29: Time Series – Classical Methods

When the order of the rows is the whole point

Every model so far assumed the rows were interchangeable

Regression, trees, logistic regression – all of them treat the dataset as a *bag* of i.i.d. rows. Shuffle the rows, nothing changes.

Time series breaks that assumption. Row order *is* the signal. Yesterday's value tells you a lot about today's. You can only train on the past, and you must predict the future – never the other way around.

Examples everywhere: daily sales, electricity demand, website traffic, the dram-to-dollar exchange rate, sensor readings, COVID case counts, your heart rate.

Two lectures. **Today (29):** the classical statistical toolkit – decomposition, stationarity, ARIMA, exponential smoothing. **Next (30):** treating forecasting as a supervised ML problem.

Outline

Vaxo jan jamy cheir asi?

What makes time series different

One column, indexed by time – and the index carries information.

The vocabulary

A time series is a sequence $y_1, y_2, \ldots, y_T$ sampled at regular steps (hourly, daily, monthly, ...).

Two jobs:

- **Forecasting** – predict future values $y_{T+1}, \ldots, y_{T+h}$. (h = forecast *horizon*.)
- **Understanding** – decompose, explain, find structure.

Two shapes of data, and two kinds of input:

- **Univariate** – one series, explained by its own past: the **endogenous** input.
- **Multivariate** – several series, or outside drivers: **exogenous** input.

The one rule that governs everything:
information flows forward in time only. Any value, feature, or statistic you feed the model must have been *knowable at prediction time*. Violate this and you get "leakage" – great backtests, useless in production.

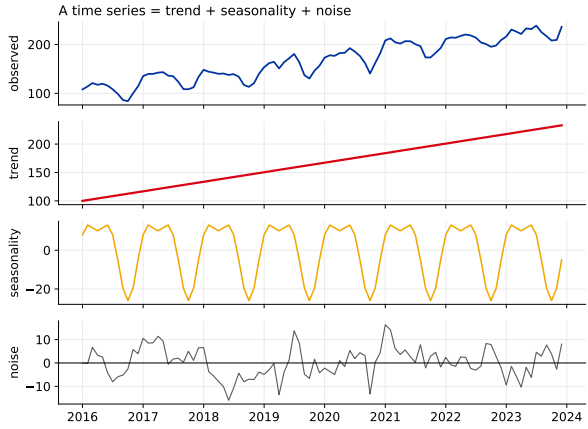
Exogenous = "generated outside": temperature for electricity demand, price for sales, a holiday calendar for traffic. Everything in lecture 29 is *endogenous*; exogenous drivers arrive in 31. **The catch:** to use one you need its *future* value too, and a temperature forecast is itself a forecast.

The four components of a series

A series is usefully thought of as a sum (or product) of:

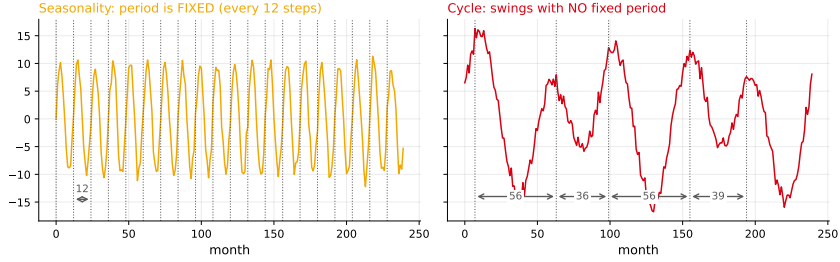
- **Trend** T_t – long-run drift up or down.
- **Seasonality** S_t – a pattern that repeats on a *fixed* period (24h, 7d, 12mo).
- **Cycle** – swings with *no* fixed period (business cycles, 2–10 years). **Not in the figure**, and not modelled separately: there is no calendar to hang it on, so it gets absorbed into T_t or R_t. Next slide.
- **Noise / residual** R_t – what's left.

Additive: $y_t = T_t + S_t + R_t$.
Multiplicative: $y_t = T_t \cdot S_t \cdot R_t$ (season grows with level – take log to make it additive).



The figure shows the three components this series actually has.

Season or cycle? Ask whether you can put it on a calendar



You can put a season on a calendar. You cannot put a cycle on one - which is why only the season gets modelled explicitly.

Left – seasonality. Fixed, *known* period. Next December is 12 months after the last one, and you knew that before seeing any data.

Right – a cycle. The gaps between peaks are $56, 36, 56, 39$ months. Obvious in hindsight, impossible to schedule.

Every seasonal tool here needs a *number* m – seasonal differencing, SARIMA's m, Holt-Winters' seasonal term. A cycle has no m to give them. **The trap:** fitting $m = 56$ to the right-hand panel would confidently schedule a peak that never arrives.

