Time Series Analysis and Forecasting

Chapter 8: Modern Extensions

Daniel Traian PELE

Bucharest University of Economic Studies
IDA Institute Digital Assets
Blockchain Research Center
AI4EFin Artificial Intelligence for Energy Finance
Romanian Academy, Institute for Economic Forecasting
MSCA Digital Finance

Chapter Outline

Learning Objectives

By the end of this chapter, you will be able to:

1. **Understand** long memory and fractional integration
2. **Distinguish** between short and long memory processes
3. **Estimate** the fractional parameter d using GPH, Local Whittle, and MLE
4. **Apply** Random Forest for time series forecasting
5. **Build** LSTM networks for sequential data
6. **Compare** classical vs ML model performance
7. **Choose** the appropriate method based on data characteristics
8. **Implement** ARFIMA, Random Forest, and LSTM in Python

From Classical Models to Machine Learning

The Evolution of Time Series Methods

- **Classical ARIMA** (Box & Jenkins, 1970) — revolutionized forecasting but has limitations:
  - Assumes **short memory**: autocorrelations decay exponentially
  - **Linear** relationships only — cannot capture complex dynamics
  - Requires **stationarity** through integer differencing

Three Paradigm Shifts

- **ARFIMA** (Granger & Joyeux, 1980)
  - Fractional integration for long memory processes
- **Random Forest** (Breiman, 2001)
  - Ensemble learning for nonlinear relationships
- **LSTM** (Hochreiter & Schmidhuber, 1997)
  - Deep learning for complex sequential patterns

When to Use Each Method?

Feature	ARIMA	ARFIMA	RF	LSTM
Long memory	×	✓	✓	✓
Nonlinear relationships	×	×	✓	✓
Interpretability	✓	✓	∼	×
Small data	✓	✓	×	×
Exogenous variables	✓	✓	✓	✓
Uncertainty quantification	✓	✓	∼	×

Principle of Parsimony (Occam's Razor)

Start **simple** (ARIMA), then increase complexity only if justified by **out-of-sample** performance gains.

Makridakis et al. (2018) M4 Competition: simple methods often outperform complex ML models.

What is Long Memory?

Short Memory (ARMA)

- **ACF Behavior**:
  - Autocorrelations ρ_k decay **exponentially**: $|\rho_k| \leq C \cdot r^k$, $r < 1$
  - Finite sum: $\sum_{k=0}^{\infty} |\rho_k| < \infty$
- **Implication**: Shock effects disappear quickly

Long Memory (ARFIMA)

- **ACF Behavior**:
  - Autocorrelations decay **hyperbolically**: $\rho_k \sim C \cdot k^{2d-1}$
  - Infinite sum: $\sum_{k=0}^{\infty} |\rho_k| = \infty$
- **Implication**: Shock effects persist for a long time

Examples

Financial volatility, river flows, network traffic, inflation, climate data

Long Memory: An Intuitive Analogy

Short Memory (ARMA)

Analogy: You only remember the last few sentences.

- Yesterday's news? Forgotten
- Last week's event? Gone
- Effect of shocks fades **quickly**

Example: Daily stock returns

Long Memory (ARFIMA)

Analogy: An elephant that never forgets.

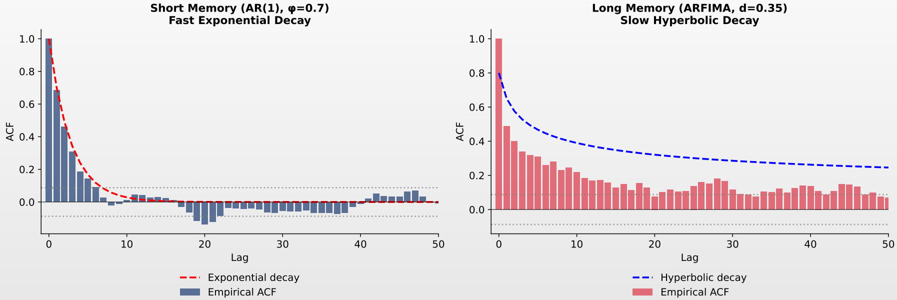
- Old shocks still matter
- Slow decay of influence
- **Persistent** patterns

Example: Stock volatility, river flows

Key Question

How fast do autocorrelations decay? **Exponentially** (short) or **hyperbolically** (long)?

ACF Comparison: Short Memory vs Long Memory



Interpretation

- **Data**: Simulated AR(1) with $\phi = 0.8$ and ARFIMA(0,d,0) with $d = 0.35$ ($n = 1000$)
- **Left**: AR(1) — autocorrelations decay exponentially (short memory)
- **Right**: ARFIMA with $d = 0.35$ — autocorrelations decay hyperbolically (long memory)

TSA_ch8_acf_comparison

The ARFIMA(p,d,q) Model

Definition 1 (ARFIMA — Granger & Joyeux (1980), Hosking (1981))

A process $\{Y_t\}$ follows an **ARFIMA(p,d,q)** model if: $\phi(L)(1-L)^d Y_t = \theta(L)\varepsilon_t$ where $d \in (-0.5, 0.5)$ is the **fractional differencing parameter**.

Fractional Differencing Operator

$(1-L)^d = \sum_{k=0}^{\infty} \binom{d}{k}(-L)^k = 1 - dL - \frac{d(1-d)}{2!}L^2 - \frac{d(1-d)(2-d)}{3!}L^3 - \cdots$

- $d = 0$: Standard ARMA (short memory)
- $0 < d < 0.5$: Long memory, stationary
- $d = 0.5$: Stationarity boundary
- $0.5 \leq d < 1$: Non-stationary, mean-reverting
- $d = 1$: Random walk (standard ARIMA)

Interpreting the Parameter d

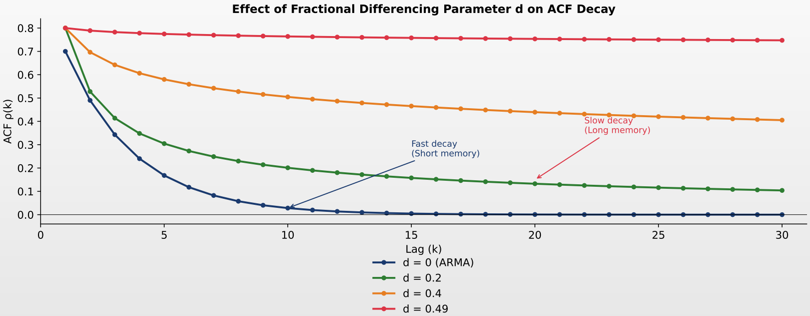
Value of d	ACF Behavior	Interpretation
$d = 0$	Exponential decay	Short memory
$0 < d < 0.5$	Hyperbolic decay	Long memory, stationary
$d = 0.5$	Non-summable ACF	At the boundary
$0.5 < d < 1$	Very slow decay	Long memory, non-stationary
$d = 1$	ACF $= 1$ (constant)	Random walk

Hurst Parameter H

Relationship with Hurst exponent: $d = H - 0.5$

- $H = 0.5$: Random walk (no memory)
- $H > 0.5$: Persistence (trend-following)
- $H < 0.5$: Anti-persistence (mean-reverting)

Effect of Parameter d on ACF

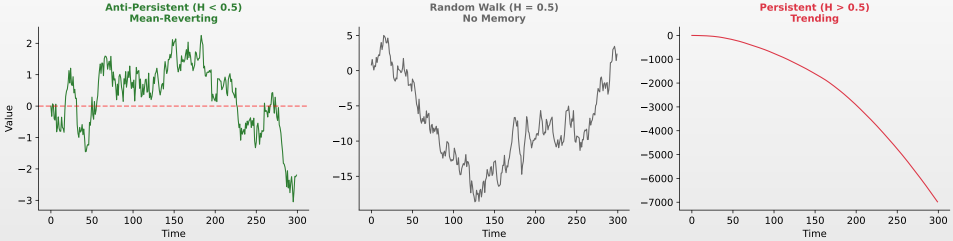


Interpretation

- **Data**: Simulated ARFIMA(0,d,0) for $d \in \{0.1, 0.2, 0.3, 0.4\}$ ($n = 1000$)
- The higher d, the slower autocorrelations decay
- As $d \to 0.5$, autocorrelations remain significant even at very large lags

TSA_ch8_arfima_d_effect

Hurst Exponent: Visual Interpretation



Interpretation

- **Data**: Simulated fractional Brownian motion with $H \in \{0.3, 0.5, 0.7\}$
- **H** $<$ **0.5**: Mean-reverting **H** $=$ **0.5**: Random walk **H** $>$ **0.5**: Persistent

TSA_ch8_hurst_interpretation

Estimating the Hurst Exponent

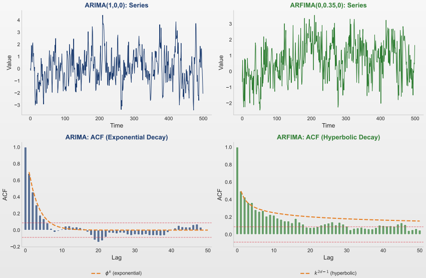
Classical Methods

- **R/S Analysis** (Hurst, 1951): Regress $\log(R/S) = c + H \cdot \log(n)$
  - Simple but sensitive to short-range dependence
- **DFA** (Peng et al., 1994): Remove local trends, compute fluctuations
  - Robust to non-stationarities and trends

Frequency Domain Methods

- **GPH estimator**: $\hat{d} = -\hat{\beta}/2$ from log-periodogram; $H = d + 0.5$
- **Wavelet-based** (Abry & Veitch, 1998): Multi-scale decomposition, robust

ARIMA vs ARFIMA: Memory Decay Patterns

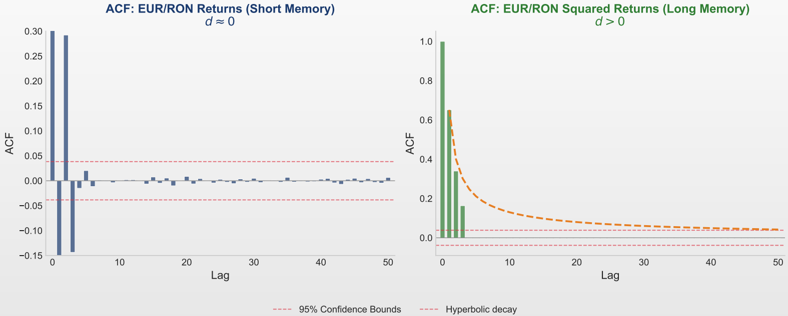


Interpretation

- **Data**: Simulated ARIMA(1,1,1) vs ARFIMA(1,d,1) with $d = 0.35$
- **ARIMA** (left): ACF decays **exponentially** – shocks are quickly “forgotten”
- **ARFIMA** (right, $d = 0.35$): ACF decays **hyperbolically** – shocks persist for long periods

TSA_ch8_arima_vs_arfima

Real Data Example: EUR/RON Long Memory Analysis

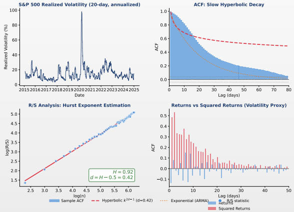


Interpretation

- **Data**: EUR/RON daily exchange rate (Yahoo Finance, 2015–2025)
- **Returns**: $H \approx 0.50$, $d \approx 0$ – short memory
- **Squared returns**: $H \approx 0.65$, $d \approx 0.15$ – long memory in volatility

TSA_ch8_eurron_long_memory

ARFIMA Example: S&P 500 Realized Volatility



Estimation Results

- **Data**: S&P 500 daily returns (Yahoo Finance, 2015–2024)
- Hurst: $H = 0.92$, $d = H - 0.5 = 0.42$ – strong long memory in realized volatility

Key Insight

Volatility has **long memory** – shocks persist longer than ARMA; use ARFIMA or FIGARCH!

TSA_ch8_arfima_sp500

ARFIMA vs ARIMA: When to Use?

Use ARFIMA when:

- ACF decays **slowly** (hyperbolically)
- H significantly $\neq 0.5$
- **Long horizon** forecasting
- Modeling **volatility**

ARFIMA Limitations

- More complex estimation
- Requires longer series
- Estimation of d is sensitive
- Not always better short-term

Use ARIMA when:

- ACF decays **rapidly** (exponentially)
- Short series ($<$ 500 obs.)
- **Short horizon** forecasting
- Simplicity is priority

ARFIMA Advantages

- Parsimonious (single d)
- Better long-horizon forecasts
- Captures slow ACF decay

Practical Applications of Long Memory

Finance

- **Volatility modeling**: GARCH may underestimate persistence
- **Risk management**: Long-horizon VaR
- **Option pricing**: Long memory affects implied volatility
- **Portfolio optimization**: Correlations persist longer

Other Domains

- **Hydrology**: River flows, precipitation
- **Network traffic**: Internet data packets
- **Economics**: Inflation, GDP growth
- **Climate**: Temperature anomalies
- **Geophysics**: Earthquake magnitudes

Key Insight

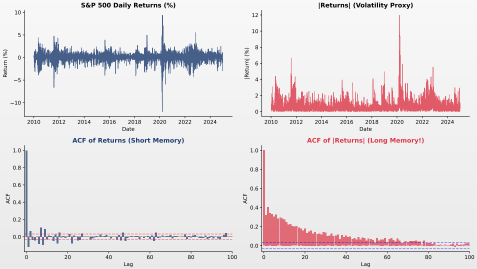
Long memory means that **shocks have lasting effects** – important for policy, risk management, and forecasting!

ARFIMA Estimation: Step-by-Step Procedure

Recommended Workflow

1. **Test for Long Memory**:
   - Examine ACF decay pattern (slow = long memory)
   - Compute $\hat{H}$ using R/S or GPH; test if $H \neq 0.5$
2. **Estimate** d:
   - Use GPH or Local Whittle for initial estimate
   - Verify $d \in (0, 0.5)$ for stationary long memory
3. **Fit Full ARFIMA(p,d,q)**:
   - Fix $\hat{d}$ from step 2, select p, q via AIC/BIC
   - Or estimate all parameters jointly via MLE
4. **Diagnostic Checking**:
   - Residuals should be white noise (Ljung-Box test)
   - Check for remaining autocorrelation structure

Real Example: Long Memory in Volatility



Interpretation

- **Data**: S&P 500 daily returns (Yahoo Finance, 2010–2025)
- **Stylized Fact**: Financial returns have short memory, but volatility ($|r_t|$) has long memory
- This is the basis for FIGARCH models

TSA_ch8_volatility_long_memory

ARFIMA Estimation: Overview of Methods

Three Main Approaches

1. **GPH (Geweke-Porter-Hudak)**: Log-periodogram regression
   - Semiparametric, uses only low frequencies
   - Simple but less efficient
2. **Local Whittle**: Frequency-domain likelihood
   - More efficient than GPH
   - Robust to short-memory contamination
3. **Exact MLE (Sowell, 1992)**: Full parametric approach
   - Most efficient, requires full model specification
   - Computationally intensive

Key Trade-off

Semiparametric (GPH, Whittle): Robust, fewer assumptions
Parametric (MLE): More efficient, requires correct model specification

GPH Estimator (Geweke & Porter-Hudak, 1983)

Definition 2 (Log-Periodogram Regression)

The GPH estimator is based on the regression:

$$\ln I(\omega_j) = c - d \ln\left(4 \sin^2\left(\frac{\omega_j}{2}\right)\right) + \text{error}$$

where $I(\omega_j)$ is the periodogram at Fourier frequency $\omega_j = \frac{2\pi j}{n}$.

Key Properties

- Uses only **lowest** m **frequencies** where long-memory dominates
- Typical choice: $m = n^{0.5}$ to $n^{0.8}$ (trade-off bias vs variance)
- **Asymptotic normality**: $\sqrt{m}(\hat{d} - d) \xrightarrow{d} N(0, \frac{\pi^2}{24})$

Bandwidth Selection

Too small m: High variance Too large m: Bias from short-memory component

Local Whittle Estimator (Robinson, 1995)

Definition 3 (Local Whittle Objective Function)

The Local Whittle estimator minimizes: $R(d) = \ln\left(\frac{1}{m}\sum_{j=1}^{m}\omega_j^{2d}I(\omega_j)\right) - \frac{2d}{m}\sum_{j=1}^{m}\ln(\omega_j)$ where m is the bandwidth parameter.

Advantages over GPH

- **More efficient**: $\sqrt{m}(\hat{d} - d) \xrightarrow{d} N(0, \frac{1}{4})$ vs $N(0, \frac{\pi^2}{24})$ for GPH
- Robust to **additive noise** and **mean shifts**

Practical Note

Both GPH and Local Whittle are **semiparametric**: they estimate d without specifying the short-memory (ARMA) structure.

Exact MLE: Sowell (1992)

Full Parametric Approach

The exact MLE maximizes the Gaussian log-likelihood:

$$\ell(\phi, d, \theta, \sigma^2) = -\frac{n}{2}\ln(2\pi\sigma^2) - \frac{1}{2}\ln|\Sigma| - \frac{1}{2\sigma^2}y'\Sigma^{-1}y$$

where Σ is the autocovariance matrix of the ARFIMA(p,d,q) process.

Advantages

- **Most efficient** (Cramér-Rao bound)
- Joint estimation of d, ϕ, θ
- Standard errors for all parameters

Disadvantages

- Requires **correct specification**
- **Computationally intensive** ($O(n^3)$)
- Sensitive to non-Gaussianity

Sowell's contribution: Efficient algorithm to compute exact autocovariances of ARFIMA.

Approximate MLE Methods

Computational Alternatives to Exact MLE

- **CSS (Conditional Sum of Squares)**:
  - Conditions on initial values, avoids matrix inversion
  - Fast but less efficient for small samples
- **Whittle Likelihood**:
  - Frequency-domain approximation: $\ell_W = -\sum_j \left[\ln f(\omega_j) + \frac{I(\omega_j)}{f(\omega_j)}\right]$
  - $O(n \log n)$ complexity via FFT
- **State-Space Representation**:
  - Kalman filter for likelihood evaluation
  - Handles missing data naturally

Practical Recommendation

For large samples ($n > 1000$): Use Whittle or CSS
For small samples ($n < 500$): Use exact MLE if feasible

ARFIMA Estimation in Python

Using the arch Package (Approximate MLE)

```
from arch.univariate import ARFIMA

# Estimate ARFIMA(1,d,1) with d estimated
model = ARFIMA(returns, p=1, d=None, q=1)
result = model.fit()

# Display results
print(f"Estimated d: {result.params['d']:.4f}")
print(f"Std Error:   {result.std_err['d']:.4f}")
```

Key Points

- `d=None`: Estimate d from data `d=0.3`: Fix d at 0.3
- Uses approximate MLE (efficient for moderate samples)

TSA_ch8_arfima_estimation

GPH and Hurst Estimation in Python

Hurst Exponent via R/S Analysis

```
from hurst import compute_Hc  # pip install hurst
H, c, data = compute_Hc(returns, kind='price')
d_rs = H - 0.5
```

GPH Estimator (Simplified)

```
def gph_estimator(y, m=None):
    n, m = len(y), m or int(len(y)**0.5)
    I = np.abs(fft(y-np.mean(y)))**2/(2*np.pi*n)
    omega = 2*np.pi*np.arange(1,m+1)/n
    x = np.log(4*np.sin(omega/2)**2)
    return -np.polyfit(x, np.log(I[1:m+1]), 1)[0]
```

TSA_ch8_gph_estimation

Comparing Estimation Methods: Summary

Method	Efficiency	Robustness	Speed	Assumptions
GPH	Low	High	Fast	Minimal
Local Whittle	Medium	High	Fast	Minimal
Whittle MLE	Medium-High	Medium	Medium	Parametric
Exact MLE	Highest	Low	Slow	Full model

Recommended Workflow

1. **Initial screening**: Use GPH or Hurst exponent to detect long memory
2. **Robust estimation**: Use Local Whittle to estimate d
3. **Final model**: Fit ARFIMA(p,d,q) via MLE with $\hat{d}$ as starting value
4. **Validation**: Compare different bandwidths/methods for robustness

Sensitivity Check

If GPH and Whittle estimates differ substantially, investigate short-memory contamination or structural breaks.

Random Forest: Basic Concepts

What is Random Forest? (Breiman, 2001)

- **Ensemble learning** method combining multiple decision trees:
  - Each tree trained on a **bootstrap sample** (bagging)
  - At each split, only $m \ll p$ **random features** considered
  - Final prediction = **average** of all tree predictions

Why It Works for Time Series

- **Flexibility**:
  - Captures nonlinear relationships and interactions automatically
  - No stationarity assumption required
- **Robustness**:
  - Resistant to outliers, noise, and irrelevant features
  - Built-in feature importance for interpretability

Why "Random" Forest? The Power of Diversity

Two Sources of Randomness

1. **Bootstrap Sampling**: Each tree sees a different subset of data
2. **Feature Sampling**: Each split considers only m random features

Analogy: Wisdom of Crowds

- Ask 100 people to guess weight of an ox
- Individual guesses: high variance
- **Average**: remarkably accurate!

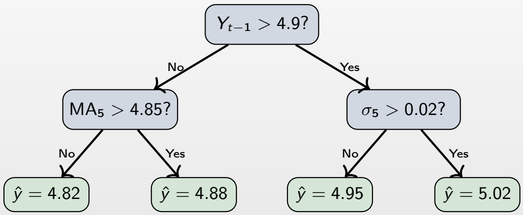
Why It Works

- Trees make **different errors**
- Errors **cancel out** when averaging
- Result: lower variance, same bias

Mathematical Insight

If trees are uncorrelated with variance σ^2, forest variance $= \frac{\sigma^2}{B}$ (B = number of trees)

How Does a Decision Tree Make Predictions?



Tree Prediction

1. Start at the root
2. Check the condition (split)
3. Go left (No) or right (Yes)
4. Repeat until a leaf
5. **Leaf value = prediction**

Random Forest

$\hat{y} = \frac{1}{B}\sum_{b=1}^{B} T_b(\text{x})$ — Average of B trees

Random Forest: Mathematical Formulation

Prediction

$\hat{y} = \frac{1}{B}\sum_{b=1}^{B} T_b(\mathrm{x})$ where $T_b(\mathrm{x})$ = prediction of tree b

Feature Importance (MDI = Mean Decrease in Impurity)

$\text{Importance}(X_j) = \frac{1}{B}\sum_{b=1}^{B}\sum_{\substack{t\in T_b \\ j_t=j}} \Delta I_t$

Sum over all nodes t in all trees where feature j was used; ΔI_t = impurity decrease

Out-of-Bag Error (OOB)

$\text{OOB} = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, \frac{1}{|B_i^-|}\sum_{b\in B_i^-} T_b(\mathrm{x}_i)\right)$

B_i^- = trees where obs. i was **not** in the bootstrap (free validation!)

Random Forest: How It Works

Training Process

1. Draw B bootstrap samples
2. For each sample b, grow tree T_b:
   - At each node, select m features
   - Find best split: $\min_{j,s} \sum(y_i - \bar{y}_{R_1})^2$
   - Continue until stopping criterion
3. Aggregate: $\hat{y} = \frac{1}{B} \sum_{b=1}^{B} T_b(x)$

Key Hyperparameters

- B: number of trees (100-500)
- m: features per split ($\sqrt{p}$)
- max_depth: tree depth
- min_samples: leaf size

Feature Engineering: The Key to ML Success

Critical Insight

ML models don't "understand" time — you must **encode temporal patterns as features**!

Lag Features

$y_{t-1}, y_{t-2}, \ldots$ – capture **AR** patterns

Rolling Statistics

Mean $\bar{y}_{k,t}$, std $\sigma_{k,t}$ – **local trends**

Calendar Features

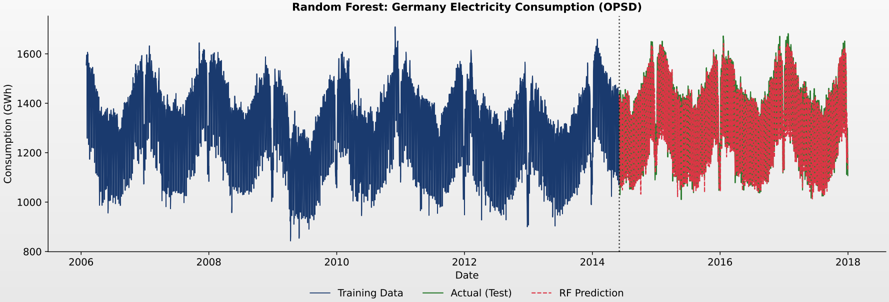
Day, month, holiday – **seasonality**

Domain Features

Weather, economics – external **regressors**

TSA_ch8_feature_engineering

Random Forest: Forecast Example



Interpretation

- **Data**: Germany daily electricity consumption (OPSD, 2012–2017)
- RF trained on historical data (blue) produces forecasts (red dashed) that closely track actual values (green)

TSA_ch8_rf_prediction

Random Forest: Why It Works for Time Series

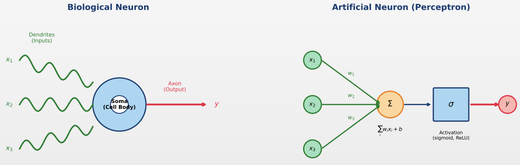
Strengths

- No linearity assumption
- Automatic interaction detection
- Handles mixed data types
- Built-in OOB validation
- Parallelizable training

Limitations

- Cannot extrapolate beyond training range
- Requires manual feature engineering
- Less interpretable than single tree

From Biological to Artificial Neurons



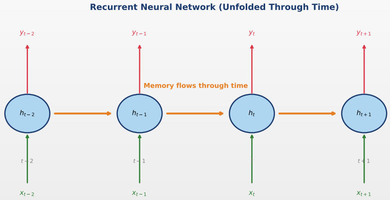
Dendrites → Inputs with weights | Soma → Weighted sum + activation | Axon → Output

The Analogy

- **Dendrites** $\rightarrow$ **Inputs** x_i **Synapses** $\rightarrow$ **Weights** w_i **Soma** $\rightarrow$ **Sum + Activation** **Axon** $\rightarrow$ **Output** y

TSA_ch8_neuron_comparison

Recurrent Neural Networks (RNN)



Key Idea

- Processes **sequences** step by step
- Hidden state h_t carries **memory**
- Update: $h_t = \tanh(W_h h_{t-1} + W_x x_t)$

Vanishing Gradient Problem

- **Gradient**: derivative to update weights
- Long sequences: $\frac{\partial L}{\partial h_1} \propto \prod W_h \to 0$
- Early steps **stop learning**
- **Solution**: LSTM/GRU gates

TSA_ch8_rnn_unfolded

LSTM: Long Short-Term Memory

The LSTM Solution (Hochreiter & Schmidhuber, 1997)

A gated architecture with **3 learned gates** that control information flow: **Forget** (f_t) – what to discard; **Input** (i_t) – what to store; **Output** (o_t) – what to transmit

LSTM Equations

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-\mathbf{1}}, x_t] + b_f) && \text{(Forget)} \\
i_t &= \sigma(W_i \cdot [h_{t-\mathbf{1}}, x_t] + b_i) && \text{(Input)} \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-\mathbf{1}}, x_t] + b_C) && \text{(Candidate)} \\
C_t &= f_t \odot C_{t-\mathbf{1}} + i_t \odot \tilde{C}_t && \text{(Cell state)} \\
o_t &= \sigma(W_o \cdot [h_{t-\mathbf{1}}, x_t] + b_o) && \text{(Output)} \\
h_t &= o_t \odot \tanh(C_t) && \text{(Hidden state)}
\end{aligned}
$$

LSTM Gates: An Intuitive Explanation

Analogy: A Smart Secretary

The LSTM cell is like a secretary managing information flow in an office.

Forget Gate f_t

"What to throw away?"

- Reviews old files
- Decides what's outdated
- $f_t \approx 0$: delete
- $f_t \approx 1$: keep

Input Gate i_t

"What to file?"

- Reviews new info
- Decides importance
- $i_t \approx 0$: ignore
- $i_t \approx 1$: store

Output Gate o_t

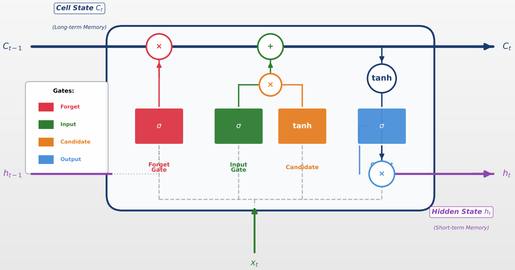
"What to report?"

- Reviews memory
- Decides relevance
- $o_t \approx 0$: hide
- $o_t \approx 1$: share

Key Insight

Gates are **learned** during training — the network discovers what to remember and forget!

LSTM Cell Architecture

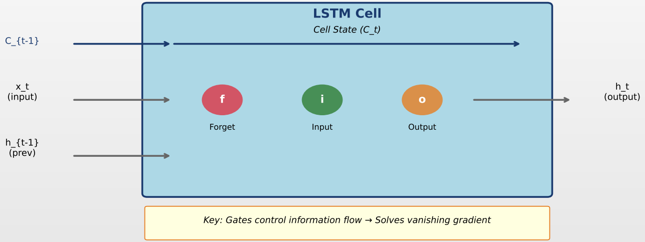


Components

- **Cell State** (C_t): Long-term memory **Hidden State** (h_t): Short-term memory
- **Gates**: forget, add, transmit

TSA_ch8_lstm_cell

LSTM Cell Architecture

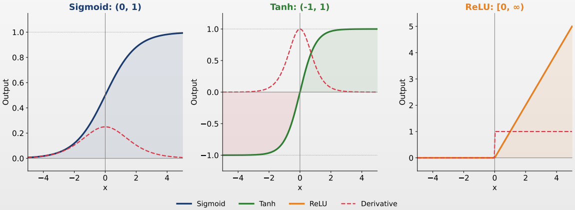


Key Insight

- The gates (forget, input, output) control what information is discarded, added, and transmitted
- **Cell state** allows gradients to "flow" without degradation

TSA_ch8_lstm_architecture

Activation Functions: Why Do We Need Them?



Why Activation Functions?

- Without them, networks can only learn **linear** relationships
- **In LSTM**: Sigmoid for gates (0-1), Tanh for cell state (-1 to 1)

LSTM Advantages for Time Series

Why LSTM?

- Captures long-term dependencies
- Variable-length sequences
- Complex nonlinear patterns
- Multivariate time series

Disadvantages

- Needs large datasets
- "Black box" model
- Sensitive to hyperparameters
- Prone to overfitting

LSTM: Key Hyperparameters

Architecture

- **Units**: neurons per layer (32-256)
- **Layers**: stacked LSTM (1-3)
- **Sequence length**: past observations (10-100)
- **Dropout**: regularization (0.1-0.3)

Training

- **Batch size**: samples per update (32-128)
- **Epochs**: training iterations (50-200)
- **Learning rate**: step size (0.001)
- **Early stopping**: prevents overfitting

Practical Tips

- **Normalize/scale** data to [0,1] or [-1,1]
- Use **validation set** for hyperparameter tuning
- Monitor **training vs validation loss** for overfitting

LSTM: When to Use It

Good Choice When:

- Large datasets (> 1000 obs)
- Complex temporal patterns
- Multivariate inputs
- Accuracy over interpretability

NOT a Good Choice When:

- Small data (< 500 obs)
- Linear relationships
- Interpretability required
- ARIMA already performs well

Evaluation Metrics

Notation: y_i = actual value, $\hat{y}_i$ = predicted value, n = number of observations

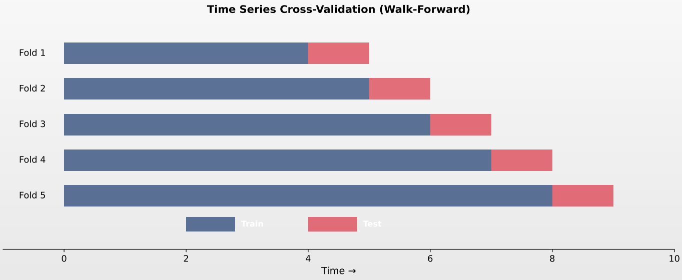
Common Metrics

- **Scale-Dependent**:
  - RMSE: $\sqrt{\frac{1}{n}\sum(y_i - \hat{y}_i)^2}$ — penalizes large errors
  - MAE: $\frac{1}{n}\sum|y_i - \hat{y}_i|$ — robust to outliers
- **Scale-Free**:
  - MAPE: $\frac{100}{n}\sum\left|\frac{y_i - \hat{y}_i}{y_i}\right|$ — percentage error
  - MASE: $\frac{\text{MAE}}{\frac{1}{n-1}\sum_{i=2}^{n}|y_i - y_{i-1}|}$ — relative to naive (random walk)

Validation for Time Series

- **Critical**: Do NOT use standard k-fold cross-validation!
  - Use Time Series CV (walk-forward validation)
  - Or temporal train/validation/test split

Time Series Cross-Validation

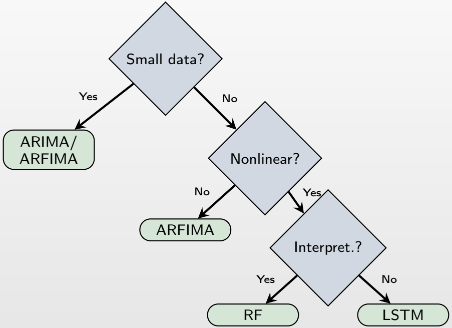


Interpretation

- **Illustration**: Schematic of expanding-window walk-forward validation (5 folds)
- Training set grows progressively; test is always in the future $\Rightarrow$ avoids data leakage

TSA_ch8_timeseries_cv

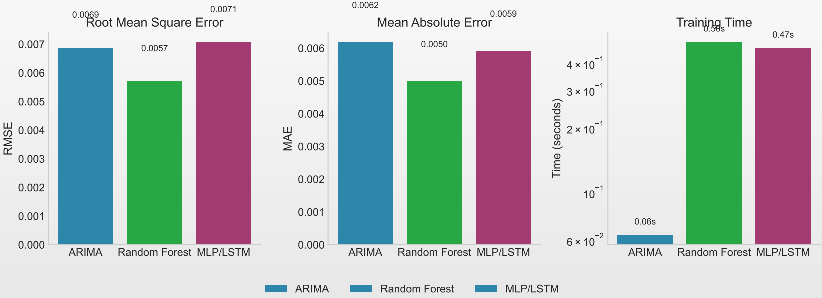
Model Selection Guide



Trade-off

ML models offer better accuracy but higher computational cost. For small data or interpretability, ARIMA/ARFIMA remain excellent choices.

Model Comparison: Accuracy vs Computational Cost



Interpretation

- **Data**: EUR/RON daily exchange rate (Yahoo Finance, 2019–2025)
- **Trade-off**: ML models may achieve better accuracy, but computational cost increases significantly
- For small data or interpretability, ARIMA/ARFIMA remain excellent choices

TSA_ch8_model_comparison

Key Formulas – Summary

ARFIMA(p,d,q)

$\phi(L)(1-L)^d Y_t = \theta(L)\varepsilon_t$

$d \in (-0.5, 0.5)$: long memory

Long Memory

ACF: $\rho_k \sim C \cdot k^{2d-1}$
Hurst: $d = H - 0.5$

$H > 0.5$: persistence

Random Forest

$\hat{y} = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$

B trees, random features

LSTM Cell

$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$
$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$

Forget, Input, Output gates

Evaluation Metrics

$\text{RMSE} = \sqrt{\frac{1}{n}\sum(y_i - \hat{y}_i)^2}$
$\text{MAPE} = \frac{100}{n}\sum\left|\frac{y_i - \hat{y}_i}{y_i}\right|$

Time Series CV

Walk-forward validation
Train $\rightarrow$ Test (temporal split)

Case Study: Energy Consumption Forecasting

Data Source

- **Series**: Germany daily electricity consumption
- **Unit**: Gigawatt-hours (GWh)
- **Period**: Jan 2012 – Dec 2017
- **Observations**: 2,162 daily values
- **Source**: Open Power System Data

Data Split (Temporal!)

- **Training**: 70% (1,513 obs)
- **Validation**: 15% (324 obs)
- **Test**: 15% (325 obs)

Key Patterns

- **Weekly**: Lower on weekends
- **Annual**: Higher in winter
- **Holidays**: Significant drops
- **Trend**: Slight decrease

Why ML works here:

Complex multi-seasonal patterns + sufficient data (2000+ obs) = ideal for ML!

Case Study: Feature Engineering

Lag Features

- Previous day: y_{t-1}
- Same day last week: y_{t-7}
- Two weeks ago: y_{t-14}
- Full week history: $y_{t-1}, \ldots, y_{t-7}$

Calendar Features

- Day of week (1–7)
- Month (1–12)
- Is weekend (0/1)
- Is holiday (0/1)

Rolling Statistics

- 7-day mean: $\bar{y}_{7,t} = \frac{1}{7} \sum_{i=1}^{7} y_{t-i}$
- 7-day std: $\sigma_{7,t}$
- 30-day mean: $\bar{y}_{30,t}$

Avoid Data Leakage!

- Use **only past data**
- Rolling stats: exclude y_t
- Scale with **training** stats only

Total: 14 features for Random Forest and LSTM models

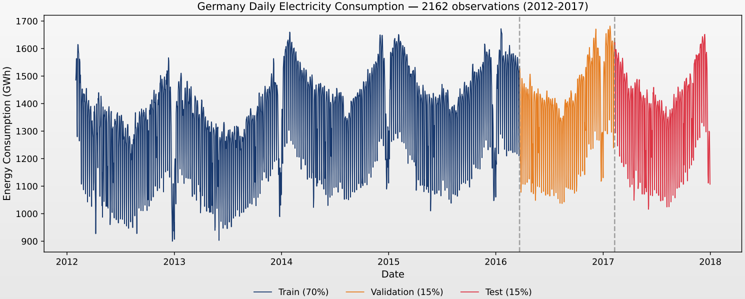
Case Study: Models Compared

Model	**Description**	**Configuration**
Baseline	Seasonal naive: $\hat{y}_t = y_{t-7}$	No parameters
SARIMA	Seasonal ARIMA with weekly seasonality	Order: $(1, 1, 1)$ Seasonal: $(1, 0, 1)_7$
ARFIMA	Fractional differencing with long memory	$H = 0.77 \Rightarrow d = 0.27$ Rolling one-step forecasts
Random Forest	Ensemble of 200 trees with all 14 features	max_depth = 15 min_samples_leaf = 5
LSTM	2-layer LSTM (64, 32 units) with all 14 features	seq_length = 7 days dropout = 0.2, early stopping

Evaluation Metric: MAPE

$\text{MAPE} = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$ — interpretable as “average % error”

Case Study: Data Overview

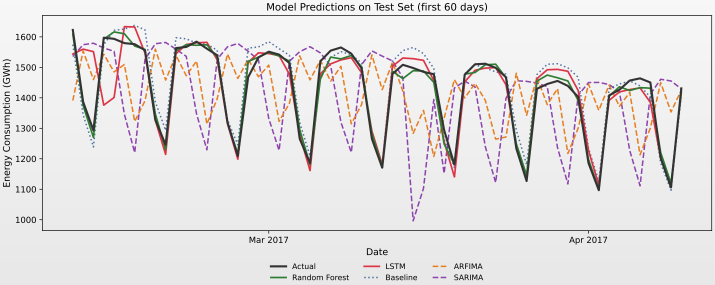


Data Overview

- **Data**: Germany daily electricity consumption (OPSD, 2012–2017)
- **Train**: 1513 obs (70%) **Validation**: 324 obs (15%) **Test**: 325 obs (15%)

TSA_ch8_data_split

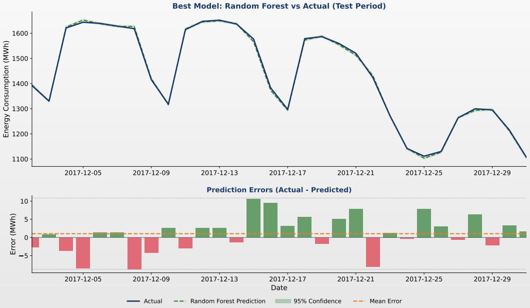
Case Study: Model Predictions



Rank	Model	MAPE	Interpretation
1	Random Forest	2.2%	Best: captures nonlinear patterns
2	LSTM	3.3%	Good, needs more data
3	Baseline	3.9%	Simple but competitive
4	ARFIMA	12.3%	Long memory not sufficient
5	SARIMA	14.6%	Struggles with patterns

TSA_ch8_model_predictions

Case Study: Best Model Performance



Random Forest Wins

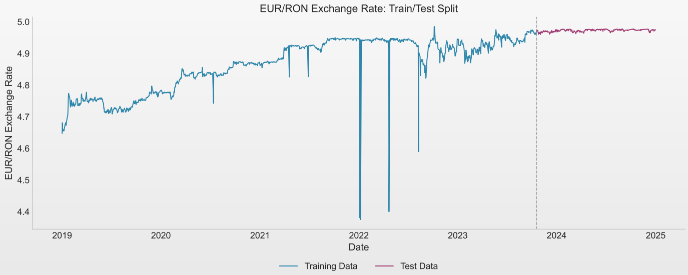
- MAPE: **2.2%**
- Captures weekly patterns

Why RF Outperformed?

- Good feature engineering
- Robust to outliers

TSA_ch8_best_model

EUR/RON Exchange Rate Visualization

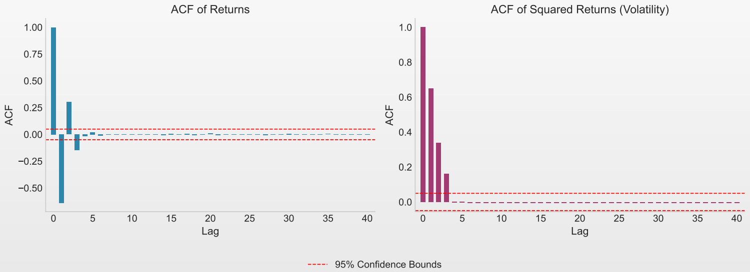


Interpretation

- **Data**: EUR/RON daily exchange rate (Yahoo Finance, 2019–2025), 80/20 train/test split
- **Level**: Depreciation trend and periods of high volatility
- **Returns**: Volatility clustering (periods of high volatility are followed by similar periods)

TSA_ch8_case_raw_data

ACF Analysis: Returns vs Squared Returns

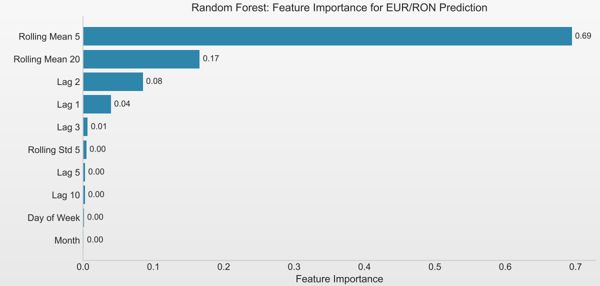


Interpretation

- **Data**: EUR/RON daily returns and squared returns (Yahoo Finance, 2019–2025)
- **Left**: ACF of returns → rapid decay, no significant autocorrelation after lag 1
- **Right**: ACF of squared returns → slow decay indicates **volatility clustering** (ARCH effects)

TSA_ch8_case_acf_analysis

Random Forest: Feature Importance

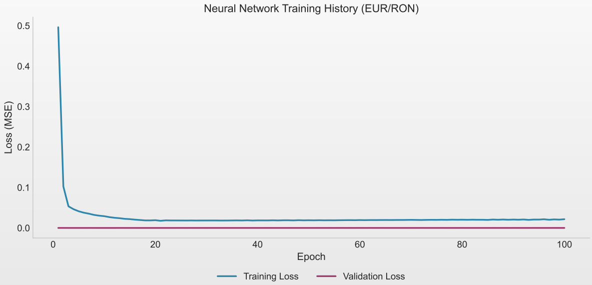


Interpretation

- **Data**: EUR/RON exchange rate (Yahoo Finance, 2019–2025) — RF with 10 engineered features
- Recent lags (lag_1, lag_2) and rolling volatility are the most important features
- Calendar features have minor impact for daily exchange rate prediction

TSA_ch8_case_feature_importance

LSTM: Learning Curve

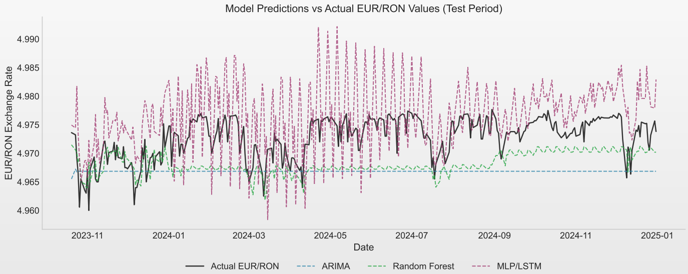


Interpretation

- **Data**: EUR/RON exchange rate (Yahoo Finance, 2019–2025) — Neural Network (100 epochs, MSE loss)
- **Training Loss**: Decreases rapidly in early epochs, then stabilizes
- **Validation Loss**: Tracks training loss → no severe overfitting

TSA_ch8_case_lstm_training

EUR/RON: Predictions vs Actual Values

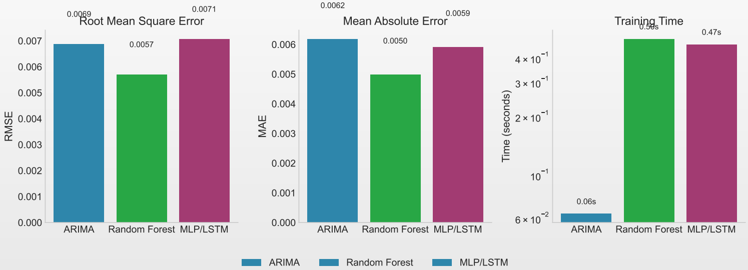


Interpretation

- **Data**: EUR/RON test period — ARIMA, Random Forest, MLP/LSTM predictions vs actual
- All models capture the general pattern, but none perfectly predicts volatility spikes
- This reflects **market efficiency** and **prediction limits** for financial series

TSA_ch8_case_predictions

EUR/RON: Model Performance Comparison



Interpretation

- **Data**: EUR/RON exchange rate (Yahoo Finance, 2019–2025) — ARIMA vs RF vs MLP/LSTM
- **Left**: Error metrics (lower = better) → RF achieves the lowest RMSE and MAE
- **Right**: Training time (log scale) → ML models require more computational resources

TSA_ch8_case_comparison

Practical Summary: Model Selection

Criterion	**ARIMA**	**ARFIMA**	**RF**	**LSTM**
Data needed	Few	Few	Medium	Many
Long memory	No	**Yes**	Partial	Partial
Nonlinearity	No	No	**Yes**	**Yes**
Interpretability	**Yes**	**Yes**	Partial	No
Computation time	Fast	Fast	Medium	Slow
Exog. variables	Limited	Limited	**Yes**	**Yes**

Rule: Start simple (ARIMA), increase complexity only if out-of-sample performance improves.

Common Mistakes to Avoid

Data Leakage

- Using future data in features
- Standard k-fold CV on time series
- Scaling with full dataset stats

Solution: Always use **walk-forward** validation

Wrong Model Choice

- LSTM with 100 observations
- ARIMA for nonlinear patterns
- Ignoring interpretability needs

Solution: Match model to **data size & complexity**

Overfitting

- Too many features
- Too complex models
- Training too long (LSTM)

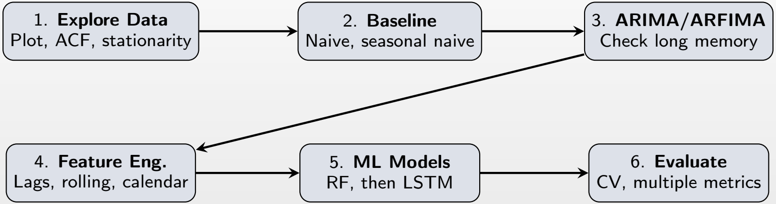
Solution: Use **validation set**, early stopping

Poor Evaluation

- Only using RMSE
- Ignoring prediction intervals
- No baseline comparison

Solution: Multiple metrics, **always compare to naive**

Practical Workflow: Step-by-Step



Golden Rules

- **Start simple**: Beat the baseline first, then add complexity
- **Validate properly**: Time series CV, not random splits
- **Iterate**: Feature engineering often matters more than model choice

Summary

What We Learned

- **ARFIMA**: Extends ARIMA for long memory processes (fractional d)
- **Random Forest**: Ensemble of trees for nonlinear relationships
- **LSTM**: Deep learning for complex sequential dependencies
- **Trade-offs**: Complexity vs interpretability vs data requirements

Key Takeaway

- **Parsimony Principle**:
  - Simple models often outperform complex ones
  - Always benchmark against naive methods

Quick Quiz

1. What does $d = 0.3$ mean in an ARFIMA model?
2. Why use Time Series CV instead of standard k-fold?
3. What is the main advantage of LSTM over simple RNNs?
4. What type of model would you choose with small data and linear relationships?
5. What does "data leakage" mean in the context of ML for time series?

Quiz Answers

1. $d = 0.3$: Long memory, the series is stationary but autocorrelations decay slowly (hyperbolically). Moderate persistence.
2. **Time Series CV**: To respect temporal order. Standard k-fold would use future data to predict the past (data leakage).
3. **LSTM vs RNN**: LSTM solves the "vanishing gradient" problem through the gating mechanism, allowing learning of long-term dependencies.
4. **Small data, linear relationships**: ARIMA or ARFIMA. ML requires lots of data to generalize well.
5. **Data leakage**: Using future information in features or training. E.g., calculating moving averages using future data, or standard k-fold that mixes temporal order.

What Comes Next?

Chapter 9: Multiple Seasonalities

- **The Challenge**: Real data often has multiple seasonal patterns (daily + weekly + yearly)
- **TBATS**: Trigonometric seasonality, Box-Cox, ARMA errors, Trend, Seasonal
  - Automatic, handles high-frequency data with Fourier terms
- **Prophet** (Taylor & Letham, 2018): Decomposable model
  - Interpretable components (trend + seasonality + holidays)

Questions?

Key References

Foundational Papers

- Box & Jenkins (1970). *Time Series Analysis: Forecasting and Control*. Holden-Day.
- Granger & Joyeux (1980). Long-memory time series models. *J. Time Series Analysis*, 1(1).
- Hosking (1981). Fractional differencing. *Biometrika*, 68(1).

Machine Learning Methods

- Breiman (2001). Random Forests. *Machine Learning*, 45(1), 5-32.
- Hochreiter & Schmidhuber (1997). Long short-term memory. *Neural Computation*, 9(8).

Forecasting Competitions & Reviews

- Makridakis et al. (2018). The M4 Competition. *Int. J. Forecasting*, 34(4).
- Hyndman & Athanasopoulos (2021). *Forecasting: Principles and Practice*, 3rd ed.

Online Resources and Code

- **Quantlet**: https://quantlet.com → Code repository for statistics
- **Quantinar**: https://quantinar.com → Learning platform for quantitative methods
- **GitHub TSA_ch8**: https://github.com/QuantLet/TSA/tree/main/TSA_ch8

Thank You!

Questions?

Course materials available at: https://danpele.github.io/Time-Series-Analysis/

Quantlet Quantinar