

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.

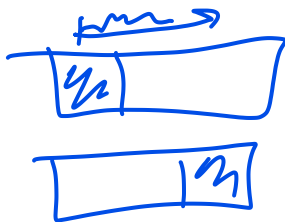
Today's idea: we already own a toolbox of powerful regressors – linear models, random forests, gradient boosting, ~~neural nets~~. What if we could point *those* at a time series?

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
- Leakage and time-aware splitting
- Feature engineering for time
- Time-aware cross-validation
- Tree models and the extrapolation trap
- Industry standards
- Deep learning and foundation models
- Recap



Vaxo jan jamy cheir asi?

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 ($y_{t-1}, y_{t-2}, \dots$). 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)`.

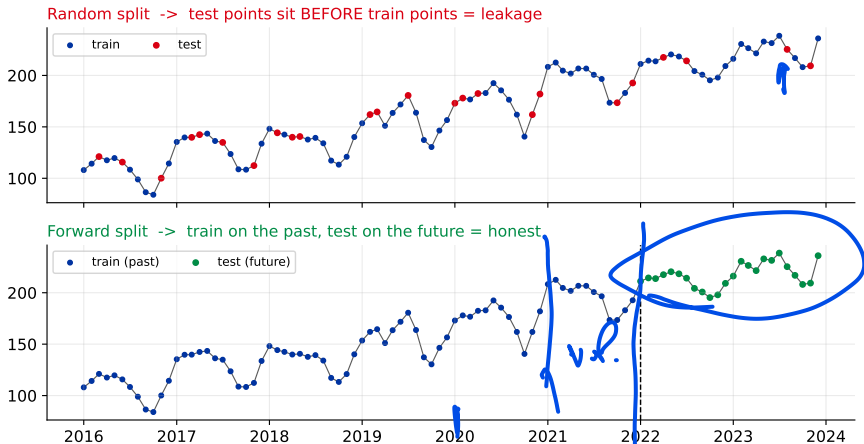
Predict: what does a random split do to a time series?

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.

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

3. Calendar / date parts – known in advance.

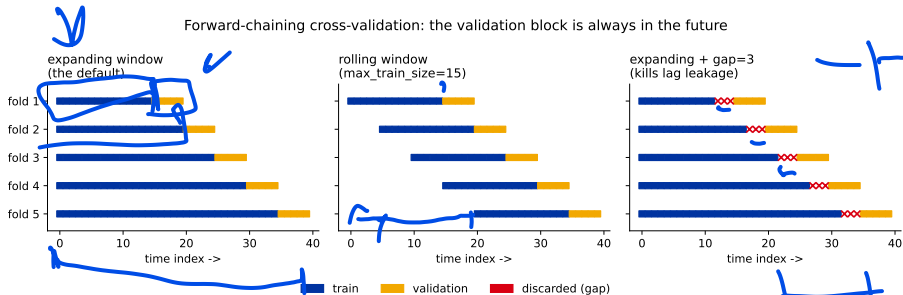
- ▶ month, day-of-week, hour, is_weekend, is_holiday.
- ▶ Cyclically encode: $\sin / \cos(2\pi \cdot \text{hour}/24)$.

4. Exogenous regressors – outside drivers.

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

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.

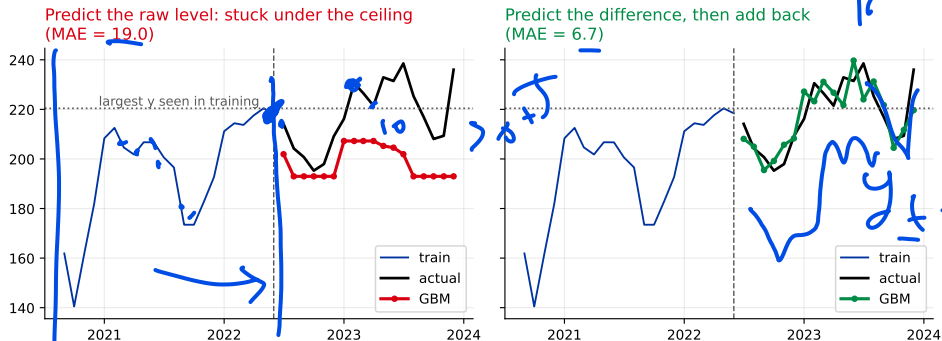
Predict: does the boosting model track the upward trend?

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?

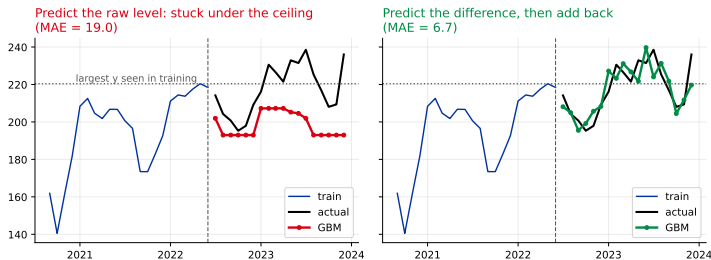
Gradient boosting on a trending series: the extrapolation trap and its fix



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

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.

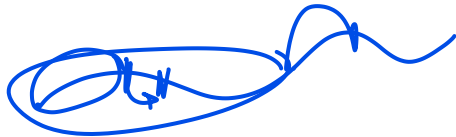
A tree predicts by *averaging training labels* in a leaf, so its output can never exceed the largest y it ever saw – and it settles well *below* that ceiling, at the mean of its top leaf (≈ 200 , against a training max of 220).

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 $19.0 \rightarrow 6.7$.

ARIMA

Forecasting more than one step ahead



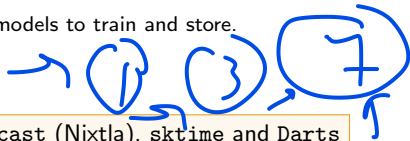
For horizon $h > 1$ you have two strategies:

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.



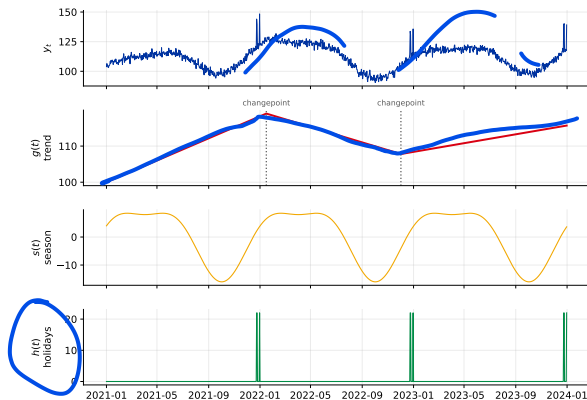
You rarely hand-code this. Libraries like `skforecast`, `mlforecast` (Nixtla), `sktime` and `Darts` wrap any scikit-learn regressor into a recursive/direct forecaster with the leakage-safe plumbing built in.



What people actually run

The tools you will meet in a real forecasting job.

Prophet: a curve, not a process



Prophet (Taylor & Letham, 2018 – Meta)
fits an additive, decomposable model:

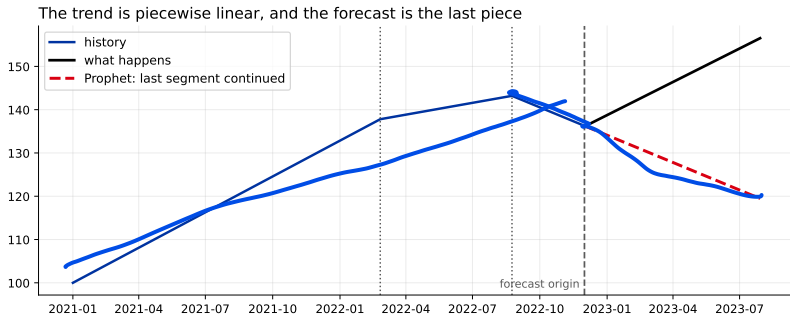
$$y_t = g(t) + s(t) + h(t) + \varepsilon_t$$

- ▶ $g(t)$ – trend, piecewise linear
- ▶ $s(t)$ – seasonality, Fourier terms
- ▶ $h(t)$ – holidays, dummy variables

The break with everything so far. ARIMA and ETS model the *process* – how today depends on yesterday. Prophet regresses on **time itself**. There is not a single lag in it.

That has a practical payoff: gaps, irregular spacing and outliers do not break it, because row t never needs row $t - 1$.

The trend is piecewise linear – and that is the sharp edge

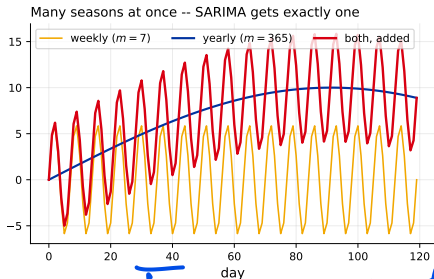
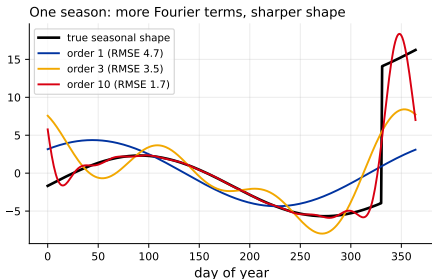


Prophet lays down **25 candidate changepoints** across the first **80%** of the history, then a sparse prior shrinks most of them to zero so only a few survive. How eagerly is set by `changepoint_prior_scale` (default 0.05).

