

Applied Morden Portfolio Theory

Optimal Portfolio Allocation

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

Asset	μ	σ	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

And:

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

Part 1

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

$$\underset{w}{\operatorname{argmin}} \frac{1}{2} w^\top \Sigma w \quad \text{s.t. } w^\top \mathbf{1} = 1$$

- Derive the analytical solution for optimal allocations w^* . Provide full mathematical workings.
- In the derivation, include the formula derivation for the Lagrangian multiplier.
- Compute optimal allocations (Global MV portfolio) for the given investment universe.

Let's consider portfolio selection problem is:

$$\min_w \frac{1}{2} w^\top \Sigma w$$

Consider a market with 4 asset classes X_1, X_2, X_3 and X_4 . Their return vector and standard deviation vector are respectively given by:

$$\mu = \begin{pmatrix} 0.05 \\ 0.07 \\ 0.15 \\ 0.22 \end{pmatrix} \quad \sigma = \begin{pmatrix} 0.07 \\ 0.28 \\ 0.25 \\ 0.31 \end{pmatrix}$$

And the correlation between the asset returns is R

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

With vector σ we first form a matrix with standard deviations on its diagonal and 0 everywhere else:

$$S = D(\sigma) = \begin{pmatrix} 0.07 & 0 & 0 & 0 \\ 0 & 0.28 & 0 & 0 \\ 0 & 0 & 0.25 & 0 \\ 0 & 0 & 0 & 0.31 \end{pmatrix}$$

The covariance matrix Σ can be obtained by pre and post multiplying the correlation matrix by the diagonal standard deviation matrix:

$$\Sigma = SRS$$

$$\Sigma = \begin{pmatrix} 0.07 & 0 & 0 & 0 \\ 0 & 0.28 & 0 & 0 \\ 0 & 0 & 0.25 & 0 \\ 0 & 0 & 0 & 0.31 \end{pmatrix} \times \begin{pmatrix} 1 & 0.4 & 0.3 & 0.3 \\ 0.4 & 1 & 0.27 & 0.42 \\ 0.3 & 0.27 & 1 & 0.5 \\ 0.3 & 0.42 & 0.5 & 1 \end{pmatrix} \times \begin{pmatrix} 0.07 & 0 & 0 & 0 \\ 0 & 0.28 & 0 & 0 \\ 0 & 0 & 0.25 & 0 \\ 0 & 0 & 0 & 0.31 \end{pmatrix}$$
$$\Sigma = \begin{pmatrix} 0.0049 & 0.00784 & 0.00525 & 0.00651 \\ 0.00784 & 0.0784 & 0.0189 & 0.036456 \\ 0.00525 & 0.0189 & 0.0625 & 0.03875 \\ 0.00651 & 0.036456 & 0.03875 & 0.0961 \end{pmatrix}$$

Consider an optimal allocation weights is a vector w^* such that

$$w^{*\top} \mathbf{1} = 1$$

For $\mathbf{1}$ is a vector of 1s is our constrain to this portfolio optimization problem.

$$\mathbf{1} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

With vector w^* , form a Lagrange function with a Lagrange multipliers λ :

$$L(w^*, \lambda) = \frac{1}{2} w^{*\top} \Sigma w^* + \lambda (1 - w^{*\top} \mathbf{1}) \tag{1}$$

Then to solve this problem, we would need to solve the necessary condition is equal zero:

$$\nabla f(w^*) = 0$$

And sufficient condition must be positive definite:

$$Hf(w^*) > 0$$

First order condition by taking the derivative with respect to the vector w^* :

$$\frac{\partial L}{\partial w^*}(w^*, \lambda) = \Sigma w^* - \lambda \mathbf{1} = 0 \tag{2}$$

Second order condition with respect to the vector w^* :

$$\frac{\partial^2 L}{\partial w^{*2}} = \Sigma$$

Let's check if Σ is positive definite

PYTHON

```
import numpy as np

Sigma = np.array([[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  ]])

eigenvalues, _ = np.linalg.eig(Sigma)

print([eigenvalue.round(4) for eigenvalue in eigenvalues])
```

PYTHON

```
[0.1471, 0.0039, 0.0546, 0.0363]
```

All the eigenvalues are positive, therefore the sufficient condition is met.

From (2), we can solve for optimal portfolio allocation w^* :

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

Substitute (3) into the constraint:

$$\begin{aligned} w^{*\top} \mathbf{1} &= 1 \\ (\Sigma^{-1} \lambda \mathbf{1})^\top \mathbf{1} &= 1 \\ \lambda \left((\Sigma^{-1} \mathbf{1})^\top \mathbf{1} \right) &= 1 \end{aligned}$$

We can compute λ with given correlation matrix Σ :

Let's compute $(\Sigma^{-1} \mathbf{1})^\top \mathbf{1}$ and solve for λ :

PYTHON

```
# Calculate the inverse of Sigma
Sigma_inv = np.linalg.inv(Sigma)

# Perform the multiplications
result = np.dot(np.dot(Sigma_inv, vector_1).T, vector_1)

# Calculate lambda
lambda_value = 1 / result[0, 0]

print("Lambda =", lambda_value)
```

PYTHON

```
Lambda = 0.004768086485896824
```

Substituting λ to (3) and compute the vector w^* :

$$w^* = \lambda \times \begin{pmatrix} 0.0049 & 0.00784 & 0.00525 & 0.00651 \\ 0.00784 & 0.0784 & 0.0189 & 0.036456 \\ 0.00525 & 0.0189 & 0.0625 & 0.03875 \\ 0.00651 & 0.036456 & 0.03875 & 0.0961 \end{pmatrix}^{-1} \times \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

PYTHON

```
w_star = np.dot(lambda_value, np.dot(Sigma_inv,vector_1))
print(w_star)
```

PYTHON

```
[[ 1.04311432]
 [-0.04131645]
 [ 0.00599093]
 [-0.0077888 ]]
```

The optimal allocation for the portfolio subject to the budget constraint is:

$$w^* = \begin{pmatrix} 1.04311432 \\ -0.04131645 \\ 0.00599093 \\ -0.0077888 \end{pmatrix}$$

Part 2

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

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

- Compute w^* and portfolio risk $\sigma_\Pi = \sqrt{w^\top \Sigma w}$ for $m = 7\%$ for three levels of correlation.
- 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.

Let's compute the new matrix Σ_1, Σ_2 and Σ_3 with scaling factor $\times 1, \times 1.3, \times 1.8$, subject to individual correlation upper limit of 0.99.

PYTHON

```
# Scaling factors
scaling_factors = [1, 1.3, 1.8]

# Apply scaling and constraints using map() function
new_corr_matrices = list(map(lambda factor: np.clip(R * factor, None, 0.99), scaling_factors))

# Set diagonal elements back to 1 for each new correlation matrix
for matrix in new_corr_matrices:
    np.fill_diagonal(matrix, 1)

# Compute new sigma

Sigma_1 = Sigma
Sigma_2 = np.dot(np.dot(S, new_corr_matrices[1]), S)
Sigma_3 = np.dot(np.dot(S, new_corr_matrices[2]), S)
```

The new matrixes:

$$\Sigma_1 = \begin{pmatrix} 0.0049 & 0.00784 & 0.00525 & 0.00651 \\ 0.00784 & 0.0784 & 0.0189 & 0.036456 \\ 0.00525 & 0.0189 & 0.0625 & 0.03875 \\ 0.00651 & 0.036456 & 0.03875 & 0.0961 \end{pmatrix}$$

$$\Sigma_2 = \begin{pmatrix} 0.0049 & 0.010192 & 0.006825 & 0.008463 \\ 0.010192 & 0.0784 & 0.02457 & 0.0473928 \\ 0.006825 & 0.02457 & 0.0625 & 0.050375 \\ 0.008463 & 0.0473928 & 0.050375 & 0.0961 \end{pmatrix}$$

$$\Sigma_3 = \begin{pmatrix} 0.0049 & 0.014112 & 0.00945 & 0.011718 \\ 0.014112 & 0.0784 & 0.03402 & 0.0656208 \\ 0.00945 & 0.03402 & 0.0625 & 0.06975 \\ 0.011718 & 0.0656208 & 0.06975 & 0.0961 \end{pmatrix}$$

With the new constrain $w^\top \mu = m = 7\%$, form a Lagrangian function with two Lagrange multipliers λ and γ :

$$L(w, \lambda, \gamma) = \frac{1}{2} w^\top \Sigma w + \lambda(m - w^\top \mu) + \gamma(1 - w^\top \mathbf{1}) \quad (4)$$

From **Question 1**, we know that (4) must satisfy both necessary condition and sufficient condition. All 3 covariance matrixes are positive definite, therefore, we only need to check for the necessary condition.

With Σ_1 :

The optimal vector w_1^* is:

$$w_1^* = \Sigma_1^{-1}(\lambda\mu + \gamma\mathbf{1}) \quad (5)$$

with vector μ is:

$$\mu = \begin{pmatrix} 0.05 \\ 0.07 \\ 0.15 \\ 0.22 \end{pmatrix}$$

Now we find the values for λ and γ :

Substituting w_1^* into the constraints, we get:

$$\mu^\top \Sigma_1^{-1}(\lambda\mu + \gamma\mathbf{1}) = \lambda\mu^\top \Sigma_1^{-1}\mu + \gamma\mu^\top \Sigma_1^{-1}\mathbf{1} = m = 0.07$$

$$\mathbf{1}^\top \Sigma_1^{-1}(\lambda\mu + \gamma\mathbf{1}) = \lambda\mathbf{1}^\top \Sigma_1^{-1}\mu + \gamma\mathbf{1}^\top \Sigma_1^{-1}\mathbf{1} = 1$$

For convenience, we define the following scalars:

$$\begin{cases} A = \mathbf{1}^\top \Sigma_1^{-1}\mathbf{1} \\ B = \mu^\top \Sigma_1^{-1}\mathbf{1} = \mathbf{1}^\top \Sigma_1^{-1}\mu \\ C = \mu^\top \Sigma_1^{-1}\mu \end{cases}$$

Note also that $AC - B^2 > 0$.

The previous system of equations for the Lagrange multipliers becomes:

$$\begin{aligned} C\lambda + B\gamma &= m \\ B\lambda + A\gamma &= 1 \end{aligned}$$

Then:

$$\begin{cases} \lambda = \frac{Am - B}{AC - B^2} \\ \gamma = \frac{C - Bm}{AC - B^2} \end{cases} \quad (6)$$

Substitute this back to (5) and we can compute the optimal portfolio with the new constraints.

We can write a python function find the optimal allocation for w^* :

```
def optimize(Sigma, mu, m):
    vector_1 = np.ones((4, 1))

    # Compute A, B, and C
    A = np.dot(np.dot(vector_1.T, Sigma_inv), vector_1)
    B = np.dot(np.dot(vector_1.T, Sigma_inv), mu)
    C = np.dot(np.dot(mu.T, Sigma_inv), mu)

    # Compute lambda and gamma
    denominator = A*C - B**2
    lambda_val = (A*m - B) / denominator
    gamma = (C - B*m) / denominator

    # Compute w_star
    w_star = np.dot(Sigma_inv, (lambda_val * mu + gamma * vector_1))

    return w_star
```

We can input Sigma_1, Sigma_2 and Sigma_3 that we compute above and find the new optimal portfolio w^* for each correlation matrix:

```
w_1_star = optimize(Sigma_1, mu, m)
w_2_star = optimize(Sigma_2, mu, m)
w_3_star = optimize(Sigma_3, mu, m)
```

Result:

$$w_1^* = \begin{pmatrix} 0.92413546 \\ -0.07289408 \\ 0.05472976 \\ 0.09402886 \end{pmatrix}$$

$$w_2^* = \begin{pmatrix} 0.99647143 \\ -0.13512639 \\ 0.01241165 \\ 0.12624331 \end{pmatrix}$$

$$w_3^* = \begin{pmatrix} 1.45087772 \\ -0.40770337 \\ -0.50705295 \\ 0.4638786 \end{pmatrix}$$

Allocation of asset given the target return $m = \mu_{\Pi} = 7\%$

Asset	μ	σ	w_1^*	w_2^*	w_3^*
A	0.05	0.07	0.92413546	0.99647143	1.45087772
B	0.07	0.28	-0.07289408	-0.13512639	-0.40770337
C	0.15	0.25	0.05472976	0.01241165	-0.50705295
D	0.22	0.31	0.09402886	0.12624331	0.4638786

Compute portfolio risks $\sigma_{\Pi} = \sqrt{w^{\top} \Sigma w}$

```
vol_1 = np.sqrt(np.dot(np.dot(w_1_star.T, Sigma_1), w_1_star))
vol_2 = np.sqrt(np.dot(np.dot(w_2_star.T, Sigma_2), w_2_star))
vol_3 = np.sqrt(np.dot(np.dot(w_3_star.T, Sigma_3), w_3_star))
```

```
Volatily for portfolio 1 [[0.07741211]]
Volatily for portfolio 2 [[0.07648599]]
Volatily for portfolio 3 [[0.04124301]]
```

Understanding Risk

Part 3

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.

- Standardise each set to satisfy $w^{\top} \mathbf{1} = 1$. In fact, generate 3 allocations and compute the 4th.
- For each set, compute $\mu_{\Pi} = w^{\top} \mu$ and portfolio risks $\sigma_{\Pi} = \sqrt{w^{\top} \Sigma w}$
- Plot the cloud of points, μ_{Π} vertically on μ_{Π} horizontally.

Consider portfolio II that has portfolio mean return $\mu_{\Pi} = w^{\top} \mu$ and portfolio risk $\sigma_{\Pi} = \sqrt{w^{\top} \Sigma w}$.

Generate 700 allocation sets and then put all the results to 1 table `portfolio_df` :

```
import pandas as pd

# Given data
returns = mu
volatility = np.array([[0.07], [0.28], [0.25], [0.31]])

# Number of random portfolios to generate
num_portfolios = 700

# Initialize arrays to store portfolio statistics
portfolio_rets = 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 random weights
    weights = np.random.random(4)
    # Normalize to ensure sum of weights is 1
    weights /= np.sum(weights)

    # Compute portfolio mean and variance
    portfolio_ret = np.dot(weights, returns)
    portfolio_vol = np.sqrt(np.dot(weights, np.dot(Sigma, weights.T)))

    # Calculate Sharpe ratio
    sharpe_ratio = portfolio_ret / portfolio_vol

    # Store results
    portfolio_rets[i] = portfolio_ret
    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_rets,
    "Volatilities": portfolio_vols,
    "Sharpe Ratios": sharpe_ratios
})

# Results
portfolio_df.head()
```

Returns μ_{Π}	Volatilities σ_{Π}	Sharpe Ratios
0.142888	0.182650	0.782301
0.110979	0.175880	0.630991
0.161582	0.211011	0.765749
0.100470	0.144007	0.697674
0.126827	0.162950	0.778317

Use that data to plot the cloud of portfolios

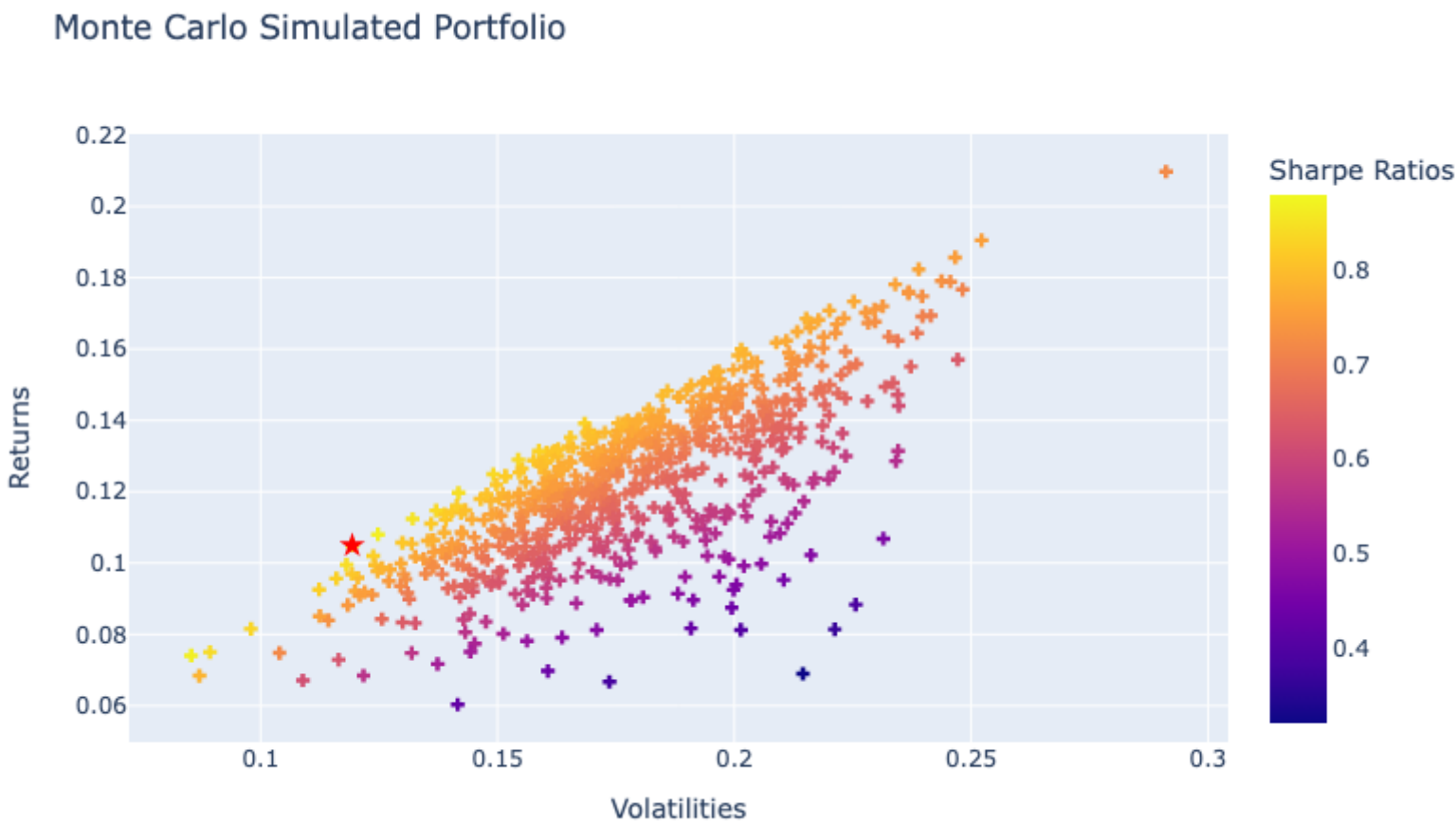
```
# Import plotly express for EF plot
import plotly.express as px

# Find the index of the portfolio with the maximum Sharpe ratio
max_sharpe_index = np.argmax(sharpe_ratios)

# Plot the data using Plotly Express
fig = px.scatter(portfolio_df, x="Volatilities", y="Returns", color="Sharpe Ratios",
                 title="Monte Carlo Simulated Portfolio",
                 labels={"Portfolio Volatilities": "Volatility",
                        "Portfolio Returns": "Return"},
                 width=800, height=500).update_traces(mode='markers', marker=dict(symbol='cross'))

# Add the portfolio with the maximum Sharpe ratio to the plot with a star symbol
fig.add_scatter(
    mode='markers',
    x=[portfolio_df.iloc[max_sharpe_index]['Volatilities']],
    y=[portfolio_df.iloc[max_sharpe_index]['Returns']],
    marker=dict(symbol='star', size=10, color='red'),
    name='Max Sharpe').update(layout_showlegend=False)

fig.show()
```



In this plot, we can see all the portfolio that we simulated from our Monte Carlo simulation. From this simulation, we can evaluate Sharpe Ratio to find the best perform portfolio allocation, in this case is around 0.88 and can easily spot with a star symbol.

Part 4

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

- The count and percentage of VaR breaches.
- The count of consecutive VaR breaches. (1, 1, 1 indicates two consecutive occurrences)
- Provide a plot which: identifies the breaches visually (crosses or other marks) and properly labels axis X with at least years.
- In your own words, describe the sequence of breaches caused by COVID pandemic news in 2020-Feb versus 2020-Mar.

VaR formula

$$VaR_{10D,t} = Factor \times \sigma_t \times \sqrt{10}$$

First, compute the value of **Factor** based on the given confident level 99%:

$$Factor = \Phi^{-1}(1 - 0.99)$$

```
from scipy.stats import norm
# Compute `factor`
factor = norm.ppf(0.01)
```

Output: −2.3263478740408408

Now, we can create a data frame `df` contains the data from `nasdaq100.csv` file.

We then calculate the log normal returns as well as 10 Days VaR using the sampling window of 21 days. Also compute the 10 Days forward returns and put everything into the data frame.

We also identify the breaches using the given condition

$$r_{10D,t+10} < VaR_{10D,t}$$

Implement everything into this python code

PYTHON

```
# Compute returns
df = pd.read_csv("../exam/nasdaq100.csv", parse_dates=["Date"], index_col="Date")
df['rets'] = np.log(df['Closing Price']).diff()
df.dropna(inplace=True)

# Compute 10 days VaR
window = 21
df['10D_VaR'] = factor * df['rets'].rolling(window=window).std() * np.sqrt(10)

# Compute 10 days forward returns
df['10D_rets'] = np.log(df['Closing Price'].shift(-(11)) / df['Closing Price'].shift(-1))

# Identify breaches
df['breach'] = (df['10D_rets'] < df['10D_VaR']).astype(int)
```

Output:

Date	Closing Price	rets	10D_VaR	10D_rets	breach
2018-01-03	6575.799805	0.009851	-0.049461	0.037228	0
2018-01-04	6584.580078	0.001334	-0.043927	0.037320	0
2018-01-05	6653.290039	0.010381	-0.045574	0.042063	0
2018-01-08	6676.629883	0.003502	-0.045511	0.035512	0
2018-01-09	6677.939941	0.000196	-0.045675	0.037362	0

We can count the number of breaches by counting the number of columns that has `breach` value equal 1.

PYTHON

```
mask = df['breach'] == 1
pct_var_breaches = df[mask]['breach'].count() / len(df)
print('No. of Breaches', df[mask]['breach'].count())
print('Percentage VaR Breaches', pct_var_breaches)
```

PYTHON

```
No. of Breaches 41
Percentage VaR Breaches 0.02973168963016679
```

We can create an algorithm to count number of consecutive breaches and then put it in the list. The algorithm takes a list of `breach` values from `df` and checks if two consecutive days have a `breach` value of 1. If both days have a 'breach' of 1, it adds a 1 to the `consecutive_counts_list`. If either day has a `breach` value other than 1. Finally, it sums up the `consecutive_counts_list` to count how many times there were consecutive breaches of 1 in the `breach` column.

PYTHON

```
# Convert the breach column to a list for easier manipulation
breach_list = df['breach'].tolist()

# Initialize variables
consecutive_counts_list = []

# Iterate through the list of breach values
for i in range(len(breach_list) - 1):
    if breach_list[i] == 1 and breach_list[i + 1] == 1:
        consecutive_counts_list.append(1)
    else:
        consecutive_counts_list.append(0)

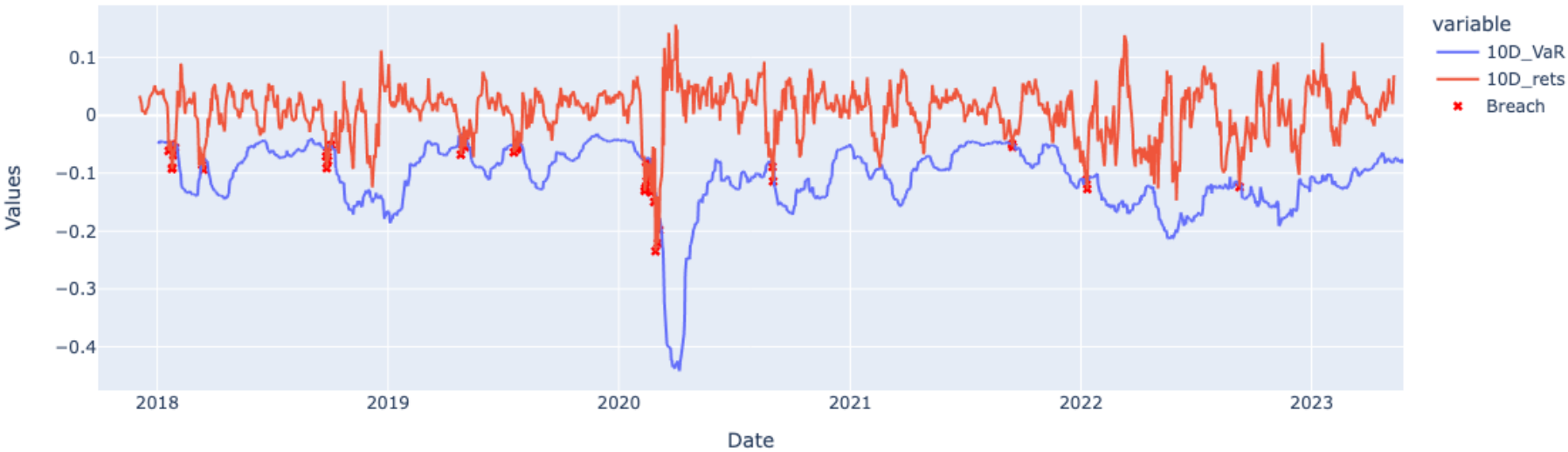
sum(consecutive_counts_list)
```

Number of consecutive breaches is **22**

Using plotly express to easily visualized and identify breaches.

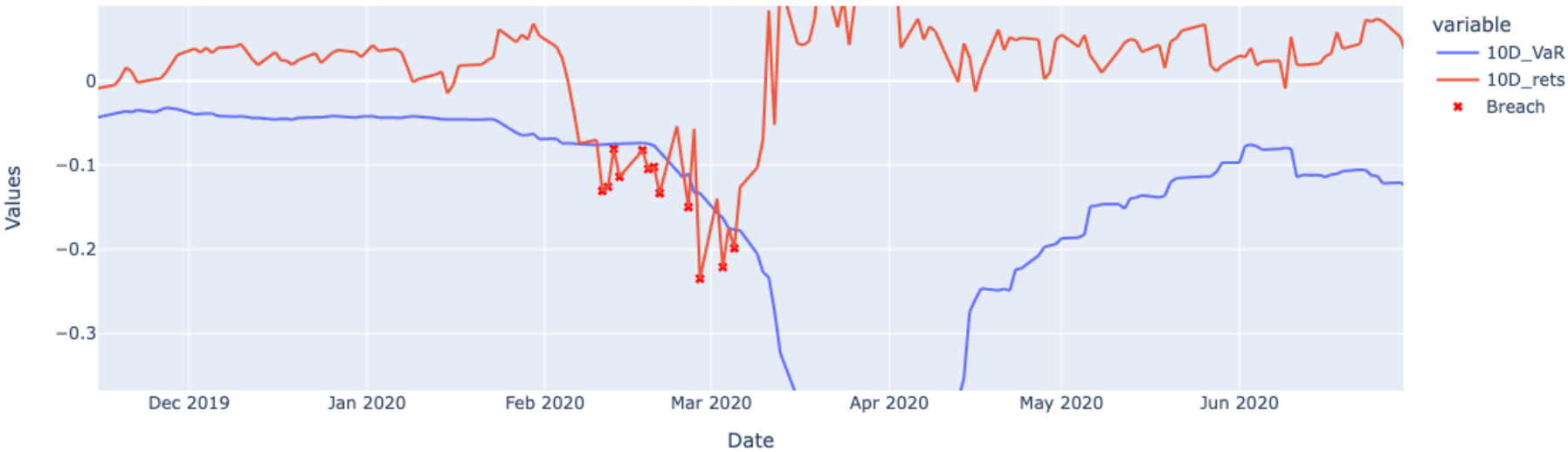

```
fig = px.line(df, x=df.index, y=['10D_VaR', '10D_rets'], title='NASDAQ100 VaR Back Testing of 99%/10day')
fig.add_scatter(x=df[df['breach'] == 1].index, y=df[df['breach'] == 1]['10D_rets'], mode='markers',
marker=dict(color='red', symbol='x'), name='Breach')
fig.update_layout(xaxis_title='Date', yaxis_title='Values')
fig.show()
```

NASDAQ100 VaR Back Testing of 99%/10day



Take a closer look at the Covid period which is from Jan-2020 to March-2020

NASDAQ100 VaR Back Testing of 99%/10day



During this period when there is a lot of uncertainty in the market due to the unprecedented epidemic. During Feb-2020, there is a first series of breaches that indicate the increasing in volatility. This is the market reaction to the global spreading of the virus. Following month of Mar-2020, there is a sharp decline of asset returns, probably this is the effect of massive selling of assets. Our 10D VaR model predict the high risk period in Apr-2020.

Part 5

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

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

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

Hint: computation of EWMA σ_{t+1}^2 is not sufficient, proceed to compute $VaR_{10D,t}$ 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.

First we need to create a new data frame **df2** and implement new EWMA σ_{t+1}^2 variance to compute the new $VaR_{10D,t}$ value:

```
# Compute returns and squared return
df2 = pd.read_csv("../exam/nasdaq100.csv", parse_dates=["Date"], index_col="Date")
df2['rets'] = np.log(df2['Closing Price']).diff()
df2['sq_rets'] = df2['rets'] ** 2
```

For the given λ value, we can compute the value for EWMA σ_{t+1}^2 . For the 2nd day, we simply choose the mean of squared returns μ^2 as our estimation for the variance σ^2 . Now we compute the 3rd day variance using our equation

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

PYTHON

```
# Given lambda
lambda_given = 0.72

# Second day estimation
second_day_variance = np.mean(df2['sq_rets'])

# Create a list of variance estimation
var_est_list = [np.nan, second_day_variance]

# Loop through the data starting from the second index and compute the estimation variance then append the
result to the list
for i in range(1, len(df2)-1):
    result = lambda_given * var_est_list[-1] + (1 - lambda_given) * df2.loc[df2.index[i], 'sq_rets']
    var_est_list.append(result)

# Create a new column in the DataFrame to store variance estimation values
df2['vol_est'] = np.sqrt(var_est_list)

# Create a new column for 10D Returns and 10D vol estimation
df2['VaR_est_10D'] = factor * df2['vol_est'] * np.sqrt(10)
df2['rets_10D'] = np.log(df2['Closing Price'].shift(-(11)) / df2['Closing Price'].shift(-1))

# Identify breaches
df2['breach'] = (df2['rets_10D'] < df2['VaR_est_10D']).astype(int)
```

The new data frame contains calculation for $VaR_{10D,t}$ and forward 10 days return.

Sample **df2**:

Date	sq_rets	vol_est	VaR_est_10D	rets_10D	breach
2017-12-01	NaN	NaN	NaN	0.039071	0
2017-12-04	1.385723e-04	0.016413	-0.120742	0.033828	0
2017-12-05	5.063770e-08	0.015256	-0.112234	0.028114	0
2017-12-06	1.979978e-05	0.012946	-0.095238	0.024461	0
2017-12-07	1.357609e-05	0.011235	-0.082647	0.018830	0
2017-12-08	1.997114e-05	0.009730	-0.071580	0.006123	0
2017-12-11	5.996283e-05	0.008588	-0.063180	0.008035	0
2017-12-12	2.569126e-06	0.008360	-0.061503	0.007284	0
2017-12-13	2.974935e-06	0.007145	-0.052559	0.001018	0
2017-12-14	5.544447e-07	0.006131	-0.045100	0.006938	0

Now we can count breaches

PYTHON

```
mask = df2['breach'] == 1
pct_var_breaches = df2[mask]['breach'].count() / len(df2)
print('No. of Breaches', df2[mask]['breach'].count())
print('Pecentage VaR Breaches', pct_var_breaches)
```

PYTHON

```
No. of Breaches 54
Pecentage VaR Breaches 0.0391304347826087
```

From this EWMA estimate model, we found 54 breaches, more than 41 which is what we found from previous model.

We now can count consecutive breaches using the same algorithm that I have developed:


```
# Convert the breach column to a list for easier manipulation
breach_list = df2['breach'].tolist()

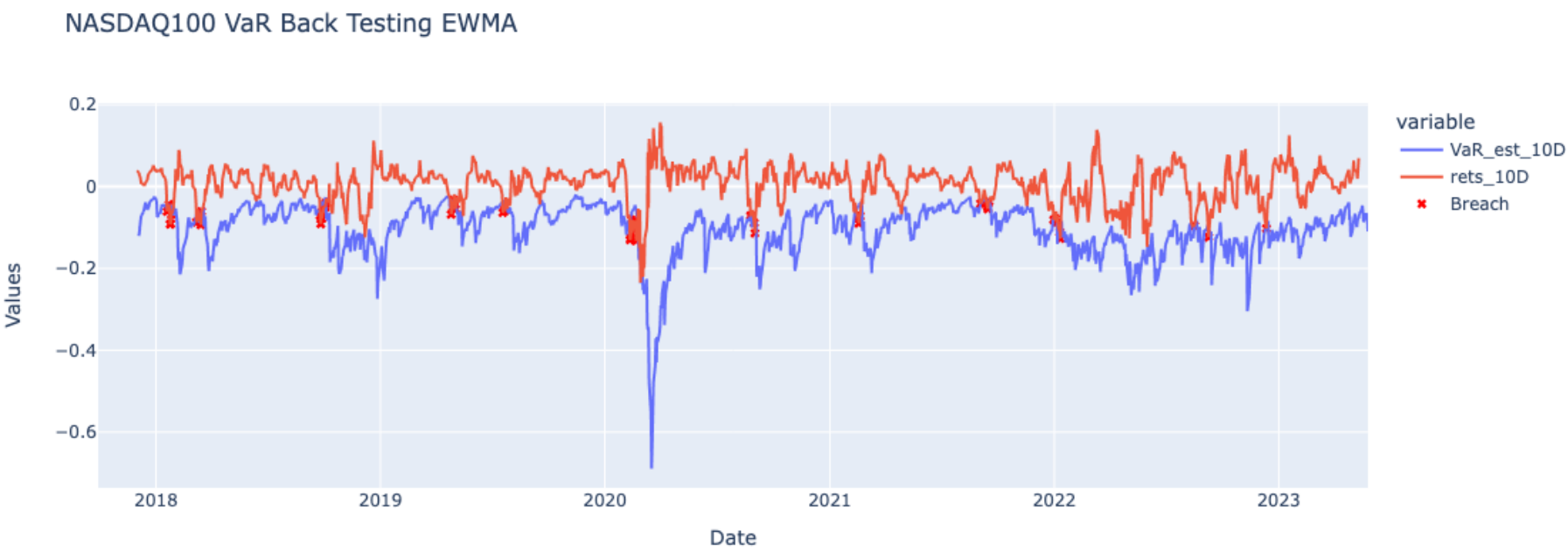
# Initialize variables
consecutive_counts_list = []

# Iterate through the list of breach values
for i in range(len(breach_list) - 1):
    if breach_list[i] == 1 and breach_list[i + 1] == 1:
        consecutive_counts_list.append(1)
    else:
        consecutive_counts_list.append(0)

sum (consecutive_counts_list)
```

And we found that the number of consecutive breaches is **32**

This is the plot for the new σ estimation using EWMA method. The breaches can easily spot as  in the plot

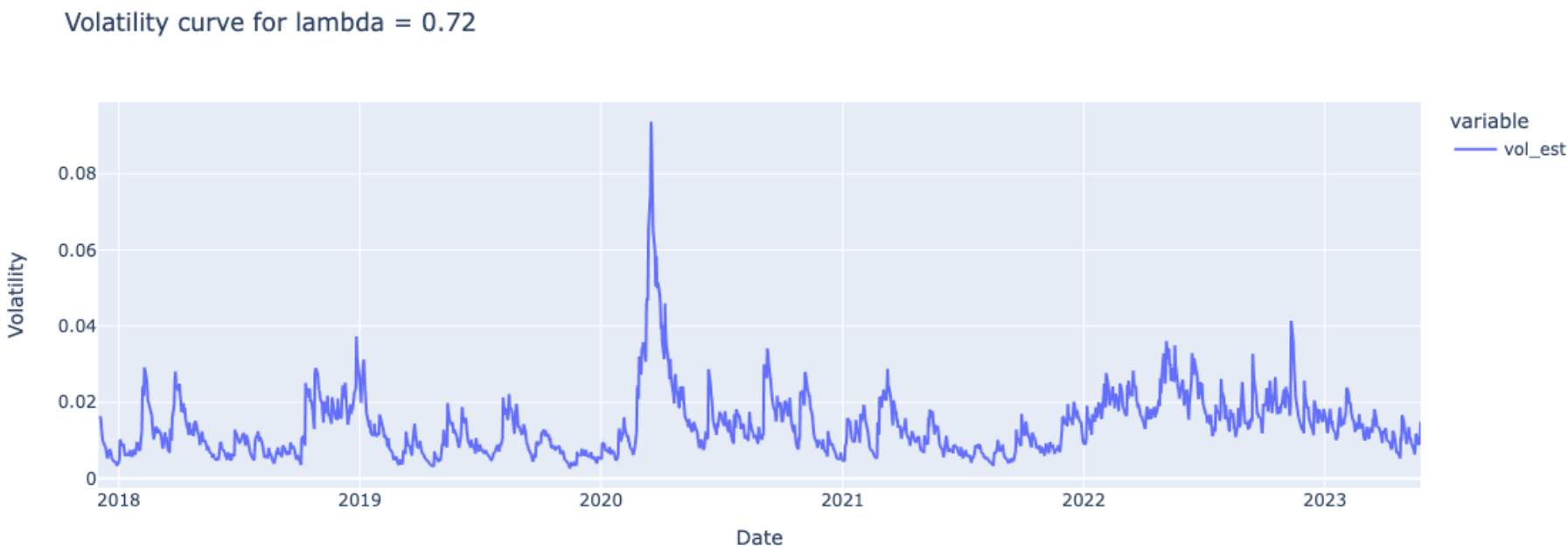


Let's look again at the new function to estimate variance

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

As we increase λ , the term of the previous day variance is increasing, and that makes the prediction for variance tomorrow depends on the variance today. Therefore, it smoothes out the VaR estimate plot. We can visualize this smoothness using the data from our volatility estimation.

For $\lambda = 0.72$



For $\lambda = 0.9$

Volatility curve for lambda = 0.90

