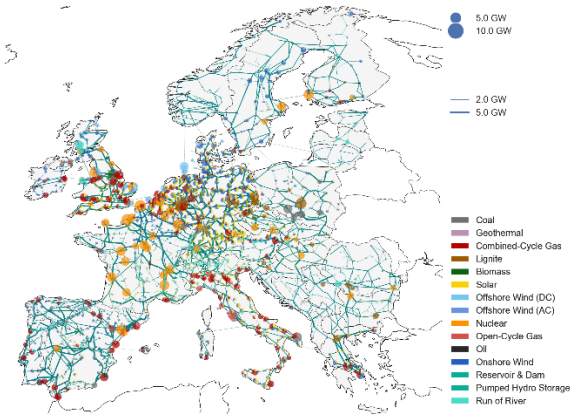
Energy System Modeling with Python



University of Freiburg (Germany) | Faculty of Engineering
Department of Sustainable Systems Engineering | INATECH
Chair for Control and Integration of Grids

Tuesday, 3. June 2025

Theoretical part

Feature Selection: From More to Meaningful

- Think back to Lecture 7: You extended your regression model with new variables. ***How did you decide which features to include?***
- Is there a more systematic way to guide this selection?

Correlation Matrix

- Measures how strongly each input feature is linearly related to the target
- Values range from **–1** (strong negative) to **+1** (strong positive)
- Fast, visual filter for redundancy and relevance

Pearson Correlation

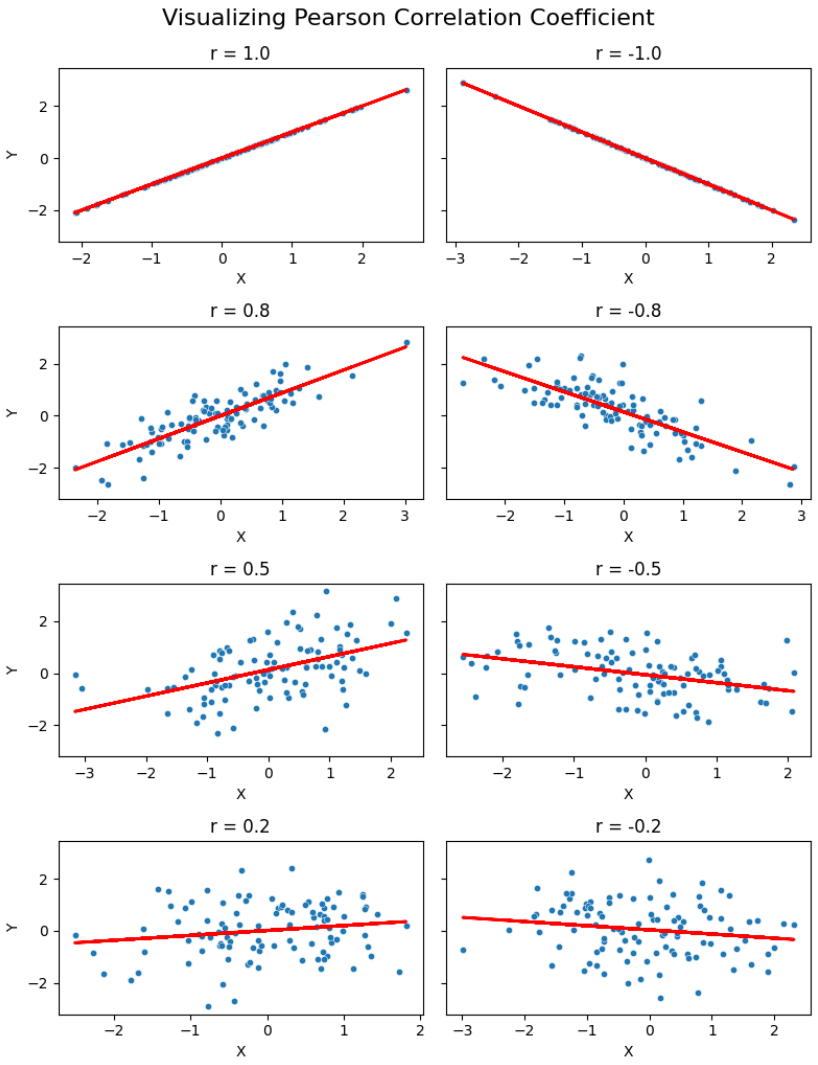
$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y}$$

where:

