

A 3D rendering of a two-story house with a brown roof and a balcony, held in a person's hand. The house is white with brown trim and has a chimney. The background is a blurred image of a person's hand holding the house. The entire scene is framed by a thin red circular line.

Scoring bancaire

Samuel Jacquot

Problématique

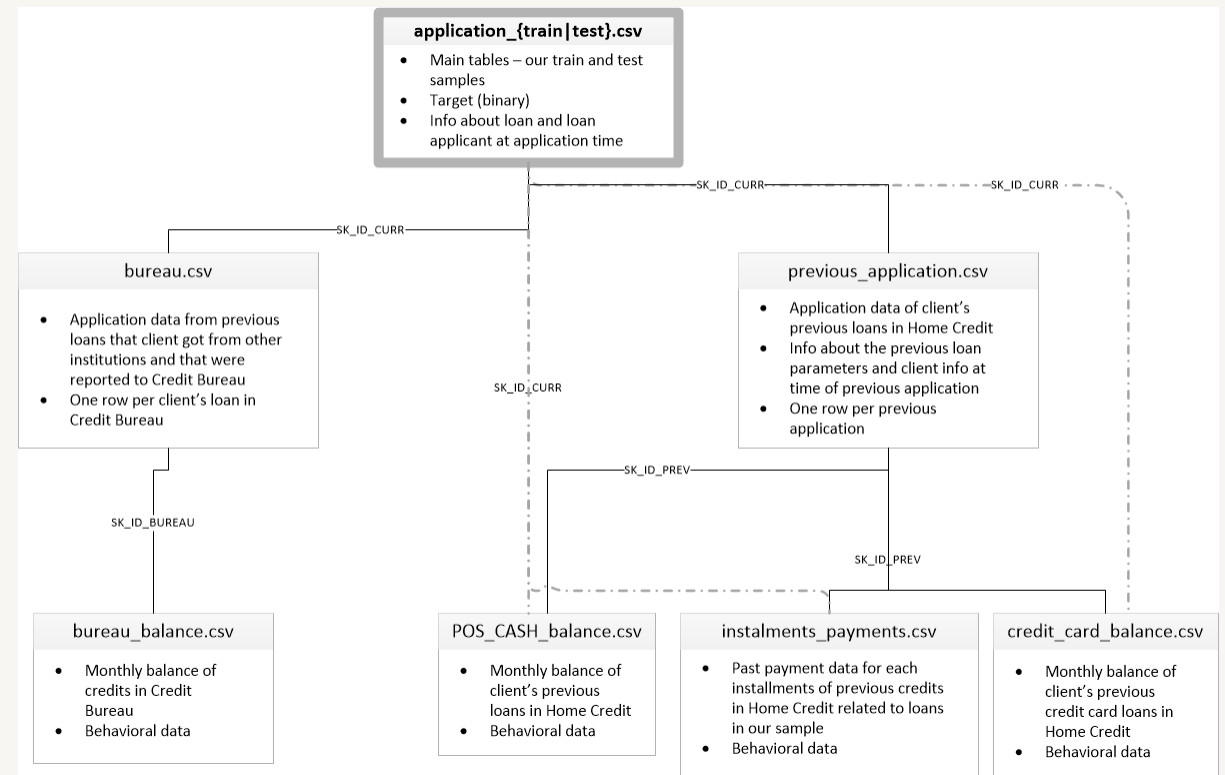


NOS DEUX OBJECTIFS

CRÉER UN
MODELE DE
SCORING

DEPLOYER LE
MODELE

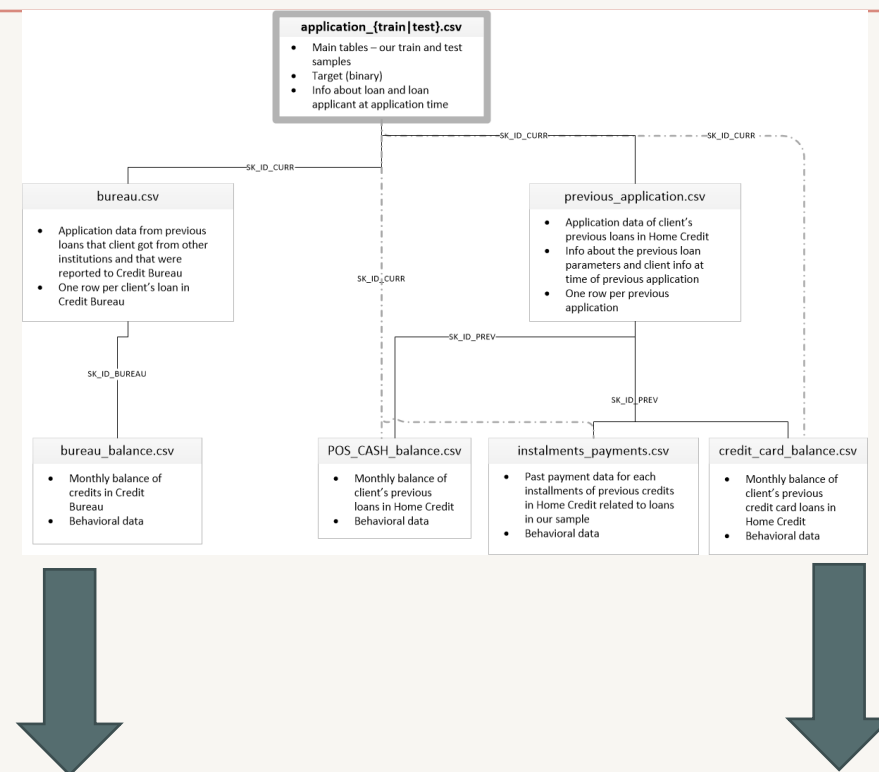
Data



Kernel

Les avantages :

- Résumé une BDD relationnelle en une table
- Générer des kpis efficaces pour la prédiction



	index	SK_ID_CURR	TARGET	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	
	0	0	100002	1.0	0	0	0	0	202500.0	406597.5	247
	1	1	100003	0.0	1	0	1	0	270000.0	1293502.5	356
	2	2	100004	0.0	0	1	0	0	67500.0	135000.0	67
	3	3	100006	0.0	1	0	0	0	135000.0	312682.5	296
	4	4	100007	0.0	0	0	0	0	121500.0	513000.0	216

356250	48739	456221	NaN	1	0	0	0	0	121500.0	412560.0	174
356251	48740	456222	NaN	1	0	1	2	157500.0	622413.0	319	
356252	48741	456223	NaN	1	1	0	1	202500.0	315000.0	332	
356253	48742	456224	NaN	0	0	1	0	225000.0	450000.0	251	
356254	48743	456250	NaN	1	1	1	0	135000.0	312768.0	247	
356251 rows x 798 columns											

Nettoyage

Suppression des colonnes trop incomplètes

```
0]: #df_train
    print("Dimension initial du dataset :",df_train.shape)
    #df_train.head()

Dimension initial du dataset : (307507, 797)

1]: # Nettoyage en fonction des valeurs manquantes
    seuil=30
    #print((((df_train.isna().sum().sort_values()/df_train.shape[0])*100)>seuil).values.sum())
    # Colonnes avec plus de 30% de valeurs manquantes
    series_na=((df_train.isna().sum().sort_values()/df_train.shape[0])*100)>30)
    liste_colonnes_na=list(series_na[series_na==False].index)
    # Liste des colonnes avec moins de 30% de valeurs manquantes
    #print(len(liste_colonnes_na))
    df_train=df_train[liste_colonnes_na]
    df_test=df_test[liste_colonnes_na]
    print("Dimension du dataset après suppression des colonnes incomplètes :",df_train.shape)
    #df_train.head()

Dimension du dataset après suppression des colonnes incomplètes : (307507, 544)
```

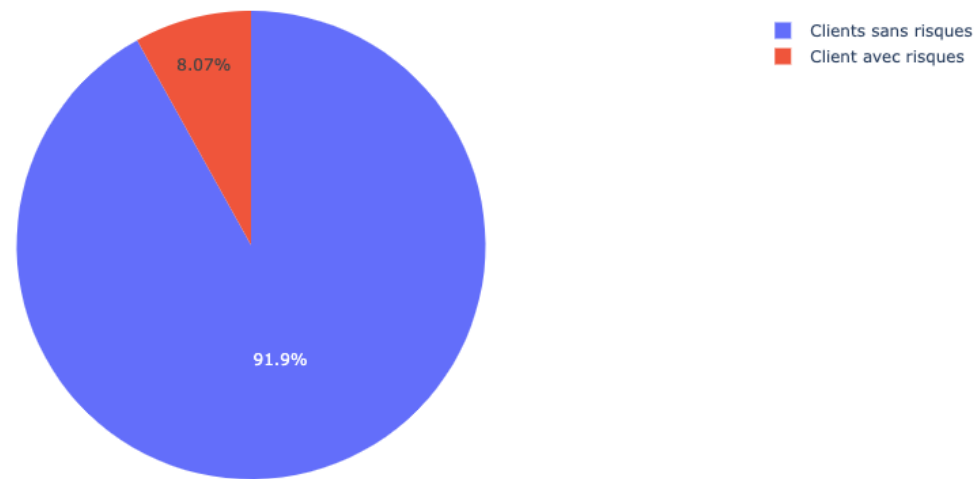
Imputation par la valeur la plus fréquentes

```
: # Imputation par la valeur la plus fréquente +infini
df_train.replace([np.inf, -np.inf], np.nan, inplace=True)

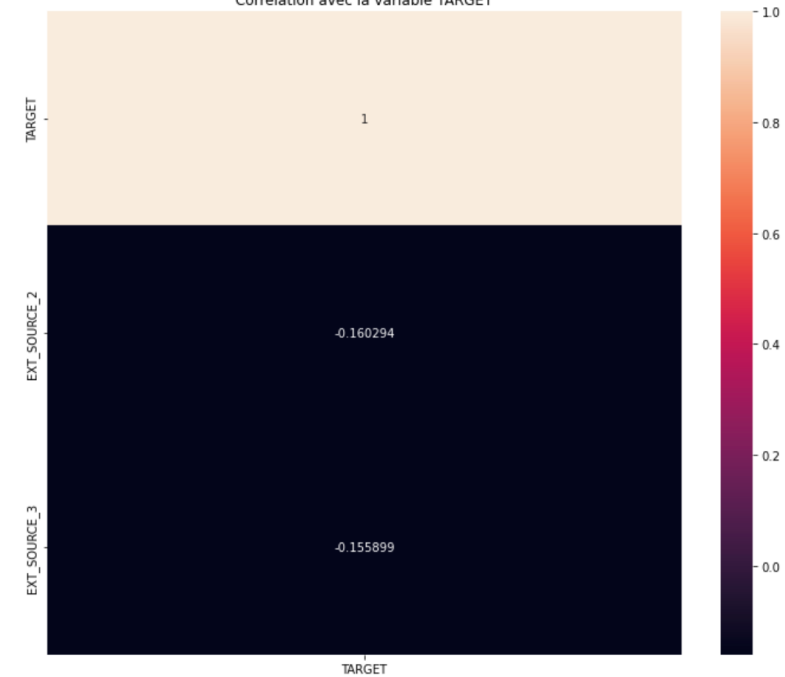
for i in range(df_train.shape[1]):
    nom_colonne=df_train.columns[i]
    if(nom_colonne!="TARGET") :
        df_train[df_train.columns[i]].fillna(df_train[df_train.columns[i]].median(),inplace=True)
        df_test[df_train.columns[i]].fillna(df_test[df_test.columns[i]].median(),inplace=True)
```

