Cours Machine Learning

Chapitre 12

Séries Temporelles et Forecasting

Objectifs d'apprentissage :

— Comprendre les composantes et propriétés des séries temporelles
— Maîtriser les modèles classiques (ARIMA, SARIMA, Prophet)
— Appliquer le Deep Learning au forecasting (LSTM, GRU, Transformers)
— Évaluer et valider les prédictions temporelles
— Implémenter des pipelines de prédiction robustes

Prérequis : Chapitres 06 (Réseaux de Neurones), 08 (RNN/LSTM)

Durée estimée : 6-8 heures

Notebooks : `12_demo_*.ipynb`, `12_exercices.ipynb`

Cours ML - Sandbox-ML
Version 1.0 - 2026

Table des matières

1 Motivation

Les séries temporelles sont omniprésentes dans notre monde : cours boursiers, météorologie, trafic réseau, consommation d'énergie, ventes de produits, signaux biologiques, etc. La capacité à analyser et prédire ces données séquentielles est cruciale pour de nombreuses applications industrielles et scientifiques.

Exemple : Problèmes de forecasting réels

- **Finance :** Prédire le prix d'une action dans les prochains jours
- **Énergie :** Anticiper la demande électrique pour optimiser la production
- **E-commerce :** Prévoir les ventes pour gérer les stocks
- **Santé :** Monitorer les signes vitaux et détecter les anomalies
- **Météo :** Prédire la température et les précipitations

Contrairement aux problèmes de ML classiques où les observations sont supposées i.i.d. (indépendantes et identiquement distribuées), les séries temporelles présentent une **dépendance temporelle** : la valeur à l'instant t dépend des valeurs passées. Cette propriété fondamentale nécessite des approches spécifiques.

1.1 Différences avec le ML classique

Table 1 – ML Classique vs Séries Temporelles

Aspect	ML Classique	Séries Temporelles
Structure	Observations i.i.d.	Dépendance temporelle
Validation	Cross-validation	Time series split
Features	Données tabulaires	Séquences temporelles
Objectif	Classification/Régression	Forecasting/Détection
Train/Test Split	Aléatoire	Chronologique

2 Fondements Théoriques

2.1 Définition et Notation

Définition : Série Temporelle

Une série temporelle est une séquence d'observations $\{y_t\}$ indexées par le temps t, où $t \in \{1, 2, \ldots, T\}$ pour des données discrètes.

$$\{y_1, y_2, y_3, \ldots, y_T\}$$

Notation :

- y_t : valeur observée au temps t
- $\hat{y}_{t+h}$: prédiction à l'horizon h (forecast)
- T : longueur de la série (nombre d'observations)
- h : horizon de prédiction (forecast horizon)

2.2 Composantes d'une Série Temporelle

Une série temporelle peut généralement être décomposée en plusieurs composantes :

Définition : Décomposition Additive

$$y_t = T_t + S_t + R_t$$

où :

- T_t : **Tendance (Trend)** - mouvement à long terme (croissance, décroissance)
- S_t : **Saisonnalité (Seasonality)** - motifs périodiques répétitifs
- R_t : **Résidu (Residual)** - fluctuations aléatoires (bruit)

Définition : Décomposition Multiplicative

$$y_t = T_t \times S_t \times R_t$$

Utilisée quand la variance augmente avec le niveau de la série (croissance exponentielle).

Exemple : Ventes mensuelles d'un produit

- **Tendance :** Croissance annuelle de 10% (expansion du marché)
- **Saisonnalité :** Pic en décembre (fêtes de fin d'année)
- **Résidu :** Variations aléatoires dues à la météo, promotions

2.3 Stationnarité

La stationnarité est une propriété fondamentale qui simplifie grandement l'analyse et la modélisation.

Définition : Stationnarité Forte

Une série $\{y_t\}$ est strictement stationnaire si sa distribution jointe est invariante par translation temporelle :

$$P(y_{t_1}, y_{t_2}, \ldots, y_{t_k}) = P(y_{t_1+h}, y_{t_2+h}, \ldots, y_{t_k+h})$$

pour tout h et tous indices $t_1, \ldots, t_k$.

Définition : Stationnarité Faible (du Second Ordre)

Une série $\{y_t\}$ est faiblement stationnaire si :

1. $\mathbb{E}[y_t] = \mu$ est constante (moyenne constante)
2. $\text{Var}(y_t) = \sigma^2$ est constante (variance constante)
3. $\text{Cov}(y_t, y_{t+h}) = \gamma(h)$ ne dépend que de h (autocovariance stationnaire)

Attention

La plupart des modèles classiques (ARIMA) supposent la stationnarité. Il faut donc transformer les séries non-stationnaires avant de les modéliser.

2.3.1 Techniques de Stationnarisation

1. **Différenciation :** $\nabla y_t = y_t - y_{t-1}$ (élimine la tendance)
2. **Transformation log :** $\log(y_t)$ (stabilise la variance)
3. **Différence saisonnière :** $\nabla_s y_t = y_t - y_{t-s}$ avec $s =$ période saisonnière
4. **Détrending :** Soustraire une tendance estimée (linéaire, polynomiale)

2.4 Autocorrélation et Autocorrélation Partielle

Définition : Fonction d'Autocorrélation (ACF)

Mesure la corrélation linéaire entre y_t et y_{t-k} (lag k) :

$$\rho(k) = \frac{\text{Cov}(y_t, y_{t-k})}{\sqrt{\text{Var}(y_t)\text{Var}(y_{t-k})}} = \frac{\gamma(k)}{\gamma(0)}$$

où $\gamma(k) = \mathbb{E}[(y_t - \mu)(y_{t-k} - \mu)]$ est l'autocovariance au lag k.

Définition : Fonction d'Autocorrélation Partielle (PACF)

Mesure la corrélation entre y_t et y_{t-k} après avoir éliminé l'influence des lags intermédiaires $1, 2, \ldots, k-1$.
Utilisée pour identifier l'ordre p d'un modèle AR(p).

Astuce

Interprétation ACF/PACF :

- ACF décroît lentement : série non-stationnaire (tendance)
- Pic significatif à lag s : saisonnalité de période s
- PACF coupe après lag p : modèle AR(p) approprié
- ACF coupe après lag q : modèle MA(q) approprié

2.5 Tests de Stationnarité

Définition : Test de Dickey-Fuller Augmenté (ADF)

Test statistique pour détecter la présence d'une racine unitaire (non-stationnarité).
Hypothèses :

- H_0 : La série possède une racine unitaire (non-stationnaire)
- H_1 : La série est stationnaire

Décision :

- Si p-value < 0.05 : rejeter $H_0 \Rightarrow$ série stationnaire
- Si p-value ≥ 0.05 : ne pas rejeter $H_0 \Rightarrow$ série non-stationnaire

```python
from statsmodels.tsa.stattools import adfuller

# Test de stationnarite
result = adfuller(timeseries)
adf_statistic = result[0]
p_value = result[1]

print(f"ADF Statistic: {adf_statistic:.4f}")
print(f"p-value: {p_value:.4f}")

if p_value < 0.05:
    print("Serie stationnaire (rejeter H0)")
else:
    print("Serie non-stationnaire (ne pas rejeter H0)")
```

Listing 1 – Test ADF avec statsmodels

3 Modèles Classiques de Séries Temporelles

3.1 Modèle AutoRégressif AR(p)

Définition : Modèle AR(p)

Un processus autorégressif d'ordre p exprime y_t comme une combinaison linéaire des p valeurs passées plus un bruit blanc :

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

où :

- c : constante (intercept)
- $\phi_1, \ldots, \phi_p$: coefficients autorégressifs
- $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$: bruit blanc (innovations)

Exemple : AR(1) - Modèle autorégressif d'ordre 1

$$y_t = c + \phi_1 y_{t-1} + \epsilon_t$$

Si $|\phi_1| < 1$: processus stationnaire convergeant vers la moyenne $\mu = \frac{c}{1-\phi_1}$

Identification de l'ordre p :

- PACF coupe après lag p
- ACF décroît exponentiellement ou avec oscillations amorties

3.2 Modèle Moyenne Mobile MA(q)

Définition : Modèle MA(q)

Un processus moyenne mobile d'ordre q exprime y_t comme une combinaison linéaire des q innovations passées :

$$y_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}$$

où :

- μ : moyenne du processus
- $\theta_1, \ldots, \theta_q$: coefficients MA
- $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$: bruit blanc

Identification de l'ordre q :

- ACF coupe après lag q
- PACF décroît exponentiellement

3.3 Modèle ARMA(p,q)

Définition : Modèle ARMA(p

Combine les composantes AR et MA :

$$y_t = c + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q}$$

Plus flexible que AR ou MA seuls, capture à la fois autocorrélations et moyennes mobiles.

3.4 Modèle ARIMA(p,d,q)

Définition : Modèle ARIMA(p

ARIMA = AutoRegressive Integrated Moving Average

Paramètres :

- p : ordre autorégressif (nombre de lags y_{t-i})
- d : ordre de différenciation (nombre de différences pour stationnariser)
- q : ordre moyenne mobile (nombre de lags ϵ_{t-i})

Formulation :

$$\nabla^d y_t = c + \phi_1 \nabla^d y_{t-1} + \cdots + \phi_p \nabla^d y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q}$$

où ∇^d représente d différenciations successives.

Exemple : ARIMA(1

- $d = 1$: une différenciation $\nabla y_t = y_t - y_{t-1}$
- $p = 1$: un terme autorégressif $\phi_1 \nabla y_{t-1}$
- $q = 1$: un terme moyenne mobile $\theta_1 \epsilon_{t-1}$

Équation : $\nabla y_t = c + \phi_1 \nabla y_{t-1} + \epsilon_t + \theta_1 \epsilon_{t-1}$

3.4.1 Sélection des paramètres (p,d,q)

1. **Ordre d :** Différencier jusqu'à obtenir la stationnarité (test ADF)
2. **Ordre p :** Analyser PACF de la série différenciée
3. **Ordre q :** Analyser ACF de la série différenciée
4. **Critères d'information :** Comparer AIC/BIC pour différents (p, q)

Définition : Critères AIC et BIC

AIC (Akaike Information Criterion) :

$$\text{AIC} = 2k - 2\ln(\hat{L})$$

BIC (Bayesian Information Criterion) :

$$\text{BIC} = k\ln(n) - 2\ln(\hat{L})$$

où k = nombre de paramètres, n = taille échantillon, $\hat{L}$ = vraisemblance maximale.
Objectif : Minimiser AIC ou BIC (pénalise la complexité du modèle).

3.5 Modèle SARIMA(p,d,q)(P,D,Q)s

Définition : Modèle SARIMA - Seasonal ARIMA

Extension d'ARIMA pour capturer la saisonnalité périodique de période s.
Paramètres non-saisonniers : (p, d, q)
Paramètres saisonniers : $(P, D, Q)_s$

- P : ordre AR saisonnier (lags $y_{t-s}, y_{t-2s}, \ldots$)
- D : ordre de différenciation saisonnière
- Q : ordre MA saisonnier
- s : période saisonnière (12 pour mensuel, 7 pour quotidien hebdomadaire)

Exemple : SARIMA(1

- Tendance : ARIMA(1,1,1)
- Saisonnalité annuelle : période $s = 12$ (12 mois)
- Composante saisonnière : AR(1) MA(1) avec différenciation saisonnière

Capture à la fois les corrélations à court terme et les patterns annuels.

```python
from statsmodels.tsa.arima.model import ARIMA

# Ajuster ARIMA(1,1,1)
model = ARIMA(timeseries, order=(1, 1, 1))
fitted_model = model.fit()

# Resume du modele
```

```
print(fitted_model.summary())

# Predictions
forecast = fitted_model.forecast(steps=10)
print(f"Predictions 10 prochains pas: {forecast}")
```

Listing 2 – ARIMA avec statsmodels

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

# SARIMA(1,1,1)(1,1,1,12) pour donnees mensuelles
model = SARIMAX(timeseries,
                order=(1, 1, 1),
                seasonal_order=(1, 1, 1, 12))
fitted_model = model.fit()

# Predictions
forecast = fitted_model.forecast(steps=24)  # 24 mois
```

Listing 3 – SARIMA avec statsmodels

3.6 Prophet - Modèle de Facebook

Définition : Prophet

Modèle de forecasting développé par Facebook (Meta) pour des séries temporelles avec :

- Tendances fortes (croissance linéaire ou logistique)
- Saisonnalités multiples (annuelle, hebdomadaire, journalière)
- Jours fériés et événements spéciaux
- Observations manquantes et outliers

Décomposition additive :

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

où :

- $g(t)$: fonction de tendance (linéaire ou logistique avec changepoints)
- $s(t)$: saisonnalité périodique (séries de Fourier)
- $h(t)$: effets des jours fériés
- ϵ_t : erreur résiduelle

Avantages de Prophet :

- Interface simple (peu de tuning)
- Robuste aux données manquantes et outliers
- Détection automatique des changepoints (ruptures de tendance)
- Saisonnalités multiples (annuelle, hebdomadaire, custom)
- Gestion native des jours fériés (calendriers prédéfinis)
- Intervalles de confiance automatiques

```
from prophet import Prophet
import pandas as pd
```

```
3
4  # Donnees au format Prophet (colonnes 'ds' et 'y')
5  df = pd.DataFrame({'ds': dates, 'y': values})
6
7  # Creer et ajuster le modele
8  model = Prophet(yearly_seasonality=True,
9                  weekly_seasonality=True,
10                 daily_seasonality=False)
11 model.fit(df)
12
13 # Predictions sur 365 jours
14 future = model.make_future_dataframe(periods=365)
15 forecast = model.predict(future)
16
17 # Visualisation
18 model.plot(forecast)
19 model.plot_components(forecast)  # tendance + saisonnalites
```

Listing 4 – Prophet de Facebook

Attention

Prophet est excellent pour des séries avec saisonnalités claires et tendances changeantes, mais peut être moins performant que ARIMA sur des séries purement stationnaires ou avec des patterns complexes non-périodiques.

4 Deep Learning pour Séries Temporelles

Les réseaux de neurones récurrents (RNN) et leurs variantes (LSTM, GRU) sont particulièrement adaptés aux données séquentielles grâce à leur capacité à modéliser les dépendances temporelles à long terme.

4.1 LSTM pour Forecasting

Définition : LSTM pour Séries Temporelles

Un réseau LSTM (Long Short-Term Memory) peut apprendre des dépendances temporelles complexes grâce à sa cellule mémoire et ses portes de régulation.

Architecture typique :

1. **Input :** Séquence de longueur L : $(y_{t-L+1}, \ldots, y_{t-1}, y_t)$
2. **LSTM layers :** 1-3 couches avec hidden size h
3. **Output :** Prédiction $\hat{y}_{t+1}$ ou $(\hat{y}_{t+1}, \ldots, \hat{y}_{t+h})$

4.1.1 Préparation des Données - Windowing

Pour utiliser un LSTM, il faut transformer la série temporelle en séquences (windows) :

Exemple : Création de fenêtres glissantes (sliding windows)

Série : $[y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8]$
Avec window size $L = 3$, horizon $h = 1$:

$$\begin{aligned} X_1 &= [y_1, y_2, y_3], \quad Y_1 = y_4 \\ X_2 &= [y_2, y_3, y_4], \quad Y_2 = y_5 \\ X_3 &= [y_3, y_4, y_5], \quad Y_3 = y_6 \\ &\vdots \end{aligned}$$

Dataset final : $\{(X_i, Y_i)\}_{i=1}^{N}$

```python
import numpy as np

def create_windows(data, window_size, horizon=1):
    X, y = [], []
    for i in range(len(data) - window_size - horizon + 1):
        X.append(data[i:i+window_size])
        y.append(data[i+window_size:i+window_size+horizon])
    return np.array(X), np.array(y)

# Exemple
timeseries = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
X, y = create_windows(timeseries, window_size=3, horizon=1)

print(f"X shape: {X.shape}")  # (7, 3)
print(f"y shape: {y.shape}")  # (7, 1)
print(f"Premier exemple: X={X[0]} -> y={y[0]}")
```

Listing 5 – Création de fenêtres glissantes

4.1.2 Architecture LSTM pour Forecasting

```python
import torch
import torch.nn as nn

class LSTMForecaster(nn.Module):
    def __init__(self, input_size=1, hidden_size=64,
                 num_layers=2, output_size=1, dropout=0.2):
        super().__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        # LSTM layers
        self.lstm = nn.LSTM(
            input_size=input_size,
            hidden_size=hidden_size,
            num_layers=num_layers,
            batch_first=True,
```

```
            dropout=dropout if num_layers > 1 else 0
        )

        # Fully connected output
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        # x: (batch, seq_len, input_size)
        lstm_out, (h_n, c_n) = self.lstm(x)

        # Prendre le dernier output
        last_output = lstm_out[:, -1, :]  # (batch, hidden_size)

        # Prediction
        output = self.fc(last_output)  # (batch, output_size)
        return output

# Instanciation
model = LSTMForecaster(
    input_size=1,       # features univariees
    hidden_size=64,     # taille hidden state
    num_layers=2,       # 2 couches LSTM
    output_size=1,      # prediction 1 pas
    dropout=0.2
)

print(model)
```

Listing 6 – LSTM avec PyTorch

4.1.3 Entraînement

```
import torch.optim as optim

# Hyperparametres
learning_rate = 0.001
num_epochs = 100
batch_size = 32

# Loss et optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# DataLoader
from torch.utils.data import TensorDataset, DataLoader

X_train = torch.FloatTensor(X_train).unsqueeze(-1)  # (N, L, 1)
y_train = torch.FloatTensor(y_train)

dataset = TensorDataset(X_train, y_train)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
```

```
20
21 # Boucle d'entrainement
22 model.train()
23 for epoch in range(num_epochs):
24     epoch_loss = 0.0
25     for batch_X, batch_y in dataloader:
26         # Forward
27         predictions = model(batch_X)
28         loss = criterion(predictions.squeeze(), batch_y)
29
30         # Backward
31         optimizer.zero_grad()
32         loss.backward()
33         optimizer.step()
34
35         epoch_loss += loss.item()
36
37     if (epoch + 1) % 10 == 0:
38         print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss/len(
    dataloader):.4f}")
```

Listing 7 – Entraînement LSTM

4.2 GRU pour Forecasting

Définition : GRU - Gated Recurrent Unit

Variante simplifiée de LSTM avec seulement 2 portes (reset et update) au lieu de 3.

Avantages :

- Moins de paramètres que LSTM (entraînement plus rapide)
- Performances souvent comparables à LSTM
- Moins susceptible au surapprentissage

Remplacer simplement `nn.LSTM` par `nn.GRU` dans le code précédent.

4.3 Séries Temporelles Multivariées

Définition : Forecasting Multivarié

Prédire y_t en utilisant plusieurs features $\mathbf{x}_t = [x_t^{(1)}, x_t^{(2)}, \ldots, x_t^{(d)}]^T$.

Exemple : Prédire la consommation électrique avec :

- $x_t^{(1)}$: température
- $x_t^{(2)}$: jour de la semaine (encodé)
- $x_t^{(3)}$: consommation passée

```
1 # input_size = nombre de features
2 model = LSTMForecaster(
3     input_size=5,       # 5 features en entree
4     hidden_size=128,
5     num_layers=2,
```

```
    output_size=1       # 1 valeur predite (univarie output)
)

# X shape: (batch, seq_len, 5)
# y shape: (batch, 1)
```

Listing 8 – LSTM multivarié

4.4 Attention pour Séries Temporelles

Le mécanisme d'attention permet au modèle de se concentrer sur les instants les plus pertinents pour la prédiction.

Définition : Temporal Attention

Au lieu de n'utiliser que le dernier hidden state, utiliser une combinaison pondérée de tous les hidden states :

$$\mathbf{c} = \sum_{t=1}^{L} \alpha_t \mathbf{h}_t$$

où α_t sont des poids d'attention calculés dynamiquement :

$$\alpha_t = \frac{\exp(e_t)}{\sum_{i=1}^{L} \exp(e_i)}, \quad e_t = \text{score}(\mathbf{h}_t, \mathbf{h}_L)$$

```
class LSTMWithAttention(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.attention = nn.Linear(hidden_size, 1)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        # LSTM
        lstm_out, _ = self.lstm(x)  # (batch, seq_len, hidden)

        # Attention scores
        scores = self.attention(lstm_out)  # (batch, seq_len, 1)
        attention_weights = torch.softmax(scores, dim=1)

        # Context vector (weighted sum)
        context = torch.sum(attention_weights * lstm_out, dim=1)  # (batch, hidden)

        # Prediction
        output = self.fc(context)
        return output
```

Listing 9 – LSTM avec Attention

4.5 Transformers pour Séries Temporelles

Les Transformers, initialement développés pour le NLP, sont de plus en plus utilisés pour les séries temporelles grâce au mécanisme d'attention multi-têtes.

Définition : Transformer pour Forecasting

Architecture basée sur :

- **Positional Encoding :** Ajouter l'information temporelle
- **Multi-Head Self-Attention :** Capturer les relations entre tous les instants
- **Feed-Forward Networks :** Transformations non-linéaires

Avantages :

- Capture des dépendances à très long terme
- Parallélisation (pas de récurrence séquentielle)
- Interprétabilité via les poids d'attention

Exemple : Modèles Transformer pour Time Series

- **Temporal Fusion Transformer (TFT)** : Google Research
- **Autoformer** : Décomposition auto-corrélation
- **Informer** : Attention sparse pour longues séquences
- **PatchTST** : Découpage en patches comme en vision

5 Métriques d'Évaluation

5.1 Métriques de Régression Classiques

Définition : Mean Absolute Error (MAE)

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Erreur absolue moyenne, interprétable dans l'unité des données.

Définition : Mean Squared Error (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Pénalise fortement les grandes erreurs (carré).

Définition : Root Mean Squared Error (RMSE)

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

Erreur quadratique moyenne, même unité que les données.

5.2 Métriques Spécifiques aux Séries Temporelles

Définition : Mean Absolute Percentage Error (MAPE)

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

Erreur en pourcentage, utile pour comparer des séries d'échelles différentes.

Attention : Indéfini si $y_i = 0$.

Définition : Symmetric MAPE (sMAPE)

$$\text{sMAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2}$$

Variante symétrique de MAPE, bornée entre 0% et 200%.

Définition : Mean Absolute Scaled Error (MASE)

$$\text{MASE} = \frac{\text{MAE}}{\text{MAE}_{\text{naive}}}$$

où $\text{MAE}_{\text{naive}}$ est l'erreur d'un modèle naïf (ex : prédire $\hat{y}_t = y_{t-1}$).

Interprétation :

- MASE < 1 : meilleur que le modèle naïf
- MASE $= 1$: équivalent au modèle naïf
- MASE > 1 : pire que le modèle naïf

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

def mape(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

def smape(y_true, y_pred):
    return np.mean(2 * np.abs(y_pred - y_true) / (np.abs(y_true) + np.
   abs(y_pred))) * 100

# Evaluation
mae = mean_absolute_error(y_test, predictions)
rmse = np.sqrt(mean_squared_error(y_test, predictions))
mape_score = mape(y_test, predictions)

print(f"MAE: {mae:.2f}")
print(f"RMSE: {rmse:.2f}")
print(f"MAPE: {mape_score:.2f}%")
```

Listing 10 – Calcul des métriques

6 Validation des Modèles de Séries Temporelles

6.1 Time Series Split

Attention

Ne JAMAIS utiliser la validation croisée classique (K-Fold aléatoire) pour les séries temporelles ! Cela créerait du **data leakage** (utiliser le futur pour prédire le passé).

Définition : Time Series Split (Forward Chaining)

Validation séquentielle respectant l'ordre temporel :

- **Fold 1 :** Train $[1, \ldots, t_1]$, Test $[t_1 + 1, \ldots, t_2]$
- **Fold 2 :** Train $[1, \ldots, t_2]$, Test $[t_2 + 1, \ldots, t_3]$
- **Fold 3 :** Train $[1, \ldots, t_3]$, Test $[t_3 + 1, \ldots, t_4]$
- $\vdots$

Le training set grandit progressivement (expanding window).

```python
from sklearn.model_selection import TimeSeriesSplit

tscv = TimeSeriesSplit(n_splits=5)

for fold, (train_idx, test_idx) in enumerate(tscv.split(X)):
    print(f"Fold {fold+1}:")
    print(f"  Train: {train_idx[0]} to {train_idx[-1]}")
    print(f"  Test:  {test_idx[0]} to {test_idx[-1]}")

    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    # Entrainer et evaluer le modele
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    print(f"  Score: {score:.4f}\n")
```

Listing 11 – Time Series Split avec scikit-learn

6.2 Backtesting

Définition : Backtesting

Simuler des prédictions historiques pour évaluer la performance du modèle :

1. Entraîner sur $[1, \ldots, t]$
2. Prédire $\hat{y}_{t+1}$
3. Observer la vraie valeur y_{t+1}
4. Calculer l'erreur $|y_{t+1} - \hat{y}_{t+1}|$
5. Avancer d'un pas : $t \leftarrow t + 1$
6. Répéter (potentiellement ré-entraîner le modèle)

Variantes :
- **Expanding window :** Training set grandit continuellement
- **Rolling window :** Training set de taille fixe (glisse)

```python
def backtest_model(model, data, window_size, test_size):
    predictions = []
    actuals = []

    for i in range(len(data) - window_size - test_size):
        # Train/test split
        train_end = window_size + i
        X_train = data[:train_end]
        y_test = data[train_end]

        # Entrainer
        model.fit(X_train)

        # Predire
        pred = model.predict(steps=1)[0]
        predictions.append(pred)
        actuals.append(y_test)

    # Evaluation
    mae = mean_absolute_error(actuals, predictions)
    return predictions, actuals, mae
```

Listing 12 – Backtesting simple

7 Détection d'Anomalies dans les Séries Temporelles

Définition : Anomalie Temporelle

Observation anormale qui dévie significativement du pattern attendu.

Types d'anomalies :
- **Point anomaly :** Valeur isolée inhabituelle (spike)
- **Contextual anomaly :** Valeur normale ailleurs mais anormale à cet instant
- **Collective anomaly :** Séquence anormale (changement de régime)

7.1 Méthodes Statistiques

Définition : Détection par Seuil (Z-score)

Une observation y_t est anormale si :

$$|z_t| = \left| \frac{y_t - \mu}{\sigma} \right| > \text{seuil}$$

Seuil typique : 3 (99.7% des données dans $[\mu - 3\sigma, \mu + 3\sigma]$)

7.2 Méthodes par Prédiction

Définition : Anomalie par Erreur de Prédiction

1. Entraîner un modèle de forecasting (ARIMA, LSTM)
2. Calculer l'erreur de prédiction $e_t = |y_t - \hat{y}_t|$
3. Anomalie si $e_t >$ seuil (ex : $\mu_e + 3\sigma_e$)

Intuition : Une valeur difficile à prédire est potentiellement anormale.

```python
# Predictions LSTM
model.eval()
predictions = model(X_test).squeeze().detach().numpy()
actuals = y_test.numpy()

# Erreurs de prediction
errors = np.abs(actuals - predictions)

# Seuil (moyenne + 3 * ecart-type)
threshold = errors.mean() + 3 * errors.std()

# Anomalies
anomalies = errors > threshold
print(f"Nombre d'anomalies detectees: {anomalies.sum()}")
print(f"Indices: {np.where(anomalies)[0]}")
```

Listing 13 – Détection d'anomalies par prédiction

8 Best Practices pour le Forecasting

8.1 Preprocessing

1. **Gestion des valeurs manquantes :**
   — Interpolation linéaire, spline, forward/backward fill
   — Modèles robustes (Prophet gère nativement)
2. **Détection et traitement des outliers :**
   — Z-score, IQR, isolation forest
   — Winsorization (clipping), remplacement
3. **Normalisation/Standardisation :**
   — Min-Max scaling : $x' = \frac{x - \min}{\max - \min}$
   — Standardisation : $x' = \frac{x - \mu}{\sigma}$
   — **Important :** Calculer stats sur train uniquement !

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# Fit sur train uniquement
scaler.fit(X_train)
```

```
7
8  # Transform train et test
9  X_train_scaled = scaler.transform(X_train)
10 X_test_scaled = scaler.transform(X_test)
11
12 # Predictions
13 predictions_scaled = model.predict(X_test_scaled)
14
15 # Inverse transform pour revenir a l'echelle originale
16 predictions = scaler.inverse_transform(predictions_scaled)
```

Listing 14 – Normalisation correcte

8.2 Feature Engineering

1. **Features temporelles :**
   — Jour de la semaine, mois, trimestre, année
   — Week-end vs jour de semaine
   — Jours fériés
2. **Lags :** $y_{t-1}, y_{t-2}, \ldots, y_{t-k}$
3. **Rolling statistics :**
   — Moyenne mobile : $\text{MA}_k(t) = \frac{1}{k}\sum_{i=0}^{k-1} y_{t-i}$
   — Écart-type mobile
   — Min/Max mobile
4. **Différenciation :** $\Delta y_t = y_t - y_{t-1}$
5. **Features cycliques :** Pour encoder périodicité

$$\text{hour\_sin} = \sin\left(\frac{2\pi \cdot \text{hour}}{24}\right)$$
$$\text{hour\_cos} = \cos\left(\frac{2\pi \cdot \text{hour}}{24}\right)$$

8.3 Sélection de Modèle

Table 2 – Quand utiliser quel modèle ?

Modèle	Cas d'usage
ARIMA	Séries univariées stationnaires, patterns linéaires, taille modérée
SARIMA	Saisonnalité claire et périodique
Prophet	Saisonnalités multiples, jours fériés, tendances changeantes, données manquantes
LSTM/GRU	Dépendances long terme, patterns non-linéaires, multivarié
Transformer	Très longues séquences, attention importante, ressources suffisantes
XGBoost	Features engineered riches, régression avec lags

8.4 Ensemble Methods

Combiner plusieurs modèles pour améliorer la robustesse :

```
# Predictions de plusieurs modeles
pred_arima = model_arima.forecast(steps=10)
pred_lstm = model_lstm.predict(X_test)
pred_prophet = model_prophet.predict(future)['yhat'].values

# Moyenne simple
pred_ensemble = (pred_arima + pred_lstm + pred_prophet) / 3

# Moyenne ponderee (poids bases sur performance validation)
weights = [0.3, 0.5, 0.2]  # ARIMA, LSTM, Prophet
pred_ensemble = (weights[0] * pred_arima +
                 weights[1] * pred_lstm +
                 weights[2] * pred_prophet)
```

Listing 15 – Ensemble de prédictions

9 Applications Pratiques

1. **Finance et Trading :**
   - Prédiction de prix d'actions, indices boursiers
   - Volatilité, détection d'anomalies (flash crashes)
   - Trading algorithmique
2. **Énergie :**
   - Forecasting de la demande électrique (load forecasting)
   - Prédiction de production éolienne/solaire
   - Optimisation de la distribution
3. **E-commerce et Retail :**
   - Prévision des ventes pour gestion des stocks
   - Optimisation des prix dynamiques
   - Détection de fraudes (transactions anormales)
4. **IoT et Industrie :**
   - Maintenance prédictive (détection de pannes)
   - Monitoring de capteurs
   - Optimisation de production
5. **Santé :**
   - Monitoring de signes vitaux (ECG, EEG)
   - Prédiction d'épidémies
   - Détection précoce de pathologies
6. **Météorologie :**
   - Prévisions météo courte/moyenne échéance
   - Modèles climatiques

10 Résumé du Chapitre

10.1 Points Clés

— **Stationnarité :** Propriété essentielle pour les modèles classiques (ARIMA)
— **Composantes :** Tendance, saisonnalité, résidu
— **ARIMA :** Modèle statistique puissant pour séries univariées
— **Prophet :** Framework simple pour saisonnalités multiples et jours fériés
— **LSTM/GRU :** Deep Learning pour dépendances complexes et long terme
— **Attention/Transformers :** État de l'art pour séquences très longues
— **Validation :** Time Series Split, backtesting (jamais de CV classique)
— **Métriques :** MAE, RMSE, MAPE, MASE
— **Feature Engineering :** Lags, rolling stats, features temporelles

10.2 Formules Essentielles

Formules à retenir

ARIMA(p,d,q) :

$$\nabla^d y_t = c + \sum_{i=1}^{p} \phi_i \nabla^d y_{t-i} + \epsilon_t + \sum_{j=1}^{q} \theta_j \epsilon_{t-j}$$

Prophet :

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

Métriques :

$$\text{MAE} = \frac{1}{n} \sum |y_i - \hat{y}_i|$$

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

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

11 Exercices

11.1 Questions de compréhension

1. Pourquoi la validation croisée classique (K-Fold) est-elle inappropriée pour les séries temporelles ?
2. Quelle est la différence entre un modèle AR et un modèle MA ?
3. Expliquez le rôle de chaque paramètre dans ARIMA(p,d,q).
4. Quand utiliser Prophet plutôt qu'ARIMA ?
5. Comment gérer une série temporelle avec tendance croissante ?

11.2 Exercices pratiques

1. **Prédiction de ventes mensuelles :**
   — Charger un dataset de ventes (ex : retail sales)

   - Analyser la décomposition (tendance, saisonnalité)
   - Tester la stationnarité (ADF test)
   - Comparer ARIMA vs Prophet
2. **Forecasting de consommation électrique :**
   - Dataset multivarié (température, jour, heure)
   - Feature engineering (lags, rolling stats)
   - Entraîner un LSTM
   - Évaluer avec RMSE et MAPE
3. **Détection d'anomalies :**
   - Charger un dataset avec anomalies
   - Entraîner un modèle LSTM
   - Détecter les anomalies via erreur de prédiction
   - Visualiser les résultats

Solutions disponibles dans `12_exercices.ipynb`

12 Pour Aller Plus Loin

12.1 Lectures Recommandées

- Hyndman, R.J., & Athanasopoulos, G. (2021). *Forecasting : Principles and Practice* (3rd ed.) - Livre de référence gratuit en ligne
- Box, G.E.P., Jenkins, G.M., Reinsel, G.C., & Ljung, G.M. (2015). *Time Series Analysis : Forecasting and Control*
- Taylor, S.J., & Letham, B. (2018). "Forecasting at Scale". *The American Statistician*
- Lim, B., et al. (2021). "Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting". *International Journal of Forecasting*

12.2 Ressources en Ligne

- Documentation statsmodels : https://www.statsmodels.org/stable/tsa.html
- Prophet (Meta) : https://facebook.github.io/prophet/
- PyTorch Forecasting : https://pytorch-forecasting.readthedocs.io/
- Darts (time series library) : https://unit8co.github.io/darts/
- Course "Time Series" de Penn State : https://online.stat.psu.edu/stat510/

12.3 Bibliothèques Python

- **statsmodels** : ARIMA, SARIMA, décomposition, tests statistiques
- **prophet** : Framework de Meta pour forecasting
- **darts** : Bibliothèque complète (classique + DL)
- **sktime** : Extension scikit-learn pour time series
- **pytorch-forecasting** : DL pour forecasting (Temporal Fusion Transformer)
- **tslearn** : ML pour séries temporelles (clustering, DTW)

12.4 Prochaines Étapes

— Approfondir les Transformers pour séries temporelles
— Étudier les modèles probabilistes (Bayésiens, GARCH)
— Explorer la causalité dans les séries temporelles (Granger causality)
— Appliquer à des problèmes réels de votre domaine

Références

1. Box, G. E. P., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time Series Analysis : Forecasting and Control* (5th ed.). John Wiley & Sons.
2. Hyndman, R. J., & Athanasopoulos, G. (2021). *Forecasting : Principles and Practice* (3rd ed.). OTexts. https://otexts.com/fpp3/
3. Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *The American Statistician*, 72(1), 37-45.
4. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780.
5. Vaswani, A., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.
6. Lim, B., Ark, S. Ö., Loeff, N., & Pfister, T. (2021). Temporal Fusion Transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4), 1748-1764.
7. Seabold, S., & Perktold, J. (2010). Statsmodels : Econometric and statistical modeling with Python. *Proceedings of the 9th Python in Science Conference*, 57, 61.