05_backtesting_with_zipline

September 29, 2021

1 Backtesting with zipline - Pipeline API with Custom Data

This notebook requires the `conda` environment `backtest`. Please see the installation instructions for running the latest Docker image or alternative ways to set up your environment.

1.1 Imports & Settings

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
from pathlib import Path
from time import time

import numpy as np
import pandas as pd
import pandas_datareader.data as web
from logbook import Logger, StderrHandler, INFO, WARNING

from zipline import run_algorithm
from zipline.api import (attach_pipeline, pipeline_output,
                         date_rules, time_rules, record,
                         schedule_function, commission, slippage,
                         set_slippage, set_commission, set_max_leverage,
                         order_target, order_target_percent,
                         get_open_orders, cancel_order)
from zipline.data import bundles
from zipline.utils.run_algo import load_extensions
from zipline.pipeline import Pipeline, CustomFactor
from zipline.pipeline.data import Column, DataSet
from zipline.pipeline.domain import US_EQUITIES
from zipline.pipeline.filters import StaticAssets
from zipline.pipeline.loaders import USEquityPricingLoader
from zipline.pipeline.loaders.frame import DataFrameLoader
from trading_calendars import get_calendar

import pyfolio as pf
from pyfolio.plotting import plot_rolling_returns, plot_rolling_sharpe
```

```
from pyfolio.timeseries import forecast_cone_bootstrap

from alphalens.tears import (create_returns_tear_sheet,
                             create_summary_tear_sheet,
                             create_full_tear_sheet)

from alphalens.performance import mean_return_by_quantile
from alphalens.plotting import plot_quantile_returns_bar
from alphalens.utils import get_clean_factor_and_forward_returns, rate_of_return

import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```
sns.set_style('whitegrid')

np.random.seed(42)
idx = pd.IndexSlice
```

[4]:
```
results_path = Path('results')
```

1.2 Alphalens Analysis

[5]:
```
DATA_STORE = Path('..', 'data', 'assets.h5')
```

[6]:
```
def get_trade_prices(tickers):
    prices = (pd.read_hdf(DATA_STORE, 'quandl/wiki/prices').swaplevel().
 ↪sort_index())
    prices.index.names = ['symbol', 'date']
    prices = prices.loc[idx[tickers, '2015':'2018'], 'adj_open']
    return (prices
            .unstack('symbol')
            .sort_index()
            .shift(-1)
            .tz_localize('UTC'))
```

[7]:
```
predictions = (pd.read_hdf(results_path / 'test_preds.h5', 'predictions')
               .iloc[:, :3]
               .mean(1)
               .to_frame('prediction'))
```

[8]:
```
factor = (predictions
          .unstack('symbol')
          .asfreq('D')
          .dropna(how='all')
          .stack()
          .tz_localize('UTC', level='date')
```

```
          .sort_index())
tickers = factor.index.get_level_values('symbol').unique()
```

[9]:
```
trade_prices = get_trade_prices(tickers)
```

[10]:
```
factor_data = get_clean_factor_and_forward_returns(factor=factor,
                                                   prices=trade_prices,
                                                   quantiles=5,
                                                   max_loss=0.3,
                                                   periods=(1, 5, 10, 21)).
 ↪sort_index()
factor_data.info()
```

```
Dropped 3.6% entries from factor data: 3.6% in forward returns computation and
0.0% in binning phase (set max_loss=0 to see potentially suppressed Exceptions).
max_loss is 30.0%, not exceeded: OK!
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 721920 entries, (2015-01-02 00:00:00+00:00, A) to (2017-11-29
00:00:00+00:00, ZION)
Data columns (total 6 columns):
1D                 721920 non-null float64
5D                 721920 non-null float64
10D                721920 non-null float64
21D                721920 non-null float64
factor             721920 non-null float32
factor_quantile    721920 non-null int64
dtypes: float32(1), float64(4), int64(1)
memory usage: 33.1+ MB
```

[11]:
```
create_summary_tear_sheet(factor_data)
```

```
Quantiles Statistics
```

factor_quantile	min	max	mean	std	count	count %
1	-0.027255	0.008472	-0.001468	0.002040	144553	20.023410
2	-0.012026	0.009544	-0.000369	0.001623	144304	19.988918
3	-0.011218	0.010532	0.000243	0.001622	144278	19.985317
4	-0.010551	0.012066	0.000887	0.001713	144304	19.988918
5	-0.009525	0.031386	0.002253	0.002399	144481	20.013436

```
Returns Analysis
```

	1D	5D	10D	21D
Ann. alpha	0.109	0.068	0.062	0.044
beta	0.153	0.200	0.236	0.241
Mean Period Wise Return Top Quantile (bps)	5.559	4.011	3.994	3.242
Mean Period Wise Return Bottom Quantile (bps)	-4.106	-3.846	-3.758	-3.019
Mean Period Wise Spread (bps)	9.665	7.839	7.725	6.240

