05_random_forest_return_signals

September 29, 2021

1 How to generate long-short trading signals with a Random Forest

1.1 Imports & Settings

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

[2]:
```
%matplotlib inline

from time import time
from io import StringIO
import sys, os
from tqdm import tqdm

from itertools import product
from pathlib import Path

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

import matplotlib.pyplot as plt
import seaborn as sns

import lightgbm as lgb

from sklearn.linear_model import LinearRegression
from scipy.stats import spearmanr
```

[3]:
```
sys.path.insert(1, os.path.join(sys.path[0], '..'))
from utils import MultipleTimeSeriesCV, format_time
```

[4]:
```
sns.set_style('whitegrid')
```

[5]:
```
np.random.seed(42)
```

[6]:
```python
YEAR = 252
idx = pd.IndexSlice
```

[7]:
```python
DATA_DIR = Path('..', 'data')
```

[8]:
```python
results_path = Path('results', 'return_predictions')
if not results_path.exists():
    results_path.mkdir(parents=True)
```

1.2 Get Data

See the notebook japanese_equity_features in this directory for data preparation.

[9]:
```python
data = pd.read_hdf('data.h5', 'stooq/japan/equities')
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
```

We start with 941 tickers.

```
[10]: len(data.index.unique('ticker'))
```

```
[10]: 941
```

1.2.1 Select universe of 250 most-liquid stocks

We rank the stocks by their daily average dollar volume and select those with the 250 lowest average ranks and thus highest average volumes for the 2010-2017 period.

```
[11]: prices = (pd.read_hdf(DATA_DIR / 'assets.h5', 'stooq/jp/tse/stocks/prices')
                .loc[idx[:, '2010': '2017'], :])
```

```
[12]: dollar_vol = prices.close.mul(prices.volume)
      dollar_vol_rank = dollar_vol.groupby(level='date').rank(ascending=False)
      universe = dollar_vol_rank.groupby(level='ticker').mean().nsmallest(250).index
```

1.3 MultipleTimeSeriesCV

See Chapter 7 - Linear Models for details.

```
[13]: cv = MultipleTimeSeriesCV(n_splits=36,
                                test_period_length=21,
                                lookahead=5,
                                train_period_length=2 * 252)
```

For each fold, the train and test periods are separated by a `lookahead` number of periods and thus do not overlap:

```
[14]: for i, (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')
          df = train.reset_index().append(test.reset_index())
          n = len(df)
          assert n== len(df.drop_duplicates())
          msg = f'Training: {train_dates.min().date()}-{train_dates.max().date()} '
          msg += f' ({train.groupby(level="ticker").size().value_counts().index[0]:,.
       ↪0f} days) | '
          msg += f'Test: {test_dates.min().date()}-{test_dates.max().date()} '
          msg += f'({test.groupby(level="ticker").size().value_counts().index[0]:,.
       ↪0f} days)'
          print(msg)
          if i == 3:
              break
```

```
Training: 2017-10-24-2019-11-25  (508 days) | Test: 2019-12-02-2019-12-30 (21
days)
Training: 2017-09-22-2019-10-24  (508 days) | Test: 2019-10-31-2019-11-29 (21
days)
Training: 2017-08-23-2019-09-20  (508 days) | Test: 2019-09-30-2019-10-30 (21
days)
Training: 2017-07-24-2019-08-21  (508 days) | Test: 2019-08-28-2019-09-27 (21
days)
```

1.4 Model Selection: Time Period and Horizon

For the model selection step, we restrict training and validation sets to the 2010-2017 period.

```
[15]: cv_data = data.loc[idx[universe, :'2017'], :]
      tickers = cv_data.index.unique('ticker')
```

Persist the data to save some time when running another experiment:

```
[16]: cv_data.to_hdf('data.h5', 'stooq/japan/equities/cv_data')
```

```
[17]: with pd.HDFStore('data.h5') as store:
          print(store.info())
```

```
<class 'pandas.io.pytables.HDFStore'>
File path: data.h5
/stooq/japan/equities                    frame        (shape->[2304509,23])
/stooq/japan/equities/cv_data            frame        (shape->[418119,23])
/us/equities/monthly                     frame        (shape->[77788,27])
/us/equities/prices                      frame        (shape->[9532628,16])
```

We're picking prediction horizons of 1, 5, 10 and 21 days:

```
[18]: lookaheads = [1, 5, 10, 21]
```

