Rapport Projet Applications of Big Data

Mohamed Ali Aboualam, Henri DuBouillon et Ambroise de Wattripont

Nous avions réalisé un projet qui avait pour but de montrer nos compétences sur les différentes notions vues durant les cours d'applications of Big Data. Pour ce faire, on nous a fourni un dataset contenant notamment : application_train.csv et application_test.csv que nous utilisâmes.

Nous avions pour tâche de créer un modèle de machine learning en essayant au mieux de respecter les conventions de programmation, en particulier :

- utiliser un GIT commun
- structurer le projet en plusieurs scripts
- utiliser un template
- utiliser conda comme environnement
- utiliser de la documentation sur un des algorithmes de base. En ce qui nous concerne, nous avons choisi Ramdom Forest.

Ensuite nous devions intégrer la librairies MLFlow a notre projet et pour finir nous devions aussi réaliser une visualisation de nos données en utilisant Shap.

Comment avons-nous fait ?

Étant trois, nous nous sommes réparti la charge de travail de manière à ce que chacun fasse une partie.

Concernant l'organisation du git repository :

- le fichier data contient nos différentes sources de données (raw, processed…)
- le fichier mlruns contient les différents runs de nos modèles enregistrés avec mlflow
- le fichier models contient nos modèles et nos prédictions
- le fichier notebook contient notre notebook avec l'EDA, le preprocessing, le feature engineering, la modélisation et la dataviz.
- le fichier report contient ce rapport ainsi que les graphiques utilisées durant nos analyses
- le fichier src contient le code python pour le preprocessing, le feature engineering, la modélisation et la dataviz

Détail des scripts python du src ainsi que de leur input et output :

- features/build_features.py :
    - utilise les données dans data/raw
    - exécute le feature engineering
    - sauvegarde les nouvelles données dans data/processed
- models/train_model.py :
    - import les données d'entrainement de data/processed
    - entraine les modèles
    - exporte les modèles dans model/
- models/predict_model.py :
    - import les données de data/processed
    - etablis les prédictions sur le test set et validation set (test_application.csv)
    - export les prédictions dans models/

- visualization/visualize.py :
    - import les modèles de models/
    - créer plusieurs visualisations

Détaille du contenu du notebook :

En ce qui concerne la production du modèle, nous avons en premier lieu importé toutes les librairies.

```
import os

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc
from sklearn import metrics

import xgboost
from xgboost.sklearn import XGBClassifier
from xgboost import plot_importance

import mlflow
import shap
```

On récupère ensuite le dataset, et on utilise alors un certain nombre de commandes pour pouvoir l'étudier et comprendre un peu mieux ce que l'on va étudier.
On utilise alors pandas pour pouvoir lire et enregistrer notre dataset qui est sous forme de fichier csv.

Exploratory data analysis

```
# Navigate to the parent directory to have access to the data directory
os.chdir("..")
```

```
# Read the application_train file
df = pd.read_csv('data/raw/application_train.csv.zip')
df.shape
```

```
(307511, 122)
```

```
# A quick-view of the dataframe
df.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
0	100002	1	Cash loans	M	N	Y	0
1	100003	0	Cash loans	F	N	N	0
2	100004	0	Revolving loans	M	Y	Y	0
3	100006	0	Cash loans	F	N	Y	0
4	100007	0	Cash loans	M	N	Y	0

5 rows × 122 columns

df.head permet de renvoyer les 5 premières itérations. Ce qui nous permet d'avoir un début de visualisation. Ensuite, on affiche les différentes colonnes et leurs types. Malheureusement, en considérant qu'il y a 122 colonnes, on ne pourra pas toutes les afficher.

```
# The name of the columns
df.columns.tolist()
# The type of the columns
df.dtypes.tolist()
```

```
Output exceeds the size limit. Open the full output data in a text editor
['SK_ID_CURR',
 'TARGET',
 'NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT',
 'AMT_ANNUITY',
 'AMT_GOODS_PRICE',
 'NAME_TYPE_SUITE',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH'
```

```
df.describe()
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000

8 rows × 106 columns

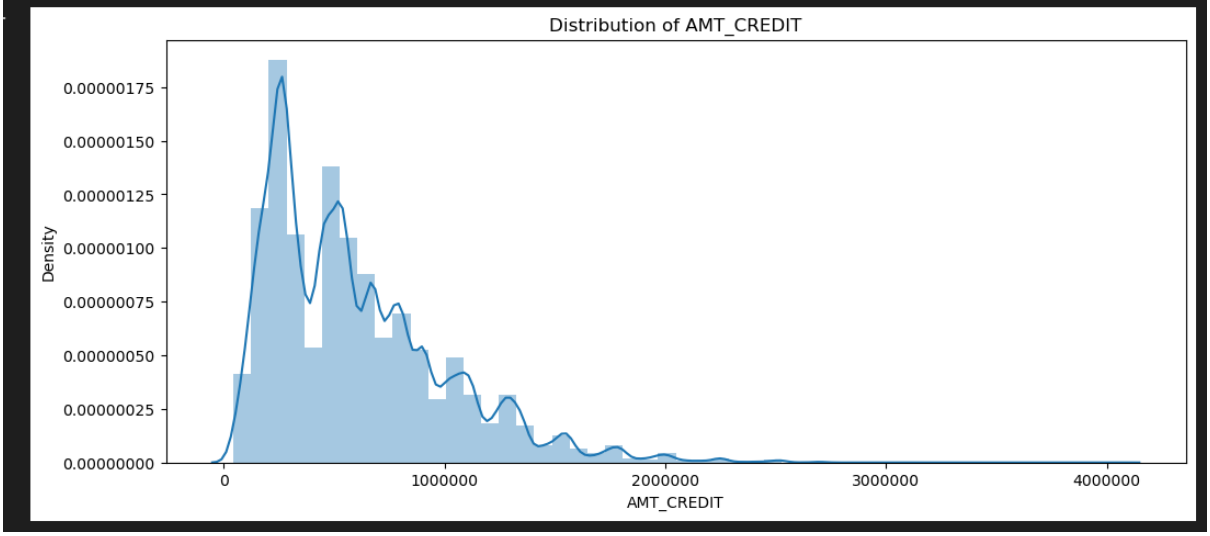
df.describe() permet de nous montrer des statistiques descriptives du dataset, nous permettant de l'étudier pour pouvoir réaliser le bon modèle. Cependant pour se faire nous devons d'abord nous assurer que notre dataset est complet.
C'est pourquoi nous avons cherché les missing values comme suit.

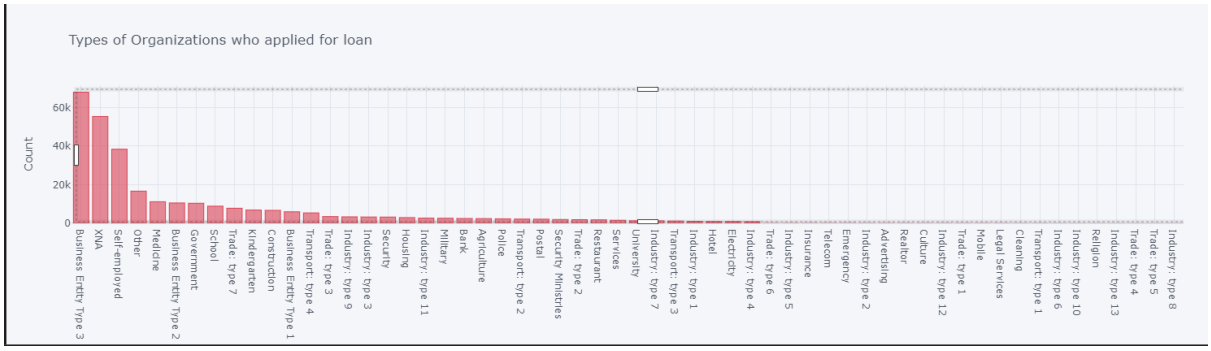
```
# checking missing data
total = df.isnull().sum().sort_values(ascending = False)
percent = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascending = False)
missing_application_train_data  = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_application_train_data.head(20)
```

	Total	Percent
COMMONAREA_MEDI	214865	69.872297
COMMONAREA_AVG	214865	69.872297
COMMONAREA_MODE	214865	69.872297
NONLIVINGAPARTMENTS_MODE	213514	69.432963
NONLIVINGAPARTMENTS_AVG	213514	69.432963
NONLIVINGAPARTMENTS_MEDI	213514	69.432963
FONDKAPREMONT_MODE	210295	68.386172
LIVINGAPARTMENTS_MODE	210199	68.354953
LIVINGAPARTMENTS_AVG	210199	68.354953
LIVINGAPARTMENTS_MEDI	210199	68.354953
FLOORSMIN_AVG	208642	67.848630
FLOORSMIN_MODE	208642	67.848630
FLOORSMIN_MEDI	208642	67.848630
YEARS_BUILD_MEDI	204488	66.497784
YEARS_BUILD_MODE	204488	66.497784
YEARS_BUILD_AVG	204488	66.497784
OWN_CAR_AGE	202929	65.990810
LANDAREA_MEDI	182590	59.376738
LANDAREA_MODE	182590	59.376738
LANDAREA_AVG	182590	59.376738

