2- Feature Selection

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

In [2]:
```
%matplotlib inline

import os, sys
from time import time

from pathlib import Path
import numpy as np
import pandas as pd
import pandas_datareader.data as web

import statsmodels.api as sm
from sklearn.feature_selection import mutual_info_regression
from sklearn.preprocessing import scale
import lightgbm as lgb
from scipy.stats import spearmanr
import shap

from alphalens.tears import (create_returns_tear_sheet,
                             create_summary_tear_sheet,
                             create_full_tear_sheet)

from alphalens import plotting
from alphalens import performance as perf
from alphalens import utils

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

In [3]:
```
sns.set_style('whitegrid')
idx = pd.IndexSlice
deciles = np.arange(.1, 1, .1).round(1)
```

In [4]:
```
MONTH = 21
YEAR = 252
```

Load Data

In [ ]:
```
DATA_STORE = 'data/stock_prices.h5'
```

In [ ]:
```
factors = (pd.read_hdf(DATA_STORE, 'model_data').sort_index())
```

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

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 1403584 entries, ('A', Timestamp('2006-01-03 00:00:00')) to ('ZMH', Timestamp('2015-07-02 00:00:00'))
Data columns (total 55 columns):
 #   Column        Non-Null Count    Dtype
---  ------        --------------    -----
 0   ex-dividend   1403584 non-null  float64
 1   split_ratio   1403584 non-null  float64
 2   adj_open      1403584 non-null  float64
 3   adj_high      1403584 non-null  float64
 4   adj_low       1403584 non-null  float64
 5   adj_close     1403584 non-null  float64
 6   adj_volume    1403584 non-null  float64
 7   ret_01        1403084 non-null  float64
 8   ret_03        1402084 non-null  float64
 9   ret_05        1401084 non-null  float64
 10  ret_10        1398584 non-null  float64
 11  ret_21        1393084 non-null  float64
 12  ret_42        1382584 non-null  float64
 13  ret_63        1372084 non-null  float64
 14  ret_126       1340584 non-null  float64
 15  ret_252       1277584 non-null  float64
 16  ret_fwd       1403584 non-null  float64
 17  BB_UP         1394084 non-null  float64
 18  BB_LOW        1394084 non-null  float64
 19  BB_SQUEEZE    1394084 non-null  float64
 20  HT            1372084 non-null  float64
 21  SAR           1403084 non-null  float64
 22  ADX           1390084 non-null  float64
 23  ADXR          1383584 non-null  float64
 24  PPO           1391084 non-null  float64
 25  AARONOSC      1396584 non-null  float64
 26  BOP           1403584 non-null  float64
 27  CCI           1397084 non-null  float64
 28  MACD          1387084 non-null  float64
 29  MACD_SIGNAL   1387084 non-null  float64
 30  MACD_HIST     1387084 non-null  float64
 31  MFI           1396584 non-null  float64
 32  RSI           1396584 non-null  float64
 33  STOCHRSI      1389084 non-null  float64
 34  STOCH         1395066 non-null  float64
```

```
 35   ULTOSC        1389584 non-null  float64
 36   WILLR         1397084 non-null  float64
 37   AD            1403584 non-null  float64
 38   ADOSC         1396996 non-null  float64
 39   OBV           1403583 non-null  float64
 40   ATR           1396584 non-null  float64
 41   ALPHA_63      1334328 non-null  float64
 42   MARKET_63     1334328 non-null  float64
 43   SMB_63        1334328 non-null  float64
 44   HML_63        1334328 non-null  float64
 45   RMW_63        1334328 non-null  float64
 46   CMA_63        1334328 non-null  float64
 47   ALPHA_252     1239828 non-null  float64
 48   MARKET_252    1239828 non-null  float64
 49   SMB_252       1239828 non-null  float64
 50   HML_252       1239828 non-null  float64
 51   RMW_252       1239828 non-null  float64
 52   CMA_252       1239828 non-null  float64
 53   month         1403584 non-null  uint8
 54   weekday       1403584 non-null  uint8
dtypes: float64(53), uint8(2)
memory usage: 575.8+ MB
```

