AUTON for Quantitative Trading

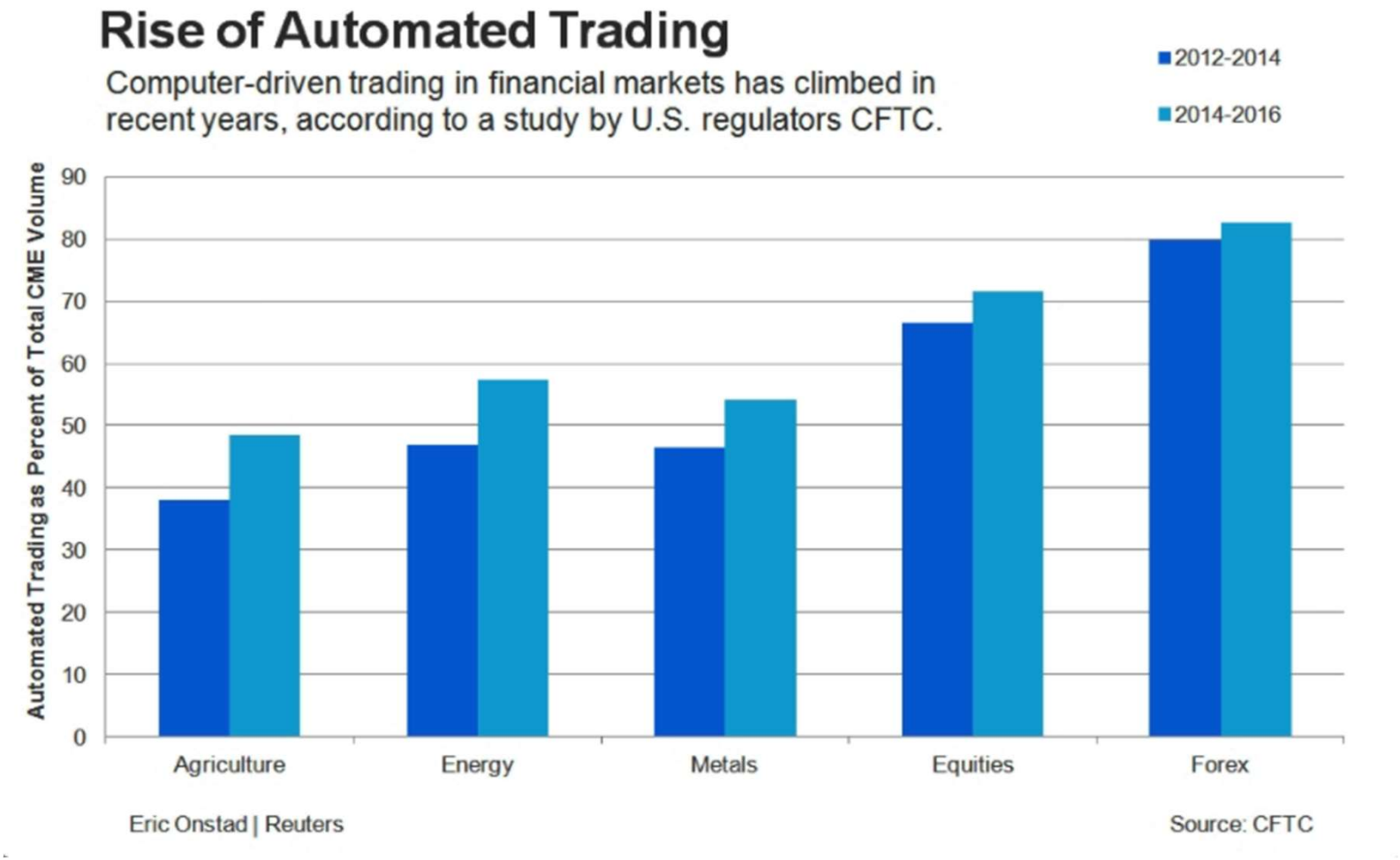
Global eSolutions (HK) Limited

Background

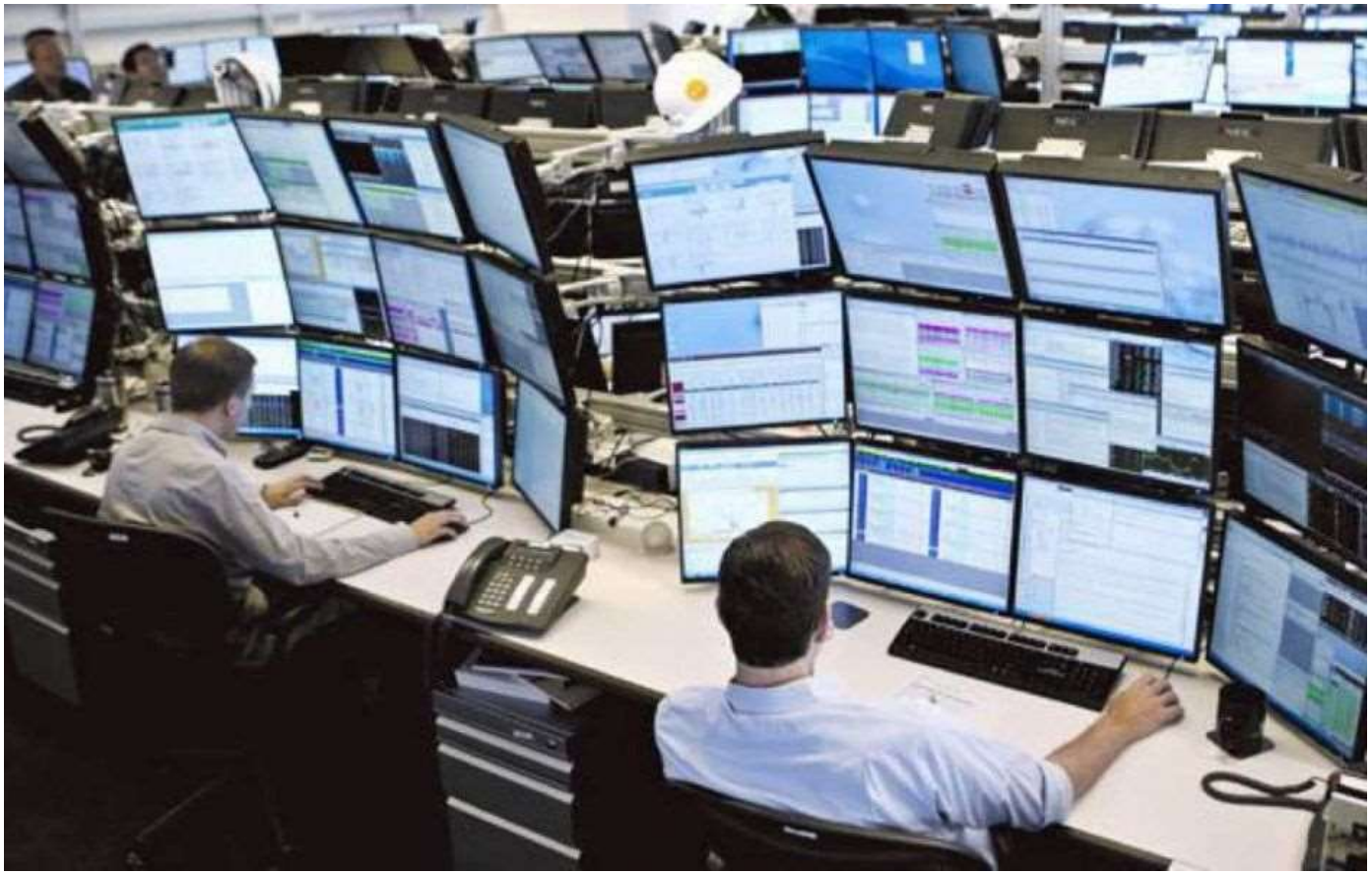
Background that has been shaping the finance industry:

- Discount Brokerage (1980s-1990s)
- Electronic Trading (NASDAQ: 1971, Globex: 1992)
- Exchange Traded Funds ETFs (1993)
- Securitization, MBS, and CDOs (2000 - 2008)
- Actively Managed ETFs (2008)
- Low-latency dark pools (2008-2013)
- Robo-advisors (2013-2020)
- ??? (2020-Future)

Trades by Volume



Traders – The Outsider's View



The Reality – Two different perspective

Trader's View

To understand what causes price to move with statistics, and take advantage of these movements. Make hundreds and thousands of trades that each have an "edge" – a higher probability of winning than losing.

