Détection de fraudes bancaires

# 1. Contexte du projet

Ce projet vise à développer un modèle de détection de fraudes à partir d’un jeu de données issu de transactions bancaires réelles.  
Les objectifs sont les suivants :

* Identifier les transactions potentiellement frauduleuses avec un bon niveau de précision
* Proposer une solution industrialisable pouvant être intégrée dans un pipeline métier ou technique

## Jeu de données :

* Source : Kaggle – Credit Card Fraud Detection
* 284 807 transactions, dont 492 fraudes (soit environ 0.17 %)
* Données prétraitées (anonymisées via PCA)

# 2. Exploration des données (EDA)

Le dataset est fortement déséquilibré, ce qui rend le problème de classification difficile.  
Les premières analyses montrent que :

* Les fraudes sont souvent associées à des **montants faibles**
* On observe une **répartition temporelle non uniforme** des fraudes (pics nocturnes possibles)
* Certaines variables (ex : V14, V17…) sont **fortement corrélées à la classe**

**Répartition des classes (fraude vs non-fraude)**  
A graph of a class

AI-generated content may be incorrect.**Distribution du montant :**

A screen shot of a white screen

AI-generated content may be incorrect.A graph of a person with a red line

AI-generated content may be incorrect.

**Répartition des transactions par heure :**

A blue graph with black lines

AI-generated content may be incorrect.

A graph of a number of red bars

AI-generated content may be incorrect.

# 3. Préparation des données (feature engineering)

* **Standardisation du montant** (Amount\_Scaled)
* **Extraction de l’heure dans la journée** à partir de la variable Time (Hour\_Scaled)
* **Sélection automatique des variables** par importance ou corrélation avec la fraude

Les variables sont prêtes à être injectées dans des modèles supervisés ou non-supervisés.

# 4. Modélisation et résultats

Le modèle utilisé est un **Random Forest** pondéré pour compenser le déséquilibre de classes.  
Les performances sont évaluées sur un échantillon test via les métriques standard de classification (fournies par sklearn).

**Matrice de confusion :**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Classe prédite** | |
|  |  | **Non-fraude** | **Fraude** |
| **Classe réelle** | **Non-fraude** | **56860** | **4** |
| **Fraude** | **24** | **74** |

**Rapport de classification**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Precision** | **recall** | **F1-score** | **support** |
| **Non-fraude** | **0,9996** | **0,9999** | **0,9998** | **56864** |
| **Fraude** | **0,9487** | **0,7551** | **0,8409** | **98** |

**Courbe ROC**

A graph with a line

AI-generated content may be incorrect.

# 5. Recommandations métier

* Le modèle peut être intégré dans un système de scoring automatique, avec un seuil ajustable (ex : score > 0.85) pour déclencher des alertes.
* Dans des cas tangents plus sensibles, une doublé vérification humaine est recommandée.
* Si l’on a très peu de faux positifs, leur impact doit tout de même être pris en compte

# 6. Perspectives

* On peut tester des modèles plus avancés (XGBoost, LightGBM...)
* On peut créer un dashboard de monitoring en temps réel

# Annexe

Exemple de test Postman pour l’image Docker :

{

  "Time": 3,

  "V1": -1.3,

  "V2": -0.07,

  "V3": 2.53,

  "V4": 1.37,

  "V5": -0.33,

  "V6": 0.46,

  "V7": 0.23,

  "V8": 0.09,

  "V9": 0.36,

  "V10": 0.09,

  "V11": -0.55,

  "V12": -0.61,

  "V13": -0.99,

  "V14": -0.31,

  "V15": 1.46,

  "V16": -0.47,

  "V17": 0.20,

  "V18": 0.02,

  "V19": 0.40,

  "V20": 0.25,

  "V21": -0.01,

  "V22": 0.27,

  "V23": -0.11,

  "V24": 0.06,

  "V25": 0.12,

  "V26": -0.18,

  "V27": 0.13,

  "V28": -0.02,

  "Amount": 20000,

  "Hour": 10,

  "Amount\_Scaled": 0.24,

  "Hour\_Scaled": 0.38

}