CQF Exam One Report - Harrison King

June 2023 Cohort

Optimal Portfolio Allocation

An investment universe of the following risky assets with a dependence structure (correlation) applies to all questions below as relevant:

Asset	$\boldsymbol{\mu}$	$\boldsymbol{\sigma}$	$\boldsymbol{w}$
A	0.05	0.07	w_1
B	0.07	0.28	w_2
C	0.15	0.25	w_3
D	0.22	0.31	w_4

$$\text{Corr} = \begin{pmatrix} 1 & 0.4 & 0.3 & 0.3 \\ 0.4 & 1 & 0.27 & 0.42 \\ 0.3 & 0.27 & 1 & 0.5 \\ 0.3 & 0.42 & 0.5 & 1 \end{pmatrix}$$

Question 1:

Global Minimum Variance portfolio is obtained subject to the budget constraint:

$$\underset{\boldsymbol{w}}{\text{argmin}}\, \frac{1}{2}\boldsymbol{w}'\boldsymbol{\Sigma}\boldsymbol{w} \quad \text{s.t. } \boldsymbol{w}'\mathbf{1} = 1$$

1.1) Derive the analytical solution for optimal allocations $\boldsymbol{w}^*$. Provide full mathematical workings.

1.2) In the derivation, include the formula derivation for the Lagrangian multiplier.

1.3) Compute optimal allocations (Global MV portfolio) for the given investment universe.

1.1) First define the diagonal standard deviation matrix, $\boldsymbol{S}$, is defined as follows:

$$\boldsymbol{S} = \begin{pmatrix} \sigma_A & 0 & 0 & 0 \\ 0 & \sigma_B & 0 & 0 \\ 0 & 0 & \sigma_C & 0 \\ 0 & 0 & 0 & \sigma_D \end{pmatrix} = \begin{pmatrix} 0.07 & 0 & 0 & 0 \\ 0 & 0.28 & 0 & 0 \\ 0 & 0 & 0.25 & 0 \\ 0 & 0 & 0 & 0.31 \end{pmatrix}$$

We can calculate the covariance matrix, Σ by pre-multiplying and post-multiplying the correlation matrix, $Corr$ by the diagonal standard deviation matrix, $\boldsymbol{S}$:

$$\Sigma = \boldsymbol{S}Corr\boldsymbol{S} = \begin{pmatrix} \sigma_A^2 & \rho_{AB}\sigma_A\sigma_B & \rho_{AC}\sigma_A\sigma_C & \rho_{AD}\sigma_A\sigma_D \\ \rho_{BA}\sigma_A\sigma_B & \sigma_B^2 & \rho_{BC}\sigma_B\sigma_C & \rho_{BD}\sigma_B\sigma_D \\ \rho_{CA}\sigma_A\sigma_C & \rho_{CB}\sigma_B\sigma_C & \sigma_C^2 & \rho_{CD}\sigma_C\sigma_D \\ \rho_{DA}\sigma_A\sigma_D & \rho_{DB}\sigma_B\sigma_D & \rho_{DC}\sigma_C\sigma_D & \sigma_D^2 \end{pmatrix}$$

Then we can define the Lagrangian function $L(x, \lambda)$ with one multiplier λ which represents the budget constraint, $\boldsymbol{w}'\mathbf{1} = 1 == \boldsymbol{w}\mathbf{1}' = 1$:

$L(x, \lambda) = \frac{1}{2}\boldsymbol{w}'\Sigma\boldsymbol{w} + \lambda(1 - \mathbf{1}'\boldsymbol{w})$

Solving the first order equation of the Lagrangian function we get two partial differential equations:

$\frac{\partial L}{\partial w}(w, \lambda) = \boldsymbol{w}'\Sigma - \lambda\mathbf{1}' = 0$

$\frac{\partial L}{\partial \lambda}(w, \lambda) = 1 - \mathbf{1}'\boldsymbol{w} = 0$

Then we can define the optimal portfolio allocation $\boldsymbol{w}^*$ by multiplying $\boldsymbol{w}'\Sigma - \lambda\mathbf{1}' = 0$ by the inverse matrix Σ^{-1}:

$\boldsymbol{w}'\Sigma = \lambda\mathbf{1}'$

$\therefore \boldsymbol{w}^* = \Sigma^{-1}(\lambda\mathbf{1})$ which we can call the candidate function.

1.2) We can then derive the Lagrangian multiplier λ by substituting $\boldsymbol{w}^*$ into the budget constraint $\boldsymbol{w}\mathbf{1}' = 1$:

$\mathbf{1}'\Sigma^{-1}(\lambda\mathbf{1}) = \lambda\mathbf{1}'\Sigma^{-1}\mathbf{1} = 1$

$\therefore \lambda = \frac{1}{\mathbf{1}'\Sigma^{-1}\mathbf{1}}$

Finally, we can define the optimal portfolio allocation $\boldsymbol{w}^*$ by substituting λ into the candidate function $\boldsymbol{w}^* = \Sigma^{-1}(\lambda\mathbf{1})$:

$$\text{Optimial Portfolio Allocation } \boldsymbol{w}^* = \Sigma^{-1}(\lambda\mathbf{1}) = \Sigma^{-1}(\frac{1}{\mathbf{1}'\Sigma^{-1}\mathbf{1}}\mathbf{1})$$

1.3) Computing allocations: Define the inputs

In [ ]:
```python
# Define the assets, standard deviations vector and correlation matrix
assets = ['A', 'B', 'C', 'D']
std_deviation_vector = np.array([0.07, 0.28, 0.25, 0.31])
correlation_matrix = np.array([[1.0, 0.4, 0.3, 0.3],
                               [0.4, 1.0, 0.27, 0.42],
                               [0.3, 0.27, 1.0, 0.5],
                               [0.3, 0.42, 0.5, 1.0]])

# Function to calculate covariance matrix using the correlation matrix and the standard deviations vector
def calculate_Covariance_Matrix(std_deviation_vector: np.ndarray, correlation_matrix: np.ndarray):
    ''' Calculates the covariance matrix from the standard deviation vector and correlation matrix
    Args:
        std_deviation_vector (np.ndarray): Vector of asset standard deviations
        correlation_matrix (np.ndarray): Correlation matrix of assets
    '''
    return np.outer(std_deviation_vector, std_deviation_vector) * correlation_matrix

covariance_matrix = calculate_Covariance_Matrix(std_deviation_vector, correlation_matrix)
print(f'Covariance Matrix:\n{covariance_matrix}')
```

In [ ]:
```python
# Initiliase the 1s vector
ones_vector = np.ones(len(assets))

# Calculate the inverse of the covariance matrix (Σ^-1)
inverse_covariance_matrix = np.linalg.inv(covariance_matrix)
print(f'Inverse covariance matrix:\n{inverse_covariance_matrix}\n')
```

Computing allocations: Calculate the Lagrangian Multiplier lambda $\lambda = \frac{1}{\mathbf{1}'\Sigma^{-1}\mathbf{1}}$

In [4]:
```python
# λ = 1 / 1^'.(Σ^-1).1^
lagrangian_lambda = 1 / ones_vector.T.dot(inverse_covariance_matrix).dot(ones_vector)
print(f'λ = {lagrangian_lambda}')
```

```
λ = 0.004768086485896823
```

Computing allocations: Optimal portfolio allocations $\boldsymbol{w}^* = \Sigma^{-1}(\lambda\mathbf{1})$

In [5]:
```python
# Optimal weights, w* = (Σ^-1).(λ.1^)
optimal_weights = inverse_covariance_matrix.dot(lagrangian_lambda * ones_vector)

# Format the weights and print
formatted_weights = [f'{weight:.2%}' for weight in optimal_weights]
portfolio = dict(zip(assets, formatted_weights))

print(tabulate.tabulate(portfolio.items(), headers=['Asset', 'Weight'], tablefmt='fancy_grid'))
```

Asset	Weight
A	104.31%
B	-4.13%
C	0.60%
D	-0.78%

Question 2:

Consider the optimization for a target return m. There is no risk-free asset.

$$\underset{w}{\operatorname{argmin}} \frac{1}{2}\boldsymbol{w}'\boldsymbol{\Sigma}\boldsymbol{w}$$
$$\boldsymbol{w}'\mathbf{1} = 1$$
$$\boldsymbol{w}'\boldsymbol{\mu} = m$$

2.1) Compute $\mathbf{w}^*$ and portfolio risk $\sigma_{\Pi} = \sqrt{\boldsymbol{w}'\boldsymbol{\Sigma}\boldsymbol{w}}$ for $m = 7\%$ for three levels of correlation.

2.2) Correlation matrix $\times 1, \times 1.3, \times 1.8$, subject to individual correlation upper limit of 0.99 , if the scaling results in correlation value above 1 . Provide all results in a single table.

The vector of returns, $\boldsymbol{\mu}$ is defined as:

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_A \\ \mu_B \\ \mu_C \\ \mu_D \end{pmatrix} = \begin{pmatrix} 0.05 \\ 0.07 \\ 0.15 \\ 0.22 \end{pmatrix}$$

In [6]:
```
# Define the min return, m = 7% and vector of mean returns of the assets, μ
min_return = 0.07
mean_returns_vector = np.array([0.05, 0.07, 0.15, 0.22])
```

2.1) We'll define $\mathbf{w}^*$ by solving a new Lagrangian function, $L(x, \lambda, \gamma)$ with two Lagrangian multipliers λ and γ which represent the budget constraint, $\boldsymbol{w}'\mathbf{1} = 1 == \boldsymbol{w}\mathbf{1}' = 1$ and the return constraint, $\boldsymbol{w}'\boldsymbol{\mu} = m == \boldsymbol{w}\boldsymbol{\mu}' = m$:

Then we can define the Lagrangian function $L(x, \lambda, \gamma)$ as:

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

Solving the first order equation of the Lagrangian function we get three partial differential equations:

$\frac{\partial L}{\partial w}(w, \lambda, \gamma) = \boldsymbol{w}'\Sigma - \lambda\boldsymbol{\mu}' - \gamma\mathbf{1}' = 0$

$\frac{\partial L}{\partial \lambda}(w, \lambda, \gamma) = m - \boldsymbol{\mu}'\boldsymbol{w} = 0$

$\frac{\partial L}{\partial \gamma}(w, \lambda, \gamma) = 1 - \mathbf{1}'\boldsymbol{w} = 0$

Then we can define the optimal portfolio allocation $\boldsymbol{w}^*$ by multiplying $\boldsymbol{w}'\Sigma - \lambda\boldsymbol{\mu}' - \gamma\mathbf{1}' = 0$ by the inverse matrix Σ^{-1}:

$\boldsymbol{w}'\Sigma = \lambda\boldsymbol{\mu}' + \gamma\mathbf{1}'$

$\therefore \boldsymbol{w}^* = \Sigma^{-1}(\lambda\boldsymbol{\mu} + \gamma\mathbf{1})$ which we can call the candidate function.

Substituting the candidate function, $\boldsymbol{w}^*$ into our two constraint equations we will try to solve for λ and γ.

First we will substitute $\boldsymbol{w}^*$ into the budget constraint equation $\boldsymbol{w}\mathbf{1}' = 1$:

$$\boldsymbol{w}^*\mathbf{1}' = 1$$
$$\Rightarrow \mathbf{1}'\Sigma^{-1}(\lambda\boldsymbol{\mu} + \gamma\mathbf{1}) = 1$$
$$[1] \quad \Rightarrow \lambda\mathbf{1}'\Sigma^{-1}\boldsymbol{\mu} + \gamma\mathbf{1}'\Sigma^{-1}\mathbf{1} = 1$$

Then we will substitute $\boldsymbol{w}^*$ into the return constraint equation $\boldsymbol{w}\boldsymbol{\mu}' = m$:

$$\boldsymbol{w}^*\boldsymbol{\mu}' = m$$
$$\Rightarrow \boldsymbol{\mu}'\Sigma^{-1}(\lambda\boldsymbol{\mu} + \gamma\mathbf{1}) = m$$
$$[2] \quad \Rightarrow \lambda\boldsymbol{\mu}'\Sigma^{-1}\boldsymbol{\mu} + \gamma\boldsymbol{\mu}'\Sigma^{-1}\mathbf{1} = m$$

For ease of computation we will define the following scalar variables:

$$A = \mathbf{1}\Sigma^{-1}\mathbf{1}$$
$$B = \boldsymbol{\mu}'\Sigma^{-1}\mathbf{1} \Rightarrow \mathbf{1}\Sigma^{-1}\boldsymbol{\mu}$$
$$C = \boldsymbol{\mu}'\Sigma^{-1}\boldsymbol{\mu}$$

We can then re-write equations $[1]$ and $[2]$ as:

$$[1] \quad B\lambda + A\gamma = 1$$

$$[2] \quad C\lambda + B\gamma = m$$

Rearranging $[1]$ and $[2]$ we define our Lagrangian multipliers, λ and γ, in terms of the constants A, B, and C:

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

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

$$\text{Optimial Portfolio Allocation } \boldsymbol{w}^* = \Sigma^{-1}(\lambda\boldsymbol{\mu} + \gamma\mathbf{1})$$

2.2) Define the scaled correlation matrices as $Corr_1$, $Corr_{1.3}$ and $Corr_{1.8}$ respectively.

$$Corr_1 = \begin{pmatrix} 1 & 0.4 & 0.3 & 0.3 \\ 0.4 & 1 & 0.27 & 0.42 \\ 0.3 & 0.27 & 1 & 0.5 \\ 0.3 & 0.42 & 0.5 & 1 \end{pmatrix}$$

$$Corr_{1.3} = \begin{pmatrix} 1 & 0.52 & 0.39 & 0.39 \\ 0.52 & 1 & 0.35 & 0.55 \\ 0.39 & 0.35 & 1 & 0.65 \\ 0.39 & 0.55 & 0.65 & 1 \end{pmatrix}$$

$$Corr_{1.8} = \begin{pmatrix} 1 & 0.72 & 0.54 & 0.54 \\ 0.72 & 1 & 0.49 & 0.76 \\ 0.54 & 0.49 & 1 & 0.9 \\ 0.54 & 0.76 & 0.9 & 1 \end{pmatrix}$$

Consequently, the covariance matrices Σ_1, $\Sigma_{1.3}$ and $\Sigma_{1.8}$ are given by:

$$\Sigma_1 = \boldsymbol{S}Corr_1\boldsymbol{S}$$

$$\Sigma_{1.3} = \boldsymbol{S}Corr_{1.3}\boldsymbol{S}$$

$$\Sigma_{1.8} = \boldsymbol{S}Corr_{1.8}\boldsymbol{S}$$

where $\boldsymbol{S}$ is the diagonal matrix of standard deviations of the assets.

Now we can compute the optimal allocation, $\boldsymbol{w}^*$ and portfolio risk, σ_Π for $m = 7\%$ for each of the three correlation matrices.

In [7]:
```python
# Define a function that uses Lagrange multipliers to find the minimum variance portfolio for a given return
def optimal_Lagrangian_Allocations(min_return: float, covariance_matrix: np.ndarray, returns_vector: np.ndarray):
    ''' Calculate the Optimal Portfolio allocations using Lagrangian Multipliers closed form solution
    Args:
        min_return (float) = The minimum or 'target' return required for the portfolio, as a decimal (e.g. 0.1 = 10%)
        covariance_matrix (np.array) = The covariance matrix of the portfolio
        returns_vector (np.array) = The mean returns vector of the portfolio
    Returns:
        optimal_weights (np.array) = The optimal weights for the portfolio
    '''
    # Initiliase the 1s vector
    ones_vector = np.ones(returns_vector.shape[0])

    # Calculate the inverse of the covariance matrix (Σ^-1)
    inverse_covariance = np.linalg.inv(covariance_matrix)

    # Get scalar values to solve the quadratic equation for the two constaint equations
    # A = 1^.(Σ^-1).1^
    A = ones_vector.dot(inverse_covariance.dot(ones_vector))

    # B = 1^.(Σ^-1).μ
    B = ones_vector.dot(inverse_covariance.dot(returns_vector))

    # C = μ'.(Σ^-1).μ
    C = returns_vector.T.dot(inverse_covariance.dot(returns_vector))

    # λ = (A*min_return - B) / (A*C - B^2)
    lagrangian_lambda = (A*min_return - B) / (A*C - B**2)

    # γ = (C - B*min_return) / (A*C - B^2)
    lagrangian_gamma = (C - B*min_return) / (A*C - B**2)

    # Calculate the optimal weights, w* for the portfolio
    # w* = (Σ^-1).(λ.μ + γ.1^)
    optimal_weights = inverse_covariance.dot(lagrangian_lambda * returns_vector + lagrangian_gamma * ones_vector)

    return optimal_weights
```

In [ ]:
```
# Create the scaled correlation matrices
corr_1 = correlation_matrix
corr_1_3 = correlation_matrix * 1.3
corr_1_8 = correlation_matrix * 1.8

# If any correlation is greater than 1, set it to 0.99
corr_1_3[corr_1_3 > 1.0] = 0.99
corr_1_8[corr_1_8 > 1.0] = 0.99

# Set the leading diagonals back to 1.0
np.fill_diagonal(corr_1_3, 1.0)
np.fill_diagonal(corr_1_8, 1.0)

print(f'Correlation Matrix x1: \n{corr_1}\n')
print(f'Correlation Matrix x1.3: \n{corr_1_3}\n')
print(f'Correlation Matrix x1.8: \n{corr_1_8}')
```

In [ ]:
```
# Calculate the covariance matrices for each of the correlation matrices
covariance_matrix_1 = covariance_matrix
covariance_matrix_1_3 = calculate_Covariance_Matrix(std_deviation_vector, corr_1_3)
covariance_matrix_1_8 = calculate_Covariance_Matrix(std_deviation_vector, corr_1_8)

# Store the matrices in a list
covariance_matrices = [covariance_matrix_1, covariance_matrix_1_3, covariance_matrix_1_8]

print(f'Covariance Matrix 1: \n{covariance_matrix_1}\n')
print(f'Covariance Matrix 1_3: \n{covariance_matrix_1_3}\n')
print(f'Covariance Matrix 1_8: \n{covariance_matrix_1_8}\n')
```

Find the portfolio risk as $\sigma_{\Pi} = \sqrt{\boldsymbol{w}'\boldsymbol{\Sigma}\boldsymbol{w}}$ defining this calculation as a function.

In [10]:
```
def calculate_Portfolio_Risk(weights: np.ndarray, covariance_matrix: np.ndarray):
    '''Calculate the annualised portfolio risk (standard deviation) given a set of weights
    Args:
        weights (np.ndarray): An array of assets weights
        covariance_matrix (np.ndarray): The covariance matrix of the assets
    Returns:
        annualised_std (float): The annualised portfolio standard deviation'''
    annualised_std = np.sqrt(np.dot(weights.T, np.dot(covariance_matrix, weights))) * np.sqrt(252)

    return annualised_std
```

In [11]:
```
# Find optimal portfolio allocations for each correlation matrix using the Lagrangian method using the minimum return of
# 0.07 and the mean returns vector
optimal_weights = [optimal_Lagrangian_Allocations(min_return, covariance_matrix, mean_returns_vector)
                   for covariance_matrix in covariance_matrices]

# Find the corresponding portfolio risk for each set of optimal weights
portfolio_risk = [calculate_Portfolio_Risk(weights, covariance_matrix)
                  for weights, covariance_matrix in zip(optimal_weights, covariance_matrices)]

# Formatting the optimal weights into percentages and column names
formatted_weights = [np.around(weights * 100, 2) for weights in optimal_weights]
formatted_assets = [asset + ' Weight %' for asset in assets]

# Create a dataframe representing the portfolios with the optimal weights, portfolio risks and correlation multipliers
# for each correlation matrix
portfolios_df = pd.DataFrame(formatted_weights, columns=formatted_assets)
portfolios_df['Portfolio Risk'] = np.around(portfolio_risk, 3)
portfolios_df['Correlation Multiplier'] = [1.0, 1.3, 1.8]
portfolios_df.head()
```

Out[11]:

	A Weight %	B Weight %	C Weight %	D Weight %	Portfolio Risk	Correlation Multiplier
0	92.41	-7.29	5.47	9.40	1.229	1.0
1	99.65	-13.51	1.24	12.62	1.214	1.3
2	145.09	-40.77	-50.71	46.39	0.655	1.8

Understanding Risk

Question 3:

"Evaluating the P&L more frequently make it appear more risky than it actually is." Make the following simple computations to demonstrate this statement.

3.1) Write down the formula for Sharpe Ratio and identify main parameter scaled with time.

3.2) Compute Daily, Monthly, and Quarterly Sharpe Ratio, for Annualised SR of 0.53. No other inputs.

3.3) Convert each Sharpe Ratio into Loss Probability (daily, monthly, quarterly, annual), using

$$\Pr(\text{P\& L} < 0) = \Pr(x < -\text{SR}).$$

where x is a standard Normal random variable.

3.1) The Sharpe Ratio, S_t is the ratio of the risk-adjusted average excess returns of an asset or portfolio to the standard deviation of those returns. It is therefore a measure of the returns per unit of risk of an asset or portfolio. The Sharpe Ratio is defined as:

$$S_t = \frac{\mu_t - R_f}{\sigma_t}$$

where μ_t is the average excess return of the asset or portfolio, R_f is the risk-free rate, and σ_t is the standard deviation of the excess returns of the asset or portfolio.

The main parameter scaled with time is the standard deviation of the excess returns or the volatility of the excess returns, σ_t as it scales non-linearly with the square root of time, $\sqrt{t}$ compared to the average excess return, μ_t which scales linearly with time. The risk-free rate, R_f is assumed to be constant over the time period.

3.2) Annualised SR = 0.53, we will write this as $S_{annual} = 0.53$ and assume there are 252 trading days in a year.

Daily Sharpe Ratio, $S_{daily} = \frac{S_{annual}}{\sqrt{252}}$

Monthly Sharpe Ratio, $S_{monthly} = \frac{S_{annual}}{\sqrt{12}}$

Quarterly Sharpe Ratio, $S_{quarterly} = \frac{S_{annual}}{\sqrt{4}}$

In [12]:
```
# Define the trading days in year and the annual sharpe ratio
TRADING_DAYS = 252
sharpe_ratio_annual = 0.52

# Calculate the daily sharpe ratio
sharpe_ratio_daily = sharpe_ratio_annual / np.sqrt(TRADING_DAYS)

# Calculate the monthly sharpe ratio
sharpe_ratio_monthly = sharpe_ratio_annual / np.sqrt(12)

# Calculate the quarterly sharpe ratio
sharpe_ratio_quarterly = sharpe_ratio_annual / np.sqrt(4)

# Store the results in a dataframe
sharpe_ratios = np.array([sharpe_ratio_annual, sharpe_ratio_monthly, sharpe_ratio_quarterly, sharpe_ratio_daily])
sharpe_df = pd.DataFrame(sharpe_ratios, index=['Annual', 'Monthly', 'Quarterly', 'Daily'], columns=['Sharpe Ratio'])
sharpe_df.head()
```

Out[12]:

	Sharpe Ratio
Annual	0.520000
Monthly	0.150111
Quarterly	0.260000
Daily	0.032757

3.3) Convert each Sharpe Ratio into Loss Probability (daily, monthly, quarterly, annual), using

$$\Pr(\text{P\& L} < 0) = \Pr(x < -\text{SR}).$$

where x is a standard Normal random variable.

Using the above derived Loss Probability, we know that the probability that a Standard Normal random variable is less than $-\text{SR}$ can be computed using $\Phi(-\text{SR})$

In [13]:
```
# Convert each Sharpe Ratio to a Loss Probability
sharpe_df['Loss Probability %'] = sharpe_df['Sharpe Ratio'].apply(lambda x: round(ss.norm.cdf(x) * 100, 2))

# Round the Sharpe Ratios to 3 decimal places
sharpe_df['Sharpe Ratio'] = sharpe_df['Sharpe Ratio'].apply(lambda x: round(x, 3))
sharpe_df.head()
```

Out[13]:

	Sharpe Ratio	Loss Probability %
Annual	0.520	69.85
Monthly	0.150	55.97
Quarterly	0.260	60.26
Daily	0.033	51.31

Question 4:

Instead of computing the optimal allocations analytically, let's conduct an experiment. Generate above 700 random allocation sets: 4×1 vectors. Those will not be optimal and can be negative.

4.1) Standardise each set to satisfy $w'\mathbf{1} = 1$. In fact, generate 3 allocations and compute the 4 th.

4.2) For each set, compute $\mu_\Pi = \boldsymbol{w}'\boldsymbol{\mu}$ and $\sigma_\Pi = \sqrt{\boldsymbol{w}'\boldsymbol{\Sigma}\boldsymbol{w}}$.

4.3) Plot the cloud of points, μ_Π vertically on σ_Π horizontally. Explain this plot.

4.1) Generate 10,000 random portfolios (allocation sets) comprised of the 4 assets, A, B, C and D. Randomly generate 3 allocations (A, B, C), with the 4th allocation (D) computed to satisfy the budget constaint $w'\mathbf{1} = 1$.

In [14]:
```
# Set the seed for reproducibility
np.random.seed(0)

# Set the seed for reproducibility
rng = np.random.default_rng(0)

# Generate 10,000 random portfolios of shape (10000, 4) with random weights between -1 and 1
NUMBER_OF_PORTFOLIOS = 10000
assets = ['A', 'B', 'C', 'D']
portfolio_weights = (1 - (-1)) * rng.random((NUMBER_OF_PORTFOLIOS, len(assets))) - 1

# Re-compute the weights of the 4th asset 'D' to satisfy budget constraint
portfolio_weights[:, len(assets) - 1] = 1 - np.sum(portfolio_weights[:, :len(assets) - 1], axis=1)

print(f'Randomly generated weights: \n {portfolio_weights}')
```

```
Randomly generated weights:
 [[ 0.27392337 -0.46042657 -0.91805295  2.10455615]
 [ 0.62654048  0.82551115  0.21327155 -0.66532318]
 [ 0.08724998  0.87014485  0.63170711 -0.58910194]
 ...
 [ 0.84326753 -0.27922569 -0.27543331  0.71139146]
 [-0.18939823  0.31098477 -0.39738803  1.27580148]
 [ 0.00801324 -0.53890075  0.13118165  1.39970586]]
```

4.2) Compute the annualised returns, $\mu_\Pi = \boldsymbol{w}'\boldsymbol{\mu} \times N$ and annualised portfolio volatility, $\sigma_\Pi = \sqrt{\boldsymbol{w}'\boldsymbol{\Sigma}\boldsymbol{w}} \times \sqrt{N}$ where N = Annual Trading Days, using the random weights $\boldsymbol{w}$ generated in **4.1)**.

In [ ]:
```
# Our inputs for the mean returns vector and covariance matrix are the same as before:
print(f'Mean Returns Vector:\n{mean_returns_vector} \n')
print(f'Covariance Matrix:\n{covariance_matrix}')
```

In [16]:
```
# Calculate the porfolio returns and volatility
portfolio_returns = np.dot(portfolio_weights, mean_returns_vector)

# Vectorised approach using np.sum() instead of using np.dot() with portfolio_weights^T in a loop
portfolio_volatility = np.sqrt(np.sum((np.dot(portfolio_weights, covariance_matrix) * portfolio_weights), axis=1))

# Create a DataFrame for analysis
portfolio_df = pd.DataFrame({
                'Portfolio Return': portfolio_returns,
                'Portfolio Volatility': portfolio_volatility
                })

for i, asset in enumerate(assets):
    portfolio_df[asset + ' Weight'] = portfolio_weights[:, i]

portfolio_df.head()
```

Out[16]:

	Portfolio Return	Portfolio Volatility	A Weight	B Weight	C Weight	D Weight
0	0.306761	0.542018	0.273923	-0.460427	-0.918053	2.104556
1	-0.025268	0.245827	0.626540	0.825511	0.213272	-0.665323
2	0.030426	0.270835	0.087250	0.870145	0.631707	-0.589102
3	0.206255	0.313833	0.714809	-0.932829	0.459311	0.758709
4	0.112121	0.184924	0.726358	0.082922	-0.400576	0.591296

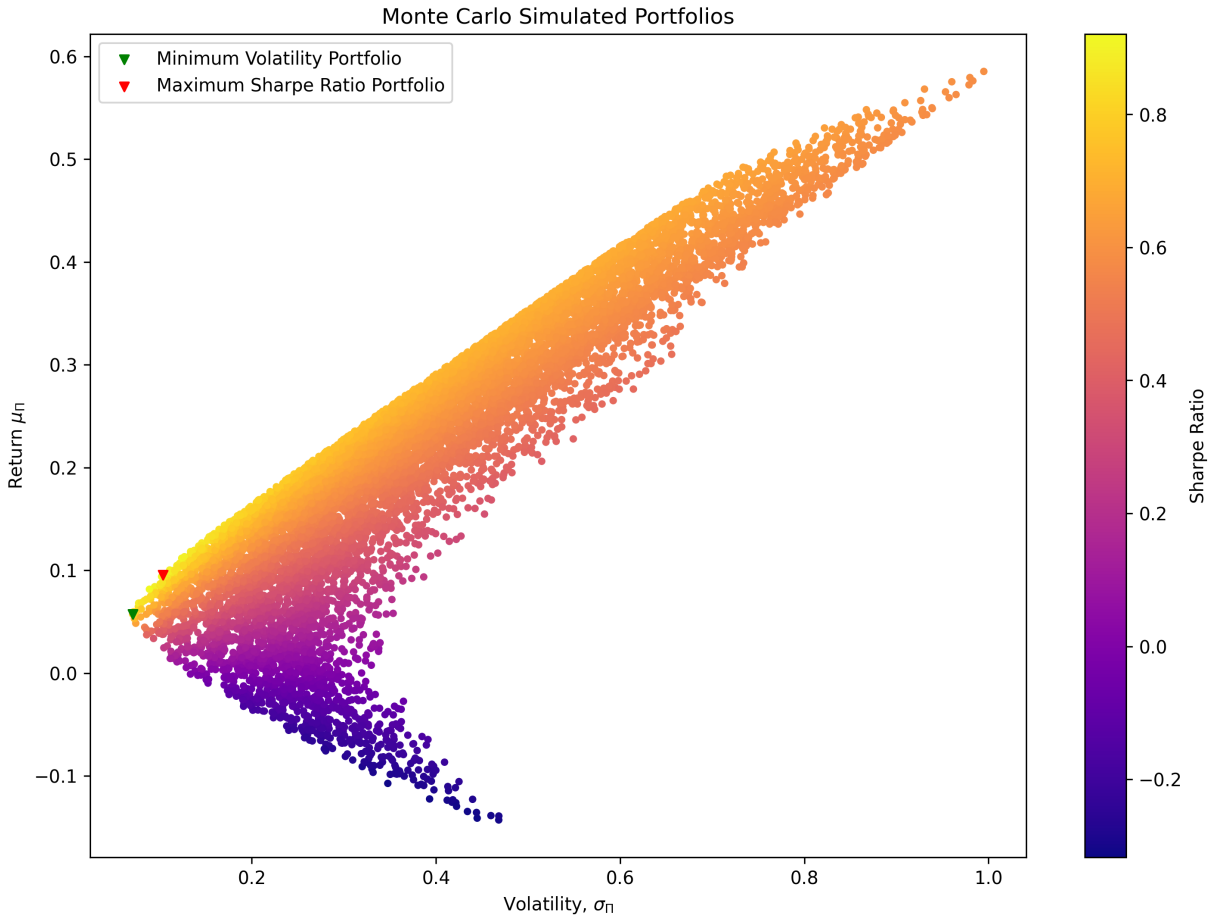
4.3) Plotting the Monte Carlo simulated portfolios with μ_Π plotted vertically and σ_Π horizontally. To help describe the plot we will identify the simulated Minimum Volatility Portfolio and Maximum Sharpe Ratio Portfolio. We will also calculate the Sharpe Ratio for each simulation and plot this as our colour scale to help describe the relationship between μ_Π and σ_Π.

In [ ]:
```
# Calculate the Sharpe Ratio for each portfolio
portfolio_df['Sharpe Ratio'] = portfolio_df['Portfolio Return'] / portfolio_df['Portfolio Volatility']
portfolio_df.head()
```

In [ ]:
```
# Find the Minimum Volatility Portfolio
min_volatility_portfolio = portfolio_df.iloc[portfolio_df['Portfolio Volatility'].idxmin()]
print(f'Minimum Volatility Portfolio:\n{min_volatility_portfolio}')
```

In [ ]:
```
# Find the Maximum Sharpe Ratio Portfolio
max_sharpe_portfolio = portfolio_df.iloc[portfolio_df['Sharpe Ratio'].idxmax()]
print(f'Max Sharpe Ratio Portfolio:\n{max_sharpe_portfolio}\n')
```

In [20]:
```
# Plot the Monte Carlo simulation using matplotlib
plt.scatter(portfolio_df['Portfolio Volatility'], portfolio_df['Portfolio Return'],
            s=10, c=portfolio_df['Sharpe Ratio'], cmap='plasma', marker='o')
plt.colorbar(label='Sharpe Ratio')

# Add the Minimum Volatility portfolio
plt.scatter(min_volatility_portfolio['Portfolio Volatility'], min_volatility_portfolio['Portfolio Return'],
            marker='v', color='g', s=25, label='Minimum Volatility Portfolio')

# Add the Max Sharpe Ratio portfolio
plt.scatter(max_sharpe_portfolio['Portfolio Volatility'], max_sharpe_portfolio['Portfolio Return'],
            marker='v', color='r', s=25, label='Maximum Sharpe Ratio Portfolio')

plt.legend(loc='upper left')
plt.xlabel('Volatility, $\sigma_{\Pi}$')
plt.ylabel('Return $\mu_{\Pi}$')
plt.title('Monte Carlo Simulated Portfolios')
plt.show()
```



Explanation of the plot:

Each dot on our plot represents one of the 10,000 randomly simulated portfolios with their randomly generated asset allocation, plotted is the portfolio's return against their volatility or 'risk' where their colour is the Sharpe Ratio of the portfolio describing the returns per unit of risk of the portfolio. Together, all of the simulated portfolios represent what we can call the oppurtunity set - the set of all possible portfolios we could create by allocating different amounts or weights to assets in our investment universe. Our oppurtunity set looks like a hyperbola as we are plotting the standard deviation or volatility μ rather than the variance of each portfolio. The shape of the oppurtunity set is also determined by the correlations between the assets. The following characterists of our plot and oppurtunity set will make it hard to achieve to achieve a high return without taking on a high level of risk:

- Tilted hyperbolic shape with low eccentricty due to the low correlation of assets all being between 0 and 0.5
- The clustering of high Sharpe Ratio portfolios at the peak of the hyperbola
- The portfolio with the Maximum Sharpe Ratio is very close to the portfolio with the Minimum Volatility along the efficient frontier, giving a short efficient frontier.

Question 5:

NASDAQ100 data provided (2017-2023) for you to implement the backtesting of 99%/10day Value at Risk and report the following:

(a) The count and percentage of VaR breaches.

(b) The count of consecutive VaR breaches. (1, 1, 1 indicates two consecutive occurrences)

(c) Provide a plot which: identifies the breaches visually (crosses or other marks) and properly labels axis X with at least years.

(d) In your own words, describe the sequence of breaches caused by COVID pandemic news in 2020-Feb versus 2020-Mar.

$$\mathrm{VaR}_{10D,t} = \text{ Factor } \times \sigma_t \times \sqrt{10}$$

- Compute the rolling standard deviation σ_t from 21 daily returns.
- Timescale of that σ_t remains 'daily' regardless of how many returns are in the sample. To make projection, use the additivity of variance $\sigma_{10D} = \sqrt{\sigma_t^2 \times 10}$.
- A breach occurs when the forward realised 10-day return is below the VaR_t quantity.

$r_{10D,t+10} < \mathrm{VaR}_{10D,t}$ means breach, given both numbers are negative.

VaR is fixed at time t and compared to the return realised from t to $t+10$, computed $\ln(S_{t+10}/S_t)$. Alternatively, you can compare to $\ln(S_{t+11}/S_{t+1})$ but state this assumption in your report upfront.

5 (a) For a 99%/10 day Value at Risk $c = 0.99$.

Assuming Factor is a percentile of the Standard Normal Distribution cutting 1% on the tail:

$$\mathrm{Factor} = \Phi^{-1}(1-c) = \Phi^{-1}(1-0.99) = \Phi^{-1}(0.01) \approx -2.32635$$

We will then use the NASDAQ100 data provided to calculate the daily log returns and from this the rolling standard deviation σ_t from 21 daily returns.

In [ ]:
```
# Calculate Analytical Factor
var_confidence = 0.99
factor = ss.norm.ppf(1 - var_confidence)
print("Factor: ", factor)
```

In [ ]:
```
# Load in the data from the CSV file
nasdaq_df = pd.read_csv('nasdaq100.csv', index_col='Date', parse_dates=True)

# Column to count the days for clarity doing 21 day rolling window
nasdaq_df['Day'] = np.arange(len(nasdaq_df))

# Calculate daily log returns as we use Analytical Factor
nasdaq_df['Log Return'] = np.log(nasdaq_df['Closing Price'] / nasdaq_df['Closing Price'].shift(1))

nasdaq_df.dropna(inplace=True)
nasdaq_df.head()
```

In [ ]:
```
# Compute the rolling standard deviation, from 21 daily log returns
nasdaq_df['Rolling Std'] = nasdaq_df['Log Return'].rolling(window=21).std()
nasdaq_df.head(25)
```

Compute the $\mathrm{VaR}_{10D,t}$ using the 21 day rolling standard deviations calculated as:

$$\mathrm{VaR}_{10D,t} = \text{ Factor } \times \sigma_t \times \sqrt{10}$$

In [ ]:
```
# Calcualte VaR_10D
nasdaq_df['VaR 10D'] = factor * nasdaq_df['Rolling Std'] * np.sqrt(10)

# Drop NaN values
nasdaq_df.dropna(inplace=True)

nasdaq_df.head()
```

To count breaches we need to count how many time $r_{10D+1,t+11} < \mathrm{VaR}_{10D,t}$ is satisfied. To avoid overlap with our rolling window for standard deviation, σ_t, we will compute returns realised from $t+1$ to $t+11$:

$$\ln(S_{t+11}/S_{t+1})$$

In [ ]:
```
# Compute the 10 day log return as ln(S_t+11 / S_t+1)
nasdaq_df['10D Log Return'] = np.log(nasdaq_df['Closing Price'].shift(-11) / nasdaq_df['Closing Price'].shift(-1))
nasdaq_df.head()
```

In [ ]:
```
# Breach = 1, No Breach = 0 if 10D Log Return < VaR_10D and both 10D Log Return and VaR_10D are negative
nasdaq_df['Breach'] = np.where((nasdaq_df['10D Log Return'] < nasdaq_df['VaR 10D'])
```

```
                                & (nasdaq_df['10D Log Return'] < 0)
                                & (nasdaq_df['VaR 10D'] < 0), 1, 0)

nasdaq_df.head()
```

The count and percentage of VaR breaches is then calculated as follows:

In [27]:
```
# Compute number and percentage of breaches
breach_count = nasdaq_df['Breach'].value_counts().iloc[1]
breach_percentage = nasdaq_df['Breach'].mean()

print(tabulate.tabulate([['Number of breaches', breach_count],
                         ['Percentage of breaches', f'{breach_percentage:.2%}']],
                         tablefmt='fancy_grid'))
```

```
╒═══════════════════════╤═══════╕
│ Number of breaches    │ 41    │
├───────────────────────┼───────┤
│ Percentage of breaches │ 3.02% │
╘═══════════════════════╧═══════╛
```

5 (b) Compute the count of consecutive VaR breaches. (1, 1, 1 indicates two consecutive occurrences)

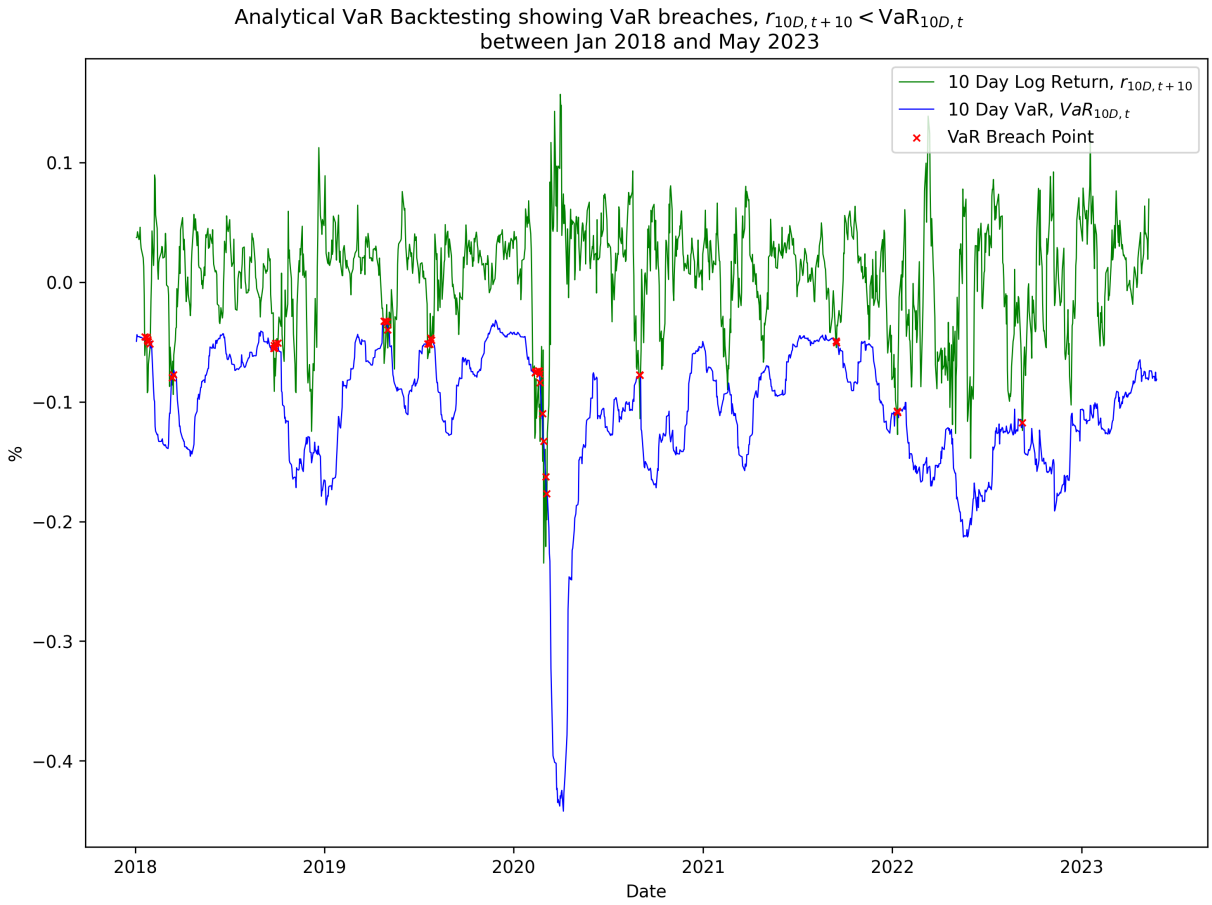
In [28]:
```
# First extract the breach column to a numpy array
breaches = nasdaq_df['Breach'].to_numpy()

# Define a function to count the number of consecutive breaches for re-use in Question 6
def count_Consecutive_Breaches(array: np.ndarray) -> int:
    ''' Counts the number of consecutive breaches in an array of 0s and 1s where [1, 1, 1]
    would return 2 consecutive breaches
    Args:
        array (np.ndarray): Array of 0s and 1s
    Returns:
        breaches (int): Total number of consecutive breaches
    '''
    # Use numpy diff with numpy where to find the indices where the value changes, add 1 as diff checks previous value
    change_indicies = np.where(np.diff(array) != 0)[0]+1

    # Use numpy split to keep only consecutive 1s
    consecutives = np.split(array, change_indicies)

    # Drop arrays without a 1 or contain only a single 1 get length - 1 as [1, 1, 1] is 2 consecutive breaches
    consecutives = np.array([len(sub_array) - 1 for sub_array in consecutives if 1 in sub_array and len(sub_array) > 1])

    # Return the sum of the array as the number of consecutive breaches
    return consecutives.sum()

# Print the number of consecutive breaches
print(f'Count of consecutive breaches: {count_Consecutive_Breaches(breaches)}')
```

```
Count of consecutive breaches: 22
```

5 (c) Plot 10 Day Log Return, 10 Day VaR and identify the breaches using red crosses.

In [ ]:
```
# Create a new column that plots the VaR 10D point where the breach occurs, otherwise NaN
nasdaq_df['VaR 10D Breach'] = np.where(nasdaq_df['Breach'] == 1, nasdaq_df['VaR 10D'], np.nan)
nasdaq_df.head()
```

In [30]:
```
def plot_VaR_Breaches(df: pd.DataFrame):
    '''Given a dataframe containing 10 day log returns, 10 day VaR and VaR 10 day breaches, plot this data
    Args:
        df (pd.DataFrame): DataFrame containing 10 day log returns, 10 day VaR and VaR 10 day breaches
    '''
    start_date = df.index[0].strftime('%b %Y')
    end_date = df.index[-1].strftime('%b %Y')

    plt.plot(df.index, df['10D Log Return'], label='10 Day Log Return, $r_{10 D, t+10}$',
             color='green', linewidth=0.7, zorder=0)
    plt.plot(df.index, df['VaR 10D'], label='10 Day VaR, ${VaR}_{10 D, t}$',
             color='blue', linewidth=0.7, zorder=0)
    plt.scatter(df.index, df['VaR 10D Breach'], label='VaR Breach Point',
                color='red', marker='x', s=15, zorder=1, linewidth=1)

    plt.xlabel('Date')
    plt.ylabel('%')
    plt.legend(loc='upper right')
    plt.title(f'Analytical VaR Backtesting showing VaR breaches, $r_{{10 D, t+10}}<\operatorname{{VaR}}_{{10 D, t}}$ \
              \n between {start_date} and {end_date}')
    plt.show()

plot_VaR_Breaches(nasdaq_df)
```

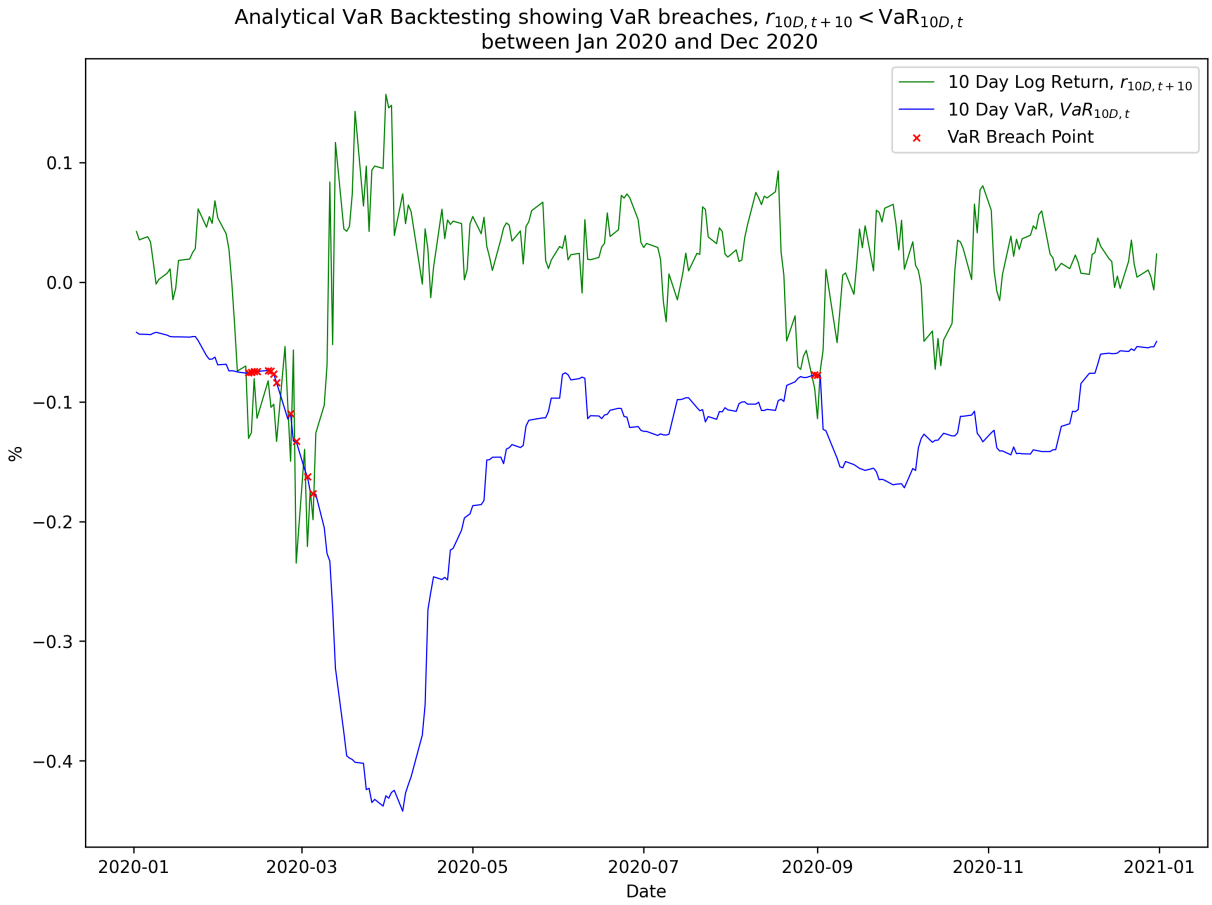


5 (d) In your own words, describe the sequence of breaches caused by COVID pandemic news in 2020-Feb versus 2020-Mar.

First we'll take a closer look at 2020 and plot the Analytical VaR backtest for that year.:

In [ ]:
```
# Filter the dataframe between 2020 and 2021
covid_df = nasdaq_df[(nasdaq_df.index >= '2020-01-01') & (nasdaq_df.index <= '2021-01-01')]
covid_df.head()
```

In [32]:
```
# Plot the VaR breaches in 2020
plot_VaR_Breaches(covid_df)
```



Explanation of the plot:

The sequence of breaches seen between February 2020 and March 2020 can clearly be seen in the above plot. These breaches would have been caused by the COVID news and the subsequent market crash. The crash is illustrated by the sharp decrease in 10 day log returns of the Nasdaq index causing it to fall below the 10 day VaR level in this period. As 10 day VaR looks forward and therefore lags compared to the 10 day log returns, the crash is immediately reflected in the log returns at the time of the news announcement between February and March where most of the breaches are clustered. Going further forward in time to March and April, the 10 Day VaR reflects the news and crashes down below -0.4%. This is because it starts to capture the higher volatilities seen in the market back in Feburary, 10 day or more prior.

Question 6:

Re-implement backtesting using the method above, recompute $\text{VaR}_{10D,t}$ but, with the input of EWMA σ^2_{t+1}. Use the variance for the entire dataset to initialise the scheme.

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

with $\lambda = 0.72$ value set to minimise out of sample forecasting error.

Hint: computation of EWMA σ^2_{t+1} is not sufficient, proceed to compute $\text{VaR}_{10D,t}$ and count breaches in VaR.

(a-c) Provide the same deliverables (a), (b) and (c) as in the previous Question.

(d) Briefly (3-4 lines) discuss the impact of λ on smoothness of EWMA-predicted volatility.

6 (a) Need to calculate the the standard deviation, σ_t of the returns but using the RiskMetrics EWMA σ^2_{t+1} approach:

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

$$\sigma_{t+1|t} = \sqrt{\lambda\sigma^2_{t|t-1} + (1-\lambda)r^2_t}$$

$$\text{where } \lambda = 0.72 \text{ and } r^2_t = \text{the squared return at time t}$$

At time $t = 0$ we will assume that our initial variance estimate is the average variance of the sample data. This is a fair assumption as the data spans a long period of time:

$$\sigma^2_0 = \mathbb{E}\left[\sigma^2\right] = \mathbb{E}\left[r^2_t\right]$$

In [ ]:
```
# Load in the data from the CSV file
risk_metrics_df = pd.read_csv('nasdaq100.csv', index_col='Date', parse_dates=True)

# Calculate daily log returns and squared log returns
risk_metrics_df['Log Return'] = np.log(risk_metrics_df['Closing Price'] / risk_metrics_df['Closing Price'].shift(1))
risk_metrics_df['Squared Return'] = risk_metrics_df['Log Return'] ** 2

# Save the average variance in a variable
average_variance = risk_metrics_df['Squared Return'].mean()

print(f'Average Variance: {average_variance}')

risk_metrics_df.dropna(inplace=True)
risk_metrics_df.head()
```

In [ ]:
```
# Set lambda value for EWMA function
ewma_lambda = 0.72

# Create a new column and set the first value to the average variance
risk_metrics_df['Variance Estimate'] = np.nan
risk_metrics_df['Variance Estimate'].iloc[0] = average_variance

# Loop through the rows using the previous row's variance estimate with the EWMA function
for i in range(1, len(risk_metrics_df)):
    risk_metrics_df['Variance Estimate'].iloc[i] = (ewma_lambda * risk_metrics_df['Variance Estimate'].iloc[i-1]) \
                                                   + (1 - ewma_lambda) * risk_metrics_df['Squared Return'].iloc[i-1]

# Calculate the 10 day standard deviation
risk_metrics_df['10D Std'] = np.sqrt(risk_metrics_df['Variance Estimate'] * 10)

# 10 day VaR at the same confidence level, therefore use same factor
risk_metrics_df['VaR 10D'] = risk_metrics_df['10D Std'] * factor

risk_metrics_df.head()
```

In [35]:
```
# Compute 10 day log return as ln(S_t+11 / S_t+1)
risk_metrics_df['10D Log Return'] = np.log(risk_metrics_df['Closing Price'].shift(-11)
                                           / risk_metrics_df['Closing Price'].shift(-1))

# Breach = 1, No Breach = 0 if 10D Log Return < VaR 10D and both 10D Log Return and VaR_10D are negative
risk_metrics_df['Breach'] = np.where((risk_metrics_df['10D Log Return'] < risk_metrics_df['VaR 10D'])
                                     & (risk_metrics_df['10D Log Return'] < 0)
                                     & (risk_metrics_df['VaR 10D'] < 0), 1, 0)

risk_metrics_df.head()
```

Out[35]:

Date	Closing Price	Log Return	Squared Return	Variance Estimate	10D Std	VaR 10D	10D Log Return	Breach
2017-12-04	6263.700195	-0.011772	1.385723e-04	0.000269	0.051902	-0.120742	0.033828	0
2017-12-05	6265.109863	0.000225	5.063770e-08	0.000233	0.048245	-0.112234	0.028114	0
2017-12-06	6293.049805	0.004450	1.979978e-05	0.000168	0.040939	-0.095238	0.024461	0
2017-12-07	6316.279785	0.003685	1.357609e-05	0.000126	0.035527	-0.082647	0.018830	0
2017-12-08	6344.569824	0.004469	1.997114e-05	0.000095	0.030769	-0.071580	0.006123	0

The count and percentage of VaR breaches is then calculated as follows. We will filter the dataframe to the same date range as in Question 5 so that the deliverable is the same and the results are comparable.

In [36]:
```
# Filter the dataframe betwee the same dates as the Question 5 final dataframe
start_date = nasdaq_df.index[0]
end_date = nasdaq_df.index[-1]

risk_metrics_df = risk_metrics_df.loc[start_date:end_date]

# Compute number and percentage of breaches
breach_count = risk_metrics_df['Breach'].value_counts().iloc[1]
breach_percentage = risk_metrics_df['Breach'].mean()

print(tabulate.tabulate([['Number of breaches', breach_count],
                         ['Percentage of breaches', f'{breach_percentage:.2%}']],
                         tablefmt='fancy_grid'))
```

```
╒═══════════════════════╤═══════╕
│ Number of breaches    │ 54    │
├───────────────────────┼───────┤
│ Percentage of breaches │ 3.97% │
╘═══════════════════════╧═══════╛
```

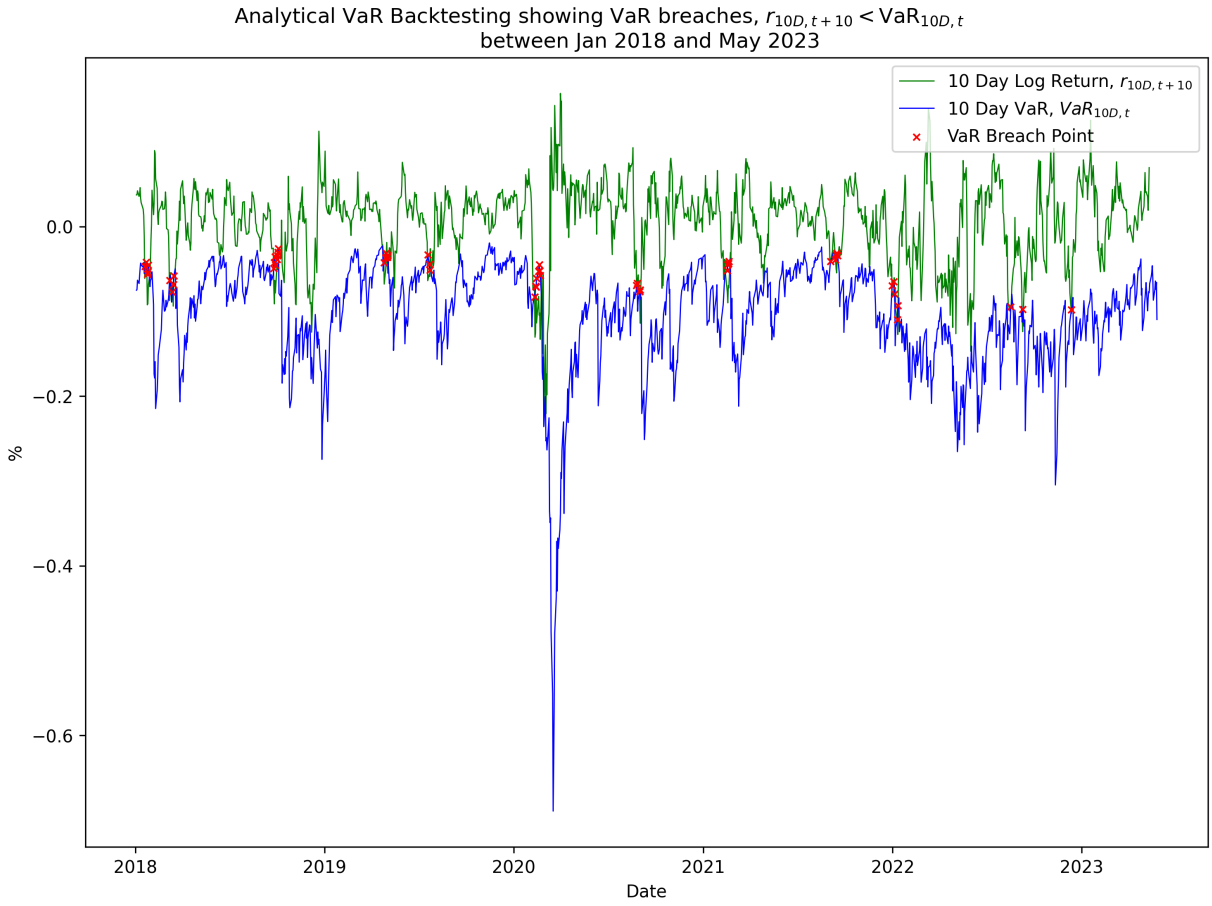
6 (b) Compute the count of consecutive VaR breaches. (1, 1, 1 indicates two consecutive occurrences)

In [37]:
```
# First extract the breach column to a numpy array
risk_metrics_breaches = risk_metrics_df['Breach'].to_numpy()

# Print the number of consecutive breaches
print(f'Count of consecutive breaches: {count_Consecutive_Breaches(risk_metrics_breaches)}')
```

```
Count of consecutive breaches: 32
```

6 (c) Plot 10 Day Log Return, 10 Day VaR when using the RiskMetrics approach and identify the breaches using red crosses.

In [38]:
```
# Create a new column that plots the VaR 10D point where the breach occurs, otherwise NaN
risk_metrics_df['VaR 10D Breach'] = np.where(risk_metrics_df['Breach'] == 1, risk_metrics_df['VaR 10D'], np.nan)

# Plot the 10 day log returns, 10 day Var and breach points
plot_VaR_Breaches(risk_metrics_df)
```



6 (d) The impact of λ on smoothness of EWMA-predicted volatility. Given the formula for EWMA-predicted volatility:

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

Looking closer at the formula for EWMA-predicted volatility, the choice of λ will make our EWMA-predicted volatility either more responsive to previous returns, less response to previous volatility and therefore more volatile (smaller λ) or less responsive to previous returns, more responsive to previous volatility and therefore more smooth (higher λ). This is because as λ increases, the weighting of previous returns in the calculation for EWMA-predicted volatility decreases, and vice versa:

$$\lim_{\lambda\to 1} \sigma^2_{t+1|t} = \sigma^2_{t|t-1}$$

$$\lim_{\lambda\to 0} \sigma^2_{t+1|t} = r^2_t$$