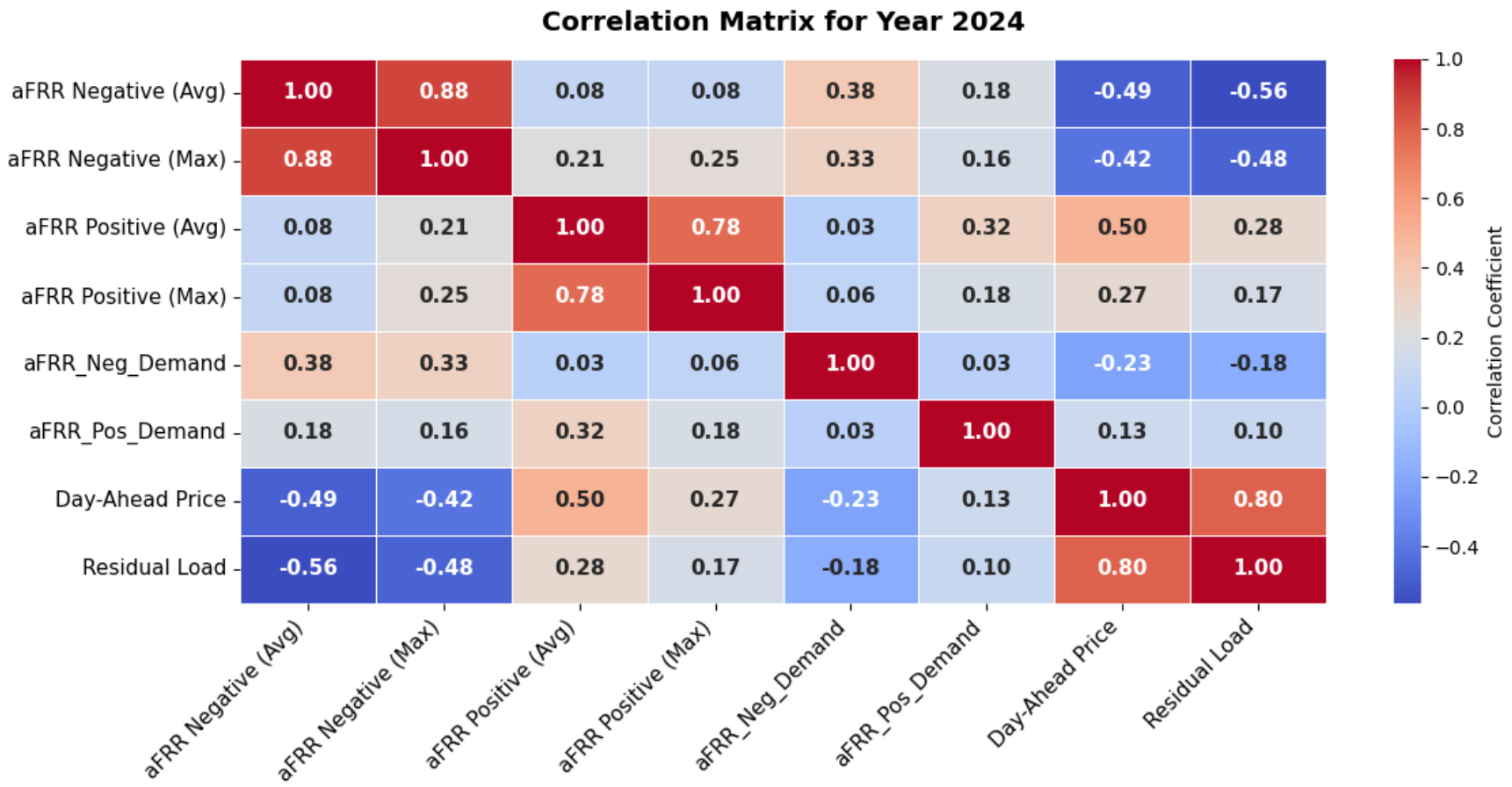
$\mathrm{cov}(X,Y)$ is the covariance

σ_X, σ_Y is the standard deviation of X, Y

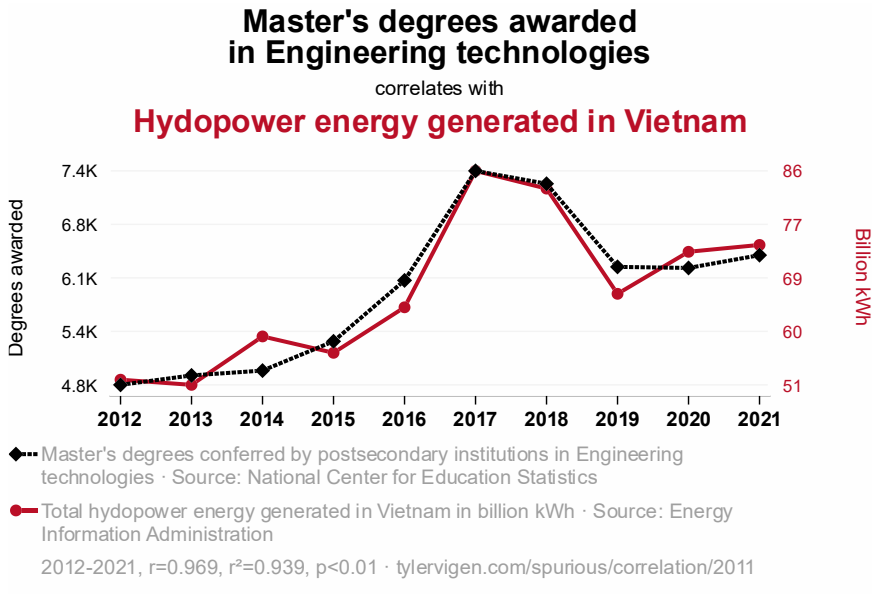
- Measures the **strength and direction** of a **linear relationship** between two variables
- Symmetric: $\rho_{X,Y} = \rho_{Y,X}$
- Values range from −1 (perfect negative) to +1 (perfect positive)

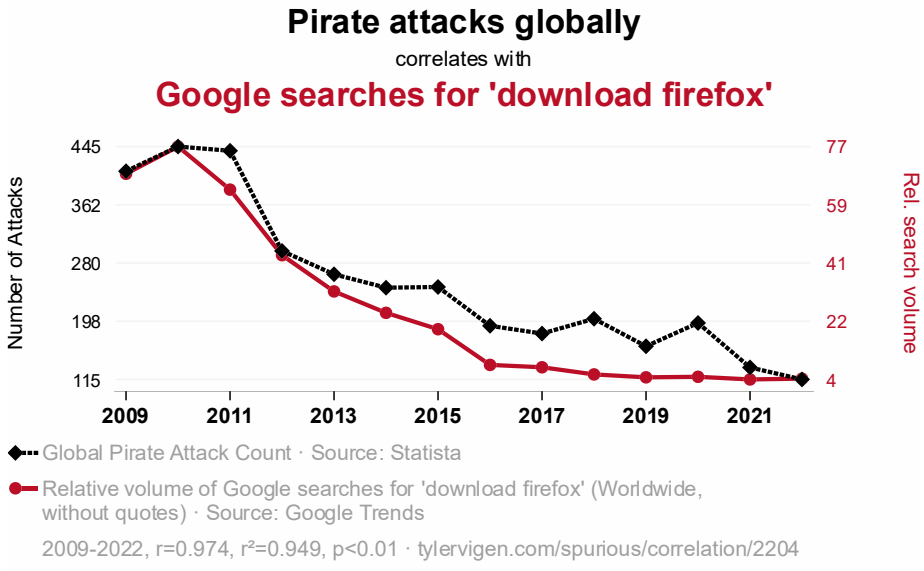


Correlation Matrix



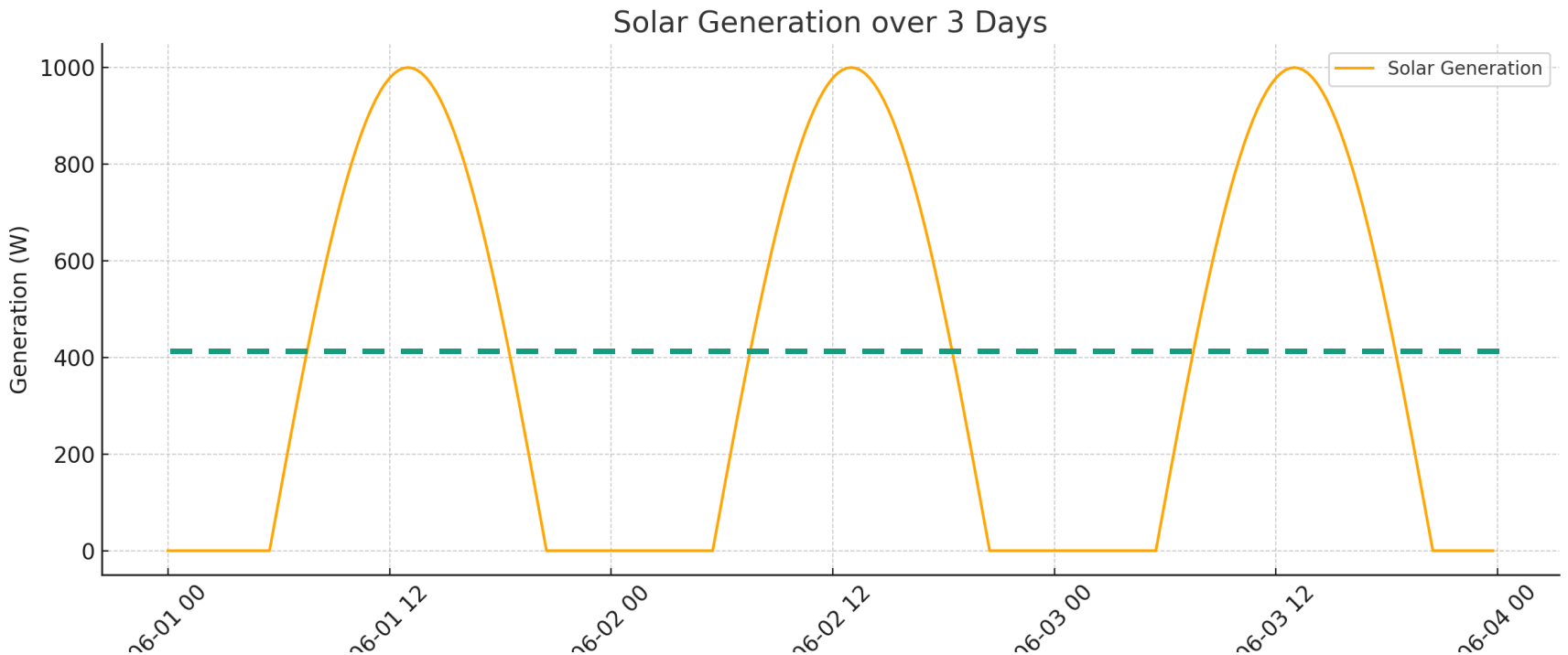
Correlation and Causation





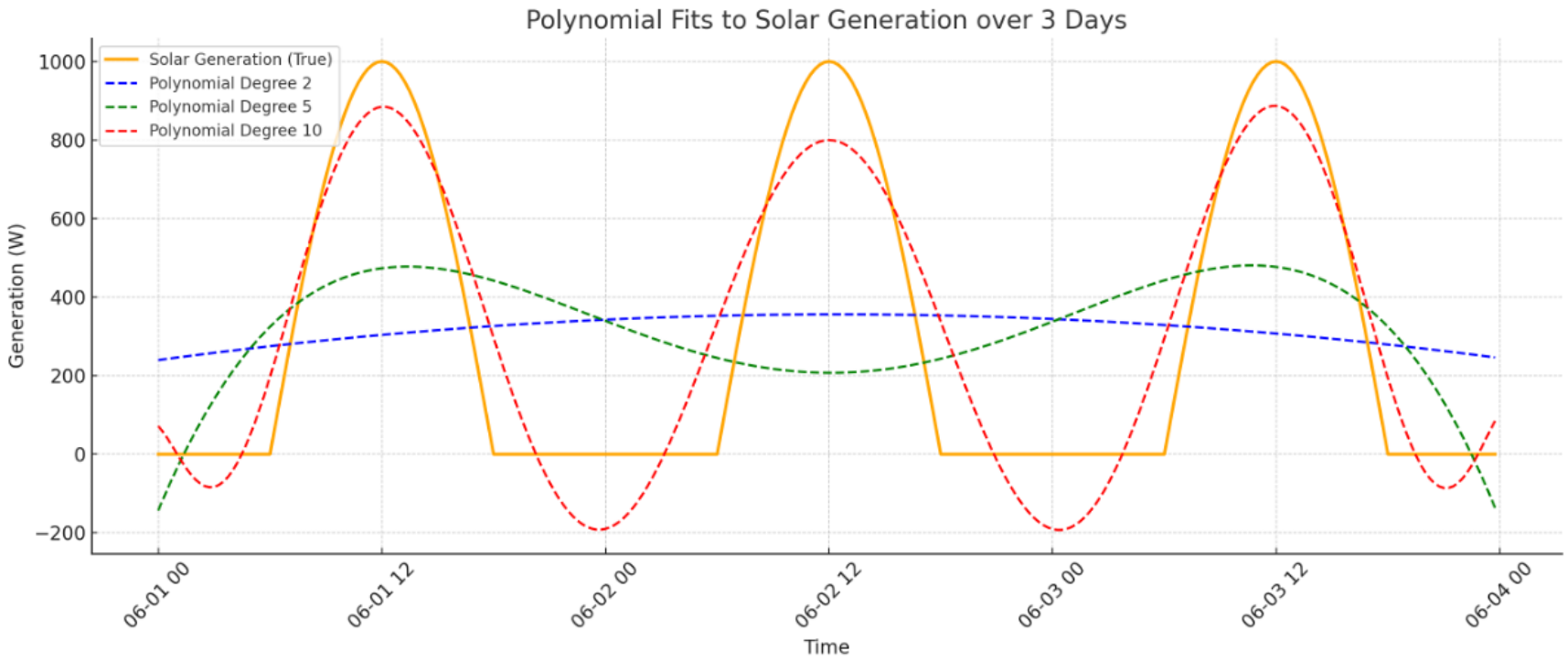
Correlation ≠ Causation

Limits of Linear Models



- **Solar output is smooth and cyclical**, not linear
- **Zero generation at night** — sharp non-linearity
- **Linear models fail** to follow this pattern

Polynomials Help — But Only Up to a Point



Polynomial model

$$\hat{y}(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_d x^d$$

INATECH

UNI FREIBURG

Linear vs Non-Linear Models: It's Not About the Shape

Linear Models
- Linear Regression
- Polynomial Regression
- Ridge / Lasso

Definition:
Linear in the parameters: prediction is a **weighted sum** of input features or transformations.

Example (Polynomial):

$$\hat{y}(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_d x^d$$

Key Insight:
Even with curved output, the model is still **linear** in how parameters influence prediction.

Non-Linear Models
- Decision Trees (Random Forests)
- Gradient Boosting
- Neural Networks

Definition:
Model output depends **non-linearly** on parameters or model structure (e.g., thresholds, activations, hierarchy).

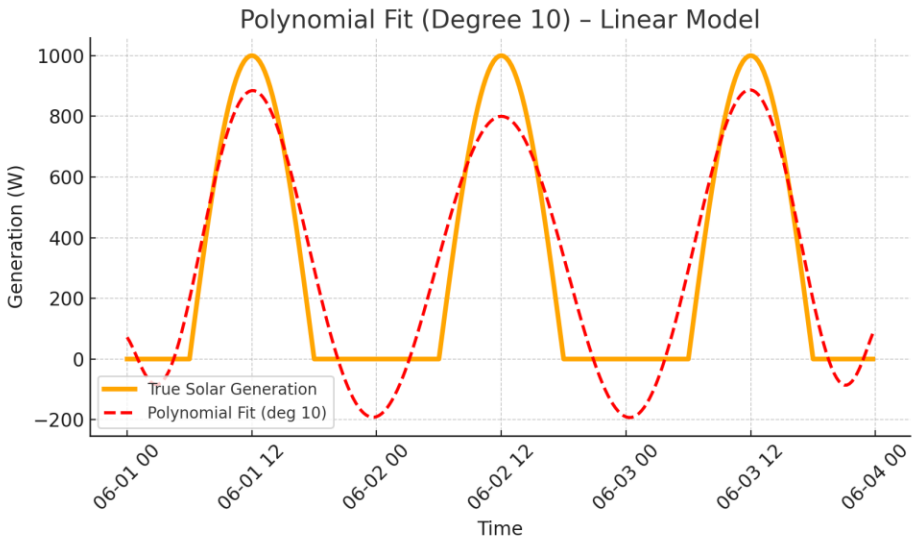
Example (Random Forest):

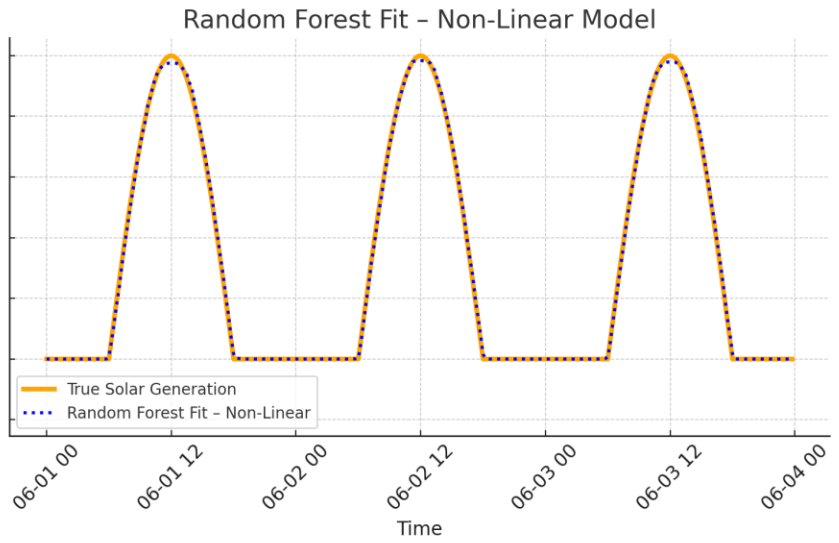
$$\hat{y} = \frac{1}{N} \sum_{i=1}^{N} T_i(x)$$

where T_i are individual decision trees.

Key Insight:
These models can **adapt structure to data**, not just tune weights.

Linear vs Non-Linear in Practice: Fitting Solar Generation



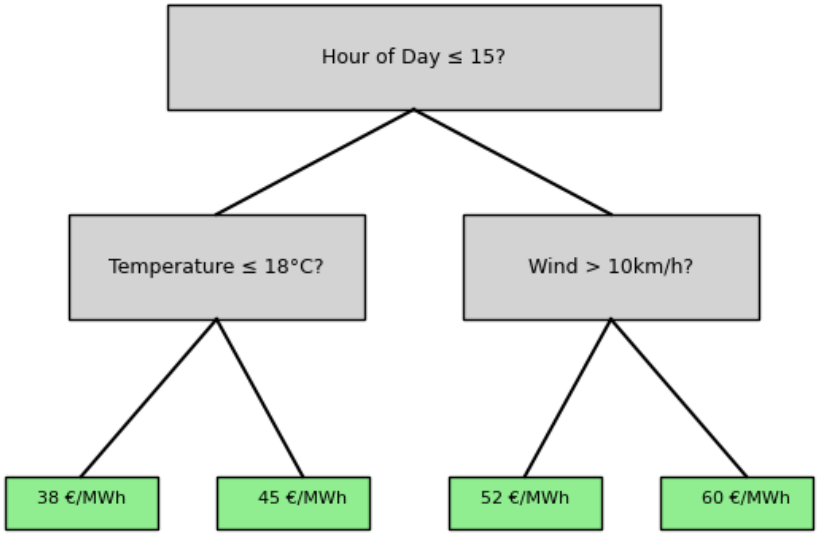


Random Forests: Many Trees, Many Views

Definition:
A Random Forest builds multiple **decision trees;** each trained on a random subset of the data and features. The final prediction is the **average** of all tree outputs.

$$\hat{y} = \frac{1}{N}\sum_{i=1}^{N} T_i(x)$$

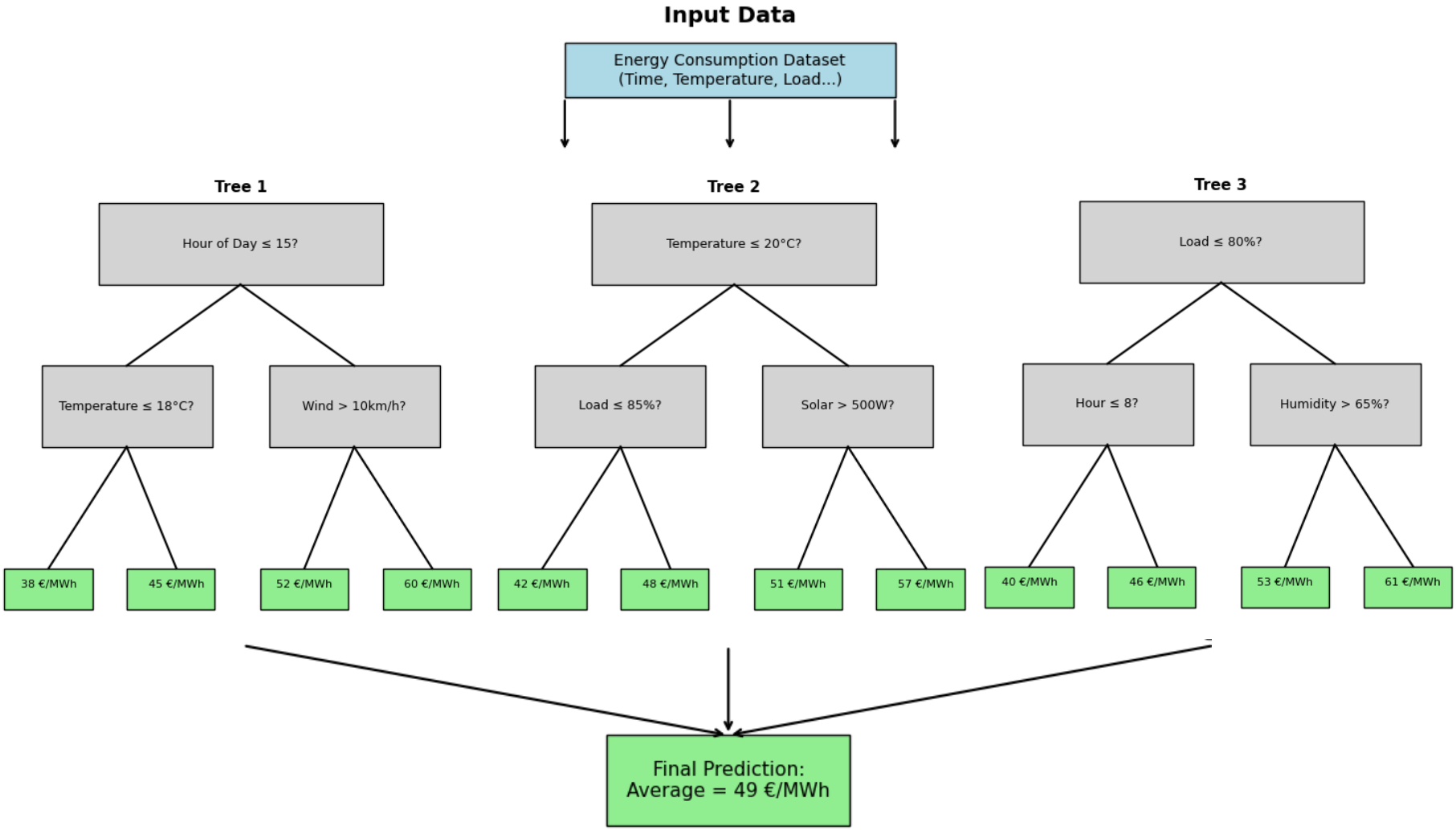
where T_i are individual decision trees.



Why Random Forests Work:

- Each tree captures **different patterns** in the data by **overfitting** to that data
- By averaging many **diverse models**, the forest becomes **stable** and can **generalize**
- Trees are **non-linear**: they model abrupt changes, interactions, and thresholds

Random Forests: Many Trees, Many Views



Random Forests: Hyperparameters

Parameter	Role and Intuition	Typical Values
Number of Trees (*n_estimators*)	How many decision trees are included in the forest. More trees reduce prediction variance via averaging. Beyond a certain point, additional trees do not improve results noticeably but increase computation time.	100 – 300 for most applications Larger for more stability
Number of Features per Split (*max_features*)	Controls how many features each tree considers when making a split. Smaller values increase randomness and reduce tree correlation (→ lower variance) but can increase bias. Larger values reduce bias but may overfit and make trees more similar.	- Regression: use full set - Classification: use √(# features) - More generalization: try 0.3×(# features)

Gradient Boosting: Learning from Mistakes, Iteratively

Key Equation:

$$F_m(x) = F_{m-1}(x) + \alpha \cdot h_m(x)$$

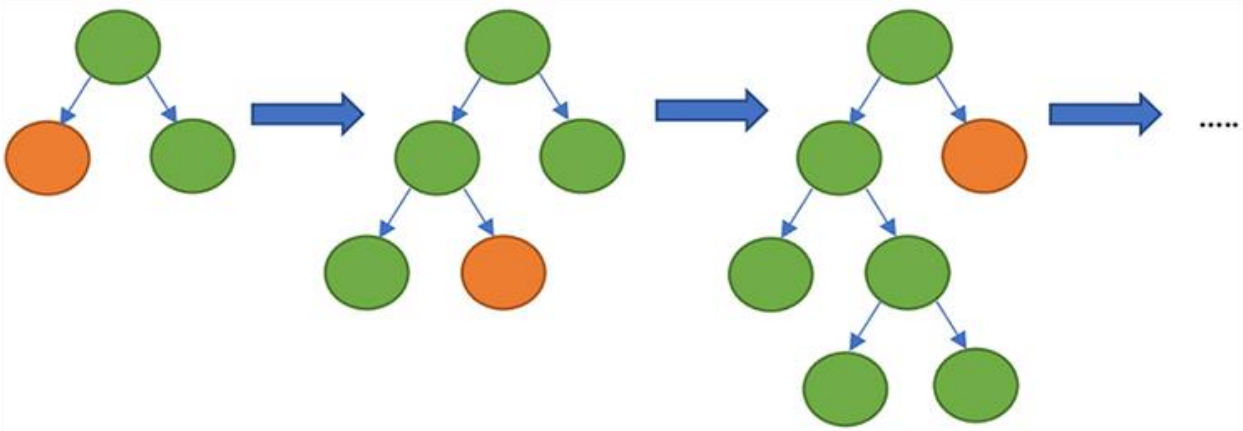
where:
$F_m(x)$ boosted model at stage m
$h_m(x)$: new weak learner fit to the residuals (the gradient)
α: learning rate

Gradient Boosting is a machine learning method that builds a strong model by **adding up many small models** one step at a time.

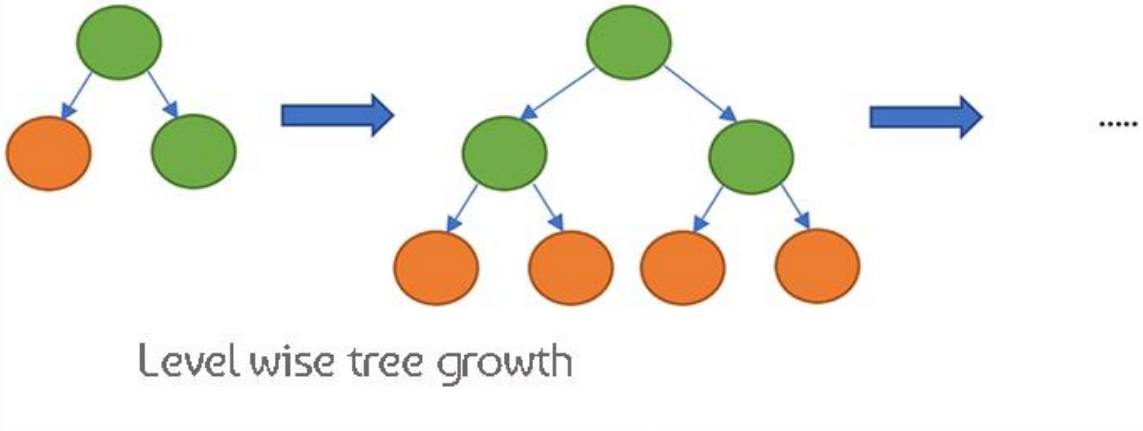
At each step, it fits a new model to the **errors (residuals)** of the current prediction, improving the overall accuracy.

Gradient Boosting in Practice: XGBoost vs. LightGBM

LightGBM



XGBoost



Gradient Boosting in Practice: XGBoost vs. LightGBM

Feature	XGBoost	LightGBM
Tree growth strategy	Level-wise (symmetric trees)	Leaf-wise (asymmetric trees with max depth limit)
Training speed	Slower	Faster
Memory usage	Higher	Lower
Accuracy	Often slightly higher due to deeper tree coverage	Competitive, sometimes overfits on small datasets
Support for categorical features	Requires encoding	Native support
Best suited for	Small to medium datasets where stability is key	Large datasets and low-latency training
Out-of-the-box stability	More robust, harder to overfit	Needs tuning (e.g., max_depth) to avoid overfitting

XGBoost: Hyperparameters

Parameter	Role & Intuition	Typical Values
max_depth	**Tree complexity control.** Limits how deep each tree grows. Deeper trees model complex interactions but increase risk of overfitting.	3–10
min_child_weight	**Split conservatism.** Minimum sum of instance Hessian weights in a child. Higher values force more samples per leaf, reducing sensitivity to noise.	1–10
learning_rate	**Shrinkage factor.** Scales the contribution of each new tree. Lower values require more boosting rounds but improve generalization by preventing large, destabilizing updates.	0.01–0.3

