

# Implémentation d'un modèle de scoring



**Prêt à dépenser**

---

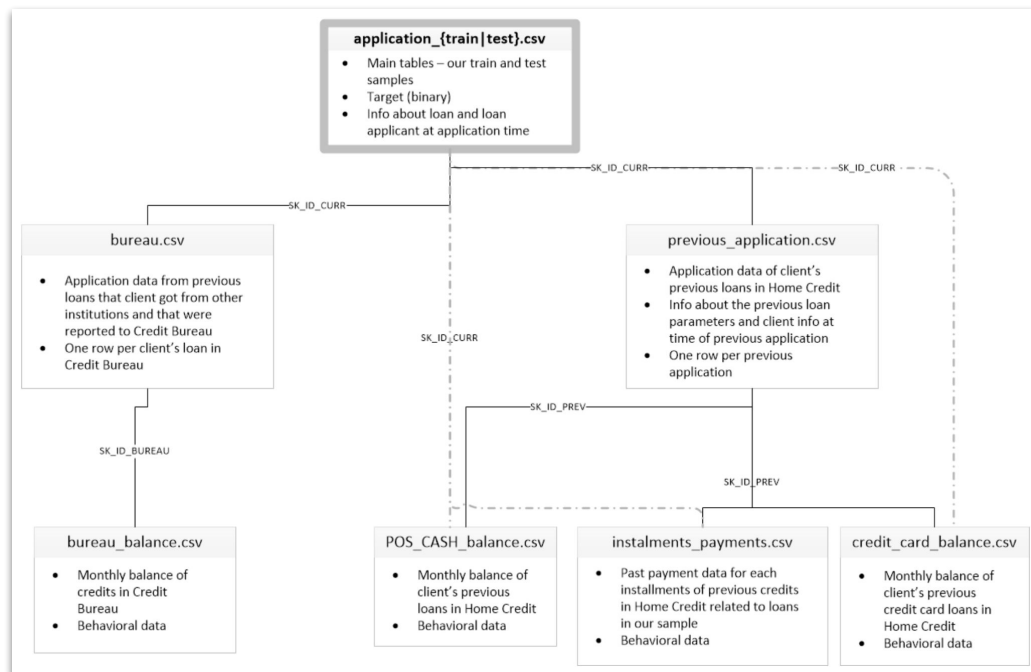
*octobre 2023*

# SOMMAIRE

- Problématiques & objectifs
- Présentation du jeu de données
- Modélisation
  - préparation des données
  - démarche de modélisation
  - métriques d'évaluation & résultats
  - optimisation du seuil
- Suivi des modèles et de l'évolution des données
  - le tracking des modèles avec MLFlow
  - l'analyse du data drift
- Le déploiement
  - le pipeline de déploiement
  - le dashboard

# PROBLÉMATIQUES & OBJECTIFS

- Déterminer la solvabilité de clients ayant peu ou pas d'historique de prêt
  - ➔ Développer un algorithme de scoring crédit
- Répondre au besoin de transparence des clients quant aux décisions d'octroi de crédit
  - ➔ Développer un dashboard interactif qui explique ces décisions

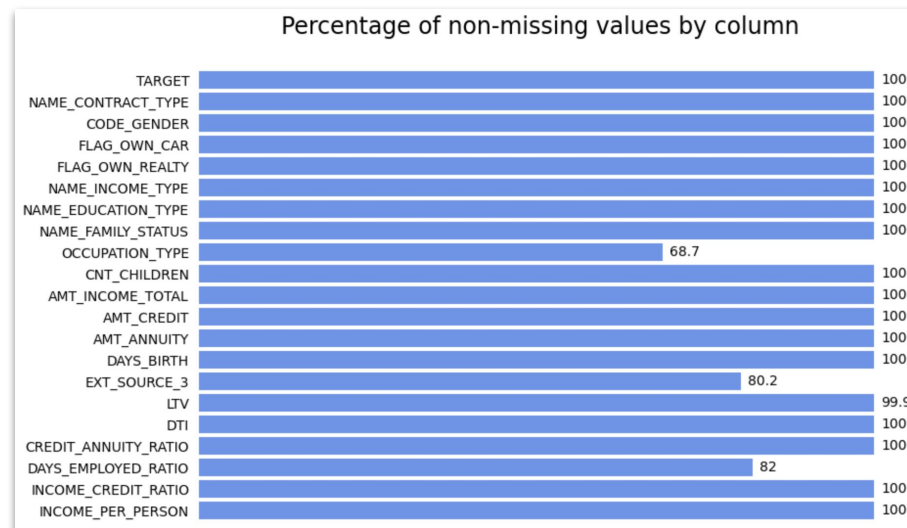


Organisation du jeu de données

→ on se concentre sur la table principale pour s'adapter à des clients ayant pas ou peu d'historique de prêt

→ train set : 307511 prêts, 120 features  
*sans doublons dans les ID*  
*24 % de valeurs manquantes*

- reprise d'éléments d'un kernel Kaggle  
*kernel 'LightGBM with Simple Features'*
- nettoyage
  - *suppression d'outliers extrêmes*
  - *suppression de catégories trop peu représentées*
- feature engineering
  - $LTV (LoanToValue) = AMT\_CREDIT / AMT\_GOODS\_PRICE$
  - $DTI = AMT\_ANNUITY / AMT\_INCOME\_TOTAL$
  - $CREDIT\_ANNUITY\_RATIO = AMT\_CREDIT / AMT\_ANNUITY$
  - $DAYS\_EMPLOYED\_RATIO = DAYS\_EMPLOYED / DAYS\_BIRTH$
  - $INCOME\_CREDIT\_RATIO = AMT\_INCOME\_TOTAL / AMT\_CREDIT$
  - $INCOME\_PER\_PERSON = AMT\_INCOME\_TOTAL / CNT\_FAM\_MEMBERS$
- suppression des variables avec trop de valeurs manquantes

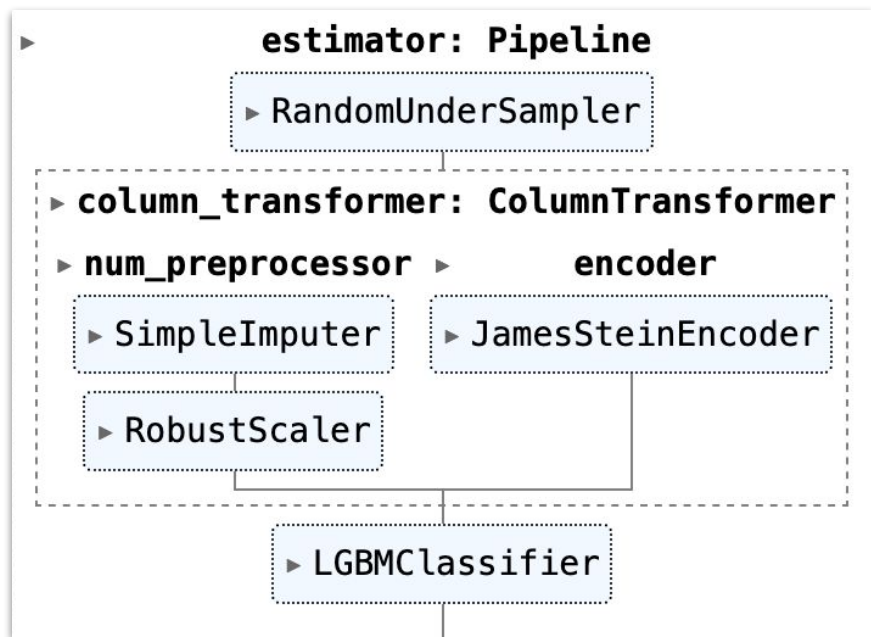


*les variables retenues et leurs pourcentages de valeurs non-manquantes*



- seulement 8,1 % de positifs
- déséquilibre qui peut nuire aux performances
- piste privilégiée : le sous-échantillonnage  
*le jeu de données est assez grand pour éviter le suréchantillonnage*

une grid search pour chaque étape du pipeline :



structure du pipeline retenu

- rééquilibrage des données :  
*test avec / sans sous-échantillonnage (RandomUnderSampler)*
- imputer  
*test de SimpleImputer (médiane et moyenne), IterativeImputer, KNNImputer*
- scaler  
*test de StandardScaler, RobustScaler, Normalizer, PowerTransformer, QuantileTransformer, MinMaxScaler*
- encoder  
*test de OneHotEncoder, WOEEncoder, GLMMEncoder, JamesSteinEncoder*
- classifieur  
*LogisticRegression, RandomForestClassifier, LGBMClassifier, GradientBoostingClassifier (+ DummyClassifier as reference)*
- hyperparamètres  
*n\_estimators : [100, 200, 300, 400, 500],  
learning\_rate : [0.15, 0.1, 0.05],  
max\_depth : [2, 3, 4, 5]*

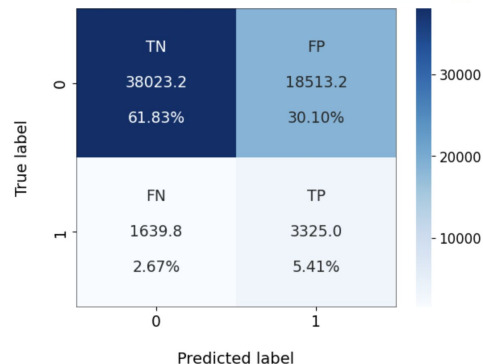
- hypothèse : les faux négatifs sont 10x plus coûteux que les faux positifs
- création de 3 métriques :
  - une fonction de perte qui compte 1 pour un faux positif, et 10 pour un faux négatif
  - F10 score : moyenne harmonique pondérée de la précision (precision) et du rappel (recall) avec 10x plus d'importance pour le rappel que pour la précision
  - P&L : différence des profits (hypothèse : les intérêts s'élèvent à 5 % du montant du prêt) et des pertes (hypothèse : les prêts non-remboursés génèrent une perte égale à 50 % du montant du prêt)
- + recall, precision, accuracy, AUC, negative log loss, fit time, score time



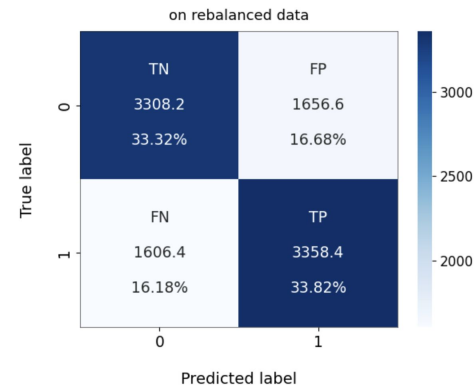
	LGBM	DummyClassifier
mean_test_roc_auc	0.73	0.5
mean_test_accuracy	0.67	0.92
mean_test_recall	0.67	0
mean_test_precision	0.15	0
mean_test_f10	0.65	0
mean_test_P&L	6.85565e+08	3.18957e+08
mean_test_custom_loss	34845.2	49648
mean_fit_time	5.4	0.35
std_fit_time	0.71	0.02
mean_score_time	1.3	0.51
std_score_time	0.32	0.01

*Les performances du modèle retenu comparées à celles du classifieur idiot (cross-validation 5 splits)*

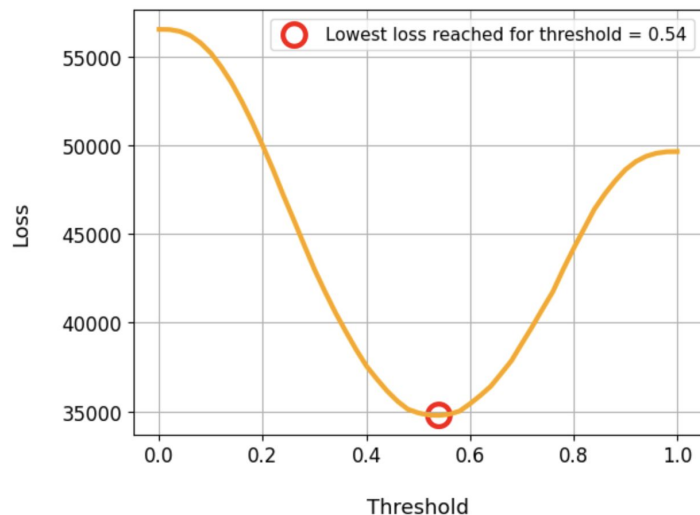
Confusion matrix with the average results over all CV test sets



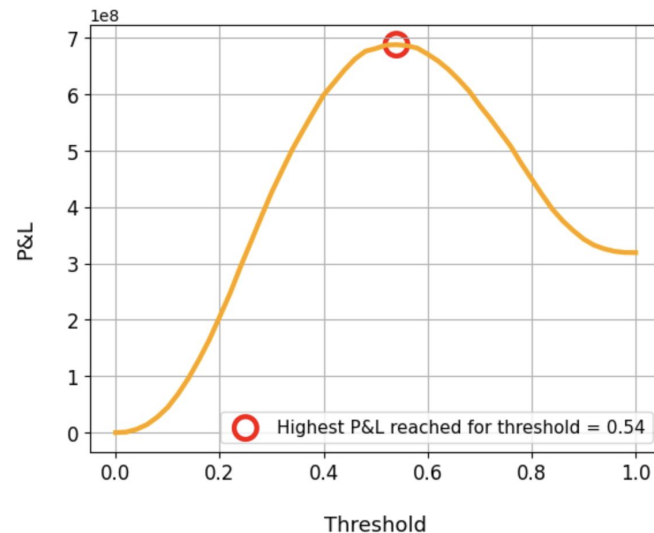
Confusion matrix with the average results over all CV test sets



Loss by threshold value

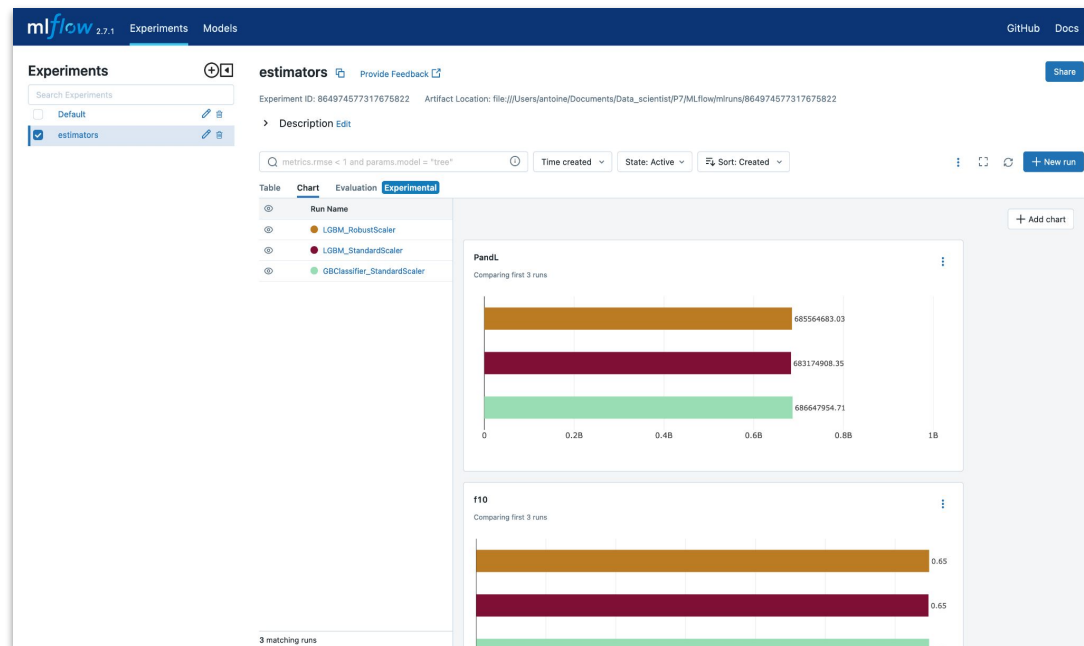


P&L by threshold value



- mesures effectuées en cross-validation
- même optimum trouvé avec les deux métriques (leurs hypothèses sont très proches)
- avec ce seuil, 70 % des demandes sont accordées

- s'inspirer d'autres kernels Kaggle
- revenir sur la sélection des features
- pour l'optimisation du seuil : améliorer les fonctions coût en affinant leurs hypothèses (coûts relatifs d'un faux positif et d'un faux négatif, etc.)
- faire plus de tests d'hyperparamètres
- réviser l'algorithme pour qu'il puisse prendre en compte l'historique de prêt du client quand il y en a un



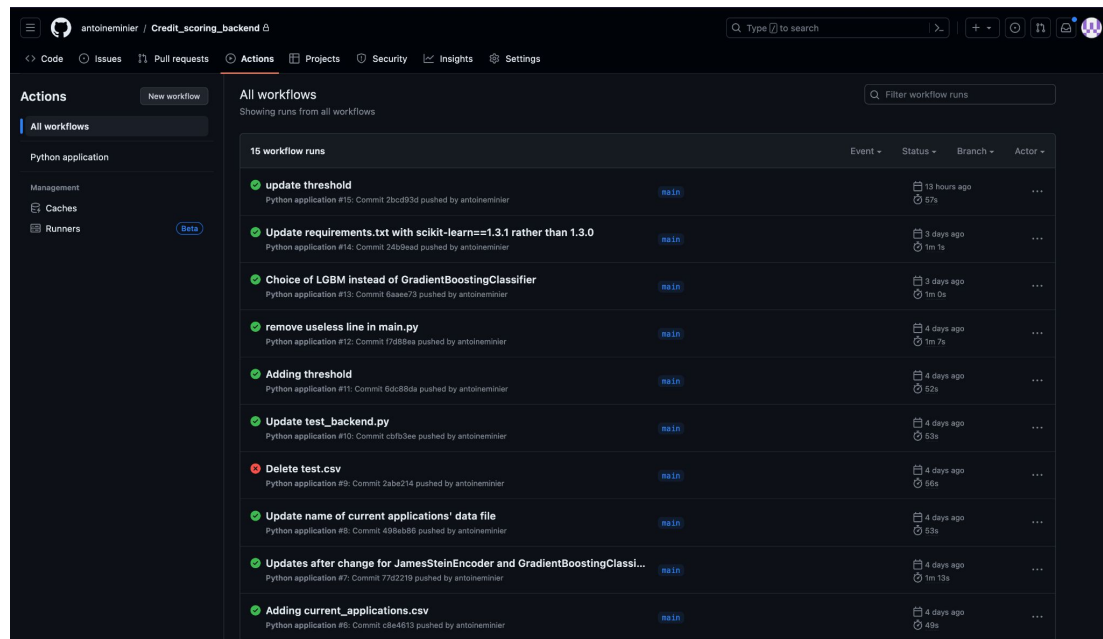
→ suivi de l'évolution des modèles et de leurs résultats

→ les prérequis au déploiement de chaque modèle sont enregistrés

*Les meilleurs modèles et leurs résultats enregistrés sur MLFlow*

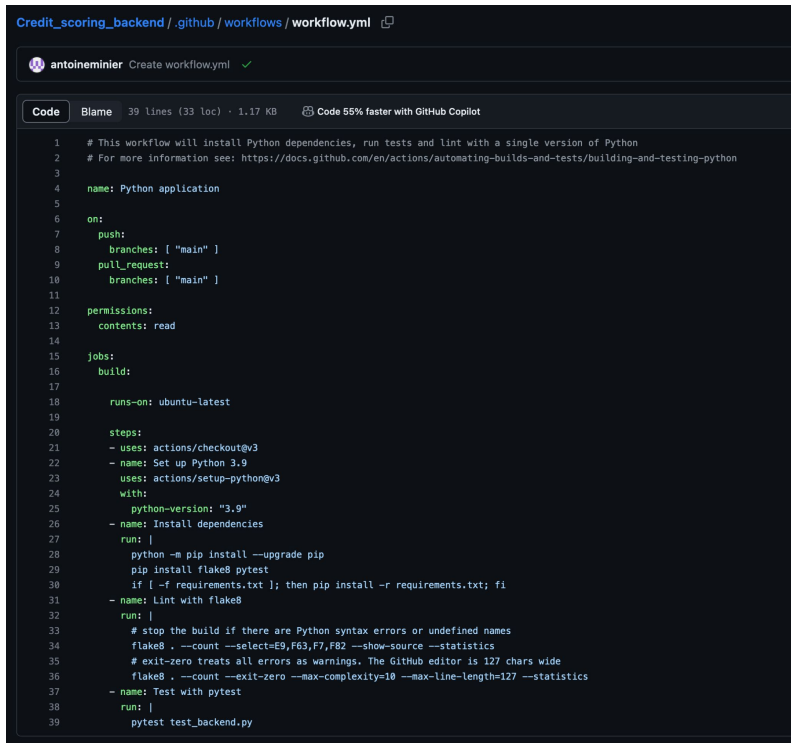


- data drift détecté sur 5 des 20 variables
- chute du taux de crédits renouvelables (revolving loans) par rapport aux prêts personnels (cash loans)
  - ratio montant du prêt/mensualités en hausse
  - augmentation des montant des prêts
  - augmentation des mensualités
- réentraîner le modèle au fur et à mesure de la mise à jour des données, puis reconstruire le modèle si le data drift augmente trop



*suivi des différents push vers le repository Github de la partie backend*

- 2 repositories Github  
pour le backend (utilisation de FastAPI)  
[https://github.com/antoine-minier/Credit\\_scoring\\_backend](https://github.com/antoine-minier/Credit_scoring_backend)  
pour le frontend (dashboard Streamlit)  
[https://github.com/antoine-minier/Credit\\_scoring\\_frontend](https://github.com/antoine-minier/Credit_scoring_frontend)
- Render & Streamlit exécutent  
les fichiers déposés sur Github
- Streamlit envoie ses requêtes à  
Render pour récupérer les  
résultats du backend



```
1 # This workflow will install Python dependencies, run tests and lint with a single version of Python
2 # For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python
3
4 name: Python application
5
6 on:
7   push:
8     branches: [ "main" ]
9   pull_request:
10    branches: [ "main" ]
11
12 permissions:
13   contents: read
14
15 jobs:
16   build:
17
18     runs-on: ubuntu-latest
19
20     steps:
21     - uses: actions/checkout@v3
22     - name: Set up Python 3.9
23       uses: actions/setup-python@v3
24       with:
25         python-version: "3.9"
26     - name: Install dependencies
27       run: |
28         python -m pip install --upgrade pip
29         pip install flake8 pytest
30         if [ -f requirements.txt ]; then pip install -r requirements.txt; fi
31     - name: Lint with flake8
32       run: |
33         # stop the build if there are Python syntax errors or undefined names
34         flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics
35         # exit-zero treats all errors as warnings. The GitHub editor is 127 chars wide
36         flake8 . --count --exit-zero --max-complexity=10 --max-line-length=127 --statistics
37     - name: Test with pytest
38       run: |
39         pytest test_backend.py
```

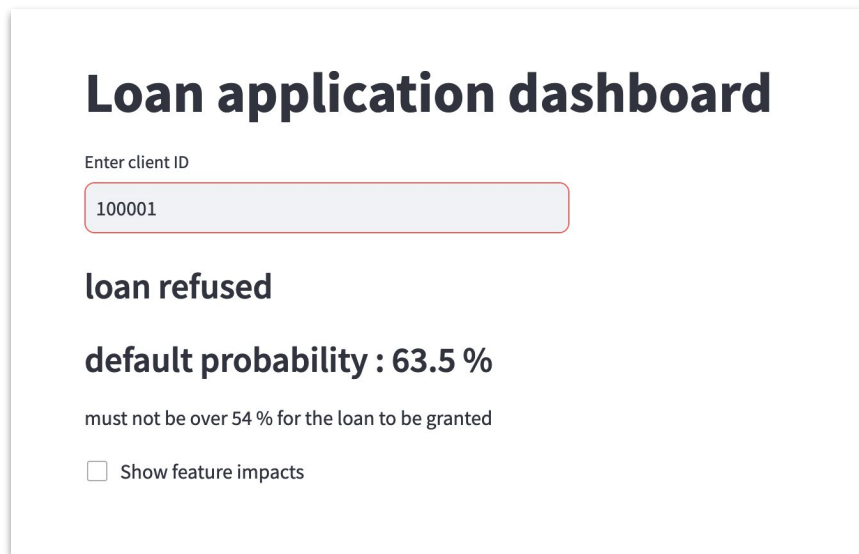
→ test de la syntaxe et vérification qu'il n'y a pas de variables non-définies

→ vérifications sur les fichiers :

- des data
- du préprocesseur des données
- du classifieur
- de l'explainer SHAP

*Un des fichiers qui détermine les tests à effectuer avant chaque push & merge*

lien vers le dashboard : <https://creditscoringfrontend-kapprmphrvmeskhibw7txor.streamlit.app/>



**Loan application dashboard**

Enter client ID

100001

**loan refused**

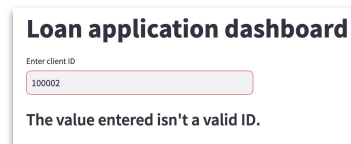
**default probability : 63.5 %**

must not be over 54 % for the loan to be granted

☐ Show feature impacts

*La première étape dans l'utilisation du dashboard*

- 1<sup>ère</sup> étape : rentrer l'identifiant du client
- la décision d'octroi de crédit s'affiche, avec :
  - la probabilité de défaut
  - l'indication du seuil critique
- message d'erreur si l'id n'est pas correct :

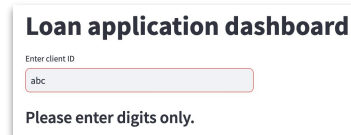


**Loan application dashboard**

Enter client ID

100002

The value entered isn't a valid ID.



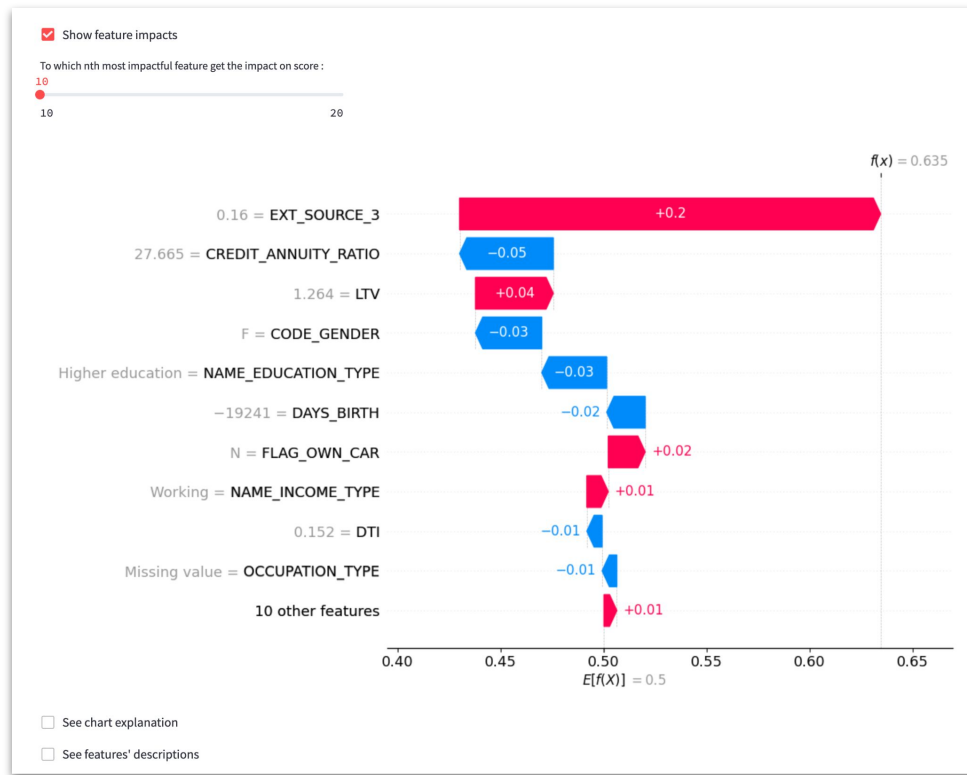
**Loan application dashboard**

Enter client ID

abc

Please enter digits only.





La deuxième étape dans l'utilisation du dashboard

→ 2<sup>ème</sup> étape : l'explication du résultat

cocher la case "Show feature impacts"

→ graphique "waterfall"

→ indique de combien chaque variable a augmenté ou diminué la probabilité de défaut de paiement

→ part de la valeur de base et remonte à travers les contributions jusqu'à la probabilité de défaut  $f(x)$

→ les données du client s'affichent à droite des noms de variables sur l'axe des ordonnées

→ possibilité d'afficher +/- de variables sur le graph

→ deux cases à cocher pour obtenir plus d'infos

→ une explication du sens du graphique

→ une définition des variables

- 3<sup>ème</sup> étape : la comparaison du client avec les autres clients  
déterminer avec la glissière sur jusqu'à quelle énième variable qui a eu le plus d'impact l'on veut afficher des informations

- pour les variables catégorielles : est indiqué pour chaque catégorie le nombre de clients dont le prêt a été accordé / refusé

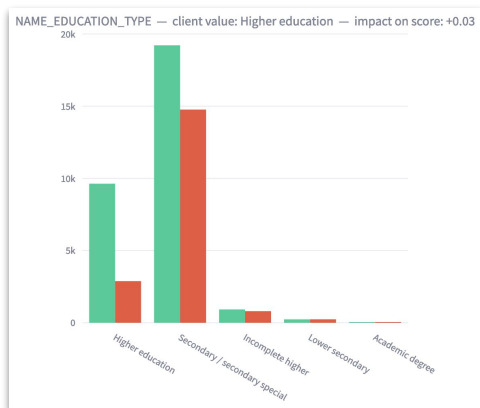
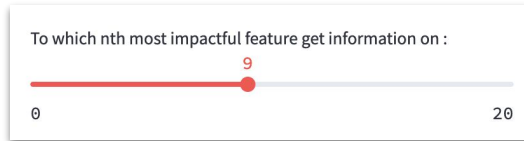


diagramme pour une variable catégorielle



la glissière pour régler le nombre de graphiques à afficher

- pour les variables numériques on compare :
- la valeur du client
  - la moyenne pour les clients dont le prêt a été refusé
  - la moyenne pour les clients dont le prêt a été accordé
  - la moyenne générale

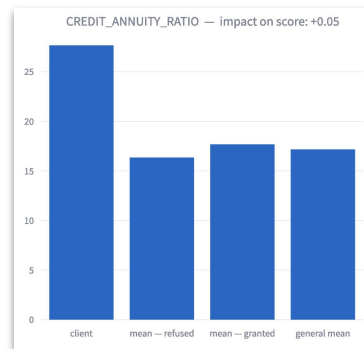


diagramme pour une variable numérique

# Conclusion

- le modèle de scoring est déployé, avec suivi de son évolution, suivi de l'évolution des données, et tests automatiques pour sécuriser ses futures modifications
- le chargé de relation client peut expliquer la décision prise par l'algorithme et peut comparer le client aux autres clients
- des axes d'amélioration du modèle sont identifiés