Investor's View

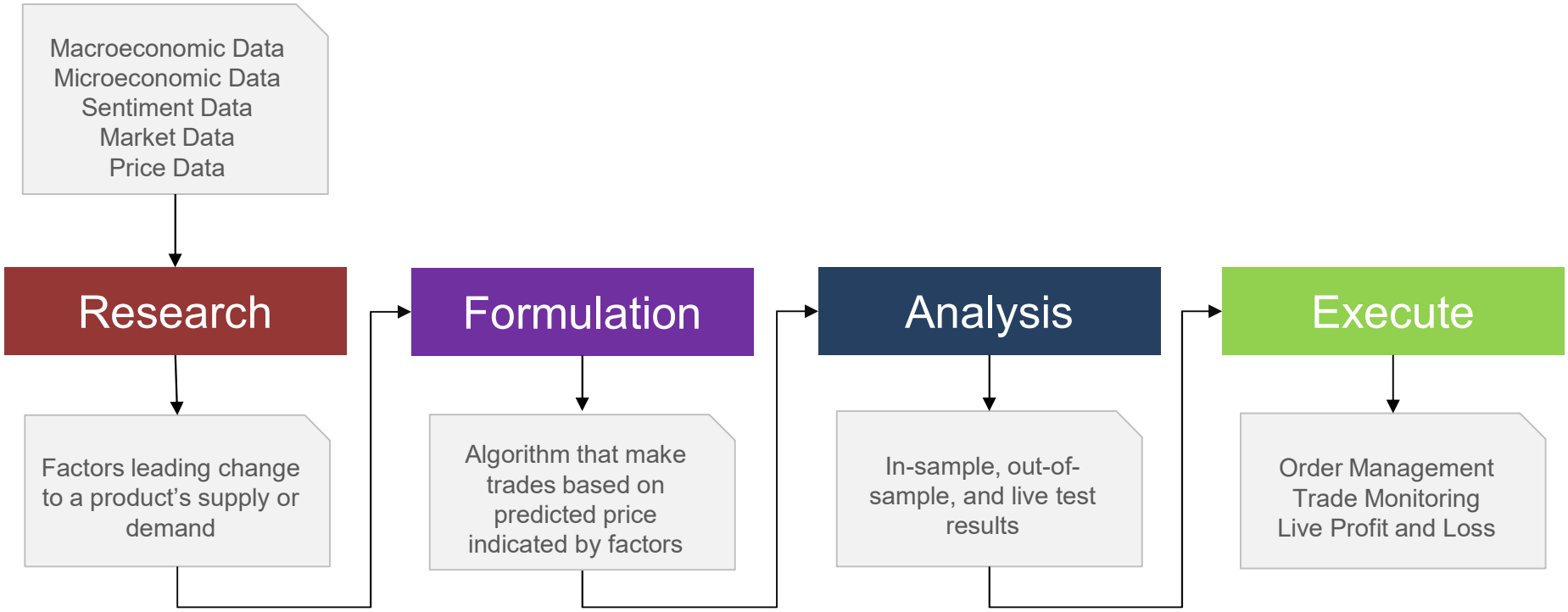
Provide a service to the financial market in return for a profit, similar to running a business. If you cannot explain the value provided then there should be no profit.

A Trader's Skills

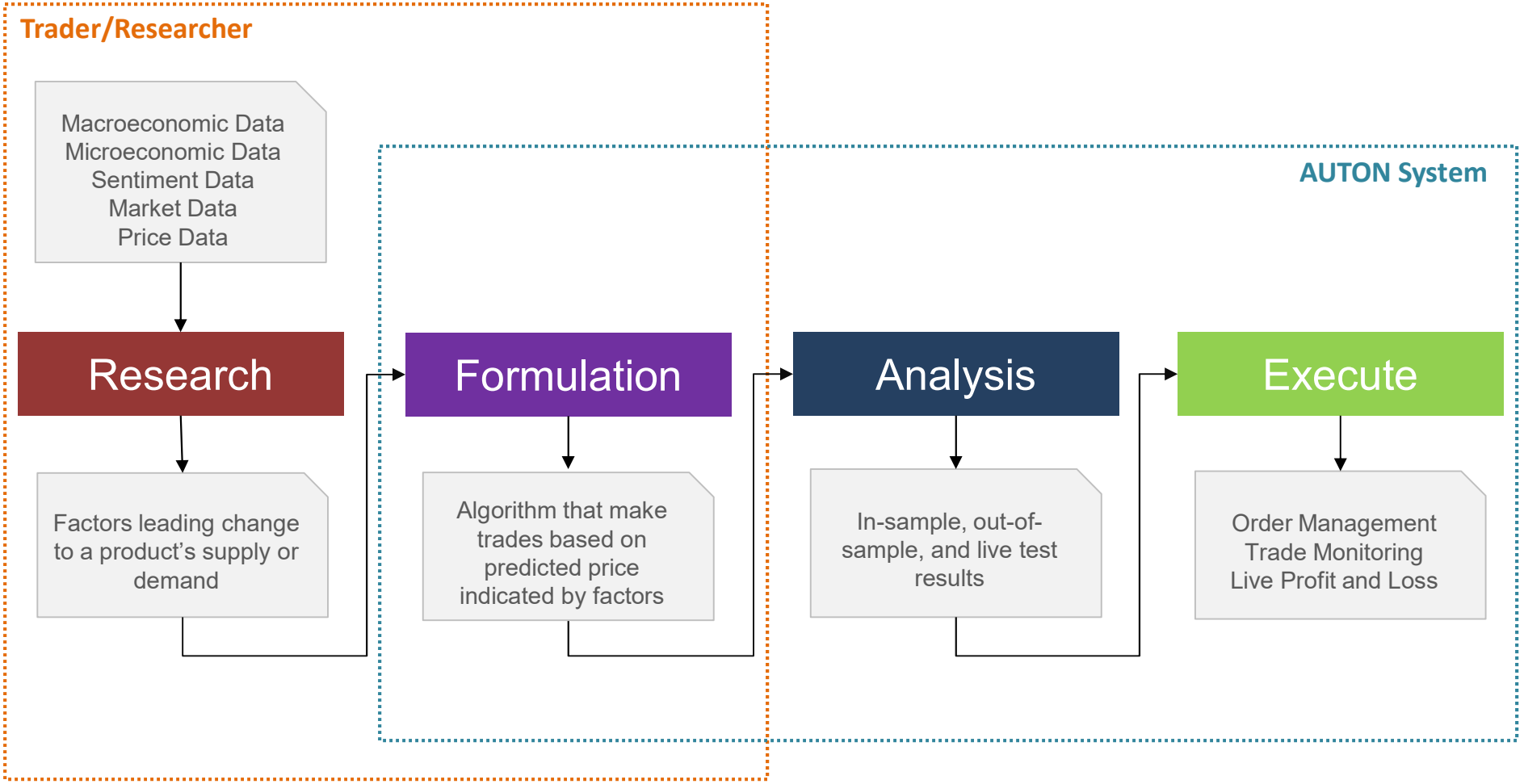
Becoming a successful trader requires a combination of abilities to:

1. Find profitable trading opportunities
2. Measure and control trading risk
3. Get access to high quality trading venue
4. Control cost
5. Managing liquidity and orders
6. Monitoring your trades

Overview of a Trading Process



Overview of a Trading Process



Overview of AUTON

AUTON is GES's proprietary multi-asset, all-inclusive solution for the **formulation**, **analysis**, and **execution** of trading algorithms. Which is designed with three components within the platform:

1. AUTON Studio (**Formulation**)
2. Algo Backtester (**Analysis**)
3. Algo Executor (**Execution**)

AUTON Studio / AlgoTrade Script

AlgoTrade script is our proprietary C# based script language used for directing AUTON into making trade execution based on various conditions.

AUTON's C#-based infrastructure means researchers have the power of **Microsoft's .NET interface** at their disposal. Researchers will have access to the same libraries our developers use: **SQL Database, HTTP Requester, TCP Sockets, File I/O, and many more.**

A Simple Momentum Strategy

Usage: Believe a security has a long term upward trend, but wish to protect from any periodic drawdown.

Strategy: If price went up in the past <period> (Year, Month, Day, etc), then we will long the security for the next one <period> , else, we will hold cash.

Psuedo Algorithm: If CLOSE(T-0) > CLOSE(T-1) then LONG X shares of security, else SETTLE until position = 0.

