Prévision des défauts sur les lignes de production Valéo

Index:

Récapitulatif du Reste à faire

1. Feature engineering: **Point 8 de l'index ci dessus**
   - Appliquer une transformation log10 pour les features dont les distributions sont asymétriques
   - Enrichir les données avec l'horodatage d'assemblage en les extrayant de l'identifiant technique 'PROC_TRACEINFO'
   - Dans les histogrammes 9.c qui représentent la distribution des "features numériques" sur les 2 classes OK et KO, on constate que la classe minoritaire se retrouve délimité à l'intérieur d'une plage de valeurs pour certains features. (ex: OP070_V_1/2_angle_value, OP110_Vissage_M8_torque_Value). Pour cela, il faudrait vérifier l'impact si on transforme ces features numériques continues en des features catégoriques mettant en avant l'existence de la classe minoritaire KO pour ces catégories.
   - **La mise en place du Feature engineering va induire la regénération du 7, du 8 et 9.**

2. Compléter avec 3 classifieurs de type différents: à base d'arbre / à base de distance / à base de reseau de Neuronne ou bien de Stacking: **Points 11, 12, 13 de l'index ci dessus**
   - Pour chaque classifieur faire: TrainTestSplit / CV / SearhGridCV
   - Pour chaque classifieur: Analyse et interpretation des résultats / Matrice de confusion / F1, Roc / Graphe F1 et ROC
   - **Si le temps le permet** alors faire pour chaque classifieur: Graphe Overfit Underfit / Graphe avec des valeurs differents des hyperparamètres.

3. Expliquer pourquoi une classification déséquilibrée pose un défi pour la modélisation prédictive.

4. Identifier la/les motivations pour avoir une distribution Normale "bell shape". Citer les avantages d'une telle distribution.
5. Mise en forme selon les indications du document "DSSP14_Guidelines_Projet_Professionnel.doc"

Questions auxquelles j'aimerais avoir une réponse:

1. Quand on a une distribution asymétrique pour une feature et qu'on voudrait lui appliquer une transformation logarithmique pour s'approcher d'une distribution normale:

   – Est ce qu'il vaut mieux appliquer la transformation sur les features asymétriques seulement ?
     Ou bien on peut l'appliquer sur la totalité des features de la dataframe (ça nous évite de choisir une-par-une les features à transformer)
   – Cette transformation sera appliqué sur le TrainSet;
     Est ce qu'il faut l'appliquer aussi sur le TestSet au moment de la prédiction ?

2. Les opérations d'imputations(ex: IterativeImputer) et de scaling (ex: RobustScaling) sont appliquées sur le TrainSet afin d'honorer les pré-requis d'apprentissage de certains algorithmes de machine learning.

   Est-ce que la prédiction sur une observation (TestSet) fonctionnera correctement au cas où ***l'observation*** pour laquelle on effectue la prédiction:

   – Manque certains features en "missing values" ?
   – Ou bien si les features de l'observation ne sont pas scalés selon l'attente de l'algon en phase d'apprentissage ?

3. Quand on fait des splits de Train/Test par Cross Validation, comme le **11.b** et le **12.b**, la méthode 'cross_validate(..)' retourne autant de fitted classifiers qu'il y de folds.

   C'est à dire: Si le Cross Validation est effectué sur 5 folds, alors la méthode 'cross_validate' retroune 5 fitted classifiers.

   Questions:

   – Parmi ces fitted classifiers, lequel faut il choisir afin de l'utiliser ?
   – Est ce qu'on choisit celui dont le roc_auc est le plus élevé ? tel que c'était fait dans le **11.b** et le **12.b**

4. Est ce qu'il faut commenter davantage les graphes et les mesures figurant dans les chapitres 'Exploration des données', chapitres 5,6 et 7 ?

5. Au niveau de ce document, est ce qu'il faut agrandir les graphes ?

Ou bien, ils seront consultés sur un support électronique (pdf, doc, projection, ...) et par conséquent ils seront agrandis électroniquement ?

6. Quelles sont les parties que je dois développer davantage ? Ou bien être plus concis ?

7. Est-ce qu'il faut garder dans le document, les petits bout de code Python qu'on trouve tout au long des chapitres (hors chapitre 16 – Annexe : code Python) ? Ou bien il faut les supprimer ?

***Merci** :-)*

1 - Contexte de l'étude

L'étude correspond à un 'Challenge Data ENS' qui a pour objectif de prévoir les défauts sur les lignes de production des démarreurs de l'équipementier Valeo. Lors de l'assemblage des démarreurs sur la ligne de production, les différentes valeurs (couples, angles ...) sont mesurées sur les différentes stations de montage.

En fin de ligne, des mesures supplémentaires sont effectuées sur deux bancs de test afin d'isoler les défauts. Par conséquent, les échantillons sont étiquetés "OK" ou "KO". L'objectif est de concevoir un modèle qui pourrait identifier de tels défauts avant l'étape du banc d'essai.

L'étude concerne la classification des données déséquilibrée avec des valeurs de données manquantes. C'est un problème classique dans l'industrie et dans bien d'autres domaines : détection de fraude, détection de spam, domaine médical, ...

Les classifications déséquilibrées posent un défi pour la modélisation prédictive. La classe minoritaire est plus importante et donc le problème est plus sensible aux erreurs de classification pour la classe minoritaire que pour la classe majoritaire.

A l’heure de la rédaction de ce document, mon meilleur modèle est basé sur le classifieur « Balanced Random Forest », il occupe le 65ème rang sur un nombre total de 116 participants. Mon score (roc_auc) est égal à 0.6344 sur une plage allant de 0.4 jusqu’à 0.76. J’ai déjà identifié certaines pistes d’amélioration que je n’ai pas encore implémentées, notamment au niveau « features engineering ».

2 - Description du data set case

a - Entrées:

Les caractéristiques d'entrée sont des mesures collectées sur différentes stations d'assemblage avec des capteurs connectés à des contrôleurs logiques programmables qui les stockent tous.

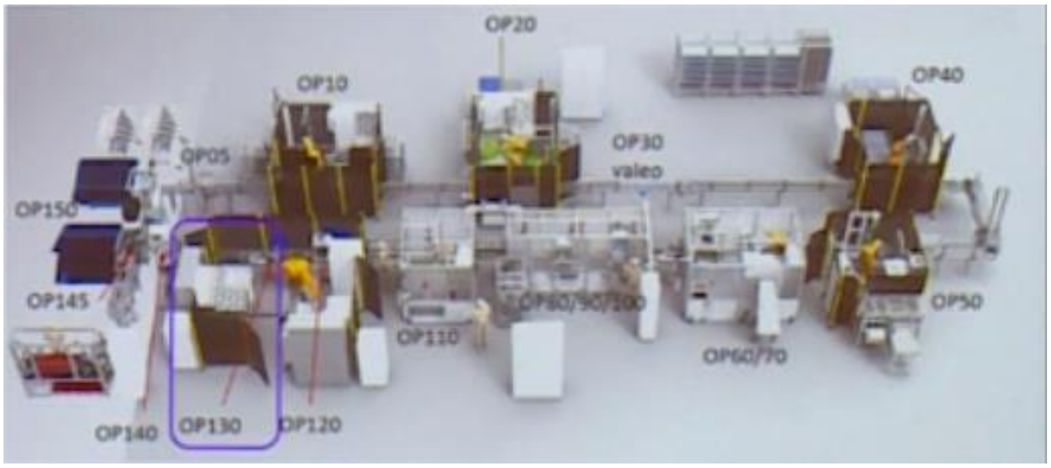
On distingue par exemple:

OP070_V_1_angle_value	V1 Valeur d'angle,
OP070_V_1_torque_value	V1 Valeur de couple,
OP070_V_2_angle_value	V2 Valeur d'angle,
OP070_V_2_torque_value	V2 Valeur de couple,
OP090_StartLinePeakForce_value	Start Line Peak Force value,
OP090_SnapRingMidPointForce_value	Anneau élastique Mid Point Force val,
OP090_SnapRingPeakForce_value	Anneau élastique Peak Force value,
OP090_SnapRingFinalStroke_value	Valeur finale du coup d'arrêt,
OP100_Capuchon_insertion_mesure	Mesure d'insertion capuchon
OP110_Vissage_M8_angle_value	Valeur d'angle Vissage M8,
OP110_Vissage_M8_torque_value	Valeur de couple Vissage M8,
OP120_Rodage_I_mesure_value	Rodage I mesure la valeur,
OP120_Rodage_U_mesure_value	Rodage U mesure value,

b - Sortie:

Il s'agit de la valeur de résultat de l'OP130, banc d'essai: Binar OP130_Resultat_Global_v.

La valeur 0 est affectée aux échantillons OK (réussie) et la valeur 1 est affectée aux échantillons KO (échoué). Il s'agit du résultat combiné de multiples tests électriques, acoustiques et vibro-acoustiques.



L'objectif est de trouver la meilleure prédiction: Sortie = f (entrées). L'ensemble de données contient 34515 échantillons d'apprentissage et 8001 échantillons de test.

c - Croisement Entées/Sortie:

Les données de training sont réparties dans 2 fichiers csv:

- [project-root]/data/train/traininginputs.csv
- [project-root]/data/train/trainingoutput.csv

Un identifiant technique 'PROC_TRACEINFO' permet de croiser le fichier d'entrée au fichier de sortie.

C'est un code unique donné attribué au démarreur assemblé.

Exemple: I-B-XA1207672-190701-00494.

- XA1207672 est la référence.
- 190701 est la date: ici le 01 juillet de l'année 2019.
- 00494 est le code unique donné au produit, ce nombre est augmenté de 1 pour chaque nouveau produit.

On dispose aussi des données d'entrée de test: [project-root]/data/test/testinputs.csv

Les données de sortie de test sont générés par l'étude et sont uploader sur la plateforme 'Data Challenge ENS' https://challengedata.ens.fr/participants/challenges/36/

3 - Import des packages et rechargement automatique des packages du projets

```
import os
import sys
import logging
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE


from imblearn.ensemble import BalancedBaggingClassifier, RUSBoostClassifier,
BalancedRandomForestClassifier
from imblearn.over_sampling import RandomOverSampler, ADASYN, SMOTE, SVMSMOTE, KMeansSMOTE,
BorderlineSMOTE
from imblearn.over_sampling.base import BaseOverSampler
from imblearn.pipeline import Pipeline

from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier, RandomForestClassifier
from sklearn.ensemble._hist_gradient_boosting.gradient_boosting import HistGradientBoostingClassifier
from sklearn.cluster import MiniBatchKMeans
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model._stochastic_gradient import SGDClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, NuSVC, LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_validate, StratifiedKFold

import xgboost as xgb

# Import "valeo" module
sys.path.append("..")
from valeo.infrastructure.LogManager import LogManager as lm
# NB: Initializing logger here allows "class loaders of application classes" to benefit from the global
initialization
logger = lm().logger(__name__)
from valeo.infrastructure import Const
from valeo.infrastructure.tools.DfUtil import DfUtil
from valeo.infrastructure.tools.ImgUtil import ImgUtil
from valeo.infrastructure.XY_Loader import XY_Loader
from valeo.infrastructure.XY_metadata import XY_metadata as XY_metadata
from valeo.domain.ValeoModeler import ValeoModeler
from valeo.domain.ValeoPredictor import ValeoPredictor
import valeo.infrastructure.Transformer as transf



# Notebook automatic reload
%load_ext autoreload
%reload_ext autoreload
%aimport valeo.infrastructure.Transformer
%aimport valeo.infrastructure.LogManager
%aimport valeo.infrastructure.Const
%aimport valeo.infrastructure.tools.DfUtil
%aimport valeo.infrastructure.tools.ImgUtil
%aimport valeo.infrastructure.XY_Loader
%aimport valeo.infrastructure.XY_metadata
%aimport valeo.domain.ValeoModeler
%aimport valeo.domain.ValeoPredictor
```

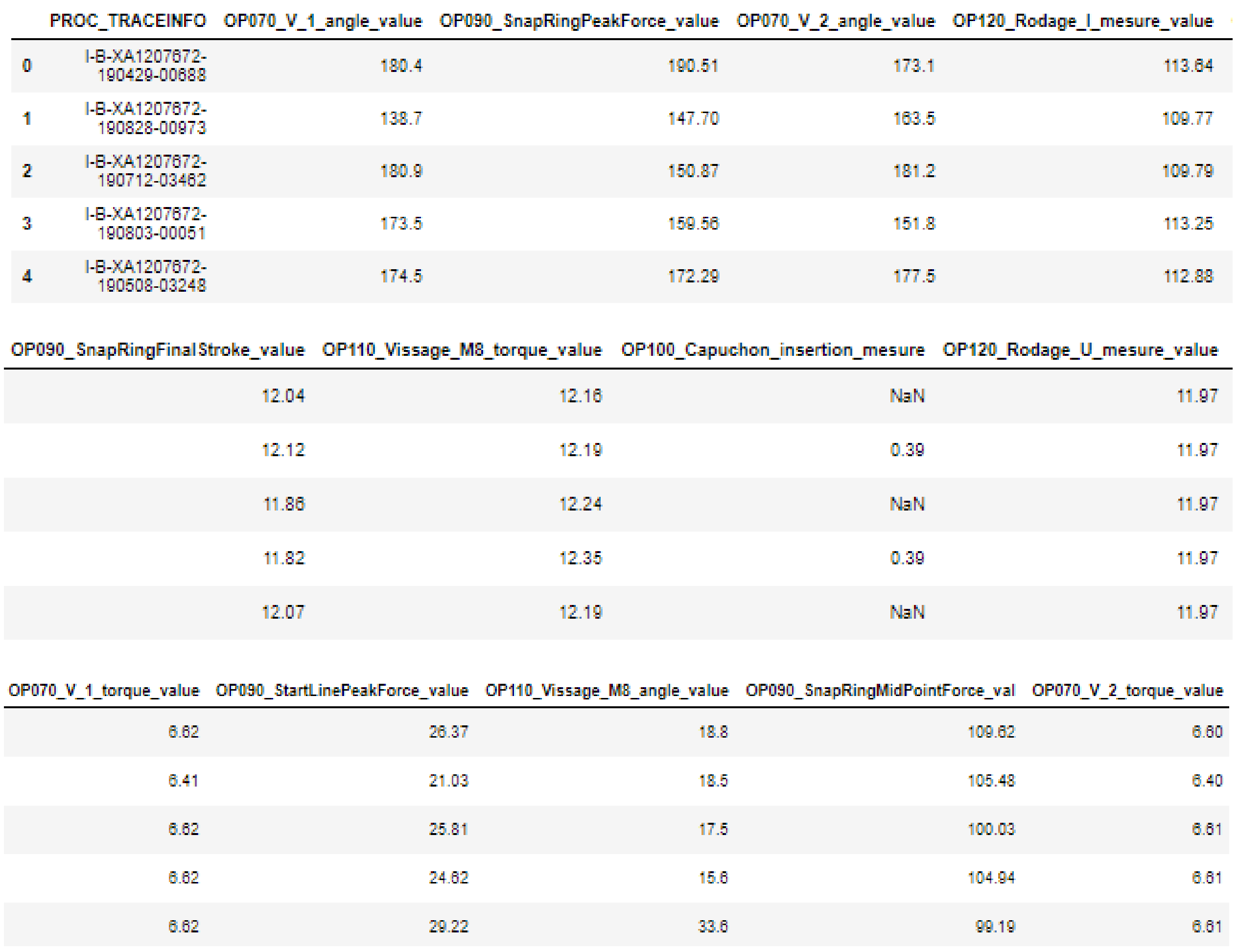
4 - Chargement des données 'Training'

```
data = DfUtil.read_csv([Const.rootDataTrain() , "traininginputs.csv"])
Y_data = DfUtil.read_csv([Const.rootDataTrain(), "trainingoutput.csv"])
```

5 - Exploration et analyse tabulaire des données

a - Visualisation tabulaire des données - Affichage du type 'head()':

```
data.head()
```



	PROC_TRACEINFO	OP070_V_1_angle_value	OP090_SnapRingPeakForce_value	OP070_V_2_angle_value	OP120_Rodage_I_mesure_value	OP090_SnapRingFinalStroke_value	OP110_Vissage_M8_torque_value	OP100_Capuchon_insertion_mesure	OP120_Rodage_U_mesure_value	OP070_V_1_torque_value	OP090_StartLinePeakForce_value	OP110_Vissage_M8_angle_value	OP090_SnapRingMidPointForce_val	OP070_V_2_torque_value
0	I-B-XA1207672-190429-00688	180.4	190.51	173.1	113.64	12.04	12.16	NaN	11.97	6.62	26.37	18.8	109.62	6.60
1	I-B-XA1207672-190828-00973	138.7	147.70	163.5	109.77	12.12	12.19	0.39	11.97	6.41	21.03	18.5	105.48	6.40
2	I-B-XA1207672-190712-03462	180.9	150.87	181.2	109.79	11.86	12.24	NaN	11.97	6.62	25.81	17.5	100.03	6.61
3	I-B-XA1207672-190803-00051	173.5	159.56	151.8	113.25	11.82	12.35	0.39	11.97	6.62	24.62	15.6	104.94	6.61
4	I-B-XA1207672-190508-03248	174.5	172.29	177.5	112.88	12.07	12.19	NaN	11.97	6.62	29.22	33.6	99.19	6.61

Un simple affichage du type 'head()' permet de voir à quoi ressemble les données.

b - Rapport semantique des données - Affichage du type 'info()':

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34515 entries, 0 to 34514
Data columns (total 14 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
```

```
 0   PROC_TRACEINFO                    34515 non-null  object 
 1   OP070_V_1_angle_value             34515 non-null  float64
 2   OP090_SnapRingPeakForce_value     34515 non-null  float64
 3   OP070_V_2_angle_value             34515 non-null  float64
 4   OP120_Rodage_I_mesure_value       34515 non-null  float64
 5   OP090_SnapRingFinalStroke_value   34515 non-null  float64
 6   OP110_Vissage_M8_torque_value     34515 non-null  float64
 7   OP100_Capuchon_insertion_mesure   15888 non-null  float64
 8   OP120_Rodage_U_mesure_value       34515 non-null  float64
 9   OP070_V_1_torque_value            34515 non-null  float64
 10  OP090_StartLinePeakForce_value    34515 non-null  float64
 11  OP110_Vissage_M8_angle_value      34515 non-null  float64
 12  OP090_SnapRingMidPointForce_val   34515 non-null  float64
 13  OP070_V_2_torque_value            34515 non-null  float64
dtypes: float64(13), object(1)
memory usage: 3.7+ MB
```

Un affichage sémantique du type 'info()' met en évidence le type des données et le nombre des valeurs manquantes 'missing values'.

On constate que:

- Toutes les features sont numériques et continues, pas de features catégoriques
- Plus de la moitié des valeurs de la feature 7 ' OP100_Capuchon_insertion_mesure' sont manquants => Cette feature doit être traitée en lui imputant des valeurs. Un imputer de type IterativeImputer(stratégie 'médiane') ser a utilisé.
- PROC_TRACEINFO de type object (=> String), c'est l'identifiant de ligne permettant de croiser les 'features' avec la 'target'. Cette feature porte l'horodatage de l'assemblage des démarreurs, la date sous jacente sera extraite et utilisée dans la phase de 'features engineering'

c - Données manquantes par type de 'feature':

```
data.isna().sum()
```

```
PROC_TRACEINFO                         0
OP070_V_1_angle_value                  0
OP090_SnapRingPeakForce_value          0
OP070_V_2_angle_value                  0
OP120_Rodage_I_mesure_value            0
OP090_SnapRingFinalStroke_value        0
OP110_Vissage_M8_torque_value          0
OP100_Capuchon_insertion_mesure    18627
OP120_Rodage_U_mesure_value            0
OP070_V_1_torque_value                 0
OP090_StartLinePeakForce_value         0
OP110_Vissage_M8_angle_value           0
OP090_SnapRingMidPointForce_val        0
OP070_V_2_torque_value                 0
dtype: int64
```

L'identifiant 'PROC_TRACEINFO' est supprimé **provisoirement** de l'ensemble des features:

```
X_data = data.drop(columns = "PROC_TRACEINFO")
X_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34515 entries, 0 to 34514
Data columns (total 13 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   OP070_V_1_angle_value            34515 non-null  float64
 1   OP090_SnapRingPeakForce_value    34515 non-null  float64
 2   OP070_V_2_angle_value            34515 non-null  float64
 3   OP120_Rodage_I_mesure_value      34515 non-null  float64
 4   OP090_SnapRingFinalStroke_value  34515 non-null  float64
 5   OP110_Vissage_M8_torque_value    34515 non-null  float64
 6   OP100_Capuchon_insertion_mesure  15888 non-null  float64
 7   OP120_Rodage_U_mesure_value      34515 non-null  float64
 8   OP070_V_1_torque_value           34515 non-null  float64
 9   OP090_StartLinePeakForce_value   34515 non-null  float64
 10  OP110_Vissage_M8_angle_value     34515 non-null  float64
 11  OP090_SnapRingMidPointForce_val  34515 non-null  float64
 12  OP070_V_2_torque_value           34515 non-null  float64
dtypes: float64(13)
memory usage: 3.4 MB
```

d - Statistique descriptive

```
X_data.sort_index(axis=1).describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
OP070_V_1_angle_value	34515.0	159.906922	15.662650	101.80	148.70	158.00	169.30	198.30
OP070_V_1_torque_value	34515.0	6.548403	0.097602	5.67	6.41	6.61	6.62	6.67
OP070_V_2_angle_value	34515.0	159.618236	15.091490	82.00	149.40	158.70	168.90	198.10
OP070_V_2_torque_value	34515.0	6.550867	0.094814	5.74	6.42	6.61	6.61	6.67
OP090_SnapRingFinalStroke_value	34515.0	11.970190	0.169873	0.00	11.85	12.04	12.08	12.19
OP090_SnapRingMidPointForce_val	34515.0	97.700978	6.837714	0.00	94.31	98.50	102.23	127.30
OP090_SnapRingPeakForce_value	34515.0	156.915055	11.271492	0.00	149.21	156.18	164.38	196.92
OP090_StartLinePeakForce_value	34515.0	23.630152	2.546341	0.00	22.28	23.88	25.29	43.41
OP100_Capuchon_insertion_mesure	15888.0	0.388173	0.024425	0.24	0.38	0.39	0.41	0.42
OP110_Vissage_M8_angle_value	34515.0	17.878398	6.785079	6.30	13.50	16.40	20.20	84.60
OP110_Vissage_M8_torque_value	34515.0	12.256785	0.065319	12.03	12.21	12.26	12.30	12.50
OP120_Rodage_I_mesure_value	34515.0	113.350222	3.528522	99.99	111.04	113.16	115.38	177.95
OP120_Rodage_U_mesure_value	34515.0	11.971027	0.003050	11.97	11.97	11.97	11.97	11.99

On constate que :

- OP070_V_2_angle_value : Outlier côté Min => Utiliser un 'robust scaler' pour réduire l'effet Outlier

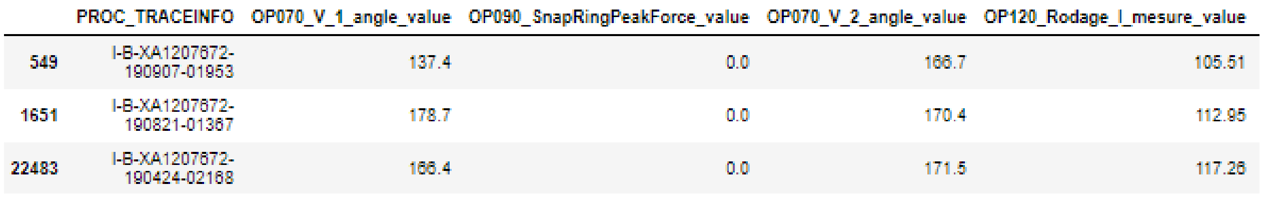
- OP090_StartLinePeakForce_value, OP090_SnapRingMidPointForce_val, OP090_SnapRingPeakForce_value, OP090_SnapRingFinalStroke_value :

  Identification de valeurs nulle, 'min' égal à 0. Normalement ces mesures physiques ne doivent pas être nulle, le fait qu'elles soient nulles laisse penser qu'elles sont nulles à tort et par conséquent il faut les considérer comme des valeurs manquantes et seront traitées dans la phase **'Feature Engineering'**

- OP090_StartLinePeakForce_value : Outlier côté Max => Utiliser un 'robust scaler'
- OP110_Vissage_M8_angle_value : Outlier côté Max => Utiliser un 'robust scaler'
- OP110_Vissage_M8_torque_value : Presque Constant
- OP100_Capuchon_insertion_mesure: Plus de la moitié sans valeurs => Utiliser 'missing values' Imputer
- OP120_Rodage_U_mesure_value : Très petite variance
- OP120_Rodage_I_mesure_value : Outlier cote Max (=> Utiliser un 'robust scaler') + petite variance

Nombre de features égales à Nulle:

```
data.query('OP090_StartLinePeakForce_value == 0 or OP090_SnapRingMidPointForce_val == 0 or
            OP090_SnapRingPeakForce_value == 0 or OP090_SnapRingFinalStroke_value == 0')
```



	PROC_TRACEINFO	OP070_V_1_angle_value	OP090_SnapRingPeakForce_value	OP070_V_2_angle_value	OP120_Rodage_I_mesure_value
549	I-B-XA1207672-190907-01953	137.4	0.0	166.7	105.51
1651	I-B-XA1207672-190821-01367	178.7	0.0	170.4	112.95
22483	I-B-XA1207672-190424-02168	166.4	0.0	171.5	117.26

Seulement 3 observations dont les valeurs des features sont égales à 0. A cela s'ajoute la moitié des valeurs de 'OP100_Capuchon_insertion_mesure' qui sont manquantes.

Ratio d'observations ayant des features en outlier:

```
Q1 = X_data.quantile(0.25)
Q3 = X_data.quantile(0.75)
IQR = Q3 - Q1
#
outliers = ((X_data < (Q1 - 1.5 * IQR)) |(X_data > (Q3 + 1.5 * IQR))).any(axis=1)
print(f"Le ratio d'outlier est de {len(X_data[outliers].index)/len(X_data.index)}")
```

```
Le ratio d'outlier est de 0.24256120527306968
```

Un ratio élevé => L'éventualité de supprimer les observations n'est pas viable. D'autant plus qu'on ne connait pas la raison de ces outliers:

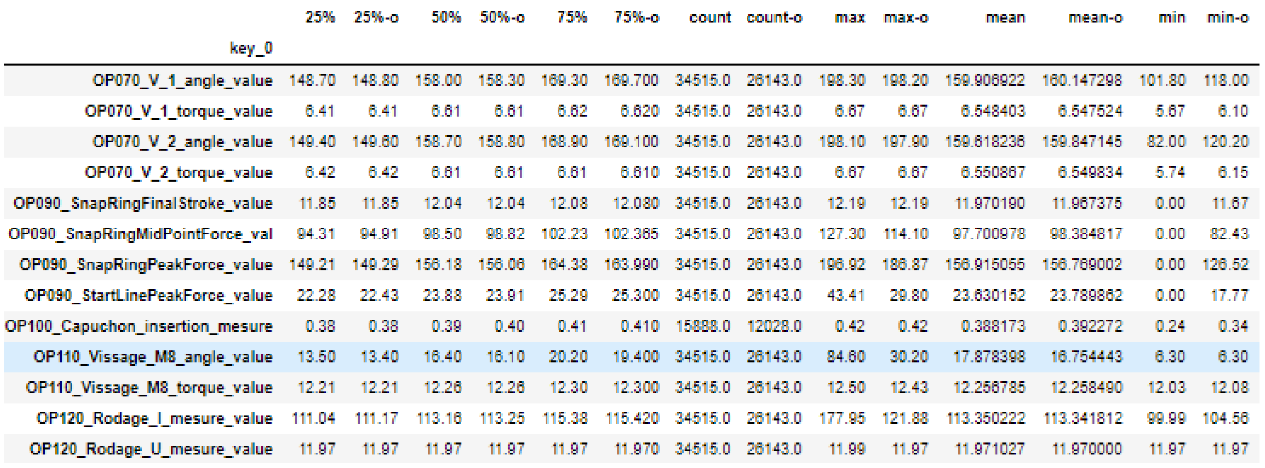
- Est ce que c'est une erreur
- Ou bien c'est une vrai donnée dont le pattern est différent

Pour limiter l effet des outliers:

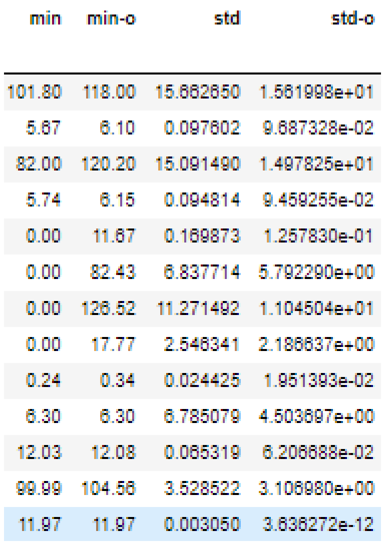
- Utliser un modèle resistant aux outliers, comme les arbres
- Tranformer les données en utilisant la fonction Log Lors de la visualisation graphique des données on va retrouver des distributions biaisée (skewed)

Comparaison des valeurs statistiques entres le dataset initiale et le dataset dépourvu des outliers

```
# 1 - Le dataset dépourvu des outliers
X_data_out = X_data[~outliers]

# 2 - Creéer les 2 dataframes des valeurs statistique descriptive
Xt =  X_data.sort_index(axis=1).describe().transpose()
Xt_out =  X_data_out.sort_index(axis=1).describe().transpose()

# 3 - Fusionner les afin de pouvoir les comparer
xt_merged = pd.merge(left=Xt, right=Xt_out, how='inner', left_on=Xt.index, right_on=Xt_out.index,
suffixes=('','-o'))
xt_merged = xt_merged.set_index(['key_0'])
xt_merged.sort_index(axis=1)
```



key_0	25%	25%-o	50%	50%-o	75%	75%-o	count	count-o	max	max-o	mean	mean-o	min	min-o
OP070_V_1_angle_value	148.70	148.80	158.00	158.30	169.30	169.700	34515.0	26143.0	198.30	198.20	159.906922	160.147298	101.80	118.00
OP070_V_1_torque_value	6.41	6.41	6.61	6.61	6.62	6.620	34515.0	26143.0	6.67	6.67	6.548403	6.547524	5.67	6.10
OP070_V_2_angle_value	149.40	149.60	158.70	158.80	168.90	169.100	34515.0	26143.0	198.10	197.90	159.618236	159.847145	82.00	120.20
OP070_V_2_torque_value	6.42	6.42	6.61	6.61	6.61	6.610	34515.0	26143.0	6.67	6.67	6.550867	6.549834	5.74	6.15
OP090_SnapRingFinalStroke_value	11.85	11.85	12.04	12.04	12.08	12.080	34515.0	26143.0	12.19	12.19	11.970190	11.967375	0.00	11.67
OP090_SnapRingMidPointForce_val	94.31	94.91	98.50	98.82	102.23	102.365	34515.0	26143.0	127.30	114.10	97.700978	98.384817	0.00	82.43
OP090_SnapRingPeakForce_value	149.21	149.29	156.18	156.06	164.38	163.990	34515.0	26143.0	196.92	186.87	156.915055	156.769002	0.00	126.52
OP090_StartLinePeakForce_value	22.28	22.43	23.88	23.91	25.29	25.300	34515.0	26143.0	43.41	29.80	23.630152	23.789862	0.00	17.77
OP100_Capuchon_insertion_mesure	0.38	0.38	0.39	0.40	0.41	0.410	15888.0	12028.0	0.42	0.42	0.388173	0.392272	0.24	0.34
OP110_Vissage_M8_angle_value	13.50	13.40	16.40	16.10	20.20	19.400	34515.0	26143.0	84.60	30.20	17.878398	16.754443	6.30	6.30
OP110_Vissage_M8_torque_value	12.21	12.21	12.26	12.26	12.30	12.300	34515.0	26143.0	12.50	12.43	12.256785	12.258490	12.03	12.08
OP120_Rodage_I_mesure_value	111.04	111.17	113.16	113.25	115.38	115.420	34515.0	26143.0	177.95	121.88	113.350222	113.341812	99.99	104.56
OP120_Rodage_U_mesure_value	11.97	11.97	11.97	11.97	11.97	11.970	34515.0	26143.0	11.99	11.97	11.971027	11.970000	11.97	11.97



min	min-o	std	std-o
101.80	118.00	15.662650	1.561998e+01
5.67	6.10	0.097602	9.687328e-02
82.00	120.20	15.091490	1.497825e+01
5.74	6.15	0.094814	9.459255e-02
0.00	11.67	0.169873	1.257830e-01
0.00	82.43	6.837714	5.792290e+00
0.00	126.52	11.271492	1.104504e+01
0.00	17.77	2.546341	2.186637e+00
0.24	0.34	0.024425	1.951393e-02
6.30	6.30	6.785079	4.503697e+00
12.03	12.08	0.065319	6.206688e-02
99.99	104.56	3.528522	3.106980e+00
11.97	11.97	0.003050	3.636272e-12

D'une manière générale, les valeurs sont approximativement similaires, sauf pour le 'max' et le 'std' de quelques features:

- OP090_StartLinePeakForce_value, OP110_Vissage_M8_angle_value, OP120_Rodage_I_mesure_value: Le 'max' a chuté considérablement
- OP110_Vissage_M8_angle_value : Variance considérablement plus petite

e - Distribution du jeux des données:

```
starter_count = len(Y_data[Const.Binar_OP130_Resultat_Global_v])
starter_count_ok = Y_data[Const.Binar_OP130_Resultat_Global_v].value_counts()[0]
starter_count_ko = Y_data[Const.Binar_OP130_Resultat_Global_v].value_counts()[1]
#
print(f'Nombre total des démarreurs : {starter_count}')
print(f'Nombre total des démarreurs OK => Nombre de Classes Negatives : {starter_count_ok} soit
{round(starter_count_ok/starter_count * 100,2)} % du dataset')
print(f'Nombre total des démarreurs KO => Nombre de Classes Positives : {starter_count_ko} soit
{round(starter_count_ko/starter_count * 100,2)} % du dataset')
```

```
Nombre total des démarreurs : 34515
Nombre total des démarreurs OK => Nombre de Classes Negatives : 34210 soit
99.12 % du dataset
Nombre total des démarreurs KO => Nombre de Classes Positives : 305 soit 0.88
% du dataset
```

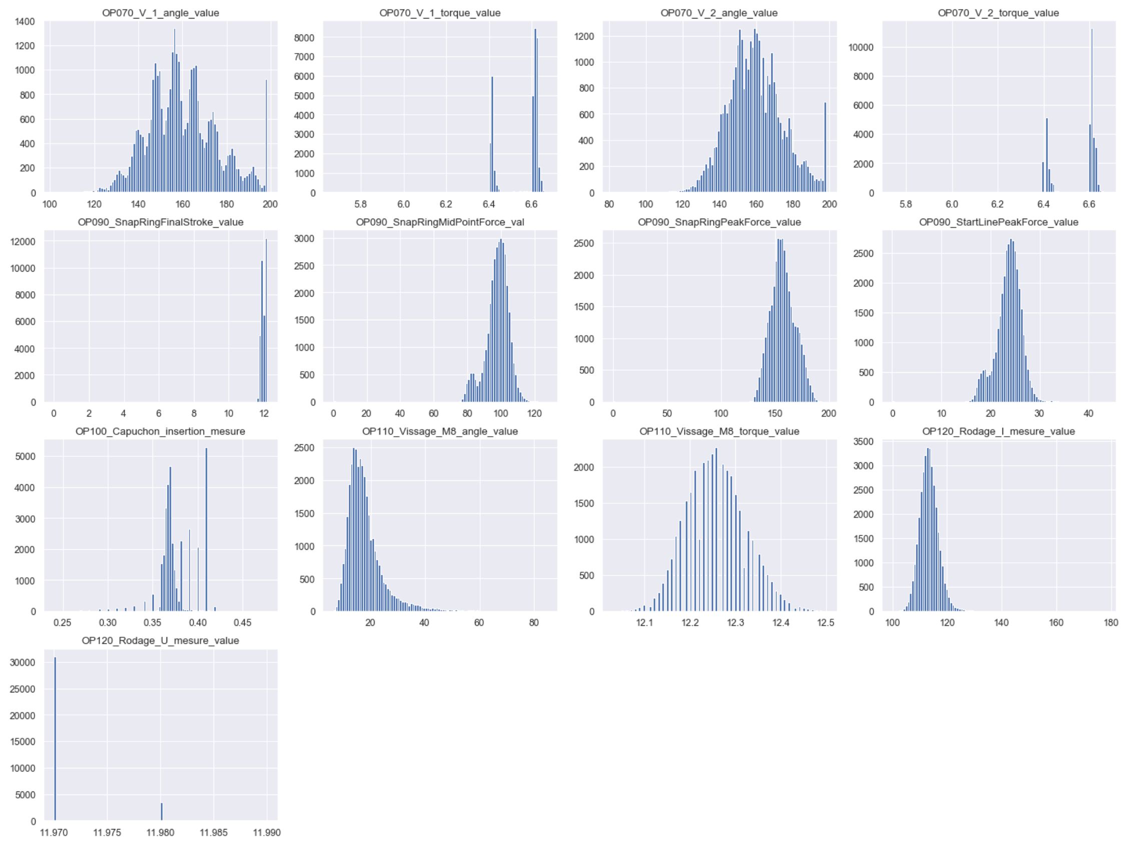
L'étude concerne la classification des données déséquilibrée avec des valeurs de données manquantes. C'est un problème classique dans l'industrie et dans bien d'autres domaines : détection de fraude, détection de spam, domaine médical, ...

On constate qu'on est sur une classification déséquilibrée dans la répartition ce qui pose un défi pour la modélisation prédictive. La classe minoritaire est plus importante et donc le problème est plus sensible aux erreurs de classification pour la classe minoritaire que pour la classe majoritaire.

6 - Exploration graphique univariable des données

a - Histogramme des features après gestion des valeurs manquantes:

```
tsf = transf.Transformer()
X_data_transformed = tsf.iterative_imputer_transform(X_data)
ImgUtil.save_df_hist_plot(X_data_transformed,"X_data_imputed",figsize=(20,15), bins=100)
plt.show()
```



NB:

tsf.iterative_imputer_transform(X_data) méthode de la classe 'Transformer' du module Python valeo.infrastructure.Transformer. Elle applique un imputer du type IterativeImputer(estimator=BayesianRidge, missing_values, initial_strategy = 'median')

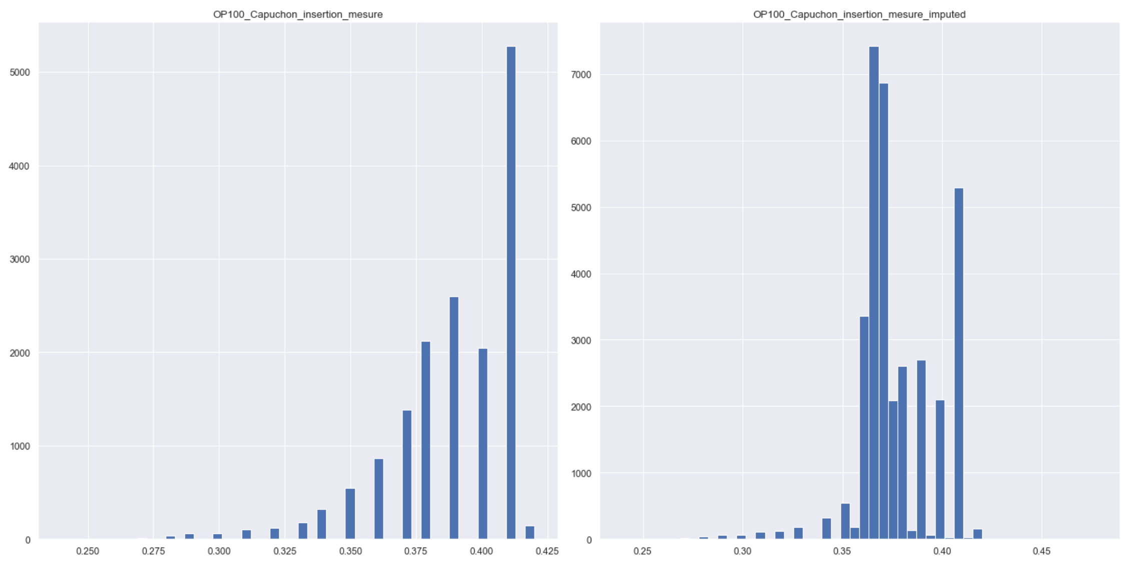
En observant les graphes des différents features, on constate:

- La plupart des distributions sont asymétriques notamment pour: (=> Appliquer une transformation logarithmique dans l'étape F.Engineering)
  - OP070_V_1_angle_value
  - OP090_SnapRingMidPointForce_val
  - OP090_SnapLinePeakForce_value

  - OP100_Capuchon_insertion_mesure # feature dont la moitié des mesures n'existait pas
  - OP110_Vissage_M8_angle_value
- Une valeur de plafonnement (capping value) pour:
  - OP070_V_1_angle_value
  - OP070_V_2_angle_value
- Les distributions suivantes représentes 2 catégories d'observations indépendantes: (=> Représenter chaque feature par 2 catégories dans l'étape F.Engin.):
  - OP070_V_1_torque_value
  - OP070_V_2_torque_value
  - OP120_Rodage_U_mesure_value
  -

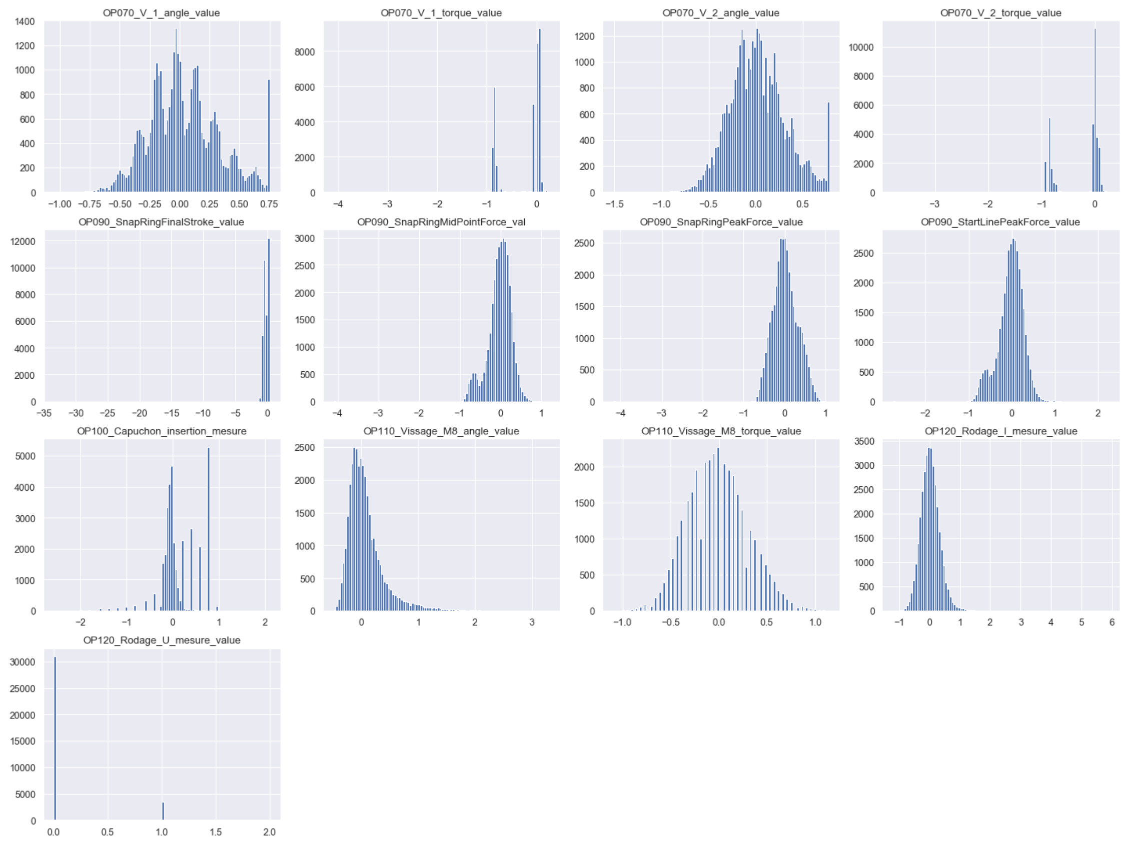
Histogramme comparant la feature 'OP100_Capuchon_insertion_mesure' avant et après la gestions des valeurs manquantes:

```
dff_ = pd.DataFrame(X_data[Const.OP100_Capuchon_insertion_mesure])
dff_[Const.OP100_Capuchon_insertion_mesure + "_imputed"] =
X_data_transformed[Const.OP100_Capuchon_insertion_mesure]
ImgUtil.save_df_hist_plot(dff_,Const.OP100_Capuchon_insertion_mesure, figsize=(20,10))
plt.show()
```



b - Histogramme des features avec gestion des valeurs manquantes + Application de 'RobustScaler':

```
X_data_transformed_scaled = tsf.robust_scaler_transform(X_data_transformed)
ImgUtil.save_df_hist_plot(X_data_transformed_scaled,"X_data_imputed_robust_scaled",bins=100)
plt.show()
```



Pourquoi appliquer un scaling:

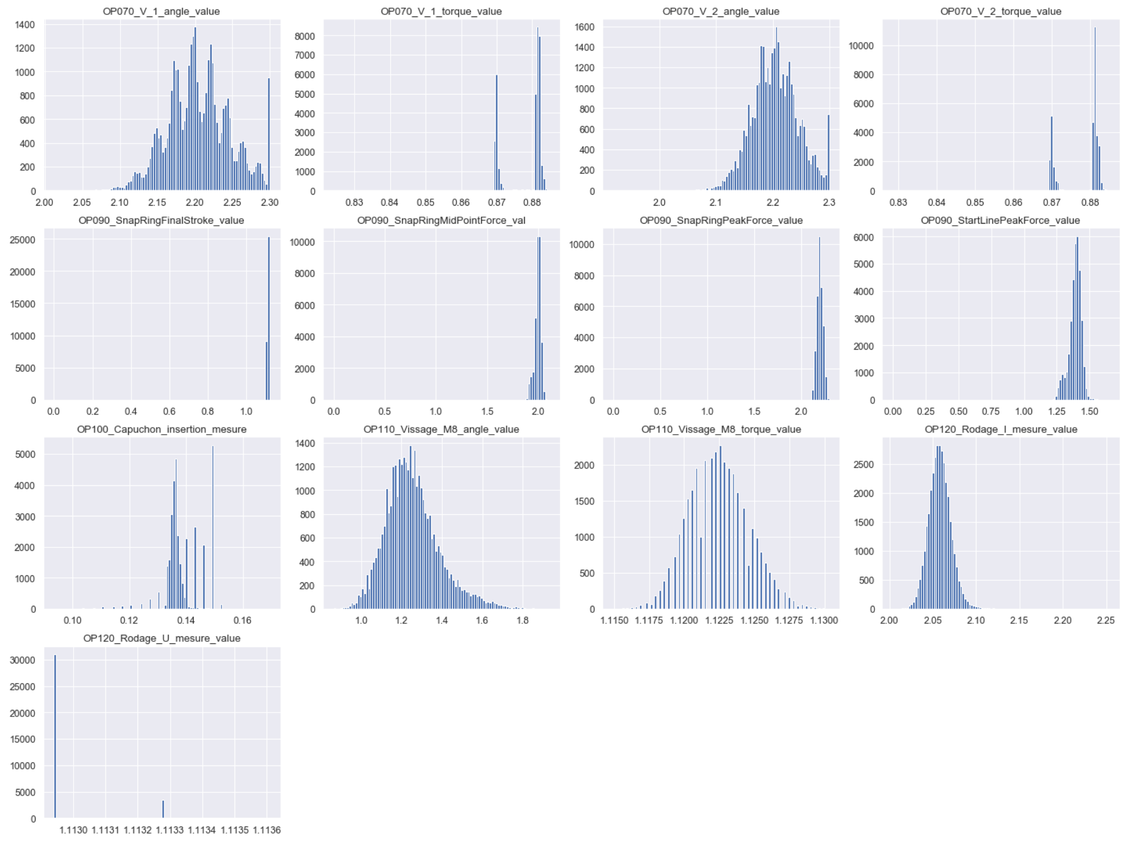
Les algorithmes d'apprentissage automatique prennent en compte uniquement la magnitude des mesures, mais pas les unités de ces mesures. Par la suite, une caractéristique exprimée en une magnitude (nombre) très élevée, peut affecter la prévision beaucoup plus qu'une caractéristique tout aussi importante.

Notez que tous les algorithmes se comportent pas de cette façon et par la suite l'application du scaling n'est pas un pré-requis pour tout les algorithmes.

Les algorithmes à base d'arbres et de Naive Bayes ne nécessitent pas de mise à l'échelle des fonctionnalités, car ils fonctionnent. Les algorithmes qui exploitent des distances ou des similitudes (par exemple sous forme de produit scalaire) entre des échantillons de données, tels que k-NN et SVM, nécessitent souvent une mise à l'échelle des fonctionnalités.

c - Histogramme des features après gestion des valeurs manquantes + Transformation 'log10':

```
tsf = transf.Transformer()
X_data_offset_1 = X_data_transformed + 1
X_data_transformed_log10 = X_data_offset_1.applymap(np.log10)
ImgUtil.save_df_hist_plot(X_data_transformed_log10,"X_data_imputed_log10", bins=100)
plt.show()
```



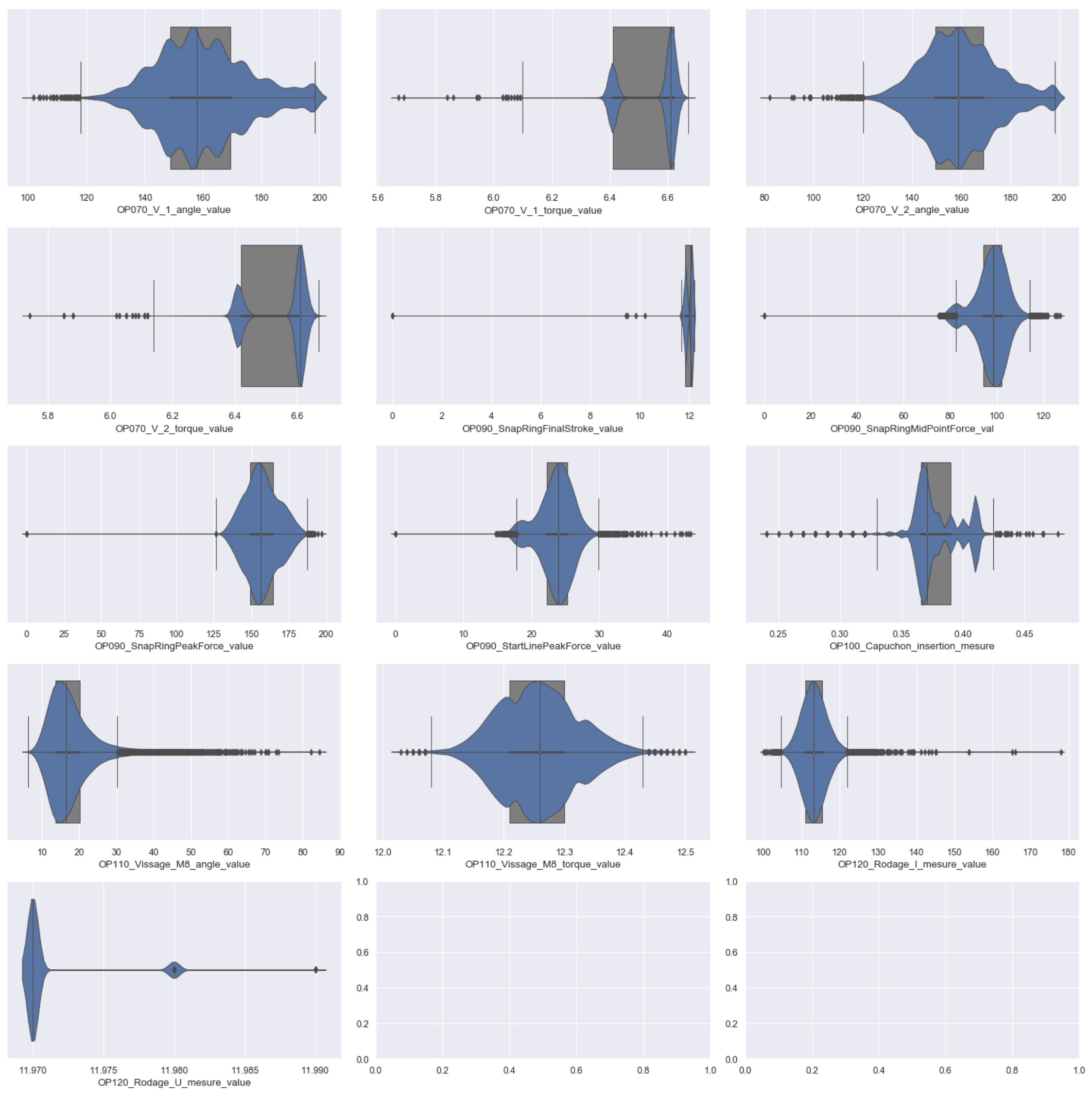
On constate que les distributions ci dessus correspondant aux features transformées par log10 **resemblent plus** à des distributions Normales.

d - 'Violon' et 'boîte à moustaches' des features avec gestion des valeurs manquantes:

Chacun des graphes suivants correspond à la superposition de 2 graphes: Celui d'une 'boîte à moustaches' et d'un 'violon'.

La 'boîte à moustache' représente clairement Q1, Q3, médianne, moustaches, min, max, outliers. Alors que le 'violon' montre bien la distribution des données à l'intérieur.

```
ImgUtil.save_df_violin_plot(X_data_transformed, 'X_data_distribution', 3)
```

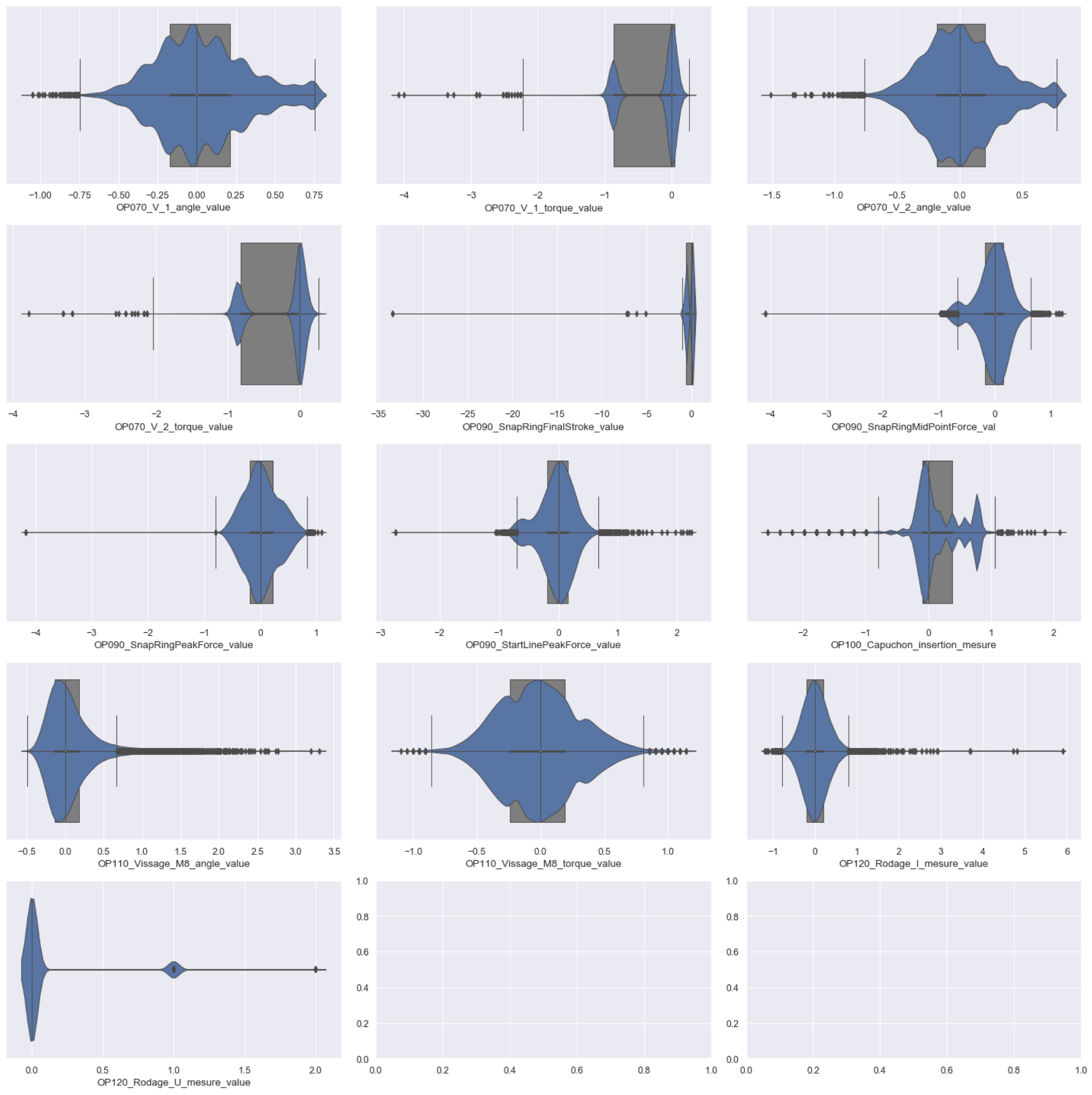


Représenter la distribution d'une feature par un 'violin plot' superposé à un 'box plot' permet de:

- Visualiser la taille de la distribution d'une feature en fonction de sa valeur
- Afficher Q1,Q3 et la médiane
- Afficher les moustaches inférieur et supérieur ainsi que les outliers

e - Violon et boîte à moustaches des features avec gestion des valeurs manquantes + application de 'RobustScaler':

```
ImgUtil.save_df_violin_plot(X_data_transformed_scaled, 'X_data_scaled_distribution_scaled', 3)
```



7 - Exploration graphique bivariables 'feature/target' des données

a - Matrice de correlation (pearson) et heatmap après gestion des valeurs manquantes :

La corrélation des données est un moyen de comprendre la relation entre plusieurs features/target dans un ensemble de données.

```
# 1 - Charger les features et la target en les croisant:
XY_data_with_id = pd.merge(left=data, right=Y_data, how='inner', left_on=Const.PROC_TRACEINFO,
right_on=Const.PROC_TRACEINFO)
XY_data_with_id.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 34515 entries, 0 to 34514
Data columns (total 15 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   PROC_TRACEINFO                   34515 non-null  object
 1   OP070_V_1_angle_value            34515 non-null  float64
 2   OP090_SnapRingPeakForce_value    34515 non-null  float64
 3   OP070_V_2_angle_value            34515 non-null  float64
 4   OP120_Rodage_I_mesure_value      34515 non-null  float64
 5   OP090_SnapRingFinalStroke_value  34515 non-null  float64
 6   OP110_Vissage_M8_torque_value    34515 non-null  float64
 7   OP100_Capuchon_insertion_mesure  15888 non-null  float64
 8   OP120_Rodage_U_mesure_value      34515 non-null  float64
 9   OP070_V_1_torque_value           34515 non-null  float64
 10  OP090_StartLinePeakForce_value   34515 non-null  float64
 11  OP110_Vissage_M8_angle_value     34515 non-null  float64
 12  OP090_SnapRingMidPointForce_val  34515 non-null  float64
 13  OP070_V_2_torque_value           34515 non-null  float64
 14  Binar OP130_Resultat_Global_v    34515 non-null  int64
dtypes: float64(13), int64(1), object(1)
memory usage: 4.2+ MB
```

```
# 2 - Rajout des missing values afin d'avoir une meilleure représentation
XY_data = XY_data_with_id.drop(columns = Const.PROC_TRACEINFO)
XY_data_transformed = tsf.iterative_imputer_transform(XY_data)
```

```
# 3 - Correlation entre la target "Binar OP130_Resultat_Global_v" et les
autres attributs
corr_matrix = XY_data_transformed.corr()
corr_matrix[Const.Binar_OP130_Resultat_Global_v].sort_values(ascending=False)
```

```
Binar OP130_Resultat_Global_v      1.000000
OP100_Capuchon_insertion_mesure    0.040366
OP090_SnapRingFinalStroke_value    0.015148
OP090_SnapRingMidPointForce_val    0.014273
OP090_StartLinePeakForce_value     0.010720
OP110_Vissage_M8_angle_value       0.005470
OP120_Rodage_I_mesure_value        0.003763
```

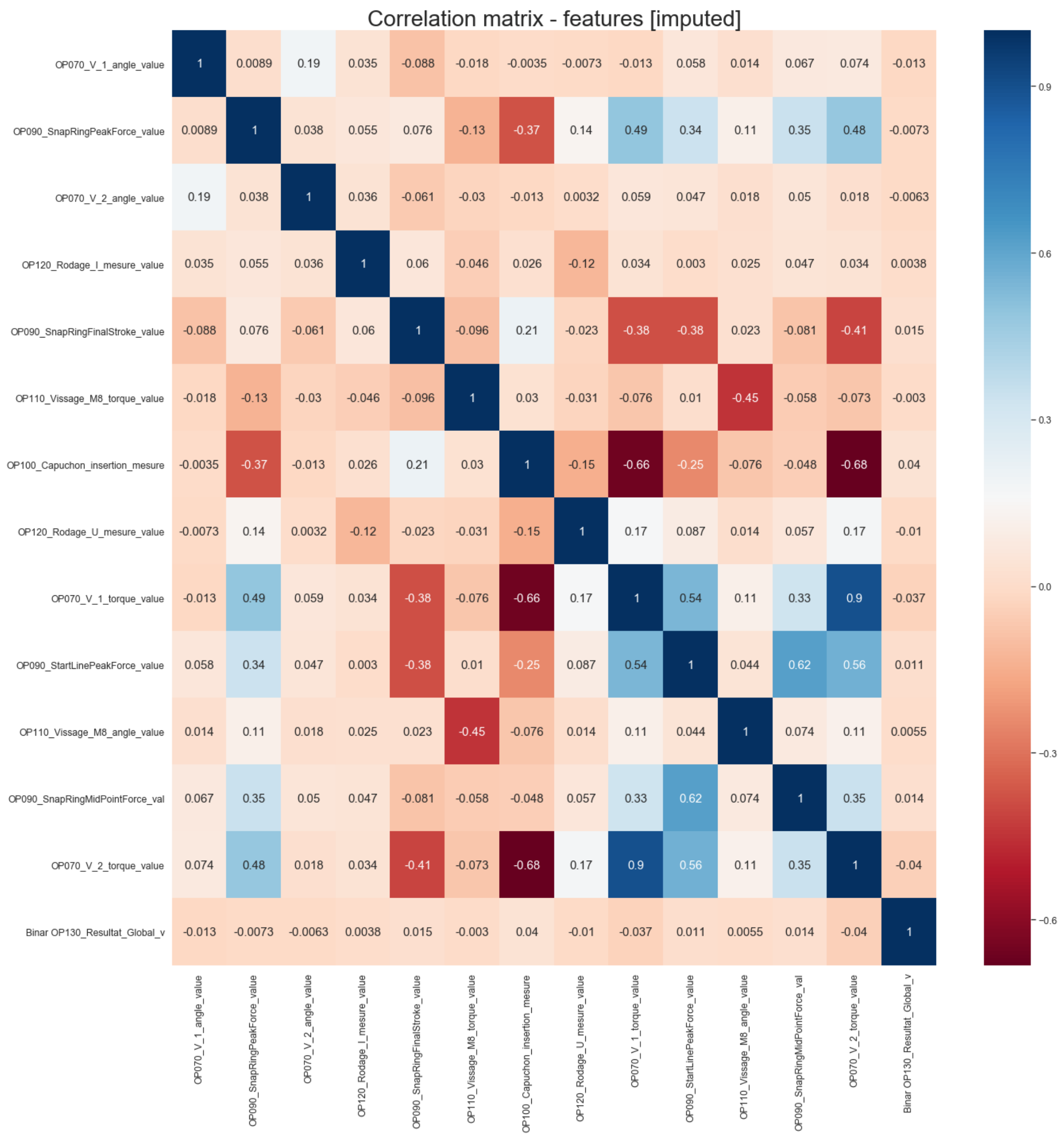
```
OP110_Vissage_M8_torque_value     -0.002984
OP070_V_2_angle_value             -0.006342
OP090_SnapRingPeakForce_value     -0.007290
OP120_Rodage_U_mesure_value       -0.010492
OP070_V_1_angle_value             -0.012793
OP070_V_1_torque_value            -0.037438
OP070_V_2_torque_value            -0.039752
Name: Binar OP130_Resultat_Global_v, dtype: float64
```

Le coefficient de correlation varie entre -1 et 1 :

- Proche de 1 => il y a une correlation forte positive.
- Proche de -1 => il y a une correlation forte négative.
- Proche de 0 => Il n'y a pas de **correlation linéaire**

Le coefficient de correlation mesure uniquement les correlations linéaires

```
# 4 - Dessiner la Heatmap
title = 'Correlation matrix - features [{0}]'
ImgUtil.save_df_heatmap_plot(corr_matrix,title.format('imputed'))
```



En observant la mattrice de correlation, on constate:

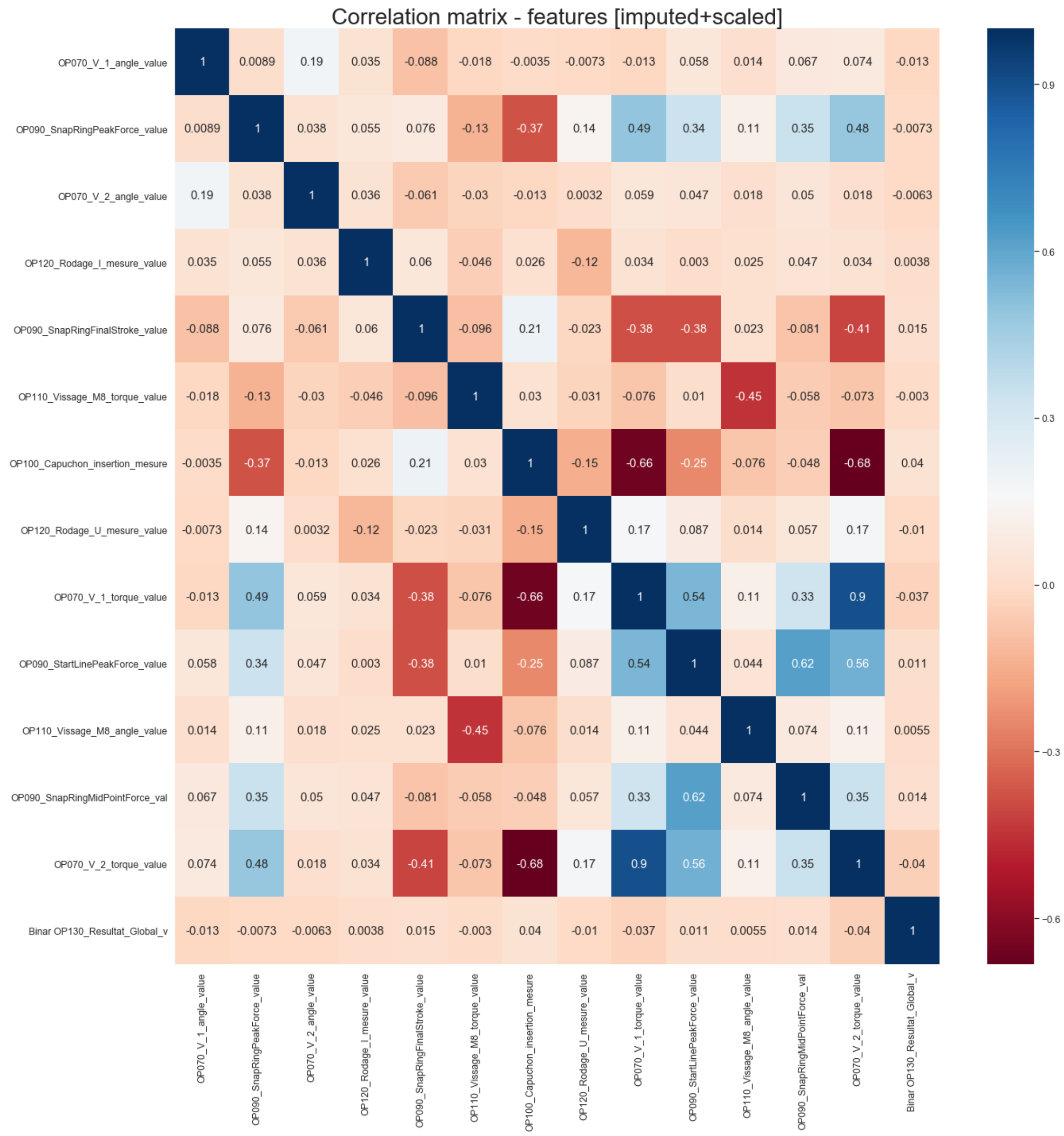
- L'inexistence d'aucune correlation forte entre la target 'Binar OP130_Resultat_Global_v' et n'importe quel feature.
- L'exitence de correlations positives (0.54, 0.49, 0.48, ..) et negatives (-0.68, -0.45, -0.38, ...) parmi les autres features

b - Matrice de correlation et heatmap avec gestion des valeurs manquantes et rescale:

```
# 1 - Appliquer la transformation 'Robust Scaler'
XY_data_transformed_scaled =
tsf.robust_scaler_transform(XY_data_transformed.drop(columns=Const.Binar_OP130_Resultat_Global_v, axis=1))

# 2 - Rajouter la target à la dataframe
XY_data_transformed_scaled[Const.Binar_OP130_Resultat_Global_v] =
XY_data_transformed[Const.Binar_OP130_Resultat_Global_v]

# 3 - Correlation entre la target "Binar OP130_Resultat_Global_v" et les
autres attributs
corr_matrix_scaled = XY_data_transformed_scaled.corr()
corr_matrix_scaled[Const.Binar_OP130_Resultat_Global_v].sort_values(ascending=False)
```

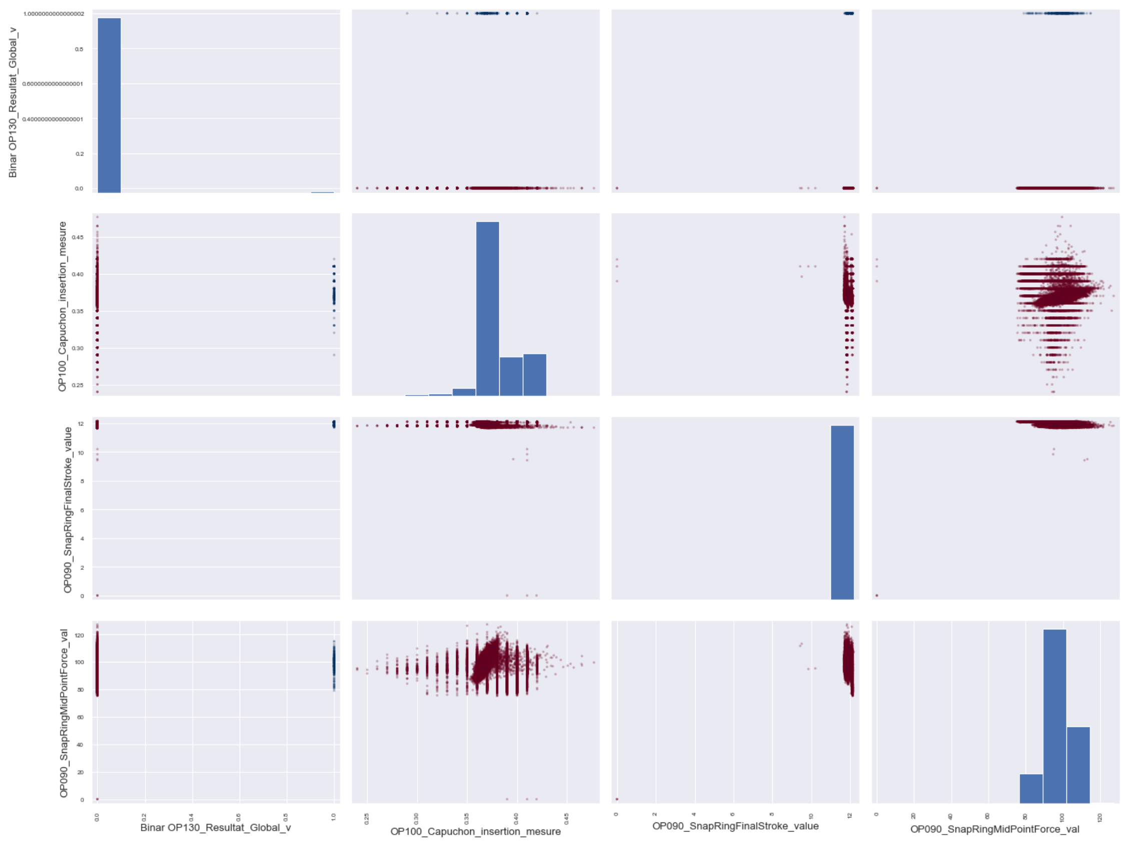
```
Binar OP130_Resultat_Global_v      1.000000
OP100_Capuchon_insertion_mesure    0.040366
OP090_SnapRingFinalStroke_value    0.015148
OP090_SnapRingMidPointForce_val    0.014273
OP090_StartLinePeakForce_value     0.010720
OP110_Vissage_M8_angle_value       0.005470
OP120_Rodage_I_mesure_value        0.003763
OP110_Vissage_M8_torque_value     -0.002984
OP070_V_2_angle_value             -0.006342
OP090_SnapRingPeakForce_value     -0.007290
OP120_Rodage_U_mesure_value       -0.010492
OP070_V_1_angle_value             -0.012793
OP070_V_1_torque_value            -0.037438
OP070_V_2_torque_value            -0.039752
Name: Binar OP130_Resultat_Global_v, dtype: float64
```

```
# 4 - Dessiner la Heatmap
ImgUtil.save_df_heatmap_plot(corr_matrix,title.format('imputed+scaled'))
```



c - Nuage de points entre la target 'Binar OP130_Resultat_Global_v' et les autres features:

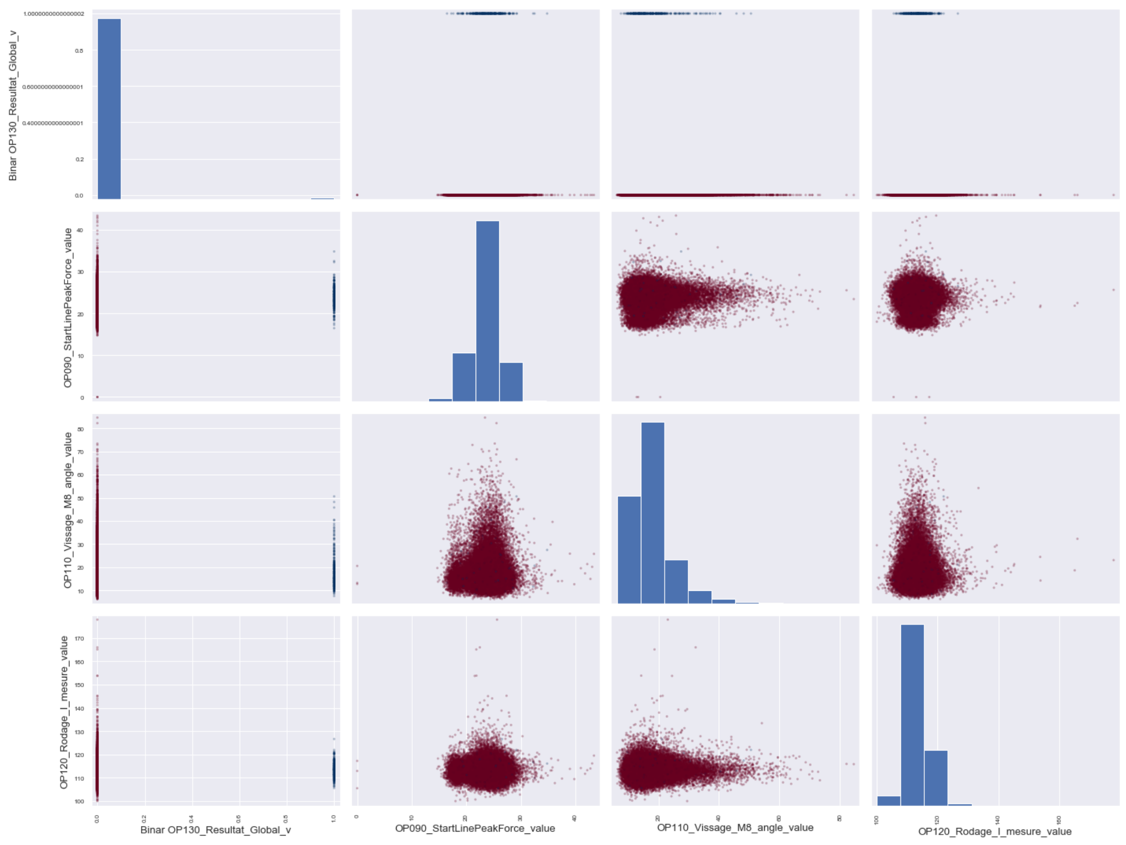
```
features = [Const.Binar_OP130_Resultat_Global_v,
            Const.OP100_Capuchon_insertion_mesure, Const.OP090_SnapRingFinalStroke_value,
Const.OP090_SnapRingMidPointForce_val]
ImgUtil.save_df_scatter_matrix_plot(XY_data_transformed[features], "XY_data_imputed_corr_pos_1",
cfield=Const.Binar_OP130_Resultat_Global_v)
```



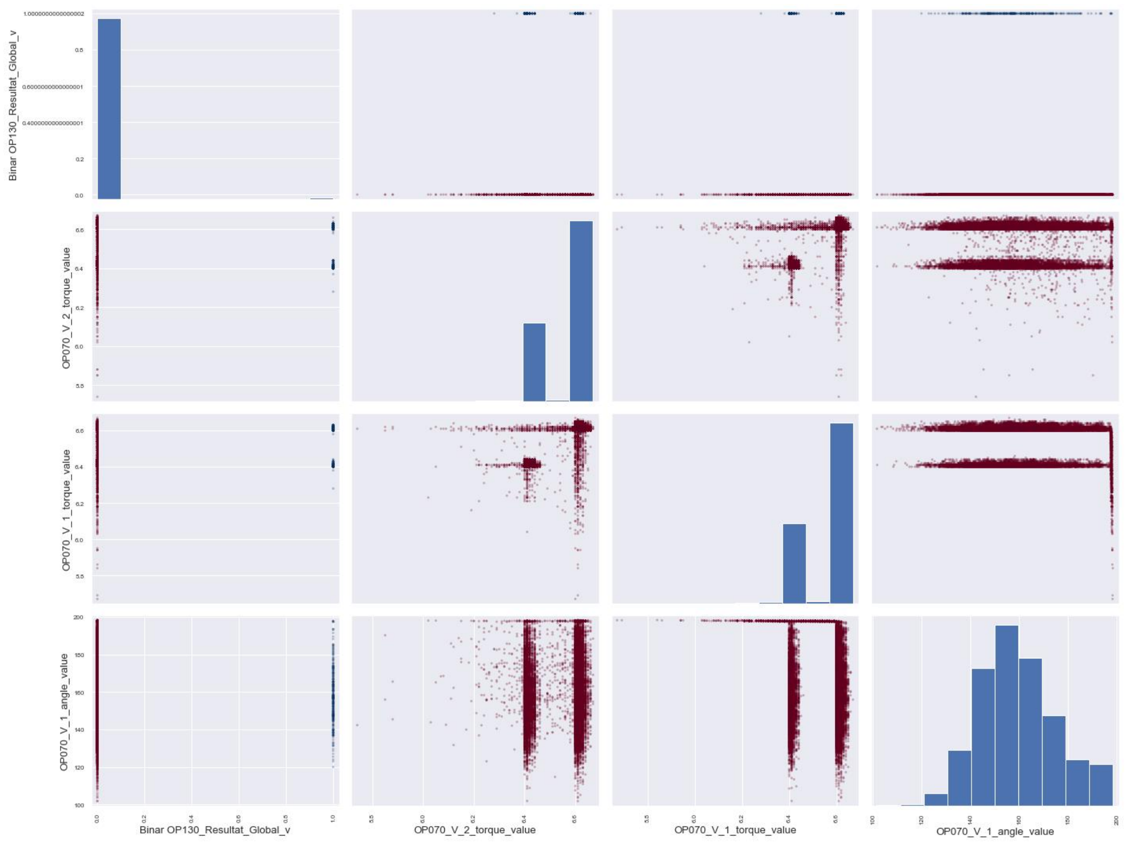
La diagonale allant du coin-gauche-haut au coin-droite-bas représente des barres droites d'histogramme, ces graphes représentent le nombre d'observations d'une feature (ou de la target) en fonction des différentes valeurs que cette feature peut prendre. Le nuage rouge représente les démarreurs étiquetés OK(O) et le bleu représente les KO(1)

NB: On constate que le graphe correspondant à la target (Binar OP130_Resultat_Global_v) représente une distribution fortement déséquilibrée entre les 2 valeurs '0' et '1' que peut prendre la target.

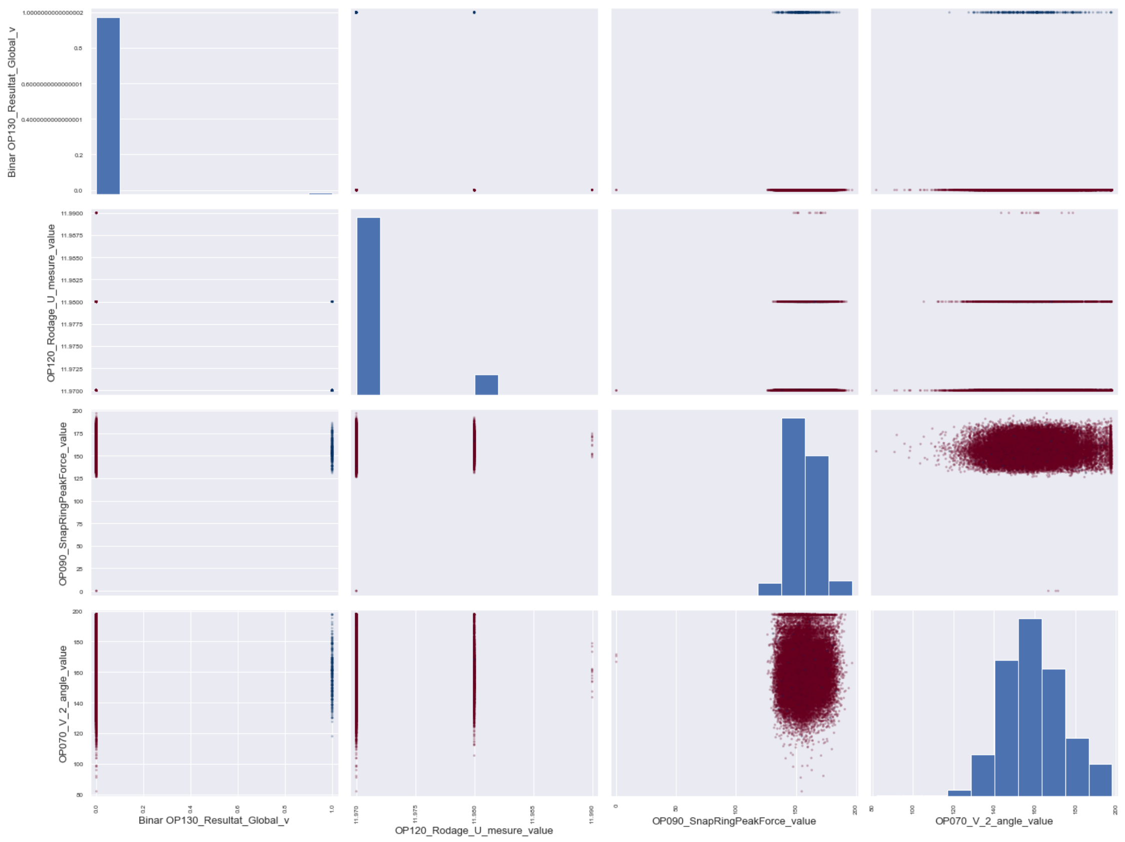
```
features = [Const.Binar_OP130_Resultat_Global_v,
            Const.OP090_StartLinePeakForce_value, Const.OP110_Vissage_M8_angle_value,
Const.OP120_Rodage_I_mesure_value]
ImgUtil.save_df_scatter_matrix_plot(XY_data_transformed[features], "XY_data_imputed_corr_pos_2",
cfield=Const.Binar_OP130_Resultat_Global_v)
```



```
features = [Const.Binar_OP130_Resultat_Global_v,
            Const.OP070_V_2_torque_value, Const.OP070_V_1_torque_value,
Const.OP070_V_1_angle_value]
ImgUtil.save_df_scatter_matrix_plot(XY_data_transformed[features], "XY_data_imputed_neg_1",
cfield=Const.Binar_OP130_Resultat_Global_v)
```



```
features = [Const.Binar_OP130_Resultat_Global_v,
            Const.OP120_Rodage_U_mesure_value, Const.OP090_SnapRingPeakForce_value,
Const.OP070_V_2_angle_value]
ImgUtil.save_df_scatter_matrix_plot(XY_data_transformed[features], "XY_data_imputed_neg_2",
cfield=Const.Binar_OP130_Resultat_Global_v)
```



D'après les graphes, on constate qu'il n'y aucune relation lineaire !!

8 - Feature Engineering/Sélection et choix faits/Hypothèses choisies : TODO

9 - Analyse de la target

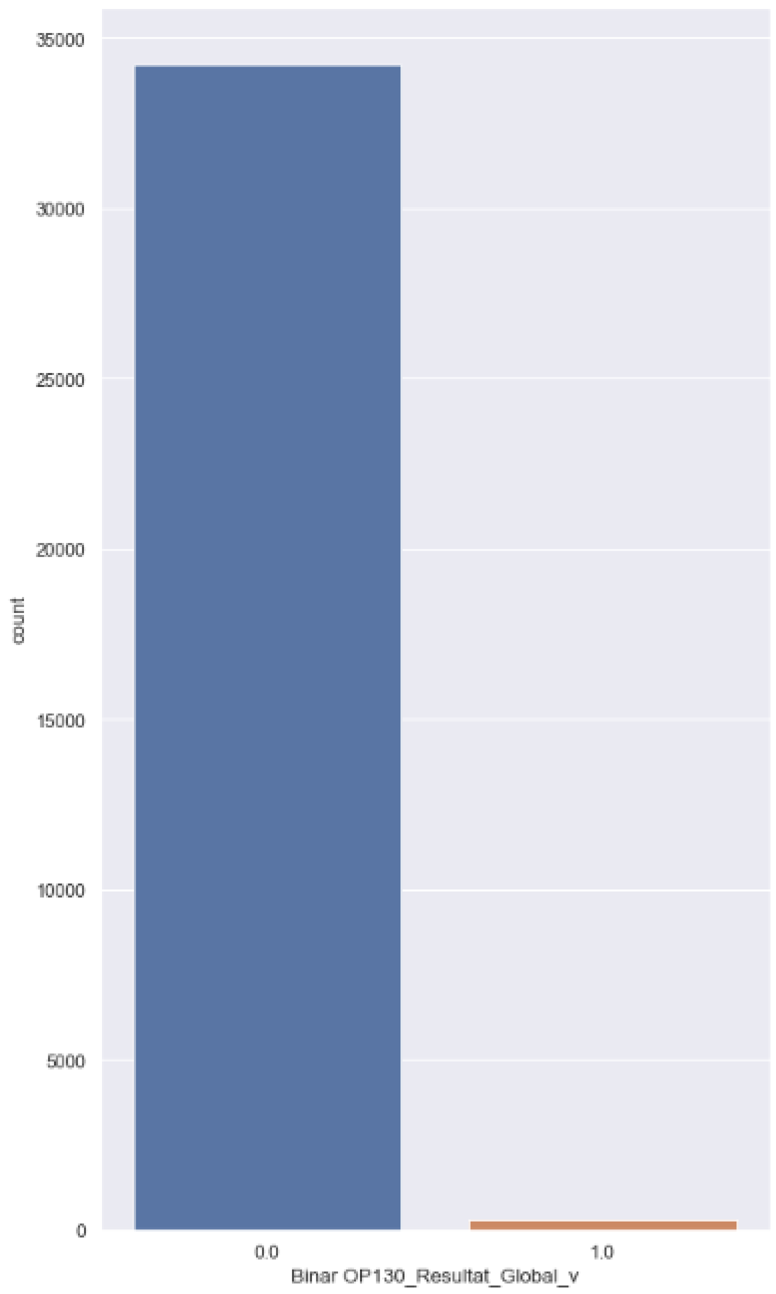
a - Vérification de l'équilibre des données:

```
starter_count = len(Y_data[Const.Binar_OP130_Resultat_Global_v])
starter_count_ok = Y_data[Const.Binar_OP130_Resultat_Global_v].value_counts()[0]
starter_count_ko = Y_data[Const.Binar_OP130_Resultat_Global_v].value_counts()[1]
#
print(f'Nombre total des démarreurs : {starter_count}')
print(f'Nombre total des démarreurs OK => Nombre de Classes Negatives : {starter_count_ok} soit
{round(starter_count_ok/starter_count * 100,2)} % du dataset')
print(f'Nombre total des démarreurs KO => Nombre de Classes Positives : {starter_count_ko} soit
{round(starter_count_ko/starter_count * 100,2)} % du dataset')
```

```
Nombre total des démarreurs : 34515
Nombre total des démarreurs OK => Nombre de Classes Negatives : 34210 soit
99.12 % du dataset
Nombre total des démarreurs KO => Nombre de Classes Positives : 305 soit 0.88
% du dataset
```

b - Distribution du dataset selon les classes de la target:

```
plt.figure(figsize=(8, 15))
sns.countplot(Const.Binar_OP130_Resultat_Global_v, data=XY_data_transformed)
```

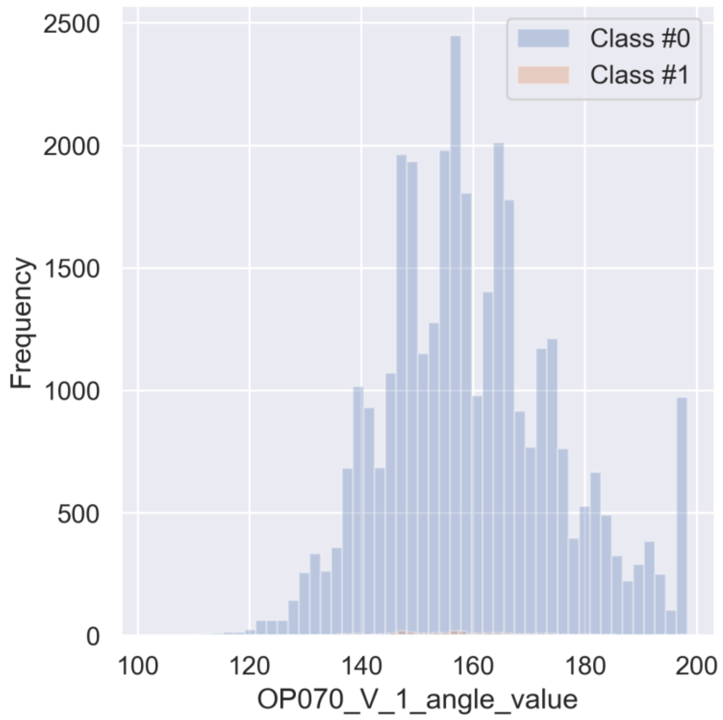


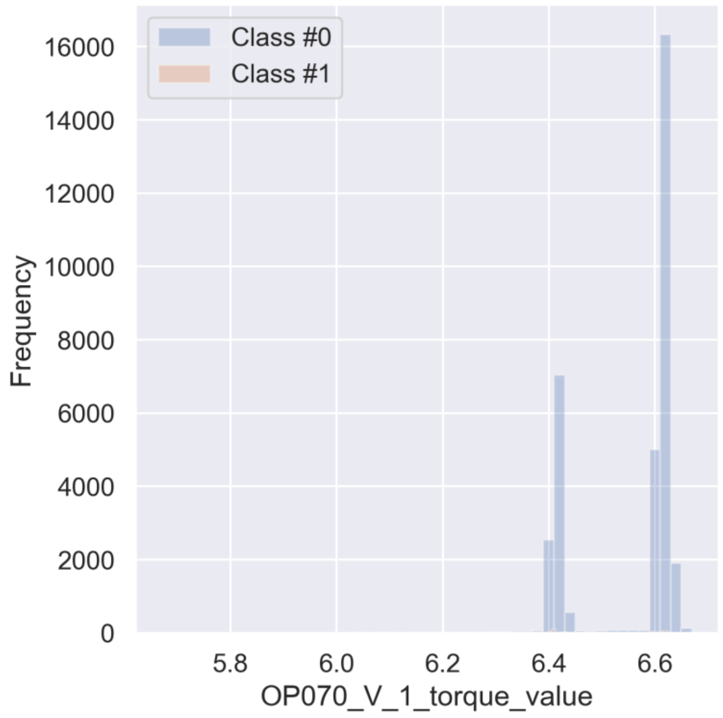
On constate que le jeu de données est fortement déséquilibré.\ Presque totalité des démarreurs (99.12%) ne sont pas défectueux lors de la sortie de la ligne de production.

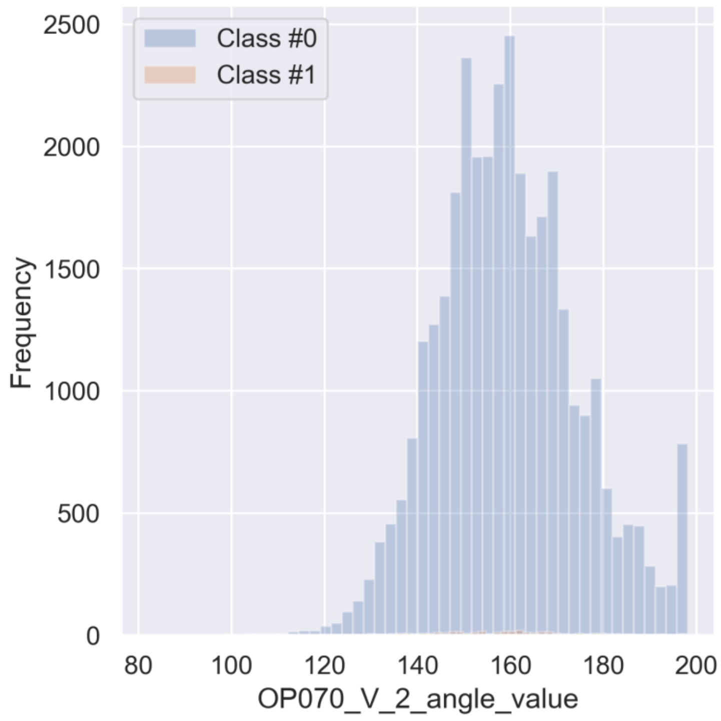
En utilisant cette base de données comme base pour les modèles prédictifs et pour les analyses, on pourrait obtenir beaucoup d'erreurs par des algorithmes inadaptés car ils 'supposeront' que les 'demarreurs' ne sont pas défectueux.\ On cherche un modèle capable de déceler les patterns qui prédisent les défauts sur les lignes de production du démarreur.

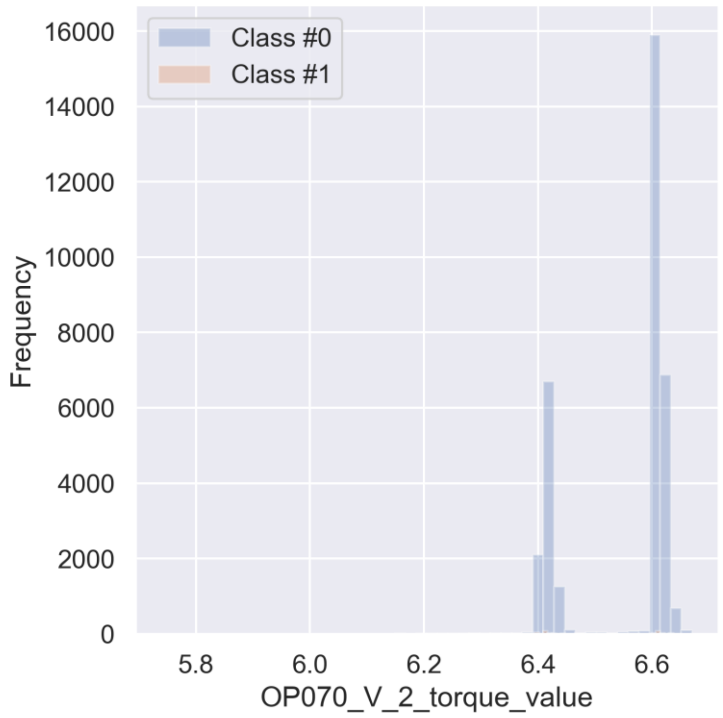
c - Histogramme de distribution du jeu de données selon les classes de la target:

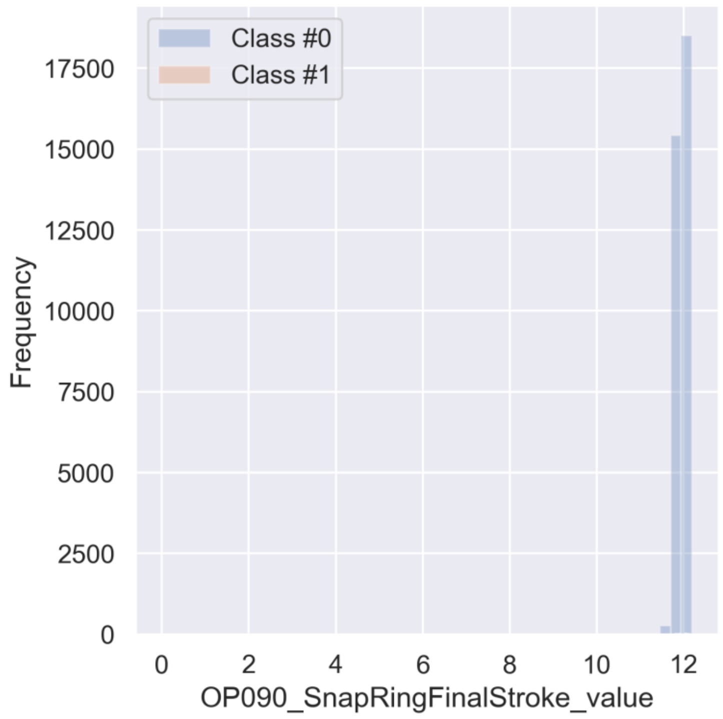
```
ImgUtil.save_df_XY_hist_plot(XY_data_transformed, "XY_imputed",
y_target_name=Const.Binar_OP130_Resultat_Global_v)
```

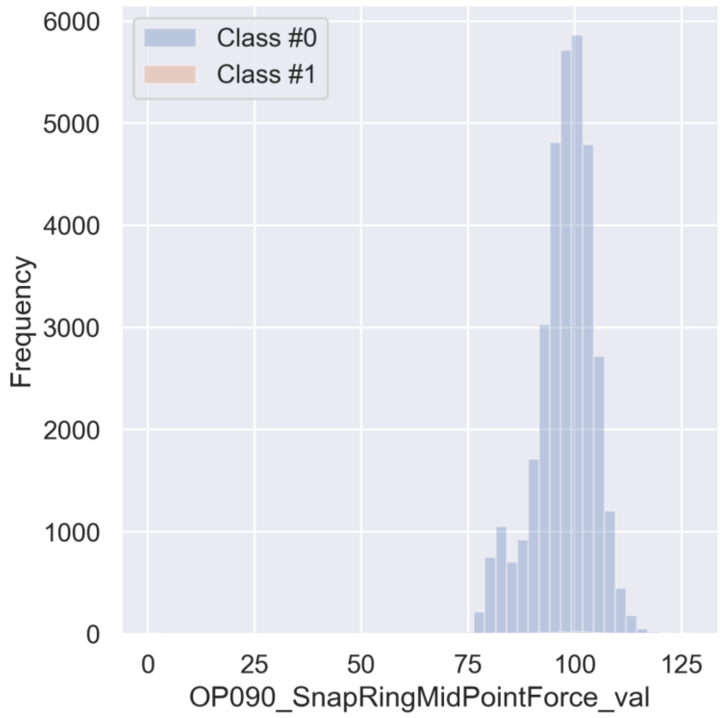


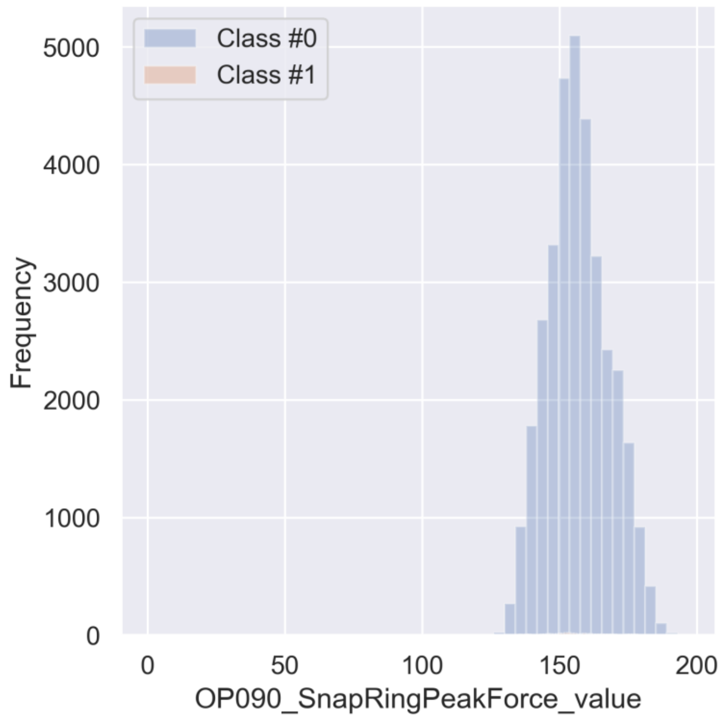


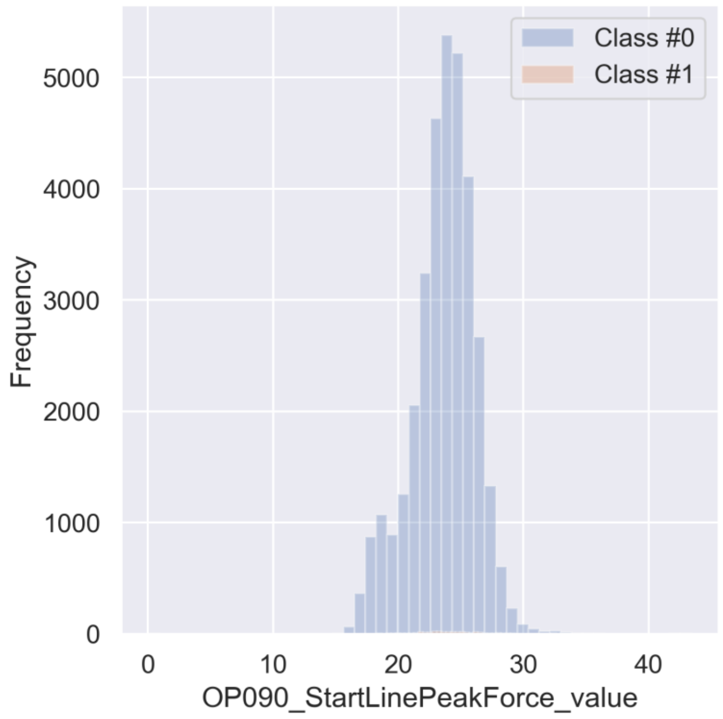

Class #0
Class #1
Frequency
2500
2000
1500
1000
500
0
80
100
120
140
160
180
200
OP070_V_2_angle_value

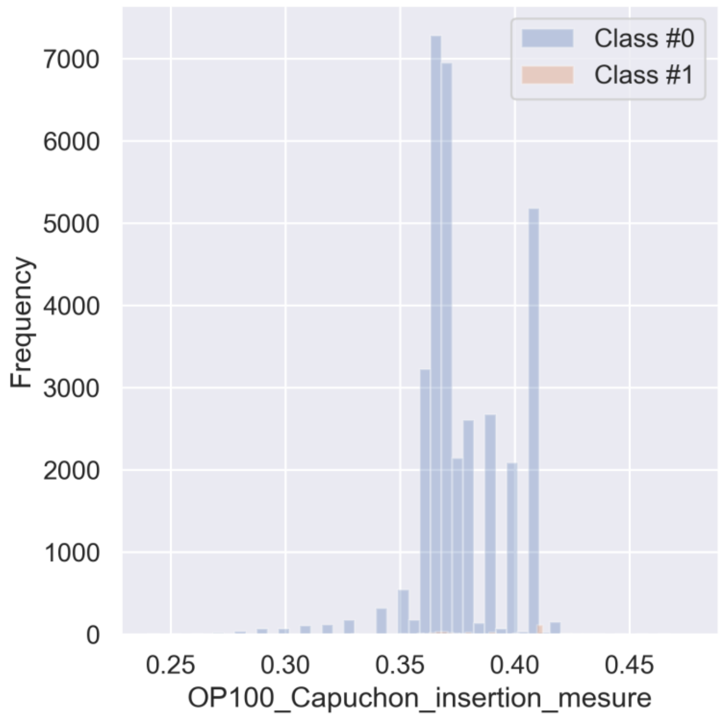

Class #0
Class #1
Frequency
16000
14000
12000
10000
8000
6000
4000
2000
0
5.8
6.0
6.2
6.4
6.6
OP070_V_2_torque_value

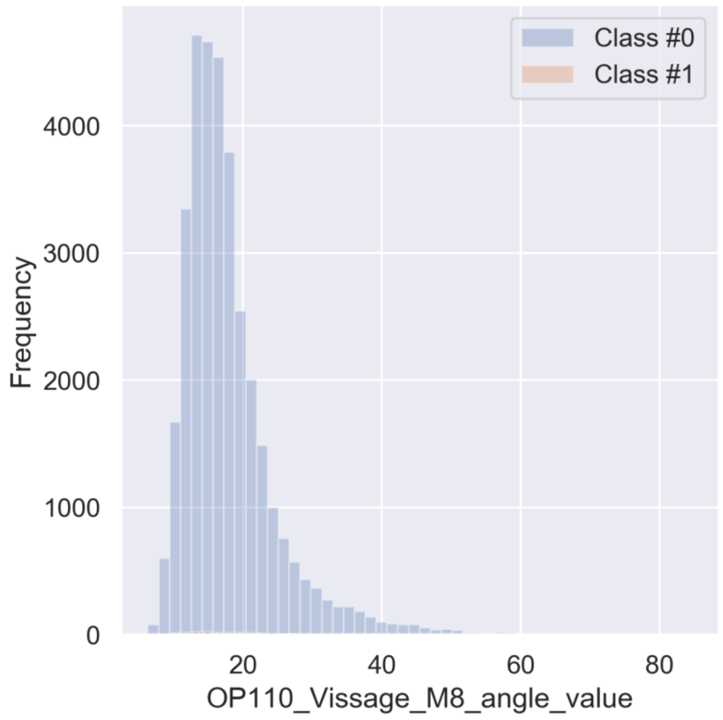

Class #0
Class #1
Frequency
0
2500
5000
7500
10000
12500
15000
17500
0
2
4
6
8
10
12
OP090_SnapRingFinalStroke_value

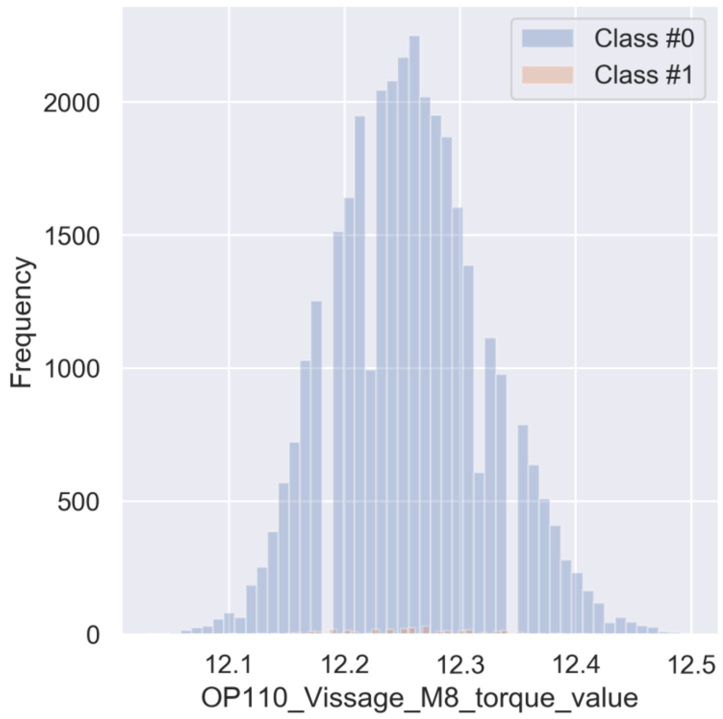

Class #0
Class #1
Frequency
0
1000
2000
3000
4000
5000
6000
0
25
50
75
100
125
OP090_SnapRingMidPointForce_val

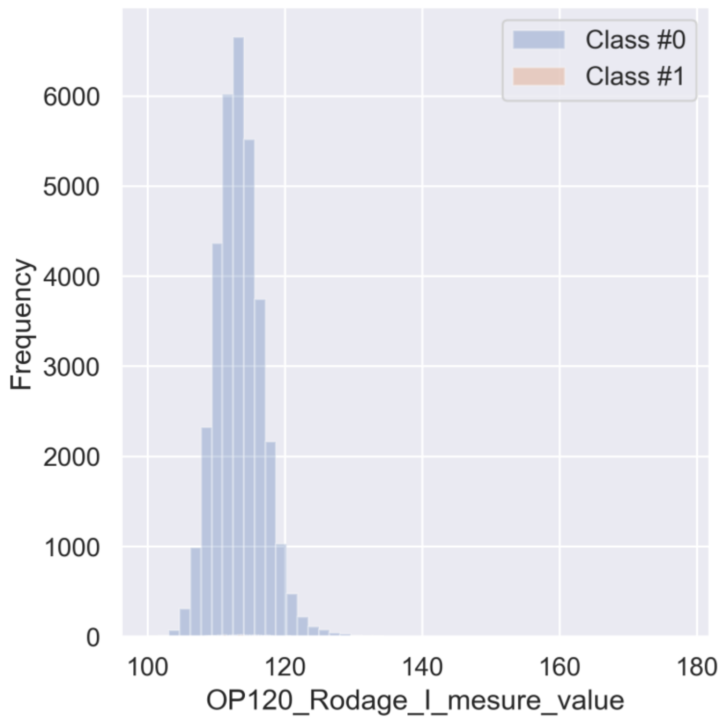

Class #0
Class #1
Frequency
0
1000
2000
3000
4000
5000
0
50
100
150
200
OP090_SnapRingPeakForce_value

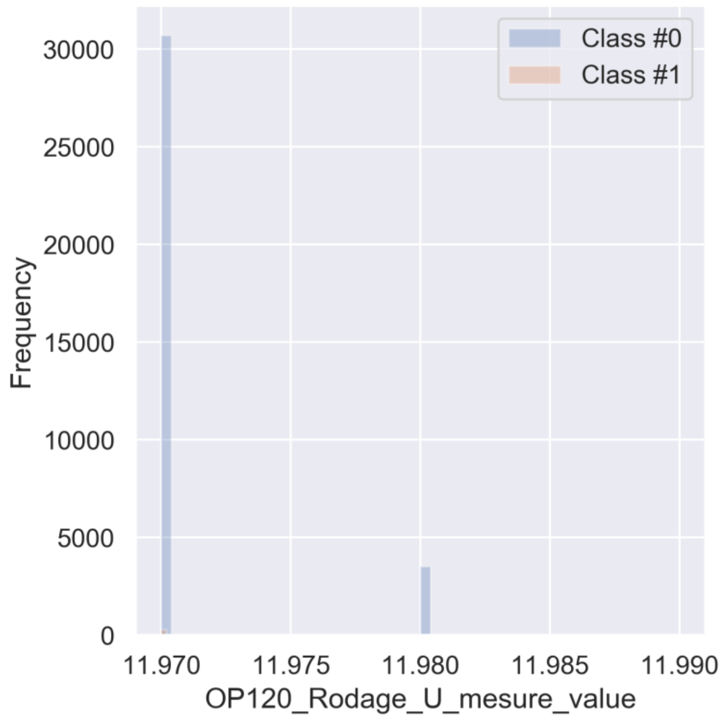

Class #0
Class #1
Frequency
0
1000
2000
3000
4000
5000
0
10
20
30
40
OP090_StartLinePeakForce_value


Class #0
Class #1
Frequency
0
1000
2000
3000
4000
5000
6000
7000
0.25
0.30
0.35
0.40
0.45
OP100_Capuchon_insertion_mesure


Class #0
Class #1
Frequency
0
1000
2000
3000
4000
20
40
60
80
OP110_Vissage_M8_angle_value


Class #0
Class #1
2000
1500
1000
500
0
Frequency
12.1
12.2
12.3
12.4
12.5
OP110_Vissage_M8_torque_value


Class #0
Class #1
6000
5000
4000
3000
2000
1000
0
Frequency
100
120
140
160
180
OP120_Rodage_I_mesure_value

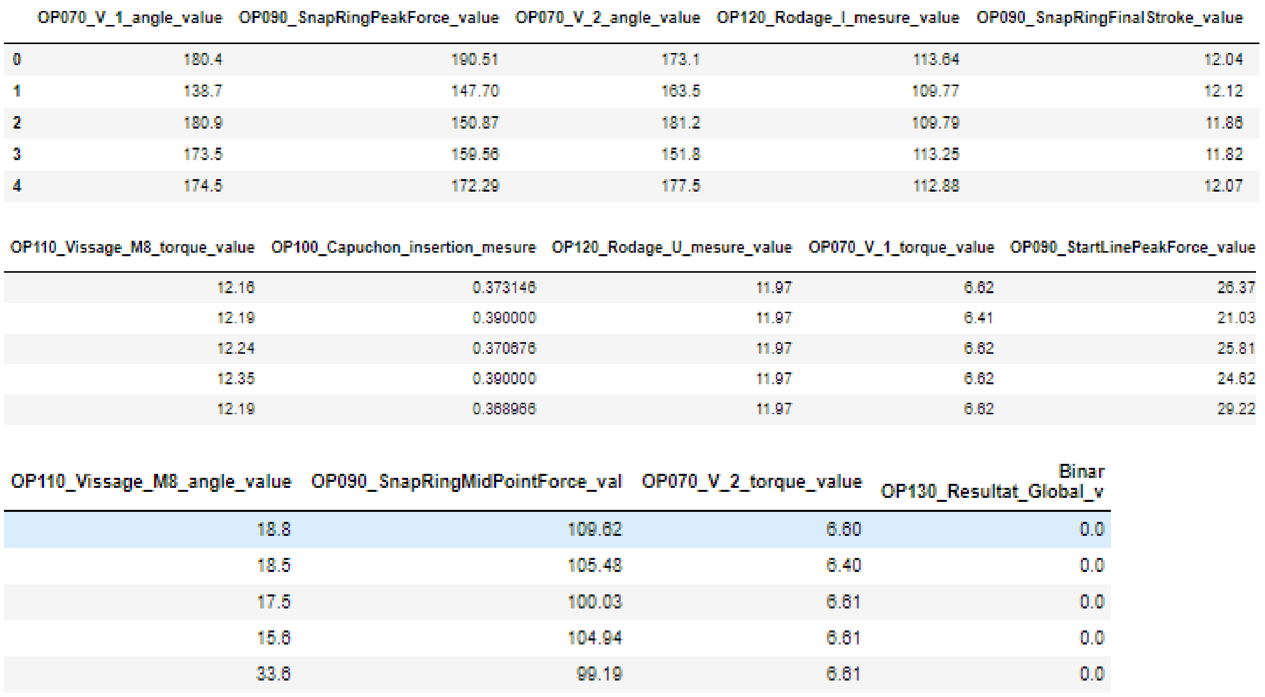


On constate que la classe minoritaire se retrouve délimité à l'intérieur d'une plage de valeurs pour certains features. (ex: OP070_V_1/2_angle_value, OP110_Vissage_M8_torque_Value).
Pour cela, il faudrait vérifier l'impact si on transforme ces features numériques continues en des features catégoriques mettant en avant l'existence de la classe minoritaire KO pour ces catégories.

10 - Analyse de la target après un oversampling SMOTE

a - Regénération SMOTE de la classe minoriataire de la target:

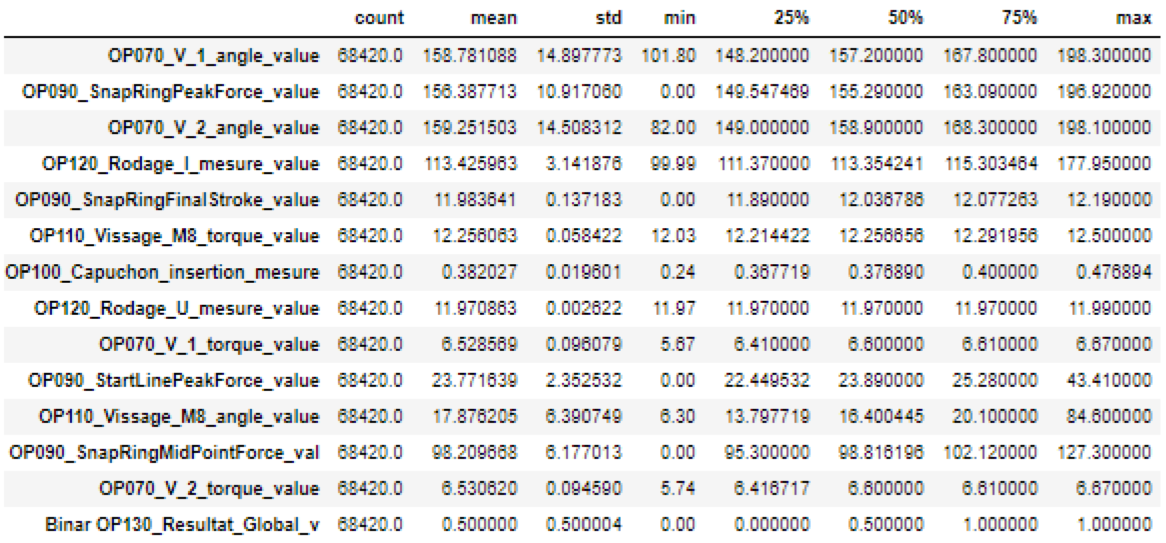
```
sm = SMOTE(sampling_strategy='minority', random_state=7)
#
oversampled_X, oversampled_Y = sm.fit_sample(XY_data_transformed.drop(Const.Binar_OP130_Resultat_Global_v,
axis=1),

XY_data_transformed[Const.Binar_OP130_Resultat_Global_v])
oversampled_XY = pd.concat([pd.DataFrame(oversampled_X), pd.DataFrame(oversampled_Y)], axis=1)
oversampled_XY.columns = XY_data_transformed_scaled.columns

oversampled_XY.head()
```



	OP070_V_1_angle_value	OP090_SnapRingPeakForce_value	OP070_V_2_angle_value	OP120_Rodage_I_mesure_value	OP090_SnapRingFinalStroke_value	OP110_Vissage_M8_torque_value	OP100_Capuchon_insertion_mesure	OP120_Rodage_U_mesure_value	OP070_V_1_torque_value	OP090_StartLinePeakForce_value	OP110_Vissage_M8_angle_value	OP090_SnapRingMidPointForce_val	OP070_V_2_torque_value	Binar OP130_Resultat_Global_v
0	180.4	190.51	173.1	113.64	12.04	12.16	0.373146	11.97	6.62	26.37	18.8	109.62	6.60	0.0
1	138.7	147.70	163.5	109.77	12.12	12.19	0.390000	11.97	6.41	21.03	18.5	105.48	6.40	0.0
2	180.9	150.87	181.2	109.79	11.86	12.24	0.370676	11.97	6.62	25.81	17.5	100.03	6.61	0.0
3	173.5	159.56	151.8	113.25	11.82	12.35	0.390000	11.97	6.62	24.62	15.6	104.94	6.61	0.0
4	174.5	172.29	177.5	112.88	12.07	12.19	0.368966	11.97	6.62	29.22	33.6	99.19	6.61	0.0

b - Statistique descriptive du nouveau dataset:

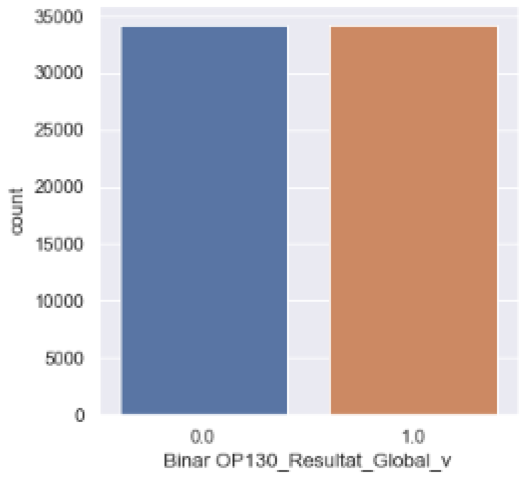
```
oversampled_XY.describe().transpose()
```



	count	mean	std	min	25%	50%	75%	max
OP070_V_1_angle_value	68420.0	158.781088	14.897773	101.80	148.200000	157.200000	167.800000	198.300000
OP090_SnapRingPeakForce_value	68420.0	156.387713	10.917060	0.00	149.547469	155.290000	163.090000	196.920000
OP070_V_2_angle_value	68420.0	159.251503	14.508312	82.00	149.000000	158.900000	168.300000	198.100000
OP120_Rodage_I_mesure_value	68420.0	113.425963	3.141876	99.99	111.370000	113.354241	115.303464	177.950000
OP090_SnapRingFinalStroke_value	68420.0	11.983641	0.137183	0.00	11.890000	12.036786	12.077263	12.190000
OP110_Vissage_M8_torque_value	68420.0	12.256063	0.058422	12.03	12.214422	12.256656	12.291956	12.500000
OP100_Capuchon_insertion_mesure	68420.0	0.382027	0.019601	0.24	0.367719	0.376890	0.400000	0.476894
OP120_Rodage_U_mesure_value	68420.0	11.970863	0.002622	11.97	11.970000	11.970000	11.970000	11.990000
OP070_V_1_torque_value	68420.0	6.528569	0.096079	5.67	6.410000	6.600000	6.610000	6.670000
OP090_StartLinePeakForce_value	68420.0	23.771639	2.352532	0.00	22.449532	23.890000	25.280000	43.410000
OP110_Vissage_M8_angle_value	68420.0	17.876205	6.390749	6.30	13.797719	16.400445	20.100000	84.600000
OP090_SnapRingMidPointForce_val	68420.0	98.209668	6.177013	0.00	95.300000	98.818196	102.120000	127.300000
OP070_V_2_torque_value	68420.0	6.530620	0.094590	5.74	6.416717	6.600000	6.610000	6.670000
Binar OP130_Resultat_Global_v	68420.0	0.500000	0.500004	0.00	0.000000	0.500000	1.000000	1.000000

c - Nouvelle distribution équilibrée du nouveau dataset:

```
plt.figure(figsize=(5, 5))
sns.countplot(Const.Binar_OP130_Resultat_Global_v, data=oversampled_XY)
```

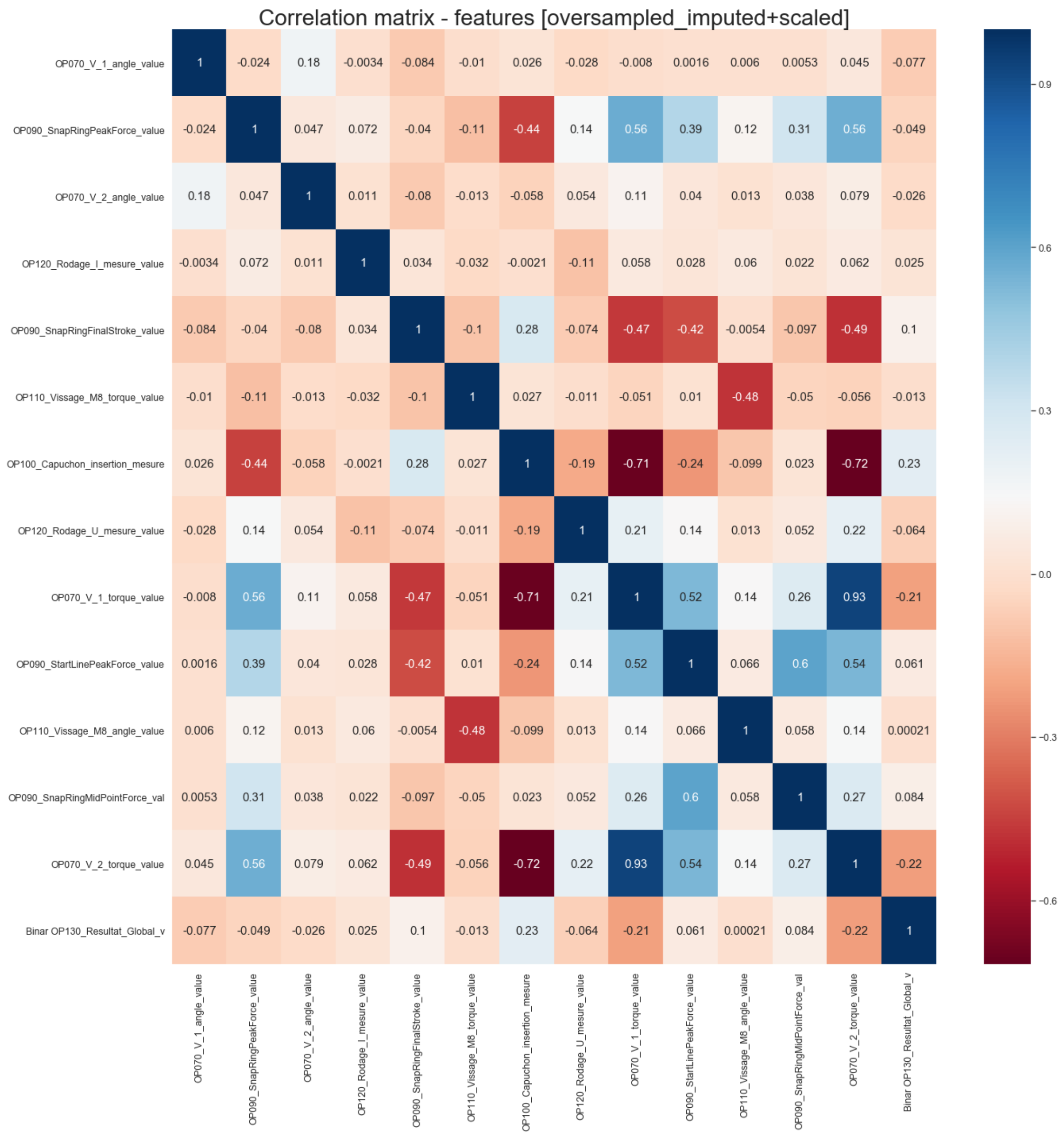


d - Matrice de correlation et heatmap du nouveau dataset:

```
# 3 - Correlation entre la target "Binar OP130_Resultat_Global_v" et les
autres attributs
corr_matrix_oversampled = oversampled_XY.corr()
corr_matrix_oversampled[Const.Binar_OP130_Resultat_Global_v].sort_values(ascending=False)
```

```
Binar OP130_Resultat_Global_v      1.000000
OP100_Capuchon_insertion_mesure    0.229819
OP090_SnapRingFinalStroke_value    0.099822
```

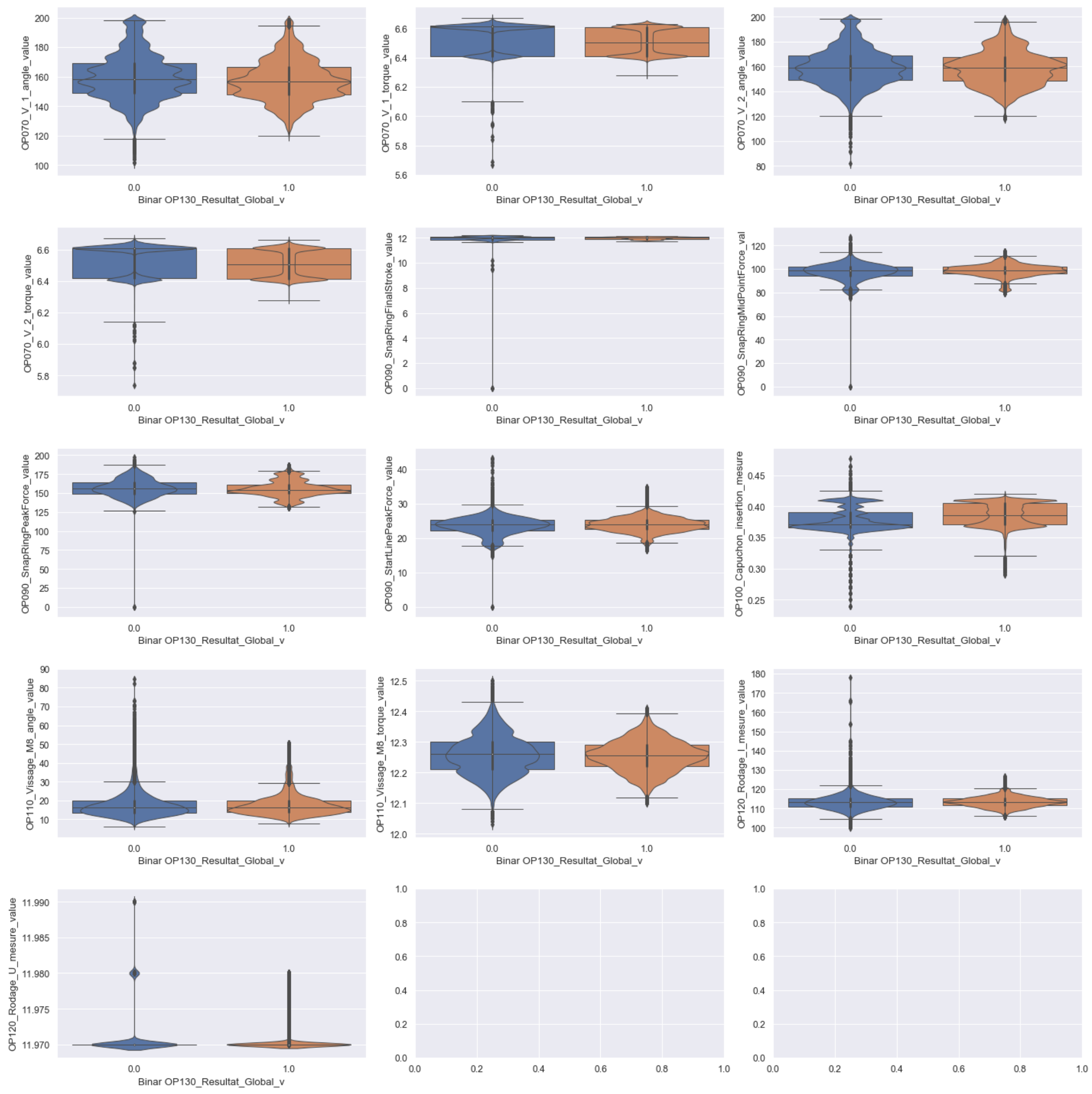
```
OP090_SnapRingMidPointForce_val    0.083845
OP090_StartLinePeakForce_value     0.061239
OP120_Rodage_I_mesure_value        0.024506
OP110_Vissage_M8_angle_value       0.000205
OP110_Vissage_M8_torque_value     -0.012686
OP070_V_2_angle_value             -0.025901
OP090_SnapRingPeakForce_value     -0.049015
OP120_Rodage_U_mesure_value       -0.064017
OP070_V_1_angle_value             -0.076841
OP070_V_1_torque_value            -0.210033
OP070_V_2_torque_value            -0.217814
Name: Binar OP130_Resultat_Global_v, dtype: float64

# 4 - Dessiner la Heatmap
ImgUtil.save_df_heatmap_plot(corr_matrix_oversampled,title.format('oversampled_imputed+scaled'))
```



	OP070_V_1_angle_value	OP090_SnapRingPeakForce_value	OP070_V_2_angle_value	OP120_Rodage_I_mesure_value	OP090_SnapRingFinalStroke_value	OP110_Vissage_M8_torque_value	OP100_Capuchon_insertion_mesure	OP120_Rodage_U_mesure_value	OP070_V_1_torque_value	OP090_StartLinePeakForce_value	OP110_Vissage_M8_angle_value	OP090_SnapRingMidPointForce_val	OP070_V_2_torque_value	Binar OP130_Resultat_Global_v
OP070_V_1_angle_value	1	-0.024	0.18	-0.0034	-0.084	-0.01	0.026	-0.028	-0.008	0.0016	0.006	0.0053	0.045	-0.077
OP090_SnapRingPeakForce_value	-0.024	1	0.047	0.072	-0.04	-0.11	-0.44	0.14	0.56	0.39	0.12	0.31	0.56	-0.049
OP070_V_2_angle_value	0.18	0.047	1	0.011	-0.08	-0.013	-0.058	0.054	0.11	0.04	0.013	0.038	0.079	-0.026
OP120_Rodage_I_mesure_value	-0.0034	0.072	0.011	1	0.034	-0.032	-0.0021	-0.11	0.058	0.028	0.06	0.022	0.062	0.025
OP090_SnapRingFinalStroke_value	-0.084	-0.04	-0.08	0.034	1	-0.1	0.28	-0.074	-0.47	-0.42	-0.0054	-0.097	-0.49	0.1
OP110_Vissage_M8_torque_value	-0.01	-0.11	-0.013	-0.032	-0.1	1	0.027	-0.011	-0.051	0.01	-0.48	-0.05	-0.056	-0.013
OP100_Capuchon_insertion_mesure	0.026	-0.44	-0.058	-0.0021	0.28	0.027	1	-0.19	-0.71	-0.24	-0.099	0.023	-0.72	0.23
OP120_Rodage_U_mesure_value	-0.028	0.14	0.054	-0.11	-0.074	-0.011	-0.19	1	0.21	0.14	0.013	0.052	0.22	-0.064
OP070_V_1_torque_value	-0.008	0.56	0.11	0.058	-0.47	-0.051	-0.71	0.21	1	0.52	0.14	0.26	0.93	-0.21
OP090_StartLinePeakForce_value	0.0016	0.39	0.04	0.028	-0.42	0.01	-0.24	0.14	0.52	1	0.066	0.6	0.54	0.061
OP110_Vissage_M8_angle_value	0.006	0.12	0.013	0.06	-0.0054	-0.48	-0.099	0.013	0.14	0.066	1	0.058	0.14	0.00021
OP090_SnapRingMidPointForce_val	0.0053	0.31	0.038	0.022	-0.097	-0.05	0.023	0.052	0.26	0.6	0.058	1	0.27	0.084
OP070_V_2_torque_value	0.045	0.56	0.079	0.062	-0.49	-0.056	-0.72	0.22	0.93	0.54	0.14	0.27	1	-0.22
Binar OP130_Resultat_Global_v	-0.077	-0.049	-0.026	0.025	0.1	-0.013	0.23	-0.064	-0.21	0.061	0.00021	0.084	-0.22	1

e - Violon et boîte à moustaches des features du nouveau dataset:

```
ImgUtil.save_df_XY_violin_plot(oversampled_XY, Const.Binar_OP130_Resultat_Global_v,
'XY_oversampled_data_distribution', 3)
```



f - Ratio d'observations ayant des features en outlier du nouveau dataset:

```
Q1 = oversampled_X.quantile(0.25)
Q3 = oversampled_X.quantile(0.75)
IQR = Q3 - Q1
#
outliers = ((oversampled_X < (Q1 - 1.5 * IQR)) |(oversampled_X > (Q3 + 1.5 * IQR))).any(axis=1)
print(f"Le ratio d'outlier est de {len(oversampled_X[outliers].index)/len(oversampled_X.index)}")
```

```
Le ratio d'outlier est de 0.2762496346097632
```

Le nombre d'outlier est considérable, à peu près 25% des données => On ne peut pas supprimer les observations correspondantes.

11 - Modèle à base d'arbre : Balanced Random Forest Classifier:

Définissons un ensemble de clés de classifieur afin d'y acceder plus facilement

```
HGBC = HistGradientBoostingClassifier(max_iter = 100 , max_depth=10,learning_rate=0.10,
l2_regularization=5)
BBC  = BalancedBaggingClassifier(base_estimator=HistGradientBoostingClassifier(),  n_estimators=300,
sampling_strategy='auto', replacement=False, random_state=48)
BRFC = BalancedRandomForestClassifier(n_estimators = 300 , max_depth=20, random_state=0)
BRFC_ = BalancedRandomForestClassifier(n_estimators = 300 , max_depth=20, random_state=0,
replacement=True)

BRFC_W = BalancedRandomForestClassifier(n_estimators = 300 , max_depth=20, random_state=0,
class_weight={0:1, 1:1})
RUSBoost = RUSBoostClassifier(n_estimators = 8 , algorithm='SAMME.R', random_state=42)
XGBC = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                              colsample_bynode=1, colsample_bytree=1, gamma=0,
                              learning_rate=0.1, max_delta_step=0, max_depth=10, #max_depth=3,
                              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
                              nthread=None, objective='binary:logistic', random_state=0,
                              reg_alpha=0, reg_lambda=1, scale_pos_weight=100, seed=42,
                              silent=None, subsample=1, verbosity=1)
KNN = KNeighborsClassifier(3)
RFC =  RandomForestClassifier(n_estimators=300, max_depth=10, max_features=10, n_jobs=4, class_weight=
{0:1,1:100})
DTC = DecisionTreeClassifier()
ADABoost = AdaBoostClassifier()
GBC  = GradientBoostingClassifier()
LRC  = LogisticRegression(max_iter=500)
# SVC = SVC(kernel="rbf", C=0.025, probability=True)
# GNB   = GaussianNB()
# NuSVC = NuSVC(probability=True),
# LinearSVC = LinearSVC(C=0.1, class_weight={'1':100})
# SGDClassifier = SGDClassifier(class_weight='balanced')
```

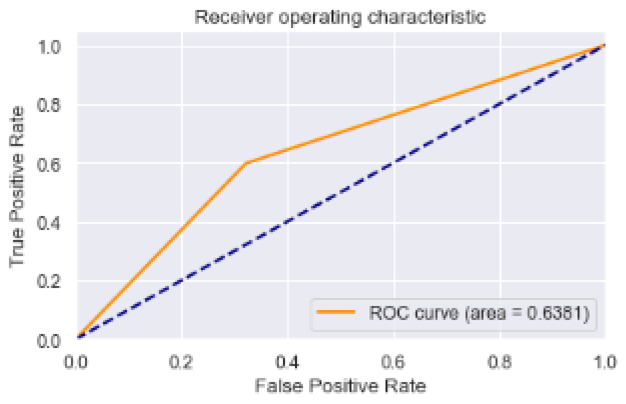
Chargement du jeu de données training - Commun à tout les modèles

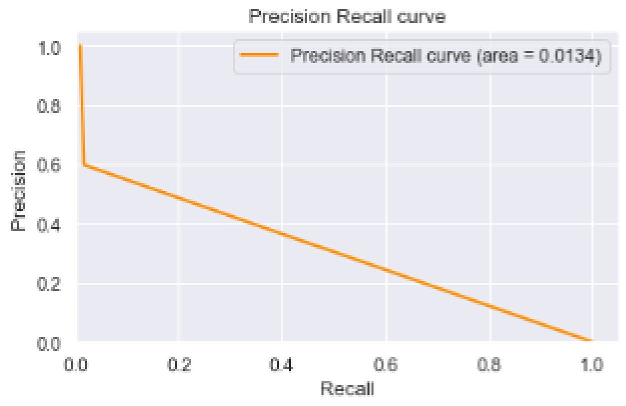
```
# 1 - Rechargement des données
mt_train = XY_metadata([Const.rootDataTrain(), 'traininginputs.csv'], [Const.rootDataTrain(),
'trainingoutput.csv'],
                       [Const.PROC_TRACEINFO], [Const.PROC_TRACEINFO],Const.Binar_OP130_Resultat_Global_v)
xy_loader = XY_Loader();
X_df, y_df = xy_loader.load_XY_df(mt_train)
```

a – Balanced Random Forest Classifier - Train / Test / Split + F1 et ROC:

```
# 2 - Split Training et Validation
X_train, X_test, y_train, y_test = train_test_split(X_df, y_df, test_size=0.3, random_state=48,
stratify=y_df)

# 3 - Imputer et Scaler + classifier
modeler = ValeoModeler()
pred = ValeoPredictor()
pl = Pipeline([('preprocessor', modeler.build_transformers_pipeline(X_train.dtypes)), # --> Imputer +
Scaler
                ('classifier', BRFC)  # --> Balanced Random Forest Classifier
              ])

# 4 - Fit, train, predict and plot ROC and F1
pl.fit(X_train, y_train)
pred.predict_and_plot(pl,X_test, y_test)

# 5 - Test using ENS data
X_ens = DfUtil.read_csv([Const.rootDataTest() , "testinputs.csv"])
y_ens = pl.predict(X_ens.drop(columns=[Const.PROC_TRACEINFO]))
DfUtil.write_y_csv(X_ens[Const.PROC_TRACEINFO], y_ens, Const.Binar_OP130_Resultat_Global_v,
[Const.rootDataTest() , "testoutput.csv"])
```

```
- Model score: 0.6776436504104297
- Accuracy score: 0.6776436504104297
- Balanced accuracy score: 0.6380926205999602 / The balanced accuracy to deal
with imbalanced datasets. It is defined as the average of recall obtained on
each class.
- Average_precision_score: 0.013370660450719168
- Precision_score: 0.016388557806912993
- Recall score: 0.5978260869565217
- Roc_auc_score: 0.6380926205999602
- F1 score: 0.031902552204176336
- [6962 3301]/[37 55] - P:0.0164 - R:0.5978 - roc_auc:0.6381 - f1:0.0319
- [[6962 3301]
 [  37   55]]
- Classification_report_imbalanced:
                   pre       rec       spe        f1       geo       iba
sup

          0       0.99      0.68      0.60      0.81      0.64      0.41
10263
          1       0.02      0.60      0.68      0.03      0.64      0.40
92

avg / total       0.99      0.68      0.60      0.80      0.64      0.41
10355

- Classification_report:
              precision    recall  f1-score   support

           0       0.99      0.68      0.81     10263
           1       0.02      0.60      0.03        92

    accuracy                           0.68     10355
   macro avg       0.51      0.64      0.42     10355
weighted avg       0.99      0.68      0.80     10355
```





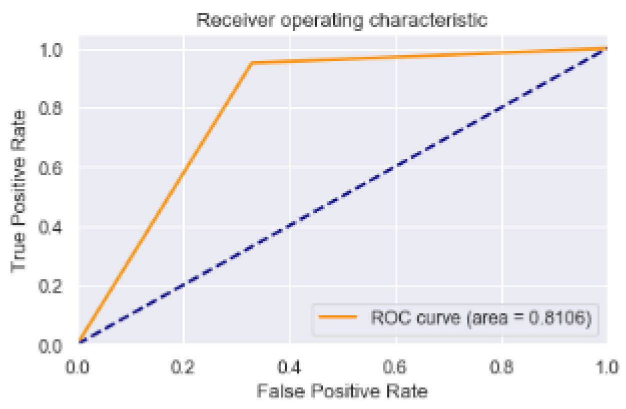
*NB: LA SURFACE F1 CALCULée n'est pas correcte (base * hauteur / 2) => A corriger*

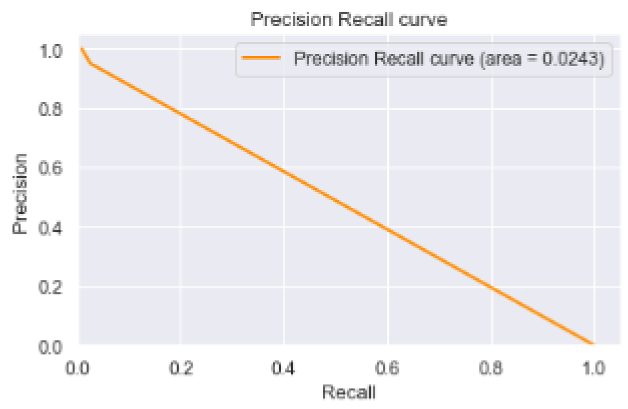
b - Balanced Random Forest Classifier - Cross Validation + F1 et ROC:

```
X_train, y_train = X_df, y_df

# 2 - Initialize a CV Split
CV = StratifiedKFold(n_splits=8) # , random_state=48, shuffle=True

# 3 - Imputer et Scaler + classifier
modeler = ValeoModeler()
pred = ValeoPredictor()
BRFC = BalancedRandomForestClassifier(n_estimators = 300 , max_depth=20, random_state=0)
pl = Pipeline([('preprocessor', modeler.build_transformers_pipeline(X_train.dtypes)),  # --> Imputer +
Scaler
              ('classifier', BRFC)  # --> Balanced Random Forest Classifier
              ])

# 4 - Cross Validate
cv_results =  cross_validate(pl, X_train, y_train, cv=CV, scoring=('f1', 'f1_micro', 'f1_macro',
'f1_weighted', 'recall', 'precision', 'average_precision', 'roc_auc'), return_train_score=True,
return_estimator=True)
fitted_estimators = []
for key in cv_results.keys() :
    if str(key) !=  "estimator" :
        print(f"{key} : {cv_results[key]}")
    fitted_estimators.append(cv_results[key])

fitted_model = cv_results["estimator"][np.argmax(cv_results["test_roc_auc"])]
pred.predict_and_plot(fitted_model,X_test, y_test)

# 5 - Test using ENS data
X_ens = DfUtil.read_csv([Const.rootDataTest() , "testinputs.csv"])
y_ens = fitted_model.predict(X_ens.drop(columns=[Const.PROC_TRACEINFO]))
DfUtil.write_y_csv(X_ens[Const.PROC_TRACEINFO], y_ens, Const.Binar_OP130_Resultat_Global_v,
[Const.rootDataTest() , "testoutput.csv"])
```

```
fit_time : [4.20418215 3.88397908 3.76627254 3.901299   3.95574546 3.66990876
 4.14086699 4.21298003]
score_time : [0.34399605 0.31799507 0.32755017 0.2992568  0.3776269
0.30384541
 0.35573673 0.38599515]
test_f1 : [0.02617801 0.03069054 0.03129445 0.03215434 0.03556772 0.02931379
 0.03125    0.02610587]
train_f1 : [0.05155933 0.04991121 0.05410353 0.0492302  0.05065933 0.0512476
 0.05085714 0.05507426]
test_f1_micro : [0.65515643 0.64866744 0.68435689 0.65113584 0.67315716
0.6622624
 0.66944831 0.68868799]
train_f1_micro : [0.6747351  0.6634106  0.69201987 0.65852124 0.66865336
0.67265985
 0.67001093 0.69663256]
test_f1_macro : [0.40832978 0.408071   0.42137812 0.40968667 0.41940261
0.41243997
 0.41598833 0.42041947]
train_f1_macro : [0.42763407 0.42269408 0.43508467 0.420559   0.42498069
0.4267282
 0.42557285 0.43719279]
test_f1_weighted : [0.78375072 0.77880466 0.80441046 0.78056799 0.79647547
0.7888166
 0.7939487  0.80778642]
train_f1_weighted : [0.79705901 0.78888536 0.80935449 0.78532214 0.79268348
0.79556972
 0.79366302 0.81255488]
test_recall : [0.52631579 0.63157895 0.56410256 0.65789474 0.68421053
0.57894737
 0.60526316 0.47368421]
train_recall : [1. 1. 1. 1. 1. 1. 1. 1.]
test_precision : [0.01342282 0.01572739 0.01609364 0.01647989 0.01825843
0.01503759
 0.01603905 0.01342282]
train_precision : [0.02646184 0.02559433 0.02780391 0.02523629 0.02598793
0.02629765
 0.02609206 0.02831689]
test_average_precision : [0.01460394 0.01852067 0.0140107  0.02250251
0.03696366 0.01684649
 0.01927391 0.01715631]
train_average_precision : [0.36489053 0.38781819 0.34316751 0.43090551
0.37367985 0.36632035
 0.38315343 0.36073824]
test_roc_auc : [0.61249892 0.64770252 0.63006404 0.70079637 0.73938383
0.65292206
 0.66054724 0.62940648]
train_roc_auc : [0.95913257 0.96622369 0.95572754 0.96683853 0.96490919
0.96163445
 0.96284347 0.96020734]
- Model score: 0.6728958423873678
- Accuracy score: 0.6728958423873678
- Balanced accuracy score: 0.8106188392810079 / The balanced accuracy to deal
with imbalanced datasets. It is defined as the average of recall obtained on
each class.
- Average_precision_score: 0.024277023825104087
- Precision_score: 0.025075659316904454
- Recall score: 0.9508196721311475
- Roc_auc_score: 0.8106188392810079
- F1 score: 0.04886267902274642
- [4587 2255]/[ 3 58] - P:0.0251 - R:0.9508 - roc_auc:0.8106 - f1:0.0489
- [[4587 2255]
 [   3   58]]
- Classification_report_imbalanced:
                   pre       rec       spe        f1       geo       iba
sup

          0       1.00      0.67      0.95      0.80      0.80      0.62
6842
          1       0.03      0.95      0.67      0.05      0.80      0.66
61

avg / total       0.99      0.67      0.95      0.80      0.80      0.62
```

```
6903

- classification_report:
              precision    recall  f1-score   support

           0       1.00      0.67      0.80      6842
           1       0.03      0.95      0.05        61

    accuracy                           0.67      6903
   macro avg       0.51      0.81      0.43      6903
weighted avg       0.99      0.67      0.80      6903

- precision_recall_curve: (array([0.00883674, 0.02507566, 1.        ]),
array([1.        , 0.95081967, 0.        ]), array([0, 1], dtype=int64))
- precision_recall_fscore_support: (array([0.99934641, 0.02507566]),
array([0.67041801, 0.95081967]), array([0.80248425, 0.04886268]),
array([6842,   61], dtype=int64))
- roc_curve: (array([0.        , 0.32958199, 1.        ]), array([0.        ,
0.95081967, 1.        ]), array([2, 1, 0], dtype=int64))
```





NB: LA SURFACE F1 CALCULée n'est pas correcte (base x hauteur / 2) => A corriger

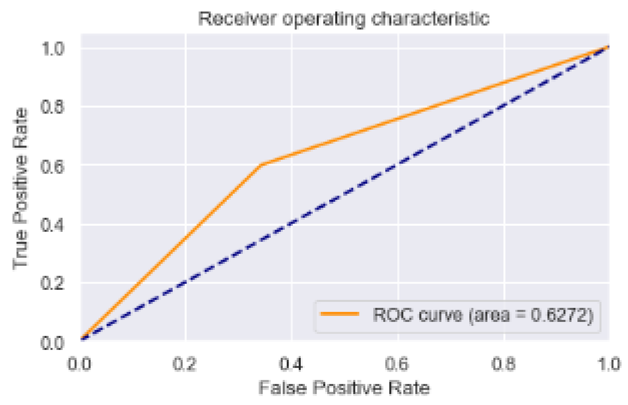
12 - Modèle à base de distance : Logistique regression avec SMOTE oversampling

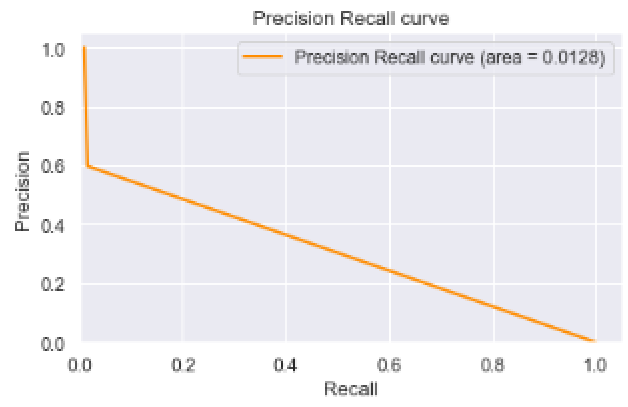
a – Logistique regression avec SMOTE - Train / Test / Split + F1 et ROC:

```
X_train, X_test, y_train, y_test = train_test_split(X_df, y_df, test_size=0.3, random_state=48,
stratify=y_df)

# 3 - Imputer et Scaler + SMOTE + Logistic Regression
modeler = ValeoModeler()
pred = ValeoPredictor()
pl = Pipeline([('preprocessor', modeler.build_transformers_pipeline(X_train.dtypes)),            # -->
Imputer + Scaler
               ('imbalancer_resampler', SMOTE(sampling_strategy='minority', random_state=7)),    # -->
SMOTE oversampling
               ('classifier', LRC)  # --> Logistic Regression Classifier
              ])
pl.fit(X_train, y_train)
pred.predict_and_plot(pl,X_test, y_test)

# 5 - Test using ENS data
X_ens = DfUtil.read_csv([Const.rootDataTest() , "testinputs.csv"])
y_ens = pl.predict(X_ens.drop(columns=[Const.PROC_TRACEINFO]))
DfUtil.write_y_csv(X_ens[Const.PROC_TRACEINFO], y_ens, Const.Binar_OP130_Resultat_Global_v,
[Const.rootDataTest() , "testoutput.csv"])
```

```
- Model score: 0.6560115886045389
- Accuracy score: 0.6560115886045389
- Balanced accuracy score: 0.6271796321950103 / The balanced accuracy to deal
with imbalanced datasets. It is defined as the average of recall obtained on
each class.
- Average_precision_score: 0.012757632055707119
- Precision_score: 0.015363128491620111
- Recall score: 0.5978260869565217
- Roc_auc_score: 0.6271796321950104
- F1 score: 0.029956427015250544
- [6738 3525]/[37 55] - P:0.0154 - R:0.5978 - roc_auc:0.6272 - f1:0.0300
- [[6738 3525]
 [  37   55]]
- classification_report_imbalanced:
                   pre       rec       spe        f1       geo       iba
sup

          0       0.99      0.66      0.60      0.79      0.63      0.39
10263
          1       0.02      0.60      0.66      0.03      0.63      0.39
92

avg / total       0.99      0.66      0.60      0.78      0.63      0.39
10355

- classification_report:
              precision    recall  f1-score   support
```

```
           0       0.99      0.66      0.79     10263
           1       0.02      0.60      0.03        92

    accuracy                           0.66     10355
   macro avg       0.50      0.63      0.41     10355
weighted avg       0.99      0.66      0.78     10355
```





NB: LA SURFACE F1 CALCULée n'est pas correcte (base x hauteur / 2) => A corriger

b - Logistique regression avec SMOTE - Cross Validation + F1 et ROC:

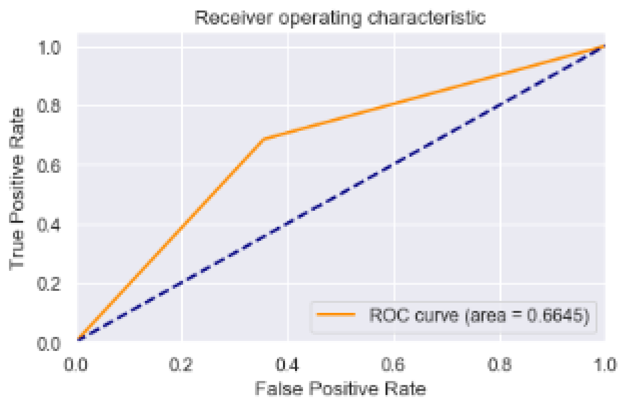
```
X_train, y_train = X_df, y_df

# 2 - Initialize a CV Split
CV = StratifiedKFold(n_splits=8) # , random_state=48, shuffle=True

# 3 - Imputer et Scaler + classifier
modeler = ValeoModeler()
pred = ValeoPredictor()
pl = Pipeline([('preprocessor', modeler.build_transformers_pipeline(X_train.dtypes)),            # -->
Imputer + Scaler
               ('imbalancer_resampler', SMOTE(sampling_strategy='minority', random_state=7)),    # -->
SMOTE oversampling
               ('classifier', LRC)  # --> Logistic Regression Classifier
              ])

# 4 - Cross Validate
```

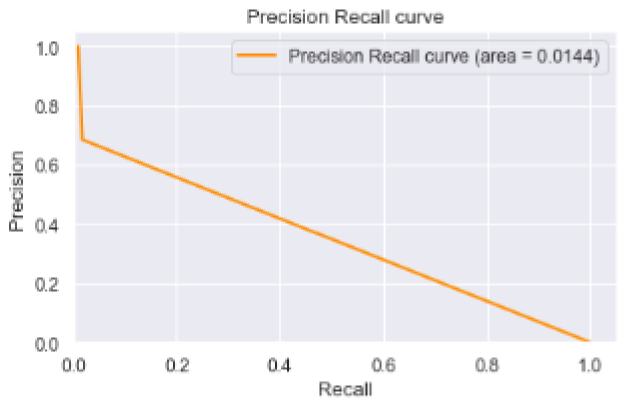
```
cv_results =  cross_validate(pl, X_train, y_train, cv=CV, scoring=('f1', 'f1_micro', 'f1_macro',
'f1_weighted', 'recall', 'precision', 'average_precision', 'roc_auc'), return_train_score=True,
return_estimator=True)
fitted_estimators = []
for key in cv_results.keys() :
    if str(key) !=  "estimator" :
        print(f"{key} : {cv_results[key]}")
    fitted_estimators.append(cv_results[key])

fitted_model = cv_results["estimator"][np.argmax(cv_results["test_roc_auc"])]
pred.predict_and_plot(fitted_model,X_test, y_test)

# 5 - Test using ENS data
X_ens = DfUtil.read_csv([Const.rootDataTest() , "testinputs.csv"])
y_ens = fitted_model.predict(X_ens.drop(columns=[Const.PROC_TRACEINFO]))
DfUtil.write_y_csv(X_ens[Const.PROC_TRACEINFO], y_ens, Const.Binar_OP130_Resultat_Global_v,
[Const.rootDataTest() , "testoutput.csv"])
```

```
fit_time : [2.36323762 1.91440773 2.06579304 2.04484057 1.85639429 1.75007534
 1.96227741 2.28214383]
score_time : [0.03199959 0.02400422 0.0418551  0.02399921 0.02499723
0.03502941
 0.03199911 0.03299737]
test_f1 : [0.02473958 0.02678028 0.0290429  0.03076923 0.0341556  0.03088803
 0.0297542  0.03050398]
train_f1 : [0.03089371 0.03005464 0.03097345 0.03165416 0.03028654 0.03054707
 0.03066471 0.03193988]
test_f1_micro : [0.65283893 0.62943221 0.65909618 0.64951321 0.64603616
0.65090403
 0.65229485 0.66110338]
train_f1_micro : [0.65099338 0.64735099 0.65192053 0.64547532 0.64590576
0.6532234
 0.65040893 0.65878613]
test_f1_macro : [0.40678761 0.39896335 0.4111483  0.40842366 0.40873382
0.40899787
 0.40897481 0.41258345]
train_f1_macro : [0.4090338  0.40727742 0.40941633 0.40733584 0.40684741
0.40969681
 0.40870743 0.41241667]
test_f1_weighted : [0.78210663 0.76459116 0.78634657 0.77942494 0.77671308
0.78044653
 0.78151467 0.7879318 ]
train_f1_weighted : [0.78048758 0.7778301  0.78119261 0.7763749  0.7767501
0.7821426
 0.78006578 0.78616605]
test_recall : [0.5        0.57894737 0.56410256 0.63157895 0.71052632
0.63157895
 0.60526316 0.60526316]
train_recall : [0.62921348 0.61797753 0.63157895 0.65543071 0.62546816
0.61797753
 0.62546816 0.63670412]
test_precision : [0.01268358 0.01370717 0.01490515 0.01576873 0.01749838
0.01583113
 0.01525199 0.01564626]
```

```
train_precision : [0.01583561 0.01540185 0.01587602 0.01621872 0.015519
0.01566059
 0.01571765 0.01638081]
test_average_precision : [0.01615652 0.01554629 0.01585269 0.03865314
0.02021889 0.01731332
 0.05091361 0.0200504 ]
train_average_precision : [0.01932375 0.01975579 0.02047861 0.01765319
0.01911704 0.02003393
 0.01881078 0.01992643]
test_roc_auc : [0.60311581 0.65022827 0.63546689 0.70105485 0.71515435
0.65763626
 0.67102801 0.64764167]
train_roc_auc : [0.68933515 0.679299   0.68218565 0.67620075 0.67052259
0.67673789
 0.68010722 0.68292153]
- Model score: 0.6446161274746499
- Accuracy score: 0.6446161274746499
- Balanced accuracy score: 0.6645193370867913 / The balanced accuracy to deal
with imbalanced datasets. It is defined as the average of recall obtained on
each class.
- Average_precision_score: 0.014416439513109575
- Precision_score: 0.016962843295638127
- Recall score: 0.6847826086956522
- Roc_auc_score: 0.6645193370867913
- F1 score: 0.03310562270099843
- [6612 3651]/[29 63] - P:0.0170 - R:0.6848 - roc_auc:0.6645 - f1:0.0331
- [[6612 3651]
 [  29   63]]
- classification_report_imbalanced:
                   pre       rec       spe        f1       geo       iba
sup

          0       1.00      0.64      0.68      0.78      0.66      0.44
10263
          1       0.02      0.68      0.64      0.03      0.66      0.44
92

avg / total       0.99      0.64      0.68      0.78      0.66      0.44
10355

- classification_report:
              precision    recall  f1-score   support

           0       1.00      0.64      0.78     10263
           1       0.02      0.68      0.03        92

    accuracy                           0.64     10355
   macro avg       0.51      0.66      0.41     10355
weighted avg       0.99      0.64      0.78     10355
```

```
- precision_recall_curve: (array([0.0088846 , 0.01696284, 1.        ]),
array([1.        , 0.68478261, 0.        ]), array([0, 1], dtype=int64))
- precision_recall_fscore_support: (array([0.99563319, 0.01696284]),
array([0.64425607, 0.68478261]), array([0.78230005, 0.03310562]),
array([10263,    92], dtype=int64))
- roc_curve: (array([0.        , 0.35574393, 1.        ]), array([0.        ,
0.68478261, 1.        ]), array([2, 1, 0], dtype=int64))
```





NB: LA SURFACE F1 CALCULée n'est pas correcte (base x hauteur / 2) => A corriger

13 - Modèle à base de réseau de neurones ou de stacking : TODO

14 - Conclusion : TODO

15 - Perspectives : TODO

16 – Annexe : Code Python

File - C:\EXED\Training\___VALEO\src\valeo\domain\ValeoModeler.py

```python
from imblearn.ensemble import BalancedBaggingClassifier, RUSBoostClassifier, BalancedRandomForestClassifier
from imblearn.over_sampling import RandomOverSampler, ADASYN, SMOTE, SVMSMOTE, KMeansSMOTE, BorderlineSMOTE
from imblearn.over_sampling.base import BaseOverSampler
from imblearn.pipeline import Pipeline
from sklearn.cluster import MiniBatchKMeans
from sklearn.compose import ColumnTransformer

from sklearn.ensemble._hist_gradient_boosting.gradient_boosting import HistGradientBoostingClassifier
from sklearn.impute import SimpleImputer
# from sklearn.impute._iterative import IterativeImputer
from sklearn.experimental import enable_iterative_imputer   # explicitly require this experimental feature
from sklearn.impute import IterativeImputer
from sklearn.linear_model import LogisticRegression, BayesianRidge
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import RobustScaler, MinMaxScaler, label_binarize, StandardScaler
from sklearn.svm import SVC
import xgboost as xgb

import pandas as pd
import numpy as np

from valeo.infrastructure.LogManager import LogManager
from valeo.infrastructure.tools.DebugPipeline import DebugPipeline
from valeo.infrastructure import Const as C

'''
https://github.com/scikit-learn-contrib/imbalanced-learn/tree/master/examples
https://towardsdatascience.com/introduction-to-bayesian-linear-regression-e66e60791ea7
https://towardsdatascience.com/custom-transformers-and-ml-data-pipelines-with-python-20ea2a7adb65
https://jorisvandenbossche.github.io/blog/2018/05/28/scikit-learn-columntransformer/
'''

class ValeoModeler :
    logger = None

    def __init__(self):
        logger = LogManager.logger(__name__)

    def build_transformers_pipeline(self, features_dtypes:pd.Series) -> ColumnTransformer:
        rand_state = 48
        numerical_features = (features_dtypes == 'int64') | (features_dtypes == 'float64')
        # categorical_features = ~numerical_features
        # nan_imputer    = SimpleImputer(strategy='mean', missing_values=np.nan, verbose=False)
        nan_imputer    = IterativeImputer(estimator=BayesianRidge(), missing_values=np.nan,  max_iter=10, initial_strategy = 'median',  add_indicator=True, random_state=rand_state)
        zeroes_imputer = IterativeImputer(estimator=BayesianRidge(), missing_values=0,  max_iter=10, initial_strategy = 'median',  add_indicator=True, random_state=rand_state)
        scaler         =  RobustScaler(with_centering=True, with_scaling=True, quantile_range=(25.0, 75.0))  # Normalizer()  # RobustScaler() #StandardScaler() # RobustScaler(with_centering=True, with_scaling=False)  # MinMaxScaler()
        # scaler  = Normalizer(norm='l1')
        # NB: When using log tranformer: Adopt this transformation -> log(-2) = -1*(log(abs(-2)+1))
        # dbg = DebugPipeline()
        num_transformers_pipeline = Pipeline([ #('dbg_0', dbg),
            ('nan_imputer', nan_imputer),       # ('dbg_1', dbg),
            ('zeroes_imputer', zeroes_imputer), # ('dbg_2', dbg),
            ('scaler', scaler),                 # ('dbg_3', dbg)
        ])
        return ColumnTransformer([('transformers_pipeline',num_transformers_pipeline, numerical_features)], remainder='passthrough')

                           # ENS(0.61) without explicit overSampling / test_roc_auc : [0.6719306  0.58851217 0.58250362 0.6094371  0.55757417]
    BBC  = "BBC"           # BalancedBaggingClassifier(base_estimator=HGBR,  sampling_strategy=1.0, replacement=False, random_state=48)
    HGBC = "HGBR"          # HistGradientBoostingClassifier(max_iter = 8 , max_depth=8,learning_rate=0.35, l2_regularization=500)

    BRFC = "BRFC"          # BalancedRandomForestClassifier(n_estimators = 50 , max_depth=20)
    RUSBoost = "RUSBoost" # RUSBoostClassifier(n_estimators = 8 , algorithm='SAMME.R', random_state=42)
    KNN = "KNN"            # KNeighborsClassifier(3),
    SVC = "SVC"            # SVC(kernel="rbf", C=0.025, probability=True)
    NuSVC = "NuSVC"        # NuSVC(probability=True),
    RFC = "RFC"            # RandomForestClassifier(n_estimators=10, max_depth=10, max_features=10, n_jobs=4))
    DTC = "DTC"            # DecisionTreeClassifier())  # so bad
    ADABoost = "ADABoost" # AdaBoostClassifier()
    GBC  = "GBC"           # GradientBoostingClassifier()
    LRC  = "LRC"           # LogisticRegression(max_iter=500))  # Best for Recall 1
    XGBC = "XGBC"          # xgb.XGBClassifier()
    #  ('classification', GaussianNB())  # 0.5881085402220386
    #  ('classification', ComplementNB())  # 0.523696690978335
    #  ('classification', MultinomialNB())  # 0.523696690978335
    Imbl_Resampler =  "Imbl_Resampler"  # ('imbalancer_resampler', self.build_resampler(sampler_type, sampling_strategy='not majority'))

    def build_predictor_pipeline(self, features_dtypes:pd.Series, clfTypes:[str]) -> Pipeline:
        cls = self.__class__
        clfs = {
            cls.HGBC : HistGradientBoostingClassifier(max_iter = 100 , max_depth=10,learning_rate=0.10, l2_regularization=5),
            cls.BBC  : BalancedBaggingClassifier(base_estimator=HistGradientBoostingClassifier(),  n_estimators=50, sampling_strategy='auto', replacement=False, random_state=48),

            # scale_pos_weight
            # ESTIM:100 depth:20 [6155 4108]/[41 51] - P:0.0123 - R:0.5543 - roc_auc:0.5770 - f1:0.0240 |
            #.ESTIM:300 depth:10 [6085 4178]/[37 55] - P:0.0130 - R:0.5978 - roc_auc:0.5954 - f1:0.0254
            # ESTIM:300 depth:15 [6306 3957]/[37 55] - P:0.0137 - R:0.5978 - roc_auc:0.6061 - f1:0.0268
            # ESTIM:300 depth:20 [6057 4206]/[33 59] - P:0.0138 - R:0.6413 - roc_auc:0.6157 - f1:0.0271   ***
            # ESTIM:300 depth:20 class_weight:{0:1, 1:100} [2860 7403]/[22 70] - P:0.0094 - R:0.7609 - roc_auc:0.5198 - f1:0.0185
            # [6127 4136]/[35 57] - P:0.0136 - R:0.6196 - roc_auc:0.6083 - f1:0.0266
            # [6184 4079]/[37 55] - P:0.0133 - R:0.5978 - roc_auc:0.6002 - f1:0.0260
            # [6121 4142]/[37 55] - P:0.0131 - R:0.5978 - roc_auc:0.5971 - f1:0.0256
            # ESTIM:300 depth:30 [6223 4040]/[36 56] - P:0.0137 - R:0.6087 - roc_auc:0.6075 - f1:0.0267
            # ESTIM:300 depth:40 [6243 4020]/[39 53] - P:0.0130 - R:0.5761 - roc_auc:0.5922 - f1:0.0255
            #.ESTIM:200 depth:10 [6236 4027]/[39 53] - P:0.0130 - R:0.5761 - roc_auc:0.5919 - f1:0.0254
            # ESTIM:200 depth:20 [6104 4159]/[34 58] - P:0.0138 - R:0.6304 - roc_auc:0.6126 - f1:0.0269
            # ESTIM:200 depth:40 [6227 4036]/[37 55] - P:0.0134 - R:0.5978 - roc_auc:0.6023 - f1:0.0263
            cls.BRFC : BalancedRandomForestClassifier(n_estimators = 300 , max_depth=20, random_state=0),

            cls.RUSBoost : RUSBoostClassifier(n_estimators = 8 , algorithm='SAMME.R', random_state=42),
            cls.XGBC :  xgb.
                XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                              colsample_bynode=1, colsample_bytree=1, gamma=0,
                              learning_rate=0.1, max_delta_step=0, max_depth=10, #max_depth=3,
                              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
                              nthread=None, objective='binary:logistic', random_state=0,
                              reg_alpha=0, reg_lambda=1, scale_pos_weight=100, seed=42,
                              silent=None, subsample=1, verbosity=1)
        }
        dbg = DebugPipeline()
        pl= Pipeline([('preprocessor', self.build_transformers_pipeline(features_dtypes)) ,
                       # ('imbalancer_resampler', self.build_resampler(C.smote_over_sampling,sampling_strategy='minority')),  # ('dbg_1', dbg),
                      ('classifier', clfs[clfTypes[0]])  # ex: bbc : ENS(0.61) without explicit overSampling / test_roc_auc : [0.6719306  0.58851217 0.58250362 0.6094371  0.55757417]
                      ])
        for i, s in enumerate(pl.steps) :
            # Ex: 0 -> ('preprocessor', ColumnTransformer( ... +  1 -> ('classifier', BalancedBaggingClassifier( base_.....
            print(f"{i} -> {s[0]} / {str(s[1])[:70]}")
        return pl


    '''
    SMOTe is a technique based on nearest neighbors judged by Euclidean Distance between data points in feature space.
    random_over_sampling : The most naive strategy is to generate new samples by randomly sampling with replacement the current available samples.
    adasyn_over_sampling : Adaptive Synthetic: focuses on generating samples next to the original samples which are wrongly classified using a k-Nearest Neighbors classifier
    smote_over_sampling  : Synth Minority Oversampling Techn: will not make any distinction between easy and hard samples to be classified using the nearest neighbors rule
    ---
    https://medium.com/towards-artificial-intelligence/application-of-synthetic-minority-over-sampling-technique-smote-for-imbalanced-data-sets-509ab55cfdaf
    https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html
    NB:
    How to apply SMOTE : Shuffling and Splitting the Dataset into Training and Validation Sets and THEN applying SMOTe on the Training Dataset.
    '''
    def build_resampler(self, sampler_type: str, sampling_strategy='auto', k_neighbors=5) -> BaseOverSampler :
        rand_state = 48
        if sampler_type.lower() == C.random_over_sampler :
            return RandomOverSampler(sampling_strategy=sampling_strategy, random_state=rand_state)
        elif sampler_type.lower() == C.adasyn_over_sampling :
            return ADASYN(sampling_strategy=sampling_strategy, random_state=rand_state, n_neighbors=k_neighbors)
        elif sampler_type.lower() == C.smote_over_sampling :
            return SMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=k_neighbors)
        # elif sampler_type.lower() == C.smote_nc_over_sampling :      # SMOTE for dataset containing continuous and categorical features.
        #     return SMOTENC(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=k_neighbors)
        elif sampler_type.lower() == C.smote_svm_over_sampling :    # Use an SVM algorithm to detect sample to use for generating new synthetic samples
            return SVMSMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=k_neighbors, svm_estimator=SVC())
        elif sampler_type.lower() == C.smote_kmeans_over_sampling : # Apply a KMeans clustering before to over-sample using SMOTE
            return KMeansSMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=k_neighbors, kmeans_estimator=MiniBatchKMeans(n_clusters=2), cluster_balance_threshold=5)
        elif sampler_type.lower() == C.smote_bline_over_sampling :  # Borderline samples will be detected and used to generate new synthetic samples.
            return BorderlineSMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=k_neighbors, m_neighbors=3)
        else :
            raise ValueError(f"Unexpected argument [sampler_type:{sampler_type}] for method '
compute_sampler_preprocessor'")


# classifiers = [
#     KNeighborsClassifier(3),
#     SVC(kernel="rbf", C=0.025, probability=True),
#     NuSVC(probability=True),
#     DecisionTreeClassifier(),
#     RandomForestClassifier(),
#     AdaBoostClassifier(),
#     GradientBoostingClassifier()
# ]
# for classifier in classifiers:
#     pipe = Pipeline(steps=[('preprocessor', preprocessor),
#                            ('classifier', classifier)])
#     pipe.fit(X_train, y_train)
#     print(classifier)
#     print("model score: %.3f" % pipe.score(X_test, y_test))
```

File - C:\EXED\Training\___VALEO\src\valeo\domain\MetricPlotter.py

```python
from valeo.infrastructure.LogManager import LogManager

import matplotlib.pyplot as plt
from valeo.infrastructure import Const as C
from sklearn.metrics import roc_auc_score, precision_recall_curve, roc_curve, average_precision_score

from valeo.infrastructure.tools.ImgUtil import ImgUtil


class MetricPlotter :
    logger = None

    def __init__(self):
        MetricPlotter.logger = LogManager.logger(__name__);

    def plot_roc(self, y_test, y_pred):
        # y_test = label_binarize(y_test.values, classes=[0, 1])  # y_test 'Series'
        # y_pred = label_binarize(y_pred, classes=[0, 1])         # y_pred  'numpy.ndarray'
        plt.figure()
        lw = 2
        roc = roc_curve(y_test, y_pred)
        plt.plot(roc[0], roc[1], color='darkorange', lw=lw, label='ROC curve (area = %0.4f)' % roc_auc_score(
y_test, y_pred))
        plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Receiver operating characteristic')
        plt.legend(loc="lower right")
        ImgUtil.save_fig("ROC_curve")
        plt.show()

    def plot_precision_recall(self, y_test, y_pred):
        average_precision = average_precision_score(y_test, y_pred)
        plt.figure()
        lw = 2
        pr = precision_recall_curve(y_test, y_pred)
        plt.plot(pr[0], pr[1], color='darkorange', lw=lw, label='Precision Recall curve (area = %0.4f)' %
average_precision)
        plt.xlim([0.0, 1.05])
        plt.ylim([0.0, 1.05])
        plt.xlabel('Recall')
        plt.ylabel('Precision')
        plt.title('Precision Recall curve')
        plt.legend(loc="upper right")
        ImgUtil.save_fig("PR_curve")
        plt.show()
        #
        # for i in range(0, len(pr[0]) ) :
        #     print(f"{i}: ({pr[0][i]},{pr[1][i]})")

```

File - C:\EXED\Training\___VALEO\src\valeo\domain\ValeoPipeline.py

```python
from imblearn.pipeline import make_pipeline, Pipeline
from imblearn.ensemble import BalancedBaggingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, auc, roc_auc_score
from sklearn.tree import DecisionTreeClassifier

import pandas as pd
from sklearn.model_selection import train_test_split

from valeo.domain.ValeoPreprocessor import ValeoPreprocessor
from valeo.infrastructure import Const as C
from valeo.infrastructure.LogManager import LogManager


class ValeoPipeline:
    logger = None

    def __init__(self):
        self.preproc = ValeoPreprocessor()
        ValeoPipeline.logger = LogManager.logger(__name__)

    def pplSmote(self):
        Pipeline([('column_preprocessor', self.preproc.build_column_preprocessor()) ,
                  ('smote_resampler', self.preproc.build_resampler(C.smote_over_sampling))])

    # https://towardsdatascience.com/custom-transformers-and-ml-data-pipelines-with-python-20ea2a7adb65
    def execute(self, X_df:pd.DataFrame, y_df:pd.DataFrame, sampler_type: str):
        # setting up testing and training sets
        X_train, X_test, y_train, y_test = train_test_split(X_df, y_df, test_size=0.25, random_state=48)

        #Create an object of the classifier.
        bbc = BalancedBaggingClassifier(base_estimator=DecisionTreeClassifier(),
                                        sampling_strategy='auto',
                                        replacement=False,
                                        random_state=0)

        p = Pipeline([('column_preprocessor', self.preproc.build_column_preprocessor()) ,
                      # ('smote_resampler', self.preproc.build_resampler(sampler_type)),
                      # ('classification', LogisticRegression())
                      # ('classifier', bbc)
                      ])

        # p.fit_resample(X_train, y_train)
        p.fit_transform(X_train, y_train)
        # p.fit(X_train, y_train)
        #
        # y_predict = p.predict(X_test)
        # x = f1_score(y_test, y_predict)
        # y = 0 # auc(y_test, y_predict)
        # z = 0 # roc_auc_score(y_test, y_predict)
        # ValeoPipeline.logger.info(f"F1:{x} - auc:{y} - roc_auc:{z}")


# ---------------------------------
# Exemple Type : PipeLine entier
# ---------------------------------
# >>> pca = PCA()
# >>> smt = SMOTE(random_state=42)
# >>> knn = KNN()
# >>> pipeline = Pipeline([('smt', smt), ('pca', pca), ('knn', knn)])



# -----------------------
# Exemple_1
# -----------------------
# from sklearn.compose import ColumnTransformer
# from sklearn.pipeline import Pipeline
#
# # 1 - Define Categorical pipe_line
# cat_col = ['sex', 'embarked', 'pclass']
# cat_pipeline = Pipeline(steps=[
#     ("constant-imputer", SimpleImputer(strategy='constant', fill_value='missing')),
#     ("ordinal-encoder", OrdinalEncoder()),
# ])
#
# # 2 - Define Numerical pipe_line
# num_cols = ['age', 'parch', 'fare']
# num_pipeline = SimpleImputer(
#     strategy="mean", add_indicator=True,
# )
#
# # 3 - Define Column Transformer
# preprocessor = ColumnTransformer(transformers=[
#     ("cat-preprocessor", cat_pipeline, cat_col),
#     ("num-preprocessor", num_pipeline, num_cols),
# ])
#
```

```python
# model = Pipeline(steps=[
#     ("preprocessor", preprocessor),
#     ("clf", RandomForestClassifier(n_estimators=100))
# ])
#
# _ = model.fit(X_train, y_train)
#
# (model.named_steps["preprocessor"]
#  .named_transformers_["cat-preprocessor"]
#  .named_steps["ordinal-encoder"].categories_)



# -----------------------
# Exemple_2
# -----------------------
# define the pipelines
# cat_pipe = make_pipeline(
#     SimpleImputer(strategy='constant', fill_value='missing'),
#     OrdinalEncoder(categories=categories)
# )
# num_pipe = SimpleImputer(strategy='mean')
#
# preprocessing = ColumnTransformer(
#     [('cat_preprocessor', cat_pipe, cat_col),
#      ('num_preprocessor', num_pipe, num_cols)]
# )
```

```python

from imblearn.metrics._classification import classification_report_imbalanced
# https://imbalanced-learn.readthedocs.io/en/stable/api.html#module-imblearn.pipeline
from sklearn.base import BaseEstimator
from sklearn.metrics import f1_score, auc, roc_auc_score, confusion_matrix, classification_report, \
    precision_recall_curve, precision_recall_fscore_support, roc_curve, plot_precision_recall_curve, \
    average_precision_score, precision_score, recall_score, accuracy_score, balanced_accuracy_score
# from sklearn.impute import SimpleImputer
from sklearn.model_selection import cross_validate, StratifiedKFold, GridSearchCV, RandomizedSearchCV

import pandas as pd

from valeo.domain.MetricPlotter import MetricPlotter
from valeo.domain.ValeoModeler import ValeoModeler
from valeo.infrastructure.tools.DfUtil import DfUtil
from valeo.infrastructure.LogManager import LogManager
from valeo.infrastructure import Const as C

import xgboost as xgb


class ValeoPredictor :
    logger = None

    def __init__(self):
        ValeoPredictor.logger = LogManager.logger(__name__)
        self.modeler = ValeoModeler()
        self.metricPlt = MetricPlotter()


    def fit_cv_grid_search(self, X:pd.DataFrame, y:pd.DataFrame, clfTypes:[str] , n_splits=5) -> ([BaseEstimator], dict): # (estimator, cv_results)
        model = self.modeler.build_predictor_pipeline(X.dtypes, clfTypes) # sampler_type)
        CV = StratifiedKFold(n_splits=n_splits) # , random_state=48, shuffle=True
        # HGBC
        # param_grid = {
        #     'classifier__n_estimators': [3, 5, 10, 20, 50],
        #     'classifier__base_estimator__l2_regularization': [5, 50, 100, 50],
        #     'classifier__base_estimator__max_iter' : [100],
        #     'classifier__base_estimator__max_depth' : [10,50,10]
        # }
        # BRFC
        param_grid = {
            'classifier__n_estimators': [250,300],
            'classifier__max_depth': [15,20,25],
            'classifier__max_features' : ['auto',13]
        }

        grid = GridSearchCV(model, param_grid=param_grid, n_jobs=-1, cv=CV) # if is_grid else
        grid.fit(X, y)
        print(f"Best Estimator: {grid.best_estimator_}")
        df_results = pd.DataFrame(grid.cv_results_)
                    # columns_to_keep = ['param_clf__max_depth', 'param_clf__n_estimators', 'mean_test_score', 'std_test_score',]
                    # df_results = df_results[columns_to_keep]
        DfUtil.write_df_csv( df_results.sort_values(by='mean_test_score', ascending=False), C.ts_pathanme([C.rootReports(), 'grid_search_cv.csv']) )

    def fit_cv_randomized_search(self, X:pd.DataFrame, y:pd.DataFrame, clfTypes:[str] , n_splits=5) -> ([BaseEstimator], dict): # (estimator, cv_results)
        model = self.modeler.build_predictor_pipeline(X.dtypes, clfTypes) # sampler_type)
        CV = StratifiedKFold(n_splits=n_splits) # , random_state=48, shuffle=True
        # HGBC
        # param_grid = {
        #     'classifier__n_estimators': [3, 5, 10, 20, 50],
        #     'classifier__base_estimator__l2_regularization': [5, 50, 100, 50],
        #     'classifier__base_estimator__max_iter' : [100],
        #     'classifier__base_estimator__max_depth' : [10,50,10]
        # }

        grid = RandomizedSearchCV(model, param_distributions=param_grid, n_jobs=-1, cv=CV) # if is_grid else
        grid.fit(X, y)
        df_results = pd.DataFrame(grid.cv_results_)
        DfUtil.write_df_csv( df_results.sort_values(by='mean_test_score', ascending=False), C.ts_pathanme([C.rootReports(), 'grid_search_csv']) )

    def print_model_params_keys(self, model:BaseEstimator):
        for param in model.get_params().keys():
            print(param)

    # 1 - Fit without any Cross Validation
    def fit_and_plot(self, X_train:pd.DataFrame, y_train:pd.DataFrame,  X_test:pd.DataFrame, y_test:pd.DataFrame, clfTypes:[str]) -> BaseEstimator:
        # Q1 = X_train.quantile(0.25)
        # Q3 = X_train.quantile(0.75)
        # IQR = Q3 - Q1
        # # print(IQR)
        # to_remove = ((X_train < (Q1 - 1.5 * IQR)) |(X_train > (Q3 + 1.5 * IQR))).any(axis=1)
        # y_train = y_train.drop(axis=0, index=X_train[to_remove].index)
        # X_train = X_train[~to_remove]
        #
        fitted_model = self.fit(X_train, y_train, clfTypes)
        # print(f"Type:{type(fitted_model)} - {fitted_model.get_params()}")
        # self.print_model_params_keys(fitted_model)
        self.predict_and_plot(fitted_model, X_test, y_test)
        return fitted_model

    def fit(self, X_train:pd.DataFrame, y_train:pd.DataFrame, clfTypes:[str]) -> BaseEstimator:
        model = self.modeler.build_predictor_pipeline(X_train.dtypes, clfTypes)
        return model.fit(X_train, y_train)

    ''' 2 - Fit with Cross Validation
        NB :
        a - roc-auc-avo + roc-auc-ovr :
            https://stackoverflow.com/questions/59453363/what-is-the-difference-of-roc-auc-values-in-sklearn
            roc_auc is the only one suitable for binary classification. The weighted, ovr and ovo are use for multi-class problems

        b - Micro-Average + Macro-Average (for Precision / Recall / F1) :
            http://rushdishams.blogspot.com/2011/08/micro-and-macro-average-of-precision.html
            https://datascience.stackexchange.com/questions/15989/micro-average-vs-macro-average-performance-in-a-multiclass-classification-settin
            Ex: Micro-P = (TP1 + TP2) / ( TP1 + FP1 + TP2 + F2)
                Macro-P = (P1 + P2) / 2
            Suitability:
            . Macro-average method can be used when you want to know how the system performs overall across the sets of data
            . Micro-average method can be a useful measure when your dataset varies in size.

        c - How can we report 'confusion matrix' while using 'cross_validate' ?
            https://stackoverflow.com/questions/40057049/using-confusion-matrix-as-scoring-metric-in-cross-validation-in-scikit-learn
            c1. Either use 'cross_val_predict' and deduce confusion-matrix:
                y_pred = cross_val_predict(clf, x, y, cv=10)
                conf_mat = confusion_matrix(y_test, y_pred)
                BUT BEWARE: Passing these predictions into an evaluation metric may not be a valid way to measure generalization performance.
                            Results can differ from cross_validate and cross_val_score unless all tests sets have equal size and the metric decomposes over samples.
            c2. If you want to obtain confusion matrices for multiple evaluation runs (such as cross validation) you have to do this by hand:
                conf_matrix_list_of_arrays = []
                kf = cross_validation.KFold(len(y), n_folds=5)
                for train_index, test_index in kf:
                   X_train, X_test = X[train_index], X[test_index]   # Panda-Column index 'train_index' are of type 'numpy array'
                   y_train, y_test = y[train_index], y[test_index]
                   #
                   model.fit(X_train, y_train)
                   conf_matrix = confusion_matrix(y_test, model.predict(X_test))
                   conf_matrix_list_of_arrays .append(conf_matrix)

                On the end you can calculate your mean of list of numpy arrays (confusion matrices) with:
                mean_of_conf_matrix_arrays = np.mean(conf_matrix_list_of_arrays, axis=0)
    '''
    def fit_cv(self, X:pd.DataFrame, y:pd.DataFrame, clfTypes:[str], n_splits=5) -> ([BaseEstimator], dict): # (estimator, cv_results)
        model = self.modeler.build_predictor_pipeline(X.dtypes, clfTypes)
        CV = StratifiedKFold(n_splits=n_splits) # , random_state=48, shuffle=True
        cv_results =  cross_validate(model, X, y, cv=CV, scoring=('f1', 'f1_micro', 'f1_macro', 'f1_weighted', 'recall', 'precision', 'average_precision', 'roc_auc'), return_train_score=True, return_estimator=True)
        fitted_estimators = []
        for key in cv_results.keys() :
            if str(key) !=  "estimator" :
                print(f"{key} : {cv_results[key]}")
            fitted_estimators.append(cv_results[key])
        return fitted_estimators, cv_results

    '''
        - Print metrics
        - Print report
        - Plot ROC : TP vs FP
        - Plot AUC : Precison vs Recall
    '''
    def predict_and_plot(self, fitted_model: BaseEstimator, X_test:pd.DataFrame, y_test:pd.DataFrame):
        y_pred = fitted_model.predict(X_test)
        #
        print(f"- Model score: {fitted_model.score(X_test, y_test)}")
        print(f"- Accuracy score: {accuracy_score(y_test, y_pred)}")
        print(f"- Balanced accuracy score: {balanced_accuracy_score(y_test, y_pred)} / The balanced accuracy to deal with imbalanced datasets. It is defined as the average of recall obtained on each class.")
        # print(f"- auc : {auc(y_test, y_pred)}")  # ValueError: x is neither increasing nor decreasing : [0 0 0 ... 0 0 0]
        print(f"- Average_precision_score: {average_precision_score(y_test, y_pred)}")
        print(f"- Precision_score: {precision_score(y_test, y_pred)}")
        print(f"- Recall score: {recall_score(y_test, y_pred)}")
        print(f"- Roc_auc_score: {roc_auc_score(y_test, y_pred)}")
        print(f"- F1 score: {f1_score(y_test, y_pred)}")
        m = confusion_matrix(y_test, y_pred)
        print(f"- {m[0]}/{m[1]} - P:{precision_score(y_test, y_pred):0.4f} - R:{recall_score(y_test, y_pred):0.4f} - roc_auc:{roc_auc_score(y_test, y_pred):0.4f} - f1:{f1_score(y_test, y_pred):0.4f}")
        print(f"- {confusion_matrix(y_test, y_pred)}")
        print(f"- classification_report_imbalanced:\n{classification_report_imbalanced(y_test, y_pred)}")
        print(f"- classification_report:\n{classification_report(y_test, y_pred)}")
        print(f"- precision_recall_curve: {precision_recall_curve(y_test, y_pred)}")
        print(f"- precision_recall_fscore_support: {precision_recall_fscore_support(y_test, y_pred)}")
        print(f"- roc_curve: {roc_curve(y_test, y_pred)}")
        #
        self.metricPlt.plot_roc(y_test, y_pred)
        self.metricPlt.plot_precision_recall(y_test, y_pred)
        # self.plot_roc(y_test, y_pred)
        # self.plot_precision_recall(y_test, y_pred)



# https://medium.com/towards-artificial-intelligence/application-of-synthetic-minority-over-sampling-technique-smote-for-imbalanced-data-sets-509ab55cfdaf
# from sklearn.ensemble import GradientBoostingClassifier
# from sklearn.model_selection import GridSearchCV
# parameters = {'n_estimators':[100,150,200,250,300,350,400,450,500],
#               'max_depth':[3,4,5]}
# clf= GradientBoostingClassifier()
# grid_search = GridSearchCV(param_grid = parameters, estimator = clf,
#                              verbose = 3)
# grid_search_2 = grid_search.fit(X_train,y_train)

# GOOGLE ON: classifier over sampled imbalanced dataset
# https://sci2s.ugr.es/imbalanced  : Tres interessant****
# https://www.datacamp.com/community/tutorials/diving-deep-imbalanced-data  : Tres interessant****
#---
# https://journalofbigdata.springeropen.com/articles/10.1186/s40537-017-0108-1  : Tres interessant****
# xperimental evaluation
# The projected technique works on binary-class/multi-class imbalanced Big Data sets in the organization to recommended LVH. Four basic classifiers viz. Random Forest (-P 100-I 100-num-slots 1-K 0-M 1.0-V 0.001-S 1), Naïve Bayes, AdaBoostM1 (-P 100-S 1-I 10-W weka.classifiers.trees.DecisionStump) and MultiLayer Perceptron (-L 0.3-M 0.2-N 500-V 0-S 0-E 20-H a) are applied to over_sampled data sets using dissimilar values of cross-validation and KNN. Lastly the results, based on the F-measure and AUC values are used to compare between benchmarking (SMOTE/Borderline-SMOTE/ADASYN/SPIDER2/SMOTEBoost/MWMOTE) and planned technique (UCPMOT). Tables 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 and 13 describe the results in detail. The analysis of results validates the superiority of UCPMOT for enhancing the classification.

# GOOGLE ON: scikit learn imbalanced dataset resampling type cross validation shuffle
# https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets
# https://machinelearningmastery.com/cross-validation-for-imbalanced-classification/
# CV = ShuffleSplit(n_splits=10, test_size=0.25, random_state=48)
# https://www.alfredo.motta.name/cross-validation-done-wrong/

# https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/
#     https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6

# http://www.cs.nthu.edu.tw/~shwu/courses/ml/labs/08_CV_Ensembling/08_CV_Ensembling.html
# https://github.com/arrayslayer/ML-Project

# https://www.kaggle.com/shiqbal/first-data-exploration/notebook  applied on Porto Seguro's


''' TODO :

-> Faire Ressortir l'importance et la contribution de chaque feature:
   https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance.html

-> Essayer ces scénarios de modélisation:
   scen1: Oversampling + LogisticR
   scen2: BaggingClassifier + Histo

'''
```

```python
from imblearn.over_sampling import RandomOverSampler, SMOTE, SMOTENC, SVMSMOTE, KMeansSMOTE, BorderlineSMOTE, ADASYN
from imblearn.over_sampling.base import BaseOverSampler
# from sklearn.experimental import enable_iterative_imputer   # explicitly require this experimental feature
# from sklearn.impute import IterativeImputer
from sklearn.impute import SimpleImputer  # now you can import normally from sklearn.impute
from sklearn.linear_model import BayesianRidge
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler, StandardScaler, MinMaxScaler

from sklearn.compose import ColumnTransformer, make_column_transformer
from imblearn.pipeline import make_pipeline, Pipeline

import pandas as pd
import numpy as np

from valeo.infrastructure import Const as C
from valeo.infrastructure.LogManager import LogManager

from valeo.infrastructure.tools.DebugPipeline import DebugPipeline


class ValeoPreprocessor:
    logger = None

    def __init__(self):
        ValeoPreprocessor.logger = LogManager.logger(__name__)

    # def build_iterative_preprocessor(self)  -> ColumnTransformer:
    #     imp_cols = [C.OP100_Capuchon_insertion_mesure]
    #     it_imp = IterativeImputer(estimator=BayesianRidge(),  max_iter=10, initial_strategy = 'median', add_indicator=True, random_state=48)
    #     return ColumnTransformer([('iterative_imputer', it_imp, imp_cols)] )

    def build_column_preprocessor(self) -> Pipeline: # ColumnTransformer:
        rand_state = 48
        # 1 - IterativeImputer : models each feature with missing values as a function of other features, and uses that estimate for imputation
        # imputer_pipe = Pipeline( IterativeImputer(estimator=BayesianRidge(), missing_values=[np.nan, 0], max_iter=10, initial_strategy = 'median') )

        # imputer_nan_values = IterativeImputer(estimator=BayesianRidge(), missing_values=np.nan,  max_iter=10, initial_strategy = 'median',  add_indicator=True, random_state=rand_state)
        # imputer_nan_values = IterativeImputer(estimator=BayesianRidge(), missing_values=np.nan,  max_iter=10, initial_strategy = 'median',  add_indicator=False, random_state=rand_state)
        imputed_nan_cols = [C.OP100_Capuchon_insertion_mesure]  # columns having too much missing values
        #
        # imputer_zeroes_values = IterativeImputer(estimator=BayesianRidge(), missing_values=0,  max_iter=10, initial_strategy = 'median',  add_indicator=False, random_state=rand_state)
        imputed_zeroes_cols = [C.OP100_Capuchon_insertion_mesure, C.OP090_StartLinePeakForce_value, C.OP090_SnapRingMidPointForce_val,  # columns equals to 0 for a few rows
                               C.OP090_SnapRingPeakForce_value, C.OP090_SnapRingFinalStroke_value]

        # 2 - Scale features using statistics that are robust to outliers.
        scaler      = StandardScaler() #  MinMaxScaler() # StandardScaler() # RobustScaler(with_centering=True, with_scaling=False)
        scaled_cols = [C.OP070_V_1_angle_value, C.OP070_V_1_torque_value,
                       C.OP070_V_2_angle_value, C.OP070_V_2_torque_value,
                       C.OP090_StartLinePeakForce_value, C.OP090_SnapRingMidPointForce_val,
                       C.OP090_SnapRingPeakForce_value,  C.OP090_SnapRingFinalStroke_value,
                       C.OP100_Capuchon_insertion_mesure,
                       C.OP110_Vissage_M8_angle_value, C.OP110_Vissage_M8_torque_value,
                       C.OP120_Rodage_I_mesure_value,  C.OP120_Rodage_U_mesure_value]

        dbg = DebugPipeline()
        return  Pipeline([ ('dbg_0', dbg, scaled_cols),
                                   ('imputer_preprocessor_nan', SimpleImputer(strategy='mean', missing_values=np.nan, verbose=True), imputed_nan_cols),
                                   ('dbg_1', dbg, scaled_cols),
                                   ('imputer_preprocessor_zeroes', SimpleImputer(strategy='mean', missing_values=0.0, verbose=True), imputed_zeroes_cols),
                                   ('dbg_2', dbg, scaled_cols),
                                   ('scaler_preprocessor', scaler, scaled_cols),
                                   ('dbg_3', dbg, scaled_cols),
                                   # #
                                   # ('imputer_preprocessor_nan_bis', SimpleImputer(strategy='mean'), scaled_cols),
                                   # ('dbg_1_bis', DebugPline(),scaled_cols)
                                   ])
                         # , remainder='passthrough')


    def execute(self, X_train:pd.DataFrame):
        imputed_nan_cols = [C.OP100_Capuchon_insertion_mesure]
        imputed_zeroes_cols = [C.OP100_Capuchon_insertion_mesure, C.OP090_StartLinePeakForce_value, C.OP090_SnapRingMidPointForce_val,  # columns equals to 0 for a few rows
                               C.OP090_SnapRingPeakForce_value, C.OP090_SnapRingFinalStroke_value]
        scaled_cols = [C.OP070_V_1_angle_value, C.OP070_V_1_torque_value,
                        C.OP070_V_2_angle_value, C.OP070_V_2_torque_value,
                        C.OP090_StartLinePeakForce_value, C.OP090_SnapRingMidPointForce_val,
                        C.OP090_SnapRingPeakForce_value,  C.OP090_SnapRingFinalStroke_value,
                        C.OP100_Capuchon_insertion_mesure,
                        C.OP110_Vissage_M8_angle_value, C.OP110_Vissage_M8_torque_value,
                        C.OP120_Rodage_I_mesure_value,  C.OP120_Rodage_U_mesure_value]
         # DebugPipeline.counter += 10
         d = DebugPipeline()
         #
         d.fit_transform(X_train[scaled_cols])
         s = SimpleImputer(strategy='mean', missing_values=np.nan, verbose=True)
         X_train[imputed_nan_cols] = s.fit_transform(X_train[imputed_nan_cols])
         d.fit_transform(X_train[scaled_cols])
         #
         s = SimpleImputer(strategy='mean', missing_values=0.0, verbose=True)
         X_train[imputed_zeroes_cols] = s.fit_transform(X_train[imputed_zeroes_cols])
         d.fit_transform(X_train[scaled_cols])
         #
         scaler      = MinMaxScaler() # StandardScaler() # RobustScaler(with_centering=True, with_scaling=False)
         X_train[scaled_cols]  = scaler.fit_transform(X_train[scaled_cols])
         d.fit_transform(X_train[scaled_cols])



     '''
     def build_column_preprocessor(self) -> ColumnTransformer:
         rand_state = 48
         # 1 - IterativeImputer : models each feature with missing values as a function of other features, and uses that estimate for imputation
         imputer_pipe = make_pipeline( IterativeImputer(estimator=BayesianRidge(), missing_values=np.nan, max_iter=10, initial_strategy = 'median',  add_indicator=True, random_state=rand_state) )
         # imputer_pipe = Pipeline( IterativeImputer(estimator=BayesianRidge(), missing_values=[np.nan, 0], max_iter=10, initial_strategy = 'median') )
         imputed_cols = [C.OP100_Capuchon_insertion_mesure]  # columns having too much missing values
                         # C.OP090_StartLinePeakForce_value, C.OP090_SnapRingMidPointForce_val,  # columns equals to 0 for a few rows
                         # C.OP090_SnapRingPeakForce_value, C.OP090_SnapRingFinalStroke_value]

         # 2 - Scale features using statistics that are robust to outliers.
         scaler_pipe = make_pipeline(RobustScaler(with_centering=True, with_scaling=False))
         scaled_cols = [C.OP070_V_1_angle_value, C.OP070_V_1_torque_value,
                        C.OP070_V_2_angle_value, C.OP070_V_2_torque_value,
                        C.OP090_StartLinePeakForce_value, C.OP090_SnapRingMidPointForce_val,
                        C.OP090_SnapRingPeakForce_value,  C.OP090_SnapRingFinalStroke_value,
                        C.OP100_Capuchon_insertion_mesure,
                        C.OP110_Vissage_M8_angle_value, C.OP110_Vissage_M8_torque_value,
                        C.OP120_Rodage_I_mesure_value,  C.OP120_Rodage_U_mesure_value,]

         return  ColumnTransformer([('imputer_preprocessor', imputer_pipe, imputed_cols),
                                    ('scaler_preprocessor', scaler_pipe, scaled_cols)] )
     '''

     '''
     SMOTe is a technique based on nearest neighbors judged by Euclidean Distance between data points in feature space.
     random_over_sampling : The most naive strategy is to generate new samples by randomly sampling with replacement the current available samples.
     adasyn_over_sampling : Adaptive Synthetic: focuses on generating samples next to the original samples which are wrongly classified using a k-Nearest Neighbors classifier
     smote_over_sampling  : Synth Minority Oversampling Techn: will not make any distinction between easy and hard samples to be classified using the nearest neighbors rule
     ---
     https://medium.com/towards-artificial-intelligence/application-of-synthetic-minority-over-sampling-technique-smote-for-imbalanced-data-sets-509ab55cfdaf
     https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html
     NB:
     How to apply SMOTE : Shuffling and Splitting the Dataset into Training and Validation Sets and THEN applying SMOTe on the Training Dataset.
     '''
     def build_resampler(self, sampler_type: str, sampling_strategy='auto', k_neighbors=5) -> BaseOverSampler :
         rand_state = 48
         if sampler_type.lower() == C.random_over_sampler :
             return RandomOverSampler(sampling_strategy=sampling_strategy, random_state=rand_state)
         elif sampler_type.lower() == C.adasyn_over_sampling :
             return ADASYN(sampling_strategy=sampling_strategy, random_state=rand_state, n_neighbors=k_neighbors)
         elif sampler_type.lower() == C.smote_over_sampling :
             return SMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=k_neighbors)
         # elif sampler_type.lower() == C.smote_nc_over_sampling :      # SMOTE for dataset containing continuous and categorical features.
         #      return SMOTENC(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=k_neighbors)
         elif sampler_type.lower() == C.smote_svm_over_sampling :    # Use an SVM algorithm to detect sample to use for generating new synthetic samples
             return SVMSMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=k_neighbors)
         elif sampler_type.lower() == C.smote_kmeans_over_sampling : # Apply a KMeans clustering before to over-sample using SMOTE
             return KMeansSMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=
```

```python
k_neighbors)
        elif sampler_type.lower() == C.smote_bline_over_sampling :  # Borderline samples will be detected and used to generate new synthetic samples.
            return BorderlineSMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=k_neighbors)
        else :
            raise ValueError(f"Unexpected argument [sampler_type:{sampler_type}] for method 'compute_sampler_preprocessor'")



# ---------------------------------
# Exemple Type : PipeLine entier
# ---------------------------------
# >>> pca = PCA()
# >>> smt = SMOTE(random_state=42)
# >>> knn = KNN()
# >>> pipeline = Pipeline([('smt', smt), ('pca', pca), ('knn', knn)])



# ----------------------
# Exemple_1
# ----------------------
# from sklearn.compose import ColumnTransformer
# from sklearn.pipeline import Pipeline
#
# # 1 - Define Categorical pipe_line
# cat_col = ['sex', 'embarked', 'pclass']
# cat_pipeline = Pipeline(steps=[
#     ("constant-imputer", SimpleImputer(strategy='constant', fill_value='missing')),
#     ("ordinal-encoder", OrdinalEncoder()),
# ])
#
# # 2 - Define Numerical pipe_line
# num_cols = ['age', 'parch', 'fare']
# num_pipeline = SimpleImputer(
#     strategy="mean", add_indicator=True,
# )
#
# # 3 - Define Column Transformer
# preprocessor = ColumnTransformer(transformers=[
#     ("cat-preprocessor", cat_pipeline, cat_col),
#     ("num-preprocessor", num_pipeline, num_cols),
# ])
#
# model = Pipeline(steps=[
#     ("preprocessor", preprocessor),
#     ("clf", RandomForestClassifier(n_estimators=100))
# ])
#
# _ = model.fit(X_train, y_train)
#
# (model.named_steps["preprocessor"]
#  .named_transformers_["cat-preprocessor"]
#  .named_steps["ordinal-encoder"].categories_)



# ----------------------
# Exemple_2
# ----------------------
# define the pipelines
# cat_pipe = make_pipeline(
#     SimpleImputer(strategy='constant', fill_value='missing'),
#     OrdinalEncoder(categories=categories)
# )
# num_pipe = SimpleImputer(strategy='mean')
#
# preprocessing = ColumnTransformer(
#     [('cat_preprocessor', cat_pipe, cat_col),
#      ('num_preprocessor', num_pipe, num_cols)]
# )
```

File - C:\EXED\Training\___VALEO\src\valeo\resources\valeo.yaml

```yaml
version: 1
data:
   input: ../data/input
   output: ../data/output

```

File - C:\EXED\Training\___VALEO\src\valeo\resources\logging.yaml

```yaml
version: 1
formatters:
  standard:
    format: '%(asctime)s - %(levelname)s - %(name)s - %(message)s'
#    format: '%(asctime)s - %(levelname)s - %(module)s - %(message)s'
  verbose:
    format: '%(asctime)s - %(levelname)s <PID %(process)d:%(processName)s> %(module)s.%(funcName)s(): %(message)s
'
handlers:
  console:
    class: logging.StreamHandler
    level: DEBUG
    formatter: standard
    stream: ext://sys.stdout
#    propagate: yes
  valeo_log:
    class: logging.FileHandler
    filename: C:/EXED/Training/___VALEO/log/valeo.log
    mode: w
    level: DEBUG
    formatter: standard
#    propagate: yes

#  errors:
#    class: logging.FileHandler
#    filename: mplog-errors.log
#    mode: w
#    level: ERROR
#    formatter: detailed
#loggers:
#  simpleExample:
#    level: DEBUG
#    handlers: [console]
#    propagate: no
root:
  level: DEBUG
  handlers: [console, valeo_log]
  formatter: standard
```

File - C:\EXED\Training\___VALEO\src\valeo\infrastructure\Const.py

```python
# ENVIRONMENT keys used to refer to configuration files
ENV_KEY_CONFIG_FILE_PATHNAME = '__VALEO__APP_CONFIG_FILE_PATHNAME' # ex: SET __VALEO__APP_CONFIG_FILE_PATHNAME =...../valeo.yaml'
ENV_KEY_LOG_FILE_PATHNAME    = '__VALEO__APP_LOG_FILE_PATHNAME'    # ex: SET __VALEO__APP_LOG_FILE_PATHNAME =...../logging.yaml'
#
# Symbolic name of configuration files
APP_DEFAULT_CONFIG_FILE = 'valeo.yaml'
APP_DEFAULT_LOG_FILE    = 'logging.yaml'
#
# Valeo Dataset columns names
PROC_TRACEINFO         = 'PROC_TRACEINFO'
OP070_V_1_angle_value  = 'OP070_V_1_angle_value'
OP070_V_1_torque_value = 'OP070_V_1_torque_value'
OP070_V_2_angle_value  = 'OP070_V_2_angle_value'
OP070_V_2_torque_value = 'OP070_V_2_torque_value'
OP090_StartLinePeakForce_value = 'OP090_StartLinePeakForce_value'
OP090_SnapRingMidPointForce_val = 'OP090_SnapRingMidPointForce_val'
OP090_SnapRingPeakForce_value     = 'OP090_SnapRingPeakForce_value'
OP090_SnapRingFinalStroke_value   = 'OP090_SnapRingFinalStroke_value'
OP100_Capuchon_insertion_mesure   = 'OP100_Capuchon_insertion_mesure'
OP110_Vissage_M8_angle_value  = 'OP110_Vissage_M8_angle_value'
OP110_Vissage_M8_torque_value = 'OP110_Vissage_M8_torque_value'
OP120_Rodage_I_mesure_value   = 'OP120_Rodage_I_mesure_value'
OP120_Rodage_U_mesure_value   = 'OP120_Rodage_U_mesure_value'
Binar_OP130_Resultat_Global_v = 'Binar OP130_Resultat_Global_v'
#
# Imbalanced resampling type

random_over_sampler = 'random_over_sampler' # The most naive strategy is to generate new samples by randomly sampling with replacement the current available samples.
adasyn_over_sampling = 'adasyn_over_sampling' # Adaptive Synthetic: focuses on generating samples next to the original samples which are wrongly classified using a k-Nearest Neighbors classifier
smote_over_sampling  = 'smote_over_sampling'  # Synth Minority Oversampling Techn: will not make any distinction  between easy and hard samples to be classified using the nearest neighbors rule
smote_nc_over_sampling   = 'smote_nc_over_sampling'
smote_svm_over_sampling  = 'smote_svm_over_sampling'
smote_kmeans_over_sampling  = 'smote_kmeans_over_sampling'
smote_bline_over_sampling   = 'smote_bline_over_sampling'


import os
from datetime import datetime
# timestamp : none / suffix / prefix
ts_none = 0
ts_sfix = 1
ts_pfix = 2

def rootProject() -> str :
    return  os.path.join(os.path.abspath(os.path.dirname(__file__)), '..', '..', '..')  # this_folder = D:/Training.git/trunk/___VALEO/src/valeo/infrastructure

def rootSrc() -> str :
    return  os.path.join(rootProject(),  'src' )

def rootData() -> str :
    return  os.path.join(rootProject(),  'data' )

def rootDataTrain() -> str :
    return  os.path.join(rootData(),  'train' )

def rootDataTest() -> str :
    return  os.path.join(rootData(),  'test' )

def rootImages() -> str :
    return  os.path.join(rootProject(),  'images' )

def rootReports() -> str :
    return  os.path.join(rootProject(),  'reports' )

def rootResources() -> str :
    return  os.path.join(rootProject(), 'src', 'valeo', 'resources')

def ts_pathanme(pathAsStrList : [], ts_type=ts_sfix) -> str:
    if not isinstance(pathAsStrList,list) :
        pathAsStrList = [pathAsStrList]
    fname_with_ext = os.path.splitext(pathAsStrList[-1])
    return os.path.join(pathAsStrList[0], '' if len(pathAsStrList) <= 2 else str(*pathAsStrList[1:-1] ),
              f"{fname_with_ext[0]}{datetime.now().strftime('_%Y_%m_%d-%H.%M.%S')}{fname_with_ext[1]}" if ts_type == ts_sfix else \
             (f"{datetime.now().strftime('%Y_%m_%d-%H.%M.%S_')}{pathAsStrList[-1]}" if ts_type == ts_pfix  else pathAsStrList[-1]) )

```

```python
import os

import pandas as pd
import numpy as np
from sklearn.model_selection import ShuffleSplit

import valeo.infrastructure.XY_metadata as XY_metadata
from valeo.infrastructure import Const
from valeo.infrastructure.LogManager import LogManager
from valeo.infrastructure.tools.DfUtil import DfUtil


class XY_Loader:
    logger = None

    def __init__(self):
        XY_Loader.logger = LogManager.logger(__name__)

    def get_cv(X, y):
        cv = ShuffleSplit(n_splits=8, test_size=0.5, random_state=57)
        return cv.split(X)


    def load_XY_df(self, mt: XY_metadata, delete_XY_join_cols=True) -> ():
        # X_df = pd.read_csv(mt.X_pathname, na_values='')  # NaN
        X_df = pd.read_csv(mt.X_pathname)  # NaN
        # print(X_df[Const.OP100_Capuchon_insertion_mesure].head(20))
        # X_df[[Const.OP100_Capuchon_insertion_mesure]] = X_df[[Const.OP100_Capuchon_insertion_mesure]].fillna(0
.0)
        # print(X_df[Const.OP100_Capuchon_insertion_mesure].head(20))

        # 1 - Check whether Y is in separate file or in the same as X
        if mt.is_XY_in_separate_file() :
            Y_df = pd.read_csv(mt.Y_pathname)
            XY_df = pd.merge(left=X_df, right=Y_df, how='inner', left_on=mt.X_join, right_on=mt.Y_join, suffixes
=('',''))
        else :
            Y_df = None
            XY_df = X_df

        # 2 - When not reading a Test dataset (it means there is a Target dataset) THEN Let X_df group only
features and Y_df only target
        if mt.is_training_set() :
            Y_df = XY_df[mt.target_col_name]
            X_df = XY_df.drop(mt.target_col_name, axis=1)

        # 3 - Check whether we should remove joining columns
        if delete_XY_join_cols :
            X_df = X_df.drop(mt.X_join, axis=1)
            try :
                X_df = X_df.drop(mt.Y_join, axis=1)
            except :
                pass

        #
        # XY_Loader.logger.debug(f'X_df.columns: {X_df.columns}')
        # if Y_df is not None :
        #     XY_Loader.logger.debug(f'type(Y_df):{type(Y_df)}\nY_df: {Y_df}')
        return X_df, Y_df

    def load_XY_values(self, mt: XY_metadata, delete_XY_join_cols=True) -> ():
        X_df, Y_df = self.load_XY_df(mt, delete_XY_join_cols)
        return X_df.values if X_df is not None else None, \
               Y_df.values if Y_df is not None else None

```

File - C:\EXED\Training\___VALEO\src\valeo\infrastructure\LogManager.py

```python
# https://docs.python.org/3/library/logging.html#logrecord-attributes + Useful Handlers
# https://docs.python-guide.org/writing/logging/
# https://github.com/Delgan/loguru
# https://kingspp.github.io/design/2017/11/06/the-head-and-tail-of-logging.html
# https://stackoverflow.com/questions/4690600/python-exception-message-capturing
import logging.config
import os

from valeo.infrastructure.tools.ConfigLoader import ConfigLoader
import valeo.infrastructure.Const as Const

class LogManager():

    # NB: The ctor() initializes the logging configuration
    def __init__(self):
        self.log_config = LogLoader().load()

    @classmethod
    def logger(cls,logname):
         # l = logging.getLogger(logname)
         # l.
        return logging.getLogger(logname)


class LogLoader(ConfigLoader):
    """
    Load the logging configuration file
    """
    def load(self) -> dict:
        try :
            dict = super().load(os.path.join(Const.rootResources(), Const.APP_DEFAULT_LOG_FILE), Const.
ENV_KEY_LOG_FILE_PATHNAME)
            logging.config.dictConfig(dict)
            return dict
        except Exception as ex:
            logging.basicConfig(level=logging.INFO)
            logging.warning(f'Error while loading logging configuration file:\n' \
                            f'\t- APP_RESOURCE_PATH = {Const.rootResources()}\n' \
                            f'\t- APP_DEFAULT_LOG_FILE = {Const.APP_DEFAULT_LOG_FILE}\n' \
                            f'\t- ENV_KEY_LOG_FILE_PATHNAME = {Const.ENV_KEY_LOG_FILE_PATHNAME}')
            logging.exception(ex)
            return None

```

File - C:\EXED\Training\___VALEO\src\valeo\infrastructure\Transformer.py

```python
# explicitly require this experimental feature
from sklearn.experimental import enable_iterative_imputer
# now you can import normally from sklearn.impute
from sklearn.impute import IterativeImputer
from sklearn.linear_model import BayesianRidge

import pandas as pd
import numpy as np
from sklearn.preprocessing import RobustScaler

from valeo.infrastructure.LogManager import LogManager


class Transformer() :
    logger = LogManager.logger(__name__)

    # def __init__(self):
    #     lm = LogManager()
    #     self.logger = lm.logger(__name__)

    '''
    A strategy for imputing missing values by modeling each feature with missing values as a function of other features in a round-robin fashion.
    Multivariate imputer that estimates each feature from all the others.
    https://scikit-learn.org/stable/modules/impute.html#iterative-imputer

    Arg:
    ----
    estimator : The estimator to use at each step of the round-robin imputation.

    Returns:
    --------
    A transformed Dataframe containing all the missing values.
    NB: The arguement Dataframe is NOT modified => It stills intact
    https://towardsdatascience.com/introduction-to-bayesian-linear-regression-e66e60791ea7
    '''
    def iterative_imputer_transform(self, df_to_transform : pd.DataFrame,  estimator=BayesianRidge(), missing_values=np.nan,  max_iter=10, initial_strategy = 'median') -> pd.DataFrame :
        cols = df_to_transform.columns
        imputer = IterativeImputer(estimator=estimator, missing_values=missing_values,  max_iter=max_iter, initial_strategy=initial_strategy,  add_indicator=False) # It models each feature with missing values as a function of other features, and uses that estimate for imputation
        df_transformed = pd.DataFrame(imputer.fit_transform(df_to_transform))
        # df_transformed.columns = df_transformed.columns[:-1]
        df_transformed.columns = cols
        return df_transformed

    def robust_scaler_transform(self, df_to_transform : pd.DataFrame, with_centering=True, with_scaling=True, quantile_range=(5.0, 95.0)):
        cols = df_to_transform.columns
        scaler  =  RobustScaler(with_centering=with_centering, with_scaling=with_scaling, quantile_range= quantile_range)
        df_transformed = pd.DataFrame(scaler.fit_transform(df_to_transform))
        df_transformed.columns = cols
        return df_transformed
```

File - C:\EXED\Training\___VALEO\src\valeo\infrastructure\XY_metadata.py

```python
import os


class XY_metadata :

    def __init__(self,  X_pathname :[], Y_pathname :[], X_join:[], Y_join:[], target_col_name:str):
        self.X_pathname = os.path.join(X_pathname[0], *X_pathname[1:])
        self.Y_pathname = None if Y_pathname is None else os.path.join(Y_pathname[0], *Y_pathname[1:])
        self.X_join = X_join
        self.Y_join = Y_join
        self.target_col_name = target_col_name

    def is_training_set(self) -> bool :
        return True if self.target_col_name is not None else False

    def is_XY_in_separate_file(self) -> bool:
        return True if self.Y_pathname is not None else False
```

```python
from sklearn.impute import SimpleImputer as _SimpleImputer

from valeo.infrastructure.tools.DfInDfOut import DfInDfOut


class SimpleImputer(_SimpleImputer, DfInDfOut):

    def transform(self, X):
        Xt = super().transform(X)
        return super().check_output(Xt, ensure_index=X, ensure_columns=X)
```

```python
from sklearn.preprocessing import StandardScaler as _StandardScaler

from valeo.infrastructure.tools.DfInDfOut import DfInDfOut


class StandardScaler(_StandardScaler, DfInDfOut):

    def transform(self, X):
        Xt = super().transform(X)
        return super().check_output(Xt, ensure_index=X, ensure_columns=X)
```

```python
import valeo.infrastructure.Const as const
from valeo.infrastructure.tools.ConfigLoader import ConfigLoader

class AppConfigManager():

    def __init__(self):
        cl = AppConfigLoader()
        self.app_config = cl.load()

    def getValue(self, nested_dict:{}, keys:[]) -> str :
        return nested_dict[keys[0]] if len(keys) ==  1 else self.getValue(nested_dict[keys[0]] , keys[1:])



class AppConfigLoader(ConfigLoader) :

    def load(self) -> dict:
        return super().load(f'{const.rootResources}{const.APP_DEFAULT_CONFIG_FILE}', const.
ENV_KEY_CONFIG_FILE_PATHNAME)
```

```python
from datetime import datetime

from pandas import Series
from sklearn.base import BaseEstimator

from valeo.infrastructure.LogManager import LogManager
from valeo.infrastructure import Const as C

import os
import pandas as pd
import numpy as np

class DfUtil() :
    logger = LogManager.logger(__name__)


    # https://stackabuse.com/pythons-classmethod-and-staticmethod-explained/
    @classmethod
    def read_csv(cls, pathAsStrList : []) -> pd.DataFrame:
        try:
            return pd.read_csv(os.path.join(pathAsStrList[0], *pathAsStrList[1:]) )
        except Exception as ex :
            cls.logger.exception("Error while load data from %s", "/".join(pathAsStrList))

    @classmethod
    def write_y_csv(cls, X_id:Series, y_target: np.ndarray, y_col_name:str, pathAsStrList : [], ts_type=C.
ts_sfix):
        DfUtil.write_df_csv( pd.DataFrame(data={X_id.name:X_id, y_col_name:y_target}),  pathAsStrList, ts_type=
ts_type)

    @classmethod
    def write_df_csv(cls, df:pd.DataFrame, pathAsStrList : [], ts_type=C.ts_sfix):
        try :
            df.to_csv( C.ts_pathanme(pathAsStrList,ts_type), index = False)
        except Exception as ex:
            cls.logger = LogManager.logger("DfUtil")
            cls.logger.exception(f"Error while writing 'df' to CSV '{pathAsStrList}'")

    @classmethod
    def df_imputer(cls, dfToImpute:pd.DataFrame, imputer:BaseEstimator):
        '''This method encodes non-null data and replace it in the original data'''
        # Retains only non-null values. dropna: Remove [rows(default) OR columns] when missing values
        nonulls = np.array(dfToImpute.dropna())
        # Reshapes the data for encoding
        impute_reshape = nonulls.reshape(-1,1)
        #     #encode date
        #     impute_ordinal = imputer.fit_transform(impute_reshape)
        # Assign back encoded values to non-null values
        dfToImpute.loc[dfToImpute.notnull()] = np.squeeze(imputer.fit_transform(impute_reshape))  # np.squeeze:
Remove single-dimensional entries from the shape of an array.
        return dfToImpute

    @classmethod
    def outlier_ratio(cls, df:pd.DataFrame) -> float:
        Q1 = df.quantile(0.25)
        Q3 = df.quantile(0.75)
        IQR = Q3 - Q1
        #
        outliers = ((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)
        return len(df[outliers].index)/len(df.index)

```

File - C:\EXED\Training\___VALEO\src\valeo\infrastructure\tools\ImgUtil.py

```python
from valeo.infrastructure import Const as C
from valeo.infrastructure.LogManager import LogManager
import os
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

class ImgUtil() :
    logger = LogManager.logger(__name__)

    # https://stackabuse.com/pythons-classmethod-and-staticmethod-explained/
    @classmethod
    def save_fig(cls, fig_id:str , tight_layout=True, fig_extension="png", resolution=300, ts_type=C.ts_sfix):
        path = C.ts_pathanme([C.rootImages() , fig_id + "." + fig_extension], ts_type=ts_type)
        # cls.logger.debug(f"Saving figure '{fig_id}'")
        if tight_layout:
            plt.tight_layout()
        # Save "the current figure plot" that is set by "df.hist(...))". @ReferTo: pyplot.py / def gcf()
        plt.savefig(path, format=fig_extension, dpi=resolution)

    @classmethod
    def save_df_hist_plot(cls, df:pd.DataFrame, fig_id:str , bins=50, figsize=(20,15), tight_layout=True,
                          fig_extension="png", resolution=300, ts_type=C.ts_sfix):
        cls.logger.debug(f"Generating 'hist' plot: bins={bins} - figsize={figsize}")
        df.hist(bins=bins, figsize=figsize)
        cls.save_fig(fig_id=f"{fig_id}_histogram_{figsize[0]}x{figsize[1]}", tight_layout=tight_layout, fig_extension=fig_extension, resolution=resolution, ts_type=ts_type)

    @classmethod
    def save_df_XY_hist_plot(cls, df_XY:pd.DataFrame, fig_id:str, bins=50, figsize=(5, 5), y_target_name=None, tight_layout=True,
                             fig_extension="png", resolution=300, ts_type=C.ts_sfix):
        cls.logger.debug(f"Generating 'XY_hist' plot: bins={bins} - figsize={figsize}")
        df_X = df_XY.drop(columns=y_target_name, axis=1)
        y    = df_XY[y_target_name]
        fig, ax = plt.subplots(figsize=figsize)
        for i, col in enumerate(sorted(df_X.columns)) :
            for clazz in y.unique() :
                df_X[y==clazz][col].plot.hist(bins=bins, figsize=figsize, alpha=0.3, label=f'Class #{int(clazz)}')
            plt.legend()
            plt.xlabel(col)
            ImgUtil.save_fig(fig_id=f"{fig_id}_{col}_histogram_{figsize[0]}x{figsize[1]}", tight_layout=tight_layout, fig_extension=fig_extension, resolution=resolution, ts_type=ts_type)
            ax.clear()
    # NB:
    # df.hist: => Plot 1 Histo per dfColumn
    # df.plot.hist: => Plot all df-referenced-Columns on same Histo

    @classmethod
    def save_df_scatter_matrix_plot(cls, df:pd.DataFrame, fig_id:str , figsize=(20,15), cfield=None, tight_layout=True,
                                    fig_extension="png", resolution=300, ts_type=C.ts_sfix):
        cls.logger.debug(f"Generating 'scatter matrix' plot: figsize:{figsize}")
        if cfield == None :
            pd.plotting.scatter_matrix(df, figsize=figsize)
        else :
            pd.plotting.scatter_matrix(df, figsize=figsize,  alpha=0.3, c=df[cfield].values, cmap='RdBu')
        cls.save_fig(fig_id=f"{fig_id}_scatter_matrix_{figsize[0]}x{figsize[1]}", tight_layout=tight_layout, fig_extension=fig_extension, resolution=resolution, ts_type=ts_type)

    @classmethod
    def save_df_heatmap_plot(cls, df:pd.DataFrame, fig_id:str , figsize=(20,20), cmap='RdBu', annot=True , annot_kws={'size':15}, fig_extension="png", resolution=300, ts_type=C.ts_sfix):
        cls.logger.debug(f"Generating 'heatmap' plot: figsize:{figsize}")
        fig, ax = plt.subplots(figsize=figsize)
        sns.set(font_scale=1.1)
        sns.heatmap(df, cmap=cmap, annot=annot , annot_kws=annot_kws, ax=ax)
        ax.set_title(fig_id, fontsize=28)
        cls.save_fig(fig_id=f"{fig_id.replace(' ','_')}_heatmap_{figsize[0]}x{figsize[1]}", tight_layout=True, fig_extension=fig_extension, resolution=resolution, ts_type=ts_type)

    @classmethod
    def save_df_violin_plot(cls, df:pd.DataFrame, fig_id:str, grid_elmt_x:int, figsize=(20,20), fig_extension="png", resolution=300, ts_type=C.ts_sfix):
        cls.logger.debug(f"Generating 'violin' plot: figsize:{figsize}")
        grid_elmt_y = len(df.columns) // grid_elmt_x if (len(df.columns) % grid_elmt_x) == 0 else (len(df.columns) // grid_elmt_x) + 1
        #
        fig, axs = plt.subplots(grid_elmt_y, grid_elmt_x, figsize=figsize)
        for i, col in enumerate(sorted(df.columns)) :
            sns.violinplot(x=df[col], linewidth=1, ax=axs[i//grid_elmt_x, i%grid_elmt_x])
            # sns.stripplot( x=df[col], color="orange", jitter=0.2, linewidth=1, ax=axs[i//3,i%3])
            sns.boxplot( x=df[col], linewidth=1, ax=axs[i//grid_elmt_x, i%grid_elmt_x], saturation=0 )
        # axs.set_title(fig_id, fontsize=28)
        cls.save_fig(fig_id=f"{fig_id.replace(' ','_')}_violin_{figsize[0]}x{figsize[1]}", tight_layout=True, fig_extension=fig_extension, resolution=resolution, ts_type=ts_type)

```

```python

def save_df_XY_violin_plot(df_XY:pd.DataFrame, y_target_name:str, fig_id:str, grid_elmt_x:int, figsize=(20,20), fig_extension="png", resolution=300, ts_type=Const.ts_sfix):
    df = df_XY.drop(columns=y_target_name, axis=1)
    grid_elmt_y = len(df.columns) // grid_elmt_x if (len(df.columns) % grid_elmt_x) == 0 else (len(df.columns) // grid_elmt_x) + 1
    #
    fig, axs = plt.subplots(grid_elmt_y, grid_elmt_x, figsize=figsize)
    for i, col in enumerate(sorted(df.columns)) :
        sns.violinplot(x=y_target_name, y=col, data=df_XY, linewidth=1, ax=axs[i//grid_elmt_x, i%grid_elmt_x])
        sns.boxplot   (x=y_target_name, y=col, data=df_XY, linewidth=1, ax=axs[i//grid_elmt_x, i%grid_elmt_x] )
    ImgUtil.save_fig(fig_id=f"{fig_id.replace(' ','_')}_violin_{figsize[0]}x{figsize[1]}", tight_layout=True, fig_extension=fig_extension, resolution=resolution, ts_type=ts_type)


    @classmethod
    # SWARM PLOT did not work correctly
    def save_df_swarm_plot(cls, df:pd.DataFrame, fig_id:str, grid_elmt_x:int, figsize=(20,20), cfield=None, fig_extension="png", resolution=300, ts_type=C.ts_sfix):
        cls.logger.debug(f"Generating 'swarm' plot: figsize:{figsize}")
        grid_elmt_y = len(df.columns) // grid_elmt_x if (len(df.columns) % grid_elmt_x) == 0 else (len(df.columns) // grid_elmt_x) + 1
        #
        fig, axs = plt.subplots(grid_elmt_y, grid_elmt_x, figsize=figsize)
        for i, col in enumerate(sorted(df.columns)) :
            sns.swarmplot(x=df[col], linewidth=1, ax=axs[i//grid_elmt_x, i%grid_elmt_x], hue=df[cfield].values)
        cls.save_fig(fig_id=f"{fig_id.replace(' ','_')}_swarm_{figsize[0]}x{figsize[1]}", tight_layout=True, fig_extension=fig_extension, resolution=resolution, ts_type=ts_type)

    # def save_df_swarm_plot(df_XY:pd.DataFrame, fig_id:str, figsize=(5,5), y_target_name=None, fig_extension="png", resolution=300, ts_type=Const.ts_sfix):
    #     df_X = df_XY.drop(columns=y_target_name, axis=1)
    #     y    = df_XY[y_target_name]
    #     fig, ax = plt.subplots(figsize=figsize)
    #     for i, col in enumerate(sorted(df_X.columns)) :
    #         for clazz in y.unique() :
    #             sns.swarmplot(x=col, hue=y_target_name, data=df_XY[y==clazz])
    #             # sns.swarmplot(x='data', y='feature',  hue='label', data=df)
    #         plt.legend()
    #         plt.xlabel(col)
    #         ImgUtil.save_fig(fig_id=f"{fig_id}_{col}_swarm_{figsize[0]}x{figsize[1]}", tight_layout=tight_layout, fig_extension=fig_extension, resolution=resolution, ts_type=ts_type)
    #         ax.clear()


# NB: Generic way to plot whathever :
# fig, ax = plt.subplots(figsize=(20,20))
# sns.heatmap(corr_matrix, cmap='RdBu', annot=True , annot_kws={'size':15}, ax=ax)
# ax.set_title("Valeo starter production correlation measures", fontsize=14)
# plt.show()


```

File - C:\EXED\Training\___VALEO\src\valeo\infrastructure\tools\DfInDfOut.py

```python

import pandas as pd

class DfInDfOut:

    # https://github.com/scikit-learn/scikit-learn/issues/5523 : Pandas in, Pandas out
    def check_output(self, X, ensure_index=None, ensure_columns=None):
        """
        Joins X with ensure_index's index or ensure_columns's columns when avaialble
        """
        if ensure_index is not None:
            if ensure_columns is not None:
                if type(ensure_index) is pd.DataFrame and type(ensure_columns) is pd.DataFrame:
                    X = pd.DataFrame(X, index=ensure_index.index, columns=ensure_columns.columns)
            else:
                if type(ensure_index) is pd.DataFrame:
                    X = pd.DataFrame(X, index=ensure_index.index)
        return X
```

File - C:\EXED\Training\___VALEO\src\valeo\infrastructure\tools\YamlLoader.py

```python

import os
import yaml
import logging

# from infrastructure.LogManager import LogManager

class YamlLoader :
    """
    Load a yaml  configuration file
    """
    logger = None

    def __init__(self):
        if YamlLoader.logger is None :
            YamlLoader.logger =  logging.getLogger(__name__)
            logging.basicConfig(level=logging.INFO)

    def load(self, file_pathname:str) -> dict :
        if os.path.exists(file_pathname):
            with open(file_pathname, 'rt') as f:
                try:
                    dict =  yaml.safe_load(f.read())
                    # YamlLoader.logger.info(f'Loading file "{file_pathname}":\n\t{dict}')
                    YamlLoader.logger.info(f'Loading file "{file_pathname}":\n{dict}')
                    return dict
                except Exception as ex:
                    YamlLoader.logger.exception(f'Error while loading file "{file_pathname}"')
        else:
            YamlLoader.logger.error(f'Error while loading file "{file_pathname}"')

        return None

```

```python

import os
import logging

from valeo.infrastructure.tools.YamlLoader import YamlLoader

class ConfigLoader(YamlLoader) :
    logger = None
    """
    Load an external or a package embedded configuration file.
    Check first if the environment variable {APP_CONFIG_PATHNAME}
    """

    def __init__(self):
        super().__init__()
        ConfigLoader.logger = logging.getLogger(__name__)

    def load(self, file_pathname:str, env_key_as_config_pathname:str) -> dict :
        try :
            path_as_key = os.getenv(env_key_as_config_pathname, None)
            return super().load(path_as_key if path_as_key else file_pathname )
        except Exception as ex :
            ConfigLoader.logger.exception(f'Error while loading file "{file_pathname}"')
            raise ex
            # self.logger.error(ex, exc_info=True)
        return None

```

File - C:\EXED\Training\___VALEO\src\valeo\infrastructure\tools\DebugPipeline.py

```python

import os
from datetime import datetime

from sklearn.base import BaseEstimator, TransformerMixin
import numpy as np

from valeo.infrastructure import Const as C


class DebugPipeline(BaseEstimator, TransformerMixin):
    OFFSET  = 10
    counter = -OFFSET

    def __init__(self):
        DebugPipeline.counter = ( (DebugPipeline.counter + DebugPipeline.OFFSET) // DebugPipeline.OFFSET) * 
DebugPipeline.OFFSET

    def transform(self, X, y=None):
        # %f : print micro seconds
        # np.savetxt(os.path.join(C.rootProject(), 'log', 'dbgPipeline_' + datetime.now().strftime("%Y_%m_%d-%H
.%M.%S_") + str(DebugPipeline.counter)) + '.txt', X, delimiter=',')
        DebugPipeline.counter += 1

        return X

    def fit(self, X, y=None, **fit_params):
        if y is not None :
            # np.savetxt(os.path.join(C.rootProject(), 'log', 'dbgPipeline_' + datetime.now().strftime("%Y_%m_%d
-%H.%M.%S_") + str(DebugPipeline.counter)) + '_Y.txt', y, delimiter=',')
            # np.savetxt(os.path.join(C.rootProject(), 'log', 'dbgPipeline_' + datetime.now().strftime("%Y_%m_%d
-%H.%M.%S_") + str(DebugPipeline.counter)) + '_X.txt', X, delimiter=',')
            DebugPipeline.counter += 1
        return self
```