Authors: CARAYON Chloé - SPATZ Cécile (BD2)
Date: 21/11/2021

Project report

Applications of Big Data

The goal of this project is to apply some concepts & tools seen in the 3 parts of the M2 *Applications of Big Data* course.

This project is organized into 3 parts :

- Part 1 : Building Classical ML projects with respect to basic ML Coding best practices
- Part 2 : Integrate MLFlow
- Part 3 : Integrate ML Interpretability

Data

Data is provided by *Home Credit*, a service dedicated to provided lines of credit (loans) to the unbanked population. Predicting whether or not a client will repay a loan or have difficulty is a critical business need. In this project we will try to help them in this task.

application_train/application_test : the main training and testing 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.

Part 1 : Building Classical ML projects with respect to basic ML Coding best practices

In this part, we built an ML Project for Home Credit Risk Classification based on the given Dataset with respect to coding best practice for production ready code.

1. To initiate our project we used a **template cookie cutter**

from https://drivendata.github.io/cookiecutter-data-science/ (https://drivendata.github.io/cookiecutter-data-science/)

It is a logical, reasonably standardized, but flexible project structure for doing and sharing data science work. We splited our project into python modules for each major part of the code. So that, we have an organized code skeleton in order to better partition our work and better visualize the different parts. We also used makefile to manage our commands.

2. Organize our work environment

Poetry to declare, manage and install dependencies of our project, ensuring we have the right stack everywhere. It is easy to use and enabled us to discover a **dependency manager**.
Makefile to define the different task to execute.

For example:

- `make install` will run `poetry install` and install all the dependencies specifies in the pyproject.toml.
- `make run` will run `poetry run python src/main.py`
- `make check` will run:

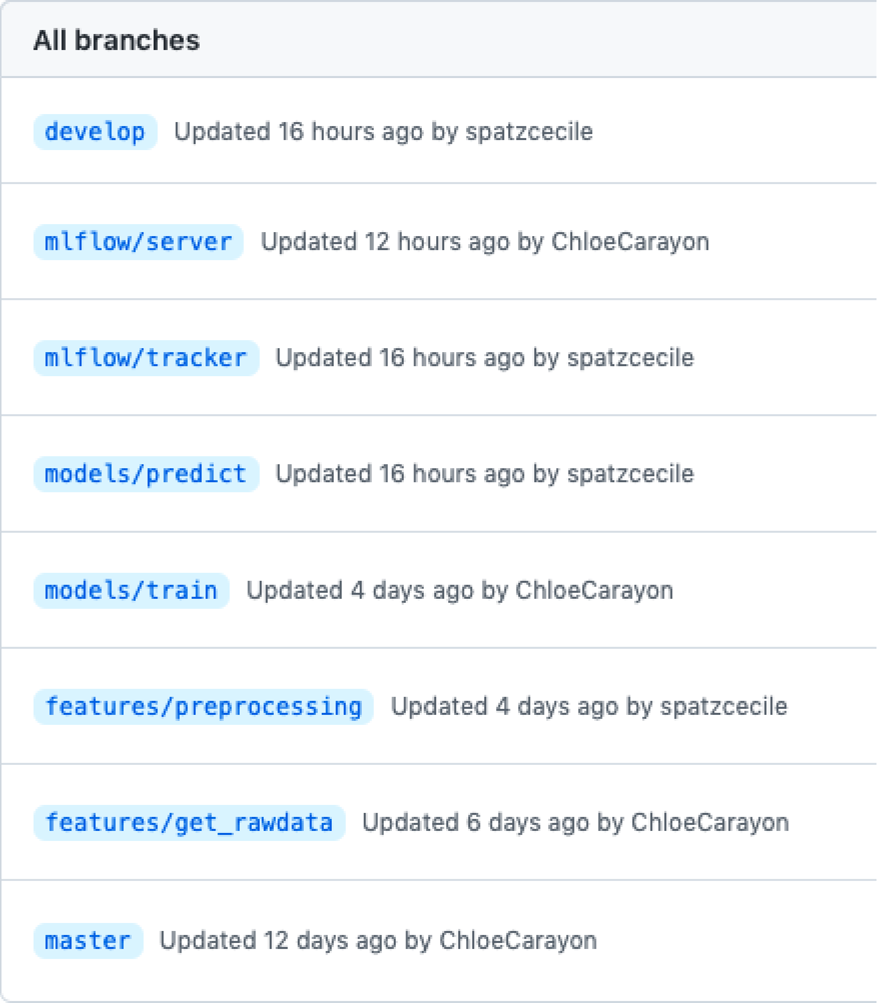
```
poetry run isort $(PY_SRC)
poetry run black $(PY_SRC)
poetry run flake8 $(PY_SRC)
```

By setting those rules, we try to make our code valid and operational for production.

3. We used GitHub

- To version control our code and try to respect Coding Best Practices:
    - we created a **repository** at the beginning of the project,
    - we defined a **branching workflow** and stick with it. We created small and well-defined branches for each part and mandatory **PRs** to check. For each branch:

{general}/{detail}



All branches

`develop` Updated 16 hours ago by spatzcecile

`mlflow/server` Updated 12 hours ago by ChloeCarayon

`mlflow/tracker` Updated 16 hours ago by spatzcecile

`models/predict` Updated 16 hours ago by spatzcecile

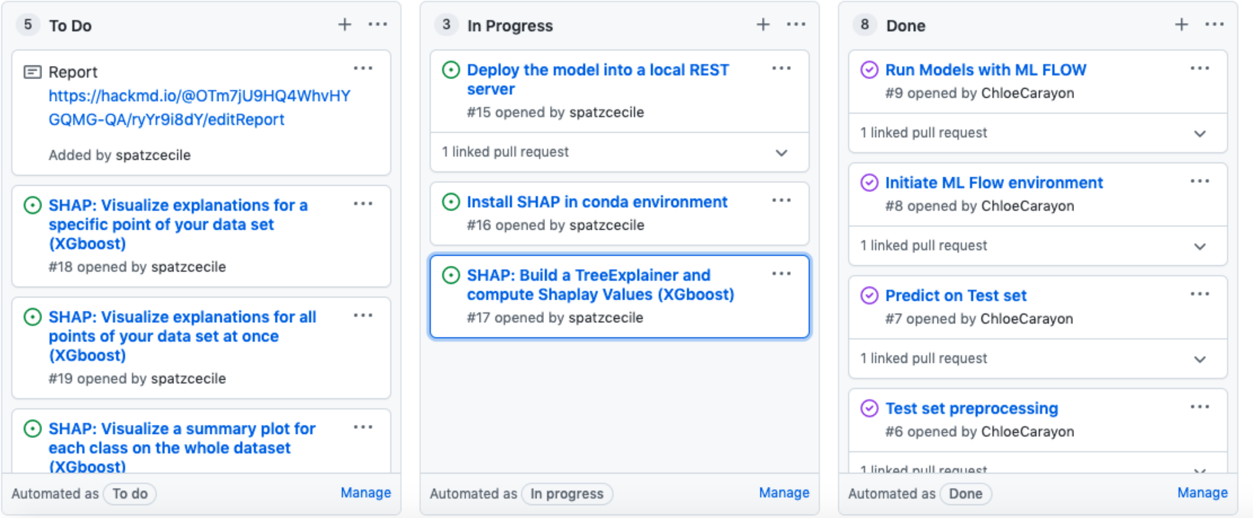
`models/train` Updated 4 days ago by ChloeCarayon

`features/preprocessing` Updated 4 days ago by spatzcecile

`features/get_rawdata` Updated 6 days ago by ChloeCarayon

`master` Updated 12 days ago by ChloeCarayon

- we **push/commit** regularly,
- we used ***.gitignore*** so that we don't push any credentials to the repository,
- we defined ***issues*** with assignement and ***project*** for better organisation,



5 To Do + ···

Report
https://hackmd.io/@OTm7jU9HQ4WhvHYGQMG-QA/ryYr9i8dY/editReport
Added by spatzcecile

SHAP: Visualize explanations for a specific point of your data set (XGboost)
#18 opened by spatzcecile

SHAP: Visualize explanations for all points of your data set at once (XGboost)
#19 opened by spatzcecile

SHAP: Visualize a summary plot for each class on the whole dataset (XGboost)

Automated as To do — Manage

3 In Progress + ···

Deploy the model into a local REST server
#15 opened by spatzcecile
1 linked pull request

Install SHAP in conda environment
#16 opened by spatzcecile

SHAP: Build a TreeExplainer and compute Shaplay Values (XGboost)
#17 opened by spatzcecile

Automated as In progress — Manage

8 Done + ···

Run Models with ML FLOW
#9 opened by ChloeCarayon
1 linked pull request

Initiate ML Flow environment
#8 opened by ChloeCarayon
1 linked pull request

Predict on Test set
#7 opened by ChloeCarayon
1 linked pull request

Test set preprocessing
#6 opened by ChloeCarayon
1 linked pull request

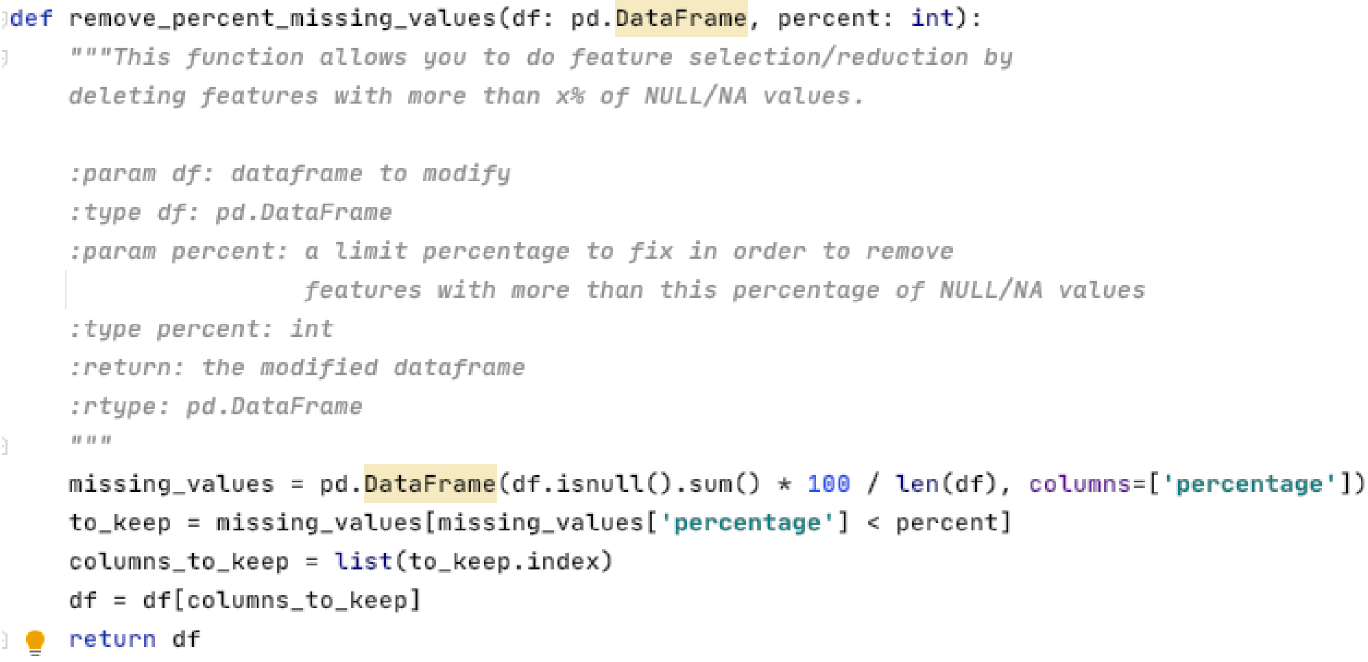
Automated as Done — Manage

- we followed https://buzut.net/cours/versioning-avec-git/bien-nommer-ses-commits (https://buzut.net/cours/versioning-avec-git/bien-nommer-ses-commits) for commits nomenclature.

4. We used a documentation library:

Sphinx with **Docstring**: it helped us to document each of our functions, refactor if the function performs several actions, briefly and efficiently explain a function, rearrange the code and avoid duplicates.

Example:



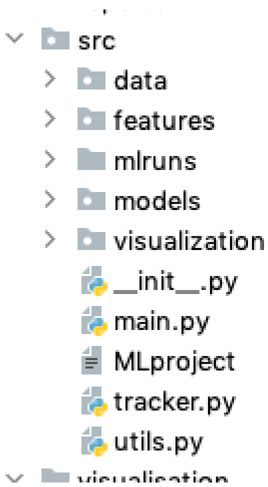
```
def remove_percent_missing_values(df: pd.DataFrame, percent: int):
    """This function allows you to do feature selection/reduction by
    deleting features with more than x% of NULL/NA values.

    :param df: dataframe to modify
    :type df: pd.DataFrame
    :param percent: a limit percentage to fix in order to remove
                    features with more than this percentage of NULL/NA values
    :type percent: int
    :return: the modified dataframe
    :rtype: pd.DataFrame
    """
    missing_values = pd.DataFrame(df.isnull().sum() * 100 / len(df), columns=['percentage'])
    to_keep = missing_values[missing_values['percentage'] < percent]
    columns_to_keep = list(to_keep.index)
    df = df[columns_to_keep]
    return df
```

5. A first approach

We first did the Machine Learning coding on a **shared notebook** for **data understanding**, **preparation**, **exploration** and **modeling** (features engineering, model building with *Xgboost*, *Random Forest* and *Gradient Boosting* and model validation on validation dataset). So that, we had a view of each other’s work and then we switched to repository to learn how to organize code into branches and also be able to use H2O not accessible on Deepnote in order to view, work and save our models in **mojo** format.

6. Modules

Then, we separated the ML projects workflow into different modules and scripts:



```
src
    data
    features
    mlruns
    models
    visualization
    __init__.py
    main.py
    MLproject
    tracker.py
    utils.py
visualisation
```

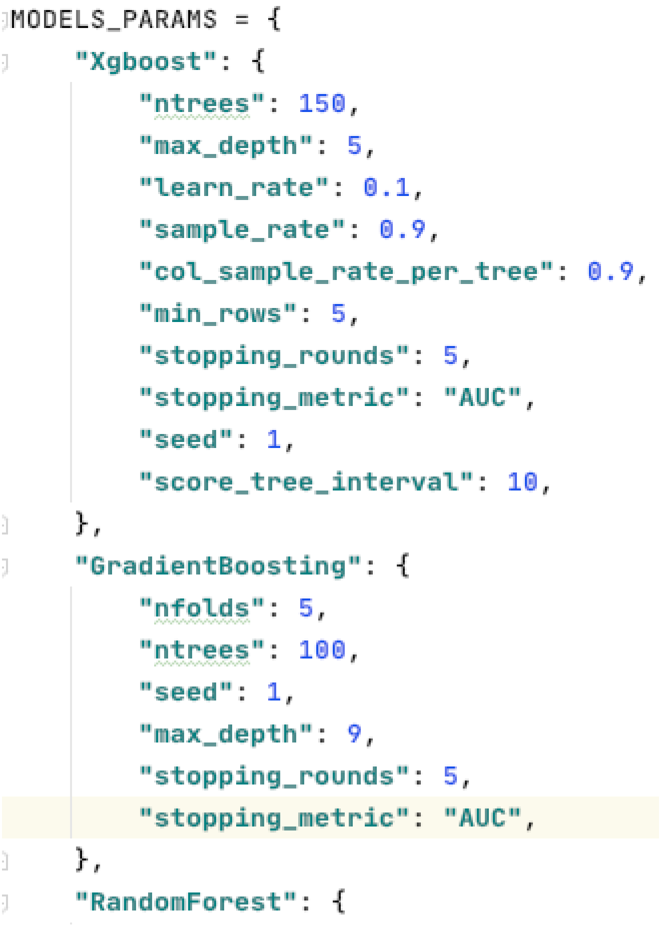
- **data module**
  To **generate dataset** we developed a code to collect data from kaggle. We used the Kaggle API to configure dataset collection functions.
  The user has to configure its Kaggle token access in order to access the competition datasets of this project.

- **features module**:
  For **data preparation (preprocessing) and feature engineering**: feature selection/reduction by deleting features with more than 30% of NULL/NA values, fill NULL/NA values with the mode for numerical columns, drop duplicates, transform days of birth column to age class, apply MinMax scaler and do one hot encoding on categorical columns, separation of the SK_ID_CURR feature corresponding to the id of each loan. We didn't forget to store scaler, mode and name of the features in a dictionary and store it in our model directory using pickle to call them during the test prediction.

- **models module**:

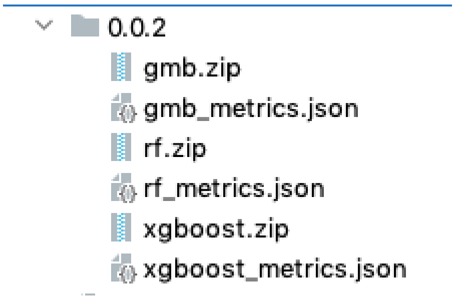
  - **training models**: we built the 3 models with **H2O** and store them in an executable format **MOJO**. H2O allows us to convert the models built to either a Plain Old Java Object (POJO) or a Model ObJect, Optimized (MOJO). It enables to visualize detailed models and it's easy to use. With it we discovered a new library.

    We also decided to use a **dictionary** to store model type associated with their hyperparameters. This allows us to easily modify the dictionnary if we wanted to change some hyperparameters.



```
MODELS_PARAMS = {
    "Xgboost": {
        "ntrees": 150,
        "max_depth": 5,
        "learn_rate": 0.1,
        "sample_rate": 0.9,
        "col_sample_rate_per_tree": 0.9,
        "min_rows": 5,
        "stopping_rounds": 5,
        "stopping_metric": "AUC",
        "seed": 1,
        "score_tree_interval": 10,
    },
    "GradientBoosting": {
        "nfolds": 5,
        "ntrees": 100,
        "seed": 1,
        "max_depth": 9,
        "stopping_rounds": 5,
        "stopping_metric": "AUC",
    },
    "RandomForest": {
```

We did **downsampling** because the dataset was very unbalanced.
We also implement a simple xgboost classifier in order to be used later in the visualisation part.
With H2O, the metrics are directly accessible and visible as we can see on the interface
Whereas for xgboost model, we compute a dictionary containing the different metrics used it for the model evaluation.
We store models as followed, given a version we store the model in mojo or pickle format.

