A stock portfolio optimiser

In this work a bi-objective optimisation method for stock portfolio is craeted. The idea is to give decision support for the investor who contemplates what are the optimal weights for each stock in their portfolio regarding the risk they are willing to take.

The method is based on the modern portfolio theory (MPT) which is a theory for finding optimal weights for an investment portfolio that maximises return (expected return) and minimises risk, which is the standard deviation of returns. The method produces an efficient frontier that shows the investor what are the optimal and sub-optimal solutions.

So, determining the weights of selected stocks is the tricky task to which an investor needs decision support. If there are several stocks in a portfolio, it is an endless task to test by hand all the possible weight variations between stocks. Therefore, Monte Carlo simulation will be used. It can run thousands of randomly generated weights for the stocks in a portfolio. Then it can be used for finding the best Sharpe ratio, that is expected return divided by expected volatility. Sharpe ratio is a common measure of the performance of an investment.

The required tools for the method are:

```
import numpy as np
import pandas as pd
import pandas_datareader.data as web
import matplotlib.pyplot as plt
import scipy.optimize as sco
```

The chosen stocks are four Finnish corporations. Time frame for pulling closing prices of the stocks is set to cover the time period from 1.1.2012 to date.

```
stocks = ['KNEBV.HE','UPM.HE','METSA.HE','PON1V.HE']
data = pd.DataFrame()

for stock in stocks:
   data[stock] = web.DataReader(stock, data_source = 'yahoo', start = '2012-1-1')['Adj
Close']
num_stocks = len(stocks)
```

After the portfolio data is created it is time to create returns, that is, gain or loss a given portfolio is expected to produce over the given time frame. The table shows percentual changes in prices.

```
returns=((data)/data.shift(1))-1
```

```
returns.head()
```

Date	KNEBV.HE	UPM.HE	METSA.HE	PON1V.HE
2012-01-02	NaN	NaN	NaN	NaN
2012-01-03	0.018312	0.025014	0.091463	0.054852
2012-01-04	-0.005103	-0.014421	-0.033519	-0.024000
2012-01-05	0.005130	-0.027012	0.011561	-0.001366
2012-01-09	0.000486	-0.005784	0.000000	-0.005472

Next, mean daily returns and covariance are calculated. Covariance is an important statistical tool in MPT. It provides diversification which in return reduces the overall volatility of a portfolio. Also, in this cell the number of runs of random portfolio weights is defined.

```
mean_daily_returns = returns.mean()
cov_matrix = returns.cov()
#Monte Carlo simulation, 25000 runs
num_portfolios = 25000
```

Next, let's have a look at a covariance matrix. In this case it doesn't look particularly good, as there should be negative covariances in order to produce that diversification. However, in this example work with four assets only, it doesn't matter.

```
variance_matrix = returns.cov()*252
variance_matrix
```

	KNEBV.HE	UPM.HE	METSA.HE	PON1V.HE
KNEBV.HE	0.053537	0.027654	0.016745	0.015850
UPM.HE	0.027654	0.081950	0.031259	0.023612
METSA.HE	0.016745	0.031259	0.122168	0.021431
PON1V.HE	0.015850	0.023612	0.021431	0.087943

Now the results will be put into an array to hold them there. The array also holds the weight values for the stocks. This is done with `4+len`. Portfolio return and volatility is calculated by multiplying mean daily returns and randomly set weights with 252 which is the number of trading days in a year.

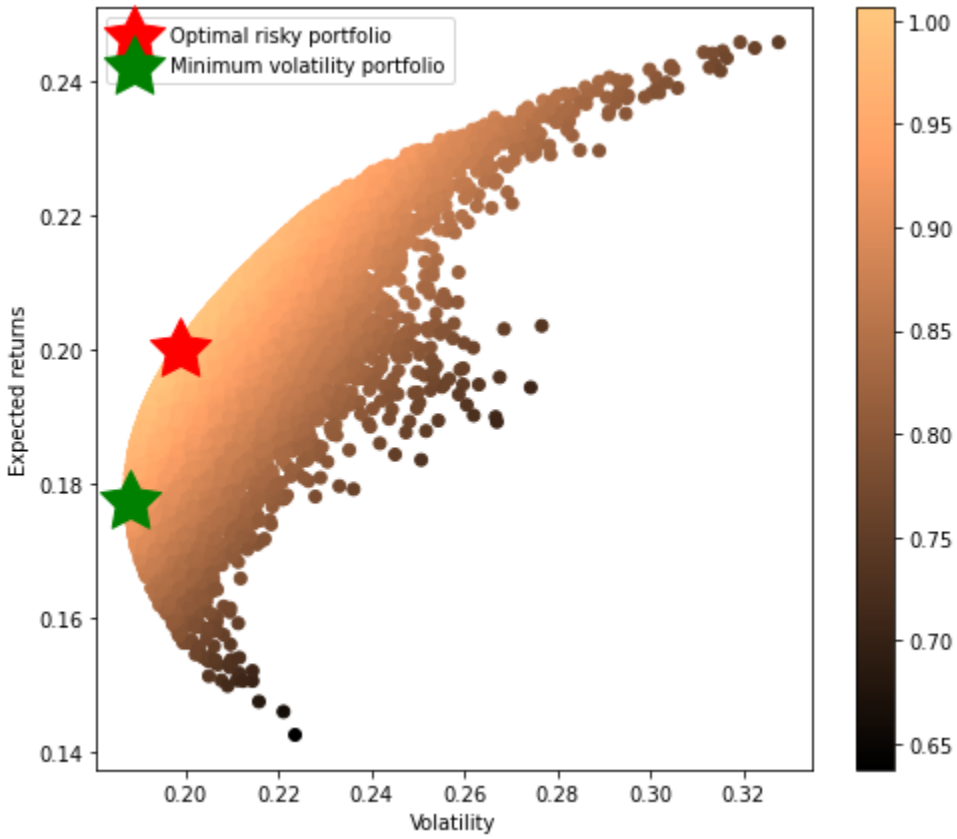
```
results = np.zeros((4+len(stocks)-1,num_portfolios))
for i in range(num_portfolios):
   #random weights for portfolio holdings
   weights = np.array(np.random.random(4))
```

```
    #rebalance weights to sum to 1
    weights /= np.sum(weights)

    #calculate portfolio return and volatility
    portfolio_return = np.sum(mean_daily_returns * weights) * 252
    portfolio_std_dev = np.sqrt(np.dot(weights.T,np.dot(cov_matrix, weights))) *
np.sqrt(252)

    #results into an array
    results[0,i] = portfolio_return
    results[1,i] = portfolio_std_dev
    #store Sharpe Ratio (return / volatility)
    results[2,i] = results[0,i] / results[1,i]
    #iterate through the weight vector and add data to results array
    for j in range(len(weights)):
        results[j+3,i] = weights[j]
#convert results array to Pandas DataFrame
results_frame =
pd.DataFrame(results.T,columns=['ret','stdev','sharpe',stocks[0],stocks[1],stocks[2],s
tocks[3]])
#locate position of portfolio with highest Sharpe Ratio
optimal_risky_port = results_frame.iloc[results_frame['sharpe'].idxmax()]
#locate position of portfolio with minimum standard deviation
min_vol_port = results_frame.iloc[results_frame['stdev'].idxmin()]
```

Finally a plot and tables

```
plt.subplots(figsize=(8, 7))
plt.scatter(results_frame.stdev,results_frame.ret,c=results_frame.sharpe,cmap='RdYlBu'
)
plt.xlabel('Volatility')
plt.ylabel('Expected returns')
plt.colorbar()
plt.scatter(optimal_risky_port[1],optimal_risky_port[0],marker=(5,1,0),color='r',
label='Optimal risky portfolio', s=1000)
plt.scatter(min_vol_port[1],min_vol_port[0],marker=(5,1,0),color='g',label='Minimum
volatility portfolio', s=1000)
plt.legend()
```

The red star shows the point where the Sharpe ratio is highest for the optimal risky portfolio, and the green star shows the point where the standard deviation is lowest for the safest optimal portfolio. The brighter the copper coloured dots are, the better is the Sharpe ratio.



And the weights are

```
print(optimal_risky_port)
```

```
ret          0.200179
stdev        0.198960
sharpe       1.006129
KNEBV.HE     0.225255
UPM.HE       0.243171
METSA.HE     0.269616
PON1V.HE     0.261957
Name: 15099, dtype: float64
```

```
print(min_vol_port)
```

```
ret          0.177238
stdev        0.187937
sharpe       0.943074
```

```
KNEBV.HE    0.475332
UPM.HE      0.142070
METSA.HE    0.135056
PON1V.HE    0.247542
Name: 12319, dtype: float64
```

Now the investor knows what are the weights needed to create risky and less risky portfolios. It has to be acknowledged though, that variance of a return distribution does not necessarily measure the investment risk, as it is expected to do in MPT. Variance can also be seen as a measure of the spread of the return distribution, and not as a factual measure of probability of loss. One could ask why would standard deviation tell about the probability of loss?

Hence, I wouldn't recommend this method for an investor who is interested in maximising returns and minimising risk. But I would recommend this method for an investor who thinks volatility equals risk.