Lee Schmalz Follow

Jun 7, 2020 · 10 min read · ✨ · Listen

Save

Dynamic Cryptocurrency Trading Backtesting Platform — Python

Is your trading strategy profitable? Watch the YouTube Series: https://www.youtube.com/watch?v=PZDFXGIAm2g&t=953s

I've recently been very interested in cryptocurrency day trading. While a quick internet search returns a slew of various indicator ideas and trading strategies, such as EMA/SMA crossings, RSI strategies, On Balance Volume, etc., I've found myself never able to find a resource that allows me to answer the question: "Well, does this actually work?". Frankly, it is impossible to know the answer to this question by looking at charts without testing the strategy over hundreds or even thousands of iterations; a tedious task for a human to pour over. I've seen videos and articles of others trying to backtest by hand, clicking, entering, and calculating the buys and sells dictated by their predetermined strategy. In some cases only to find the answer to their question after hours and hours of backtesting: "No, this strategy doesn't work. On to the next one." Other, more programmatic ways of backtesting can speed up the process, such as the TradingView pine editor, or a basic python script; but what if you want to dynamically alter things like stop loss/take profit levels, the cryptocurrencies you're choosing to trade, or specifics of the strategy itself? Answers to these questions would entail time-consuming alterations to a simple backtesting script. Since this initial interest in cryptocurrency trading, I decided I needed a better platform that was not available anywhere on the web (that I could find free of cost) that would allow me to dynamically iterate over any alterations in any strategy that pique my curiosity. I decided to learn more about dynamic python programming and the Binance API, to ultimately create a tool that would help my former self as a beginner python programmer and trader to answer these questions at the forefront of every "Cryptocurrency Trading Strategy" post on the internet.

The following is a trading environment in which all possible trading strategies can be tested in a very dynamic way that allows even a beginner python programmer to create and backtest their own trading ideas and ultimately, give them an answer to their questions. The only prerequisite knowledge to make use of the environment and implement your own ideas are: a basic understanding of python structures such as for loops and if statements, enough knowledge of pandas to grab the proper price datum from the correct column/index location of the dataframe, and potentially a low-level understanding of object oriented programming, depending on the complexity of your strategy.

is a single script that reads minute-by-minute price changes for 12 different cryptocurrencies from March 28, 2019 through June 1, 2020 for a total of 618,290 instances of prices for each of 12 cryptocurrencies, resulting in a 211 MB dataframe (this can be expanded, but requires a bit of patience). On my machine, this read from the binance API took approx. 5 hours to complete, but with a simple write to csv can be near instantly retrieved at any future use of the script.

A few notable things about the data read:

1. To access the binance API, simply pip install binance-client from the command line. Other dependency installations are commented on their respective import line.
2. The binance client asks for API keys, these are only necessary for actually sending trade orders to the binance exchange in real-time, and are not necessary for reading data. I left the formatting as is in case this is a desired use-case for any readers.
3. After the data from each coin is gathered, we let the script sleep for 60 seconds. This is to prevent the binance API from kicking us out for asking for too much data too fast. I haven't had issues with this format, but if an error is encountered, this time may have to be increased.
4. Included here is a Simple Moving Average function to show an example of how features can be engineered from the price data and added to the dataframe. This can be a custom function of your choosing, or simply a function of a common indicator that can be found online, as shown here.
5. Volume data is also available, but not used in this example. Add this to line 32 if this is of interest.
6. Custom naming of the csv file can be edited in quotes in line 72.

```
1  import numpy as np #pip install numpy
2  from tqdm import tqdm #pip install tqdm
3  from binance.client import Client #pip install python-binance
```

```
SMA_HIGH = 150

def compute_sma(data, window):
    sma = data.rolling(window=window).mean()
    return sma


#select cryptocurrencies you'd like to gather and time interval
ratios = ['BTC','ETH','LTC','XLM','XRP','XMR','TRX','LINK','IOTA','EOS','DASH','ZRX']
START_TIME = '28 Mar, 2019'
END_TIME = '1 Jun, 2020'
api_key=''
api_secret=''

client = Client(api_key=api_key,api_secret=api_secret)

merge = False
for ratio in ratios:
    print(f'Gathering {ratio} data...')
    data = client.get_historical_klines(symbol=f'{ratio}USDT',interval=Client.KLINE_INTERVAL_1MINU
    cols = ['time','Open','High','Low',f'{ratio}-USD_close',f'{ratio}-USD_volume','CloseTime','Quo

    temp_df = pd.DataFrame(data,columns=cols)
    temp_df = temp_df[['time',f'{ratio}-USD_close']]

    if merge == False:
        df = temp_df
    else:
        df = pd.merge(df,temp_df,how='inner',on='time')
    merge = True
    print('complete')
    time.sleep(60) #sleep for a bit so the binance api doesn't kick you out for too many data asks


for col in df.columns:
    if col != 'time':
        df[col] = df[col].astype(np.float64)

for ratio in ratios:
    df[f'{ratio}_{SMA_LOW}'] = compute_sma(df[f'{ratio}-USD_close'], SMA_LOW)
```

```
54
55  #convert binance timestamp to datetime
56  for i in tqdm(range(len(df))):
57      df['time'][i] = datetime.fromtimestamp(int(df['time'][i]/1000))
58
59  df.to_csv('12-coins-Mar18_Jun20')
```

readdata_binance.py hosted with ❤ by GitHub view raw

```
In [6]: df.head()
```

Out[6]:

	time	BTC-USD_close	ETH-USD_close	LTC-USD_close	XLM-USD_close	XRP-USD_close	XMR-USD_close	TRX-USD_close	LINK-USD_close	IOTA-USD_close	...	LINK_40	LINK_150	IOTA_40	IO
0	2019-03-28 01:44:00	4025.99	138.09	60.58	0.10665	0.30631	52.68	0.02307	0.4907	0.3036	...	0.491242	0.493144	0.304802	0.
1	2019-03-28 01:45:00	4024.29	138.08	60.60	0.10664	0.30626	52.68	0.02306	0.4907	0.3039	...	0.491135	0.493091	0.304770	0.
2	2019-03-28 01:46:00	4025.31	138.01	60.56	0.10662	0.30619	52.68	0.02307	0.4909	0.3034	...	0.491087	0.493045	0.304720	0.
3	2019-03-28 01:47:00	4024.46	138.01	60.56	0.10668	0.30606	52.68	0.02307	0.4909	0.3029	...	0.491075	0.493001	0.304650	0.
4	2019-03-28 01:48:00	4024.50	138.06	60.59	0.10666	0.30611	52.68	0.02308	0.4909	0.3029	...	0.491062	0.492956	0.304587	0.

Example data read from binance API

Great, now we have a large dataset of cryptocurrency prices. Before we get into the trading environment, let's create a simple memory class that will allow us to store information about each backtest episode.

Memory

This class is pretty self-explanatory; a simple memory object that allows us to append buy/sell actions and also to clear the memory after each episode.

Now that we have our data and our memory object, let's get into the trading environment.

Our environment takes in two parameters: the dataframe that we just created as well as the ratios of the currencies you wish to include in the backtesting process (ex. 'BTC', 'LTC', etc.).

Our environment also has two methods: reset() and step().

Env.reset() Method

This is another item that needs little explanation: it simply resets the trading environment to its default state. The purpose of this is when running multiple episodes of backtesting, we will call env.reset() following each episode to reset all of the local variables such as coin balances, the index location of the episode dataframe, any variables that might be


1. Notice each reset will grab a small chunk of the main dataframe and call it 'episode_df', rather than running through the entire 600,000+ instances of data during backtesting. This is the interval that the strategy will be backtested on for each episode. The duration of this episode dataframe can be specified using either of the constants MINUTES_PER_EPISODE or DAYS_PER_EPISODE. In this example, we use an episode size of 1 day or 1440 minutes.

2. The reset() method is called upon initiation of the environment (in the __init__) method. This sets all environment variables upon first call of the Env object.

Env.step() Method

The step method is where the magic happens. I'll divide this method into two sections: implementing the trading strategy and calculating performance metrics.

The Strategy

In this example, rather than using a more complex trading strategy you might encounter in deep technical analysis discussions, I'm going to instead be debunking the proclaimed efficacy of a common strategy we've all seen elsewhere online: the Moving Average Crossing Strategy. If you are unfamiliar, the gist of this idea is that when a short window moving average crosses above a longer window moving average → Buy; and when the short window moving average crosses back below the longer window moving average → Sell.

Now, I'm not sure if anyone believes this strategy actually works (it may in some cases), but let's consider it for sake of demonstration of the backtesting environment (and because implementing more complex strategies would remove some of the fun from tinkering with your own ideas). As apparent in the code below, we have two key if statements that examine the moving average changes and decide whether to record a buy or sell (or neither). A couple local variables to be familiar with: *money_in* and *to_buy*. These are string variables that keep track of where your money is currently held, and where it's about to go when a buy is initiated. Once the buy and sell *if* statements are


using BNB token (the cheapest option available on the binance exchange), but can be altered based on the trading fees of your specific exchange.

Performance Metrics

The two metrics we will use to evaluate performance of the strategy over the backtesting interval are: the net worth of your balances following the backtesting interval and the average market change of the backtesting interval. The net worth of your balances are simply calculated by converting your holdings to USD based on the current price of the currency you hold and the amount of that currency you are holding. The average market change of the interval is calculated by taking the average of the prices of all of the selected currencies at both the beginning and the end of episode, and dividing these values accordingly. The ultimate goal, based on these metrics, is to complete the episode with a higher net worth than what would have happened had you held an average stake in the market (and to have a higher net worth than when you started, obviously).