Analyse exploratoire

Répartition du risque de défaut des clients



Corrélation avec la variable TARGET



Modélisation



PATTERN DES PIPELINES TUNES

1/ CENTREE
REDUIRE



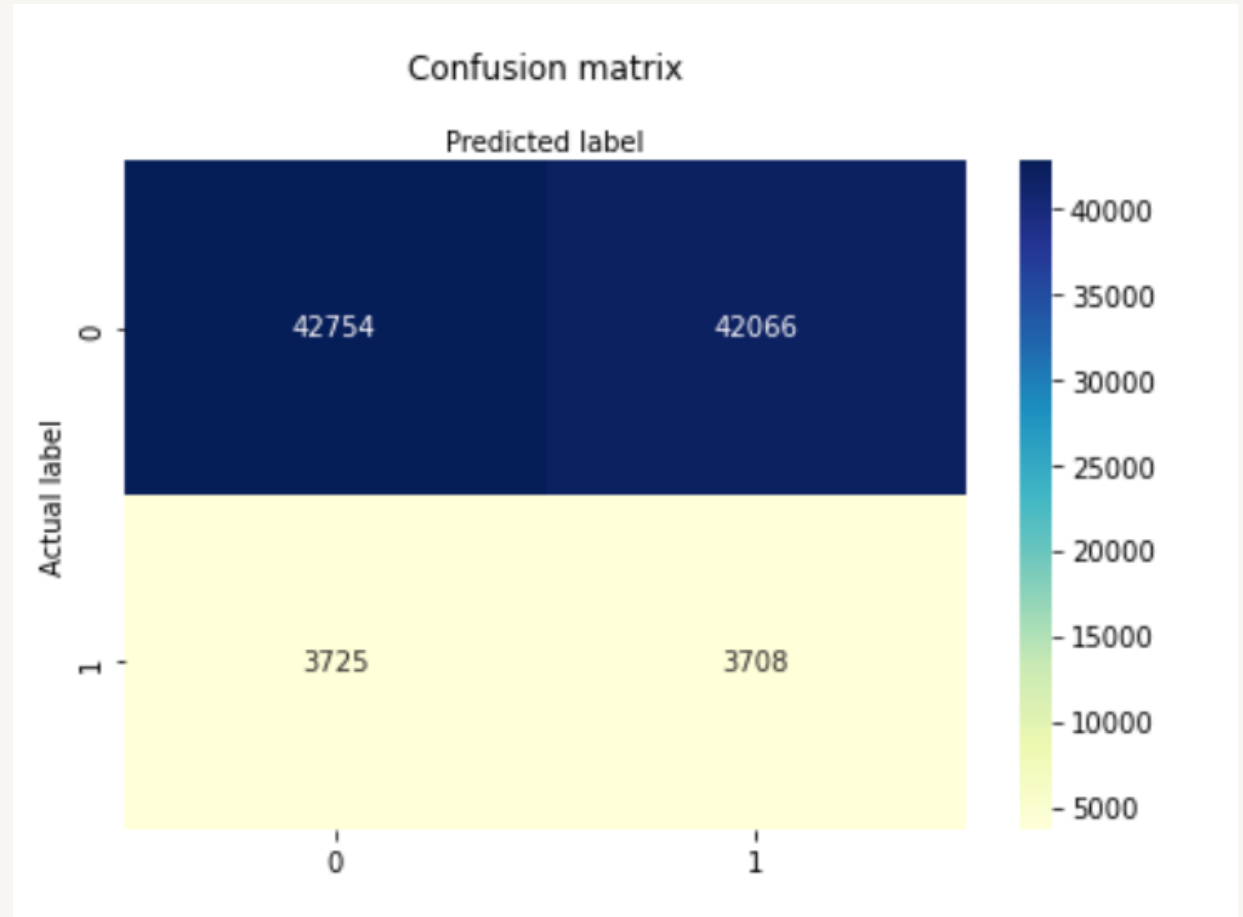
```
graph TD; A[1/ CENTREE REDUIRE] --> B[2/ SMOTE]; B --> C[3/ MODELE];
```

