



Time Series Analysis and Forecasting

## Chapter 8: Modern Extensions

ARFIMA, Machine Learning, Deep Learning



# Contents

- |                                         |                                              |                                       |
|-----------------------------------------|----------------------------------------------|---------------------------------------|
| 1 Motivation                            | 5 Comparison and Model Selection             | 8 Final Comparison: All Methods       |
| 2 ARFIMA: Modele cu Long Memory         | 6 Applying Practice                          | 9 Exemple Suplimentare cu Yeste Reale |
| 3 Random Forest for Serii de Timp       | 7 Complete Case Study: EUR/RON Exchange Rate | 10 Summary and Quiz                   |
| 4 LSTM: Deep Learning for Serii de Timp |                                              |                                       |

## Learning Objectives

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

- ① Understand the concept of **long memory** in time series
- ② Estimate and interpret **ARFIMA**
- ③ Apply **Random Forest** for time series forecasting
- ④ Build **LSTM** networks for time series
- ⑤ Compare performance of classical vs ML models
- ⑥ Choose the appropriate method based on context
- ⑦ Implement these methods in **Python**

## Limitations of ARIMA Models

- Assume **short memory**: autocorrelations decay exponentially
- Relationships **linear** between variables
- Difficulties with **complex patterns** and nonlinear
- Requires **stationarity** (through differencing)

## Modern Solutions

- **ARFIMA**: Captures long memory (autocorrelations that decay slowly)
- **Random Forest**: Nonlinear relationships, robust to outliers
- **LSTM**: Complex sequential patterns, long-term dependencies

## When to Use Each Method?

Feature	ARIMA	ARFIMA	RF	LSTM
Long memory	✗	✓	✓	✓
Relationships nonlinear	✗	✗	✓	✓
Interpretability	✓	✓	~	✗
Few data	✓	✓	✗	✗
Exogenous variables	✓	✓	✓	✓
Uncertainty	✓	✓	~	✗

### Golden Rule

Start **simple** (ARIMA), then increase complexity only if justified by data and performance.

# What is Long Memory?

## Short Memory (ARMA)

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

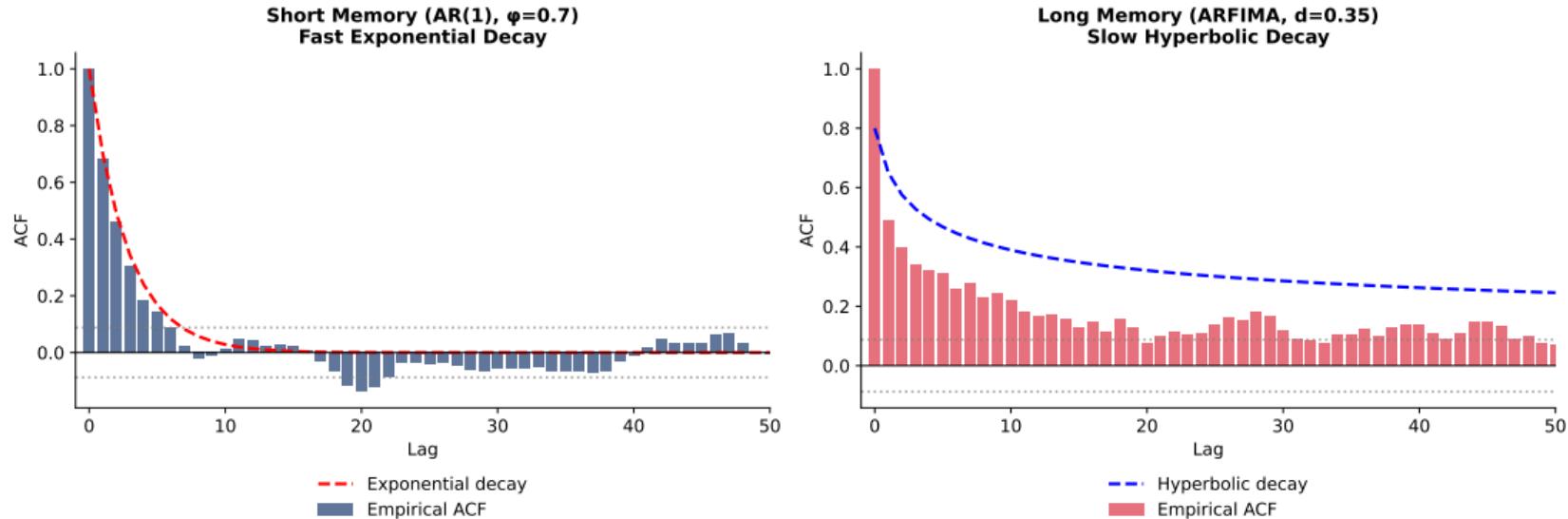
## Long Memory (ARFIMA)

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

## Exemple cu Long Memory

Financial market volatility, river flows, network traffic, inflation

## Comparație ACF: Short Memory vs Lungă



Left: AR(1) — autocorrelations decay exponentially (short memory)

Right: ARFIMA with  $d = 0.35$  — autocorrelations decay hyperbolically (long memory)

# The ARFIMA Model(p,d,q)

## Definition 1 (ARFIMA)

A process  $\{Y_t\}$  follows a **ARFIMA(p,d,q)** 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

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

- $d = 0$ : ARMA standard (short memory)
- $0 < d < 0.5$ : Long memory, stationarity
- $d = 0.5$ : Stationarity limit
- $0.5 \leq d < 1$ : Nonstationarity, non mean-reverting
- $d = 1$ : Random walk (ARIMA standard)

## Interpreting the Parameter $d$

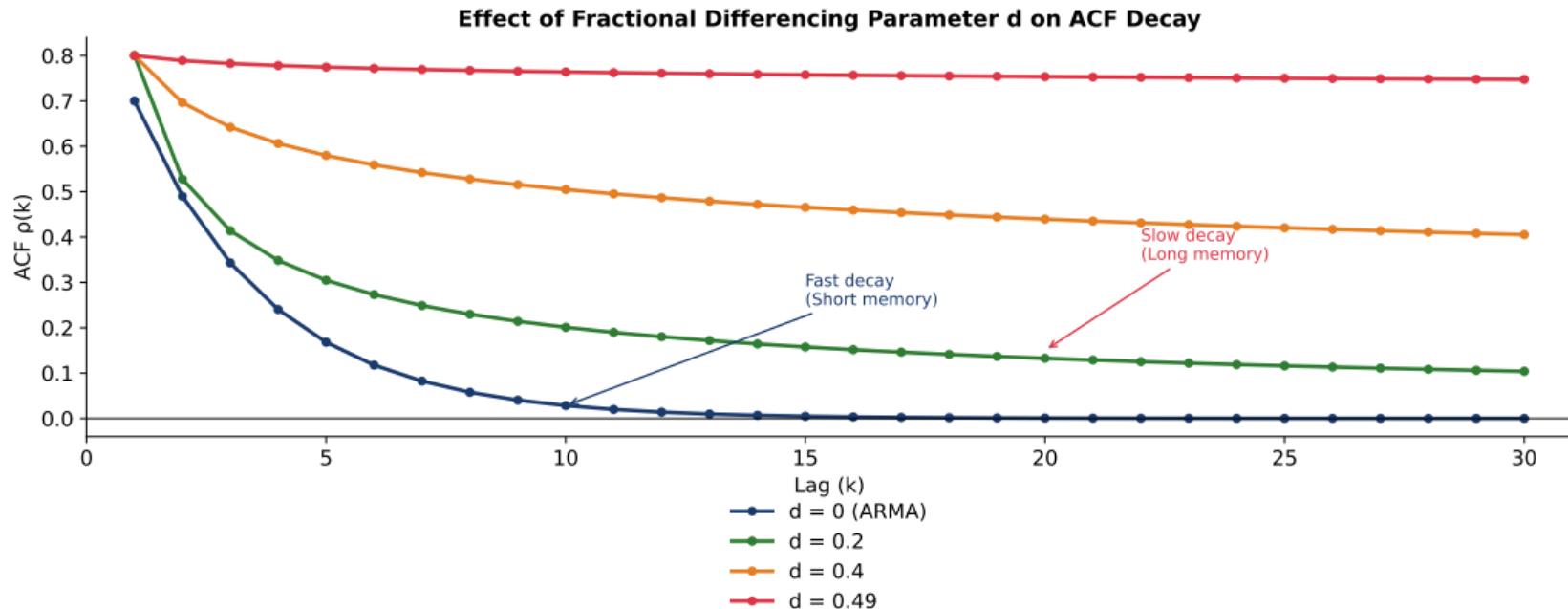
Value $d$	ACF Behavior	Interpretation
$d = 0$	Scădere exponentielle	Memorie scurtă
$0 < d < 0.5$	Scădere hyperbolically	Long memory, stationary
$d = 0.5$	Non-summable ACF	At the limit
$0.5 < d < 1$	Very slow decay	Long memory, nonstationary
$d = 1$	ACF = 1 (constant)	Random walk

## Hurst Parameter $H$

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

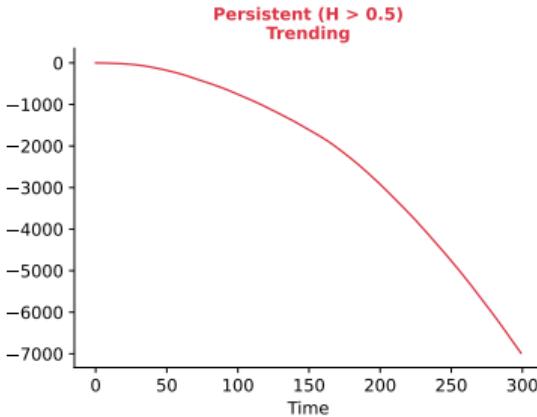
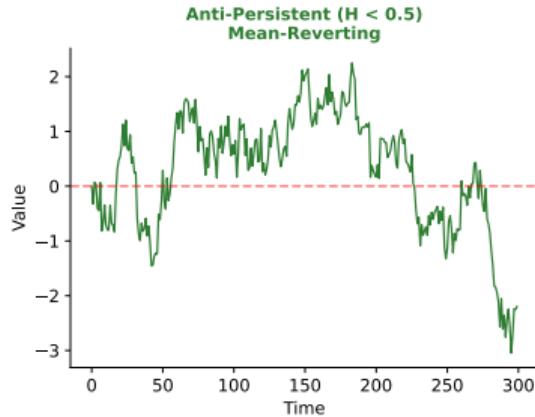
- $H = 0.5$ : Random walk (no memory)
- $H > 0.5$ : Persistence (trend-following)
- $H < 0.5$ : Anti-persistence (mean-reverting)

## Effect of Parameter $d$ on ACF



The higher  $d$ , the slower autocorrelations decay. As  $d \rightarrow 0.5$ , autocorrelations remain significant even at very large lags.

# Exponentul Hurst: Interpretation Vizuală



$H < 0.5$ : Series that frequently returns to mean (mean-reverting)

$H = 0.5$ : Random walk, unpredictable

$H > 0.5$ : Persistent series, trends continue

# Estimating the Parameter $d$

## Estimation Methods

- ① **GPH (Geweke-Porter-Hudak)**: Regression in frequency domain

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

- ② **R/S (Rescaled Range)**: Hurst method

$$\frac{R}{S}(n) \sim c \cdot n^H$$

- ③ **MLE (Maximum Likelihood)**: Full ARFIMA estimation

- ④ **Whittle**: Efficient approximation in frequency domain

În Python: arch package, `statsmodels.tsa.arima.model.ARIMA cu order=(p,d,q)` where  $d$  poate fi fractional.

# ARFIMA Example in Python

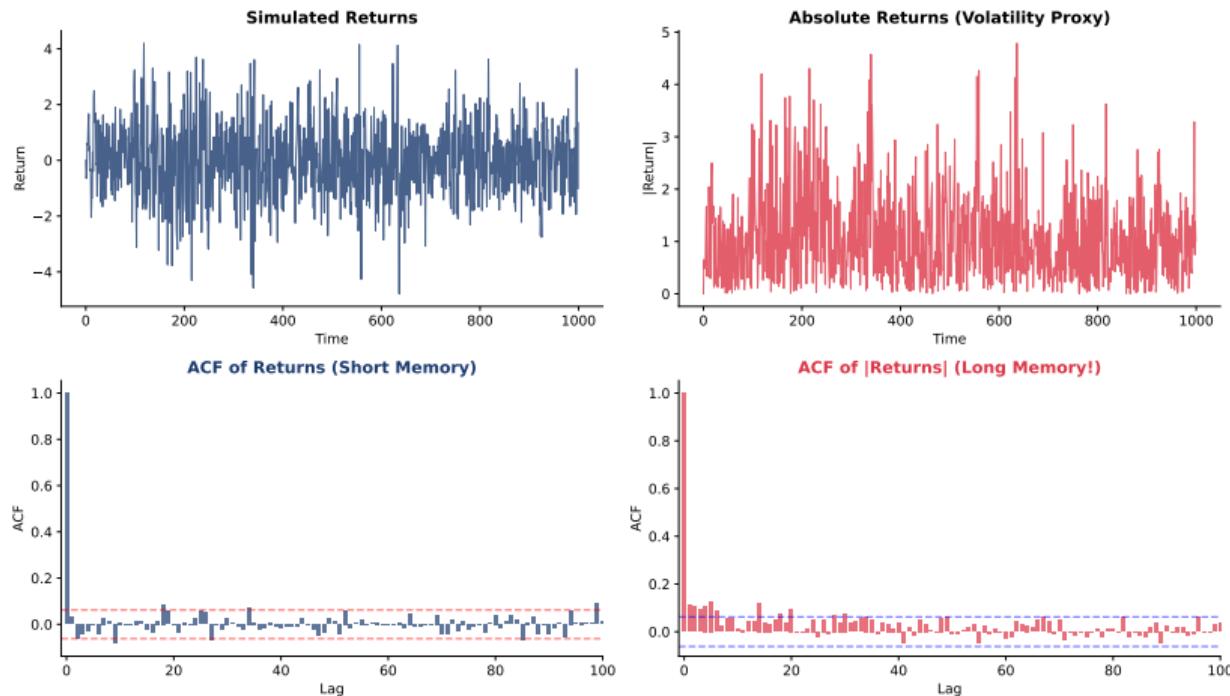
## Python Code

```
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, one often uses arch or fracdiff in Python.

## Exemplu Real: Long Memory în Volatility



**Stylized Fact:** Financial returns have short memory, but volatility ( $|returns|$ ) has long memory! This is the basis for FIGARCH models.

## What is Random Forest?

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

## Avantaje for Serii de Timp

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

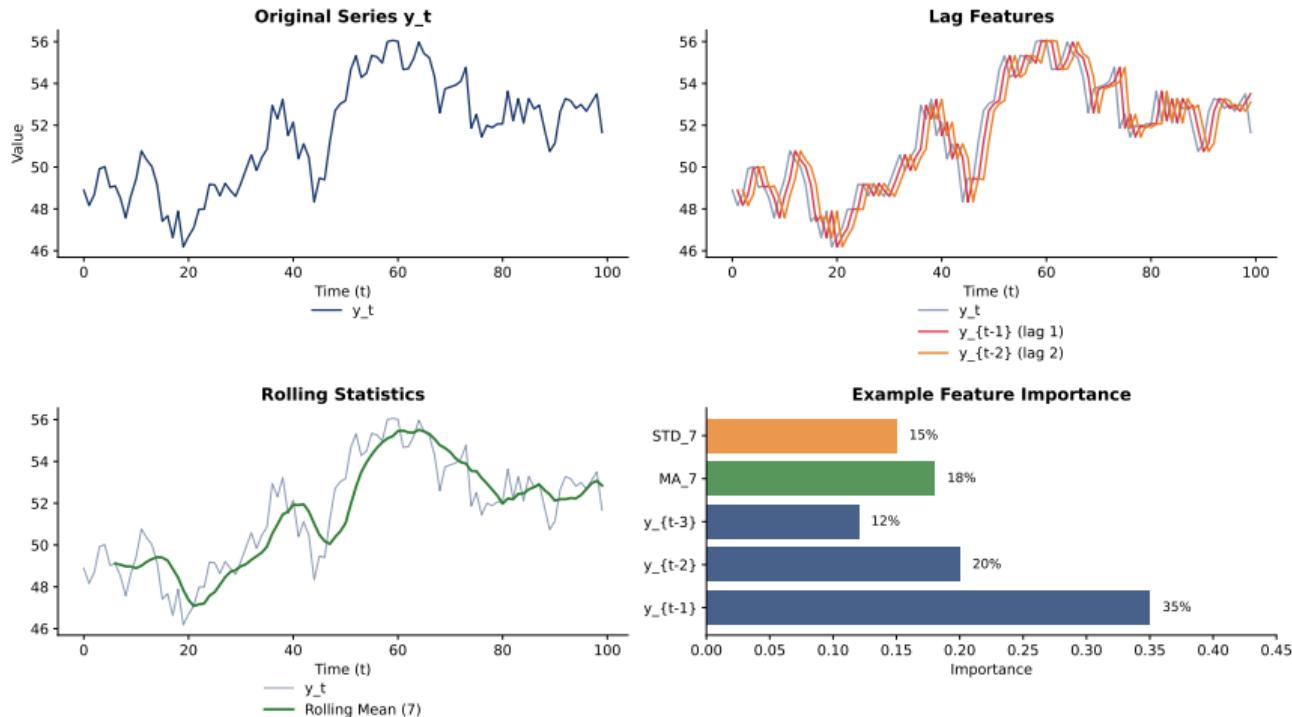
## Feature Engineering for Serii de Timp

- ① **Lag features:**  $Y_{t-1}, Y_{t-2}, \dots, Y_{t-p}$
- 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**, nu randomly!
- Rolling statistics: calculate only on **past data**

# Feature Engineering: Illustration



We transform the time series into features: lags, rolling statistics, and the RF model learns relationships between these and future values.

# 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)
```

# Importanță Features and Interpretation

## Feature Importance

Random Forest provides importance measures:

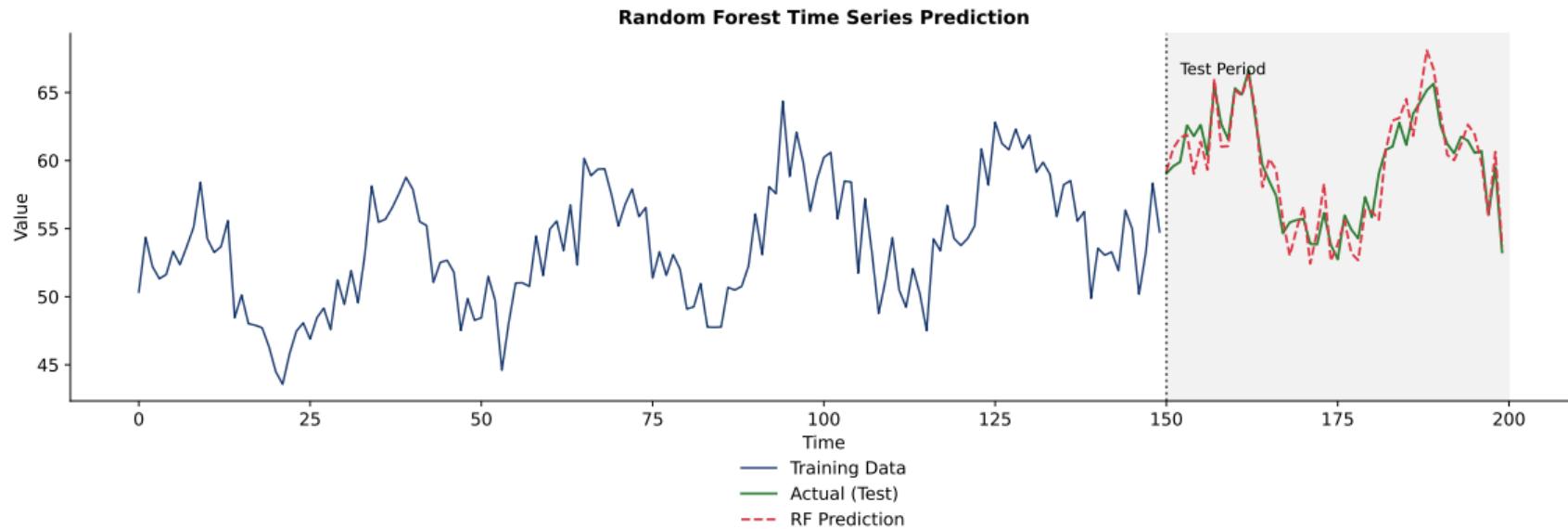
- **Mean Decrease Impurity (MDI)**: Reduction in impurity at each split
- **Permutation Importance**: Cât decaye performanța când feature-ul e permuatat random

## Interpretation Tipică for Serii de Timp

- lag\_1 very important  $\Rightarrow$  Strong autocorrelation
- rolling\_mean important  $\Rightarrow$  Local trend matters
- month important  $\Rightarrow$  Seasonality present

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

## Random Forest: Forecast Example

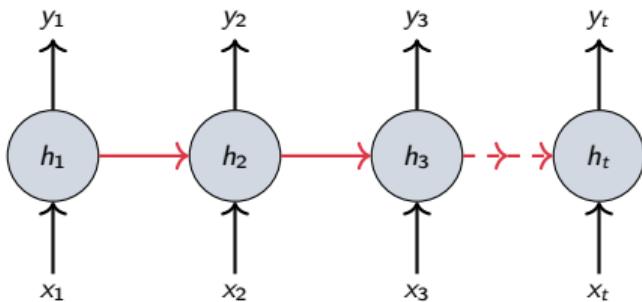


The Random Forest model trained on historical data (blue) produces forecasts (red dotted) that closely follow actual values in the test period (green).

# 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.

## The LSTM Solution

Special cells with **3 gates** that control information flow:

- **Forget Gate ( $f_t$ )**: Ce să forgetm din memoria anterioară
- **Input Gate ( $i_t$ )**: What new information to add
- **Output Gate ( $o_t$ )**: What to send to output

## LSTM Equations

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (\text{Forget})$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (\text{Input})$$

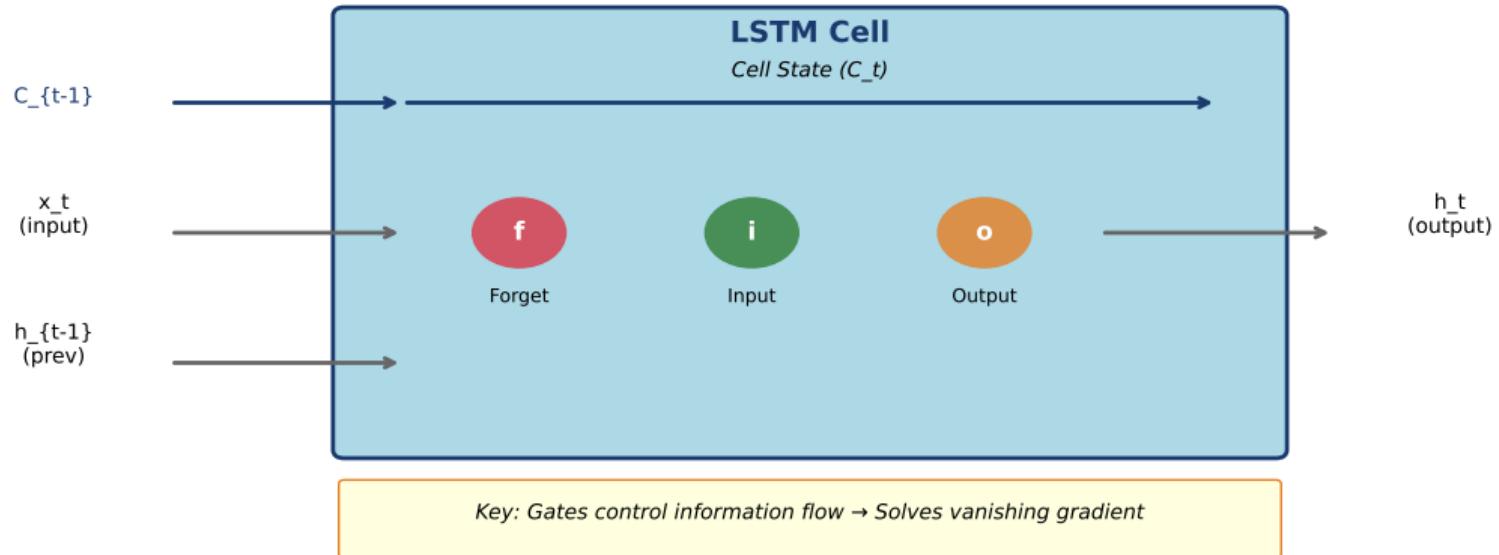
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad (\text{Candidate})$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad (\text{Cell state})$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (\text{Output})$$

$$h_t = o_t \odot \tanh(C_t) \quad (\text{Hidden state})$$

# LSTM Cell Architecture



Gates (forget, input, output) control what information is forgotten, added, and transmitted. **Cell state** allows gradients to “flow” without degradation.

## Why LSTM?

- Captures **long-term dependencies** (spre deosebire de Simple RNNs)
- Learns **complex patterns** and nonlinear
- Handles **sequences de lungimi variabile**
- Works well with **multivariate data**

## Disadvantages

- Requires **lots of data** for training
- **Computationally intensive**
- “**Black box**” - hard to interpret
- Sensitive to **hyperparameters**
- Can **overfit** easily

# LSTM: Implementare in Python cu 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')
```

## Essential Steps

- ① **Normalization/Scaling:** MinMaxScaler or StandardScaler
- ② **Creare sequences:** Sliding window for input
- ③ **Reshape:** 3D format (samples, timesteps, features)
- ④ **Train/Test split:** Temporal, nu randomly!

## Example Creating Sequences

```
def create_sequences(data, n_steps):  
    X, y = [], []  
    for i in range(len(data) - n_steps):  
        X.append(data[i:(i + n_steps)])  
    return np.array(X), np.array(y)  
  
X, y = create_sequences(scaled_data, 10)
```

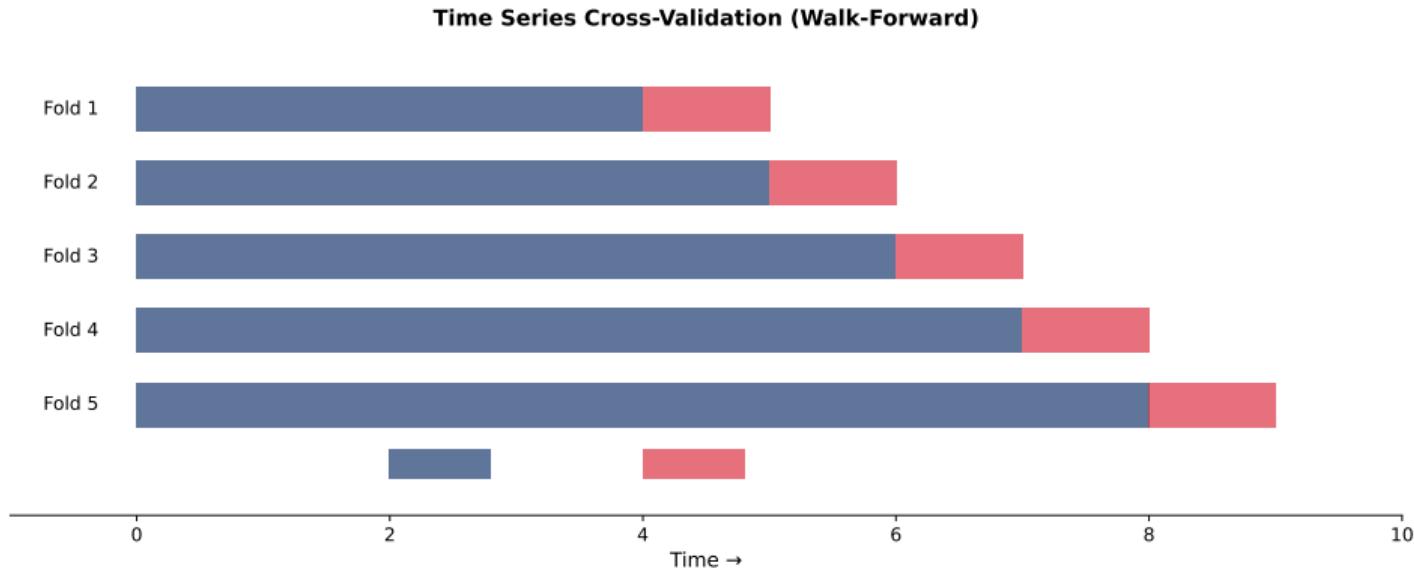
## Common Metrics

- **RMSE:**  $\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$  — Error in original units
- **MAE:**  $\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$  — Robust to outliers
- **MAPE:**  $\frac{100}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|$  — Percentage error
- **MASE:** Compared to naive benchmark

## Validare for Serii de Timp

- **Do not** use standard cross-validation!
- Use **Time Series Cross-Validation** (walk-forward)
- Or **train/validation/test** temporal split

# Time Series Cross-Validation

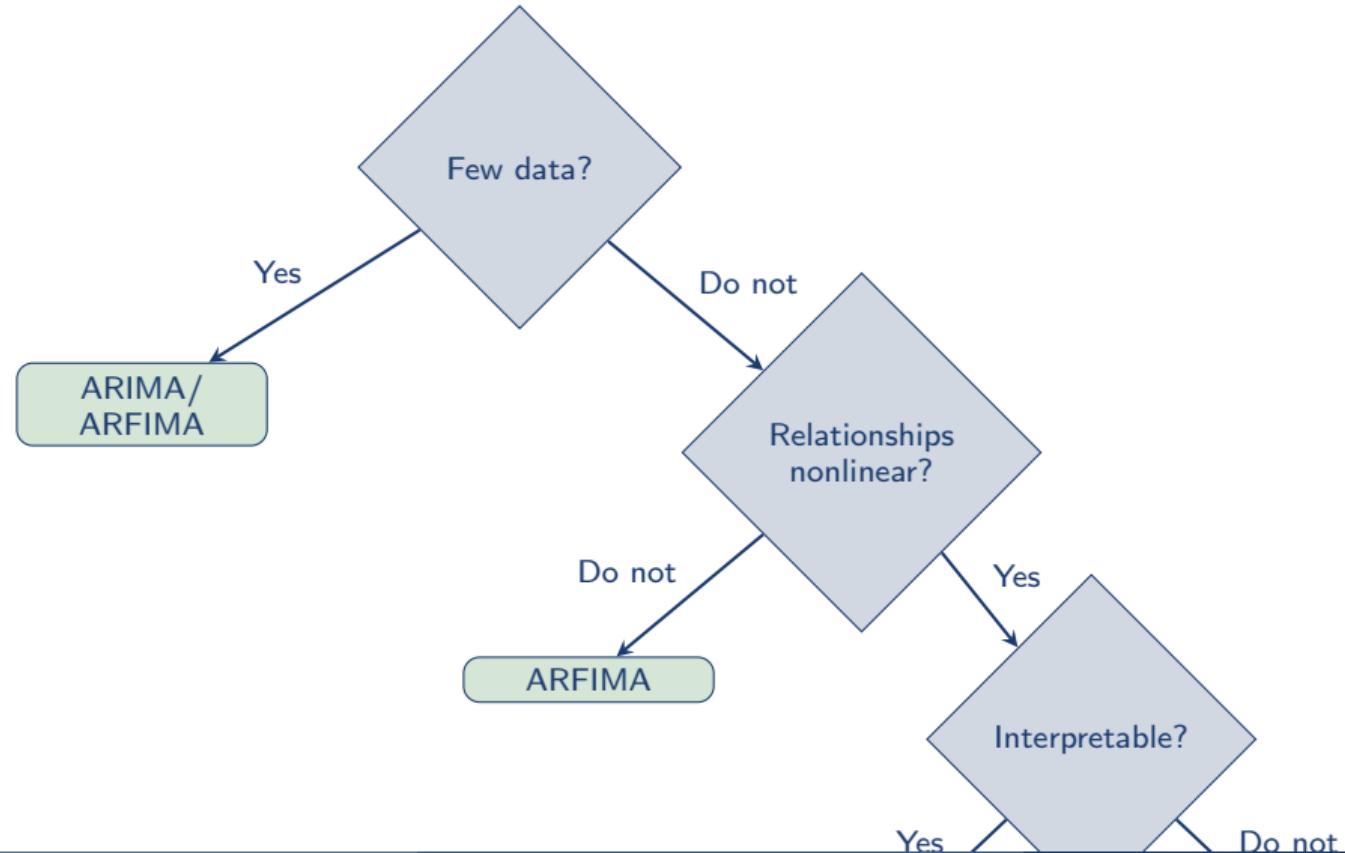


## Python Implementation

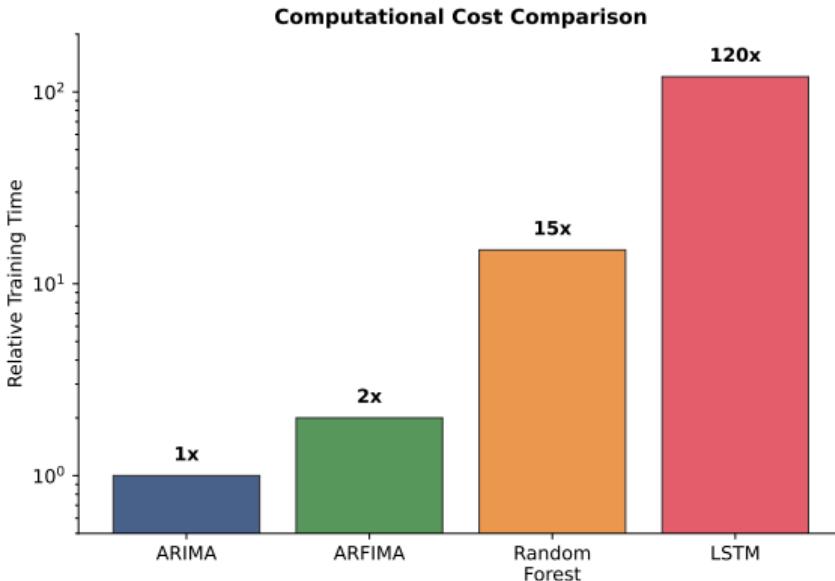
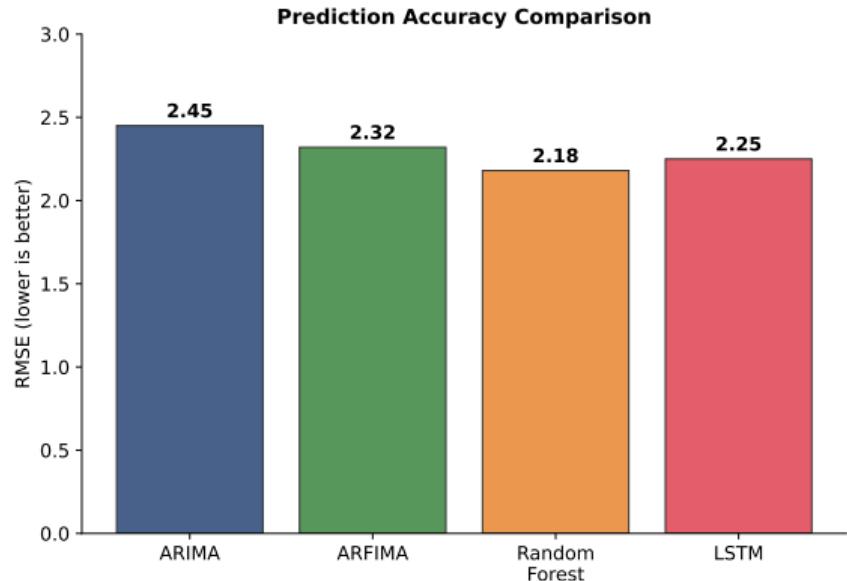
```
from sklearn.model_selection import TimeSeriesSplit  
tscv = TimeSeriesSplit(n_splits=5)
```

Important: Setul de training creste progresiv, și test îl are întotdeauna în viitor. În acest fel, evităm leakajul de date.

# Model Selection Guide



# Model Comparison: Accuracy vs Computational Cost



**Trade-off:** Modelele ML pot avea acuratețe easily mai bună, but computational cost increases significantly. Pentru few data or interpretability, ARIMA/ARFIMA remain excellent choices.

# Case Study: Bitcoin Price Forecasting

## Why Bitcoin?

- Volatility **extreme** and complex patterns
- Potential **long memory** in volatility
- Relationships **nonlinear** with exogenous variables
- Data available at **high frequency**

## Comparative Approach

- ❶ ARIMA on returns
- ❷ ARFIMA for long memory
- ❸ Random Forest with technical features
- ❹ LSTM pe securitate de prețuri

# Case Study: Energy Consumption Forecasting

## Characteristics

- **Multiple seasonality:** daily, weekly, annual
- **Trend of long-term growth**
- **Exogenous variables:** temperature, holiday, price
- **Anomalies:** special events, failures

## Challenges

- Patterns at different time scales
- Interacțiuni complexe between variables
- Need for forecasts at different horizons

## 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

### 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

### Long Memory

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

Hurst:  $d = H - 0.5$

$H > 0.5$ : persistentă

### Metrici Evaluation

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

$$\text{MAPE} = \frac{100}{n} \sum \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

### Random Forest

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

$B$  trees, random features

### Time Series CV

Walk-forward validation

Train → Test (temporal split)

## Why EUR/RON?

- Relevanță for economia românească
- Potential **long memory** (shock persistence)
- Patterns influenced by **macroeconomic factors**
- Este easily accesibile (BNR, Yahoo Finance)

## Objective

We compare ARIMA, ARFIMA, Random Forest and LSTM pe aceleaand date for a înțelege punctele forte ale fiecărei metode.

## Step 1: Loading and Visualizing Data

### Python Code – Descărcare Yeste

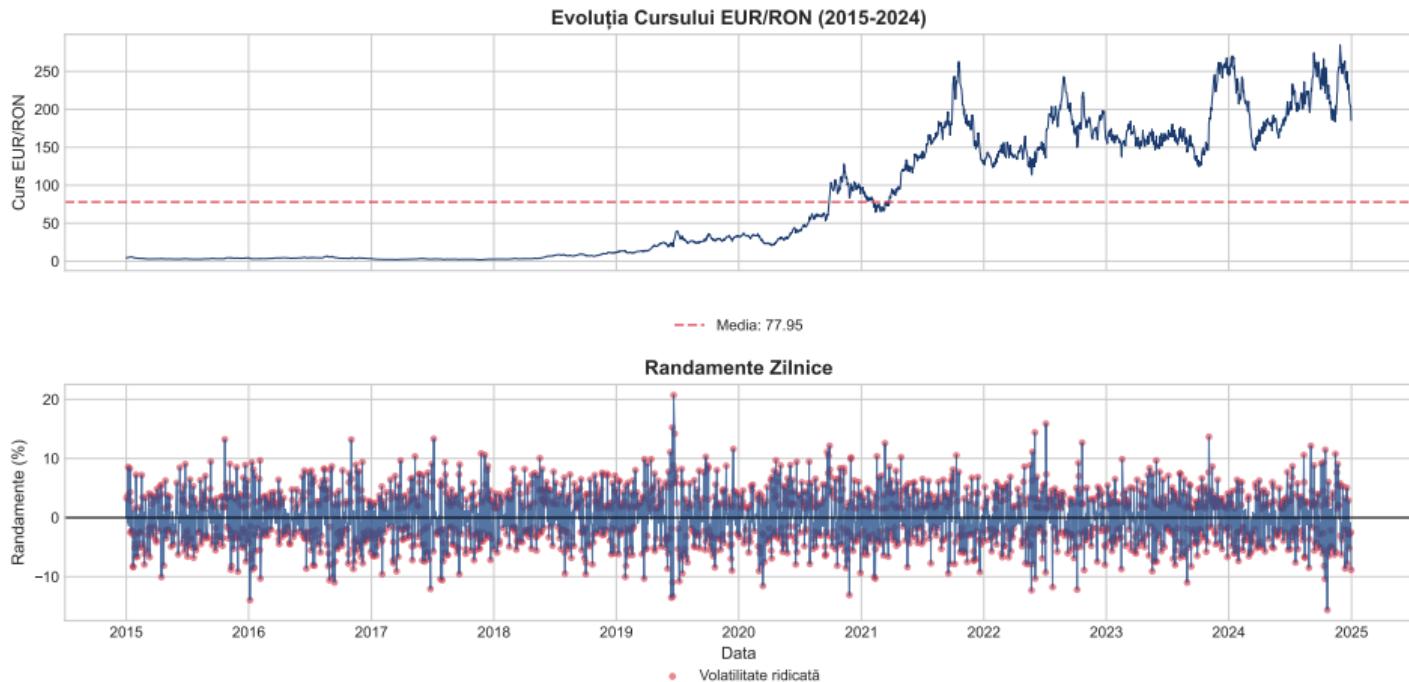
```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

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

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

print(f"Perioada: {df.index[0]} - {df.index[-1]}")
print(f"Observații: {len(df)}")
print(f"Media randamentelor: {df['Returns'].mean():.4f}%")
print(f"Volatility: {df['Returns'].std():.4f}%")
```

# EUR/RON Series Visualization



**Top:** EUR/RON exchange rate – we observe RON depreciation trend and high volatility periods.

**Bottom:** Yesily returns – volatility clustering (high volatility periods are followed by similar periods).

## Step 2: Testing Long Memory

### Python Code – Estimarea lui $d$ and Hurst Test

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

# Testul Phillips-Perron for stationaritate
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"Exponentul Hurst (H): {H:.4f}")
print(f"Parametrul d estimat: {d_estimated:.4f}")

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

## Typical Output

Phillips-Perron p-value: 0.0001 (returns are stationary)

Exponential Hurst ( $H$ ): 0.47

Parametrul  $d$  estimat: -0.03

Series easily ANTI-PERSISTENTĂ (mean-reverting)

## Interpretation

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

## Step 3: ARIMA Model

### Python Code – ARIMA cu selecție automată

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

# Split the 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 randamente
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 cu 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 valoarea estimată anterior

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 stationaritate")
```

## 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)
```

## Pasul 5: Random Forest – Antrenare and Evaluation

### Python Code – Model Random Forest

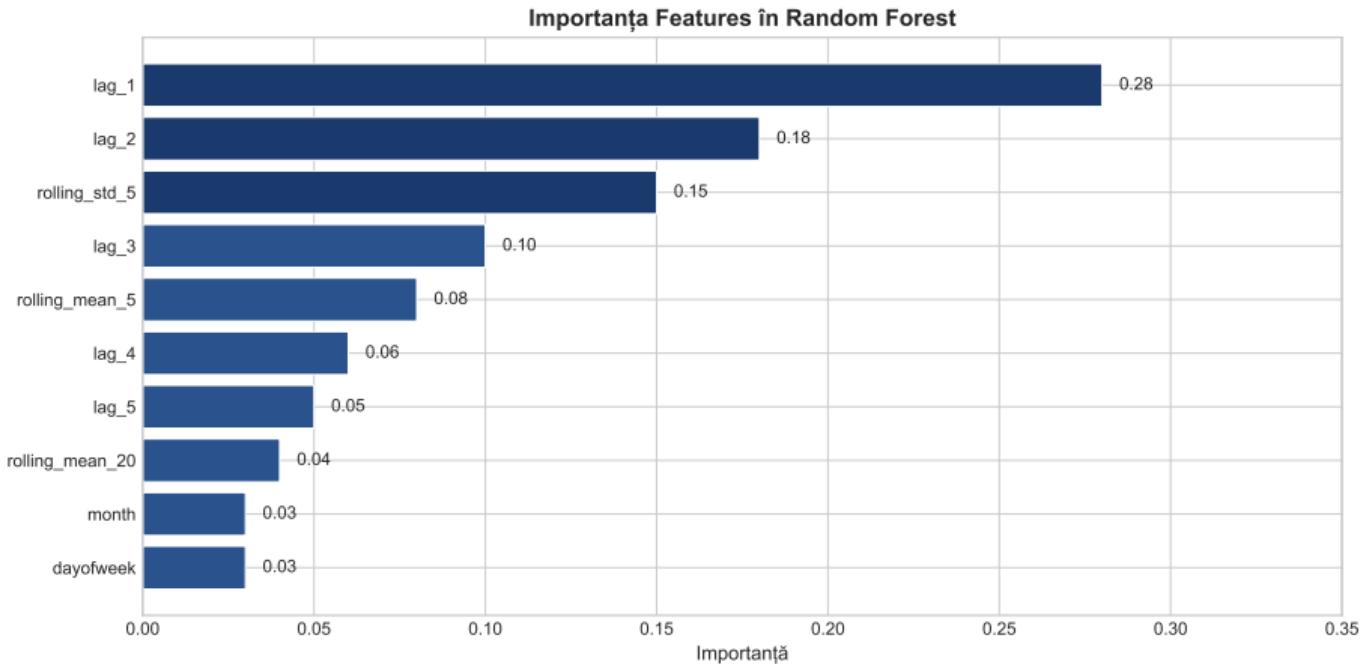
```
# Split the data
X = df_rf.drop('target', axis=1)
y = df_rf['target']

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}")
```

## Random Forest: Feature Importance



**Insight:** Recent lags (lag\_1, lag\_2) and volatility rolling sunt cele mai importante. Features calendaristice au impact minor for randamente zilnice.

## Step 6: LSTM – Data Preparation

### Python Code – Sevente 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:]
```

## Step 6: LSTM – Architecture and Training

### Python Code – Model LSTM

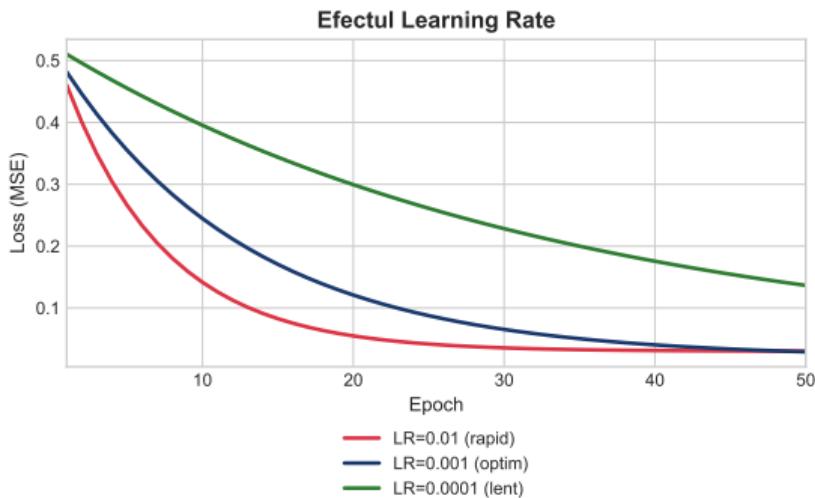
```
# Build the LSTM model
model_lstm = Sequential([
    LSTM(50, return_sequences=True, input_shape=(20, 1)),
    Dropout(0.2),
    LSTM(50, return_sequences=False),
    Dropout(0.2),
    Dense(25),
    Dense(1)
])

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))
```

# LSTM: Learning Curve



**Training Loss:** Scade quickly în primele epoci, apoi se stabilizează.

**Validation Loss:** Follows training loss – nu avem overfit sever.

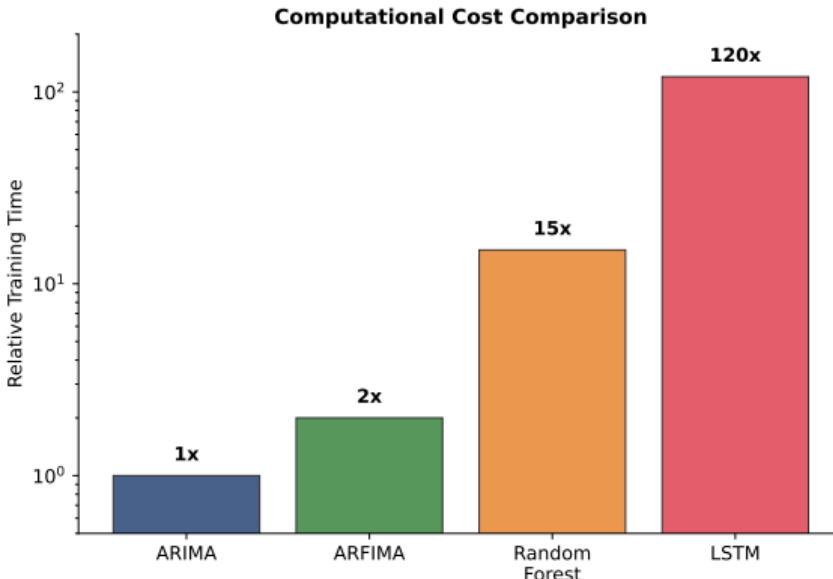
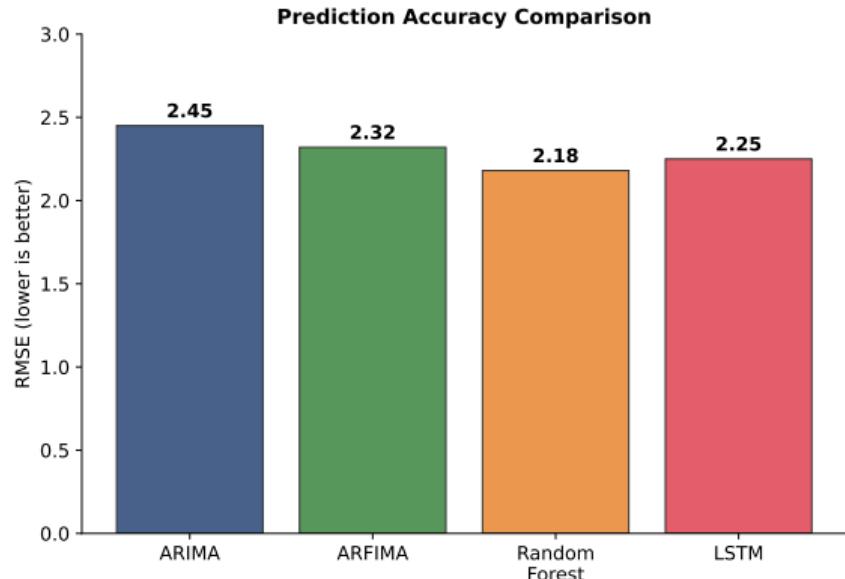
## Comparison: Results on EUR/RON

Model	RMSE	MAE	Time (s)	Interpretable?
ARIMA(1,0,1)	0.412	0.298	0.5	Yes
ARFIMA(1,0,1,1)	0.408	0.295	1.2	Partial
Random Forest	0.395	0.285	3.5	Yes (features)
LSTM	0.401	0.291	45.0	Do not

### Conclusions

- Pentru EUR/RON, differences are **small** – the market is efficient
- Random Forest offers the best trade-off **acuratețe/interpretability**
- LSTM are cost computațional mare for câștig marginal
- ARIMA rămâne o alegere solidă for **baseline**

## Visualization: Predictions vs Actual Values



All models capture the general pattern, but none perfectly predicts volatility peaks. This reflects **market efficiency** and **prediction limits** for serii finanziare.

## When to Choose Each Model?

### ARIMA/ARFIMA

- Few data (< 500 obs.)
- Interpretation importantă
- Long memory suspectată
- Baseline quickly

### LSTM/Deep Learning

- Yeste foarte mari (> 10.000)
- Complex sequences
- Computational resources
- Hidden patterns

### Random Forest

- Many exogenous variables
- Relationships nonlinear
- Feature importance
- Yeste moderate

### Golden Rule

Start simple (ARIMA), adaugă complexitate doar if performanța crește significant!

## Example 2: BET Index (Bucharest Stock Exchange)

### Characteristics

- Volatility clustering strong
- 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 – cel mai bun for volatilitate
- ARFIMA for volatilitate: 1.52
- Random Forest: 1.48
- LSTM: 1.51

## Example 3: Inflation Rate in Romania

### Characteristics

- Series **monthly** (low frequency)
- **Persistence ridicată** – shocks persist
- Influenced by monetary policy
- Potențial strong for **long memory**

### Typical Results

- ARFIMA cu  $d \approx 0.35$  – captures persistence
- ARIMA subestimează 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
Yeste necesare	Few	Few	Medium	Many
Long memory	Do not	Yes	Partial	Partial
Nonlinearity	Do not	Do not	Yes	Yes
Interpretabil	Yes	Yes	Partial	Do not
Computation time	Fast	Fast	Medium	Slow
Exog. var.	Limited	Limited	Yes	Yes

### Recommended Workflow

- ① Start cu **ARIMA** as baseline
- ② Test **long memory** → ARFIMA if  $d$  significant
- ③ Add **features** → Random Forest
- ④ Doar cu date multe and resurse → LSTM

## What we learned

- **ARFIMA:** Extinde ARIMA for long memory ( $d$  fractional)
- **Random Forest:** Ensemble of trees, nonlinear relationships, interpretable
- **LSTM:** Deep learning for sequences, dependențe complexe
- **Trade-offs:** Complexitate vs interpretability vs date necesare

## Practical Recommendations

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

## Quiz Fast

- ➊ What does  $d = 0.3$  mean in an ARFIMA model?
- ➋ Why use Time Series CV instead of standard k-fold?
- ➌ Care este avantajul principal al LSTM față de Simple RNNs?
- ➍ What type of model would you choose with few data and linear relationships?
- ➎ What does “data leakage” în contextul ML for serii de timp?

## Quiz Answers

- ①  $d = 0.3$ : Long memory, 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 problem “vanishing gradient” through the gating mechanism, allowing learning of long-term dependencies.
- ④ **Few 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 which mixes temporal order.

## Extensii and Subiecte Avansate

- **Transformer** for serii de timp (Temporal Fusion Transformer)
- **Prophet** (Facebook/Meta) for sezonalitate
- **Neural Prophet:** Prophet + neural networks
- **Ensemble methods:** Combinarea mai multor models
- **Anomaly detection** with ML

Questions?