Automated Algorithmic Cryptocurrency Trading Bot Using VWAP and Bollinger Bands Strategies

Automated Trading Strategies with VWAP and Bollinger Bands

Frederick Marvel, F.M.

University of Nottingham Malaysia Campus

hcyfm2@nottingham.edu.my

In this project, researcher aim to evaluate the effectiveness of Volume Weighted Average Price (VWAP) and Bollinger Bands in trading cryptocurrency. Historical data of selected trading pairs will be analyzed to test the algorithm’s reactions. Subsequently, a Python-based trading bot will be developed using various libraries and a leading cryptocurrency broker API. The bot would make API call to gather BTC-USDT trading pair data, and then execute trades based on the VWAP and Bollinger Bands algorithms using the broker’s WebSocket API. Data collected from reputable resources (Yahoo Finance, OKX) ensured data reliability. While the algorithm runs smoothly throughout the trading days, the results are satisfactory for market making. However, the algorithm’s weakness in trend following was noted, indicating room for improvement. Overall, the research adds to the studies of automated trading in the cryptocurrency market and highlights potential for further development in optimizing trading strategies.

1. Introduction

Algorithmic trading has gained popularity starting in recent years as a means of executing trades more efficiently, accurately, and less emotionally than manual trading. The use of algorithms to make trading decisions is particularly relevant in the cryptocurrency market, which operates not only on weekdays but also 24/7 and can be very volatile. In this case, Volume Weighted Average Price (VWAP), which is calculated based on the volume and price of an asset over a given period and also another popular indicators Bollinger Bands, which use a moving average and standard deviation to identify potential trading opportunities is used inside the trading bot.

Several studies have examined the effectiveness of VWAP and Bollinger Bands in different financial markets like stock, foreign exchange, commodities market. For example, a research by Ananth (2002) found that VWAP is amazingly have great upside on volatile market and also have straightforward characteristics from a conceptual points. Similarly, Wei Lu and Xudong (2006) indicates that Black-Sholes stock price model can be correlated to Bollinger Bands indicators too.

In the context of cryptocurrency market, many people especially retail traders studied to make trading bot on their own using VWAP or Bollinger Bands alone or combine it with some other indicators. Some experienced traders and fund managers also using VWAP and Bollinger Bands to market making strategies. Although there are no specific studies or research around cryptocurrency markets.

Despite these studies, there is still a need to examine the performance of VWAP and Bollinger Bands in the cryptocurrency market further. In this study, we aim to evaluate the effectiveness of VWAP and Bollinger Bands in algorithmic trading of cryptocurrencies, with a focus on the BTC-USDT trading pair. We will analyze historical data to test the algorithms' reactions, develop a Python trading bot, and execute trades based on the algorithms using a leading cryptocurrency broker API. The results of this study could provide insights into the potential benefits and limitations of algorithmic trading in the cryptocurrency market and inform future research in this area.

1. Related Works

* **VWAP Execution and Guaranted VWAP by Guéant and Royer (2018)**

In this research they developed a framework to price guaranteed VWAP contracts in the presence of permanent market impact. They find that Monte Carlo simulation and dynamic programming can be used to price guaranteed VWAP contracts. Their study provides insights into the complexities of trading in modern financial markets and contributes to the literature on optimal liquidation and VWAP strategies.

* **Empirical Optimization of Bollinger Bands for Profitability by Oliver Williams (2013)**

This project evaluates the profitability of Bollinger Bands in capturing sudden price fluctuations by testing a trading rule on the DOW 30 index. The study concludes that a single moving average cannot be used to derive an all-encompassing trading rule for all securities, contributing to the literature on technical analysis and Bollinger Bands.

1. Methodology

In order to accomplish the goal of this research, there are several steps before implementing the main program. In the next part, researchers will explain the steps and methodology that used in this project.

* 1. Algorithm Testing

Before making and running the actual implementation of this algorithm, I should make a study first on the performance of this strategy. One of the way to define the effectiveness is from back test the algorithm.

In this case, I will be using Google Collab Python Notebook to calculate the result of this strategy. I am taking the data from Yahoo Finance, which is reputable for price feed data providing. This step is very straightforward and simple. I took data from 1st January 2020 to 1st January 2023, which includes many volatile uptrend and downtrend market.

