PA-C21044_Nathan_Brunet

January 25, 2021

[Arts et Métiers]

Projet 2020 - 2021

Analyse et traitement de données - Prédiction du rendement des actions - Qube RT

Nathan BRUNET

Descriptif du travail attendu

- Contexte:

Les stratégies d'investissement quantitatives nécessitent l'analyse des données historiques pour prédire la tendance d'une action dans un proche avenir. Cependant, le niveau extrêmement bas de signal / bruit en fait un problème très difficile. Creuser de légères informations parmi l'énorme quantité de données disponibles sur le marché est un objectif clé pour Qube RT. Pour ce faire, les techniques d'apprentissage automatique peuvent être utilisées pour prendre de meilleures décisions commerciales grâce à une analyse approfondie de milliers de sources de données différentes. Dans un monde financier en constante évolution, il est extrêmement difficile de détecter des schémas qui font monter ou descendre un titre. Ce défi est une illustration de la prévision des stocks financiers.

- Description:

Le défi proposé vise à prédire le rendement d'une action sur le marché américain en utilisant des données historiques sur une période récente de 20 jours. Dans ce défi, nous considérons le rendement résiduel de l'action, qui correspond au rendement d'une action sans impact de marché. Les données historiques sont composées des rendements résiduels des actions et des volumes relatifs, échantillonnés chaque jour au cours des 20 derniers jours ouvrables (environ un mois).

###

Travaux

1 I) Intégration

Importations des librairies

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
# Format des données
import pandas as pd
import numpy as np

# Outils de graphs
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Scaling
from sklearn.preprocessing import scale

# Temps
import time


# Machine Learning Models - Classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# Evaluation
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score

# Amélioration des modèles
from sklearn.model_selection import GridSearchCV
```

Importation des données

[3]:
```
X_train = pd.read_csv('./Données projet Qube RT/x_train_Lafd4AH.csv', sep = ',')
X_test = pd.read_csv('./Données projet Qube RT/x_test_c7ETL4q.csv', sep = ',')
y_train = pd.read_csv('./Données projet Qube RT/y_train_JQU4vbI.csv', sep = ',')
y_test = pd.read_csv('./Données projet Qube RT/test_rand.csv', sep = ',')
```

[4]:
```
X_train.head()
```

[4]:

	ID	DATE	STOCK	INDUSTRY	INDUSTRY_GROUP	SECTOR	SUB_INDUSTRY	RET_1	\
0	0	0	2	18	5	3	44	-0.015748	
1	1	0	3	43	15	6	104	0.003984	
2	2	0	4	57	20	8	142	0.000440	
3	3	0	8	1	1	1	2	0.031298	
4	4	0	14	36	12	5	92	0.027273	

	VOLUME_1	RET_2	…	RET_16	VOLUME_16	RET_17	VOLUME_17	\
0	0.147931	-0.015504	…	0.059459	0.630899	0.003254	-0.379412	
1	NaN	-0.090580	…	0.015413	NaN	0.003774	NaN	

```
2 -0.096282 -0.058896  …  0.008964  -0.010336 -0.017612  -0.354333
3 -0.429540  0.007756  … -0.031769   0.012105  0.033824  -0.290178
4 -0.847155 -0.039302  … -0.038461  -0.277083 -0.012659   0.139086

     RET_18  VOLUME_18    RET_19  VOLUME_19    RET_20  VOLUME_20
0  0.008752  -0.110597 -0.012959   0.174521 -0.002155  -0.000937
1 -0.018518        NaN -0.028777        NaN -0.034722        NaN
2 -0.006562  -0.519391 -0.012101  -0.356157 -0.006867  -0.308868
3 -0.001468  -0.663834 -0.013520  -0.562126 -0.036745  -0.631458
4  0.004237  -0.017547  0.004256   0.579510 -0.040817   0.802806

[5 rows x 47 columns]
```

[5]:
```
y_train.head()
```

[5]:
```
   ID    RET
0   0   True
1   1   True
2   2  False
3   3  False
4   4  False
```

2 II) Data Cleaning

Y a-t-il des values non complétées (NaN) ?

[6]:
```
X_train.shape
```

[6]:
```
(418595, 47)
```

[7]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418595 entries, 0 to 418594
Data columns (total 47 columns):
ID                418595 non-null int64
DATE              418595 non-null int64
STOCK             418595 non-null int64
INDUSTRY          418595 non-null int64
INDUSTRY_GROUP    418595 non-null int64
SECTOR            418595 non-null int64
SUB_INDUSTRY      418595 non-null int64
RET_1             416236 non-null float64
VOLUME_1          353570 non-null float64
RET_2             416130 non-null float64
VOLUME_2          352209 non-null float64
```

```
RET_3             416088 non-null float64
VOLUME_3          350776 non-null float64
RET_4             416051 non-null float64
VOLUME_4          347598 non-null float64
RET_5             416011 non-null float64
VOLUME_5          343902 non-null float64
RET_6             415998 non-null float64
VOLUME_6          343881 non-null float64
RET_7             416010 non-null float64
VOLUME_7          344742 non-null float64
RET_8             415972 non-null float64
VOLUME_8          344697 non-null float64
RET_9             415913 non-null float64
VOLUME_9          345297 non-null float64
RET_10            415903 non-null float64
VOLUME_10         345290 non-null float64
RET_11            415634 non-null float64
VOLUME_11         346570 non-null float64
RET_12            415409 non-null float64
VOLUME_12         356072 non-null float64
RET_13            415235 non-null float64
VOLUME_13         359587 non-null float64
RET_14            414182 non-null float64
VOLUME_14         357666 non-null float64
RET_15            413605 non-null float64
VOLUME_15         352222 non-null float64
RET_16            413315 non-null float64
VOLUME_16         351333 non-null float64
RET_17            413294 non-null float64
VOLUME_17         356281 non-null float64
RET_18            413288 non-null float64
VOLUME_18         351009 non-null float64
RET_19            413282 non-null float64
VOLUME_19         351266 non-null float64
RET_20            413254 non-null float64
VOLUME_20         350738 non-null float64
dtypes: float64(40), int64(7)
memory usage: 150.1 MB
```

On remarque que certaines colonnes (notamment les VOLUME) contiennent beaucoup de NaN, qui ne peuvent pas être intégrées dans nos modèles de Machine Learning. Il y a plusieurs manières de gérer les NaN : - Nous pouvons compléter les valeurs manquantes en les remplaçant par la moyenne ou médiane de la colonne - Nous pouvons utiliser un algorithme prédictif pour compléter ces valeurs - Ou nous pouvons supprimer les lignes correspondantes

Dans notre cas, completer les valeurs devra être fait de manière précise. En effet, il faudrait isoler les lignes par Categorie, Sous Categorie, … afin de ne pas fausser notre moyenne (ou médiane). Le même argument est valable pour un algorithme de prédiction. Étudions l'option de suppression des lignes : nous avons plus de 418 000 lignes pour ce DataFrame d'entrainement. En supprimant les

NaN, nous en conservons 314 000 ce qui est largement suffisant pour l'entrainement d'un modèle. Nous optons pour cette option !

[8]:
```
X_train = X_train.dropna()
```

[9]:
```
X_train.reset_index(inplace = True, drop = True)
```

[10]:
```
X_train.shape
```

[10]:
```
(314160, 47)
```

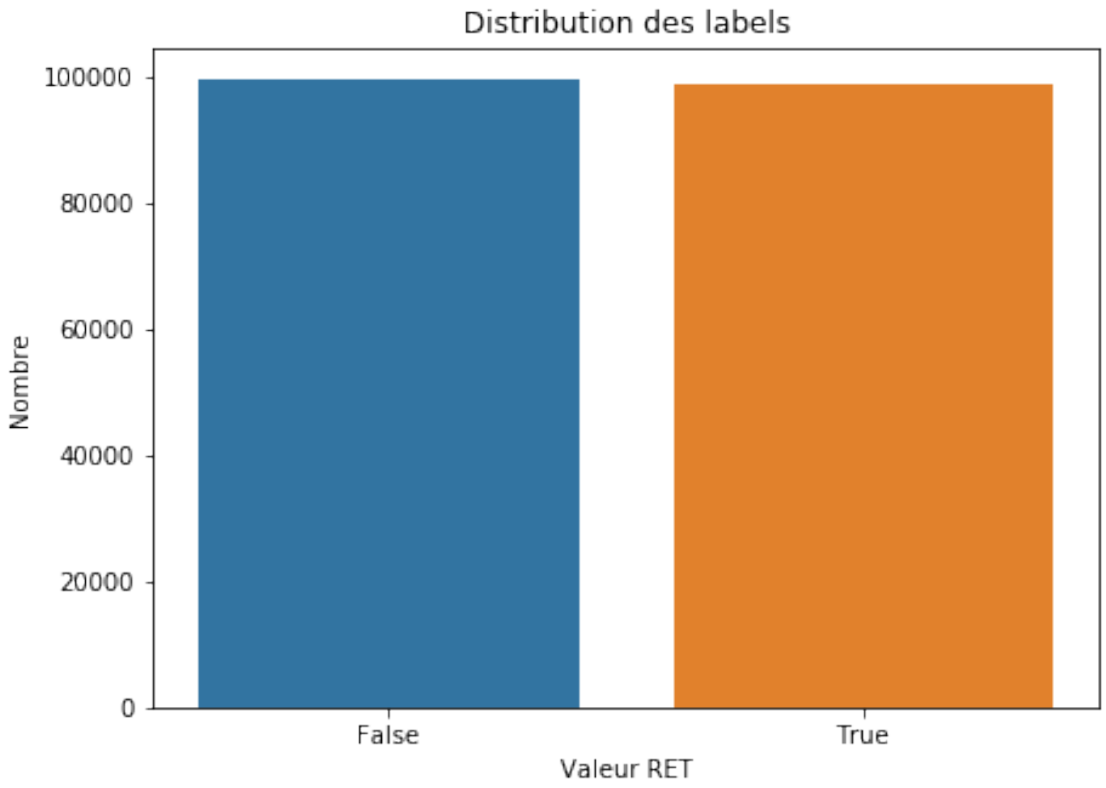
3 III) Data Exploration

3.1 III.a) Analyses des distributions

Distribution des données suivant les Labels Il est important de vérifier la distribution des valeurs à prédire dans notre jeu de données. En effet un jeu de données non équilibré peut donner des résultats faussés. Exemple :

Si nous souhaitons determiner si un patient est malade ou non, et que le jeu de données contiennent 100 lignes, dont 99 de patients sains et 1 patient malade. Il suffit d'avoir un modèle prédisant toujours 'patient sain' pour avoir 99% de bonnes réponses.

[11]:
```
plt.figure(figsize=(7,5))
sns.countplot(x='RET', data=y_test)
plt.ylabel('Nombre')
plt.xlabel('Valeur RET')
plt.title("Distribution des labels")
plt.show()
```

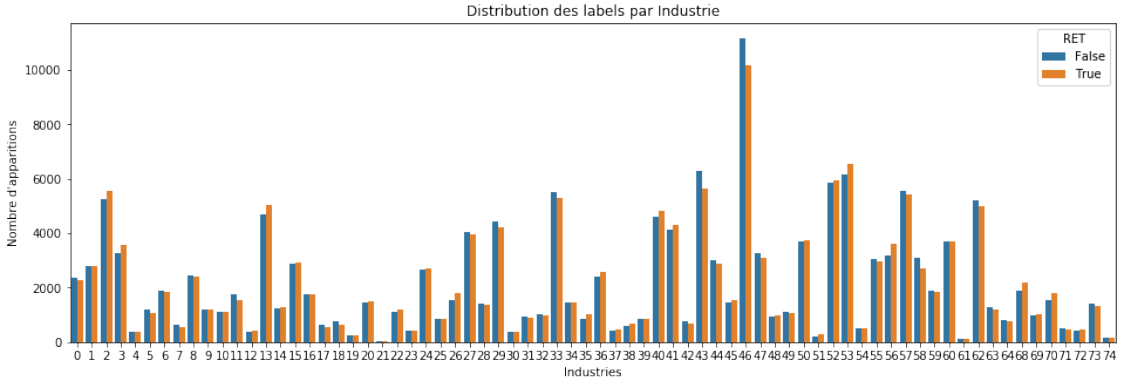


Le jeu de données est parfaitement équilibré. Cela nous donne un indice sur la metrique qui devra être utilisée pour mesurer la performance de notre modèle.

Distribution des industries

[12]:
```
data_train = X_train.merge(y_train,
                           how = 'left',
                           on = 'ID',
                           validate = '1:1')
```

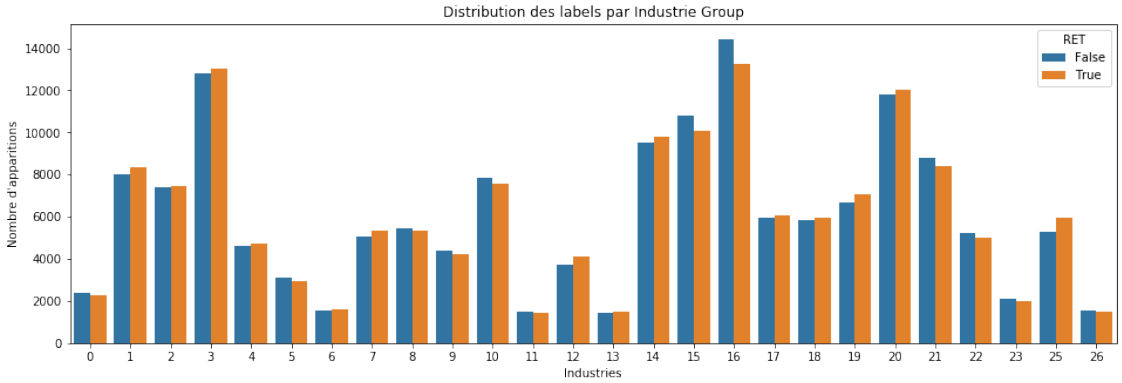
[13]:
```
plt.figure(figsize=(16,5))
sns.countplot(x='INDUSTRY', hue = 'RET', data=data_train)
plt.ylabel("Nombre d'apparitions")
plt.xlabel('Industries')
plt.title("Distribution des labels par Industrie")
plt.show()
```



Nous remarquons une assez grande disparité dans la nombre d'actions par Industrie, allant de quelques centaines à plusieurs milliers. De plus, la répartition du RET au sein des Industries semble être équilibrée, même si nous remarquons de faible variations entre le nombre de 0 et de 1. Etant donné que nous cherchons des informations très fines, une approche par industrie peut s'avérer pertinente.

Répartition des groupes d'industries

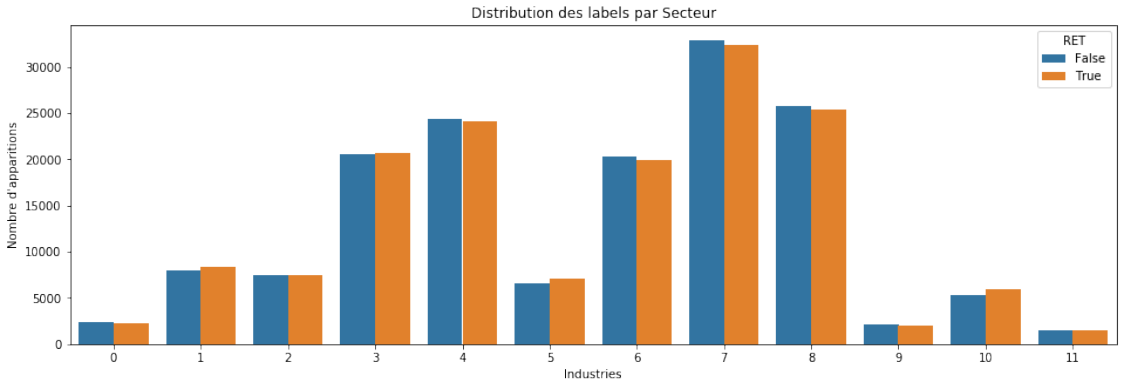
[14]:
```
plt.figure(figsize=(16,5))
sns.countplot(x='INDUSTRY_GROUP', hue = 'RET', data=data_train)
plt.ylabel("Nombre d'apparitions")
plt.xlabel('Industries')
plt.title("Distribution des labels par Industrie Group")
plt.show()
```



Nous retrouvons les mêmes insights que pour le cas des industries. Toutefois, nous avons beaucoup moins d'individus (26 pour 74 dans le cas précédents). Une approche par modèle dédié à chaque groupe d'industrie peut être pertinente, car moins concentrée sur une seule industrie, mais prenant en compte les fluctuations présentées par le graphe. Elle n'est peut être pas assez fine pour capter les disparités de notre jeu de données.

Répartition des secteurs

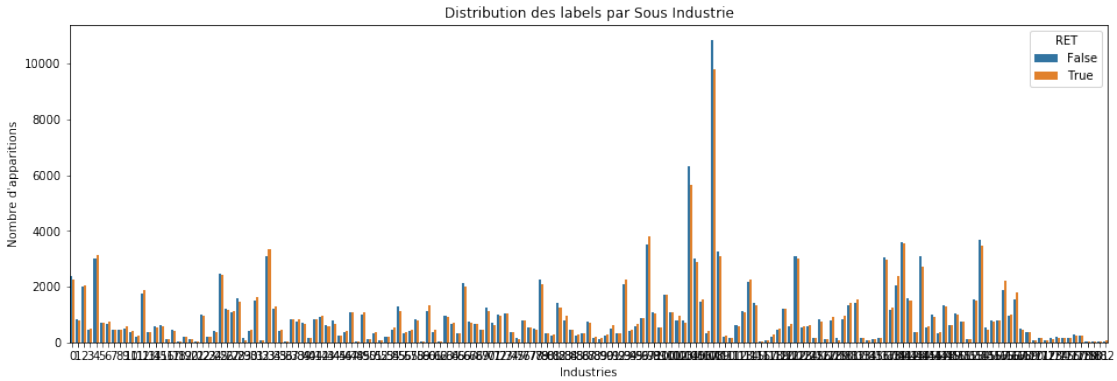
[15]:
```
plt.figure(figsize=(16,5))
sns.countplot(x='SECTOR', hue = 'RET', data=data_train)
plt.ylabel("Nombre d'apparitions")
plt.xlabel('Industries')
plt.title("Distribution des labels par Secteur")
plt.show()
```



Nous gagnons en diminution du nombre d'individus, mais nous perdons en finesse d'information. Il y a un équilibre à trouver pour ne pas tomber dans l'overfiting, tout en tirant au maximum profit des flutuactions.

Répartition des sous industries

[16]:
```
plt.figure(figsize=(16,5))
sns.countplot(x='SUB_INDUSTRY', hue = 'RET', data=data_train)
plt.ylabel("Nombre d'apparitions")
plt.xlabel('Industries')
plt.title("Distribution des labels par Sous Industrie")
plt.show()
```



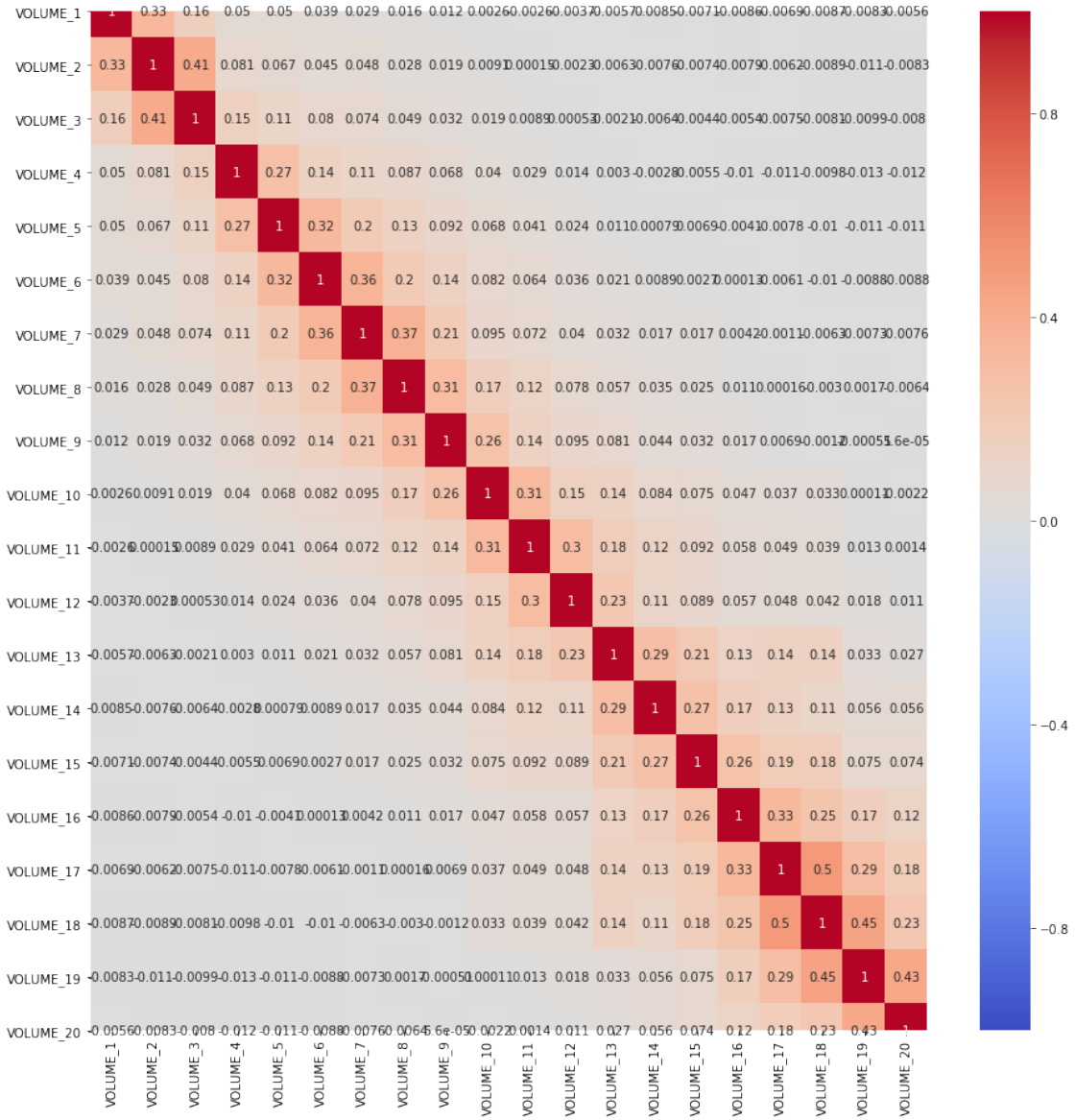
Ce cas est semblable à celui des industries : nous avons une énorme finesse, qui met en évidence beaucoup de flutuactions. Toutefois, trop de sous industries sont présentes pour pouvoir en faire un modèle pour chacune sans overfitter.

3.2 III.b) Analyses des corrélations

Correlation entre les volumes

[17]:
```
columns = ['VOLUME_' + str(i) for i in range(1,21)]
```

[18]:
```
corr = data_train[columns].corr()
ax, fig = plt.subplots(figsize=(15,15))
sns.heatmap(corr, vmin=-1, cmap='coolwarm', annot=True)
plt.show()
```



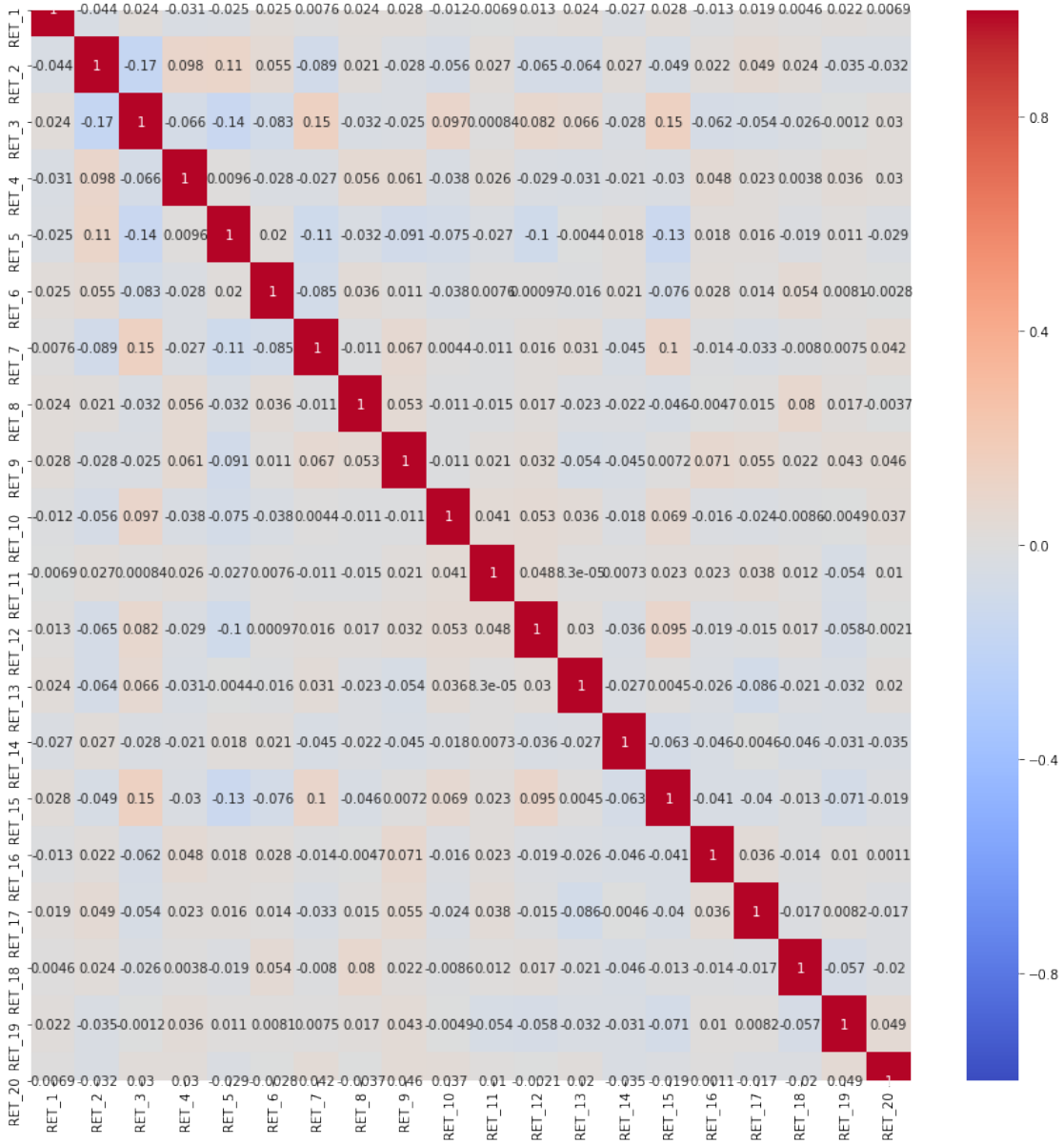
Nous remarquons que le Volume d'un jour i est corrêlé en moyenne à 30% au jour i-1 le précédent. Puis la corrélation décroit au fur et à mesure que l'on s'éloigne du jour i. Entre le jour i et le jour i-20, nous avons une corrélation d'environ 5%. Cela explique le choix de ne conserver uniquement les 20 jours antérieurs. Au delà, la corrélation est trop mince pour être utile à la prédiction et sera inutile.

Correlation entre les Retours

[19]:
```
columns = ['RET_' + str(i) for i in range(1,21)]
```

[20]:
```
corr = data_train[columns].corr()
ax, fig = plt.subplots(figsize=(15,15))
```

```
sns.heatmap(corr, vmin=-1, cmap='coolwarm', annot=True)
plt.show()
```

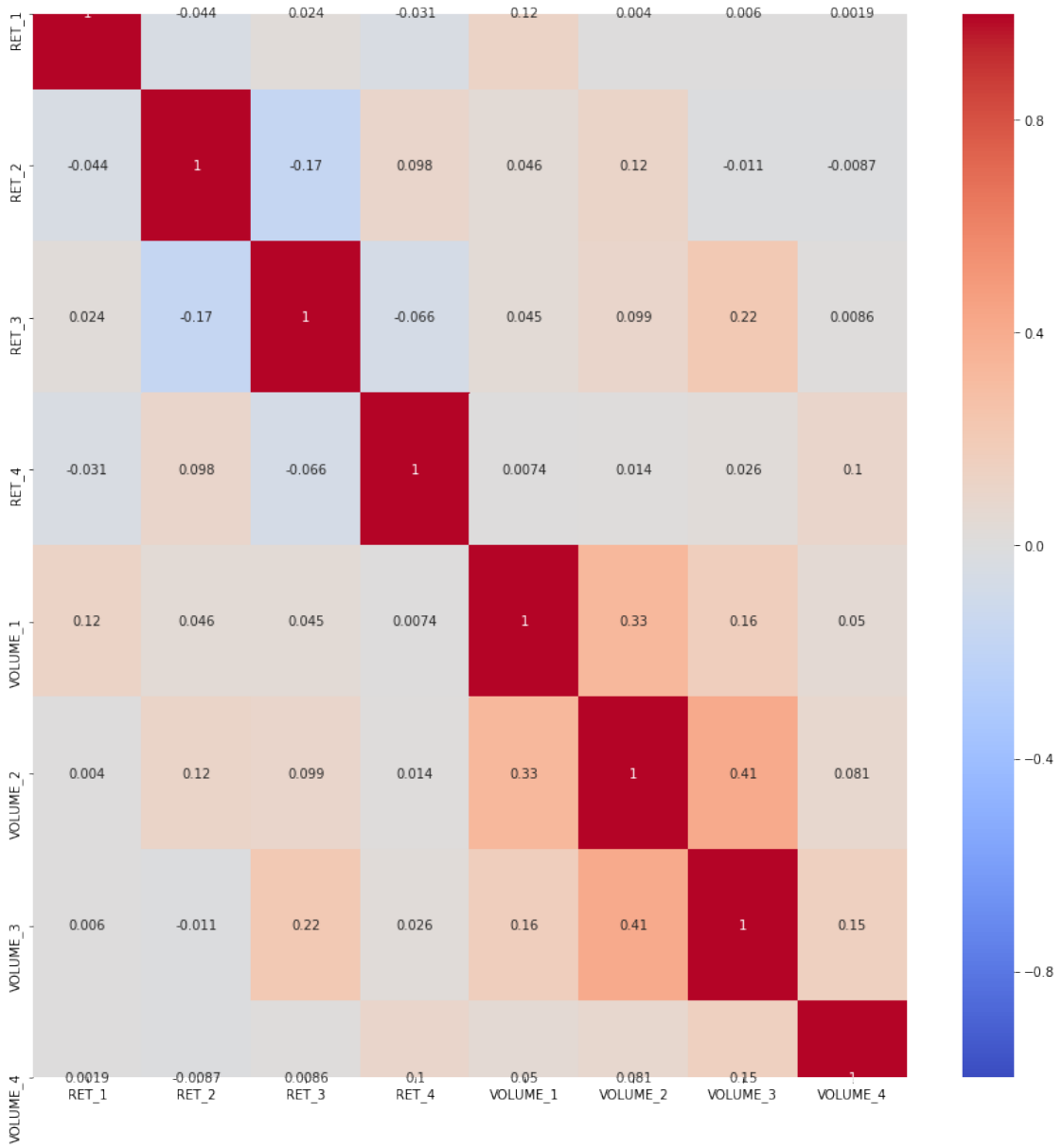


La corrélation entre les RET semble être aléatoire. En effet, dans certains cas, le RET peut être corrêlé à 17% avec le RET précent, et dans d'autres cas, avoir très peu de corrélation.

Correlation entre les Retours et Volumes

[21]:
```
columns = ['RET_' + str(i) for i in range(1,5)]
columns += ['VOLUME_' + str(i) for i in range(1,5)]
```

[22]:
```
corr = data_train[columns].corr()
ax, fig = plt.subplots(figsize=(15,15))
sns.heatmap(corr, vmin=-1, cmap='coolwarm', annot=True)
plt.show()
```



Nous remarquons une corrélation non négligeable entre le Volume_i et le RET_i, pouvant aller de 10% à 22%. Le volume est donc une information importante qui doit être conservée pour la prédiction.

4 IV) Features Engineering

4.1 IV.1) Créations de nouvelles features

4.1.1 IV.1.A) Création de nouvelles features - première couche

[23]:
```
data_train = X_train.merge(y_train,
                           how = 'left',
                           on = 'ID',
                           validate = '1:1')
```

[24]:
```
### Creation de nouvelles features

# Moyenne de RET_1 conditionnellement à la Date et au Secteur
means_ret1_by_date_sector = data_train.groupby(['DATE','SECTOR']).agg({'RET_1' :
 ↪ 'mean'}).reset_index()
means_ret1_by_date_sector.columns = ['DATE','SECTOR','Mean_RET1_by_Date_Sector']

# Moyenne de RET_1 conditionnellement à la Date et à l'Industrie
means_ret1_by_date_industry = data_train.groupby(['DATE','INDUSTRY']).
 ↪agg({'RET_1' : 'mean'}).reset_index()
means_ret1_by_date_industry.columns =␣
 ↪['DATE','INDUSTRY','Mean_RET1_by_Date_Industry']

# Moyenne de Volume_1 conditionnellement à la Date et au Secteur
means_Vol1_by_date_sector = data_train.groupby(['DATE','SECTOR']).
 ↪agg({'VOLUME_1' : 'mean'}).reset_index()
means_Vol1_by_date_sector.columns = ['DATE','SECTOR','Mean_Vol1_by_Date_Sector']

# Moyenne de Volume_1 conditionnellement à la Date et à l'Industrie
means_Vol1_by_date_industry = data_train.groupby(['DATE','INDUSTRY']).
 ↪agg({'VOLUME_1' : 'mean'}).reset_index()
means_Vol1_by_date_industry.columns =␣
 ↪['DATE','INDUSTRY','Mean_Vol1_by_Date_Industry']
```

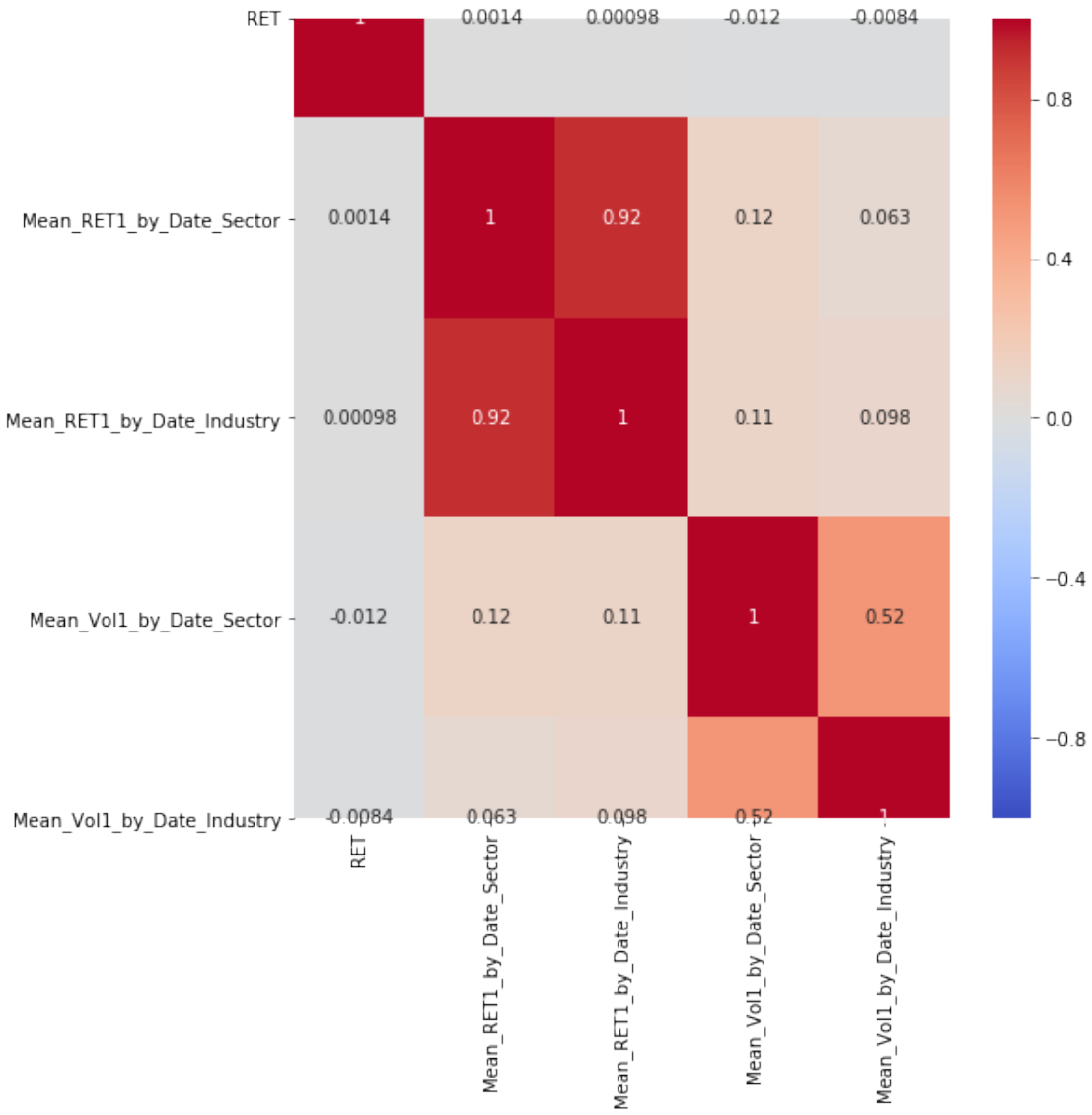
[25]:
```
### Ajout des colonnes aux données

data_train = data_train.merge(means_ret1_by_date_sector,
                              how = 'left',
                              on = ['DATE','SECTOR'],
                              validate = 'm:1')

data_train = data_train.merge(means_ret1_by_date_industry,
                              how = 'left',
                              on = ['DATE','INDUSTRY'],
                              validate = 'm:1')
```

```
data_train = data_train.merge(means_Vol1_by_date_sector,
                              how = 'left',
                              on = ['DATE','SECTOR'],
                              validate = 'm:1')

data_train = data_train.merge(means_Vol1_by_date_industry,
                              how = 'left',
                              on = ['DATE','INDUSTRY'],
                              validate = 'm:1')
```

Correlation entre RET et les nouvelles features - première couche

[26]:
```
columns = ['RET',
       'Mean_RET1_by_Date_Sector', 'Mean_RET1_by_Date_Industry',
       'Mean_Vol1_by_Date_Sector', 'Mean_Vol1_by_Date_Industry']
```

[27]:
```
corr = data_train[columns].corr()
ax, fig = plt.subplots(figsize=(8,8))
sns.heatmap(corr, vmin=-1, cmap='coolwarm', annot=True)
plt.show()
```



Nous obtenons une corrélation de plus d'1% pour les moyennes de RET_1 et VOLUME_1 conditionnellement à la date et au Secteur. Nous décidons de les conserver pour la suite de notre étude.

4.1.2 IV.1.B) Création de nouvelles features - deuxième couche

[28]:
```
### Creation de nouvelles features

# Moyenne de RET_1 conditionnellement à la Date et au groupe d'industrie
means_ret1_by_date_GrpInd = data_train.groupby(['DATE','INDUSTRY_GROUP']).
 ↪agg({'RET_1' : 'mean'}).reset_index()
```

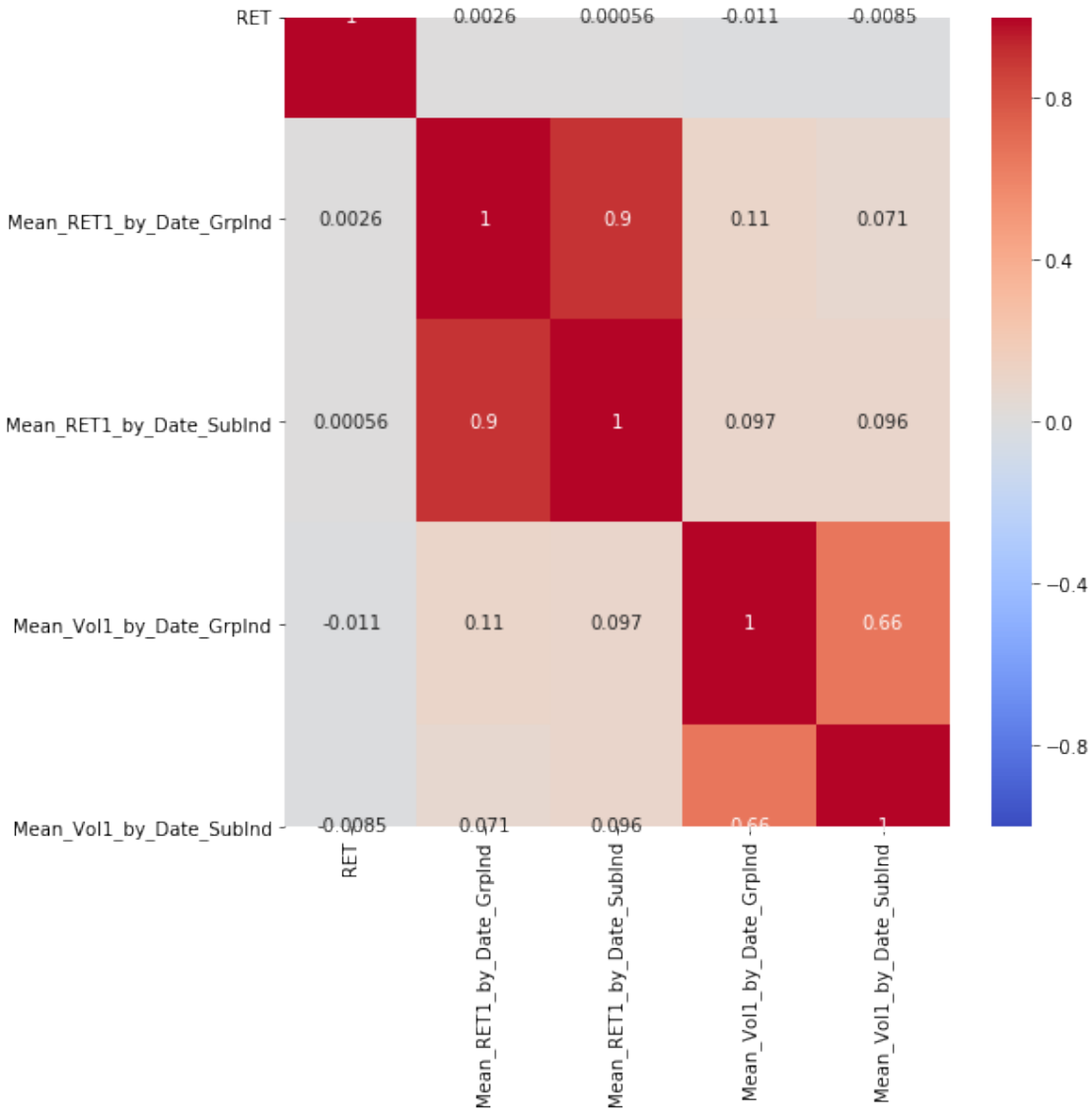
```
means_ret1_by_date_GrpInd.columns =␣
 ↪['DATE','INDUSTRY_GROUP','Mean_RET1_by_Date_GrpInd']

# Moyenne de RET_1 conditionnellement à la Date et à la sous industrie
means_ret1_by_date_SubInd = data_train.groupby(['DATE','SUB_INDUSTRY']).
 ↪agg({'RET_1' : 'mean'}).reset_index()
means_ret1_by_date_SubInd.columns =␣
 ↪['DATE','SUB_INDUSTRY','Mean_RET1_by_Date_SubInd']

# Moyenne de Volume_1 conditionnellement à la Date et au groupe d'industrie
means_Vol1_by_date_GrpInd = data_train.groupby(['DATE','INDUSTRY_GROUP']).
 ↪agg({'VOLUME_1' : 'mean'}).reset_index()
means_Vol1_by_date_GrpInd.columns =␣
 ↪['DATE','INDUSTRY_GROUP','Mean_Vol1_by_Date_GrpInd']

# Moyenne de Volume_1 conditionnellement à la Date et à la sous industrie
means_Vol1_by_date_SubInd = data_train.groupby(['DATE','SUB_INDUSTRY']).
 ↪agg({'VOLUME_1' : 'mean'}).reset_index()
means_Vol1_by_date_SubInd.columns =␣
 ↪['DATE','SUB_INDUSTRY','Mean_Vol1_by_Date_SubInd']
```

[29]:
```
### Ajout des colonnes aux données

data_train = data_train.merge(means_ret1_by_date_GrpInd,
                              how = 'left',
                              on = ['DATE','INDUSTRY_GROUP'],
                              validate = 'm:1')

data_train = data_train.merge(means_ret1_by_date_SubInd,
                              how = 'left',
                              on = ['DATE','SUB_INDUSTRY'],
                              validate = 'm:1')

data_train = data_train.merge(means_Vol1_by_date_GrpInd,
                              how = 'left',
                              on = ['DATE','INDUSTRY_GROUP'],
                              validate = 'm:1')

data_train = data_train.merge(means_Vol1_by_date_SubInd,
                              how = 'left',
                              on = ['DATE','SUB_INDUSTRY'],
                              validate = 'm:1')
```

Correlation entre RET et les nouvelles features - seconde couche

[30]:
```
columns = ['RET',
       'Mean_RET1_by_Date_GrpInd', 'Mean_RET1_by_Date_SubInd',
       'Mean_Vol1_by_Date_GrpInd', 'Mean_Vol1_by_Date_SubInd']
```

[31]:
```
corr = data_train[columns].corr()
ax, fig = plt.subplots(figsize=(8,8))
sns.heatmap(corr, vmin=-1, cmap='coolwarm', annot=True)
plt.show()
```



Seule la moyenne de RET_1 par Date et Sous Industrie a une corrélation supérieur à 1%. Nous décidons de la conserver.

4.2 IV.2) Préparation du Dataset

- Nous décidons de ne conserver uniquement une antériorité de 5 jours. En effet, dans la corrélation des Volumes, nous avons constaté qu'au delà de 5 jours, le seuil des 5% de corrélation n'était plus atteint. Nous appliquons la même règle pour les Retours.
- De plus, nous conservons les nouvelles variables créées suivantes : Mean_RET1_Date_Sector, Mean_VOL1_Date_Sector, Mean_RET1_Date_SubInd
- Les dates ayant été anonymisées, nous ne pouvons nous en servir en tant que Série Temporelle. Nous avons pris au maximum en compte leurs effets dans nos variables conditionnelles. Nous les supprimons de l'analyse.
- Les variables Stock, Industry, Sector, Sub Industry, Industry Group, sont des variables catégorielles ayant été encodées. Nous les avons prise en compte dans nos variables conditionnelles, nous préférons les supprimer dans un premier temps.

[32]:
```
df_train = data_train[['RET_1', 'VOLUME_1', 'RET_2', 'VOLUME_2', 'RET_3',
                       'VOLUME_3', 'RET_4', 'VOLUME_4', 'RET_5', 'VOLUME_5',␣
↪'RET',
                       'Mean_RET1_by_Date_Sector', 'Mean_Vol1_by_Date_Sector',
                       'Mean_RET1_by_Date_SubInd']]
```

[33]:
```
df_train.head()
```

[33]:

	RET_1	VOLUME_1	RET_2	VOLUME_2	RET_3	VOLUME_3	RET_4	VOLUME_4	RET_5	VOLUME_5	RET	Mean_RET1_by_Date_Sector	Mean_Vol1_by_Date_Sector	Mean_RET1_by_Date_SubInd
0	-0.015748	0.147931	-0.015504	0.179183	0.010972	0.033832	-0.014672	-0.362868	0.016483	-0.972920	True	0.009588	0.005566	0.008289
1	0.000440	-0.096282	-0.058896	0.084771	-0.009042	-0.298777	0.024852	-0.157421	0.009354	0.091455	False	0.013431	0.124533	0.014051
2	0.031298	-0.429540	0.007756	-0.089919	-0.004632	-0.639737	-0.019677	-0.940163	0.003544	-0.882464	False	0.017253	-0.121974	0.026870
3	0.027273	-0.847155	-0.039302	-0.943033	0.000000	-1.180629	0.000000	-1.313896	0.022321	-1.204398	False	0.006940	-0.190113	0.000692
4	0.010938	-0.238878	0.021548	-0.322706	-0.016097	0.747003	-0.027120	0.688104	-0.007958	-0.182443	False	0.006940	-0.190113	0.009595

5 V) Classification, Test et Optimisation

5.1 V.I) Modèle général

5.1.1 V.I.1) Construction du modèle

Choix du modèle

[34]:
```
models = []
models.append(('Logistic Regression',LogisticRegression()))
models.append(('Decision Tree Classifier',DecisionTreeClassifier()))
models.append(('Random Forest Classifier',RandomForestClassifier()))
```

Normalization

[35]:
```
#  On normalise les données :

X1 = df_train.loc[:,['RET_1', 'VOLUME_1', 'RET_2', 'VOLUME_2', 'RET_3',
                     'VOLUME_3', 'RET_4', 'VOLUME_4', 'RET_5', 'VOLUME_5',
                     'Mean_RET1_by_Date_Sector', 'Mean_Vol1_by_Date_Sector',
                     'Mean_RET1_by_Date_SubInd']]
y1 = df_train.RET
X1_scale = (X1-X1.max())/(X1.max() - X1.min())
```

Cross Validation Afin de déterminer quel modèle est le plus adapté pour notre jeu de données, nous en avons préselectionné 3 que nous allons évaluer. Pour cette évaluation : - Nous choisissons la métrique “Accuracy” qui est adapté dans notre cas, notre problème étant une classification avec un jeu de données équilibré - Nous évaluons nos modèles grâce à une cross validation avec k = 7

[36]:
```
accuracy_results = list()

index = [m[0] for m in models]
for nom_model,model in models:
    print(nom_model)
    cv_results = cross_val_score(model, X1_scale, y1, cv=7, scoring='accuracy')
    accuracy_results.append(cv_results.mean()*100)

Performance_results_scaled = pd.DataFrame(accuracy_results , index=index ,␣
 ↪columns= ['Performance en %'])
```

```
Logistic Regression
Decision Tree Classifier
Random Forest Classifier
```

[37]:
```
Performance_results_scaled
```

[37]:
```
                           Performance en %
Logistic Regression               50.531258
Decision Tree Classifier          49.815059
Random Forest Classifier          50.416347
```

L'accuracy peut sembler très faible (proche de 50% qui est le minimum), la vidéo explicative du challenge indique qu'il est courant dans ce cas de tourner autour de 50%. Essayons d'augmenter nos résultats et d'atteindre 52% en optimisant nos paramètres.

5.1.2 V.I.2) Optimisation des hyperparamètres

Parmi les 3 algorithmes testés, la Régression Logistique et le Random Forest se sont démarqués. Toutefois, la régression logistique n'a pas pu énormément être améliorée, donc nous nous concentrons sur l'optimisation du Random Forest.

[38]:
```
best_clf = RandomForestClassifier()
best_clf.get_params
```

[38]:
```
<bound method BaseEstimator.get_params of RandomForestClassifier(bootstrap=True,
class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators='warn',
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)>
```

[39]:
```
param_grid={'max_depth' : [8,12], 'n_estimators' : [100,200]}
```

[40]:
```
clf = GridSearchCV(best_clf,param_grid, cv=3, scoring = 'accuracy')
```

[41]:
```
tic = time.time()
clf.fit(X1_scale,y1)
toc = time.time()
print(toc-tic)
```

```
1370.0102667808533
```

[42]:
```
clf.best_score_
```

[42]:
```
0.5153361344537815
```

[43]:
```
clf.best_params_
```

[43]:
```
{'max_depth': 12, 'n_estimators': 200}
```

D'après GridSearch, et sous réserve des paramètres testés, nous pouvons obtenir au maximum une accuracy de 51.56% pour les paramètres suivants : 'max_depth': 12, 'n_estimators': 100

5.2 V.II) Modèle par Industrie

Choix du modèle par Industrie

[44]:
```
def get_data_per_industrie(ind):
    data = data_train[data_train.INDUSTRY == ind]
    X = data.loc[:,['RET_1', 'VOLUME_1', 'RET_2', 'VOLUME_2', 'RET_3',
                    'VOLUME_3', 'RET_4', 'VOLUME_4', 'RET_5', 'VOLUME_5',
                    'Mean_RET1_by_Date_Sector', 'Mean_Vol1_by_Date_Sector',
                    'Mean_RET1_by_Date_SubInd']]

    X = (X-X.max())/(X.max() - X.min())
    y = data.RET
    return X, y
```

[45]:
```
models = []
models.append(('Logistic Regression',LogisticRegression()))
models.append(('Decision Tree Classifier',DecisionTreeClassifier()))
models.append(('Random Forest Classifier',RandomForestClassifier()))
```

[46]:
```
accuracy_results = []
ind_list = []
mod = []
index = [m[0] for m in models]

for ind in data_train.INDUSTRY.unique():
    for nom_model,model in models:
        print(ind, nom_model)
        X, y = get_data_per_industrie(ind)
        cv_results = cross_val_score(model, X, y, cv=7, scoring='accuracy')
        accuracy_results.append(cv_results.mean()*100)
        mod.append(nom_model)
        ind_list.append(ind)
```

```
18 Logistic Regression
18 Decision Tree Classifier
18 Random Forest Classifier
57 Logistic Regression
57 Decision Tree Classifier
57 Random Forest Classifier
1 Logistic Regression
1 Decision Tree Classifier
1 Random Forest Classifier
36 Logistic Regression
36 Decision Tree Classifier
```

```
36 Random Forest Classifier
37 Logistic Regression
37 Decision Tree Classifier
37 Random Forest Classifier
52 Logistic Regression
52 Decision Tree Classifier
52 Random Forest Classifier
56 Logistic Regression
56 Decision Tree Classifier
56 Random Forest Classifier
44 Logistic Regression
44 Decision Tree Classifier
44 Random Forest Classifier
25 Logistic Regression
25 Decision Tree Classifier
25 Random Forest Classifier
50 Logistic Regression
50 Decision Tree Classifier
50 Random Forest Classifier
41 Logistic Regression
41 Decision Tree Classifier
41 Random Forest Classifier
33 Logistic Regression
33 Decision Tree Classifier
33 Random Forest Classifier
13 Logistic Regression
13 Decision Tree Classifier
13 Random Forest Classifier
46 Logistic Regression
46 Decision Tree Classifier
46 Random Forest Classifier
43 Logistic Regression
43 Decision Tree Classifier
43 Random Forest Classifier
54 Logistic Regression
54 Decision Tree Classifier
54 Random Forest Classifier
55 Logistic Regression
55 Decision Tree Classifier
55 Random Forest Classifier
60 Logistic Regression
60 Decision Tree Classifier
60 Random Forest Classifier
71 Logistic Regression
71 Decision Tree Classifier
71 Random Forest Classifier
16 Logistic Regression
16 Decision Tree Classifier
```

```
16 Random Forest Classifier
27 Logistic Regression
27 Decision Tree Classifier
27 Random Forest Classifier
62 Logistic Regression
62 Decision Tree Classifier
62 Random Forest Classifier
53 Logistic Regression
53 Decision Tree Classifier
53 Random Forest Classifier
47 Logistic Regression
47 Decision Tree Classifier
47 Random Forest Classifier
2 Logistic Regression
2 Decision Tree Classifier
2 Random Forest Classifier
30 Logistic Regression
30 Decision Tree Classifier
30 Random Forest Classifier
39 Logistic Regression
39 Decision Tree Classifier
39 Random Forest Classifier
6 Logistic Regression
6 Decision Tree Classifier
6 Random Forest Classifier
14 Logistic Regression
14 Decision Tree Classifier
14 Random Forest Classifier
3 Logistic Regression
3 Decision Tree Classifier
3 Random Forest Classifier
9 Logistic Regression
9 Decision Tree Classifier
9 Random Forest Classifier
29 Logistic Regression
29 Decision Tree Classifier
29 Random Forest Classifier
0 Logistic Regression
0 Decision Tree Classifier
0 Random Forest Classifier
40 Logistic Regression
40 Decision Tree Classifier
40 Random Forest Classifier
15 Logistic Regression
15 Decision Tree Classifier
15 Random Forest Classifier
24 Logistic Regression
24 Decision Tree Classifier
```

```
24 Random Forest Classifier
11 Logistic Regression
11 Decision Tree Classifier
11 Random Forest Classifier
4 Logistic Regression
4 Decision Tree Classifier
4 Random Forest Classifier
49 Logistic Regression
49 Decision Tree Classifier
49 Random Forest Classifier
23 Logistic Regression
23 Decision Tree Classifier
23 Random Forest Classifier
58 Logistic Regression
58 Decision Tree Classifier
58 Random Forest Classifier
45 Logistic Regression
45 Decision Tree Classifier
45 Random Forest Classifier
28 Logistic Regression
28 Decision Tree Classifier
28 Random Forest Classifier
26 Logistic Regression
26 Decision Tree Classifier
26 Random Forest Classifier
12 Logistic Regression
12 Decision Tree Classifier
12 Random Forest Classifier
8 Logistic Regression
8 Decision Tree Classifier
8 Random Forest Classifier
59 Logistic Regression
59 Decision Tree Classifier
59 Random Forest Classifier
63 Logistic Regression
63 Decision Tree Classifier
63 Random Forest Classifier
34 Logistic Regression
34 Decision Tree Classifier
34 Random Forest Classifier
31 Logistic Regression
31 Decision Tree Classifier
31 Random Forest Classifier
48 Logistic Regression
48 Decision Tree Classifier
48 Random Forest Classifier
38 Logistic Regression
38 Decision Tree Classifier
```

```
38 Random Forest Classifier
35 Logistic Regression
35 Decision Tree Classifier
35 Random Forest Classifier
64 Logistic Regression
64 Decision Tree Classifier
64 Random Forest Classifier
68 Logistic Regression
68 Decision Tree Classifier
68 Random Forest Classifier
72 Logistic Regression
72 Decision Tree Classifier
72 Random Forest Classifier
32 Logistic Regression
32 Decision Tree Classifier
32 Random Forest Classifier
7 Logistic Regression
7 Decision Tree Classifier
7 Random Forest Classifier
70 Logistic Regression
70 Decision Tree Classifier
70 Random Forest Classifier
20 Logistic Regression
20 Decision Tree Classifier
20 Random Forest Classifier
17 Logistic Regression
17 Decision Tree Classifier
17 Random Forest Classifier
5 Logistic Regression
5 Decision Tree Classifier
5 Random Forest Classifier
42 Logistic Regression
42 Decision Tree Classifier
42 Random Forest Classifier
22 Logistic Regression
22 Decision Tree Classifier
22 Random Forest Classifier
10 Logistic Regression
10 Decision Tree Classifier
10 Random Forest Classifier
61 Logistic Regression
61 Decision Tree Classifier
61 Random Forest Classifier
69 Logistic Regression
69 Decision Tree Classifier
69 Random Forest Classifier
19 Logistic Regression
19 Decision Tree Classifier
```

```
19 Random Forest Classifier
51 Logistic Regression
51 Decision Tree Classifier
51 Random Forest Classifier
73 Logistic Regression
73 Decision Tree Classifier
73 Random Forest Classifier
74 Logistic Regression
74 Decision Tree Classifier
74 Random Forest Classifier
21 Logistic Regression
21 Decision Tree Classifier
21 Random Forest Classifier
```

[47]:
```
Performance_results_ind = pd.DataFrame( columns=␣
↪['Industrie','Model','Performance en %'])
Performance_results_ind['Industrie'] = ind_list
Performance_results_ind['Model'] = mod
Performance_results_ind['Performance en %'] = accuracy_results
```

On garde le meilleur modèle pour chaque industrie

[48]:
```
A = Performance_results_ind.groupby(['Industrie']).agg({'Performance en %':
↪'max'}).reset_index()
Best_model = A.merge(Performance_results_ind,
           how ='left',
           on = ['Industrie','Performance en %'])
```

[49]:
```
Best_model.head()
```

[49]:

	Industrie	Performance en %	Model
0	0	51.154297	Logistic Regression
1	1	46.143883	Logistic Regression
2	2	51.294754	Logistic Regression
3	3	52.608616	Logistic Regression
4	4	52.854446	Logistic Regression

Calcul de la moyenne

[50]:
```
Best_model['Performance en %'].mean()
```

[50]:
```
50.79636616390323
```

En moyenne, nous obtenons 50.79% d'accuracy avec cette approche basé sur un model par Industrie. Toutefois, nous remarquons que cette industrie ont une accuracy allant jusqu'à 55% (ce qui est exceptionnel), et d'autres ne sont qu'à 45%. Une idée serait d'optimiser les hyperparamètres

pour tous ces modèles, mais l'algorithme est beaucoup trop long, un ordinateur plus puissant est nécessaire.

6 VI) Prédiction finale

6.1 VI.1) Mise en forme du dataset de Test

Retrait des Nan

[51]:
```
# Retrait des Nan
X_test.dropna(inplace = True)
```

Ajout des nouvelles features

[52]:
```
### Creation de nouvelles features

# Moyenne de RET_1 conditionnellement à la Date et au Secteur
means_ret1_by_date_sector = X_test.groupby(['DATE','SECTOR']).agg({'RET_1' :␣
↪'mean'}).reset_index()
means_ret1_by_date_sector.columns = ['DATE','SECTOR','Mean_RET1_by_Date_Sector']

# Moyenne de Volume_1 conditionnellement à la Date et au Secteur
means_Vol1_by_date_sector = X_test.groupby(['DATE','SECTOR']).agg({'VOLUME_1' :␣
↪'mean'}).reset_index()
means_Vol1_by_date_sector.columns = ['DATE','SECTOR','Mean_Vol1_by_Date_Sector']

# Moyenne de RET_1 conditionnellement à la Date et à la sous industrie
means_ret1_by_date_SubInd = X_test.groupby(['DATE','SUB_INDUSTRY']).
↪agg({'RET_1' : 'mean'}).reset_index()
means_ret1_by_date_SubInd.columns =␣
↪['DATE','SUB_INDUSTRY','Mean_RET1_by_Date_SubInd']
```

[53]:
```
### Ajout des colonnes aux données

X_test = X_test.merge(means_ret1_by_date_sector,
                      how = 'left',
                      on = ['DATE','SECTOR'],
                      validate = 'm:1')

X_test = X_test.merge(means_Vol1_by_date_sector,
                      how = 'left',
                      on = ['DATE','SECTOR'],
                      validate = 'm:1')

X_test = X_test.merge(means_ret1_by_date_SubInd,
                      how = 'left',
```

```
                                on = ['DATE','SUB_INDUSTRY'],
                                validate = 'm:1')
```

Sélection des variables

[54]:
```
df_test = X_test.merge(y_test,
                       how = 'left',
                       on = 'ID')
```

[55]:
```
#  On normalise les données :

X1_test = df_test.loc[:,['RET_1', 'VOLUME_1', 'RET_2', 'VOLUME_2', 'RET_3',
                   'VOLUME_3', 'RET_4', 'VOLUME_4', 'RET_5', 'VOLUME_5',
                   'Mean_RET1_by_Date_Sector', 'Mean_Vol1_by_Date_Sector',
                   'Mean_RET1_by_Date_SubInd']]
y1_test = df_test.RET
X1_test_scale = scale(X1_test)
```

6.2 VI.2) Modèle général

6.2.1 VI.2.A) Entrainement du modèle général

[56]:
```
model_général = RandomForestClassifier(max_depth = 12, n_estimators = 200)
```

[57]:
```
model_général.fit(X1_scale,y1)
```

[57]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=12, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

6.2.2 VI.2.B) Prédiction du Test Set avec le modèle général

Prédiction

[58]:
```
y_pred = model_général.predict(X1_test_scale)
```

Evaluation

[59]:
```
accuracy_score(y1_test,y_pred)
```

[59]:
```
0.5018061510414908
```

Nous obtenons une accuracy de 50.18% avec le modèle général.

6.3 VI.3) Modèle par industrie

6.3.1 VI.3.A) Entrainement des modèles

Ajout des modèles à la base des meilleurs modèles

[60]:
```
models = { 'Logistic Regression' : LogisticRegression(),
           'Decision Tree Classifier' : DecisionTreeClassifier(),
           'Random Forest Classifier' : RandomForestClassifier()}
```

[61]:
```
Best_model['Model_code'] = Best_model.Model.map(models)
```

[62]:
```
Best_model.head()
```

[62]:
```
   Industrie  Performance en %                Model  \
0          0         51.154297  Logistic Regression
1          1         46.143883  Logistic Regression
2          2         51.294754  Logistic Regression
3          3         52.608616  Logistic Regression
4          4         52.854446  Logistic Regression

                                          Model_code
0  LogisticRegression(C=1.0, class_weight=None, d…
1  LogisticRegression(C=1.0, class_weight=None, d…
2  LogisticRegression(C=1.0, class_weight=None, d…
3  LogisticRegression(C=1.0, class_weight=None, d…
4  LogisticRegression(C=1.0, class_weight=None, d…
```

Entrainement pour chaque modèle

[63]:
```
for i in Best_model.index:
    ind = Best_model.loc[i,'Industrie']
    print("Industrie en cours d'entrainement : " , ind)
    mod = Best_model.loc[i,'Model_code']
    X, y = get_data_per_industrie(ind)
    mod.fit(X,y)
```

```
Industrie en cours d'entrainement :  0
Industrie en cours d'entrainement :  1
Industrie en cours d'entrainement :  2
Industrie en cours d'entrainement :  3
Industrie en cours d'entrainement :  4
Industrie en cours d'entrainement :  5
Industrie en cours d'entrainement :  6
Industrie en cours d'entrainement :  7
```

```
Industrie en cours d'entrainement :  8
Industrie en cours d'entrainement :  9
Industrie en cours d'entrainement :  10
Industrie en cours d'entrainement :  11
Industrie en cours d'entrainement :  12
Industrie en cours d'entrainement :  13
Industrie en cours d'entrainement :  14
Industrie en cours d'entrainement :  15
Industrie en cours d'entrainement :  16
Industrie en cours d'entrainement :  17
Industrie en cours d'entrainement :  18
Industrie en cours d'entrainement :  19
Industrie en cours d'entrainement :  20
Industrie en cours d'entrainement :  21
Industrie en cours d'entrainement :  22
Industrie en cours d'entrainement :  23
Industrie en cours d'entrainement :  24
Industrie en cours d'entrainement :  25
Industrie en cours d'entrainement :  26
Industrie en cours d'entrainement :  27
Industrie en cours d'entrainement :  28
Industrie en cours d'entrainement :  29
Industrie en cours d'entrainement :  30
Industrie en cours d'entrainement :  31
Industrie en cours d'entrainement :  32
Industrie en cours d'entrainement :  33
Industrie en cours d'entrainement :  34
Industrie en cours d'entrainement :  35
Industrie en cours d'entrainement :  36
Industrie en cours d'entrainement :  37
Industrie en cours d'entrainement :  38
Industrie en cours d'entrainement :  39
Industrie en cours d'entrainement :  40
Industrie en cours d'entrainement :  41
Industrie en cours d'entrainement :  42
Industrie en cours d'entrainement :  43
Industrie en cours d'entrainement :  44
Industrie en cours d'entrainement :  45
Industrie en cours d'entrainement :  46
Industrie en cours d'entrainement :  47
Industrie en cours d'entrainement :  48
Industrie en cours d'entrainement :  49
Industrie en cours d'entrainement :  50
Industrie en cours d'entrainement :  51
Industrie en cours d'entrainement :  52
Industrie en cours d'entrainement :  53
Industrie en cours d'entrainement :  54
Industrie en cours d'entrainement :  55
```

```
Industrie en cours d'entrainement :  56
Industrie en cours d'entrainement :  57
Industrie en cours d'entrainement :  58
Industrie en cours d'entrainement :  59
Industrie en cours d'entrainement :  60
Industrie en cours d'entrainement :  61
Industrie en cours d'entrainement :  62
Industrie en cours d'entrainement :  63
Industrie en cours d'entrainement :  64
Industrie en cours d'entrainement :  68
Industrie en cours d'entrainement :  69
Industrie en cours d'entrainement :  70
Industrie en cours d'entrainement :  71
Industrie en cours d'entrainement :  72
Industrie en cours d'entrainement :  73
Industrie en cours d'entrainement :  74
```

6.3.2 VI.3.B) Prédiction du Test Set avec le modèle par Industrie

Certaines industries présentes dans le jeu de test ne sont pas dans le jeu d'entrainement. Dans ces cas là, nous lançons une prédiction avec le modèle général.

Prédiction par Industrie

[86]:
```
def get_data_per_industrie_test(ind):
    data = df_test[df_test.INDUSTRY == ind]
    X = data.loc[:,['RET_1', 'VOLUME_1', 'RET_2', 'VOLUME_2', 'RET_3',
                    'VOLUME_3', 'RET_4', 'VOLUME_4', 'RET_5', 'VOLUME_5',
                    'Mean_RET1_by_Date_Sector', 'Mean_Vol1_by_Date_Sector',
                    'Mean_RET1_by_Date_SubInd']]

    X = (X-X.max())/(X.max() - X.min())
    y = data.RET
    return X, y
```

[65]:
```
list_industrie = []
valeur_accuracy = []
for ind in df_test.INDUSTRY.unique():
    print("Prédiction de l'industrie : ",ind)
    list_industrie.append(ind)
    try:
        X,y = get_data_per_industrie_test(ind)
        model = Best_model.loc[Best_model.Industrie == ind, 'Model_code'].
 ↪values[0]
        y_pred = model.predict(X)
        valeur_accuracy.append(accuracy_score(y,y_pred))
```

```
    except:
        X,y = get_data_per_industrie_test(ind)
        y_pred = model_général.predict(X)
        valeur_accuracy.append(accuracy_score(y,y_pred))
```

```
Prédiction de l'industrie :  37
Prédiction de l'industrie :  15
Prédiction de l'industrie :  57
Prédiction de l'industrie :  35
Prédiction de l'industrie :  1
Prédiction de l'industrie :  2
Prédiction de l'industrie :  10
Prédiction de l'industrie :  36
Prédiction de l'industrie :  18
Prédiction de l'industrie :  13
Prédiction de l'industrie :  48
Prédiction de l'industrie :  53
Prédiction de l'industrie :  29
Prédiction de l'industrie :  54
Prédiction de l'industrie :  6
Prédiction de l'industrie :  50
Prédiction de l'industrie :  70
Prédiction de l'industrie :  31
Prédiction de l'industrie :  41
Prédiction de l'industrie :  11
Prédiction de l'industrie :  14
Prédiction de l'industrie :  0
Prédiction de l'industrie :  52
Prédiction de l'industrie :  22
Prédiction de l'industrie :  3
Prédiction de l'industrie :  26
Prédiction de l'industrie :  16
Prédiction de l'industrie :  56
Prédiction de l'industrie :  59
Prédiction de l'industrie :  71
Prédiction de l'industrie :  44
Prédiction de l'industrie :  62
Prédiction de l'industrie :  64
Prédiction de l'industrie :  47
Prédiction de l'industrie :  58
Prédiction de l'industrie :  40
Prédiction de l'industrie :  25
Prédiction de l'industrie :  42
Prédiction de l'industrie :  33
Prédiction de l'industrie :  43
Prédiction de l'industrie :  46
Prédiction de l'industrie :  45
```

```
Prédiction de l'industrie :  60
Prédiction de l'industrie :  23
Prédiction de l'industrie :  49
Prédiction de l'industrie :  8
Prédiction de l'industrie :  55
Prédiction de l'industrie :  28
Prédiction de l'industrie :  17
Prédiction de l'industrie :  27
Prédiction de l'industrie :  32
Prédiction de l'industrie :  63
Prédiction de l'industrie :  39
Prédiction de l'industrie :  19
Prédiction de l'industrie :  34
Prédiction de l'industrie :  30
Prédiction de l'industrie :  68
Prédiction de l'industrie :  20
Prédiction de l'industrie :  38
Prédiction de l'industrie :  7
Prédiction de l'industrie :  69
Prédiction de l'industrie :  5
Prédiction de l'industrie :  9
Prédiction de l'industrie :  24
Prédiction de l'industrie :  4
Prédiction de l'industrie :  72
Prédiction de l'industrie :  12
Prédiction de l'industrie :  61
Prédiction de l'industrie :  21
Prédiction de l'industrie :  51
Prédiction de l'industrie :  74
Prédiction de l'industrie :  73
Prédiction de l'industrie :  66
Prédiction de l'industrie :  65
Prédiction de l'industrie :  67
```

[66]:
```
prediction_per_industrie = pd.DataFrame()
prediction_per_industrie['Industrie'] = list_industrie
prediction_per_industrie['Valeur_accuracy'] = valeur_accuracy
```

[67]:
```
prediction_per_industrie.head()
```

[67]:

	Industrie	Valeur_accuracy
0	37	0.436314
1	15	0.504019
2	57	0.507822
3	35	0.487613
4	1	0.490718

Evaluation

[68]:
```
prediction_per_industrie.Valeur_accuracy.mean()
```

[68]:
```
0.49817306550826923
```

Avec cette seconde approche, nous obtenons en moyenne 49.81% d'accuracy. C'est légèrement moins bien qu'avec le modèle général, mais certaines industries ont d'excellents résultats, avec jusqu'à 54% d'accuracy.

6.4 VI.4) Modèle mixte : général et par industrie

Nous avions remarqué que, dans le modèle par industrie, certaines industries avaient de très bons résultats, tandis que d'autres réagissaient mieux au modèle général. Dans cette dernière approche, nous essayons de mixer nos deux méthodes en gardant un modèle individuel pour les industries ayant une bonne accuracy seule, et en utilisant le modèle général pour les autres.

[69]:
```
Best_model_Mixte = Best_model.copy()
```

Nous avons obtenu 50.2% d'accuracy avec le modèle général. Nous utilisons ce résultat comme seuil. Toutes les industries ayant une meilleure accuracy seule garde leur model, pour les autres, nous prenons le modèle général.

[70]:
```
Best_model_Mixte.loc[Best_model_Mixte['Performance en %'] < 50.2,'Model'] =␣
↪'Model General'
```

[71]:
```
Best_model_Mixte.loc[Best_model_Mixte.Model == 'Model General','Model_code'] =␣
↪'model_général'
```

[72]:
```
Best_model_Mixte.head()
```

[72]:

	Industrie	Performance en %	Model	Model_code
0	0	51.154297	Logistic Regression	LogisticRegression(C=1.0, class_weight=None, d…
1	1	46.143883	Model General	model_général
2	2	51.294754	Logistic Regression	LogisticRegression(C=1.0, class_weight=None, d…
3	3	52.608616	Logistic Regression	LogisticRegression(C=1.0, class_weight=None, d…
4	4	52.854446	Logistic Regression	LogisticRegression(C=1.0, class_weight=None, d…

6.4.1 VI.4.A) Entrainement des modèles

Le modèle général est déjà entrainé (sur toutes les données). Les autres modèles ont déjà été entrainés dans l'approche précédente. Il ne reste plus qu'à prédire !

6.4.2 VI.4.B) Prédiction du Test Set avec le modèle mixte

Prédiction par Industrie

[73]:
```
def get_data_per_industrie_test(ind):
    data = df_test[df_test.INDUSTRY == ind]
    X = data.loc[:,['RET_1', 'VOLUME_1', 'RET_2', 'VOLUME_2', 'RET_3',
                    'VOLUME_3', 'RET_4', 'VOLUME_4', 'RET_5', 'VOLUME_5',
                    'Mean_RET1_by_Date_Sector', 'Mean_Vol1_by_Date_Sector',
                    'Mean_RET1_by_Date_SubInd']]

    X = (X-X.max())/(X.max() - X.min())
    y = data.RET
    return X, y
```

[78]:
```
list_industrie = []
valeur_accuracy = []
for ind in df_test.INDUSTRY.unique():
    print("Prédiction de l'industrie : ",ind)
    list_industrie.append(ind)
    X,y = get_data_per_industrie_test(ind)

    try:
        name_model = Best_model_Mixte.loc[Best_model_Mixte.Industrie == ind,␣
↪'Model'].values[0]

        if name_model != 'Model General':

            model = Best_model_Mixte.loc[Best_model_Mixte.Industrie == ind,␣
↪'Model_code'].values[0]
            y_pred = model.predict(X)
            valeur_accuracy.append(accuracy_score(y,y_pred))

        else:
            y_pred = model_général.predict(X)
            valeur_accuracy.append(accuracy_score(y,y_pred))
    except:
        y_pred = model_général.predict(X)
        valeur_accuracy.append(accuracy_score(y,y_pred))
```

```
Prédiction de l'industrie :  37
Prédiction de l'industrie :  15
Prédiction de l'industrie :  57
```

```
Prédiction de l'industrie :  35
Prédiction de l'industrie :  1
Prédiction de l'industrie :  2
Prédiction de l'industrie :  10
Prédiction de l'industrie :  36
Prédiction de l'industrie :  18
Prédiction de l'industrie :  13
Prédiction de l'industrie :  48
Prédiction de l'industrie :  53
Prédiction de l'industrie :  29
Prédiction de l'industrie :  54
Prédiction de l'industrie :  6
Prédiction de l'industrie :  50
Prédiction de l'industrie :  70
Prédiction de l'industrie :  31
Prédiction de l'industrie :  41
Prédiction de l'industrie :  11
Prédiction de l'industrie :  14
Prédiction de l'industrie :  0
Prédiction de l'industrie :  52
Prédiction de l'industrie :  22
Prédiction de l'industrie :  3
Prédiction de l'industrie :  26
Prédiction de l'industrie :  16
Prédiction de l'industrie :  56
Prédiction de l'industrie :  59
Prédiction de l'industrie :  71
Prédiction de l'industrie :  44
Prédiction de l'industrie :  62
Prédiction de l'industrie :  64
Prédiction de l'industrie :  47
Prédiction de l'industrie :  58
Prédiction de l'industrie :  40
Prédiction de l'industrie :  25
Prédiction de l'industrie :  42
Prédiction de l'industrie :  33
Prédiction de l'industrie :  43
Prédiction de l'industrie :  46
Prédiction de l'industrie :  45
Prédiction de l'industrie :  60
Prédiction de l'industrie :  23
Prédiction de l'industrie :  49
Prédiction de l'industrie :  8
Prédiction de l'industrie :  55
Prédiction de l'industrie :  28
Prédiction de l'industrie :  17
Prédiction de l'industrie :  27
Prédiction de l'industrie :  32
```

```
Prédiction de l'industrie :  63
Prédiction de l'industrie :  39
Prédiction de l'industrie :  19
Prédiction de l'industrie :  34
Prédiction de l'industrie :  30
Prédiction de l'industrie :  68
Prédiction de l'industrie :  20
Prédiction de l'industrie :  38
Prédiction de l'industrie :  7
Prédiction de l'industrie :  69
Prédiction de l'industrie :  5
Prédiction de l'industrie :  9
Prédiction de l'industrie :  24
Prédiction de l'industrie :  4
Prédiction de l'industrie :  72
Prédiction de l'industrie :  12
Prédiction de l'industrie :  61
Prédiction de l'industrie :  21
Prédiction de l'industrie :  51
Prédiction de l'industrie :  74
Prédiction de l'industrie :  73
Prédiction de l'industrie :  66
Prédiction de l'industrie :  65
Prédiction de l'industrie :  67
```

[79]:
```
prediction_mixte = pd.DataFrame()
prediction_mixte['Industrie'] = list_industrie
prediction_mixte['Valeur_accuracy'] = valeur_accuracy
```

[80]:
```
prediction_mixte
```

[80]:

	Industrie	Valeur_accuracy
0	37	0.436314
1	15	0.506431
2	57	0.500602
3	35	0.487613
4	1	0.510333
5	2	0.494260
6	10	0.503956
7	36	0.504241
8	18	0.493865
9	13	0.501796
10	48	0.507830
11	53	0.489914
12	29	0.482468
13	54	0.513064
14	6	0.508753

15	50	0.493173
16	70	0.484414
17	31	0.523671
18	41	0.510544
19	11	0.481872
20	14	0.471556
21	0	0.501666
22	52	0.492767
23	22	0.517783
24	3	0.497474
25	26	0.503289
26	16	0.490847
27	56	0.499268
28	59	0.496918
29	71	0.540070
..	…	…
45	8	0.512809
46	55	0.509960
47	28	0.492812
48	17	0.488487
49	27	0.500768
50	32	0.523329
51	63	0.483841
52	39	0.479282
53	19	0.478386
54	34	0.489309
55	30	0.472669
56	68	0.495887
57	20	0.500000
58	38	0.505000
59	7	0.503817
60	69	0.482143
61	5	0.524096
62	9	0.484988
63	24	0.515044
64	4	0.477143
65	72	0.523517
66	12	0.500000
67	61	0.500000
68	21	0.512195
69	51	0.485938
70	74	0.502326
71	73	0.501312
72	66	0.528053
73	65	0.500911
74	67	0.519608

```
[75 rows x 2 columns]
```

[81]:
```
prediction_mixte.Valeur_accuracy.mean()
```

[81]:
```
0.4985709968931686
```

Cette approche s'avère être la moins précise de toutes.

6.5 VI.5) Conclusions

Ainsi, pour répondre à la problématique de ce challenge, nous avons : - Filtré les colonnes pour ne conserver que : - Les volumes sur les 5 derniers jours - Les retours sur les 5 derniers jours

- Créé trois nouvelles features, à savoir:
  - 'Mean_RET1_by_Date_Sector' : la moyenne du Retour à j-1 conditionnellement à la Date et au Secteur
  - 'Mean_Vol1_by_Date_Sector' : la moyenne du Volume à j-1 conditionnellement à la Date et au Secteur
  - 'Mean_RET1_by_Date_SubInd' : la moyenne du Retour à j-1 conditionnellement à la Date et à la Sous Industrie
- Essayé trois approches d'apprentissage, à savoir :
  - Un modèle général entrainé sur toutes les données sans distinction. Nous y avons appliqué une légère otpimisation des hyper paramètres et obtenus une accuracy de 50.79%
  - Un modèle par Industrie, sans pouvoir y effectuer d'optimisation des hyper paramètres, par limitation de la puissance de calcule. Nous obtenons une accuracy de 49.81%
  - Enfin, un modèle mixte, en appliquant le modèle général sur les industries dont le modèle propre donnait une très faible accuracy. Nous obtenons une accuracy de 49.85%

La méthode utilisant un classifieur par Industrie a montré d'excellents résultats dans certains cas et mérite d'être approfondie. De plus, d'autres variables aurait pu être générée pour apporter encore plus d'informations aux modèles.

[ ]:
```

```