The forecast is the last segment, extended. Above, the final segment slopes $-0.07/\text{day}$, so the forecast keeps falling while the series turns back up – finishing **37 units** low. A recent blip becomes your long-run view.

Fourier seasonality: many seasons at once

$F_{b \sim}$



$s(t)$ is a sum of sin / cos pairs. Its **order** sets how sharp a shape it can bend into: on the left, RMSE $4.7 \rightarrow 3.5 \rightarrow 1.7$ for orders 1, 3, 10. Defaults are order 10 yearly, 3 weekly. Holidays are separate dummies, each with its own coefficient.

This answers a limit lecture 30 left open. SARIMA carries *one* m . Fourier terms are just regressors, so daily + weekly + yearly can all sit in one model at once.

$\sin \dots$

P

The honest verdict on Prophet

What it is genuinely good at

- ▶ Fast and fully automatic – a sane forecast with no tuning.
- ▶ Missing data, outliers, irregular spacing.
- ▶ Multiple seasonalities and holiday calendars.
- ▶ Components you can *show* a stakeholder.

Built at Meta so that analysts who are *not* forecasters could produce hundreds of forecasts. That is the problem it solves.

What the textbook says. Hyndman & Athanasopoulos (*fpp3* §12.2) find Prophet worse than **both** ETS and ARIMA on their example, with enough left-over residual autocorrelation to leave its prediction intervals **too narrow**. Their verdict: it rarely gives better forecast accuracy than the alternatives.

So: a strong, cheap **default** – not a winner. Same rule as everything else here, score it against seasonal naive before you believe it.



The rest of the production stack

Library	Reach for it when
statsforecast	classical models, many series, fast
mlforecast	sklearn regressor + lag features
neuralforecast	deep models behind one API
skforecast	recursive/direct around sklearn
sktime	sklearn-style pipelines, unified API
Darts	one API over classical + deep
GluonTS	probabilistic / deep (AWS)
Prophet	business series, holidays, no tuning

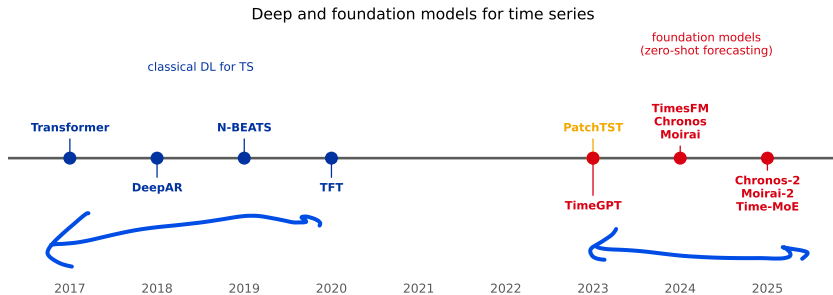
The first three are **Nixtla's** stack and share one API. Nixtla's own benchmarks put statsforecast's AutoARIMA orders of magnitude ahead of Prophet on speed – a vendor number, so check it on your own series, but the gap is real and it is why the stack spread.

None of these change the rules: time-ordered splits, features that were knowable at t , and an honest baseline.

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.

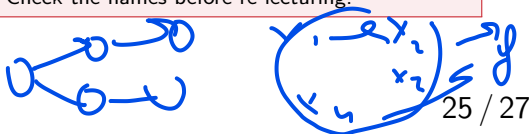
The current crop (mid-2026)

- ▶ **TimesFM** (Google) – patched decoder, pre-trained on $\sim 100\text{B}$ 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.



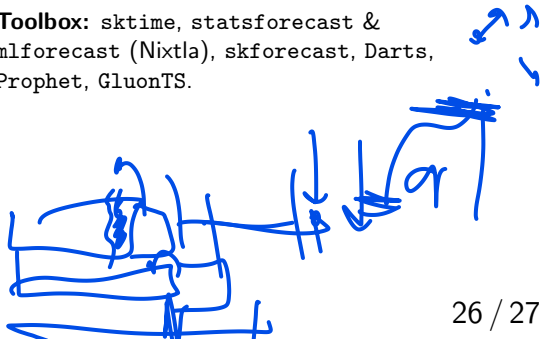
Recap – ML for time series

- ▶ Forecasting → 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 (lecture 30's rule still binds).
- ▶ Prophet regresses on *time* – piecewise trend + Fourier seasonality + holidays. Automatic, multi-seasonal, and rarely the most accurate.

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.