# Not Over Thinking

## Paris Trading with International ETFs

Algorithmic Trading Strategy with Full Code

Haixiang
2023.07 | Vol 18.
hxyan.2015@gmail.com | github.com/hxyan2020

## STRATEGY & ECONOMIC RATIONALE

The investment universe is comprised of 22 international ETFs, with each ETF having a normalized cumulative total return index (including dividends) created, where the starting price during the formation period is set to $1 using price normalization.

After a 120-day formation period, pairs of ETFs are selected based on their distance, which is calculated as the sum of squared deviations between two normalized price series. The five pairs with the smallest distance are used for trading during the subsequent 20-day period.

The trading strategy is monitored daily, and a trade is opened when the divergence between the pairs exceeds 0.5 times the historical standard deviation. In this strategy, investors buy the undervalued ETF and sell the overvalued ETF (i.e., take a long position on the undervalued ETF and a short position on the overvalued ETF). The trade is closed if a pair converges or after 20 days if the pair does not converge within the next 20 business days.

| BUY | SELL |
|---|---|
| the undervalued ETF | the overvalued ETF |

## PARAMETER & VARIABLES

| PARAMETER | VALUE |
|---|---|
| **MARKETS TRADED** | Equities |
| **FINANCIAL INSTRUMENTS** | ETFs |
| **REGION** | Global |
| **PERIOD OF REBALANCING** | Daily |
| **NO. OF TRADED INSTRUMENTS** | 22 |
| **WEIGHTING** | Equally |
| **LOOKBACK PERIODS** | 120 days |
| **HOLDING PERIODS** | 20 days |
| **LONG/SHORT** | Long Only |

## ALGORITHM

```
import numpy as np
from AlgorithmImports import *
import itertools as it

class PairsTradingwithCountryETFs(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2000, 1, 1)
        self.SetCash(100000)

        self.symbols = [
                        "EWA",  # iShares MSCI Australia Index ETF
                        "EWO",  # iShares MSCI Austria Investable Mkt Index ETF
                        "EWK",  # iShares MSCI Belgium Investable Market Index ETF
                        "EWZ",  # iShares MSCI Brazil Index ETF
                        "EWC",  # iShares MSCI Canada Index ETF
                        "FXI",  # iShares China Large-Cap ETF
                        "EWQ",  # iShares MSCI France Index ETF
                        "EWG",  # iShares MSCI Germany ETF
```

```
                            "EWH",  # iShares MSCI Hong Kong Index ETF
                            "EWI",  # iShares MSCI Italy Index ETF
                            "EWJ",  # iShares MSCI Japan Index ETF
                            "EWM",  # iShares MSCI Malaysia Index ETF
                            "EWW",  # iShares MSCI Mexico Inv. Mt. Idx
                            "EWN",  # iShares MSCI Netherlands Index ETF
                            "EWS",  # iShares MSCI Singapore Index ETF
                            "EZA",  # iShares MSCI South Africe Index ETF
                            "EWY",  # iShares MSCI South Korea ETF
                            "EWP",  # iShares MSCI Spain Index ETF
                            "EWD",  # iShares MSCI Sweden Index ETF
                            "EWL",  # iShares MSCI Switzerland Index ETF
                            "EWT",  # iShares MSCI Taiwan Index ETF
                            "THD",  # iShares MSCI Thailand Index ETF
                            "EWU",  # iShares MSCI United Kingdom Index ETF
                            "SPY",  # SPDR S&P 500 ETF
                            ]

        self.period = 120
        self.max_traded_pairs = 5 # The top 5 pairs with the smallest distance are used.

        self.history_price = {}
        self.traded_pairs = []
        self.traded_quantity = {}

        for symbol in self.symbols:
            data = self.AddEquity(symbol, Resolution.Daily)
            data.SetFeeModel(CustomFeeModel())
            data.SetLeverage(5)
            symbol_obj = data.Symbol

            if symbol not in self.history_price:
                self.history_price[symbol] = RollingWindow[float](self.period)

                history = self.History(self.Symbol(symbol), self.period, Resolution.Daily)
                if history.empty:
                    self.Log(f"Note enough data for {symbol} yet")
                else:
                    closes = history.loc[symbol].close[:-1]
                    for time, close in closes.iteritems():
                        self.history_price[symbol].Add(close)

        self.sorted_pairs = []
        self.symbol_pairs = list(it.combinations(self.symbols, 2))
        self.days = 20

    def OnData(self, data):
        # Update the price series everyday
        for symbol in self.history_price:
            symbol_obj = self.Symbol(symbol)
            if symbol_obj in data and data[symbol_obj]:
                price = data[symbol_obj].Value
                self.history_price[symbol].Add(price)

        # Start of trading period.
        if self.days == 20:
            # minimize the sum of squared deviations
            distances = {}
            for pair in self.symbol_pairs:
                if self.history_price[pair[0]].IsReady and self.history_price[pair[1]].IsReady:
                    if (self.Time.date() - self.Securities[pair[0]].GetLastData().Time.date()).
days <= 3 and (self.Time.date() - self.Securities[pair[1]].GetLastData().Time.date()).days <= 3:
                        distances[pair] = self.Distance([x for x in self.history_price[pair
[0]]], [x for x in self.history_price[pair[1]]])
```

```
            if len(distances) != 0:
                self.sorted_pairs = sorted(distances.items(), key = lambda x: x[1])[:self.max_t
raded_pairs]
                self.sorted_pairs = [x[0] for x in self.sorted_pairs]

            self.Liquidate()
            self.traded_pairs.clear()
            self.traded_quantity.clear()

            self.days = 0

        self.days += 1

        if self.sorted_pairs is None: return

        pairs_to_remove = []

        for pair in self.sorted_pairs:
            # Calculate the spread of two price series.
            price_a = [x for x in self.history_price[pair[0]]]
            price_b = [x for x in self.history_price[pair[1]]]
            norm_a = np.array(price_a) / price_a[-1]
            norm_b = np.array(price_b) / price_b[-1]

            spread = norm_a - norm_b
            mean = np.mean(spread)
            std = np.std(spread)
            actual_spread = spread[0]

            # Long-short position is opened when pair prices have diverged by two standard devi
ations.
            traded_portfolio_value = self.Portfolio.TotalPortfolioValue / self.max_traded_pairs
            if actual_spread > mean + 0.5*std or actual_spread < mean - 0.5*std:
                if pair not in self.traded_pairs:
                    # open new position for pair, if there's place for it.
                    if len(self.traded_pairs) < self.max_traded_pairs:
                        symbol_a = pair[0]
                        symbol_b = pair[1]
                        a_price_norm = norm_a[0]
                        b_price_norm = norm_b[0]
                        a_price = price_a[0]
                        b_price = price_b[0]

                        # a etf's price > b etf's price
                        if a_price_norm > b_price_norm:
                            long_q = traded_portfolio_value / b_price    # long b etf
                            short_q = -traded_portfolio_value / a_price  # short a etf
                            if self.Securities.ContainsKey(symbol_a) and self.Securities.Contai
nsKey(symbol_b) and \
                                self.Securities[symbol_a].Price != 0 and self.Securities[symbol
_a].IsTradable and \
                                self.Securities[symbol_b].Price != 0 and self.Securities[symbol
_b].IsTradable:
                                self.MarketOrder(symbol_a, short_q)
                                self.MarketOrder(symbol_b, long_q)

                                self.traded_quantity[pair] = (short_q, long_q)
                                self.traded_pairs.append(pair)
                        # b etf's price > a etf's price
                        else:
                            long_q = traded_portfolio_value / a_price    # long a etf
                            short_q = -traded_portfolio_value / b_price  # short b etf
```

```
                    if self.Securities.ContainsKey(symbol_a) and self.Securities.Contai
nsKey(symbol_b) and \
                            self.Securities[symbol_a].Price != 0 and self.Securities[symbol
_a].IsTradable and \
                            self.Securities[symbol_b].Price != 0 and self.Securities[symbol
_b].IsTradable:
                            self.MarketOrder(symbol_a, long_q)
                            self.MarketOrder(symbol_b, short_q)

                            self.traded_quantity[pair] = (long_q, short_q)
                            self.traded_pairs.append(pair)
            # The position is closed when prices revert back.
            else:
                if pair in self.traded_pairs and pair in self.traded_quantity:
                    # make opposite order to opened position
                    self.MarketOrder(pair[0], -self.traded_quantity[pair][0])
                    self.MarketOrder(pair[1], -self.traded_quantity[pair][1])
                    pairs_to_remove.append(pair)

        for pair in pairs_to_remove:
            self.traded_pairs.remove(pair)
            del self.traded_quantity[pair]

    def Distance(self, price_a, price_b):
        # Calculate the sum of squared deviations between two normalized price series.
        norm_a = np.array(price_a) / price_a[-1]
        norm_b = np.array(price_b) / price_b[-1]
        return sum((norm_a - norm_b)**2)

# Custom fee model.
class CustomFeeModel(FeeModel):
    def GetOrderFee(self, parameters):
        fee = parameters.Security.Price * parameters.Order.AbsoluteQuantity * 0.00005
        return OrderFee(CashAmount(fee, "USD"))
```

## BACKTESTING PERFORMANCE


*Fig 1. Overall Performance*

| | | | |
|---|---|---|---|
| PSR | 0.126% | Sharpe Ratio | 0.392 |
| Total Trades | 9994 | Average Win | 0.40% |
| Average Loss | -0.38% | Compounding Annual Return | 3.209% |
| Drawdown | 13.200% | Expectancy | 0.033 |
| Net Profit | 70.819% | Loss Rate | 50% |
| Win Rate | 50% | Profit-Loss Ratio | 1.06 |
| Alpha | 0.023 | Beta | 0.027 |
| Annual Standard Deviation | 0.061 | Annual Variance | 0.004 |
| Information Ratio | -0.121 | Tracking Error | 0.169 |
| Treynor Ratio | 0.9 | Total Fees | $13644.12 |
| Estimated Strategy Capacity | $240000.00 | Lowest Capacity Asset | EWN R735QTJ8XC9X |

*Fig 2. Performance Metrics*


*Fig 3. Drawdown*

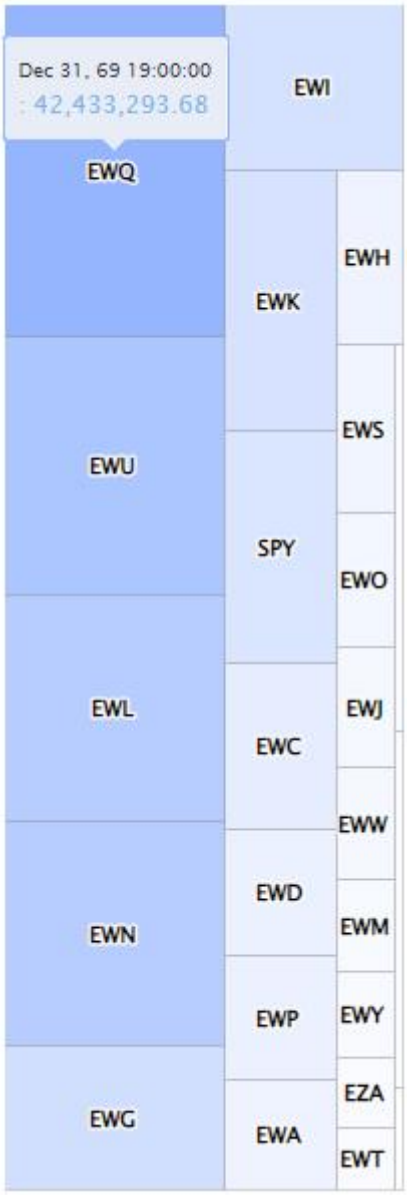


*Fig 4. Assets Sales Volume*