Member-only story

Beyond Trends: Can Volatility Swings Predict Profits?

PyQuantLab Following 10 min read · Jun 6, 2025

In the dynamic world of financial markets, understanding and reacting to changing volatility is a cornerstone of many trading strategies. This article delves into an intriguing concept: volatility ratio reversion. We'll explore a simplified strategy that leverages the relationship between short-term and long-term volatility, using the powerful Backtrader framework for robust backtesting and analysis.

Looking to supercharge your algorithmic trading research? The Ultimate Algorithmic Strategy Bundle has you covered with over

80 Python strategies, fully documented in comprehensive PDF manuals:

Ultimate Algorithmic Strategy Bundle

The Core Idea: Volatility Reversion

Volatility, a measure of price fluctuations, is not constant. Periods of high volatility are often followed by periods of lower volatility, and vice versa. This tendency for volatility to revert to its mean is a key principle our strategy aims to exploit.

At the heart of our approach is the Volatility Ratio. This indicator is simply the ratio of short-term volatility to long-term volatility.

- When the volatility ratio is low, it suggests that short-term price movements are much calmer than what the market has experienced over a longer period. This could indicate a calm before a storm, or a consolidation phase, potentially leading to an upward movement as volatility increases.
- Conversely, when the volatility ratio is high, it implies that the market is experiencing significantly more turbulence in the short term compared to its longer-term average. This often occurs during periods of fear, uncertainty, or rapid price changes, which can sometimes be followed by a calming down and a potential downward correction or consolidation.

Enhancing the Signal: The Trend Filter

Trading solely on volatility reversion can be risky, especially in strong trending markets. A strategy might try to "revert" against a powerful trend, leading to losses. To mitigate this, we introduce a trend filter using a Simple Moving Average (SMA).

- We'll only consider long positions when the volatility ratio is low *and* the current price is above its long-term SMA, confirming an existing uptrend.
- Similarly, we'll only consider short positions when the volatility ratio is high *and* the current price is below its long-term SMA, aligning with a downtrend.

This combination aims to capture reversion opportunities that are "in line" with the broader market direction, potentially leading to more favorable trades.

Risk Management: ATR Trailing Stops

No strategy is complete without robust risk management. For our exits, we employ an Average True Range (ATR) trailing stop. ATR is a measure of market volatility, and using it to set stops means our stop-loss levels adapt to current market conditions:

- In volatile markets, the ATR will be higher, resulting in wider stops, giving the trade more room to breathe.

- In calmer markets, the ATR will be lower, leading to tighter stops, protecting profits more closely.

The trailing nature of the stop ensures that as a profitable trade moves in our favor, the stop-loss also moves, locking in gains.

Strategy Implementation with Backtrader

Let's dive into the Python code to implement this strategy using `backtrader`, a powerful and flexible backtesting framework.

First, we define our custom `VolatilityRatioIndicator`:

```
import backtrader as bt
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings("ignore")

# Ensure matplotlib plots are shown inline in Jupyter/IPython envir
# %matplotlib inline

class VolatilityRatioIndicator(bt.Indicator):
    """Calculate volatility ratio (short-term vol / long-term vol)"

    lines = ('vol_ratio', 'trend_sma')

    params = (
        ('short_vol_window', 7),
        ('long_vol_window', 30),
```

```
        ('trend_sma_window', 50),
    )

    def __init__(self):
        # Daily returns (manual calculation for Backtrader)
        daily_returns = (self.data.close - self.data.close(-1)) / s

        # Rolling volatilities using standard deviation of returns
        short_vol = bt.indicators.StdDev(daily_returns, period=self
        long_vol = bt.indicators.StdDev(daily_returns, period=self.

        # Volatility ratio: short-term volatility relative to long-
        self.lines.vol_ratio = short_vol / long_vol

        # Trend filter: Simple Moving Average of closing price
        self.lines.trend_sma = bt.indicators.SMA(self.data.close, p
```

Next, we build the `VolatilityRatioStrategy` itself, incorporating the indicator, entry/exit logic, and risk management:

```
class VolatilityRatioStrategy(bt.Strategy):
    """
    Volatility Ratio Reversion Strategy:
    - Long when vol ratio < lower_threshold AND price > trend
    - Short when vol ratio > upper_threshold AND price < trend
    - ATR trailing stops for exits
    """

    params = (
        # Volatility parameters
        ('short_vol_window', 7),
        ('long_vol_window', 30),
        ('upper_threshold', 1.2), # Volatility is high, anticipate
        ('lower_threshold', 0.8), # Volatility is low, anticipate r
        ('trend_sma_window', 50),
```

```
        # Risk management
        ('atr_period', 14),
        ('atr_multiplier', 1.0), # Distance of stop from price, in
        ('position_size', 0.95), # Percentage of cash to use per tr

        # Output
        ('printlog', True),
    )

    def __init__(self):
        # Initialize our custom Volatility Ratio Indicator
        self.vol_ratio = VolatilityRatioIndicator(
            short_vol_window=self.params.short_vol_window,
            long_vol_window=self.params.long_vol_window,
            trend_sma_window=self.params.trend_sma_window
        )
        # Initialize ATR for trailing stops
        self.atr = bt.indicators.ATR(period=self.params.atr_period)

        # To keep track of pending orders and trade details
        self.entry_price = None
        self.trailing_stop = None
        self.order = None

        # Performance tracking
        self.trade_count = 0
        self.winning_trades = 0

    def log(self, txt, dt=None):
        """Logging function for strategy"""
        if self.params.printlog:
            dt = dt or self.datas[0].datetime.date(0)
            print(f'{dt}: {txt}')

    def notify_order(self, order):
        """Handles notifications for orders (submitted, accepted, c
        if order.status in [order.Completed]:
            if order.isbuy():
                self.log(f'BUY EXECUTED: Price ${order.executed.pri
            else:
                self.log(f'SELL EXECUTED: Price ${order.executed.pr
        elif order.status in [order.Canceled, order.Margin, order.R
            self.log(f'Order Canceled/Margin/Rejected: {order.statu
        self.order = None # Clear pending order
```

```
    def notify_trade(self, trade):
        """Handles notifications for trades (open, closed, profit/l
        if trade.isclosed:
            self.trade_count += 1
            if trade.pnl > 0:
                self.winning_trades += 1

            win_rate = (self.winning_trades / self.trade_count) * 1
            self.log(f'TRADE CLOSED: PnL ${trade.pnl:.2f}, Commissi

    def next(self):
        """Main strategy logic executed on each new bar (day)"""
        # Skip if insufficient data for indicators or a pending ord
        if (self.order or
            len(self.data) < max(self.params.long_vol_window, self.
            np.isnan(self.vol_ratio.vol_ratio[0]) or
            np.isnan(self.atr[0])):
            return

        current_price = self.data.close[0]
        vol_ratio = self.vol_ratio.vol_ratio[0]
        trend_sma = self.vol_ratio.trend_sma[0]

        # Update trailing stop if in position
        if self.position:
            self._update_trailing_stop()
            if self._check_stop_exit():
                # If a stop exit occurred, we're out of the market,
                return

        # Generate signals based on volatility ratio and trend filt
        signal = 0

        # Long signal: Low volatility ratio AND price above trend S
        if vol_ratio < self.params.lower_threshold and current_pric
            signal = 1

        # Short signal: High volatility ratio AND price below trend
