31: Time Series – Machine Learning Approach

Forecasting as a supervised learning problem

Where we left off

Last time: ARIMA, SARIMA, exponential smoothing – purpose-built statistical models. They are excellent, and often hard to beat on a single clean series.

Today's idea: we already own a toolbox of powerful regressors – linear models, random forests, gradient boosting, neural nets. What if we could point *those* at a time series?

We can. The trick is to turn "predict the next value" into an ordinary $(\mathbf{X}, \mathbf{y})$ table. Once it's a table, every model from this course applies.

The catch: the i.i.d. assumption is still false, so we have to be very careful about *how* we build the table and *how* we validate. That discipline is most of this lecture.

Outline

Forecasting as supervised learning

Slide a window over the series and out falls a design matrix.

The reframing

Pick a target "now" (y_t) and describe it with values from its own past ($y_{t-1}, y_{t-2}, \ldots$). Each time step becomes one training row.

Forecasting IS supervised learning: sliding a window makes (X, y) rows

y(t-3)	y(t-2)	y(t-1)	y(t) = target
108	114	121	118
114	121	118	120
121	118	120	116
118	120	116	108
120	116	108	99
116	108	99	86

That's it – the columns on the left are features, the right column is the label. Feed it to `LinearRegression`, `RandomForest`, `HistGradientBoosting` – whatever you like.

Leakage and time-aware splitting

The single most common way to fool yourself with time series.

Predict-first: what's wrong with a random split?

You have your $(\mathbf{X}, \mathbf{y})$ table. Instinct says `train_test_split(X, y, shuffle=True)`.

Predict: what does a random split do to a time series?

Predict-first: what's wrong with a random split?

You have your $(\mathbf{X}, \mathbf{y})$ table. Instinct says `train_test_split(X, y, shuffle=True)`.

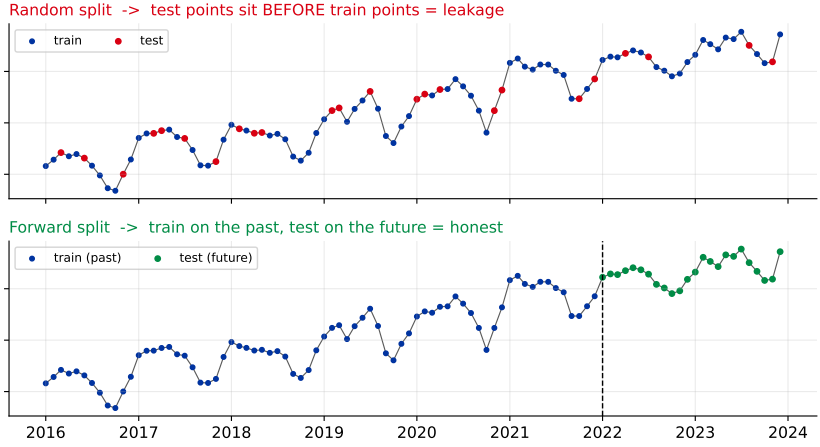
Predict: what does a random split do to a time series?

It puts *future* rows in the training set and *past* rows in the test set. The model peeks at the future to "predict" the past. Your test score is a fantasy.

Worse, the rows aren't even independent: y_t and y_{t-1} share almost everything, so a random test point sits right next to its own neighbours in train. Textbook **data leakage**.

The fix is boring and non-negotiable: **split by time**.

Random split vs forward split



Train on an earlier block, test on the later block. Same rule as lecture 30: only ever learn from the past. This also means **no shuffling**, and any scaler / encoder must be fit on train only.

Feature engineering for time

This is where an ML forecaster is won or lost.

The four families of time features

1. Lags – past values.

- `y.shift(1)`, `shift(7)`, `shift(12)`.
- Pick lags at meaningful horizons (yesterday, last week, last year).

2. Rolling / window stats – local summaries.

- `y.shift(1).rolling(7).mean()` / std / min / max.
- **Always shift before rolling** – else the window includes y_t itself = leakage.

3. Calendar / date parts – known in advance.

- month, day-of-week, hour, is_weekend, is_holiday.
- Cyclically encode: $\sin / \cos(2\pi \cdot \text{hour}/24)$.

4. Exogenous regressors – outside drivers.

- price, promotion, weather, events.
- The big advantage of ML over plain ARIMA.

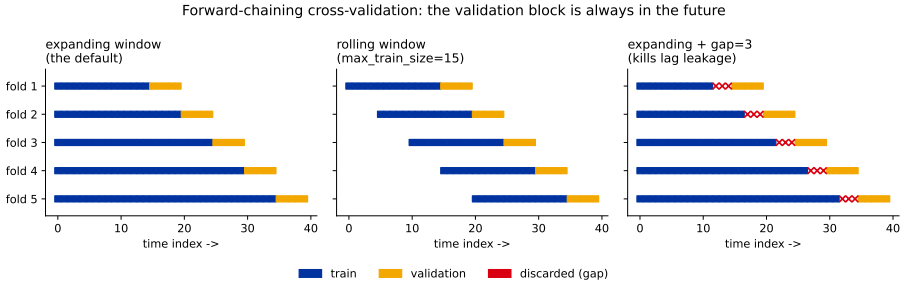
Golden rule: every feature for row t must be computable using *only* information available strictly before t.

Time-aware cross-validation

K-fold, but respecting the arrow of time.

TimeSeriesSplit

Ordinary K-fold shuffles – forbidden here. Use forward-chaining: each fold trains on an expanding past and validates on the next block.




```
TimeSeriesSplit(n_splits=5,
                gap=3)
```

`max_train_size` – fix the window instead of growing it, when the distant past is stale.

`gap` – drop the rows straddling the boundary. A `rolling(7)` feature at the start of validation was built from the last training days, so the two sets *overlap* even after an honest split.

Tree models and the extrapolation trap

Gradient boosting is the tabular champion – with one sharp edge.

Predict-first: point gradient boosting at a trending series

Gradient boosting – a **Gradient Boosting Machine**, **GBM** on the charts that follow – wins most tabular competitions (XGBoost / LightGBM / `HistGradientBoosting`). We built lag + calendar features. Let it forecast the trending series from lecture 30.

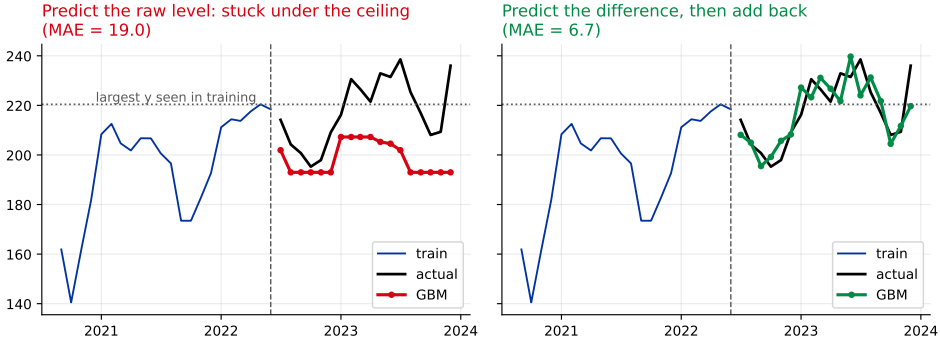
Predict: does the boosting model track the upward trend?

Predict-first: point gradient boosting at a trending series

Gradient boosting – a **Gradient Boosting Machine**, **GBM** on the charts that follow – wins most tabular competitions (XGBoost / LightGBM / `HistGradientBoosting`). We built lag + calendar features. Let it forecast the trending series from lecture 30.

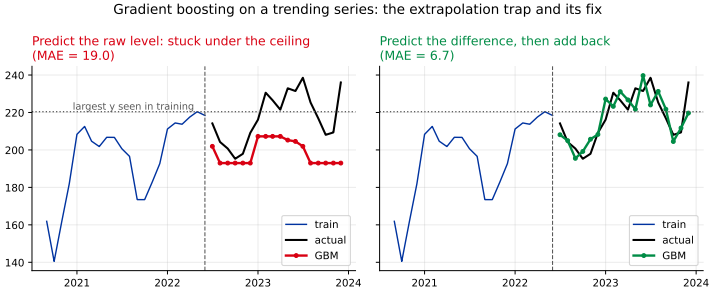
Predict: does the boosting model track the upward trend?

Gradient boosting on a trending series: the extrapolation trap and its fix



Both panels are 1-step-ahead: the model is handed the TRUE previous value each month. Even so, the raw-level tree cannot climb.

Trees cannot extrapolate – so difference first

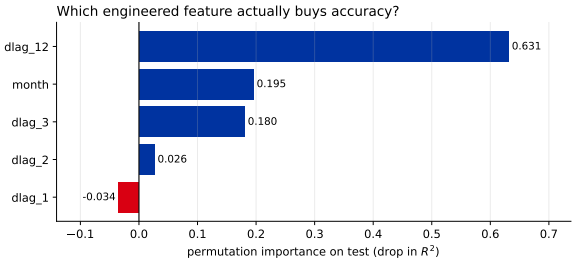


A tree predicts by *averaging training labels* in a leaf, so its output can never exceed the largest y it ever saw – and it settles well *below* that ceiling, at the mean of its top leaf (≈ 200, against a training max of 220).

Same lesson as ARIMA's "I". Linear models and neural networks extrapolate fine; *trees do not*.

The fix: make the *target* stationary – predict the difference $y_t - y_{t-1}$, then add it back. Now the targets live in a fixed range trees can handle: MAE 19.0 $\rightarrow$ 6.7.

Which features actually mattered?



Permutation importance, scored on the **test** split (lecture 23's rule – on train it reports what the model *leans* on, on test what actually *buys* accuracy):

- `dlag_12`, the seasonal lag, dominates at $+0.63$.
- `month` and `dlag_3` split the rest.
- `dlag_1` is **negative** (-0.03).

A *negative* importance means shuffling the column **helped** – last month's change was noise, and the model is better off without it.

Domain knowledge beats brute force: knowing the season is 12 tells you which lag to build.

Forecasting more than one step ahead

For horizon $h > 1$ you have two strategies:

Recursive – predict $t{+}1$, feed it back in as a lag, predict $t{+}2$, ...

- One model, reused.
- Errors compound down the horizon.

Direct – train a separate model per horizon ($h{=}1$ model, $h{=}2$ model, ...).

- No error feedback.
- More models to train and store.

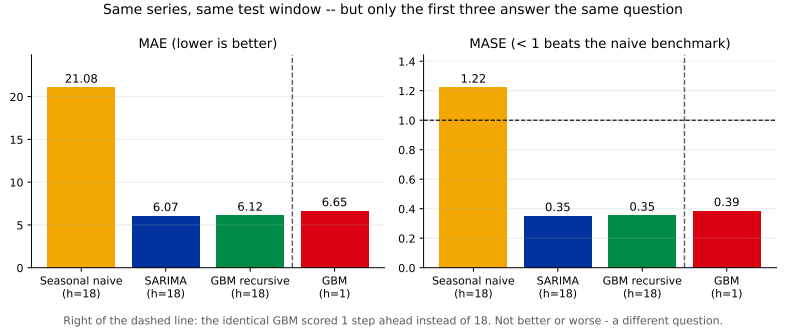
You rarely hand-code this. Libraries like `skforecast`, `mlforecast` (Nixtla), `sktime` and `Darts` wrap any scikit-learn regressor into a recursive/direct forecaster with the leakage-safe plumbing built in.

So which approach wins?

Classical vs ML – the honest answer.

Benchmark everything against the naive baseline

Three contenders, one held-out block, and – this is the part people skip – **one stated horizon**: 18 months, forecast from a single origin.



A properly specified SARIMA (MASE 0.35) and the gradient booster (0.35) are **dead level**, both far past seasonal naive (1.22). On one short, clean series the classical model was never the underdog.

Say your horizon out loud

The red bar is the *same trained model* as the green one. The only difference is what we asked it.

$h = 18$ **(recursive)** – forecast 18 months from one origin, feeding each prediction back in as the next lag. Nothing observed after the origin. MASE **0.35**.

$h = 1$ – forecast one month, then get told the true value, then forecast the next. MASE **0.39**.

Neither is "the" accuracy of the model. They answer different business questions – "plan next year's capacity" versus "restock tomorrow" – and a model can be excellent at one and useless at the other.

The failure mode: quoting a 1-step number and a competitor's 12-step number on the same chart. It is not a lie about either model, and it is still meaningless.

We still owe you an interval

Lecture 30 was blunt: *a point forecast without an interval is only half an answer*. SARIMA hands you one for free – it is a probability model. A gradient booster hands you a single number.

Two ways to get one anyway:

- **Quantile regression** – train three boosters with a pinball loss at $q = 0.05, 0.5, 0.95$ (`LGBMRegressor(objective="quantile", alpha=0.95)`).
- **Conformal prediction** – fit once, measure your errors on a held-out calibration block, and use their empirical quantiles as the band. Model-agnostic, and distribution-free.

Both need the calibration block to sit *after* training and *before* test – the arrow of time again. And both give you the width *at a stated horizon*: a 1-step band is far narrower than an 18-step one, so a single "±" number is meaningless without h.

Libraries: `MAPIE`, `neuralforecast`'s conformal wrapper, `Darts`. Connects straight back to lecture 14 – an interval you do not check the coverage of is decoration.

When to reach for ML vs classical

Classical (ARIMA / ETS) shines

- one or a few series,
- short history,
- strong, stable seasonality,
- you need calibrated prediction intervals cheaply.

ML (boosting / NN) shines

- many related series (retail: 10k products),
- long history,
- rich exogenous features (price, weather, promos),
- non-linear interactions between drivers.