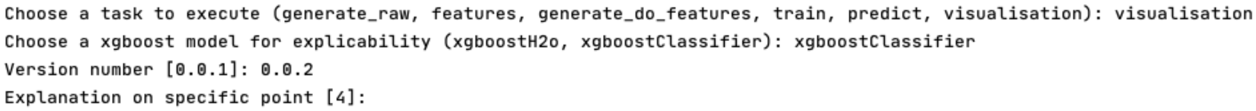


```
0.0.2
    gmb.zip
    gmb_metrics.json
    rf.zip
    rf_metrics.json
    xgboost.zip
    xgboost_metrics.json
```

- **predictions models**: we collected metadata and models in respectivelly pickle and MOJO format and then did preprocessing / feature engineering on the test set using the previously created functions for the preprocessing part. Predictions are accessible in the directory of the model which has done the prediction.

  For code partitioning, we used the library ***click*** to have a command line interface to work on the different modules.

Example with the visualisation part:



```
Choose a task to execute (generate_raw, features, generate_do_features, train, predict, visualisation): visualisation
Choose a xgboost model for explicability (xgboostH2o, xgboostClassifier): xgboostClassifier
Version number [0.0.1]: 0.0.2
Explanation on specific point [4]:
```

7. Versionning

Thank to click, the user has the possibility to select a version to store its model and use specific version for prediction part too.

This first approach on versionning models and metadata was interresting but limited even if with H2O models we have the possibility to look at the metrics which have been stored in MOJO format, we may encounter some issues with this method of versionning on the long term.

8. H2O an interesting library for MLOps introduction

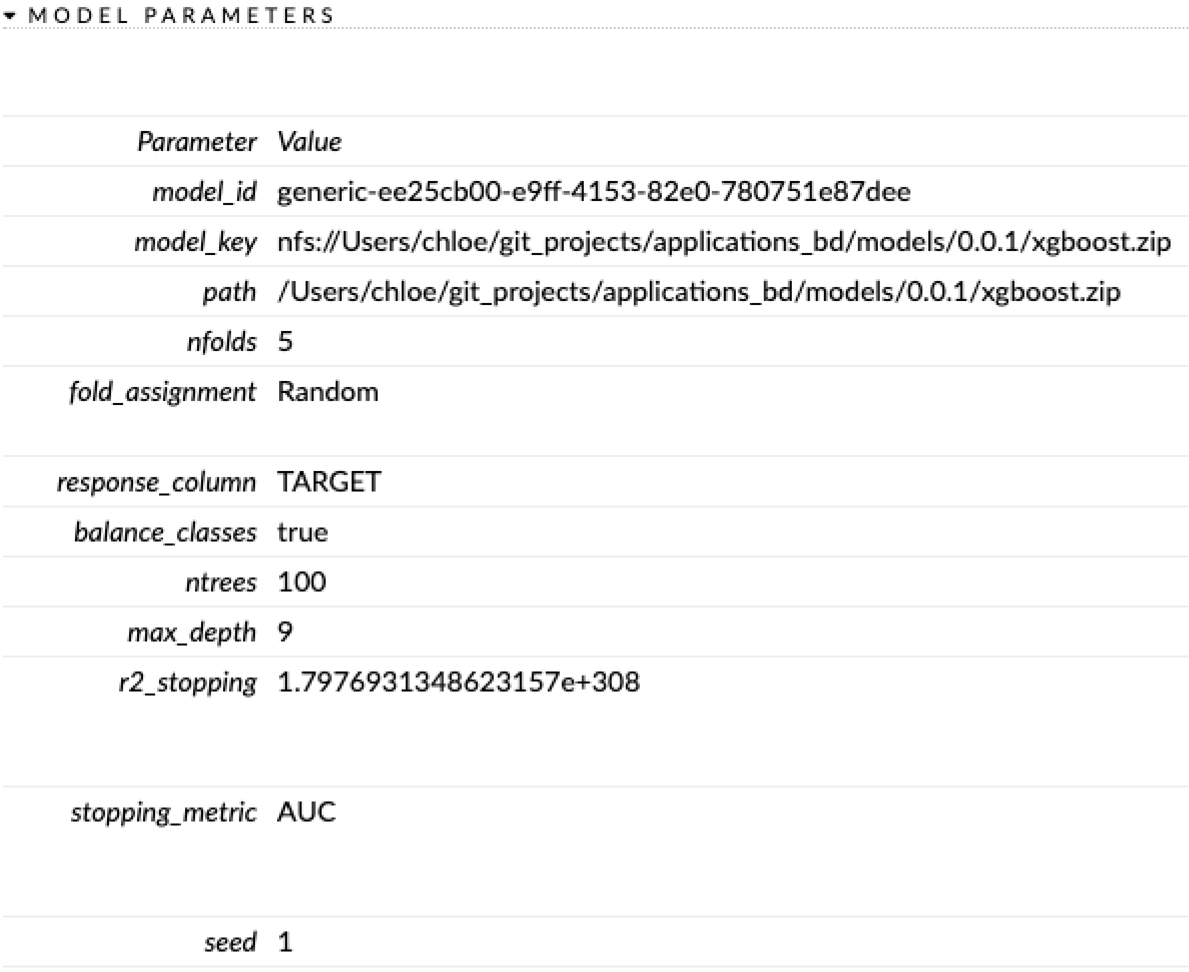
We had the opportunity to work with H2O library, we implemented GradientBoosting, Random Forest and Xgboost models using H2O.
Thanks to the web server of H2O, we can visualize our model behavior and metrics.

The H2O server is accessible when running in local at the following address:
http://127.0.0.1:54327/ (http://127.0.0.1:54327/)

As we stored our models in MOJO format, we can open them in the server to visualise the model metrics, parameters and behavior.

For example, we can look at the Xgboost model:

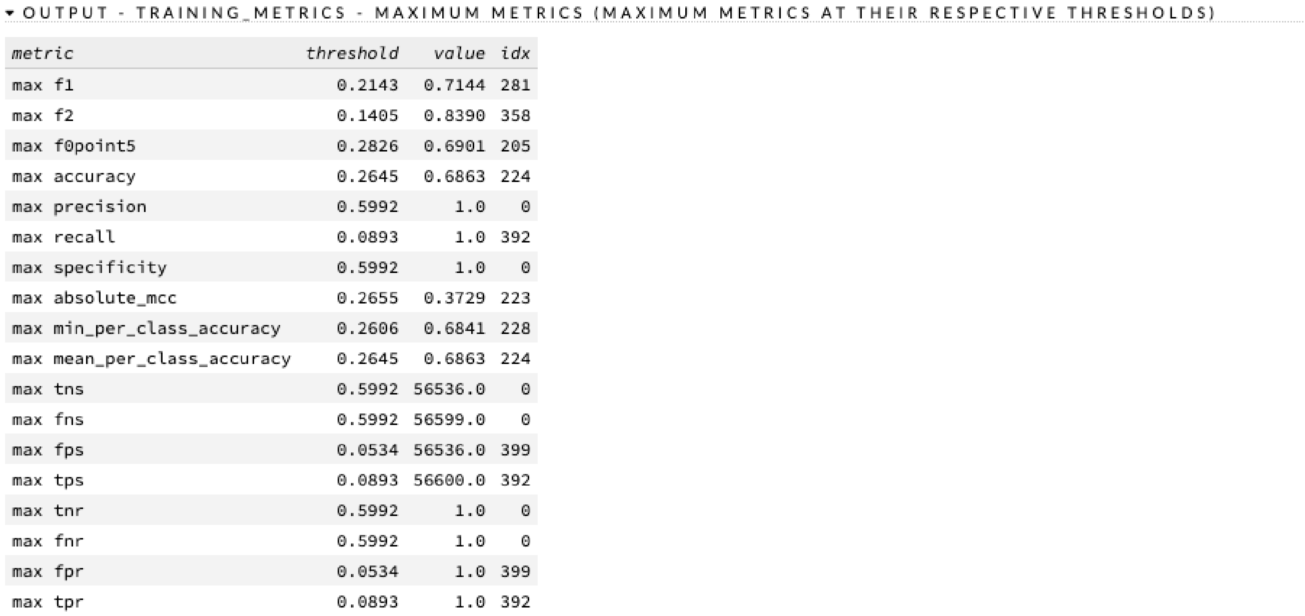
- The parameters:



▾ MODEL PARAMETERS

Parameter	Value
model_id	generic-ee25cb00-e9ff-4153-82e0-780751e87dee
model_key	nfs://Users/chloe/git_projects/applications_bd/models/0.0.1/xgboost.zip
path	/Users/chloe/git_projects/applications_bd/models/0.0.1/xgboost.zip
nfolds	5
fold_assignment	Random
response_column	TARGET
balance_classes	true
ntrees	100
max_depth	9
r2_stopping	1.7976931348623157e+308
stopping_metric	AUC
seed	1

- The maximum Metrics

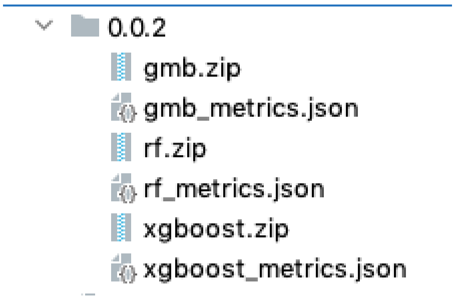


▾ OUTPUT - TRAINING_METRICS - MAXIMUM METRICS (MAXIMUM METRICS AT THEIR RESPECTIVE THRESHOLDS)

metric	threshold	value	idx
max f1	0.2143	0.7144	281
max f2	0.1405	0.8390	358
max f0point5	0.2826	0.6901	205
max accuracy	0.2645	0.6863	224
max precision	0.5992	1.0	0
max recall	0.0893	1.0	392
max specificity	0.5992	1.0	0
max absolute_mcc	0.2655	0.3729	223
max min_per_class_accuracy	0.2606	0.6841	228
max mean_per_class_accuracy	0.2645	0.6863	224
max tns	0.5992	56536.0	0
max fns	0.5992	56599.0	0
max fps	0.0534	56536.0	399
max tps	0.0893	56600.0	392
max tnr	0.5992	1.0	0
max fnr	0.5992	1.0	0
max fpr	0.0534	1.0	399
max tpr	0.0893	1.0	392

9. Scores of the models

We extract those scores from the 0.0.2 version of our models. We store the metrics in json to easily access them:



```
0.0.2
    gmb.zip
    gmb_metrics.json
    rf.zip
    rf_metrics.json
    xgboost.zip
    xgboost_metrics.json
```

In our case, we are mostly interested in the **f1score** and the **AUC**.

Moreover, as we want to predict loan acceptance, it is preferable to refuse a loan to someone who can afford it than the inverse. So we can also look at a metric which can give some weight to the precision, the **f0.5score**.

We obtain the following results:

- H2O Xgboost

```
f1: 0.68
f05: 0.67
f2: 0.79
auc: 0.80
```

- H2O Gradient Boosting

```
f1: 0.80
f05: 0.83
f2: 0.86
auc: 0.92
```

- H2O Random Forest

```
f1: 0.62
f05: 0.59
f2: 0.77
auc: 0.73
```

And with the library Xgboost

- Xgboost Classifier

```
f1score: 0.68
auc: 0.67
```

We clearly see that the predictions with the Xgboost Classifier are similar to the one with the H2O xgboost for the f1 score but the AUC is better with the H2O model.

The H20 Gradient Boosting offers result very good which can be positiv or demonstrate of some overfitting from our model. By looking at the model on the H20 server, we can access to the metrics for the validation set and compare it to the metric from the train set. It appears that the AUC is at 0.73 for validation and 0.92 for the train set. So this model is overfitting.

As we just discover the H2O library, we may have to go deeper on the model hyperparameters by doing some grid search in order to improve it. One solution could be to use MLFlow to visualize your hyperparameters and try some changes.

Part 2 : Integrate MLFlow

In this second part, we introduced MLFlow Library to the project. We followed these steps :

1. Install MLFlow in our environment
2. Track parameters of a model and display the results in our local mlflow UI
3. Deploy the model into a local REST server that enabled us to score predictions

1) Install MLFlow in our environment

In order to install MlFlow in our environment, we had to execute the following command: `poetry add mlflow`

Our python code corresponding to the mlflow part is in the tracker.py (http://tracker.py) file.

We did not work in a conda environment, so we had to specify `--no-conda` when running Mlflow commands.

We can run Mlflow examples in two different ways to visualize the results:

From the principal directory

Before going further, we had to define a MLproject in the same directory as the tracking.py (http://tracking.py) file with the following parameters:

```
name: mlflow project

entry_points:
  main:
    parameters:
      ntrees: {type: int, default: 200}
      max_depth: {type: int, default: 10}
      learn_rate: {type: float, default: 0.01}
      min_rows: {type: int, default: 5}
    command: "python tracker.py {ntrees} {max_depth} {learn_rate} {min_rows}"
```

And then, we can run:

```
poetry run mlflow run src --no-conda
```

or with hyperparameters:

```
poetry run mlflow run src --no-conda -P
ntrees=250 -P max_depth=3 -P
learn_rate=0.3 -P min_rows=8
```

The first executing creates a directory mlruns where each run will be stored.

From the `src` directory.

We can run the MLfLow project with or without the parameters

```
poetry run python tracker.py {ntrees} {max_depth} {learn_rate} {min_rows}
```

example:

```
poetry run python tracker.py 200 5 0.09 5
poetry run python tracker.py
```

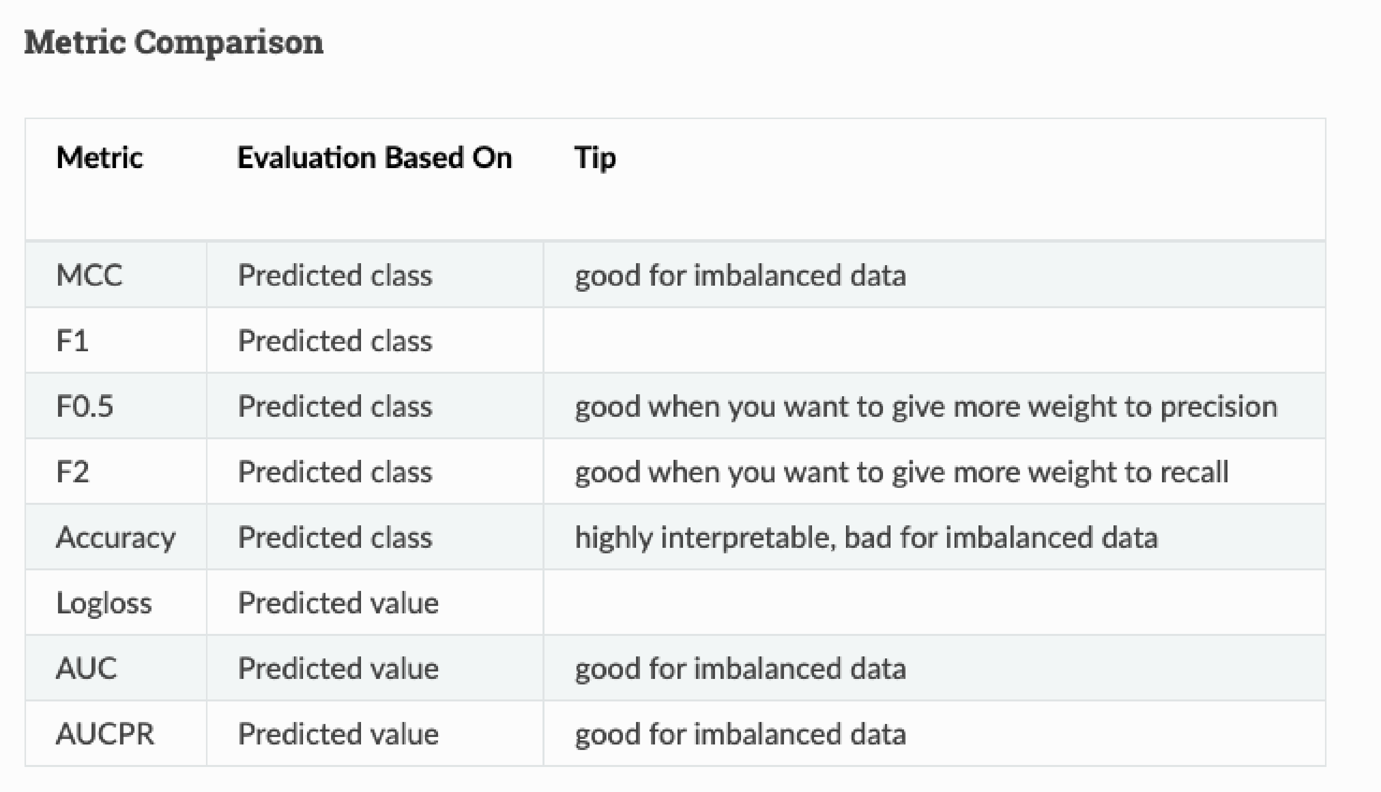
2) Track parameters of a model and display the results in our local mlflow UI

To Compare the models with MLflow UI we will run:

```
poetry run mlflow ui
```

We decide to track the parameters of one of our H2O models, the xgboost one.

Based on the documentation on H2O metrics recommanded for classification with unbalance dataset:



Metric Comparison

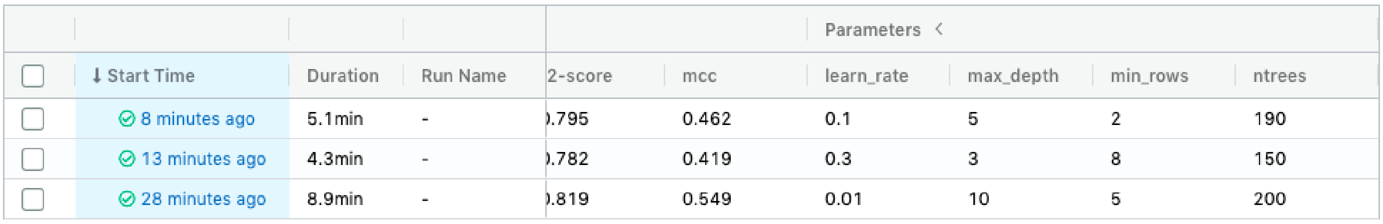
Metric	Evaluation Based On	Tip
MCC	Predicted class	good for imbalanced data
F1	Predicted class	
F0.5	Predicted class	good when you want to give more weight to precision
F2	Predicted class	good when you want to give more weight to recall
Accuracy	Predicted class	highly interpretable, bad for imbalanced data
Logloss	Predicted value	
AUC	Predicted value	good for imbalanced data
AUCPR	Predicted value	good for imbalanced data

We decide to focus on the AUC, the F1 and the F0.5.

We can visualize the mlflow ui in local server at:
http://127.0.0.1:5000/#/ (http://127.0.0.1:5000/#/)

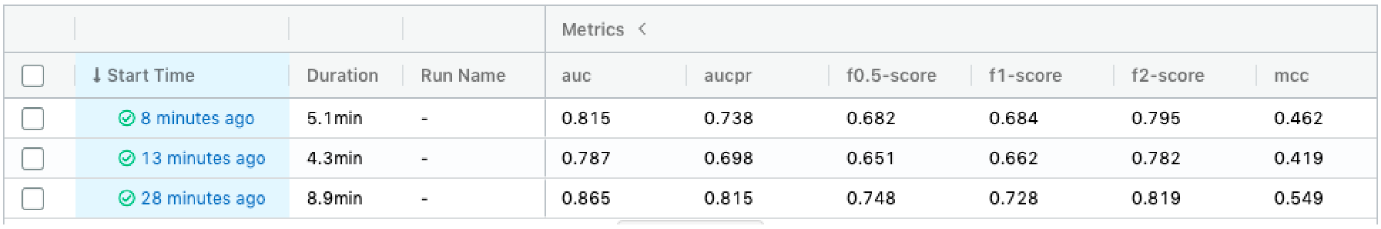
Let’s look at three experimentations.
For each experimentation, we can easily check on the parameters and metrics.

Parameters:



						Parameters ‹			
□	↓Start Time	Duration	Run Name	2-score	mcc	learn_rate	max_depth	min_rows	ntrees
□	✓ 8 minutes ago	5.1min	-	.795	0.462	0.1	5	2	190
□	✓ 13 minutes ago	4.3min	-	.782	0.419	0.3	3	8	150
□	✓ 28 minutes ago	8.9min	-	.819	0.549	0.01	10	5	200

Metrics:

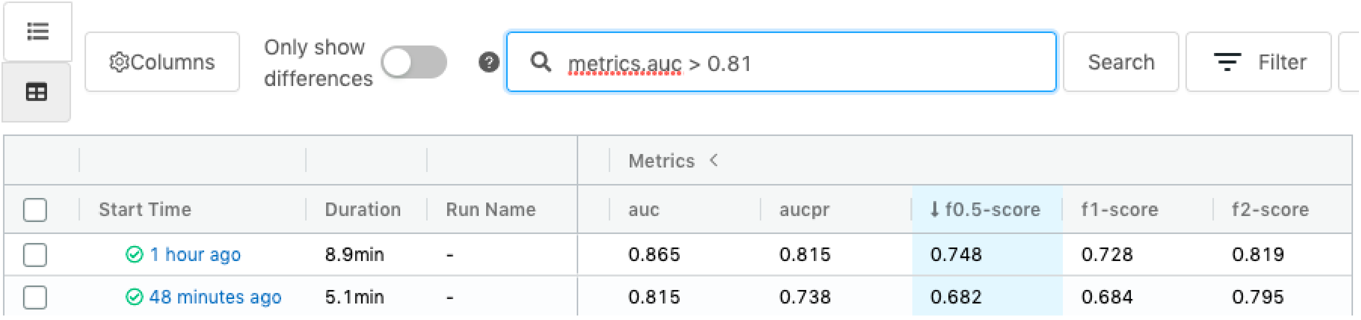


				Metrics ‹					
□	↓Start Time	Duration	Run Name	auc	aucpr	f0.5-score	f1-score	f2-score	mcc
□	✓ 8 minutes ago	5.1min	-	0.815	0.738	0.682	0.684	0.795	0.462
□	✓ 13 minutes ago	4.3min	-	0.787	0.698	0.651	0.662	0.782	0.419
□	✓ 28 minutes ago	8.9min	-	0.865	0.815	0.748	0.728	0.819	0.549

It appears that the model which gives the best performances is the first one with more than 200 trees and a max_depth of 10.
Compare to the other models, its scores are higher.

Mlflow gives us the opportunity to run several models and visualize their results.
It could be interesting to use this model which is the more performant in order to predict on the test set.

For example, let’s run a new mlflow experimentation and compare it with the old ones by doing a filter.

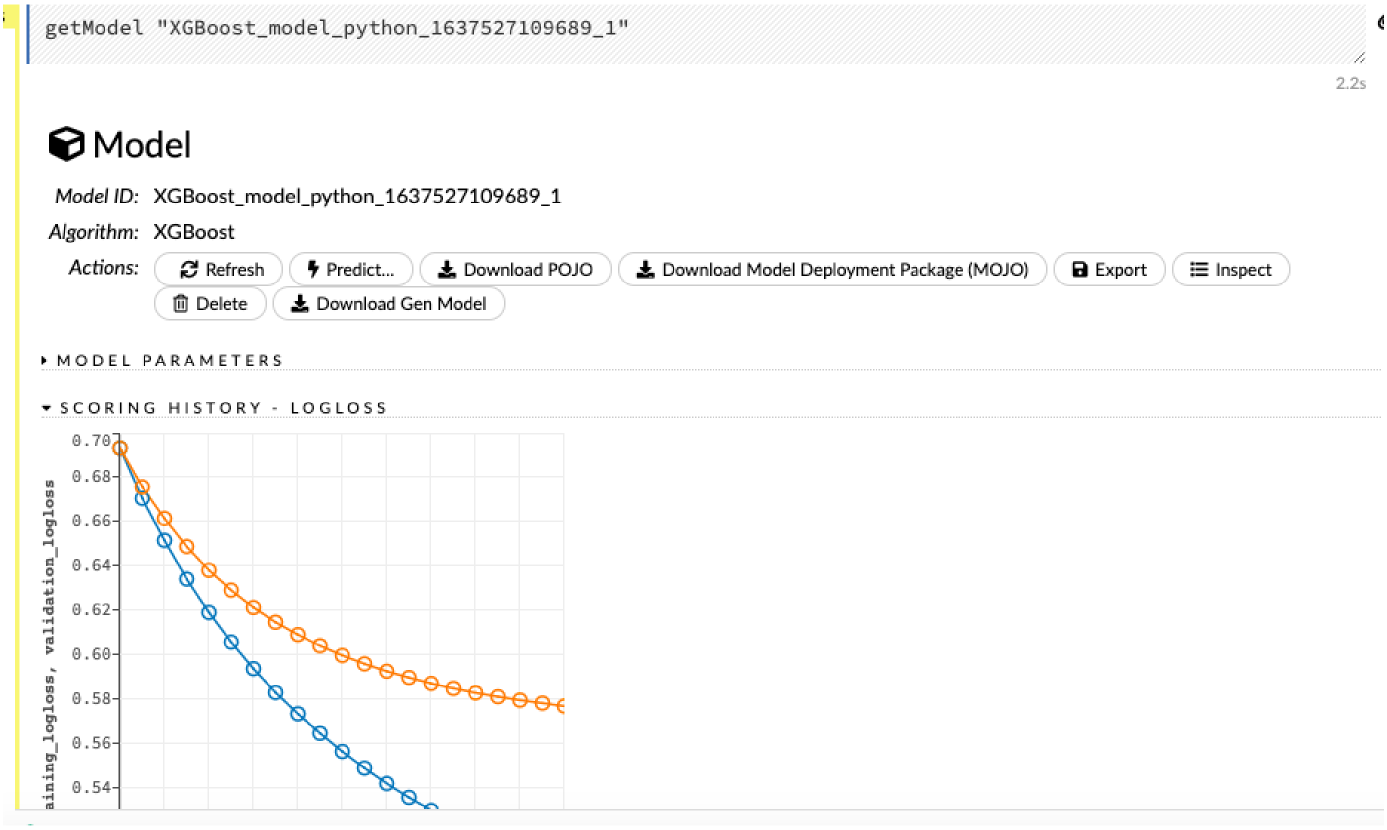


metrics.auc > 0.81

				Metrics ‹				
□	Start Time	Duration	Run Name	auc	aucpr	↓f0.5-score	f1-score	f2-score
□	✓ 1 hour ago	8.9min	-	0.865	0.815	0.748	0.728	0.819
□	✓ 48 minutes ago	5.1min	-	0.815	0.738	0.682	0.684	0.795

Like this, we can isolate some particular models.

Finally, as we used H2O models, we can from the mlflow ui interface get the path to the corresponding model we are interested in and visualize it in H20 interface:



```
getModel "XGBoost_model_python_1637527109689_1"
```

Model ID: XGBoost_model_python_1637527109689_1

Algorithm: XGBoost

3. Deploy the model into a local REST server that enabled us to score predictions

When running for example:

```
poetry run mlflow models serve -m
runs:/0a599c7b8fe6494eac8f2aeebb3d7b2e/model
--port 1234 --no-conda
```

It appears that there is an issue when trying to deploy the different models with the mlflow server. It is maybe link to our environment.

We try to use Databrick instead of the local rest but once again we encountered issues as Databrick communities free edition does not seem to include the token privilege to use with the Databrick API.

Part 3 : Integrate ML Interpretability

In this last part, we introduced **SHAP** library in our Project. We followed these steps :

1. Install SHAP in our environment
2. Use it to explain our XGboost model predictions :
    - Build a TreeExplainer and compute Shaplay Values
    - Visualize explanations for a specific point of your data set,
    - Visualize explanations for all points of your data set at once,
    - Visualize a summary plot for each class on the whole dataset.

1) Install SHAP in our environment

To install shap, we simply had to do
`poetry add shap`

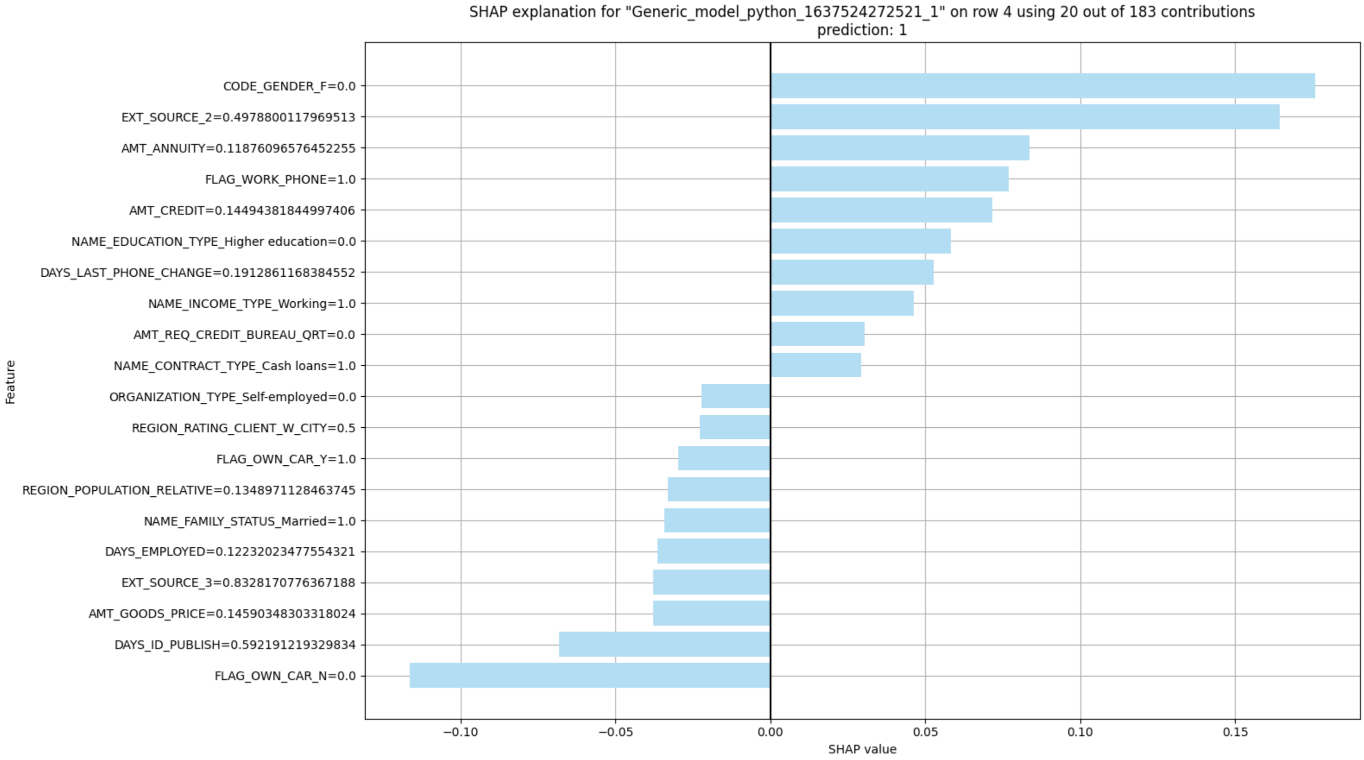
2) Use it to explain our XGboost model predictions

We can split this part in two sub parts. For both models, we will load the model (respectively import Mojo and pickle load) and the preprocessed test set and will do explicaility on it.

H20 xgboost model
Firstly, we wanted to work on the h2O xgboost model. By looking at the documentation of H2O, we found out that H2O models integrate shap library: https://docs.h2o.ai/h2o/latest-stable/h2o-docs/explain.html (https://docs.h2o.ai/h2o/latest-stable/h2o-docs/explain.html).

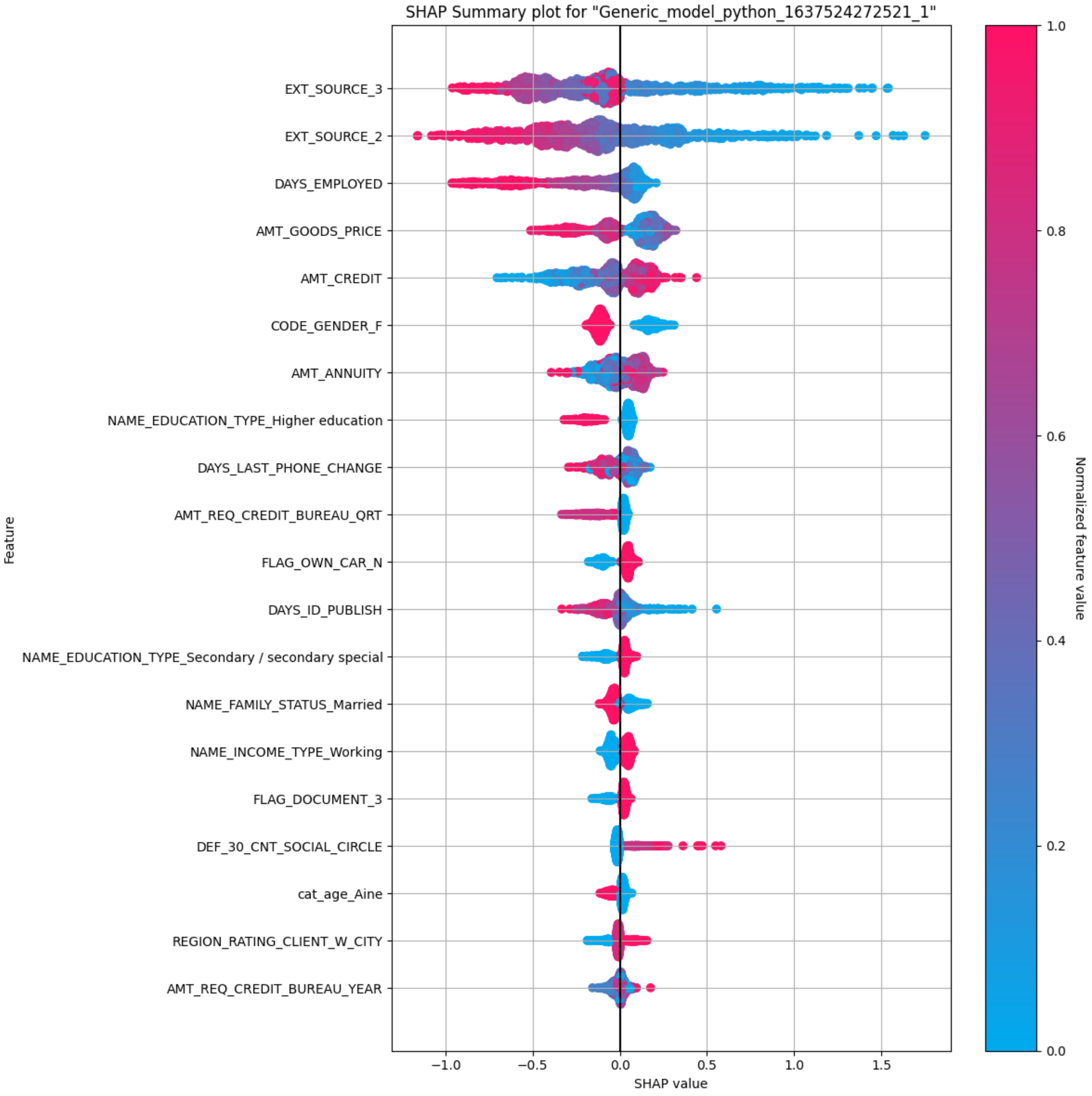
- Build a TreeExplainer and compute Shaplay Values
  H2O already implements a TreeExplainer so we don't have to compute it.

- Visualize explanations for a specific point of your data set,
  Let's take the point 4 and visualize the shap explanation.
  by running
  `model.shap_explain_row_plot(test_h2o, row_index=row_index)`
  with row_index = 4



For row 4, CODE_GENDER and EXT_SOURCE_2 are influential variables for the model whereas DAYS_ID_PUBLISH and FLAG_OWN_CAR_N have a negative influence on the model.

- Visualize a summary plot for each class on the whole dataset.
  by running `model.shap_summary_plot(test_h2o)`



Here, we can conclude that:

- low values of EXT_SOURCE_2 and EXT_SOURCE_3 caused higher predictions (the loan was not repaid) and high values caused low predictions (the loan was repaid);
- high values of DAYS_EMPLOYED caused low predictions so the more the person worked in his life the more he has chance to refund his credit;
- being a woman influence positively the credit refund because high values of CODE_GENDER_F caused low predictions.

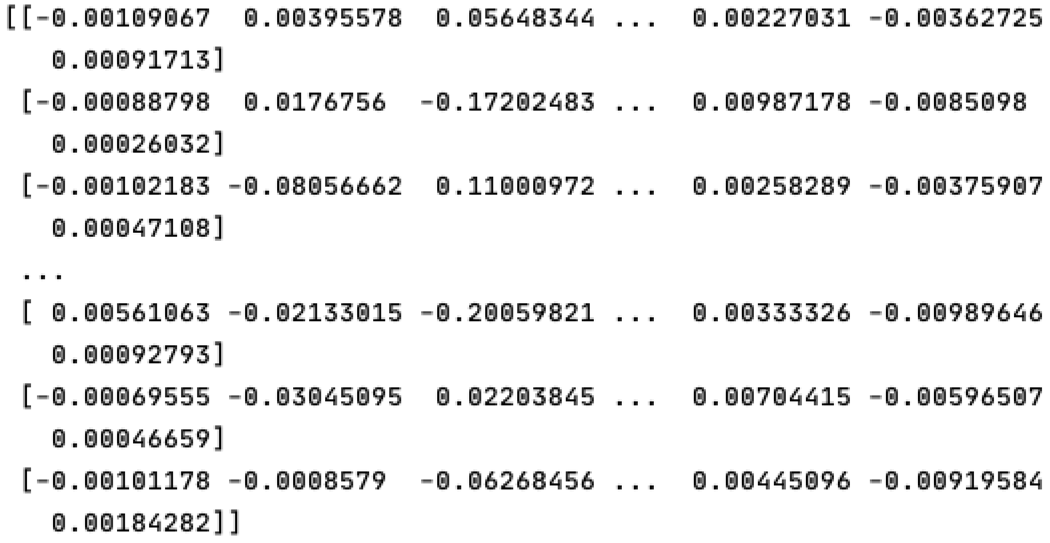
Nevertheless, it only includes two functions of shap and it was too complicate to transform the contribution scores.
That is why, we decide to also work on the xgboost model train in the previous part.

xgboost model

- Build a TreeExplainer and compute Shaplay Values

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

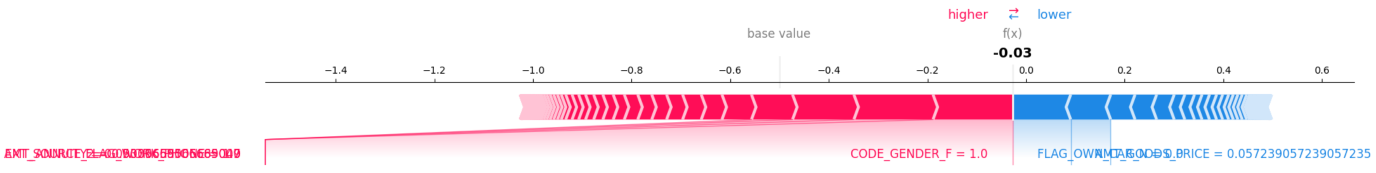
Shap values



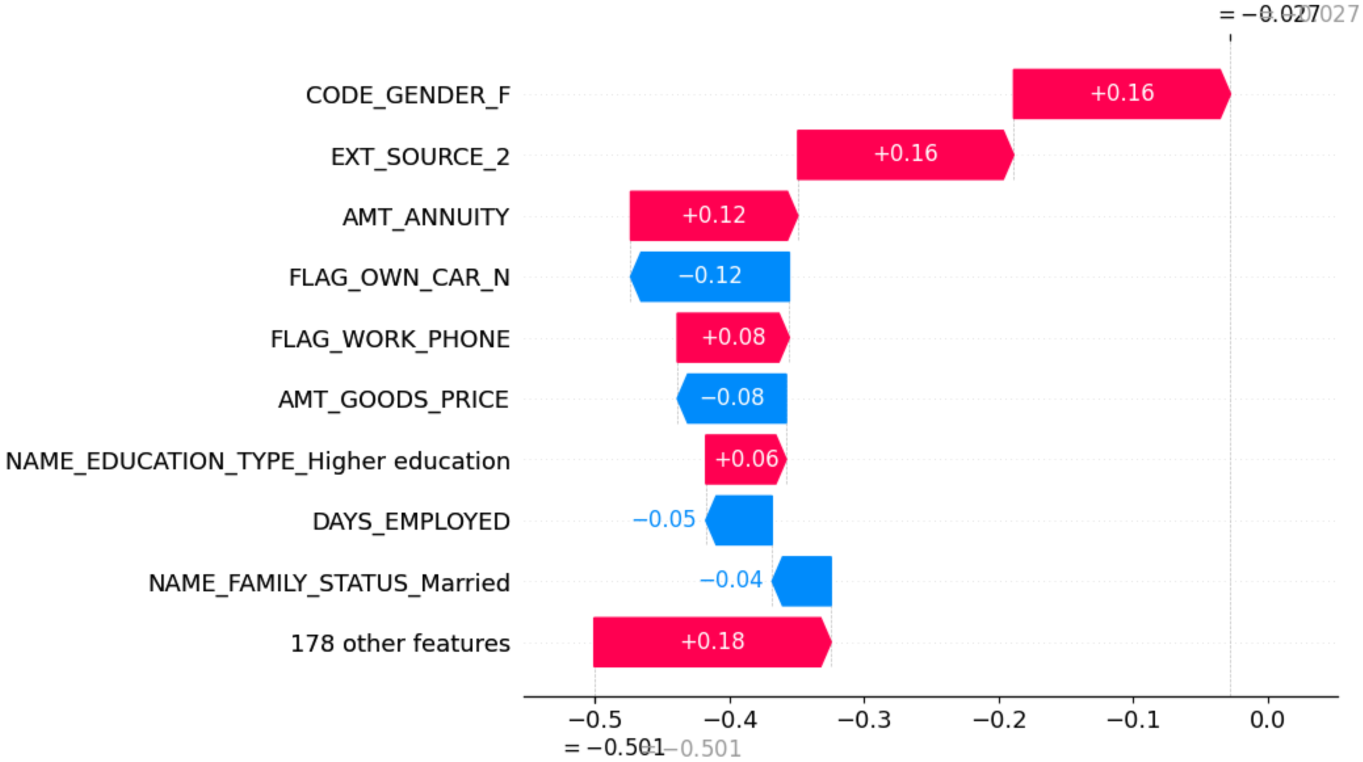
```
[[-0.00109067  0.00395578  0.05648344 ...  0.00227031 -0.00362725
   0.00091713]
 [-0.00088798  0.0176756  -0.17202483 ...  0.00987178 -0.0085098
   0.00026032]
 [-0.00102183 -0.08056662  0.11000972 ...  0.00258289 -0.00375907
   0.00047108]
 ...
 [ 0.00561063 -0.02133015 -0.20059821 ...  0.00333326 -0.00989646
   0.00092793]
 [-0.00069555 -0.03045095  0.02203845 ...  0.00704415 -0.00596507
   0.00046659]
 [-0.00101178 -0.0008579  -0.06268456 ...  0.00445096 -0.00919584
   0.00184282]]
```

- Visualize explanations for a specific point of your data set

We firstly implement a force plot but as we have many columns, the output is difficult to interpret.

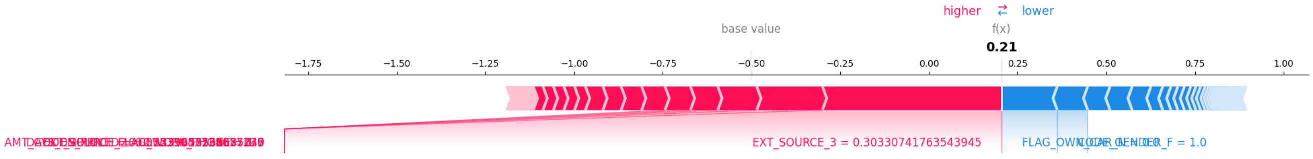


So we decide to do a **waterfall** to focus on the most important features for the point 4.

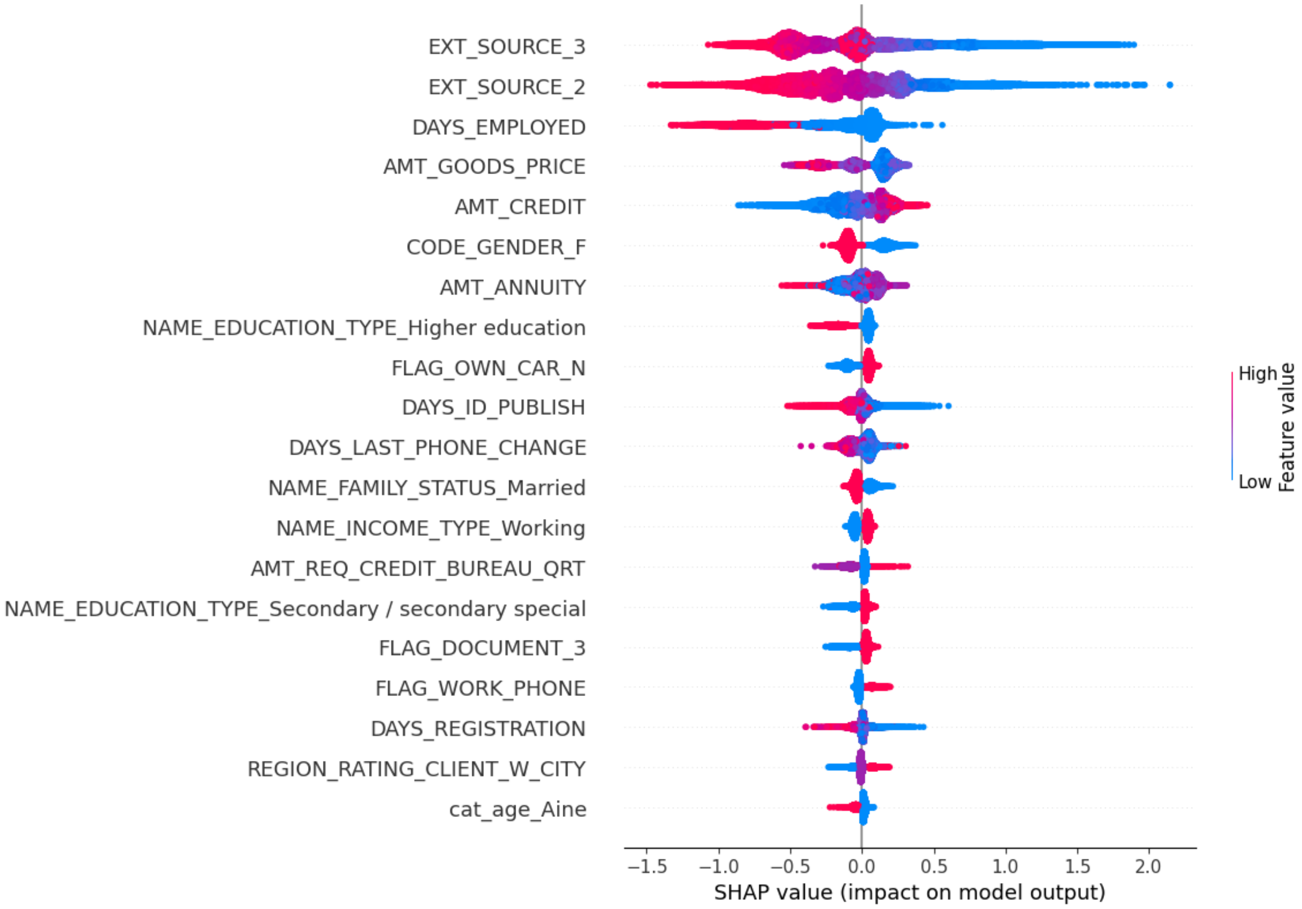


Here, for the point 4, we can say that being married decreases the prediction, then the days employed also decreases the predictions and then having a “higher education” increase the prediction etc etc.

- Visualize explanations for all points of your data set at once,
  Once again, we wanted to plot a force plot but it is not lisible.



Like with the H2O xgboost, we plot a summary showing the importance of the features for the entire dataset

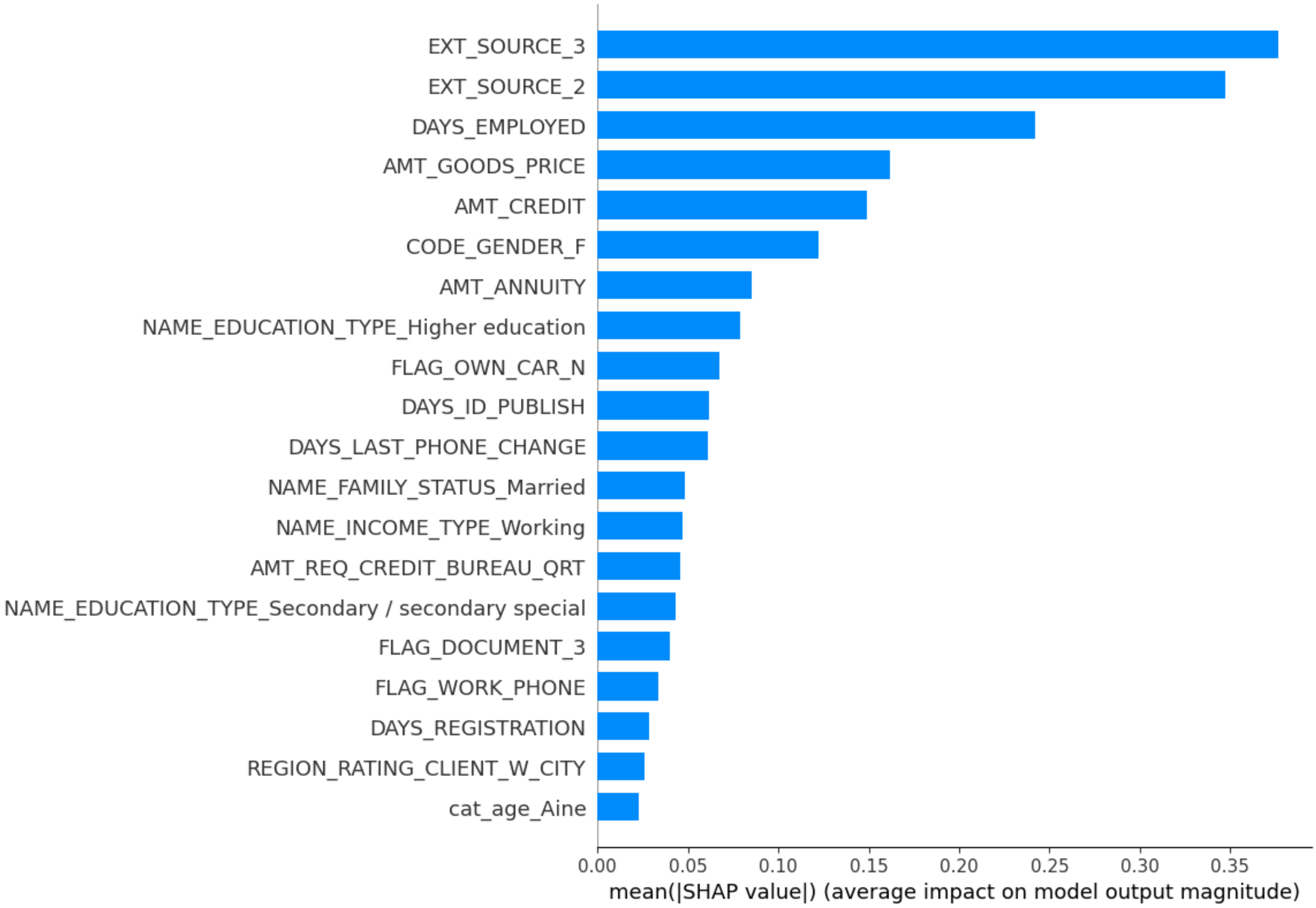


Here, we can conclude, as above, that:

- low values of EXT_SOURCE_2 and EXT_SOURCE_3 caused higher predictions (the loan was not repaid) and high values caused low predictions (the loan was repaid);
- high values of DAYS_EMPLOYED caused low predictions so the more the person worked in his life the more he has chance to refund his credit;
- being a woman influence positively the credit refund because high values of CODE_GENDER_F caused low predictions.

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

So, the shap summary in bar plot the features by decreasing importance:



We can say that EXT_SOURCE_3 is the most important feature, changing the prediction probability on average by nearly 40 percentage points (0.04 on x-axis). EXT_SOURCE_2 and DAYS_EMPLOYED are also important features.

Conclusion

This project was very interesting as we could discover how to do **MLOps**:

- building a classical ML project with respect to basic ML coding best practices,
- integrate MLFlow,
- integrate ML Interpretability.

We discoverd some **new technologies** like Sphinx, MLFlow or SHAP.

Finaly, using GIT was very important for **good collaboration** within the team. Version control through Git allowed us to easily manage changes to projects, keep track of various versions of source code and collaborate on any similar section of code without creating conflicting issues.