Certificate in Quantitative Finance
Exam 1 Report
June 2023 Program

Lusha Li

August 6, 2023

Table of Content

1. Optimal Portfolio Allocation

Question 1 – Global Minimum Variance Portfolio

To solve for the optimal allocation w^*, we formulate:

$$\underset{w}{\operatorname{argmin}} \frac{1}{2} w' \Sigma w \quad s.t. w'1 = 1$$

We form the Lagrange function with one Lagrange multiplier γ :

$$L(w, \lambda) = \frac{1}{2} w' \Sigma w + \gamma(1 - w'1) \tag{1}$$

Next, we solve for the first order condition (FOC) by taking the derivative with respect

to the vector w:

$$\frac{\partial L}{\partial w}(w, \gamma) = \Sigma w - \gamma 1 = 0 \tag{2}$$

$$w^* = \gamma 1 \Sigma^{-1} \tag{3}$$

All we have to do now is to find the values for γ and then substitute it into (3). Remember we have constraints:

$$\frac{\partial L}{\partial \gamma}(w, \gamma) = 1 - w'1 = 0 \tag{4}$$

Insert the result (3) into (4), we get:

$$1 - \gamma 1 \Sigma^{-1} 1 = 0$$

$$\gamma = \frac{1}{1' \Sigma^{-1} 1} \tag{5}$$

Then we insert the (5) back to (3), we get:

$$w^* = \frac{1 \Sigma^{-1}}{1' \Sigma^{-1} 1} \tag{6}$$

Question 2 – Optimization for A Target Return m

To solve for the minimum variance portfolio under the target return m without risk-free

asset, we formulate:

$$\underset{w}{\operatorname{argmin}} \frac{1}{2} w' \Sigma w$$

Subject to:

$$w'1 = 1$$

$$w'\mu = m \tag{7}$$

Thus, we could formulate another Lagrange function with 2 optimization constraint – there is one additional budget equation:

$$L(w, \lambda, \gamma) = \frac{1}{2} w' \Sigma w + \gamma(1 - w'1) + \lambda(m - w'\mu) \tag{8}$$

Set the FOC to zero yields the optimal solution for the weight:

$$\frac{\partial L}{\partial w}(w, \lambda, \gamma) = \Sigma w - \gamma 1 - \lambda \mu = 0 \tag{9}$$

$$w^* = (\gamma 1 + \lambda \mu) \Sigma^{-1} \tag{10}$$

To find the multiplier γ and λ analytically, we need to substitute w^* into FOCs $\frac{\partial L}{\partial \gamma}(w, \lambda, \gamma) = 0$ and $\frac{\partial L}{\partial \lambda}(w, \lambda, \gamma) = 0$:

$$(\gamma 1 + \lambda \mu) \Sigma^{-1} 1' = 1 \tag{11}$$

$$(\gamma 1 + \lambda \mu) \Sigma^{-1} \mu' = m \tag{12}$$

We then define the following scalars:

$$A = 1' \Sigma^{-1} 1$$

$$B = \mu' \Sigma^{-1} 1$$

$$C = \mu' \Sigma^{-1} \mu$$

So we can get:

$$\lambda = \frac{Am - B}{AC - B^2}$$

$$\gamma = \frac{C - Bm}{AC - B^2}$$

Insert these into (10) we get:

$$w^* = \frac{1}{AC - B^2} \Sigma^{-1} [(\mathrm{A}\mu - B1)m + (C1 - B\mu)] \tag{13}$$

Then we insert m=7% into above formula to calculate w^* and portfolio risk σ_Π. First we construct the variance-covariance matrix Σ from the correlation matrix:

$$\Sigma = SRS = \begin{pmatrix} 0.0049 & 0.00784 & 0.00525 & 0.00651 \\ 0.00784 & 0.0784 & 0.0189 & 0.036456 \\ 0.00525 & 0.0189 & 0.0625 & 0.03875 \\ 0.00651 & 0.036456 & 0.03875 & 0.0961 \end{pmatrix}$$

Python code I used to calculated it is included in the Appendix 3.1 Covariance Matrix session.

Then we can calculate the w^* and σ_Π using the code included in the Appendix 3.2 Optimal Weight and Portfolio Risk session:

Correlation Matrix	x1
$\boldsymbol{\lambda}$	0.135915
$\boldsymbol{\gamma}$	-0.001817
$\boldsymbol{W_A}$	75.8514%
$\boldsymbol{W_B}$	-11.6851%
$\boldsymbol{W_C}$	12.2575%
$\boldsymbol{W_D}$	23.5762%
$\boldsymbol{\sigma_\Pi}$	0.011775

Similarly, we can repeat the same process to get below computational results for the targeted return m = 7% for other two level of correlation.

$$\Sigma_{1.3} = SRS = \begin{pmatrix} 0.004851 & 0.010192 & 0.006825 & 0.008463 \\ 0.010192 & 0.077616 & 0.02457 & 0.0473928 \\ 0.006825 & 0.02457 & 0.061875 & 0.050375 \\ 0.008463 & 0.0473928 & 0.050375 & 0.095139 \end{pmatrix}$$

$$\Sigma_{1.8} = SRS = \begin{pmatrix} 0.004851 & 0.014112 & 0.00945 & 0.011718 \\ 0.014112 & 0.077616 & 0.03402 & 0.0656208 \\ 0.00945 & 0.03402 & 0.061875 & 0.06975 \\ 0.011718 & 0.0656208 & 0.06975 & 0.095139 \end{pmatrix}$$

Summary table is as follow. Detailed calculation is included in Appendix "Calculation for x1.3 and x1.8 Correlation Matrix".

Correlation Matrix	x1	x1.3	x1.8
$\boldsymbol{\lambda}$	0.135915	0.133255	-0.018567
$\boldsymbol{\gamma}$	-0.001817	-0.001792	0.001480
$\boldsymbol{W_A}$	75.8514%	87.4711%	171.0530%
$\boldsymbol{W_B}$	-11.6851%	21.0721%	-68.2898%
$\boldsymbol{W_C}$	12.2575%	4.1534%	-97.6505%

$\boldsymbol{W_D}$	23.5762%	29.4477%	94.8874%
$\boldsymbol{\sigma_\Pi}$	0.108511	0.107396	NA

2. Understanding Risk

Question 3

(a.) Sharpe Ratio formula and main parameter scaled with time

This is the Sharpe ratio of a portfolio C:

$$Sharpe\ Ratio_c = \frac{\mu_c - R}{\sigma_c}$$

The numerator is the difference over time between realized/expected returns and a benchmark such as the risk-free rate of return. Its denominator is the standard deviation of returns over the same period of time. It represents the excess return per unit of total risk taken, so be used to measure the portfolio efficiency – higher Sharpe ratio means more efficient.

For example, if we use monthly return to calculate 1-year Sharpe ratio then the numerator is the average of the 12 monthly return differentials:

$$numerator = \frac{1}{n}\sum_{i=1}^{n}(R_i - RF_i) = \overline{R}^e$$

And the denominator is the monthly standard deviation of excess returns:

$$denominator = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(R_i - RF_i - \overline{R}^e)^2}$$

The 1-year Sharpe Ratio is the product of the 12 monthly Sharpe Ratio and $\sqrt{12}$. This is equivalent to multiplying the numerator by 12 and the denominator by $\sqrt{12}$:

$$SR_{1-year} = SR_M \times \sqrt{12}$$

Generalize this formula to n months:

$$SR_{n-month} = SR_M \times \sqrt{n}$$

We can see the Sharpe ratio is scaled with time ($\sqrt{n}$).

