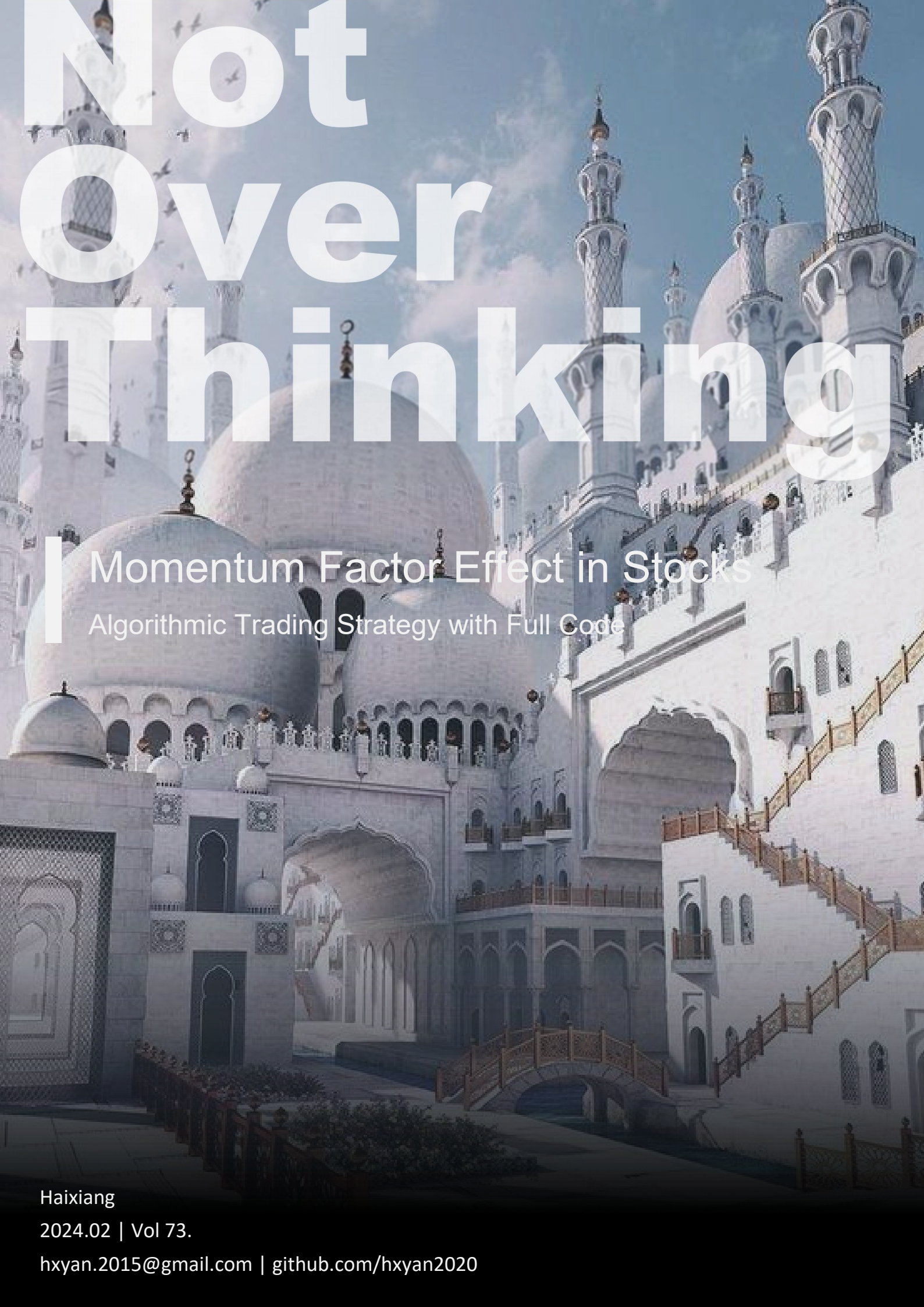

Not Over Thinking

Momentum Factor Effect in Stocks

Algorithmic Trading Strategy with Full Code

Haixiang

2024.02 | Vol 73.

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

STRATEGY & ECONOMIC RATIONALE

The investment universe consists of NYSE, AMEX, and NASDAQ stocks. We define momentum as the past 12-month return, skipping the most recent month's return (to avoid microstructure and liquidity biases). To capture "momentum", UMD portfolio goes long stocks that have high relative past one-year returns and short stocks that have low relative past one-year returns.

BUY	SELL
goes long stocks that have high relative past one-year returns	short stocks that have low relative past one-year returns

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equity
FINANCIAL INSTRUMENTS	Stocks
REGION	Unites States
PERIOD OF REBALANCING	Monthly
NO. OF TRADED INSTRUMENTS	1000
WEIGHTING	Equal weighting
LOOKBACK PERIODS	N/A
LONG/SHORT	Long & short

ALGORITHM

```
from AlgorithmImports import *
from typing import List, Dict
from pandas.core.frame import DataFrame
# endregion

class MomentumFactorEffectinStocks(QCAlgorithm):

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

        symbol:Symbol = self.AddEquity('SPY', Resolution.Daily).Symbol

        self.weight:Dict[Symbol, float] = {}
        self.data:Dcit[Symbol, RollingWindow] = {}
        self.period:int = 12 * 21
        self.quantile:int = 5
        self.leverage:int = 5

        self.coarse_count:int = 500
        self.selection_flag:bool = False
        self.UniverseSettings.Resolution = Resolution.Daily
        self.AddUniverse(self.CoarseSelectionFunction, self.FineSelectionFunction)
        self.Schedule.On(self.DateRules.MonthStart(symbol), self.TimeRules.AfterMarketOpen(symbol), self.Selection)

    def OnSecuritiesChanged(self, changes:SecurityChanges) -> None:
        for security in changes.AddedSecurities:
```

```
            security.SetFeeModel(CustomFeeModel())
            security.SetLeverage(self.leverage)

    def CoarseSelectionFunction(self, coarse:List[CoarseFundamental]) -> List[Symbol]:
        # update the rolling window every day
        for stock in coarse:
            symbol:Symbol = stock.Symbol

            # Store monthly price.
            if symbol in self.data:
                self.data[symbol].Add(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'],
                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] = RollingWindow[float](self.period)
            history:DataFrame = self.History(symbol, self.period, Resolution.Daily)
            if history.empty:
                self.Log(f"Not enough data for {symbol} yet")
                continue
            closes:pd.Series = history.loc[symbol].close
            for time, close in closes.iteritems():
                self.data[symbol].Add(close)

        return [x for x in selected if self.data[x].IsReady]

    def FineSelectionFunction(self, fine:List[FineFundamental]) -> List[Symbol]:
        fine = [x for x in fine if x.MarketCap != 0 and \
                    ((x.SecurityReference.ExchangeId == "NYS") or (x.SecurityReference.Exchange
Id == "NAS") or (x.SecurityReference.ExchangeId == "ASE"))]

        # if len(fine) > self.coarse_count:
        #     sorted_by_market_cap = sorted(fine, key = lambda x:x.MarketCap, reverse=True)
        #     top_by_market_cap = [x for x in sorted_by_market_cap[:self.coarse_count]]
        # else:
        #     top_by_market_cap = fine

        perf:Dict[Symbol, float] = {x.Symbol : self.data[x.Symbol][0] / self.data[x.Symbol][sel
f.period-1] - 1 for x in fine}

        if len(perf) >= self.quantile:
            sorted_by_perf:List = sorted(perf.items(), key = lambda x:x[1], reverse=True)
            quantile:int = int(len(sorted_by_perf) / self.quantile)
            long:List[Symbol] = [x[0] for x in sorted_by_perf[:quantile]]
            short:List[Symbol] = [x[0] for x in sorted_by_perf[-quantile:]]

            long_count:int = len(long)
            short_count:int = len(short)

            for symbol in long:
                self.weight[symbol] = 1 / long_count
            for symbol in short:
                self.weight[symbol] = -1 / short_count
```

```
        return list(self.weight.keys())

    def OnData(self, data:Slice) -> None:
        if not self.selection_flag:
            return
        self.selection_flag = False

        # trade execution
        stocks_invested:List[Symbol] = [x.Key for x in self.Portfolio if x.Value.Invested]
        for symbol in stocks_invested:
            if symbol not in self.weight:
                self.Liquidate(symbol)

        for symbol, w in self.weight.items():
            if symbol in data and data[symbol]:
                self.SetHoldings(symbol, w)

        self.weight.clear()

    def Selection(self) -> None:
        self.selection_flag = True
# 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

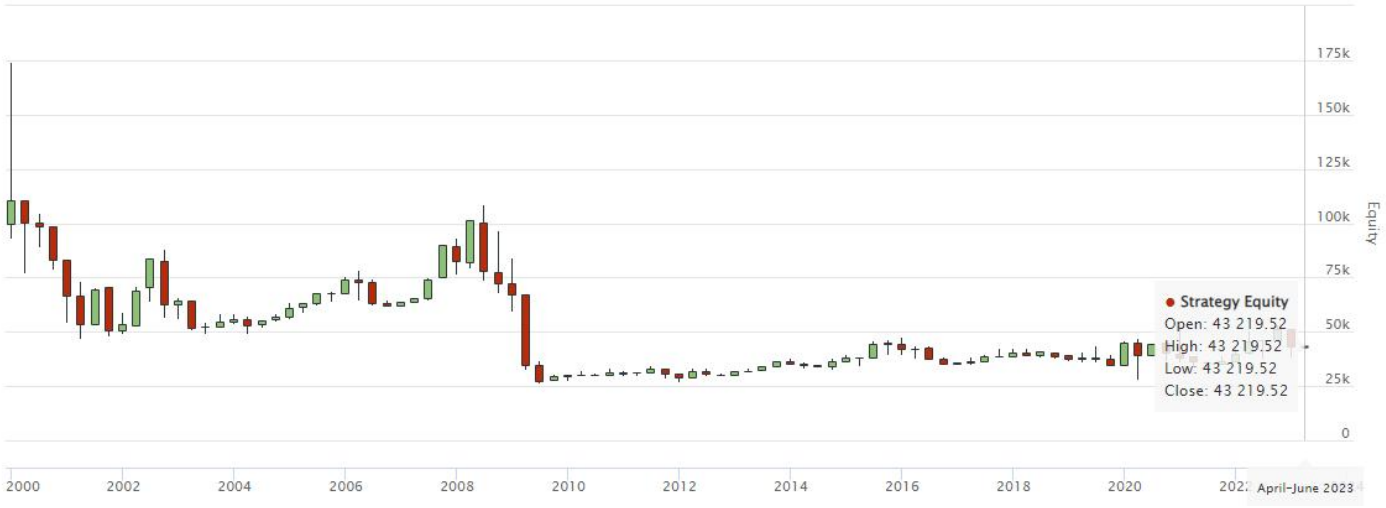

Fig 1. Overall Performance

Total Trades	60335	Average Win	0.09%
Average Loss	-0.11%	Compounding Annual Return	-3.541%
Drawdown	84.800%	Expectancy	-0.026
Net Profit	-56.780%	Sharpe Ratio	-0.003
Probabilistic Sharpe Ratio	0.000%	Loss Rate	46%
Win Rate	54%	Profit-Loss Ratio	0.82
Alpha	0.02	Beta	-0.358
Annual Standard Deviation	0.219	Annual Variance	0.048
Information Ratio	-0.192	Tracking Error	0.305
Treynor Ratio	0.002	Total Fees	$974.04
Estimated Strategy Capacity	$42000000.00	Lowest Capacity Asset	PWSC XQIXVMC7JPT1
Portfolio Turnover	4.64%		

Fig 2. Performance Metrics