Application of big data
Project Report

LAHLOU Mamoun
AJAZ HAIDER Awn

Part 1 : Building ML Models with coding best practices

Coding best practices :

For this project, we used git and commit and push often with clear messages. Our code is very simple to understand and everyone can run it

The Dataset :

The dataset used for this project is the Home Credit Default Risk, from Kaggle.
We download only 2 datasets, application_train/application_test: The main training data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating 0: the loan was repaid or 1: the loan was not repaid.

Data preparation & Feature engineering :

After importing our 2 datasets, appliation_train.csv and application_test.csv, we can have a quick look at them :

appliation_train.csv :

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_
0	100002	1	Cash loans	M	N	Y	0	202500.0	
1	100003	0	Cash loans	F	N	N	0	270000.0	1
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	
4	100007	0	Cash loans	M	N	Y	0	121500.0	

application_test.csv :

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT
0	100001	Cash loans	F	N	Y	0	135000.0	568800.0
1	100005	Cash loans	M	N	Y	0	99000.0	222768.0
2	100013	Cash loans	M	Y	Y	0	202500.0	663264.0
3	100028	Cash loans	F	N	Y	2	315000.0	1575000.0
4	100038	Cash loans	M	Y	N	1	180000.0	625500.0

The differences between these two datasets are that one has the ‘Target’ column and not the other. That’s why we are going to use only the training data.

As we want to do a classification, we focus on the column ‘target’. The ‘Target’ variable is a binary variable that indicates in our case, if the loan is paid (0) or not paid yet (1).

```
0    282686
1     24825
Name: TARGET, dtype: int64
```

We can see that we have more loans paid than not paid.

It is also important to deal with the missing values Indeed, in our train dataset, we also have some missing values : we have 67 columns that contain missing values.

For having a better view of these columns, we plot a dataframe that shows the 20 first columns with the most missing values and their percentage.

	Missing Values	% of Total Values
COMMONAREA_MEDI	214865	69.9
COMMONAREA_AVG	214865	69.9
COMMONAREA_MODE	214865	69.9
NONLIVINGAPARTMENTS_MEDI	213514	69.4
NONLIVINGAPARTMENTS_MODE	213514	69.4
NONLIVINGAPARTMENTS_AVG	213514	69.4
FONDKAPREMONT_MODE	210295	68.4
LIVINGAPARTMENTS_MODE	210199	68.4
LIVINGAPARTMENTS_MEDI	210199	68.4
LIVINGAPARTMENTS_AVG	210199	68.4
FLOORSMIN_MODE	208642	67.8
FLOORSMIN_MEDI	208642	67.8
FLOORSMIN_AVG	208642	67.8
YEARS_BUILD_MODE	204488	66.5
YEARS_BUILD_MEDI	204488	66.5
YEARS_BUILD_AVG	204488	66.5
OWN_CAR_AGE	202929	66.0
LANDAREA_AVG	182590	59.4
LANDAREA_MEDI	182590	59.4
LANDAREA_MODE	182590	59.4

Having so much missing values is not good at all for the training of our models. Although some columns have about 70% of missing values, we decided to not delete them to ensure the best accuracy for our models. However, we have decided to impute these missing values with the most frequent value of each column. With this strategy, we are able to maintain all the columns and each observation of our dataset.

Models training & prediction :

For this project, we train & test 3 models :

- Xgboost,
- Random Forest
- Gradient Boosting

We split our data into training and testing sets. We use a split of the train set to check our models :

- `X_train : (246008, 242)`
- `Y_train : (246008,)`
- `X_test : (61503, 242)`
- `Y_ test : (61503,)`

We got from each models the following scores :

-XGBoost :

```
In [41]: gb_clf = GradientBoostingClassifier(n_estimators=20, learning_rate=0.01, max_features=2, max_depth=2, random_state=0)
         gb_clf.fit(X_train,y_train)
         pred_gb = gb_clf .predict(X_test)

         print("Accuracy score : ", (accuracy_score(y_test, pred_gb)))

         Accuracy score :  0.9200851990959791
```