To get all features, we'll select the columns NOT containing forward returns:

In [8]:
```
fwd_returns = factors.filter(like='fwd').columns
features = factors.columns.difference(fwd_returns).tolist()
```

So here are the available features:

In [9]:
```
features
```

Out[9]:
```
['AARONOSC',
 'AD',
 'ADOSC',
 'ADX',
 'ADXR',
 'ALPHA_252',
 'ALPHA_63',
 'ATR',
 'BB_LOW',
 'BB_SQUEEZE',
 'BB_UP',
 'BOP',
 'CCI',
 'CMA_252',
 'CMA_63',
 'HML_252',
 'HML_63',
 'HT',
 'MACD',
 'MACD_HIST',
 'MACD_SIGNAL',
 'MARKET_252',
 'MARKET_63',
 'MFI',
 'OBV',
 'PPO',
 'RMW_252',
 'RMW_63',
 'RSI',
 'SAR',
 'SMB_252',
 'SMB_63',
 'STOCH',
 'STOCHRSI',
 'ULTOSC',
 'WILLR',
 'adj_close',
 'adj_high',
 'adj_low',
 'adj_open',
```

```
 'adj_volume',
 'ex-dividend',
 'month',
 'ret_01',
 'ret_03',
 'ret_05',
 'ret_10',
 'ret_126',
 'ret_21',
 'ret_252',
 'ret_42',
 'ret_63',
 'split_ratio',
 'weekday']
```

Factor Correlation

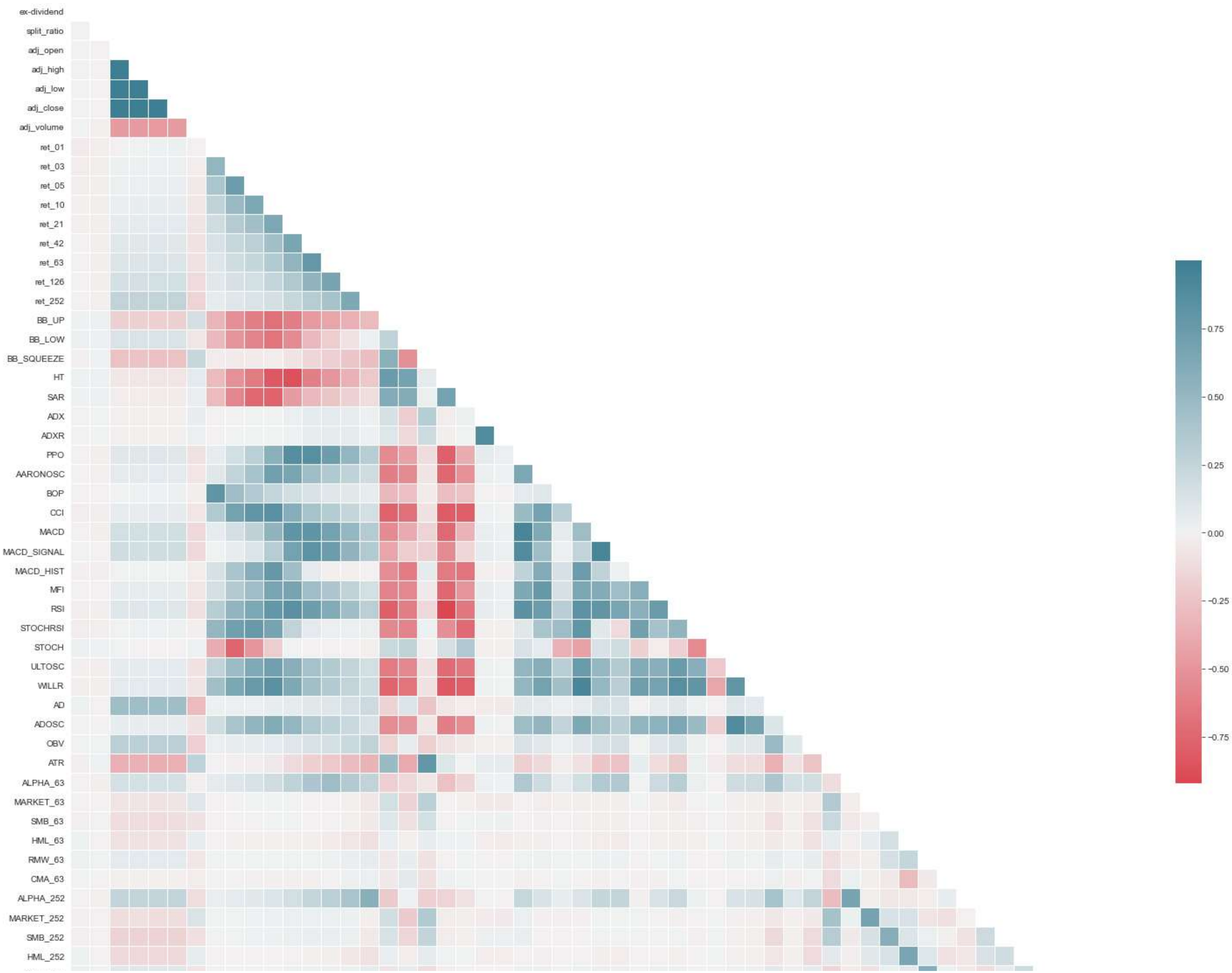
Which features are most alike in terms of their (rank) correlation?

In [10]:
```
corr_common = factors.drop(fwd_returns, axis=1).corr(method='spearman')
```

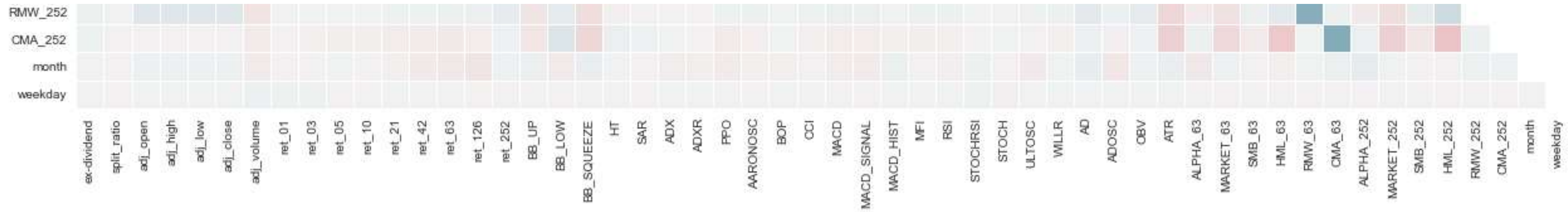
Based on this example from the seaborn docs.

In [11]:
```
mask = np.triu(np.ones_like(corr_common, dtype=np.bool))
fig, ax = plt.subplots(figsize=(22, 18))
cmap = sns.diverging_palette(10, 220, as_cmap=True)

sns.heatmap(corr_common, mask=mask, cmap=cmap, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
fig.tight_layout();
```

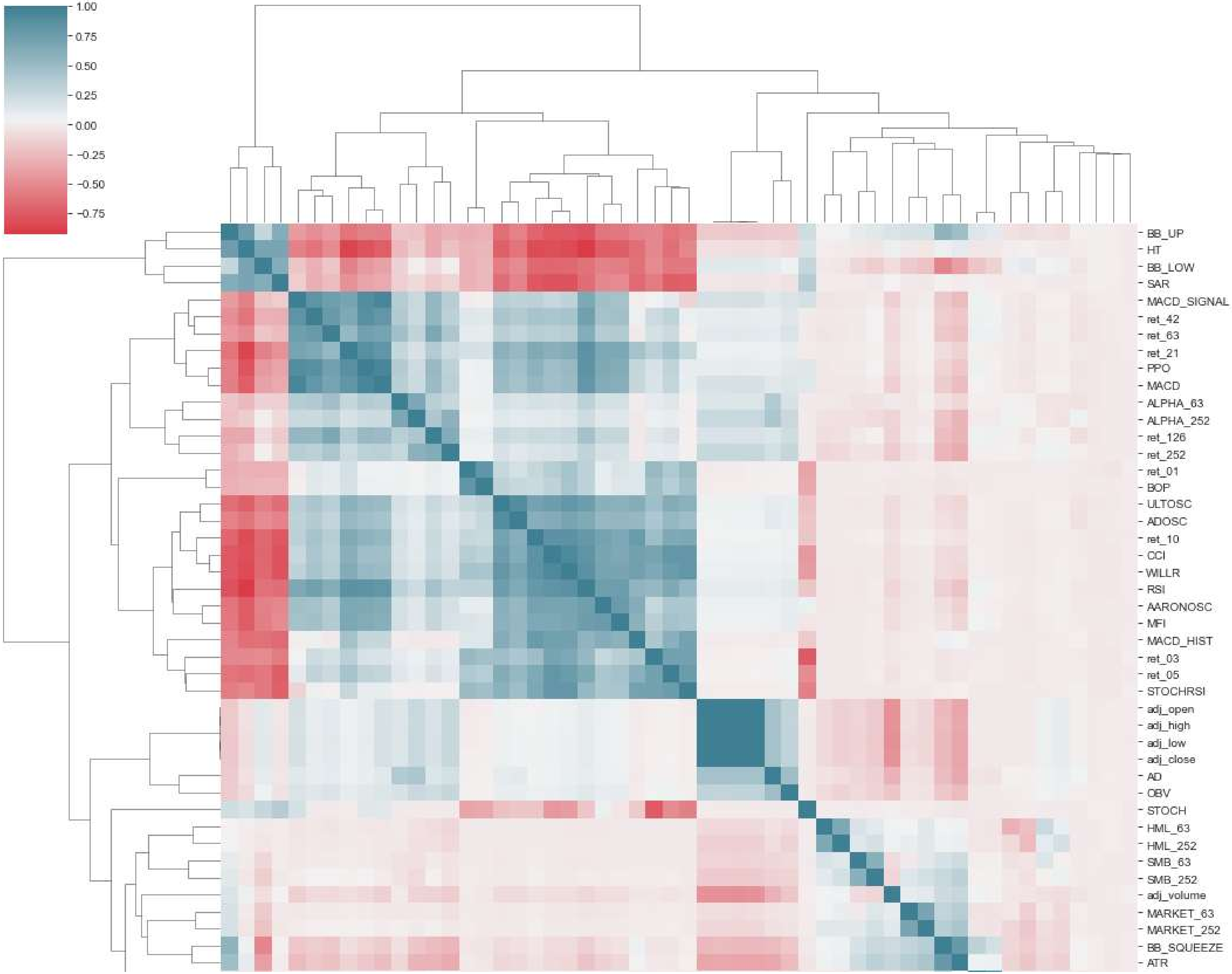


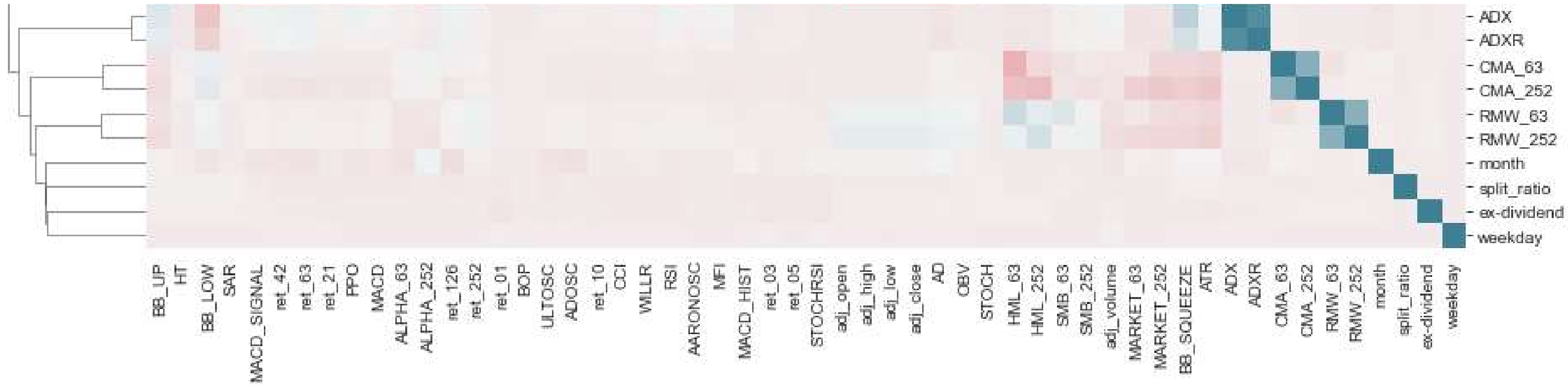
ex-dividend
split_ratio
adj_open
adj_high
adj_low
adj_close
adj_volume
ret_01
ret_03
ret_05
ret_10
ret_21
ret_42
ret_63
ret_126
ret_252
BB_UP
BB_LOW
BB_SQUEEZE
HT
SAR
ADX
ADXR
PPO
AARONOSC
BOP
CCI
MACD
MACD_SIGNAL
MACD_HIST
MFI
RSI
STOCHRSI
STOCH
ULTOSC
WILLR
AD
ADOSC
OBV
ATR
ALPHA_63
MARKET_63
SMB_63
HML_63
RMW_63
CMA_63
ALPHA_252
MARKET_252
SMB_252
HML_252
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75



The clusters show a few clusters of features that are relatively highly correlated (in absolute terms). If you are interested in reducing the number of features to speed up training or reduce memory, selecting one representative of each cluster would be a reasonable approach.

In [12]:
```
g = sns.clustermap(corr_common, cmap=cmap, figsize=(15, 15))
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
BB_UP
HT
BB_LOW
SAR
MACD_SIGNAL
ret_42
ret_63
ret_21
PPO
MACD
ALPHA_63
ALPHA_252
ret_126
ret_252
ret_01
BOP
ULTOSC
ADOSC
ret_10
CCI
WILLR
RSI
AARONOSC
MFI
MACD_HIST
ret_03
ret_05
STOCHRSI
adj_open
adj_high
adj_low
adj_close
AD
OBV
STOCH
HML_63
HML_252
SMB_63
SMB_252
adj_volume
MARKET_63
MARKET_252
BB_SQUEEZE
ATR



To inspect the correlation matrix, let's isolate the unique values from the corrlation matrix, excluding the diagonal:

In [13]:
```
corr_ = corr_common.stack().reset_index() # move column labels to rows as 2nd MultiIndex level
corr_.columns = ['x1', 'x2', 'rho']
corr_ = corr_[corr_.x1 != corr_.x2].drop_duplicates('rho')
```

Top ten most correlated features

Select highest and lowest five correlation values:

In [14]:
```
corr_.nlargest(5, columns='rho').append(corr_.nsmallest(5, columns='rho'))
```

Out[14]:

	x1	x2	rho
111	adj_open	adj_high	0.999791
167	adj_high	adj_close	0.999790
221	adj_low	adj_close	0.999779
112	adj_open	adj_low	0.999762
113	adj_open	adj_close	0.999636
1057	HT	RSI	-0.923128
613	ret_21	HT	-0.867788
559	ret_10	HT	-0.827081
1061	HT	WILLR	-0.826688
1052	HT	CCI	-0.805418

Forward return correlation

Which features are most correlated with the forward returns?

In [15]:
```
fwd_corr = factors.drop(['ret_fwd'], axis=1).corrwith(factors.ret_fwd, method='spearman')
```

In [16]:
```
fwd_corr = fwd_corr.dropna()
fwd_corr.to_csv('forward_correlation.csv')
```

The features that are more correlated with the outome are more likely to help predicting it.

In [17]:
```
top50 = fwd_corr.abs().nlargest(25).index
fwd_corr.loc[top50].sort_values().plot.barh(figsize=(8, 10),
                                            legend=False);
```

Mutual Information

In [18]:

```
mi = {}
for feature in features:
    print(feature)
    df = (factors
          .loc[:, ['ret_fwd', feature]]
          .dropna()
          .sample(n=100000)) # if it takes too long or you run into resource constraints, reduce the sample size
    discrete_features = df[feature].nunique() < 20
    mi[feature] = mutual_info_regression(X=df[[feature]],
                                         y=df.ret_fwd,
                                         discrete_features=discrete_features)[0]
```

AARONOSC
AD
ADOSC
ADX
ADXR
ALPHA_252
ALPHA_63
ATR
BB_LOW
BB_SQUEEZE
BB_UP
BOP
CCI
CMA_252
CMA_63
HML_252
HML_63
HT
MACD
MACD_HIST
MACD_SIGNAL
MARKET_252
MARKET_63
MFI
OBV
PPO
RMW_252
RMW_63
RSI
SAR
SMB_252
SMB_63
STOCH
STOCHRSI
ULTOSC
WILLR
adj_close
adj_high
adj_low
adj_open

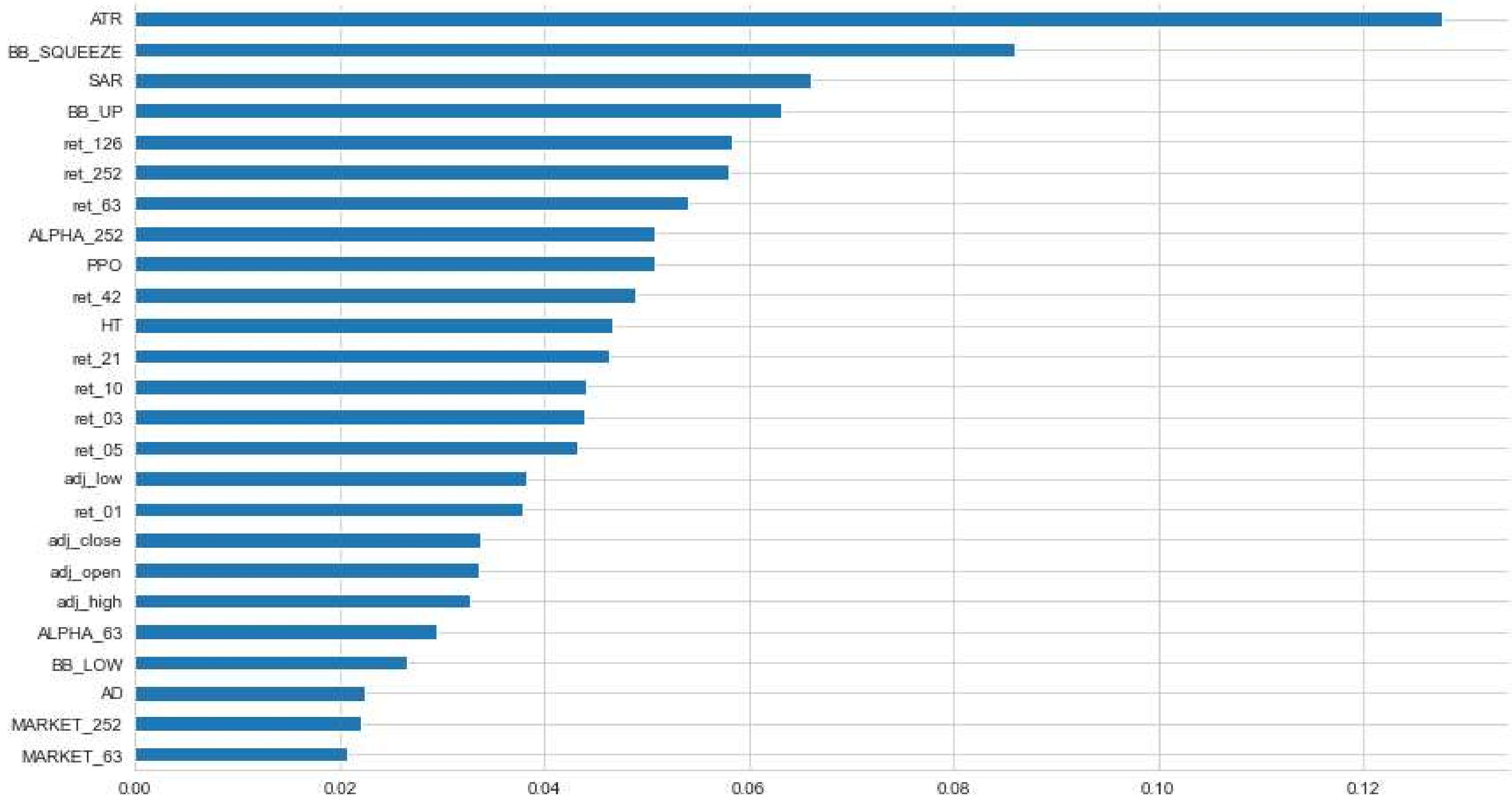
```
adj_volume
ex-dividend
month
ret_01
ret_03
ret_05
ret_10
ret_126
ret_21
ret_252
ret_42
ret_63
split_ratio
weekday
```

In [19]:

```python
mi = pd.Series(mi)
mi.to_csv('mutual_info.csv')
```

In [ ]:

```python
mi.nlargest(25).sort_values().plot.barh(figsize=(14,8))
sns.despine();
```



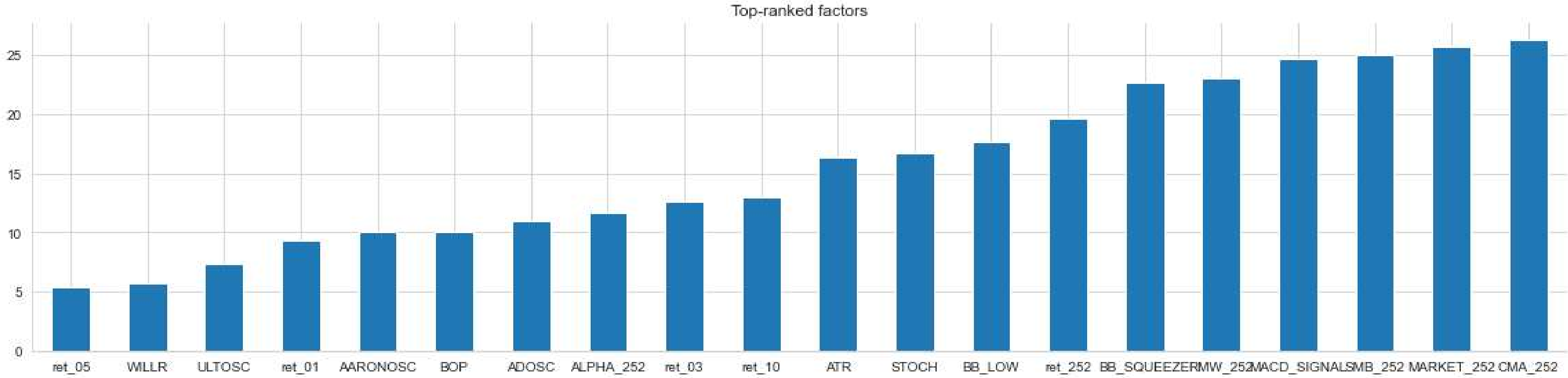
Comparison

In [ ]:
```
top_ranked = stats.abs().rank(ascending=False).mean(1)
```

In [2]:
```
#Note: feature based on mutual information did not used in the final tuning. Also features selection changed for the
```

In [ ]:
```
top_ranked.to_csv('top_features.csv')
```

In [ ]:
```
top_ranked.drop(categoricals).nsmallest(20).plot.bar(figsize=(16, 4), rot=0, title='Top-ranked factors')
sns.despine()
plt.tight_layout();
```



In [ ]: