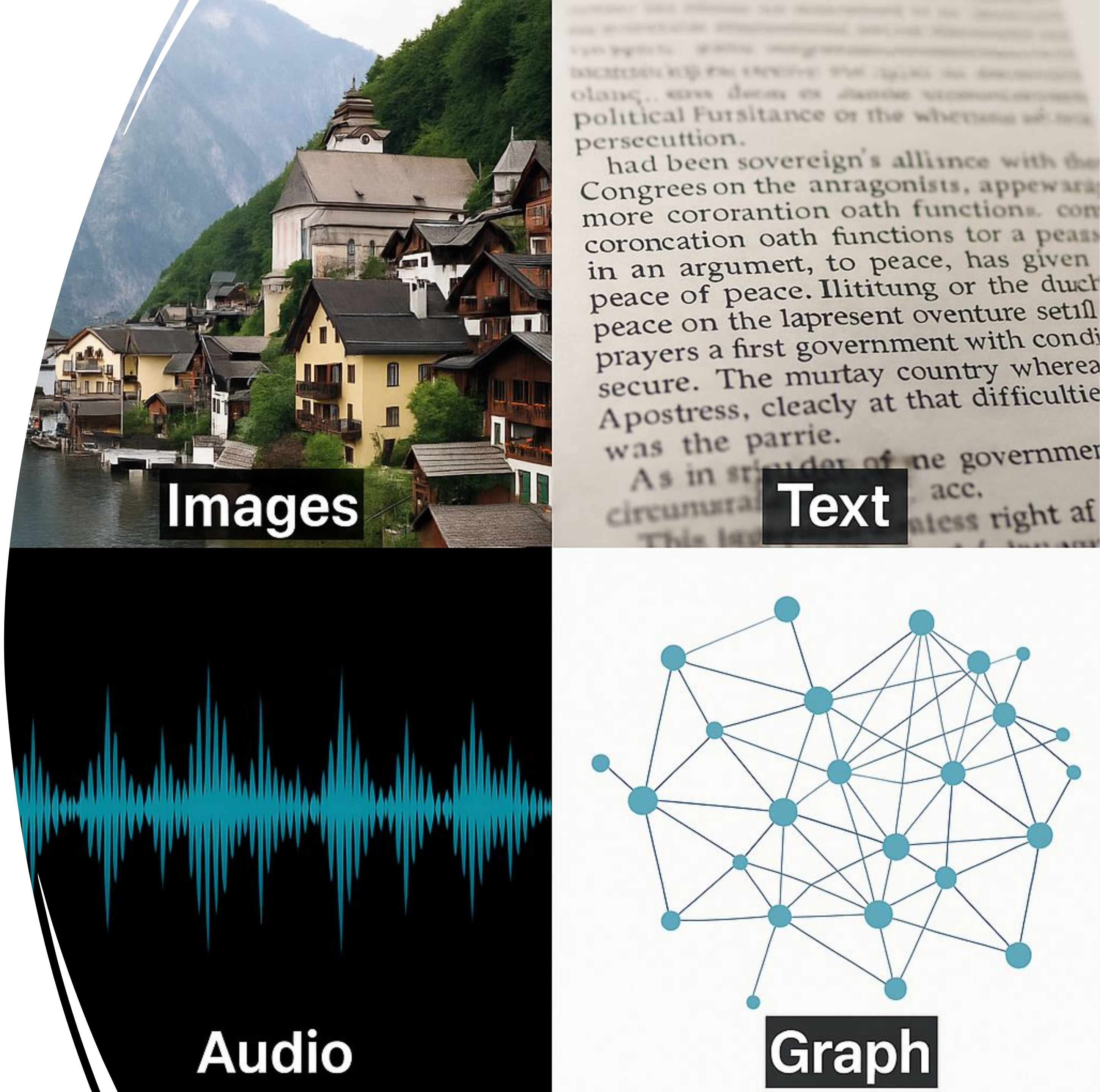
What is tabular data?

Name	Age	Income		Target
Alice	25	45,000		**Yes**
Bob	32	60,000		**No**
Carol	37	52,000		**No**
David	29	70,000		**Yes**
Eve	45	38,000		**No**

- Anything that can be represented as a **flat table**
  - Rows = instances (e.g. people, transactions)
  - Columns = features (e.g. age, income)
  - Often includes a target column (what we are trying to predict)

What is *not* tabular data?

- **Images:**
  - Pixels arranged in grids, not tables
  - Example: photos, X-ray images
- **Text (Unstructured):**
  - Raw text without clear structure
  - Example: books, articles, customer reviews
- **Audio & Video:**
  - Continuous streams of sound or images
  - Example: voice recordings, movie clips
- **Graph Data (Networks):**
  - Nodes connected by edges, relationships matter
  - Example: social networks, web links

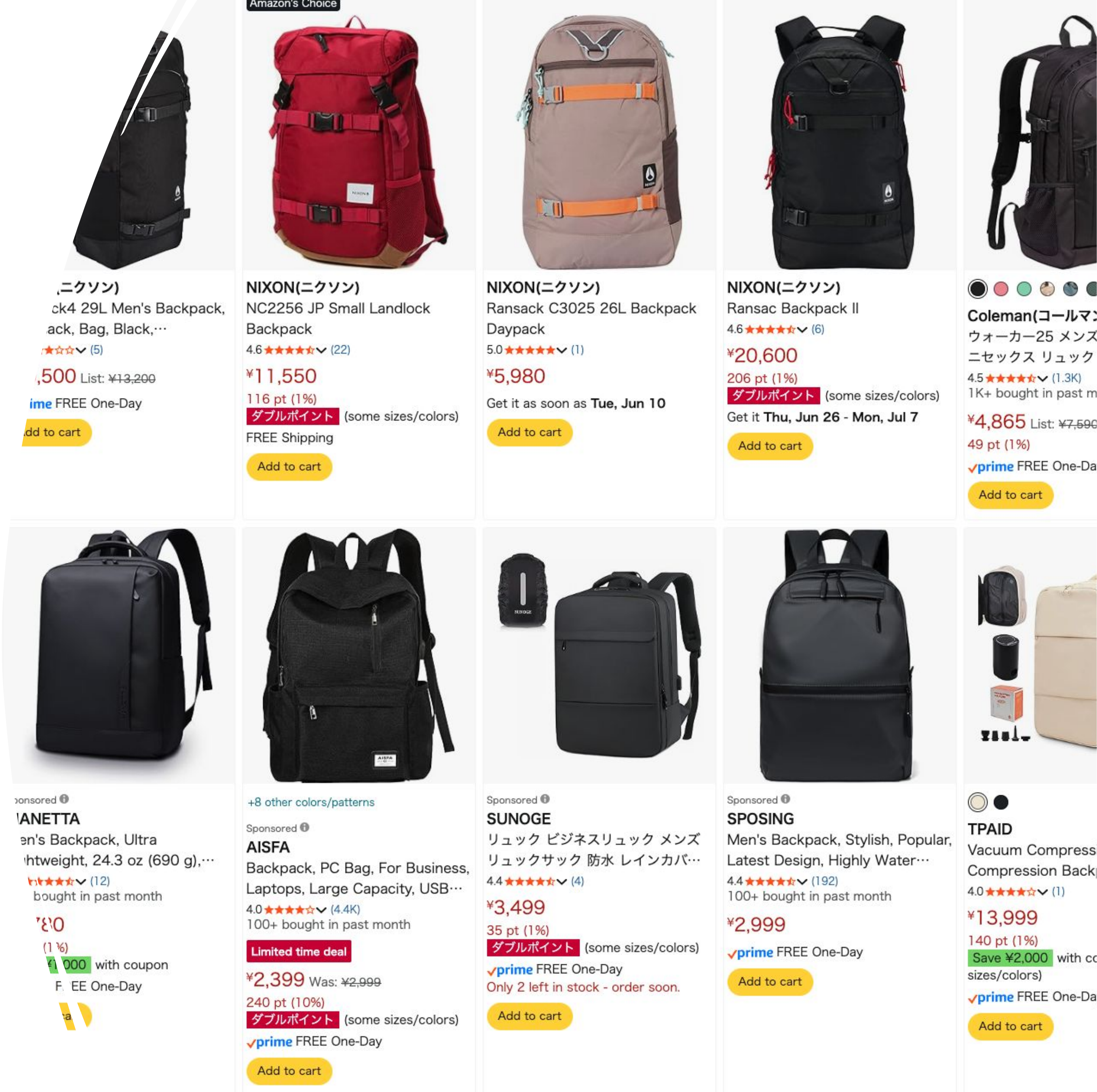


Why is tabular data important in ML?

- Ubiquitous in business and government
  - Easily >90% of models we build in Amazon are based on tabular data.
- Examples:
  - Finance: fraud detection
  - Healthcare: diagnostics from clinical imaging
  - Marketing: lifetime value prediction
  - E-commerce: recommendations

Amazon-specific applications

- Search results - *ranking*
- Amazon's choice badge - *classification*
- Double points badge – *regression / classification*
- Price and points – *regression*
- Title - *LLM generative processing + regression*
- Sponsored revenue - *regression*

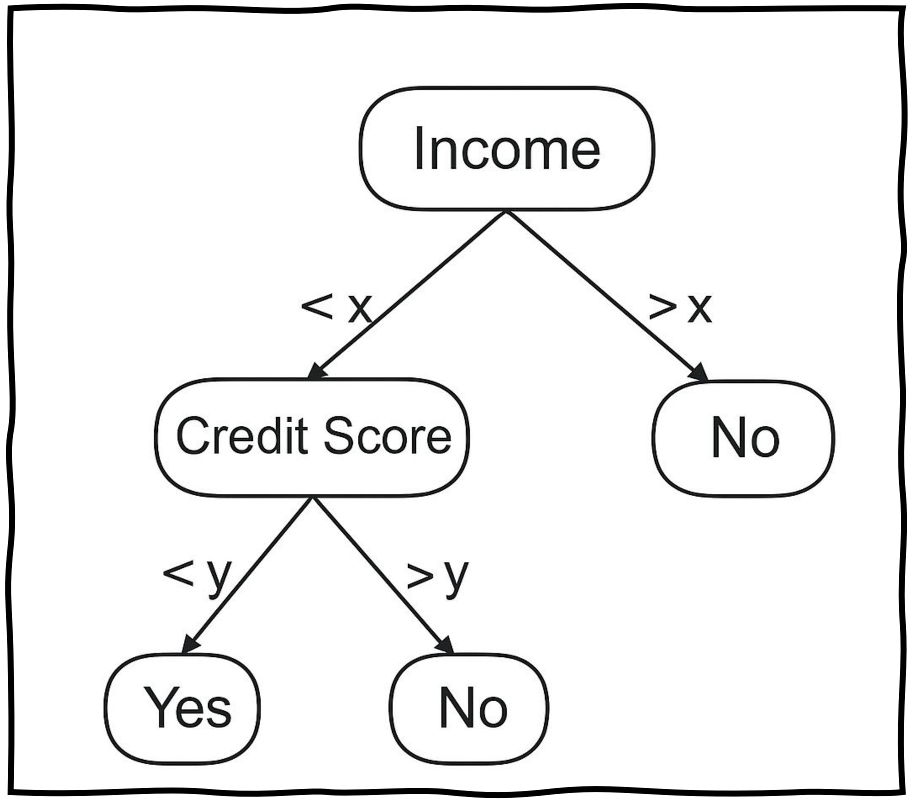


Tabular ML problem types

1. **Classification**
   - Output: a **discrete label** (e.g. Yes/No, category)
   - Examples:
     - Will the customer cancel? *(Yes/No)*
     - What product category is this? *(T-shirt, Pants, Jacket)*
2. **Regression**
   - Output: a **continuous number**
   - Examples:
     - What will sales be next month?
     - How much will the house sell for?
3. **Ranking / Scoring** *(special case)*
   - Output: a **score** used to sort or prioritize
   - Example: Which products should appear first in search results?

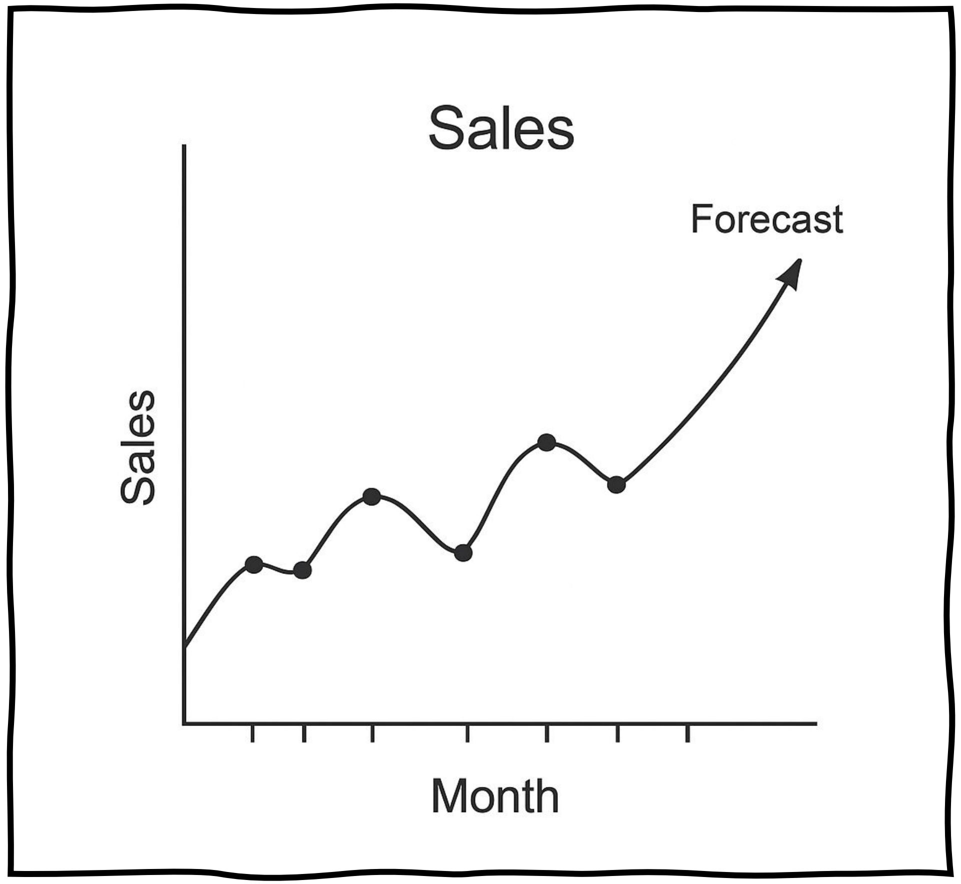
ML Task Type: Classification

- Output: a **category or label**
- Examples:
  - Will an applicant default on a loan? *(Yes / No)*
  - Is this email spam? *(Spam / Not Spam)*
  - Will a customer click on a recommendation? *(Yes / No)*
  - What age group is this customer (0-18, 18-35, 35-50, 50+)



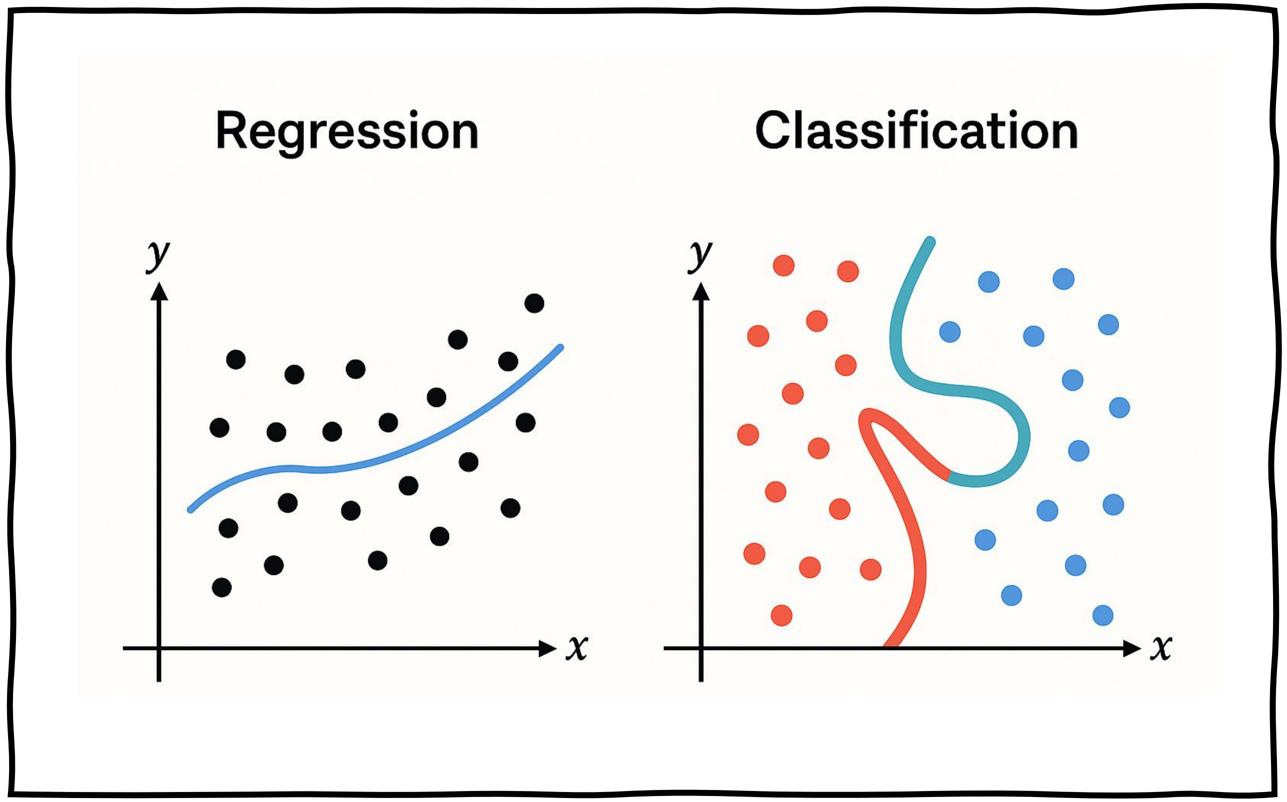
ML Task Type: Regression

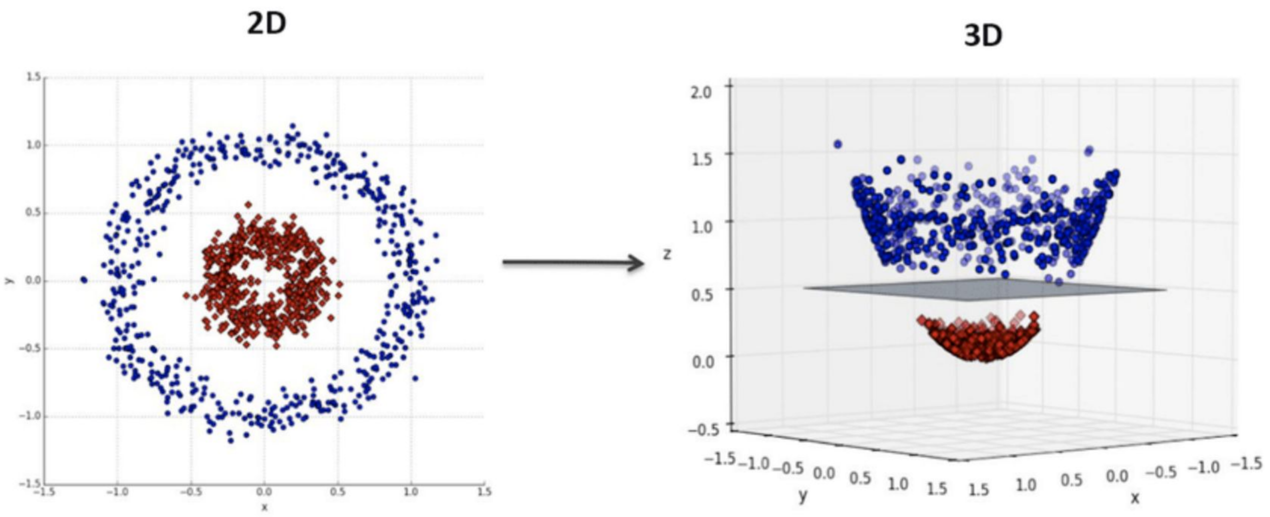
- Output: a **continuous value**
- Examples:
  - Predicting customer spending
  - Estimating exam scores
  - Forecasting next month's sales

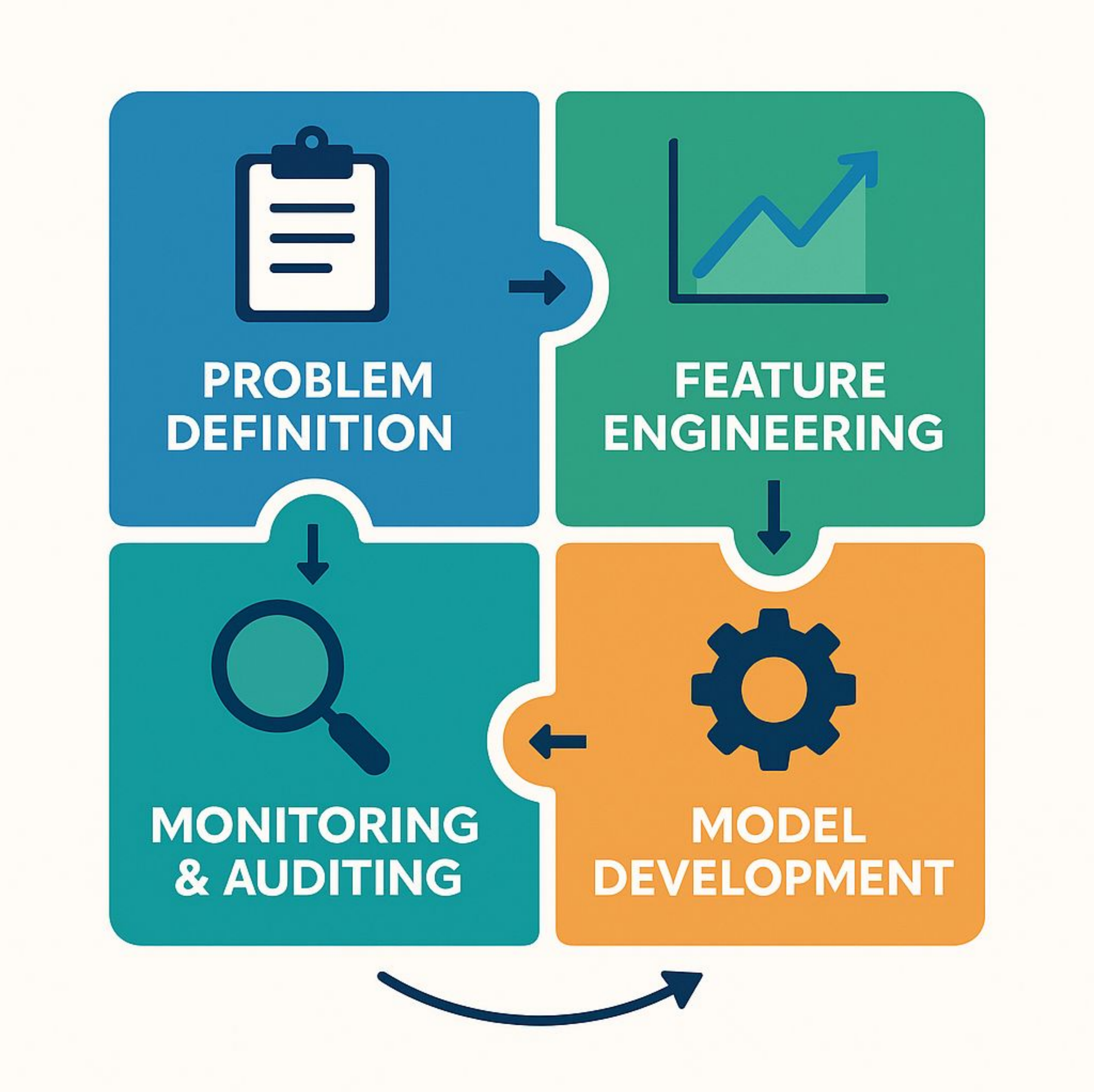


A unifying view of tabular ML

- **Tabular ML = curve fitting**
- **What makes a problem "easy" to solve with ML?**
- We want to **transform features** so the task becomes linearly separable (for classification) or linearly predictable (for regression).
- This can happen through:
  - **Explicit Feature Engineering** (e.g., log, ratios, date parts)
  - **Nonlinear mapping** of features to spaces where they become linearly separable (e.g. NN)
  - **Nonlinear curve fitting** to fit very complex functions or decision boundaries







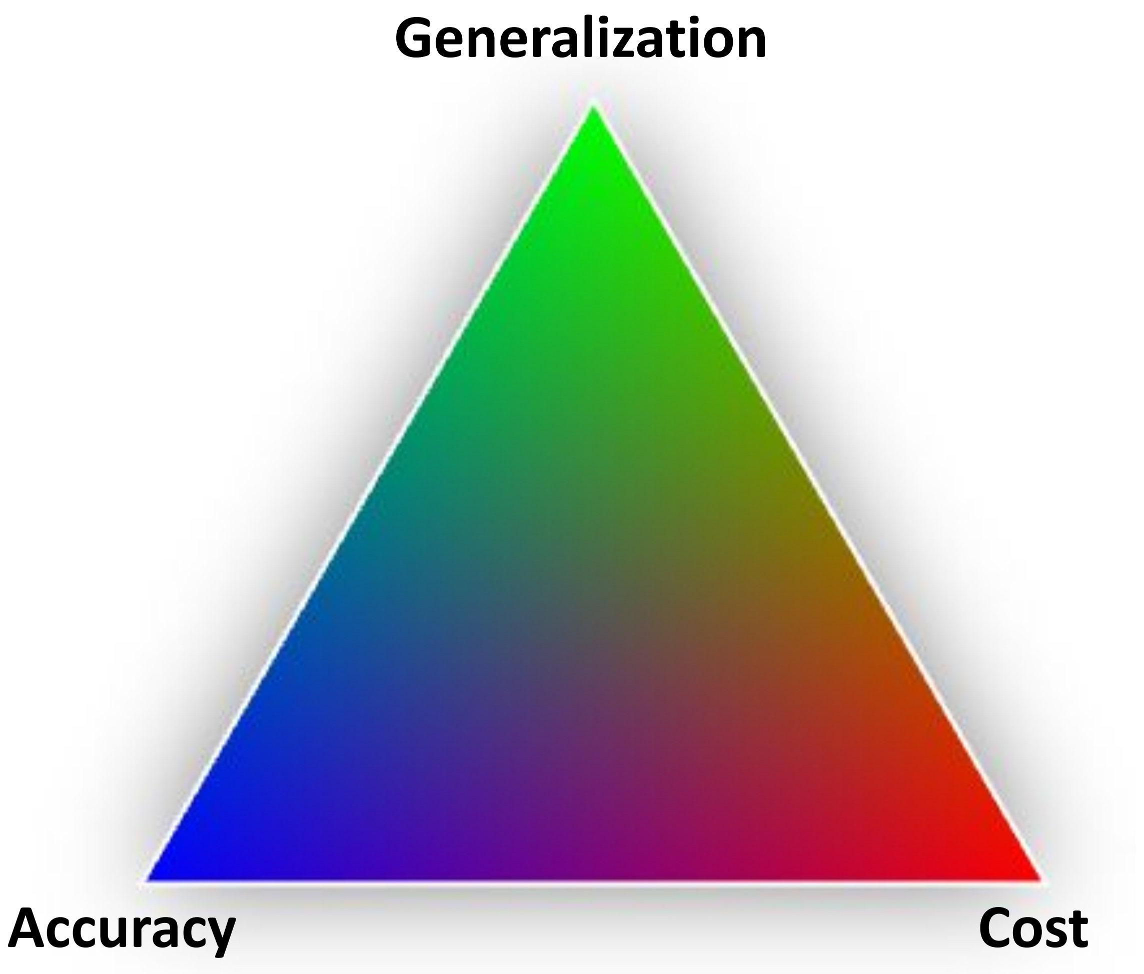
Real ML workflows

- Majority of ML applications:
  - Batch **load** → Batch **train** → Batch **predict** (scheduled regularly)
  - Frontend simply consumes predictions from dataset
- Increasingly commoditized stage:
  - **Model Training & Deployment** *("boilerplate")*
- Crucial strategic stages:
  - **Problem Definition** → **Requirements** *("Are we solving the right problem?")*
  - **Feature Engineering** *(GIGO-Law: Garbage-In, Garbage-Out)*
  - **Monitoring & Auditing** *(Is it still performing as intended?)*

2. Model family selection

Model family selection

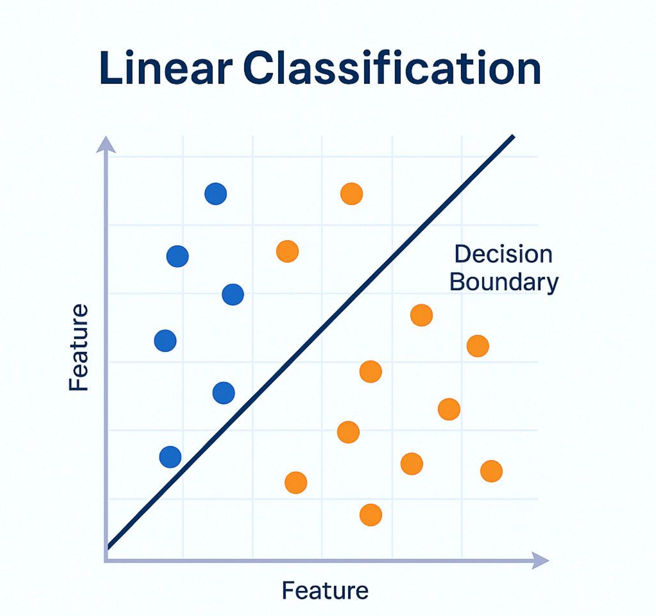
- Overview of main algorithm families for tabular data
- Strengths, weaknesses, and when to pick each
- Why it matters:
  - Different models suit different problems
  - **No Free Lunch Theorem:** no model is best for all problems
  - Tradeoffs: accuracy vs. generalization vs. cost



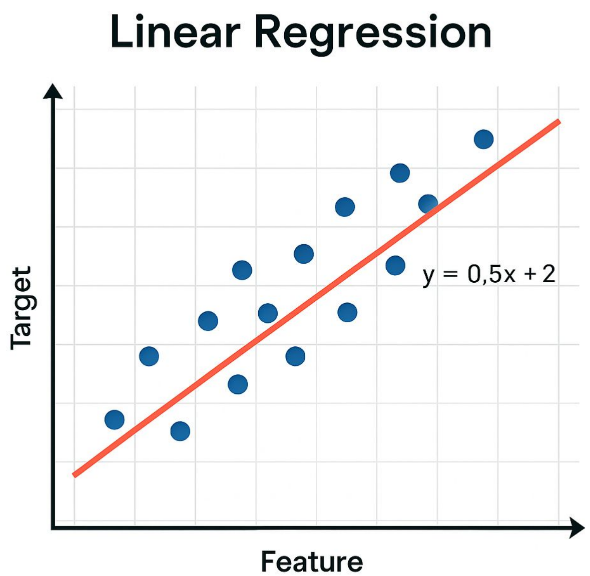
Linear & Logistic Regression

- **How they work:** assume linear relationship between features and target
- **Pros:** interpretable coefficients, fast to train, simple
- **Cons:** struggle with complex, non-linear patterns
- **When to use:** small-medium datasets, interpretability important, baseline checks

- Logistic regression
  - $P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$



- Linear regression
  - $y = \mathbf{w}^\top \mathbf{x} + b$



Tree-based models

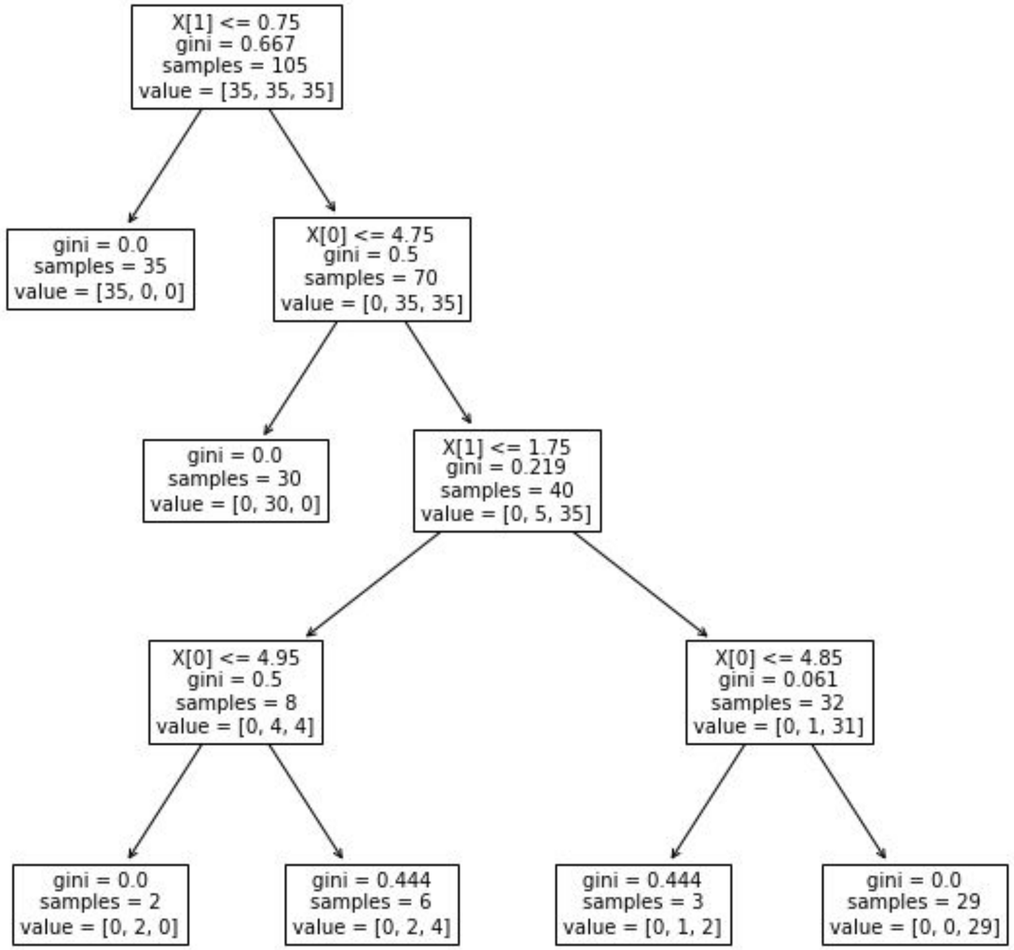
- **Decision Trees:** partition data into decisions (interpretable, risk of overfit)
- **Random Forests:** multiple trees, robust and accurate, less interpretability
- **Gradient Boosting (XGBoost, LightGBM):** builds models sequentially, highly accurate, slightly less transparent

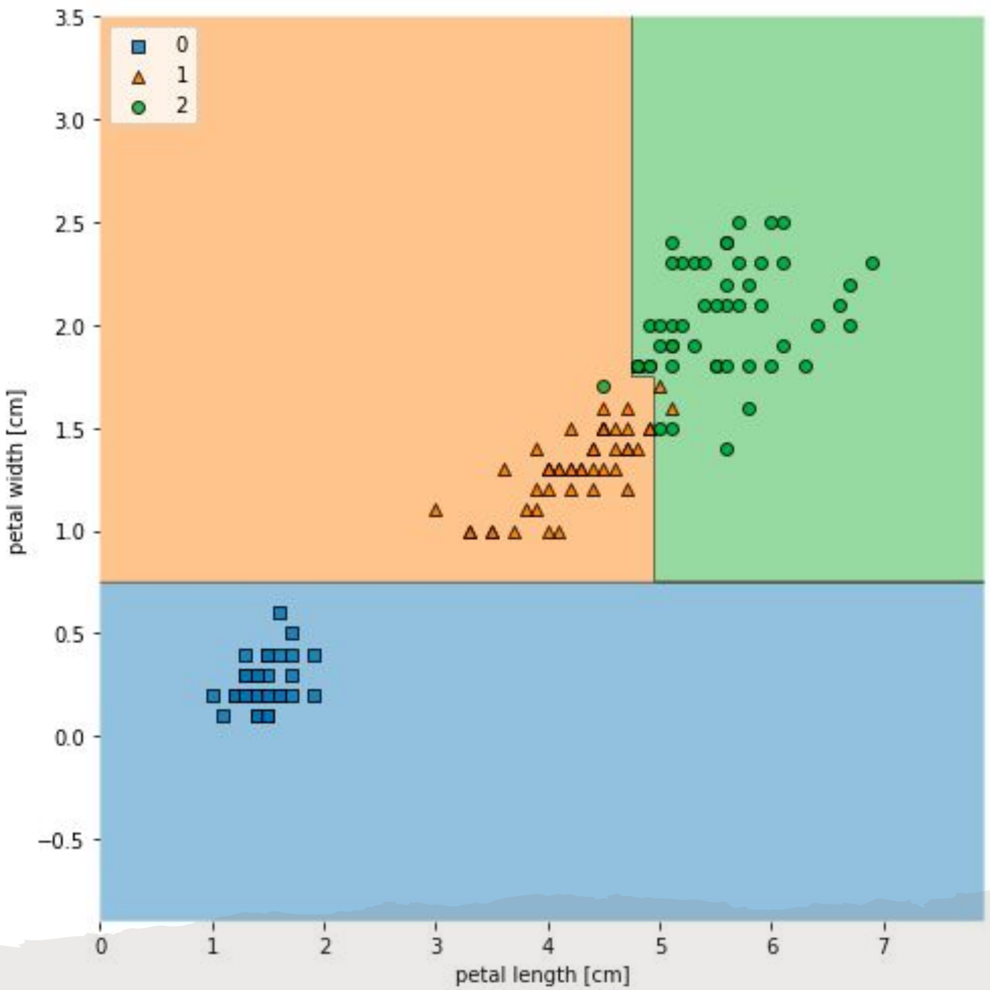
Decision trees

1. Split data recursively by feature values
2. At each node, choose the feature & threshold that best separates target values
3. Continue until stopping criteria (max depth, purity, min samples, etc.)

```
function build_tree(data):
    if stopping_criteria(data):
        return leaf_node(prediction)
    best_split = find_best_split(data)
    left_data, right_data = split(data, best_split)
    return node(
        condition = best_split,
        left = build_tree(left_data),
        right = build_tree(right_data)
)
```

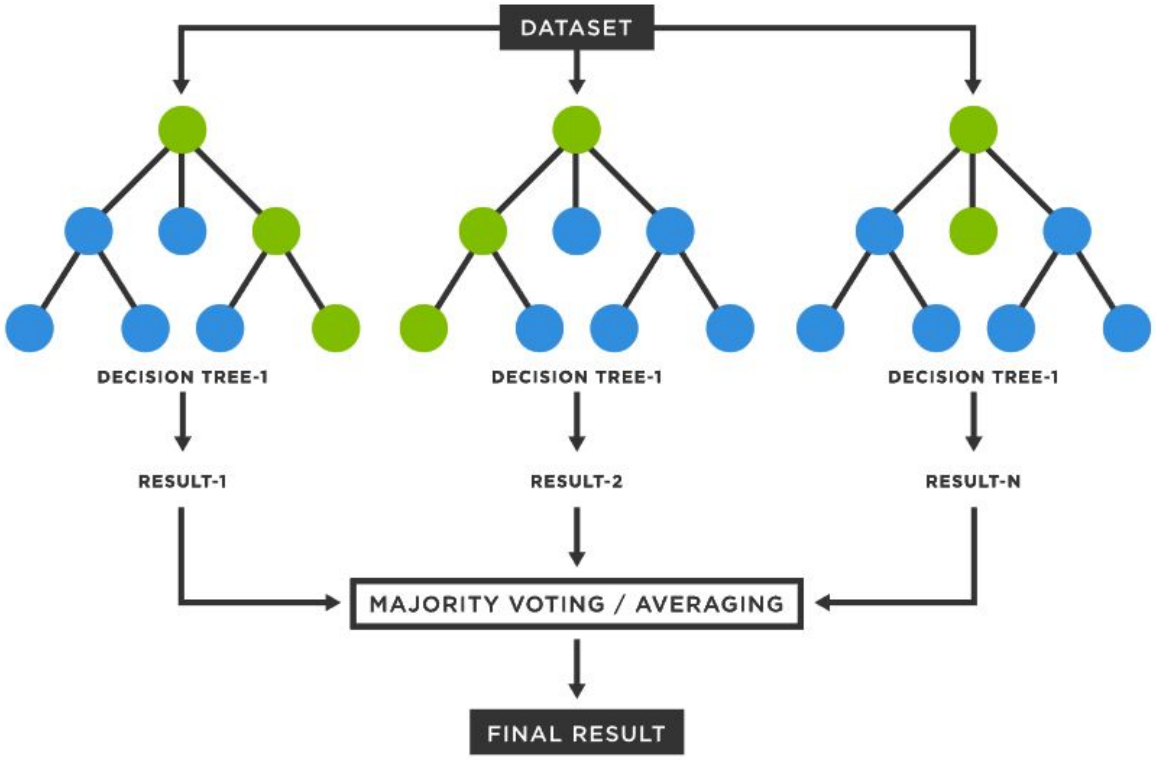
Decision trees in action



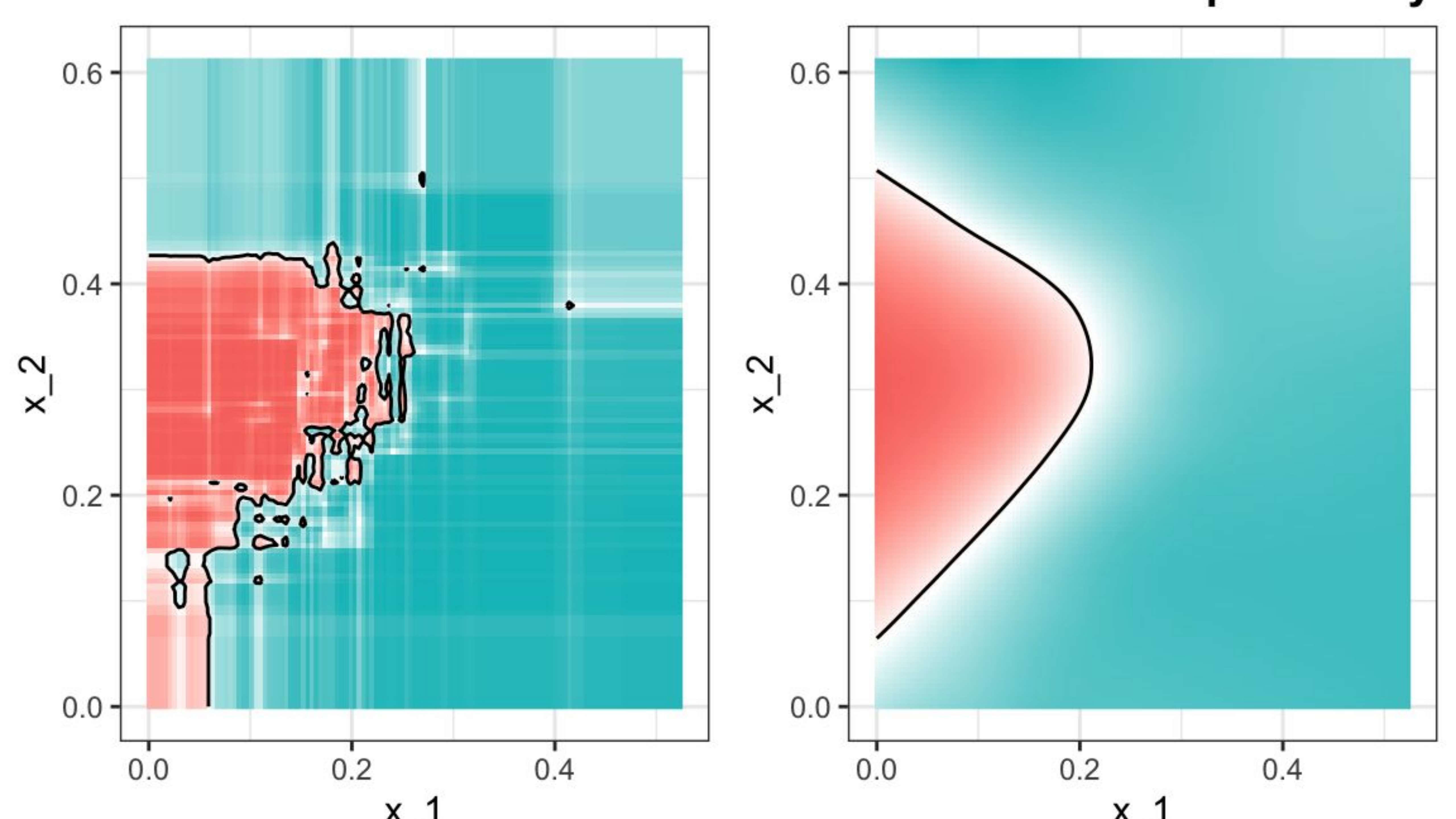


Random forests

- Forest = lots of trees
- Each tree uses a **random subset of features** at each split
- Final prediction:
    - **Classification:** majority vote
    - **Regression:** average of predictions
- Bagging (***b****ootstrap* ***agg****regat****ing***): ensemble multiple weak learners trained on different features
    - Increases robustness as it will not be dominated by outliers




```
function random_forest_predict(X):
   trees = [build_tree(sample_data()) for _ in range(N)]
   predictions = [tree.predict(X) for tree in trees]
   return aggregate(predictions)  # vote or average
```

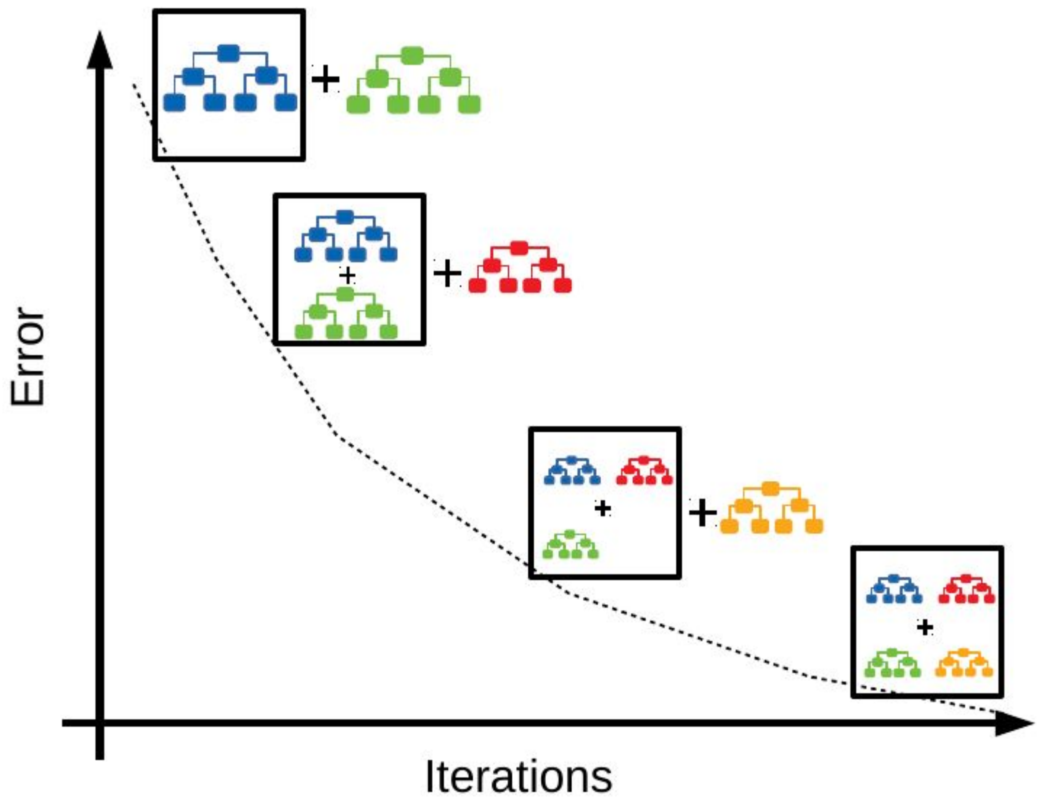


0.6
0.4
0.2
0.0
x_2
0.0
0.2
0.4
x_1

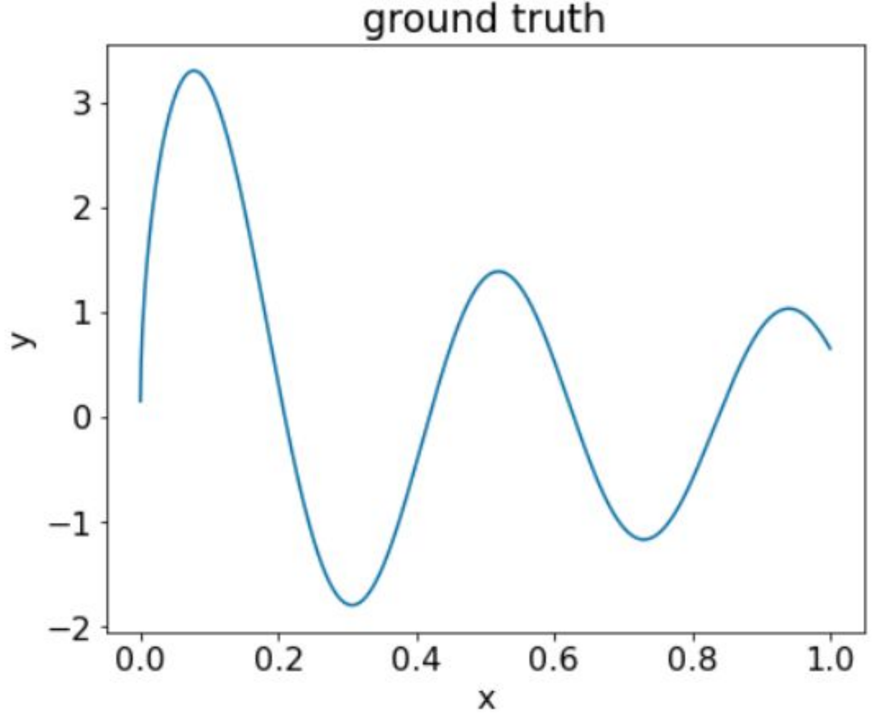
Gradient boosted trees

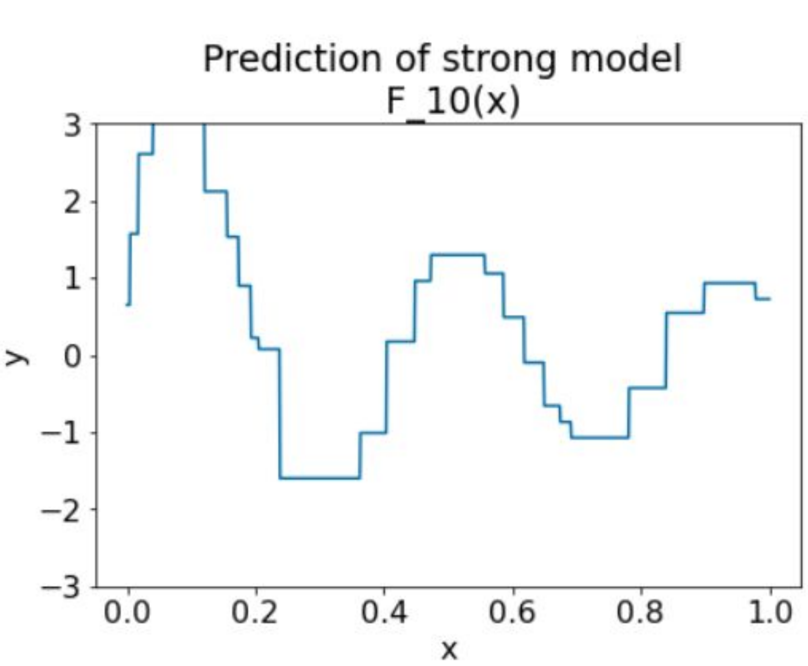
- Build trees sequentially — each one corrects the previous model's errors
- Final prediction is a **weighted sum** of weak learners
- Common libraries: XGBoost, LightGBM, CatBoost

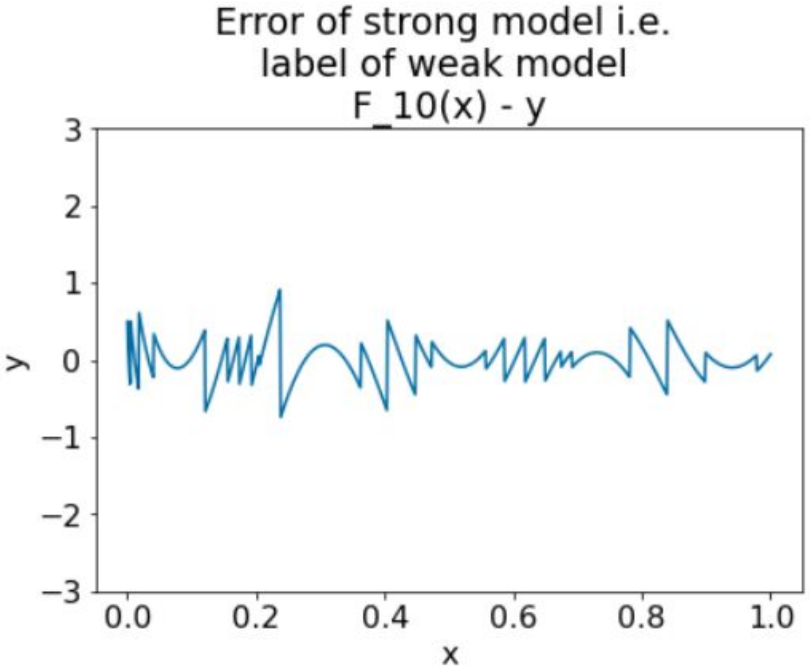
```
F0 = constant_prediction()
for m in 1 to M:
   residuals = compute_gradient_loss(y, Fm-1)
   tree = train_tree(X, residuals)
   Fm = Fm-1 + learning_rate * tree.predict(X)
return Fm
```

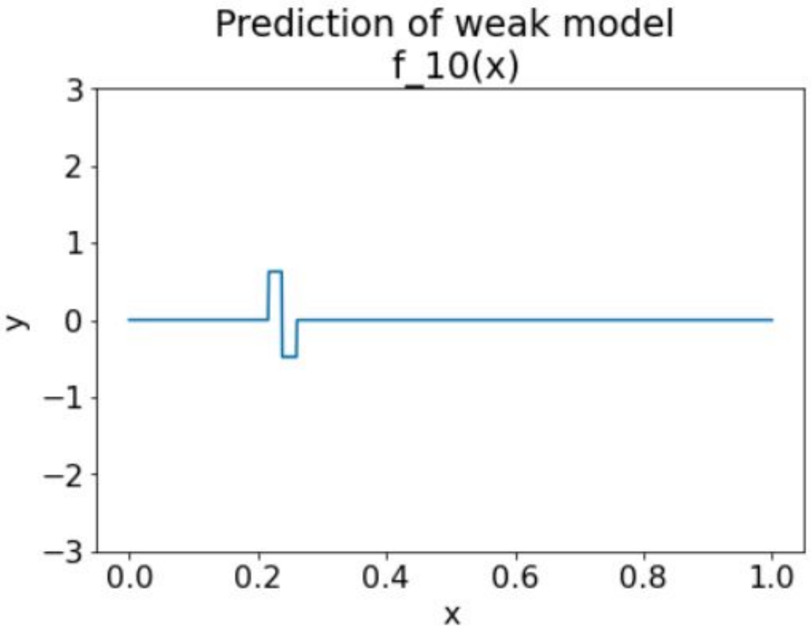


Gradient boosted iterations

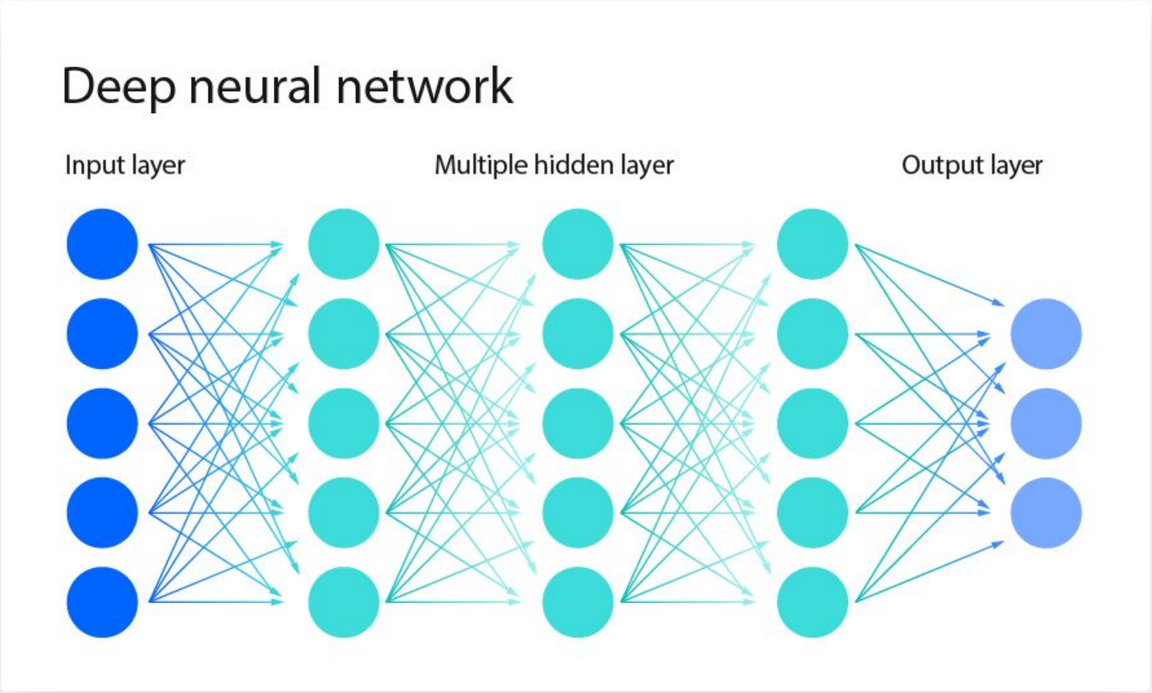






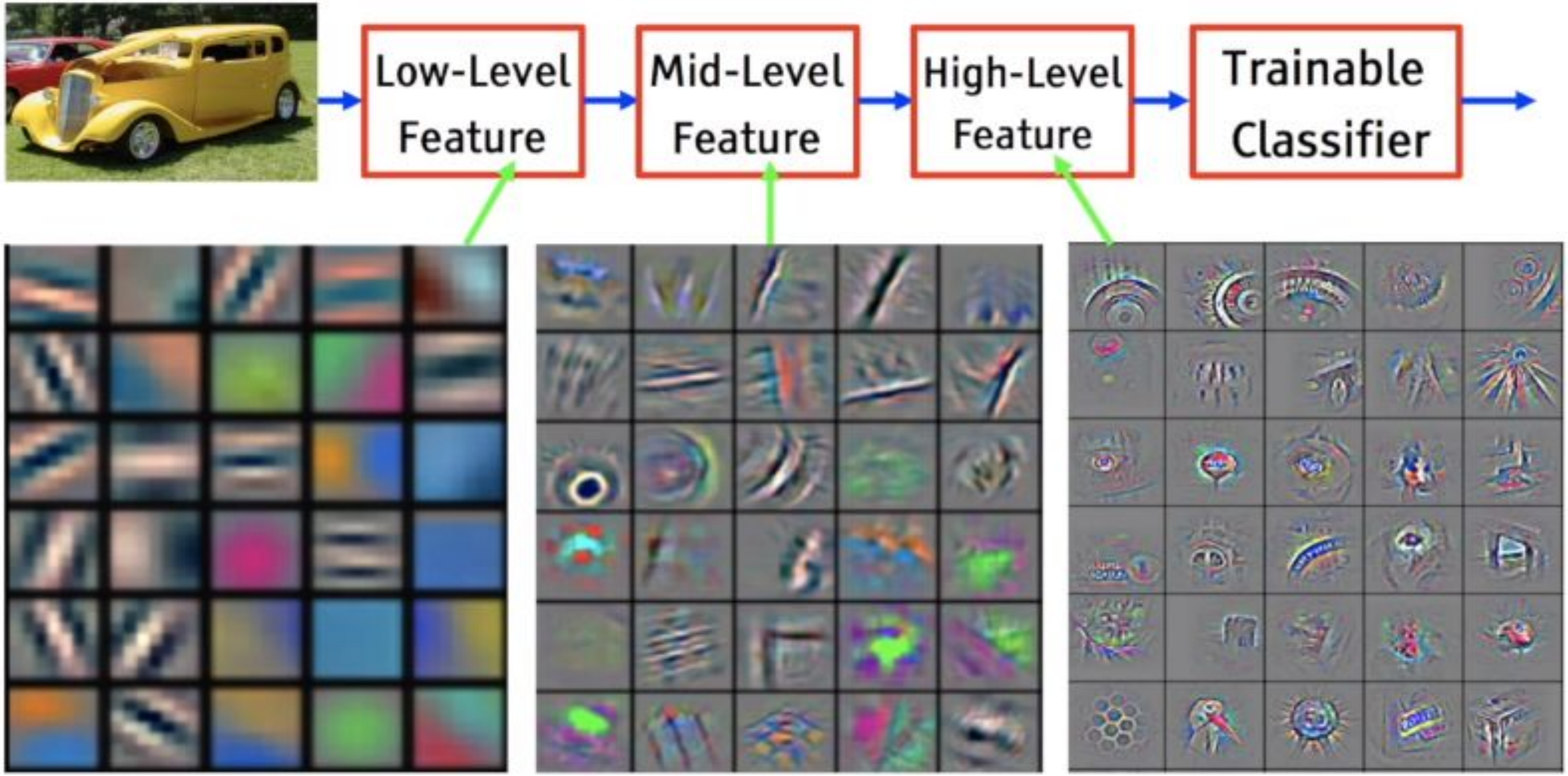


Neural networks



- **Neural Networks (NNs):** layers of interconnected nodes learn complex patterns
  - Pros: flexible, captures complex relationships
  - Cons: data-intensive, less interpretability
- Nonlinear transformation of all elements at each layer. Output learns a very nonlinear mapping of the inputs $y = f(\mathbf{x})$

Nonlinear feature mapping in action



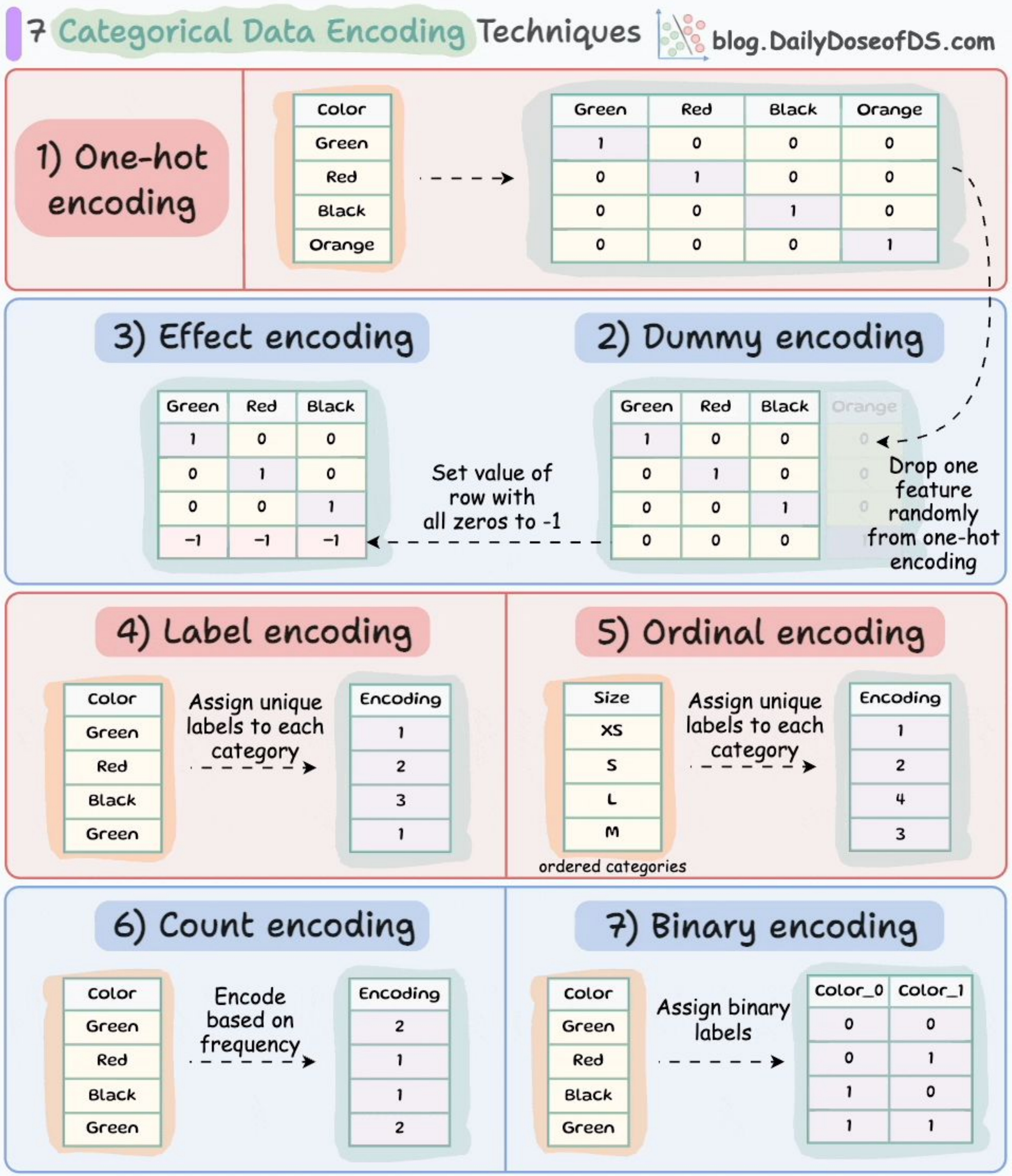
3. Data preprocessing

Data preprocessing / feature engineering

1. Encode data to tabular
2. Handle missing values
3. Scale, normalize, and remove outliers
4. Engineer new features… *or remove redundant ones*

Encoding non-tabular data

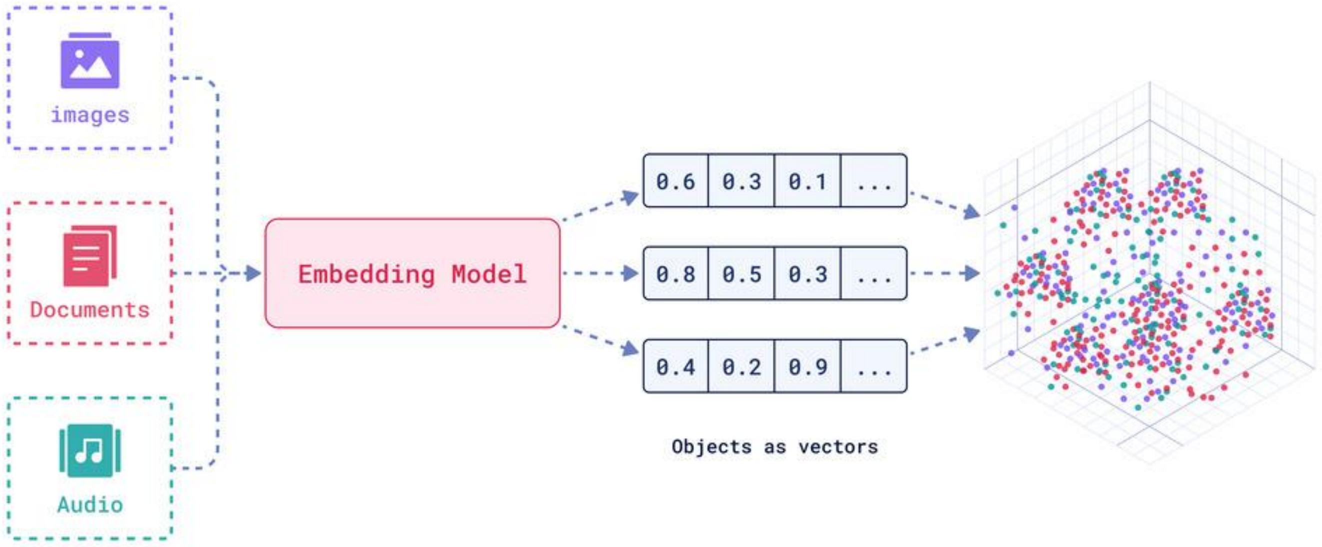
- Categorical features
  - One-hot: for nominal categories
  - Ordinal/label: for true rank order
    - E.g. low/medium/high
  - Watch out!
    - label encoding on nominal features can mislead models

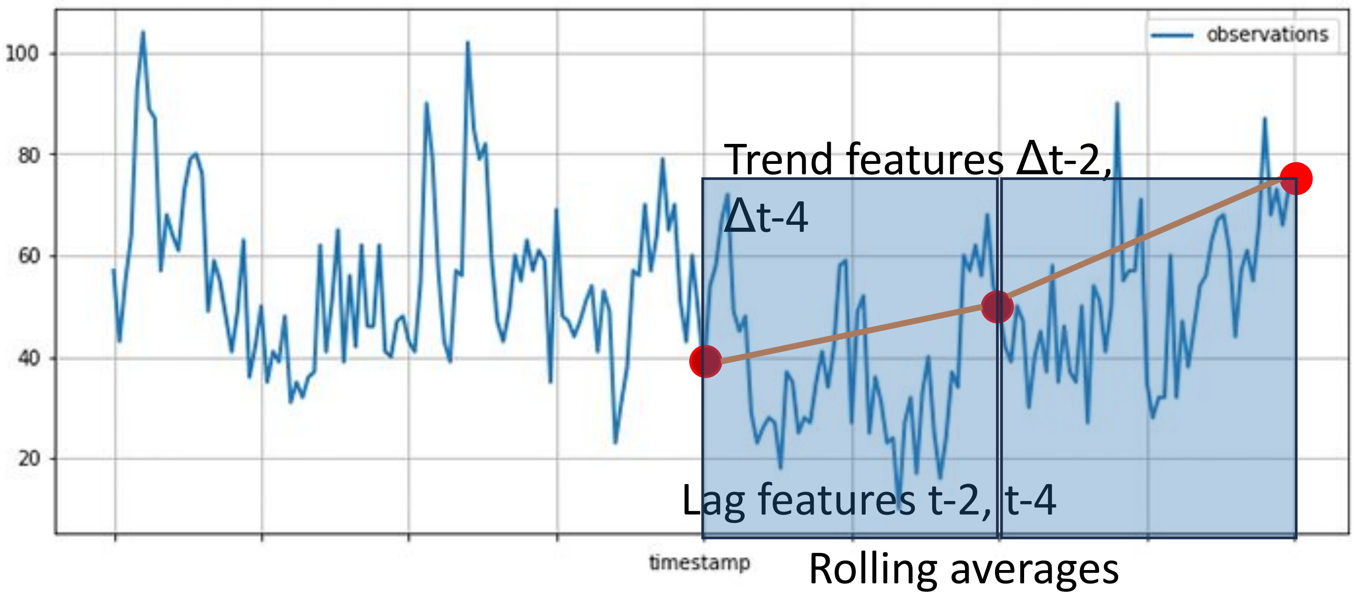


Encoding non-tabular data (advanced)

- Text, images
  - Bag-of-words/features
  - Embeddings

- Time series
  - Lag features
  - Rolling statistics





Handling missing values

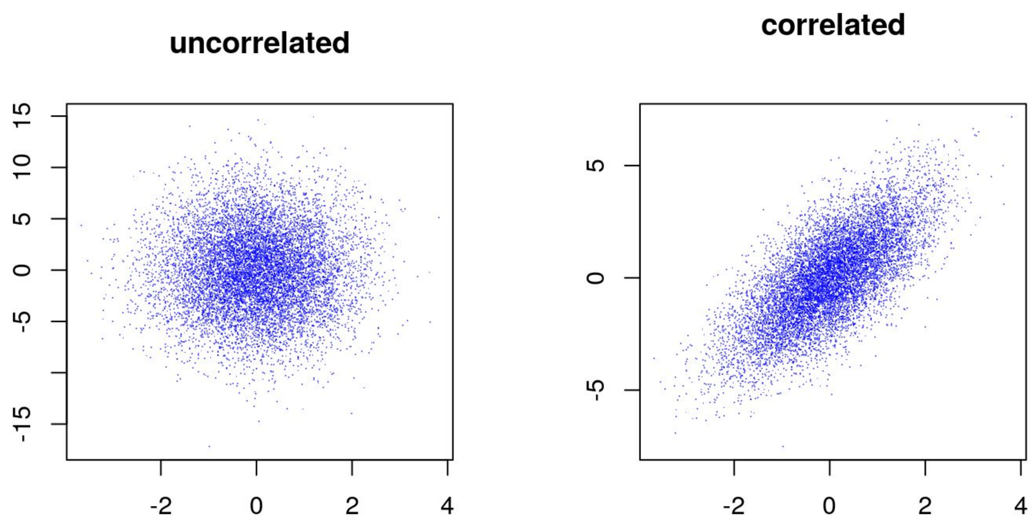
Populate from closest neighbour

id	x_1	x_2	x_3	x_4
1	London	57	B	10.1
2	Paris	42	C	31.5
3	Tokyo	55	C	8.4
4	London	51	B	
5			A	
6	Paris		A	45.0

Fill with column average

Too many missing values - drop

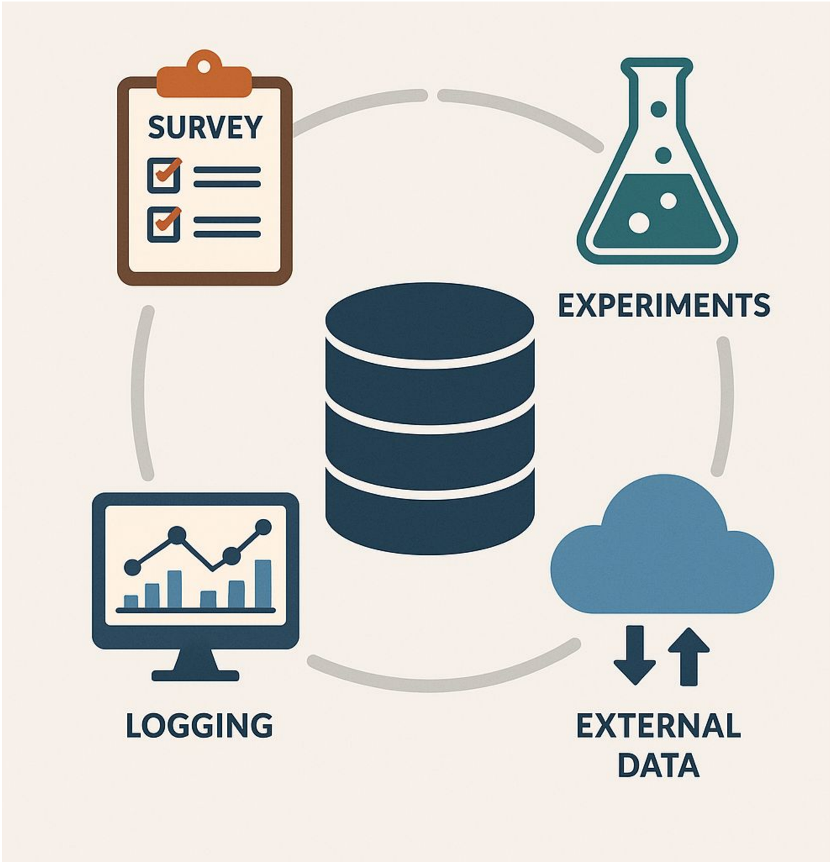
Feature engineering



- Feature engineering enables you to create your own nonlinear dependencies
  - E.g. loan screening: given “debt” and “income” features, ideal model would assess based on debt/income ratio.
  - Useless for models that do nonlinear mapping (e.g. NN)
  - Can reduce the depth needed by models that don’t (e.g. trees/forests)
- More features != better models
  - What is the *actual information* that additional features bring?
  - Only include minimum set of features that contribute to the prediction
    - **Can you think of why?**
  - **Ideal world**: no cross correlation between features

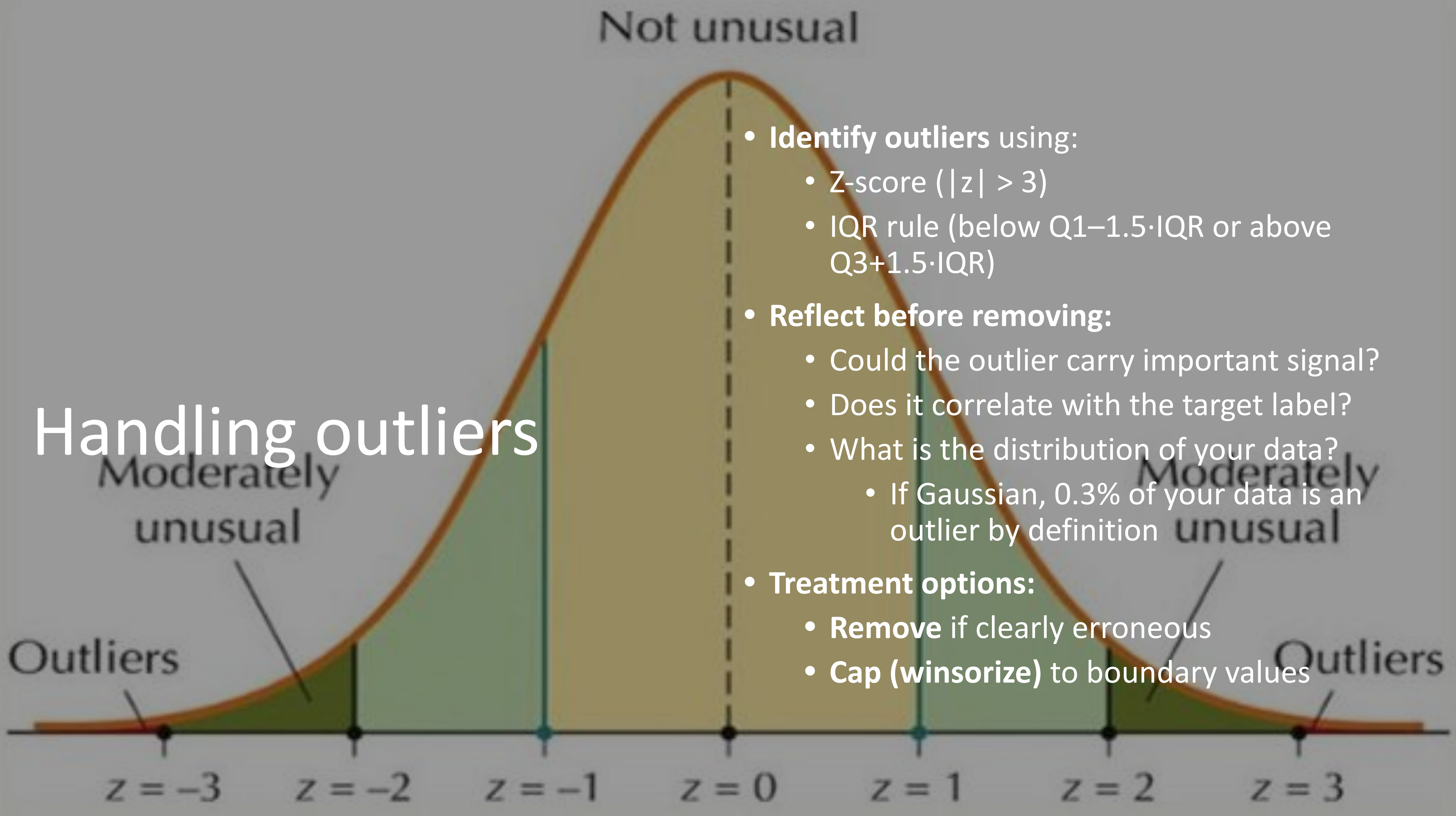
Feature engineering – beyond engineering

- Most times we don't have the features that we need
- Feature engineering only transforms existing features
- Be a data leader - acquire what you need!
  - **Direct customer surveys** – capture attitudes, intents
  - **A/B tests & experiments** – observe behavior under controlled conditions
  - **Instrumentation & logs** – add tracking in your app or website
  - **External sources & APIs** – enrich with weather, demographics, market data



Handling outliers

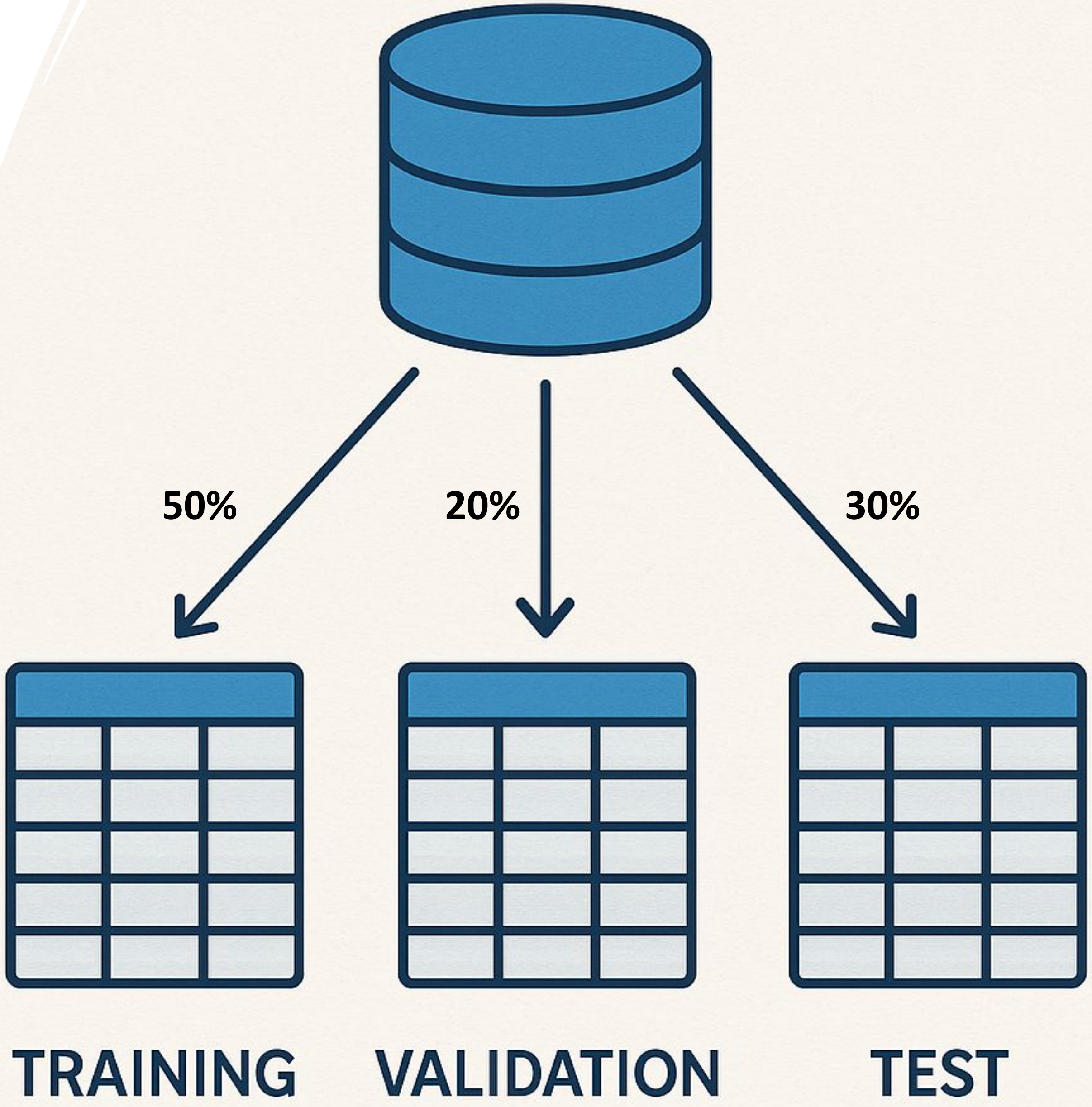
- **Identify outliers** using:
  - Z-score (|z| > 3)
  - IQR rule (below Q1–1.5·IQR or above Q3+1.5·IQR)
- **Reflect before removing:**
  - Could the outlier carry important signal?
  - Does it correlate with the target label?
  - What is the distribution of your data?
    - If Gaussian, 0.3% of your data is an outlier by definition
- **Treatment options:**
  - **Remove** if clearly erroneous
  - **Cap (winsorize)** to boundary values



4. Model training

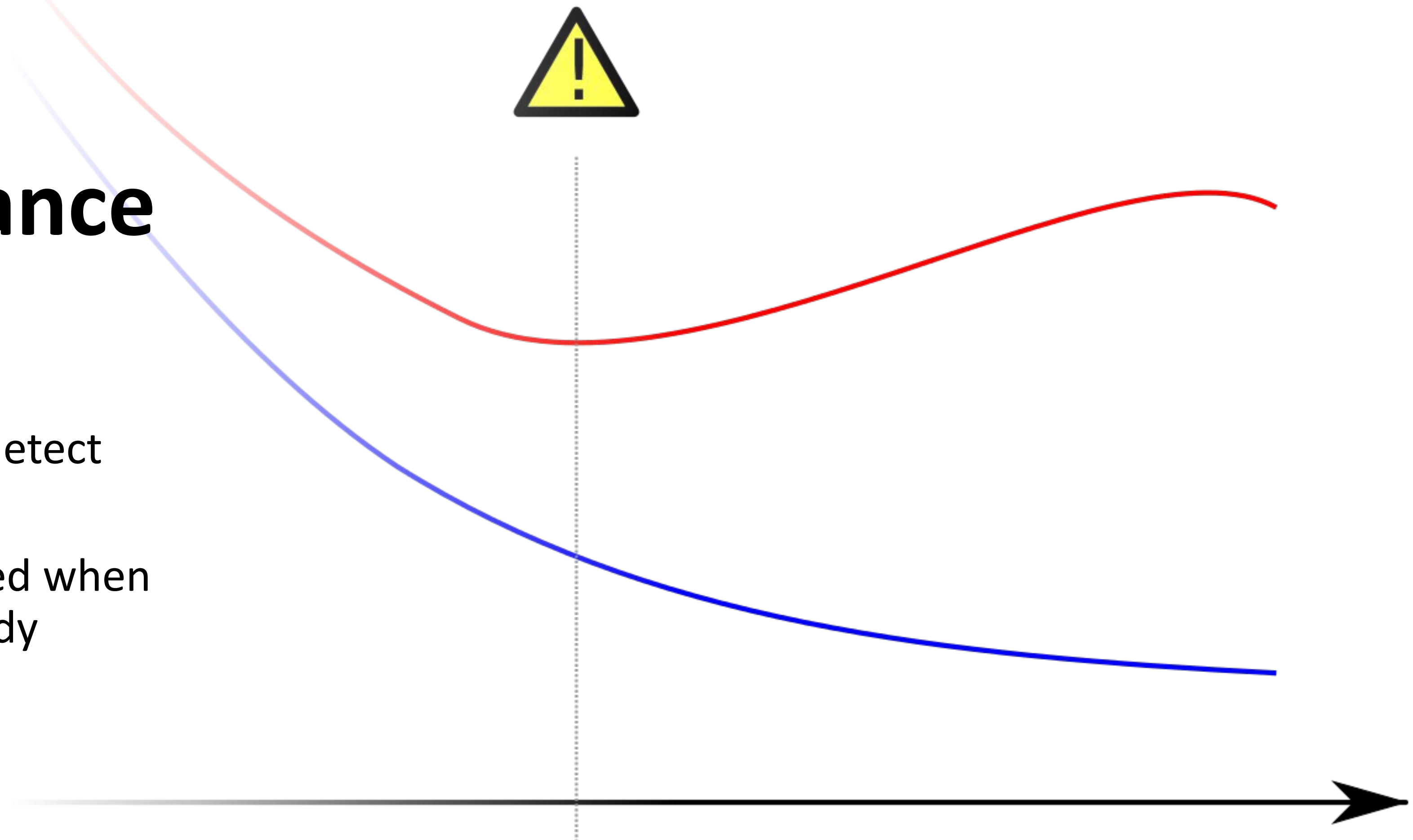
Train / Validation / Test split

- **Training set:** fit the model
- **Validation set:** tune hyperparameters, select model
- **Test set:** evaluate generalization on unseen data
  - Never tune against this! Otherwise it becomes a validation set
- Ratios depend on data volume, frequency of defects and your ability to validate
- Goal: assess generalization performance as honestly as possible
- Challenge: create representative test set with rare classes

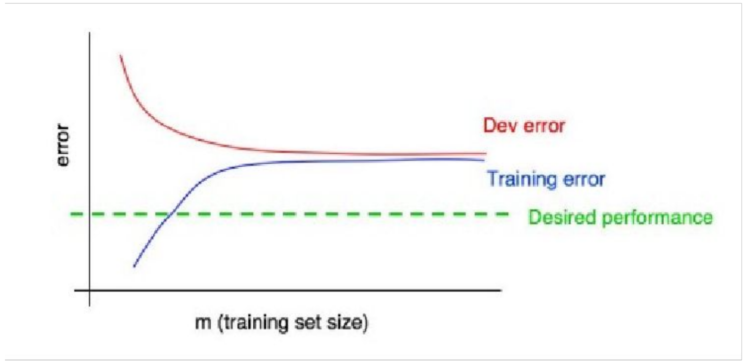


Bias and variance

- Validation set is used to detect overfitting
- Training should be stopped when validation set shows steady performance decrease

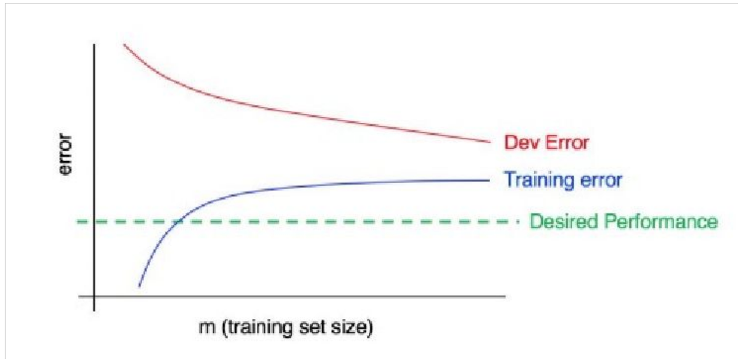


High bias, low variance



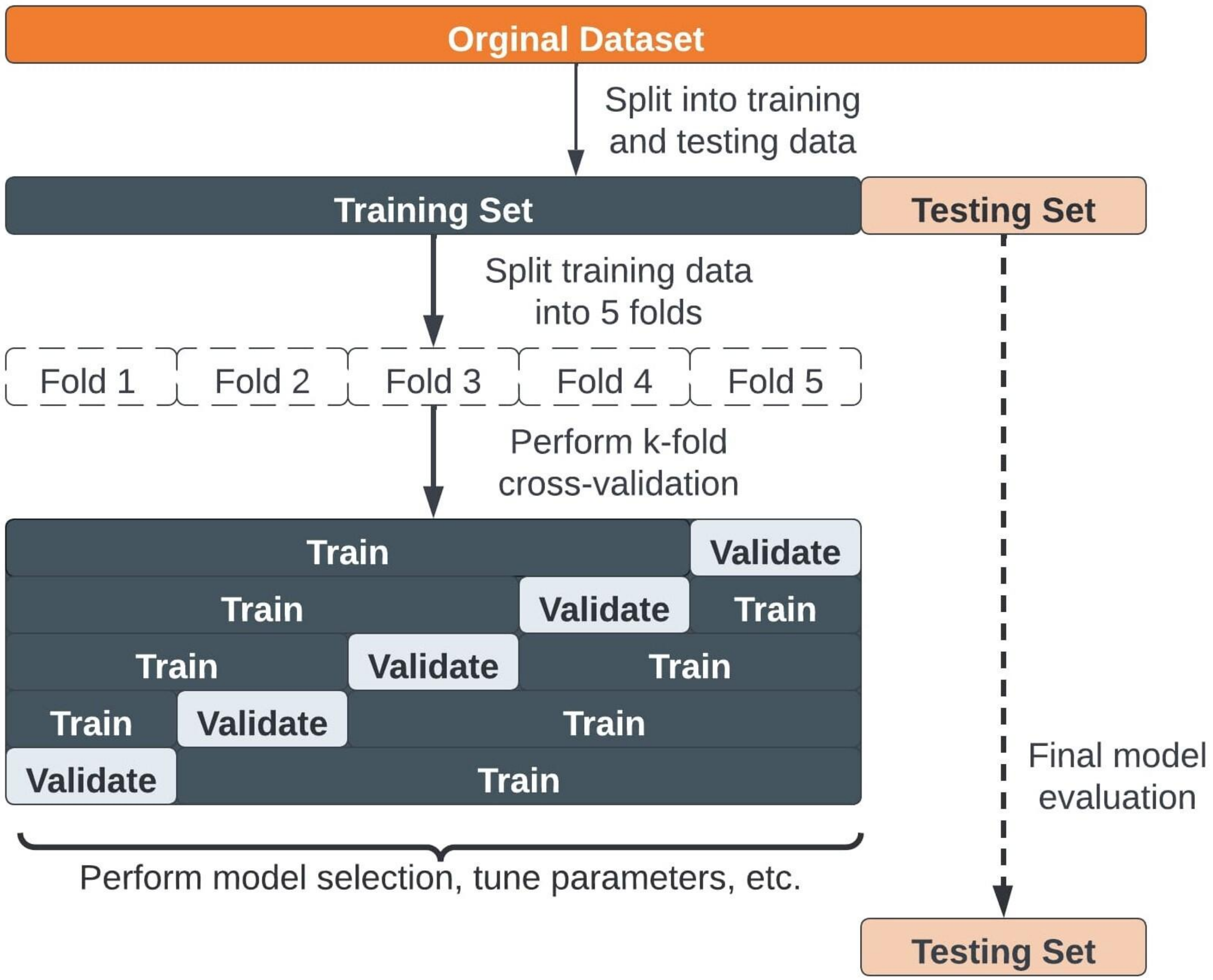
Low bias, high variance

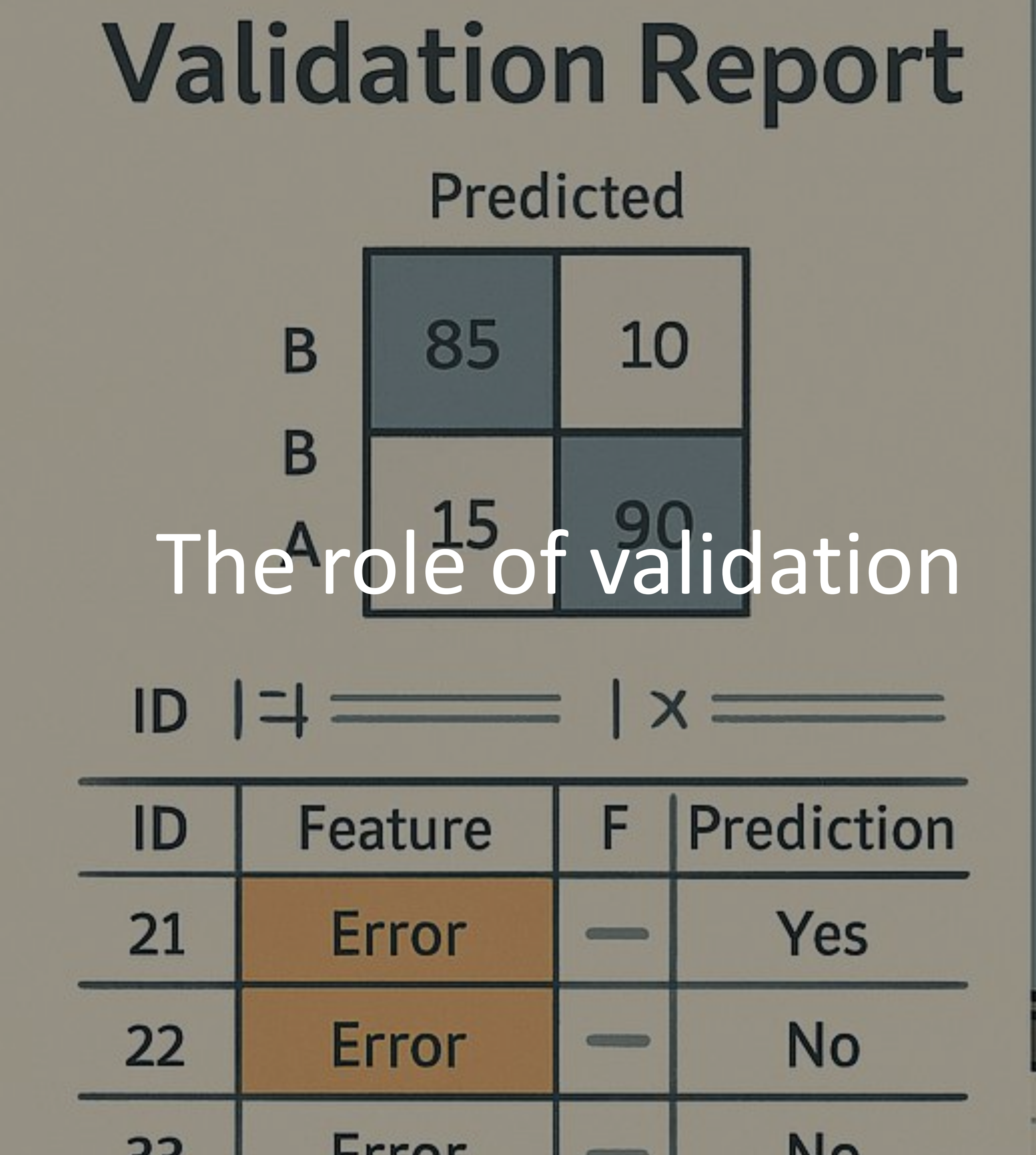
High bias, high variance



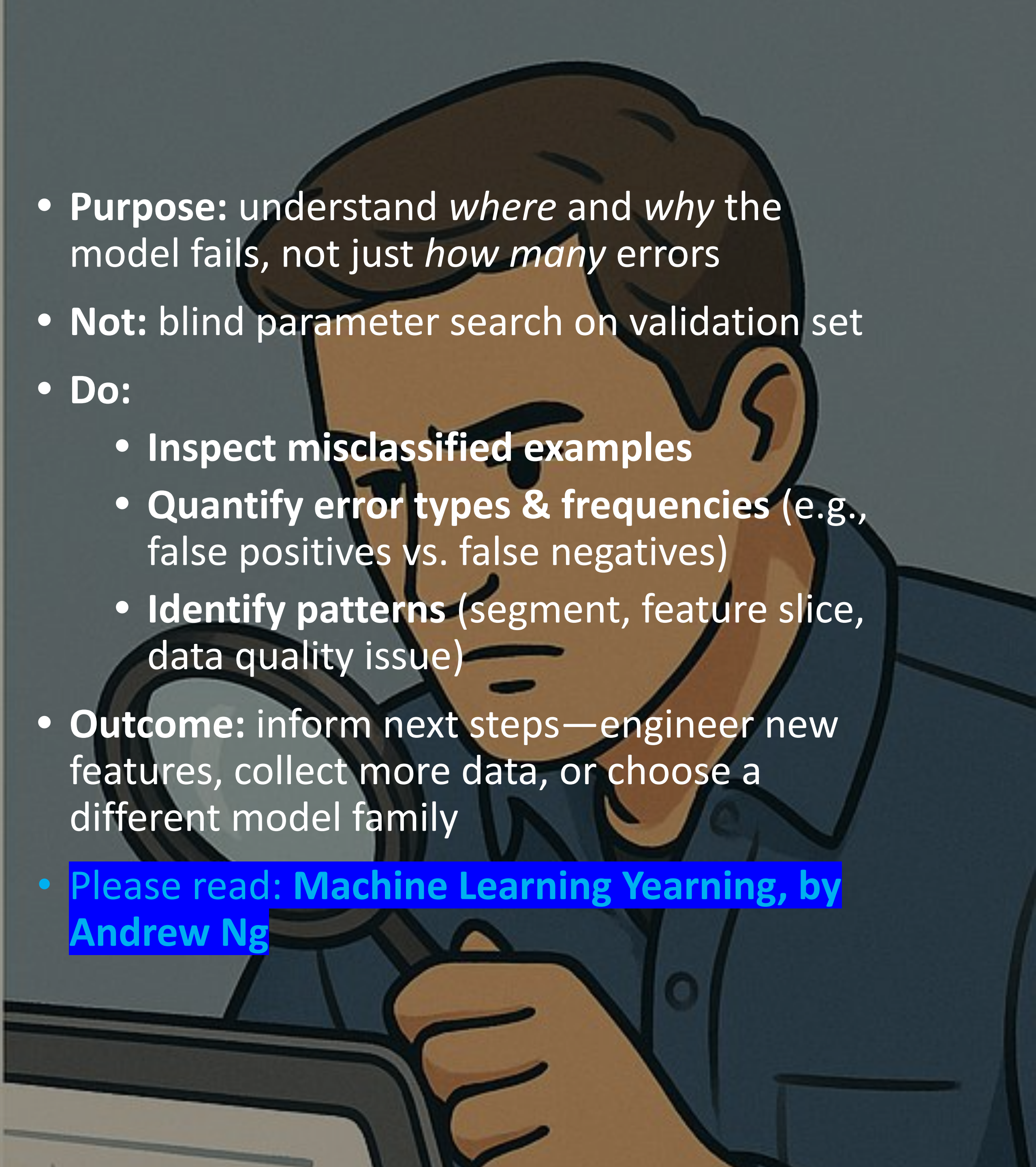
Cross-validation

- Instead of one split, rotate through **k folds**
- Train on k-1 folds, validate on the 1 left out — repeat
  - Common choice: k = 10
  - **Don’t use the test set!!**
- Average performance across all folds
- Helps reduce variance due to unlucky splits



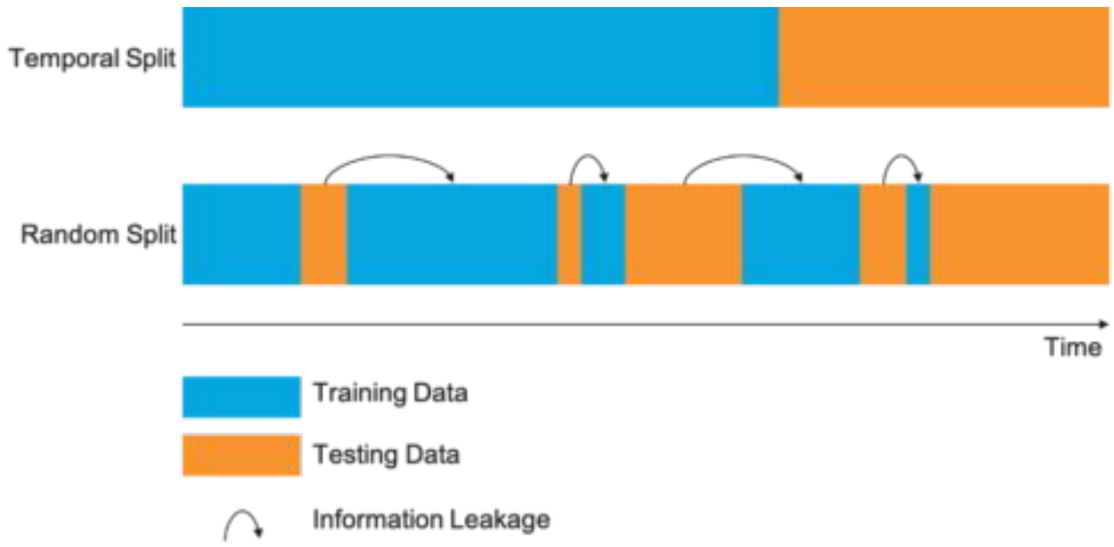


The role of validation



- **Purpose:** understand *where* and *why* the model fails, not just *how many* errors
- **Not:** blind parameter search on validation set
- **Do:**
  - **Inspect misclassified examples**
  - **Quantify error types & frequencies** (e.g., false positives vs. false negatives)
  - **Identify patterns** (segment, feature slice, data quality issue)
- **Outcome:** inform next steps—engineer new features, collect more data, or choose a different model family
- Please read: **Machine Learning Yearning, by Andrew Ng**

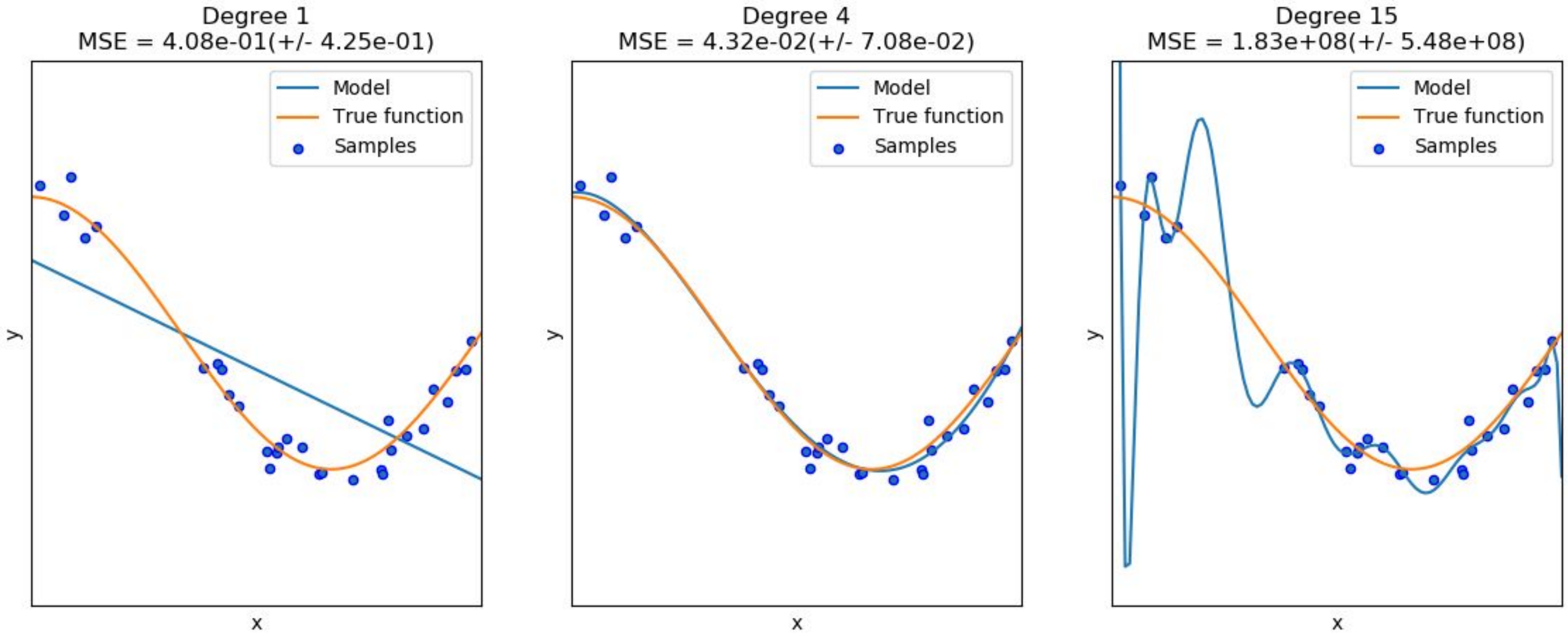
Pitfalls: Data leakage



- **What is data leakage?**
  When information from outside the training process (especially from validation/test or future data) "sneaks in," giving overly optimistic performance.
- **Common Leakage Examples:**
  - **Global Imputation:** filling missing values using the mean/median of the *entire* dataset (train+validation+test)
  - **Pre-Split Scaling:** computing scaler parameters (mean/std or min/max) on the full dataset before splitting
  - **Target-Derived Features:** creating features that directly or indirectly encode the label (e.g., "days until churn" when predicting churn)
  - **Time Leakage:** including future data points as features (e.g., last month's sales when predicting this month's demand)

Pitfalls: Overfitting

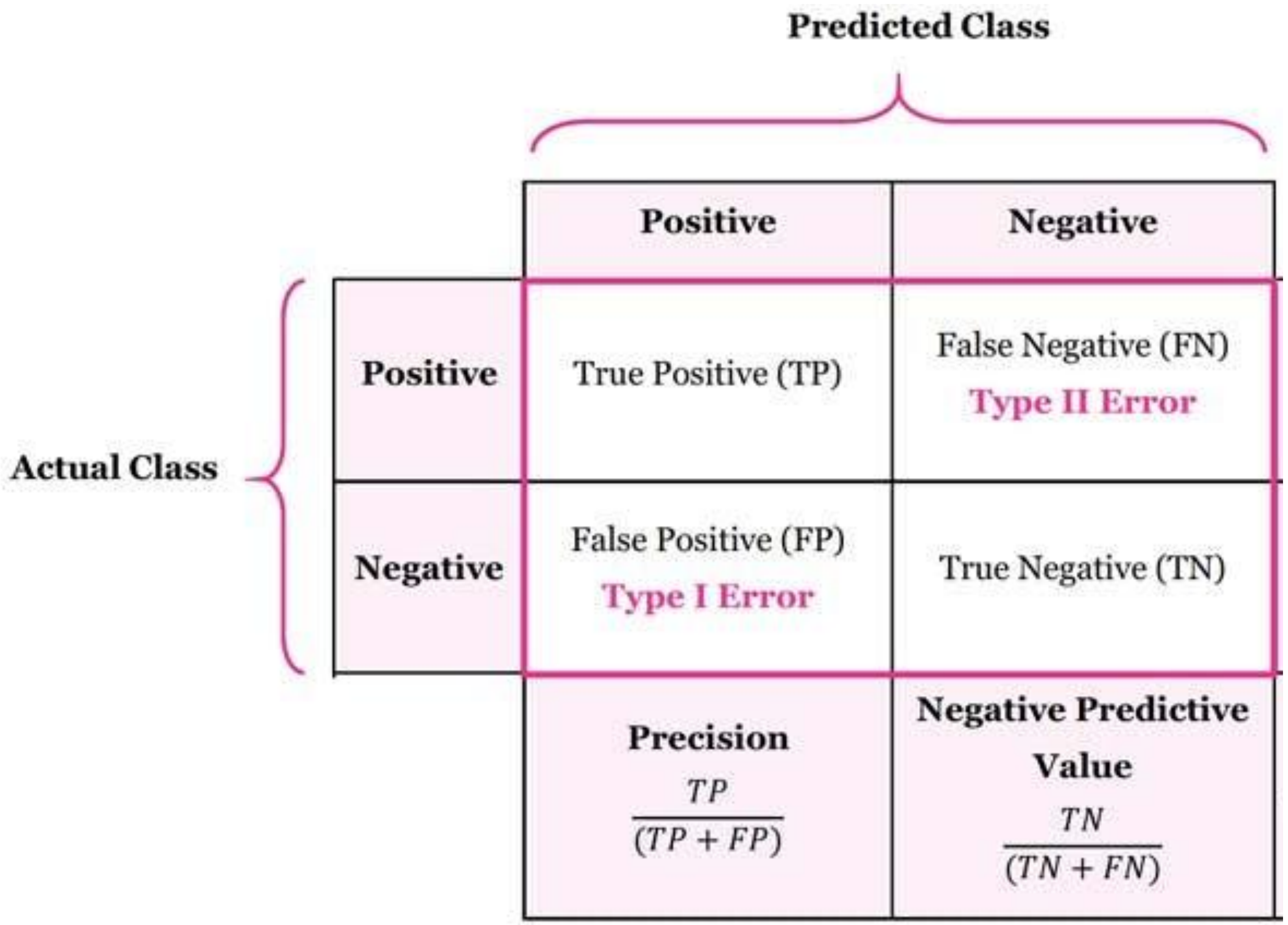
- ML models iteratively fit input data
  - “Capacity”: the ability of a model to fit complex functions
  - (Almost) monotonic increase in performance
  - Excessive training on the dataset will result in overfitting
  - Model fits noise and statistical irregularities rather than the true signal



4. Model evaluation

Classification metrics

- **Accuracy:** % of correct predictions
    - **Pitfall:** misleading on imbalanced data
    - **Use when:** classes roughly balanced, equal error costs
- **Precision:** TP/(TP+FP) → “When I say positive, am I usually right?”
- **Recall:** TP/(TP+FN) → “Of all actual positives, how many did I find?”
- **F1-score:** harmonic mean of P & R, balances the two

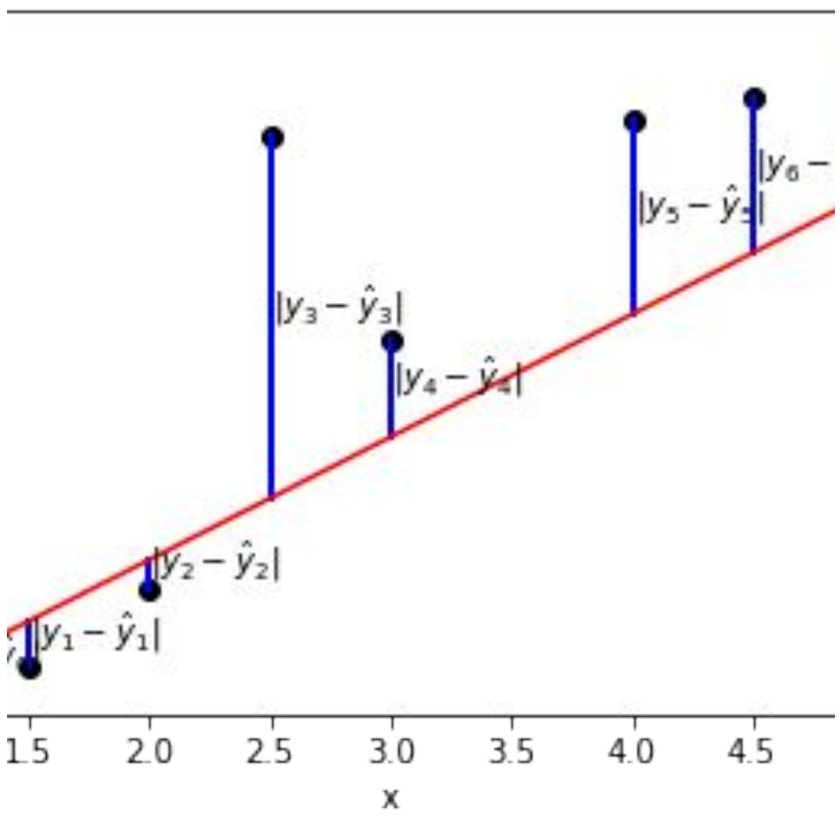


Confusion Matrix:

	Predicted Positive	Predicted Negative
Actual Positive	8	2
Actual Negative	3	87

Regression metrics

- **Mean Squared Error (MSE):**
  - Average of squared errors $\frac{1}{N}\sum(y_i - \widehat{y_i})^2$
  - Used as a **training loss**—smooth and differentiable
- **Root Mean Squared Error (RMSE):**
  - $\sqrt{MSE}$
  - **Back in original units**—easy to interpret ("average error of $5 k")
- **Mean Absolute Error (MAE):**
  - Average of absolute errors $\frac{1}{N}\sum|y_i - \widehat{y_i}|$
  - **More robust** to outliers, equal weight to all errors



Choosing metrics

- Match your metric to what really matters: missed fraud vs. false alarms, or big price misses vs. average error. That’s how you drive real value.
- **Always align metric choice to business impact!**

Choosing metrics

- **Scenario:** Predicting customer churn for a subscription service
  - **Common ML choice:** maximize **accuracy (e.g. 95%)**
  - **Business goal: Minimize lost revenue** from high-value customers
- Model may optimize ignoring churners worth $1000/yr vs churners worth $50/yr!
- **Better metric:**
  - **Weighted recall** on top 10% highest-value customers
  - Or **expected revenue retained**: sum(value_i × TP_i)

6. Challenges: Imbalanced data and interpretability

Why imbalance matters

- Rare-class problems: fraud (1 %), disease (5 %), churn (10 %)
- High accuracy can hide **0 % recall** on the minority class
- Technical risk → missed cases
- Ethical risk → under-represented groups harmed

Confusion (1 000 samples)	Predicted +	Predicted –
Actual + (50)	**0**	**50**
Actual – (950)	0	950

Countermeasures

Oversample minority → duplicates / SMOTE

- Pros: keeps majority data
- Cons: risk of overfitting

Undersample majority → drop excess negatives

- Pros: faster training
- Cons: discards information

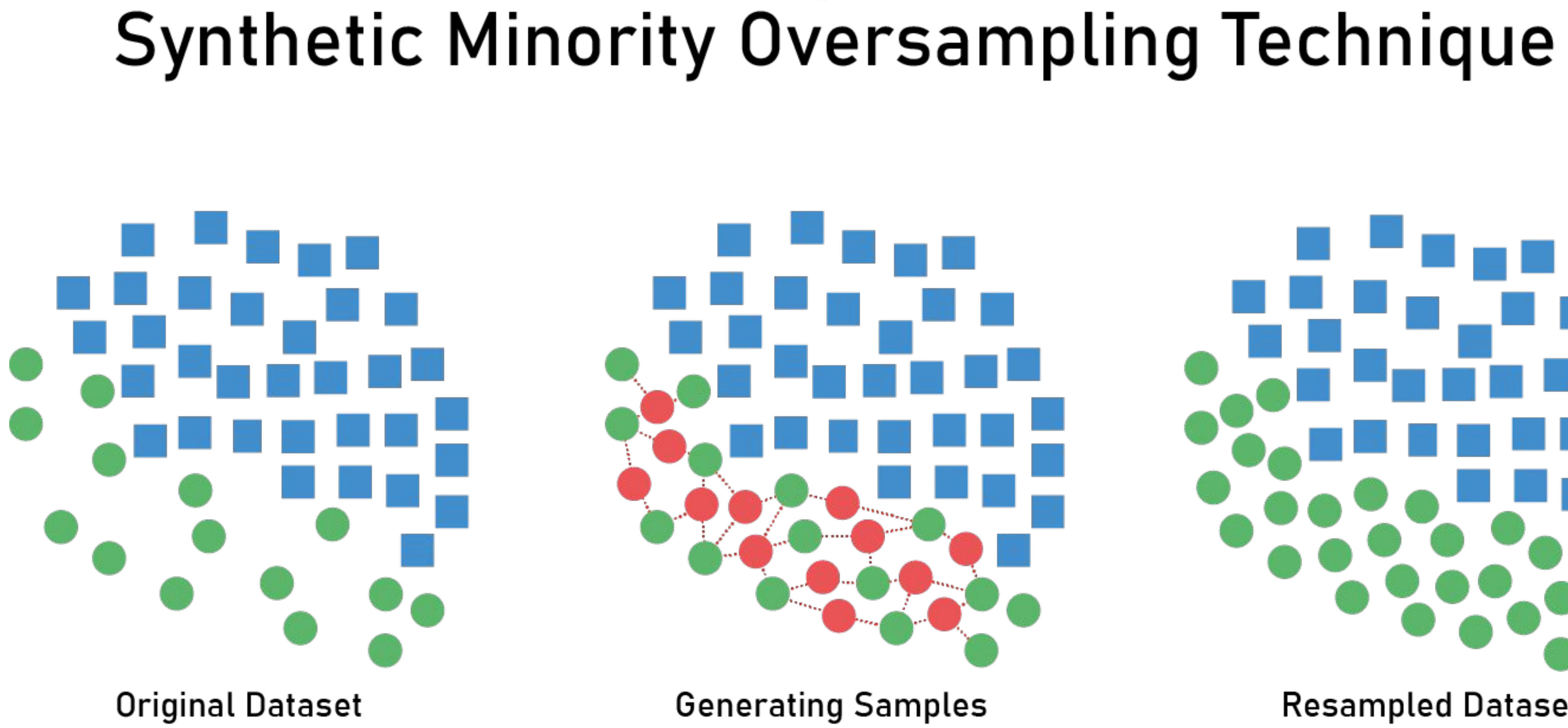
Weights

- Give error weights to minority class inversely proportional to frequency

Metrics

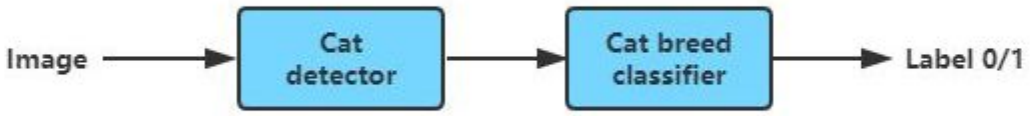
- Use recall, F1 measures for minority focus

Countermeasures - SMOTE



Why interpretability matters

- **Trust:** stakeholders want to know *why* a prediction happens
- **Debugging:** surface spurious correlations & data errors
- **Fairness & compliance:** detect bias, meet regulations (GDPR, banking, healthcare)
- Two types of interpretability
  - System interpretability = auditability. Can I find out what is wrong?



  - Feature interpretability = attribution. Can I understand the reason behind the prediction?
    - NN = attention models
    - Trees = SHAP, LIME, etc.

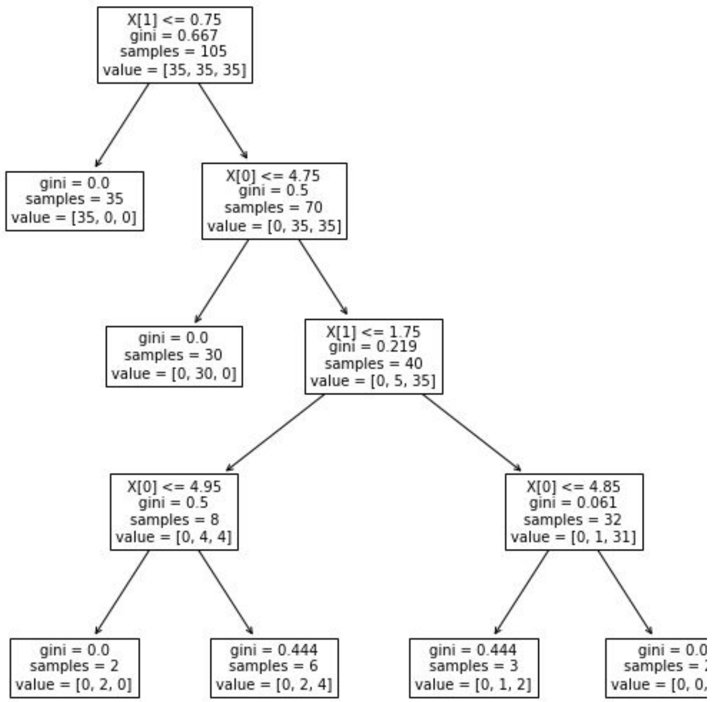
Feature interpretability – feature importance

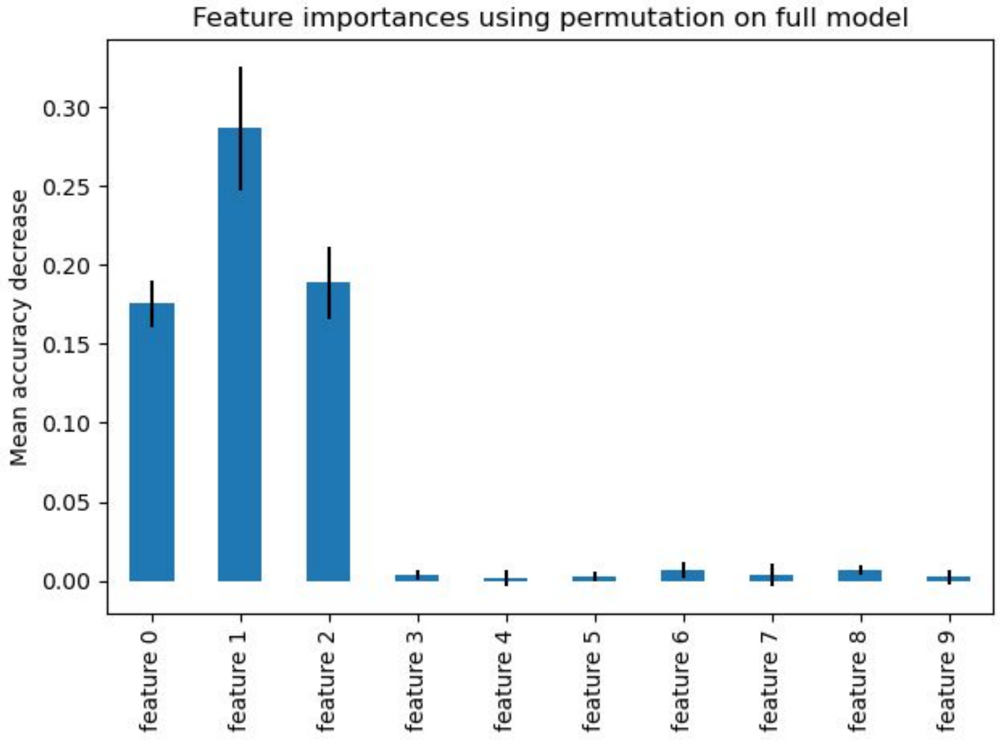
Recall that in trees, decision features close to the root are better at separating samples between branches than those near the leaves

Two feature importance methods:

1. Average predictive power
2. Permutation importance: shuffle each column, measure drop in score

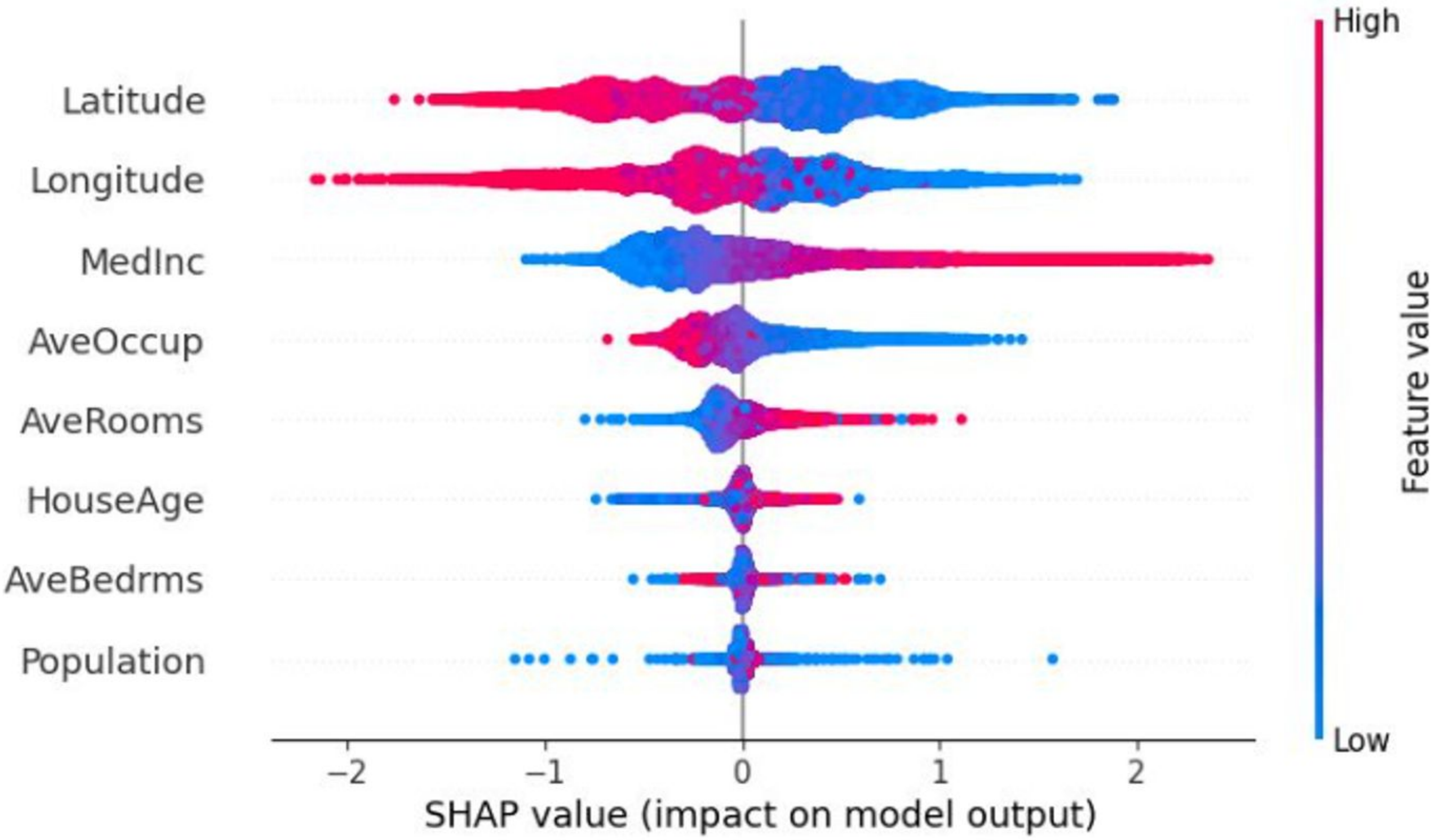
- **Limitation:** only global; can be misleading if features are correlated





Feature interpretability - SHAP

- Based on **Shapley values** from cooperative game theory
- Each feature = a 'player'; SHAP fairly distributes a prediction's gain
- **Advantages:**
  • Unified framework for any model (tree, DL, linear)
  • **Local + global** explanations (waterfall & summary plots)
  • Additive: contributions sum to the prediction
- **Practical:** pip install shap, then shap.TreeExplainer(model)



A word of caution…

- 99% of ML models today are based on curve fitting / correlations.
- Feature importance indicates correlation, ***not*** causality.
- Business is interested about identifying the ***why*** of observations
  - This requires active experimentation: A/B tests, hypothesis validation, etc.
  - Drive an active experimentation mentality in your organization.

Concluding …

Data and ML professionals work as a bridge between business leadership and numerical observations

Go beyond being a data provider. Inform leadership on:

- What data would be needed to really solve the problem
  - Even if it does not exist today!
- What new technologies the business needs to invest in to solve the problem
- What test would be needed to really validate a hypothesis
  - Get the data from the customer, not from our own biases!

If you treat ML as a parameter optimization problem, your work **will** be automated

- Real value for the business comes from intimate knowledge of the data, connecting it with the business priorities, and suggesting new initiatives to move forward. **ML is just a tool**.

Thank you!