ML4T Workflow with zipline

We can also integrate the model training into our backtest. The goal is to replicate the daily return predictions we used in backtesting_with_zipline and generated in Chapter 7.

We will, however, use a few additional Pipeline factors to illustrate their usage. The principal new element is a CustomFactor that receives features and returns as inputs to train a model and produce predictions. We follow the workflow displayed in the following figure:

Imports and Settings

Imports

In [38]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [39]:
```
from collections import OrderedDict
from time import time

import numpy as np
import pandas as pd
import pandas_datareader.data as web
#from zipline_binance import create_bundle
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.stats import spearmanr

from logbook import Logger, StderrHandler, INFO

from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error

from zipline.pipeline import Pipeline, CustomFactor
from zipline.pipeline.data import USEquityPricing
from zipline.pipeline.factors import AverageDollarVolume, EWMA, Returns
from zipline.pipeline.factors.technical import RSI, MACDSignal, TrueRange
from zipline import run_algorithm
from zipline.api import (attach_pipeline, pipeline_output,
                         date_rules, time_rules,
                         schedule_function, commission, slippage,
                         set_slippage, set_commission,
                         record,
                         order_target, order_target_percent)

import pyfolio as pf
```

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

In [40]:
```
sns.set_style('whitegrid')
```

Logging Setup

In [41]:
```
log_handler = StderrHandler(
        format_string='[{record.time:%Y-%m-%d %H:%M:%S.%f}]: ' +
                      '{record.level_name}: {record.func_name}: {record.message}',
        level=INFO
)
log_handler.push_application()
log = Logger('Algorithm')
```

Algo Parameters

In [42]:
```
# Target long/short positions
N_LONGS = 25
N_SHORTS = 25
MIN_POSITIONS = 15

UNIVERSE = 250

# Length of the training period (memory-intensive due to pre-processing)
TRAINING_PERIOD = 252 * 2

# train to predict N day forward returns
N_FORWARD_DAYS = 1

# How often to trade; align with prediction horizon
# for weekly, set to date_rules.week_start(days_offset=1)
TRADE_FREQ = date_rules.every_day()
```

In [43]:
```
import pytz
import pandas as pd

start = pd.Timestamp('2018-01-01', tz='utc')
end = pd.Timestamp('2024-03-12', tz='utc')

# Convert start and end timestamps to naive UTC timestamps
start_utc = start.replace(tzinfo=None)
end_utc = end.replace(tzinfo=None)
```

In [44]:
```
!zipline bundles
```

```
binance_daily <no ingestions>
binance_min <no ingestions>
csvdir <no ingestions>
forex-bundle <no ingestions>
iex <no ingestions>
quandl 2024-03-17 17:17:17.268185
quantopian-quandl <no ingestions>
yahoo_csv 2024-03-01 21:33:22.578122
yahoo_csv 2024-03-01 21:29:10.348857
yahoo_csv 2024-03-01 21:26:22.343593
yahoo_direct 2024-03-17 18:00:30.409248
yahoo_direct 2024-03-01 21:10:08.538342
yahoo_direct 2024-03-01 20:58:45.718941
yahoo_direct 2024-03-01 20:55:20.888197
```

```
C:\Users\srika\anaconda3\envs\king\lib\site-packages\pandas\core\computation\expr
essions.py:21: UserWarning: Pandas requires version '2.8.4' or newer of 'numexpr'
(version '2.7.3' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
C:\Users\srika\anaconda3\envs\king\lib\site-packages\exchange_calendars\exchange_
calendar.py:2345: FutureWarning: 'T' is deprecated and will be removed in a futur
e version. Please use 'min' instead of 'T'.
  align: pd.Timedelta | str = pd.Timedelta(1, "T"),
```

Factor Engineering

To create a Pipeline Factor, we need one or more input variables, a window_length that indicates the number of most recent data points for each input and security, and the computation we want to conduct.

We will use ten custom and built-in factors as features for our model to capture risk factors like momentum and volatility. Next, we'll come up with a CustomFactor that trains our model.

Momentum

In [45]:
```
def Price_Momentum_3M():
    return Returns(window_length=63)
```

In [46]:
```
def Returns_39W():
    return Returns(window_length=215)
```

Volatility

In [47]:
```
class Vol_3M(CustomFactor):
    # 3 months volatility
    inputs = [Returns(window_length=2)]
    window_length = 63

    def compute(self, today, assets, out, rets):
        out[:] = np.nanstd(rets, axis=0)
```

Mean Reversion

In [48]:
```
class Mean_Reversion_1M(CustomFactor):
    # standardized difference between latest monthly return
    # and their annual average
    inputs = [Returns(window_length=21)]
    window_length = 252

    def compute(self, today, assets, out, monthly_rets):
        out[:] = (monthly_rets[-1] - np.nanmean(monthly_rets, axis=0)) / \
                 np.nanstd(monthly_rets, axis=0)
```

Money Flow Volume

In [49]:
```
class Moneyflow_Volume_5d(CustomFactor):
    inputs = [USEquityPricing.close, USEquityPricing.volume]
    window_length = 5

    def compute(self, today, assets, out, close, volume):

        mfvs = []

        for col_c, col_v in zip(close.T, volume.T):

            # denominator
            denominator = np.dot(col_c, col_v)

            # numerator
            numerator = 0.
            for n, price in enumerate(col_c.tolist()):
                if price > col_c[n - 1]:
                    numerator += price * col_v[n]
                else:
                    numerator -= price * col_v[n]

            mfvs.append(numerator / denominator)
        out[:] = mfvs
```

Price Trend

A linear price trend that we estimate using linear regression (see Chapter 7 works as follows: we use the 252 latest close prices to compute the regression coefficient on a linear time trend:

In [50]:
```
class Trendline(CustomFactor):
    # linear 12 month price trend regression
    inputs = [USEquityPricing.close]
    window_length = 252

    def compute(self, today, assets, out, close):
        X = np.arange(self.window_length).reshape(-1, 1).astype(float)
        X -= X.mean()
        Y = close - np.nanmean(close, axis=0)
        out[:] = (X.T @ Y / np.var(X)) / self.window_length
```

Price Oscillator

In [51]:
```
class Price_Oscillator(CustomFactor):
    inputs = [USEquityPricing.close]
    window_length = 252

    def compute(self, today, assets, out, close):
        four_week_period = close[-20:]
        out[:] = (np.nanmean(four_week_period, axis=0) /
                  np.nanmean(close, axis=0)) - 1.
```

Combine Features

In [52]:
```
vol_3M = Vol_3M()
mean_reversion_1M = Mean_Reversion_1M()
macd_signal_10d = MACDSignal()
moneyflow_volume_5d = Moneyflow_Volume_5d()
trendline = Trendline()
price_oscillator = Price_Oscillator()
price_momentum_3M = Price_Momentum_3M()
returns_39W = Returns_39W()
true_range = TrueRange()
```

In [53]:
```
features = {
    'Vol 3M'              : vol_3M,
    'Mean Reversion 1M'   : mean_reversion_1M,
    'MACD Signal 10d'     : macd_signal_10d,
    'Moneyflow Volume 5D': moneyflow_volume_5d,
    'Trendline'           : trendline,
    'Price Oscillator'    : price_oscillator,
    'Price Momentum 3M'   : price_momentum_3M,
    '39 Week Returns'     : returns_39W,
    'True Range'          : true_range
}
```

ML CustomFactor

Our `CustomFactor` called LinearModel will have the `StandardScaler` and a stochastic gradient descent (SGD) implementation of ridge regression as instance attributes, and we will train the model on three days a week.

- The `compute` method generates predictions (addressing potential missing values), but first checks if the model should be trained.
- The `_train_model` method is the centerpiece of the puzzle. It shifts the returns and aligns the resulting forward returns with the Factor features, removing missing values in the process. It scales the remaining data points and trains the linear SGDRegressor.