```
Information Analysis
```

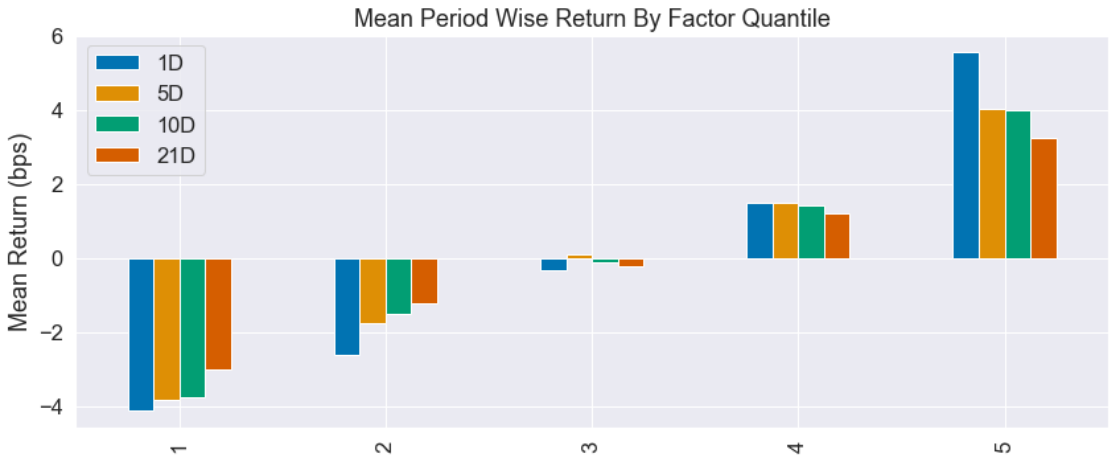
	1D	5D	10D	21D
IC Mean	0.020	0.035	0.049	0.058
IC Std.	0.147	0.153	0.151	0.137
Risk-Adjusted IC	0.136	0.228	0.323	0.421
t-stat(IC)	3.675	6.179	8.751	11.404
p-value(IC)	0.000	0.000	0.000	0.000
IC Skew	0.593	1.028	1.073	0.300
IC Kurtosis	5.385	5.988	5.712	0.592

```
Turnover Analysis
```

	1D	5D	10D	21D
Quantile 1 Mean Turnover	0.465	0.572	0.638	0.727
Quantile 2 Mean Turnover	0.660	0.716	0.747	0.773
Quantile 3 Mean Turnover	0.692	0.745	0.767	0.785
Quantile 4 Mean Turnover	0.651	0.714	0.746	0.775
Quantile 5 Mean Turnover	0.426	0.533	0.597	0.681

	1D	5D	10D	21D
Mean Factor Rank Autocorrelation	0.629	0.456	0.349	0.173

```
<Figure size 432x288 with 0 Axes>
```



1.2.1 Load zipline extensions

Only need this in notebook to find bundle.

[12]:
```
load_extensions(default=True,
                extensions=[],
                strict=True,
                environ=None)
```

[13]:
```
log_handler = StderrHandler(format_string='[{record.time:%Y-%m-%d %H:%M:%S.%f}]:
↪ ' +
                            '{record.level_name}: {record.func_name}: {record.
↪message}',
                            level=WARNING)
log_handler.push_application()
log = Logger('Algorithm')
```

1.3 Algo Params

[14]:
```
N_LONGS = 25
N_SHORTS = 25
MIN_POSITIONS = 10
```

1.4 Load Data

1.4.1 Quandl Wiki Bundel

[15]:
```
bundle_data = bundles.load('quandl')
```

1.4.2 ML Predictions

[16]:
```
def load_predictions(bundle):
    predictions = (pd.read_hdf(results_path / 'test_preds.h5', 'predictions')
                   .iloc[:, :3]
                   .mean(1)
                   .to_frame('prediction'))
    tickers = predictions.index.get_level_values('symbol').unique().tolist()

    assets = bundle.asset_finder.lookup_symbols(tickers, as_of_date=None)
    predicted_sids = pd.Int64Index([asset.sid for asset in assets])
    ticker_map = dict(zip(tickers, predicted_sids))

    return (predictions
            .unstack('symbol')
            .rename(columns=ticker_map)
            .prediction
            .tz_localize('UTC')), assets
```

[17]:
```
predictions, assets = load_predictions(bundle_data)
```

[18]:
```
predictions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 756 entries, 2014-11-28 to 2017-11-29
Columns: 995 entries, 0 to 3188
```

```
dtypes: float32(995)
memory usage: 2.9 MB
```

1.4.3 Define Custom Dataset

[19]:
```
class SignalData(DataSet):
    predictions = Column(dtype=float)
    domain = US_EQUITIES
```

1.4.4 Define Pipeline Loaders

[20]:
```
signal_loader = {SignalData.predictions:
                     DataFrameLoader(SignalData.predictions, predictions)}
```

1.5 Pipeline Setup

1.5.1 Custom ML Factor

[21]:
```
class MLSignal(CustomFactor):
    """Converting signals to Factor
        so we can rank and filter in Pipeline"""
    inputs = [SignalData.predictions]
    window_length = 1

    def compute(self, today, assets, out, predictions):
        out[:] = predictions
```

1.5.2 Create Pipeline

[22]:
```
def compute_signals():
    signals = MLSignal()
    return Pipeline(columns={
        'longs' : signals.top(N_LONGS, mask=signals > 0),
        'shorts': signals.bottom(N_SHORTS, mask=signals < 0)},
            screen=StaticAssets(assets))
```

1.6 Initialize Algorithm

[23]:
```
def initialize(context):
    """
    Called once at the start of the algorithm.
    """
    context.longs = context.shorts = None
    set_slippage(slippage.FixedSlippage(spread=0.00))
#     set_commission(commission.PerShare(cost=0.001, min_trade_cost=0))

    schedule_function(rebalance,
```

```
                    date_rules.every_day(),
                    time_rules.market_open(hours=1, minutes=30))

    schedule_function(record_vars,
                    date_rules.every_day(),
                    time_rules.market_close())

    pipeline = compute_signals()
    attach_pipeline(pipeline, 'signals')
```

1.6.1 Get daily Pipeline results

[24]:
```
def before_trading_start(context, data):
    """
    Called every day before market open.
    """
    output = pipeline_output('signals')
    longs = pipeline_output('signals').longs.astype(int)
    shorts = pipeline_output('signals').shorts.astype(int)
    holdings = context.portfolio.positions.keys()

    if longs.sum() > MIN_POSITIONS and shorts.sum() > MIN_POSITIONS:
        context.longs = longs[longs!=0].index
        context.shorts = shorts[shorts!=0].index
        context.divest = holdings - set(context.longs) - set(context.shorts)
    else:
        context.longs = context.shorts = pd.Index([])
        context.divest = set(holdings)
```

1.7 Define Rebalancing Logic

[25]:
```
def rebalance(context, data):
    """
    Execute orders according to schedule_function() date & time rules.
    """

    for symbol, open_orders in get_open_orders().items():
        for open_order in open_orders:
            cancel_order(open_order)

    for stock in context.divest:
        order_target(stock, target=0)

#     log.warning('{} {:,.0f}'.format(len(context.portfolio.positions), context.
 ↪portfolio.portfolio_value))
    if not (context.longs.empty and context.shorts.empty):
```

```
        for stock in context.shorts:
            order_target_percent(stock, -1 / len(context.shorts))
        for stock in context.longs:
            order_target_percent(stock, 1 / len(context.longs))
```

1.8 Record Data Points

[26]:
```
def record_vars(context, data):
    """
    Plot variables at the end of each day.
    """
    record(leverage=context.account.leverage,
           longs=context.longs,
           shorts=context.shorts)
```

1.9 Run Algorithm

[27]:
```
dates = predictions.index.get_level_values('date')
start_date, end_date = dates.min(), dates.max()
```

[28]:
```
print('Start: {}\nEnd:   {}'.format(start_date.date(), end_date.date()))
```

```
Start: 2014-11-28
End:   2017-11-29
```

[29]:
```
start = time()
results = run_algorithm(start=start_date,
                        end=end_date,
                        initialize=initialize,
                        before_trading_start=before_trading_start,
                        capital_base=1e5,
                        data_frequency='daily',
                        bundle='quandl',
                        custom_loader=signal_loader)  # need to modify zipline

print('Duration: {:.2f}s'.format(time() - start))
```

```
[2020-06-22 14:59:13.911299]: WARNING: _load_cached_data: Refusing to download
new benchmark data because a download succeeded at 2020-06-22
14:53:37.126521+00:00.

Duration: 48.13s
```

1.10 PyFolio Analysis

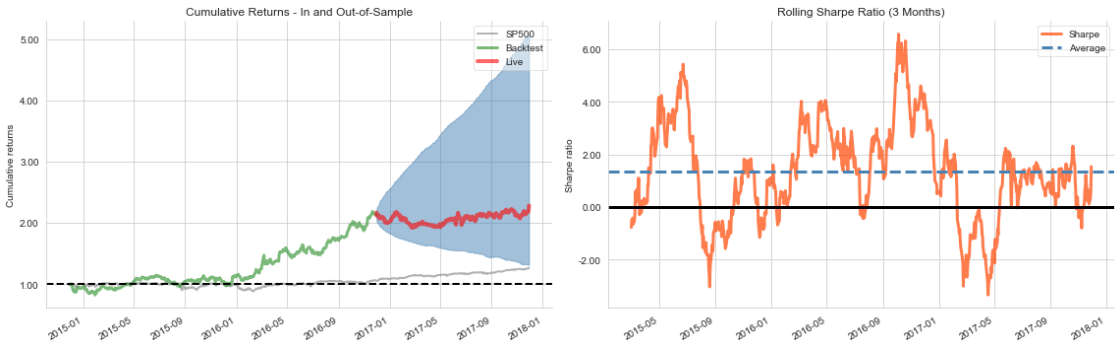
[30]:
```
returns, positions, transactions = pf.utils.
 ↪extract_rets_pos_txn_from_zipline(results)
```

[31]:
```
benchmark = web.DataReader('SP500', 'fred', '2014', '2018').squeeze()
benchmark = benchmark.pct_change().tz_localize('UTC')
```

1.10.1 Custom Plots

[32]:
```
LIVE_DATE = '2016-11-30'
```

[36]:
```
fig, axes = plt.subplots(ncols=2, figsize=(16, 5))
plot_rolling_returns(returns,
                     factor_returns=benchmark,
                     live_start_date=LIVE_DATE,
                     logy=False,
                     cone_std=2,
                     legend_loc='best',
                     volatility_match=False,
                     cone_function=forecast_cone_bootstrap,
                     ax=axes[0])
plot_rolling_sharpe(returns, ax=axes[1], rolling_window=63)
axes[0].set_title('Cumulative Returns - In and Out-of-Sample')
axes[1].set_title('Rolling Sharpe Ratio (3 Months)')
sns.despine()
fig.tight_layout()
fig.savefig((results_path / 'pyfolio_out_of_sample').as_posix(), dpi=300)
```

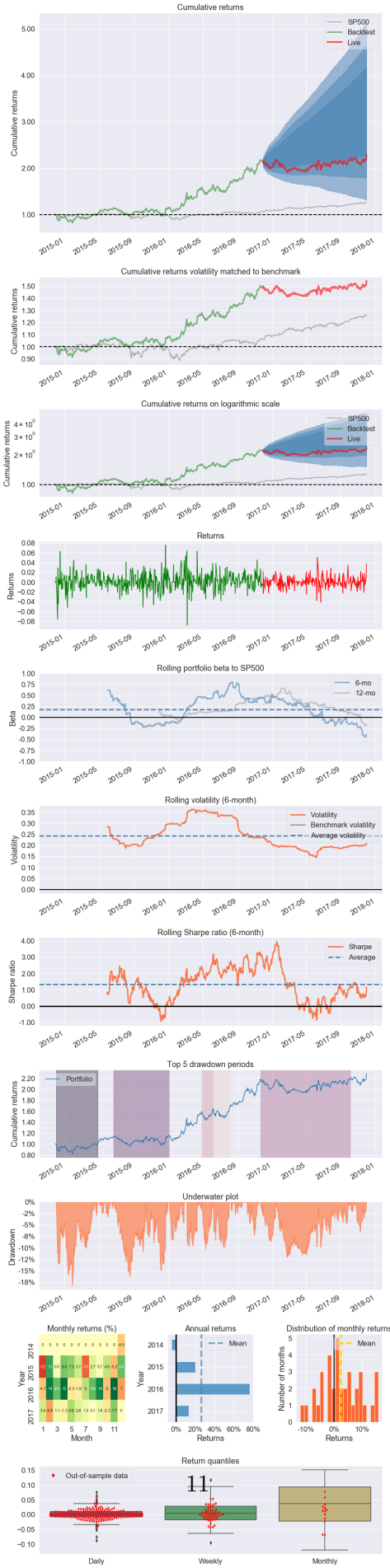


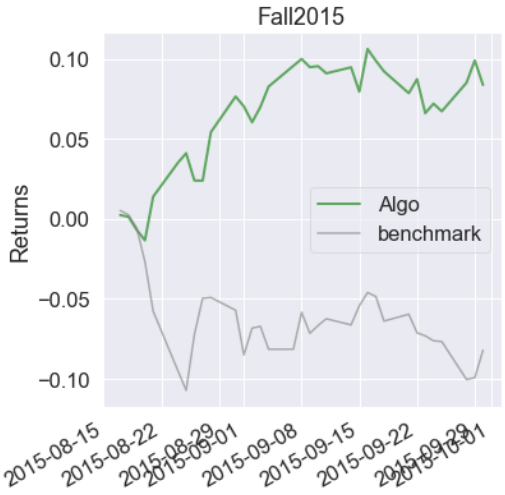
1.10.2 Tear Sheets

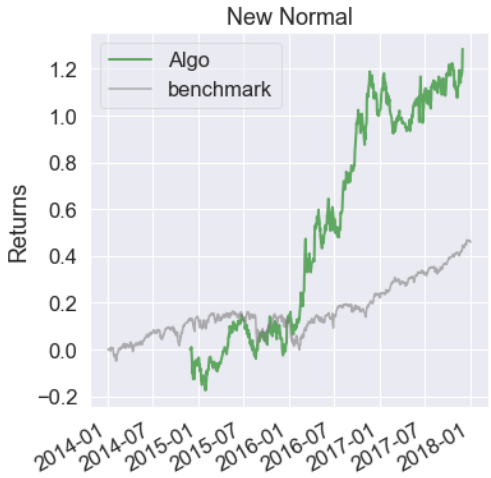
[37]:
```
pf.create_full_tear_sheet(returns,
                          positions=positions,
                          transactions=transactions,
```

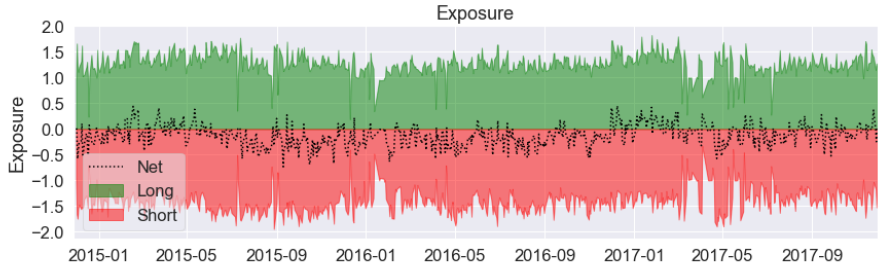
```
                            benchmark_rets=benchmark,
                            live_start_date=LIVE_DATE,
                            round_trips=True)
```

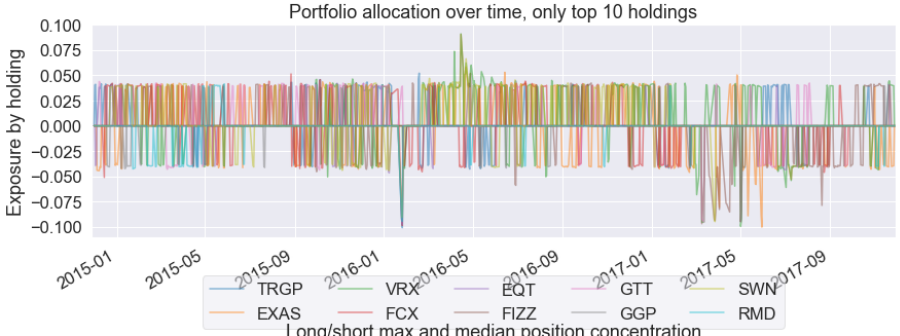
```
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML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

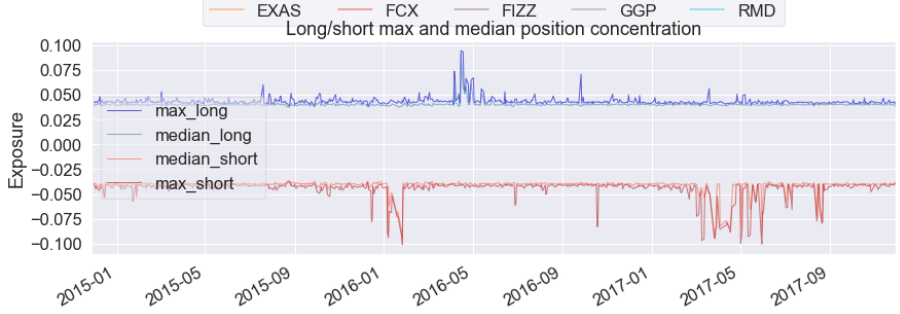

Cumulative returns
SP500
Backtest
Live
Cumulative returns volatility matched to benchmark
Cumulative returns on logarithmic scale
Returns
Rolling portfolio beta to SP500
6-mo
12-mo
Beta
Rolling volatility (6-month)
Volatility
Benchmark volatility
Average volatility
Rolling Sharpe ratio (6-month)
Sharpe
Average
Sharpe ratio
Top 5 drawdown periods
Portfolio
Underwater plot
Drawdown
Monthly returns (%)
Year
Month
Annual returns
Mean
Returns
Distribution of monthly returns
Number of months
Return quantiles
Out-of-sample data
Daily
Weekly
Monthly

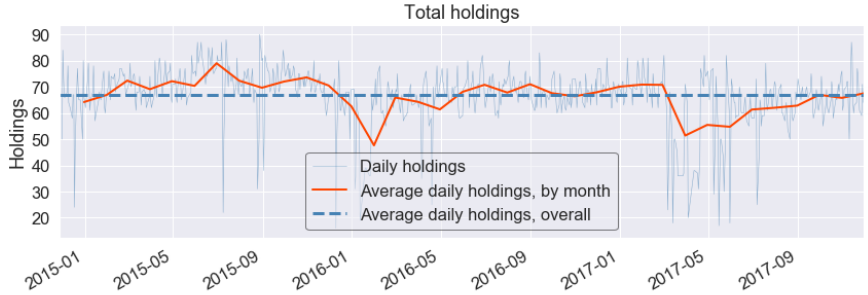

Fall2015
Returns
0.10
0.05
0.00
−0.05
−0.10
Algo
benchmark
2015-08-15
2015-08-22
2015-08-29
2015-09-01
2015-09-08
2015-09-15
2015-09-22
2015-09-29
2015-10-01

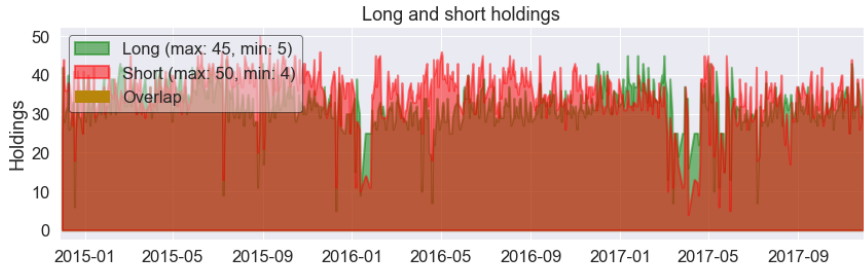

New Normal
Returns
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
Algo
benchmark
2014-01
2014-07
2015-01
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01

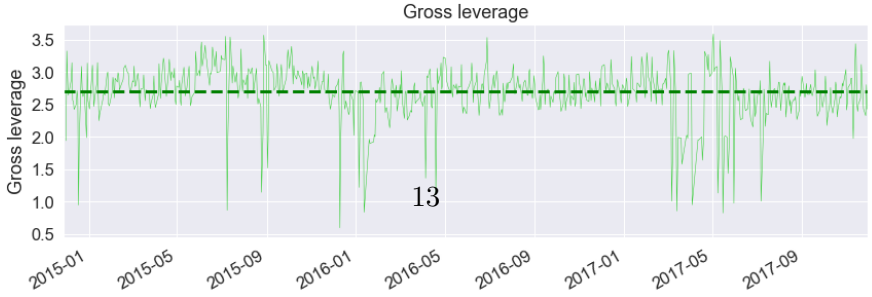

Exposure
Exposure
Net
Long
Short
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09

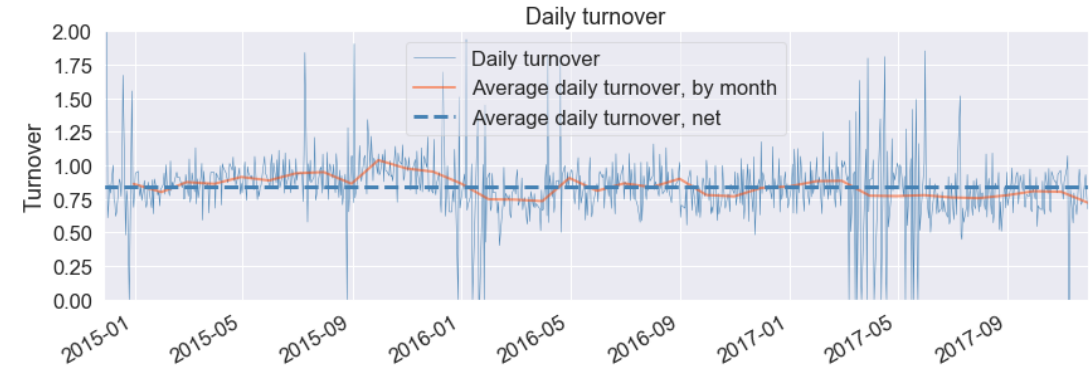

Portfolio allocation over time, only top 10 holdings
Exposure by holding
TRGP
VRX
EQT
GTT
SWN
EXAS
FCX
FIZZ
GGP
RMD

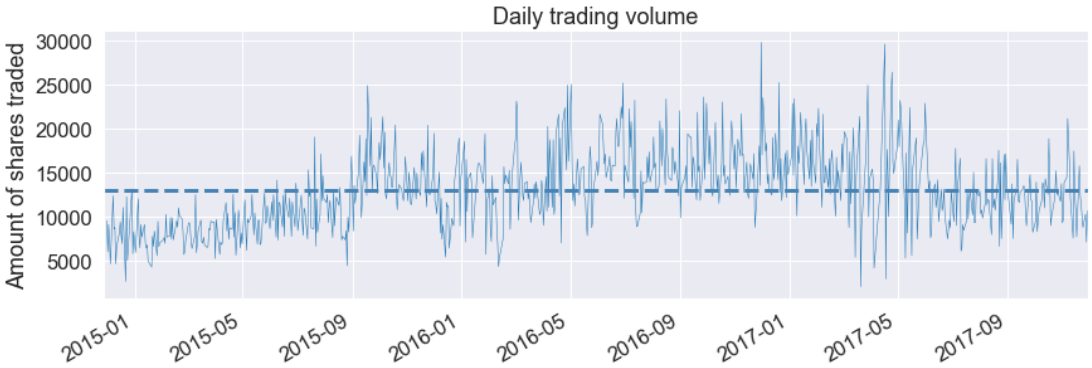

Long/short max and median position concentration
Exposure
max_long
median_long
median_short
max_short

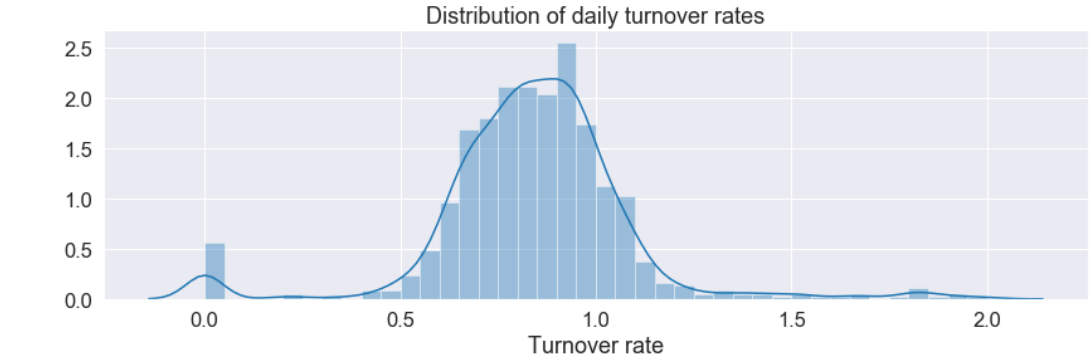

Total holdings
Holdings
Daily holdings
Average daily holdings, by month
Average daily holdings, overall

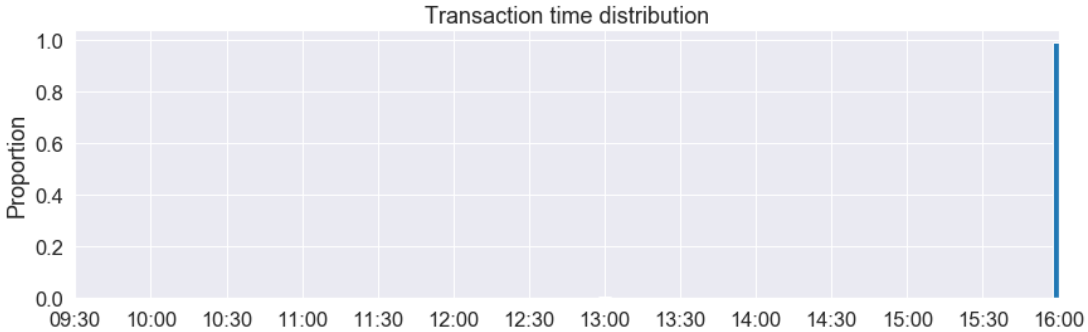

Long and short holdings
Holdings
Long (max: 45, min: 5)
Short (max: 50, min: 4)
Overlap

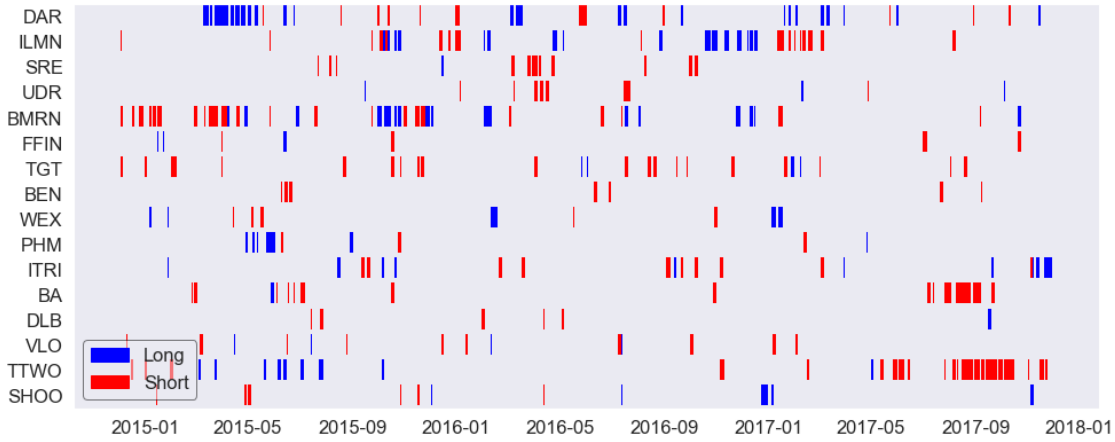

Gross leverage
Gross leverage

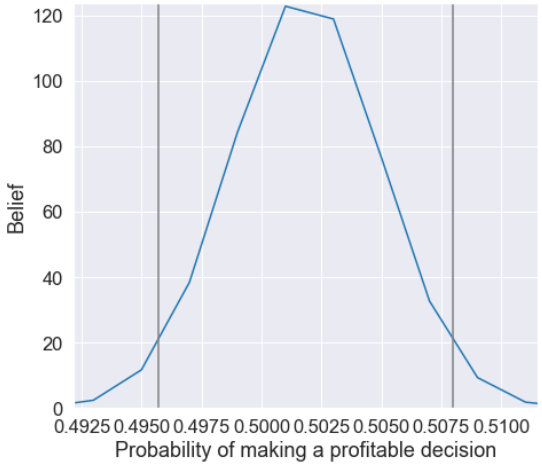

Daily turnover
Daily turnover
Average daily turnover, by month
Average daily turnover, net
Turnover
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09

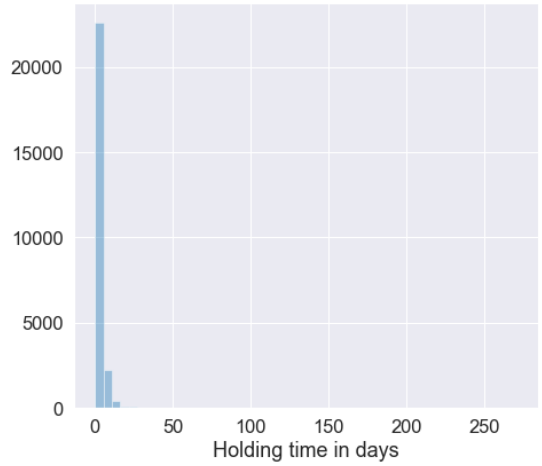

Daily trading volume
Amount of shares traded
30000
25000
20000
15000
10000
5000
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09

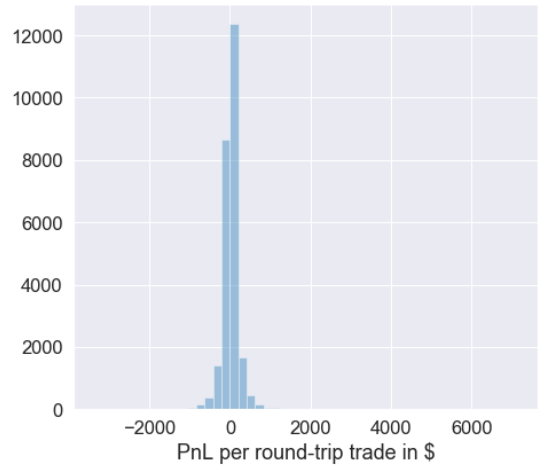

Distribution of daily turnover rates
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
Turnover rate

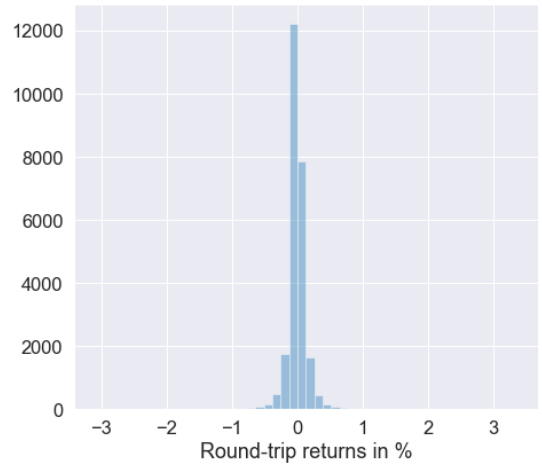

Transaction time distribution
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
09:30
10:00
10:30
11:00
11:30
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00


DAR
ILMN
SRE
UDR
BMRN
FFIN
TGT
BEN
WEX
PHM
ITRI
BA
DLB
VLO
TTWO
SHOO
Long
Short
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01


Belief
Probability of making a profitable decision


Holding time in days


PnL per round-trip trade in $


Round-trip returns in %

[ ]: