05_predicting_stock_returns_with_linear_regression

September 29, 2021

1 Prediction stock returns with linear regression

1.1 Imports & Settings

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

[2]:
```
%matplotlib inline

from time import time
from pathlib import Path
import pandas as pd
import numpy as np
from scipy.stats import spearmanr

from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.pipeline import Pipeline

import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
```

[3]:
```
sns.set_style('darkgrid')
idx = pd.IndexSlice
```

[4]:
```
YEAR = 252
```

1.2 Load Data

[5]:
```
with pd.HDFStore('data.h5') as store:
    data = (store['model_data']
            .dropna()
            .drop(['open', 'close', 'low', 'high'], axis=1))
```

[6]:
```
data.index.names = ['symbol', 'date']
```

[7]:
```
data = data.drop([c for c in data.columns if 'lag' in c], axis=1)
```

1.2.1 Select Investment Universe

[8]:
```
data = data[data.dollar_vol_rank<100]
```

[9]:
```
data.info(null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 109675 entries, ('AAL', Timestamp('2013-07-25 00:00:00')) to ('ZTS',
Timestamp('2014-12-04 00:00:00'))
Data columns (total 45 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   volume               109675 non-null  float64
 1   dollar_vol           109675 non-null  float64
 2   dollar_vol_1m        109675 non-null  float64
 3   dollar_vol_rank      109675 non-null  float64
 4   rsi                  109675 non-null  float64
 5   bb_high              109675 non-null  float64
 6   bb_low               109675 non-null  float64
 7   atr                  109675 non-null  float64
 8   macd                 109675 non-null  float64
 9   return_1d            109675 non-null  float64
 10  return_5d            109675 non-null  float64
 11  return_10d           109675 non-null  float64
 12  return_21d           109675 non-null  float64
 13  return_42d           109675 non-null  float64
 14  return_63d           109675 non-null  float64
 15  target_1d            109675 non-null  float64
 16  target_5d            109675 non-null  float64
 17  target_10d           109675 non-null  float64
 18  target_21d           109675 non-null  float64
 19  year_2014            109675 non-null  uint8
 20  year_2015            109675 non-null  uint8
 21  year_2016            109675 non-null  uint8
 22  year_2017            109675 non-null  uint8
 23  month_2              109675 non-null  uint8
 24  month_3              109675 non-null  uint8
 25  month_4              109675 non-null  uint8
 26  month_5              109675 non-null  uint8
 27  month_6              109675 non-null  uint8
 28  month_7              109675 non-null  uint8
 29  month_8              109675 non-null  uint8
 30  month_9              109675 non-null  uint8
 31  month_10             109675 non-null  uint8
 32  month_11             109675 non-null  uint8
```

```
 33   month_12                109675 non-null  uint8
 34   capital_goods           109675 non-null  uint8
 35   consumer_durables       109675 non-null  uint8
 36   consumer_non-durables   109675 non-null  uint8
 37   consumer_services       109675 non-null  uint8
 38   energy                  109675 non-null  uint8
 39   finance                 109675 non-null  uint8
 40   health_care             109675 non-null  uint8
 41   miscellaneous           109675 non-null  uint8
 42   public_utilities        109675 non-null  uint8
 43   technology              109675 non-null  uint8
 44   transportation          109675 non-null  uint8
dtypes: float64(19), uint8(26)
memory usage: 19.8+ MB
```

1.2.2 Create Model Data

[10]:
```
y = data.filter(like='target')
X = data.drop(y.columns, axis=1)
X = X.drop(['dollar_vol', 'dollar_vol_rank', 'volume', 'consumer_durables'],␣
↪axis=1)
```

1.3 Custom MultipleTimeSeriesCV

[11]:
```
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,
                 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

    def split(self, X, y=None, groups=None):
        unique_dates = X.index.get_level_values('date').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()[['date']]
        for train_start, train_end, test_start, test_end in split_idx:
            train_idx = dates[(dates.date > 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, test_idx

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

1.3.1 Verify that it works

[12]:
```
train_period_length = 63
test_period_length = 10
n_splits = int(3 * YEAR/test_period_length)
lookahead =1

cv = MultipleTimeSeriesCV(n_splits=n_splits,
                          test_period_length=test_period_length,
                          lookahead=lookahead,
                          train_period_length=train_period_length)
```

[13]:
```
i = 0
for train_idx, test_idx in 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')
    df = train.reset_index().append(test.reset_index())
    n = len(df)
    assert n== len(df.drop_duplicates())
    print(train.groupby(level='symbol').size().value_counts().index[0],
          train_dates.min().date(), train_dates.max().date(),
          test.groupby(level='symbol').size().value_counts().index[0],
          test_dates.min().date(), test_dates.max().date())
```

```
    i += 1
    if i == 10:
        break
```

```
63 2017-08-16 2017-11-14 10 2017-11-15 2017-11-29
63 2017-08-02 2017-10-30 10 2017-10-31 2017-11-14
63 2017-07-19 2017-10-16 10 2017-10-17 2017-10-30
63 2017-07-05 2017-10-02 10 2017-10-03 2017-10-16
63 2017-06-20 2017-09-18 10 2017-09-19 2017-10-02
63 2017-06-06 2017-09-01 10 2017-09-05 2017-09-18
63 2017-05-22 2017-08-18 10 2017-08-21 2017-09-01
63 2017-05-08 2017-08-04 10 2017-08-07 2017-08-18
63 2017-04-24 2017-07-21 10 2017-07-24 2017-08-04
62 2017-04-10 2017-07-07 10 2017-07-10 2017-07-21
```

1.4 Visualization helper functions

1.4.1 Prediction vs Actual Scatter Plot

[14]:
```
def plot_preds_scatter(df, ticker=None):
    if ticker is not None:
        idx = pd.IndexSlice
        df = df.loc[idx[ticker, :], :]
    j = sns.jointplot(x='predicted', y='actuals',
                      robust=True, ci=None,
                      line_kws={'lw': 1, 'color': 'k'},
                      scatter_kws={'s': 1},
                      data=df,
                      kind='reg')
    j.ax_joint.yaxis.set_major_formatter(
        FuncFormatter(lambda y, _: '{:.1%}'.format(y)))
    j.ax_joint.xaxis.set_major_formatter(
        FuncFormatter(lambda x, _: '{:.1%}'.format(x)))
    j.ax_joint.set_xlabel('Predicted')
    j.ax_joint.set_ylabel('Actuals')
```

1.4.2 Daily IC Distribution

[15]:
```
def plot_ic_distribution(df, ax=None):
    if ax is not None:
        sns.distplot(df.ic, ax=ax)
    else:
        ax = sns.distplot(df.ic)
    mean, median = df.ic.mean(), df.ic.median()
    ax.axvline(0, lw=1, ls='--', c='k')
    ax.text(x=.05, y=.9,
            s=f'Mean: {mean:8.2f}\nMedian: {median:5.2f}',
```

```
            horizontalalignment='left',
            verticalalignment='center',
            transform=ax.transAxes)
    ax.set_xlabel('Information Coefficient')
    sns.despine()
    plt.tight_layout()
```

1.4.3 Rolling Daily IC

[16]:
```
def plot_rolling_ic(df):
    fig, axes = plt.subplots(nrows=2, sharex=True, figsize=(14, 8))
    rolling_result = df.sort_index().rolling(21).mean().dropna()
    mean_ic = df.ic.mean()
    rolling_result.ic.plot(ax=axes[0],
                           title=f'Information Coefficient (Mean: {mean_ic:.
 ↪2f})',
                           lw=1)
    axes[0].axhline(0, lw=.5, ls='-', color='k')
    axes[0].axhline(mean_ic, lw=1, ls='--', color='k')

    mean_rmse = df.rmse.mean()
    rolling_result.rmse.plot(ax=axes[1],
                             title=f'Root Mean Squared Error (Mean: {mean_rmse:.
 ↪2%})',
                             lw=1,
                             ylim=(0, df.rmse.max()))
    axes[1].axhline(df.rmse.mean(), lw=1, ls='--', color='k')
    sns.despine()
    plt.tight_layout()
```

1.5 Linear Regression with sklearn

1.5.1 Set up cross-validation

[17]:
```
train_period_length = 63
test_period_length = 10
n_splits = int(3 * YEAR / test_period_length)
lookahead = 1

cv = MultipleTimeSeriesCV(n_splits=n_splits,
                          test_period_length=test_period_length,
                          lookahead=lookahead,
                          train_period_length=train_period_length)
```

1.5.2 Run cross-validation with LinearRegression

[18]:
```
%%time
target = f'target_{lookahead}d'
lr_predictions, lr_scores = [], []
lr = LinearRegression()
for i, (train_idx, test_idx) in enumerate(cv.split(X), 1):
    X_train, y_train, = X.iloc[train_idx], y[target].iloc[train_idx]
    X_test, y_test = X.iloc[test_idx], y[target].iloc[test_idx]
    lr.fit(X=X_train, y=y_train)
    y_pred = lr.predict(X_test)

    preds = y_test.to_frame('actuals').assign(predicted=y_pred)
    preds_by_day = preds.groupby(level='date')
    scores = pd.concat([preds_by_day.apply(lambda x: spearmanr(x.predicted,
                                                                 x.actuals)[0] *␣
 ↪100)
                        .to_frame('ic'),
                        preds_by_day.apply(lambda x: np.
 ↪sqrt(mean_squared_error(y_pred=x.predicted,
                                                                              ␣
 ↪y_true=x.actuals)))
                        .to_frame('rmse')], axis=1)

    lr_scores.append(scores)
    lr_predictions.append(preds)

lr_scores = pd.concat(lr_scores)
lr_predictions = pd.concat(lr_predictions)
```

```
CPU times: user 4.14 s, sys: 0 ns, total: 4.14 s
Wall time: 1.5 s
```

1.5.3 Persist results

[19]:
```
lr_scores.to_hdf('data.h5', 'lr/scores')
lr_predictions.to_hdf('data.h5', 'lr/predictions')
```

[20]:
```
lr_scores = pd.read_hdf('data.h5', 'lr/scores')
lr_predictions = pd.read_hdf('data.h5', 'lr/predictions')
```

1.5.4 Evaluate results

[21]:
```
lr_r, lr_p = spearmanr(lr_predictions.actuals, lr_predictions.predicted)
print(f'Information Coefficient (overall): {lr_r:.3%} (p-value: {lr_p:.4%})')
```

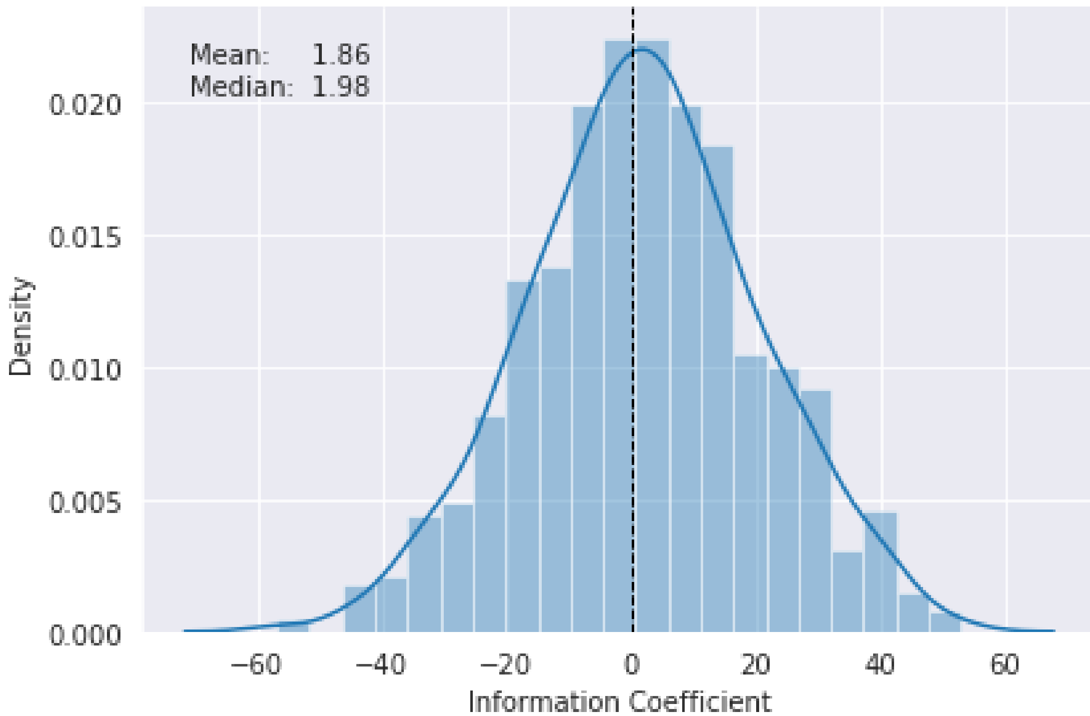
```
Information Coefficient (overall): 1.531% (p-value: 0.0031%)
```

Prediction vs Actuals Scatter

[22]:
```
plot_preds_scatter(lr_predictions)
```

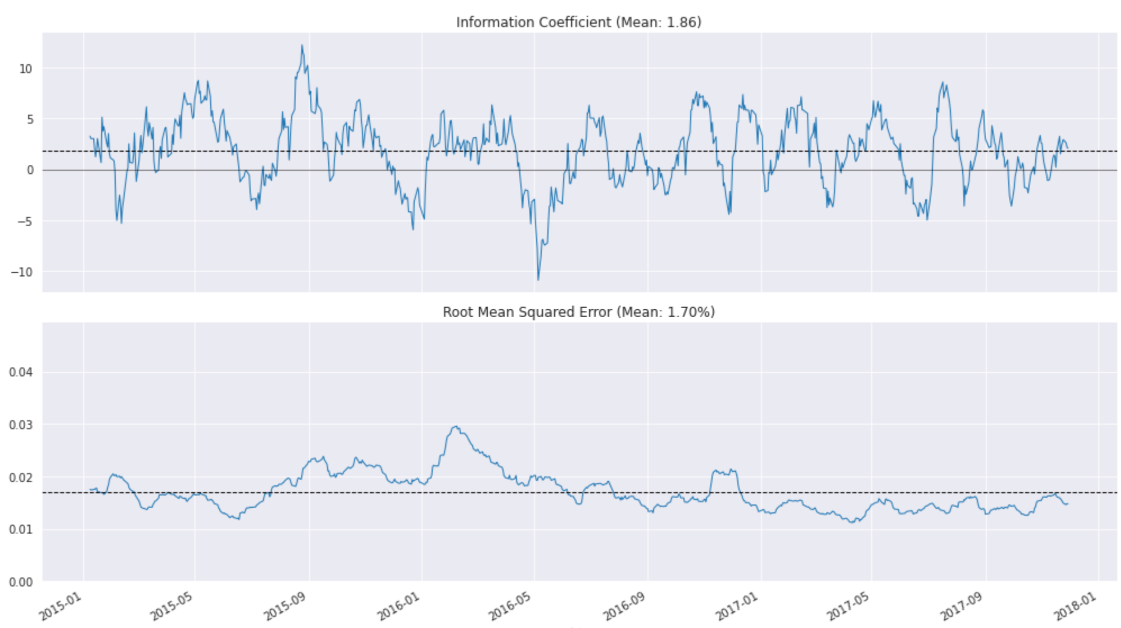
Daily IC Distribution

[23]:
```
plot_ic_distribution(lr_scores)
```



Rolling Daily IC

[24]:
```
plot_rolling_ic(lr_scores)
```



1.6 Ridge Regression

1.6.1 Define cross-validation parameters

[25]:
```
ridge_alphas = np.logspace(-4, 4, 9)
ridge_alphas = sorted(list(ridge_alphas) + list(ridge_alphas * 5))
```

[26]:
```
n_splits = int(3 * YEAR/test_period_length)
train_period_length = 63
test_period_length = 10
lookahead = 1

cv = MultipleTimeSeriesCV(n_splits=n_splits,
                          test_period_length=test_period_length,
                          lookahead=lookahead,
                          train_period_length=train_period_length)
```

1.6.2 Run cross-validation

[27]:
```
target = f'target_{lookahead}d'

X = X.drop([c for c in X.columns if 'year' in c], axis=1)
```

[28]:
```
%%time
ridge_coeffs, ridge_scores, ridge_predictions = {}, [], []

for alpha in ridge_alphas:
    print(alpha, end=' ', flush=True)
    start = time()
    model = Ridge(alpha=alpha,
                  fit_intercept=False,
                  random_state=42)

    pipe = Pipeline([
        ('scaler', StandardScaler()),
        ('model', model)])

    coeffs = []
    for i, (train_idx, test_idx) in enumerate(cv.split(X), 1):
        X_train, y_train, = X.iloc[train_idx], y[target].iloc[train_idx]
        X_test, y_test = X.iloc[test_idx], y[target].iloc[test_idx]

        pipe.fit(X=X_train, y=y_train)
        y_pred = pipe.predict(X_test)

        preds = y_test.to_frame('actuals').assign(predicted=y_pred)
        preds_by_day = preds.groupby(level='date')
        scores = pd.concat([preds_by_day.apply(lambda x: spearmanr(x.predicted,
```

```
                                                                x.
↪actuals)[0] * 100)
                              .to_frame('ic'),
                              preds_by_day.apply(lambda x: np.
↪sqrt(mean_squared_error(y_pred=x.predicted,
                                                                        ␣
↪    y_true=x.actuals)))
                              .to_frame('rmse')], axis=1)

        ridge_scores.append(scores.assign(alpha=alpha))
        ridge_predictions.append(preds.assign(alpha=alpha))

        coeffs.append(pipe.named_steps['model'].coef_)
    ridge_coeffs[alpha] = np.mean(coeffs, axis=0)

print('\n')
```

```
0.0001 0.0005 0.001 0.005 0.01 0.05 0.1 0.5 1.0 5.0 10.0 50.0 100.0 500.0 1000.0
5000.0 10000.0 50000.0

CPU times: user 1min 17s, sys: 73.8 ms, total: 1min 17s
Wall time: 30.7 s
```

1.6.3 Persist results

[29]:
```
ridge_scores = pd.concat(ridge_scores)
ridge_scores.to_hdf('data.h5', 'ridge/scores')

ridge_coeffs = pd.DataFrame(ridge_coeffs, index=X.columns).T
ridge_coeffs.to_hdf('data.h5', 'ridge/coeffs')

ridge_predictions = pd.concat(ridge_predictions)
ridge_predictions.to_hdf('data.h5', 'ridge/predictions')
```

[30]:
```
ridge_scores = pd.read_hdf('data.h5', 'ridge/scores')
ridge_coeffs = pd.read_hdf('data.h5', 'ridge/coeffs')
ridge_predictions = pd.read_hdf('data.h5', 'ridge/predictions')
```

1.6.4 Evaluate Ridge Results

[31]:
```
ridge_r, ridge_p = spearmanr(ridge_predictions.actuals, ridge_predictions.
↪predicted)
print(f'Information Coefficient (overall): {ridge_r:.3%} (p-value: {ridge_p:.
↪4%})')
```

```
Information Coefficient (overall): 1.551% (p-value: 0.0000%)
```

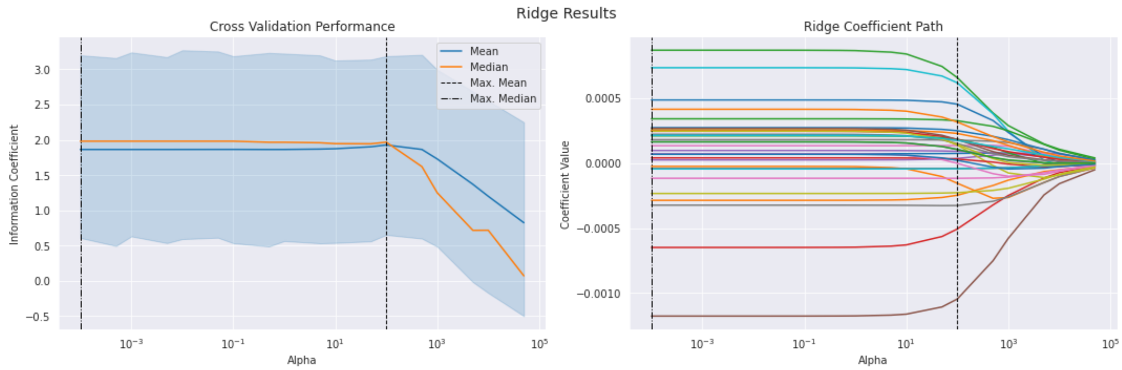
[32]:
```python
ridge_scores.groupby('alpha').ic.describe()
```

[32]:
```
            count      mean        std        min        25%       50%  \
alpha
0.0001      750.0  1.863889  18.565640 -56.788054 -10.005566  1.981447
0.0005      750.0  1.863889  18.565640 -56.788054 -10.005566  1.981447
0.0010      750.0  1.863889  18.565640 -56.788054 -10.005566  1.981447
0.0050      750.0  1.863890  18.565617 -56.788054 -10.005566  1.981447
0.0100      750.0  1.864012  18.565426 -56.788054 -10.005566  1.981447
0.0500      750.0  1.864657  18.566158 -56.788054 -10.005566  1.981447
0.1000      750.0  1.864743  18.566752 -56.788054 -10.005566  1.981447
0.5000      750.0  1.863531  18.566137 -56.835055  -9.996599  1.966605
1.0000      750.0  1.863910  18.566893 -56.835055  -9.996599  1.966605
5.0000      750.0  1.869287  18.564978 -56.770738 -10.117811  1.959184
10.0000     750.0  1.874358  18.566459 -56.962452 -10.077304  1.946197
50.0000     750.0  1.901971  18.576579 -57.418854  -9.945578  1.944341
100.0000    750.0  1.927645  18.601408 -57.450394 -10.313971  1.967842
500.0000    750.0  1.864157  18.730666 -57.341550 -10.484169  1.620846
1000.0000   750.0  1.728983  18.904051 -57.709517 -10.913729  1.249845
5000.0000   750.0  1.370344  19.333264 -55.576994 -11.595857  0.717070
10000.0000  750.0  1.200334  19.512661 -67.857143 -11.753738  0.718615
50000.0000  750.0  0.825230  19.569137 -54.989487 -12.066245  0.072047

                  75%        max
alpha
0.0001      14.096177  53.021645
0.0005      14.096177  53.021645
0.0010      14.096177  53.021645
0.0050      14.096177  53.021645
0.0100      14.096177  53.021645
0.0500      14.096177  53.021645
0.1000      14.096177  53.021645
0.5000      14.124337  53.034014
1.0000      14.110091  53.161410
5.0000      14.086282  53.153989
10.0000     14.069584  53.024119
50.0000     14.089672  53.241806
100.0000    14.054578  53.713049
500.0000    13.907854  54.706246
1000.0000   14.053644  56.910328
5000.0000   14.788806  62.554113
10000.0000  14.949091  62.326531
50000.0000  14.763142  55.633890
```

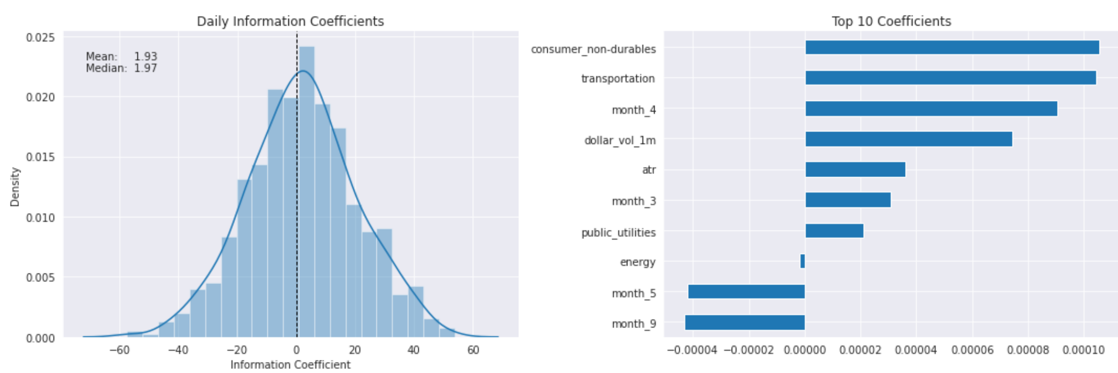
[33]:
```python
fig, axes = plt.subplots(ncols=2, sharex=True, figsize=(15, 5))

scores_by_alpha = ridge_scores.groupby('alpha').ic.agg(['mean', 'median'])
```

```python
best_alpha_mean = scores_by_alpha['mean'].idxmax()
best_alpha_median = scores_by_alpha['median'].idxmax()

ax = sns.lineplot(x='alpha',
                  y='ic',
                  data=ridge_scores,
                  estimator=np.mean,
                  label='Mean',
                  ax=axes[0])

scores_by_alpha['median'].plot(logx=True,
                               ax=axes[0],
                               label='Median')

axes[0].axvline(best_alpha_mean,
                ls='--',
                c='k',
                lw=1,
                label='Max. Mean')
axes[0].axvline(best_alpha_median,
                ls='-.',
                c='k',
                lw=1,
                label='Max. Median')
axes[0].legend()
axes[0].set_xscale('log')
axes[0].set_xlabel('Alpha')
axes[0].set_ylabel('Information Coefficient')
axes[0].set_title('Cross Validation Performance')

ridge_coeffs.plot(logx=True,
                  legend=False,
                  ax=axes[1],
                  title='Ridge Coefficient Path')