(b.) Compute SR based on Annualised SR

$$SR_A = 0.35$$

If we annualize the Sharpe ratio based on daily returns, then we usually multiply the daily Sharpe ratio by $\sqrt{252}$, representing the number of trading days in a year:

$$SR_A = SR_D \times \sqrt{252}$$

So:

$$SR_D = SR_A \div \sqrt{252}$$

$$SR_D = SR_A \div \sqrt{252} = 0.022048$$

Similarly for monthly:

$$SR_A = SR_M \times \sqrt{12}$$

$$SR_M = SR_A \div \sqrt{12} = 0.101036$$

For quarterly:

$$SR_A = SR_Q \times \sqrt{4}$$

$$SR_Q = SR_A \div \sqrt{4} = 0.175$$

(c.) Convert Sharpe Ratio into Loss Probability

To convert Sharpe Ratio into Loss Probability, several assumptions are needed:

- ✓ portfolio returns are normally distributed
- ✓ returns are independent

We denote the returns of the portfolio by R. Assuming that R is Gaussian with mean $\sigma = E[R]$ and $\sigma = \sqrt{Var(R)}$:

$$Sharpe\ Ratio_R = \frac{E[R]}{\sqrt{Var(R)}} = \frac{\mu}{\sigma}$$

To calculate the probability of the portfolio losing value we can write the formula as:

$$Pr(R < 0) = \Pr\left(\frac{R - \mu\sigma}{\sigma} < -\frac{\mu}{\sigma}\right) = \Pr(x < -SR)$$

where x is a standard Normal random variable: $x \sim N(0,1)$.

For daily SR_D:

$$Pr(R < 0) = \Pr(x < -0.022048)$$

Use Excel formula:

$$Pr(R < 0) = \text{NORM.S.DIST}(-0.022048, TRUE) = 0.491205$$

For monthly SR_M:

$$Pr(R < 0) = \Pr(x < -0.101036)$$

$$Pr(R < 0) = \text{NORM.S.DIST}(-0.101036, TRUE) = 0.459761$$

For quarterly SR_Q:

$$Pr(R < 0) = \Pr(x < -0.175)$$

$$Pr(R < 0) = \text{NORM.S.DIST}(-0.175, TRUE) = 0.430540$$

Question 4 – 700 Simulations and Efficient Frontier

To get random numbers for weights, I use the np.random.random() function. But remember that the sum of weights must be 1 and we need to use formula $w'1 = 1$ to calculated the 4th weight:

```
# Generate 3 weight allocations and compute the 4th.
weights_3 = np.random.random(3)
weights_4 = 1-np.sum(weights_3)
weights = np.hstack((weights_3,weights_4))
```

Formula used to calculate the return and standard deviation:

$$\mu_{\Pi} = w'\mu$$

$$\sigma_{\Pi} = \sqrt{w'\Sigma w}$$

Corresponding Python code is below:

```
# Compute mu and sigma
portfolio_rtn = np.sum(rtn * weights)
portfolio_vol = sqrt(weights.T*cov*weights)
```

I also want to mark the optimal portfolio (max Sharpe ratio) in the chart, so I also include Sharpe ratio in the iteration. In each iteration, the loop considers different weights for assets and calculates the return and volatility of that particular portfolio combination.
Corresponding Python code is below:

```python
num_portfolios = 700

# Initialize arrays to store portfolio statistics
portfolio_rtns = np.zeros(num_portfolios)
portfolio_vols = np.zeros(num_portfolios)
sharpe_ratios = np.zeros(num_portfolios)

# Generate random portfolio allocations and compute statistics
for i in range(num_portfolios):
    # Generate 3 weight allocations and compute the 4th.
    weights_3 = np.random.random(3)
    weights_4 = 1-np.sum(weights_3)
    weights = mat(np.hstack((weights_3,weights_4))).T

    # Compute mu and sigma
    portfolio_rtn = np.sum(rtn * weights)
    portfolio_vol = sqrt(weights.T*cov*weights)

    # Calculate Sharpe ratio
    sharpe_ratio = portfolio_rtn / portfolio_vol

    # Store results
    portfolio_rtns[i] = portfolio_rtn
    portfolio_vols[i] = portfolio_vol
    sharpe_ratios[i] = sharpe_ratio

# Create a DataFrame to store portfolio returns, volatilities, and Sharpe ratios
portfolio_df = pd.DataFrame({
    "Returns": portfolio_rtns,
    "Volatilities": portfolio_vols,
    "Sharpe Ratios": sharpe_ratios
})

# Results
portfolio_df
```
✓ 0.0s Python

The final output is a data frame called "portfolio_df", which contains 3 series: Returns, Volatilities and Sharpe Ratios:

	Returns	Volatilities	Sharpe Ratios
0	0.142732	0.208956	0.683073
1	-0.015803	0.200626	-0.078770
2	0.115892	0.187705	0.617413
3	0.030010	0.267506	0.112185
4	-0.007187	0.270409	-0.026579
...	...	...	...
695	-0.030488	0.214437	-0.142179
696	0.055789	0.230399	0.242141
697	0.014223	0.240072	0.059244
698	-0.039554	0.231586	-0.170796
699	0.028827	0.129949	0.221832

700 rows × 3 columns

First I plot the 700 simulated points in a scatter chart:

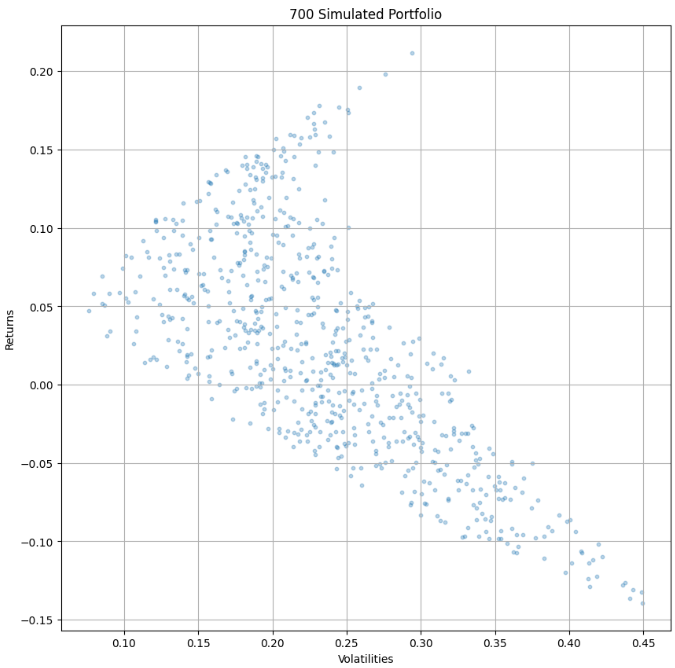


Figure 1 Scatter chart for 700 Simulated Portfolio

Corresponding Python code is below:

```
# Plot efficient frontier
portfolio_df.plot.scatter(x='Volatilities', y='Returns',
    marker='o', s=10, alpha=0.3, grid=True, figsize=[10,10],
    title="700 Simulated Portfolio")
```

✓ 0.2s Python

This is the efficient frontier I am expected to see - it is a graph with 'returns' on the Y-axis and 'volatility' on the X-axis and can show you all combination of weights that will give you all possible combinations. It can tell us the maximum return we can get for a set level of volatility, or put it on the other word, the volatility that we need to accept for certain level of returns.

Each point on the left edge represents an optimal portfolio of stocks that maximizes the returns for any given level of risk. The points in the interior are sub-optimal for a given risk level, meaning that for every interior point, there is always another combination that can offer a higher return for the same risk.

