

Prévision des défauts sur les lignes de production Valéo

Index:

1. Contexte de l'étude
2. Description du dataset case
 - a. Entrées
 - b. Sortie
 - c. Croisement Entrées/Sortie
3. Import des packages Python et rechargement automatique des packages du projet
4. Chargement des données 'Training'
5. Exploration tabulaire des données:
 - a. Visualisation tabulaire des données - Affichage du type 'head()'
 - b. Rapport sémantique des données - Affichage du type 'info()'
 - c. Données manquantes par type de 'feature'
 - d. Statistique descriptive
 - Nombre de features égales à Nulle (0)
 - Ratio d'observations ayant des features en outlier
 - Comparaison des valeurs statistiques entre le dataset initial et le dataset dépourvu des outliers
 - e. Distribution du jeu des données:
6. Exploration graphique univariable des données:
 - a. Histogramme des features après gestion des valeurs manquantes
 - Histogramme comparant la feature 'OP100_Capuchon_insertion_mesure' avant et après la gestion des valeurs manquantes
 - b. Histogramme des features après gestion des valeurs manquantes + application de 'RobustScaler'
 - c. Histogramme des features après gestion des valeurs manquantes + transformation 'log10'
 - d. Violon et boîte à moustaches des features avec gestion des valeurs manquantes
 - e. Violon et boîte à moustaches des features avec gestion des valeurs manquantes + application de 'RobustScaler'
7. Exploration graphique bivariées 'feature/target' des données:
 - a. Matrice de corrélation et heatmap après gestion des valeurs manquantes
 - b. Matrice de corrélation et heatmap après gestion des valeurs manquantes et rescale
 - c. Nuage de points entre la 'target' et les autres 'features'
8. Feature Engineering/Sélection et choix faits/Hypothèses choisies
9. Analyse de la target:
 - a. Vérification de l'équilibre des données
 - b. Distribution du dataset selon les classes de la target
 - c. Histogramme de distribution du dataset selon les classes de la target
10. Analyse de la target après un oversampling SMOTE

A FAIRE

- a. Régénération SMOTE de la classe minoritaire de la target
 - b. Statistique descriptive du nouveau dataset
 - c. Nouvelle distribution équilibrée du nouveau dataset
 - d. Matrice de corrélation et heatmap du nouveau dataset
 - e. Violon et boîte à moustaches des features du nouveau dataset
 - f. Ratio d'observations ayant des features en outlier du nouveau dataset
11. Modèle à base d'arbres : Balanced Random Forest Classifier **FAIT PARTIELLEMENT => A COMPLETER**
- a. Train / Test / Split + F1 et ROC
 - b. Cross Validation + F1 et ROC
12. Modèle à base de distance : SMOTE et Logistique regression **FAIT PARTIELLEMENT => A COMPLETER**
- a. Train / Test / Split + F1 et ROC
 - b. Cross Validation + F1 et ROC
13. Modèle à base de Réseau de Neuronne ou bien de Stacking (Ensemble learning) **A FAIRE**
14. Conclusions **A FAIRE**
15. Perspectives **A FAIRE**
16. Annexe: Code Python

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 régénération 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 réseau de Neuronne ou bien de Stacking: **Points 11, 12, 13 de l'index ci dessus**
 - Pour chaque classifieur faire: TrainTestSplit / CV / SearchGridCV
 - Pour chaque classifieur: Analyse et interprétation 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 différents 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 appliquée 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'algorithme 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 a de folds.

C'est à dire: Si le Cross Validation est effectué sur 5 folds, alors la méthode 'cross_validate' retourne 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 65^{ème} 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.

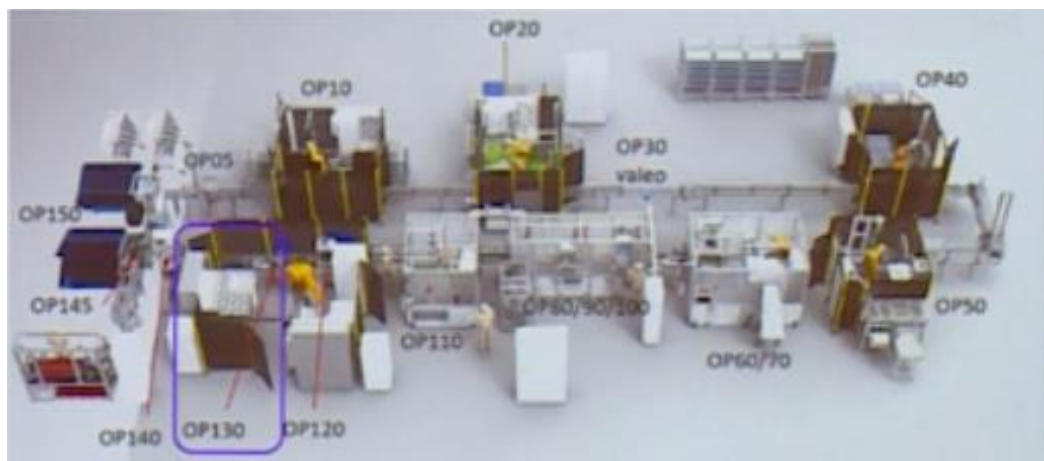
On distingue par exemple:

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: $\text{Sortie} = f(\text{entrées})$. L'ensemble de données contient 34515 échantillons d'apprentissage et 8001 échantillons de test.

c - Croisement Entrées/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
0	I-B-XA1207872-190429-00688	180.4	190.51	173.1	113.64
1	I-B-XA1207872-190828-00973	138.7	147.70	163.5	109.77
2	I-B-XA1207872-190712-03462	180.9	150.87	181.2	109.79
3	I-B-XA1207872-190803-00051	173.5	159.56	151.8	113.25
4	I-B-XA1207872-190508-03248	174.5	172.29	177.5	112.88

OP090_SnapRingFinalStroke_value	OP110_Vissage_M8_torque_value	OP100_Capuchon_insertion_mesure	OP120_Rodage_U_mesure_value
12.04	12.16	NaN	11.97
12.12	12.19	0.39	11.97
11.86	12.24	NaN	11.97
11.82	12.35	0.39	11.97
12.07	12.19	NaN	11.97

OP070_V_1_torque_value	OP090_StartLinePeakForce_value	OP110_Vissage_M8_angle_value	OP090_SnapRingMidPointForce_val	OP070_V_2_torque_value
6.62	26.37	18.8	109.62	6.60
6.41	21.03	18.5	105.48	6.40
6.62	25.81	17.5	100.03	6.61
6.62	24.62	15.6	104.94	6.61
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 sémantique 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') sera 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-XA1207872-190907-01953	137.4	0.0	166.7	105.51
1651	I-B-XA1207872-190821-01367	178.7	0.0	170.4	112.95
22483	I-B-XA1207872-190424-02168	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 connaît pas la raison de ces outliers:

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

Pour limiter l'effet des outliers:

- Utiliser un modèle résistant aux outliers, comme les arbres
- Transformer les données en utilisant la fonction Log Lors de la visualisation graphique des données on va retrouver des distributions biaisées (skewed)

Comparaison des valeurs statistiques entre le dataset initial et le dataset dépourvu des outliers

```
# 1 - Le dataset dépourvu des outliers  
X_data_out = X_data[~outliers]  
  
# 2 - Créer les 2 dataframes des valeurs statistiques descriptives  
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)
```

	25%	25%-o	50%	50%-o	75%	75%-o	count	count-o	max	max-o	mean	mean-o	min	min-o
key_0														
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	180.147298	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	6.550887	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	102.385	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	198.92	188.87	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	23.789882	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 jeu 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

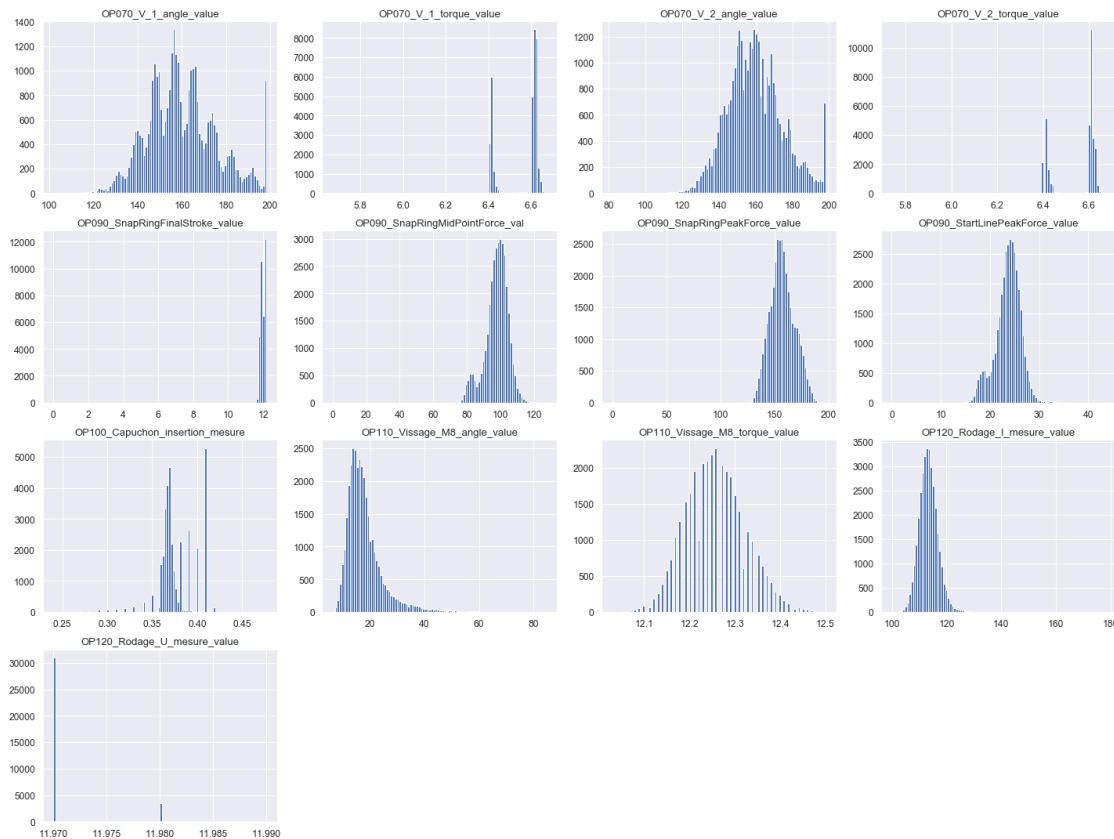
L'étude concerne la classification des données déséquilibrée avec des valeurs de données manquantes. C'est un problème classique dans l'industrie et dans bien d'autres domaines : détection de fraude, détection de spam, domaine médical,

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')`

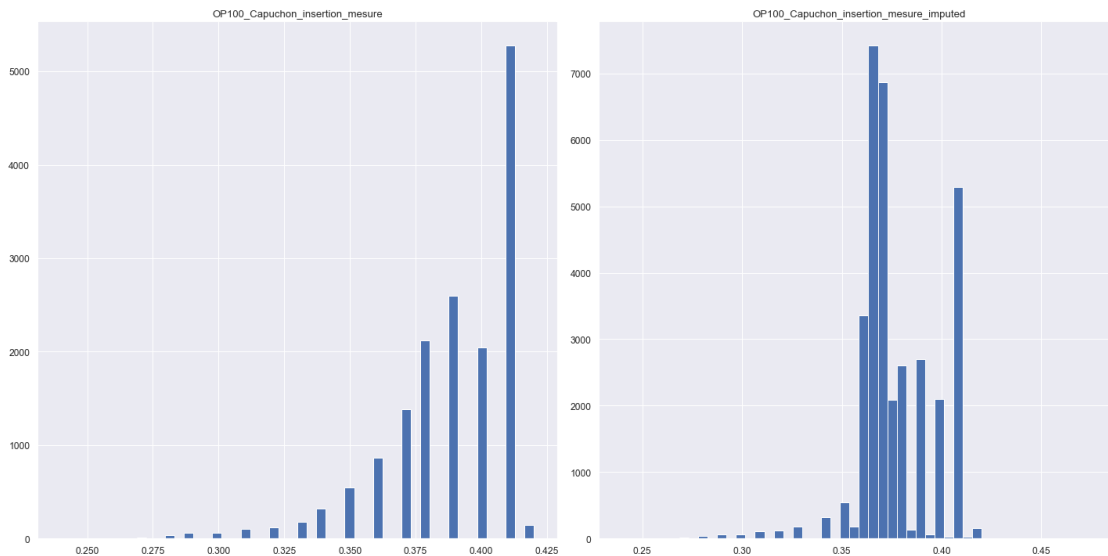
En observant les graphes des différents features, on constate:

- 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 représentent 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
 -

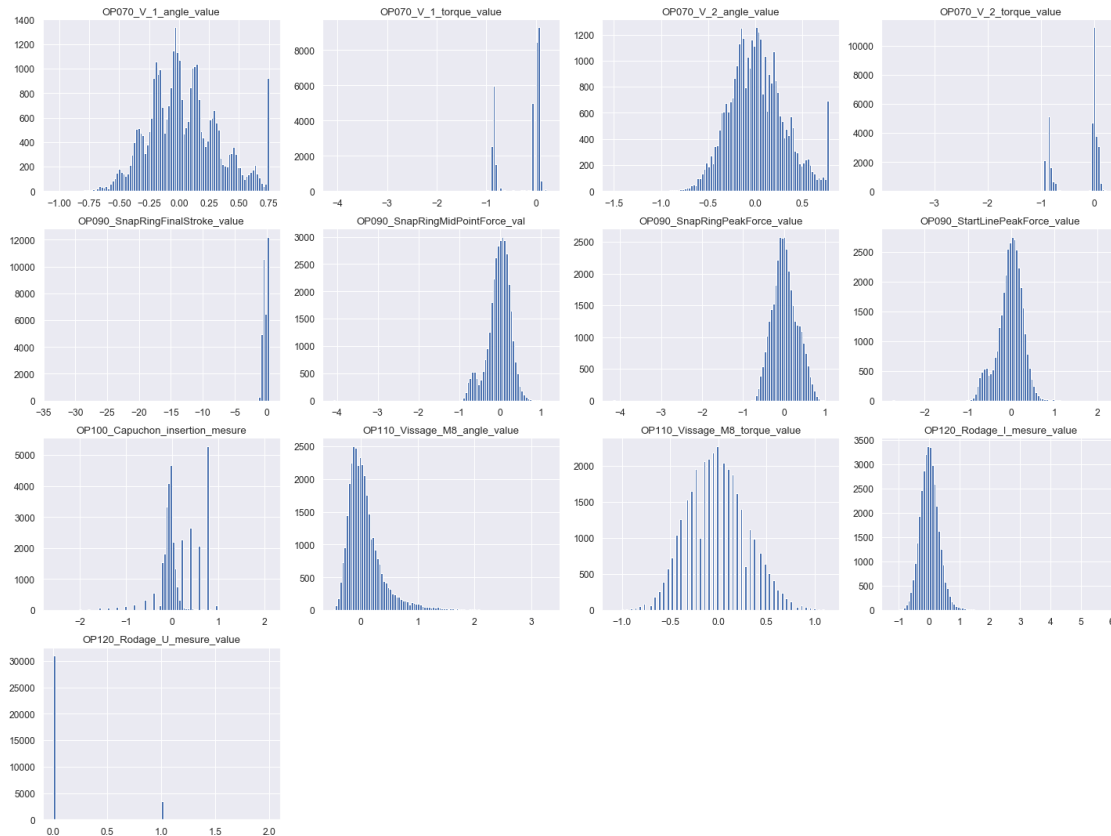
Histogramme comparant la feature 'OP100_Capuchon_insertion_mesure' avant et après la gestion des valeurs manquantes:

```
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:

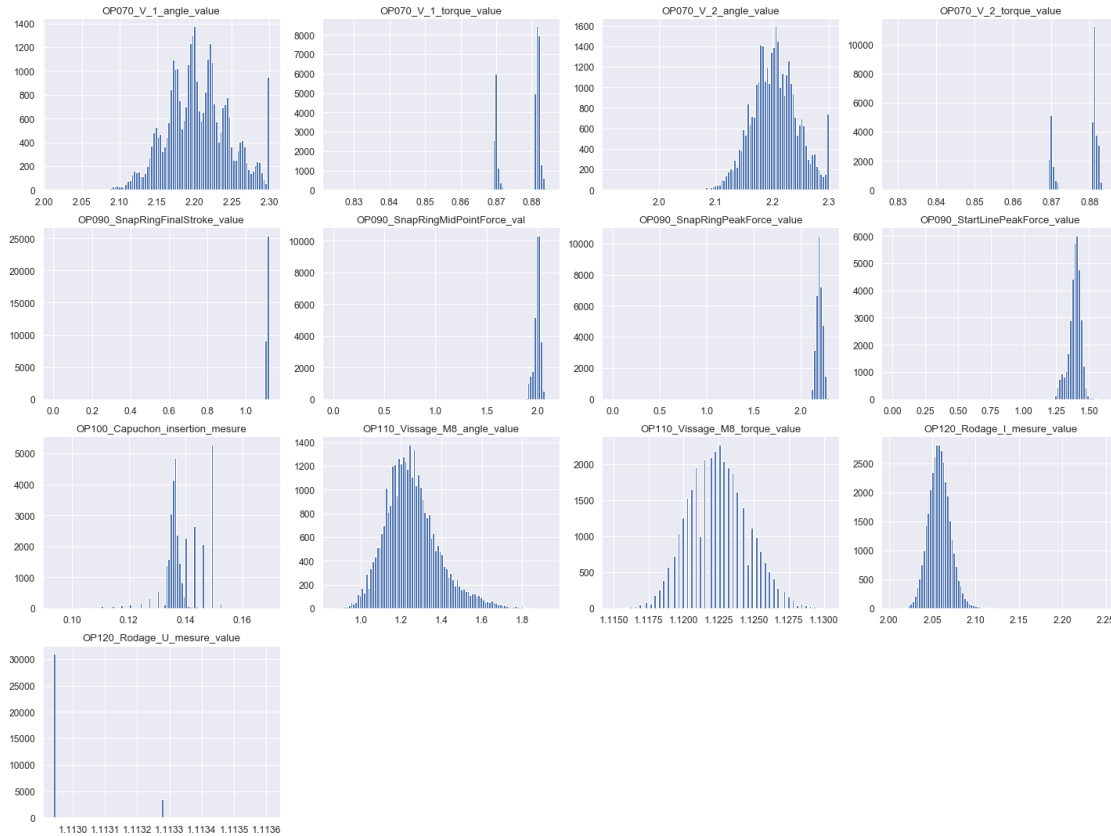
Les algorithmes d'apprentissage automatique prennent en compte uniquement la magnitude des mesures, mais pas les unités de ces mesures. Par la suite, une caractéristique exprimée en une magnitude (nombre) très élevée, peut affecter la prévision beaucoup plus qu'une caractéristique tout aussi importante.

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()
```



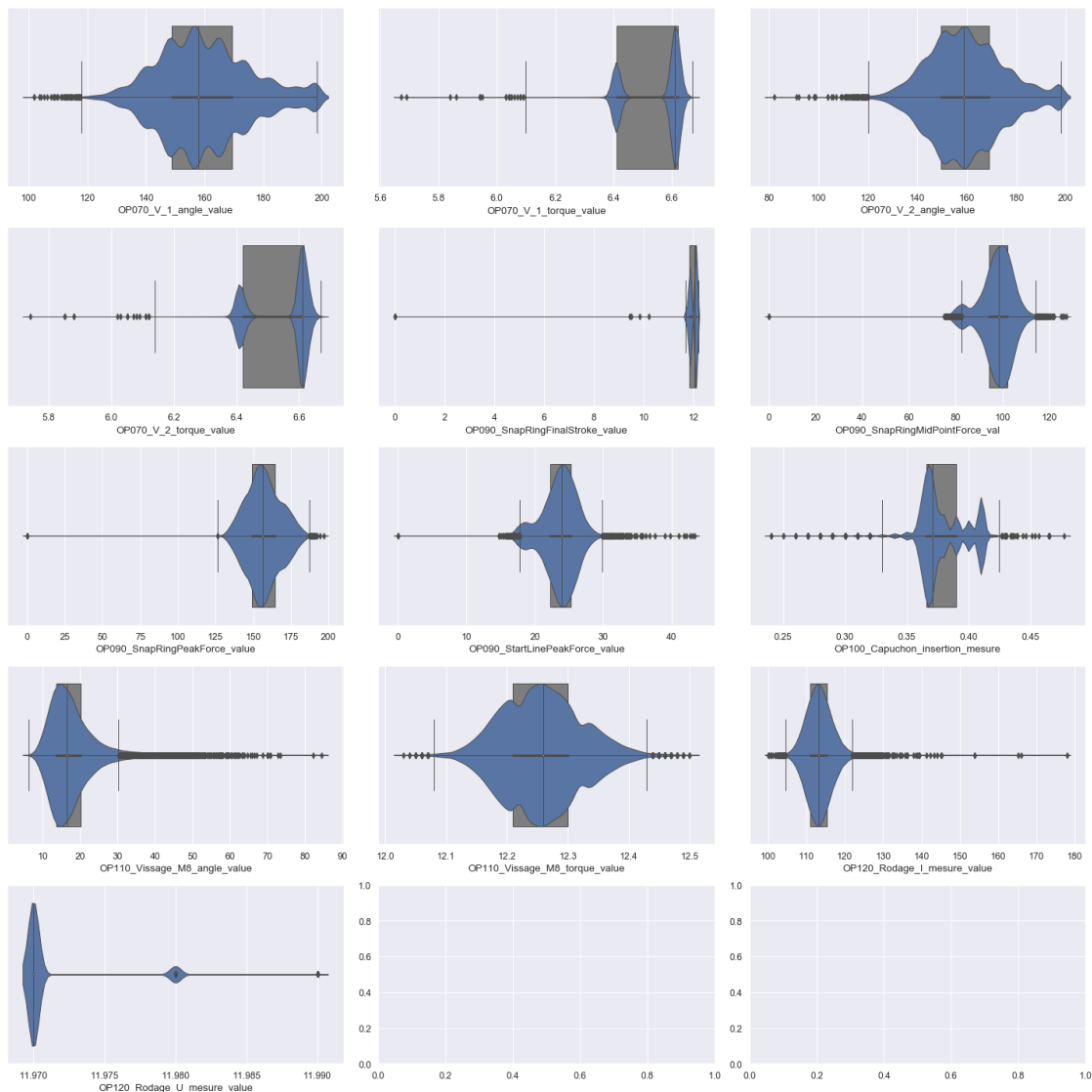
On constate que les distributions ci dessus correspondant aux features transformées par log10 **ressemblent plus** à des distributions Normales.

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, médiane, 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)
```

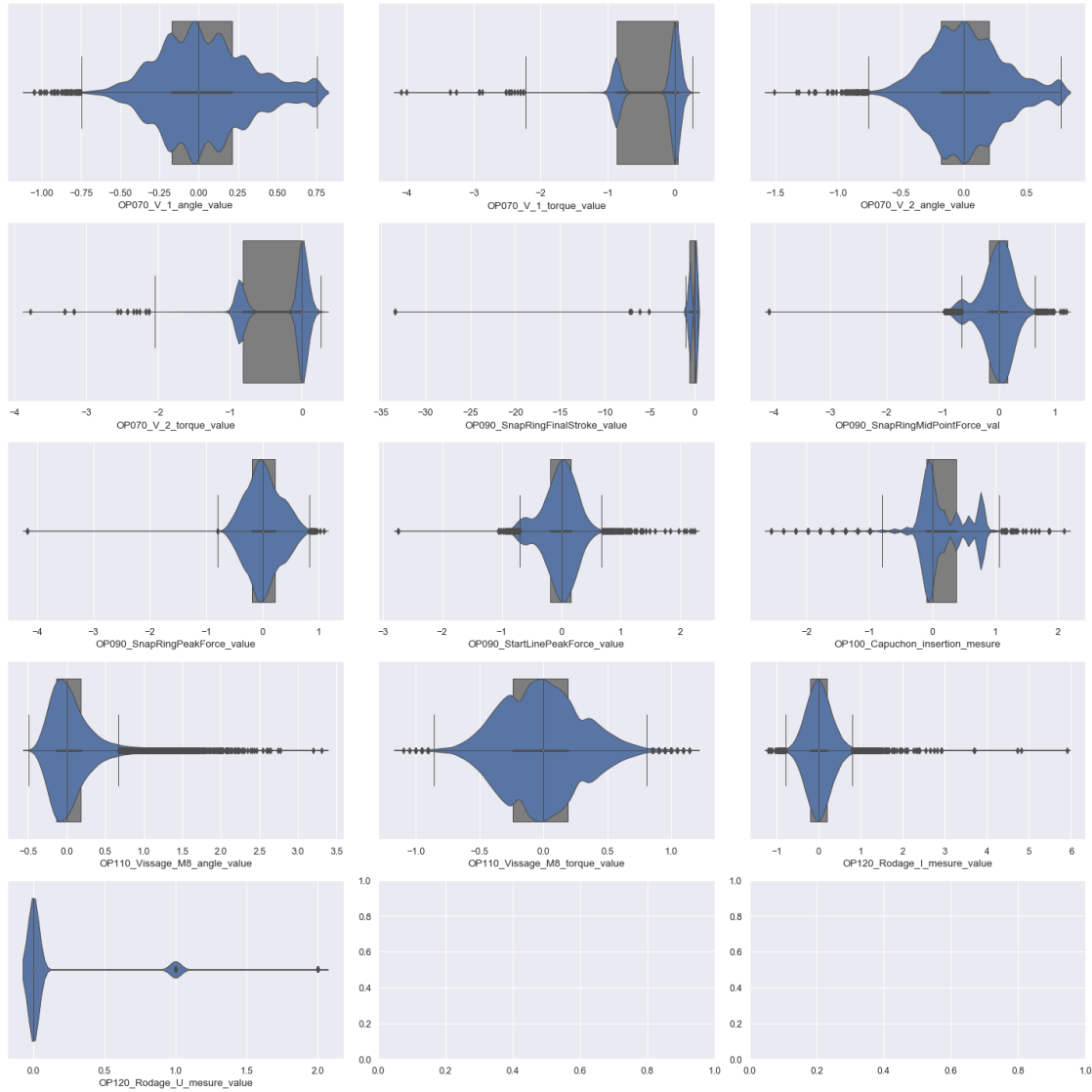


Représenter la distribution d'une feature par un 'violin plot' superposé à un 'box plot' permet de:

- 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 bivariées 'feature/target' des données

a - Matrice de corrélation (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 - Corrélation 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
```

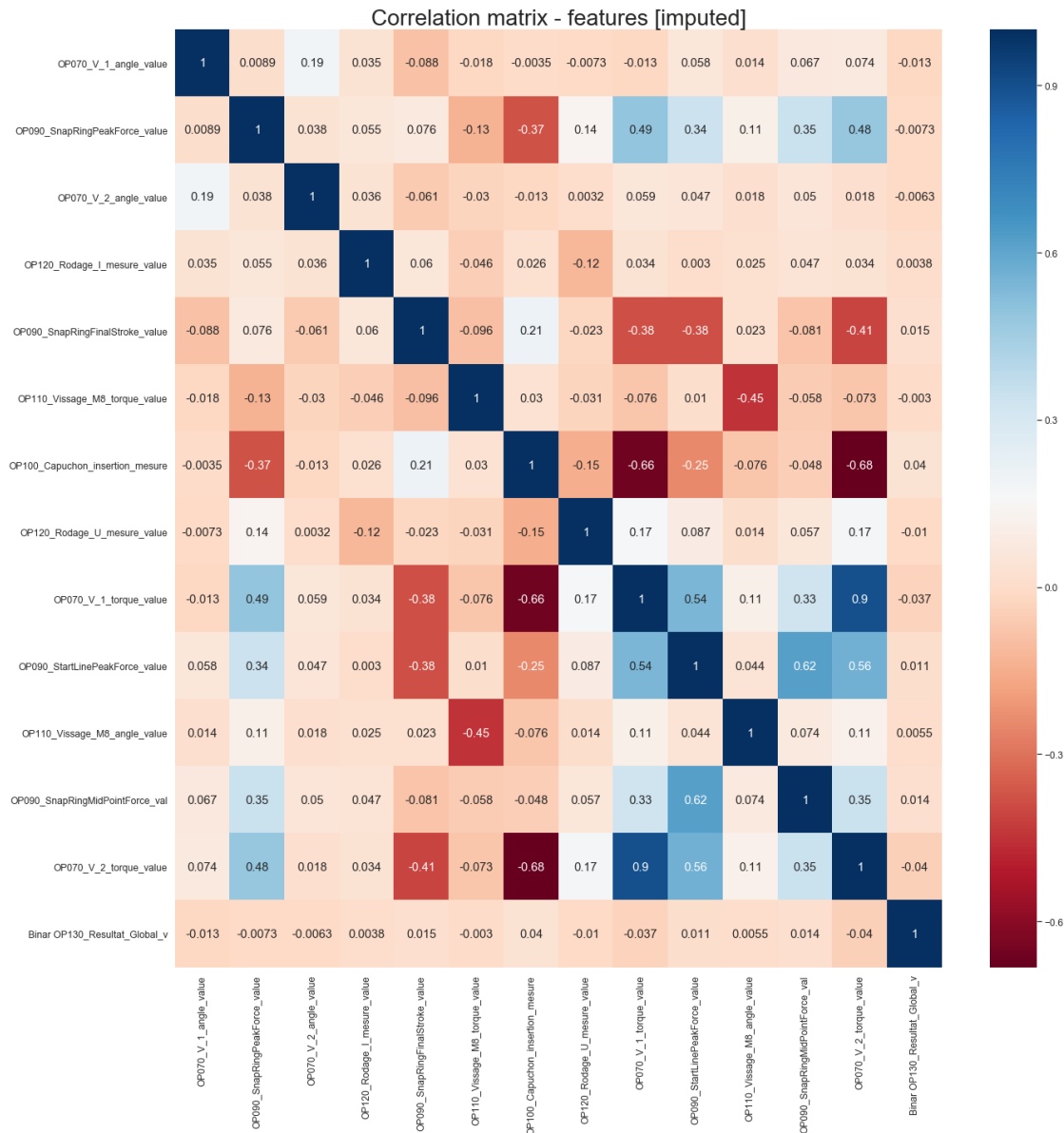
Le coefficient de correlation varie entre -1 et 1 :

- 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 matrice de correlation, on constate:

- L'absence de toute corrélation forte entre la cible 'Binar OP130_Résultat_Global_v' et n'importe quelle fonction.
- L'existence de corrélations positives (0.54, 0.49, 0.48, ...) et négatives (-0.68, -0.45, -0.38, ...) parmi les autres fonctionnalités

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]
```

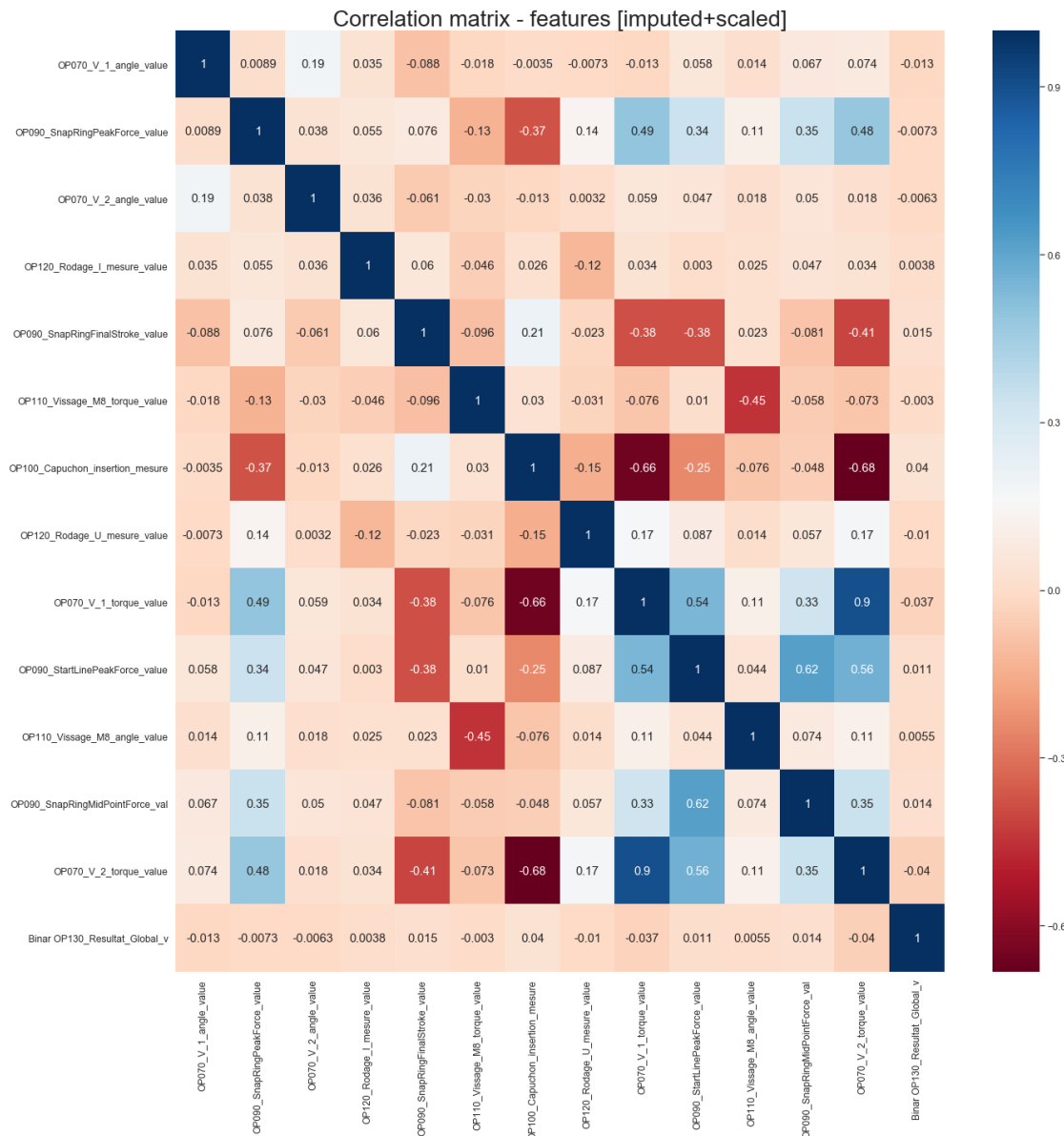
3 - Correlation entre la target "Binar OP130_Resultat_Global_v" et les autres attributs

```
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
```

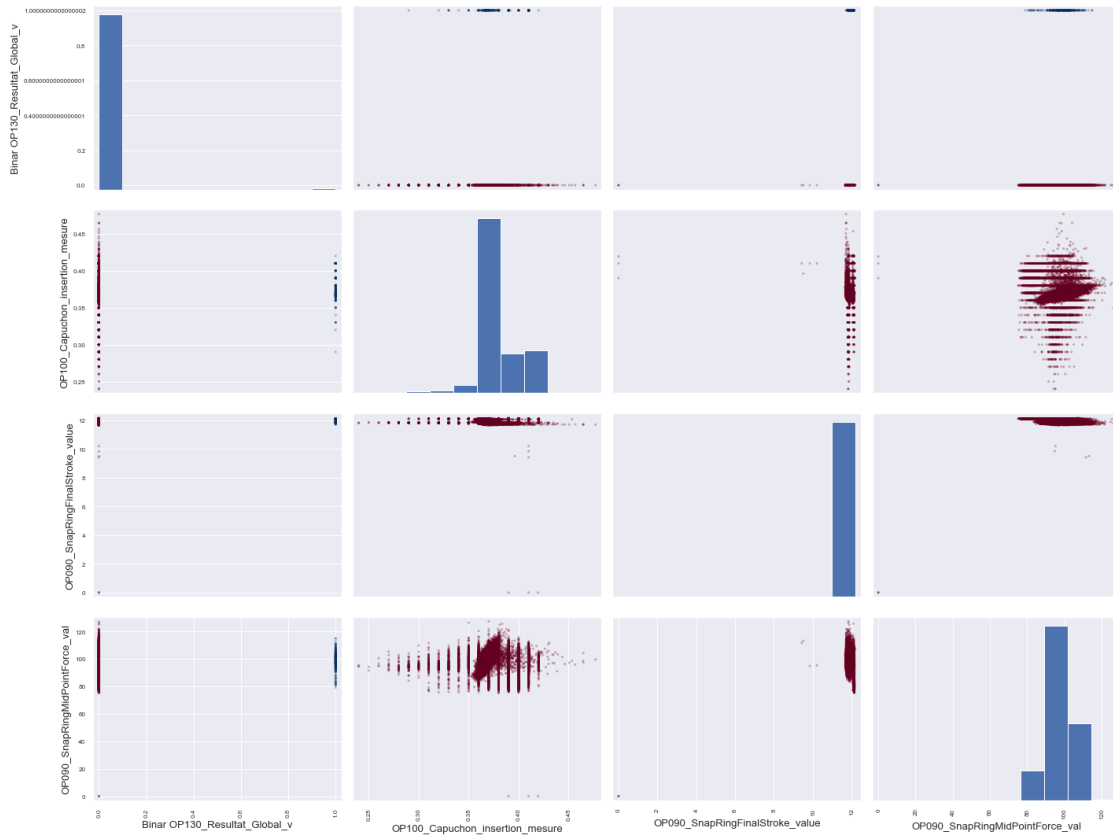
4 - Dessiner la Heatmap

```
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:

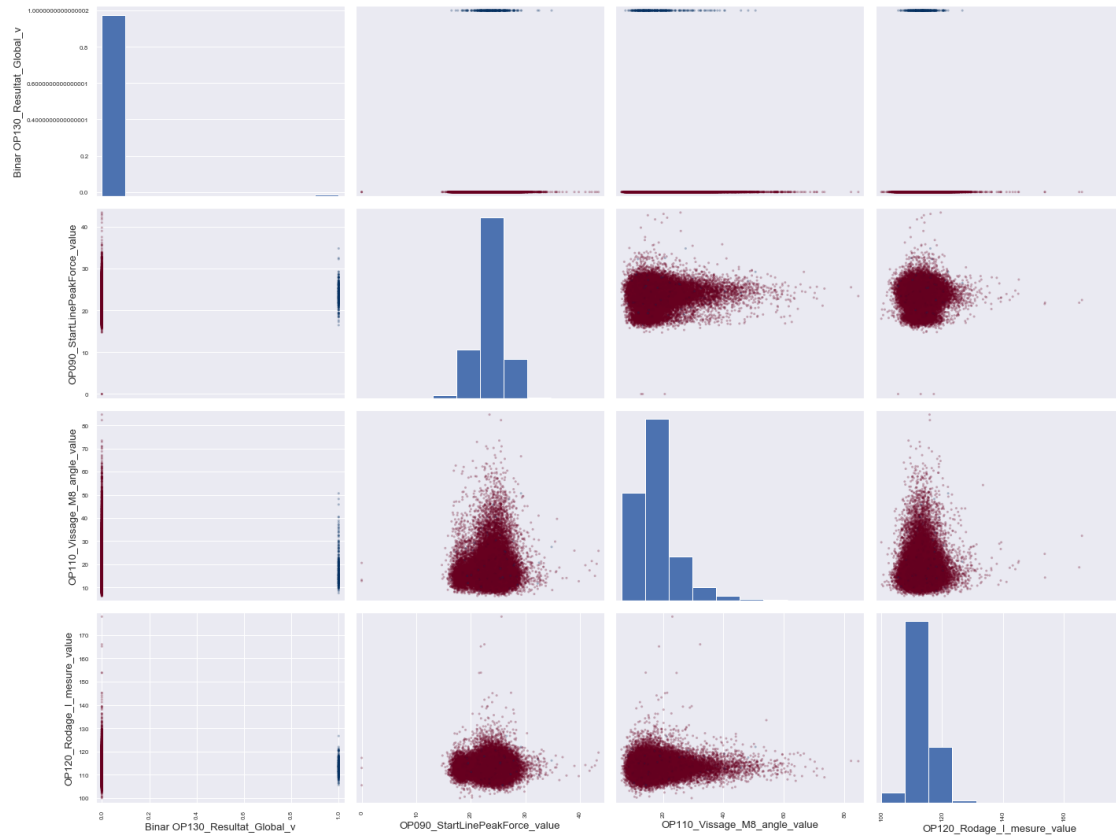
```
features = [Const.Binar_OP130_Resultat_Global_v,
            Const.OP100_Capuchon_insertion_mesure, Const.OP090_SnapRingFinalStroke_value,
            Const.OP090_SnapRingMidPointForce_val]
ImgUtil.save_df_scatter_matrix_plot(XY_data_transformed[features], "XY_data_imputed_corr_pos_1",
cfield=Const.Binar_OP130_Resultat_Global_v)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



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.

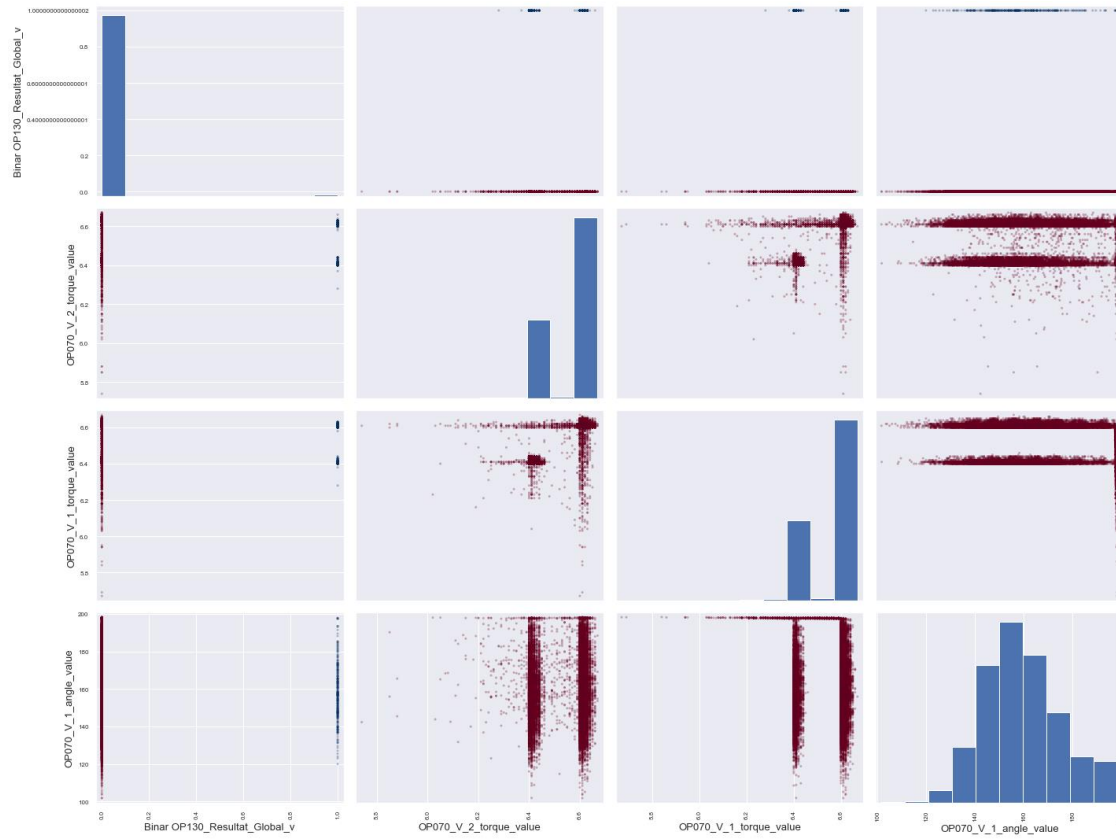
```
features = [Const.Binar_OP130_Resultat_Global_v,
            Const.OP090_StartLinePeakForce_value, Const.OP110_Vissage_M8_angle_value,
            Const.OP120_Rodage_I_mesure_value]
ImgUtil.save_df_scatter_matrix_plot(XY_data_transformed[features], "XY_data_imputed_corr_pos_2",
cfield=Const.Binar_OP130_Resultat_Global_v)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



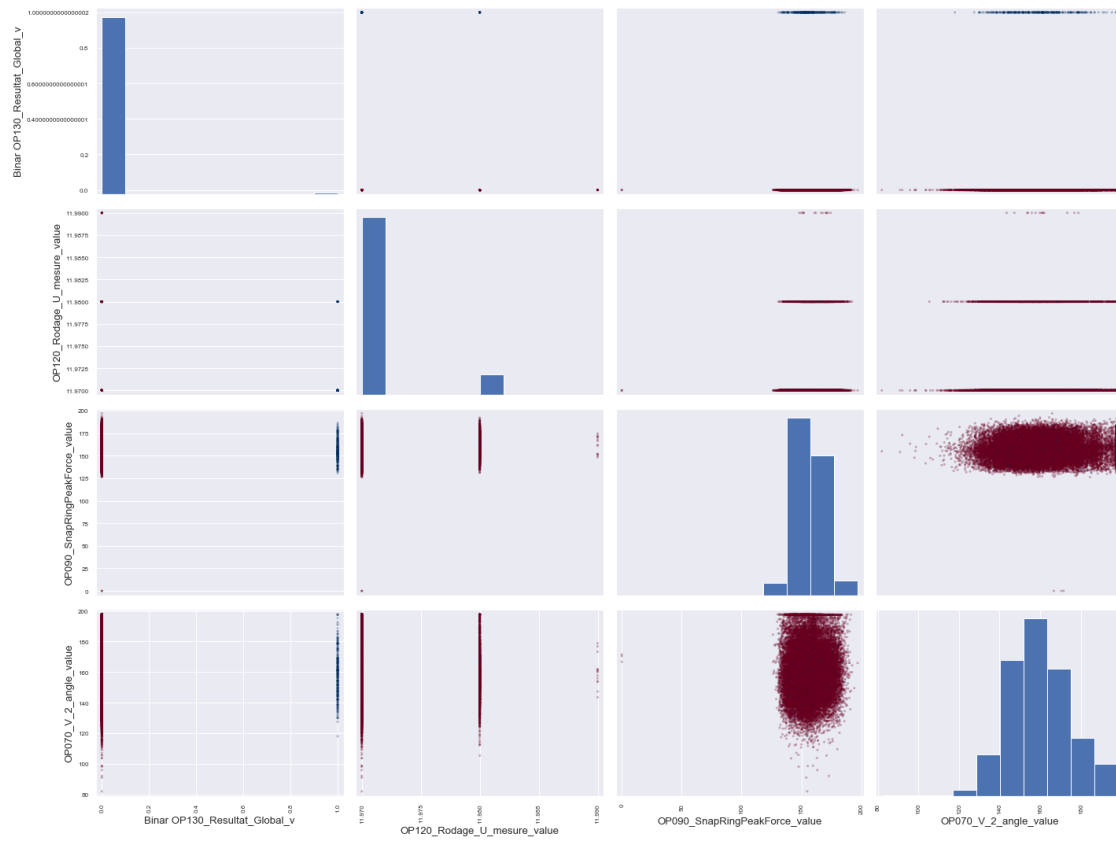
```

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')
```

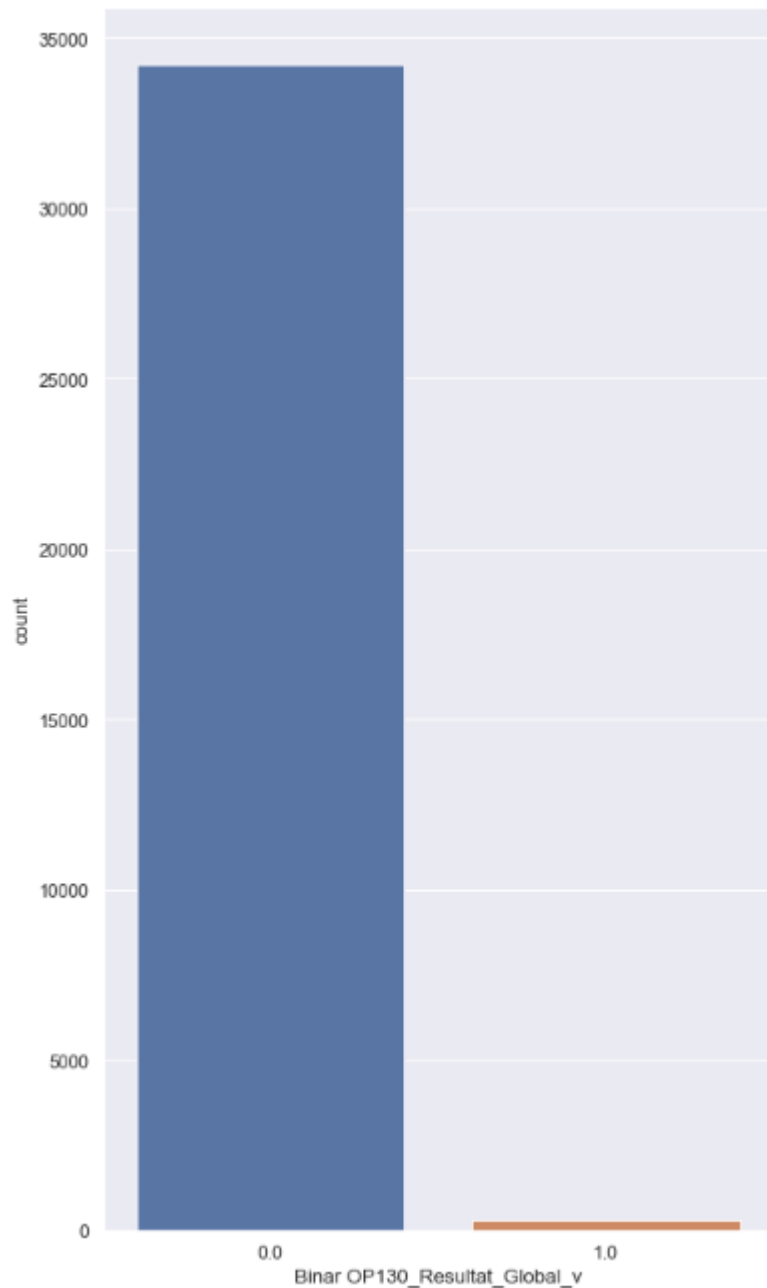
Nombre total des démarreurs : 34515

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)
```

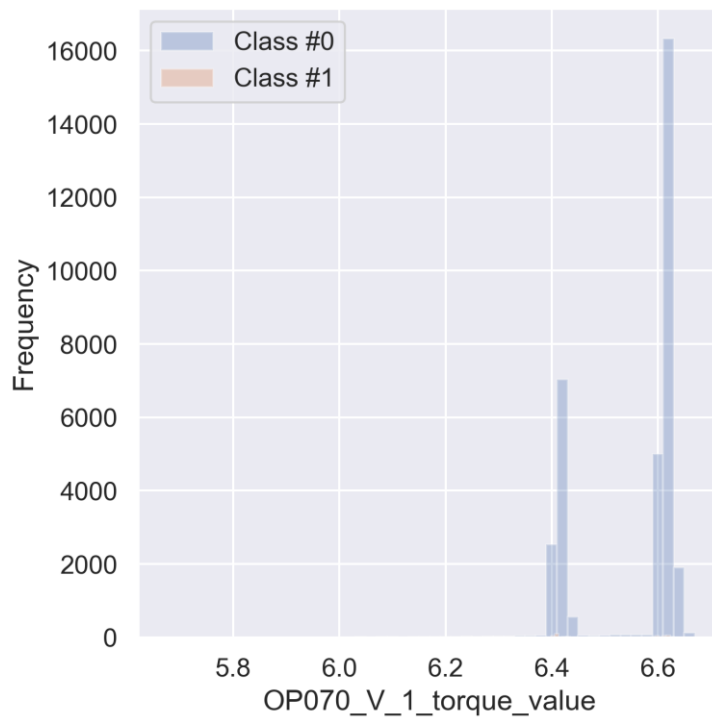
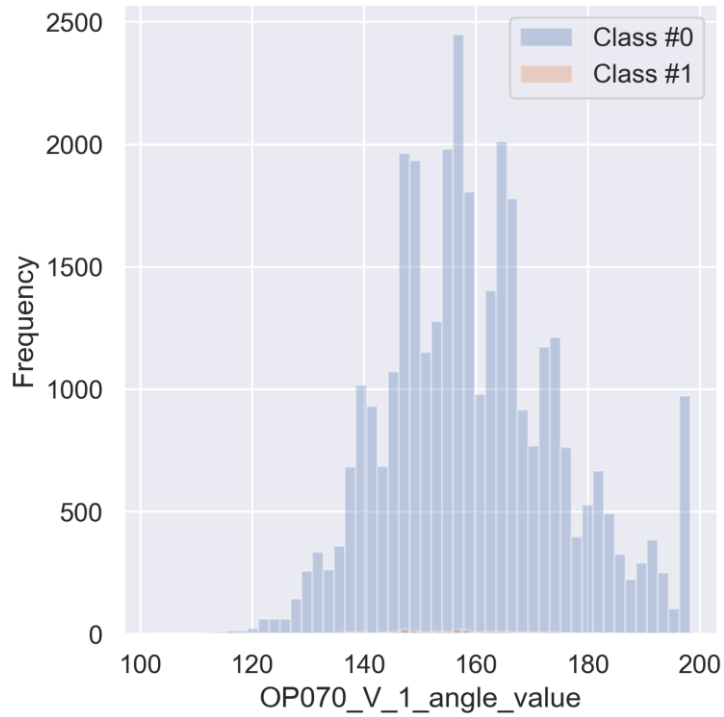


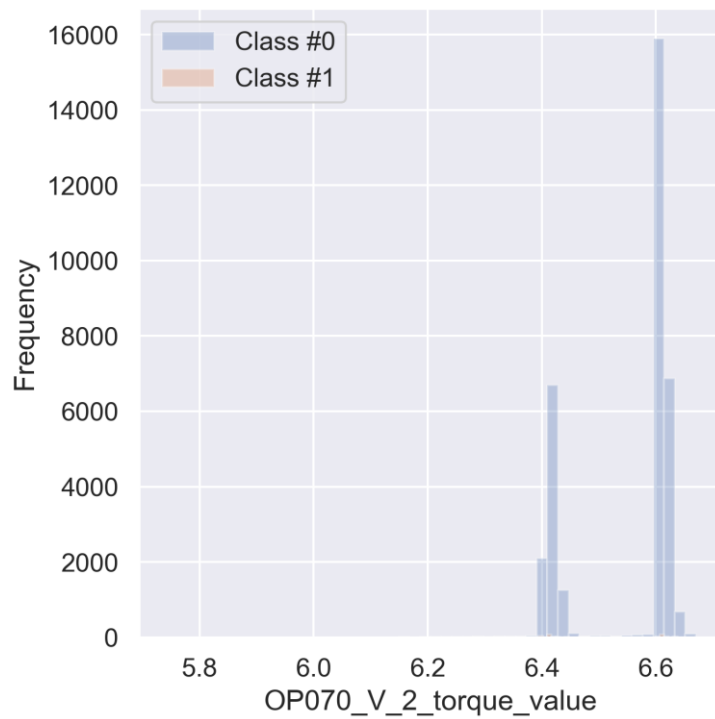
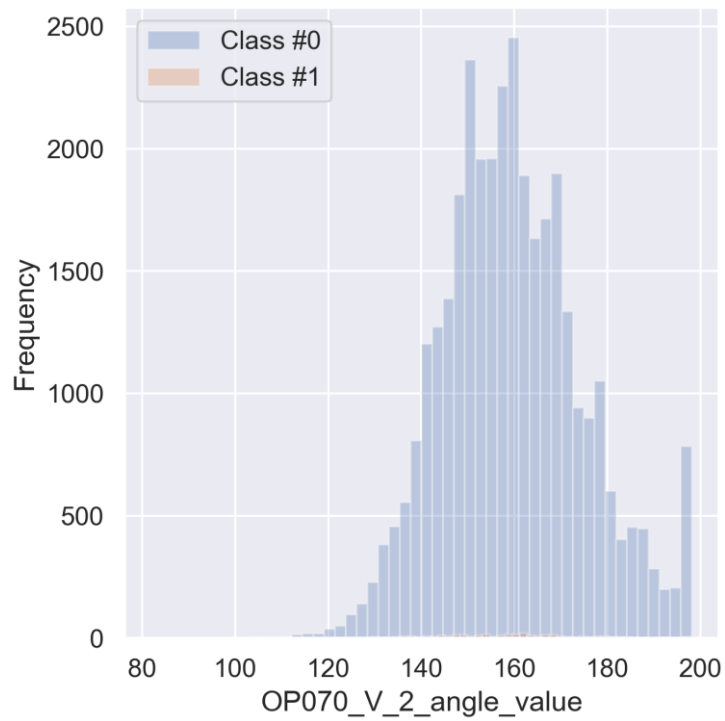
On constate que le jeu de données est fortement déséquilibré. \ Presque totalité des démarreurs (99.12%) ne sont pas défectueux lors de la sortie de la ligne de production.

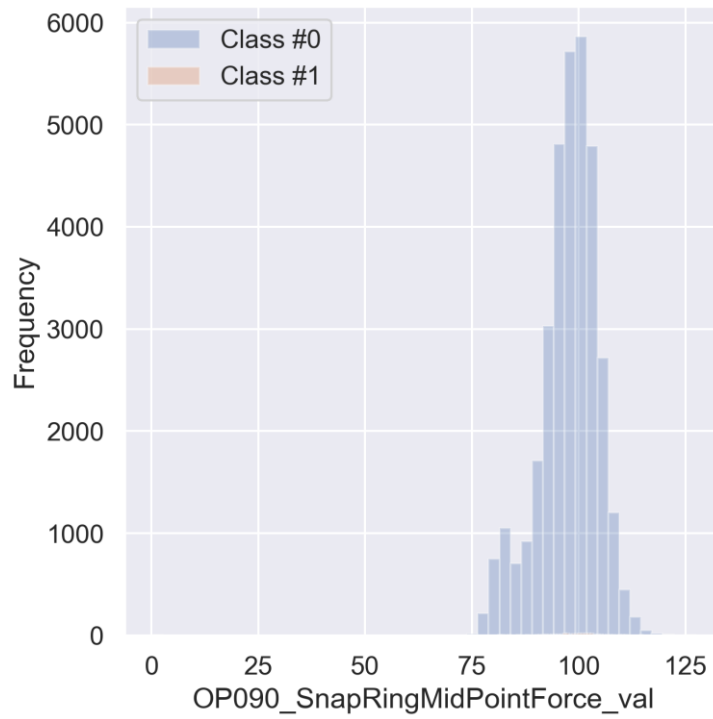
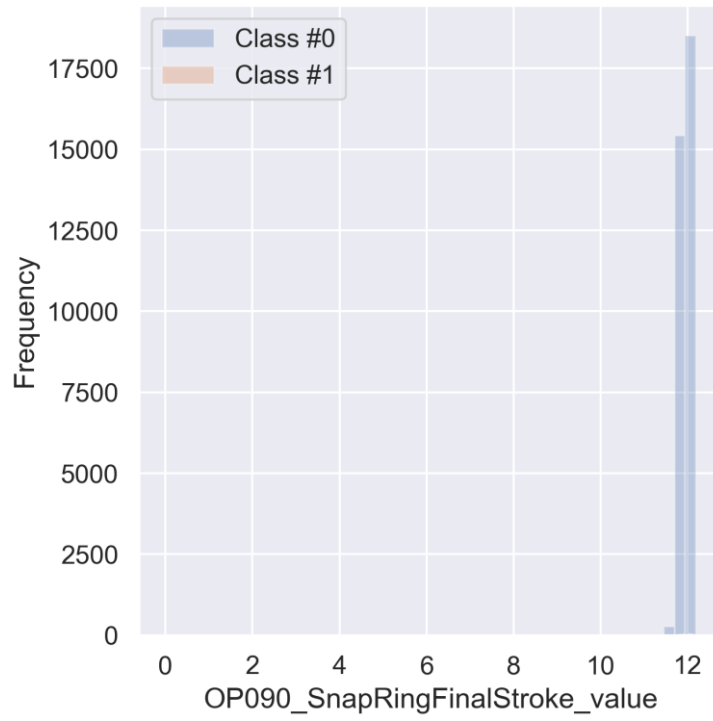
En utilisant cette base de données comme base pour les modèles prédictifs et pour les analyses, on pourrait obtenir beaucoup d'erreurs par des algorithmes inadaptés car ils 'supposeront' que les 'demarreurs' ne sont pas défectueux. \ On cherche un modèle capable de détecter les patterns qui prédisent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

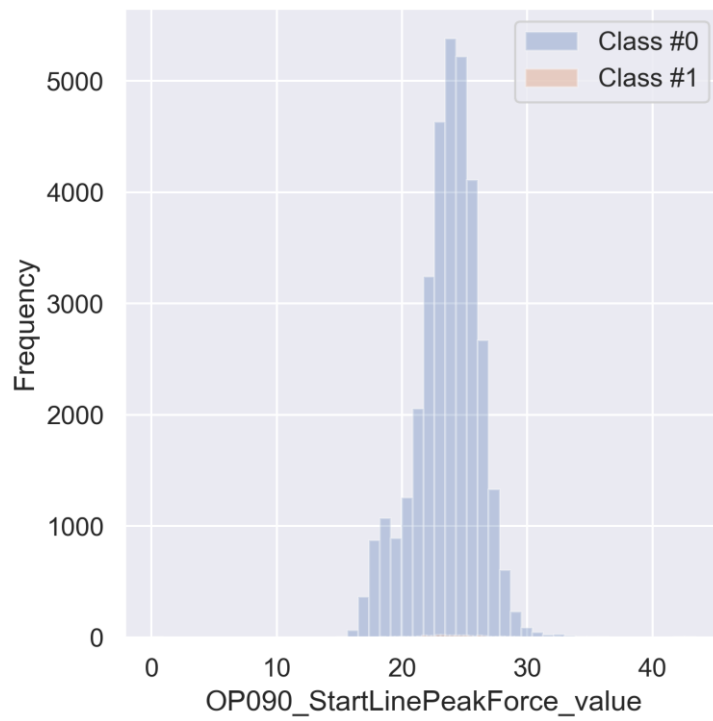
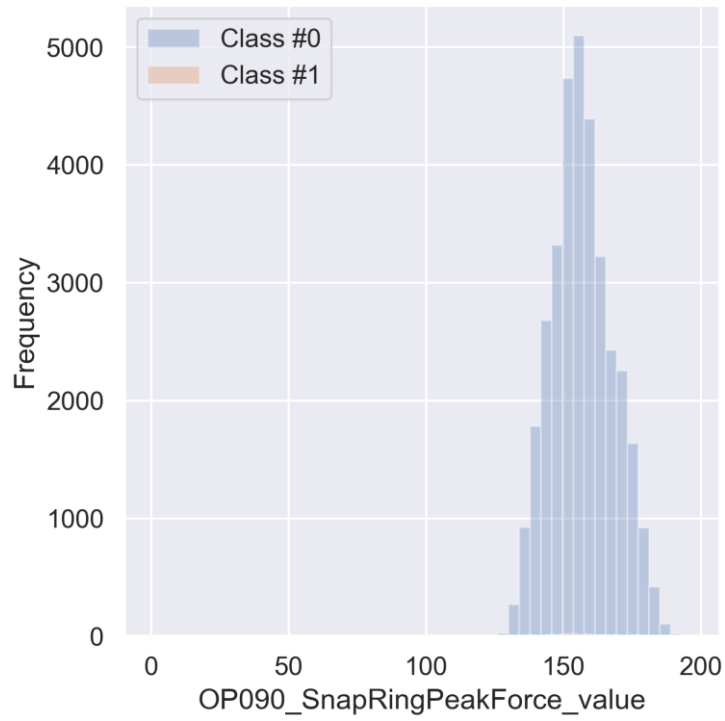
c - Histogramme de distribution du jeu de données selon les classes de la target:

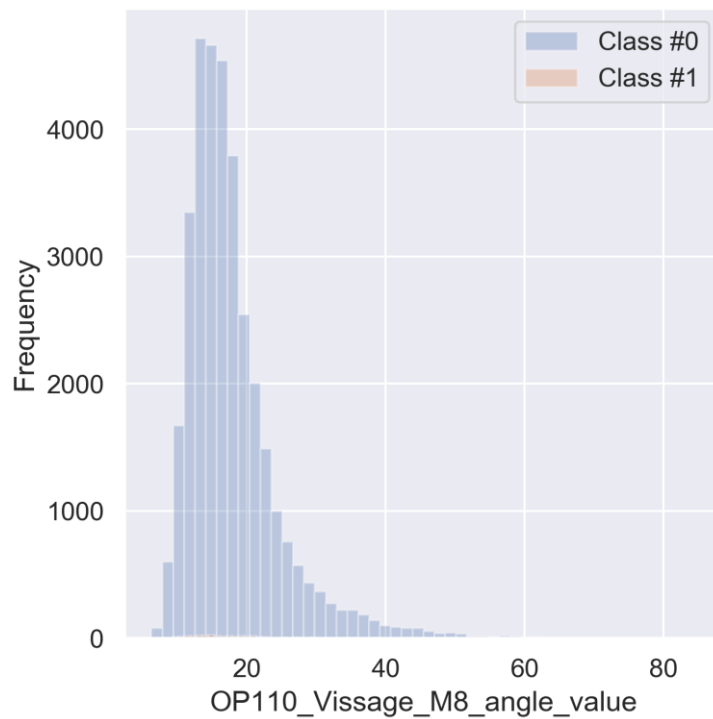
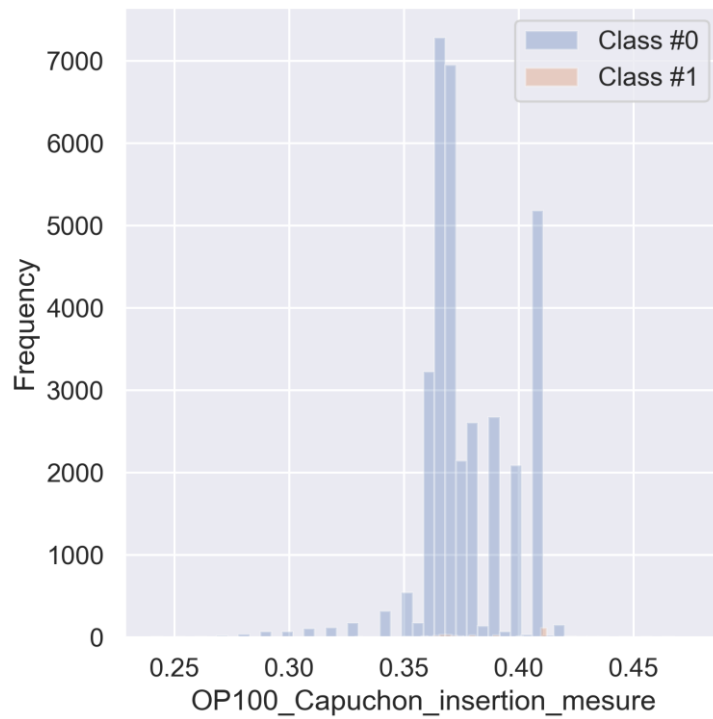
```
ImgUtil.save_df_XY_hist_plot(XY_data_transformed, "XY_imputed",  
y_target_name=Const.Binar_OP130_Resultat_Global_v)
```

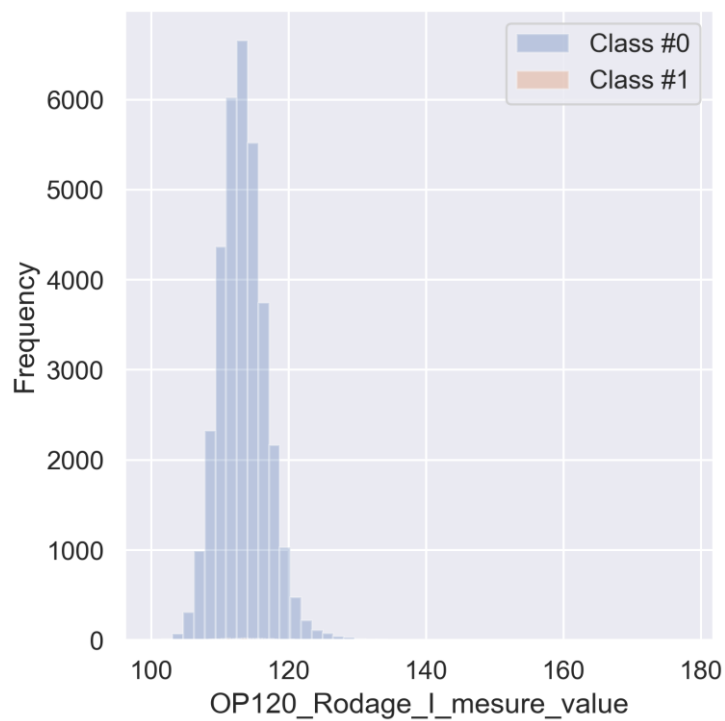
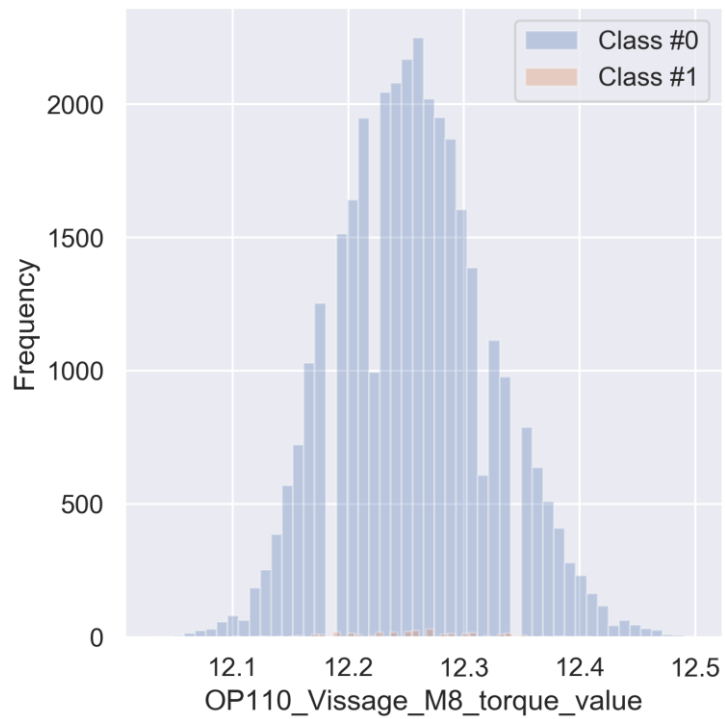


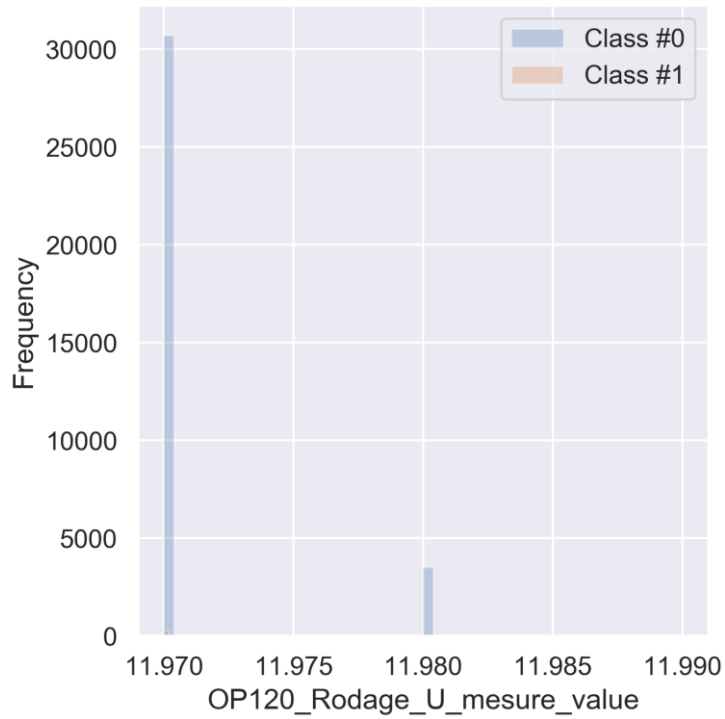












On constate que la classe minoritaire se retrouve délimité à l'intérieur d'une plage de valeurs pour certains features. (ex: OP070_V_1/2_angle_value, OP110_Vissage_M8_torque_Value). Pour cela, il faudrait vérifier l'impact si on transforme ces features numériques continues en des features catégoriques 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 minoritaire 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
0	180.4	190.51	173.1	113.64	12.04
1	138.7	147.70	163.5	109.77	12.12
2	180.9	150.87	181.2	109.79	11.88
3	173.5	159.56	151.8	113.25	11.82
4	174.5	172.29	177.5	112.88	12.07

	OP110_Vissage_M8_torque_value	OP100_Capuchon_insertion_mesure	OP120_Rodage_U_mesure_value	OP070_V_1_torque_value	OP090_StartLinePeakForce_value
	12.16	0.373146	11.97	6.62	26.37
	12.19	0.390000	11.97	6.41	21.03
	12.24	0.370876	11.97	6.62	25.81
	12.35	0.390000	11.97	6.62	24.62
	12.19	0.368966	11.97	6.62	29.22

	OP110_Vissage_M8_angle_value	OP090_SnapRingMidPointForce_val	OP070_V_2_torque_value	OP130_Resultat_Global_v	Binar
	18.8	109.62	6.60		0.0
	18.5	105.48	6.40		0.0
	17.5	100.03	6.61		0.0
	15.6	104.94	6.61		0.0
	33.6	99.19	6.61		0.0

b - Statistique descriptive du nouveau dataset:

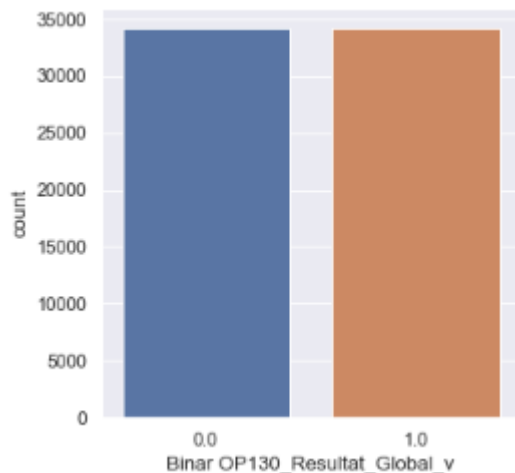
```
oversampled_XY.describe().transpose()
```

	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	158.387713	10.917060	0.00	149.547489	155.290000	163.090000	198.920000
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	12.036788	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	98.816196	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 corrélation 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
```

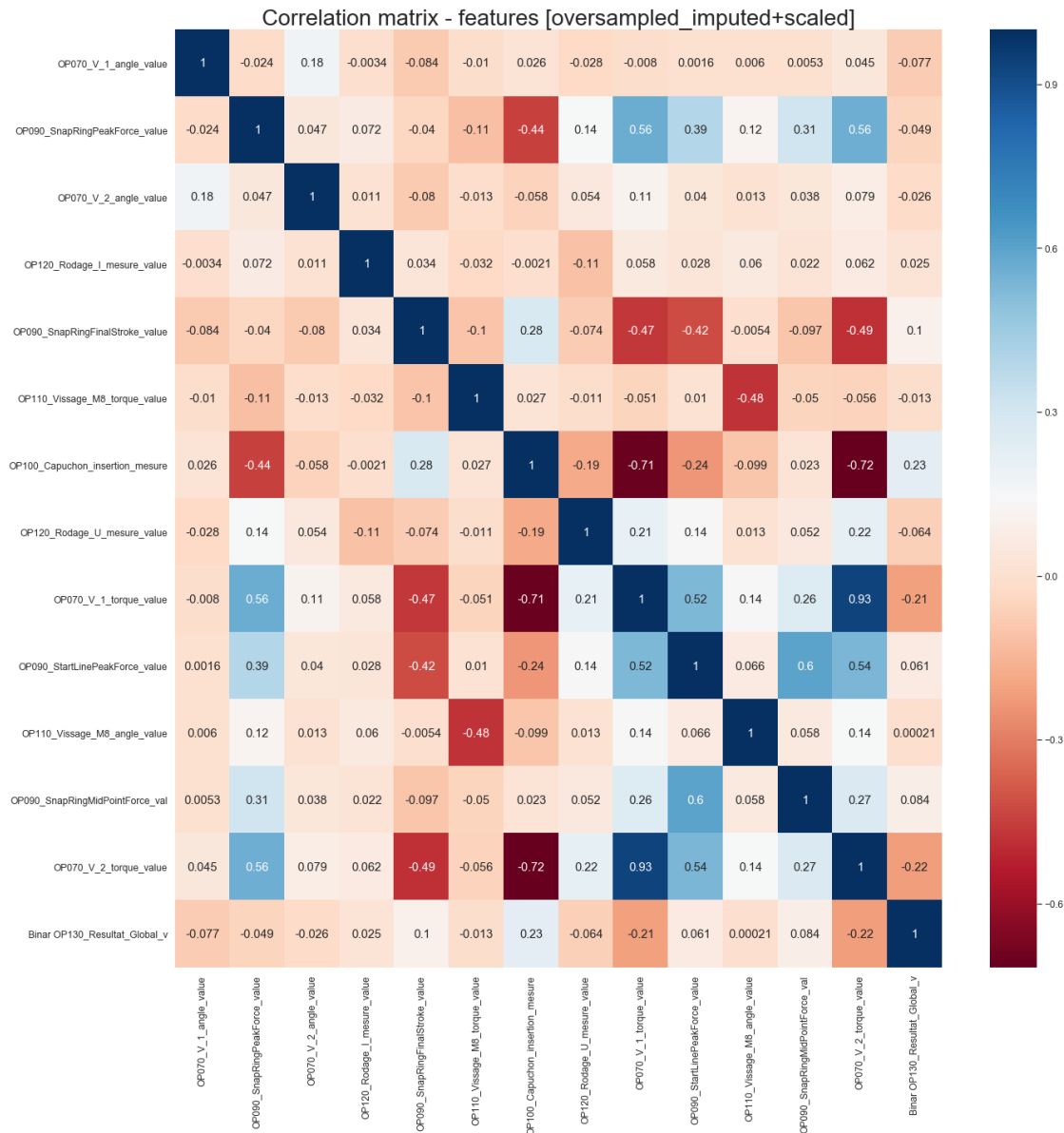
```

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

```

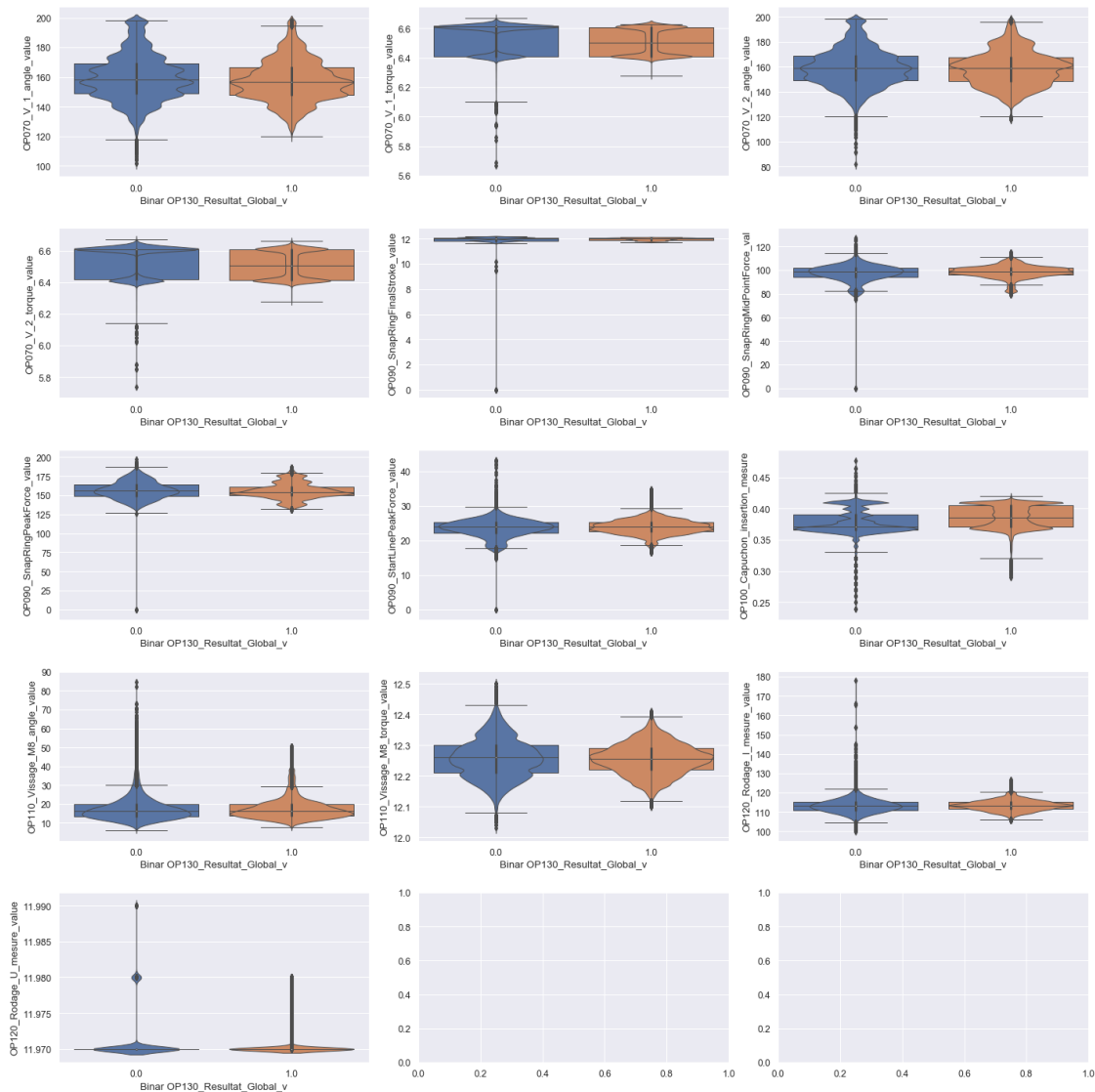
4 - Dessiner la Heatmap

```
ImgUtil.save_df_heatmap_plot(corr_matrix_oversampled,title.format('oversampled_imputed+scaled'))
```



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 accéder 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 + classifieur
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"])
```

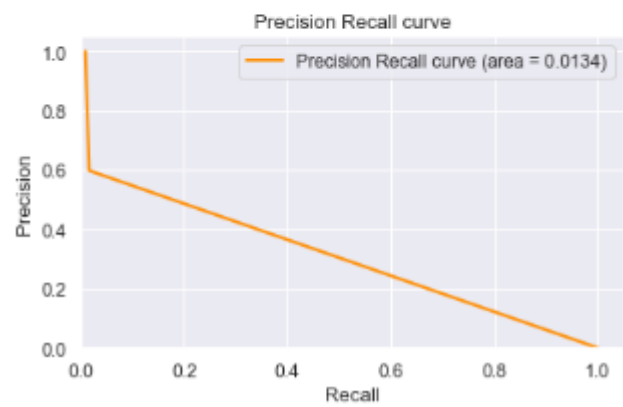
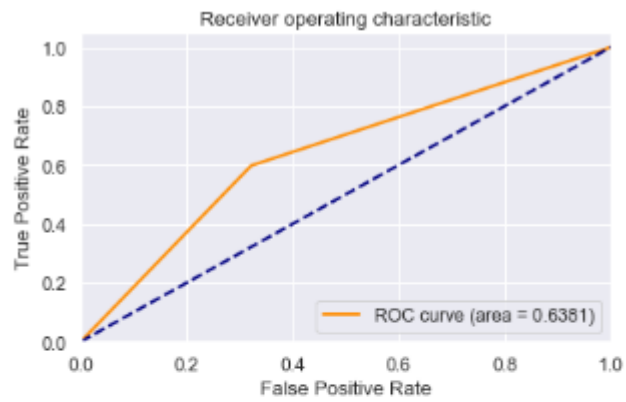
- Model score: 0.6776436504104297
- Accuracy score: 0.6776436504104297
- Balanced accuracy score: 0.6380926205999602 / The balanced accuracy to deal with imbalanced datasets. It is defined as the average of recall obtained on each class.
- Average_precision_score: 0.013370660450719168
- Precision_score: 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 ($\text{base} * \text{hauteur} / 2$) => 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:

```

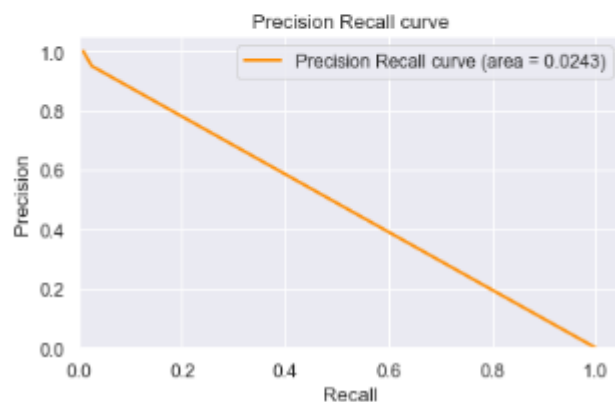
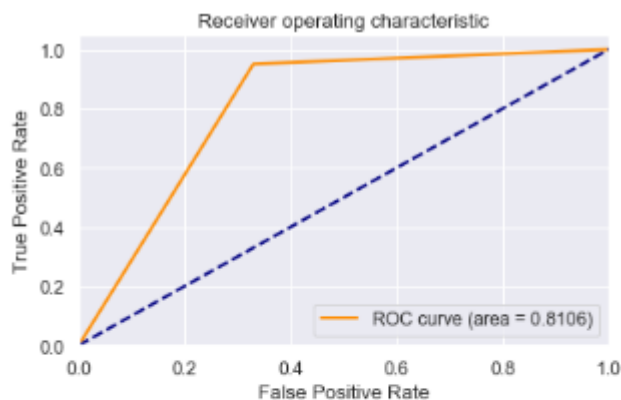
	pre	rec	spe	f1	geo	iba
sup						
0	1.00	0.67	0.95	0.80	0.80	0.62
1	0.03	0.95	0.67	0.05	0.80	0.66
avg / total	0.99	0.67	0.95	0.80	0.80	0.62

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: 0.02995642701525054
- [6738 3525]/[37 55] - P:0.0154 - R:0.5978 - roc_auc:0.6272 - f1:0.0300
- [[6738 3525]
  [ 37  55]]
```

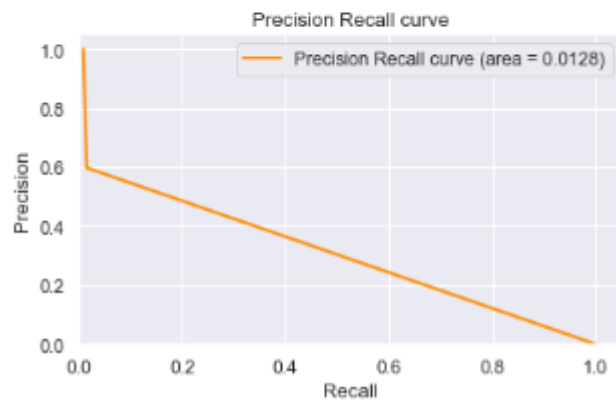
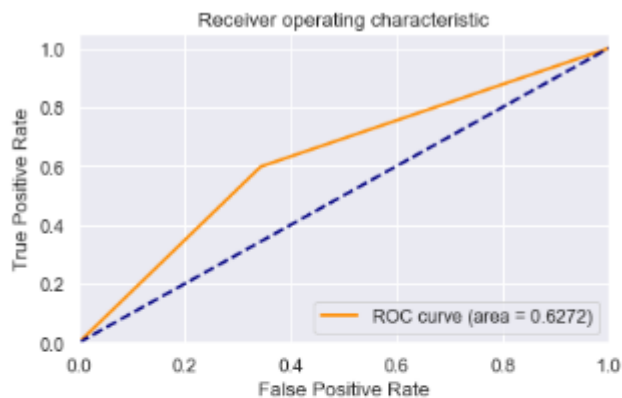
```
- classification_report_imbalanced:
```

		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:

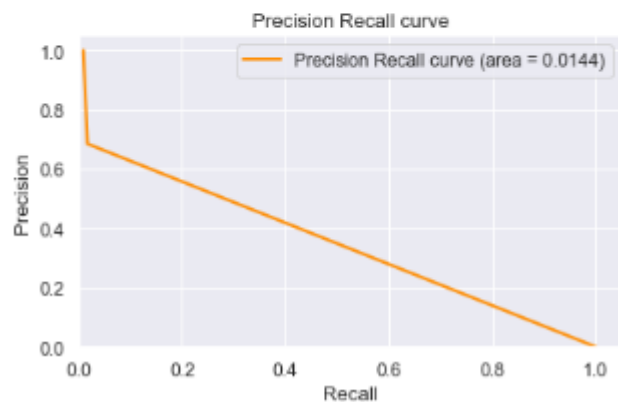
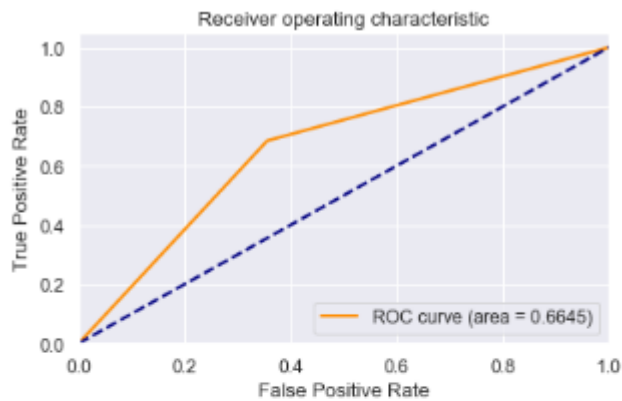
```

		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

```
1 from imblearn.ensemble import BalancedBaggingClassifier, RUSBoostClassifier, BalancedRandomForestClassifier
2 from imblearn.over_sampling import RandomOverSampler, ADASYN, SMOTE, SVMSMOTE, KMeansSMOTE, BorderlineSMOTE
3 from imblearn.over_sampling.base import BaseOverSampler
4 from imblearn.pipeline import Pipeline
5 from sklearn.cluster import MiniBatchKMeans
6 from sklearn.compose import ColumnTransformer
7
8 from sklearn.ensemble._hist_gradient_boosting.gradient_boosting import HistGradientBoostingClassifier
9 from sklearn.impute import SimpleImputer
10 # from sklearn.impute._iterative import IterativeImputer
11 from sklearn.experimental import enable_iterative_imputer # explicitly require this experimental feature
12 from sklearn.impute import IterativeImputer
13 from sklearn.linear_model import LogisticRegression, BayesianRidge
14 from sklearn.preprocessing import Normalizer
15 from sklearn.preprocessing import RobustScaler, MinMaxScaler, label_binarize, StandardScaler
16 from sklearn.svm import SVC
17 import xgboost as xgb
18
19 import pandas as pd
20 import numpy as np
21
22 from valeo.infrastructure.LogManager import LogManager
23 from valeo.infrastructure.tools.DebugPipeline import DebugPipeline
24 from valeo.infrastructure import Const as C
25
26 '''
27 https://github.com/scikit-learn-contrib/imbalanced-learn/tree/master/examples
28 https://towardsdatascience.com/introduction-to-bayesian-linear-regression-e66e60791ea7
29 https://towardsdatascience.com/custom-transformers-and-ml-data-pipelines-with-python-20ea2a7adb65
30 https://jorisvandenbossche.github.io/blog/2018/05/28/scikit-learn-columntransformer/
31 '''
32
33 class ValeoModeler :
34     logger = None
35
36     def __init__(self):
37         logger = LogManager.logger(__name__)
38
39     def build_transformers_pipeline(self, features_dtypes:pd.Series) -> ColumnTransformer:
40         rand_state = 48
41         numerical_features = (features_dtypes == 'int64') | (features_dtypes == 'float64')
42         # categorical_features = ~numerical_features
43         # nan_imputer = SimpleImputer(strategy='mean', missing_values=np.nan, verbose=False)
44         nan_imputer = IterativeImputer(estimator=BayesianRidge(), missing_values=np.nan, max_iter=10,
45 initial_strategy = 'median', add_indicator=True, random_state=rand_state)
46 zeroes_imputer = IterativeImputer(estimator=BayesianRidge(), missing_values=0, max_iter=10,
47 initial_strategy = 'median', add_indicator=True, random_state=rand_state)
48 scaler = RobustScaler(with_centering=True, with_scaling=True, quantile_range=(25.0, 75.0)) #
49 Normalizer() # RobustScaler() #StandardScaler() # RobustScaler(with_centering=True, with_scaling=False) #
50 MinMaxScaler()
51 # scaler = Normalizer(norm='l1')
52 # NB: When using Log transformer: Adopt this transformation -> Log(-2) = -1*(Log(abs(-2)+1))
53 # dbg = DebugPipeline()
54 num_transformers_pipeline = Pipeline([ #('dbg_0', dbg),
55 ('nan_imputer', nan_imputer), # ('dbg_1', dbg),
56 ('zeroes_imputer', zeroes_imputer), # ('dbg_2', dbg),
57 ('scaler', scaler), # ('dbg_3', dbg)
58 ])
59 return ColumnTransformer([('transformers_pipeline', num_transformers_pipeline, numerical_features)],
60 remainder='passthrough')
61
62 # ENS(0.61) without explicit overSampling / test_roc_auc : [0.6719306 0.58851217 0.
63 58250362 0.6094371 0.55757417]
64 BBC = "BBC" # BalancedBaggingClassifier(base_estimator=HGBR, sampling_strategy=1.0, replacement=
65 False, random_state=48)
66 HGBC = "HGBC" # HistGradientBoostingClassifier(max_iter = 8 , max_depth=8, learning_rate=0.35,
67 L2_regularization=500)
68
69 BRFC = "BRFC" # BalancedRandomForestClassifier(n_estimators = 50 , max_depth=20)
70 RUSBoost = "RUSBoost" # RUSBoostClassifier(n_estimators = 8 , algorithm='SAMME.R', random_state=42)
71 KNN = "KNN" # KNeighborsClassifier(3),
72 SVC = "SVC" # SVC(kernel="rbf", C=0.025, probability=True)
73 NuSVC = "NuSVC" # NuSVC(probability=True),
74 RFC = "RFC" # RandomForestClassifier(n_estimators=10, max_depth=10, max_features=10, n_jobs=4))
75 DTC = "DTC" # DecisionTreeClassifier() # so bad
76 ADABOOST = "ADABOOST" # AdaBoostClassifier()
77 GBC = "GBC" # GradientBoostingClassifier()
78 LRC = "LRC" # LogisticRegression(max_iter=500)) # Best for Recall 1
79 XGBC = "XGBC" # xgb.XGBClassifier()
80 # ('classification', GaussianNB()) # 0.5881085402220386
81 # ('classification', ComplementNB()) # 0.523696690978335
82 # ('classification', MultinomialNB()) # 0.523696690978335
83 Imbl_Resampler = "Imbl_Resampler" # ('imbalancer_resampler', self.build_resampler(sampler_type,
84 sampling_strategy='not majority'))
85
86 def build_predictor_pipeline(self, features_dtypes:pd.Series, clfTypes:[str]) -> Pipeline:
87     cls = self.__class__
88     clfs = {
```

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

```
80     cls.HGBC : HistGradientBoostingClassifier(max_iter = 100 , max_depth=10,learning_rate=0.10,
12_regularization=5),
81     cls.BBC : BalancedBaggingClassifier(base_estimator=HistGradientBoostingClassifier(), n_estimators=
50, sampling_strategy='auto', replacement=False, random_state=48),
82
83     # scale_pos_weight
84     # ESTIM:100 depth:20 [6155 4108]/[41 51] - P:0.0123 - R:0.5543 - roc_auc:0.5770 - f1:0.0240 |
85     #.ESTIM:300 depth:10 [6085 4178]/[37 55] - P:0.0130 - R:0.5978 - roc_auc:0.5954 - f1:0.0254
86     # ESTIM:300 depth:15 [6306 3957]/[37 55] - P:0.0137 - R:0.5978 - roc_auc:0.6061 - f1:0.0268
87     # ESTIM:300 depth:20 [6057 4206]/[33 59] - P:0.0138 - R:0.6413 - roc_auc:0.6157 - f1:0.0271 ***
88     # 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
89     # [6127 4136]/[35 57] - P:0.0136 - R:0.6196 - roc_auc:0.6083 - f1:0.0266
90     # [6184 4079]/[37 55] - P:0.0133 - R:0.5978 - roc_auc:0.6002 - f1:0.0260
91     # [6121 4142]/[37 55] - P:0.0131 - R:0.5978 - roc_auc:0.5971 - f1:0.0256
92     # ESTIM:300 depth:30 [6223 4040]/[36 56] - P:0.0137 - R:0.6087 - roc_auc:0.6075 - f1:0.0267
93     # ESTIM:300 depth:40 [6243 4020]/[39 53] - P:0.0130 - R:0.5761 - roc_auc:0.5922 - f1:0.0255
94     #.ESTIM:200 depth:10 [6236 4027]/[39 53] - P:0.0130 - R:0.5761 - roc_auc:0.5919 - f1:0.0254
95     # ESTIM:200 depth:20 [6104 4159]/[34 58] - P:0.0138 - R:0.6304 - roc_auc:0.6126 - f1:0.0269
96     # ESTIM:200 depth:40 [6227 4036]/[37 55] - P:0.0134 - R:0.5978 - roc_auc:0.6023 - f1:0.0263
97     cls.BRFC : BalancedRandomForestClassifier(n_estimators = 300 , max_depth=20, random_state=0),
98
99     cls.RUSBoost : RUSBoostClassifier(n_estimators = 8 , algorithm='SAMME.R', random_state=42),
100     cls.XGBC : xgb.
101         XGBCClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
102             colsample_bynode=1, colsample_bytrees=1, gamma=0,
103             learning_rate=0.1, max_delta_step=0, max_depth=10, #max_depth=3,
104             min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
105             nthread=None, objective='binary:logistic', random_state=0,
106             reg_alpha=0, reg_lambda=1, scale_pos_weight=100, seed=42,
107             silent=None, subsample=1, verbosity=1)
108     }
109     dbg = DebugPipeline()
110     pl= Pipeline([('preprocessor', self.build_transformers_pipeline(features_dtypes)) ,
111         # ('imbalancer_resampler', self.build_resampler(C.smote_over_sampling,sampling_strategy='
minority')), # ('dbg_1', dbg),
112         ('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]
113     ])
114     for i, s in enumerate(pl.steps) :
115         # Ex: 0 -> ('preprocessor', ColumnTransformer( ... + 1 -> ('classifier', BalancedBaggingClassifier(
base_.....
116         print(f"{i} -> {s[0]} / {str(s[1]):>70}")
117     return pl
118
119
120
121     SMOTE is a technique based on nearest neighbors judged by Euclidean Distance between data points in feature
space.
122     random_over_sampling : The most naive strategy is to generate new samples by randomly sampling with
replacement the current available samples.
123     adasyn_over_sampling : Adaptive Synthetic: focuses on generating samples next to the original samples which
are wrongly classified using a k-Nearest Neighbors classifier
124     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
125     ---
126     https://medium.com/towards-artificial-intelligence/application-of-synthetic-minority-over-sampling-technique
-smote-for-imbalanced-data-sets-509ab55cfdaf
127     https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html
128     NB:
129     How to apply SMOTE : Shuffling and Splitting the Dataset into Training and Validation Sets and THEN applying
SMOTE on the Training Dataset.
130     ...
131     def build_resampler(self, sampler_type: str, sampling_strategy='auto', k_neighbors=5) -> BaseOverSampler :
132         rand_state = 48
133         if sampler_type.lower() == C.random_over_sampler :
134             return RandomOverSampler(sampling_strategy=sampling_strategy, random_state=rand_state)
135         elif sampler_type.lower() == C.adasyn_over_sampling :
136             return ADASYN(sampling_strategy=sampling_strategy, random_state=rand_state, n_neighbors=k_neighbors)
137         elif sampler_type.lower() == C.smote_over_sampling :
138             return SMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=k_neighbors)
139         # elif sampler_type.lower() == C.smote_nc_over_sampling : # SMOTE for dataset containing
continuous and categorical features.
140         # return SMOTENC(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=
k_neighbors)
141         elif sampler_type.lower() == C.smote_svm_over_sampling : # Use an SVM algorithm to detect sample to
use for generating new synthetic samples
142             return SVMSMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=
k_neighbors, svm_estimator=SVC())
143         elif sampler_type.lower() == C.smote_kmeans_over_sampling : # Apply a KMeans clustering before to over-
sample using SMOTE
144             return KMeansSMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=
k_neighbors, kmeans_estimator=MiniBatchKMeans(n_clusters=2), cluster_balance_threshold=5)
145         elif sampler_type.lower() == C.smote_bline_over_sampling : # Borderline samples will be detected and
used to generate new synthetic samples.
146             return BorderlineSMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=
k_neighbors, m_neighbors=3)
147         else :
```

```
148         raise ValueError(f"Unexpected argument [{sampler_type}:{sampler_type}] for method '  
compute_sampler_preprocessor'")  
149  
150  
151 # classifiers = [  
152 #     KNeighborsClassifier(3),  
153 #     SVC(kernel="rbf", C=0.025, probability=True),  
154 #     NuSVC(probability=True),  
155 #     DecisionTreeClassifier(),  
156 #     RandomForestClassifier(),  
157 #     AdaBoostClassifier(),  
158 #     GradientBoostingClassifier()  
159 # ]  
160 # for classifier in classifiers:  
161 #     pipe = Pipeline(steps=[('preprocessor', preprocessor),  
162 #                             ('classifier', classifier)])  
163 #     pipe.fit(X_train, y_train)  
164 #     print(classifier)  
165 #     print("model score: %.3f" % pipe.score(X_test, y_test))
```

```

1 from valeo.infrastructure.LogManager import LogManager
2
3 import matplotlib.pyplot as plt
4 from valeo.infrastructure import Const as C
5 from sklearn.metrics import roc_auc_score, precision_recall_curve, roc_curve, average_precision_score
6
7 from valeo.infrastructure.tools.ImgUtil import ImgUtil
8
9
10 class MetricPlotter :
11     logger = None
12
13     def __init__(self):
14         MetricPlotter.logger = LogManager.logger(__name__);
15
16     def plot_roc(self, y_test, y_pred):
17         # y_test = Label_binarize(y_test.values, classes=[0, 1]) # y_test 'Series'
18         # y_pred = Label_binarize(y_pred, classes=[0, 1]) # y_pred 'numpy.ndarray'
19         plt.figure()
20         lw = 2
21         roc = roc_curve(y_test, y_pred)
22         plt.plot(roc[0], roc[1], color='darkorange', lw=lw, label='ROC curve (area = %0.4f)' % roc_auc_score(
y_test, y_pred))
23         plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
24         plt.xlim([0.0, 1.0])
25         plt.ylim([0.0, 1.05])
26         plt.xlabel('False Positive Rate')
27         plt.ylabel('True Positive Rate')
28         plt.title('Receiver operating characteristic')
29         plt.legend(loc="lower right")
30         ImgUtil.save_fig("ROC_curve")
31         plt.show()
32
33     def plot_precision_recall(self, y_test, y_pred):
34         average_precision = average_precision_score(y_test, y_pred)
35         plt.figure()
36         lw = 2
37         pr = precision_recall_curve(y_test, y_pred)
38         plt.plot(pr[0], pr[1], color='darkorange', lw=lw, label='Precision Recall curve (area = %0.4f)' %
average_precision)
39         plt.xlim([0.0, 1.05])
40         plt.ylim([0.0, 1.05])
41         plt.xlabel('Recall')
42         plt.ylabel('Precision')
43         plt.title('Precision Recall curve')
44         plt.legend(loc="upper right")
45         ImgUtil.save_fig("PR_curve")
46         plt.show()
47         #
48         # for i in range(0, len(pr[0]) ) :
49         #     print(f"{i}: ({pr[0][i]}, {pr[1][i]})")
50

```

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

```
1 from imblearn.pipeline import make_pipeline, Pipeline
2 from imblearn.ensemble import BalancedBaggingClassifier
3 from sklearn.linear_model import LogisticRegression
4 from sklearn.metrics import f1_score, auc, roc_auc_score
5 from sklearn.tree import DecisionTreeClassifier
6
7 import pandas as pd
8 from sklearn.model_selection import train_test_split
9
10 from valeo.domain.ValeoPreprocessor import ValeoPreprocessor
11 from valeo.infrastructure import Const as C
12 from valeo.infrastructure.LogManager import LogManager
13
14
15 class ValeoPipeline:
16     logger = None
17
18     def __init__(self):
19         self.preproc = ValeoPreprocessor()
20         ValeoPipeline.logger = LogManager.logger(__name__)
21
22     def pplSmote(self):
23         Pipeline([('column_preprocessor', self.preproc.build_column_preprocessor()),
24                  ('smote_resampler', self.preproc.build_resampler(C.smote_over_sampling))])
25
26     # https://towardsdatascience.com/custom-transformers-and-ml-data-pipelines-with-python-20ea2a7adb65
27     def execute(self, X_df:pd.DataFrame, y_df:pd.DataFrame, sampler_type: str):
28         # setting up testing and training sets
29         X_train, X_test, y_train, y_test = train_test_split(X_df, y_df, test_size=0.25, random_state=48)
30
31         #Create an object of the classifier.
32         bbc = BalancedBaggingClassifier(base_estimator=DecisionTreeClassifier(),
33                                       sampling_strategy='auto',
34                                       replacement=False,
35                                       random_state=0)
36
37         p = Pipeline([('column_preprocessor', self.preproc.build_column_preprocessor()),
38                      # ('smote_resampler', self.preproc.build_resampler(sampler_type)),
39                      # ('classification', LogisticRegression())
40                      # ('classifier', bbc)
41                      ])
42
43         # p.fit_resample(X_train, y_train)
44         p.fit_transform(X_train, y_train)
45         # p.fit(X_train, y_train)
46         #
47         # y_predict = p.predict(X_test)
48         # x = f1_score(y_test, y_predict)
49         # y = 0 # auc(y_test, y_predict)
50         # z = 0 # roc_auc_score(y_test, y_predict)
51         # ValeoPipeline.logger.info(f"F1:{x} - auc:{y} - roc_auc:{z}")
52
53
54 # -----
55 # Exemple Type : Pipeline entier
56 # -----
57 # >>> pca = PCA()
58 # >>> smt = SMOTE(random_state=42)
59 # >>> knn = KNN()
60 # >>> pipeline = Pipeline([('smt', smt), ('pca', pca), ('knn', knn)])
61
62
63
64 # -----
65 # Exemple_1
66 # -----
67 # from sklearn.compose import ColumnTransformer
68 # from sklearn.pipeline import Pipeline
69 #
70 # # 1 - Define Categorical pipe_line
71 # cat_col = ['sex', 'embarked', 'pclass']
72 # cat_pipeline = Pipeline(steps=[
73 #     ("constant-imputer", SimpleImputer(strategy='constant', fill_value='missing')),
74 #     ("ordinal-encoder", OrdinalEncoder()),
75 # ])
76 #
77 # # 2 - Define Numerical pipe_line
78 # num_cols = ['age', 'parch', 'fare']
79 # num_pipeline = SimpleImputer(
80 #     strategy="mean", add_indicator=True,
81 # )
82 #
83 # # 3 - Define Column Transformer
84 # preprocessor = ColumnTransformer(transformers=[
85 #     ("cat-preprocessor", cat_pipeline, cat_col),
86 #     ("num-preprocessor", num_pipeline, num_cols),
87 # ])
88 #
```

```
89 # model = Pipeline(steps=[
90 #     ("preprocessor", preprocessor),
91 #     ("clf", RandomForestClassifier(n_estimators=100))
92 # ])
93 #
94 # _ = model.fit(X_train, y_train)
95 #
96 # (model.named_steps["preprocessor"]
97 #  .named_transformers_["cat-preprocessor"]
98 #  .named_steps["ordinal-encoder"].categories_)
99
100
101
102 # -----
103 # Exemple_2
104 # -----
105 # define the pipelines
106 # cat_pipe = make_pipeline(
107 #     SimpleImputer(strategy='constant', fill_value='missing'),
108 #     OrdinalEncoder(categories=categories)
109 # )
110 # num_pipe = SimpleImputer(strategy='mean')
111 #
112 # preprocessing = ColumnTransformer(
113 #     [ ('cat_preprocessor', cat_pipe, cat_col),
114 #       ('num_preprocessor', num_pipe, num_cols)]
115 # )
```



```

1
2 from imblearn.metrics._classification import classification_report_imbalanced
3 # https://imbalanced-Learn.readthedocs.io/en/stable/api.html#module-imblearn.pipeline
4 from sklearn.base import BaseEstimator
5 from sklearn.metrics import f1_score, auc, roc_auc_score, confusion_matrix, classification_report, \
6     precision_recall_curve, precision_recall_fscore_support, roc_curve, plot_precision_recall_curve, \
7     average_precision_score, precision_score, recall_score, accuracy_score, balanced_accuracy_score
8 # from sklearn.impute import SimpleImputer
9 from sklearn.model_selection import cross_validate, StratifiedKFold, GridSearchCV, RandomizedSearchCV
10
11 import pandas as pd
12
13 from valeo.domain.MetricPlotter import MetricPlotter
14 from valeo.domain.ValeoModeler import ValeoModeler
15 from valeo.infrastructure.tools.DfUtil import DfUtil
16 from valeo.infrastructure.LogManager import LogManager
17 from valeo.infrastructure import Const as C
18
19 import xgboost as xgb
20
21
22 class ValeoPredictor :
23     logger = None
24
25     def __init__(self):
26         ValeoPredictor.logger = LogManager.logger(__name__)
27         self.modeler = ValeoModeler()
28         self.metricPlt = MetricPlotter()
29
30
31     def fit_cv_grid_search(self, X:pd.DataFrame, y:pd.DataFrame, clfTypes:[str] , n_splits=5) -> ([BaseEstimator
32 ], dict): # (estimator, cv_results)
33         model = self.modeler.build_predictor_pipeline(X.dtypes, clfTypes) # sampler_type)
34         CV = StratifiedKFold(n_splits=n_splits) # , random_state=48, shuffle=True
35         # HGBC
36         # param_grid = {
37         #     'classifier__n_estimators': [3, 5, 10, 20, 50],
38         #     'classifier__base_estimator__l2_regularization': [5, 50, 100, 50],
39         #     'classifier__base_estimator__max_iter' : [100],
40         #     'classifier__base_estimator__max_depth' : [10,50,10]
41         # }
42         # BRFC
43         param_grid = {
44             'classifier__n_estimators': [250,300],
45             'classifier__max_depth': [15,20,25],
46             'classifier__max_features' : ['auto',13]
47         }
48         grid = GridSearchCV(model, param_grid=param_grid, n_jobs=-1, cv=CV) # if is_grid else
49         grid.fit(X, y)
50         print(f"Best Estimator: {grid.best_estimator_}")
51         df_results = pd.DataFrame(grid.cv_results_)
52         # columns_to_keep = ['param_clf__max_depth', 'param_clf__n_estimators', 'mean_test_score', '
53         std_test_score',]
54         # df_results = df_results[columns_to_keep]
55         DfUtil.write_df_csv( df_results.sort_values(by='mean_test_score', ascending=False), C.ts_pathname([C.
56 rootReports(), 'grid_search_cv.csv']) )
57
58     def fit_cv_randomized_search(self, X:pd.DataFrame, y:pd.DataFrame, clfTypes:[str] , n_splits=5) -> ([
59 BaseEstimator], dict): # (estimator, cv_results)
60         model = self.modeler.build_predictor_pipeline(X.dtypes, clfTypes) # sampler_type)
61         CV = StratifiedKFold(n_splits=n_splits) # , random_state=48, shuffle=True
62         # HGBC
63         # param_grid = {
64         #     'classifier__n_estimators': [3, 5, 10, 20, 50],
65         #     'classifier__base_estimator__l2_regularization': [5, 50, 100, 50],
66         #     'classifier__base_estimator__max_iter' : [100],
67         #     'classifier__base_estimator__max_depth' : [10,50,10]
68         # }
69         grid = RandomizedSearchCV(model, param_distributions=param_grid, n_jobs=-1, cv=CV) # if is_grid else
70         grid.fit(X, y)
71         df_results = pd.DataFrame(grid.cv_results_)
72         DfUtil.write_df_csv( df_results.sort_values(by='mean_test_score', ascending=False), C.ts_pathname([C.
73 rootReports(), 'grid_search_cv.csv']) )
74
75     def print_model_params_keys(self, model:BaseEstimator):
76         for param in model.get_params().keys():
77             print(param)
78
79     # 1 - Fit without any Cross Validation
80     def fit_and_plot(self, X_train:pd.DataFrame, y_train:pd.DataFrame, X_test:pd.DataFrame, y_test:pd.DataFrame
81 , clfTypes:[str]) -> BaseEstimator:
82         # Q1 = X_train.quantile(0.25)
83         # Q3 = X_train.quantile(0.75)
84         # IQR = Q3 - Q1
85         # # print(IQR)
86         # to_remove = ((X_train < (Q1 - 1.5 * IQR)) | (X_train > (Q3 + 1.5 * IQR))).any(axis=1)

```

File - C:\EXED\Training__VALEO\src\valeo\domain\ValeoPredictor.py

```
83     # y_train = y_train.drop(axis=0, index=X_train[to_remove].index)
84     # X_train = X_train[~to_remove]
85     #
86     fitted_model = self.fit(X_train, y_train, clfTypes)
87     # print(f"Type:{type(fitted_model)} - {fitted_model.get_params()}")
88     # self.print_model_params_keys(fitted_model)
89     self.predict_and_plot(fitted_model, X_test, y_test)
90     return fitted_model
91
92 def fit(self, X_train:pd.DataFrame, y_train:pd.DataFrame, clfTypes:[str]) -> BaseEstimator:
93     model = self.modeler.build_predictor_pipeline(X_train.dtypes, clfTypes)
94     return model.fit(X_train, y_train)
95
96 ''' 2 - Fit with Cross Validation
97 NB :
98 a - roc-auc-avo + roc-auc-ovr :
99     https://stackoverflow.com/questions/59453363/what-is-the-difference-of-roc-auc-values-in-sklearn
100     roc_auc is the only one suitable for binary classification. The weighted, ovr and ovo are use for
multi-class problems
101
102     b - Micro-Average + Macro-Average (for Precision / Recall / F1) :
103     http://rushdshams.blogspot.com/2011/08/micro-and-macro-average-of-precision.html
104     https://datascience.stackexchange.com/questions/15989/micro-average-vs-macro-average-performance-in-
a-multiclass-classification-settin
105     Ex: Micro-P = (TP1 + TP2) / ( TP1 + FP1 + TP2 + F2)
106         Macro-P = (P1 + P2) / 2
107     Suitability:
108     . Macro-average method can be used when you want to know how the system performs overall across the
sets of data
109     . Micro-average method can be a useful measure when your dataset varies in size.
110
111     c - How can we report 'confusion matrix' while using 'cross_validate' ?
112     https://stackoverflow.com/questions/40057049/using-confusion-matrix-as-scoring-metric-in-cross-
validation-in-scikit-learn
113     c1. Either use 'cross_val_predict' and deduce confusion-matrix:
114         y_pred = cross_val_predict(clf, X, y, cv=10)
115         conf_mat = confusion_matrix(y_test, y_pred)
116         BUT BEWARE: Passing these predictions into an evaluation metric may not be a valid way to
measure generalization performance.
117         Results can differ from cross_validate and cross_val_score unless all tests sets
have equal size and the metric decomposes over samples.
118     c2. If you want to obtain confusion matrices for multiple evaluation runs (such as cross validation
) you have to do this by hand:
119         conf_matrix_list_of_arrays = []
120         kf = cross_validation.KFold(len(y), n_folds=5)
121         for train_index, test_index in kf:
122             X_train, X_test = X[train_index], X[test_index] # Panda-Column index 'train_index' are of
type 'numpy array'
123             y_train, y_test = y[train_index], y[test_index]
124             #
125             model.fit(X_train, y_train)
126             conf_matrix = confusion_matrix(y_test, model.predict(X_test))
127             conf_matrix_list_of_arrays.append(conf_matrix)
128
129         On the end you can calculate your mean of list of numpy arrays (confusion matrices) with:
130         mean_of_conf_matrix_arrays = np.mean(conf_matrix_list_of_arrays, axis=0)
131
132 def fit_cv(self, X:pd.DataFrame, y:pd.DataFrame, clfTypes:[str], n_splits=5) -> ([BaseEstimator], dict):
# (estimator, cv_results)
133     model = self.modeler.build_predictor_pipeline(X.dtypes, clfTypes)
134     CV = StratifiedKFold(n_splits=n_splits) # , random_state=48, shuffle=True
135     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)
136     fitted_estimators = []
137     for key in cv_results.keys() :
138         if str(key) != "estimator" :
139             print(f"{key} : {cv_results[key]}")
140             fitted_estimators.append(cv_results[key])
141     return fitted_estimators, cv_results
142
143 '''
144 - Print metrics
145 - Print report
146 - Plot ROC : TP vs FP
147 - Plot AUC : Precision vs Recall
148
149 def predict_and_plot(self, fitted_model: BaseEstimator, X_test:pd.DataFrame, y_test:pd.DataFrame):
150     y_pred = fitted_model.predict(X_test)
151     #
152     print(f"- Model score: {fitted_model.score(X_test, y_test)}")
153     print(f"- Accuracy score: {accuracy_score(y_test, y_pred)}")
154     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.")
155     # print(f"- auc : {auc(y_test, y_pred)}") # ValueError: x is neither increasing nor decreasing : [0 0
0 ... 0 0 0]
156     print(f"- Average_precision_score: {average_precision_score(y_test, y_pred)}")
157     print(f"- Precision_score: {precision_score(y_test, y_pred)}")
```

File - C:\EXED\Training__VALEO\src\valeo\domain\ValeoPredictor.py

```
158     print(f"- Recall score: {recall_score(y_test, y_pred)}")
159     print(f"- Roc_auc_score: {roc_auc_score(y_test, y_pred)}")
160     print(f"- F1 score: {f1_score(y_test, y_pred)}")
161     m = confusion_matrix(y_test, y_pred)
162     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}")
163     print(f"- {confusion_matrix(y_test, y_pred)}")
164     print(f"- classification_report_imbalanced:\n{classification_report_imbalanced(y_test, y_pred)}")
165     print(f"- classification_report:\n{classification_report(y_test, y_pred)}")
166     print(f"- precision_recall_curve: {precision_recall_curve(y_test, y_pred)}")
167     print(f"- precision_recall_fscore_support: {precision_recall_fscore_support(y_test, y_pred)}")
168     print(f"- roc_curve: {roc_curve(y_test, y_pred)}")
169     #
170     self.metricPlt.plot_roc(y_test, y_pred)
171     self.metricPlt.plot_precision_recall(y_test, y_pred)
172     # self.plot_roc(y_test, y_pred)
173     # self.plot_precision_recall(y_test, y_pred)
174
175
176
177 # https://medium.com/towards-artificial-intelligence/application-of-synthetic-minority-over-sampling-technique-smote-for-imbalanced-data-sets-509ab55cfdaf
178 # from sklearn.ensemble import GradientBoostingClassifier
179 # from sklearn.model_selection import GridSearchCV
180 # parameters = {'n_estimators':[100,150,200,250,300,350,400,450,500],
181 #               'max_depth':[3,4,5]}
182 # clf= GradientBoostingClassifier()
183 # grid_search = GridSearchCV(param_grid = parameters, estimator = clf,
184 #                             verbose = 3)
185 # grid_search_2 = grid_search.fit(X_train,y_train)
186
187 # GOOGLE ON: classifier over sampled imbalanced dataset
188 # https://sci2s.ugr.es/imbalanced : Tres interessant****
189 # https://www.datacamp.com/community/tutorials/diving-deep-imbalanced-data : Tres interessant****
190 # ---
191 # https://journalofbigdata.springeropen.com/articles/10.1186/s40537-017-0108-1 : Tres interessant****
192 # xperimental evaluation
193 # 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.
194
195 # GOOGLE ON: scikit Learn imbalanced dataset resampling type cross validation shuffle
196 # https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets
197 # https://machinelearningmastery.com/cross-validation-for-imbalanced-classification/
198 # CV = ShuffleSplit(n_splits=10, test_size=0.25, random_state=48)
199 # https://www.alfredo.motta.name/cross-validation-done-wrong/
200
201 # https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/
202 # https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6
203
204 # http://www.cs.nthu.edu.tw/~shwu/courses/ML/Labs/08_CV_Ensembling/08_CV_Ensembling.html
205 # https://github.com/arraylayer/ML-Project
206
207 # https://www.kaggle.com/shiqbal/first-data-exploration/notebook applied on Porto Seguro's
208
209
210 ''' TODO :
211
212 -> Faire Ressortir l'importance et la contribution de chaque feature:
213     https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance.html
214
215 -> Essayer ces scénarios de modélisation:
216     scen1: Oversampling + LogisticR
217     scen2: BaggingClassifier + Histo
218
219 '''
```

File - C:\EXED\Training__VALEO\src\valeo\domain\ValeoPreprocessor.py

```
1 from imblearn.over_sampling import RandomOverSampler, SMOTE, SMOTENC, SVMSMOTE, KMeansSMOTE, BorderlineSMOTE,
  ADASYN
2 from imblearn.over_sampling.base import BaseOverSampler
3 # from sklearn.experimental import enable_iterative_imputer # explicitly require this experimental feature
4 # from sklearn.impute import IterativeImputer
5 from sklearn.impute import SimpleImputer # now you can import normally from sklearn.impute
6 from sklearn.linear_model import BayesianRidge
7 from sklearn.model_selection import train_test_split
8 from sklearn.preprocessing import RobustScaler, StandardScaler, MinMaxScaler
9
10 from sklearn.compose import ColumnTransformer, make_column_transformer
11 from imblearn.pipeline import make_pipeline, Pipeline
12
13 import pandas as pd
14 import numpy as np
15
16 from valeo.infrastructure import Const as C
17 from valeo.infrastructure.LogManager import LogManager
18
19 from valeo.infrastructure.tools.DebugPipeline import DebugPipeline
20
21
22 class ValeoPreprocessor:
23     logger = None
24
25     def __init__(self):
26         ValeoPreprocessor.logger = LogManager.logger(__name__)
27
28         # def build_iterative_preprocessor(self) -> ColumnTransformer:
29         #     imp_cols = [C.OP100_Capuchon_insertion_mesure]
30         #     it_imp = IterativeImputer(estimator=BayesianRidge(), max_iter=10, initial_strategy = 'median',
31         add_indicator=True, random_state=48)
32         #     return ColumnTransformer([('iterative_imputer', it_imp, imp_cols)] )
33
34     def build_column_preprocessor(self) -> Pipeline: # ColumnTransformer:
35         rand_state = 48
36         # 1 - IterativeImputer : models each feature with missing values as a function of other features, and
37         # uses that estimate for imputation
38         # imputer_pipe = Pipeline( IterativeImputer(estimator=BayesianRidge(), missing_values=[np.nan, 0],
39         max_iter=10, initial_strategy = 'median') )
40
41         # imputer_nan_values = IterativeImputer(estimator=BayesianRidge(), missing_values=np.nan, max_iter=10,
42         initial_strategy = 'median', add_indicator=True, random_state=rand_state)
43         # imputer_nan_values = IterativeImputer(estimator=BayesianRidge(), missing_values=np.nan, max_iter=10,
44         initial_strategy = 'median', add_indicator=False, random_state=rand_state)
45         imputed_nan_cols = [C.OP100_Capuchon_insertion_mesure] # columns having too much missing values
46         #
47         # imputer_zeroes_values = IterativeImputer(estimator=BayesianRidge(), missing_values=0, max_iter=10,
48         initial_strategy = 'median', add_indicator=False, random_state=rand_state)
49         imputed_zeroes_cols = [C.OP100_Capuchon_insertion_mesure, C.OP090_StartLinePeakForce_value, C.
50         OP090_SnapRingMidPointForce_val, # columns equals to 0 for a few rows
51         C.OP090_SnapRingPeakForce_value, C.OP090_SnapRingFinalStroke_value]
52
53         # 2 - Scale features using statistics that are robust to outliers.
54         scaler = StandardScaler() # MinMaxScaler() # StandardScaler() # RobustScaler(with_centering=True,
55         with_scaling=False)
56         scaled_cols = [C.OP070_V_1_angle_value, C.OP070_V_1_torque_value,
57         C.OP070_V_2_angle_value, C.OP070_V_2_torque_value,
58         C.OP090_StartLinePeakForce_value, C.OP090_SnapRingMidPointForce_val,
59         C.OP090_SnapRingPeakForce_value, C.OP090_SnapRingFinalStroke_value,
60         C.OP100_Capuchon_insertion_mesure,
61         C.OP110_Vissage_M8_angle_value, C.OP110_Vissage_M8_torque_value,
62         C.OP120_Rodage_I_mesure_value, C.OP120_Rodage_U_mesure_value]
63
64         dbg = DebugPipeline()
65         return Pipeline([ ('dbg_0', dbg, scaled_cols),
66         ('imputer_preprocessor_nan', SimpleImputer(strategy='mean', missing_values=np.
67         nan, verbose=True), imputed_nan_cols),
68         ('dbg_1', dbg, scaled_cols),
69         ('imputer_preprocessor_zeroes', SimpleImputer(strategy='mean', missing_values=
70         0.0, verbose=True), imputed_zeroes_cols),
71         ('dbg_2', dbg, scaled_cols),
72         ('scaler_preprocessor', scaler, scaled_cols),
73         ('dbg_3', dbg, scaled_cols),
74         # #
75         # ('imputer_preprocessor_nan_bis', SimpleImputer(strategy='mean'), scaled_cols
76         ),
77         # ('dbg_1_bis', DebugPLine(), scaled_cols)
78         ])
79         # , remainder='passthrough')
80
81     def execute(self, X_train:pd.DataFrame):
82         imputed_nan_cols = [C.OP100_Capuchon_insertion_mesure]
83         imputed_zeroes_cols = [C.OP100_Capuchon_insertion_mesure, C.OP090_StartLinePeakForce_value, C.
84         OP090_SnapRingMidPointForce_val, # columns equals to 0 for a few rows
85         C.OP090_SnapRingPeakForce_value, C.OP090_SnapRingFinalStroke_value]
86         scaled_cols = [C.OP070_V_1_angle_value, C.OP070_V_1_torque_value,
```

```

76         C.OP070_V_2_angle_value, C.OP070_V_2_torque_value,
77         C.OP090_StartLinePeakForce_value, C.OP090_SnapRingMidPointForce_val,
78         C.OP090_SnapRingPeakForce_value, C.OP090_SnapRingFinalStroke_value,
79         C.OP100_Capuchon_insertion_mesure,
80         C.OP110_Vissage_M8_angle_value, C.OP110_Vissage_M8_torque_value,
81         C.OP120_Rodage_I_mesure_value, C.OP120_Rodage_U_mesure_value]
82     # DebugPipeline.counter += 10
83     d = DebugPipeline()
84     #
85     d.fit_transform(X_train[scaled_cols])
86     s = SimpleImputer(strategy='mean', missing_values=np.nan, verbose=True)
87     X_train[imputed_nan_cols] = s.fit_transform(X_train[imputed_nan_cols])
88     d.fit_transform(X_train[scaled_cols])
89     #
90     s = SimpleImputer(strategy='mean', missing_values=0.0, verbose=True)
91     X_train[imputed_zeroes_cols] = s.fit_transform(X_train[imputed_zeroes_cols])
92     d.fit_transform(X_train[scaled_cols])
93     #
94     scaler = MinMaxScaler() # StandardScaler() # RobustScaler(with_centering=True, with_scaling=False)
95     X_train[scaled_cols] = scaler.fit_transform(X_train[scaled_cols])
96     d.fit_transform(X_train[scaled_cols])
97
98
99
100
101     def build_column_preprocessor(self) -> ColumnTransformer:
102         rand_state = 48
103         # 1 - IterativeImputer : models each feature with missing values as a function of other features, and
104         uses that estimate for imputation
105         imputer_pipe = make_pipeline( IterativeImputer(estimator=BayesianRidge(), missing_values=np.nan,
106 max_iter=10, initial_strategy = 'median', add_indicator=True, random_state=rand_state) )
107         # imputer_pipe = Pipeline( IterativeImputer(estimator=BayesianRidge(), missing_values=[np.nan, 0],
108 max_iter=10, initial_strategy = 'median') )
109         imputed_cols = [C.OP100_Capuchon_insertion_mesure] # columns having too much missing values
110         # C.OP090_StartLinePeakForce_value, C.OP090_SnapRingMidPointForce_val, # columns equals
111         to 0 for a few rows
112         # C.OP090_SnapRingPeakForce_value, C.OP090_SnapRingFinalStroke_value]
113
114         # 2 - Scale features using statistics that are robust to outliers.
115         scaler_pipe = make_pipeline(RobustScaler(with_centering=True, with_scaling=False))
116         scaled_cols = [C.OP070_V_1_angle_value, C.OP070_V_1_torque_value,
117 C.OP070_V_2_angle_value, C.OP070_V_2_torque_value,
118 C.OP090_StartLinePeakForce_value, C.OP090_SnapRingMidPointForce_val,
119 C.OP090_SnapRingPeakForce_value, C.OP090_SnapRingFinalStroke_value,
120 C.OP100_Capuchon_insertion_mesure,
121 C.OP110_Vissage_M8_angle_value, C.OP110_Vissage_M8_torque_value,
122 C.OP120_Rodage_I_mesure_value, C.OP120_Rodage_U_mesure_value,]
123
124         return ColumnTransformer([('imputer_preprocessor', imputer_pipe, imputed_cols),
125 ('scaler_preprocessor', scaler_pipe, scaled_cols)] )
126
127
128
129
130     SMOTE is a technique based on nearest neighbors judged by Euclidean Distance between data points in feature
131     space.
132     random_over_sampling : The most naive strategy is to generate new samples by randomly sampling with
133     replacement the current available samples.
134     adasyn_over_sampling : Adaptive Synthetic: focuses on generating samples next to the original samples which
135     are wrongly classified using a k-Nearest Neighbors classifier
136     smote_over_sampling : Synth Minority Oversampling Techn: will not make any distinction between easy and
137     hard samples to be classified using the nearest neighbors rule
138     ---
139     https://medium.com/towards-artificial-intelligence/application-of-synthetic-minority-over-sampling-technique
140     -smote-for-imbalanced-data-sets-509ab55cfdaf
141     https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html
142     NB:
143     How to apply SMOTE : Shuffling and Splitting the Dataset into Training and Validation Sets and THEN applying
144     SMOTE on the Training Dataset.
145
146     def build_resampler(self, sampler_type: str, sampling_strategy='auto', k_neighbors=5) -> BaseOverSampler :
147         rand_state = 48
148         if sampler_type.lower() == C.random_over_sampler :
149             return RandomOverSampler(sampling_strategy=sampling_strategy, random_state=rand_state)
150         elif sampler_type.lower() == C.adasyn_over_sampling :
151             return ADASYN(sampling_strategy=sampling_strategy, random_state=rand_state, n_neighbors=k_neighbors)
152         elif sampler_type.lower() == C.smote_over_sampling :
153             return SMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=k_neighbors)
154         # elif sampler_type.lower() == C.smote_nc_over_sampling : # SMOTE for dataset containing
155         continuous and categorical features.
156         # return SMOTENC(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=
157         k_neighbors)
158         elif sampler_type.lower() == C.smote_svm_over_sampling : # Use an SVM algorithm to detect sample to
159         use for generating new synthetic samples
160             return SVMSMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=
161         k_neighbors)
162         elif sampler_type.lower() == C.smote_kmeans_over_sampling : # Apply a KMeans clustering before to over-
163         sample using SMOTE
164             return KMeansSMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=

```

```

148 k_neighbors)
149     elif sampler_type.lower() == C.smote_bline_over_sampling : # Borderline samples will be detected and
    used to generate new synthetic samples.
150     return BorderlineSMOTE(sampling_strategy=sampling_strategy, random_state=rand_state, k_neighbors=
k_neighbors)
151     else :
152         raise ValueError(f"Unexpected argument [sampler_type:{sampler_type}] for method '
compute_sampler_preprocessor'")
153
154
155
156 # -----
157 # Exemple Type : Pipeline entier
158 # -----
159 # >>> pca = PCA()
160 # >>> smt = SMOTE(random_state=42)
161 # >>> knn = KNN()
162 # >>> pipeline = Pipeline([('smt', smt), ('pca', pca), ('knn', knn)])
163
164
165
166 # -----
167 # Exemple_1
168 # -----
169 from sklearn.compose import ColumnTransformer
170 from sklearn.pipeline import Pipeline
171 #
172 # # 1 - Define Categorical pipe_line
173 # cat_col = ['sex', 'embarked', 'pclass']
174 # cat_pipeline = Pipeline(steps=[
175 #     ("constant-imputer", SimpleImputer(strategy='constant', fill_value='missing')),
176 #     ("ordinal-encoder", OrdinalEncoder()),
177 # ])
178 #
179 # # 2 - Define Numerical pipe_line
180 # num_cols = ['age', 'parch', 'fare']
181 # num_pipeline = SimpleImputer(
182 #     strategy="mean", add_indicator=True,
183 # )
184 #
185 # # 3 - Define Column Transformer
186 # preprocessor = ColumnTransformer(transformers=[
187 #     ("cat-preprocessor", cat_pipeline, cat_col),
188 #     ("num-preprocessor", num_pipeline, num_cols),
189 # ])
190 #
191 # model = Pipeline(steps=[
192 #     ("preprocessor", preprocessor),
193 #     ("clf", RandomForestClassifier(n_estimators=100))
194 # ])
195 #
196 # _ = model.fit(X_train, y_train)
197 #
198 # (model.named_steps["preprocessor"]
199 # .named_transformers_["cat-preprocessor"]
200 # .named_steps["ordinal-encoder"].categories_)
201
202
203
204 # -----
205 # Exemple_2
206 # -----
207 # define the pipelines
208 # cat_pipe = make_pipeline(
209 #     SimpleImputer(strategy='constant', fill_value='missing'),
210 #     OrdinalEncoder(categories=categories)
211 # )
212 # num_pipe = SimpleImputer(strategy='mean')
213 #
214 # preprocessing = ColumnTransformer(
215 #     [('cat_preprocessor', cat_pipe, cat_col),
216 #      ('num_preprocessor', num_pipe, num_cols)]
217 # )

```

File - C:\EXED\Training__VALEO\src\valeo\resources\valeo.yaml

```
1 version: 1
2 data:
3   input: ../data/input
4   output: ../data/output
5
```

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

```
1 version: 1
2 formatters:
3   standard:
4     format: '%(asctime)s - %(levelname)s - %(name)s - %(message)s'
5   # format: '%(asctime)s - %(levelname)s - %(module)s - %(message)s'
6   verbose:
7     format: '%(asctime)s - %(levelname)s <PID %(process)d: %(processName)s> %(module)s.%(funcName)s(): %(message)s'
8   ,
9 handlers:
10  console:
11    class: logging.StreamHandler
12    level: DEBUG
13    formatter: standard
14    stream: ext://sys.stdout
15  # propagate: yes
16  valeo_log:
17    class: logging.FileHandler
18    filename: C:/EXED/Training/__VALEO/log/valeo.log
19    mode: w
20    level: DEBUG
21    formatter: standard
22  # propagate: yes
23  # errors:
24  #   class: logging.FileHandler
25  #   filename: mplog-errors.log
26  #   mode: w
27  #   level: ERROR
28  #   formatter: detailed
29 #Loggers:
30 # simpleExample:
31 #   level: DEBUG
32 #   handlers: [console]
33 #   propagate: no
34 root:
35   level: DEBUG
36   handlers: [console, valeo_log]
37   formatter: standard
```


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

```
1 # ENVIRONMENT keys used to refer to configuration files
2 ENV_KEY_CONFIG_FILE_PATHNAME = '__VALEO__APP_CONFIG_FILE_PATHNAME' # ex: SET __VALEO__APP_CONFIG_FILE_PATHNAME
   =...../valeo.yaml'
3 ENV_KEY_LOG_FILE_PATHNAME    = '__VALEO__APP_LOG_FILE_PATHNAME'    # ex: SET __VALEO__APP_LOG_FILE_PATHNAME
   =...../logging.yaml'
4 #
5 # Symbolic name of configuration files
6 APP_DEFAULT_CONFIG_FILE = 'valeo.yaml'
7 APP_DEFAULT_LOG_FILE    = 'logging.yaml'
8 #
9 # Valeo Dataset columns names
10 PROC_TRACEINFO         = 'PROC_TRACEINFO'
11 OP070_V_1_angle_value  = 'OP070_V_1_angle_value'
12 OP070_V_1_torque_value = 'OP070_V_1_torque_value'
13 OP070_V_2_angle_value  = 'OP070_V_2_angle_value'
14 OP070_V_2_torque_value = 'OP070_V_2_torque_value'
15 OP090_StartLinePeakForce_value = 'OP090_StartLinePeakForce_value'
16 OP090_SnapRingMidPointForce_val = 'OP090_SnapRingMidPointForce_val'
17 OP090_SnapRingPeakForce_value = 'OP090_SnapRingPeakForce_value'
18 OP090_SnapRingFinalStroke_value = 'OP090_SnapRingFinalStroke_value'
19 OP100_Capuchon_insertion_mesure = 'OP100_Capuchon_insertion_mesure'
20 OP110_Vissage_M8_angle_value = 'OP110_Vissage_M8_angle_value'
21 OP110_Vissage_M8_torque_value = 'OP110_Vissage_M8_torque_value'
22 OP120_Rodage_I_mesure_value = 'OP120_Rodage_I_mesure_value'
23 OP120_Rodage_U_mesure_value = 'OP120_Rodage_U_mesure_value'
24 Binar_OP130_Resultat_Global_v = 'Binar_OP130_Resultat_Global_v'
25 #
26 # Imbalanced resampling type
27
28 random_over_sampler = 'random_over_sampler' # The most naive strategy is to generate new samples by randomly
   sampling with replacement the current available samples.
29 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
30 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
31 smote_nc_over_sampling = 'smote_nc_over_sampling'
32 smote_svm_over_sampling = 'smote_svm_over_sampling'
33 smote_kmeans_over_sampling = 'smote_kmeans_over_sampling'
34 smote_bline_over_sampling = 'smote_bline_over_sampling'
35
36
37 import os
38 from datetime import datetime
39 # timestamp : none / suffix / prefix
40 ts_none = 0
41 ts_sfix = 1
42 ts_pfix = 2
43
44 def rootProject() -> str :
45     return os.path.join(os.path.abspath(os.path.dirname(__file__)), '..', '..', '..') # this_folder = D:/
   Training.git/trunk/__VALEO/src/valeo/infrastructure
46
47 def rootSrc() -> str :
48     return os.path.join(rootProject(), 'src' )
49
50 def rootData() -> str :
51     return os.path.join(rootProject(), 'data' )
52
53 def rootDataTrain() -> str :
54     return os.path.join(rootData(), 'train' )
55
56 def rootDataTest() -> str :
57     return os.path.join(rootData(), 'test' )
58
59 def rootImages() -> str :
60     return os.path.join(rootProject(), 'images' )
61
62 def rootReports() -> str :
63     return os.path.join(rootProject(), 'reports' )
64
65 def rootResources() -> str :
66     return os.path.join(rootProject(), 'src', 'valeo', 'resources')
67
68 def ts_pathanne(pathAsStrList : [], ts_type=ts_sfix) -> str:
69     if not isinstance(pathAsStrList, list) :
70         pathAsStrList = [pathAsStrList]
71         fname_with_ext = os.path.splitext(pathAsStrList[-1])
72         return os.path.join(pathAsStrList[0], ' ' if len(pathAsStrList) <= 2 else str(*pathAsStrList[1:-1]) ,
73             f"{fname_with_ext[0]}{datetime.now().strftime('%Y_%m_%d-%H.%M.%S')}{fname_with_ext[1]}" if
74             ts_type == ts_sfix else \
75             (f"{datetime.now().strftime('%Y_%m_%d-%H.%M.%S_')}{pathAsStrList[-1]}" if ts_type == ts_pfix else
76             pathAsStrList[-1]) )
77
```

```

1 import os
2
3 import pandas as pd
4 import numpy as np
5 from sklearn.model_selection import ShuffleSplit
6
7 import valeo.infrastructure.XY_metadata as XY_metadata
8 from valeo.infrastructure import Const
9 from valeo.infrastructure.LogManager import LogManager
10 from valeo.infrastructure.tools.DfUtil import DfUtil
11
12
13 class XY_Loader:
14     logger = None
15
16     def __init__(self):
17         XY_Loader.logger = LogManager.logger(__name__)
18
19     def get_cv(X, y):
20         cv = ShuffleSplit(n_splits=8, test_size=0.5, random_state=57)
21         return cv.split(X)
22
23
24     def load_XY_df(self, mt: XY_metadata, delete_XY_join_cols=True) -> ():
25         # X_df = pd.read_csv(mt.X_pathname, na_values='') # NaN
26         X_df = pd.read_csv(mt.X_pathname) # NaN
27         # print(X_df[Const.OP100_Capuchon_insertion_mesure].head(20))
28         # X_df[[Const.OP100_Capuchon_insertion_mesure]] = X_df[[Const.OP100_Capuchon_insertion_mesure]].fillna(0
29         .0)
30         # print(X_df[Const.OP100_Capuchon_insertion_mesure].head(20))
31
32         # 1 - Check whether Y is in separate file or in the same as X
33         if mt.is_XY_in_separate_file():
34             Y_df = pd.read_csv(mt.Y_pathname)
35             XY_df = pd.merge(left=X_df, right=Y_df, how='inner', left_on=mt.X_join, right_on=mt.Y_join, suffixes
36             =('', ''))
37         else:
38             Y_df = None
39             XY_df = X_df
40
41         # 2 - When not reading a Test dataset (it means there is a Target dataset) THEN Let X_df group only
42         # features and Y_df only target
43         if mt.is_training_set():
44             Y_df = XY_df[mt.target_col_name]
45             X_df = XY_df.drop(mt.target_col_name, axis=1)
46
47         # 3 - Check whether we should remove joining columns
48         if delete_XY_join_cols:
49             X_df = X_df.drop(mt.X_join, axis=1)
50             try:
51                 X_df = X_df.drop(mt.Y_join, axis=1)
52             except:
53                 pass
54
55         #
56         # XY_Loader.logger.debug(f'X_df.columns: {X_df.columns}')
57         # if Y_df is not None:
58         #     XY_Loader.logger.debug(f'type(Y_df):{type(Y_df)}\nY_df: {Y_df}')
59         return X_df, Y_df
60
61     def load_XY_values(self, mt: XY_metadata, delete_XY_join_cols=True) -> ():
62         X_df, Y_df = self.load_XY_df(mt, delete_XY_join_cols)
63         return X_df.values if X_df is not None else None, \
64             Y_df.values if Y_df is not None else None

```

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

```
1 # https://docs.python.org/3/Library/Logging.html#Logrecord-attributes + Useful Handlers
2 # https://docs.python-guide.org/writing/Logging/
3 # https://github.com/Delgan/Loguru
4 # https://kingspp.github.io/design/2017/11/06/the-head-and-tail-of-logging.html
5 # https://stackoverflow.com/questions/4690600/python-exception-message-capturing
6 import logging.config
7 import os
8
9 from valeo.infrastructure.tools.ConfigLoader import ConfigLoader
10 import valeo.infrastructure.Const as Const
11
12 class LogManager():
13
14     # NB: The ctor() initializes the logging configuration
15     def __init__(self):
16         self.log_config = LogLoader().load()
17
18     @classmethod
19     def logger(cls, logname):
20         # l = logging.getLogger(logname)
21         # l.
22         return logging.getLogger(logname)
23
24
25 class LogLoader(ConfigLoader):
26     """
27     Load the logging configuration file
28     """
29     def load(self) -> dict:
30         try :
31             dict = super().load(os.path.join(Const.rootResources(), Const.APP_DEFAULT_LOG_FILE), Const.
32 ENV_KEY_LOG_FILE_PATHNAME)
33             logging.config.dictConfig(dict)
34             return dict
35         except Exception as ex:
36             logging.basicConfig(level=logging.INFO)
37             logging.warning(f'Error while loading logging configuration file:\n' \
38 f'\t- APP_RESOURCE_PATH = {Const.rootResources()}\n' \
39 f'\t- APP_DEFAULT_LOG_FILE = {Const.APP_DEFAULT_LOG_FILE}\n' \
40 f'\t- ENV_KEY_LOG_FILE_PATHNAME = {Const.ENV_KEY_LOG_FILE_PATHNAME}')
41             logging.exception(ex)
42             return None
```

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

```
1 # explicitly require this experimental feature
2 from sklearn.experimental import enable_iterative_imputer
3 # now you can import normally from sklearn.impute
4 from sklearn.impute import IterativeImputer
5 from sklearn.linear_model import BayesianRidge
6
7 import pandas as pd
8 import numpy as np
9 from sklearn.preprocessing import RobustScaler
10
11 from valeo.infrastructure.LogManager import LogManager
12
13
14 class Transformer() :
15     logger = LogManager.logger(__name__)
16
17     # def __init__(self):
18     #     lm = LogManager()
19     #     self.logger = lm.Logger(__name__)
20
21     '''
22     A strategy for imputing missing values by modeling each feature with missing values as a function of other
23     features in a round-robin fashion.
24     Multivariate imputer that estimates each feature from all the others.
25     https://scikit-learn.org/stable/modules/impute.html#iterative-imputer
26
27     Arg:
28     ----
29     estimator : The estimator to use at each step of the round-robin imputation.
30
31     Returns:
32     -----
33     A transformed DataFrame containing all the missing values.
34     NB: The argument DataFrame is NOT modified => It stills intact
35     https://towardsdatascience.com/introduction-to-bayesian-linear-regression-e66e60791ea7
36     '''
37     def iterative_imputer_transform(self, df_to_transform : pd.DataFrame, estimator=BayesianRidge(),
38     missing_values=np.nan, max_iter=10, initial_strategy = 'median') -> pd.DataFrame :
39         cols = df_to_transform.columns
40         imputer = IterativeImputer(estimator=estimator, missing_values=missing_values, max_iter=max_iter,
41         initial_strategy=initial_strategy, add_indicator=False) # It models each feature with missing values as a
42         function of other features, and uses that estimate for imputation
43         df_transformed = pd.DataFrame(imputer.fit_transform(df_to_transform))
44         # df_transformed.columns = df_transformed.columns[:-1]
45         df_transformed.columns = cols
46         return df_transformed
47
48     def robust_scaler_transform(self, df_to_transform : pd.DataFrame, with_centering=True, with_scaling=True,
49     quantile_range=(5.0, 95.0)):
50         cols = df_to_transform.columns
51         scaler = RobustScaler(with_centering=with_centering, with_scaling=with_scaling, quantile_range=
52         quantile_range)
53         df_transformed = pd.DataFrame(scaler.fit_transform(df_to_transform))
54         df_transformed.columns = cols
55         return df_transformed
```

```
1 import os
2
3
4 class XY_metadata :
5
6     def __init__(self, X_pathname :[], Y_pathname :[], X_join:[], Y_join:[], target_col_name:str):
7         self.X_pathname = os.path.join(X_pathname[0], *X_pathname[1:])
8         self.Y_pathname = None if Y_pathname is None else os.path.join(Y_pathname[0], *Y_pathname[1:])
9         self.X_join = X_join
10        self.Y_join = Y_join
11        self.target_col_name = target_col_name
12
13    def is_training_set(self) -> bool :
14        return True if self.target_col_name is not None else False
15
16    def is_XY_in_separate_file(self) -> bool:
17        return True if self.Y_pathname is not None else False
```

File - C:\EXED\Training__VALEO\src\valeo\infrastructure\SimpleImputer.py

```
1 from sklearn.impute import SimpleImputer as _SimpleImputer
2
3 from valeo.infrastructure.tools.DfInDfOut import DfInDfOut
4
5
6 class SimpleImputer(_SimpleImputer, DfInDfOut):
7
8     def transform(self, X):
9         Xt = super().transform(X)
10        return super().check_output(Xt, ensure_index=X, ensure_columns=X)
```

File - C:\EXED\Training__VALEO\src\valeo\infrastructure\StandardScaler.py

```
1 from sklearn.preprocessing import StandardScaler as _StandardScaler
2
3 from valeo.infrastructure.tools.DfInDfOut import DfInDfOut
4
5
6 class StandardScaler(_StandardScaler, DfInDfOut):
7
8     def transform(self, X):
9         Xt = super().transform(X)
10        return super().check_output(Xt, ensure_index=X, ensure_columns=X)
```

File - C:\EXED\Training__VALEO\src\valeo\infrastructure\AppConfigManager.py

```
1 import valeo.infrastructure.Const as const
2 from valeo.infrastructure.tools.ConfigLoader import ConfigLoader
3
4 class AppConfigManager():
5
6     def __init__(self):
7         cl = AppConfigLoader()
8         self.app_config = cl.load()
9
10    def getValue(self, nested_dict:{}, keys:[]) -> str :
11        return nested_dict[keys[0]] if len(keys) == 1 else self.getValue(nested_dict[keys[0]] , keys[1:])
12
13
14
15 class AppConfigLoader(ConfigLoader) :
16
17     def load(self) -> dict:
18         return super().load(f'{const.rootResources}{const.APP_DEFAULT_CONFIG_FILE}', const.
ENV_KEY_CONFIG_FILE_PATHNAME)
```



```

1 from datetime import datetime
2
3 from pandas import Series
4 from sklearn.base import BaseEstimator
5
6 from valeo.infrastructure.LogManager import LogManager
7 from valeo.infrastructure import Const as C
8
9 import os
10 import pandas as pd
11 import numpy as np
12
13 class DfUtil() :
14     logger = LogManager.logger(__name__)
15
16
17     # https://stackabuse.com/pythons-classmethod-and-staticmethod-explained/
18     @classmethod
19     def read_csv(cls, pathAsStrList : []) -> pd.DataFrame:
20         try:
21             return pd.read_csv(os.path.join(pathAsStrList[0], *pathAsStrList[1:]) )
22         except Exception as ex :
23             cls.logger.exception("Error while load data from %s", "/" .join(pathAsStrList))
24
25     @classmethod
26     def write_y_csv(cls, X_id:Series, y_target: np.ndarray, y_col_name:str, pathAsStrList : [], ts_type=C.
27         ts_sfix):
28         DfUtil.write_df_csv( pd.DataFrame(data={X_id.name:X_id, y_col_name:y_target}), pathAsStrList, ts_type=
29         ts_type)
30
31     @classmethod
32     def write_df_csv(cls, df:pd.DataFrame, pathAsStrList : [], ts_type=C.ts_sfix):
33         try :
34             df.to_csv( C.ts_pathname(pathAsStrList,ts_type), index = False)
35         except Exception as ex:
36             cls.logger = LogManager.logger("DfUtil")
37             cls.logger.exception(f"Error while writing 'df' to CSV '{pathAsStrList}'")
38
39     @classmethod
40     def df_imputer(cls, dfToImpute:pd.DataFrame, imputer:BaseEstimator):
41         '''This method encodes non-null data and replace it in the original data'''
42         # Retains only non-null values. dropna: Remove [rows(default) OR columns] when missing values
43         nonulls = np.array(dfToImpute.dropna())
44         # Reshapes the data for encoding
45         impute_reshape = nonulls.reshape(-1,1)
46         # #encode date
47         # impute_ordinal = imputer.fit_transform(impute_reshape)
48         # Assign back encoded values to non-null values
49         dfToImpute.loc[dfToImpute.notnull()] = np.squeeze(imputer.fit_transform(impute_reshape)) # np.squeeze:
50         Remove single-dimensional entries from the shape of an array.
51         return dfToImpute
52
53     @classmethod
54     def outlier_ratio(cls, df:pd.DataFrame) -> float:
55         Q1 = df.quantile(0.25)
56         Q3 = df.quantile(0.75)
57         IQR = Q3 - Q1
58         #
59         outliers = ((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)
60         return len(df[outliers].index)/len(df.index)

```

```

1 from valeo.infrastructure import Const as C
2 from valeo.infrastructure.LogManager import LogManager
3 import os
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 import pandas as pd
7
8 class ImgUtil():
9     logger = LogManager.logger(__name__)
10
11     # https://stackoverflow.com/questions/17907813/python-classmethod-and-staticmethod-explained/
12     @classmethod
13     def save_fig(cls, fig_id:str , tight_layout=True, fig_extension="png", resolution=300, ts_type=C.ts_sfix):
14         path = C.ts_pathname([C.rootImages() , fig_id + "." + fig_extension], ts_type=ts_type)
15         # cls.logger.debug(f"Saving figure '{fig_id}'")
16         if tight_layout:
17             plt.tight_layout()
18         # Save "the current figure plot" that is set by "df.hist(...)". @ReferTo: pyplot.py / def gcf()
19         plt.savefig(path, format=fig_extension, dpi=resolution)
20
21     @classmethod
22     def save_df_hist_plot(cls, df:pd.DataFrame, fig_id:str , bins=50, figsize=(20,15), tight_layout=True,
23                         fig_extension="png", resolution=300, ts_type=C.ts_sfix):
24         cls.logger.debug(f"Generating 'hist' plot: bins={bins} - figsize={figsize}")
25         df.hist(bins=bins, figsize=figsize)
26         cls.save_fig(fig_id=f"{fig_id}_histogram_{figsize[0]}x{figsize[1]}", tight_layout=tight_layout,
27                     fig_extension=fig_extension, resolution=resolution, ts_type=ts_type)
28
29     @classmethod
30     def save_df_XY_hist_plot(cls, df_XY:pd.DataFrame, fig_id:str, bins=50, figsize=(5, 5), y_target_name=None,
31                             tight_layout=True,
32                             fig_extension="png", resolution=300, ts_type=C.ts_sfix):
33         cls.logger.debug(f"Generating 'XY_hist' plot: bins={bins} - figsize={figsize}")
34         df_X = df_XY.drop(columns=y_target_name, axis=1)
35         y = df_XY[y_target_name]
36         fig, ax = plt.subplots(figsize=figsize)
37         for i, col in enumerate(sorted(df_X.columns)) :
38             for clazz in y.unique() :
39                 df_X[y==clazz][col].plot.hist(bins=bins, figsize=figsize, alpha=0.3, label=f'Class #{int(clazz)}')
40
41         plt.legend()
42         plt.xlabel(col)
43         ImgUtil.save_fig(fig_id=f"{fig_id}_{col}_histogram_{figsize[0]}x{figsize[1]}", tight_layout=
44         tight_layout, fig_extension=fig_extension, resolution=resolution, ts_type=ts_type)
45         ax.clear()
46
47     # NB:
48     # df.hist: => Plot 1 Histo per dfColumn
49     # df.plot.hist: => Plot all df-referenced-Columns on same Histo
50
51     @classmethod
52     def save_df_scatter_matrix_plot(cls, df:pd.DataFrame, fig_id:str , figsize=(20,15), cfield=None, tight_layout
53     =True,
54                                     fig_extension="png", resolution=300, ts_type=C.ts_sfix):
55         cls.logger.debug(f"Generating 'scatter matrix' plot: figsize={figsize}")
56         if cfield == None :
57             pd.plotting.scatter_matrix(df, figsize=figsize)
58         else :
59             pd.plotting.scatter_matrix(df, figsize=figsize, alpha=0.3, c=df[cfield].values, cmap='RdBu')
60         cls.save_fig(fig_id=f"{fig_id}_scatter_matrix_{figsize[0]}x{figsize[1]}", tight_layout=tight_layout,
61                     fig_extension=fig_extension, resolution=resolution, ts_type=ts_type)
62
63     @classmethod
64     def save_df_heatmap_plot(cls, df:pd.DataFrame, fig_id:str , figsize=(20,20), cmap='RdBu', annot=True ,
65                             annot_kws={'size':15}, fig_extension="png", resolution=300, ts_type=C.ts_sfix):
66         cls.logger.debug(f"Generating 'heatmap' plot: figsize={figsize}")
67         fig, ax = plt.subplots(figsize=figsize)
68         sns.set(font_scale=1.1)
69         sns.heatmap(df, cmap=cmap, annot=annot , annot_kws=annot_kws, ax=ax)
70         ax.set_title(fig_id, fontsize=28)
71         cls.save_fig(fig_id=f"{fig_id.replace(' ','_')}heatmap_{figsize[0]}x{figsize[1]}", tight_layout=True,
72                     fig_extension=fig_extension, resolution=resolution, ts_type=ts_type)
73
74     @classmethod
75     def save_df_violin_plot(cls, df:pd.DataFrame, fig_id:str, grid_elmt_x:int, figsize=(20,20), fig_extension="
76     png", resolution=300, ts_type=C.ts_sfix):
77         cls.logger.debug(f"Generating 'violin' plot: figsize={figsize}")
78         grid_elmt_y = len(df.columns) // grid_elmt_x if (len(df.columns) % grid_elmt_x) == 0 else (len(df.
79         columns) // grid_elmt_x) + 1
80
81         #
82         fig, axs = plt.subplots(grid_elmt_y, grid_elmt_x, figsize=figsize)
83         for i, col in enumerate(sorted(df.columns)) :
84             sns.violinplot(x=df[col], linewidth=1, ax=axs[i//grid_elmt_x, i%grid_elmt_x])
85             # sns.stripplot( x=df[col], color="orange", jitter=0.2, linewidth=1, ax=axs[i//3,i%3])
86             sns.boxplot( x=df[col], linewidth=1, ax=axs[i//grid_elmt_x, i%grid_elmt_x], saturation=0 )
87             # axs.set_title(fig_id, fontsize=28)
88         cls.save_fig(fig_id=f"{fig_id.replace(' ','_')}violin_{figsize[0]}x{figsize[1]}", tight_layout=True,
89                     fig_extension=fig_extension, resolution=resolution, ts_type=ts_type)
90
91

```

```

78
79 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):
80     df = df_XY.drop(columns=y_target_name, axis=1)
81     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
82     #
83     fig, axs = plt.subplots(grid_elmt_y, grid_elmt_x, figsize=figsize)
84     for i, col in enumerate(sorted(df.columns)) :
85         sns.violinplot(x=y_target_name, y=col, data=df_XY, linewidth=1, ax=axs[i//grid_elmt_x, i%grid_elmt_x])
86         sns.boxplot (x=y_target_name, y=col, data=df_XY, linewidth=1, ax=axs[i//grid_elmt_x, i%grid_elmt_x] )
87         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)
88
89
90 @classmethod
91 # SWARM PLOT did not work correctly
92 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):
93     cls.logger.debug(f"Generating 'swarm' plot: figsize:{figsize}")
94     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
95     #
96     fig, axs = plt.subplots(grid_elmt_y, grid_elmt_x, figsize=figsize)
97     for i, col in enumerate(sorted(df.columns)) :
98         sns.swarmplot(x=df[col], linewidth=1, ax=axs[i//grid_elmt_x, i%grid_elmt_x], hue=df[cfield].values)
99     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)
100
101 # 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):
102     # df_X = df_XY.drop(columns=y_target_name, axis=1)
103     # y = df_XY[y_target_name]
104     # fig, ax = plt.subplots(figsize=figsize)
105     # for i, col in enumerate(sorted(df_X.columns)) :
106     #     for clazz in y.unique() :
107     #         sns.swarmplot(x=col, hue=y_target_name, data=df_XY[y==clazz])
108     #         # sns.swarmplot(x='data', y='feature', hue='Label', data=df)
109     #         plt.legend()
110     #         plt.xlabel(col)
111     #         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)
112     #         ax.clear()
113
114
115 # NB: Generic way to plot whatever :
116 # fig, ax = plt.subplots(figsize=(20,20))
117 # sns.heatmap(corr_matrix, cmap='RdBu', annot=True, annot_kws={'size':15}, ax=ax)
118 # ax.set_title("Valeo starter production correlation measures", fontsize=14)
119 # plt.show()
120
121

```

```
1
2 import pandas as pd
3
4 class DfInDfOut:
5
6     # https://github.com/scikit-learn/scikit-learn/issues/5523 : Pandas in, Pandas out
7     def check_output(self, X, ensure_index=None, ensure_columns=None):
8         """
9         Joins X with ensure_index's index or ensure_columns's columns when available
10        """
11        if ensure_index is not None:
12            if ensure_columns is not None:
13                if type(ensure_index) is pd.DataFrame and type(ensure_columns) is pd.DataFrame:
14                    X = pd.DataFrame(X, index=ensure_index.index, columns=ensure_columns.columns)
15            else:
16                if type(ensure_index) is pd.DataFrame:
17                    X = pd.DataFrame(X, index=ensure_index.index)
18        return X
```

```
1
2 import os
3 import yaml
4 import logging
5
6 # from infrastructure.LogManager import LogManager
7
8 class YamlLoader :
9     """
10     Load a yaml configuration file
11     """
12     logger = None
13
14     def __init__(self):
15         if YamlLoader.logger is None :
16             YamlLoader.logger = logging.getLogger(__name__)
17             logging.basicConfig(level=logging.INFO)
18
19     def load(self, file_pathname:str) -> dict :
20         if os.path.exists(file_pathname):
21             with open(file_pathname, 'rt') as f:
22                 try:
23                     dict = yaml.safe_load(f.read())
24                     # YamlLoader.logger.info(f'Loading file "{file_pathname}":\n\t{dict}')
25                     YamlLoader.logger.info(f'Loading file "{file_pathname}":\n{dict}')
26                     return dict
27                 except Exception as ex:
28                     YamlLoader.logger.exception(f'Error while loading file "{file_pathname}"')
29         else:
30             YamlLoader.logger.error(f'Error while loading file "{file_pathname}"')
31
32     return None
33
```

```
1
2 import os
3 import logging
4
5 from valeo.infrastructure.tools.YamlLoader import YamlLoader
6
7 class ConfigLoader(YamlLoader) :
8     logger = None
9     """
10     Load an external or a package embedded configuration file.
11     Check first if the environment variable {APP_CONFIG_PATHNAME}
12     """
13
14     def __init__(self):
15         super().__init__()
16         ConfigLoader.logger = logging.getLogger(__name__)
17
18     def load(self, file_pathname:str, env_key_as_config_pathname:str) -> dict :
19         try :
20             path_as_key = os.getenv(env_key_as_config_pathname, None)
21             return super().load(path_as_key if path_as_key else file_pathname )
22         except Exception as ex :
23             ConfigLoader.logger.exception(f'Error while loading file "{file_pathname}"')
24             raise ex
25         # self.logger.error(ex, exc_info=True)
26     return None
27
```

```
1
2 import os
3 from datetime import datetime
4
5 from sklearn.base import BaseEstimator, TransformerMixin
6 import numpy as np
7
8 from valeo.infrastructure import Const as C
9
10
11 class DebugPipeline(BaseEstimator, TransformerMixin):
12     OFFSET = 10
13     counter = -OFFSET
14
15     def __init__(self):
16         DebugPipeline.counter = ( (DebugPipeline.counter + DebugPipeline.OFFSET) // DebugPipeline.OFFSET) *
17         DebugPipeline.OFFSET
18
19     def transform(self, X, y=None):
20         # %f : print micro seconds
21         # np.savetxt(os.path.join(C.rootProject(), 'log', 'dbgPipeline_' + datetime.now().strftime("%Y_%m_%d-%H
22         .%M.%S_") + str(DebugPipeline.counter)) + '.txt', X, delimiter=',')
23         DebugPipeline.counter += 1
24
25     def fit(self, X, y=None, **fit_params):
26         if y is not None :
27             # np.savetxt(os.path.join(C.rootProject(), 'log', 'dbgPipeline_' + datetime.now().strftime("%Y_%m_%d
28             -%H.%M.%S_") + str(DebugPipeline.counter)) + '_Y.txt', y, delimiter=',')
29             # np.savetxt(os.path.join(C.rootProject(), 'log', 'dbgPipeline_' + datetime.now().strftime("%Y_%m_%d
30             -%H.%M.%S_") + str(DebugPipeline.counter)) + '_X.txt', X, delimiter=',')
31             DebugPipeline.counter += 1
32         return self
```