1.5 Baseline: Linear Regression

Since it's quick to run and quite informative, we generate linear regression baseline predictions. See Chapter 7 - Linear Models for details.

```
[19]: lr = LinearRegression()
```

```
[20]: labels = sorted(cv_data.filter(like='fwd').columns)
      features = cv_data.columns.difference(labels).tolist()
```

1.5.1 CV Parameters

We set five different training lengths from 3 months to 5 years, and two test periods as follows:

[21]:
```
train_lengths = [5 * YEAR, 3 * YEAR, YEAR, 126, 63]
test_lengths = [5, 21]
```

Since linear regression has no hyperparameters, our CV parameters are the cartesian product of prediction horizon and train/test period lengths:

[22]:
```
test_params = list(product(lookaheads, train_lengths, test_lengths))
```

Now we iterate over these parameters and train/validate the linear regression model while capturing the information coefficient of the model predictions, measure both on a daily basis and for each complete fold:

[23]:
```
lr_metrics = []
for lookahead, train_length, test_length in tqdm(test_params):
    label = f'fwd_ret_{lookahead:02}'
    df = cv_data.loc[:, features + [label]].dropna()
    X, y = df.drop(label, axis=1), df[label]

    n_splits = int(2 * YEAR / test_length)
    cv = MultipleTimeSeriesCV(n_splits=n_splits,
                              test_period_length=test_length,
                              lookahead=lookahead,
                              train_period_length=train_length)

    ic, 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)
        preds.append(y_test.to_frame('y_true').assign(y_pred=y_pred))
        ic.append(spearmanr(y_test, y_pred)[0])
    preds = pd.concat(preds)
    lr_metrics.append([
        lookahead, train_length, test_length,
        np.mean(ic),
        spearmanr(preds.y_true, preds.y_pred)[0]
    ])

columns = ['lookahead', 'train_length', 'test_length', 'ic_by_day', 'ic']
lr_metrics = pd.DataFrame(lr_metrics, columns=columns)
```

```
100%|      | 40/40 [02:58<00:00,  4.47s/it]
```

[24]:
```
lr_metrics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40 entries, 0 to 39
```

```
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   lookahead     40 non-null     int64
 1   train_length  40 non-null     int64
 2   test_length   40 non-null     int64
 3   ic_by_day     40 non-null     float64
 4   ic            40 non-null     float64
dtypes: float64(2), int64(3)
memory usage: 1.7 KB
```

1.5.2 Information Coefficient distribution by Lookahead

Convert the data to long seaborn-friendly format:

[25]:
```
lr_metrics_long = pd.concat([(lr_metrics.drop('ic', axis=1)
                              .rename(columns={'ic_by_day': 'ic'})
                              .assign(Measured='By Day')),
                             lr_metrics.drop('ic_by_day', axis=1)
                             .assign(Measured='Overall')])
lr_metrics_long.columns=['Lookahead', 'Train Length', 'Test Length', 'IC',␣
 ↪'Measure']
lr_metrics_long.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 80 entries, 0 to 39
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Lookahead     80 non-null     int64
 1   Train Length  80 non-null     int64
 2   Test Length   80 non-null     int64
 3   IC            80 non-null     float64
 4   Measure       80 non-null     object
dtypes: float64(1), int64(3), object(1)
memory usage: 3.8+ KB
```

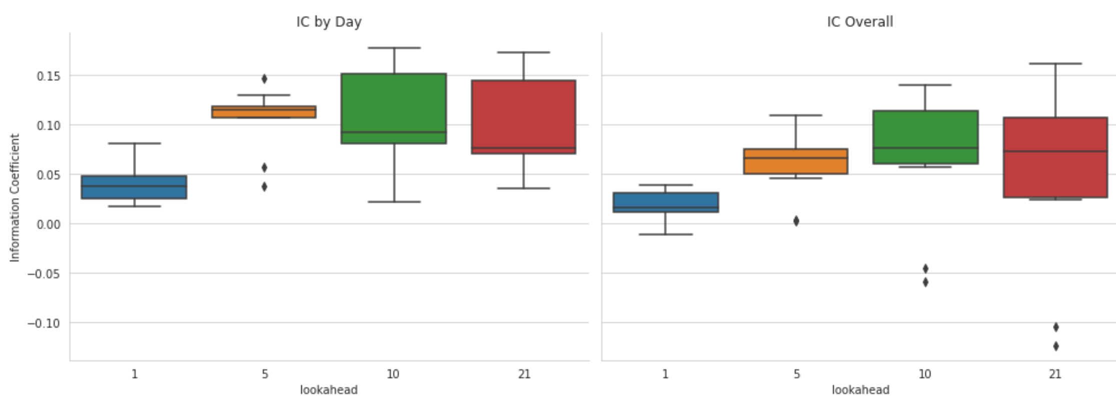
Plot both IC measures for the various CV parameters:

[26]:
```
sns.catplot(x='Train Length',
            y='IC',
            hue='Test Length',
            col='Lookahead',
            row='Measure',
            data=lr_metrics_long,
            kind='bar')
```

[26]:
```
<seaborn.axisgrid.FacetGrid at 0x7f99c77e9310>
```

Compare the distributions of each IC metric for the different prediction horizons:

[27]:
```
fig, axes =plt.subplots(ncols=2, figsize=(14,5), sharey=True)
sns.boxplot(x='lookahead', y='ic_by_day',data=lr_metrics, ax=axes[0])
axes[0].set_title('IC by Day')
sns.boxplot(x='lookahead', y='ic',data=lr_metrics, ax=axes[1])
axes[1].set_title('IC Overall')
axes[0].set_ylabel('Information Coefficient')
axes[1].set_ylabel('')
sns.despine()
fig.tight_layout()
```



1.5.3 Best Train/Test Period Lengths

Show the best train/test period settings for the four prediction horizons:

```
[29]: (lr_metrics.groupby('lookahead', group_keys=False)
       .apply(lambda x: x.nlargest(3, 'ic')))
```

	lookahead	train_length	test_length	ic_by_day	ic
7	1	126	21	0.017073	0.038653
9	1	63	21	0.033069	0.037626
5	1	252	21	0.042142	0.031726
19	5	63	21	0.147644	0.109061
17	5	126	21	0.116522	0.102297
16	5	126	5	0.113916	0.075632
26	10	126	5	0.099448	0.140622
27	10	126	21	0.157678	0.137302
28	10	63	5	0.080282	0.123428
39	21	63	21	0.173752	0.161578
38	21	63	5	0.072219	0.113831
37	21	126	21	0.162853	0.113148

```
[30]: lr_metrics.to_csv(results_path / 'lin_reg_performance.csv', index=False)
```

1.6 LightGBM Random Forest Model Tuning

Helper function to obtain the LightGBM feature importance metrics:

```
[31]: def get_fi(model):
          fi = model.feature_importance(importance_type='gain')
          return (pd.Series(fi / fi.sum(),
                            index=model.feature_name()))
```

LightGBM base parameter settings that are independent of hyperparameter tuning:

```
[32]: base_params = dict(boosting_type='rf',
                         objective='regression',
                         bagging_freq=1,
                         verbose=-1)
```

1.6.1 Hyperparameter Options

We run this experiment with different parameters for the bagging and feature fractions that determine the degree of randomization as well as the minimum number of samples for a split to control overfitting:

```
[33]: bagging_fraction_opts = [.5, .75, .95]
      feature_fraction_opts = [.75, .95]
      min_data_in_leaf_opts = [250, 500, 1000]
```

This gives us 3x2x3=18 parameter combinations:

[34]:
```
cv_params = list(product(bagging_fraction_opts,
                         feature_fraction_opts,
                         min_data_in_leaf_opts))
n_cv_params = len(cv_params)
n_cv_params
```

[34]: 18

Random Sample To limit the running time, we can randomly sample a subset of the parameter combinations (here: 50%):

[35]:
```
sample_proportion = .5
sample_size = int(sample_proportion * n_cv_params)

cv_param_sample = np.random.choice(list(range(n_cv_params)),
                                    size=int(sample_size),
                                    replace=False)
cv_params_ = [cv_params[i] for i in cv_param_sample]
print('# CV parameters:', len(cv_params_))
```

```
# CV parameters: 9
```

We tune the number of trees by evaluating a fully grown forest for various smaller sizes:

[36]:
```
num_iterations = [25] + list(range(50, 501, 25))
num_boost_round = num_iterations[-1]
```

1.6.2 Train/Test Period Lenghts

As above for linear regression, we define a range of train/test period length:

Define parameters

[37]:
```
train_lengths = [5 * YEAR, 3 * YEAR, YEAR, 126, 63]
test_lengths = [5, 21]
```

