Python for Options Trading (1): Selecting a Call Spread with the Highest Probability of Profit

Roberto Gomes, PhD · Follow

6 min read · Aug 7, 2023

Listen Share More

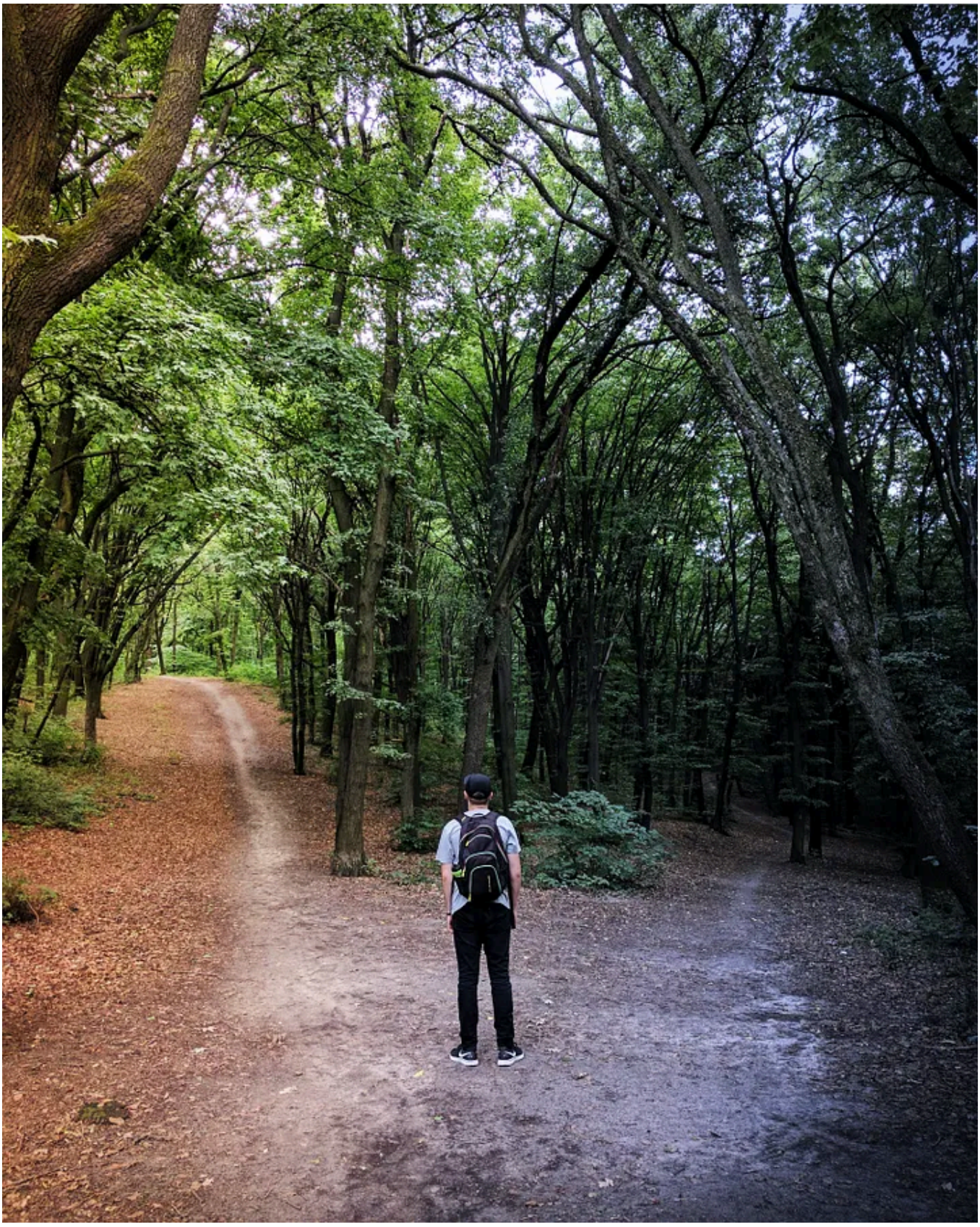


Photo by Vladislav Babienko on Unsplash

EDIT: The library has undergone significant changes, rendering the code from the previous version of this article incompatible. As a result, this article has been updated to reflect the latest version of ***OptionLab****.*

Welcome to my blog on Medium.com. In this article, I will demonstrate how to use Python to select the combination of any two call options to construct a **call**

spread with the highest probability of profit (PoP). To achieve this, I will utilize the **OptionLab** library introduced in a previous article to perform the PoP calculations.

First and foremost, a caveat is necessary: *options trading is a highly risky activity and should be approached with due diligence, as it can lead to significant losses. The content of this article is not a recommendation and is purely for educational purposes.*

Moreover, **OptionLab** is provided as is, with no guarantee that its results are accurate. As such, I am not responsible for any losses caused by the use of the code. Bugs can be reported as issues on the project's GitHub page. I am open to hearing any questions, corrections, comments or suggestions about the code. You can also reach me on Linkedin or X.

Okay, now we can proceed.

A call spread, as you probably already know, is a type of vertical spread. As such, it's a two-legged strategy where the trader buys a call and simultaneously sells another call, both with same expiration but different strikes. A call spread provides a downside protection (which is beneficial for risk management) while also capping the maximum profit.

If the trader buys the lower strike call and sells the higher strike call, being the first one more expensive than the latter, we have a **bull call spread** that results in a net debit. This strategy benefits from a bullish market, resulting in a profit when the price of the underlying asset rises.

On the contrary, when the trader sells the more expensive call with a lower strike and buys the cheaper call with a higher strike, he/she pockets the difference in premiums of these options. In this case, it is a **bear call spread** that performs best with the decline in the stock price.

After this very short introduction to call spreads, let's go ahead and code!

The first step is to install **OptionLab** on your computer. Provided you've already installed a Python distribution such as Anaconda, open a terminal, run `pip install optionlab` and it's done.

Next, you can see a Python code snippet where the required libraries are imported, some input data is defined, and the option chain data (calls only) is loaded from a .csv file


```
from __future__ import print_function

import datetime as dt

import matplotlib.pyplot as plt
import pandas as pd
from numpy import zeros

from optionlab import Inputs, run_strategy

# Input data (Microsoft, MSFT)
stockprice = 342.97
volatility = 0.18
startdate = dt.date(2021, 11, 22)
targetdate = dt.date(2021, 12, 17)
interestrate = 0.001
minstock = 0.0
maxstock = stockprice + round(stockprice * 0.5, 2)

# Load the option chain data (calls only) from a .csv file
df = pd.read_csv("msft_22-November-2021.csv")
chain = []

for i, _ in enumerate(df["Expiration"]):
    if (
        df["Expiration"][i] == targetdate.strftime("%Y-%m-%d")
        and df["Type"][i] == "call"
    ):
        chain.append([df["Strike"][i], df["Bid"][i], df["Ask"][i]])
```

The underlying asset is Microsoft stock, traded for $342.97 on November 2021, 22. A list called `chain` is created and the strike, bid price and ask price of each call option expiring on December 2021, 17 is appended to it.

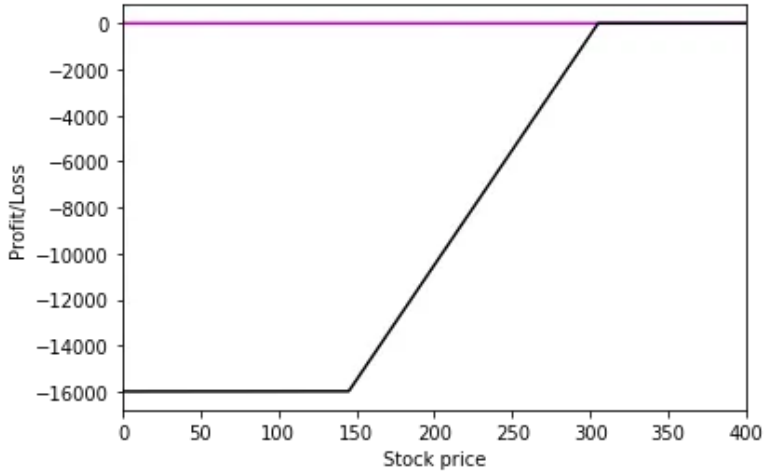
In the function `get_highest_pop()` below, a Cartesian product is used to construct pairs of call options. For each pair, there are two possibilities: buy the first call and sell the second one, or vice versa. The calculations are performed by calling the function `run_strategy()`. The function `get_highest_pop()` returns the call spread with the highest PoP.

```
def get_highest_pop():
    maxpop = 0.0

    best_strategy = None
```

```
        for i in range(len(chain) - 1):
            for j in range(i + i, len(chain)):
                for k in (("sell", "buy"), ("buy", "sell")):
                    if k[0] == "sell":
                        premium = [chain[i][1], chain[j][2]]
                    else:
                        premium = [chain[i][2], chain[j][1]]

                    strategy = [
                        {
                            "type": "call",
                            "strike": chain[i][0],
                            "premium": premium[0],
                            "n": 100,
                            "action": k[0],
                        },
                        {
                            "type": "call",
                            "strike": chain[j][0],
                            "premium": premium[1],
                            "n": 100,
                            "action": k[1],
                        },
                    ]

                    inputs = Inputs(
                        stock_price=stockprice,
                        start_date=startdate,
                        target_date=targetdate,
                        volatility=volatility,
                        interest_rate=interestrate,
                        min_stock=minstock,
                        max_stock=maxstock,
                        strategy=strategy,
                    )

                    out = run_strategy(inputs)

                    if maxpop < out.probability_of_profit:
                        maxpop = out.probability_of_profit
                        best_strategy = strategy

        return best_strategy
```

If you analyze this call spread with the highest PoP afterward, you will find an astonishingly high PoP of 99.1%. Have we discovered the Holy Grail of options trading? Well, let's take it slow here.

This bull call spread consists of buying 100 calls with a strike price of $145.00 (very ITM, indeed) at $198.05 each and selling 100 calls with a strike price of $305.00 (less ITM than the bought leg) at $38.10 each. This would result in a net debit of $15,995.00 in your brokerage account, which is also the maximum loss, for a maximum profit of... $5.00.



Profit/loss diagram of the highest PoP call spread.

This disapointing outcome suggests that *unconstrained maximization of the PoP alone is not enough to achieve a winning combination of sold and bought calls with a favorable return for the trader*.

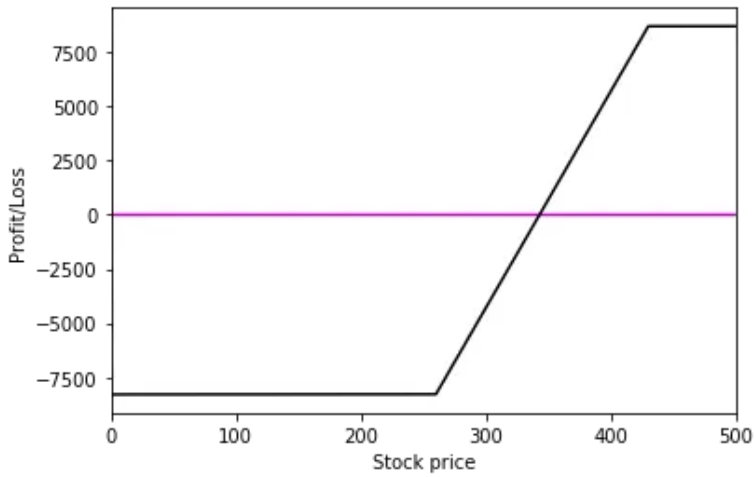
The extremely low maximum profit/maximum loss ratio of the previously found call spread (1:3199) and its very high cost, despite the PoP being above 99%, definitely do not make it very appealing in practice. Another possibility is to set a threshold for the maximum profit/maximum loss ratio that the trader deems acceptable and then search for a combination of calls that satisfies this *constraint*.

Let's assume that the trader is very risk averse and requires that the maximum profit/maximum loss ratio be, at worst, 1:1. Next, we change the function `get_highest_pop()` to take this constraint into account.

```
def get_highest_pop():
    maxpop = 0.0

    best_strategy = None
    for i in range(len(chain) - 1):
        for j in range(i + i, len(chain)):
            for k in (("sell", "buy"), ("buy", "sell")):
                if k[0] == "sell":
                    premium = [chain[i][1], chain[j][2]]
                else:
                    premium = [chain[i][2], chain[j][1]]

                strategy = [
                    {
                        "type": "call",
```

```
                        "strike": chain[i][0],
                        "premium": premium[0],
                        "n": 100,
                        "action": k[0],
                    },
                    {
                        "type": "call",
                        "strike": chain[j][0],
                        "premium": premium[1],
                        "n": 100,
                        "action": k[1],
                    },
                ]

                inputs = Inputs(
                    stock_price=stockprice,
                    start_date=startdate,
                    target_date=targetdate,
                    volatility=volatility,
                    interest_rate=interestrate,
                    min_stock=minstock,
                    max_stock=maxstock,
                    strategy=strategy,
                )

                out = run_strategy(inputs)

                if out.return_in_the_domain_ratio >= 1.0:
                    if maxpop < out.probability_of_profit:
                        maxpop = out.probability_of_profit
                        best_strategy = strategy

    return best_strategy
```

With this constraint, the (bull) call spread with the highest PoP of 49% is now constructed as follows: we should buy 100 ITM calls with a strike of $260.00 at $83.00 each and sell 100 OTM calls with a strike of $430.00 at $0.07 each. This strategy incurs a total cost of $8,293.00 (which is also the maximum loss) while it offers a maximum profit of $8,707.00, thus resulting in a maximum profit/maximum loss ratio of about 1:1.



Profit/loss diagram of the highest PoP call spread with a maximum profit/maximum loss ratio of at least 1:1.

The two strategies shown above illustrate quite well the trade-off that always exists between maximum return and the probability of profit in an options strategy. These are two objectives that typically move in opposite directions: the higher the probability of profit, the lower the maximum profit; if the maximum profit increases, it becomes less likely for the strategy to end in profit.

The type of analysis presented in this article indicates a pathway for fine-tuning the trader's return expectations and the maximum they are willing to take.

Options Trading | Options Strategy | Call Spread | Python

Follow

Written by Roberto Gomes, PhD

903 Followers · 14 Following

I am a Professor of Materials Engineering specialized in computational research in materials science, physics, and engineering. Also interested in finance.

Responses (8)

What are your thoughts?

Respond

Gerardo Gomez
Dec 7, 2023

" This strategy incurs a total cost of $8,293.00 (which is also the maximum loss) while it offers a maximum profit of $8,707.00, thus resulting in a maximum profit/maximum loss ratio of about 1:1. "

Even if this a hypothetical assumption. Why would a... more

2 1 reply Reply

Наталья Андриец
Feb 17, 2024 (edited)

To find out a probability of profit we should place PDF (probability density function) on the PL diagram centered at the current underlying price and calculate the space under our model (or empirical?) PDF function and the spread payoff function... more

1 1 reply Reply

D

lbmderekfung
Aug 8, 2023

Can you please share the whole project in Github? Thank you!

1 2 replies Reply

See all responses

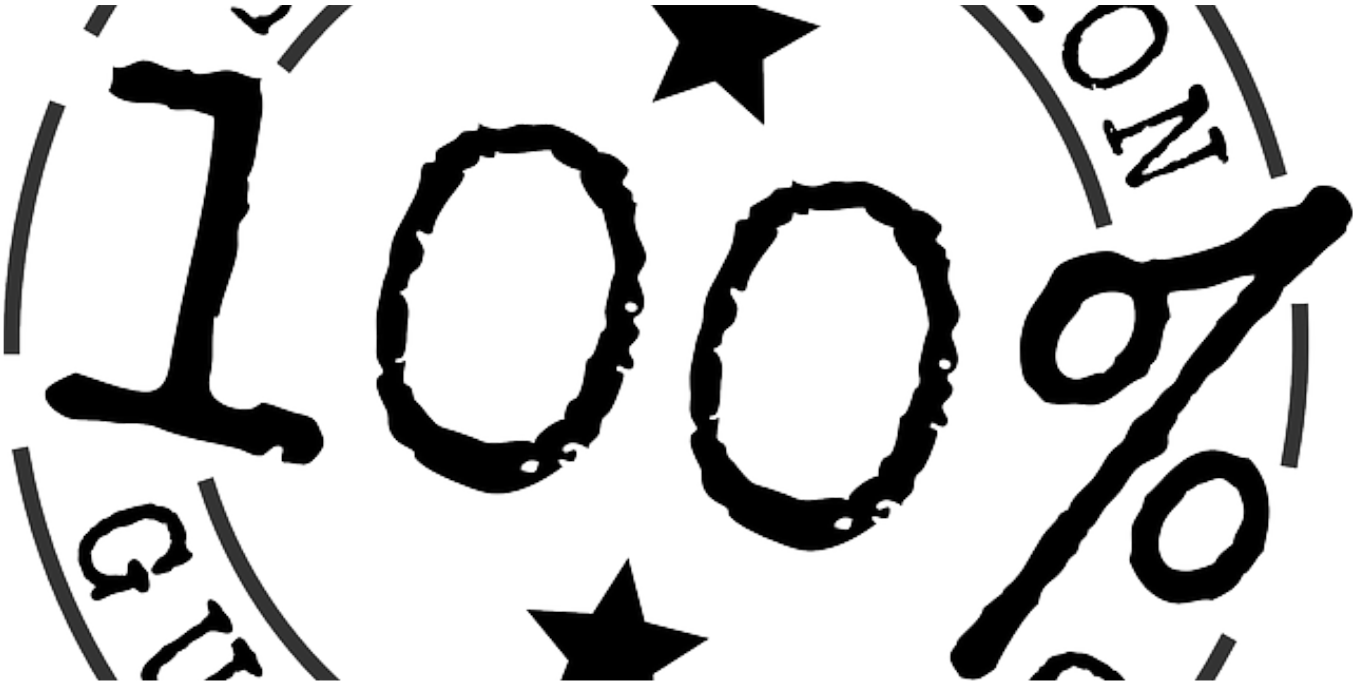
More from Roberto Gomes, PhD

Roberto Gomes, PhD

Is Delta the same as In-The-Money probability?

Using OptionLab calculations to address this question

Dec 13, 2023 62 2



Roberto Gomes, PhD

Python for Options Trading (3): A Trade with 100% Probability of Profit

NOTE: The library has undergone significant changes, rendering the code from the previous version of this article incompatible. As a...

Oct 30, 2023 590 29

Roberto Gomes, PhD

Probability of profit of an options strategy from the Black-Scholes model

Very important in the evaluation of an options strategy, but often neglected or poorly determined, the Probability of Profit (PoP) is the...

Jul 24, 2023 29 1

Roberto Gomes, PhD

Python for Options Trading (5): Reducing risk on a losing long call

NOTE: The library has undergone significant changes, rendering the code from the previous version of this article incompatible. As a...

Jan 15, 2024 80 3

See all from Roberto Gomes, PhD

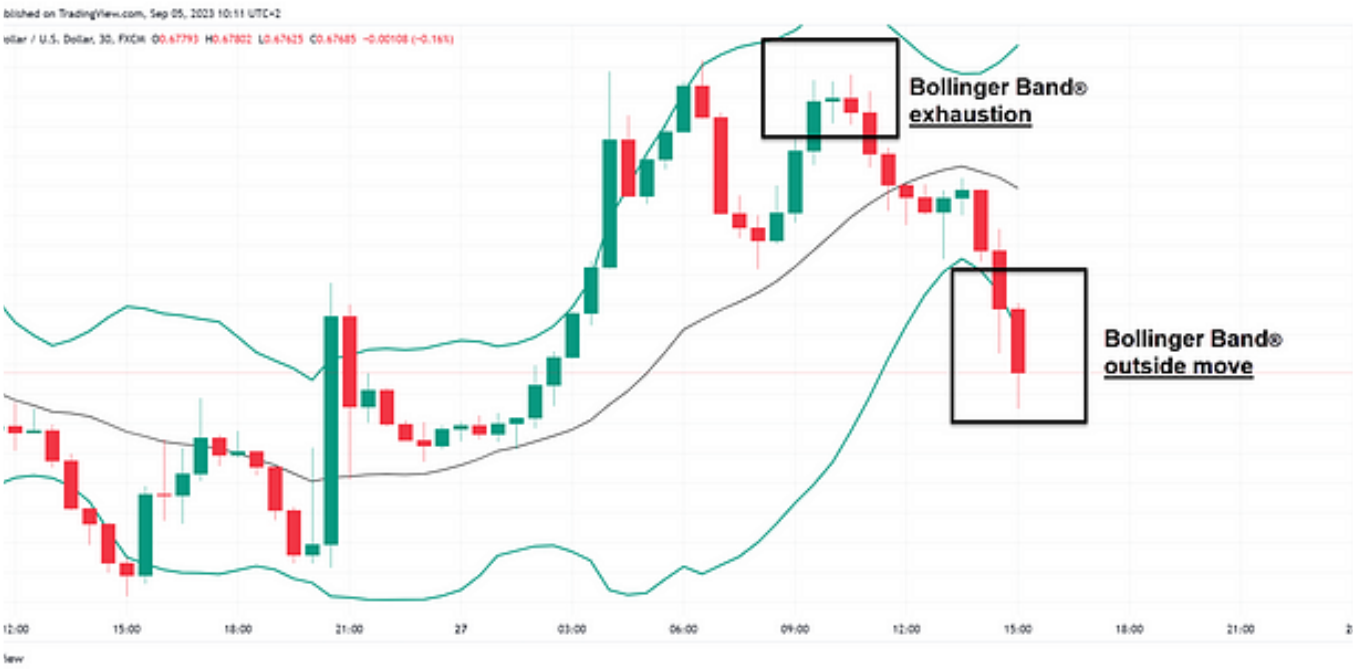
Recommended from Medium

Unicorn Day

A Simple Yet Powerful Trading Strategy: The Moving Average Slope Method 📈

Ever wondered how the pros know when to enter and exit the market? What if I told you there's a way to ride trends with mathematical...

6d ago 225 3



In InsiderFinance Wire by Huzaifa Zahoor

How to calculate Bollinger Bands in Python

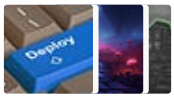
Ready to explore another powerful technical indicator? Let's dive into Bollinger Bands and learn how to calculate them using Python. This...

Aug 26, 2024 119

Lists

Coding & Development

11 stories · 1002 saves



Predictive Modeling w/ Python

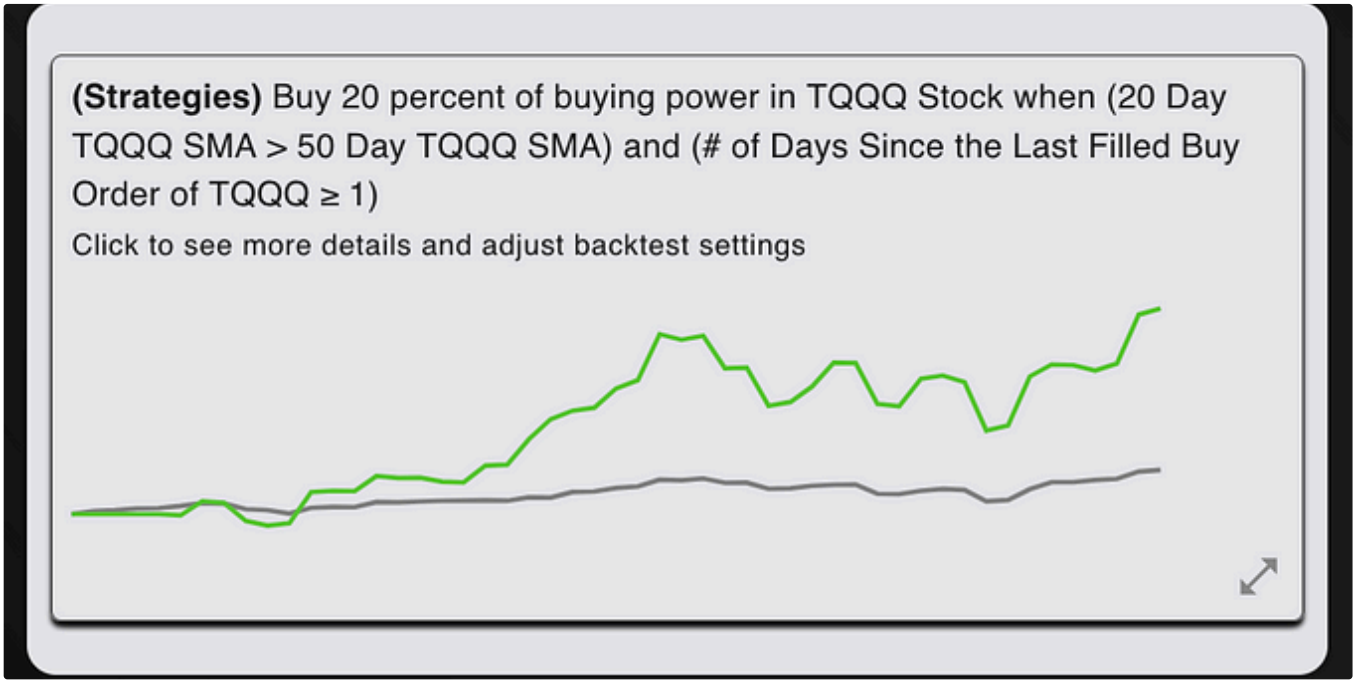
20 stories · 1822 saves

Practical Guides to Machine Learning

10 stories · 2194 saves

ChatGPT

21 stories · 959 saves

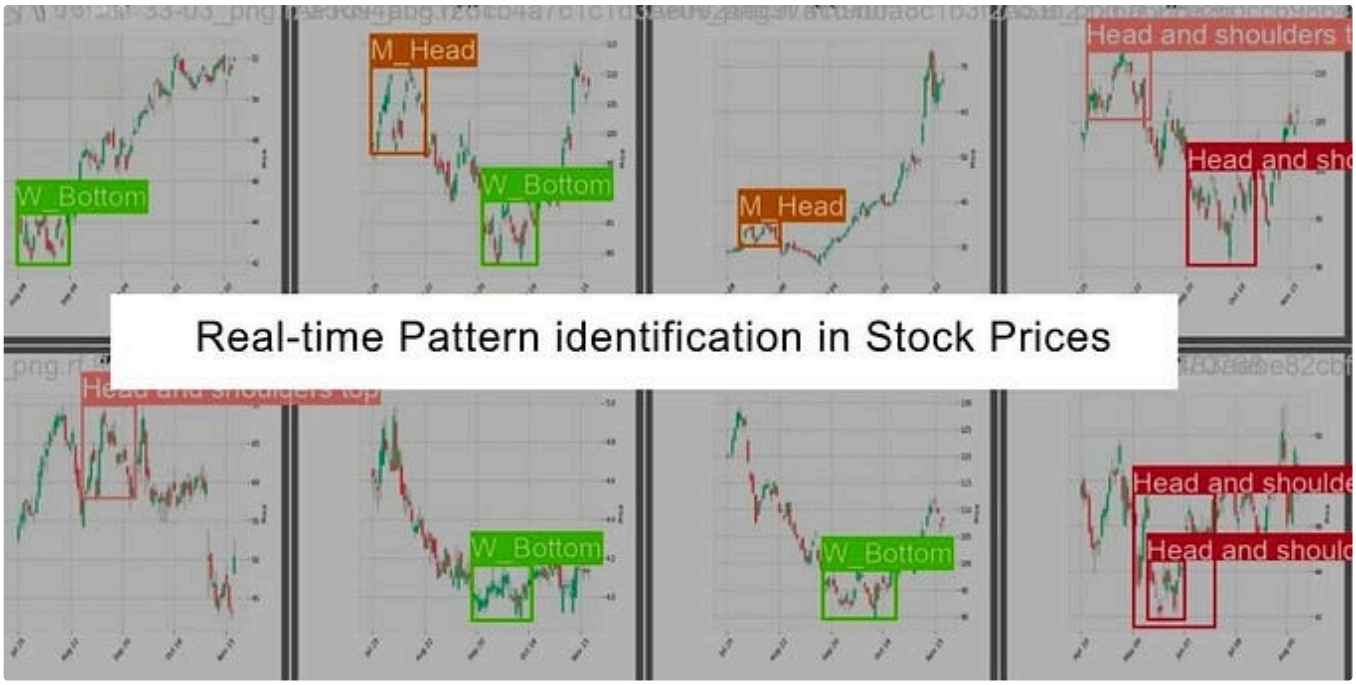


In DataDrivenInvestor by Austin Starks

I used OpenAI's brand new O3-mini model to create a trading strategy. It's DESTROYING the market

You can copy this strategy for yourself in a single click

Feb 1 338 13

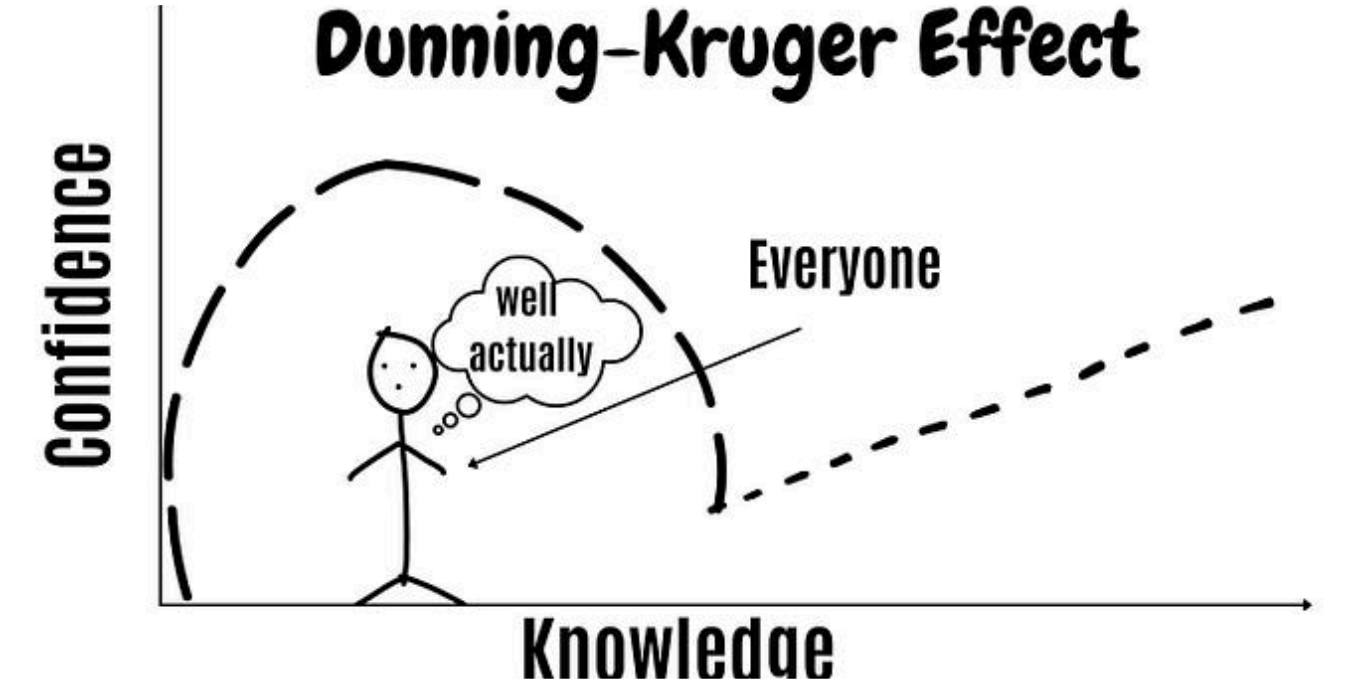


Cristian Velasquez

Candlestick Pattern Recognition with YOLO

Implementation of a Fine-tuned YOLO Model for Stock Price Patterns Identification. End-to-end Notebook Provided.

Jan 30 119 2



In Making of a Millionaire by Ben Le Fort

Confidently Wrong: How the Dunning-Kruger Effect Impacts Your Money

“The first rule of the Dunning-Kruger club is you don’t know you’re a member of the Dunning-Kruger club”

Mar 15, 2023 2.4K 47

Sayedali

The Secret Trading Strategy That's Turning Beginners into Pros

A Step-by-Step Guide to This High-Profit Indicator Setup

Jan 12 64 2

See more recommendations