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 Entées/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 entres le dataset initiale et le dataset dépourvu des outliers
   * e. Distribution du jeux 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 bivariables 'feature/target' des données:
   * a. Matrice de correlation et heatmap après gestion des valeurs manquantes
   * b. Matrice de correlation 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 **A FAIRE**
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. 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 regénération du 7, du 8 et 9.**
2. Compléter avec 3 classifieurs de type différents: à base d'arbre / à base de distance / à base de reseau de Neuronne ou bien de Stacking: **Points 11, 12, 13 de l'index ci dessus**
   * Pour chaque classifieur faire: TrainTestSplit / CV / SearhGridCV
   * Pour chaque classifieur: Analyse et interpretation des résultats / Matrice de confusion / F1, Roc / Graphe F1 et ROC
   * **Si le temps le permet** alors faire pour chaque classifieur: Graphe Overfit Underfit / Graphe avec des valeurs differents des hyperparamètres.
3. Expliquer pourquoi une classification déséquilibrée pose un défi pour la modélisation prédictive.
4. Identifier la/les motivations pour avoir une distribution Normale "bell shape". Citer les avantages d'une telle distribution.
5. Mise en forme selon les indications du document "DSSP14\_Guidelines\_Projet\_Professionnel.doc"

### Questions auxquelles j’aimerais avoir une réponse:

1. Quand on a une distribution asymétrique pour une feature et qu'on voudrait lui appliquer une transformation logarithmique pour s'approcher d'une distribution normale:
   * Est ce qu'il vaut mieux appliquer la transformation sur les features asymétriques seulement ?

Ou bien on peut l'appliquer sur la totalité des features de la dataframe (ça nous évite de choisir une-par-une les features à transformer)

* + Cette transformation sera appliqué sur le TrainSet;

Est ce qu'il faut l'appliquer aussi sur le TestSet au moment de la prédiction ?

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

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

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

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

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

**Questions:**

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

1. Est ce qu'il faut commenter davantage les graphes et les mesures figurant dans les chapitres ‘Exploration des données’, chapitres 5,6 et 7 ?
2. 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 ?

1. Quelles sont les parties que je dois développer davantage ? Ou bien être plus concis ?
2. Est-ce qu’il faut garder dans le document, les petits bout de code Python qu’on trouve tout au long des chapitres (hors chapitre 16 – Annexe : code Python) ? Ou bien il faut les supprimer ?

***Merci*** *:-)*

## 1 - Contexte de l'étude

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

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

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

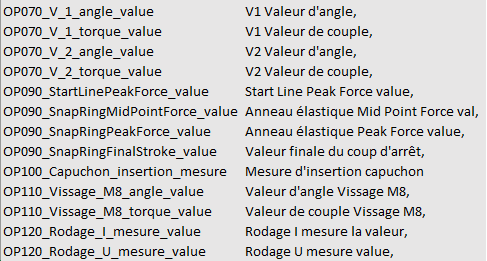
Les classifications déséquilibrées posent un défi pour la modélisation prédictive. La classe minoritaire est plus importante et donc le problème est plus sensible aux erreurs de classification pour la classe minoritaire que pour la classe majoritaire.

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

## 2 - Description du data set case

#### a - Entrées:

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

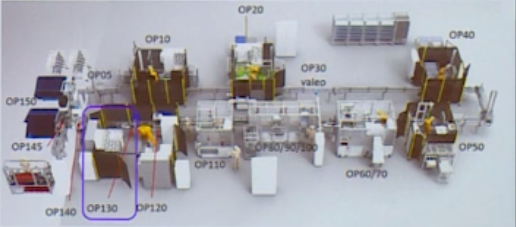
On distingue par exemple:

#### 

#### b - Sortie:

Il s'agit de la valeur de résultat de l'OP130, banc d'essai: Binar OP130\_Resultat\_Global\_v.

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



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

#### c - Croisement Entées/Sortie:

Les données de training sont réparties dans 2 fichiers csv:

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

Un identifiant technique 'PROC\_TRACEINFO' permet de croiser le fichier d'entrée au fichier de sortie.

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

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

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

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

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

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

import os  
import sys  
import logging  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from pandas.plotting import scatter\_matrix  
import seaborn as sns  
%matplotlib inline  
  
from sklearn.model\_selection import train\_test\_split  
from imblearn.over\_sampling import SMOTE  
  
  
from imblearn.ensemble import BalancedBaggingClassifier, RUSBoostClassifier, BalancedRandomForestClassifier  
from imblearn.over\_sampling import RandomOverSampler, ADASYN, SMOTE, SVMSMOTE, KMeansSMOTE, BorderlineSMOTE  
from imblearn.over\_sampling.base import BaseOverSampler  
from imblearn.pipeline import Pipeline  
  
from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier, RandomForestClassifier  
from sklearn.ensemble.\_hist\_gradient\_boosting.gradient\_boosting import HistGradientBoostingClassifier  
from sklearn.cluster import MiniBatchKMeans  
from sklearn.compose import ColumnTransformer  
from sklearn.linear\_model import LogisticRegression  
from sklearn.linear\_model.\_stochastic\_gradient import SGDClassifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.svm import SVC, NuSVC, LinearSVC  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.model\_selection import cross\_validate, StratifiedKFold  
  
import xgboost as xgb  
  
# Import "valeo" module  
sys.path.append("..")  
from valeo.infrastructure.LogManager import LogManager as lm  
# NB: Initializing logger here allows "class loaders of application classes" to benefit from the global initialization  
logger = lm().logger(\_\_name\_\_)   
from valeo.infrastructure import Const   
from valeo.infrastructure.tools.DfUtil import DfUtil  
from valeo.infrastructure.tools.ImgUtil import ImgUtil  
from valeo.infrastructure.XY\_Loader import XY\_Loader  
from valeo.infrastructure.XY\_metadata import XY\_metadata as XY\_metadata  
from valeo.domain.ValeoModeler import ValeoModeler  
from valeo.domain.ValeoPredictor import ValeoPredictor  
import valeo.infrastructure.Transformer as transf  
  
  
  
# Notebook automatic reload  
%load\_ext autoreload  
%reload\_ext autoreload  
%aimport valeo.infrastructure.Transformer   
%aimport valeo.infrastructure.LogManager  
%aimport valeo.infrastructure.Const  
%aimport valeo.infrastructure.tools.DfUtil  
%aimport valeo.infrastructure.tools.ImgUtil  
%aimport valeo.infrastructure.XY\_Loader  
%aimport valeo.infrastructure.XY\_metadata  
%aimport valeo.domain.ValeoModeler  
%aimport valeo.domain.ValeoPredictor

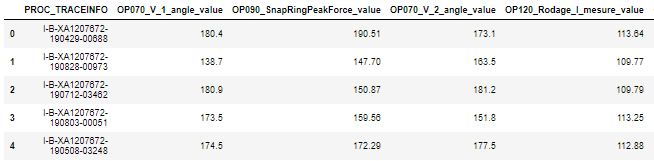
## 4 - Chargement des données 'Training'

data = DfUtil.read\_csv([Const.rootDataTrain() , "traininginputs.csv"])  
Y\_data = DfUtil.read\_csv([Const.rootDataTrain(), "trainingoutput.csv"])

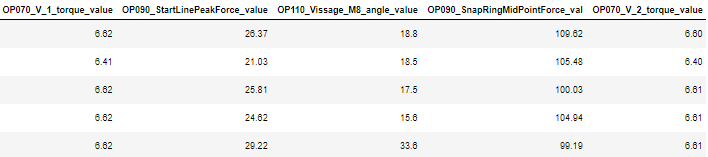
## 5 - Exploration et analyse tabulaire des données

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

data.head()







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

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

data.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 34515 entries, 0 to 34514  
Data columns (total 14 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 PROC\_TRACEINFO 34515 non-null object   
 1 OP070\_V\_1\_angle\_value 34515 non-null float64  
 2 OP090\_SnapRingPeakForce\_value 34515 non-null float64  
 3 OP070\_V\_2\_angle\_value 34515 non-null float64  
 4 OP120\_Rodage\_I\_mesure\_value 34515 non-null float64  
 5 OP090\_SnapRingFinalStroke\_value 34515 non-null float64  
 6 OP110\_Vissage\_M8\_torque\_value 34515 non-null float64  
 7 OP100\_Capuchon\_insertion\_mesure 15888 non-null float64  
 8 OP120\_Rodage\_U\_mesure\_value 34515 non-null float64  
 9 OP070\_V\_1\_torque\_value 34515 non-null float64  
 10 OP090\_StartLinePeakForce\_value 34515 non-null float64  
 11 OP110\_Vissage\_M8\_angle\_value 34515 non-null float64  
 12 OP090\_SnapRingMidPointForce\_val 34515 non-null float64  
 13 OP070\_V\_2\_torque\_value 34515 non-null float64  
dtypes: float64(13), object(1)  
memory usage: 3.7+ MB

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

On constate que:

* Toutes les features sont numériques et continues, pas de features catégoriques
* Plus de la moitié des valeurs de la feature 7 ' OP100\_Capuchon\_insertion\_mesure' sont manquants

=> Cette feature doit être traitée en lui imputant des valeurs. Un imputer de type IterativeImputer(stratégie 'médiane') ser a utilisé.

* PROC\_TRACEINFO de type object (=> String), c'est l'identifiant de ligne permettant de croiser les 'features' avec la 'target'. Cette feature porte l'horodatage de l'assemblage des démarreurs, la date sous jacente sera extraite et utilisée dans la phase de 'features engineering'

#### c - Données manquantes par type de 'feature':

data.isna().sum()

PROC\_TRACEINFO 0  
OP070\_V\_1\_angle\_value 0  
OP090\_SnapRingPeakForce\_value 0  
OP070\_V\_2\_angle\_value 0  
OP120\_Rodage\_I\_mesure\_value 0  
OP090\_SnapRingFinalStroke\_value 0  
OP110\_Vissage\_M8\_torque\_value 0  
OP100\_Capuchon\_insertion\_mesure 18627  
OP120\_Rodage\_U\_mesure\_value 0  
OP070\_V\_1\_torque\_value 0  
OP090\_StartLinePeakForce\_value 0  
OP110\_Vissage\_M8\_angle\_value 0  
OP090\_SnapRingMidPointForce\_val 0  
OP070\_V\_2\_torque\_value 0  
dtype: int64

L'identifiant 'PROC\_TRACEINFO' est supprimé **provisoirement** de l'ensemble des features:

X\_data = data.drop(columns = "PROC\_TRACEINFO")  
X\_data.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 34515 entries, 0 to 34514  
Data columns (total 13 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 OP070\_V\_1\_angle\_value 34515 non-null float64  
 1 OP090\_SnapRingPeakForce\_value 34515 non-null float64  
 2 OP070\_V\_2\_angle\_value 34515 non-null float64  
 3 OP120\_Rodage\_I\_mesure\_value 34515 non-null float64  
 4 OP090\_SnapRingFinalStroke\_value 34515 non-null float64  
 5 OP110\_Vissage\_M8\_torque\_value 34515 non-null float64  
 6 OP100\_Capuchon\_insertion\_mesure 15888 non-null float64  
 7 OP120\_Rodage\_U\_mesure\_value 34515 non-null float64  
 8 OP070\_V\_1\_torque\_value 34515 non-null float64  
 9 OP090\_StartLinePeakForce\_value 34515 non-null float64  
 10 OP110\_Vissage\_M8\_angle\_value 34515 non-null float64  
 11 OP090\_SnapRingMidPointForce\_val 34515 non-null float64  
 12 OP070\_V\_2\_torque\_value 34515 non-null float64  
dtypes: float64(13)  
memory usage: 3.4 MB