Github Repoistories Link : https://github.com/frederickmarvel/DIA\_Project

Text

Description automatically generated

Firstly I will download di Yahoo Finance packages. This packages is important to connecting the notebook to Yahoo Finance. After that, I import some libraries to accommodate the backtesting algorithm.

Text

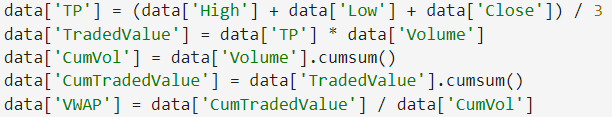
Description automatically generated with medium confidence

Using Yahoo Finance, I downloaded the BTC-USD data with start and end date that we have discussed earlier. The data would be saved on ‘data’.

**VWAP**

Firstly, we need to know first how to calculate VWAP. Below is the formula for VWAP strategies.

VWAP is simply the total average price multiply by the volume and divided buy cumulative volume.

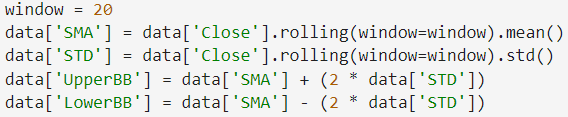


Using historical data, we calculate the VWAP for each trading data and save it on the ‘VWAP’.

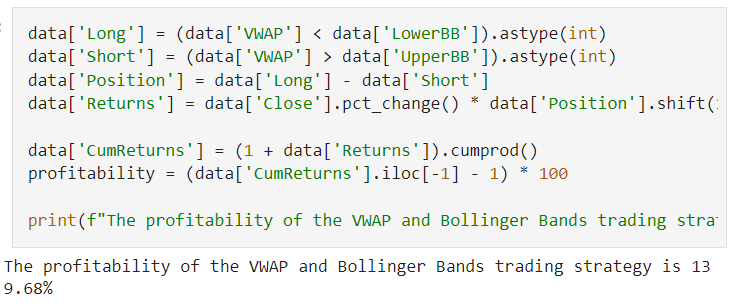
**Bollinger Bands**

After making the VWAP Formula, I will make the Bollinger Bands algorithm on the notebook with Bollinger Bands formula with 20 window period. The window period will be the main variable to justify the formula later on. Different window period means different bands. Below I will rewrite the Bollinger Bands formula.

Bollinger Bands consist of three lines, and a popular way to calculate them is by using a 20-day simple moving average (SMA) as the middle line. The upper line is determined by adding two times the daily standard deviation to the middle line, while the lower line is calculated by subtracting twice the daily standard deviation from the middle line.



The same data would be modified to calculate the Bollinger Bands. Firstly, SMA (Simple Moving Average) and standard deviation both with the same window period would be calculated. UpperBB is for the upper bands limit and LowerBB is for lower bands limit. The formula is the same as I shown before.



The profitability of this algorithm is counted based on the algorithm using 2 sides trading (long and short). The “long” trades only happened when VWAP is lower that lowerBB, which means price is already oversold. Different from long, Short trade is executed when VWAP value is larger than upperBB this means that the price is overbought and likely to make a downtrend retracement. The profitability is counted based on the difference between long and short data.

The test turns out outperform the general index market such as S&P500 and Nasdaq with 139.68% of profitability. Although this is just back test, and real-time market would require much more adjustment.

* 1. Real Market Implementation

After back test the main algorithm with historical data, next we are going to make the implementation of our previous algorithm. There are several things to consider when trying to build the trading bots.

* **Compatible Broker/Exchange**

This research would require cryptocurrencies exchange / broker that can connect to WebSocket API. I should tell that there are many broker that have this specification but regulatory issue would make this function unavailable.

* **Demo Account**

As I found the compatible broker with a lot of flexibility, I found that I have to invest real money on this algorithm. As the algorithm only work on certain range of deposits, I would search for another reputable broker that facilitates demo account.

* **Higher Liquidity**

As we know, that top broker might have higher liquidity. But in some cases there are liquidity crunch that would damage the liquidity and also our algorithms too.

After examining some of the exchanger, I decided to choose OKX as the broker to test my trading bots. OKX offer demo trading account so I can launch my trading bots without the fear of losing money right away. Chart

Description automatically generated

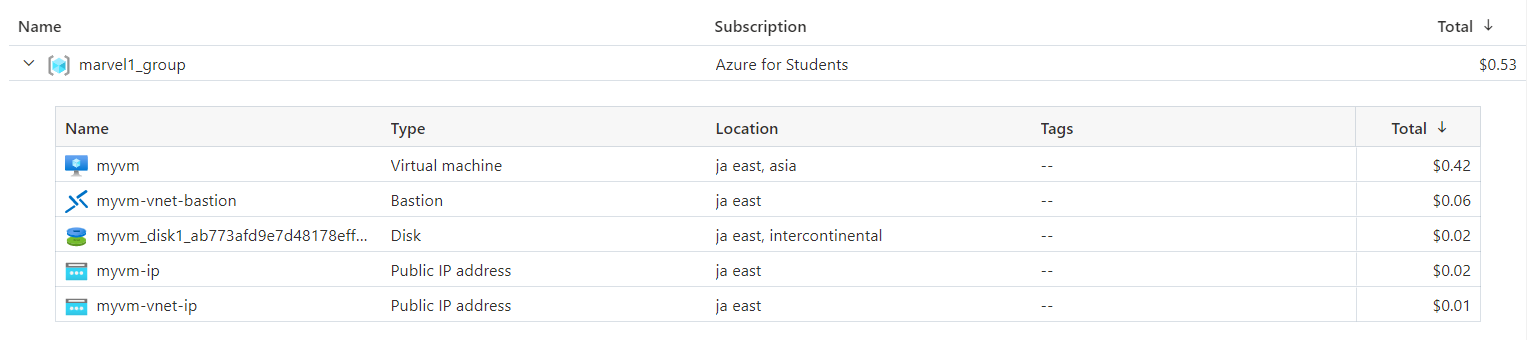
The demo trading is using real time orderbook data, but the asset is not the part of real trading account. In the account I was given around 150k demo account fund.

Graphical user interface, text, application

Description automatically generated

**Server Setup**

After finding the broker, I need to rent server instances for the python code to run 24/7. As I was told that I have student account in Azure and was given $100 free credit, I built my server on azure using basic 2 Core / 4GB RAM specification. Along configuration, I found difficulty to connect my laptop to my Azure server CLI. Searching for solutions, I read that azure offer web-based CLI named Bastion. After subscribing to Bastion (Azure web-based CLI), I opened up the CLI and its working fine for some period of time before force exit after connecting just for several minutes.



Finding out that this issue would delay my project, I change my subscription to DigitalOcean. Using lower specification and much cheaper cost, I finding out that DigitalOcean somehow more comfortable rather than Azure for small size project.

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

This is the code inside the server. I use Visual Studio code because it is so much easier to access the server SSH and code through rather than using VIM on the basic CLI. As the server is setup, I code the main program for the trading bot and install libraries to support my code.

**Trading Pair**

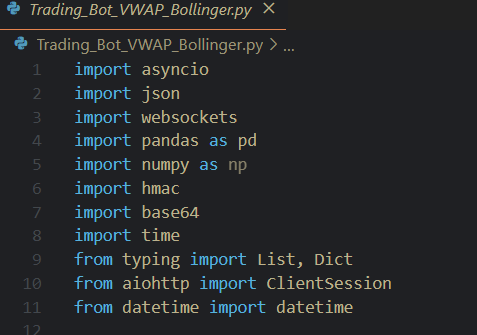
After last back testing, the real algorithm will be quite the same as the back testing algorithm. The only difference is only on the timeframe of the trade and also a little difference from the trading pairs (ticker). OKX have many trading pairs available on their platform, but as we back test our algorithms on BTC/USD, I should do the real trading on the BTC/USD pair.

Researchers realize that there are no BTC/USD ticker on OKX, but there is BTC/USDT or BTC/USDC. To simplify the confusion, USDT or USDC is a stablecoin that made by several entities to make USD pair trading on cryptocurrency market more seamless. Rather than transferring the money through wire or SWIFT, people usually use USDT or USDC that they get from third-party exchanges to trade in cryptocurrency markets. USDT and USDC will pegged to the USD by 1:1, making it reliable for our bot. Recently, the custodians of USDC having some problems with the collapse of Signature Bank, and making USDC de-pegged to 1:0.97 making it not reliable to be a trading pair.

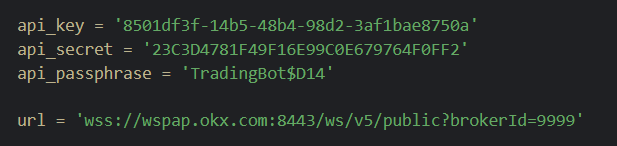
In conclusion, I choose BTC/USDT pair as the trading pair for the trading bot. Although some people might say that USDT or USDC still have the same threat to de-pegging I would say the popularity and trading volume in USDT is higher than USDC (especially in OKX), making it more subtle to our trading bot.

**Code Implementation**

After server configuration and other preparation, I make the trading bot to execute the trades on OKX demo-trading platform.



Firstly, I have to import certain library to support this bot. Many libraries are basic tools for python environments, but I like to point out asyncio. Asyncio used to asynchronous programming in python. They provide a way to write concurrent code that can handle many tasks or operations simultaneously without blocking or waiting for a response.



Next, I have to make API from OKX platform. API is used by my bot to communicate with the exchange. The url is used for WebSocket API that I used for this bot.

Text

Description automatically generated

The **create\_signed\_request** function generates a signed request for the OKX API to ensure secure and authenticated communication. It takes the HTTP method, API endpoint, query parameters, and request body as inputs. After constructing the inputs, it creates HMAC signature with API secret key and base64 encoding to the signature. It then returns a dictionary containing the required API headers, endpoint URL, query parameters, and request body, which is used for making authenticated requests to the OKX platform.

Text

Description automatically generated

The **request** function is an asynchronous function that makes HTTP requests to the OKX API using the provided method, endpoint, query parameters, and request body. It first calls the **create\_signed\_request** function to obtain a signed request containing the necessary API headers, endpoint URL, query parameters, and request body. Then, it creates an asynchronous HTTP client session using the aiohttp library's ClientSession. After sending the HTTP request to the modified url, it will returns the JSON response from the OKX API as a dictionary.

Text

Description automatically generated

The **CandlestickData** class is used to store and manage candlestick data for a trading pair. The class has pandas dataframe to store the historical data, and provides method to request new candlestick data and retrieve the data for certain frequency.

Same as the notebook code, the **calculate\_vwap** function takes the candlestick data as input and calculate the VWAP value according to the formula given above. It will return the VWAP result to the function caller later on.

The **calculate\_bollinger\_bands** do exactly the same with the formula I have given above. It will get the historical data from the API, receive window size, and number of standard deviation as input. After calculating the Bollinger Bands formula, it will return the value of lower and upper band of the price retrieved before.

Text

Description automatically generated

The **process\_data\_and\_execute\_trades** function is an asynchronous function that processes the candlestick data and execute trades based on a simple trading strategy. This function implemented trading strategy to buy when the current price is below both the lowerBB and the VWAP, and also to sell the current price is above the upperBB and the VWAP. VWAP and Bollinger Bands are using latest 60 candlestick data points.

Text

Description automatically generated

The **execute\_order** function is an asynchronous function that sends a request to execute a limit order on the OKX platform. The function would send the limit order to OKX platform. It would choose whether to buy / sell according to side. The response from the API is stored on the response variable. Limit order would require requested price point, so it would required **px** for the price point and **sz** for the order size.

Text

Description automatically generated

The **subscribe\_authenticated** function is an asynchronous function that establishes a WebSocket connection to the OKX platform and subscribes to specified authenticated channels. The function establishes a WebSocket connection using the **websockets.connect** method, passing in the url and the headers from the signed request. It then iterates through the channels list, sending a JSON-formatted subscription message for each channel.

Once subscribed, the function enters a loop that continually receives messages from the WebSocket. Upon receiving a message, it prints the message and loads it as a JSON object into the data variable. If the data object contains a key named 'data', the function calls the **process\_data\_and\_execute\_trades** function, passing in the data object to process the received data and potentially execute trades based on the implemented trading strategy.

Text

Description automatically generated

The **subscribe** function is an asynchronous function that connects to a WebSocket, subscribes to specified public channels, and processes the received data using the **process\_data\_and\_execute\_trades** function. It establishes a connection, sends out subscription messages for each channel, and also listens for incoming messages. Next, the function would proces the data if it contains a specific data key. This function is designed to work with the OKX trading platform and facilitates the execution of trades based on a given trading strategy.

Text

Description automatically generated

The **main** function initializes the subscription to the OKX trading platform's public WebSocket channel for candlestick data of the BTC-USDT trading pair. It sets up an asyncio event loop and runs the subscribe function to connect to the WebSocket and process the received data. This script serves as the main entry point for the algorithmic trading application.