LEARNDATASCI

Author: Georgios Efstathopoulos
Quantitative Analyst

Python for Finance, Part 3: Moving Average Trading Strategy

Expanding on the previous article, we'll be looking at how to incorporate recent price behaviors into our strategy

Contents Index +

In the previous article of this series, we continued to discuss general concepts which are fundamental to the design and backtesting of any quantitative trading strategy. In detail, we have discussed about

1. relative and log-returns, their properties, differences and how to use each one,
2. a generic representation of a trading strategy using the normalised asset weights $w_i\ (t)$ for a set of N

tradable assets and

1. a very simple, yet profitable strategy, the way to represent it and how to calculate its total return.

If you just found this article, see Part 1 and Part 2.

In this article, we will start designing a more complex trading strategy, which will have non-constant weights $w_i\ (t)$, and thus adapt in some way to the

recent behaviour of the price of our assets.

We will again assume we ha
and Microsoft stocks (with t
500 Index (ticker **^GSPC**).

As a reminder, the datafram
the following format:


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
%matplotlib inline
import seaborn as sns
sns.set(style='darkgrid', context='talk', palette='Dark2')

my_year_month_fmt = mdates.DateFormatter('%m/%y')

data = pd.read_pickle('./data.pkl')
data.head(10)
```

	AAPL	MSFT	^GSPC
2000-01-03	3.625643		
2000-01-04	3.319964		
2000-01-05	3.368548		
2000-01-06	3.077039	37.120080	1403.449951
2000-01-07	3.222794	37.605172	1441.469971
2000-01-10	3.166112	37.879354	1457.599976
2000-01-11	3.004162	36.909170	1438.560059
2000-01-12	2.823993	35.706986	1432.250000
2000-01-13	3.133722	36.381897	1449.680054
2000-01-14	3.253159	37.879354	1465.150024

MOVING AVERAGE CONSIDERATIONS

One of the oldest and simplest trading strategies that exist is the one that uses a moving average of the price (or returns) timeseries to proxy the recent trend of the price.

The idea is quite simple, yet powerful; if we use a (say) 100-day moving average of our price time-series, then a significant portion of the daily price noise will have been "averaged-out". Thus, we can can observe more closely the longer-term behaviour of the asset.

Let us, again, calculate the rolling *simple moving averages (SMA)* of these three timeseries as follows. Remember, again, that when calculating the M days SMA, the first $M-1$ are not valid, as M prices are required for the first moving average data point.

```
# Calculating the short-window simple moving average
short_rolling = data.rolling(window=20).mean()
```

```
short_rolling.head(20)
```


	AAPL		
2000-01-03	NaN		
2000-01-04	NaN		
2000-01-05	NaN	NaN	NaN
2000-01-06	NaN	NaN	NaN
2000-01-07	NaN	NaN	NaN
2000-01-10	NaN	NaN	NaN
2000-01-11	NaN	NaN	NaN
2000-01-12	NaN	NaN	NaN
2000-01-13	NaN	NaN	NaN
2000-01-14	NaN	NaN	NaN
2000-01-17	NaN	NaN	NaN
2000-01-18	NaN	NaN	NaN
2000-01-19	NaN	NaN	NaN
2000-01-20	NaN	NaN	NaN
2000-01-21	NaN	NaN	NaN
2000-01-24	NaN	NaN	NaN
2000-01-25	NaN	NaN	NaN
2000-01-26	NaN	NaN	NaN
2000-01-27	NaN	NaN	NaN
2000-01-28	3.342434	36.389278	1429.120007

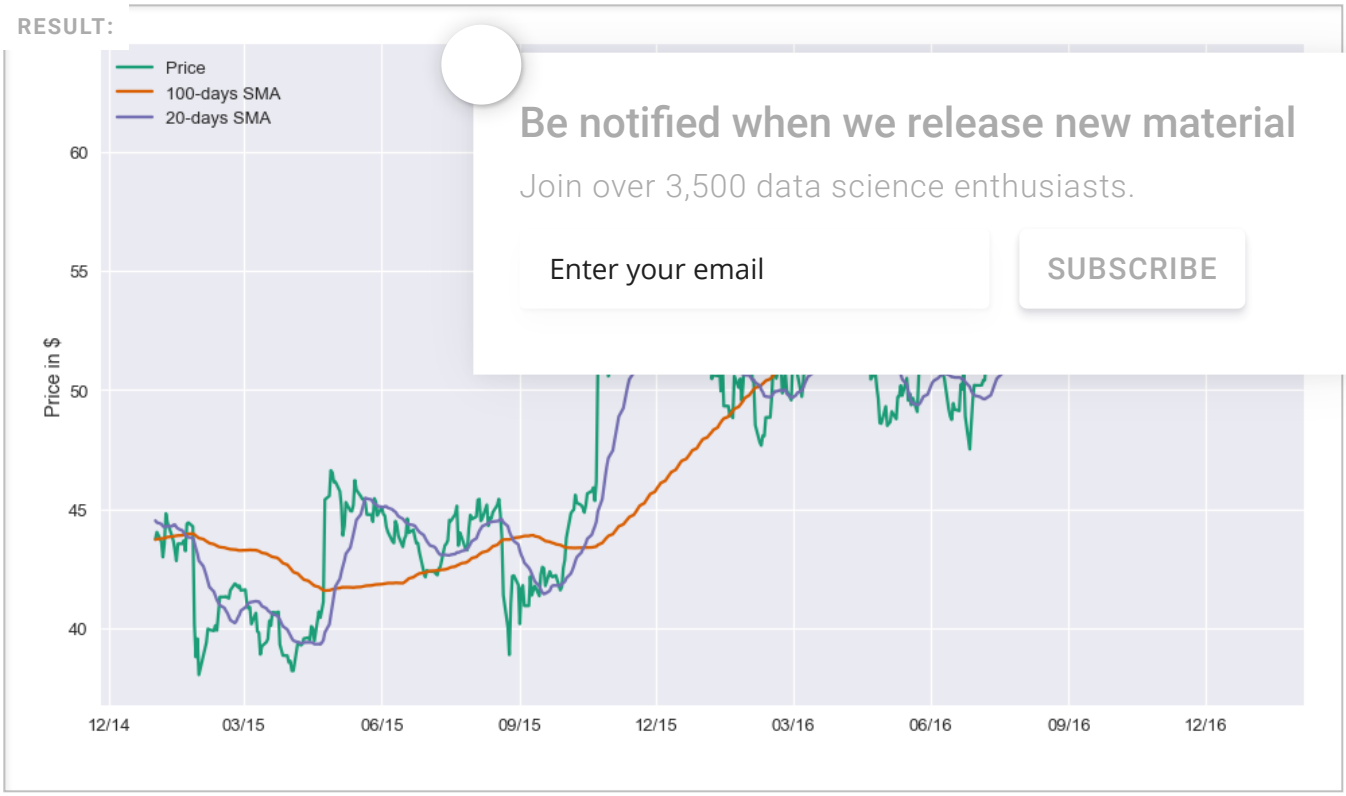
```
# Calculating the long-window simple moving average
long_rolling = data.ro
long_rolling.tail()
```


	AAPL		
2016-12-26	110.958205	58.418182	2176.628791
2016-12-27	111.047874	58.476117	2177.500190
2016-12-28	111.140589	58.532936	2178.244490
2016-12-29	111.233698	58.586112	2178.879189
2016-12-30	111.315270	58.635267	2179.426990

Let us plot the last 22 years for these three timeseries for Microsoft stock, to get a feeling about how these behave.

```
start_date = '2015-01-01'
end_date = '2016-12-31'

fig, ax = plt.subplots(figsize=(16,9))

ax.plot(data.loc[start_date:end_date, :].index, data.loc[start_date
ax.plot(long_rolling.loc[start_date:end_date, :].index, long_rolling
ax.plot(short_rolling.loc[start_date:end_date, :].index, short_roll

ax.legend(loc='best')
ax.set_ylabel('Price in $')
ax.xaxis.set_major_formatter(my_year_month_fmt)
```

RESULT:



It is straightforward to observe that SMA timeseries are much less noisy than the original price timeseries. However, this comes at a cost: SMA timeseries lag the original price timeseries, which means that changes in the trend are only seen with a delay (lag) of L days.

How much is this lag L? For a SMA moving average calculated using M days, the lag is roughly $\frac{M}{2}$ days. Thus, if we are using a 100 days SMA, this means we may be late by almost 50 days, which can significantly affect our strategy.

One way to reduce the lag induced by the use of the SMA is to use the so-called Exponential Moving Average (EMA), defined as

$$\begin{aligned}\mathrm{EMA}\,(t) &= (1-\alpha)\,\mathrm{EMA}\,(t-1) + \alpha\, p\,(t)\\ \mathrm{EMA}\,(t_0) &= p\,(t_0)\end{aligned}$$

where $p\,(t)$ is the price at time t and α is called the decay parameter for the EMA. α is related to the lag as

$$\alpha = \frac{1}{L+1}$$

and the length of the window (span) M as

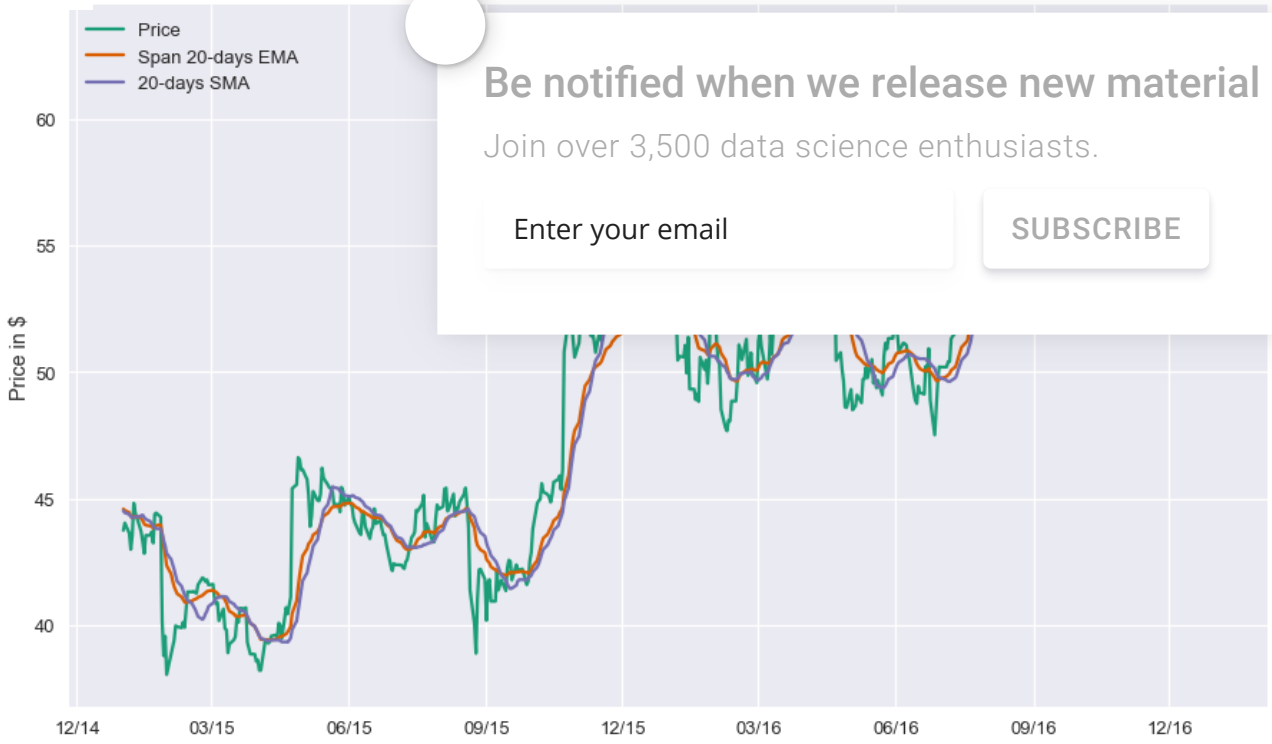
$$\alpha = \frac{2}{}$$

.

The reason why EMA reduce
observations, whereas the S
Pandas, calculating the expo
a lag value, from which the decay parameter α is automatically calculated. To be able to compare with the short-time SMA we will use a span value of 20.

```
# Using Pandas to calculate a 20-days span EMA. adjust=False specif
ema_short = data.ewm(span=20, adjust=False).mean()

fig, ax = plt.subplots(figsize=(15,9))

ax.plot(data.loc[start_date:end_date, :].index, data.loc[start_date
ax.plot(ema_short.loc[start_date:end_date, :].index, ema_short.loc[
ax.plot(short_rolling.loc[start_date:end_date, :].index, short_roll

ax.legend(loc='best')
ax.set_ylabel('Price in $')
ax.xaxis.set_major_formatter(my_year_month_fmt)
```

RESULT:



A MOVING AVERAGE TRADING STRATEGY

Let us attempt to use the moving averages calculated above to design a trading strategy. Our first attempt is going to be relatively straghtforward and is going to take advantage of the fact that a moving average timeseries (whether SMA or EMA) lags the actual price behaviour.

Bearing this in mind, it is natural to assume that when a change in the long term behaviour of the asset occurs, the actual price timeseries will react faster than the EMA one. Therefore, we will consider the crossing of the two as potential trading signals.

1. When the price timeseries $p\left(t\right)$ crosses the EMA timeseries $e\left(t\right)$ from below, we will close any existing short position and go long (buy) one unit of the asset.
2. When the price timeseries $p\left(t\right)$ crosses the EMA timeseries $e\left(t\right)$ from above, we will close any existing long position and go short (sell) one unit of the asset.

Moving average crossover illustration

How is this translated to the framework described in our previous article about the weights $w(t)$?

Well for this strategy it is pretty straightforward. All we need is to have a long position, i.e. $w_i(t)$ >0, as long as the price timeseries is above the EMA timeseries and a short position, i.e. $w_i(t)$ < 0, as long as the price timeseries is below the EMA timeseries.

Since, at this point, we are not interested yet in position sizing, we will assume that we use all our funds available to trade asset i. We will also assume that our funds are split equally across all 3 assets (MSFT, AAPL and ^GSPC).

Based on these assumptions, our strategy for each of the assets $i, i = 1, \ldots, 3$ can be translated as follows:

- Go long condition: If $p_i(t) > e_i(t)$, then $w_i(t) = \frac{1}{3}$
- Go short condition: If $p_i(t) < e_i(t)$, then $w_i(t) = -\frac{1}{3}$

Whenever, the trade conditi[illegible] are satisfied, the weights are $\frac{1}{3}$ because $\frac{1}{3}$ of the total funds are assigne[illegible] all of the available funds are

How is this implemented in [illegible] difference between then pric[illegible]


```
# Taking the difference between the prices and the EMA timeseries
trading_positions_raw = data - ema_short
trading_positions_raw.tail()
```

	AAPL	MSFT	^GSPC
2016-12-26	2.039488	1.040200	20.465712
2016-12-27	2.511890	0.977103	23.121693
2016-12-28	1.822235	0.623249	3.765377
2016-12-29	1.621664	0.482954	2.809706
2016-12-30	0.647437	-0.246519	-6.894490

```
# Taking the sign of the difference to determine whether the price
trading_positions = trading_positions_raw.apply(np.sign) * 1/3
trading_positions.tail()
```

	AAPL	MSFT	^GSPC
2016-12-26	0.333333		
2016-12-27	0.333333		
2016-12-28	0.333333		
2016-12-29	0.333333	0.333333	0.333333
2016-12-30	0.333333	-0.333333	-0.333333

One Final Caveat

Before seeing the performance of this strategy, let us focus on the first day t_o when the price timeseries $p(t_o)$ crosses above and EMA timeseries $e_i(t_o)$. Since $p(t_o) > e_i(t_o)$. At that point the trading weight $w_i(t_o)$ becomes positive, and thus according to our trading strategy, we need to set for that day $w_i(t_o) = \frac{1}{3}$.

However, bear in mind that $p(t_o)$ is the price of the asset at the close of day t_o. For this reason, we will not know that $p(t_o) > e_i(t_o)$ until the close of the trading day. Therefore, when calculating the returns of the strategy, to assume that on day t_o we had a long position is an error; it is equivalent to us peaking into the future, since we only know we have to go long at the end of day t_o.

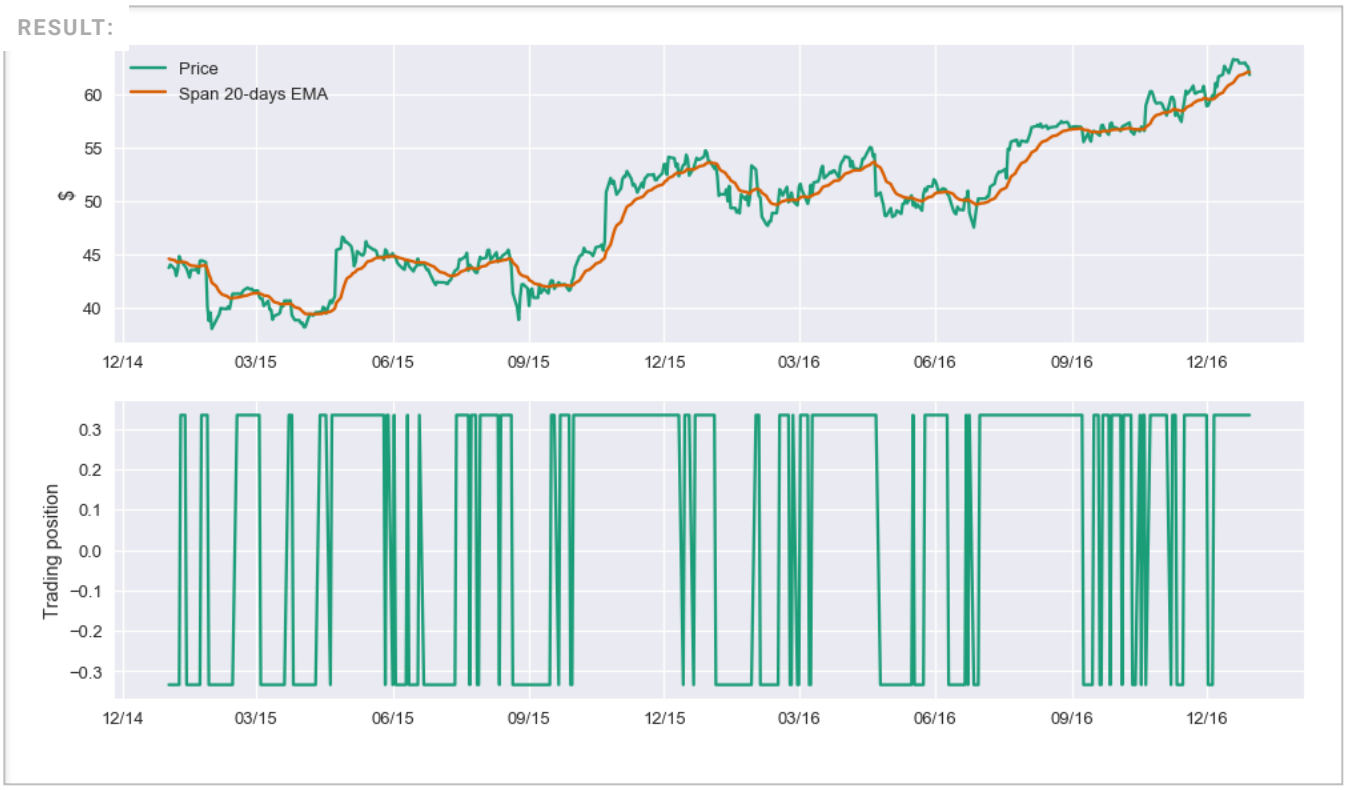
The best we can do is assume that we traded at the close of this day t_o. Therefore our position will be long starting on the following day, $t_o + 1$. This is easily corrected for by lagging our trading positions by one day, so that on day t_o our actual position is that of the previous day $t_o - 1$ and only on day $t_o + 1$ do we have a long position. Thus:

```
# Lagging our trading signals by one day.
trading_positions_final = trading_positions.shift(1)
```

Let us examine what the timeseries and the respective trading position look like for one of our assets, Microsoft.

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(16,9))

ax1.plot(data.loc[start_date:end_date, :].index,
ax1.plot(ema_short.loc[start_date:end_date, :].index,

ax1.set_ylabel('$')
ax1.legend(loc='best')
ax1.xaxis.set_major_formatter(my_year_month_fmt)

ax2.plot(trading_positions_final.loc[start_date:end_date, :].index,
         label='Trading position')

ax2.set_ylabel('Trading position')
ax2.xaxis.set_major_formatter(my_year_month_fmt)
```


RESULT:



Now that the position our strategy dictates each day has been calculated, the performance of this strategy can be easily estimated. To that end, we will need again the log-returns of the three assets $r_i\left(t\right)$. These are calculated as:

```
# Log returns - First the logarithm of the prices is taken and the
asset_log_returns = np
asset_log_returns.head
```


	AAPL		
2000-01-03	NaN	NaN	NaN
2000-01-04	-0.088078	-0.034364	-0.039099
2000-01-05	0.014528	0.010489	0.001920
2000-01-06	-0.090514	-0.034072	0.000955
2000-01-07	0.046281	0.012984	0.026730

Note that our strategy trades each asset separately and is agnostic of what the behaviour of the other assets is. Whether we are going to be long or short (and how much) in MSFT is in no way affected by the other two assets. With this in mind, the daily log-returns of the strategy for each asset i, $r_i^s\left(t\right)$ are calculated as

$$r_i^s\left(t\right) = w_i\left(t\right) r_i\left(t\right)$$

where $w_i\left(t\right)$ is the strategy position on day t which has already been attained at the end of trading day $t-1$.

What does this mean?

Assume that $p\left(t\right)$ crosses above $e_i\left(t\right)$ sometime during the trading session on Monday, day $t-1$. We assume that at the close on Monday we buy enough units of asset i to spend $\frac{1}{3}$ of our total funds, that is $\$\frac{N}{3}$ and that the price we bought at is $p\left(t-1\right) = \$10$. Let us also assume that on Tuesday, day t, the price closes at $p\left(t\right) = \$10.5$. Then our log-return for asset i on Tuesday, is simply

$$\frac{1}{3} \times \log\left(\frac{\$10.5}{\$10}\right) \simeq \frac{0.049}{3}$$

The actual return $r^s_{\text{rel},i}(t)$ is

$$r^s_{\text{rel},i}(t) =$$

In terms of dollars, on Tuesc

To get all the strategy log-re
strategy positions with the asset log-returns.


```
strategy_asset_log_returns = trading_positions_final * asset_log_re
strategy_asset_log_returns.tail()
```

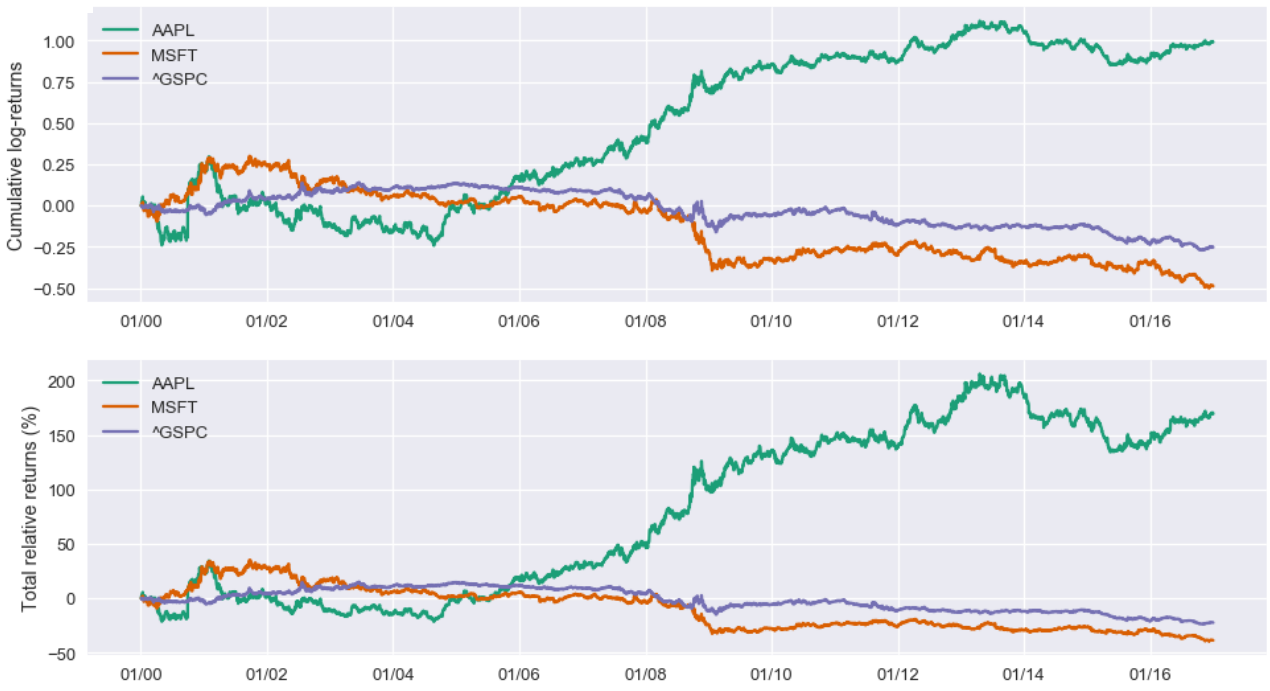
	AAPL	MSFT	^GSPC
2016-12-26	0.000000	0.000000	0.000000
2016-12-27	0.002110	0.000211	0.000749
2016-12-28	-0.001424	-0.001531	-0.002797
2016-12-29	-0.000086	-0.000477	-0.000098
2016-12-30	-0.002609	-0.004052	-0.001549

Remembering that the log-returns can be added to show performance across time, let us plot the cumulative log-returns and the cumulative total relative returns of our strategy for each of the assets.

```
# Get the cumulative log-returns per asset
cum_strategy_asset_log_returns = strategy_asset_log_returns.cumsum(

# Transform the cumulative log returns to relative returns
cum_strategy_asset_relative_returns = np.exp(cum_strategy_asset_log_

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(16,9))
```

```
for c in asset_log_re
    ax1.plot(cum_strat

ax1.set_ylabel('Cumula
ax1.legend(loc='best')
ax1.xaxis.set_major_formatter(my_year_month_fmt)

for c in asset_log_returns:
    ax2.plot(cum_strategy_asset_relative_returns.index, 100*cum_str

ax2.set_ylabel('Total relative returns (%)')
ax2.legend(loc='best')
ax2.xaxis.set_major_formatter(my_year_month_fmt)
```


RESULT:



WHAT IS THE TOTAL RETURN OF THE STRATEGY?

Strictly speaking, we can only add relative returns to calculate the strategy returns. Therefore

.

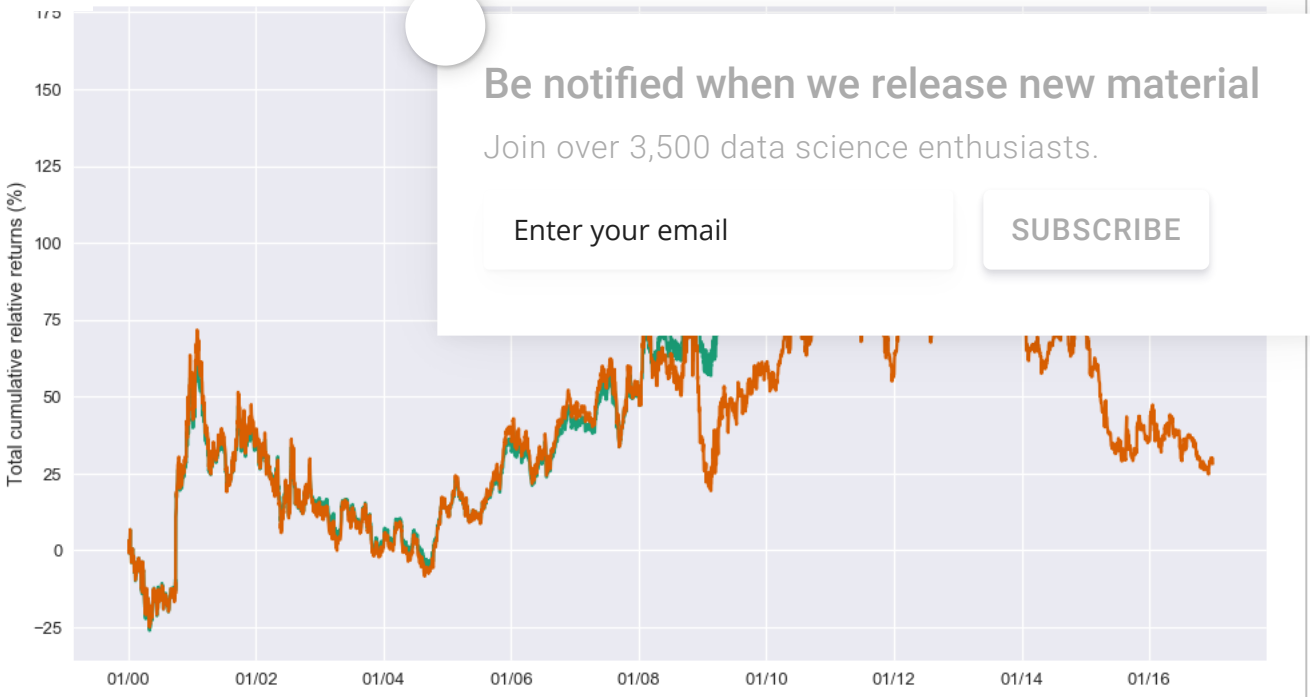
We saw in the previous article, however, that for small values of the relative returns, the following approximation holds

$$r_i\left(t\right) \simeq r_{\mathrm{rel},i}\left(t\right)$$

Thus, an alternative way is to simply add all the strategy log-returns first and then convert these to relative returns. Let us examine how good this approximation is.

```
# Total strategy relative returns. This is the exact calculation.
cum_relative_return_exact = cum_strategy_asset_relative_returns.sum

# Get the cumulative log-returns per asset
cum_strategy_log_return = cum_strategy_asset_log_returns.sum(axis=1

# Transform the cumulative log returns to relative returns. This is
cum_relative_return_approx = np.exp(cum_strategy_log_return) - 1

fig, ax = plt.subplots(figsize=(16,9))

ax.plot(cum_relative_return_exact.index, 100*cum_relative_return_exa
ax.plot(cum_relative_return_approx.index, 100*cum_relative_return_ap

ax.set_ylabel('Total cumulative relative returns (%)')
ax.legend(loc='best')
ax.xaxis.set_major_formatter(my_year_month_fmt)
```

RESULT:



As we can see, for relatively small time-intervals and as long the assumption that relative returns are small enough, the calculation of the total strategy returns using the log-return approximation can be satisfactory. However, when the small scale assumption breaks down, then the approximation is poor. Therefore what we need to remember the following:

1. Log-returns can and should be added across time for a single asset to calculate cumulative return timeseries across time.
2. However, when summing (or averaging) log-returns across assets, care should be taken. Relative returns can be added, but log-returns only if we can safely assume they are a good-enough approximation of the relative returns.

The overall, yearly, performance of our strategy can be calculated again as:

```
def print_portfolio_yearly_statistics(portfolio_cumulative_relative_

    total_days_in_simulation = portfolio_cumulative_relative_return
    number_of_years = total_days_in_simulation / days_per_year
```

```
    # The last data p  t will give us the total portfolio return
    total_portfolio_re
    # Average portfoli
    average_yearly_retu

    print('Total portf
    print('Average yearly return is: ' + '{:5.2f}'.format(100*averag

print_portfolio_yearly_statistics(cum_relative_return_exact)
```


Total portfolio return is: 108.24% Average yearly return is: 4.39%

WHAT NEXT?

One can observe that this strategy significantly underperforms the buy and hold strategy that was presented in the previous article. Let's compare them again:

```
# Define the weights matrix for the simple buy-and-hold strategy
simple_weights_matrix = pd.DataFrame(1/3, index = data.index, columr

# Get the buy-and-hold strategy log returns per asset
simple_strategy_asset_log_returns = simple_weights_matrix * asset_l

# Get the cumulative log-returns per asset
simple_cum_strategy_asset_log_returns = simple_strategy_asset_log_r

# Transform the cumulative log returns to relative returns
simple_cum_strategy_asset_relative_returns = np.exp(simple_cum_strat

# Total strategy relative returns. This is the exact calculation.
simple_cum_relative_return_exact = simple_cum_strategy_asset_relati
```

```
fig, ax = plt.subplots

ax.plot(cum_relative_r
ax.plot(simple_cum_rel

ax.set_ylabel('Total cumulative relative returns (%)')
ax.legend(loc='best')
ax.xaxis.set_major_formatter(my_year_month_fmt)

print_portfolio_yearly_statistics(simple_cum_relative_return_exact)
```


Total portfolio return is: 248.51% Average yearly return is: 7.59%

RESULT:

Which Strategy is Better?

This is not a simple question for one to answer at this point. When we need to choose between two or more strategies, we need to define a metric (or metrics)

based on which to compare them. This very important topic will be covered in the next article.

In addition, we observe in th
strategies is not constant ac
outperforms the other and o
that naturally arises is how c
backtesting performance in

Continue Learning

- **Python for Financial Analysis and Algorithmic Trading** (Udemy)
  Goes over numpy, pandas, matplotlib, Quantopian, ARIMA models, statsmodels, and important metrics, like the Sharpe ratio

Meet the Authors

Georgios Efstathopoulos | *QUANTITATIVE ANALYST*

PhD in Applied Mathematics and Statistics. Analyst working on quantitative trading, market and credit risk management and behavioral modelling at Barclays Investment Bank. Founder and CEO of QuAnalytics Limited.

BACK TO BLOG INDEX

R B • 4 months ago

Hello, I see you've made a different way of calculating the portfolio returns by taking the cumsum of element-wise multiplication between weights and aset returns. In part 2 you've made a matrix multiplication, took the diagonal and made the cumulative summation. Does it give the same result either way ?
Thank you for your posts they're greatly made!
Greetings,

Reply • Share ›

Roberto Lopez Maya • 5 months ago

Hello I don't understand why the short period is not set to 0.. As we sell the position, the return should be 0 during this period and not -0.33* return ....

Reply • Share ›

Rohit Khanna ➚ Roberto Lopez Maya • 4 months ago

In the moving average trading strategy, we are going short in the short period..hence -0.33 percent return

Reply • Share ›