Open in app ↗

Medium Search Write

Member-only story

An Ornstein-Uhlenbeck Mean-Reversion Strategy with Python and Backtrader

PyQuantLab Following 9 min read · Jun 7, 2025

10

In the dynamic world of financial markets, prices often exhibit seemingly random walks. However, certain theories suggest that prices, particularly for some assets, tend to revert to their long-term average. This concept, known as mean reversion, forms the basis for numerous trading strategies. But can a sophisticated mathematical model, like the Ornstein-Uhlenbeck (OU) process, truly capture and profit from this elusive market behavior?

This article delves into the implementation of an Ornstein-Uhlenbeck Mean Reversion strategy using `backtrader`, a

powerful Python backtesting framework, and `yfinance` for data acquisition. We'll explore how to model asset prices as an OU process, estimate its parameters on a rolling basis, and generate trading signals based on deviations from the estimated mean.

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

Understanding the Ornstein-Uhlenbeck Process

The Ornstein-Uhlenbeck process is a continuous-time stochastic process that describes the velocity of a particle undergoing Brownian motion under the influence of a spring-like force that pulls it towards a central equilibrium position. In finance, it's often used to model variables that are mean-reverting, such as interest rates, commodity prices, or, as in our case, the logarithm of asset prices.

The key parameters of an OU process are:

- μ (mu): The long-term mean or equilibrium level to which the process reverts.

- θ (theta): The speed of reversion, indicating how quickly the process tends to pull back towards the mean. A higher implies faster reversion.
- σ (sigma): The volatility or the standard deviation of the noise term, representing the magnitude of random fluctuations.

The Strategy: OU Mean Reversion with Trend Filter

Our strategy, implemented in `backtrader`, aims to identify when an asset's price has deviated significantly from its estimated OU mean and then trade on the expectation that it will revert. To enhance the robustness of the strategy and avoid false signals in strong trends, we incorporate a simple moving average (SMA) as a trend filter.

Here's how it works:

1. Rolling OU Parameter Estimation: The strategy continuously estimates the , , and parameters of the Ornstein-Uhlenbeck process over a defined `lookback` period using Ordinary Least Squares (OLS) regression on the log prices. This allows the strategy to adapt to changing market conditions.
2. Z-Score Calculation: For each bar, a Z-score is calculated, representing how many standard deviations the current log price is away from the estimated long-term mean, normalized by the equilibrium standard deviation.

3. Trend Filtering: A Simple Moving Average (SMA) of the closing prices is calculated. This acts as a filter to ensure trades are placed in alignment with the broader trend.
4. Entry Signals:

- Long Entry: If the Z-score falls below a negative `entry_threshold` (indicating the price is significantly below its mean) AND the current price is above the SMA (indicating an uptrend), a long position is initiated. The rationale is to buy a "cheap" asset in an upward-trending market, expecting it to revert to its mean.
- Short Entry: If the Z-score rises above a positive `entry_threshold` (indicating the price is significantly above its mean) AND the current price is below the SMA (indicating a downtrend), a short position is initiated. Here, we sell a "expensive" asset in a downward-trending market, expecting it to revert downwards.

1. Exit Signals:

- Long Exit: A long position is exited when the Z-score rises above a negative `exit_threshold` (i.e., the price has moved back closer to or above its mean).
- Short Exit: A short position is exited when the Z-score falls below a positive `exit_threshold` (i.e., the price has moved

back closer to or below its mean).

