**Implications for asset returns in the implied volatility skew**

Beaucoup d’études ont montrés que l’information contenue dans les prix des options ont des implications sur les rendements et la volatilité du marché equity. => ici on s’intéresse à l’information contenue dans la volatilité implicite skew et sa relation avec les rendements futurs.

Actions avec grandes différences pour sa volatilité implicite entre call et put ont souvent des futurs rendements mensuels plus grands.

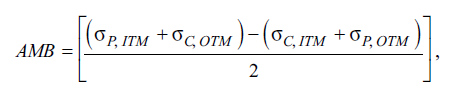
Data provenant d’OptionMetrics

Echantillon de 4 161 entreprises (pas utilities ni Financial) en réalisant 5 mesures sur sa volatilité implicite skew sur le dernier jour de trading du mois. (et share > 5$ and options use dhad to trade for 0.25$ or more) Objectif : regarder sa relation entre ces 5 mesures de skew et ses monthly returns.

3 portfolios :

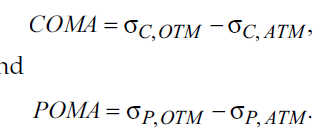
* 229 ADRs (titre de propriété d’actions d’une entreprise étrangère sur le marché US)
* 17 ETFs international
* 477 US ETFs

**First measure** : “ above minus below “ (AMB): represents the difference between the mean implied volatility of the option pair whose strike prices are above the current underlying price and the mean implied volatility of the option pair whose strike prices are below the current, underlying price.



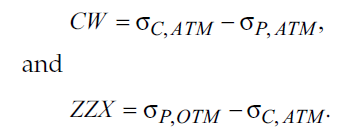
On prend des options avec des niveau de strike/spot qui ne dépasse pas 0.95 pour OTM put et ITM call. On prend des options avec un strike/spot ratio qui dépasse 1.05 pour OTM call et ITM put. Si c’est en dessous de 0.8 ou 1.2 on supprime l’observation. Pour que l’AMB soit calculé les 4 doivent exister.

**Second and third measures:** capture la diff entre les volatilities OTM and ATM.

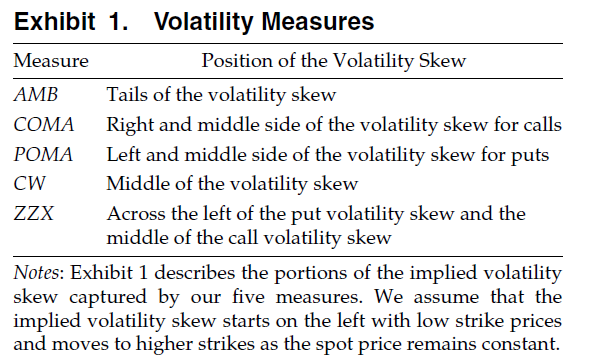


Strike to spot ratio proche de 1 (entre 0.95 et 1.05).

**Fourth and fifth measures :**



The five measures should capture all information contained in the cross section of implied volatilities. ZZX + CW = POMA

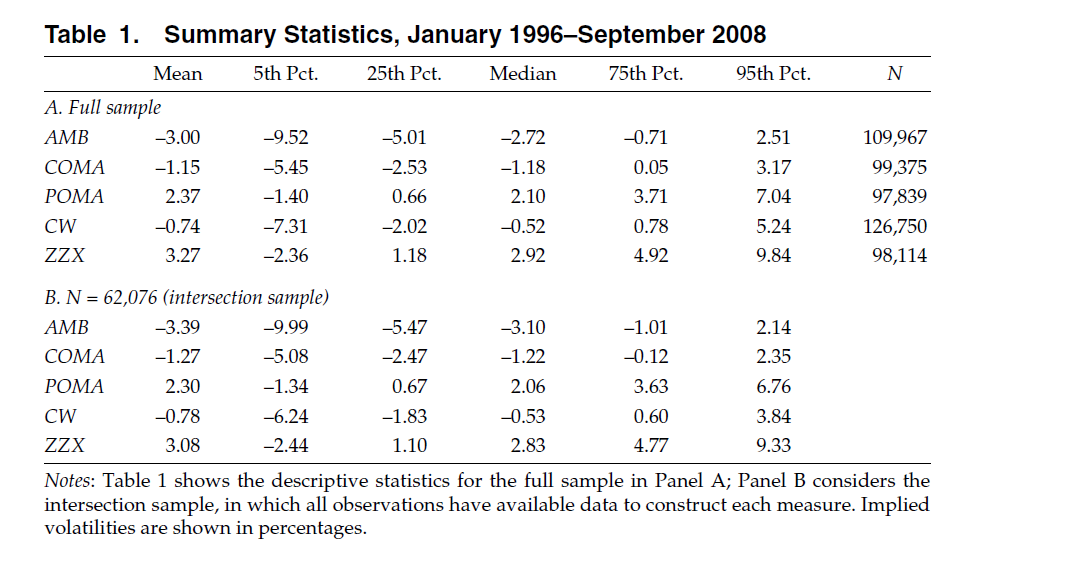


Après chaque mesure, on classe les compagnies en quantile (en fonction des mesures) et on calcule les returns du mois suivant. On rééquilibre les portefeuilles le dernier jour du mois suivant. On prend des options qui expirent dans deux mois.

Long-high quantile et short-low quantile portfolios for each of the measures. On calcule les rendements de chaque quantile ainsi que Fama French alpha.

On a utilisé la procédure Fama-Macbeth pour faire des tests de signifiance entre nos mesures et des données entreprises pour vérifier qu’il y a un lien significatif.

**Table 1 :**

****

Sample A : contient toutes les compagnies dès lors que nous pouvons calculer une mesure

Sample B : contient seulement les compagnies où les 5 mesures sont possibles à calculer.

Relation positive en CW et future returns. Et relation négative entre ZZX et future returns.

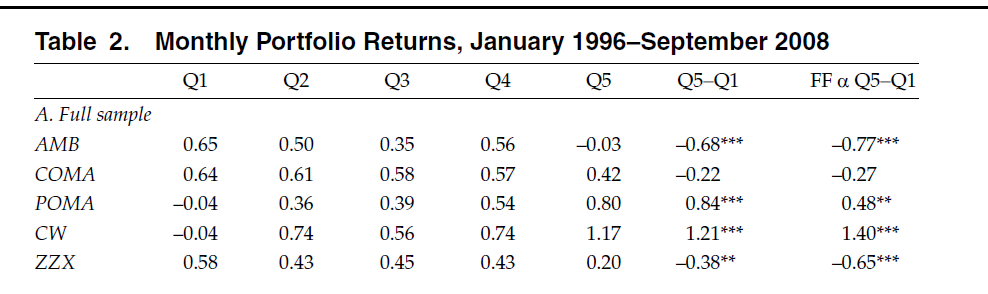
Relation négative entre AMB et future returns.

**Table 2 :**

Il contient les rendements mensuels de chaque portfolio (qui correspondent à chaque quantile des mesures). La dernière colonne est une stratégie de long les hauts quantiles et short les bas quantiles.

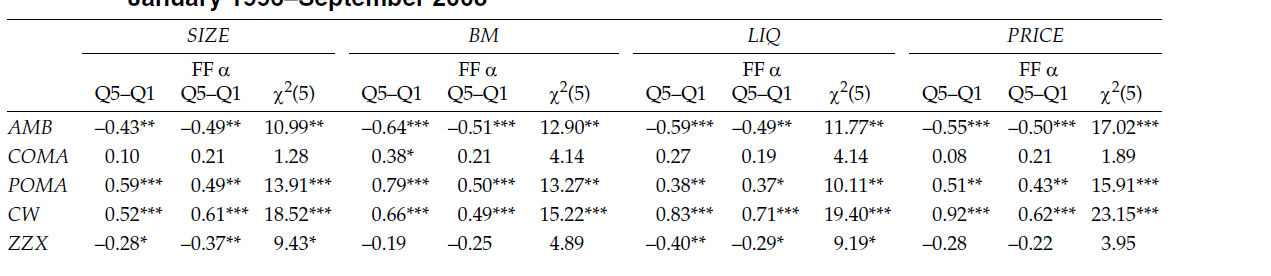
Il semblerait que plus on a un skew negatif (POMA élevé) plus on s’attends à de fort future return.

The right-hand side of the implied volatility skew contains little information.

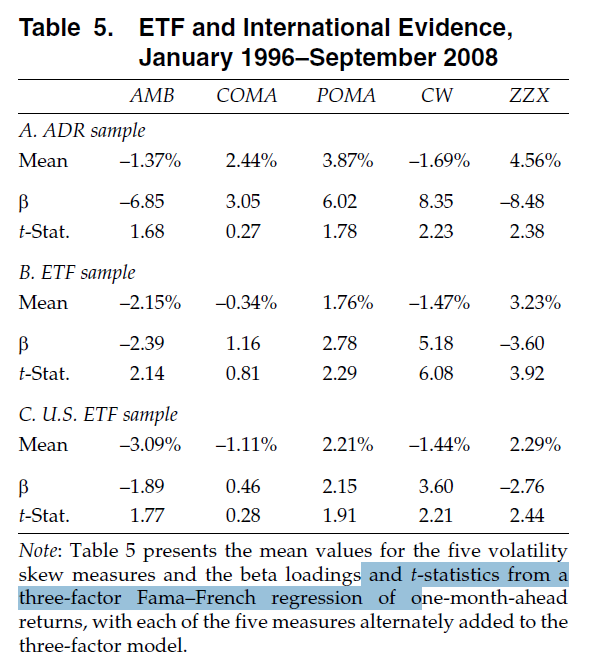


**Characteristic controls :** on a envie de savoir si certaines caractéristiques d’entreprises sont corrélées avec la volatilité implicite. On classe les entreprises dans des quantiles en fonction de leur SIZE,…etc et dans chaque quantile on reclasse les compagnies en fonction de la mesure skew.

**Table 3** : equally weighted monthly returns for Q5 – Q1 portfolio and Fama French alphas pour les cinq measures de skew par categories.



**Table 5 :** on souhaite maintenant à partir de portefeuilles (etf) étrangers et US voir s’il y a un lien entre vol implicite et perf future.



Voir à partir d’un modèle Fama French à 5 facteurs basée sur les mesures de skew si on arrive à prédire.

First,

future underlying returns are positively linked to

the higher differences in volatilities between ATM

calls and puts. Second, higher returns are positively

related to more-negative skews as measured on the

left-hand side of the volatility skew, between the

OTM and ATM puts.

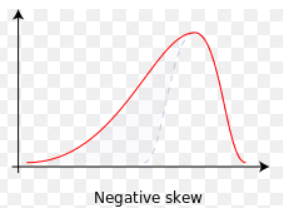
**Méthodologie : INDEX SKEW**

Le Skew correspond à une mesure globale de la pente de la courbe de volatilité implicité. Skew est calculé sur les prix des options du S&P500 avec une méthode similaire du VIX.

Pourquoi il n’y a plus de smile de volatilité ? Les investisseurs pricent plus cher les puts avec des bas strikes que les calls avec des strikes hauts.

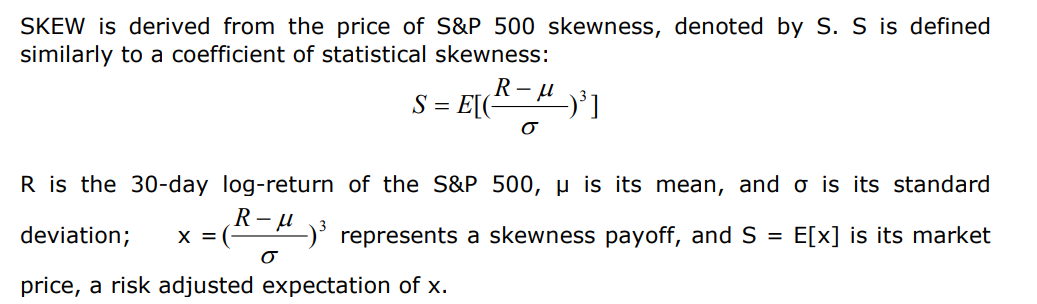
* S’explique par la crise de 1987, qui a sensibilisé le marché a de grands sauts baissiers sur le S&P500. La distribution des log-returns du S&P500 n’est pas normale s’il y a des grands sauts dans les rendements (=> sauts augmentent le poids des queues de distribution et les sauts asymétriques faussent la distribution).

Risques de queues plus importantes sur la distribution du S&P500 que sur la distribution normale et la distribution du S&P500 a une negative skew.



VIX conçu pour mesurer attente du marché en matière de vol sur 30 jours sur les options du S&P500, cependant : il ne prend peu en compte les risques de queue de distribution et se base sur une distribution normale.

SKEW mesure un risque implicite de rendements futurs réalisant un comportement extrême.







Plus on est au-dessus de 100, plus le marché des options perçoit un gros risque de rendements extrêmes futures.

Lorsque skew = 100, la distribution perçue du S&P500 est normale, donc faible proba d’avoir des returns extrêmes.

**CAPCA (Cross-asset put call ratio) et fund sensitivities towards equities:**

CAPCA ratio est dérivé du marché des options, comme une combinaison de put call ratio de différents indice equity et US Treasuries pour jauger le positionnement dans les marchés d’options.

**Put call ratio is calculated by dividing the number of traded put options by the number of traded call options.**

Un grand put call ratio dans l’equity => demande de protection (put) dépasse le nombre de paris haussiers (sentiment de risque baissier)

Un grand put call ratio dans le Fixed => un plus grand nombre de paris baissier (donc un sentiment de risque haussier).

Plus ratio put call est élevé pour les actions et faible pour le FI, plus les investisseurs seront déjà plus baissiers. => ainsi les actifs à risque devraient surperformer (indicateur à contre-courant)

Sur les graphes :

=> hautes valeurs => bullish risk sentiment

=> faibles valeurs => bearish risk sentiment

* Envoyer un signal à contre-courant d’achat lorsque la différence entre le put call ratio d’aujourd’hui et il y a 3 mois tombe dans le 10th quantile sur une fenêtre de 5 ans.
* Envoyer un signal à contre-courant d’achat lorsque la différence entre le put call ratio d’aujourd’hui et il y a 3 mois tombe dans le 90th quantile sur une fenêtre de 5 ans.

Scoring :

# -\*- coding: utf-8 -\*-

"""

Created on Wed Apr 5 10:11:37 2023

@author: amorlat

"""

#pilotage.py

import numpy as np

import pandas as pd

import xlwings as xw

import os

import seaborn as sns

#from openpyxl import load\_workbook

import sys

import matplotlib.pyplot as plt

import sklearn

from sklearn.decomposition import PCA

import plotly.express as px

#fonction pour calculer le score en souhaitant maximiser ce critère

def high\_score(critere,df,percent\_high,percent\_low,i):

rep=0

if df[critere][i]>np.percentile(np.array(df[critere].dropna()),percent\_high):

rep=1

if df[critere][i]<np.percentile(np.array(df[critere].dropna()),percent\_low):

rep=-1

if df[critere][i]>np.percentile(np.array(df[critere].dropna()),percent\_low) and df[critere][i]<np.percentile(np.array(df[critere].dropna()),percent\_high):

rep=0

return rep

#fonction pour calculer le score en souhaitant minimiser ce critère

def low\_score(critere, df,percent\_high,percent\_low,i):

rep=0

if df[critere][i]>np.percentile(np.array(df[critere].dropna()),percent\_high):

rep=-1

if df[critere][i]<np.percentile(np.array(df[critere].dropna()),percent\_low):

rep=1

if df[critere][i]>np.percentile(np.array(df[critere].dropna()),percent\_low) and df[critere][i]<np.percentile(np.array(df[critere].dropna()),percent\_high):

rep=0

return rep

def normalize(vec):

moy=np.mean(vec.dropna())

stda=np.std(vec.dropna())

rep=[]

if pd.isna(stda):

for i in vec:

rep.append(0)

elif stda ==0:

for i in vec:

rep.append(0)

else :

for i in vec:

rep.append((i-moy)/stda)

return rep

#fonction qui regarde si la clé en paramètre appartient à l'ensemble des clés du dictionnaire en paramètre

def dicMemberCheck(key, dicObj):

if key in dicObj:

return True

else:

return False

def calculate\_score(df, percent\_high, percent\_low, selection\_criteres=None, coeff=None, new\_criteria=None, score\_type=None):

n = len(df["ISIN"])

df["Score " + score\_type + str(percent\_high)] = 0.0

for i in range(n):

total\_score = 0

j = 0

for critere in selection\_criteres:

if dicMemberCheck(critere, new\_criteria):

if new\_criteria[critere] == 'L':

total\_score += coeff[j] \* low\_score(critere, df, percent\_high, percent\_low, i)

elif new\_criteria[critere] == 'H':

total\_score += coeff[j] \* high\_score(critere, df, percent\_high, percent\_low, i)

else:

print("Erreur pour le critere", score\_type, critere, "l'objectif n'est pas L ou H")

j += 1

rep = total\_score / sum(coeff)

df.loc[i, "Score " + score\_type + str(percent\_high)] = rep

def feuille\_existe(classeur, nom\_feuille):

for feuille in classeur.sheets:

if feuille.name == nom\_feuille:

return True

return False

def CreaFeuille(wb,nom\_feuille):

if feuille\_existe(wb, nom\_feuille):

resultats = wb.sheets[nom\_feuille]

else:

resultats = wb.sheets.add(nom\_feuille) #ajout d'une feuille

return resultats

def SuppFeuille(wb,sheet\_name):

if sheet\_name in [sheet.name for sheet in wb.sheets]:

sheet\_to\_delete = wb.sheets[sheet\_name]

sheet\_to\_delete.delete()

def start(nomExcel,chemin\_init,nomFichier,SeuilMin,SeuilMax,SeuilPas,PoidPerf,PoidRisk,PoidESG,SelectedCategory,ValAdjustFees,Perso = False,list\_criteria\_Perf = None, list\_criteria\_Risk = None, list\_criteria\_ESG = None, coeff\_perf = None, coeff\_risk = None, coeff\_esg = None, dic\_unknow\_criteria\_perf = None, dic\_unknow\_criteria\_risk = None, dic\_unknow\_criteria\_esg = None, graphique\_option = False):

print(chemin\_init,nomFichier,SeuilMin,SeuilMax,SeuilPas,PoidPerf,PoidRisk,PoidESG,SelectedCategory,ValAdjustFees, coeff\_perf, coeff\_risk, coeff\_esg, dic\_unknow\_criteria\_perf, dic\_unknow\_criteria\_risk, dic\_unknow\_criteria\_esg, graphique\_option)

chemin\_fichier = os.path.join(chemin\_init+"\Extraction\_Morningstar", nomFichier)

print("Voici le chemin de nos fichiers",chemin\_fichier)

app2 = xw.App(visible=False) #ouvre la feuille sans l'afficher à l'utilisateur

try:

wb2 = xw.Book(chemin\_fichier)

except Exception as e:

print("Une erreur s'est produite lors de l'ouverture du fichier Excel, fichier inconnu ")

# Sélection de la feuille des fonds

#if nomFichier == 'emd\_global.xlsx':

# feuille = wb2.sheets['EMD Global (Test) + ESG']

#elif nomFichier == 'hy\_europe.xlsx':

# feuille = wb2.sheets['HY Europe (test)']

#else :

feuille = wb2.sheets[0]

#sélection des sous catégories des fonds

plage\_donnees = feuille.range('A1').expand()

df\_score = plage\_donnees.options(pd.DataFrame, index=False, header=True).value

if SelectedCategory == 'Toutes Catégories' or SelectedCategory == '':

df\_score = df\_score

else:

try:

df\_score = df\_score.loc[df\_score['Morningstar Category'] == SelectedCategory]

df\_score = df\_score.reset\_index(drop=True)

except KeyError:

print("La colonne 'Morningstar Category' n'existe pas dans le DataFrame.")

#fermeture du classeur et de l'application xlwings

wb2.close()

app2.quit()

ValAdjustFees = ValAdjustFees/10

if(Perso == False):

main(nomExcel,chemin\_init,df\_score,SeuilMin,SeuilMax,SeuilPas,PoidPerf,PoidRisk,PoidESG,ValAdjustFees)

else:

main(nomExcel,chemin\_init,df\_score,SeuilMin,SeuilMax,SeuilPas,PoidPerf,PoidRisk,PoidESG,ValAdjustFees, True,list\_criteria\_Perf, list\_criteria\_Risk, list\_criteria\_ESG, coeff\_perf, coeff\_risk, coeff\_esg, dic\_unknow\_criteria\_perf, dic\_unknow\_criteria\_risk, dic\_unknow\_criteria\_esg, graphique\_option)

def main(nomExcel,chemin\_init,df\_data,SeuilMin,SeuilMax,SeuilPas,PoidPerf,PoidRisk,PoidESG,ValAdjustFees, Perso = False, list\_criteria\_Perf = None, list\_criteria\_Risk = None, list\_criteria\_ESG = None, coeff\_perf = None, coeff\_risk = None, coeff\_esg = None, dic\_unknow\_criteria\_perf = None, dic\_unknow\_criteria\_risk = None, dic\_unknow\_criteria\_esg = None, graphique\_option=False):

cwd = chemin\_init

chemin\_pilotage = nomExcel

app = xw.App(visible=False)

wb = xw.Book(chemin\_pilotage)

SuppFeuille(wb, "Données")

SuppFeuille(wb, "Graphiques")

SuppFeuille(wb, "Résultats")

sheet = wb.sheets['Classements']

#trouver la dernière ligne non vide dans la colonne A (ou une autre colonne de référence)

last\_row = sheet.range('A' + str(sheet.cells.last\_cell.row)).end('up').row

#effacer le contenu de toutes les cellules sauf la première ligne

sheet.range(f'A2:Z{last\_row}').clear\_contents()

#feuille Données

feuille\_data = CreaFeuille(wb, 'Données')

feuille\_data.range('A1').value = df\_data

df\_score = df\_data.copy()

#y=pd.DataFrame()

nom\_fichier = nomExcel

feuille = 'Sélection des critères'

wb = xw.Book(nom\_fichier)

feuille = wb.sheets[feuille]

error\_message\_2 = ""

error\_0\_criteria = False

#Lever les erreurs lorsque le paramètre n'existe pas dans le fichier

#Cas des erreurs dans perf

criteria\_errors = [] #contiendra l'ensemble des critères inexistants des trois catégories

new\_criteria\_perf = dict() #dictionnaire qui contient les critères de perf dont on ne connais pas la façon de calculer le score mais bien présent dans le dataset

if(dic\_unknow\_criteria\_perf is not None): #si il existe des critères non reconnus

for key, value in dic\_unknow\_criteria\_perf.items(): #on parcourt le dictionnaire et si une clé n'est pas présente dans notre dataset, je la supprime ainsi que son coefficient dans le tableau des coefficients

if key not in df\_score.columns or (key in df\_score.columns and (df\_score[key].nunique() < 2)) :

criteria\_errors.append(key)

indice = list\_criteria\_Perf.index(key)

list\_criteria\_Perf.remove(key)

del coeff\_perf[indice]

else:

print(key, "est valide")

new\_criteria\_perf[key] = value

if(len(new\_criteria\_perf)==0):

print("ERREUR, la list perf est vide, mettez au moins un critère valide dans perf")

error\_message\_2 = "la list perf ne comporte aucun critère "

error\_0\_criteria = True

#Cas des erreurs dans risques

new\_criteria\_risk = dict() #dictionnaire qui contient les critères de risk dont on ne connais pas la façon de calculer le score mais bien présent dans le dataset

if(dic\_unknow\_criteria\_risk is not None): #si il existe des critères non reconnus

for key, value in dic\_unknow\_criteria\_risk.items():#on parcourt le dictionnaire et si une clé n'est pas présente dans notre dataset, je la supprime ainsi que son coefficient dans le tableau des coefficients

if key not in df\_score.columns or (key in df\_score.columns and (df\_score[key].nunique() < 2)) :

criteria\_errors.append(key)

indice = list\_criteria\_Risk.index(key)

list\_criteria\_Risk.remove(key)

del coeff\_risk[indice]

else:

print(key, "est valide")

new\_criteria\_risk[key] = value

if(len(new\_criteria\_risk)==0):

print("ERREUR, la list risk est vide, mettez au moins un critère valide dans risk")

error\_message\_2 = error\_message\_2 + ", la list risk ne comporte aucun critère "

error\_0\_criteria = True

#Cas des erreurs dans ESG

new\_criteria\_esg = dict() #dictionnaire qui contient les critères d'esg dont on ne connais pas la façon de calculer le score mais bien présent dans le dataset

if(dic\_unknow\_criteria\_esg is not None): #si il existe des critères non reconnus

for key, value in dic\_unknow\_criteria\_esg.items(): #on parcourt le dictionnaire et si une clé n'est pas présente dans notre dataset, je la supprime ainsi que son coefficient dans le tableau des coefficients

if key not in df\_score.columns or (key in df\_score.columns and (df\_score[key].nunique() < 2)) :

criteria\_errors.append(key)

indice = list\_criteria\_ESG.index(key)

list\_criteria\_ESG.remove(key)

del coeff\_esg[indice]

else:

print(key, "est valide")

new\_criteria\_esg[key] = value

if(len(new\_criteria\_esg)==0):

print("ERREUR, la list esg est vide, mettez au moins un critère valide dans esg")

error\_message\_2 = error\_message\_2 + ", la list esg ne comporte aucun critère"

error\_0\_criteria = True

error\_message = "Les critères suivants ne sont pas présents dans le fichier Morning star: " + ", ".join(criteria\_errors)

feuille['M6'].value = error\_message #on affiche les erreurs dans la case M6

feuille['M14'].value = error\_message\_2

if(error\_0\_criteria== False):

tab\_column= []

tab\_column.extend(new\_criteria\_perf.keys())

tab\_column.extend(new\_criteria\_risk.keys())

tab\_column.extend(new\_criteria\_esg.keys())

x=pd.DataFrame()

for i in tab\_column:

x[i]=normalize(df\_score[i])

x.fillna(0,inplace=True)

for i in np.arange(SeuilMin,SeuilMax,SeuilPas): #on parcours nos seuils

if Perso :

calculate\_score(df\_score, i, 100-i, list\_criteria\_Perf, coeff\_perf, new\_criteria\_perf, "Perf")

calculate\_score(df\_score, i, 100-i, list\_criteria\_Risk, coeff\_risk, new\_criteria\_risk, "Risks")

calculate\_score(df\_score, i, 100-i, list\_criteria\_ESG, coeff\_esg, new\_criteria\_esg, "ESG")

noms = df\_score["Name"]

df\_score=df\_score.set\_index('Name')

#rajoute des colonnes pour calculer les moyennes des scores de chaque dataset

df\_score["Score\_Perf\_Moy"] = 0.0

df\_score["Score\_Risks\_Moy"] = 0.0

df\_score["Score\_ESG\_Moy"] = 0.0

for i in df\_score.index:

score\_perf\_columns = [col for col in df\_score.columns if col.startswith("Score Perf")]

df\_score["Score\_Perf\_Moy"] = df\_score[score\_perf\_columns].mean(axis=1)

score\_risks\_columns = [col for col in df\_score.columns if col.startswith("Score Risks")]

df\_score["Score\_Risks\_Moy"] = df\_score[score\_risks\_columns].mean(axis=1)

score\_esg\_columns = [col for col in df\_score.columns if col.startswith("Score ESG")]

df\_score["Score\_ESG\_Moy"] = df\_score[score\_esg\_columns].mean(axis=1)

######### Calcul of R^2 with Alpha

def R2\_Alpha\_Calc(abscisse, ordonnee):

Alpha\_r\_df=df\_score[[ordonnee,abscisse,"Score Moyenne"]].dropna()

Alpha\_r\_df["Score quali"] = ''

Alpha\_r\_df.index = np.arange(0, len(Alpha\_r\_df), 1)

for i in range(len(Alpha\_r\_df["Score Moyenne"])):

score\_moyenne = Alpha\_r\_df["Score Moyenne"][i]

score\_percentiles = np.array(Alpha\_r\_df["Score Moyenne"])

if score\_moyenne >= np.percentile(score\_percentiles, 75):

Alpha\_r\_df.loc[i, "Score quali"] = "Très bon"

elif score\_moyenne < np.percentile(score\_percentiles, 75) and score\_moyenne >= np.percentile(score\_percentiles, 50):

Alpha\_r\_df.loc[i, "Score quali"] = "Bon"

elif score\_moyenne < np.percentile(score\_percentiles, 50) and score\_moyenne >= np.percentile(score\_percentiles, 25):

Alpha\_r\_df.loc[i, "Score quali"] = "Moyen"

elif score\_moyenne < np.percentile(score\_percentiles, 25):

Alpha\_r\_df.loc[i, "Score quali"] = "Mauvais"

return Alpha\_r\_df

######### Plot of R^2 with Alpha

def R2\_Alpha\_Fig(Alpha\_r\_df, abscisse, ordonnee):

name = abscisse+"\_" +ordonnee+".png"

print(abscisse)

print(ordonnee)

fig = plt.figure(figsize = (8,8))

ax = fig.add\_subplot(1,1,1)

ax.set\_xlabel(abscisse, fontsize = 15)

ax.set\_ylabel(ordonnee, fontsize = 15)

targets = ['Très bon','Bon', 'Moyen', 'Mauvais']

colors = ['r', 'g', 'black','blue']

for target, color in zip(targets,colors):

indicesToKeep = Alpha\_r\_df["Score quali"] == target

ax.scatter(Alpha\_r\_df.loc[indicesToKeep, abscisse]

, Alpha\_r\_df.loc[indicesToKeep, ordonnee]

, c = color

, s = 50)

for i in range(Alpha\_r\_df.shape[0]):

plt.text(x=Alpha\_r\_df[abscisse][i],y=Alpha\_r\_df[ordonnee][i],s=str(i), fontdict=dict(color='black',size=10))

ax.legend(targets)

ax.grid()

plt.savefig(os.path.join(cwd+"\Graphiques",name))

######### Calcul of PCA

def PCA\_Calc():

pca = PCA(n\_components=2)

principalComponents = pca.fit\_transform(x)

principalDf = pd.DataFrame(data=principalComponents, columns=['principal component 1', 'principal component 2'])

df\_score2 = df\_score.copy(deep=True)

df\_score2["Score quali"] = ''

df\_score2.index = np.arange(0, len(df\_score2), 1)

for i in range(0, len(df\_score2["Score Moyenne"])):

if df\_score2.loc[i, "Score Moyenne"] >= np.percentile(np.array(df\_score2["Score Moyenne"]), 75):

df\_score2.loc[i, "Score quali"] = "Très bon"

elif df\_score2.loc[i, "Score Moyenne"] < np.percentile(np.array(df\_score2["Score Moyenne"]), 75) and df\_score2.loc[i, "Score Moyenne"] >= np.percentile(np.array(df\_score2["Score Moyenne"]), 50):

df\_score2.loc[i, "Score quali"] = "Bon"

elif df\_score2.loc[i, "Score Moyenne"] < np.percentile(np.array(df\_score2["Score Moyenne"]), 50) and df\_score2.loc[i, "Score Moyenne"] >= np.percentile(np.array(df\_score2["Score Moyenne"]), 25):

df\_score2.loc[i, "Score quali"] = "Moyen"

else:

df\_score2.loc[i, "Score quali"] = "Mauvais"

finalDf=pd.concat([principalDf,df\_score2[["Score quali"]]],axis=1)

#print(pca.explained\_variance\_ratio\_)

return(finalDf)

######### Figure of PCA

def PCA\_Fig(finalDf):

fig = plt.figure(figsize = (8,8))

ax = fig.add\_subplot(1,1,1)

ax.set\_xlabel('Principal Component 1', fontsize = 15)

ax.set\_ylabel('Principal Component 2', fontsize = 15)

ax.set\_title('2 component PCA', fontsize = 20)

targets = ['Très bon','Bon', 'Moyen', 'Mauvais']

colors = ['r', 'g', 'black','blue']

for target, color in zip(targets,colors):

indicesToKeep = finalDf['Score quali'] == target

ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']

, finalDf.loc[indicesToKeep, 'principal component 2']

, c = color

, s = 50)

ax.legend(targets)

ax.set\_xlim(-5,5)

ax.grid()

plt.savefig(os.path.join(cwd+"\Graphiques",'PCA\_2D.png'))

######### Calcul of Score\_Perf\_Moy with Score\_Risks\_Moy

def Perf\_Risk\_Calc():

Perf\_risk\_df=df\_score[["ISIN","Score\_Perf\_Moy","Score\_Risks\_Moy","Score Moyenne"]].copy()

Perf\_risk\_df.sort\_values(by=["Score Moyenne"],inplace=True,ascending=False)

#Version AM

Perf\_risk\_df["Score quali"] = ''

Perf\_risk\_df.index = np.arange(0, len(Perf\_risk\_df), 1)

for i in range(0, len(Perf\_risk\_df["Score Moyenne"])):

if Perf\_risk\_df.loc[i, "Score Moyenne"] >= np.percentile(np.array(Perf\_risk\_df["Score Moyenne"]), 75):

Perf\_risk\_df.loc[i, "Score quali"] = "Très bon"

elif Perf\_risk\_df.loc[i, "Score Moyenne"] < np.percentile(np.array(Perf\_risk\_df["Score Moyenne"]), 75) and Perf\_risk\_df.loc[i, "Score Moyenne"] >= np.percentile(np.array(Perf\_risk\_df["Score Moyenne"]), 50):

Perf\_risk\_df.loc[i, "Score quali"] = "Bon"

elif Perf\_risk\_df.loc[i, "Score Moyenne"] < np.percentile(np.array(Perf\_risk\_df["Score Moyenne"]), 50) and Perf\_risk\_df.loc[i, "Score Moyenne"] >= np.percentile(np.array(Perf\_risk\_df["Score Moyenne"]), 25):

Perf\_risk\_df.loc[i, "Score quali"] = "Moyen"

else:

Perf\_risk\_df.loc[i, "Score quali"] = "Mauvais"

return(Perf\_risk\_df)

######## Plot Score\_Perf\_Moy with Score\_Risks\_Moy

def Perf\_Risk\_Fig(Perf\_risk\_df):

fig = plt.figure(figsize = (8,8))

ax = fig.add\_subplot(1,1,1)

ax.set\_xlabel('Score\_Perf\_Moy', fontsize = 15)

ax.set\_ylabel('Score\_Risks\_Moy', fontsize = 15)

ax.set\_title('Scatter Plot 2D', fontsize = 20)

targets = ['Très bon','Bon', 'Moyen', 'Mauvais']

colors = ['r', 'g', 'black','blue']

for target, color in zip(targets,colors):

indicesToKeep = Perf\_risk\_df["Score quali"] == target

ax.scatter(Perf\_risk\_df.loc[indicesToKeep, 'Score\_Perf\_Moy']

, Perf\_risk\_df.loc[indicesToKeep, 'Score\_Risks\_Moy']

, c = color

, s = 50)

for i in range(Perf\_risk\_df.shape[0]):

plt.text(x=Perf\_risk\_df["Score\_Perf\_Moy"][i],y=Perf\_risk\_df["Score\_Risks\_Moy"][i],s=str(i), fontdict=dict(color='black',size=10))

ax.legend(targets)

ax.grid()

plt.savefig(os.path.join(cwd+"\Graphiques",'Score\_Perf\_Moy\_Score\_Risks\_Moy.png'))

############### Adding Score\_Perf\_Moy with Score\_ESG\_Moy

######### Calcul of Score\_Perf\_Moy with Score\_ESG\_Moy

def Perf\_ESG\_Calc():

Perf\_ESG\_df=df\_score[["ISIN","Score\_Perf\_Moy","Score\_ESG\_Moy","Score Moyenne"]].copy()

Perf\_ESG\_df.sort\_values(by=["Score Moyenne"],inplace=True,ascending=False)

#Version AM

Perf\_ESG\_df["Score quali"] = ''

Perf\_ESG\_df.index = np.arange(0, len(Perf\_ESG\_df), 1)

for i in range(0, len(Perf\_ESG\_df["Score Moyenne"])):

if Perf\_ESG\_df.loc[i, "Score Moyenne"] >= np.percentile(np.array(Perf\_ESG\_df["Score Moyenne"]), 75):

Perf\_ESG\_df.loc[i, "Score quali"] = "Très bon"

elif Perf\_ESG\_df.loc[i, "Score Moyenne"] < np.percentile(np.array(Perf\_ESG\_df["Score Moyenne"]), 75) and Perf\_ESG\_df.loc[i, "Score Moyenne"] >= np.percentile(np.array(Perf\_ESG\_df["Score Moyenne"]), 50):

Perf\_ESG\_df.loc[i, "Score quali"] = "Bon"

elif Perf\_ESG\_df.loc[i, "Score Moyenne"] < np.percentile(np.array(Perf\_ESG\_df["Score Moyenne"]), 50) and Perf\_ESG\_df.loc[i, "Score Moyenne"] >= np.percentile(np.array(Perf\_ESG\_df["Score Moyenne"]), 25):

Perf\_ESG\_df.loc[i, "Score quali"] = "Moyen"

else:

Perf\_ESG\_df.loc[i, "Score quali"] = "Mauvais"

return(Perf\_ESG\_df)

######## Plot Score\_Perf\_Moy with Score\_ESG\_Moy

def Perf\_ESG\_Fig(Perf\_ESG\_df):

fig = plt.figure(figsize = (8,8))

ax = fig.add\_subplot(1,1,1)

ax.set\_xlabel('Score\_Perf\_Moy', fontsize = 15)

ax.set\_ylabel('Score\_ESG\_Moy', fontsize = 15)

ax.set\_title('Scatter Plot 2D', fontsize = 20)

targets = ['Très bon','Bon', 'Moyen', 'Mauvais']

colors = ['r', 'g', 'black','blue']

for target, color in zip(targets,colors):

indicesToKeep = Perf\_ESG\_df["Score quali"] == target

ax.scatter(Perf\_ESG\_df.loc[indicesToKeep, 'Score\_Perf\_Moy']

, Perf\_ESG\_df.loc[indicesToKeep, 'Score\_ESG\_Moy']

, c = color

, s = 50)

for i in range(Perf\_ESG\_df.shape[0]):

plt.text(x=Perf\_ESG\_df["Score\_Perf\_Moy"][i],y=Perf\_ESG\_df["Score\_ESG\_Moy"][i],s=str(i), fontdict=dict(color='black',size=10))

ax.legend(targets)

ax.grid()

plt.savefig(os.path.join(cwd+"\Graphiques",'Score\_Perf\_Moy\_Score\_ESG\_Moy.png'))

############### End of Score\_Perf\_Moy with Score\_ESG\_Moy

def Perf\_fees\_Calc():

Perf\_fees\_df = df\_score[["ISIN", "Score\_Perf\_Moy", "Score\_Risks\_Moy", "Score Moyenne", "Management Fee (ex Distribution fees) Actual"]]

Perf\_fees\_df = Perf\_fees\_df.sort\_values(by=["Score Moyenne"], ascending=False)

Perf\_fees\_df.loc[:, "Fees Quali"] = ''

Perf\_fees\_df["Fees Quali"] = ''

for fund in Perf\_fees\_df.index:

if Perf\_fees\_df.loc[fund,"Management Fee (ex Distribution fees) Actual"] is not None:

fee = float(Perf\_fees\_df.loc[fund,"Management Fee (ex Distribution fees) Actual"])

else:

fee = np.nan

fee\_percentiles = np.array(Perf\_fees\_df["Management Fee (ex Distribution fees) Actual"].dropna(), dtype=float)

if np.isnan(fee):

Perf\_fees\_df.loc[fund, "Fees Quali"] = 'Non précisées'

elif fee >= np.percentile(fee\_percentiles, 75):

Perf\_fees\_df.loc[fund, "Fees Quali"] = "Élevées +"

elif fee >= np.percentile(fee\_percentiles, 50):

Perf\_fees\_df.loc[fund, "Fees Quali"] = "Élevées"

elif fee >= np.percentile(fee\_percentiles, 25):

Perf\_fees\_df.loc[fund, "Fees Quali"] = "Moyennes"

else:

Perf\_fees\_df.loc[fund, "Fees Quali"] = "Basses"

return(Perf\_fees\_df)

def Perf\_fees\_Fig(Perf\_fees\_df):

fig = plt.figure(figsize = (8,8))

ax = fig.add\_subplot(1,1,1)

ax.set\_xlabel('Score\_Perf\_Moy', fontsize = 15)

ax.set\_ylabel('Score\_Risks\_Moy', fontsize = 15)

ax.set\_title('Scatter Plot avec Fees', fontsize = 20)

targets = ['Élevées +','Élevées', 'Moyennes', 'Basses',"Non précisées"]

colors = ['red', 'green', 'black','blue',"grey"]

for target, color in zip(targets,colors):

indicesToKeep = Perf\_fees\_df["Fees Quali"] == target

ax.scatter(Perf\_fees\_df.loc[indicesToKeep, 'Score\_Perf\_Moy']

, Perf\_fees\_df.loc[indicesToKeep, 'Score\_Risks\_Moy']

, c = color

, s = 50)

for i in range(Perf\_fees\_df.shape[0]):

plt.text(x=Perf\_fees\_df["Score\_Perf\_Moy"][i],y=Perf\_fees\_df["Score\_Risks\_Moy"][i],s=str(i), fontdict=dict(color='black',size=10))

ax.legend(targets)

ax.grid()

plt.savefig(os.path.join(cwd+"\Graphiques",'Score\_Perf\_fees.png'))

w1 = PoidPerf

w2 = PoidRisk

w3 = PoidESG

print("poids de la performance ", w1)

print("poids des risques ",w2)

print("poids de l'ESG ",w3)

df\_score["Score Moyenne"]=w1\*df\_score["Score\_Perf\_Moy"]+w2\*df\_score["Score\_Risks\_Moy"]+w3\*df\_score["Score\_ESG\_Moy"]

df\_score.sort\_values(by=["Score Moyenne"],inplace=True,ascending=False) #mettre dans l'ordre de classement

coeff=20/(1+df\_score["Score Moyenne"].iloc[0])

tab\_notes=[]

for i in df\_score["Score Moyenne"]:

tab\_notes.append(round(coeff\*(1+i),2))

df\_score["Notes"]=tab\_notes

#feuille Résultats

resultats = CreaFeuille(wb,'Résultats')

#colonnes\_ajoutees = df\_score.iloc[:, -28:]

#création d'une liste de préfixes des colonnes à sélectionner

prefixes = ['ISIN','Notes','Score']

#sélection des colonnes en fonction des préfixes

selected\_columns = [col for prefix in prefixes for col in df\_score.filter(like=prefix).columns]