On peut noter qu'il manque des données, cependant ce n'est pas préjudiciable pour la suite. Nous avons ensuite décidé de déterminer la distribution de la variable AMT_CREDIT.
De plus, nous avons aussi réalisé d'autres vérifications :comme la proportion des target, les types des loan ou bien les 20 variables corrélées positivement.

```
# Distribution of AMT_CREDIT
plt.figure(figsize=(12,5))
plt.title("Distribution of AMT_CREDIT")
ax = sns.distplot(df["AMT_CREDIT"])
```





Tous ces graphiques sont certes intéressants, mais ils ne sont pas utiles dans l'avancement de notre modèle.
De ce fait, nous allons vous présenter maintenant les transformations des données brutes vers des données utilisables.
Fichier SRC

Passons maintenant au feature engineering, où nous allons utiliser les données et les transformées pour préparer nos modèles.
Nous avons décidé de transformer les missing value en médiane sinon nous perdions trop de valeurs ce qui aurait été dommageable.

```
# Median imputation of missing values
imputer = SimpleImputer(missing_values=np.nan, strategy='median')
imputer.fit(df)
df = pd.DataFrame(imputer.transform(df), columns=df.columns)

print('Training data shape: ', df.shape)
```

Nous pouvons alors split le dataset pour le modèle :

```
# Splitting the data in train and test set
X = df.drop(['TARGET'], axis='columns')
y = df['TARGET']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Ce qui permet d'initialiser le modèle.

```python
#Random Forest
rf = RandomForestClassifier(random_state=0, n_estimators=450, n_jobs=-1)
rf.fit(X_train, y_train)
rf_y_pred = (rf.predict_proba(X_test) <= 0.8)[:,0].astype(int)

#Confusion matrix
rf_conf = confusion_matrix(y_test, rf_y_pred)
print("confusion_matrix XGB_sk_learn :\n", rf_conf, '\n')

#Balanced accuracy
balanced_accuracy_rf = balanced_accuracy_score(y_test, rf_y_pred)
print("Balanced accuracy : ", balanced_accuracy_rf)

#AUC
rf_a, rf_b, _ = roc_curve(y_test, rf_y_pred)
print("AUC_rf : ", metrics.auc(rf_a, rf_b), '\n')

#Check the proportion of the target
print("Check proportion on target (should be around 8%): ", rf_y_pred.sum()/len(rf_y_pred))
```

```
confusion_matrix XGB_sk_learn :
 [[43114  2984]
 [ 3211  1119]]

Balanced accuracy :  0.5968489512803179
AUC_rf :  0.5968489512803179

Check proportion on target (should be around 8%):  0.08136352819861982
```

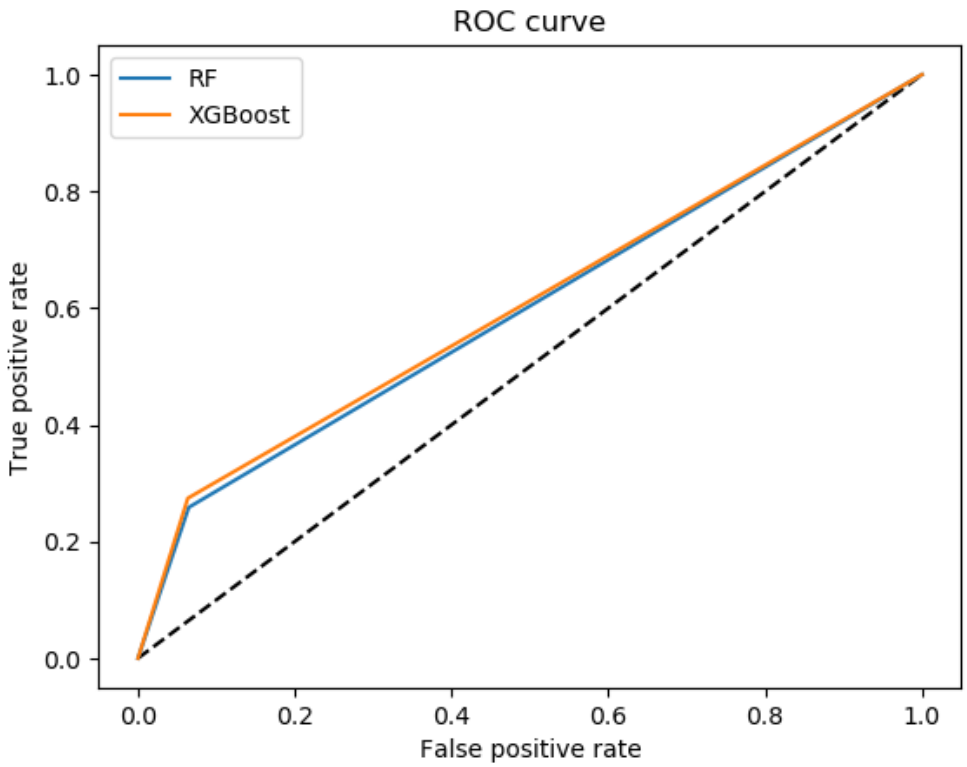
On choisit de mettre le niveau auquel on attribue les prédictions à 1 ou 0 à 0.8 de sorte de respecter la proportion de 8.07% pour la classe 1. De plus, comme nous somme sur un cas ou les données de la target sont fortement déséquilibrées, nous utilisons l'AUC (l'air sous la courbe ROC) et la balanced accuracy pour évaluer notre modèle.

```python
#XGBoost
mlflow.xgboost.autolog()

xgb = xgboost.XGBClassifier(learning_rate =0.15,
 n_estimators=300,
 max_depth=8,
 n_jobs=-1,
 random_state=0)
xgb.fit(X_train, y_train)
xgb_y_pred = (xgb.predict_proba(X_test) <= 0.786)[:,0].astype(int)

#Confusion matrix
xgb_conf = confusion_matrix(y_test, xgb_y_pred)
print("confusion_matrix XGB_sk_learn :\n", xgb_conf, '\n')

#Balanced accuracy
balanced_accuracy_xgb = balanced_accuracy_score(y_test, xgb_y_pred)
print("Balanced accuracy : ", balanced_accuracy_xgb)

#AUC
xgb_a, xgb_b, _ = roc_curve(y_test, xgb_y_pred)
print("AUC_xgb : ", metrics.auc(xgb_a, xgb_b), '\n')

print("Check proportion on target (should be around 8%): ", xgb_y_pred.sum()/len(xgb_y_pred))
```

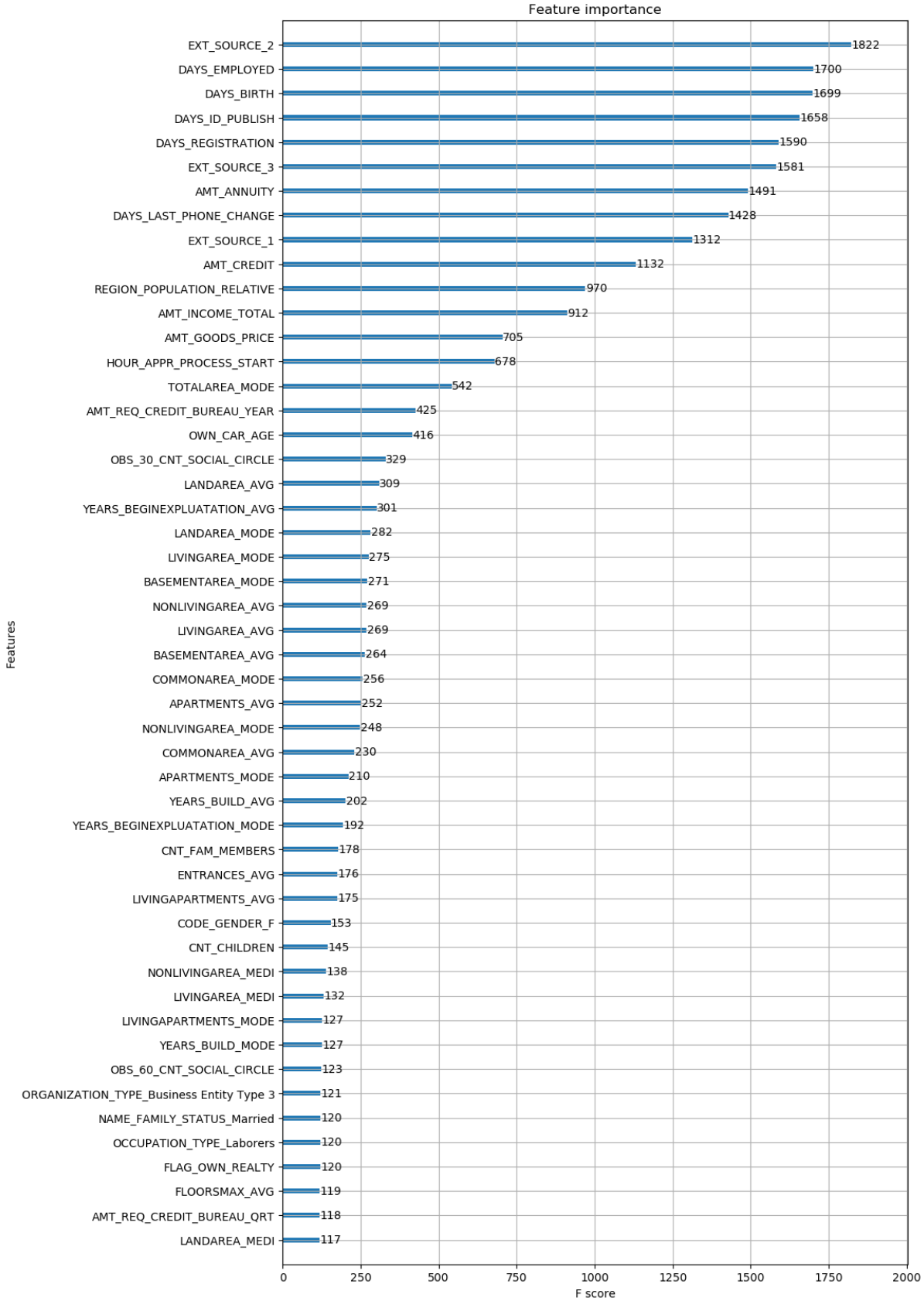
```
2023/01/31 00:08:48 WARNING mlflow.utils.autologging_utils: You are using an unsupported version of xgboost.
supported version, or try upgrading MLflow.
2023/01/31 00:08:48 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID 'f76a81e5c744
artifacts, and lineage information for the current xgboost workflow

confusion_matrix XGB_sk_learn :
 [[43168  2930]
 [ 3143  1187]]

Balanced accuracy :  0.605286853983235
AUC_xgb :  0.605286853983235

Check proportion on target (should be around 8%):  0.08164115174109622
```

De même pour XGBoost, on choisit de mettre le niveau auquel on attribue les prédictions à 1 ou 0 à 0.786 pour respecter la proportion de 8.07% pour la classe 1 puis, nous affichons l'AUC et la balanced accuracy du modèle.

Nous affichons ensuite la courbe ROC pour les deux modèles.


ROC curve
RF
XGBoost
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0

Ensuite, on regarde la feature importance de XGBoost :

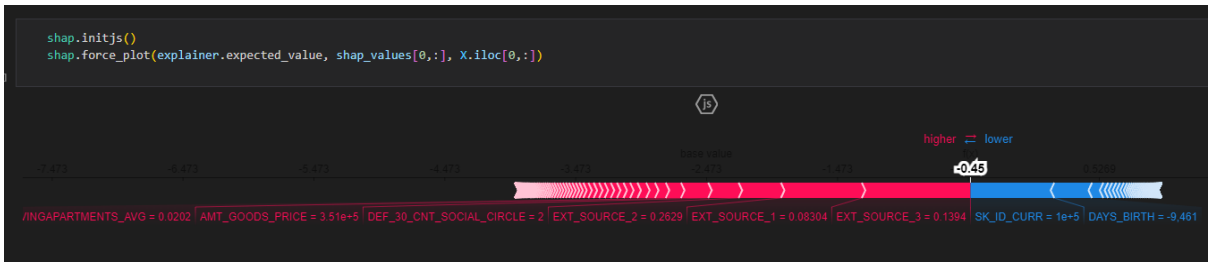


Les trois variables qui ont le F score le plus utilisé sont EXT_SOURCE_2, DAYS_EMPLOYED et DAYS_BIRTH.

Nous utilisons ensuite Shap pour réaliser une data visualisation plus explicite:

```
explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(X)
```

Nous nous retrouvons alors avec plusieurs graphes qui permettent de mettre en corrélation les valeurs attendues avec celle que nous retrouvons.



Puis nous réalisons un summary pour mettre en lumière les différences de corrélation.