AI Engineer – P04
OpenClassrooms

Construisez un modèle de scoring

Anthony VIDRAGO – Octobre 2024

Introduction

CONTEXTE DU PROJET

Nous travaillons au sein de la société « Prêt à Dépenser » qui propose des crédits à la consommation.

Nous sommes chargés de réaliser un outil de scoring crédit qui calculera la probabilité que le client rembourse son prêt ou non.

Michaël notre manager nous fournit différents jeux de données dont un qui fournit les indications de chaque colonne des différents dataset.

1er Notebook

Analyse Exploratoire
&
Feature Engineering

8

Jeux de données

+

un fichier descriptif nommé :

HomeCredit Colomns descriptions

Application Test

```
nb_lignes, nb_colonnes = data_test.shape
type_colonnes = data_test.dtypes.value_counts()
print(f"Nombre de lignes : {nb_lignes}")
print(f"Nombre de colonnes : {nb_colonnes}")
print(f"Type des colonnes : \n{type_colonnes}")
```

✓ 0.0s

```
Nombre de lignes : 48744
Nombre de colonnes : 121
Type des colonnes :
float64    65
int64      40
object     16
Name: count, dtype: int64
```

Application Train

```
nb_lignes, nb_colonnes = data_train.shape
type_colonnes = data_train.dtypes.value_counts()
print(f"Nombre de lignes : {nb_lignes}")
print(f"Nombre de colonnes : {nb_colonnes}")
print(f"Type des colonnes : \n{type_colonnes}")
```

✓ 0.0s

```
Nombre de lignes : 307511
Nombre de colonnes : 122
Type des colonnes :
float64    65
int64      41
object     16
Name: count, dtype: int64
```

Bureau Balance

```
nb_lignes, nb_colonnes = data_bb.shape
type_colonnes = data_bb.dtypes.value_counts()
print(f"Nombre de lignes : {nb_lignes}")
print(f"Nombre de colonnes : {nb_colonnes}")
print(f"Type des colonnes : \n{type_colonnes}")
```

✓ 0.0s

```
Nombre de lignes : 27299925
Nombre de colonnes : 3
Type des colonnes :
int64     2
object    1
Name: count, dtype: int64
```

Bureau

```
nb_lignes, nb_colonnes = data_bureau.shape
type_colonnes = data_bureau.dtypes.value_counts()
print(f"Nombre de lignes : {nb_lignes}")
print(f"Nombre de colonnes : {nb_colonnes}")
print(f"Type des colonnes : \n{type_colonnes}")
```

✓ 0.0s

```
Nombre de lignes : 1716428
Nombre de colonnes : 17
Type des colonnes :
float64    8
int64      6
object     3
Name: count, dtype: int64
```

Crédit Card Balance

```
nb_lignes, nb_colonnes = data_ccb.shape
type_colonnes = data_ccb.dtypes.value_counts()
print(f"Nombre de lignes : {nb_lignes}")
print(f"Nombre de colonnes : {nb_colonnes}")
print(f"Type des colonnes : \n{type_colonnes}")
```

✓ 0.0s

```
Nombre de lignes : 3840312
Nombre de colonnes : 23
Type des colonnes :
float64    15
int64       7
object      1
Name: count, dtype: int64
```

Installments Payments

```
nb_lignes, nb_colonnes = data_ip.shape
type_colonnes = data_ip.dtypes.value_counts()
print(f"Nombre de lignes : {nb_lignes}")
print(f"Nombre de colonnes : {nb_colonnes}")
print(f"Type des colonnes : \n{type_colonnes}")
```

✓ 0.0s

```
Nombre de lignes : 13605401
Nombre de colonnes : 8
Type des colonnes :
float64    5
int64      3
Name: count, dtype: int64
```

POS Cash Balance

```
nb_lignes, nb_colonnes = data_pcb.shape
type_colonnes = data_pcb.dtypes.value_counts()
print(f"Nombre de lignes : {nb_lignes}")
print(f"Nombre de colonnes : {nb_colonnes}")
print(f"Type des colonnes : \n{type_colonnes}")
```

✓ 0.0s

```
Nombre de lignes : 10001358
Nombre de colonnes : 8
Type des colonnes :
int64      5
float64    2
object     1
Name: count, dtype: int64
```

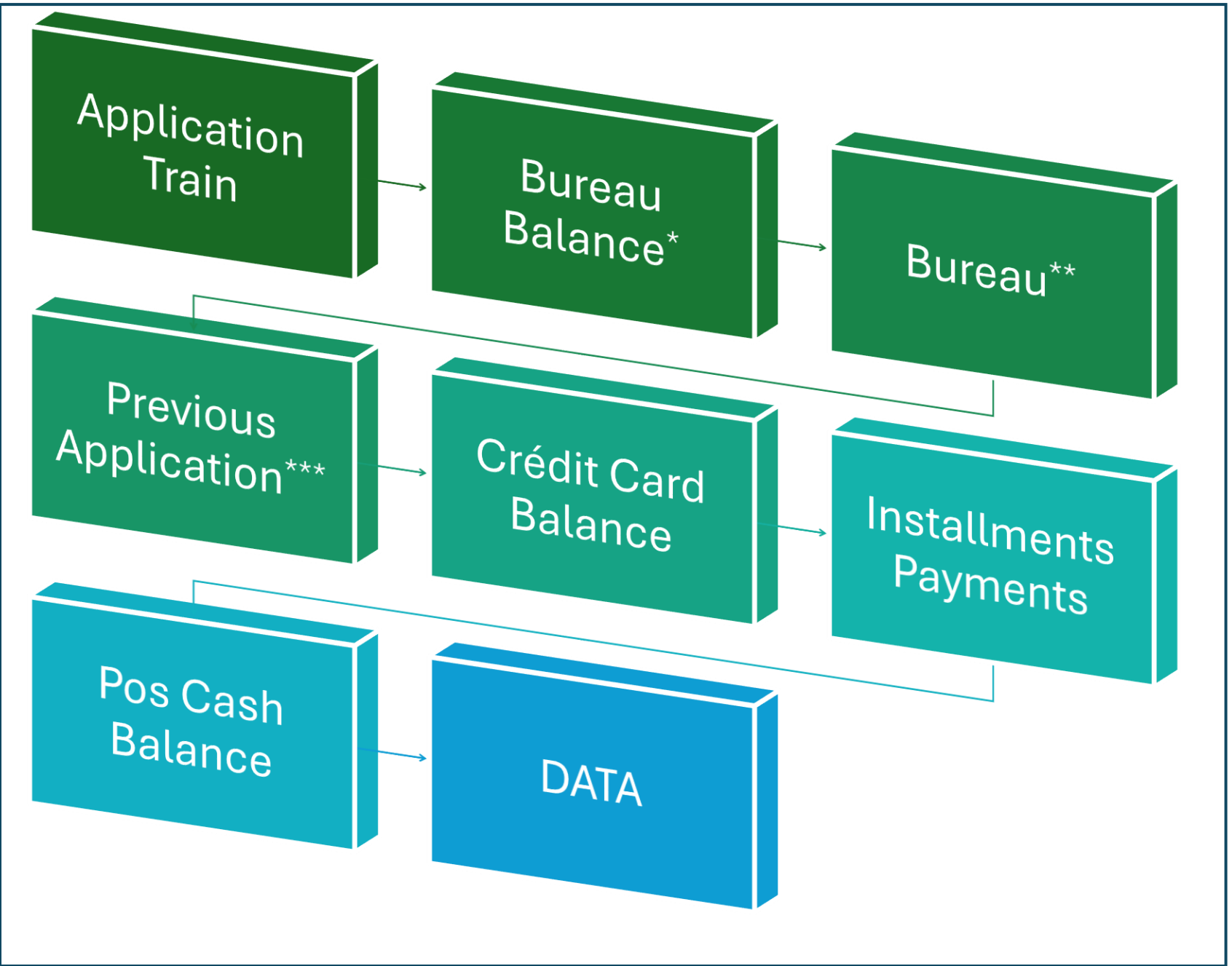
Previous Application

```
nb_lignes, nb_colonnes = data_app.shape
type_colonnes = data_app.dtypes.value_counts()
print(f"Nombre de lignes : {nb_lignes}")
print(f"Nombre de colonnes : {nb_colonnes}")
print(f"Type des colonnes : \n{type_colonnes}")
```

✓ 0.0s

```
Nombre de lignes : 1670214
Nombre de colonnes : 37
Type des colonnes :
object     16
float64    15
int64       6
Name: count, dtype: int64
```

Gestion des dataset



Transformation mathématique :

* / Calcul des crédits précédant pris par les clients

** / Calcul du soldes Mensuel moyen par clients

*** / Nombre de demandes de crédit immobilier

Ajout de nouvelles variables :

1 – Le taux d'endettement (DTI)

2 – La durée du paiement (Loan_Term)

3 – Revenu divisé par le nombre de membres du foyer (IFM)

Encodage de DATA

Encodage One Hot Encoding

```
# Sélection des colonnes restantes de type "object"
ohe_encode_cols = data.select_dtypes(include=["object"]).columns

# Appliquer un one-hot encoding
data = pd.get_dummies(data, columns=ohe_encode_cols)
```

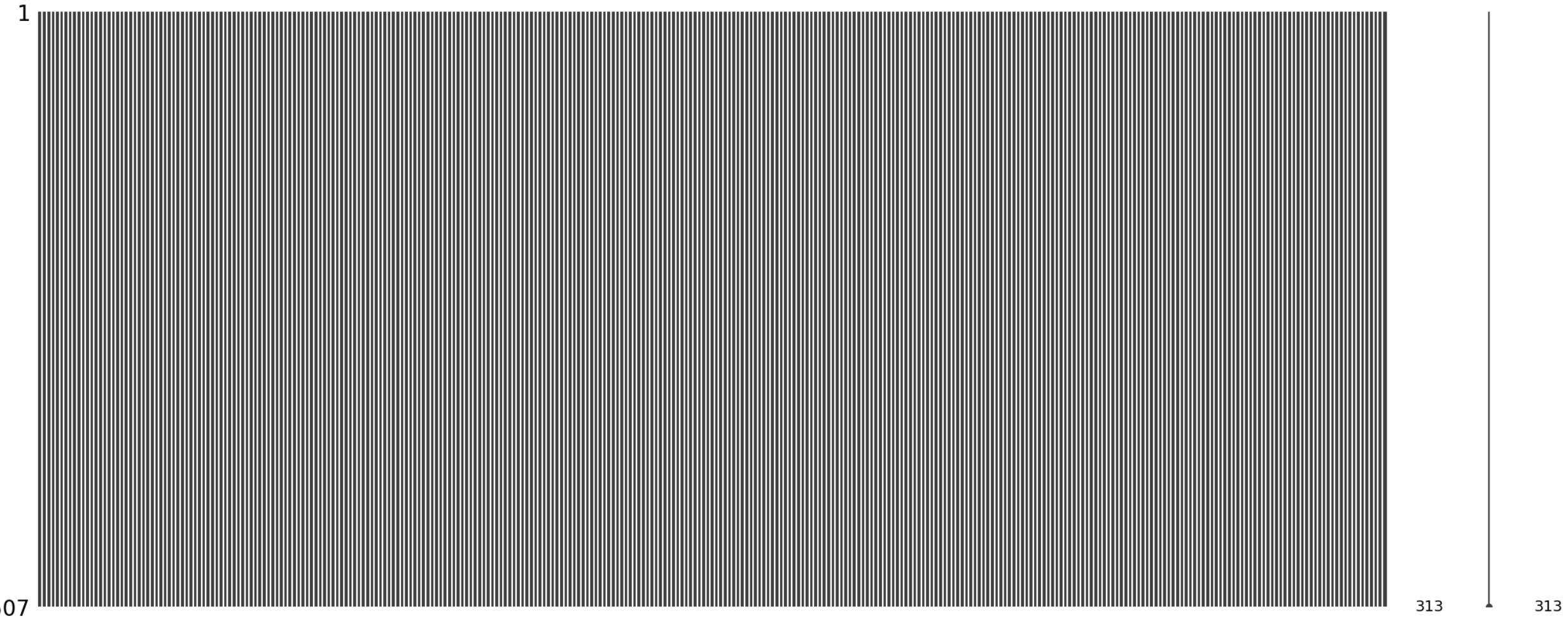
✓ 0.5s

Utilisation du One-Hot-Encoding de la librairie « Pandas » pour encoder nos variables catégorielles

Imputation des valeurs manquante dans DATA

```
# Imputation des valeurs manquantes
imputer = SimpleImputer(strategy="median")
data.iloc[:,:] = imputer.fit_transform(data)
```

✓ 9.7s



Normalisation de DATA

```
# Séparation des données
X = data.drop("TARGET", axis=1)
y = data["TARGET"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=77, stratify=y)

# Normalisation de X_train
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)

# Application de la transformation sur X_test
X_test_scaled = scaler.transform(X_test)
```

✓ 1.4s

MinMaxScaler a été utilisé pour normaliser les données, c'est utile pour les algorithmes sensibles à l'échelle des variables

```
nb_lignes, nb_colonnes = data.shape
type_colonnes = data.dtypes.value_counts()
print(f"Nombre de lignes : {nb_lignes}")
print(f"Nombre de colonnes : {nb_colonnes}")
print(f"Type des colonnes : \n{type_colonnes}")
```

