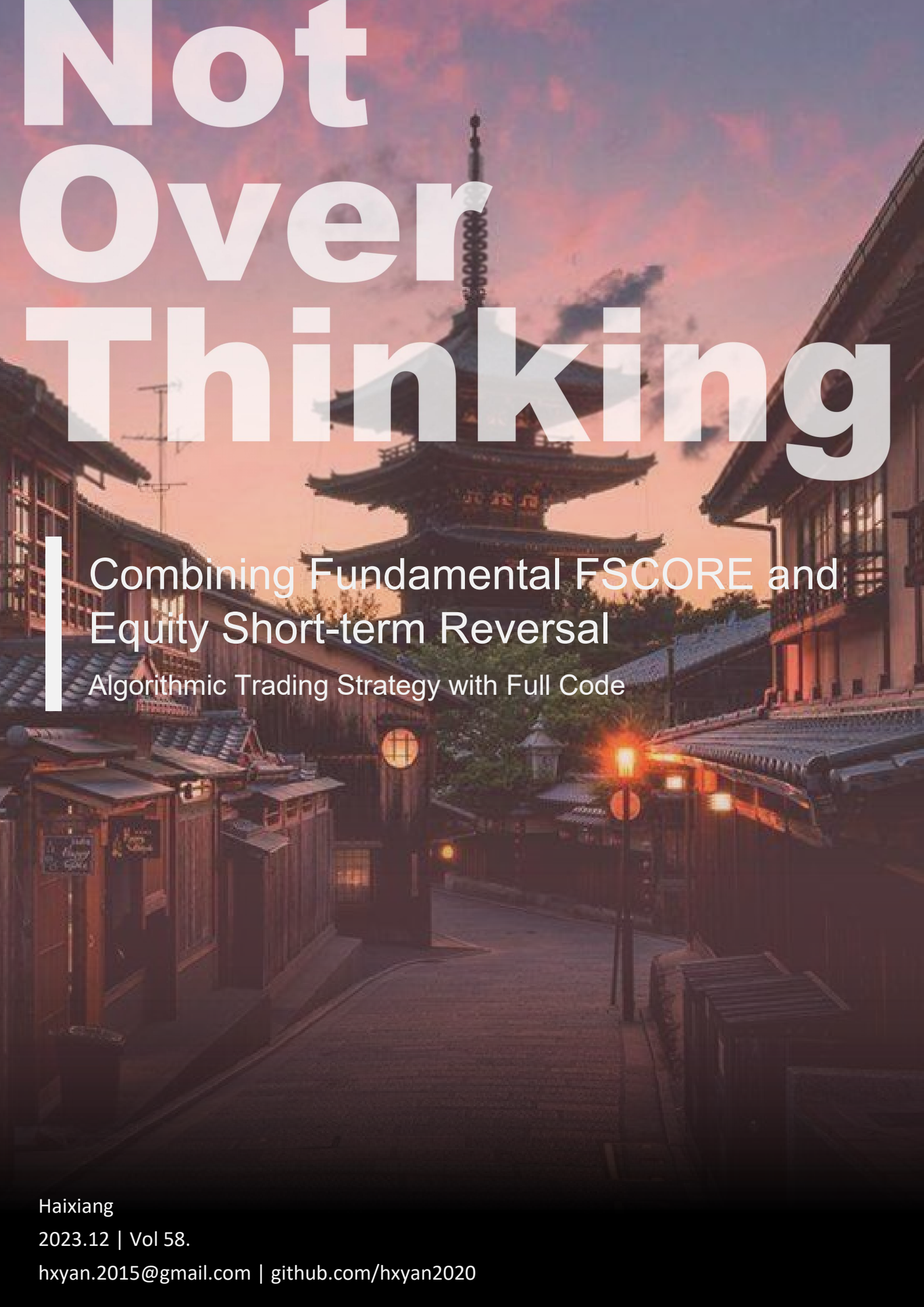

Not Over Thinking

Combining Fundamental FSCORE and Equity Short-term Reversal

Algorithmic Trading Strategy with Full Code

Haixiang

2023.12 | Vol 58.

hxyan.2015@gmail.com | github.com/hxyan2020

STRATEGY & ECONOMIC RATIONALE

The investment universe consists of common stocks (share code 10 or 11) listed in NYSE, AMEX, and NASDAQ exchanges. Stocks with prices less than $5 at the end of the formation period are excluded.

The range of FSCORE is from zero to nine points. Each signal is equal to one (zero) point if the signal indicates a positive (negative) financial performance. A firm scores one point if it has realized a positive return-on-assets (ROA), positive cash flow from operations, a positive change in ROA, a positive difference between net income from operations (Accrual), a decrease in the ratio of long-term debt to total assets, a positive change in the current ratio, no-issuance of new common equity, a positive change in gross margin ratio and lastly a positive change in asset turnover ratio. Firstly, construct a quarterly FSCORE using the most recently available quarterly financial statement information.

Monthly reversal data are matched each month with a most recently available quarterly FSCORE. The firm is classified as a fundamentally strong firm if the firm's FSCORE is greater than or equal to seven (7-9), fundamentally middle firm (4-6) and fundamentally weak firm (0-3). Secondly, identify the large stocks subset – those in the top 40% of all sample stocks in terms of market capitalization at the end of formation month t. After that, stocks are sorted on the past 1-month returns and firm's most recently available quarterly FSCORE. Take a long position in past losers with favourable fundamentals (7-9) and simultaneously a short position in past winners with unfavourable fundamentals (0-3). The strategy is equally weighted and rebalanced monthly.

BUY	SELL
(see above)	(see above)

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equity
FINANCIAL INSTRUMENTS	Stocks
REGION	United States
PERIOD OF REBALANCING	Monthly
NO. OF TRADED INSTRUMENTS	1000
WEIGHTING	Equal weighting
LOOKBACK PERIODS	Depends
LONG/SHORT	Long & Short

ALGORITHM

<fk_tools.py>

```
from AlgorithmImports import *

import numpy as np

def Return(values):
    return (values[-1] - values[0]) / values[0]

def Volatility(values):
    values = np.array(values)
    returns = (values[1:] - values[:-1]) / values[:-1]
```

```
    return np.std(returns)

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

# Quandl free data
class QuandlFutures(PythonQuandl):
    def __init__(self):
        self.ValueColumnName = "settle"

# Quantpedia data
# NOTE: IMPORTANT: Data order must be ascending (datewise)
class QuantpediaFutures(PythonData):
    def GetSource(self, config, date, isLiveMode):
        return SubscriptionDataSource("data.quantpedia.com/backtesting_data/futures/{0}.csv".fo
rmat(config.Symbol.Value), SubscriptionTransportMedium.RemoteFile, FileFormat.Csv)

    def Reader(self, config, line, date, isLiveMode):
        data = QuantpediaFutures()
        data.Symbol = config.Symbol

        if not line[0].isdigit(): return None
        split = line.split(';')

        data.Time = datetime.strptime(split[0], "%d.%m.%Y") + timedelta(days=1)
        data['settle'] = float(split[1])
        data.Value = float(split[1])

        return data
```

<main.py>

```
from AlgorithmImports import *

# Monthly reversal data are matched each month with a most recently available quarterly FSCORE.
 The firm is classified as a fundamentally
# strong firm if the firm's FSCORE is greater than or equal to seven (7-9), fundamentally middl
e firm (4-6) and fundamentally weak firm (0-3).
# Secondly, identify the large stocks subset – those in the top 40% of all sample stocks in ter
ms of market capitalization
# at the end of formation month t. After that, stocks are sorted on the past 1-month returns an
d firm's most recently available quarterly FSCORE.
# Take a long position in past losers with favorable fundamentals (7-9) and simultaneously a sh
ort position in past winners with unfavorable
# fundamentals (0-3). The strategy is equally weighted and rebalanced monthly.
#
# QC implementation changes:
#   - Instead of all listed stock, we select 500 most liquid stocks traded on NYSE, AMEX, or NA
SDAQ.

class CombiningFSCOREShortTermReversals(QCAlgorithm):

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

        self.SetSecurityInitializer(lambda x: x.SetMarketPrice(self.GetLastKnownPrice(x)))

        self.coarse_count = 500
```

```
        self.long = []
        self.short = []

        self.stock_data = {}
        self.data = {}
        self.period = 21
        self.symbol = self.AddEquity('SPY', Resolution.Daily).Symbol

        self.selection_flag = False
        self.UniverseSettings.Resolution = Resolution.Daily
        self.AddUniverse(self.CoarseSelectionFunction, self.FineSelectionFunction)
        self.Schedule.On(self.DateRules.MonthEnd(self.symbol), self.TimeRules.AfterMarketOpen(s
elf.symbol), self.Selection)

    def OnSecuritiesChanged(self, changes):
        for security in changes.AddedSecurities:
            security.SetFeeModel(CustomFeeModel())
            security.SetLeverage(10)

    def CoarseSelectionFunction(self, coarse):
        # Update the rolling window every day.
        for stock in coarse:
            symbol = stock.Symbol

            # Store monthly price.
            if symbol in self.data:
                self.data[symbol].update(stock.AdjustedPrice)

        if not self.selection_flag:
            return Universe.Unchanged

        # selected = [x.Symbol for x in coarse if x.HasFundamentalData and x.Market == 'usa' an
d x.Price > 5]
        selected = [x.Symbol
            for x in sorted([x for x in coarse if x.HasFundamentalData and x.Market == 'usa' an
d x.Price > 5],
                key = lambda x: x.DollarVolume, reverse = True)[:self.coarse_count]]

        # Warmup price rolling windows.
        for symbol in selected:
            if symbol in self.data:
                continue

            self.data[symbol] = SymbolData(symbol, self.period)
            history = self.History(symbol, self.period, Resolution.Daily)
            if history.empty:
                self.Log(f"Not enough data for {symbol} yet.")
                continue
            closes = history.loc[symbol].close
            for time, close in closes.iteritems():
                self.data[symbol].update(close)

        return [x for x in selected if self.data[x].is_ready()]

    def FineSelectionFunction(self, fine):
        fine = [x for x in fine if (x.EarningReports.BasicAverageShares.ThreeMonths != 0) and
