In [6]:

```
import importlib

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import datetime

from model.Portfolio import Portfolio
from model.Optimizer import Optimizer
plt.rcParams["figure.figsize"] = 10, 15
```

In [2]:

```
names = ["C38U", "ND8U", "V01", "AGS"]

p = Portfolio()

# Set risk-free investment as 2%, approximately SSB's returns
p.rf = 0.02

# Add all assets
for name in names:
    p.addAsset(f"data/{name}.csv", name)

# Convert non SGD assets to SGD
p.addExchangeRate("data/forex/SGDEUR.csv", "EUR", True)
p.addExchangeRate("data/forex/USDSGD.csv", "USD", False)
```

Backtest on current allcation

To test the stocks across a longer horizon, I've removed two stocks:

- S63 - Financial data is incomplete and even with secondary sources, price are not adjusted for dividends
- CJLU - For lack of long history

In [5]:

```
currentWeight = [20, 20, 20, 10]

normalisedWeight = np.array(currentWeight)/np.sum(currentWeight)
normalisedWeight
```

Out[5]:

```
array([0.28571429, 0.28571429, 0.28571429, 0.14285714])
```

In [48]:

```
currentResult, currentBtPlot = p.backtest(normalisedWeight)
```

In [50]:

```
currentResult
```

Out[50]:

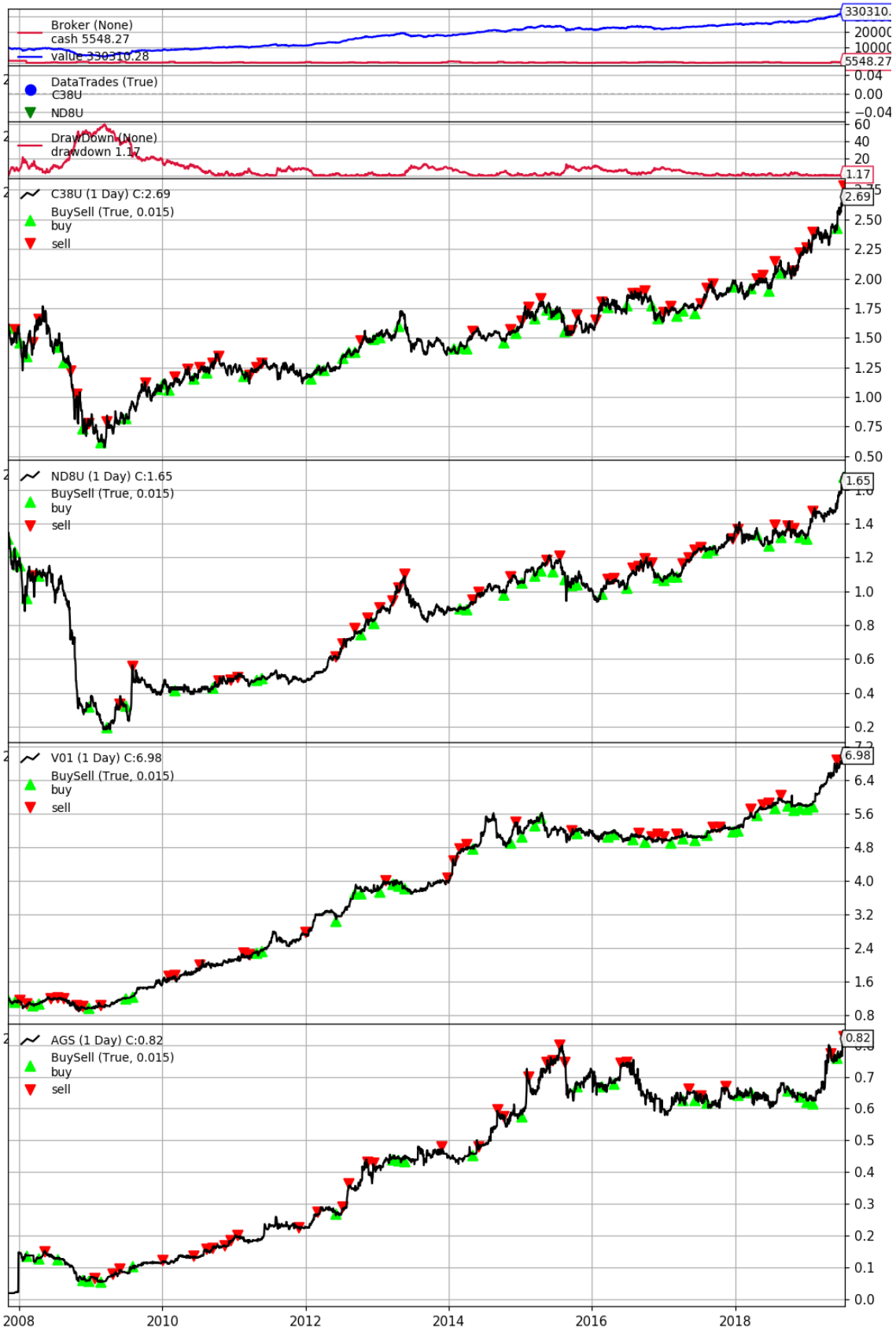
```
{'dateStart': Timestamp('2007-11-05 00:00:00'),
 'dateEnd': Timestamp('2019-07-11 00:00:00'),
 'days': 4266,
 'valueStart': 100000.0,
 'valueEnd': 330310.28483599995,
 'sharpe': 0.4243182173572743,
 'drawdown': 1.1671769036033843,
 'drawdownPeriod': 4,
 'moneydown': 3900.835000000021,
 'maxDrawdown': 59.77069884400001,
 'maxDrawdownPeriod': 725,
 'maxMoneydown': 59770.698844000006,
 'averageReturns': 0.1300993435347989,
 'standardDeviation': 0.2594735248948684,
 'positiveYears': 10,
 'negativeYears': 3,
 'noChangeYears': 0,
 'bestYearReturns': 0.5847528334722043,
 'worstYearReturns': -0.46300804213596736}
```

Testing from 2007 to 2019, we can see that the annual returns is 13%, with standard deviation of 25.9%. At a risk-free rate of 2%, the sharpe is 0.424. If we started with 100k, we will end with 330k. The max drawdown is 59%, taking 725 days to recover. This made sense as investor started right in the global financial crisis. Of the 13 years, 10 have positive returns and 3 have negative returns.

Overall, it seemed like this portfolio cannot weather a financial crisis. 59% drawdown can be quite scary for many investors.

In [49]:

```
currentBtPlot()
```



Out[49]:

```
[[<Figure size 720x1080 with 7 Axes>]]
```

Optimisation

Next, we will attempt to optimise the portfolio without introducing look ahead bias (using time-series k-folds). We will incrementally train the data over longer range of data and getting the average optimised weights for the portfolio.

In [45]:

```
o = Optimizer(p)
optimisedWeight, tests = o.kfoldTs(10)
```

In [46]:

```
optimisedWeight
```

Out[46]:

```
[0.09636578018146558,
 4.44370424788551e-16,
 0.7748326733212899,
 0.12880154649725103]
```

In [47]:

```
tests
```

Out[47]:

```
{'sharpeRaw': [8.353800062585762,
  6.262666882614975,
  9.99640669387286,
  21.908780146514747,
  10.855553336270178,
  13.5232214922503,
  -8.232865251321046,
  -7.44996726220682,
  23.391056060743185,
  27.380874141786872],
 'sharpeAvg': 10.5989526303111,
 'sharpeStd': 11.341993697294722,
 'weightsRaw': [array([0.        , 0.        , 0.61152848, 0.3884715
2]),
  array([0.        , 0.        , 0.87618703, 0.12381297]),
  array([0.06590646, 0.        , 0.82001572, 0.11407782]),
  array([7.98309230e-02, 2.94303969e-15, 8.19529461e-01, 1.00639616e-0
1]),
  array([1.26461972e-01, 5.62545073e-16, 7.79742095e-01, 9.37959326e-0
2]),
  array([1.16944957e-01, 5.71157704e-16, 7.93120050e-01, 8.99349929e-0
2]),
  array([1.49447208e-01, 2.38510925e-16, 7.59266679e-01, 9.12861130e-0
2]),
  array([1.41395098e-01, 1.07295357e-16, 7.60584845e-01, 9.80200568e-0
2]),
  array([1.38339514e-01, 2.11554949e-17, 7.63938577e-01, 9.77219096e-0
2]),
  array([0.14533167, 0.        , 0.76441379, 0.09025454])],
 'weightsStd': 0    5.487068e-02
 1    8.601370e-16
 2    6.481616e-02
 3    8.718141e-02
 dtype: float64}
```

The test is shows the stability of the portfolio. From the sharpe ratio column, we can see that this portfolio's sharpe can swing quite widely.

From the standard deviation report, we can see that the allocation can generally change by 5-10% or so for most stocks. Also, we can see that the allocation for ND8U does not change much for the different tested periods. In this case, it recommends to keep it to very low percentage (around 1.39% of total allocation)

In [42]:

```
optimisedResult, optimisedBtPlot = p.backtest(optimisedWeight)
```

In [43]:

```
optimisedResult
```

Out[43]:

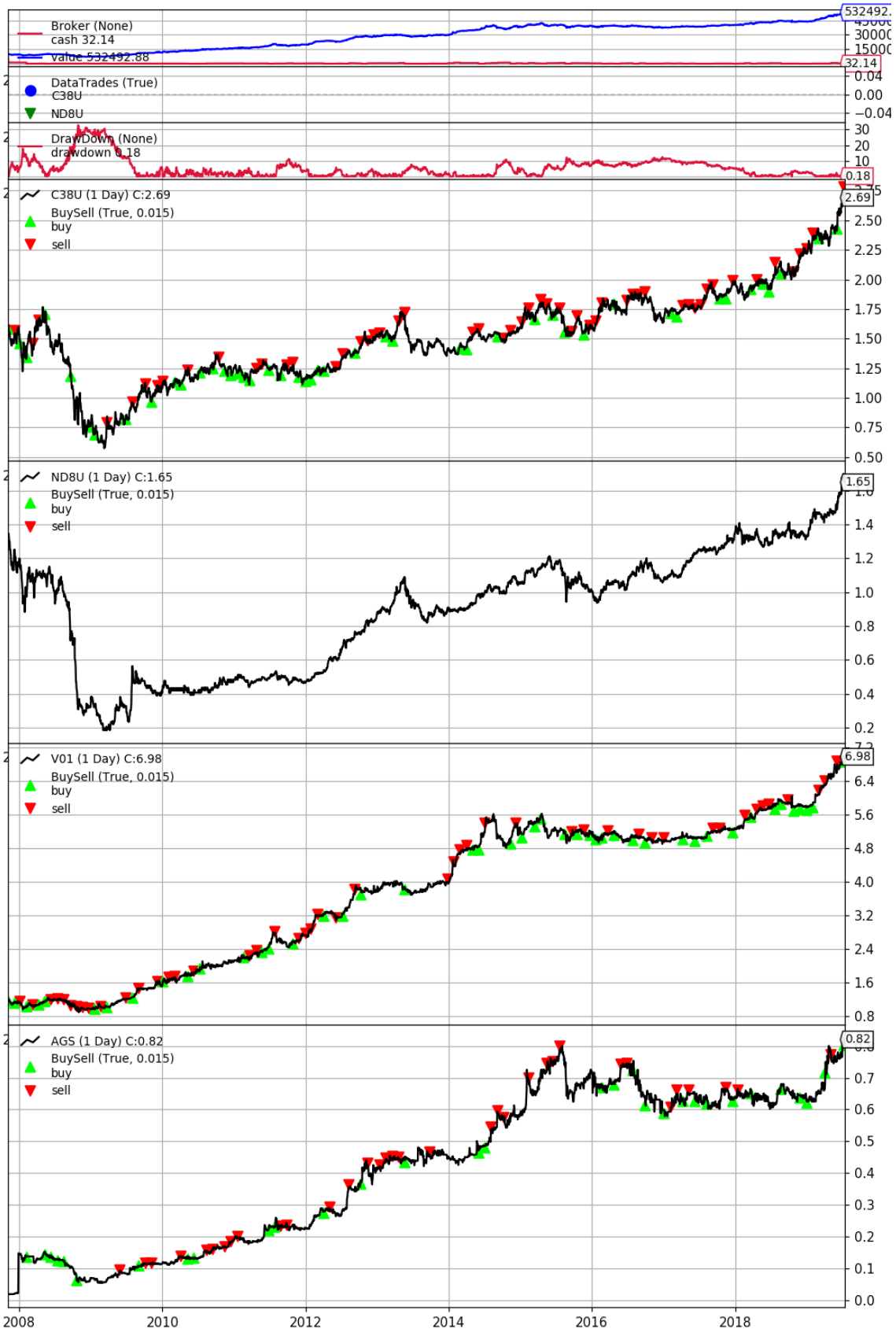
```
{'dateStart': Timestamp('2007-11-05 00:00:00'),
 'dateEnd': Timestamp('2019-07-11 00:00:00'),
 'days': 4266,
 'valueStart': 100000.0,
 'valueEnd': 532492.8793720001,
 'sharpe': 0.6373748545708656,
 'drawdown': 0.17912416163516315,
 'drawdownPeriod': 5,
 'moneydown': 955.5349999999162,
 'maxDrawdown': 32.74167518314408,
 'maxDrawdownPeriod': 730,
 'maxMoneydown': 52715.60014700005,
 'averageReturns': 0.1575017395941445,
 'standardDeviation': 0.215731352763708,
 'positiveYears': 10,
 'negativeYears': 3,
 'noChangeYears': 0,
 'bestYearReturns': 0.5651507584246767,
 'worstYearReturns': -0.23151817385723472}
```

From here, we can see tha thet optimisation increase returns from 13% to 15.7%, reduced standard deviation of returns from to 25.9% to 21.6%.

Note that the optimiser does not return the weights for the most optimised performance in this backtest. We can theorotically find a weight better than this results but it will run the risk of over-fitting it to this test.

In [51]:

```
optimisedBtPlot()
```



Out[51]:

```
[[<Figure size 720x1080 with 7 Axes>]]
```

In [52]:

```
dict(zip(names, np.array(optimisedWeight)*100))
```

Out[52]:

```
{'C38U': 9.636578018146558,
 'ND8U': 4.4437042478855104e-14,
 'V01': 77.48326733212899,
 'AGS': 12.880154649725103}
```

In summary, the optimiser recommends the above allocation of portfolio. In this case, it also chose to NOT invest in ND8U.