Not Over Thinking

FED Model

Algorithmic Trading Strategy with Full Code

Haixiang

2023.09 | Vol 32.

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

STRATEGY & ECONOMIC RATIONALE

Each month, the investor conducts a one-month predictive regression (using all available data up to that date) predicting excess stock market returns using the yield gap as an independent variable. The “Yield gap” is calculated as YG = EY – y, with earnings yield EY ≡ ln (1 ++ E/P) and y = ln (1 ++ Y) is the log 10 year Treasury bond yield. Then, the strategy allocates 100% in the risky asset if the forecasted excess returns are positive, and otherwise, it invests 100% in the risk-free rate.

BUY	SELL
allocates 100% in the risky asset if the forecasted excess returns are positive	

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Bond, Equity
FINANCIAL INSTRUMENTS	ETFs, funds, futures
REGION	Global
PERIOD OF REBALANCING	Monthly
NO. OF TRADED INSTRUMENTS	2
WEIGHTING	Equal weighting
LOOKBACK PERIODS	Monthly
LONG/SHORT	Long only

ALGORITHM

```python
from collections import deque
from AlgorithmImports import *
import numpy as np
from scipy import stats

class FEDModel(QCAlgorithm):

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

        # monthly price data and yield gap data
        self.data = {}

        self.period = 12 * 21
        self.SetWarmUp(self.period)

        self.market = self.AddEquity('SPY', Resolution.Daily).Symbol
        self.market_data = deque()

        self.cash = self.AddEquity('SHY', Resolution.Daily).Symbol
        # risk free rate
        self.risk_free_rate = self.AddData(QuandlValue, 'FRED/DGS3MO',
Resolution.Daily).Symbol

        # 10Y bond yield symbol
        self.bond_yield = self.AddData(QuantpediaBondYield, 'US10YT',
Resolution.Daily).Symbol

        # SP500 earnings yield data
        self.sp_earnings_yield = self.AddData(QuandlValue,
'MULTPL/SP500_EARNINGS_YIELD_MONTH', Resolution.Daily).Symbol

        self.yield_gap = deque()

        self.recent_month = -1

    def OnData(self, data):
        rebalance_flag = False

        if self.sp_earnings_yield in data and data[self.sp_earnings_yield]:
            if self.Time.month != self.recent_month:
                self.recent_month = self.Time.month
                rebalance_flag = True

        if not rebalance_flag:
            # earnings yield data is no longer comming in
            if self.Securities[self.sp_earnings_yield].GetLastData():
                if (self.Time.date() -
self.Securities[self.sp_earnings_yield].GetLastData().Time.date()).days > 31:
                    self.Liquidate()
            return

        # pdate market price data
        if self.market in data and self.risk_free_rate in data and self.bond_yield in data:
            if data[self.market] and data[self.risk_free_rate] and data[self.bond_yield]:
                market_price = data[self.market].Value
                rf_rate = data[self.risk_free_rate].Value
                bond_yield = data[self.bond_yield].Value
                sp_ey = data[self.sp_earnings_yield].Value
                if market_price != 0 and rf_rate != 0 and bond_yield != 0 and sp_ey != 0:
                    self.market_data.append((market_price, rf_rate))

                    yield_gap = np.log(sp_ey) - np.log(bond_yield)
                    self.yield_gap.append(yield_gap)
                    rebalance_flag = True


        # ensure minimum data points to calculate regression
        min_count = 6
        if len(self.market_data) >= min_count:
            market_closes = np.array([x[0] for x in self.market_data])
            market_returns = (market_closes[1:] - market_closes[:-1]) / market_closes[:-1]
            rf_rates = np.array([x[1] for x in self.market_data][1:])
            excess_returns = market_returns - rf_rates

            yield_gaps = [x for x in self.yield_gap]

            # linear regression
            # Y = α + (β ∗ X)
            # intercept = alpha
            # slope = beta
            beta, alpha, r_value, p_value, std_err = stats.linregress(yield_gaps[1:-1], 
market_returns[1:])
            X = yield_gaps[-1]

            # predicted market return
            Y = alpha + (beta * X)

            # trade execution / rebalance
            if Y > 0:
                if self.Portfolio[self.cash].Invested:
                    self.Liquidate(self.cash)
                self.SetHoldings(self.market, 1)
            else:
                if self.Portfolio[self.market].Invested:
                    self.Liquidate(self.market)
                self.SetHoldings(self.cash, 1)

# 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

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

BACKTESTING PERFORMANCE

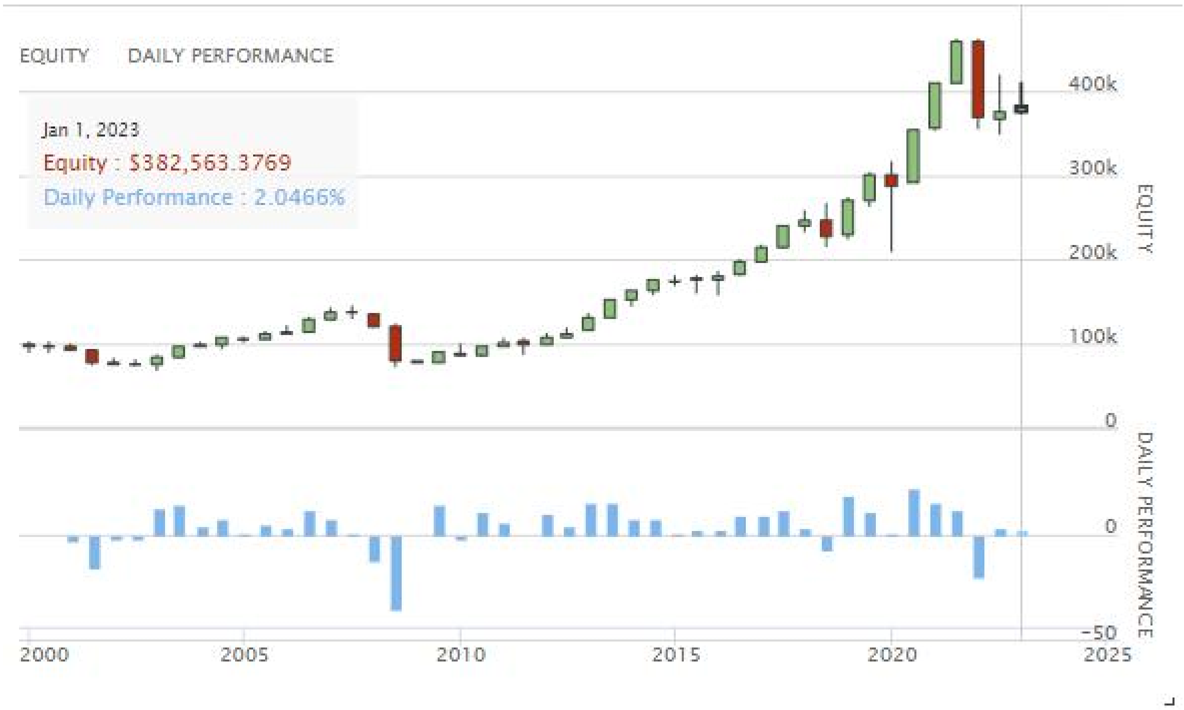
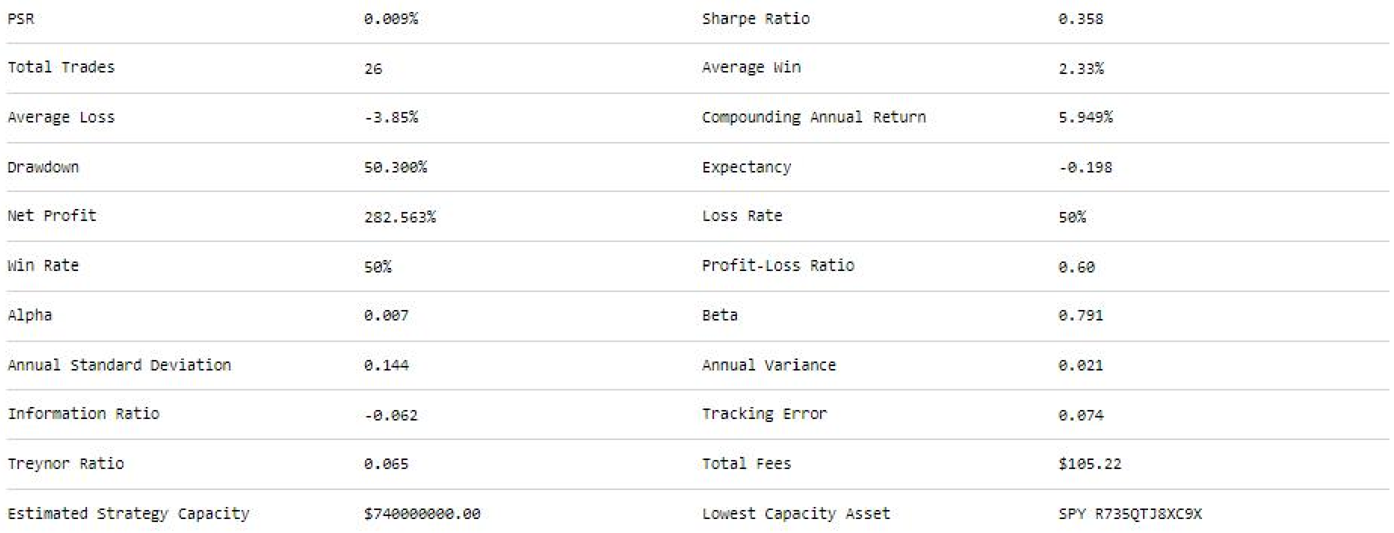


Fig 1. Overall Performance



PSR	0.009%	Sharpe Ratio	0.358
Total Trades	26	Average Win	2.33%
Average Loss	-3.85%	Compounding Annual Return	5.949%
Drawdown	50.300%	Expectancy	-0.198
Net Profit	282.563%	Loss Rate	50%
Win Rate	50%	Profit-Loss Ratio	0.60
Alpha	0.007	Beta	0.791
Annual Standard Deviation	0.144	Annual Variance	0.021
Information Ratio	-0.062	Tracking Error	0.074
Treynor Ratio	0.065	Total Fees	$105.22
Estimated Strategy Capacity	$740000000.00	Lowest Capacity Asset	SPY R735QTJ8XC9X

Fig 2. Performance Metrics

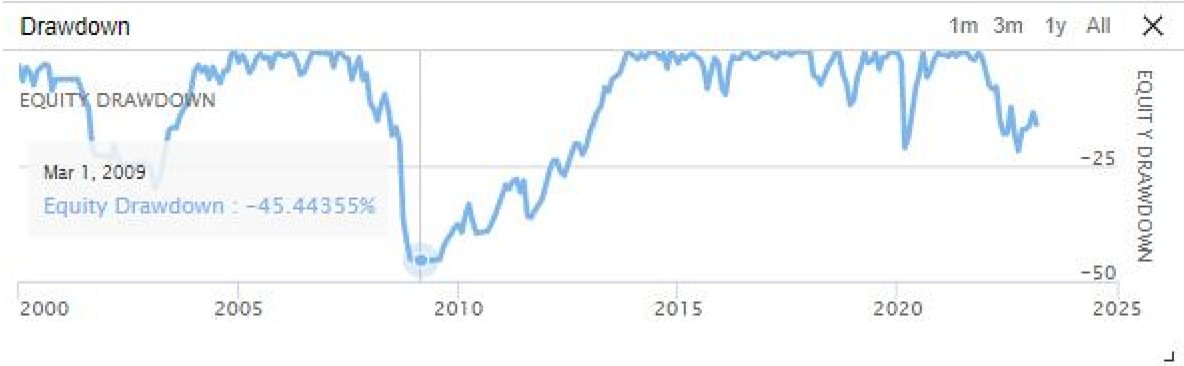


Fig 3. Drawdown