-Random Forest :

```
# Model Accuracy
accuracy_score(y_test, y_pred_rm)
```

```
0.9200689397265174
```

```
print(classification_report(y_test,y_pred_rm))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     56588
           1       0.40      0.00      0.00      4915

   micro avg       0.92      0.92      0.92     61503
   macro avg       0.66      0.50      0.48     61503
weighted avg       0.88      0.92      0.88     61503
```

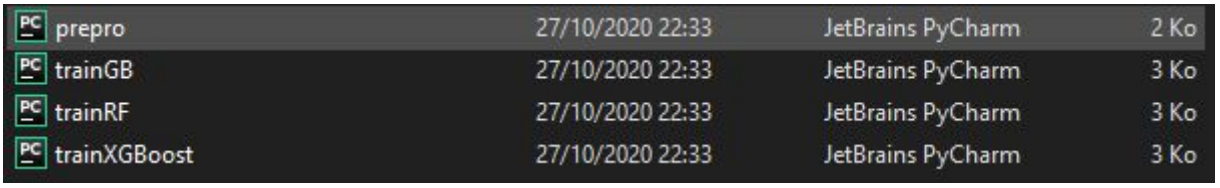
-Gradient Boosting :

```
p_xgboost = xgb_clf.predict(X_test)
print ("Score Train -->", accuracy_score(y_test, p_xgboost), " %")
```

```
Score Train --> 0.9199063460319009  %
```

Part 2 : Using MLFLow

For the mlflow part we will split our codes into different scripts, one for the preprocessing part, one for the xgboost model, one for the random forest model and one other for the gradient boosting model.



prepro	27/10/2020 22:33	JetBrains PyCharm	2 Ko
trainGB	27/10/2020 22:33	JetBrains PyCharm	3 Ko
trainRF	27/10/2020 22:33	JetBrains PyCharm	3 Ko
trainXGBoost	27/10/2020 22:33	JetBrains PyCharm	3 Ko

We have created a preprocessing function that clean and return the train dataset :

```
def preprocessing():
    df_train = pd.read_csv('application_train.csv')
    df_test = pd.read_csv('application_test.csv')
    le = LabelEncoder()
    le_count = 0
    for col in df_train:
        if df_train[col].dtype == 'object':

            if len(list(df_train[col].unique())) <= 2:

                le.fit(df_train[col])

                df_train[col] = le.transform(df_train[col])
                df_test[col] = le.transform(df_test[col])
                le_count += 1
    df_train = pd.get_dummies(df_train)
    df_test = pd.get_dummies(df_test)
    imp = SimpleImputer(strategy="most_frequent")
    train = pd.DataFrame(imp.fit_transform(df_train))
    train.columns = df_train.columns
    train.index = df_train.index
    train = pd.DataFrame(imp.fit_transform(df_train))
    train.columns = df_train.columns
    train.index = df_train.index
    print('%d columns were label encoded.' % le_count)

    print('%d columns were label encoded.' % le_count)
    return train
```

we import that function so we can use it in our different train models :

```
from prepro import preprocessing
```

After implementing our different models , we will track the best metrics for each of them. Since it's a classification problems the best ones are fscore, precision, recall and support :

```
from sklearn.metrics import accuracy_score
```

```
precision, recall, fscore, support = score(y_test, y_pred_rm)
precision0 = precision[0]
precision1 = precision[1]
recall0 = recall[0]
recall1 = recall[1]
fscore0 = fscore[0]
fscore1 = fscore[1]
support0 = support[0]
support1 = support[1]
accuracy=accuracy_score(y_test,y_pred_rm)
```

We will also track the parameters of ours models to find the best ones :

for the random forest :

```
NEST = int(sys.argv[1])
MWFL = float(sys.argv[2])
MID = float(sys.argv[3])
with mlflow.start_run():
    rf_clf = RandomForestClassifier(n_estimators=NEST, min_weight_fraction_leaf=MWFL,min_impurity_decrease=MID)
```

for the XGBoost :

```
LR = float(sys.argv[1])
NEST = int(sys.argv[2])
RS = int(sys.argv[3])

with mlflow.start_run():
    xgb_clf = XGBClassifier(learning_rate=LR,n_estimators=NEST,random_state=RS)
```

for the gradient boosting :

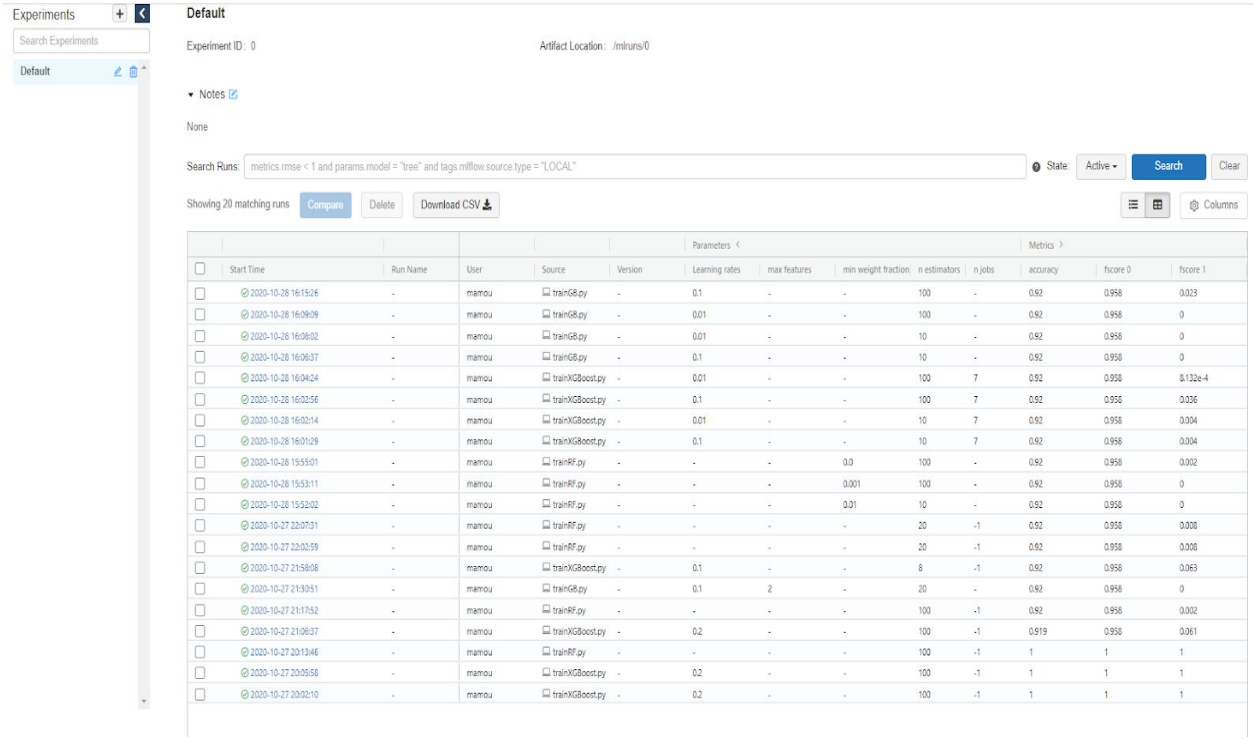
```
LR = float(sys.argv[1])
NEST = int(sys.argv[2])
MF = int(sys.argv[3])

with mlflow.start_run():
    gb_clf = GradientBoostingClassifier(n_estimators=NEST, learning_rate=LR, max_features=MF
```

We open mlflow ui with the following command :

```
(pythonProject) C:\Users\mamou\PycharmProjects\pythonProject>mlflow ui
c:\users\mamou\anaconda3\envs\pythonproject\lib\site-packages\waitress\a
his value to False, or opt-in explicitly by setting this to True.
  warnings.warn(
Serving on http://SHADOW-QA290G67:5000
WARNING:waitress.queue:Task queue depth is 1
```

We can track now the result of our models :



Now we put our model in a server :

```
>mlflow models serve -m mlruns\0\9f5e0c0227f14b789e7191b71a7cbb00\artifacts\gb_clf -p 50000
  warnings.warn(
Serving on http://SHADOW-QA290G67:5000
```

Part 3 : XAI with SHAP Method

In this part, we will interpret and explain our model by using SHAP (SHapley Additive exPlanations).