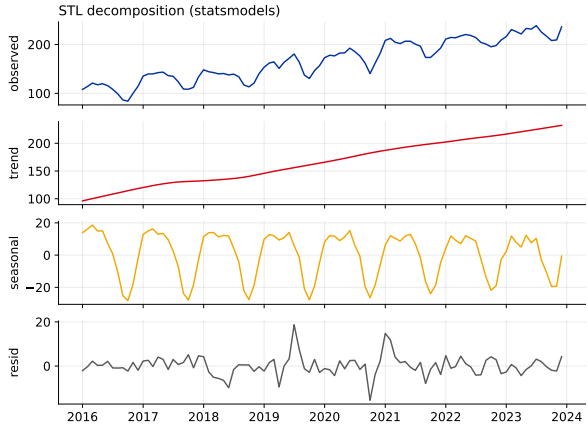
Decomposition: STL

STL = "Seasonal-Trend decomposition using Loess". It splits the series into trend + seasonal + residual, robustly.

Loess = *locally estimated scatterplot smoothing*: slide a window along the data, fit a tiny regression inside each window, keep only its value at the centre, and join those into a curve. No global formula, so it bends to whatever shape the data has.

Why bother: see the structure before modelling, deseasonalize to read the trend cleanly, and the residual is where anomalies show up.

```
from statsmodels.tsa.seasonal import STL
res = STL(y, period=12).fit()
res.trend, res.seasonal, res.resid
```



Stationarity and differencing

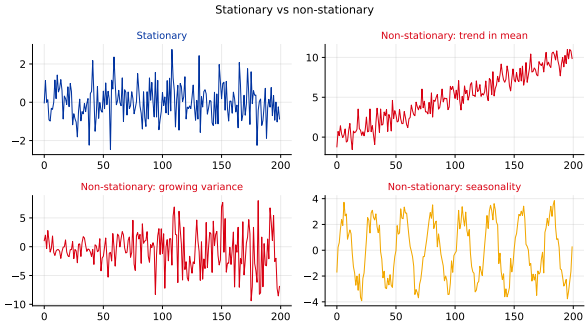
The classical models need a series whose behaviour does not drift.

What is stationarity?

A series is (weakly) **stationary** if its statistical behaviour does not depend on *when* you look:

- constant mean (no trend),
- constant variance (no growing swings),
- the link between two points depends only on *how far apart* they are, never on *when* they are.

That third condition has a name: the **autocorrelation** at each lag must not drift over time. We measure it two sections from now.



Why we care: ARMA-type models assume stationarity, and for a good reason – if the relationships keep changing, a model fitted on the first half is describing a series that no longer exists in the second. Trends and seasonality come out *first*, then get added back to the forecast.

Differencing makes a series stationary

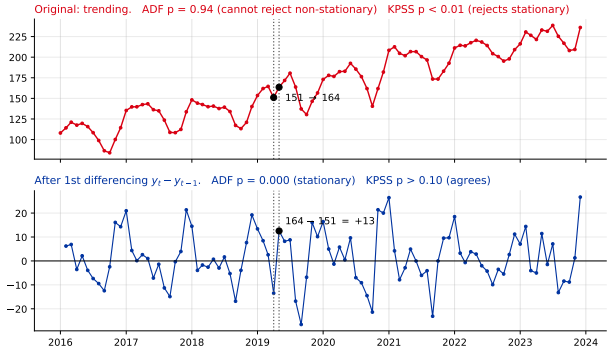
Differencing: model the change instead of the level.

$$y'_t = y_t - y_{t-1}$$

- Removes a trend.
- **Seasonal differencing** $y_t - y_{t-m}$ removes a season of period m.

Testing it – two tests, opposite nulls (small p rejects, so they point opposite ways):

- **Augmented Dickey-Fuller** (**ADF**) – H_0: non-stationary.
- **Kwiatkowski-Phillips-Schmidt-Shin** (**KPSS**) – H_0: stationary.



What does "difference d times" actually mean?

d is a **count**: how many times you apply the difference operator, one after another. Nothing more mysterious than that.

Why once is normally enough. Let the trend be a straight line, $y_t = a + b\,t$:

$$y_t - y_{t-1} = (a + bt) - \big(a + b(t-1)\big) = b$$

The line is gone in a single pass, and a *constant* is left.

When you need $d = 2$. If the slope itself grows, $y_t = a + bt + ct^2$, one difference leaves $b + c(2t-1)$ – still a line in t. Difference again and you get $2c$. Hence $d = 2$.

In practice $d \in \{0, 1, 2\}$. Wanting $d = 3$ almost never means a trend – it means a variance that grows (take log) or a one-off level shift (model the break).

Seasonal differencing is counted *separately*, as D at lag m. One of each, $(1 - B)(1 - B^{12})$, is the usual monthly combination.

Stop as soon as the test passes. On our series the standard deviation goes $40.9 \to 10.8 \to \mathbf{11.7}$. The second difference had no trend left to remove, so it differenced the *noise* – and the difference of noise is noisier still. Over-differencing costs accuracy and bolts on a spurious MA term.

The ARIMA family

The workhorse of classical forecasting.

AR(p): regress the series on its own past

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t$$

AR = **autoregressive**. **Every symbol, named:**

- y_{t-i} – the value i steps ago. *Observed*: it is sitting in your data.
- ϕ_i (**phi**) – the **weight** on that past value. This is what gets fitted. $\phi_1 = 0.8$ reads as "today starts at 80% of yesterday".
- c – a constant, setting the level the series returns to.
- ε_t (**epsilon**) – the **shock** at time t: the part of today the past could not predict. Mean 0, constant variance, uncorrelated. All the randomness enters here.

One step, by hand. AR(1) with $c = 2$, $\phi_1 = 0.8$. Yesterday $y_{t-1} = 10$ and today's shock $\varepsilon_t = -1$ give

$$y_t = 2 + 0.8(10) - 1 = \mathbf{9}$$

Then with $y_t = 9$ and a shock of $+3$: $y_{t+1} = 2 + 0.8(9) + 3 = \mathbf{12.2}$.

ε **is not something the fitting picks:** only c and the ϕ's are chosen, and ε_t is whatever is left over.

Next slide: how.

What ϕ's size means: $|\phi| < 1$ pulls back toward the mean (stationary); $\phi = 1$ is a random walk, shocks *never* fade; $|\phi| > 1$ explodes.

AR: it really is just least squares

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \varepsilon_t$$

Everything on the right is **in your data**. So lay it out as a table:

y_{t-2}	y_{t-1}	y_t
112	118	132
118	132	129
132	129	121
⋮		

That is a design matrix, and this is *exactly* the reframing lecture 30 opens with. Then it is **ordinary least squares** (**OLS**): pick ϕ to minimise $\sum_t (y_t - \hat{y}_t)^2$, closed form, no search:

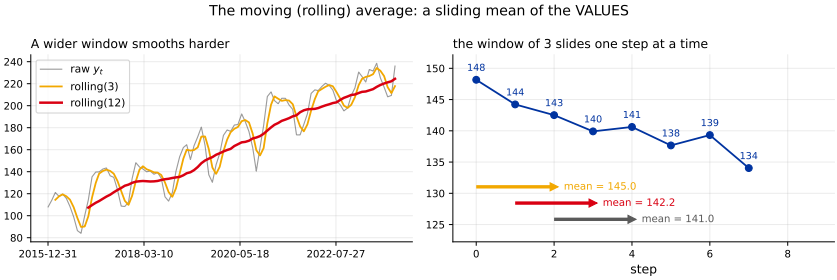
$$\hat{\phi} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

Measured on our series. Plain OLS on one lag column gives $\phi = \mathbf{0.9686}$. `statsmodels`' exact maximum likelihood gives **0.9749**. A gap of 0.006.

Why any gap at all? OLS quietly throws away the first observations – it can only start once it has p lags in hand. Exact likelihood models those startup values too. With $T = 96$ this barely matters; with $T = 20$ it can.

A second closed-form route, **Yule-Walker**, solves for ϕ straight from the autocorrelations you plotted earlier. The correlogram is not only a diagnostic – it can hand you the coefficients.

Interlude: the moving average you already know



Replace each point by the mean of the last k values:

$$\mathrm{MA}_3(t) = \tfrac{1}{3}\left(y_t + y_{t-1} + y_{t-2}\right)$$

The window slides one step at a time: 145.0, then 142.2, then 141.0. Bigger k smooths harder – $k = 12$ on monthly data erases the season outright, which is roughly how a trend estimate gets made.

A **descriptive smoother**, not a model: no parameters to fit, and it makes no forecast by itself.

Note for lecture 30: `y.rolling(3).mean()` includes y_t itself, so as a *feature* for predicting y_t it is leakage. Shift first: `y.shift(1).rolling(3).mean()`.

MA(q): regress on your own past *mistakes*

$$y_t \;=\; c \;+\; \varepsilon_t \;+\; \theta_1\varepsilon_{t-1} \;+\; \ldots \;+\; \theta_q\varepsilon_{t-q}$$

MA = **moving average**, a historical misnomer (next slide).

- ε_t – the same shock as in AR.
  Unobserved: it is the model's own error, so it does not exist until there is a model.
- θ_j (**theta**) – how much of the surprise from j steps ago is *still echoing* today.

Unrolled, by hand. MA(1), $c = 50$, $\theta_1 = 0.7$, with shocks $\varepsilon = +4, -2, +6, 0$:

$$\begin{aligned} y_1 &= 50 + 4 && = 54 \\ y_2 &= 50 + (-2) + 0.7(4) && = 50.8 \\ y_3 &= 50 + 6 + 0.7(-2) && = 54.6 \\ y_4 &= 50 + 0 + 0.7(6) && = 54.2 \end{aligned}$$

The one-line contrast. AR carries forward the **value**; MA carries forward the **surprise**.

The big surprise at $t = 3$ still lifts $t = 4$, and is then gone *completely* at $t = 5$. An MA(q) has a memory exactly q steps long – a hard cut-off. An AR's memory decays forever without ever quite reaching zero. That difference is what the ACF and PACF are about to expose.

MA: the same trick breaks, and the way around it

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1}$$

Try to build the same table. The first column would have to be ε_{t-1} – and **it is not in your data**. The shocks are the model's own errors, so they do not exist until you already have a model. Circular.

The escape, and it is the whole idea: *fix a candidate θ_1 and the shocks become computable.* Set $\varepsilon_0 = 0$, then walk forward:

$$\varepsilon_t = y_t - c - \theta_1 \varepsilon_{t-1}$$

Each step only needs the ε you just worked out.

So *any* θ_1 produces a full residual series, which you can score by its **sum of squared errors** (**SSE**):

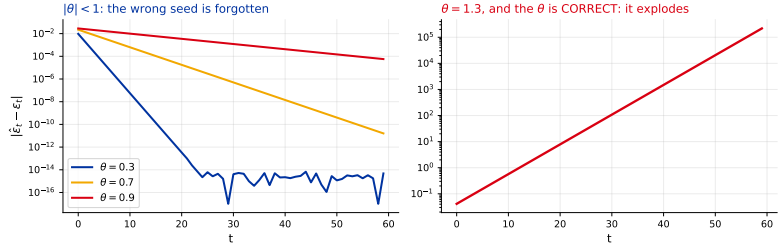
$$\mathrm{SSE}(\theta_1) = \sum_t \varepsilon_t^2$$

Hand that function to a numerical optimiser and ask for the minimum.

This is **conditional maximum likelihood**: when the shocks are Gaussian, maximising the likelihood *is* minimising that sum of squares.

No formula means a *search*, and searches can fail: sensitivity to the starting guess, local minima, non-convergence. That is why `.fit()` sometimes warns, and why the warning is not decoration.

Can we really recover the shocks? And where $|\theta| < 1$ comes from



We seeded that recursion with $\varepsilon_0 = 0$, which is simply *false*. It stops mattering, because each step multiplies the leftover error by θ: at $\theta = 0.7$ the recovery error falls from $2.2 \cdot 10^{-2}$ to $1.4 \cdot 10^{-8}$ by $t = 40$. Closer to 1 is slower – $\theta = 0.9$ is still at $4 \cdot 10^{-4}$.

Now push θ past 1. The same step *multiplies* the error instead: even with the **correct** $\theta = 1.3$ the shocks diverge past 10^5. **Invertibility** ($|\theta| < 1$) is precisely what makes the shocks recoverable – hence the constraint.

And what is not estimated: an MA(1) fit searches **three** numbers – c, θ, σ^2 – however long the series. The ε are **residuals**, not parameters, and they are exactly the one-step-ahead forecast errors – so "minimise $\sum_t \varepsilon_t^2$" and "minimise the one-step error" are one sentence.

Careful: "moving average" means two different things

The MA in ARIMA is not the rolling mean from two slides ago.

The one you already know – a *rolling* or *moving* average: a sliding mean of the **values**, used to smooth a curve.

$$\tfrac{1}{3}\left(y_{t-1} + y_{t-2} + y_{t-3}\right)$$

You compute it *from data you can see*.

MA(q) in ARIMA – a weighted sum of the past **shocks** (the model's own errors):

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \dots$$

The ε's are *unobserved*. You never measure them; the fitting procedure infers them. That is why MA orders cannot be read off the data directly the way a rolling mean can – and why they need the ACF.

Two unrelated ideas, one name, purely for historical reasons. Next lecture we build rolling-mean *features* (`y.shift(1).rolling(7).mean()`) and feed them to a tree – that is the *other* moving average. Keep them apart.

Putting them together: ARMA and ARIMA

ARMA(p, q) – both terms at once, for a series that is *already stationary*:

$$y_t = c + \underbrace{\sum_{i=1}^{p} \phi_i y_{t-i}}_{\text{past values}} + \varepsilon_t + \underbrace{\sum_{j=1}^{q} \theta_j \varepsilon_{t-j}}_{\text{past shocks}}$$

ARIMA(p, d, q) adds the "I" = **Integrated** = difference d times first, so it handles a trending series directly.

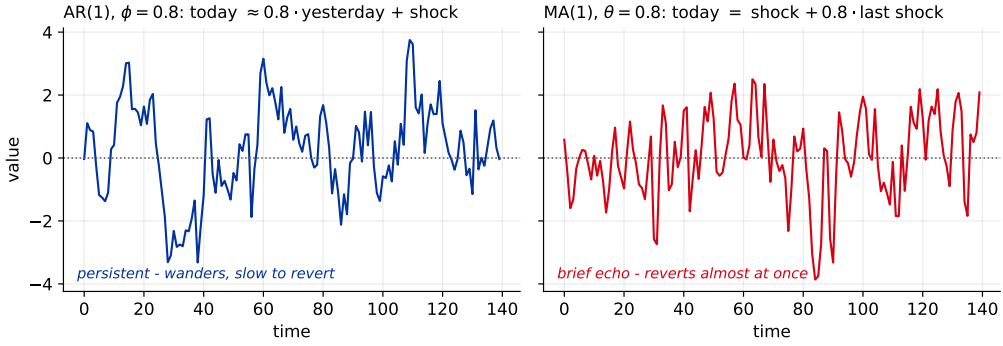
Where each number comes from:

- p – AR order, read from the **PACF**
- d – differencing, from ADF / KPSS
- q – MA order, read from the **ACF**

Those two plots are the next section.

ARIMA(0,1,0) is just the random walk – the "tomorrow = today" baseline. Every fancier model has to beat it.

What AR and MA actually look like



Same machinery, different memory: an **AR** term makes the series *persist* (today leans on yesterday's *value*); an **MA** term is a short *echo* of the last random *shock*. ARMA blends the two.

SARIMA: adding seasonality

Real series are seasonal, so add a second, seasonal triple:

$$\text{SARIMA}(p, d, q)(P, D, Q)_m$$

- lower-case (p, d, q) – the short-term, non-seasonal part;
- upper-case (P, D, Q) – the same idea at the seasonal lag;
- m – periods per season (12 monthly, 7 daily-with-weekly, 24 hourly-with-daily).

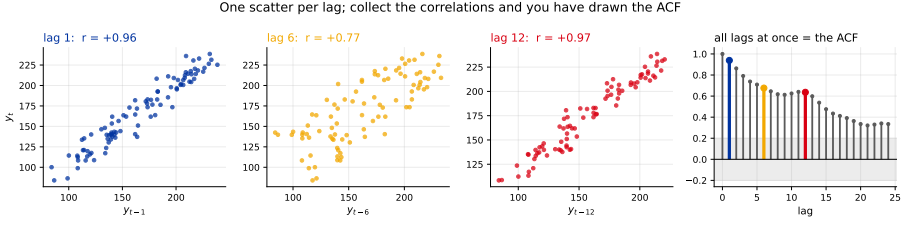
One m only. Hourly electricity demand has a daily *and* a weekly *and* a yearly season – SARIMA can model one of them. That limitation is exactly why Fourier seasonal terms, and models like Prophet, exist.

Autocorrelation

How a series correlates with delayed copies of itself.

Where autocorrelation comes from

Autocorrelation is simply the measurement of "yesterday tells you about today" – the ordinary correlation you already know, applied to the series against a shifted copy of itself.



The recipe: copy the series, shift the copy k steps, line the two up, scatter one against the other, take Pearson's r. That number *is* the autocorrelation at lag k. Repeat for every k, draw one bar per lag, and you have drawn the **autocorrelation function** (**ACF**). Nothing else happens in that plot.

Neighbouring months are nearly identical (+0.96); six months apart is the *weakest* link (+0.77), being half a season away; twelve months apart is as tight as one (+0.97) – the season announcing itself. **What it buys you:** which lags belong in the model.

ACF and PACF: why you need both

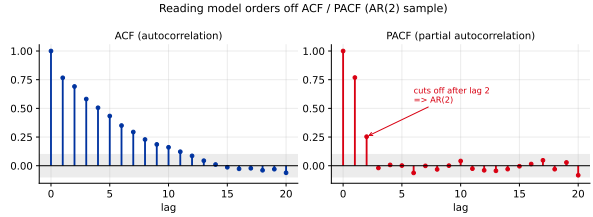
The ACF double-counts. Today depends on yesterday, and yesterday on the day before. So today correlates with the day before *for free*, borrowed through yesterday, even with no direct link. The ACF reports that borrowed correlation.

The **partial autocorrelation function** (**PACF**) is the fix: lag k with lags $1 \dots k-1$ stripped out.

ACF = direct + borrowed; **PACF** = direct only.

How lag 1 gets removed, at $k = 2$: predict y_t from y_{t-1} and keep the residual a_t; predict y_{t-2} from y_{t-1} and keep the residual b_t; then $\mathrm{PACF}(2) = \mathrm{corr}(a_t, b_t)$.

Both ends have had everything y_{t-1} explains **subtracted off**, so what is left cannot travel through lag 1.



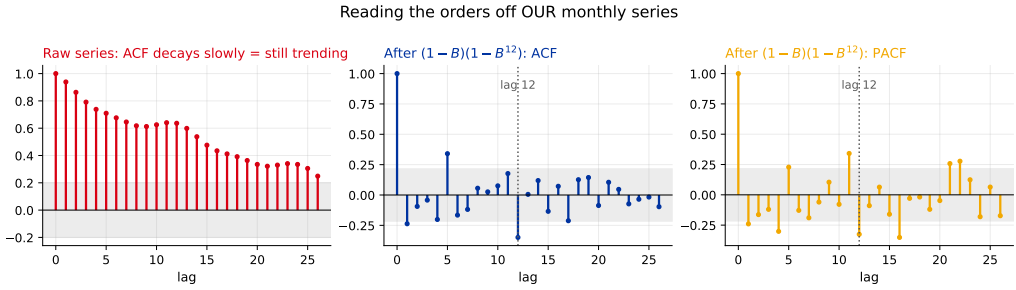
The classic way to read the orders off the data:

- PACF cuts off after lag $p \Rightarrow$ **AR(p)**.
- ACF cuts off after lag $q \Rightarrow$ **MA(q)**.
- Both decay $\Rightarrow$ a mix (ARMA).

That cut-off is exactly the "memory exactly q steps long" from the MA slide.

The shaded band is the "not significantly different from zero" region ($\pm 1.96/\sqrt{T}$).

Choosing the orders – on our own series



Left: the raw ACF decays slowly (lag 1 = 0.94) – classic "still trending", so we need $d \geq 1$.

Middle: after $(1-B)(1-B^{12})$ (B is **backshift**, $B\,y_t = y_{t-1}$) – one ordinary and one seasonal difference – one clear *negative* spike at **lag 12** (-0.35). A single negative seasonal spike in the ACF is the signature of a **seasonal MA** term, so $Q = 1$.

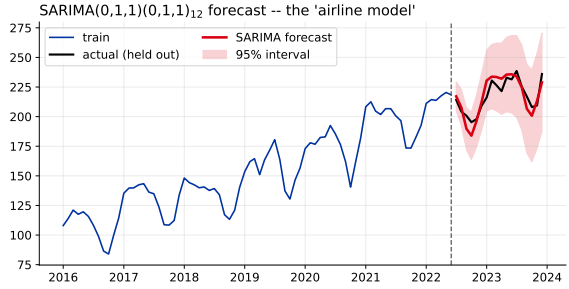
That gives SARIMA$(0,1,1)(0,1,1)_{12}$ – known as the **airline model**, because Box and Jenkins fitted exactly this to monthly airline passengers.

Did reading the plot actually help?

We could have guessed a seasonal *AR* term instead. Compare the two by **Akaike Information Criterion** (**AIC**) – fit quality penalised by the number of parameters, lower is better:

orders	AIC	test MAE
$(1,1,1)(1,1,0)_{12}$	356.4	7.24
$(0,1,1)(0,1,1)_{12}$	**343.8**	**6.07**

The plot was right: the seasonal MA wins on both, and we never touched the test set to find that out.



The forecast comes with a **prediction interval** – uncertainty grows with the horizon. A point forecast without an interval is only half an answer.

In practice: don't hand-tune p, d, q at all. Use `pmdarima.auto_arima` or `statsforecast.AutoARIMA`, which search orders by AIC for you. Read the plot to *understand*, let the search *choose*.

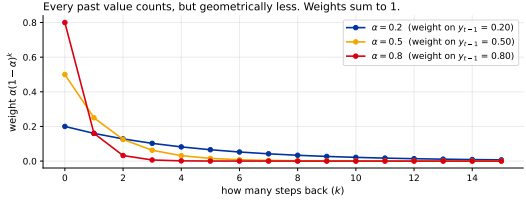
Exponential smoothing

Weighted averages where the recent past counts most.

Why average the past at all – and why not equally?

You want one number summarising "where the series is right now". Three ways to get it:

1. **Mean of everything.** Uses all the data – but a value from five years ago counts exactly as much as yesterday. Hopelessly sluggish.
2. **Mean of the last** k (the rolling mean). Reacts, but it is a cliff: the k-th oldest point counts *fully*, the $(k+1)$-th counts *zero*, everything older is thrown away.
3. **Weight everything, decaying with age.** Nothing discarded, nothing old dominating. This is **exponential smoothing**.



The weights run α, $\alpha(1-\alpha)$, $\alpha(1-\alpha)^2, \ldots$ – geometric, hence "exponential". $\alpha = 0.8$ gives $0.80, 0.16, 0.03$ (almost everything on the last two points); $\alpha = 0.2$ gives $0.20, 0.16, 0.13, 0.10$ (a long memory). α **is the single dial between "trust the latest reading" and "trust the history".**

Simple exponential smoothing, written two ways

Recursive form

$$\hat{y}_{t+1} = \alpha y_t + (1-\alpha)\hat{y}_t$$

"the new forecast is the old forecast, blended with what actually happened."

Error-correction form – the same line, rearranged:

$$\hat{y}_{t+1} = \hat{y}_t + \alpha\big(y_t - \hat{y}_t\big)$$

Read it: α is *the fraction of your latest error you correct for*. At $\alpha = 0.2$ you close a fifth of the gap.

Where the geometric weights come from. Substitute the rule into itself:

$$\begin{aligned}\hat{y}_{t+1} &= \alpha y_t + (1-\alpha)\hat{y}_t \\ &= \alpha y_t + (1-\alpha)\big[\alpha y_{t-1} + (1-\alpha)\hat{y}_{t-1}\big] \\ &= \alpha y_t + \alpha(1-\alpha) y_{t-1} + (1-\alpha)^2 \hat{y}_{t-1}\end{aligned}$$

Keep going and you get the weights on the previous slide. The recursion *is* the infinite weighted average, computed two numbers at a time.

The limits: $\alpha \to 1$ gives $\hat{y}_{t+1} = y_t$, the naive forecast. $\alpha \to 0$ freezes at a constant. And beyond one step the forecast is **flat**: $\hat{y}_{t+h} = \hat{y}_{t+1}$ for every h. That flatness is precisely why Holt and Holt-Winters exist.

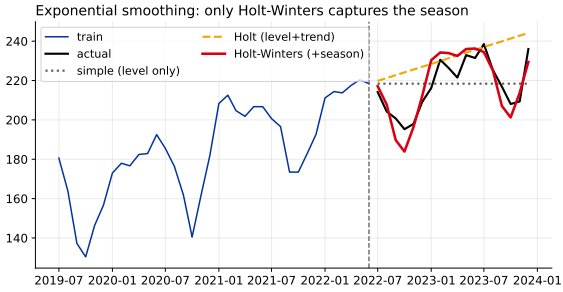
From a simple average to Holt-Winters

The family is written **ETS – Error, Trend, Seasonality**, the three components you switch on or off.

Simple Exponential Smoothing (SES) – level only, one dial α. Flat forecast.

Holt – adds a *slope*, smoothed by its own dial β exactly the way α smooths the level. Forecast is a line.

Holt-Winters – adds a *seasonal profile*, smoothed by γ. Level + trend + season, three dials, one idea applied three times.



ETS and ARIMA overlap but are not the same: ETS is built from interpretable *components*, ARIMA from *autocorrelation*. On many business series they perform similarly – try both.

By hand: choosing α for exponential smoothing

Series $y = 10, 14, 12, 18, 16$. Start the level at $\hat{y}_1 = y_1 = 10$ and score with the summed squared **one-step-ahead** errors.

$\alpha = 0.2$ (trust the history)

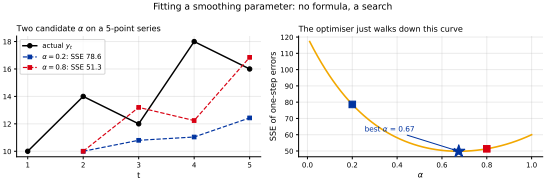
t	forecast	actual	error
2	10.00	14	+4.00
3	10.80	12	+1.20
4	11.04	18	+6.96
5	12.43	16	+3.57
		SSE	**78.61**

$\alpha = 0.8$ (trust the latest reading)

t	forecast	actual	error
2	10.00	14	+4.00
3	13.20	12	−1.20
4	12.24	18	+5.76
5	16.85	16	−0.85
		SSE	**51.34**

One update in full: $\hat{y}_3 = 0.8(14) + 0.2(10) = 11.2 + 2.0 = \mathbf{13.2}$. $\alpha = 0.8$ wins on this series – it jumps around, so leaning on the latest value pays. Notice we did not *choose* that; we **scored** it.

Same recipe everywhere



The two squares are the tables you just filled in. Sweep every α and the minimum sits at $\alpha = \mathbf{0.67}$ (SSE 49.8). An optimiser simply walks downhill to it.

The recipe, unchanged all lecture: write the one-step-ahead error, square it, sum it, minimise it.

- **AR** – regressors observed $\Rightarrow$ *closed form*.
- **MA / ARMA / SARIMA** – regressors unobserved $\Rightarrow$ *numerical search*.
- **ETS** (α, β, γ) – numerical search, and the starting level / trend / season get fitted too.

Do not confuse two loops. Fitting picks the *coefficients* for one fixed (p, d, q). **AIC** picks between *different* (p, d, q). Inner loop and outer loop.

Real libraries refine the inner loop, but the shape never changes: *propose, score, improve.*

Baselines and evaluation

A forecast is only "good" relative to the dumbest thing that works.

