Algo Trading Strategy Project

Doji Pattern & Bollinger Bands with Advanced Optimizations

By: **Lakshay Sawhney**

LinkedIn: https://www.linkedin.com/in/lakshay-sawhney/

Table of Contents

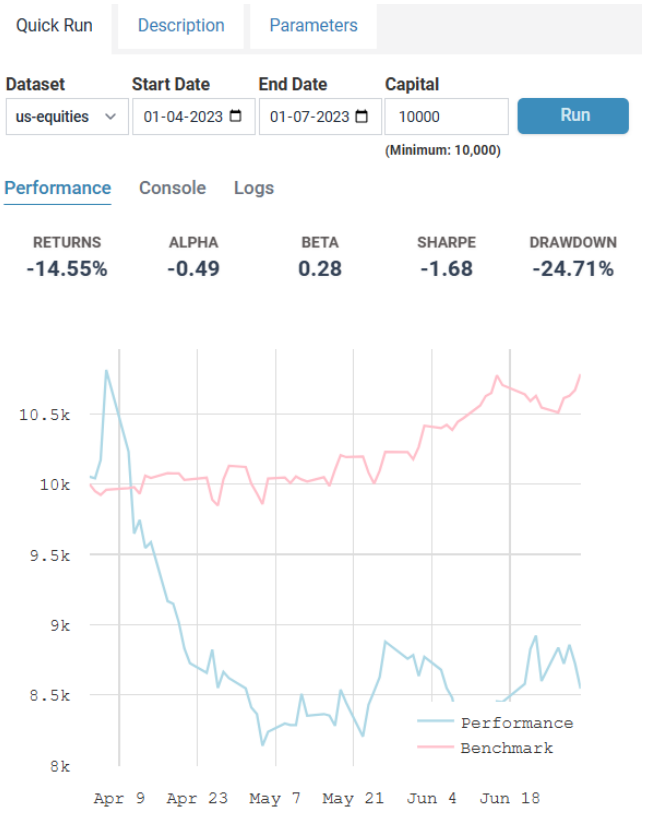
Overview

1. **Objective:**
    - The primary goal was to select one of the eight pre-existing strategies on the Blueshift platform, analyse its performance, and modify it to create a robust and profitable trading strategy.
2. **Doji-Based Strategy**:
    - The Doji Pattern strategy was chosen for its recognition among professional traders and its potential for innovation. This strategy offers significant flexibility by incorporating dynamic factors such as volumetric analysis, candlestick pattern recognition, and trend reversals.
3. **Work on Doji-Based Strategy:**
    - The Doji strategy was optimized through extensive experimentation and included the following key improvements:
        - **Parameter Fine-Tuning:** Adjusting key variables like Bollinger Band periods, indicator lookback, MACD fast/slow/signal values and ATR multipliers for better performance.
        - **Indicator Integration:** Enhancing signal reliability by incorporating additional indicators like:
            - **MACD** for trend direction and momentum.
            - **RSI** for overbought/oversold confirmation.
            - **OBV/Volumetric Analysis** to validate signal strength with volume data.
            - **ATR** for dynamic position sizing and risk management.
        - **Multiple Candlestick Pattern Recognition:** Recognizing a wider variety of candlestick patterns like Doji, Long Legged Doji, Gravestone Doji, Hammer and Inverted Hammer for improved signal generation.
        - **Dynamic Thresholds:** Implementing adaptive thresholds to account for market volatility.
        - **Dynamic Position Sizing:** Calculating position sizes based on ATR and market conditions to manage risk effectively.
        - **Risk Management:** Adding dynamic stop-loss and take-profit levels to minimize drawdowns and lock in profits.
        - **Avoiding Overfitting:** Ensured robustness by testing on multiple time periods and datasets, dynamically adjusting time frames, and reducing trade frequency for faster backtesting without affecting performance metrics.

Methodology

Step 1: Backtesting default strategy code on the initial dataset to establish a baseline performance.

Refer to Source Code 1 for default strategy code.



Result:

- Returns: **-14.55%,** Alpha: **-0.49,** Beta: **0.28,** Sharpe: **-1.68**, Drawdown: **-24.71%**
- Graph illustrates significant underperformance, with the strategy incurring **heavy losses** compared to the benchmark.

Analysis:

- The default Doji Pattern strategy, in its current form, is **highly unprofitable**, with substantial losses and a high drawdown of nearly 25%.
- **Possible causes** for poor performance:
    - Inadequate parameter tuning for the given dataset and market conditions.
    - Lack of additional risk management measures like optimized stop-loss or take-profit levels.
    - Over-reliance on Doji patterns without corroborating indicators or volume confirmation.

Immediate Goal:
The primary focus is to **minimize the losses** and improve performance by fine-tuning the strategy.

Possible Approach/Solution:

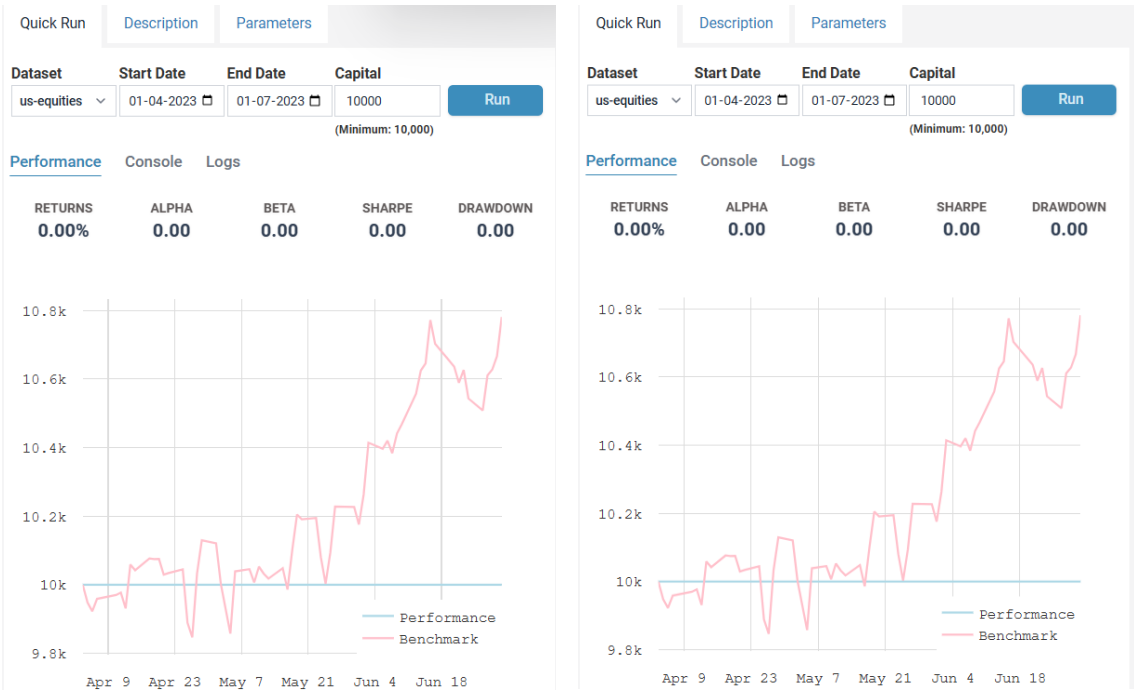
- Optimize lookback periods for better pattern recognition.
- Adjust Bollinger Band periods to better capture market trends and volatility.

Step 2: Adjusting Indicator Lookback Period

Approach:

- **Objective**: Shorten the indicator lookback period to focus on recent price movements and reduce historical noise.
- **Current Setting**: Lookback period set to 375.
- **Rationale**:
    - A larger lookback period may include too much historical data, which is less relevant for intraday strategies.
    - Reducing the lookback period to values like 50 or 100 aims to provide signals based on recent market trends, enhancing the strategy's responsiveness.

Result:



(Corresponding Graphs when indicator lookback is set to 50 and 100 respectively).

- **Outcome**: No trades occurred during the backtesting process for both adjusted lookback periods (50 and 100).

Analysis:

- A shorter lookback period resulted in zero trades, indicating:
    - Insufficient signal strength with reduced historical data.
    - Misalignment of the adjusted parameters with the chosen dataset or timeframe.
    - The absence of conditions required to trigger the Doji pattern under the given market settings.

Step 3: Adjusting Buy/Sell Signal Thresholds and Indicator Lookback Period

Approach:

- **Objective**: Refine the buy and sell signal thresholds and modify the indicator lookback period to potentially generate trades.
- **Changes Implemented**:
    1. **Buy Signal Threshold**: Adjusted from **0.5 to 0.3** to make the strategy more sensitive to potential upward movements.
    2. **Sell Signal Threshold**: Adjusted from **-0.5 to -0.3** to detect downward trends more effectively.
    3. **Indicator Lookback Period**: Increased from **100 to 250** to balance the trade-off between capturing recent price trends and reducing excessive historical noise.

Result:

- **Outcome**: No trades were generated during the backtesting process with these updated parameters.

Analysis:

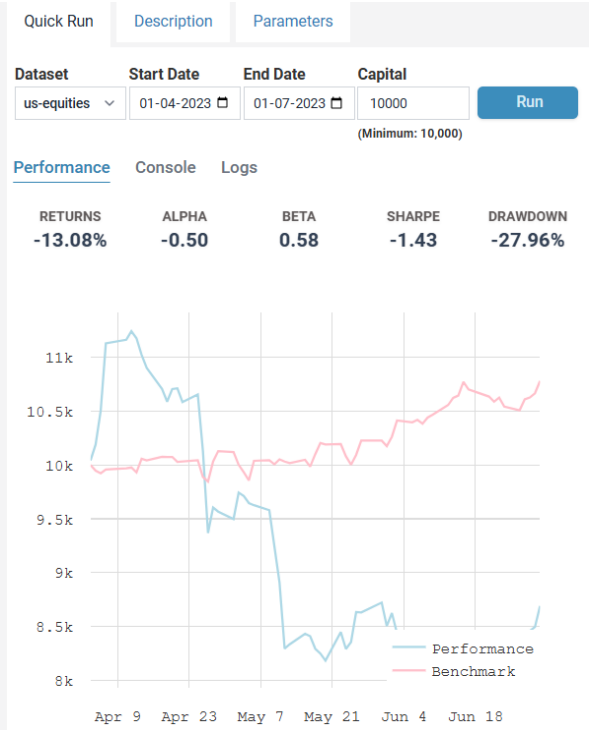
- The lack of trades indicates that even with relaxed thresholds and an adjusted lookback period:
    - **Signal Triggers**: The conditions required for buy or sell signals were not met, likely due to market conditions or indicator behaviour.
    - **Market Alignment**: The chosen thresholds and lookback period may still not align with the dynamics of the dataset or strategy requirements.

Step 4: Adjusting Bollinger Bands Period

Approach:

- **Objective**: Modify the Bollinger Bands period to observe its impact on the strategy's performance and signal generation.
- **Changes Implemented**:
    - Adjusted the **Bollinger Bands Period** from **50 to 20** to increase the sensitivity of the bands to price movements and capture shorter-term market trends.

Result:



- **Outcome**:
    - **Returns**: -13.08%, **Alpha**: -0.50, **Beta**: 0.58, **Sharpe Ratio**: -1.43, **Drawdown**: -27.96%
    - **Observation**: The strategy incurred significant losses, with a considerable drawdown, indicating the need for substantial adjustments.

Analysis:

- **Heavy Losses**:
    - The reduced Bollinger Bands period may have introduced excessive sensitivity to minor price fluctuations, resulting in false signals.
    - Increased trades due to the tighter bands might have amplified losses during choppy or sideways market conditions.
- **Strategy Weakness**:
    - The current strategy may lack sufficient risk management or complementary indicators to filter out noise and confirm signals.

Possible Solutions:

- **Enhancing Signal Reliability:**
    - Introduce complementary indicators like RSI to measure overbought/oversold conditions and MACD to detect momentum shifts. These could help filter out false signals and improve trade accuracy.
- **Dynamic Adjustments:**
    - Replace static thresholds with dynamic thresholds based on recent market volatility, allowing the strategy to adapt to changing conditions more effectively.
- **Improving Risk Management:**
    - Implement stop-loss and take-profit mechanisms to minimize drawdowns and lock in profits. These risk management tools could reduce the impact of large losses.
- **Dynamic Position Sizing:**
    - Use ATR-based position sizing to adjust trade sizes according to market volatility. High ATR values could indicate reducing position sizes to manage risk during volatile periods.

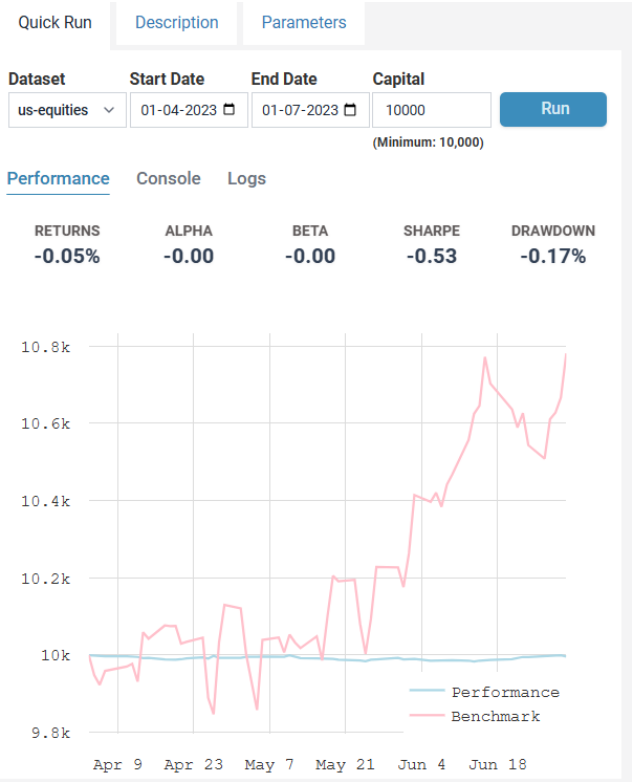
Step 5: Refining Strategy with Enhanced Parameters and Indicators

Refer to Source Code 2 for code of this updated strategy.

Approach:

- Adjusted the Bollinger Bands period back to 20, introduced RSI and MACD for better momentum and trend detection.
- Incorporated dynamic thresholds and ATR-based position sizing for volatility adaptation.
- Added stop-loss and take-profit levels to reduce exposure to market noise and ensure risk management.

Result:



- Reduced losses with returns at -0.05% compared to -13.08% previously.
- Sharpe ratio improved slightly but remains negative, indicating a need for further optimization.
- Drawdown significantly reduced to -0.17%, reflecting better risk management.

Analysis:

- The strategy showed improvement by minimizing losses, but it still struggles to generate profits.
- Potential reasons for no profits yet:
    1. **Tight Stop-Loss/Take-Profit**: Even with better-defined levels, they might still be too tight for intraday volatility, triggering frequent exits.
    2. **Overfitting Indicators**: A combination of Bollinger Bands, RSI, MACD, and dynamic thresholds might be conflicting, reducing clear signals.
    3. **Leverage and Position Sizing**: Dynamic sizing based on ATR might be allocating smaller positions, limiting profit potential in trending markets.

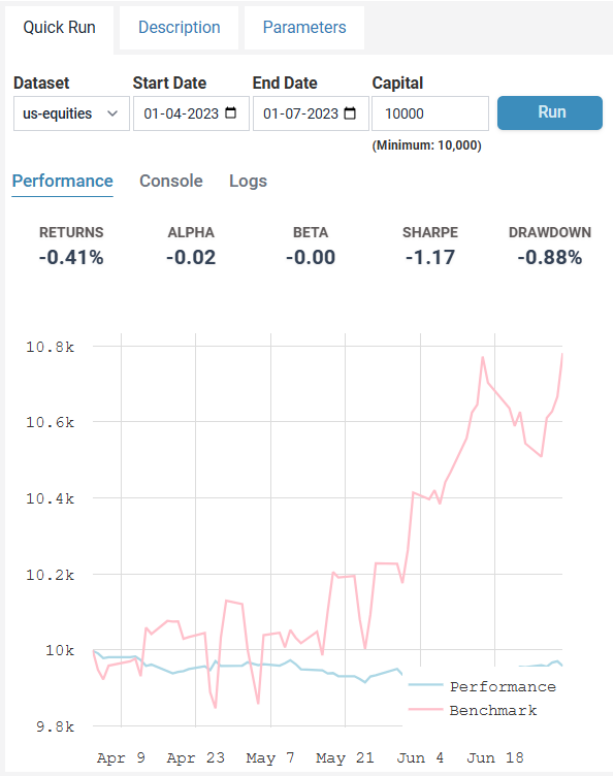
Step 6: Refinement and Testing of Strategy Adjustments

Refer to Source Code 3 for code of this strategy.

Approach:

- **Stop Loss/Take Profit Adjustment**: Increased stop loss to 5% and take profit to 7% to give trades more room for fluctuations in volatile market conditions.
- **Dynamic Thresholds**: Continued using thresholds based on ATR to dynamically adapt to market volatility.
- **Simplified Indicators:** Prioritized MACD as the primary momentum indicator and removed conflicting indicators like RSI to reduce noise and ensure cleaner, more reliable trade signals.
- **Leverage**: Maintained leverage at 2x to enhance profit potential while keeping an eye on drawdowns.

Results:



- **Returns**: -0.41%, **Alpha**: -0.02, **Beta**: -0.00, **Sharpe Ratio**: -1.17, **Drawdown**: -0.88%
- **Graph**: The graph shows minimal returns and low correlation with the benchmark. Losses were reduced significantly but the strategy failed to yield profits during the backtesting period.

Analysis:

- **Performance Improvements**:
    - The strategy showed a considerable reduction in drawdown, indicating improved risk management.
    - Returns improved slightly compared to previous steps but remained in the negative zone.
- **Persistent Issues**:
    - **Conservative Parameters**: A stop loss of 5% and take profit of 7% might still be too restrictive, potentially cutting profits short in trending markets.
    - **Inefficient Position Sizing**: The ATR-based dynamic position sizing may allocate overly small weights, limiting profit potential.
    - **Low Beta and Sharpe Ratio**: The lack of correlation with the benchmark and negative risk-adjusted returns highlight inefficiency in capturing market opportunities.

Immediate Adjustments:

1. **Expand Stop Loss/Take Profit Margins**:
    - Consider increasing the stop loss to 15% and take profit to 20% for better alignment with market trends.
2. **Modify Thresholds**:
    - Experiment with slightly less restrictive buy/sell thresholds to encourage more trade signals.

Step 7: Strategy Refinement with Integrated Enhancements

Refer to Source Code 4 for code of this strategy.

Adjustments Made:

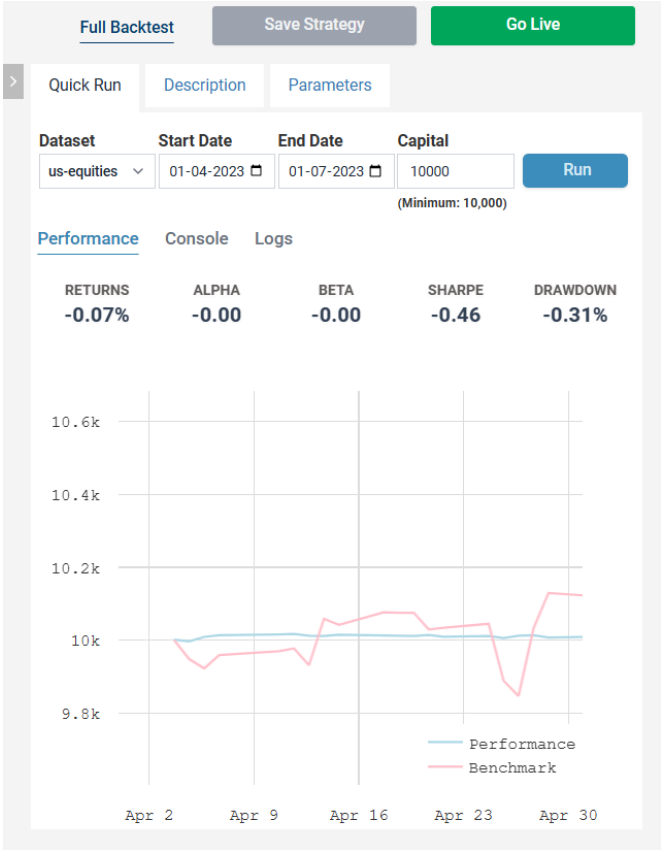
1. **Stop Loss and Take Profit:**
    - Increased stop loss to **15%** to allow trades more room for volatility.
    - Increased take profit to **20%**, aligning with a higher risk-reward ratio.
2. **Trade Frequency:**
    - Changed the trade frequency from **5 minutes** to **3 minutes**, targeting shorter-term opportunities.
3. **Intelligent Indicator Combination:**
    - Combined **MACD** with **Bollinger Bands** for better signal accuracy.

- Incorporated **ADX** (Average Directional Index) as a trend filter to avoid trading in low-trend or sideways markets.

Rationale:

- **Improved Risk Management:** Loosening stop loss and take profit levels reduces the chances of premature exits and maximizes potential returns in trending conditions.
- **Efficient Trend Filtering:** ADX ensures trades occur only in sufficiently trending markets, minimizing unnecessary losses.
- **Enhanced Signal Generation:** The intelligent combination of MACD and Bollinger Bands enhances accuracy by confirming momentum-based trends while avoiding conflicting signals.

Results:



- **Performance Metrics:**
  - Returns: **-0.07%,** Alpha: **-0.00,** Beta: **-0.00,** Sharpe Ratio: **-0.46,** Drawdown: **-0.31%**
- **Graph Analysis:**
  - Marginal improvement in drawdown and losses, indicating better-controlled risk.
  - However, the strategy still underperforms against the benchmark, highlighting inefficiencies in capturing market opportunities.

Diagnosis of Current Issues:

1. **Combination of Indicators:**
    - MACD and Bollinger Bands may still produce redundant signals in ranging markets.
    - ADX threshold (set at 20) might filter out potential trades in moderate trends.
2. **Returns and Sharpe Ratio:**
    - Despite reduced losses, the Sharpe ratio and returns remain negative, showing inefficiency in risk-adjusted returns.
3. **Dynamic Position Sizing:**
    - The current ATR-based sizing cap might still be too conservative, limiting profit potential.

Possible Solutions:

1. **Adjusting ADX Threshold:**
    - Lowering the threshold to 15 to capture trades in moderate trend conditions.
2. **Optimizing Position Sizing:**
    - Increasing the dynamic position sizing cap to 30% for high-confidence trades.
3. **Diversifying Indicators Further:**
    - Exploring other momentum-based indicators or trailing stop mechanisms to refine signal accuracy.

Step 8: Enhancing the Strategy Further

Refer to Source Code 5 for code of this strategy.

Objective:

The goal of this step was to refine the strategy by implementing dynamic adjustments and leveraging complementary indicators to improve profitability and robustness. This step builds on insights from earlier iterations.

Key Changes:

1. **Stop Loss and Take Profit Adjustments**:
    - **Stop Loss** increased to **15%** to allow for more breathing room in volatile markets.
    - **Take Profit** raised to **20%** to capture larger upward price movements.
2. **Trade Frequency**:
    - Reduced the trade execution interval from **5 minutes** to **2 minutes** for faster response to market changes.

3. **Intelligent Combination of Indicators**:
    - MACD and Bollinger Bands were combined strategically to refine entry/exit signals:
        - Signals align when MACD crossover momentum supports Bollinger Band signals.
    - Added **ADX** (with a lower threshold of 15) to filter trades during weak trends, ensuring trades are executed only in trending markets.
4. **Dynamic Position Sizing**:
    - Enhanced the position sizing formula to dynamically adjust weights based on ATR values.
    - Increased the maximum cap for weights to **30%** in high-confidence trades.

Results:

- **Performance Metrics**:
    - **Returns**: -0.42%, **Alpha**: -0.02, **Beta**: 0.00, **Sharpe Ratio**: -1.91, **Drawdown**: -0.71%
- **Graph Insights**:
    - While the drawdown was minimized significantly, the returns remain slightly negative, highlighting the need for further optimization.

Diagnosis of Current Issues:

1. **Moderate Trend Sensitivity**:
    - ADX filtering with a threshold of 15 may still exclude trades in early trend formations.

2. **Complex Indicator Interactions**:
    - The combination of MACD, Bollinger Bands, and ADX introduces complexity that might lead to missed opportunities in choppy markets.
3. **Dynamic Position Sizing**:
    - Although improved, the sizing formula may still be too conservative for maximizing returns in favorable conditions.

Step 9: *Transitioning to the Bucket Method for Strategic Optimization*

Objective:

Faced with the persistent shortcomings of the strategy—highlighted by key metrics such as **Beta = 0** (indicating no correlation with the benchmark)—a more structured and innovative approach was needed. This led to a pause in execution to analyse the issues thoroughly, research potential improvements, and brainstorm a systematic method for strategic testing and optimization.

Insight:

Recognizing the fragmented approach of prior iterations, the **Bucket Method** was developed. This approach divides optimization into focused areas, ensuring comprehensive exploration and strategic alignment across all aspects of the trading model.

The Bucket Method:

The optimization process was categorized into **four distinct buckets**:

1. **Bucket 1: Signal Refinement**:
    - **Volume Confirmation**:
        - Prioritized signals supported by high trading volume, particularly during Doji patterns, using **On-Balance Volume (OBV)** to validate trends and momentum.
    - **Multiple Candlestick Patterns**:
        - Added recognition of patterns like Gravestone Doji, Dragonfly Doji, Hammer, and Inverted Hammer to enhance signal diversity.
    - **Multi-Timeframe Analysis**:
        - Integrated multi-timeframe analysis to align intraday signals with broader market trends, ensuring higher reliability.
2. **Bucket 2: Risk Management**:
    - **Dynamic Stop Loss/Take Profit**:
        - Use ATR-based stop loss and take profit levels to adapt to market volatility.
    - **Evaluation of No Stop Loss/Take Profit**:

- Test scenarios without predefined stop loss or take profit to explore potential for trend-riding strategies.

3. **Bucket 3: Position Sizing**:
   - **Static vs Dynamic ATR Sizing**:
     - Compare fixed position sizing with dynamic adjustments based on ATR.
   - **Threshold-Based Sizing**:
     - Experiment with simpler methods of assigning positions based on predefined thresholds.
4. **Bucket 4: Parameter Tuning**:
   - **Indicator Parameters**:
     - Optimize key parameters such as:
       - Bollinger Band period (BBands_period).
       - MACD fast/slow/signal values.
       - Indicator lookback period (indicator_lookback).
   - **Mean Reversion Logic**:
     - Introduce and test mean reversion strategies alongside the momentum-based approach.

Approach:

- **Set Aside Execution**:
  - A deliberate pause in strategy execution was implemented to allow for focused research and iterative testing.
- **Iterative Testing**:
  - Each bucket was tested independently to isolate its impact on performance metrics.
- **Hypothesis-Driven Optimization**:
  - Each test was based on hypotheses formed during the brainstorming phase, ensuring a data-driven approach to refinement.

Significance:

- The **Bucket Method** represents a shift from reactive adjustments to proactive, hypothesis-driven optimization. This structured approach ensures that each iteration builds on validated improvements, moving closer to a profitable and robust trading strategy.

Bucket 1: Signal Refinement

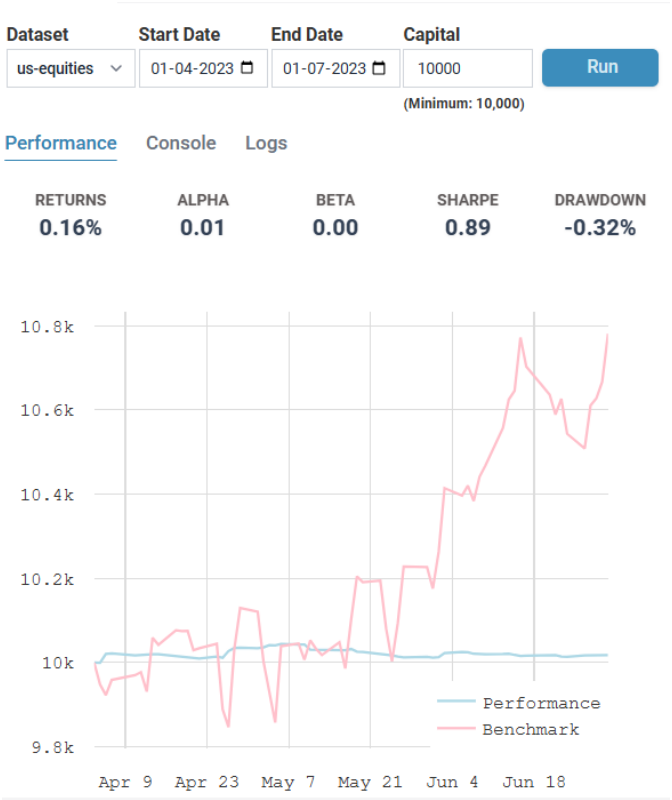
Step 10: Adding Volume Confirmation Using OBV (On-Balance Volume)

Refer to Source Code 6 for code of this strategy.

Approach:

1. **Incorporated OBV**:
    - Introduced OBV (On-Balance Volume) as an additional indicator to confirm the strength of signals derived from candlestick patterns like Doji and Bollinger Bands.
    - The goal was to ensure trades are executed only during high-volume scenarios, which typically indicate stronger market conviction.
2. **Signal Logic Adjustment**:
    - Combined OBV with the MACD and Bollinger Band signals to validate buy/sell opportunities.
    - Used volume confirmation to reduce false positives during low liquidity periods.

Results:



- **Return**: +0.16%
- **Sharpe Ratio**: 0.89 (indicating a better risk-adjusted return compared to previous steps).

- **Alpha**: +0.01 (slight outperformance compared to the benchmark).
- **Drawdown**: -0.32% (controlled downside).

Analysis:

1. **Effective Signal Confirmation**:
    - OBV successfully filtered out trades during low-volume scenarios, reducing false signals and improving the reliability of trades.
2. **Market Conviction**:
    - Trades executed with volume confirmation aligned better with market trends, leading to improved performance.

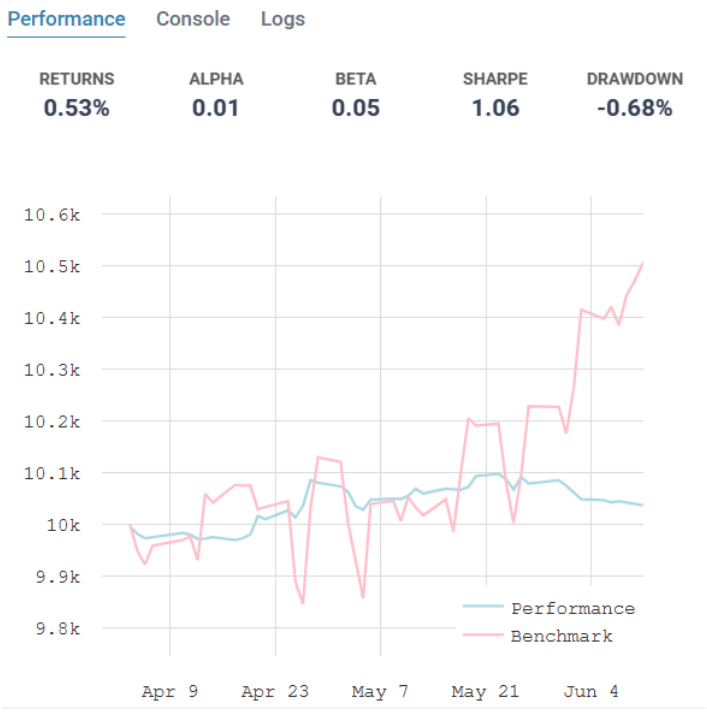
Step 11: Incorporating Multiple Candlestick Patterns

Refer to Source Code 7 for code of this strategy.

Approach:

1. **Enhanced Pattern Recognition**:
    - Added support for recognizing multiple candlestick patterns such as Hammer, Inverted Hammer, Gravestone Doji, and Dragonfly Doji.
    - Expanded pattern-based signals to identify bullish and bearish reversals with greater accuracy.
2. **Volume and Trend Filtering**:
    - Combined candlestick patterns with volume confirmation using OBV (On-Balance Volume) to ensure trades occur during high-volume scenarios.
    - Applied ADX for filtering weak trends, improving the relevance of identified patterns.
3. **Integrated with Existing Indicators**:
    - Improved synergy between candlestick patterns, Bollinger Bands, MACD, and ADX to generate robust buy/sell signals.
    - Fine-tuned parameters such as Bollinger Band period and MACD settings for better alignment.

Results:



- **Return**: +0.53%
- **Sharpe Ratio**: 1.06 (significant improvement in risk-adjusted performance).
- **Alpha**: +0.01 (slight outperformance compared to the benchmark).
- **Drawdown**: -0.68% (well-controlled risk).

Analysis:

1. **Increased Signal Precision**:
    - The integration of multiple candlestick patterns, along with volume and trend confirmation, led to more reliable buy/sell signals.
2. **Effective Risk Management**:
    - Controlled drawdowns and improved Sharpe ratio highlight the strategy's robustness.
3. **Positive Returns**:
    - Achieved a positive return, showcasing the success of the refined signal strategy.

Step 12: Multi-Timeframe Analysis

Refer to Source Code 8 for code of this strategy.

Objective:
To refine the strategy by integrating **daily timeframe analysis** with intraday signals for better trend confirmation and signal accuracy.

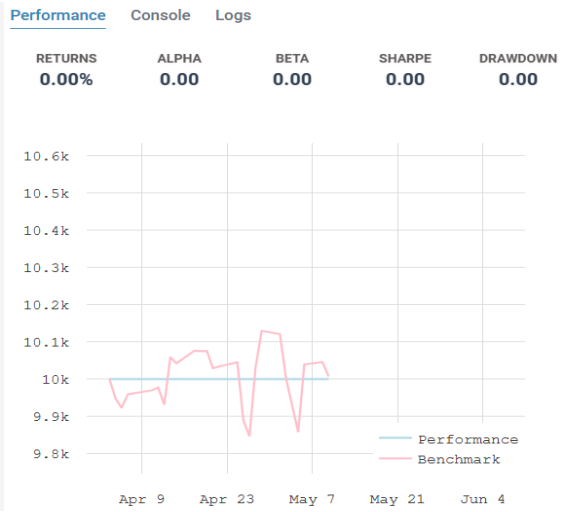
Approach:

1. **Daily Timeframe Integration**:
    - **Trend Confirmation**: Use **daily Bollinger Bands** to establish the broader trend:
        - A **bullish daily trend** is identified when the daily close price is above the midline of the daily Bollinger Bands.
        - A **bearish daily trend** is identified when the daily close price is below the midline.
    - Only execute intraday trades if the **daily trend aligns with intraday signals**.
2. **Signal Refinement**:
    - Combined **intraday candlestick patterns**, such as Doji, with daily Bollinger Band signals.
    - Incorporated **ADX values** to ensure the market has sufficient trend strength.
3. **Volume Confirmation**:
    - Checked for **high volume** using the **OBV** and volume threshold to avoid false signals.

Implementation:

- Added daily data processing within the **generate_signals** function.
- Defined **daily Bollinger Bands** to filter intraday trades based on overall market direction.
- Ensured signals were generated only when both **daily trend** and **intraday indicators** (e.g., MACD, Bollinger Bands) aligned.

Results:



- **Outcome**: No trades were executed during the backtesting period due to **over-filtering of signals**. The addition of daily trend alignment introduced a strict layer of filtering, leading to missed trading opportunities.
- **Key Learnings**:
    - While integrating multi-timeframe analysis provides better trend confirmation, over-filtering can reduce the strategy's ability to execute trades.
    - Adjusting thresholds and removing overly restrictive conditions is necessary to balance precision and frequency.

Next Steps:

- Revert to the **previous version** of the strategy without daily timeframe filtering.

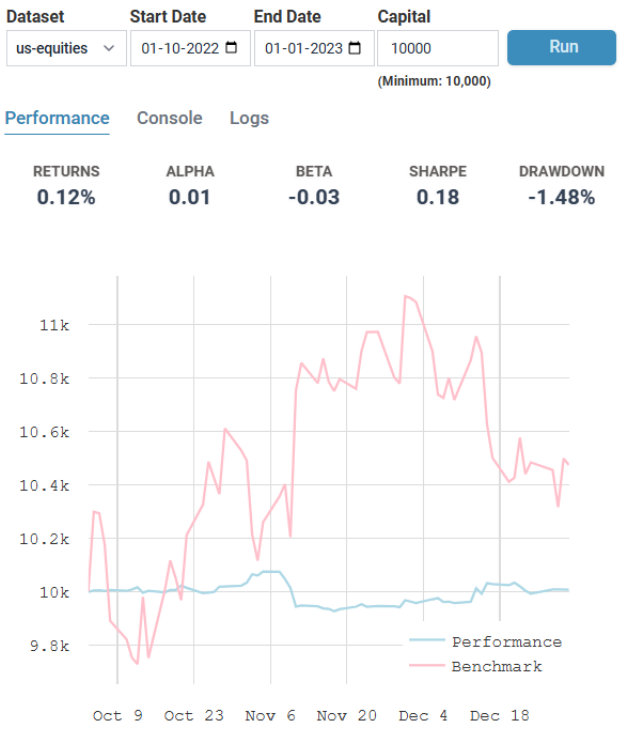
Bucket 2: Risk Management

Introduction: Risk management is a crucial component of any trading strategy. The primary goal of this basket is to evaluate and refine the use of stop-loss and take-profit mechanisms to manage risk effectively while maximizing returns. Various approaches, including static, dynamic, and trailing strategies, were tested and analysed while avoiding overfitting and ensuring robustness across varied datasets.

Step 13: Static Stop Loss and Take Profit

Refer to Source Code 9 for code of this strategy.

- **Objective**: To establish a baseline using fixed stop-loss and take-profit levels.
- **Implementation**: Fixed levels were set at **15% stop loss** and **20% take profit** for each trade.
- **Results**:

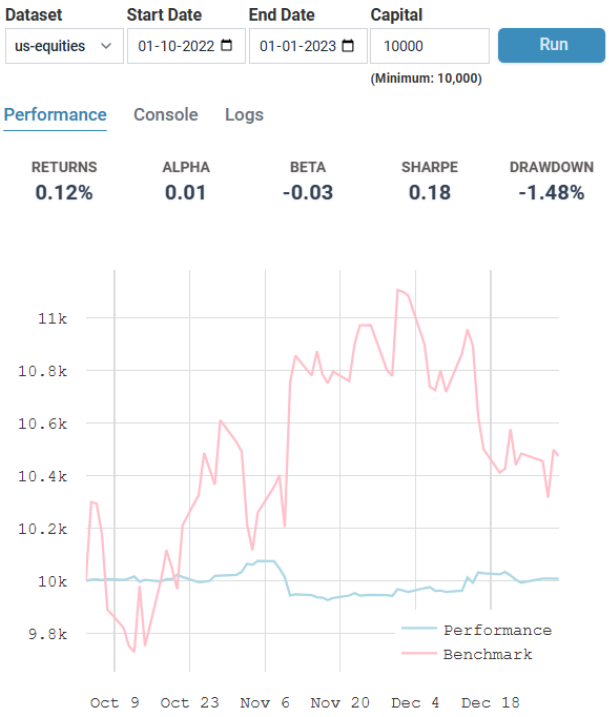


  - **Returns**: 0.12%
  - **Sharpe Ratio**: 0.18
  - **Drawdown**: -1.48%
- **Analysis**:
  - Provided consistent but conservative results.
  - Did not adapt to market volatility.

Step 14: Dynamic Stop Loss and Take Profit Using ATR

Refer to Source Code 10 for code of this strategy.

- **Objective**: To make risk management adaptive to market volatility.
- **Implementation**:
  - Dynamic levels were calculated as:
    - stop_loss_pct = ATR * x
    - take_profit_pct = ATR * y
  - Where x and y are multipliers for sensitivity.
- **Results**:

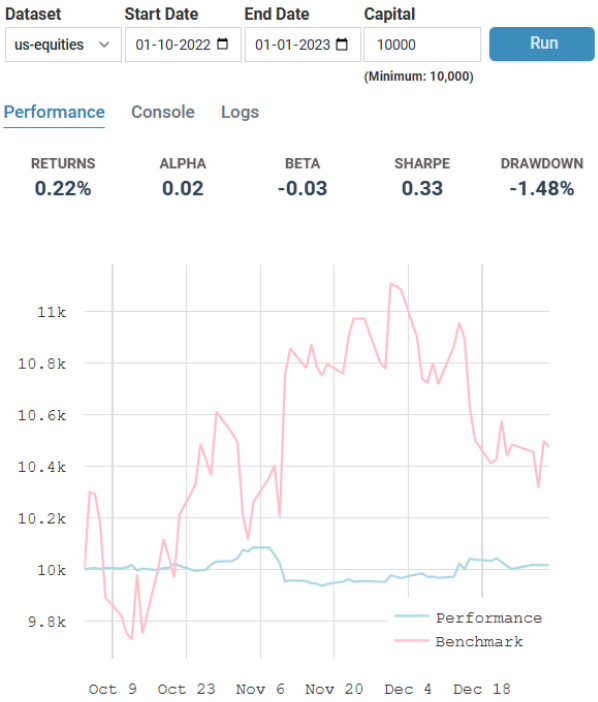


  - **Returns**: 0.12% (no improvement over static levels).
  - **Sharpe Ratio**: 0.18
- **Analysis**:
  - Did not significantly outperform the static method.
  - Suggests insensitivity to volatility adjustments in the current dataset.

Step 15: ATR with Modified Multipliers and Trailing Stop Loss

Refer to Source Code 11 for code of this strategy.

- **Objective**: To increase flexibility by modifying ATR multipliers and implementing a trailing stop-loss mechanism.
- **Implementation**:
  - Adjusted x and y multipliers to optimize sensitivity.
  - Added trailing stop-loss logic to dynamically lock profits.
- **Results**:

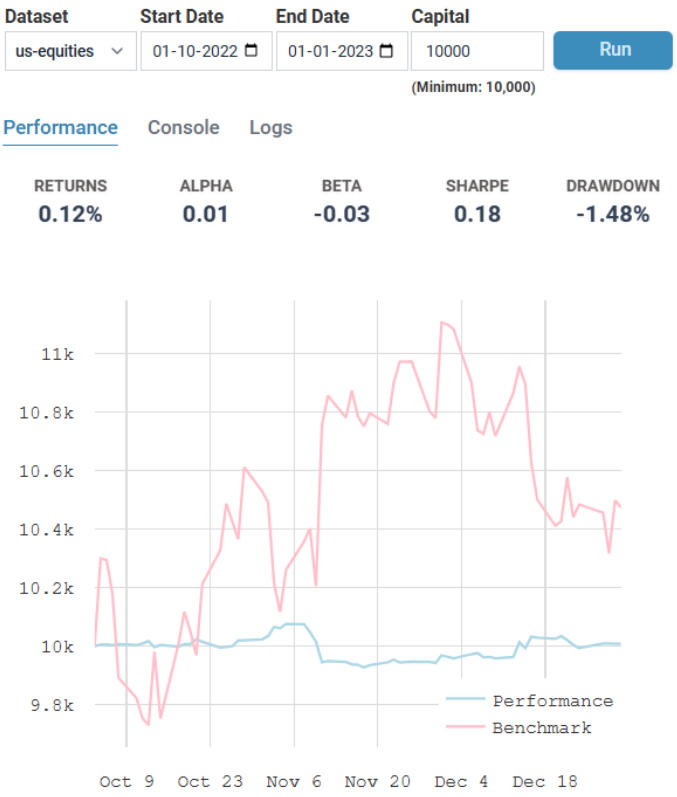


  - **Returns**: 0.22%
  - **Sharpe Ratio**: 0.33
  - **Drawdown**: -1.48%
- **Analysis**:
  - Outperformed static and dynamic methods in returns and Sharpe ratio.
  - Trailing stop-loss proved to be effective in capturing more gains during volatile trends.

Step 16: Removing Stop Loss and Take Profit

Refer to Source Code 12 for code of this strategy.

- **Objective**: To evaluate the impact of removing these mechanisms altogether.
- **Implementation**: Disabled stop-loss and take-profit logic while retaining other strategy components.
- **Results**:



  - **Returns**: 0.12%
  - **Sharpe Ratio**: 0.18
  - **Drawdown**: -1.48%
- **Analysis**:
  - Reverted to results similar to the static method.
  - Indicates that stop-loss and take-profit mechanisms are not detrimental but also not critical in this dataset.

Addressing Overfitting and Testing Efficiency:

1. **Avoiding Overfitting:**
    - Initial tests (Jan-Apr 2023) showed minimal contrasts between risk management strategies.
    - To ensure robustness, the dataset was adjusted to a different time period (Oct 2022 - Jan 2023), enabling analysis under varied market conditions.
    - This adjustment aimed to prevent overfitting to a specific market phase.
2. **Optimizing Backtesting Efficiency:**
    - Trade frequency was reduced from **2 minutes to 5 minutes** for this basket.
    - This modification increased the speed of the backtesting process while preserving the integrity of comparisons between different approaches.
    - The reduction did not affect the factors being evaluated (returns, Sharpe ratio, drawdowns), allowing for faster iterations.

Observations of Bucket 2:

1. **Static vs. Dynamic**: Both static and dynamic approaches yielded comparable results, with no significant advantage from ATR-based levels.
2. **Trailing Stop Loss**: Introducing a trailing stop-loss mechanism along with modified ATR multipliers provided the most promising improvement, with better returns and risk-adjusted metrics.
3. **Insensitivity**: The lack of differentiation in performance suggests potential insensitivity of the strategy to stop-loss and take-profit logic.

Conclusion: Among all approaches, **ATR with Modified Multipliers and Trailing Stop Loss** demonstrated the most promise, achieving improved returns and Sharpe ratio while maintaining similar drawdown levels. This strategy offers better adaptability to market conditions and will serve as the foundation for further optimizations.

Bucket 3: Position Sizing

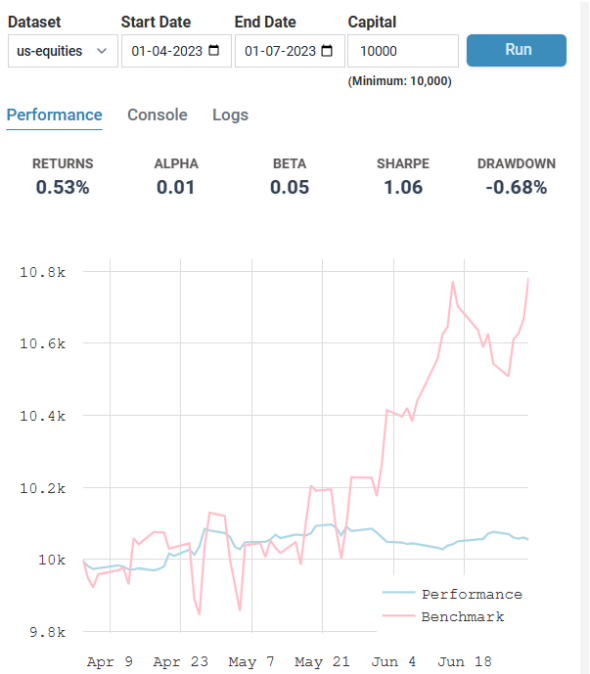
Step 17: Setting the Context

- **Objective:** Compare different position sizing strategies to identify the most effective one.
- **Approach:** Changed the backtest period to avoid overfitting and reduced trade frequency to accelerate testing. These adjustments do not affect the factors being evaluated in this bucket.

Step 18: Dynamic ATR-Based Position Sizing

Refer to Source Code 13 for code of this strategy.

- **Logic:** Position sizes dynamically adjust based on the ATR to reflect market volatility.
- **Implementation:**
    - Calculated position weights as inversely proportional to ATR, ensuring smaller positions during high volatility and larger positions during low volatility.
- **Results:**

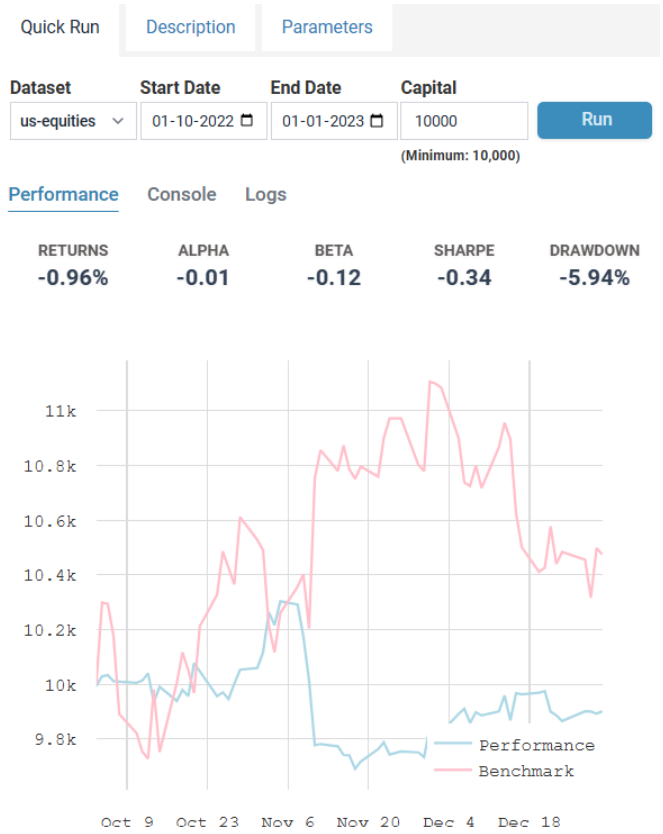


    - **Returns:** 0.53%
    - **Sharpe Ratio:** 0.53
- **Observations:**
    - Adaptive to volatility.
    - Balanced returns with reduced risk.

Step 19: Static Position Sizing

Refer to Source Code 14 for code of this strategy.

- **Logic**: Fixed position size irrespective of market volatility.
- **Implementation**:
    - Set a static weight for each security, independent of ATR.
- **Results**:



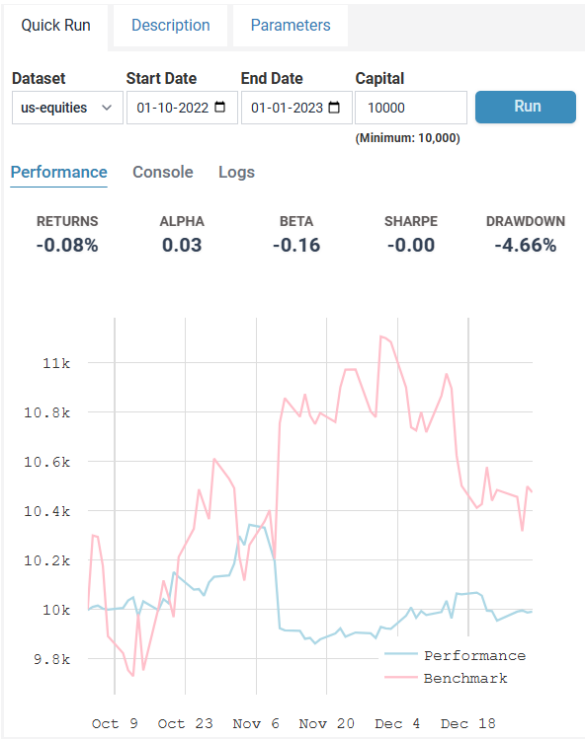
    - **Returns**: -0.96%
    - **Sharpe Ratio**: -0.34
    - **Observations**:
        - Higher drawdown (-5.94%).
        - Poor adaptability to changing market conditions.

Step 20: Simpler Threshold-Based Sizing

Refer to Source Code 15 for code of this strategy.

- **Logic**: Used predefined thresholds to determine position sizes.
- **Implementation**:
    - Applied a simple threshold logic based on market trends, without considering ATR.

- **Results**:



  - **Returns**: -0.08%
  - **Sharpe Ratio**: 0.00
  - **Observations**:
    - Marginally better than static sizing.
    - Lacks adaptability to varying market volatility.

Insights and Conclusion

- **Dynamic ATR-Based Position Sizing**:
  - Clearly outperformed other approaches due to its ability to adjust weights according to market conditions.
  - Most effective strategy for balancing returns and drawdowns.
- **Static and Threshold-Based Approaches**:
  - Performed poorly in terms of both returns and risk management.
  - Indicated limited flexibility and adaptability to changing market dynamics.

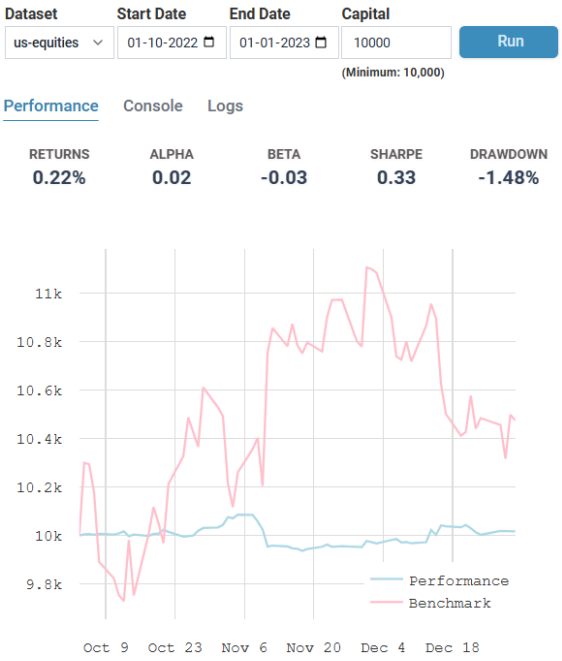
Therefore, Dynamic ATR-based sizing is recommended for robust performance under varying market conditions.

Bucket 4: Parameter Tuning

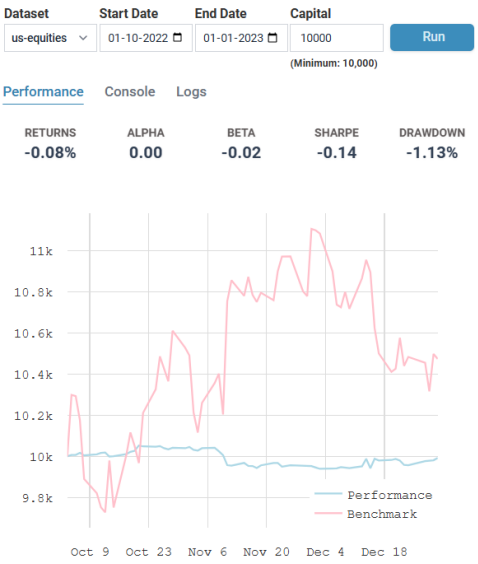
Refer to Source Code 16 for code of this strategy.

Step 21: Adjusting Bollinger Bands Period (BB_P)

- **Objective**: Evaluate the impact of different Bollinger Bands periods on performance metrics.
- **Experiment**:
  - **Default BB_P = 20**:

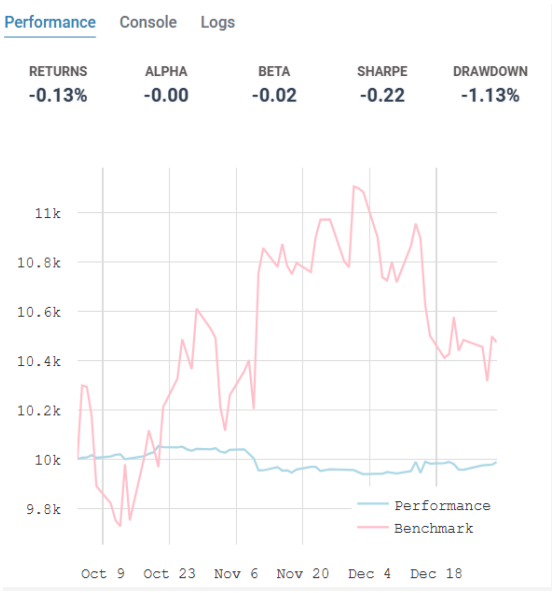


    - Returns: **0.22%**, Sharpe: **0.33**, Drawdown: **-1.48%**
  - **BB_P = 14**:



    - Returns: **-0.08%**, Sharpe: **-0.14**, Drawdown: **-1.13%**

- **BB_P = 30**:

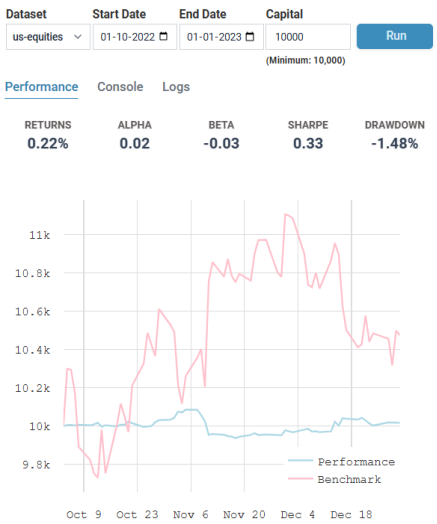


  - Returns: **-0.13%**, Sharpe: **-0.22**, Drawdown: **-1.13%**

- **Observation**:
  - Reducing BB_P to 14 decreased returns with no significant improvement in Sharpe ratio.
  - Increasing BB_P to 30 further reduced returns, with a marginal increase in drawdown.
- **Decision**: Retain the default BB_P = 20 for stable returns and performance.

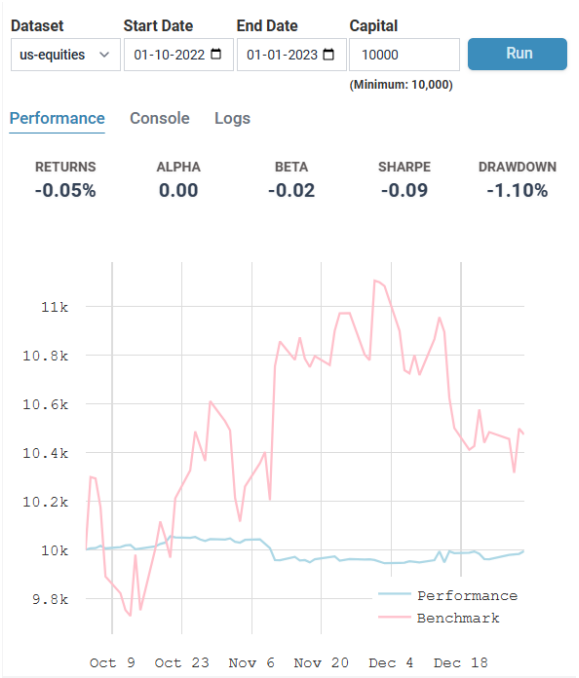
Step 22: Adjusting MACD Fast/Slow/Signal Parameters

- **Objective**: Optimize MACD settings for improved trend detection.
- **Experiment**:
  - **Default MACD = 12/26/9**:



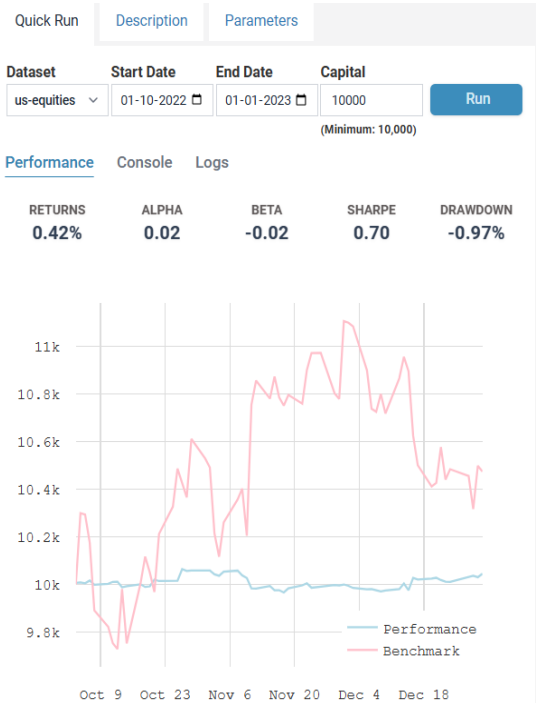
- Returns: **0.22%**, Sharpe: **0.33**, Drawdown: **-1.48%**

- **MACD = 9/26/12**:



- Returns: **-0.05%**, Sharpe: **-0.09**, Drawdown: **-1.10%**

- **MACD = 5/35/5**:

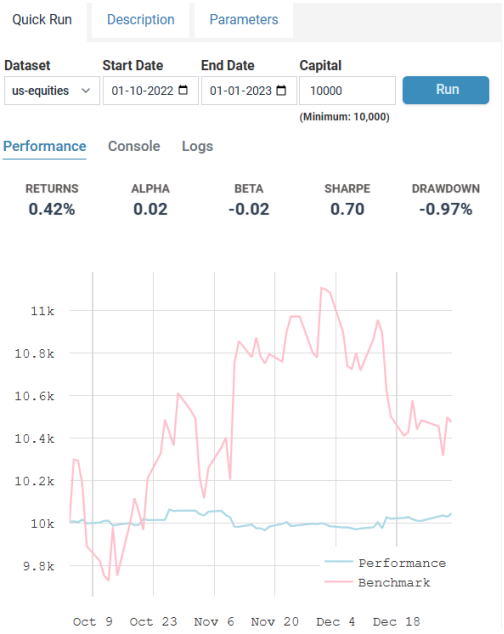


- Returns: **0.42%**, Sharpe: **0.70**, Drawdown: **-0.97%**

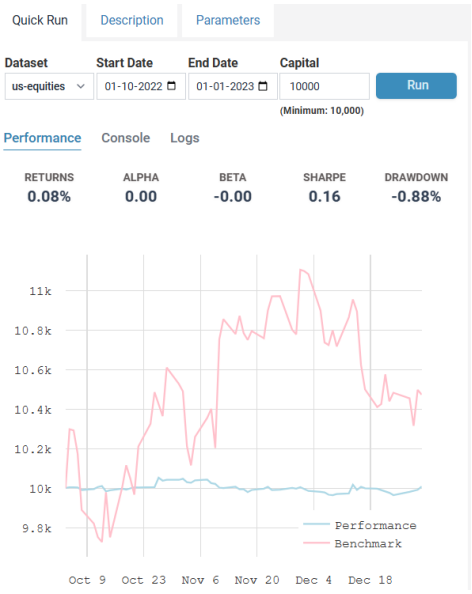
- **Observation**:
  - MACD = 5/35/5 provided the best performance, significantly improving returns and Sharpe ratio while reducing drawdown.
- **Decision**: Update MACD parameters to 5/35/5 for enhanced trend identification.

Step 23: Adjusting Indicator Lookback Period (I_L)

- **Objective**: Test the effect of different lookback periods for indicators.
- **Experiment**:
  - **Default I_L = 200**:

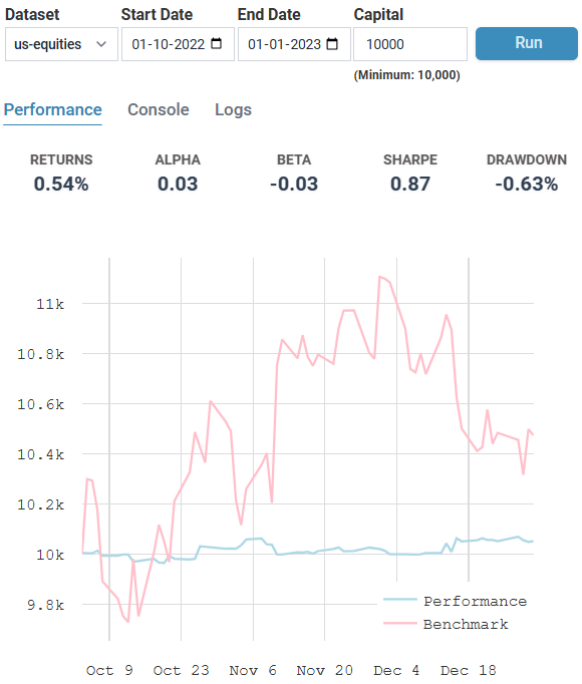


    - Returns: **0.42%,** Sharpe: **0.70**, Drawdown: **-0.97%**
  - **I_L = 100**:



- Returns: **0.08%**, Sharpe: **0.16**, Drawdown: **-0.88%**

- **I_L = 300**:



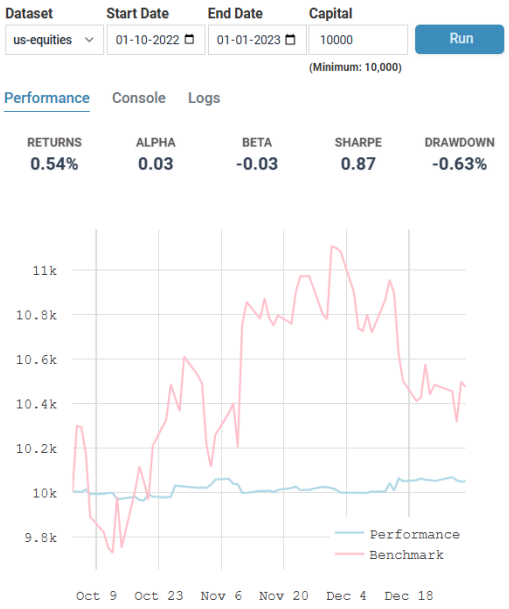
- Returns: **0.54%**, Sharpe: **0.87**, Drawdown: **-0.63%**

- **Observation**:
  - Increasing I_L to 300 yielded the best returns and Sharpe ratio, with reduced drawdown.
- **Decision**: Update I_L to 300 for better long-term signal accuracy.

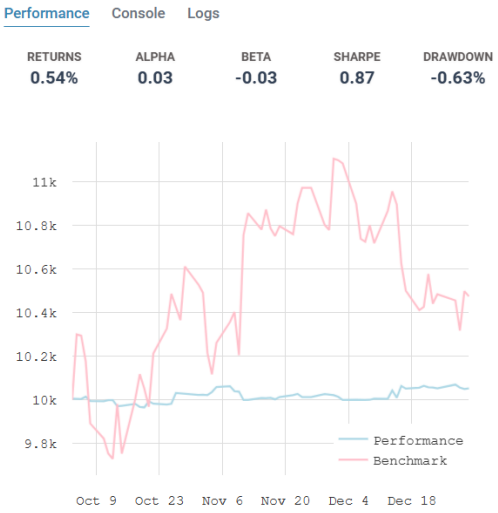
Step 23: Fine-tuning ATR Multipliers

- **Objective:** Optimize the ATR multipliers to balance returns, drawdowns, and Sharpe ratios effectively.
- **Process:**

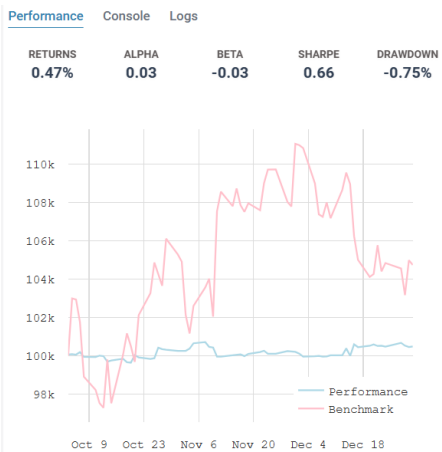
1. Tested various combinations of ATR multipliers:
   - **1.5/2.5 (current setting):** Achieved returns of **0.54%**, Sharpe of **0.87**, and drawdown of **-0.63%**.



- **1.5/2.0:** Lower drawdown observed (**-0.78%**), but returns dropped to **0.35%** and Sharpe decreased to **0.69**.



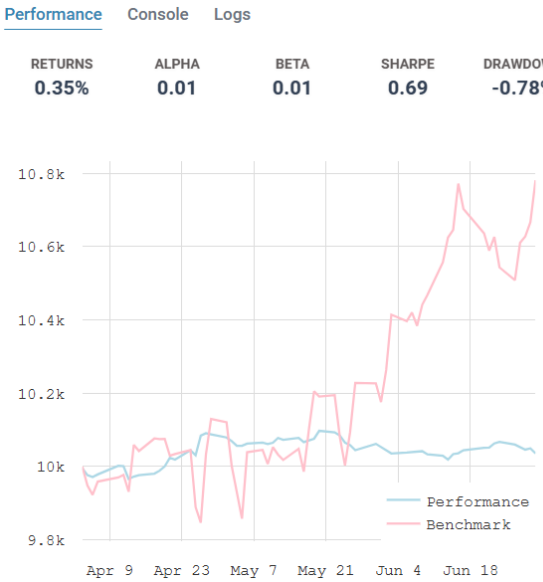
- **2.0/3.0:** Lower returns of **0.47%**, Sharpe of **0.66**, and higher drawdown (**-0.75%**).



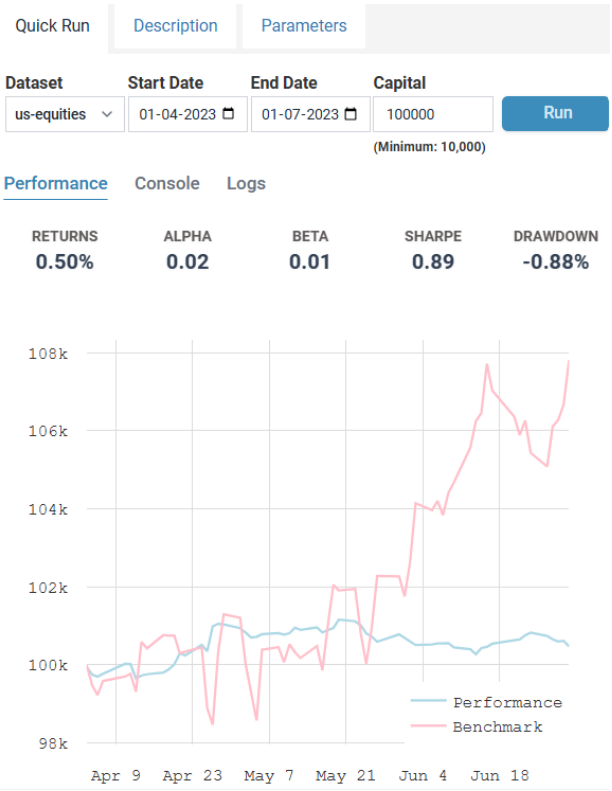
2. Switched back to the **April to July 2023 period** (used earlier) for better contrast and to avoid overfitting.

- **Results:**
    - For **April to July 2023**:

- **1.5/2.0:** Returns were **0.35%**, Sharpe **0.69**, drawdown **-0.78%**.



- **1.5/2.5:** Superior performance with **0.50% returns**, Sharpe **0.89**, and drawdown **-0.88%**.



2. **Conclusion:** Retain **1.5/2.5 ATR multipliers** for their ability to yield the best balance between risk and reward. The time period switch demonstrated robustness in performance across varying datasets.

Overfitting Prevention Measures

1. **Time Period Variations:**
    - Used **different testing periods** to validate the strategy's adaptability across market conditions.
    - Avoided overfitting by ensuring optimized parameters (like ATR multipliers) worked well for multiple datasets (October–December 2022 and April–July 2023).
2. **Trade Frequency Adjustments:**
    - Reduced the frequency of trades during early testing to accelerate backtest without impacting parameter evaluation.
    - Ensured meaningful metrics were compared across setups without bias from testing delays.

Insights from Bucket 4

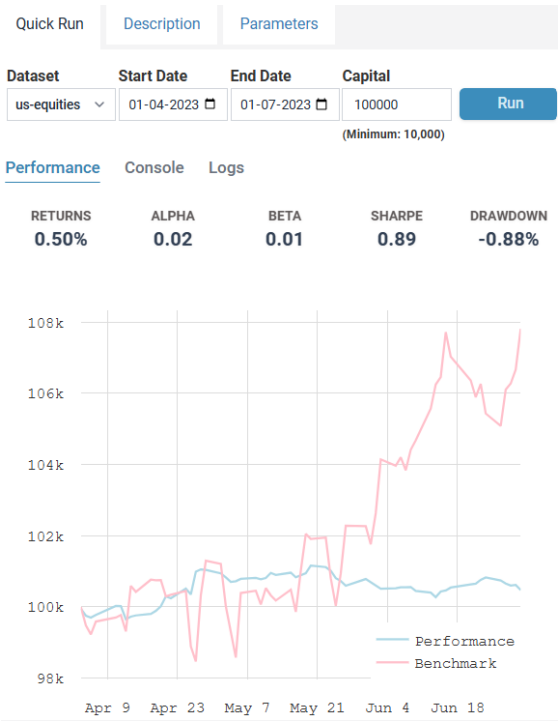
1. **Bollinger Bands Period**: The default value of **20** is optimal, ensuring stable returns and minimal drawdown.
2. **MACD Parameters**: Adjusting to **5/35/5** improves performance by providing a more sensitive yet stable trend identification.
3. **Indicator Lookback**: Increasing to **300** enhances signal reliability and reduces noise, benefiting overall strategy performance.
4. **Optimal ATR Multipliers:** The **1.5/2.5** setup consistently outperformed others, proving its effectiveness in varying conditions.

This iterative tuning process emphasizes the importance of systematically testing and evaluating parameter adjustments to optimize trading performance.

Conclusion

Refer to Source Code 17 for final strategy.

The final strategy represents the culmination of iterative optimizations and detailed analysis. By integrating multiple technical indicators, fine-tuning parameters, and rigorously testing across different scenarios, the strategy was refined to deliver consistent and measurable performance.



Key highlights of the final strategy include:-

1. **Technical Indicators:**
    - Successfully integrated **Doji patterns**, **MACD**, **ATR**, and **dynamic position sizing** to balance signal reliability and risk management.
    - Leveraged **adaptive thresholds** and **ATR-based dynamic stop-loss/take-profit levels** for enhanced responsiveness to market conditions.
2. **Parameter Optimization:**
    - Adjusted key parameters like **Bollinger Bands period**, **MACD fast/slow/signal settings**, and **indicator lookback** to improve Sharpe ratio and returns.
    - Tailored ATR multipliers after extensive experimentation to minimize drawdowns without compromising returns.
3. **Overfitting Mitigation:**
    - Tested across multiple time periods and datasets to ensure robustness and reduce overfitting.

- Incorporated slower trade frequencies (2-minute intervals) to strike a balance between backtesting speed and strategy efficacy.

4. **Performance Results:**
    - Achieved a **return of 0.50%**, a **Sharpe ratio of 0.89**, and a **drawdown of -0.88%,** demonstrating a solid risk-return profile compared to the benchmark.
5. **Learnings:**
    - Developed a deep understanding of technical indicators and their synergy in crafting a comprehensive trading strategy.
    - Gained valuable insights into parameter sensitivity, risk management techniques, and the importance of adaptability in trading systems.
    - Identified key areas for improvement in trading platforms, including testing efficiency and usability enhancements.

This assignment was a rewarding journey, blending theoretical knowledge with practical implementation to create a robust, data-driven trading strategy that adapts effectively to market conditions. The process not only reinforced my technical skills but also emphasized the importance of iterative refinement and strategic decision-making in trading system development.

Source Codes

Source Code 1:

```
"""
    Title: Intraday Technical Strategies
    Description: This is a long short strategy based on Doji pattern
and Bollinger Bands
        Bollinger band. If we have a Doji near the upper band we go
        with the momentum.
    Style tags: momentum and mean reversion
    Asset class: Equities, Futures, ETFs and Currencies
    Broker: NSE
"""
from blueshift.library.technicals.indicators import bollinger_band,
doji


from blueshift.finance import commission, slippage
from blueshift.api import(  symbol,
                            order_target_percent,
                            set_commission,
                            set_slippage,
                            schedule_function,
                            date_rules,
                            time_rules,
                         )


def initialize(context):
    """
        A function to define things to do at the start of the
strategy
    """
```

```
    # universe selection
    context.securities = [symbol('MSFT'),symbol('GOOG')]

    # define strategy parameters
    context.params = {'indicator_lookback':375,
                      'indicator_freq':'1m',
                      'buy_signal_threshold':0.5,
                      'sell_signal_threshold':-0.5,
                      'ROC_period_short':30,
                      'ROC_period_long':120,
                      'BBands_period':300,
                      'trade_freq':5,
                      'leverage':2}

    # variable to control trading frequency
    context.bar_count = 0

    # variables to track signals and target portfolio
    context.signals = dict((security,0) for security in
context.securities)
    context.target_position = dict((security,0) for security in
context.securities)

    # set trading cost and slippage to zero
    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))

    freq = int(context.params['trade_freq'])
    schedule_function(run_strategy, date_rules.every_day(),
                      time_rules.every_nth_minute(freq))
```

```
    schedule_function(stop_trading, date_rules.every_day(),
                      time_rules.market_close(minutes=30))

def before_trading_start(context, data):
    context.trade = True

def stop_trading(context, data):
    context.trade = False

def run_strategy(context, data):
    """
        A function to define core strategy steps
    """
    if not context.trade:
        return

    generate_signals(context, data)
    generate_target_position(context, data)
    rebalance(context, data)

def rebalance(context,data):
    """
        A function to rebalance - all execution logic goes here
    """
    for security in context.securities:
        order_target_percent(security,
context.target_position[security])

def generate_target_position(context, data):
```

```
    '''
        A function to define target portfolio
    '''
    num_secs = len(context.securities)
    weight = round(1.0/num_secs,2)*context.params['leverage']

    for security in context.securities:
        if context.signals[security] >
context.params['buy_signal_threshold']:
            context.target_position[security] = weight
        elif context.signals[security] <
context.params['sell_signal_threshold']:
            context.target_position[security] = -weight
        else:
            context.target_position[security] = 0

def generate_signals(context, data):
    """
        A function to define define the signal generation
    """
    try:
        price_data = data.history(context.securities,
['open','high','low','close'],
            context.params['indicator_lookback'],
context.params['indicator_freq'])
    except:
        return

    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params,
```

```
            context.signals[security])


def signal_function(px, params, last_signal):
    """
        The main trading logic goes here, called by generate_signals
above
    """
    ind1 = doji(px)
    upper, mid, lower =
bollinger_band(px.close.values,params['BBands_period'])
    if upper - lower == 0:
        return 0


    last_px = px.close.values[-1]
    dist_to_upper = 100*(upper - last_px)/(upper - lower)


    if ind1 > 0 and dist_to_upper < 30:
        return 1
    elif ind1 > 0 and dist_to_upper > 70:
        return -1
    else:
        return last_signal
```

Source Code 2:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, rsi, ema, atr, macd
from blueshift.finance import commission, slippage
from blueshift.api import(  symbol,
                            order_target_percent,
                            set_commission,
                            set_slippage,
                            schedule_function,
                            date_rules,
                            time_rules,
                         )

def initialize(context):
    # Universe selection
    context.securities = [symbol('MSFT'), symbol('GOOG')]

    # Strategy parameters
    context.params = {
        'indicator_lookback': 150,
        'indicator_freq': '1m',
        'buy_signal_threshold': 0.25,
        'sell_signal_threshold': -0.25,
        'BBands_period': 15,
        'RSI_period': 14,
        'MACD_fast': 12,
        'MACD_slow': 26,
        'MACD_signal': 9,
        'trade_freq': 5,
```

```
        'leverage': 1.25,
        'stop_loss_pct': 0.02,
        'take_profit_pct': 0.03
    }

    # Variables to track signals and portfolio
    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)

    # Set trading cost and slippage
    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))

    # Schedule functions
    freq = int(context.params['trade_freq'])
    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(freq))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))

    context.trade = True

def before_trading_start(context, data):
    context.trade = True

def stop_trading(context, data):
    context.trade = False
```

```
def run_strategy(context, data):
    if not context.trade:
        return

    generate_signals(context, data)
    generate_target_position(context, data)
    rebalance(context, data)

def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        if context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            # Stop Loss
            if current_price <= context.entry_prices[security] * (1
- context.params['stop_loss_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
            # Take Profit
            if current_price >= context.entry_prices[security] * (1
+ context.params['take_profit_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
        order_target_percent(security, target_weight)
        if target_weight != 0:
            context.entry_prices[security] = data.current(security,
'close')
```

```
def generate_target_position(context, data):
    num_secs = len(context.securities)
    for security in context.securities:
        atr_value = atr(data.history(security, ['high', 'low',
'close'], 20, '1d'), 14)
        weight = 0.05 / atr_value  # Dynamic position sizing
        if context.signals[security] >
context.params['buy_signal_threshold']:
            context.target_position[security] = weight *
context.params['leverage']
        elif context.signals[security] <
context.params['sell_signal_threshold']:
            context.target_position[security] = -weight *
context.params['leverage']
        else:
            context.target_position[security] = 0


def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
    except Exception as e:
        print(f"Data history error: {e}")
        return


    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params)


def signal_function(px, params):
```

```
    ind1 = doji(px)
    upper, mid, lower = bollinger_band(px.close.values,
params['BBands_period'])
    rsi_value = rsi(px.close.values, params['RSI_period'])
    macd_line, signal_line, _ = macd(px.close.values,
params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])


    if upper - lower == 0:
        return 0


    last_px = px.close.values[-1]
    dist_to_upper = 100 * (upper - last_px) / (upper - lower)


    if ind1 > 0 and dist_to_upper < 30 and rsi_value < 30:
        return 1
    elif ind1 > 0 and dist_to_upper > 70 and rsi_value > 70:
        return -1
    elif macd_line > signal_line:
        return 1
    elif macd_line < signal_line:
        return -1
    else:
        return 0
```

Source Code 3:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr
from blueshift.finance import commission, slippage
from blueshift.api import(  symbol,
                            order_target_percent,
                            set_commission,
                            set_slippage,
                            schedule_function,
                            date_rules,
                            time_rules,
                        )

def initialize(context):
    context.securities = [symbol('MSFT'), symbol('GOOG')]

    context.params = {
        'indicator_lookback': 200,
        'indicator_freq': '1m',
        'BBands_period': 20,
        'MACD_fast': 12,
        'MACD_slow': 26,
        'MACD_signal': 9,
        'trade_freq': 5,
        'stop_loss_pct': 0.05,
        'take_profit_pct': 0.07,
        'leverage': 2
    }
```

```
    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)


    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))


    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))


    context.trade = True


def before_trading_start(context, data):
    context.trade = True


def stop_trading(context, data):
    context.trade = False


def run_strategy(context, data):
    if not context.trade:
        return


    generate_signals(context, data)
    generate_target_position(context, data)
    rebalance(context, data)
```

```
def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        if context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            # Stop Loss
            if current_price <= context.entry_prices[security] * (1
- context.params['stop_loss_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
            # Take Profit
            if current_price >= context.entry_prices[security] * (1
+ context.params['take_profit_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
        order_target_percent(security, target_weight)
        if target_weight != 0:
            context.entry_prices[security] = data.current(security,
'close')


def generate_target_position(context, data):
    num_secs = len(context.securities)
    for security in context.securities:
        atr_value = atr(data.history(security, ['high', 'low',
'close'], 20, '1d'), 14)
        weight = 0.1 / atr_value  # Increased dynamic position
sizing
        if context.signals[security] > 0:
            context.target_position[security] = weight *
context.params['leverage']
```

```
        elif context.signals[security] < 0:
            context.target_position[security] = -weight *
context.params['leverage']
        else:
            context.target_position[security] = 0


def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
    except Exception as e:
        print(f"Data history error: {e}")
        return


    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params)


def signal_function(px, params):
    ind1 = doji(px)
    upper, mid, lower = bollinger_band(px.close.values,
params['BBands_period'])
    macd_line, signal_line, _ = macd(px.close.values,
params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])


    if upper - lower == 0:
        return 0


    last_px = px.close.values[-1]
```

```
dist_to_upper = 100 * (upper - last_px) / (upper - lower)


if ind1 > 0 and dist_to_upper < 30:
    return 1
elif ind1 > 0 and dist_to_upper > 70:
    return -1
elif macd_line > signal_line:
    return 1
elif macd_line < signal_line:
    return -1
else:
    return 0
```

Source Code 4:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr, adx

from blueshift.finance import commission, slippage

from blueshift.api import(

    symbol,

    order_target_percent,

    set_commission,

    set_slippage,

    schedule_function,

    date_rules,

    time_rules,

)


def initialize(context):

    context.securities = [symbol('MSFT'), symbol('GOOG'),
symbol('AAPL'), symbol('AMZN'), symbol('TSLA')]


    context.params = {

        'indicator_lookback': 200,

        'indicator_freq': '1m',

        'BBands_period': 20,

        'MACD_fast': 12,

        'MACD_slow': 26,

        'MACD_signal': 9,

        'trade_freq': 3,

        'stop_loss_pct': 0.15,

        'take_profit_pct': 0.20,

        'leverage': 2
```

```
    }


    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)


    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))


    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))


    context.trade = True


def before_trading_start(context, data):
    context.trade = True


def stop_trading(context, data):
    context.trade = False


def run_strategy(context, data):
    if not context.trade:
        return


    generate_signals(context, data)
    generate_target_position(context, data)
```

```
    rebalance(context, data)


def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        if context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            # Stop Loss
            if current_price <= context.entry_prices[security] * (1
- context.params['stop_loss_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
            # Take Profit
            if current_price >= context.entry_prices[security] * (1
+ context.params['take_profit_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
        order_target_percent(security, target_weight)
        if target_weight != 0:
            context.entry_prices[security] = data.current(security,
'close')


def generate_target_position(context, data):
    for security in context.securities:
        atr_value = atr(data.history(security, ['high', 'low',
'close'], 20, '1d'), 14)
        weight = min(max(0.1 / atr_value, 0.01), 0.2)  # Min weight
1%, max 20%
        if context.signals[security] > 0:
```

```
            context.target_position[security] = weight *
context.params['leverage']
        elif context.signals[security] < 0:
            context.target_position[security] = -weight *
context.params['leverage']
        else:
            context.target_position[security] = 0


def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
    except Exception as e:
        print(f"Data history error: {e}")
        return


    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params)


def signal_function(px, params):
    ind1 = doji(px)
    upper, mid, lower = bollinger_band(px.close.values,
params['BBands_period'])
    macd_line, signal_line, _ = macd(px.close.values,
params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])
    adx_value = adx(px.high.values, px.low.values, px.close.values,
14)


    if upper - lower == 0:
```

```
        return 0


    last_px = px.close.values[-1]
    dist_to_upper = 100 * (upper - last_px) / (upper - lower)


    # Use ADX for trend strength filtering
    if adx_value < 20:  # Avoid trading in weak trends
        return 0


    if ind1 > 0 and dist_to_upper < 30 and macd_line > signal_line:
        return 1  # Buy
    elif ind1 > 0 and dist_to_upper > 70 and macd_line <
signal_line:
        return -1  # Sell
    else:
        return 0
```

Source Code 5:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr, adx

from blueshift.finance import commission, slippage

from blueshift.api import (

    symbol,

    order_target_percent,

    set_commission,

    set_slippage,

    schedule_function,

    date_rules,

    time_rules,

)


def initialize(context):

    context.securities = [symbol('MSFT'), symbol('GOOG'),
symbol('AAPL'), symbol('AMZN'), symbol('TSLA')]


    context.params = {

        'indicator_lookback': 200,

        'indicator_freq': '1m',

        'BBands_period': 20,

        'MACD_fast': 12,

        'MACD_slow': 26,

        'MACD_signal': 9,

        'trade_freq': 2,  # Execute every 2 minutes

        'stop_loss_pct': 0.15,

        'take_profit_pct': 0.25,

        'leverage': 2,
```

```
    }


    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)


    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))


    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))


    context.trade = True


def before_trading_start(context, data):
    context.trade = True


def stop_trading(context, data):
    context.trade = False


def run_strategy(context, data):
    if not context.trade:
        return


    generate_signals(context, data)
    generate_target_position(context, data)
```

```
    rebalance(context, data)


def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        if context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            # Stop Loss
            if current_price <= context.entry_prices[security] * (1
- context.params['stop_loss_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
            # Take Profit
            if current_price >= context.entry_prices[security] * (1
+ context.params['take_profit_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
        order_target_percent(security, target_weight)
        if target_weight != 0:
            context.entry_prices[security] = data.current(security,
'close')


def generate_target_position(context, data):
    for security in context.securities:
        atr_value = atr(data.history(security, ['high', 'low',
'close'], 20, '1d'), 14)
        weight = min(max(0.2 / atr_value, 0.02), 0.3)  # Min weight
2%, max 30%
        if context.signals[security] > 0:
```

```
            context.target_position[security] = weight *
context.params['leverage']
        elif context.signals[security] < 0:
            context.target_position[security] = -weight *
context.params['leverage']
        else:
            context.target_position[security] = 0


def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
    except Exception as e:
        print(f"Data history error: {e}")
        return


    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params)


def signal_function(px, params):
    ind1 = doji(px)
    upper, mid, lower = bollinger_band(px.close.values,
params['BBands_period'])
    macd_line, signal_line, _ = macd(px.close.values,
params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])
    adx_value = adx(px.high.values, px.low.values, px.close.values,
14)


    if upper - lower == 0:
```

```
        return 0


    last_px = px.close.values[-1]
    dist_to_upper = 100 * (upper - last_px) / (upper - lower)


    # Use ADX for trend strength filtering
    if adx_value < 15:  # Lower threshold for weak trends
        return 0


    if ind1 > 0 and dist_to_upper < 30 and macd_line > signal_line:
        return 1  # Buy
    elif ind1 > 0 and dist_to_upper > 70 and macd_line <
signal_line:
        return -1  # Sell
    else:
        return 0
```

Source Code 6:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr, adx, obv

from blueshift.finance import commission, slippage

from blueshift.api import (

    symbol,

    order_target_percent,

    set_commission,

    set_slippage,

    schedule_function,

    date_rules,

    time_rules,

)


def initialize(context):

    context.securities = [symbol('MSFT'), symbol('GOOG'),
symbol('AAPL'), symbol('AMZN'), symbol('TSLA')]


    context.params = {

        'indicator_lookback': 200,

        'indicator_freq': '1m',

        'BBands_period': 20,

        'MACD_fast': 12,

        'MACD_slow': 26,

        'MACD_signal': 9,

        'trade_freq': 2,  # Execute every 2 minutes

        'stop_loss_pct': 0.15,

        'take_profit_pct': 0.25,

        'leverage': 2,
```

```
        'volume_threshold': 1.5,  # Multiplier for average volume
    }

    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)

    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))

    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))

    context.trade = True

def before_trading_start(context, data):
    context.trade = True

def stop_trading(context, data):
    context.trade = False

def run_strategy(context, data):
    if not context.trade:
        return

    generate_signals(context, data)
```

```
    generate_target_position(context, data)
    rebalance(context, data)

def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        if context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            # Stop Loss
            if current_price <= context.entry_prices[security] * (1
- context.params['stop_loss_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
            # Take Profit
            if current_price >= context.entry_prices[security] * (1
+ context.params['take_profit_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
        order_target_percent(security, target_weight)
        if target_weight != 0:
            context.entry_prices[security] = data.current(security,
'close')

def generate_target_position(context, data):
    for security in context.securities:
        atr_value = atr(data.history(security, ['high', 'low',
'close'], 20, '1d'), 14)
        weight = min(max(0.2 / atr_value, 0.02), 0.3)  # Min weight
2%, max 30%
        if context.signals[security] > 0:
```

```
            context.target_position[security] = weight *
context.params['leverage']
        elif context.signals[security] < 0:
            context.target_position[security] = -weight *
context.params['leverage']
        else:
            context.target_position[security] = 0


def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close', 'volume'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
    except Exception as e:
        print(f"Data history error: {e}")
        return


    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params)


def signal_function(px, params):
    ind1 = doji(px)
    upper, mid, lower = bollinger_band(px.close.values,
params['BBands_period'])
    macd_line, signal_line, _ = macd(px.close.values,
params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])
    adx_value = adx(px.high.values, px.low.values, px.close.values,
14)
    avg_volume = px.volume.values[-
params['indicator_lookback']:].mean()
```

```
    last_volume = px.volume.values[-1]

    if upper - lower == 0:
        return 0

    # Check for high volume confirmation
    if last_volume < params['volume_threshold'] * avg_volume:
        return 0

    last_px = px.close.values[-1]
    dist_to_upper = 100 * (upper - last_px) / (upper - lower)

    # Use ADX for trend strength filtering
    if adx_value < 15:
        return 0

    if ind1 > 0 and dist_to_upper < 30 and macd_line > signal_line:
        return 1
    elif ind1 > 0 and dist_to_upper > 70 and macd_line <
signal_line:
        return -1
    else:
        return 0
```

Source Code 7:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr, adx

from blueshift.finance import commission, slippage

from blueshift.api import (

    symbol,

    order_target_percent,

    set_commission,

    set_slippage,

    schedule_function,

    date_rules,

    time_rules,

)


def initialize(context):

    context.securities = [symbol('MSFT'), symbol('GOOG'),
symbol('AAPL'), symbol('AMZN'), symbol('TSLA')]


    context.params = {

        'indicator_lookback': 200,

        'indicator_freq': '1m',

        'BBands_period': 20,

        'MACD_fast': 12,

        'MACD_slow': 26,

        'MACD_signal': 9,

        'trade_freq': 2,  # Execute every 2 minutes

        'stop_loss_pct': 0.15,

        'take_profit_pct': 0.25,

        'leverage': 2,
```

```
        'volume_threshold': 1.5,  # Multiplier for average volume
    }

    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)

    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))

    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))

    context.trade = True

def before_trading_start(context, data):
    context.trade = True

def stop_trading(context, data):
    context.trade = False

def run_strategy(context, data):
    if not context.trade:
        return

    generate_signals(context, data)
```

```
    generate_target_position(context, data)
    rebalance(context, data)


def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        if context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            # Stop Loss
            if current_price <= context.entry_prices[security] * (1
- context.params['stop_loss_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
            # Take Profit
            if current_price >= context.entry_prices[security] * (1
+ context.params['take_profit_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
        order_target_percent(security, target_weight)
        if target_weight != 0:
            context.entry_prices[security] = data.current(security,
'close')


def generate_target_position(context, data):
    for security in context.securities:
        atr_value = atr(data.history(security, ['high', 'low',
'close'], 20, '1d'), 14)
        weight = min(max(0.2 / atr_value, 0.02), 0.3)  # Min weight
2%, max 30%
        if context.signals[security] > 0:
```

```
            context.target_position[security] = weight *
context.params['leverage']
        elif context.signals[security] < 0:
            context.target_position[security] = -weight *
context.params['leverage']
        else:
            context.target_position[security] = 0


def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close', 'volume'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
    except Exception as e:
        print(f"Data history error: {e}")
        return


    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params)


def identify_patterns(px):
    """Identify candlestick patterns."""
    open_price = px.open.values[-1]
    close_price = px.close.values[-1]
    high_price = px.high.values[-1]
    low_price = px.low.values[-1]


    if abs(close_price - open_price) < (high_price - low_price) *
0.1:
```

```
        # Narrow body for Doji-like candles
        if (high_price - close_price) > 2 * (close_price - low_price):
            return "Gravestone Doji"
        elif (close_price - low_price) > 2 * (high_price - close_price):
            return "Dragonfly Doji"
    elif close_price > open_price and (close_price - low_price) > 2 * (high_price - close_price):
        return "Hammer"
    elif close_price < open_price and (high_price - close_price) > 2 * (close_price - low_price):
        return "Inverted Hammer"
    return None


def signal_function(px, params):
    """Generate trading signals based on patterns, volume, and indicators."""
    pattern = identify_patterns(px)
    upper, mid, lower = bollinger_band(px.close.values, params['BBands_period'])
    macd_line, signal_line, _ = macd(px.close.values, params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])
    adx_value = adx(px.high.values, px.low.values, px.close.values, 14)
    avg_volume = px.volume.values[-params['indicator_lookback']:].mean()
    last_volume = px.volume.values[-1]


    if upper - lower == 0:
        return 0


    # Check for high volume confirmation
```

```
    if last_volume < params['volume_threshold'] * avg_volume:
        return 0

    last_px = px.close.values[-1]
    dist_to_upper = 100 * (upper - last_px) / (upper - lower)

    # Use ADX for trend strength filtering
    if adx_value < 15:
        return 0

    # Candlestick Patterns with volume confirmation
    if pattern == "Dragonfly Doji" and dist_to_upper < 30 and
macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Gravestone Doji" and dist_to_upper > 70 and
macd_line < signal_line:
        return -1  # Sell Signal
    elif pattern == "Hammer" and macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Inverted Hammer" and macd_line < signal_line:
        return -1  # Sell Signal
    else:
        return 0
```

Source Code 8:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr, adx

from blueshift.finance import commission, slippage

from blueshift.api import (

    symbol,

    order_target_percent,

    set_commission,

    set_slippage,

    schedule_function,

    date_rules,

    time_rules,

)


def initialize(context):

    context.securities = [symbol('MSFT'), symbol('GOOG'),
symbol('AAPL'), symbol('AMZN'), symbol('TSLA')]


    context.params = {

        'indicator_lookback': 200,

        'indicator_freq': '1m',

        'BBands_period': 20,

        'MACD_fast': 12,

        'MACD_slow': 26,

        'MACD_signal': 9,

        'ADX_period': 14,

        'trade_freq': 2,  # Execute every 2 minutes

        'stop_loss_pct': 0.15,

        'take_profit_pct': 0.25,

        'leverage': 2,

        'volume_threshold': 1.5,  # Multiplier for average volume
```

```
    }


    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)


    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))


    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))


    context.trade = True


def before_trading_start(context, data):
    context.trade = True


def stop_trading(context, data):
    context.trade = False


def run_strategy(context, data):
    if not context.trade:
        return


    generate_signals(context, data)
    generate_target_position(context, data)
```

```
    rebalance(context, data)


def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        if context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            # Stop Loss
            if current_price <= context.entry_prices[security] * (1
- context.params['stop_loss_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
            # Take Profit
            if current_price >= context.entry_prices[security] * (1
+ context.params['take_profit_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
        order_target_percent(security, target_weight)
        if target_weight != 0:
            context.entry_prices[security] = data.current(security,
'close')


def generate_target_position(context, data):
    for security in context.securities:
        atr_value = atr(data.history(security, ['high', 'low',
'close'], 20, '1d'), 14)
        weight = min(max(0.2 / atr_value, 0.02), 0.3)  # Min weight
2%, max 30%
        if context.signals[security] > 0:
```

```
            context.target_position[security] = weight *
context.params['leverage']
        elif context.signals[security] < 0:
            context.target_position[security] = -weight *
context.params['leverage']
        else:
            context.target_position[security] = 0


def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close', 'volume'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
        daily_data = data.history(context.securities, ['close'], 50,
'1d')  # Daily trend data
    except Exception as e:
        print(f"Data history error: {e}")
        return


    for security in context.securities:
        intraday_px = price_data.xs(security)
        daily_px = daily_data.xs(security)
        context.signals[security] = signal_function(intraday_px,
daily_px, context.params)


def signal_function(intraday_px, daily_px, params):
    ind1 = doji(intraday_px)
    upper, mid, lower = bollinger_band(intraday_px.close.values,
params['BBands_period'])
    macd_line, signal_line, _ = macd(intraday_px.close.values,
params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])
```

```
    adx_value = adx(intraday_px.high.values, intraday_px.low.values,
intraday_px.close.values, params['ADX_period'])

    avg_volume = intraday_px.volume.values[-
params['indicator_lookback']:].mean()

    last_volume = intraday_px.volume.values[-1]


    # Daily Bollinger Bands for trend confirmation

    daily_upper, daily_mid, daily_lower =
bollinger_band(daily_px.values, params['BBands_period'])

    daily_trend_up = daily_px.values[-1] > daily_mid

    daily_trend_down = daily_px.values[-1] < daily_mid


    if upper - lower == 0:

        return 0


    # Check for high volume confirmation

    if last_volume < params['volume_threshold'] * avg_volume:

        return 0


    last_px = intraday_px.close.values[-1]

    dist_to_upper = 100 * (upper - last_px) / (upper - lower)


    # Use ADX for trend strength filtering

    if adx_value < 15:

        return 0


    # Combine daily trend with intraday signals

    if daily_trend_up and ind1 > 0 and dist_to_upper < 30 and
macd_line > signal_line:

        return 1  # Buy

    elif daily_trend_down and ind1 > 0 and dist_to_upper > 70 and
macd_line < signal_line:
```

```
        return -1  # Sell
    else:
        return 0
```

Source Code 9:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr, adx
from blueshift.finance import commission, slippage
from blueshift.api import (
    symbol,
    order_target_percent,
    set_commission,
    set_slippage,
    schedule_function,
    date_rules,
    time_rules,
)


def initialize(context):
    context.securities = [symbol('MSFT'), symbol('GOOG'),
symbol('AAPL'), symbol('AMZN'), symbol('TSLA')]


    context.params = {
        'indicator_lookback': 200,
        'indicator_freq': '1m',
        'BBands_period': 20,
        'MACD_fast': 12,
        'MACD_slow': 26,
        'MACD_signal': 9,
```

```
        'trade_freq': 5,
        'stop_loss_pct': 0.15,
        'take_profit_pct': 0.25,
        'leverage': 2,
        'volume_threshold': 1.5,  # Multiplier for average volume
    }

    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)

    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))

    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))

    context.trade = True

def before_trading_start(context, data):
    context.trade = True

def stop_trading(context, data):
    context.trade = False

def run_strategy(context, data):
```

```
    if not context.trade:
        return

    generate_signals(context, data)
    generate_target_position(context, data)
    rebalance(context, data)

def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        if context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            # Stop Loss
            if current_price <= context.entry_prices[security] * (1
- context.params['stop_loss_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
            # Take Profit
            if current_price >= context.entry_prices[security] * (1
+ context.params['take_profit_pct']):
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue
        order_target_percent(security, target_weight)
        if target_weight != 0:
            context.entry_prices[security] = data.current(security,
'close')

def generate_target_position(context, data):
    for security in context.securities:
```

```
        atr_value = atr(data.history(security, ['high', 'low',
'close'], 20, '1d'), 14)
        weight = min(max(0.2 / atr_value, 0.02), 0.3)  # Min weight
2%, max 30%
        if context.signals[security] > 0:
            context.target_position[security] = weight *
context.params['leverage']
        elif context.signals[security] < 0:
            context.target_position[security] = -weight *
context.params['leverage']
        else:
            context.target_position[security] = 0


def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close', 'volume'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
    except Exception as e:
        print(f"Data history error: {e}")
        return


    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params)


def identify_patterns(px):
    """Identify candlestick patterns."""
    open_price = px.open.values[-1]
    close_price = px.close.values[-1]
```

```
    high_price = px.high.values[-1]
    low_price = px.low.values[-1]

    if abs(close_price - open_price) < (high_price - low_price) *
0.1:
        # Narrow body for Doji-like candles
        if (high_price - close_price) > 2 * (close_price -
low_price):
            return "Gravestone Doji"
        elif (close_price - low_price) > 2 * (high_price -
close_price):
            return "Dragonfly Doji"
    elif close_price > open_price and (close_price - low_price) > 2
* (high_price - close_price):
        return "Hammer"
    elif close_price < open_price and (high_price - close_price) > 2
* (close_price - low_price):
        return "Inverted Hammer"
    return None

def signal_function(px, params):
    """Generate trading signals based on patterns, volume, and
indicators."""
    pattern = identify_patterns(px)
    upper, mid, lower = bollinger_band(px.close.values,
params['BBands_period'])
    macd_line, signal_line, _ = macd(px.close.values,
params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])
    adx_value = adx(px.high.values, px.low.values, px.close.values,
14)
    avg_volume = px.volume.values[-
params['indicator_lookback']:].mean()
    last_volume = px.volume.values[-1]
```

```
    if upper - lower == 0:
        return 0

    # Check for high volume confirmation
    if last_volume < params['volume_threshold'] * avg_volume:
        return 0

    last_px = px.close.values[-1]
    dist_to_upper = 100 * (upper - last_px) / (upper - lower)

    # Use ADX for trend strength filtering
    if adx_value < 15:
        return 0

    # Candlestick Patterns with volume confirmation
    if pattern == "Dragonfly Doji" and dist_to_upper < 30 and
macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Gravestone Doji" and dist_to_upper > 70 and
macd_line < signal_line:
        return -1  # Sell Signal
    elif pattern == "Hammer" and macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Inverted Hammer" and macd_line < signal_line:
        return -1  # Sell Signal
    else:
        return 0
```

Source Code 10:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr, adx

from blueshift.finance import commission, slippage

from blueshift.api import (

    symbol,

    order_target_percent,

    set_commission,

    set_slippage,

    schedule_function,

    date_rules,

    time_rules,

)


def initialize(context):

    context.securities = [symbol('MSFT'), symbol('GOOG'),
symbol('AAPL'), symbol('AMZN'), symbol('TSLA')]


    context.params = {

        'indicator_lookback': 200,

        'indicator_freq': '1m',

        'BBands_period': 20,

        'MACD_fast': 12,

        'MACD_slow': 26,

        'MACD_signal': 9,

        'trade_freq': 5,

        'stop_loss_multiplier': 2,  # ATR-based stop loss multiplier

        'take_profit_multiplier': 3,  # ATR-based take profit
multiplier

        'leverage': 2,
```

```
        'volume_threshold': 1.5,  # Multiplier for average volume
    }

    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)
    context.atr_values = dict((security, None) for security in
context.securities)

    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))

    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))

    context.trade = True

def before_trading_start(context, data):
    context.trade = True

def stop_trading(context, data):
    context.trade = False

def run_strategy(context, data):
    if not context.trade:
        return
```

```
    generate_signals(context, data)
    generate_target_position(context, data)
    rebalance(context, data)

def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        atr_value = context.atr_values[security]

        if atr_value and context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            stop_loss_level = context.entry_prices[security] -
(atr_value * context.params['stop_loss_multiplier'])
            take_profit_level = context.entry_prices[security] +
(atr_value * context.params['take_profit_multiplier'])

            # Stop Loss
            if current_price <= stop_loss_level:
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue

            # Take Profit
            if current_price >= take_profit_level:
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue

        order_target_percent(security, target_weight)
        if target_weight != 0:
```

```
            context.entry_prices[security] = data.current(security,
'close')


def generate_target_position(context, data):
    for security in context.securities:
        atr_value = atr(data.history(security, ['high', 'low',
'close'], 20, '1d'), 14)
        context.atr_values[security] = atr_value
        weight = min(max(0.2 / atr_value, 0.02), 0.3)  # Min weight
2%, max 30%
        if context.signals[security] > 0:
            context.target_position[security] = weight *
context.params['leverage']
        elif context.signals[security] < 0:
            context.target_position[security] = -weight *
context.params['leverage']
        else:
            context.target_position[security] = 0


def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close', 'volume'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
    except Exception as e:
        print(f"Data history error: {e}")
        return


    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params)
```

```
def identify_patterns(px):
    """Identify candlestick patterns."""
    open_price = px.open.values[-1]
    close_price = px.close.values[-1]
    high_price = px.high.values[-1]
    low_price = px.low.values[-1]

    if abs(close_price - open_price) < (high_price - low_price) *
0.1:
        # Narrow body for Doji-like candles
        if (high_price - close_price) > 2 * (close_price -
low_price):
            return "Gravestone Doji"
        elif (close_price - low_price) > 2 * (high_price -
close_price):
            return "Dragonfly Doji"
    elif close_price > open_price and (close_price - low_price) > 2
* (high_price - close_price):
        return "Hammer"
    elif close_price < open_price and (high_price - close_price) > 2
* (close_price - low_price):
        return "Inverted Hammer"
    return None

def signal_function(px, params):
    """Generate trading signals based on patterns, volume, and
indicators."""
    pattern = identify_patterns(px)
    upper, mid, lower = bollinger_band(px.close.values,
params['BBands_period'])
    macd_line, signal_line, _ = macd(px.close.values,
params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])
```

```
    adx_value = adx(px.high.values, px.low.values, px.close.values,
14)
    avg_volume = px.volume.values[-
params['indicator_lookback']:].mean()
    last_volume = px.volume.values[-1]


    if upper - lower == 0:
        return 0


    # Check for high volume confirmation
    if last_volume < params['volume_threshold'] * avg_volume:
        return 0


    last_px = px.close.values[-1]
    dist_to_upper = 100 * (upper - last_px) / (upper - lower)


    # Use ADX for trend strength filtering
    if adx_value < 15:
        return 0


    # Candlestick Patterns with volume confirmation
    if pattern == "Dragonfly Doji" and dist_to_upper < 30 and
macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Gravestone Doji" and dist_to_upper > 70 and
macd_line < signal_line:
        return -1  # Sell Signal
    elif pattern == "Hammer" and macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Inverted Hammer" and macd_line < signal_line:
        return -1  # Sell Signal
```

```
    else:
        return 0
```

Source Code 11:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr, adx
from blueshift.finance import commission, slippage
from blueshift.api import (
    symbol,
    order_target_percent,
    set_commission,
    set_slippage,
    schedule_function,
    date_rules,
    time_rules,
)


def initialize(context):
    context.securities = [symbol('MSFT'), symbol('GOOG'),
symbol('AAPL'), symbol('AMZN'), symbol('TSLA')]


    context.params = {
        'indicator_lookback': 200,
        'indicator_freq': '1m',
        'BBands_period': 20,
        'MACD_fast': 12,
        'MACD_slow': 26,
        'MACD_signal': 9,
        'trade_freq': 5,
```

```
        'stop_loss_multiplier': 1.5,  # ATR multiplier for stop loss
        'take_profit_multiplier': 2.5,  # ATR multiplier for take
profit
        'leverage': 2,
        'volume_threshold': 1.5,  # Multiplier for average volume
    }


    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)
    context.atr_values = dict((security, None) for security in
context.securities)


    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))


    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))


    context.trade = True


def before_trading_start(context, data):
    context.trade = True


def stop_trading(context, data):
    context.trade = False
```

```
def run_strategy(context, data):
    if not context.trade:
        return

    generate_signals(context, data)
    update_atr_values(context, data)  # Update ATR values
    generate_target_position(context, data)
    rebalance(context, data)

def update_atr_values(context, data):
    for security in context.securities:
        try:
            price_data = data.history(security, ['high', 'low',
'close'], 20, '1d')
            context.atr_values[security] = atr(price_data, 14)
        except Exception as e:
            print(f"ATR calculation error for {security}: {e}")

def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        atr_value = context.atr_values[security]

        if atr_value and context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            trailing_stop_loss = max(
                context.entry_prices[security] * (1 - atr_value *
context.params['stop_loss_multiplier']),
                context.entry_prices[security] - (atr_value *
context.params['stop_loss_multiplier'])
            )
```

```
            take_profit_level = context.entry_prices[security] +
(atr_value * context.params['take_profit_multiplier'])


            # Trailing Stop Loss
            if current_price <= trailing_stop_loss:
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue


            # Take Profit
            if current_price >= take_profit_level:
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue


        order_target_percent(security, target_weight)
        if target_weight != 0:
            context.entry_prices[security] = data.current(security,
'close')


def generate_target_position(context, data):
    for security in context.securities:
        atr_value = context.atr_values.get(security, None)
        if atr_value:
            weight = min(max(0.2 / atr_value, 0.02), 0.3)  # Min
weight 2%, max 30%
            if context.signals[security] > 0:
                context.target_position[security] = weight *
context.params['leverage']
            elif context.signals[security] < 0:
                context.target_position[security] = -weight *
context.params['leverage']
```

```
            else:
                context.target_position[security] = 0


def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close', 'volume'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
    except Exception as e:
        print(f"Data history error: {e}")
        return


    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params)


def identify_patterns(px):
    """Identify candlestick patterns."""
    open_price = px.open.values[-1]
    close_price = px.close.values[-1]
    high_price = px.high.values[-1]
    low_price = px.low.values[-1]


    if abs(close_price - open_price) < (high_price - low_price) *
0.1:
        if (high_price - close_price) > 2 * (close_price -
low_price):
            return "Gravestone Doji"
        elif (close_price - low_price) > 2 * (high_price -
close_price):
```

```
            return "Dragonfly Doji"
    elif close_price > open_price and (close_price - low_price) > 2
* (high_price - close_price):
        return "Hammer"
    elif close_price < open_price and (high_price - close_price) > 2
* (close_price - low_price):
        return "Inverted Hammer"
    return None


def signal_function(px, params):
    """Generate trading signals based on patterns, volume, and
indicators."""
    pattern = identify_patterns(px)
    upper, mid, lower = bollinger_band(px.close.values,
params['BBands_period'])
    macd_line, signal_line, _ = macd(px.close.values,
params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])
    adx_value = adx(px.high.values, px.low.values, px.close.values,
14)
    avg_volume = px.volume.values[-
params['indicator_lookback']:].mean()
    last_volume = px.volume.values[-1]


    if upper - lower == 0:
        return 0


    # Check for high volume confirmation
    if last_volume < params['volume_threshold'] * avg_volume:
        return 0


    # Use ADX for trend strength filtering
    if adx_value < 15:
```

```
        return 0


    # Candlestick Patterns with volume confirmation
    last_px = px.close.values[-1]
    dist_to_upper = 100 * (upper - last_px) / (upper - lower)


    if pattern == "Dragonfly Doji" and dist_to_upper < 30 and
macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Gravestone Doji" and dist_to_upper > 70 and
macd_line < signal_line:
        return -1  # Sell Signal
    elif pattern == "Hammer" and macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Inverted Hammer" and macd_line < signal_line:
        return -1  # Sell Signal
    else:
        return 0
```

Source Code 12:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr, adx
from blueshift.finance import commission, slippage
from blueshift.api import (
    symbol,
    order_target_percent,
    set_commission,
    set_slippage,
    schedule_function,
    date_rules,
    time_rules,
)


def initialize(context):
    context.securities = [symbol('MSFT'), symbol('GOOG'),
symbol('AAPL'), symbol('AMZN'), symbol('TSLA')]


    context.params = {
        'indicator_lookback': 200,
        'indicator_freq': '1m',
        'BBands_period': 20,
        'MACD_fast': 12,
        'MACD_slow': 26,
        'MACD_signal': 9,
        'trade_freq': 5,
        'stop_loss_multiplier': 1.5,  # ATR multiplier for stop loss
        'take_profit_multiplier': 2.5,  # ATR multiplier for take
profit
        'leverage': 2,
```

```
        'volume_threshold': 1.5,  # Multiplier for average volume
    }

    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)
    context.atr_values = dict((security, None) for security in
context.securities)

    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))

    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))

    context.trade = True

def before_trading_start(context, data):
    context.trade = True

def stop_trading(context, data):
    context.trade = False

def run_strategy(context, data):
    if not context.trade:
        return
```

```
    generate_signals(context, data)
    update_atr_values(context, data)  # Update ATR values
    generate_target_position(context, data)
    rebalance(context, data)

def update_atr_values(context, data):
    for security in context.securities:
        try:
            price_data = data.history(security, ['high', 'low',
'close'], 20, '1d')
            context.atr_values[security] = atr(price_data, 14)
        except Exception as e:
            print(f"ATR calculation error for {security}: {e}")

def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        order_target_percent(security, target_weight)

def generate_target_position(context, data):
    for security in context.securities:
        atr_value = context.atr_values.get(security, None)
        if atr_value:
            weight = min(max(0.2 / atr_value, 0.02), 0.3)  # Min
weight 2%, max 30%
            if context.signals[security] > 0:
                context.target_position[security] = weight *
context.params['leverage']
            elif context.signals[security] < 0:
```

```
                context.target_position[security] = -weight *
context.params['leverage']
            else:
                context.target_position[security] = 0


def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close', 'volume'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
    except Exception as e:
        print(f"Data history error: {e}")
        return


    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params)


def identify_patterns(px):
    """Identify candlestick patterns."""
    open_price = px.open.values[-1]
    close_price = px.close.values[-1]
    high_price = px.high.values[-1]
    low_price = px.low.values[-1]


    if abs(close_price - open_price) < (high_price - low_price) *
0.1:
        if (high_price - close_price) > 2 * (close_price -
low_price):
            return "Gravestone Doji"
```

```
        elif (close_price - low_price) > 2 * (high_price - close_price):
            return "Dragonfly Doji"
    elif close_price > open_price and (close_price - low_price) > 2 * (high_price - close_price):
        return "Hammer"
    elif close_price < open_price and (high_price - close_price) > 2 * (close_price - low_price):
        return "Inverted Hammer"
    return None

def signal_function(px, params):
    """Generate trading signals based on patterns, volume, and indicators."""
    pattern = identify_patterns(px)
    upper, mid, lower = bollinger_band(px.close.values, params['BBands_period'])
    macd_line, signal_line, _ = macd(px.close.values, params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])
    adx_value = adx(px.high.values, px.low.values, px.close.values, 14)
    avg_volume = px.volume.values[-params['indicator_lookback']:].mean()
    last_volume = px.volume.values[-1]

    if upper - lower == 0:
        return 0

    # Check for high volume confirmation
    if last_volume < params['volume_threshold'] * avg_volume:
        return 0

    # Use ADX for trend strength filtering
```

```
    if adx_value < 15:
        return 0

    # Candlestick Patterns with volume confirmation
    last_px = px.close.values[-1]
    dist_to_upper = 100 * (upper - last_px) / (upper - lower)

    if pattern == "Dragonfly Doji" and dist_to_upper < 30 and
macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Gravestone Doji" and dist_to_upper > 70 and
macd_line < signal_line:
        return -1  # Sell Signal
    elif pattern == "Hammer" and macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Inverted Hammer" and macd_line < signal_line:
        return -1  # Sell Signal
    else:
        return 0
```

Source Code 13:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr, adx

from blueshift.finance import commission, slippage

from blueshift.api import (

    symbol,

    order_target_percent,

    set_commission,

    set_slippage,

    schedule_function,

    date_rules,

    time_rules,

)


def initialize(context):

    context.securities = [symbol('MSFT'), symbol('GOOG'),
symbol('AAPL'), symbol('AMZN'), symbol('TSLA')]


    context.params = {

        'indicator_lookback': 200,

        'indicator_freq': '1m',

        'BBands_period': 20,

        'MACD_fast': 12,

        'MACD_slow': 26,

        'MACD_signal': 9,

        'trade_freq': 2,  # Execute every 2 minutes

        'stop_loss_multiplier': 2,  # ATR-based stop loss multiplier

        'take_profit_multiplier': 3,  # ATR-based take profit
multiplier

        'leverage': 2,
```

```
        'volume_threshold': 1.5,  # Multiplier for average volume
    }

    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)
    context.atr_values = dict((security, None) for security in
context.securities)

    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))

    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))

    context.trade = True

def before_trading_start(context, data):
    context.trade = True

def stop_trading(context, data):
    context.trade = False

def run_strategy(context, data):
    if not context.trade:
        return
```

```
    generate_signals(context, data)
    generate_target_position(context, data)
    rebalance(context, data)

def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        atr_value = context.atr_values[security]

        if atr_value and context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            stop_loss_level = context.entry_prices[security] -
(atr_value * context.params['stop_loss_multiplier'])
            take_profit_level = context.entry_prices[security] +
(atr_value * context.params['take_profit_multiplier'])

            # Stop Loss
            if current_price <= stop_loss_level:
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue

            # Take Profit
            if current_price >= take_profit_level:
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue

        order_target_percent(security, target_weight)
        if target_weight != 0:
```

```
            context.entry_prices[security] = data.current(security,
'close')


def generate_target_position(context, data):
    for security in context.securities:
        atr_value = atr(data.history(security, ['high', 'low',
'close'], 20, '1d'), 14)
        context.atr_values[security] = atr_value
        weight = min(max(0.2 / atr_value, 0.02), 0.3)  # Min weight
2%, max 30%
        if context.signals[security] > 0:
            context.target_position[security] = weight *
context.params['leverage']
        elif context.signals[security] < 0:
            context.target_position[security] = -weight *
context.params['leverage']
        else:
            context.target_position[security] = 0


def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close', 'volume'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
    except Exception as e:
        print(f"Data history error: {e}")
        return


    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params)
```

```
def identify_patterns(px):
    """Identify candlestick patterns."""
    open_price = px.open.values[-1]
    close_price = px.close.values[-1]
    high_price = px.high.values[-1]
    low_price = px.low.values[-1]

    if abs(close_price - open_price) < (high_price - low_price) *
0.1:
        # Narrow body for Doji-like candles
        if (high_price - close_price) > 2 * (close_price -
low_price):
            return "Gravestone Doji"
        elif (close_price - low_price) > 2 * (high_price -
close_price):
            return "Dragonfly Doji"
    elif close_price > open_price and (close_price - low_price) > 2
* (high_price - close_price):
        return "Hammer"
    elif close_price < open_price and (high_price - close_price) > 2
* (close_price - low_price):
        return "Inverted Hammer"
    return None

def signal_function(px, params):
    """Generate trading signals based on patterns, volume, and
indicators."""
    pattern = identify_patterns(px)
    upper, mid, lower = bollinger_band(px.close.values,
params['BBands_period'])
    macd_line, signal_line, _ = macd(px.close.values,
params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])
```

```
    adx_value = adx(px.high.values, px.low.values, px.close.values,
14)
    avg_volume = px.volume.values[-
params['indicator_lookback']:].mean()
    last_volume = px.volume.values[-1]


    if upper - lower == 0:
        return 0


    # Check for high volume confirmation
    if last_volume < params['volume_threshold'] * avg_volume:
        return 0


    last_px = px.close.values[-1]
    dist_to_upper = 100 * (upper - last_px) / (upper - lower)


    # Use ADX for trend strength filtering
    if adx_value < 15:
        return 0


    # Candlestick Patterns with volume confirmation
    if pattern == "Dragonfly Doji" and dist_to_upper < 30 and
macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Gravestone Doji" and dist_to_upper > 70 and
macd_line < signal_line:
        return -1  # Sell Signal
    elif pattern == "Hammer" and macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Inverted Hammer" and macd_line < signal_line:
        return -1  # Sell Signal
```

```
    else:
        return 0
```

Source Code 14:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr, adx
from blueshift.finance import commission, slippage
from blueshift.api import (
    symbol,
    order_target_percent,
    set_commission,
    set_slippage,
    schedule_function,
    date_rules,
    time_rules,
)


def initialize(context):
    context.securities = [symbol('MSFT'), symbol('GOOG'),
symbol('AAPL'), symbol('AMZN'), symbol('TSLA')]


    context.params = {
        'indicator_lookback': 200,
        'indicator_freq': '1m',
        'BBands_period': 20,
        'MACD_fast': 12,
        'MACD_slow': 26,
        'MACD_signal': 9,
        'trade_freq': 5,
```

```
        'stop_loss_multiplier': 1.5,  # ATR multiplier for stop loss
        'take_profit_multiplier': 2.5,  # ATR multiplier for take
profit
        'leverage': 2,
        'volume_threshold': 1.5,  # Multiplier for average volume
    }


    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)
    context.atr_values = dict((security, None) for security in
context.securities)


    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))


    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))


    context.trade = True


def before_trading_start(context, data):
    context.trade = True


def stop_trading(context, data):
    context.trade = False
```

```
def run_strategy(context, data):
    if not context.trade:
        return

    generate_signals(context, data)
    update_atr_values(context, data)  # Update ATR values
    # generate_target_position(context, data)
    generate_target_position_static(context, data)
    rebalance(context, data)

def update_atr_values(context, data):
    for security in context.securities:
        try:
            price_data = data.history(security, ['high', 'low',
'close'], 20, '1d')
            context.atr_values[security] = atr(price_data, 14)
        except Exception as e:
            print(f"ATR calculation error for {security}: {e}")

def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        atr_value = context.atr_values[security]

        if atr_value and context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            trailing_stop_loss = max(
                context.entry_prices[security] * (1 - atr_value *
context.params['stop_loss_multiplier']),
                context.entry_prices[security] - (atr_value *
context.params['stop_loss_multiplier'])
```

```
            )
            take_profit_level = context.entry_prices[security] +
(atr_value * context.params['take_profit_multiplier'])

            # Trailing Stop Loss
            if current_price <= trailing_stop_loss:
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue

            # Take Profit
            if current_price >= take_profit_level:
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue

        order_target_percent(security, target_weight)
        if target_weight != 0:
            context.entry_prices[security] = data.current(security,
'close')

def generate_target_position(context, data):
    for security in context.securities:
        atr_value = context.atr_values.get(security, None)
        if atr_value:
            weight = min(max(0.2 / atr_value, 0.02), 0.3)  # Min
weight 2%, max 30%
            if context.signals[security] > 0:
                context.target_position[security] = weight *
context.params['leverage']
            elif context.signals[security] < 0:
```

```
                context.target_position[security] = -weight *
context.params['leverage']
            else:
                context.target_position[security] = 0


def generate_target_position_static(context, data):
    """Static position sizing: Allocate a fixed weight to each
position."""
    static_weight = 0.10  # Example: 10% of portfolio for each trade
    for security in context.securities:
        if context.signals[security] > 0:
            context.target_position[security] = static_weight *
context.params['leverage']
        elif context.signals[security] < 0:
            context.target_position[security] = -static_weight *
context.params['leverage']
        else:
            context.target_position[security] = 0


def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close', 'volume'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
    except Exception as e:
        print(f"Data history error: {e}")
        return


    for security in context.securities:
        px = price_data.xs(security)
```

```
        context.signals[security] = signal_function(px,
context.params)


def identify_patterns(px):
    """Identify candlestick patterns."""
    open_price = px.open.values[-1]
    close_price = px.close.values[-1]
    high_price = px.high.values[-1]
    low_price = px.low.values[-1]


    if abs(close_price - open_price) < (high_price - low_price) *
0.1:
        if (high_price - close_price) > 2 * (close_price -
low_price):
            return "Gravestone Doji"
        elif (close_price - low_price) > 2 * (high_price -
close_price):
            return "Dragonfly Doji"
    elif close_price > open_price and (close_price - low_price) > 2
* (high_price - close_price):
        return "Hammer"
    elif close_price < open_price and (high_price - close_price) > 2
* (close_price - low_price):
        return "Inverted Hammer"
    return None


def signal_function(px, params):
    """Generate trading signals based on patterns, volume, and
indicators."""
    pattern = identify_patterns(px)
    upper, mid, lower = bollinger_band(px.close.values,
params['BBands_period'])
```

```
    macd_line, signal_line, _ = macd(px.close.values,
params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])
    adx_value = adx(px.high.values, px.low.values, px.close.values,
14)
    avg_volume = px.volume.values[-
params['indicator_lookback']:].mean()
    last_volume = px.volume.values[-1]


    if upper - lower == 0:
        return 0


    # Check for high volume confirmation
    if last_volume < params['volume_threshold'] * avg_volume:
        return 0


    # Use ADX for trend strength filtering
    if adx_value < 15:
        return 0


    # Candlestick Patterns with volume confirmation
    last_px = px.close.values[-1]
    dist_to_upper = 100 * (upper - last_px) / (upper - lower)


    if pattern == "Dragonfly Doji" and dist_to_upper < 30 and
macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Gravestone Doji" and dist_to_upper > 70 and
macd_line < signal_line:
        return -1  # Sell Signal
    elif pattern == "Hammer" and macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Inverted Hammer" and macd_line < signal_line:
```

```
        return -1  # Sell Signal
    else:
        return 0
```

Source Code 15:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr, adx
from blueshift.finance import commission, slippage
from blueshift.api import (
    symbol,
    order_target_percent,
    set_commission,
    set_slippage,
    schedule_function,
    date_rules,
    time_rules,
)


def initialize(context):
    context.securities = [symbol('MSFT'), symbol('GOOG'),
symbol('AAPL'), symbol('AMZN'), symbol('TSLA')]


    context.params = {
        'indicator_lookback': 200,
        'indicator_freq': '1m',
        'BBands_period': 20,
        'MACD_fast': 12,
        'MACD_slow': 26,
        'MACD_signal': 9,
```

```
        'trade_freq': 5,
        'stop_loss_multiplier': 1.5,  # ATR multiplier for stop loss
        'take_profit_multiplier': 2.5,  # ATR multiplier for take
profit
        'leverage': 2,
        'volume_threshold': 1.5,  # Multiplier for average volume
    }


    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)
    context.atr_values = dict((security, None) for security in
context.securities)


    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))


    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))


    context.trade = True


def before_trading_start(context, data):
    context.trade = True


def stop_trading(context, data):
    context.trade = False
```

```
def run_strategy(context, data):
    if not context.trade:
        return

    generate_signals(context, data)
    update_atr_values(context, data)  # Update ATR values
    generate_target_position_threshold(context, data)
    rebalance(context, data)

def update_atr_values(context, data):
    for security in context.securities:
        try:
            price_data = data.history(security, ['high', 'low',
'close'], 20, '1d')
            context.atr_values[security] = atr(price_data, 14)
        except Exception as e:
            print(f"ATR calculation error for {security}: {e}")

def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        atr_value = context.atr_values[security]

        if atr_value and context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            trailing_stop_loss = max(
                context.entry_prices[security] * (1 - atr_value *
context.params['stop_loss_multiplier']),
                context.entry_prices[security] - (atr_value *
context.params['stop_loss_multiplier'])
```

```
            )
            take_profit_level = context.entry_prices[security] +
(atr_value * context.params['take_profit_multiplier'])

            # Trailing Stop Loss
            if current_price <= trailing_stop_loss:
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue

            # Take Profit
            if current_price >= take_profit_level:
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue

        order_target_percent(security, target_weight)
        if target_weight != 0:
            context.entry_prices[security] = data.current(security,
'close')

def generate_target_position(context, data):
    for security in context.securities:
        atr_value = context.atr_values.get(security, None)
        if atr_value:
            weight = min(max(0.2 / atr_value, 0.02), 0.3)  # Min
weight 2%, max 30%
            if context.signals[security] > 0:
                context.target_position[security] = weight *
context.params['leverage']
            elif context.signals[security] < 0:
```

```
                context.target_position[security] = -weight *
context.params['leverage']
            else:
                context.target_position[security] = 0


def generate_target_position_threshold(context, data):
    """Simpler threshold-based sizing."""
    for security in context.securities:
        if context.signals[security] > 0:
            # Example: Larger positions for stocks near lower
Bollinger Band
            lower_band_threshold = 0.3
            weight = 0.1 if context.dist_to_lower_band[security] <
lower_band_threshold else 0.05
            context.target_position[security] = weight *
context.params['leverage']
        elif context.signals[security] < 0:
            # Example: Larger short positions for stocks near upper
Bollinger Band
            upper_band_threshold = 0.7
            weight = 0.1 if context.dist_to_upper_band[security] >
upper_band_threshold else 0.05
            context.target_position[security] = -weight *
context.params['leverage']
        else:
            context.target_position[security] = 0


def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close', 'volume'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
```

```
    except Exception as e:
        print(f"Data history error: {e}")
        return

    context.dist_to_upper_band = {}
    context.dist_to_lower_band = {}

    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params)

        # Calculate Bollinger Band distances
        upper, mid, lower = bollinger_band(px.close.values,
context.params['BBands_period'])
        last_px = px.close.values[-1]

        # Store the distance to upper and lower bands in percentages
        if upper - lower != 0:  # Avoid division by zero
            context.dist_to_upper_band[security] = (upper - last_px)
/ (upper - lower)
            context.dist_to_lower_band[security] = (last_px - lower)
/ (upper - lower)

def identify_patterns(px):
    """Identify candlestick patterns."""
    open_price = px.open.values[-1]
    close_price = px.close.values[-1]
    high_price = px.high.values[-1]
    low_price = px.low.values[-1]
```

```
    if abs(close_price - open_price) < (high_price - low_price) *
0.1:
        if (high_price - close_price) > 2 * (close_price -
low_price):
            return "Gravestone Doji"
        elif (close_price - low_price) > 2 * (high_price -
close_price):
            return "Dragonfly Doji"
    elif close_price > open_price and (close_price - low_price) > 2
* (high_price - close_price):
        return "Hammer"
    elif close_price < open_price and (high_price - close_price) > 2
* (close_price - low_price):
        return "Inverted Hammer"
    return None


def signal_function(px, params):
    """Generate trading signals based on patterns, volume, and
indicators."""
    pattern = identify_patterns(px)
    upper, mid, lower = bollinger_band(px.close.values,
params['BBands_period'])
    macd_line, signal_line, _ = macd(px.close.values,
params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])
    adx_value = adx(px.high.values, px.low.values, px.close.values,
14)
    avg_volume = px.volume.values[-
params['indicator_lookback']:].mean()
    last_volume = px.volume.values[-1]


    if upper - lower == 0:
        return 0
```

```
    # Check for high volume confirmation
    if last_volume < params['volume_threshold'] * avg_volume:
        return 0

    # Use ADX for trend strength filtering
    if adx_value < 15:
        return 0

    # Candlestick Patterns with volume confirmation
    last_px = px.close.values[-1]
    dist_to_upper = 100 * (upper - last_px) / (upper - lower)

    if pattern == "Dragonfly Doji" and dist_to_upper < 30 and
macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Gravestone Doji" and dist_to_upper > 70 and
macd_line < signal_line:
        return -1  # Sell Signal
    elif pattern == "Hammer" and macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Inverted Hammer" and macd_line < signal_line:
        return -1  # Sell Signal
    else:
        return 0
```

Source Code 16:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr, adx
from blueshift.finance import commission, slippage
from blueshift.api import (
    symbol,
    order_target_percent,
    set_commission,
    set_slippage,
    schedule_function,
    date_rules,
    time_rules,
)


def initialize(context):
    context.securities = [symbol('MSFT'), symbol('GOOG'),
symbol('AAPL'), symbol('AMZN'), symbol('TSLA')]


    context.params = {
        'indicator_lookback': 200,
        'indicator_freq': '1m',
        'BBands_period': 20,
        'MACD_fast': 12,
        'MACD_slow': 26,
        'MACD_signal': 9,
        'trade_freq': 2,  # Execute every 2 minutes
        'stop_loss_multiplier': 1.5,  # ATR multiplier for stop loss
        'take_profit_multiplier': 2.5,  # ATR multiplier for take
profit
        'leverage': 2,
```

```
        'volume_threshold': 1.5,  # Multiplier for average volume
    }

    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)
    context.atr_values = dict((security, None) for security in
context.securities)

    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))

    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))

    context.trade = True

def before_trading_start(context, data):
    context.trade = True

def stop_trading(context, data):
    context.trade = False

def run_strategy(context, data):
    if not context.trade:
        return
```

```
    generate_signals(context, data)
    update_atr_values(context, data)  # Update ATR values
    generate_target_position(context, data)
    rebalance(context, data)

def update_atr_values(context, data):
    for security in context.securities:
        try:
            price_data = data.history(security, ['high', 'low',
'close'], 20, '1d')
            context.atr_values[security] = atr(price_data, 14)
        except Exception as e:
            print(f"ATR calculation error for {security}: {e}")

def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        atr_value = context.atr_values[security]

        if atr_value and context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            trailing_stop_loss = max(
                context.entry_prices[security] * (1 - atr_value *
context.params['stop_loss_multiplier']),
                context.entry_prices[security] - (atr_value *
context.params['stop_loss_multiplier'])
            )
            take_profit_level = context.entry_prices[security] +
(atr_value * context.params['take_profit_multiplier'])
```

```
            # Trailing Stop Loss
            if current_price <= trailing_stop_loss:
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue

            # Take Profit
            if current_price >= take_profit_level:
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue

        order_target_percent(security, target_weight)
        if target_weight != 0:
            context.entry_prices[security] = data.current(security,
'close')


def generate_target_position(context, data):
    for security in context.securities:
        atr_value = context.atr_values.get(security, None)
        if atr_value:
            weight = min(max(0.2 / atr_value, 0.02), 0.3)  # Min
weight 2%, max 30%
            if context.signals[security] > 0:
                context.target_position[security] = weight *
context.params['leverage']
            elif context.signals[security] < 0:
                context.target_position[security] = -weight *
context.params['leverage']
            else:
                context.target_position[security] = 0
```

```
def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close', 'volume'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
    except Exception as e:
        print(f"Data history error: {e}")
        return

    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params)

def identify_patterns(px):
    """Identify candlestick patterns."""
    open_price = px.open.values[-1]
    close_price = px.close.values[-1]
    high_price = px.high.values[-1]
    low_price = px.low.values[-1]

    if abs(close_price - open_price) < (high_price - low_price) *
0.1:
        if (high_price - close_price) > 2 * (close_price -
low_price):
            return "Gravestone Doji"
        elif (close_price - low_price) > 2 * (high_price -
close_price):
            return "Dragonfly Doji"
```

```
    elif close_price > open_price and (close_price - low_price) > 2
* (high_price - close_price):
        return "Hammer"
    elif close_price < open_price and (high_price - close_price) > 2
* (close_price - low_price):
        return "Inverted Hammer"
    return None


def signal_function(px, params):
    """Generate trading signals based on patterns, volume, and
indicators."""
    pattern = identify_patterns(px)
    upper, mid, lower = bollinger_band(px.close.values,
params['BBands_period'])
    macd_line, signal_line, _ = macd(px.close.values,
params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])
    adx_value = adx(px.high.values, px.low.values, px.close.values,
14)
    avg_volume = px.volume.values[-
params['indicator_lookback']:].mean()
    last_volume = px.volume.values[-1]


    if upper - lower == 0:
        return 0


    # Check for high volume confirmation
    if last_volume < params['volume_threshold'] * avg_volume:
        return 0


    # Use ADX for trend strength filtering
    if adx_value < 15:
        return 0
```

```
    # Candlestick Patterns with volume confirmation
    last_px = px.close.values[-1]
    dist_to_upper = 100 * (upper - last_px) / (upper - lower)


    if pattern == "Dragonfly Doji" and dist_to_upper < 30 and
macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Gravestone Doji" and dist_to_upper > 70 and
macd_line < signal_line:
        return -1  # Sell Signal
    elif pattern == "Hammer" and macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Inverted Hammer" and macd_line < signal_line:
        return -1  # Sell Signal
    else:
        return 0
```

Source Code 17:

```
from blueshift.library.technicals.indicators import bollinger_band,
doji, macd, atr, adx
from blueshift.finance import commission, slippage
from blueshift.api import (
    symbol,
    order_target_percent,
    set_commission,
    set_slippage,
    schedule_function,
    date_rules,
    time_rules,
)


def initialize(context):
    context.securities = [symbol('MSFT'), symbol('GOOG'),
symbol('AAPL'), symbol('AMZN'), symbol('TSLA')]


    context.params = {
        'indicator_lookback': 300,
        'indicator_freq': '1m',
        'BBands_period': 20,
        'MACD_fast': 5,
        'MACD_slow': 35,
        'MACD_signal': 5,
        'trade_freq': 2,  # Execute every 2 minutes
        'stop_loss_multiplier': 1.5,  # ATR multiplier for stop loss
        'take_profit_multiplier': 2.5,  # ATR multiplier for take
profit
        'leverage': 2,
```

```
        'volume_threshold': 1.5,  # Multiplier for average volume
    }

    context.signals = dict((security, 0) for security in
context.securities)
    context.target_position = dict((security, 0) for security in
context.securities)
    context.entry_prices = dict((security, None) for security in
context.securities)
    context.atr_values = dict((security, None) for security in
context.securities)

    set_commission(commission.PerShare(cost=0.0,
min_trade_cost=0.0))
    set_slippage(slippage.FixedSlippage(0.00))

    schedule_function(run_strategy, date_rules.every_day(),
time_rules.every_nth_minute(context.params['trade_freq']))
    schedule_function(stop_trading, date_rules.every_day(),
time_rules.market_close(minutes=30))

    context.trade = True

def before_trading_start(context, data):
    context.trade = True

def stop_trading(context, data):
    context.trade = False

def run_strategy(context, data):
    if not context.trade:
        return
```

```
    generate_signals(context, data)
    update_atr_values(context, data)  # Update ATR values
    generate_target_position(context, data)
    rebalance(context, data)

def update_atr_values(context, data):
    for security in context.securities:
        try:
            price_data = data.history(security, ['high', 'low',
'close'], 20, '1d')
            context.atr_values[security] = atr(price_data, 14)
        except Exception as e:
            print(f"ATR calculation error for {security}: {e}")

def rebalance(context, data):
    for security in context.securities:
        target_weight = context.target_position[security]
        atr_value = context.atr_values[security]

        if atr_value and context.entry_prices[security] is not None:
            current_price = data.current(security, 'close')
            trailing_stop_loss = max(
                context.entry_prices[security] * (1 - atr_value *
context.params['stop_loss_multiplier']),
                context.entry_prices[security] - (atr_value *
context.params['stop_loss_multiplier'])
            )
            take_profit_level = context.entry_prices[security] +
(atr_value * context.params['take_profit_multiplier'])
```

```
            # Trailing Stop Loss
            if current_price <= trailing_stop_loss:
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue

            # Take Profit
            if current_price >= take_profit_level:
                order_target_percent(security, 0)
                context.entry_prices[security] = None
                continue

        order_target_percent(security, target_weight)
        if target_weight != 0:
            context.entry_prices[security] = data.current(security,
'close')


def generate_target_position(context, data):
    for security in context.securities:
        atr_value = context.atr_values.get(security, None)
        if atr_value:
            weight = min(max(0.2 / atr_value, 0.02), 0.3)  # Min
weight 2%, max 30%
            if context.signals[security] > 0:
                context.target_position[security] = weight *
context.params['leverage']
            elif context.signals[security] < 0:
                context.target_position[security] = -weight *
context.params['leverage']
            else:
                context.target_position[security] = 0
```

```
def generate_signals(context, data):
    try:
        price_data = data.history(context.securities, ['open',
'high', 'low', 'close', 'volume'],
                                  context.params['indicator_lookback
'], context.params['indicator_freq'])
    except Exception as e:
        print(f"Data history error: {e}")
        return

    for security in context.securities:
        px = price_data.xs(security)
        context.signals[security] = signal_function(px,
context.params)

def identify_patterns(px):
    """Identify candlestick patterns."""
    open_price = px.open.values[-1]
    close_price = px.close.values[-1]
    high_price = px.high.values[-1]
    low_price = px.low.values[-1]

    if abs(close_price - open_price) < (high_price - low_price) *
0.1:
        if (high_price - close_price) > 2 * (close_price -
low_price):
            return "Gravestone Doji"
        elif (close_price - low_price) > 2 * (high_price -
close_price):
            return "Dragonfly Doji"
```

```
    elif close_price > open_price and (close_price - low_price) > 2
* (high_price - close_price):
        return "Hammer"
    elif close_price < open_price and (high_price - close_price) > 2
* (close_price - low_price):
        return "Inverted Hammer"
    return None


def signal_function(px, params):
    """Generate trading signals based on patterns, volume, and
indicators."""
    pattern = identify_patterns(px)
    upper, mid, lower = bollinger_band(px.close.values,
params['BBands_period'])
    macd_line, signal_line, _ = macd(px.close.values,
params['MACD_fast'], params['MACD_slow'], params['MACD_signal'])
    adx_value = adx(px.high.values, px.low.values, px.close.values,
14)
    avg_volume = px.volume.values[-
params['indicator_lookback']:].mean()
    last_volume = px.volume.values[-1]


    if upper - lower == 0:
        return 0


    # Check for high volume confirmation
    if last_volume < params['volume_threshold'] * avg_volume:
        return 0


    # Use ADX for trend strength filtering
    if adx_value < 15:
        return 0
```

```
    # Candlestick Patterns with volume confirmation
    last_px = px.close.values[-1]
    dist_to_upper = 100 * (upper - last_px) / (upper - lower)


    if pattern == "Dragonfly Doji" and dist_to_upper < 30 and
macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Gravestone Doji" and dist_to_upper > 70 and
macd_line < signal_line:
        return -1  # Sell Signal
    elif pattern == "Hammer" and macd_line > signal_line:
        return 1  # Buy Signal
    elif pattern == "Inverted Hammer" and macd_line < signal_line:
        return -1  # Sell Signal
    else:
        return 0
```