Member-only story

Implementing Grid Indicator using Backtrader

JohnJoy · Follow

3 min read · Feb 16, 2023

The Grid Indicator is a technical analysis tool that is used to identify key price levels on a chart. It is based on the concept of price ranges, which are defined as a percentage of the current price. The Grid Indicator plots a series of horizontal lines on the chart, each representing a different price range. These lines can be used as potential support or resistance levels, and traders may use them to help identify areas where the price is likely to encounter buying or selling pressure. The Grid Indicator is a simple and easy-to-use tool that can be applied to any market and time frame.

here's an implementation of the grid indicator in Backtrader:

```
import backtrader as bt

class GridIndicator(bt.Indicator):
    params = (('price', 0.0), ('price_range', 0.5))

    def __init__(self):
        grid1up = self.p.price + ((self.p.price_range/2) * self.p.price / 100)
        grid1down = self.p.price - ((self.p.price_range/2) * self.p.price / 100
        grid2up = grid1up + ((self.p.price_range) * self.p.price / 100)
        grid2down = grid1down - ((self.p.price_range) * self.p.price / 100)
        grid3up = grid2up + ((self.p.price_range) * self.p.price / 100)
        grid3down = grid2down - ((self.p.price_range) * self.p.price / 100)
        grid4up = grid3up + ((self.p.price_range) * self.p.price / 100)
        grid4down = grid3down - ((self.p.price_range) * self.p.price / 100)
        grid5up = grid4up + ((self.p.price_range) * self.p.price / 100)
        grid5down = grid4down - ((self.p.price_range) * self.p.price / 100)

        self.lines.grid1up = bt.LinePlotterIndicator(self, plotmaster=self.data
        self.lines.grid1up.plot(grid1up, color='g')
        self.lines.grid1down = bt.LinePlotterIndicator(self, plotmaster=self.da
        self.lines.grid1down.plot(grid1down, color='r')
        self.lines.grid2up = bt.LinePlotterIndicator(self, plotmaster=self.data
        self.lines.grid2up.plot(grid2up, color='g')
        self.lines.grid2down = bt.LinePlotterIndicator(self, plotmaster=self.da
        self.lines.grid2down.plot(grid2down, color='r')
        self.lines.grid3up = bt.LinePlotterIndicator(self, plotmaster=self.data
        self.lines.grid3up.plot(grid3up, color='g')
        self.lines.grid3down = bt.LinePlotterIndicator(self, plotmaster=self.da
        self.lines.grid3down.plot(grid3down, color='r')
        self.lines.grid4up = bt.LinePlotterIndicator(self, plotmaster=self.data
        self.lines.grid4up.plot(grid4up, color='g')
        self.lines.grid4down = bt.LinePlotterIndicator(self, plotmaster=self.da
        self.lines.grid4down.plot(grid4down, color='r')
        self.lines.grid5up = bt.LinePlotterIndicator(self, plotmaster=self.data
        self.lines.grid5up.plot(grid5up, color='g')
        self.lines.grid5down = bt.LinePlotterIndicator(self, plotmaster=self.da
        self.lines.grid5down.plot(grid5down, color='r')
```

The code defines a custom indicator named `GridIndicator` that inherits from `bt.Indicator` class in the backtrader library.

The indicator has two parameters, `price` and `price_range`, with default values of 0.0 and 0.5 respectively. These parameters are used to calculate various grids which will

be plotted on the chart.

In the constructor of the class, various grid levels are calculated using the formulae provided in the original indicator. These levels are assigned to variables `grid1up`, `grid1down`, `grid2up`, `grid2down`, `grid3up`, `grid3down`, `grid4up`, `grid4down`, `grid5up`, and `grid5down`.

The grid levels are plotted using the `bt.LinePlotterIndicator` class, which creates a line plot of a single value for each grid level. The `plot()` method is used to plot each grid level, with the `color` parameter specifying the color of the plot. The `plotmaster` parameter specifies the data line that the indicator should be plotted on.

After plotting each grid level, the lines are assigned to various `self.lines` attributes named `grid1up`, `grid1down`, `grid2up`, `grid2down`, `grid3up`, `grid3down`, `grid4up`, `grid4down`, `grid5up`, and `grid5down`. These attributes are defined by the `bt.Indicator` class and store the plot data for each line.

Thank you for reading this article! If you're interested in more content like this, please follow me on Medium

Follow

Written by JohnJoy

248 Followers

Finance, Programming

More from JohnJoy

JohnJoy

Improve Performance:Backtrader Hacks

Backtrader's cerebro engine provides a number of ways to optimize the performance of your backtesting process. Optimization can be...

3 min read · Jan 10, 2023

JohnJoy

Implementing Bollinger Band strategy with trailing Stoploss in Backtrader

Bollinger Bands are a popular technical indicator used in finance to measure volatility. They consist of a moving average and two standard...

3 min read · Jan 15, 2023

31

JohnJoy

Portfolio Rebalancing: Using Backtrader

Portfolio rebalancing is a strategy used by investors to maintain a desired level of asset allocation in their portfolio. It involves...

4 min read · Jan 8, 2023

29 1

JohnJoy in Quant Factory

How to Make Simple Futures Trading Algorithm using QuantConnect

Buy and Hold a Single S&P Emini Contract

6 min read · Jun 18, 2023

64

See all from JohnJoy

Recommended from Medium

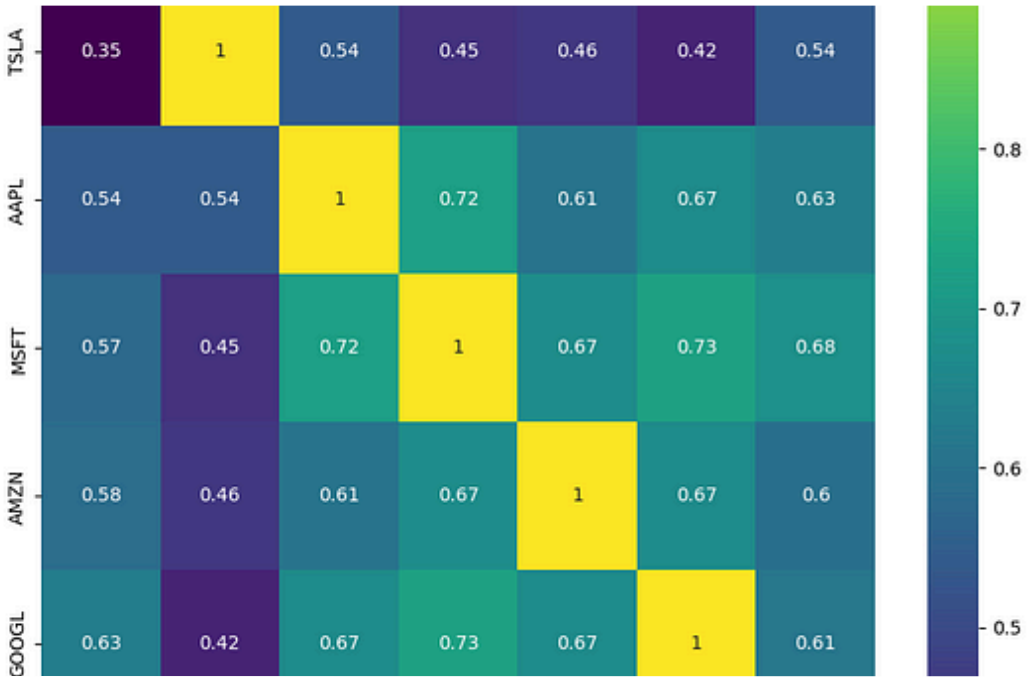
Fateh Mehmood

Revolutionizing Forex Trading: A Comprehensive Overview of AI-Driven Back-testing System, Web...

I recently did my end semester project of Artificial Intelligence in which I made a forex Back-testing system, Web Dashboard and Smart Bot...

7 min read · May 21, 2024

2



Jakub Polec

Portfolio optimization: from the highest Sharpe Ratio to minimum volatility

In this article, you will learn about the following:

✦ · 5 min read · Jan 30, 2024

63 1

Lists

Staff Picks
653 stories · 1009 saves

Stories to Help You Level-Up at Work
19 stories · 628 saves

Self-Improvement 101
20 stories · 1933 saves

Productivity 101
20 stories · 1753 saves

```
ticker = "GOOGL"
df = yf.download(ticker, start="2021-01-02", end="2022-1-1")
df.head()
```

✓ 0.4s

```
[**********100%**********]  1 of 1 completed
```

Date	Open	High	Low	Close	Adj Close	Volume
2021-01-04	88.000000	88.124496	85.357002	86.306503	86.306503	37324000
2021-01-05	86.254501	87.341499	85.845001	87.002502	87.002502	20360000
2021-01-06	85.013000	87.198502	84.805000	86.143997	86.143997	46588000
2021-01-07	86.337997	88.890999	86.337997	88.717003	88.717003	41936000
2021-01-08	88.858002	89.968002	88.060997	89.891502	89.891502	35484000

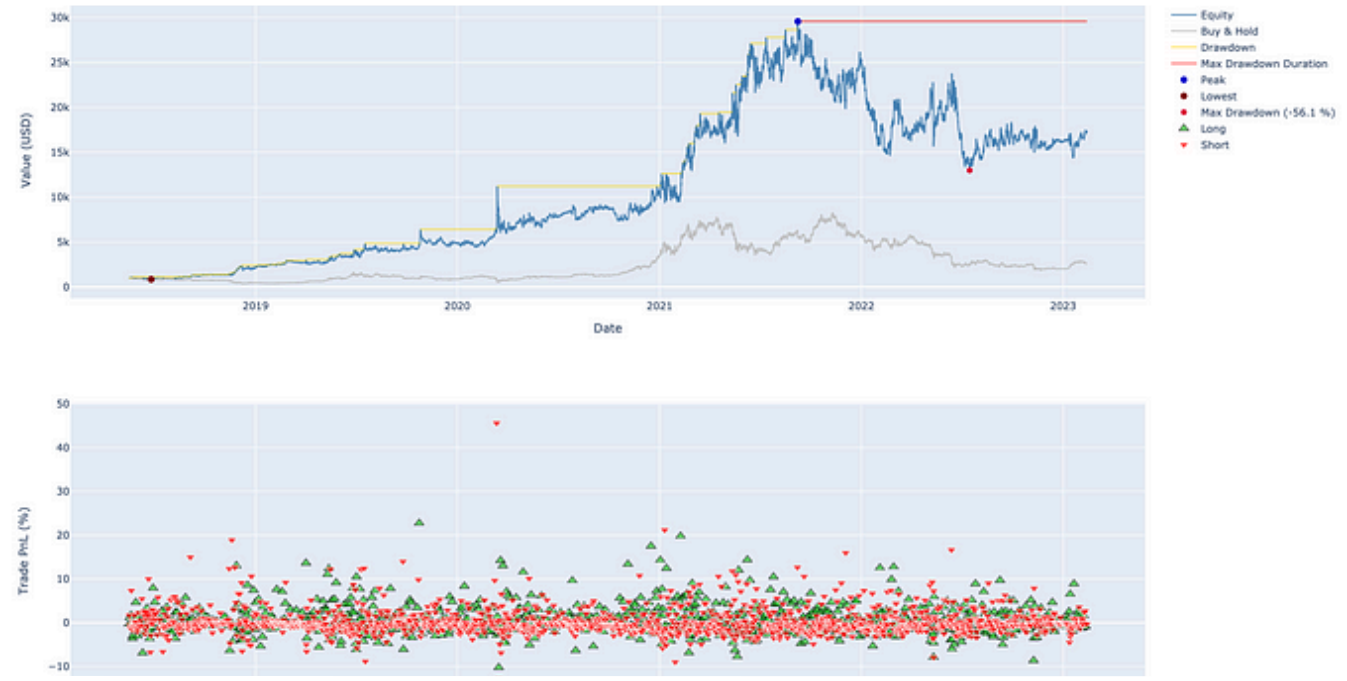
SR

Simple Moving Average Crossover Trading Strategy with Python and Yahoo Finance Data

Let’s break down the provided code part by part:

4 min read · Dec 27, 2023

20 2



Diogo Matos Chaves

How to Implement a Backtester in Python

And which library to use if you don’t want to code one yourself

8 min read · Feb 12, 2024

129 2

Armando Aguilar L.

MACD with Python

The world of technical analysis in trading can seem daunting for beginners, but fear not—we're here to unravel the mystery behind one of...

4 min read · Jan 7, 2024

2

Kaushik Jegannathan

Generating Swing Trading Signals using Donchian Strategy in Python

Swing trading aims to capture short-term market movements, holding positions for days or weeks rather than hours or minutes. Swing trading...

6 min read · Dec 25, 2023

150 2