02_pf_optimization_with_hrp_zipline_benchmark

September 29, 2021

1 PF Optimization: HRP vs Markowitz and Equal-Weighted Positions

1.1 Imports & Settings

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.

[2]:
```
from time import time
import warnings
import sys

from pathlib import Path
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import pandas_datareader.data as web

from logbook import (NestedSetup, NullHandler, Logger,
                     StreamHandler, StderrHandler,
                     INFO, WARNING, DEBUG, ERROR)

from zipline import run_algorithm
from zipline.api import (attach_pipeline, pipeline_output,
                         date_rules, time_rules, record,get_datetime,
                         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.frame import DataFrameLoader
```

```python
from pypfopt.efficient_frontier import EfficientFrontier
from pypfopt.hierarchical_portfolio import HRPOpt
from pypfopt import risk_models
from pypfopt import expected_returns

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

[3]:
```python
sns.set_style('darkgrid')
warnings.filterwarnings('ignore')
np.random.seed(42)
```

1.1.1 Load zipline extensions

Only need this in notebook to find bundle.

[4]:
```python
load_extensions(default=True,
                extensions=[],
                strict=True,
                environ=None)
```

1.1.2 Logging Setup

[5]:
```python
# setup stdout logging
format_string = '[{record.time: %H:%M:%S.%f}]: {record.level_name}: {record.
 ↪message}'
zipline_logging = NestedSetup([NullHandler(level=DEBUG),
                               StreamHandler(sys.stdout,␣
 ↪format_string=format_string, level=INFO),
                               StreamHandler(sys.stdout,␣
 ↪format_string=format_string, level=WARNING),
                               StreamHandler(sys.stderr, level=ERROR)])
zipline_logging.push_application()
log = Logger('Algorithm')
```

1.2 Algo Params

[6]:
```python
N_LONGS = 25
MIN_POSITIONS = 20
```

1.3 Load Data

1.3.1 Quandl Wiki Bundel

[7]:
```
bundle_data = bundles.load('quandl')
```

1.3.2 ML Predictions

[8]:
```
def load_predictions(bundle):
    path = Path('../../12_gradient_boosting_machines/data')
    predictions = (pd.read_hdf(path / 'predictions.h5', 'lgb/train/01')
                   .append(pd.read_hdf(path / 'predictions.h5', 'lgb/test/01').
 ↪drop('y_test', axis=1)))
    predictions = (predictions.loc[~predictions.index.duplicated()]
                   .iloc[:, :10]
                   .mean(1)
                   .sort_index()
                   .dropna()
                  .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
```

[9]:
```
predictions, assets = load_predictions(bundle_data)
```

[10]:
```
predictions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 753 entries, 2015-01-02 to 2017-12-28
Columns: 995 entries, 0 to 3188
dtypes: float64(995)
memory usage: 5.7 MB
```

1.3.3 Define Custom Dataset

[11]:
```
class SignalData(DataSet):
    predictions = Column(dtype=float)
    domain = US_EQUITIES
```

1.3.4 Define Pipeline Loaders

[12]:
```
signal_loader = {SignalData.predictions:
                     DataFrameLoader(SignalData.predictions, predictions)}
```

1.4 Pipeline Setup

1.4.1 Custom ML Factor

[13]:
```
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
```

1.4.2 Create Pipeline

[14]:
```
def compute_signals():
    signals = MLSignal()
    return Pipeline(columns={
        'longs' : signals.top(N_LONGS, mask=signals > 0)
    },
            screen=StaticAssets(assets))
```

1.4.3 Get daily Pipeline results

[15]:
```
def before_trading_start(context, data):
    """
    Called every day before market open.
    """
    output = pipeline_output('signals')['longs'].astype(int)
    context.longs = output[output!=0].index
    if len(context.longs) < MIN_POSITIONS:
        context.divest = set(context.portfolio.positions.keys())
    else:
        context.divest = context.portfolio.positions.keys() - context.longs
```

1.5 Define Rebalancing Logic

1.5.1 Equal Weights

[16]:
```
def rebalance_equal_weighted(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 asset in context.divest:
        order_target(asset, target=0)

    if len(context.longs) > context.min_positions:
        for asset in context.longs:
            order_target_percent(asset, 1/len(context.longs))
```

1.5.2 Markowitz Mean-Variance Optimization

[56]:
```
def optimize_weights(prices, short=False):
    """Uses PyPortfolioOpt to optimize weights"""
    returns = expected_returns.mean_historical_return(prices=prices,
                                                       frequency=252)
    cov = risk_models.sample_cov(prices=prices, frequency=252)

    # get weights that maximize the Sharpe ratio
    # using solver SCS which produces slightly fewer errors than default
    # see https://github.com/robertmartin8/PyPortfolioOpt/issues/221
    ef = EfficientFrontier(expected_returns=returns,
                           cov_matrix=cov,
                           weight_bounds=(0, 1),
                           solver='SCS')

    weights = ef.max_sharpe()
    if short:
        return {asset: -weight for asset, weight in ef.clean_weights().items()}
    else:
        return ef.clean_weights()
```

[57]:
```
def rebalance_markowitz(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 asset in context.divest:
        order_target(asset, target=0)

    if len(context.longs) > context.min_positions:
        prices = data.history(context.longs, fields='price',
                              bar_count=252+1, # for 1 year of returns
                              frequency='1d')
        try:
            markowitz_weights = optimize_weights(prices)
            for asset, target in markowitz_weights.items():
                order_target_percent(asset=asset, target=target)
        except Exception as e:
            log.warn('{} {}'.format(get_datetime().date(), e))
```

1.5.3 Hierarchical Risk Parity

[58]:
```
def rebalance_hierarchical_risk_parity(context, data):
    """
    Execute orders according to schedule_function() date & time rules.
    Uses PyPortfolioOpt to optimize weights
    """
    for symbol, open_orders in get_open_orders().items():
        for open_order in open_orders:
            cancel_order(open_order)

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

    if len(context.longs) > context.min_positions:
        returns = (data.history(context.longs, fields='price',
                              bar_count=252+1, # for 1 year of returns
                              frequency='1d')
                   .pct_change()
                   .dropna(how='all'))
        hrp_weights = HRPOpt(returns=returns).optimize()
        for asset, target in hrp_weights.items():
            order_target_percent(asset=asset, target=target)
```

1.6 Record Additional Data Points

Define addtional variables to capture in the results `DataFrame`.

[59]:
```
def record_vars(context, data):
    """
    Plot variables at the end of each day.
```

```
    """
    record(leverage=context.account.leverage,
           longs=context.longs)
```

1.7 Initialize Algorithm with PF optimization algorithm

Execute `run_algorithm` once with each algorithm:

1.7.1 Select portfolio optimization algorithm

[60]:
```
pf_algos = {
    'ew': rebalance_equal_weighted,
    'markowitz': rebalance_markowitz,
    'hrp': rebalance_hierarchical_risk_parity
}
```

[61]:
```
# more descriptive labels for plots
algo_labels = {
    'ew': 'Equal Weighted',
    'markowitz': 'Markowitz (MFT)',
    'hrp': 'Hierarchical Risk Parity'
    }
```

Here we set the algo we are going to use:

[69]:
```
selected_pf_algo = 'hrp'
```

1.7.2 Schedule rebalancing using selected algo

[70]:
```
def initialize(context):
    """
    Called once at the start of the algorithm.
    """
    context.n_longs = N_LONGS
    context.min_positions = MIN_POSITIONS
    context.universe = assets
    context.trades = pd.Series()
    context.longs = 0
    context.pf_algo = pf_algos.get(selected_pf_algo)

    set_slippage(slippage.FixedSlippage(spread=0.00))
    set_commission(commission.PerShare(cost=0.001, min_trade_cost=1))

    schedule_function(context.pf_algo,
                      # run every day after market open
                      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')
```

1.8 Run trading algorithm for each PF optimization approach

[71]:
```
dates = predictions.index.get_level_values('date')
start_date, end_date = dates.min(), dates.max()
```

[72]:
```
print('Start: {}\nEnd:   {}'.format(start_date.date(), end_date.date()))
```

```
Start: 2015-01-02
End:   2017-12-28
```

[73]:
```
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))
```

```
[ 18:22:16.112701]: INFO: after split: asset: Equity(406 [BOFI]), amount: 208,
cost_basis: 21.58, last_sale_price: 83.29
[ 18:22:16.113149]: INFO: returning cash: 0.0
[ 18:23:15.358703]: INFO: Simulated 754 trading days
first open: 2015-01-02 14:31:00+00:00
last close: 2017-12-28 21:00:00+00:00
Duration: 82.44s
```

1.8.1 Persist experiment results

[74]:
```
returns, positions, transactions = pf.utils.
 ↪extract_rets_pos_txn_from_zipline(results)
```

[75]:
```
with pd.HDFStore('backtests.h5') as store:
    store.put('returns/{}'.format(selected_pf_algo), returns)
    store.put('positions/{}'.format(selected_pf_algo), positions)
    store.put('transactions/{}'.format(selected_pf_algo), transactions)
```

```
[76]: with pd.HDFStore('backtests.h5') as store:
          print(store.info())
```

```
<class 'pandas.io.pytables.HDFStore'>
File path: backtests.h5
/positions/ew                      frame        (shape->[700,1])
/positions/hrp                     frame        (shape->[700,1])
/positions/markowitz               frame        (shape->[691,1])
/returns/ew                        series       (shape->[754])
/returns/hrp                       series       (shape->[754])
/returns/markowitz                 series       (shape->[754])
/transactions/ew                   frame        (shape->[24443,8])
/transactions/hrp                  frame        (shape->[25332,8])
/transactions/markowitz            frame        (shape->[4489,8])
```

1.9 Comparing results using pyfolio

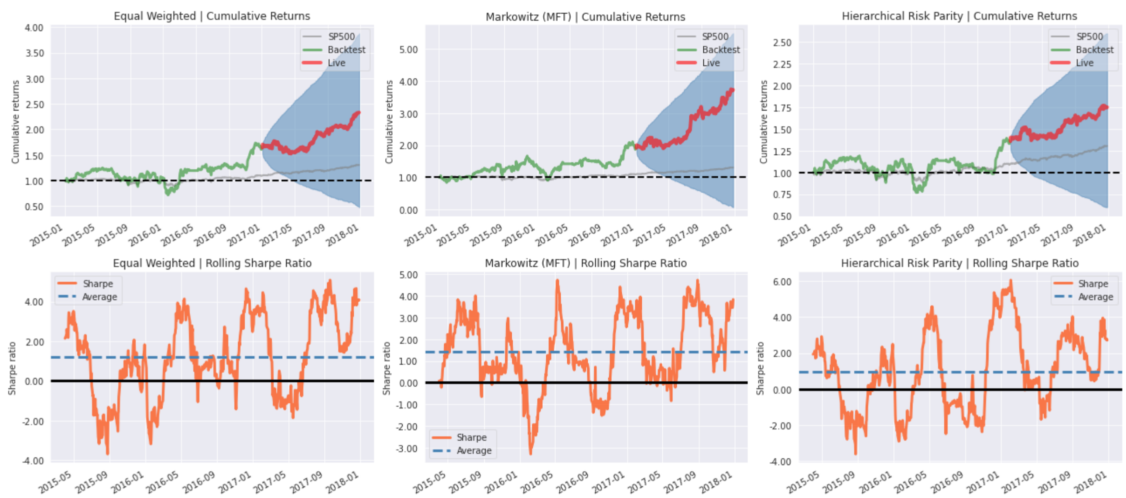
Once you're run the three algorithms (or those you're interested in), let's compare the results.

1.9.1 Load benchmark

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

1.9.2 Cumulative Returns & Rolling Sharpe Ratios

```
[78]: fig, axes = plt.subplots(ncols=3, nrows=2, figsize=(18, 8))

      for i, (algo, label) in enumerate(algo_labels.items()):
          returns = pd.read_hdf('backtests.h5', f'returns/{algo}')
          plot_rolling_returns(returns,
                                factor_returns=benchmark,
                                live_start_date='2017-01-01',
                                logy=False,
                                cone_std=2,
                                legend_loc='best',
                                volatility_match=False,
                                cone_function=forecast_cone_bootstrap,
                               ax=axes[0][i])
          plot_rolling_sharpe(returns, ax=axes[1][i], rolling_window=63)
          axes[0][i].set_title(f'{label} | Cumulative Returns')
          axes[1][i].set_title(f'{label} | Rolling Sharpe Ratio')
          fig.tight_layout()
```



1.9.3 Tear Sheets

[79]:
```
def load_results(experiment='hrp'):
    with pd.HDFStore('backtests.h5') as store:
        returns = store.get('returns/{}'.format(experiment))
        positions = store.get('positions/{}'.format(experiment))
        transactions = store.get('transactions/{}'.format(experiment))
    return returns, positions, transactions
```

Equal Weighted

[80]:
```
experiment = 'ew'
returns, positions, transactions = load_results(experiment)