#ajout de la colonne 'Max Drawdown' à la liste des colonnes sélectionnées

#selected\_columns.append('Max Drawdown 2023-03-01 to 2024-02-29 Base Currency')

#extraction des colonnes sélectionnées à partir de df\_score

colonnes\_ajoutees = df\_score[selected\_columns]

#ecriture dans la Feuille de Resultats

resultats.range('A1').value = colonnes\_ajoutees

Perf\_fees\_df = Perf\_fees\_Calc()

############################# Feuille Graphique

if(graphique\_option):

abscisse = feuille['P5'].value

ordonnee = feuille['P6'].value

feuille['M12'].value = None

feuille['M13'].value = None

feuille.range(f'P5').color = (0, 255, 0)

feuille.range(f'P6').color = (0, 255, 0)

if abscisse not in df\_score.columns:

feuille['M11'].value = "Aucun graphique"

graphique\_option = False

feuille['M12'].value = "Abscisse pas valide"

feuille.range(f'P5').color = (255, 0, 0)

if ordonnee not in df\_score.columns :

feuille['M11'].value = "Aucun graphique"

graphique\_option = False

feuille['M13'].value = "Ordonnée pas valide"

feuille.range(f'P6').color = (255, 0, 0)

if(graphique\_option):

feuille\_graphiques = CreaFeuille(wb,'Graphiques')

FigLeftPos = 10

FigTopPos = 10

FigWidth = 350

FigHeight = 350

name = abscisse+"\_" +ordonnee+".png"

Alpha\_r\_df = R2\_Alpha\_Calc(abscisse, ordonnee)

R2\_Alpha\_Fig(Alpha\_r\_df, abscisse, ordonnee)

# Affichage du Graphique1 dans Excel

graphique1 = feuille\_graphiques.pictures.add(os.path.join(cwd+"\Graphiques", name ), name='Graphique1', update=True)

# Position sur la Feuille

graphique1.left = FigLeftPos

graphique1.top = FigTopPos

FigTopPos += FigWidth

# Redimensionnement

graphique1.width = FigWidth # Largeur en points

graphique1.height = FigHeight # Hauteur en points

finalDf = PCA\_Calc()

PCA\_Fig(finalDf)

graphique2 = feuille\_graphiques.pictures.add(os.path.join(cwd+"\Graphiques", 'PCA\_2D.png'), name='Graphique2', update=True)

# Position sur la Feuille

graphique2.left = FigLeftPos

graphique2.top = FigTopPos

FigTopPos += FigWidth

# Redimensionnement

graphique2.width = FigWidth # Largeur en points

graphique2.height = FigHeight # Hauteur en points

##wb.save(chemin\_pilotage)

Perf\_risk\_df = Perf\_Risk\_Calc()

Perf\_Risk\_Fig(Perf\_risk\_df)

graphique3 = feuille\_graphiques.pictures.add(os.path.join(cwd+"\Graphiques", 'Score\_Perf\_Moy\_Score\_Risks\_Moy.png'), name='Graphique3', update=True)

# Position sur la Feuille

graphique3.left = FigLeftPos

graphique3.top = FigTopPos

FigTopPos += FigWidth

# Redimensionnement

graphique3.width = FigWidth # Largeur en points

graphique3.height = FigHeight # Hauteur en points

Perf\_ESG\_df = Perf\_ESG\_Calc()

Perf\_ESG\_Fig(Perf\_ESG\_df)

graphique4 = feuille\_graphiques.pictures.add(os.path.join(cwd+"\Graphiques", 'Score\_Perf\_Moy\_Score\_ESG\_Moy.png'), name='Graphique4', update=True)

# Position sur la Feuille

graphique4.left = FigLeftPos

graphique4.top = FigTopPos

FigTopPos += FigWidth

# Redimensionnement

graphique4.width = FigWidth # Largeur en points

graphique4.height = FigHeight # Hauteur en points

Perf\_fees\_Fig(Perf\_fees\_df)

graphique5 = feuille\_graphiques.pictures.add(os.path.join(cwd+"\Graphiques", 'Score\_Perf\_fees.png'), name='Graphique5', update=True)

# Position sur la Feuille

graphique5.left = FigLeftPos

graphique5.top = FigTopPos

FigTopPos += FigWidth

# Redimensionnement

graphique5.width = FigWidth # Largeur en points

graphique5.height = FigHeight # Hauteur en points

def Find\_color(data):

col = (0,250,250)

if(data=='Très bon'):

col = (255, 0, 0) #Rouge

if(data=='Bon'):

col = (0, 255, 0) #Vert

if(data=='Moyen'):

col = (0, 0, 0) #Noir

if(data=='Mauvais'):

col = (0, 0, 255) #Bleu

return col

# Légende: Numéro des Fonds

for i in range(0,len(noms)):

couleur = Find\_color(Perf\_risk\_df.iloc[i,-1])

feuille\_graphiques.range(f'I{i + 1}').color = couleur

feuille\_graphiques.range(f'J{i + 1}').value = i

#feuille\_graphiques.range(f'K{i + 1}').color = couleur

feuille\_graphiques.range(f'K{i + 1}').value = df\_score.index[i]

#feuille\_graphiques.range(f'L{i + 1}').color = couleur

feuille\_graphiques.range(f'L{i + 1}').value = df\_score.iloc[i,0]

#Redimensionner les colonnes J et K en fonction de la taille du contenu

feuille\_graphiques.range('J:J').api.EntireColumn.AutoFit()

feuille\_graphiques.range('K:K').api.EntireColumn.AutoFit()

feuille\_graphiques.range('L:L').api.EntireColumn.AutoFit()

############################# Feuille Classement

top = CreaFeuille(wb,'Classements')

nbtop = 50

merged\_df = df\_score.reset\_index().merge(Perf\_fees\_df.reset\_index(), on='Name').set\_index('Name')

merged\_df.sort\_values(by="Notes",ascending=False,inplace=True)

colonnes\_ajoutees = ['ISIN\_x','Notes']

top.range('A1').value = merged\_df[colonnes\_ajoutees]

#.head(nbtop)

def adjust\_score(score, fees, max\_fees, adjustment\_factor):

#adjusadjustment\_factor = 0.25

if fees is not None:

adjustment = (fees / max\_fees) \* adjustment\_factor

adjusted\_score = round(score \* (1 - adjustment),2)

else:

adjusted\_score=0

return adjusted\_score

# Calculer la moyenne des valeurs non nulles

mean\_fee = merged\_df[merged\_df["Management Fee (ex Distribution fees) Actual\_x"] > 0]["Management Fee (ex Distribution fees) Actual\_x"].mean()

median\_fee = merged\_df[merged\_df["Management Fee (ex Distribution fees) Actual\_x"] > 0]["Management Fee (ex Distribution fees) Actual\_x"].median()

# Remplacer les valeurs de 0 par la moyenne

merged\_df.loc[merged\_df["Management Fee (ex Distribution fees) Actual\_x"] == 0, "Management Fee (ex Distribution fees) Actual\_x"] = mean\_fee

merged\_df = merged\_df[merged\_df["Notes"] > 0]

max\_fees = 1.7

merged\_df["Notes Ajustées des frais"] = merged\_df.apply(lambda row: adjust\_score(row["Notes"], row["Management Fee (ex Distribution fees) Actual\_x"],max\_fees,ValAdjustFees), axis=1)

merged\_df.sort\_values(by="Notes Ajustées des frais",ascending=False,inplace=True)

coeff=20/(1+merged\_df["Notes Ajustées des frais"].iloc[0])

tab\_notes=[]

for i in merged\_df["Notes Ajustées des frais"]:

tab\_notes.append(round(coeff\*(1+i),2))

merged\_df["Notes avec Frais"]=tab\_notes

colonnes\_ajoutees = ['ISIN\_x','Notes avec Frais','Fees Quali']

top.range('E1').value = merged\_df[colonnes\_ajoutees].head(nbtop)

top.range('A:H').api.EntireColumn.AutoFit()

############### Statistiques Descriptives pour l'étude de la valeur de adjustment\_factor

# Analyse de sensibilité : On peut effectuer une analyse de sensibilité en faisant varier adjustment\_factor

# sur une plage de valeurs et en calculant des statistiques descriptives pour les notes ajustées

# (par exemple, moyenne, médiane, écart-type).

# Cela permet de voir comment les notes ajustées changent en fonction des différentes valeurs de adjustment\_factor.

if(graphique\_option):

list\_adjustment\_factors = [0,0.2,0.4,0.6,0.8,1]

def sensitivity\_analysis(adjustment\_factors, df):

results = []

for factor in adjustment\_factors:

df[f"Adjusted Notes {factor}"] = df.apply(

lambda row: adjust\_score(

row["Notes"], row["Management Fee (ex Distribution fees) Actual\_x"], max\_fees, factor

),

axis=1,

)

mean = df[f"Adjusted Notes {factor}"].mean()

median = df[f"Adjusted Notes {factor}"].median()

std\_dev = df[f"Adjusted Notes {factor}"].std()

results.append({"Adjustment Factor": factor, "Mean": mean, "Median": median, "Standard Deviation": std\_dev})

return pd.DataFrame(results)

sensitivity\_results = sensitivity\_analysis(list\_adjustment\_factors, merged\_df)

feuille\_graphiques.range('N2').value = sensitivity\_results

# Analyse de corrélation : On peut calculer la corrélation entre les frais de gestion et les notes ajustées

# pour différentes valeurs de adjustment\_factor. Une corrélation négative indiquera que les frais de gestion ont un impact

# négatif sur les notes ajustées, et plus la corrélation est négative, plus l'impact est important.

def correlation\_analysis(adjustment\_factors, df):

correlations = []

for factor in adjustment\_factors:

df[f"Adjusted Notes {factor}"] = df.apply(

lambda row: adjust\_score(

row["Notes"], row["Management Fee (ex Distribution fees) Actual\_x"], max\_fees, factor

),

axis=1,

)

if df["Management Fee (ex Distribution fees) Actual\_x"].isnull().all():

corr = 0

else:

corr = df[[f"Adjusted Notes {factor}", "Management Fee (ex Distribution fees) Actual\_x"]].corr().iloc[0, 1]

correlations.append({"Adjustment Factor": factor, "Correlation": corr})

return pd.DataFrame(correlations)

correlation\_results = correlation\_analysis(list\_adjustment\_factors, merged\_df)

# Visualisation de l'impact sur le classement : On peut créer un graphique qui montre comment le classement des

# fonds change en fonction des différentes valeurs de adjustment\_factor. Par exemple, un diagramme

# à barres empilées où chaque barre représente un fonds, et les segments de chaque barre représentent les notes

# ajustées pour différentes valeurs de adjustment\_factor. Cela permet de voir comment les classements des

# fonds sont affectés par les différentes valeurs de adjustment\_factor.

def plot\_ranking\_impact(adjustment\_factors, df):

rankings = pd.DataFrame()

for factor in adjustment\_factors:

df[f"Adjusted Notes {factor}"] = df.apply(

lambda row: adjust\_score(

row["Notes"], row["Management Fee (ex Distribution fees) Actual\_x"], max\_fees, factor

),

axis=1,

)

df[f"Rank {factor}"] = df[f"Adjusted Notes {factor}"].rank(ascending=False)

rankings[f"Rank {factor}"] = df[f"Rank {factor}"]

rankings.plot(kind='bar', figsize=(15, 6))

plt.xlabel("Funds")

plt.ylabel("Rank")

plt.title("Impact of Adjustment Factors on Fund Ranking")

plt.legend([f"Adjustment Factor: {factor}" for factor in adjustment\_factors])

plt.tight\_layout()

plt.savefig(os.path.join(cwd+"\Graphiques",'Impact\_Adjustment\_Factor.png'))

plot\_ranking\_impact(list\_adjustment\_factors, merged\_df)

graphique10 = feuille\_graphiques.pictures.add(os.path.join(cwd+"\Graphiques", 'Impact\_Adjustment\_Factor.png'), name='Graphique10', update=True)

# Position sur la Feuille

graphique10.left = 900

graphique10.top = 200

# Redimensionnement

#§graphique6.width = 300 # Largeur en points

#raphique6.height = 450 # Hauteur en points

# Optimisation basée sur les objectifs : La métrique d'évaluation proposée mesure la corrélation

# entre les notes ajustées des frais et les frais de gestion.

# L'idée de cette métrique est de maximiser la relation inverse entre les notes et les frais de gestion.

# En d'autres termes, on cherche à obtenir une situation où les fonds avec des frais de gestion plus élevés

# ont des notes ajustées plus faibles et vice versa.

def evaluation\_metric(df):

res = -1 \* df["Notes ajustées des frais"].corr(df["Management Fee (ex Distribution fees) Actual\_x"])

return res

def calculate\_adjusted\_notes(df, adjustment\_factor, max\_fees):

df["Notes ajustées des frais"] = df.apply(lambda row: adjust\_score(row["Notes"], row["Management Fee (ex Distribution fees) Actual\_x"], max\_fees, adjustment\_factor), axis=1)

return df

def optimize\_adjustment\_factor(df, max\_fees, possible\_factors):

best\_factor = None

best\_metric = float('inf')

for factor in possible\_factors:

adjusted\_notes\_df = calculate\_adjusted\_notes(df.copy(), factor, max\_fees)

metric\_value = evaluation\_metric(adjusted\_notes\_df)

if metric\_value < best\_metric:

best\_metric = metric\_value

best\_factor = factor

return best\_factor

possible\_factors = np.linspace(0, 1, num=20)

best\_adjustment\_factor = optimize\_adjustment\_factor(merged\_df.copy(), max\_fees, possible\_factors)

###############

df\_fees = Perf\_fees\_Calc()

print("L'exécution du code a été réalisée avec succès")

#fonction pour renvoyer le tableau avec la liste des coefficients dans l'ordre

def verify\_coeff(nom\_fichier, colonneCritere, colonneCoeff, feuille ="Sélection des critères"):

wb = xw.Book(nom\_fichier)

feuille = wb.sheets[feuille]

coeff = []

derniere\_ligne = feuille.range(colonneCritere + str(feuille.cells.last\_cell.row)).end('up').row

for i in range(2, derniere\_ligne + 1):

valeur = feuille.range(colonneCoeff + str(i)).value

if valeur is None or (isinstance(valeur, int) and valeur < 0): #si pas de valeur alors le coefficient est égal à 1 par défaut

valeur = 1

coeff.append(valeur)

return coeff

#fonction pour vérifier si les mots saisis sont déjà connu par l'algorithme ou non

def verify\_existence\_word(liste\_mot, nom\_fichier,colonneCritere, colonneObject, feuille = 'Sélection des critères'):

wb = xw.Book(nom\_fichier)

feuille = wb.sheets[feuille]

listCriteria = []

derniere\_ligne = feuille.range(colonneCritere + str(feuille.cells.last\_cell.row)).end('up').row

error = False

dic\_unknow\_criteria = dict()

for i in range(2, derniere\_ligne + 1): #parcours les critères

valeur = feuille.range(colonneCritere + str(i)).value

if valeur not in liste\_mot or valeur in listCriteria : #si le critère n'appartient ni à la liste connue ni est déjà présente dans la colonne

objectif = feuille.range(colonneObject + str(i)).value

if objectif not in ['H', 'L']: #si c'est le cas alors je vérifie que la case objectif a été remplie correctement

print(valeur, " est inconnu et vous n'avez pas renseigné la case objectif")

feuille.range(colonneCritere + str(i)).color = (255, 0, 0)

error = True

else :

dic\_unknow\_criteria[valeur] = objectif #si la case objectif a été remplie alors je l'ajoute au dictionnaire des critères inconnu et je l'associe à son objectif (L ou H)

listCriteria.append(valeur)

feuille.range(colonneCritere + str(i)).color = (200,32, 100)

else :

listCriteria.append(valeur)

feuille.range(colonneCritere + str(i)).color = (0, 255, 0)

print("la liste des critères est ",listCriteria)

print("les éléments non connus par notre algorithme sont", dic\_unknow\_criteria)

return listCriteria, error, dic\_unknow\_criteria

#fonction afin de vérifier si les poids renseignés sont valides

def verify\_weight(nom\_fichier, feuille="Sélection des critères"):

wb = xw.Book(nom\_fichier)

feuille = wb.sheets[feuille]

PoidsPerf = round(feuille['J2'].value,2)

PoidsRisk = round(feuille['K2'].value,2)

PoidsESG = round(feuille['L2'].value,2)

#erreur si un des poids n'a pas été renseigné

if None in (PoidsPerf, PoidsRisk, PoidsESG):

if PoidsPerf is None:

feuille.range(f'J2').color = (255, 0, 0)

if PoidsRisk is None:

feuille.range(f'K2').color = (255, 0, 0)

if PoidsESG is None:

feuille.range(f'L2').color = (255, 0, 0)

return PoidsPerf,PoidsRisk,PoidsESG,"Données manquantes dans poids"

#erreur les poids ne sont pas compris entre 0 et 1

if not all(0 <= poids <= 1 for poids in (PoidsPerf, PoidsRisk, PoidsESG)):

feuille.range(f'J2').color = (255, 0, 0)

feuille.range(f'K2').color = (255, 0, 0)

feuille.range(f'L2').color = (255, 0, 0)

return PoidsPerf,PoidsRisk,PoidsESG,"Poids doit être compris entre 0 et 100%"

#erreur sur la somme des poids non égale à 1

somme\_poids = PoidsPerf + PoidsRisk + PoidsESG

if somme\_poids != 1:

feuille.range(f'J2').color = (255, 0, 0)

feuille.range(f'K2').color = (255, 0, 0)

feuille.range(f'L2').color = (255, 0, 0)

return PoidsPerf,PoidsRisk,PoidsESG,"Somme des poids != 100%"

#aucune erreur pour les poids

feuille.range(f'J2').color = (0, 255, 0)

feuille.range(f'K2').color = (0, 255, 0)

feuille.range(f'L2').color = (0, 255, 0)

return PoidsPerf,PoidsRisk,PoidsESG, None

#fonction afin de vérifier le nom du fichier pour importer la data

'''def verify\_file\_name(nom\_fichier, feuille="Sélection des critères"):

wb = xw.Book(nom\_fichier)

feuille = wb.sheets[feuille]

error\_name = ""

data\_file = feuille['M2'].value

if(data\_file != 'emd\_global.xlsx' and data\_file != 'hy\_europe.xlsx'): #il n'y a que deux possibilités de nom de fichier soit emd\_global.xlsx ou hy\_europe.xlsx

feuille.range(f'M2').color = (255, 0, 0)

return "Nom de fichier invalide soit emd\_global.xlsx ou hy\_europe.xlsx", data\_file

feuille.range(f'M2').color = (0, 255, 0)

return None, data\_file'''

def graph\_option(nom\_fichier, feuille= "Sélection des critères"):

wb = xw.Book(nom\_fichier)

feuille = wb.sheets[feuille]

graphique\_option = feuille['P4'].value

if(graphique\_option == 'y' or graphique\_option == 'Y' or graphique\_option == 'yes' or graphique\_option == 'oui'):

graphique\_option = graphique\_option.lower()

feuille.range(f'P4').color = (0, 255, 0)

return True

else :

feuille.range(f'P4').color = (255, 165, 0)

return False

#fonction afin de vérifier la pertinence des seuils renseignés

def verify\_seuil(nom\_fichier, feuille="Sélection des critères"):

wb = xw.Book(nom\_fichier)

feuille = wb.sheets[feuille]

seuilmin = feuille['N2'].value

seuilmax = feuille['O2'].value

seuilpas = feuille['P2'].value

if(seuilmin is None or seuilmax is None or seuilpas is None): #si un des seuils n'a pas de valeur

feuille.range(f'N2').color = (255, 0, 0)

feuille.range(f'O2').color = (255, 0, 0)

feuille.range(f'P2').color = (255, 0, 0)

return seuilmin, seuilmax, seuilpas, "Problème(s) sur les seuils, vous n'avez pas renseigné une valeur"

if any(condition for condition in [seuilmin >= seuilmax,seuilmin < 0,seuilmax < 0,seuilmax > 1,seuilmin > 1,seuilpas > seuilmax - seuilmin]): #si ne respecte pas les conditions d'encadrement

feuille.range(f'N2').color = (255, 0, 0)

feuille.range(f'O2').color = (255, 0, 0)

feuille.range(f'P2').color = (255, 0, 0)

return seuilmin, seuilmax, seuilpas, "Problème(s) sur les seuils, vérifiez: 0 <seuilmin < seuilmax =<100%"

#si aucun pb

feuille.range(f'N2').color = (0, 255, 0)

feuille.range(f'O2').color = (0, 255, 0)

feuille.range(f'P2').color = (0, 255, 0)

return seuilmin, seuilmax, seuilpas, None

#fonction pour vérifier que la sélection du fees est bien compris entre 0 et 10 inclus

def verify\_fees(nom\_fichier, feuille="Sélection des critères"):

wb = xw.Book(nom\_fichier)

feuille = wb.sheets[feuille]

ValAdjustFees = int(feuille['Q2'].value)

if 0 <= ValAdjustFees <= 10:

feuille.range(f'Q2').color = (0, 255, 0)

return None, ValAdjustFees

else: #aucune erreur

feuille.range(f'Q2').color = (255, 0, 0)

return "ValAdjFees incorrecte doit être entre 0 et 10", ValAdjustFees

#Vérifier la catégorie selectionnée parmi les 6 possibles

'''def verify\_SelectedCategory(nom\_fichier, feuille="Sélection des critères"):

wb = xw.Book(nom\_fichier)

feuille = wb.sheets[feuille]

cell = feuille['R2'].value

liste\_selectedcategory = ['EAA Fund Global Emerging Markets Bond - EUR Biased','EAA Fund Global Emerging Markets Bond', 'EAA Fund Global Emerging Markets Corporate Bond - EUR Biased','EAA Fund Global Emerging Markets Corporate Bond','EAA Fund EUR High Yield Bond',None]

if cell not in liste\_selectedcategory: #si la celulle contient un élément hors des possibilités

feuille.range(f'R2').color = (255, 0, 0)

return "La catégorie n'est pas existante ou mettez un vide", cell

if cell != None :

feuille.range(f'R2').color = (0, 255, 0)

return None, cell #si la cellule contient un élément qui fait partie de la liste

else :

feuille.range(f'R2').color = (0, 255, 0)

return None, '' #si la cellule est vide, on prends toutes les catégories'''

def verify\_data(nomExcel):

#liste\_mot est la variable qui contient l'ensemble des fonctions dont nous avons un moyen de calculer son score

#liste\_mot = ["Max Drawdown"]

liste\_mot = []

nom\_fichier = nomExcel

feuille = 'Sélection des critères'

wb = xw.Book(nom\_fichier)

feuille = wb.sheets[feuille]

feuille['M14'].value = None

#vérification de l'existence des critères

list\_criteria\_Perf, error1, dic\_unknow\_criteria\_perf = verify\_existence\_word(liste\_mot, nom\_fichier, 'A', 'C')

list\_criteria\_Risk, error2, dic\_unknow\_criteria\_risk = verify\_existence\_word(liste\_mot, nom\_fichier, 'D', 'F')

list\_criteria\_ESG, error3, dic\_unknow\_criteria\_esg= verify\_existence\_word(liste\_mot, nom\_fichier, 'G','I')

#tableau qui contiendra l'ensemble des colonnes où il y a au moins une erreur

columns\_with\_errors = []

if error1:

columns\_with\_errors.append("A")

if error2:

columns\_with\_errors.append("D")

if error3:

columns\_with\_errors.append("G")

#vérification de la cohérence des poids

PoidPerf, PoidRisk, PoidESG, error\_weight = verify\_weight(nom\_fichier)

#vérification nom fichier pour la data

#error\_name, nomFichier = verify\_file\_name(nom\_fichier)

nomFichier = feuille['M2'].value

#vérification seuil

SeuilMin, SeuilMax, SeuilPas, error\_seuil = verify\_seuil(nom\_fichier)

#vérification fees

error\_fees, ValAdjustFees = verify\_fees(nom\_fichier)

#vérification SelectedCategory

#error\_category, SelectedCategory = verify\_SelectedCategory(nom\_fichier)

cell = feuille['R2'].value

if cell != None :

SelectedCategory = cell #si la cellule contient un élément qui fait partie de la liste

else :

SelectedCategory = '' #si la cellule est vide, on prends toutes les catégories

#vérification graph\_option

graphique\_option = graph\_option(nom\_fichier)

#condition qui s'exécute lorsqu'il y a au moins une erreur, pour afficher les différents messages d'erreurs

if len(columns\_with\_errors)!=0 or error\_weight is not None or error\_seuil is not None or error\_fees is not None:

feuille['M5'].value = ""

if(len(columns\_with\_errors)!=0):

error\_message = "Remplissez la colonne objectif pour l'élement en rouge colonne(s): " + ", ".join(columns\_with\_errors),

feuille['M5'].value = error\_message

feuille['M6'].value = error\_weight

#feuille['M7'].value = error\_name

feuille['M8'].value = error\_seuil

feuille['M9'].value = error\_fees

#feuille['M10'].value = error\_category

raise ValueError

#réinitialiser les valeurs des cases qui affiche les erreurs si il n'y a pas d'erreur

feuille['M5'].value = "Aucune erreur de remplissage"

feuille['M6'].value = None

feuille['M7'].value = None

feuille['M8'].value = None

feuille['M9'].value = None

feuille['M10'].value = None

if graphique\_option :

feuille['M11'].value = "Le graphique sera tracé"

else:

feuille['M11'].value = "Aucun graphique"

#récupère les coefficients des critères

coeff\_perf = verify\_coeff(nom\_fichier, 'A', 'B')

coeff\_risk = verify\_coeff(nom\_fichier, 'D', 'E')

coeff\_esg = verify\_coeff(nom\_fichier, 'G', 'H')

chemin\_init = os.path.dirname(os.path.abspath(\_\_file\_\_))

#chemin\_init = "U:\\GDA\\PFC\\02\_Ressources\\Scoring"

#traiter le cas du EAA Fund EUR High Yield Bond qui n'est disponible que dans le fichier hy\_europe.xlsx

#if(nomFichier == 'emd\_global.xlsx' and SelectedCategory == 'EAA Fund EUR High Yield Bond'):

# nomFichier = 'hy\_europe.xlsx

return(nomExcel,chemin\_init,nomFichier, int(round(SeuilMin\*100,0)),int(round(SeuilMax\*100,0)), int(round(SeuilPas\*100,0)), int(round(PoidPerf\*100,0)), int(round(PoidRisk\*100,0)), int(round(PoidESG\*100,0)), SelectedCategory, ValAdjustFees,list\_criteria\_Perf, list\_criteria\_Risk, list\_criteria\_ESG, coeff\_perf, coeff\_risk, coeff\_esg, dic\_unknow\_criteria\_perf, dic\_unknow\_criteria\_risk, dic\_unknow\_criteria\_esg, graphique\_option)

def startperso(nomExcel):

error = False

try:

nomExcel, chemin\_init, nomFichier, SeuilMin, SeuilMax, SeuilPas, PoidPerf, PoidRisk, PoidESG, SelectedCategory, ValAdjustFees, list\_criteria\_Perf, list\_criteria\_Risk, list\_criteria\_ESG, coeff\_perf, coeff\_risk, coeff\_esg, dic\_unknow\_criteria\_perf, dic\_unknow\_criteria\_risk, dic\_unknow\_criteria\_esg, graphique\_option = verify\_data(nomExcel)

except ValueError:

print("Oops! That was not a valid input. Try again...")

error = True

if error == False:

start(nomExcel,chemin\_init, nomFichier, SeuilMin, SeuilMax, SeuilPas, PoidPerf, PoidRisk, PoidESG, SelectedCategory, ValAdjustFees, True, list\_criteria\_Perf, list\_criteria\_Risk, list\_criteria\_ESG, coeff\_perf, coeff\_risk, coeff\_esg, dic\_unknow\_criteria\_perf, dic\_unknow\_criteria\_risk, dic\_unknow\_criteria\_esg, graphique\_option)

#pour faire démarrer directement en exécutant depuis VS

#start(r'U:\GDA\PFC\02\_Ressources\Scoring','emd\_global.xlsx',80,90,5,100,0,0,'Toutes Catégories',0)

Sur vba :

Sub Classement\_perso()

Dim dynamicPath As String

dynamicPath = ThisWorkbook.Path

dynamicPath = Replace(dynamicPath, "\", "\\")

RunPython "import sys; sys.path.insert(0, '" & dynamicPath & "'); import Pilotage; Pilotage.startperso('" & ThisWorkbook.Name & "')"

' Exécute Pilotage.py en utilisant xlwings et en passant nomFichier en argument

' RunPython "import sys; sys.path.insert(0, 'U:\\GDA\\PFC\\02\_Ressources\\scoring-tmp'); import Pilotage; Pilotage.startperso()"

End Sub

Scoring MX3

import os

import datetime

import shutil

import pandas as pd

def select\_files(path\_input, path\_output, date\_list):

selected\_files = {}

original\_data\_path = os.path.join(path\_output, "originally\_data\_files")

if not os.path.exists(original\_data\_path):

os.makedirs(original\_data\_path)

for date\_input in date\_list:

date\_obj = datetime.datetime.strptime(date\_input, '%Y%m%d')

max\_timestamp = 0

selected\_file = None

for filename in os.listdir(path\_input):

if filename.startswith('Stock\_Action\_DOPE\_'):

file\_split = filename.split('\_')[3]

if file\_split.endswith('.csv'):

file\_split = file\_split.replace('.csv', '')

file\_date = datetime.datetime.strptime(file\_split, '%Y%m%d')

if file\_date.date() == date\_obj.date():

timestamp = int(filename.split('\_')[-1][:-4])

if timestamp > max\_timestamp:

max\_timestamp = timestamp

selected\_file = filename

selected\_files[date\_input] = selected\_file

if selected\_file is not None:

source\_file = os.path.join(path\_input, selected\_file)

destination\_file = os.path.join(original\_data\_path, selected\_file)

shutil.copy(source\_file, destination\_file)

return selected\_files

def process\_selected\_files(selected\_files, path\_output, identifier):

for date, filename in selected\_files.items():

if filename is not None:

source\_file = os.path.join(path\_output, "originally\_data\_files", filename)

identifier\_path = os.path.join(path\_output, "portfolio", identifier)

if not os.path.exists(identifier\_path):

os.makedirs(identifier\_path)

extract\_data\_path = os.path.join(identifier\_path, "extract\_data")

if not os.path.exists(extract\_data\_path):

os.makedirs(extract\_data\_path)

date\_path = os.path.join(extract\_data\_path,date)

if not os.path.exists(date\_path):

os.makedirs(date\_path)

df = pd.read\_csv(source\_file, sep=';')

filtered\_df = df[df['Portefeuille'] == identifier]

if(len(filtered\_df)==0):

raise Exception(f"Le portefeuille '{identifier}' n'existe pas à la date '{date}'.")

output\_file = os.path.join(date\_path, f"{identifier}\_{date}.xlsx")

filtered\_df.to\_excel(output\_file, index=False)

def construction\_morning\_star\_files(path\_input, numero\_ptf, date\_list, portfolio\_name, portfolio\_id, path\_output):

selected\_files = select\_files(path\_input, path\_output, date\_list)

process\_selected\_files(selected\_files, path\_output, numero\_ptf)

for date in date\_list:

extract\_folder = os.path.join(path\_output, "portfolio", numero\_ptf, "extract\_data",date)

file\_path\_ms = os.path.join(path\_output, "portfolio", numero\_ptf, "files\_morningstar")

os.makedirs(file\_path\_ms, exist\_ok=True)

for filename in os.listdir(extract\_folder):

if filename.endswith('.xlsx'):

extract\_data\_file = os.path.join(extract\_folder, filename)

df = pd.read\_excel(extract\_data\_file)

new\_filename = f"{filename.split('.')[0]}\_morningstar.xlsx"

new\_file\_path = os.path.join(file\_path\_ms, new\_filename)

df = df.rename(columns={

'Libelle Titre': 'Nom ss jacents',

'Code Isin': 'ISIN',

'Code sedol': 'sedol',

'Nb Titre': 'qt'

})

date =datetime.datetime.strptime(date, "%Y%m%d")

date = date.strftime("%d/%m/%Y")

df['date'] = date

df['Nom portefeuille'] = portfolio\_name

df['ID portefeuille'] = portfolio\_id

df = df[['date','Nom portefeuille', 'ID portefeuille', 'Nom ss jacents', 'ISIN', 'sedol', 'qt']]

df.to\_excel(new\_file\_path, index=False, engine='openpyxl')

def construction\_bloomberg\_files(path\_input, numero\_ptf, date\_list, portfolio\_name, portfolio\_id, path\_output):

selected\_files = select\_files(path\_input, path\_output, date\_list)

process\_selected\_files(selected\_files, path\_output, numero\_ptf)

for date in date\_list:

extract\_folder = os.path.join(path\_output, "portfolio", numero\_ptf, "extract\_data",date)

file\_path\_bloomberg = os.path.join(path\_output, "portfolio", numero\_ptf, "files\_bloomberg")

os.makedirs(file\_path\_bloomberg, exist\_ok=True)

for filename in os.listdir(extract\_folder):

if filename.endswith('.xlsx'):

extract\_data\_file = os.path.join(extract\_folder, filename)

df = pd.read\_excel(extract\_data\_file)

new\_filename = f"{filename.split('.')[0]}\_bloomberg.xlsx"

new\_file\_path = os.path.join(file\_path\_bloomberg, new\_filename)

#'code sedol': 'SEDOL',

df = df.rename(columns={

'Libelle Titre': 'SECURITY NAME',

'Code sedol': 'SEDOL',

'Code Isin': 'ISIN',

'Nb Titre': 'QUANTITY',

'Prix revient Moyen': 'Cost Price'

})

date\_obj = datetime.datetime.strptime(date, "%Y%m%d")

date\_str = date\_obj.strftime("%m/%d/%Y")

df['PORTFOLIO NAME'] = portfolio\_name

df['As of Date'] = date\_str

df['QUANTITY'] = df['QUANTITY'].str.replace(',', '.').astype(float)

df['Cost Price'] = df['Cost Price'].str.replace(',', '.').astype(float)

df = df[['PORTFOLIO NAME', 'SEDOL','ISIN', 'SECURITY NAME', 'QUANTITY', 'Cost Price', 'As of Date']]

df.to\_excel(new\_file\_path, index=False, engine='openpyxl')

path\_input = r"U:\TRANSFERTS\_FICHIERS\PROD\SIGMA\Com\_StockAction"

path\_output = r"U:\GDA\PFC\02\_Ressources\Scoring-ESG & Extraction MX3"

date\_list = ["20240822"]

numero\_ptf = 'GF150'

portfolio\_name = "CXA"

portfolio\_id = "CXA"

construction\_morning\_star\_files(path\_input,numero\_ptf, date\_list,portfolio\_name, portfolio\_id, path\_output)

construction\_bloomberg\_files(path\_input, numero\_ptf, date\_list, portfolio\_name, portfolio\_id, path\_output)

Tableau de bord – Forward

Sub BoutonUnique\_Cliquer()

Call ImportShiller2("EquityEU")

Call Download\_HY\_US2("HYCorpoUS")

Call Download\_HY\_EU2("HYCorpoEU")

Call Bouton\_Cliquer("IGCorpoUS")

Call Bouton\_Cliquer("IGCorpoEU")

Call Bouton\_Cliquer("HYCorpoUS")

Call Bouton\_Cliquer("HYCorpoEU")

' Sheets("EquityEU").Calculate

Sheets("Performancev2").Activate

End Sub

Sub Download\_HY\_US2(sheetName As String)

Dim FichierCSV As Workbook

Dim FeuilleCible As Worksheet

Dim FeuilleSource As Worksheet

Dim NombreLignes As Long

Dim DestRange As Range

Dim CheminFichier As String

Dim Fichiers() As String

Dim i As Long

Dim DateFormat As String

Dim DateMaJ As Long

Dim FichierMaJ As String

Dim FichiersCount As Long

' Change le chemin qui correspond au dossier où il y a les fichiers de data

CheminFichier = "U:\GDA\PFC\02\_Ressources\Tableau de bord - Forward\"

' Récupérer la liste des fichiers dans le dossier

Dim Fichier As String

Fichier = Dir(CheminFichier & "GDAPFC\*", vbNormal)

i = 0

While Fichier <> ""

ReDim Preserve Fichiers(i)

Fichiers(i) = Fichier

Fichier = Dir

i = i + 1

Wend

FichiersCount = i

DateMaJ = 19000101

FichierMaJ = ""

For i = 0 To UBound(Fichiers)

DateFormat = Mid(Fichiers(i), InStrRev(Fichiers(i), "-20") + 1, 8)

If DateFormat > DateMaJ Then

DateMaJ = DateFormat

FichierMaJ = Fichiers(i)

End If

Next i

MsgBox "Le fichier sélectionné est : " & FichierMaJ

Set FichierCSV = Workbooks.Open(CheminFichier & FichierMaJ, , True)

Set FeuilleCible = FichierCSV.Worksheets(1)

Set FeuilleSource = ThisWorkbook.Worksheets(sheetName)

FeuilleSource.Range("Z1:AF14").Borders.LineStyle = xlNone

FeuilleSource.Range("Z1:AF14").ClearContents

NombreLignes = FeuilleCible.UsedRange.Rows.count

FeuilleCible.Range("A1:Z12").Copy Destination:=FeuilleSource.Range("Z1")

FichierCSV.Close SaveChanges:=False

' Renommer colonne A

FeuilleSource.Range("A2") = "3M"

FeuilleSource.Range("A3") = "6M"

For i = 4 To 13

FeuilleSource.Range("A" & i) = i - 3

Next i

' Mettre la date de la data en A1

FeuilleSource.Range("A1") = FeuilleSource.Range("AB4").Value

For i = 2 To 8

FeuilleSource.Range("B" & i) = FeuilleSource.Range("AF" & i)

Next i

For i = 10 To 13

FeuilleSource.Range("B" & i) = FeuilleSource.Range("AF" & i - 1)

Next i

FeuilleSource.Range("B1") = "YTM"

FeuilleSource.Range("B9").FormulaLocal = "=MOYENNE(B8;B10)"

FeuilleSource.Range("A1:B13").Borders.LineStyle = xlContinuous

FeuilleSource.Range("A1:B13").WrapText = True

FeuilleSource.Range("A1:B13").HorizontalAlignment = xlCenter

FeuilleSource.Range("A1:B13").VerticalAlignment = xlCenter

End Sub

Sub Download\_HY\_EU2(sheetName As String)

Dim FichierCSV As Workbook

Dim FeuilleCible As Worksheet

Dim FeuilleSource As Worksheet

Dim NombreLignes As Long

Dim DestRange As Range

Dim CheminFichier As String

Dim Fichiers() As String

Dim i As Long

Dim DateFormat As String

Dim DateMaJ As Long

Dim FichierMaJ As String

Dim FichiersCount As Long

' Change le chemin qui correspond au dossier où il y a les fichiers de data

CheminFichier = "U:\GDA\PFC\02\_Ressources\Tableau de bord - Forward\"

' Récupérer la liste des fichiers dans le dossier

Dim Fichier As String

Fichier = Dir(CheminFichier & "GDAPFC\*", vbNormal)

i = 0

While Fichier <> ""

ReDim Preserve Fichiers(i)

Fichiers(i) = Fichier

Fichier = Dir

i = i + 1

Wend

FichiersCount = i

DateMaJ = 19000101

FichierMaJ = ""

For i = 0 To UBound(Fichiers)

DateFormat = Mid(Fichiers(i), InStrRev(Fichiers(i), "-20") + 1, 8)

If DateFormat > DateMaJ Then

DateMaJ = DateFormat

FichierMaJ = Fichiers(i)

End If

Next i

MsgBox "Le fichier sélectionné est : " & FichierMaJ

Set FichierCSV = Workbooks.Open(CheminFichier & FichierMaJ, , True)

Set FeuilleCible = FichierCSV.Worksheets(1)

Set FeuilleSource = ThisWorkbook.Worksheets(sheetName)

FeuilleSource.Range("Z1:AF14").Borders.LineStyle = xlNone

FeuilleSource.Range("Z1:AF14").ClearContents

NombreLignes = FeuilleCible.UsedRange.Rows.count

FeuilleCible.Range("A13:Z23").Copy Destination:=FeuilleSource.Range("Z1")

FichierCSV.Close SaveChanges:=False

' Renommer colonne A

FeuilleSource.Range("A2") = "3M"

FeuilleSource.Range("A3") = "6M"

For i = 4 To 13

FeuilleSource.Range("A" & i) = i - 3

Next i

' Mettre la date de la data en A1

FeuilleSource.Range("A1") = FeuilleSource.Range("AB4").Value

For i = 2 To 8

FeuilleSource.Range("B" & i) = FeuilleSource.Range("AF" & i - 1)

Next i

For i = 10 To 13

FeuilleSource.Range("B" & i) = FeuilleSource.Range("AF" & i - 2)

Next i

FeuilleSource.Range("B1") = "YTM"

FeuilleSource.Range("B9").FormulaLocal = "=MOYENNE(B8;B10)"

FeuilleSource.Range("A1:B13").Borders.LineStyle = xlContinuous

FeuilleSource.Range("A1:B13").WrapText = True

FeuilleSource.Range("A1:B13").HorizontalAlignment = xlCenter

FeuilleSource.Range("A1:B13").VerticalAlignment = xlCenter

End Sub

Sub Bouton\_Cliquer(sheetName As String)

Worksheets(sheetName).Activate

SolverOk SetCell:="$O$2", MaxMinVal:=3, ValueOf:="0", ByChange:="$C$5"

SolverSolve True

SolverOk SetCell:="$P$2", MaxMinVal:=3, ValueOf:="0", ByChange:="$C$6"

SolverSolve True

SolverOk SetCell:="$Q$2", MaxMinVal:=3, ValueOf:="0", ByChange:="$C$7"

SolverSolve True

SolverOk SetCell:="$R$2", MaxMinVal:=3, ValueOf:="0", ByChange:="$C$8"

SolverSolve True

SolverOk SetCell:="$S$2", MaxMinVal:=3, ValueOf:="0", ByChange:="$C$9"

SolverSolve True

SolverOk SetCell:="$T$2", MaxMinVal:=3, ValueOf:="0", ByChange:="$C$10"

SolverSolve True

SolverOk SetCell:="$U$2", MaxMinVal:=3, ValueOf:="0", ByChange:="$C$11"

SolverSolve True

SolverOk SetCell:="$V$2", MaxMinVal:=3, ValueOf:="0", ByChange:="$C$12"

SolverSolve True

SolverOk SetCell:="$W$2", MaxMinVal:=3, ValueOf:="0", ByChange:="$C$13"