Always start with a naive baseline

Before any model, write down the trivial forecast:

- **Naive:** $\hat{y}_t = y_{t-1}$ (tomorrow = today).
- **Seasonal naive:** $\hat{y}_t = y_{t-m}$ (this December = last December).
- **Drift:** last value + average change.

If your fancy model can't beat seasonal naive, it is not helping. This happens far more often than people admit.

Error metrics

- **Mean Absolute Error** (**MAE**) – in the target's units.
- **Root Mean Squared Error** (**RMSE**) – penalizes big misses more.
- **Mean Absolute Percentage Error** (**MAPE**) – breaks when $y \approx 0$, and punishes over-forecasting more than under-forecasting.
- **Mean Absolute Scaled Error** (**MASE**) – scale-free; the one to report.

MASE, by hand

$$\text{MASE} = \frac{\text{MAE of your forecast on the } \textbf{test} \text{ set}}{\text{MAE of the seasonal naive forecast } \textbf{in-sample}\text{, on the training set}}$$

Denominator – walk the *training* series and predict each month with the same month last year. Say those errors are $|-12|, 20, |-18|, 22$:

$$\frac{12+20+18+22}{4} = 18$$

Numerator – your model's test errors, say $5, |-9|, 7, 3$:

$$\frac{5+9+7+3}{4} = 6$$

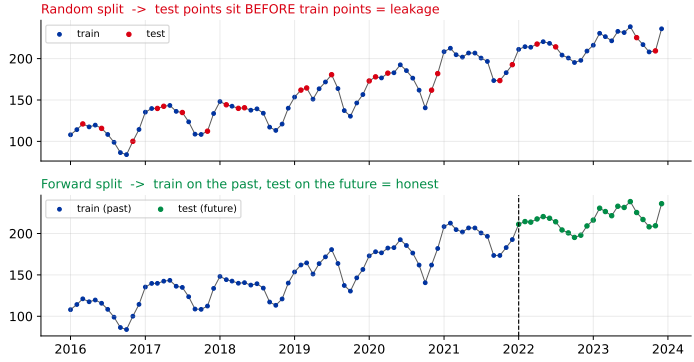
$$\text{MASE} = \frac{6}{18} = \mathbf{0.33}$$

Your errors are a third of what naive repetition would cost. **Below 1 is a win, above 1 means you lost to copying last year.**

Note the asymmetry: the denominator is measured *in-sample*, on train. So the seasonal naive evaluated on the *test* set does **not** have to score exactly 1.00 – it can land either side. Two slides from now ours comes out at 1.22.

Evaluate on the future, never by random split

You cannot shuffle-and-split a time series: that trains on the future to predict the past.



The **k-fold cross-validation** you have used since Ridge assumes exchangeable rows. Shuffling breaks that *twice*: you train on the future, *and* adjacent rows are **autocorrelated**, so neighbouring folds leak into each other. Full machinery – lag features, `TimeSeriesSplit` – in lecture 30.

One more trap: even the baseline has to respect the arrow

Seasonal naive means "this December = last December". Writing that as `y.shift(12)` across a test block looks harmless – and is wrong the moment your horizon exceeds one season.

Forecasting 18 months from one origin, steps 13–18 would reach back only 12 and land *inside the test window* – the "baseline" reads the future it is supposed to predict.

On our series that single fix moves the seasonal naive from MASE **0.90** to **1.22**.

Which is the whole point: a leaky baseline is the most dangerous kind. It sets the bar *too high*, so a good model looks mediocre – and you go tune something that was never the problem.

The honest version always reaches back to an *observed* value:

$$\hat{y}_{T+k} \;=\; y_{T+k-m\lceil k/m \rceil}$$

Rule of thumb for picking an approach: short, single, strongly-seasonal series → ARIMA / ETS are hard to beat. Many related series, long history, exogenous drivers → the ML and deep methods of the next lecture start to win.

Recap – classical time series

Ideas that transfer:

- Time order is information; only train on the past.
- Decompose into trend + season + residual (STL). A *cycle* has no fixed period, so it never gets its own term.
- Difference the *minimum* amount; check with ADF **and** KPSS.
- AR carries the *value* forward, MA the *shock*. MA(q) is not a rolling mean.
- Autocorrelation is just r between a series and a shifted copy; ACF is the total link, PACF the direct one.
- Coefficients are not chosen by taste: **minimise the summed squared one-step error**. AR has a closed form; everything else is a search.

Next (30): ML for time series. Reframe forecasting as supervised learning –

- time-aware splits and leakage,
- lag / rolling / calendar features,
- `TimeSeriesSplit` cross-validation,
- gradient boosting (and the trend-extrapolation trap),
- deep learning and the new foundation models.

The lag table on the "AR is least squares" slide is literally the first slide of lecture 30. Classical and ML forecasting are closer than the two names suggest.

Questions?

Next: forecasting as a supervised ML problem.