



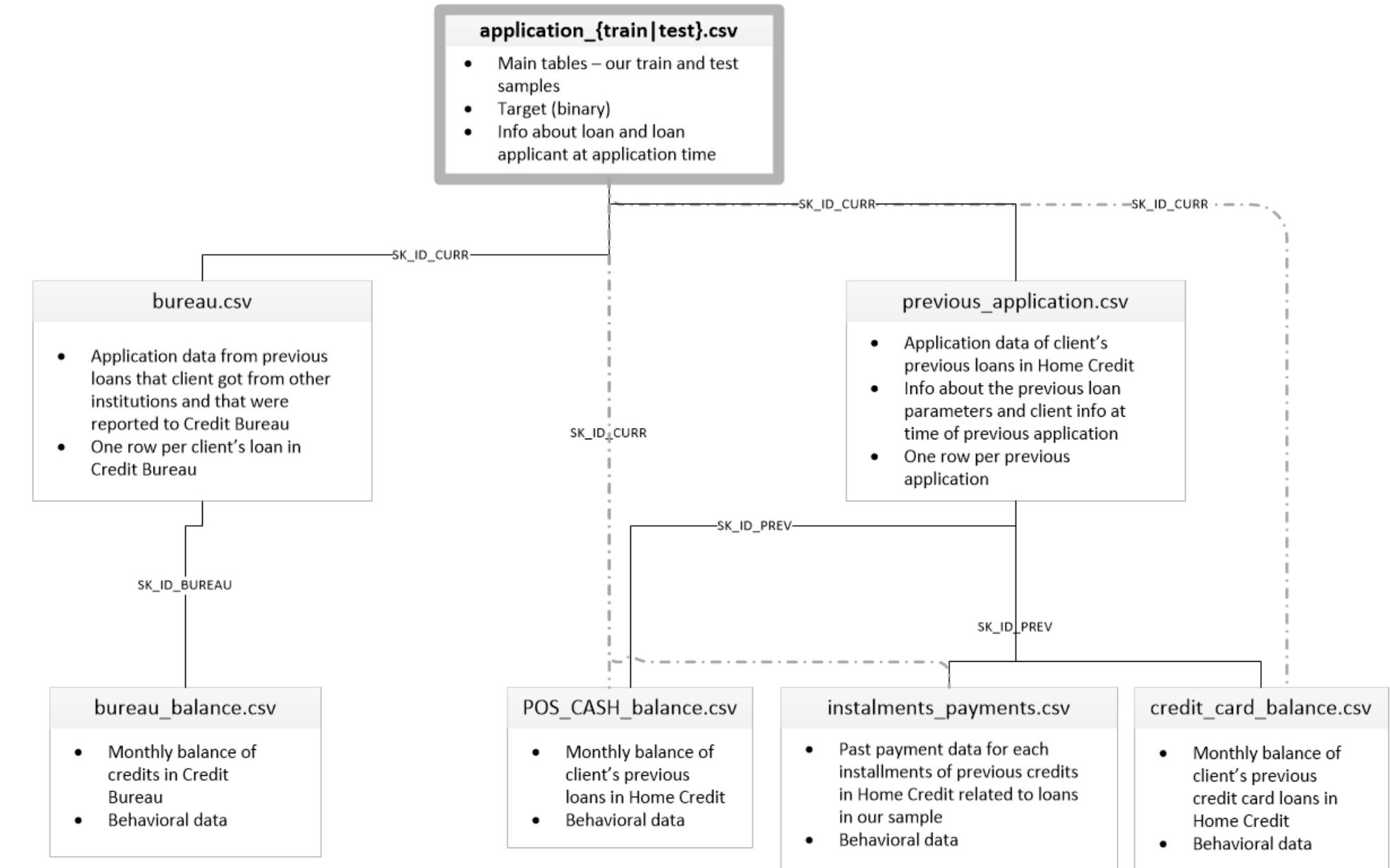
CREDIT SCORE

P7 - DS | ALHASSANE AHMED

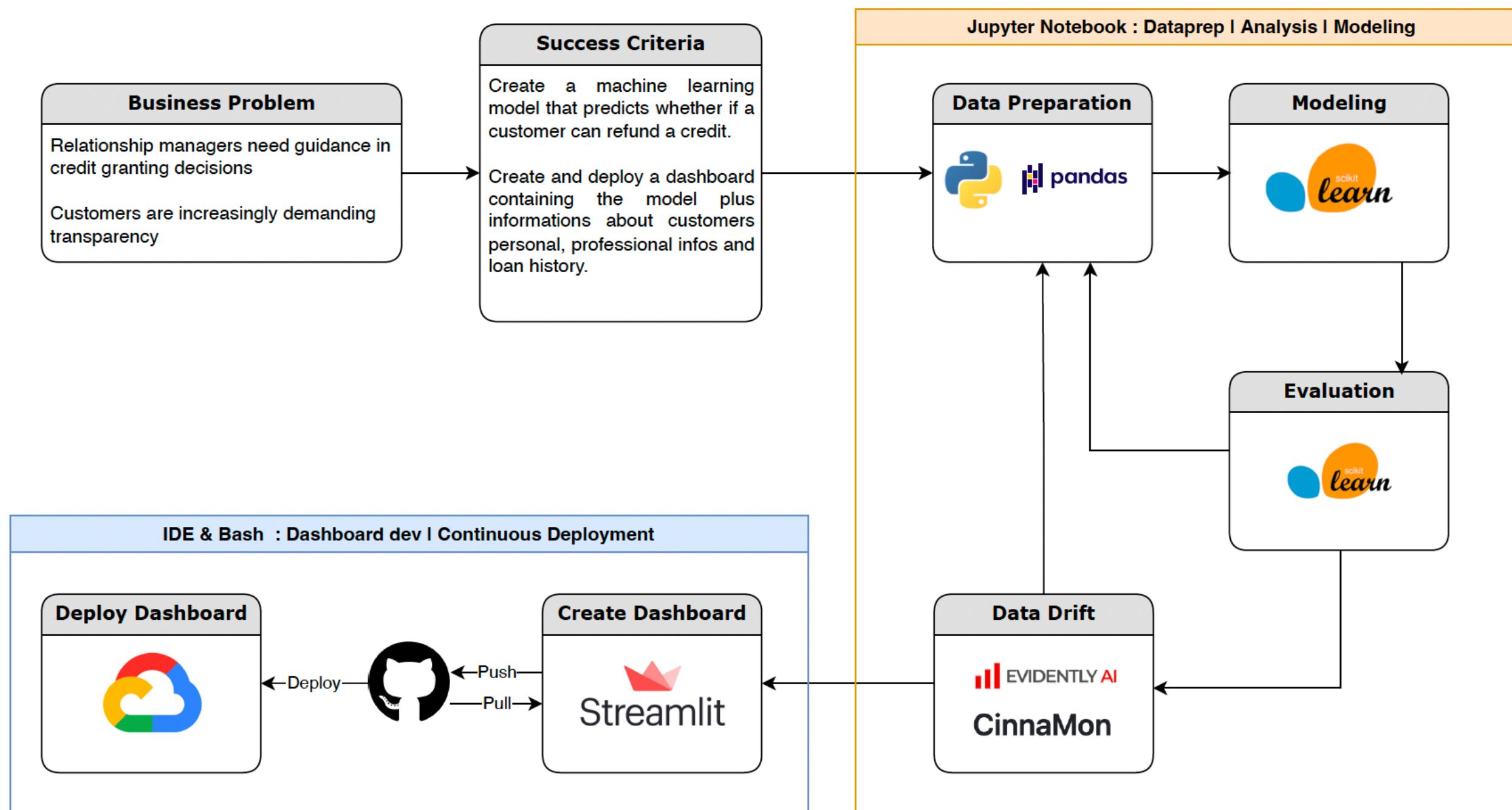
Contexte

L'objectif principal de cette mission est de développer un dashboard interactif pour que les chargés de relation client puissent communiquer de façon la plus transparente possible les décisions d'octroi de crédit.

Ce dashboard pourra aussi servir d'aide à la décision d'octroi...



RoadMap | Lifecycle



Plan d'action

Préparation : cette partie consiste à l'exploration et la préparation des données afin de préparer les sets d'entrainements.

Modélisation : dans cette partie, plusieurs models seront testés, les metrics et paramètres pertinents seront traqués avec MLflow. Une fois le meilleur model déterminé, celui-ci sera interprété puis optimisé afin de réduire certains risques vis à vis du métier.

Dashboard: cette partie consiste à la création d'une application web avec le modèle meilleur. Les information qui suivent sont à prendre en compte :

- End-users : les chargés de relation client.
- Transparence vis-à-vis des décisions d'octroi de crédit.

Industrialisation : la première étape de cette partie est la construction d'une image Docker qui sera ensuite déployée sur Google Cloud Plateforme. Un déploiement continu sera mis en place afin d'intégrer toute modification de manière automatisée avec des pipelines CICD.



PHASE 1

Préparation des données

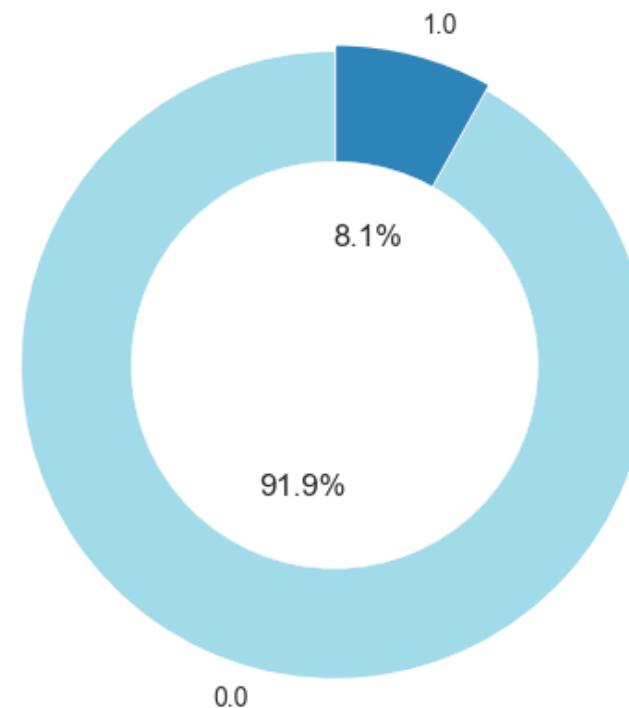
Suite à l'application du **kernel fourni dans l'énoncé** pour consolider les données, le nouveau dataset comprend à présent **799 colonnes** et **307 511 lignes**.

Il est important de noter que plusieurs nouveaux indicateurs ont été calculés, durant le process notamment :

- 'Income_credit_perc',
- 'annuity_income_perc',
- 'days_employed_perc',
- etc...

Exploration

Tout bon projet de Data Science comprend une partie exploration. Cette partie permet de mieux comprendre les données auxquelles nous sommes confrontées. Dans notre cas, celle-ci nous permis de constater que nous faisons face à des **données déséquilibrées**.



Nan Imputing

Pour gérer les valeurs manquantes, Simple Imputer a été utilisé avec comme stratégie

imputation par la médiane.

Cross Validation

Une **validation croisée sur 10 folds** a été appliquée aux données avec comme output nos trains et tests sets.

```
def cross_val_split(X, y, num_folds=10, stratified=False, debug=False):
    """This function implement a cross validation. It takes as input a some features(X),a target variable and integer as the number of folds. It return the trainings and testing sets after the cross validation."""
    # Cross validation model
    if stratified:
        folds = StratifiedKFold(n_splits=num_folds, shuffle=True, random_state=5)
    else:
        folds = KFold(n_splits=num_folds, shuffle=True, random_state=5)
    for n_fold, (train_idx, valid_idx) in enumerate(folds.split(X, y)):
        train_x, train_y = X.iloc[train_idx], y.iloc[train_idx]
        valid_x, valid_y = X.iloc[valid_idx], y.iloc[valid_idx]
    for key, value in {
        "X_train": train_x,
        "X_test": valid_x,
        "y_train": train_y,
        "y_test": valid_y,
    }.items():
        print(f"{key} shape : {value.shape}")
    return train_x, valid_x, train_y, valid_y
```

Resampling

Étant donné que nous faisons face à des données déséquilibrées, des méthodes de rééquilibrage des trains sets ont été appliquées, à savoir :

- **Undersampling** : supprimer des observations de la classe majoritaire afin de rééquilibrer le jeu de données
- **Oversampling** : répéter des observations de la classe minoritaire afin de rééquilibrer le jeu de données
- **Weight_balanced** : indiquer au modèle le déséquilibre afin qu'il en tienne compte directement



PHASE 2

Modélisation

Dans cette partie plusieurs modèles seront testés et chaque experimentation sera traquée via mlflow. Le modèle le plus performant sera retenu pour la suite de l'analyse...

Une custom score permettant de minimiser certains risques vis à vi du métier sera intégrer au modèle.

Evidently Ai et Shape permettront de vérifier le data drift et la contribution des variables.

Automatisation du process

une fonction permettant d'entrainer un modèle choisi a été définie. L'autolog de mlflow permet de logger tous les paramètres ainsi que les metrics du model. Cette même fonction affiche un classification_report ainsi qu'une matrice de confusion.

```
def run_experiment(experiment_name, name, model, X_train, X_test, y_train, y_test, model_name, balancing_method):
    mlflow.set_experiment(experiment_name)
    mlflow.sklearn.autolog()
    mlflow.lightgbm.autolog()
    with mlflow.start_run(run_name=name):
        mlflow.set_tag("deleveloper", "Alassane")
        my_model = model
        my_model.fit(X_train, y_train)
        y_pred = my_model.predict(X_test)
        print("*" * 25, f"Test scores - {name}", "*" * 25)
        print(classification_report(y_test, y_pred))
        cm = ConfusionMatrix(my_model, classes=y_train.value_counts().index)
        cm.score(X_test, y_test)
        cm.show()
    metrics = evaluate_model(my_model, X_train, y_train, X_test, y_test, model_name, balancing_method)
    return metrics
```

Tuning | Evaluation

GridSearchCV et Hyperplot permettront de tuner les models. Une fonction qui permettra de récupérer l'AUC et le temps de prédiction sera définie. En de cette fonction locale, un tracking via la fonction autolog de mlflow est prévue. Celle permet de traquer tous les paramètres et métriques à chaque fit. Ces artifacts traqués seront à travers l'ui de mlflow. (cf image)

	Algorithm	Balancing_method	AUC	AUC_test	Time
0	Baseline	Original	0.771	0.758	0.000000e+00
1	Baseline	Undersampling	0.775	0.758	0.000000e+00
2	Baseline	Oversampling	0.804	0.741	0.000000e+00
3	Baseline	Balanced	0.773	0.760	9.536743e-07
4	LogisticRegression	Original	0.766	0.758	0.000000e+00
5	LogisticRegression	Undersampling	0.765	0.758	0.000000e+00
6	LogisticRegression	Oversampling	0.803	0.741	7.152557e-07
7	LogisticRegression	Balanced	0.766	0.760	0.000000e+00
8	lightGBM	Normal_data	0.783	0.777	9.536743e-07
9	lightGBM	Undersampling	0.780	0.774	0.000000e+00
10	lightGBM	Oversampling	0.970	0.718	0.000000e+00
11	lightGBM	Balanced	0.784	0.778	0.000000e+00

Buziness Risk Minimization



Accorder un crédit à un client qui ne peut rembourser.
C'est une **situation de perte** : à éviter absolument.



Accorder un crédit à un client qui peut remboursé. C'est une **situation de gain** : c'est l'idéal.



Refuser à un client qui ne peut rembourser. C'est une **situation sans risque** (sans perte, ni gain)



Refuser à un client qui peut rembourser. C'est une **situation de perte**, à éviter.

Un score métier à été créer et intégré dans le model afin de prendre ces situation. **Une pondération en fonction de chaque situation** a permis de considérablement minimiser le risque d'accorder un crédit à un client qui ne peut rembourser.

```
def custom_score(y_true, y_pred) :
    """
    This function aims to minimize the risk for a given loan.

    Parameters
    -----
    - y_true : true values of the target
    - y_pred : predicted value for the target

    Returns
    -----
    - score : score for the prediction
    """

    (TN, FP, FN, TP) = confusion_matrix(y_true, y_pred).ravel()
    total_neg = TN + FP      # total negatives cases
    total_pos = TP + FN      # total positives cases

    dangerous_situation = -10  # The loan is granted but the customer defaults : the bank loses money
    nice_situation = 1         # The loan is refunded => the bank makes money
    nulle_situation = 0        # The loan is (rightly) refused : the bank neither wins nor loses money
    no_good_situation = -1     # Loan is refused by mistake : the bank loses money it could have made,
    | 

    # total gains
    gain = TP*nulle_situation + TN*nice_situation + FP*no_good_situation + FN*dangerous_situation

    # if all observations are correctly predicted
    best = total_neg*nice_situation + total_pos*nulle_situation

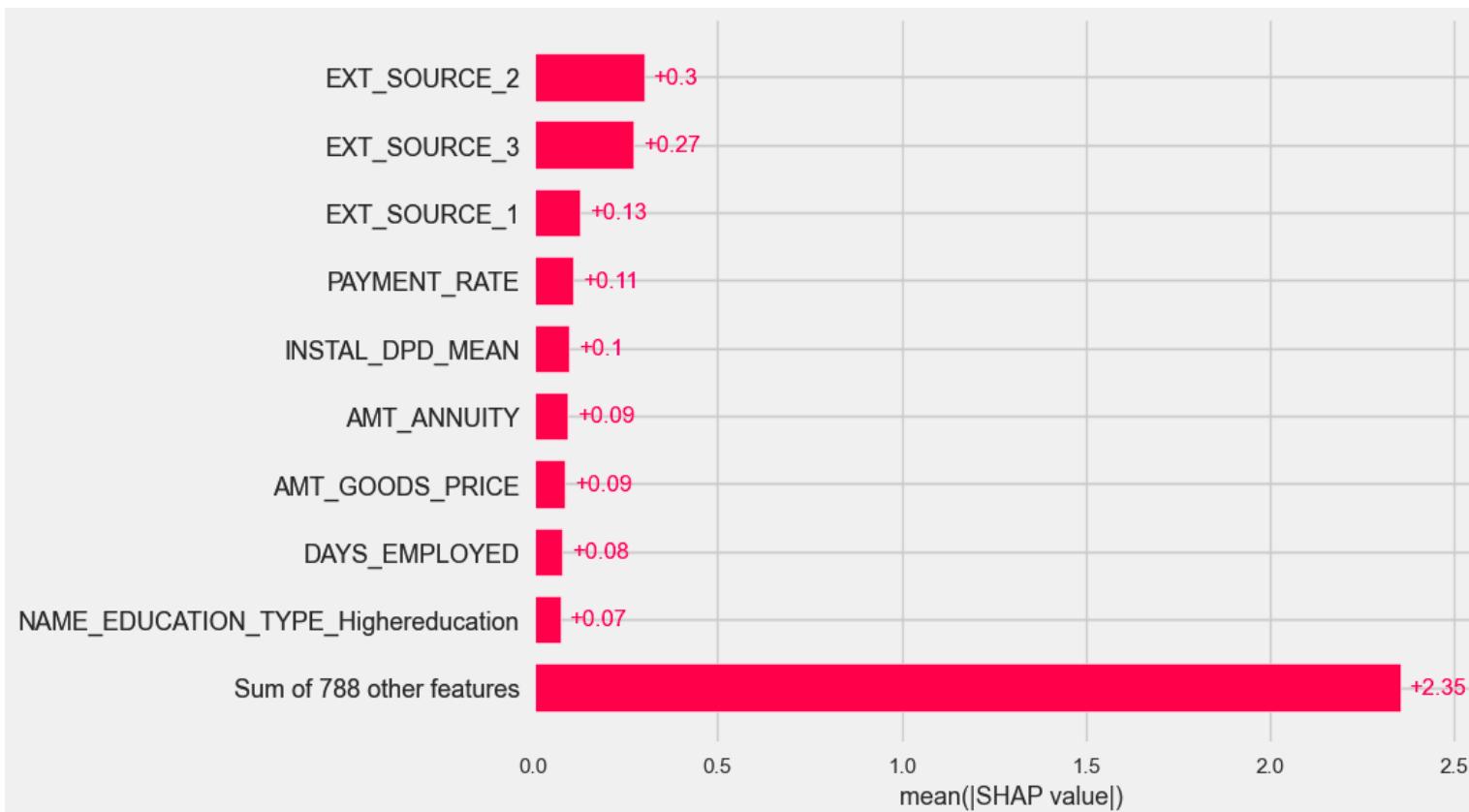
    # if all observations are predicted = 0
    baseline = total_neg*nice_situation + total_pos*dangerous_situation

    # normalize to get score between 0 (baseline) and 1
    score = (gain - baseline) / (best - baseline)

    return score
```

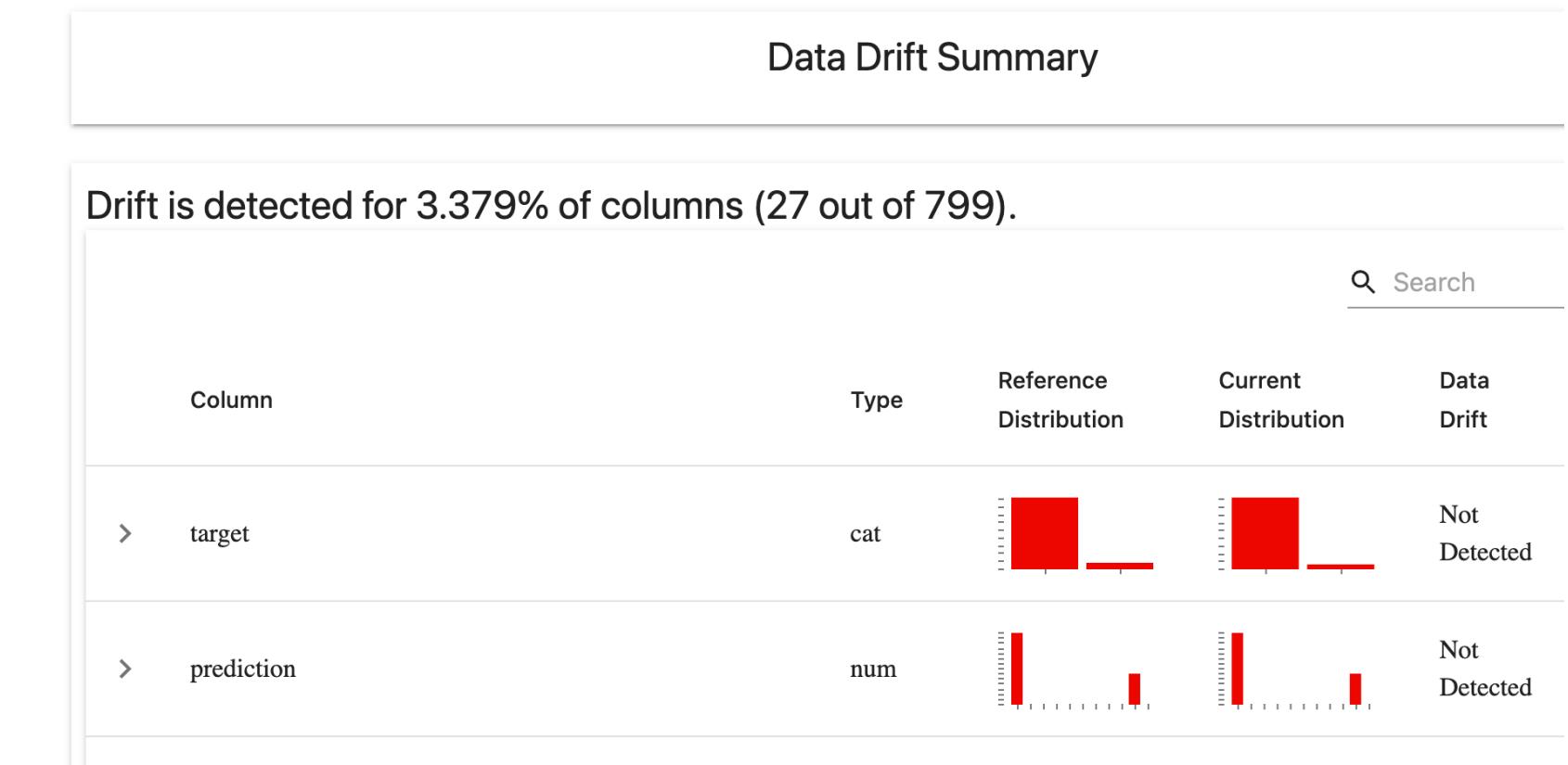
Global Explainability

La librairie shape a permis d'analyser les contributions de chaque variable du model.



Data Drift

Le rapport sur le data drift généré avec evidently permet d'observer une faible quantité de drift dans les données





PHASE 3

Le Dashboard

Cette partie consiste au développement d'un dashboard avec comme utilisateur final : le chargé de relation client.

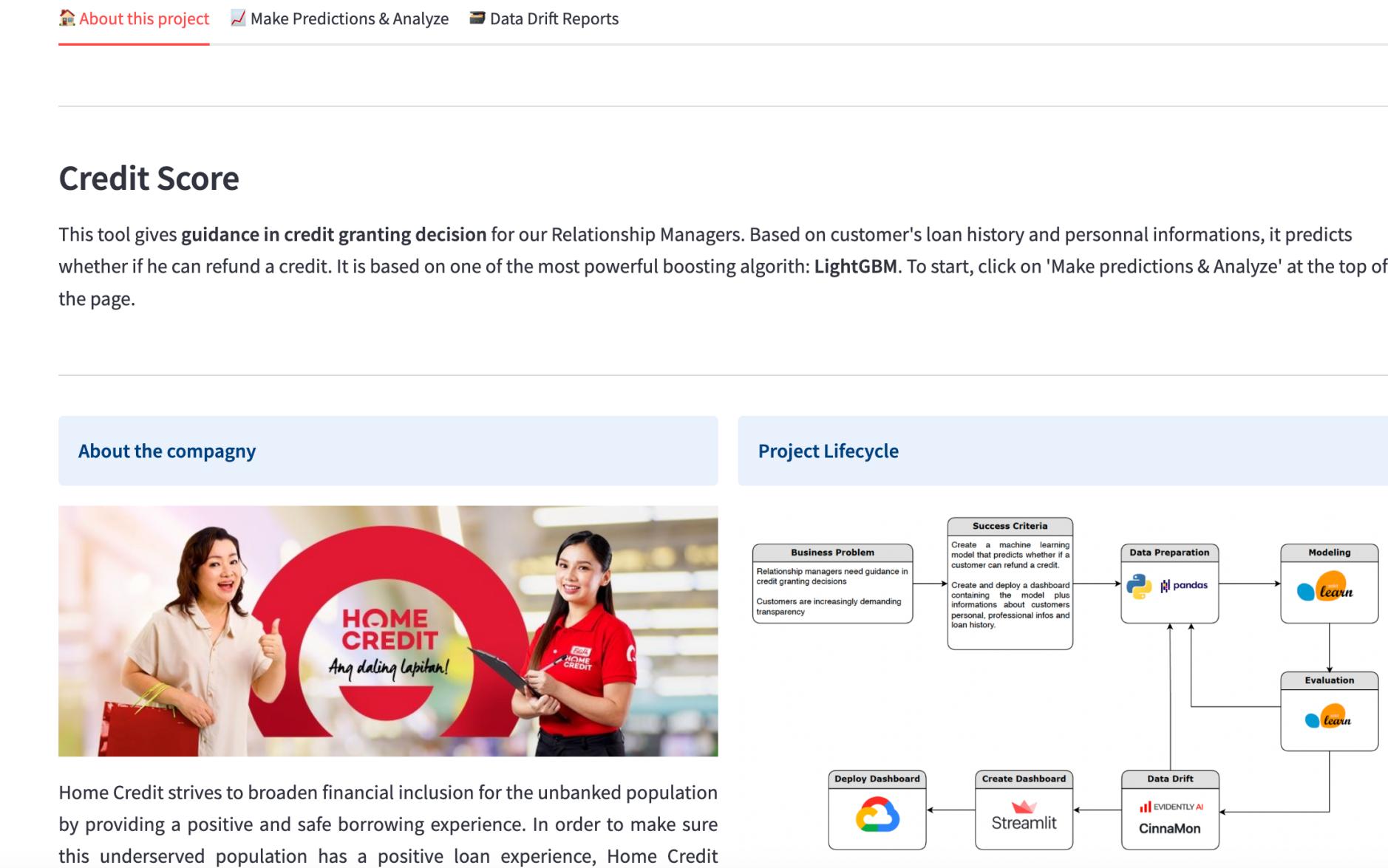
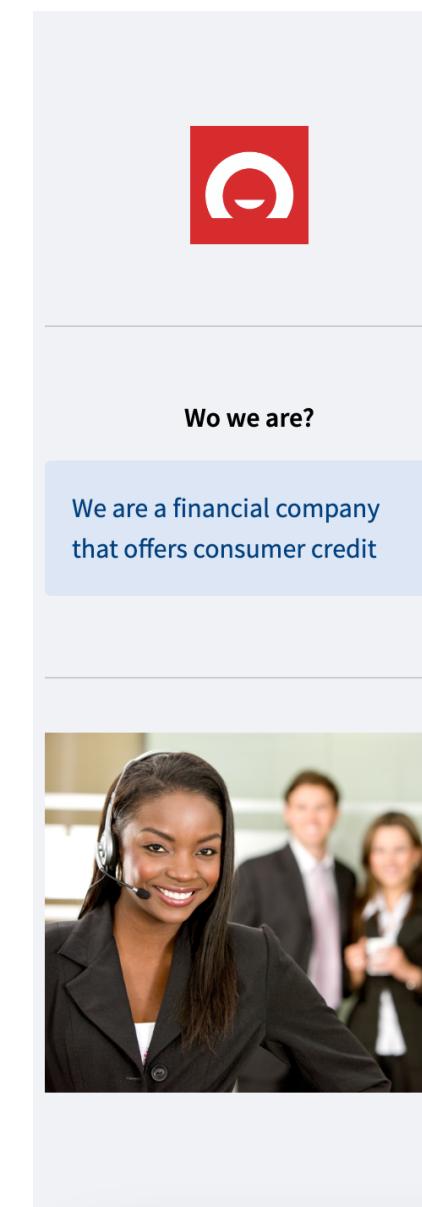
Ce dashboard va mettre en avant diverses informations qui ont un impact sur la capacité du client à rembourser. Il s'agit de :

- certaines informations personnelles et professionnelles (status, ancienneté travail, revenu, etc...)
- informations sur l'historique de prêt du client
- un graphe sur les variables suivantes : revenu, crédit en cours et annuités + data explainability
- le score crédit prédit par le modèle.

Développement et Interface

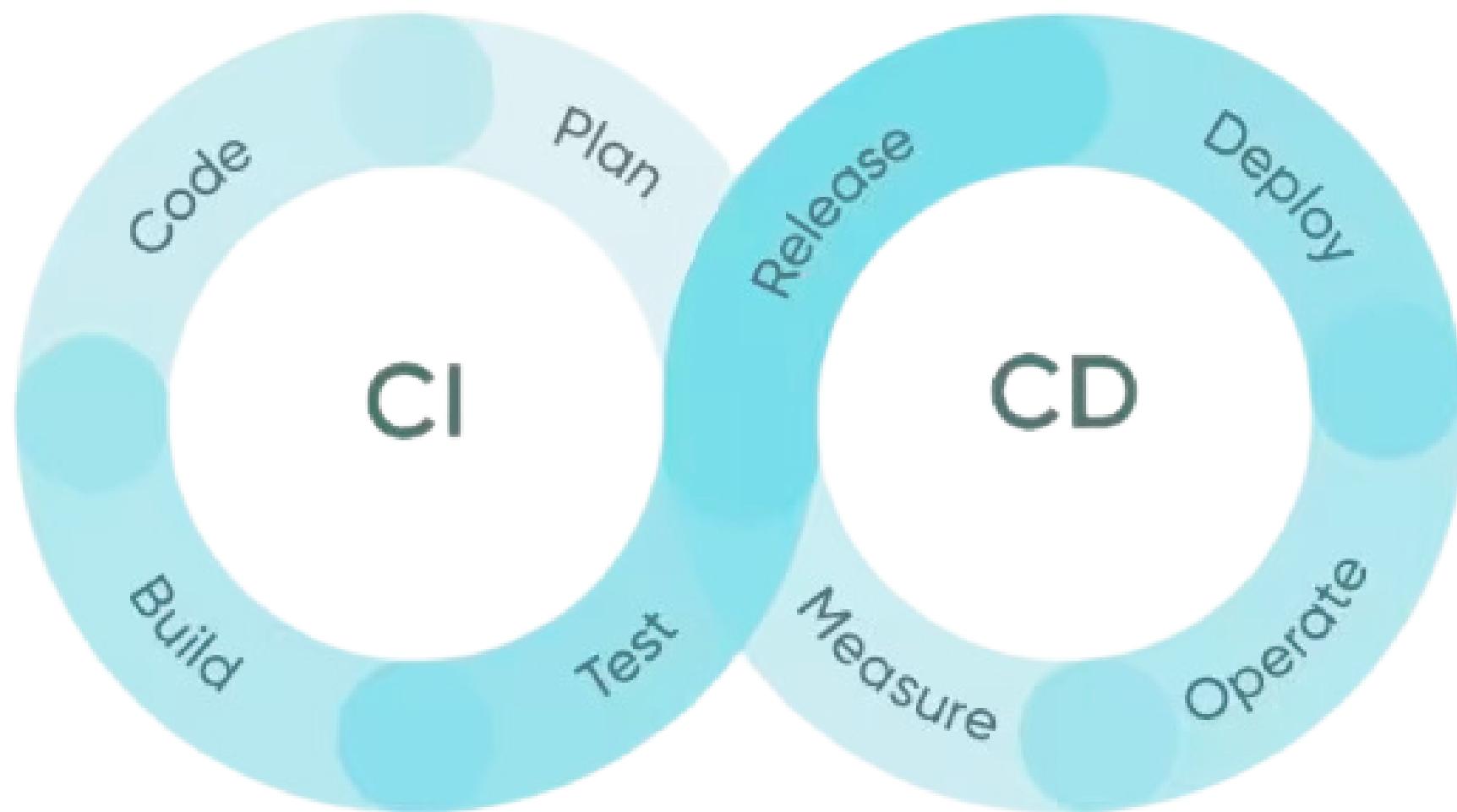
L'outil choisi pour développer l'application web qui servira de dashboard aux chargés de relation client est **streamlit**. Les fichiers de l'application sont stockés à l'emplacement : loan-scoring/app

```
├── Dockerfile
├── README.md
├── app.py
└── data
    ├── cover.jpg
    ├── data_drift_report.html
    ├── full_data.pkl
    ├── logo.png
    └── training_features.pkl
└── models
    ├── best_lightGBM_model.pkl
    ├── best_model.pkl
    ├── lgbm_balanced.pkl
    └── model_opti.pkl
├── poetry.lock
├── pyproject.toml
└── tests
    ├── __init__.py
    └── test_unit.py
```



PHASE 4

Industrialisation



La première étape de cette partie est la **Dockerization**. En effet, nous allons empaqueter l'application web ainsi que toutes les librairies et dépendances nécessaires dans un container créé à partir d'une image Docker. Cette image sera déployée sur Google Cloud Plateforme via Artifact Registry.

Un **déploiement continu** est prévu. La cicd lancera différentes actions à différents niveaux, à savoir : niveau Github : des linters, et des tests unitaires seront lancés à ce niveau.

niveau GCP : une image docker sera construite à partir du Dockerfile puis déployée via CloudRun avec un accès public.

Le Dockerfile

Le container sera créé à partir d'une image docker python slim.

L'intégralité du dossier app sera copié dans l'image.

```
FROM python:3.9-slim as python

WORKDIR /app

RUN apt-get update -y && apt-get install git -y && \
    apt-get -y -q clean all && \
    rm -rf /var/lib/apt/lists/*

RUN apt-get update && apt-get install -y --no-install-recommends apt-utils
RUN apt-get -y install curl
RUN apt-get install libomp1

RUN useradd -ms /bin/bash user && \
    chown -R 1000:3000 /app

# 1. Install Poetry

## Poetry env variables
ENV PYTHONUNBUFFERED=true \
    POETRY_HOME=/opt/poetry \
    POETRY_VIRTUALENVS_CREATE=false \
    POETRY_VERSION="1.2.2"

## install poetry
RUN python -c 'from urllib.request import urlopen; print(urlopen("https://install.python-poetry.org").read().decode())' | python -

## Add poetry to Path
ENV PATH="$POETRY_HOME/bin:$PATH"

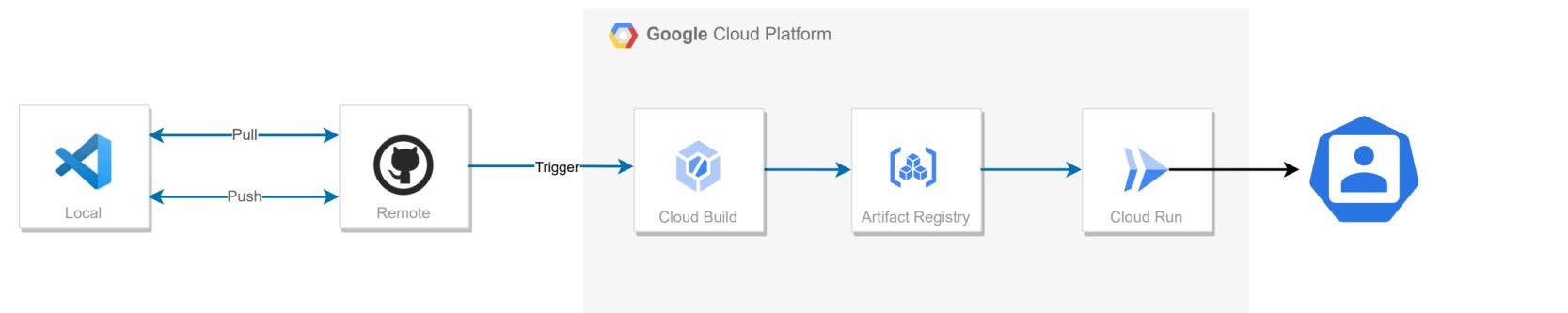
# 2. Install analytics package
COPY pyproject.toml poetry.lock ./ 
RUN poetry install

EXPOSE 8501

COPY . . Alhassane, il y a 12 heures • v1 ...

HEALTHCHECK CMD curl --fail http://localhost:8501/_stcore/health
```

Le Déploiement continu



Un déploiement continu depuis **Github** permettra d'assurer facilement l'intégration des changements.

Un changement approuvé et mergé depuis GitHub déclanchera la CICD qui lancera une série d'action à savoir :

- des tests unitaires et d'intégration
- des checks linter pour la qualité du code
- compilation puis déploiement niveau GCP

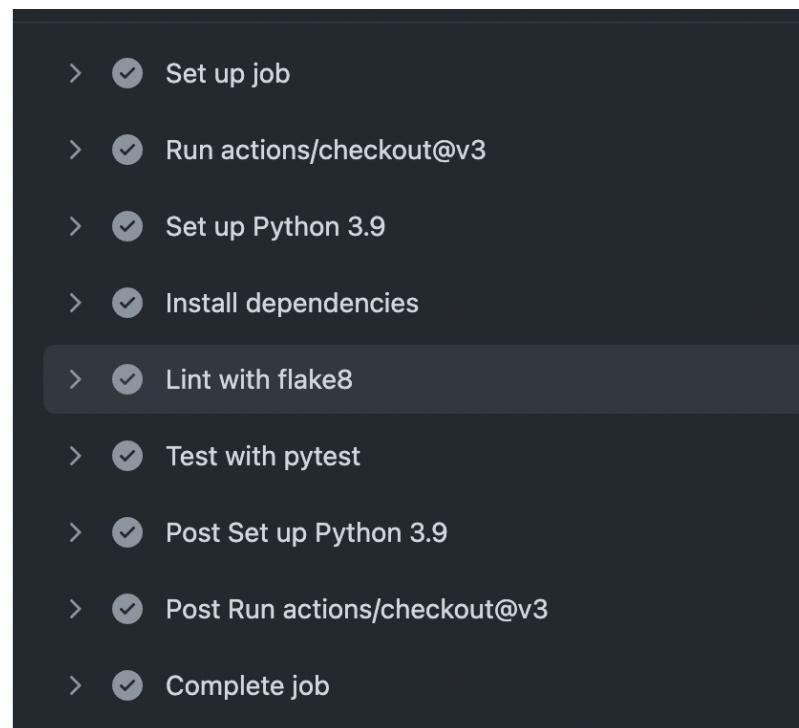
Si le code ne répond pas aux exigences de Black python ainsi que que des tests unitaires définis, la CI échouera et il sera impossible de merger niveau git, tout comme en cas d'erreur niveau compilation la ci s'arrêtera: il n'y aura donc pas de déploiement...

CICD Pipelines

A chaque modification, deux pipelines sont lancées . Ces pipelines sont gérées via des fichiers de configuration. Voir l'exemple de la pipeline du linter black à cet emplacement : .github/workflow/black.yaml)..

Niveau Github avec GithubActions

Deux pipelines **se lance à chaque pull request**. Elles ont pour objectif de tester la qualité du code avec **black** et **flake8** et de lancer des **tests unitaires** pour vérifier le nombre de training features puis la présence des colonnes nécessaires pour afficher les informations personnelles du client. (voir le dossier app/tests). Pour pourvoir merger une PR, toutes les étapes de la pipeline doivent être validées.



Pour pourvoir merger une PR, toutes les étapes de la pipeline doivent être validées.

Require status checks to pass before merging
Choose which [status checks](#) must pass before branches commits must first be pushed to another branch, then m

Niveau GCP avec CloudBuild

Un trigger Cloudbuild est lancé aussitôt qu'une branch est mergeé. Celui-ci lance les actions suivantes :

- Build : construit l'image docker à partir du fichier Dockerfile à l'emplacement : loan-scoring/app/Dockerfile
- Push : envoi l'image dans artifact registry
- Deploy : deploie via le service CloudRun

	Build Summary	00:06:57
3 Steps		
	0: Build build --no-cache -t euro...	00:02:54
	1: Push push europe-west1-doc...	00:02:00
	2: Deploy gcloud run services upd...	00:01:47



La mise à jour est déployée dès que toutes les étapes de la pipeline sont validées.

CONCLUSION

Ce projet a été lancé dans l'objectif d'assister la prise de décision sur l'octroi des crédits et de favoriser la transparence dans la communication du résultat de cette décision au client .

En partant des données fournies, le modèle de classification le plus performant possible a été conçu. Celui-ci est de type boosting : **lightGBM**.

Une application web destinée aux chargés de relation client, se basant sur le modèle conçu et permettant de classifier le client en bon ou mauvais payeur a ensuite été créée. Cette même application fournie des informations sur l'historique de prêt du client, sur sa situation professionnelle et même personnelle.... Ainsi, le chargé de relation disposera de toutes les informations nécessaires pour répondre au client en toute transparence.

Il est important de noter qu'en plus de l'aspect transparence dans la communication, l'application peut aussi faciliter la prise de décision sur l'accord de prêt grâce Credit Score qui indique la probabilité qu'un client puisse rembourser.

Le Repo Github : <https://github.com/Alhasdata/loan-scoring>

La Web App : <https://loan-scoring-3hnypmlvvq-ew.a.run.app>

**N'hésitez pas à m'envoyer un
message pour toute question ou tout
commentaire**

Merci !