Lorenzo Ampil Follow
Jul 30, 2020 · 9 min read · Listen

Backtest Your Trading Strategy with Only 3 Lines of Python

Introducing fastquant, a simple backtesting framework for data driven investors

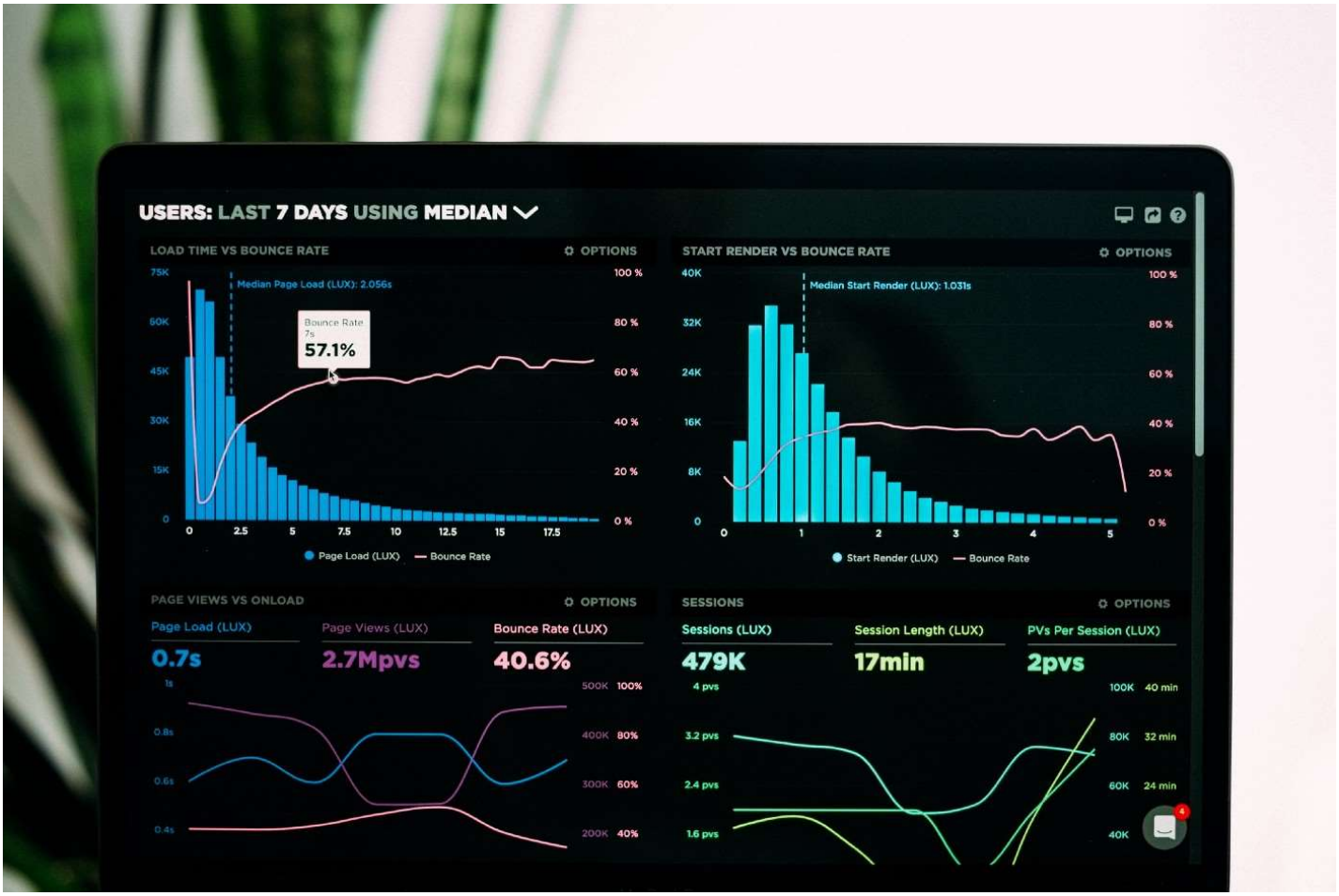


Photo by Luke Chesser on Unsplash

Ever since I started investing back in college, I was exposed to the different ways of analyzing stocks — *technical analysis* and *fundamental analysis*. I've even read books and countless articles about these techniques.

In a nutshell, *technical analysis* argues that you can identify the right time to buy and sell a stock using *technical* indicators that are based on the stock's historical price and volume movements. On the other hand, fundamental analysis argues that you can measure the actual intrinsic value of a stock based on the *fundamental* information found in a company's financial statements.

practice, most trades still end up as "gut feel" decisions that are not driven by data.

So how can we possibly assess these strategies? We can do this by comparing the expected return on investment (ROI) that we can get from each approach. The best way to do this, is with a method called *backtesting* — where a strategy is assessed by simulating how it would have performed had you used it in the past.

Now, there are already quite a few backtesting frameworks out there, but most of them require advanced knowledge of coding. It's typical for a simple hello world implementation to require as much as ~30 lines of code.

To fill this gap, I decided to create *fastquant*, with the goal of bringing backtesting to the mainstream by making it as simple as possible. **With *fastquant*, we can backtest trading strategies with as few as 3 lines of code!**

fastquant is essentially a wrapper for the popular backtrader framework that allows us to significantly simplify the process of backtesting from requiring at least 30 lines of code on backtrader, to as few as 3 lines of code on fastquant.

For the rest of this article, I will walk you through how to backtest a simple moving average crossover (SMAC) strategy through the historical data of Jollibee Food Corp. (JFC).

Also, if you're interested in performing this type of analysis without even having to code, do try out Hawksight, this new tool that I recently built to make advanced investment analysis more accessible — even for non coders!

Let's get started!

Backtest our first strategy

Install fastquant

It's as simple as using pip install!

```
# Run this on your terminal
pip install fastquant

# Alternatively, you can run this from jupyter this way
!pip install fastquant
```

Get stock data

Import the get_stock_data function from fastquant and use it to pull the stock data of Jollibee Food Corp. (JFC) from January 1, 2018 to January 1, 2019. Notice that we have columns corresponding to the date (dt), and closing price (close).

```
from fastquant import get_stock_data
jfc = get_stock_data("JFC", "2018-01-01", "2019-01-01")
print(jfc.head())

#           dt  close
#   2019-01-01  293.0
#   2019-01-02  292.0
#   2019-01-03  309.0
#   2019-01-06  323.0
#   2019-01-07  321.0
```

Backtest your trading strategy

Backtest a simple moving average crossover (SMAC) strategy through the historical stock data of Jollibee Food Corp. (JFC) using the backtest function of fastquant.

go above, this is considered a "buy" signal, while if it crosses over from above to go below, this is considered a "sell" signal. For more information on how this works, please check out the explanation in one of my previous articles.

To start out, let's initialize the fast_period and slow_period as 15, and 40, respectively.

You should see the final portfolio value below at the bottom of the logs. This value can be interpreted as how much money your portfolio would have been worth at the end of the backtesting period (in this case January 1, 2019). If you get the difference between your "Final Portfolio Value" and your "Starting Portfolio Value", this will be your expected earnings for that same period based on your backtest (in this case PHP 411.83).

```
from fastquant import backtest
backtest('smac', jfc, fast_period=15, slow_period=40)

# Starting Portfolio Value: 100000.00
# Final Portfolio Value: 100411.83
```

Bringing it all together — backtesting in 3 lines of Python

The code below shows how we can perform all the steps above in just 3 lines of python:

```
from fastquant import backtest, get_stock_data
jfc = get_stock_data("JFC", "2018-01-01", "2019-01-01")
backtest('smac', jfc, fast_period=15, slow_period=40)

# Starting Portfolio Value: 100000.00
# Final Portfolio Value: 100411.83
```

Simple Moving Average Crossover (15 day MA vs 40 day MA)

Improve our SMAC strategy

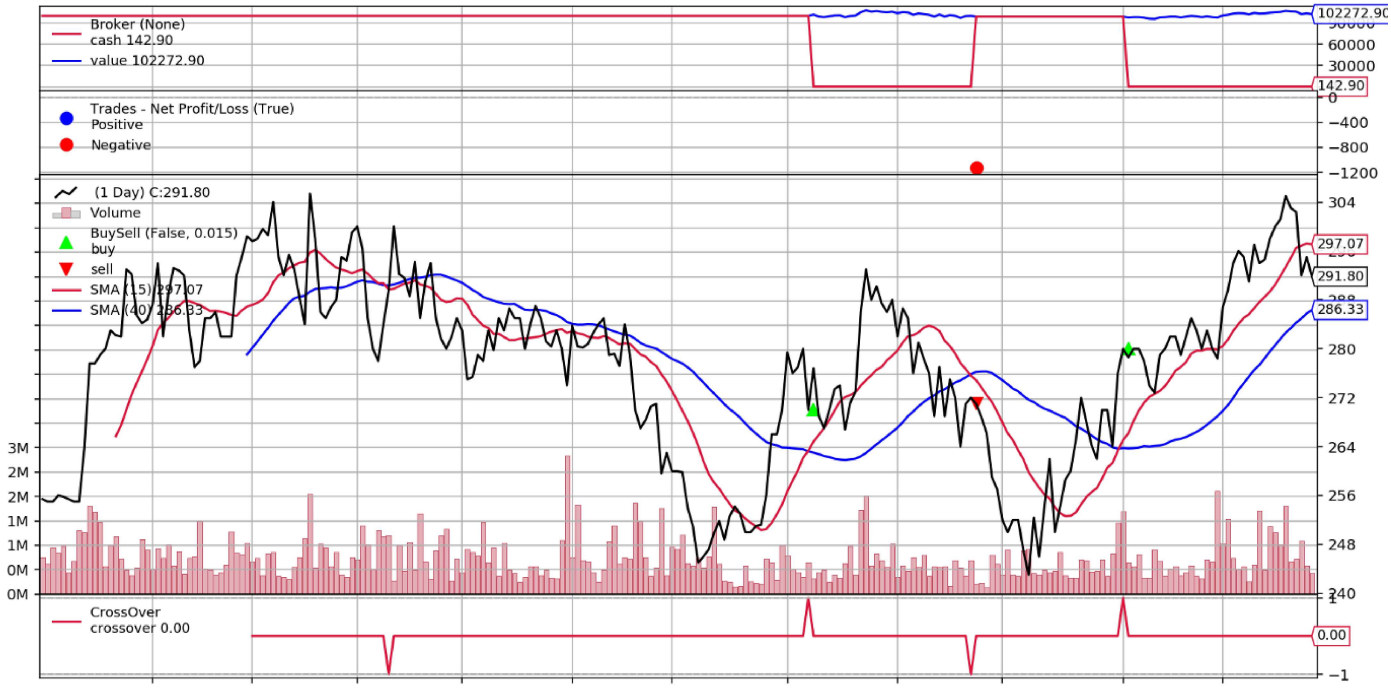
Increase both the fast period and the slow period

```
backtest('smac', jfc, fast_period=30, slow_period=50)

# Starting Portfolio Value: 100000.00
# Final Portfolio Value: 83946.83
```

Decrease the slow period while keeping the fast period the same

In this case, the performance of our strategy actually improved! Our final portfolio value went up from PHP 100,412 to PHP 102,273 (PHP 1,861 increase), after decreasing the slow period to 35, and keeping the fast period the same at 15.

```
backtest('smac', jfc, fast_period=15, slow_period=35)

# Starting Portfolio Value: 100000.00
# Final Portfolio Value: 102272.90
```



Simple Moving Average Crossover (15 day MA vs 35 day MA)

The table below compares the performance of our 3 SMAC strategies:

	Fast Period	Slow Period	Final Portfolio Value
Strategy 1	15 days	40 days	100411.83
Strategy 2	30 days	50 days	83946.83
Strategy 3	15 days	35 days	102272.9

Strategy 3 (highlighted in yellow) was the most accurate SMAC strategy based on our backtest!

Backtesting has quite a few limitations and overcoming them will often require additional steps to increase our confidence in the reliability of our backtest's results & recommendations.

Below are two of backtesting's limitations followed by safeguards to overcome them:

Overfitting

This refers to the situation where the "optimal parameters" that you derived were fit too much to the patterns of a previous time period. This means that the expected profitability of your strategy will not translate to actual profitability in the future when you decide to use it.

One safeguard for this would be to test your strategies out-of-sample, which is similar to using a "test set" in machine learning. The idea is that you hold out some data, that you only use once later when you want to assess the profitability of your trading strategy. This way, it's harder to overfit your parameters since you're not optimizing your strategy based on that dataset.

Look ahead bias

This is the bias that results from utilizing information during your backtest that would not have been available during the time period being tested. For example, you could be testing the effectiveness of a strategy on JFC that assumes that you would have known about its financial performance (e.g. net income) a month before it was actually made available publicly. This would give you unreliable confidence in your strategy that could lose you a lot of money later.

In this case, one of the best things you can do to avoid this bias is to thoroughly validate the assumptions that you make when you're backtesting your strategy. It pays to rigorously assess your strategy, and the information that has to be available for the strategy to be properly executed.

These are only 2 of the many limitations that come with backtesting. I do plan to write an article that discusses these in more detail in the future so stay tuned!

After addressing the above limitations, we should be more confident in our chosen strategy; however, do remember that while we can be more confident with our strategy, its performance in the unseen *real world* will never be 100% for sure.

I recommend that once you adopt a strategy in the real world, start off with a relatively small amount of money and only increase it as the strategy shows more consistent success; otherwise, be ready to kill it in the case that it's proven to work poorly in the real world.

Take note of fastquant's default parameters

"get_stock_data" function

To make the "get_stock_data" function as simple as possible to use, we've designed it to only return the closing price of the stock (used for most trading strategies), which follows the format "c" (c = closing price). But, if you want to have more pricing data points (e.g. OHLCV for "open", "high", "low", "close", "volume"), just set the "format" argument in "get_stock_data" to your desired data format.

Example below for the format (OHLCV) for Tesla stock:

```
tsla = get_stock_data("TSLA", "2018-01-01", "2019-01-01", format="ohlcv")
```

Note: This format feature should be stable for international stocks listed on Yahoo finance. For symbols from PSE, we recommend sticking to the default "c" format.

"backtest" function

edit these defaults by setting the values in the arguments in parentheses.

Example below where I backtest Tesla assuming buy_prop = 50%, sell_prop = 50% and commission_per_transaction = 1%.

```
backtest("smac", tsla, buy_prop=0.50, sell_prop=0.50, commission=0.01)
```

Contribute more strategies to fastquant

Remember that *fastquant* has as many strategies as are present in its existing library of strategies. There are 8 strategy types to choose from so far — including the Simple Moving Average Crossover (SMAC), Relative Strength Index (RSI), and even a sentiment analysis based strategy!

As I've mentioned in the introduction of this article, there are a large number of different strategies that can be applied for trading. With this, the fastquant dev team, and I could really use some help adding more of these strategies into *fastquant.* If you're interested in contributing, please do check out the *strategies module* in the fastquant package.

If you're not familiar with the finance concepts or the low level backtesting framework being used, don't worry! We have a strong community of contributors that can help out once you send your first PR. Just follow these docs on contributing and you should be well on your way!

If you want to interact with us directly, you can also reach us on the Hawksight discord. Feel free to ask about fastquant in the #feedback-suggestions and #bug-report channels.

For this next article in this fastquant series, I'll be discussing about how to apply grid search to automatically optimize your trading strategies, over hundreds of parameter combinations!

Thanks for reading this article, and please feel free to comment below or contact me via email (*lorenzo.ampil@gmail.com),* twitter, or linkedin if you have any further questions about *fastquant* or anything related to applying data science for finance!

Want to do this without coding at all?

If you want to make this kind of analysis even more simple without having to code at all (or want to avoid the pain of doing all of the setup required), you can try out Hawksight — this new no-code tool that I recently built to democratize data driven investments.

We just launched launched open beta so anyone can signup, and use the tool for free. Hoping to make these kinds of powerful analyses accessible to more people!

Note from Towards Data Science's editors: *While we allow independent authors to publish articles in accordance with our rules and guidelines, we do not endorse each author's contribution. You should not rely on an author's works without seeking professional advice. See our Reader Terms for details.*