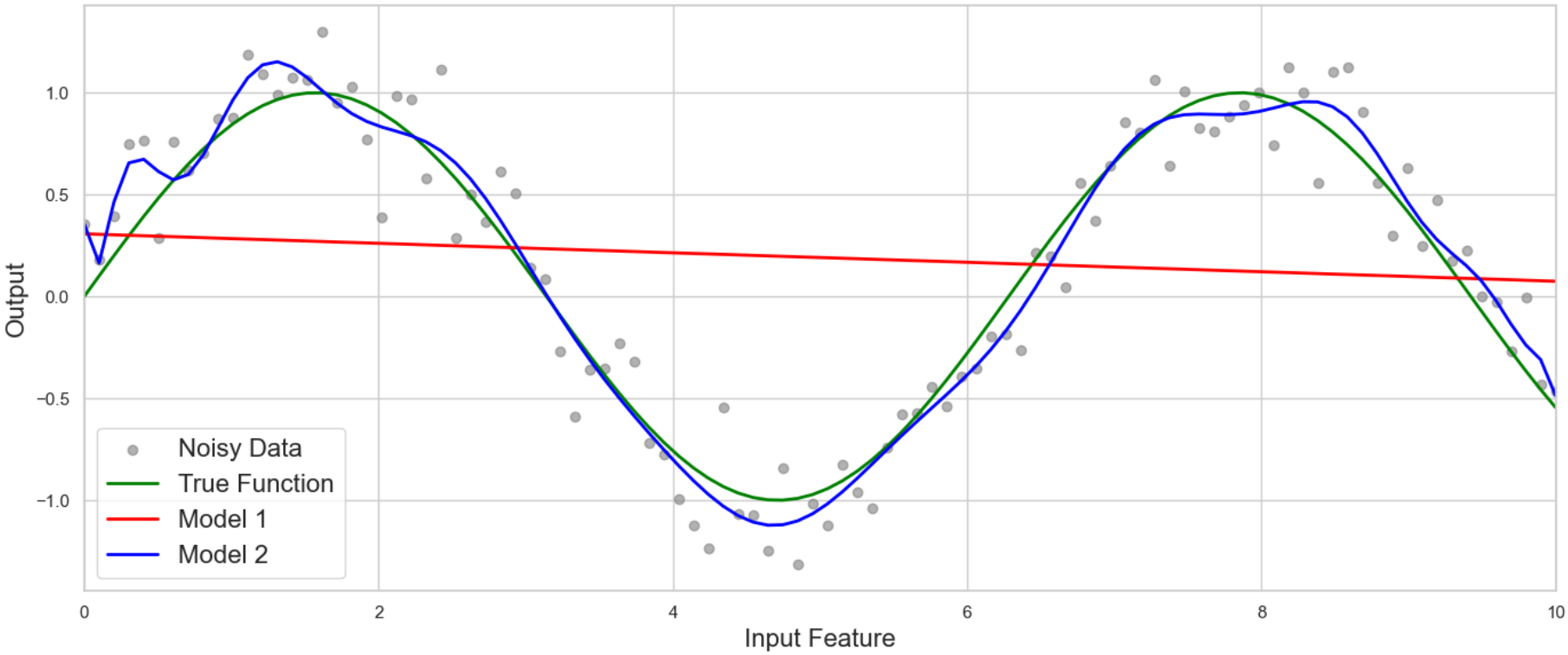
LightGBM: Hyperparameters

Parameter	Role & Intuition	Typical Values
num_leaves	**Main complexity control** Leaf-wise growth can yield very deep trees; set num_leaves ≤ 2^max_depth to avoid runaway depth and overfitting.	30
min_data_in_leaf	**Regularization of leaf size** Requires a minimum number of samples per leaf. Larger values (hundreds–thousands) prevent overly specific leaves and guard against overfitting, especially on large datasets.	100–1 000
max_depth	**Explicit depth cap** Bounds leaf-wise growth; when used, ensure num_leaves ≤ 2^max_depth. Controls the maximum levels of splits per tree.	5–15

Random Forest vs. Gradient Boosting

Feature	Random Forest	Gradient Boosting
Training Strategy	Parallel, independent trees (bagging)	Sequential, each tree learns from residuals (boosting)
Tree Depth	Typically deep trees (low bias, higher variance)	Typically shallow trees (higher bias, lower variance)
Bias vs. Variance	Reduces variance through averaging	Reduces bias by sequentially correcting errors
Interpretability	Moderate (feature importance available)	Lower (complex ensemble of corrections)
Overfitting Risk	Lower due to averaging over diverse trees	Higher—requires careful regularization (e.g., learning rate, early stopping)
Performance	Robust, faster to train, good for large datasets	Often more accurate with proper tuning, but more sensitive to noise and hyperparameters

Generalization



Generalization Error

In machine learning, the goal isn't merely to achieve high performance on the training dataset. Rather, the key focus is on how well the **model generalizes** — that is, ***how accurately it can make predictions on previously unseen data.***

$$GE = E\left[\left(y - \hat{f}(x)\right)^2\right] = \sigma^2 + \left(E\left[\hat{f}(x)\right] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E\left[\hat{f}(x)\right]\right)^2\right]$$

Generalization error (test error) is the error or the loss that a model incurs when it is tested against possibly all the infinite data that follows the same distribution as the training data.

Bias-Variance Decomposition

$$GE = E\left[\left(y - \hat{f}(x)\right)^2\right] = \sigma^2 + \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right]$$

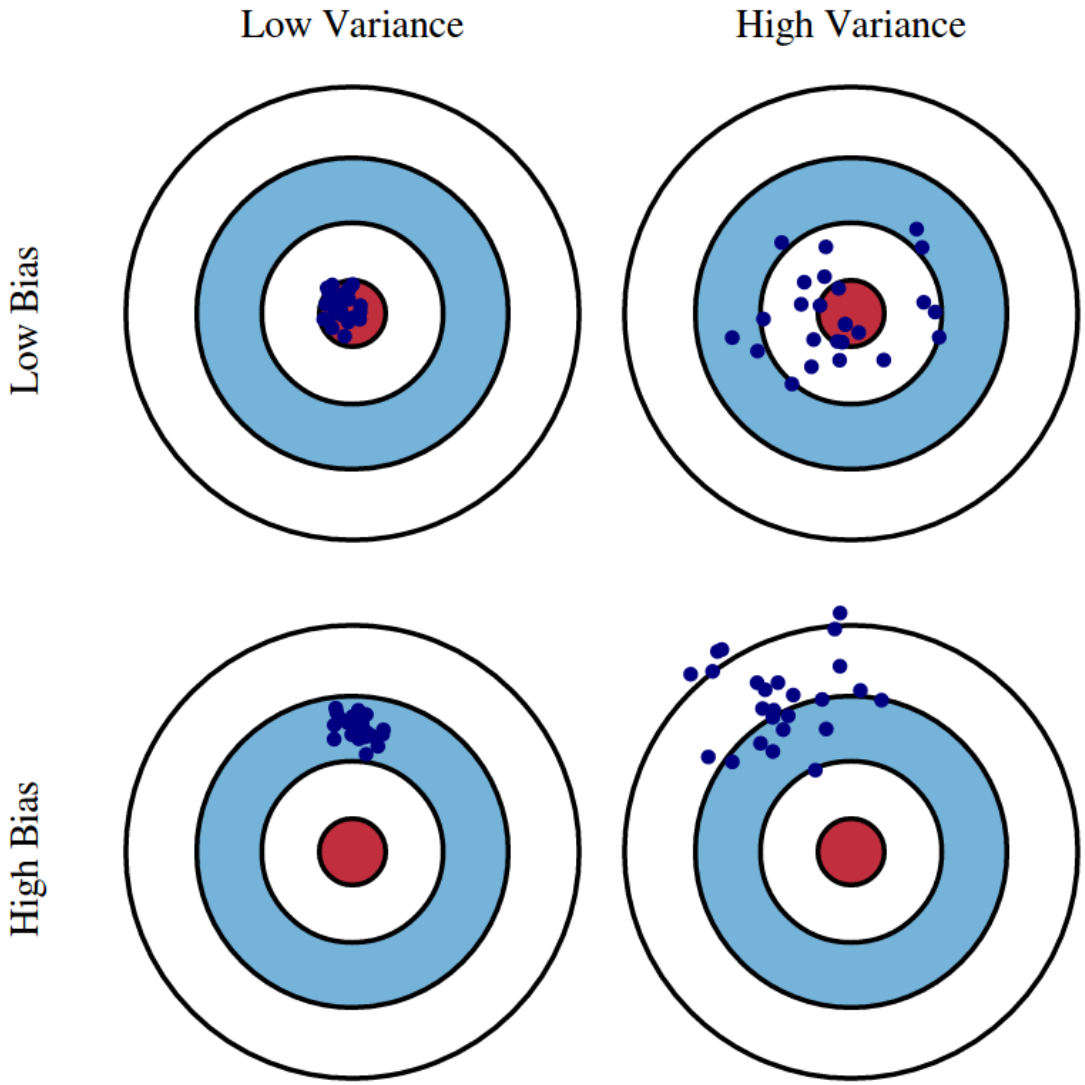
$E\left[\left(y - \hat{f}(x)\right)^2\right]$ – **expected squared error** of the model's prediction $\hat{f}(x)$ for a given input x, averaged over all possible training datasets and the inherent randomness in the output y. It represents the model's overall expected prediction error at point x.

σ^2 – **irreducible error**, which is the variance of the noise in the data. It accounts for the variability in y that cannot be captured by any model, reflecting the inherent unpredictability in the data.

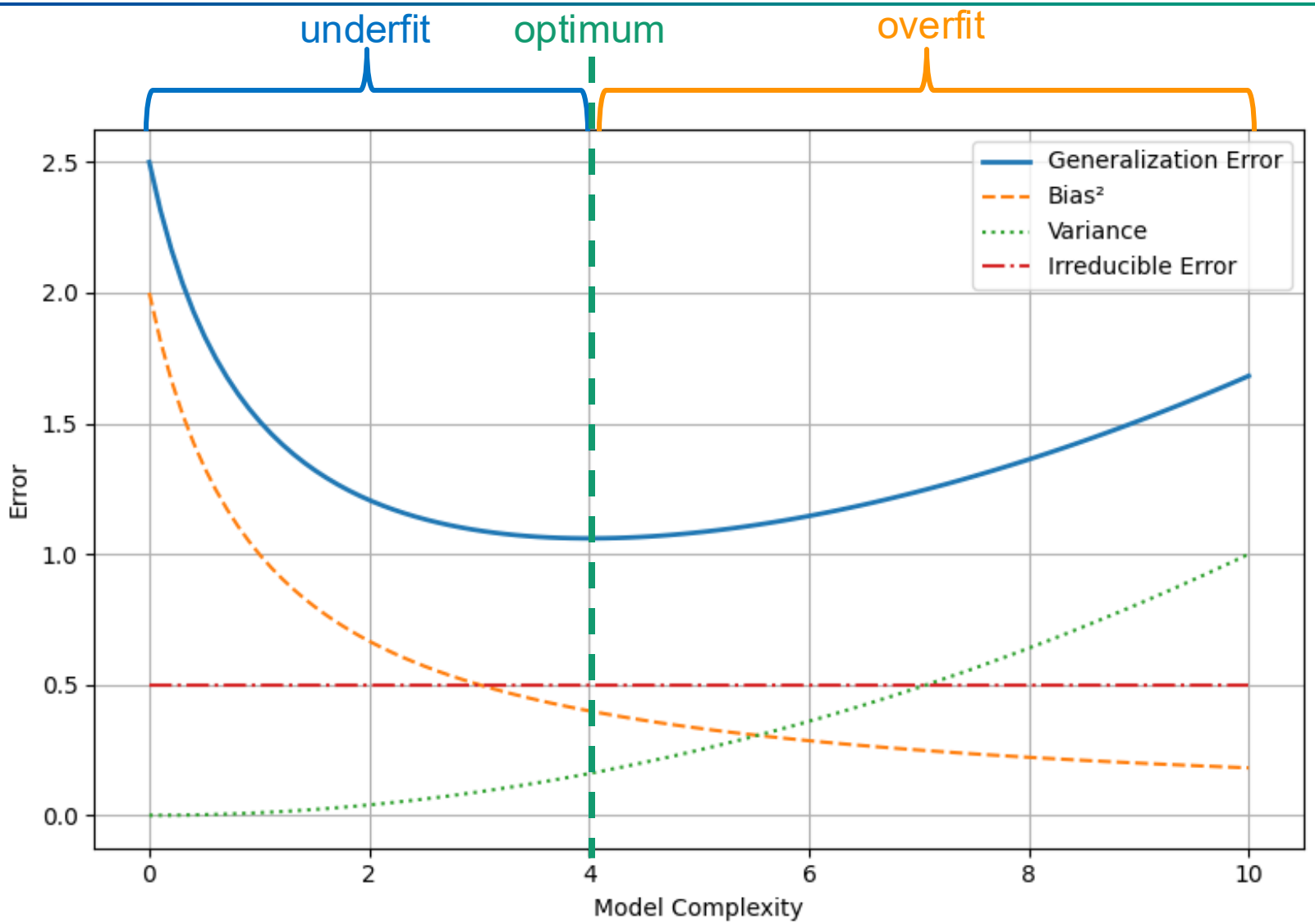
$\left(E[\hat{f}(x)] - f(x)\right)^2$ – squared **bias** of the model at point x. It measures the difference between the average prediction of the model (over all possible training datasets) and the true function $f(x)$. A high bias indicates that the model systematically deviates from the true function.

$E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right]$ – **variance** of the model's predictions at point x. It quantifies how much the model's predictions fluctuate for different training datasets. A high variance indicates that the model's predictions are sensitive to the specific data it was trained on.

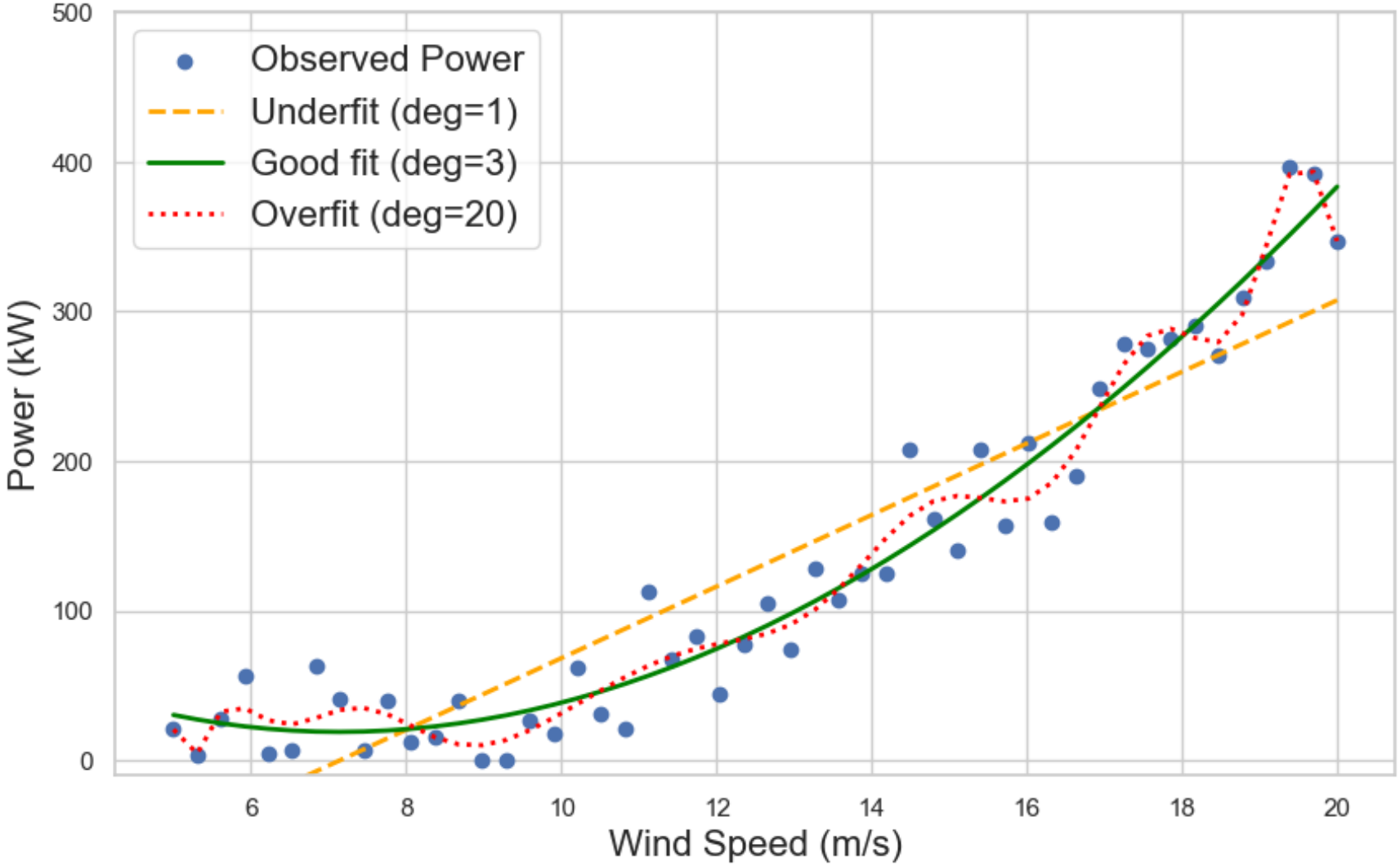
Bias-Variance: Guess what is what



Bias-Variance Tradeoff



Bias-Variance Tradeoff



INATECH

UNI FREIBURG

Train–Test Split & Evaluation

Training Set (80 %)

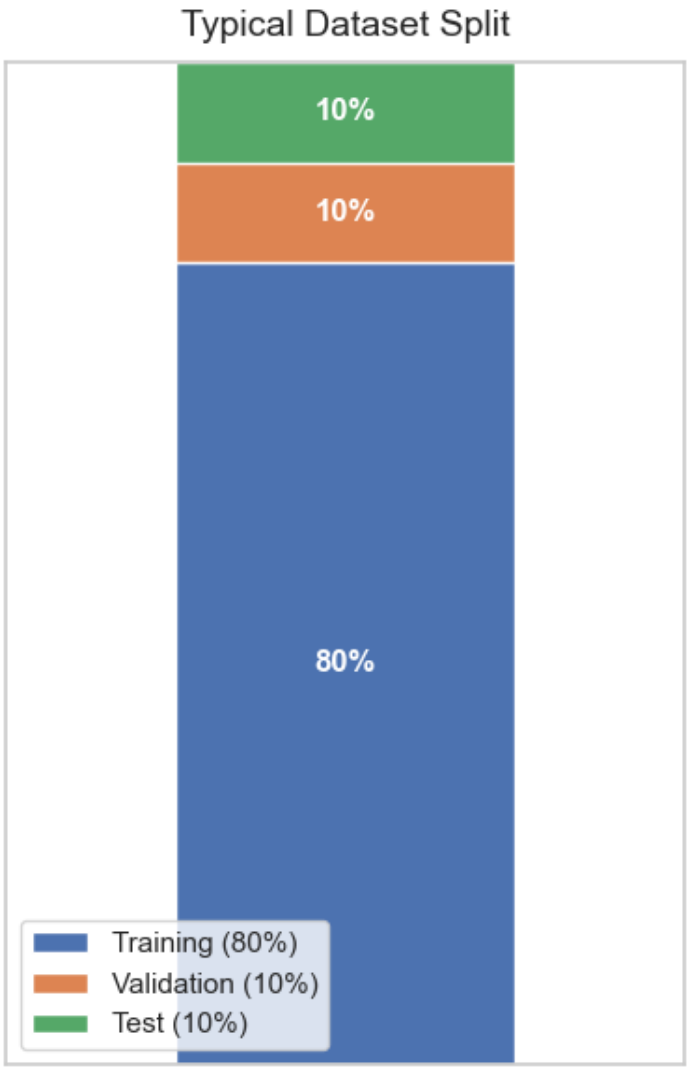
- Fit model parameters
- Monitor training loss for underfitting

Validation Set (10 %)

- Tune hyperparameters & select models
- Early-stop to prevent overfitting

Test Set (10 %)

- Held out until final evaluation
- Provides unbiased measure of performance



Best Practices & Cross-Validation

Best practices

- **Shuffle** data before splitting (unless time-series)
- **Stratify** splits for imbalanced labels
- **Scale/normalize** using statistics from training set only
- **Document** split ratios and random seed for reproducibility
- **Keep test set untouched** until the very end

K-Fold Cross-Validation (CV)

1. Partition data into K equal folds (e.g. K=5 or 10)
2. Rotate: train on K–1 folds, validate on 1-fold
3. Aggregate validation metrics across folds

When to use:

- Small datasets (reduce sampling noise)
- Robust estimate of model performance

Reminder: Always reserve a final test set for the **unbiased** evaluation after CV

Coffee Break

Time to put everything into code

START CODING SESSION

What You'll Do in Code Today

1. **Correlation Analysis & Matrix.** Load the dataset, compute Pearson correlations between all feature pairs, and visualize the resulting correlation matrix to identify multicollinearity or particularly strong linear relationships.
2. **Train/Validation/Test Split.** Partition your data into training, validation, and test sets—shuffling as appropriate or applying a time-series split—to ensure robust evaluation.
3. **Random Forest Implementation.** Instantiate and train a scikit-learn RandomForestRegressor, tune key hyperparameters (e.g. number of trees, max depth), and record its validation performance.
4. **LightGBM Implementation.** Set up a LightGBM dataset, train a GradientBoosting model with LightGBM's Python API, adjust learning rate and tree parameters, and log its metrics on the validation set.
5. **Performance Comparison.** Evaluate both Random Forest and LightGBM on the held-out test set using MAE and RMSE, then directly compare these non-linear results to the linear models from Lecture 7.
6. **Feature Importance Visualization.** Extract feature-importance scores from each tree-based model, plot them side-by-side, and discuss which predictors drive the best performance.

Takeaways

- ✓ **Non-linear models unlock richer patterns.** Tree ensembles (Random Forest, LightGBM) often outperform linear regression when relationships are complex or exhibit thresholds.
- ✓ **Correlation is just a first filter.** A high Pearson coefficient signals linear relevance, but non-linear dependencies require flexible models.
- ✓ **Robust evaluation matters.** A clear train/validation/test split and careful hyperparameter tuning guard against both under- and over-fitting.
- ✓ **Interpretability through importance.** Feature-importance metrics in ensemble methods help translate “black-box” predictions into actionable insights.

Further Questions to Think About

- ➢ When might you still choose a simple linear model in production, despite lower accuracy?
- ➢ How could different data-splitting strategies (e.g., time-series cross-validation) change your evaluation?
- ➢ What are the limitations of permutation- or gain-based importance in ensemble models?
- ➢ How might you combine domain expertise with automated feature selection to improve forecasting?