Mean-Reversion strategy backtested

Bright Arafat Bello

2025-12-21

Table of contents

GitHub URL: https://github.com/bbarafat/Mean-Reversion

Introduction and overview

This project was undertaken to backtest a simple mean reversion strategy using historical close prices for various ETFs. For the purposes of this project, I will download 10 year adjusted historical close prices for the following;

1. QQQ -> An ETF that tracks the NASDAQ 100 index
2. SPY -> An ETF that tracks the S&P 500
3. TLT -> An ETF that tracks US treasury instruments

These datasets were downloaded using the yfinance library.[1]

```
tickers = ["SPY","QQQ","TLT"]
start = "2015-01-01"
end = "2025-01-01"
DATA_DIR = "../data"
```

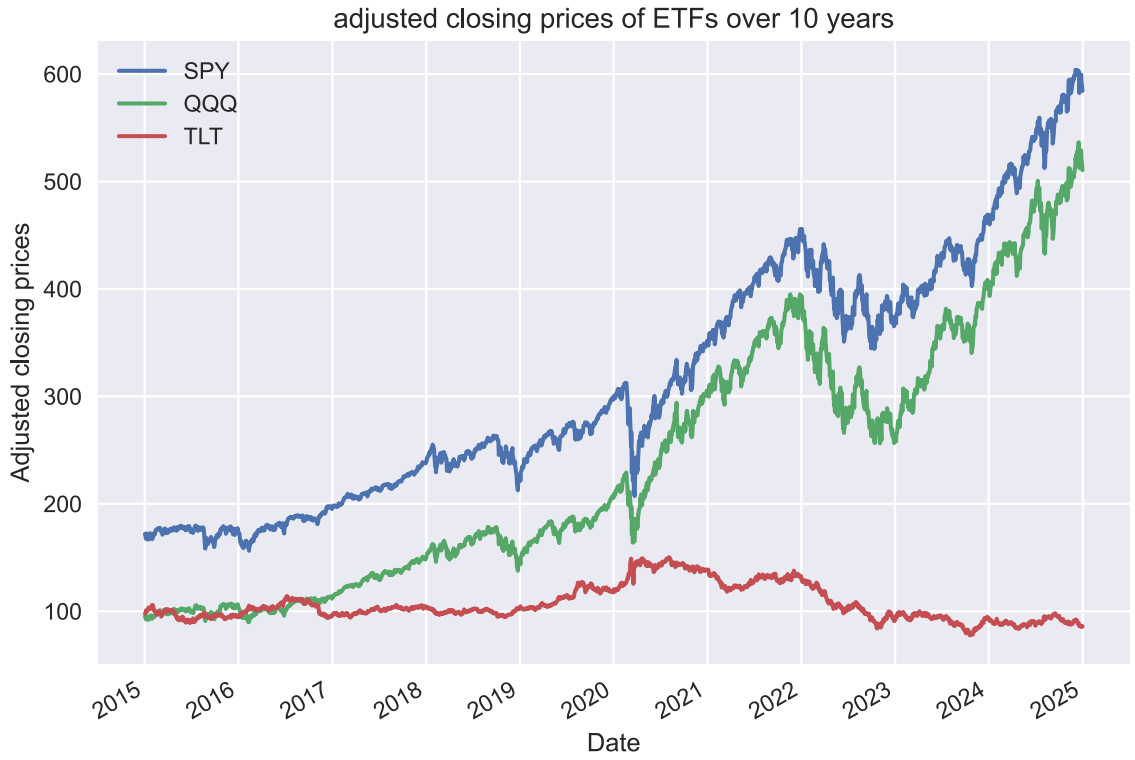
Visuals and summary statistics

Let us see how the first few lines of our downloaded data looks.

```
Loading cached prices...
```

Date	SPY	QQQ	TLT
2015-01-02	172.069	95.0215	97.6041
2015-01-05	168.957	93.6168	99.0879
2015-01-06	167.378	92.3698	100.9170
2015-01-07	169.435	93.5619	100.6960
2015-01-08	172.469	95.3706	99.3622

This would not be a worthy project without visualisations, and as visual learners, let us plot the closing prices to see the trend over time.



We observe an overall upward trend in closing prices over the 10 year period, except for TLT which was largely constrained. This makes a lot of sense considering TLT tracks US treasury securities which are usually low risk and therefore low reward, but safe. It is always important, at least for me, to include a 'safe' security in a portfolio. For QQQ and SPY however, the upward trend holds, with some periods of decline. We see certain fluctuations in the prices indicating

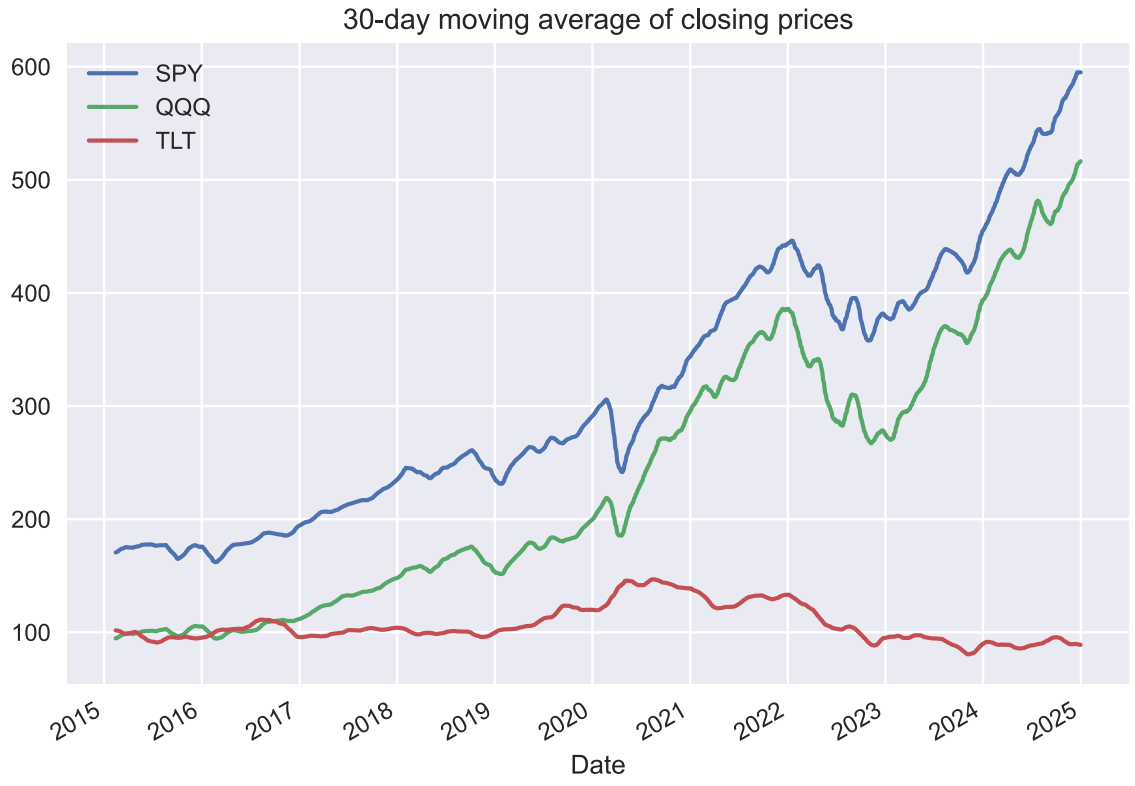
volatility in the stock, also possibly due to market events. Let us examine the general statistics of the prices to gain more insight.

	SPY	QQQ	TLT
count	2516.000000	2516.000000	2516.000000
mean	313.002506	237.620431	106.920562
std	115.489199	118.414797	16.472233
min	156.315000	89.811200	77.599000
25%	214.020750	133.576500	95.613175
50%	275.560000	195.998500	101.750500
75%	402.790750	328.705000	119.293750
max	603.956000	536.504000	150.242000

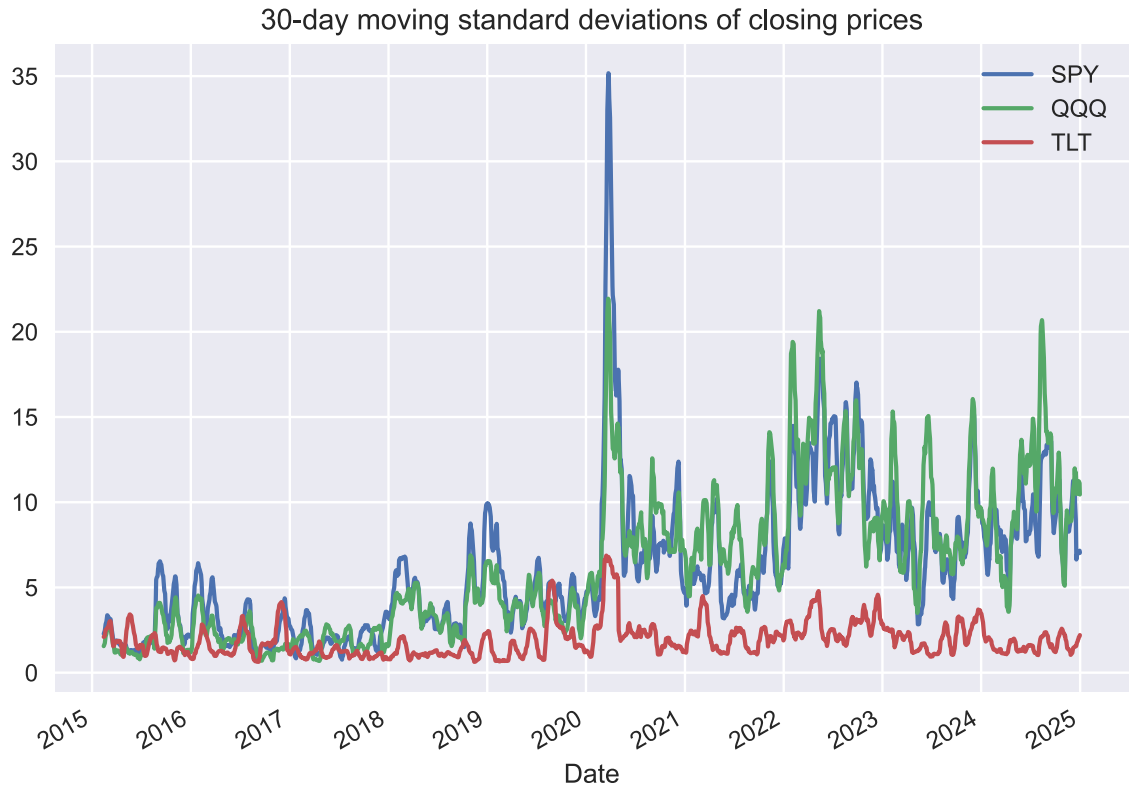
As expected we observe a high mean and standard deviation in the prices. This is characteristic of stocks that appreciate over time, but also indicates the risk associated with investing in them. Notice that the TLT ETF has the lowest deviation among the ETFs, behaving as we hoped it would. For the high deviation instruments, there is a high chance of return, but also a high chance of loss. Or as I like to call it, the no-free lunch situation.

Out of curiosity, let us visualise the rolling mean and rolling standard deviation of the prices over a 30 day window to see how they evolve over time.

```
Text(0.5, 1.0, '30-day moving average of closing prices')
```




```
Text(0.5, 1.0, '30-day moving standard deviations of closing prices')
```



The rolling mean shows the smooth upward trend we observed earlier, and a flatter curve for TLT as expected. The rolling standard deviation for the period was quite volatile, which is to be expected given the fluctuations we observed.

Returns Computation

Let us now compute our daily returns for the price series. We use simple returns as we are dealing with a portfolio of ETFs, and using log returns would compound our computations across instruments, which is not what we want to do, at all.

$$\text{Simple Returns} = \frac{\text{new price}}{\text{old price}} - 1$$

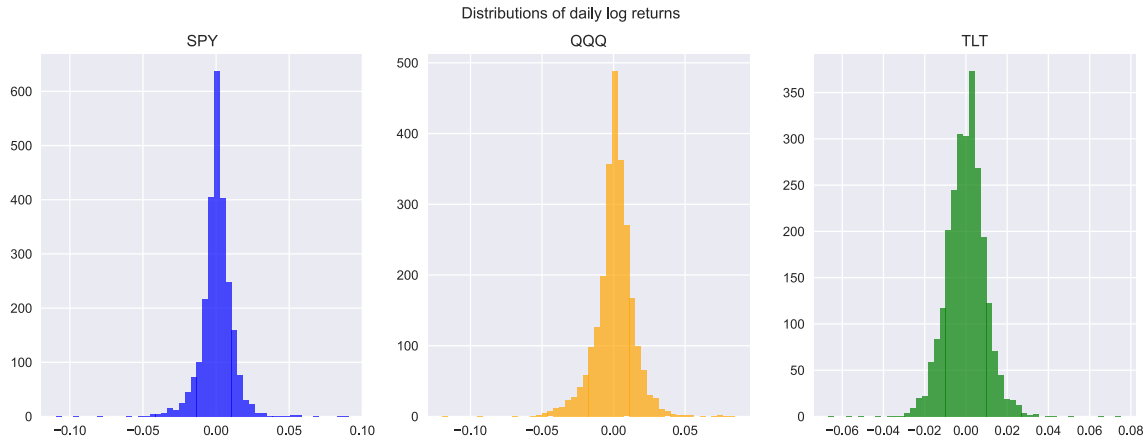
Simon Leung [2]

Lets obtain some summary statistics of our returns

	SPY	QQQ	TLT
count	2515.000000	2515.000000	2515.000000
mean	0.000548	0.000763	−0.000005
std	0.011096	0.013742	0.009640
min	−0.109464	−0.119754	−0.066695

	SPY	QQQ	TLT
25%	−0.003699	−0.005047	−0.005818
50%	0.000606	0.001185	0.000346
75%	0.005932	0.007838	0.005658
max	0.090602	0.084759	0.075234

We plot the returns for each ticker symbol. Visually, everything is as expected, normal with a nice little peak.



Defining The Mean Reversion Strategy (Bollinger Bands)

Mean Reversion : Financial instruments are periodically oversold/overbought, and when this occurs, they are sometimes expected to revert to their historical mean levels

Bollinger Bands: Consists of a simple moving average (SMA) (e.g. 10) and Upper and Lower Bands +- (1) Std Dev away from SMA.

```
# Let us create a simple moving
# average strategy to observe how it works
SMA = 10
dev = 1
```

```
#This strategy creates bollinger bands,
# two standard deviations away from the mean
sma = data.rolling(window=SMA).mean()
std = data.rolling(window=SMA).std()
upper = sma + (std * dev)
lower = sma - (std * dev)
```

We define our trading strategy as follows:

- When price crosses below the lower band, our model generates a buy signal, as we expect that our stock is oversold and will revert to the mean
- When price crosses above the upper band, our model generates a sell signal, as we expect that our stock is overbought and will revert to the mean

```
# Helper column. It indicates how far price is from the moving average.
# This is important because for certain periods,
# our price will be neither above nor below the bands,
# and we will need to know how to deal with those situations.
distance = data - sma

position = pd.DataFrame(np.nan, index = data.index, columns=data.columns)
position[data < lower] = 1 # 1. oversold -> go long

position[data > upper] = -1 # 2. overbought -> go short
```

```
# 3. crossing SMA ("Middle Band") -> go neutral
position[(distance * distance.shift(1)) < 0] = 0
```

```
position = position.ffill()
position = position.fillna(0) # where 1-3 isn´t applicable -> hold previous
position
```

Defining our portfolio weights

Buy and hold weights (Benchmark)

I assign equal weights for all active assets. This is to be our benchmark that we compare our strategy against.

```
n_assets = returns.shape[1]

bh_weights = np.repeat(1/n_assets, n_assets)
bh_returns = returns @ bh_weights
bh_equity = (1 + bh_returns).cumprod()
```

Strategy Weights

```
#This is to normalize weights across assets
# so our weights always sum to 1 regardless of which positions are active.
#This also prevents zero division and 'infinity'
# issues that could impact the computations downstream.
gross = position.abs().sum(axis=1)
gross = gross.replace(0, np.nan)
```

```
weights = position.shift(1).div(gross, axis=0).fillna(0)
weights = weights.loc[returns.index]
```

```
weight_diff = weights.diff().abs()

turnover = weight_diff.sum(axis=1)
```

Strategy returns computation

let us compute our strategy returns using our precalculated weights from the previous section.

```
strategy_returns = (returns.loc[weights.index] * weights).sum(axis=1)
strategy_equity = (1 + strategy_returns).cumprod()
```

To make our backtest more realistic, we shall include transaction costs of 0.1% per trade.

```
costs_per_turnover = 0.001 #10 basis points or 0.1% per trade

transaction_costs = turnover * costs_per_turnover
transaction_costs = transaction_costs.fillna(0)
```

```
strategy_returns_gross = strategy_returns.copy()

strategy_returns_net = strategy_returns_gross - transaction_costs
strategy_equity_net = (1 + strategy_returns_net).cumprod()
```

Now that the hard part is done, let us see some nice visuals to compare performance.

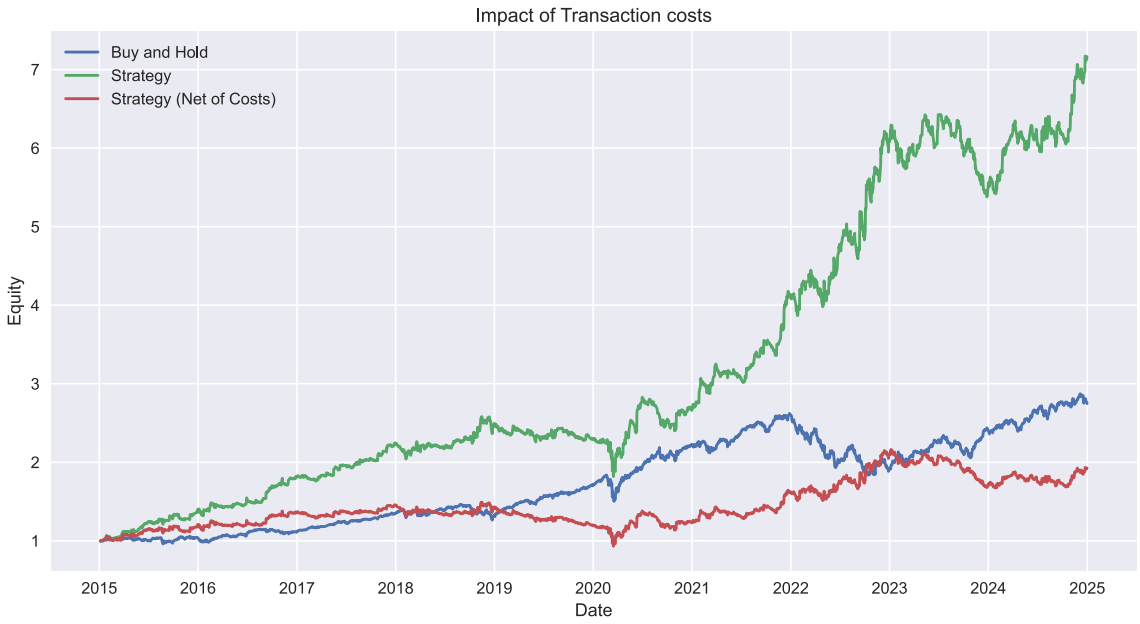


Figure 1: Impact of Transaction Costs

We see from Figure 1 that our strategy outperformed buy and hold. Our strategy after transaction costs however did not. This highlights the impact transaction costs can have on a trading strategy, regardless of how profitable it initially seems.

Optimization

Let us try to optimize our backtest by testing other window and standard deviation values to see if performance improves

```
def test_strategy(data, window_sizes, devs):
    result = {}
    for window in window_sizes:
        for dev in devs:
            tc = 0.001  # transaction cost
            returns = data.pct_change()
            sma = data.rolling(window=window).mean()
            std = data.rolling(window=window).std()
            lower = sma - std * dev
            upper = sma + std * dev

            position = pd.DataFrame(np.nan, index=data.index,
columns=data.columns)
            position[data < lower] = 1  # Go long, oversold
            position[data > upper] = -1  # Go short, overbought

            distance = data - sma
            position[(distance * distance.shift(1)) < 0] = 0
```

```
            position = position.ffill().fillna(0)

            divisor = position.abs().sum(axis=1).replace(0, np.nan)
            weights = position.shift(1).div(divisor, axis=0).fillna(0)

            turnover = weights.diff().abs().sum(axis=1)
            cost = turnover * tc


            strategy_returns = (returns * weights).sum(axis=1)
            strategy_returns = strategy_returns.dropna()

            strategy_equity = (1 + strategy_returns).cumprod()

            strategy_returns_net = strategy_returns -
cost.loc[strategy_returns.index]
            strategy_equity_net = (1 + strategy_returns_net).cumprod()

            ending_equity = strategy_equity_net.iloc[-1]
            ending_returns = strategy_returns_net.sum()

            result[(window, dev)] = {
                "ending_equity": ending_equity,
                "ending_returns": ending_returns
            }

    return result
```

I perform a search for window and deviation sizes. I define a search from 10 to 160 in intervals of 20 for the window sizes and standard deviations in the list [1,1.5,2,2.5,3].

The result is saved into a dataframe and sorted in descending order so we can observe the best performers immediately, there was one winner.

		ending_equity	ending_returns
10	1.000000	1.922422	0.819410
30	1.000000	1.668412	0.648138
	2.000000	1.596386	0.607474
	1.500000	1.520956	0.555534
50	1.000000	1.326227	0.411234
10	1.500000	1.158998	0.295181
150	3.000000	1.135735	0.195868

It appears that the optimal window size is 10 with a standard deviation of 1, this resulted in the highest ending equity.

If you've made it this this far, thank you for reading through my notebook!. You would have realised that the strategy implemented here is probably quite basic and can be improved in many ways.

One other thing to note is that when I was working my way through this exercise, I came all the way to the bottom, performed the optimisations and then replaced the initial window size and standard deviation with the optimal values I found.

The optimized value performed way better than my initial guess. The gross strategy significantly outperformed buy and hold, which is promising. The net version after transaction costs was less impressive, underperforming buy and hold slightly. This indicates that while the strategy has potential, transaction costs can significantly impact profitability. Further refinements and optimizations could enhance the strategy's effectiveness.

Bibliography

[1] Yahoo Finance, "Historical price data for SPY, TLT, and QQQ." 2025.

[2] Simon Leung, "A brief overview on simple returns and log returns in financial data." 2024.