axes[1].axvline(best_alpha_mean,
                ls='--',
                c='k',
                lw=1,
                label='Max. Mean')
axes[1].axvline(best_alpha_median,
                ls='-.',
                c='k',
                lw=1,
                label='Max. Median')
axes[1].set_xlabel('Alpha')
axes[1].set_ylabel('Coefficient Value')
```

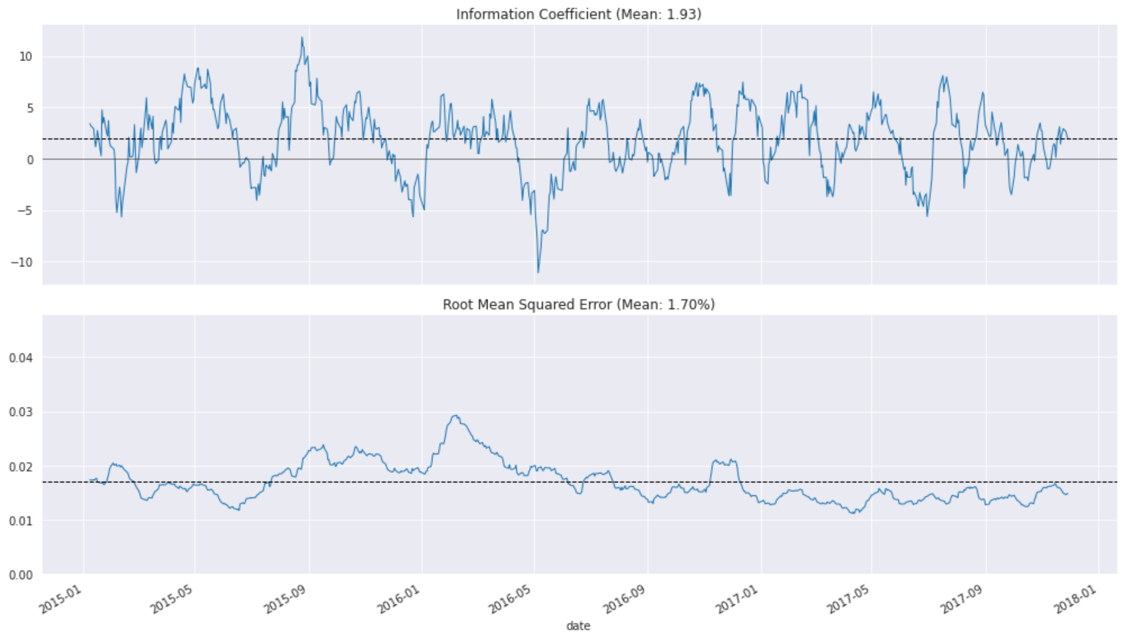
```
fig.suptitle('Ridge Results', fontsize=14)
sns.despine()
fig.tight_layout()
fig.subplots_adjust(top=.9)
```



[34]:
```
best_alpha = ridge_scores.groupby('alpha').ic.mean().idxmax()
fig, axes = plt.subplots(ncols=2, figsize=(15, 5))
plot_ic_distribution(ridge_scores[ridge_scores.alpha == best_alpha],
                     ax=axes[0])
axes[0].set_title('Daily Information Coefficients')
top_coeffs = ridge_coeffs.loc[best_alpha].abs().sort_values().head(10).index
top_coeffs.tolist()
ridge_coeffs.loc[best_alpha, top_coeffs].sort_values().plot.barh(ax=axes[1],
                                                                title='Top 10␣
 ↪Coefficients')
sns.despine()
fig.tight_layout()
```



[35]:
```
plot_rolling_ic(ridge_scores[ridge_scores.alpha==best_alpha])
```



1.7 Lasso CV

1.7.1 Define cross-validation parameters

[36]:
```
lasso_alphas = np.logspace(-10, -3, 8)
```

[37]:
```
train_period_length = 63
test_period_length = 10
YEAR = 252
n_splits = int(3 * YEAR / test_period_length) # three years
lookahead = 1
```

[38]:
```
cv = MultipleTimeSeriesCV(n_splits=n_splits,
                          test_period_length=test_period_length,
                          lookahead=lookahead,
                          train_period_length=train_period_length)
```

1.7.2 Run cross-validation with Lasso regression

[39]:
```
target = f'target_{lookahead}d'

scaler = StandardScaler()
X = X.drop([c for c in X.columns if 'year' in c], axis=1)
```

[40]:
```
%%time
```

```
lasso_coeffs, lasso_scores, lasso_predictions = {}, [], []
for alpha in lasso_alphas:
    print(alpha, end=' ', flush=True)
    model = Lasso(alpha=alpha,
                  fit_intercept=False,  # StandardScaler centers data
                  random_state=42,
                  tol=1e-3,
                  max_iter=1000,
                  warm_start=True,
                  selection='random')

    pipe = Pipeline([
        ('scaler', StandardScaler()),
        ('model', model)])
    coeffs = []
    for i, (train_idx, test_idx) in enumerate(cv.split(X), 1):
        t = time()
        X_train, y_train, = X.iloc[train_idx], y[target].iloc[train_idx]
        X_test, y_test = X.iloc[test_idx], y[target].iloc[test_idx]

        pipe.fit(X=X_train, y=y_train)
        y_pred = pipe.predict(X_test)

        preds = y_test.to_frame('actuals').assign(predicted=y_pred)
        preds_by_day = preds.groupby(level='date')
        scores = pd.concat([preds_by_day.apply(lambda x: spearmanr(x.predicted,
                                                                   x.
↪actuals)[0] * 100)
                            .to_frame('ic'),
                            preds_by_day.apply(lambda x: np.
↪sqrt(mean_squared_error(y_pred=x.predicted,
                                                                                 ␣
↪    y_true=x.actuals)))
                            .to_frame('rmse')],
                           axis=1)

        lasso_scores.append(scores.assign(alpha=alpha))
        lasso_predictions.append(preds.assign(alpha=alpha))

        coeffs.append(pipe.named_steps['model'].coef_)

    lasso_coeffs[alpha] = np.mean(coeffs, axis=0)
```

```
1e-10 1e-09 1e-08 1e-07 1e-06 1e-05 0.0001 0.001 CPU times: user 2min 29s, sys:
4.96 s, total: 2min 34s
Wall time: 44.1 s
```

1.7.3 Persist results

[41]:
```
lasso_scores = pd.concat(lasso_scores)
lasso_scores.to_hdf('data.h5', 'lasso/scores')

lasso_coeffs = pd.DataFrame(lasso_coeffs, index=X.columns).T
lasso_coeffs.to_hdf('data.h5', 'lasso/coeffs')

lasso_predictions = pd.concat(lasso_predictions)
lasso_predictions.to_hdf('data.h5', 'lasso/predictions')
```

1.7.4 Evaluate Lasso Results

[42]:
```
best_alpha = lasso_scores.groupby('alpha').ic.mean().idxmax()
preds = lasso_predictions[lasso_predictions.alpha==best_alpha]

lasso_r, lasso_p = spearmanr(preds.actuals, preds.predicted)
print(f'Information Coefficient (overall): {lasso_r:.3%} (p-value: {lasso_p:.
 ↪4%})')
```

```
Information Coefficient (overall): 3.595% (p-value: 0.0000%)
```

[43]:
```
lasso_scores.groupby('alpha').ic.agg(['mean', 'median'])
```

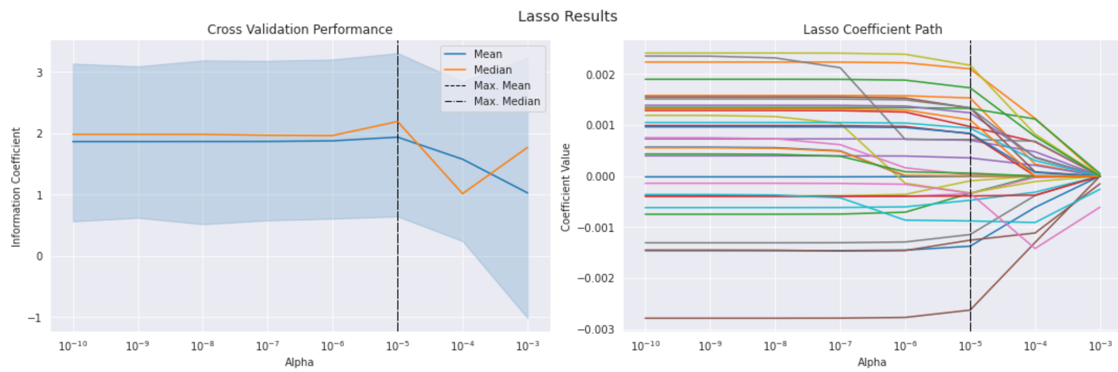
[43]:

alpha	mean	median
1.000000e-10	1.863889	1.981447
1.000000e-09	1.863758	1.981447
1.000000e-08	1.864487	1.981447
1.000000e-07	1.865393	1.966605
1.000000e-06	1.875294	1.962276
1.000000e-05	1.935876	2.191108
1.000000e-04	1.575376	1.012989
1.000000e-03	1.025462	1.768092

1.7.5 Lasso Coefficient Path

[44]:
```
fig, axes = plt.subplots(ncols=2, sharex=True, figsize=(15, 5))

scores_by_alpha = lasso_scores.groupby('alpha').ic.agg(['mean', 'median'])
best_alpha_mean = scores_by_alpha['mean'].idxmax()
best_alpha_median = scores_by_alpha['median'].idxmax()

ax = sns.lineplot(x='alpha', y='ic', data=lasso_scores, estimator=np.mean,␣
 ↪label='Mean', ax=axes[0])

scores_by_alpha['median'].plot(logx=True, ax=axes[0], label='Median')
```

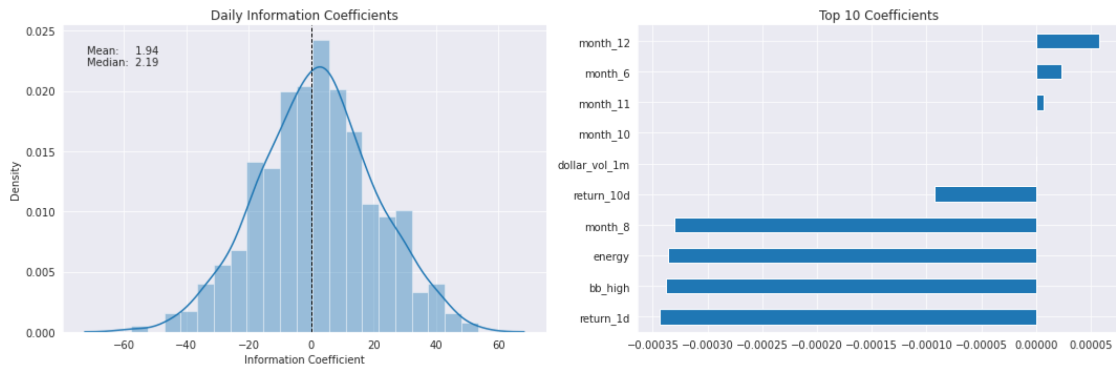
```
axes[0].axvline(best_alpha_mean, ls='--', c='k', lw=1, label='Max. Mean')
axes[0].axvline(best_alpha_median, ls='-.', c='k', lw=1, label='Max. Median')
axes[0].legend()
axes[0].set_xscale('log')
axes[0].set_xlabel('Alpha')
axes[0].set_ylabel('Information Coefficient')
axes[0].set_title('Cross Validation Performance')

lasso_coeffs.plot(logx=True, legend=False, ax=axes[1], title='Lasso Coefficient␣
 ↪Path')
axes[1].axvline(best_alpha_mean, ls='--', c='k', lw=1, label='Max. Mean')
axes[1].axvline(best_alpha_median, ls='-.', c='k', lw=1, label='Max. Median')
axes[1].set_xlabel('Alpha')
axes[1].set_ylabel('Coefficient Value')

fig.suptitle('Lasso Results', fontsize=14)
fig.tight_layout()
fig.subplots_adjust(top=.9)
sns.despine();
```



1.7.6 Lasso IC Distribution and Top 10 Features

[45]:
```
best_alpha = lasso_scores.groupby('alpha').ic.mean().idxmax()

fig, axes = plt.subplots(ncols=2, figsize=(15, 5))
plot_ic_distribution(lasso_scores[lasso_scores.alpha==best_alpha], ax=axes[0])
axes[0].set_title('Daily Information Coefficients')

top_coeffs = lasso_coeffs.loc[best_alpha].abs().sort_values().head(10).index
top_coeffs.tolist()
lasso_coeffs.loc[best_alpha, top_coeffs].sort_values().plot.barh(ax=axes[1],␣
 ↪title='Top 10 Coefficients')
```

```
sns.despine()
fig.tight_layout();
```



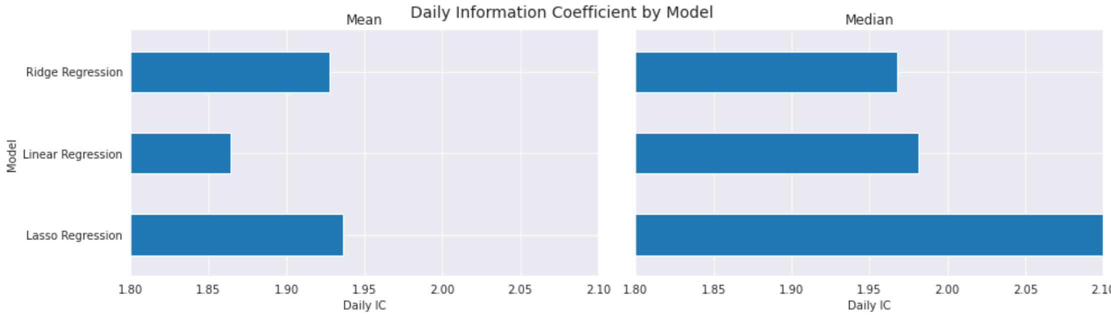
1.8 Compare results

[46]:
```
best_ridge_alpha = ridge_scores.groupby('alpha').ic.mean().idxmax()
best_ridge_preds = ridge_predictions[ridge_predictions.alpha==best_ridge_alpha]
best_ridge_scores = ridge_scores[ridge_scores.alpha==best_ridge_alpha]
```

[47]:
```
best_lasso_alpha = lasso_scores.groupby('alpha').ic.mean().idxmax()
best_lasso_preds = lasso_predictions[lasso_predictions.alpha==best_lasso_alpha]
best_lasso_scores = lasso_scores[lasso_scores.alpha==best_lasso_alpha]
```

[48]:
```
df = pd.concat([lr_scores.assign(Model='Linear Regression'),
               best_ridge_scores.assign(Model='Ridge Regression'),
               best_lasso_scores.assign(Model='Lasso Regression')]).
 ↪drop('alpha', axis=1)
df.columns = ['IC', 'RMSE', 'Model']
```

[49]:
```
scores = df.groupby('Model').IC.agg(['mean', 'median'])
fig, axes = plt.subplots(ncols=2, figsize=(14,4), sharey=True, sharex=True)

scores['mean'].plot.barh(ax=axes[0], xlim=(1.85, 2), title='Mean')
scores['median'].plot.barh(ax=axes[1], xlim=(1.8, 2.1), title='Median')
axes[0].set_xlabel('Daily IC')
axes[1].set_xlabel('Daily IC')

fig.suptitle('Daily Information Coefficient by Model', fontsize=14)
sns.despine()
fig.tight_layout()
fig.subplots_adjust(top=.9)
```


Daily Information Coefficient by Model
Mean
Median
Ridge Regression
Linear Regression
Lasso Regression
Model
1.80
1.85
1.90
1.95
2.00
2.05
2.10
Daily IC