Generating trading signals with Linear Regression

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

In [2]:
```
%matplotlib inline

from pathlib import Path
import sys, os
from time import time
from collections import defaultdict
from itertools import product

import numpy as np
import pandas as pd
import statsmodels.api as sm

from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import MinMaxScaler

from scipy.stats import spearmanr

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

Settings

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

In [4]:
```
np.random.seed(42)
```

In [5]:
```
idx = pd.IndexSlice
```

In [6]:
```
YEAR = 252 # days
MONTH = 21 # days
```

In [7]:
```
def format_time(t):
    """Return a formatted time string 'HH:MM:SS
    based on a numeric time() value"""
    m, s = divmod(t, 60)
    h, m = divmod(m, 60)
    return f'{h:0>2.0f}:{m:0>2.0f}:{s:0>2.0f}'
```

Get Data

In [10]:
```
DATA_PATH = Path('data')
```

In [11]:
```
data = (pd.read_hdf(DATA_PATH / 'stock_prices.h5', 'model_data')
        .sort_index())
```

In [12]:
```
data.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
```

In [13]:
```
target = 'ret_fwd'
features = data.columns.drop(target)
```

In [14]:
```
categoricals = ['month', 'weekday']
```

In [15]:
```
class MultipleTimeSeriesCV:
    """Generates tuples of train_idx, test_idx pairs
    Assumes the MultiIndex contains levels 'symbol' and 'date'
    purges overlapping outcomes"""

    def __init__(self,
                 n_splits=3,
                 train_period_length=126,
                 test_period_length=21,
                 lookahead=None,
                 date_idx='date',
```

```
                 shuffle=False):
        self.n_splits = n_splits
        self.lookahead = lookahead
        self.test_length = test_period_length
        self.train_length = train_period_length
        self.shuffle = shuffle
        self.date_idx = date_idx

    def split(self, X, y=None, groups=None):
        unique_dates = X.index.get_level_values(self.date_idx).unique()
        days = sorted(unique_dates, reverse=True)
        split_idx = []
        for i in range(self.n_splits):
            test_end_idx = i * self.test_length
            test_start_idx = test_end_idx + self.test_length
            train_end_idx = test_start_idx + self.lookahead - 1
            train_start_idx = train_end_idx + self.train_length + self.lookahead - 1
            split_idx.append([train_start_idx, train_end_idx,
                              test_start_idx, test_end_idx])

        dates = X.reset_index()[[self.date_idx]]
        for train_start, train_end, test_start, test_end in split_idx:

            train_idx = dates[(dates[self.date_idx] > days[train_start])
                              & (dates.date <= days[train_end])].index
            test_idx = dates[(dates.date > days[test_start])
                             & (dates.date <= days[test_end])].index
            if self.shuffle:
                np.random.shuffle(list(train_idx))
            yield train_idx.to_numpy(), test_idx.to_numpy()

    def get_n_splits(self, X, y, groups=None):
        return self.n_splits
```

In [16]:
```
train_length = 5 * YEAR
test_length = 3 * MONTH
```

In [17]:
```
lookahead = 1
```

In [18]:
```
n_splits = 16
```

In [19]:
```
cv = MultipleTimeSeriesCV(n_splits=n_splits,
                          test_period_length=test_length,
                          lookahead=lookahead,
                          train_period_length=train_length)
```

In [20]:
```
for n_split, (train_idx, test_idx) in enumerate(cv.split(X=data)):
    train = data.iloc[train_idx]
    train_dates = train.index.get_level_values('date')

    test = data.iloc[test_idx]
    test_dates = test.index.get_level_values('date')

    train_days = train.groupby(level="ticker").size().value_counts().index[0]
    train_start, train_end = train_dates.min().date(), train_dates.max().date()

    test_days = test.groupby(level="ticker").size().value_counts().index[0]
    test_start, test_end = test_dates.min().date(), test_dates.max().date()

    print(f'Split: {n_split:02} | # Train: {train_start} - {train_end} ({train_days:5,.0f} days) | '
          f'Test: {test_start} - {test_end} ({test_days} days)')
```

```
Split: 00 | # Train: 2011-11-08 - 2016-09-29 (1,260 days) | Test: 2016-09-30 - 2016-12-29 (63 days)
Split: 01 | # Train: 2011-08-11 - 2016-06-30 (1,260 days) | Test: 2016-07-01 - 2016-09-29 (63 days)
Split: 02 | # Train: 2011-05-16 - 2016-04-01 (1,260 days) | Test: 2016-04-04 - 2016-06-30 (63 days)
Split: 03 | # Train: 2011-02-14 - 2015-12-30 (1,260 days) | Test: 2015-12-31 - 2016-04-01 (63 days)
Split: 04 | # Train: 2010-11-17 - 2015-09-30 (1,260 days) | Test: 2015-10-01 - 2015-12-30 (63 days)
Split: 05 | # Train: 2010-08-20 - 2015-07-01 (1,260 days) | Test: 2015-07-02 - 2015-09-30 (63 days)
Split: 06 | # Train: 2010-05-25 - 2015-04-01 (1,260 days) | Test: 2015-04-02 - 2015-07-01 (63 days)
Split: 07 | # Train: 2010-02-25 - 2014-12-30 (1,260 days) | Test: 2014-12-31 - 2015-04-01 (63 days)
Split: 08 | # Train: 2009-11-26 - 2014-09-30 (1,260 days) | Test: 2014-10-01 - 2014-12-30 (63 days)
Split: 09 | # Train: 2009-08-31 - 2014-07-03 (1,260 days) | Test: 2014-07-04 - 2014-09-30 (63 days)
Split: 10 | # Train: 2009-06-03 - 2014-04-07 (1,260 days) | Test: 2014-04-08 - 2014-07-03 (63 days)
Split: 11 | # Train: 2009-03-06 - 2014-01-08 (1,260 days) | Test: 2014-01-09 - 2014-04-07 (63 days)
Split: 12 | # Train: 2008-12-09 - 2013-10-11 (1,260 days) | Test: 2013-10-14 - 2014-01-08 (63 days)
Split: 13 | # Train: 2008-09-11 - 2013-07-16 (1,260 days) | Test: 2013-07-17 - 2013-10-11 (63 days)
Split: 14 | # Train: 2008-06-16 - 2013-04-18 (1,260 days) | Test: 2013-04-19 - 2013-07-16 (63 days)
Split: 15 | # Train: 2008-03-19 - 2013-01-21 (1,260 days) | Test: 2013-01-22 - 2013-04-18 (63 days)
```

Baseline: Linear Regression

In [21]:
```
df = (pd.get_dummies(data, columns=categoricals, drop_first=True)
      .dropna()
      .sort_index())

X = df.drop(target, axis=1)
y = df[target]
```

Run cross-validation

In [22]:
```
lr = LinearRegression()
```

In [23]:
```
%%time
lr_preds = []
for i, (train_idx, test_idx) in enumerate(cv.split(X=X)):
    X_train, y_train = X.iloc[train_idx], y.iloc[train_idx]
    X_test, y_test = X.iloc[test_idx], y.iloc[test_idx]
    lr.fit(X_train, y_train)
    y_pred = lr.predict(X_test)
    lr_preds.append(y_test.to_frame('y_true').assign(y_pred=y_pred))
lr_preds = pd.concat(lr_preds)
```

```
Wall time: 21.3 s
```

In [24]:
```
lr_preds.to_hdf(DATA_PATH / 'predictions.h5', 'linear_regression')
```

Evaluate Predictions using the Information Coefficient

In [25]:
```
lr_ic, p = spearmanr(lr_preds.y_true, lr_preds.y_pred)
f'{lr_ic:.4f} ({p:.2%})'
```

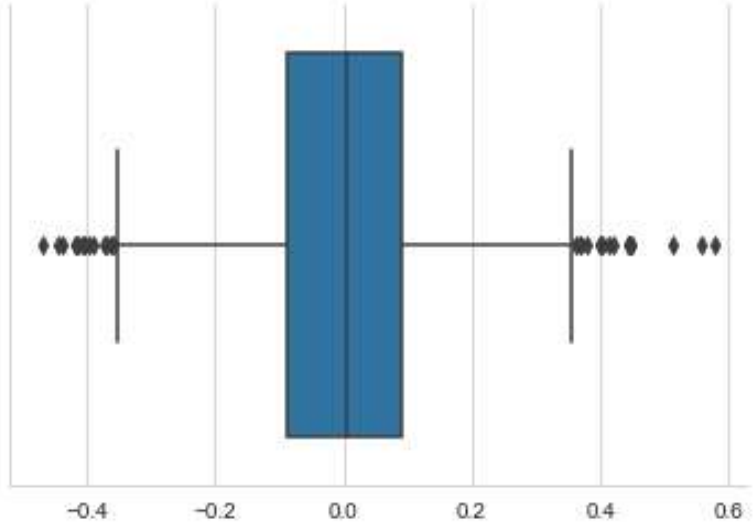
Out[25]:
```
'0.0221 (0.00%)'
```

In [26]:
```
daily_ic_lr = lr_preds.groupby(level='date').apply(lambda x: spearmanr(x.y_true, x.y_pred)[0])
```

In [27]:
```
f'Daily IC - Mean: {daily_ic_lr.mean():.4f} | CV: {daily_ic_lr.std()/daily_ic_lr.mean():.2f}'
```

Out[27]:
```
'Daily IC - Mean: 0.0026 | CV: 58.01'
```

In [28]:
```
sns.boxplot(daily_ic_lr)
sns.despine();
```



Regularized Linear Regression

In [29]:
```
ridge_alphas = np.logspace(-5, 15, 21)[::2]
```

In [30]:
```
ridge_alphas
```

Out[30]:
```
array([1.e-05, 1.e-03, 1.e-01, 1.e+01, 1.e+03, 1.e+05, 1.e+07, 1.e+09,
       1.e+11, 1.e+13, 1.e+15])
```

Ridge Regression

In [31]:
```
%%time
```

```
ridge_preds = []
scaler = MinMaxScaler()
for i, (train_idx, test_idx) in enumerate(cv.split(X=X)):
    print(f'{i}', end=' ', flush=True)
    X_train, y_train = X.iloc[train_idx], y.iloc[train_idx]
    X_train = scaler.fit_transform(X_train)
    X_test, y_test = X.iloc[test_idx], y.iloc[test_idx]
    X_test = scaler.transform(X_test)
    cv_predictions = []
    for alpha in ridge_alphas:
        ridge = Ridge(alpha=alpha)
        ridge.fit(X_train, y_train)
        y_pred = ridge.predict(X_test)
        cv_predictions.append(pd.DataFrame({alpha: y_pred},
                                           index=y_test.index))
    ridge_preds.append(pd.concat(cv_predictions, axis=1).assign(y_test=y_test))
ridge_preds = pd.concat(ridge_preds)
print()
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Wall time: 1min 11s
```

Evaluate Predictions

In [32]:
```
ic_by_day_ridge = {}
for alpha in ridge_alphas:
    ic_by_day_ridge[alpha] = (ridge_preds.groupby(level='date')
                              .apply(lambda x: spearmanr(x[alpha], x.y_test)[0])
                              .mean())
```

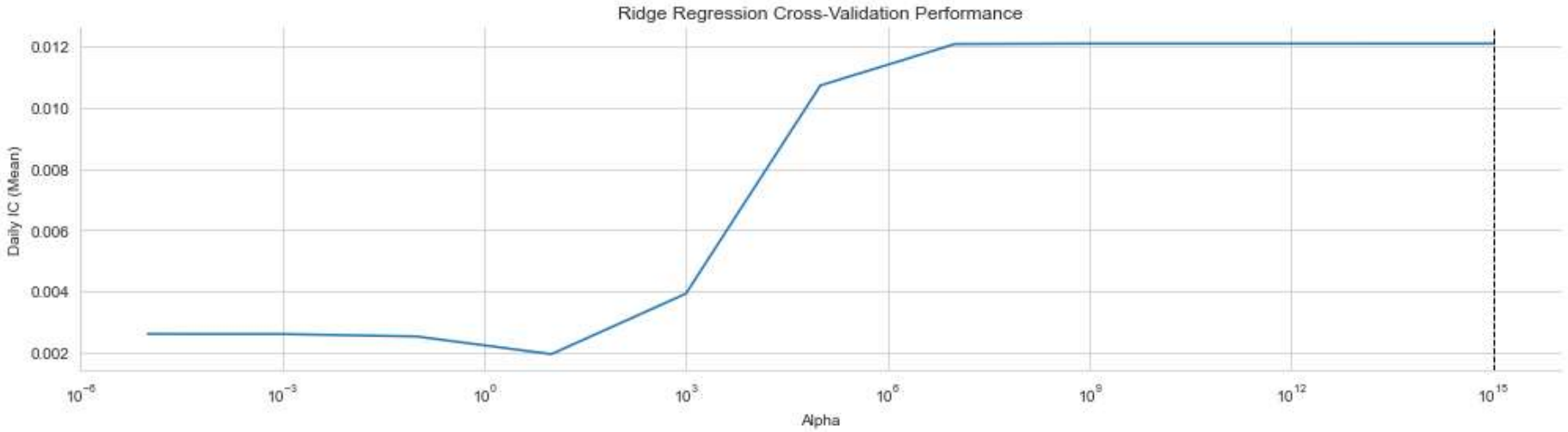
In [33]:
```
best_alpha_ridge = pd.Series(ic_by_day_ridge).idxmax()
```

In [34]:
```
best_ic_by_day_ridge = (ridge_preds.groupby(level='date')
                        .apply(lambda x: spearmanr(x[best_alpha_ridge], x.y_test)[0]))
```

In [35]:
```
print(f'Daily IC - Mean: {best_ic_by_day_ridge.mean():.4f} | '
      f'CV: {best_ic_by_day_ridge.std()/best_ic_by_day_ridge.mean():.2f}')
```

```
Daily IC - Mean: 0.0121 | CV: 12.22
```

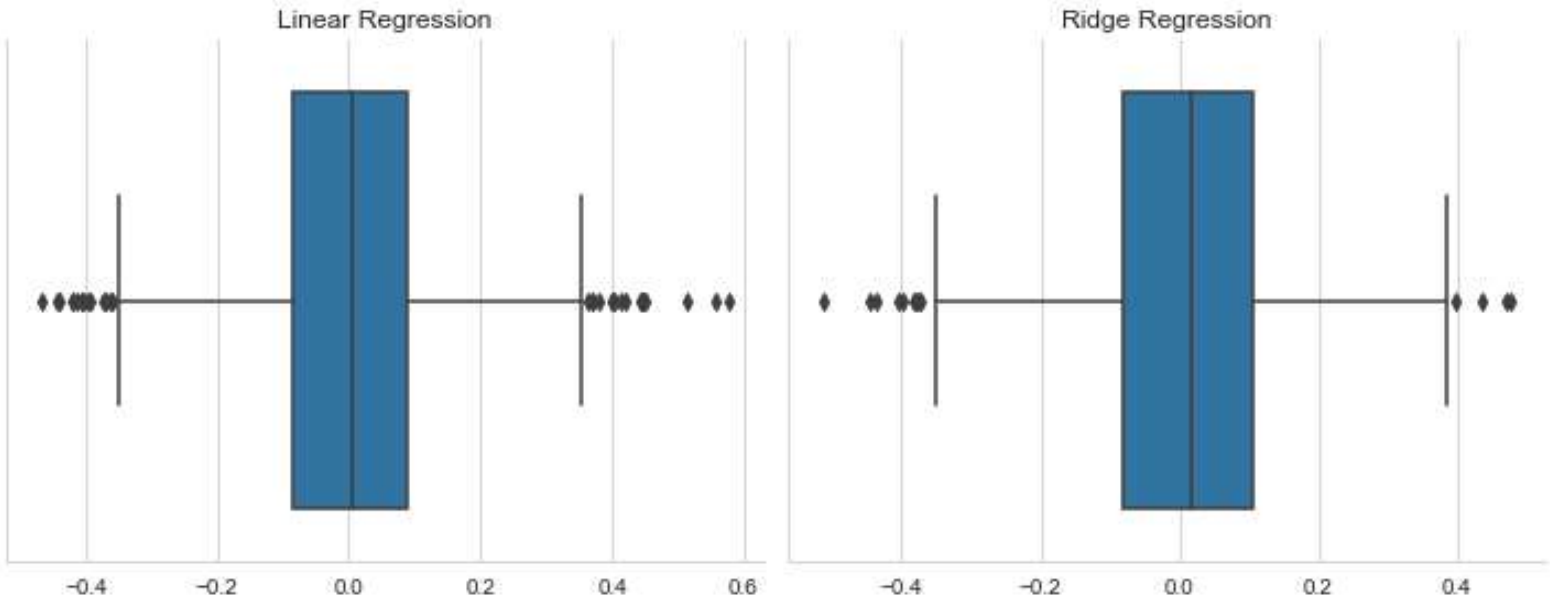
In [36]:
```
ax = pd.Series(ic_by_day_ridge).plot(figsize=(14, 4),
                                     logx=True,
                                     title='Ridge Regression Cross-Validation Performance')
ax.axvline(best_alpha_ridge, c='k', ls='--', lw=1)
ax.set_ylabel('Daily IC (Mean)')
ax.set_xlabel('Alpha')
sns.despine()
plt.tight_layout();
```



In [37]:
```
fig, axes = plt.subplots(ncols=2, figsize=(10, 4), sharey=True)

sns.boxplot(daily_ic_lr, ax=axes[0], orient='v')
axes[0].set_title('Linear Regression')

ridge_daily_ic = ridge_preds.groupby(level='date').apply(lambda x: spearmanr(x[best_alpha_ridge], x.y_test)[0])
sns.boxplot(ridge_daily_ic, ax=axes[1], orient='v')
axes[0].set_title('Linear Regression')
axes[1].set_title('Ridge Regression')
sns.despine()
fig.tight_layout();
```



Persist results

In [38]:
```
ridge_preds[best_alpha_ridge].to_hdf(DATA_PATH / 'predictions.h5', 'ridge_regression')
```

Lasso Regression

In [39]:
```
lasso_alphas = np.logspace(-15, -5, 11)
```

In [40]:
```
lasso_preds = []
scaler = MinMaxScaler()
for i, (train_idx, test_idx) in enumerate(cv.split(X=X)):
    print(f'\nFold {i}')
    X_train, y_train = X.iloc[train_idx], y.iloc[train_idx]
    X_test, y_test = X.iloc[test_idx], y.iloc[test_idx]
    X_train = scaler.fit_transform(X_train)
    X_test, y_test = X.iloc[test_idx], y.iloc[test_idx]
    X_test = scaler.transform(X_test)
    cv_predictions = []
    for alpha in lasso_alphas:
```

```
        print(f'{alpha:.0e}', end=' ', flush=True)
        lasso = Lasso(alpha=alpha)
        lasso.fit(X_train, y_train)
        y_pred = lasso.predict(X_test)
        cv_predictions.append(pd.DataFrame({alpha: y_pred},
                                           index=y_test.index))
    lasso_preds.append(pd.concat(cv_predictions, axis=1).assign(y_test=y_test))
lasso_preds = pd.concat(lasso_preds)
```

```
Fold 0
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 1
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 2
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 3
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 4
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 5
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 6
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 7
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 8
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 9
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 10
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 11
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 12
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 13
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 14
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
Fold 15
1e-15 1e-14 1e-13 1e-12 1e-11 1e-10 1e-09 1e-08 1e-07 1e-06 1e-05
```

Evaluate Predictions

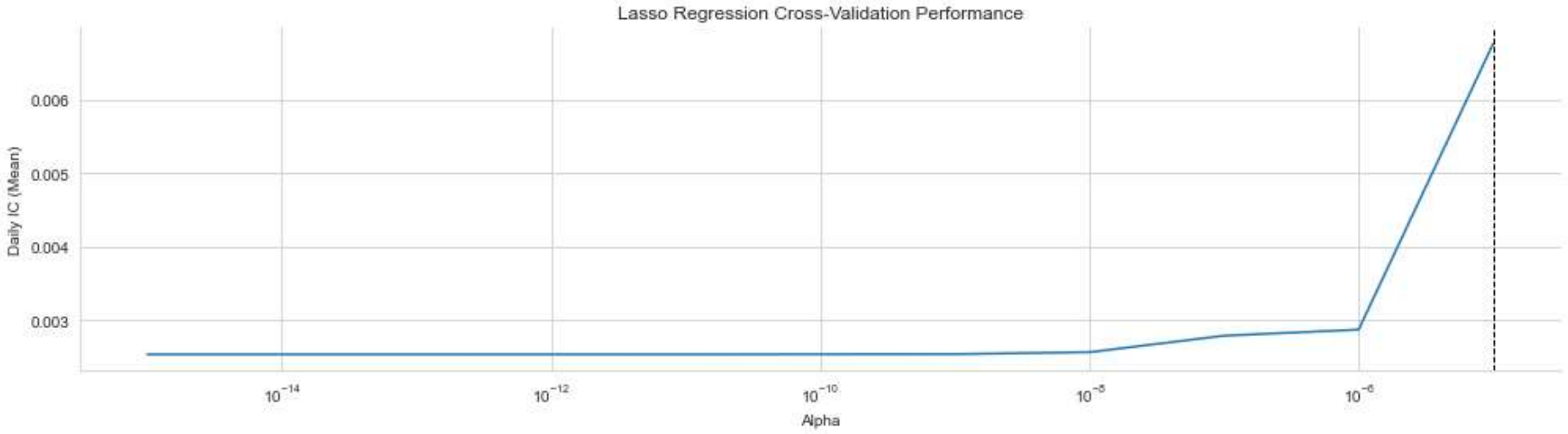
In [41]:

```
ic_by_day_lasso = {}
for alpha in lasso_alphas:
    ic_by_day_lasso[alpha] = (lasso_preds.groupby(level='date')
                              .apply(lambda x: spearmanr(x[alpha], x.y_test)[0]).mean())
```

In [42]:
```
pd.Series(ic_by_day_lasso)
```

Out[42]:
```
1.000000e-15    0.002540
1.000000e-14    0.002540
1.000000e-13    0.002540
1.000000e-12    0.002540
1.000000e-11    0.002540
1.000000e-10    0.002541
1.000000e-09    0.002543
1.000000e-08    0.002570
1.000000e-07    0.002794
1.000000e-06    0.002876
1.000000e-05    0.006765
dtype: float64
```

In [43]:
```
ax = pd.Series(ic_by_day_lasso).plot(figsize=(14, 4),
                                     logx=True,
                                     title='Lasso Regression Cross-Validation Performance')
ax.axvline(pd.Series(ic_by_day_lasso).idxmax(), c='k', ls='--', lw=1)
ax.set_ylabel('Daily IC (Mean)')
ax.set_xlabel('Alpha')
sns.despine()
plt.tight_layout();
```



In [ ]: