Note méthodologique – Prédiction du taux de risque bancaire (panel multi-banques)

> **Objectif** : expliquer clairement *comment* le taux de risque est prédit à partir des indicateurs prudentiels (CET1, ratios, LCR/NSFR, SREP…) sur un jeu de données panel (plusieurs banques, plusieurs années), et documenter le code pas-à-pas.

---

1) Problème posé et définition de la cible

- **Unité d'observation** : (banque *i*, période *t*), p.ex. `Bank_A – 2021`.
- **Cible à prédire** : un **taux de risque** (ex.: *coût du risque* en % des encours, *NPL ratio*, etc.). On note $y_{i,t}$ la valeur observée à *t*.
- **Prédiction à horizon T+1** : pour éviter toute fuite d'information, on prédit $y_{i,t+1}$ à partir de variables connues à *t*. Dans le code :

$$\texttt{risk\_rate\_t1} = \text{shift}_{-1}(\texttt{risk\_rate}) \text{ par banque.}$$

Ainsi, la ligne (i,t) sert à prédire la période suivante (i,t+1).

---

2) Variables explicatives (features)

2.1. Données brutes (exemples)

- **Capitaux** (en M€) : `CET1`, `Tier1`, `TotalCapital`, `RWA`.
- **Ratios** (en %) : `CET1_ratio`, `Tier1_ratio`, `Total_ratio`, `Leverage_ratio`, `LCR`, `NSFR`.
- **Exigences SREP / buffers** (en %) : `EU_7a..EU_7d`, `CCB`, `CCyB`.

2.2. Variables dérivées (ingénierie)

- **Distance au requirement** (marge de solvabilité) :

$$\texttt{distance\_CET1} = \texttt{CET1\_ratio} - (\texttt{EU\_7d} + \texttt{CCB} + \texttt{CCyB}).$$

  Interprétation : marge de sécurité du ratio CET1 au-dessus du total SREP + buffers macro.

- **Densité de RWA** (si `TotalAssets` dispo) :

$$\texttt{RWA\_density} = \frac{\texttt{RWA}}{\texttt{TotalAssets}}.$$

- **Dynamiques temporelles** (par banque) :
- *Lags* : $x_{i,t-1}, x_{i,t-4} \rightarrow$ `feature_l1`, `feature_l4`.
- *Variations* : $\Delta x_{i,t} = x_{i,t} - x_{i,t-1}$.
- *Moyennes mobiles* : $\overline{x}^{(4)}_{i,t} \rightarrow$ `feature_ma4`.

  Ces transformations capturent **tendance**, **inertie** et **chocs** des indicateurs prudentiels.

3) Préparation des données (pipeline)

1. **Colonne temporelle** : si seules des années existent, on crée `date = to_datetime(year)`.
2. **Tri** : `sort_values(["bank_id","date"])`.
3. **Création de la cible** : `risk_rate_t1 = groupby(bank_id)["risk_rate"].shift(-1)`.
4. **Suppression des lignes sans cible future** : `dropna(subset=["risk_rate_t1"])`.
5. **Création des lags & moyennes** par banque pour une liste de colonnes de base.
6. **Sélection des features** : toutes les colonnes numériques pertinentes **hors** `bank_id`, `date/year`, `risk_rate`, `risk_rate_t1`.

   ⚠️Les lags et moyennes sont calculés **après tri par temps** et **par banque** pour garantir l'ordre et éviter les fuites.

---

4) Modélisation

4.1. Modèles utilisés

- **Ridge** (linéaire régularisé) – interprétable, baseline solide.
- **Gradient Boosting Regressor** (non-linéaire) – capte interactions/effets non linéaires.

Chaque modèle est encapsulé dans un **pipeline** :

```
Pipeline([
  ("prep", StandardScaler() sur les features numériques),
  ("model", Ridge ou GradientBoostingRegressor)
])
```

4.2. Validation temporelle (backtest)

- **TimeSeriesSplit (walk-forward)** en K plis sans mélange du futur.
- À chaque pli : fit sur le passé, test sur la fenêtre suivante.
- **Prédictions OOF** (*out-of-fold*) collectées pour chaque observation test → estimation honnête de l'erreur.

4.3. Métriques

- **MAE** : $\frac{1}{n}\sum|\hat{y}-y|$ (robuste aux outliers en échelle %)
- **RMSE** : $\sqrt{\frac{1}{n}\sum(\hat{y}-y)^2}$ (pénalise fort les grandes erreurs)

  Dans le code, compatibilité large : `rmse = sqrt(mean_squared_error(y_true, y_pred))`.

---

5) Importance des variables (explicabilité)

Permutation importance sur un *jeu de test* (idéalement le **dernier pli** – le plus récent) : 1. On permute une feature (on casse son lien avec *y*), 2. On mesure la baisse de performance (Δ MAE/RMSE), 3. Plus la dégradation est forte → plus la feature est importante.

Sortie : tableau `feature`, `importance_mean`, `importance_std` (moyenne sur plusieurs répétitions).

---

6) Prévision T+1 par banque

Objectif : produire la prévision $\hat{y}_{i,T+1}$ **pour chaque banque** à partir de la **dernière observation disponible** (i, T) .

Algorithme : 1. Pour chaque `bank_id`, on prend la **dernière ligne** (la plus récente). 2. On applique **le pipeline entraîné** aux features de cette ligne. 3. On assemble un DataFrame : `bank_id`, `date` (ou `year`), `pred_risk_rate_Tplus1`.

> Cela donne la **meilleure estimation** du taux de risque pour la **période suivante**.

---

7) Structure du code – explication étape par étape

7.1. Lecture & normalisation du temps

```
# Lecture CSV et date
df = pd.read_csv("banks_panel.csv")
if "date" not in df.columns and "year" in df.columns:
    df["date"] = pd.to_datetime(df["year"].astype(int), format="%Y")

df = df.sort_values(["bank_id","date"]).copy()
```

7.2. Cible décalée (T+1)

```
# Cible future par banque
df["risk_rate_t1"] = df.groupby("bank_id")["risk_rate"].shift(-1)
df = df.dropna(subset=["risk_rate_t1"]).copy()
```

7.3. Ingénierie de features

```
# Distance au requirement si colonnes présentes
if set(["CET1_ratio","EU_7d","CCB","CCyB"]).issubset(df.columns):
    df["distance_CET1"] = df["CET1_ratio"] - (df["EU_7d"] + df["CCB"] +
df["CCyB"])
```

```
# Lags & moyennes par banque
base_cols = [c for c in [
  "CET1_ratio","Tier1_ratio","Total_ratio","Leverage_ratio","LCR","NSFR",
  "EU_7a","EU_7b","EU_7c","EU_7d","distance_CET1","RWA_density",
  "CET1","Tier1","TotalCapital","RWA"
] if c in df.columns]

def make_lags(group, cols, lags=(1,4)):
    g = group.sort_values("date").copy()
    for c in cols:
        for L in lags:
            g[f"{c}_l{L}"] = g[c].shift(L)
    return g

def make_rollings(group, cols, windows=(4,)):
    g = group.sort_values("date").copy()
    for c in cols:
        for w in windows:
            g[f"{c}_ma{w}"] = g[c].rolling(w, min_periods=1).mean()
    return g

# Application groupée
df = df.groupby("bank_id", group_keys=False).apply(make_lags, cols=base_cols,
lags=(1,4))
df = df.groupby("bank_id", group_keys=False).apply(
    make_rollings,
    cols=[c for c in base_cols if ("ratio" in c) or (c in ["LCR","NSFR"])],
    windows=(4,)
)
```

7.4. Sélection des features & backtest

```
# Colonnes explicatives
a_exclure = ["bank_id","date","year","risk_rate","risk_rate_t1"]
feature_cols = [c for c in df.columns if c not in a_exclure]

# Entraînement (Ridge ou GBRT)
from math import sqrt
from sklearn.model_selection import TimeSeriesSplit
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.linear_model import Ridge
from sklearn.ensemble import GradientBoostingRegressor

preproc = ColumnTransformer([("num", StandardScaler(), feature_cols)],
remainder="drop")
model = GradientBoostingRegressor(random_state=42)  # ou Ridge(...)
pipe = Pipeline([("prep", preproc), ("model", model)])
```

```
tscv = TimeSeriesSplit(n_splits=5)
X, y = df[feature_cols], df["risk_rate_t1"]

import numpy as np
OOF = np.zeros(len(df))
metrics = []

for fold, (tr, te) in enumerate(tscv.split(X, y), 1):
    pipe.fit(X.iloc[tr], y.iloc[tr])
    y_pred = pipe.predict(X.iloc[te])
    OOF[te] = y_pred
    mae = mean_absolute_error(y.iloc[te], y_pred)
    rmse = sqrt(mean_squared_error(y.iloc[te], y_pred))
    metrics.append({"fold": fold, "MAE": mae, "RMSE": rmse})

# Fit final sur tout l'historique
pipe.fit(X, y)
```

7.5. Importances par permutation (sur le dernier pli)

```
from sklearn.inspection import permutation_importance
last_tr, last_te = list(tscv.split(X, y))[-1]
pi = permutation_importance(pipe, X.iloc[last_te], y.iloc[last_te],
n_repeats=10, random_state=42)
fi_df = pd.DataFrame({
    "feature": feature_cols,
    "importance_mean": pi.importances_mean,
    "importance_std": pi.importances_std,
}).sort_values("importance_mean", ascending=False)
```

7.6. Prévision T+1

```
def forecast_last_period(df, model, feature_cols):
    last_rows = df.sort_values("date").groupby("bank_id").tail(1).copy()
    preds = model.predict(last_rows[feature_cols])
    out = last_rows[["bank_id","date"]].copy()
    out["pred_risk_rate_Tplus1"] = preds
    return out

forecast_df = forecast_last_period(df, pipe, feature_cols)
```

7.7. Exports & visualisation

```
# Sauvegardes CSV
pd.DataFrame(metrics).to_csv("metrics_backtest.csv", index=False)
fi_df.to_csv("feature_importance_permutation_lastfold.csv", index=False)
```

```
# Ajout des prédictions OOF
out_oof = df[["bank_id","year","date","risk_rate"]].copy()
out_oof["predicted_risk_rate"] = OOF
out_oof.to_csv("oof_predictions.csv", index=False)

# Prévisions T+1 par banque
forecast_df.to_csv("forecast_Tplus1_by_bank.csv", index=False)
```

---

8) Interprétation des résultats

- **MAE / RMSE par pli** : stabilité des erreurs dans le temps (éviter un pli très dégradé = drift potentiel).
- **OOF vs réel** : vérifier biais systématique (sur/sous-estimation), par banque et globalement.
- **Features importantes** : valider qu'elles sont plausibles (ex. *distance_CET1*, *LCR*, *NSFR*, *Tier1_ratio*). Si une feature étrange domine, suspecter un proxy temporel ou une fuite.
- **Prévisions T+1** : cohérentes avec la trajectoire des fondamentaux ?

---

9) Bonnes pratiques & pièges courants

- **Pas de fuite d'info** :
- Cible **décalée** ($t+1$),
- Lags/MAs créés *uniquement* à partir d'informations à t,

- Validation temporelle (pas de shuffle aléatoire).

- **Colonnes temporelles** : uniformiser `date` (datetime) même si vous partez de `year`.

- **Compatibilité sklearn** : pour le RMSE, utiliser `sqrt(mean_squared_error(...))` (compatible anciennes versions).
- **Taille par banque** : au moins 2 périodes pour créer `risk_rate_t1` ; avec peu d'historique, préférer un modèle simple.
- **Robustesse** : comparer à une baseline naïve (persistance : $\hat{y}_{t+1} = y_t$).

---

10) Extensions possibles

- **Variables macro** par pays (PIB, chômage, taux directeurs, spreads) → jointure par *pays d'origine x période*.
- **Effets fixes** (statistiques panel) pour capter l'hétérogénéité structurelle par banque.
- **Modèles gradientés modernes** : LightGBM/XGBoost/CatBoost (avec prudence sur les fuites et la validation temporelle).
- **Stress testing** : simuler des chocs (–10% CET1_ratio, –20 pts de LCR) et mesurer l'élasticité du risque prédit.

TL;DR – Chaîne de calcul

1. Normaliser le temps → `date`.
2. Créer **cible future** → `risk_rate_t1 = shift(-1)` par banque.
3. Featurer : lags, moyennes, `distance_CET1`, etc.
4. Backtest **walk-forward** (TimeSeriesSplit) → MAE/RMSE + OOF.
5. **Permutation importance** sur le dernier pli.
6. **Forecast T+1** par banque à partir de la dernière période connue.
7. Exporter tableaux (metrics, OOF, importances, forecasts) et tracer 1–2 graphiques d sanity check.