**Building a Trading System**

**Introduction**

This document outlines our algorithmic trading system, covering its architecture, data handling, trading logic, risk management, and performance evaluation. The system automates market data retrieval, applies a rule-based strategy, executes trades, and continuously monitors market conditions while managing risk.

By integrating technical indicators, statistical analysis, and volatility-based adjustments, the model ensures informed decision-making and controlled exposure. A structured performance evaluation framework tracks key metrics like profitability, drawdowns, and risk-adjusted returns, enabling continuous optimization. The goal is to maintain consistency, minimize risk, and maximize returns while adhering to regulatory standards.

**Market Data Retrieval**

The system retrieves market data from Alpaca using the Alpaca API via the alpaca\_trade\_api library, allowing access to both historical and real-time data. Market data is fetched at daily intervals, allowing the computation of essential technical indicators such as Simple Moving Averages (SMA), Relative Strength Index (RSI), and Moving Average Convergence Divergence (MACD), which serve as the foundation for trade decision-making.

To ensure efficiency and reliability, the system dynamically selects actively tradable stocks from Alpaca’s asset list, filtering based on liquidity and market availability. A maximum of 100 stocks are retrieved per session to maintain performance and prevent excessive API calls. The data is stored in a structured format within a dedicated market\_data/directory, with each stock’s historical data saved in both CSV format for easy accessibility and Pickle format for faster retrieval in Python-based computations.

To enhance system robustness, error handling mechanisms are implemented to detect and manage API rate limits, connection failures, and missing data issues. This ensures uninterrupted data flow and prevents incomplete or corrupt datasets from impacting trade analysis.

Code Implementation for Data Retrieval:

The following snippet demonstrates how the system fetches, processes, and stores historical market data using Alpaca’s API:



This code snippet fetches and processes historical stock data, handling potential API failures and ensuring that sufficient data points are available for technical analysis. By structuring market data retrieval efficiently, the system ensures reliable access to accurate pricing information, forming the foundation for informed trading decisions.

**Data Storage Strategy**

To ensure efficient data retrieval and management, the system employs a structured file-based storage strategy, balancing performance, accessibility, and reliability. Market data is stored in two formats:

* CSV format for human readability and external analysis.
* Pickle format for rapid retrieval and optimized performance in Python-based computations.

This dual-format storage enhances flexibility, allowing seamless integration with technical analysis, strategy computations, and machine learning models. The data is systematically organized in a dedicated market\_data/directory, where each stock's historical data is saved as separate files, enabling quick lookup, efficient processing, and smooth integration with trading execution scripts.

Storage Considerations

Timestamp Management

* All timestamps are stored in Coordinated Universal Time (UTC) to maintain consistency across different trading exchanges and time zones. This ensures accurate synchronization when backtesting strategies or aligning with real-time market data.

Data Structure

* Each stock’s historical data is structured in a tabular format, capturing essential trading metrics that form the foundation for market analysis and strategy computation. The data includes:
  + Open Price (open)
  + High Price (high)
  + Low Price (low)
  + Closing Price (close)
  + Trading Volume (volume)
* By structuring the data in this format, the system ensures that technical indicators can be computed efficiently, enabling data-driven decision-making for trade execution.

Data Update Frequency

* Market data is retrieved daily at the close of trading hours, ensuring that the system bases strategy computations on the most recent available information.
* The system is designed to automate data updates, replacing outdated records while retaining historical data for trend analysis and backtesting.

**Trading Strategy Development**

The trading strategy employed in this system is designed to leverage a combination of trend-following and momentum indicators, ensuring systematic and data-driven trade decisions. The strategy integrates technical analysis, risk management rules, and position sizing techniques to maintain controlled exposure while optimizing profitability. By utilizing a rule-based decision-making process, the system aims to capture high-probability trading opportunities while minimizing potential losses.

Strategy Components

Technical Indicators Used: A combination of trend, momentum, and volatility-based indicators is used to assess trade opportunities.

* Simple Moving Averages (SMA-50 & SMA-200):
  + Identifies long-term and short-term trends.
  + The Golden Cross (SMA-50 > SMA-200) signals an uptrend.
  + The Death Cross (SMA-50 < SMA-200) signals a downtrend.
* Relative Strength Index (RSI):
  + Measures market momentum on a scale of 0-100.
  + RSI < 40 suggests an oversold condition (potential buy opportunity).
  + RSI > 60 suggests an overbought condition (potential sell opportunity).
* Moving Average Convergence Divergence (MACD):
  + Measures trend strength and momentum.
  + A positive MACD indicates bullish momentum, while a negative MACD suggests bearish pressure.
* Average True Range (ATR):
  + Used for setting dynamic stop-loss levels based on market volatility.
  + Helps prevent premature exits due to short-term price fluctuations.

Trade Signal Generation: The system generates buy, sell, or hold signals based on the interaction of the selected indicators. This approach helps filter out false signals and ensures that only high-probability trades are executed.

* Buy Signal:
  + SMA(50) > SMA(200) (short-term trend stronger than long-term).
  + RSI < 40 (indicating an oversold condition and potential price recovery).
* Sell Signal:
  + SMA(50) < SMA(200) (short-term trend weaker than long-term).
  + RSI > 60 (indicating an overbought condition and potential pullback).
* Hold Condition:
  + If neither buy nor sell conditions are met, the system remains on hold to avoid unnecessary trades.

Risk Management

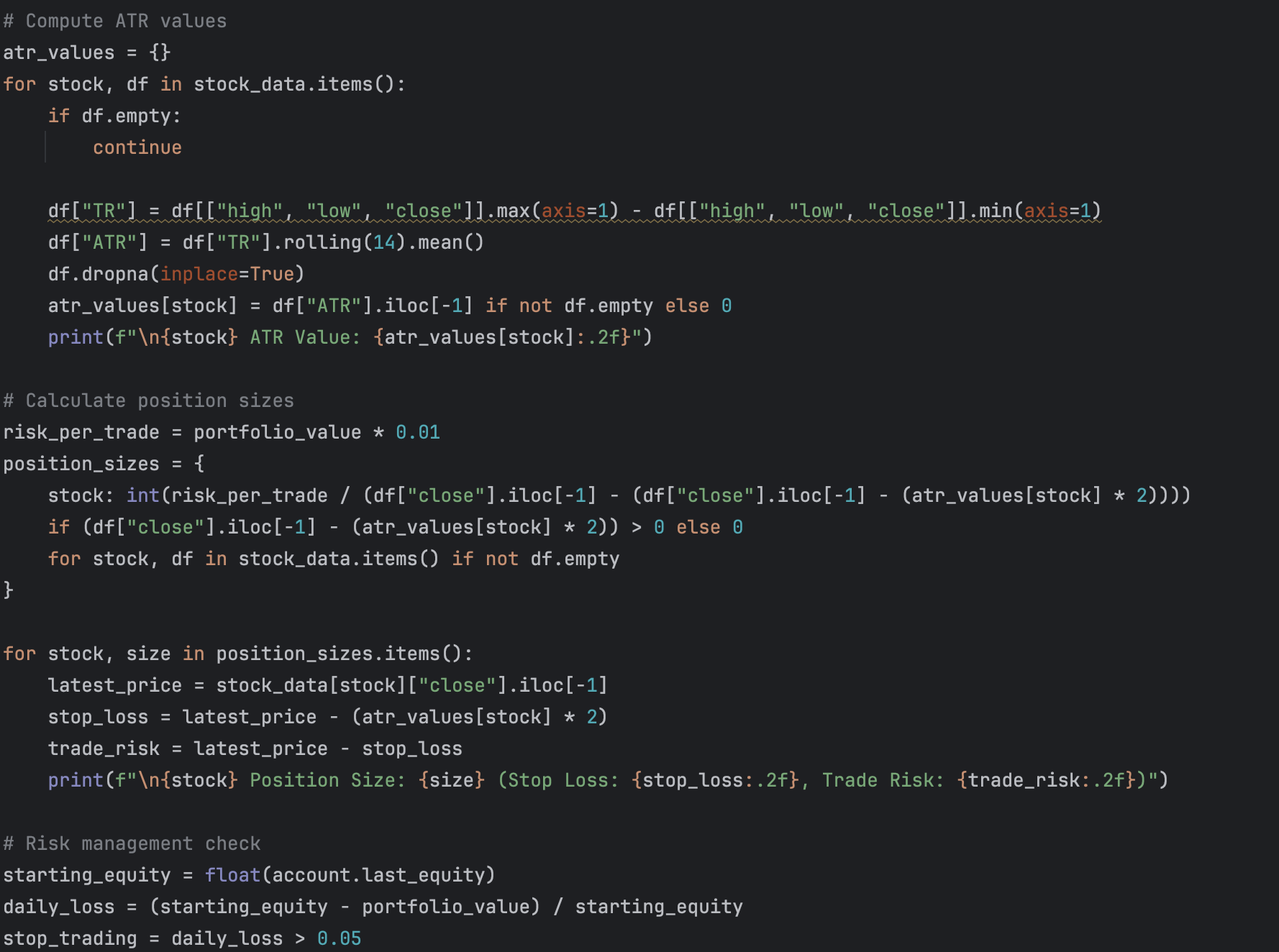
Effective risk control measures are implemented to protect capital and maintain portfolio stability.

* Position Sizing:
  + Each trade risks a maximum of 1% of the total portfolio value.
  + Position size is dynamically adjusted based on stop-loss distance and ATR to maintain a consistent risk-per-trade approach.
* Stop-Loss Strategy:
  + Stop-loss levels are set at twice the ATR value, adjusting to market volatility to prevent premature exits.
  + This allows room for price fluctuations while ensuring that losses are limited.
* Daily Loss Limit:
  + Trading is automatically halted if losses exceed 5% of portfolio value in a single day.
  + This prevents emotional or revenge trading and ensures capital preservation.

**Code Explanation**



The trade signal generation process follows a structured approach to analyze market data, compute technical indicators, and make informed buy, sell, or hold decisions. The script first initializes an empty dictionary, trade\_signals, which will store the trade decision for each stock. It then iterates through the stock\_data dictionary, which contains historical price data for multiple stocks. If a stock has no available data, it is skipped to avoid processing errors. For each stock, the script computes key technical indicators that help identify potential trading opportunities. After computing these indicators, the script drops any NaN values introduced during the calculations to ensure that only complete and reliable data is used. If dropping missing values results in an empty dataset, the stock is skipped. The latest row of data is then extracted, representing the most recent price action.

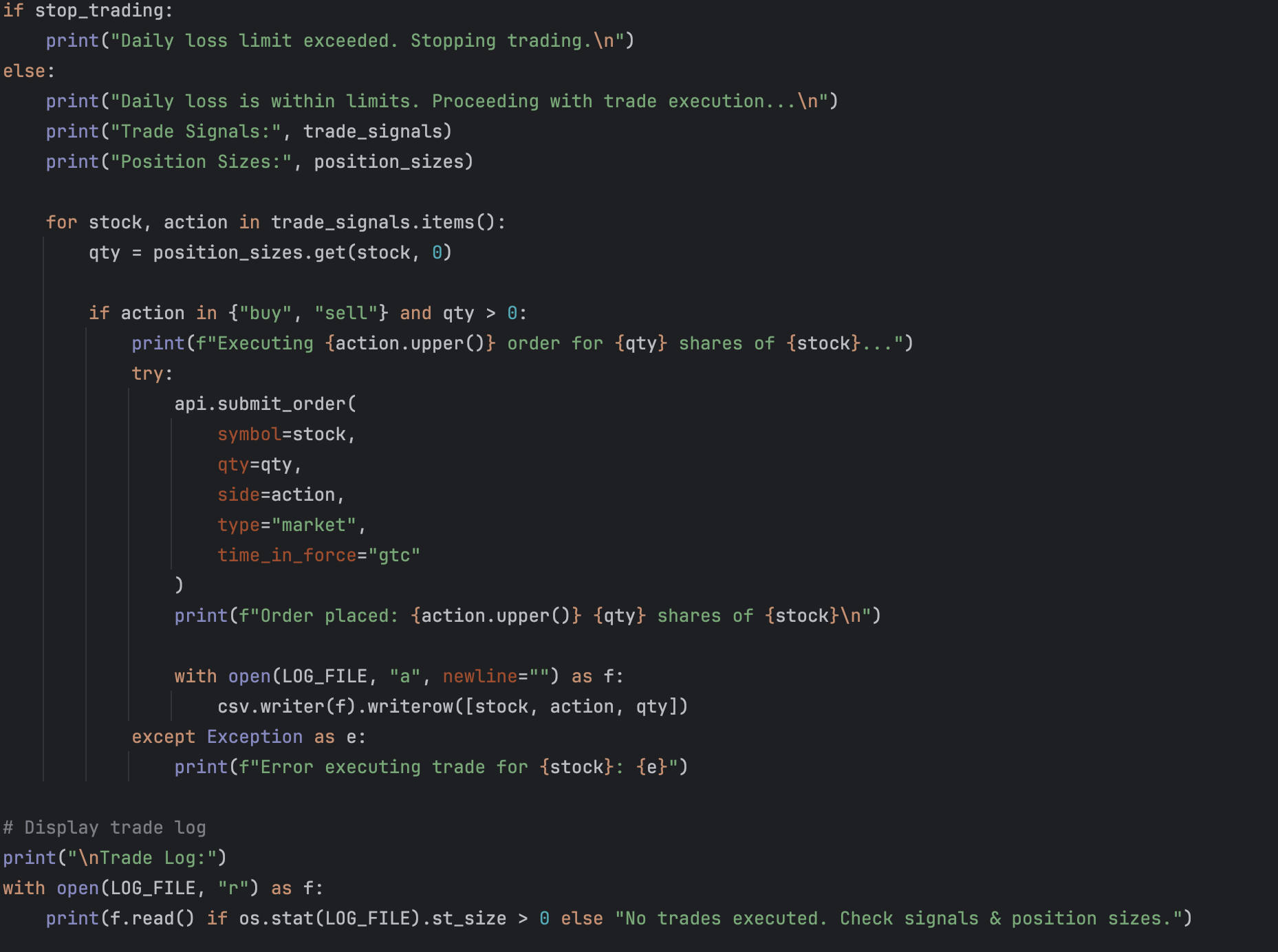


The risk management component ensures that trades are executed with controlled exposure, utilizing ATR-based stop-loss levels, dynamic position sizing, and daily loss limits to prevent excessive drawdowns. The script first initializes the Alpaca API and retrieves the total portfolio value, which is essential for calculating risk per trade. By using 1% of the total portfolio as the maximum risk for each trade, the system maintains consistent and controlled risk exposure across different stocks.

To determine market volatility, the script calculates the Average True Range (ATR) for each stock. ATR is computed using the largest value among High-Low, High-Previous Close, and Low-Previous Close, followed by a 14-period rolling average. This measure allows the system to adjust stop-loss levels dynamically, ensuring that highly volatile stocks have wider stop-loss ranges, while less volatile stocks have tighter stops.

Once ATR values are computed, the script determines position sizes by dividing the 1% portfolio risk allocation by the distance between the latest close price and the ATR-based stop-loss level. This ensures that trades in high-volatility stocks take smaller positions, while trades in low-volatility stocks allow for larger allocations, maintaining a consistent risk profile. After calculating position sizes, the script enforces stop-loss levels by setting a stop at twice the ATR below the latest closing price. This ensures that losses are kept within a manageable range, allowing for normal price fluctuations while limiting downside risk.

Finally, the script implements a daily loss limit, where trading stops if total losses exceed 5% of the portfolio value in a single day. This protective measure prevents significant capital erosion, ensuring that no single day drastically impacts the overall account balance. By integrating ATR-based stop-loss placement, dynamic position sizing, and daily loss limits, this risk management framework creates a structured, disciplined, and sustainable trading system.



The script checks the daily loss limit by referencing the stop\_trading flag from the risk management module. If the daily loss exceeds 5% of the portfolio, trading is halted, and no orders are executed to prevent further losses. If the daily loss is within the acceptable range, the script proceeds with trade execution. For each stock in the trade\_signals dictionary, the script retrieves the corresponding trade action (buy/sell/hold) and position size from the position\_sizes dictionary. If a stock has a valid buy or sell signal and a nonzero position size, a market order is placed using Alpaca’s API.

* Order Execution Process:
  + Retrieves trade action (buy, sell, hold).
  + Checks if the position size is greater than zero.
  + If the signal is buy or sell, submits a market order using Alpaca’s API.
  + Logs the trade in trade\_log.csv for future reference.

If an order is successfully placed, a confirmation message is printed, and the transaction is recorded in the trade log file. In case of an error (e.g., insufficient funds, invalid stock symbol, API failure), an error message is displayed without disrupting the execution of other trades. At the end of the script, the trade log is displayed, showing all executed trades. If no trades were placed, the system notifies the user to check trade signals and position sizes, ensuring transparency in execution.

**Testing and Optimization:** Discuss how you tested your trading strategy, including backtesting and optimization steps. Explain any adjustments made based on testing results.

**Automation and Scheduling**

The trading system is fully automated using scheduled scripts that retrieve market data, generate trade signals, execute orders, and log transactions at predefined intervals. By leveraging automation, the system ensures consistent data updates, timely trade execution, and robust error handling, eliminating the need for manual intervention.

**Paper Trading and Monitoring**

Extensive testing was conducted using Alpaca’s paper trading environment, which simulates real market conditions without financial risk. This step was crucial in:

* Validating trade execution logic in real-time market conditions.
* Assessing the accuracy of trade signals and risk management measures.
* Analyzing execution performance by comparing expected vs. actual fill prices.
* Identifying and fixing potential issues before transitioning to real capital deployment.

Throughout the paper trading phase, the system's logs were monitored to ensure that all components—from data retrieval to trade execution—performed as expected. Adjustments were made to refine stop-loss placements, adjust trade frequency, and optimize signal accuracy based on observed results.

**Results and Lessons Learned**

The testing and optimization process provided key insights into the system's strengths and areas for improvement:

Strengths:

* Successfully automated market data retrieval, trade execution, and risk management.
* Implemented dynamic position sizing based on market volatility.
* Enforced risk control measures, preventing excessive drawdowns.
* Maintained an efficient trade logging and monitoring framework for debugging and evaluation.

Challenges Encountered:

* API rate limits required optimization, leading to the implementation of efficient API call batching.
* False trade signals were reduced by refining indicator thresholds and trade confirmation rules.
* Execution slippage was observed, prompting adjustments in trade order types and execution timing.

Future Improvements:

* Integrate machine learning models to dynamically adjust trade signals.
* Expand asset coverage to include multiple asset classes (e.g., crypto, ETFs).
* Implement real-time monitoring dashboards for enhanced strategy visualization.

**Compliance and Legal Considerations**

The algorithm adheres to regulatory standards, avoiding market manipulation techniques and ensuring ethical execution of trades. Trades are executed in compliance with financial industry best practices.

**Conclusion**

This algorithmic trading system effectively automates the entire trading pipeline, from market data retrieval and trade signal generation to order execution and risk management. By implementing scheduled automation, error handling, and logging mechanisms, the system ensures efficient and reliable trading operations. The use of paper trading allowed for extensive validation before real deployment, helping identify and optimize key areas of improvement.

Moving forward, future enhancements will focus on integrating adaptive machine learning models, expanding the range of tradable assets, and refining execution strategies to improve overall performance.