

04_japanese_equity_features

September 29, 2021

1 Japanese Equity Data - Feature Engineering

1.1 Imports & Settings

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

```
[2]: %matplotlib inline

from pathlib import Path

import numpy as np
import pandas as pd
import talib

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

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

```
[4]: idx = pd.IndexSlice
```

1.2 Get Data

1.2.1 Stooq Japanese Equity data 2014-2019

```
[5]: DATA_DIR = Path('.', 'data')
```

```
[6]: prices = (pd.read_hdf(DATA_DIR / 'assets.h5', 'stooq/jp/tse/stocks/prices')
               .loc[idx[:, '2010': '2019'], :])
               .loc[lambd df: ~df.index.duplicated(), :])
```

```
[7]: prices.info(show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 7491767 entries, ('1301.JP', Timestamp('2010-01-04 00:00:00')) to
('9997.JP', Timestamp('2019-12-30 00:00:00'))
Data columns (total 5 columns):
```

#	Column	Non-Null Count	Dtype
0	open	7491767 non-null	float64
1	high	7491767 non-null	float64
2	low	7491767 non-null	float64
3	close	7491767 non-null	float64
4	volume	7491767 non-null	int64

dtypes: float64(4), int64(1)
memory usage: 314.7+ MB

```
[8]: before = len(prices.index.unique('ticker').unique())
```

1.2.2 Remove symbols with missing values

```
[9]: prices = (prices.unstack('ticker')
               .sort_index()
               .ffill(limit=5)
               .dropna(axis=1)
               .stack('ticker')
               .swaplevel())
prices.info(show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 5578822 entries, ('1301.JP', Timestamp('2010-01-04 00:00:00')) to
('9997.JP', Timestamp('2019-12-30 00:00:00'))
Data columns (total 5 columns):
#   Column  Non-Null Count  Dtype
---  -
0   open    5578822 non-null  float64
1   high    5578822 non-null  float64
2   low     5578822 non-null  float64
3   close   5578822 non-null  float64
4   volume  5578822 non-null  float64
dtypes: float64(5)
memory usage: 234.3+ MB
```

```
[10]: after = len(prices.index.unique('ticker').unique())
print(f'Before: {before:,.0f} after: {after:,.0f}')
```

Before: 3,664 after: 2,278

1.2.3 Keep most traded symbols

```
[11]: dv = prices.close.mul(prices.volume)
keep = dv.groupby('ticker').median().nlargest(1000).index.tolist()
```

```
[13]: prices = prices.loc[idx[keep, :], :]
prices.info(show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 2449000 entries, ('9984.JP', Timestamp('2010-01-04 00:00:00')) to
('8107.JP', Timestamp('2019-12-30 00:00:00'))
Data columns (total 5 columns):
#   Column  Non-Null Count  Dtype
---  -
0   open    2449000 non-null  float64
1   high    2449000 non-null  float64
2   low     2449000 non-null  float64
3   close   2449000 non-null  float64
4   volume  2449000 non-null  float64
dtypes: float64(5)
memory usage: 103.0+ MB
```

1.3 Feature Engineering

1.3.1 Compute period returns

```
[14]: intervals = [1, 5, 10, 21, 63]
```

```
[15]: returns = []
by_ticker = prices.groupby(level='ticker').close
for t in intervals:
    returns.append(by_ticker.pct_change(t).to_frame(f'ret_{t}'))
returns = pd.concat(returns, axis=1)
```

```
[16]: returns.info(show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 2449000 entries, ('9984.JP', Timestamp('2010-01-04 00:00:00')) to
('8107.JP', Timestamp('2019-12-30 00:00:00'))
Data columns (total 5 columns):
#   Column  Non-Null Count  Dtype
---  -
0   ret_1    2448000 non-null  float64
1   ret_5    2444000 non-null  float64
2   ret_10   2439000 non-null  float64
3   ret_21   2428000 non-null  float64
4   ret_63   2386000 non-null  float64
dtypes: float64(5)
memory usage: 103.0+ MB
```

1.3.2 Remove outliers

```
[20]: max_ret_by_sym = returns.groupby(level='ticker').max()
```

```
[21]: percentiles = [0.001, .005, .01, .025, .05, .1]
percentiles += [1-p for p in percentiles]
max_ret_by_sym.describe(percentiles=sorted(percentiles)[6:])
```

```
[21]:
```

	ret_1	ret_5	ret_10	ret_21	ret_63
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.170931	0.348342	0.468836	0.620602	1.015105
std	0.083577	0.293201	0.527677	0.802252	1.498513
min	0.066825	0.105314	0.106831	0.130860	0.204482
50%	0.150563	0.252961	0.307507	0.389720	0.638036
90%	0.268196	0.608058	0.837042	1.061642	1.740919
95%	0.340981	0.934732	1.381340	1.643186	2.573712
97.5%	0.379277	1.319257	2.042933	2.889063	4.228995
99%	0.471767	1.668250	3.283656	4.457870	8.028316
99.5%	0.555778	1.958242	3.590300	5.666164	9.627482
99.9%	0.789489	2.166924	4.385537	8.049485	17.754458
max	0.826025	2.402364	5.336406	9.085616	22.067797

```
[22]: quantiles = max_ret_by_sym.quantile(.95)
to_drop = []
for ret, q in quantiles.items():
    to_drop.extend(max_ret_by_sym[max_ret_by_sym[ret]>q].index.tolist())
```

```
[23]: to_drop = pd.Series(to_drop).value_counts()
to_drop = to_drop[to_drop > 1].index.tolist()
len(to_drop)
```

```
[23]: 59
```

```
[24]: prices = prices.drop(to_drop, level='ticker')
prices.info(show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 2304509 entries, ('9984.JP', Timestamp('2010-01-04 00:00:00')) to
('8107.JP', Timestamp('2019-12-30 00:00:00'))
Data columns (total 5 columns):
#   Column  Dtype
---  -
0   open    float64
1   high    float64
2   low     float64
3   close   float64
4   volume  float64
dtypes: float64(5)
```

memory usage: 96.9+ MB

1.3.3 Calculate relative return percentiles

```
[25]: returns = []
      by_sym = prices.groupby(level='ticker').close
      for t in intervals:
          ret = by_sym.pct_change(t)
          rel_perc = (ret.groupby(level='date')
                      .apply(lambda x: pd.qcut(x, q=20, labels=False,
→duplicates='drop'))
                      returns.extend([ret.to_frame(f'ret_{t}'), rel_perc.
→to_frame(f'ret_rel_perc_{t}')]))
      returns = pd.concat(returns, axis=1)
```

1.3.4 Technical Indicators

Percentage Price Oscillator

```
[26]: ppo = prices.groupby(level='ticker').close.apply(talib.PPO).to_frame('PPO')
```

Normalized Average True Range

```
[27]: natr = prices.groupby(level='ticker', group_keys=False).apply(lambda x: talib.
→NATR(x.high, x.low, x.close)).to_frame('NATR')
```

Relative Strength Indicator

```
[28]: rsi = prices.groupby(level='ticker').close.apply(talib.RSI).to_frame('RSI')
```

Bollinger Bands

```
[29]: def get_bollinger(x):
      u, m, l = talib.BBANDS(x)
      return pd.DataFrame({'u': u, 'm': m, 'l': l})
```

```
[30]: bbands = prices.groupby(level='ticker').close.apply(get_bollinger)
```

1.3.5 Combine Features

```
[31]: data = pd.concat([prices, returns, ppo, natr, rsi, bbands], axis=1)
```

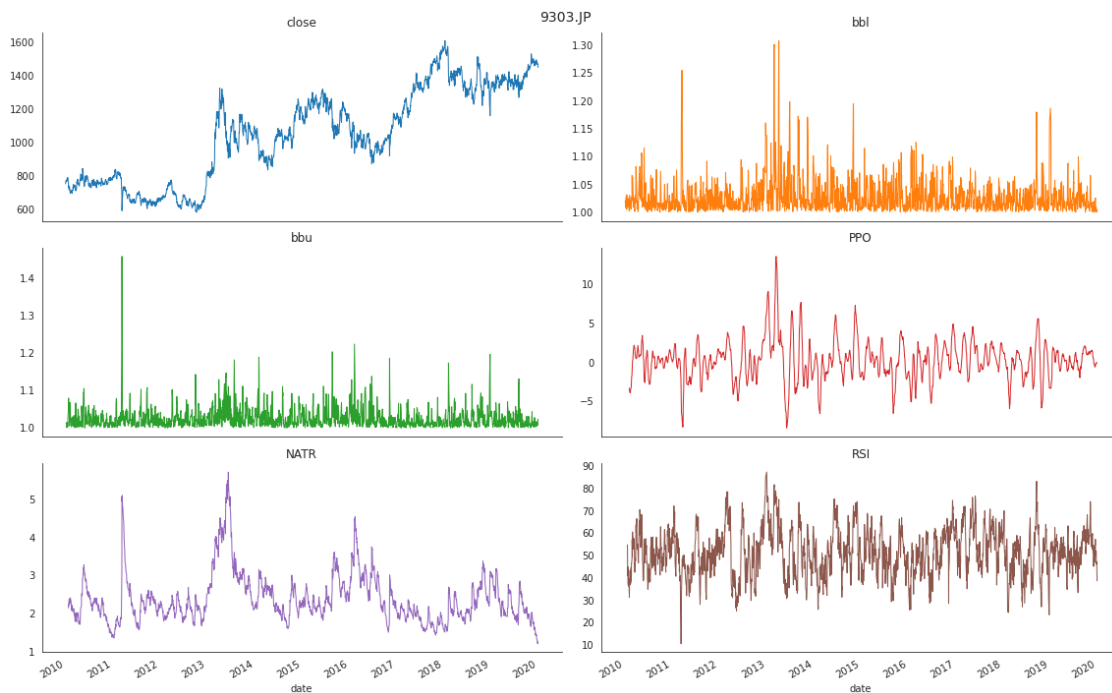
```
[32]: data['bbl'] = data.close.div(data.l)
      data['bbu'] = data.u.div(data.close)
      data = data.drop(['u', 'm', 'l'], axis=1)
```

```
[33]: data.bbu.corr(data.bbl, method='spearman')
```

```
[33]: -0.1598831761711913
```

1.3.6 Plot Indicators for randomly sample ticker

```
[34]: indicators = ['close', 'bbi', 'bbu', 'PPO', 'NATR', 'RSI']
ticker = np.random.choice(data.index.get_level_values('ticker'))
(data.loc[idx[ticker, :], indicators].reset_index('ticker', drop=True)
 .plot(lw=1, subplots=True, figsize=(16, 10), title=indicators, layout=(3, 2),
      legend=False))
plt.suptitle(ticker, fontsize=14)
sns.despine()
plt.tight_layout()
plt.subplots_adjust(top=.95)
```



```
[35]: data = data.drop(prices.columns, axis=1)
```

1.3.7 Create time period indicators

```
[36]: dates = data.index.get_level_values('date')
data['weekday'] = dates.weekday
data['month'] = dates.month
data['year'] = dates.year
```

1.4 Compute forward returns

```
[37]: outcomes = []
      by_ticker = data.groupby('ticker')
      for t in intervals:
          k = f'fwd_ret_{t:02}'
          outcomes.append(k)
          data[k] = by_ticker[f'ret_{t}'].shift(-t)
```

```
[38]: data.info(null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 2304509 entries, ('1332.JP', Timestamp('2010-01-04 00:00:00')) to
('9990.JP', Timestamp('2019-12-30 00:00:00'))
Data columns (total 23 columns):
#   Column                Non-Null Count  Dtype
---  -
0   ret_1                  2303568 non-null float64
1   ret_rel_perc_1         2303568 non-null float64
2   ret_5                  2299804 non-null float64
3   ret_rel_perc_5         2299804 non-null float64
4   ret_10                 2295099 non-null float64
5   ret_rel_perc_10        2295099 non-null float64
6   ret_21                 2284748 non-null float64
7   ret_rel_perc_21        2284748 non-null float64
8   ret_63                 2245226 non-null float64
9   ret_rel_perc_63        2245226 non-null float64
10  PPO                    2280984 non-null float64
11  NATR                   2291335 non-null float64
12  RSI                    2291335 non-null float64
13  bbl                    2300745 non-null float64
14  bbu                    2300745 non-null float64
15  weekday                2304509 non-null int64
16  month                  2304509 non-null int64
17  year                   2304509 non-null int64
18  fwd_ret_01             2303568 non-null float64
19  fwd_ret_05             2299804 non-null float64
20  fwd_ret_10             2295099 non-null float64
21  fwd_ret_21             2284748 non-null float64
22  fwd_ret_63             2245226 non-null float64
dtypes: float64(20), int64(3)
memory usage: 413.3+ MB
```

```
[39]: data.to_hdf('data.h5', 'stoq/japan/equities')
```