Backtesting Trading Strategies with Ease:

An excursion with FireDucks

Introduction

Traders and those who analyze numbers to predict market trends are always looking for better ways to go through past data and test out their trading ideas. That's where FireDucks comes in—it's a fresh and innovative Python library made for quick and smart data handling and analysis. We're about to take a deep dive into the ways FireDucks is changing the game for testing trading methods, especially the Simple Moving Average (SMA) crossover strategy. It's a popular strategy that's known for being straightforward and really effective, and its key for a lot of traders out there.

In addition to that, we're going to check out how FireDucks does when it comes to speed and how much computer memory it uses, compared to Pandas. For anyone who relies on being able to do things quickly and efficiently, seeing how FireDucks stacks up in these areas will be really helpful.

Discovering the Strength of Simple Moving Averages

The Simple Moving Average (SMA) is a key tool that lots of traders use to figure out what's happening in the market. It works by smoothing out price fluctuations over a certain amount of time, giving a clearer picture of the direction in which prices are moving. Usually, traders use two different SMAs together to get a better read on potential trends—a short-term one and a long-term one. For example, pairing a 50-day SMA with a 200-day SMA is a common strategy that many investors trust. This duo helps them spot important patterns that could suggest whether it's a good time to buy or sell.

The Golden Cross: A Trader's Signal for Potential Profit

When it comes to trading, one of the moments that gets traders excited is called the Golden Cross. This happens when the short-term SMA rises and cuts through the long-term SMA from below. It's like a green light flashing, telling traders that the market could be getting stronger, and prices might start climbing. This signal gets a lot of traders thinking about buying because it could mean there's a chance for prices to go up more. For those looking to make a move on rising trends, the Golden Cross is a key hint that it might be time to jump in and take advantage of the potential for profits.

The Death Cross: A Cautionary Tale for Traders

On the flip side of the Golden Cross is what's known in trading as the Death Cross. This is the signal that tends to raise a red flag for traders. It happens when the short-term SMA drops and goes below the long-term SMA. Think of it as a warning bell, hinting that the market could be weakening and that prices may start to fall. Traders often see the Death Cross as a cue to consider selling or to bet on falling prices with short positions. It’s like a heads-up that there might be rougher market seas ahead, and it's time to think about protecting against potential losses.

Benchmarking Environment

- Evaluation environment details: Ubuntu 18.04.6 LTS (GNU/Linux 5.4.0-150-generic x86_64)
- System Memory: 64GB
- CPU: Count-16, Model-Intel(R) Xeon(R) Gold 6252 CPU @ 2.10GHz
- Pandas Version: 1.5.3
- FireDucks Version: 0.10.3

Calculating trends: - 20-day and 50-day moving averages

For this we have used World Stock Prices (Daily Updating) from:

https://www.kaggle.com/datasets/nelgiriyewithana/world-stock-prices-daily-updating/data

This dataset offers a comprehensive historical record of stock prices for the world's most famous brands, with daily updates. The data that we are using spans from January 1, 2000, till 2nd quarter of FY'23-24 , providing an extensive timeline of stock market information for various global brands.

Note: All the below used code is run with FireDucks as follows: `python -mfireducks.imhook backtest_sma_crossover.py`

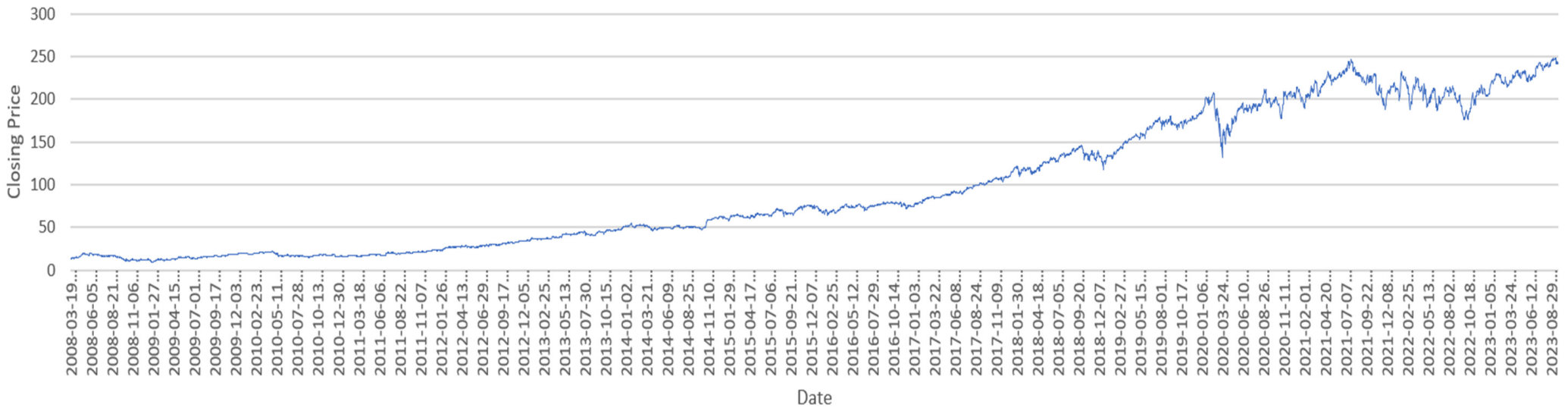
```
data = pd.read_csv('World-Stock-Prices-Dataset.csv')
print (data)
```

```
                       Date        Open        High         Low  ... Brand_Name Ticker Industry_Tag Country
2023-09-20 00:00:00-04:00  243.630005  244.529999  241.669998  ...       visa      V      finance     usa
2023-09-19 00:00:00-04:00  245.000000  245.119995  241.460007  ...       visa      V      finance     usa
2023-09-18 00:00:00-04:00  240.940002  245.229996  240.770004  ...       visa      V      finance     usa
2023-09-15 00:00:00-04:00  241.899994  244.100006  240.029999  ...       visa      V      finance     usa
2023-09-15 00:00:00-04:00  241.899994  244.100006  240.029999  ...       visa      V      finance     usa
```

We would be analysing the SMA of VISA for our analysis:

data[data['Brand_Name'] == 'visa'].sort_values(by='Date', ascending=True)

Below is a plot for the general variations in closing price for VISA:



Create new columns in our dataframe for both the long(i.e. 50 days) and short (i.e. 20 days) simple moving averages (SMAs):

```
1    short_window = 20
3    long_window = 50
4    data['Short_MA'] = data['Close'].rolling(window=short_window).mean()
5    data['Long_MA'] = data['Close'].rolling(window=long_window).mean()
```

	Date	Brand_Name	Short_MA	Long_MA
2008-03-19	00:00:00-04:00	visa	12.679034	12.679034
2008-03-20	00:00:00-04:00	visa	13.559833	13.559833
2008-03-24	00:00:00-04:00	visa	13.507844	13.507844
2008-03-25	00:00:00-04:00	visa	13.679329	13.679329
2008-03-26	00:00:00-04:00	visa	13.814085	13.814085
	...	...	...	...
2023-09-15	00:00:00-04:00	visa	241.443670	220.646944
2023-09-15	00:00:00-04:00	visa	241.510811	220.849078
2023-09-18	00:00:00-04:00	visa	241.602040	221.085610
2023-09-19	00:00:00-04:00	visa	241.638132	221.322623
2023-09-20	00:00:00-04:00	visa	241.618266	221.560098

Generating signal from crossovers

In our existing FireDucks dataframe, create a new column ‘Signal’ such that if 20-day SMA is greater than 50-day SMA then set Signal value as 1 else when 50-day SMA is greater than 20-day SMA then let the default as 0.

```
data['Signal'] = 0
data.loc[(data['Short_MA'] > data['Long_MA']), 'Signal'] = 1
```

From these ‘Signal’ values, buy or sell signals can be generated to represent trading signals. Crossover happens when the faster moving average and the slower moving average cross, or in other words the ‘Signal’ changes from 0 to 1 (or 1 to 0). So, to incorporate this information, create a new column ‘Crossover’ which nothing but a day-to-day difference of the ‘Signal’ column.

```
data['Crossover'] = data['Signal'].diff()
```

Above chart shows a snapshot for year 2022, buy and sell points are marked with green and red arrows respectively.

As can be seen from the plot, the orange line represents the faster moving average (20 day SMA), the grey line represents the slower moving average(50 day SMA) and the blue line represents the actual closing price. If you carefully observe, these moving averages are nothing but the smoothed versions of the actual price but lagging by certain period of time. The short-term moving average closely resembles the actual price which perfectly makes sense as it takes into consideration more recent prices. In contrast, the long-term moving average has comparatively more lag and loosely resembles the actual price curve.

A signal to buy (as represented by green up-arrow) is triggered when the fast moving average crosses above the slow moving average. This shows a shift in trend i.e., the average price over last 20 days has risen above the average price of past 50 days. Likewise, a signal to sell (as represented by red down-arrow) is triggered when the fast moving average crosses below the slow moving average indicating that the average price in last 20 days has fallen below the average price of the last 50 days.

Complete python code:

```
import pandas as pd

# (1) load_data
data = pd.read_csv('World-Stock-Prices-Dataset.csv')
data = data[data['Brand_Name'] == 'visa'].sort_values(by='Date',
ascending=True)

# (2) rolling_mean
short_window = 20
long_window = 50
#Calculate short and long moving average
data['Short_MA'] = data['Close'].rolling(window=short_window,
min_periods=1).mean()
data['Long_MA'] = data['Close'].rolling(window=long_window,
min_periods=1).mean()

# (3) add_signal: Add a signal column and set values
data['Signal'] = 0
data.loc[(data['Short_MA'] > data['Long_MA']), 'Signal'] = 1

# (4) add_crossover: Add a Crossover column and set values
data['Crossover'] = data['Signal'].diff()
```

Few take aways from above:

- As can be seen the code used above with FireDucks is completely compatible with Pandas.
- Running time comparison with Pandas for above code:

	Pandas exec time (sec)	FireDucks exec time (sec)	FireDucks speed-up over pandas
load_data	0.4688	0.0647	7.25x
rolling_mean	0.0018	0.0175	0.10x
add_signal	0.0008	0.0033	0.25x
add_crossover	0.0005	0.0029	0.18x
Total	**0.4719**	**0.0885**	**5.33x**

- Memory consumption comparison with Pandas for above code:

Step	FireDucks	Pandas
Data load memory usage (MB)	0.126131	78.49759
Moving average memory usage (MB)	0.600346	0.170537
Set Signal memory usage (MB)	0.025246	0.088572
Set Crossover memory usage (MB)	0.013923	0.070127
Total (MB)	**0.765646**	**78.82682**

Note:

- Python time module has been used for time measurement.
- Python tracemalloc module has been used for memory usage calculation.
- The program, backtest_sma_crossover_profile.py is executed as follows for pandas:
  $ python backtest_sma_crossover_profile.py
- The program, backtest_sma_crossover_profile.py is executed as follows for FireDucks: (just added the option -mfireducks.imhook)
  $ FIREDUCKS_FLAGS="--benchmark-mode" python -mfireducks.imhook backtest_sma_crossover_profile.py

It might be found that FireDucks is not performing very well for the SMA analysis in the above evaluation. The root cause behind this lies in the size of the input data. There were only 3906 samples for the category “visa” in the input data. Pandas itself might be sufficient to analyze such a small scale of data. In order to demonstrate the true power of FireDucks, we have experimented with another piece of relatively larger data from the following source:

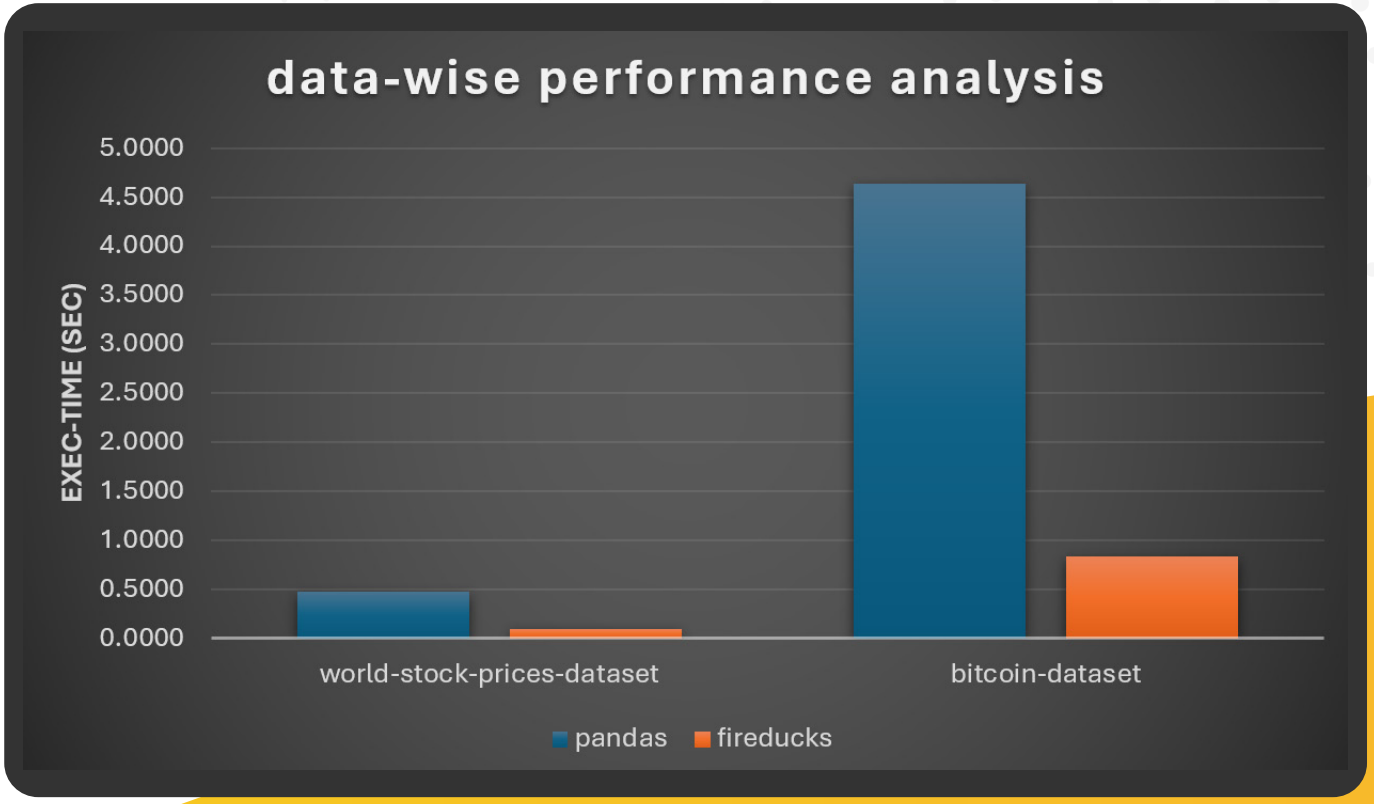
https://www.kaggle.com/datasets/mczielinski/bitcoin-historical-data?select=bitstampUSD_1-min_data_2012-01-01_to_2021-03-31.csv

This is a historical data related to Bitcoin exchange having ~5 million samples. When we attempted to perform a similar analysis as shown below with this data, we experienced a significant speed-up from FireDucks.

Complete python code:

```
import pandas as pd

# (1) load_data
data =
pd.read_csv('bitstampUSD_1-min_data_2012-01-01_to_2021-03-31.csv')
data["Timestamp"] = pd.to_datetime(data["Timestamp"], unit="s")
data["Close"] = data["Close"].bfill()

# (2) rolling_mean
tot_mins_per_day = 24 * 60 # the data has entries for each minute
short_window = 20 * tot_mins_per_day # window for last 20 days
long_window = 50 * tot_mins_per_day  # window for last 50 days
#Calculate short and long moving average
data['Short_MA'] = data['Close'].rolling(window=short_window).mean()
data['Long_MA'] = data['Close'].rolling(window=long_window).mean()

# (3) add_signal: Add a signal column and set values
data['Signal'] = 0
data.loc[(data['Short_MA'] > data['Long_MA']), 'Signal'] = 1

# (4): add_crossover: Add a Crossover column and set values
data['Crossover'] = data['Signal'].diff()
```

	Pandas exec time (sec)	FireDucks exec time (sec)	FireDucks speed-up over pandas
load_data	3.7128	0.6866	5.41x
rolling_mean	0.3668	0.0255	14.40x
add_signal	0.0302	0.0350	0.86x
add_crossover	0.0405	0.0095	4.25x
Total	**4.6323**	**0.8348**	**5.55x**



It is very evident how we can speed up the complex SMA analysis with FireDucks when data grows in size and that too just by adding an option (-mfireducks.imhook) without any changes in the pandas program.

Conclusion

FireDucks demonstrates significant performance advantages over Pandas for this specific trading strategy. FireDucks dominates Pandas with an overall 5.3~5.5 times faster execution. Even more compelling is the memory efficiency of FireDucks. It consumes 103 times less memory than Pandas throughout the entire process (0.76 MB vs. 78.82 MB). These results suggest that FireDucks is a strong alternative for data-intensive backtesting and algorithmic trading tasks, particularly when dealing with large datasets, where speed and memory efficiency are critical factors. However, it's important to note that this benchmark focuses on a specific use case. Further exploration might be needed to determine FireDucks' effectiveness across a wider range of trading strategies and data manipulation tasks.

References

FireDucks Documentation: https://github.com/fireducks-dev
FireDucks PyPI Page: https://pypi.org/project/fireducks/