Not Over Thinking

Value and Momentum Factors across Asset Classes

Algorithmic Trading Strategy with Full Code

Haixiang
2023.08 | Vol 26.
hxyan.2015@gmail.com | github.com/hxyan2020

STRATEGY & ECONOMIC RATIONALE

Create an investment universe containing investable asset classes (could be US large-cap, mid-cap stocks, US REITS, UK, Japan, Emerging market stocks, US treasuries, US Investment grade bonds, US high yield bonds, Germany bonds, Japan bonds, US cash) and find a good tracking vehicle for each asset class (best vehicles are ETFs or index funds).

Momentum ranking is done on price series. Valuation ranking is done on adjusted yield measure for each asset class. E/P (Earning/Price) measure is used for stocks, and YTM (Yield-to-maturity) is used for bonds. US, Japan, and Germany treasury yield are adjusted by -1%, US investment-grade bonds are adjusted by -2%, US High yield bonds are adjusted by -6%, emerging markets equities are adjusted by -1%, and US REITs are adjusted by -2% to get unbiased structural yields for each asset class.

Rank each asset class by 12-month momentum, 1-month momentum, and by valuation and weight all three strategies (25% weight to 12m momentum, 25% weight to 1-month momentum, 50% weight to value strategy). Go long top quartile portfolio and go short bottom quartile portfolio.

BUY	SELL
top quartile portfolio	bottom quartile portfolio

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Bond, Equity, REITs
FINANCIAL INSTRUMENTS	ETFs, funds, futures
REGION	Global
PERIOD OF REBALANCING	Monthly
NO. OF TRADED INSTRUMENTS	6
WEIGHTING	Depends
LOOKBACK PERIODS	12 months
LONG/SHORT	Long and Short

ALGORITHM

<data_tools.py>

```python
#region imports
from AlgorithmImports import *
#endregion
# Bond yields
class QuandlAAAYield(PythonQuandl):
    def __init__(self):
        self.ValueColumnName = 'BAMLC0A1CAAAEY'

class QuandlHighYield(PythonQuandl):
    def __init__(self):
        self.ValueColumnName = 'BAMLH0A0HYM2EY'

# Quantpedia bond yield data.
# NOTE: IMPORTANT: Data order must be ascending (datewise)
```

```
class QuantpediaBondYield(PythonData):
    def GetSource(self, config, date, isLiveMode):
        return
SubscriptionDataSource("data.quantpedia.com/backtesting_data/bond_yield/{0}.csv".format(co
nfig.Symbol.Value), SubscriptionTransportMedium.RemoteFile, FileFormat.Csv)

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

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

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

        return data

# Country PE data
# NOTE: IMPORTANT: Data order must be ascending (date-wise)
from dateutil.relativedelta import relativedelta

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

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

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

        data.Time = datetime.strptime(split[0], "%Y") + relativedelta(years=1)
        self.symbols =
['Argentina','Australia','Austria','Belgium','Brazil','Canada','Chile','China','Egypt','Fr
ance','Germany','Hong
Kong','India','Indonesia','Ireland','Israel','Italy','Japan','Malaysia','Mexico','Netherla
nds','New Zealand','Norway','Philippines','Poland','Russia','Saudi
Arabia','Singapore','South Africa','South
Korea','Spain','Sweden','Switzerland','Taiwan','Thailand','Turkey','United
Kingdom','United States']
        index = 1
        for symbol in self.symbols:
            data[symbol] = float(split[index])
            index += 1

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

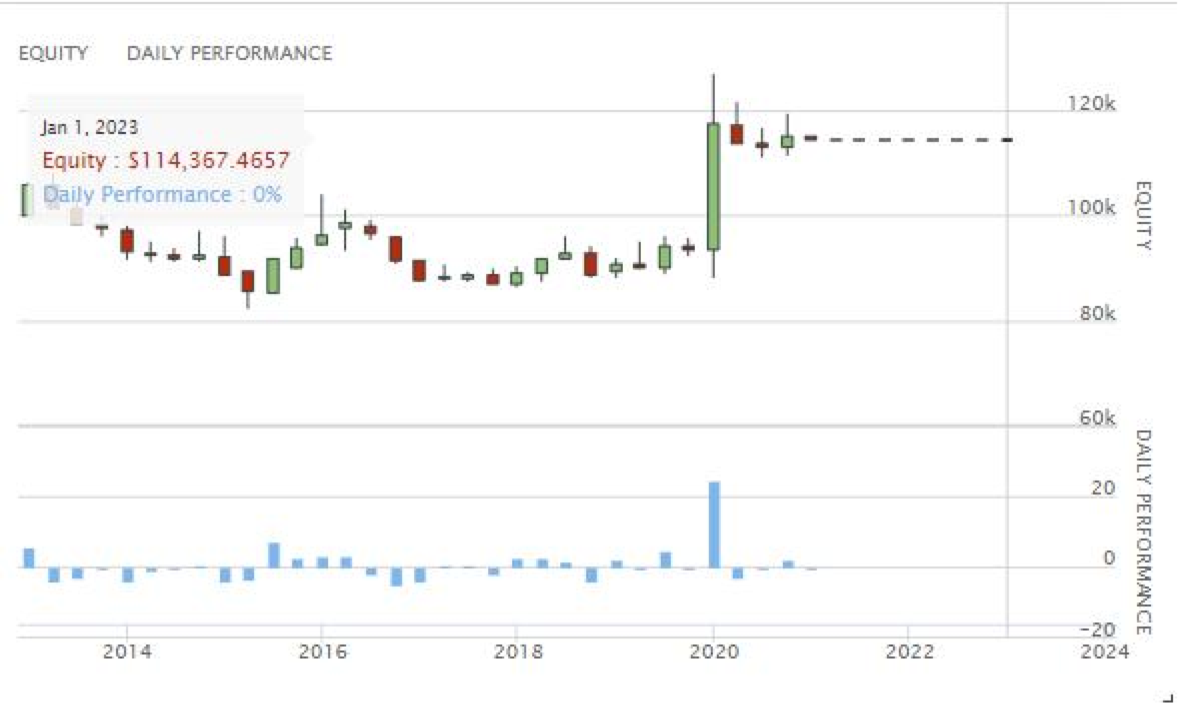
```
# Quandl "value" data
class QuandlValue(PythonQuandl):
    def __init__(self):
        self.ValueColumnName = 'Value'

# Quantpedia PE ratio data.
# NOTE: IMPORTANT: Data order must be ascending (datewise)
class QuantpediaPERatio(PythonData):
    def GetSource(self, config, date, isLiveMode):
        return
SubscriptionDataSource("data.quantpedia.com/backtesting_data/economic/{0}.csv".format(conf
ig.Symbol.Value), SubscriptionTransportMedium.RemoteFile, FileFormat.Csv)

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

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

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

        return data

# Quantpedia bond yield data.
# NOTE: IMPORTANT: Data order must be ascending (datewise)
class QuantpediaBondYield(PythonData):
    def GetSource(self, config, date, isLiveMode):
        return
SubscriptionDataSource("data.quantpedia.com/backtesting_data/bond_yield/{0}.csv".format(co
nfig.Symbol.Value), SubscriptionTransportMedium.RemoteFile, FileFormat.Csv)

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

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

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

        return data

# 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".format(confi
g.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['back_adjusted'] = float(split[1])
        data['spliced'] = float(split[2])
        data.Value = float(split[1])

        return data
```

<main.py>

```
from AlgorithmImports import *
import data_tools
#endregion

class ValueandMomentumFactorsacrossAssetClasses(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2013, 1, 1)
        self.SetCash(100000)

        # investable asset, yield symbol, yield data access function, yield adjustment, 
reverse flag(PE -> EP)
        self.assets = [
            ('SPY', 'MULTPL/SP500_EARNINGS_YIELD_MONTH', data_tools.QuandlValue, 0, 
True),  # US large-cap
            ('MDY', 'MID_CAP_PE', data_tools.QuantpediaPERatio, 0, 
True),                   # US mid-cap stocks
            ('IYR', 'REITS_DIVIDEND_YIELD', data_tools.QuantpediaPERatio, -2, 
False),       # US REITS - same csv data format as PERatio files
            ('EWU', 'United Kingdom', None, 0, 
True),                                       # UK
            ('EWJ', 'Japan', None, 0, 
True),                                                # Japan
            ('EEM', 'EMERGING_MARKET_PE', data_tools.QuantpediaPERatio, -1, 
True),          # Emerging market stocks
            
            ('LQD', 'ML/AAAEY', data_tools.QuandlAAAYield, -2, 
False),                      # US Investment grade bonds
            ('HYG', 'ML/USTRI', data_tools.QuandlHighYield, -6, 
False),                     # US high yield bonds
            
            ('CME_TY1', 'US10YT', data_tools.QuantpediaBondYield, -1, 
False),               # US bonds
```

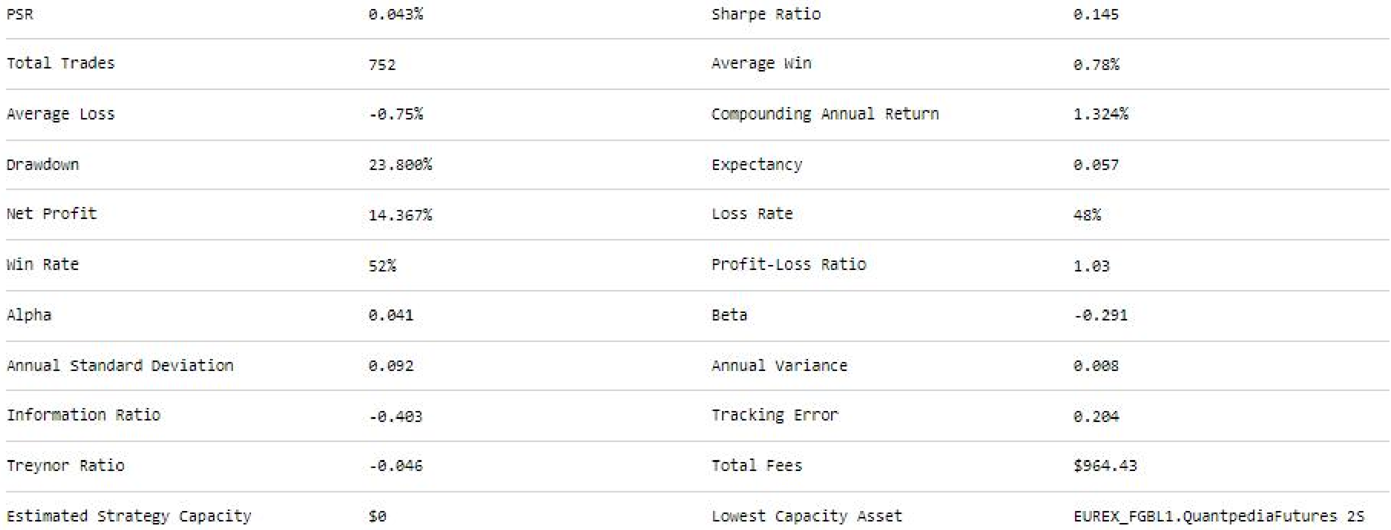
```
            ('EUREX_FGBL1', 'DE10YT', data_tools.QuantpediaBondYield, -1,
False),           # Germany bonds
            ('SGX_JB1', 'JP10YT', data_tools.QuantpediaBondYield, -1,
False),               # Japan bonds

            ('BIL', 'OECD/KEI_IRSTCI01_USA_ST_M', data_tools.QuandlValue, 0,
False)         # US cash
        ]

        # country pe data
        self.country_pe_data = self.AddData(data_tools.CountryPE, 'CountryData').Symbol

        self.data = {}
        self.period = 12 * 21
        self.SetWarmUp(self.period)

        for symbol, yield_symbol, yield_access, _, _ in self.assets:
            # investable asset
            if yield_access == data_tools.QuantpediaBondYield:
                data = self.AddData(data_tools.QuantpediaFutures, symbol, Resolution.Daily)
            else:
                data = self.AddEquity(symbol, Resolution.Daily)

            # yield
            if yield_access != None:
                self.AddData(yield_access, yield_symbol, Resolution.Daily)

            self.data[symbol] = RollingWindow[float](self.period)

            data.SetFeeModel(CustomFeeModel())
            data.SetLeverage(5)

        self.recent_month = -1

    def OnData(self, data):
        if self.IsWarmingUp:
            return

        # store investable asset price data
        for symbol, yield_symbol, _, _, _ in self.assets:
            symbol_obj = self.Symbol(symbol)
            if symbol_obj in data and data[symbol_obj]:
                self.data[symbol].Add(data[symbol_obj].Value)

        if self.Time.month == self.recent_month:
            return
        self.recent_month = self.Time.month

        performance_1M = {}
        performance_12M = {}
        valuation  = {}

        # performance and valuation calculation
        if self.Securities[self.country_pe_data].GetLastData() and (self.Time.date() - 
self.Securities[self.country_pe_data].GetLastData().Time.date()).days <= 365:
            for symbol, yield_symbol, yield_access, bond_adjustment, reverse_flag in 
self.assets:
                if self.Securities[symbol].GetLastData() and (self.Time.date() - 
self.Securities[symbol].GetLastData().Time.date()).days < 3:
                    if self.data[symbol].IsReady:
                        closes = [x for x in self.data[symbol]]
                        performance_1M[symbol] = closes[0] / closes[21] - 1
                        performance_12M[symbol] = closes[0] / closes[len(closes) - 1] - 1

                        if yield_access == None:
                            country_pb_data = self.Securities['CountryData'].GetLastData()
                            if country_pb_data:
                                pe = country_pb_data[yield_symbol]
                                yield_value = pe
                        else:
                            yield_value = self.Securities[self.Symbol(yield_symbol)].Price

                        # reverse if needed, EP->PE
                        if reverse_flag:
                            yield_value = 1/yield_value

                        if yield_value != 0:
                            valuation[symbol] = yield_value + bond_adjustment

        long = []
        short = []

        if len(valuation) != 0:
            # sort assets by metrics
            sorted_by_p1 = sorted(performance_1M.items(), key = lambda x: x[1])
            sorted_by_p12 = sorted(performance_12M.items(), key = lambda x: x[1])
            sorted_by_value = sorted(valuation.items(), key = lambda x: x[1])

            # rank assets
            score = {}
            for i, (symbol, _) in enumerate(sorted_by_p1):
                score[symbol] = i * 0.25
            for i, (symbol, _) in enumerate(sorted_by_p12):
                score[symbol] += i * 0.25
            for i, (symbol, _) in enumerate(sorted_by_value):
                score[symbol] += i * 0.5

            # sort by rank
            sorted_by_rank = sorted(score, key = lambda x: score[x], reverse = True)
            quartile = int(len(sorted_by_rank) / 4)
            long = sorted_by_rank[:quartile]
            short = sorted_by_rank[-quartile:]

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

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

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

# 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



Jan 1, 2023
Equity : $114,367.4657
Daily Performance : 0%

Fig 1. Overall Performance



PSR	0.043%	Sharpe Ratio	0.145
Total Trades	752	Average Win	0.78%
Average Loss	-0.75%	Compounding Annual Return	1.324%
Drawdown	23.800%	Expectancy	0.057
Net Profit	14.367%	Loss Rate	48%
Win Rate	52%	Profit-Loss Ratio	1.03
Alpha	0.041	Beta	-0.291
Annual Standard Deviation	0.092	Annual Variance	0.008
Information Ratio	-0.403	Tracking Error	0.204
Treynor Ratio	-0.046	Total Fees	$964.43
Estimated Strategy Capacity	$0	Lowest Capacity Asset	EUREX_FGBL1.QuantpediaFutures 2S

Fig 2. Performance Metrics

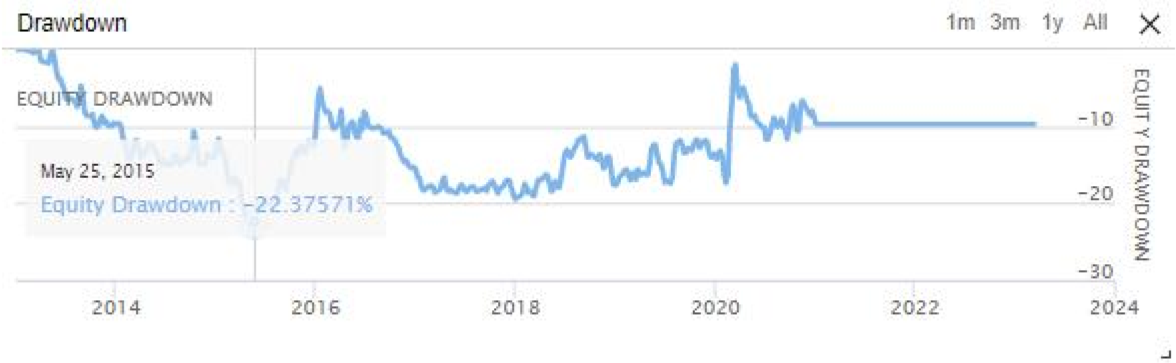


Fig 3. Drawdown

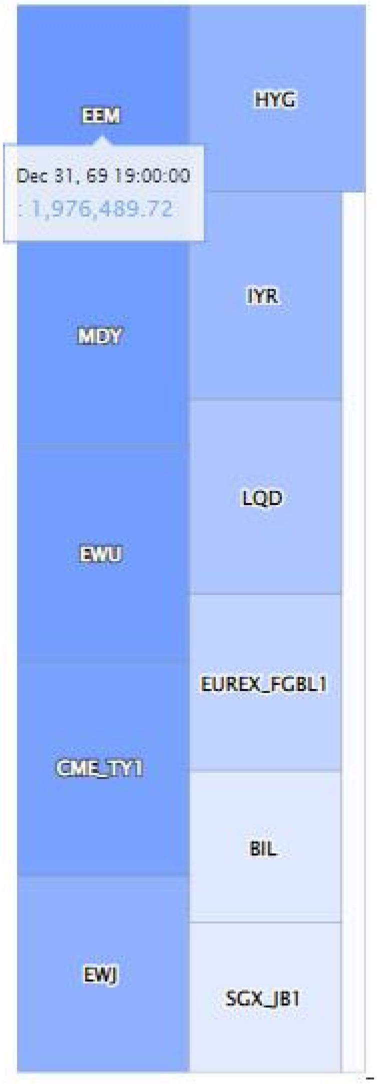


Fig 4. Assets Sales Volume