

03_preparing_the_model_data

September 29, 2021

1 Preparing Alpha Factors and Features to predict Stock Returns

1.1 Imports & Settings

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

```
[2]: %matplotlib inline

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from scipy.stats import pearsonr, spearmanr
from talib import RSI, BBANDS, MACD, ATR
```

```
[3]: MONTH = 21
YEAR = 12 * MONTH
```

```
[4]: START = '2013-01-01'
END = '2017-12-31'
```

```
[5]: sns.set_style('whitegrid')
idx = pd.IndexSlice
```

1.2 Loading Quandl Wiki Stock Prices & Meta Data

```
[6]: ohlcv = ['adj_open', 'adj_close', 'adj_low', 'adj_high', 'adj_volume']
```

```
[7]: DATA_STORE = '../data/assets.h5'
```

```
[8]: with pd.HDFStore(DATA_STORE) as store:
    prices = (store['quandl/wiki/prices']
              .loc[idx[START:END, :], ohlcv]
              .rename(columns=lambda x: x.replace('adj_', '')))
```

```

        .assign(volume=lambda x: x.volume.div(1000))
        .swaplevel()
        .sort_index()

stocks = (store['us_equities/stocks']
         .loc[:, ['marketcap', 'ipoyear', 'sector']])

```

1.3 Remove stocks with few observations

```

[9]: # want at least 2 years of data
min_obs = 2 * YEAR

# have this much per ticker
nobs = prices.groupby(level='ticker').size()

# keep those that exceed the limit
keep = nobs[nobs > min_obs].index

prices = prices.loc[idx[keep, :], :]

```

1.3.1 Align price and meta data

```

[10]: stocks = stocks[~stocks.index.duplicated() & stocks.sector.notnull()]
stocks.sector = stocks.sector.str.lower().str.replace(' ', '_')
stocks.index.name = 'ticker'

```

```

[11]: shared = (prices.index.get_level_values('ticker').unique()
               .intersection(stocks.index))
stocks = stocks.loc[shared, :]
prices = prices.loc[idx[shared, :], :]

```

```

[12]: prices.info(show_counts=True)

```

```

<class 'pandas.core.frame.DataFrame'>
MultiIndex: 2904233 entries, ('A', Timestamp('2013-01-02 00:00:00')) to ('ZUMZ',
Timestamp('2017-12-29 00:00:00'))
Data columns (total 5 columns):
#   Column   Non-Null Count  Dtype
---  -
0   open     2904233 non-null  float64
1   close    2904233 non-null  float64
2   low      2904233 non-null  float64
3   high     2904233 non-null  float64
4   volume   2904233 non-null  float64
dtypes: float64(5)
memory usage: 122.6+ MB

```

```
[13]: stocks.info(show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2348 entries, A to ZUMZ
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  -
0   marketcap    2345 non-null   float64
1   ipoyear      1026 non-null   float64
2   sector       2348 non-null   object
dtypes: float64(2), object(1)
memory usage: 73.4+ KB
```

```
[14]: stocks.sector.value_counts()
```

```
[14]: consumer_services      440
finance                    393
technology                 297
health_care                297
capital_goods              227
basic_industries           138
consumer_non-durables      126
energy                     123
public_utilities           105
consumer_durables           78
miscellaneous              69
transportation             55
Name: sector, dtype: int64
```

Optional: persist intermediate results:

```
[15]: # with pd.HDFStore('tmp.h5') as store:
#     store.put('prices', prices)
#     store.put('stocks', stocks)
```

```
[16]: # with pd.HDFStore('tmp.h5') as store:
#     prices = store['prices']
#     stocks = store['stocks']
```

1.4 Compute Rolling Average Dollar Volume

```
[17]: # compute dollar volume to determine universe
prices['dollar_vol'] = prices[['close', 'volume']].prod(axis=1)
```

```
[18]: prices['dollar_vol_1m'] = (prices.dollar_vol.groupby('ticker')
                              .rolling(window=21, level='date')
                              .mean()).values
```

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

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 2904233 entries, ('A', Timestamp('2013-01-02 00:00:00')) to ('ZUMZ',
Timestamp('2017-12-29 00:00:00'))
Data columns (total 7 columns):
#   Column          Non-Null Count  Dtype
---  -
0   open            2904233 non-null  float64
1   close           2904233 non-null  float64
2   low             2904233 non-null  float64
3   high            2904233 non-null  float64
4   volume          2904233 non-null  float64
5   dollar_vol      2904233 non-null  float64
6   dollar_vol_1m   2857273 non-null  float64
dtypes: float64(7)
memory usage: 166.9+ MB
```

```
[20]: prices['dollar_vol_rank'] = (prices.groupby('date')
                                   .dollar_vol_1m
                                   .rank(ascending=False))
```

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

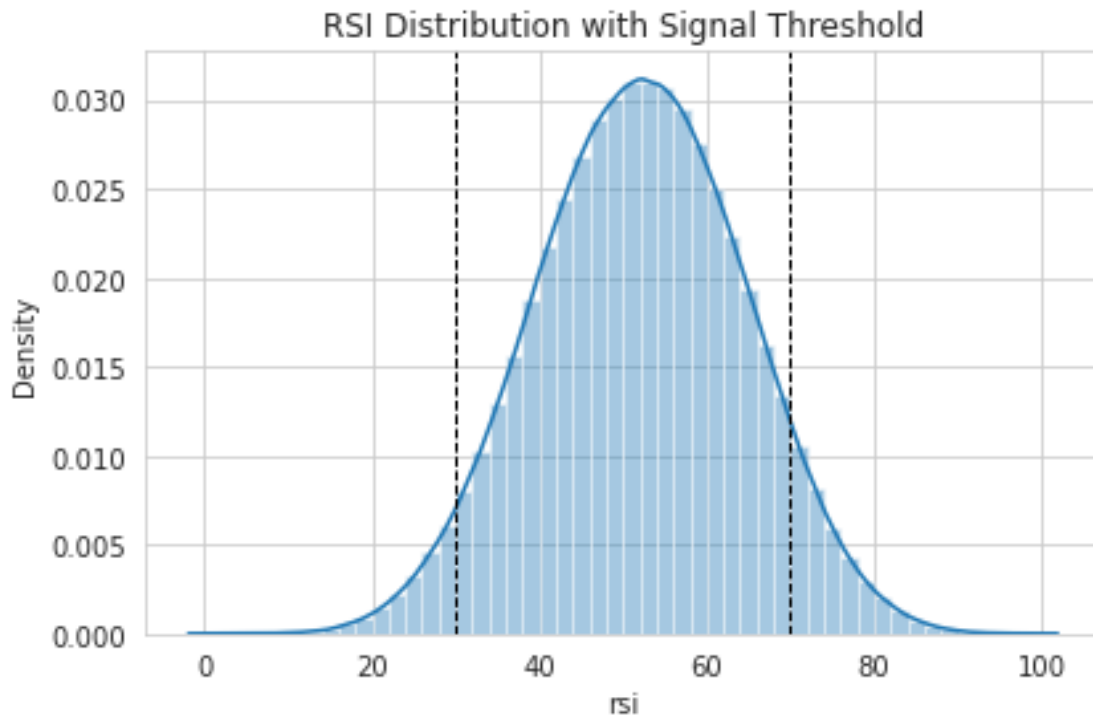
```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 2904233 entries, ('A', Timestamp('2013-01-02 00:00:00')) to ('ZUMZ',
Timestamp('2017-12-29 00:00:00'))
Data columns (total 8 columns):
#   Column          Non-Null Count  Dtype
---  -
0   open            2904233 non-null  float64
1   close           2904233 non-null  float64
2   low             2904233 non-null  float64
3   high            2904233 non-null  float64
4   volume          2904233 non-null  float64
5   dollar_vol      2904233 non-null  float64
6   dollar_vol_1m   2857273 non-null  float64
7   dollar_vol_rank 2857273 non-null  float64
dtypes: float64(8)
memory usage: 189.1+ MB
```

1.5 Add some Basic Factors

1.5.1 Compute the Relative Strength Index

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

```
[23]: ax = sns.distplot(prices.rsi.dropna())
ax.axvline(30, ls='--', lw=1, c='k')
ax.axvline(70, ls='--', lw=1, c='k')
ax.set_title('RSI Distribution with Signal Threshold')
plt.tight_layout();
```



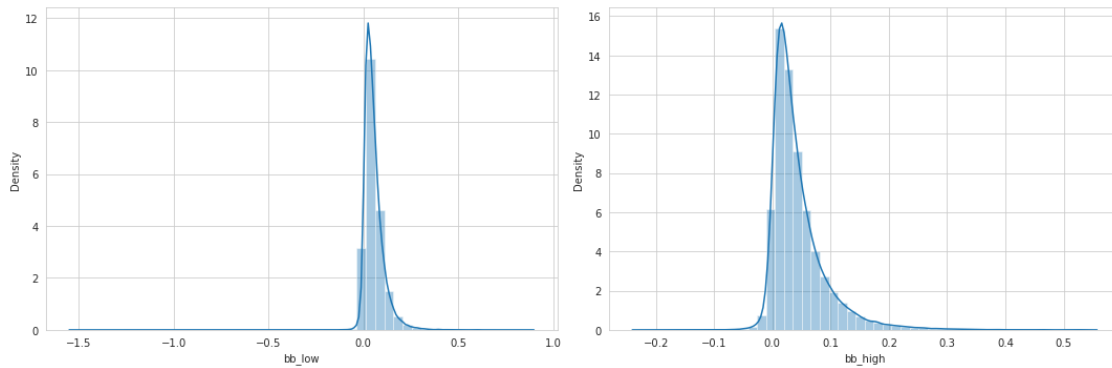
1.5.2 Compute Bollinger Bands

```
[24]: def compute_bb(close):
    high, mid, low = BBANDS(close, timeperiod=20)
    return pd.DataFrame({'bb_high': high, 'bb_low': low}, index=close.index)
```

```
[25]: prices = (prices.join(prices
    .groupby(level='ticker')
    .close
    .apply(compute_bb)))
```

```
[26]: prices['bb_high'] = prices.bb_high.sub(prices.close).div(prices.bb_high).
    ↪ apply(np.log1p)
prices['bb_low'] = prices.close.sub(prices.bb_low).div(prices.close).apply(np.
    ↪ log1p)
```

```
[27]: fig, axes = plt.subplots(ncols=2, figsize=(15, 5))
sns.distplot(prices.loc[prices.dollar_vol_rank<100, 'bb_low'].dropna(),
             ↪ax=axes[0])
sns.distplot(prices.loc[prices.dollar_vol_rank<100, 'bb_high'].dropna(),
             ↪ax=axes[1])
plt.tight_layout();
```

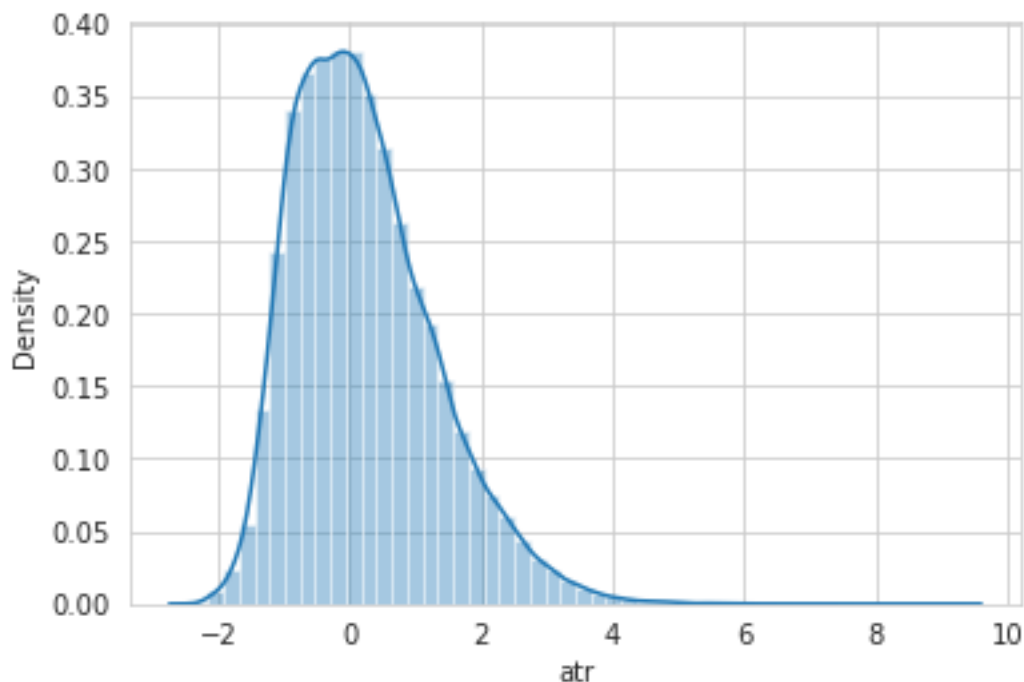


1.5.3 Compute Average True Range

```
[28]: def compute_atr(stock_data):
        df = ATR(stock_data.high, stock_data.low,
                  stock_data.close, timeperiod=14)
        return df.sub(df.mean()).div(df.std())

[29]: prices['atr'] = (prices.groupby('ticker', group_keys=False)
                      .apply(compute_atr))

[30]: sns.distplot(prices[prices.dollar_vol_rank<50].atr.dropna());
```



1.5.4 Compute Moving Average Convergence/Divergence

```
[31]: def compute_macd(close):
      macd = MACD(close)[0]
      return (macd - np.mean(macd))/np.std(macd)
```

```
[32]: prices['macd'] = (prices
                        .groupby('ticker', group_keys=False)
                        .close
                        .apply(compute_macd))
```

```
[33]: prices.macd.describe(percentiles=[.001, .01, .02, .03, .04, .05, .95, .96, .97,
    ↪ .98, .99, .999]).apply(lambda x: f'{x:,.1f}')
```

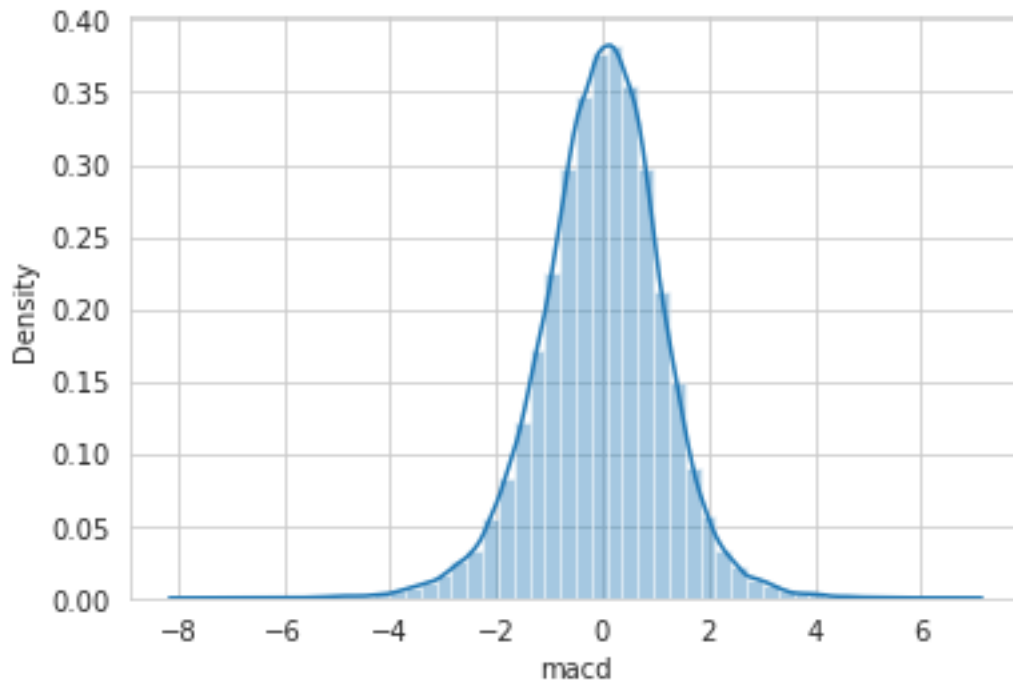
```
[33]: count    2,826,749.0
      mean      -0.0
      std       1.0
      min     -10.5
      0.1%     -4.1
      1%       -2.6
      2%       -2.2
      3%       -2.0
      4%       -1.8
      5%       -1.6
```

```

50%          0.0
95%          1.6
96%          1.7
97%          1.9
98%          2.1
99%          2.6
99.9%        4.0
max           8.7
Name: macd, dtype: object

```

```
[34]: sns.distplot(prices[prices.dollar_vol_rank<100].macd.dropna());
```



1.6 Compute Lagged Returns

```
[35]: lags = [1, 5, 10, 21, 42, 63]
```

```
[36]: returns = prices.groupby(level='ticker').close.pct_change()
percentiles=[.0001, .001, .01]
percentiles+= [1-p for p in percentiles]
returns.describe(percentiles=percentiles).iloc[2:].to_frame('percentiles').
↳style.format(lambda x: f'{x:,.2%}')
```

```
[36]: <pandas.io.formats.style.Styler at 0x7fc45043cd90>
```



```
[37]: q = 0.0001
```

1.6.1 Winsorize outliers

```
[38]: for lag in lags:
        prices[f'return_{lag}d'] = (prices.groupby(level='ticker').close
                                     .pct_change(lag)
                                     .pipe(lambda x: x.clip(lower=x.quantile(q),
                                                             upper=x.quantile(1 - q)))
                                     .add(1)
                                     .pow(1 / lag)
                                     .sub(1)
                                     )
```

1.6.2 Shift lagged returns

```
[39]: for t in [1, 2, 3, 4, 5]:
        for lag in [1, 5, 10, 21]:
            prices[f'return_{lag}d_lag{t}'] = (prices.groupby(level='ticker')
                                                [f'return_{lag}d'].shift(t * lag))
```

1.7 Compute Forward Returns

```
[40]: for t in [1, 5, 10, 21]:
        prices[f'target_{t}d'] = prices.groupby(level='ticker')[f'return_{t}d'].
        ↪shift(-t)
```

1.8 Combine Price and Meta Data

```
[41]: prices = prices.join(stocks[['sector']])
```

1.9 Create time and sector dummy variables

```
[42]: prices['year'] = prices.index.get_level_values('date').year
        prices['month'] = prices.index.get_level_values('date').month
```

```
[43]: prices.info(null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 2904233 entries, ('A', Timestamp('2013-01-02 00:00:00')) to ('ZUMZ',
Timestamp('2017-12-29 00:00:00'))
Data columns (total 46 columns):
#   Column          Non-Null Count  Dtype
---  -
0   open            2904233 non-null  float64
1   close           2904233 non-null  float64
```

2	low	2904233	non-null	float64
3	high	2904233	non-null	float64
4	volume	2904233	non-null	float64
5	dollar_vol	2904233	non-null	float64
6	dollar_vol_1m	2857273	non-null	float64
7	dollar_vol_rank	2857273	non-null	float64
8	rsi	2871361	non-null	float64
9	bb_high	2859618	non-null	float64
10	bb_low	2859585	non-null	float64
11	atr	2871361	non-null	float64
12	macd	2826749	non-null	float64
13	return_1d	2901885	non-null	float64
14	return_5d	2892493	non-null	float64
15	return_10d	2880753	non-null	float64
16	return_21d	2854925	non-null	float64
17	return_42d	2805617	non-null	float64
18	return_63d	2756309	non-null	float64
19	return_1d_lag1	2899537	non-null	float64
20	return_5d_lag1	2880753	non-null	float64
21	return_10d_lag1	2857273	non-null	float64
22	return_21d_lag1	2805617	non-null	float64
23	return_1d_lag2	2897189	non-null	float64
24	return_5d_lag2	2869013	non-null	float64
25	return_10d_lag2	2833793	non-null	float64
26	return_21d_lag2	2756309	non-null	float64
27	return_1d_lag3	2894841	non-null	float64
28	return_5d_lag3	2857273	non-null	float64
29	return_10d_lag3	2810313	non-null	float64
30	return_21d_lag3	2707001	non-null	float64
31	return_1d_lag4	2892493	non-null	float64
32	return_5d_lag4	2845533	non-null	float64
33	return_10d_lag4	2786833	non-null	float64
34	return_21d_lag4	2657693	non-null	float64
35	return_1d_lag5	2890145	non-null	float64
36	return_5d_lag5	2833793	non-null	float64
37	return_10d_lag5	2763353	non-null	float64
38	return_21d_lag5	2608385	non-null	float64
39	target_1d	2901885	non-null	float64
40	target_5d	2892493	non-null	float64
41	target_10d	2880753	non-null	float64
42	target_21d	2854925	non-null	float64
43	sector	2904233	non-null	object
44	year	2904233	non-null	int64
45	month	2904233	non-null	int64

dtypes: float64(43), int64(2), object(1)

memory usage: 1.1+ GB

```
[44]: prices.assign(sector=pd.factorize(prices.sector, sort=True)[0]).to_hdf('data.
    ↪h5', 'model_data/no_dummies')
```

```
[45]: prices = pd.get_dummies(prices,
    columns=['year', 'month', 'sector'],
    prefix=['year', 'month', ''],
    prefix_sep=['_', '_', ''],
    drop_first=True)
```

```
[46]: prices.info(null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 2904233 entries, ('A', Timestamp('2013-01-02 00:00:00')) to ('ZUMZ',
Timestamp('2017-12-29 00:00:00'))
Data columns (total 69 columns):
```

#	Column	Non-Null Count	Dtype
0	open	2904233 non-null	float64
1	close	2904233 non-null	float64
2	low	2904233 non-null	float64
3	high	2904233 non-null	float64
4	volume	2904233 non-null	float64
5	dollar_vol	2904233 non-null	float64
6	dollar_vol_1m	2857273 non-null	float64
7	dollar_vol_rank	2857273 non-null	float64
8	rsi	2871361 non-null	float64
9	bb_high	2859618 non-null	float64
10	bb_low	2859585 non-null	float64
11	atr	2871361 non-null	float64
12	macd	2826749 non-null	float64
13	return_1d	2901885 non-null	float64
14	return_5d	2892493 non-null	float64
15	return_10d	2880753 non-null	float64
16	return_21d	2854925 non-null	float64
17	return_42d	2805617 non-null	float64
18	return_63d	2756309 non-null	float64
19	return_1d_lag1	2899537 non-null	float64
20	return_5d_lag1	2880753 non-null	float64
21	return_10d_lag1	2857273 non-null	float64
22	return_21d_lag1	2805617 non-null	float64
23	return_1d_lag2	2897189 non-null	float64
24	return_5d_lag2	2869013 non-null	float64
25	return_10d_lag2	2833793 non-null	float64
26	return_21d_lag2	2756309 non-null	float64
27	return_1d_lag3	2894841 non-null	float64
28	return_5d_lag3	2857273 non-null	float64
29	return_10d_lag3	2810313 non-null	float64

```

30 return_21d_lag3      2707001 non-null float64
31 return_1d_lag4      2892493 non-null float64
32 return_5d_lag4      2845533 non-null float64
33 return_10d_lag4     2786833 non-null float64
34 return_21d_lag4     2657693 non-null float64
35 return_1d_lag5      2890145 non-null float64
36 return_5d_lag5      2833793 non-null float64
37 return_10d_lag5     2763353 non-null float64
38 return_21d_lag5     2608385 non-null float64
39 target_1d           2901885 non-null float64
40 target_5d           2892493 non-null float64
41 target_10d          2880753 non-null float64
42 target_21d          2854925 non-null float64
43 year_2014           2904233 non-null uint8
44 year_2015           2904233 non-null uint8
45 year_2016           2904233 non-null uint8
46 year_2017           2904233 non-null uint8
47 month_2             2904233 non-null uint8
48 month_3             2904233 non-null uint8
49 month_4             2904233 non-null uint8
50 month_5             2904233 non-null uint8
51 month_6             2904233 non-null uint8
52 month_7             2904233 non-null uint8
53 month_8             2904233 non-null uint8
54 month_9             2904233 non-null uint8
55 month_10            2904233 non-null uint8
56 month_11            2904233 non-null uint8
57 month_12            2904233 non-null uint8
58 capital_goods       2904233 non-null uint8
59 consumer_durables   2904233 non-null uint8
60 consumer_non-durables 2904233 non-null uint8
61 consumer_services   2904233 non-null uint8
62 energy              2904233 non-null uint8
63 finance              2904233 non-null uint8
64 health_care         2904233 non-null uint8
65 miscellaneous       2904233 non-null uint8
66 public_utilities    2904233 non-null uint8
67 technology          2904233 non-null uint8
68 transportation      2904233 non-null uint8
dtypes: float64(43), uint8(26)
memory usage: 1.1+ GB

```

1.10 Store Model Data

```
[47]: prices.to_hdf('data.h5', 'model_data')
```

1.11 Explore Data

1.11.1 Plot Factors

```
[48]: target = 'target_5d'
      top100 = prices[prices.dollar_vol_rank<100].copy()
```

1.11.2 RSI

```
[49]: top100.loc[:, 'rsi_signal'] = pd.cut(top100.rsi, bins=[0, 30, 70, 100])
```

```
[50]: top100.groupby('rsi_signal')['target_5d'].describe()
```

```
[50]:
```

	count	mean	std	min	25%	50% \
rsi_signal						
(0, 30]	4209.0	0.001126	0.010457	-0.067138	-0.003606	0.001051
(30, 70]	107244.0	0.000446	0.007711	-0.170571	-0.003054	0.000650
(70, 100]	10634.0	0.000018	0.006354	-0.087857	-0.002818	0.000145

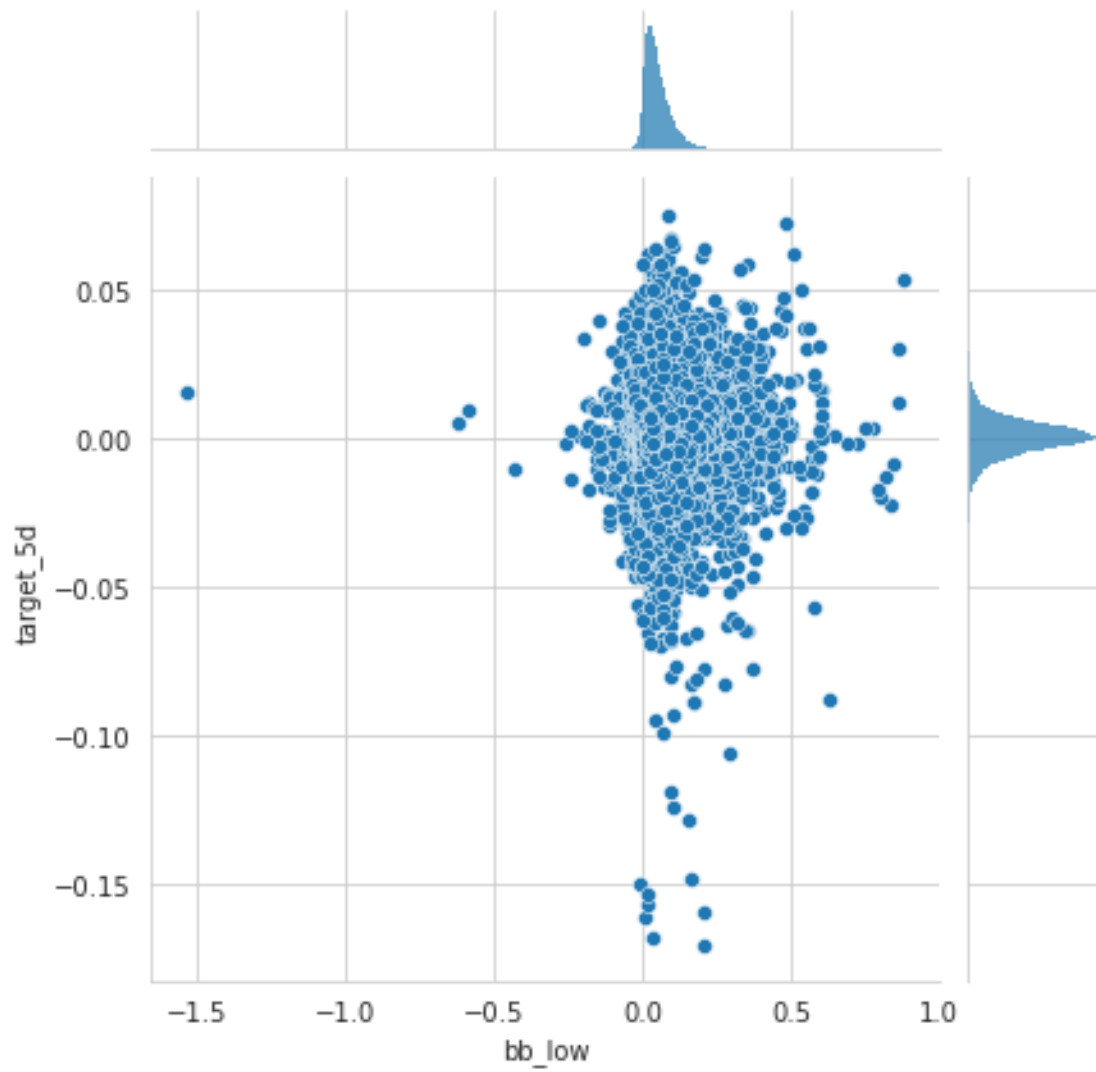
	75%	max
rsi_signal		
(0, 30]	0.006156	0.061889
(30, 70]	0.004246	0.075653
(70, 100]	0.003121	0.058570

1.11.3 Bollinger Bands

```
[51]: metric = 'bb_low'
      j=sns.jointplot(x=metric, y=target, data=top100)

      df = top100[[metric, target]].dropna()
      r, p = spearmanr(df[metric], df[target])
      print(f'{r:,.2%} ({p:.2%})')
```

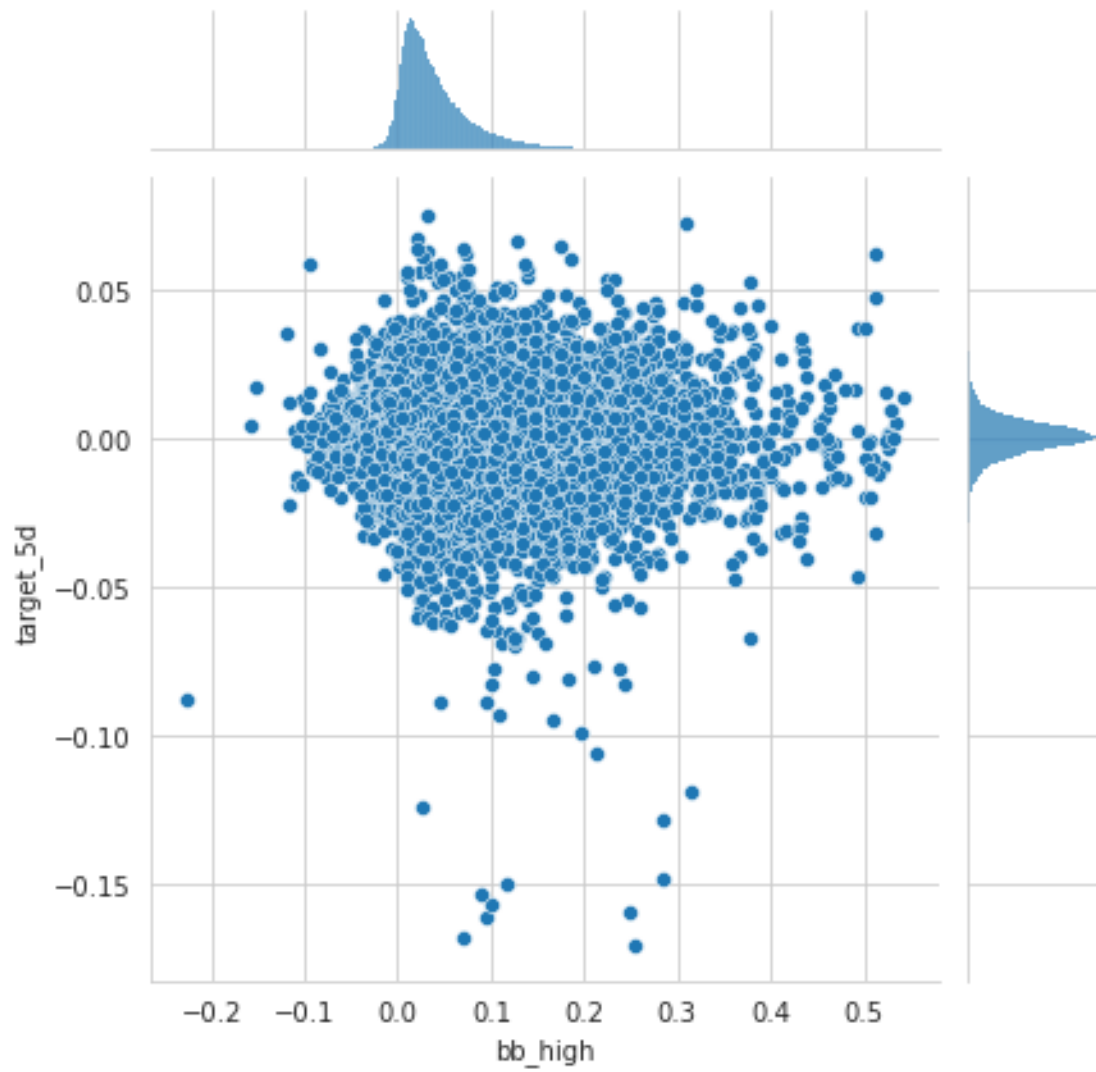
-2.68% (0.00%)



```
[52]: metric = 'bb_high'
j=sns.jointplot(x=metric, y=target, data=top100)

df = top100[[metric, target]].dropna()
r, p = spearmanr(df[metric], df[target])
print(f'{r:,.2%} ({p:.2%})')
```

4.21% (0.00%)

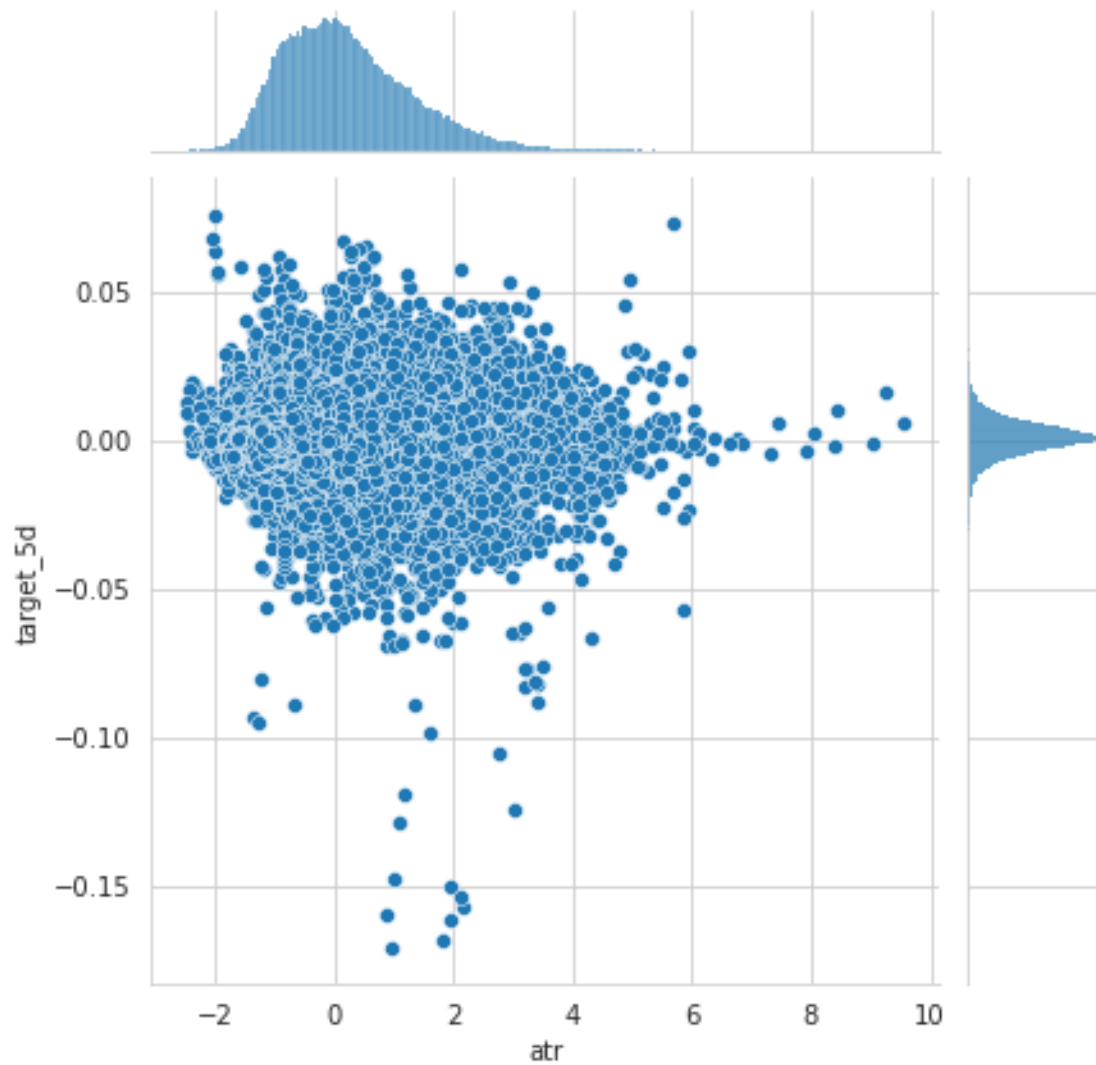


1.11.4 ATR

```
[53]: metric = 'atr'
j=sns.jointplot(x=metric, y=target, data=top100)

df = top100[[metric, target]].dropna()
r, p = spearmanr(df[metric], df[target])
print(f'{r:,.2%} ({p:,.2%})')
```

0.07% (80.08%)



1.11.5 MACD

```
[54]: metric = 'macd'
j=sns.jointplot(x=metric, y=target, data=top100)

df = top100[[metric, target]].dropna()
r, p = spearmanr(df[metric], df[target])
print(f'{r:,.2%} ({p:,.2%})')
```

-4.72% (0.00%)