✓ 0.0s

```
Nombre de lignes : 307507
Nombre de colonnes : 313
Type des colonnes :
float64    273
int64       40
Name: count, dtype: int64
```

Sélection des variables

Sélection de 30 variables les plus corrélés avec TARGET

```
var_corr = data[["TARGET",
    # Corrélations positives
    "PREV_DESK_MEAN_DAYS_CREDIT",
    "DAYS_BIRTH",
    "NAME_EDUCATION_TYPE_Secondary / secondary special",
    "NAME_INCOME_TYPE_Working",
    "PREV_DESK_MEAN_DAYS_CREDIT_UPDATE",
    "REGION_RATING_CLIENT_W_CITY",
    "REGION_RATING_CLIENT",
    "DAYS_LAST_PHONE_CHANGE",
    "FLAG_DOCUMENT_3",
    "PREV_APP_MEAN_DAYS_DECISION",
    "CODE_GENDER_M",
    "PREV_DESK_MEAN_DAYS_ENDDATE_FACT",
    "DAYS_ID_PUBLISH",
    "PREV_APP_MEAN_CC_MEAN_CNT_DRAWINGS_ATM_CURRENT",
    "OCCUPATION_TYPE_Laborers",

    # Corrélations négatives
    "EXT_SOURCE_2",
    "EXT_SOURCE_3",
    "EXT_SOURCE_1",
    "NAME_EDUCATION_TYPE_Higher education",
    "DAYS_EMPLOYED",
    "EMERGENCYSTATE_MODE_No",
    "ELEVATORS_AVG",
    "ELEVATORS_MEDI",
    "FLOORSMAX_AVG",
    "FLOORSMAX_MEDI",
    "CODE_GENDER_F",
    "ELEVATORS_MODE",
    "FLOORSMAX_MODE",
    "HOUSETYPE_MODE_block of flats",
    "NAME_INCOME_TYPE_Pensioner"
]]
```

✓ 0.0s

Suppression des variables redondante

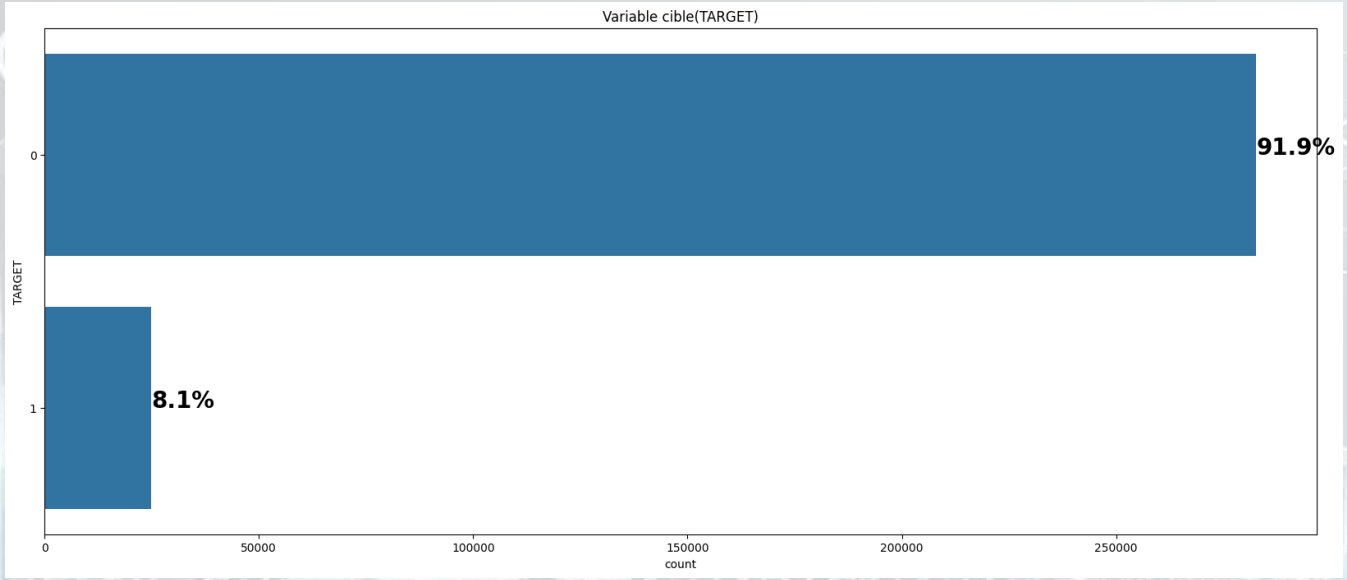
1. ELEVATORS_AVG / ELEVATORS_MEDI
2. ELEVATORS_AVG / ELEVATORS_MODE
3. ELEVATORS_MEDI / ELEVATORS_MODE
4. FLOORSMAX_AVG / FLOORSMAX_MEDI / FLOORSMAX_MODE
5. FLOORSMAX_MODE / HOUSETYPE_mode

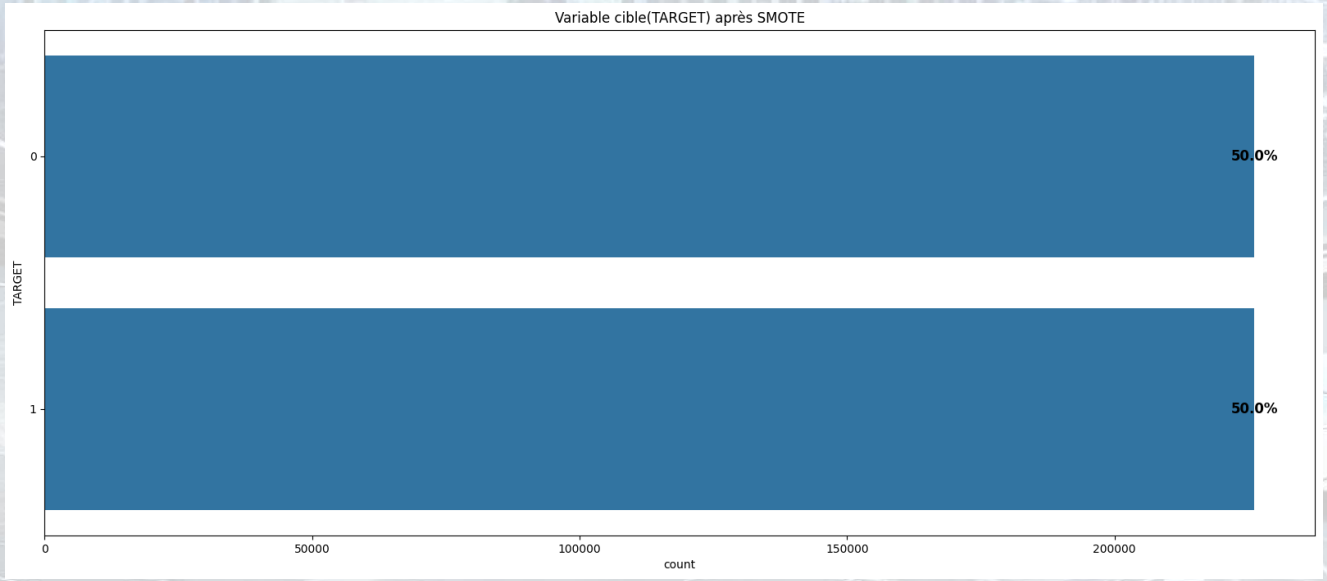
```
new_var = var_corr.drop(columns=[
    "FLOORSMAX_AVG",
    "FLOORSMAX_MEDI",
    "FLOORSMAX_MODE",
    "ELEVATORS_AVG",
    "ELEVATORS_MEDI",
    "ELEVATORS_MODE"
])
```

✓ 0.0s

Cette action est nécessaire afin d'éviter la multicolinéarité

Equilibrage de la variable cible





Utilisation du SMOTE, pour avoir une cible à 50/50 afin d'éviter une orientation du modèle vers l'indice « 0 »

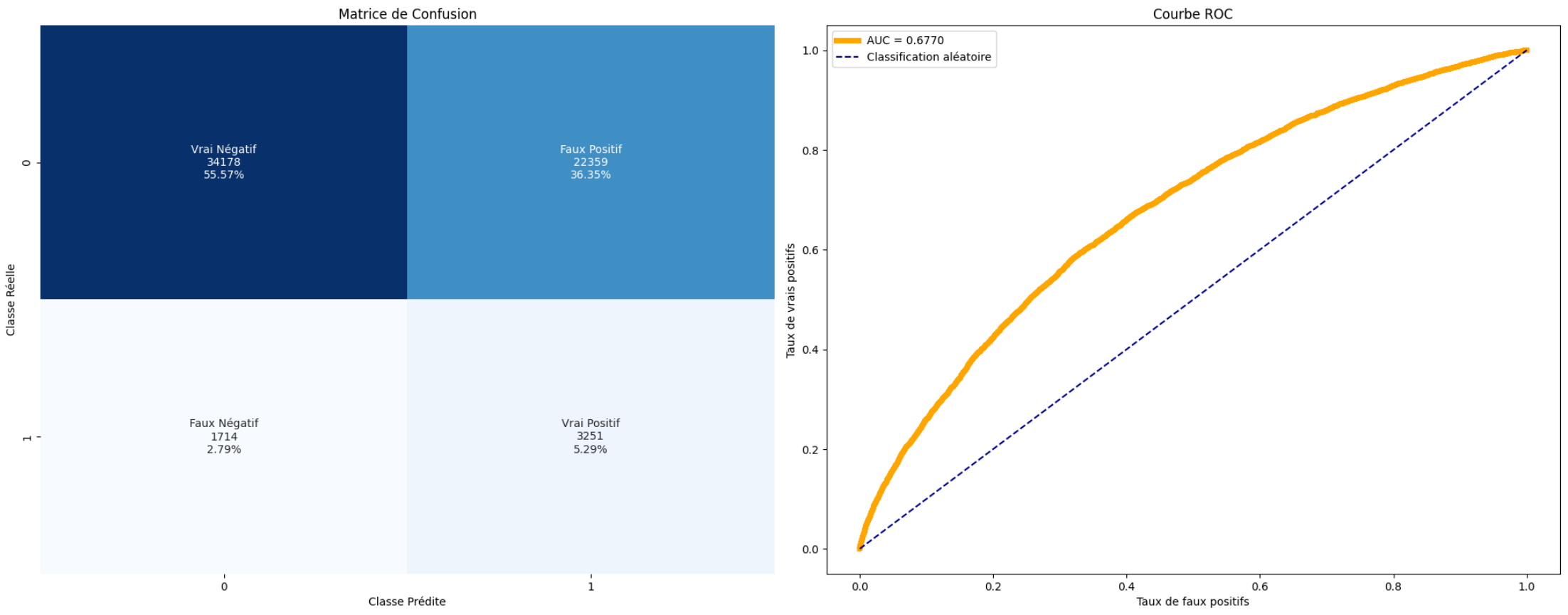
Conclusion notebook 1

- Exploration des données et correction des anomalies
- Assemblage des différents jeux de données
- Encodage et normalisation
- Imputation des valeurs manquante
- Sélection des variables corrélés et suppression des redondantes
- Equilibrage de la variable cible
- Cross validation avec StratifiedKfold

2ème Notebook

La Modélisation

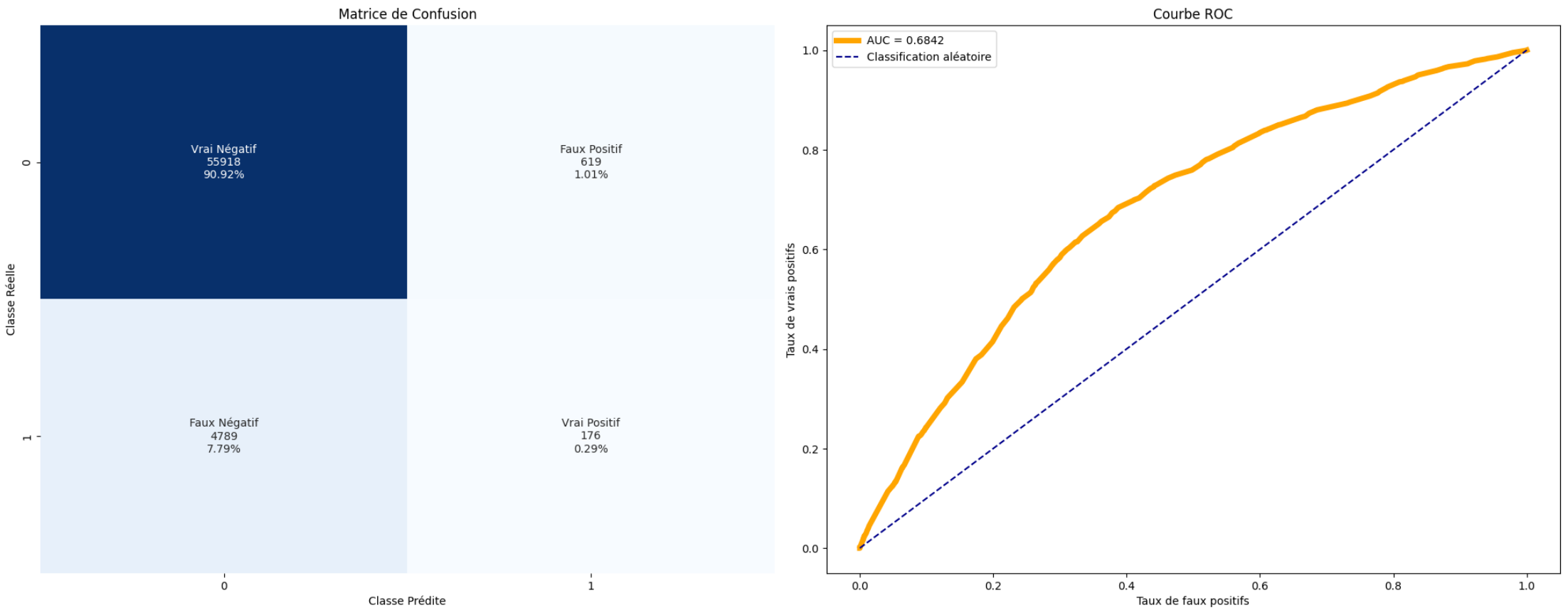
La Régression Logistique



AUC Mesure la capacité du modèle à distinguer les classes
Elle représente la probabilité que le modèle classe un échantillon positif aléatoire plus haut qu'un échantillon négatif aléatoire

La Courbe ROC, trace le taux de vrai positif en fonction du taux de faux positifs.

L’arbre de Décision



Les autres métriques restent utiles comme le recall, le f1-score, l’accuracy, mais elles sont sensibles au seuil choisi et peuvent être trompeuse dans des données déséquilibrées.

CatBoost

```python
# Définition de la grille d'hyperparamètres à tester
param_grid = {
    "iterations": [100, 200, 300],
    "learning_rate": [0.01, 0.1, 0.2],
    "depth": [4, 6, 8]
}

# Initialisation du modèle CatBoost
model = CatBoostClassifier(logging_level="Silent")

# Démarrage du chronomètre
debut = time.time()

# GridSearchCV pour trouver les meilleurs hyperparamètres
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring="roc_auc", n_jobs=-1)
grid_search.fit(X_train_sm, y_train_sm)

# Arrêt du chronomètre
fin = time.time()

# Calcul du temps d'exécution
temps_execution = fin - debut
print(f"Temps d'exécution pour la recherche des hyperparamètres : {temps_execution:.4f} secondes")

# Affichage des meilleurs hyperparamètres
print("Meilleurs hyperparamètres pour CatBoost:", grid_search.best_params_)
print("Meilleur score AUC:", grid_search.best_score_)
```

```
Temps d'exécution pour la recherche des hyperparamètres : 406.0631 secondes
Meilleurs hyperparamètres pour CatBoost: {'depth': 8, 'iterations': 300, 'learning_rate': 0.2}
Meilleur score AUC: 0.9640911891777749
```

XGBoost

```python
# Grille d'hyperparamètres à tester
param_dist = {
    "n_estimators": [50, 100, 200],
    "learning_rate": [0.01, 0.1, 0.2],
    "max_depth": [3, 5, 7],
    "colsample_bytree": [0.3, 0.5, 0.7],
    "subsample": [0.5, 0.7, 1.0]
}

# Initialisation du modèle XGBoost
xgb_model = XGBClassifier()

# Démarrage du chronomètre
debut = time.time()

# RandomizedSearchCV pour tester un échantillon aléatoire d'hyperparamètres
random_search = RandomizedSearchCV(estimator=xgb_model, param_distributions=param_dist, n_iter=10, cv=5, scoring="roc_auc", n_jobs=-1)
random_search.fit(X_train_sm, y_train_sm)

# Arrêt du chronomètre
fin = time.time()

# Calcul du temps d'exécution
temps_execution = fin - debut
print(f"Temps d'exécution pour la recherche des hyperparamètres : {temps_execution:.4f} secondes")

# Affichage des meilleurs hyperparamètres
print("Meilleurs hyperparamètres pour XGBoost:", random_search.best_params_)
print("Meilleur score AUC:", random_search.best_score_)
```

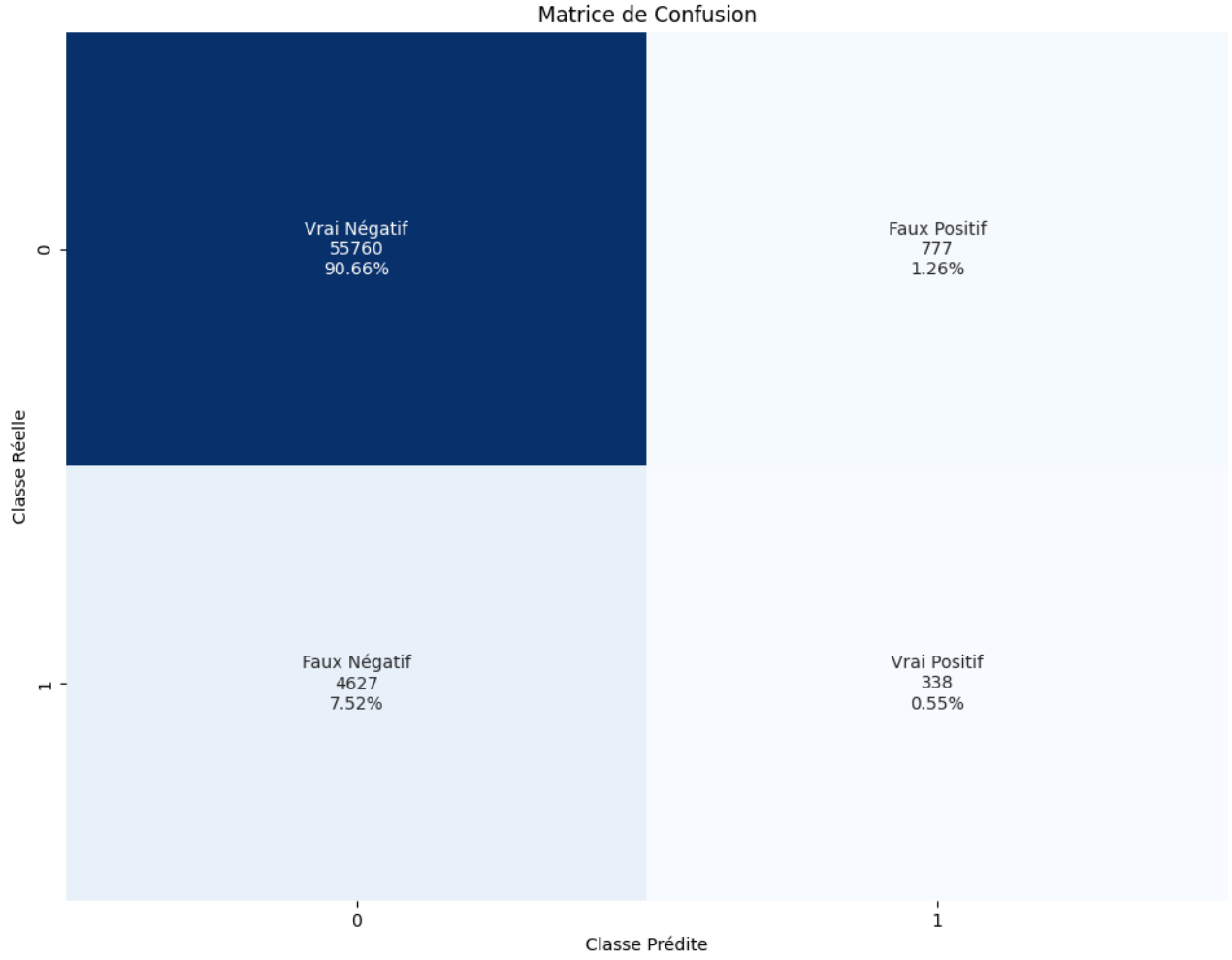
```
Temps d'exécution pour la recherche des hyperparamètres : 33.7570 secondes
Meilleurs hyperparamètres pour XGBoost: {'subsample': 0.7, 'n_estimators': 50, 'max_depth': 7, 'learning_rate': 0.2, 'colsample_bytree': 0.7}
Meilleur score AUC: 0.9614953968840062
```

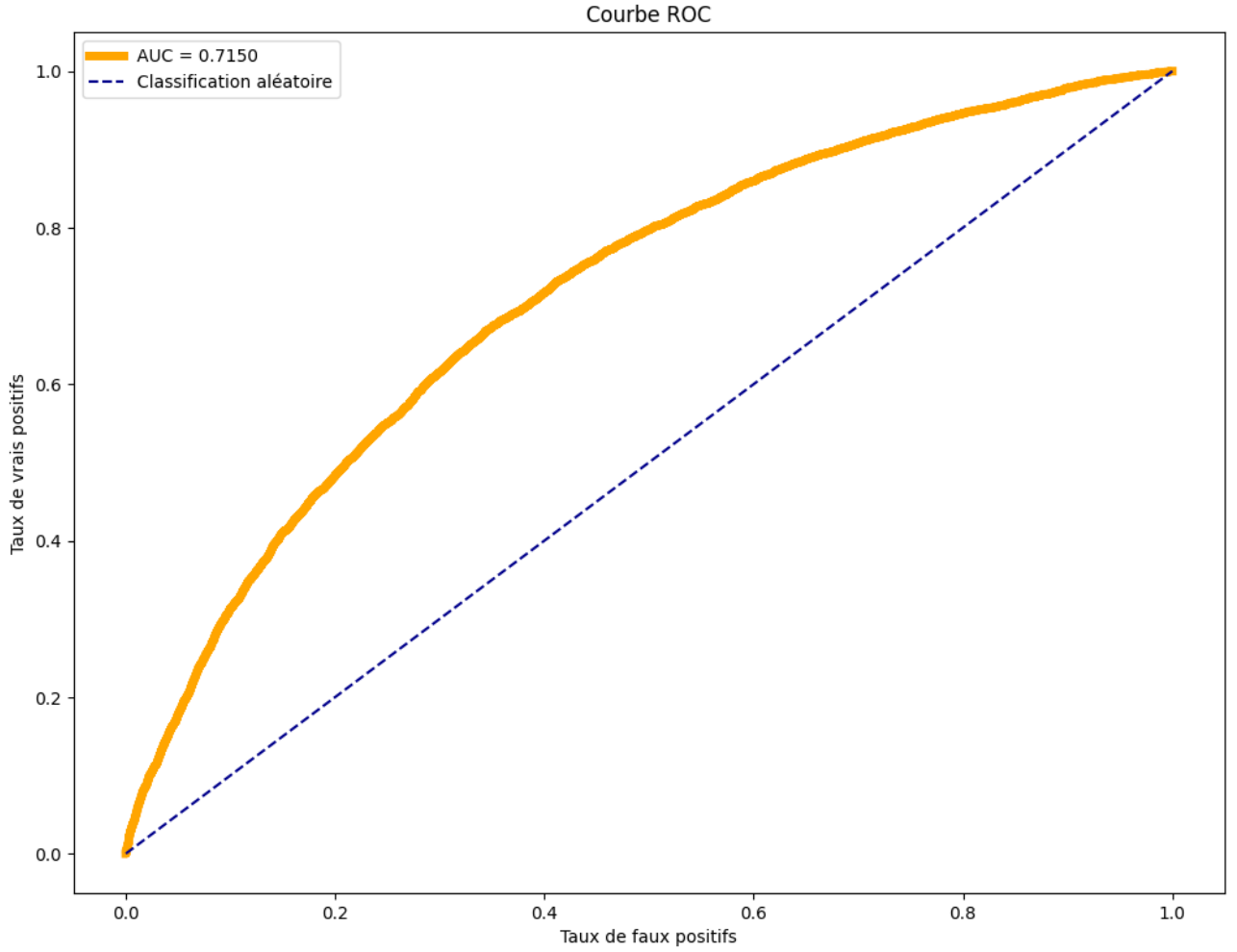
LightGBM

```python
# Définition de la grille d'hyperparamètres à tester
param_grid = {
    "n_estimators": [50, 100, 200],
    "learning_rate": [0.01, 0.1, 0.2],
    "num_leaves": [31, 50, 100],
    "boosting_type": ["gbdt", "dart"]
}

# Initialisation du modèle LightGBM
lgb_model = LGBMClassifier()

# Démarrage du chronomètre
debut = time.time()

# GridSearchCV pour trouver les meilleurs hyperparamètres
grid_search = GridSearchCV(estimator=lgb_model, param_grid=param_grid, cv=5, scoring="roc_auc", n_jobs=-1)
grid_search.fit(X_train_sm, y_train_sm)

# Arrêt du chronomètre
fin = time.time()

# Calcul du temps d'exécution
temps_execution = fin - debut
print(f"Temps d'exécution pour la recherche des hyperparamètres : {temps_execution:.4f} secondes")

# Affichage des meilleurs hyperparamètres
print("Meilleurs hyperparamètres pour LightGBM:", grid_search.best_params_)
print("Meilleur score AUC:", grid_search.best_score_)
```

```
Temps d'exécution pour la recherche des hyperparamètres : 467.0145 secondes
Meilleurs hyperparamètres pour LightGBM: {'boosting_type': 'gbdt', 'learning_rate': 0.2, 'n_estimators': 200, 'num_leaves': 100}
Meilleur score AUC: 0.9664713585069924
```

LightGBM


Matrice de Confusion
Vrai Négatif
55760
90.66%
Faux Positif
777
1.26%
Faux Négatif
4627
7.52%
Vrai Positif
338
0.55%
Classe Réelle
Classe Prédite
0
1


Courbe ROC
AUC = 0.7150
Classification aléatoire
Taux de vrais positifs
Taux de faux positifs
0.0
0.2
0.4
0.6
0.8
1.0

Cout métier

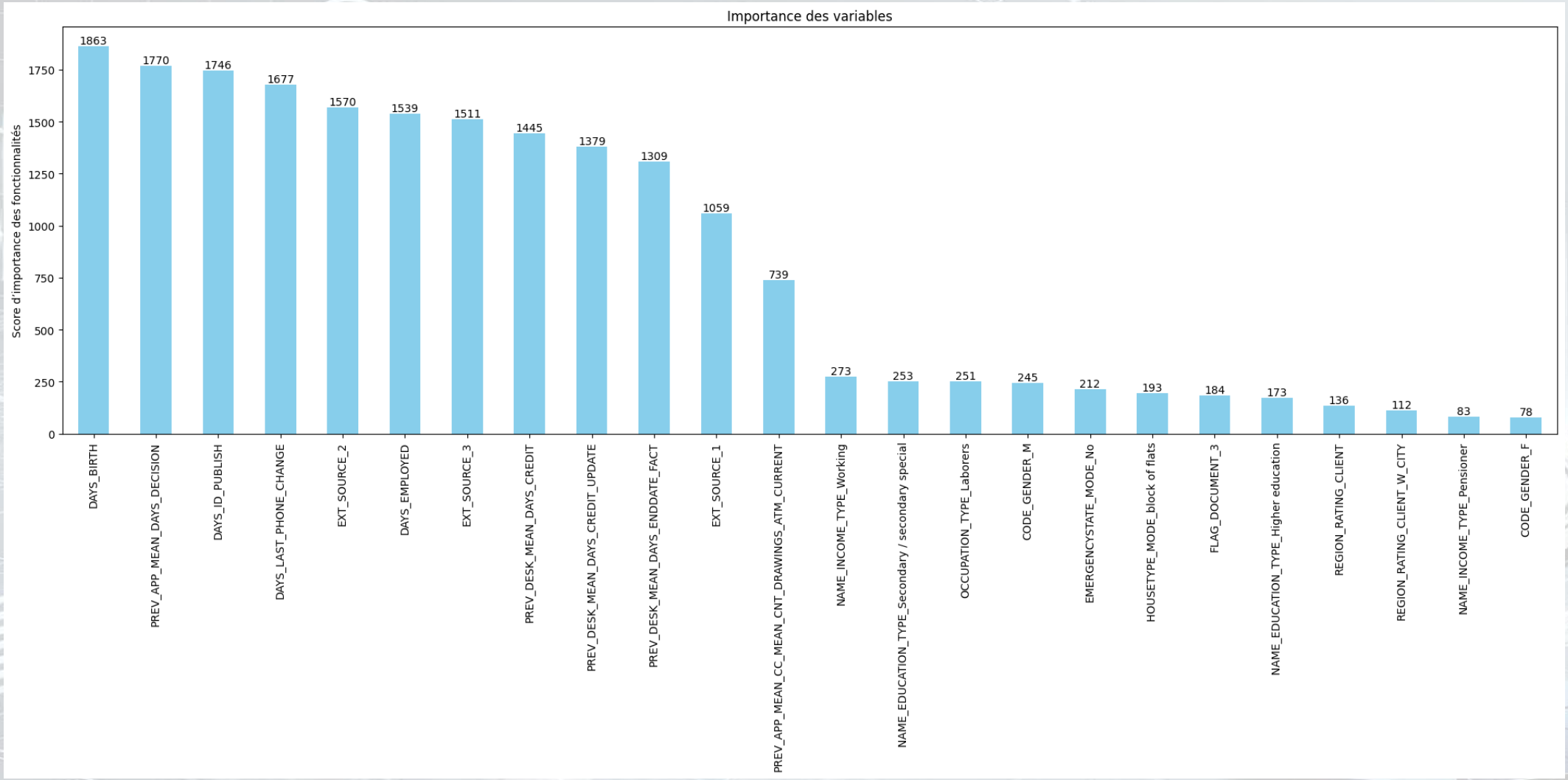
```
thresholds = np.linspace(0, 1, 100)
costs = []

for threshold in thresholds:
    cost = calculate_total_cost(y_test, y_pred_proba, threshold)
    costs.append(cost)

# Trouver le seuil avec le coût minimum
optimal_threshold = thresholds[np.argmin(costs)]
print(f"Seuil optimal : {optimal_threshold:.2f}")
```

```
Seuil optimal : 0.17
```

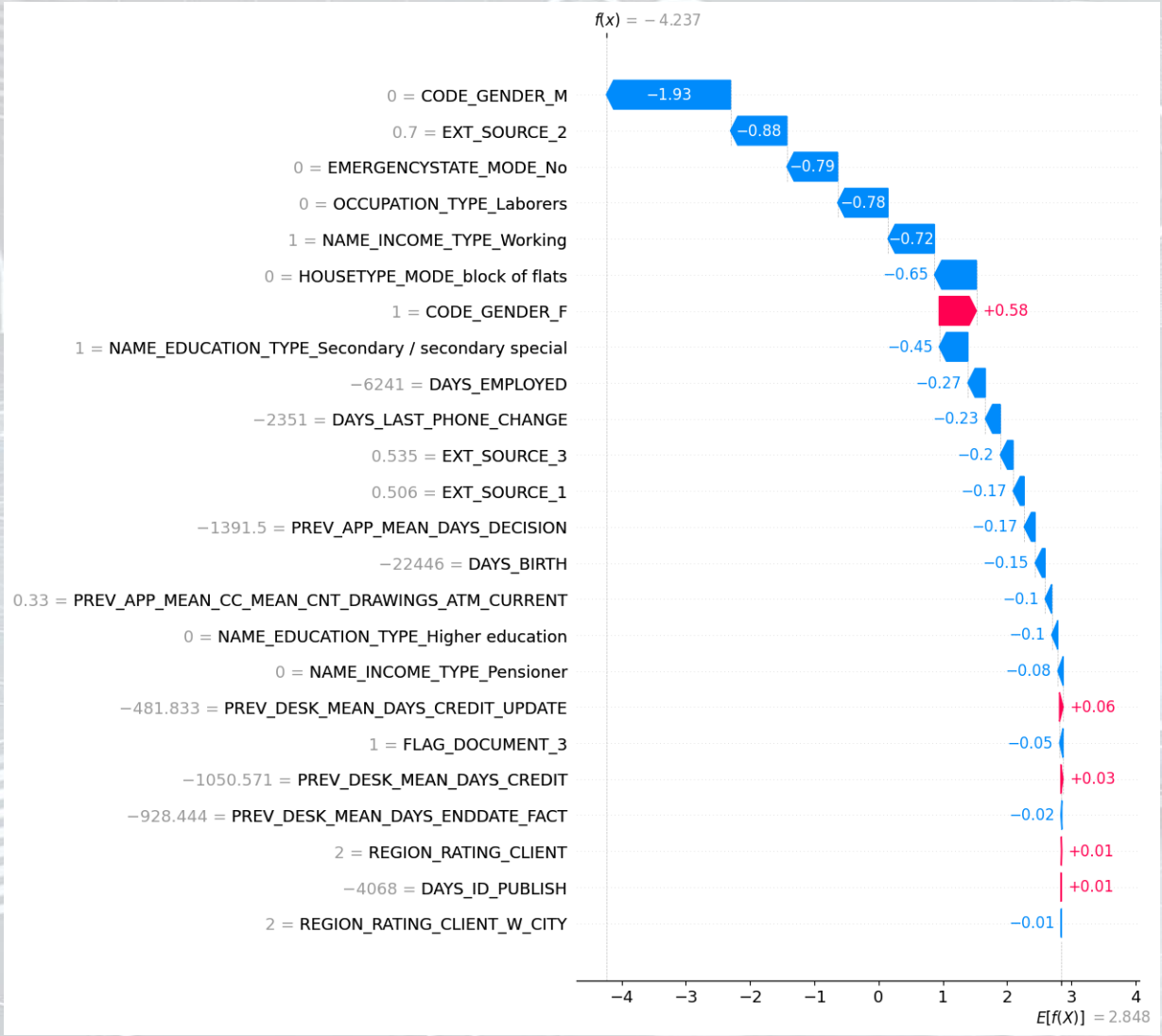
```
[ ] model = best_lgbm_model
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Calcul du coût total pour le modèle
total_cost = calculate_total_cost(y_test, y_pred_proba, optimal_threshold)
print(f"Coût total pour le modèle : {total_cost}")
```

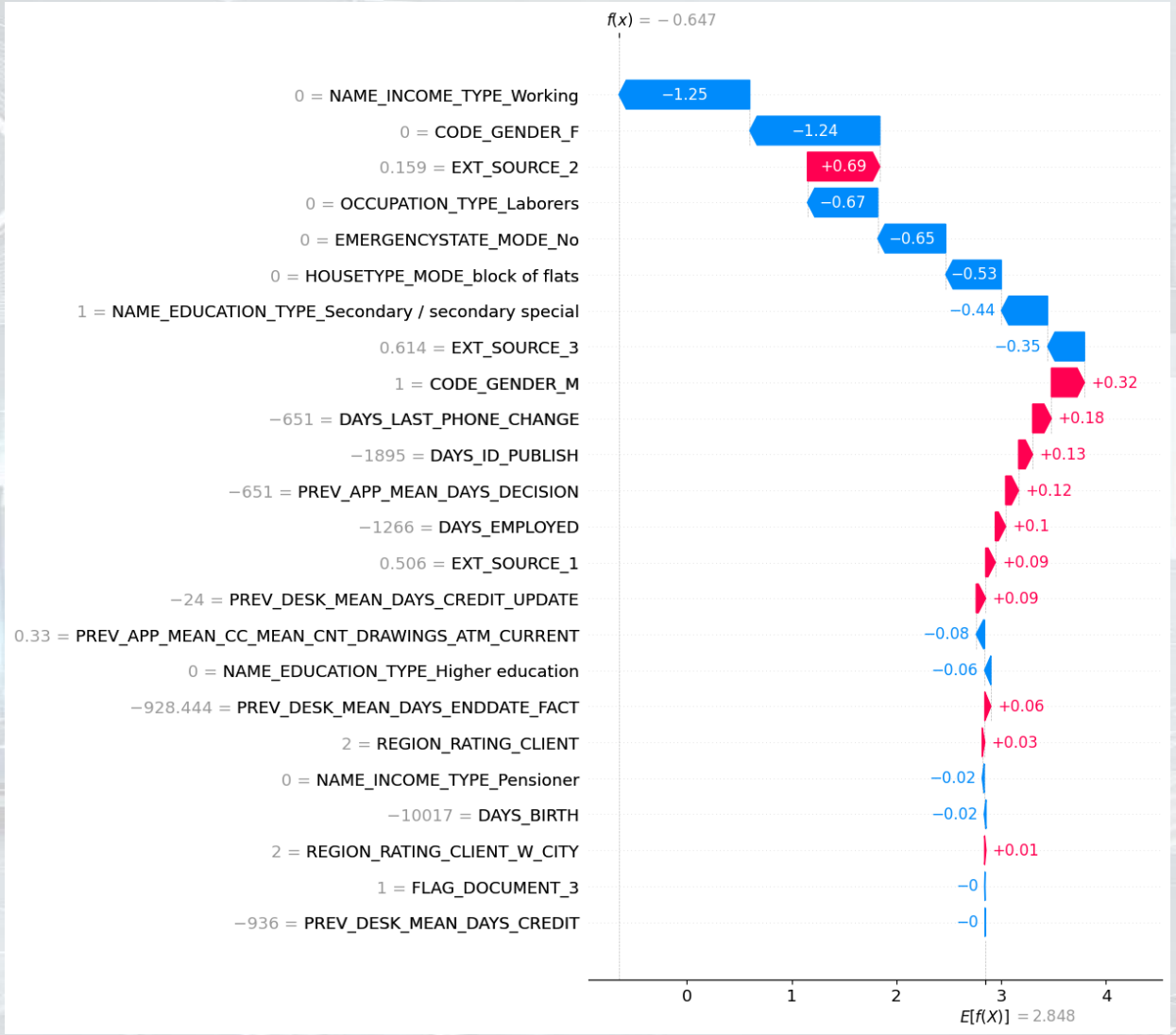
```
Coût total pour le modèle : 37465
```

Feature Importance



Feature Importance (Locale)





Conclusion notebook 2

Voici les modèles sélectionnés :

- **DummyClassifier** (Baseline)
- **La Régression Logistique** (modèle linéaire)
- **L'arbre de Décision** (modèle non linéaire)
- **Catboost** (gradient boosting)
- **LightGBM** (gradient boosting)
- **XGBoost** (gradient boosting)

- **Fonction Cout métier**
- **Feature Importance** (globale & locale)

MERCI