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	✓	✓	∼	×

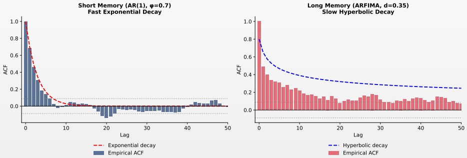
Principle of Parsimony (Occam's Razor)

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.

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

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

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

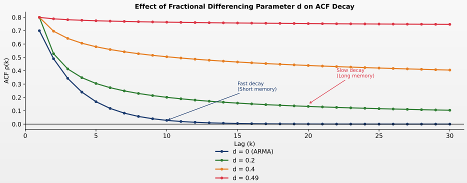
$(1-L)^d = \sum_{k=0}^{\infty} \binom{d}{k}(-L)^k = 1 - dL - \frac{d(1-d)}{2!}L^2 - \frac{d(1-d)(2-d)}{3!}L^3 - \cdots$

- $d = 0$: 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)

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

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

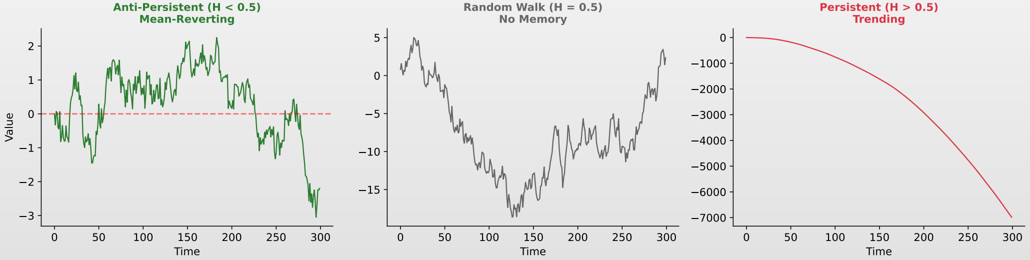
Relationship with Hurst exponent: $d = H - 0.5$

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

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

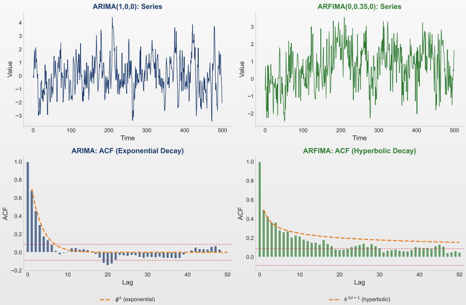


TSA_ch8_hurst_interpretation

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

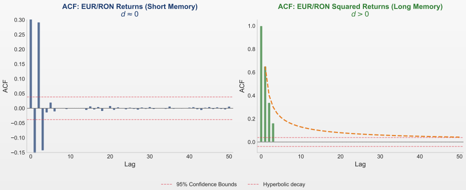


TSA_ch8_arima_vs_arfima

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

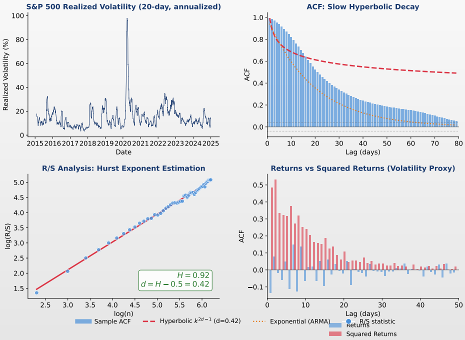
ARFIMA Example: S&P 500 Realized Volatility

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!

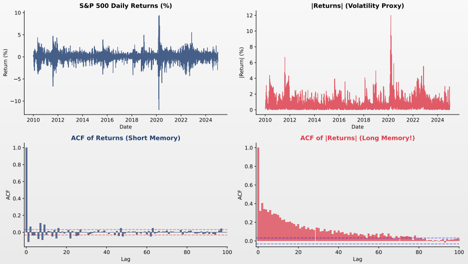


TSA_ch8_arfima_sp500

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

Estimating the Parameter d

Estimation Methods

- **GPH (Geweke-Porter-Hudak)**: Frequency-domain regression
  - $\ln I(\omega_j) = c - d \cdot \ln\left(4 \sin^2 \frac{\omega_j}{2}\right) + \varepsilon_j$
- **R/S (Rescaled Range)**: Hurst's method
  - $\frac{R}{S}(n) \sim c \cdot n^H$
- **MLE (Maximum Likelihood)**: Full ARFIMA estimation
- **Whittle**: Efficient frequency-domain approximation

Implementation

- In Python: `arch` package, `statsmodels.tsa.arima.model.ARIMA` with `order=(p,d,q)` where d can be fractional

ARFIMA Example in Python

Python Code

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(y, order=(1, 0.3, 1))
results = model.fit()
```

Note

- ARFIMA estimation requires specialized packages. In practice, the `arch` or `fracdiff` packages are commonly used in Python.

Random Forest: Basic Concepts

What is Random Forest? (Breiman, 2001)

- **Ensemble** of decision trees
- Each tree trained on a **bootstrap subset** of the data
- At each node, a **random** subset of features is selected
- Final prediction = **average** of all tree predictions

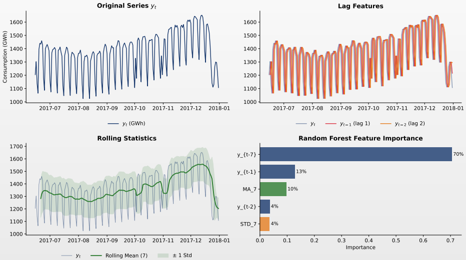
Advantages for Time Series

- Captures **nonlinear relationships**
- **Robust** to outliers and noise
- Does not require **stationarity**
- Provides **feature importance** (interpretability)
- Works well with **many variables**

Feature Engineering: Illustration

Interpretation

- **Data**: Germany daily electricity consumption (OPSD, 2012–2017)
- We transform the time series into features: lags, rolling statistics
- The RF model learns relationships between these and future values



TSA_ch8_feature_engineering

Data Preparation for Random Forest

Feature Engineering for Time Series

1. **Lag features**: $Y_{t-1}, Y_{t-2}, \ldots, Y_{t-p}$
2. **Rolling statistics**: moving average, standard deviation
3. **Calendar features**: day of week, month, season
4. **Trend features**: time, quadratic trend
5. **Exogenous variables**: economic indicators, events

Warning: Data Leakage!

- Do not use future information in features
- Train/test split: **temporal**, not random!
- Rolling statistics: compute only on **past** data

Random Forest: Python Implementation

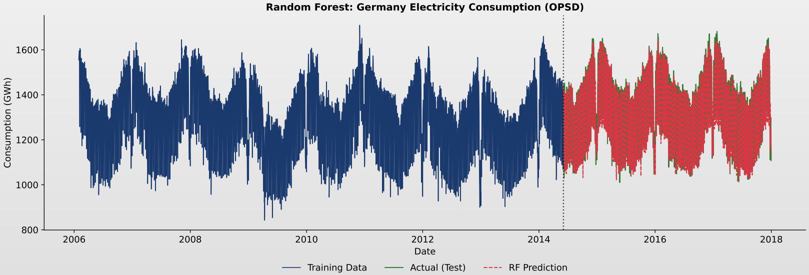
Python Code

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators=100, max_depth=10)
rf.fit(X_train, y_train)
predictions = rf.predict(X_test)
```

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

Feature Importance and Interpretation

Feature Importance

- **Mean Decrease Impurity (MDI)**: Impurity reduction at each split
- **Permutation Importance**: How much performance drops when a feature is randomly permuted

Typical Interpretation for Time Series

- `lag_1` very important $\Rightarrow$ Strong autocorrelation
- `rolling_mean` important $\Rightarrow$ Local trend matters
- `month` important $\Rightarrow$ Seasonality present

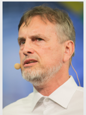
Code

- `rf.feature_importances_` or `permutation_importance(rf, X_test, y_test)`

Researcher Spotlight: Hochreiter & Schmidhuber

Sepp Hochreiter (*1967)

W Wikipedia



Jürgen Schmidhuber (*1963)

W Wikipedia

Biography

- **Sepp Hochreiter**: Austrian computer scientist, Professor at Johannes Kepler University Linz and head of ELLIS Unit Linz
- **Jürgen Schmidhuber**: German-Swiss computer scientist, Scientific Director of IDSIA
- Together they solved the vanishing gradient problem

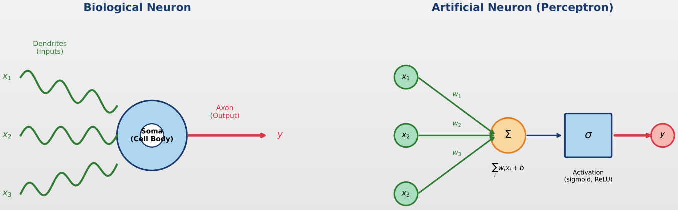
Key Contributions

- **Long Short-Term Memory** (LSTM, 1997) — gated recurrent architecture solving the vanishing gradient problem
- **Vanishing gradient analysis** (Hochreiter, 1991) — identified the fundamental training problem in deep networks
- **Forget gate** extension (Gers et al., 2000) — crucial addition enabling practical LSTM use
- Foundation for modern sequence modeling in NLP, speech, and time series

From Biological to Artificial Neurons

The Analogy

- **Dendrites → Inputs** x_i **Synapses → Weights** w_i **Soma → Sum + Activation** **Axon → Output** y



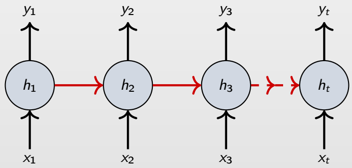
Dendrites → Inputs with weights | Soma → Weighted sum + activation | Axon → Output

TSA_ch8_neuron_comparison

Recurrent Neural Networks (RNN)

Basic Idea

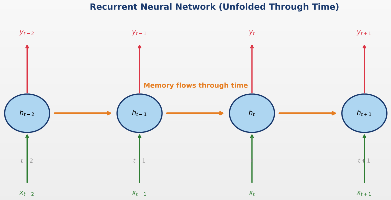
- Networks that process **sequences** of data
- Have **internal memory** (hidden state)
- Current state depends on input + previous state



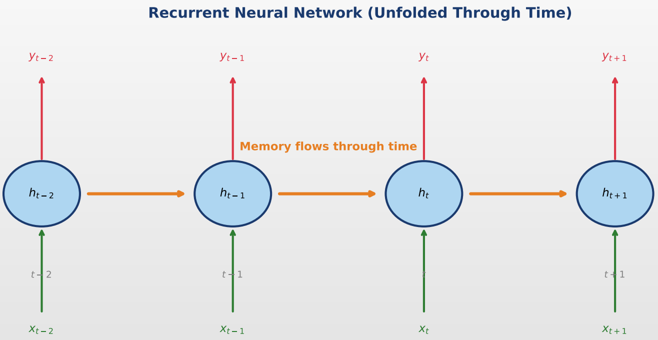
Problem: Vanishing Gradient

- Simple RNNs "forget" information from the distant past.

RNN Unfolded in Time

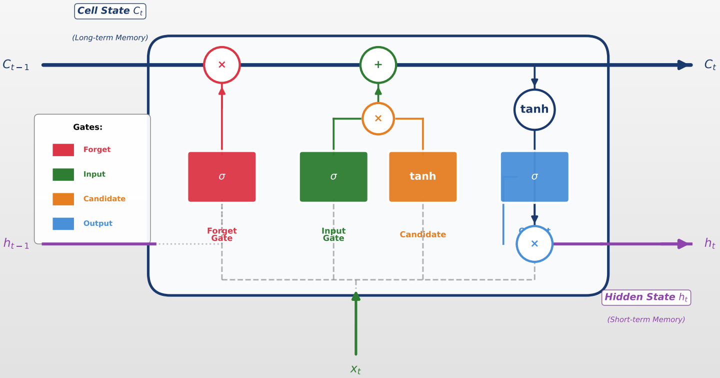


RNN Unfolded in Time



TSA_ch8_rnn_unfolded

LSTM Cell: Detailed Diagram



Forget Gate f_t
$\sigma(W_f[h_{t-1}, x_t] + b_f)$
What to forget?

Input Gate i_t
$\sigma(W_i[h_{t-1}, x_t] + b_i)$
What to store?

Output Gate o_t
$\sigma(W_o[h_{t-1}, x_t] + b_o)$
What to transmit?

TSA_ch8_lstm_cell

LSTM: Long Short-Term Memory

The LSTM Solution

- **Concept**: Special cells with 3 gates that control information flow
- **Forget Gate** (f_t): What to forget from previous memory
- **Input Gate** (i_t): What new information to add
- **Output Gate** (o_t): What to send to output

LSTM Equations

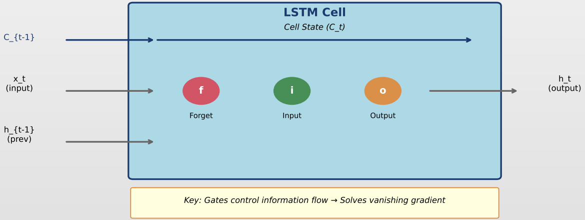
-

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

LSTM Cell Architecture

Interpretation

- 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

LSTM Advantages for Time Series

Why LSTM?

- Captures **long-term dependencies** (unlike simple RNN)
- Learns **complex patterns** and nonlinear relationships
- Handles **variable-length sequences**
- Works well with **multivariate data**

Disadvantages

- Requires **large datasets** for training
- **Computationally intensive**
- "**Black box**" – difficult to interpret
- Sensitive to **hyperparameters**
- Can **overfit** easily

LSTM: Python Implementation with Keras

Python Code

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(n, 1)),
    Dropout(0.2),
    LSTM(50),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')
```

Data Preparation for LSTM

Essential Steps

1. **Normalization/Scaling**: MinMaxScaler or StandardScaler
2. **Create sequences**: Sliding window for input
3. **Reshape**: 3D format (samples, timesteps, features)
4. **Train/Test split**: Temporal, not random!

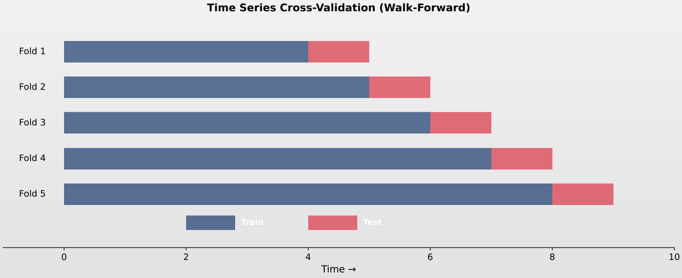
Sequence Creation Example

```
def create_sequences(data, n_steps):
    X, y = [], []
    for i in range(len(data) - n_steps):
        X.append(data[i:(i + n_steps)])
        y.append(data[i + n_steps])
    return np.array(X), np.array(y)

X, y = create_sequences(scaled_data, 10)
```

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

Evaluation Metrics

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

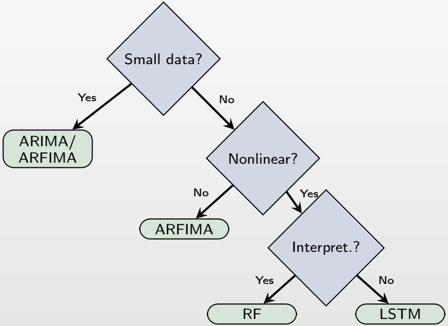
Common Metrics

- **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

Model Selection Guide



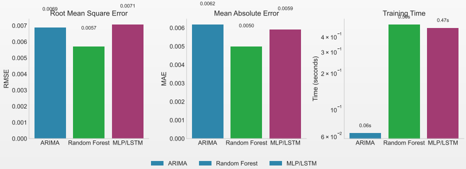
Trade-off

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

Bitcoin: Price Evolution and Returns

Key Observations

- Exponential price growth → strongly **leptokurtic** distribution
- Daily returns: mean ≈ 0.15%, volatility ≈ 3.5%
- Volatility clustering evident → crisis periods (2018, 2020, 2022)
- Kurtosis ≈ 10–15 (well above the normal's 3)

Case Study: Bitcoin Price Forecasting

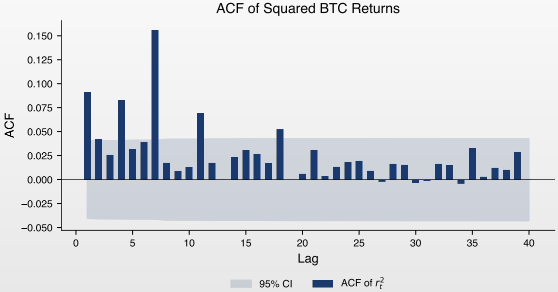
Why Bitcoin?

- **Extreme** volatility and complex patterns
- Potential **long memory** in volatility
- **Nonlinear** relationships with exogenous variables
- Data available at **high frequency**

Comparative Approach

1. ARIMA on returns
2. ARFIMA for long memory
3. Random Forest with technical features
4. LSTM on price sequences

Bitcoin: ACF and Evidence for Long Memory



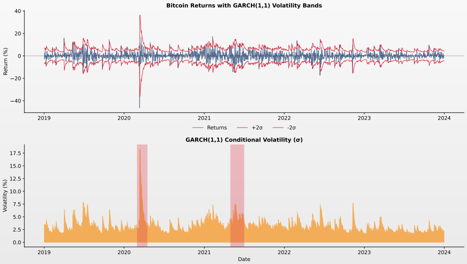
ACF Analysis

- ACF of returns: rapid decay → short memory in the mean
- ACF of squared returns: **slow, hyperbolic** decay
  - Indicates **long memory in volatility**
  - Hurst $H \approx 0.65$–0.70 ($d \approx 0.15$–0.20)
- ARFIMA on volatility > ARMA → captures shock persistence

Bitcoin: GARCH and Risk Management

Conclusions – Bitcoin Case Study

- Differences between models are **small** for mean returns
- Major added value: **volatility modeling** (GARCH, EGARCH)
- ARFIMA captures volatility persistence (long memory)
- Random Forest: useful for **nonlinear features** (volume, sentiment)
- Optimal combination: ARFIMA-GARCH + exogenous features via RF



Bitcoin: ARFIMA Estimation and Model Comparison

Python Code – Bitcoin Long Memory Estimation

```
import yfinance as yf
btc = yf.download('BTC-USD', start='2018-01-01', end='2024-12-31')
returns = np.log(btc['Close']).diff().dropna() * 100

# Hurst exponent on squared returns (volatility)
from hurst import compute_Hc
H, c, _ = compute_Hc(returns.values**2, kind='change')
print(f"Hurst (volatility): {H:.3f}, d = {H-0.5:.3f}")

# Comparison ARIMA vs Random Forest
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
# ... (similar to EUR/RON, with adapted features)
```

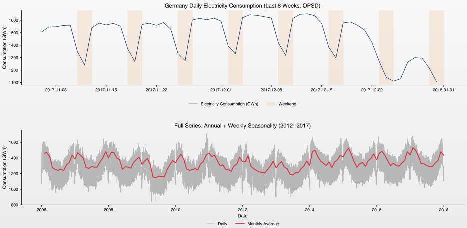
Typical Bitcoin Results (RMSE on returns)

Model	**RMSE**	**MAE**	**Interpretable?**
ARIMA(1,0,1)	3.82	2.41	Yes
ARFIMA(1,d,1)	3.79	2.38	Yes
Random Forest	3.65	2.29	Partial
LSTM	3.71	2.33	No

Energy: Demand Visualization and Multiple Seasonality

Identified Patterns

- **Daily** (24h): peak morning (8–10) and evening (18–21), minimum at night
- **Weekly** (168h): reduced consumption on weekends ($\sim$15–20% less)
- **Annual** (8766h): peak in summer (AC) and winter (heating)
- SARIMA cannot simultaneously model these 3 periods!



Case Study: Energy Consumption Forecasting

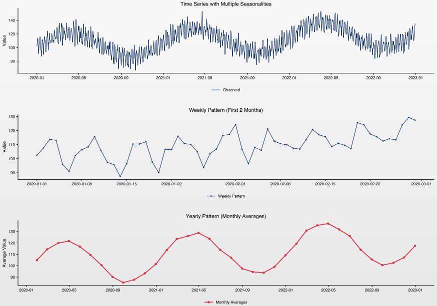
Characteristics

- **Multiple seasonality**: daily, weekly, annual
- **Long-term trend** of growth
- **Exogenous variables**: temperature, holidays, price
- **Anomalies**: special events, outages

Challenges

- Patterns at different temporal scales
- Complex interactions between variables
- Need for forecasts at different horizons

Energy: Why Prophet and TBATS?



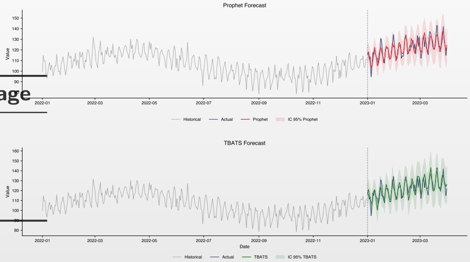
Solution: Models with Multiple Seasonality

- **TBATS**: periods $[24, 168, 8766] \to$ Fourier for each season
  - Automatic, no manual tuning, good for production
- **Prophet**: additive/multiplicative seasonality + regressors
  - Add temperature, holidays, special events
- **Classical ARIMA**: can handle only 1 season $\to$ MAPE $\approx$ 8–10%

Energy: Prophet Decomposition and Results

Comparison Results on Energy Data (MAPE)

Model	MAPE	RMSE (MW)	95% Coverage
SARIMA (1 season)	8.5%	450	75%
TBATS	4.2%	220	82%
Prophet	4.8%	250	85%
Prophet + regressors	**3.9%**	**200**	**88%**



Energy: Conclusions and Practical Recommendations

Lessons Learned

- Models with **multiple seasonality** reduce MAPE by ∼50% compared to SARIMA
- **Exogenous variables** (temperature) bring an additional 10–15% gain
- Prophet excels at **interpretability**: trend + season + holiday decomposition
- TBATS: best **out-of-the-box** → no hyperparameter tuning needed

When to Choose Each Model?

- **Prophet**: when you have external regressors + interpretation for management
- **TBATS**: automation, production, no human intervention
- **LSTM/RF**: if you have >100,000 observations and complex nonlinear patterns

Full details on Prophet and TBATS → Chapter 9

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: EUR/RON Exchange Rate Forecasting

Why EUR/RON?

- Relevance for the Romanian economy
- Potential **long memory** (shock persistence)
- Patterns influenced by **macroeconomic factors**
- Easily accessible data (BNR, Yahoo Finance)

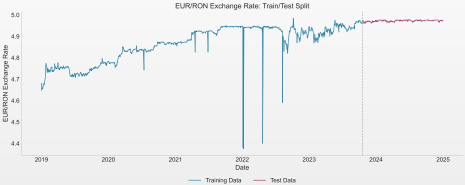
Objective

- We compare ARIMA, ARFIMA, Random Forest, and LSTM on the same data to understand the strengths of each method

EUR/RON Exchange Rate Visualization

Interpretation

- **Data**: EUR/RON daily exchange rate (Yahoo Finance, 2019–2025)
- **Top**: EUR/RON rate — depreciation trend and periods of high volatility
- **Bottom**: Daily returns — volatility clustering (periods of high volatility are followed by similar periods)



TSA_ch8_case_raw_data

Step 1: Loading and Visualizing the Data

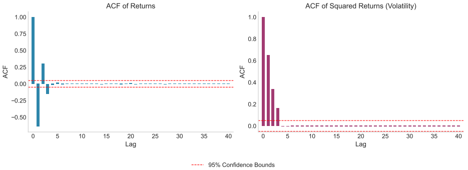
Python Code – Data Download

```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Download EUR/RON data (or EURRON=X)
data = yf.download('EURRON=X', start='2015-01-01', end='2024-12-31')
df = data[['Close']].dropna()
df.columns = ['EURRON']

# Compute log returns
df['Returns'] = np.log(df['EURRON']).diff() * 100
df = df.dropna()

print(f"Period: {df.index[0]} - {df.index[-1]}")
print(f"Observations: {len(df)}")
print(f"Mean returns: {df['Returns'].mean():.4f}%")
print(f"Volatility: {df['Returns'].std():.4f}%")
```

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

Step 2: Testing for Long Memory

Python Code – Estimating *d* and the Hurst Test

```
from arch.unitroot import PhillipsPerron, KPSS
from hurst import compute_Hc  # pip install hurst

# Phillips-Perron test for stationarity
pp_test = PhillipsPerron(df['Returns'])
print(f"Phillips-Perron p-value: {pp_test.pvalue:.4f}")

# Estimating the Hurst exponent
H, c, data_rs = compute_Hc(df['Returns'].values, kind='change')
d_estimated = H - 0.5

print(f"Hurst Exponent (H): {H:.4f}")
print(f"Estimated parameter d: {d_estimated:.4f}")

# Interpretation
if H > 0.5:
    print("PERSISTENT series (trend-following)")
elif H < 0.5:
    print("ANTI-PERSISTENT series (mean-reverting)")
else:
    print("Random walk")
```

Long Memory Test Results – EUR/RON

Typical Output

- `Phillips-Perron p-value: 0.0001` (returns are stationary)
- `Hurst Exponent (H): 0.47`
- `Estimated parameter d: -0.03`
- `Slightly ANTI-PERSISTENT series (mean-reverting)`

Interpretation

- EUR/RON returns are **stationary** (p-value < 0.05)
- $H \approx 0.47 < 0.5$: slight mean-reversion tendency
- $d \approx 0$: **short memory** – ARMA may be sufficient
- However, **volatility** may have long memory!

Step 3: ARIMA Model

Python Code – ARIMA with Automatic Selection

```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error, mean_absolute_error
import warnings
warnings.filterwarnings('ignore')

# Split data: 80% train, 20% test
train_size = int(len(df) * 0.8)
train, test = df['Returns'][:train_size], df['Returns'][train_size:]

# Fit ARIMA(1,0,1) - simple and efficient for returns
model_arima = ARIMA(train, order=(1, 0, 1))
results_arima = model_arima.fit()

# Forecast
forecast_arima = results_arima.forecast(steps=len(test))

# Evaluation
rmse_arima = np.sqrt(mean_squared_error(test, forecast_arima))
mae_arima = mean_absolute_error(test, forecast_arima)
print(f"ARIMA(1,0,1) - RMSE: {rmse_arima:.4f}, MAE: {mae_arima:.4f}")
```

Step 4: ARFIMA Model (Long Memory)

Python Code – ARFIMA with arch package

```
from arch import arch_model

# ARFIMA(1,d,1) using arch for robust estimation
# Note: arch estimates d automatically in GARCH context

# Alternatively, use statsmodels with fractional d
from statsmodels.tsa.arima.model import ARIMA

# Estimate d using GPH or set manually
d_frac = 0.1  # or the previously estimated value

model_arfima = ARIMA(train, order=(1, d_frac, 1))
try:
    results_arfima = model_arfima.fit()
    forecast_arfima = results_arfima.forecast(steps=len(test))
    rmse_arfima = np.sqrt(mean_squared_error(test, forecast_arfima))
    print(f"ARFIMA(1,{d_frac},1) - RMSE: {rmse_arfima:.4f}")
except:
    print("ARFIMA requires d between -0.5 and 0.5 for stationarity")
```

Step 5: Random Forest – Data Preparation

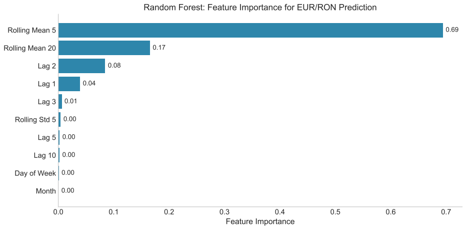
Python Code – Feature Engineering

```
from sklearn.ensemble import RandomForestRegressor

# Create features for Random Forest
def create_features(data, lags=5):
    df_feat = pd.DataFrame(index=data.index)
    df_feat['target'] = data.values

    # Lag features
    for i in range(1, lags + 1):
        df_feat[f'lag_{i}'] = data.shift(i)

    # Rolling statistics
    df_feat['rolling_mean_5'] = data.rolling(5).mean()
    df_feat['rolling_std_5'] = data.rolling(5).std()
    df_feat['rolling_mean_20'] = data.rolling(20).mean()

    # Calendar features
    df_feat['dayofweek'] = data.index.dayofweek
    df_feat['month'] = data.index.month

    return df_feat.dropna()

df_rf = create_features(df['Returns'], lags=10)
```

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 return prediction



TSA_ch8_case_feature_importance

Step 5: Random Forest – Training and Evaluation

Python Code – Random Forest Model

```
# Split data
X = df_rf.drop('target', axis=1)
y = df_rf['target']

train_size = int(len(df_rf) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Train Random Forest
rf_model = RandomForestRegressor(
    n_estimators=100,
    max_depth=10,
    min_samples_split=5,
    random_state=42
)
rf_model.fit(X_train, y_train)

# Prediction and evaluation
pred_rf = rf_model.predict(X_test)
rmse_rf = np.sqrt(mean_squared_error(y_test, pred_rf))
print(f"Random Forest - RMSE: {rmse_rf:.4f}")
```

Step 6: LSTM – Data Preparation

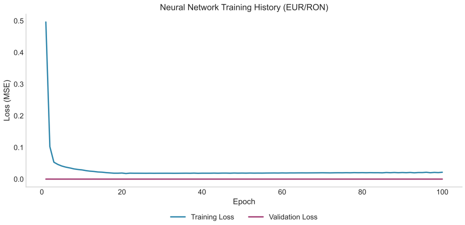
Python Code – Sequences for LSTM

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.preprocessing import MinMaxScaler

# Scale data between 0 and 1
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df['Returns'].values.reshape(-1, 1))

# Create sequences
def create_sequences(data, seq_length=20):
    X, y = [], []
    for i in range(seq_length, len(data)):
        X.append(data[i-seq_length:i, 0])
        y.append(data[i, 0])
    return np.array(X), np.array(y)

X_lstm, y_lstm = create_sequences(scaled_data, seq_length=20)
X_lstm = X_lstm.reshape((X_lstm.shape[0], X_lstm.shape[1], 1))

# Split
split = int(len(X_lstm) * 0.8)
X_train_lstm, X_test_lstm = X_lstm[:split], X_lstm[split:]
y_train_lstm, y_test_lstm = y_lstm[:split], y_lstm[split:]
```

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

Step 6: LSTM – Architecture and Training

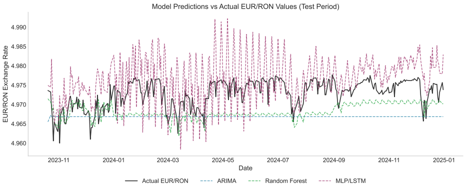
Python Code – LSTM Model

```
# Build the LSTM model
model_lstm = Sequential([
    LSTM(50, return_sequences=True, input_shape=(20, 1)),
    Dropout(0.2),
    LSTM(50, return_sequences=False),
    Dropout(0.2),
    Dense(25),
    Dense(1)
])

model_lstm.compile(optimizer='adam', loss='mse')

# Train
history = model_lstm.fit(
    X_train_lstm, y_train_lstm,
    epochs=50, batch_size=32,
    validation_split=0.1, verbose=0
)

# Prediction
pred_lstm_scaled = model_lstm.predict(X_test_lstm)
pred_lstm = scaler.inverse_transform(pred_lstm_scaled)
y_test_original = scaler.inverse_transform(y_test_lstm.reshape(-1, 1))
rmse_lstm = np.sqrt(mean_squared_error(y_test_original, pred_lstm))
print(f"LSTM - RMSE: {rmse_lstm:.4f}")
```

Visualization: Predictions vs Actual Values

Interpretation

- **Data**: EUR/RON test period — ARIMA, Random Forest, MLP/LSTM predictions vs actual values
- 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

Comparison: Results on EUR/RON

Model	RMSE	MAE	Time (s)	Interpretable?
ARIMA(1,1,1)	0.0069	0.0062	0.08	Yes
Random Forest	0.0057	0.0050	0.51	Yes (features)
MLP/LSTM	0.0071	0.0059	0.47	No

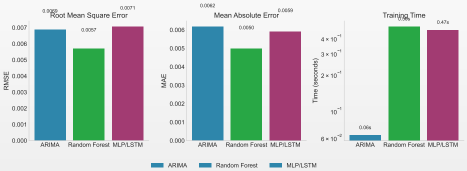
Conclusions

- For EUR/RON, the differences are **small** – the market is efficient
- Random Forest offers the best **accuracy/interpretability** trade-off
- LSTM has high computational cost for marginal gain
- ARIMA remains a solid choice for **baseline**

Model Comparison: Performance Metrics

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

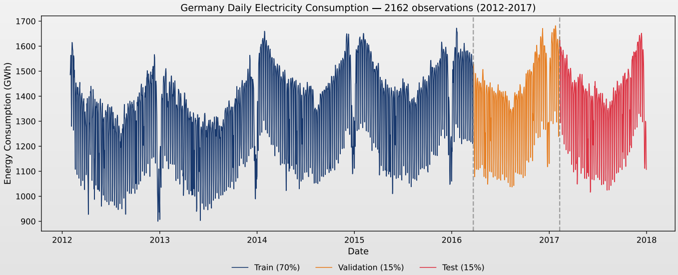


TSA_ch8_case_comparison

Case Study: Data Overview

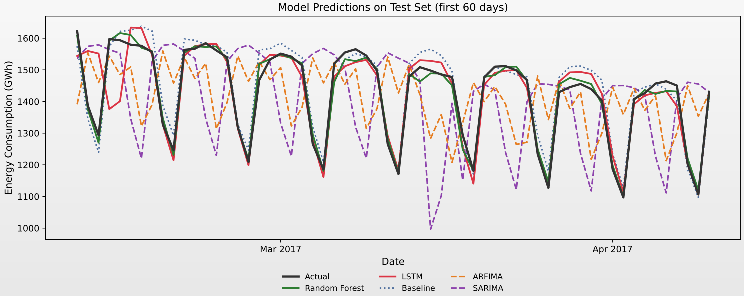
Interpretation

- **Train**: 1513 obs (70%) **Validation**: 324 obs (15%) **Test**: 325 obs (15%)



TSA_ch8_data_split

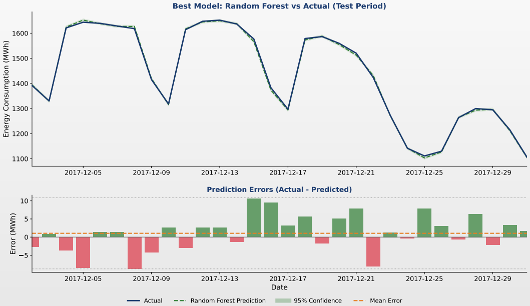
Case Study: Model Predictions



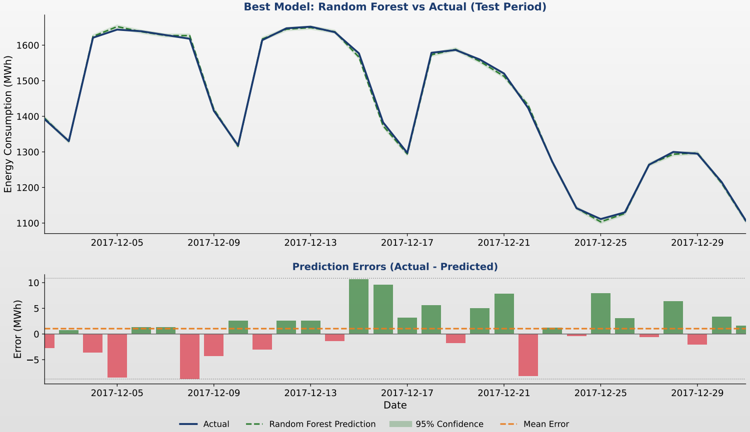
Rank	Model	MAPE	Interpretation
1	Random Forest	2.2%	Best: captures nonlinear patterns
2	LSTM	3.3%	Good, needs more data
3	Baseline	3.9%	Simple but competitive
4	ARFIMA	12.3%	Long memory not sufficient
5	SARIMA	14.6%	Struggles with patterns

TSA_ch8_model_predictions

Case Study: Best Model Performance



Case Study: Best Model Performance



TSA_ch8_best_model

When to Choose Each Model?

ARIMA/ARFIMA

- Few data (< 500 obs.)
- Interpretation important
- Suspected long memory
- Quick baseline

LSTM/Deep Learning

- Very large data ($> 10,000$)
- Complex sequences
- Computational resources
- Hidden patterns

Random Forest

- Many exogenous variables
- Nonlinear relationships
- Feature importance
- Moderate data

Golden Rule

- Start simple (ARIMA), add complexity only if performance increases significantly!

Example 2: BET Index (Bucharest Stock Exchange)

Characteristics

- Strong **volatility clustering**
- Influenced by international markets
- Lower liquidity than developed markets
- Potential for long memory in volatility

Typical Results (RMSE on returns)

- GARCH(1,1): 1.45 – best for volatility
- ARFIMA for volatility: 1.52
- Random Forest: 1.48
- LSTM: 1.51

Example 3: Romania Inflation Rate

Characteristics

- **Monthly** series (low frequency)
- **High persistence** – shocks last long
- Influenced by monetary policy
- Strong potential for **long memory**

Typical Results

- ARFIMA with $d \approx 0.35$ – captures persistence
- ARIMA underestimates shock persistence
- ML does not work well (few data, 300 obs.)

- **Lesson**: For monthly series with few data, classical models (ARFIMA) are superior!

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**

Recommended Workflow

1. Start with **ARIMA** as baseline
2. Test for **long memory** $\to$ ARFIMA if d is significant
3. Add **features** $\to$ Random Forest
4. Only with lots of data and resources $\to$ LSTM

AI Exercise: Critical Thinking

Prompt to test in ChatGPT / Claude / Copilot

"I have 5 years of daily electricity demand data. Compare ARIMA, Random Forest, and LSTM for 7-day-ahead forecasting. Which model is best? Give me complete Python code with comparison."

Exercise:

1. Run the prompt in an LLM of your choice and critically analyze the response.
2. How are features engineered for Random Forest? Lags, calendar variables, Fourier terms?
3. Is the LSTM properly structured? Input shape, scaling, train/test split without leakage?
4. Does it use walk-forward validation or just a single train/test split?
5. Does it mention interpretability and computational cost trade-offs?

Warning: AI-generated code may run without errors and look professional. *That does not mean it is correct.*

Summary

What We Learned

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

Practical Recommendations

- Start with **simple** models (ARIMA) as baseline
- Use **Time Series CV** for proper evaluation
- ML requires careful **feature engineering**
- LSTM: only with **lots of data** and computational resources

Quiz Question 1

Question

What does $d = 0.3$ mean in an ARFIMA model?

(A) The series needs 0.3 differences to become stationary

(B) Long memory: stationary but ACF decays hyperbolically (slowly)

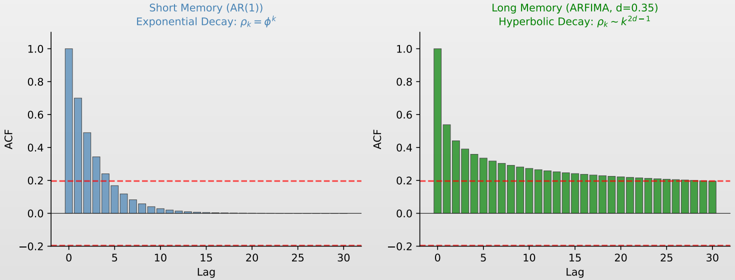
(C) The series is non-stationary with a unit root

(D) Short memory: ACF decays exponentially (fast)

Quiz Question 1: Answer

Correct Answer: (B) Long memory with hyperbolic ACF decay

For $0 < d < 0.5$: stationary but ACF $\sim k^{2d-1}$ decays much slower than exponential. This "long memory" means distant observations still matter.



TSA_ch8_quiz1_long_memory

Quiz Question 2

Question

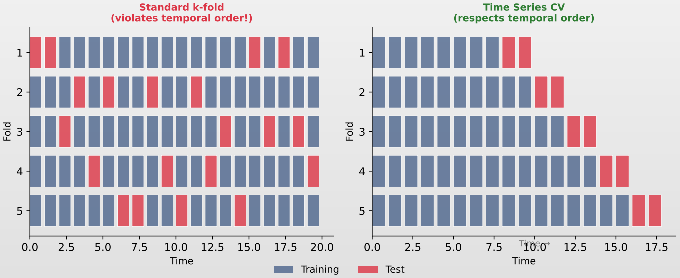
Why should you use Time Series Cross-Validation instead of standard k-fold?

(A) k-fold is computationally more expensive
(B) Time Series CV uses more training data
(C) k-fold violates temporal order, causing data leakage
(D) There is no difference; both methods are equivalent

Quiz Question 2: Answer

Correct Answer: (C) k-fold violates temporal order

Standard k-fold randomly shuffles data, using future observations to predict past ones. Time Series CV always trains on past and tests on future, respecting causality.



TSA_ch8_quiz2_timeseries_cv

Quiz Question 3

Question

What is the main advantage of LSTM over simple RNNs?

(A) LSTM uses fewer parameters

(B) LSTM solves the vanishing gradient problem via gating mechanisms

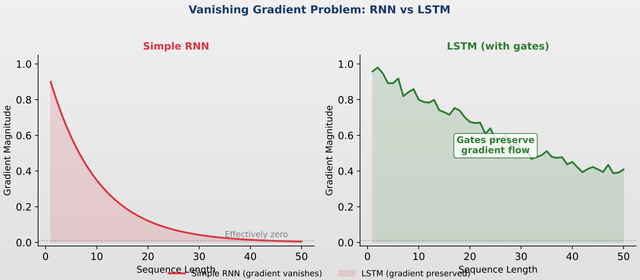
(C) LSTM is faster to train

(D) LSTM does not require sequential data

Quiz Question 3: Answer

Correct Answer: (B) Solves vanishing gradient via gates

LSTM's forget, input, and output gates control information flow, preserving gradients across long sequences. Simple RNNs lose gradient signal after ∼10–20 steps.



TSA_ch8_quiz3_lstm_gates

Quiz Question 4

Question

You have a small dataset (100 observations) with linear relationships. Which model is most appropriate?

(A) LSTM — deep learning captures all patterns

(B) Random Forest — handles any relationship

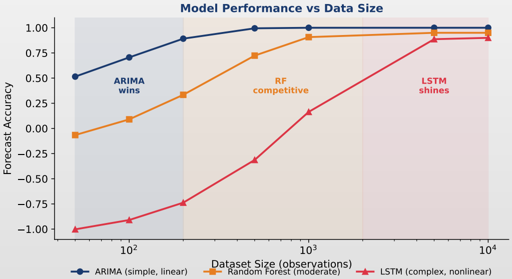
(C) ARIMA/ARFIMA — parsimonious and effective with small data

(D) Ensemble of all models for best accuracy

Quiz Question 4: Answer

Correct Answer: (C) ARIMA/ARFIMA — parsimonious for small data

ML models (RF, LSTM) need large datasets to generalize. With 100 observations and linear dynamics, ARIMA's few parameters avoid overfitting and often outperform complex models.



TSA_ch8_quiz4_model_complexity

Quiz Question 5

Question

What is "data leakage" in the context of ML for time series?

(A) Missing values in the dataset

(B) Using future information in features or during training

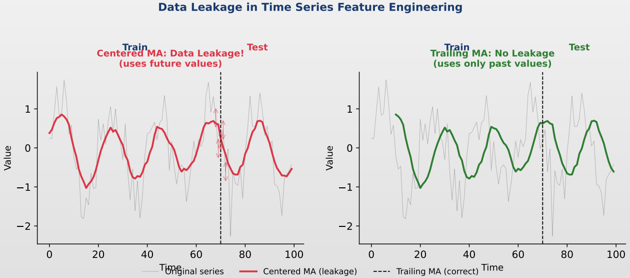
(C) Having too many features relative to observations

(D) The model memorizing the training data

Quiz Question 5: Answer

Correct Answer: (B) Using future information in features or training

Examples: centered moving averages (use future values), standard k-fold (mixes temporal order), computing statistics over the full dataset before splitting.



TSA_ch8_quiz5_data_leakage

What Comes Next?

Extensions and Advanced Topics

- **Transformer** for time series (Temporal Fusion Transformer)
- **Prophet** (Facebook/Meta) for seasonality
- **Neural Prophet**: Prophet + neural networks
- **Ensemble methods**: Combining multiple models
- **Anomaly detection** with ML

Questions?

Bibliography I

Long Memory and ARFIMA

- Granger, C.W.J., & Joyeux, R. (1980). An Introduction to Long-Memory Time Series Models and Fractional Differencing, *Journal of Time Series Analysis*, 1(1), 15–29.
- Baillie, R.T. (1996). Long Memory Processes and Fractional Integration in Econometrics, *Journal of Econometrics*, 73(1), 5–59.
- Beran, J. (1994). *Statistics for Long-Memory Processes*, Chapman & Hall.

Neural Networks and Deep Learning for Time Series

- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory, *Neural Computation*, 9(8), 1735–1780.
- Bai, J., & Perron, P. (2003). Computation and Analysis of Multiple Structural Change Models, *Journal of Applied Econometrics*, 18(1), 1–22.

Bibliography II

Threshold and Regime-Switching Models

- Hansen, B.E. (2011). Threshold Autoregression in Economics, *Statistics and Its Interface*, 4(2), 123–127.
- Hamilton, J.D. (1989). A New Approach to the Economic Analysis of Nonstationary Time Series and the Business Cycle, *Econometrica*, 57(2), 357–384.
- Petropoulos, F., et al. (2022). Forecasting: Theory and Practice, *International Journal of Forecasting*, 38(3), 845–1054.

Online Resources and Code

- **Quantlet**: https://quantlet.com → Code repository for statistics
- **Quantinar**: https://quantinar.com → Learning platform for quantitative methods
- **GitHub TSA**: https://github.com/QuantLet/TSA → Python code for this course

Thank You!

Questions?

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

Quantlet Quantinar