07_pairs_trading_backtest

September 29, 2021

1 Pair Trading: Backtest with Backtrader

1.1 Imports & Settings

[1]:

```
import warnings
warnings.filterwarnings('ignore')
```

[2]:

```
import csv
from collections import defaultdict
from dataclasses import dataclass, asdict
from datetime import date
from pathlib import Path
from time import time

import numpy as np
import pandas as pd
import pandas_datareader.data as web

import matplotlib.pyplot as plt
import seaborn as sns

import backtrader as bt
from backtrader.feeds import PandasData

import pyfolio as pf
```

[3]:

```
sns.set_style('dark')
pd.set_option('display.float_format', lambda x: f'{x:,.2f}')
idx = pd.IndexSlice
```

[4]:

```
STORE = 'backtest.h5'
```

[5]:

```
def format_time(t):
    m_, s = divmod(t, 60)
    h, m = divmod(m_, 60)
    return f'{h:>02.0f}:{m:>02.0f}:{s:>02.0f}'
```

1.2 Pairs Trading Backtest

1.2.1 Pairs DataClass

[6]:

```python
@dataclass
class Pair:
    period: int
    s1: str
    s2: str
    size1: float
    size2: float
    long: bool
    hr: float
    p1: float
    p2: float
    pos1: float
    pos2: float
    exec1: bool = False
    exec2: bool = False
    active: bool = False
    entry_date: date = None
    exit_date: date = None
    entry_spread: float = np.nan
    exit_spread: float = np.nan

    def executed(self):
        return self.exec1 and self.exec2

    def get_constituent(self, name):
        if name == self.s1:
            return 1
        elif name == self.s2:
            return 2
        else:
            return 0

    def compute_spread(self, p1, p2):
        return p1 * self.size1 + p2 * self.size2

    def compute_spread_return(self, p1, p2):
        current_spread = self.compute_spread(p1, p2)
        delta = self.entry_spread - current_spread
        return (delta / (np.sign(self.entry_spread) *
                         self.entry_spread))
```

1.2.2 PandasData definition

[7]:
```
class CustomData(PandasData):
    """
    Define pandas DataFrame structure
    """
    cols = ['open', 'high', 'low', 'close', 'volume']

    # create lines
    lines = tuple(cols)

    # define parameters
    params = {c: -1 for c in cols}
    params.update({'datetime': None})
    params = tuple(params.items())
```

1.2.3 Define Trading Strategy

[8]:
```
class StatisticalArbitrageCointegration(bt.Strategy):
    params = (('trades', None),
              ('risk_limit', -.2),
              ('verbose', True),
              ('log_file', 'backtest.csv'))

    def __init__(self):
        self.active_pairs = {}
        self.closing_pairs = {}
        self.exposure = []
        self.metrics = []
        self.last_close = {}
        self.cnt = 0
        self.today = None
        self.clear_log()
        self.order_status = dict(enumerate(['Created', 'Submitted', 'Accepted',
                                            'Partial', 'Completed', 'Canceled',
                                            'Expired', 'Margin', 'Rejected']))

    def clear_log(self):
        if Path(self.p.log_file).exists():
            Path(self.p.log_file).unlink()
        with Path(self.p.log_file).open('a') as f:
            log_writer = csv.writer(f)
            log_writer.writerow(
                    ['Date', 'Pair', 'Symbol', 'Order #', 'Reason',
                     'Status', 'Long', 'Price', 'Size', 'Position'])

    def log(self, txt, dt=None):
        """ Logger for the strategy"""
        dt = dt or self.datas[0].datetime.datetime(0)
        with Path(self.p.log_file).open('a') as f:
            log_writer = csv.writer(f)
            log_writer.writerow([dt.date()] + txt.split(','))

    def get_pair_id(self, s1, s2, period):
        return f'{s1}.{s2}.{period}'

    def check_risk_limit(self):
        for pair_id, pair in list(self.active_pairs.items()):
            if pair.active:
                p1 = self.last_close.get(pair.s1)
                p2 = self.last_close.get(pair.s2)
                ret = pair.compute_spread_return(p1, p2)
                if ret < self.p.risk_limit:
                    self.log(f'{pair_id},{pair.s1},{pair.s2},Risk Limit,{ret},')
                    del self.active_pairs[pair_id]
                    self.sell_pair(pair_id, pair)

    def sell_pair(self, pair_id, pair, reason='close'):
        info = {'pair': pair_id, 'type': reason}
        if pair.long:
            o1 = self.sell(data=pair.s1, size=abs(pair.size1), info=info)
            o2 = self.buy(data=pair.s2, size=abs(pair.size2), info=info)
        else:
            o1 = self.buy(data=pair.s1, size=abs(pair.size1), info=info)
            o2 = self.sell(data=pair.s2, size=abs(pair.size2), info=info)
        pair.active = False
        pair.exec1 = pair.exec2 = False
        self.closing_pairs[pair_id] = pair

        self.log(f'{pair_id},{pair.s1},{o1.ref},{reason},Created,{pair.
↪long},,{pair.size1},')
        self.log(f'{pair_id},{pair.s2},{o2.ref},{reason},Created,{pair.
↪long},,{pair.size2},')

    def notify_order(self, order):
        symbol = order.data._name
        if order.status in [order.Submitted, order.Accepted]:
            return
        if order.status in [order.Completed]:
            p = order.executed.price
            s = order.executed.size
            order_type = order.info.info['type']
            if order_type in ['open', 'close']:
                pair_id = order.info.info['pair']
                if order_type == 'open':
                    pair = self.active_pairs.get(pair_id)
                else:
                    pair = self.closing_pairs.get(pair_id)
                if pair is None:
                    self.log(f'{pair_id},{symbol},{order.
↪ref},{order_type},Completed (missing),,{p},{s},{p * s}')
                    return
                component = pair.get_constituent(symbol)
                if component == 1:
                    pair.p1 = p
                    pair.exec1 = True
                elif component == 2:
                    pair.p2 = p
                    pair.exec2 = True
                if pair.executed():
                    pair.exec1 = False
                    pair.exec2 = False
                    if order_type == 'open':
                        pair.entry_spread = pair.compute_spread(p1=pair.p1,␣
↪p2=pair.p2)
                        pair.entry_date = self.today
                        pair.active = True
                    elif order_type == 'close':
                        pair.exit_spread = pair.compute_spread(p1=pair.p1,␣
↪p2=pair.p2)
                        pair.exit_date = self.today
                        pair.active = False
                        self.closing_pairs.pop(pair_id)
                self.log(f'{pair_id},{symbol},{order.
↪ref},{order_type},Completed,{pair.long},{p},{s},{p * s}')
            else:
                self.log(f',{symbol},{order.
↪ref},{order_type},Completed,,{p},{s},{p * s}')

        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            order_type = order.info.info['type']
            self.log(f',{symbol},{order.ref},{order_type},{self.
↪order_status[order.status]},,,,')

    def enter_pairs(self, df, long=True):
        for s1, s2, hr, period in zip(df.s1, df.s2, df.hedge_ratio, df.period):
            pair_id = self.get_pair_id(s1, s2, period)
            if self.active_pairs.get(pair_id):
                continue
            p1 = self.last_close[s1]
            p2 = self.last_close[s2]
            if long:
                size1 = self.target_value / p1
                size2 = hr * size1
            else:
                size2 = self.target_value / p2
                size1 = 1 / hr * size2

            pair = Pair(s1=s1, s2=s2, period=period, size1=size1, size2=size2,
                        pos1=p1 * size1, pos2=p2 * size2,
                        hr=hr, long=long, p1=p1, p2=p2, entry_date=self.today)
            info = {'pair': pair_id, 'type': 'open'}
            if long:
                o1 = self.buy(data=s1, size=size1, info=info)
                o2 = self.sell(data=s2, size=abs(size2), info=info)
            else:
                o1 = self.sell(data=pair.s1, size=abs(pair.size1), info=info)
                o2 = self.buy(data=pair.s2, size=abs(pair.size2), info=info)

            self.active_pairs[pair_id] = pair

            self.log(f'{pair_id},{s1},{o1.
↪ref},Open,Created,{long},{p1},{size1},{pair.pos1}')
            self.log(f'{pair_id},{s2},{o2.
↪ref},Open,Created,{long},{p2},{size2},{pair.pos2}')

    def adjust_pairs(self):
        orders = defaultdict(float)
        pairs = defaultdict(list)
        for pair_id, pair in self.active_pairs.items():
            p1, p2 = self.last_close[pair.s1], self.last_close[pair.s2]
            pos1, pos2 = pair.size1 * p1, pair.size2 * p2

            if pair.long:
                target_size1 = self.target_value / p1
                orders[pair.s1] += target_size1 - pair.size1
                target_size2 = pos2 / pos1 * self.target_value / p2
                orders[pair.s2] += target_size2 - pair.size2
            else:
                target_size2 = self.target_value / p2
                orders[pair.s2] += target_size2 - pair.size2
                target_size1 = pos1 / pos2 * self.target_value / p1
                orders[pair.s1] += target_size1 - pair.size1
            pair.size1 = target_size1
            pair.size2 = target_size2
            pairs[pair.s1].append(pair_id)
            pairs[pair.s2].append(pair_id)

        for symbol, size in orders.items():
            info = {'pairs': pairs[symbol], 'type': 'adjust'}
            if size > 0:
                order = self.buy(symbol, size=size, info=info)
            elif size < 0:
                order = self.sell(symbol, size=abs(size), info=info)
            else:
                continue
            self.log(f',{symbol},{order.ref},Adjust,Created,{size}')

    def prenext(self):
        self.next()

    def next(self):
        self.today = pd.Timestamp(self.datas[0].datetime.date())
        if self.today not in self.p.trades.index:
            return
        self.cnt += 1

        pf = self.broker.get_value()
        cash = self.broker.get_cash()

        exp = {d._name: pos.size for d, pos in self.getpositions().items() if␣
↪pos}
        self.last_close = {d._name: d.close[0] for d in self.datas}
        exposure = pd.DataFrame({'price'   : pd.Series(self.last_close),
                                 'position': pd.Series(exp)}).replace(0, np.
↪nan).dropna()
        exposure['value'] = exposure.price * exposure.position
        positions = exposure.value.to_dict()
        positions['date'] = self.today
        positions['cash'] = cash
        if not exposure.empty:
            self.exposure.append(positions)
            long_pos = exposure[exposure.value > 0].value.sum()
            short_pos = exposure[exposure.value < 0].value.sum()
            for symbol, row in exposure.iterrows():
                self.log(f',{symbol},,Positions,Log,,{row.price},{row.
↪position},{row.value}')
        else:
            long_pos = short_pos = 0

        trades = self.p.trades.loc[self.today]
        if isinstance(trades, pd.Series):
            trades = trades.to_frame().T
        close = trades[trades.side == 0].sort_values('period')
        for s1, s2, period in zip(close.s1, close.s2, close.period):
            pair_id = self.get_pair_id(s1, s2, period)
            pair = self.active_pairs.pop(pair_id, None)
            if pair is None:
                self.log(f'{pair_id},,,Close Attempt,Failed,,,,')
                continue
            self.sell_pair(pair_id, pair)

        if len(self.active_pairs) > 0:
            self.check_risk_limit()

        long = trades[trades.side == 1]
        short = trades[trades.side == -1]
        if long.empty and short.empty: return
        target = 1 / (len(long) + len(short) + len(self.active_pairs))
        self.target_value = pf * target
        metrics = [self.today, pf, pf - cash, cash, len(exposure), len(self.
↪active_pairs), long_pos, short_pos,
                   target, self.target_value, len(long), len(short), len(close)]
        self.metrics.append(metrics)
        if self.cnt % 21 == 0:
            holdings = pf - cash
            msg = f'PF: {pf:11,.0f} | Net: {holdings: 11,.0f} | # Pos:␣
↪{len(exposure):3,.0f} | # Pairs: {len(self.active_pairs):3,.0f} | '
            msg += f'Long: {long_pos: 10,.0f} | Short: {short_pos: 10,.0f}'
            print(self.today, msg)

        self.adjust_pairs()

        if not long.empty:
            self.enter_pairs(long, long=True)

        if not short.empty:
            self.enter_pairs(short, long=False)
```

1.2.4 Load Trades

[9]:
```
trades = pd.read_hdf(STORE, 'pair_trades').sort_index()
trades.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 134450 entries, 2017-01-03 to 2019-12-18
Data columns (total 6 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
```

```
 0   s1           134450 non-null  object 
 1   s2           134450 non-null  object 
 2   hedge_ratio  134450 non-null  float64
 3   period       134450 non-null  int64  
 4   pair         134450 non-null  int64  
 5   side         134450 non-null  int64  
dtypes: float64(1), int64(3), object(2)
memory usage: 7.2+ MB
```

[10]:

```
trade_dates = np.unique(trades.index)
start = trade_dates.min()
end = trade_dates.max()
traded_symbols = trades.s1.append(trades.s2).unique()
```

1.2.5 Load Prices

[11]:

```
prices = (pd.read_hdf(STORE, 'prices')
          .sort_index()
          .loc[idx[traded_symbols, str(start):str(end)], :])
```

[12]:

```
prices.info(show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 232003 entries, ('AA.US', Timestamp('2017-01-03 00:00:00')) to
('GS.US', Timestamp('2019-12-18 00:00:00'))
Data columns (total 5 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   open    232003 non-null  float64
 1   high    232003 non-null  float64
 2   low     232003 non-null  float64
 3   close   232003 non-null  float64
 4   volume  232003 non-null  int64  
dtypes: float64(4), int64(1)
memory usage: 10.2+ MB
```

1.2.6 Configure Cerebro

[13]:

```
cerebro = bt.Cerebro()
cash = 1000000
cerebro.broker.setcash(cash)
```

1.2.7 Add Data

[14]:

```
for symbol in traded_symbols:
    df = prices.loc[idx[symbol, :], :].droplevel('ticker', axis=0)
    df.index.name = 'datetime'
    bt_data = CustomData(dataname=df)
    cerebro.adddata(bt_data, name=symbol)
```

1.2.8 Add Strategy & Analyzer

[15]:

```
cerebro.addstrategy(StatisticalArbitrageCointegration,
                    trades=trades, verbose=True,
                    log_file='bt_log.csv')

cerebro.addanalyzer(bt.analyzers.PyFolio, _name='pyfolio')
```

1.2.9 Run Strategy

[16]:

```
start = time()
results = cerebro.run()

ending_value = cerebro.broker.getvalue()
duration = time() - start

print(f'Final Portfolio Value: {ending_value:,.2f} | Duration:␣
 ↪{format_time(duration)}')
```

```
2017-02-01 00:00:00 PF:   1,004,229 | Net:      33,384 | # Pos: 275 | # Pairs:
265 | Long:    780,496 | Short:   -747,113
2017-03-03 00:00:00 PF:   1,028,845 | Net:      55,858 | # Pos: 280 | # Pairs:
235 | Long:    779,003 | Short:   -723,145
2017-04-03 00:00:00 PF:   1,029,713 | Net:      64,817 | # Pos: 292 | # Pairs:
689 | Long:    813,349 | Short:   -748,532
2017-05-03 00:00:00 PF:   1,036,655 | Net:      73,245 | # Pos: 299 | # Pairs:
263 | Long:    907,078 | Short:   -833,833
2017-06-02 00:00:00 PF:   1,014,250 | Net:      80,586 | # Pos: 282 | # Pairs:
218 | Long:    844,852 | Short:   -764,266
2017-07-03 00:00:00 PF:     997,286 | Net:      65,692 | # Pos: 284 | # Pairs:
161 | Long:    797,628 | Short:   -731,936
2017-08-02 00:00:00 PF:   1,031,169 | Net:      41,116 | # Pos: 307 | # Pairs:
445 | Long:    719,039 | Short:   -677,923
2017-08-31 00:00:00 PF:   1,048,305 | Net:      12,173 | # Pos: 297 | # Pairs:
123 | Long:    851,316 | Short:   -839,143
2017-10-02 00:00:00 PF:   1,071,228 | Net:      57,038 | # Pos: 308 | # Pairs:
402 | Long:    841,222 | Short:   -784,183
2017-10-31 00:00:00 PF:   1,073,403 | Net:      67,754 | # Pos: 302 | # Pairs:
308 | Long:    828,308 | Short:   -760,554
```

```
2017-11-30 00:00:00 PF:   1,082,732 | Net:      67,242 | # Pos: 301 | # Pairs:
117 | Long:    902,876 | Short:   -835,634
2018-01-02 00:00:00 PF:   1,091,401 | Net:      37,340 | # Pos: 299 | # Pairs:
76 | Long:    999,405 | Short:   -962,065
2018-02-01 00:00:00 PF:   1,103,410 | Net:      49,068 | # Pos: 305 | # Pairs:
306 | Long:    868,493 | Short:   -819,425
2018-03-05 00:00:00 PF:   1,107,687 | Net:      44,104 | # Pos: 302 | # Pairs:
96 | Long:    957,032 | Short:   -912,928
2018-04-04 00:00:00 PF:   1,079,159 | Net:      22,311 | # Pos: 311 | # Pairs:
2,472 | Long:    555,688 | Short:   -533,376
2018-05-03 00:00:00 PF:   1,088,873 | Net:      65,666 | # Pos: 308 | # Pairs:
518 | Long:    764,980 | Short:   -699,314
2018-06-04 00:00:00 PF:   1,120,888 | Net:     135,075 | # Pos: 305 | # Pairs:
446 | Long:    931,262 | Short:   -796,187
2018-08-02 00:00:00 PF:   1,070,713 | Net:      56,885 | # Pos: 305 | # Pairs:
365 | Long:    837,295 | Short:   -780,409
2018-08-31 00:00:00 PF:   1,062,973 | Net:      40,063 | # Pos: 303 | # Pairs:
235 | Long:    828,683 | Short:   -788,621
2018-10-31 00:00:00 PF:   1,093,611 | Net:      86,639 | # Pos: 306 | # Pairs:
427 | Long:    906,958 | Short:   -820,318
2018-11-30 00:00:00 PF:   1,069,840 | Net:      45,130 | # Pos: 304 | # Pairs:
78 | Long:    936,790 | Short:   -891,661
2019-02-04 00:00:00 PF:   1,109,595 | Net:      69,074 | # Pos: 302 | # Pairs:
187 | Long:    946,751 | Short:   -877,677
2019-03-06 00:00:00 PF:   1,105,897 | Net:      78,295 | # Pos: 298 | # Pairs:
153 | Long:    919,108 | Short:   -840,812
2019-04-04 00:00:00 PF:   1,064,659 | Net:      25,119 | # Pos: 311 | # Pairs:
732 | Long:    740,735 | Short:   -715,616
2019-05-06 00:00:00 PF:   1,081,817 | Net:      98,903 | # Pos: 310 | # Pairs:
528 | Long:    831,293 | Short:   -732,390
2019-06-05 00:00:00 PF:   1,140,606 | Net:      99,417 | # Pos: 308 | # Pairs:
371 | Long:  1,051,369 | Short:   -951,951
2019-07-05 00:00:00 PF:   1,129,195 | Net:      21,267 | # Pos: 311 | # Pairs:
1,554 | Long:    475,318 | Short:   -454,052
2019-08-05 00:00:00 PF:   1,124,039 | Net:      74,430 | # Pos: 309 | # Pairs:
258 | Long:    921,601 | Short:   -847,171
2019-09-04 00:00:00 PF:   1,133,501 | Net:      62,659 | # Pos: 304 | # Pairs:
125 | Long:    967,723 | Short:   -905,064
Final Portfolio Value: 1,079,614.04 | Duration: 00:07:22
```

1.2.10 Get PyFolio Inputs

[17]:

```
pyfolio_analyzer = results[0].analyzers.getbyname('pyfolio')
returns, positions, transactions, gross_lev = pyfolio_analyzer.get_pf_items()
```

[18]:

```
returns.to_hdf(STORE, 'returns')
positions.to_hdf(STORE, 'positions')
```

```
transactions.to_hdf(STORE, 'transactions/')
gross_lev.to_hdf(STORE, 'gross_lev')
```

1.2.11 Get Positions

The PyFolio integration is somewhat broken due to API changes after version 0.5.1 so we need to retrieve the positions manually.

[19]:

```
traded_pairs = pd.DataFrame(results[0].exposure)
traded_pairs.date = pd.to_datetime(traded_pairs.date)
traded_pairs = traded_pairs.set_index('date').tz_localize('UTC')
traded_pairs.to_hdf(STORE, 'traded_pairs')
traded_pairs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 736 entries, 2017-01-04 00:00:00+00:00 to 2019-12-18
00:00:00+00:00
Columns: 312 entries, AA.US to cash
dtypes: float64(312)
memory usage: 1.8 MB
```

1.2.12 Get Metrics

[20]:

```
metrics = pd.DataFrame(results[0].metrics,
                       columns=['date', 'pf', 'net_holdings', 'cash',
                                'npositions', 'npairs', 'nlong_pos',␣
 ↪'nshort_pos',
                                'target', 'target_val', 'nlong_trades',
                                'nshort_trades', 'nclose_trades'])
metrics.to_hdf(STORE, 'metrics')
```

1.3 Run PyFolio Analysis

[21]:

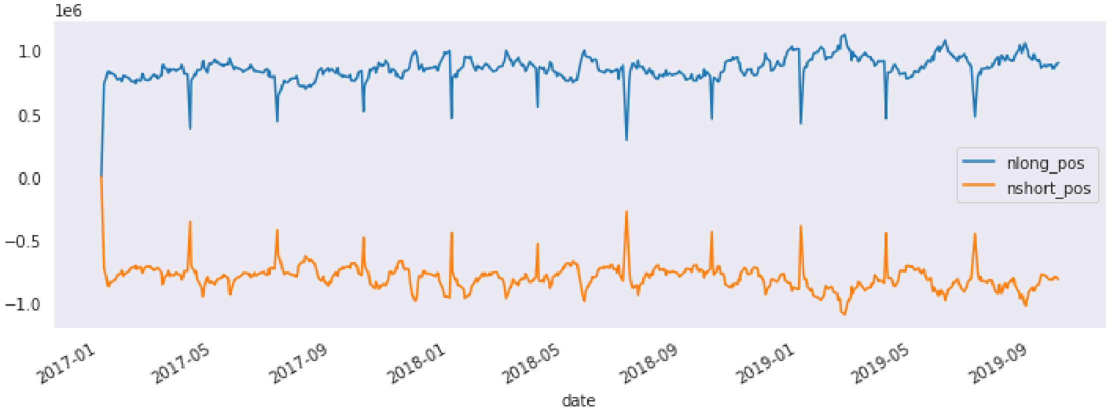
```
returns = pd.read_hdf(STORE, 'returns')
transactions = pd.read_hdf(STORE, 'transactions/')
gross_lev = pd.read_hdf(STORE, 'gross_lev')
metrics = pd.read_hdf(STORE, 'metrics').set_index('date')
```

[22]:

```
metrics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 676 entries, 2017-01-03 to 2019-09-30
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   pf             676 non-null    float64
 1   net_holdings   676 non-null    float64
 2   cash           676 non-null    float64
 3   npositions     676 non-null    int64
 4   npairs         676 non-null    int64
 5   nlong_pos      676 non-null    float64
 6   nshort_pos     676 non-null    float64
 7   target         676 non-null    float64
 8   target_val     676 non-null    float64
 9   nlong_trades   676 non-null    int64
 10  nshort_trades  676 non-null    int64
 11  nclose_trades  676 non-null    int64
dtypes: float64(7), int64(5)
memory usage: 68.7 KB
```

[23]:
```
metrics[['nlong_pos', 'nshort_pos']].plot(figsize=(12, 4));
```

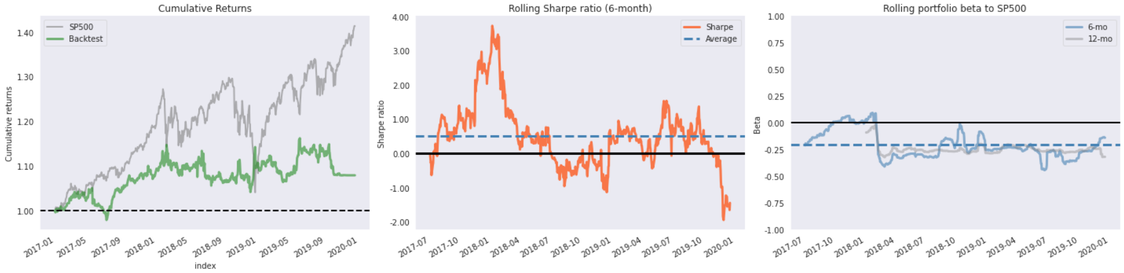


1.3.1 Get Benchmark

[24]:
```
start = str(returns.index.min().year)
end = str(returns.index.max().year + 1)
```

[25]:
```
benchmark = web.DataReader('SP500', 'fred',
                           start=start,
                           end=end).squeeze()
benchmark = benchmark.pct_change().tz_localize('UTC')
```

[26]:
```
fig, axes = plt.subplots(ncols=3, figsize=(20,5))
pf.plotting.plot_rolling_returns( returns, factor_returns=benchmark, ax=axes[0])
axes[0].set_title('Cumulative Returns')
pf.plotting.plot_rolling_sharpe(returns, ax=axes[1])
pf.plotting.plot_rolling_beta(returns, benchmark, ax=axes[2])
sns.despine()
fig.tight_layout();
```

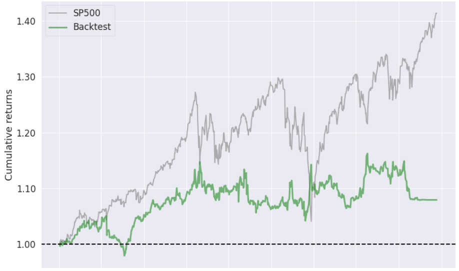


1.3.2 Create full tearsheet

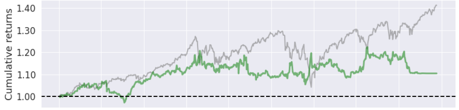
[27]:
```
pf.create_full_tear_sheet(returns,
                          positions=positions,
                          transactions=transactions,
                          benchmark_rets=benchmark.loc[returns.index],
                          estimate_intraday=False)
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

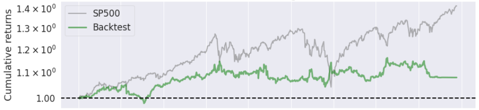
Cumulative returns



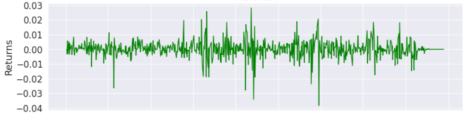
Cumulative returns volatility matched to benchmark



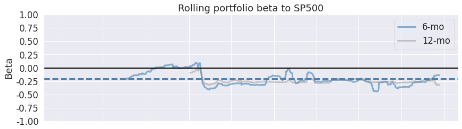
Cumulative returns on logarithmic scale



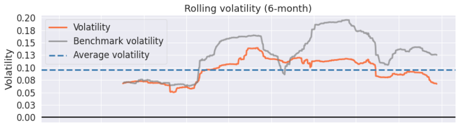
Returns



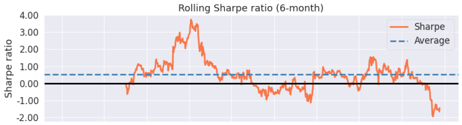
Rolling portfolio beta to SP500



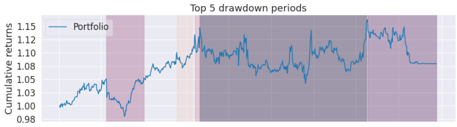
Rolling volatility (6-month)



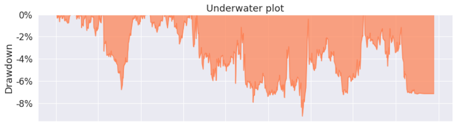
Rolling Sharpe ratio (6-month)



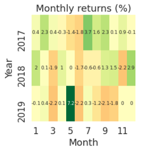
Top 5 drawdown periods



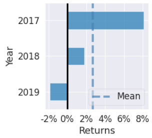
Underwater plot



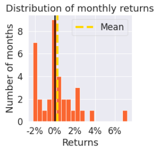
Monthly returns (%)



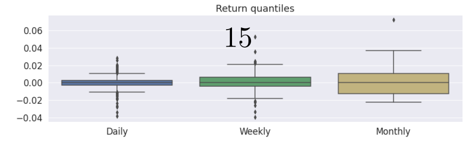
Annual returns

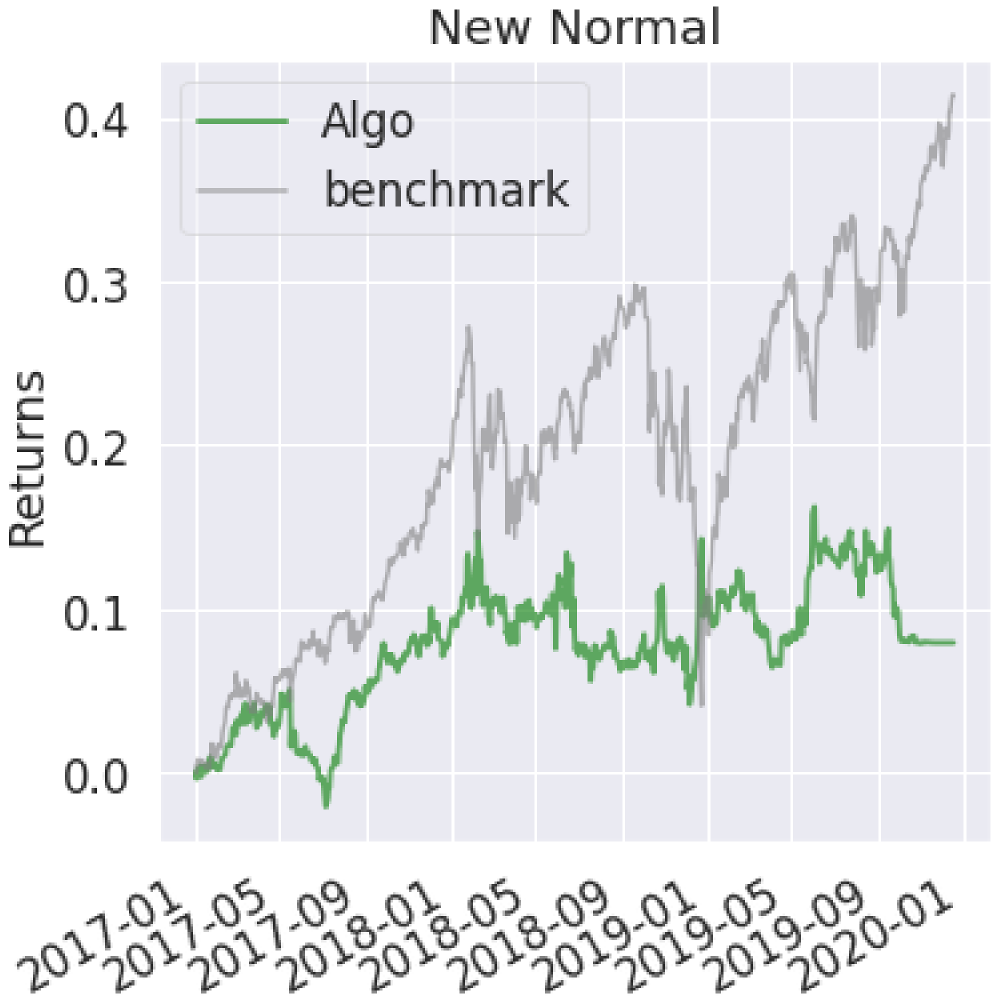


Distribution of monthly returns

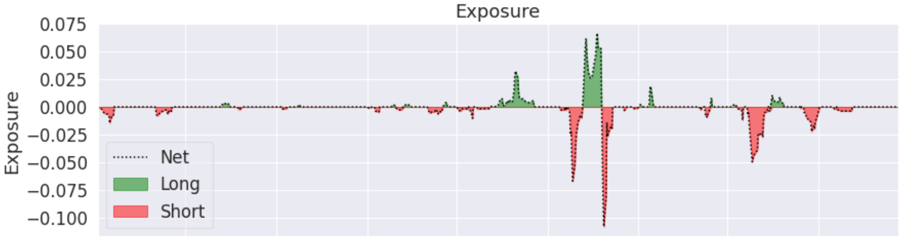


Return quantiles

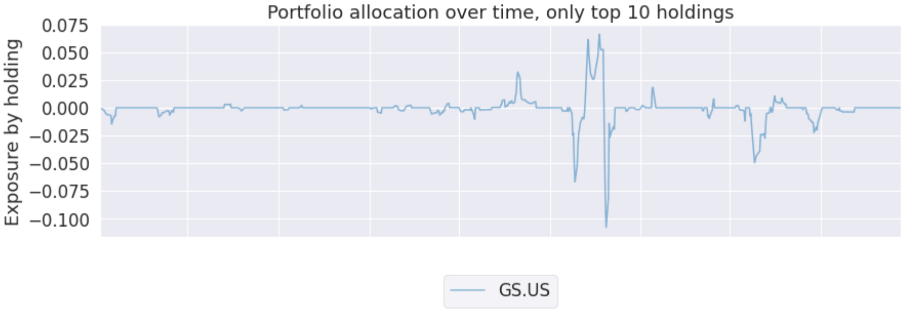



New Normal
Returns
0.4
0.3
0.2
0.1
0.0
Algo
benchmark
2017-01
2017-05
2017-09
2018-01
2018-05
2018-09
2019-01
2019-05
2019-09
2020-01

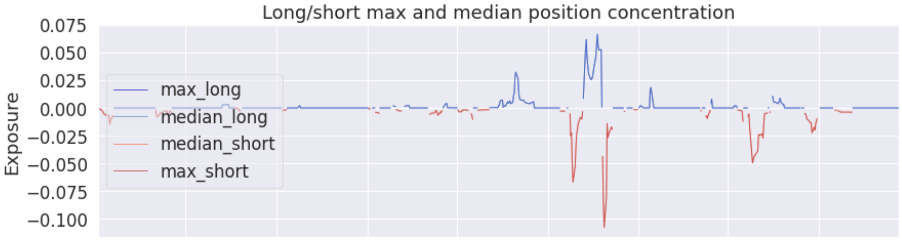
Exposure



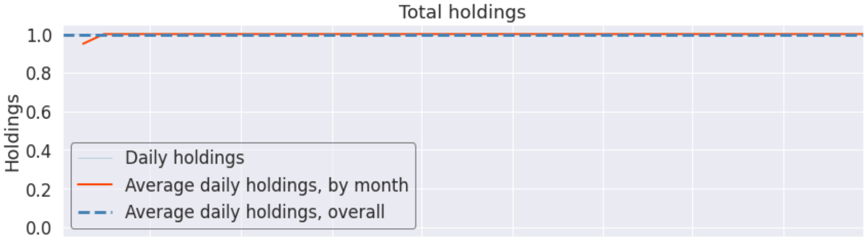
Portfolio allocation over time, only top 10 holdings



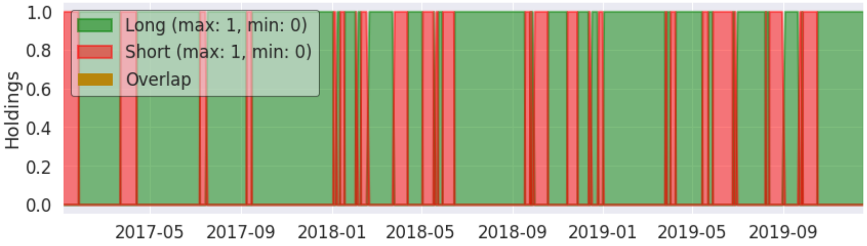
Long/short max and median position concentration



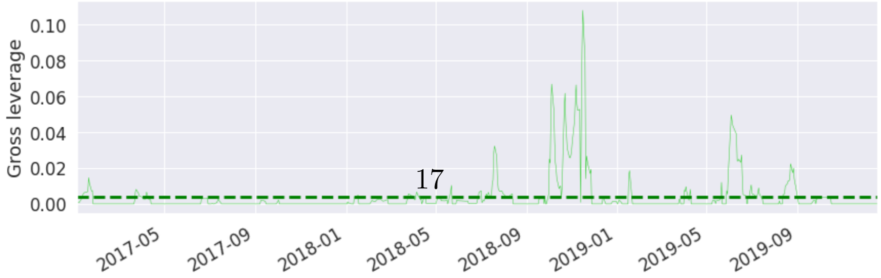
Total holdings

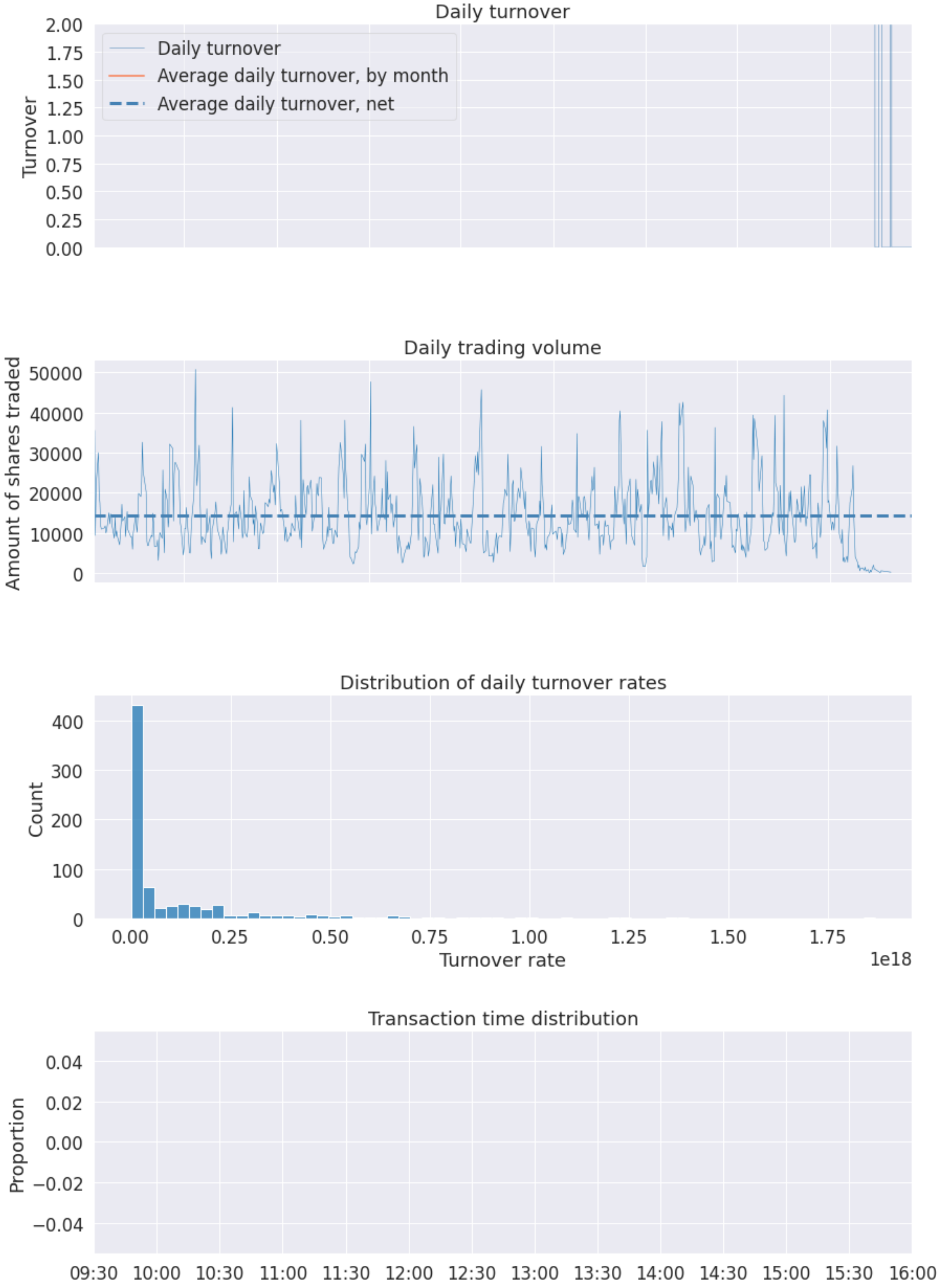


Long and short holdings



Gross leverage




Daily turnover
Turnover
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Daily turnover
Average daily turnover, by month
Average daily turnover, net
Daily trading volume
Amount of shares traded
50000
40000
30000
20000
10000
0
Distribution of daily turnover rates
Count
400
300
200
100
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
Turnover rate
1e18
Transaction time distribution
Proportion
0.04
0.02
0.00
−0.02
−0.04
09:30
10:00
10:30
11:00
11:30
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00