AlgoTrade Script Skeleton

```
public class AI : Program
{
    // These parameters will be prompted for input before AlgoTrade launch.
    // Set default values here and confirm/change them before running AlgoTrade.
    [InputParameter] public int InputInt = 30;
    [InputParameter] public double InputDouble = 3.14;
    [InputParameter] public string InputString = "EXAlgoTrade";
    [InputParameter] public bool InputBool = false;
    [InputParameter] public Color InputColor = Color.Red;
    [InputParameter] public DateTime InputDateTime = DateTime.UtcNow;

    public AI(IEXAlgoTradeFunction<double> func) : base(func)
    {
        // Do not modify this
    }

    public override void Start()
    {
        // When this AlgoTrade program has started
        string message = string.Format("Start,  InputInt:{0}  InputDouble:{1}  InputString:{
            InputInt, InputDouble, InputString, InputBool, InputColor, InputDateTime);
        Log(message);
    }

    public override void OnTick()
    {
        // When tick is updated, from chart or from market data
        IChartPoint<double> point = ChartPoints.LastOrDefault();
        Log("OnTick. Point count:" + ChartPoints.Count + ". LastNominal:" + (point != null ?
    }
```

Input Parameters

Constructor

Start

OnTick Callback

AlgoTrade Script Skeleton

Order Callback

```
public override void OnOrderExecuted(IOrder order, DateTime tradeTime, decimal price, de
{
    // When an order is executed (partially or fully)
    Log("OnOrderExecuted, orderNo:" + order.OrderNo);
}

public override void OnOrderChanged(IOrder order, DateTime tradeTime, decimal? price, de
{
    // When an order is changed
    Log("OnOrderChanged, orderNo:" + order.OrderNo);
}
```

Bid/Ask Callback

```
public override void OnBidPrice(decimal bid)
{
    // When bid price arrived. Works only when program type is RUN_WITH_PRICE.
    Log("OnBidPrice, new bid:" + bid);
}

public override void OnAskPrice(decimal ask)
{
    // When ask price arrived. Works only when program type is RUN_WITH_PRICE.
    Log("OnAskPrice, new ask:" + ask);
}

public override void OnBidPriceDepth(IList<IPriceDepthItem> bidDepth)
{
    // When bid price depth arrived. Works only when program type is RUN_WITH_PRICE.
    Log("OnBidPriceDepth, depth total order count:" + bidDepth.Select(x => x.OrderCount)
}
```

AlgoTrade Script Skeleton

Bid/Ask Callback

```
public override void OnAskPriceDepth(IList<IPriceDepthItem> askDepth)
{
    // When ask price depth arrived. Works only when program type is RUN_WITH_PRICE.
    Log("OnAskPriceDepth, depth total order count:" + askDepth.Select(x => x.OrderCount
}

public override void OnTradeTicker(IList<ITradeTickerItem> tt)
{
    // When trade ticker arrived. Works only when program type is RUN_WITH_PRICE.
    ITradeTickerItem item = tt.FirstOrDefault();
    if (item != null)
        Log("OnTradeTicker, last price:" + item.Price + " last vol:" + item.Volume);
}

public override void OnBidOrderDepth(IList<IOrderDepthPriceItem> bidDepth)
{
    // When bid order depth arrived. Works only when program type is RUN_WITH_PRICE.
    Log("OnBidOrderDepth, depth price count:" + bidDepth.Count + " order count:" + bidD
}

public override void OnAskOrderDepth(IList<IOrderDepthPriceItem> askDepth)
{
    // When ask order depth arrived. Works only when program type is RUN_WITH_PRICE.
    Log("OnAskOrderDepth, depth price count:" + askDepth.Count + " order count:" + askD
}
```

AlgoTrade Script Skeleton

Time Trigger

```
public override void OnSecond()
{
    // Called every second (approx.)
    Log("OnSecond");
}

public override void OnMinute()
{
    // Called every minute (approx.)
    Log("OnMinute");
}

public override void OnHour()
{
    // Called every hour (approx.)
    Log("OnHour");
}
```

AUTON Backtester

All strategies written in **AlgoScript** can be back tested via **AUTON Backtester**.

User can input their own historical pricing data via a file, with variable input parameters.

AUTON's server includes back-end database of a *current limit of* 2,000 points of historical price data (Minute, Hourly, Daily, Weekly, And Monthly) for major instruments from the **Forex**, **Hong Kong Equities**, and **Commodities market**.

AUTON Backtester

AUTON Executor

Once an algorithm has been thoroughly back-tested, the **AUTON Algo Executor** can be used to forward-test the same script via a **demonstration account**.

The executor removes the need of a live trader in spending time to monitor market conditions throughout the day.

Once the algorithm has proven itself profitable, it can be transferred to a **live account** for live trade.

Back-testing Limitations

OnTick Frequency Misconception

In back-testing, the OnTick() function is called at the frequency as defined by the selected frequency of data used to run the back-test. So if the selected frequency / period of data is Daily Close for 3 years, the OnTick function will only be called once a day for 1095 days.

However, when the same script is put to live run - the OnTick function is called far more frequently - typically once every few seconds whenever a new tick arrives.

Back-testing Limitations

Limit Order Misconception

Limit Order must be used with caution during back-testing because the quotation arrive on a discrete basis on the selected interval, so any true quotation that exists in between the interval will not trigger the limit to execute. If possible, try to stick with Market Order during the back-testing stage.

Back-testing Limitations

Dividends, Splits, and Interests

The historical price of the securities in AUTON are NOT adjusted for dividends, splits, and interests to preserve the true price at historical point of time.

However AUTON has the C# functionality read and load external file with which coder can use to load in dividend information using C# StreamReader. Or user can input their custom adjusted price for back-testing.

Back-testing Limitations

Dividends, Splits, and Interests

We also have a tool which would allow you to convert downloaded historical price data to a format that can be loaded into AUTON for execution.

The tool allows for both the use of adjusted and adjusted price in the back-testing of algorithms.

Back-testing Limitations

HKEX Limit Price Validity

Hong Kong Stock Exchange has restriction on the submission of Limit Order price requiring the price be rounded to the closest decimal points in reference to the price of the security.

To aid traders in rounding their desired limit price as according to HKEX requirement, AUTON has included the function **RoundPriceBySpread** for calculation of the closest HKEX valid price.

Back-testing Limitations

HKEX Limit Price Validity

SECOND SCHEDULE

Spread Table

(applicable to all types of currencies)

Part A

All securities, other than those securities covered under Part B and/or Part C, shall be traded in accordance with the following scale of spreads:

Currency unit

From	0.01 to	0.25	_____	0.001
Over	0.25 to	0.50	_____	0.005
Over	0.50 to	10.00	_____	0.010
Over	10.00 to	20.00	_____	0.020
Over	20.00 to	100.00	_____	0.050
Over	100.00 to	200.00	_____	0.100
Over	200.00 to	500.00	_____	0.200
Over	500.00 to	1,000.00	_____	0.500
Over	1,000.00 to	2,000.00	_____	1.000
Over	2,000.00 to	5,000.00	_____	2.000
Over	5,000.00 to	9,995.00	_____	5.000

Back-testing Limitations

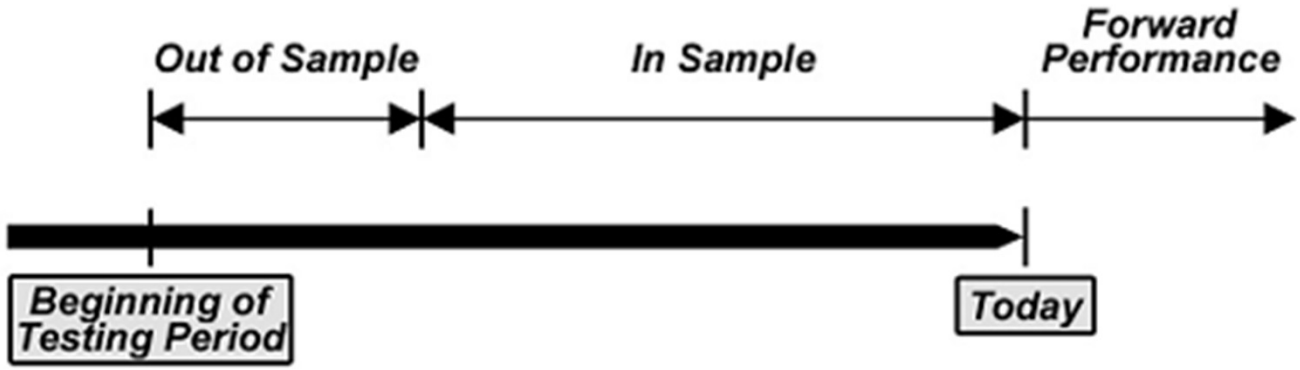
Data mining / Selection Bias

Traders must take special care in ensuring that he/she does not simply tailor the back-test parameters simply to make the historical performance “work”.

An out-of-sample test, and a live run test must be conducted to ensure the data is free of such bias.

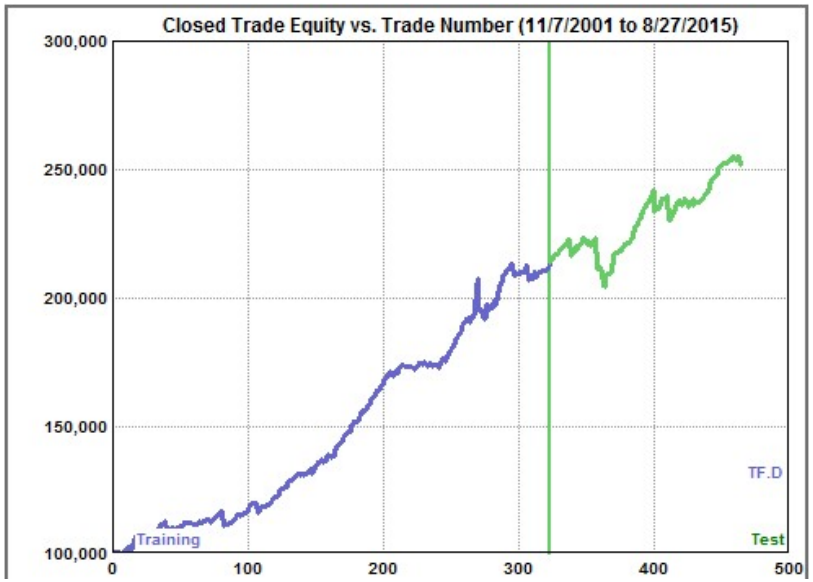
Back-testing Limitations

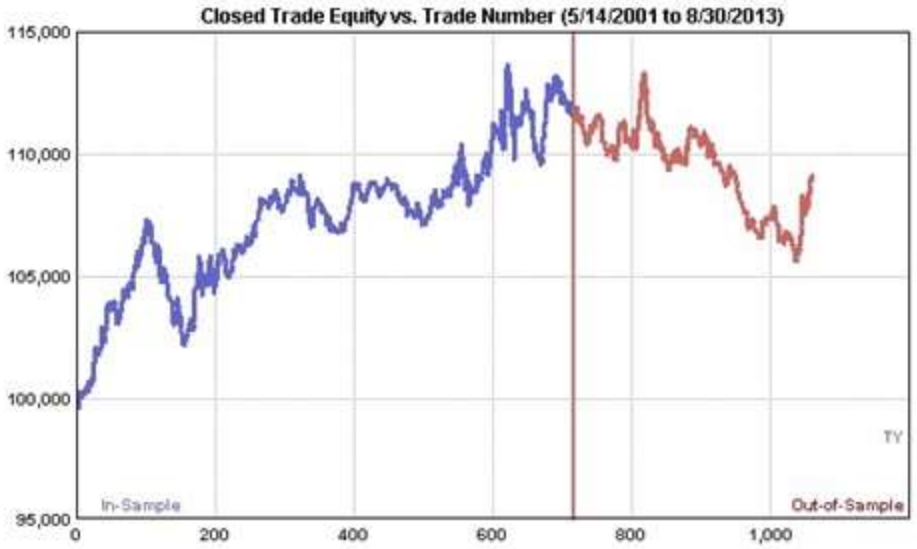
Proper Backtesting



Back-testing Limitations

In Sample / Out of Sample Testing





Recommendation

Step by step procedure:

1. Conduct research on viable trading strategy method using SSRN / Google Scholar.
2. Select a few viable strategies to be tested
3. Code strategy in back-tester – determine if the strategy is profitable or not.
4. Conduct In-Sample / Out-of-Sample testing.
5. Conduct live forward test.
6. Execute the trading strategy on live account.