SolverSolve True

Dim seuil\_sup As Double

seuil\_sup = Range("B35").Value

Dim seuil\_inf As Double

seuil\_inf = Range("B33").Value

Dim rngA As Range, cell As Range

Dim ligne As Long

Set rngA = Range("A4:A13")

For Each cell In rngA

If cell.Value = seuil\_inf Then

ligne = cell.Row

Range("C33").Value = Range("C" & ligne).Value

End If

Next cell

For Each cell In rngA

If cell.Value = seuil\_sup Then

ligne = cell.Row

Range("C35").Value = Range("C" & ligne).Value

End If

Next cell

End Sub

Sub ImportShiller2(sheetName As String)

Dim ws As Worksheet

Dim csvFilePath As String

Dim wbCSV As Workbook

Dim rawDate As Long

Dim endDate As Date

Dim lastDateInCSV As Date

Dim i As Long

Worksheets(sheetName).Activate

Set ws = ThisWorkbook.Sheets("EquityEU")

csvFilePath = "U:\GDA\PFC\02\_Ressources\Tableau de bord - Forward\Historic-cape-ratios.csv"

rawDate = ws.Range("P14").Value

endDate = DateSerial(Left(rawDate, 4), Mid(rawDate, 5, 2), Mid(rawDate, 7, 2))

endDate = DateSerial(Year(endDate), Month(endDate), 0)

ws.Columns("B:C").Clear

ws.Columns("D").Clear

Set wbCSV = Workbooks.Open(Filename:=csvFilePath, Format:=2) ' Format:=2 spécifie un délimiteur de virgule

With wbCSV.Sheets(1)

For i = 1 To .Cells(.Rows.count, 1).End(xlUp).Row

If IsDate(.Cells(i, 1).Value) Then

lastDateInCSV = .Cells(i, 1).Value

If lastDateInCSV > endDate Then Exit For

End If

Next i

.Range("A1:B" & i - 1).Copy ws.Range("B1")

End With

wbCSV.Close SaveChanges:=False

ws.Columns("D").Clear

ws.Activate

End Sub

Modèle arbitrage :

import numpy as np

import pandas as pd

from tia.bbg import LocalTerminal

from collections import OrderedDict

import pickle

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

from sklearn.metrics import r2\_score

from datetime import datetime

from dateutil.relativedelta import relativedelta

from tia.bbg import LocalTerminal

import datetime

import os

import luigi

import shutil

from functions\_arbitrage import Results,get\_periods\_per\_year,trades\_visu, trades\_visu,strategy\_weights,tradoffline\_visu,trading\_rule

from global\_parameter import ClassGlobal, SelectData

from gestion\_hdf5 import h5load, h5store

from growth\_estimation import get\_GrowthRate\_LongTerm, get\_GrowthRate\_ShortTerm

from edr.get\_edr import get\_EDR

from vola\_download.vola\_data\_download import Collecte\_data\_Model\_Vola

class Arbitrage(luigi.Task):

Doc\_name = luigi.Parameter()

class\_path\_output = luigi.Parameter()

class\_path\_output=os.path.join(ClassGlobal().path\_output,"Simulation\{}".format(ClassGlobal().simulation))

date1 = SelectData().date1

def output(self):

return luigi.LocalTarget(os.path.join(self.class\_path\_output,"Results\Returns"+ClassGlobal().Doc\_name+".xlsx")),luigi.LocalTarget(os.path.join(self.class\_path\_output,"Results\Volatility"+ClassGlobal().Doc\_name+".xlsx"))

def run(self):

path\_data = os.path.join(ClassGlobal().path\_output,"Simulation\{}\Assets\Assets\_{}".format(ClassGlobal().simulation,datetime.datetime.now().strftime("%Y\_%m\_%d")))

file\_Prices = "Prices\_"+self.Doc\_name+".h5"

file\_Vol = "Vol\_"+self.Doc\_name+".h5"

file\_Yields = "Yields\_"+self.Doc\_name+".h5"

prices = pd.read\_hdf(path\_data+"\\"+file\_Prices)

prices,metadata = h5load(path\_data+"\\"+file\_Prices)

volatilities = pd.read\_hdf(path\_data+"\\"+file\_Vol)

yields = pd.read\_hdf(path\_data+"\\"+file\_Yields)

#Slope

esp\_vol = volatilities.dropna()

exp\_ret = yields.dropna()

X = esp\_vol

Y = exp\_ret

intercept\_values = exp\_ret.filter(like="Cash")[exp\_ret.filter(like="Cash").columns[0]]

first\_date\_vol = esp\_vol.index[0]

slopes = []

for date in exp\_ret.index:

if(date >= first\_date\_vol):

X\_date = esp\_vol.loc[date].values.reshape(-1, 1)

y\_date = exp\_ret.loc[date].values

intercept\_value = intercept\_values.loc[date]

y\_adjusted = y\_date - intercept\_value

model = LinearRegression(fit\_intercept=False)

model.fit(X\_date, y\_adjusted)

slope = model.coef\_[0]

slopes.append(slope)

else:

slopes.append(0)

slopes\_df = pd.DataFrame(data={'slope': slopes}, index=exp\_ret.index)

#Affichage Graphique

# plt.figure(figsize=(12, 6))

# plt.plot(slopes\_df.index, slopes\_df['slope'])

# plt.xlabel('Date')

# plt.ylabel('Slope')

# plt.title('Slope vs Date')

# plt.show()

slopeReg = slopes\_df['slope']

slope=slopeReg.to\_frame("slope")

theo\_ret = esp\_vol.multiply(slope["slope"], axis=0).add(intercept\_values, axis=0)

theo\_ret.dropna(inplace=True)

diff\_ret = exp\_ret - theo\_ret

plot = diff\_ret.plot()

fig = plot.get\_figure()

#################

fig.savefig(os.path.join(self.class\_path\_output,'Ecart\_Droite\_'+ClassGlobal().Doc\_name+'.png'))

diff\_ret.to\_excel(os.path.join(self.class\_path\_output,'Results\Ecart\_Droite\_'+ClassGlobal().Doc\_name+'.xlsx'))

h5store(os.path.join(self.class\_path\_output,"Results\Ecart"+ClassGlobal().Doc\_name+".h5"), esp\_vol, metadata)

#################

date1 = self.date1

tradoffline\_visu(date1,slope,exp\_ret,esp\_vol,SelectData().zone,ClassGlobal().simulation,ClassGlobal().Doc\_name,self.class\_path\_output)

exp\_ret.reindex(index=exp\_ret.index[::-1]).to\_excel(os.path.join(self.class\_path\_output,"Results\Returns"+ClassGlobal().Doc\_name+".xlsx"))

# exp\_ret.to\_hdf(os.path.join(self.class\_path\_output,"Results\Returns"+ClassGlobal().Doc\_name+".h5"))

h5store(os.path.join(self.class\_path\_output,"Results\Returns"+ClassGlobal().Doc\_name+".h5"), exp\_ret, metadata)

esp\_vol.reindex(index=esp\_vol.index[::-1]).to\_excel(os.path.join(self.class\_path\_output,"Results\Volatility"+ClassGlobal().Doc\_name+".xlsx"))

# esp\_vol.to\_hdf(os.path.join(self.class\_path\_output,"Results\Volatility"+ClassGlobal().Doc\_name+".h5"))

h5store(os.path.join(self.class\_path\_output,"Results\Volatility"+ClassGlobal().Doc\_name+".h5"), esp\_vol, metadata)

#################

trades = diff\_ret.apply(trading\_rule, axis=1) # Sur les colonnes

trades.dropna(inplace=True)

trades\_visu(trades,SelectData().period,SelectData().zone,SelectData().Earnings\_Provider,self.class\_path\_output)

trades.to\_excel(os.path.join(self.class\_path\_output,'Results\Trades\_LongShort\_'+ClassGlobal().Doc\_name+'.xlsx'))

#################

weights = strategy\_weights(trades,SelectData().strategy,prices,esp\_vol,get\_periods\_per\_year(SelectData().period),exp\_ret)

res = Results()

res.compute(trades,weights,prices,exp\_ret,get\_periods\_per\_year(SelectData().period))

plt.clf()

df = res.cumulative\_performance.to\_frame("Stratégie "+SelectData().strategy)

df = df[df.index>"2000"]

df\_perf = prices.join(df).dropna()/prices.join(df).dropna().iloc[0,:]

# plt.plot(res.cumulative\_performance,label="Performance cumulative de la stratégie "+ClassGlobal().strategy+" en zone "+ClassGlobal().zone)

df\_perf.plot()

plt.savefig(os.path.join(self.class\_path\_output,'Cumulative\_Perf'+ClassGlobal().Doc\_name+'.png'))

if \_\_name\_\_ == "\_\_main\_\_":

luigi.build([ClassGlobal(), SelectData()],local\_scheduler = True)

Collecte\_data\_Model\_Vola()

luigi.build([Arbitrage(Doc\_name=ClassGlobal().Doc\_name)],local\_scheduler = True,no\_lock=True)

# filename1=r"U:\GDA\PFC\03\_Gerants\03\_08\_AM\pyfinanceadvise\Simulation\Simulation\_007\DataSet\Model\_EDR\_Data\_EU\_mArbitrageFact\_EU007.h5"

# filename2=r"U:\GDA\PFC\03\_Gerants\03\_08\_AM\pyfinanceadvise\Simulation\Simulation\_03\_07\_US\DataSet\Model\_EDR\_Data\_US\_mArbitrageShiller\_US.h5"

# df,meta = h5load(filename1)

# print(df)

# print(meta)

Preparation dataset:

import luigi

import pandas as pd

import os

import datetime

import matplotlib.pyplot as plt

from gestion\_hdf5 import h5store, h5load

from global\_parameter import ClassGlobal, SelectData

from CPI\_download.CPI\_data\_download import Collecte\_CPI

from earnings\_download.earnings\_data\_download import Collecte\_Earnings

from rates\_download.rates\_data\_download import Collecte\_Rate

from prices\_download.prices\_data\_download import Collecte\_Prix

from real\_rates\_download.real\_rates\_data\_download import Collecte\_Infla

from vola\_download.vola\_data\_download import Collecte\_data\_Model\_Vola

import time

if \_\_name\_\_ == "\_\_main\_\_":

luigi.build([ClassGlobal(), SelectData()],local\_scheduler = True)

Collecte\_Infla()

Collecte\_CPI()

Collecte\_Earnings()

Collecte\_Prix()

Collecte\_Rate()

Gestion hdf55

import pandas as pd

def h5store(filename, df, dic):

store = pd.HDFStore(filename)

store.put('mydata', df)

store.get\_storer('mydata').attrs.metadata = dic

store.close()

def h5load(filename):

with pd.HDFStore(filename) as store:

data = store['mydata']

metadata = store.get\_storer('mydata').attrs.metadata

data.attrs = metadata

return data, metadata

luigi.cfg

[ClassGlobal]

path\_output=U:\GDA\PFC\02\_Ressources\Modele\_Arbitrage

simulation=20240906-Simulation-US-NB

Doc\_name=20240906-Simulation-US-NB

[SelectData]

zone=US

period =MONTHLY

start=19931231

#start=19931231

#chine mettre start = 20130430

end=20240930

#Earnings\_Provider=Bloomberg

#Earnings\_Provider=Factset

Earnings\_Provider=Shiller

Model\_Vol=Historical

strategy=Long\_Only

date1=2024-09-30

#date1 doit être dernier jour d'un mois

[StrategyPypfopt]

known\_future\_cov\_matrix = False

known\_future\_expected\_returns = False

rolling = 25

# (mois)

Rates\_download

import luigi

import datetime

import os

from collections import OrderedDict

from global\_parameter import ClassGlobal

from functions\_data import \_get\_generic\_series, end\_month\_data

from gestion\_hdf5 import h5store

import calendar

import pandas as pd

class Recuperation\_Rate\_Bloomberg(luigi.Task): #classe parent

class\_path\_output = luigi.Parameter()

zone = luigi.Parameter()

period= luigi.Parameter()

indice = luigi.Parameter(default = "GECU10YR Index") #indice devient un paramètre

def \_get\_px\_last(self,Rate): #recup prix

assets = OrderedDict()

assets["A1"]={}

assets["A1"]["ticker"]= Rate

assets["A1"]["des"]="Rate"

Prices = \_get\_generic\_series(assets, ["PX\_LAST"], self.period)

print(Prices)

return Prices

def output(self): #nom du fichier de sortie

suffix = ""

if self.period == "DAILY":

suffix ="\_d"

if self.period == "MONTHLY" :

suffix="\_m"

if self.period == "QUARTERLY" :

suffix="\_q"

if self.period == "YEARLY":

suffix = "\_y"

return luigi.LocalTarget(f'{self.class\_path\_output}\Rate\_'+ self.zone +suffix+'.h5')

def fetch\_data(self):

return self.\_get\_px\_last(self.indice), self.indice

def run(self):

data,indice1 = self.fetch\_data()

if self.period != "DAILY" :

data = end\_month\_data(data, force\_last\_date=True)

if self.period == "QUARTERLY":

last\_day\_of\_month = calendar.monthrange(data.index[-1].year, data.index[-1].month)[1]

last\_index = data.index[-1].replace(day=last\_day\_of\_month)

data = data.set\_index(data.index[:-1].append(pd.Index([last\_index])))

metadata = dict(data="Rate",provider="Bloomberg",indice=indice1)

h5store(self.output().path,data,metadata)

excel\_file = os.path.splitext(self.output().path)[0] + '.xlsx'

data.to\_excel(excel\_file)

def Collecte\_Rate():#fonction pour collecter les taux US et EUR

path\_init = os.path.join(ClassGlobal().path\_output,"Data\{}\Rates".format(datetime.datetime.now().strftime("%Y\_%m\_%d")))

class\_path\_output = path\_init

if not os.path.exists(class\_path\_output):

os.makedirs(class\_path\_output)

task\_eu\_d = Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="EU", period="DAILY",indice="GECU10YR Index")

task\_eu\_m = Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="EU", period="MONTHLY",indice="GECU10YR Index")

task\_eu\_q = Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="EU", period="QUARTERLY",indice="GECU10YR Index")

task\_eu\_y = Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="EU", period="YEARLY",indice="GECU10YR Index")

task\_us\_d = Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="US", period="DAILY",indice="USGG10YR Index")

task\_us\_m = Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="US", period="MONTHLY",indice="USGG10YR Index")

task\_us\_q = Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="US", period="QUARTERLY",indice="USGG10YR Index")

task\_us\_y = Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="US", period="YEARLY",indice="USGG10YR Index")

task\_uk\_d =Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="UK", period="DAILY",indice="GUKG10 Index")

task\_uk\_m =Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="UK", period="MONTHLY",indice="GUKG10 Index")

task\_uk\_q =Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="UK", period="QUARTERLY",indice="GUKG10 Index")

task\_uk\_y =Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="UK", period="YEARLY",indice="GUKG10 Index")

task\_cn\_d =Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="CN", period="DAILY",indice="GCNY10YR Index")

task\_cn\_m =Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="CN", period="MONTHLY",indice="GCNY10YR Index")

task\_cn\_q =Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="CN", period="QUARTERLY",indice="GCNY10YR Index")

task\_cn\_y =Recuperation\_Rate\_Bloomberg(class\_path\_output, zone="CN", period="YEARLY",indice="GCNY10YR Index")

luigi.build([task\_cn\_d,task\_cn\_m, task\_cn\_q, task\_cn\_y, task\_eu\_d,task\_eu\_m,task\_eu\_q,task\_eu\_y,task\_us\_d,task\_us\_m,task\_us\_q,task\_us\_y,task\_uk\_d,task\_uk\_m,task\_uk\_y,task\_uk\_q],local\_scheduler = True)

prices download :

import os

import datetime

from global\_parameter import ClassGlobal

import luigi

from collections import OrderedDict

from functions\_data import \_get\_generic\_series, end\_month\_data

from gestion\_hdf5 import h5store

import calendar

import pandas as pd

class Recuperation\_Prices\_Bloomberg(luigi.Task): #classe parent qui ressemble à Recuperation\_Earnings\_Bloomberg afin de récuperer les prix des assets

class\_path\_output = luigi.Parameter()

zone = luigi.Parameter()

period= luigi.Parameter()

indice = luigi.Parameter(default="SXXP Index")

indiceTR = luigi.Parameter(default="SXXR Index")

def \_get\_px\_last(self, indice,indiceTR): #fonction qui retourne les prix des deux indices mis en paramètres

assets = OrderedDict()

assets["A1"]={}

assets["A1"]["ticker"]= indice

assets["A1"]["des"]="Prices Bloomberg"

assets["A2"]={}

assets["A2"]["ticker"]= indiceTR

assets["A2"]["des"]="Prices Bloomberg TR"

Prices = \_get\_generic\_series(assets, ["PX\_LAST"], self.period)

return Prices #comme précédemment

def output(self): #nom du fichier de sortie qui varie selon si monthly ou quarterly

suffix = ""

if self.period == "DAILY" :

suffix="\_d"

if self.period == "MONTHLY" :

suffix="\_m"

if self.period == "QUARTERLY" :

suffix="\_q"

if self.period == "YEARLY":

suffix="\_y"

return luigi.LocalTarget(f'{self.class\_path\_output}\Bloomberg\_'+ self.zone +suffix+'.h5')

def fetch\_data(self):

return self.\_get\_px\_last(self.indice,self.indiceTR),self.indice,self.indiceTR

def run(self):

data,indice\_ExDiv,indice\_TR = self.fetch\_data()

if self.period != "DAILY" :

data = end\_month\_data(data, force\_last\_date=True) #vérifie l'exactitude des mois

# data.to\_hdf(self.output().path,key='df',mode="w")

if self.period == "QUARTERLY":

last\_day\_of\_month = calendar.monthrange(data.index[-1].year, data.index[-1].month)[1]

last\_index = data.index[-1].replace(day=last\_day\_of\_month)

data = data.set\_index(data.index[:-1].append(pd.Index([last\_index])))

metadata = dict(data="Prices",provider="Bloomberg",indice=indice\_ExDiv,indiceTR=indice\_TR)

h5store(self.output().path,data,metadata) #on enregistre les data final dans un hdf5

excel\_file = os.path.splitext(self.output().path)[0] + '.xlsx'

data.to\_excel(excel\_file)

def Collecte\_Prix():

path\_init = os.path.join(ClassGlobal().path\_output,"Data\{}\Prices".format(datetime.datetime.now().strftime("%Y\_%m\_%d")))

class\_path\_output = path\_init

if not os.path.exists(class\_path\_output):

os.makedirs(class\_path\_output)

task\_prices\_eu\_d = Recuperation\_Prices\_Bloomberg(class\_path\_output, zone = "EU", period="DAILY", indice ="SXXP Index", indiceTR = "SXXR Index")

task\_prices\_us\_d = Recuperation\_Prices\_Bloomberg(class\_path\_output,zone = "US",period="DAILY", indice = "SPX Index", indiceTR = "SPXT Index")

task\_prices\_uk\_d = Recuperation\_Prices\_Bloomberg(class\_path\_output,zone = "UK",period="DAILY", indice = "UKX Index", indiceTR = "UKXDUK Index")

task\_prices\_cn\_d = Recuperation\_Prices\_Bloomberg(class\_path\_output,zone = "CN",period="DAILY", indice = "M3CN Index", indiceTR = "NDEUCHF Index")

task\_prices\_eu\_m = Recuperation\_Prices\_Bloomberg(class\_path\_output, zone = "EU", period="MONTHLY", indice ="SXXP Index", indiceTR = "SXXR Index")

task\_prices\_us\_m = Recuperation\_Prices\_Bloomberg(class\_path\_output,zone = "US",period="MONTHLY", indice = "SPX Index", indiceTR = "SPXT Index")

task\_prices\_uk\_m = Recuperation\_Prices\_Bloomberg(class\_path\_output,zone = "UK",period="MONTHLY", indice = "UKX Index", indiceTR = "UKXDUK Index")

task\_prices\_cn\_m = Recuperation\_Prices\_Bloomberg(class\_path\_output,zone = "CN",period="MONTHLY", indice = "M3CN Index", indiceTR = "NDEUCHF Index")

task\_prices\_eu\_q = Recuperation\_Prices\_Bloomberg(class\_path\_output, zone = "EU", period="QUARTERLY", indice ="SXXP Index", indiceTR = "SXXR Index")

task\_prices\_us\_q = Recuperation\_Prices\_Bloomberg(class\_path\_output,zone = "US",period="QUARTERLY", indice = "SPX Index", indiceTR = "SPXT Index")

task\_prices\_uk\_q = Recuperation\_Prices\_Bloomberg(class\_path\_output,zone = "UK",period="QUARTERLY", indice = "UKX Index", indiceTR = "UKXDUK Index")

task\_prices\_cn\_q = Recuperation\_Prices\_Bloomberg(class\_path\_output,zone = "CN",period="QUARTERLY", indice = "M3CN Index", indiceTR = "NDEUCHF Index")

task\_prices\_eu\_y = Recuperation\_Prices\_Bloomberg(class\_path\_output, zone = "EU", period="YEARLY", indice ="SXXP Index", indiceTR = "SXXR Index")

task\_prices\_us\_y = Recuperation\_Prices\_Bloomberg(class\_path\_output,zone = "US",period="YEARLY", indice = "SPX Index", indiceTR = "SPXT Index")

task\_prices\_uk\_y = Recuperation\_Prices\_Bloomberg(class\_path\_output,zone = "UK",period="YEARLY", indice = "UKX Index", indiceTR = "UKXDUK Index")

task\_prices\_cn\_y = Recuperation\_Prices\_Bloomberg(class\_path\_output,zone = "CN",period="YEARLY", indice = "M3CN Index", indiceTR = "NDEUCHF Index")

luigi.build([task\_prices\_eu\_d,task\_prices\_us\_d,task\_prices\_uk\_d,task\_prices\_cn\_d,task\_prices\_eu\_m,task\_prices\_us\_m,task\_prices\_uk\_m,task\_prices\_cn\_m,task\_prices\_eu\_q,task\_prices\_us\_q,task\_prices\_uk\_q,task\_prices\_cn\_q,task\_prices\_eu\_y,task\_prices\_uk\_y,task\_prices\_us\_y,task\_prices\_cn\_y],local\_scheduler = True)

real rates download :

import os

import datetime

import luigi

from global\_parameter import ClassGlobal

from collections import OrderedDict

from functions\_data import \_get\_generic\_series, end\_month\_data

from gestion\_hdf5 import h5store

import calendar

import pandas as pd

class Recuperation\_Infla\_Bloomberg(luigi.Task): #classe parent pour récupérer l'inflation des pays

class\_path\_output = luigi.Parameter()

zone = luigi.Parameter()

period= luigi.Parameter()

indice = luigi.Parameter(default ="GFRN10 Index")

indiceTIPS = luigi.Parameter(default = "GFRGEN10 Index")

def \_get\_px\_last(self,TS,TIPS):#récup les prix à la fois des oat et oat sur inflation

assets = OrderedDict()

assets["A1"]={}

assets["A1"]["ticker"]= TS

assets["A1"]["des"]="TS"

assets["A2"]={}

assets["A2"]["ticker"]= TIPS

assets["A2"]["des"]="TIPS"

Prices = \_get\_generic\_series(assets, ["PX\_LAST"], self.period)

Prices["Expected Infla"]= Prices["TS"]-Prices["TIPS"] #expected inflation = TS-TIPS

return Prices

def output(self): #fonction pour fichier de sortie

suffix = ""

if self.period == "DAILY":

suffix ="\_d"

if self.period == "MONTHLY" :

suffix="\_m"

if self.period == "QUARTERLY" :

suffix="\_q"

if self.period == "YEARLY":

suffix = "\_y"

return luigi.LocalTarget(f'{self.class\_path\_output}\ExpInf\_'+ self.zone +suffix+'.h5')

def fetch\_data(self):

return self.\_get\_px\_last(self.indice,self.indiceTIPS),self.indice,self.indiceTIPS # Cette méthode doit être implémentée par chaque sous-classe

def run(self):

data,indice1,indiceTIPS = self.fetch\_data()

if self.period != "DAILY" :

data = end\_month\_data(data, force\_last\_date=True) #vérifier la fin de mois

if self.period == "QUARTERLY":

last\_day\_of\_month = calendar.monthrange(data.index[-1].year, data.index[-1].month)[1]

last\_index = data.index[-1].replace(day=last\_day\_of\_month)

data = data.set\_index(data.index[:-1].append(pd.Index([last\_index])))

metadata = dict(data="Inflation",provider="Bloomberg",indice=indice1,indiceTIPS=indiceTIPS)

h5store(self.output().path,data,metadata) #stocker en HDF5

excel\_file = os.path.splitext(self.output().path)[0] + '.xlsx'

data.to\_excel(excel\_file)

class Recuperation\_Infla\_Bloomberg\_CN(luigi.Task): #classe parent pour récupérer l'inflation des pays

class\_path\_output = luigi.Parameter()

zone = luigi.Parameter()

period= luigi.Parameter()

indice = luigi.Parameter(default ="GCNY10YR Index")

indiceCPI = luigi.Parameter(default = "ECOPCNN Index")

def \_get\_px\_last(self,bond,cpi):#récup les prix à la fois du cpi et du bond

assets = OrderedDict()

assets["A1"]={}

assets["A1"]["ticker"]= bond

assets["A1"]["des"]="BOND"

assets["A2"]={}

assets["A2"]["ticker"]= cpi

assets["A2"]["des"]="CPI"

Prices = \_get\_generic\_series(assets, ["PX\_LAST"], self.period)

Prices= Prices.ffill()

Prices = Prices.dropna()

Prices = end\_month\_data(Prices)

Prices = Prices[~Prices.index.duplicated(keep='first')]

index\_prices = Prices.index

if(self.period == 'MONTHLY'):

y = 12

if(self.period =='QUARTERLY'):

y = 4

if(self.period == 'YEARLY'):

y = 1

Prices = Prices.reset\_index(drop=False)

for i in range(y,len(Prices.index)) :

exp\_infla = ((Prices.loc[i, 'CPI'] /Prices.loc[i-y, 'CPI'])-1)\*100

Prices.loc[i,'Expected Infla'] = Prices.loc[i,'BOND'] - (exp\_infla)

Prices = Prices.set\_index('date')

return Prices

def output(self): #fonction pour fichier de sortie

suffix = ""

if self.period == "DAILY":

suffix ="\_d"

if self.period == "MONTHLY" :

suffix="\_m"

if self.period == "QUARTERLY" :

suffix="\_q"

if self.period == "YEARLY":

suffix = "\_y"

return luigi.LocalTarget(f'{self.class\_path\_output}\ExpInf\_'+ self.zone +suffix+'.h5')

def fetch\_data(self):

return self.\_get\_px\_last(self.indice,self.indiceCPI),self.indice,self.indiceCPI # Cette méthode doit être implémentée par chaque sous-classe

def run(self):

data,indice1,indiceTIPS = self.fetch\_data()

if self.period != "DAILY" :

data = end\_month\_data(data, force\_last\_date=True) #vérifier la fin de mois

if self.period == "QUARTERLY":

last\_day\_of\_month = calendar.monthrange(data.index[-1].year, data.index[-1].month)[1]

last\_index = data.index[-1].replace(day=last\_day\_of\_month)

data = data.set\_index(data.index[:-1].append(pd.Index([last\_index])))

metadata = dict(data="Inflation",provider="Bloomberg",indice=indice1,indiceTIPS=indiceTIPS)

h5store(self.output().path,data,metadata) #stocker en HDF5

excel\_file = os.path.splitext(self.output().path)[0] + '.xlsx'

data.to\_excel(excel\_file)

class Recuperation\_InflaSwap\_Bloomberg(luigi.Task): #classe pour recup inflaswap

path\_output = luigi.Parameter()

zone = luigi.Parameter()

period= luigi.Parameter()

InflaSwap = luigi.Parameter(default = "EUSWI10 BGN Curncy")

def \_get\_px\_last(self,InflaSwap): #recup les prix

assets = OrderedDict()

assets["A1"]={}

assets["A1"]["ticker"]= InflaSwap

assets["A1"]["des"]="InflaSwap"

InflaSwap = \_get\_generic\_series(assets, ["PX\_LAST"], self.period)

return InflaSwap

def output(self): #définition du nom du fichier

suffix = ""

if self.period == "DAILY":

suffix ="\_d"

if self.period == "MONTHLY" :

suffix="\_m"

if self.period == "QUARTERLY" :

suffix="\_q"

if self.period == "YEARLY":

suffix = "\_y"

return luigi.LocalTarget(f'{self.path\_output}\InflaSwap\_'+ self.zone +suffix+'.h5')

def fetch\_data(self):

return self.\_get\_px\_last(self.InflaSwap),self.InflaSwap

def run(self):

data,indice1 = self.fetch\_data()

if self.period != "DAILY" :

#data = end\_month\_data(data)

data = end\_month\_data(data, force\_last\_date=True)

if self.period == "QUARTERLY":

last\_day\_of\_month = calendar.monthrange(data.index[-1].year, data.index[-1].month)[1]

last\_index = data.index[-1].replace(day=last\_day\_of\_month)

data = data.set\_index(data.index[:-1].append(pd.Index([last\_index])))

# data.to\_hdf(self.output().path,key='df',mode="w")

metadata = dict(data="Inflation",provider="Bloomberg",indice=indice1)

h5store(self.output().path,data,metadata) #stocker dans un hdf5

excel\_file = os.path.splitext(self.output().path)[0] + '.xlsx'

data.to\_excel(excel\_file)

def Collecte\_Infla(): #fonction qui calculer les tasks inflation

path\_init = os.path.join(ClassGlobal().path\_output,"Data\{}\Inflation".format(datetime.datetime.now().strftime("%Y\_%m\_%d")))

# dirctory\_date = "Infla\_{}".format(datetime.datetime.now().strftime("%Y\_%m\_%d"))

# class\_path\_output = os.path.join(path\_init,dirctory\_date)

class\_path\_output = path\_init

if not os.path.exists(class\_path\_output):

os.makedirs(class\_path\_output)

task\_infla\_eu\_d = Recuperation\_Infla\_Bloomberg(class\_path\_output, zone="EU", period="DAILY", indice = "GFRN10 Index", indiceTIPS = "GFRGEN10 Index")

task\_infla\_eu\_m = Recuperation\_Infla\_Bloomberg(class\_path\_output, zone="EU", period="MONTHLY", indice = "GFRN10 Index", indiceTIPS = "GFRGEN10 Index")

task\_infla\_eu\_q = Recuperation\_Infla\_Bloomberg(class\_path\_output, zone="EU", period="QUARTERLY", indice = "GFRN10 Index", indiceTIPS = "GFRGEN10 Index")

task\_infla\_eu\_y = Recuperation\_Infla\_Bloomberg(class\_path\_output, zone="EU", period="YEARLY", indice = "GFRN10 Index", indiceTIPS = "GFRGEN10 Index")

task\_infla\_us\_d = Recuperation\_Infla\_Bloomberg(class\_path\_output, zone="US", period="DAILY", indice = "USGG10YR Index", indiceTIPS = "USGGT10Y Index")

task\_infla\_us\_m = Recuperation\_Infla\_Bloomberg(class\_path\_output, zone="US", period="MONTHLY", indice = "USGG10YR Index", indiceTIPS = "USGGT10Y Index")

task\_infla\_us\_q = Recuperation\_Infla\_Bloomberg(class\_path\_output, zone="US", period="QUARTERLY", indice = "USGG10YR Index", indiceTIPS = "USGGT10Y Index")

task\_infla\_us\_y = Recuperation\_Infla\_Bloomberg(class\_path\_output, zone="US", period="YEARLY", indice = "USGG10YR Index", indiceTIPS = "USGGT10Y Index")

#task\_infla\_cn\_d = Recuperation\_Infla\_Bloomberg\_CN(class\_path\_output, zone="CN", period="DAILY", indice = "GCNY10YR Index", indiceCPI = "ECOPCNN Index")

task\_infla\_cn\_m = Recuperation\_Infla\_Bloomberg\_CN(class\_path\_output, zone="CN", period="MONTHLY", indice = "GCNY10YR Index", indiceCPI = "ECOPCNN Index")

task\_infla\_cn\_q = Recuperation\_Infla\_Bloomberg\_CN(class\_path\_output, zone="CN", period="QUARTERLY", indice = "GCNY10YR Index", indiceCPI = "ECOPCNN Index")

task\_infla\_cn\_y = Recuperation\_Infla\_Bloomberg\_CN(class\_path\_output, zone="CN", period="YEARLY", indice = "GCNY10YR Index", indiceCPI = "ECOPCNN Index")

task\_inflaswap\_eu\_d = Recuperation\_InflaSwap\_Bloomberg(class\_path\_output,zone="EU",period="DAILY", InflaSwap="EUSWI10 BGN Curncy")

task\_inflaswap\_eu\_m = Recuperation\_InflaSwap\_Bloomberg(class\_path\_output,zone="EU",period="MONTHLY", InflaSwap="EUSWI10 BGN Curncy")

task\_inflaswap\_eu\_q = Recuperation\_InflaSwap\_Bloomberg(class\_path\_output,zone="EU",period="QUARTERLY", InflaSwap="EUSWI10 BGN Curncy")

task\_inflaswap\_eu\_y = Recuperation\_InflaSwap\_Bloomberg(class\_path\_output,zone="EU",period="YEARLY", InflaSwap="EUSWI10 BGN Curncy")

luigi.build([task\_infla\_cn\_m,task\_infla\_cn\_q,task\_infla\_cn\_y,task\_infla\_eu\_d, task\_infla\_eu\_m,task\_infla\_eu\_q,task\_infla\_eu\_y,task\_infla\_us\_d,task\_infla\_us\_m,task\_infla\_us\_q,task\_infla\_us\_y, task\_inflaswap\_eu\_d,task\_inflaswap\_eu\_m,task\_inflaswap\_eu\_q,task\_inflaswap\_eu\_y],local\_scheduler = True)

functions\_assets :

import pandas as pd

from functions\_data import \_get\_generic\_series2

from tia.bbg import LocalTerminal

from collections import OrderedDict

import numpy as np

def get\_assets(zone):

assets = OrderedDict()

if (zone=="US"):

assets['LUATTRUU'] = {}

assets['LUATTRUU']['ticker'] = "LUATTRUU Index"

assets['LUATTRUU']['des'] = "Treasuries US"

assets['LUACTRUU'] = {}

assets['LUACTRUU']['ticker'] = "LUACTRUU Index"

assets['LUACTRUU']['des'] = "HG Corpo US"

assets['LF98TRUU'] = {}

assets['LF98TRUU']['ticker'] = "LF98TRUU Index"

assets['LF98TRUU']['des'] = "HY Corpo US"

assets['SPXT'] = {}

assets['SPXT']['ticker'] = "SPXT Index"

assets['SPXT']['des'] = "Equity US"

assets['US0001M'] = {}

# assets['USC0TR03']['ticker'] = "USC0TR03 Index"

assets['US0001M']['ticker'] = "US0001M Index"

assets['US0001M']['des'] = "Cash US"

if (zone=="EU"):

assets['LEATTREU'] = {}

assets['LEATTREU']['ticker'] = "LEATTREU Index"

assets['LEATTREU']['des'] = "Treasuries EU"

assets['LECPTREU'] = {}

assets['LECPTREU']['ticker'] = "LECPTREU Index"

assets['LECPTREU']['des'] = "HG Corpo EU"

assets['I02501EU'] = {}

assets['I02501EU']['ticker'] = "I02501EU Index"

assets['I02501EU']['des'] = "HY Corpo EU"

assets['SXXP'] = {}

assets['SXXP']['ticker'] = "SXXR Index"

assets['SXXP']['des'] = "Equity EU"

# assets['Bloom'] = {}

# assets['Bloom']['ticker'] = "SX5T Index"

# assets['Bloom']['des'] = "Equity EU Bloom"

# assets['SCXP'] = {}

# assets['SCXP']['ticker'] = "SCXP Index"

# assets['SCXP']['des'] = "Small Cap EU"

# assets['EUROT'] = {}

# assets['EUROT']['ticker'] = "EUROT Index"

# assets['EUROT']['des'] = "Large Cap EU"

assets['ECC0TR03'] = {}

# assets['ECC0TR03']['ticker'] = "ECC0TR03 Index"

assets['ECC0TR03']['ticker'] = "ECC0TR03 Index"

assets['ECC0TR03']['des'] = "Cash EU"

if (zone=="FR"):

assets['LTFRTREU'] = {}

assets['LTFRTREU']['ticker'] = "LTFRTREU Index"

assets['LTFRTREU']['des'] = "Treasuries FR"

assets['LECPTREU'] = {}

assets['LECPTREU']['ticker'] = "LECPTREU Index"

assets['LECPTREU']['des'] = "HG Corpo FR"

assets['I02501EU'] = {}

assets['I02501EU']['ticker'] = "I02501EU Index"

assets['I02501EU']['des'] = "HY Corpo FR"

assets['NCAC'] = {}

assets['NCAC']['ticker'] = "NCAC Index"

assets['NCAC']['des'] = "Equity FR"

assets['ECC0TR03'] = {}

assets['ECC0TR03']['ticker'] = "ECC0TR03 Index"

assets['ECC0TR03']['des'] = "Cash FR"

if (zone=="DE"):

assets['LETGTREU'] = {}

assets['LETGTREU']['ticker'] = "LETGTREU Index"

assets['LETGTREU']['des'] = "Treasuries DE"

assets['LECPTREU'] = {}

assets['LECPTREU']['ticker'] = "LECPTREU Index"

assets['LECPTREU']['des'] = "HG Corpo DE"

assets['I02501EU'] = {}

assets['I02501EU']['ticker'] = "I02501EU Index"

assets['I02501EU']['des'] = "HY Corpo DE"

assets['DAX'] = {}

assets['DAX']['ticker'] = "DAX Index"

assets['DAX']['des'] = "Equity DE"

assets['ECC0TR03'] = {}

assets['ECC0TR03']['ticker'] = "ECC0TR03 Index"

assets['ECC0TR03']['des'] = "Cash DE"

if (zone=="UK"):

assets['H09027CH'] = {}

assets['H09027CH']['ticker'] = "H09027CH Index"

# assets['FTFIRDY7']['ticker'] = "FTFIRDY7 Index"

assets['H09027CH']['des'] = "Treasuries UK"

assets['I17389GB'] = {}

assets['I17389GB']['ticker'] = "I17389GB Index"

assets['I17389GB']['des'] = "HG Corpo UK"

assets['CBPDHYI'] = {}

assets['CBPDHYI']['ticker'] = "I05892GB Index"

assets['CBPDHYI']['des'] = "HY Corpo UK"

assets['TUKXG'] = {}

assets['TUKXG']['ticker'] = "TUKXG Index"

assets['TUKXG']['des'] = "Equity UK"

assets['DBMMSONI'] = {}

assets['DBMMSONI']['ticker'] = "DBMMSONI Index"

assets['DBMMSONI']['des'] = "Cash UK"

if (zone=="CN"):

assets['I32561US'] = {}

assets['I32561US']['ticker'] = "I32561US Index"

assets['I32561US']['des'] = "Treasuries CN"

assets['JBMXCNTR'] = {}

assets['JBMXCNTR']['ticker'] = "JBMXCNTR Index"

assets['JBMXCNTR']['des'] = "HG Corpo CN"

assets['JBMQCNTR'] = {}

assets['JBMQCNTR']['ticker'] = "JBMQCNTR Index"

assets['JBMQCNTR']['des'] = "HY Corpo CN"

assets['M3CN'] = {}

assets['M3CN']['ticker'] = "M3CN Index"

assets['M3CN']['des'] = "Equity CN"

assets['CHBM7D'] = {}

assets['CHBM7D']['ticker'] = "CHBM7D Index"

assets['CHBM7D']['des'] = "Cash CN"

return assets

def get\_prices(start, end, period,zone):

assets = get\_assets(zone)

print(period)

prices = \_get\_generic\_series2(assets, ['PX\_LAST'], start, end, period)

prices.attrs = assets

print('okkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk')

print(prices)

return prices

def get\_short\_term\_yields\_us(start, end, period, \*\*kwargs):

indexes = ['US0001M Index', 'SOFRRATE Index']

resp = LocalTerminal.get\_historical(indexes, 'PX\_LAST',

start=start, end=end, period=period, \*\*kwargs)

data = resp.as\_frame()

data.columns = data.columns.droplevel(1)

str\_series = pd.Series(index=data.index, name='ST index', dtype='float64')

ind = np.isnan(data[indexes[1]])

str\_series.loc[~ind] = data.loc[~ind, indexes[1]]

str\_series.loc[ind] = data.loc[ind, indexes[0]]

return str\_series

def get\_short\_term\_yields\_eu(start, end, period, \*\*kwargs):

indexes = ['EONIA Index', 'ESTRON Index']

resp = LocalTerminal.get\_historical(indexes, 'PX\_LAST',

start=start, end=end, period=period, \*\*kwargs)

data = resp.as\_frame()

data.columns = data.columns.droplevel(1)

str\_series = pd.Series(index=data.index, name='ST index', dtype='float64')

ind = np.isnan(data[indexes[1]])

str\_series.loc[~ind] = data.loc[~ind, indexes[1]]

str\_series.loc[ind] = data.loc[ind, indexes[0]]

return str\_series

def get\_short\_term\_yields\_uk(start, end, period, \*\*kwargs):

indexes = OrderedDict()

indexes['SONIO'] = {}

indexes['SONIO']['ticker'] = "SONIO/N Index"

indexes['SONIO']['des'] = "Cash UK"

str\_series = \_get\_generic\_series2(indexes, ['PX\_LAST'], start, end, period)

return str\_series

def get\_short\_term\_yields\_cn(start, end, period, \*\*kwargs):

indexes = OrderedDict()

indexes['CHBM7D'] = {}

indexes['CHBM7D']['ticker'] = "CHBM7D Index"

indexes['CHBM7D']['des'] = "Cash CN"

str\_series = \_get\_generic\_series2(indexes, ['PX\_LAST'], start, end, period)

return str\_series

def get\_short\_term\_yields(zone,start, end, period, \*\*kwargs):

if(zone=="US"):

yields = get\_short\_term\_yields\_us(start, end, period)

if(zone=="EU"):

yields = get\_short\_term\_yields\_eu(start, end, period)

if(zone=="UK"):

yields = get\_short\_term\_yields\_uk(start, end, period)

if(zone=="CN"):

yields = get\_short\_term\_yields\_cn(start, end, period)

return yields

def get\_assets\_yields(assets, start, end, period):

return \_get\_generic\_series2(assets, ['BX207'], start, end, period)

def get\_yields(start, end, period,zone,BDD):

assets = get\_assets(zone)

yields = get\_assets\_yields(assets, start, end, period)

yields["Cash "+zone] = get\_short\_term\_yields(zone, start, end, period)

# For HG Corporate Bonds : Yield less 50bp to take into account the average historical loss due to downgrades.

if(zone=='CN'):

assets ={}

assets['JBMXCNSW'] = {}

assets['JBMXCNSW']['ticker'] = "JBMXCNSW Index"

assets['JBMXCNSW']['des'] = "IG Spread"

assets['JBMQCNSW'] = {}

assets['JBMQCNSW']['ticker'] = "JBMQCNSW Index"

assets['JBMQCNSW']['des'] = "HY Spread"

assets['GCNY10YR'] = {}

assets['GCNY10YR']['ticker'] = "GCNY10YR Index"

assets['GCNY10YR']['des'] = "Bond 10Y"

hy\_ig = \_get\_generic\_series2(assets, ['PX\_LAST'], start, end, period)

yields["HG Corpo "+zone] =(hy\_ig['IG Spread']/100)+ hy\_ig['Bond 10Y']

yields["HY Corpo "+zone] =(hy\_ig['HY Spread']/100)+ hy\_ig['Bond 10Y']

yields["HG Corpo "+zone]=yields["HG Corpo "+zone]-0.5

# For HY Corporate Bonds : Yield less 250bp to take into account the average historical loss due to defaults.

yields["HY Corpo "+zone]=yields["HY Corpo "+zone]-2.5

# yields["Equity "+zone] = get\_EDR("SPX Index",zone,"MONTHLY",start,end)

yields.attrs = assets

return yields

function\_data :

import numpy as np

import pandas as pd

from tia.bbg import LocalTerminal

from collections import OrderedDict

import pickle

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

from sklearn.metrics import r2\_score

from datetime import datetime

from tia.bbg import LocalTerminal

import datetime

from datetime import date

import os

import luigi

import matplotlib.dates as mdates

def \_get\_generic\_series2(assets, fields, start, end, period, \*\*kwargs):

resp = LocalTerminal.get\_historical([assets[i]["ticker"] for i in assets.keys()],#utilise tia de bloom pour récupérer données historiques pour l'ensemble des assets détenus dans l'indice

fields, start=start, end=end, period=period, \*\*kwargs) #à partir d'une date de début et de fin

data = resp.as\_frame() #stocke dans data frame

data.columns = data.columns.droplevel(1) #on supprime la col 1

data.ffill(inplace=True) #rempli les valeurs Nan avec la méthode forward fill

data.rename(columns=\_build\_easy\_map(assets), inplace=True) #rename les colonnes avec la méthode \_build\_easy\_map

return data #retourne le dataframe

def \_get\_generic\_series(assets, fields, period, \*\*kwargs):

#if(fields[0] =='T12\_EPS\_AGGTE' and assets['']["ticker"] == 'MXCN Index' ): #case where we need to convert in CNY currency

# resp = LocalTerminal.get\_historical([assets[i]["ticker"] for i in assets.keys()],fields, start='1/1/1900', end = '1/1/2030', period=period, currency ='CNY')

#else:

resp = LocalTerminal.get\_historical([assets[i]["ticker"] for i in assets.keys()],fields, start='1/1/1900', end = '1/1/2030', period=period, \*\*kwargs) #à partir d'une date de début et de fin

data = resp.as\_frame() #stocke dans data frame

data.columns = data.columns.droplevel(1) #on supprime la col 1

print(data)

data.ffill(inplace=True) #rempli les valeurs Nan avec la méthode forward fill

data.rename(columns=\_build\_easy\_map(assets), inplace=True) #rename les colonnes avec la méthode \_build\_easy\_map

return data #retourne le dataframe

def \_build\_easy\_map(assets):

assets\_easy\_map = {}

for k in assets.keys(): #récupère les tickers des assets

assets\_easy\_map[assets[k]['ticker']] = assets[k]['des']

return assets\_easy\_map

def end\_month\_data(df, force\_last\_date=False): #vérifie si les mois sont bien sont succesifs

df = df.reset\_index("date")

last\_date = df["date"].iloc[-1]

print(last\_date)

if force\_last\_date:

df["date"] = df["date"].apply(lambda x: x+pd.offsets.MonthEnd(1) if (x.month == (x+pd.offsets.MonthEnd(1)).month) else x)

else:

df["date"] = df["date"].apply(lambda x: x+pd.offsets.MonthEnd(1) if (x.month == (x+pd.offsets.MonthEnd(1)).month and x!=last\_date) else x)

df = df.set\_index("date")

return df

'''assets = OrderedDict()

assets[""]={}

assets[""]["ticker"]= "EACPI Index"

assets[""]["des"]="CPI Bloomberg"

Prices = \_get\_generic\_series(assets, ["PX\_LAST"], "MONTHLY")

print(Prices)'''

convert to pdf :

import os

from selenium import webdriver

from selenium.webdriver.edge.service import Service

import json

import time

from selenium.common.exceptions import TimeoutException

import shutil

from datetime import datetime, timedelta

#initialise les paths correspondants à nos chemins

edge\_service = Service(r"C:\Users\nbonneau\Downloads\edgedriver\_win64\msedgedriver.exe")

path\_files\_to\_convert = r"C:\Local\xhtml2pdf"

savefile\_directory = r"C:\Local\xhtml2pdf\dl"

errors = savefile\_directory + r"\errors\_convert\_to\_pdf.txt"

#state pour choisir l'imprimante save as pdf

app\_state = {

"recentDestinations": [

{

"id": "Save as PDF",

"origin": "local",

"account": ""

}

],

"selectedDestinationId": "Save as PDF",

"version": 2

}

i = 0

# itérer nos files présents

directory = path\_files\_to\_convert

for filename in os.listdir(directory):

if filename.endswith(".html") or filename.endswith(".xhtml"):

#new

prefs = {

'printing.print\_preview\_sticky\_settings.appState': json.dumps(app\_state),

'savefile.default\_directory': savefile\_directory,

'savefile.default\_filename': filename

}

options = webdriver.EdgeOptions()

options.add\_experimental\_option('prefs', prefs)

options.add\_argument('--kiosk-printing')

#end

i= i+1

print("helloooo i ==",i)

if(i>0):

file\_path = os.path.join(directory, filename)

print(f"Processing file: {filename}")

#poids du file

file\_size\_bytes = os.path.getsize(file\_path)

file\_size\_mb = file\_size\_bytes / (1024 \*\* 2)

print(file\_size\_mb)

driver = webdriver.Edge(service=edge\_service, options=options)

driver.get(file\_path)

#driver.set\_script\_timeout(10)

#driver.execute\_async\_script('window.print()')

if file\_size\_mb < 5 :

driver.execute\_script('window.print()')

else :

try:

driver.set\_script\_timeout(35+file\_size\_mb)

driver.execute\_async\_script('window.print()')

except TimeoutException:

print(f"Timed out while printing {filename}")

driver.quit()

old\_filename = max([savefile\_directory + "\\" + f for f in os.listdir(savefile\_directory)],key=os.path.getctime)

current\_time = datetime.now()

file\_creation\_time = datetime.fromtimestamp(os.path.getctime(old\_filename))

time\_difference = current\_time - file\_creation\_time

if time\_difference.total\_seconds() <= 32+file\_size\_mb:

new\_filename = filename + '.pdf'

shutil.move(old\_filename, os.path.join(savefile\_directory, new\_filename))

print(f"Le fichier '{os.path.basename(old\_filename)}' a été renommé en '{new\_filename}'.")

else:

print("Aucun fichier trouvé avec une heure de création égale à l'heure actuelle ou à la dernière minute.")

full\_name = os.path.join(savefile\_directory, filename) +".pdf"

if os.path.exists(full\_name):

pass

else:

with open(errors , "a") as error\_log:

error\_log.write(f"{filename} \n")

print(f"Finished processing file: {filename}")

Jp download :

from selenium import webdriver

from selenium.webdriver.edge.service import Service

from selenium.webdriver.edge.options import Options

import time

import config as cf

from selenium.webdriver.common.by import By

from selenium.webdriver.common.action\_chains import ActionChains

import os

import datetime

import shutil

from selenium.webdriver.support.ui import Select

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

import pandas as pd

def market\_jp(path\_for\_saving, path\_default\_download, date):

options = webdriver.EdgeOptions()

options.add\_argument(f"user-data-dir={cf.local['userDataDir']}")

service = Service(cf.local["executablePath"])

driver = webdriver.Edge(service=service, options=options)

date\_init = datetime.datetime.now()

#ouvrir le site

driver.get("https://markets.jpmorgan.com/research/CFP?page=EMBI")

wait = WebDriverWait(driver, 10)

#cliquer sur le calendrier

td\_element = driver.find\_element(By.ID, "historicalDate")

td\_element.click()

#identifier la date mise en paramètre

year = int(date[:4])

month = int(date[4:6].lstrip('0'))

day = int(date[6:8].lstrip('0'))

wait = WebDriverWait(driver, 10)

try :

#choisir le bon mois

select\_month = wait.until(

EC.visibility\_of\_element\_located((By.CLASS\_NAME, "ui-datepicker-month"))

)

month\_option = select\_month.find\_element(By.XPATH, f".//option[@value='{month-1}']")

month\_option.click()

#choisir la bonne année

select\_year = wait.until(

EC.visibility\_of\_element\_located((By.CLASS\_NAME, "ui-datepicker-year"))

)

year\_option = select\_year.find\_element(By.XPATH, f".//option[@value='{year}']")

year\_option.click()

#choisir le bon jour

day\_td = wait.until(EC.element\_to\_be\_clickable((By.XPATH, f"//td[@data-handler='selectDay' and .//a[text()='{day}']]")))

day\_td.click()

except Exception as e:

#affiche la date si elle n'est pas disponible sur le site (que 3 mois d'historical data disponible )

print(f"Une erreur est survenue : {e}")

print("La date d/m/y suivante n'est pas disponible", day,"/",month, "/", year)

time.sleep(5)

date\_init = datetime.datetime.now()

#EMBI GLOBAL DIV

embi\_global\_row = wait.until(

EC.presence\_of\_element\_located((By.XPATH, "//td[text()='EMBI Global Div.']/.."))

)

#embi global div composition

composition\_cell = embi\_global\_row.find\_element(By.CLASS\_NAME, "EMBI\_index\_historical\_data\_COMPOSITION")

download\_link = composition\_cell.find\_element(By.TAG\_NAME, "a")

download\_link.click()

time.sleep(2)

#embi global div sub-index

composition\_cell = embi\_global\_row.find\_element(By.CLASS\_NAME, "EMBI\_index\_historical\_data\_SUB\_INDEX")

download\_link = composition\_cell.find\_element(By.TAG\_NAME, "a")

download\_link.click()

time.sleep(2)

#embi global div monthly country weights

composition\_cell = embi\_global\_row.find\_element(By.CLASS\_NAME, "EMBI\_index\_historical\_data\_MONTHLY\_COUNTRY\_WEIGHTS")

download\_link = composition\_cell.find\_element(By.TAG\_NAME, "a")

download\_link.click()

time.sleep(2)

#embi global div monthly country preview

composition\_cell = embi\_global\_row.find\_element(By.CLASS\_NAME, "EMBI\_index\_historical\_data\_PREVIEW")

download\_link = composition\_cell.find\_element(By.TAG\_NAME, "a")

download\_link.click()

time.sleep(2)

#EMBI GLOBAL

embi\_global\_row = wait.until(

EC.presence\_of\_element\_located((By.XPATH, "//td[text()='EMBI Global']/.."))

)

#embi global composition

composition\_cell = embi\_global\_row.find\_element(By.CLASS\_NAME, "EMBI\_index\_historical\_data\_COMPOSITION")

download\_link = composition\_cell.find\_element(By.TAG\_NAME, "a")

download\_link.click()

time.sleep(2)

#embi global sub-index

composition\_cell = embi\_global\_row.find\_element(By.CLASS\_NAME, "EMBI\_index\_historical\_data\_SUB\_INDEX")

download\_link = composition\_cell.find\_element(By.TAG\_NAME, "a")

download\_link.click()

time.sleep(2)

#embi global monthly country weights

composition\_cell = embi\_global\_row.find\_element(By.CLASS\_NAME, "EMBI\_index\_historical\_data\_MONTHLY\_COUNTRY\_WEIGHTS")

download\_link = composition\_cell.find\_element(By.TAG\_NAME, "a")

download\_link.click()

time.sleep(2)

#embi global monthly country preview

composition\_cell = embi\_global\_row.find\_element(By.CLASS\_NAME, "EMBI\_index\_historical\_data\_PREVIEW")

download\_link = composition\_cell.find\_element(By.TAG\_NAME, "a")

download\_link.click()

time.sleep(2)

#EMBI+

embi\_global\_row = wait.until(

EC.presence\_of\_element\_located((By.XPATH, "//td[text()='EMBI Global']/.."))

)

#embi + composition

composition\_cell = embi\_global\_row.find\_element(By.CLASS\_NAME, "EMBI\_index\_historical\_data\_COMPOSITION")

download\_link = composition\_cell.find\_element(By.TAG\_NAME, "a")

download\_link.click()

time.sleep(2)

#embi + sub-index

composition\_cell = embi\_global\_row.find\_element(By.CLASS\_NAME, "EMBI\_index\_historical\_data\_SUB\_INDEX")

download\_link = composition\_cell.find\_element(By.TAG\_NAME, "a")

download\_link.click()

time.sleep(2)

#embi + monthly country preview

composition\_cell = embi\_global\_row.find\_element(By.CLASS\_NAME, "EMBI\_index\_historical\_data\_PREVIEW")

download\_link = composition\_cell.find\_element(By.TAG\_NAME, "a")

download\_link.click()

time.sleep(2)

#EMBIG Core

embi\_global\_row = wait.until(

EC.presence\_of\_element\_located((By.XPATH, "//td[text()='EMBIG Core']/.."))

)

# EMBIG Core composition

composition\_cell = embi\_global\_row.find\_element(By.CLASS\_NAME, "EMBI\_index\_historical\_data\_COMPOSITION")

download\_link = composition\_cell.find\_element(By.TAG\_NAME, "a")

download\_link.click()

time.sleep(2)

#EMBIG Core monthly country preview

composition\_cell = embi\_global\_row.find\_element(By.CLASS\_NAME, "EMBI\_index\_historical\_data\_PREVIEW")

download\_link = composition\_cell.find\_element(By.TAG\_NAME, "a")

download\_link.click()

time.sleep(2)

time.sleep(10)

#créer des tuples avec le nom des fichiers et la date de création du fichier

file\_names\_with\_timestamps = []

download\_folder = path\_default\_download

now = datetime.datetime.now()

num\_files\_downloaded = 0

for filename in os.listdir(download\_folder):

if os.path.getmtime(os.path.join(download\_folder, filename)) > date\_init.timestamp() and os.path.getmtime(os.path.join(download\_folder, filename)) < now.timestamp():

file\_names\_with\_timestamps.append((filename, os.path.getmtime(os.path.join(download\_folder, filename))))

#faire une liste avec le nom des fichiers rangés par date d'anciennete dans file\_name (du plus vieux au plus jeune)

file\_names\_with\_timestamps.sort(key=lambda x: x[1], reverse=True)

file\_names = [x[0] for x in file\_names\_with\_timestamps]

file\_names = file\_names[::-1]

print(f"Nombre de fichiers téléchargés: {len(file\_names)}")

#13 fichiers attendus

if(len(file\_names)!=13):

print("erreur dans le chargement des fichiers, ces fichiers ont été downloads correctement :")

print(file\_names)

raise Exception("Impossible de télécharger tous les fichiers nécessaires.")

for file\_name in file\_names:

print(file\_name)

#identifier les noms des dossiers

embi\_global\_div\_dir = os.path.join(path\_for\_saving, "EMBI Global Div.")

embi\_global\_dir = os.path.join(path\_for\_saving, "EMBI Global")

embi\_plus\_dir = os.path.join(path\_for\_saving, "EMBI+")

embig\_core\_dir = os.path.join(path\_for\_saving, "EMBIG Core")

embi\_global\_div\_compo = os.path.join(embi\_global\_div\_dir, "Composition")

embi\_global\_div\_sub = os.path.join(embi\_global\_div\_dir, "Sub-Index")

embi\_global\_div\_monthly\_c\_w = os.path.join(embi\_global\_div\_dir, "Monthly-Country-Weights")

embi\_global\_div\_preview = os.path.join(embi\_global\_div\_dir, "Preview")

embi\_global\_compo = os.path.join(embi\_global\_dir, "Composition")

embi\_global\_sub = os.path.join(embi\_global\_dir, "Sub-Index")

embi\_global\_monthly\_c\_w = os.path.join(embi\_global\_dir, "Monthly-Country-Weights")

embi\_global\_preview = os.path.join(embi\_global\_dir, "Preview")

embi\_plus\_compo = os.path.join(embi\_plus\_dir, "Composition")

embi\_plus\_sub = os.path.join(embi\_plus\_dir, "Sub-Index")

embi\_plus\_preview = os.path.join(embi\_plus\_dir, "Preview")

embig\_core\_compo = os.path.join(embig\_core\_dir, "Composition")

embig\_core\_preview = os.path.join(embig\_core\_dir, "Preview")

#création des dossiers à condition qu'il n'existe déja pas

if not os.path.exists(embi\_global\_div\_dir):

os.makedirs(embi\_global\_div\_dir)

if not os.path.exists(embi\_global\_dir):

os.makedirs(embi\_global\_dir)

if not os.path.exists(embi\_global\_div\_monthly\_c\_w):

os.makedirs(embi\_global\_div\_monthly\_c\_w)

if not os.path.exists(embig\_core\_dir):

os.makedirs(embig\_core\_dir)

if not os.path.exists(embi\_global\_div\_compo):

os.makedirs(embi\_global\_div\_compo)

if not os.path.exists(embi\_global\_div\_sub):

os.makedirs(embi\_global\_div\_sub)

if not os.path.exists(embi\_plus\_dir):

os.makedirs(embi\_plus\_dir)

if not os.path.exists(embi\_global\_div\_preview):

os.makedirs(embi\_global\_div\_preview)

if not os.path.exists(embi\_global\_compo):

os.makedirs(embi\_global\_compo)

if not os.path.exists(embi\_global\_sub):

os.makedirs(embi\_global\_sub)

if not os.path.exists(embi\_global\_monthly\_c\_w):

os.makedirs(embi\_global\_monthly\_c\_w)

if not os.path.exists(embi\_global\_preview):

os.makedirs(embi\_global\_preview)

if not os.path.exists(embi\_plus\_compo):

os.makedirs(embi\_plus\_compo)

if not os.path.exists(embi\_plus\_sub):

os.makedirs(embi\_plus\_sub)

if not os.path.exists(embi\_plus\_preview):

os.makedirs(embi\_plus\_preview)

if not os.path.exists(embig\_core\_compo):

os.makedirs(embig\_core\_compo)

if not os.path.exists(embig\_core\_preview):

os.makedirs(embig\_core\_preview)

#renommer les fichiers correctement

new\_file\_names = []

for i, file\_name in enumerate(file\_names):

if i == 0:

new\_file\_name = f"{date}-jpmm-EMBI Global Div.-Composition.xlsx"

new\_file\_names.append(new\_file\_name)

source\_path = os.path.join(download\_folder, file\_name)

excel\_file = os.path.splitext(source\_path)[0] + ".xlsx"

df = pd.read\_csv(source\_path, delimiter=',')

df.to\_excel(excel\_file, index=False)

source\_path = excel\_file

target\_path = os.path.join(embi\_global\_div\_compo, new\_file\_name)

shutil.move(source\_path, target\_path)

elif i == 1:

new\_file\_name = f"{date}-jpmm-EMBI Global Div.-Sub-Index.xlsx"

new\_file\_names.append(new\_file\_name)

source\_path = os.path.join(download\_folder, file\_name)

excel\_file = os.path.splitext(source\_path)[0] + ".xlsx"

df = pd.read\_csv(source\_path, delimiter=',')

df.to\_excel(excel\_file, index=False)

source\_path = excel\_file

target\_path = os.path.join(embi\_global\_div\_sub, new\_file\_name)

shutil.move(source\_path, target\_path)

elif i == 2:

new\_file\_name = f"{date}-jpmm-EMBI Global Div.-Monthly-Country-Weights.xlsx"

new\_file\_names.append(new\_file\_name)

source\_path = os.path.join(download\_folder, file\_name)

excel\_file = os.path.splitext(source\_path)[0] + ".xlsx"

df = pd.read\_csv(source\_path, delimiter=',')

df.to\_excel(excel\_file, index=False)

source\_path = excel\_file

target\_path = os.path.join(embi\_global\_div\_monthly\_c\_w, new\_file\_name)

shutil.move(source\_path, target\_path)

elif i == 3:

new\_file\_name = f"{date}-jpmm-EMBI Global Div.-Preview.xlsx"

new\_file\_names.append(new\_file\_name)

source\_path = os.path.join(download\_folder, file\_name)

excel\_file = os.path.splitext(source\_path)[0] + ".xlsx"

df = pd.read\_csv(source\_path, delimiter=',')

df.to\_excel(excel\_file, index=False)

source\_path = excel\_file

target\_path = os.path.join(embi\_global\_div\_preview, new\_file\_name)

shutil.move(source\_path, target\_path)

elif i == 4:

new\_file\_name = f"{date}-jpmm-EMBI Global-Composition.xlsx"

new\_file\_names.append(new\_file\_name)

source\_path = os.path.join(download\_folder, file\_name)

excel\_file = os.path.splitext(source\_path)[0] + ".xlsx"

df = pd.read\_csv(source\_path, delimiter=',')

df.to\_excel(excel\_file, index=False)

source\_path = excel\_file

target\_path = os.path.join(embi\_global\_compo, new\_file\_name)

shutil.move(source\_path, target\_path)

elif i == 5:

new\_file\_name = f"{date}-jpmm-EMBI Global-Sub-Index.xlsx"

new\_file\_names.append(new\_file\_name)

source\_path = os.path.join(download\_folder, file\_name)

excel\_file = os.path.splitext(source\_path)[0] + ".xlsx"

df = pd.read\_csv(source\_path, delimiter=',')

df.to\_excel(excel\_file, index=False)

source\_path = excel\_file

target\_path = os.path.join(embi\_global\_sub, new\_file\_name)

shutil.move(source\_path, target\_path)

elif i == 6:

new\_file\_name = f"{date}-jpmm-EMBI Global-Monthly-Country-Weights.xlsx"

new\_file\_names.append(new\_file\_name)

source\_path = os.path.join(download\_folder, file\_name)

excel\_file = os.path.splitext(source\_path)[0] + ".xlsx"

df = pd.read\_csv(source\_path, delimiter=',')

df.to\_excel(excel\_file, index=False)

source\_path = excel\_file

target\_path = os.path.join(embi\_global\_monthly\_c\_w, new\_file\_name)

shutil.move(source\_path, target\_path)

elif i == 7:

new\_file\_name = f"{date}-jpmm-EMBI Global-Preview.xlsx"

new\_file\_names.append(new\_file\_name)

source\_path = os.path.join(download\_folder, file\_name)

excel\_file = os.path.splitext(source\_path)[0] + ".xlsx"

df = pd.read\_csv(source\_path, delimiter=',')

df.to\_excel(excel\_file, index=False)

source\_path = excel\_file

target\_path = os.path.join(embi\_global\_preview, new\_file\_name)

shutil.move(source\_path, target\_path)

elif i == 8:

new\_file\_name = f"{date}-jpmm-EMBI+-Composition.xlsx"

new\_file\_names.append(new\_file\_name)

source\_path = os.path.join(download\_folder, file\_name)

excel\_file = os.path.splitext(source\_path)[0] + ".xlsx"

df = pd.read\_csv(source\_path, delimiter=',')

df.to\_excel(excel\_file, index=False)

source\_path = excel\_file

target\_path = os.path.join(embi\_plus\_compo, new\_file\_name)

shutil.move(source\_path, target\_path)

elif i == 9:

new\_file\_name = f"{date}-jpmm-EMBI+-Sub-Index.xlsx"

new\_file\_names.append(new\_file\_name)

source\_path = os.path.join(download\_folder, file\_name)

excel\_file = os.path.splitext(source\_path)[0] + ".xlsx"

df = pd.read\_csv(source\_path, delimiter=',')

df.to\_excel(excel\_file, index=False)

source\_path = excel\_file

target\_path = os.path.join(embi\_plus\_sub, new\_file\_name)

shutil.move(source\_path, target\_path)

elif i == 10:

new\_file\_name = f"{date}-jpmm-EMBI+-Preview.xlsx"

new\_file\_names.append(new\_file\_name)

source\_path = os.path.join(download\_folder, file\_name)

excel\_file = os.path.splitext(source\_path)[0] + ".xlsx"

df = pd.read\_csv(source\_path, delimiter=',')

df.to\_excel(excel\_file, index=False)

source\_path = excel\_file

target\_path = os.path.join(embi\_plus\_preview, new\_file\_name)

shutil.move(source\_path, target\_path)

elif i == 11:

new\_file\_name = f"{date}-jpmm-EMBIG Core-Composition.xlsx"

new\_file\_names.append(new\_file\_name)

source\_path = os.path.join(download\_folder, file\_name)

excel\_file = os.path.splitext(source\_path)[0] + ".xlsx"

df = pd.read\_csv(source\_path, delimiter=',')

df.to\_excel(excel\_file, index=False)

source\_path = excel\_file

target\_path = os.path.join(embig\_core\_compo, new\_file\_name)

shutil.move(source\_path, target\_path)

elif i == 12:

new\_file\_name = f"{date}-jpmm-EMBIG Core-Preview.xlsx"

new\_file\_names.append(new\_file\_name)

source\_path = os.path.join(download\_folder, file\_name)

excel\_file = os.path.splitext(source\_path)[0] + ".xlsx"

df = pd.read\_csv(source\_path, delimiter=',')

df.to\_excel(excel\_file, index=False)

source\_path = excel\_file

target\_path = os.path.join(embig\_core\_preview, new\_file\_name)

shutil.move(source\_path, target\_path)

else:

break

driver.quit()

download\_folder = os.path.expandvars(r"%userprofile%\Downloads")

market\_jp(r"U:\GDA\PFC\03\_Gerants\03\_12\_NB\JP", download\_folder, '20240830')

MS download :

from selenium import webdriver

from selenium.webdriver.edge.service import Service

from selenium.webdriver.edge.options import Options

import time

import config as cf

from selenium.webdriver.common.by import By

from selenium.webdriver.common.action\_chains import ActionChains

import os

import datetime

import shutil

import datetime

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

import re

def matrix\_ms(path\_for\_saving, path\_default\_download):

options = webdriver.EdgeOptions()

options.add\_argument(f"user-data-dir={cf.local['userDataDir']}")

download\_dir = path\_default\_download

options.add\_experimental\_option("prefs", {

"download.default\_directory": download\_dir,

"download.prompt\_for\_download": False, # Prevents the prompt for download

"download.directory\_upgrade": True,

"safebrowsing.enabled": True

})

service = Service(cf.local["executablePath"])

driver = webdriver.Edge(service=service, options=options)

date\_init = datetime.datetime.now()

#supprimer l'historique de téléchargement

driver.get("edge://downloads/all")

try :

wait = WebDriverWait(driver, 10)

first\_button = wait.until(EC.element\_to\_be\_clickable((By.CSS\_SELECTOR, 'button[title="Effacer tout"]')))

first\_button.click()

second\_button = wait.until(EC.element\_to\_be\_clickable((By.ID, 'confirmModalPrimaryButton')))

second\_button.click()

except Exception as e:

print("historique déjà vide:")

#commencement du téléchargement

driver.get("https://ny.matrix.ms.com/eqr/research/docs/content/equitystrategy/ui/index.html")

#trouver la date

time.sleep(5)

view\_info\_div = driver.find\_element(By.CLASS\_NAME, 'view-info')

full\_text = view\_info\_div.text

match = re.search(r'of\s(.\*?)\s\(', full\_text)

if match:

extracted\_text = match.group(1)

print(extracted\_text)

date\_obj = datetime.datetime.strptime(extracted\_text, '%d %B %Y')

yesterday = date\_obj.strftime('%Y%m%d')

#initialisation des téléchargements

i = 1

while i <8:

print(i)

date\_before = datetime.datetime.now()

dropdownbox = driver.find\_elements(by=By.TAG\_NAME, value="Option")

dropdownbox[i].click()

#vérifie que le fichier est présent dans téléchargement

time.sleep(1)

fname = []

while(len(fname)!=1):

now = datetime.datetime.now()

for filename in os.listdir(path\_default\_download):

if os.path.getmtime(os.path.join(path\_default\_download, filename)) > date\_before.timestamp() and os.path.getmtime(os.path.join(path\_default\_download, filename)) < now.timestamp():

fname.append((filename, os.path.getmtime(os.path.join(path\_default\_download, filename))))

time.sleep(1)

#je vérifie que le fichier à son telechargement fini

name = dropdownbox[i].text

driver.get("edge://downloads/all")

downloads\_list = driver.find\_elements(By.XPATH, "//div[@role='listitem']")

for download\_item in downloads\_list :

while download\_item.text[-7:]!='dossier' :

time.sleep(1)

driver.get("https://ny.matrix.ms.com/eqr/research/docs/content/equitystrategy/ui/index.html")

if(name=='QuantIndia'):

break

i = i+1

time.sleep(5)

#clic et download du bouton MOST (3 month Horizon)

date\_before = datetime.datetime.now()

buttons = driver.find\_elements(By.TAG\_NAME, 'button')

for button in buttons:

if 'downloadXLSIcon' in button.get\_attribute('class') and 'enabled' in button.get\_attribute('class'):

button.click()

fname = []

while(len(fname)!=1):

now = datetime.datetime.now()

for filename in os.listdir(path\_default\_download):

if os.path.getmtime(os.path.join(path\_default\_download, filename)) > date\_before.timestamp() and os.path.getmtime(os.path.join(path\_default\_download, filename)) < now.timestamp():

fname.append((filename, os.path.getmtime(os.path.join(path\_default\_download, filename))))

time.sleep(1)

#je vérifie que le fichier à son telechargement fini

driver.get("edge://downloads/all")

downloads\_list = driver.find\_elements(By.XPATH, "//div[@role='listitem']")

for download\_item in downloads\_list :

while download\_item.text[-7:]!='dossier' :

time.sleep(1)

driver.get("https://ny.matrix.ms.com/eqr/research/docs/content/equitystrategy/ui/index.html")

time.sleep(5)

date\_before = datetime.datetime.now()

buttons = driver.find\_elements(By.TAG\_NAME, 'button')

div\_element = driver.find\_element(By.XPATH, "//div[label[text()='BEST']]")

div\_element.click()

for button in buttons:

if 'downloadXLSIcon' in button.get\_attribute('class') and 'enabled' in button.get\_attribute('class'):

button.click()

fname = []

while(len(fname)!=1):

now = datetime.datetime.now()

for filename in os.listdir(path\_default\_download):

if os.path.getmtime(os.path.join(path\_default\_download, filename)) > date\_before.timestamp() and os.path.getmtime(os.path.join(path\_default\_download, filename)) < now.timestamp():

fname.append((filename, os.path.getmtime(os.path.join(path\_default\_download, filename))))

time.sleep(1)

driver.get("edge://downloads/all")

downloads\_list = driver.find\_elements(By.XPATH, "//div[@role='listitem']")

for download\_item in downloads\_list :

while download\_item.text[-7:]!='dossier' :

time.sleep(1)

file\_names\_with\_timestamps = []

file\_names = []

now = datetime.datetime.now()

num\_files\_downloaded = 0

for filename in os.listdir(download\_folder):

if os.path.getmtime(os.path.join(download\_folder, filename)) > date\_init.timestamp() and os.path.getmtime(os.path.join(download\_folder, filename)) < now.timestamp():

file\_names\_with\_timestamps.append((filename, os.path.getmtime(os.path.join(download\_folder, filename))))

#faire une liste avec le nom des fichiers rangés par date d'anciennete dans file\_name (du plus vieux au plus jeune)

file\_names\_with\_timestamps.sort(key=lambda x: x[1], reverse=True)

file\_names = [x[0] for x in file\_names\_with\_timestamps]

file\_names = file\_names[::-1]

print(f"Nombre de fichiers téléchargés: {len(file\_names)}")

if(len(file\_names)!=9):

print("erreur dans le chargement des fichiers, ces fichiers ont été downloads correctement :")

print(file\_names)

raise Exception("Impossible de télécharger tous les fichiers nécessaires.")

for file\_name in file\_names:

print(file\_name)

#renommer les fichiers correctement

new\_file\_names = []

for i, file\_name in enumerate(file\_names):

if i == 0:

new\_file\_name = f"{yesterday}-Weekly Global Model Rankings-Global-DM.xlsx"

new\_file\_names.append(new\_file\_name)

elif i == 1:

new\_file\_name = f"{yesterday}-Weekly Global Model Rankings-Emerging.xlsx"

new\_file\_names.append(new\_file\_name)

elif i == 2:

new\_file\_name = f"{yesterday}-Weekly Global Model Rankings-China A.xlsx"

new\_file\_names.append(new\_file\_name)

elif i == 3:

new\_file\_name = f"{yesterday}-QuanTopix SAS - HC.xlsx"

new\_file\_names.append(new\_file\_name)

elif i == 4:

new\_file\_name = f"{yesterday}-QuantETF\_model\_output.xlsx"

new\_file\_names.append(new\_file\_name)

elif i == 5:

new\_file\_name = f"{yesterday}-QuantChina\_model.xlsx"

new\_file\_names.append(new\_file\_name)

elif i == 6:

new\_file\_name = f"{yesterday}-QuantIndia\_model.xlsx"

new\_file\_names.append(new\_file\_name)

elif i == 7:

new\_file\_name = f"{yesterday}-EquityStrategyCompanies-Most3.xlsx"

new\_file\_names.append(new\_file\_name)

elif i == 8:

new\_file\_name = f"{yesterday}-EquityStrategyCompanies-Best24.xlsx"

new\_file\_names.append(new\_file\_name)

else:

break

old\_path = os.path.join(download\_folder, file\_name)

new\_path = os.path.join(download\_folder, new\_file\_name)

shutil.move(old\_path, new\_path)

print(f"Fichier {file\_name} renommé en {new\_file\_name}")

#créer le dossier où l'on va déplacer nos fichiers

target\_directory = os.path.join(path\_for\_saving, f"{yesterday} - MS matrix Data")

if not os.path.exists(target\_directory):

os.makedirs(target\_directory)

#déplacer les fichiers renommés dans le nouveau dossier

for new\_file\_name in new\_file\_names:

source\_path = os.path.join(download\_folder, new\_file\_name)

target\_path = os.path.join(target\_directory, new\_file\_name)

shutil.move(source\_path, target\_path)

print(f"Fichier {new\_file\_name} a été déplacé")

driver.quit()

download\_folder = os.path.expandvars(r"%userprofile%\Downloads")

#download\_folder = r"U:\GDA\PFC\03\_Gerants\03\_12\_NB\ok"

matrix\_ms(r"U:\GDA\PFC\03\_Gerants\03\_12\_NB\MS", download\_folder)

**Config :**

local = {

"executablePath" : r"C:\Users\nbonneau\Downloads\edgedriver\_win64\msedgedriver.exe",

"userDataDir" : r"C:\Users\nbonneau\Downloads\edgedriver\_win64\profile",

}

Nordic

from tia.bbg import LocalTerminal

import pandas as pd

import datetime

from xbbg import blp

from blp import blp as bld

import pybbg as pybbg

import matplotlib.pyplot as plt

import os

from dateutil.relativedelta import relativedelta

import numpy as np

from datetime import datetime, date

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix

import plotly.graph\_objects as go

import pypfopt

from collections import OrderedDict

from pypfopt.efficient\_frontier import EfficientFrontier

from pandas import Timestamp

import json

from pypfopt import plotting

import cvxpy as cp

from pypfopt.objective\_functions import ex\_ante\_tracking\_error

def end\_month\_data(df, force\_last\_date=False): #vérifie si les mois sont bien sont succesifs

df = df.reset\_index("date")

last\_date = df["date"].iloc[-1]

print(last\_date)

if force\_last\_date:

df["date"] = df["date"].apply(lambda x: x+pd.offsets.MonthEnd(1) if (x.month == (x+pd.offsets.MonthEnd(1)).month) else x)

else:

df["date"] = df["date"].apply(lambda x: x+pd.offsets.MonthEnd(1) if (x.month == (x+pd.offsets.MonthEnd(1)).month and x!=last\_date) else x)

df = df.set\_index("date")

return df

def expected\_returns(start\_date, end\_date):

bbg = pybbg.Pybbg()

bquery = bld.BlpQuery().start()

#retrouver le taux attendu en allemagne

crp\_msci\_europe = blp.bdp("VOW3 GY Equity", "COUNTRY\_RISK\_MARKET\_RETURN", COUNTRY\_RISK\_EFFECTIVE\_DATE=end\_date).iloc[0,0]/100

#retrouver le taux attendu sur les mids cap

beta\_mid\_europe = blp.bdp("MCXR Index", "BETA\_ADJ\_OVERRIDABLE", BETA\_OVERRIDE\_REL\_INDEX="MXEUSXNE Index",BETA\_OVERRIDE\_END\_DT = end\_date,BETA\_OVERRIDE\_START\_DT = start\_date).iloc[0,0]

crp\_mid\_europe = beta\_mid\_europe\*crp\_msci\_europe

#retrouver le taux attendu sur les small cap

beta\_small\_europe = blp.bdp("SCXR Index", "BETA\_ADJ\_OVERRIDABLE", BETA\_OVERRIDE\_REL\_INDEX="MXEUSXNE Index",BETA\_OVERRIDE\_END\_DT = end\_date,BETA\_OVERRIDE\_START\_DT = start\_date).iloc[0,0]

crp\_small\_europe = beta\_small\_europe\*crp\_msci\_europe

#retrouver le taux attendus sur letf msci nordic tr

#sweden crp

crp\_sweden = blp.bdp("VOLVB SS Equity", "COUNTRY\_RISK\_MARKET\_RETURN", COUNTRY\_RISK\_EFFECTIVE\_DATE=end\_date).iloc[0,0]/100

#denmark crp

crp\_denmark = blp.bdp("NOVOB DC Equity", "COUNTRY\_RISK\_MARKET\_RETURN", COUNTRY\_RISK\_EFFECTIVE\_DATE=end\_date).iloc[0,0]/100

#norway crp

crp\_norway = blp.bdp("DNB NO Equity", "COUNTRY\_RISK\_MARKET\_RETURN", COUNTRY\_RISK\_EFFECTIVE\_DATE=end\_date).iloc[0,0]/100

#finland crp

crp\_finland = blp.bdp("NDA SS Equity", "COUNTRY\_RISK\_MARKET\_RETURN", COUNTRY\_RISK\_EFFECTIVE\_DATE=end\_date).iloc[0,0]/100

#recup les poids de chaque pays dans le MSCI Nordic

datefin\_format\_tirets = f"{end\_date[:4]}-{end\_date[4:6]}-{end\_date[6:]}"

poids = bquery.bql(f"for(holdings('XDN0 GY Equity',dates='{datefin\_format\_tirets}')) get(sum(group(id().weights,COUNTRY\_FULL\_NAME)))")

denmark\_weight = poids.loc[poids['secondary\_value']=='DENMARK']['value'].iloc[0]/100

sweden\_weight = poids.loc[poids['secondary\_value']=='SWEDEN']['value'].iloc[0]/100

finland\_weight = poids.loc[poids['secondary\_value']=='FINLAND']['value'].iloc[0]/100

norway\_weight = poids.loc[poids['secondary\_value']=='NORWAY']['value'].iloc[0]/100

crp\_msci\_nordic = denmark\_weight\*crp\_denmark + sweden\_weight\*crp\_sweden +crp\_norway\* norway\_weight+finland\_weight\*crp\_finland

return crp\_msci\_europe, crp\_mid\_europe, crp\_msci\_nordic, crp\_small\_europe

def h5store(filename, df, dic):

store = pd.HDFStore(filename)

store.put('mydata', df)

store.get\_storer('mydata').attrs.metadata = dic

store.close()

def h5load(filename):

with pd.HDFStore(filename) as store:

data = store['mydata']

metadata = store.get\_storer('mydata').attrs.metadata

data.attrs = metadata

return data, metadata

def \_build\_easy\_map(assets):

assets\_easy\_map = {}

for k in assets.keys(): #récupère les tickers des assets

assets\_easy\_map[assets[k]['ticker']] = assets[k]['des']

return assets\_easy\_map

def download\_data(start\_date, end\_date):

assets = OrderedDict()

assets['MXEUSXNE'] = {}

assets['MXEUSXNE']['ticker'] = "MXEUSXNE Index"

assets['MXEUSXNE']['des'] = "MSCI Europe"

assets['MCXR'] = {}

assets['MCXR']['ticker'] = "MCXR Index"

assets['MCXR']['des'] = "MCXR Index"

assets['XDN0 GY'] = {}

assets['XDN0 GY']['ticker'] = "MSDENCN Index"

assets['XDN0 GY']['des'] = "MSCI Nordic ETF"

assets['SCXR'] = {}

assets['SCXR']['ticker'] = "SCXR Index"

assets['SCXR']['des'] = "SCXR Index"

fields = ['PX\_LAST']

#Telechargement des prix et des returns en daily

period = 'DAILY'

resp = LocalTerminal.get\_historical([assets[i]["ticker"] for i in assets.keys()],

fields, start=start\_date, end=end\_date, period=period)

daily\_price = resp.as\_frame()

daily\_price.columns = daily\_price.columns.droplevel(1)

daily\_price.ffill(inplace=True)

daily\_price

daily\_price.rename(columns=\_build\_easy\_map(assets), inplace=True)

daily\_return = daily\_price.pct\_change()

daily\_return = daily\_return[start\_date:]

index\_daily = daily\_return.index

#prochaine date de fin de mois

today = datetime.date.today()

prec\_month = datetime.date(today.year, today.month, 1) - datetime.timedelta(days=1)

prec\_month = prec\_month.strftime("%Y-%m-%d")

next\_month = datetime.date(today.year, today.month, 1) + relativedelta(months=+1) - datetime.timedelta(days=1)

next\_month = next\_month.strftime("%Y-%m-%d")

#Telechargement des prix et des returns en monthly

period1 = 'MONTHLY'

resp = LocalTerminal.get\_historical([assets[i]["ticker"] for i in assets.keys()],

fields, start=start\_date, end=end\_date, period=period1)

monthly\_price = resp.as\_frame()

monthly\_price.columns = monthly\_price.columns.droplevel(1)

monthly\_price.ffill(inplace=True)

monthly\_price

monthly\_price.rename(columns=\_build\_easy\_map(assets), inplace=True)

monthly\_return = monthly\_price.pct\_change()

monthly\_return = monthly\_return[start\_date:]

index\_monthly = monthly\_return.index

expected\_returns\_df = pd.DataFrame(columns = ['date','MSCI Europe', 'MCXR Index', 'MSCI Nordic ETF','SCXR Index'])

#on enregistre la covariance du mois dans le dictionnaire covariance\_dict avec des données daily, la clé correspond au mois où la covariance est observée

for date in monthly\_return.index:

crp\_msci\_europe, crp\_mid\_europe, crp\_msci\_nordic, crp\_small\_europe = expected\_returns(start\_date, date.strftime('%Y%m%d'))

expected\_returns\_df = expected\_returns\_df.\_append({'date': date, 'MSCI Europe' :crp\_msci\_europe , 'MCXR Index':crp\_mid\_europe , 'MSCI Nordic ETF':crp\_msci\_nordic,'SCXR Index':crp\_small\_europe}, ignore\_index= True)

#forcer la dernière valeur sur la fin du prochain mois

expected\_returns\_next\_month = expected\_returns(start\_date, today.strftime('%Y%m%d'))

last\_index = expected\_returns\_df.index[-1]+1

expected\_returns\_df.loc[last\_index,'date'] = next\_month

expected\_returns\_df.loc[last\_index,'MSCI Europe'] = expected\_returns\_next\_month[0]

expected\_returns\_df.loc[last\_index,'MCXR Index'] = expected\_returns\_next\_month[1]

expected\_returns\_df.loc[last\_index,'MSCI Nordic ETF'] = expected\_returns\_next\_month[2]

expected\_returns\_df.loc[last\_index,'SCXR Index'] = expected\_returns\_next\_month[3]

monthly\_return.loc[next\_month,'MSCI Europe'] = (daily\_price.iloc[-1]['MSCI Europe']-daily\_price.loc[prec\_month,'MSCI Europe'])/daily\_price.loc[prec\_month,'MSCI Europe']

monthly\_return.loc[next\_month,'MCXR Index'] = (daily\_price.iloc[-1]['MCXR Index']-daily\_price.loc[prec\_month,'MCXR Index'])/daily\_price.loc[prec\_month,'MCXR Index']

monthly\_return.loc[next\_month,'SCXR Index'] = (daily\_price.iloc[-1]['SCXR Index']-daily\_price.loc[prec\_month,'SCXR Index'])/daily\_price.loc[prec\_month,'SCXR Index']

monthly\_return.loc[next\_month,'MSCI Nordic ETF'] = (daily\_price.iloc[-1]['MSCI Nordic ETF']-daily\_price.loc[prec\_month,'MSCI Nordic ETF'])/daily\_price.loc[prec\_month,'MSCI Nordic ETF']

expected\_returns\_df = expected\_returns\_df.set\_index('date')

expected\_returns\_df= expected\_returns\_df.ffill()

print(expected\_returns\_df)

dossier\_initial = r'U:\GDA\PFC\03\_Gerants\03\_12\_NB\nordic'

monthly\_return = monthly\_return.reset\_index(drop=False)

file\_name\_h5 = f"df\_monthly\_return.h5"

df\_member\_path\_h5 = os.path.join(dossier\_initial, file\_name\_h5)

metadata = dict(data="data monthly return",provider="Bloomberg",indice="MSCIEU, MXCR, SXCR, MSCI Nordic")

h5store(df\_member\_path\_h5 ,monthly\_return,metadata)

file\_name\_xlsx = f"df\_monthly\_return.xlsx"

df\_member\_path = os.path.join(dossier\_initial, file\_name\_xlsx)

monthly\_return.to\_excel(df\_member\_path, index=False)

expected\_returns\_df = expected\_returns\_df.reset\_index(drop=False)

file\_name\_h5 = f"df\_expected\_returns.h5"

df\_member\_path\_h5 = os.path.join(dossier\_initial, file\_name\_h5)

metadata = dict(data="expected returns",provider="Bloomberg",indice="MSCIEU, MXCR, SXCR, MSCI Nordic")

h5store(df\_member\_path\_h5 ,expected\_returns\_df,metadata)

file\_name\_xlsx = f"df\_expected\_returns.xlsx"

df\_member\_path = os.path.join(dossier\_initial, file\_name\_xlsx)

expected\_returns\_df.to\_excel(df\_member\_path, index=False)

daily\_return = daily\_return.reset\_index(drop=False)

file\_name\_h5 = f"df\_daily\_return.h5"

df\_member\_path\_h5 = os.path.join(dossier\_initial, file\_name\_h5)

metadata = dict(data="daily return",provider="Bloomberg",indice="MSCIEU, MXCR, SXCR, MSCI Nordic")

h5store(df\_member\_path\_h5 ,daily\_return,metadata)

file\_name\_xlsx = f"df\_daily\_return.xlsx"

df\_member\_path = os.path.join(dossier\_initial, file\_name\_xlsx)

daily\_return.to\_excel(df\_member\_path, index=False)

#download\_data("20150902","20240630") #date final correspond à la date du jour

def compute\_optimal\_ptf(start\_date, end\_date, an, rolling = False):

dossier\_initial = r'U:\GDA\PFC\03\_Gerants\03\_12\_NB\nordic'

file\_name\_h5 = f"df\_monthly\_return.h5"

monthly\_return, metadata1 = h5load(os.path.join(dossier\_initial, file\_name\_h5))

monthly\_return = monthly\_return.set\_index('date')

file\_name\_h5 = f"df\_expected\_returns.h5"

expected\_returns\_df, metadata2 = h5load(os.path.join(dossier\_initial, file\_name\_h5))

expected\_returns\_df = expected\_returns\_df.set\_index('date')

file\_name\_h5 = f"df\_daily\_return.h5"

daily\_return, metadata3 = h5load(os.path.join(dossier\_initial, file\_name\_h5))

daily\_return = daily\_return.set\_index('date')

monthly\_return = monthly\_return[start\_date:end\_date]

daily\_return = daily\_return[start\_date:monthly\_return.index[-1]]

index\_monthly = monthly\_return.index

expected\_returns\_df = expected\_returns\_df[start\_date:end\_date]

#mettre les expected returns au meme niveau

#expected\_returns\_df['MCXR Index'] = expected\_returns\_df['MSCI Europe']

#expected\_returns\_df['SCXR Index'] = expected\_returns\_df['MSCI Europe']

#expected\_returns\_df['MSCI Nordic ETF'] = expected\_returns\_df['MSCI Europe']

index\_daily = daily\_return.index

covariance\_dict = {}

cor\_matrix = {}

for date in monthly\_return.index:

copy\_daily\_return = daily\_return.loc[(index\_daily.month == date.month) & (index\_daily.year == date.year)]

copy\_daily\_return = copy\_daily\_return.dropna()

cov\_1\_month = copy\_daily\_return.cov(min\_periods=20)

covariance\_dict[date] = cov\_1\_month\*252

cor\_matrix[date] = copy\_daily\_return.corr(min\_periods = 20)

print(covariance\_dict)

#download taux sans risque europe

indexes = ['EONIA Index', 'ESTRON Index']

resp = LocalTerminal.get\_historical(indexes, 'PX\_LAST',start=start\_date, end=end\_date, period='MONTHLY')

data = resp.as\_frame()

data.columns = data.columns.droplevel(1)

str\_series = pd.Series(index=pd.to\_datetime(data.index), name='ST index', dtype='float64')

ind = np.isnan(data[indexes[1]])

str\_series.loc[~ind] = data.loc[~ind, indexes[1]]

str\_series.loc[ind] = data.loc[ind, indexes[0]]

print(str\_series)

monthly\_return = monthly\_return.iloc[1:]

weights = []

exp\_ptf\_perf = []

date\_tab = []

datedeb = monthly\_return.index[an+1]

datedeb\_perf = monthly\_return.index[an]

for i, date in enumerate(monthly\_return.index[:-1]):

if i>0:

if(date.strftime("%Y-%m-%d")<monthly\_return.index[an].strftime("%Y-%m-%d") ):

continue

#monthly\_returns\_know contient toutes les returns monthly avant la date d'aujourd'hui donc le mois en cours et les mois d'avants

monthly\_returns\_know= monthly\_return.loc[:date]

if rolling :

if(len(monthly\_returns\_know) >an): #conservation des an dernières années pr l'estimation de la cov matrix

monthly\_returns\_know = monthly\_returns\_know[-an:]

cov\_matrix = monthly\_returns\_know.cov()\*12

else :

cov\_matrix = covariance\_dict[Timestamp(monthly\_return.index[i])]

#la cov est calculée uniquement avec les données daily du mois en cours

#cov\_matrix = covariance\_dict[Timestamp(date)]

#constraint tracking error

benchmark\_weights = np.array([0.75,0.125,0,0.125])

#dict\_init = {'initvals': benchmark\_weights}

#dict\_init = {"feastol": 0.0001}

dict\_init = {}

ef = EfficientFrontier(expected\_returns\_df.loc[date], cov\_matrix, verbose=True, solver\_options = dict\_init)

ex = lambda w: ex\_ante\_tracking\_error(w, ef.cov\_matrix, benchmark\_weights) <= 0.015

#ef.add\_constraint(ex)

#ef.add\_constraint(ex\_ante\_tracking\_error, cov\_matrix=ef.cov\_matrix, benchmark\_weights=benchmark\_weights)

#ligne lorsque l'on ne connais pas les vrais expected YTM du prochain mois

#ef = pypfopt.efficient\_frontier.EfficientFrontier(expected\_returns\_aligned.loc[date], cov\_matrix, weight\_bounds=(0, 1), solver=None, verbose=False, solver\_options=None)

new\_keys = ['MSCI Europe', 'MCXR Index', 'MSCI Nordic ETF', 'SCXR Index']

try :

riskfree = str\_series.loc[date]

poid = ef.max\_sharpe(risk\_free\_rate=riskfree/100)

poids = OrderedDict(zip(new\_keys, poid.values()))

weights.append({'date': date, 'weights': poids})

print(date)

exp\_ptf\_perf.append({'date':date, 'perf':ef.portfolio\_performance(risk\_free\_rate=riskfree/100)})

except Exception as e :

print("error for date", date)

#poids = 0

poids = weights[-1]['weights'] # Get the previous weights

weights.append({'date': date, 'weights': poids})

#perf attendue sur le portefeuille pareil que celle d'avant car les poids ne bouge pas car l'optimiseur n'a pas réussi à résourdre le pb d'opti

ptf\_perf = exp\_ptf\_perf[-1]['perf']

exp\_ptf\_perf.append({'date':date, 'perf':ptf\_perf})

data\_ptf = pd.DataFrame()

for element in exp\_ptf\_perf:

date = element['date'] # extraction de la date

ptf\_perf = element['perf'][0]

ptf\_sd = element['perf'][1]

ratio\_sortino = element['perf'][2] # extraction des perf

data\_ptf.loc[date,'perf'] = ptf\_perf

data\_ptf.loc[date,'ptf\_sd'] = ptf\_sd

data\_ptf.loc[date,'ratio\_sortino'] = ratio\_sortino

data\_ptf.index = monthly\_return[datedeb\_perf:monthly\_return.index[-2]].index

print(data\_ptf)

data = {}

for element in weights:

date = element['date'] # extraction de la date

poids = element['weights'] # extraction des poids

data[date] = poids

df = pd.DataFrame(data)

df = df.transpose()

df\_weights = df.copy()

df\_weights.index = monthly\_return[datedeb:].index

weighted\_returns = monthly\_return[datedeb:]\*df\_weights

strategy\_returns = weighted\_returns.sum(axis=1) #somme les poids \* les returns

strategy\_returns\_df = pd.DataFrame(data={'strategy\_returns': strategy\_returns})

print(strategy\_returns\_df)

cumulative\_performance = (1 + strategy\_returns\_df['strategy\_returns']).cumprod()

print(cumulative\_performance)

years = (strategy\_returns\_df.index[-1] - strategy\_returns\_df.index[0]).days / 365

annualized\_return = (cumulative\_performance[-1])\*\*(1/years) - 1

print(annualized\_return)

print(df\_weights)

#cas LONG ONLY MSCI Europe

copy = df

copy['MSCI Europe'] =1

copy['MCXR Index'] = 0

copy['MSCI Nordic ETF'] = 0

copy['SCXR Index'] = 0

print(copy)

copy.index = monthly\_return[datedeb:].index

weighted\_returns\_mscieu = monthly\_return[datedeb:]\*copy

strategy\_returns\_mscieu = weighted\_returns\_mscieu.sum(axis=1)

strategy\_returns\_df\_mscieu = pd.DataFrame(data={'strategy\_returns': strategy\_returns\_mscieu})

print(strategy\_returns\_df\_mscieu)

cumulative\_performance\_mscieu = (1 + strategy\_returns\_df\_mscieu['strategy\_returns']).cumprod()

#cas LONG ONLY MCXR Index

copy\_mid = df

copy\_mid['MSCI Europe'] =0

copy\_mid['MCXR Index'] = 1

copy\_mid['MSCI Nordic ETF'] = 0

copy\_mid['SCXR Index'] = 0

print(copy\_mid)

copy\_mid.index = monthly\_return[datedeb:].index

weighted\_returns\_mid = monthly\_return[datedeb:]\*copy\_mid

strategy\_returns\_mid = weighted\_returns\_mid.sum(axis=1)

strategy\_returns\_df\_mid = pd.DataFrame(data={'strategy\_returns': strategy\_returns\_mid})

print(strategy\_returns\_df\_mid)

cumulative\_performance\_mid = (1 + strategy\_returns\_df\_mid['strategy\_returns']).cumprod()

#cas LONG ONLY MSCI Nordic ETF

copy\_nordic= df

copy\_nordic['MSCI Europe'] =0

copy\_nordic['MCXR Index'] = 0

copy\_nordic['MSCI Nordic ETF'] = 1

copy\_nordic['SCXR Index'] = 0

print(copy\_nordic)

copy\_nordic.index = monthly\_return[datedeb:].index

weighted\_returns\_nordic = monthly\_return[datedeb:]\*copy\_nordic

strategy\_returns\_nordic = weighted\_returns\_nordic.sum(axis=1)

strategy\_returns\_df\_nordic = pd.DataFrame(data={'strategy\_returns': strategy\_returns\_nordic})

print(strategy\_returns\_df\_nordic)

cumulative\_performance\_nordic = (1 + strategy\_returns\_df\_nordic['strategy\_returns']).cumprod()

#cas LONG ONLY small

copy\_small= df

copy\_small['MSCI Europe'] =0

copy\_small['MCXR Index'] = 0

copy\_small['MSCI Nordic ETF'] = 0

copy\_small['SCXR Index'] = 1

print(copy\_small)

copy\_small.index = monthly\_return[datedeb:].index

weighted\_returns\_small = monthly\_return[datedeb:]\*copy\_small

strategy\_returns\_small = weighted\_returns\_small.sum(axis=1)

strategy\_returns\_df\_small = pd.DataFrame(data={'strategy\_returns': strategy\_returns\_small})

print(strategy\_returns\_df\_small)

cumulative\_performance\_small = (1 + strategy\_returns\_df\_small['strategy\_returns']).cumprod()

#cas LONG portfeuille europe cdc

copy\_ptf= df

copy\_ptf['MSCI Europe'] =0.75

copy\_ptf['MCXR Index'] = 0.125

copy\_ptf['MSCI Nordic ETF'] = 0

copy\_ptf['SCXR Index'] = 0.125

print(copy\_ptf)

copy\_ptf.index = monthly\_return[datedeb:].index

weighted\_returns\_ptf = monthly\_return[datedeb:]\*copy\_ptf

strategy\_returns\_ptf = weighted\_returns\_small.sum(axis=1)

strategy\_returns\_df\_ptf = pd.DataFrame(data={'strategy\_returns': strategy\_returns\_ptf})

print(strategy\_returns\_df\_ptf)

cumulative\_performance\_ptf = (1 + strategy\_returns\_df\_ptf['strategy\_returns']).cumprod()

#création du graphe de performance cumulative

plt.figure(figsize=(10, 6))

#plt.figure(figsize=(10, 6))

plt.plot(cumulative\_performance.index, cumulative\_performance.values, label='Performance cumulative', color='brown')

plt.plot(cumulative\_performance\_small.index, cumulative\_performance\_small.values, label='Performance cumulative small', color='green')

plt.plot(cumulative\_performance\_nordic.index, cumulative\_performance\_nordic.values, label='Performance cumulative nordic', color='lightskyblue')

plt.plot(cumulative\_performance\_mid.index, cumulative\_performance\_mid.values, label='Performance cumulative mid', color='red')

plt.plot(cumulative\_performance\_mscieu.index, cumulative\_performance\_mscieu.values, label='Performance cumulative MSCI EUROPE', color='black')

plt.plot(cumulative\_performance\_ptf.index, cumulative\_performance\_ptf.values, label='Performance cumulative Portefeuille action EUROPE', color='violet')

plt.title('Performance cumulative')

plt.xlabel('Date')

plt.ylabel('Valeur cumulative')

plt.legend()

plt.grid(True)

destination\_file = r'U:\GDA\PFC\03\_Gerants\03\_12\_NB\nordic'

name = f"performance\_cumulative\_{start\_date}\_{end\_date}.png"

plt.savefig(os.path.join(destination\_file, name))

plt.show()

fig, ax = plt.subplots(figsize=(10, 6))

colors = ['black','red','lightskyblue','green']

i = 0

for col in df\_weights.columns:

ax.plot(df\_weights.index, df\_weights[col], label=col,color=colors[i] )

i +=1

ax.legend()

ax.set\_title('Évolution des poids au fil du temps with ')

ax.set\_xlabel('Date')

ax.set\_ylabel('Poids')

#plt.savefig(os.path.join(destination\_folder, "weight\_matrix.png"))

#print("La figure a été enregistrée sous :", destination\_file)

name = f"weight\_matrix\_{start\_date}\_{end\_date}.png"

plt.savefig(os.path.join(destination\_file, name))

plt.show()

#plot les expected returns et les expected standard deviation du ptf

dates = data\_ptf.index.tolist()

perf = data\_ptf['perf'].tolist()

ptf\_sd = data\_ptf['ptf\_sd'].tolist()

plt.plot(dates, perf, label='Expected Portfolio Return')

plt.plot(dates, ptf\_sd, label='Expected Portfolio Semivariance')

plt.xlabel('Date')

plt.ylabel('Valeur')

plt.title('Expected Portfolio Performance et Semivariance en fonction du temps')

plt.legend()

name = f"expected\_ptf\_perf\_{start\_date}\_{end\_date}.png"

plt.savefig(os.path.join(destination\_file, name))

plt.show()

#ratio de sortino

dates = data\_ptf.index.tolist()

sortino\_ratio = data\_ptf['ratio\_sortino'].tolist()

plt.plot(dates, sortino\_ratio, label='Expected Portfolio Ratio de Sortino')

plt.xlabel('Date')

plt.ylabel('Valeur')

plt.title('Expected Portfolio Ratio de Sortino en fonction du temps')

plt.legend()

name = f"sortino\_ratio\_{start\_date}\_{end\_date}.png"

plt.savefig(os.path.join(destination\_file, name))

plt.show()

#pour tracer la frontière efficiente

S = cov\_matrix

mu = expected\_returns\_df.loc[end\_date]

n\_samples = 1000

w = np.random.dirichlet(np.ones(len(mu)), n\_samples)

rets = w.dot(mu)

stds = np.sqrt((w.T \* (S @ w.T)).sum(axis=0))

sharpes = rets / stds

ef = pypfopt.efficient\_frontier.EfficientFrontier(mu, S)

fig, ax = plt.subplots()

plotting.plot\_efficient\_frontier(ef, ax=ax, show\_assets=False)

''# Find and plot the tangency portfolio

ef2 = pypfopt.efficient\_frontier.EfficientFrontier(mu, S)

ef2.max\_sharpe()

ret\_tangent, std\_tangent, \_ = ef2.portfolio\_performance()

# Plot random portfolios

ax.scatter(stds, rets, marker=".", c=sharpes, cmap="gray")

print(mu)

print(S.iloc[0,0])

ax.set\_title("Efficient Frontier with random portfolios with ")

#max\_sharpe =monthly\_returns[end\_date]\*df.loc[end\_date]

#ordonnee = max\_sharp.sum(axis=1)

ax.scatter(std\_tangent, ret\_tangent, marker="o", color='r', label='Portfolio Optimal')

ax.legend()

plt.tight\_layout()

plt.savefig(os.path.join(destination\_file, "Efficient Frontier.png"))

plt.close(fig)

print("Les derniers poids optimaux proposés sont :")

print(df\_weights.iloc[-1])

compute\_optimal\_ptf("20151030","20240630",60,False)

Skew :

from tia.bbg import LocalTerminal

import pandas as pd

import datetime

from xbbg import blp

import pybbg as pybbg

import matplotlib.pyplot as plt

import os

from dateutil.relativedelta import relativedelta

import numpy as np

def h5store(filename, df, dic):

store = pd.HDFStore(filename)

store.put('mydata', df)

store.get\_storer('mydata').attrs.metadata = dic

store.close()

def h5load(filename):

with pd.HDFStore(filename) as store:

data = store['mydata']

metadata = store.get\_storer('mydata').attrs.metadata

data.attrs = metadata

return data, metadata

def download\_vol\_data(date\_hist):

#charger l'ensemble des tickers existants

bbg = pybbg.Pybbg()

#tickers\_dispo = blp.bds('SPX Index','OPT\_CHAIN')

date\_str = date\_hist.strftime('%Y%m%d')

tickers\_dispo = blp.bds('SPX Index','OPT\_CHAIN',SINGLE\_DATE\_OVERRIDE=date\_str)

tickers\_dispo = tickers\_dispo['security\_description']

print(tickers\_dispo)

#récupérer le vrai nom du ticker

tickers\_names = blp.bdp(tickers\_dispo, "SECURITY\_DES")

tickers\_names = tickers\_names['security\_des'].tolist()

print(tickers\_names)

dates\_availables = []

for ticker in tickers\_names:

parts = ticker.split()

for part in parts:

if "/" in part and len(part) == 8:

dates\_availables.append(part)

#for i in range(0,len(tickers\_names)) :

# ticker = tickers\_names[i]

# dates\_availables.append(ticker[-14:-6])

dates\_availables\_unique = list(set(dates\_availables)) #recupérer que les éléments uniques

print(dates\_availables\_unique)

#cherche le ticker le plus proche dans deux mois

#today = datetime.date.today()

today = date\_hist

next\_date = today + datetime.timedelta(days=60)

next\_date\_format = next\_date.strftime('%m/%d/%y')

tomorrow = today + datetime.timedelta(days=1)

tomorrow\_format\_compact = tomorrow.strftime('%Y%m%d')

yesterday = today - datetime.timedelta(days=1)

yesterday\_format\_compact = yesterday.strftime('%Y%m%d')

trouve = False

sens\_chrono = True

jour = 1

while(trouve==False):

if next\_date\_format in dates\_availables\_unique:

trouve = True

print("l'option la plus proche de la maturité dans 2 mois est à la date ", next\_date\_format)

else :

if(sens\_chrono):

next\_date = next\_date + datetime.timedelta(days=jour)

sens\_chrono = False

else :

next\_date = next\_date - datetime.timedelta(days=jour)

sens\_chrono = True

jour +=1

next\_date\_format = next\_date.strftime('%m/%d/%y')

#recup les noms des tickers qui correspond a next\_date\_format

tickers\_date = [tickers\_names[i] for i, date in enumerate(dates\_availables) if date == next\_date\_format]

print(tickers\_date)

today\_format\_compact = today.strftime('%Y%m%d')

spot1 = LocalTerminal.get\_historical('SPX Index', 'PX\_LAST',f'{today\_format\_compact}',f'{today\_format\_compact}').as\_frame()

spot = spot1.iloc[0,0]

print("Le prix du SPX pour le jour ", today\_format\_compact, " est égal : ", spot)

#cas du call

df\_call = pd.DataFrame()

df\_call['ticker'] = [tickers\_date[i] for i, date in enumerate(tickers\_date) if date[-5] == 'C']

df\_call['strike'] = df\_call['ticker'].apply(lambda x: x[-4:])

df\_call = df\_call.sort\_values('strike', ascending = True)

#cas du put

df\_put = pd.DataFrame()

df\_put['ticker'] = [tickers\_date[i] for i, date in enumerate(tickers\_date) if date[-5] == 'P']

df\_put['strike'] = df\_put['ticker'].apply(lambda x: x[-4:])

df\_put = df\_put.sort\_values('strike', ascending = True)

#calcul des bornes inf et sup en fonction du spot et ne garder que ca dans le dataframe

inf = int(round(spot\*0.8,-1))

sup = int(round(spot\*1.2,-1)) #remettre 1.2

#cas du call

df\_call['strike'] = df\_call['strike'].astype(int)

df\_call = df\_call[(df\_call['strike'] >= inf) & (df\_call['strike'] <= sup)]

#cas du put

df\_put['strike'] = df\_put['strike'].astype(int)

df\_put = df\_put[(df\_put['strike'] >= inf) & (df\_put['strike'] <= sup)]

#mettre le nombre de contrats ouverts

df\_call['ticker'] = df\_call['ticker'] + ' Index'

df\_put['ticker'] = df\_put['ticker'] + ' Index'

#cas du call

resp1 = pd.DataFrame()

tickers1 = df\_call['ticker'].tolist()

#resp1 = LocalTerminal.get\_reference\_data(tickers1, 'OPEN\_INT', today\_format\_compact, 'days=a').as\_frame()

#resp1 = resp1.rename(columns={'OPEN\_INT': 'agreement'})

#resp1['ticker'] = resp1.index

#df\_call = pd.merge(df\_call, resp1, on='ticker')

#source de warnings

resp1 = pd.DataFrame(index=tickers1, columns=['OPEN\_INT'])

for ticker in tickers1:

try:

df = bbg.bdh([ticker], 'OPEN\_INT', today\_format\_compact, tomorrow\_format\_compact)

resp1.loc[ticker, 'OPEN\_INT'] = df.iloc[0,0]

except:

try:

df = bbg.bdh([ticker], 'OPEN\_INT', yesterday\_format\_compact, today\_format\_compact)

resp1.loc[ticker, 'OPEN\_INT'] = df.iloc[0,0]

except:

resp1.loc[ticker, 'OPEN\_INT'] = 0

resp1 = resp1.rename(columns={'OPEN\_INT': 'agreement'})

resp1['ticker'] = resp1.index

df\_call = pd.merge(df\_call, resp1, on='ticker')

#cas du put

resp2 = pd.DataFrame()

tickers2 = df\_put['ticker'].tolist()

#resp2 = LocalTerminal.get\_reference\_data(tickers2, 'OPEN\_INT', today\_format\_compact, 'days=a').as\_frame()

#resp2 = resp2.rename(columns = {'OPEN\_INT': 'agreement'})

#resp2['ticker'] = resp2.index

#df\_put = pd.merge(df\_put, resp2, on='ticker')

resp2 = pd.DataFrame(index=tickers2, columns=['OPEN\_INT'])

for ticker in tickers2:

try:

df2 = bbg.bdh([ticker], 'OPEN\_INT', today\_format\_compact, tomorrow\_format\_compact)

resp2.loc[ticker, 'OPEN\_INT'] = df2.iloc[0,0]

except:

try:

df2 = bbg.bdh([ticker], 'OPEN\_INT', yesterday\_format\_compact, today\_format\_compact)

resp2.loc[ticker, 'OPEN\_INT'] = df2.iloc[0,0]

except:

resp2.loc[ticker, 'OPEN\_INT'] = 0

resp2 = resp2.rename(columns={'OPEN\_INT': 'agreement'})

resp2['ticker'] = resp2.index

df\_put = pd.merge(df\_put, resp2, on='ticker')

#rajouter une colonne avec la volatilité implicite

#cas du call

resp4 = pd.DataFrame()

tickers4 = df\_call['ticker'].tolist()

if(datetime.date.today()==date\_hist):

resp4 = LocalTerminal.get\_historical(tickers4, 'IVOL\_MID',f'{today\_format\_compact}').as\_frame()

else:

resp4 = LocalTerminal.get\_historical(tickers4, 'IVOL\_MID',f'{today\_format\_compact}',f'{today\_format\_compact}').as\_frame()

resp4 = resp4.iloc[0]

new\_index = [index[0] for index in resp4.index]

resp4 = resp4.copy()

resp4.index = new\_index

resp4 = resp4.to\_frame() #car il s'agit d'une series

resp4.columns = [ 'implied\_vol']

resp4['ticker'] = resp4.index

df\_call= pd.merge(df\_call, resp4, on='ticker')

resp3 = pd.DataFrame()

tickers3 = df\_put['ticker'].tolist()

if(datetime.date.today()==date\_hist):

resp3 = LocalTerminal.get\_historical(tickers3, 'IVOL\_MID',f'{today\_format\_compact}').as\_frame()

else :

resp3 = LocalTerminal.get\_historical(tickers3, 'IVOL\_MID',f'{today\_format\_compact}',f'{today\_format\_compact}').as\_frame()

resp3 = resp3.iloc[0]

new\_index = [index[0] for index in resp3.index]

resp3 = resp3.copy()

resp3.index = new\_index

resp3 = resp3.to\_frame() #car il s'agit d'une series

resp3.columns = [ 'implied\_vol']

resp3['ticker'] = resp3.index

df\_put= pd.merge(df\_put, resp3, on='ticker')

#faire des interpolations lorsque le nombre de contrats est égale à 0 pour l'index en question

df\_call = clean\_dataframe\_interpolation(df\_call)

df\_put = clean\_dataframe\_interpolation(df\_put)

#faire un merge sur les strikes en commun

df\_put2 = df\_put.rename(columns={'ticker':'ticker put','strike': 'strike', 'agreement':'agreement put', 'implied\_vol' :'implied\_vol\_put'})

df\_call2 = df\_call.rename(columns={'ticker':'ticker call','strike': 'strike', 'agreement':'agreement call', 'implied\_vol' :'implied\_vol\_call'})

df\_call\_put = pd.merge(df\_call2, df\_put2, on = 'strike',how='inner') #avec correspondance uniquement

df\_call\_put = df\_call\_put.dropna()

#tracer le graphe skew call et put

dossier\_initial = r'U:\GDA\PFC\03\_Gerants\03\_12\_NB\skew-download'

dossier\_path = os.path.join(dossier\_initial, f"{today\_format\_compact}\_Options\_Expiring\_{next\_date\_format.replace('/', '-')}")

os.makedirs(dossier\_path, exist\_ok=True)

plt.plot(df\_call\_put['strike'], df\_call\_put['implied\_vol\_call'], 'b', label='Implied Volatility du Call')

plt.plot(df\_call\_put['strike'], df\_call\_put['implied\_vol\_put'], 'r', label='Implied Volatility du Put')

plt.axvline(x=spot, color='g', linestyle='--', label='Spot')

plt.text(spot, 0.5, 'ATM', color='g', va='center', ha='right')

plt.xlabel('Strike')

plt.ylabel('Volatility')

title = f'Implied Volatility : Call vs Put ({next\_date\_format})'

plt.title(title)

plt.legend()

graphique\_path = os.path.join(dossier\_path, 'call-put-skew.png')

plt.savefig(graphique\_path)

plt.close()

df\_call\_path = os.path.join(dossier\_path, 'df\_call.xlsx')

df\_call.to\_excel(df\_call\_path, index=False)

df\_call\_path\_h5 = os.path.join(dossier\_path, 'df\_call.h5')

metadata = dict(data="Ivol call",provider="Bloomberg",indice="SPX")

h5store(df\_call\_path\_h5,df\_call,metadata)

df\_put\_path = os.path.join(dossier\_path, 'df\_put.xlsx')

df\_put.to\_excel(df\_put\_path, index=False)

df\_put\_path\_h5 = os.path.join(dossier\_path, 'df\_put.h5')

metadata = dict(data="Ivol put",provider="Bloomberg",indice="SPX")

h5store(df\_put\_path\_h5,df\_put,metadata)

df\_call\_put\_path = os.path.join(dossier\_path, 'df\_call\_put.xlsx')

df\_call\_put.to\_excel(df\_call\_put\_path, index=False)

df\_call\_put\_path\_h5 = os.path.join(dossier\_path, 'df\_call\_put.h5')

metadata = dict(data="Ivol callput",provider="Bloomberg",indice="SPX")

h5store(df\_call\_put\_path\_h5 ,df\_call\_put,metadata)

spot\_path\_h5 = os.path.join(dossier\_path, 'spot.h5')

metadata = dict(data="Spot SPX",provider="Bloomberg",indice="SPX")

h5store(spot\_path\_h5,spot1,metadata)

plt.show()

return dossier\_path

def AMB\_mean(df\_call\_put, spot):

inf\_bas\_strike = int(round(spot \* 0.8, -1))

sup\_bas\_strike = int(round(spot \* 0.95, -1))

inf\_haut\_strike = int(round(spot \* 1.05, -1))

sup\_haut\_strike = int(round(spot \* 1.2, -1))

partie\_gauche = df\_call\_put[

(df\_call\_put['strike'] >= inf\_haut\_strike) & (df\_call\_put['strike'] <= sup\_haut\_strike)

]['implied\_vol\_put'].mean() + df\_call\_put[

(df\_call\_put['strike'] >= inf\_haut\_strike) & (df\_call\_put['strike'] <= sup\_haut\_strike)

]['implied\_vol\_call'].mean()

partie\_droite = df\_call\_put[

(df\_call\_put['strike'] >= inf\_bas\_strike) & (df\_call\_put['strike'] <= sup\_bas\_strike)

]['implied\_vol\_put'].mean() + df\_call\_put[

(df\_call\_put['strike'] >= inf\_bas\_strike) & (df\_call\_put['strike'] <= sup\_bas\_strike)

]['implied\_vol\_call'].mean()

AMB = (partie\_gauche - partie\_droite) / 2

return AMB

def COMA\_mean(df\_call\_put, spot):

inf\_bas\_strike = int(round(spot\*0.8,-1))

sup\_bas\_strike = int(round(spot\*0.95,-1))

inf\_atm = int(round(spot\*0.95,-1))

sup\_atm = int(round(spot\*1.05,-1))

inf\_haut\_strike = int(round(spot\*1.05,-1))

sup\_haut\_strike = int(round(spot\*1.2,-1))

coma = df\_call\_put[(df\_call\_put['strike']>= inf\_haut\_strike)&(df\_call\_put['strike']<=sup\_haut\_strike)]['implied\_vol\_call'].mean() - df\_call\_put[(df\_call\_put['strike']>= inf\_atm)&(df\_call\_put['strike']<=sup\_atm)]['implied\_vol\_call'].mean()

return coma

def POMA\_mean(df\_call\_put, spot):

inf\_bas\_strike = int(round(spot\*0.8,-1))

sup\_bas\_strike = int(round(spot\*0.95,-1))

inf\_atm = int(round(spot\*0.95,-1))

sup\_atm = int(round(spot\*1.05,-1))

poma = df\_call\_put[(df\_call\_put['strike']>= inf\_bas\_strike)&(df\_call\_put['strike']<=sup\_bas\_strike)]['implied\_vol\_put'].mean() - df\_call\_put[(df\_call\_put['strike']>= inf\_atm)&(df\_call\_put['strike']<=sup\_atm)]['implied\_vol\_put'].mean()

return poma

def CW\_mean(df\_call\_put, spot):

inf\_atm = int(round(spot\*0.95,-1))

sup\_atm = int(round(spot\*1.05,-1))

cw = df\_call\_put[(df\_call\_put['strike']>= inf\_atm)&(df\_call\_put['strike']<=sup\_atm)]['implied\_vol\_call'].mean() - df\_call\_put[(df\_call\_put['strike']>= inf\_atm)&(df\_call\_put['strike']<=sup\_atm)]['implied\_vol\_put'].mean()

return cw

def ZZX\_mean(df\_call\_put, spot):

inf\_bas\_strike = int(round(spot\*0.8,-1))

sup\_bas\_strike = int(round(spot\*0.95,-1))

inf\_atm = int(round(spot\*0.95,-1))

sup\_atm = int(round(spot\*1.05,-1))

zzx = df\_call\_put[(df\_call\_put['strike']>= inf\_bas\_strike)&(df\_call\_put['strike']<=sup\_bas\_strike)]['implied\_vol\_put'].mean() - df\_call\_put[(df\_call\_put['strike']>= inf\_atm)&(df\_call\_put['strike']<=sup\_atm)]['implied\_vol\_call'].mean()

return zzx

def COMA\_nearest(df\_call\_put, spot):

inf\_haut\_strike = int(round(spot \* 1.05, -1))

sup\_haut\_strike = int(round(spot \* 1.2, -1))

inf\_atm = int(round(spot\*0.95,-1))

sup\_atm = int(round(spot\*1.05,-1))

df\_call\_atm = df\_call\_put[(df\_call\_put['strike']>= inf\_atm) & (df\_call\_put['strike']<=sup\_atm)].sort\_values(by ='strike',ascending = True)

spot\_strike = df\_call\_atm['strike'].sub(spot).abs()

idx\_nearest\_strike = spot\_strike.idxmin()

call\_atm = df\_call\_atm.loc[idx\_nearest\_strike,'implied\_vol\_call']

df\_call\_otm = df\_call\_put[(df\_call\_put['strike'] >= inf\_haut\_strike) & (df\_call\_put['strike'] <= sup\_haut\_strike)].sort\_values(by ='strike',ascending=True)

call\_otm = df\_call\_otm.loc[df\_call\_otm.index[0], 'implied\_vol\_call']

COMA = call\_otm - call\_atm

return COMA

def POMA\_nearest(df\_call\_put, spot):

inf\_bas\_strike = int(round(spot\*0.8,-1))

sup\_bas\_strike = int(round(spot\*0.95,-1))

inf\_atm = int(round(spot\*0.95,-1))

sup\_atm = int(round(spot\*1.05,-1))

df\_put\_atm = df\_call\_put[(df\_call\_put['strike']>= inf\_atm) & (df\_call\_put['strike']<=sup\_atm)].sort\_values(by ='strike',ascending = True)

spot\_strike = df\_put\_atm['strike'].sub(spot).abs()

idx\_nearest\_strike = spot\_strike.idxmin()

put\_atm = df\_put\_atm.loc[idx\_nearest\_strike,'implied\_vol\_put']

df\_put\_otm = df\_call\_put[(df\_call\_put['strike']>= inf\_bas\_strike)&(df\_call\_put['strike']<=sup\_bas\_strike)].sort\_values(by ='strike',ascending=False)

put\_otm = df\_put\_otm.loc[df\_put\_otm.index[0], 'implied\_vol\_put']

POMA = put\_otm - put\_atm

return POMA

def CW\_nearest(df\_call\_put, spot):

inf\_atm = int(round(spot\*0.95,-1))

sup\_atm = int(round(spot\*1.05,-1))

df\_atm = df\_call\_put[(df\_call\_put['strike']>= inf\_atm) & (df\_call\_put['strike']<=sup\_atm)].sort\_values(by ='strike',ascending = True)

spot\_strike = df\_atm['strike'].sub(spot).abs()

idx\_nearest\_strike = spot\_strike.idxmin()

put\_atm = df\_atm.loc[idx\_nearest\_strike,'implied\_vol\_put']

call\_atm = df\_atm.loc[idx\_nearest\_strike,'implied\_vol\_call']

CW = call\_atm - put\_atm

return CW

def ABM\_nearest\_strike(df\_call\_put, spot):

inf\_haut\_strike = int(round(spot \* 1.05, -1))

inf\_bas\_strike = int(round(spot \* 0.8, -1))

sup\_haut\_strike = int(round(spot \* 1.2, -1))

sup\_bas\_strike = int(round(spot \* 0.95, -1))

df\_put\_itm = df\_call\_put[(df\_call\_put['strike'] >= inf\_haut\_strike) & (df\_call\_put['strike'] <= sup\_haut\_strike)].sort\_values(by ='strike',ascending=True)

put\_itm = df\_put\_itm.loc[df\_put\_itm.index[0], 'implied\_vol\_put']

df\_call\_otm = df\_call\_put[(df\_call\_put['strike'] >= inf\_haut\_strike) & (df\_call\_put['strike'] <= sup\_haut\_strike)].sort\_values(by ='strike',ascending=True)

call\_otm = df\_call\_otm.loc[df\_call\_otm.index[0], 'implied\_vol\_call']

df\_put\_otm = df\_call\_put[(df\_call\_put['strike'] <= sup\_bas\_strike) & (df\_call\_put['strike'] >= inf\_bas\_strike)].sort\_values(by ='strike',ascending=False)

put\_otm = df\_put\_otm.loc[df\_put\_otm.index[0], 'implied\_vol\_put']

df\_call\_itm = df\_call\_put[(df\_call\_put['strike'] <= sup\_bas\_strike) & (df\_call\_put['strike'] >= inf\_bas\_strike)].sort\_values(by ='strike',ascending=False)

call\_itm = df\_call\_itm.loc[df\_call\_itm.index[0], 'implied\_vol\_call']

AMB = ((put\_itm + call\_otm) - (call\_itm + put\_otm)) / 2

return AMB

def ZZX\_nearest(df\_call\_put, spot):

inf\_bas\_strike = int(round(spot\*0.8,-1))

sup\_bas\_strike = int(round(spot\*0.95,-1))

inf\_atm = int(round(spot\*0.95,-1))

sup\_atm = int(round(spot\*1.05,-1))

df\_call\_atm = df\_call\_put[(df\_call\_put['strike']>= inf\_atm) & (df\_call\_put['strike']<=sup\_atm)].sort\_values(by ='strike',ascending = True)

spot\_strike = df\_call\_atm['strike'].sub(spot).abs()

idx\_nearest\_strike = spot\_strike.idxmin()

call\_atm = df\_call\_atm.loc[idx\_nearest\_strike,'implied\_vol\_call']

df\_put\_otm = df\_call\_put[(df\_call\_put['strike']>= inf\_bas\_strike)&(df\_call\_put['strike']<=sup\_bas\_strike)].sort\_values(by ='strike',ascending=False)

put\_otm = df\_put\_otm.loc[df\_put\_otm.index[0], 'implied\_vol\_put']

ZZX = put\_otm - call\_atm

return ZZX

def compute\_metrics(dossier\_path, date\_hist):

df\_call\_put, metadata1 = h5load(dossier\_path+'\\df\_call\_put.h5')

df\_call, metadata2 = h5load(dossier\_path+ '\\df\_call.h5')

df\_put, metadata3 = h5load(dossier\_path+ '\\df\_put.h5')

df\_spot, metadata4 = h5load(dossier\_path+ '\\spot.h5')

spot = df\_spot.iloc[0,0]

date\_str = date\_hist.strftime('%Y%m%d')

df\_metrics = pd.DataFrame()

df\_metrics.loc[0,'Date'] = date\_str

#calcul de l'AMB

df\_metrics.loc[0,'Mean AMB'] = AMB\_mean(df\_call\_put, spot)

df\_metrics.loc[0,'AMB'] = ABM\_nearest\_strike(df\_call\_put, spot)

#calcul du COMA

df\_metrics.loc[0,'Mean COMA'] = COMA\_mean(df\_call\_put, spot)

df\_metrics.loc[0,'COMA'] = COMA\_nearest(df\_call\_put, spot)

#calcul du POMA

df\_metrics.loc[0,'Mean POMA'] = POMA\_mean(df\_call\_put, spot)

df\_metrics.loc[0,'POMA'] = POMA\_nearest(df\_call\_put, spot)

#calcul du CW

df\_metrics.loc[0, 'Mean CW'] = CW\_mean(df\_call\_put, spot)

df\_metrics.loc[0,'CW'] = CW\_nearest(df\_call\_put, spot)

#calcul du ZZX

df\_metrics.loc[0, 'Mean ZZX'] = ZZX\_mean(df\_call\_put, spot)

df\_metrics.loc[0, 'ZZX'] = ZZX\_nearest(df\_call\_put, spot)

#calcul des returns du mois suivants

#l'objectif de ces 5 lignes c'est d'avoir la date de fin de mois ouvré du mois suivant pour pouvoir calculer la perf du mois suivant

date\_plus\_2month = date\_hist+ relativedelta(months=2)

first\_day\_next\_month = datetime.date(date\_plus\_2month.year, date\_plus\_2month.month, 1)

date\_suiv = first\_day\_next\_month - datetime.timedelta(days=10)

date\_plus\_2month = date\_plus\_2month.strftime('%Y%m%d')

liste = fin\_de\_mois\_jours\_ouvres( date\_suiv, date\_plus\_2month)

spot\_in\_1month = LocalTerminal.get\_historical('SPX Index', 'PX\_LAST',f'{liste[0]}',f'{liste[0]}').as\_frame()

spot\_in\_1month = spot\_in\_1month.iloc[0,0]

return\_1m = (spot\_in\_1month-spot)/spot

df\_metrics.loc[0,'Future return'] = return\_1m

return df\_metrics

def clean\_dataframe\_interpolation(df):

#si des strikes sont en doublons, je garde la ligne qui contient le nombre d'agreements le plus important

keep\_lines = {}

for index, row in df.iterrows():

strike = row["strike"]

if strike in keep\_lines:

if row["agreement"] > keep\_lines[strike]["agreement"]:

keep\_lines[strike] = row

else:

keep\_lines[strike] = row

df = pd.DataFrame.from\_dict(keep\_lines, orient="index").reset\_index(drop=True)

i = 0

while(df.loc[i,'agreement'] == 0) :

df = df.drop(i,axis=0)

print("supression de la première ligne contenant un agreement = 0 , interpolation impossible, pas de data avant")

i +=1

i = len(df.index)-1

while(df.loc[i,'agreement'] == 0) :

df = df.drop(i,axis=0)

print("supression de la dernière ligne contenant un agreement = 0 , interpolation impossible, pas de data après")

i -=1

for index\_ticker in df.index:

if df.loc[index\_ticker,'agreement'] == 0 :

prec\_vol = df.loc[index\_ticker- 1, 'implied\_vol']

prec\_strike = df.loc[index\_ticker - 1, 'strike']

i = index\_ticker +1

#on doit selectionner la volatilité suivante dont le nombre d'agreement est =! 0

while(df.loc[i,'agreement'] == 0):

i+=1

suiv\_vol = df.loc[i, 'implied\_vol']

suiv\_strike = df.loc[i, 'strike']

interpo = prec\_vol+(suiv\_vol-prec\_vol)\*((df.loc[index\_ticker, 'strike']-prec\_strike)/(suiv\_strike-prec\_strike))

df.loc[index\_ticker, 'implied\_vol'] = interpo

return df

def fin\_de\_mois\_jours\_ouvres(datedeb, datefin):

bbg = pybbg.Pybbg()

df= pd.DataFrame()

df = bbg.bdh("SPX Index", 'PX\_LAST', datedeb, datefin)

df['Date'] = df.index

df = df.reset\_index(drop=True)

liste = []

for i in range(0,len(df.index)-2) :

if(df.loc[i,'Date'].month != df.loc[i+1,'Date'].month):

liste.append(df.loc[i,'Date'])

return liste

def download\_data(datedeb, datefin):

liste\_fin\_mois = fin\_de\_mois\_jours\_ouvres(datedeb, datefin)

for i in liste\_fin\_mois :

download\_vol\_data(i.date())

return

#download\_data(20240101,20240605)

df\_metrics\_decembre =compute\_metrics(r'U:\GDA\PFC\03\_Gerants\03\_12\_NB\skew-download\20231229\_Options\_Expiring\_02-16-24', datetime.date(2023,12,29))

df\_metrics\_janvier =compute\_metrics(r'U:\GDA\PFC\03\_Gerants\03\_12\_NB\skew-download\20240131\_Options\_Expiring\_03-15-24', datetime.date(2024,1,31))

df\_metrics\_fevrier =compute\_metrics(r'U:\GDA\PFC\03\_Gerants\03\_12\_NB\skew-download\20240229\_Options\_Expiring\_04-19-24', datetime.date(2024,2,29))

df\_metrics\_mars = compute\_metrics(r'U:\GDA\PFC\03\_Gerants\03\_12\_NB\skew-download\20240328\_Options\_Expiring\_05-17-24', datetime.date(2024,3,28))

df\_metrics\_avril = compute\_metrics(r'U:\GDA\PFC\03\_Gerants\03\_12\_NB\skew-download\20240430\_Options\_Expiring\_06-21-24',datetime.date(2024, 4, 30))

dataframes = [df\_metrics\_decembre, df\_metrics\_janvier, df\_metrics\_fevrier, df\_metrics\_mars, df\_metrics\_avril]

df\_merged = pd.concat(dataframes, ignore\_index=True)

print(df\_merged)

dossier\_path = r'U:\GDA\PFC\03\_Gerants\03\_12\_NB\skew-download'

df\_merged\_path= os.path.join(dossier\_path, 'resultats\_metrics\_IVOL.xlsx')

df\_merged.to\_excel(df\_merged\_path, index=False)

df = df\_merged

df['Date'] = pd.to\_datetime(df['Date'], format='%Y%m%d')

plt.figure(figsize=(12, 6))

x = np.arange(len(df))

width = 0.15

plt.bar(x - 2\*width, df['AMB'], width=width, label='AMB')

plt.bar(x - width, df['ZZX'], width=width, label='ZZX')

plt.bar(x, df['COMA'], width=width, label='COMA')

plt.bar(x + width, df['POMA'], width=width, label='POMA')

plt.bar(x + 2\*width, df['CW'], width=width, label='CW')

for i, future\_return in enumerate(df['Future return']):

plt.text(i, df['AMB'][i] + df['ZZX'][i] + df['COMA'][i] + df['POMA'][i] + df['CW'][i] + 0.2, f"{future\_return\*100:.2f}", ha='center')

plt.plot(x, 100\*df['Future return'], color='red', label='Future return en %')

plt.xticks(x, df['Date'], rotation=45)

plt.legend()

plt.title('relation entre les métriques sur la vol imp et le rendement mensuel du mois suivant')

plt.xlabel('Date')

plt.ylabel('Valeur')

plt.show()

Small vs large :

#importation des librairies nécessaires

from tia.bbg import LocalTerminal

import pandas as pd

import datetime

from xbbg import blp

from blp import blp as bld

import pybbg as pybbg

import matplotlib.pyplot as plt

import os

from sklearn.metrics import mean\_absolute\_error

from dateutil.relativedelta import relativedelta

import numpy as np

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix

import shap

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import accuracy\_score

from sklearn.linear\_model import LogisticRegressionCV

from datetime import datetime, timedelta, date

from tqdm import tqdm

import shutil

import re

working\_folder = r"U:\GDA\PFC\03\_Gerants\03\_12\_NB\Small VS Large"

class PrevisionError(Exception):

pass

def create\_or\_replace\_directory(directory\_path):

if os.path.exists(directory\_path):

shutil.rmtree(directory\_path)

os.makedirs(directory\_path)

def h5store(filename, df, dic):

store = pd.HDFStore(filename)

store.put('mydata', df)

store.get\_storer('mydata').attrs.metadata = dic

store.close()

def h5load(filename):

with pd.HDFStore(filename) as store:

data = store['mydata']

metadata = store.get\_storer('mydata').attrs.metadata

data.attrs = metadata

return data, metadata

def find\_latest\_file(working\_folder, prefix): #trouver le dernier (le + récent) fichier .hdf5 qui comportent les données des index LCXR et SCXR

data\_index\_folder = os.path.join(working\_folder, 'data\_index')

files = os.listdir(data\_index\_folder)

filtered\_files = [f for f in files if f.startswith(f'df\_{prefix}') and f.endswith('.h5')]

def extract\_date(filename):

match = re.search(r'(\d{8})\.h5$', filename)

if match:

return match.group(1)

return None

latest\_file = max(filtered\_files, key=lambda f: extract\_date(f))

latest\_file\_path = os.path.join(data\_index\_folder, latest\_file)

return latest\_file\_path

#PARTIE CHARGEMENT DE LA DATA

#à partir de deux dates, on trouve la date de début de mois où il y a eu une cotation

def find\_prochaine\_date\_mois(datedeb, datefin):

bbg = pybbg.Pybbg()

df= pd.DataFrame()

df = bbg.bdh("LCXR Index", 'PX\_LAST', datedeb, datefin)

df['Date'] = df.index

df = df.reset\_index(drop=True)

for i in range(1,len(df.index)) :

if(df.loc[i,'Date'].month != df.loc[i-1,'Date'].month):

return df.loc[i,'Date']

raise ValueError("Aucun changement de mois n'a été trouvé entre les dates fournies (rendement inconnus à cette date) ",datefin)

def download\_data\_per(index,datefin, working\_folder = working\_folder) :

#récup les poids dans l'index à une certaine date

datefin\_format\_tirets = f"{datefin[:4]}-{datefin[4:6]}-{datefin[6:]}"

bbg = pybbg.Pybbg()

bquery = bld.BlpQuery().start()

df\_member = bquery.bql(f"for(MEMBERS('{index}',DATES='{datefin\_format\_tirets}')) get(ID().WEIGHTS)")

df\_member = df\_member.rename(columns={'security': 'index\_member'})

df\_member = df\_member.drop(['field', 'secondary\_name', 'secondary\_value'], axis=1)

df\_member = df\_member.rename(columns={'value': 'weight'})

print(df\_member)

#recup les secteurs

sector = blp.bdp(df\_member['index\_member'], "GICS\_SECTOR\_NAME")

sector['index\_member'] = sector.index

df\_member = pd.merge(df\_member, sector, on = 'index\_member', how='left')

#recup la dette short et long terme

lt\_debt = bquery.bql(f"for(members('{index}',dates='{datefin\_format\_tirets}')) get(bs\_lt\_borrow(dates='{datefin\_format\_tirets}', AE=A, fill=prev, currency=EUR).value)")

st\_debt = bquery.bql(f"for(members('{index}',dates='{datefin\_format\_tirets}')) get(bs\_st\_borrow(dates='{datefin\_format\_tirets}', AE=A, fill=prev, currency=EUR).value)")

lt\_debt= lt\_debt.groupby('security').agg(list)

st\_debt= st\_debt.groupby('security').agg(list)

for i in range(max(st\_debt['value'].str.len())):

st\_debt[f'value\_{i+1}'] = st\_debt['value'].apply(lambda x: x[i] if len(x) > i else '')

for i in range(max(lt\_debt['value'].str.len())):

lt\_debt[f'value\_{i+1}'] = lt\_debt['value'].apply(lambda x: x[i] if len(x) > i else '')

lt\_debt = lt\_debt.drop('value', axis=1)

lt\_debt = lt\_debt.rename(columns={

'value\_1': 'lt\_debt',

})

st\_debt = st\_debt.drop('value', axis=1)

st\_debt = st\_debt.rename(columns={

'value\_1': 'st\_debt',

})

lt\_debt= lt\_debt.drop(['field', 'secondary\_name', 'secondary\_value'], axis=1)

st\_debt= st\_debt.drop(['field', 'secondary\_name', 'secondary\_value'], axis=1)

df\_member = pd.merge(df\_member, lt\_debt,left\_on='index\_member', right\_on='security', how='left')

df\_member = pd.merge(df\_member, st\_debt,left\_on='index\_member', right\_on='security', how='left')

#calcul la somme lt debt et st term debt

df\_member['st\_lt\_debt'] = df\_member['lt\_debt'] + df\_member['st\_debt']

#recup le wacc

wacc = blp.bdh(df\_member['index\_member'],'WACC\_COST\_DEBT',datefin, datefin, Days = 'NON\_TRADING\_WEEKDAYS' )

wacc['index\_member'] = wacc.index

wacc = wacc.iloc[0]

new\_index = [index[0] for index in wacc.index]

new\_index = new\_index.copy()

wacc.index = new\_index

wacc['index\_member'] = wacc.index

wacc = wacc[:-1]

wacc = wacc.rename('wacc')

df\_member = pd.merge(df\_member, wacc,left\_on='index\_member', right\_on=wacc.index, how='left')

#recup le currency adjusted enterprise value

crncy = bquery.bql(f"for(members('{index}',dates='{datefin\_format\_tirets}')) get(curr\_entp\_val(dates='{datefin\_format\_tirets}', fill=prev, currency = EUR).value)")

crncy= crncy.groupby('security').agg(list)

for i in range(max(crncy['value'].str.len())):

crncy[f'value\_{i+1}'] = crncy['value'].apply(lambda x: x[i] if len(x) > i else '')

crncy = crncy.drop('value', axis=1)

crncy= crncy.rename(columns={'value\_1': 'crncy\_adj\_ev' })

crncy= crncy.drop(['field', 'secondary\_name', 'secondary\_value'], axis=1)

df\_member = pd.merge(df\_member, crncy,left\_on='index\_member', right\_on='security', how='left')

#recup le long terme price earnings

lt\_pe = blp.bdh(df\_member['index\_member'],'LONG\_TERM\_PRICE\_EARNINGS\_RATIO',datefin, datefin, Days = 'NON\_TRADING\_WEEKDAYS' )

lt\_pe['index\_member'] = lt\_pe.index

if(len(lt\_pe)!= 0):

lt\_pe= lt\_pe.iloc[0]

new\_index = [index[0] for index in lt\_pe.index]

new\_index = new\_index.copy()

lt\_pe.index = new\_index

lt\_pe['index\_member'] = lt\_pe.index

lt\_pe= lt\_pe[:-1]

lt\_pe = lt\_pe.rename('lt\_pe')

df\_member= pd.merge(df\_member, lt\_pe,left\_on='index\_member', right\_on=lt\_pe.index, how='left')

#recup l'earnings, pe ratio et best pe ratio

bbg = pybbg.Pybbg()

bquery = bld.BlpQuery().start()

#df\_pe = bquery.bql(f"for(members('{index}',dates='{datefin\_format\_tirets}')) get(cur\_mkt\_cap(dates='{datefin\_format\_tirets}', fill=prev, currency=EUR).value,(pe\_ratio(FPT=LTM,FPO=0, AE=E, dates='{datefin\_format\_tirets}', fill=prev).value),(pe\_ratio(dates = '{datefin\_format\_tirets}', fill=prev).value))")

df\_pe = bquery.bql(f"for(members('{index}',dates='{datefin\_format\_tirets}')) get(cur\_mkt\_cap(dates='{datefin\_format\_tirets}', fill=prev, currency=EUR).value,(pe\_ratio(FPT=LTM,FPO=1, AE=E, as\_of\_date='{datefin\_format\_tirets}', fill=prev).value),(pe\_ratio(as\_of\_date = '{datefin\_format\_tirets}', fill=prev).value))")

print(df\_member)

df\_pe = df\_pe.groupby('security').agg(list)

for i in range(max(df\_pe['value'].str.len())):

df\_pe[f'value\_{i+1}'] = df\_pe['value'].apply(lambda x: x[i] if len(x) > i else '')

df\_pe = df\_pe.drop('value', axis=1)

df\_pe = df\_pe.rename(columns={

'value\_1': 'cur\_mrk\_cap',

'value\_2': 'best\_pe\_ratio',

'value\_3': 'pe\_ratio'

})

df\_pe = df\_pe.drop(['field', 'secondary\_name', 'secondary\_value'], axis=1)

df\_member = pd.merge(df\_member, df\_pe,left\_on='index\_member', right\_on='security', how='left')

print(df\_member)

df\_member.replace('', np.nan, inplace=True)

#df\_member['best\_pe\_ratio'] = df\_member['best\_pe\_ratio'].fillna(df\_member['pe\_ratio'])

for i, value in enumerate(df\_member['best\_pe\_ratio']):

if pd.isna(value) or value == '' or value == 'NaN' or value =='nan':

df\_member.at[i, 'best\_pe\_ratio'] = df\_member.at[i, 'pe\_ratio']

for i, value in enumerate(df\_member['pe\_ratio']):

if pd.isna(value) or value == '' or value == 'NaN' or value =='nan':

df\_member.at[i, 'pe\_ratio'] = df\_member.at[i, 'best\_pe\_ratio']

dossier\_deb = working\_folder

dossier\_initial = os.path.join(dossier\_deb, f"members\_{datefin}")

os.makedirs(dossier\_initial, exist\_ok=True)

dossier\_path = os.path.join(dossier\_initial, f"{index}\_members\_")

os.makedirs(dossier\_path, exist\_ok=True)

file\_name\_xlsx = f"df\_member\_{index}\_{datefin\_format\_tirets}.xlsx"

df\_member\_path = os.path.join(dossier\_path, file\_name\_xlsx)

df\_member.to\_excel(df\_member\_path, index=False)

file\_name\_h5 = f"df\_member\_{index}\_{datefin\_format\_tirets}.h5"

df\_member\_path\_h5 = os.path.join(dossier\_path, file\_name\_h5)

metadata = dict(data="data index member",provider="Bloomberg",indice="SXXR")

h5store(df\_member\_path\_h5 ,df\_member,metadata)

#fonction pour calculer une nouvelle version du pe

def compute\_pe\_u(index,datefin, compute\_futur\_return = True, working\_folder = working\_folder):

datefin\_format\_tirets = f"{datefin[:4]}-{datefin[4:6]}-{datefin[6:]}"

bbg = pybbg.Pybbg()

bquery = bld.BlpQuery().start()

file\_name\_h5 = f"df\_member\_{index}\_{datefin\_format\_tirets}.h5"

dossier\_deb = working\_folder

dossier\_initial = os.path.join(dossier\_deb, f"members\_{datefin}")

os.makedirs(dossier\_initial, exist\_ok=True)

dossier\_path = os.path.join(dossier\_initial, f"{index}\_members\_")

df\_member, metadata1 = h5load(os.path.join(dossier\_path, file\_name\_h5))

print(df\_member)

if(compute\_futur\_return==True):

date\_obj = datetime.strptime(datefin, "%Y%m%d")

date\_next\_month = date\_obj + relativedelta(days=35)

date1mois = date\_next\_month.strftime("%Y%m%d")

next\_date = find\_prochaine\_date\_mois(datefin, date1mois)

next\_date = next\_date.strftime("%Y%m%d")

next\_month\_return = LocalTerminal.get\_reference\_data(df\_member['index\_member'],"CUST\_TRR\_RETURN\_HOLDING\_PER", CUST\_TRR\_END\_DT=next\_date, CUST\_TRR\_START\_DT=datefin, CUST\_TRR\_CRNCY="EUR").as\_frame()

df\_member = pd.merge(df\_member, next\_month\_return, left\_on='index\_member', right\_on=next\_month\_return.index, how='left')

df\_member = df\_member.rename(columns={'CUST\_TRR\_RETURN\_HOLDING\_PER': 'next\_month\_return'})

#convertir les lignes en numeric

cols\_to\_convert = ['cur\_mrk\_cap', 'st\_lt\_debt', 'crncy\_adj\_ev', 'wacc', 'best\_pe\_ratio', 'pe\_ratio']

df\_member[cols\_to\_convert] = df\_member[cols\_to\_convert].apply(pd.to\_numeric, errors='coerce')

df\_member.replace('', np.nan, inplace=True)

#df\_member['best\_pe\_ratio'] = df\_member['best\_pe\_ratio'].fillna(df\_member['pe\_ratio'])

#calcul du pe\_u à partir de la formule donnée

df\_member['pe\_u'] = df\_member.apply(lambda row:

row['cur\_mrk\_cap'] /

((row['st\_lt\_debt'] / row['crncy\_adj\_ev']) \*

(((row['wacc']/100) \* row['cur\_mrk\_cap']) - (row['cur\_mrk\_cap'] / row['best\_pe\_ratio'])) +

(row['cur\_mrk\_cap'] / row['best\_pe\_ratio'])) if not np.isnan(row['cur\_mrk\_cap']) and not np.isnan(row['st\_lt\_debt']) and not np.isnan(row['crncy\_adj\_ev']) and not np.isnan(row['wacc']) and not np.isnan(row['best\_pe\_ratio']) else np.nan,

axis=1)

#df\_member['pe\_u'] = df\_member['cur\_mrk\_cap']/((df\_member['st\_lt\_debt']/df\_member['crncy\_adj\_ev'])\*(df\_member['wacc']\*df\_member['cur\_mrk\_cap']-(df\_member['cur\_mrk\_cap']/df\_member['best\_pe\_ratio']))+(df\_member['cur\_mrk\_cap']/df\_member['best\_pe\_ratio']))

file\_name\_xlsx = f"df\_pe\_u\_{index}\_{datefin\_format\_tirets}.xlsx"

df\_member\_path = os.path.join(dossier\_path, file\_name\_xlsx)

df\_member.to\_excel(df\_member\_path, index=False)

file\_name\_h5 = f"df\_pe\_u\_{index}\_{datefin\_format\_tirets}.h5"

df\_member\_path\_h5 = os.path.join(dossier\_path, file\_name\_h5)

metadata = dict(data="data index member pe\_u",provider="Bloomberg",indice="SXXR")

h5store(df\_member\_path\_h5 ,df\_member,metadata)

return 0

#compare\_pe\_u qui permet de créer un fichier .h5 et excel pour chaque date de début de mois commun aux larges, mids et smalls caps.

#avec notamment le pe median,le pe\_u median, les returns du mois suivant par catégorie (large, small et mids) et par secteur

def compare\_pe\_u(datefin, compute\_futur\_return=True, working\_folder = working\_folder):

datefin\_format\_tirets = f"{datefin[:4]}-{datefin[4:6]}-{datefin[6:]}"

bbg = pybbg.Pybbg()

bquery = bld.BlpQuery().start()

dossier\_deb = working\_folder

dossier\_initial = os.path.join(dossier\_deb, f"members\_{datefin}")

os.makedirs(dossier\_initial, exist\_ok=True)

dossier\_path\_scxr = os.path.join(dossier\_initial, f"SCXR Index\_members\_")

file\_name\_h5\_scxr = f"df\_pe\_u\_SCXR Index\_{datefin\_format\_tirets}.h5"

file\_scxr = os.path.join(dossier\_path\_scxr, file\_name\_h5\_scxr)

if os.path.exists(file\_scxr):

print("Le fichier SCXR existe.")

else:

print("Le fichier SCXR n'existe pas à la date demandée.")

return 0

dossier\_path\_mcxr = os.path.join(dossier\_initial, f"MCXR Index\_members\_")

file\_name\_h5\_mcxr = f"df\_pe\_u\_MCXR Index\_{datefin\_format\_tirets}.h5"

file\_mcxr = os.path.join(dossier\_path\_mcxr, file\_name\_h5\_mcxr)

if os.path.exists(file\_mcxr):

print("Le fichier MCXR existe.")

else:

print("Le fichier MCXR n'existe pas à la date demandée.")

return 0

dossier\_path\_lcxr = os.path.join(dossier\_initial, f"LCXR Index\_members\_")

file\_name\_h5\_lcxr = f"df\_pe\_u\_LCXR Index\_{datefin\_format\_tirets}.h5"

file\_lcxr = os.path.join(dossier\_path\_lcxr, file\_name\_h5\_lcxr)

if os.path.exists(file\_lcxr):

print("Le fichier LCXR existe.")

else:

print("Le fichier LCXR n'existe pas à la date demandée.")

return 0

df\_scxr, metadata1 = h5load(os.path.join(dossier\_path\_scxr, file\_name\_h5\_scxr))

df\_mcxr, metadata2 = h5load(os.path.join(dossier\_path\_mcxr, file\_name\_h5\_mcxr))

df\_lcxr, metadata3 = h5load(os.path.join(dossier\_path\_lcxr, file\_name\_h5\_lcxr))

df\_scxr[df\_scxr == 'NaN'] = np.nan

df\_mcxr[df\_mcxr == 'NaN'] = np.nan

df\_lcxr[df\_lcxr == 'NaN'] = np.nan

df\_scxr.replace('', np.nan, inplace=True)

df\_mcxr.replace('', np.nan, inplace=True)

df\_lcxr.replace('', np.nan, inplace=True)

df = pd.DataFrame()

categories = df\_scxr['gics\_sector\_name'].unique()

index = 0

for categorie in categories :

if categorie != 'Financials':

#sélection des dataframes correspondants

print(categorie)

category\_scxr = df\_scxr[df\_scxr['gics\_sector\_name']==categorie]

category\_scxr = category\_scxr[category\_scxr['pe\_u'].notna()]

category\_scxr = category\_scxr[category\_scxr['pe\_u'] != '']

category\_mcxr = df\_mcxr[df\_mcxr['gics\_sector\_name']==categorie]

category\_mcxr = category\_mcxr[category\_mcxr['pe\_u'].notna()]

category\_mcxr = category\_mcxr[category\_mcxr['pe\_u'] != '']

category\_lcxr = df\_lcxr[df\_lcxr['gics\_sector\_name']==categorie]

category\_lcxr = category\_lcxr[category\_lcxr['pe\_u'].notna()]

category\_lcxr = category\_lcxr[category\_lcxr['pe\_u'] != '']

df.loc[index, 'secteur'] = categorie

#calcul du pe median par catégorie

df.loc[index, 'Median SCXR pe'] = category\_scxr['pe\_ratio'].median()

df.loc[index, 'Median MCXR pe'] = category\_mcxr['pe\_ratio'].median()

df.loc[index, 'Median LCXR pe'] = category\_lcxr['pe\_ratio'].median()

#calcul du pe\_u médian par catégorie

df.loc[index, 'Median SCXR pe\_u'] = category\_scxr['pe\_u'].median()

df.loc[index, 'Median MCXR pe\_u'] = category\_mcxr['pe\_u'].median()

df.loc[index, 'Median LCXR pe\_u'] = category\_lcxr['pe\_u'].median()

#calcul de pe en faisant une pondération par market cap par catégorie

df.loc[index, 'SCXR pe mrk cap weighted'] = ((category\_scxr['cur\_mrk\_cap'] / category\_scxr['cur\_mrk\_cap'].sum()) \* category\_scxr['pe\_ratio']).sum()

df.loc[index, 'MCXR pe mrk cap weighted'] = ((category\_mcxr['cur\_mrk\_cap'] / category\_mcxr['cur\_mrk\_cap'].sum()) \* category\_mcxr['pe\_ratio']).sum()

df.loc[index, 'LCXR pe mrk cap weighted'] = ((category\_lcxr['cur\_mrk\_cap'] / category\_lcxr['cur\_mrk\_cap'].sum()) \* category\_lcxr['pe\_ratio']).sum()

#calcul de pe en faisant une pondération par market cap par catégorie

df.loc[index, 'SCXR pe\_u mrk cap weighted'] = ((category\_scxr['cur\_mrk\_cap'] / category\_scxr['cur\_mrk\_cap'].sum()) \* category\_scxr['pe\_u']).sum()

df.loc[index, 'MCXR pe\_u mrk cap weighted'] = ((category\_mcxr['cur\_mrk\_cap'] / category\_mcxr['cur\_mrk\_cap'].sum()) \* category\_mcxr['pe\_u']).sum()

df.loc[index, 'LCXR pe\_u mrk cap weighted'] = ((category\_lcxr['cur\_mrk\_cap'] / category\_lcxr['cur\_mrk\_cap'].sum()) \* category\_lcxr['pe\_u']).sum()

#calcul returns du mois suivant de la cat

if(compute\_futur\_return == True):

category\_scxr = category\_scxr[category\_scxr['next\_month\_return'].notna()]

category\_scxr = category\_scxr[category\_scxr['next\_month\_return'] != '']

category\_mcxr = category\_mcxr[category\_mcxr['next\_month\_return'].notna()]

category\_mcxr = category\_mcxr[category\_mcxr['next\_month\_return'] != '']

category\_lcxr = category\_lcxr[category\_lcxr['next\_month\_return'].notna()]

category\_lcxr = category\_lcxr[category\_lcxr['next\_month\_return'] != '']

df.loc[index, 'SCXR next month return'] = ((category\_scxr['cur\_mrk\_cap'] / category\_scxr['cur\_mrk\_cap'].sum()) \* category\_scxr['next\_month\_return']).sum()

df.loc[index, 'MCXR next month return'] = ((category\_mcxr['cur\_mrk\_cap'] / category\_mcxr['cur\_mrk\_cap'].sum()) \* category\_mcxr['next\_month\_return']).sum()

df.loc[index, 'LCXR next month return'] = ((category\_lcxr['cur\_mrk\_cap'] / category\_lcxr['cur\_mrk\_cap'].sum()) \* category\_lcxr['next\_month\_return']).sum()

index +=1

df\_scxr = df\_scxr[df\_scxr['pe\_u'].notna() & (df\_scxr['pe\_u'] != '') & (df\_scxr['pe\_u'] != 'NaN') & (df\_scxr['pe\_u'] != 'Na') & (~df\_scxr['pe\_u'].isin(['nan']))]

df\_scxr = df\_scxr[df\_scxr['pe\_ratio'].notna() & (df\_scxr['pe\_ratio'] != '') & (df\_scxr['pe\_ratio'] != 'NaN') & (df\_scxr['pe\_ratio'] != 'Na') & (~df\_scxr['pe\_ratio'].isin(['nan']))]

df\_mcxr = df\_mcxr[df\_mcxr['pe\_u'].notna() & (df\_mcxr['pe\_u'] != '') & (df\_mcxr['pe\_u'] != 'NaN') & (df\_mcxr['pe\_u'] != 'Na') & (~df\_mcxr['pe\_u'].isin(['nan']))]

df\_mcxr = df\_mcxr[df\_mcxr['pe\_ratio'].notna() & (df\_mcxr['pe\_ratio'] != '') & (df\_mcxr['pe\_ratio'] != 'NaN') & (df\_mcxr['pe\_ratio'] != 'Na') & (~df\_mcxr['pe\_ratio'].isin(['nan']))]

df\_lcxr = df\_lcxr[df\_lcxr['pe\_u'].notna() & (df\_lcxr['pe\_u'] != '') & (df\_lcxr['pe\_u'] != 'NaN') & (df\_lcxr['pe\_u'] != 'Na') & (~df\_lcxr['pe\_u'].isin(['nan']))]

df\_lcxr = df\_lcxr[df\_lcxr['pe\_ratio'].notna() & (df\_lcxr['pe\_ratio'] != '') & (df\_lcxr['pe\_ratio'] != 'NaN') & (df\_lcxr['pe\_ratio'] != 'Na') & (~df\_lcxr['pe\_ratio'].isin(['nan']))]

df.loc[index, 'secteur'] = 'Full Index'

df.loc[index, 'Median SCXR pe'] = df\_scxr['pe\_ratio'].median()

df.loc[index, 'Median MCXR pe'] = df\_mcxr['pe\_ratio'].median()

df.loc[index, 'Median LCXR pe'] = df\_lcxr['pe\_ratio'].median()

df.loc[index, 'Median SCXR pe\_u'] = df\_scxr['pe\_u'].median()

df.loc[index, 'Median MCXR pe\_u'] = df\_mcxr['pe\_u'].median()

df.loc[index, 'Median LCXR pe\_u'] = df\_lcxr['pe\_u'].median()

df.loc[index, 'SCXR pe mrk cap weighted'] = ((df\_scxr['cur\_mrk\_cap'] / df\_scxr['cur\_mrk\_cap'].sum()) \* df\_scxr['pe\_ratio']).sum()

df.loc[index, 'MCXR pe mrk cap weighted'] = ((df\_mcxr['cur\_mrk\_cap'] / df\_mcxr['cur\_mrk\_cap'].sum()) \* df\_mcxr['pe\_ratio']).sum()

df.loc[index, 'LCXR pe mrk cap weighted'] = ((df\_lcxr['cur\_mrk\_cap'] / df\_lcxr['cur\_mrk\_cap'].sum()) \* df\_lcxr['pe\_ratio']).sum()

df.loc[index, 'SCXR pe\_u mrk cap weighted'] = ((df\_scxr['cur\_mrk\_cap'] / df\_scxr['cur\_mrk\_cap'].sum()) \* df\_scxr['pe\_u']).sum()

df.loc[index, 'MCXR pe\_u mrk cap weighted'] = ((df\_mcxr['cur\_mrk\_cap'] / df\_mcxr['cur\_mrk\_cap'].sum()) \* df\_mcxr['pe\_u']).sum()

df.loc[index, 'LCXR pe\_u mrk cap weighted'] = ((df\_lcxr['cur\_mrk\_cap'] / df\_lcxr['cur\_mrk\_cap'].sum()) \* df\_lcxr['pe\_u']).sum()

if(compute\_futur\_return == True):

df\_scxr = df\_scxr[df\_scxr['next\_month\_return'].notna() & (df\_scxr['next\_month\_return'] != '')]

df\_mcxr = df\_mcxr[df\_mcxr['next\_month\_return'].notna() & (df\_mcxr['next\_month\_return'] != '')]

df\_lcxr = df\_lcxr[df\_lcxr['next\_month\_return'].notna() & (df\_lcxr['next\_month\_return'] != '')]

df.loc[index, 'SCXR next month return'] = ((df\_scxr['cur\_mrk\_cap'] / df\_scxr['cur\_mrk\_cap'].sum()) \* df\_scxr['next\_month\_return']).sum()

df.loc[index, 'MCXR next month return'] = ((df\_mcxr['cur\_mrk\_cap'] / df\_mcxr['cur\_mrk\_cap'].sum()) \* df\_mcxr['next\_month\_return']).sum()

df.loc[index, 'LCXR next month return'] = ((df\_lcxr['cur\_mrk\_cap'] / df\_lcxr['cur\_mrk\_cap'].sum()) \* df\_lcxr['next\_month\_return']).sum()

file\_name\_xlsx = f"df\_score\_pe\_{datefin\_format\_tirets}.xlsx"

df\_score\_path = os.path.join(dossier\_initial, file\_name\_xlsx)

df.to\_excel(df\_score\_path, index=False)

file\_name\_h5 = f"df\_score\_pe\_{datefin\_format\_tirets}.h5"

df\_score\_path\_h5 = os.path.join(dossier\_initial, file\_name\_h5)

metadata = dict(data="results pe\_u",provider="Bloomberg",indice="SXXR")

h5store(df\_score\_path\_h5 ,df,metadata)

print(df)

return 0

#fonction à appeler pour charger la data entre deux données et mettre la data dans un folder

def deb\_de\_mois\_jours\_ouvres(datedeb, datefin):

bbg = pybbg.Pybbg()

df= pd.DataFrame()

df = bbg.bdh("LCXR Index", 'PX\_LAST', datedeb, datefin)

df['Date'] = df.index

df = df.reset\_index(drop=True)

liste = []

for i in range(1,len(df.index)) :

if(df.loc[i,'Date'].month != df.loc[i-1,'Date'].month):

liste.append(df.loc[i,'Date'])

liste = [date.strftime('%Y%m%d') for date in liste]

for i in liste :

download\_data\_per('SCXR Index', i)

download\_data\_per('LCXR Index', i)

download\_data\_per('MCXR Index', i)

now = datetime.now()

date\_obj = datetime.strptime(i, "%Y%m%d")

if date\_obj.month == now.month and date\_obj.year == now.year:

compute\_pe\_u('SCXR Index', i, False)

compute\_pe\_u('MCXR Index', i, False)

compute\_pe\_u('LCXR Index', i, False)

compare\_pe\_u(i, False)

else :

compute\_pe\_u('SCXR Index', i)

compute\_pe\_u('MCXR Index', i)

compute\_pe\_u('LCXR Index', i)

compare\_pe\_u(i)

#fonction à appeler pour télécharger l'ensemble des données entre la date1 et la date2 en pas mensuel début de mois de jour ouvrés

#deb\_de\_mois\_jours\_ouvres(20240625,20240728)

#PARTIE CALCUL POUR LA REGRESSION LOGISTIQUE ET LA PREDICTION

def sharpe\_ratio(df\_pred\_test,df\_result\_test,diff\_real\_large\_small, name, directory\_path):

#calculer le ratio de sharpe de la difference entre large reconstitué - small reconstitué

returns\_test\_reconstituted\_large\_small = df\_pred\_test['next\_month\_return\_large'] - df\_pred\_test['next\_month\_return\_small']

sharpe\_ratio\_test\_reconstituted\_large\_small = np.mean(returns\_test\_reconstituted\_large\_small ) / np.std(returns\_test\_reconstituted\_large\_small)

#calculer le ratio de sharpe de la différence entre small reconstitué - large reconstitué

returns\_test\_reconstituted\_small\_large = df\_pred\_test['next\_month\_return\_small'] - df\_pred\_test['next\_month\_return\_large']

sharpe\_ratio\_test\_reconstituted\_small\_large = np.mean(returns\_test\_reconstituted\_small\_large ) / np.std(returns\_test\_reconstituted\_small\_large)

#calculer le ratio de sharpe de la stratégie appliqué sur le large-small reconstitué (si prevision = 1) et sur le small-large reconstitué (si prévision = 0)

returns\_test\_reconstituted\_strat = df\_result\_test['result']

sharpe\_ratio\_test\_reconstituted\_strat = np.mean(returns\_test\_reconstituted\_strat) / np.std(returns\_test\_reconstituted\_strat)

#calculer le ratio de sharpe de la stratégie appliqué sur le vrai index LCXR-SCXR (si prévision =1)et sur le SCXR-LCXR (si prévision =0)

returns\_test\_real\_diff\_large\_small = diff\_real\_large\_small

sharpe\_ratio\_test\_real\_diff\_large\_small = np.mean(returns\_test\_real\_diff\_large\_small) / np.std(returns\_test\_real\_diff\_large\_small)

sharpe\_ratios = [

sharpe\_ratio\_test\_reconstituted\_large\_small,

sharpe\_ratio\_test\_reconstituted\_small\_large,

sharpe\_ratio\_test\_reconstituted\_strat,

sharpe\_ratio\_test\_real\_diff\_large\_small

]

labels = [

'Reconstituted Large - Small',

'Reconstituted Small - Large',

'Reconstituted Strategy',

'Strategy on Real Large - Small']

#ploter avec la bonne légende les ratios de sharpe sur un même graphe dans le bon dossier

fig, ax = plt.subplots(figsize=(10, 6))

ax.plot(labels, sharpe\_ratios, 'o')

ax.set\_title('Ratios de Sharpe')

ax.set\_xlabel('')

ax.set\_ylabel('Ratio de Sharpe')

ax.grid(True)

full\_directory\_sharpe = os.path.join(directory\_path,'Sharpe Ratio -'+ name +'.png')

plt.savefig(full\_directory\_sharpe)

plt.close()

#ratio information moyenne des écart de rendemments avec le bench(excess returns) sur l'écart type

#attention ratio d'information mensuel

def IF\_ratio(portfolio\_returns,benchmark\_returns):

excess\_returns = portfolio\_returns - benchmark\_returns

information\_ratio = excess\_returns.mean() / excess\_returns.std()

return information\_ratio

#plot l'information ratio

def information\_ratio(df\_result\_test,df\_pred\_test,diff\_real\_large\_small, name,directory\_path):

#ratio d'information de la stratégie appliquée aux index reconstitués (long large short small ou inversement) avec comme benchmark le large-small reconstitué

test\_reconstituted\_strat\_returns = df\_result\_test['result']

reconstitued\_large\_small\_benchmark\_returns = df\_pred\_test['next\_month\_return\_large'] - df\_pred\_test['next\_month\_return\_small']

information\_ratio\_test\_reconstituted\_strat\_vs\_reconstitued\_large\_small = IF\_ratio(test\_reconstituted\_strat\_returns,reconstitued\_large\_small\_benchmark\_returns)

#ratio d'information de la stratégie appliquée aux index reconstitués (long large short small ou inversement) avec comme benchmark le small-large reconstitué

reconstitued\_small\_large\_benchmark\_returns = df\_pred\_test['next\_month\_return\_small'] - df\_pred\_test['next\_month\_return\_large']

information\_ratio\_test\_reconstituted\_strat\_vs\_reconstitued\_small\_large = IF\_ratio(test\_reconstituted\_strat\_returns,reconstitued\_small\_large\_benchmark\_returns)

#ratio d'information de la stratégié appliqué aux vrais index (long LCXR short SCXR ou inversement) avec comme benchmark le large-small reconstitué

test\_strat\_returns\_real\_large\_small =diff\_real\_large\_small

information\_ratio\_test\_strat\_real\_vs\_reconstitued\_large\_small = IF\_ratio(test\_strat\_returns\_real\_large\_small,reconstitued\_large\_small\_benchmark\_returns)

#ratio d'information de la stratégié appliqué aux vrais index (long LCXR short SCXR ou inversement) avec comme benchmark le small-large reconstitué

information\_ratio\_test\_strat\_real\_vs\_reconstitued\_small\_large = IF\_ratio(test\_strat\_returns\_real\_large\_small,reconstitued\_small\_large\_benchmark\_returns)

x = [1, 2, 3, 4]

y = [information\_ratio\_test\_reconstituted\_strat\_vs\_reconstitued\_large\_small,

information\_ratio\_test\_reconstituted\_strat\_vs\_reconstitued\_small\_large,

information\_ratio\_test\_strat\_real\_vs\_reconstitued\_large\_small,

information\_ratio\_test\_strat\_real\_vs\_reconstitued\_small\_large]

#plot la figure

plt.figure(figsize=(10, 6))

plt.scatter(x, y)

plt.xticks(x, ['Strat Reconst I & b=Reconst L-S',

'Strat Reconst I & b=Reconst S-L',

'Strat Real I & b=Reconst L-S',

'Strat Real I & b=Reconst S-L'])

plt.xlabel('Comparison')

plt.ylabel('Information Ratio')

plt.title('Information Ratios')

plt.grid()

full\_directory\_IF = os.path.join(directory\_path,'information\_ratios'+ name+'.png')

plt.savefig(full\_directory\_IF)

plt.close()

def lasso\_data(datedeb, datefin, working\_folder = working\_folder):

bbg = pybbg.Pybbg()

df= pd.DataFrame()

df = bbg.bdh("LCXR Index", 'PX\_LAST', datedeb, datefin)

df['Date'] = df.index

df = df.reset\_index(drop=True)

liste = []

for i in range(1,len(df.index)) :

if(df.loc[i,'Date'].month != df.loc[i-1,'Date'].month):

liste.append(df.loc[i,'Date'])

liste = [date.strftime('%Y%m%d') for date in liste]

score\_dict = {}

#vérification de l'existence des dossiers

for i in liste :

i\_format\_tirets = f"{i[:4]}-{i[4:6]}-{i[6:]}"

dossier\_deb = working\_folder

dossier\_initial = os.path.join(dossier\_deb, f"members\_{i}")

file\_name\_h5 = f"df\_score\_pe\_{i\_format\_tirets}.h5"

file = os.path.join(dossier\_initial, file\_name\_h5)

if os.path.exists(file):

pass

print("Le fichier score existe à la date : ", i)

else:

print("Le fichier score à la date : ", i," n'existe pas.")

return 0

df\_score, metadata1 = h5load(file)

score\_dict[i] = df\_score

print("L'ensemble des fichiers score vont pouvoir être download")

dates = []

df\_scores = []

for date\_str, df\_score in score\_dict.items():

dates.append(date\_str)

df\_scores.append(df\_score)

dates= dates[1:]

df\_scores = df\_scores[:-1]

print(dates)

print(df\_scores)

#objectif c'est davoir rdt qui contient les rendements du mois précédent (dans nos dataframes initiaux, il contenait les rdt du mois suivant)

rdt = {}

for i, date in enumerate(dates):

num\_df = df\_scores[i]

df\_rdt = pd.DataFrame()

df\_rdt['secteur'] = num\_df['secteur']

df\_rdt['SCXR return'] = num\_df['SCXR next month return']

df\_rdt['MCXR return'] = num\_df['MCXR next month return']

df\_rdt['LCXR return'] = num\_df['LCXR next month return']

rdt[date] = df\_rdt

print(rdt)

return dates, df\_scores, rdt, score\_dict

#fonction qui charge les données du GDP europe et du GDP world en daily et qui renvoie les deux dataframes correspondants

def retrieve\_gdp():

resp\_eu = LocalTerminal.get\_historical('EHGDEUY Index', 'PX\_LAST',start="19001231", period='DAILY')

data\_eu = resp\_eu.as\_frame()

resp\_wrld = LocalTerminal.get\_historical('GDPGAWLD Index', 'PX\_LAST',start="19001231", period='DAILY')

data\_wrld = resp\_wrld.as\_frame()

return data\_eu, data\_wrld

#fonction qui sert à charger les fichiers df\_LCXR (qui contient l'ensemble des prix en daily) et df\_SCXR disponible en .h5 sur data\_index folder

#cela retourne les rendemments à chaque debut de mois du LCXR et SCXR

def retrieve\_index(file,date):

merged\_df1 = pd.DataFrame()

merged\_df1, metadata3 = h5load(file)

merged\_df1.index = pd.to\_datetime(merged\_df1.index).strftime('%Y%m%d')

merged\_df1 = merged\_df1.loc[merged\_df1.index.isin(date)]

merged\_df1 = merged\_df1.pct\_change()

return merged\_df1

def lasso(date1, date2,mois, trinaire, decalage\_interpolation\_gdp=0, working\_folder =working\_folder):

#on modifie la date de fin (date2) qd on souhaite un décalage de l'interpolation

#car sinon nous allons au delà de décembre 2023 et après décembre 2023 nous ne connaisons

#pas le gdp 2024 pour faire l'interpolation

date = datetime.strptime(str(date2), "%Y%m%d")

new\_date = date - timedelta(days=decalage\_interpolation\_gdp\*30)

new\_date\_2 = new\_date.strftime("%Y%m%d")

#prepare les datasets avec la fonction lasso\_date

dates, df\_scores, rdt, score\_dict = lasso\_data(date1, new\_date\_2)

#recupere les données du gdp pour procéder aux interpolations

data\_eu, data\_wrld = retrieve\_gdp()

#on calcule l'ensemble des rendements pour estimer la distribution des rendemments large-small pour trouver le quantile 50%

perf\_largevssmall = []

for date, df in rdt.items():

largevssmall = (1+(df.loc[df['secteur'] == 'Full Index', 'LCXR return'].values[0]/100) ) - (1+(df.loc[df['secteur'] == 'Full Index', 'SCXR return'].values[0]/100))

perf\_largevssmall.append(largevssmall)

print(perf\_largevssmall)

#on définit un seuil à la moitié de la distribution au dessus on attribuera 1 (large qui perf par rapport aux smalls) en dessous 0 (small qui perf par rapport aux larges)

perf\_largevssmall.sort()

n = len(perf\_largevssmall)

seuil = perf\_largevssmall[int((n)/2)]

print(f"Seuil : {seuil}")

rep\_gdp = []

dataframe\_tree = pd.DataFrame()

index = 0

for date, score\_df in tqdm(score\_dict.items()):

#calcul de la diff gdp eur vs gdp world

date\_obj = datetime.strptime(date, '%Y%m%d')

#décalage de decalage\_interpolation\_gdp mois

#ca décale l'interpolation du gdp

date\_obj = date\_obj + timedelta(days=decalage\_interpolation\_gdp\*30)

#on cherche la prochaine année à retrouver dans le dataframe

month = date\_obj.month

year = date\_obj.year

previous\_year= year-1

previous\_date = f"{previous\_year}-12-31"

next\_date = f"{year}-12-31"

#on mets en pourcentage le gdp et on trouve le point du prochain gdp en n+1

previous\_value\_gdp\_eu = data\_eu.loc[previous\_date].iloc[0]/100

previous\_value\_gdp\_wrld = data\_wrld.loc[previous\_date].iloc[0]/100

next\_value\_gdp\_eu = data\_eu.loc[next\_date].iloc[0]/100

next\_value\_gdp\_wrld = data\_wrld.loc[next\_date].iloc[0]/100

#réalisation de l'interpolation

interpolation\_gdp\_eu = previous\_value\_gdp\_eu+(month-1)\*((next\_value\_gdp\_eu-previous\_value\_gdp\_eu)/11)

interpolation\_gdp\_wrld = previous\_value\_gdp\_wrld+(month-1)\*((next\_value\_gdp\_wrld-previous\_value\_gdp\_wrld)/11)

#on exprime la différence de gdp entre europe et world

diff\_gdp = interpolation\_gdp\_eu-interpolation\_gdp\_wrld

rep\_gdp.append(diff\_gdp) #gros dataframe avec toutes les diff de gdp

dataframe\_tree.loc[index,'Date'] = date

rmse = 0

diff\_l= 0

diff\_s = 0

#on parcours l'ensemble des valeurs de pe\_u pour chaque secteur

sectors = ['Utilities','Consumer Discretionary', 'Industrials', 'Energy', 'Information Technology', 'Materials', 'Real Estate', 'Consumer Staples', 'Communication Services', 'Health Care']

for sector in sectors:

value\_peu\_s = score\_df.loc[score\_df['secteur'] == sector, 'Median SCXR pe\_u'].values[0]

value\_peu\_l = score\_df.loc[score\_df['secteur'] == sector, 'Median LCXR pe\_u'].values[0]

value\_pe\_s = score\_df.loc[score\_df['secteur'] == sector, 'Median SCXR pe'].values[0]

value\_pe\_l = score\_df.loc[score\_df['secteur'] == sector, 'Median LCXR pe'].values[0]

if (sector != 'Full Index'):

rmse += (value\_peu\_l - value\_peu\_s)\*\*2 #on calcule le rmse comme étant la somme des écarts au carré des pe\_u large et small pour chaque secteur

#diff\_l += (value\_peu\_l-value\_pe\_l)\*\*2 #on calcule la diff\_l comme étant la somme des écarts au carré des pe\_u et pe des larges pour chaque secteur

#diff\_s += (value\_peu\_s-value\_pe\_s)\*\*2 #on calcule la diff\_l comme étant la somme des écarts au carré des pe\_u et pe des smalls pour chaque secteur

rmse = (rmse)\*\*(0.5)

#rmse\_l = (diff\_l)\*\*(0.5)

#rmse\_s = (diff\_s)\*\*(0.5)

#on rajoute en variable le pe\_u médian des larges de l'ensemble des secteurs (donc full index)

#on rajoute en variable le pe\_u médian des smalls de l'ensemble des secteurs (donc full index)

dataframe\_tree.loc[index, 'Median SCXR pe\_u Full Index'] = score\_df.loc[score\_df['secteur'] == 'Full Index', 'Median SCXR pe\_u'].values[0]

dataframe\_tree.loc[index, 'Median LCXR pe\_u Full Index'] = score\_df.loc[score\_df['secteur'] == 'Full Index', 'Median LCXR pe\_u'].values[0]

dataframe\_tree.loc[index, 'rmse'] = rmse #rmse en variable

#dataframe\_tree.loc[index, 'diff LXCR'] = rmse\_l #si on ne le mets pas en commentaire alors il sera considérer comme variable dans la régression logistique

#dataframe\_tree.loc[index, 'diff SXCR'] = rmse\_s #si on ne le mets pas en commentaire alors il sera considérer comme variable dans la régression logistique

dataframe\_tree.loc[index, 'diff GDP'] = diff\_gdp #diff du gdp en variable

#rajouter le rendement passé (du dernier mois) en variable pr la reg logisitique

if (index!=0):

dff = rdt[dates[index-1]]

dataframe\_tree.loc[index, 'rdt 0'] = (1+(dff.loc[dff['secteur'] == 'Full Index', 'LCXR return'].values[0]/100) ) - (1+(dff.loc[dff['secteur'] == 'Full Index', 'SCXR return'].values[0]/100))

#recuperer le rendement le mois suivant

if((index+mois)> len(dates)-1):

break

else:

df = rdt[dates[index+mois]]

#on stocke les returns du mois suivant pour ensuite pouvoir faire les calculs de perf plus facilement (ils seront enlevés lors de la régression logistiques)

dff1 = rdt[dates[index]]

#rdt du mois suivant uniquement large -small reconstitué

dataframe\_tree.loc[index, 'month\_return'] = (1+(dff1.loc[dff1['secteur'] == 'Full Index', 'LCXR return'].values[0]/100) ) - (1+(dff1.loc[dff1['secteur'] == 'Full Index', 'SCXR return'].values[0]/100))

#rdt du mois suivant uniquement large reconstitué

dataframe\_tree.loc[index, 'next\_month\_return\_large'] = dff1.loc[dff1['secteur'] == 'Full Index', 'LCXR return'].values[0]/100

#rdt du mois suivant uniquement small reconstitué

dataframe\_tree.loc[index, 'next\_month\_return\_small'] = dff1.loc[dff1['secteur'] == 'Full Index', 'SCXR return'].values[0]/100

#on transforme le rdt en score 1 ou 0 (en fonction de sa position dans la distribution historique ) et l'objectif est que le modèle prédise ses valeurs

largevssmall\_score = (1+(df.loc[df['secteur'] == 'Full Index', 'LCXR return'].values[0]/100) ) - (1+(df.loc[df['secteur'] == 'Full Index', 'SCXR return'].values[0]/100))

dataframe\_tree.loc[index,'lvs'] = largevssmall\_score

if largevssmall\_score > seuil: #si supérieur au seuil on est à droite de la distribution donc bon signe => 1

dataframe\_tree.loc[index,'score large\_vs\_small'] = 1

else:

dataframe\_tree.loc[index,'score large\_vs\_small'] = 0

#cas pour rdt0 ajouté à notre dataframe (variables)

if(index==0):

dataframe\_tree = dataframe\_tree.iloc[1:, :]

index +=1

dataframe\_tree = dataframe\_tree.iloc[:-1]

print(dataframe\_tree)

#rendre discrete la variable gdp, si ternaire vaut true je converti le gdp en variable ternaire

#à l'aide des quantiles 0.33 et 0.66

if trinaire :

seuil1 = np.quantile(rep\_gdp, 0.33)

seuil2 = np.quantile(rep\_gdp, 0.66)

for index, row in dataframe\_tree.iterrows():

if row['diff GDP'] < seuil1:

dataframe\_tree.loc[index, 'diff GDP'] = -1

if row['diff GDP'] > seuil2:

dataframe\_tree.loc[index, 'diff GDP'] = 1

else:

dataframe\_tree.loc[index, 'diff GDP'] = 0

#change le dataset de test et de train : le train est la période la plus récente et on essaie de prédire les scores large vs small des années plus vieille (train)

num\_rows = dataframe\_tree.shape[0]

#split\_index = int(num\_rows \* 0.35)

#df\_part1 = dataframe\_tree.iloc[split\_index+mois:]

#df\_part2 = dataframe\_tree.iloc[:split\_index+mois]

#par défaut les années anciennes (2006-2017) prédisent les rendements des années récentes (2017-2023)

split\_index = int(num\_rows \* 0.65)

df\_part1 = dataframe\_tree.iloc[:split\_index+mois]

df\_part2 = dataframe\_tree.iloc[split\_index+mois:]

df\_pred\_train = pd.DataFrame()

df\_pred\_train = df\_part1.copy()

df\_part1\_equi= df\_part1.drop(['lvs', 'Date','month\_return','next\_month\_return\_large','next\_month\_return\_small'], axis=1)

df\_part1 = df\_part1.drop(['lvs', 'Date','month\_return','next\_month\_return\_large','next\_month\_return\_small'], axis=1)

#si on souhaite diviser le dataset de train en supprimant une ligne sur 3 par exemple

#df\_part1 = df\_part1.loc[df\_part1.index %3 != 0, :]

#df\_part1= df\_part1.sample(frac=0.8, random\_state=31)

#L'objectif du code ci dessous est d'avoir un dataset de train équipondéré

#avoir le même nb de 0 et de 1 dans df\_part1

df\_large\_vs\_small\_0 = df\_part1\_equi[df\_part1\_equi['score large\_vs\_small'] == 0]

df\_large\_vs\_small\_1 = df\_part1\_equi[df\_part1\_equi['score large\_vs\_small'] == 1]

num\_rows\_0 = len(df\_large\_vs\_small\_0)

num\_rows\_1 = len(df\_large\_vs\_small\_1)

if num\_rows\_0 > num\_rows\_1:

df\_large\_vs\_small\_0 = df\_large\_vs\_small\_0.iloc[:num\_rows\_1]

else:

df\_large\_vs\_small\_1 = df\_large\_vs\_small\_1.iloc[:num\_rows\_0]

df\_part1\_equi = pd.concat([df\_large\_vs\_small\_0, df\_large\_vs\_small\_1], ignore\_index=True)

#différentes copie de dataset et on garde comme variables pour la régression logistique uniquement

#rmse : écart entre le pe\_u large et pe\_u small de chaque secteur au carré

#Median SCXR pe\_u Full Index : le pe\_u médian sur l'ensemble des secteurs des smalls

#Median LCXR pe\_u Full Index : le pe\_u médian sur l'ensemble des secteurs des larges

#diff GDP : l'interpolation du gdp du mois correspondant (avec ou sans décalage en fonction des paramètres)

#rdt0 : rendement du large-small reconstitué du mois passé

df\_pred\_test = pd.DataFrame()

df\_pred\_test = df\_part2.copy() #sert de copie pour faire les calculs de perf

df\_part2= df\_part2.drop(['lvs', 'Date','month\_return','next\_month\_return\_large','next\_month\_return\_small'], axis=1)

#dataset de train équipondéré (qui sert à fit le modèle)

y = df\_part1\_equi['score large\_vs\_small']

X = df\_part1\_equi.drop('score large\_vs\_small', axis=1)

features\_list1 = list(X.columns)

X = np.array(X)

#dataset de train non équipondéré (en entier mais seulement une partie a entrainer le modèle)

y\_full\_train = df\_part1['score large\_vs\_small']

X\_full\_train = df\_part1.drop('score large\_vs\_small', axis=1)

features\_list1\_full = list(X\_full\_train.columns)

X\_full\_train = np.array(X\_full\_train)

#dataset de test

labels2 = df\_part2['score large\_vs\_small']

features2 = df\_part2.drop('score large\_vs\_small', axis=1)

features\_list2 = list(features2.columns)

features2 = np.array(features2)

#régression logistique avec crossvalidation sur le dataset de train équipondéré

#la cross validation permet de divierser l'ensemble de donnée en k plis de taille égale

#ensuite le modèle est entrainé k fois en utilisant k-1 plis comme ensemble d'apprentissage et le pli restant comme test

#après on a k estimations de la perf du modèle et on peut faire une estimation finale du modèle

class\_weights = {0: 1, 1: 1}

reg = LogisticRegressionCV(cv=20, random\_state=7, penalty='l2', solver='lbfgs', multi\_class='multinomial', class\_weight = class\_weights,max\_iter=1000)

reg.fit(X, y)

y\_pred = reg.predict(features2)

#voir le degré d'importance des variables avec shap

explainer = shap.KernelExplainer(reg.predict\_proba, X)

shap\_values = explainer.shap\_values(features2)

shap.summary\_plot(shap\_values, features2, feature\_names=features\_list1, plot\_type="bar", show=False)

fig = plt.gcf()

fig.savefig('summary\_shap\_plot.png', bbox\_inches='tight')

plt.close()

#précision de la prédiction avec nos données de test uniquement

accuracy = accuracy\_score(labels2, y\_pred)

print(y\_pred)

print(f"Taux de réussite sur l'échantillon de test uniquement : {accuracy:.2f}")

#perf cumulative échantillon de test uniquement

df\_pred\_test['pred'] = y\_pred

df\_pred\_test['pred'] = df\_pred\_test['pred'].replace(0, -1)

df\_result\_test = pd.DataFrame(index=df\_pred\_test.index)

df\_result\_test['result'] = df\_pred\_test['month\_return'] \* df\_pred\_test['pred']

#objectif est d'avoir datum\_test (une liste de date) qui permet de mesurer la performance de notre stratégie

#nos prévisions s'effectuent en T pour la période T+1

#nous rajoutons donc un mois sur les index après notre dernière prédiction pour calculer les performances

datum\_test = df\_pred\_test['Date'].to\_list()

date\_obj1 = datetime.strptime(datum\_test[-1], "%Y%m%d")

date\_next\_month = date\_obj1 + relativedelta(days=35)

date1mois = date\_next\_month.strftime("%Y%m%d")

next\_date = find\_prochaine\_date\_mois(datum\_test[-1], date1mois)

next\_date = next\_date.strftime("%Y%m%d")

datum\_test.append(next\_date)

#charger data LCXR et SCXR, qui correspond aux index réels

lcxr\_latest\_path = find\_latest\_file(working\_folder, 'LCXR')

scxr\_latest\_path = find\_latest\_file(working\_folder, 'SCXR')

LCXR\_index = retrieve\_index(lcxr\_latest\_path, datum\_test)

SCXR\_index = retrieve\_index(scxr\_latest\_path, datum\_test)

LCXR\_index = LCXR\_index['LCXR Index'].squeeze()

SCXR\_index = SCXR\_index['SCXR Index'].squeeze()

#supprime la première ligne qui contient un Nan car on effectue le calcul du rdt avec .pct\_change()

LCXR\_index = LCXR\_index.iloc[1:]

SCXR\_index = SCXR\_index.iloc[1:]

#on mets les mêmes index que df\_result\_test pour simplifier les calculs

diff\_real\_large\_small = (df\_pred\_test['pred']).reset\_index(drop=True) \*(LCXR\_index - SCXR\_index).reset\_index(drop=True)

diff\_real\_large\_small.index = df\_result\_test.index

#sharpe ratio for test only

sharpe\_ratio(df\_pred\_test,df\_result\_test,diff\_real\_large\_small, 'test', working\_folder )

#information ratio for test only

information\_ratio(df\_result\_test,df\_pred\_test,diff\_real\_large\_small,'test', working\_folder )

#cumulatives performances :

#cumulative\_performance\_test\_reconstituted\_strat : la performance cumulative de notre stratégie appliqué sur le large-small reconstitué et small-large reconstitué sur notre dataframe de test uniquement

#cumulative\_performance\_diff\_reconstituted\_large\_small : performance cumulative de l'index reconstitué large - small (si on fait tjrs long large et short small pendant toute la durée de notre dataframe de test)

#cumulative\_performance\_diff\_reconstituted\_small\_large : performance cumulative de l'index reconstitué small - large (si on fait tjrs long small et short large pendant toute la durée de notre dataframe de test)

#cumulative\_performance\_test\_real\_diff\_large\_small : la performance cumulative de notre stratégie appliqué sur le large-small réel (LCXR et SCXR sur bloom) et small-large réel sur notre dataframe de test uniquement

cumulative\_performance\_test = pd.DataFrame({

'cumulative\_performance\_test\_reconstituted\_strat': (1 + df\_result\_test['result']).cumprod(),

'cumulative\_performance\_diff\_reconstituted\_large\_small': (1 + (df\_pred\_test['next\_month\_return\_large'] - df\_pred\_test['next\_month\_return\_small'])).cumprod(),

'cumulative\_performance\_diff\_reconstituted\_small\_large': (1 + (df\_pred\_test['next\_month\_return\_small'] - df\_pred\_test['next\_month\_return\_large'])).cumprod(),

'cumulative\_performance\_test\_real\_diff\_large\_small' : (1+diff\_real\_large\_small).cumprod()})

#on change les index car les prévision sont en T pour la période T+1 donc on doit décaler d'un mois le debut et la fin des dates pour voir les performances de la stratégie

cumulative\_performance\_test = cumulative\_performance\_test.reset\_index(drop=True)

cumulative\_performance\_test.index = datum\_test[1:]

#LONG RECONSTITUED LARGE (if pred =1) or LONG reconstitued small (if pred = -1)

df\_pred\_test.loc[df\_pred\_test['pred'] == 1, 'long\_large\_or\_small'] = df\_pred\_test['next\_month\_return\_large']

df\_pred\_test.loc[df\_pred\_test['pred'] == -1, 'long\_large\_or\_small'] = df\_pred\_test['next\_month\_return\_small']

#LONG le rendement le plus élévé à chaque date entre large reconstitué et small reconstitué

df\_pred\_test['max\_large\_or\_small'] = df\_pred\_test[['next\_month\_return\_large', 'next\_month\_return\_small']].max(axis=1)

#50% long le large reconstitué et 50% long le small reconstitué pendant l'ensemble du dataframe

df\_pred\_test['50\_50\_large\_small'] = df\_pred\_test['next\_month\_return\_large']\*0.5 + 0.5\*df\_pred\_test['next\_month\_return\_small']

#cumulative\_performance\_test\_reconstituted\_large : la performance cumulative de l'index du large reconstitué (long large only) sur le dataframe de test

#cumulative\_performance\_test\_reconstituted\_small : perfomance cumulative de l'index du small reconstitué (long small only) sur le dataframe de test

#cumulative\_performance\_long\_large\_or\_small\_reconstituted : on applique la stratégie de cette facon : si la prévision c'est 1 : long large reconstitué sinon (prévision = 0) long small reconstitué

#cumulative\_performance\_max\_large\_or\_small\_reconstituted : on long chaque mois le rendement le plus élévé entre le large reconstitué et le small reconstitué

#cumulative\_performance\_50\_50\_large\_small\_reconstituted : on fait un 50% long le large reconstitué et 50% long le small reconstitué

cumulative\_performance\_test\_long = pd.DataFrame({

'cumulative\_performance\_test\_reconstituted\_large': (1 + df\_pred\_test['next\_month\_return\_large']).cumprod(),

'cumulative\_performance\_test\_reconstituted\_small': (1 + df\_pred\_test['next\_month\_return\_small']).cumprod(),

'cumulative\_performance\_long\_large\_or\_small\_reconstituted': (1 +df\_pred\_test['long\_large\_or\_small']).cumprod(),

'cumulative\_performance\_max\_large\_or\_small\_reconstituted': (1 + df\_pred\_test['max\_large\_or\_small']).cumprod(),

'cumulative\_performance\_50\_50\_large\_small\_reconstituted': (1 + df\_pred\_test['50\_50\_large\_small']).cumprod(),

})

#décale l'index de 1 car on prédit à T pour T+1

cumulative\_performance\_test\_long = cumulative\_performance\_test\_long.reset\_index(drop=True)

cumulative\_performance\_test\_long.index = datum\_test[1:]

#affichage du premier dataframe cumulative\_performance\_test

plt.figure(figsize=(12, 6))

cumulative\_performance\_test.plot()

plt.title('Performance Cumulative Small VERSUS Large - Dataset Test')

plt.xlabel('Date')

plt.ylabel('Cumulative Return')

plt.legend(['Stratégie sur Index reconstitué', 'Long Index reconstitué Large-Small', 'Long Index reconstitué Small-Large', 'Stratégie sur Index réel'])

file\_path = os.path.join(working\_folder, 'Performance Cumulative Stratégie diff SVL -Test.png')

plt.savefig(file\_path)

plt.close()

#affichage du second dataframe cumulative\_performance\_test\_long

plt.figure(figsize=(12, 6))

cumulative\_performance\_test\_long.plot()

plt.title('Performance Cumulative LONG Small VERSUS Large- Dataset Test')

plt.xlabel('Date')

plt.ylabel('Cumulative Return')

plt.legend(['Index reconstitué Large', 'Index reconstitué Small', 'Long Large or Long Small with predictions', 'Long max between Large et Long return','50% large et 50% small reconstitué'])

file\_path = os.path.join(working\_folder, 'Performance Cumulative LONG SVL - Test.png')

plt.savefig(file\_path)

plt.close()

#calcul precision échantillon train (donc équipondéré)

y\_pred\_train\_equi = reg.predict(X)

accuracy\_train\_equi = accuracy\_score(y, y\_pred\_train\_equi)

print(f"Taux de réussite sur l'échantillon de train équipondéré uniquement : {accuracy\_train\_equi:.2f}")

#calcul precision échantillon train

#perf cumulative échantillon train (l'ensemble du dataset pas seulement le dataset equi) + test

y\_pred\_train\_full = reg.predict(X\_full\_train)

accuracy\_train\_full = accuracy\_score(y\_full\_train, y\_pred\_train\_full)

print(f"Taux de réussite sur l'échantillon de train full (modèle entrainé sur l'échantillon train équipondéré) uniquement : {accuracy\_train\_full:.2f}")

#on reconstruit les prédictions sur l'ensemble du dataframe (train + test)

df\_pred\_train['pred'] = y\_pred\_train\_full

df\_pred\_train['pred'] = df\_pred\_train['pred'].replace(0, -1)

df\_pred = pd.concat([df\_pred\_train, df\_pred\_test], ignore\_index=True)

df\_result = pd.DataFrame(index=df\_pred.index)

#on calcule la performance de la stratégie sur l'ensemble du dataframe

df\_result['result'] = df\_pred['month\_return'] \* df\_pred['pred']

#objectif est d'avoir datum\_full (une liste de date) qui permet de mesurer la performance de notre stratégie

#nos prévisions s'effectuent en T pour la période T+1

#nous rajoutons donc un mois sur les index après notre dernière prédiction pour calculer les performances

datum\_full = df\_pred['Date'].to\_list()

date\_obj1 = datetime.strptime(datum\_full[-1], "%Y%m%d")

date\_next\_month = date\_obj1 + relativedelta(days=35)

date1mois = date\_next\_month.strftime("%Y%m%d")

next\_date = find\_prochaine\_date\_mois(datum\_full[-1], date1mois)

next\_date = next\_date.strftime("%Y%m%d")

datum\_full.append(next\_date)

#charger data LCXR et SCXR, qui correspond aux index réels

LCXR\_index = retrieve\_index(lcxr\_latest\_path, datum\_full)

SCXR\_index = retrieve\_index(scxr\_latest\_path, datum\_full)

LCXR\_index = LCXR\_index['LCXR Index'].squeeze()

SCXR\_index = SCXR\_index['SCXR Index'].squeeze()

LCXR\_index = LCXR\_index.iloc[1:]

SCXR\_index = SCXR\_index.iloc[1:]

#on mets les mêmes index que df\_result pour simplifier les calculs

diff\_real\_large\_small = (df\_pred['pred']).reset\_index(drop=True) \*(LCXR\_index - SCXR\_index).reset\_index(drop=True)

diff\_real\_large\_small.index = df\_pred['pred'].index

#sharpe ratio et information ratio for test+ train only

sharpe\_ratio(df\_pred,df\_result,diff\_real\_large\_small, 'Test + Train',working\_folder)

information\_ratio(df\_result,df\_pred,diff\_real\_large\_small,'Test + Train',working\_folder)

#cumulatives performances :

#cumulative\_performance\_full\_reconstituted\_strat : la performance cumulative de notre stratégie appliqué sur le large-small reconstitué et small-large reconstitué sur notre dataframe de train et test

#cumulative\_performance\_full\_reconstituted\_diff\_large\_small : performance cumulative de l'index reconstitué large - small (si on fait tjrs long large et short small pendant toute la durée de notre dataframe )

#cumulative\_performance\_full\_diff\_reconstituted\_small\_large : performance cumulative de l'index reconstitué small - large (si on fait tjrs long small et short large pendant toute la durée de notre dataframe)

#cumulative\_performance\_full\_real\_diff\_large\_small : la performance cumulative de notre stratégie appliqué sur le large-small réel (LCXR et SCXR sur bloom) et small-large réel sur notre dataframe

cumulative\_performance\_full = pd.DataFrame({

'cumulative\_performance\_full\_reconstituted\_strat': (1 + df\_result['result']).cumprod(),

'cumulative\_performance\_full\_reconstituted\_diff\_large\_small': (1 + (df\_pred['next\_month\_return\_large'] - df\_pred['next\_month\_return\_small'])).cumprod(),

'cumulative\_performance\_full\_diff\_reconstituted\_small\_large': (1 + (df\_pred['next\_month\_return\_small'] - df\_pred['next\_month\_return\_large'])).cumprod(),

'cumulative\_performance\_full\_real\_diff\_large\_small' : (1+diff\_real\_large\_small).cumprod()

})

#on change les index car les prévision sont en T pour la période T+1 donc on doit décaler d'un mois le debut et la fin des dates pour voir les performances de la stratégie

cumulative\_performance\_full = cumulative\_performance\_full.reset\_index(drop=True)

cumulative\_performance\_full.index = datum\_full[1:]

#LONG RECONSTITUED LARGE (if pred =1) or LONG reconstitued small (if pred = -1)

df\_pred.loc[df\_pred['pred'] == 1, 'long\_large\_or\_small'] = df\_pred['next\_month\_return\_large']

df\_pred.loc[df\_pred['pred'] == -1, 'long\_large\_or\_small'] = df\_pred['next\_month\_return\_small']

#LONG le rendement le plus élévé à chaque date entre large reconstitué et small reconstitué

df\_pred['max\_large\_or\_small'] = df\_pred[['next\_month\_return\_large', 'next\_month\_return\_small']].max(axis=1)

#50% long le large reconstitué et 50% long le small reconstitué pendant l'ensemble du dataframe

df\_pred['50\_50\_large\_small'] = df\_pred['next\_month\_return\_large']\*0.5 + 0.5\*df\_pred['next\_month\_return\_small']

#cumulative\_performance\_full\_reconstituted\_large : la performance cumulative de l'index du large reconstitué (long large only) sur le dataframe de test et train

#cumulative\_performance\_full\_reconstituted\_small : perfomance cumulative de l'index du small reconstitué (long small only) sur le dataframe de test et train

#cumulative\_performance\_long\_large\_or\_small\_reconstituted : on applique la stratégie de cette facon : si la prévision c'est 1 : long large reconstitué sinon (prévision = 0) long small reconstitué

#cumulative\_performance\_max\_large\_or\_small\_reconstituted : on long chaque mois le rendement le plus élévé entre le large reconstitué et le small reconstitué

#cumulative\_performance\_50\_50\_large\_small\_reconstituted : on fait un 50% long le large reconstitué et 50% long le small reconstitué

cumulative\_performance\_full\_long = pd.DataFrame({

'cumulative\_performance\_full\_reconstituted\_large': (1 + df\_pred['next\_month\_return\_large']).cumprod(),

'cumulative\_performance\_full\_reconstituted\_small': (1 + df\_pred['next\_month\_return\_small']).cumprod(),

'cumulative\_performance\_long\_large\_or\_small\_reconstituted': (1 + df\_pred['long\_large\_or\_small']).cumprod(),

'cumulative\_performance\_max\_large\_or\_small\_reconstituted': (1 + df\_pred['max\_large\_or\_small']).cumprod(),

'cumulative\_performance\_50\_50\_large\_small\_reconstituted': (1 + df\_pred['50\_50\_large\_small']).cumprod(),

})

#décale l'index de 1 car on prédit à T pour T+1

cumulative\_performance\_full\_long = cumulative\_performance\_full\_long.reset\_index(drop=True)

cumulative\_performance\_full\_long.index = datum\_full[1:]

#plot le dataframe cumulative\_performance\_full

plt.figure(figsize=(12, 6))

cumulative\_performance\_full.plot()

plt.title('Performance Cumulative Small VERSUS Large- Dataset Train et Test')

plt.xlabel('Date')

plt.ylabel('Cumulative Return')

plt.legend(['Stratégie sur Index reconstitué', 'Long Index reconstitué Large-Small', 'Long Index reconstitué Small-Large', 'Stratégie sur Index réel'])

file\_path = os.path.join(working\_folder,'Performance Cumulative Stratégie diff SVL - Train et Test.png')

plt.savefig(file\_path)

plt.close()

#plot le dataframe cumulative\_performance\_full\_long

plt.figure(figsize=(12, 6))

cumulative\_performance\_full\_long.plot()

plt.title('Performance Cumulative LONG Small VERSUS Large- Dataset Train et Test')

plt.xlabel('Date')

plt.ylabel('Cumulative Return')

plt.legend(['Index reconstitué Large', 'Index reconstitué Small', 'Long Large or Long Small with predictions', 'Long max between Large et Long return','50% large et 50% small reconstitué'])

file\_path = os.path.join(working\_folder,'Performance Cumulative LONG SVL - Train et Test.png')

plt.savefig(file\_path)

plt.close()

return reg, seuil

##fonction afin de réaliser des simulaires sur des données sans connaisance du réel gdp

def simulation\_utilisation\_modele(date1, date2, mois,reg,seuil1, trinaire,decalage\_interpolation\_gdp=0, rep\_gdp=0, data\_eu=0, data\_wrld=0, working\_folder = working\_folder):

#prepare les datasets avec la fonction lasso\_date

dates, df\_scores, rdt, score\_dict = lasso\_data(date1, date2)

dataframe\_tree = pd.DataFrame()

index = 0

if trinaire == False :

rep\_gdp = []

for date, score\_df in score\_dict.items():

if trinaire == True:

if len(rep\_gdp)!=len(dates)+1:

raise PrevisionError(f"Erreur: il faut spécifier {len(dates)+1} prévisions dans rep.")

if trinaire == False :

date\_obj = datetime.strptime(date, '%Y%m%d')

#décalage de decalage\_interpolation\_gdp mois

#ca décale l'interpolation du gdp

date\_obj = date\_obj + timedelta(days=decalage\_interpolation\_gdp\*30)

#on cherche la prochaine année à retrouver dans le dataframe

month = date\_obj.month

year = date\_obj.year

previous\_year= year-1

previous\_date = f"{previous\_year}-12-31"

next\_date = f"{year}-12-31"

previous\_value\_gdp\_eu = data\_eu.loc[previous\_date].iloc[0]/100

previous\_value\_gdp\_wrld = data\_wrld.loc[previous\_date].iloc[0]/100

next\_value\_gdp\_eu = data\_eu.loc[next\_date].iloc[0]/100

next\_value\_gdp\_wrld = data\_wrld.loc[next\_date].iloc[0]/100

interpolation\_gdp\_eu = previous\_value\_gdp\_eu+(month-1)\*((next\_value\_gdp\_eu-previous\_value\_gdp\_eu)/11)

interpolation\_gdp\_wrld = previous\_value\_gdp\_wrld+(month-1)\*((next\_value\_gdp\_wrld-previous\_value\_gdp\_wrld)/11)

#on exprime la différence de gdp entre europe et world

diff\_gdp = interpolation\_gdp\_eu-interpolation\_gdp\_wrld

rep\_gdp.append(diff\_gdp) #gros dataframe avec toutes les diff de gdp

dataframe\_tree.loc[index,'Date'] = date

rmse = 0

diff\_l= 0

diff\_s = 0

#on parcours l'ensemble des valeurs de pe\_u pour chaque secteur

sectors = ['Utilities','Consumer Discretionary', 'Industrials', 'Energy', 'Information Technology', 'Materials', 'Real Estate', 'Consumer Staples', 'Communication Services', 'Health Care']

for sector in sectors:

value\_peu\_s = score\_df.loc[score\_df['secteur'] == sector, 'Median SCXR pe\_u'].values[0]

value\_peu\_l = score\_df.loc[score\_df['secteur'] == sector, 'Median LCXR pe\_u'].values[0]

value\_pe\_s = score\_df.loc[score\_df['secteur'] == sector, 'Median SCXR pe'].values[0]

value\_pe\_l = score\_df.loc[score\_df['secteur'] == sector, 'Median LCXR pe'].values[0]

if (sector != 'Full Index'):

rmse += (value\_peu\_l - value\_peu\_s)\*\*2 #on calcule le rmse comme étant la somme des écarts au carré des pe\_u large et small pour chaque secteur

#diff\_l += (value\_peu\_l-value\_pe\_l)\*\*2 #on calcule la diff\_l comme étant la somme des écarts au carré des pe\_u et pe des larges pour chaque secteur

#diff\_s += (value\_peu\_s-value\_pe\_s)\*\*2 #on calcule la diff\_l comme étant la somme des écarts au carré des pe\_u et pe des smalls pour chaque secteur

rmse = (rmse)\*\*(0.5)

#rmse\_l = (diff\_l)\*\*(0.5)

#rmse\_s = (diff\_s)\*\*(0.5)

#on rajoute en variable le pe\_u médian des larges de l'ensemble des secteurs (donc full index)

#on rajoute en variable le pe\_u médian des smalls de l'ensemble des secteurs (donc full index)

dataframe\_tree.loc[index, 'Median SCXR pe\_u Full Index'] = score\_df.loc[score\_df['secteur'] == 'Full Index', 'Median SCXR pe\_u'].values[0]

dataframe\_tree.loc[index, 'Median LCXR pe\_u Full Index'] = score\_df.loc[score\_df['secteur'] == 'Full Index', 'Median LCXR pe\_u'].values[0]

dataframe\_tree.loc[index, 'rmse'] = rmse #rmse en variable

#dataframe\_tree.loc[index, 'diff LXCR'] = rmse\_l #si on ne le mets pas en commentaire alors il sera considérer comme variable dans la régression logistique

#dataframe\_tree.loc[index, 'diff SXCR'] = rmse\_s #si on ne le mets pas en commentaire alors il sera considérer comme variable dans la régression logistique

dataframe\_tree.loc[index, 'diff GDP'] = rep\_gdp[index] #diff du gdp en variable

#rajouter le rendement passé (du dernier mois) en variable pr la reg logisitique

if (index!=0):

dff = rdt[dates[index-1]]

dataframe\_tree.loc[index, 'rdt 0'] = (1+(dff.loc[dff['secteur'] == 'Full Index', 'LCXR return'].values[0]/100) ) - (1+(dff.loc[dff['secteur'] == 'Full Index', 'SCXR return'].values[0]/100))

#recuperer le rendement le mois suivant

if((index+mois)<= len(dates)-1):

shape = rdt[dates[index+mois]].shape

df = rdt[dates[index+mois]]

#on stocke les returns du mois suivant pour ensuite pouvoir faire les calculs de perf plus facilement (ils seront enlevés lors de la régression logistiques)

dff1 = rdt[dates[index]]

#rdt du mois suivant uniquement large -small reconstitué

dataframe\_tree.loc[index, 'month\_return'] = (1+(dff1.loc[dff1['secteur'] == 'Full Index', 'LCXR return'].values[0]/100) ) - (1+(dff1.loc[dff1['secteur'] == 'Full Index', 'SCXR return'].values[0]/100))

#rdt du mois suivant uniquement large reconstitué

dataframe\_tree.loc[index, 'next\_month\_return\_large'] = dff1.loc[dff1['secteur'] == 'Full Index', 'LCXR return'].values[0]/100

#rdt du mois suivant uniquement small reconstitué

dataframe\_tree.loc[index, 'next\_month\_return\_small'] = dff1.loc[dff1['secteur'] == 'Full Index', 'SCXR return'].values[0]/100

#on transforme le rdt en score 1 ou 0 (en fonction de sa position dans la distribution historique ) et l'objectif est que le modèle prédise ses valeurs

largevssmall\_score = (1+(df.loc[df['secteur'] == 'Full Index', 'LCXR return'].values[0]/100) ) - (1+(df.loc[df['secteur'] == 'Full Index', 'SCXR return'].values[0]/100))

dataframe\_tree.loc[index,'lvs'] = largevssmall\_score

if largevssmall\_score > seuil1: #si supérieur au seuil on est à droite de la distribution donc bon signe => 1

dataframe\_tree.loc[index,'score large\_vs\_small'] = 1

else:

dataframe\_tree.loc[index,'score large\_vs\_small'] = 0

else :

df = np.nan

dff1 = pd.DataFrame(np.zeros(shape))

#cas pour rdt0 ajouté à notre dataframe (variables)

if(index==0):

dataframe\_tree = dataframe\_tree.iloc[1:, :]

index +=1

print(dataframe\_tree)

if trinaire :

name\_folder = "Test-sans-connaissance-GDP\_Trinaire\_"+ str(date1)+ "\_" + str(date2)

else :

name\_folder = "Test-sans-connaissance-GDP\_Interpolation\_"+ str(date1)+ "\_" + str(date2)

directory\_path = working\_folder

full\_directory\_path = os.path.join(directory\_path, name\_folder)

create\_or\_replace\_directory(full\_directory\_path)

#change le dataset de test et de train : le train est la période la plus récente et on essaie de prédire les scores large vs small des années plus vieille (train)

num\_rows = dataframe\_tree.shape[0]

#split\_index = int(num\_rows \* 0.35)

#df\_part1 = dataframe\_tree.iloc[split\_index+mois:]

df\_part2 = dataframe\_tree.copy()

#différentes copie de dataset et on garde comme variables pour la régression logistique uniquement

#rmse : écart entre le pe\_u large et pe\_u small de chaque secteur au carré

#Median SCXR pe\_u Full Index : le pe\_u médian sur l'ensemble des secteurs des smalls

#Median LCXR pe\_u Full Index : le pe\_u médian sur l'ensemble des secteurs des larges

#diff GDP : l'interpolation du gdp du mois correspondant (avec ou sans décalage en fonction des paramètres)

#rdt0 : rendement du large-small reconstitué du mois passé

df\_pred\_test = pd.DataFrame()

df\_pred\_test = dataframe\_tree.copy() #sert de copie pour faire les calculs de perf

df\_part2= df\_part2.drop(['lvs', 'Date','month\_return','next\_month\_return\_large','next\_month\_return\_small'], axis=1)

#dataset de test

labels2 = df\_part2['score large\_vs\_small']

features2 = df\_part2.drop('score large\_vs\_small', axis=1)

features\_list2 = list(features2.columns)

features2 = np.array(features2)

#régression logistique avec crossvalidation sur le dataset de train équipondéré

#la cross validation permet de divierser l'ensemble de donnée en k plis de taille égale

#ensuite le modèle est entrainé k fois en utilisant k-1 plis comme ensemble d'apprentissage et le pli restant comme test

#après on a k estimations de la perf du modèle et on peut faire une estimation finale du modèle

y\_pred = reg.predict(features2)

#précision de la prédiction avec nos données de test uniquement

if np.isnan(labels2[len(labels2)]) :

labels2 = labels2.iloc[:-1]

last\_prediction = y\_pred[-1]

y\_pred = y\_pred[:-1]

print("La dernière valeur prédite pour la période ", dates[-1], " est ", last\_prediction)

print("Rappel 1 = Large superforme Small et 0 = Small superforme Large")

df\_pred\_test = df\_pred\_test.iloc[:-1]

accuracy = accuracy\_score(labels2, y\_pred)

print(y\_pred)

print(f"Taux de réussite sur l'échantillon de test uniquement : {accuracy:.2f}")

#perf cumulative échantillon de test uniquement

df\_pred\_test['pred'] = y\_pred

df\_pred\_test['pred'] = df\_pred\_test['pred'].replace(0, -1)

df\_result\_test = pd.DataFrame(index=df\_pred\_test.index)

df\_result\_test['result'] = df\_pred\_test['month\_return'] \* df\_pred\_test['pred']

#objectif est d'avoir datum\_test (une liste de date) qui permet de mesurer la performance de notre stratégie

#nos prévisions s'effectuent en T pour la période T+1

#nous rajoutons donc un mois sur les index après notre dernière prédiction pour calculer les performances

datum\_test = df\_pred\_test['Date'].to\_list()

date\_obj1 = datetime.strptime(datum\_test[-1], "%Y%m%d")

date\_next\_month = date\_obj1 + relativedelta(days=35)

date1mois = date\_next\_month.strftime("%Y%m%d")

next\_date = find\_prochaine\_date\_mois(datum\_test[-1], date1mois)

next\_date = next\_date.strftime("%Y%m%d")

datum\_test.append(next\_date)

#charger data LCXR et SCXR, qui correspond aux index réels

lcxr\_latest\_path = find\_latest\_file(working\_folder, 'LCXR')

scxr\_latest\_path = find\_latest\_file(working\_folder, 'SCXR')

LCXR\_index = retrieve\_index(lcxr\_latest\_path, datum\_test)

SCXR\_index = retrieve\_index(scxr\_latest\_path, datum\_test)

LCXR\_index = LCXR\_index['LCXR Index'].squeeze()

SCXR\_index = SCXR\_index['SCXR Index'].squeeze()

#supprime la première ligne qui contient un Nan car on effectue le calcul du rdt avec .pct\_change()

LCXR\_index = LCXR\_index.iloc[1:]

SCXR\_index = SCXR\_index.iloc[1:]

#on mets les mêmes index que df\_result\_test pour simplifier les calculs

diff\_real\_large\_small = (df\_pred\_test['pred']).reset\_index(drop=True) \*(LCXR\_index - SCXR\_index).reset\_index(drop=True)

diff\_real\_large\_small.index = df\_result\_test.index

#sharpe ratio for test only

sharpe\_ratio(df\_pred\_test,df\_result\_test,diff\_real\_large\_small, 'test', full\_directory\_path)

#information ratio for test only

information\_ratio(df\_result\_test,df\_pred\_test,diff\_real\_large\_small,'test',full\_directory\_path )

#cumulatives performances :

#cumulative\_performance\_test\_reconstituted\_strat : la performance cumulative de notre stratégie appliqué sur le large-small reconstitué et small-large reconstitué sur notre dataframe de test uniquement

#cumulative\_performance\_diff\_reconstituted\_large\_small : performance cumulative de l'index reconstitué large - small (si on fait tjrs long large et short small pendant toute la durée de notre dataframe de test)

#cumulative\_performance\_diff\_reconstituted\_small\_large : performance cumulative de l'index reconstitué small - large (si on fait tjrs long small et short large pendant toute la durée de notre dataframe de test)

#cumulative\_performance\_test\_real\_diff\_large\_small : la performance cumulative de notre stratégie appliqué sur le large-small réel (LCXR et SCXR sur bloom) et small-large réel sur notre dataframe de test uniquement

cumulative\_performance\_test = pd.DataFrame({

'cumulative\_performance\_test\_reconstituted\_strat': (1 + df\_result\_test['result']).cumprod(),

'cumulative\_performance\_diff\_reconstituted\_large\_small': (1 + (df\_pred\_test['next\_month\_return\_large'] - df\_pred\_test['next\_month\_return\_small'])).cumprod(),

'cumulative\_performance\_diff\_reconstituted\_small\_large': (1 + (df\_pred\_test['next\_month\_return\_small'] - df\_pred\_test['next\_month\_return\_large'])).cumprod(),

'cumulative\_performance\_test\_real\_diff\_large\_small' : (1+diff\_real\_large\_small).cumprod()})

#on change les index car les prévision sont en T pour la période T+1 donc on doit décaler d'un mois le debut et la fin des dates pour voir les performances de la stratégie

cumulative\_performance\_test = cumulative\_performance\_test.reset\_index(drop=True)

cumulative\_performance\_test.index = datum\_test[1:]

#LONG RECONSTITUED LARGE (if pred =1) or LONG reconstitued small (if pred = -1)

df\_pred\_test.loc[df\_pred\_test['pred'] == 1, 'long\_large\_or\_small'] = df\_pred\_test['next\_month\_return\_large']

df\_pred\_test.loc[df\_pred\_test['pred'] == -1, 'long\_large\_or\_small'] = df\_pred\_test['next\_month\_return\_small']

#LONG le rendement le plus élévé à chaque date entre large reconstitué et small reconstitué

df\_pred\_test['max\_large\_or\_small'] = df\_pred\_test[['next\_month\_return\_large', 'next\_month\_return\_small']].max(axis=1)

#50% long le large reconstitué et 50% long le small reconstitué pendant l'ensemble du dataframe

df\_pred\_test['50\_50\_large\_small'] = df\_pred\_test['next\_month\_return\_large']\*0.5 + 0.5\*df\_pred\_test['next\_month\_return\_small']

#cumulative\_performance\_test\_reconstituted\_large : la performance cumulative de l'index du large reconstitué (long large only) sur le dataframe de test

#cumulative\_performance\_test\_reconstituted\_small : perfomance cumulative de l'index du small reconstitué (long small only) sur le dataframe de test

#cumulative\_performance\_long\_large\_or\_small\_reconstituted : on applique la stratégie de cette facon : si la prévision c'est 1 : long large reconstitué sinon (prévision = 0) long small reconstitué

#cumulative\_performance\_max\_large\_or\_small\_reconstituted : on long chaque mois le rendement le plus élévé entre le large reconstitué et le small reconstitué

#cumulative\_performance\_50\_50\_large\_small\_reconstituted : on fait un 50% long le large reconstitué et 50% long le small reconstitué

cumulative\_performance\_test\_long = pd.DataFrame({

'cumulative\_performance\_test\_reconstituted\_large': (1 + df\_pred\_test['next\_month\_return\_large']).cumprod(),

'cumulative\_performance\_test\_reconstituted\_small': (1 + df\_pred\_test['next\_month\_return\_small']).cumprod(),

'cumulative\_performance\_long\_large\_or\_small\_reconstituted': (1 +df\_pred\_test['long\_large\_or\_small']).cumprod(),

'cumulative\_performance\_max\_large\_or\_small\_reconstituted': (1 + df\_pred\_test['max\_large\_or\_small']).cumprod(),

'cumulative\_performance\_50\_50\_large\_small\_reconstituted': (1 + df\_pred\_test['50\_50\_large\_small']).cumprod(),

})

#décale l'index de 1 car on prédit à T pour T+1

cumulative\_performance\_test\_long = cumulative\_performance\_test\_long.reset\_index(drop=True)

cumulative\_performance\_test\_long.index = datum\_test[1:]

#affichage du premier dataframe cumulative\_performance\_test

plt.figure(figsize=(12, 6))

cumulative\_performance\_test.plot()

plt.title('Performance Cumulative Small VERSUS Large')

plt.xlabel('Date')

plt.ylabel('Cumulative Return')

plt.legend(['Stratégie sur Index reconstitué', 'Long Index reconstitué Large-Small', 'Long Index reconstitué Small-Large', 'Stratégie sur Index réel'])

full\_directory\_path\_cum = os.path.join(full\_directory\_path,"Performance Cumulative Stratégie diff SVL -TEST sur periode sans connaisance GDP.png")

plt.savefig(full\_directory\_path\_cum)

plt.close()

#affichage du second dataframe cumulative\_performance\_test\_long

plt.figure(figsize=(12, 6))

cumulative\_performance\_test\_long.plot()

plt.title('Performance Cumulative LONG Small VERSUS Large')

plt.xlabel('Date')

plt.ylabel('Cumulative Return')

plt.legend(['Index reconstitué Large', 'Index reconstitué Small', 'Long Large or Long Small with predictions', 'Long max between Large et Long return','50% large et 50% small reconstitué'])

full\_directory\_path\_cum2 = os.path.join(full\_directory\_path,"Performance Cumulative LONG SVL - TEST sur periode sans connaisance GDP.png")

plt.savefig(full\_directory\_path\_cum2)

plt.close()

#lasso(date1, date2,mois, trinaire, decalage\_interpolation\_gdp)

#date1 correspond à la date de début pour la réalisation du dataframe

#date2 correspond à la date de fin pour la réalisation du dataframe

#mois : on réalise le calcul des métriques(rmse (sur les pe\_u),le dernier rdt des 20 derniers jours, une interpolation du gdp...) à la date t et nous souhaitons prédire les rendements à la date t+mois

# /!\ par défaut mois = 0 ca veut dire on souhaite prédire les rendement en t+1mois

#trinaire : si trinaire = false alors on utilise l'interpolation du gdp (connaisance du futur) si trinaire = true alors on transforme l'information de la différence de gdp en 1, 0 ou -1 en fonction des quantiles 0.33 et 0.66

#decalage\_interpolation\_gdp : est un nombre de mois par defaut vaut 0 ca correspond au décalage de l'interpolation à effectuer sur le gdp

# par exemple si on met decalage\_interpolation\_gdp = 2 alors la valeur du gdp au mois de mai 2023 correspondra à l'interpolation entre le gdp 2022 et gdp 2023 mais du mois de juillet

#A modifier

#en trinaire

reg, seuil= lasso(20060401, 20231231, 0, True, 0)

rep = [1,1,1,1,1,1,1,1,-1,1,1,0,1,1,1]

simulation\_utilisation\_modele(20230505,20240810,0,reg,seuil,True,0,rep, 0,0)

#def simulation\_utilisation\_modele(date1, date2, mois,reg,seuil, trinaire,decalage\_interpolation\_gdp=0, rep\_gdp=0, data\_eu=0, data\_wrld=0):

#def lasso(date1, date2,mois, trinaire, decalage\_interpolation\_gdp=0):

#en non trinaire

reg, seuil= lasso(20060401, 20231231, 0, False, 0)

data\_eu, data\_wrld = retrieve\_gdp()

print(data\_eu)

print(data\_wrld)

data\_eu.loc[pd.to\_datetime('2024-12-31')] = float(0.7)

data\_wrld.loc[pd.to\_datetime('2024-12-31')] = float(3.10)

simulation\_utilisation\_modele(20240101,20240810,0,reg,seuil,False,0,0, data\_eu, data\_wrld)

download data scxr et lcxr

from tia.bbg import LocalTerminal

import pandas as pd

import datetime

from xbbg import blp

from blp import blp as bld

import pybbg as pybbg

import matplotlib.pyplot as plt

import os

def h5store(filename, df, dic):

store = pd.HDFStore(filename)

store.put('mydata', df)

store.get\_storer('mydata').attrs.metadata = dic

store.close()

working\_folder = r'U:\GDA\PFC\03\_Gerants\03\_12\_NB\Small VS Large'

def retrieve\_index(index1, index2,date\_1, date\_2, dossier\_initial =working\_folder):

data\_index\_folder = os.path.join(working\_folder, 'data\_index')

if not os.path.exists(data\_index\_folder):

os.makedirs(data\_index\_folder)

resp\_index1 = LocalTerminal.get\_historical(index1, 'PX\_LAST',start=date\_1, end= date\_2)

merged\_df1 = resp\_index1.as\_frame()

resp\_index2 = LocalTerminal.get\_historical(index2, 'PX\_LAST',start=date\_1, end= date\_2)

merged\_df2 = resp\_index2.as\_frame()

#SAVE first index

file\_name\_h5 = f"df\_{index1}\_{date\_1}\_{date\_2}.h5"

df\_index1\_path\_h5 = os.path.join(data\_index\_folder, file\_name\_h5)

metadata = dict(data="data monthly return",provider="Bloomberg",indice=index1)

h5store(df\_index1\_path\_h5 ,merged\_df1,metadata)

file\_name\_index1\_xlsx = f"df\_{index1}\_{date\_1}\_{date\_2}.xlsx"

df\_index1\_path = os.path.join(data\_index\_folder, file\_name\_index1\_xlsx)

merged\_df1.to\_excel(df\_index1\_path)

#SAVE the second index

file\_name\_h5 = f"df\_{index2}\_{date\_1}\_{date\_2}.h5"

df\_index2\_path\_h5 = os.path.join(data\_index\_folder, file\_name\_h5)

metadata = dict(data="data monthly return",provider="Bloomberg",indice=index2)

h5store(df\_index2\_path\_h5 ,merged\_df2,metadata)

file\_name\_index2\_xlsx = f"df\_{index2}\_{date\_1}\_{date\_2}.xlsx"

df\_index2\_path = os.path.join(data\_index\_folder, file\_name\_index2\_xlsx)

merged\_df2.to\_excel(df\_index2\_path)

return merged\_df1, merged\_df2

def h5load(filename):

with pd.HDFStore(filename) as store:

data = store['mydata']

metadata = store.get\_storer('mydata').attrs.metadata

data.attrs = metadata

return data, metadata

retrieve\_index('LCXR Index', 'SCXR Index', '20051201', '20240720')

pypfopt modele arbitrage

import pypfopt

import pypfopt.objective\_functions

from gestion\_hdf5 import h5load

from tia.bbg import LocalTerminal

import numpy as np

import pandas as pd

from tia.bbg import LocalTerminal

from collections import OrderedDict

import pickle

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

from sklearn.metrics import r2\_score

from datetime import datetime

from dateutil.relativedelta import relativedelta

from tia.bbg import LocalTerminal

import datetime

from pandas import Timestamp

import os

import luigi

from global\_parameter import ClassGlobal, SelectData, StrategyPypfopt

from functions\_assets import get\_assets

from functions\_data import \_build\_easy\_map, end\_month\_data

class Strategy\_Pypfopt(luigi.Task):

#zone = luigi.Parameter()

#period = luigi.Parameter()

#Doc\_name = luigi.Parameter()

def run(self):

known\_future\_cov\_matrix= StrategyPypfopt().known\_future\_cov\_matrix == "True"

known\_future\_expected\_returns= StrategyPypfopt().known\_future\_expected\_returns == "True"

rolling = int(StrategyPypfopt().rolling)

start\_date = '2001-05-31' #start\_date minimum possible pour l'EU : 2001-01-31

end\_date = '2024-06-30'

#pour la chine

#start\_date = '2019-07-31'

#mettre une date de fin de mois

#(pour know\_future\_cov\_known) : end\_date doit être il y a plus de deux mois par rapport à la date du jour car la cov du mois suivant doit être connue

#chargement de la data

#Expected returns estimés (EDR + YTM historique moins la prime de risque)

path\_exp\_returns = os.path.join(ClassGlobal().path\_output,"Simulation\{}\Assets\Assets\_{}".format(ClassGlobal().simulation,datetime.datetime.now().strftime("%Y\_%m\_%d")))

file\_exp\_returns = "Yields\_"+ClassGlobal().Doc\_name+".h5" #Expected YTM (donc vrai YTM - un certain pourcentage %)

expected\_returns, metadata1 = h5load(path\_exp\_returns + "\\" + file\_exp\_returns)

#transforme YTM en vrai YTM enregistrés par bloom avec la colonne equity qui correspond à l'EDR (on restructuera l'ensemble du dataframe par la suite)

YTM, metadata = h5load(path\_exp\_returns + "\\" + file\_exp\_returns)

YTM['HG Corpo '+SelectData().zone] = YTM['HG Corpo '+SelectData().zone] +0.005

YTM['HY Corpo '+SelectData().zone] = YTM['HY Corpo '+SelectData().zone] +0.025

#Prix des index enregistrés par bloom

path\_prices = os.path.join(ClassGlobal().path\_output,"Simulation\{}\Assets\Assets\_{}".format(ClassGlobal().simulation,datetime.datetime.now().strftime("%Y\_%m\_%d")))

file\_prices = "Prices\_"+ClassGlobal().Doc\_name+".h5"

prices, metabase2 = h5load(path\_prices + "\\" + file\_prices)

path\_vol = os.path.join(ClassGlobal().path\_output,"Simulation\{}\Assets\Assets\_{}".format(ClassGlobal().simulation,datetime.datetime.now().strftime("%Y\_%m\_%d")))

file\_vol = "Vol\_"+ClassGlobal().Doc\_name+".h5"

vol, metabase3 = h5load(path\_vol+ "\\" + file\_vol)

#Affichage de la data

print("Expected Returns:")

print(expected\_returns)

print("YTM:")

print(YTM)

print("Prices:")

print(prices)

print("Vol :")

print(vol)

#nom pour les fichiers finaux

nom = ''

if(known\_future\_cov\_matrix):

nom = nom +'\_known\_cov\_+1m' #je connais déjà la matrice de variance covariance du mois suivant

if(known\_future\_cov\_matrix==False):

nom = nom+'\_cov\_estimated' #j'estime la matrice de variance covariance du mois suivant par la matrice variance covariance historique

if(known\_future\_expected\_returns):

nom = nom + '\_known\_expReturns\_+1m' #je connais déjà les rendemments du mois suivant

if(known\_future\_expected\_returns==False):

nom = nom+'\_expReturns\_estimated' #j'estime les rendemments du mois suivant (EDR + data historique moins prime de risques)

if(rolling and known\_future\_cov\_matrix==False):

nom = nom+'\_roll\_'+str(rolling)+'\_M'

if(rolling== 1 and known\_future\_cov\_matrix==False):

nom = nom+'\_cov\_1monthInDaily' #on estime la covariance en prenant des données daily sur le mois et la cov dans le modele correspond à la cov du mois actuel seulement

#calcul des rendemments des index en mensuel

monthly\_returns = prices.pct\_change()

print("Monthly Returns : ")

print(monthly\_returns)

covariance\_dict = {} #dictionnaire de covariance qui contiendra la covariance mensuelle estimée

#restructure le fichier YTM en gardant uniquement les index : on fait une annualisation du return mensuel constasté pour l'ensemble des classes d'actifs

#UTILE UNIQUEMENT lorsque je connais en avance les vrais returns du m+1 (car sinon j'utilise le dataframe expected\_returns)

for date in YTM.index:

YTM.loc[date,'Treasuries '+ SelectData().zone] = ((1+ monthly\_returns.loc[date,'Treasuries '+SelectData().zone])\*\*(12))-1

YTM.loc[date, 'Equity '+SelectData().zone] =((1+ monthly\_returns.loc[date,'Equity '+SelectData().zone])\*\*(12))-1

YTM.loc[date, 'HG Corpo '+SelectData().zone] =((1+ monthly\_returns.loc[date,'HG Corpo '+SelectData().zone])\*\*(12))-1

YTM.loc[date, 'HY Corpo '+SelectData().zone] =((1+ monthly\_returns.loc[date,'HY Corpo '+SelectData().zone])\*\*(12))-1

#chargement des données pour les prices en daily

assets = OrderedDict()

assets = get\_assets(SelectData().zone)

assets.popitem() #nous allons recalculer nous-même cette colonne car erreurs pour le cash

fields = ['PX\_LAST']

period = 'DAILY'

resp = LocalTerminal.get\_historical([assets[i]["ticker"] for i in assets.keys()],

fields, start=start\_date, end=end\_date, period=period)

daily\_price = resp.as\_frame()

daily\_price.columns = daily\_price.columns.droplevel(1)

daily\_price.ffill(inplace=True)

daily\_price

daily\_price.rename(columns=\_build\_easy\_map(assets), inplace=True)

daily\_return = daily\_price.pct\_change()

daily\_return = daily\_return[start\_date:]

index = daily\_return.index

#on enregistre la covariance du mois dans le dictionnaire covariance\_dict avec des données daily, la clé correspond au mois où la covariance est observée

for date in monthly\_returns.index:

copy\_daily\_return = daily\_return.loc[(index.month == date.month) & (index.year == date.year)]

copy\_daily\_return = copy\_daily\_return.dropna()

cov\_1\_month = copy\_daily\_return.cov(min\_periods=20)

covariance\_dict[date] = cov\_1\_month\*252

#chargement des données pr le cash EU, utile aussi pour le paramètre du taux sans risque dans le .max\_sharpe

if SelectData().zone == 'EU':

indexes = ['EONIA Index', 'ESTRON Index']

if SelectData().zone == 'US':

indexes = ['US0001M Index', 'SOFRRATE Index']

if SelectData().zone == 'CN':

indexes = ['CHBM7D Index']

resp = LocalTerminal.get\_historical(indexes, 'PX\_LAST',start=start\_date, end=end\_date, period='DAILY')

data = resp.as\_frame()

if SelectData().zone=='CN':

data = data.rename(columns={'CHBM7D Index': 'ST index'})

data.columns = data.columns.droplevel(1)

str\_series = pd.Series(index=pd.to\_datetime(data.index), name='ST index', dtype='float64')

if SelectData().zone !='CN':

ind = np.isnan(data[indexes[1]])

str\_series.loc[~ind] = data.loc[~ind, indexes[1]]

str\_series.loc[ind] = data.loc[ind, indexes[0]]

else :

str\_series = data['ST index']

print(str\_series)

dataframe\_cash = str\_series.to\_frame()

print(dataframe\_cash["ST index"]/100) # on annualise les perf journalières

dataframe\_cash["perf\_annualise"] = ((1 + dataframe\_cash["ST index"] / 100) \*\* (1 / 365)) - 1

dataframe\_cash["cum\_perf"] = dataframe\_cash["perf\_annualise"]

dataframe\_cash["cum\_perf"] = (1 + dataframe\_cash["perf\_annualise"]).cumprod() #on calcule le produit cumulative

print(dataframe\_cash)

dataframe\_cash["daily\_return"] = dataframe\_cash["cum\_perf"].pct\_change()

#on choisi des sous datasets pour avoir la meme date de depart et de fin

#end\_date = min(monthly\_returns.index[-1], expected\_returns.index[-1]) #

monthly\_returns\_aligned = monthly\_returns.loc[start\_date:end\_date]

expected\_returns\_aligned = expected\_returns.loc[start\_date:end\_date]

print(monthly\_returns\_aligned.head())

print(expected\_returns\_aligned.head())

#drop colonnes cash ~dans expected\_returns\_aligned car (initile dans le calcul du portfolio opti)

monthly\_returns\_aligned.drop('Cash '+ SelectData().zone, axis=1, inplace=True)

expected\_returns\_aligned.drop('Cash '+ SelectData().zone, axis=1, inplace=True)

YTM.drop('Cash '+ SelectData().zone, axis=1, inplace=True)

#on calcule la perf cumulative du cash car mauvaise initialement (utile pr la partie où l'on va long cash only)

#cum\_perf est comme un indice en daily, nous allons maintenant récupérer les valeurs de cet indice fictif mensuellement.

for date in monthly\_returns\_aligned.index[1:] :

month = date.month

year = date.year

data\_for\_month = dataframe\_cash[(dataframe\_cash.index.month == month) & (dataframe\_cash.index.year == year)]

cash\_end\_month = data\_for\_month.iloc[-1]["cum\_perf"]

date\_precedent = date - relativedelta(months=1)

data\_for\_month\_prev = dataframe\_cash[(dataframe\_cash.index.month == date\_precedent.month ) & (dataframe\_cash.index.year == date\_precedent.year)]

cash\_prev\_month = data\_for\_month\_prev.iloc[-1]["cum\_perf"]

#monthly\_returns\_aligned.at[date, 'Cash EU'] = (cash\_eu\_end\_month - cash\_eu\_prev\_month) /(cash\_eu\_prev\_month)

monthly\_returns.at[date, 'Cash '+SelectData().zone] = (cash\_end\_month - cash\_prev\_month) /(cash\_prev\_month)

'''with pd.ExcelWriter('alignés.xlsx') as writer:

monthly\_returns\_aligned.to\_excel(writer, sheet\_name='Monthly Returns Aligned')

expected\_returns\_aligned.to\_excel(writer, sheet\_name='Expected Returns Aligned')'''

#pour obtenir le risk free rate utile dans le parametre du max\_sharpe

resp2 = LocalTerminal.get\_historical(indexes, 'PX\_LAST',start=start\_date, end=end\_date, period='MONTHLY')

data2 = resp2.as\_frame()

data2.columns = data2.columns.droplevel(1)

if SelectData().zone=='CN':

data2 = data2.rename(columns={'CHBM7D Index': 'ST index'})

str\_series2 = pd.Series(index=pd.to\_datetime(data2.index), name='ST index', dtype='float64')

if SelectData().zone !='CN':

ind2 = np.isnan(data2[indexes[1]])

str\_series2.loc[~ind2] = data2.loc[~ind2, indexes[1]]

str\_series2.loc[ind2] = data2.loc[ind2, indexes[0]]

else :

str\_series2 = data2['ST index']

str\_series2 = end\_month\_data(str\_series2, True)

weights = []

date\_tab = []

estimated\_vol = pd.DataFrame(index=monthly\_returns\_aligned.index)

datedeb = monthly\_returns\_aligned.index[rolling+1]

for i, date in enumerate(monthly\_returns\_aligned.index[:-1]):

if i>0:

if(date.strftime("%Y-%m-%d")<monthly\_returns\_aligned.index[rolling].strftime("%Y-%m-%d")):

continue

if(date.strftime("%Y-%m-%d") =='2022-09-30'):

print("ok")

#monthly\_returns\_know contient toutes les returns monthly avant la date d'aujourd'hui donc le mois en cours et les mois d'avants

monthly\_returns\_know= monthly\_returns\_aligned.loc[:date]

#cov\_matrix du mois d'après que l'on suppose déjà connu

if(known\_future\_cov\_matrix):

cov\_matrix = covariance\_dict[Timestamp(monthly\_returns\_aligned.index[i+1])]

else :

if rolling >1 :

if(len(monthly\_returns\_know) >rolling): #conservation des 5 dernières années pr l'estimation de la cov matrix

monthly\_returns\_know = monthly\_returns\_know[-rolling:]

cov\_matrix = monthly\_returns\_know.cov()\*12

else :

cov\_matrix = covariance\_dict[Timestamp(monthly\_returns\_aligned.index[i])]

#trace the volatilities

estimated\_vol.loc[date, 'Treasuries '+SelectData().zone] = np.sqrt(cov\_matrix.loc['Treasuries '+SelectData().zone, 'Treasuries '+SelectData().zone]\*252)

estimated\_vol.loc[date, 'HG Corpo '+SelectData().zone] = np.sqrt(cov\_matrix.loc['HG Corpo '+SelectData().zone, 'HG Corpo '+SelectData().zone]\*252)

estimated\_vol.loc[date, 'HY Corpo '+SelectData().zone] = np.sqrt(cov\_matrix.loc['HY Corpo '+SelectData().zone, 'HY Corpo '+SelectData().zone]\*252)

estimated\_vol.loc[date, 'Equity '+SelectData().zone] = np.sqrt(cov\_matrix.loc['Equity '+SelectData().zone, 'Equity '+SelectData().zone]\*252)

#en connaisant les vrais YTM du mois suivant

if(known\_future\_expected\_returns):

ef = pypfopt.efficient\_frontier.EfficientFrontier(YTM.loc[Timestamp(monthly\_returns\_aligned.index[i+1])], cov\_matrix, weight\_bounds=(0, 1), solver=None, verbose=False, solver\_options=None)

else:

ef = pypfopt.efficient\_frontier.EfficientFrontier(expected\_returns\_aligned.loc[date], cov\_matrix, weight\_bounds=(0, 1), solver=None, verbose=False, solver\_options=None)

#ligne lorsque l'on ne connais pas les vrais expected YTM du prochain mois

#ef = pypfopt.efficient\_frontier.EfficientFrontier(expected\_returns\_aligned.loc[date], cov\_matrix, weight\_bounds=(0, 1), solver=None, verbose=False, solver\_options=None)

print(date)

try :

riskfree = str\_series2.loc[date,'ST index']

poids = ef.max\_sharpe(risk\_free\_rate=riskfree/100)

weights.append({'date': date, 'weights': poids})

except Exception as e :

print("error for date", date," ", e)

if(len(weights)==0):

poids = OrderedDict([('Treasuries CN', 0.0),('HG Corpo CN', 0.0),('HY Corpo CN', 0.0),('Equity CN', 0.0)])

else :

poids = weights[-1]['weights'] # Get the previous weights

weights.append({'date': date, 'weights': poids})

data = {}

# Parcours de chaque élément dans la liste weights et stocke dans le dataframe df

for element in weights:

date = element['date'] # extraction de la date

poids = element['weights'] # extraction des poids

data[date] = poids

df = pd.DataFrame(data)

df = df.transpose()

df\_weights = df.copy()

df\_weights.index = monthly\_returns[datedeb:end\_date].index

weighted\_returns = monthly\_returns[datedeb:end\_date]\*df\_weights

#les rendemments maximaux pr chaque dates parmis les 5 catégories d'actifs

max\_values = monthly\_returns[datedeb:end\_date].max(axis=1)

strategy\_returns = weighted\_returns.sum(axis=1) #somme les poids \* les returns

strategy\_returns\_df = pd.DataFrame(data={'strategy\_returns': strategy\_returns})

print(strategy\_returns\_df)

cumulative\_performance = (1 + strategy\_returns\_df['strategy\_returns']).cumprod()

print(cumulative\_performance)

years = (strategy\_returns\_df.index[-1] - strategy\_returns\_df.index[0]).days / 365

annualized\_return = (cumulative\_performance[-1])\*\*(1/years) - 1

print(annualized\_return)

print(df\_weights)

#cas LONG ONLY HG

copy = df

copy['Treasuries '+ SelectData().zone] =0

copy['HG Corpo ' + SelectData().zone] = 1

copy['HY Corpo ' + SelectData().zone] = 0

copy['Equity '+ SelectData().zone] = 0

copy['Cash '+ SelectData().zone] = 0

print(copy)

copy.index = monthly\_returns[datedeb:end\_date].index

weighted\_returns\_HG = monthly\_returns[datedeb:end\_date]\*copy

strategy\_returns\_HG = weighted\_returns\_HG.sum(axis=1)

strategy\_returns\_df\_HG = pd.DataFrame(data={'strategy\_returns': strategy\_returns\_HG})

print(strategy\_returns\_df\_HG)

cumulative\_performance\_HG = (1 + strategy\_returns\_df\_HG['strategy\_returns']).cumprod()

#cas LONG la classe qui perf le mieux a chaque fois

strategy\_returns\_df\_maxV = pd.DataFrame(data={'strategy\_returns':max\_values})

print(strategy\_returns\_df\_maxV )

cumulative\_performance\_maxV = (1 + strategy\_returns\_df\_maxV ['strategy\_returns']).cumprod()

#cas LONG ONLY HY

copy\_HY = df

copy\_HY['Treasuries '+ SelectData().zone] = 0

copy\_HY['HG Corpo '+ SelectData().zone] = 0

copy\_HY['HY Corpo '+ SelectData().zone] = 1

copy\_HY['Equity '+ SelectData().zone] = 0

copy\_HY['Cash '+ SelectData().zone] = 0

copy\_HY.index = monthly\_returns[datedeb:end\_date].index

weighted\_returns\_HY = monthly\_returns[datedeb:]\*copy\_HY

strategy\_returns\_HY = weighted\_returns\_HY.sum(axis=1)

strategy\_returns\_df\_HY = pd.DataFrame(data={'strategy\_returns': strategy\_returns\_HY})

print(strategy\_returns\_df\_HY)

cumulative\_performance\_HY = (1 + strategy\_returns\_df\_HY['strategy\_returns']).cumprod()

#cas LONG ONLY EQUITY

copy\_e = df

copy\_e['Treasuries '+ SelectData().zone] = 0

copy\_e['HG Corpo '+ SelectData().zone] = 0

copy\_e['HY Corpo '+ SelectData().zone] = 0

copy\_e['Equity '+ SelectData().zone] = 1

copy\_e['Cash '+ SelectData().zone] = 0

copy\_e.index = monthly\_returns[datedeb:end\_date].index

weighted\_returns\_e = monthly\_returns[datedeb:]\*copy\_e

strategy\_returns\_e = weighted\_returns\_e.sum(axis=1)

strategy\_returns\_df\_e = pd.DataFrame(data={'strategy\_returns': strategy\_returns\_e})

print(strategy\_returns\_df\_e)

cumulative\_performance\_e = (1 + strategy\_returns\_df\_e['strategy\_returns']).cumprod()

#cas LONG ONLY CASH EU

copy\_c = df

copy\_c['Treasuries '+SelectData().zone] = 0

copy\_c['HG Corpo '+SelectData().zone] = 0

copy\_c['HY Corpo '+SelectData().zone] = 0

copy\_c['Equity '+SelectData().zone] = 0

copy\_c['Cash '+SelectData().zone] = 1

copy\_c.index = monthly\_returns[datedeb:end\_date].index

weighted\_returns\_c = monthly\_returns[datedeb:]\*copy\_c

strategy\_returns\_c = weighted\_returns\_c.sum(axis=1)

strategy\_returns\_df\_c = pd.DataFrame(data={'strategy\_returns': strategy\_returns\_c})

print(strategy\_returns\_df\_c)

cumulative\_performance\_c = (1 + strategy\_returns\_df\_c['strategy\_returns']).cumprod()

#cas LONG ONLY TREASURIES

copy\_t = df

copy\_t['Treasuries '+SelectData().zone] = 1

copy\_t['HG Corpo '+SelectData().zone] = 0

copy\_t['HY Corpo '+SelectData().zone] = 0

copy\_t['Equity '+SelectData().zone] = 0

copy\_t['Cash '+SelectData().zone] = 0

copy\_t.index = monthly\_returns[datedeb:end\_date].index

weighted\_returns\_t = monthly\_returns[datedeb:]\*copy\_t

strategy\_returns\_t = weighted\_returns\_t.sum(axis=1)

strategy\_returns\_df\_t = pd.DataFrame(data={'strategy\_returns': strategy\_returns\_t})

print(strategy\_returns\_df\_t)

cumulative\_performance\_t = (1 + strategy\_returns\_df\_t['strategy\_returns']).cumprod()

#création du dossier pypfopt et du sous dossier correspond aux nom des paramètres de la fonction

pypfopt\_dossier = os.path.join(ClassGlobal().path\_output,"Simulation\{}\Pypfopt".format(ClassGlobal().simulation,datetime.datetime.now().strftime("%Y\_%m\_%d"),"Pypfopt"))

if not os.path.exists(pypfopt\_dossier):

os.makedirs(pypfopt\_dossier)

destination\_folder = os.path.join(pypfopt\_dossier,nom)

if not os.path.exists(destination\_folder):

os.makedirs(destination\_folder)

plt.figure(figsize=(12, 6))

cumulative\_performance\_maxV.plot()

plt.title('Rendement Cumulé maximal')

plt.xlabel('Date')

plt.ylabel('Rendement Cumulé maximal')

plt.grid()

destination\_file = destination\_folder + "/performance\_cumulative\_maximale"+nom+".png"

plt.savefig(destination\_file)

plt.close()

#création du graphe de performance cumulative

plt.figure(figsize=(10, 6))

#plt.figure(figsize=(10, 6))

plt.plot(cumulative\_performance.index, cumulative\_performance.values, label='Performance cumulative', color='brown')

plt.plot(cumulative\_performance\_HG.index, cumulative\_performance\_HG.values, label='Performance cumulative HG', color='orange')

plt.plot(cumulative\_performance\_HY.index, cumulative\_performance\_HY.values, label='Performance cumulative HY', color='green')

plt.plot(cumulative\_performance\_e.index, cumulative\_performance\_e.values, label='Performance cumulative equities', color='red')

plt.plot(cumulative\_performance\_t.index, cumulative\_performance\_t.values, label='Performance cumulative treasuries', color='lightskyblue')

plt.plot(cumulative\_performance\_c.index, cumulative\_performance\_c.values, label='Cash '+ SelectData().zone, color='black')

plt.title('Performance cumulative with '+nom)

plt.xlabel('Date')

plt.ylabel('Valeur cumulative')

plt.legend()

plt.grid(True)

destination\_file = destination\_folder + "/performance\_cumulative\_"+nom+".png"

plt.savefig(destination\_file)

plt.close()

fig, ax = plt.subplots(figsize=(10, 6))

colors = ['lightskyblue', 'orange', 'green','red']

i = 0

for col in df\_weights.columns:

ax.plot(df\_weights.index, df\_weights[col], label=col,color=colors[i] )

i +=1

ax.legend()

ax.set\_title('Évolution des poids au fil du temps with '+nom)

ax.set\_xlabel('Date')

ax.set\_ylabel('Poids')

plt.savefig(os.path.join(destination\_folder, "weight\_matrix"+nom+".png"))

print("La figure a été enregistrée sous :", destination\_file)

plt.close(fig)

#affichons le graphe de la volatilité

if (known\_future\_cov\_matrix) : #correspond à la racine de la diagonale de la matrice covariance variance estimée en daily

plt.figure(figsize=(10, 6))

i = 0

colors = ['lightskyblue', 'orange', 'green','red']

for column in estimated\_vol.columns:

plt.plot(estimated\_vol.index, estimated\_vol[column], label=column, color=colors[i])

i +=1

plt.xlabel('Date')

plt.ylabel('Volatilité estimée par le modele')

plt.title('Volatilité en fonction du time '+nom)

plt.legend()

plt.savefig(os.path.join(destination\_folder, "volatility\_estimated" +nom +".png"))

print("La figure a été enregistrée sous :", destination\_file)

plt.close(fig)

else : #sinon on utilise le fichier Vol déjà calculé dans l'algo précédemment

vol = vol[start\_date:]

plt.figure(figsize=(10, 6))

i=0

colors = ['lightskyblue', 'orange', 'green','red', 'black']

for column in vol.columns:

plt.plot(vol.index, vol[column], label=column, color=colors[i])

i +=1

plt.xlabel('Date')

plt.ylabel('Volatilité estimée par notre algo')

plt.title('Volatilité en fonction du time ' + nom)

plt.legend()

plt.savefig(os.path.join(destination\_folder, "volatility\_estimated"+nom+".png"))

print("La figure a été enregistrée sous :", destination\_file)

plt.close(fig)

#printons les expected returns

plt.figure(figsize=(10, 6))

i = 0

colors = ['lightskyblue', 'orange', 'green','red']

for column in expected\_returns\_aligned:

plt.plot(expected\_returns\_aligned.index, expected\_returns\_aligned[column], label=column, color=colors[i])

i+=1

plt.xlabel('Date')

plt.ylabel('Expected returns with '+nom)

plt.title('Expected returns en fonction du time '+nom)

plt.legend()

plt.savefig(os.path.join(destination\_folder, "expected\_returns"+nom+".png"))

print("La figure a été enregistrée sous :", destination\_file)

#printer le graphe de la théorie moderne du ptf markowitz

from pypfopt import plotting

S = cov\_matrix

mu = expected\_returns\_aligned.loc[end\_date]

n\_samples = 1000

w = np.random.dirichlet(np.ones(len(mu)), n\_samples)

rets = w.dot(mu)

stds = np.sqrt((w.T \* (S @ w.T)).sum(axis=0))

sharpes = rets / stds

ef = pypfopt.efficient\_frontier.EfficientFrontier(mu, S)

fig, ax = plt.subplots()

plotting.plot\_efficient\_frontier(ef, ax=ax, show\_assets=False)

''# Find and plot the tangency portfolio

ef2 = pypfopt.efficient\_frontier.EfficientFrontier(mu, S)

ef2.max\_sharpe()

ret\_tangent, std\_tangent, \_ = ef2.portfolio\_performance()

# Plot random portfolios

ax.scatter(stds, rets, marker=".", c=sharpes, cmap="gray")

print(mu)

print(S.iloc[0,0])

ax.set\_title("Efficient Frontier with random portfolios with "+nom)

#max\_sharpe =monthly\_returns[end\_date]\*df.loc[end\_date]

#ordonnee = max\_sharp.sum(axis=1)

ax.scatter(std\_tangent, ret\_tangent, marker="o", color='r', label='Portfolio Optimal')

ax.legend()

plt.tight\_layout()

plt.savefig(os.path.join(destination\_folder, "Efficient Frontier with "+nom+".png"))

plt.close(fig)

if \_\_name\_\_ == "\_\_main\_\_":

luigi.build([ClassGlobal(), SelectData(), StrategyPypfopt()],local\_scheduler = True)

luigi.build([Strategy\_Pypfopt()],local\_scheduler = True, no\_lock=True)