Published in CodeX

Nikhil Adithyan Follow

Mar 21, 2021 · 9 min read · Listen

Save

CODEX

Algorithmic Trading with SMA in Python

Creating and backtesting an SMA trading strategy in python

Photo by Markus Spiske on Unsplash

Introduction

With the increasing number of technological innovations, many industries deprecated their traditional methods and hoped into the latest tech ones to keep themselves updated. And so is the financial industry! In the past few years, there has been a lot going on with the financial industry. An enormous amount of technological solutions for finance are invented and enhanced. In this article, we are going to see one of the notable technological innovations in finance which became a quantum leap to the stock trading space and that is nothing but Algorithmic Trading! Let's dig deeper into this topic.

Algorithmic Trading

In most cases, a stock trade is influenced by humans' emotions or irrational thinking (sometimes). As a result, it reflected bad results in most of the trades done by the trader. People were thinking of a trading method where they could keep their emotions aside and it was this time the concept of Algorithmic trading was invented. Algorithmic trading is the process of enabling computers to trade stocks under certain conditions or rules. A trade will be performed by the computer automatically when the given condition gets satisfied. Also, these conditions are given to the computers by human traders. The conditions or nothing but trading strategies are defined by human traders.

In this article, we will be creating a trading strategy using a technical indicator called Simple Moving Average (SMA). Before moving on, if you want to backtest your trading strategies without any coding, there is a solution for it. It is BacktestZone. It is a platform to backtest any number of trading strategies on different types of tradeable assets for free without coding. You can use the tool right away using the link here: https://www.backtestzone.com/

Simple Moving Average

Simple Moving Average (SMA) is nothing but the average price of a specified period of time. It is a technical indicator and widely used in creating trading strategies. Usually, two SMAs are calculated to build a trading strategy, one with a short period of time and the other with longer than the first one. Now, let's get an intuition on the trading strategy we

period crosses above the SMA calculated with a longer period. Likewise, the computer sells the stock when the SMA calculated with a longer period crosses above the SMA calculates with a shorter period. This is how our condition for the trading strategy looks like:

```
IF SMA(SHORT PERIOD) > SMA(LONG PERIOD) => BUY
IF SMA(LONG PERIOD) > SMA(SHORT PERIOD) => SELL
```

With that, we have finished our theory part. Now let's implement the trading strategy in python and see it in action.

Implementation in Python

Now, we are going to code our trading strategy in python and see how well it works.

Importing Packages

In this step, we are going to import the required packages into our python environment. The primary packages are going to be Pandas to work with dataframes, Matplotlib to create plots, Requests to make API calls, NumPy to work with arrays. The additional packages are Math to perform mathematical calculations and Termcolor to customize fonts in python.

Python Implementation:


We have imported all the required packages into our python environment. Now let's extract the historical data of Microsoft (MSFT) from IEX Cloud. Before moving on, if you don't know what is IEX Cloud and how to pull data from it, I highly recommend you to view my article on it (click here to view the article). Let's pull some data!

Extracting data from IEX Cloud

In this step, we are going to pull the historic data of Microsoft using an API provided by IEX Cloud.

Python Implementation:

Output:

	open	high	low	close
2021-03-11	234.96	239.170	234.31	237.13
2021-03-12	234.01	235.820	233.23	235.75
2021-03-15	234.96	235.185	231.81	234.81
2021-03-16	236.28	240.055	235.94	237.71
2021-03-17	236.15	238.550	233.23	237.04


Code Explanation: First we are defining a function named 'get_historic_data' that takes a stock's ticker as the parameter. Inside the function, we are storing the API key and the URL into their respective variables, and then using the 'GET' method provided by the Request package, we are extracting the data in a JSON format. Next, we are doing some data manipulation tasks to clean and make the data usable. Finally, we are returning the dataframe. After finished defining the function, we are calling it and stored the data into the 'msft' variable. Let's calculate the SMA values out of the extracted data.

SMA Calculation

In this step, we are going to calculate two SMA values (SMA 20, 50) and append those values to our dataframe.

Python Implementation:


Output:

	open	high	low	close	sma_20	sma_50
2021-04-14	257.475	258.83	255.1600	255.59	242.3545	239.176853
2021-04-15	257.931	259.93	257.7300	259.50	243.4775	239.587613
2021-04-16	259.470	261.00	257.6014	260.74	244.9785	239.953532
2021-04-19	260.190	261.48	257.8210	258.74	246.3980	240.299206
2021-04-20	257.820	260.20	256.8400	258.26	247.5115	240.631489

Image by Author

Code Explanation: Firstly, we are defining a function named 'sma' that takes data and the number of periods as the parameters. Inside the function, we are using the 'rolling' function provided by the Pandas package to calculate the SMA for the given number of periods. We are storing the calculated values into the 'sma' variable and returned it. Next, we are calling the function and calculated two SMA values, the shorter one with 20 as the number of periods, and the longer one with 50 as the number of periods. Now, let's make a plot out of the calculated SMA values.

Plotting the SMA Values

In this step, we are going to plot the calculated SMA values to make more sense out of them.

Python Implementation:

Output:

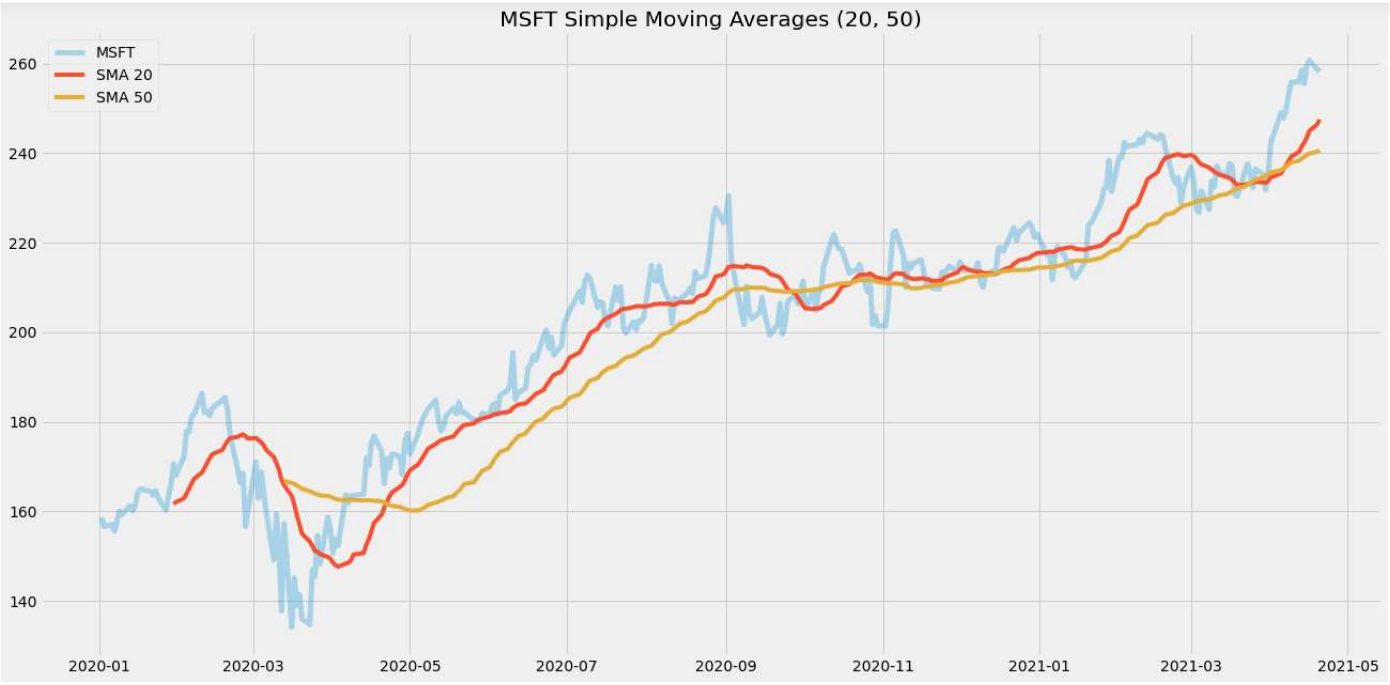

Image by Author

Code Explanation: Using the 'plot' function provided by the Matplotlib package, we have plotted the SMA values along with the 'close' prices of Microsoft. Now let's observe the graph. The light blue line represents the 'close' prices of Microsoft, and the red and gold line represents SMA 20 & SMA 50 respectively. It is observed that the golden line (SMA 50) is way smoother than the red line (SMA 20) because the specified period of time for the golden line's values is considerably higher than the red line's values.

