Market risk: Project

Alexandre Deroux and Stanislas Bérard[*]

December 28, 2023

[*]ESILV Paris and De Vinci Research Center

Contents

1 Question A (Ex2, part of Q1 and of Q2 of TD1)

1.1 Part a)

First we retrieved the dataset, calculated the geometric returns and sorted them in ascending order.

Then we implemented:

$$\hat{F}(x) = \frac{1}{n}\sum_{i=1}^{n} \kappa\left(\frac{x - X_i}{h}\right)$$

Where here:

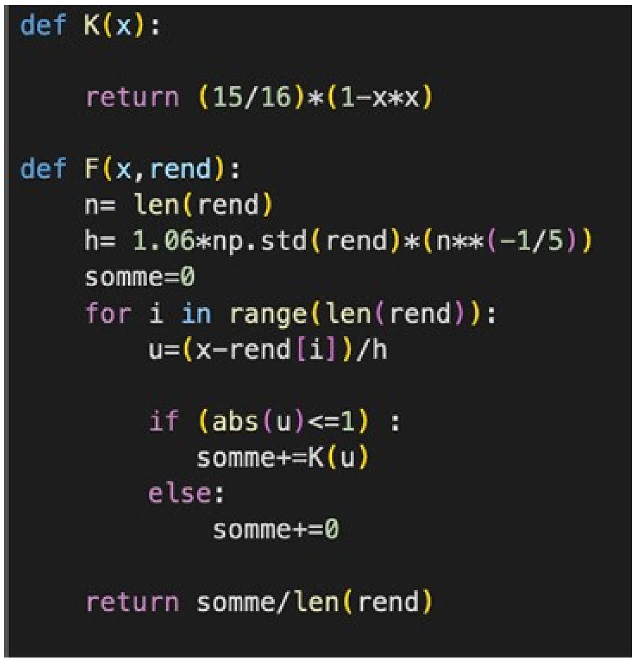
$$\kappa(u) = \frac{15}{16}\left(\left(1 - u^2\right)^2\right)\mathbf{1}_{|u|\leq 1}$$

And estimate h with the Scot Rule of the thumb:

$$h = \left(\frac{4\sigma^5}{3n}\right)^{\frac{1}{5}}$$

Here:

- h is the bandwidth.
- σ represents the standard deviation of the data.
- n is the sample size.

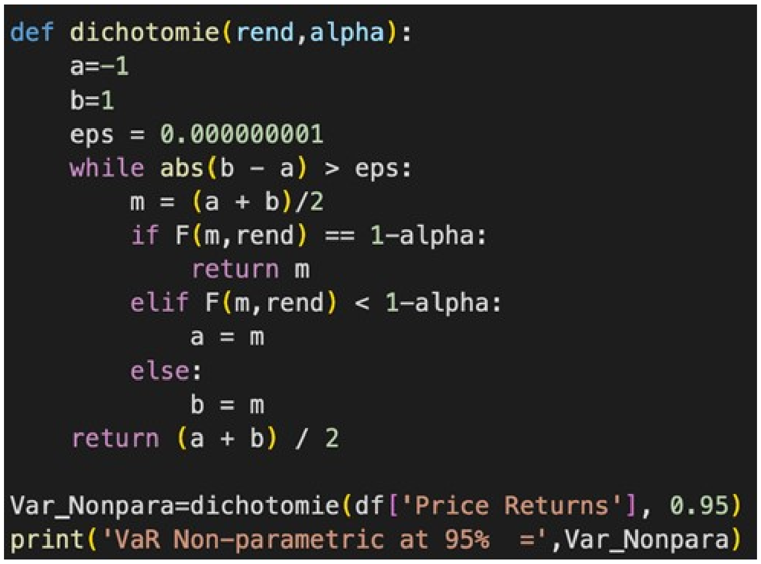


```python
def K(x):

    return (15/16)*(1-x*x)

def F(x,rend):
    n= len(rend)
    h= 1.06*np.std(rend)*(n**(-1/5))
    somme=0
    for i in range(len(rend)):
        u=(x-rend[i])/h

        if (abs(u)<=1) :
           somme+=K(u)
        else:
            somme+=0

    return somme/len(rend)
```

Then, to calculate the VaR, we solve the following problem:

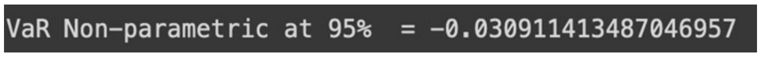
$$\arg\min(F(x) - \alpha) = VaR_\alpha$$

To do this, we used a dichotomy algorithm:



```python
def dichotomie(rend,alpha):
    a=-1
    b=1
    eps = 0.000000001
    while abs(b - a) > eps:
        m = (a + b)/2
        if F(m,rend) == 1-alpha:
            return m
        elif F(m,rend) < 1-alpha:
            a = m
        else:
            b = m
    return (a + b) / 2

Var_Nonpara=dichotomie(df['Price Returns'], 0.95)
print('VaR Non-parametric at 95%  =',Var_Nonpara)
```

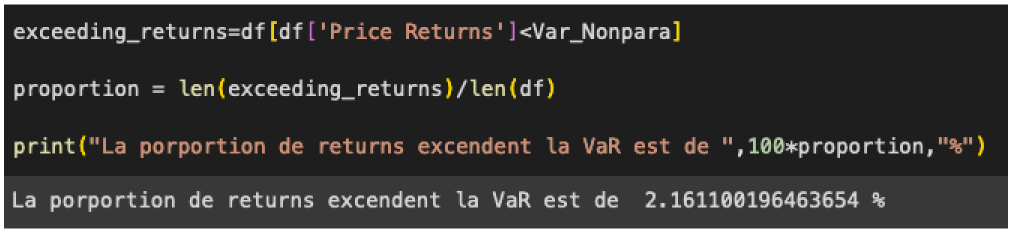
The result is:



```
VaR Non-parametric at 95%  = -0.030911413487046957
```

1.2 Part b)

We calculate the number of returns that exceed the VaR calculated above and calculate the ratio of this number to the number of returns between January 2017 and December 2018.



```python
exceeding_returns=df[df['Price Returns']<Var_Nonpara]

proportion = len(exceeding_returns)/len(df)

print("La porportion de returns excendent la VaR est de ",100*proportion,"%")
```

```
La porportion de returns excendent la VaR est de  2.161100196463654 %
```

The observed proportion of 2.16% slightly exceeds our Value at Risk (VaR) of 5%, indicating that our VaR estimate may be a little too conservative.

2 Question B (Ex2, Q5 of TD2)

To calculate the expected shortfall, we do it empirically, i.e. as the average of the values after the VaR calculated with the previous question.

```python
def expected_shortfall(var,data):
  losses = [-losse for losse in data if losse<var ]
  ES= sum(losses)/len(losses)
  return ES
```

We then find that:

```
L'Expected shortfall pour la VaR non-parametric est : 0.0462440611896367
```

This means that our Expected Shortfall is greater than our non-parametric VaR, which is consistent since ES covers all losses above VaR.

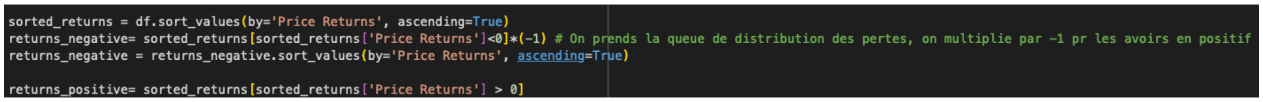
3 Question C (Ex2, Q1 and Q2 of TD3)

First, we analyzed the dataset, cleaned it, and then computed the geometric returns.

```python
import yfinance as yf
import pandas as pd

with open('/content/sample_data/Natixis stock (dataset TD1&2).txt', 'r') as file:
  lines = file.readlines()


dates =[]
prices = []

for line in lines:
  line=line.strip()
  if line:
    date,price= line.split('\t')
    dates.append(date)
    prices.append(price)

d = {'Date':dates, 'Price':prices}
df=pd.DataFrame(d)

df['Price']= df['Price'].str.replace(',','.').astype(float)
df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y')
df.set_index('Date', inplace=True)

returns =[]
for i in range(1,len(df)):
  price_yesterday= df['Price'][i-1]
  price_today = df['Price'][i]
  returns.append((price_today-price_yesterday)/price_yesterday)

df['Price Returns'] =  [None] + returns
df = df.iloc[1:]
```

3.1 Part a)

We created two dataframes according to the two tails of the returns distribution: the positive returns and the negative returns.



```
sorted_returns = df.sort_values(by='Price Returns', ascending=True)
returns_negative= sorted_returns[sorted_returns['Price Returns']<0]*(-1) # On prends la queue de distribution des pertes, on multiplie par -1 pr les avoirs en positif
returns_negative = returns_negative.sort_values(by='Price Returns', ascending=True)

returns_positive= sorted_returns[sorted_returns['Price Returns'] > 0]
```

We estimate the GEV parameters with the estimator of Pickands:

$$\xi^P_{\kappa(n),n} = \frac{1}{\log(2)} \log\left(\frac{X_{n-\kappa(n)+1:n} - X_{n-2\kappa(n)+1:n}}{X_{n-2\kappa(n)+1:n} - X_{n-4\kappa(n)+1:n}}\right)$$

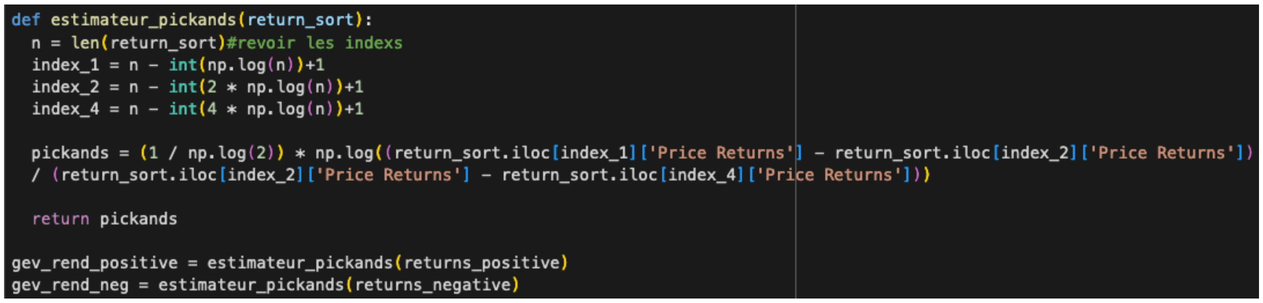
The function $\kappa(n)$ must satisfy the following two conditions:

$$\lim_{n\to\infty} \kappa(n) = \infty \tag{1}$$

$$\lim_{n\to\infty} \frac{\kappa(n)}{n} = 0, \tag{2}$$

We take the function $\log(n)$ for $\kappa(n)$ as it satisfies the two conditions.

There is the implementation:



```
def estimateur_pickands(return_sort):
  n = len(return_sort)#revoir les indexs
  index_1 = n - int(np.log(n))+1
  index_2 = n - int(2 * np.log(n))+1
  index_4 = n - int(4 * np.log(n))+1

  pickands = (1 / np.log(2)) * np.log((return_sort.iloc[index_1]['Price Returns'] - return_sort.iloc[index_2]['Price Returns'])
  / (return_sort.iloc[index_2]['Price Returns'] - return_sort.iloc[index_4]['Price Returns']))

  return pickands

gev_rend_positive = estimateur_pickands(returns_positive)
gev_rend_neg = estimateur_pickands(returns_negative)
```

And we have:

GEV Parameters for positive returns: 0.7031608888285327
GEV Parameters for negative returns: -0.5025788568345055

According to the GEV parameters, the extreme gains follow a Fréchet distribution, and the extreme losses follow a Weibull distribution.

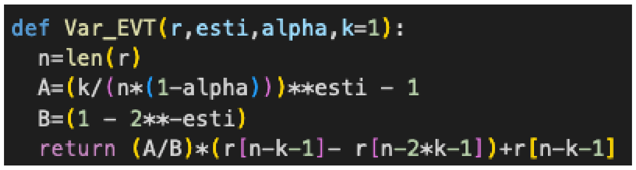
3.2 Part b)

To calculate the value at risk based on EVT we used this formula:

$$VaR(p) = \frac{\left(\frac{k}{n(1-p)}\right)^{\xi^P} - 1}{1 - 2^{-\xi^P}} (X_{n-k+1:n} - X_{n-2k+1:n}) + X_{n-k+1:n}$$

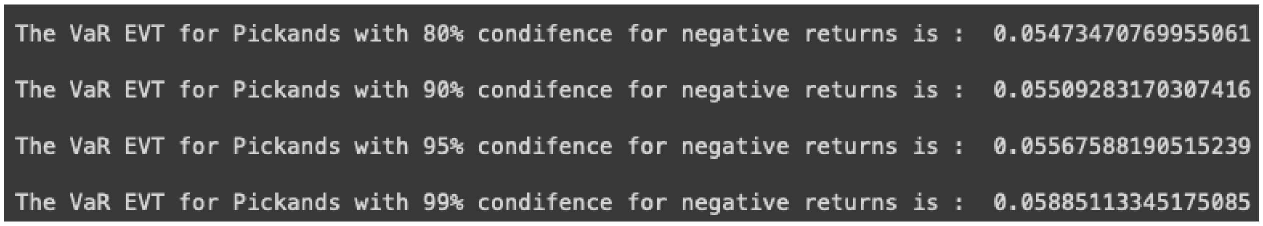
where ξ^P is the Pickands estimator of the GEV parameter.

There is our implementation:



```
def Var_EVT(r,esti,alpha,k=1):
  n=len(r)
  A=(k/(n*(1-alpha)))**esti - 1
  B=(1 - 2**-esti)
  return (A/B)*(r[n-k-1]- r[n-2*k-1])+r[n-k-1]
```

And there are our different results for various confidence levels, with the assumption of iid returns:



```
The VaR EVT for Pickands with 80% condifence for negative returns is :  0.05473470769955061

The VaR EVT for Pickands with 90% condifence for negative returns is :  0.05509283170307416

The VaR EVT for Pickands with 95% condifence for negative returns is :  0.05567588190515239

The VaR EVT for Pickands with 99% condifence for negative returns is :  0.05885113345175085
```

4 Question D (Ex2, Q3 and Q4 of TD4)

(You also have the notebook to see all the code.)

Our goal here is to estimate all the parameters of the Almgren and Chriss model, that is, η from the quadratic cost model, and the volatility of returns with respect to the time step.

$$x_k = \frac{sinh\left(K\left(T-\left(k-\frac{1}{2}\tau\right)\right)\right)}{sinh\left(KT\right)}X$$

Where:

$$K = \sqrt{\frac{\lambda\sigma^2}{\eta}} + O\left(\tau\right)$$

To find η, we use the quadratic cost model h:

$$h\left(\frac{n_k}{\tau}\right) = \xi\,\mathrm{sgn}(n_k) + \eta\frac{n_k}{\tau}$$

h is our transient impact. We must first determine it, knowing that:

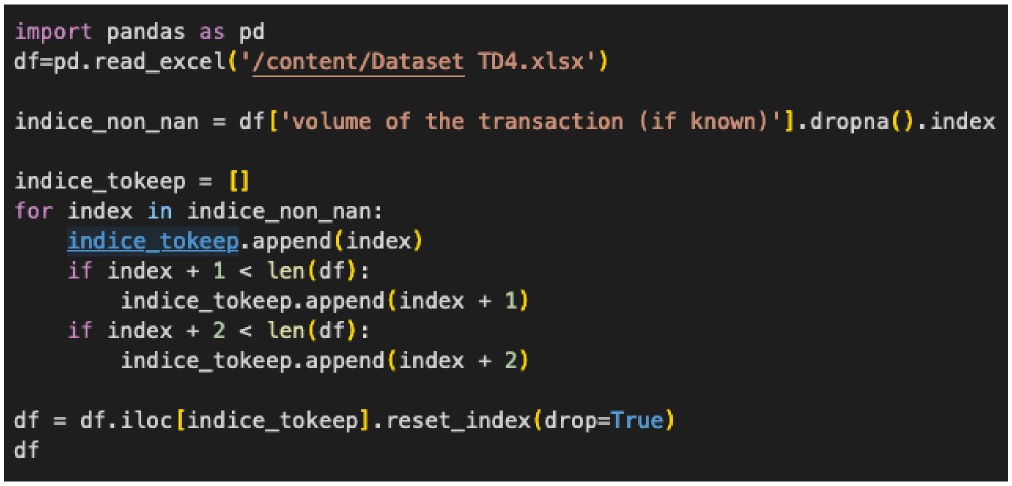
$$p(t+1) - p(t) = \text{permanent impact} + \text{transient impact}$$

$$p(t+2) - p(t) = \text{permanent impact}$$

$$p(t+1) - p(t+2) = \text{transient impact}$$

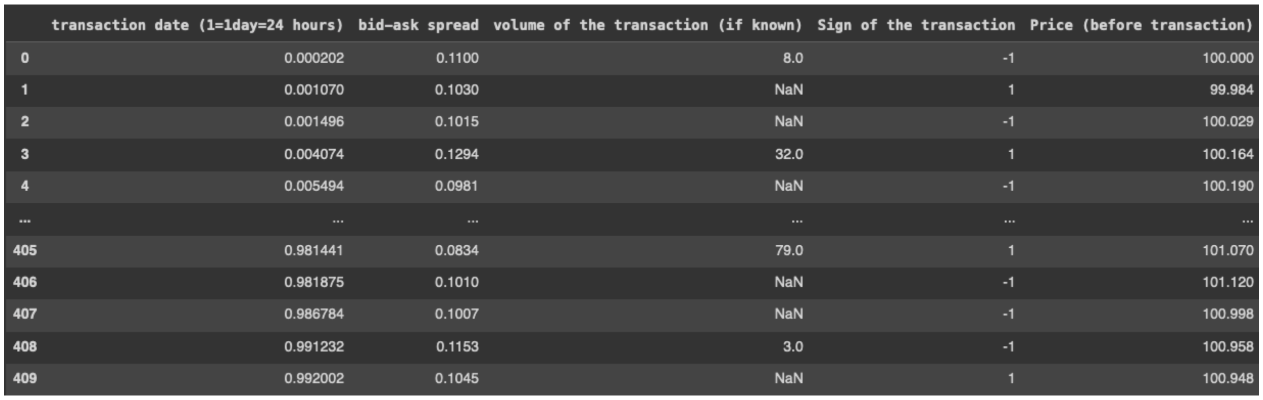
4.1 Part a)

For this, we had to clean the entire dataset so that there was no overlap between the transient impacts on different transactions. For each transaction with a volume, we check that the volumes of the following two prices are empty:



```python
import pandas as pd
df=pd.read_excel('/content/Dataset TD4.xlsx')

indice_non_nan = df['volume of the transaction (if known)'].dropna().index

indice_tokeep = []
for index in indice_non_nan:
    indice_tokeep.append(index)
    if index + 1 < len(df):
        indice_tokeep.append(index + 1)
    if index + 2 < len(df):
        indice_tokeep.append(index + 2)

df = df.iloc[indice_tokeep].reset_index(drop=True)
df
```

There is our dataframe:



	transaction date (1=1day=24 hours)	bid-ask spread	volume of the transaction (if known)	Sign of the transaction	Price (before transaction)
0	0.000202	0.1100	8.0	-1	100.000
1	0.001070	0.1030	NaN	1	99.984
2	0.001496	0.1015	NaN	-1	100.029
3	0.004074	0.1294	32.0	1	100.164
4	0.005494	0.0981	NaN	-1	100.190
...	...	...	...	...	...
405	0.981441	0.0834	79.0	1	101.070
406	0.981875	0.1010	NaN	-1	101.120
407	0.986784	0.1007	NaN	-1	100.998
408	0.991232	0.1153	3.0	-1	100.958
409	0.992002	0.1045	NaN	1	100.948

Transient impact: $h(n/t) = p(t+1) - p(t+2)$.

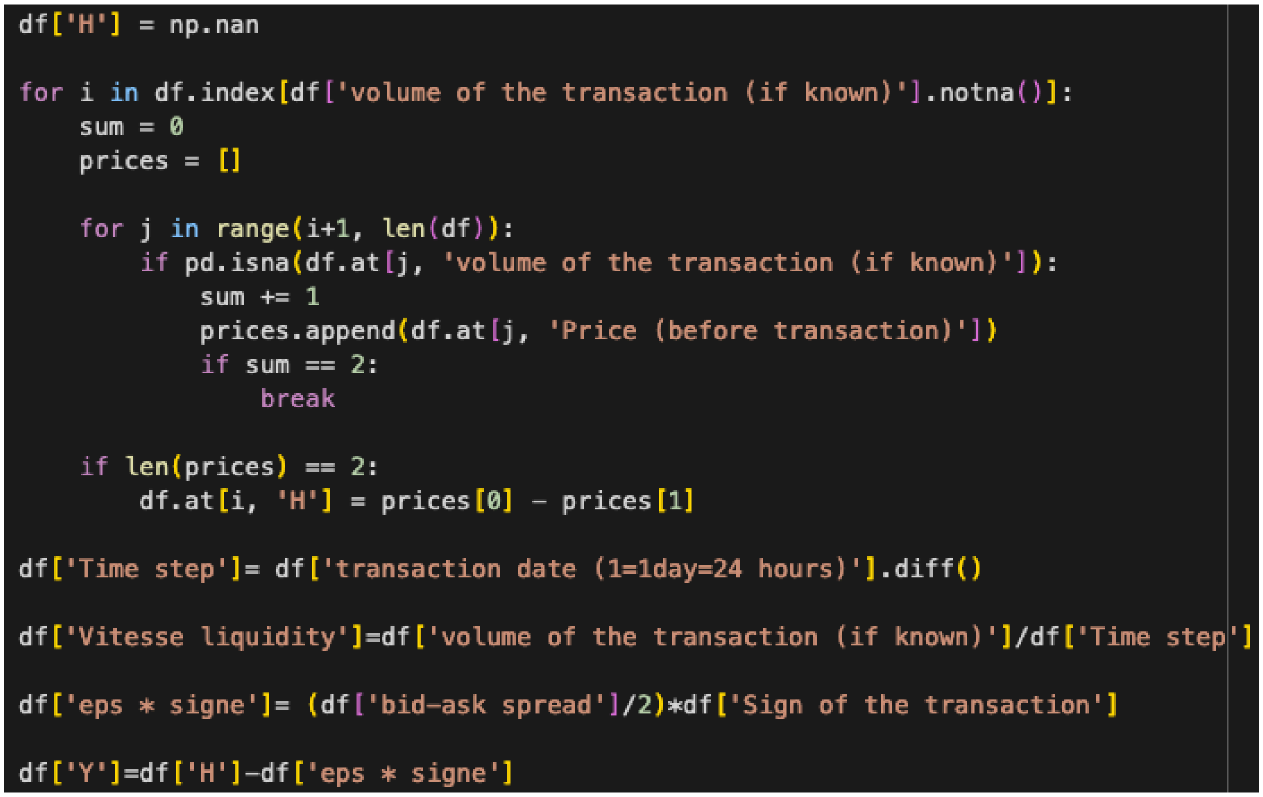
Time step: $\tau(t) = \text{transaction date}(t+1) - \text{transaction date}(t)$.

$\xi \cdot \text{sgn}(n_k) = \frac{\text{bid-ask spread}}{2} \times \text{transaction sign}$.

Transaction velocity of liquidity: $\frac{n_k}{\tau} = \frac{\text{volume of transaction}}{\text{time step}}$.

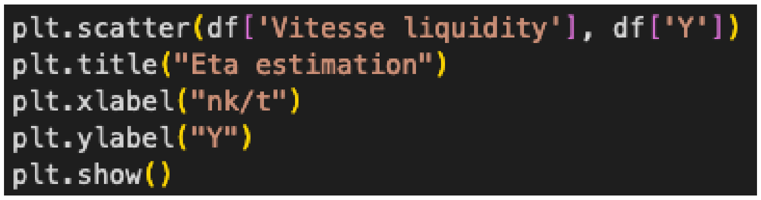
We then create a column $Y = h(n/t) - \xi \cdot \text{sgn}(n_k)$ such that we have an equation of the form: $Y = \eta \frac{n_k}{\tau}$.

Afterwards, we will perform a least squares estimation to find η.



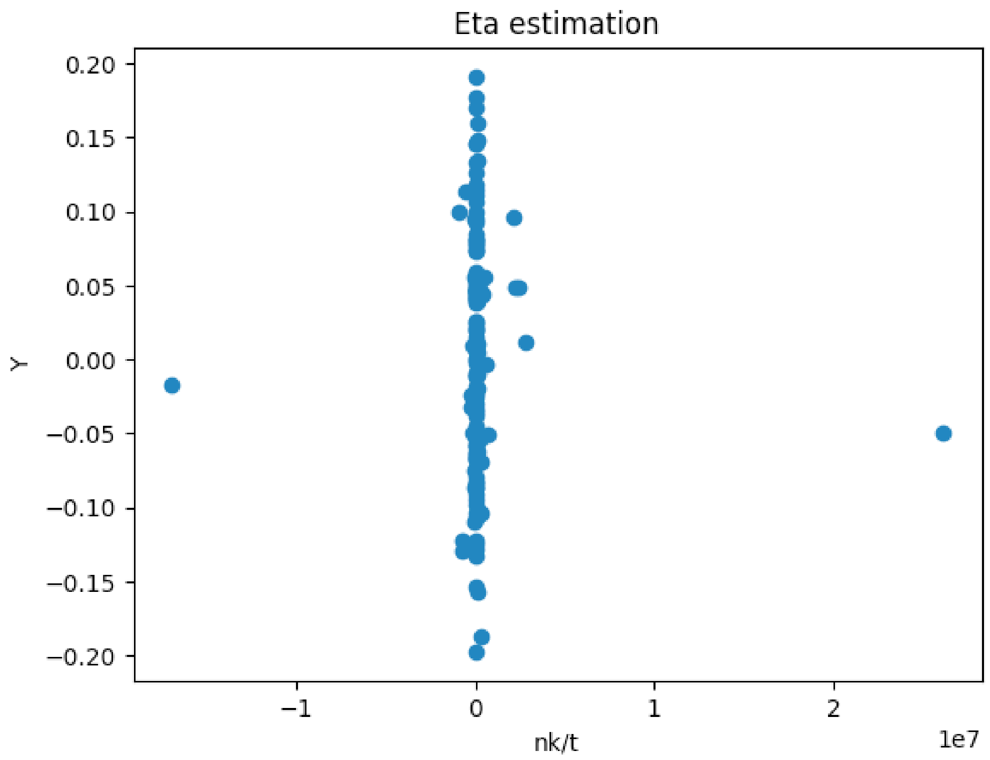
```
df['H'] = np.nan

for i in df.index[df['volume of the transaction (if known)'].notna()]:
    sum = 0
    prices = []

    for j in range(i+1, len(df)):
        if pd.isna(df.at[j, 'volume of the transaction (if known)']):
            sum += 1
            prices.append(df.at[j, 'Price (before transaction)'])
            if sum == 2:
                break

    if len(prices) == 2:
        df.at[i, 'H'] = prices[0] - prices[1]

df['Time step']= df['transaction date (1=1day=24 hours)'].diff()

df['Vitesse liquidity']=df['volume of the transaction (if known)']/df['Time step']

df['eps * signe']= (df['bid-ask spread']/2)*df['Sign of the transaction']

df['Y']=df['H']-df['eps * signe']
```

We aim to assess the efficacy of our linear regression model in terms of its predictive accuracy:

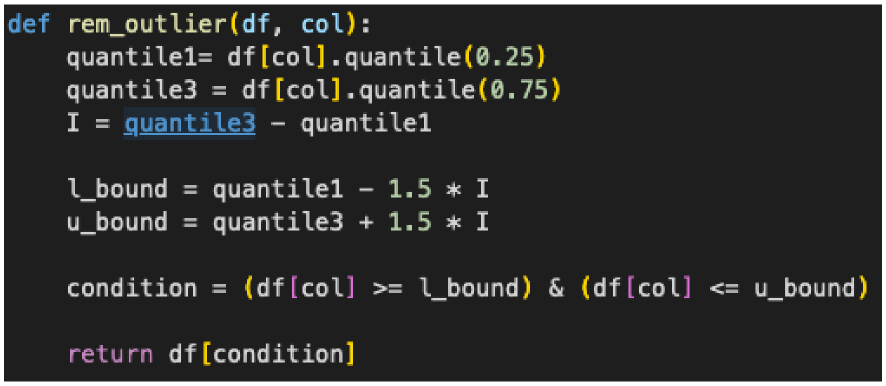


```
plt.scatter(df['Vitesse liquidity'], df['Y'])
plt.title("Eta estimation")
plt.xlabel("nk/t")
plt.ylabel("Y")
plt.show()
```

And we obtain this plot:

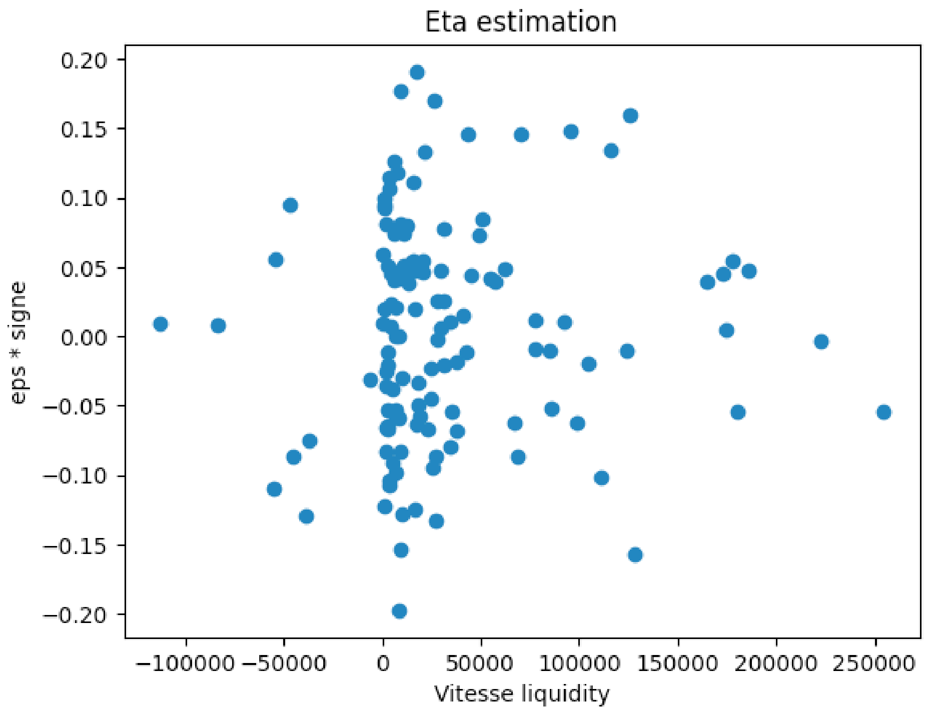


We have identified the presence of outliers in our dataset, which necessitates a cleaning process to ensure the robustness of our linear regression analysis:

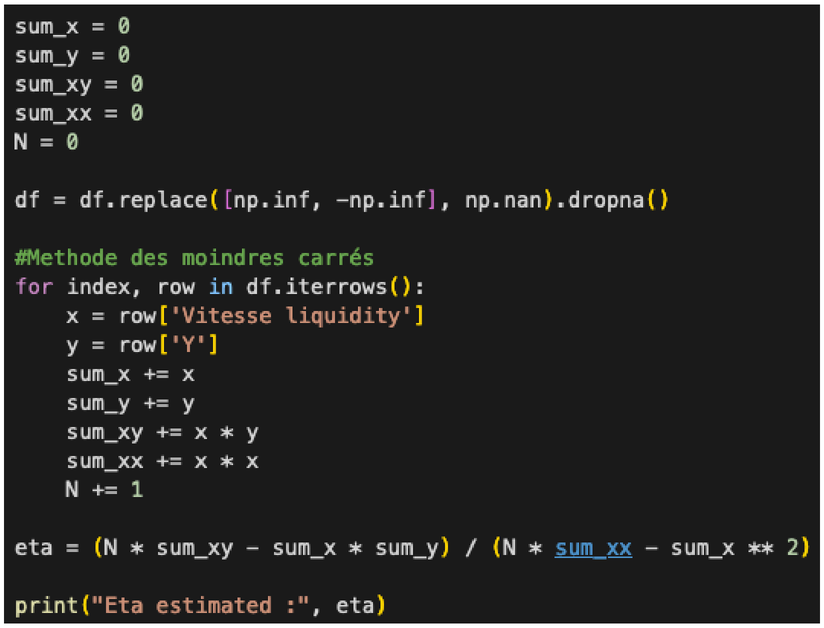


```
def rem_outlier(df, col):
    quantile1= df[col].quantile(0.25)
    quantile3 = df[col].quantile(0.75)
    I = quantile3 - quantile1

    l_bound = quantile1 - 1.5 * I
    u_bound = quantile3 + 1.5 * I

    condition = (df[col] >= l_bound) & (df[col] <= u_bound)

    return df[condition]
```

Our new graphic:

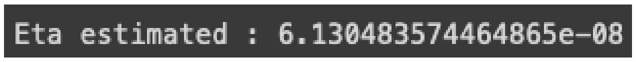


The scatter plot displays a wide dispersion of data points, indicating challenges in achieving a reliable linear regression fit.



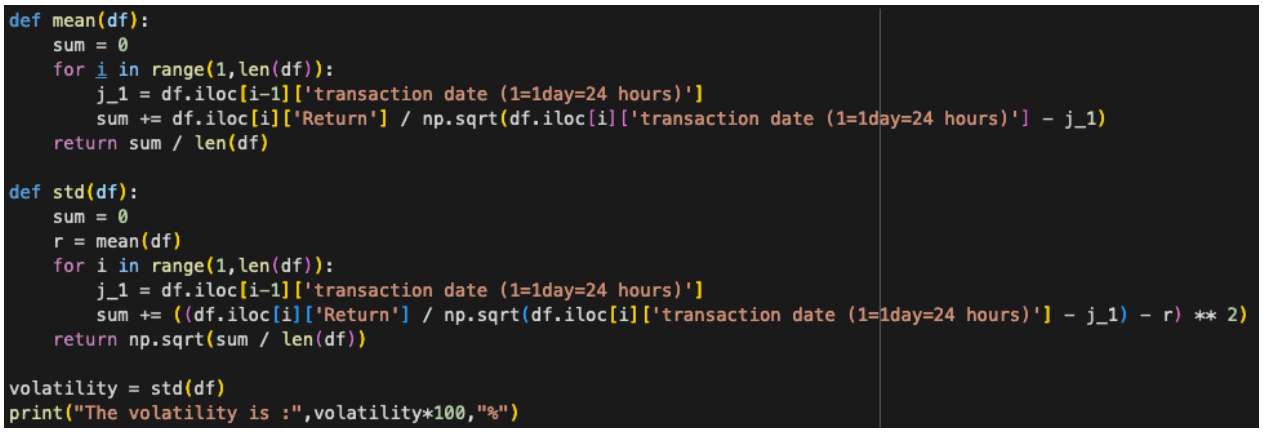
```
sum_x = 0
sum_y = 0
sum_xy = 0
sum_xx = 0
N = 0

df = df.replace([np.inf, -np.inf], np.nan).dropna()

#Methode des moindres carrés
for index, row in df.iterrows():
    x = row['Vitesse liquidity']
    y = row['Y']
    sum_x += x
    sum_y += y
    sum_xy += x * y
    sum_xx += x * x
    N += 1

eta = (N * sum_xy - sum_x * sum_y) / (N * sum_xx - sum_x ** 2)

print("Eta estimated :", eta)
```

Estimated η: $6.130483574464865 \times 10^{-8}$.



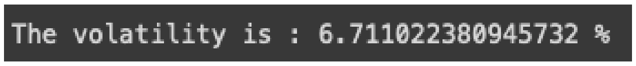
```
Eta estimated : 6.130483574464865e-08
```

After that we have to evaluate the volatility:



```
def mean(df):
    sum = 0
    for i in range(1,len(df)):
        j_1 = df.iloc[i-1]['transaction date (1=1day=24 hours)']
        sum += df.iloc[i]['Return'] / np.sqrt(df.iloc[i]['transaction date (1=1day=24 hours)'] - j_1)
    return sum / len(df)

def std(df):
    sum = 0
    r = mean(df)
    for i in range(1,len(df)):
        j_1 = df.iloc[i-1]['transaction date (1=1day=24 hours)']
        sum += ((df.iloc[i]['Return'] / np.sqrt(df.iloc[i]['transaction date (1=1day=24 hours)'] - j_1) - r) ** 2)
    return np.sqrt(sum / len(df))

volatility = std(df)
print("The volatility is :",volatility*100,"%")
```

And we found the volatility wich represents the second and final parameter required to fully specify the Almgren-Chriss model:

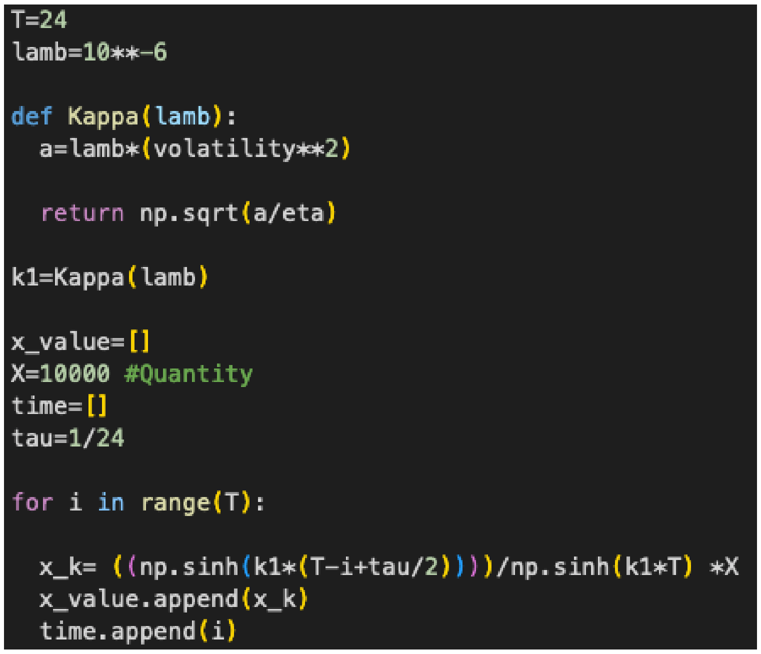


```
The volatility is : 6.711022380945732 %
```

Due to the wide dispersion of data points we can said that the model is not well specified for our dataset.

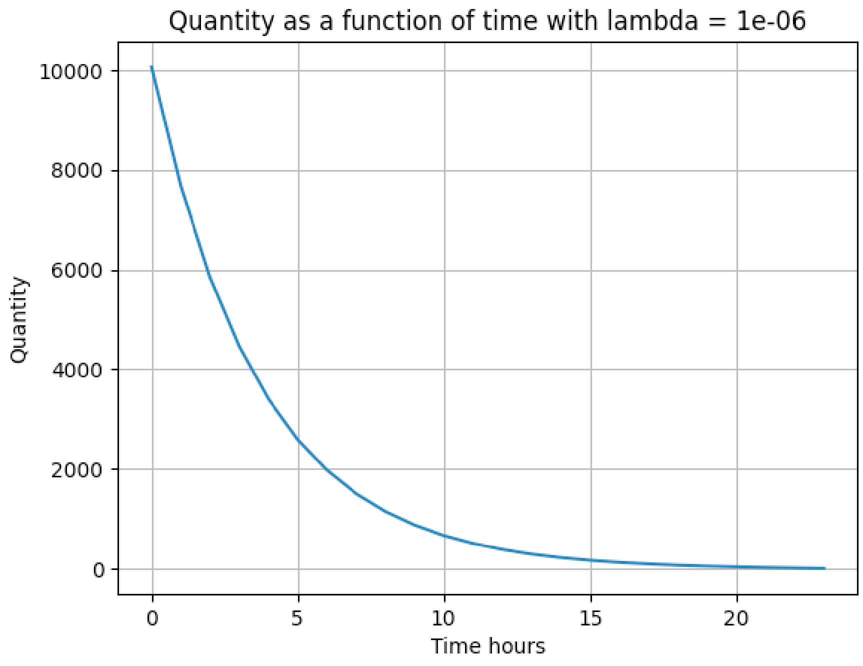
4.2 Part b)

The parameters have been estimated; now we need to calculate the remaining quantity X_k in the portfolio at each transaction for different levels of risk aversion, considering that only one transaction per hour is possible:

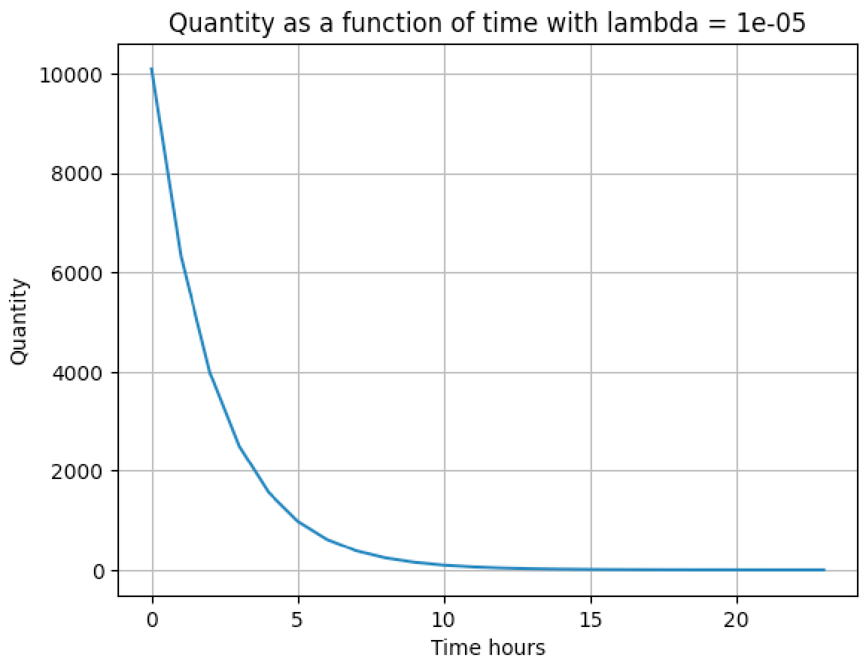


```
T=24
lamb=10**-6

def Kappa(lamb):
  a=lamb*(volatility**2)

  return np.sqrt(a/eta)

k1=Kappa(lamb)

x_value=[]
X=10000 #Quantity
time=[]
tau=1/24

for i in range(T):

  x_k= ((np.sinh(k1*(T-i+tau/2))))/np.sinh(k1*T) *X
  x_value.append(x_k)
  time.append(i)
```

So for $\lambda = 10^{-6}$, we have:



And for $\lambda = 10^{-5}$, we have:



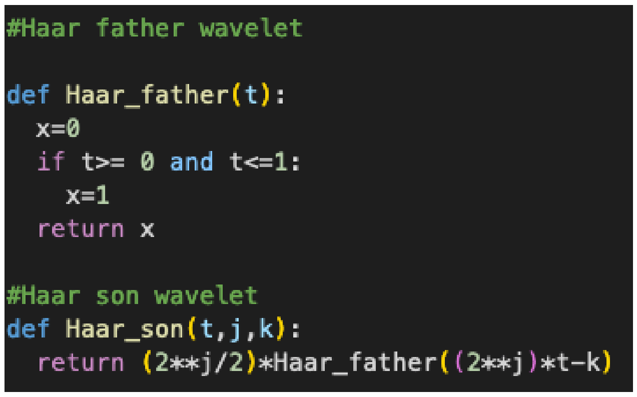
We can see from this graph that the quantity is liquidated quickly to counter market risk.

5 Question E (Q4 of TD5)

(You also have the notebook to see all the code.)

5.1 Part a)

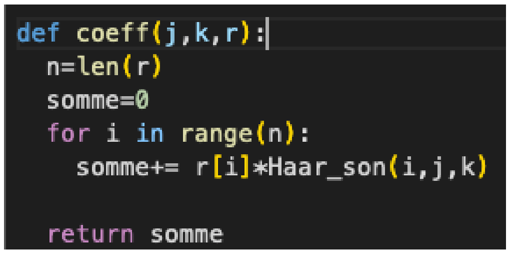
First, we implemted the Haar wavelet:



```
#Haar father wavelet

def Haar_father(t):
  x=0
  if t>= 0 and t<=1:
    x=1
  return x

#Haar son wavelet
def Haar_son(t,j,k):
  return (2**j/2)*Haar_father((2**j)*t-k)
```

After we implent the scale coefficient:

$$c_{j_0,k} = \int z(t)\,\phi_{j_0,k}(t)$$

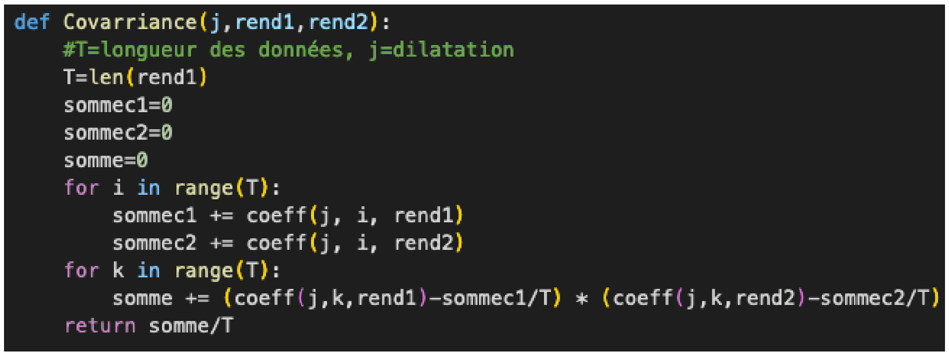


```
def coeff(j,k,r):
  n=len(r)
  somme=0
  for i in range(n):
    somme+= r[i]*Haar_son(i,j,k)

  return somme
```

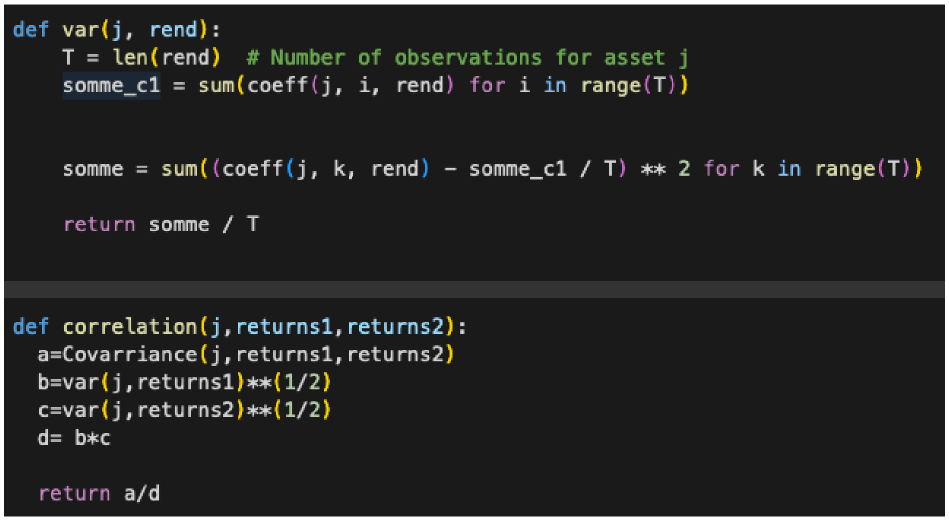
We also implemented the variance, covariance and correlation:

$$Cov(j) = \frac{1}{T}\sum_{k=1}^{T}\left(c_{j,k}^{1} - \frac{1}{T}\sum_{l=1}^{T} c_{j,l}^{1}\right)\left(c_{j,k}^{2} - \frac{1}{T}\sum_{l=1}^{T} c_{j,l}^{2}\right)$$

$$Var(j) = \frac{1}{T}\sum_{k=1}^{T}\left(c_{j,k} - \frac{1}{T}\sum_{i=1}^{T} c_{j,i}\right)^{2}$$



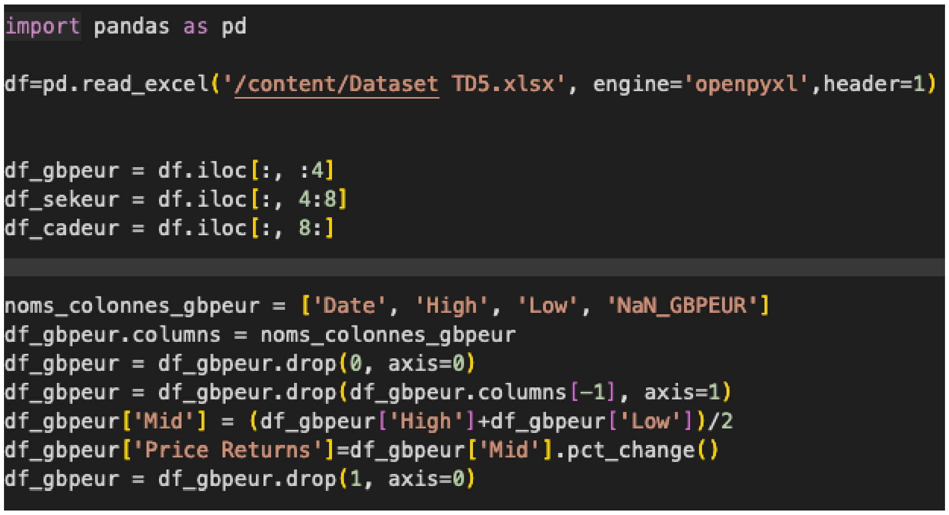
```
def Covarriance(j,rend1,rend2):
    #T=longueur des données, j=dilatation
    T=len(rend1)
    sommec1=0
    sommec2=0
    somme=0
    for i in range(T):
        sommec1 += coeff(j, i, rend1)
        sommec2 += coeff(j, i, rend2)
    for k in range(T):
        somme += (coeff(j,k,rend1)-sommec1/T) * (coeff(j,k,rend2)-sommec2/T)
    return somme/T
```



```
def var(j, rend):
    T = len(rend)  # Number of observations for asset j
    somme_c1 = sum(coeff(j, i, rend) for i in range(T))


    somme = sum((coeff(j, k, rend) - somme_c1 / T) ** 2 for k in range(T))

    return somme / T
```

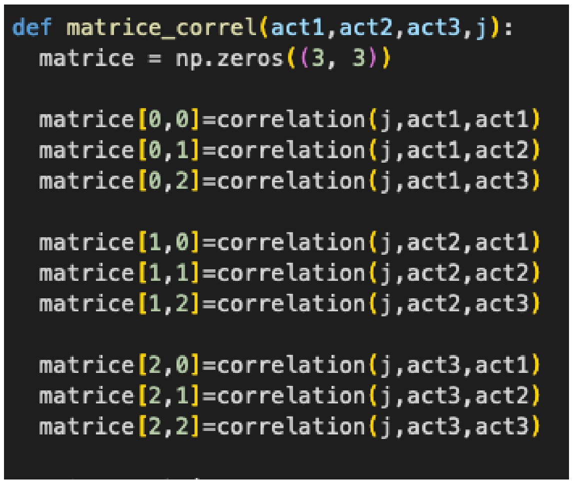
```
def correlation(j,returns1,returns2):
  a=Covarriance(j,returns1,returns2)
  b=var(j,returns1)**(1/2)
  c=var(j,returns2)**(1/2)
  d= b*c

  return a/d
```

This is the code we used to clean the dataset and we only take the 512 last data (due to $2^8 = 512$):



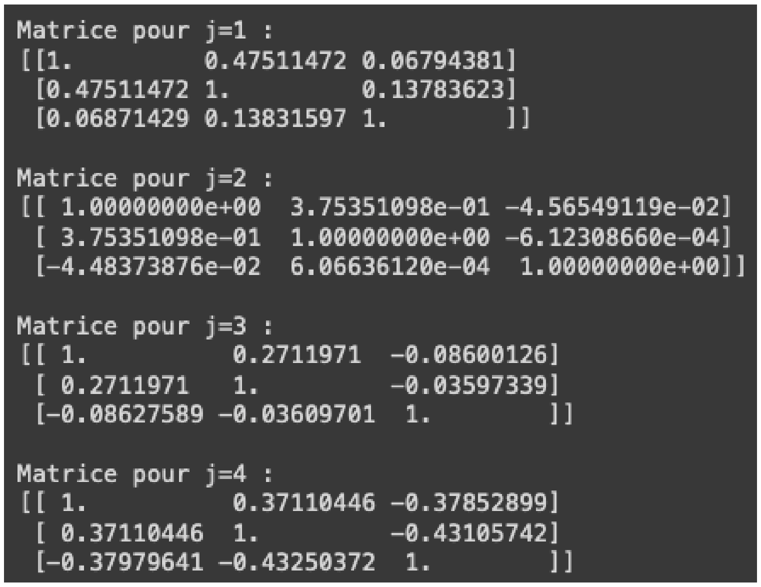
```
import pandas as pd

df=pd.read_excel('/content/Dataset TD5.xlsx', engine='openpyxl',header=1)


df_gbpeur = df.iloc[:, :4]
df_sekeur = df.iloc[:, 4:8]
df_cadeur = df.iloc[:, 8:]


noms_colonnes_gbpeur = ['Date', 'High', 'Low', 'NaN_GBPEUR']
df_gbpeur.columns = noms_colonnes_gbpeur
df_gbpeur = df_gbpeur.drop(0, axis=0)
df_gbpeur = df_gbpeur.drop(df_gbpeur.columns[-1], axis=1)
df_gbpeur['Mid'] = (df_gbpeur['High']+df_gbpeur['Low'])/2
df_gbpeur['Price Returns']=df_gbpeur['Mid'].pct_change()
df_gbpeur = df_gbpeur.drop(1, axis=0)
```

The code for the correlation matrix:



```
def matrice_correl(act1,act2,act3,j):
  matrice = np.zeros((3, 3))

  matrice[0,0]=correlation(j,act1,act1)
  matrice[0,1]=correlation(j,act1,act2)
  matrice[0,2]=correlation(j,act1,act3)

  matrice[1,0]=correlation(j,act2,act1)
  matrice[1,1]=correlation(j,act2,act2)
  matrice[1,2]=correlation(j,act2,act3)

  matrice[2,0]=correlation(j,act3,act1)
  matrice[2,1]=correlation(j,act3,act2)
  matrice[2,2]=correlation(j,act3,act3)
```

There is exemple of correlation matrice using different times scales ($j = 1$: 15 min):



```
Matrice pour j=1 :
[[1.         0.47511472 0.06794381]
 [0.47511472 1.         0.13783623]
 [0.06871429 0.13831597 1.        ]]

Matrice pour j=2 :
[[ 1.00000000e+00  3.75351098e-01 -4.56549119e-02]
 [ 3.75351098e-01  1.00000000e+00 -6.12308660e-04]
 [-4.48373876e-02  6.06636120e-04  1.00000000e+00]]

Matrice pour j=3 :
[[ 1.          0.2711971  -0.08600126]
 [ 0.2711971   1.         -0.03597339]
 [-0.08627589 -0.03609701  1.        ]]

Matrice pour j=4 :
[[ 1.          0.37110446 -0.37852899]
 [ 0.37110446  1.         -0.43105742]
 [-0.37979641 -0.43250372  1.        ]]
```

To determine the volatility vector with a scaling thanks to the Hurst exponents we have to calculate the hurst exponent and the annual volatility for each assets. We use this formula for the Hurst exponents:

$$\hat{H} = \frac{1}{2} \log_2 \left(\frac{M_2'}{M_2} \right)$$

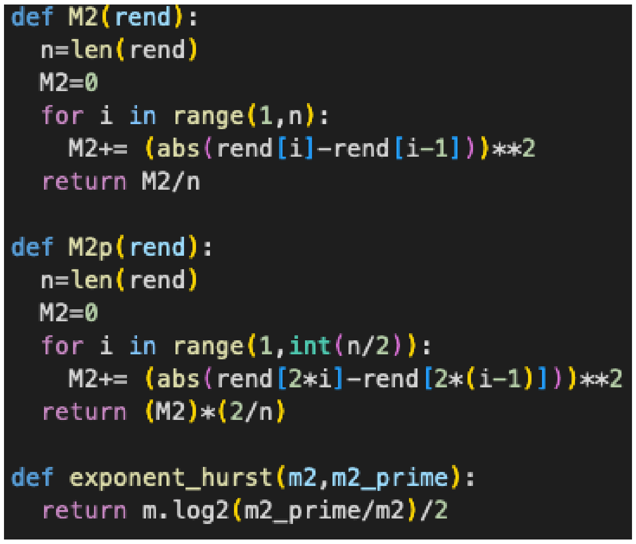
Where:

$$M_k = \frac{1}{NT} \sum_{i=1}^{NT} \left| X\left(\frac{i}{N}\right) - X\left(\frac{i-1}{N}\right) \right|^k$$

$$M_2' = \frac{2}{NT} \sum_{i=1}^{\frac{NT}{2}} \left| X\left(\frac{2i}{N}\right) - X\left(\frac{2(i-1)}{N}\right) \right|^2$$

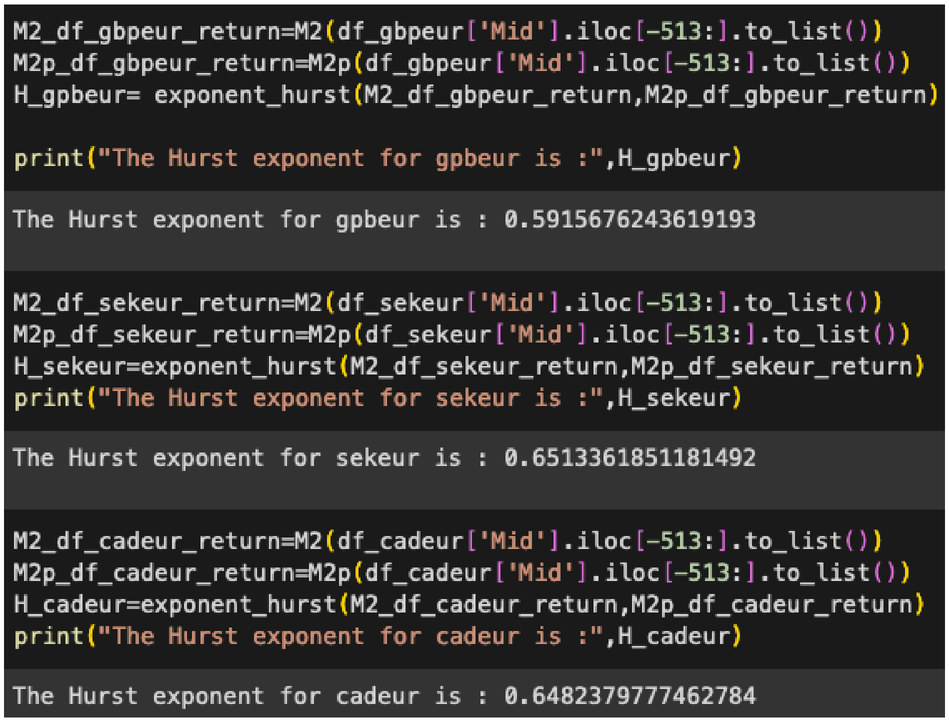
We take $k = 2$, $N = 1$, and $T =$ length of the data. To calculate annualized volatility, we use:

$$\sigma_{1\text{an}} = \sigma_{15\text{min}} \cdot (4 \cdot 24 \cdot 252)^H$$



```python
def M2(rend):
  n=len(rend)
  M2=0
  for i in range(1,n):
    M2+= (abs(rend[i]-rend[i-1]))**2
  return M2/n

def M2p(rend):
  n=len(rend)
  M2=0
  for i in range(1,int(n/2)):
    M2+= (abs(rend[2*i]-rend[2*(i-1)]))**2
  return (M2)*(2/n)

def exponent_hurst(m2,m2_prime):
  return m.log2(m2_prime/m2)/2
```

The following is the outcome for the Hurst exponents:



```python
M2_df_gbpeur_return=M2(df_gbpeur['Mid'].iloc[-513:].to_list())
M2p_df_gbpeur_return=M2p(df_gbpeur['Mid'].iloc[-513:].to_list())
H_gpbeur= exponent_hurst(M2_df_gbpeur_return,M2p_df_gbpeur_return)

print("The Hurst exponent for gpbeur is :",H_gpbeur)
```

```
The Hurst exponent for gpbeur is : 0.5915676243619193
```

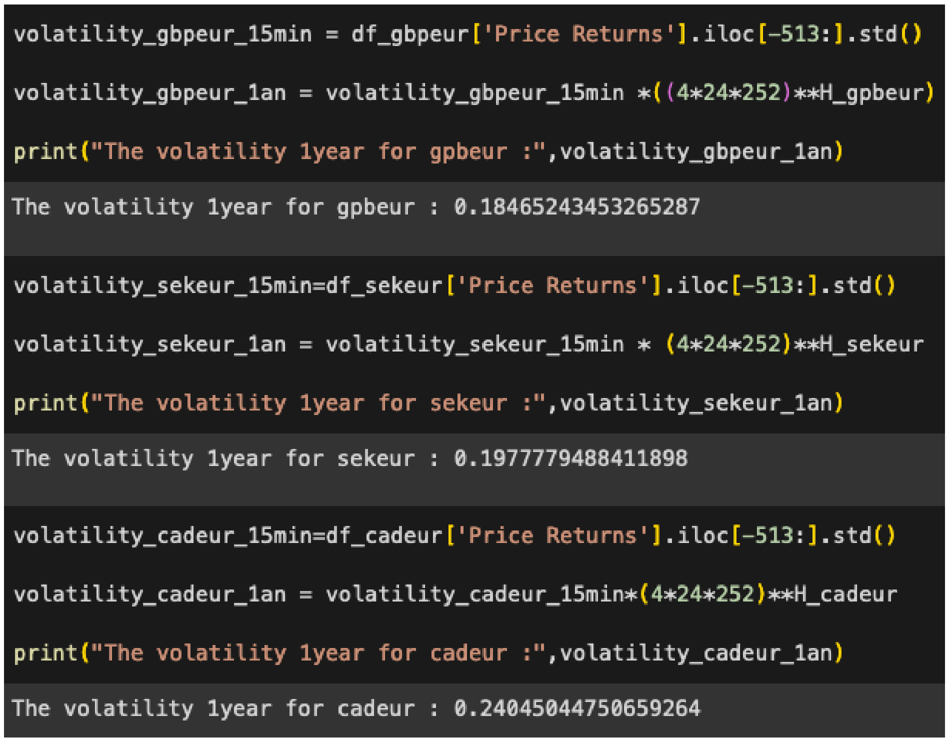
```python
M2_df_sekeur_return=M2(df_sekeur['Mid'].iloc[-513:].to_list())
M2p_df_sekeur_return=M2p(df_sekeur['Mid'].iloc[-513:].to_list())
H_sekeur=exponent_hurst(M2_df_sekeur_return,M2p_df_sekeur_return)
print("The Hurst exponent for sekeur is :",H_sekeur)
```

```
The Hurst exponent for sekeur is : 0.6513361851181492
```

```python
M2_df_cadeur_return=M2(df_cadeur['Mid'].iloc[-513:].to_list())
M2p_df_cadeur_return=M2p(df_cadeur['Mid'].iloc[-513:].to_list())
H_cadeur=exponent_hurst(M2_df_cadeur_return,M2p_df_cadeur_return)
print("The Hurst exponent for cadeur is :",H_cadeur)
```

```
The Hurst exponent for cadeur is : 0.6482379777462784
```

Here are the annual volatilities for each asset:



```
volatility_gbpeur_15min = df_gbpeur['Price Returns'].iloc[-513:].std()

volatility_gbpeur_1an = volatility_gbpeur_15min *((4*24*252)**H_gpbeur)

print("The volatility 1year for gpbeur :",volatility_gbpeur_1an)
```

```
The volatility 1year for gpbeur : 0.18465243453265287
```

```
volatility_sekeur_15min=df_sekeur['Price Returns'].iloc[-513:].std()

volatility_sekeur_1an = volatility_sekeur_15min * (4*24*252)**H_sekeur

print("The volatility 1year for sekeur :",volatility_sekeur_1an)
```

```
The volatility 1year for sekeur : 0.1977779488411898
```

```
volatility_cadeur_15min=df_cadeur['Price Returns'].iloc[-513:].std()

volatility_cadeur_1an = volatility_cadeur_15min*(4*24*252)**H_cadeur

print("The volatility 1year for cadeur :",volatility_cadeur_1an)
```

```
The volatility 1year for cadeur : 0.24045044750659264
```

We used this formula to compute the volatility for different times scales:

$$\mathrm{vol}^{(1)}(j) = \left(\frac{j}{256}\right)^{H^{(1)}} \cdot \mathrm{Vol}^{(1)}_{1\mathrm{year}}$$

$$\mathrm{vol}^{(2)}(j) = \left(\frac{j}{256}\right)^{H^{(2)}} \cdot \mathrm{Vol}^{(2)}_{1\mathrm{year}}$$

$$\mathrm{vol}^{(3)}(j) = \left(\frac{j}{256}\right)^{H^{(3)}} \cdot \mathrm{Vol}^{(3)}_{1\mathrm{year}}$$

So the volatility vector is:

$$\mathrm{Vol}(j) = \begin{bmatrix} \mathrm{vol}^1(j) \\ \mathrm{vol}^2(j) \\ \mathrm{vol}^3(j) \end{bmatrix}$$

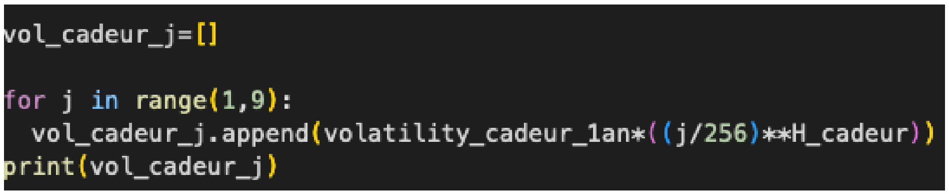
The relation to obtain the covariance matrix with the volatility vector and correlation matrix:

$$\mathrm{Cov}(j) = \mathrm{Vol}(j) \cdot \mathbf{I}_3 \cdot \mathrm{Corr}(j) \cdot \mathbf{I}_3 \cdot \mathrm{Vol}(j)$$

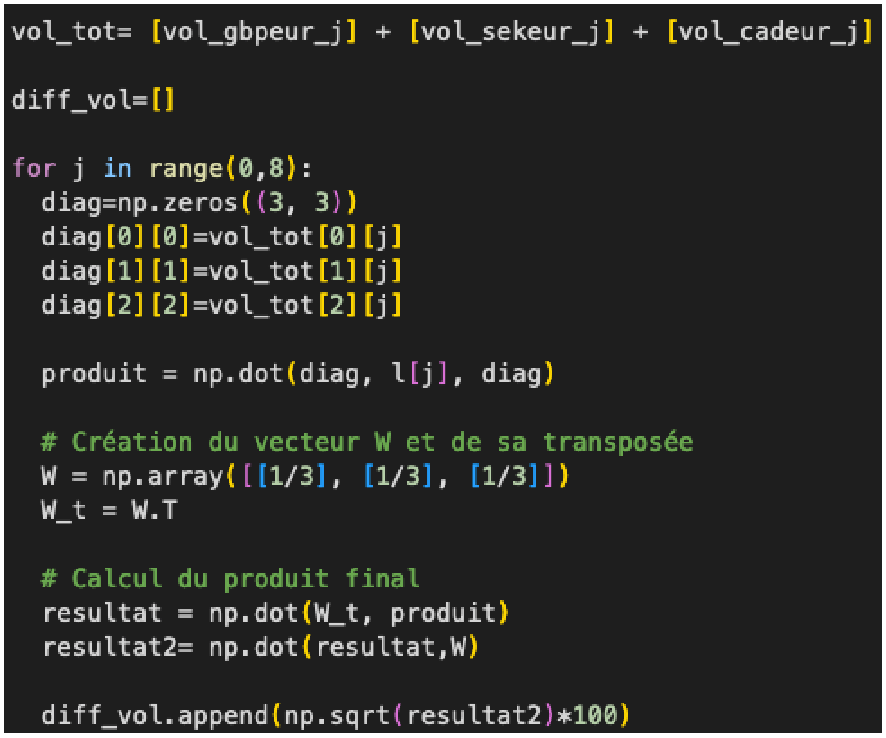
The volatility is given by:

$$\mathrm{Vol}^{\pi}(j) = \sqrt{\mathbf{W}^{T} \cdot \mathrm{Cov}(j) \cdot \mathbf{W}}$$

With $\mathbf{W}$ the weight matrix where $\mathbf{W} = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$.

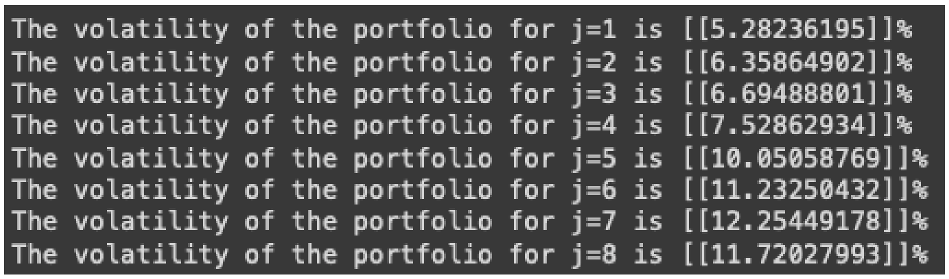


```
vol_cadeur_j=[]

for j in range(1,9):
  vol_cadeur_j.append(volatility_cadeur_1an*((j/256)**H_cadeur))
print(vol_cadeur_j)
```



```
vol_tot= [vol_gbpeur_j] + [vol_sekeur_j] + [vol_cadeur_j]

diff_vol=[]

for j in range(0,8):
  diag=np.zeros((3, 3))
  diag[0][0]=vol_tot[0][j]
  diag[1][1]=vol_tot[1][j]
  diag[2][2]=vol_tot[2][j]

  produit = np.dot(diag, l[j], diag)

  # Création du vecteur W et de sa transposée
  W = np.array([[1/3], [1/3], [1/3]])
  W_t = W.T

  # Calcul du produit final
  resultat = np.dot(W_t, produit)
  resultat2= np.dot(resultat,W)

  diff_vol.append(np.sqrt(resultat2)*100)
```

Finally we get:

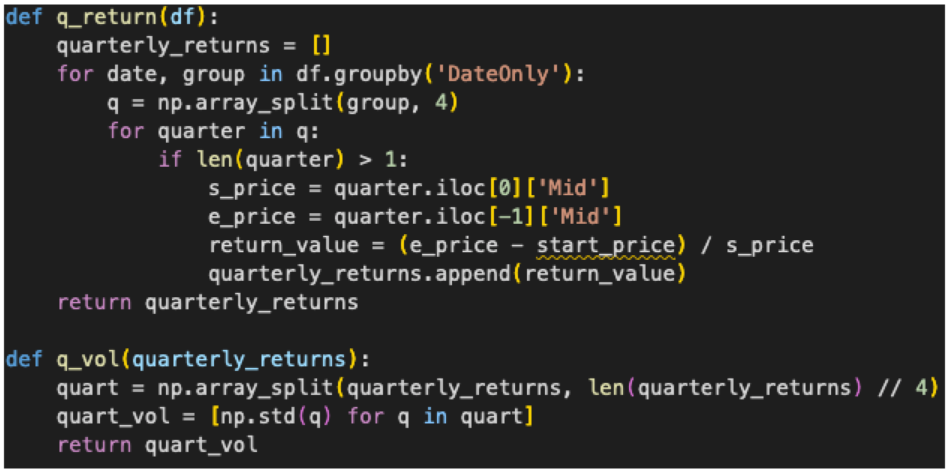


```
The volatility of the portfolio for j=1 is [[5.28236195]]%
The volatility of the portfolio for j=2 is [[6.35864902]]%
The volatility of the portfolio for j=3 is [[6.69488801]]%
The volatility of the portfolio for j=4 is [[7.52862934]]%
The volatility of the portfolio for j=5 is [[10.05058769]]%
The volatility of the portfolio for j=6 is [[11.23250432]]%
The volatility of the portfolio for j=7 is [[12.25449178]]%
The volatility of the portfolio for j=8 is [[11.72027993]]%
```

5.2 Part b)

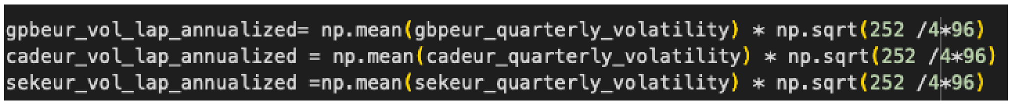
To accurately determine the portfolio's volatility from its price series, we will employ non-overlapping returns, calculated as quarter-daily returns, yielding H=4×96.

When considering a total analysis period T=512, the ratio H/T exceeds 68%. This substantial ratio validates our choice to use non-overlapping returns. Such a method is particularly advantageous in our context, as it effectively reduces potential biases introduced by autocorrelation, while fully leveraging the available data set for a more precise assessment of the portfolio's volatility.



```
def q_return(df):
    quarterly_returns = []
    for date, group in df.groupby('DateOnly'):
        q = np.array_split(group, 4)
        for quarter in q:
            if len(quarter) > 1:
                s_price = quarter.iloc[0]['Mid']
                e_price = quarter.iloc[-1]['Mid']
                return_value = (e_price - start_price) / s_price
                quarterly_returns.append(return_value)
    return quarterly_returns

def q_vol(quarterly_returns):
    quart = np.array_split(quarterly_returns, len(quarterly_returns) // 4)
    quart_vol = [np.std(q) for q in quart]
    return quart_vol
```

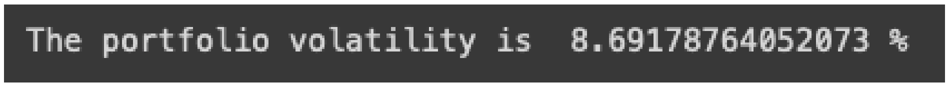
So we took the mean of the quarterly volatility and annualized it.



```
gpbeur_vol_lap_annualized= np.mean(gbpeur_quarterly_volatility) * np.sqrt(252 /4*96)
cadeur_vol_lap_annualized = np.mean(cadeur_quarterly_volatility) * np.sqrt(252 /4*96)
sekeur_vol_lap_annualized =np.mean(sekeur_quarterly_volatility) * np.sqrt(252 /4*96)
```

To compute the volatility of the portoflio we used :

$\text{Vol}^{\pi} = \sqrt{(w \cdot \text{Vol}^{1})^2 + (w \cdot \text{Vol}^{2})^2 + (w \cdot \text{Vol}^{3})^2}$
And we have :



```
The portfolio volatility is  8.69178764052073 %
```