Prêt à dépenser

# Implémentez un modèle de scoring

Prêt à dépenser

# Sommaire

Prêt à dépenser

# Présentation de la problématique

L'entreprise "Prêt à dépenser" propose des crédits à la consommation à des personnes ayant peu ou pas d'historique de prêt. Ici, la problématique va être de chercher à évaluer la probabilité de remboursement de chaque client et ainsi classer les demandes en crédits accordés ou refusés. C'est pourquoi, l'entreprise souhaite développer un algorithme de classification. Pour cela, elle se basera sur des sources de données variées, telles que les données comportementales et celles provenant d'autres institutions financières.
Comme les clients sont de plus en plus demandeurs de transparence en ce qui concerne les décisions d'octroi de crédit, l'entreprise a décidé de développer un dashboard interactif. Ce tableau de bord offrira une transparence maximale aux clients et permettra aux chargés de relation client d'expliquer facilement les décisions d'octroi de crédit.

Prêt à dépenser

## Jeu de données mis à notre disposition:

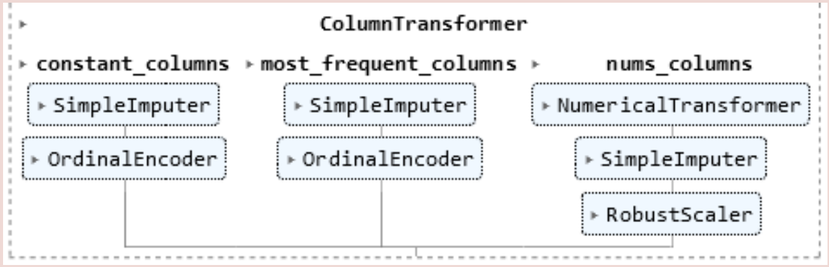
- 300 000 clients
- + de 300 variables
- Présente un gros déséquilibre

Prêt à dépenser

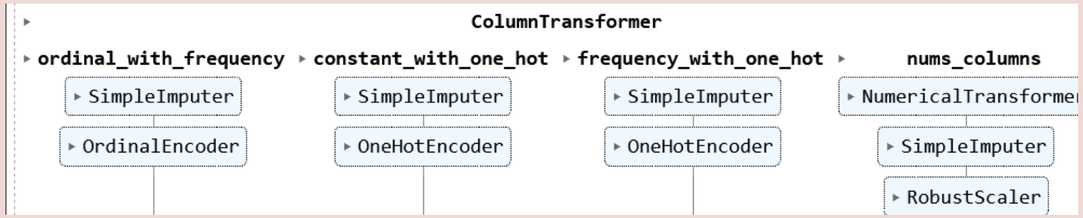
# Présentation de la modélisation

# Gestion des valeurs manquantes

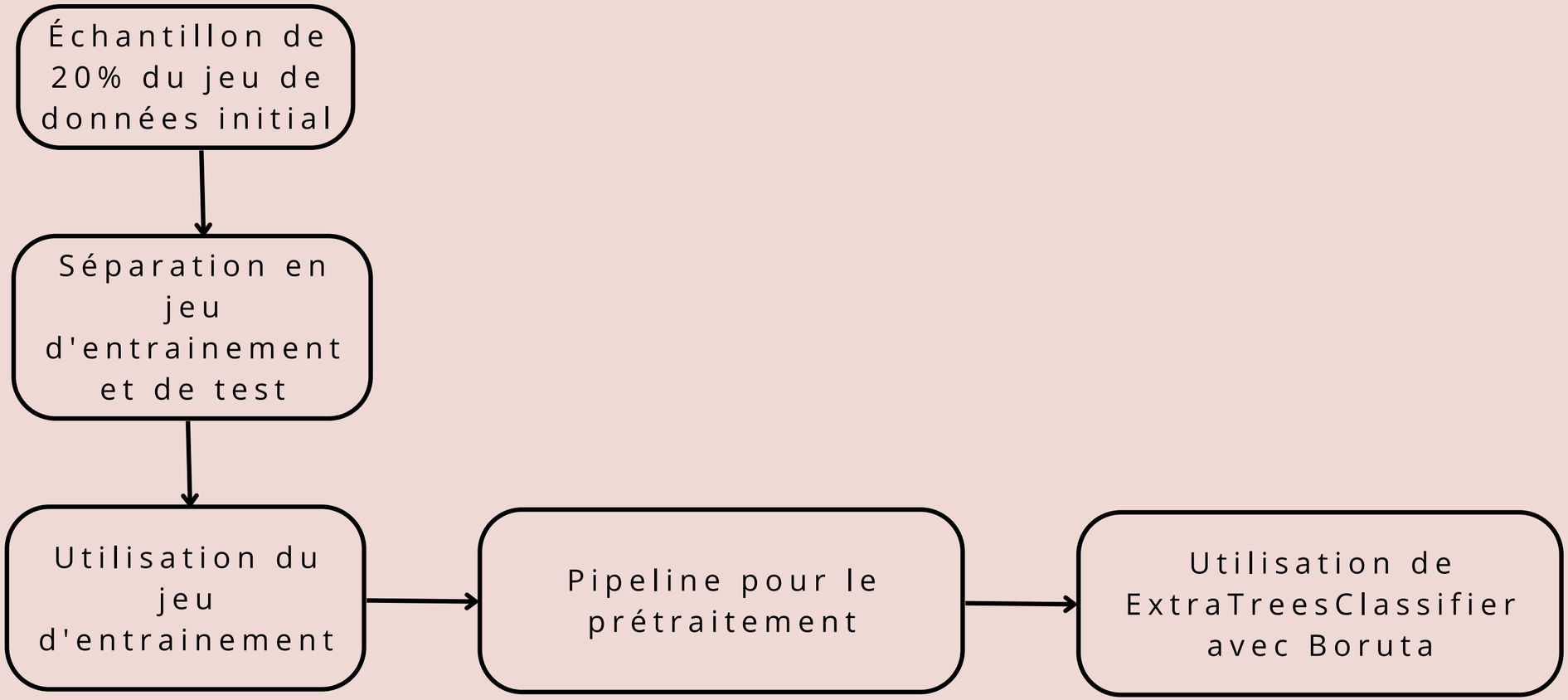
## Modèle basée sur les arbres de décision




## Modèle linéaire



# Démarche pour la sélection de features



# Métrique choisis

| | Pred Negative class | Pred Positive class |
|---|---|---|
| Negative class (Bon payeur) | TN (Bon payeur bien prédit) | FP (Bon payeur prédit mauvais payeur) |
| Positive class (Mauvais payeur) | FN (Mauvais payeur prédit bon payeur) | TP (Mauvais payeur bien prédit) |

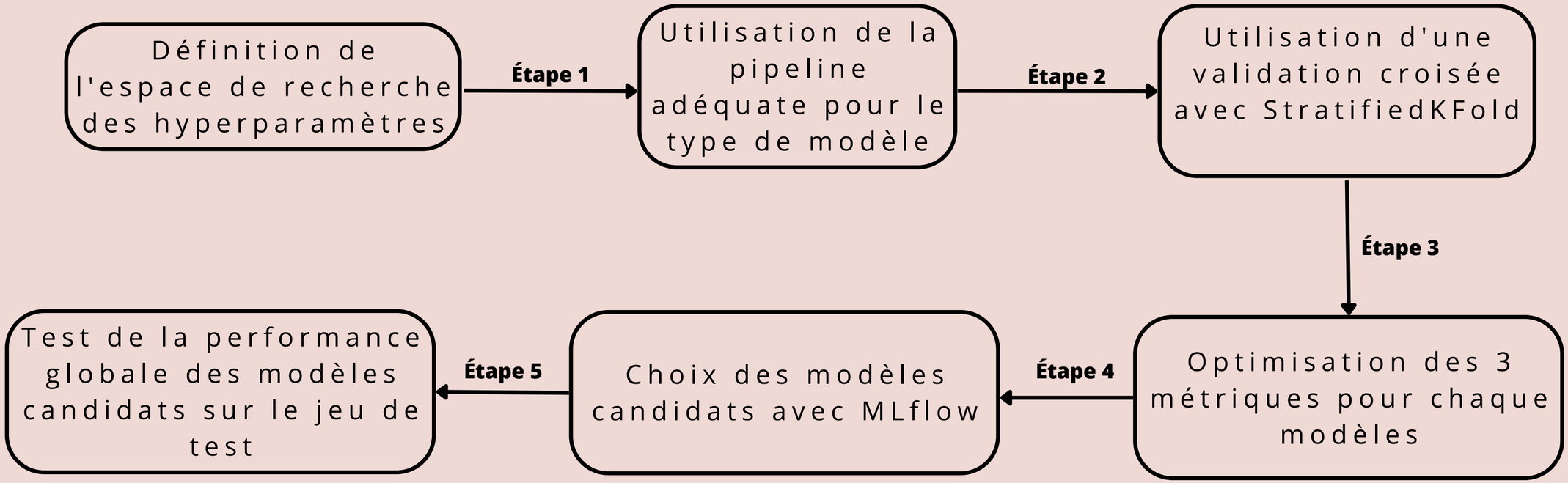
## Métrique technique choisis

- **L'aire sous la courbe (AUC)**: mesure la capacité du modèle à distinguer les mauvais payeurs (TP) des faux mauvais payeurs (FP)
- **Le fbeta score**: combine la précision et le rappel tout en accordant un poids à l'une ou l'autre de ces métriques. Un bêta de 2 sera défini.

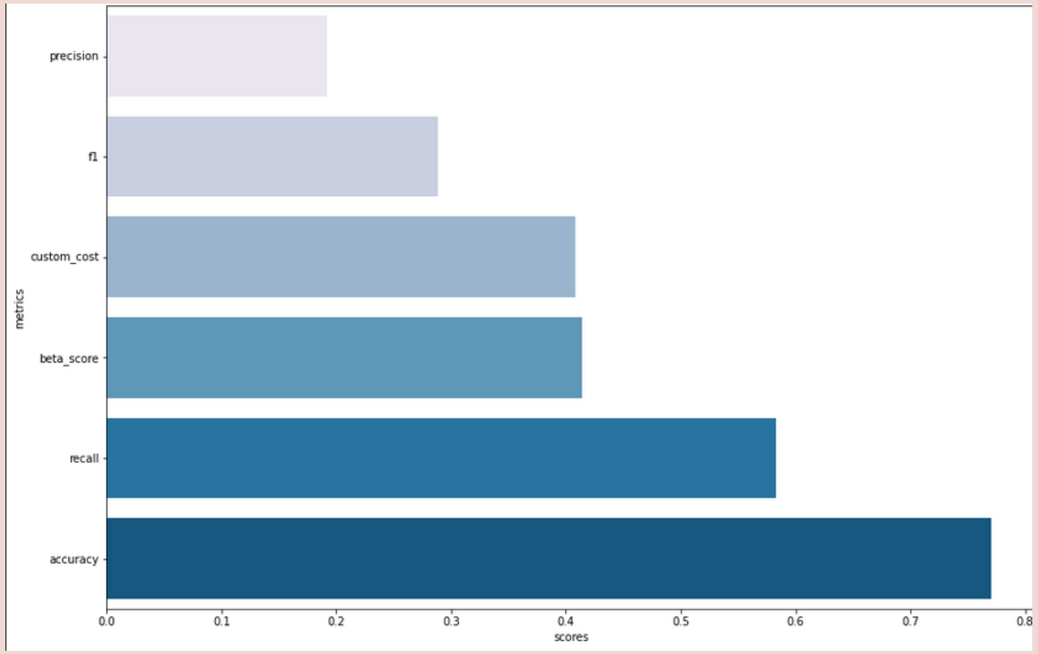
## Métrique métier

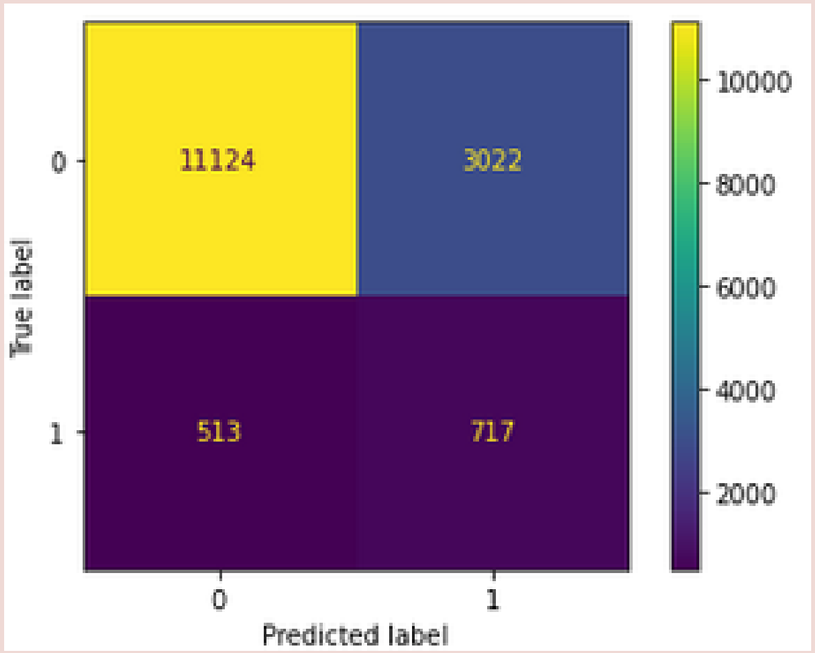
```python
def cost_function(y, y_pred, cost):

  Conf= confusion_matrix(y, y_pred)

  tn, fp, fn, tp = Conf[0,0], Conf[0,1], Conf[1,0], Conf[1,1]

  return (cost * fn + fp) / (tn + cost * fn + fp + tp)
```

# Démarche pour le choix du modèle finale



# Choix du seuil




- Seuil: 0.35

# Interprétation globale

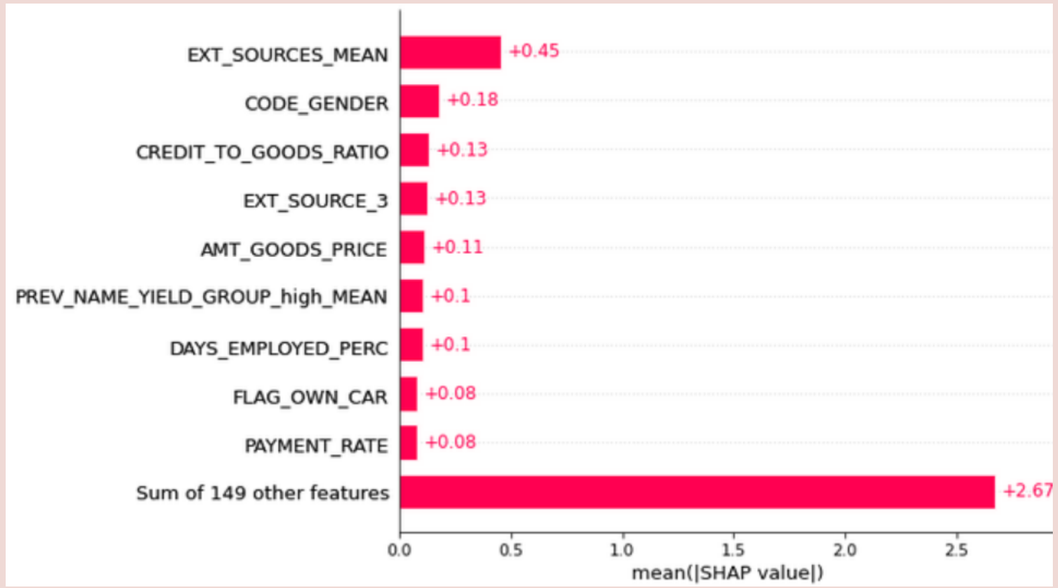



Figure 1

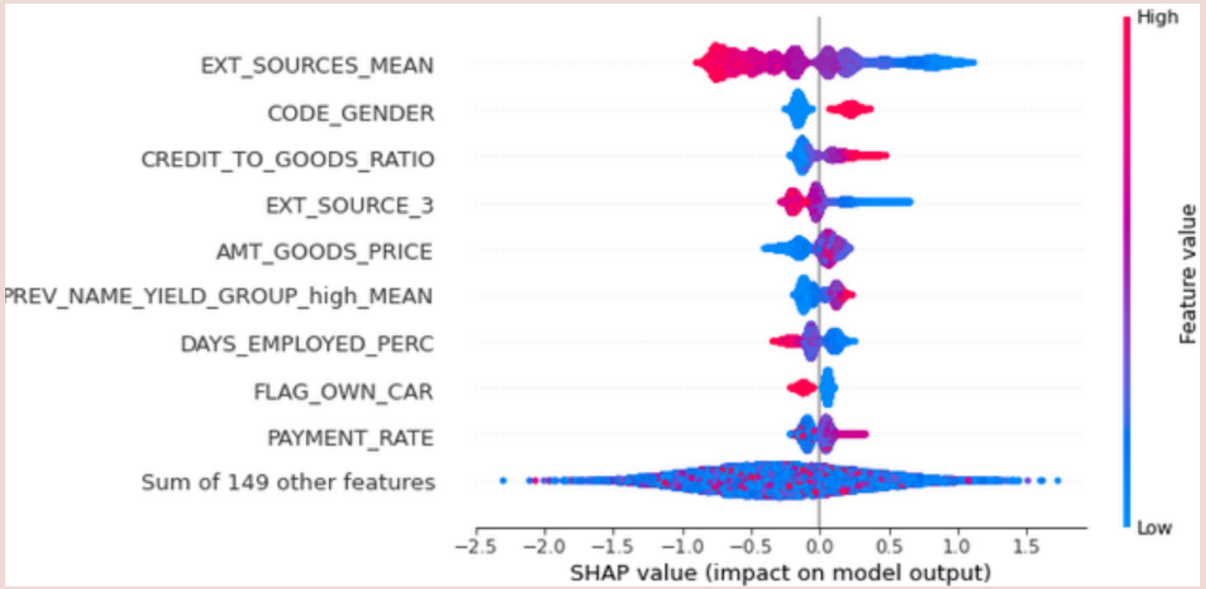
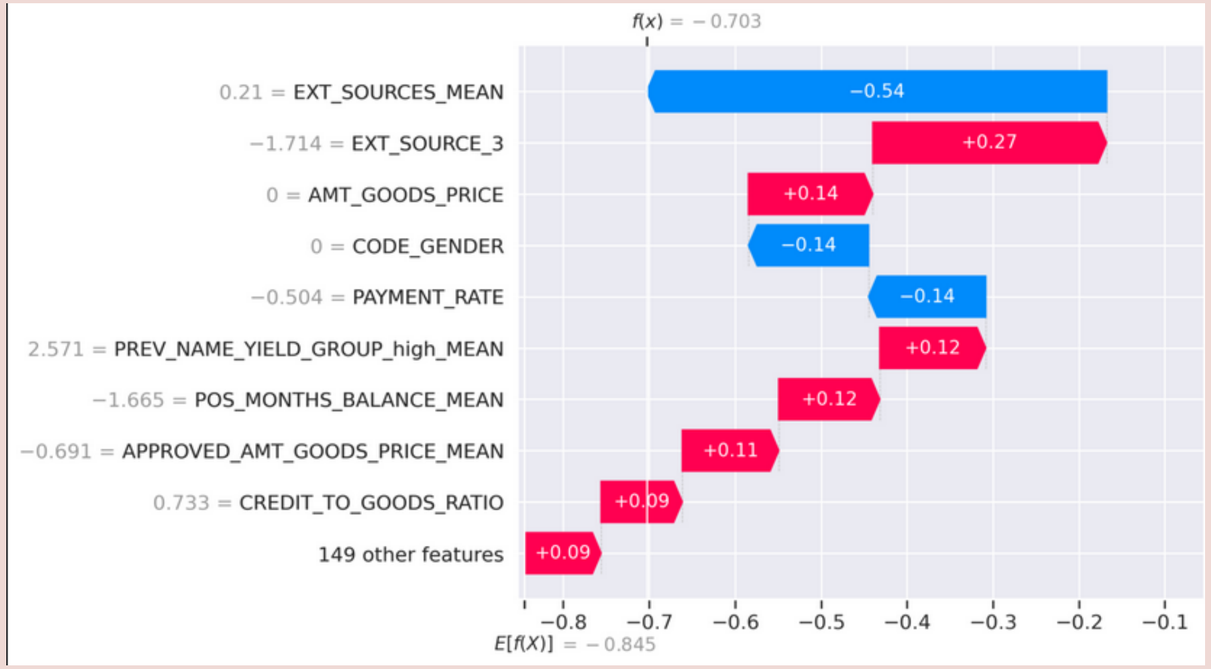



Figure 2

# Interprétation locale



Prêt à dépenser

# Analyse du data drift

# Analyse du data drift

## Dataset Drift

Dataset Drift is NOT detected. Dataset drift detection threshold is 0.5

| 158 | 12 | 0.0759 |
|---|---|---|
| Columns | Drifted Columns | Share of Drifted Columns |

### Data Drift Summary

Drift is detected for 7.595% of columns (12 out of 158).

Search

| Column | Type | Reference Distribution | Current Distribution | Data Drift | Stat Test | Drift Score |
|---|---|---|---|---|---|---|
| PAYMENT_RATE | num |  |  | Detected | Wasserstein distance (normed) | 0.577878 |
| AMT_GOODS_PRICE | num |  |  | Detected | Wasserstein distance (normed) | 0.212276 |
| NAME_CONTRACT_TYPE | cat |  |  | Detected | Jensen-Shannon distance | 0.14782 |
| PREV_NAME_PORTFOLIO_XNA_MEAN | num |  |  | Detected | Wasserstein distance (normed) | 0.136548 |
| PREV_NAME_YIELD_GROUP_XNA_MEAN | num |  |  | Detected | Wasserstein distance (normed) | 0.133758 |
| FLAG_EMAIL | num |  |  | Detected | Jensen-Shannon distance | 0.124014 |
| PHONE_TO_BIRTH_RATIO | num |  |  | Detected | Wasserstein distance (normed) | 0.123733 |
| PREV_NAME_CONTRACT_STATUS_Approved_MEAN | num |  |  | Detected | Wasserstein distance (normed) | 0.117387 |
| CLOSED_DAYS_CREDIT_UPDATE_MEAN | num |  |  | Detected | Wasserstein distance (normed) | 0.112929 |
| PREV_NAME_PAYMENT_TYPE_XNA_MEAN | num |  |  | Detected | Wasserstein distance (normed) | 0.111889 |

Prêt à dépenser

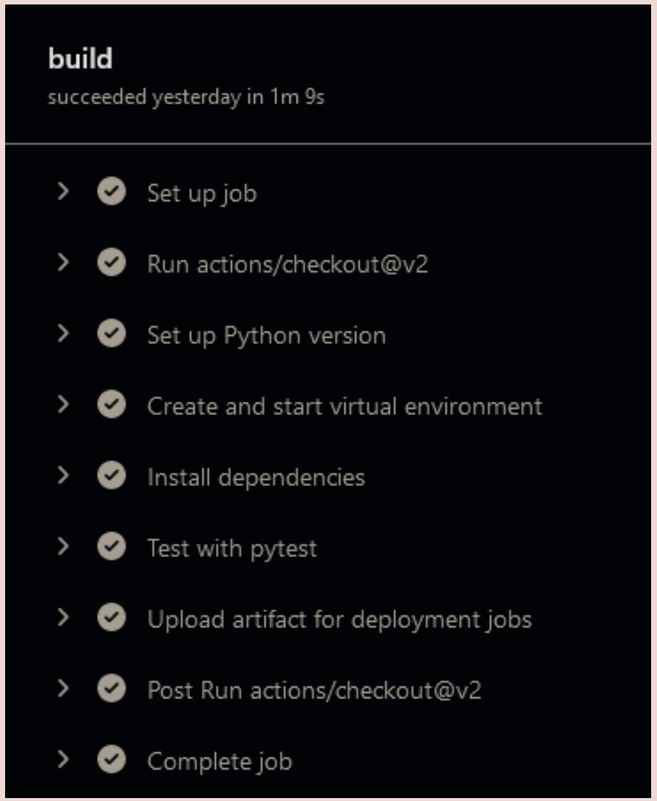
# Conclusion

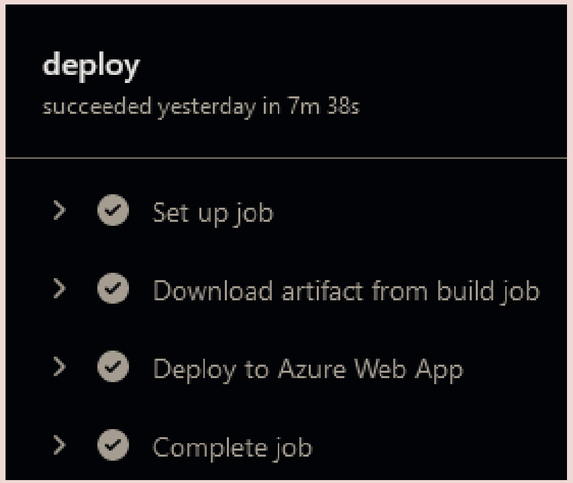
- Bonnes performances globales
- Mieux comprendre le jeu de données
- Tester d'autres méthodes de gestion des déséquilibres des classes
- Affiner la recherche d'hyperparamètres
- Affiner la métrique avec l'équipe métier

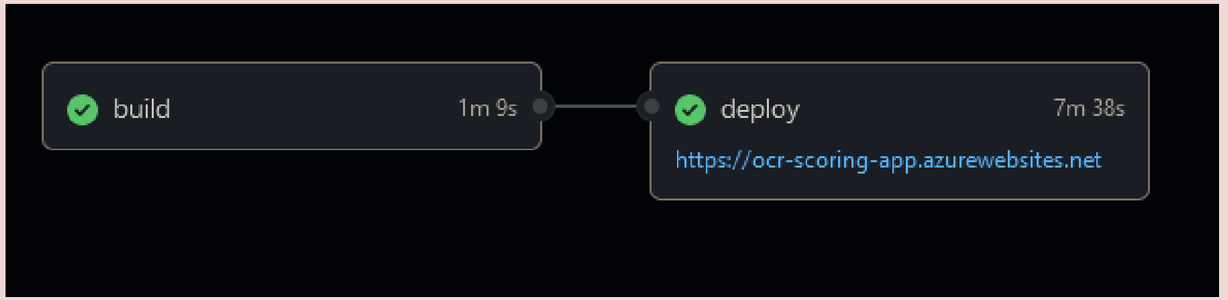
Prêt à dépenser

# Présentation de la pipeline de déploiement

# Github actions



place de marché

# Tests unitaires

```python
def test_check_columns():
    df = pd.read_json(test_data)
    columns = df.columns
    assert all(item in columns for item in columns_names)

def test_check_pipeline_predict():
    df = pd.read_json(test_data)
    y_pred = pipeline.predict(df)
    y_proba = pipeline.predict_proba(df)
    assert y_pred[0] in [0, 1]

    assert len([y_proba[0][1]]) == 1

    assert (y_proba[0][1] >= 0) & (y_proba[0][1] <= 1)

def test_shape_features_prep():
     df = pd.read_json(test_data)
     actual_shape = pipeline[0].transform(df).shape
     expected_shape = (1, 158)
     assert actual_shape == expected_shape

def test_check_trans_nums():
     df = pd.read_json(test_data)
     data = pipeline_nums.transform(df[nums_columns_name])
     assert data.shape == (1, 147)
```

Test with pytest

```
▶ Run pytest
============================= test session starts ==============================
platform linux -- Python 3.9.16, pytest-7.2.1, pluggy-1.0.0
rootdir: /home/runner/work/scoring-app/scoring-app
collected 4 items

tests/test_api.py ....                                                   [100%]

============================== 4 passed in 1.76s ===============================
```

Prêt à dépenser

# Démo du dashboard

Prêt à dépenser

MERCI DE VOTRE

# ATTENTION