Université Mohammed Premier
Faculté des Sciences Département d'Informatique Oujda

A.U 2024 /2025

Rapport du Projet :

Prédiction des défauts de crédit

Master Intelligence Artificielle et Science des Données

Module : Machine Learning

Réalisé par :

- ✓ ***BEN AMER LAHCEN***
- ✓ ***EL GHAFFOULI OUMAYMA***

Encadré par :

- ✓ ***Mr. M. Amine Kasmi***

Introduction :

La prédiction des défauts de crédit bancaire constitue un enjeu crucial pour les institutions financières. Elle permet d'évaluer la probabilité qu'un client ne rembourse pas ses dettes, et ainsi de réduire les risques financiers liés à l'octroi de crédits. Dans ce projet, nous avons développé un système intelligent de prédiction de défauts de paiement à l'aide de techniques de Machine Learning (ML).

L'objectif principal est d'analyser les données de crédit de clients, de construire des modèles prédictifs robustes et de comparer leurs performances afin d'identifier les clients susceptibles de présenter un risque de non-remboursement. Pour cela, le projet s'appuie sur un ensemble de données réel provenant d'une base de données publique bien connue : **UCI Credit Card Dataset**.

Ce projet suit une démarche complète comprenant :

- Le chargement et la préparation des données.
- Une analyse exploratoire approfondie (EDA) pour mieux comprendre les caractéristiques des clients et la distribution des défauts.
- L'équilibrage du jeu de données à l'aide de la technique **SMOTE**, afin de traiter le problème de déséquilibre des classes.
- La construction et l'entraînement de plusieurs modèles prédictifs, tels que : **Random Forest**, **Decision Tree**, **XGBoost**.
- L'évaluation des modèles selon plusieurs métriques de performance (accuracy, f1-score, matrice de confusion, etc.).
- La comparaison des résultats pour sélectionner les modèles les plus performants.

Explication des Modèles Utilisés

1. Decision Tree Classifier (Arbre de Décision)

Principe :

L'arbre de décision est un modèle de classification basé sur une structure hiérarchique en forme d'arbre. Chaque nœud interne représente une condition sur une caractéristique (feature), chaque branche un résultat possible, et chaque feuille une classe de sortie.

Son objectif est diviser l'espace de décision en segments homogènes selon les valeurs des variables d'entrée, pour prédire si un client va faire défaut ou non.

Fonctionnement :

- **Construction :**
  - À chaque nœud, le modèle choisit la **meilleure variable** et le **meilleur seuil** pour séparer les données selon un **critère d'impureté** (comme l'**indice de Gini** ou **l'entropie**).
  - Exemple : "Si PAY_1 > 2 → aller à droite, sinon à gauche".
- **Division récursive :**

- Le processus continue récursivement pour chaque sous-groupe jusqu'à ce qu'un critère d'arrêt soit atteint (profondeur max, nombre minimum d'échantillons, etc.).

- **Prédiction :**

  - Un échantillon suit un chemin depuis la racine jusqu'à une feuille, et la **classe majoritaire** dans cette feuille est la prédiction.

2. Random Forest Classifier

Principe :

Random Forest est un ensemble (bagging) de plusieurs arbres de décision, où chaque arbre est construit sur un sous-échantillon aléatoire du jeu de données et avec un sous-ensemble aléatoire de caractéristiques.

L'objectif est d'améliorer la robustesse et la précision des arbres de décision en utilisant plusieurs arbres indépendants entraînés sur des sous-échantillons.

Fonctionnement :

- **Bagging (Bootstrap Aggregating)** :

  - Le modèle crée **plusieurs sous-ensembles aléatoires** du jeu de données (avec remplacement) pour entraîner chaque arbre.
  - Cela réduit la variance du modèle global.

- **Sélection aléatoire de caractéristiques** :

  - À chaque division dans un arbre, un **sous-ensemble aléatoire de variables** est sélectionné.
  - **Cela rend les arbres** moins corrélés **entre eux.**

- **Vote majoritaire** :

  - Chaque arbre prédit indépendamment. Le résultat final est la **classe la plus votée** parmi tous les arbres.

3. XGBoost (Extreme Gradient Boosting)

Principe :

XGBoost est un algorithme de **boosting**, qui construit les arbres de décision de façon séquentielle, où chaque nouvel arbre tente de corriger les erreurs faites par les précédents. Il utilise une fonction de perte optimisée à chaque étape.

L'objectif est Construire des modèles très performants en combinant des **petits arbres faibles** (stumps) ajoutés séquentiellement, en minimisant les erreurs précédentes.

Fonctionnement :

- **Boosting :**
  - Contrairement à Random Forest (parallèle), XGBoost **ajoute un arbre à la fois**, où chaque nouvel arbre apprend à **corriger les erreurs** des arbres précédents.
- **Optimisation de la fonction de perte** :
  - À chaque itération, XGBoost cherche à **minimiser une fonction de coût** (log-loss ici) via **le gradient du résidu** (d'où "Gradient Boosting").
- **Régularisation L1 & L2** :
  - XGBoost pénalise les arbres trop complexes, ce qui **évite l'overfitting**.
- **Importance des variables** :
  - XGBoost fournit une mesure fine de l'impact de chaque variable dans les prédictions.

Fonctionnement du Projet :

1. Collecte des données

J'ai télécharger le fichier des données depuis kaggle.

```
# Chargement des données
df = pd.read_csv("../data/UCI_Credit_Card.csv")
```

Forme du dataset : (30000, 25)

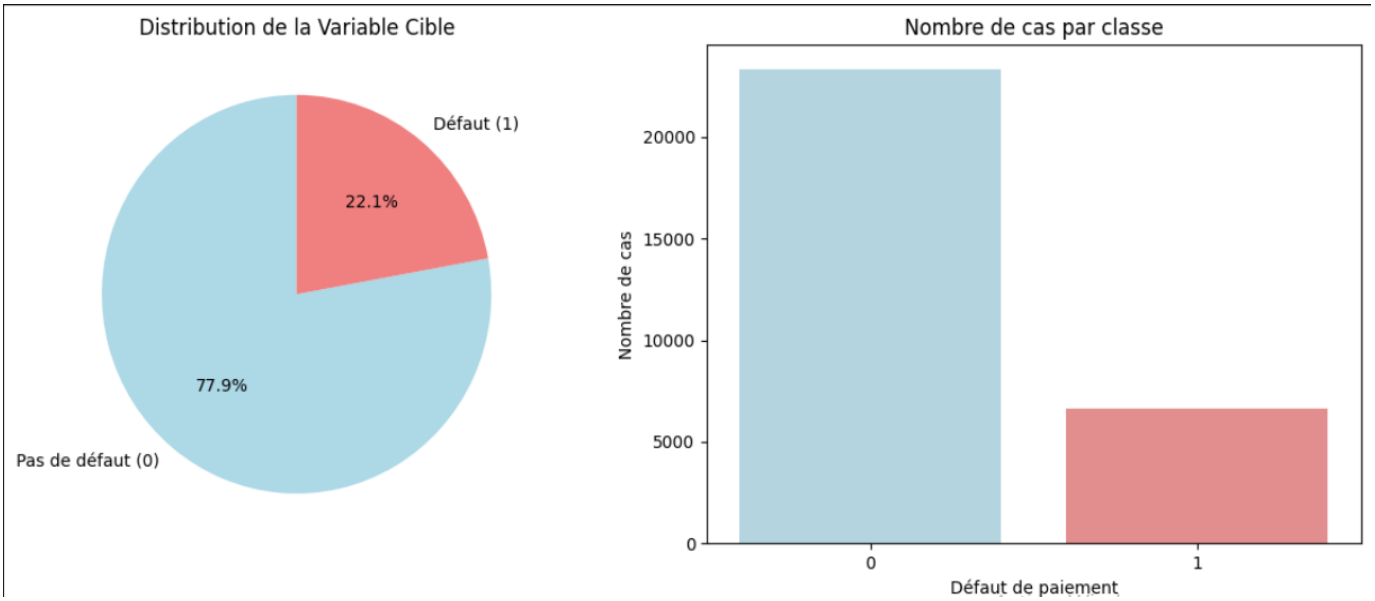
Taille du dataset : 30000 lignes, 25 colonnes

```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  PAY_4 \
0   1    20000.0    2          2         1   24      2      2     -1     -1
1   2   120000.0    2          2         2   26     -1      2      0      0
2   3    90000.0    2          2         2   34      0      0      0      0
3   4    50000.0    2          2         1   37      0      0      0      0
4   5    50000.0    1          2         1   57     -1      0     -1      0

   ...  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3 \
0  ...        0.0        0.0        0.0       0.0     689.0       0.0
1  ...     3272.0     3455.0     3261.0       0.0    1000.0    1000.0
2  ...    14331.0    14948.0    15549.0    1518.0    1500.0    1000.0
3  ...    28314.0    28959.0    29547.0    2000.0    2019.0    1200.0
4  ...    20940.0    19146.0    19131.0    2000.0   36681.0   10000.0

   PAY_AMT4  PAY_AMT5  PAY_AMT6  default.payment.next.month
0       0.0       0.0       0.0                          1
1    1000.0       0.0    2000.0                          1
2    1000.0    1000.0    5000.0                          0
3    1100.0    1069.0    1000.0                          0
4    9000.0     689.0     679.0                          0

[5 rows x 25 columns]
```

2. **Analyse exploratoire de données** (EDA Exploratory Data Analysis)

Répartition des classes :

Classe 0 (Pas de défaut) : 23364 (77.9%)
Classe 1 (Défaut) : 6636 (22.1%)
Le dataset est **déséquilibré** (il y a beaucoup plus de bons payeurs que de mauvais).



3. **Prétraitement des données**

```
# Nettoyage
df['EDUCATION'] = df['EDUCATION'].replace({0: 4, 5: 4, 6: 4})
df['MARRIAGE'] = df['MARRIAGE'].replace({0: 3})
df['avg_bill'] = df[[f'BILL_AMT{i}' for i in
range(1,7)]].mean(axis=1)
df['avg_pay'] = df[[f'PAY_AMT{i}' for i in
range(1,7)]].mean(axis=1)
df.drop(columns=['ID'] + [f'BILL_AMT{i}' for i in range(1, 7)] +
[f'PAY_AMT{i}' for i in range(1, 7)], inplace=True)
```

- Certaines valeurs dans les colonnes EDUCATION et MARRIAGE n'ont pas de signification claire (par exemple 0, 5 ou 6).
- On les regroupe dans une nouvelle catégorie "autres" pour simplifier.
- Chaque client a 6 mois d'historique de factures (BILL_AMT) et de paiements (PAY_AMT).
- On calcule la moyenne de ces 6 mois pour chaque client, ce qui donne deux nouvelles colonnes :
  - avg_bill : montant moyen facturé.
  - avg_pay : montant moyen payé.

- Ensuite, on supprime les anciennes colonnes devenues inutiles.
- La colonne ID est juste un identifiant, donc elle ne sert pas pour la prédiction → on la supprime.

```
# Encodage

df = pd.get_dummies(df, columns=['SEX', 'EDUCATION', 'MARRIAGE'])
```

• Les colonnes comme SEX, EDUCATION et MARRIAGE contiennent des textes ou chiffres représentant des catégories.

• On les transforme en plusieurs colonnes avec des 0 et 1 (c'est ce qu'on appelle **"one-hot encoding"**) pour que les modèles comprennent mieux.

```
# Features et target

X = df.drop('default.payment.next.month', axis=1)
y = df['default.payment.next.month']
```

• X contient les **informations sur les clients** (sexe, âge, factures, etc.).

• y contient la **variable à prédire** : est-ce que ce client va faire défaut ou pas ?

```
# Équilibrage avec SMOTE

X_res, y_res = SMOTE (random_state=42).fit_resample(X, y)
```

• Le dataset est **déséquilibré** (il y a beaucoup plus de bons payeurs que de mauvais).

• SMOTE est une technique qui **ajoute artificiellement des clients en défaut**, pour que le modèle apprenne mieux à les reconnaître.

4. Division des données

```
X_train, X_test, y_train, y_test = train_test_split(X_res_scaled,
y_res, stratify=y_res, test_size=0.25, random_state=42)
```

Séparer les données en un **ensemble d'entraînement** et un **ensemble de test**, pour pouvoir entraîner les modèles sur une partie des données, puis les tester sur des données jamais vues.

25% des données sont mises de côté pour le test, les **75% restantes** servent à l'entraînement.

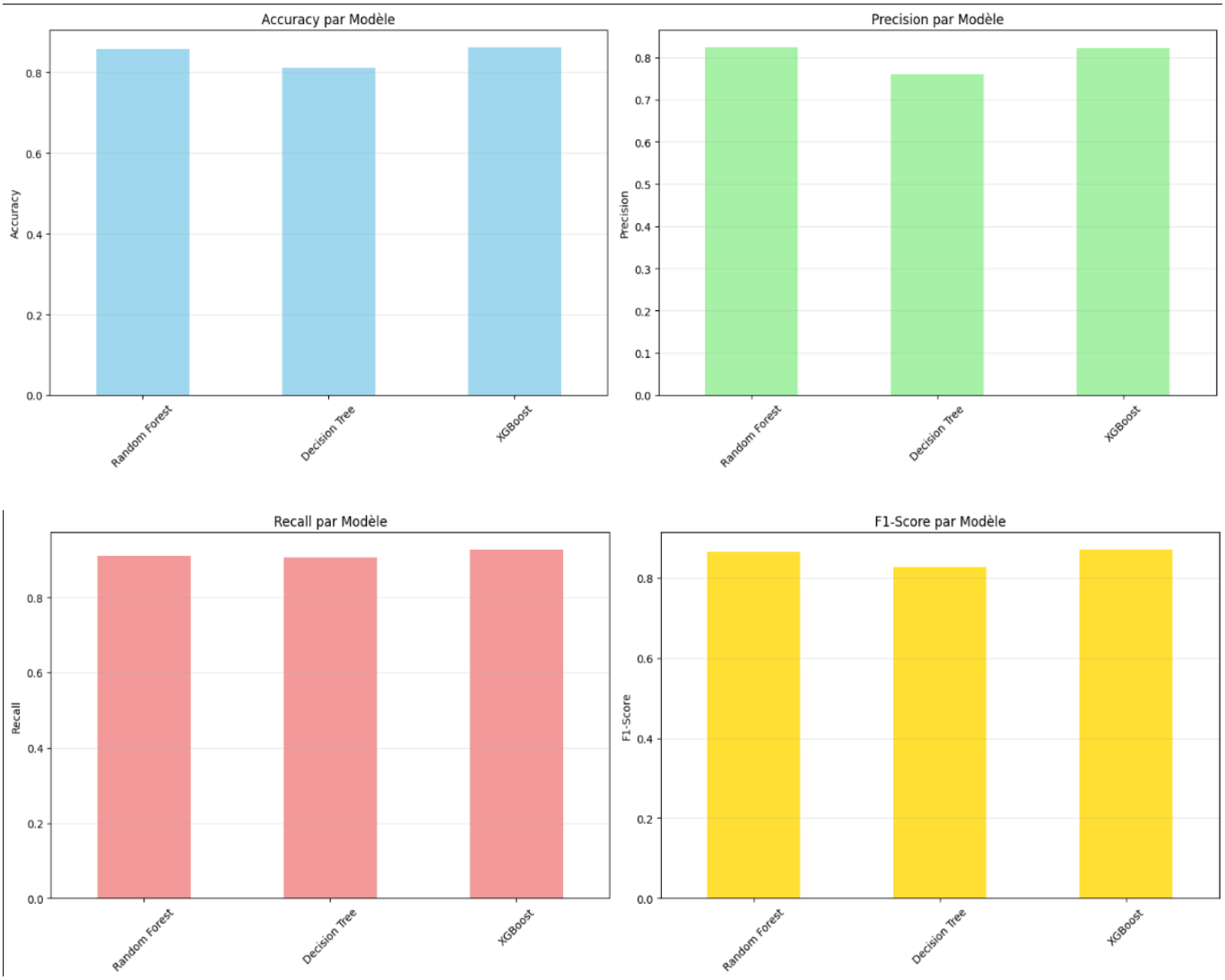
5. Création des modèles

```
# Modèles
models = {
```

```
    'Random Forest': RandomForestClassifier(n_estimators=150,
max_depth=10,random_state=42),

    'Decision Tree': DecisionTreeClassifier(max_depth=6,
random_state=42),

    'XGBoost': XGBClassifier(use_label_encoder=False,
eval_metric="logloss", random_state=42)
}
```

6. Evaluation des modèles et comparaison de plusieurs modèles avec interprétation des résultats

Visualisation des performances



Matrice de Confusion

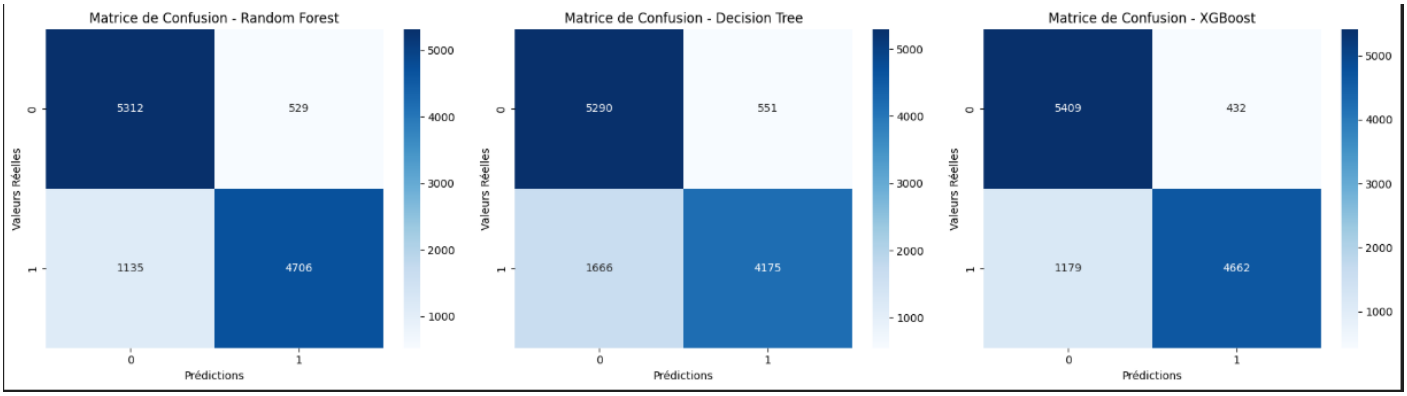


Tableau récapitulatif :

Random Forest

	precision	recall	f1-score	support
0	0.82	0.91	0.86	5841
1	0.90	0.81	0.85	5841
accuracy			0.86	11682
macro avg	0.86	0.86	0.86	11682
weighted avg	0.86	0.86	0.86	11682

Decision Tree

	precision	recall	f1-score	support
0	0.76	0.91	0.83	5841
1	0.88	0.71	0.79	5841
accuracy			0.81	11682
macro avg	0.82	0.81	0.81	11682
weighted avg	0.82	0.81	0.81	11682

XGBoost

	precision	recall	f1-score	support
0	0.82	0.93	0.87	5841
1	0.92	0.80	0.85	5841
accuracy			0.86	11682
macro avg	0.87	0.86	0.86	11682
weighted avg	0.87	0.86	0.86	11682

XGBoost :

- **Meilleur F1-Score** (0.8704) → équilibre parfait entre détection (recall) et fausses alertes (precision).
- **Meilleur Recall** (0.926) → il détecte le plus grand nombre de vrais défauts.
- **Très bon Accuracy et Precision** également.

Random Forest:

- Très bon **F1-Score** (0.8646) et **Recall** (0.9094), proche de XGBoost.
- Légèrement **meilleure Precision** que XGBoost (moins de fausses alertes).

Decision Tree :

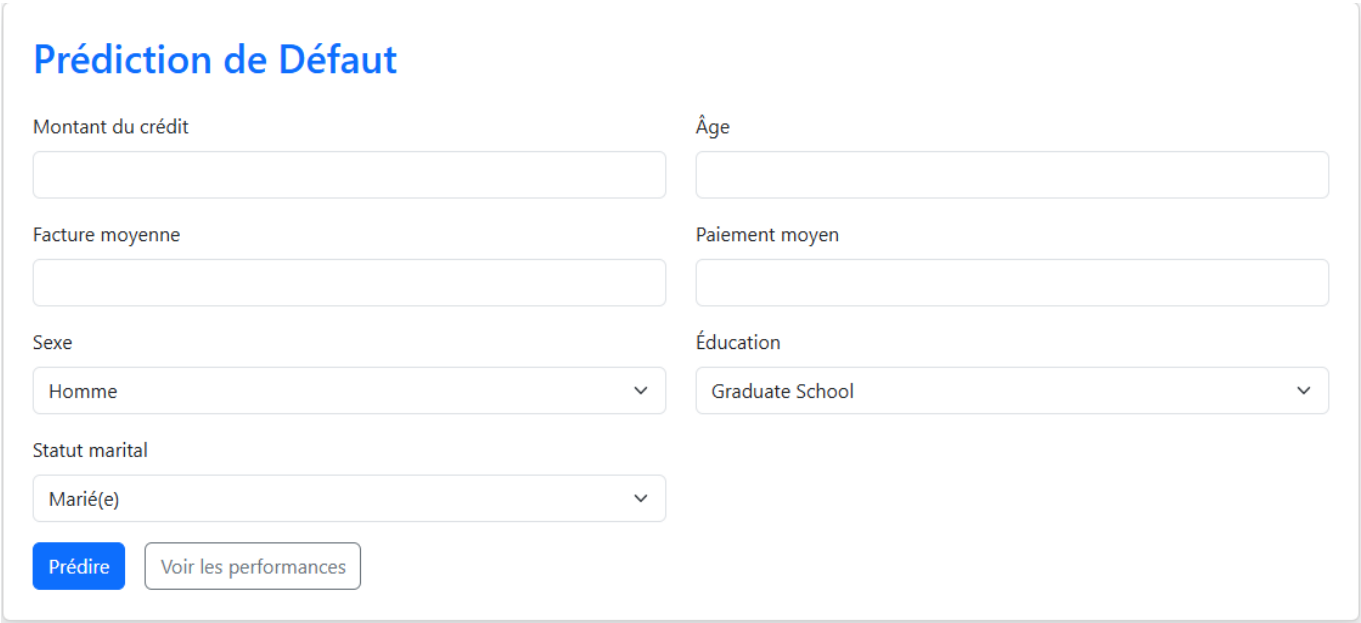
- Moins performant sur tous les indicateurs.
- Son principal avantage reste la **simplicité et l'interprétabilité**, mais il fait **plus d'erreurs** que les autres.

7. Choix du modèle qui a donné les meilleurs résultats

XGBoost est le modèle recommandé **pour ce projet, car il combine** haute précision**,** grande capacité de détection des clients à risque**, et** fiabilité générale **sur les données financières.**

8. Déploiement du modèle avec une petite app

Interface de Prédiction de Défaut



Cette interface web a été conçue pour permettre à un utilisateur (analyste de crédit, banquier, etc.) de **prédire si un client est susceptible de faire défaut sur un crédit** bancaire.

Objectif de l'interface

Permettre à l'utilisateur de **saisir les informations clés** d'un client et d'obtenir une **prédiction automatique** sur son risque de défaut, à l'aide d'un modèle d'apprentissage automatique (comme XGBoost).

• Bouton **"Prédire"** :

- Lorsque l'utilisateur clique dessus, les données saisies sont envoyées au modèle prédictif.
- Le système affiche alors si le client est **à risque de défaut** ou **non**.

• Bouton **"Voir les performances"** :

- Permet à l'utilisateur de consulter les **métriques d'évaluation du modèle** utilisé (accuracy, recall, etc.), pour comprendre la fiabilité de la prédiction.

Exemple de test