        elif vol_ratio > self.params.upper_threshold and current_pr
            signal = -1

        # Execute signals if not already in a position
        if signal == 1 and not self.position:
```

```
            self._enter_long()
        elif signal == -1 and not self.position:
            self._enter_short()

    def _enter_long(self):
        """Places a buy order and sets initial trailing stop"""
        # Calculate size based on position_size parameter and avail
        # Note: Backtrader's PercentSizer will handle the actual si
        # This manual calculation is for logging/initial stop setti
        size = int(self.broker.getcash() * self.params.position_siz
        if size > 0:
            self.order = self.buy(size=size)
            self.entry_price = self.data.close[0]
            # Initial trailing stop for long position: entry_price
            self.trailing_stop = self.entry_price - (self.params.at
            self.log(f'LONG SIGNAL: Vol Ratio {self.vol_ratio.vol_r

    def _enter_short(self):
        """Places a sell (short) order and sets initial trailing st
        size = int(self.broker.getcash() * self.params.position_siz
        if size > 0:
            self.order = self.sell(size=size) # For shorting, this
            self.entry_price = self.data.close[0]
            # Initial trailing stop for short position: entry_price
            self.trailing_stop = self.entry_price + (self.params.at
            self.log(f'SHORT SIGNAL: Vol Ratio {self.vol_ratio.vol_

    def _update_trailing_stop(self):
        """Adjusts the trailing stop upwards for long, downwards fo
        if self.trailing_stop is None:
            return

        current_price = self.data.close[0]
        atr_value = self.atr[0]

        if self.position.size > 0:  # Long position
            new_stop = current_price - (self.params.atr_multiplier
            if new_stop > self.trailing_stop: # Only move stop up
                self.trailing_stop = new_stop
        else:  # Short position
            new_stop = current_price + (self.params.atr_multiplier
            if new_stop < self.trailing_stop: # Only move stop down
                self.trailing_stop = new_stop
```

```
    def _check_stop_exit(self):
        """Checks if price has hit the trailing stop and closes pos
        if self.trailing_stop is None:
            return False

        # For long position, exit if low crosses below trailing sto
        if self.position.size > 0:  # Long
            if self.data.low[0] <= self.trailing_stop:
                self.order = self.close() # Close current long posi
                self.log(f'STOP EXIT (LONG): Trailing Stop @ ${self
                self._reset_position()
                return True
        # For short position, exit if high crosses above trailing s
        else:  # Short
            if self.data.high[0] >= self.trailing_stop:
                self.order = self.close() # Close current short pos
                self.log(f'STOP EXIT (SHORT): Trailing Stop @ ${sel
                self._reset_position()
                return True
        return False

    def _reset_position(self):
        """Resets position tracking variables after a trade closure
        self.entry_price = None
        self.trailing_stop = None

    def stop(self):
        """Called at the end of the backtest to print final summary
        final_value = self.broker.getvalue()
        win_rate = (self.winning_trades / self.trade_count * 100) i

        print('='*50)
        print('STRATEGY RESULTS')
        print('='*50)
        print(f'Final Portfolio Value: ${final_value:,.2f}')
        print(f'Total Trades Executed: {self.trade_count}')
        print(f'Winning Trades: {self.winning_trades}')
        print(f'Win Rate: {win_rate:.1f}%')
        print('='*50)
```

Finally, we set up a `run_backtest` function to encapsulate the backtesting process, allowing easy configuration and execution:

```
def run_backtest(
    # Data Parameters
    ticker="BTC-USD",
    start_date="2021-01-01",
    end_date="2024-12-31",
    initial_cash=100000,
    commission=0.001,

    # Strategy Parameters
    short_vol_window=7,
    long_vol_window=30,
    upper_threshold=1.2,
    lower_threshold=0.8,
    trend_sma_window=50,
    atr_period=14,
    atr_multiplier=1.0,
    position_size=0.95,

    # Output Parameters
    printlog=True,
    show_plot=True
):
    """
    Run the volatility ratio strategy backtest with configurable pa
    """

    print(f"Volatility Ratio Strategy Backtest")
    print(f"=" * 50)
    print(f"Asset: {ticker}")
    print(f"Period: {start_date} to {end_date}")
    print(f"Initial Cash: ${initial_cash:,}")
    print(f"Commission: {commission*100:.2f}%")
    print(f"Volatility Windows: {short_vol_window}d / {long_vol_win
    print(f"Thresholds: Long < {lower_threshold}, Short > {upper_th
    print(f"Trend Filter: SMA {trend_sma_window}")
    print(f"ATR Stop: {atr_multiplier}x ATR({atr_period})")
```

```
print(f"Position Size: {position_size*100:.0f}% of cash")
print("-" * 50)

# Download data using yfinance. As per instructions, using auto
print("Downloading data...")
df = yf.download(ticker, start=start_date, end=end_date, auto_a

if df.empty:
    print(f"No data for {ticker} in the specified period.")
    return None, None

# Ensure standard OHLCV column names for Backtrader
df = df[['Open', 'High', 'Low', 'Close', 'Volume']].copy()
print(f"Downloaded {len(df)} bars.")

# Setup Cerebro (the "brain" of Backtrader)
cerebro = bt.Cerebro()

# Add data to Cerebro
data = bt.feeds.PandasData(dataname=df)
cerebro.adddata(data)

# Add the strategy with its configurable parameters
cerebro.addstrategy(
    VolatilityRatioStrategy,
    short_vol_window=short_vol_window,
    long_vol_window=long_vol_window,
    upper_threshold=upper_threshold,
    lower_threshold=lower_threshold,
    trend_sma_window=trend_sma_window,
    atr_period=atr_period,
    atr_multiplier=atr_multiplier,
    position_size=position_size,
    printlog=printlog
)

# Setup broker settings
cerebro.broker.setcash(initial_cash)
cerebro.broker.setcommission(commission=commission)

# Set position sizing (e.g., using 95% of available cash for ea
cerebro.addsizer(bt.sizers.PercentSizer, percents=position_size

# Add analyzers for performance metrics
```

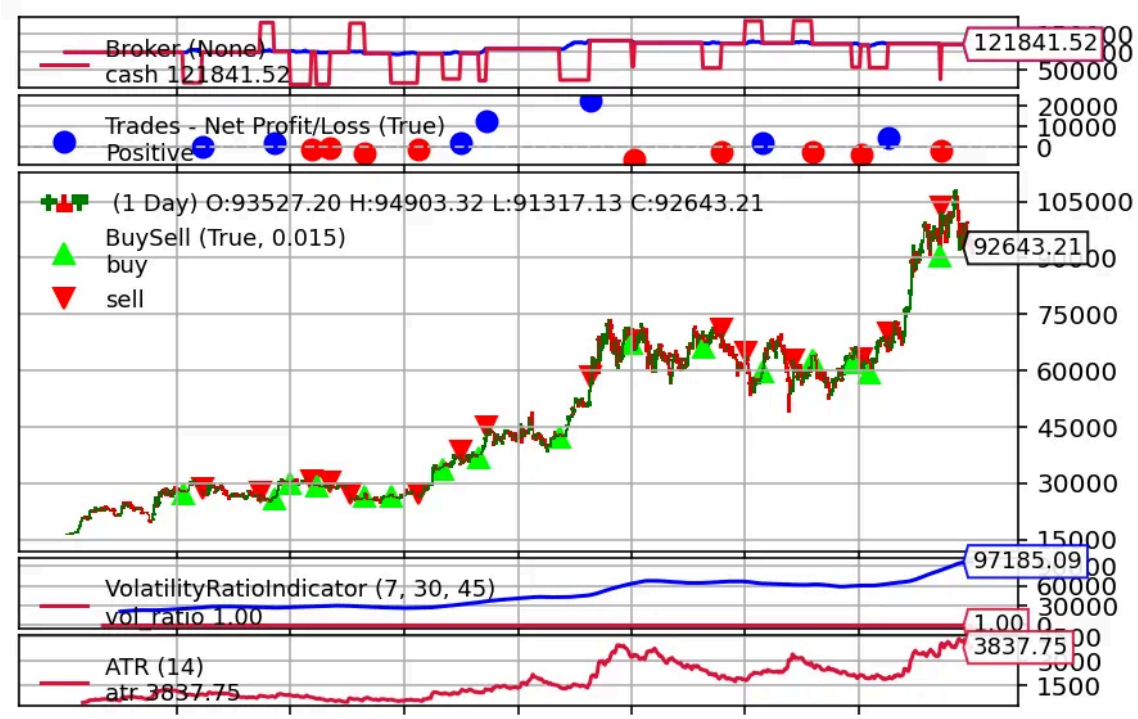
```
cerebro.addanalyzer(bt.analyzers.SharpeRatio, _name='sharpe', t
cerebro.addanalyzer(bt.analyzers.DrawDown, _name='drawdown')
cerebro.addanalyzer(bt.analyzers.TradeAnalyzer, _name='trades')
cerebro.addanalyzer(bt.analyzers.Returns, _name='returns')

# Run the backtest
print("\nRunning backtest...")
results = cerebro.run()
strategy = results[0] # Get the strategy instance from the resu

# Print comprehensive results
final_value = cerebro.broker.getvalue()
total_return = (final_value / initial_cash - 1) * 100

print(f"\nPERFORMANCE SUMMARY:")
print(f"Final Portfolio Value: ${final_value:,.2f}")
print(f"Initial Cash: ${initial_cash:,.2f}")
print(f"Total Return: {total_return:.2f}%")

# Sharpe ratio
sharpe_data = strategy.analyzers.sharpe.get_analysis()
sharpe_ratio = sharpe_data.get('sharperatio', 'N/A')
if sharpe_ratio != 'N/A':
    print(f"Sharpe Ratio (Daily): {sharpe_ratio:.2f}")

# Max drawdown
drawdown_data = strategy.analyzers.drawdown.get_analysis()
max_dd = drawdown_data.get('max', {}).get('drawdown', 0)
print(f"Max Drawdown: {max_dd:.2f}%")

# Trade stats
trades_data = strategy.analyzers.trades.get_analysis()
total_trades = trades_data.get('total', {}).get('total', 0)
print(f"Total Trades: {total_trades}")

if total_trades > 0:
    won_trades = trades_data.get('won', {}).get('total', 0)
    win_rate = (won_trades / total_trades) * 100
    print(f"Win Rate: {win_rate:.1f}%")

    # Optional: Print more detailed trade stats if available
    # trades_data['long']['total']
    # trades_data['short']['total']
    # trades_data['len']['average']
```

```
    # Buy & Hold comparison (benchmark)
    buy_hold_return = ((df['Close'].iloc[-1] / df['Close'].iloc[0])
    print(f"Buy & Hold Return ({ticker}): {buy_hold_return:.2f}%")
    print(f"Excess Return (vs. Buy & Hold): {total_return - buy_hol

    # Plot results
    if show_plot:
        print("\nGenerating charts (may take a moment)...")
        # Ensure plot shows custom indicators
        cerebro.plot(style='candlestick', volume=False, figsize=(16
        plt.suptitle(f'Volatility Ratio Strategy - {ticker} ({start
        plt.show()

    return cerebro, results


if __name__ == "__main__":
    """
    Main execution block to run the backtest with specific paramete
    """

    cerebro, results = run_backtest(
        ticker="BTC-USD",           # Asset to test (e.g., "BTC-USD"
        start_date="2023-01-01",    # Start of backtest period
        end_date="2024-06-01",      # End of backtest period (adjus
        upper_threshold=1.5,        # Higher threshold for short si
        lower_threshold=0.5,        # Lower threshold for long sign
        atr_multiplier=2.0,         # Wider stops (2x ATR)
        short_vol_window=10,        # Slightly longer short-term vo
        long_vol_window=40,         # Slightly longer long-term vol
        trend_sma_window=100,       # Longer-term trend filter
        printlog=True,              # Print trade logs
        show_plot=True              # Show equity curve and trades
    )
```

Running the Backtest

When you execute the `run_backtest` function, it will:

1. Download Historical Data: Fetches data for the specified `ticker` and date range. Here, we're using "BTC-USD" (Bitcoin to US Dollar), a highly volatile asset, from 2023-01-01 to 2024-06-01.
2. Initialize Backtrader: Sets up the trading environment with initial cash and commission.
3. Add Strategy and Analyzers: Integrates our `VolatilityRatioStrategy` and adds various `bt.analyzers` to provide detailed performance metrics (Sharpe Ratio, Drawdown, Trade Analysis, Returns).
4. Execute Backtest: Runs the simulation day by day, applying the strategy's logic.
5. Print Summary: Outputs key performance indicators like total return, Sharpe Ratio, max drawdown, and trade statistics. It also compares the strategy's return to a simple Buy & Hold benchmark.
6. Plot Results: Generates a visual representation of the equity curve, trades, and underlying price action.



Limitations and Further Exploration

This exploratory strategy, while robust in its implementation, has several areas for further investigation:

1. Parameter Optimization: The current parameters are chosen arbitrarily. Extensive optimization (e.g., using Backtrader's `optstrategy` or grid search) would be necessary to find combinations that yield better risk-adjusted returns across different assets and market conditions. However, beware of over-optimization (curve fitting) to historical data, which may not translate to future performance.
2. Asset Class Sensitivity: Volatility reversion might work better in certain asset classes (e.g., commodities, currencies) or

indices that are more mean-reverting, as opposed to growth-oriented stocks or cryptocurrencies that can experience long, strong trends.

3. Market Regimes: The strategy's performance can vary significantly in different market regimes (trending, range-bound, high volatility, low volatility). More advanced strategies often include logic to adapt to these regimes.
4. Exit Conditions: While ATR trailing stops are good, considering additional exit conditions (e.g., fixed profit targets, time-based exits, or a reversal of the volatility ratio signal) could improve performance.
5. Position Sizing: The current position sizing is a fixed percentage of cash. More sophisticated methods like fixed fractional position sizing (risk a fixed percentage of equity per trade) or Kelly Criterion could be explored.
6. Slippage and Real-World Costs: The backtest only accounts for commission. In live trading, slippage (the difference between expected and actual execution price) can significantly impact profitability, especially for assets like cryptocurrencies.

Conclusion

The "Volatility Ratio Reversion Strategy" provides a fascinating glimpse into leveraging volatility dynamics for trading. The

Backtrader implementation is clean, modular, and provides a solid foundation for backtesting.

While our initial backtest results on BTC-USD were not spectacular compared to a simple buy-and-hold, this is just one slice of data and one set of parameters. The true value of such an exploration lies in:

- Understanding the Mechanics: Grasping how indicators are built and how strategy logic is applied.
- Learning Backtesting Tools: Gaining proficiency with powerful frameworks like Backtrader.
- Formulating Hypotheses: Using initial results to refine the strategy, explore different parameters, or even pivot to entirely new ideas.

This strategy serves as an excellent starting point for further research and development in quantitative trading. The journey from a basic idea to a profitable, robust trading system is iterative, involving continuous testing, refinement, and a deep understanding of market behavior.

Python | Algorithmic Trading | Quantitative Finance | Backtrader | Volatility Trading

Written by PyQuantLab

655 followers · 6 following

Following

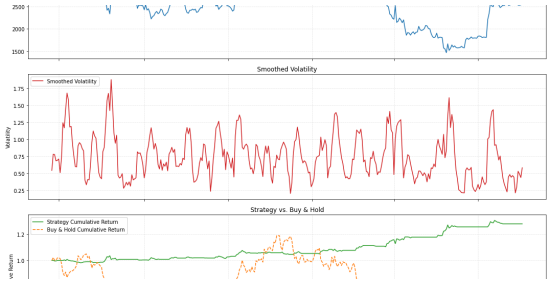
Your go-to place for Python-based quant tutorials, strategy deep-dives, and reproducible code. For more visit our website: www.pyquantlab.com

No responses yet

Steven Feng CAI

What are your thoughts?

More from PyQuantLab



PyQuantLab

Volatility Clustering Trading Strategy with Python

Ultimate Algorithmic Strategy Bundle has you covered with over 80 Python...

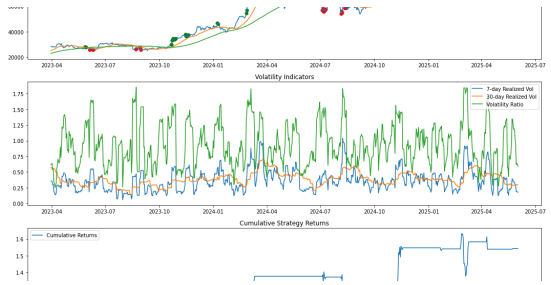
Jun 3 32 3

PyQuantLab

An Algorithmic Exploration of Volume Spread Analysis...

Note: You can read all articles for free on our website: pyquantlab.com

Jun 9 54 1

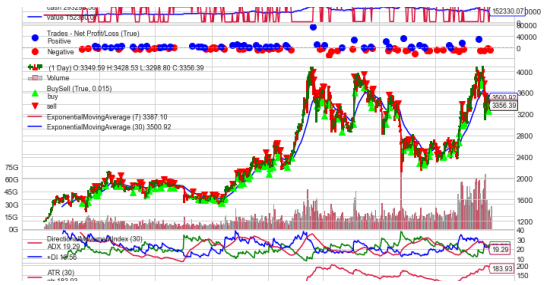


PyQuantLab

Trend-Volatility Confluence Trading Strategy

The Ultimate Algorithmic Strategy Bundle has you covered with over 80...

Jun 3 60



PyQuantLab

Building an Adaptive Trading Strategy with Backtrader: A...

Note: You can read all articles for free on our website: pyquantlab.com

Jun 4 64

See all from PyQuantLab

Recommended from Medium

Swapnilphutane

How I Built a Multi-Market Trading Strategy That Pass...

When I first got into trading, I had no plans of building a full-blown system....

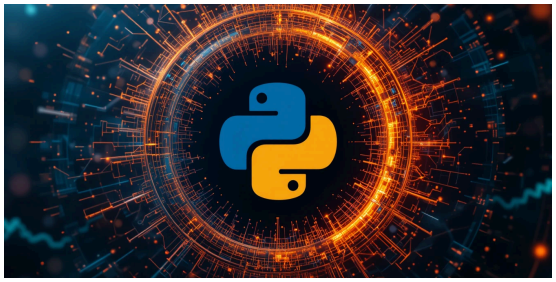
6d ago 16 1

Candence

Exposing Bernd Skorupinski Strategy: How I Profited ove...

I executed a single Nikkei Futures trade that banked $16,400 with a...

Jun 19 2



Unicorn Day

The Quest for the Perfect Trading Score: Turning...

Navigating the financial markets... it feels like being hit by a tsunami of da...

3d ago 43

MarketMuse

“I Let an AI Bot Trade for Me for 7 Days—It Made $8,000...

Subtitle: While you’re analyzing candlestick patterns, AI bots are fron...

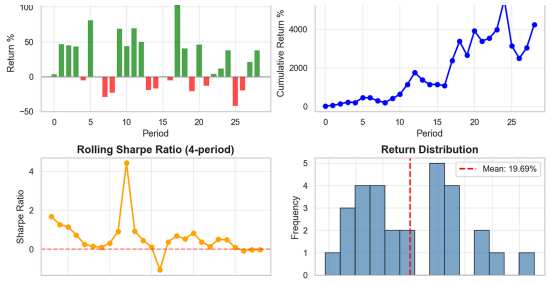
Jun 3 75 3

Remedy

How I Turned My Trading Strategy into a Professional...

By Chinedu Uzochukwu

6d ago 1 1



PyQuantLab

Trend Following with Kalman Filter and Trailing Stops: A...

This article explores a quantitative trading strategy built using the...

Jun 20 6

See more recommendations