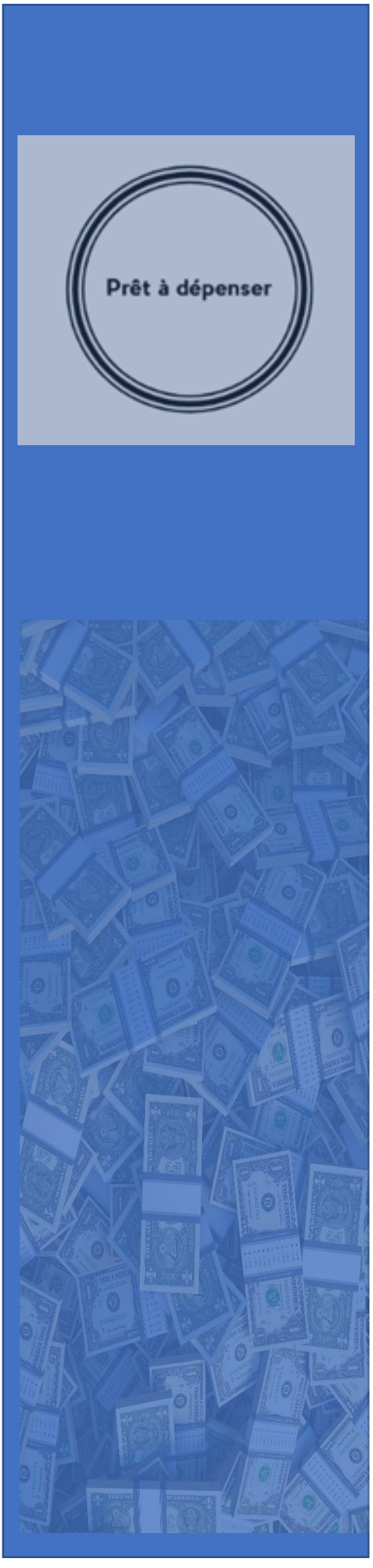
Note Méthodologique

Implémentez un modèle de scoring



Mark Creasey

Projet 7 du Parcours Data Scientist d'Openclassrooms

Table des matières

1 Problématique 3
1.1 Mission 3
2 La méthodologie d'entrainement du modèle 4
2.1 Nettoyage, simple feature engineering et fusion des données 4
2.2 Exploration 4
2.3 Train-Test Split et Preprocessing 4
2.4 Resampling – équilibrage des classes cibles 4
2.5 Feature sélection 5
2.6 Entraînement via GridSearch avec StratifiedKFold cross-validation 5
2.7 Evaluation des performances et choix du meilleur modèle 5
3 La fonction coût métier, algorithme d'optimisation et la métrique d'évaluation 6
3.1 La Fonction cout métier 6
3.2 L'algorithme d'optimisation 6
4 L'interprétabilité globale et locale du modèle 7
4.1 Importance des Features 7
4.2 SHAP (SHapley Additive exPlanations) 7
4.3 Interprétabilité globale: 7
4.4 Interprétabilité locale: 7
5 Les limites et les améliorations possibles 8
5.1 Les limites 8
5.2 Les améliorations possibles 8
6 Conception de dashboard 9
6.1 Le modèle sous forme d'API (application Flask sous Heroku) 9
6.2 La visualisation du dashboard (application Streamlit) 9

1 Problématique

1.1 Mission

La société financière « Prêt à dépenser » propose de crédits à la consommation pour les personnes ayant peu ou pas de tout d'historique de prêt.

Les données financières de 307511 clients anonymisés ont été fournies en sept tables, (descriptions et data : https://www.kaggle.com/c/home-credit-default-risk/data) avec un colonne cible 'TARGET' informant si le client a remboursé son prêt (0) ou était défaillant (1)

Ce projet a pour mission :

- Développer **un modèle de scoring** pour prédire la probabilité de défaut de paiement de ces clients, en s'appuyant sur des sources de données financières
- Développer **un dashboard interactif** pour que les chargés de relation client puis expliquer de façon la plus transparent possible les décisions d'octroi de crédit

Cette note méthodologique présente les processus de modélisation et l'interprétabilité du modèle.

2 La méthodologie d'entrainement du modèle

2.1 Nettoyage, simple feature engineering et fusion des données

Les tables financières (demande de prêt, l'historique de remboursement, des prêts antérieurs, données externes) ont été fusionnes par JOIN sur la clé du client (SK_ID_CURR), avec un script qui a déjà produit des bons résultats de classification. (https://www.kaggle.com/jsaguiar/lightgbm-with-simple-features/script), avec quelques adaptations légères.

- Elimination des valeurs aberrantes (valeurs infinies, DAYS > 100 ans remplacés para NaN)
- Agrégations sur les tables secondaires (MIN, MAX, MEAN, VAR, SUM, COUNT, PERC, DIFF)
- Label Encoder pour colonnes bi-valeur (ex. GENDER)
- One Hot Encoder des colonnes catégoriques (avec création des colonnes _MISSING pour les valeurs NaN)
- Elimination de 31 colonnes avec un seule valeur (pas de variation)
- Elimination de 164 colonnes avec plus de 70% données manquantes

Ces données nettoyées consistent de 602 colonnes numériques pour 307507 clients

2.2 Exploration

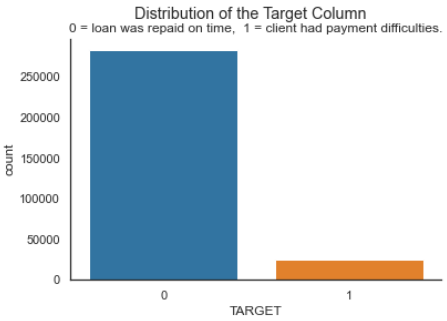


Figure 1: Classification déséquilibrée

Les données ont déjà été explorés en détail pendant un compétition kaggle :

- https://www.kaggle.com/code/willkoehrsen/start-here-a-gentle-introduction/notebook
- https://www.kaggle.com/gpreda/home-credit-default-risk-extensive-eda

Cette exploration montre que la distribution de la cible est très déséquilibrée : moins que 8% des clients sont défaillants. Si on fait une prédiction que tous les clients sont bons, on aura une précision de 93% pour la classe majoritaire, mais on aura identifié aucun client défaillant.

La plupart des 602 colonnes ont très peu de corrélation avec la cible, et simplement ajout du bruit dans la modélisation : Pour l'interprétabilité et la performance du modèle, les features les plus importantes sont sélectionnés (voir ci-dessous) .

2.3 Train-Test Split et Preprocessing

Le jeu de données nettoyés (après sélection des 100 meilleur features) était divisé entre les jeux de données train (80%) et test (20%). Un pipeline de preprocessing était mis en place pour éviter data leakage.

Les valeurs manquantes ont été remplacées par la valeur médiane (toutes les colonnes sont déjà numériques).Pour feature sélection et modélisation, si besoin, les données ont été mise à l'échelle avec StandardScaler.

2.4 Resampling – équilibrage des classes cibles

Pour beaucoup des classificateurs, l'hyperparametre **class_weight = 'balanced'** permet de prendre en compte les déséquilibres dans la classe cible (cost-sensitive).

Plusieurs stratégies de la bibliothèque imbalanced-learn ont été testés aussi pour rééquilibrer les classes cibles : Random undersampling (classe majoritaire) ;Random oversampling (classe minoritaire) ; Synthetic Minority Oversampling Technique (SMOTE); SMOTE TomekLinks - (under sampling classe majoritaire).

2.5 Feature sélection

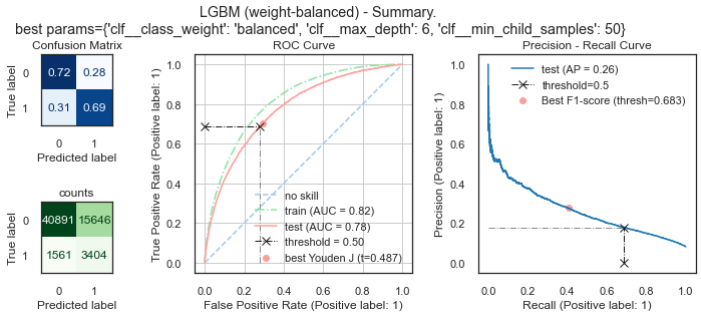
Pour réduire le bruit et améliorer le temps de modélisation, les 100 meilleurs features étaient sélectionnés par un ensemble de méthodes de feature sélection (https://www.kaggle.com/code/sz8416/6-ways-for-feature-selection/): Filter(KBest,Chi2), Wrapper(RFE),Embedded(SelectFromModel : LogisticRegression, RandomForest, LightGBM).

Dans les colonnes restantes, 21 colonnes ont été éliminés car ils étaient hautement colinéaires (exemple : drop `BURO_DAYS_CREDIT_MIN`, `BURO_DAYS_CREDIT_MAX` et garde `BURO_DAYS_CREDIT_MEAN`) (https://www.researchgate.net/publication/350525439_Feature_Selection_in_a_Credit_Scoring_Model_Mathematics_ISSN_2227-7390)

Le jeu de données final était composé de 79 features pour 307507 clients.

2.6 Entraînement via GridSearch avec StratifiedKFold cross-validation

Pour comparer l'influence de stratégie de sampling sur la performance des modèles dans un temps acceptable, un échantillon de jeu de données de 10000 était utilisé. Un fois la stratégie de sampling, les hyperparamètres et modèle choisi, le modèle final était entrainé et optimisé sur l'ensemble de jeu de données.



Les classifiers testés ont été : Dummy (Baseline), RidgeClassifier, LogisticRegression, RandomForest, et LightGBM(Gradient boosting)

Un pipeline imblearn permettait de régler le choix de preprocessing, sampling et classificateur, pour assurer que les scores de cross-validation été testés sur des données sans rééquilibrage.

Plusieurs métriques d'évaluation ont été calculés : précision, recall, F1-score, ROC_AUC, le but étant de minimiser les faux positives (maximum précision) et faux négatives (maximum recall)

2.7 Evaluation des performances et choix du meilleur modèle

Le choix du meilleur modèle a été effectué en retenant le modèle avec le meilleur score ROC_AUC sur le jeu de test.

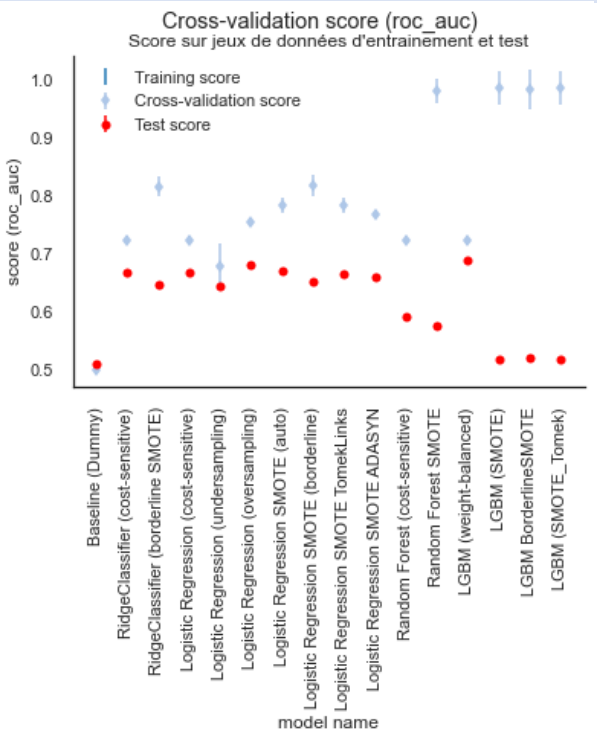
Le ROC_AUC mesure l'Area Under the Curve (aire sous la courbe). Il montre le tradeoff entre specificité et sensibilité

(https://en.wikipedia.org/wiki/Sensitivity_and_specificity)

Plus la courbe approche le coin gauche en haut, meilleur sont le specificité et sensitivité (et donc précision et recall)

Pour les méthodes d'arbre de décision, le SMOTE semble avoir l'effet de overfitting sur le jeu d'entrainement, car sur le jeu de test on voit une baisse significative de capacité de prédiction.

Le modèle Light LGBM sans resampling, mais avec paramètres {class_weight = balanced, max_depth=6} est la plus performant haut score ROC_AUC sur les données test, plus rapide à calculé), et alors est choisi comme meilleur modèle.



3 La fonction coût métier, algorithme d'optimisation et la métrique d'évaluation

3.1 La Fonction cout métier

Pour la banque, le cout de fournir un prêt à un client qui ne rembourse pas son prêt (faux negative (FN)- erreur type II) est plus que la perte de refuser un prêt à un client qui n'aura pas des problèmes de paiement (faux positive (FP) – erreur type I).

		Predicted Label 0	Predicted Label 1
True Label	0	TN	FP
True Label	1	FN	TP

- Recall = TP/(TP+FN) : maximiser recall == minimiser les faux négatives
- Précision = TP/(TP+FP) : maximiser précision == minimiser les faux positives
- F1 est un équilibre entre précision et recall. = 2 * precision * recall / (precision + recall)

Pour mettre plus de poids sur recall, on peut utiliser F(beta>1) score :

$$F_\beta = (1+\beta^2)\cdot\frac{\text{precision}\cdot\text{recall}}{(\beta^2\cdot\text{precision})+\text{recall}}.$$

Une approximation du cout pour la banque sera d'utiliser F(beta=2) score :
f2_score = 5*TP/(5*TP+4*FN + FP)

Une fonction qui estime le cout pour la banque (normalisé pour rester entre 0 et 1, comme les autres scores):

- profit = (TN * valeur_par_pret_fait_bon_client + TP * valeur_de_refuser_un_pret_au_mauvais_payeur)
- loss = (FP * cout_par_pret_perdu_bon_client + FN * cout_de_donner_un_pret_au_mauvais_payeur)
- custom_credit_score = (profit + loss) / (max_profit – max loss)

Ou max_profit = (TN + FP)*tn_profit + (FN+TP)*tp_profit (prêter seulement aux bon payeurs) ; et max_loss = (TP+FN)*fn_loss + (TN+FP)*fp_loss (préter seulement au mauvais payeurs)

Pour cette modele, on suppose : `tn_profit=1, fp_loss=-0.5, fn_loss=-10, tp_profit=0.2`

Donc, custom_credit_score = (TN + 0.2*TP - 10*FN - 0.5*FP) / (max_profit-max_loss)

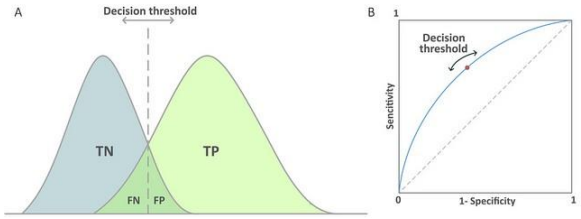
3.2 L'algorithme d'optimisation

Le modèle fourni les valeurs de probabilité (« pred_proba ») qu'un client sera bon payeur(0) et défaillant (1)

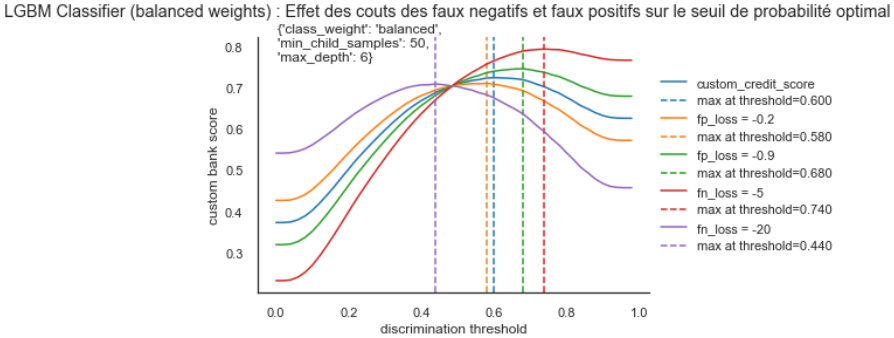
- Si y_pred = (pred_proba[:,1] > threshold) ==1 (True), on considère que le client et défaillant

Les métriques sont calculées en comparant y_pred avec les vrais valeurs (y_true). On récupère le taux des faux positives et négatives de la matrice de confusion :

- `(TN, FP, FN, TP) = metrics.confusion_matrix(y_test_, y_pred_).ravel()`



En changeant le discrimination threshold (seuil de solvabilité), on peut calculer la fonction cout métier, pour trouver le seuil optimal pour une fonction métier donnée ;



Pour la modèle choisi, le seuil optimal est 0.600

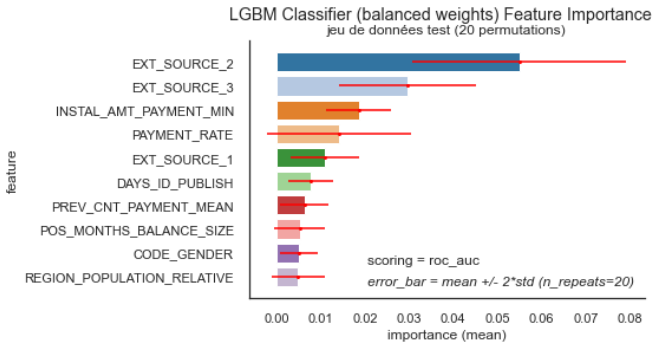
On optimise le modèle sur AUC, puis prédit si le prêt est accepté ou refusé en utilisant le seuil optimal

4 L'interprétabilité globale et locale du modèle

4.1 Importance des Features

La modèle fourni les (impurity-based) poids des `model.feature_importances_`, basés sur les données d'entrainement.

On peut aussi utilisé **`sklearn.inspection.permutation_importance`** pour estimer le (entropy-based) feature importance, basé sur la permutation de valeurs dans chaque feature des données de test – qui nous intéresse plus que les données train



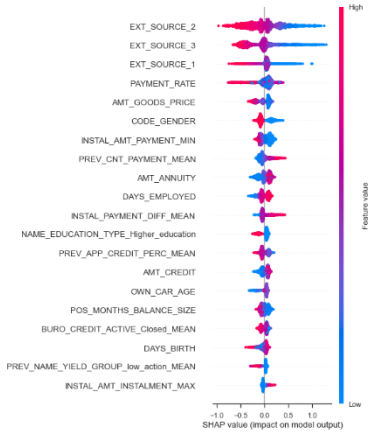
4.2 SHAP (SHapley Additive exPlanations)

La méthode SHAP (https://shap.readthedocs.io/) calcule les shap_values : l'impact d'une variable (sur la prédiction) pour chaque ligne des données. Les valeurs SHAP sont additives : les valeurs en rouge augment la valeur prédit (risque d'être défaillant), une valeur bleu réduit la prédiction (basse de la risque d'être défaillant).

4.3 Interprétabilité globale:

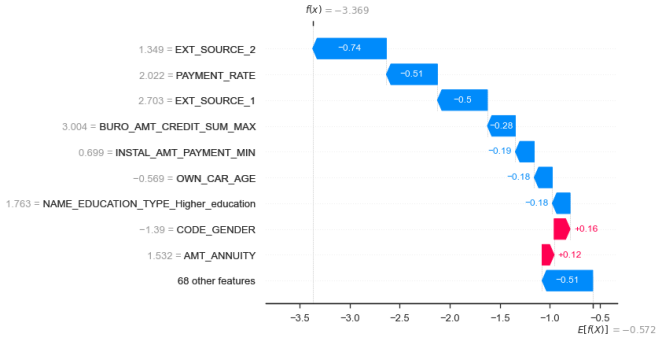
Si on prend la moyenne des SHAP values pour chaque feature, on obtient l'importance des features pour la prédiction.

On peut visualiser la distribution des valeurs de pour les features plus importants via un 'summary plot', en forme de beeswarm ou violin

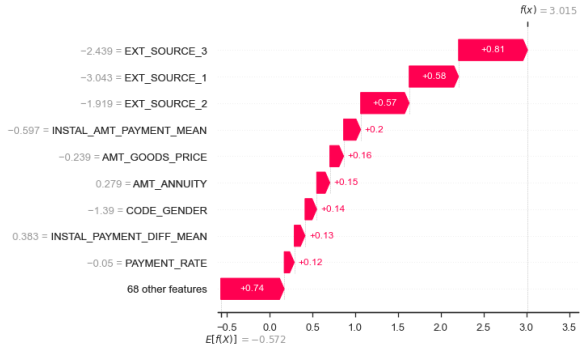


4.4 Interprétabilité locale:

Les contributions négatives ont un effet de réduire la valeur de la prédiction.



Client bas risque (prob=0.03)



Client haut risque (prob=0.95)

5 Les limites et les améliorations possibles

5.1 Les limites

Les modèles ont été calculé sur une partie des données : il faut analyser l'effet de sample size sur les résultats (ex via des courbes d'apprentissage (learning curves)`

On n'arrive pas a séparer complètement les bons payeurs des clients défaillants (le ROC_AUC des données d'entrainement reste entre 0.7 et 0.8)

L'application de SMOTE améliore les scores d'entrainement, mais pas les scores de validation, pour la taille d'échantillon utilisé.

SMOTE devient rapidement trop lourde à appliquer sur le jeu de données entières : la géneration des points synthètiques est très lent, et les modeles crées avec SMOTE (via imblearn.Pipeline) sont trop grands pour être enregistrés

Il faut faire le choix entre des erreurs de type I (précision) et les erreurs de type II (recall)

Pour la banque, le recall est la plus important

5.2 Les améliorations possibles

1. Revoir la création des features avec les experts du métier : Le script de nettoyage, agrégation, fusion et feature engineering utilisé semble avoir été fait sans connaissance du métier – beaucoup des variables crée par le script sont sans intérêt ou dupliquées.
2. Revoir la stratégie de traiter les valeurs manquantes (médiane par défaut)
3. Améliorer la sélection de features pour être adapté à chaque modèle (Wrapper/Embedded)
4. Faire des learning curves pour optimiser la taille d'échantillon pour les modeles
5. Augmenter la recherche des meilleurs hyperparamètres des modèles
6. Changer de Flask API vers fastapi (https://fastapi.tiangolo.com/) – plus rapide, documentation automatique des requêtes, moins de lignes de code, inclus authentification et sécurité
7. Ajout authentification pour accéder au dashboard
8. Ajout encryptage des données client
9. Stocker les données client séparément de l'API (ça exige de les mettre en cache dans la mémoire de l'API, sinon ça devient trop lent) – par exemple dans un S3 bucket sur AWS
10. Visualiser la distribution de chacun des features les plus importantes pour un client donnée pour comprendre mieux ou se situer la client parmi les clients

6 Conception de dashboard

6.1 Le modèle sous forme d'API (application Flask sous Heroku)

La prédiction est faite par un application Flask, écrit en python avec les routes :

- Liste des client ids : /clients/
- Données d'un client : /client/<id>
- Prédiction (probabilité de défaut) : /predict/<id>
- Explication SHAP client : /explain/<id>
- Explication SHAP globale : /explain/

Le déploiement est sous Heroku à l'adresse https://mc-oc-7.herokuapp.com

Le code source de l'api se trouve dans le dossier `api` du repository ' https://github.com/mrcreasey/oc-ds-p7-scoring-dashboard (voir README.md dans ce dossier pour instructions)

6.2 La visualisation du dashboard (application Streamlit)

Le dashboard fait des requêtes à l'API, car il n'a pas accès ni aux données, ni au modèle.

Il est écrit en python et streamlit, et déployé sur share.streamlit.io à l'adresse :

https://mrcreasey-oc-ds-p7-scoring-dashboard-dashboardmain-70agjx.streamlitapp.com/

Le code source du dashboard se trouve dans le dossier `dashboard` du repository ' https://github.com/mrcreasey/oc-ds-p7-scoring-dashboard (voir README.md dans ce dossier pour instructions)