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

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

import pandas as pd
import numpy as np

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

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

class ValeoModeler :
    logger = None

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

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

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

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

    def build_predictor_pipeline(self, features_dtypes:pd.Series, clfTypes:[str]) -> Pipeline:
        cls = self.__class__
        clfs = {
```

```python
            cls.HGBC : HistGradientBoostingClassifier(max_iter = 100 , max_depth=10,learning_rate=0.10, l2_regularization=5),
            cls.BBC  : BalancedBaggingClassifier(base_estimator=HistGradientBoostingClassifier(),  n_estimators=50, sampling_strategy='auto', replacement=False, random_state=48),

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

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


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

```
            raise ValueError(f"Unexpected argument [sampler_type:{sampler_type}] for method 'compute_sampler_preprocessor'")


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

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

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

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

from valeo.infrastructure.tools.ImgUtil import ImgUtil


class MetricPlotter :
    logger = None

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

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

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

```

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

import pandas as pd
from sklearn.model_selection import train_test_split

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


class ValeoPipeline:
    logger = None

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

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

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

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

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

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


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



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

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



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

```python

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

import pandas as pd

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

import xgboost as xgb


class ValeoPredictor :
    logger = None

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


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

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

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

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

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

    # 1 - Fit without any Cross Validation
    def fit_and_plot(self, X_train:pd.DataFrame, y_train:pd.DataFrame,  X_test:pd.DataFrame, y_test:pd.DataFrame, clfTypes:[str]) -> BaseEstimator:
        # Q1 = X_train.quantile(0.25)
        # Q3 = X_train.quantile(0.75)
        # IQR = Q3 - Q1
        # # print(IQR)
        # to_remove = ((X_train < (Q1 - 1.5 * IQR)) |(X_train > (Q3 + 1.5 * IQR))).any(axis=1)
```

```
        # y_train = y_train.drop(axis=0, index=X_train[to_remove].index)
        # X_train = X_train[~to_remove]
        #
        fitted_model = self.fit(X_train, y_train, clfTypes)
        # print(f"Type:{type(fitted_model)} - {fitted_model.get_params()}")
        # self.print_model_params_keys(fitted_model)
        self.predict_and_plot(fitted_model, X_test, y_test)
        return fitted_model

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

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

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

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

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

    '''
        - Print metrics
        - Print report
        - Plot ROC : TP vs FP
        - Plot AUC : Precison vs Recall
    '''
    def predict_and_plot(self, fitted_model: BaseEstimator, X_test:pd.DataFrame, y_test:pd.DataFrame):
        y_pred = fitted_model.predict(X_test)
        #
        print(f"- Model score: {fitted_model.score(X_test, y_test)}")
        print(f"- Accuracy score: {accuracy_score(y_test, y_pred)}")
        print(f"- Balanced accuracy score: {balanced_accuracy_score(y_test, y_pred)} / The balanced accuracy to deal with imbalanced datasets. It is defined as the average of recall obtained on each class.")
        # print(f"- auc : {auc(y_test, y_pred)}")  # ValueError: x is neither increasing nor decreasing : [0 0 0 ... 0 0 0]
        print(f"- Average_precision_score: {average_precision_score(y_test, y_pred)}")
        print(f"- Precision_score: {precision_score(y_test, y_pred)}")
```

```
        print(f"- Recall score: {recall_score(y_test, y_pred)}")
        print(f"- Roc_auc_score: {roc_auc_score(y_test, y_pred)}")
        print(f"- F1 score: {f1_score(y_test, y_pred)}")
        m = confusion_matrix(y_test, y_pred)
        print(f"- {m[0]}/{m[1]} - P:{precision_score(y_test, y_pred):0.4f} - R:{recall_score(y_test, y_pred):0.4f} - roc_auc:{roc_auc_score(y_test, y_pred):0.4f} - f1:{f1_score(y_test, y_pred):0.4f}")
        print(f"- {confusion_matrix(y_test, y_pred)}")
        print(f"- classification_report_imbalanced:\n{classification_report_imbalanced(y_test, y_pred)}")
        print(f"- classification_report:\n{classification_report(y_test, y_pred)}")
        print(f"- precision_recall_curve: {precision_recall_curve(y_test, y_pred)}")
        print(f"- precision_recall_fscore_support: {precision_recall_fscore_support(y_test, y_pred)}")
        print(f"- roc_curve: {roc_curve(y_test, y_pred)}")
        #
        self.metricPlt.plot_roc(y_test, y_pred)
        self.metricPlt.plot_precision_recall(y_test, y_pred)
        # self.plot_roc(y_test, y_pred)
        # self.plot_precision_recall(y_test, y_pred)



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

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

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

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

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

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


''' TODO :

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

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

'''
```

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

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

import pandas as pd
import numpy as np

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

from valeo.infrastructure.tools.DebugPipeline import DebugPipeline


class ValeoPreprocessor:
    logger = None

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

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

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

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

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

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


    def execute(self, X_train:pd.DataFrame):
        imputed_nan_cols = [C.OP100_Capuchon_insertion_mesure]
        imputed_zeroes_cols = [C.OP100_Capuchon_insertion_mesure, C.OP090_StartLinePeakForce_value, C.OP090_SnapRingMidPointForce_val,  # columns equals to 0 for a few rows
                               C.OP090_SnapRingPeakForce_value, C.OP090_SnapRingFinalStroke_value]
        scaled_cols = [C.OP070_V_1_angle_value, C.OP070_V_1_torque_value,
```

```
                       C.OP070_V_2_angle_value, C.OP070_V_2_torque_value,
                       C.OP090_StartLinePeakForce_value, C.OP090_SnapRingMidPointForce_val,
                       C.OP090_SnapRingPeakForce_value,  C.OP090_SnapRingFinalStroke_value,
                       C.OP100_Capuchon_insertion_mesure,
                       C.OP110_Vissage_M8_angle_value, C.OP110_Vissage_M8_torque_value,
                       C.OP120_Rodage_I_mesure_value,  C.OP120_Rodage_U_mesure_value]
        # DebugPipeline.counter += 10
        d = DebugPipeline()
        #
        d.fit_transform(X_train[scaled_cols])
        s = SimpleImputer(strategy='mean', missing_values=np.nan, verbose=True)
        X_train[imputed_nan_cols] = s.fit_transform(X_train[imputed_nan_cols])
        d.fit_transform(X_train[scaled_cols])
        #
        s = SimpleImputer(strategy='mean', missing_values=0.0, verbose=True)
        X_train[imputed_zeroes_cols] = s.fit_transform(X_train[imputed_zeroes_cols])
        d.fit_transform(X_train[scaled_cols])
        #
        scaler      = MinMaxScaler() # StandardScaler() # RobustScaler(with_centering=True, with_scaling=False)
        X_train[scaled_cols]  = scaler.fit_transform(X_train[scaled_cols])
        d.fit_transform(X_train[scaled_cols])



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

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

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

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

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



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



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



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

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


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

```

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

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

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

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

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


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

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

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

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

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

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

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

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

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

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

```python
import os

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

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


class XY_Loader:
    logger = None

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

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


    def load_XY_df(self, mt: XY_metadata, delete_XY_join_cols=True) -> ():
        # X_df = pd.read_csv(mt.X_pathname, na_values='')  # NaN
        X_df = pd.read_csv(mt.X_pathname)  # NaN
        # print(X_df[Const.OP100_Capuchon_insertion_mesure].head(20))
        # X_df[[Const.OP100_Capuchon_insertion_mesure]] = X_df[[Const.OP100_Capuchon_insertion_mesure]].fillna(0.0)
        # print(X_df[Const.OP100_Capuchon_insertion_mesure].head(20))

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

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

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

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

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

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

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

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

class LogManager():

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

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


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

```

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

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

from valeo.infrastructure.LogManager import LogManager


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

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

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

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

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

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

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

```python
import os


class XY_metadata :

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

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

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

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

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

from valeo.infrastructure.tools.DfInDfOut import DfInDfOut


class SimpleImputer(_SimpleImputer, DfInDfOut):

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

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

from valeo.infrastructure.tools.DfInDfOut import DfInDfOut


class StandardScaler(_StandardScaler, DfInDfOut):

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

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

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

class AppConfigManager():

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

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



class AppConfigLoader(ConfigLoader) :

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

```python
from datetime import datetime

from pandas import Series
from sklearn.base import BaseEstimator

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

import os
import pandas as pd
import numpy as np

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


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

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

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

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

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

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

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

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

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

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

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

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

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

```

```python

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


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

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


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


```

```python

import pandas as pd

class DfInDfOut:

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

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

```python

import os
import yaml
import logging

# from infrastructure.LogManager import LogManager

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

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

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

        return None

```

File - C:\EXED\Training___VALEO\src\valeo\infrastructure\tools\ConfigLoader.py

```python

import os
import logging

from valeo.infrastructure.tools.YamlLoader import YamlLoader

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

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

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

```

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

```python

import os
from datetime import datetime

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

from valeo.infrastructure import Const as C


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

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

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

        return X

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