

# Not Over Thinking



Short-term Reversal Effect in Stocks

Algorithmic Trading Strategy with Full Code

Haixiang

2024.02 | Vol 72.

[hxyan.2015@gmail.com](mailto:hxyan.2015@gmail.com) | [github.com/hxyan2020](https://github.com/hxyan2020)

## STRATEGY & ECONOMIC RATIONALE

The investment universe consists of the 100 biggest companies by market capitalization. The investor goes long on the ten stocks with the lowest performance in the previous week and goes short on the ten stocks with the greatest performance of the prior month. The portfolio is rebalanced weekly.

BUY	SELL
goes long on the ten stocks with the lowest performance in the previous week	goes short on the ten stocks with the greatest performance of the prior month

## PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equity
FINANCIAL INSTRUMENTS	Stocks
REGION	Global
PERIOD OF REBALANCING	Weekly
NO. OF TRADED INSTRUMENTS	20
WEIGHTING	Equal weighting
LOOKBACK PERIODS	1 month
LONG/SHORT	Long & short

## ALGORITHM

```
from AlgorithmImports import *from pandas.core.frame import DataFramefrom typing import List, Dict#endregion
class ShortTermReversalEffectinStocks(QCAlgorithm):

    def Initialize(self) -> None:
        self.SetStartDate(2000, 1, 1)
        self.SetCash(100000)

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

        self.coarse_count:int = 500
        self.stock_selection:int = 10
        self.top_by_market_cap_count:int = 100
        self.leverage:int = 5

        self.period:int = 21

        self.long:List[Symbol] = []
        self.short:List[Symbol] = []

        # daily close data
        self.data:Dict[Symbol, SymbolData] = {}

        self.day:int = 1
        self.selection_flag:bool = False
        self.UniverseSettings.Resolution = Resolution.Daily
        self.AddUniverse(self.CoarseSelectionFunction, self.FineSelectionFunction)
```

Not Over Thinking – where I share my journey to algorithmic trading and investments in shortest words possible

```
self.Schedule.On(self.DateRules.EveryDay(self.symbol), self.TimeRules.AfterMarketOpen(self.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].update(stock.AdjustedPrice)

    if not self.selection_flag:
        return Universe.Unchanged

    selected:List[CoarseFundamental] = sorted([x for x in coarse if x.HasFundamentalData and x.Market == 'usa' and x.Price > 1],
        key=lambda x: x.DollarVolume, reverse=True)
    selected:List[Symbol] = [x.Symbol for x in selected][:self.coarse_count]

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

        self.data[symbol] = SymbolData(self.period+1)
        history:DataFrame = self.History(symbol, self.period+1, Resolution.Daily)
        if history.empty:
            self.Log(f"Not enough data for {symbol} yet")
            continue
        closes:pd.Series = history.loc[symbol]
        for time, row in closes.iterrows():
            self.data[symbol].update(row['close'])

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

def FineSelectionFunction(self, fine:List[FineFundamental]) -> List[Symbol]:
    fine:List[FineFundamental] = [x for x in fine if x.MarketCap != 0]

    sorted_by_market_cap:List = sorted(fine, key = lambda x:x.MarketCap, reverse = True)
    top_by_market_cap:List[Symbol] = [x.Symbol for x in sorted_by_market_cap[:self.top_by_market_cap_count]]

    month_performances:Dict[Symbol, float] = {symbol : self.data[symbol].performance(self.period) for symbol in top_by_market_cap}
    week_performances:Dict[Symbol, float] = {symbol : self.data[symbol].performance(5) for symbol in top_by_market_cap}

    sorted_by_month_perf:List[Symbol] = [x[0] for x in sorted(month_performances.items(), key=lambda item: item[1], reverse=True)]
    sorted_by_week_perf:List[Symbol] = [x[0] for x in sorted(week_performances.items(), key=lambda item: item[1])]

    self.long = sorted_by_week_perf[:self.stock_selection]
    self.short = sorted_by_month_perf[:self.stock_selection]

    return self.long + self.short

def OnData(self, data:Slice) -> None:
```

```
if not self.selection_flag:
    return
self.selection_flag = False

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

# leveraged portfolio - 100% long, 100% short
for symbol in self.long:
    if symbol in data and data[symbol]:
        self.SetHoldings(symbol, 1 / len(self.long))

for symbol in self.short:
    if symbol in data and data[symbol]:
        self.SetHoldings(symbol, -1 / len(self.short))

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

def Selection(self) -> None:
    if self.day == 5:
        self.selection_flag = True

    self.day += 1
    if self.day > 5:
        self.day = 1
        class SymbolData():
def __init__(self, period:float) -> None:
    self._daily_close = RollingWindow[float](period)

def update(self, close:float) -> None:
    self._daily_close.Add(close)

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

def performance(self, period:int) -> float:
    return self._daily_close[0] / self._daily_close[period] - 1
# Custom fee modelclass 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	37718	Average Win	0.37%
Average Loss	-0.27%	Compounding Annual Return	19.237%
Drawdown	39.400%	Expectancy	0.082
Net Profit	5900.750%	Sharpe Ratio	0.722
Probabilistic Sharpe Ratio	2.544%	Loss Rate	55%
Win Rate	45%	Profit-Loss Ratio	1.38
Alpha	0.139	Beta	0.286
Annual Standard Deviation	0.215	Annual Variance	0.046
Information Ratio	0.407	Tracking Error	0.24
Treynor Ratio	0.543	Total Fees	\$713534.27
Estimated Strategy Capacity	\$28000000.00	Lowest Capacity Asset	TD R735QTJ8XC9X
Portfolio Turnover	39.10%		

Fig 2. Performance Metrics