Python Code

Let's break down the Python code for this strategy.

```
import backtrader as bt
import yfinance as yf
import numpy as np
import pandas as pd
from scipy import stats
import warnings

warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt
%matplotlib inline

class OUMeanReversionStrategy(bt.Strategy):
    """
    Ornstein-Uhlenbeck Mean Reversion Strategy

    The strategy estimates OU process parameters over a rolling win
    and generates trading signals based on deviations from the esti
    """

    params = (
        ('lookback', 60),           # Rolling window for OU paramet
        ('sma_period', 30),         # sma period for trend
        ('entry_threshold', 1.5),   # Z-score threshold for entry
        ('exit_threshold', 0.5),    # Z-score threshold for exit
        ('printlog', False),        # Print trade logs
    )

    def __init__(self):
        # Data feeds
        self.dataclose = self.datas[0].close
        self.sma = bt.indicators.SimpleMovingAverage(self.dataclose
```

```
        # Track our position
        self.order = None
        self.position_type = None  # 'long', 'short', or None

        # Store OU parameters and signals
        self.ou_params = []
        self.z_scores = []

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

    def notify_order(self, order):
        """Handle order notifications"""
        if order.status in [order.Submitted, order.Accepted]:
            return

        if order.status in [order.Completed]:
            if order.isbuy():
                self.log(f'BUY EXECUTED: Price: {order.executed.pri
                         f'Cost: {order.executed.value:.2f}, Comm:
            else:
                self.log(f'SELL EXECUTED: Price: {order.executed.pr
                         f'Cost: {order.executed.value:.2f}, Comm:

        elif order.status in [order.Canceled, order.Margin, order.R
            self.log('Order Canceled/Margin/Rejected')

        self.order = None

    def estimate_ou_parameters(self, log_prices):
        """
        Estimate Ornstein-Uhlenbeck parameters using OLS regression

        OU process: dX = θ(μ - X)dt + σdW
        Discretized: X_t - X_{t-1} = θμΔt - θX_{t-1}Δt + ε_t

        Returns: (mu, theta, sigma, equilibrium_std)
        """
        if len(log_prices) < 10:  # Need minimum data points
            return None, None, None, None
```

```
        # Prepare regression data
        x_lag = log_prices[:-1]  # X_{t-1}
        dx = np.diff(log_prices)  # X_t - X_{t-1}

        try:
            # OLS regression: dx = alpha + beta * x_lag
            slope, intercept, r_value, p_value, std_err = stats.lin

            # Convert to OU parameters (assuming dt = 1)
            theta = -slope
            mu = intercept / theta if theta > 1e-6 else np.mean(log

            # Estimate sigma from residuals
            residuals = dx - (intercept + slope * x_lag)
            sigma = np.std(residuals)

            # Equilibrium standard deviation
            equilibrium_std = sigma / np.sqrt(2 * theta) if theta >

            return mu, theta, sigma, equilibrium_std

        except Exception as e:
            return None, None, None, None

    def next(self):
        """Main strategy logic called on each bar"""

        # Need enough data for parameter estimation
        if len(self.dataclose) < self.params.lookback:
            return

        # Get recent log prices for parameter estimation
        recent_log_prices = np.array([np.log(self.dataclose[-i]) fo

        # Estimate OU parameters
        mu, theta, sigma, eq_std = self.estimate_ou_parameters(rece

        if mu is None or eq_std is None or eq_std <= 0:
            return

        # Calculate current deviation and z-score
        current_log_price = np.log(self.dataclose[0])
        deviation = current_log_price - mu
```

```
            z_score = deviation / eq_std

            # Store for analysis
            self.ou_params.append({'mu': mu, 'theta': theta, 'sigma': s
            self.z_scores.append(z_score)

            self.log(f'Close: {self.dataclose[0]:.4f}, Log Price: {curr
                     f'μ: {mu:.4f}, Z-Score: {z_score:.2f}')

            # Skip if we have a pending order
            if self.order:
                return

            # Trading logic
            if not self.position:  # No position
                if z_score < -self.params.entry_threshold and self.data
                    # Price below mean AND uptrending - go long (expect
                    self.log(f'LONG SIGNAL: Z-Score {z_score:.2f}')
                    self.order = self.buy()
                    self.position_type = 'long'

                elif z_score > self.params.entry_threshold and self.dat
                    # Price above mean AND downtrending - go short (exp
                    self.log(f'SHORT SIGNAL: Z-Score {z_score:.2f}')
                    self.order = self.sell()
                    self.position_type = 'short'

            else:  # We have a position
                if self.position_type == 'long' and z_score > -self.par
                    # Exit long position
                    self.log(f'EXIT LONG: Z-Score {z_score:.2f}')
                    self.order = self.sell()
                    self.position_type = None

                elif self.position_type == 'short' and z_score < self.p
                    # Exit short position
                    self.log(f'EXIT SHORT: Z-Score {z_score:.2f}')
                    self.order = self.buy()
                    self.position_type = None


    def run_ou_strategy(ticker='EURUSD=X', start_date='2020-01-01', end
                        cash=10000, lookback=60, sma_period=30, entry_t
        """
```

```
    Run the OU Mean Reversion strategy
    """

    print(f"=== OU Mean Reversion Strategy ===")
    print(f"Ticker: {ticker}")
    print(f"Period: {start_date} to {end_date}")
    print(f"Lookback: {lookback} days")
    print(f"Entry Threshold: ±{entry_threshold}")
    print(f"Exit Threshold: ±{exit_threshold}")
    print(f"Initial Cash: ${cash:,.2f}")
    print("=" * 50)

    # Download data
    print("Downloading data...")
    data = yf.download(ticker, start=start_date, end=end_date, auto

    if data.empty:
        print(f"No data found for {ticker}")
        return None

    # Convert to Backtrader format
    bt_data = bt.feeds.PandasData(dataname=data)

    # Create Cerebro engine
    cerebro = bt.Cerebro()

    # Add strategy
    cerebro.addstrategy(OUMeanReversionStrategy,
                        lookback=lookback,
                        sma_period=sma_period,
                        entry_threshold=entry_threshold,
                        exit_threshold=exit_threshold,
                        printlog=False)  # Set to True for detailed

    # Add data
    cerebro.adddata(bt_data)

    # Set cash
    cerebro.broker.setcash(cash)

    # Add commission (0.1% per trade)
    cerebro.broker.setcommission(commission=0.001)

    cerebro.addsizer(bt.sizers.PercentSizer, percents=95)
```

```python
# Add analyzers
cerebro.addanalyzer(bt.analyzers.Returns, _name='returns')
cerebro.addanalyzer(bt.analyzers.SharpeRatio, _name='sharpe')
cerebro.addanalyzer(bt.analyzers.DrawDown, _name='drawdown')
cerebro.addanalyzer(bt.analyzers.TradeAnalyzer, _name='trades')

# Run strategy
print("Running strategy...")
results = cerebro.run()
strat = results[0]

# Print results
print("\n=== PERFORMANCE SUMMARY ===")

final_value = cerebro.broker.getvalue()
total_return = (final_value - cash) / cash * 100
print(f"Initial Portfolio Value: ${cash:,.2f}")
print(f"Final Portfolio Value: ${final_value:,.2f}")
print(f"Total Return: {total_return:.2f}%")

# Get analyzer results
returns_analysis = strat.analyzers.returns.get_analysis()
sharpe_analysis = strat.analyzers.sharpe.get_analysis()
drawdown_analysis = strat.analyzers.drawdown.get_analysis()
trades_analysis = strat.analyzers.trades.get_analysis()

print(f"\nAnnualized Return: {returns_analysis.get('rnorm100',
print(f"Sharpe Ratio: {sharpe_analysis.get('sharperatio', 0):.2
print(f"Max Drawdown: {drawdown_analysis.get('max', {}).get('dr

if 'total' in trades_analysis:
    total_trades = trades_analysis['total']['total']
    won_trades = trades_analysis['won']['total'] if 'won' in tr
    win_rate = (won_trades / total_trades * 100) if total_trade
    print(f"Total Trades: {total_trades}")
    print(f"Win Rate: {win_rate:.1f}%")

# Plot results
print("\nGenerating plots...")
plt.rcParams['figure.figsize'] = [10, 6]
cerebro.plot(style='candlestick', barup='green', bardown='red',
```

```
    return cerebro, results


# Example usage
if __name__ == '__main__':
    # Run the strategy
    cerebro, results = run_ou_strategy(
        ticker='ETH-USD',
        start_date='2020-01-01',
        end_date='2024-12-31',
        cash=10000,
        lookback=30,
        sma_period=30,
        entry_threshold=1.2,
        exit_threshold=0.8
    )

    # You can also test with other assets like stocks or crypto
    # cerebro, results = run_ou_strategy(ticker='AAPL', start_date=
    # cerebro, results = run_ou_strategy(ticker='BTC-USD', start_da
```

`OUMeanReversionStrategy` **Class:**

- `params` : Defines the configurable parameters of the strategy:
- `lookback` : The number of past data points (days) to use for estimating OU parameters.
- `sma_period` : The period for the Simple Moving Average (SMA) trend filter.
- `entry_threshold` : The Z-score deviation required to initiate a trade.
- `exit_threshold` : The Z-score deviation at which to close an open position.

- `printlog` : A boolean to control detailed logging.
- `__init__(self)` :
- Initializes `self.dataclose` to easily access the closing price.
- Creates a `SimpleMovingAverage` indicator.
- Initializes `self.order` to track pending orders and `self.position_type` to track the current position (long/short/none).
- `self.ou_params` and `self.z_scores` lists are used to store calculated values for later analysis/plotting.
- `log(self, txt, dt=None)` : A utility function for logging messages, controlled by `printlog` .
- `notify_order(self, order)` : A `backtrader` callback method that is invoked when an order changes its status (e.g., submitted, completed, canceled). It logs the details of executed trades and handles pending order status.
- `estimate_ou_parameters(self, log_prices)` : This is the core mathematical function:
- It takes a series of `log_prices` (logarithm of asset prices).
- It prepares data for Ordinary Least Squares (OLS) regression by creating `x_lag` (lagged log prices) and `dx` (daily changes in log prices).

- It performs a linear regression (`stats.linregress`) to find the `slope` and `intercept` of `dx` against `x_lag` .
- These `slope` and `intercept` values are then used to calculate the OU parameters:
- `theta = -slope` : The speed of reversion.
- `mu = intercept / theta` : The long-term mean.
- `sigma` : Estimated from the standard deviation of the regression residuals.
- `equilibrium_std = sigma / np.sqrt(2 * theta)` : The standard deviation of the process at equilibrium.
- Error handling is included to return `None` if parameters cannot be estimated.
- `next(self)` : This method is called by `backtrader` for each new bar of data:
- It first checks if enough data is available for the `lookback` period.
- It retrieves `recent_log_prices` for the `lookback` window.
- Calls `estimate_ou_parameters` to get the OU parameters.
- Calculates the `z_score` of the current log price relative to the estimated mean and equilibrium standard deviation.
- Logs the current market data and calculated Z-score.

- Trading Logic:
- If there's no open position:
- It checks for long entry conditions: Z-score below negative entry threshold AND current price above SMA.
- It checks for short entry conditions: Z-score above positive entry threshold AND current price below SMA.
- If there's an open position:
- It checks for long exit conditions: Z-score has reverted above the negative exit threshold.
- It checks for short exit conditions: Z-score has reverted below the positive exit threshold.
- `self.buy()` and `self.sell()` are `backtrader` functions to place market orders.

`run_ou_strategy` Function:

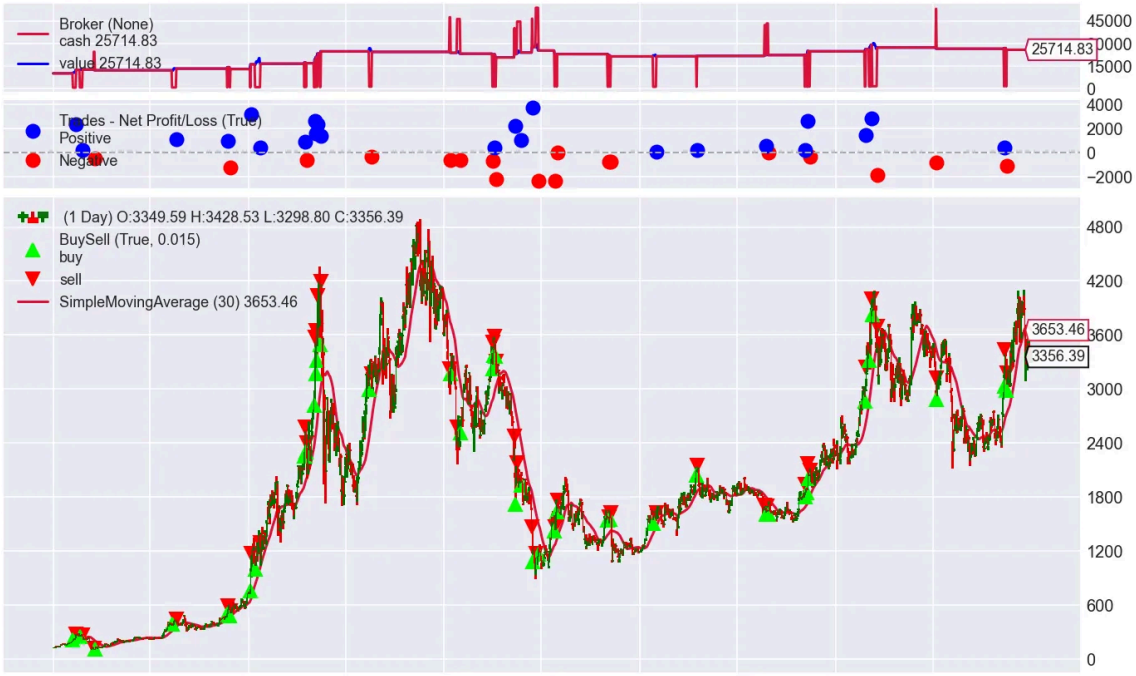
- This function encapsulates the backtesting process.
- Parameter Setup: Takes `ticker`, `start_date`, `end_date`, `cash`, and strategy parameters as input.
- Data Download: Uses `yfinance.download` to fetch historical data for the specified ticker and date range.
- `auto_adjust=False` and `droplevel(1, 1)` are used as per the user's saved preference for `yfinance` data handling.

- Cerebro Engine:
- `cerebro = bt.Cerebro()` creates the main backtesting engine.
- `cerebro.addstrategy(...)` : Adds an instance of our `OUMeanReversionStrategy` with the specified parameters.
- `cerebro.adddata(...)` : Feeds the downloaded data to the engine.
- `cerebro.broker.setcash(...)` : Sets the initial capital.
- `cerebro.broker.setcommission(...)` : Sets a commission for trades (0.1%).
- `cerebro.addsizer(bt.sizers.PercentSizer, percents=95)` : This sizer ensures that 95% of the available cash is used for each trade.
- `cerebro.addanalyzer(...)` : Adds various `backtrader` analyzers to calculate performance metrics (Returns, Sharpe Ratio, Drawdown, Trade statistics).
- Execution: `cerebro.run()` executes the backtest.
- Performance Summary: After the backtest, it prints a detailed summary of the strategy's performance, including total return, annualized return, Sharpe Ratio, maximum drawdown, and trade statistics.
- Plotting: `cerebro.plot()` generates a visual representation of the trades and equity curve.

Running the Example

The `if __name__ == '__main__':` block demonstrates how to run the strategy:

```
if __name__ == '__main__':
    # Run the strategy
    cerebro, results = run_ou_strategy(
        ticker='ETH-USD',
        start_date='2020-01-01',
        end_date='2024-12-31',
        cash=10000,
        lookback=30,
        sma_period=30,
        entry_threshold=1.2,
        exit_threshold=0.8
    )
```



This example runs the strategy on `ETH-USD` (Ethereum to USD) data from 2020 to 2024 with specific parameters. You can easily modify the `ticker`, `start_date`, `end_date`, and strategy parameters to test it on different assets (e.g., `'AAPL'` for Apple stock or `'BTC-USD'` for Bitcoin) and optimize its performance.

Conclusion

The Ornstein-Uhlenbeck Mean Reversion strategy offers an intriguing approach to trading, attempting to capitalize on the tendency of prices to revert to their historical averages. By dynamically estimating the OU process parameters and incorporating a trend filter, this strategy aims to generate robust trading signals. While mean reversion strategies can be effective in certain market regimes, it's crucial to remember that past performance does not guarantee future results. Further research, optimization, and robust out-of-sample testing are always recommended before deploying any trading strategy in a live environment.

Python | Algorithmic Trading | Quantitative Finance | Trading Strategy | Backtesting

Written by PyQuantLab

655 followers · 6 following

Following

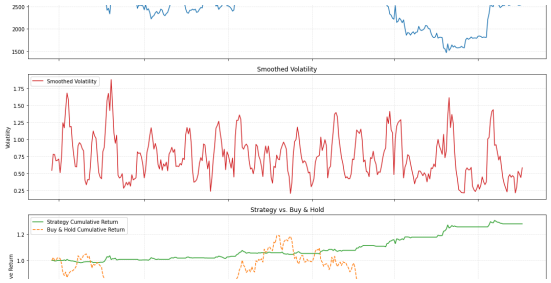
Your go-to place for Python-based quant tutorials, strategy deep-dives, and reproducible code. For more visit our website: www.pyquantlab.com

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...

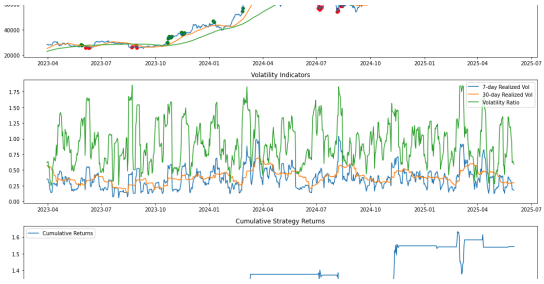
Jun 3 32 3

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

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...

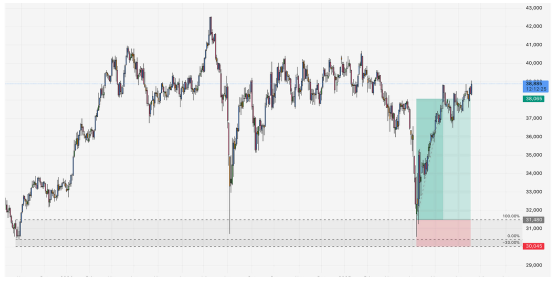
Jun 3 75 3

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...

In InsiderFinance Wire by Itay V

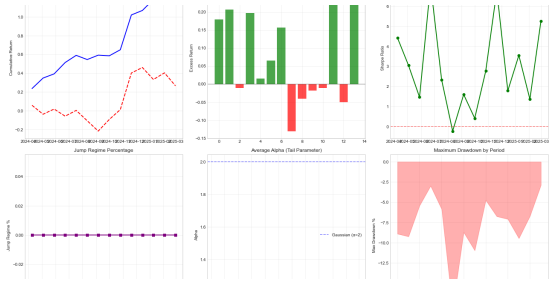
I tried trading with Agentic AI, and it’s mind blowing

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

Jun 19 2

Discover how agentic systems are transforming the retail-trading scene...

May 27 705 11

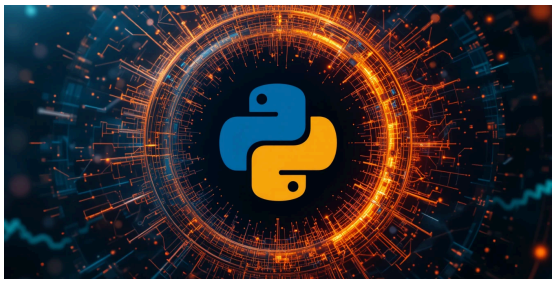


PyQuantLab

Capturing Market Momentum with Levy Flights: A Python...

Financial markets are complex systems, often exhibiting behaviors...

Jun 5 102



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

See more recommendations