🔍 Amélioration d'un pipeline de classification supervisée

Ce notebook applique une série d'améliorations sur un pipeline de classification en machine learning en suivant les axes suivants :

- Feature engineering
- Validation croisée
- Comparaison de modèles
- Optimisation des hyperparamètres
- Évaluation approfondie

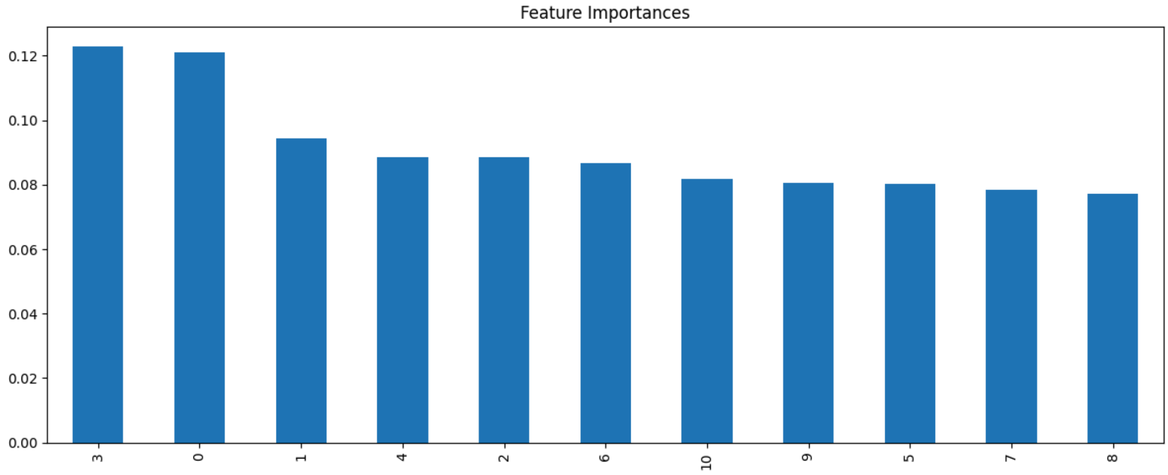
In [4]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMa
from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier,StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

from xgboost import XGBClassifier
import lightgbm as lgb

import warnings
import shap
from sklearn.feature_selection import SelectFromModel
from sklearn.decomposition import PCA
import joblib
import warnings
warnings.filterwarnings('ignore')
```

In [5]:

```
# Chargement des données prétraitées
Xtrain = pd.read_csv('../data/data_preprocessed/data_resampled_Xtrain.csv')
ytrain = pd.read_csv('../data/data_preprocessed/data_resampled_ytrain.csv')
Xtest = pd.read_csv('../data/data_preprocessed/data_resampled_Xtest.csv')
ytest = pd.read_csv('../data/data_preprocessed/data_resampled_ytest.csv')

# Conversion éventuelle des y en Series
if isinstance(ytrain, pd.DataFrame):
    ytrain = ytrain.iloc[:, 0]
if isinstance(ytest, pd.DataFrame):
    ytest = ytest.iloc[:, 0]

scaler = StandardScaler()
Xtrain_scaled = scaler.fit_transform(Xtrain)
Xtest_scaled = scaler.transform(Xtest)
```

In [8]:
```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(Xtrain, ytrain)
importances = pd.Series(rf.feature_importances_, index=Xtrain.columns)
importances.sort_values(ascending=False).plot(kind='bar', figsize=(12,5), title=
plt.tight_layout()
plt.show()
```



In [9]:
```
scaler = StandardScaler()
Xtrain_scaled = scaler.fit_transform(Xtrain)
Xtest_scaled = scaler.transform(Xtest)
```

In [10]:
```
cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
```

In [11]:
```
model = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
scores = cross_val_score(model, Xtrain_scaled, ytrain, cv=cv, scoring='f1_weight
print("✅ Random Forest:")
print("F1 score moyen = {:.4f} (+/- {:.4f})".format(scores.mean(), scores.std())
```

```
✅ Random Forest:
F1 score moyen = 0.8689 (+/- 0.0003)
```

In [12]:
```
model = LogisticRegression(max_iter=1000)
scores = cross_val_score(model, Xtrain_scaled, ytrain, cv=cv, scoring='f1_weight
print("✅ Logistic Regression:")
print("F1 score moyen = {:.4f} (+/- {:.4f})".format(scores.mean(), scores.std())
```

```
✅ Logistic Regression:
F1 score moyen = 0.5039 (+/- 0.0009)
```

In [13]:
```
model = KNeighborsClassifier()
scores = cross_val_score(model, Xtrain_scaled, ytrain, cv=cv, scoring='f1_weight
print("✅ KNN:")
print("F1 score moyen = {:.4f} (+/- {:.4f})".format(scores.mean(), scores.std())
```

```
✅ KNN:
F1 score moyen = 0.8063 (+/- 0.0002)
```

In [19]:
```
from sklearn.svm import LinearSVC
from sklearn.model_selection import cross_val_score, StratifiedKFold, train_test

# Échantillonnage pour accélérer (20% des données)
X_small, _, y_small, _ = train_test_split(Xtrain_scaled, ytrain, train_size=0.5,

# SVM linéaire, bien plus rapide
model = LinearSVC(max_iter=10000)
```

```
cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
scores = cross_val_score(model, X_small, y_small, cv=cv, scoring='f1_weighted')

print("✅ Linear SVM (20% données) :")
print("F1 score moyen = {:.4f} (+/- {:.4f})".format(scores.mean(), scores.std())
```

```
✅ Linear SVM (20% données) :
F1 score moyen = 0.4676 (+/- 0.0015)
```

In [20]:

```
model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_st
scores = cross_val_score(model, Xtrain_scaled, ytrain, cv=cv, scoring='f1_weight
print("✅ XGBoost:")
print("F1 score moyen = {:.4f} (+/- {:.4f})".format(scores.mean(), scores.std())
```

```
✅ XGBoost:
F1 score moyen = 0.6612 (+/- 0.0010)
```

In [21]:

```
model = lgb.LGBMClassifier()
scores = cross_val_score(model, Xtrain_scaled, ytrain, cv=cv, scoring='f1_weight
print("✅ LightGBM:")
print("F1 score moyen = {:.4f} (+/- {:.4f})".format(scores.mean(), scores.std())
```

```
✅ LightGBM:
F1 score moyen = 0.6125 (+/- 0.0005)
```

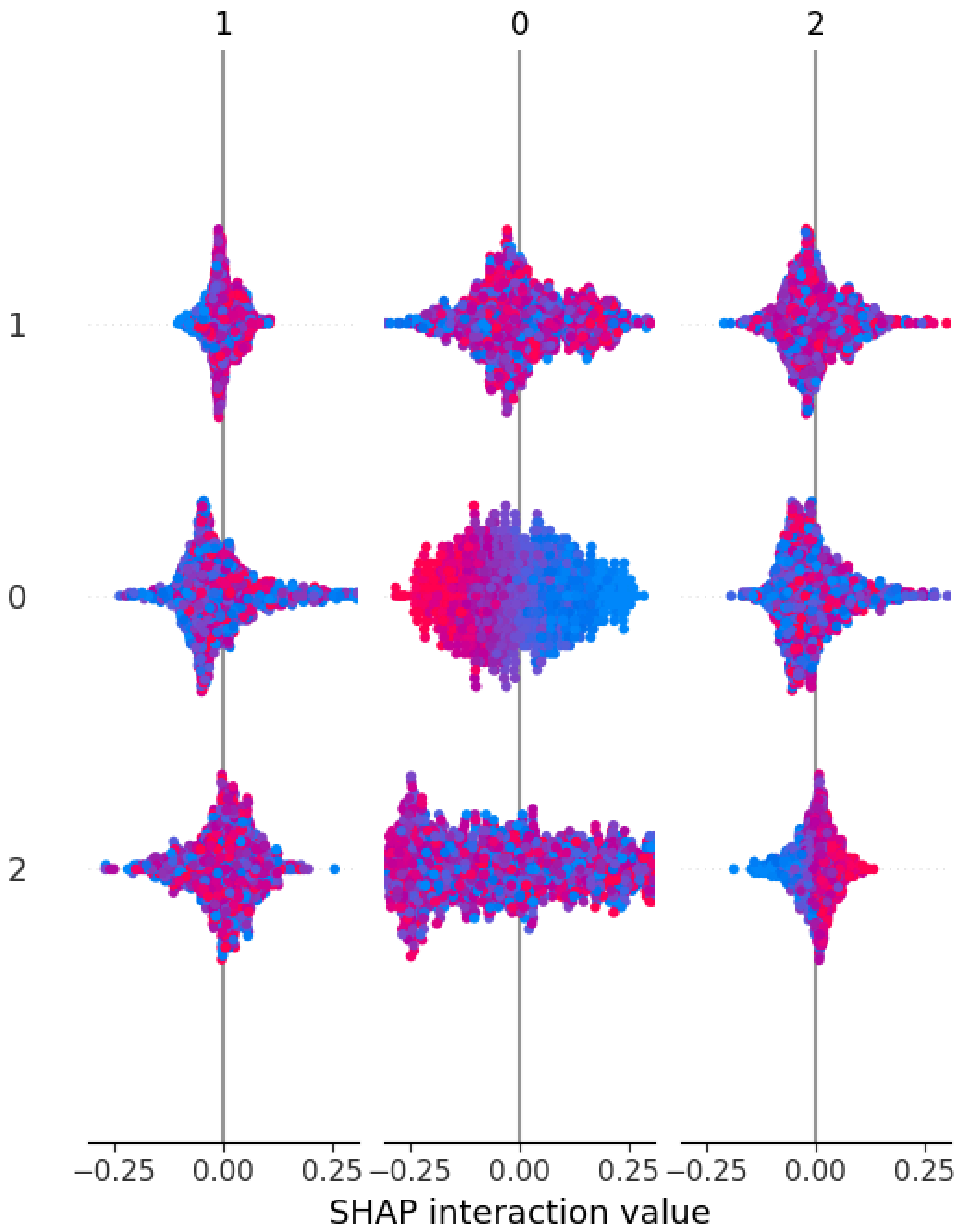
In [6]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold, train_t

# ⏱ Échantillonnage (optionnel mais recommandé pour rapidité)
X_small, _, y_small, _ = train_test_split(Xtrain_scaled, ytrain, train_size=0.3,

# 🔧 Moins de combinaisons à tester, mais variées
param_dist = {
    'n_estimators': [100, 200, 300],
    'max_depth': [20, 40, None],
    'min_samples_split': [2, 5, 10],
    'max_features': ['sqrt', 'log2']
}

# 📊 Validation croisée stratifiée à 3 plis
cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

# 🎲 RandomizedSearch avec 10 itérations seulement
rf = RandomForestClassifier(random_state=42, n_jobs=-1)
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist,
    n_iter=10,
    cv=cv,
    scoring='f1_weighted',
    random_state=42,
    n_jobs=-1,
    verbose=1
)

# 🔄 Entraînement
random_search.fit(X_small, y_small)

# 📌 Résultats
```

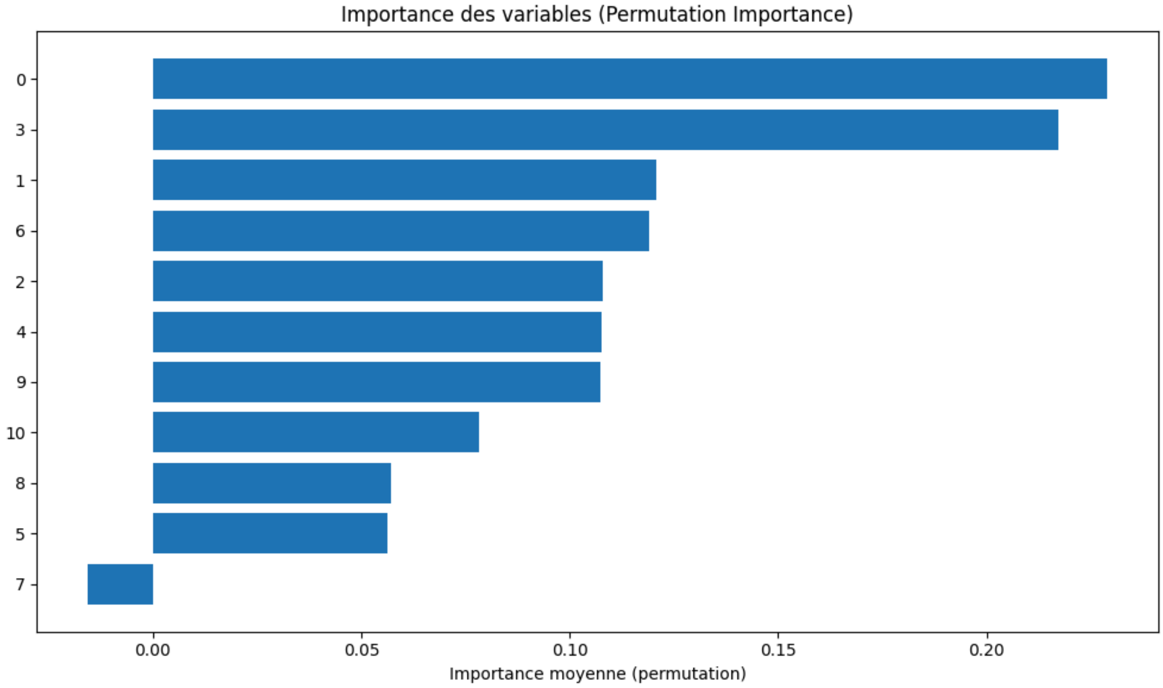
```
print("✅ Meilleurs paramètres :", random_search.best_params_)
best_rf = random_search.best_estimator_
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
✅ Meilleurs paramètres : {'n_estimators': 200, 'min_samples_split': 2, 'max_fea
tures': 'sqrt', 'max_depth': 40}
```

In [7]:
```
# ⚠️ Échantillonnage pour accélérer le calcul (1000 lignes suffisent pour le rés
X_sample = Xtrain_scaled[:1000]

# Initialisation de l'explainer uniquement sur un sous-ensemble
explainer = shap.TreeExplainer(best_rf)

# Calcul des valeurs SHAP sur l'échantillon
shap_values = explainer.shap_values(X_sample)

# Affichage du graphique résumé
shap.summary_plot(shap_values, X_sample, feature_names=Xtrain.columns.tolist())
```



```
In [13]: from sklearn.inspection import permutation_importance

         # ⚙️ Importance par permutation sur le modèle optimisé (best_rf)
         result = permutation_importance(
             best_rf,
             Xtest_scaled,
             ytest,
             n_repeats=10,
             random_state=42,
             scoring='f1_weighted',
             n_jobs=-1
         )

         # 📊 Affichage des 15 variables les plus importantes selon permutation
         perm_sorted_idx = result.importances_mean.argsort()[::-1][:15]
         plt.figure(figsize=(10, 6))
         plt.barh(range(len(perm_sorted_idx)), result.importances_mean[perm_sorted_idx][:
         plt.yticks(range(len(perm_sorted_idx)), np.array(Xtrain.columns)[perm_sorted_idx
         plt.xlabel("Importance moyenne (permutation)")
         plt.title("Importance des variables (Permutation Importance)")
         plt.tight_layout()
         plt.show()
```

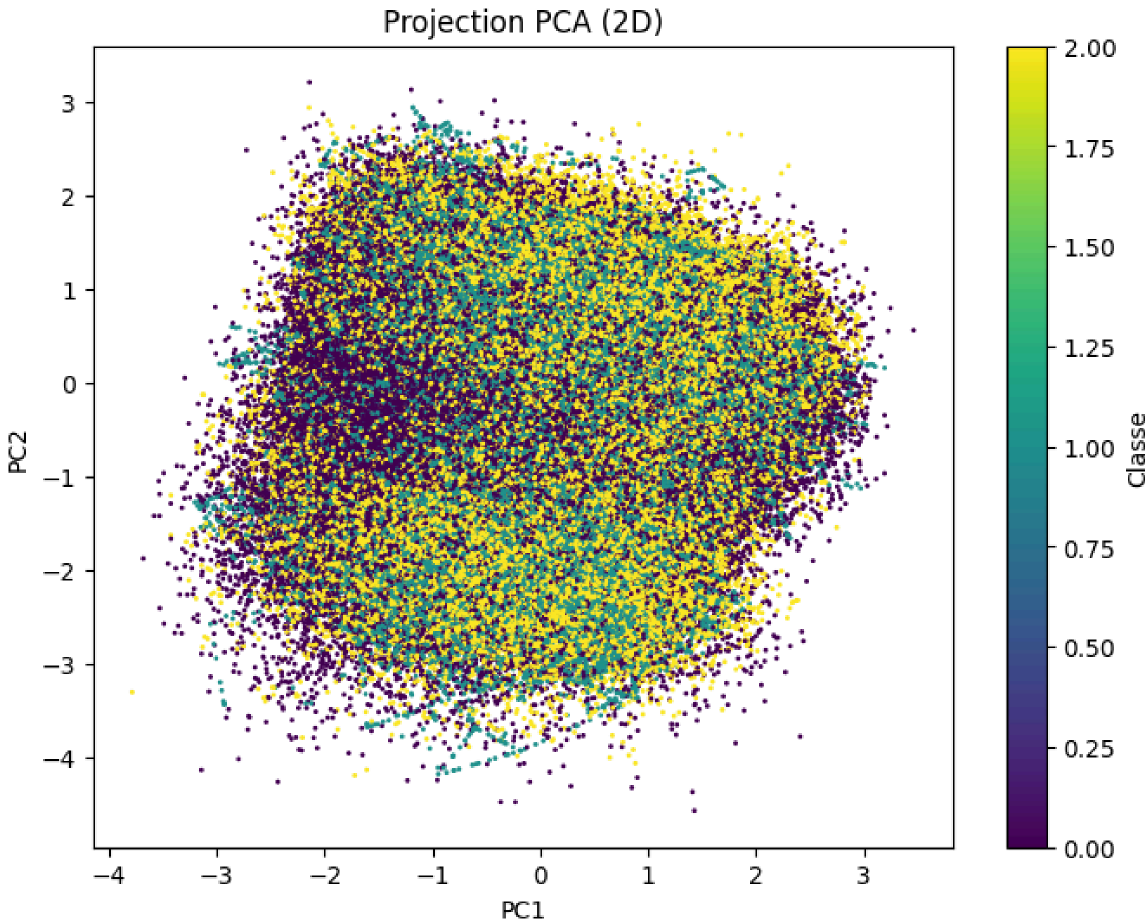



```
In [14]: selector = SelectFromModel(best_rf, threshold='median')
         Xtrain_reduced = selector.fit_transform(Xtrain_scaled, ytrain)
         Xtest_reduced = selector.transform(Xtest_scaled)
         print("✅ Nombre de variables après réduction :", Xtrain_reduced.shape[1])
```

✅ Nombre de variables après réduction : 6

```
In [15]: pca = PCA(n_components=2)
         X_vis = pca.fit_transform(Xtrain_scaled)
         plt.figure(figsize=(8, 6))
         plt.scatter(X_vis[:, 0], X_vis[:, 1], c=ytrain, cmap='viridis', s=1)
         plt.title("Projection PCA (2D)")
         plt.xlabel("PC1")
         plt.ylabel("PC2")
```

```
plt.colorbar(label="Classe")
plt.show()
```



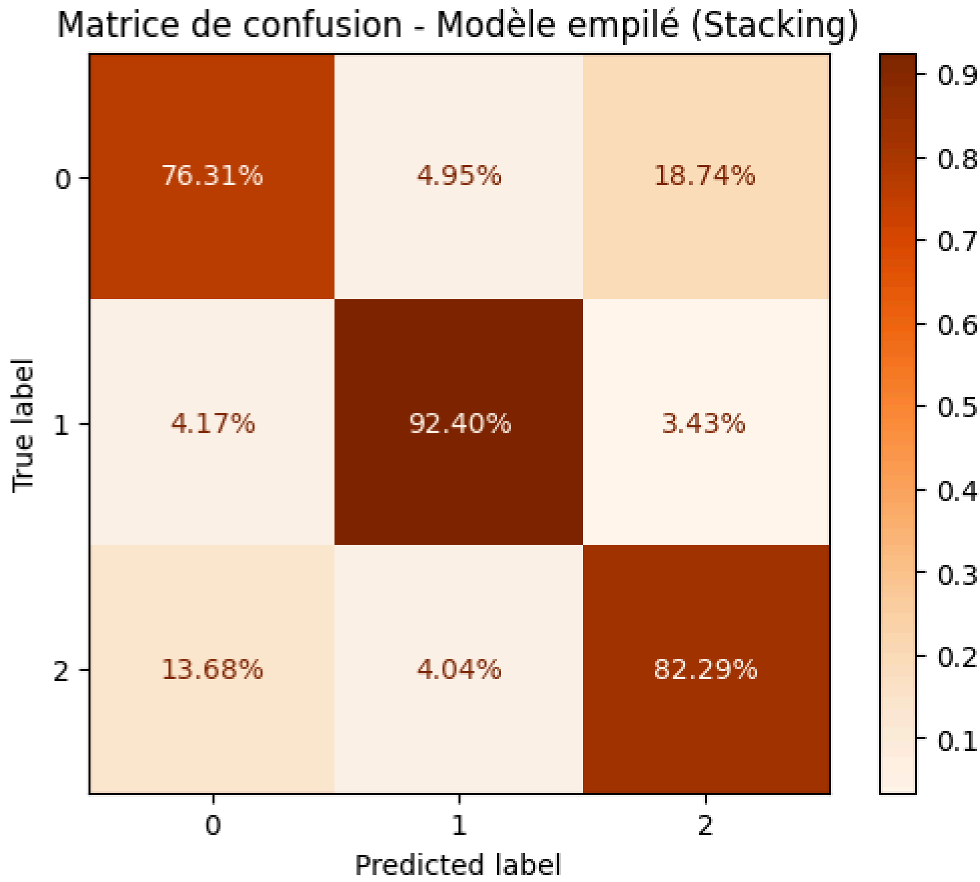
In [16]:

```
estimators = [
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)),
    ('knn', KNeighborsClassifier(n_neighbors=5))
]
stack_model = StackingClassifier(estimators=estimators, final_estimator=Logistic
stack_model.fit(Xtrain_reduced, ytrain)
y_pred_stack = stack_model.predict(Xtest_reduced)
print(classification_report(ytest, y_pred_stack))
```

```
              precision    recall  f1-score   support

         0.0       0.81      0.76      0.79     37783
         1.0       0.91      0.92      0.92     38258
         2.0       0.79      0.82      0.81     37992

    accuracy                           0.84    114033
   macro avg       0.84      0.84      0.84    114033
weighted avg       0.84      0.84      0.84    114033
```

In [17]:

```
cm = confusion_matrix(ytest, y_pred_stack, normalize='true')
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Oranges', values_format='.2%')
plt.title("Matrice de confusion - Modèle empilé (Stacking)")
plt.show()
```



In [12]:
```
joblib.dump(stack_model, "stacked_model.joblib")
print("✅ Modèle sauvegardé dans 'stacked_model.joblib'")
```

✅ Modèle sauvegardé dans 'stacked_model.joblib'

In [18]:
```
# 🧪 Analyse des erreurs de prédiction sur Xtest
import pandas as pd

# Ajouter prédictions au jeu de test
Xtest_df = pd.DataFrame(Xtest_scaled, columns=Xtrain.columns)
Xtest_df["true"] = ytest.values
Xtest_df["pred"] = best_rf.predict(Xtest_scaled)

# Filtrer les erreurs
erreurs = Xtest_df[Xtest_df["true"] != Xtest_df["pred"]]

# 🔍 Exemple : erreurs sur la classe 0
erreurs_classe_0 = erreurs[erreurs["true"] == 0]
print(f"Nombre d’erreurs sur la classe 0 : {len(erreurs_classe_0)}")

# Afficher un échantillon d’erreurs
erreurs_classe_0.head(10)
```

Nombre d’erreurs sur la classe 0 : 11042

Out[18]:

	0	1	2	3	4	5	6	7
15	-0.013036	-0.491255	0.678795	-0.770446	-0.459360	-1.012925	2.129400	0.873723
17	0.470416	0.155337	-1.435974	-0.251029	0.142824	-0.598109	0.144071	-0.566692
26	-0.538886	0.092102	0.828546	0.095932	0.515383	-0.919232	0.950807	-1.181823
29	-0.165292	0.683842	-0.481751	0.176246	-0.258394	1.029060	0.763822	0.476298
37	0.082752	0.436854	-0.518907	0.779657	0.499399	1.161913	0.332596	0.225632
38	0.160016	0.940078	-0.871336	-0.236726	-0.314107	0.094005	0.626870	1.888606
43	0.498578	1.104981	0.347925	0.482281	0.080814	-1.506598	0.881008	-1.783969
82	0.852024	1.591006	-1.144207	0.026440	-0.261239	0.349142	1.460657	-0.658323
87	-0.412293	-0.057977	0.075903	0.593495	-0.156388	0.368608	0.371145	-0.744539
94	1.576851	0.139133	-2.597633	2.527451	-0.485173	1.673172	0.190080	0.562760