Documentation technique *pipeline framework*

Auteur :
Jérémy Combe

Table des matières

1 Class Best Estimator

Cette *class* contient plusieurs méthodes permettant de visualiser les *feature importance* avec plusieurs metrics, les corrélations.
Elle contient également le *pipeline* permettant de faire un choix d'algorithme automatique et de l'optimiser ainsi que différentes méthodes pour la prédiction.

1.1 Call

L'appel de cette *class* se fait de la manière suivante :

```
BE = BestEstimator(type_esti = 'Classifier')
```

type_ esti correspond au type d'estimateur que l'on souhaite utiliser, *Classifier* ou *Regressor*.

1.2 Feature Importance et corrélations

Ces méthodes donnent les variables qui ont le plus d'importance dans nos modèles ce qui permet de donner une interprétabilité.

1.2.1 Metric Gini

Cette méthode donne les *feature importance* en utilisant la metric Gini.

```
# Feature importance using Gini score

BE.Feature_Importances_Tree(Train,
                            Target,
                            ID = 'ID',            # ID column name to drop
                            value = 0,            # Value for filling NaN
                            n = 1000,             # sample size
                            figsize = (15,10),    # set the figure size
                            nb_features = 15)     # Show this number of highes
                                                  # feature importance score
```

— Train correspond au dataset d'entrainement
— Target est la variable à prédire
— ID est le nom de la variable correspondant à la clé primaire de chaque observation
— *value* est la valeur avec laquelle on souhaite remplir les valeurs manquantes
— n est la taille d'échantillon à utiliser pour l'analyse

— *figsize* est la taille de la figure
— *nb_features* correspond au nombre des plus importantes *features* à visualiser

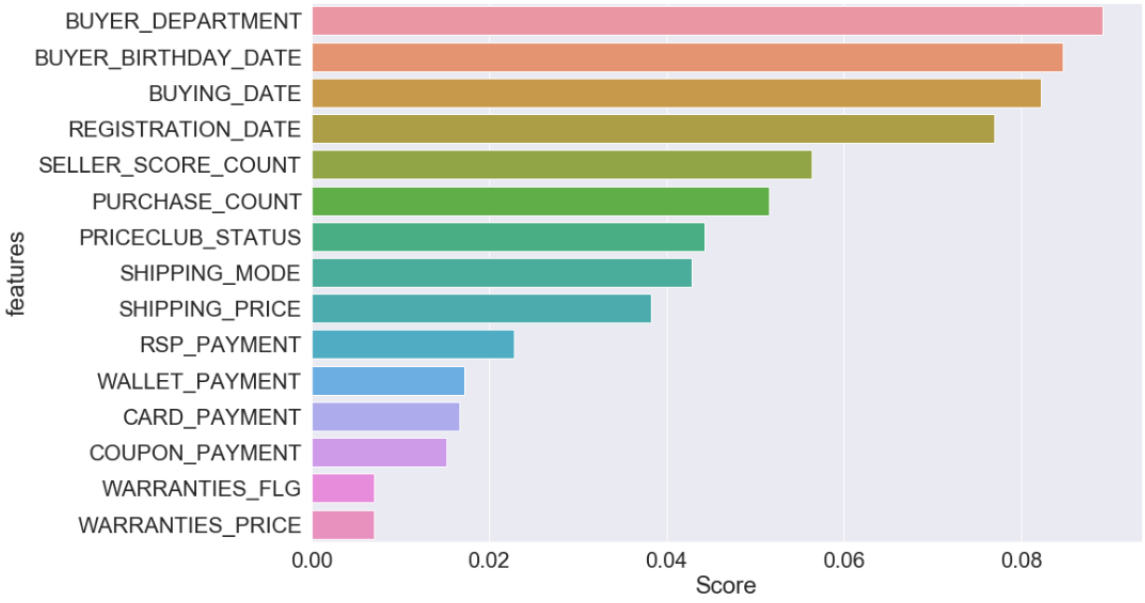

FIGURE 1 – Sortie de la méthode *Feature_Importance_Tree*

1.2.2 Statistique de test

Cette méthode utilise les statistiques de test pour donner les *feature importance*. Les statistiques de test utilisables sont disponibles ici.

```
# Feature importance using a statistic test to choose

BE.Feature_Importances_Test(Train,
                            Target,
                            ID='ID',
                            value=0,
                            n=1000,
                            nb_features=15,
                            test_used = f_classif)
```

— Train correspond au dataset d'entrainement
— Target est la variable à prédire
— ID est le nom de la variable correspondant à la clé primaire de chaque observation

— *value* est la valeur avec laquelle on souhaite remplir les valeurs manquantes
— n est la taille d'échantillon à utiliser pour l'analyse
— *figsize* est la taille de la figure
— *nb_features* correspond au nombre des plus importantes *features* à visualiser
— *test_used* est le type de test à utiliser parmi ceux ici

	Features	f_classif
0	SELLER_SCORE_AVERAGE	17.216671
1	ITEM_PRICE	8.594897
2	SELLER_SCORE_COUNT	5.947787
3	SELLER_DEPARTMENT	3.133100
4	SHIPPING_MODE	2.481947
5	REGISTRATION_DATE	2.361468
6	SHIPPING_PRICE	2.288408
7	PURCHASE_COUNT	1.875314
8	BUYER_BIRTHDAY_DATE	1.792215
9	SELLER_COUNTRY	1.395068
10	PRODUCT_FAMILY	1.117381
11	BUYING_DATE	0.939125
12	PRODUCT_TYPE	0.936226
13	PRICECLUB_STATUS	0.929278
14	CARD_PAYMENT	0.919525

FIGURE 2 – Sortie de la méthode *Feature_Importance_Test*

1.2.3 Corrélations feature - Target

La méthode suivante donne les variables les plus corrélées avec la Target (en valeur absolue).

```
# Display the most corralated features with the target
# and show tha absolute correlation

BE.get_highest_corr_target(Train,
                           Target,
                           ID='ID',
                           value=0,
                           n = 500,
                           nb_features = 15)
```

— Train correspond au dataset d'entrainement
— Target est la variable à prédire
— ID est le nom de la variable correspondant à la clé primaire de chaque observation
— *value* est la valeur avec laquelle on souhaite remplir les valeurs manquantes
— *n* est la taille d'échantillon à utiliser pour l'analyse
— *nb_features* correspond au nombre des plus importantes *features* à visualiser

	features	Target Correlation
0	SELLER_SCORE_AVERAGE	0.199755
1	ITEM_PRICE	0.185426
2	PRODUCT_TYPE	0.135933
3	SHIPPING_PRICE	0.095060
4	SELLER_SCORE_COUNT	0.091340
5	WARRANTIES_PRICE	0.081010
6	PRODUCT_FAMILY	0.076133
7	SELLER_COUNTRY	0.072617
8	WARRANTIES_FLG	0.071631
9	PRICECLUB_STATUS	0.069498
10	CARD_PAYMENT	0.067164
11	REGISTRATION_DATE	0.065092
12	PURCHASE_COUNT	0.044525
13	BUYING_DATE	0.025035
14	SELLER_DEPARTMENT	0.024491

FIGURE 3 – Sortie de la méthode *get_highest_corr_target*

1.2.4 Corrélation du Train dataset

Cette méthode donne les paires de *features* les plus corrélées du Train dataset.

```
# Show the most corraleted pairs feature

BE.get_highest_corr(Train,
                    ID = 'ID',
                    value = 0,
                    n_pairs = 8,
                    n = 500)
```

— Train correspond au dataset d'entrainement
— ID est le nom de la variable correspondant à la clé primaire de chaque observation
— *value* est la valeur avec laquelle on souhaite remplir les valeurs manquantes
— *n_pairs* est le nombre de paires de *features* à exposer
— *n* est la taille d'échantillon à utiliser pour l'analyse

	feature_1	feature_2	correlation_abs
0	WARRANTIES_FLG	WARRANTIES_PRICE	0.982630
1	WALLET_PAYMENT	CARD_PAYMENT	0.655717
2	RSP_PAYMENT	PRICECLUB_STATUS	0.540061
3	REGISTRATION_DATE	PURCHASE_COUNT	0.442369
4	SHIPPING_PRICE	SELLER_DEPARTMENT	0.374038
5	PRICECLUB_STATUS	PURCHASE_COUNT	0.360268
6	SELLER_SCORE_COUNT	SELLER_SCORE_AVERAGE	0.359478
7	PURCHASE_COUNT	RSP_PAYMENT	0.300264

FIGURE 4 – Sortie de la méthode *get_highest_corr*

1.2.5 Matrice de corrélations colorée

Cette méthode donne la matrice de corrélation du Train et Target dataset.

```
# Show the correlation matrix colored

BE.corr_mat(Train,
            Target,
            ID = 'ID',
            value = 0,
            figsize = (20, 15),
            n = 1000,
            n_pairs = 8)
```

— Train correspond au dataset d'entrainement
— ID est le nom de la variable correspondant à la clé primaire de chaque observation
— *value* est la valeur avec laquelle on souhaite remplir les valeurs manquantes
— *figsize* est la taille de la figure
— *n_pairs* est le nombre de paires de *features* à exposer
— *n* est la taille d'échantillon à utiliser pour l'analyse

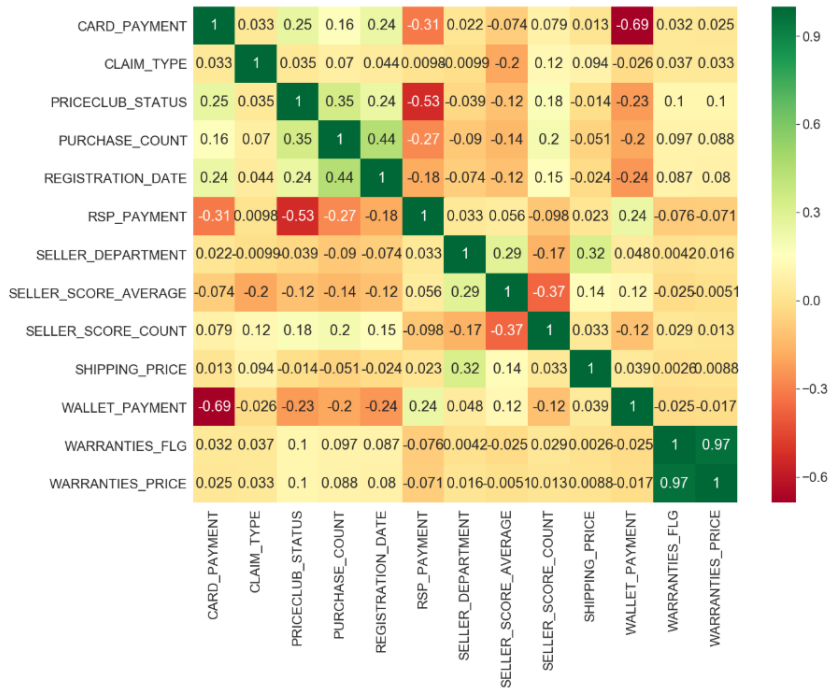

FIGURE 5 – Sortie de la méthode *corr_mat*

1.3 Fit

Cette méthode lance une série d'algorithmes de Machine Learning pour prédire la Target. Ces algorithmes proviennent de Scikit-Learn et permettent de choisir un algorithme optimal suivant les paramètres par défaut de Scikit-Learn.
Ensuite, le meilleur algorithme est optimisé par un Grid Search CV.

```
FE = FeatureEngineering(Train)          # Dataset
```

- Train correspond au dataset d'entrainement
- Target est la variable à prédire
- ID est le nom de la variable correspondant à la clé primaire de chaque observation
- *target_ID*, True si la Target possède la clé primaire ID
- *n* est la taille d'échantillon à utiliser pour l'analyse
- *n_grid* est la taille de l'échantillon pour le Grid Search CV
- *view_nan*, si True, donne des statistiques sur les valeurs manquantes
- *value* est la valeur avec laquelle on souhaite remplir les valeurs manquantes
- *scoring* est la metric (parmi celles de Scikit-Learn) à utiliser pour le calcul de la performance
- cv correspond au nombre de *folds* pour le premier check
- *grid*, si True, lance un Grid Search CV avec quelques hyperparamètres embarqués
- *hard_grid*, si True, lance un Grid Search CV avec plus d'hyperparamètres que la version précédente
- *cv_grid* est le nombre de *folds* pour le Grid Search CV

```
Missing Values :

                     Total      %
WARRANTIES_PRICE     96603  96.603
SHIPPING_PRICE       67610  67.610
BUYER_BIRTHDAY_DATE   5836   5.836
SHIPPING_MODE          315   0.315
PRICECLUB_STATUS        57   0.057
SELLER_SCORE_AVERAGE     6   0.006
SELLER_SCORE_COUNT       6   0.006

Missing values filled by 0

Searching for the best Classifier on 10000 datas using accuracy loss...


 Bagging: 0.497301 (+/- 1.542e-06)

 Gradient Boosting: 0.531000 (+/- 1.417e-06)

 XGBoost: 0.533299 (+/- 1.076e-05)

 Random Forest: 0.531702 (+/- 1.047e-05)

 Decision Tree: 0.377100 (+/- 5.191e-05)

 Extra Tree: 0.517400 (+/- 5.680e-07)

 KNN: 0.468103 (+/- 4.768e-05)

 SVM: 0.512300 (+/- 1.099e-06)

 Searching for the best hyperparametres of XGBoost using hard_grid on 10000 data among :

{'eta': [0.001, 0.01, 0.1, 0.3, 1], 'max_depth': [5, 10, 15, 20, 25], 'gamma': [0, 0.1, 0.01, 0.001]}

Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  8.4min
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed: 36.8min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 57.0min finished


 In the end, the best estimator is : XGBoost Classifier

 Using these hyperparametres : {'eta': 0.001, 'gamma': 0.01, 'max_depth': 5}

 With this accuracy score : 0.5349
```

FIGURE 6 – Sortie de la méthode *fit*

1.3.1 Grille d'hyperparamètres customisée

Cette méthode permet d'utiliser un dictionnaire d'hyperparamètres customisé.

```
# We can use a custom grid for the GridSearchCV
# with this method (using a Dict)

BE.custom_grid(Train, Target,       # Train and Target dataset
               ID='ID',             # ID column name
               target_ID=True,      # True if the Target have an ID column
               n=1000,              # Sample size for GridSearchCV
               metric='accuracy',   # metric to optimize
```

```
params=None,          # Dict of hyperparametres
cv=3,                 # fold number for cross validation
DF=None,              # estimator used
value=0)              # value for filling NaN
```

— Train correspond au dataset d'entrainement
— Target est la variable à prédire
— ID est le nom de la variable correspondant à la clé primaire de chaque observation
— *target_ID*, True si la Target possède la clé primaire ID
— n est la taille d'échantillon à utiliser pour l'analyse
— *metric* est la metric (parmi celles de Scikit-Learn) à utiliser pour le calcul de la performance
— *params* est le dictionnaire de paramètres utilisé
— cv correspond au nombre de *folds*
— *DF* est l'estimateur à utiliser parmi ceux proposer par Scikit-Learn
— *value* est la valeur avec laquelle on souhaite remplir les valeurs manquantes

1.3.2 Méthode bagging

Cette méthode permet d'utiliser un *bagging* sur le meilleur estimateur trouver dans la méthode *fit*.

```
# When an algorithm is optimized with the fit method,
# we can improve the prediction using an ensembling method called Bagging
# then, the estimator is saved for later prediction

BE.Bagg_fit(Train, Target,                  # Train and Target dataset
            ID='ID',                        # ID column name
            n=1000,                         # Sample size for GridSearchCV
            metric='accuracy',              # metric to optimize
            cv=3,                           # fold number for cross validation
            n_estimators=[5,15,30])         # set of estimators to use
```

- Train correspond au dataset d'entrainement
- Target est la variable à prédire
- ID est le nom de la variable correspondant à la clé primaire de chaque observation
- n est la taille d'échantillon à utiliser pour l'analyse
- *metric* est la metric (parmi celles de Scikit-Learn) à utiliser pour le calcul de la performance
- cv correspond au nombre de *folds*
- *n_estimators* est la liste du nombre d'estimateurs à tester

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   9 out of   9 | elapsed: 30.2min finished

Best hyperparametres : {'n_estimators': 5}

Giving this accuracy score : 0.52
```

FIGURE 7 – Sortie de la méthode *Bagg_fit*

1.3.3 Méthode best size

Cette méthode permet de tester plusieurs taille d'échantillon pour le *fit* et voir à partir de quelle taille l'*overfitting* apparait.

```
# This method use the the GridSearchCV estimator
# from the fit method to generated a prediction

BE.pred_grid(Test[0:30],      # Dataset to predict
             ID='ID',         # ID column name
             value=0)         # value to fill NaN
```

— n correspond à une liste de taille d'échantillon
— *metric* est la fonction d'erreur utilisée dans l'évaluation de la performance.

```
Fitting 100000 datas...
100000 datas -> mean_absolute_error = 15.285074820353667

Fitting 300000 datas...
300000 datas -> mean_absolute_error = 14.940633226703698

Fitting 600000 datas...
600000 datas -> mean_absolute_error = 14.981778906369133


 In the end, the best data size is 300000

 With this mean_absolute_error : 14.940633226703698
```

FIGURE 8 – Sortie de la méthode *best_size*

Attention, la sortie précédente ne correspond pas à l'appel donnée ci-dessus.

1.4 Prédiction

1.4.1 Prédiction à partir de la méthode fit

Cette méthode permet de prédire une Target à partir d'un jeu de donnée Test en utilisant l'estimateur résultant de la méthode fit.

```
# This method use the the GridSearchCV estimator
# from the fit method to generated
# a probability prediction of each label

BE.pred_grid_proba(Test[0:30],   # Dataset to predict
                   ID = 'ID',    # ID column name
                   value = 0)    # value for filling NaN
```

— Test est le dataset à prédire
— ID est le nom de la clé primaire du Test
— *value* servira à remplir les valeurs manquantes du Test

	ID	Target
0	100000	-
1	100001	-
2	100002	SELLER_CANCEL_POSTERIORI
3	100003	-
4	100004	NOT_RECEIVED
5	100005	-
6	100006	-
7	100007	SELLER_CANCEL_POSTERIORI
8	100008	-
9	100009	-

FIGURE 9 – Sortie de la méthode *pred_grid*

1.4.2 Prédiction des probabilités à partir de la méthode fit

Comme la précédente méthode, celle-ci donne la prédiction des probabilités des *labels* à partir de l'estimateur de la méthode *fit*.
Attention, ne fonctionne uniquement pour la classification.

```
# This method fit an estimator using the best hyperparametres
# of the algorithm found in the fit method
# then, a prediction is generated from this estimator


BE.pred(Test[0:30],        # dataset to predict
        ID='ID',           # ID column name
        value=0,           # value for filling NaN
        n=1000,            # size sample to fit if refit = True
        refit = True)      # if True, refit the Train and Target
```

— Test est le dataset à prédire
— ID est le nom de la clé primaire du Test
— *value* servira à remplir les valeurs manquantes du Test

	ID	-	DAMAGED	DIFFERENT	NOT_RECEIVED	SELLER_CANCEL_POSTERIORI	UNDEFINED	WITHDRAWAL
0	100000	0.426388	0.061266	0.027067	0.181528	0.160734	0.056410	0.086607
1	100001	0.391425	0.015610	0.056432	0.154746	0.188582	0.037072	0.156132
2	100002	0.303740	0.070769	0.030000	0.121350	0.327808	0.095156	0.051176
3	100003	0.655295	0.032618	0.018257	0.103509	0.113336	0.044204	0.032780
4	100004	0.223868	0.070068	0.122983	0.293512	0.114028	0.070699	0.104843
5	100005	0.408444	0.035000	0.005333	0.139382	0.146032	0.193031	0.072778
6	100006	0.386869	0.060667	0.086929	0.113024	0.214455	0.076667	0.061389
7	100007	0.289872	0.062308	0.021667	0.152747	0.310000	0.096484	0.066923
8	100008	0.501426	0.101667	0.023056	0.128053	0.149308	0.019649	0.076843
9	100009	0.422931	0.068028	0.042833	0.137049	0.170095	0.067519	0.091544

FIGURE 10 – Sortie de la méthode *pred_grid_prob*

1.4.3 Prédiction à partir d'un nouvelle estimateur

Cette méthode permet de réentrainer un estimateur *from scratch* en réutilisant les informations de la méthode *fit* et de faire une prédiction ensuite.

```
# This method fit an estimator using the best hyperparametres
# of the algorithm found in the fit method
# then, a prediction is generated from this estimator


BE.pred(Test[0:30],        # dataset to predict
        ID='ID',           # ID column name
        value=0,           # value for filling NaN
        n=1000,            # size sample to fit if refit = True
        refit = True)      # if True, refit the Train and Target
```

- Test est le dataset à prédire
- ID est le nom de la clé primaire du Test
- *value* servira à remplir les valeurs manquantes du Test
- n est la taille d'échantillon à utiliser pour le *fit*
- *refit* si True, refit à chaque appel de cette méthode (si l'estimateur n'est pas aléatoire, ne sert à rien)

	ID	Target
0	100000	-
1	100001	-
2	100002	SELLER_CANCEL_POSTERIORI
3	100003	-
4	100004	-
5	100005	SELLER_CANCEL_POSTERIORI
6	100006	-
7	100007	NOT_RECEIVED
8	100008	-
9	100009	-

FIGURE 11 – Sortie de la méthode *pred*

1.4.4 Prédiction des probabilités à partir d'un nouvelle estimateur

Cette méthode permet de réentrainer un estimateur *from scratch* en réutilisant les informations de la méthode *fit* et de faire une prédiction des probabilités de chaque *label* ensuite.

```
# This method fit an estimator using the best hyperparametres
# of the algorithm found in the fit method
# then, a probability of each label is predict


BE.pred_proba(Test[0:30],       # dataset to predict
              ID='ID',          # ID column name
              value=0,          # value for filling NaN
              n=1000,           # size sample to fit if refit = True
              refit = True)     # if True, refit the Train and Target
```

- Test est le dataset à prédire
- ID est le nom de la clé primaire du Test
- *value* servira à remplir les valeurs manquantes du Test
- n est la taille d'échantillon à utiliser pour le *fit*
- *refit* si True, refit à chaque appel de cette méthode (si l'estimateur n'est pas aléatoire, ne sert à rien)

	ID	-	DAMAGED	DIFFERENT	FAKE	NOT_RECEIVED	SELLER_CANCEL_POSTERIORI	UNDEFINED	WITHDRAWAL
0	100000	0.468157	0.051475	0.034024	0.002021	0.128015	0.189042	0.058905	0.068362
1	100001	0.441692	0.057201	0.053548	0.003901	0.098317	0.193801	0.058603	0.092936
2	100002	0.193064	0.026272	0.063224	0.006667	0.160381	0.416967	0.040262	0.093163
3	100003	0.444557	0.061163	0.033258	0.001667	0.134842	0.208536	0.043392	0.072586
4	100004	0.278700	0.108784	0.060705	0.035127	0.194992	0.108385	0.088250	0.125056
5	100005	0.262841	0.021584	0.037911	0.003333	0.123834	0.311878	0.055490	0.183128
6	100006	0.437169	0.072358	0.044418	0.001667	0.119462	0.166316	0.058569	0.100042
7	100007	0.279811	0.076933	0.066468	0.005000	0.239921	0.183531	0.047720	0.100616
8	100008	0.383340	0.081723	0.049003	0.010000	0.133582	0.186715	0.049264	0.106373
9	100009	0.347293	0.058823	0.052727	0.018333	0.215458	0.148061	0.037527	0.121777

FIGURE 12 – Sortie de la méthode *pred_proba*

2 Class Feature Engineering

Cette *class* permet de voir quelques statistiques sur les données et de faire certains types de *Feature Engineering*.

2.1 Call

Cette *class* s'appelle de la façon suivante :

```
FE.Missing_Values()  # Show some statistics about missing values
```

Où le Train est le dataset d'entrainement.

2.2 Statistiques et DataViz

2.2.1 Valeurs manquantes

Cette méthode permet d'avoir un aperçu des valeurs manquantes du Train.

```
FE.Unique() # Show all categorical features
            # with their labels of the dataset used
```

```
Missing Values :

                      Total      %
WARRANTIES_PRICE      96603  96.603
SHIPPING_PRICE        67610  67.610
BUYER_BIRTHDAY_DATE    5836   5.836
SHIPPING_MODE           315   0.315
PRICECLUB_STATUS         57   0.057
SELLER_SCORE_AVERAGE      6   0.006
SELLER_SCORE_COUNT        6   0.006
```

FIGURE 13 – Sortie de la méthode *Missing_ Values*

2.2.2 Valeurs uniques

Cette méthode donne les colonnes catégorielles et leurs valeurs uniques.

```
FE.Plot(feature1 = 'REGISTRATION_DATE',
        feature2 = 'SELLER_DEPARTMENT', # features to plot
        Data_base = True,  # if True, use the original dataset else,
                           # use the transform dataset from methods above
        figsize = (15,10), # set the figure size
        n = 1000)          # sample size to use for plotting
```

```
SHIPPING_MODE :

['NORMAL' 'RECOMMANDE' 'EXPRESS_DELIVERY' 'SUIVI' 'SO_RECOMMANDE'
'MONDIAL_RELAY' 'MONDIAL_RELAY_PREPAYE' 'SO_POINT_RELAIS' nan
'CHRONOPOST' 'PICKUP' 'Kiala']


SHIPPING_PRICE :

[nan '5<10' '10<20' '<1' '1<5' '>20']


WARRANTIES_FLG :

[False  True]


WARRANTIES_PRICE :

[nan '5<20' '<5' '20<50' '50<100' '100<500']


PRICECLUB_STATUS :

['UNSUBSCRIBED' 'PLATINUM' 'SILVER' 'REGULAR' 'GOLD' nan]
```

FIGURE 14 – Sortie de la méthode *Unique*

2.2.3 Scatter Plot

Cette méthode permet créer un *scatter plot* d'une *feature* en fonction d'une autre.

```
FE.feature_dist(feature = 'BUYER_BIRTHDAY_DATE', # Feature to analyse
                Data_base = True,                # if True, use the original datase
                                                 # else, use the transform dataset f
                figsize = (17,12),               # set the figure size
                n = 300                          # sample size to use for the linep
                bins = 100)                      # set the bins for histogram
```

- *feature1* : première *feature*
- *feature2* : deuxième *feature*
- *Data_base* si True, le *plot* est fait sur le Train donné à l'appel de la *class*, sinon, le *plot* est fait sur le Train modifié par les méthodes de *Feature Engineering* expliqué dans le paragraphe suivant
- *figsize* permet de donner la taille de la figure
- n est la taille de l'échantillon utilisé dans le *plot*

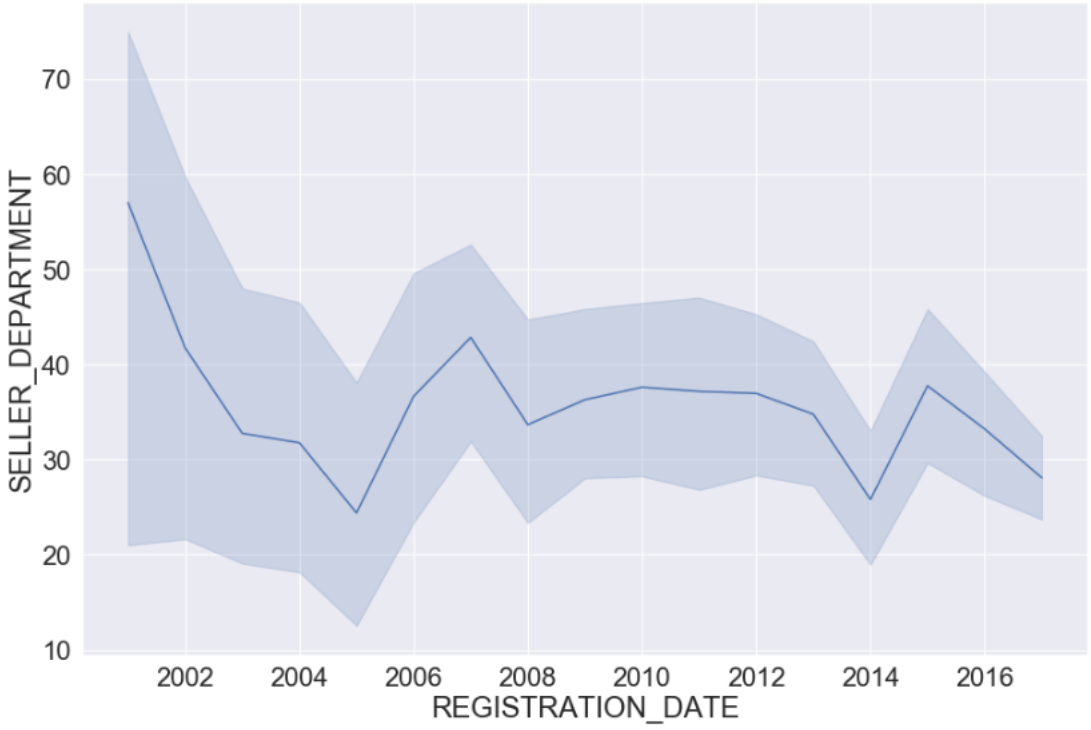

FIGURE 15 – Sortie de la méthode *Plot*

2.2.4 Distribution et lineplot

Cette méthode permet de voir la distribution d'une *feature* ainsi que son *lineplot*.

```
FE.To_numeric_freq(columns = 'all')  # Transform a categorical column into
                                     # columns = 'all' or ['column1','colum
```

— *feature* : *feature* à étudier
— *Data_base* si True, le *plot* est fait sur le Train donné à l'appel de la *class*, sinon, le *plot* est fait sur le Train modifié par les méthodes de *Feature Engineering* expliqué dans le paragraphe suivant
— *figsize* permet de donner la taille de la figure
— n est la taille de l'échantillon utilisé dans le *plot*
— *bins*, permet d'ajuster la taille des barres dans l'histogramme

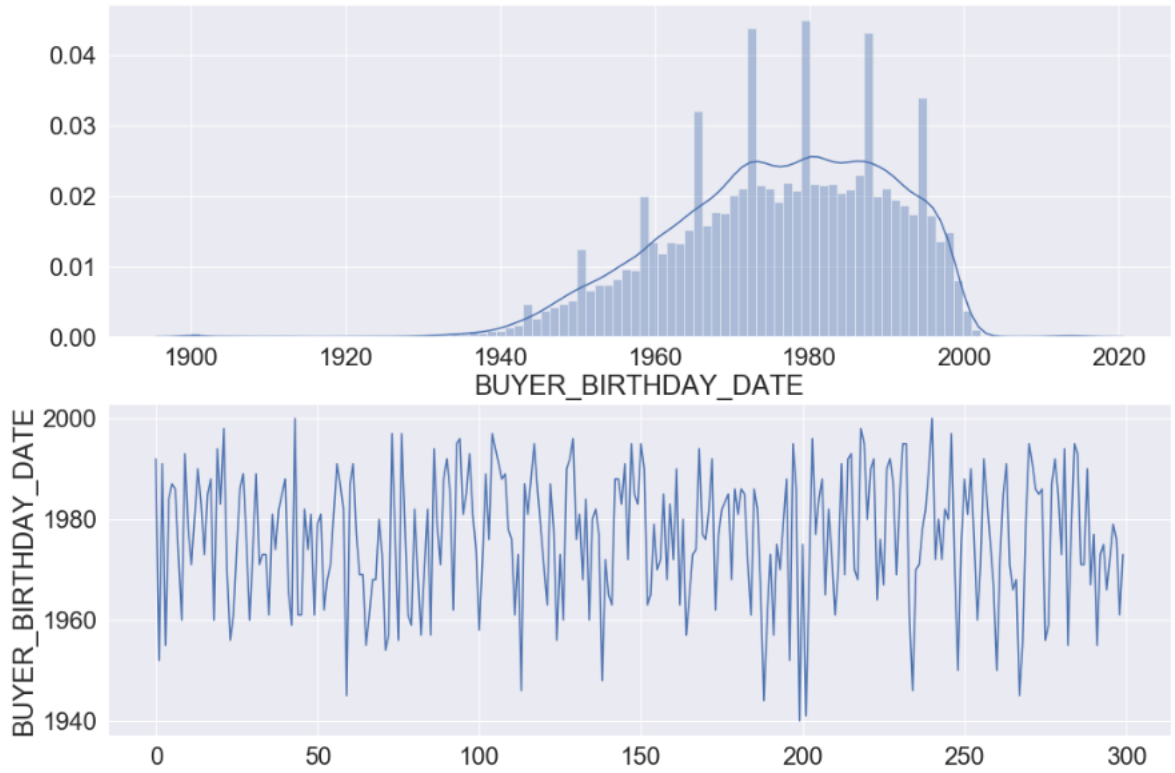

FIGURE 16 – Sortie de la méthode *feature_dist*

2.3 Feature Engineering

2.3.1 Transformation en fréquence

Cette méthode permet de transformer une variable catégorielle en une variable dont chaque *label* est remplacé par sa fréquence d'apparition.

```
FE.To_numeric_quant(columns = 'all')

# Transform a categorical column into item count
# columns = 'all' or ['column1','column2',...]
```

L'argument *columns* prend comme valeur soit *"all"* si toutes les colonnes catégorielles sont à convertir, soit une liste contenant le nom des *features* à convertir.

Attention ! : si des *labels* apparaissent en proportion égale, les fréquences dans la variable transformée seront identiques.

2.3.2 Transformation en nombre d'apparition

Cette méthode permet de transformer une variable catégorielle en une variable dont chaque *label* est remplacé par son nombre d'apparition contenant.

```
FE.To_numeric_quant(columns = 'all')

# Transform a categorical column into item count
# columns = 'all' or ['column1','column2',...]
```

L'argument *columns* prend comme valeur soit *"all"* si toutes les colonnes catégorielles sont à convertir, soit une liste contenant le nom des *features* à convertir.

Attention ! : si des *labels* apparaissent en proportion égale, les fréquences dans la variable transformée seront identiques.

2.3.3 Transformation customiser

Cette méthode permet de transformer une variable catégorielle en une variable numérique de façon customiser à partir d'un dictionnaire Python.

```
FE.OneHotEncoder(columns)

# One hot encode a feature
# columns = ['column1', 'column2',...]
```

2.3.4 OneHotEncoder

Cette méthode permet de d'appliquer un *OneHotEncoder* à une variable catégorielle. C'est-à-dire qu'à partir d'une variable catégorielle contenant n *labels*, n nouvelles variables seront crées portant chacune le nom d'un *label*, si le *label* A apparait dans la variable de départ un 1 sera affecté à la nouvelle variable nommée A et 0 sinon.

```
FE.OneHotEncoder(columns)

# One hot encode a feature
# columns = ['column1', 'column2',...]
```

3 Collaboration

Ce *framework* est disponible sur GitHub via le lien suivant :

`https://github.com/jeremycombe/Pipeline`

Si vous avez des remarques ou des bugs merci de m'en faire part en m'envoyant un email à l'adresse suivante :

jeremy.combe@etu.upmc.fr