Stock Market Analysis

By:
Russell Moore
Mike Cutno
Jeff Smith
Francisco Franco

Executive Summary

The goal of the project is to analyze the ideal portfolio of stocks held by four billionaires through the use of quantitative analysis, Monte Carlo simulations and Vectorbt, a backtesting library that allows us to quickly test a variety of trading strategies.

Warren Buffett	Cathie Wood
◦ Apple ◦ Bank of America ◦ Coca-Cola ◦ Chevron ◦ American Express	◦ Tesla Inc. ◦ Zoom Video Communication - Class A ◦ Teledoc Health Inc. ◦ Roku
Bill Ackman	**Ray Dalio**
◦ Lowe's ◦ Chipotle ◦ Restaurant Brands international ◦ Hilton worldwide Holdings Inc. ◦ Howard Hughes Corporation	◦ SPY ◦ Procter & Gamble ◦ Vanguard VWO ◦ Pepsi Co ◦ Johnson & Johnson

Quantitative Analysis of the Stocks

We eliminated Roku and Vanguard VWO because Roku did not have 5 years of data to analyze and since we are going for a long term investment, we felt it was wise to reject the stock. Vanguard VWO is not a stock, but rather an emerging markets ETF and we thought we should continue only with actually stocks, with the exception below.

We did include SPY, because it is the S&P500 ETF. It represents the market metric against which to analyze the individual stocks and most investors have a slice of SPY in their portfolio.

Sharpe Ratio vs. Sortino Ratio Functions

def sharpe_ratio(daily_return):

annual_average_return = daily_return.mean() * 255

annual_std = daily_return.std() * np.sqrt(255)

return annual_average_return / annual_std

sharpe_ratios = daily_returns_df.apply(sharpe_ratio)

```
SPY      0.690552
HLT      0.840971
LOW      0.840991
TSLA     0.850041
CMG      1.418322
```

(simplified and modified from the website https://www.codearmo.com/blog/sharpe-sortino-and-calmar-ratios-python)

def sortino_ratio(daily_return):

annual_average_return = daily_return.mean() * 255

annual_std_neg = daily_return[daily_return < 0].std()*np.sqrt(255)

Return annual_average_return/annual_std_neg

sortino_ratios = daily_returns_df.apply(sortino_ratio)

```
HLT      0.598208
SPY      0.601334
LOW      0.727482
TSLA     0.764491
CMG      0.963386
```

Initial Observations

We selected the stocks to continue with our analysis based on what we learned in class, but there are the following issues worth mentioning:

- Standard deviations were affected by the stocks splits, in particular, the 20:1 stock split of Amazon on June 6, 2022, which affected its standard deviation and its Sharpe Ratio. We nonetheless continued to exclude it from the Monte Carlo Simulation.
- When we were deciding about how to decide on the best stock to choose, we went with the Sharpe ratio as that was taught to us in class, but it dawned on us that the denominator is the standard deviation and that understates the quality of performance. The main problem is that the risk in the denominator doesn't distinguish between upside and downside volatility. With respect to the risk measure, large gains are viewed as equally as bad as large losses. The Sortino ratio, which was raised in pre-work section, uses losses instead of volatility as the risk measure, thereby eliminating penalization for large gains. It seemed interesting to raise this observation. This Sortino ratio analysis was inspired by the book Unknown Market Wizards: The best traders you've never heard by Jack D.

Beta

“Beta is a measure of a stock's volatility in relation to the overall market. By definition, the market, such as the S&P 500 Index, has a beta of 1.0, and individual stocks are ranked according to how much they deviate from the market.” (https://www.investopedia.com/investing/beta-know-risk/#:~:text=Beta%20is%20a%20measure%20of,has%20a%20beta%20above%201.0.)

Based on our analysis, Tesla has the highest beta, which means it swings more than the market (1.598)

Chipotle has the lowest beta (0.8869), so it swings less than the market.

We recognize their strength because of this relationship and because they have the two highest annual average returns.

Monte Carlo Simulations

Monte Carlo Simulations: are a large set of computational algorithms designed to predict an outcome based on the examination of random variables and their probability distributions.

In our case, Monte Carlo is used to predict future prices by noting standard deviation of daily returns and using them to predict next day prices. The model may be projected for a larger amount of time and an assessment of probability of annual returns for a certain portfolio may be inferred.

Abstracting MC

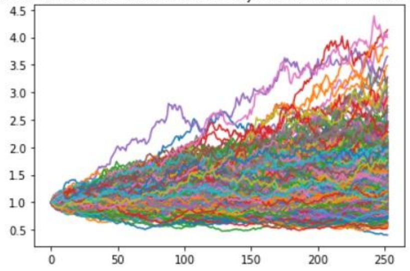
```python
def Monte_Sim_Ideal(data):
    """
    Finds portfolio weights for optimal portfolio based on Monte Carlo Simulation

    data: dataframe
    """
    #First we make a list of all possible weights for 5 stocks in increments of 0.1:
    from itertools import product
    s = [list(p/20 for p in prd)
         for prd in product(range(11), repeat=5) if sum(prd) == 20]
    #We need dummy variables to store best simulations
    weight_ideal=[]
    mean_ideal_cumulative_return = 0
    for i in range(len(s)):
        MC = MCSimulation(
        portfolio_data = data,
        weights = s[i],
        num_simulation = 100,
        num_trading_days = 252)
        MC.calc_cumulative_return()
        MC_table=MC.summarize_cumulative_return()
        if round(MC_table[1],2) > mean_ideal_cumulative_return:
            weight_ideal = s[i]
            mean_ideal_cumulative_return = round(MC_table[1],2)
            ci_upper_ideal_cumulative_return = round(MC_table[9],2)
            ci_lower_ideal_cumulative_return = round(MC_table[8],2)
    MC_ideal =MCSimulation(
        portfolio_data = data,
        weights = weight_ideal,
        num_simulation = 500,
        num_trading_days = 252)
    MC_ideal.calc_cumulative_return()
    print(MC_ideal.plot_simulation())
    print(f"The weights with the highest mean portfolio return are ${weight_ideal}, wh
```

Sequential Search Algorithm:

- List is every possible iteration of portfolio weights
- Loops over list and calculates Monte Carlo for each set of weights
- Stores the highest mean cumulative return
- Returns highest mean cumulative return weights and results upon termination

```
The weights with the highest mean portfolio return are $[0.1, 0.05, 0.5, 0.3, 0.05], which with a probability of 95% will
fall between $0.75 and $3.02 for a $1 investment within the next trading year.
```

500 Simulations of Cumulative Portfolio Return Trajectories Over the Next 252 Trading Days.



1 year MC simulation summary statistics

```
count            500.000000
mean               1.221597
std                0.193586
min                0.793317
25%                1.078374
50%                1.198896
75%                1.341152
max                1.893410
95% CI Lower       0.895817
95% CI Upper       1.654095
Name: 252, dtype: float64
```

Project goal: use a new library

1. Research by trial and error.
2. Roadblocks: what works with our data?
3. Best to use a library that is currently supported.
4. VectorBT: works with Alpaca data, many features, and useful guides.

How should we use it?

1. Compare different trading strategies.
2. Strengthen our analysis with backtesting.
3. Retest simulations.

Trading the moving average: higher returns?

```
Start                          2017-08-01 04:00:00+00:00
End                            2022-08-01 04:00:00+00:00
Period                              1259 days 00:00:00
Start Value                                      100.0
End Value                                   190.075805
Total Return [%]                             90.075805
Benchmark Return [%]                        169.592536
Max Gross Exposure [%]                           100.0
Total Fees Paid                               8.732789
Max Drawdown [%]                             44.387325
Max Drawdown Duration                444 days 00:00:00
Total Trades                                      33.4
Total Closed Trades                               32.4
Total Open Trades                                  1.0
Open Trade PnL                               25.703749
Win Rate [%]                                 40.648516
Best Trade [%]                               75.418853
Worst Trade [%]                             -26.427111
Avg Winning Trade [%]                        16.253428
Avg Losing Trade [%]                         -5.743422
Avg Winning Trade Duration   38 days 03:38:26.445554445
Avg Losing Trade Duration    12 days 14:39:50.016131237
Profit Factor                                  1.32738
Expectancy                                    1.929943
Sharpe Ratio                                  0.597522
Calmar Ratio                                  0.536495
Omega Ratio                                   1.144958
Sortino Ratio                                 0.895613
Name: agg_func_mean, dtype: object
```

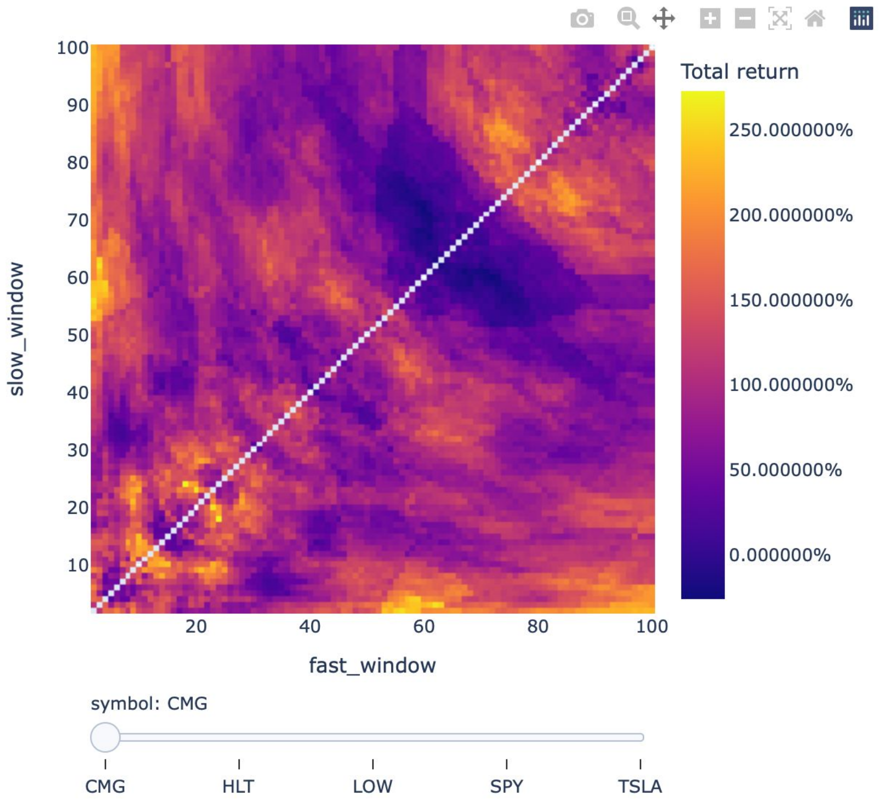
[106]:
```
fast_ma = vbt.MA.run(pf_4_data_pivoted, 10, short_name='fast')
slow_ma = vbt.MA.run(pf_4_data_pivoted, 20, short_name='slow')

entries = fast_ma.ma_crossed_above(slow_ma)
entries
```

[106]:

fast_window	10									
slow_window	20									
	close					open				
symbol	CMG	HLT	LOW	SPY	TSLA	CMG	HLT	LOW	SPY	TSLA
timestamp										
2017-08-01 04:00:00+00:00	False	False	False	False	False	False	False	False	False	False
2017-08-02 04:00:00+00:00	False	False	False	False	False	False	False	False	False	False
2017-08-03 04:00:00+00:00	False	False	False	False	False	False	False	False	False	False
2017-08-04 04:00:00+00:00	False	False	False	False	False	False	False	False	False	False
2017-08-07 04:00:00+00:00	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...
2022-07-26 04:00:00+00:00	True	False	False	False	False	True	False	False	False	False
2022-07-27 04:00:00+00:00	False	False	False	False	False	False	False	False	False	False
2022-07-28 04:00:00+00:00	False	False	False	False	False	False	False	False	False	False
2022-07-29 04:00:00+00:00	False	False	False	False	False	False	False	False	False	False
2022-08-01 04:00:00+00:00	False	False	False	False	False	False	False	False	False	False

1259 rows × 10 columns

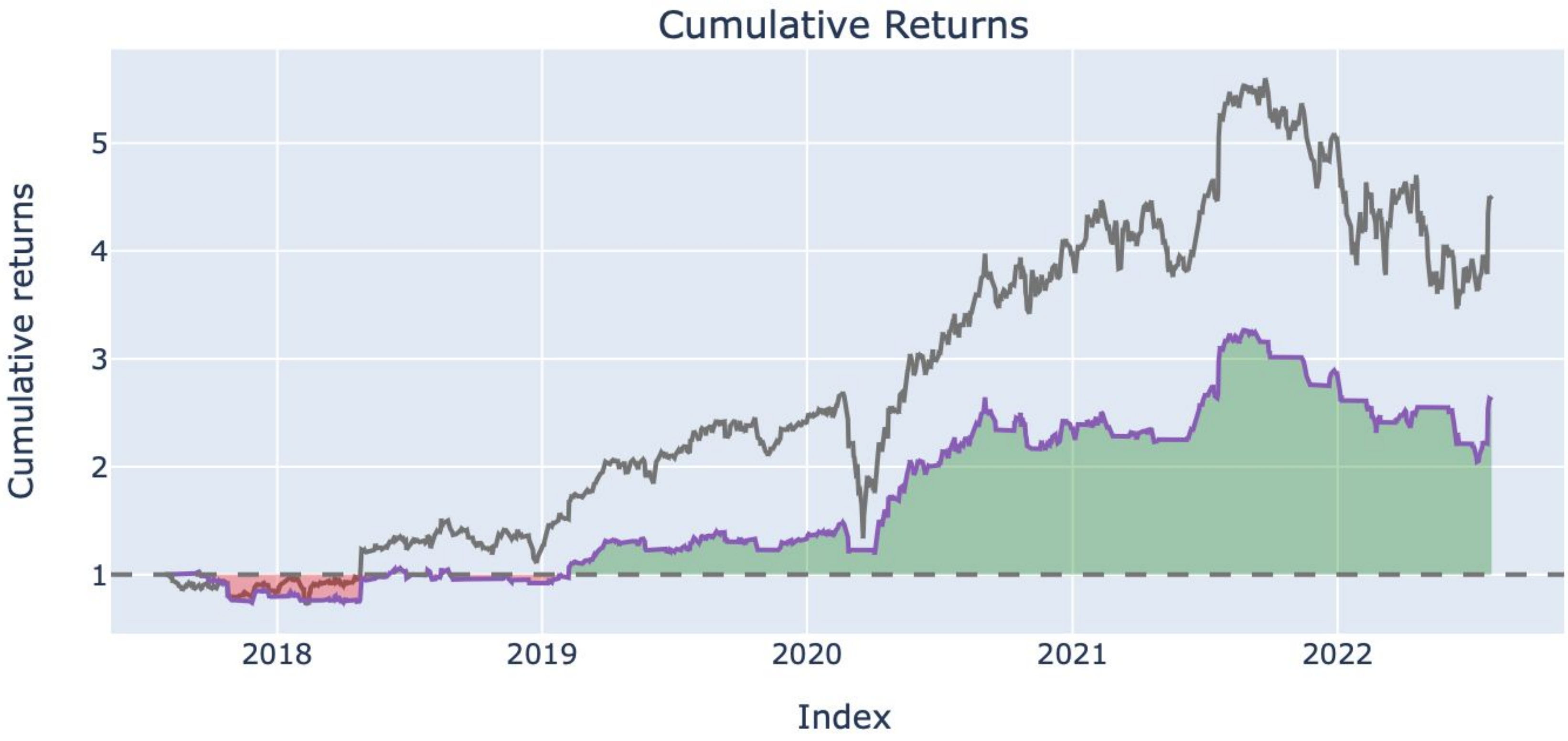
Visualize the return windows



Buy/Sell

Chipotle
Close
Buy
Sell
Closed - Profit
Closed - Loss
Open
Benchmark
Value
Orders
Price
2000
1500
1000
500

Cumulative Returns



Portfolio Optimization through backtesting (VBT)

Using a random search function to generate weights

```
[233]: #define parameters
num_tests = 2000

vbt.settings.array_wrapper['freq'] = 'days'
vbt.settings.returns['year_freq'] = '252 days'
vbt.settings.portfolio['seed'] = 42
vbt.settings.portfolio.stats['incl_unrealized'] = True
```

```
[188]: np.random.seed(42)

# Generate random weights, n times
weights = []
for i in range(num_tests):
    w = np.random.random_sample(len(pf_4_tickers))
    w = w / np.sum(w)
    weights.append(w)

print(len(weights))
```

```
2000
```

2000 weight combinations

```
# Build column hierarchy such that one weight corresponds to one price series
_price = price.vbt.tile(num_tests, keys=pd.Index(np.arange(num_tests), name='symbol_group'))
_price = _price.vbt.stack_index(pd.Index(np.concatenate(weights), name='weights'))

print(_price.columns)
```

```
MultiIndex([( 0.13319702814025883,    0,  'CMG'),
            ( 0.33810081711389406,    0,  'HLT'),
            ( 0.26031768763785473,    0,  'LOW'),
            (  0.2128998389048247,    0,  'SPY'),
            ( 0.05548462820316767,    0, 'TSLA'),
            ( 0.06528491964469331,    1,  'CMG'),
            ( 0.02430844330237927,    1,  'HLT'),
            (  0.3625014516740258,    1,  'LOW'),
            (  0.2515713061862386,    1,  'SPY'),
            ( 0.29633387919266296,    1, 'TSLA'),
            ...
            (  0.2056564359049325, 1998,  'CMG'),
            ( 0.14846396871443943, 1998,  'HLT'),
            ( 0.21512097636364197, 1998,  'LOW'),
            (  0.3738566007394396, 1998,  'SPY'),
            (0.056902018277546554, 1998, 'TSLA'),
            ( 0.25860265182212094, 1999,  'CMG'),
            (  0.2706191852849979, 1999,  'HLT'),
            (  0.2854538191129893, 1999,  'LOW'),
            ( 0.11985160754099378, 1999,  'SPY'),
            (  0.0654727362388982, 1999, 'TSLA')],
           names=['weights', 'symbol_group', 'symbol'], length=10000)
```

Find the best weight group

[194]:

```
# Get index of the best group according to the target metric
best_symbol_group = pf.sharpe_ratio().idxmax()

print(best_symbol_group)
```

```
995
```

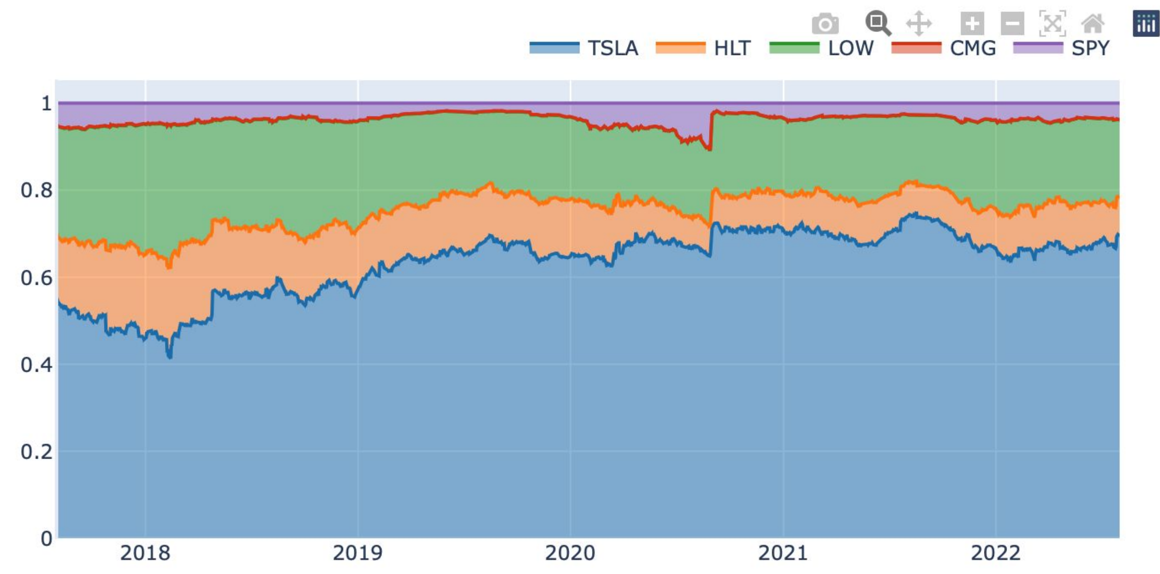
[195]:

```
# Print best weights
print(weights[best_symbol_group])
```

```
[0.54721425 0.15121899 0.2523137  0.00115563 0.04809743]
```

Returns and stats (yearly)

```
Start                                2017-08-01 04:00:00+00:00
End                                  2022-08-01 04:00:00+00:00
Period                                      1259 days 00:00:00
Start Value                                              100.0
End Value                                           352.965381
Total Return [%]                                    252.965381
Benchmark Return [%]                                169.611598
Max Gross Exposure [%]                                   100.0
Total Fees Paid                                            0.0
Max Drawdown [%]                                     50.241248
Max Drawdown Duration                        178 days 00:00:00
Total Trades                                                 5
Total Closed Trades                                          0
Total Open Trades                                            5
Open Trade PnL                                      252.965381
Win Rate [%]                                               NaN
Best Trade [%]                                             NaN
Worst Trade [%]                                            NaN
Avg Winning Trade [%]                                      NaN
Avg Losing Trade [%]                                       NaN
Avg Winning Trade Duration                                 NaT
Avg Losing Trade Duration                                  NaT
Profit Factor                                              NaN
Expectancy                                                 NaN
Sharpe Ratio                                          0.965701
Calmar Ratio                                          0.571568
Omega Ratio                                           1.204248
Sortino Ratio                                         1.434835
Name: 995, dtype: object
```



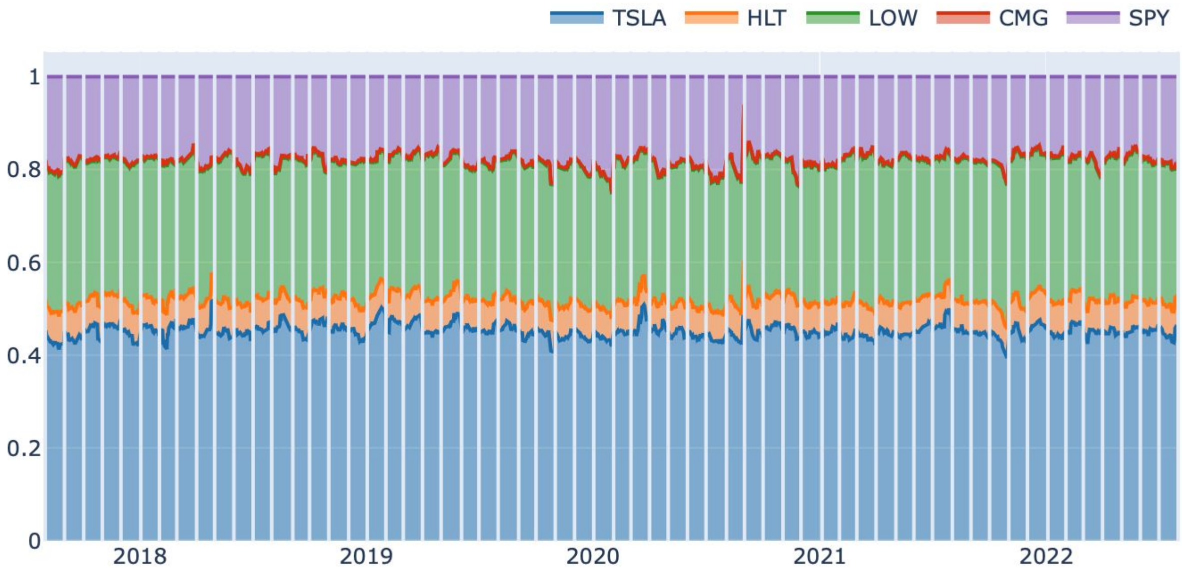
What if we rebalance our portfolio monthly?

```
print(weights[rb_best_symbol_group])
```

```
[0.45112508 0.06630845 0.30034533 0.00317653 0.17904461]
```

```
print(rb_pf.iloc[rb_best_symbol_group].stats())
```

```
Start                                2017-08-01 04:00:00+00:00
End                                  2022-08-01 04:00:00+00:00
Period                                      1259 days 00:00:00
Start Value                                              100.0
End Value                                            429.99929
Total Return [%]                                     329.99929
Benchmark Return [%]                                169.611598
Max Gross Exposure [%]                                   100.0
Total Fees Paid                                            0.0
Max Drawdown [%]                                     51.208036
Max Drawdown Duration                        184 days 00:00:00
Total Trades                                               143
Total Closed Trades                                        138
Total Open Trades                                            5
Open Trade PnL                                      139.587613
Win Rate [%]                                         95.652174
Best Trade [%]                                      396.882035
Worst Trade [%]                                     -17.808683
Avg Winning Trade [%]                                 53.86536
Avg Losing Trade [%]                                -10.356529
Avg Winning Trade Duration         657 days 02:21:49.090909096
Avg Losing Trade Duration                    349 days 08:00:00
Profit Factor                                       119.265739
Expectancy                                            1.379795
Sharpe Ratio                                          1.070751
Calmar Ratio                                          0.662087
Omega Ratio                                           1.227845
Sortino Ratio                                         1.515595
Name: 296, dtype: object
```



Run the MC Sim again.

```
count             500.000000
mean                1.221597
std                 0.193586
min                 0.793317
25%                 1.078374
50%                 1.198896
75%                 1.341152
max                 1.893410
95% CI Lower        0.895817
95% CI Upper        1.654095
Name: 252, dtype: float64
```

```
count             500.000000
mean                1.271606
std                 0.248733
min                 0.719494
25%                 1.095463
50%                 1.247633
75%                 1.417314
max                 2.223889
95% CI Lower        0.861348
95% CI Upper        1.854673
Name: 252, dtype: float64
```

Next steps: what are the possibilities?

1. Can we use to machine learning to refine our model, make a stronger portfolio, reduce computational load, and increase efficiency?
2. Will more innovative analysis change our model?
3. How can we use forecasting to make a better model?