Not Over Thinking

Value Factor – CAPE Effect with Countries

Algorithmic Trading Strategy with Full Code

Haixiang

2023.11 | Vol 52.

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

STRATEGY & ECONOMIC RATIONALE

The investment universe consists of 32 countries with easily accessible equity markets (via ETFs, for example). At the end of every year, the investor calculates Shiller's "CAPE" Cyclically Adjusted PE) ratio, for each country in his investment universe. CAPE is the ratio of the real price of the equity market (adjusted for inflation) to the 10-year average of the country's equity index (again adjusted for inflation). The whole methodology is explained well on Shiller's home page (http://www.econ.yale.edu/~shiller/data.htm) or http://turnkeyanalyst.com/2011/10/the-shiller-pe-ratio/). The investor then invests in the cheapest 33% of countries from his sample if those countries have a CAPE below 15. The portfolio is equally weighted (the investor holds 0% cash instead of countries with a CAPE higher than 15) and rebalanced yearly.

BUY	SELL
the cheapest 33% of countries from his sample if those countries have a CAPE below 15	The opposite

PARAMETER & VARIABLES

PARAMETER	VALUE
MARKETS TRADED	Equity
FINANCIAL INSTRUMENTS	ETFs
REGION	Global
PERIOD OF REBALANCING	Yearly
NO. OF TRADED INSTRUMENTS	10
WEIGHTING	Equal weighting
LOOKBACK PERIODS	N/A
LONG/SHORT	Long Only

ALGORITHM

```
class ValueFactorCAPEEffectwithinCountries(QCAlgorithm):

    def Initialize(self):
        self.SetStartDate(2008, 1, 1)
        self.SetCash(100000)

        self.symbols = {
            "Australia"     : "EWA",  # iShares MSCI Australia Index ETF
            "Brazil"        : "EWZ",  # iShares MSCI Brazil Index ETF
            "Canada"        : "EWC",  # iShares MSCI Canada Index ETF
            "Switzerland"   : "EWL",  # iShares MSCI Switzerland Index ETF
            "China"         : "FXI",  # iShares China Large-Cap ETF
            "France"        : "EWQ",  # iShares MSCI France Index ETF
            "Germany"       : "EWG",  # iShares MSCI Germany ETF
            "Hong Kong"     : "EWH",  # iShares MSCI Hong Kong Index ETF
            "Italy"         : "EWI",  # iShares MSCI Italy Index ETF
            "Japan"         : "EWJ",  # iShares MSCI Japan Index ETF
            "Korea"         : "EWY",  # iShares MSCI South Korea ETF
```

```
            "Mexico"        : "EWW",  # iShares MSCI Mexico Inv. Mt. Idx
            "Netherlands"   : "EWN",  # iShares MSCI Netherlands Index ETF
            "South Africa"  : "EZA",  # iShares MSCI South Africe Index ETF
            "Singapore"     : "EWS",  # iShares MSCI Singapore Index ETF
            "Spain"         : "EWP",  # iShares MSCI Spain Index ETF
            "Sweden"        : "EWD",  # iShares MSCI Sweden Index ETF
            "Taiwan"        : "EWT",  # iShares MSCI Taiwan Index ETF
            "UK"            : "EWU",  # iShares MSCI United Kingdom Index ETF
            "USA"           : "SPY",  # SPDR S&P 500 ETF

            "Russia"        : "ERUS",  # iShares MSCI Russia ETF
            "Israel"        : "EIS",   # iShares MSCI Israel ETF
            "India"         : "INDA",  # iShares MSCI India ETF
            "Poland"        : "EPOL",  # iShares MSCI Poland ETF
            "Turkey"        : "TUR"    # iShares MSCI Turkey ETF
        }

        self.quantile:int = 3
        self.max_missing_days:int = 31
        self.leverage:int = 2

        for country, etf_symbol in self.symbols.items():
            data = self.AddEquity(etf_symbol, Resolution.Daily)
            data.SetLeverage(self.leverage)
            data.SetFeeModel(CustomFeeModel())

        # CAPE data import.
        self.cape_data = self.AddData(CAPE, 'CAPE',  Resolution.Daily).Symbol

        self.recent_month:int = -1

    def OnData(self, data:Slice) -> None:
        if self.Time.month == self.recent_month:
            return
        self.recent_month = self.Time.month

        if self.recent_month != 12:
            return

        price = {}
        for country, etf_symbol in self.symbols.items():
            if etf_symbol in data and data[etf_symbol]:
                # cape data is still comming in
                if self.Securities[self.cape_data].GetLastData() and (self.Time.date() -
self.Securities[self.cape_data].GetLastData().Time.date()).days <= self.max_missing_days:
                    country_cape =
self.Securities['CAPE'].GetLastData().GetProperty(country)
                    if country_cape < 15:
                        price[etf_symbol] = data[etf_symbol].Value

        long = []

        # Cape and price sorting.
```

```
        if len(price) >= self.quantile:
            sorted_by_price = sorted(price.items(), key = lambda x: x[1], reverse = True)
            tercile = int(len(sorted_by_price) / self.quantile)
            long = [x[0] for x in sorted_by_price[-tercile:]]

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

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

# NOTE: IMPORTANT: Data order must be ascending (datewise)
# Data source: https://indices.barclays/IM/21/en/indices/static/historic-cape.app
class CAPE(PythonData):
    def GetSource(self, config, date, isLiveMode):
        return
SubscriptionDataSource("data.quantpedia.com/backtesting_data/economic/cape_by_country.csv",
SubscriptionTransportMedium.RemoteFile, FileFormat.Csv)

    def Reader(self, config, line, date, isLiveMode):
        data = CAPE()
        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['Australia'] = float(split[1])
        data['Brazil'] = float(split[2])
        data['Canada'] = float(split[3])
        data['Switzerland'] = float(split[4])
        data['China'] = float(split[5])
        data['France'] = float(split[6])
        data['Germany'] = float(split[7])
        data['Hong Kong'] = float(split[8])
        data['India'] = float(split[9])
        data['Israel'] = float(split[10])
        data['Italy'] = float(split[11])
        data['Japan'] = float(split[12])
        data['Korea'] = float(split[13])
        data['Mexico'] = float(split[14])
        data['Netherlands'] = float(split[15])
        data['Poland'] = float(split[16])
        data['Russia'] = float(split[17])
        data['South Africa'] = float(split[18])
        data['Singapore'] = float(split[19])
        data['Spain'] = float(split[20])
        data['Sweden'] = float(split[21])
```

```
        data['Taiwan'] = float(split[22])
        data['Turkey'] = float(split[23])
        data['UK'] = float(split[24])
        data['USA'] = float(split[25])

        data.Value = float(split[1])

        return data

# 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

Fig 1. Overall Performance

PSR	0.247%	Sharpe Ratio	0.416
Total Trades	43	Average Win	10.48%
Average Loss	-6.52%	Compounding Annual Return	8.418%
Drawdown	41.500%	Expectancy	1.058
Net Profit	243.643%	Loss Rate	21%
Win Rate	79%	Profit-Loss Ratio	1.61
Alpha	0.033	Beta	0.533
Annual Standard Deviation	0.179	Annual Variance	0.032
Information Ratio	-0.016	Tracking Error	0.173
Treynor Ratio	0.14	Total Fees	$178.27
Estimated Strategy Capacity	$420000.00	Lowest Capacity Asset	EPOL UMWMI2IDARTX
Portfolio Turnover	0.26%		

Fig 2. Performance Metrics

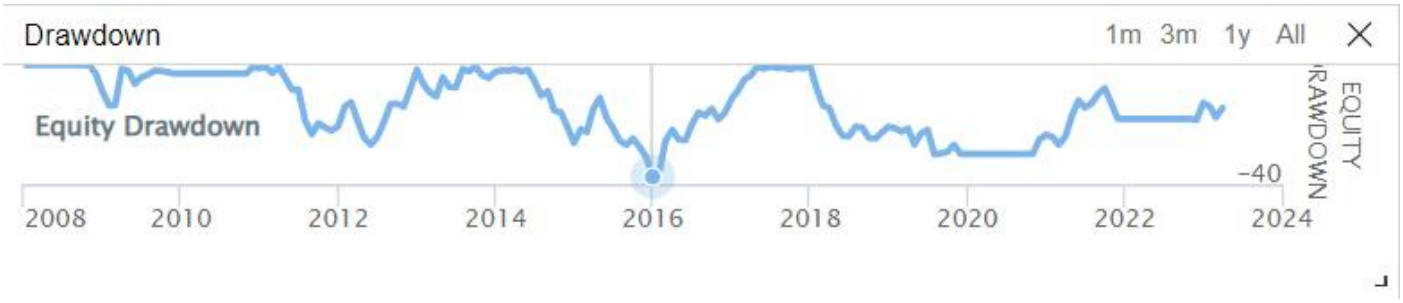

Fig 3. Drawdown

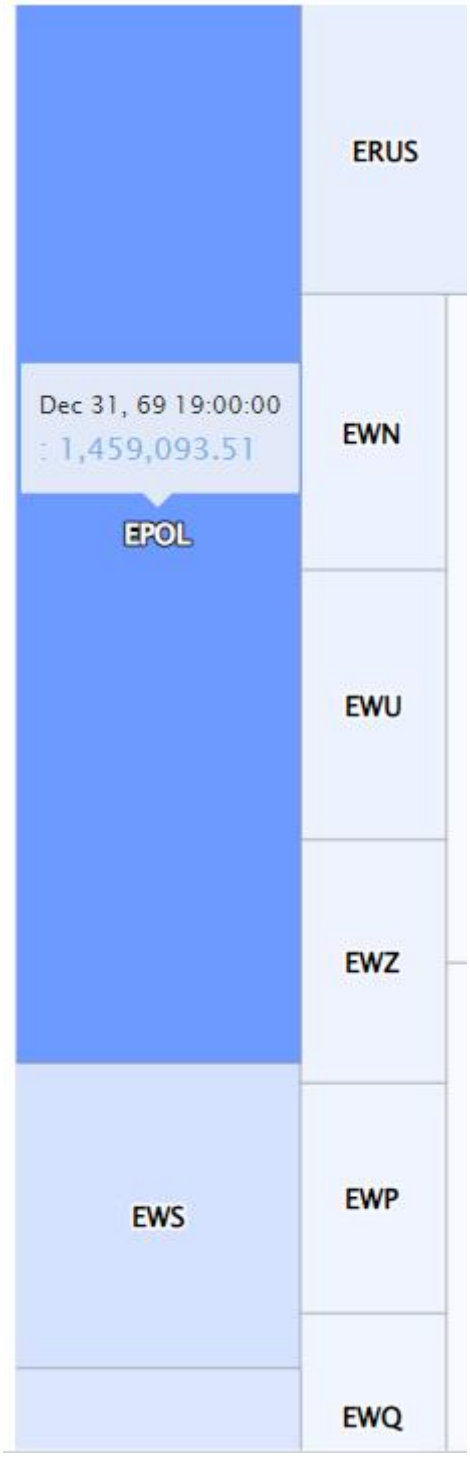

Fig 4. Assets Sales Volume