

PRÉDICTION D'ADMISSIBILITÉ A UN PRÊT

Github Repo :

https://github.com/Pimouss75/Projet_7

Adresse :

<http://35.180.189.117:8501/>

Exemple de clients :

	100013
	456250
	456222



Marc Sellam

Projet 7

Data-Scientist

Mars 2023

Sommaire :

Mission	P3
Préparation des données	P4
Elaboration d'un modèle d'apprentissage	P7
Choix du modèle, optimisation, calcul du seuil optimal	P12
Interprétabilité du modèle	P14
Performance du modèle	P16
Création de l'API et du Dashboard	P17
Déploiement du Dashboard	P19
Le Projet sur GitHub	P23
Conclusion	P24





Mission

Création d'un Dashboard ayant pour objectif la prédiction d'obtention ou non d'un emprunt pour les client disponibles dans un fichier de test :

Trois cases sont proposées:

- "Prédiction" : réponse a la demande apres renseignement du numero client, et affichage d'informations relatives a ce client et sa position par rapport aux autres.
- "Interprétation du client" : réponse à la demande du client renseigné + graphiques affichant les variables qui on favorisés l'accessibilité à l'emprunt et celles qui ne l ont pas favorisé dans leurs proportions respectives.
- "Comparaison avec les autres clients" : réponse a la demande du client renseigné + graphiques du client + graphique des variables qui influencent tous les clients de test par ordre décroissants.

Préparation des données

Les données fournies se composent de **10 fichiers au format csv** :

<https://www.kaggle.com/c/home-credit-default-risk/data>

Le **feature engineering** effectué avec le **kernel kaggle** :

<https://www.kaggle.com/code/jsaguiar/lightgbm-with-simple-features/script>

Dataset : 307507 clients dont nous avons la réponse a la demande de crédit

['TRAGET'] 0 pour « Accepté » ou 1 pour « Refusé ».

et 48744 clients dont nous ne le savons pas.

['TRAGET'] est vide

Nous disposons de 797 variables numériques par client, générés par le Kernel Kaggle.

Préparation des données

Nettoyage des données :

- Les variables avec plus de 30% de valeurs manquantes.
- Les variables dont la corrélations est supérieure à 0.5.

Sélection des variables :

- Sélection de 20 variables avec SelectKbest

(sur un dataset temporaire imputé avec KNNImputer(n_neighbors=5)) :

```
['CODE_GENDER', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_ID_PUBLISH', 'REGION_RATING_CLIENT', 'REG_CITY_NOT_LIVE_CITY',  
'REG_CITY_NOT_WORK_CITY', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_LAST_PHONE_CHANGE', 'NAME_INCOME_TYPE_Working',  
'NAME_EDUCATION_TYPE_Higher education', 'NAME_EDUCATION_TYPE_Secondary / secondary special', 'DAYS_EMPLOYED_PERC',  
'BURO_DAYS_CREDIT_MIN', 'BURO_CREDIT_ACTIVE_Closed_MEAN', 'PREV_DAYS_DECISION_MIN',  
'PREV_NAME_CONTRACT_STATUS_Refused_MEAN', 'PREV_CODE_REJECT_REASON_SCOFR_MEAN', 'PREV_NAME_PRODUCT_TYPE_walk-  
in_MEAN']
```

Préparation des données

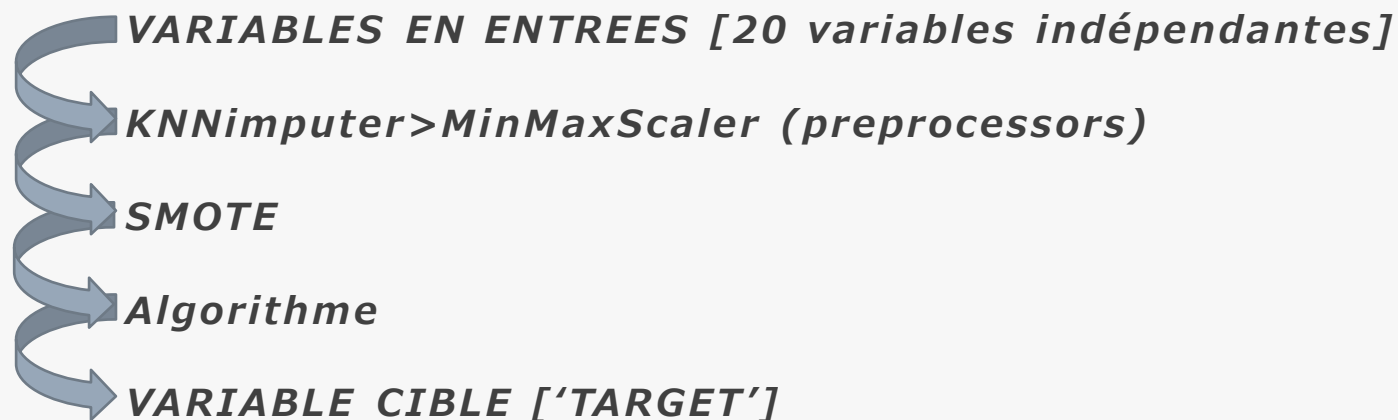
Description des variables

	Description
REG_CITY_NOT_WORK_CITY	Flag if client's permanent address does not match work address (1=different, 0=same, at city level)
REGION_RATING_CLIENT	Our rating of the region where client lives (1,2,3)
DAYS_ID_PUBLISH	How many days before the application did client change the identity document with which he applied for the loan
DAYS_BIRTH	Client's age in days at the time of application
CODE_GENDER	Gender of the client
REG_CITY_NOT_LIVE_CITY	Flag if client's permanent address does not match contact address (1=different, 0=same, at city level)
DAYS_LAST_PHONE_CHANGE	How many days before application did client change phone
EXT_SOURCE_2	Normalized score from external data source
DAYS_EMPLOYED	How many days before the application the person started current employment
EXT_SOURCE_3	Normalized score from external data source

Elaboration d'un modèle d'apprentissage

Utilisation de SMOTE (pour équilibrer les données) :

- Les classes "prêt accordé" et "prêt non accordé" de la variable cible 'TARGET' sont déséquilibrées dans le jeu de donnée : Intégration de SMOTE dans un pipeline de la librairie imblearn afin que cela n'influe pas sur le test dans l'ordre suivant :



Elaboration d'un modèle d'apprentissage

Création d'une fonction de cout métier et définition des métriques pour comparer les modèles :

La fonction de **coût métier** est définie de la manière suivante :

`f(cout-metier): 10*FN + FP`

-Intégration a `make_scorer` afin de prévenir l'algorithme que le but est de la minimiser (`greater_is_better = False`).

Les **métriques** retenues pour comparer les modèles sont :

-Score-metier, l'accuracy , AUC_test, ainsi que la precision, recall, et f1_score.

Elaboration d'un modèle d'apprentissage

Comparaison des modèles :

Publication sur MLFLOW les résultats obtenus avec les 5 modèles :

- **DummyClassifier**
- **KNeighborsClassifier**
- **LogisticRegression**
- **RandomForestClassifier**
- **LGBMClassifier**

Entraînement et sélection du modèle avec GridSearchCV.

Quelques valeurs et hyperparamètres.

Données d'entraînement : 10% du dataset (30750 ,20) et 5% de données de test (15376, 20).













Elaboration d'un modèle d'apprentissage

Meilleurs modèles obtenus avec chacun des algorithmes

	X_shape	accuracy	precision	recall	F1_score	auc_test	best_sc	score_test	score_train	best_pa	model_name
0	(30750, 20)	0.919225	NaN	0.000000	NaN	0.500000	-4964.0	12420	24820	{'DummyClassifier__strategy': 'most_frequent'}	DummyClassifier
1	(30750, 20)	0.680151	0.128837	0.513688	0.206006	0.641000	-4309.0	10354	7824	{'KNeighborsClassifier__metric': 'manhattan', 'KNeighborsClassifier__n_neighbors': 11}	KNeighborsClassifier
2	(30750, 20)	0.684834	0.156893	0.663446	0.253773	0.740893	-3409.8	8608	17044	{'LogisticRegression__C': 10, 'LogisticRegression__penalty': 'l2'}	LogisticRegression
3	(30750, 20)	0.706686	0.156867	0.601449	0.248834	0.711281	-3728.6	8965	17597	{'RandomForestClassifier__max_depth': 6, 'RandomForestClassifier__min_samples_split': 7, 'RandomForestClassifier__n_estimators': 150}	RandomForestClassifier
4	(30750, 20)	0.776795	0.169982	0.454106	0.247368	0.702589	-3844.2	9534	17701	{'LGBMClassifier__learning_rate': 0.05, 'LGBMClassifier__n_estimators': 50}	LGBMClassifier

Choix de LGBMClassifier : bon accuracy et AUC.

Elaboration d'un modèle d'apprentissage

Run Name	Created	Duration	Models	AUC_test	Recall	accuracy	score-metier	model
intrigued-sponge-984	✓ 26 days ago	5.0s	 sklearn	0.652	0.195	0.851	11290	-
clumsy-hound-1	✓ 26 days ago	2.8s	 sklearn	-	-	-	-	-
bedecked-duck-781	✓ 26 days ago	59ms	-	-	-	-	-	-
 LGBMClassifier	✓ 26 days ago	11.6min	 sklearn, 1 more	0.703	0.454	0.777	9534	LGBMClassifier
righteous-sloth-618	✓ 26 days ago	69ms	-	-	-	-	-	-
 RandomForestClassifier	✓ 26 days ago	31.3min	 sklearn, 1 more	0.711	0.601	0.707	8965	RandomForestClassifier
thundering-stag-242	✓ 26 days ago	67ms	-	-	-	-	-	-
 LogisticRegression	✓ 26 days ago	12.3min	 sklearn, 1 more	0.741	0.663	0.685	8608	LogisticRegression
wistful-goat-588	✓ 26 days ago	55ms	-	-	-	-	-	-
 KNeighborsClassifier	✓ 26 days ago	10.4min	 sklearn, 1 more	0.641	0.514	0.68	10354	KNeighborsClassifier
defiant-auk-918	✓ 26 days ago	55ms	-	-	-	-	-	-
 DummyClassifier	✓ 26 days ago	5.0min	 sklearn, 1 more	0.5	0	0.919	12420	DummyClassifier

Résultats et tracking enregistrés dans MLFLOW.

Choix du modèle, optimisation et calcul du seuil optimal

Optimisation du modèle :

Après sélection du modèle

- Optimisation avec Optuna :

(+d'hyperparamètres et de valeurs)







coût métier:

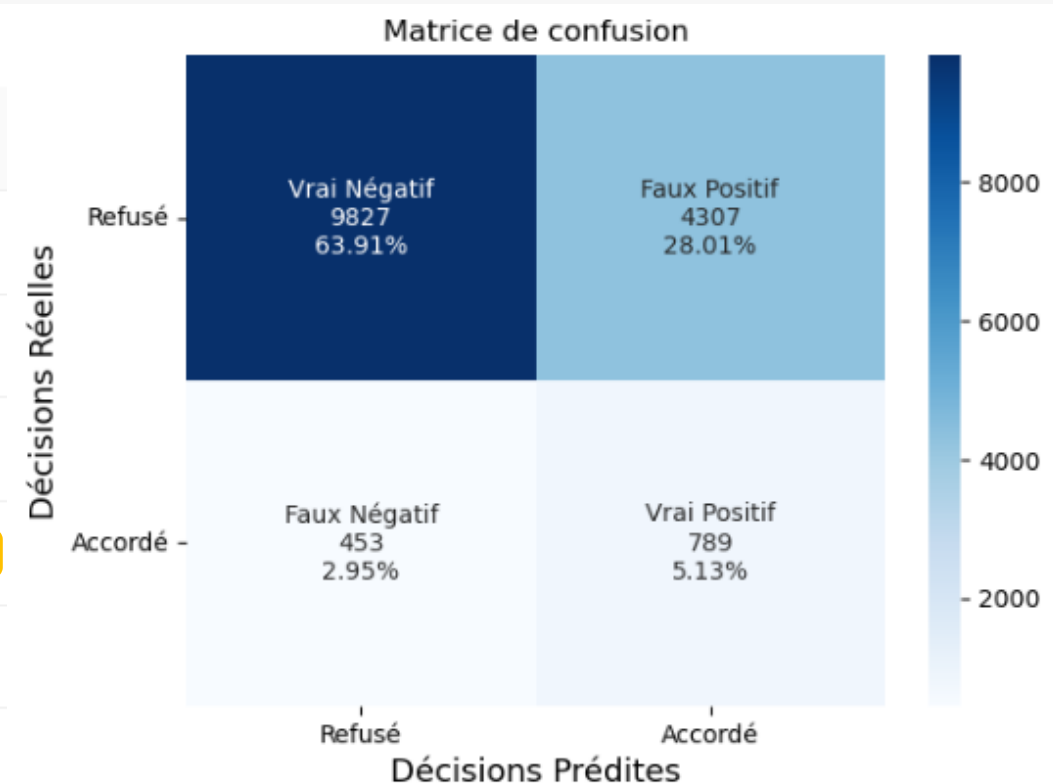
"my_scorer" ($10 \cdot \text{FN} + \text{FP}$)

Paramètre : "direction = minimize".

Résultats enregistrés dans MLFLOW.

Metrics (6)

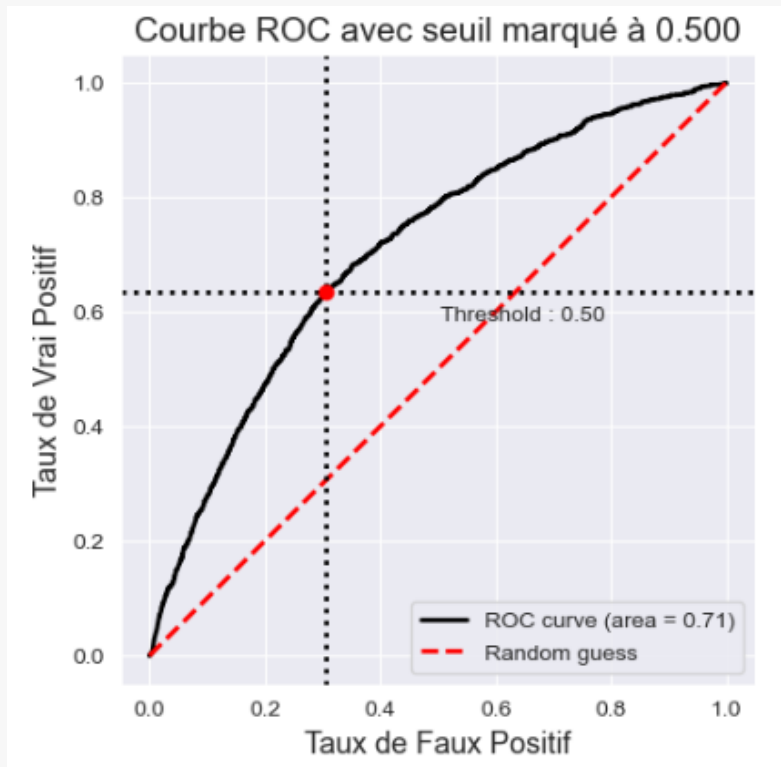
Name	Value
AUC_test 	0.707
Precision 	0.154
Recall 	0.626
accuracy 	0.691
f1_score 	0.247
score-metier 	8925



Choix du modèle, optimisation et calcul du seuil optimal

Etape 2 : Calcul du seuil optimal du model final

Une fonction est créée afin de calculer le seuil optimal :



Coût de la prédiction pour chaque seuil de probabilité possible avec "my_scorer" ($10 \cdot \text{FN} + \text{FP}$)

Le coût le plus faible = seuil optimal.

seuil optimal = 0,5

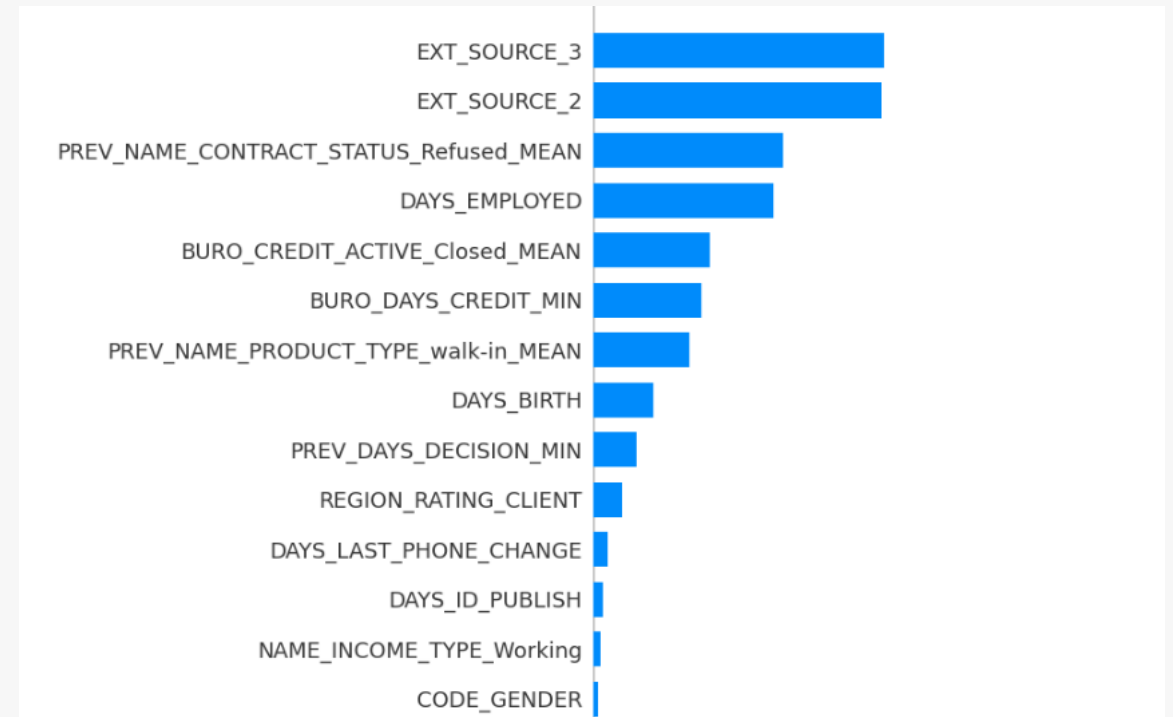
Sauvegarde du model.

Sauvegarde du notebook et du model final dans GitHub

Interprétabilité du modèle

La configuration du modèle fixée, l'explicabilité de celui-ci (locale et globale) avec SHAP :

- Un graphique permettant de représenter l'importance de chaque feature dans les prédictions à l'échelle de tout le jeu de données (interprétabilité globale).



Interprétabilité du modèle

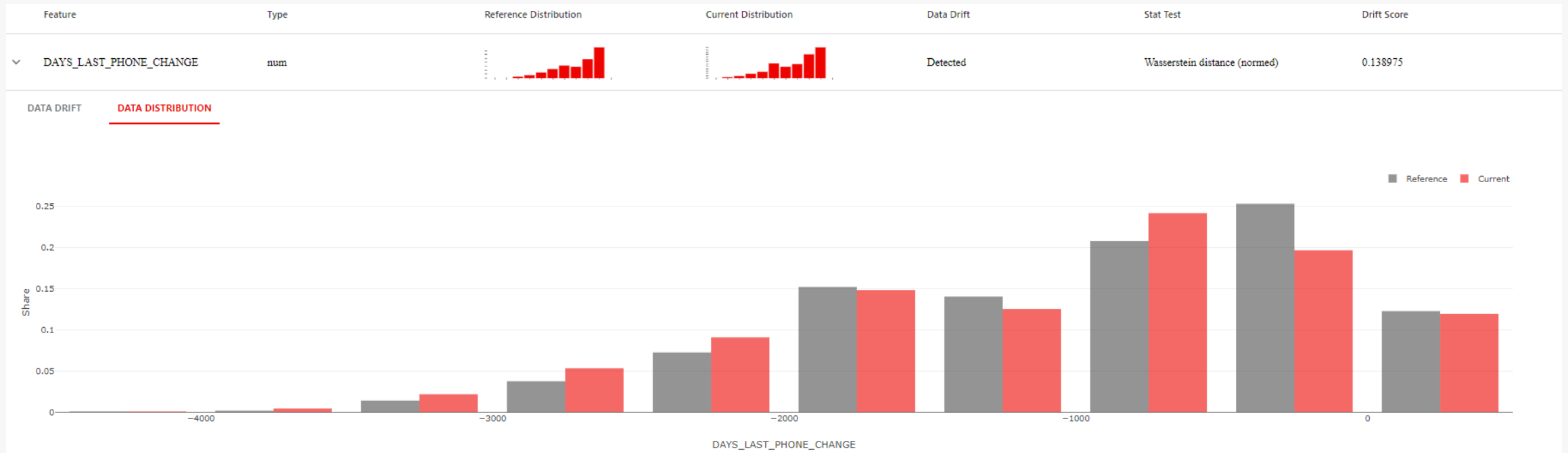
- Un graphique permettant de représenter l'importance de chaque feature dans la prédiction effectuée pour une observation donnée (interprétabilité locale, un client).

```
shap = shap.force_plot(explainer.expected_value, shap_values[0], feature_names=X_test_sample.columns)
shap
```



Performance du modèle

Rapport d'analyse de Datadrift avec librairie Evidently

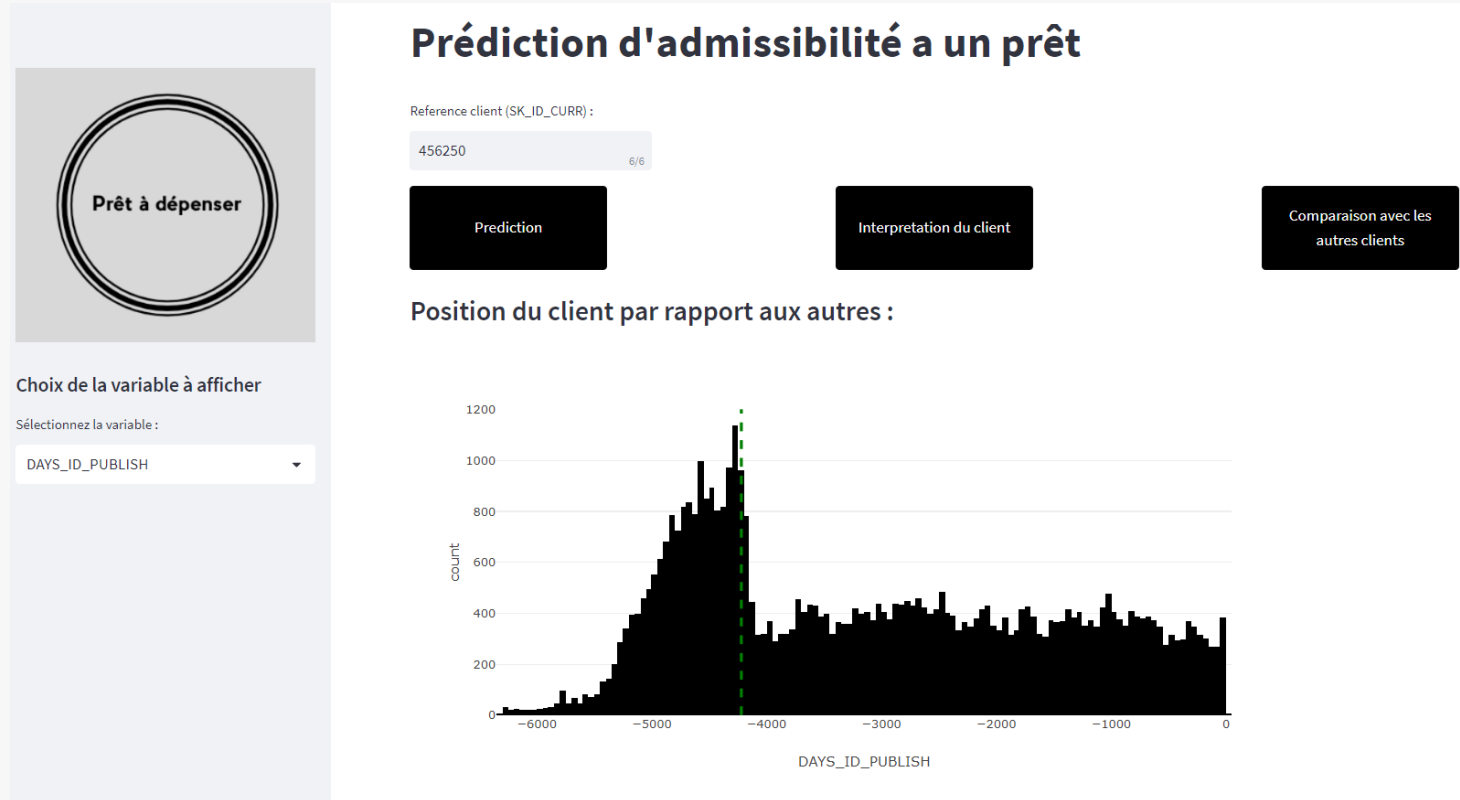


Dérive de données détectée sur DAYS_LAST_PHONE_CHANGE, soit 5% des données.

Rapport enregistré dans le GitHub.

Création de l'API et du Dashboard

Un Dashboard avec un champs à compléter par un numero client a 6 chiffres, affichera au clique sur :



Interface simplifiée pour les malvoyants.

-« **Prediction** » par une **réponse** à la demande, la **valeur de chaque variable** et sa **position par rapport aux autres**.

-« **Interpretation du client** » par une **réponse** à la demande et un **graphique feature importance locale**.

-« **Comparaison avec les autres clients** » par un **graphique feature importance locale** et un **graphique feature importance globale**.

Création de l'API et du Dashboard

L'affichage du Dashboard va être construit avec 2 outils :

Une application Dashboard (app.py) réalisée avec Streamlit qui fait des requêtes à l'API :

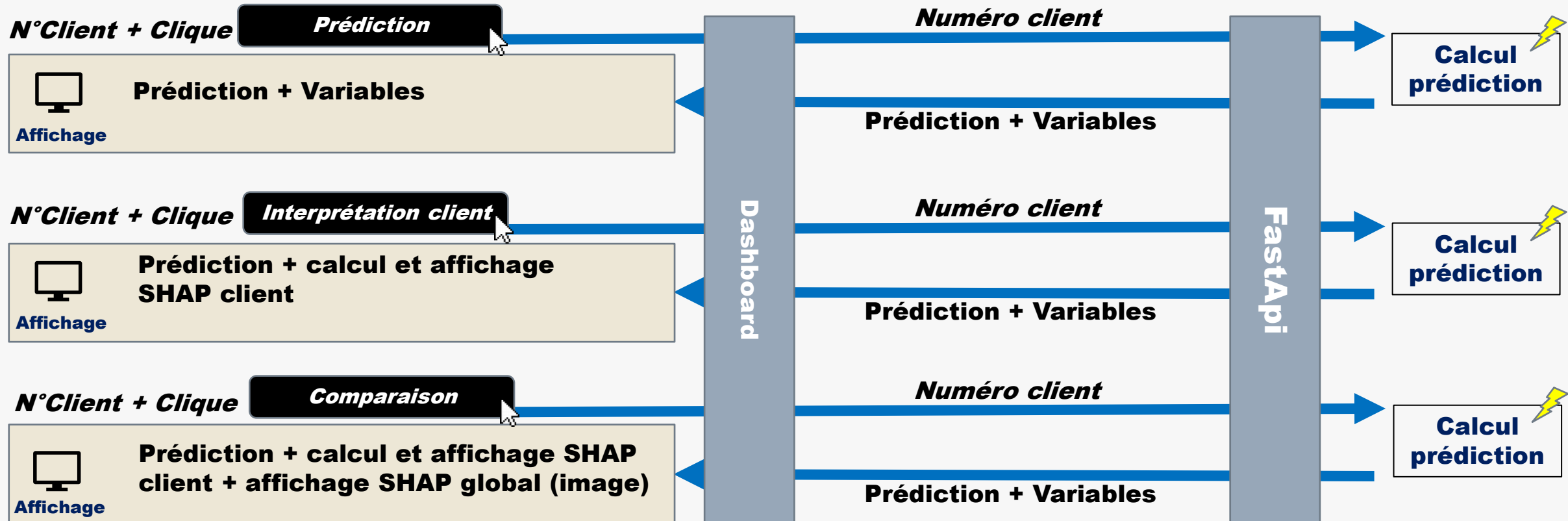
- Réponse à la demande de crédit, les variables et leurs valeurs respectives du client ou récupérer les variables de tous les clients.

Une API (main.py) réalisée avec Fastapi:

- Renvoiera les informations requêtées décrites ci-dessus demandées par Streamlit (calcul de la prédiction du client, variables du clients et variables de tous les clients).

Création de l'API et du Dashboard

Fonctionnement entre le Dashboard et l'Api :



Enregistrement GitHub apres test local.

Déploiement du Dashboard

- Dans un premier temps :
- Déploiement effectué par clonage du projet GitHub sur le serveur AWS EC2.
- Installation des requirement et packages.
- Application accessible : test sur navigateur web.

```
ubuntu@ip-172-31-43-96:~$ uvicorn main:app --reload
INFO: Will watch for changes in these directories: ['/home/ubuntu']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [56334] using StatReload
INFO: Started server process [56336]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

```
ubuntu@ip-172-31-43-96:~$ streamlit run app.py

Collecting usage statistics. To deactivate, set browser.gatherUsageStats to False.

You can now view your Streamlit app in your browser.

Network URL: http://172.31.43.96:8501
External URL: http://52.47.135.193:8501
```

Déploiement du Dashboard

Mise en place avec GitHub Actions d'un déploiement automatisé :

Script de pipeline CI/CD, à chaque push sur la branche « dev » :

1. Récupération code branche "dev".
2. Installation Python + dépendances.
3. Installation pytest + exécution des tests.
4. Si tests pas OK : notification échec + arrêt.
5. Si test OK : mise à jour branche main.
6. Si test OK : Configuration et déploiement EC2 sur AWS avec SSH.

The screenshot displays the GitHub Actions interface for a workflow named 'test_and_deploy'. The workflow has successfully completed 5 minutes ago. The left sidebar shows the 'Jobs' section with 'test_and_deploy' selected, and the 'Run details' section with 'Usage' and 'Workflow file' links. The main panel shows a list of steps in the workflow, each with a status icon (checkmark) and a duration. The steps are: 'Set up job' (1s), 'Checkout source code' (13s), 'Set up Python' (10s), 'Install dependencies' (4m 31s), 'Install pytest' (2s), 'Run tests' (2s), 'Update main branch' (17s), 'Configure and deploy AWS EC2' (8s), 'Notify on failure' (0s), 'Post Set up Python' (0s), 'Post Checkout source code' (0s), and 'Complete job' (0s).

Step	Status	Duration
> Set up job	✓	1s
> Checkout source code	✓	13s
> Set up Python	✓	10s
> Install dependencies	✓	4m 31s
> Install pytest	✓	2s
> Run tests	✓	2s
> Update main branch	✓	17s
> Configure and deploy AWS EC2	✓	8s
> Notify on failure	✓	0s
> Post Set up Python	✓	0s
> Post Checkout source code	✓	0s
> Complete job	✓	0s

Déploiement du Dashboard

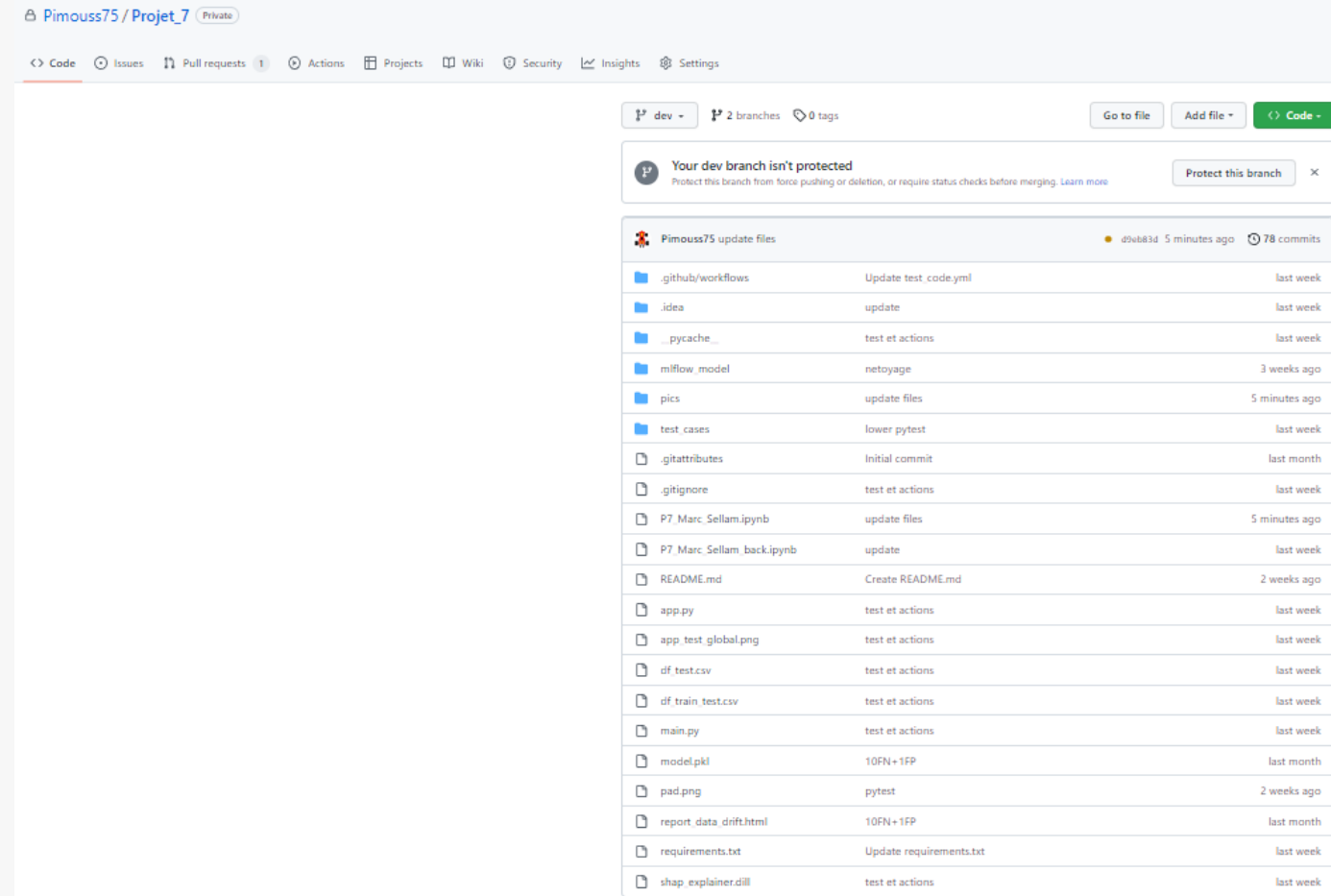
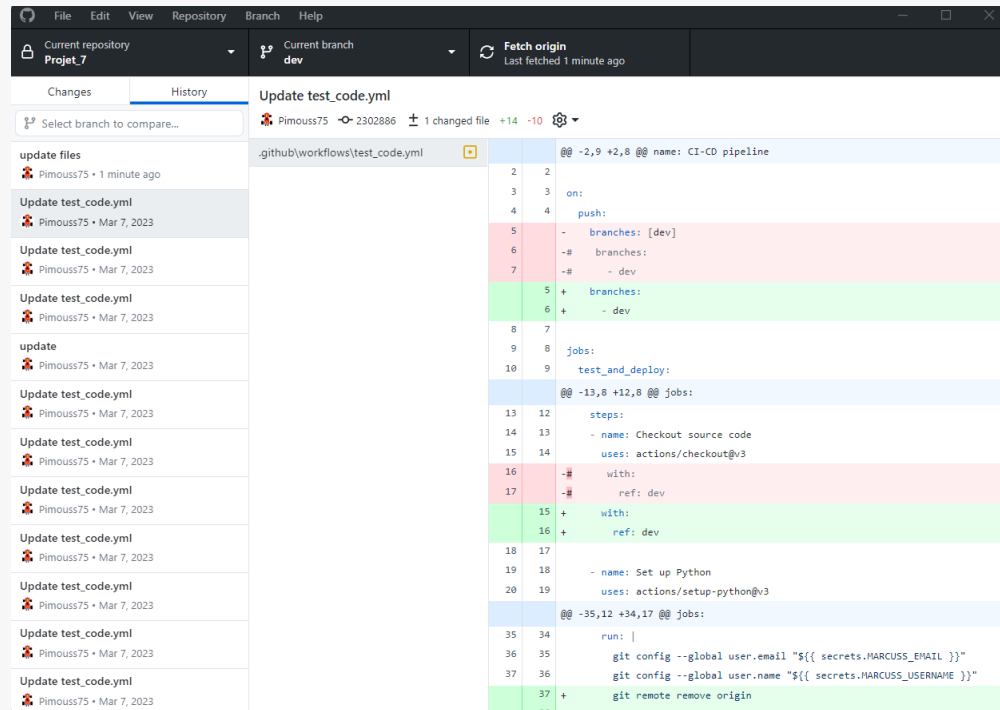
Code de test :

```
1  import joblib
2  import numpy as np
3  import pandas as pd
4
5  import warnings
6  warnings.filterwarnings('ignore')
7
8  model = joblib.load("model.pkl")
9  data = pd.read_csv("df_train_test.csv")
10
11  def test_prediction():
12
13      X_test = data.drop(['TARGET', 'SK_ID_CURR'], axis=1)
14      y_test = data.TARGET
15
16
17      accuracy = model.score(X_test, y_test)
18
19      assert accuracy >= 0.50 # validation du test a partir de 50 % hmmm !!! (67%)
```


Le projet sur GitHub

Commit et push sur GitHub avec GitHub desktop sur la branche dev de :

https://github.com/Pimouss75/Projet_7





Conclusion

En conclusion, le modèle créé est satisfaisant (AUC et accuracy à 0,7) et a été déployé sous forme d'une application qui peut être utilisée par les agents bancaires.

Il existe plusieurs améliorations possibles qui peuvent être apportées pour en augmenter la précision et la capacité de généralisation.

Ces améliorations pourraient comprendre l'ajout de variables supplémentaires ou l'utilisation de techniques d'apprentissage automatique plus avancées par exemple.