Executing the Environment

Here we step through the environment one minute at a time, buying and selling as dictated by the strategy, and keep track of performance metrics along the way to be outputted to the log.


How did we do?

Well, as previously foreshadowed due to the simplicity of the strategy, not very well. As shown, we would expect the implementation of this strategy to cause a net worth decline of 4% per day, on average (ouch), in a market that grew by 0.2% per day, on average. But, besides the point, it is shown in the log we now have a quick, dynamic, and easy to read the output of one strategy backtested over 100 randomly selected 1-day intervals between March 2019 and June 2020 along with buy/sell information for each trade. Now, there are some answers to your questions!

```
episode: 96
['Buy BTC: 5255.99', 'Sell BTC: 5227.76', 'Buy ETH: 155.86', 'Sell
ETH: 155.41', 'Buy LTC: 67.46', 'Sell LTC: 67.45', 'Buy IOTA: 0.3167',
'Sell IOTA: 0.3176', 'Buy IOTA: 0.3167', 'Sell IOTA: 0.3151', 'Buy
LTC: 67.67', 'Sell LTC: 67.42', 'Buy XMR: 61.38', 'Sell XMR: 61.59',
'Buy EOS: 4.5225', 'Sell EOS: 4.5076', 'Buy ZRX: 0.2743', 'Sell ZRX:
0.2723', 'Buy ETH: 155.2', 'Sell ETH: 156.98', 'Buy XLM: 0.09605',
'Sell XLM: 0.09597', 'Buy LINK: 0.439', 'Sell LINK: 0.439', 'Buy LINK:
0.4418', 'Sell LINK: 0.4672', 'Buy BTC: 5308.08', 'Sell BTC: 5304.94',
'Buy XLM: 0.09966', 'Sell XLM: 0.09965', 'Buy XMR: 62.29', 'Sell XMR:
62.22', 'Buy TRX: 0.02312', 'Sell TRX: 0.02307', 'Buy LTC: 72.82']


interval: 2019-04-29 12:25:00 - 2019-04-30 12:26:00
net worth after 1 day(s): 1.0221672847272814
average market change: 0.988959335567251

episode: 99
['Buy XRP: 0.30917', 'Sell XRP: 0.31171', 'Buy BTC: 9641.57', 'Sell
BTC: 9635.6', 'Buy TRX: 0.02252', 'Sell TRX: 0.02238', 'Buy XLM:
0.08462', 'Sell XLM: 0.08457', 'Buy LINK: 2.2158', 'Sell LINK: 2.211',
'Buy EOS: 4.2811', 'Sell EOS: 4.2527', 'Buy ETH: 212.07', 'Sell ETH:
211.26', 'Buy IOTA: 0.2835', 'Sell IOTA: 0.2833', 'Buy XLM: 0.08375',
'Sell XLM: 0.08317999999999999', 'Buy LTC: 89.2', 'Sell LTC: 89.34',
'Buy XMR: 79.39', 'Sell XMR: 79.15', 'Buy BTC: 9570.01', 'Sell BTC:
9548.48', 'Buy LINK: 2.1819', 'Sell LINK: 2.1595', 'Buy LTC: 89.33',
'Sell LTC: 89.4', 'Buy EOS: 4.1803', 'Sell EOS: 4.1903', 'Buy BTC:
```

```
interval: 2019-07-28 16:39:00 - 2019-07-29 16:40:00
net worth after 1 day(s): 0.9218801235080804
average market change: 0.9984598063740531


net worth average after 100 backtest episodes: 0.9656146271760888
average, average market change over 100 episodes: 1.0021440510159623
```

Shown is the output of just two of the 100 episodes. For the sake of brevity, I chose one profitable episode (96) and one unprofitable episode (99) to display, while the actual output displays all 100 episode results. What we really care about, though, is not the results of individual episodes, but the question: “How does my strategy perform on average?”. This is shown in the final average episode log.

The python logging package might be useful here for someone running many different strategies and logging in a separate file to compare results.

Ultimately, day trading is a dangerous space. I hope someone reading this will find this tool useful in exploring their trading ideas, allowing them to be informed in their decisions before bringing their strategies to market. There seems to be a lack of resources out there for someone with a limited knowledge of programming and algorithmic trading to make informed trading decisions; I hope this can partially fill that void.

Some Additional Thoughts

In my opinion, using the environment as we have in this example was a nice way to introduce dynamic algorithmic trading to someone who has limited knowledge of object-oriented programming. Personally, I’ve used this tool for backtesting many different strategies that incorporate far more features of the price data, and also allow for some additional functionality; namely, the ability to trade cryptocurrency to cryptocurrency (ex. BTC/LTC pair, etc.), rather than only cryptocurrency to USD and vice-versa. This greater functionality is not an extremely difficult addition; though if there is interest, an outline of a more complex example could be an idea for a future article.

contribution. You should not rely on an author's works without seeking professional advice. See our Reader Terms for details.