In [54]:
```
class LinearModel(CustomFactor):
    """Obtain model predictions"""
    train_on_weekday = [0, 2, 4]

    def __init__(self, *args, **kwargs):
        super().__init__(self, *args, **kwargs)
```

```
        self._scaler = StandardScaler()
        self._model = SGDRegressor(penalty='l2')
        self._trained = False

    def _train_model(self, today, returns, inputs):

        scaler = self._scaler
        model = self._model

        shift_by = N_FORWARD_DAYS + 1
        outcome = returns[shift_by:].flatten()
        features = np.dstack(inputs)[:-shift_by]
        n_days, n_stocks, n_features = features.shape
        features = features.reshape(-1, n_features)
        features = features[~np.isnan(outcome)]
        outcome = outcome[~np.isnan(outcome)]
        outcome = outcome[np.all(~np.isnan(features), axis=1)]
        features = features[np.all(~np.isnan(features), axis=1)]
        features = scaler.fit_transform(features)

        start = time()
        model.fit(X=features, y=outcome)
#         log.info('{} | {:.2f}s'.format(today.date(), time() - start))
        self._trained = True

    def _maybe_train_model(self, today, returns, inputs):
        if (today.weekday() in self.train_on_weekday) or not self._trained:
            self._train_model(today, returns, inputs)

    def compute(self, today, assets, out, returns, *inputs):
        self._maybe_train_model(today, returns, inputs)

        # Predict most recent feature values
        X = np.dstack(inputs)[-1]
        missing = np.any(np.isnan(X), axis=1)
        X[missing, :] = 0
        X = self._scaler.transform(X)
        preds = self._model.predict(X)
        out[:] = np.where(missing, np.nan, preds)
```

Pipeline

The `make_ml_pipeline()` function preprocesses and combines the outcome, feature, and model parts into a Pipeline with a column for predictions.

In [55]:
```
def make_ml_pipeline(universe, window_length=21, n_forward_days=5):
    pipeline_columns = OrderedDict()

    # ensure that returns is the first input
    pipeline_columns['Returns'] = Returns(inputs=[USEquityPricing.open],
                                          mask=universe,
                                          window_length=n_forward_days + 1)

    # convert factors to ranks; append to pipeline
    pipeline_columns.update({k: v.rank(mask=universe)
                             for k, v in features.items()})

    # Create ML pipeline factor.
```

```
    # window_length = length of the training period
    pipeline_columns['predictions'] = LinearModel(inputs=pipeline_columns.values
                                          window_length=window_length + n_forward
                                          mask=universe)

    return Pipeline(screen=universe, columns=pipeline_columns)
```

Universe

In [56]:
```
def make_universe():
    # Set screen
    dollar_volume = AverageDollarVolume(window_length=90)
    return dollar_volume.top(UNIVERSE)
```

In [57]:
```
universe = make_universe()
```

Initialize Algorithm

In [58]:
```
def initialize(context):
    """
    Called once at the start of the algorithm.
    """

    set_slippage(slippage.FixedSlippage(spread=0.00))
    set_commission(commission.PerShare(cost=0, min_trade_cost=0))

    # Schedule rebalance and record functions
    schedule_function(rebalance, date_rules.every_day(), time_rules.market_open(
    schedule_function(record_vars, date_rules.every_day(), time_rules.market_clo

    # Create machine learning pipeline
    ml_pipeline = make_ml_pipeline(universe, N_FORWARD_DAYS, TRAINING_PERIOD)
    attach_pipeline(ml_pipeline, 'ml_model')

    # Initialize context variables
    context.past_predictions = {}
    context.ic = 0
    context.rmse = 0
    context.mae = 0
    context.returns_spread_bps = 0
```

Evaluate Predictive Accuracy

The `evaluate_predictions()` function does exactly this: it tracks the past predictions of our model and evaluates them once returns for the relevant time horizon materialize (in our example the next day):

In [59]:
```
def evaluate_predictions(output, context):
    # Look at past predictions to evaluate model performance
    # A day has passed, shift days and drop old ones
    context.past_predictions = {
        k - 1: v for k, v in context.past_predictions.items() if k > 0
    }
```

```
    if 0 in context.past_predictions:
        # Use today's n-day returns to evaluate predictions

        returns, predictions = (output['Returns'].dropna()
                                .align(context.past_predictions[0].dropna(),
                                       join='inner'))
        if len(returns) > 0 and len(predictions) > 0:
            context.ic = spearmanr(returns, predictions)[0]
            context.rmse = np.sqrt(
                mean_squared_error(returns, predictions))
            context.mae = mean_absolute_error(returns, predictions)

            long_rets = returns[predictions > 0].mean()
            short_rets = returns[predictions < 0].mean()
            context.returns_spread_bps = (long_rets - short_rets) * 10000

    # Store current predictions
    context.past_predictions[N_FORWARD_DAYS] = context.predicted_returns
```

Get Pipeline Output

We obtain new predictions using the `before_trading_start()` function that runs every morning before market open:

In [60]:
```
def before_trading_start(context, data):
    """
    Called every day before market open.
    """
    output = pipeline_output('ml_model')
    context.predicted_returns = output['predictions']
    #context.predicted_returns.index.rename(['date', 'equity'], inplace=True)
    context.predicted_returns.index.rename('date_equity', inplace=True)


    evaluate_predictions(output, context)

    # These are the securities that we are interested in trading each day.
    context.security_list = context.predicted_returns.index
```

Rebalance

In [61]:
```
def rebalance(context, data):
    """
    Execute orders according to our schedule_function() timing.
    """
    predictions = context.predicted_returns

    # Drop stocks that can not be traded
    predictions = predictions.loc[data.can_trade(predictions.index)]
    longs = (predictions[predictions > 0]
             .sort_values(ascending=False)[:N_LONGS]
             .index
             .tolist())
    shorts = (predictions[predictions < 0]
              .sort_values()[:N_SHORTS]
```

```
            .index
            .tolist())
    targets = set(longs + shorts)
    for position in context.portfolio.positions:
        if position not in targets:
            order_target(position, 0)

    n_longs, n_shorts = len(longs), len(shorts)
    if n_longs > MIN_POSITIONS and n_shorts > MIN_POSITIONS:
        for stock in longs:
            order_target_percent(stock, target=1/n_longs)
        for stock in shorts:
            order_target_percent(stock, target=-1/n_shorts)
    else:
        for stock in targets:
            if stock in context.portfolio.positions:
                order_target(stock, 0)
```

Logging

In [62]:
```
def record_vars(context, data):
    """
    Plot variables at the end of each day.
    """
    record(
            leverage=context.account.leverage,
            ic=context.ic,
            rmse=context.rmse,
            mae=context.mae,
            returns_spread_bps=context.returns_spread_bps
    )
```

Run Algo

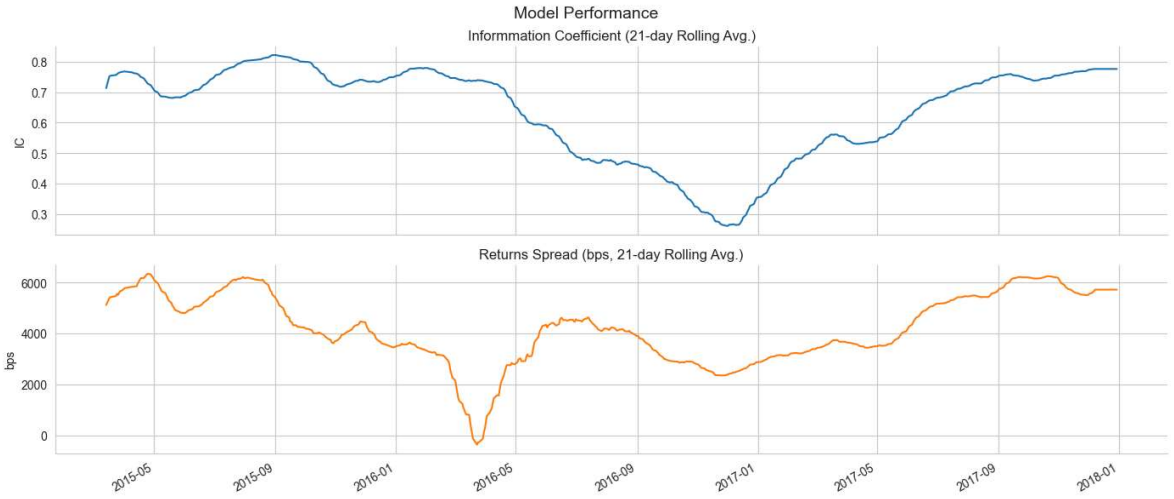
In [ ]:
```
go = time()
results = run_algorithm(start=start_utc,
                        end=end_utc,
                        initialize=initialize,
                        before_trading_start=before_trading_start,
                        capital_base=1e6,
                        data_frequency='daily',
                        bundle='quandl')

print('{:.2f}'.format(time()-go))
```

Analyze using PyFolio

In [28]:
```
axes = (results[['ic', 'returns_spread_bps']]
        .dropna()
        .rolling(21)
        .mean()
        .plot(subplots=True,
              layout=(2,1),
              figsize=(14, 6),
              title=['Informmation Coefficient (21-day Rolling Avg.)', 'Returns
```

```
                legend=False))
axes = axes.flatten()
axes[0].set_ylabel('IC')
axes[1].set_ylabel('bps')
plt.suptitle('Model Performance', fontsize=14)
sns.despine()
plt.tight_layout()
plt.subplots_adjust(top=.9);
```



Get PyFolio Input

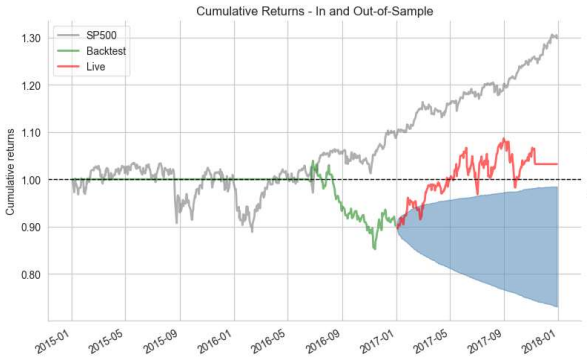
In [29]:
```
returns, positions, transactions = pf.utils.extract_rets_pos_txn_from_zipline(re
```

Get Benchmark Data

In [30]:
```
benchmark = web.DataReader('SP500', 'fred', '2014', '2018').squeeze()
benchmark = benchmark.pct_change().tz_localize('UTC')
```

Custom Performance Plots

In [31]:
```
fig, axes = plt.subplots(ncols=2, figsize=(16, 5))
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])
plot_rolling_sharpe(returns, ax=axes[1])
axes[0].set_title('Cumulative Returns - In and Out-of-Sample')
sns.despine()
fig.tight_layout();
```



Full Tearsheet

In [32]:
```
pf.create_full_tear_sheet(returns,
                          positions=positions,
                          transactions=transactions,
                          benchmark_rets=benchmark,
                          live_start_date='2017-01-01',
                          round_trips=True)
```

Start date	2015-01-02
End date	2017-12-29
In-sample months	24
Out-of-sample months	11

	In-sample	Out-of-sample	All
Annual return	-5.047%	14.546%	1.063%
Cumulative returns	-9.839%	14.484%	3.22%
Annual volatility	7.36%	14.808%	10.449%
Sharpe ratio	-0.67	0.99	0.15
Calmar ratio	-0.28	1.51	0.06
Stability	0.41	0.60	0.00
Max drawdown	-18.042%	-9.612%	-18.042%
Omega ratio	0.80	1.20	1.04
Sortino ratio	-0.94	1.34	0.21
Skew	0.95	-0.76	-0.38
Kurtosis	18.79	2.73	7.98
Tail ratio	0.64	0.74	0.86
Daily value at risk	-0.947%	-1.807%	-1.31%
Gross leverage	2.00	2.00	2.00
Daily turnover	37.261%	33.322%	34.73%
Alpha	-0.05	0.06	0.01
Beta	-0.03	0.47	0.02

Worst drawdown periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	18.04	2016-06-27	2016-11-14	2017-05-24	238
1	9.61	2017-08-31	2017-09-26	NaT	NaN
2	9.25	2017-06-07	2017-07-03	2017-08-21	54
3	1.13	2017-08-24	2017-08-25	2017-08-29	4
4	0.78	2017-05-31	2017-06-01	2017-06-02	3

Stress Events	mean	min	max
Fall2015	0.00%	0.00%	0.00%
New Normal	0.01%	-3.72%	4.07%

Top 10 long positions of all time	max
sid	
SBAC	8.42%
MAR	8.08%
RCL	8.05%
CAT	8.00%
NVDA	5.19%
SRPT	4.82%
X	4.78%
ADSK	4.59%
NFLX	4.49%
ULTA	4.46%

Top 10 short positions of all time	max
sid	
ORLY	-8.32%
ULTA	-8.28%
ESRX	-8.24%
NWL	-8.08%
CHK	-6.52%
WLL	-6.43%
RIG	-6.27%
WDC	-6.16%
DAL	-6.05%
KMI	-6.02%

Top 10 positions of all time sid	max
SBAC	8.42%
ORLY	8.32%
ULTA	8.28%
ESRX	8.24%
MAR	8.08%
NWL	8.08%
RCL	8.05%
CAT	8.00%
CHK	6.52%
WLL	6.43%

Summary stats	All trades	Short trades	Long trades
Total number of round_trips	8704.00	4728.00	3976.00
Percent profitable	0.55	0.46	0.66
Winning round_trips	4798.00	2193.00	2605.00
Losing round_trips	3881.00	2518.00	1363.00
Even round_trips	25.00	17.00	8.00

PnL stats	All trades	Short trades	Long trades
Total profit	$2624.99	$-199252.90	$201877.89
Gross profit	$1460860.52	$752889.72	$707970.80
Gross loss	$-1458235.53	$-952142.62	$-506092.91
Profit factor	$1.00	$0.79	$1.40
Avg. trade net profit	$0.30	$-42.14	$50.77
Avg. winning trade	$304.47	$343.31	$271.77
Avg. losing trade	$-375.74	$-378.13	$-371.31
Ratio Avg. Win:Avg. Loss	$0.81	$0.91	$0.73
Largest winning trade	$14587.13	$13203.82	$14587.13
Largest losing trade	$-16378.17	$-15669.29	$-16378.17

Duration stats	All trades	Short trades	Long trades
Avg duration	24 days 00:29:04.163602941	17 days 22:59:38.680203045	31 days 05:29:31.931589537
Median duration	7 days 00:30:00	7 days 00:00:00	9 days 00:00:00
Longest duration	277 days 00:00:00	145 days 23:00:00	277 days 00:00:00
Shortest duration	1 days 00:00:00	1 days 00:00:00	1 days 00:00:00

Return stats	All trades	Short trades	Long trades
Avg returns all round_trips	-0.00%	-0.00%	0.00%
Avg returns winning	0.03%	0.03%	0.02%
Avg returns losing	-0.04%	-0.04%	-0.04%
Median returns all round_trips	0.00%	-0.00%	0.00%
Median returns winning	0.00%	0.00%	0.00%
Median returns losing	-0.00%	-0.00%	-0.00%
Largest winning trade	1.38%	1.38%	0.67%
Largest losing trade	-1.58%	-1.52%	-1.58%

Symbol stats	AA	AAL	AAOI	AAP	AAPL	ABBV	ABC	ABT	ADBE	A
Avg returns all round_trips	-0.01%	0.09%	-0.06%	0.01%	-0.01%	0.00%	0.01%	-0.02%	0.01%	-0.0
Avg returns winning	NaN	0.12%	0.05%	0.04%	0.02%	0.03%	0.06%	0.04%	0.02%	0.0
Avg returns losing	-0.01%	-0.10%	-0.14%	-0.01%	-0.05%	-0.05%	-0.04%	-0.05%	-0.02%	-0.0
Median returns all round_trips	-0.01%	0.03%	-0.00%	-0.00%	0.00%	0.00%	0.00%	-0.00%	0.00%	0.0
Median returns winning	NaN	0.05%	0.02%	0.01%	0.00%	0.00%	0.02%	0.01%	0.00%	0.0
Median returns losing	-0.01%	-0.10%	-0.01%	-0.00%	-0.03%	-0.00%	-0.00%	-0.00%	-0.00%	-0.0
Largest winning trade	-0.01%	0.47%	0.56%	0.20%	0.18%	0.13%	0.52%	0.22%	0.20%	0.0
Largest losing trade	-0.01%	-0.10%	-1.58%	-0.16%	-0.26%	-0.19%	-0.23%	-0.33%	-0.11%	-0.1

Profitability (PnL / PnL total) per name	
symbol	
NVDA	2364.14%
LRCX	1063.13%
X	790.18%
BA	661.60%
ADSK	653.26%
UAA	569.34%
CTL	552.08%
CHTR	534.23%
FL	532.44%
NWL	476.97%
NFLX	461.11%
M	454.73%
AMAT	409.80%
ALGN	399.64%
AVGO	390.71%
WLL	363.72%
AAP	359.21%
HPE	352.68%
CMI	325.49%
GE	323.61%
MU	253.16%
NBL	241.39%
URI	236.64%
AMZN	231.63%
COTY	227.73%
AAL	220.56%
ILMN	216.49%
ULTA	214.54%
CSX	214.31%
AIG	212.38%
LVS	208.50%
VIAB	176.59%

Profitability (PnL / PnL total) per name	
symbol	
PYPL	168.65%
PCLN	166.69%
UNH	162.51%
EW	162.44%
ANTM	152.78%
GILD	152.15%
SHW	147.19%
GWW	145.71%
QCOM	145.62%
ESRX	144.99%
GOOGL	141.83%
TMUS	139.94%
STT	138.39%
GOOG	137.27%
AZO	136.78%
MDVN	129.85%
MS	129.01%
MCHP	127.61%
RHT	123.37%
ADBE	119.79%
PNC	116.71%
TSLA	114.04%
TTWO	112.53%
CAT	108.12%
KR	101.70%
F	98.71%
NSC	93.04%
AMD	86.88%
FB	82.79%
SPG	82.61%
RRC	81.63%
EA	79.94%

Profitability (PnL / PnL total) per name	
symbol	
TDG	77.85%
MAR	75.74%
OXY	74.19%
SBAC	72.80%
GD	71.80%
COL	69.63%
ABC	69.33%
GS	69.18%
AGN	67.86%
CI	66.16%
FDX	65.60%
ATVI	64.26%
CFG	63.44%
TSN	61.35%
STZ	56.85%
BIIB	51.38%
LOW	51.23%
PRU	48.05%
BSX	46.78%
JPM	41.09%
EQT	40.62%
WDC	39.54%
ALB	37.93%
BCR	35.10%
EXC	34.87%
NOW	34.15%
SRPT	32.32%
VMC	32.12%
TWX	29.10%
LMT	28.20%
EQR	27.00%
NOC	26.52%

Profitability (PnL / PnL total) per name	
symbol	
TMO	25.44%
EOG	25.30%
GPS	19.96%
RTN	19.39%
VOD	19.27%
STI	18.82%
UNP	18.60%
AYI	18.19%
XOM	16.82%
YUM	12.09%
MA	11.40%
PSX	11.29%
SPGI	11.20%
HCA	10.44%
ITW	9.94%
EL	8.50%
AEP	7.61%
TGT	7.58%
NKE	7.35%
CELG	7.11%
DLTR	6.67%
MSFT	4.28%
TXN	4.10%
ABBV	3.42%
PNRA	2.53%
GIS	2.31%
VMW	0.49%
SYMC	0.37%
WWAV	-0.63%
FLT	-1.18%
JNJ	-1.43%
CBS	-1.51%

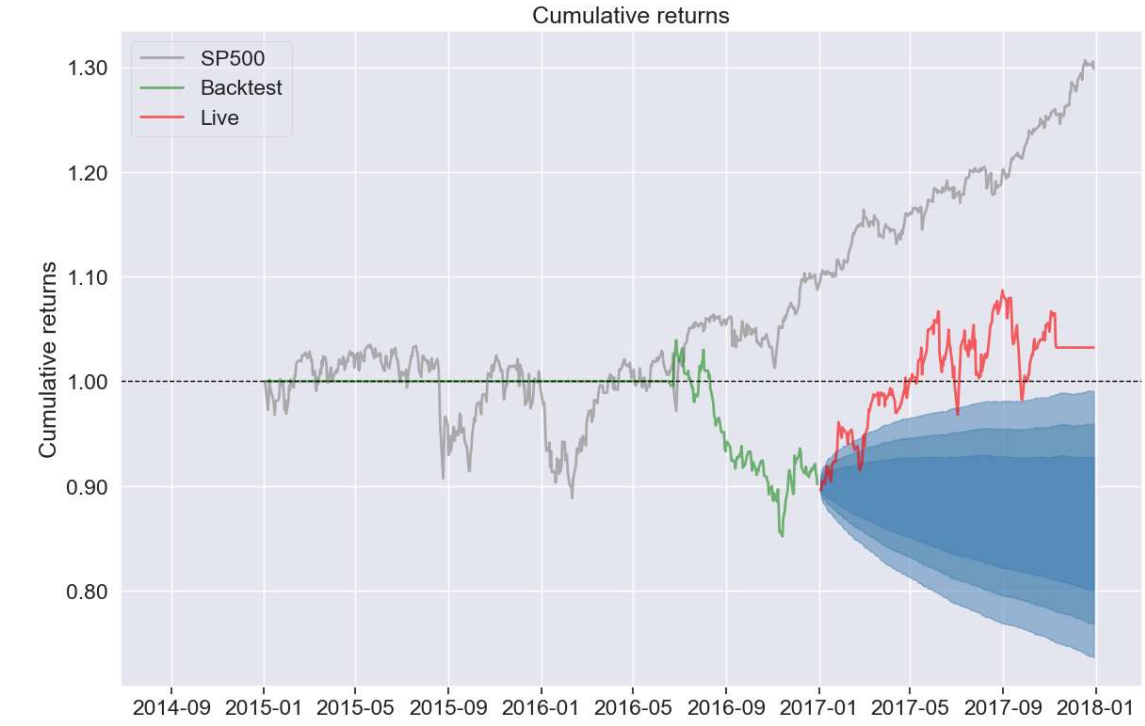
Profitability (PnL / PnL total) per name	
symbol	
TSRO	-2.56%
COST	-2.92%
MRK	-3.25%
ETN	-4.74%
HPQ	-5.03%
AA	-5.60%
LNKD	-7.20%
CAH	-7.34%
CVX	-7.99%
STJ	-8.03%
IBM	-8.70%
KEY	-8.79%
ARNC	-9.33%
CL	-11.80%
MJN	-12.29%
PH	-12.42%
DFS	-12.46%
TAP	-12.70%
MMM	-13.23%
MDT	-13.76%
CLF	-14.15%
YHOO	-15.43%
NEE	-15.81%
INTC	-16.18%
BAX	-17.48%
AMT	-18.27%
MON	-18.37%
PPL	-19.30%
APD	-20.76%
CF	-23.22%
SYY	-25.90%
ROST	-27.21%

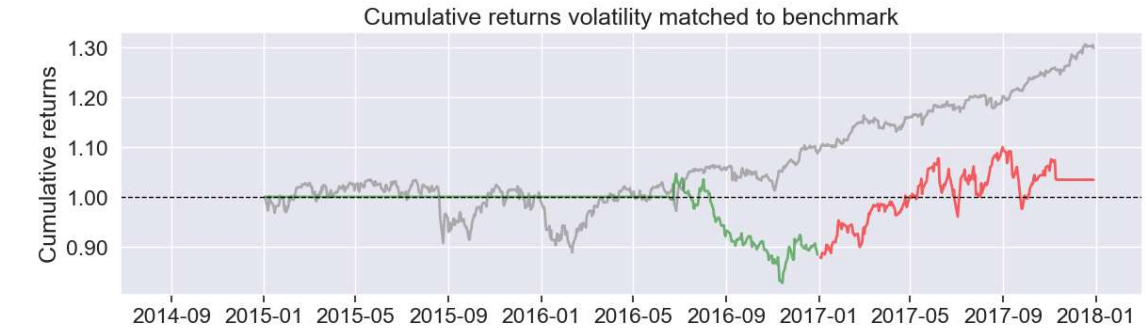
Profitability (PnL / PnL total) per name	
symbol	
TJX	-27.34%
KHC	-27.75%
DD	-31.57%
OMC	-32.65%
DIS	-32.98%
LYB	-34.67%
BHI	-35.42%
DLPH	-35.65%
MPC	-38.75%
MET	-41.37%
NUE	-41.85%
VZ	-42.08%
KORS	-42.36%
FOXA	-42.40%
MCD	-44.18%
HUM	-45.19%
WBA	-46.33%
SWN	-49.72%
WYNN	-52.97%
CLR	-53.81%
BLK	-54.46%
SE	-57.62%
DUK	-58.72%
ADI	-60.92%
KSS	-62.48%
NRG	-64.74%
PFE	-68.02%
WFM	-70.48%
TIF	-73.14%
PPG	-74.52%
K	-78.82%
ORCL	-79.62%

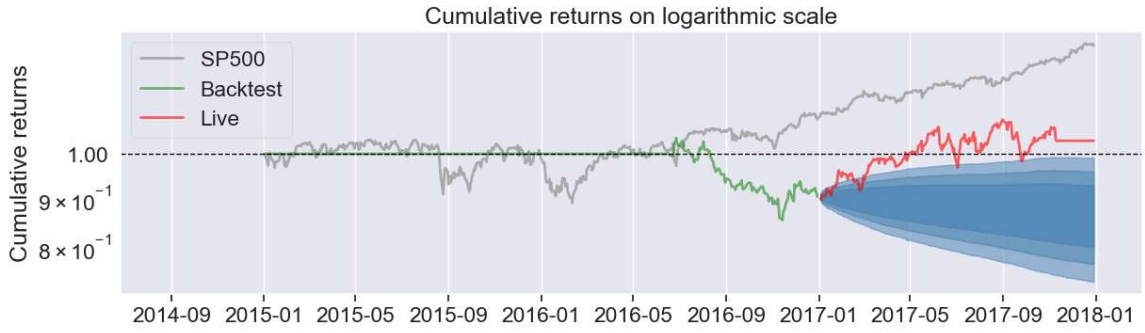
Profitability (PnL / PnL total) per name	
symbol	
PM	-82.41%
C	-82.57%
VRTX	-85.09%
WFC	-85.20%
AXP	-85.63%
T	-86.85%
KO	-90.71%
TSO	-93.74%
CME	-95.01%
APA	-97.37%
AAPL	-97.37%
DE	-97.59%
PANW	-101.36%
JCI	-103.74%
MDLZ	-106.20%
BBY	-107.20%
RIG	-113.73%
CCE	-114.56%
PXD	-115.69%
VFC	-119.87%
DAL	-120.67%
EXPE	-121.31%
CCL	-131.67%
AET	-133.33%
CHK	-133.82%
MLM	-134.13%
BAC	-135.52%
JWN	-138.69%
SBUX	-152.02%
LUV	-152.12%
APC	-166.17%
DVN	-168.48%

Profitability (PnL / PnL total) per name	
symbol	
KMI	-168.67%
SO	-169.64%
SIG	-170.77%
MYL	-182.74%
COP	-183.74%
PRGO	-191.72%
HSY	-195.12%
ADS	-201.27%
KMB	-201.80%
CTSH	-202.57%
PSA	-211.20%
MCK	-211.23%
DLR	-218.71%
VRX	-221.75%
EQIX	-226.88%
SWKS	-232.43%
SLB	-239.16%
CMG	-243.04%
SYF	-244.34%
LLY	-244.77%
VLO	-248.13%
NEM	-253.55%
RCL	-254.24%
ABT	-257.80%
UAL	-259.26%
FCX	-259.75%
HES	-268.53%
BMY	-271.96%
WMB	-279.94%
ZBH	-281.01%
CVS	-287.14%
HAL	-295.51%

Profitability (PnL / PnL total) per name	
symbol	
ORLY	-296.59%
CXO	-324.85%
REGN	-339.50%
ALXN	-359.36%
STX	-395.96%
ENDP	-434.56%
LB	-438.52%
MRO	-517.65%
DG	-632.48%
ISRG	-641.68%
INCY	-730.69%
AAOI	-903.35%
TWTR	-1542.83%


Cumulative returns
SP500
Backtest
Live
Cumulative returns
1.30
1.20
1.10
1.00
0.90
0.80
2014-09
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01


Cumulative returns volatility matched to benchmark
Cumulative returns
1.30
1.20
1.10
1.00
0.90
2014-09
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01

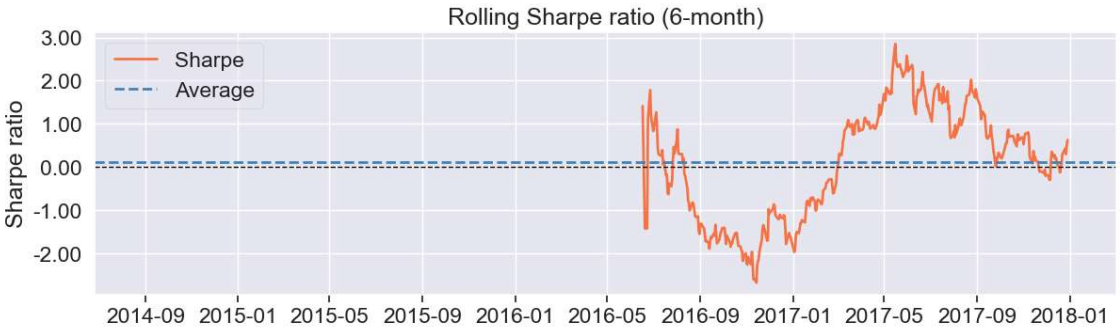

Cumulative returns on logarithmic scale
SP500
Backtest
Live
Cumulative returns
1.00
9×10^{-1}
8×10^{-1}
2014-09
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01

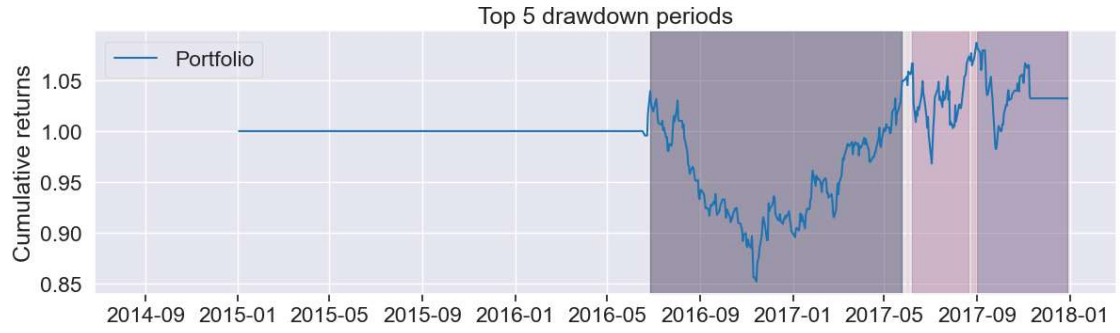

Returns
Returns
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04
2014-09
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01

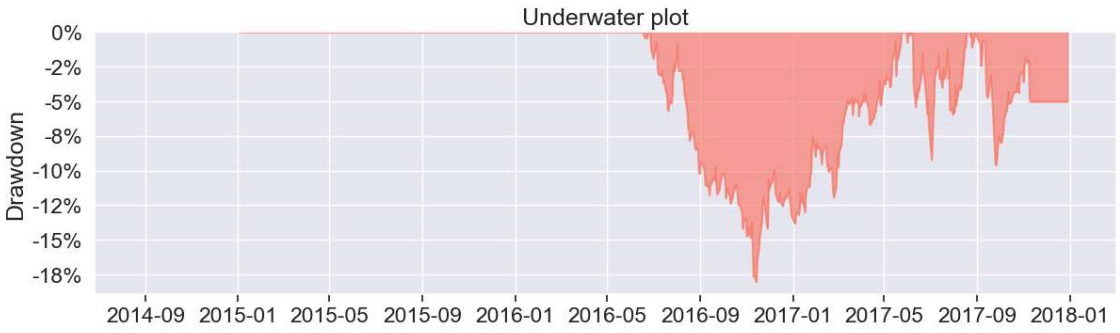

Rolling portfolio beta to SP500
1.00
6-mo
0.80

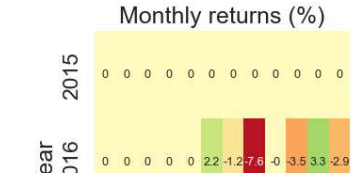

12-mo
6-mo Average
Beta
0.60
0.40
0.20
0.00
-0.20
2014-09
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01

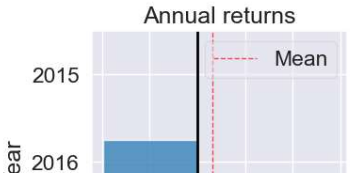

Rolling volatility (6-month)
Volatility
0.20
0.18
0.15
0.13
0.10
0.08
0.05
0.03
0.00
Volatility
Benchmark volatility
Average volatility
2014-09
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01

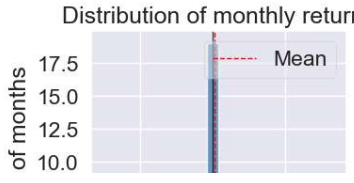

Rolling Sharpe ratio (6-month)
Sharpe ratio
3.00
2.00
1.00
0.00
-1.00
-2.00
Sharpe
Average
2014-09
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01

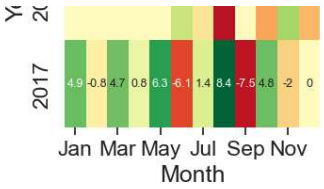

Top 5 drawdown periods
Cumulative returns
1.05
1.00
0.95
0.90
0.85
Portfolio
2014-09
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01

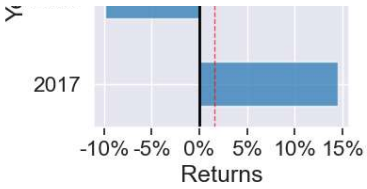

Underwater plot
Drawdown
0%
-2%
-5%
-8%
-10%
-12%
-15%
-18%
2014-09
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01

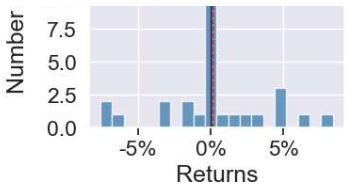

Monthly returns (%)
2015
2016
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 2.2 -1.2 -7.6 -0 -3.5 3.3 -2.9

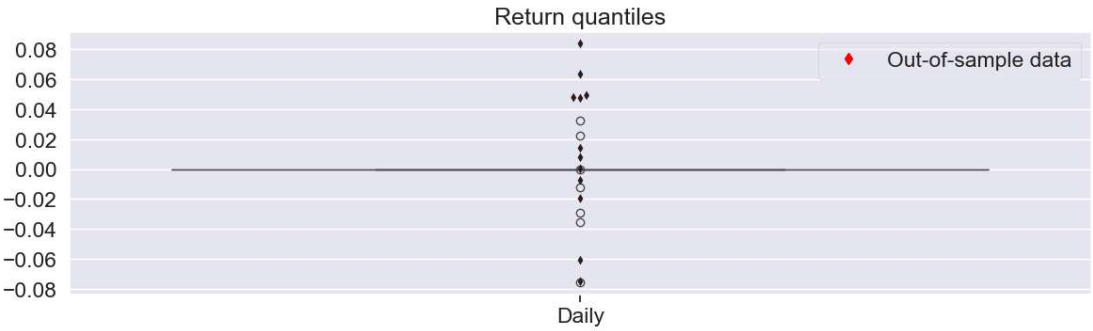

Annual returns
2015
2016
Mean

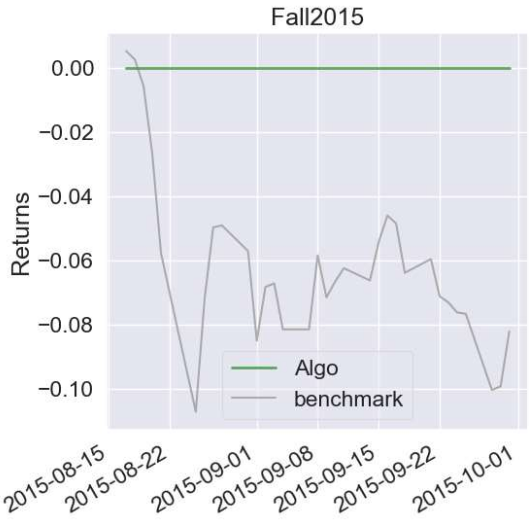

Distribution of monthly returns
of months
17.5
15.0
12.5
10.0
Mean

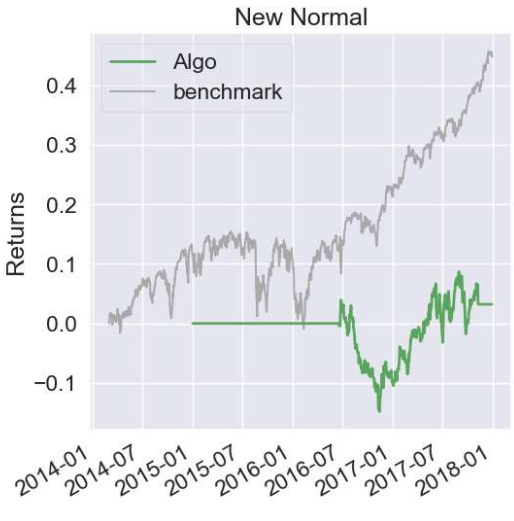

2017
4.9 -0.8 4.7 0.8 6.3 -6.1 1.4 8.4 -7.5 4.8 -2 0
Jan Mar May Jul Sep Nov
Month

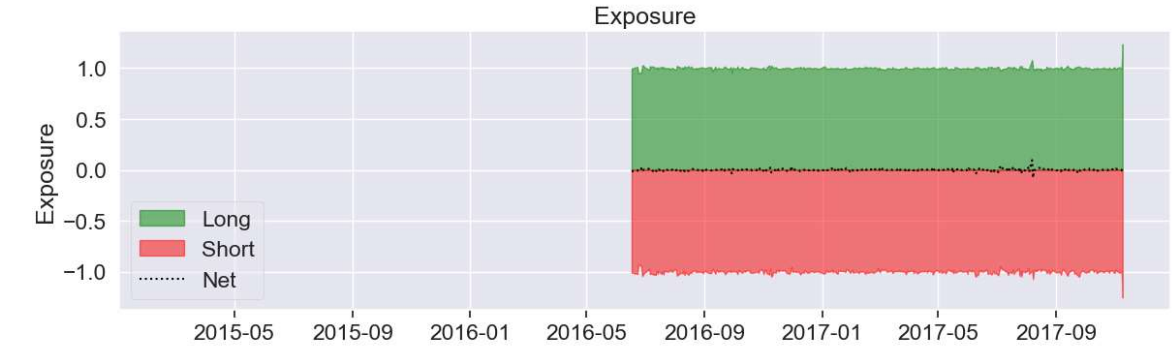

2017
-10% -5% 0% 5% 10% 15%
Returns

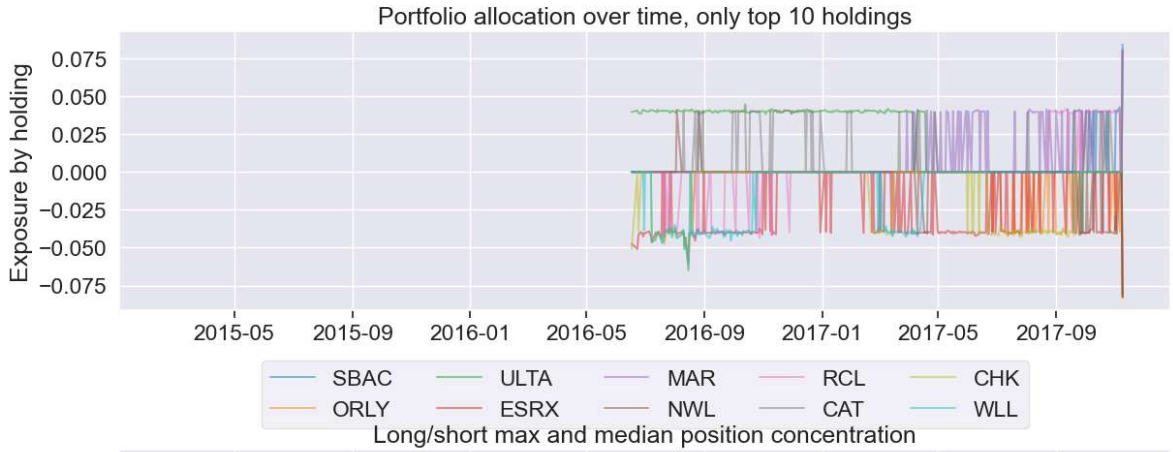

Number
7.5
5.0
2.5
0.0
-5% 0% 5%
Returns


Return quantiles
Out-of-sample data
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
Daily


Fall2015
Returns
0.00
−0.02
−0.04
−0.06
−0.08
−0.10
Algo
benchmark
2015-08-15
2015-08-22
2015-09-01
2015-09-08
2015-09-15
2015-09-22
2015-10-01


New Normal
Returns
0.4
0.3
0.2
0.1
0.0
−0.1
Algo
benchmark
2014-01
2014-07
2015-01
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01

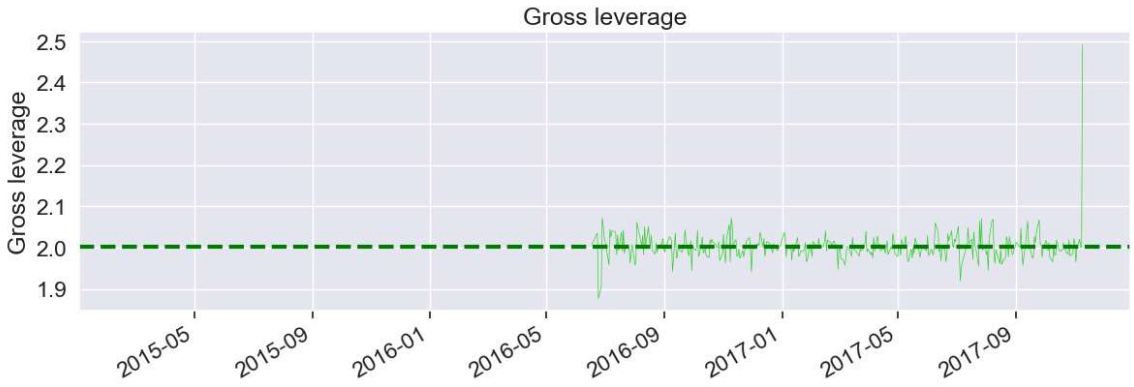

Exposure
Exposure
1.0
0.5
0.0
−0.5
−1.0
Long
Short
Net
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09

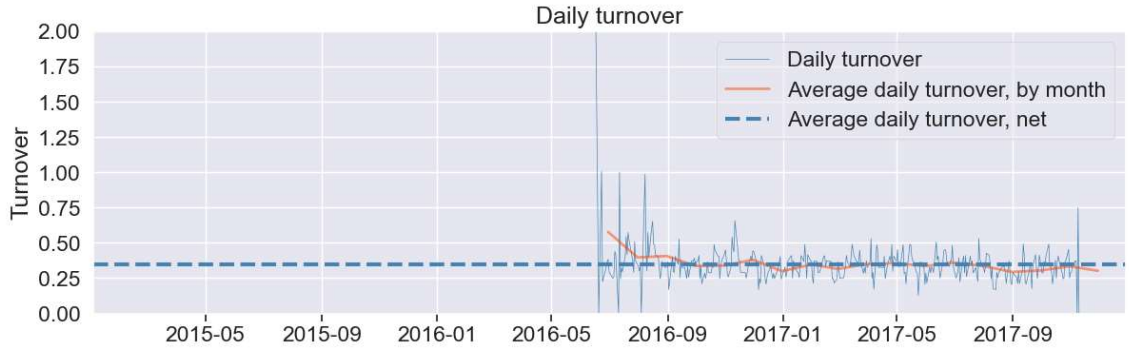

Portfolio allocation over time, only top 10 holdings
Exposure by holding
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
SBAC
ULTA
MAR
RCL
CHK
ORLY
ESRX
NWL
CAT
WLL


Long/short max and median position concentration
Exposure
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
max_long
median_long
median_short
max_short
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09

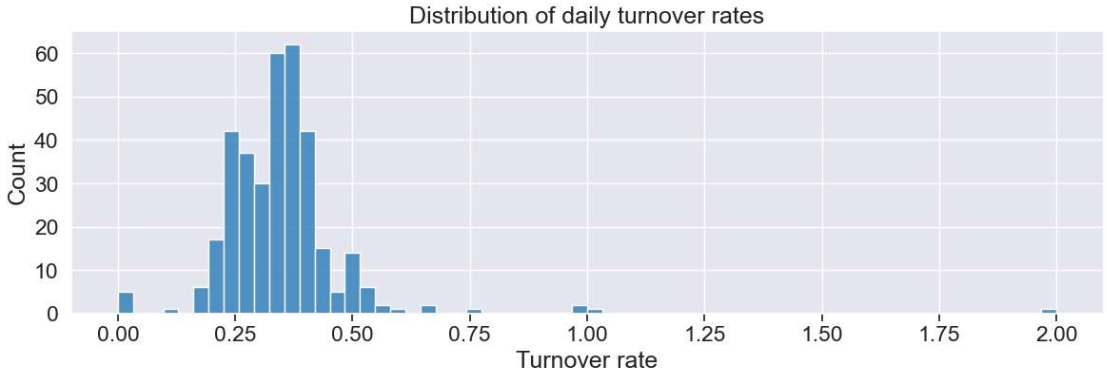

Total holdings
Holdings
60.0
57.5
55.0
52.5
50.0
47.5
45.0
42.5
Daily holdings
Average daily holdings, by month
Average daily holdings, overall
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09

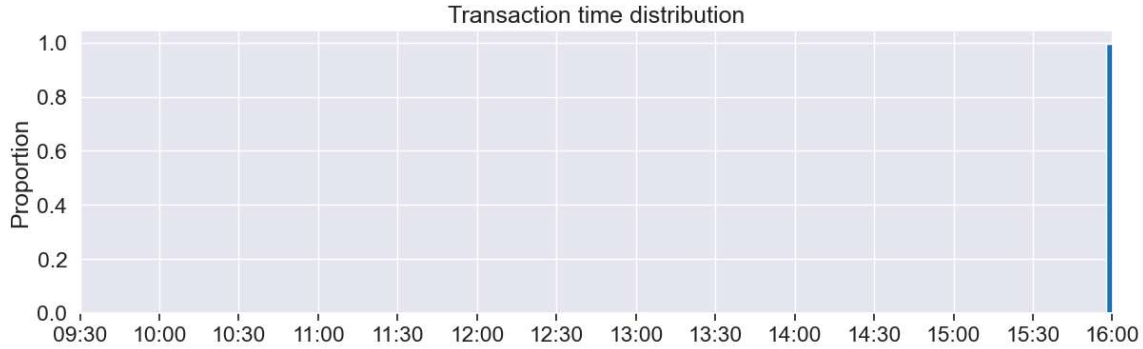

Long and short holdings
Holdings
30
25
20
15
10
5
0
Long (max: 31, min: 24)
Short (max: 30, min: 17)
Overlap
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09

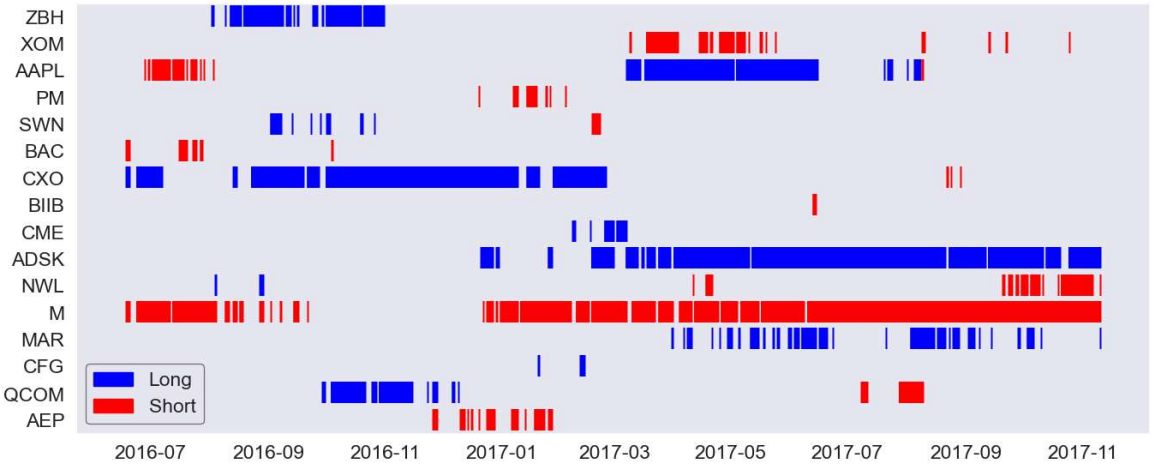

Gross leverage
Gross leverage
2.5
2.4
2.3
2.2
2.1
2.0
1.9
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09

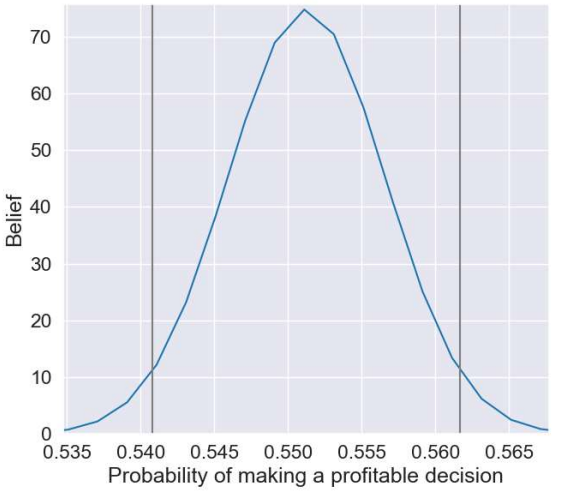

Daily turnover
Daily turnover
Average daily turnover, by month
Average daily turnover, net
Turnover
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09

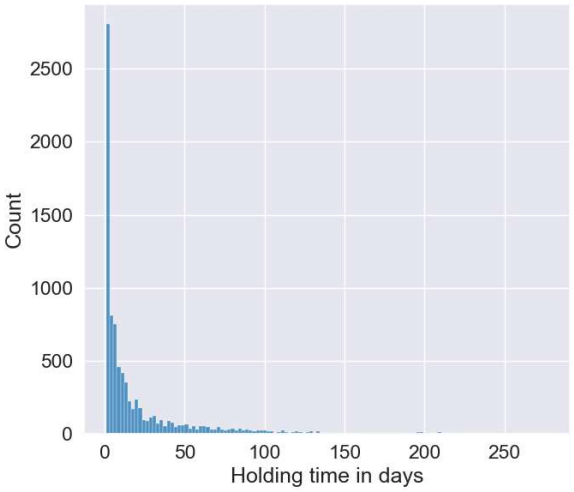

Daily trading volume
Amount of shares traded
60000
50000
40000
30000
20000
10000
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09

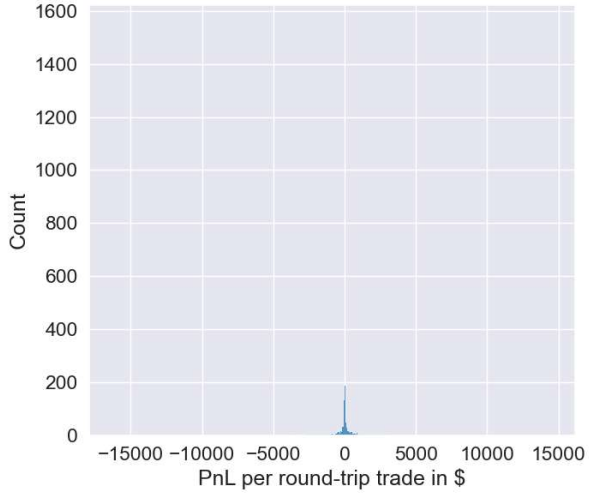

Distribution of daily turnover rates
Count
60
50
40
30
20
10
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Turnover rate

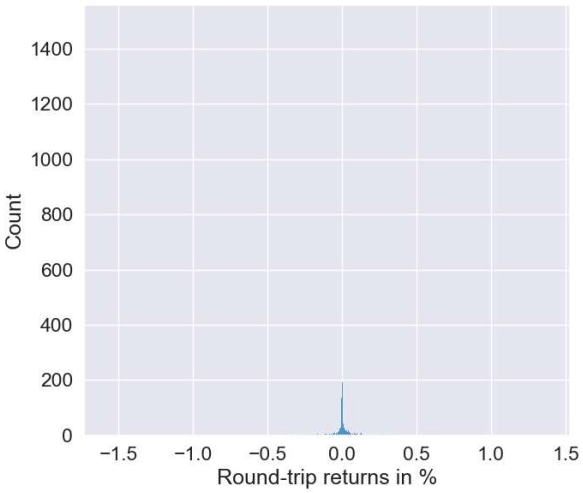

Transaction time distribution
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
09:30
10:00
10:30
11:00
11:30
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00


ZBH
XOM
AAPL
PM
SWN
BAC
CXO
BIIB
CME
ADSK
NWL
M
MAR
CFG
QCOM
AEP
Long
Short
2016-07
2016-09
2016-11
2017-01
2017-03
2017-05
2017-07
2017-09
2017-11


Belief
Probability of making a profitable decision


Count
Holding time in days


Count
PnL per round-trip trade in $


Count
Round-trip returns in %

In [ ]:

In [ ]: