04_single_factor_zipline

September 29, 2021

1 Zipline Backtest with Single Factor

This notebook develops and test a simple mean-reversion factor that measures how much recent performance has deviated from the historical average. Short-term reversal is a common strategy that takes advantage of the weakly predictive pattern that stock price increases are likely to mean-revert back down over horizons from less than a minute to one month.

1.1 Imports & Settings

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

[2]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
```

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

We are first going to illustrate the zipline alpha factor research workflow in an offline environment. In particular, we will develop and test a simple mean-reversion factor that measures how much recent performance has deviated from the historical average.

Short-term reversal is a common strategy that takes advantage of the weakly predictive pattern that stock price increases are likely to mean-revert back down over horizons from less than a minute to one month.

To this end, the factor computes the z-score for the last monthly return relative to the rolling monthly returns over the last year. At this point, we will not place any orders to simply illustrate the implementation of a CustomFactor and record the results during the simulation.

After some basic settings, `MeanReversion` subclasses `CustomFactor` and defines a `compute()` method. It creates default inputs of monthly returns over an also default year-long window so that the monthly_return variable will have 252 rows and one column for each security in the Quandl dataset on a given day.

The `compute_factors()` method creates a `MeanReversion` factor instance and creates long, short, and ranking pipeline columns. The former two contain Boolean values that could be used to place orders, and the latter reflects that overall ranking to evaluate the overall factor performance. Furthermore, it uses the built-in `AverageDollarVolume` factor to limit the computation to more liquid stocks

The result would allow us to place long and short orders. We will see in the next chapter how to build a portfolio by choosing a rebalancing period and adjusting portfolio holdings as new signals arrive.

- The `initialize()` method registers the compute_factors() pipeline, and the before_trading_start() method ensures the pipeline runs on a daily basis.
- The `record()` function adds the pipeline's ranking column as well as the current asset prices to the performance DataFrame returned by the `run_algorithm()` function

Run using jupyter notebook extension

[4]:
```
%load_ext zipline
```

Using the `quandl` bundle instead of the default `quantopian-quandl` because the latter has a bug that requires (manually) fixing the SQL database. If you have a file with benchmark returns you can provide this instead of `--no-benchmark` (see docs).

[5]:
```
%%zipline --start 2015-1-1 --end 2018-1-1 --output single_factor.pickle␣
↪--no-benchmark --bundle quandl

from zipline.api import (
    attach_pipeline,
    date_rules,
    time_rules,
    order_target_percent,
    pipeline_output,
    record,
    schedule_function,
    get_open_orders,
    calendars
)
from zipline.finance import commission, slippage
from zipline.pipeline import Pipeline, CustomFactor
from zipline.pipeline.factors import Returns, AverageDollarVolume
import numpy as np
import pandas as pd

MONTH = 21
YEAR = 12 * MONTH
N_LONGS = N_SHORTS = 25
VOL_SCREEN = 1000


class MeanReversion(CustomFactor):
    """Compute ratio of latest monthly return to 12m average,
       normalized by std dev of monthly returns"""
    inputs = [Returns(window_length=MONTH)]
    window_length = YEAR
```

```
    def compute(self, today, assets, out, monthly_returns):
        df = pd.DataFrame(monthly_returns)
        out[:] = df.iloc[-1].sub(df.mean()).div(df.std())


def compute_factors():
    """Create factor pipeline incl. mean reversion,
        filtered by 30d Dollar Volume; capture factor ranks"""
    mean_reversion = MeanReversion()
    dollar_volume = AverageDollarVolume(window_length=30)
    return Pipeline(columns={'longs': mean_reversion.bottom(N_LONGS),
                             'shorts': mean_reversion.top(N_SHORTS),
                             'ranking': mean_reversion.rank(ascending=False)},
                    screen=dollar_volume.top(VOL_SCREEN))


def exec_trades(data, assets, target_percent):
    """Place orders for assets using target portfolio percentage"""
    for asset in assets:
        if data.can_trade(asset) and not get_open_orders(asset):
            order_target_percent(asset, target_percent)


def rebalance(context, data):
    """Compute long, short and obsolete holdings; place trade orders"""
    factor_data = context.factor_data
    record(factor_data=factor_data.ranking)

    assets = factor_data.index
    record(prices=data.current(assets, 'price'))

    longs = assets[factor_data.longs]
    shorts = assets[factor_data.shorts]
    divest = set(context.portfolio.positions.keys()) - set(longs.union(shorts))

    exec_trades(data, assets=divest, target_percent=0)
    exec_trades(data, assets=longs, target_percent=1 / N_LONGS)
    exec_trades(data, assets=shorts, target_percent=-1 / N_SHORTS)


def initialize(context):
    """Setup: register pipeline, schedule rebalancing,
        and set trading params"""
    attach_pipeline(compute_factors(), 'factor_pipeline')
    schedule_function(rebalance,
                      date_rules.week_start(),
                      time_rules.market_open(),
```

```
                        calendar=calendars.US_EQUITIES)
    context.set_commission(commission.PerShare(cost=.01, min_trade_cost=0))
    context.set_slippage(slippage.VolumeShareSlippage())


def before_trading_start(context, data):
    """Run factor pipeline"""
    context.factor_data = pipeline_output('factor_pipeline')
```

[5]:
```
                                       period_open              period_close
shorts_count   long_value  short_value  long_exposure           pnl
capital_used  short_exposure                                              orders
…     sortino max_drawdown  max_leverage  excess_return
treasury_period_return  trading_days  period_label  algorithm_period_return
factor_data                                              prices
2015-01-02 21:00:00+00:00 2015-01-02 14:31:00+00:00 2015-01-02 21:00:00+00:00
0        0.000        0.000          0.000      0.000000  0.000000e+00
0.000                                                 []  …         NaN
0.000000      0.000000            0.0                     0.0             1
2015-01                 0.000000
NaN                                                  NaN
2015-01-05 21:00:00+00:00 2015-01-05 14:31:00+00:00 2015-01-05 21:00:00+00:00
0        0.000        0.000          0.000      0.000000  0.000000e+00
0.000  [{'id': '4ef8337dc8614f9183a8e8a7a0df3c18', 'd…  …         NaN
0.000000      0.000000            0.0                     0.0             2
2015-01                 0.000000  Equity(0 [A])          2707.0
Equity(2 [AAL]) …  Equity(0 [A])           39.800
Equity(2 [AAL])…
2015-01-06 21:00:00+00:00 2015-01-06 14:31:00+00:00 2015-01-06 21:00:00+00:00
4  4731525.565 -1617262.705    4731525.565  -3799.475085 -3.118062e+06
-1617262.705  [{'id': '4ef8337dc8614f9183a8e8a7a0df3c18', 'd…  …   -9.165151
-0.000380      0.635120            0.0                     0.0             3
2015-01                -0.000380  Equity(0 [A])          2707.0
Equity(2 [AAL]) …  Equity(0 [A])           39.800
Equity(2 [AAL])…
2015-01-07 21:00:00+00:00 2015-01-07 14:31:00+00:00 2015-01-07 21:00:00+00:00
4  4757100.850 -1629987.410    4757100.850  12850.580000  0.000000e+00
-1629987.410                                                 []  …   18.918320
-0.000380      0.638131            0.0                     0.0             4
2015-01                 0.000905  Equity(0 [A])          2707.0
Equity(2 [AAL]) …  Equity(0 [A])           39.800
Equity(2 [AAL])…
2015-01-08 21:00:00+00:00 2015-01-08 14:31:00+00:00 2015-01-08 21:00:00+00:00
4  4835941.280 -1645106.080    4835941.280  63721.760000  0.000000e+00
-1645106.080                                                 []  …  135.877107
-0.000380      0.643422            0.0                     0.0             5
2015-01                 0.007277  Equity(0 [A])          2707.0
```

```
Equity(2 [AAL]) …  Equity(0 [A])           39.800
Equity(2 [AAL])…
…                                                  …                          …
…           …           …             …            …            …
…                                                    …  …          …
…            …             …                       …            …
…                        …                                                  …
…
2017-12-22 21:00:00+00:00 2017-12-22 14:31:00+00:00 2017-12-22 21:00:00+00:00
10  5266915.040 -4598653.320    5266915.040 -16960.030000  0.000000e+00
-4598653.320                                                  []  …    0.609931
-0.142578      1.259576            0.0                      0.0           751
2017-12                 0.102076  Equity(0 [A])          2393.0
Equity(1 [AA])  …  Equity(0 [A])          67.66
Equity(1 [AA])   …
2017-12-26 21:00:00+00:00 2017-12-26 14:31:00+00:00 2017-12-26 21:00:00+00:00
10  5217746.950 -4592983.200    5217746.950 -43497.970000  0.000000e+00
-4592983.200  [{'id': 'fb0fdf179c8641beaeec4f1bff3d1ca4', 'd…  …    0.587086
-0.142578      1.259576            0.0                      0.0           752
2017-12                 0.097727  Equity(0 [A])          2363.0
Equity(1 [AA])  …  Equity(0 [A])          67.25
Equity(1 [AA])   …
2017-12-27 21:00:00+00:00 2017-12-27 14:31:00+00:00 2017-12-27 21:00:00+00:00
13  3966687.510 -5240774.395    3966687.510  41248.841483  1.940099e+06
-5240774.395  [{'id': 'fb0fdf179c8641beaeec4f1bff3d1ca4', 'd…  …    0.607626
-0.142578      1.259576            0.0                      0.0           753
2017-12                 0.101852  Equity(0 [A])          2363.0
Equity(1 [AA])  …  Equity(0 [A])          67.25
Equity(1 [AA])   …
2017-12-28 21:00:00+00:00 2017-12-28 14:31:00+00:00 2017-12-28 21:00:00+00:00
12  3967663.630 -5158764.315    3967663.630  25543.807669 -5.744239e+04
-5158764.315  [{'id': '5e21283eb12b47749c70fb46d0525f10', 'd…  …    0.620128
-0.142578      1.259576            0.0                      0.0           754
2017-12                 0.104406  Equity(0 [A])          2363.0
Equity(1 [AA])  …  Equity(0 [A])          67.25
Equity(1 [AA])   …
2017-12-29 21:00:00+00:00 2017-12-29 14:31:00+00:00 2017-12-29 21:00:00+00:00
12  3953902.420 -5097704.930    3953902.420  47298.175000  0.000000e+00
-5097704.930                                                  []  …    0.643540
-0.142578      1.259576            0.0                      0.0           755
2017-12                 0.109136  Equity(0 [A])          2363.0
Equity(1 [AA])  …  Equity(0 [A])          67.25
Equity(1 [AA])   …

[755 rows x 39 columns]
```

1.2 Inspect Results

We can get the result `DataFrame` using `_` which captures the last cell output (only works when you run it right after the above cell)

[6]:
```
result = _
```

[7]:
```
result.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 755 entries, 2015-01-02 21:00:00+00:00 to 2017-12-29
21:00:00+00:00
Data columns (total 39 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   period_open              755 non-null    datetime64[ns, UTC]
 1   period_close             755 non-null    datetime64[ns, UTC]
 2   shorts_count             755 non-null    int64
 3   long_value               755 non-null    float64
 4   short_value              755 non-null    float64
 5   long_exposure            755 non-null    float64
 6   pnl                      755 non-null    float64
 7   capital_used             755 non-null    float64
 8   short_exposure           755 non-null    float64
 9   orders                   755 non-null    object
 10  transactions             755 non-null    object
 11  positions                755 non-null    object
 12  gross_leverage           755 non-null    float64
 13  starting_exposure        755 non-null    float64
 14  net_leverage             755 non-null    float64
 15  ending_exposure          755 non-null    float64
 16  starting_value           755 non-null    float64
 17  ending_value             755 non-null    float64
 18  starting_cash            755 non-null    float64
 19  ending_cash              755 non-null    float64
 20  portfolio_value          755 non-null    float64
 21  returns                  755 non-null    float64
 22  longs_count              755 non-null    int64
 23  algo_volatility          754 non-null    float64
 24  benchmark_period_return  755 non-null    float64
 25  benchmark_volatility     754 non-null    float64
 26  alpha                    0 non-null      object
 27  beta                     0 non-null      object
 28  sharpe                   753 non-null    float64
 29  sortino                  753 non-null    float64
 30  max_drawdown             755 non-null    float64
 31  max_leverage             755 non-null    float64
 32  excess_return            755 non-null    float64
```

```
 33  treasury_period_return   755 non-null    float64
 34  trading_days             755 non-null    int64
 35  period_label             755 non-null    object
 36  algorithm_period_return  755 non-null    float64
 37  factor_data              754 non-null    object
 38  prices                   754 non-null    object
dtypes: datetime64[ns, UTC](2), float64(26), int64(3), object(8)
memory usage: 235.9+ KB
```

[8]:
```
fig, axes = plt.subplots(nrows=2, figsize=(14,6))
result.algorithm_period_return.plot(ax=axes[0], title='Cumulative Return')
result.algo_volatility.plot(ax=axes[1], title='Volatility')
sns.despine()
fig.tight_layout();
```