

# Not Over Thinking

## ESG Factor Momentum Strategy

Algorithmic Trading Strategy with Full Code

Haixiang

2024.01 | Vol 61.

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

## STRATEGY & ECONOMIC RATIONALE

The investment universe consists of stocks in the MSCI World Index. Paper uses MSCI ESG Ratings as the ESG database. The ESG Momentum strategy is built by overweighting, relative to the MSCI World Index, companies that increased their ESG ratings most during the recent past and underweight those with decreased ESG ratings, where the increases and decreases are based on a 12-month ESG momentum. The paper uses the Barra Global Equity Model (GEM3) for portfolio construction with constraints that can be found in Appendix 2. Therefore, this strategy is very specific, but we aim to present the idea, not the portfolio construction. The strategy is rebalanced monthly.

BUY	SELL
(see above)	(see above)

## PARAMETER & VARIABLES

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

## ALGORITHM

```

from AlgorithmImports import *
from numpy import floor
#endregion

class ESGFactorMomentumStrategy(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2009, 6, 1)
        self.SetCash(100000)

        # Decile weighting.
        # True - Value weighted
        # False - Equally weighted
        self.value_weighting = True

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

        self.esg_data = self.AddData(ESGData, 'ESG', Resolution.Daily)
        self.tickers = []

        self.holding_period = 3
        self.managed_queue = []
        self.quantile = 10

        # Monthly ESG decile data.
        self.esg = {}

```

```

self.period = 14

self.latest_price = {}

self.selection_flag = False
self.UniverseSettings.Resolution = Resolution.Daily
self.AddUniverse(self.CoarseSelectionFunction, self.FineSelectionFunction)

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

def CoarseSelectionFunction(self, coarse):
    if not self.selection_flag:
        return Universe.Unchanged

    self.latest_price.clear()

    selected = [x for x in coarse if (x.Symbol.Value).lower() in self.tickers]

    for stock in selected:
        symbol = stock.Symbol
        self.latest_price[symbol] = stock.AdjustedPrice

    return [x.Symbol for x in selected]

def FineSelectionFunction(self, fine):
    fine = [x for x in fine if x.MarketCap != 0]

    momentum = {}

    # Momentum calc.
    for stock in fine:
        symbol = stock.Symbol
        ticker = symbol.Value
        # ESG data for 14 months is ready.
        if ticker in self.esg and self.esg[ticker].IsReady:
            esg_data = [x for x in self.esg[ticker]]

            esg_decile_2_months_ago = esg_data[1]
            esg_decile_14_months_ago = esg_data[13]

            if esg_decile_14_months_ago != 0 and esg_decile_2_months_ago != 0:
                # Momentum as difference.
                # momentum_ = esg_decile_2_months_ago - esg_decile_14_months_ago

                # Momentum as ratio.
                momentum_ = (esg_decile_2_months_ago / esg_decile_14_months_ago) - 1

                # Store momentum/market cap pair.
                momentum[stock] = momentum_

    if len(momentum) <= self.quantile:
        return Universe.Unchanged

    # Momentum sorting.
    sorted_by_momentum = sorted(momentum.items(), key = lambda x: x[1], reverse = True)
    quantile = int(len(sorted_by_momentum) / self.quantile)
    long = [x[0] for x in sorted_by_momentum[:quantile]]
    short = [x[0] for x in sorted_by_momentum[-quantile:]]

    long_symbol_q = []
    short_symbol_q = []

```

```

# ew
if not self.value_weighting:
    if len(long) != 0:
        long_w = self.Portfolio.TotalPortfolioValue / self.holding_period / len(long)
        long_symbol_q = [(x.Symbol, floor(long_w / self.latest_price[x.Symbol])) for x
in long]

        if len(short) != 0:
            short_w = self.Portfolio.TotalPortfolioValue / self.holding_period / len(short)
            short_symbol_q = [(x.Symbol, -floor(short_w / self.latest_price[x.Symbol])) for
x in short]
        # vw
    else:
        if len(long) != 0:
            total_market_cap_long = sum([x.MarketCap for x in long])
            long_w = self.Portfolio.TotalPortfolioValue / self.holding_period
            long_symbol_q = [(x.Symbol, floor((long_w * (x.MarketCap / total_market_cap_long))) / self.latest_price[x.Symbol]) for x in long]

            short_symbol_q = []
            if len(short) != 0:
                total_market_cap_short = sum([x.MarketCap for x in short])
                short_w = self.Portfolio.TotalPortfolioValue / self.holding_period
                short_symbol_q = [(x.Symbol, -floor((short_w * (x.MarketCap / total_market_cap_short))) / self.latest_price[x.Symbol]) for x in short]

            self.managed_queue.append(RebalanceQueueItem(long_symbol_q + short_symbol_q))

        return [x.Symbol for x in long + short]

def OnData(self, data):
    new_data_arrived = False

    if 'ESG' in data and data['ESG']:
        # Store universe tickers.
        if len(self.tickers) == 0:
            # TODO '_typename' in storage dictionary?
            self.tickers = [x.Key for x in self.esg_data.GetLastData().GetStorageDictionary
()][:-1]

        # Store history for every ticker.
        for ticker in self.tickers:
            ticker_u = ticker.upper()
            if ticker_u not in self.esg:
                self.esg[ticker_u] = RollingWindow[float](self.period)

            decile = self.esg_data.GetLastData()[ticker]
            self.esg[ticker_u].Add(decile)

            # trigger selection after new esg data arrived.
            if not self.selection_flag:
                new_data_arrived = True

    if new_data_arrived:
        self.selection_flag = True
        return

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

    # Trade execution
    remove_item = None

```

```
# Rebalance portfolio
for item in self.managed_queue:
    if item.holding_period == self.holding_period:
        for symbol, quantity in item.symbol_q:
            self.MarketOrder(symbol, -quantity)

        remove_item = item

    elif item.holding_period == 0:
        open_symbol_q = []

        for symbol, quantity in item.symbol_q:
            if quantity >= 1:
                if symbol in data and data[symbol]:
                    self.MarketOrder(symbol, quantity)
                    open_symbol_q.append((symbol, quantity))

        # Only opened orders will be closed
        item.symbol_q = open_symbol_q

    item.holding_period += 1

if remove_item:
    self.managed_queue.remove(remove_item)

class RebalanceQueueItem():
    def __init__(self, symbol_q):
        # symbol/quantity collections
        self.symbol_q = symbol_q
        self.holding_period = 0

# ESG data.
class ESGData(PythonData):
    def __init__(self):
        self.tickers = []

    def GetSource(self, config, date, isLiveMode):
        return SubscriptionDataSource("data.quantpedia.com/backtesting_data/economic/esg_decile
s_data.csv", SubscriptionTransportMedium.RemoteFile, FileFormat.Csv)

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

        if not line[0].isdigit():
            self.tickers = [x for x in line.split(';')][1:]
            return None

        split = line.split(';')

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

        index = 1
        for ticker in self.tickers:
            data[ticker] = float(split[index])
            index += 1

        data.Value = float(split[1])
        return data

# Custom fee model.
class CustomFeeModel(FeeModel):
    def GetOrderFee(self, parameters):
```

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

```

fee = parameters.Security.Price * parameters.Order.AbsoluteQuantity * 0.00005
return OrderFee(CashAmount(fee, "USD"))

```

## BACKTESTING PERFORMANCE



Fig 1. Overall Performance

Total Trades	14167	Average Win	0.06%
Average Loss	-0.04%	Compounding Annual Return	15.863%
Drawdown	26.100%	Expectancy	0.718
Net Profit	620.347%	Sharpe Ratio	0.843
Probabilistic Sharpe Ratio	18.964%	Loss Rate	36%
Win Rate	64%	Profit-Loss Ratio	1.67
Alpha	0.036	Beta	0.822
Annual Standard Deviation	0.14	Annual Variance	0.02
Information Ratio	0.233	Tracking Error	0.077
Treynor Ratio	0.143	Total Fees	\$1160.83
Estimated Strategy Capacity	\$150000000.00	Lowest Capacity Asset	UDL S3QTIGASA5WL

Fig 2. Performance Metrics