Published in Towards Data Science

Eryk Lewinson Follow

Oct 6, 2019 · 8 min read · ✨ · Listen

Save

Source: unsplash

Learn how to generate comprehensive performance reports with one line of Python code!

This is the third part of a series of articles on backtesting trading strategies in Python. The previous ones described the following topics:

- introducing the `zipline` framework and presenting how to test basic strategies (link)
- importing custom data to use with `zipline` (link)

This time, the goal of the article is to show how to quickly and efficiently evaluate the performance of our strategies using a library called `pyfolio` (developed by Quantopian, the creators of `zipline`). `pyfolio` can be used as a standalone library and provide performance results based only on a provided series of returns. However, it works efficiently with `zipline` and I present this combination in this article.

Simple Moving Average Strategy

In this article, I only show a basic strategy, as the main focus is on evaluating the performance. For that purpose, I chose a strategy based on the simple moving average (SMA). The logic of the strategy can be summarized by the following:

- when the price crosses the 20-day SMA upwards — buy 20 shares
- when the price crosses the 20-day SMA downwards — sell all the shares
- we can only have a maximum of 20 shares at any given time
- there is no short-selling in the strategy

I run a backtest of the strategy using IBM's stock over the years 2016–2017. For details on how to create such a strategy and what do the particular components actually do, please refer to my previous article.

I start by loading the required libraries:


```
import zipline
import warnings
import pyfolio as pf
```

Then, I run the strategy:

```
%%zipline --start 2016-1-1 --end 2017-12-31 --capital-base 10000.0 -o simple_moving_average.pkl

# imports
from zipline.api import record, symbol, order_target
from zipline.finance import commission

# parameters
MA_PERIODS = 20
SELECTED_STOCK = 'IBM'
N_STOCKS_TO_BUY = 20

def initialize(context):
    context.time = 0
    context.asset = symbol(SELECTED_STOCK)
    context.set_commission(commission.PerShare(cost=0.0, min_trade_cost=0))
    context.has_position = False

def handle_data(context, data):
    context.time += 1
    if context.time < MA_PERIODS:
        return

    price_history = data.history(context.asset, fields="price", bar_count=MA_PERIODS, frequency="1
    ma = price_history.mean()

    # cross up
    if (price_history[-2] < ma) & (price_history[-1] > ma) & (not context.has_position):
        order_target(context.asset, N_STOCKS_TO_BUY)
        context.has_position = True
    # cross down
    elif (price_history[-2] > ma) & (price_history[-1] < ma) & (context.has_position):
        order_target(context.asset, 0)
```

ibm_sma_strategy.py hosted with ❤ by GitHub view raw

`zipline` automatically creates a performance DataFrame, which you can also see in the output of the code. For convenience, I stored the output in a pickle file called `simple_moving_average.pkl`. To make the analysis as smooth as possible, we can use a utility function provided by `pyfolio` and load the 3 most important elements of the performance DataFrame — the returns, positions, and transactions. To do so, I use `pf.utils.extract_rets_pos_txn_from_zipline`.

```
sma_results = pd.read_pickle('simple_moving_average.pkl')
returns, positions, transactions =
pf.utils.extract_rets_pos_txn_from_zipline(sma_results)
```

We can now inspect the extracted elements:

```
returns.head()
```

```
2016-01-04 00:00:00+00:00    0.0
2016-01-05 00:00:00+00:00    0.0
2016-01-06 00:00:00+00:00    0.0
2016-01-07 00:00:00+00:00    0.0
2016-01-08 00:00:00+00:00    0.0
Name: returns, dtype: float64
```

There could be no returns over the first 20 trading days of 2016, as it is still the warm-up period for the moving average.

index	sid	Equity(1441 [IBM])	cash
2016-02-05 00:00:00+00:00		2571.4	7427.3143
2016-02-08 00:00:00+00:00		2539.6	7427.3143
2016-02-09 00:00:00+00:00		2481.4	7427.3143
2016-02-17 00:00:00+00:00		2522.0	7306.6514
2016-02-18 00:00:00+00:00		2649.0	7306.6514

The `positions` DataFrame contains entries for each day when we do have a position in the considered assets and shows the capital split between equities and cash. We can see that by buying 20 IBM shares we still keep the majority of the capital in cash. For more advanced strategies, we can allocate a percentage of the capital to each order by using `order_percent`.

```
transactions.head()
```

	amount	commission	dt	order_id	price	sid	symbol	txn_dollars
2016-02-05 21:00:00+00:00	20	None	2016-02-05 21:00:00+00:00	0d3db2fe8645410b965b45fb858a2e9d	128.634285	Equity(1441 [IBM])	Equity(1441 [IBM])	-2572.6857
2016-02-10 21:00:00+00:00	-20	None	2016-02-10 21:00:00+00:00	58d24c56622d464db9204a6b78592812	120.129905	Equity(1441 [IBM])	Equity(1441 [IBM])	2402.5981
2016-02-17 21:00:00+00:00	20	None	2016-02-17 21:00:00+00:00	cd804a15646041919fed0c80145b244c	126.163050	Equity(1441 [IBM])	Equity(1441 [IBM])	-2523.2610
2016-04-20 20:00:00+00:00	-20	None	2016-04-20 20:00:00+00:00	5f2ed3c16afa4a5e811e0da818e971d4	146.036945	Equity(1441 [IBM])	Equity(1441 [IBM])	2920.7389
2016-04-28 20:00:00+00:00	20	None	2016-04-28 20:00:00+00:00	47f921d87df14010983c241d20fdae6b	147.143535	Equity(1441 [IBM])	Equity(1441 [IBM])	-2942.8707

Simple tear sheet

To evaluate the performance of strategies, portfolios or even single assets, we use `pyfolio` to create a tear sheet. A tear sheet is a concise document (often a single-paged one) that contains the most important information — such as financial metrics — regarding a public company. `pyfolio` provides much more functionalities than can be contained on a single sheet of paper, but for simplicity, we start with a simple tear sheet, which contains only the most vital information.

To create it, we run the following:

```
pf.create_simple_tear_sheet(returns)
```

What is truly remarkable about ` `pyfolio` ` is that it generates so much information with a single line of code!

Start date	2016-01-04
End date	2017-12-29
Total months	23
	Backtest
Annual return	3.4%
Cumulative returns	6.9%
Annual volatility	4.1%
Sharpe ratio	0.84
Calmar ratio	1.39
Stability	0.82
Max drawdown	-2.4%
Omega ratio	1.24
Sortino ratio	1.30
Skew	1.05
Kurtosis	24.58

The first table we see presents the dates of the test, how many months it lasted and a lot of financial metrics such as:

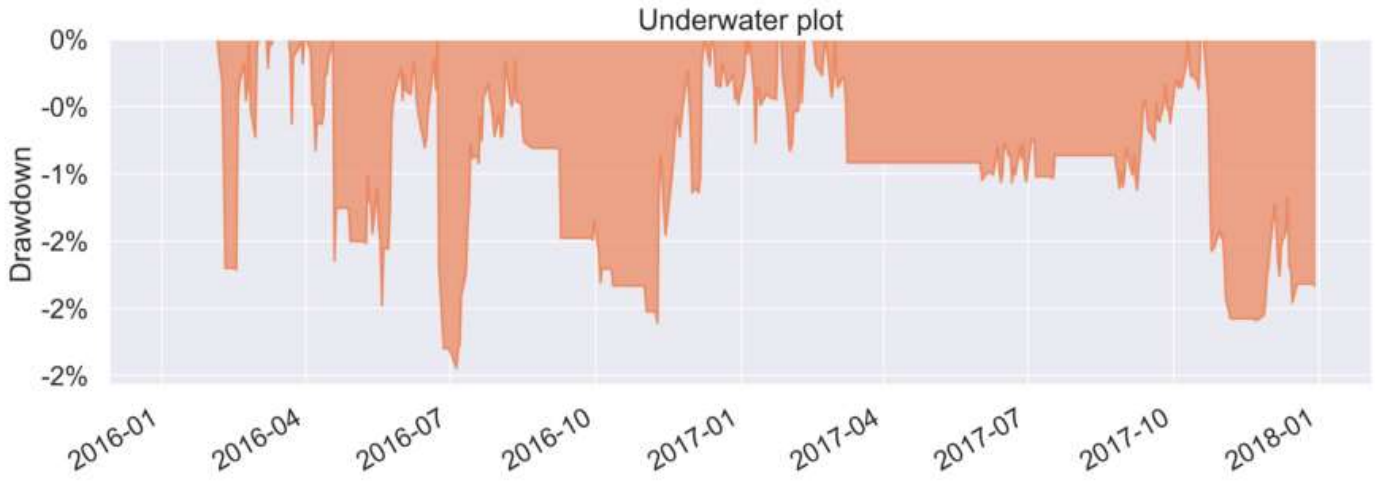
- annualized returns/standard deviation
- skewness — the third moment describes how skewed is the distribution
- kurtosis — the fourth moment indicates if there is more mass in the tails of the distribution
- Sharpe ratio — a very popular risk metric. It indicates the amount of excess return (over the risk-free rate) per unit of risk (measured by standard deviation).
- Sortino ratio — a modified version of the Sharpe ratio, where the standard deviation is replaced by downside deviation. Downside deviation measures only the negative volatility of the series, strictly speaking below a predefined level called minimum acceptable return.
- Omega ratio — another kind of risk-return performance metric. Its biggest advantage over the Sharpe ratio is that — by construction — it takes into account all statistical moments, while the Sharpe ratio only considers the first two.
- Maximum drawdown — indicates the largest (expressed in %) drop between a peak and a valley
- daily Value-at-Risk — another very popular risk metric. In this case, it indicates that in 95% of the cases, we will not lose more than 0.5% by keeping the position/portfolio for 1 more day.

The first 3 bullets are connected to the stylized facts of asset returns, which I described in one of the previous articles.


The next chart presents the cumulative returns on our strategy — we can observe the evolution of our portfolio's worth over the two years of running this strategy. The flat periods represent periods when we held no assets.

at the rolling one we see how volatile it was over time (calculated using rolling 6 months of data, not entire sample!).

The last chart — the underwater plot — shows the investment from a pessimistic point of view. By pessimistic, I mean that it focuses on losses. It depicts drawdowns and shows how long it took for the portfolio's value to recover to the previous peak, after suffering a loss. This chart makes it easy to distinguish between normal and extended periods of losses.



We created all these plots by passing only the `returns` object to the function. We can also pass the previously extracted positions and transactions, to automatically receive even more information on the simple tear sheet! Additionally, we can also specify a `live_start_date`, which splits the periods into the backtest and live-trading ones (`zipline` offers such a possibility). To do so, we simply run:

```
pf.create_simple_tear_sheet(returns, positions, transactions,
live_start_date='2017-01-01')
```

In the table below we immediately see the changes:

- there are two separate periods for analysis

The simple tear sheet also contains more plots related to positions and transactions, however, I do not show them for brevity.

Start date			2016-01-04
End date			2017-12-29
In-sample months			12
Out-of-sample months			11
	All	In-sample	Out-of-sample
Annual return	3.4%	3.7%	3.1%
Cumulative returns	6.9%	3.7%	3.1%
Annual volatility	4.1%	4.5%	3.6%
Sharpe ratio	0.84	0.83	0.86
Calmar ratio	1.39	1.52	1.47
Stability	0.82	0.37	0.52
Max drawdown	-2.4%	-2.4%	-2.1%
Omega ratio	1.24	1.21	1.28
Sortino ratio	1.30	1.16	1.72
Skew	1.05	-1.02	5.10
Kurtosis	24.58	11.08	53.87
Tail ratio	1.35	1.34	1.06
Daily value at risk	-0.5%	-0.6%	-0.4%

Full Tear Sheet

In the previous part, we only created a simple tear sheet, which gave a concise overview of the strategy's performance. It is possible to easily obtain much more information, by creating a full tear sheet (it's level of detail also depends on the provided information — here we only use returns).

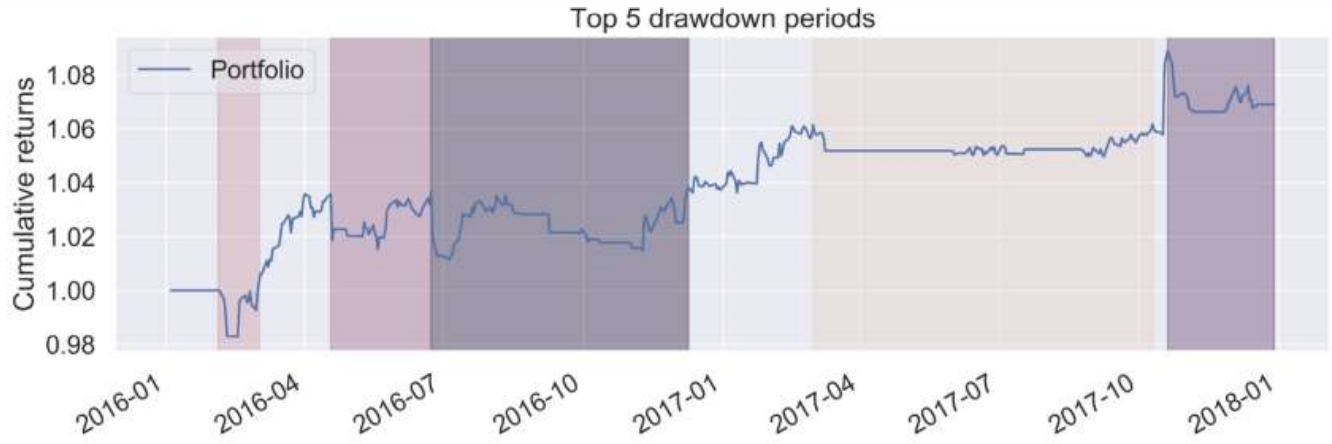
```
pf.create_full_tear_sheet(returns)
```

I present a selection of new tables/charts included in the full tear sheet:

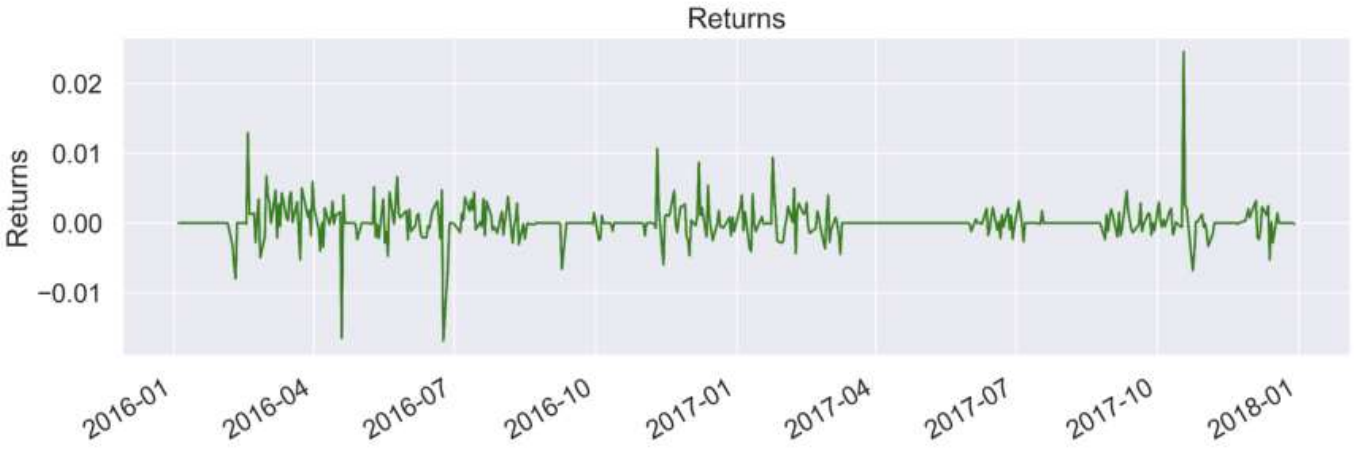
- A table presenting top 5 worst drawdowns, together with information such as peak/valley date and the duration

Worst drawdown periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	2.44	2016-06-23	2016-07-05	2016-12-09	122
1	2.09	2017-10-20	2017-11-22	NaT	NaN
2	1.98	2016-04-18	2016-05-19	2016-06-23	49
3	1.71	2016-02-04	2016-02-17	2016-03-02	20
4	1.12	2017-03-01	2017-09-08	2017-10-10	160

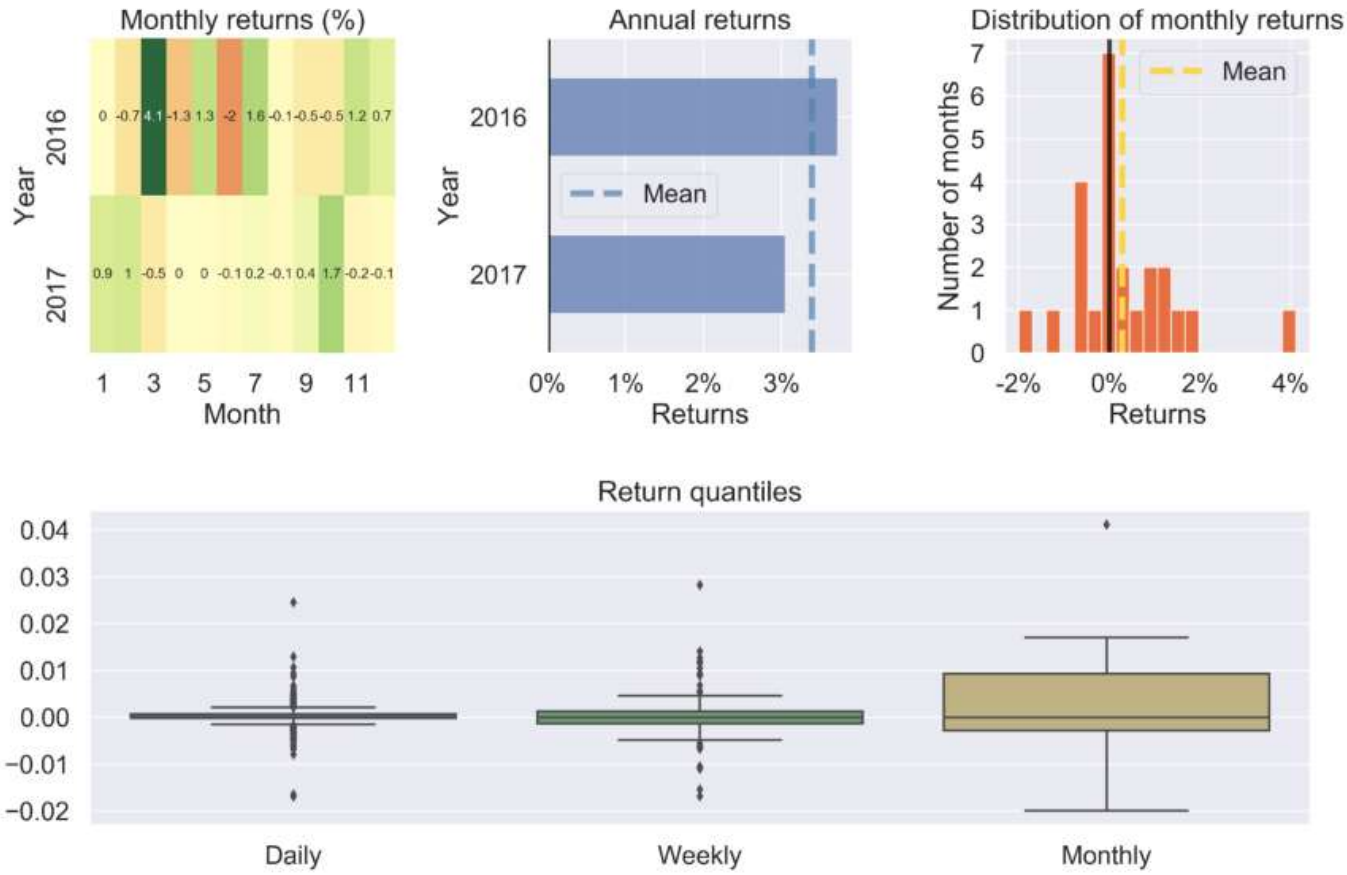
- Top 5 drawdown periods visualized on top of the cumulative returns. In our case, the drawdown periods cover almost the entire investment horizon. One of the reasons is that for extended periods of time we had no open positions, so the cumulative returns did not change, thus extending the recovery time.



- A simple plot showing the daily returns over time. The flat periods reflect the times when we had no position in the underlying asset.



- A selection of plots summarizing the strategy's returns by breaking them down into: monthly returns (separately per each year), annual returns per each year, a histogram presenting the distribution of monthly returns and finally quantiles of returns expressed in different frequencies.



Conclusions

In this article, I showed how to use `pyfolio` to easily obtain a plethora of financial metrics and plots describing the performance of an asset/strategy/portfolio. This post gives a high-level introduction, while there are still many aspects to cover.

Some of the other functionalities of `pyfolio` include:

- plots can be accessed manually without creating a simple/full tear sheet — for example, we can create a "top 5 worst drawdowns plot" by running

  ```
  pf.plot_drawdown_periods(returns, top=5)
  ```

- we can account for slippage while creating the tear sheets
- create Bayesian tear sheets (based on PyMC3, require separate dependencies installed)

interested in calculating one metric, such as the Omega ratio or the Sortino ratio. If that is the case, I certainly recommend `empyrical` for the task.

Kudos to Quantopian for creating such comprehensive libraries as `zipline`, `pyfolio` and `empyrical`!

As always, any constructive feedback is welcome. You can reach out to me on Twitter or in the comments. You can find the code used for this article on my GitHub.

Below you can find the other articles in the series:

- building algorithmic trading strategies based on Technical Analysis (link)
- building algorithmic trading strategies based on the mean-variance analysis (link)

I recently published a book on using Python for solving practical tasks in the financial domain. If you are interested, I posted an article introducing the contents of the book. You can get the book on Amazon or Packt's website.

References

- https://github.com/quantopian
- https://www.youtube.com/watch?v=BCLgXjxYONg