UNIVERSITÉ DE NANTES
UFR DE SCIENCES ET TECHNIQUES

Master 2

Spécialité : **Ingénierie Statistique**

Rédigé par :

Loubna N'HAR
Soufian ECHABARRI

DATA CHALLENGE 2021

Stock trading : prediction of auction volumes

Année universitaire : 2021/2022

Table des matières

1.1 Introduction 2

1.2 Exploration et préparation des données 3

1.2.1 Visualisation des données 3

1.2.2 Pré-traitement des données 4

1.3 Résolution du problème 5

1.3.1 Features Engineering 5

1.3.2 Features selection 6

1.3.3 Découper en Train et Validation selon la date 7

1.3.4 Modélisation avec LightGBM (meilleur modèle) 8

1.3.5 Modélisation avec XGBoost 9

1.3.6 Modélisation avec Regression Lasso (très faible) 10

1.3.7 Le choix du modèle 11

1.4 Conlusion 11

Bibliographie 12

Annexe 13

1.1 Introduction

Le challenge sujet de notre projet est un challenge publié par CFM, nommé **Stock trading : prediction of auction volumes** et consiste à prédire le volume (valeur totale des actions échangées) disponible aux enchères, pour 900 actions sur environ 350 jours.
Ce type de problématique est un peu difficile à traiter au vu du caractère presque aléatoire des données. Pour y arriver, il faut passer par une étape de préparation de données, puis une transformation de celles-ci grâce au feature engineering et features selection avant de procéder à l'optimisation d'un modèle de Machine Learning. C'est sur ces dernières étapes que nous intervenons et qui sont détaillées dans ce rapport.
Premièrement, nous commencerons par présenter le jeu de données que nous aurons à notre disposition, les enjeux qui y sont associés, ainsi que les modifications et transformations qui y seront effectuées. Ensuite, nous traitons la partie de features engineering (une peu manuel) et features selection, en se basant sur plusieurs méthodes. Enfin, nous passons à la modélisation dans laquelle nous mettons en évidence les méthodes utilisés et les résultats obtenues.

Le but de ce challenge

L'objectif principal de ce projet est d'estimer le volume des enchères pour un stock et jour donnés. Les données passées peuvent être utilisées pour estimer les données futures. Pour cela, on doit faire attention au découpage des données en train et validation (le découpage sera basé sur la date). C'est un problème de régression de l'apprentissage supervisé.

Les outils utilisés

Afin de travailler plus efficacement, de gagner un peu d'espace et de temps, nous avons utilisé les outils suivants :

- Google Collab, Anaconda, Jupyter
- Frameworks : XGBoost, LightGBM (très rapide), scikit-learn, pandas,...
- Ordinateur windows 10, 8 GO de RAM.

Description des données

Nous disposons de 3 jeux de données au format csv. Les données d'entrée d'apprentissage et les données de sortie d'apprentissage. Ces données participent à l'entraînement et et à la validation du modèle, ils représentent respectivement les variables explicatives et la variable réponse. Ces échantillons sont composés de 684482 lignes.
Le $3^{ème}$ jeu de données est l'échantillon test, celui-ci n'intervient à aucun moment dans la construction du modèle et permet uniquement d'évaluer les performances du modèle retenu.

Il est composé de 311744 lignes.
On remarque donc, un découpage d'environ 70% - 30% de nos données initiales.

Données d'entrée fournies

- id : chaque point de données a un identifiant, unique pour un stock, un jour donné
- pid : l'ID produit d'un stock
- day : jour de l'échantillon
- abs_retn (n = 0 à 60) : valeurs absolues des rendements sur la journée, séparées en 61 périodes
- rel_voln (n = 0 à 60) : fraction du volume échangé sur la journée, séparée en 61 périodes (total à 1 chaque jour)
- LS et NLV : variables mystérieuses dont la finalité n'est pas divulguée dans le challenge.

Données de sortie fournie

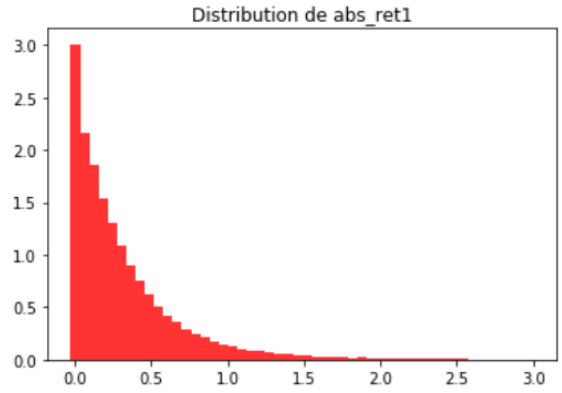
- id : correspond à l'identifiant dans les données d'entrée
- target : log naturel du volume d'enchères comme volume total dans les 61 périodes
- Si le volume d'enchères est de 10% du volume du jour, target $= log(0.10) = -2.30$

Les 900 valeurs dans les données d'entraînement et de test sont les mêmes : il est donc en principe possible d'élaborer des prédictions personnalisées pour chaque valeur. Les données d'entraînement contiennent des informations sur environ 800 jours différents, tandis que les données de test nécessitent des prévisions de volume d'enchères sur environ 350 jours. De plus, les entrées de test correspondent aux jours qui suivent ceux des données d'entraînement.

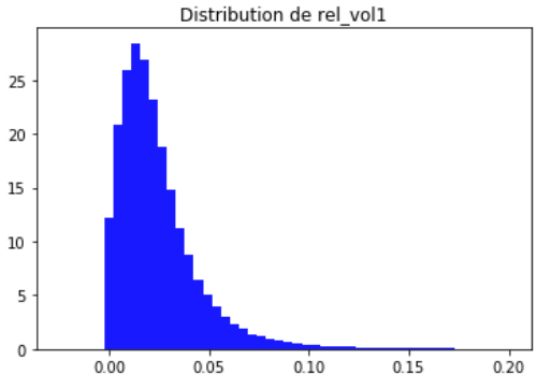
1.2 Exploration et préparation des données

1.2.1 Visualisation des données

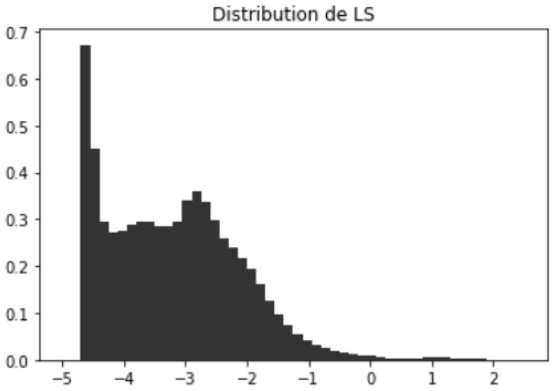
Pour avoir une idée sur les variables explicatives et pour mieux comprendre comment elles sont distribuées, nous présentons dans la figure (1.1), les distributions des variables d'entrée suivantes : abs_ret1, rel_vol1, LS et NLV.



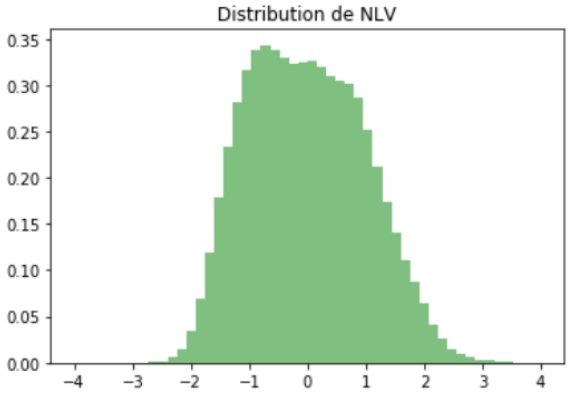
(a) Distribution de la variable abs_ret1



(b) Distribution de la variable rel_vol1



(c) Distribution de la variable LS



(d) Distribution de la variable abs_ret1

FIGURE 1.1 – Les distributions de certaines variables des données d'entrée

1.2.2 Pré-traitement des données

Nous commençons par vérifier que tous les stocks des données test sont dans les données train.

```
for i in range(0,len(input_test["pid"])):
    if input_test["pid"][i] not in Input_train["pid"]:
        print(input_test["pid"][i])
```

Nous calculons d'abord le nombre et le pourcentage de valeurs manquantes dans l'échantillon train et test. D'après la figure (1.2), parmi 126 colonnes, 122 colonnes ont des valeurs manquantes avec un pourcentage moyen de 4%, et ceci pour les deux échantillons.

	Missing Values	% of Total Values
abs_ret45	42857	6.3
rel_vol45	42857	6.3
abs_ret44	41499	6.1
rel_vol44	41499	6.1
abs_ret47	40879	6.0
...	...	...
rel_vol12	21343	3.1
rel_vol6	19803	2.9
abs_ret6	19803	2.9
abs_ret0	4996	0.7
rel_vol0	4996	0.7

(a) Le nombre et le pourcentage des valeurs manquantes dans les données train

	Missing Values	% of Total Values
rel_vol2	17004	5.5
abs_ret2	17004	5.5
abs_ret47	15490	5.0
rel_vol47	15490	5.0
rel_vol42	15053	4.8
...	...	...
rel_vol7	8045	2.6
abs_ret14	6967	2.2
rel_vol14	6967	2.2
abs_ret0	766	0.2
rel_vol0	766	0.2

(b) Le nombre et le pourcentage des valeurs manquantes dans les données test

FIGURE 1.2 – Les valeurs manquantes dans les données train et test

Nous remplaçons ces valeurs manquantes par la moyenne, car il y avait des valeurs qui sont un peu loin de 0, surtout les rendement, donc le fait de remplacer les valeurs manquantes par des zéros ne semble pas raisonnable dans notre cas.

Une dernière étape dans cette partie, consiste à transformer les **pid** en numériques et ceci pour un bon traitement par LGBM et XGBoost, autrement dit, les pid seront représentés par des catégories de 0 à 899. Notons que nous avons les mêmes stocks en train et en test. Ainsi, nous pouvons utiliser les mêmes stocks en train et en validation.

1.3 Résolution du problème

1.3.1 Features Engineering

Lors de l'entraînement d'un modèle algorithmique, le feature engineering est une étape complexe. Elle joue un rôle important dans l'organisation des jeux de données pour les différentes techniques de machine learning. C'est un processus qui consiste à rendre les données exploitables pour le modèle. Il est essentiel pour garantir que nos modèles de Machine Learning fonctionnent correctement. Ainsi, l'analyse et la visualisation des données nous amène à ajouter quelques features élémentaires qui peuvent nous aider à mieux décrire la variable target, à savoir : minimum, maximum, écart type, somme totale, moyenne, skewness,

kurtosis. Ces nouvelles variables seront calculées sur les 61 périodes et sur chaque observation i, et ceci pour les rendements (abs_ret) et les volumes (rel_vol).

1.3.2 Features selection

L'extraction des features est une phase cruciale pour l'implantation du modèle. Pour cela, nous allons mettre en évidence les variables les plus significatifs en se basant sur trois méthodes, à savoir : XGBoost, LGBM et Régression Lasso. Notons que dans cette partie, il ne s'agit pas de modélisation mais juste une idée sur les variables les plus significatives.

Extreme Gradient Boosting

XGBoost est une bibliothèque open source fournissant une implémentation haute performance d'arbres de décision renforcés par Gradient Boost. Les performances de XGBoost ne sont pas une blague, elle est devenue la bibliothèque de référence pour gagner de nombreux concours Kaggle. Elle utilise des arbres de décision comme apprenants de base, combiner de nombreux apprenants faibles pour faire un apprenant fort. En conséquence, il est appelé méthode d'apprentissage d'ensemble car il utilise la sortie de nombreux modèles dans la prédiction finale. Parmi ses avantages, on peut citer : sa puissance du traitement parallèle (c'est plus rapide que le Gradient Boosting), Il prend en charge la régularisation et l'utilisateur peut exécuter une validation croisée après chaque itération.

LightGBM

LightGBM est un framework de gradient boosting qui utilise un algorithme d'apprentissage basé sur les arbres de décision. Elle utilise un algorithme basé sur un histogramme, c'est-à-dire qu'il regroupe les valeurs de caractéristiques continues dans des bacs discrets qui accélèrent la procédure d'apprentissage.

Régression Lasso

La régression Lasso optimise le problème qui suit :

$$\sum_{i=0}^{n}(Y_i - X_i\beta)^2 + ||\beta||_1$$

C'est une régression linéaire avec une contrainte linéaire sur les coefficients. C'est utile lorsque les variables sont très corrélées, ce qui fausse souvent la résolution numérique. La résolution utilise une méthode à base de gradient.

Nous représentons la qualité des variables sélectionnées selon les trois méthodes dans la figure (1.3). Pour plus de figures sur la qualités de sélection, veuillez voir l'annexe (1.7, 1.6).

	cols	fea_imp
128	sum_ret	0.113257
125	NLV	0.075170
63	rel_vol0	0.039473
130	std_ret	0.032122
137	std_vol	0.029535
139	kurt_vol	0.029462
1	day	0.028878
124	LS	0.020914
0	pid	0.019775
64	rel_vol1	0.015269
123	rel_vol60	0.014979
133	min_vol	0.014479
135	sum_vol	0.014063

(a) Features importance avec XGboost

	cols	fea_imp
0	pid	7698
1	day	6822
124	LS	3732
125	NLV	3489
63	rel_vol0	1723
128	sum_ret	1289
133	min_vol	1280
137	std_vol	1141
130	std_ret	1074
64	rel_vol1	969
86	rel_vol23	763
135	sum_vol	740
123	rel_vol60	734

(b) Features importance avec LightGBM

	cols	fea_imp
128	sum_ret	0.134557
125	NLV	0.130061
1	day	0.071455
63	rel_vol0	0.029656

(c) Features importance avec Regression Lasso

FIGURE 1.3 – La qualités des variables sélectionnées avec XGBoost, LightGBM et Regression Lasso.