#### d - Statistique descriptive

X\_data.sort\_index(axis=1).describe().transpose()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | count | mean | std | min | 25% | 50% | 75% | max |
| OP070\_V\_1\_angle\_value | 34515.0 | 159.906922 | 15.662650 | 101.80 | 148.70 | 158.00 | 169.30 | 198.30 |
| OP070\_V\_1\_torque\_value | 34515.0 | 6.548403 | 0.097602 | 5.67 | 6.41 | 6.61 | 6.62 | 6.67 |
| OP070\_V\_2\_angle\_value | 34515.0 | 159.618236 | 15.091490 | 82.00 | 149.40 | 158.70 | 168.90 | 198.10 |
| OP070\_V\_2\_torque\_value | 34515.0 | 6.550867 | 0.094814 | 5.74 | 6.42 | 6.61 | 6.61 | 6.67 |
| OP090\_SnapRingFinalStroke\_value | 34515.0 | 11.970190 | 0.169873 | 0.00 | 11.85 | 12.04 | 12.08 | 12.19 |
| OP090\_SnapRingMidPointForce\_val | 34515.0 | 97.700978 | 6.837714 | 0.00 | 94.31 | 98.50 | 102.23 | 127.30 |
| OP090\_SnapRingPeakForce\_value | 34515.0 | 156.915055 | 11.271492 | 0.00 | 149.21 | 156.18 | 164.38 | 196.92 |
| OP090\_StartLinePeakForce\_value | 34515.0 | 23.630152 | 2.546341 | 0.00 | 22.28 | 23.88 | 25.29 | 43.41 |
| OP100\_Capuchon\_insertion\_mesure | 15888.0 | 0.388173 | 0.024425 | 0.24 | 0.38 | 0.39 | 0.41 | 0.42 |
| OP110\_Vissage\_M8\_angle\_value | 34515.0 | 17.878398 | 6.785079 | 6.30 | 13.50 | 16.40 | 20.20 | 84.60 |
| OP110\_Vissage\_M8\_torque\_value | 34515.0 | 12.256785 | 0.065319 | 12.03 | 12.21 | 12.26 | 12.30 | 12.50 |
| OP120\_Rodage\_I\_mesure\_value | 34515.0 | 113.350222 | 3.528522 | 99.99 | 111.04 | 113.16 | 115.38 | 177.95 |
| OP120\_Rodage\_U\_mesure\_value | 34515.0 | 11.971027 | 0.003050 | 11.97 | 11.97 | 11.97 | 11.97 | 11.99 |

On constate que :

* OP070\_V\_2\_angle\_value : Outlier côté Min => Utiliser un 'robust scaler' pour réduire l'effet Outlier
* OP090\_StartLinePeakForce\_value, OP090\_SnapRingMidPointForce\_val, OP090\_SnapRingPeakForce\_value, OP090\_SnapRingFinalStroke\_value :

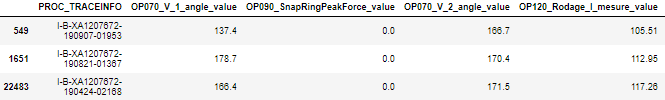
Identification de valeurs nulle, 'min' égal à 0. Normalement ces mesures physiques ne doivent pas être nulle, le fait qu'elles soient nulles laisse penser qu'elles sont nulles à tort et par conséquent il faut les considérer comme des valeurs manquantes et seront traitées dans la phase **'Feature Engineering'**

* OP090\_StartLinePeakForce\_value : Outlier côté Max => Utiliser un 'robust scaler'
* OP110\_Vissage\_M8\_angle\_value : Outlier côté Max => Utiliser un 'robust scaler'
* OP110\_Vissage\_M8\_torque\_value : Presque Constant
* OP100\_Capuchon\_insertion\_mesure: Plus de la moitié sans valeurs => Utiliser 'missing values' Imputer
* OP120\_Rodage\_U\_mesure\_value : Très petite variance
* OP120\_Rodage\_I\_mesure\_value : Outlier cote Max (=> Utiliser un 'robust scaler') + petite variance

###### 

###### *Nombre de features égales à Nulle:*

data.query('OP090\_StartLinePeakForce\_value == 0 or OP090\_SnapRingMidPointForce\_val == 0 or   
 OP090\_SnapRingPeakForce\_value == 0 or OP090\_SnapRingFinalStroke\_value == 0')



Seulement 3 observations dont les valeurs des features sont égales à 0. A cela s'ajoute la moitié des valeurs de 'OP100\_Capuchon\_insertion\_mesure' qui sont manquantes.

###### *Ratio d'observations ayant des features en outlier:*

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

Le ratio d'outlier est de 0.24256120527306968

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

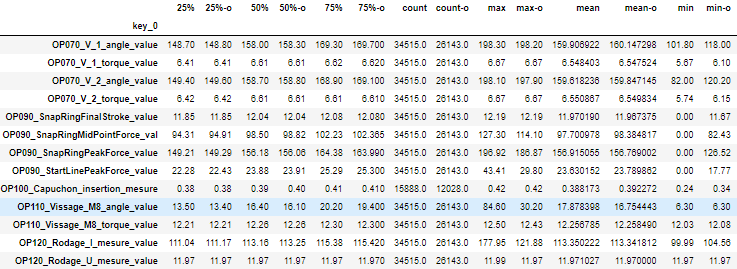
* Est ce que c'est une erreur
* Ou bien c'est une vrai donnée dont le pattern est différent

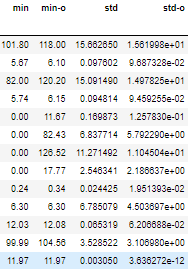
Pour limiter l effet des outliers:

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

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

# 1 - Le dataset dépourvu des outliers  
X\_data\_out = X\_data[~outliers]  
  
# 2 - Creéer les 2 dataframes des valeurs statistique descriptive  
Xt = X\_data.sort\_index(axis=1).describe().transpose()  
Xt\_out = X\_data\_out.sort\_index(axis=1).describe().transpose()  
  
# 3 - Fusionner les afin de pouvoir les comparer  
xt\_merged = pd.merge(left=Xt, right=Xt\_out, how='inner', left\_on=Xt.index, right\_on=Xt\_out.index, suffixes=('','-o'))  
xt\_merged = xt\_merged.set\_index(['key\_0'])  
xt\_merged.sort\_index(axis=1)





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

* OP090\_StartLinePeakForce\_value, OP110\_Vissage\_M8\_angle\_value, OP120\_Rodage\_I\_mesure\_value: Le 'max' a chuté considérablement
* OP110\_Vissage\_M8\_angle\_value : Variance considérablement plus petite

#### e - Distribution du jeux des données:

starter\_count = len(Y\_data[Const.Binar\_OP130\_Resultat\_Global\_v])  
starter\_count\_ok = Y\_data[Const.Binar\_OP130\_Resultat\_Global\_v].value\_counts()[0]  
starter\_count\_ko = Y\_data[Const.Binar\_OP130\_Resultat\_Global\_v].value\_counts()[1]  
#   
print(f'Nombre total des démarreurs : {starter\_count}')  
print(f'Nombre total des démarreurs OK => Nombre de Classes Negatives : {starter\_count\_ok} soit {round(starter\_count\_ok/starter\_count \* 100,2)} % du dataset')  
print(f'Nombre total des démarreurs KO => Nombre de Classes Positives : {starter\_count\_ko} soit {round(starter\_count\_ko/starter\_count \* 100,2)} % du dataset')

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

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

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

## 6 - Exploration graphique univariable des données

#### a - Histogramme des features après gestion des valeurs manquantes:

tsf = transf.Transformer()  
X\_data\_transformed = tsf.iterative\_imputer\_transform(X\_data)  
ImgUtil.save\_df\_hist\_plot(X\_data\_transformed,"X\_data\_imputed",figsize=(20,15), bins=100)  
plt.show()

![](data:image/png;base64;base64,)

###### NB:

tsf.iterative\_imputer\_transform(X\_data) méthode de la classe 'Transformer' du module Python valeo.infrastructure.Transformer. Elle applique un imputer du type IterativeImputer(estimator=BayesianRidge, missing\_values, initial\_strategy = 'median')

En observant les graphes des différents features, on constate:

* La plupart des distributions sont asymétriques notamment pour: (=> Appliquer une transformation logarithmique dans l'étape F.Engineering)
  + OP070\_V\_1\_angle\_value
  + OP090\_SnapRingMidPointForce\_val
  + OP090\_SnapLinePeakForce\_value
  + OP100\_Capuchon\_insertion\_mesure # feature dont la moitié des mesures n'existait pas
  + OP110\_Vissage\_M8\_angle\_value
* Une valeur de plafonnement (capping value) pour:
  + OP070\_V\_1\_angle\_value
  + OP070\_V\_2\_angle\_value
* Les distributions suivantes représentes 2 catégories d'observations indépendantes: (=> Représenter chaque feature par 2 catégories dans l'étape F.Engin.):
  + OP070\_V\_1\_torque\_value
  + OP070\_V\_2\_torque\_value
  + OP120\_Rodage\_U\_mesure\_value

###### *Histogramme comparant la feature 'OP100\_Capuchon\_insertion\_mesure' avant et après la gestions des valeurs manquantes:*

dff\_ = pd.DataFrame(X\_data[Const.OP100\_Capuchon\_insertion\_mesure])  
dff\_[Const.OP100\_Capuchon\_insertion\_mesure + "\_imputed"] = X\_data\_transformed[Const.OP100\_Capuchon\_insertion\_mesure]  
ImgUtil.save\_df\_hist\_plot(dff\_,Const.OP100\_Capuchon\_insertion\_mesure, figsize=(20,10))  
plt.show()

![](data:image/png;base64;base64,)

#### 

#### 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()

![](data:image/png;base64;base64,)

###### 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()

![](data:image/png;base64;base64,)

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

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

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

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

ImgUtil.save\_df\_violin\_plot(X\_data\_transformed, 'X\_data\_distribution', 3)

![](data:image/png;base64;base64,)

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)

![](data:image/png;base64;base64,)

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

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

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

# 1 - Charger les features et la target en les croisant:  
XY\_data\_with\_id = pd.merge(left=data, right=Y\_data, how='inner', left\_on=Const.PROC\_TRACEINFO, right\_on=Const.PROC\_TRACEINFO)  
XY\_data\_with\_id.info()

<class 'pandas.core.frame.DataFrame'>  
Int64Index: 34515 entries, 0 to 34514  
Data columns (total 15 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 PROC\_TRACEINFO 34515 non-null object   
 1 OP070\_V\_1\_angle\_value 34515 non-null float64  
 2 OP090\_SnapRingPeakForce\_value 34515 non-null float64  
 3 OP070\_V\_2\_angle\_value 34515 non-null float64  
 4 OP120\_Rodage\_I\_mesure\_value 34515 non-null float64  
 5 OP090\_SnapRingFinalStroke\_value 34515 non-null float64  
 6 OP110\_Vissage\_M8\_torque\_value 34515 non-null float64  
 7 OP100\_Capuchon\_insertion\_mesure 15888 non-null float64  
 8 OP120\_Rodage\_U\_mesure\_value 34515 non-null float64  
 9 OP070\_V\_1\_torque\_value 34515 non-null float64  
 10 OP090\_StartLinePeakForce\_value 34515 non-null float64  
 11 OP110\_Vissage\_M8\_angle\_value 34515 non-null float64  
 12 OP090\_SnapRingMidPointForce\_val 34515 non-null float64  
 13 OP070\_V\_2\_torque\_value 34515 non-null float64  
 14 Binar OP130\_Resultat\_Global\_v 34515 non-null int64   
dtypes: float64(13), int64(1), object(1)  
memory usage: 4.2+ MB

# 2 - Rajout des missing values afin d'avoir une meilleure représentation   
XY\_data = XY\_data\_with\_id.drop(columns = Const.PROC\_TRACEINFO)  
XY\_data\_transformed = tsf.iterative\_imputer\_transform(XY\_data)

# 3 - Correlation entre la target "Binar OP130\_Resultat\_Global\_v" et les autres attributs  
corr\_matrix = XY\_data\_transformed.corr()  
corr\_matrix[Const.Binar\_OP130\_Resultat\_Global\_v].sort\_values(ascending=False)

Binar OP130\_Resultat\_Global\_v 1.000000  
OP100\_Capuchon\_insertion\_mesure 0.040366  
OP090\_SnapRingFinalStroke\_value 0.015148  
OP090\_SnapRingMidPointForce\_val 0.014273  
OP090\_StartLinePeakForce\_value 0.010720  
OP110\_Vissage\_M8\_angle\_value 0.005470  
OP120\_Rodage\_I\_mesure\_value 0.003763  
OP110\_Vissage\_M8\_torque\_value -0.002984  
OP070\_V\_2\_angle\_value -0.006342  
OP090\_SnapRingPeakForce\_value -0.007290  
OP120\_Rodage\_U\_mesure\_value -0.010492  
OP070\_V\_1\_angle\_value -0.012793  
OP070\_V\_1\_torque\_value -0.037438  
OP070\_V\_2\_torque\_value -0.039752  
Name: Binar OP130\_Resultat\_Global\_v, dtype: float64

Le coefficient de correlation varie entre -1 et 1 :

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

**Le coefficient de correlation mesure uniquement les correlations linéaires**

# 4 - Dessiner la Heatmap  
title = 'Correlation matrix - features [{0}]'  
ImgUtil.save\_df\_heatmap\_plot(corr\_matrix,title.format('imputed'))

![](data:image/png;base64;base64,)

En observant la mattrice de correlation, on constate:

* L'inexistence d'aucune correlation forte entre la target 'Binar OP130\_Resultat\_Global\_v' et n'importe quel feature.
* L'exitence de correlations positives (0.54, 0.49, 0.48, .. ) et negatives (-0.68, -0.45, -0.38, ... ) parmi les autres features

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

# 1 - Appliquer la transformation 'Robust Scaler'  
XY\_data\_transformed\_scaled = tsf.robust\_scaler\_transform(XY\_data\_transformed.drop(columns=Const.Binar\_OP130\_Resultat\_Global\_v, axis=1))  
  
# 2 - Rajouter la target à la dataframe  
XY\_data\_transformed\_scaled[Const.Binar\_OP130\_Resultat\_Global\_v] = XY\_data\_transformed[Const.Binar\_OP130\_Resultat\_Global\_v]  
  
# 3 - Correlation entre la target "Binar OP130\_Resultat\_Global\_v" et les autres attributs  
corr\_matrix\_scaled = XY\_data\_transformed\_scaled.corr()  
corr\_matrix\_scaled[Const.Binar\_OP130\_Resultat\_Global\_v].sort\_values(ascending=False)

Binar OP130\_Resultat\_Global\_v 1.000000  
OP100\_Capuchon\_insertion\_mesure 0.040366  
OP090\_SnapRingFinalStroke\_value 0.015148  
OP090\_SnapRingMidPointForce\_val 0.014273  
OP090\_StartLinePeakForce\_value 0.010720  
OP110\_Vissage\_M8\_angle\_value 0.005470  
OP120\_Rodage\_I\_mesure\_value 0.003763  
OP110\_Vissage\_M8\_torque\_value -0.002984  
OP070\_V\_2\_angle\_value -0.006342  
OP090\_SnapRingPeakForce\_value -0.007290  
OP120\_Rodage\_U\_mesure\_value -0.010492  
OP070\_V\_1\_angle\_value -0.012793  
OP070\_V\_1\_torque\_value -0.037438  
OP070\_V\_2\_torque\_value -0.039752  
Name: Binar OP130\_Resultat\_Global\_v, dtype: float64

# 4 - Dessiner la Heatmap  
ImgUtil.save\_df\_heatmap\_plot(corr\_matrix,title.format('imputed+scaled'))

![](data:image/png;base64;base64,)

#### 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)

![](data:image/png;base64;base64,)

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)

![](data:image/png;base64;base64,)

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)

![](data:image/png;base64;base64,)

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)

![](data:image/png;base64;base64,)

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

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

## 9 - Analyse de la target

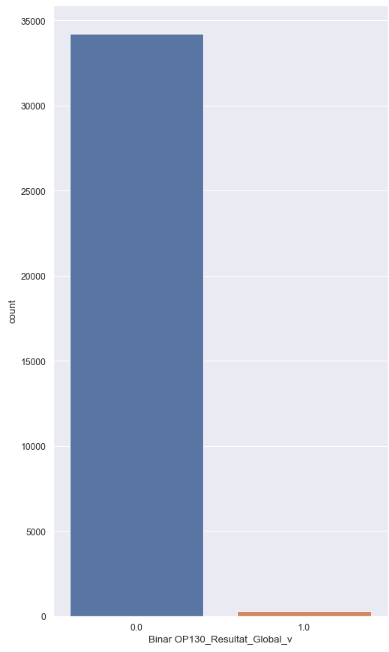
#### a - Vérification de l'équilibre des données:

starter\_count = len(Y\_data[Const.Binar\_OP130\_Resultat\_Global\_v])  
starter\_count\_ok = Y\_data[Const.Binar\_OP130\_Resultat\_Global\_v].value\_counts()[0]  
starter\_count\_ko = Y\_data[Const.Binar\_OP130\_Resultat\_Global\_v].value\_counts()[1]  
#   
print(f'Nombre total des démarreurs : {starter\_count}')  
print(f'Nombre total des démarreurs OK => Nombre de Classes Negatives : {starter\_count\_ok} soit {round(starter\_count\_ok/starter\_count \* 100,2)} % du dataset')  
print(f'Nombre total des démarreurs KO => Nombre de Classes Positives : {starter\_count\_ko} soit {round(starter\_count\_ko/starter\_count \* 100,2)} % du dataset')

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

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

plt.figure(figsize=(8, 15))  
sns.countplot(Const.Binar\_OP130\_Resultat\_Global\_v, data=XY\_data\_transformed)

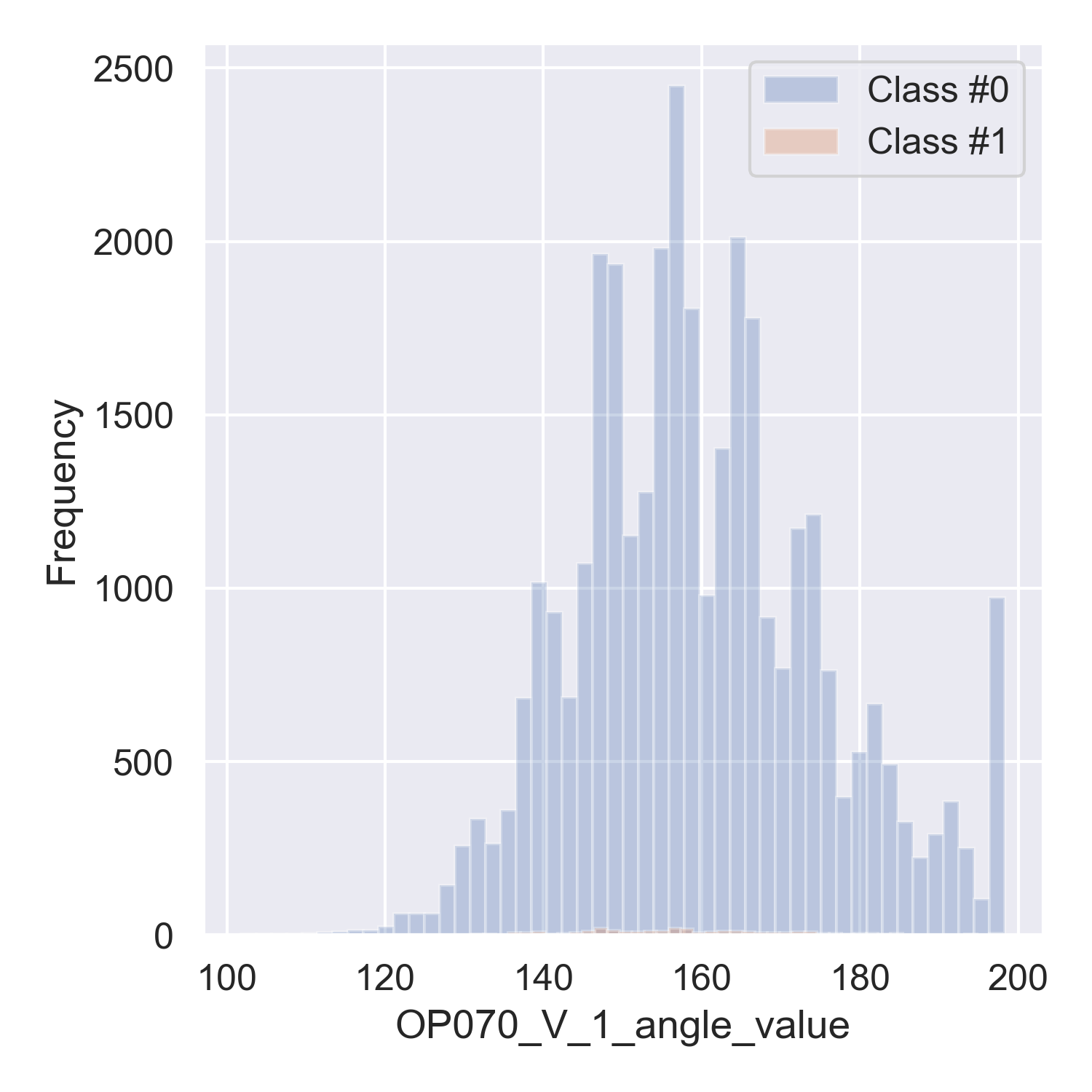
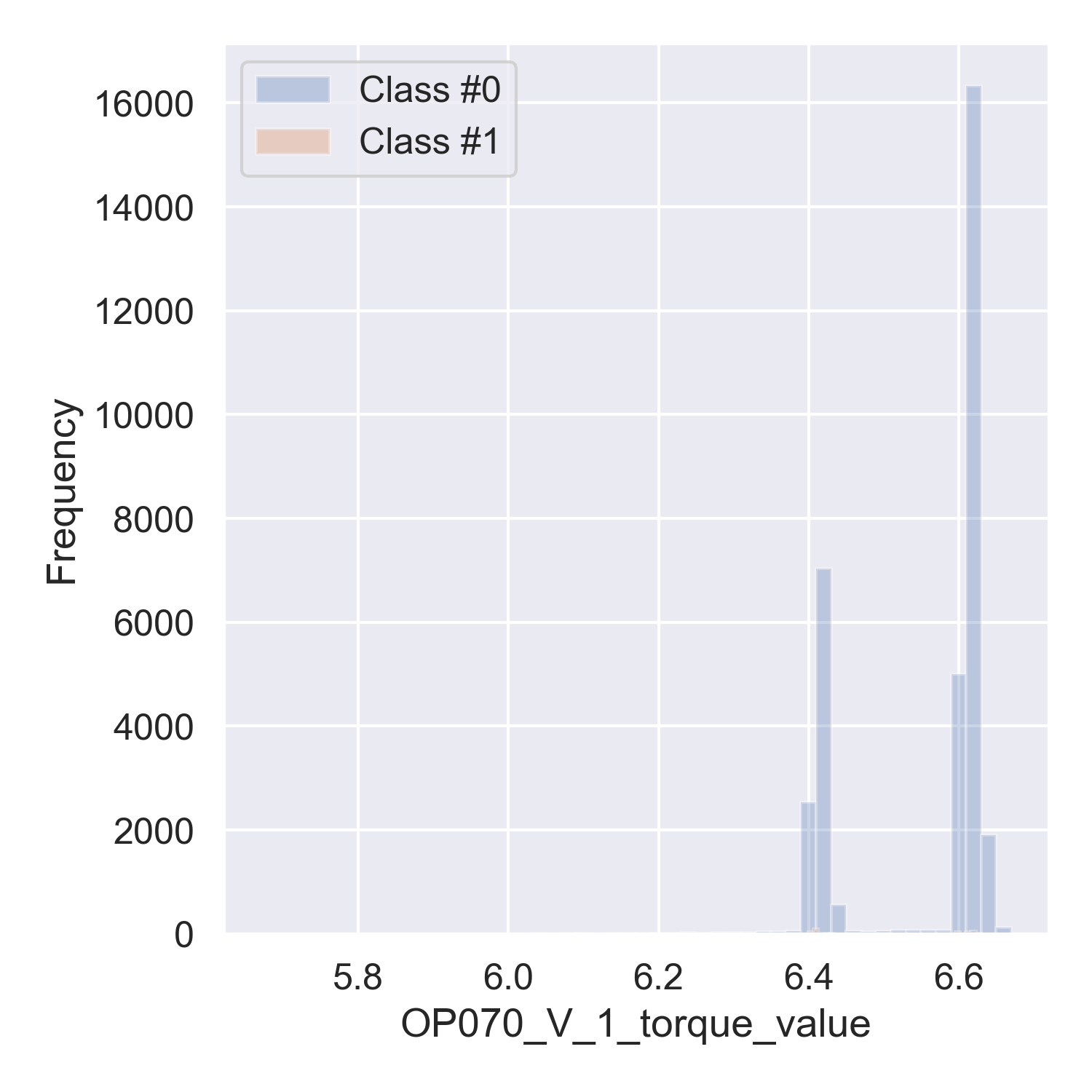
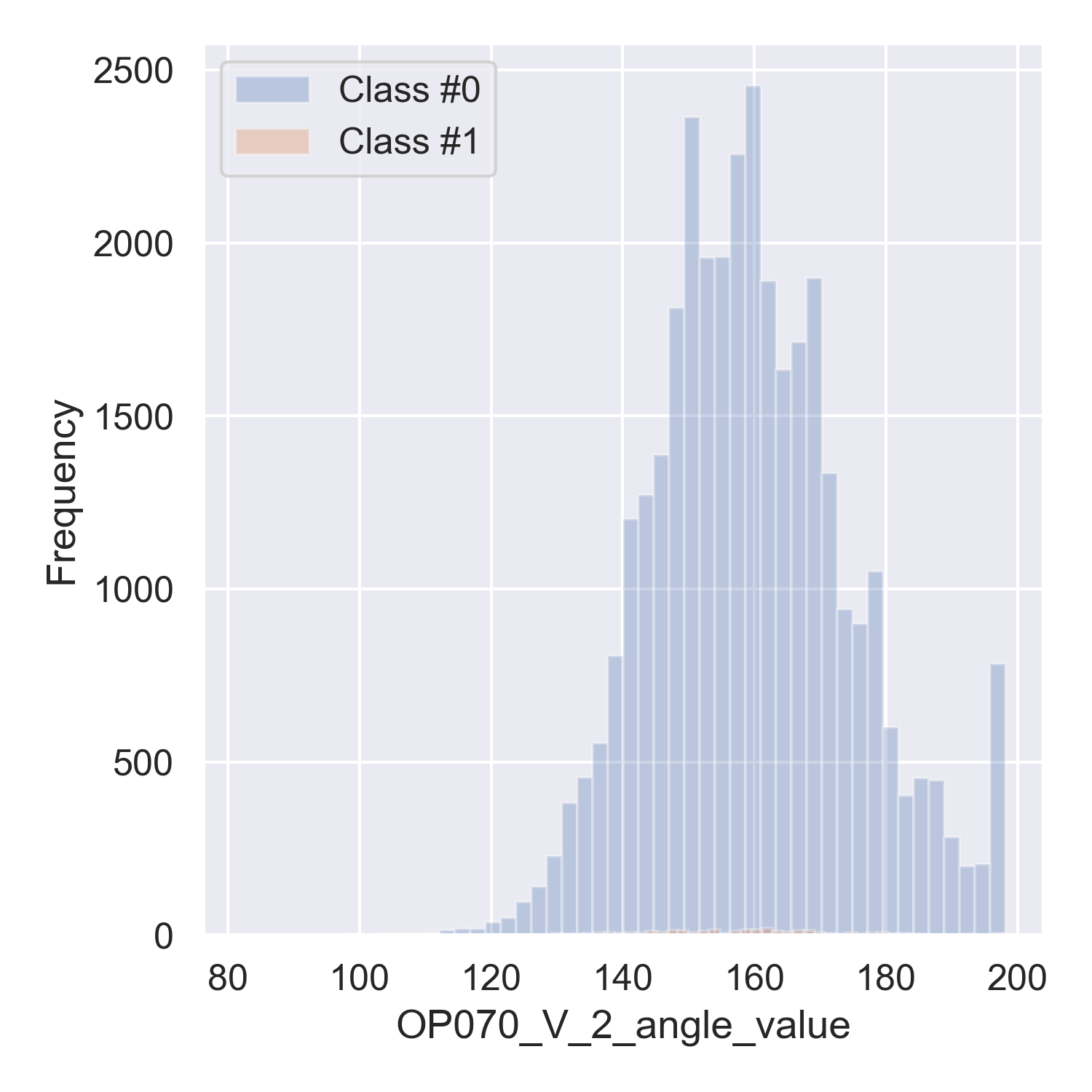
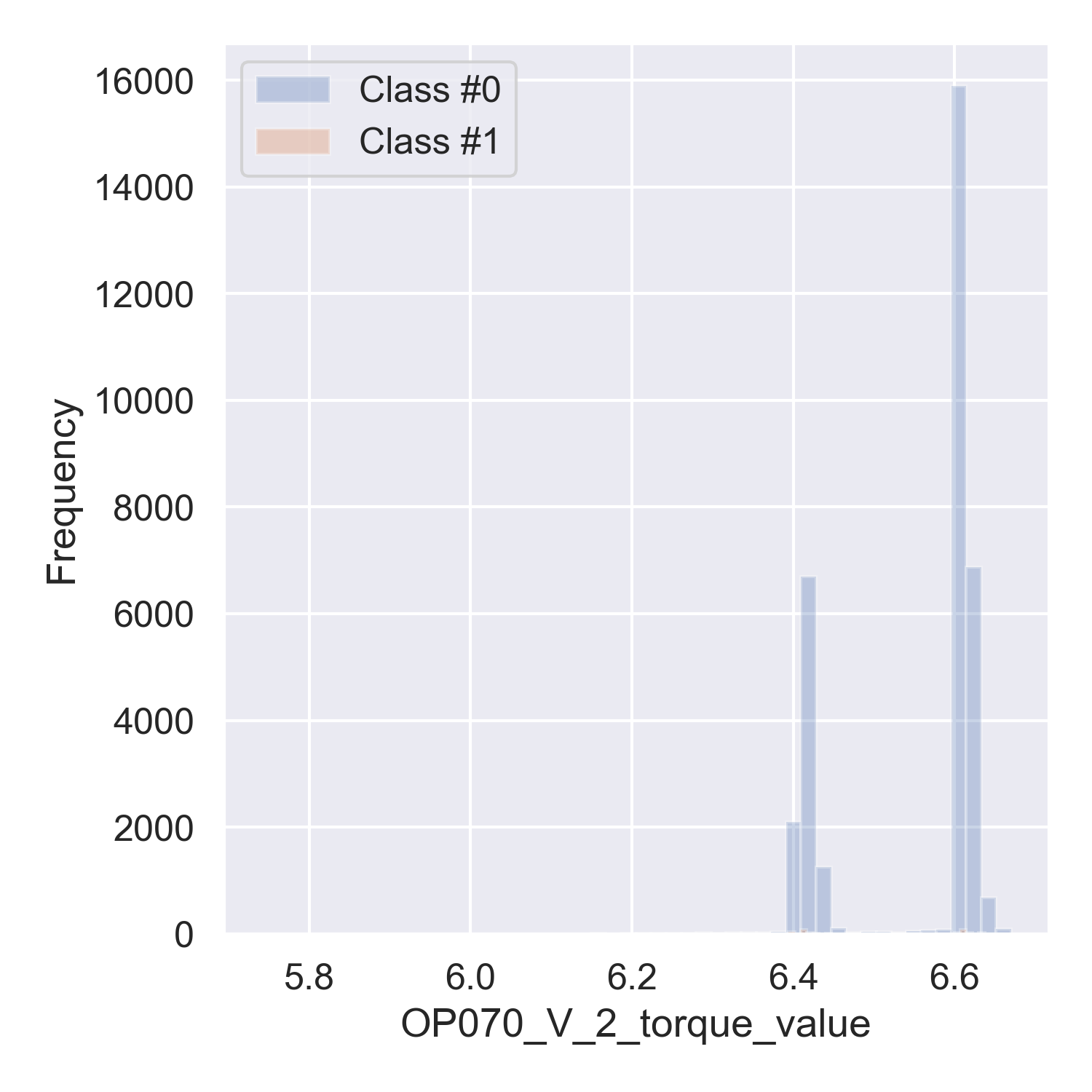
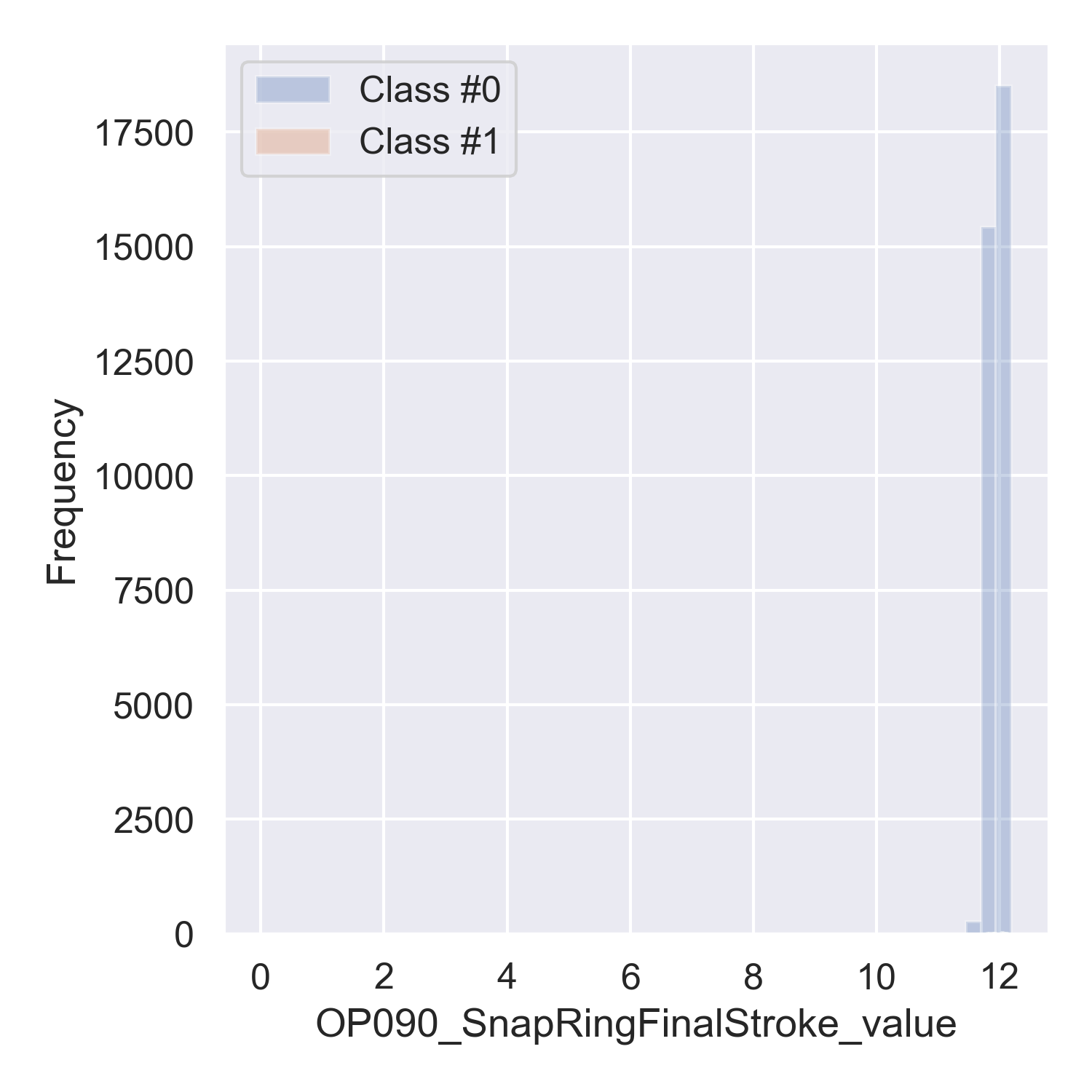
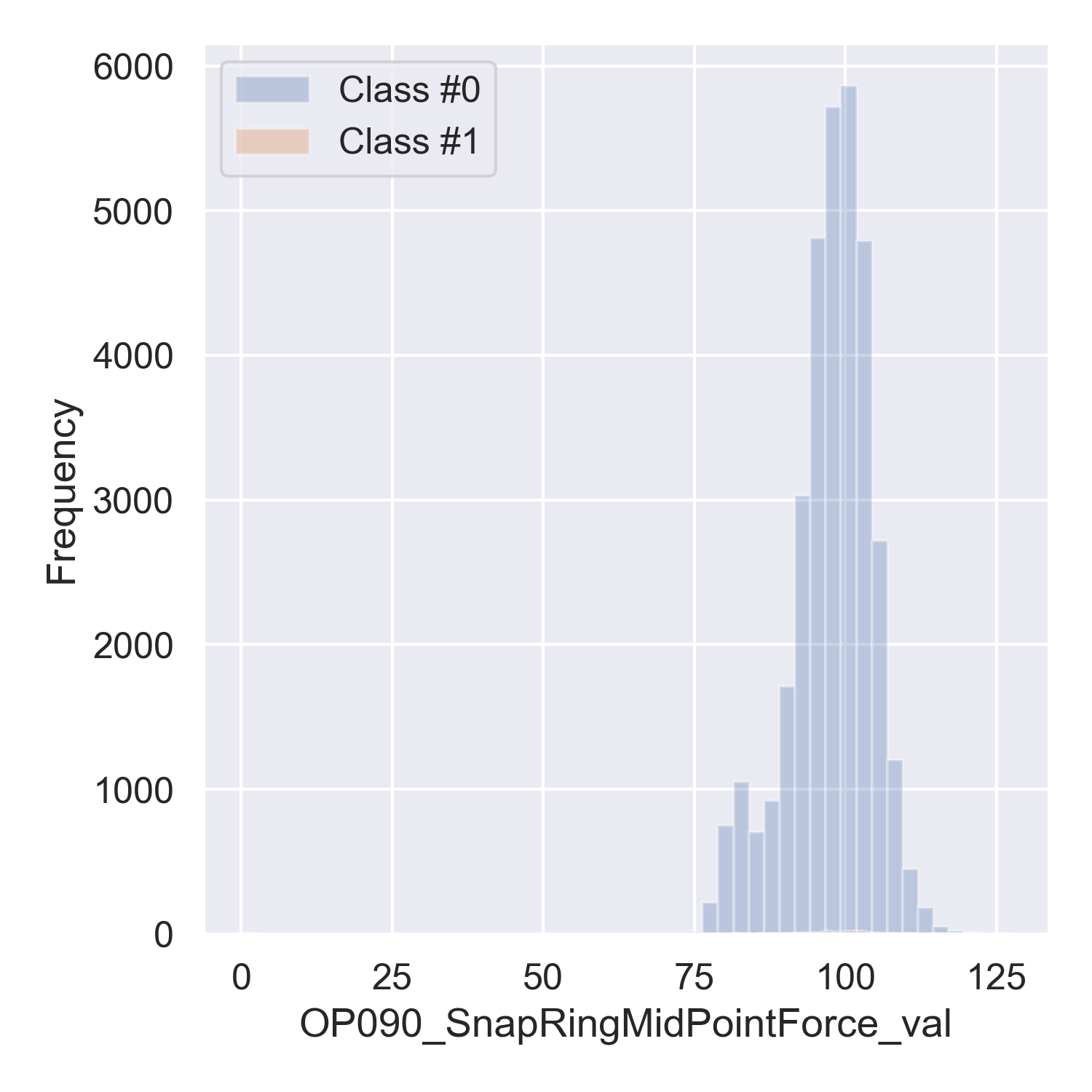
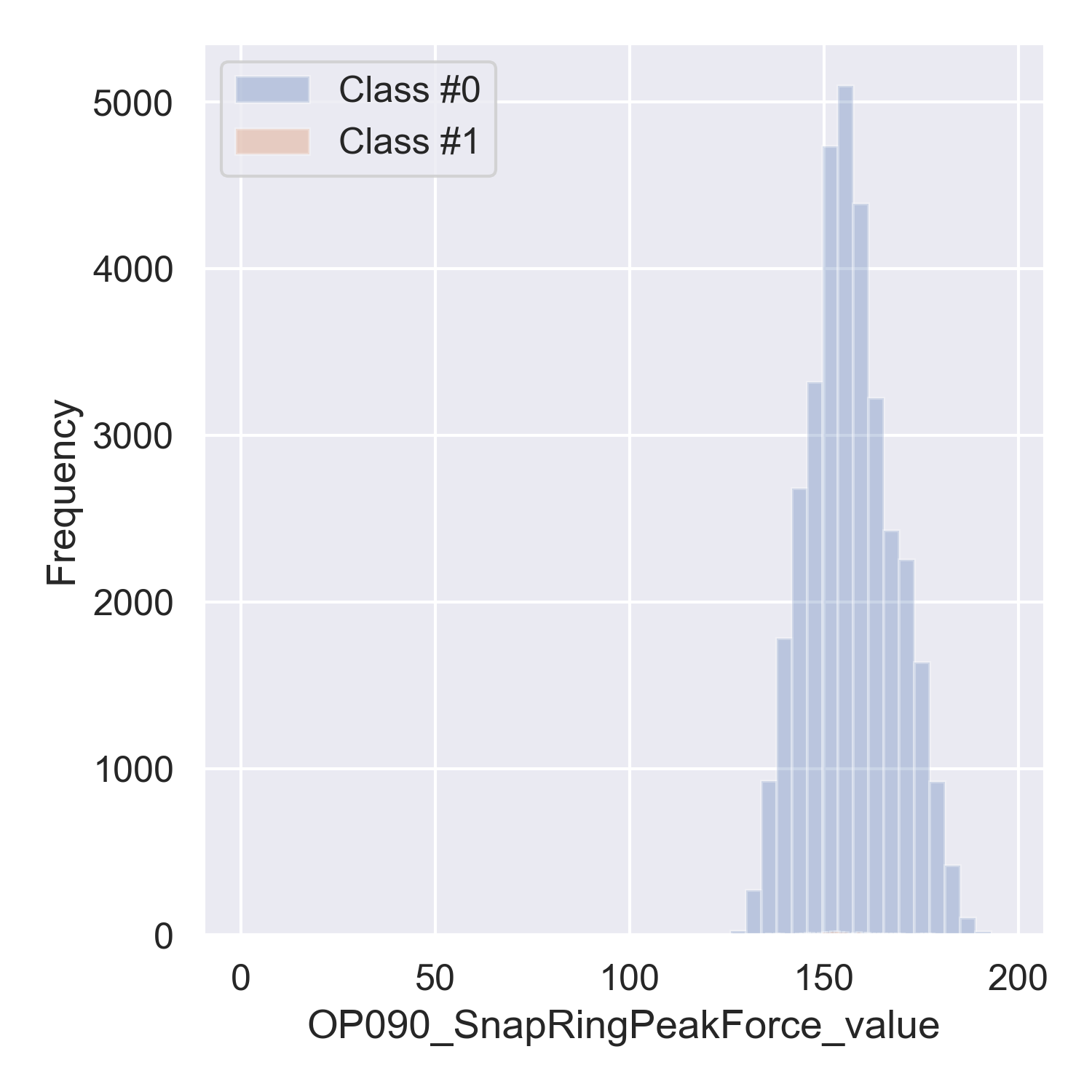
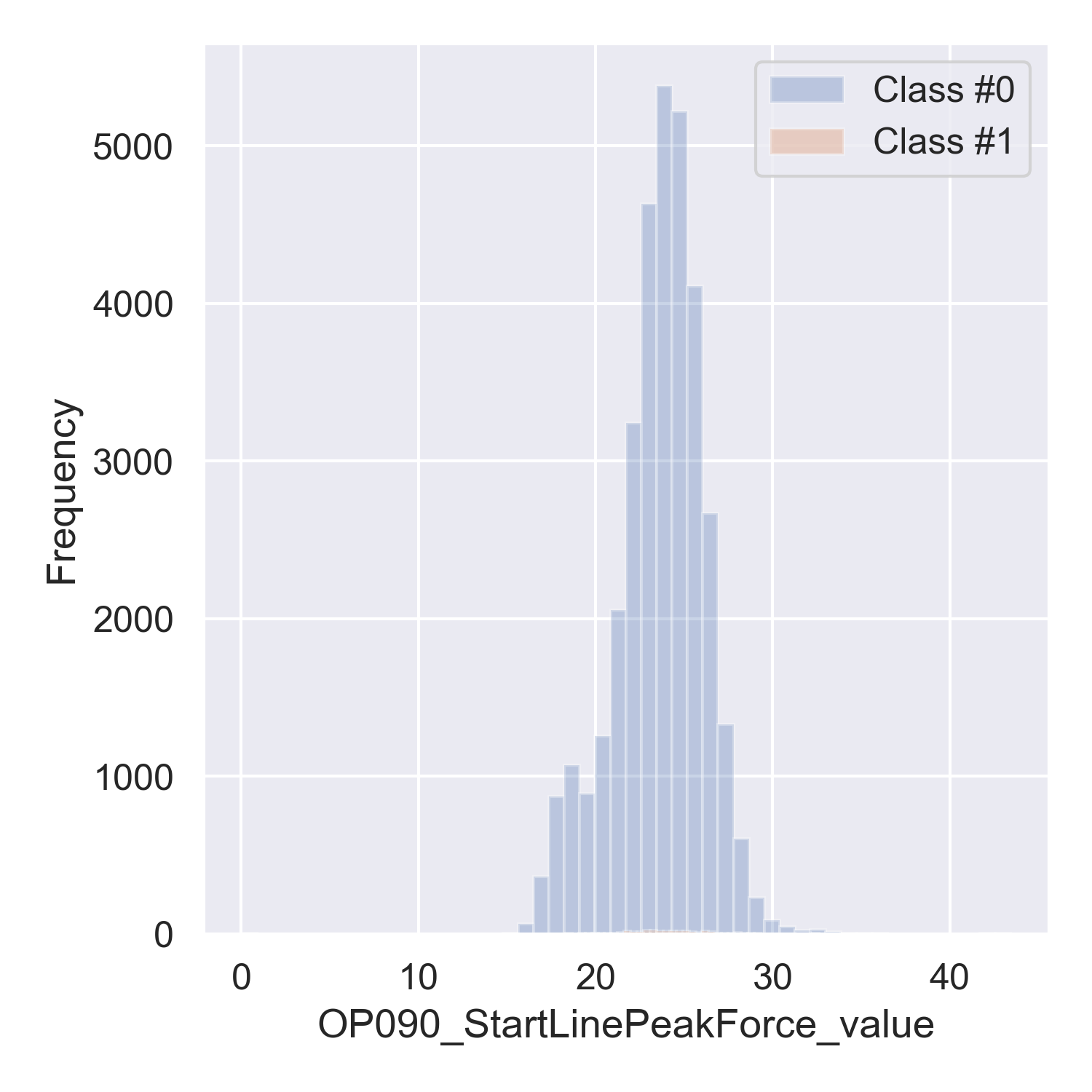
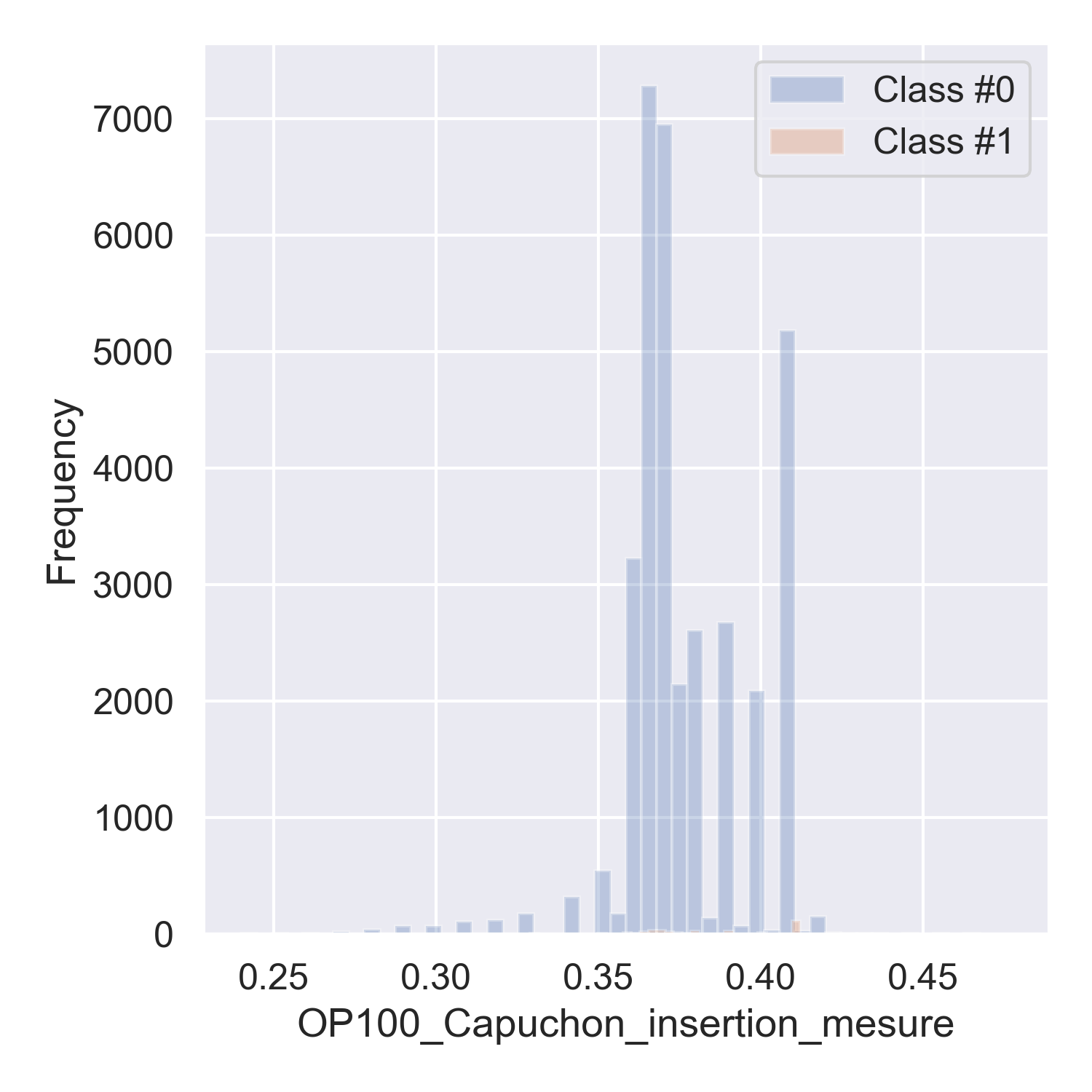
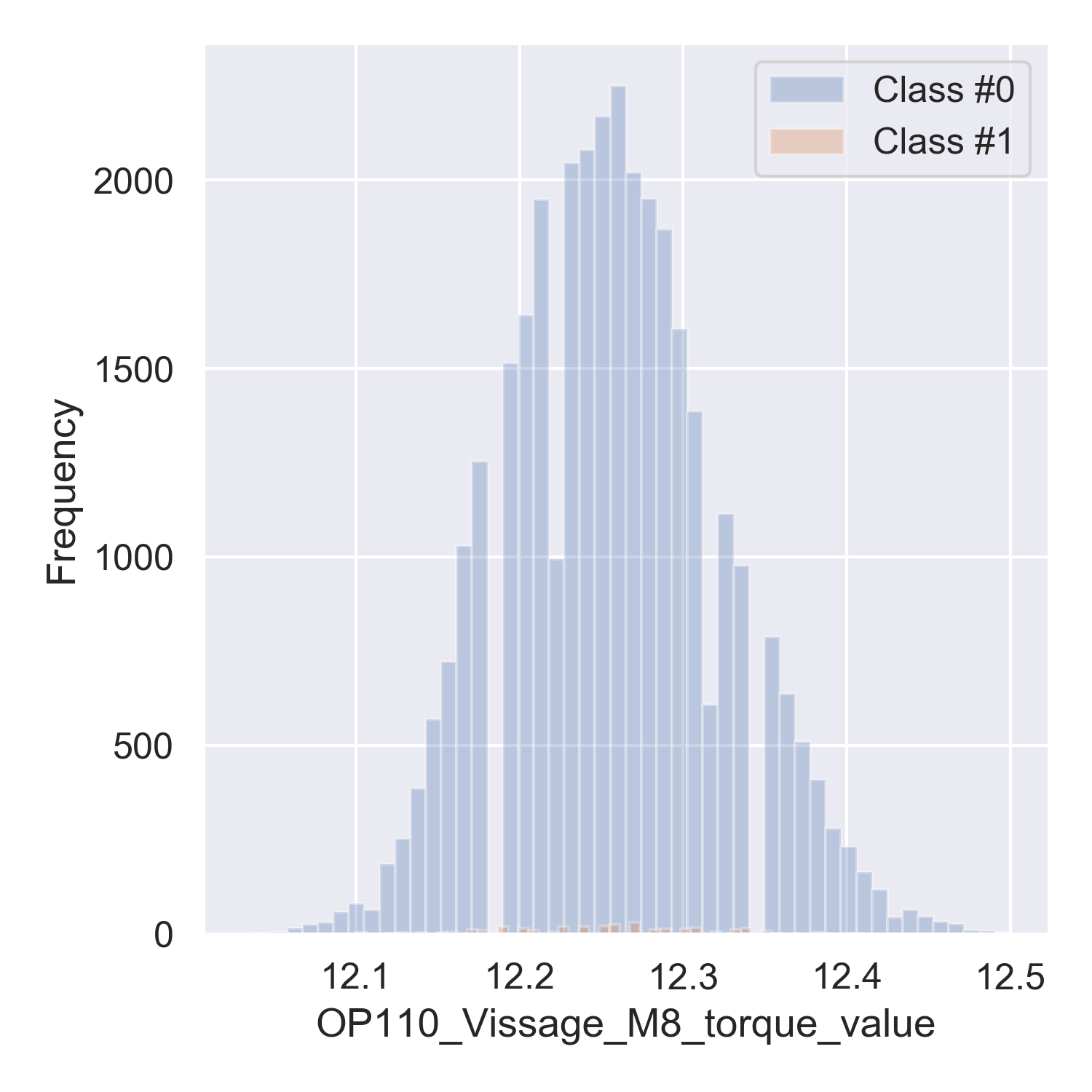
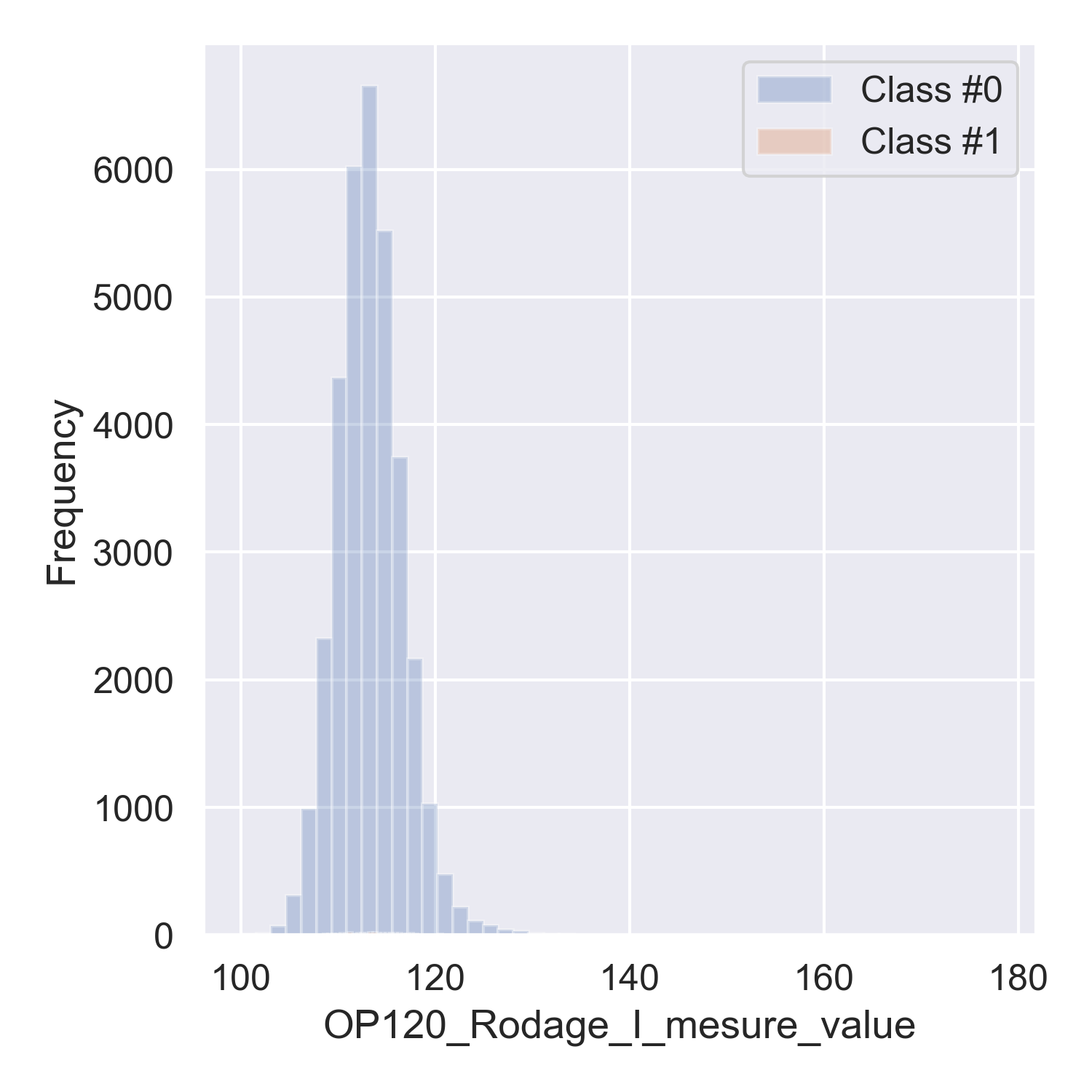
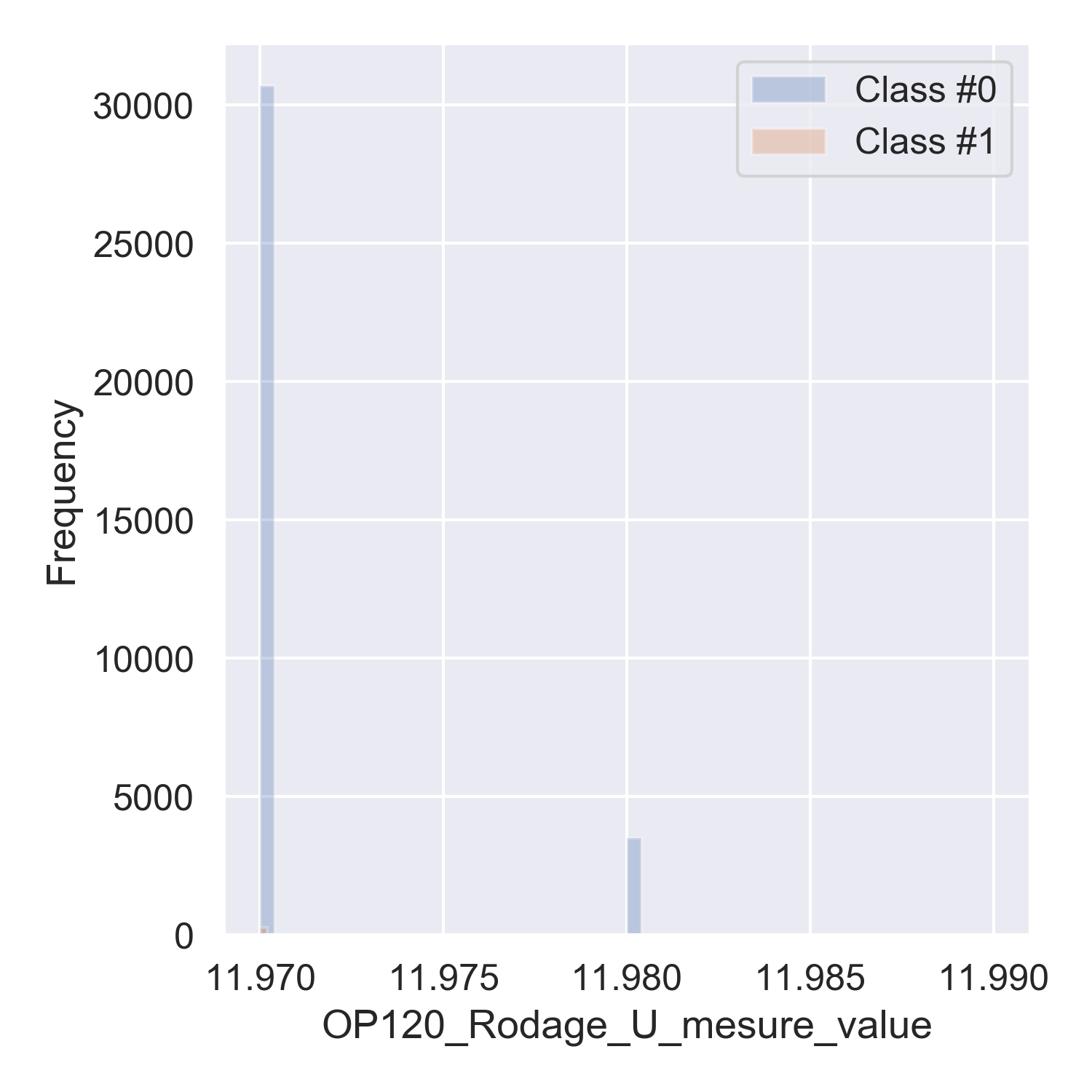


On constate que le jeu de données est fortement déséquilibré.\ Presque totalité des démarreurs (99.12%) ne sont pas défectueux lors de la sortie de la ligne de production.

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

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

ImgUtil.save\_df\_XY\_hist\_plot(XY\_data\_transformed, "XY\_imputed", y\_target\_name=Const.Binar\_OP130\_Resultat\_Global\_v)

On constate que la classe minoritaire se retrouve délimité à l'intérieur d'une plage de valeurs pour certains features. (ex: OP070\_V\_1/2\_angle\_value, OP110\_Vissage\_M8\_torque\_Value).

Pour cela, il faudrait vérifier l'impact si on transforme ces features numériques continues en des features catégoriques mettant en avant l'existence de la classe minoritaire KO pour ces catégories.

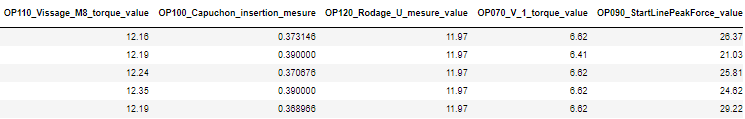
## 10 - Analyse de la target après un oversampling SMOTE

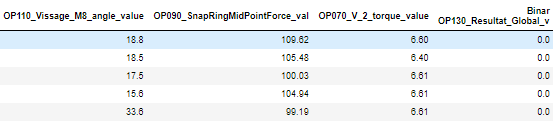
#### a - Regénération SMOTE de la classe minoriataire de la target:

sm = SMOTE(sampling\_strategy='minority', random\_state=7)  
#  
oversampled\_X, oversampled\_Y = sm.fit\_sample(XY\_data\_transformed.drop(Const.Binar\_OP130\_Resultat\_Global\_v, axis=1),   
 XY\_data\_transformed[Const.Binar\_OP130\_Resultat\_Global\_v])  
oversampled\_XY = pd.concat([pd.DataFrame(oversampled\_X), pd.DataFrame(oversampled\_Y)], axis=1)  
oversampled\_XY.columns = XY\_data\_transformed\_scaled.columns

oversampled\_XY.head()

#### 





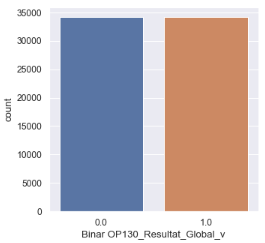
#### b - Statistique descriptive du nouveau dataset:

oversampled\_XY.describe().transpose()



#### c - Nouvelle distribution équilibrée du nouveau dataset:

plt.figure(figsize=(5, 5))  
sns.countplot(Const.Binar\_OP130\_Resultat\_Global\_v, data=oversampled\_XY)



#### d - Matrice de correlation et heatmap du nouveau dataset:

# 3 - Correlation entre la target "Binar OP130\_Resultat\_Global\_v" et les autres attributs  
corr\_matrix\_oversampled = oversampled\_XY.corr()  
corr\_matrix\_oversampled[Const.Binar\_OP130\_Resultat\_Global\_v].sort\_values(ascending=False)

Binar OP130\_Resultat\_Global\_v 1.000000  
OP100\_Capuchon\_insertion\_mesure 0.229819  
OP090\_SnapRingFinalStroke\_value 0.099822  
OP090\_SnapRingMidPointForce\_val 0.083845  
OP090\_StartLinePeakForce\_value 0.061239  
OP120\_Rodage\_I\_mesure\_value 0.024506  
OP110\_Vissage\_M8\_angle\_value 0.000205  
OP110\_Vissage\_M8\_torque\_value -0.012686  
OP070\_V\_2\_angle\_value -0.025901  
OP090\_SnapRingPeakForce\_value -0.049015  
OP120\_Rodage\_U\_mesure\_value -0.064017  
OP070\_V\_1\_angle\_value -0.076841  
OP070\_V\_1\_torque\_value -0.210033  
OP070\_V\_2\_torque\_value -0.217814  
Name: Binar OP130\_Resultat\_Global\_v, dtype: float64

# 4 - Dessiner la Heatmap  
ImgUtil.save\_df\_heatmap\_plot(corr\_matrix\_oversampled,title.format('oversampled\_imputed+scaled'))

![](data:image/png;base64;base64,)

#### 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)

![](data:image/png;base64;base64,)

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

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

Le ratio d'outlier est de 0.2762496346097632

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

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

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

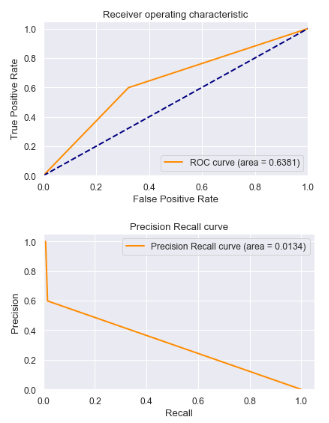
HGBC = HistGradientBoostingClassifier(max\_iter = 100 , max\_depth=10,learning\_rate=0.10, l2\_regularization=5)  
BBC = BalancedBaggingClassifier(base\_estimator=HistGradientBoostingClassifier(), n\_estimators=300, sampling\_strategy='auto', replacement=False, random\_state=48)  
BRFC = BalancedRandomForestClassifier(n\_estimators = 300 , max\_depth=20, random\_state=0)  
BRFC\_ = BalancedRandomForestClassifier(n\_estimators = 300 , max\_depth=20, random\_state=0, replacement=True)   
  
BRFC\_W = BalancedRandomForestClassifier(n\_estimators = 300 , max\_depth=20, random\_state=0, class\_weight={0:1, 1:1})   
RUSBoost = RUSBoostClassifier(n\_estimators = 8 , algorithm='SAMME.R', random\_state=42)  
XGBC = xgb.XGBClassifier(base\_score=0.5, booster='gbtree', colsample\_bylevel=1,  
 colsample\_bynode=1, colsample\_bytree=1, gamma=0,  
 learning\_rate=0.1, max\_delta\_step=0, max\_depth=10, #max\_depth=3,  
 min\_child\_weight=1, missing=None, n\_estimators=100, n\_jobs=1,  
 nthread=None, objective='binary:logistic', random\_state=0,  
 reg\_alpha=0, reg\_lambda=1, scale\_pos\_weight=100, seed=42,  
 silent=None, subsample=1, verbosity=1)   
KNN = KNeighborsClassifier(3)  
RFC = RandomForestClassifier(n\_estimators=300, max\_depth=10, max\_features=10, n\_jobs=4, class\_weight= {0:1,1:100})  
DTC = DecisionTreeClassifier()   
ADABoost = AdaBoostClassifier()  
GBC = GradientBoostingClassifier()  
LRC = LogisticRegression(max\_iter=500)   
# SVC = SVC(kernel="rbf", C=0.025, probability=True)  
# GNB = GaussianNB()  
# NuSVC = NuSVC(probability=True),  
# LinearSVC = LinearSVC(C=0.1, class\_weight={'1':100})  
# SGDClassifier = SGDClassifier(class\_weight='balanced')

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

# 1 - Rechargement des données  
mt\_train = XY\_metadata([Const.rootDataTrain(), 'traininginputs.csv'], [Const.rootDataTrain(), 'trainingoutput.csv'],   
 [Const.PROC\_TRACEINFO], [Const.PROC\_TRACEINFO],Const.Binar\_OP130\_Resultat\_Global\_v)  
xy\_loader = XY\_Loader();  
X\_df, y\_df = xy\_loader.load\_XY\_df(mt\_train)

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

# 2 - Split Training et Validation  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_df, y\_df, test\_size=0.3, random\_state=48, stratify=y\_df)  
  
# 3 - Imputer et Scaler + classifier  
modeler = ValeoModeler()  
pred = ValeoPredictor()  
pl = Pipeline([('preprocessor', modeler.build\_transformers\_pipeline(X\_train.dtypes)), # --> Imputer + Scaler  
 ('classifier', BRFC) # --> Balanced Random Forest Classifier   
 ])  
  
# 4 - Fit, train, predict and plot ROC and F1  
pl.fit(X\_train, y\_train)   
pred.predict\_and\_plot(pl,X\_test, y\_test)  
  
# 5 - Test using ENS data  
X\_ens = DfUtil.read\_csv([Const.rootDataTest() , "testinputs.csv"])  
y\_ens = pl.predict(X\_ens.drop(columns=[Const.PROC\_TRACEINFO]))  
DfUtil.write\_y\_csv(X\_ens[Const.PROC\_TRACEINFO], y\_ens, Const.Binar\_OP130\_Resultat\_Global\_v, [Const.rootDataTest() , "testoutput.csv"])

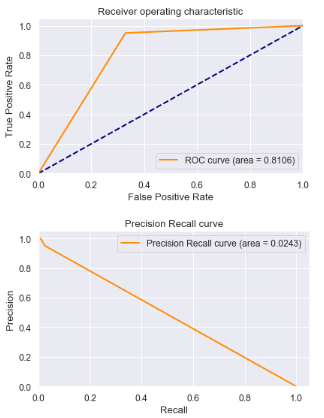
- Model score: 0.6776436504104297  
- Accuracy score: 0.6776436504104297  
- Balanced accuracy score: 0.6380926205999602 / The balanced accuracy to deal with imbalanced datasets. It is defined as the average of recall obtained on each class.  
- Average\_precision\_score: 0.013370660450719168  
- Precision\_score: 0.016388557806912993  
- Recall score: 0.5978260869565217  
**- Roc\_auc\_score: 0.6380926205999602  
- F1 score: 0.031902552204176336  
- [6962 3301]/[37 55] - P:0.0164 - R:0.5978 - roc\_auc:0.6381 - f1:0.0319**  
**- [[6962 3301]  
 [ 37 55]]**  
- Classification\_report\_imbalanced:  
 pre rec spe f1 geo iba sup  
  
 0 0.99 0.68 0.60 0.81 0.64 0.41 10263  
 1 0.02 0.60 0.68 0.03 0.64 0.40 92  
  
avg / total 0.99 0.68 0.60 0.80 0.64 0.41 10355  
  
- Classification\_report:  
 precision recall f1-score support  
  
 0 0.99 0.68 0.81 10263  
 1 0.02 0.60 0.03 92  
  
 accuracy 0.68 10355  
 macro avg 0.51 0.64 0.42 10355  
weighted avg 0.99 0.68 0.80 10355  
  


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

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

X\_train, y\_train = X\_df, y\_df  
  
# 2 - Initialize a CV Split  
CV = StratifiedKFold(n\_splits=8) # , random\_state=48, shuffle=True  
  
# 3 - Imputer et Scaler + classifier  
modeler = ValeoModeler()  
pred = ValeoPredictor()  
BRFC = BalancedRandomForestClassifier(n\_estimators = 300 , max\_depth=20, random\_state=0)  
pl = Pipeline([('preprocessor', modeler.build\_transformers\_pipeline(X\_train.dtypes)), # --> Imputer + Scaler  
 ('classifier', BRFC) # --> Balanced Random Forest Classifier   
 ])  
  
# 4 - Cross Validate  
cv\_results = cross\_validate(pl, X\_train, y\_train, cv=CV, scoring=('f1', 'f1\_micro', 'f1\_macro', 'f1\_weighted', 'recall', 'precision', 'average\_precision', 'roc\_auc'), return\_train\_score=True, return\_estimator=True)  
fitted\_estimators = []  
for key in cv\_results.keys() :  
 if str(key) != "estimator" :  
 print(f"{key} : {cv\_results[key]}")  
 fitted\_estimators.append(cv\_results[key])  
   
fitted\_model = cv\_results["estimator"][np.argmax(cv\_results["test\_roc\_auc"])]  
pred.predict\_and\_plot(fitted\_model,X\_test, y\_test)  
  
# 5 - Test using ENS data  
X\_ens = DfUtil.read\_csv([Const.rootDataTest() , "testinputs.csv"])  
y\_ens = fitted\_model.predict(X\_ens.drop(columns=[Const.PROC\_TRACEINFO]))  
DfUtil.write\_y\_csv(X\_ens[Const.PROC\_TRACEINFO], y\_ens, Const.Binar\_OP130\_Resultat\_Global\_v, [Const.rootDataTest() , "testoutput.csv"])

fit\_time : [4.20418215 3.88397908 3.76627254 3.901299 3.95574546 3.66990876  
 4.14086699 4.21298003]  
score\_time : [0.34399605 0.31799507 0.32755017 0.2992568 0.3776269 0.30384541  
 0.35573673 0.38599515]  
test\_f1 : [0.02617801 0.03069054 0.03129445 0.03215434 0.03556772 0.02931379  
 0.03125 0.02610587]  
train\_f1 : [0.05155933 0.04991121 0.05410353 0.0492302 0.05065933 0.0512476  
 0.05085714 0.05507426]  
test\_f1\_micro : [0.65515643 0.64866744 0.68435689 0.65113584 0.67315716 0.6622624  
 0.66944831 0.68868799]  
train\_f1\_micro : [0.6747351 0.6634106 0.69201987 0.65852124 0.66865336 0.67265985  
 0.67001093 0.69663256]  
test\_f1\_macro : [0.40832978 0.408071 0.42137812 0.40968667 0.41940261 0.41243997  
 0.41598833 0.42041947]  
train\_f1\_macro : [0.42763407 0.42269408 0.43508467 0.420559 0.42498069 0.4267282  
 0.42557285 0.43719279]  
test\_f1\_weighted : [0.78375072 0.77880466 0.80441046 0.78056799 0.79647547 0.7888166  
 0.7939487 0.80778642]  
train\_f1\_weighted : [0.79705901 0.78888536 0.80935449 0.78532214 0.79268348 0.79556972  
 0.79366302 0.81255488]  
test\_recall : [0.52631579 0.63157895 0.56410256 0.65789474 0.68421053 0.57894737  
 0.60526316 0.47368421]  
train\_recall : [1. 1. 1. 1. 1. 1. 1. 1.]  
test\_precision : [0.01342282 0.01572739 0.01609364 0.01647989 0.01825843 0.01503759  
 0.01603905 0.01342282]  
train\_precision : [0.02646184 0.02559433 0.02780391 0.02523629 0.02598793 0.02629765  
 0.02609206 0.02831689]  
test\_average\_precision : [0.01460394 0.01852067 0.0140107 0.02250251 0.03696366 0.01684649  
 0.01927391 0.01715631]  
train\_average\_precision : [0.36489053 0.38781819 0.34316751 0.43090551 0.37367985 0.36632035  
 0.38315343 0.36073824]  
test\_roc\_auc : [0.61249892 0.64770252 0.63006404 0.70079637 0.73938383 0.65292206  
 0.66054724 0.62940648]  
train\_roc\_auc : [0.95913257 0.96622369 0.95572754 0.96683853 0.96490919 0.96163445  
 0.96284347 0.96020734]  
- Model score: 0.6728958423873678  
- Accuracy score: 0.6728958423873678  
- Balanced accuracy score: 0.8106188392810079 / The balanced accuracy to deal with imbalanced datasets. It is defined as the average of recall obtained on each class.  
- Average\_precision\_score: 0.024277023825104087  
- Precision\_score: 0.025075659316904454  
**- Recall score: 0.9508196721311475  
- Roc\_auc\_score: 0.8106188392810079  
- F1 score: 0.04886267902274642  
- [4587 2255]/[ 3 58] - P:0.0251 - R:0.9508 - roc\_auc:0.8106 - f1:0.0489  
- [[4587 2255]  
 [ 3 58]]**  
- Classification\_report\_imbalanced:  
 pre rec spe f1 geo iba sup  
  
 0 1.00 0.67 0.95 0.80 0.80 0.62 6842  
 1 0.03 0.95 0.67 0.05 0.80 0.66 61  
  
avg / total 0.99 0.67 0.95 0.80 0.80 0.62 6903  
  
- classification\_report:  
 precision recall f1-score support  
  
 0 1.00 0.67 0.80 6842  
 1 0.03 0.95 0.05 61  
  
 accuracy 0.67 6903  
 macro avg 0.51 0.81 0.43 6903  
weighted avg 0.99 0.67 0.80 6903  
  
- precision\_recall\_curve: (array([0.00883674, 0.02507566, 1. ]), array([1. , 0.95081967, 0. ]), array([0, 1], dtype=int64))  
- precision\_recall\_fscore\_support: (array([0.99934641, 0.02507566]), array([0.67041801, 0.95081967]), array([0.80248425, 0.04886268]), array([6842, 61], dtype=int64))  
- roc\_curve: (array([0. , 0.32958199, 1. ]), array([0. , 0.95081967, 1. ]), array([2, 1, 0], dtype=int64))

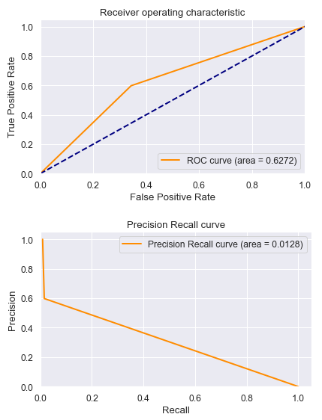


*NB: LA SURFACE F1 CALCULée n'est pas correcte (base x hauteur / 2) => A corriger*

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

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_df, y\_df, test\_size=0.3, random\_state=48, stratify=y\_df)  
  
# 3 - Imputer et Scaler + SMOTE + Logistic Regression  
modeler = ValeoModeler()  
pred = ValeoPredictor()  
pl = Pipeline([('preprocessor', modeler.build\_transformers\_pipeline(X\_train.dtypes)), # --> Imputer + Scaler  
 ('imbalancer\_resampler', SMOTE(sampling\_strategy='minority', random\_state=7)), # --> SMOTE oversampling  
 ('classifier', LRC) # --> Logistic Regression Classifier  
 ])  
pl.fit(X\_train, y\_train)   
pred.predict\_and\_plot(pl,X\_test, y\_test)  
  
# 5 - Test using ENS data  
X\_ens = DfUtil.read\_csv([Const.rootDataTest() , "testinputs.csv"])  
y\_ens = pl.predict(X\_ens.drop(columns=[Const.PROC\_TRACEINFO]))  
DfUtil.write\_y\_csv(X\_ens[Const.PROC\_TRACEINFO], y\_ens, Const.Binar\_OP130\_Resultat\_Global\_v, [Const.rootDataTest() , "testoutput.csv"])

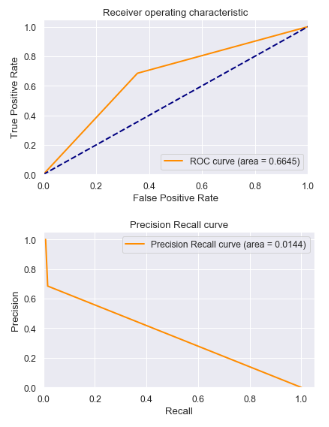
- Model score: 0.6560115886045389  
- Accuracy score: 0.6560115886045389  
- Balanced accuracy score: 0.6271796321950103 / The balanced accuracy to deal with imbalanced datasets. It is defined as the average of recall obtained on each class.  
- Average\_precision\_score: 0.012757632055707119  
**- Precision\_score: 0.015363128491620111  
- Recall score: 0.5978260869565217  
- Roc\_auc\_score: 0.6271796321950104  
- F1 score: 0.02995642701525054  
- [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