2/ SMOTE

3/ MODELE

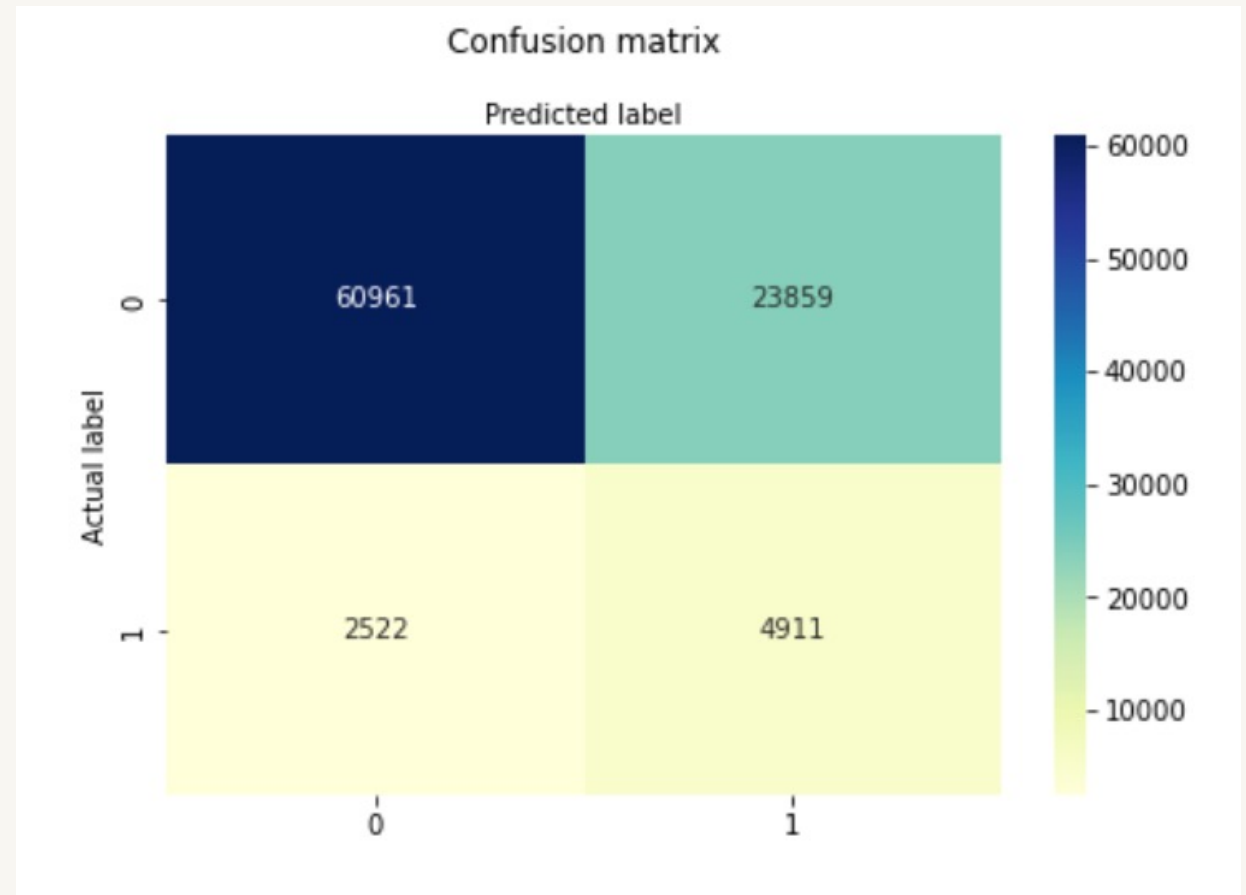
Classifieur Dummy

- AUC = 0.5
- Recall = 0.5
- Accuracy = 0.5
- Pas de tuning



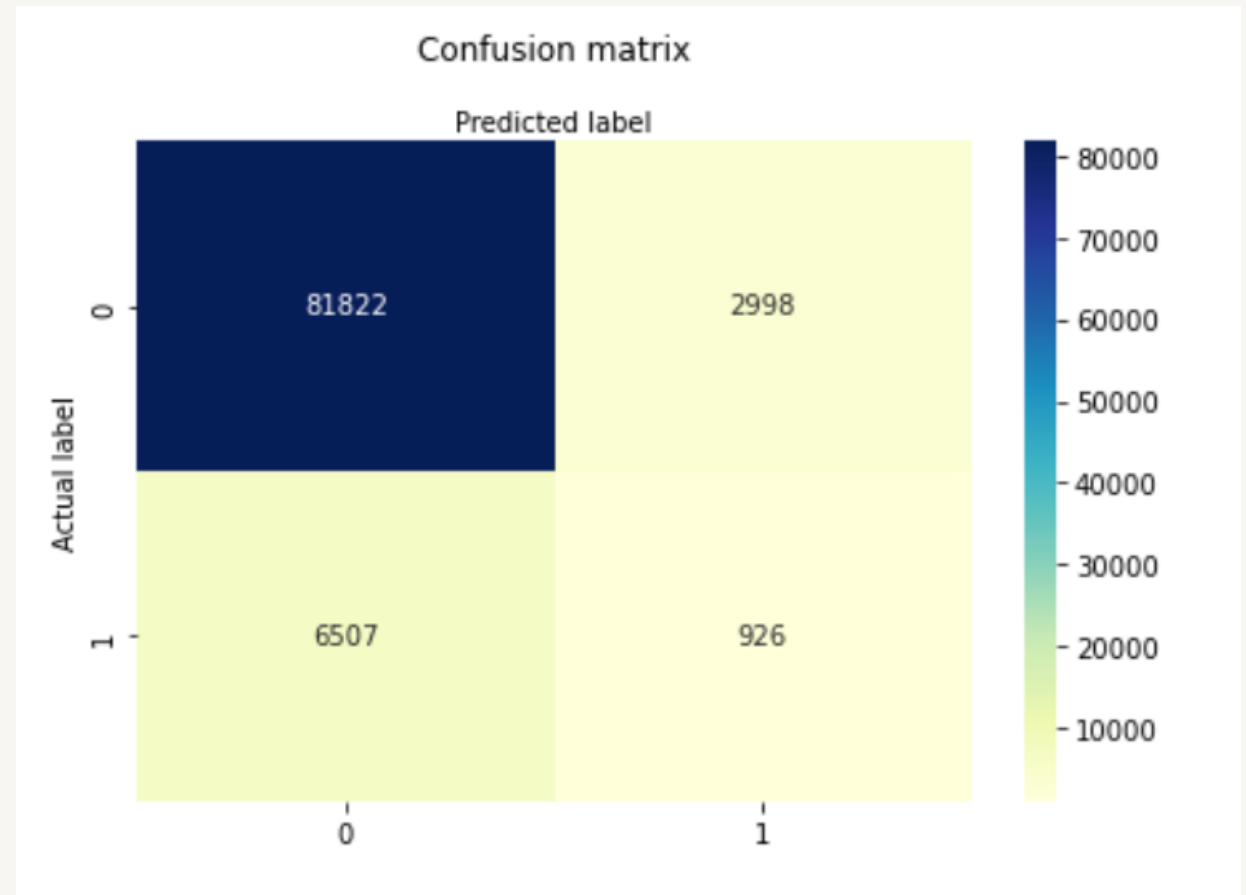
Classifieur logistique

- AUC = 0.75
- Recall = 0.66
- Accuracy = 0.71
- Hyperparamètres : {'C': 1, 'max_iter': 300}



CLASSIFIEUR RANDOMFOREST

- AUC = 0.71
- Recall = 0.66
- Accuracy = 0.89
- Hyperparamètres :{'max_depth': 15, 'n_estimators': 100}

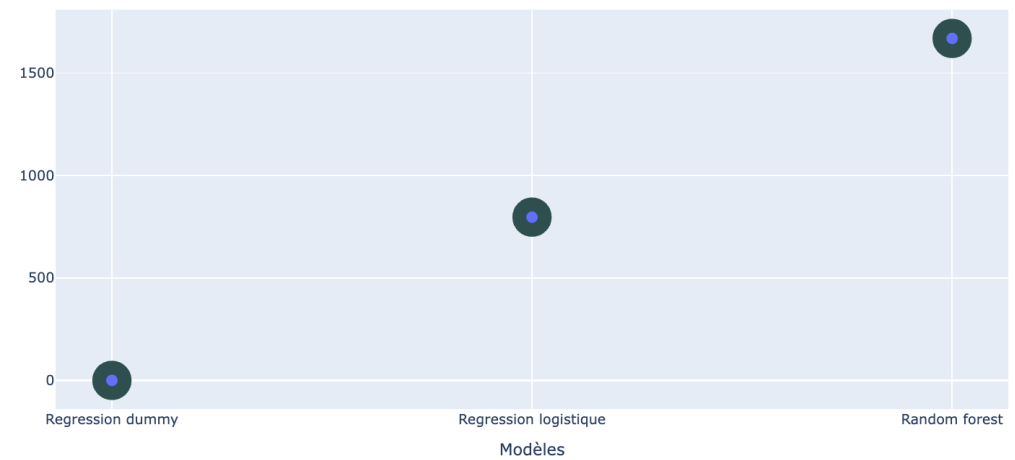


Comparaison des modèles

Comparaison de la performance des modèles

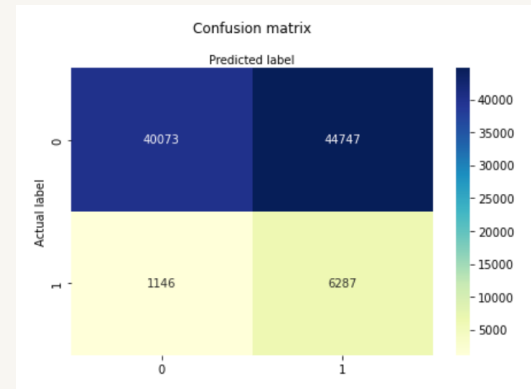


Comparaison du temps des modèles



Optimisation du recall

- Tuning base sur un score fbeta
- Modification du seuil de decision proba



```
# Tuning avec fbeta logreg
ftwo_scorer = make_scorer(fbeta_score, beta=10)

start = time.time()
model = Pipeline([
    ('scaler', StandardScaler()),
    ('sampling', SMOTE()),
    ('classification', LogisticRegression())
])

params = { 'classification__C':[1,5,10,16], 'classification__max_iter':[100,300,400]}

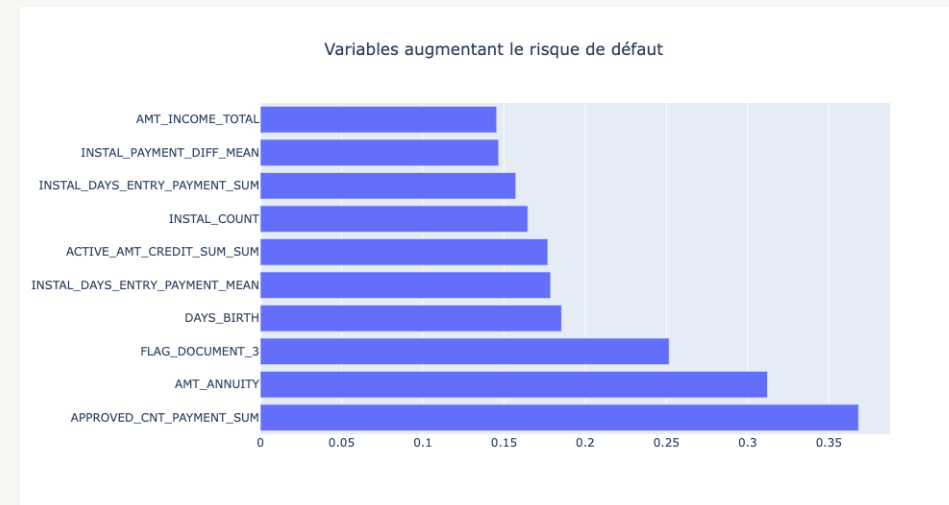
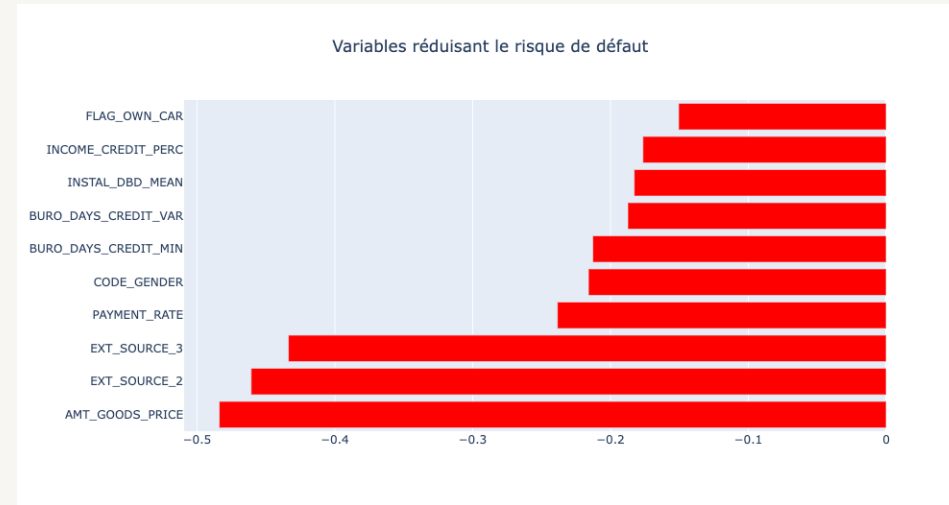
clf = GridSearchCV(model,params, cv=3, scoring =ftwo_scorer)

clf.fit(X_train, y_train)
end = time.time()
temps_logreg_ajuste = end - start
```

Comparaison de la performance des modèles

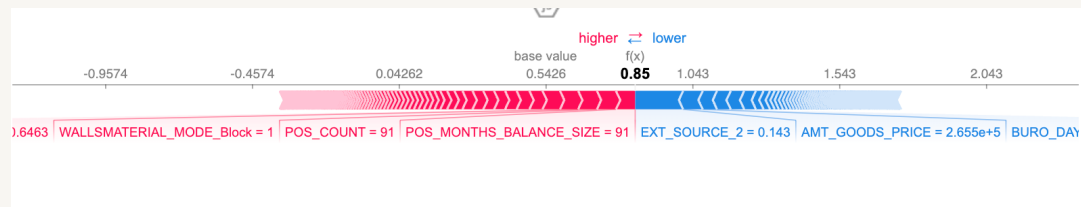


Coefficient du modèle

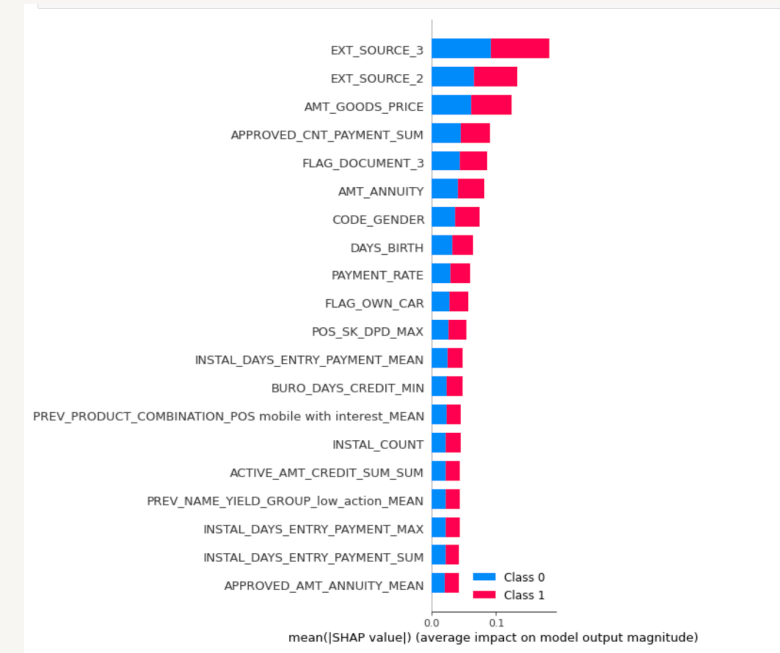


SHAP

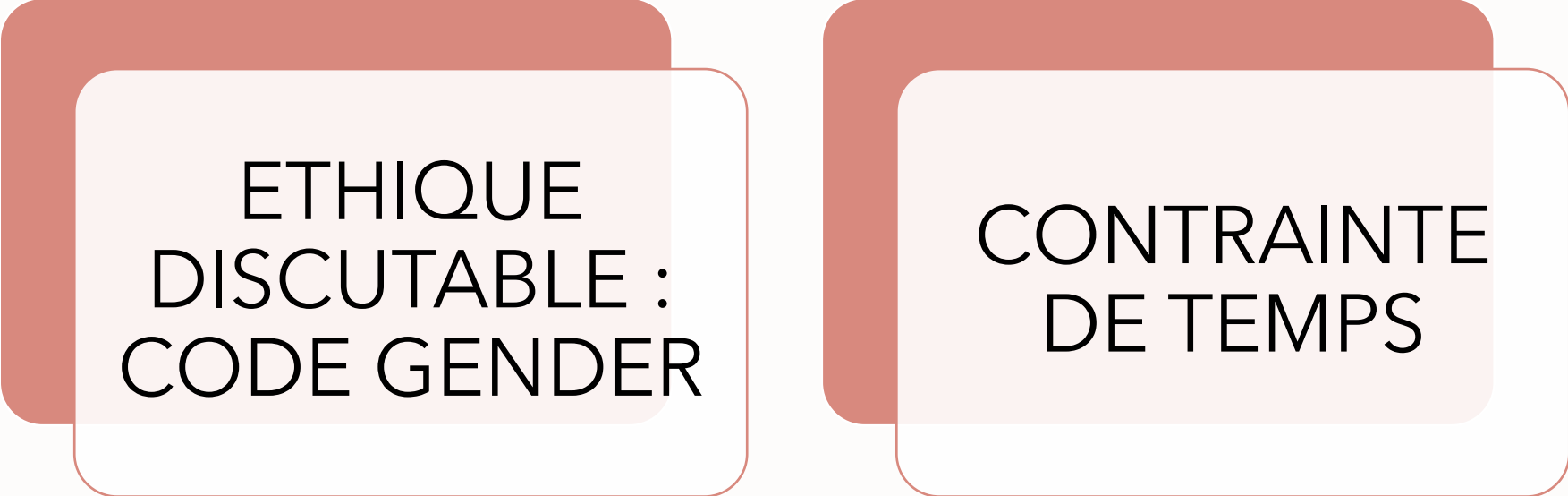
FEATURE LOCALE



FEATURE GLOBALE



Limites du modèle



ETHIQUE
DISCUTABLE :
CODE GENDER

CONTRAINTE
DE TEMPS

Application



API

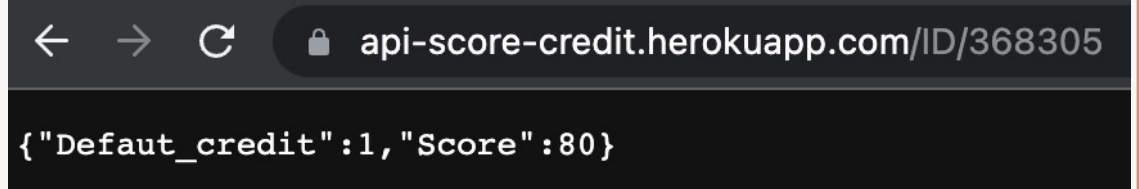
```
#Application
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Bienvenue sur le modèle de scoring'

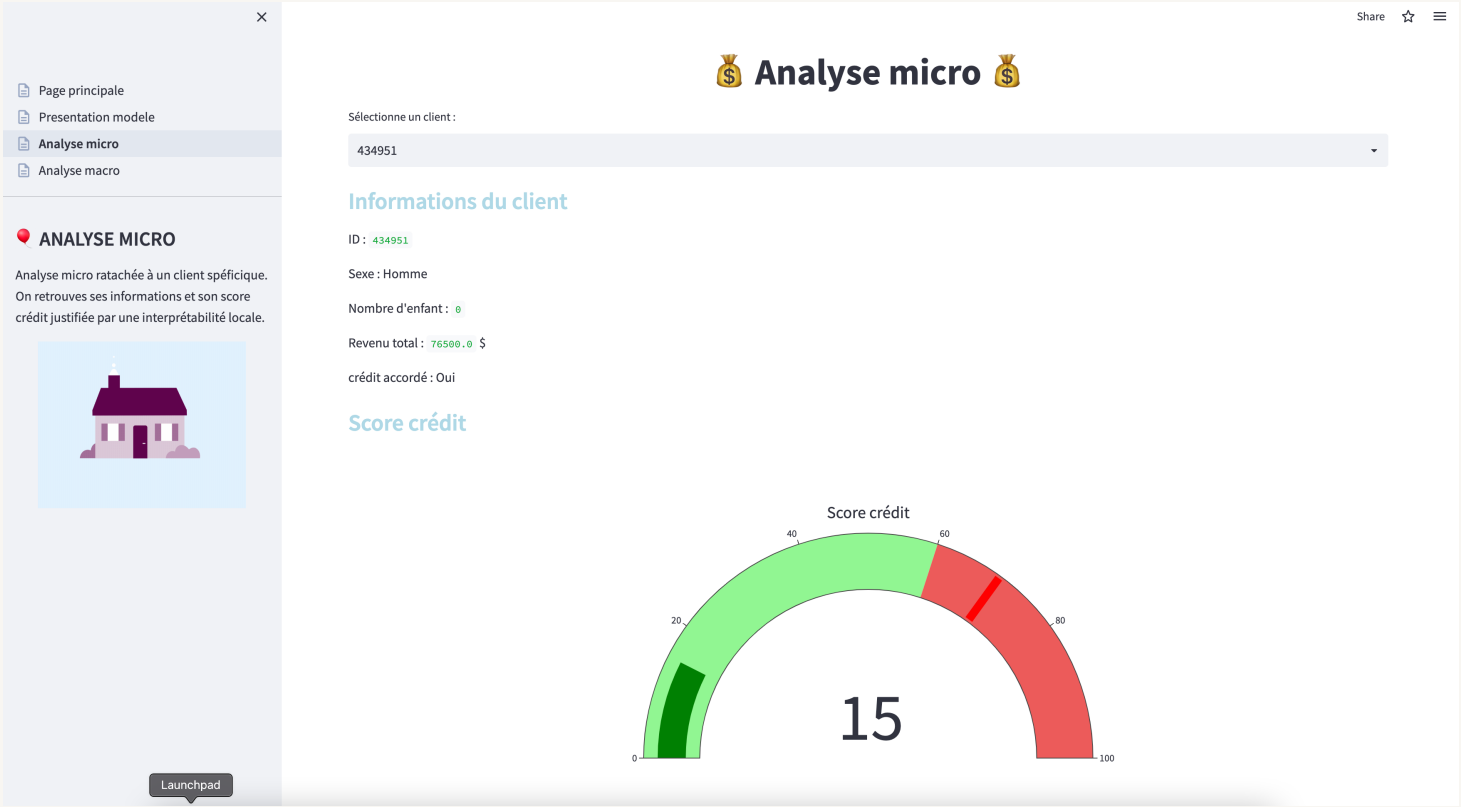
@app.route('/ID')
def ID():
    return 'ID du client'

@app.route('/ID/<id>', methods=['GET'])
def Prediction(id):
    try :
        ID=int(id) #100194
        index=df[df["SK_ID_CURR"]==ID].index.values[0]
        score=round(model.predict_proba(df.iloc[index:index+1,:])[0][1]*100) #368305
        default_credit=0
        if (score>70) : default_credit=1
        return jsonify({'Score' : score, "Default_credit" : default_credit})
    except :
        return jsonify({"erreur" : S3_connexion})

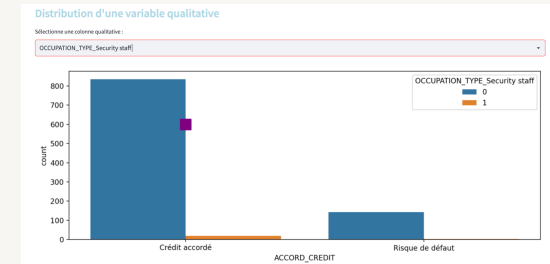
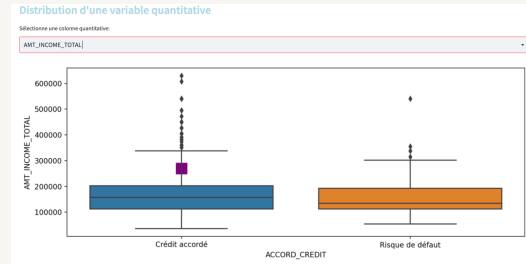
if __name__ == '__main__':
    app.run()
```



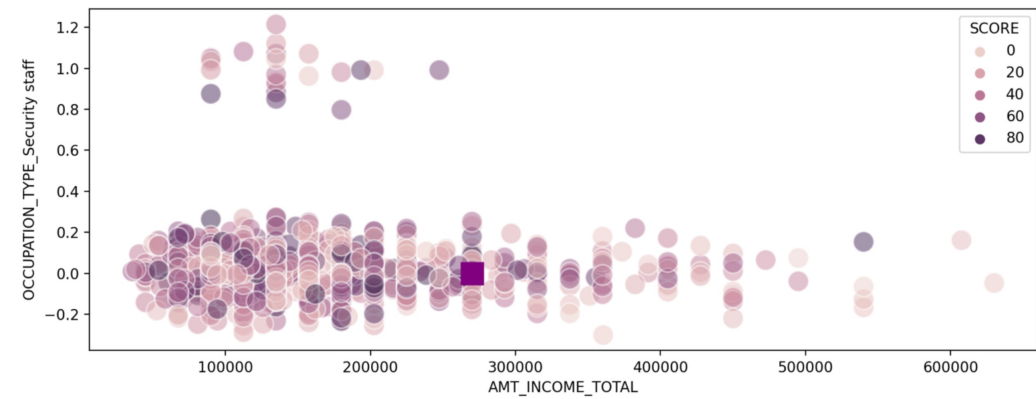
Dashboard



Dashboard



Analyse bivarée



FIN