Now that we have our SMA values. So let's proceed in creating the trading strategy.

Creating a Trading Strategy

In this step, we are going to implement the discussed trading strategy in python.

Python Implementation:

Code Explanation: First, we are defining a function named 'implement_sma_strategy' which takes data, and the period of time for both short and long SMA as the parameters.

Inside the function, we are first storing the specified period of time into the 'sma1' and the 'sma2' variable. Next, we are storing three empty lists in which the values will be appended while creating the trading strategy.

After that, we are implementing the trading strategy through a for-loop. Inside the for-loop, we are passing certain conditions, and if the conditions are satisfied, the respective values will be appended to the empty lists. If the condition to buy the stock gets satisfied, the buying price will be appended to the 'buy_price' list, and the signal value will be appended as 1 representing to buy the stock. Similarly, if the condition to sell the stock gets satisfied, the selling price will be appended to the 'sell_price' list, and the signal value

sense unless we plot the values. So, let's plot the values of the created trading lists.

Plotting the Trading lists

In this step, we are going to plot the created trading lists to make sense out of them.

Python Implementation:

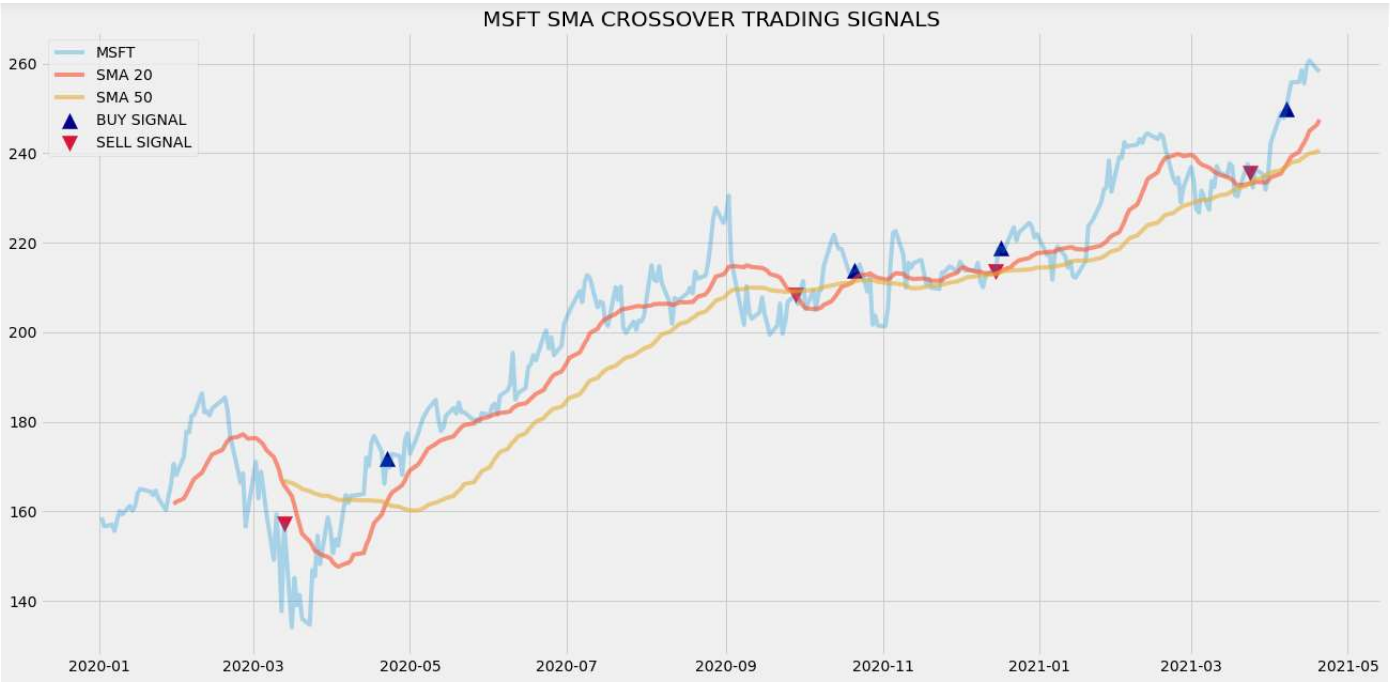

Image by Author

Code Explanation: We are plotting the SMA values along with the buy and sell signals generated by the trading strategy. We can observe that whenever the red line (SMA 20) crosses above the golden line (SMA 50), a buy signal is plotted in blue color, similarly, whenever the golden line crosses above the red line, a sell signal is plotted in red color.

Now, using the trading signals, let's create our position on the stock.

Creating our Position

In this step, we are going to create a list that indicates 1 if we hold the stock or 0 if we don't own or hold the stock.

Python Implementation:

Output:

	sma_20	sma_50	sma_signal	sma_position
181	212.509089	209.198639	0	1
182	212.197134	209.215602	0	1
183	211.407047	209.071227	0	1
184	210.513472	208.984750	0	1
185	209.580095	209.072212	0	1
186	208.611392	209.212561	-1	0
187	207.702394	209.136406	0	0
188	206.859568	209.177929	0	0

Image by Author

Code Explanation: First, we are creating an empty list named 'position'. We are passing two for-loops, one is to generate values for the 'position' list to just match the length of the 'signal' list. The other for-loop is the one we are using to generate actual position values. Inside the second for-loop, we are iterating over the values of the 'signal' list, and the values of the 'position' list get appended concerning which condition gets satisfied. The value of the position remains 1 if we hold the stock or remains 0 if we sold or don't own the stock. Finally, we are doing some data manipulations to combine all the created lists into one dataframe.

From the output being shown, we can see that from row 181–185, our position in the stock has remained 1 (since there isn't any change in the SMA signal) but our position suddenly turned to 0 as we sold the stock when the SMA trading signal represents a sell

Backtesting

Before moving on, it is essential to know what backtesting is. Backtesting is the process of seeing how well our trading strategy has performed on the given stock data. In our case, we are going to implement a backtesting process for our SMA crossover strategy over the Microsoft stock data.

Python Implementation:

```
Profit gained from the strategy by investing $100K in MSFT : $19971.23
in 1 Year
```

Code Explanation: First, we are calculating the returns of the Microsoft stock using the 'diff' function provided by the NumPy package and we have stored it as a dataframe into the 'msft_ret' variable. Next, we are passing a for-loop to iterate over the values of the 'msft_ret' variable to calculate the returns we gained from our SMA trading strategy, and these returns values are appended to the 'sma_strategy_ret' list. Next, we are converting the 'sma_strategy_ret' list into a dataframe and stored it into the 'sma_strategy_ret_df' variable.

Next comes the backtesting process. We are going to backtest our strategy by investing a hundred thousand USD into our trading strategy. So first, we are storing the amount of investment into the 'investment_value' variable. After that, we are calculating the number of Microsoft stocks we can buy using the investment amount. You can notice that I've used the 'floor' function provided by the Math package because, while dividing the investment amount by the closing price of Microsoft stock, it spits out an output with decimal numbers. The number of stocks should be an integer but not a decimal number. Using the 'floor' function, we can cut out the decimals. Remember that the 'floor' function is way more complex than the 'round' function. Then, we are passing a for-loop to find the investment returns followed by some data manipulations tasks.

Finally, we are printing the total return we got by investing a hundred thousand into our trading strategy and it is revealed that we have made an approximate profit of twenty thousand USD in one year. That's not bad!

Final Thoughts!

We've come along a long way from exploring what is Algorithmic Trading and SMA to implement and backtesting our trading strategy in python. There are still a lot of spaces to improve our code and gain better results. Some ways are to use Machine Learning to find the best stock to implement the trading strategy, tune and improve the trading strategy,

the coding parts, don't worry. I've provided the full source code used in this article. Hope you've learned something useful from this article.

Full code: