# Python Trading Setup, Scripting and Automation Proposal - Detailed Breakdown

08-23-2023

Contents

[Python Trading Setup, Scripting and Automation Proposal - Detailed Breakdown 1](#_Toc143725603)

[Project steps 3](#_Toc143725604)

[Flowchart 6](#_Toc143725605)

[Backtesting and Optimization 7](#_Toc143725606)

[- Backtesting 7](#_Toc143725607)

[- Optimization 8](#_Toc143725608)

[Integration with broker's API for live trading 10](#_Toc143725609)

[Database structure 13](#_Toc143725610)

[API Calls 14](#_Toc143725611)

[Input/Output Variable 15](#_Toc143725612)

[Frontend 17](#_Toc143725613)

## Project steps

1. Analyze project requirements and objectives:

* Trading strategy: Determine the specific algorithm/strategy to implement (e.g., trend-following, mean-reverting, breakout, etc.).
* Assets to trade: Forex, stocks, cryptocurrencies, commodities, etc.
* Timeframe: Intraday, daily, weekly, etc.
* Broker: Identify the broker you want to use, such as Oanda, Forex.com, etc.

1. Establish project timeline and steps:

* Step 1: Research and planning
* Step 2: Data retrieval and preprocessing
* Step 3: Implement trading algorithm
* Step 4: Backtesting and optimization
* Step 5: Integration with the broker's API for live trading
* Step 6: GUI development
* Step 7: Testing and deployment

1. List of required technologies and libraries:

* Python programming language
* Third-party Python libraries: pandas, numpy, TA-Lib, ccxt, backtrader, requests, matplotlib, etc.
* Broker's API: Oanda API, Forex.com API, or similar
* GUI framework: Matplotlib

1. Data retrieval and preprocessing:

* Acquire historical Forex data from a reliable data provider (e.g., Forex.com, Oanda, or other sources).
* Use Python code to clean, preprocess, and store this data into a suitable format (e.g., pandas DataFrame or CSV files).
* Maintain a data repository for efficient data retrieval and management.

1. Trading algorithm implementation:

* Based on the selected trading strategy, implement the core logic in Python.
* Use the TA-Lib library (or similar) to compute technical indicators.
* Define trading rules (buy/sell conditions), money management (position sizing), and risk management (stop-loss, take-profit) as part of the algorithm.

1. Backtesting and optimization:

* Conduct backtesting using the historical data to evaluate the performance of the trading algorithm.
* Develop a custom Python script or use an existing backtesting library like Backtrader.
* Optimize the algorithm's parameters as needed to improve the risk-adjusted returns.

1. Integration with broker's API for live trading:

* Connect the trading algorithm with the chosen broker's API (e.g., Oanda API, Forex.com API).
* Implement trade execution and order management logic to manage the positions in real-time.
* Handle API errors and exceptions to ensure smooth functioning during live trading.

1. GUI development:

* Create a user-friendly graphical interface for the trading system using a Python GUI framework (e.g., MatplotLib).
* MatplotLib is a powerful python library that can be used to create GUI with an easy-to-use and customizable interface.
* Include features such as login/authentication, settings customization, algorithm control (start, pause, resume, stop), real-time performance monitoring, and trading history.

1. Testing and deployment:

* Perform thorough testing of the complete system in a controlled environment to ensure all modules work as intended.
* Fix any bugs or issues identified during the testing phase.
* Deploy the final version of the system and monitor its performance during live trading.

1. Post-deployment support:

* Monitor the system and address any issues or bugs that come up during live trading to ensure consistent performance.
* Periodically update the system to accommodate any changes in market conditions, trading strategy or APIs.

Following this detailed execution plan, you will be able to create and deploy a successful algorithmic trading system in Python.

## Flowchart

Initial setup

1. Data Retrieval

Historical\_data.csv

Best\_profit = -inf, I = 0

2. Strategy creation

3. Backtesting

Change short\_period, long\_period

I++

Profit = max(profit, best\_profit)

4. Optimization

Is I >= n ?

no

yes

5. Trade management and automation

6. Historical data analysis

Historical\_trades.csv

7. GUI development

## Backtesting and Optimization

### - Backtesting

Input

1. Strategy: Choose a strategy from available ones.
2. Timeframe: Specify the timeframe of the data (1 minute, 5 minutes, 15 minutes, 1 hour, etc.).
3. Currency Pair Groups: Select pre-defined groups of currency pairs for backtesting. (e.g., JPY\_Group)
4. SL: The chosen Stop Loss level for your strategy.
5. TP: The chosen Take Profit level for your strategy.
6. EMA (Big, Medium, Small): The chosen Exponential Moving Average periods for your strategy.
7. Data Series (Price based on, Type, Value): Optional input to activate trailing stop loss.
8. Time frame (Start date, End date, Trading hours): The specific period you want to backtest your strategy on, including start date, end date, and trading hours.
9. Indicator Inputs: If the chosen strategy uses indicator inputs, you can specify them here.
10. Order Settings Template: Optional input to use a pre-defined order settings template (e.g., lot\_1\_20\_10)

Output

1. Total Return: The cumulative profit or loss generated by the strategy during the backtesting period.
2. Win Rate: The percentage of trades that resulted in a profit.
3. Average Profit per Trade: The average profit per trade, taking into account both winning and losing trades.
4. Maximum Drawdown: The largest decrease in account value from the highest point during the backtesting period.
5. Risk/Reward Ratio: The ratio of average loss per losing trade to average gain per winning trade.
6. Sharp Ratio: A measure of risk-adjusted return, comparing the strategy's excess return to its volatility.
7. Trade Count: The total number of trades executed during the backtesting period.
8. Exposure Time: The percentage of time the strategy was in a trade during the backtesting period.

Flowchart

1. Import necessary libraries.

2. Load historical data and set the date as the index.

3. Define required parameters.

4. Calculate the required EMA values and trading signals.

5. Backtest the strategy, calculate the PnL, and store the results.

6. Calculate the output metrics.

7. Print the results.

### - Optimization

Input

1. Strategy: Choose a strategy from available ones.
2. Currency Pair Groups: Select pre-defined groups of currency pairs for optimization. (e.g., JPY\_Group)
3. SL (Min, Max, Increment): Stop Loss levels to test for optimization.
4. TP (Min, Max, Increment): Take Profit levels to test for optimization.
5. Trailing Stop Loss (Optional): Include trailing stop loss in optimization.
6. EMA (Big, Medium, Small): Exponential Moving Average periods to test for optimization.
7. Data Series (Price based on, Type, Value): Specify the timeframe of the data (1 minute, 5 minutes, 15 minutes, 1 hour, etc.).
8. Time frame (Start date, End date, Trading hours): The specific period you want to optimize your strategy on, including start date, end date, and trading hours.
9. Indicator Inputs Optimization: If the strategy contains indicator inputs, then optimization will allow optimizing them (Min, Max, Increment for each input parameter).
10. Optimize (Keep best # results, Optimize on): The optimization method to use (e.g. highest return, lowest drawdown) and the number of best results you want to keep.
11. Order Settings Template: Optional input to use a pre-defined order settings template (e.g., lot\_1\_20\_10)

Output

1. Optimal Inputs: The values of the input parameters (SL, TP, EMA, etc.) that resulted in the best performance according to the chosen optimization metric.
2. Indicator Optimized Parameters: If the strategy contains indicator inputs, a summary of optimized parameters will be included.
3. Optimization Metric: The chosen metric for assessing the performance of different parameter sets, such as highest return, lowest drawdown, or highest risk-adjusted return.
4. Best Parameter Combinations: A list of the best combinations for each parameter tested in the optimization.
5. Performance Summary: A summary table or chart showing the performance of each parameter set tested during the optimization process, including the optimization metric and other performance measures as applicable.
6. Ranking: A ranking of the various parameter sets, sorted by their performance according to the optimization metric.
7. Sensitivity Analysis: A graphical representation of the strategy's performance across a range of parameter values, such as a heatmap or surface plot, which can help to identify robust parameter regions and avoid over-optimizing on a specific parameter set.

Flowchart

1. import the required libraries

2. define input parameters.

3. load and prepare data

4. define trading strategy, calculate the performance metrics, and optimize the parameters

6. Calculate the output metrics.

7. Print the results.

## Integration with broker's API for live trading

Input

1. API Key: An API key is an access token provided by Oanda to authorize your trading application to connect to their trading servers. You'll need to create an account with Oanda and generate an API key.
2. Account ID: Your Oanda account ID is a unique identifier assigned to your trading account. You can find this on the "Account" page after logging in to your Oanda account.
3. Currency Pair Groups: Select pre-defined group of currency pairs to trade, such as JPY\_Group.
4. Timeframe: Set the trading timeframe or period, which determines the granularity of the data used in your trading algorithm. Common timeframes include 1-minute, 5-minute, 15-minute, 1-hour, 4-hour, or daily charts.
5. Risk Management Settings: Configure stop loss, take profit, and trailing stop values (if applicable) to ensure you're managing risk proactively. These parameters help to protect your trading capital.
6. Trade Execution: Specify how to open and close trades, including the use of limit and stop orders and trade execution logic. This includes the entry and exit rules for your trading strategy.
7. Position Sizing: Set the position size for each trade, either as a fixed number of units or as a percentage of your total account balance.
8. Leverage: Define the leverage you want to use in your trading strategy. Leverage increases both the potential profit and risk in a trade.
9. Trade Frequency: Set how often your trading algorithm should be looking for trades. This could range from multiple times per minute to once per day or week, depending on your strategy.
10. Backtesting and Optimization: Before using the Oanda API to execute live trades, it's crucial to backtest and optimize your trading strategy using historical data. This process helps to identify the best parameter settings for your algorithm and minimize the risk of overfitting.
11. Monitoring and Reporting: Configure how you wish to monitor and receive reports about your trading performance, such as email alerts, logging actions, or live performance charts.
12. Error Handling: Consider how your algorithm should handle potential errors like connection issues, invalid order requests, or unexpected data inputs. Having appropriate error handling in place is crucial for successful trading using the Oanda API.
13. Order Settings Template: Optional input to use a pre-defined order settings template (e.g., lot\_1\_20\_10)

Output

1. Account Information: This includes account ID, account type (practice or live), balance, margin available, margin used, open trade count, open position count, currency, and leverage.
2. Instrument Information: This includes instrument name, instrument type (currency pairs, commodities, indices, etc.), daily open price, previous close price, bid/ask prices, daily high/low prices, spread, interest rates, and trading session information.
3. Order Information: This includes order ID, instrument, order type (buy/sell, stop/limit), order price, order size, stop loss, take profit, order creation time, order expiration time, order status (pending, filled, canceled, etc.), and order fill price.
4. Trade Information: This includes trade ID, instrument, trade size, open price, current price, unrealized profit/loss, realized profit/loss, trade open time, trade close time (if closed), and attached stop loss and take profit orders.
5. Position Information: This includes instrument, total units, unrealized profit/loss, average open price, long/short units, long/short average open prices, and position status (open, closed).
6. Performance Metrics: These include total profit/loss, winning trade count, losing trade count, win rate percentage, average win amount, average loss amount, profit factor, maximum drawdown, and return on investment (ROI).
7. Event Handling Data: This includes handling data for events like account state change, trade state change, order state change, and instrument price update.
8. Error Handling Data: This includes handling data for error events like invalid API requests, server errors, and account-related errors.
9. API Request Limits and Quotas: Information on the number of API requests made, the current request rate, and the remaining requests available per user.
10. Historical Price Data: This includes historical price data for various timeframes, such as daily, hourly, and minute data, including Open, High, Low, and Close (OHLC) values, candlestick data, and price-related indicators.
11. Trading Strategy Implementation: Details of any algorithmic trading strategies running through the API, including entry and exit rules, position sizing, risk management settings, and performance evaluation.

Flowchart

1. Import necessary libraries.

2. Create an instance of the OANDA API.

3. Define required parameters.

4. Setup the stream

5. Start the stream

6. Live trading implementation

7. Error handling and best practices

8. Test before deploying

## Database structure

1. Users Table:
   1. user\_id (Primary Key, Integer): Unique Identifier for each user.
   2. username (Varchar): Username of the user.
   3. password (Varchar): Encrypted password of the user.
   4. email (Varchar): Email address of the user.
   5. created\_at (DateTime): Account creation timestamp.
2. Strategies Table:
   1. strategy\_id (Primary Key, Integer): Unique Identifier for each strategy.
   2. user\_id (Foreign Key referencing users.user\_id, Integer): The user who created the strategy.
   3. strategy\_name (Varchar): Name of the trading strategy.
   4. created\_at (DateTime): Strategy creation timestamp.
3. Strategy\_Configuration Table:
   1. config\_id (Primary Key, Integer): Unique Identifier for each strategy configuration.
   2. strategy\_id (Foreign Key referencing strategies.strategy\_id, Integer): The strategy that the configuration belongs to.
   3. config\_name (Varchar): Name of the configuration.
   4. config\_value (Varchar): The value(s) associated with the configuration.
4. Trading\_Pairs Table:
   1. pair\_id (Primary Key, Integer): Unique Identifier for each trading pair.
   2. base\_currency (Varchar): Base currency of the trading pair.
   3. quote\_currency (Varchar): Quote currency of the trading pair.
5. Historical\_Data Table:
   1. data\_id (Primary Key, Integer): Unique Identifier for each historical data record.
   2. pair\_id (Foreign Key referencing trading\_pairs.pair\_id, Integer): The trading pair that the historical data belongs to.
   3. timestamp (DateTime): The timestamp of the historical data IN US CST timezone. Format: MM-DD-YYYY
   4. open (Decimal): Opening price of the trading pair.
   5. high (Decimal): Highest price of the trading pair.
   6. low (Decimal): Lowest price of the trading pair.
   7. close (Decimal): Closing price of the trading pair
   8. volume (Decimal): The trading volume for the selected timeframe.
6. Trades Table:
   1. trade\_id (Primary Key, Integer): Unique Identifier for each trade.
   2. user\_id (Foreign Key referencing users.user\_id, Integer): The user who executed the trade.
   3. strategy\_id (Foreign Key referencing strategies.strategy\_id, Integer): The strategy applied for the trade.
   4. pair\_id (Foreign Key referencing trading\_pairs.pair\_id, Integer): The trading pair involved in the trade.
   5. entry\_price (Decimal): Entry price of the trade.
   6. exit\_price (Decimal, Nullable): Exit price of the trade (if applicable).
   7. stop\_loss (Decimal): Stop loss price of the trade.
   8. trailing\_stop\_loss (Decimal): Trailing stop loss for the trade.
   9. take\_profit (Decimal): Take profit price of the trade.
   10. position\_size (Decimal): The size of the trade position.
   11. position\_type (Varchar): The position type (Long/Short).
   12. entry\_datetime (DateTime): Timestamp of the trade entry.
   13. exit\_datetime (DateTime, Nullable): Timestamp of the trade exit (if applicable).
   14. trade\_status (Varchar): The status of the trade (Open/Closed).
   15. commission (Decimal): The commission or brokerage fees paid during the transaction.

## API Calls

1. Get account information:

Endpoint: https://api-fxpractice.oanda.com/v3/accounts/your\_account\_id Method: GET

1. Get a list of available instruments (trading pairs):

Endpoint: https://api-fxpractice.oanda.com/v3/accounts/your\_account\_id/instruments Method: GET

1. Get instrument history (candles):

Endpoint: https://api-fxpractice.oanda.com/v3/instruments/EUR\_USD/candles Method: GET Parameters: {"count": "100", "granularity": "H1", "price": "M"}

1. Get the current exchange rate (pricing) for a list of instruments:

Endpoint: https://api-fxpractice.oanda.com/v3/accounts/your\_account\_id/pricing Method: GET Parameters: {"instruments": "EUR\_USD,USD\_JPY"}

1. Place an order:

Endpoint: https://api-fxpractice.oanda.com/v3/accounts/your\_account\_id/orders Method: POST Body: {"order": {"units": "100", "instrument": "EUR\_USD", "timeInForce": "FOK", "type": "MARKET", "positionFill": "DEFAULT"}}

1. Get a list of all open trades:

Endpoint: https://api-fxpractice.oanda.com/v3/accounts/your\_account\_id/openTrades Method: GET

1. Close a trade:

Endpoint: https://api-fxpractice.oanda.com/v3/accounts/your\_account\_id/trades/trade\_id/close Method: PUT Replace trade\_id with the actual trade you want to close.

For each API call, you should set the following headers:

{

"Authorization": "Bearer your\_api\_key",

"Content-Type": "application/json"

}

## Input/Output Variable

1. Data retrieval

Input: instrument, params

Output: historical\_data.csv (filename format: "instrument\_timeframe\_startdate\_enddate.csv", e.g., "USD\_JPY\_M15\_01-01-2020\_05-01-2020.csv")

1. Strategy creation

Input: strategy\_class, strategy\_parameters (should be flexible to support any strategy, not limited to moving average)

Output: strategy\_instance

1. Backtesting

Input: strategy, data

Output: profit

1. Optimization

Input: strategy\_class, strategy\_parameters, data

Output: best\_profit, best\_strategy\_parameters

1. Trade management and automation = live trading

Input: strategy\_instance, stop\_loss\_percentage, take\_profit\_percentage

Output: Final portfolio value

1. Historical data analysis

Input: data

Output: historical\_trades.csv

1. GUI development

* Login
* Register
* Settings interface
* Visual data presentation (based on custom requirements)

## Frontend

1. Settings interface

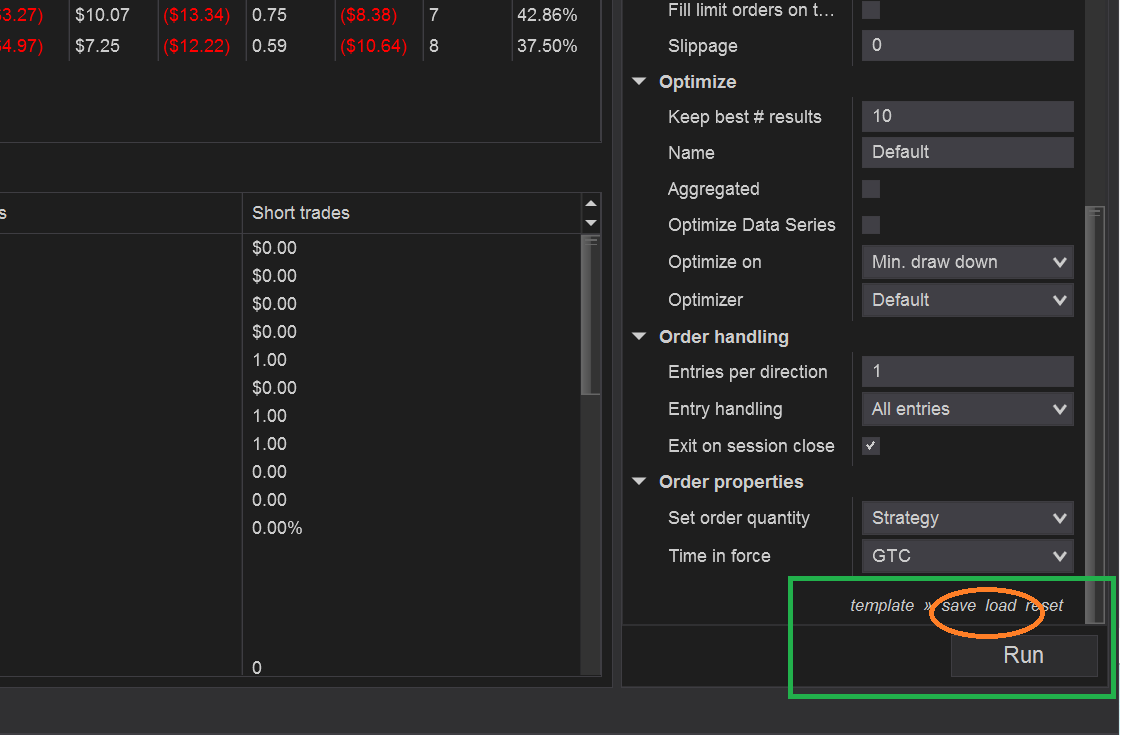
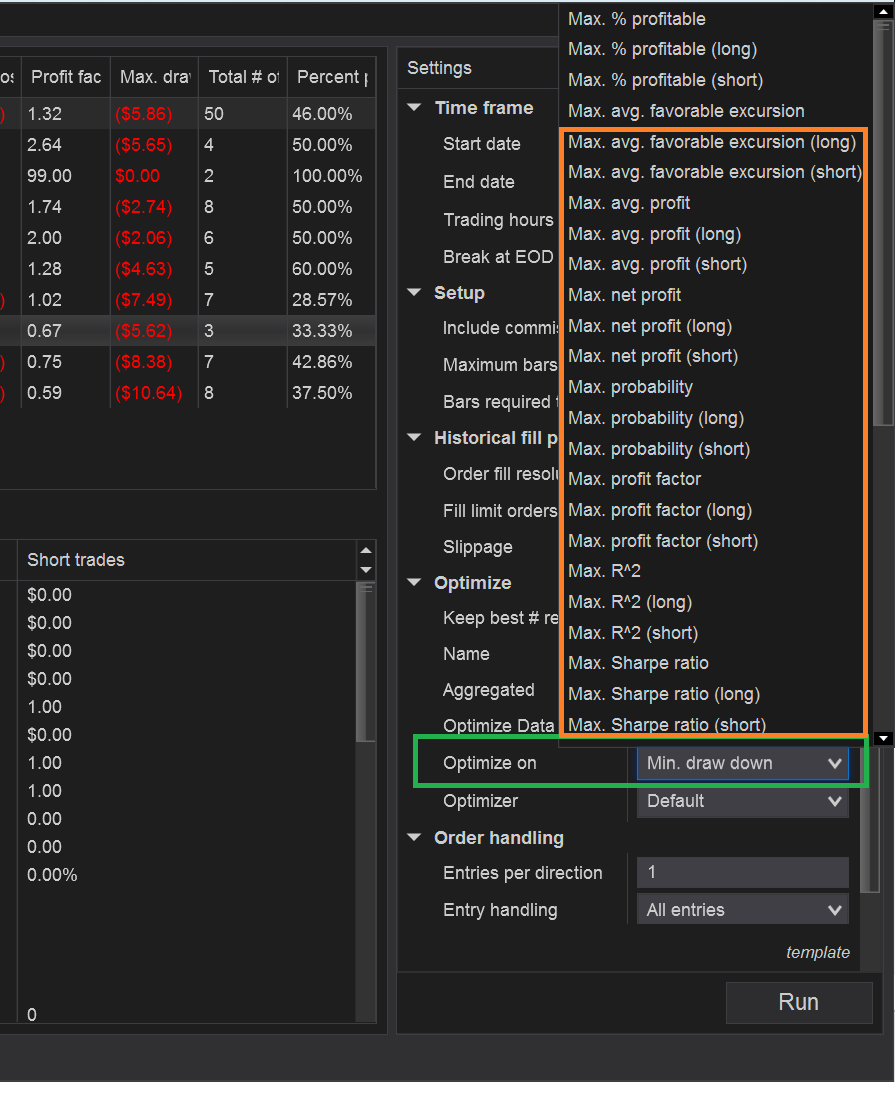
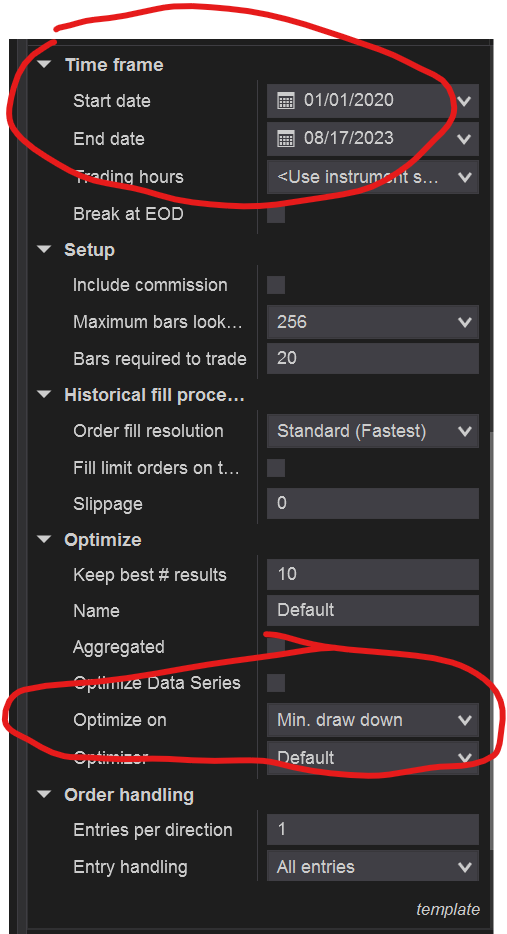
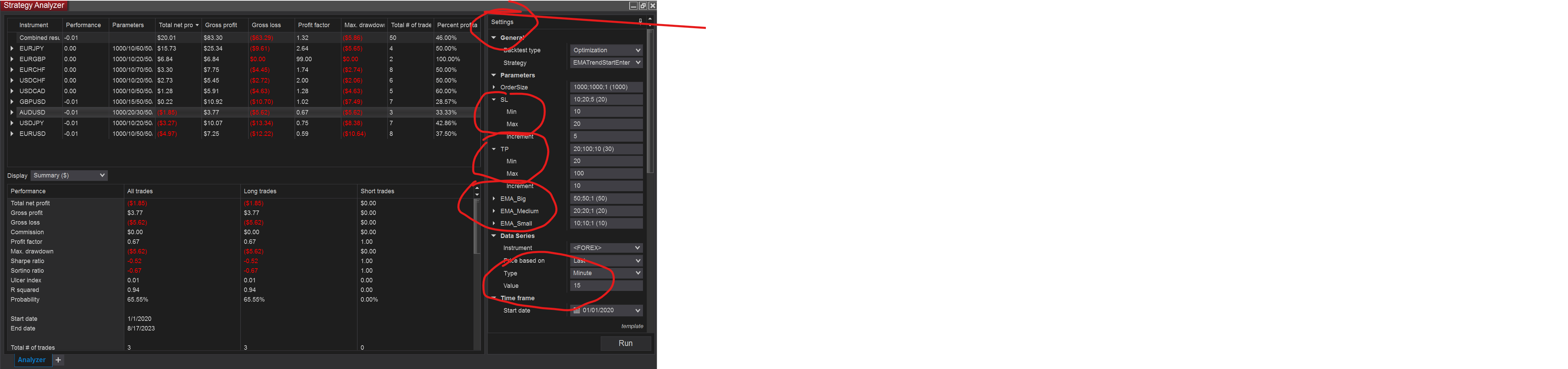


Figure1. Settings interface

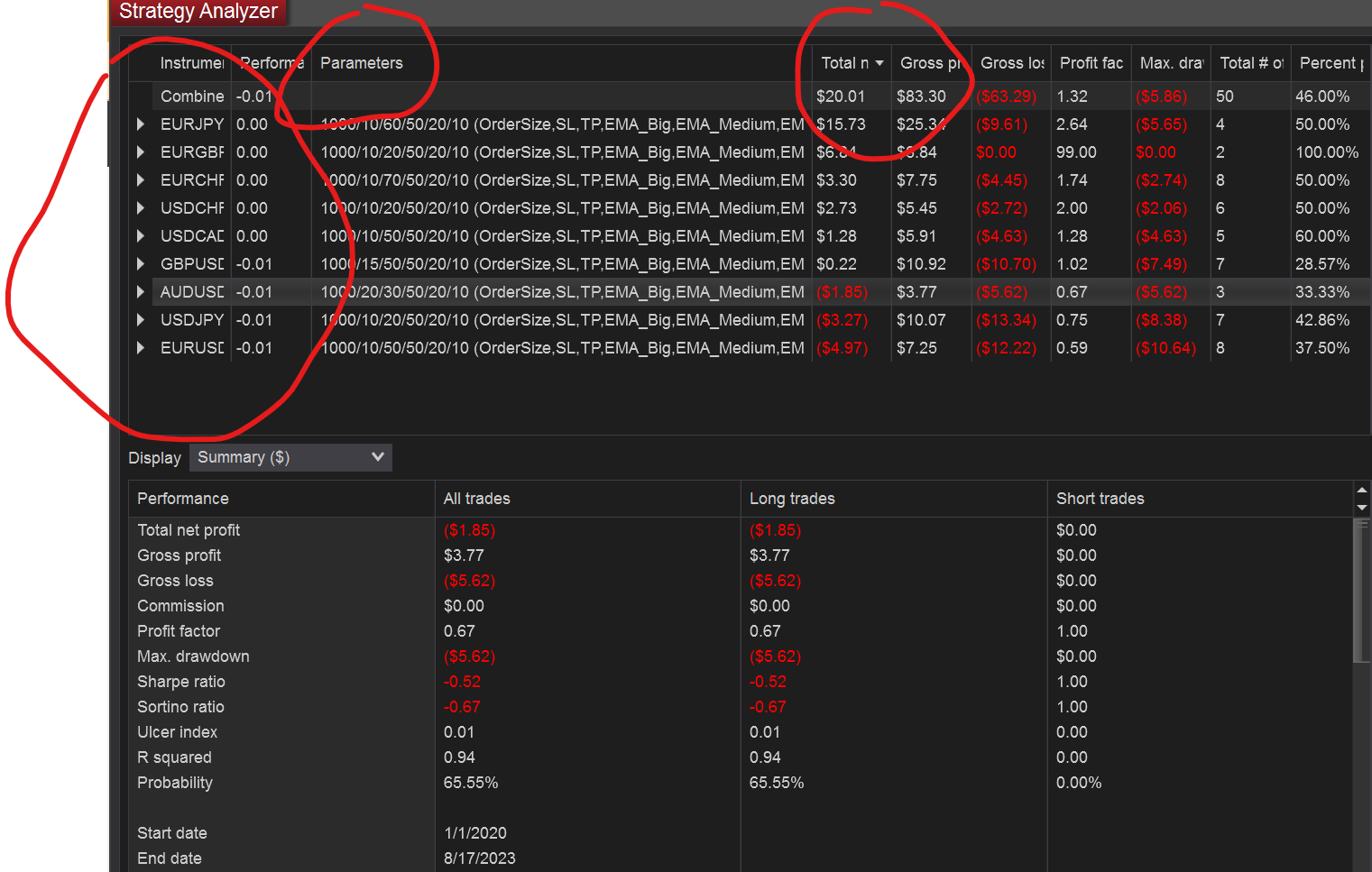
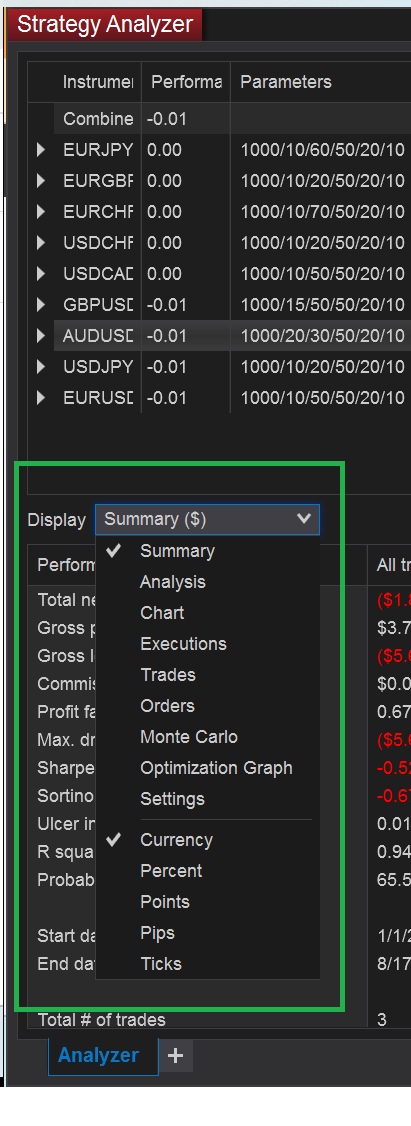
2. Visual data presentation (based on custom requirements) 

Figure2. Visual data presentation (based on custom requirements)