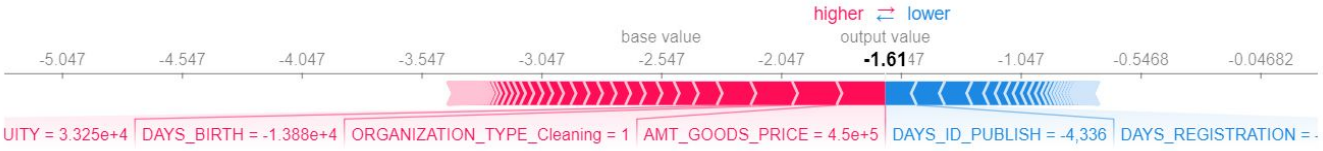
First, we start by building a TreeExplainer, which is an explainer optimised for tree based models. We will use it with our Xgboost model.

Then, we use the Shap Values method which explains how a feature can impact our model. The SHAP values for XGBoost explains the margin output of the model.

- Explanations for a specific point of our data set by using force_plot

We visualize explanations for a specific point of our data set. In the following plot, it's a specific explanation for the 1st observation of our dataset.

This force_plot shows us how each feature of our dataset contributes to shifting the prediction from the base value to the output value of the Xgboost model either by decreasing or increasing the probability of our class. In other words, how the features drive the prediction.

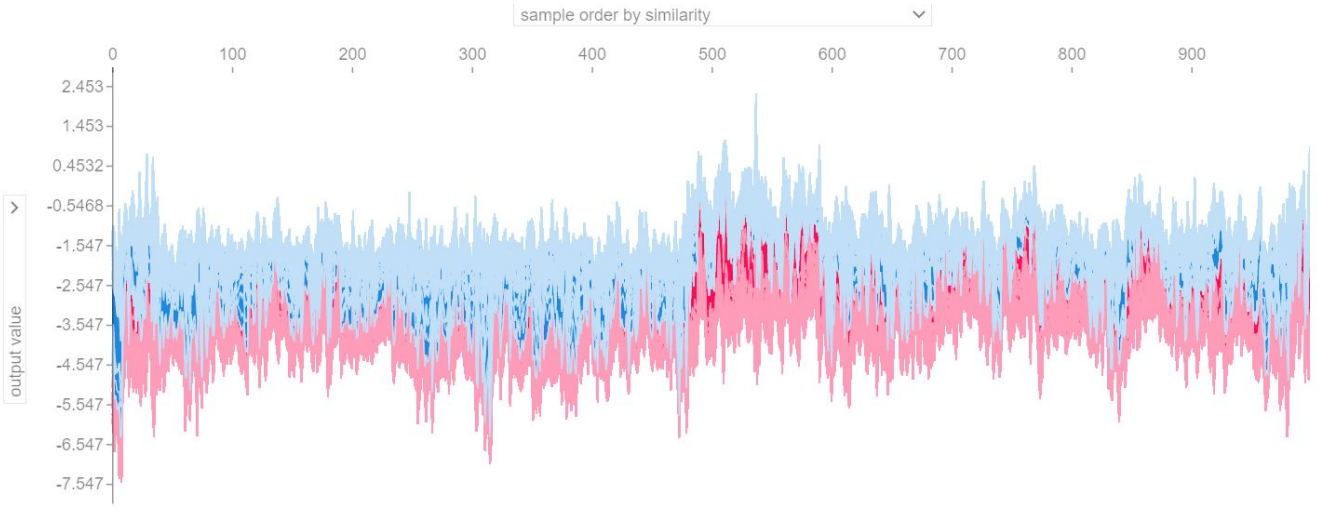


In this plot, red arrows represent the feature effects (SHAP values) that drive the prediction value higher and blue arrows are those that drive the prediction value lower. In our case, the features 'DAYS_ID_PUBLISH' and 'DAYS_REGISTRATION' pushes the prediction negatively while the features in red as 'DAYS_BIRTH' or 'ORGANIZATION_TYPE_Cleaning' pushes the prediction positively.

The "output value" represents the prediction for the instance, and equals -1,61.

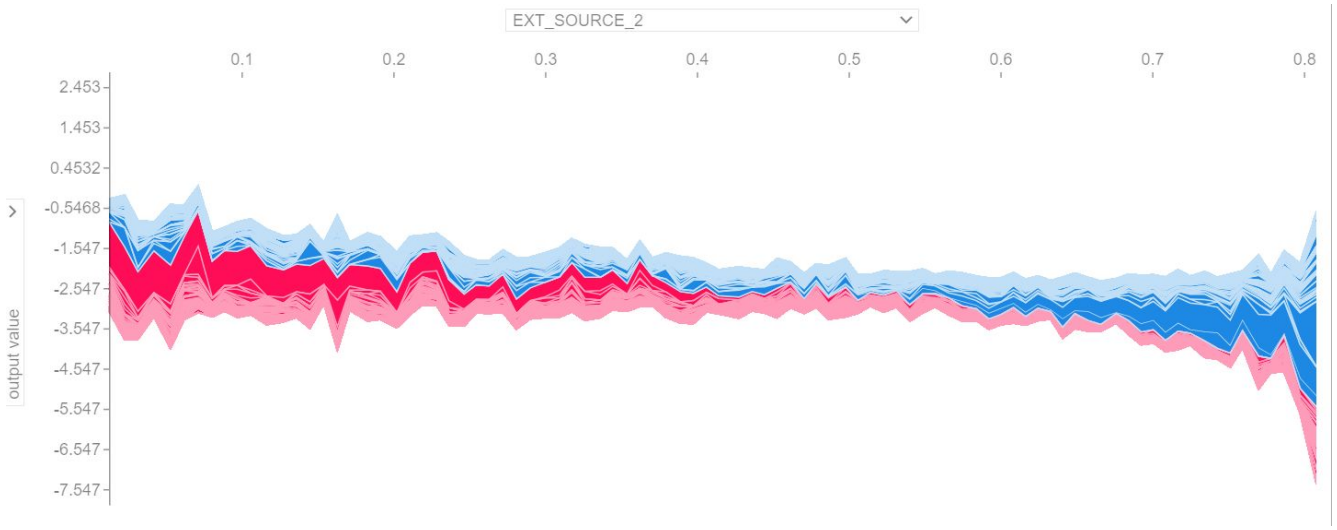
Other example: for another observation of the dataset.

- Explanations for all points of our data set at once:



We plot a force_plot to visualize all points of our data. In fact, we decided to show it for 1000 observations as we can see on the x axis. It shows us a global understanding of our model. Blue values correspond to the probability decreasing, and the red values, the probability increasing.

We can also use it to visualise the impact of a specific feature, for example 'EXT_SOURCE_2'only.

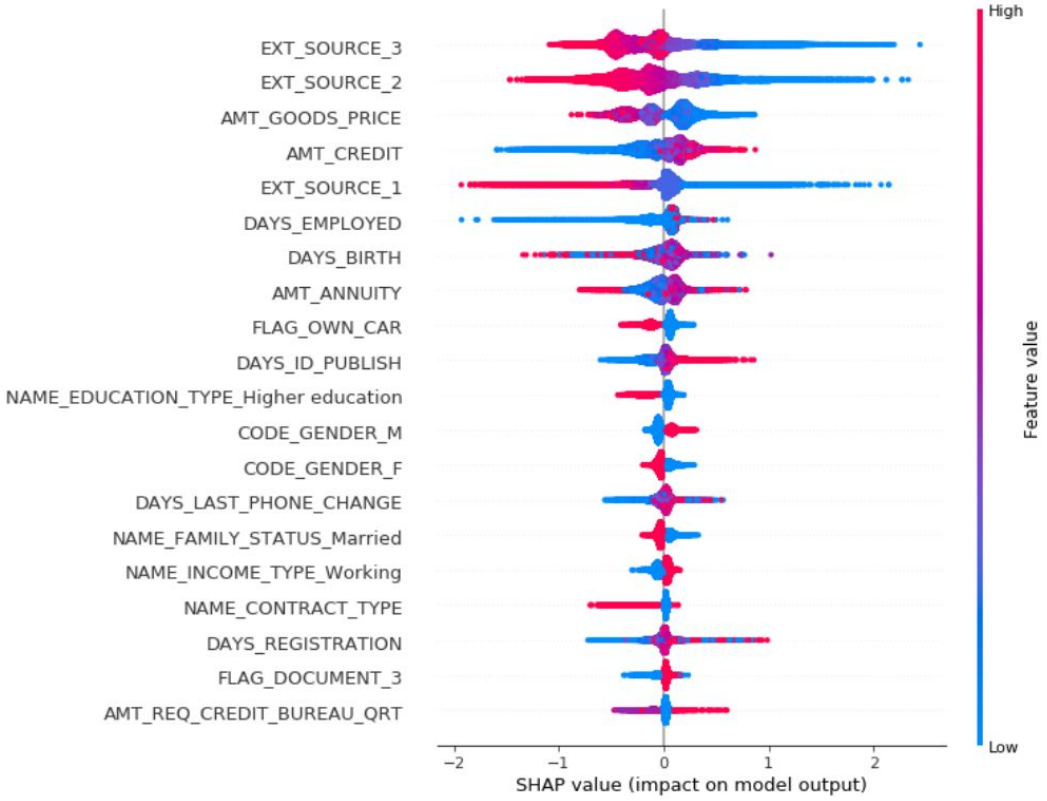


.

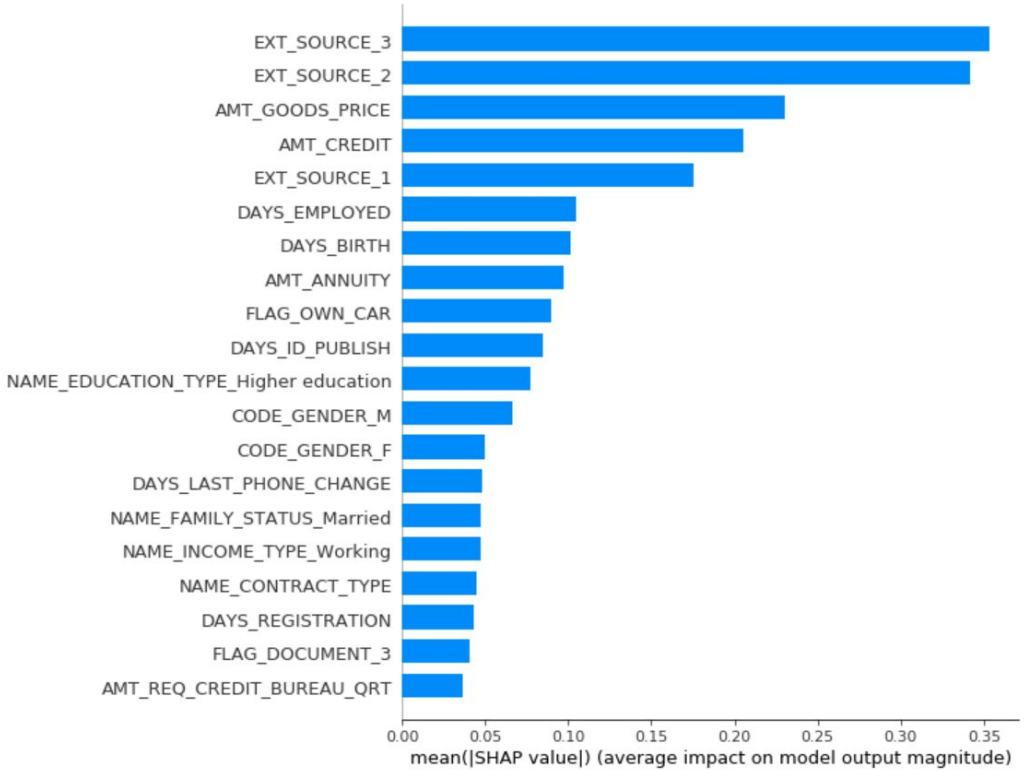
- Visualize a summary plot for each class on the whole dataset:

This summary plot shows us the features importance. Indeed, it tells which features are the most important, and also their range of effects over the dataset. In our case, the 'EXT_SOURCE_3' is the feature that affects the most, our model's prediction.

For example, we can interpret that the lower we have a value for the features ‘EXT_SOURCE_1’, ‘EXT_SOURCE_2’, ‘EXT_SOURCE_3’, the more it has an impact in our model decision.



We can have the same type of visualization by plotting the same function but with a bart type. This time we have just a rank of the feature’s importance.



Conclusion :

To conclude the database used in this project is kind of hard but it was very instructive since we had to do several models and implement it with mlflow.
It was interessant to use SHAP for a better explainability and interpretability of our Xgboost model. With this type of visualization, we are able to find which features can influence our prediction model.