En se basant sur les corrélations (1.5), la qualité des features sélectionnées par les méthodes XGBoost, LightGBM et Regression Lasso, nous décidons de ne garder que les variables suivantes : pid, day, NLV, LS, rel_vol0, sum_ret, std_vol, min_vol, rel_vol1, std_ret, mean_ret, kurt_vol.

1.3.3 Découper en Train et Validation selon la date

Avant de découper les données en train et validation. Nous devons nous concentrer sur les points suivants :

— Chaque stock (pid) apparaît plusieurs fois

— Chaque date (day) apparaît plusieurs fois
— Chaque couple (pid,day) n'apparaît qu'une seule fois
— Les stocks sont les mêmes dans train et test (nous avons déjà vérifié ça).

Nous devons estimer les données futures en utilisant les données passées. Pour cela, il ne faut pas utiliser les mêmes jours pour training et validation. Donc, les jours doivent être différents entre train et validation. Par conséquent, nous découpons les données en train-validation à hauteur de 70-30% selon les jours. Notons que nous avons testé 4 découpages selon random state (7, 42, 210 et 666) et la division qui donne un meilleur score est celle qui correspond à random_state=42.

```
nday=Input.day.unique()
n_train,n_val=train_test_split(nday,random_state=42,train_size=0.7)
l_train=n_train.tolist()
l_val=n_val.tolist()
Input["target"]=output.target
XY_train=Input[Input.day.isin(n_train)]
XY_val=Input[Input.day.isin(n_val)]
X_train=XY_train.drop('target',axis=1)
Y_train=XY_train.target
X_val=XY_val.drop('target',axis=1)
Y_val=XY_val.target
```

1.3.4 Modélisation avec LightGBM (meilleur modèle)

Les performances de LightGBM dépend fortement du réglage des hyperparamètres. C'est une méthode d'ensemble utilisant une technique d'amplification pour combiner des arbres de décision. La complexité d'un arbre individuel est également un facteur déterminant du surajustement. Il peut être contrôlé avec les paramètres **max_depth** et **num_leaves**. Max_depth détermine la profondeur maximale d'un arbre tandis que num_leaves limite le nombre maximal de feuilles qu'un arbre peut avoir. Étant donné que LightGBM adapte la croissance des arbres au niveau des feuilles, il est important d'ajuster ces deux paramètres ensemble. Le paramètre **min_data_in_leaf** est un moyen de réduire le surajustement. Il faut que chaque feuille ait le nombre d'observations spécifié afin que le modèle ne devienne pas trop spécifique. Sans oublier bien sur d'estimer le paramètre **n_estimators** qui représente le nombre d'arbres boostés à adapter. Un autre paramètre important est le **learning_rate**. Les taux d'apprentissage plus faibles sont généralement meilleurs, mais cela entraîne un apprentissage plus lent du modèle.

Il existe de nombreux hyperparamètres. Certains sont plus importants en termes de

précision et de vitesse. Certains d'entre eux sont principalement utilisés pour éviter le surajustement.

Grâce à la fonction GridSearchCV, nous allons optimiser ces hyperparamètres.

```
import lightgbm as lgb
lgb_model=lgb.LGBMRegressor()
lgb_params = {"learning_rate": [0.01,0.05, 0.1],
              "n_estimators": [100,500, 1000,1500,2000],
               "max_depth": [5, 8, 10],
              "num_leaves":[32,64,128],
              "min_data_in_leaf":[100,1000]}
lgb_cv_model = GridSearchCV(lgb_model,
                            lgb_params,
                            cv=10,
                            n_jobs=-1,
                            verbose=2,scoring='neg_mean_squared_error').fit(X_train,Y_train,categorical_feature=['pid'])
lgb_cv_model.best_params_
```

```
{'learning_rate': 0.05,
 'max_depth': 8,
 'min_data_in_leaf': 100,
 'n_estimators': 1500,
 'num_leaves': 128}
```

Après l'optimisation des paramètres, nous passons à l'ajustement du modèle sur les données d'entraînement et nous représentons la qualité de ce modèle selon l'erreur quadratique moyenne, training score et validation score.

mean square error	training score	validation score
0.3985	0.6939	0.5104

1.3.5 Modélisation avec XGBoost

XGBoost et LighGBM suivent tous deux le principe du boosting de gradient. Il y a cependant la différence dans les détails de la modélisation. Plus précisément, XGBoost utilise une formalisation de modèle plus régularisée pour contrôler le sur-ajustement. Nous représentons dans le tableau suivant les participiales différences entre ces deux modèles.

	XGBoost	LightGBM
Construction des arbres	Par niveau ou par feuille	Par feuille
Algorithme de pondération des échantillons pour la segmentation sur les grands datasets	Aucun	Gradient-based One-Side Sampling (GOSS)
Variables catégorielles	Non-supportées (à traiter séparément)	À encoder indiquer lors de l'entrainement (segmentation intelligente)
Calcul de l'importance des variables	Gain moyen sur la fonction de coût suite au noeuds où la feature est utilisée	Nombre de noeuds où la feature est utilisée

Comme précédemment, nous commençons par l'estimation et le choix des hyperparamètres, en utilisant la foncion GridSearchCV avec un choix de score qui minimise l'erreur quadratique moyenne. Les paramètres sélectionnés pour ajuster ce modèle sont : {'learning_rate' : 0.05, 'max_depth' : 8, 'n_estimators' : 1500}.
Nous présentons les résultats de cette méthode dans le tableau suivant :

mean square error	training score	validation score
0.4790	0.7104	0.4145

1.3.6 Modélisation avec Regression Lasso (très faible)

Nous devons maintenant optimiser l'hyperparamètre alpha de la régression Lasso. Pour cela, nous allons tester les valeurs de 0,1 à 6 avec un pas de 0.2. Pour chaque valeur, nous calculons la valeur moyenne de l'erreur quadratique moyenne dans une validation croisée de 5 folds et sélectionnons la valeur de qui minimise ces mesures de performance moyennes. Nous utilisons aussi l'objet GridSearchCV à cette fin. Le meilleur choix de alpha est : alpha = 0.1.

Les performances et les résultats du modèle sont les suivants :

mean square error	training score	validation score
0.6839	0.1671	0.1598

Nous avons essayé de modéliser d'autres modèles, tels que Random Forest, Régression Lars,... Ces modèles ont donné des résultats différents, notamment en terme de l'erreur quadratique moyenne. Mais Random Forest a donnée un meilleur résultat en terme de training score, qui est de 0.9179, mais avec un erreur quadratique moyenne de 0.5793. Pour plus de détail, veuillez voir le jupyter notebook.

1.3.7 Le choix du modèle

D'après les résultats obtenus, nous avons essayé de comparer les performances de chaque méthode. **LightGBM** est la meilleure méthode pour prédire les volume d'enchères, notamment en terme de l'erreur quadratique moyenne. Nous avons obtenu des meilleurs résultats en cross-validation avec max_depth = 8. Mais, puisque nous avons plus de données à notre disposition, et pour eviter le risque de sur-ajustement, nous essayons d'ajouter un degré de liberté et nous prenons max_depth= 9.

Avant de passer à l'étape de prédiction, nous ajustons notre modèle sur tout l'ensemble des données train avec les mêmes paramètres que nous avons obtenu précédemment, sauf avec un choix de max_depth= 9.

1.4 Conlusion

A partir de ce projet, on a pu montrer comment Machine Learning peut être appliqué dans un domaine comme la finance. Le problème à traiter était un problème de régression. Nous avons commencé par une visualisation et analyse des données qui nous ont permis de bien comprendre le problème et élargir notre espace de solutions à proposer. Après le pré-traitement des données, nous avons pensé à ajouter quelques variables dans notre data et ceci dans le but d'améliorer la qualité de prédiction de variable target, avant de passer à l'étape d'extraction des fetaures. La dernière étape était la modélisation, dans laquelle nous avons essayé de trouver le meilleur méthode qui modélise les volumes d'enchères.

Nos résultats ne nous ont pas permis d'atteindre le haut du classement sur ce challenge, mais nous avons appris à exploiter des méthodes de traitements de données : l'extraction de variables, visualisation et analyse de données. Mais aussi des nouvelles méthodes de prédiction : XGBoost, LightGBM, Regression Lasso,... Et surtout à savoir sous quelles conditions utiliser ces dernières. Nous ne manquerons pas de les utiliser à nouveau lorsque l'occasion se présentera pendant la suite de nos études (au cours d'Apprentissage Statistque Avancée par exemple) ou de notre vie professionnelle. Ainsi, nous voulons dire que ce challenge était un meilleur moyen de mettre en pratique les méthodes de Machine Learning que nous avons vues dans notre cours d'Apprentissage Statistique, et nous sommes toujours motivé de commencer d'autres challenges pour approfondir nos connaissances dans ce domaine.

Pour les travaux futurs concernant ce challenge, nous aimerions que ces données soient réexaminées à l'aide de méthodes d'analyse telles que les séries chronologiques afin d'avoir une meilleure vue sur les données ainsi que peut-être de les utiliser pour mieux comprendre pourquoi les 11 variables que nous avons sélectionnées sont les meilleurs prédicteurs. Nous aimerions également approfondir l'analyse de ces données pour en extraire d'autres informations et les ajouter aux données en tant que nouvelles variables pouvant nous aider à mieux décrire la variable target.

Comme nous avons déjà dit, notre score du challenge n'était pas aussi proche de la référence que nous le souhaiterions, mais avec des recherches supplémentaires comme décrit ci-dessus, nous essayerons de l'améliorer après dans un futur travail.

Bibliographie

[1] Butch, Q. (2020). Next-Generation Machine Learning with Spark (Covers XGBoost, LightGBM, Spark NLP, Distributed Deep Learning with Keras, and More). DOI : 10.1007/978-1-4842-5669-5.

[2] Khalid, S. and al (2014). [IEEE Science and Information Conference (SAI) - London, UK (2014.8.27-2014.8.29)] Science and Information Conference - A survey of feature selection and feature extraction techniques in machine learning. DOI : 10.1109-sai.2014.6918213.

[3] Bastien L. - Machine Learning : Définition, fonctionnement et secteurs d'application http ://www.artificiel.net/machine-learning-definition

[4] XGBoost's Documentation. site web. https ://xgboost.readthedocs.io/en/stable

[5] LightGBM's documentation. Site web : https ://lightgbm.readthedocs.io/en/latest

[6] Scikit-learn's documentation. Site web. https ://scikit-learn.org/stable/index.html

[7] Feature Engineering : définition et importance en Machine Learning. Site web. https ://datascientest.com/feature-engineering

Annexe

Jetons un coup d'oeil sur les données d'entraînement

```
Input.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 684482 entries, 0 to 684481
Columns: 127 entries, ID to NLV
dtypes: float64(124), int64(3)
memory usage: 663.2 MB
```

```
output.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 684482 entries, 0 to 684481
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   ID      684482 non-null  int64
 1   target  684482 non-null  float64
dtypes: float64(1), int64(1)
memory usage: 10.4 MB
```

FIGURE 1.4 – Les données d'entraînement

Corrélations des variables explicatives

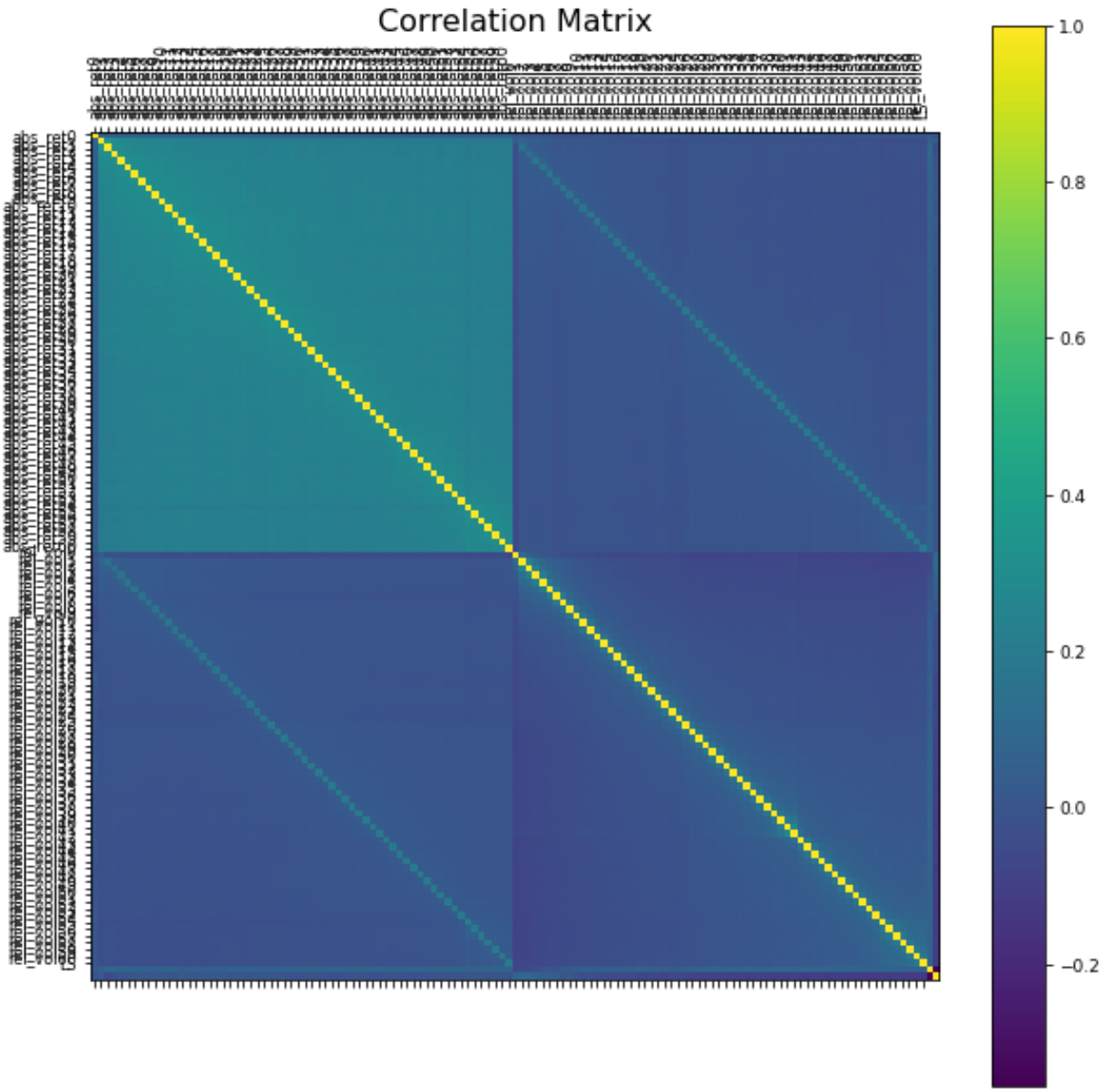


FIGURE 1.5 – Corrélations des variables explicatives

Fatures importance avec XGBoost

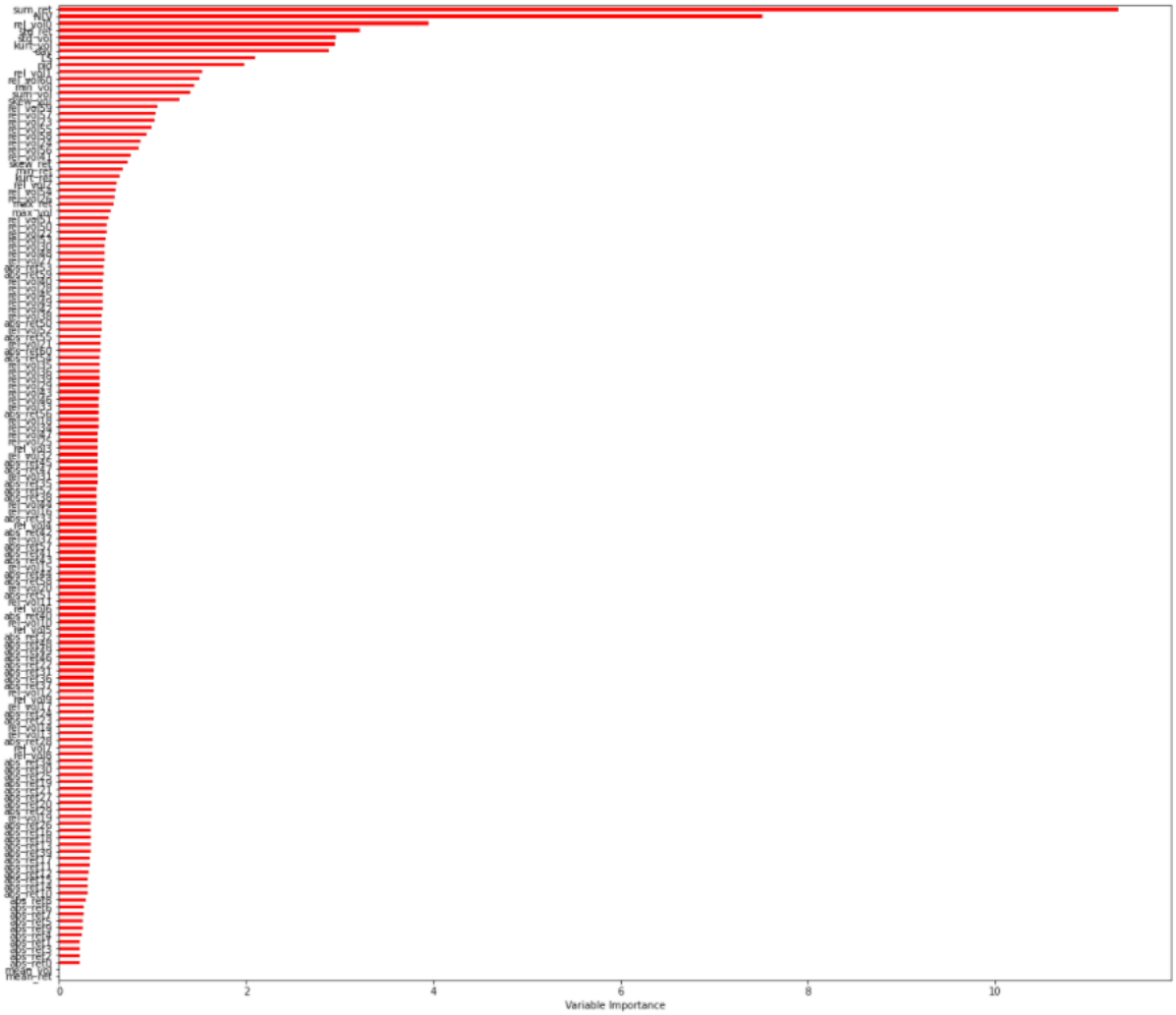


FIGURE 1.6 – Features importance selon XGBoost

Fatures importance avec LightGBM

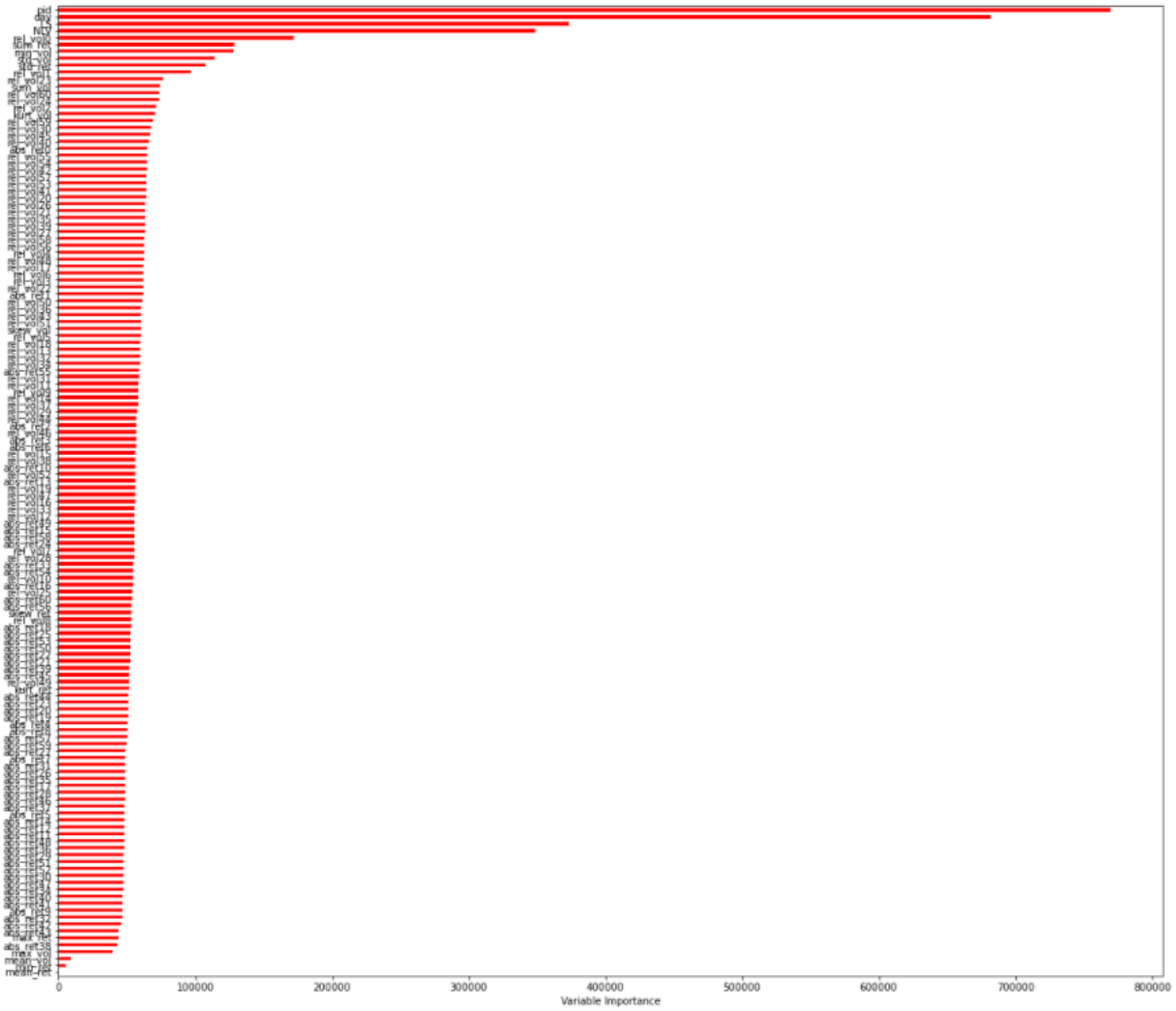


FIGURE 1.7 – Features importance selon LightGBM