CQF Exam One Portfolio and Risk Techniques

January 2022 Cohort

William Harris

March 21, 2022

Problem 1

(24 marks) An investment universe of the following risky assets with a dependence structure (correlation) is given. Use the ready appropriate formulae from Portfolio Optimisation Lecture.

Asset	μ	σ	ω
A	0.02	0.05	ω_1
B	0.07	0.12	ω_2
C	0.15	0.17	ω_3
D	0.20	0.25	ω_4

$$R = \begin{pmatrix} 1 & 0.3 & 0.3 & 0.3 \\ 0.3 & 1 & 0.6 & 0.6 \\ 0.3 & 0.6 & 1 & 0.6 \\ 0.3 & 0.6 & 0.6 & 1 \end{pmatrix}$$

Consider minimum variance portfolio with a target return m.

$$\arg\min_x \frac{1}{2}\omega' \Sigma \omega \tag{1}$$

s.t

$$\omega' 1 = 1 \tag{2}$$

$$\mu_\Pi = \omega' \mu = m \tag{3}$$

A) Formulate the Lagrangian function and give its partial derivatives. No further derivation required.

B) Compute the allocations ω^* and portfolio risk $\sigma_\Pi = \sqrt{\omega' \Sigma \omega}$ for $m = 4.5\%$. Now, stress the correlation matrix: multiply all correlations by x1.25 and x1.5, and compute the respective optimal allocations and portfolio risk (the same $m = 4.5\%$).

C) Inverse optimisation: generate $> 2{,}000$ random allocations sets ω' - these will not be optimal allocations. Plot the cloud of points of μ_Π vertically on σ_Π horizontally. Before computing μ_Π,σ_Π you can standardise to satisfy $\omega' 1 = 1$.

Solution.

A) Lagrangian Function and partial derivatives.

$$L(w, \lambda, \gamma) = \frac{1}{2}\omega' \Sigma \omega + \lambda(m - \omega' \mu) + \gamma(1 - \omega' 1) \tag{4}$$

$$\frac{\partial L}{\partial \omega} = \Sigma \omega - \lambda \mu - \gamma 1 \tag{5}$$

$$\frac{\partial L}{\partial \lambda} = m - \omega' \mu \tag{6}$$

$$\frac{\partial L}{\partial \gamma} = 1 - \omega' 1 \tag{7}$$

$$\frac{\partial^2 L}{\partial \omega^2} = \Sigma \tag{8}$$

B) Solving the Lagrangian Function 4 produces the optimal weight vector equation 9.

$$\omega^* = \Sigma^{-1}(\lambda\mu + \gamma 1) \tag{9}$$

with

$$\lambda = \frac{Am - B}{AC - B^2} \tag{10}$$

$$\gamma = \frac{C - Bm}{AC - B^2} \tag{11}$$

and

$$A = 1'\Sigma^{-1}1 \tag{12}$$

$$B = \mu'\Sigma 1 \tag{13}$$

$$C = \mu'\Sigma\mu \tag{14}$$

This solution was implemented as Python code in a Jupyter Notebook. The computation leverages the Python package numpy to perform the vector arithmetic.

```python
import pandas as pd
import numpy as np

def compute_portfolio_weights_and_risk(mu, sigma, R, m):

    S = np.diag(sigma.reshape(4,))
    Sigma = np.linalg.multi_dot([S, R, S])

    ones = np.array([1,1,1,1]).reshape(1,4)
    A = np.linalg.multi_dot([ones, np.linalg.inv(Sigma), ones.T])
    B = np.linalg.multi_dot([mu.T, np.linalg.inv(Sigma), ones.T])
    C = np.linalg.multi_dot([mu.T, np.linalg.inv(Sigma), mu])

    lambd = ((A * m - B) / (A*C - B**2))
    gamma = ((C - B * m) / (A*C - B**2))

    omega_star = np.dot(np.linalg.inv(Sigma), (lambd * mu + gamma * ones.T))
    sigma_Pi = np.sqrt(np.linalg.multi_dot([omega_star.T, Sigma, omega_star]))[0][0]

    weights = {f"W_{i+1}": round(w* 100, 4) for i, w in enumerate(omega_star.reshape(4,))}
    print(f'Portfolio Weights {weights}')
    print(f'Portfolio Risk {sigma_Pi*100:.4}%')

    return omega_star, sigma_Pi
```

The input variables were defined and run through this Python function.

```python
mu = np.array([0.02, 0.07, 0.15, 0.2]).reshape(4,1)
sigma = np.array([0.05, 0.12, 0.17, 0.25]).reshape(4,1)
R = np.array([1, 0.3, 0.3, 0.3,
              0.3, 1, 0.6, 0.6,
              0.3, 0.6, 1, 0.6,
              0.3, 0.6, 0.6, 1]).reshape(4,4)
omega_star, sigma_pi = compute_portfolio_weights_and_risk(mu, sigma, R, m=0.045)
```

Which produced the following portfolio weights and portfolio risk.

$$\omega^* = \begin{pmatrix} 78.5111\% \\ 5.3864\% \\ 13.3555\% \\ 2.747\% \end{pmatrix} \quad \sigma_\Pi = 5.84\%$$

The correlations were then multiplied by x1.25 into the variable *R_x1_25* and run.

```python
R_x1_25 = np.array([1, 0.375, 0.375, 0.375,
                    0.375, 1, 0.75, 0.75,
                    0.375, 0.75, 1, 0.75,
                    0.375, 0.75, 0.75, 1]).reshape(4,4)
omega_star, sigma_pi = compute_portfolio_weights_and_risk(mu, sigma, R_x1_25, m=0.045)
```

$$\omega^* = \begin{pmatrix} 81.8189\% \\ -0.9403\% \\ 17.8966\% \\ 1.2248\% \end{pmatrix} \quad \sigma_\Pi = 6.071\%$$

And again, with the correlations multiplied by 1.5x.

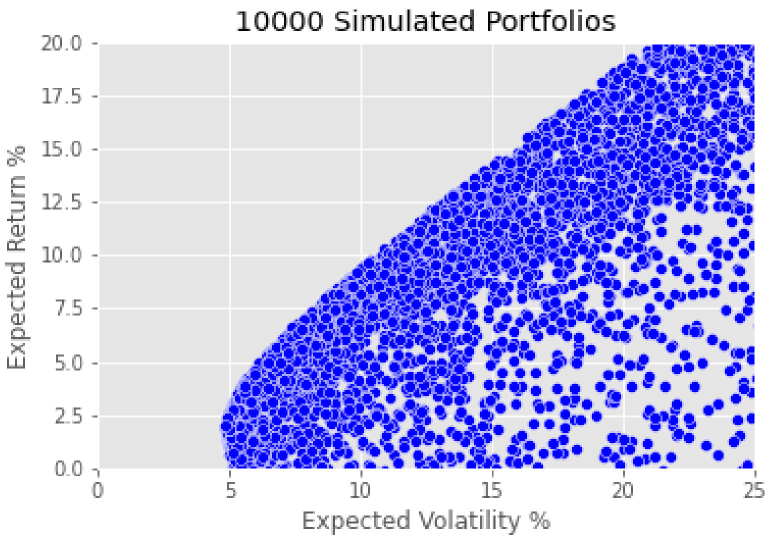
```python
R_x1_5 = np.array([1, 0.45, 0.45, 0.45,
                   0.45, 1, 0.9, 0.9,
                   0.45, 0.9, 1, 0.9,
                   0.45, 0.9, 0.9, 1]).reshape(4,4)
omega_star, sigma_pi = compute_portfolio_weights_and_risk(mu, sigma, R_x1_5, m=0.045)
```

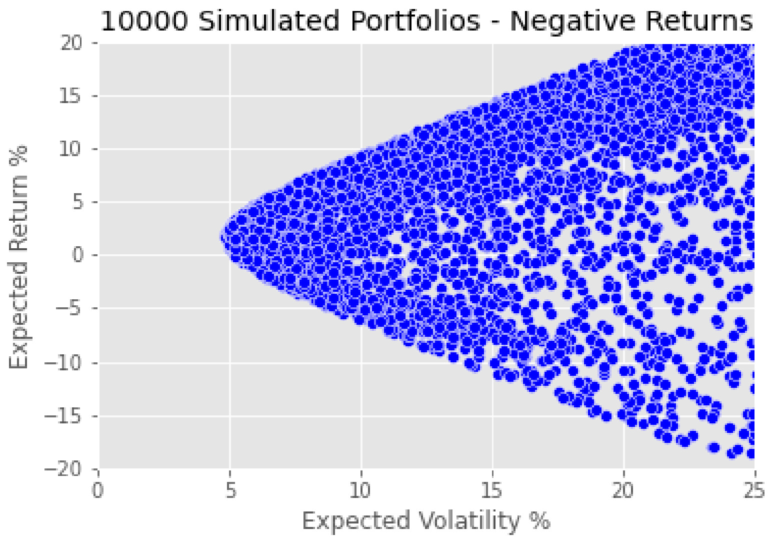
$$\omega^* = \begin{pmatrix} 87.6176\% \\ -14.613\% \\ 32.5701\% \\ -5.5748\% \end{pmatrix} \quad \sigma_\Pi = 6.109\%$$

C) The following Python code was used to generate random allocations of ω' and thus the portfolios expected return and expected volatility.

```python
import pandas as pd

def get_weights():
    a = np.round(np.random.normal(), 4)
    b = np.round(np.random.normal() * (1-a), 4)
    c = np.round(np.random.normal() * (1-a-b), 4)
    d = 1.0-a-b-c
    weights = np.array([a, b, c, d]).reshape(4,1)
    assert abs(weights.sum() - 1.0) <= 0.001, weights
    return weights

def portfolio_simulation(mu, sigma, R, num_ports):
    S = np.diag(sigma.reshape(4,))
    Sigma = np.linalg.multi_dot([S, R, S])

    returns = []
    vols = []
    w = []
    for _ in range(num_ports):
        weights = get_weights() #np.random.dirichlet((1, 1, 1, 1),size=1).T
        returns.append(np.dot(weights.T, mu)[0][0])
        vols.append(np.sqrt(np.linalg.multi_dot([weights.T, Sigma, weights]))[0][0])
        w.append(weights)

    df = 100*pd.DataFrame({
        'returns': returns,
        'vols': vols,
        'weights': w
        })

    return df
```

The variables *mu, sigma* and *R* defined in part B were used as inputs. 10,000 portfolios were randomly generated which was definied in the variable $num\_ports = 10000$. The portfolio weights are standardised in the *get_weights* function and validated with an accuracy within 0.001. Originally, the numpy function $np.random.dirichlet((1, 1, 1, 1), size = 1).T$ was used to simulate portfolio weights, however this was later refactored to *get_weights* as it did not generate negative position weights.


10000 Simulated Portfolios
Expected Return %
Expected Volatility %
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
5
10
15
20
25


10000 Simulated Portfolios - Negative Returns
Expected Return %
Expected Volatility %
20
15
10
5
0
−5
−10
−15
−20
0
5
10
15
20
25

Problem 2

(20 marks) Continue with the data from Question 1 and consider a tangency portfolio.

A) Formulate the Lagrangian function and give its partial derivatives only.

B) For the range of tangency portfolios given by $r_f = 50bps, 100bps, 150bps, 175bps$ optional compute allocations (ready formula) and σ_Π. Present results in a table.

C) Plot the true Efficient Frontier in the presence of risk-free earning asset for $r_f = 100bps, 175bps$ and specifically identify its shape.

Solution.
A) The following optimisation problem can be used to obtain the Lagrangian function for a tangency portfolio.

$$\arg\min_x \frac{1}{2}\omega' \Sigma \omega \tag{15}$$

s.t

$$r + \omega'(\mu - r1) = m \tag{16}$$

Thus the Lagrangian function and its partial derivatives are:

$$L(w, \lambda) = \frac{1}{2}\omega' \Sigma \omega + \lambda(m - r - \omega'(\mu - r1)) \tag{17}$$

$$\frac{\partial L}{\partial \omega} = \Sigma \omega - \lambda(\mu - r1) \tag{18}$$

$$\frac{\partial L}{\partial \lambda} = m - r - \omega'(\mu - r1) \tag{19}$$

$$\frac{\partial^2 L}{\partial \omega^2} = \Sigma \tag{20}$$

B) The solution to the Lagrangian Function 17 can be used to calculate the tangency portfolio's optimal weights. This is done by taking into consideration that $\omega' 1 = 1$ because tangency portfolios consist of positions only invested in risky assets.

$$\omega_t = \frac{\Sigma^{-1}(\mu - r1)}{B - Ar} \tag{21}$$

This equation was then implemented as a function in Python Jupyter Notebooks. In addition the function calculates the portfolios expected return and volatility. It was then called four times with the variable r equalling one of the following risk-free rates $r_f = 50bps, 100bps, 150bps, 175bps$.

```python
import pandas as pd
import numpy as n
import scipy.optimize as sco

def compute_tangent_portfolio(mu, sigma, R, r):

    S = np.diag(sigma.reshape(4,))
    Sigma = np.linalg.multi_dot([S, R, S])

    A = np.linalg.multi_dot([np.array([1,1,1,1]),
                            np.linalg.inv(Sigma),
                            np.array([[1],[1],[1],[1]])])
    B = np.linalg.multi_dot([mu.T, np.linalg.inv(Sigma), np.array([[1],[1],[1],[1]])])
    C = np.linalg.multi_dot([mu.T, np.linalg.inv(Sigma), mu])

    omega_t = np.dot(np.linalg.inv(Sigma), (mu-r*np.array([[1],[1],[1],[1]]))) / (B - A*r)
    assert round(omega_t.sum(),2) == 1.0

    sigma_t = np.sqrt(np.linalg.multi_dot([omega_t.T, Sigma, omega_t]))[0][0]
    m_t = np.dot(omega_t.T, mu)[0][0]

    weights = {f"W_{i}": round(w* 100, 4) for i, w in enumerate(omega_t.reshape(4,))}
    print(f'Portfolio Weights {weights}')
    print(f'Portfolio Risk {sigma_t*100}%')
    print(f'Portfolio Returns {m_t*100}%')

    return omega_t, sigma_t, m_t
```

r_f	ω_1	ω_2	ω_3	ω_4	σ_Π
50bps	1.68%	-22.94%	81.4%	39.82%	19.65%
100bps	-74.59%	-51.04%	149.02%	76.63%	35.06%
150bps	-864.49%	-342.26%	848.97%	457.78%	197.35%
175bps	810.35%	275.19%	-635.14%	-350.39%	147.87%

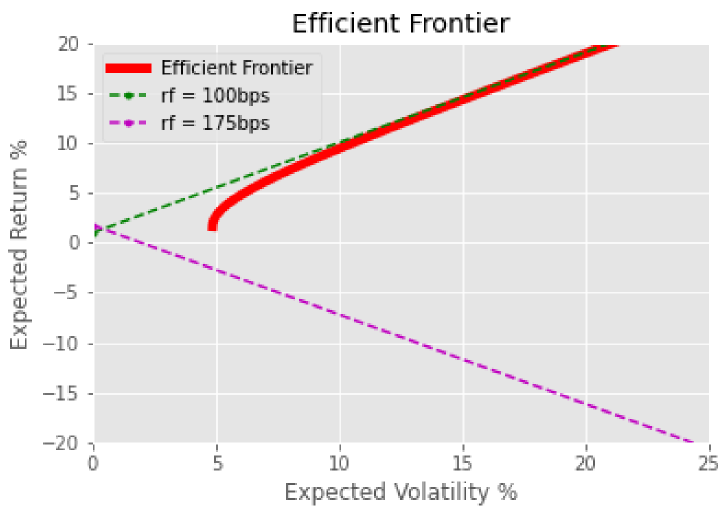
C) The true Efficient Frontier was calculated by fixing the portfolio expected returns at particular values and computing the corresponding optimal portfolio weights. The below code uses the Python package scipy to perform the optimisation problems. Firstly, the minimum variance portfolio was calculated to define the start of the Efficient Frontier. This was done by constructing a optimisation problem, shown below. For the purpose of seeing the entire Efficient Frontier on the plots, the bounds for the minimum and maximum weights were chosen to be -500% to 500%.

```python
initial_weights = np.array([0.25,0.25,0.25,0.25]).reshape(4,1)
bounds = tuple((-5, 5) for _ in range(4))

def portfolio_stats(weights):
    S = np.diag(sigma.reshape(4,))
    Sigma = np.linalg.multi_dot([S, R, S])
    returns = np.dot(weights.T, mu)[0]
    vols = np.sqrt(np.linalg.multi_dot([weights.T, Sigma, weights]))
    return returns, vols

def min_variance(weights):
    return portfolio_stats(weights)[1]**2

constraints=({'type': 'eq', 'fun': lambda x: sum(x) - 1})
result = sco.minimize(min_variance, initial_weights, method='SLSQP',
                      bounds=bounds, constraints=constraints)

min_var_return , min_var_vol = portfolio_stats(result.x)
```

Now, with the expected return for the minimum variance portfolio, a range of portfolio expected returns are created through the numpy function *np.linespace*. For each of these target returns, an optimisation calculation attempts to find the portfolio's optimal weights through minimising the portfolio's expected volatility. While taking into consideration the constraints 1. the portfolio weights must sum to one and 2. the portfolio expected return must equal the target return.

```python
def min_volatility(weights):
    return portfolio_stats(weights)[1]

target_returns = np.linspace(min_var_return, 2.0, 1000)
vols = []

for target_return in target_returns:

    constraints = ({'type': 'eq', 'fun': lambda x: portfolio_stats(x)[0] - target_return},
                   {'type': 'eq', 'fun': lambda x: sum(x) - 1})

    optimsiation = sco.minimize(min_volatility, initial_weights, method='SLSQP',
                                bounds=bounds, constraints=constraints)
    vols.append(optimsiation['fun'])

target_vols = np.array(vols)

efport = pd.DataFrame({
    'target_returns' : 100*target_returns,
    'target_vols': 100*target_vols,
})
```

Lastly, the plot below shows the tangent portfolio's $r_f = 100bps, 175bps$ Capital Market Lines and the risky assets Efficient Frontier.



This graph demonstrates the hyperbolic shape of the risky assets Efficient Frontier. As well as showing the linear behaviour of the Capital Market Lines (CML) for portfolios with risky assets and a risk-free asset. With the introduction of the risk-free asset, portfolio expected returns can be increased while maintaining the same level of expected volatility. It's important to consider that the CML's slope becomes negative when the risk-free rate is higher than the minimum variance portfolio's expected return. These CML situations involve shorting the tangency portfolio and lending/borrowing the risk-free rate, which is inefficient. This produces the cone shape from the CML lines which encases the risk assets Frontier.

Problem 3

(42 marks) As a market risk analyst, each day you calculate VaR from available prior data. Then, you wait ten days to compare your prediction value VaR_{t-10} to the realised return and check if the prediction about the worst loss was breached. You are given a dataset with *Closing Prices.*

A) Implement VaR backtesting by computing $99\%/10day$ Value at Risk using the rolling window of 21 returns to compute σ. (a) Report the percentage of VaR breaches and (b) number of consecutive breaches. (c) Provide a plot which clearly identifies breanches.

$$VaR_{10D,t} = Factor * \sigma_t * \sqrt{10} \tag{22}$$

B) For comparison, implement backtesting using variance forecast equation below (recompute on each day). Rolling window of 21 remains for σ_t^2 (past variance) computation. The equation is known as EWMA model, and you can check how variance forecast is done in the relevant lecture.

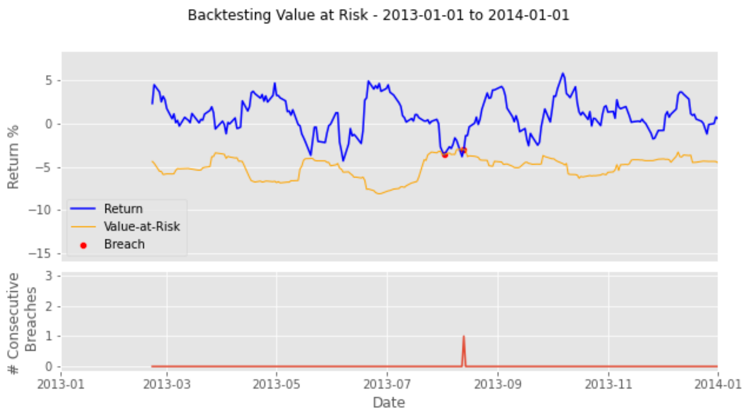
$$\sigma^2_{t_+1|t} = \lambda\sigma^2_{t|t-1} + (1-\lambda)r_t^2 \tag{23}$$

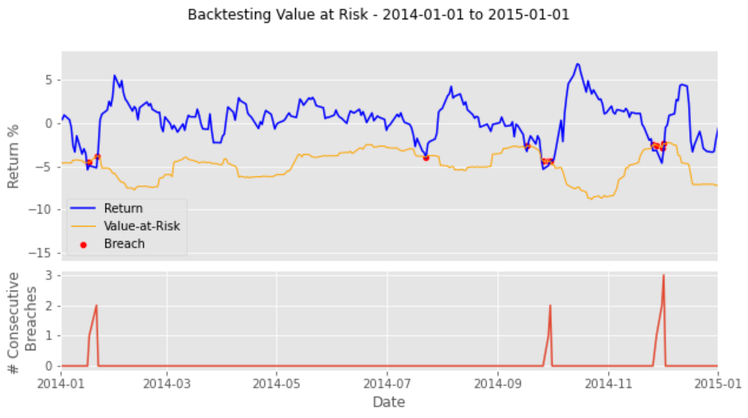
with $\lambda = 0.72$ value set to minimise out of sample forecasting error, and r_t refers to a return. Provide the same deliverables (a), (b) and (c).

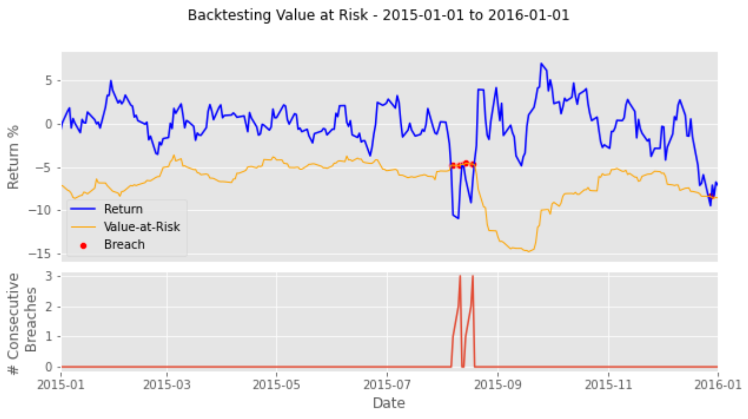
Solution.
A) The VaR backtesting was conducted by using the Python package pandas in Python Jupyter Notebooks. S&P500 log returns were calculated and thus used to produce two columns expressing the rolling 21 day mean and standard deviation. After which, the VaR 99% 10D column was computed following 22 equation and the forward return by taking the following 10 day log return. The backtesting statitics for the number of breaches and consecutive breaches was then calculated by using the pandas mean, groupby, cumsum and cumcount functions. The plot was separated by year into subplots to clearly show the breaches.

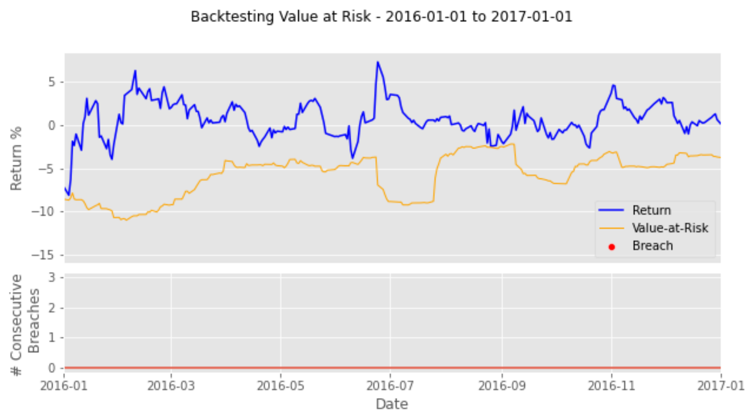
```python
import pandas as pd
import numpy as np
from scipy.stats import norm

def var_df():
    df = pd.read_csv("Data_SP500.csv", index_col=0, parse_dates=True).rename(columns={'SP500': 'close'})
    df['returns'] = np.log(df.close) - np.log(df.close.shift(1))
    df['mean'] = df['returns'].rolling(21).mean()
    df['stdev'] = df['returns'].rolling(21).std()
    df['VaR_99_10D'] = norm.ppf(1-0.99) * df['stdev'] * np.sqrt(10) * 100
    df['Forward_Return'] = (np.log(df.close.shift(-11)) - np.log(df.close.shift(-1))) * 100
    df = df.dropna()
    df['Breaches'] = np.where(df['Forward_Return'] < df['VaR_99_10D'], 1, 0)
    print('Percentage of Breaches: ', df['Breaches'].mean()*100)
    count = df['Breaches'].groupby((df['Breaches'] != df['Breaches'].shift()).cumsum()).cumcount()
    df['Breachers_Conc'] = df['Breaches'] * count
    return df
```

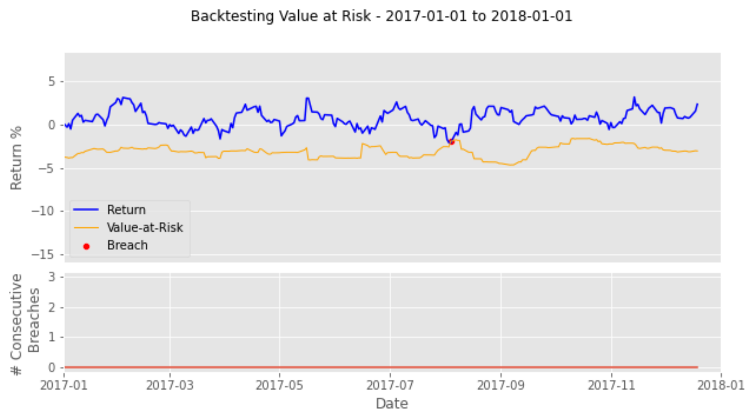
(a) Percentage of VaR breaches: 2.05% (b) Number of consecutive breaches: 3





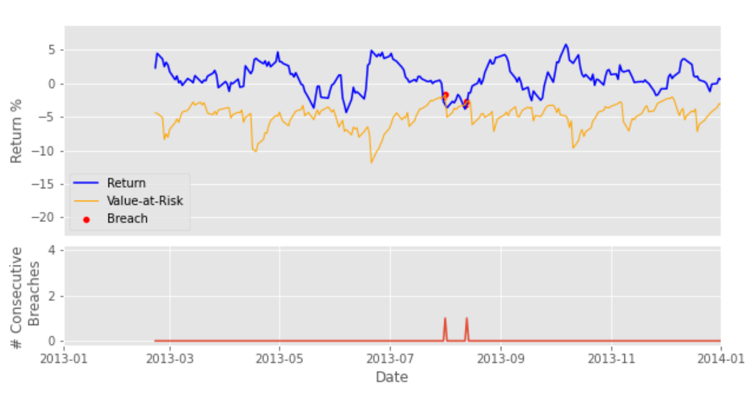


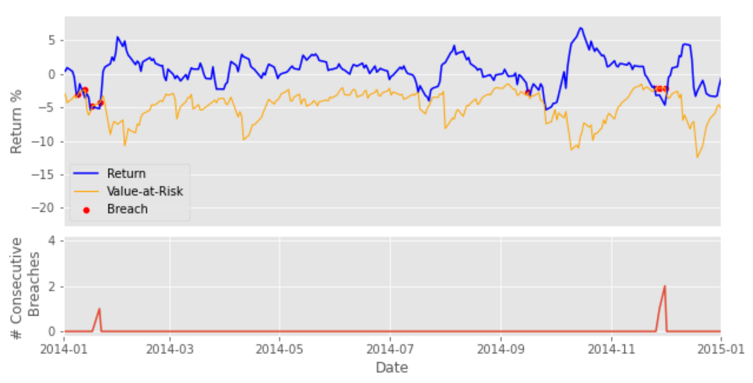


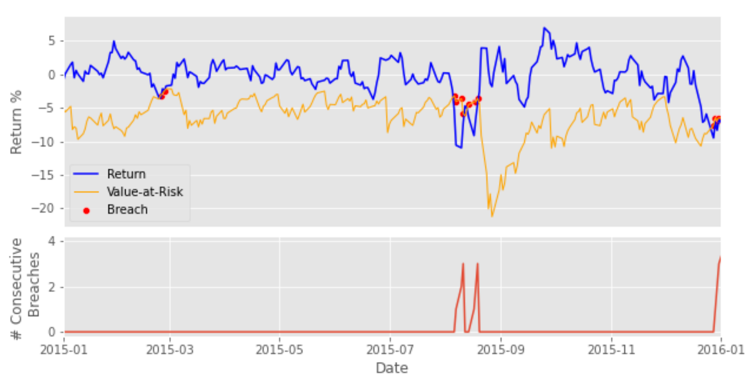


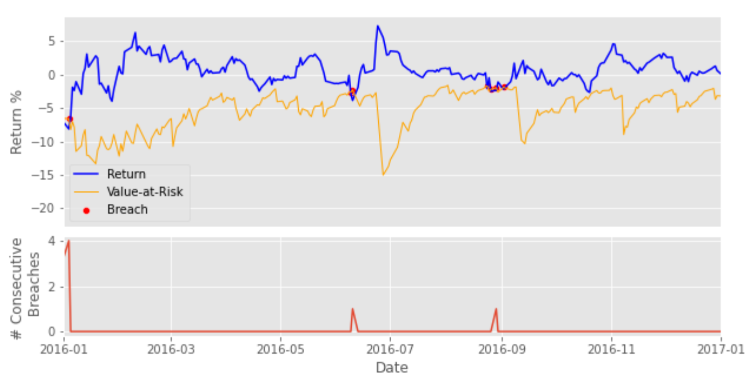
B) Similarly to part A, The S&P log returns were found and used to produce the rolling 21 day mean and standard deviation columns. The returns were then squared and used in a function which implemented the EWMA equation 23 to produce the new estimated standard deviation column. After which, the VaR 99% 10D and future return columns could be calculated along with the backtesting statistics. (a) Percentage of VaR breaches: 2.96% (b) Number of consecutive breaches: 4

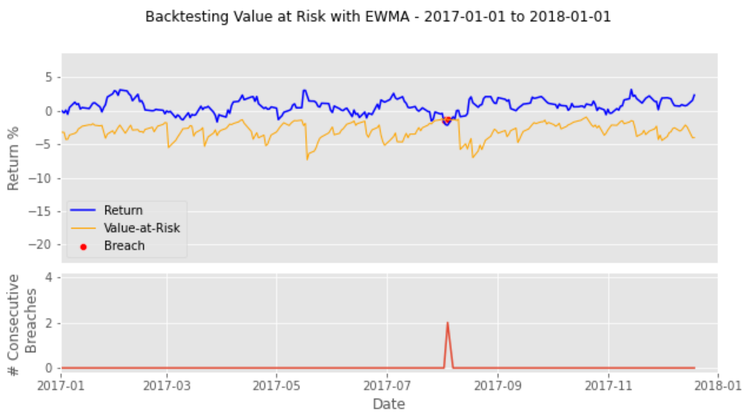
```python
def ewma(start_var: float, sqrd_rets: list, lam: float):

    var = [start_var]
    for i in range(1,len(sqrd_rets)):
        var.append(lam * var[i-1] + (1-lam) * sqrd_rets[i-1])

    return np.array(var)

def var_ewma_df():
    df = pd.read_csv("Data_SP500.csv", index_col=0).rename(columns={'SP500': 'close'})
    df.index = pd.to_datetime(df.index, format='%d/%m/%Y')
    df['returns'] = np.log(df.close) - np.log(df.close.shift(1))
    df['mean'] = df['returns'].rolling(21).mean()
    df['stdev'] = df['returns'].rolling(21).std()
    df = df.dropna()
    df['Squared_returns'] = df['returns']**2
    df['Var_Estimate'] = ewma(df.iloc[0]['stdev']**2, df['Squared_returns'].values, 0.72)
    df['Std_Estimate'] = np.sqrt(df['Var_Estimate'])
    df['VaR_99_10D_EWMA'] = norm.ppf(1-0.99) * df['Std_Estimate'] * np.sqrt(10) * 100
    df['Forward_Return'] = (np.log(df.close.shift(-11)) - np.log(df.close.shift(-1))) * 100
    df['Breaches_EWMA'] = np.where(df['Forward_Return'] < df['VaR_99_10D_EWMA'], 1, 0)
    df = df.dropna()
    print('Percentage of Breaches: ', df['Breaches_EWMA'].mean()*100)
    groupby_func = df['Breaches_EWMA'] != df['Breaches_EWMA'].shift()
    count = (df['Breaches_EWMA'].groupby(groupby_func.cumsum()).cumcount())
    df['Breaches_EWMA_Conc'] = df['Breaches_EWMA'] * count
    return df
```


Backtesting Value at Risk with EWMA - 2013-01-01 to 2014-01-01


Backtesting Value at Risk with EWMA - 2014-01-01 to 2015-01-01


Backtesting Value at Risk with EWMA - 2015-01-01 to 2016-01-01


Backtesting Value at Risk with EWMA - 2016-01-01 to 2017-01-01


Backtesting Value at Risk with EWMA - 2017-01-01 to 2018-01-01
Return %
5
0
−5
−10
−15
−20
Return
Value-at-Risk
Breach
# Consecutive Breaches
4
2
0
2017-01
2017-03
2017-05
2017-07
2017-09
2017-11
2018-01
Date

Problem 4

(14 marks) Liquidity-Adjusted VaR (LVaR) is effectively VaR itself plus VaR of the bid/ask spread. The latter is our liquidity adjustment. It has not been introduced in Market Risk lecture, however to compute LVaR simply use the formula:

$$LVaR = VaR + \delta_{Liquidity} \tag{24}$$

$$LVaR = PortfolioValue * [-\mu + Factor * \sigma + \frac{1}{2}(\mu_{Spread} + Factor * \sigma_{Spread})] \tag{25}$$

Use the positive value of the Standard Normal Factor for the correct percentile. For the following cases, report (a) the proportion attributed to VaR and (b) the proportion attributed to liquidity adjustment.

A) Consider a portfolio of USD 16 million composed of shares in a technology company. Daily mean and volatility of its returns are 1% and 3%, respectively. Bid-ask spread also varies with time, its daily mean and volatility are 35 bps and 150 bps. Compute 99%/1D LVaR and attributions to it,

B) Now consider GBP 40 million invested in UK gilts. Take the daily volatility of portfolio returns as 3% and bid-ask spread is 15 bps (no spread volatility). Compute 99%/1D LVaR and attributions. What would happen if the bid-ask spread increases to 125 bps?

Solution.
The LVaR equation 25 was implemented in Python Jupyter Notebooks. A confidence interval of 0.01 was used to produce the positive Standard Normal Factor.

```python
from scipy.stats import norm
def l_var(portfolio_value, mu, sigma, mu_spread, sigma_spread, conf_int):
    var = -1*mu + norm.ppf(1-conf_int) * sigma
    delta_l = 0.5*(mu_spread + norm.ppf(1-conf_int) *sigma_spread)
    return portfolio_value * (var + delta_l), portfolio_value *var, portfolio_value * delta_l
```

```python
l_var(16000000, 0.01, 0.03, 0.0035, 0.015, 0.01)
```

A) $LVaR = \$1,263,808.72$, contributions: $VaR = \$956,646.98$ and $\delta_{Liquidity} = \$307,161.74$

```python
l_var(40000000, 0, 0.03, 0.0015, 0, 0.01)
```

B) $LVaR = £2,821,617.45$, contributions: $VaR = £2,791,617.45$ and $\delta_{Liquidity} = £30,000.0$

```python
l_var(40000000, 0, 0.03, 0.0125, 0, 0.01)
```

When the bid-ask spread increases to 125bps the LVaR increases as it will cost more to exit the position due to the higher spread.
$LVaR = £3,041,617.45$, contributions: $VaR = £2,791,617.45$ and $\delta_{Liquidity} = £250,000.0$