Implémentez un modèle de scoring

PLAN

1/ Présentation de la problématique
2/ Présentation du jeu de données
3/ Exploration et préparation du jeu de données
4/ Modélisation
5/ Métrique métier
6/ API et Dashboard
7/ Conclusion et perspectives

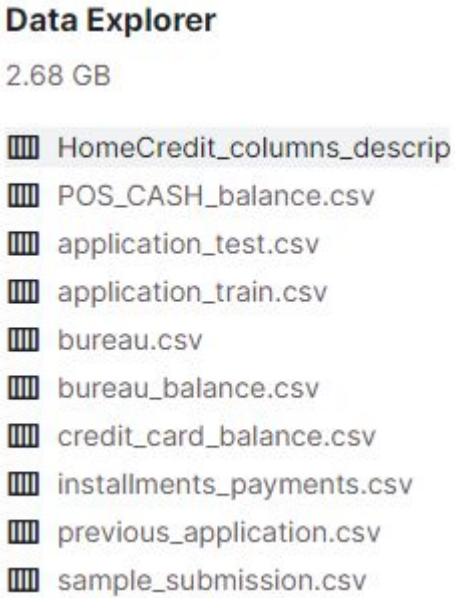
Data scientist dans société financière **"Prêt à dépenser"**

Mettre en oeuvre un outil de **"scoring crédit"** : probabilité qu'un client rembourse son crédit et **classification** de la probabilité

Objectifs :

- Construire modèle de scoring
- Créer dashboard interactif

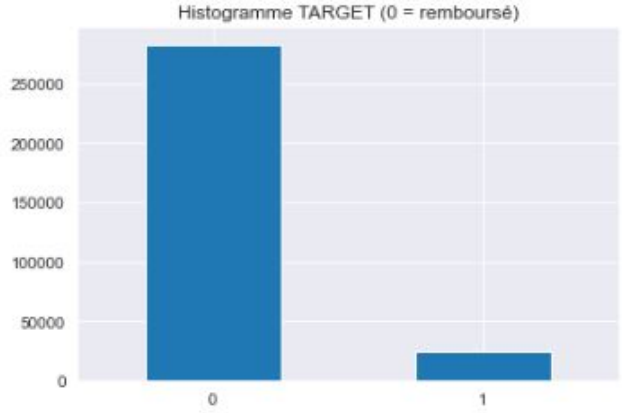
- Fichiers CSV
- Kernel Kaggle

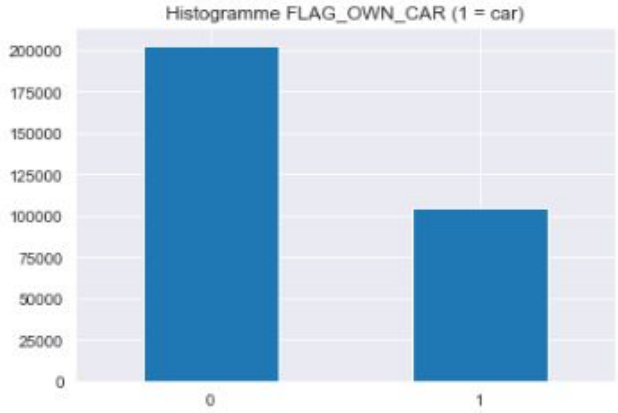


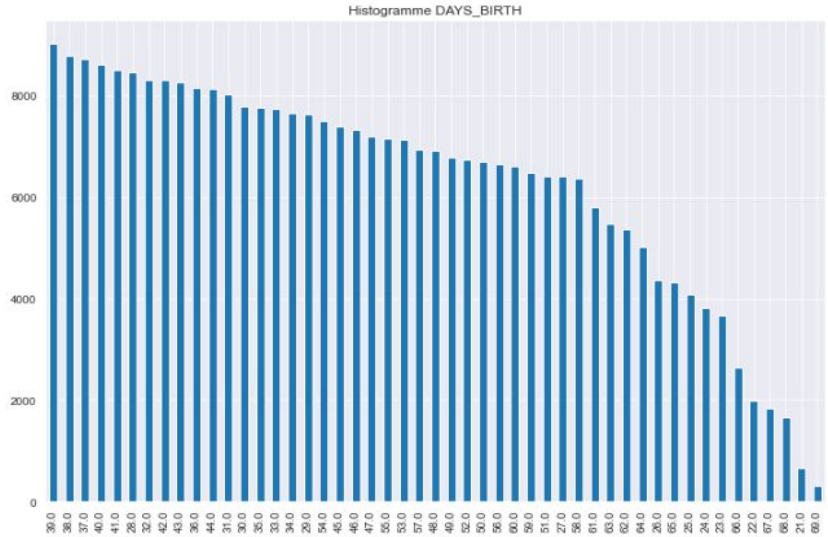
- Préparer le jeu de données avec kernel Kaggle : 'factorize' et encodage des variables catégorielles
- Supprimer les variables avec taux de valeurs manquantes > 50%
- Identifier les variables numériques (non encodées ou factorisées)
- Imputer les valeurs manquantes par la valeur médiane pour chaque variable numérique

Jeux de données "final" de
307 507 lignes (clients),
247 colonnes

Exploration et préparation du jeu de données








```
data[data['FLAG_OWN_CAR']==0]['OWN_CAR_AGE'].value_counts(dropna=False)
```

```
NaN    202922
Name: OWN_CAR_AGE, dtype: int64
```

- **Classifieurs :**
  - Dummy Classifier : retourner la classe la plus fréquente
  - Régression Logistique : prédire la probabilité d'appartenir à une classe selon un seuil (0.5)
  - Random Forest Classifier : méthode ensembliste de Decision Trees
  - Decision Tree Classifier : retourner une prédiction en répondant à des règles logiques par niveaux
  - LGBM Classifier : méthode de boosting 'light' (permettant plus de rapidité de calcul)

- **Métriques :**
  - Score ROC AUC : score pour classification, indiquant la performance d'un modèle. Score constitué par la sensibilité et la spécificité

  - Score F1 : score pour classification, moyenne harmonique de la précision et rappel

  - Score rappel : pourcentage de 'positifs' bien prédits par le modèle

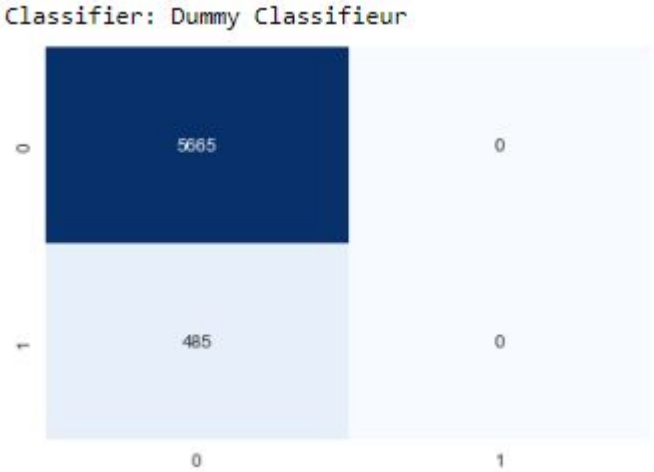
- Créer une liste de classifieurs et listes d'hyperparamètres

```
models = [
          {'name': 'Dummy Classifieur', 'clf': DummyClassifier(random_state=0)},
          {'name': 'Régression Logistique', 'clf': LogisticRegression(random_state=0)},
          {'name': 'Random Forest', 'clf': RandomForestClassifier(),
           'params': {'randomforestclassifier__random_state': [0],
                      'randomforestclassifier__n_estimators': [200],
                      'randomforestclassifier__max_depth': range(10, 20, 2)}},
          {'name': 'Decision Tree', 'clf': DecisionTreeClassifier(),
           'params': {'decisiontreeclassifier__random_state': [0],
                      'decisiontreeclassifier__max_depth': range(10, 20, 2)}},
          {'name': 'LGBM', 'clf': LGBMClassifier(),
           'params': {'lgbmclassifier__random_state': [0],
                      'lgbmclassifier__n_estimators': [200],
                      'lgbmclassifier__max_depth': range(2, 6),
                      'lgbmclassifier__learning_rate': [0.01, 0.02, 0.03]}}
]
```

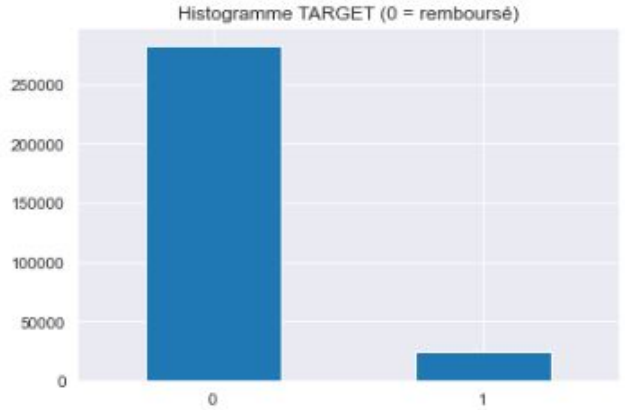
- Créer une fonction permettant d'entraîner chaque classifieur avec recherche des meilleurs hyperparamètres (GridSearchCV) et de calculer des scores de performance (entraînement sur échantillon, 10%, en conservant la proportion des classes)

Modélisation - Résultats

name	AUC_train	AUC_test	f1_train	f1_test	recall_train	recall_test
Dummy Classifieur	0.50	0.50	0.00	0.00	0.00	0.00
Régression Logistique	0.76	0.75	0.04	0.03	0.02	0.01
Random Forest	0.72	0.74	0.00	0.00	0.00	0.00
Decision Tree	0.66	0.61	0.38	0.07	0.25	0.05
LGBM	0.74	0.75	0.02	0.00	0.01	0.00



- **Équilibrer les classes**
  Cas de figure avec déséquilibre des classes. Plusieurs approches pour corriger :
  - SMOTE : densifier la classe minoritaire (over-sampling)
  - SMOTETomek : densifier la classe minoritaire et réduire la majoritaire (over-sampling + under-sampling)
  - Etablir le paramètre 'class_weight' dans nos classifieurs




```
class_weight='balanced'
```

name	AUC_train	AUC_test	f1_train	f1_test	recall_train	recall_test
Régression Logistique	0.76	0.75	0.27	0.26	0.69	0.66
Random Forest	0.71	0.74	0.52	0.27	0.79	0.38
Decision Tree	0.63	0.64	0.33	0.22	0.87	0.57
LGBM	0.74	0.75	0.29	0.25	0.71	0.64

Améliorations des scores F1 et rappel

- Conserver meilleur classifieur : LGBM

```
[{'lgbmclassifier__class_weight': 'balanced',
  'lgbmclassifier__learning_rate': 0.03,
  'lgbmclassifier__max_depth': 3,
  'lgbmclassifier__n_estimators': 200,
  'lgbmclassifier__random_state': 0}]
```

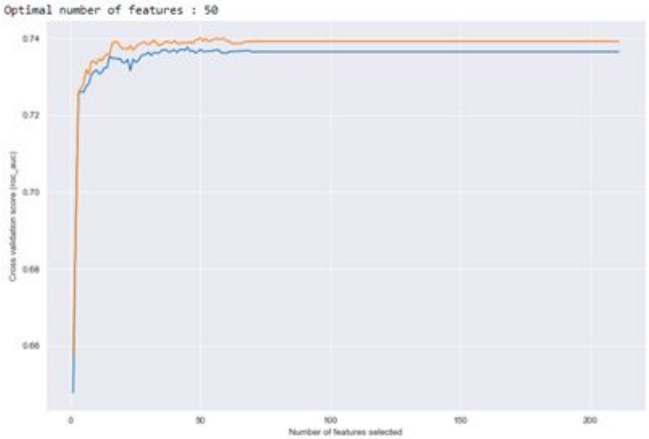
- Feature engineering

```
data['credit_annuity_ratio'] = data['AMT_CREDIT'] / data['AMT_ANNUITY']
data['credit_goods_price_ratio'] = data['AMT_CREDIT'] / data['AMT_GOODS_PRICE']
data['credit_downpayment'] = data['AMT_GOODS_PRICE'] / data['AMT_CREDIT']
```

name	AUC_train	AUC_test	f1_train	f1_test	recall_train	recall_test
LGBM	0.79	0.76	0.29	0.27	0.71	0.67

Améliorations légères sur tous les scores

- Sélection de variables avec RFECV (Recursive Feature Elimination with Cross-Validation)

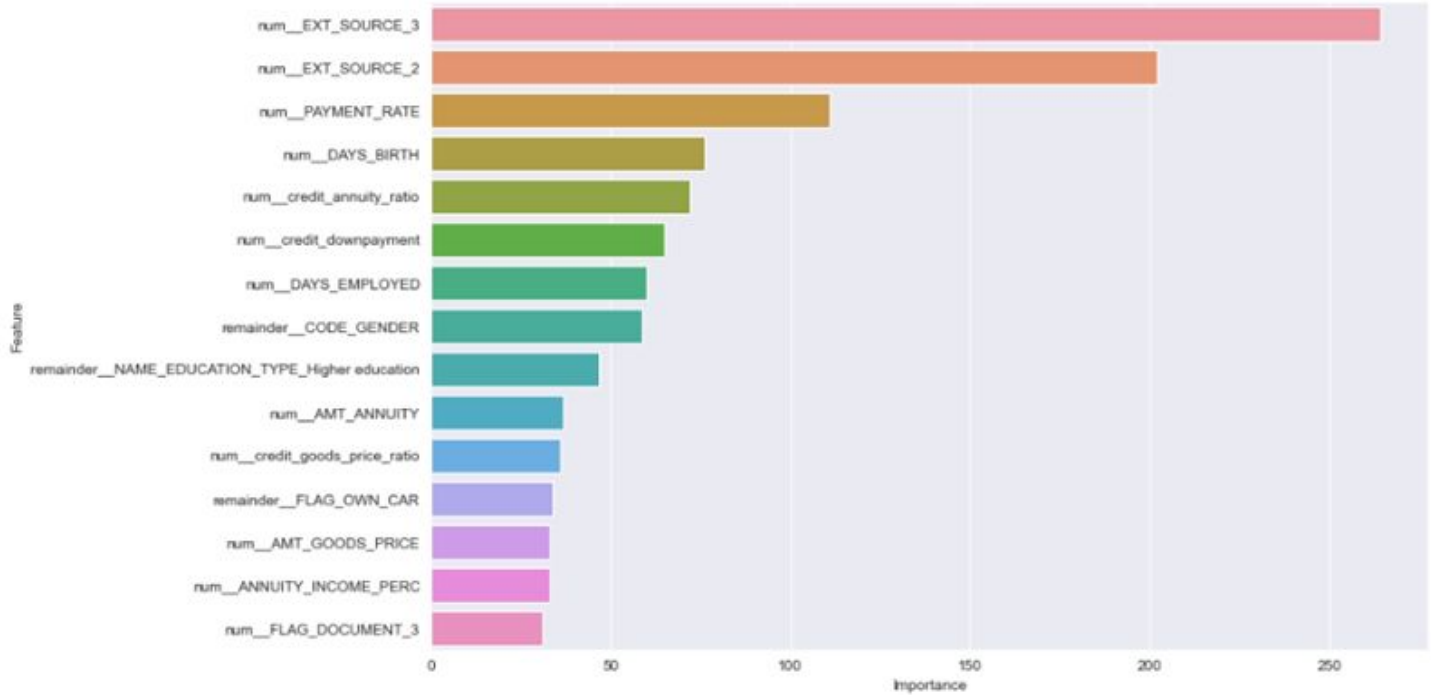


- Entraîner le classifieur sur l'ensemble du jeu de données

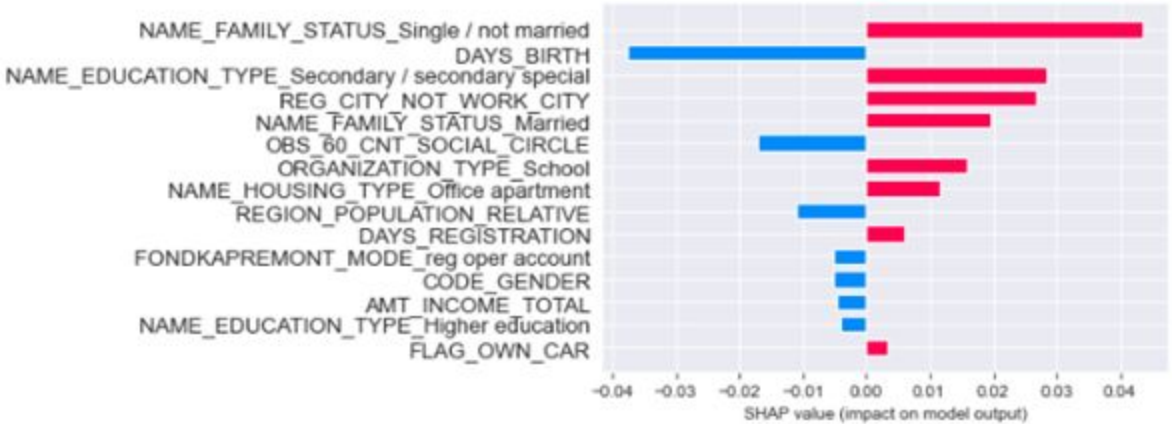
```
AUC_train = 0.75, AUC_test = 0.75
f1_train = 0.26, f1_test = 0.26
recall_train = 0.68, recall_test = 0.68
```

Légères diminutions des scores avec égalité entre les jeux d'entraînement et test

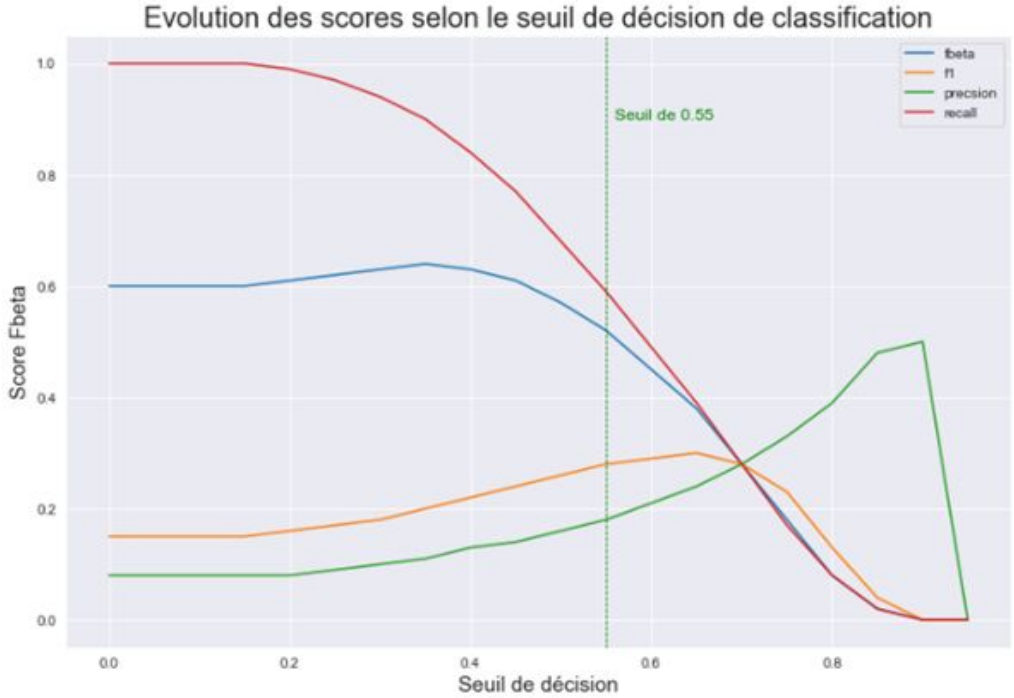
- Interprétation globale du modèle



- Interprétation locale du modèle



- Création d'une métrique métier : score fbeta
  Valeur beta = 4



- Création de l'API avec FastAPI

  Création d'un modèle dynamique sur base de BaseModel

```
ClientModel = create_model(
    'ClientClassifierModel',
    **features_dict,
    __base__=BaseModel
)
```

  Fonction POST : transmission des informations d'un client et retour de la probabilité d'avoir des difficultés de paiement

  Déploiement de l'API sur Heroku

- Création du dashboard avec Streamlit

  Création de trois onglets :

  - ❏ Base de données clients avec filtres
  - ❏ Informations d'un client avec accord ou refus du crédit et explications
  - ❏ Comparaison d'un client avec l'ensemble des clients

  Création d'une requête avec l'API pour obtenir la probabilité d'accord ou de refus du crédit

  Déploiement du dashboard sur Heroku

- **Conclusion**

  Capacité d'améliorer le modèle de classification en apportant des améliorations à notre jeu de données initial :
  - Gérer le déséquilibre des classes
  - Créer de nouvelles informations (variables)
  - Sélectionner les informations (variables) pertinentes
  - Créer une métrique métier

- **Comment améliorer le modèle ?**

  Préparation du jeu de données plus approfondie : utilisation des CSV à disposition, feature engineering, sélection de variables (réduction de dimensions, variables corrélées), imputation des variables manquantes

  Equilibre des classes : essayer/approfondir d'autres méthodes