In the **M5** competition (Walmart sales, thousands of related series), gradient-boosted trees on engineered features won. On the earlier **M3**/**M4** single-series benchmarks, statistical methods were extremely strong. Match the tool to the shape of the problem.

But the competitions gave a better answer than "it depends"

M4 (2018), 100,000 series.

- **1st – Slawek Smyl (Uber): ES-RNN.** Exponential smoothing for level and season, its parameters learned *inside* a recurrent neural network. Not statistics *or* ML – one model, both.
- **2nd – FFORMA** (Montero-Manso, Athanasopoulos, **Hyndman**, Talagala). Gradient boosting, but it does not forecast: it looks at features of each series and predicts *how to weight* a pool of classical models.

Neither winner picked a side. One put a statistical model inside a neural network; the other used ML to *referee* statistical models.

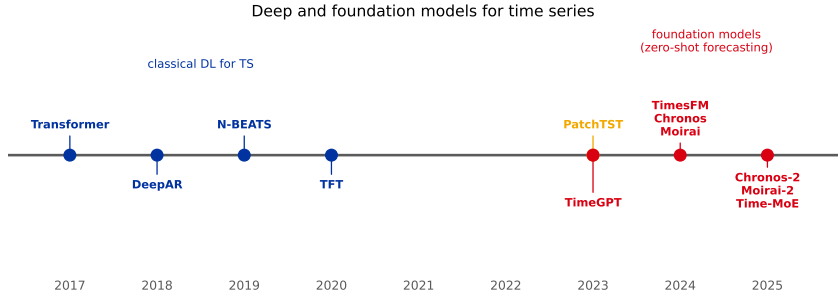
And the runner-up's third author wrote the textbook this chapter is built on. The classical-vs-ML framing is a teaching device, not a real dividing line.

Practical version of the same idea: average your SARIMA and your booster. Forecast combinations are one of the most reliably reproduced results in the field.

Deep learning and foundation models

Awareness level – what exists and when it's worth it.

The deep learning lineage



Specialist deep models

- **DeepAR** – a **recurrent** network (it carries a hidden state along the sequence), trained across many series, predicting a *distribution* not a point.
- **N-BEATS / N-HiTS** – deep stacks of plain layers.
- **Temporal Fusion Transformer** (**TFT**) – attention + built-in importances.
- **PatchTST** – patch the series, feed a Transformer.

Why they connect to us

Recurrence and attention are the subject of two later chapters, so treat these as a map, not a syllabus. What matters today: they need a *lot* of data – many series, long history – to beat a well-tuned gradient booster. On a single short series like ours, they usually don't.

Foundation models: zero-shot forecasting

Since 2023: pre-train one Transformer on billions of time points, then forecast a *new* series with no training at all – the same trick a **Large Language Model** (**LLM**) does when it answers a question it was never specifically trained on.

The current crop (mid-2026)

- **TimesFM** (Google) – patched decoder, pre-trained on ∼100B points.
- **Chronos-2** (Amazon) – tokenizes values like language; five sizes from 9M to 710M parameters; handles covariates.
- **Moirai-2** (Salesforce) – multivariate, any frequency.
- **TimeGPT** (Nixtla), **Sundial**, **Time-MoE**, **Lag-Llama**.

Don't take a vendor's word for it – compare on a public leaderboard (**GIFT-Eval**, **ProbTS**) *and* on your own series.

Reality check. Zero-shot is astonishing for a cold start with no history. But a gradient booster with good features and exogenous drivers is still a very strong, cheap, interpretable baseline – and today it beat a tuned SARIMA by nothing at all. Try the simple thing first.

This slide dates fastest of anything in the chapter. Check the names before re-lecturing.

Recap – ML for time series

- Forecasting $\rightarrow$ supervised learning by sliding a window.
- **Never** random-split; split and cross-validate by time (`TimeSeriesSplit`, with a gap if your features roll).
- Features: lags, (shifted) rolling stats, calendar, exogenous.
- Trees can't extrapolate – difference the target first.
- Multi-step: recursive vs direct (let a library handle it).
- Report MASE vs an *honest* baseline, and **state the horizon**.
- Point forecast $\neq$ answer: quantile or conformal intervals.
- The M4 winners were **hybrids**, not one side or the other.

The one habit to keep:

every feature and every split must respect the arrow of time. If a value wasn't knowable yet, it can't be in the row.

Toolbox: `sktime`, `statsforecast` & `mlforecast` (Nixtla), `skforecast`, `Darts`, `Prophet`, `GluonTS`.

Questions?

Homework: build a forecaster the leakage-safe way. See the chapter page.