pf.create_full_tear_sheet(returns,
                          positions=positions,
                          transactions=transactions,
                          benchmark_rets=benchmark,
                          live_start_date='2017-01-01',
                          round_trips=True)
```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

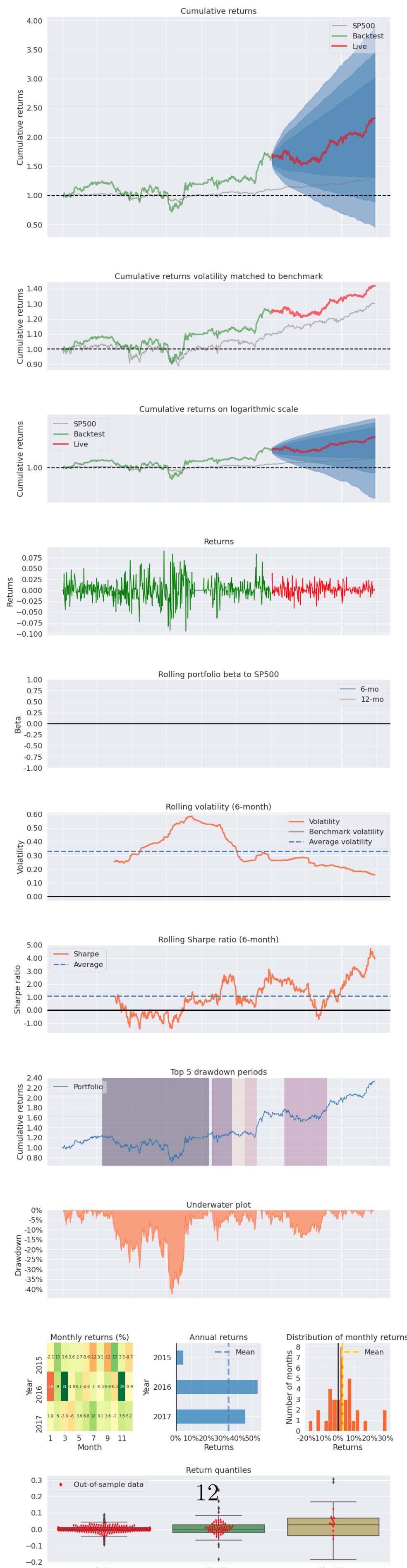
<IPython.core.display.HTML object>

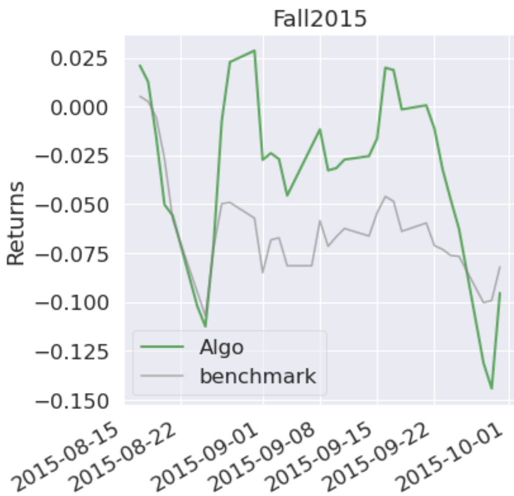
<IPython.core.display.HTML object>

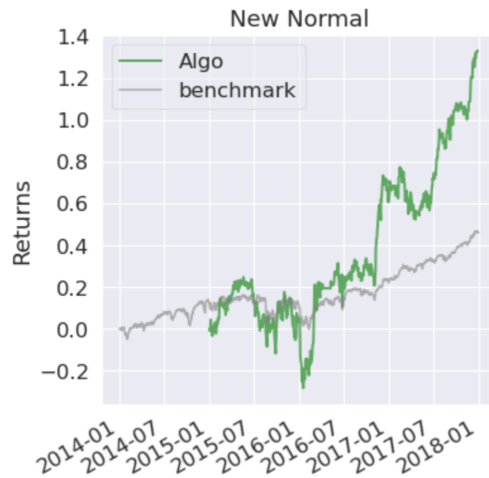
<IPython.core.display.HTML object>

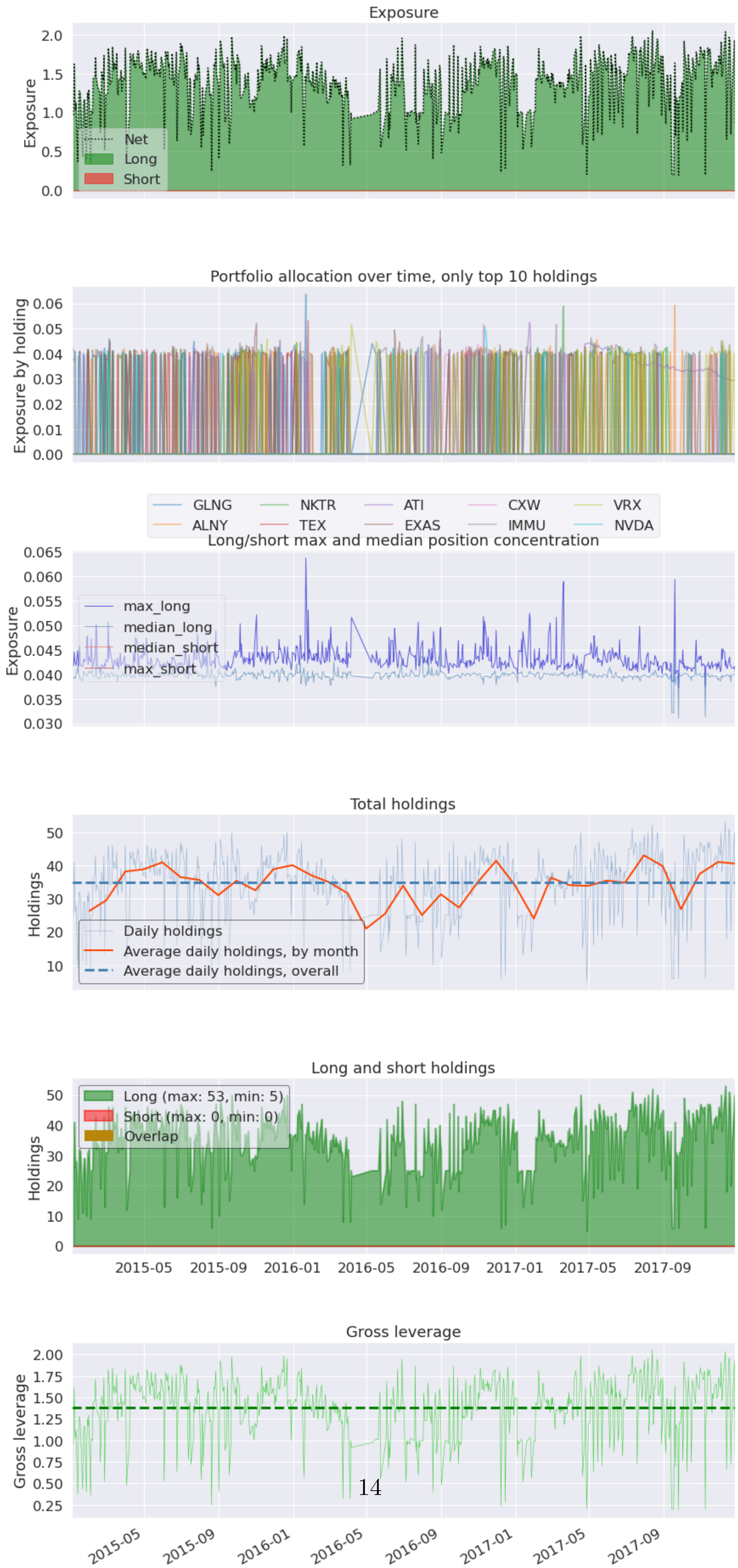
<IPython.core.display.HTML object>

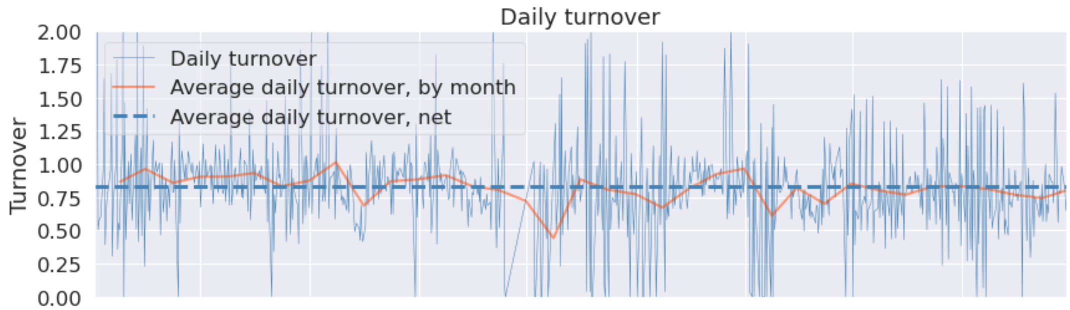
```
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
```

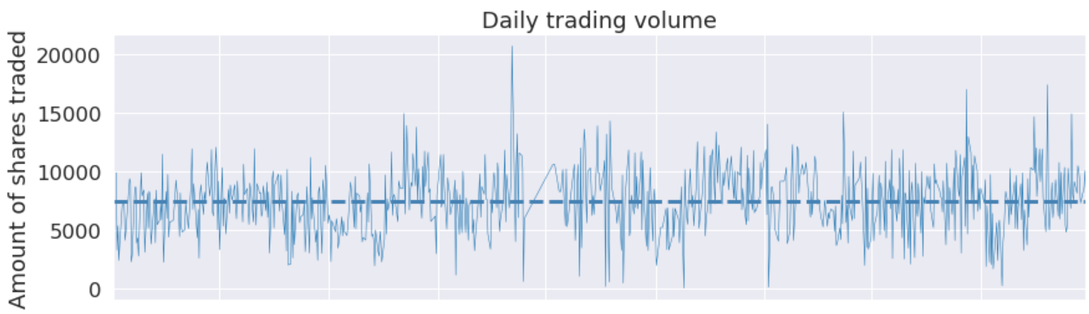

Cumulative returns
Cumulative returns
SP500
Backtest
Live
Cumulative returns volatility matched to benchmark
Cumulative returns
Cumulative returns on logarithmic scale
Cumulative returns
SP500
Backtest
Live
Returns
Returns
Rolling portfolio beta to SP500
Beta
6-mo
12-mo
Rolling volatility (6-month)
Volatility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

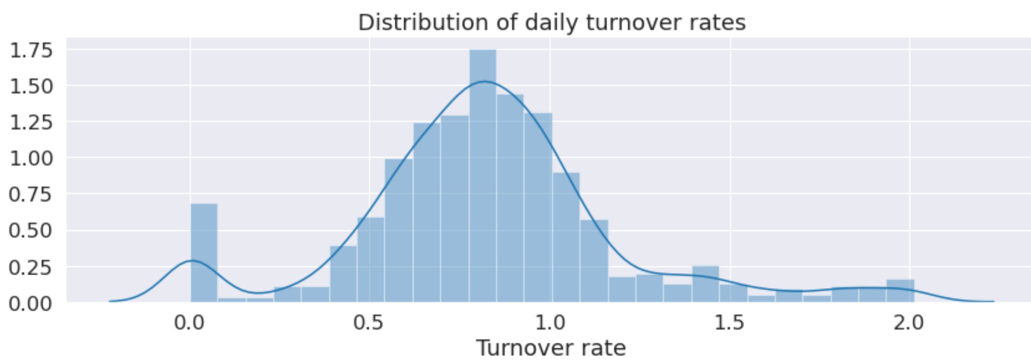

Fall2015
Returns
Algo
benchmark
0.025
0.000
−0.025
−0.050
−0.075
−0.100
−0.125
−0.150
2015-08-15
2015-08-22
2015-09-01
2015-09-08
2015-09-15
2015-09-22
2015-10-01

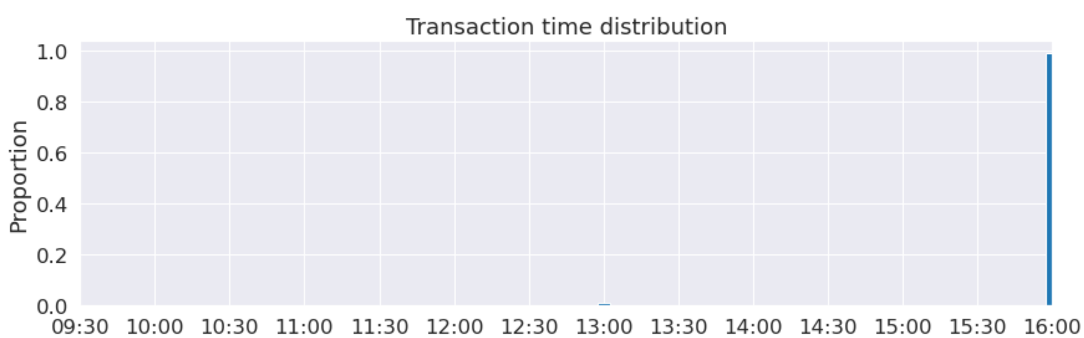

New Normal
Returns
Algo
benchmark
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
2014-01
2014-07
2015-01
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01

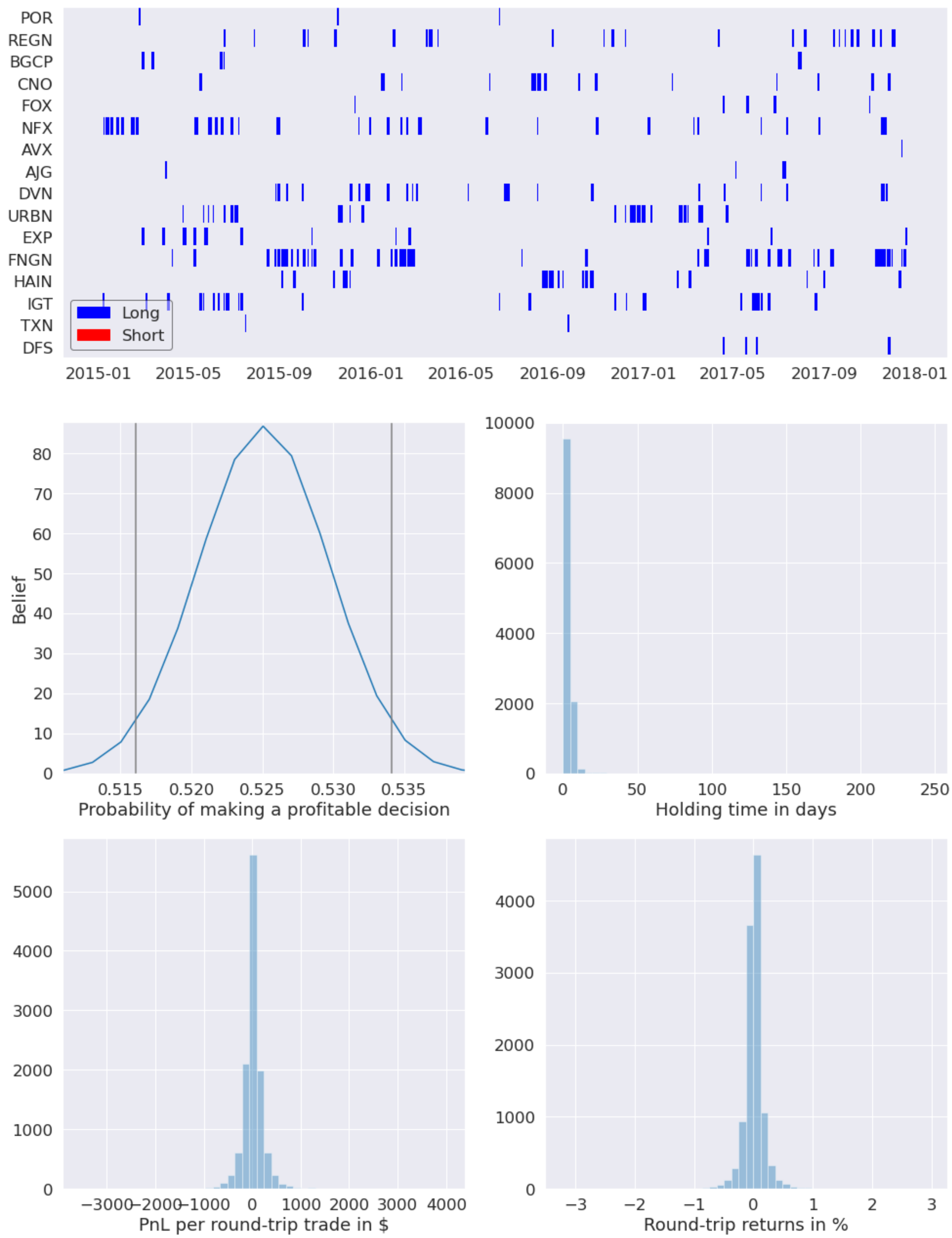

Exposure
Net
Long
Short
Portfolio allocation over time, only top 10 holdings
Exposure by holding
GLNG
NKTR
ATI
CXW
VRX
ALNY
TEX
EXAS
IMMU
NVDA
Long/short max and median position concentration
max_long
median_long
median_short
max_short
Total holdings
Holdings
Daily holdings
Average daily holdings, by month
Average daily holdings, overall
Long and short holdings
Long (max: 53, min: 5)
Short (max: 0, min: 0)
Overlap
Gross leverage
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09


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
20000
15000
10000
5000
0


Distribution of daily turnover rates
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
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



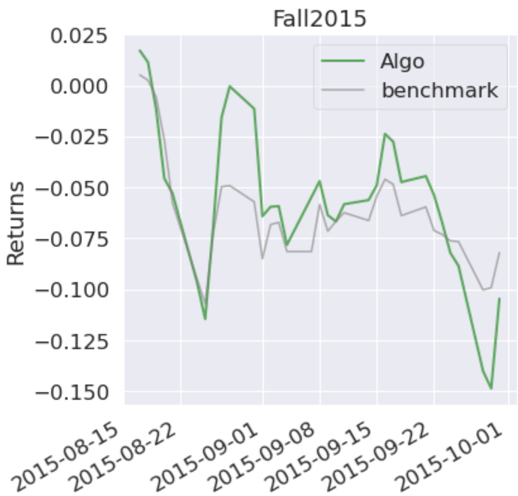
HRP

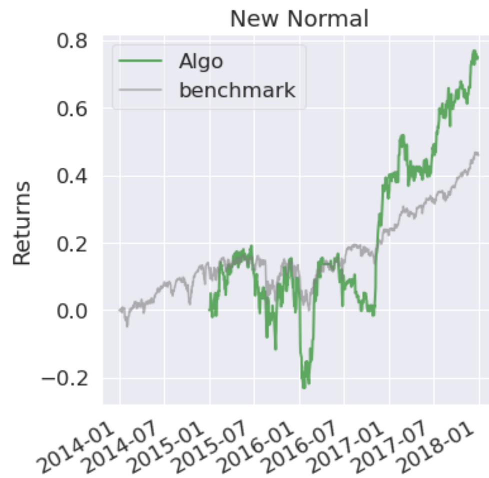
[81]:
```
experiment = 'hrp'
returns, positions, transactions = load_results(experiment)
```

```python
pf.create_full_tear_sheet(returns,
                          positions=positions,
                          transactions=transactions,
                          benchmark_rets=benchmark,
                          live_start_date='2017-01-01',
                          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>
```

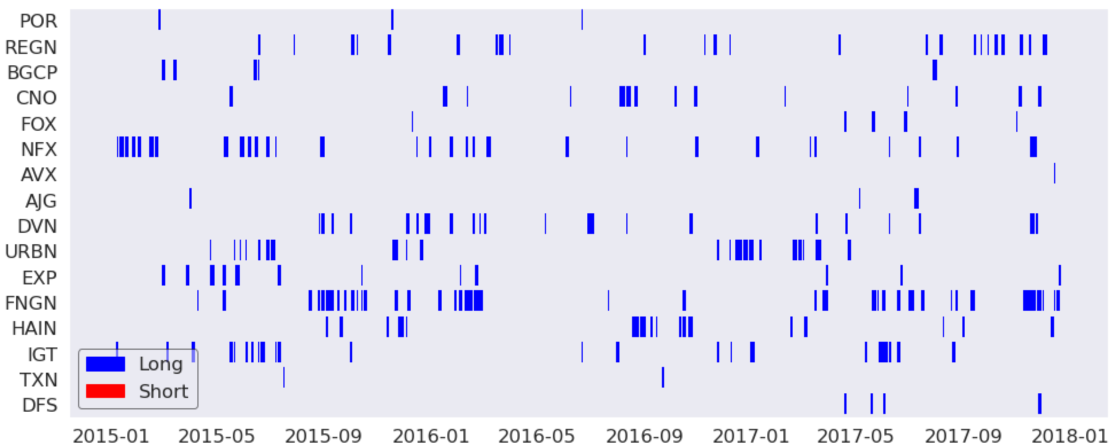

Cumulative returns
SP500
Backtest
Live
Cumulative returns volatility matched to benchmark
Cumulative returns on logarithmic scale
Returns
Rolling portfolio beta to SP500
6-mo
12-mo
Beta
Rolling volatility (6-month)
Volatility
Benchmark volatility
Average volatility
Rolling Sharpe ratio (6-month)
Sharpe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

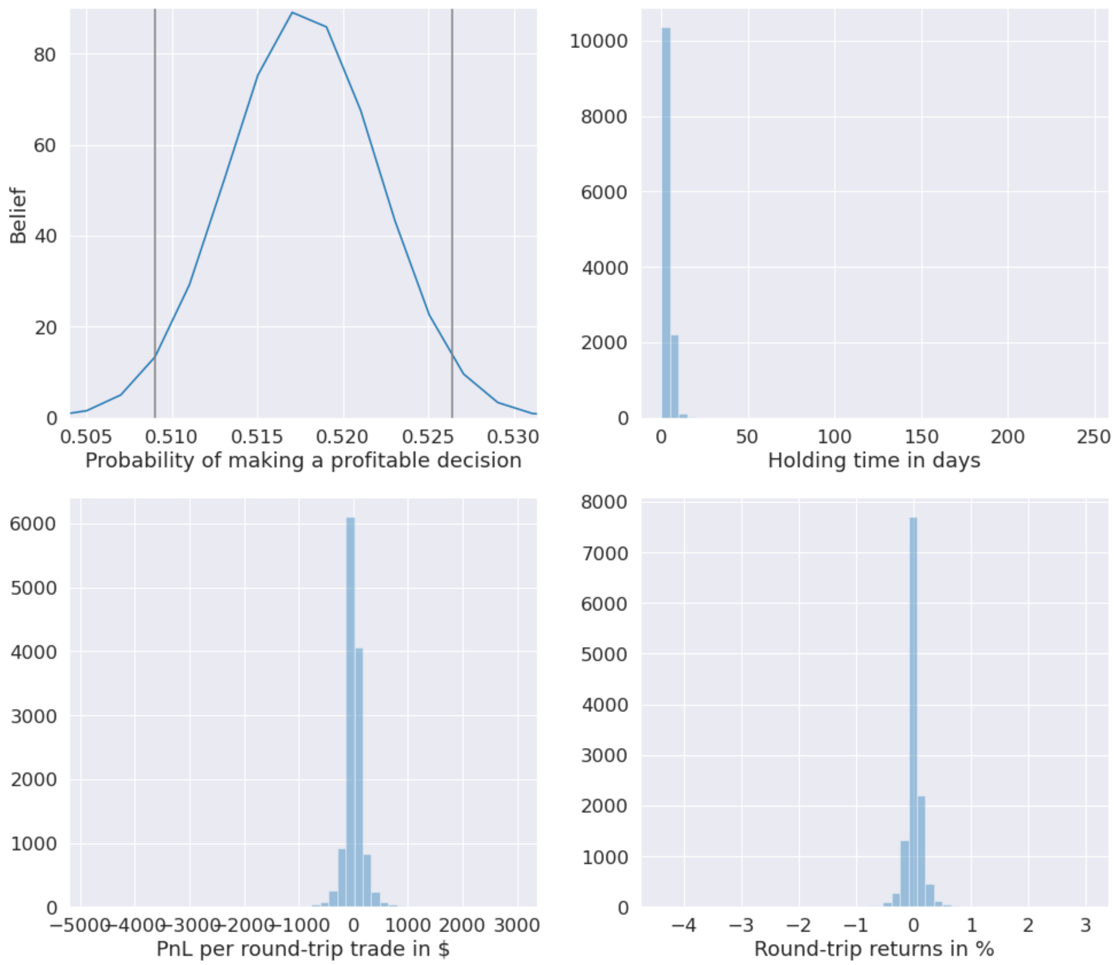

Fall2015
Returns
Algo
benchmark
0.025
0.000
−0.025
−0.050
−0.075
−0.100
−0.125
−0.150
2015-08-15
2015-08-22
2015-09-01
2015-09-08
2015-09-15
2015-09-22
2015-10-01


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


Exposure
Exposure
Net
Long
Short
Portfolio allocation over time, only top 10 holdings
Exposure by holding
DHR
STWD
CLX
ANAT
TWO
RNR
ACGL
AIV
MAA
TCO
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: 53, min: 5)
Short (max: 0, min: 0)
Overlap
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
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
Daily trading volume
Amount of shares traded
16000
14000
12000
10000
8000
6000
4000
2000
0
Distribution of daily turnover rates
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
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



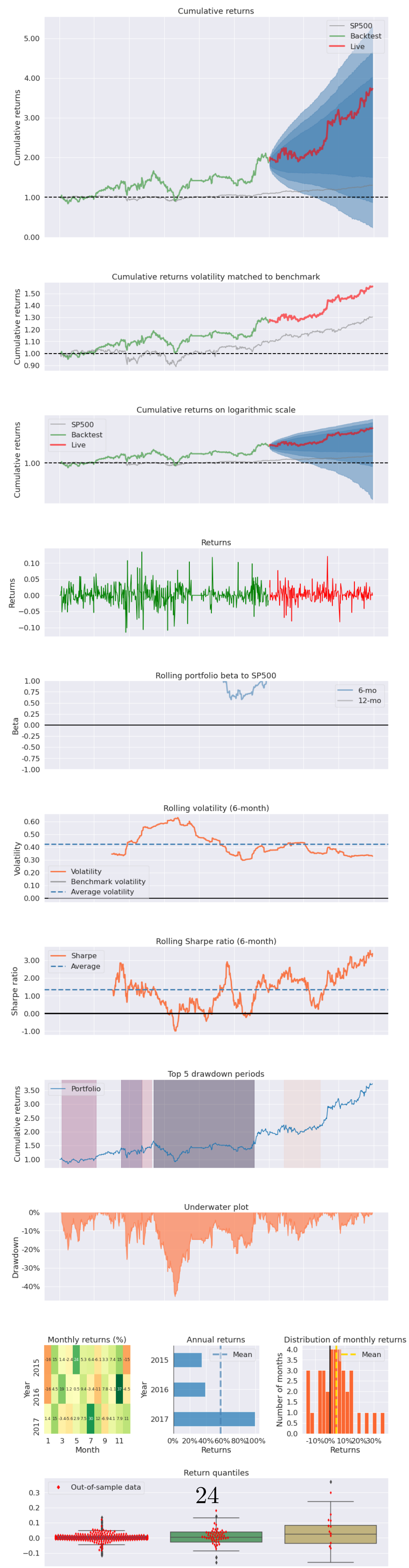


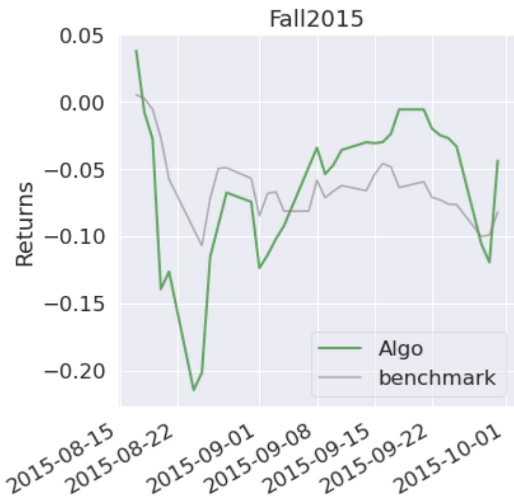
Markowitz

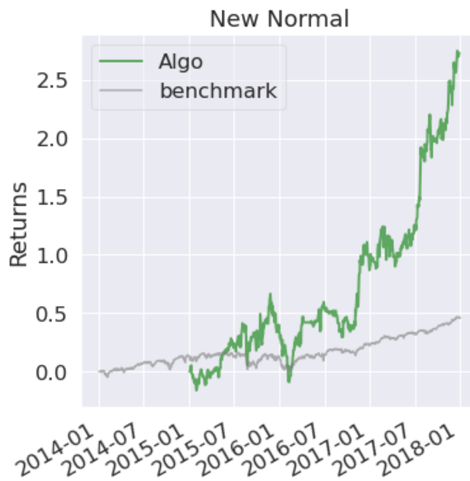
[82]:
```
experiment = 'markowitz'
returns, positions, transactions = load_results(experiment)
```

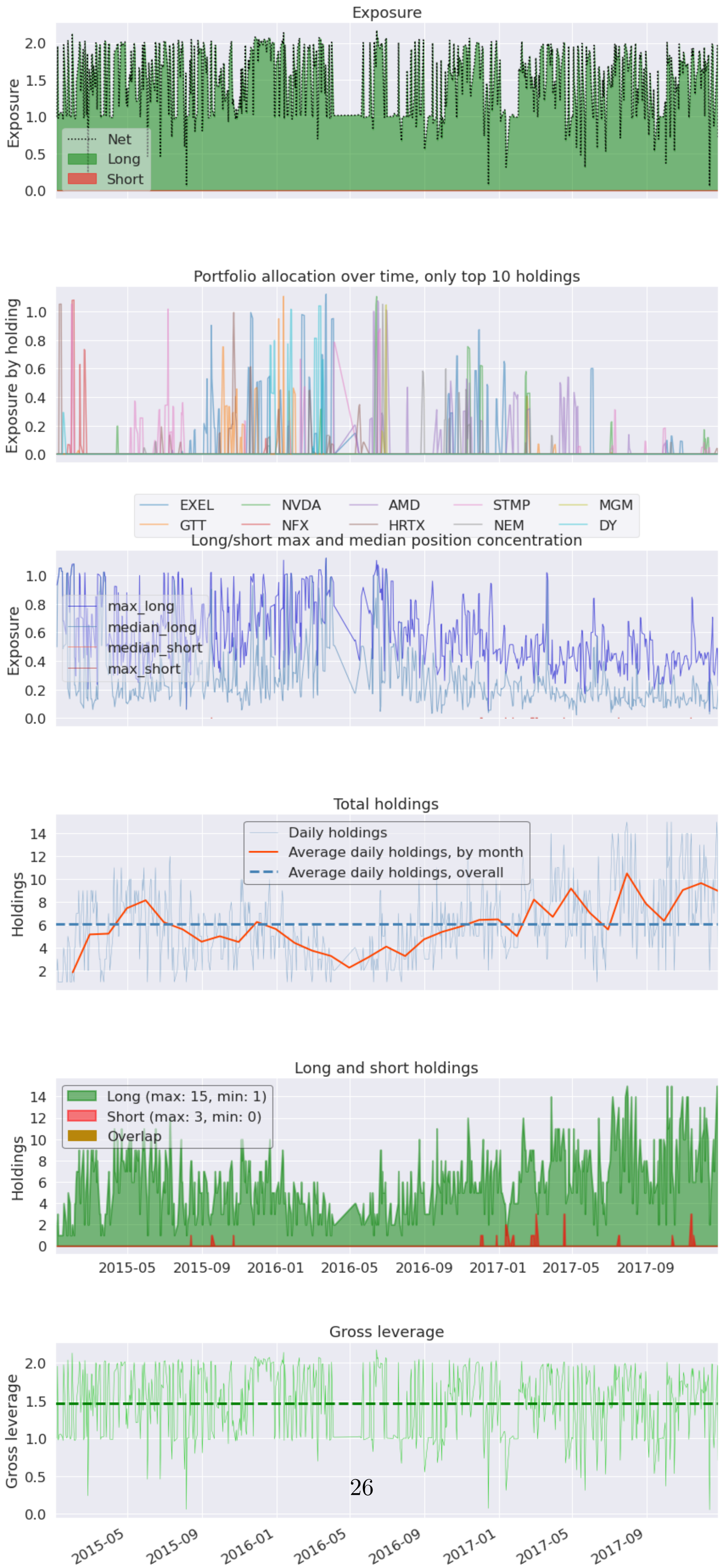
```python
pf.create_full_tear_sheet(returns,
                          positions=positions,
                          transactions=transactions,
                          benchmark_rets=benchmark,
                          live_start_date='2017-01-01',
                          round_trips=True)
```

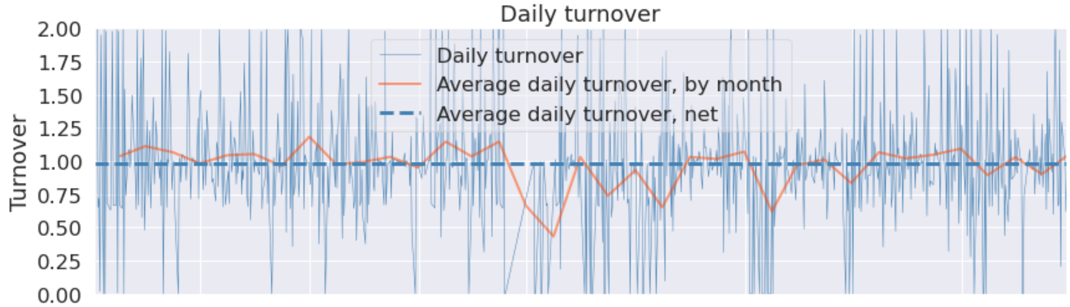
```
<IPython.core.display.HTML object>

<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>
```

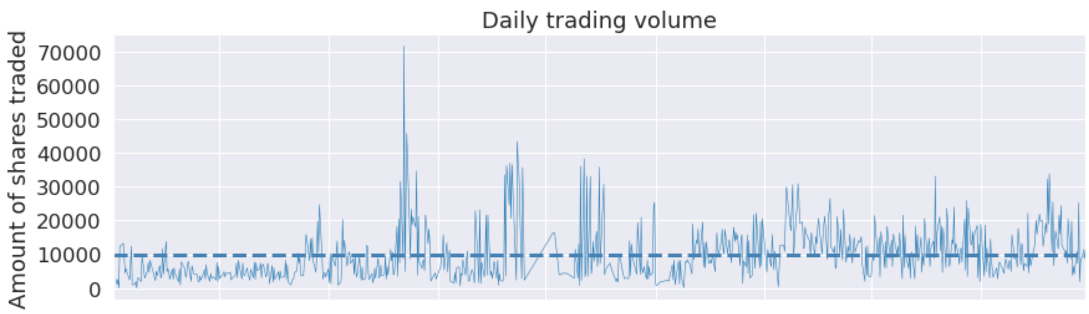

Cumulative returns
SP500
Backtest
Live
Cumulative returns
Cumulative returns volatility matched to benchmark
Cumulative returns on logarithmic scale
Returns
Rolling portfolio beta to SP500
6-mo
12-mo
Beta
Rolling volatility (6-month)
Volatility
Benchmark volatility
Average volatility
Rolling Sharpe ratio (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

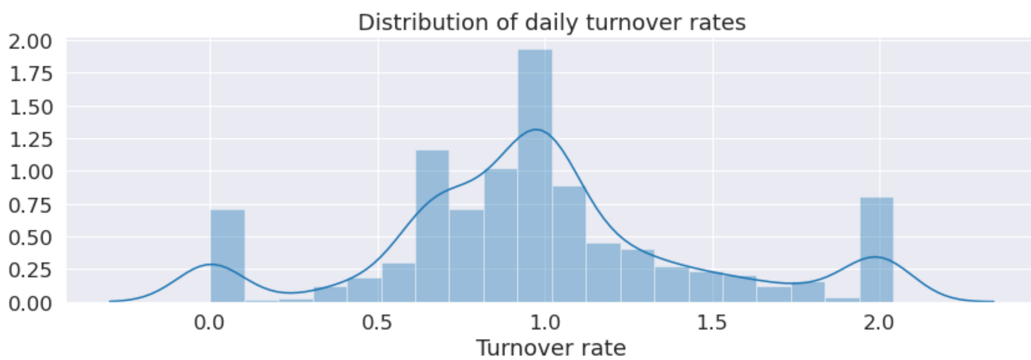

Fall2015
Returns
Algo
benchmark
0.05
0.00
−0.05
−0.10
−0.15
−0.20
2015-08-15
2015-08-22
2015-09-01
2015-09-08
2015-09-15
2015-09-22
2015-10-01

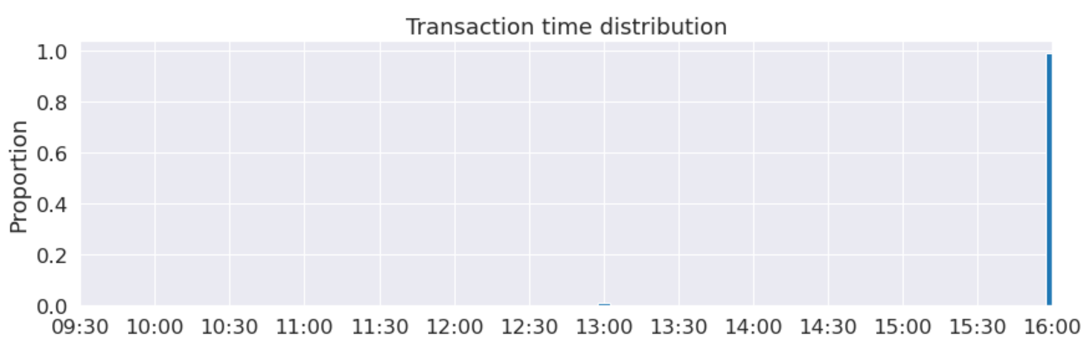

New Normal
Returns
Algo
benchmark
2.5
2.0
1.5
1.0
0.5
0.0
2014-01
2014-07
2015-01
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01

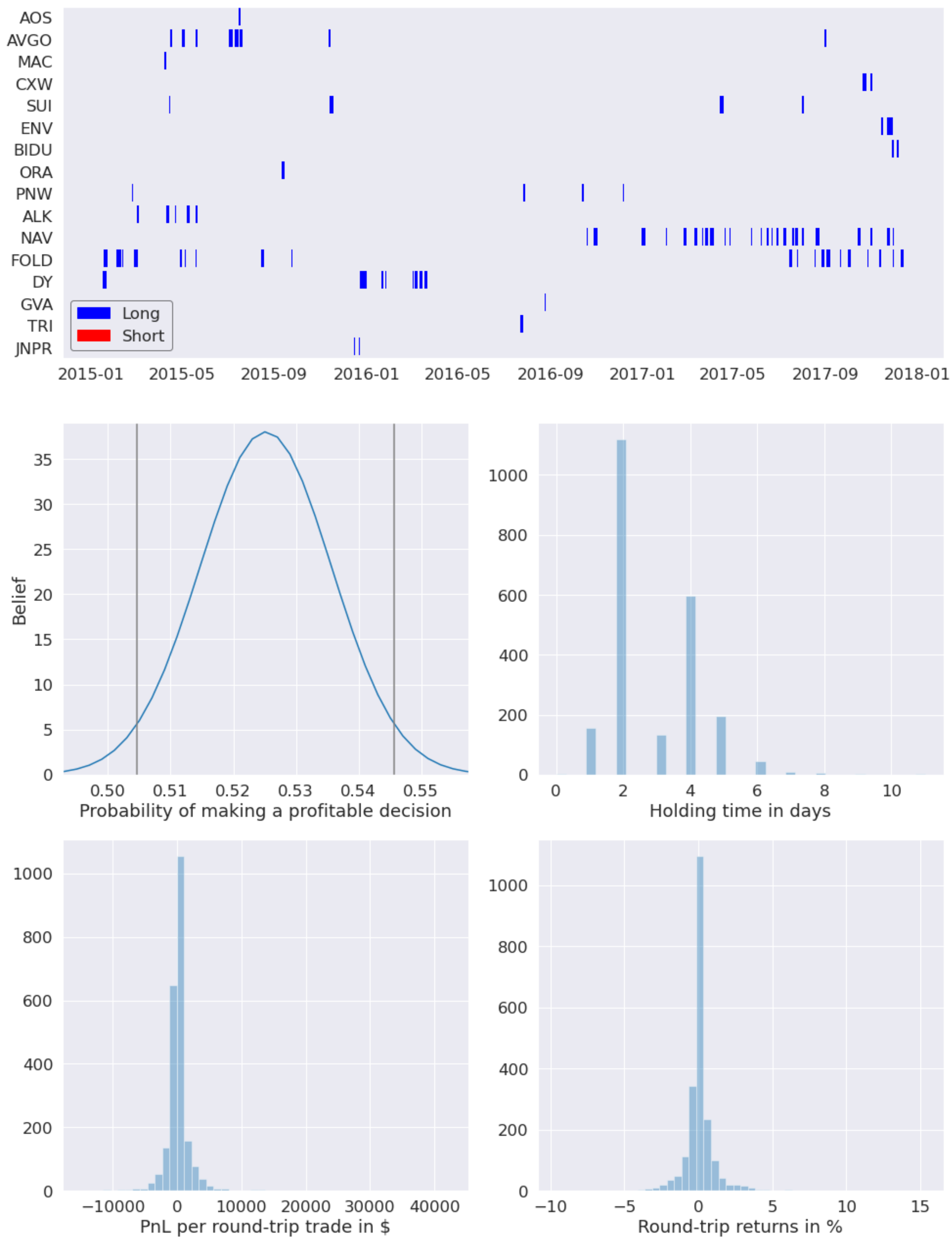

Exposure
Exposure
2.0
1.5
1.0
0.5
0.0
Net
Long
Short
Portfolio allocation over time, only top 10 holdings
Exposure by holding
1.0
0.8
0.6
0.4
0.2
0.0
EXEL
GTT
NVDA
NFX
AMD
HRTX
STMP
NEM
MGM
DY
Long/short max and median position concentration
Exposure
1.0
0.8
0.6
0.4
0.2
0.0
max_long
median_long
median_short
max_short
Total holdings
Holdings
14
12
10
8
6
4
2
Daily holdings
Average daily holdings, by month
Average daily holdings, overall
Long and short holdings
Holdings
14
12
10
8
6
4
2
0
Long (max: 15, min: 1)
Short (max: 3, min: 0)
Overlap
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
Gross leverage
Gross leverage
2.0
1.5
1.0
0.5
0.0
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09


Daily turnover
Turnover
Daily turnover
Average daily turnover, by month
Average daily turnover, net
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00


Daily trading volume
Amount of shares traded
70000
60000
50000
40000
30000
20000
10000
0


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


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


AOS
AVGO
MAC
CXW
SUI
ENV
BIDU
ORA
PNW
ALK
NAV
FOLD
DY
GVA
TRI
JNPR
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 %