Not Over Thinking

Ramadan Effect

Algorithmic Trading Strategy with Full Code

Haixiang

2023.11 | Vol 49.

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

STRATEGY & ECONOMIC RATIONALE

The investment universe consists of countries for which stock market index data are available a nd in which the proportion of the population professing Muslim faith exceeded 50%. Most of the countries could be easily tracked via index ETFs. The research paper we use as an example uses 14 Muslim countries.

Ramadan is the ninth month in the Islamic calendar, which is based on the motion of the moon. T he Ramadan month could be calculated by using the information on the lunar phases and sunset ti mes from the astronomical calendar or information about Ramadan dates from various public sourc es.

The trading strategy is simple. The investor holds an equally weighted portfolio of ETFs during the Ramadan month. He/she is otherwise invested in cash.

BUY	SELL
holds an equally weighted po rtfolio of ETFs during the R amadan month.	The opposite

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equity
FINANCIAL INSTRUMENTS	ETFs, funds
REGION	Emerging Markets
PERIOD OF REBALANCING	Monthly
NO. OF TRADED INSTRUMENTS	14
WEIGHTING	Equal weighting
LOOKBACK PERIODS	Month
LONG/SHORT	Long only

ALGORITHM

```
from AlgorithmImports import *

class RamadanEffect(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2015, 1, 1)
        self.SetCash(100000)

        self.symbols = ['TUR', 'GULF', 'GAF', 'PAK', 'UAE', 'QAT', 'EGPT', 'EWM', 'EIDO',
'KSA']
        for symbol in self.symbols:
            self.AddEquity(symbol, Resolution.Daily)

        csv_string_file =
self.Download('data.quantpedia.com/backtesting_data/economic/ramadan_dates.csv')
        date_pairs_str = csv_string_file.split('\r\n')
        date_pairs = []
```

```
        for pair in date_pairs_str:
            split = pair.split(';')
            date_pairs.append([datetime.strptime(split[0], "%d.%m.%Y"),
datetime.strptime(split[1], "%d.%m.%Y")])

        self.start_dates = [pair[0].date() for pair in date_pairs]
        self.end_dates = [pair[1].date() for pair in date_pairs]

    def OnData(self, data):
        # open trades
        if self.Time.date() in self.start_dates or self.Time.date()-timedelta(days=1) in
self.start_dates:
            if not self.Portfolio.Invested:
                long = []
                for symbol in self.symbols:
                    if self.Securities[symbol].GetLastData() and (self.Time.date() -
self.Securities[symbol].GetLastData().Time.date()).days < 5:
                        long.append(symbol)

                count = len(long)
                for symbol in long:
                    self.SetHoldings(symbol, 1 / count)

        # close trades
        if self.Time.date() in self.end_dates or self.Time.date()-timedelta(days=1) in
self.end_dates:
            self.Liquidate()
```

BACKTESTING PERFORMANCE

Fig 1. Overall Performance

PSR	0.101%	Sharpe Ratio	0.1
Total Trades	152	Average Win	0.41%
Average Loss	-0.44%	Compounding Annual Return	0.406%
Drawdown	12.500%	Expectancy	0.010
Net Profit	3.406%	Loss Rate	47%
Win Rate	53%	Profit-Loss Ratio	0.91
Alpha	0	Beta	0.036
Annual Standard Deviation	0.034	Annual Variance	0.001
Information Ratio	-0.539	Tracking Error	0.152
Treynor Ratio	0.094	Total Fees	$391.60
Estimated Strategy Capacity	$19000.00	Lowest Capacity Asset	QAT VQ6KGBSR66AT
Portfolio Turnover	0.56%		

Fig 2. Performance Metrics

Fig 3. Drawdown

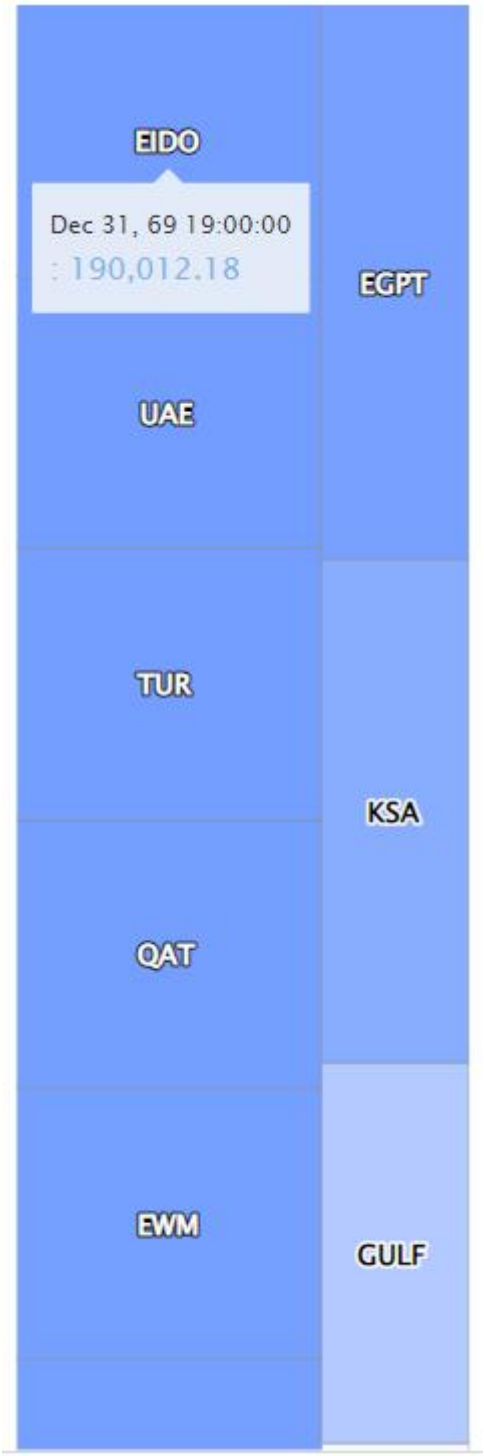

Fig 4. Assets Sales Volume