                                   (x.EarningReports.BasicEPS.TwelveMonths != 0) and
                                   (x.ValuationRatios.PERatio != 0) and
                                   (x.OperationRatios.ROA.ThreeMonths != 0) and
                                   (x.FinancialStatements.CashFlowStatement.CashFlowFromContin
uingOperatingActivities.ThreeMonths != 0) and
                                   (x.FinancialStatements.IncomeStatement.NormalizedIncome.Thr
eeMonths != 0) and
```

```
                                    (x.FinancialStatements.BalanceSheet.LongTermDebt.ThreeMonth
s != 0) and
                                    (x.FinancialStatements.BalanceSheet.TotalAssets.ThreeMonths
 != 0) and
                                    (x.OperationRatios.CurrentRatio.ThreeMonths != 0) and
                                    (x.FinancialStatements.BalanceSheet.OrdinarySharesNumber.Th
reeMonths != 0) and
                                    (x.OperationRatios.GrossMargin.ThreeMonths != 0) and
                                    (x.FinancialStatements.IncomeStatement.TotalRevenueAsReport
ed.ThreeMonths != 0) and
                                    ((x.SecurityReference.ExchangeId == "NYS") or (x.SecurityRe
ference.ExchangeId == "NAS") or (x.SecurityReference.ExchangeId == "ASE"))]

        # BM sorting
        sorted_by_market_cap = sorted(fine, key = lambda x: x.MarketCap, reverse = True)
        lenght = int((len(sorted_by_market_cap) / 100) * 40)
        top_by_market_cap = [x for x in sorted_by_market_cap[:lenght]]

        fine_symbols = [x.Symbol for x in top_by_market_cap]

        score_performance = {}

        for stock in top_by_market_cap:
            symbol = stock.Symbol

            if symbol not in self.stock_data:
                self.stock_data[symbol] = StockData()   # Contains latest data.

            roa = stock.OperationRatios.ROA.ThreeMonths
            cfo = stock.FinancialStatements.CashFlowStatement.CashFlowFromContinuingOperatingAc
tivities.ThreeMonths
            leverage = stock.FinancialStatements.BalanceSheet.LongTermDebt.ThreeMonths / stock.
FinancialStatements.BalanceSheet.TotalAssets.ThreeMonths
            liquidity = stock.OperationRatios.CurrentRatio.ThreeMonths
            equity_offering = stock.FinancialStatements.BalanceSheet.OrdinarySharesNumber.Three
Months
            gross_margin = stock.OperationRatios.GrossMargin.ThreeMonths
            turnover = stock.FinancialStatements.IncomeStatement.TotalRevenueAsReported.ThreeMo
nths / stock.FinancialStatements.BalanceSheet.TotalAssets.ThreeMonths

            # Check if data has previous year's data ready.
            stock_data = self.stock_data[symbol]
            if (stock_data.ROA == 0) or (stock_data.Leverage == 0) or (stock_data.Liquidity ==
0) or (stock_data.Equity_offering == 0) or (stock_data.Gross_margin == 0) or (stock_data.Turnov
er == 0):
                stock_data.Update(roa, leverage, liquidity, equity_offering, gross_margin, turn
over)
                continue

            score = 0

            if roa > 0:
                score += 1
            if cfo > 0:
                score += 1
            if roa > stock_data.ROA:   # ROA change is positive
                score += 1
            if cfo > roa:
                score += 1
            if leverage < stock_data.Leverage:
                score += 1
            if liquidity > stock_data.Liquidity:
                score += 1
            if equity_offering < stock_data.Equity_offering:
```

```
                score += 1
            if gross_margin > stock_data.Gross_margin:
                score += 1
            if turnover > stock_data.Turnover:
                score += 1

            score_performance[symbol] = (score, self.data[symbol].performance())

            # Update new (this year's) data.
            stock_data.Update(roa, leverage, liquidity, equity_offering, gross_margin, turnover)

        # Clear out not updated data.
        for symbol in self.stock_data:
            if symbol not in fine_symbols:
                self.stock_data[symbol] = StockData()

        # Performance sorting and F score sorting.
        self.long = [x[0] for x in score_performance.items() if x[1][0] >= 7 and x[1][1] < 0]
        self.short = [x[0] for x in score_performance.items() if x[1][0] <= 3 and x[1][1] > 0]

        return self.long + self.short

    def OnData(self, data):
        if not self.selection_flag:
            return
        self.selection_flag = False

        # Trade execution.
        long_count = len(self.long)
        short_count = len(self.short)

        stocks_invested = [x.Key for x in self.Portfolio if x.Value.Invested]
        for symbol in stocks_invested:
            if symbol not in self.long + self.short:
                self.Liquidate(symbol)

        for symbol in self.long:
            self.SetHoldings(symbol, 1 / long_count)
        for symbol in self.short:
            self.SetHoldings(symbol, -1 / short_count)

        self.long.clear()
        self.short.clear()

    def Selection(self):
        self.selection_flag = True

class StockData():
    def __init__(self):
        self.ROA = 0
        self.Leverage = 0
        self.Liquidity = 0
        self.Equity_offering = 0
        self.Gross_margin = 0
        self.Turnover = 0

    def Update(self, ROA, leverage, liquidity, eq_offering, gross_margin, turnover):
        self.ROA = ROA
        self.Leverage = leverage
        self.Liquidity = liquidity
        self.Equity_offering = eq_offering
        self.Gross_margin = gross_margin
        self.Turnover = turnover
```

```
class SymbolData():
    def __init__(self, symbol, period):
        self.Symbol = symbol
        self.Price = RollingWindow[float](period)

    def update(self, value):
        self.Price.Add(value)

    def is_ready(self) -> bool:
        return self.Price.IsReady

    def performance(self, values_to_skip = 0) -> float:
        closes = [x for x in self.Price][values_to_skip:]
        return (closes[0] / closes[-1] - 1)

# 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

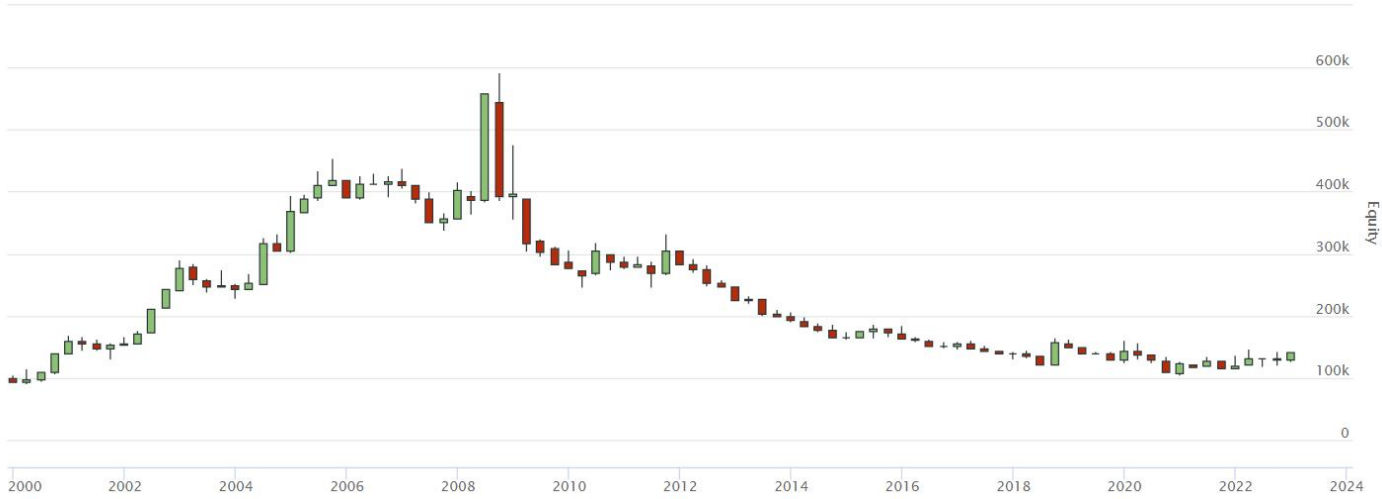

Fig 1. Overall Performance

Total Trades	21722	Average Win	0.22%
Average Loss	-0.16%	Compounding Annual Return	1.493%
Drawdown	82.500%	Expectancy	0.031
Net Profit	41.006%	Sharpe Ratio	0.148
Probabilistic Sharpe Ratio	0.000%	Loss Rate	57%
Win Rate	43%	Profit-Loss Ratio	1.41
Alpha	0.058	Beta	-0.566
Annual Standard Deviation	0.176	Annual Variance	0.031
Information Ratio	-0.104	Tracking Error	0.295
Treynor Ratio	-0.046	Total Fees	$7434.20
Estimated Strategy Capacity	$190000000.00	Lowest Capacity Asset	KHC W1YY4MGI5CH1

Fig 2. Performance Metrics