# 5 - Value at Risk (VaR)

Espen Sirnes

2024-10-30

## Table of contents



# 1 VaR

Value at Risk, commonly known as VaR, is a foundational concept in risk management, widely used by financial institutions to quantify and control risk exposure. Essentially, VaR answers the question, *"What is the maximum potential loss over a given time period within a specified confidence interval?"* The confidence interval here represents the range within which we expect a certain percentage of returns to fall. For instance, with a 95% confidence interval, we anticipate that only 5% of cases will breach this lower threshold. The closer the model's expected breaches are to the actual empirical frequency, the better the model's accuracy.

VaR provides a standardized way to assess risk, helping firms to understand

potential losses and allocate capital effectively. Regulators also rely heavily on VaR; in fact, this method is embedded directly in EU financial regulations, requiring banks to hold capital reserves proportional to their risk exposure. By setting a standardized measure for risk, VaR supports financial stability and resilience, making it invaluable for both risk managers and regulators.

In this lecture, we'll see that VaR is not a single, fixed method. Its accuracy and effectiveness depend on one crucial element: how volatility is predicted. Different approaches to forecasting volatility result in varying VaR outcomes, each with its strengths and limitations. By exploring these nuances, we'll gain a deeper understanding of both the power and the limitations of VaR as a risk management tool.

We start by loading the data from lecture 3:

```
import pandas as pd
df = pd.read_pickle('output/X.df')
df
```

| Symbol<br>Date | EQNR | NHY | TEL | YAR | Optimal |
|---|---|---|---|---|---|
| 2016-01-10 | -0.118288 | -0.137636 | -0.008125 | -0.058065 | -0.135241 |
| 2016-01-17 | -0.060966 | -0.054818 | -0.085838 | -0.047905 | -0.058974 |
| 2016-01-24 | 0.060966 | 0.023505 | 0.049143 | 0.001741 | 0.058405 |
| 2016-01-31 | 0.074498 | 0.024710 | -0.007077 | -0.053584 | 0.079774 |
| 2016-02-07 | 0.027490 | 0.065780 | -0.029552 | 0.024170 | 0.040226 |
| ... | ... | ... | ... | ... | ... |
| 2024-03-10 | 0.026170 | 0.016989 | -0.026157 | -0.040846 | 0.033350 |
| 2024-03-17 | 0.023882 | 0.038669 | 0.019681 | 0.016898 | 0.028047 |
| 2024-03-24 | 0.032784 | 0.048790 | 0.022694 | 0.026865 | 0.037538 |
| 2024-03-31 | -0.003143 | -0.023257 | 0.026117 | -0.004525 | -0.009523 |
| 2024-04-07 | 0.046639 | 0.126744 | 0.025854 | 0.044496 | 0.066078 |

# 2 Backtesting VaR

To understand the effectiveness of Value at Risk (VaR), we can test its accuracy by evaluating actual financial data and examining how well different VaR models predict risk.

Specifically, VaR is expected to forecast a certain number of "breaches"—cases

where the return falls below the calculated lower bound—relative to the total sample of returns. The closer the model's predictions align with the actual frequency of breaches, the more reliable the model is.

To carry out this backtest, we'll define a function to calculate VaR for historical returns from a financial time series. The function takes the following inputs: - A general function, `f(x, sigmalist)`, which calculates the 95% and 99% VaR and returns an updated volatility (if applicable). This function uses historical data `x` and past volatility `sigmalist`. The specific function provided will determine the method used for VaR calculation. - A `pandas` DataFrame `df` containing historical data. - The name of the data series in `df`. - An estimation window, specifying the number of periods used to estimate the current VaR.

The function will return: - The 95% and 99% VaR for each date, - The estimated volatility (if applicable), - The corresponding dates, - The associated returns.

Below is the function for generating the backtest:

```
import numpy as np
def generate_backtest(f, df, name, estimation_win_size):
    # Initialize lists to store calculated values
    datelist = []
    sigmalist = []
    d95list = []
    d99list = []
    ret = []

    # Iterate over returns to calculate and store VaR and volatility e
    for t in range(estimation_win_size, len(df)):

        # Record date and current return
        datelist.append(df.index[t].date())
        ret.append(df[name].iloc[t])

        # Extract data from the estimation window (t-estimation_win_si
        x = df[name].iloc[t-estimation_win_size:t-1]

        # Apply the provided VaR estimation function using the histori
        d95, d99, sigma = f(x, sigmalist)

        # Append the estimates to their respective lists
```

```
        sigmalist.append(sigma)
        d95list.append(d95)
        d99list.append(d99)

    # Return the results as numpy arrays for ease of analysis
    return (np.array(d95list),
            np.array(d99list),
            np.array(sigmalist),
            np.array(datelist),
            np.array(ret))
```

This setup allows for a flexible approach to testing different VaR models by simply passing different VaR estimation functions into `generate_backtest`. The returned data can then be used to evaluate the model's effectiveness by comparing predicted breaches against actual outcomes.

Here's a refined version of the evaluation function and its description, with improved clarity and formatting:

---

# 3 Evaluation Function

After generating the VaR time series, we need an `evaluate` function to assess the model's effectiveness. This function will take the following arguments: - A plot object `plt` to visualize the performance, - The estimated 95% and 99% VaR series, - The returns, - The dates, - A heading for the plot.

The function will produce an informative graph with bars wher there are breaches, and display the model's performance by calculating the percentage of breaches. Here's the function:

```
import matplotlib.pyplot as plt
def evaluate(plt, d95, d99, ret, dates, heading):
    # Clear the plot area to avoid overlapping plots
    plt.cla()

    # Plot the 95% VaR, 99% VaR, and actual returns
    plt.plot(dates, d95, label='95% Confidence Level')
    plt.plot(dates, d99, label='99% Confidence Level')
```

```
    plt.plot(dates, ret, label='Actual Return')

    # Highlight instances where returns breach the 95% VaR
    maxret = max(ret)
    breaches_95 = [maxret if d > r else 0 for d, r in zip(d95, ret)]
    plt.bar(dates, breaches_95, color='gray', alpha=0.5, width=0.5, la

    # Set labels and title
    plt.ylabel('VaR')
    plt.xlabel('Date')
    plt.title(heading)
    plt.xticks(rotation=90)
    plt.legend(loc="lower right")
    plt.subplots_adjust(bottom=0.15)
    plt.show()

    # Calculate and print the breach percentage for each confidence le
    backtest_results = [np.round(sum(d > ret) / len(ret) * 100, 1) for

    for i, level in enumerate([95, 99]):
        breaches = sum([d95, d99][i] > ret)
        print(f"{heading} with {level}% confidence interval:\n"
              f"Breaches: {breaches}\n"
              f"Backtesting (Realized VaR - % breaches): {backtest_res
```

This function provides a visual and numerical evaluation of VaR accuracy. The plot shows the 95% and 99% VaR levels alongside the actual returns, with breaches of the 95% VaR highlighted. The function then calculates and prints the percentage of breaches, allowing us to assess how well the model aligns with expected outcomes.

Here's an improved version with clearer explanations and consistent formatting:

---

# 4 Methods for Calculating VaR

Calculating Value at Risk (VaR) fundamentally involves predicting volatility. A model that can accurately predict volatility will yield a more reliable VaR estimate. Here, we'll explore three different methods for calculating VaR: using the normal

distribution, empirical frequencies, and past volatility.

## 4.1 The Normal Distribution Model

The most straightforward approach is to assume that returns follow a normal distribution. We can estimate the standard deviation (volatility) of a sample `x` and apply the normal distribution to compute VaR. This is implemented as follows:

```
PVALS = [0.05, 0.01]  # Confidence intervals (95% and 99%)
from scipy.stats import norm

def normal_est(x, sigmalist):
    z = norm.ppf(PVALS)  # Z-scores for the specified confidence level
    sigma = np.std(x, ddof=1)  # Sample standard deviation
    return z[0] * sigma, z[1] * sigma, sigma
```

The global variable `PVALS` sets the confidence levels, making it easy to adjust later, though 95% and 99% are almost always used in practice.

## 4.2 Empirical Frequencies

An alternative method is to rely on empirical data, using historical quantiles to set the VaR thresholds. This method establishes limits that would have perfectly predicted historical data breaches at the 95% and 99% levels:

```
def historical_est(x, sigmalist):
    q95 = abs(np.quantile(x, PVALS[0]))  # 95th percentile of historic
    q99 = abs(np.quantile(x, PVALS[1]))  # 99th percentile of historic
    return -q95, -q99, None  # VaR values are negative to indicate pot
```

This approach is straightforward and does not assume any specific distribution of returns.

## 4.3 Past Volatility

A third method involves modeling volatility based on its past behavior. Using only the volatility from the previous period would imply constant volatility, so we introduce an innovation term—specifically, the squared error from the previous period. This error is added to the previous period's variance and then square-rooted to estimate current volatility. If no prior volatility exists (e.g., in the first period), historical volatility is used.

After estimating the volatility, we can use the normal distribution to calculate VaR for each confidence interval. This type of volatility-based VaR estimation is required for financial firms under EU regulations.

```
def last_volat(x, sigmalist):
    x = np.array(x)
    z = norm.ppf(PVALS)
    if not sigmalist:  # If sigmalist is empty, use initial standard d
        sigma = np.std(x, ddof=1)
    else:  # Update sigma based on past volatility and recent error
        sigma = (0.1 * (x[0] - np.mean(x))**2 + 0.9 * sigmalist[-1]**2
    return z[0] * sigma, z[1] * sigma, sigma
```

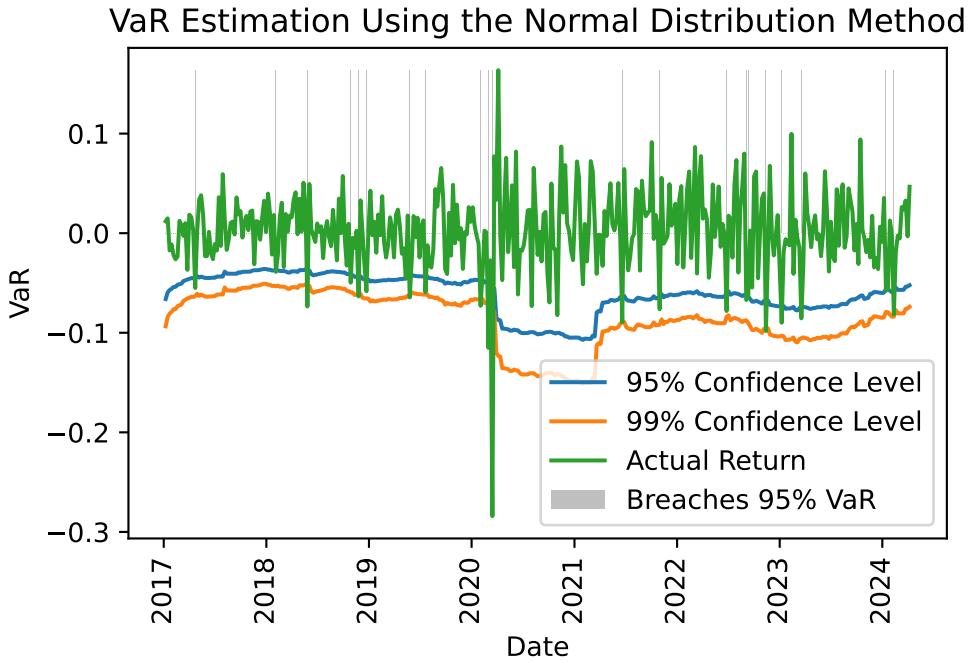
These methods offer varied approaches to calculating VaR, each with specific strengths: the normal model assumes a normal distribution, empirical quantiles base predictions on historical breaches, and the volatility-based method adjusts dynamically, often aligning with regulatory requirements.

# 5 Evaluation

We can now easily evaluate the different methods. First the normal distribution:

```
NAME = 'EQNR'
ESTIMATION_WINSIZE = 52

(normal95, normal99,
 sigma, dates, ret )= generate_backtest(normal_est,
        df, NAME, ESTIMATION_WINSIZE)
evaluate(plt, normal95, normal99, ret, dates,
         'VaR Estimation Using the Normal Distribution Method')
```

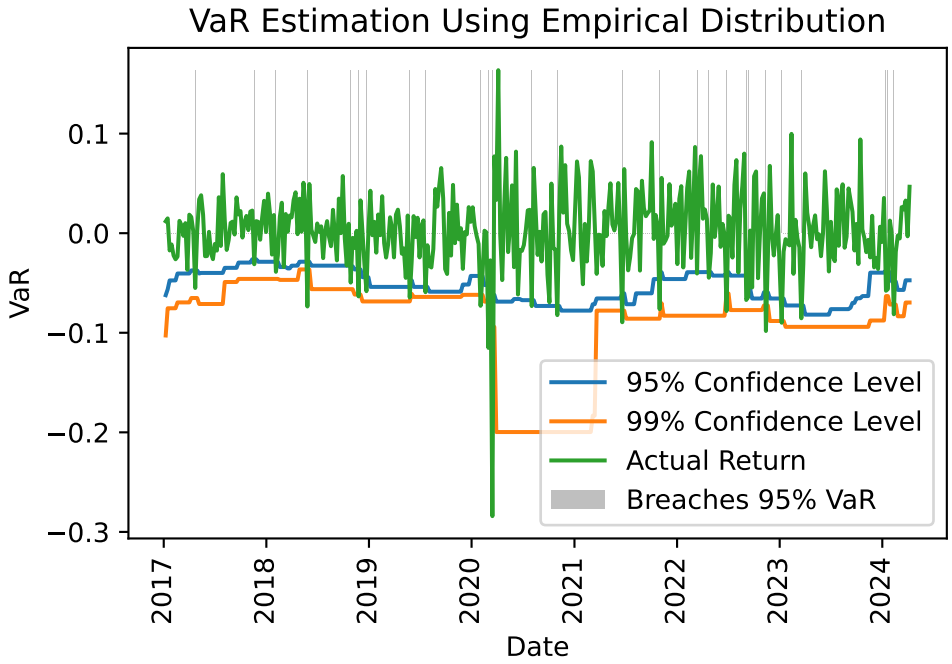



```
VaR Estimation Using the Normal Distribution Method with 95% confidenc
Breaches: 21
Backtesting (Realized VaR - % breaches): 5.5%

VaR Estimation Using the Normal Distribution Method with 99% confidenc
Breaches: 8
Backtesting (Realized VaR - % breaches): 2.1%
```

Then the historcal:

```
(hist95, hist99,
 sigma, dates, ret )= generate_backtest(historical_est,
        df, NAME, ESTIMATION_WINSIZE)
evaluate(plt, hist95, hist99, ret, dates,
         'VaR Estimation Using Empirical Distribution')
```

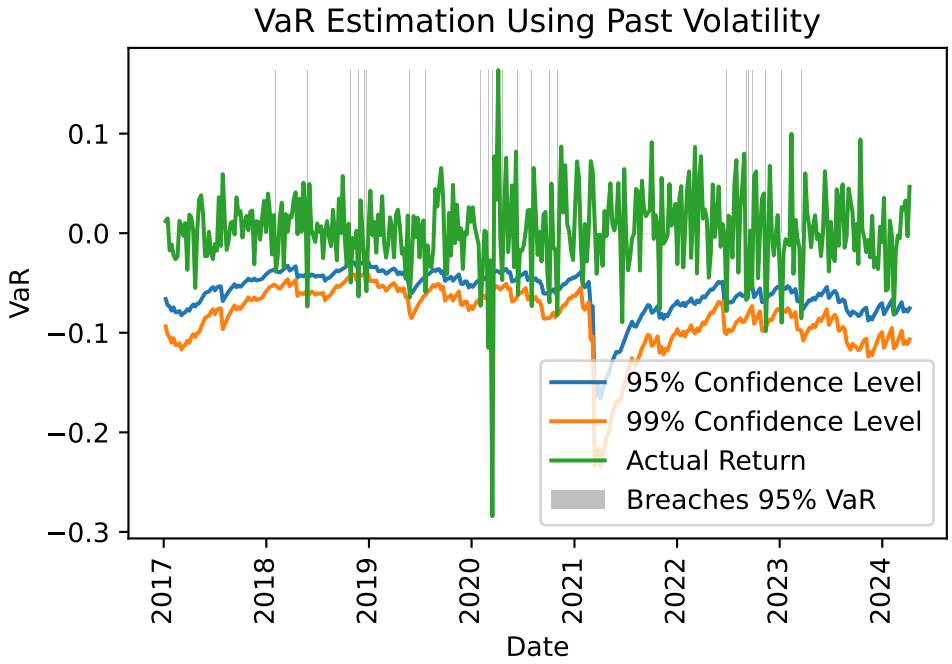



```
VaR Estimation Using Empirical Distribution with 95% confidence interv
Breaches: 27
Backtesting (Realized VaR - % breaches): 7.1%

VaR Estimation Using Empirical Distribution with 99% confidence interv
Breaches: 10
Backtesting (Realized VaR - % breaches): 2.6%
```

And at last, using the previous volatilty

```
(last95, last99,
 sigma, dates, ret )= generate_backtest(last_volat,
        df, NAME, ESTIMATION_WINSIZE)
evaluate(plt, last95, last99, ret, dates,
         'VaR Estimation Using Past Volatility')
```




```
VaR Estimation Using Past Volatility with 95% confidence interval:
Breaches: 23
Backtesting (Realized VaR - % breaches): 6.1%

VaR Estimation Using Past Volatility with 99% confidence interval:
Breaches: 11
Backtesting (Realized VaR - % breaches): 2.9%
```