Then I want to calculate the Minimum Volatility portfolio (left most point) and Maximum Sharpe Ratio portfolio using below code:

```python
# to get the minimum volatility (left most point)
min_vol_port = portfolio_df.iloc[portfolio_df['Volatilities'].idxmin()]
# idxmin() gives us the minimum value in the column specified.
min_vol_port
```
✓ 0.0s Python

```
Returns           0.044040
Volatilities      0.073524
Sharpe Ratios     0.598985
Name: 561, dtype: float64
```

```python
# Finding the optimal portfolio
optimal_risky_port = portfolio_df.iloc[(portfolio_df['Sharpe Ratios']).idxmax()]
optimal_risky_port
```
✓ 0.0s Python

```
Returns           0.084668
Volatilities      0.096957
Sharpe Ratios     0.873258
Name: 126, dtype: float64
```

Then plot them in the Figure 1 to get new Figure 2:

```python
# Plotting optimal portfolio
plt.subplots(figsize=(10, 10))
plt.scatter(portfolio_df['Volatilities'], portfolio_df['Returns'],marker='o', s=10, alpha=0.3
plt.scatter(min_vol_port[1], min_vol_port[0], color='r', marker='*', s=500)
plt.scatter(optimal_risky_port[1], optimal_risky_port[0], color='g', marker='*', s=500)
```
✓ 0.1s Python

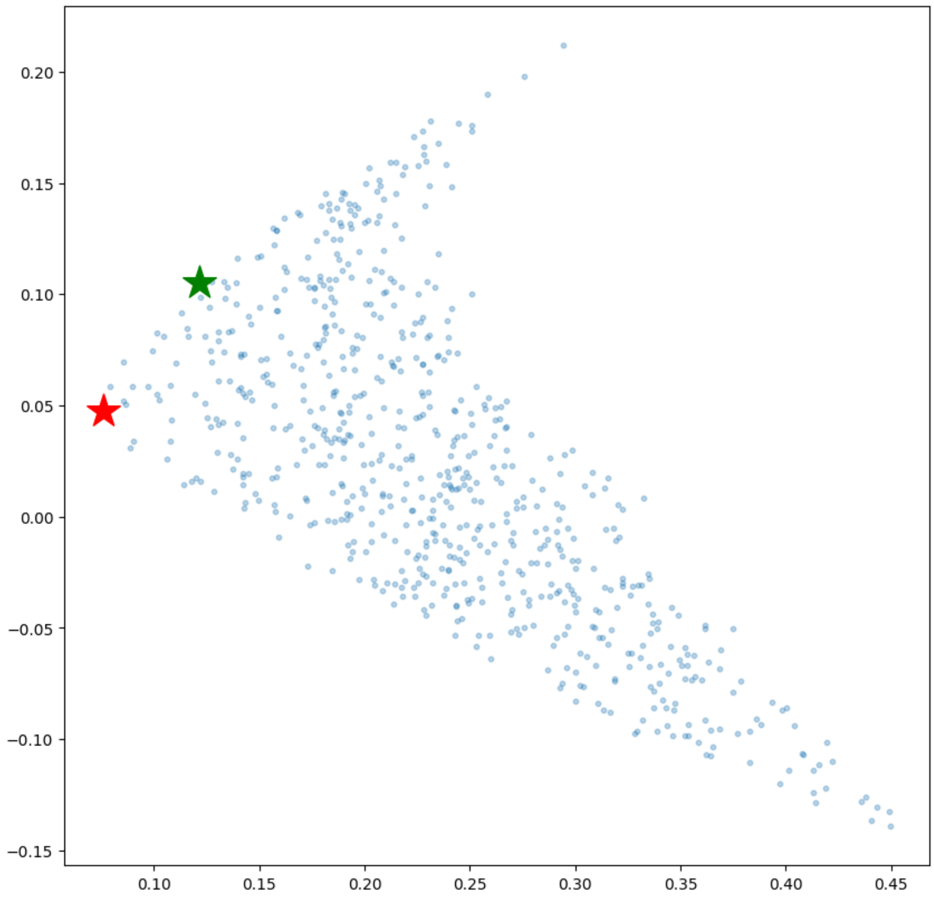


Figure 2 Scatter chart for 700 Simulated Portfolio with min volatility portfolio and max Sharpe Ratio portfolio marked

The red star denotes the most efficient portfolio with minimum volatility – can see that the return on this portfolio is actually pretty low. So next question is how to find the best tradeoff between risk and return to find the optimal portfolio?

We can use Sharpe Ratio to find it - the optimal risky portfolio is the one with the highest Sharpe ratio. In my case, the optimal portfolio is demoted by the green star.

We compare the output for "min_vol_port" and "optimal_risky_port" in the previous code and can see that while the difference in risk between minimum volatility portfolio (0.073524) and optimal risky portfolio (0.096957) is just 2.34%, the difference in returns is almost double (4.06%).

Question 5 – VaR Breaches using Sample Standard Deviation

(a.) The count and percentage of VaR breaches

1-day log return is calculated as:

$$ln(S_{t+1}/\ S_t)$$

Corresponding Python code:

```python
#1D realised log return
data['1D_log_rtn'] = ''

for i in range(0, len(data['Closing Price']), 1):
    if i < 1:
        data['1D_log_rtn'][i] = 0
        continue
    else:
        rtn = data['Closing Price'][i]/data['Closing Price'][i - 1]
        log_rtn_1D = np.log(rtn)
        data['1D_log_rtn'][i] = log_rtn_1D
```
✓ 0.2s Python

10-day forward realised return is calculated as:

$$ln(S_{t+10}/\ S_t)$$

Corresponding Python code:

```python
0D forward realised log return
ta['10D_forward_log_rtn'] = ''

r i in range(0, len(data['Closing Price']), 1):
  if i > len(data['Closing Price']) - 11:
      data['10D_forward_log_rtn'][i] = 0
      continue
  else:
      rtn_10D = data['Closing Price'][i]/data['Closing Price'][i + 10] #to get negative values
      forward_log_rtn_10D = np.log(rtn_10D)
      data['10D_forward_log_rtn'][i] = forward_log_rtn_10D
```
✓ 0.2s Python

Then we can compute the rolling standard deviation using 21 daily log returns, corresponding Python code:

```python
# Compute the rolling standard deviation from 21 daily returns
data['sigma_21D'] = ''

for i in range(0, len(data['Closing Price']), 1):
    if i < 21:
        data['sigma_21D'][i] = 0
        continue
    else:
        init = i - 20
        std_dev = np.std(data['1D_log_rtn'].iloc[init:i])
        data['sigma_21D'][i] = std_dev
```
0.4s Python

This is essentially “1-day standard deviation”, next to make projection we will use below formula to calculate the 10-day standard deviation:

$$\sigma_{10-day} = \sqrt{10\ \times\ \sigma_{1-day}^2}$$

Corresponding Python code:

```python
# Compute 10-day sigma for projection
data['sigma_10D'] = np.sqrt(10) * data['sigma_21D']
data.iloc[0:30]
```

Then we can calculate the VaR using below formula:

$$VaR_{10-day} = Factor \times \sigma_{1-day} \times \sqrt{10} = Factor * \sigma_{10-day}$$

```python
# Calculate 10-day VaR
data['VAR_10D'] = ''
item = 1

for i in range(0, len(data), 1):
    if data['sigma_21D'][i] == '':
        data['VAR_10D'][i] = 0
        continue
    else:
        if item == 11 or item == 1:
            var = - factor * data['sigma_21D'][i] * np.sqrt(10)
            item = 1
        else:
            var = var
        data['VAR_10D'][i] = var
        item = item + 1
```

We define the breaches as:

$$r_{10-day,t+10} < VaR_{10-day,t}$$

Corresponding Python code:

```python
# Compute the VaR breaches
breaches = 0
consecutive_breaches = 0
item = 0
breach_list = []

for i in range(0, len(data), 1):
    if data['10D_forward_log_rtn'][i] == 0 or data['VAR_10D'][i] == 0:
        breach_list.append([data.index[i], 0])
        continue
    else:
        item += 1

        if data['10D_forward_log_rtn'][i] < data['VAR_10D'][i]:
            breach_list.append([data.index[i], data['10D_forward_log_rtn'][i]])
            breaches += 1
        else:
            breach_list.append([data.index[i], 0])

    if data['10D_forward_log_rtn'][i - 1] == 0 or data['VAR_10D'][i - 1] == 0:
        continue
    else:
        if data['10D_forward_log_rtn'][i - 1] < data['VAR_10D'][i - 1] and data['10D_forward_log_r
            consecutive_breaches += 1

breach_list = pd.DataFrame(breach_list)
breach_list.columns = ['Date', 'breaches']
breach_list.set_index('Date', inplace=True)
```

Hence, we can calculate the count and percentage of breaches are 21 and 1.57% respectively. Corresponding Python code:

```
# Number of breaches
breaches
✓ 0.0s                                                    Python

21

# Percentage of breaches
Percentage = (breaches/item) * 100
Percentage
✓ 0.0s                                                    Python

1.5671641791044775
```

(b.) The count of consecutive VaR breaches

Use below code, we can calculate the count of consecutive VaR breaches is 10.

```
if data['10D_forward_log_rtn'][i - 1] == 0 or data['VAR_10D'][i - 1] == 0:
    continue
else:
    if data['10D_forward_log_rtn'][i - 1] < data['VAR_10D'][i - 1] and data['10D_forward_log_rtn'][i] < data['VAR_10D'][i]:
        consecutive_breaches += 1
```

```
# Number of consecutive breaches
consecutive_breaches
✓ 0.0s                                                    Python

10
```

Method	Sample Standard Deviation method
Number of breaches	21
Percentage of breaches	1.57%
Number of consecutive breaches	10

Table 1 VaR Breaches Summary using Sample Standard Deviation Method

(c.) Plot

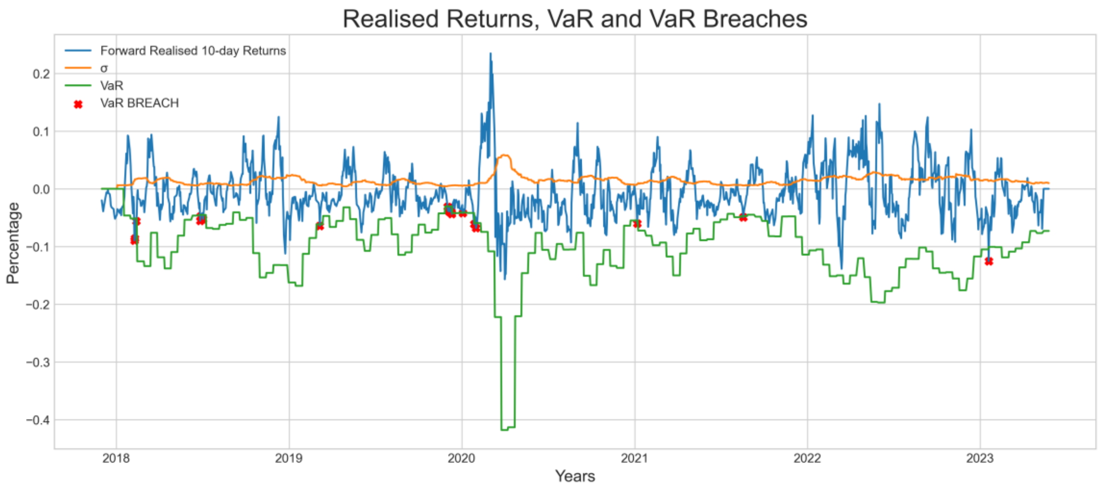


Figure 3 Realised Returns, VaR and VaR Breaches using Sample Standard Deviation method

Corresponding Python code:

```python
# Add breaches into the chart
plt.figure(dpi=300, figsize=(15,6))
plt.plot(data.index, data['10D_forward_log_rtn'], label='Forward Realised 10-day Returns')
plt.plot(data['sigma_21D'], label='σ')
plt.plot(data['VAR_10D'], label='VaR')

# plt.bar(data.index, height=breach_list['breaches'], label='VaR BREACH', color='red')
plt.scatter(data.index[breach_list['breaches']!=0],breach_list[breach_list['breaches']!=0], label='VaR BREACH',marker='X', color = 'red')

plt.title('Realised Returns, VaR and VaR Breaches', fontsize=20)
plt.xlabel('Years', fontsize=13)
plt.ylabel('Percentage', fontsize=13)
plt.legend()
plt.show()
```

✓ 0.4s

(d.) COVID pandemic VaR breach sequence

From Figure 1 we can see3 breaches happened in the early 2020, just before the period when the volatility of realised returns became very high and VaR became very low.

With below code, I am trying to see the exact dates of the breaches:

```python
# Convert the date to datetime64
data['Date'] = data.index
data['Date'] = pd.to_datetime(data['Date'], format='%Y-%m-%d')

# Filter data between two dates
filtered_df = data.loc[(data['Date'] >= '2020-01-01')
                        & (data['Date'] < '2020-04-01')
                        & (data['10D_forward_log_rtn'] < data['VAR_10D'])]

# Display
filtered_df
```

✓ 0.0s Python

	Closing Price	1D_log_rtn	10D_forward_log_rtn	sigma_21D	sigma_10D	VAR_10D	Date
Date							
2020-01-03	8793.900391	-0.008867	-0.042286	0.00525	0.016603	-0.041266	2020-01-03
2020-01-27	8952.179688	-0.020924	-0.061166	0.006623	0.020943	-0.041823	2020-01-27
2020-01-31	8991.509766	-0.015952	-0.067936	0.008515	0.026926	-0.059474	2020-01-31

As we can see from the output table – all 3 breaches happened before the Covid pandemic period and no breach happened in February or March. Taking a look back on the VaR calculation we could see it was calculated based on the standard deviation and scaled by the normalization factor. This being said, when an event like the Covid pandemic happens, the market becomes more volatile (can see the amber line in Figure 1 becomes higher) and hence the VaR. The higher the VaR, the lower chance of breaches. With the scale factor, the VaR is magnified by more than one time (2.33 *) of volatility, which makes the breach even less likely to happen.

Question 6 – VaR Breaches using $\boldsymbol{EWMA}_{\boldsymbol{\sigma^2_{t+1}}}$

(a.)-(b.) The count and percentage of VaR breaches and Consecutive breaches

In this case, I use the average variance as the initial variance estimate:

```python
# Average Variance
Avg_Var = np.mean(data['Squared Return'])
Avg_Var
```

✓ 0.0s Python

```
0.0002691865041913592
```

Then use below formula to estimate the rest variance:

$$\sigma^2_{t+1|t} = \lambda\sigma^2_{t|t-1} + (1-\lambda)r_t^2$$

```python
# Compute the variance estimate using EMWA method
λ = 0.72
data['Variance Estimate'] = ''

for i in range(0, len(data['Closing Price']), 1):
    if i <= 1:
        data['Variance Estimate'][i] = Avg_Var
        continue
    else:
        variance = (λ * data['Variance Estimate'][i-1]) + ((1-λ) * data['Squared Return'][i-1]
        data['Variance Estimate'][i] = variance
```

✓ 0.3s Python

This will give me a series of daily variance estimate. Using them I can calculate the daily standard deviation "sigma_1D":

```python
# Compute 1-day sigma for projection
data['sigma_1D'] = pow(data['Variance Estimate'],0.5)
data.head()
```

✓ 0.0s Python

Similarly, I use $\sigma_{10-day} = \sqrt{10 \times \sigma_{1-day}^2}$ to calculate the 10-day standard deviation "sigma_10D":

```python
# Compute 10-day sigma for projection
data['sigma_10D'] = np.sqrt(10) * data['sigma_1D']
data.head()
```

✓ 0.0s Python

With new VaR values, I can calculate the new VaR breaches and consecutive breaches – in this case they are:

Method	EWMA method
Number of breaches	37
Percentage of breaches	2.70%
Number of consecutive breaches	21

Table 2 VaR Breaches Summary using EWMA method

(c.) Plot using EWMA method

With updated standard deviation values, I can calculate updated VaR value and plot them. Below is the new chart generated:

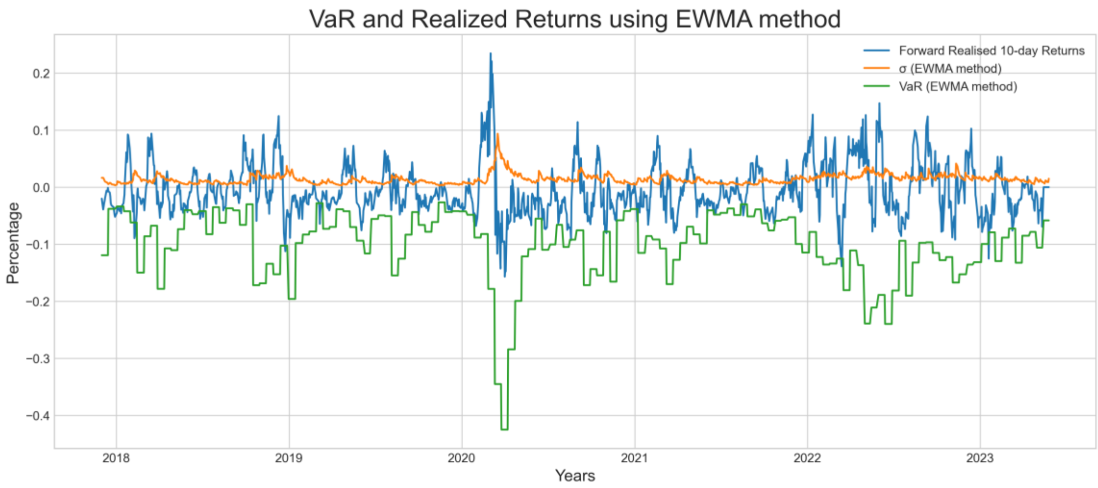


Figure 4 Realised Returns and VaR using EWMA method

Then I add VaR breaches information from Table 2 into the chart and mark them by red cross:

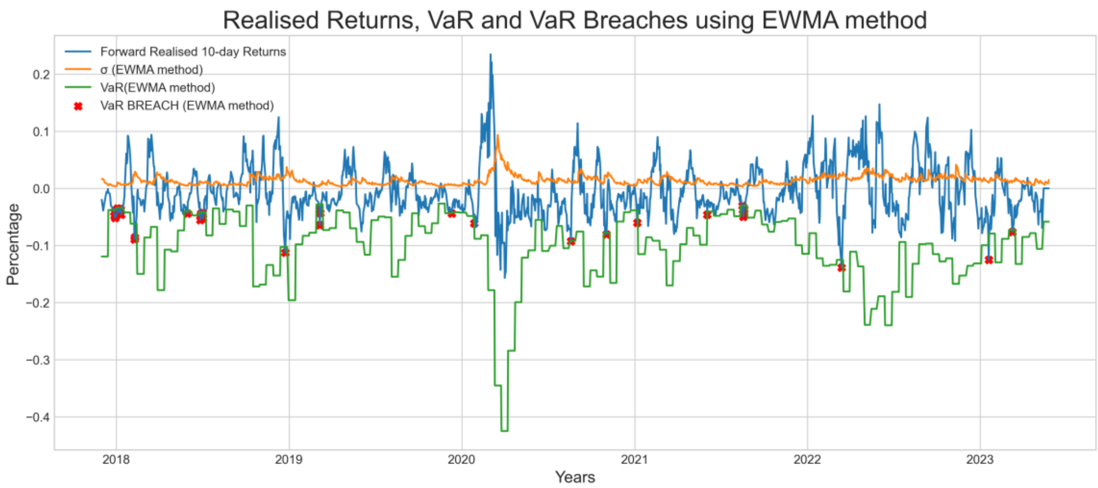


Figure 5 Realised Returns, VaR and VaR Breaches using EWMA method

Complete code for this case is attached in Appendix session "VaR Breaches using EWMA method".

(d.) Impact of λ on smoothness of EWMA-predicted volatility

Using EWMA method, today's variance is a function of previous day's variance (weighted by λ) and squared return. λ must be less than one to make sure the variance estimate is biased toward more recent data. In this method, λ is working as a smoothing parameter and higher value indicates slower decay in the series.

3. Appendix – Python Code

For Q2 – Covariance Matrix

```
#Import Libraries
import pandas as pd

import numpy as np
from numpy import *
from numpy.linalg import multi_dot

import matplotlib.pyplot as plt
from matplotlib.pyplot import rcParams
rcParams['figure.figsize'] = 16, 8

import math
```
✓ 0.0s Python

```
rtn=np.mat(array([0.05,0.07,0.15,0.22])).T;
rtn
```
✓ 0.0s Python

```
matrix([[0.05],
        [0.07],
        [0.15],
        [0.22]])
```

```
#SD=mat(array([0.07,0.28,0.25,0.31])).T;
SD_diag=mat(diag(array([0.07,0.28,0.25,0.31])))
SD_diag
```
✓ 0.0s Python

```
matrix([[0.07, 0.  , 0.  , 0.  ],
        [0.  , 0.28, 0.  , 0.  ],
        [0.  , 0.  , 0.25, 0.  ],
        [0.  , 0.  , 0.  , 0.31]])
```

```
Corr=mat([[1,0.4,0.3,0.3],[0.4,1,0.27,0.42],[0.3,0.27,1,0.5],[0.3,0.42,0.5,1]]);
Corr
```
✓ 0.0s Python

```
matrix([[1.  , 0.4 , 0.3 , 0.3 ],
        [0.4 , 1.  , 0.27, 0.42],
        [0.3 , 0.27, 1.  , 0.5 ],
        [0.3 , 0.42, 0.5 , 1.  ]])
```

```
#diagonal SD matrix * correlation matrix * diagonal SD matrix
#求Σ
#求SRS
cov = matmul(matmul(SD_diag,Corr),SD_diag.T)
cov
```
[16] ✓ 0.0s Python

```
matrix([[0.0049  , 0.00784 , 0.00525 , 0.00651 ],
        [0.00784 , 0.0784  , 0.0189  , 0.036456],
        [0.00525 , 0.0189  , 0.0625  , 0.03875 ],
        [0.00651 , 0.036456, 0.03875 , 0.0961  ]])
```

For Q2 – Optimal Weight and Portfolio Risk

```
[36]
one = np.ones((1,4)).T
✓ 0.0s                                                     Python
```

```
[37]
A = matmul(matmul(one.T,cov.I),one)
B = matmul(matmul(rtn.T,cov.I),one)
C = matmul(matmul(rtn.T,cov.I),rtn)
✓ 0.0s                                                     Python
```

```
[38]
A_value=A.item()
B_value=B.item()
C_value=C.item()

A_value,B_value,C_value
✓ 0.0s                                                     Python

... (209.72773941031215, 10.161029622941891, 0.8715791291363939)
```

```
[39]
m = 0.1
✓ 0.0s                                                     Python
```

```
[40]
λ = (A_value*m-B_value)/(A_value*C_value-pow(B_value,2))
λ
✓ 0.0s                                                     Python

... 0.13591506819283442
```

```
[41]
γ = (C_value-B_value*m)/(A_value*C_value-pow(B_value,2))
γ
✓ 0.0s                                                     Python

... -0.0018168175329735226
```

```
[42]
w_optimal=cov.I*(λ*rtn+γ*one)
w_optimal
✓ 0.0s                                                     Pytho

... matrix([[ 0.75851388],
            [-0.11685093],
            [ 0.12257544],
            [ 0.23576162]])
```

```
risk=w_optimal.T*cov*w_optimal
risk
✓ 0.0s                                                     Python

matrix([[0.01177469]])
```

For Q2 – Calculation for x1.3 and x1.8 Correlation Matrix

x1.3

```python
#To calculate x1.3
Corr_1=1.3*mat([[1,0.4,0.3,0.3],[0.4,1,0.27,0.42],[0.3,0.27,1,0.5],[0.3,0.42,0.5,1]]);
Corr_1
```

✓ 0.0s Python

```
matrix([[1.3  , 0.52 , 0.39 , 0.39 ],
        [0.52 , 1.3  , 0.351, 0.546],
        [0.39 , 0.351, 1.3  , 0.65 ],
        [0.39 , 0.546, 0.65 , 1.3  ]])
```

```python
#To satisfy individual correlation upper limit of 0.99
Corr_1_mod = mat([[0.99,0.52,0.39,0.39],[0.52,0.99,0.351,0.546],[0.39,0.351,0.99,0.65],[0.39,0.546
Corr_1_mod
```

✓ 0.0s Python

```
matrix([[0.99 , 0.52 , 0.39 , 0.39 ],
        [0.52 , 0.99 , 0.351, 0.546],
        [0.39 , 0.351, 0.99 , 0.65 ],
        [0.39 , 0.546, 0.65 , 0.99 ]])
```

```python
cov_1 = matmul(matmul(SD_diag,Corr_1_mod),SD_diag.T)
cov_1
```

✓ 0.0s Python

```
matrix([[0.004851 , 0.010192 , 0.006825 , 0.008463 ],
        [0.010192 , 0.077616 , 0.02457  , 0.0473928],
        [0.006825 , 0.02457  , 0.061875 , 0.050375 ],
        [0.008463 , 0.0473928, 0.050375 , 0.095139 ]])
```

```python
A_1 = matmul(matmul(one.T,cov_1.I),one)
B_1= matmul(matmul(rtn.T,cov_1.I),one)
C_1 = matmul(matmul(rtn.T,cov_1.I),rtn)

A_value_1=A_1.item()
B_value_1=B_1.item()
C_value_1=C_1.item()
```

✓ 0.0s Python

```python
λ_1 = (A_value_1*m-B_value_1)/(A_value_1*C_value_1-pow(B_value_1,2))
γ_1 = (C_value_1-B_value_1*m)/(A_value_1*C_value_1-pow(B_value_1,2))
λ_1, γ_1
```

✓ 0.0s Python

```
(0.13325480198096828, -0.001791564668528049)
```

```python
w_optimal_1=cov_1.I*(λ_1*rtn+γ_1*one)
w_optimal_1
```

✓ 0.0s Python

```
matrix([[ 0.87471062],
        [-0.21072126],
        [ 0.04153403],
        [ 0.2944766 ]])
```

```python
risk_1=sqrt(w_optimal_1.T*cov_1*w_optimal_1)
risk_1
```

✓ 0.0s Python

```
matrix([[0.10739607]])
```

x1.8

```python
#To calculate x1.8
Corr_2=1.8*mat([[1,0.4,0.3,0.3],[0.4,1,0.27,0.42],[0.3,0.27,1,0.5],[0.3,0.42,0.5,1]]);
Corr_2
```

✓ 0.0s Python

```
matrix([[1.8  , 0.72 , 0.54 , 0.54 ],
        [0.72 , 1.8  , 0.486, 0.756],
        [0.54 , 0.486, 1.8  , 0.9  ],
        [0.54 , 0.756, 0.9  , 1.8  ]])
```

```python
#To satisfy individual correlation upper limit of 0.99
Corr_2_mod = mat([[0.99,0.72,0.54,0.54],[0.72 ,0.99,0.486,0.756],[0.54,0.486,0.99,0.9],[0.54,0.756
Corr_2_mod
```

✓ 0.0s Python

```
matrix([[0.99 , 0.72 , 0.54 , 0.54 ],
        [0.72 , 0.99 , 0.486, 0.756],
        [0.54 , 0.486, 0.99 , 0.9  ],
        [0.54 , 0.756, 0.9  , 0.99 ]])
```

```python
cov_2 = matmul(matmul(SD_diag,Corr_2_mod),SD_diag.T)
cov_2
```

✓ 0.0s Python

```
matrix([[0.004851 , 0.014112 , 0.00945  , 0.011718 ],
        [0.014112 , 0.077616 , 0.03402  , 0.0656208],
        [0.00945  , 0.03402  , 0.061875 , 0.06975  ],
        [0.011718 , 0.0656208, 0.06975  , 0.095139 ]])
```

```python
A_2 = matmul(matmul(one.T,cov_2.I),one)
B_2 = matmul(matmul(rtn.T,cov_2.I),one)
C_2 = matmul(matmul(rtn.T,cov_2.I),rtn)

A_value_2=A_2.item()
B_value_2=B_2.item()
C_value_2=C_2.item()
```
✓ 0.0s Python

```python
λ_2 = (A_value_2*m-B_value_2)/(A_value_2*C_value_2-pow(B_value_2,2))
γ_2 = (C_value_2-B_value_2*m)/(A_value_2*C_value_2-pow(B_value_2,2))
λ_2, γ_2
```
✓ 0.0s Python

```
(-0.01856673072118691, 0.0014799844651424603)
```

```python
w_optimal_2=cov_2.I*(λ_2*rtn+γ_2*one)
w_optimal_2
```
✓ 0.0s Python

```
matrix([[ 1.71052985],
        [-0.68289799],
        [-0.97650538],
        [ 0.94887352]])
```

```python
risk_2=sqrt(w_optimal_2.T*cov_2*w_optimal_2)
risk_2
```
✓ 0.0s Python

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_16344\1869642439.py:1: RuntimeWarning: invalid value enc
  risk_2=sqrt(w_optimal_2.T*cov_2*w_optimal_2)
```

```
matrix([[nan]])
```

```python
w_optimal_2.T*cov_2*w_optimal_2
```
✓ 0.0s Python

```
matrix([[-0.00037669]])
```

For Q5 – VaR Breaches using Sample Standard Deviation method

```python
#Import packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as pltticker
plt.style.use('seaborn-whitegrid')
import math
from scipy.stats import norm
```
✓ 0.6s Python

```python
# Import data
data = pd.read_excel('C:/Users/Lenovo/Desktop/CQF June 2023 Lusha Li/E1_LUSHA LI_CODE/nasdaq100.xlsx', index_col=0, parse_dates=True)
data.head()
```

✓ 0.3s

	Closing Price
Date	
2017-12-01	6337.870117
2017-12-04	6263.700195
2017-12-05	6265.109863
2017-12-06	6293.049805
2017-12-07	6316.279785

```python
#1D realised log return
data['1D_log_rtn'] = ''

for i in range(0, len(data['Closing Price']), 1):
    if i < 1:
        data['1D_log_rtn'][i] = 0
        continue
    else:
        rtn = data['Closing Price'][i]/data['Closing Price'][i - 1]
        log_rtn_1D = np.log(rtn)
        data['1D_log_rtn'][i] = log_rtn_1D
```

✓ 0.2s

```python
data.head()
```

✓ 0.0s

	Closing Price	1D_log_rtn
Date		
2017-12-01	6337.870117	0
2017-12-04	6263.700195	-0.011772
2017-12-05	6265.109863	0.000225
2017-12-06	6293.049805	0.00445
2017-12-07	6316.279785	0.003685

```python
#10D forward realised log return
data['10D_forward_log_rtn'] = ''

for i in range(0, len(data['Closing Price']), 1):
    if i > len(data['Closing Price']) - 11:
        data['10D_forward_log_rtn'][i] = 0
        continue
    else:
        rtn_10D = data['Closing Price'][i]/data['Closing Price'][i + 10] #to get negative values
        forward_log_rtn_10D = np.log(rtn_10D)
        data['10D_forward_log_rtn'][i] = forward_log_rtn_10D
```

✓ 0.2s

```python
data.head()
```

✓ 0.0s

	Closing Price	1D_log_rtn	10D_forward_log_rtn
Date			
2017-12-01	6337.870117	0	-0.020064
2017-12-04	6263.700195	-0.011772	-0.039071
2017-12-05	6265.109863	0.000225	-0.033828
2017-12-06	6293.049805	0.00445	-0.028114
2017-12-07	6316.279785	0.003685	-0.024461

```python
# Compute the rolling standard deviation from 21 daily returns
data['sigma_21D'] = ''

for i in range(0, len(data['Closing Price']), 1):
    if i < 21:
        data['sigma_21D'][i] = 0
        continue
    else:
        init = i - 20
        std_dev = np.std(data['1D_log_rtn'].iloc[init:i])
        data['sigma_21D'][i] = std_dev
```

✓ 0.3s

```python
data.iloc[0:30]
```

✓ 0.0s

	Closing Price	1D_log_rtn	10D_forward_log_rtn	sigma_21D
Date				
2017-12-01	6337.870117	0	-0.020064	0
2017-12-04	6263.700195	-0.011772	-0.039071	0
2017-12-05	6265.109863	0.000225	-0.033828	0
2017-12-06	6293.049805	0.00445	-0.028114	0
2017-12-07	6316.279785	0.003685	-0.024461	0
2017-12-08	6344.569824	0.004469	-0.01883	0
2017-12-11	6393.890137	0.007744	-0.006123	0
2017-12-12	6383.649902	-0.001603	-0.008035	0
2017-12-13	6394.669922	0.001725	-0.007284	0
2017-12-14	6389.910156	-0.000745	-0.001018	0
2017-12-15	6466.319824	0.011887	-0.006938	0
2017-12-18	6513.270020	0.007234	-0.009555	0
2017-12-19	6480.669922	-0.005018	-0.015907	0
2017-12-20	6472.479980	-0.001265	-0.027552	0
2017-12-21	6472.689941	0.000032	-0.031022	0
2017-12-22	6465.169922	-0.001162	-0.03238	0
2017-12-26	6433.160156	-0.004963	-0.035053	0
2017-12-27	6435.149902	0.000309	-0.041599	0
2017-12-28	6441.419922	0.000974	-0.048058	0
2017-12-29	6396.419922	-0.007011	-0.051897	0
2018-01-02	6511.339844	0.017807	-0.044888	0
2018-01-03	6575.799805	0.009851	-0.035199	0.006462
2018-01-04	6584.580078	0.001334	-0.037228	0.005966
2018-01-05	6653.290039	0.010381	-0.03732	0.005951
2018-01-08	6676.629883	0.003502	-0.042063	0.006184
2018-01-09	6677.939941	0.000196	-0.035512	0.006183
2018-01-10	6662.660156	-0.002291	-0.037362	0.006195
2018-01-11	6708.490234	0.006855	-0.045812	0.006161

```python
# Compute 10-day sigma for projection
data['sigma_10D'] = np.sqrt(10) * data['sigma_21D']
data.iloc[0:30]
```

✓ 0.0s

Date	Closing Price	1D_log_rtn	10D_forward_log_rtn	sigma_21D	sigma_10D
2017-12-01	6337.870117	0	-0.020064	0	0.0
2017-12-04	6263.700195	-0.011772	-0.039071	0	0.0
2017-12-05	6265.109863	0.000225	-0.033828	0	0.0
2017-12-06	6293.049805	0.00445	-0.028114	0	0.0
2017-12-07	6316.279785	0.003685	-0.024461	0	0.0
2017-12-08	6344.569824	0.004469	-0.01883	0	0.0
2017-12-11	6393.890137	0.007744	-0.006123	0	0.0
2017-12-12	6383.649902	-0.001603	-0.008035	0	0.0
2017-12-13	6394.669922	0.001725	-0.007284	0	0.0
2017-12-14	6389.910156	-0.000745	-0.001018	0	0.0
2017-12-15	6466.319824	0.011887	-0.006938	0	0.0
2017-12-18	6513.270020	0.007234	-0.009555	0	0.0
2017-12-19	6480.669922	-0.005018	-0.015907	0	0.0
2017-12-20	6472.479980	-0.001265	-0.027552	0	0.0
2017-12-21	6472.689941	0.000032	-0.031022	0	0.0
2017-12-22	6465.169922	-0.001162	-0.03238	0	0.0
2017-12-26	6433.160156	-0.004963	-0.035053	0	0.0
2017-12-27	6435.149902	0.000309	-0.041599	0	0.0
2017-12-28	6441.419922	0.000974	-0.048058	0	0.0
2017-12-29	6396.419922	-0.007011	-0.051897	0	0.0
2018-01-02	6511.339844	0.017807	-0.044888	0	0.0
2018-01-03	6575.799805	0.009851	-0.035199	0.006462	0.020436
2018-01-04	6584.580078	0.001334	-0.037228	0.005966	0.018867
2018-01-05	6653.290039	0.010381	-0.03732	0.005951	0.018818
2018-01-08	6676.629883	0.003502	-0.042063	0.006184	0.019557

```python
# 99% level of confidence factor
factor = 2.3
```

✓ 0.0s

```python
# Calculate 10-day VaR
data['VAR_10D'] = ''
item = 1

for i in range(0, len(data), 1):
    if data['sigma_21D'][i] == '':
        data['VAR_10D'][i] = 0
        continue
    else:
        if item == 11 or item == 1:
            var = - factor * data['sigma_21D'][i] * np.sqrt(10)
            item = 1
        else:
            var = var
        data['VAR_10D'][i] = var
        item = item + 1
```

✓ 0.2s

```python
# data.iloc[0:50]
data
```

✓ 0.0s

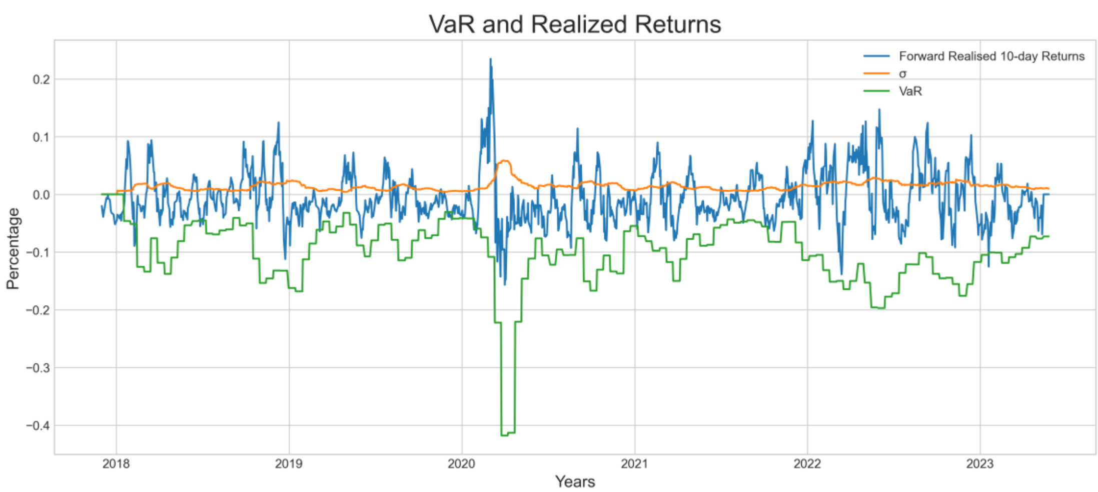
Date	Closing Price	1D_log_rtn	10D_forward_log_rtn	sigma_21D	sigma_10D	VAR_10D
2017-12-01	6337.870117	0	-0.020064	0	0.0	-0.0
2017-12-04	6263.700195	-0.011772	-0.039071	0	0.0	-0.0
2017-12-05	6265.109863	0.000225	-0.033828	0	0.0	-0.0
2017-12-06	6293.049805	0.00445	-0.028114	0	0.0	-0.0
2017-12-07	6316.279785	0.003685	-0.024461	0	0.0	-0.0
...	...	...	...	...	...	...
2023-05-22	13849.740230	0.003345	0	0.01071	0.03387	-0.073022
2023-05-23	13672.540040	-0.012877	0	0.010639	0.033644	-0.073022
2023-05-24	13604.480470	-0.00499	0	0.010061	0.031817	-0.073022
2023-05-25	13938.530270	0.024258	0	0.010208	0.03228	-0.073022
2023-05-26	14298.410160	0.025491	0	0.00987	0.031212	-0.073022

1380 rows × 6 columns

```python
plt.figure(dpi=300, figsize=(15,6))
plt.plot(data.index, data['10D_forward_log_rtn'], label='Forward Realised 10-day Returns')
plt.plot(data['sigma_21D'], label='σ')
plt.plot(data['VAR_10D'], label='VaR')

plt.title('VaR and Realized Returns', fontsize=20)
plt.xlabel('Years', fontsize=13)
plt.ylabel('Percentage', fontsize=13)
plt.legend()
plt.show()
```

✓ 0.4s




```python
# Compute the VaR breaches
breaches = 0
consecutive_breaches = 0
item = 0
breach_list = []

for i in range(0, len(data), 1):
    if data['10D_forward_log_rtn'][i] == 0 or data['VAR_10D'][i] == 0:
        breach_list.append([data.index[i], 0])
        continue
    else:
        item += 1

        if data['10D_forward_log_rtn'][i] < data['VAR_10D'][i]:
            breach_list.append([data.index[i], data['10D_forward_log_rtn'][i]])
            breaches += 1
        else:
            breach_list.append([data.index[i], 0])

    if data['10D_forward_log_rtn'][i - 1] == 0 or data['VAR_10D'][i - 1] == 0:
        continue
    else:
        if data['10D_forward_log_rtn'][i - 1] < data['VAR_10D'][i - 1] and data['10D_forward_log_rtn'][i] < data['VAR_10D'][i]:
            consecutive_breaches += 1

breach_list = pd.DataFrame(breach_list)
breach_list.columns = ['Date', 'breaches']
breach_list.set_index('Date', inplace=True)
```

✓ 0.0s

```python
# Number of breaches
breaches
```
0.0s

```
21
```

```python
# Percentage of breaches
Percentage = (breaches/item) * 100
Percentage
```
0.0s

```
1.5671641791044775
```

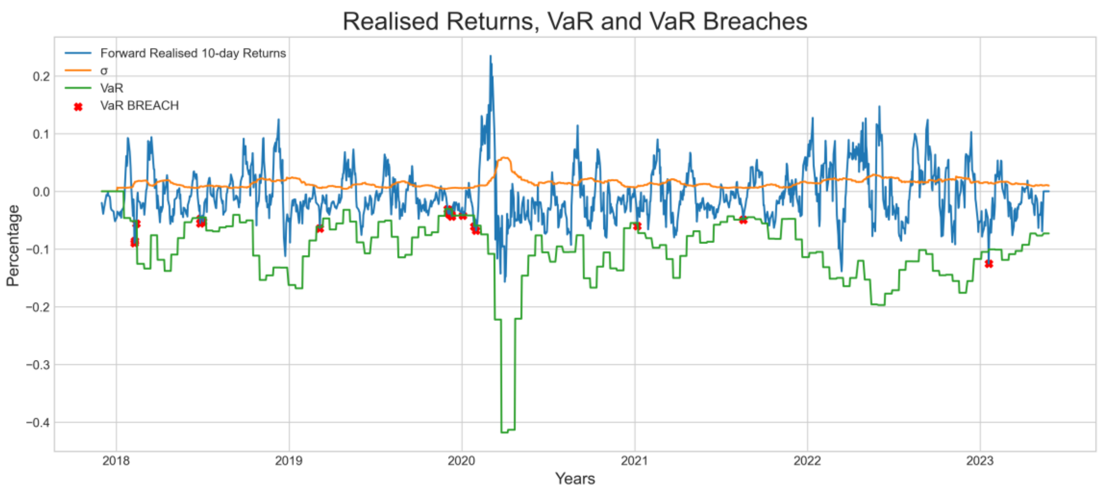
```python
# Number of consecutive breaches
consecutive_breaches
```
0.0s

```
10
```

```python
# Add breaches into the chart
plt.figure(dpi=300, figsize=(15,6))
plt.plot(data.index, data['10D_forward_log_rtn'], label='Forward Realised 10-day Returns')
plt.plot(data['sigma_21D'], label='σ')
plt.plot(data['VAR_10D'], label='VaR')

# plt.bar(data.index, height=breach_list['breaches'], label='VaR BREACH', color='red')
plt.scatter(data.index[breach_list['breaches']!=0],breach_list[breach_list['breaches']!=0], label='VaR BREACH',marker='X', color = 'red')

plt.title('Realised Returns, VaR and VaR Breaches', fontsize=20)
plt.xlabel('Years', fontsize=13)
plt.ylabel('Percentage', fontsize=13)
plt.legend()
plt.show()
```
0.5s




```python
# Convert the date to datetime64
data['Date'] = data.index
data['Date'] = pd.to_datetime(data['Date'], format='%Y-%m-%d')

# Filter data between two dates
filtered_df = data.loc[(data['Date'] >= '2020-01-01')
                    & (data['Date'] < '2020-04-01')
                    & (data['10D_forward_log_rtn'] < data['VAR_10D'])]

# Display
filtered_df
```
0.0s

	Closing Price	1D_log_rtn	10D_forward_log_rtn	sigma_21D	sigma_10D	VAR_10D	Date
Date							
2020-01-03	8793.900391	-0.008867	-0.042286	0.00525	0.016603	-0.041266	2020-01-03
2020-01-27	8952.179688	-0.020924	-0.061166	0.006623	0.020943	-0.041823	2020-01-27
2020-01-31	8991.509766	-0.015952	-0.067936	0.008515	0.026926	-0.059474	2020-01-31

For Q6 – VaR Breaches using EWMA method

```python
# Compute the variance estimate using EMWA method
λ = 0.72
data['Variance Estimate'] = ''

for i in range(0, len(data['Closing Price']), 1):
    if i <= 1:
        data['Variance Estimate'][i] = Avg_Var
        continue
    else:
        variance = (λ * data['Variance Estimate'][i-1]) + ((1-λ) * data['Squared Return'][i-1])
        data['Variance Estimate'][i] = variance
```

✓ 0.2s Python

```python
data.head()
```

✓ 0.0s Python

Date	Closing Price	1D_log_rtn	10D_forward_log_rtn	Squared Return	Variance Estimate
2017-12-01	6337.870117	0	-0.020064	0	0.000269
2017-12-04	6263.700195	-0.011772	-0.039071	0.000139	0.000269
2017-12-05	6265.109863	0.000225	-0.033828	0.0	0.000233
2017-12-06	6293.049805	0.00445	-0.028114	0.00002	0.000167
2017-12-07	6316.279785	0.003685	-0.024461	0.000014	0.000126

```
# Add breaches into the chart
plt.figure(dpi=300, figsize=(15,6))
plt.plot(data.index, data['10D_forward_log_rtn'], label='Forward Realised 10-day Returns')
plt.plot(data['sigma_1D'], label='σ (EWMA method)')
plt.plot(data['VAR_10D'], label='VaR(EWMA method)')

# plt.bar(data.index, height=breach_list['breaches'], label='VaR BREACH', color='red')
plt.scatter(data.index[breach_list['breaches']!=0], breach_list[breach_list['breaches']!=0], labe

plt.title('Realised Returns, VaR and VaR Breaches using EWMA method', fontsize=20)
plt.xlabel('Years', fontsize=13)
plt.ylabel('Percentage', fontsize=13)
plt.legend()
plt.show()
```

✓ 0.4s Pythor