[38]:
```
test_params = list(product(train_lengths, test_lengths))
n_test_params = len(test_params)
```

Random sample Just as for the model parameters, we can randomly sample from the 5 x 2 = 8 training configurations (here: 50%):

[39]:
```
sample_proportion = 1.0
sample_size = int(sample_proportion * n_test_params)

test_param_sample = np.random.choice(list(range(n_test_params)),
                                      size=int(sample_size),
                                      replace=False)
```

```python
test_params_ = [test_params[i] for i in test_param_sample]
print('Train configs:', len(test_params_))
print('CV Iterations:', len(cv_params_) * len(test_params_))
```

```
Train configs: 10
CV Iterations: 90
```

1.6.3 Categorical Variables

To leverage LightGBM's ability to handle categorical variables, we need to define them; we'll also `factorize` them so they are both integer-encoded and start at zero (optional, but otherwise throws a warning) as expected by LightGBM:

[40]:
```python
categoricals = ['year', 'weekday', 'month']
for feature in categoricals:
    data[feature] = pd.factorize(data[feature], sort=True)[0]
```

1.6.4 Run Cross-Validation

Set up some helper variabels and storage locations to faciliate the CV process and result storage:

[41]:
```python
labels = sorted(cv_data.filter(like='fwd').columns)
features = cv_data.columns.difference(labels).tolist()
```

[42]:
```python
label_dict = dict(zip(lookaheads, labels))
```

[43]:
```python
cv_store = Path(results_path / 'parameter_tuning.h5')
```

[44]:
```python
ic_cols = ['bagging_fraction',
           'feature_fraction',
           'min_data_in_leaf',
           't'] + [str(n) for n in num_iterations]
```

Now we take the following steps: - we iterate over the prediction horizons and train/test period length, - set up the `MultipleTimeSeriesCV` accordingly - create the binary LightGBM dataset with the appropriate target, and - iterate over the model hyperparamters to train and validate the model while capturing the relevant performance metrics:

[43]:
```python
for lookahead in lookaheads:
    for train_length, test_length in test_params_:
        n_splits = int(2 * YEAR / test_length)
        print(f'Lookahead: {lookahead:2.0f} | Train: {train_length:3.0f} | '
              f'Test: {test_length:2.0f} | Params: {len(cv_params_):3.0f}')

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

```
        label = label_dict[lookahead]
        outcome_data = data.loc[:, features + [label]].dropna()

        lgb_data = lgb.Dataset(data=outcome_data.drop(label, axis=1),
                               label=outcome_data[label],
                               categorical_feature=categoricals,
                               free_raw_data=False)
        predictions, daily_ic, ic, feature_importance = [], [], [], []
        key = f'{lookahead}/{train_length}/{test_length}'
        T = 0
        for p, (bagging_fraction, feature_fraction, min_data_in_leaf) in␣
↪enumerate(cv_params_):
            params = base_params.copy()
            params.update(dict(bagging_fraction=bagging_fraction,
                               feature_fraction=feature_fraction,
                               min_data_in_leaf=min_data_in_leaf))

            start = time()
            cv_preds, nrounds = [], []
            for i, (train_idx, test_idx) in enumerate(cv.split(X=outcome_data)):
                lgb_train = lgb_data.subset(train_idx.tolist()).construct()
                lgb_test = lgb_data.subset(test_idx.tolist()).construct()

                model = lgb.train(params=params,
                                  train_set=lgb_train,
                                  num_boost_round=num_boost_round,
                                  verbose_eval=False)
                if i == 0:
                    fi = get_fi(model).to_frame()
                else:
                    fi[i] = get_fi(model)

                test_set = outcome_data.iloc[test_idx, :]
                X_test = test_set.loc[:, model.feature_name()]
                y_test = test_set.loc[:, label]
                y_pred = {str(n): model.predict(X_test, num_iteration=n)
                          for n in num_iterations}
                cv_preds.append(y_test.to_frame(
                    'y_test').assign(**y_pred).assign(i=i))
                nrounds.append(model.best_iteration)
            feature_importance.append(fi.T.describe().T.
↪assign(bagging_fraction=bagging_fraction,
                                                                          ␣
↪feature_fraction=feature_fraction,
                                                                          ␣
↪min_data_in_leaf=min_data_in_leaf))
```

```
            cv_preds = pd.concat(cv_preds).
↪assign(bagging_fraction=bagging_fraction,
                                                                ␣
↪feature_fraction=feature_fraction,
                                                                ␣
↪min_data_in_leaf=min_data_in_leaf)

            predictions.append(cv_preds)
            by_day = cv_preds.groupby(level='date')
            ic_by_day = pd.concat([by_day.apply(lambda x: spearmanr(x.y_test,
                                                                                  ␣
↪x[str(n)])[0]).to_frame(n)
                                   for n in num_iterations], axis=1)

            daily_ic.append(ic_by_day.assign(bagging_fraction=bagging_fraction,
                                             feature_fraction=feature_fraction,
                                             min_data_in_leaf=min_data_in_leaf))

            cv_ic = [spearmanr(cv_preds.y_test, cv_preds[str(n)])[0]
                  for n in num_iterations]

            T += time() - start
            ic.append([bagging_fraction, feature_fraction,
                       min_data_in_leaf, lookahead] + cv_ic)

            msg = f'{p:3.0f} | {format_time(T)} | '
            msg += f'{bagging_fraction:3.0%} | {feature_fraction:3.0%} |␣
↪{min_data_in_leaf:5,.0f} | '
            msg += f'{max(cv_ic):6.2%} | {ic_by_day.mean().max(): 6.2%} |␣
↪{ic_by_day.median().max(): 6.2%}'
            print(msg)

        m = pd.DataFrame(ic, columns=ic_cols)
        m.to_hdf(cv_store, 'ic/' + key)
        pd.concat(daily_ic).to_hdf(cv_store, 'daily_ic/' + key)
        pd.concat(feature_importance).to_hdf(cv_store, 'fi/' + key)
        pd.concat(predictions).to_hdf(cv_store, 'predictions/' + key)
```

```
Lookahead:  1 | Train:  63 | Test: 21 | Params:  18
  0 | 00:01:08 | 50% | 75% |   250 |  2.03% |  1.09% |  0.91%
  1 | 00:02:19 | 50% | 75% |   500 |  2.14% |  1.25% |  1.15%
  2 | 00:03:35 | 75% | 75% | 1,000 |  2.07% |  1.35% |  1.20%
  3 | 00:04:32 | 50% | 95% | 1,000 |  2.29% |  1.04% |  1.07%
  4 | 00:05:36 | 50% | 95% |   250 |  2.63% |  1.03% |  0.82%
  5 | 00:06:43 | 95% | 75% |   500 |  1.84% |  0.88% |  0.83%
  6 | 00:07:49 | 95% | 95% |   500 |  2.58% |  0.86% |  0.62%
  7 | 00:08:56 | 95% | 95% |   250 |  2.04% |  0.77% |  0.36%
```

```
  8 | 00:10:00 | 75% | 95% | 1,000 |  2.51% |  1.06% |  0.68%
  9 | 00:10:57 | 50% | 75% | 1,000 |  1.99% |  1.11% |  1.25%
 10 | 00:12:02 | 75% | 95% |   250 |  2.06% |  0.93% |  0.38%
 11 | 00:13:07 | 95% | 95% | 1,000 |  2.33% |  0.92% |  1.27%
 12 | 00:14:09 | 50% | 95% |   500 |  2.57% |  1.19% |  1.15%
 13 | 00:15:16 | 95% | 75% |   250 |  1.42% |  0.91% |  0.69%
 14 | 00:16:23 | 75% | 75% |   500 |  1.49% |  1.04% |  1.14%
 15 | 00:17:28 | 75% | 95% |   500 |  2.01% |  0.85% |  0.49%
 16 | 00:18:34 | 95% | 75% | 1,000 |  1.85% |  0.96% |  1.08%
 17 | 00:19:40 | 75% | 75% |   250 |  1.45% |  1.06% |  0.97%
Lookahead:  1 | Train: 252 | Test: 21 | Params:  18
  0 | 00:01:45 | 50% | 75% |   250 |  2.09% |  1.21% |  1.40%
  1 | 00:03:31 | 50% | 75% |   500 |  1.93% |  1.11% |  1.49%
  2 | 00:05:30 | 75% | 75% | 1,000 |  2.22% |  0.92% |  1.42%
  3 | 00:07:27 | 50% | 95% | 1,000 |  2.17% |  1.25% |  1.64%
  4 | 00:09:21 | 50% | 95% |   250 |  2.67% |  0.90% |  1.18%
  5 | 00:11:40 | 95% | 75% |   500 |  2.62% |  1.10% |  1.83%
  6 | 00:13:49 | 95% | 95% |   500 |  3.14% |  0.87% |  1.24%
  7 | 00:15:57 | 95% | 95% |   250 |  3.40% |  0.95% |  1.26%
  8 | 00:17:57 | 75% | 95% | 1,000 |  2.39% |  0.85% |  1.52%
  9 | 00:19:46 | 50% | 75% | 1,000 |  1.56% |  1.17% |  2.22%
 10 | 00:21:44 | 75% | 95% |   250 |  3.15% |  1.08% |  1.21%
 11 | 00:23:53 | 95% | 95% | 1,000 |  2.85% |  0.76% |  0.88%
 12 | 00:25:40 | 50% | 95% |   500 |  2.35% |  0.85% |  0.92%
 13 | 00:27:44 | 95% | 75% |   250 |  2.70% |  1.07% |  1.56%
 14 | 00:29:41 | 75% | 75% |   500 |  2.43% |  1.12% |  1.44%
 15 | 00:31:39 | 75% | 95% |   500 |  2.86% |  1.01% |  1.16%
 16 | 00:33:46 | 95% | 75% | 1,000 |  2.47% |  0.92% |  1.45%
 17 | 00:35:41 | 75% | 75% |   250 |  2.61% |  1.19% |  1.53%
Lookahead:  1 | Train: 756 | Test: 21 | Params:  18
  0 | 00:03:39 | 50% | 75% |   250 |  2.53% |  0.90% |  1.00%
  1 | 00:07:23 | 50% | 75% |   500 |  2.48% |  1.10% |  1.34%
  2 | 00:11:55 | 75% | 75% | 1,000 |  2.27% |  1.09% |  1.33%
  3 | 00:15:49 | 50% | 95% | 1,000 |  1.99% |  1.10% |  1.11%
  4 | 00:19:31 | 50% | 95% |   250 |  1.15% |  0.89% |  1.33%
  5 | 00:24:28 | 95% | 75% |   500 |  1.31% |  0.92% |  1.34%
  6 | 00:29:44 | 95% | 95% |   500 |  0.21% |  0.93% |  1.27%
  7 | 00:34:55 | 95% | 95% |   250 | -0.05% |  1.07% |  1.32%
  8 | 00:39:44 | 75% | 95% | 1,000 |  0.81% |  0.92% |  1.20%
  9 | 00:43:37 | 50% | 75% | 1,000 |  2.78% |  1.18% |  1.23%
 10 | 00:48:38 | 75% | 95% |   250 |  0.13% |  0.96% |  1.08%
 11 | 00:54:04 | 95% | 95% | 1,000 |  0.61% |  0.74% |  0.76%
 12 | 00:57:50 | 50% | 95% |   500 |  1.57% |  0.94% |  1.10%
 13 | 01:02:44 | 95% | 75% |   250 |  1.14% |  0.85% |  1.18%
 14 | 01:07:10 | 75% | 75% |   500 |  1.94% |  1.11% |  1.12%
 15 | 01:11:48 | 75% | 95% |   500 |  0.39% |  0.90% |  1.19%
 16 | 01:16:54 | 95% | 75% | 1,000 |  1.71% |  0.99% |  1.60%
 17 | 01:21:12 | 75% | 75% |   250 |  1.81% |  0.99% |  1.09%
```

```
Lookahead:  5 | Train:  63 | Test: 21 | Params:  18
   0 | 00:01:04 | 50% | 75% |   250 | -3.99% | -2.15% | -1.68%
   1 | 00:02:06 | 50% | 75% |   500 | -4.28% | -2.13% | -1.97%
   2 | 00:03:11 | 75% | 75% | 1,000 | -4.27% | -2.32% | -1.79%
   3 | 00:04:06 | 50% | 95% | 1,000 | -4.62% | -2.52% | -2.08%
   4 | 00:05:09 | 50% | 95% |   250 | -4.42% | -2.27% | -1.89%
   5 | 00:06:17 | 95% | 75% |   500 | -5.14% | -2.11% | -1.63%
   6 | 00:07:25 | 95% | 95% |   500 | -5.14% | -1.84% | -1.13%
   7 | 00:08:33 | 95% | 95% |   250 | -5.15% | -1.86% | -1.42%
   8 | 00:09:38 | 75% | 95% | 1,000 | -4.80% | -2.11% | -1.43%
   9 | 00:10:36 | 50% | 75% | 1,000 | -4.40% | -2.38% | -2.21%
  10 | 00:11:41 | 75% | 95% |   250 | -4.71% | -2.31% | -1.66%
  11 | 00:12:48 | 95% | 95% | 1,000 | -5.61% | -2.27% | -1.52%
  12 | 00:13:51 | 50% | 95% |   500 | -4.72% | -2.34% | -2.20%
  13 | 00:15:01 | 95% | 75% |   250 | -5.22% | -1.97% | -1.39%
  14 | 00:16:08 | 75% | 75% |   500 | -4.10% | -2.34% | -1.98%
  15 | 00:17:13 | 75% | 95% |   500 | -4.78% | -2.26% | -1.86%
  16 | 00:18:21 | 95% | 75% | 1,000 | -5.57% | -2.41% | -2.02%
  17 | 00:19:28 | 75% | 75% |   250 | -4.25% | -2.04% | -1.43%
Lookahead:  5 | Train: 252 | Test: 21 | Params:  18
   0 | 00:01:45 | 50% | 75% |   250 | -0.52% |  1.63% |  2.39%
   1 | 00:03:30 | 50% | 75% |   500 | -0.50% |  1.41% |  2.15%
   2 | 00:05:28 | 75% | 75% | 1,000 | -0.68% |  0.73% |  1.54%
   3 | 00:07:15 | 50% | 95% | 1,000 | -1.10% |  0.75% |  1.39%
   4 | 00:09:01 | 50% | 95% |   250 | -0.96% |  1.17% |  1.79%
   5 | 00:11:08 | 95% | 75% |   500 | -0.82% |  1.70% |  2.06%
   6 | 00:13:18 | 95% | 95% |   500 | -0.97% |  1.27% |  1.60%
   7 | 00:15:30 | 95% | 95% |   250 | -0.91% |  1.24% |  1.54%
   8 | 00:17:30 | 75% | 95% | 1,000 | -0.76% |  1.37% |  1.96%
   9 | 00:19:15 | 50% | 75% | 1,000 | -0.84% |  0.93% |  1.95%
  10 | 00:21:14 | 75% | 95% |   250 | -1.06% |  1.58% |  2.10%
  11 | 00:23:28 | 95% | 95% | 1,000 | -0.85% |  1.31% |  1.60%
  12 | 00:25:15 | 50% | 95% |   500 | -0.92% |  1.46% |  2.16%
  13 | 00:27:22 | 95% | 75% |   250 | -0.75% |  1.63% |  1.93%
  14 | 00:29:18 | 75% | 75% |   500 | -0.72% |  1.16% |  2.15%
  15 | 00:31:16 | 75% | 95% |   500 | -1.04% |  1.48% |  2.09%
  16 | 00:33:23 | 95% | 75% | 1,000 | -0.79% |  1.50% |  1.96%
  17 | 00:35:19 | 75% | 75% |   250 | -0.70% |  1.37% |  2.03%
Lookahead:  5 | Train: 756 | Test: 21 | Params:  18
   0 | 00:03:56 | 50% | 75% |   250 |  1.97% |  1.78% |  1.93%
   1 | 00:07:48 | 50% | 75% |   500 |  2.05% |  1.75% |  1.94%
   2 | 00:12:24 | 75% | 75% | 1,000 |  2.00% |  1.78% |  1.91%
   3 | 00:16:21 | 50% | 95% | 1,000 |  2.41% |  1.55% |  1.04%
   4 | 00:20:16 | 50% | 95% |   250 |  2.08% |  1.36% |  1.54%
   5 | 00:25:31 | 95% | 75% |   500 |  2.24% |  1.77% |  2.00%
   6 | 00:31:17 | 95% | 95% |   500 |  2.40% |  1.76% |  1.69%
   7 | 00:37:03 | 95% | 95% |   250 |  2.37% |  1.72% |  1.50%
   8 | 00:41:60 | 75% | 95% | 1,000 |  2.24% |  1.40% |  1.58%
```

```
  9 | 00:45:48 | 50% | 75% | 1,000 |  2.48% |  1.77% |  1.90%
 10 | 00:50:50 | 75% | 95% |   250 |  2.05% |  1.35% |  1.48%
 11 | 00:56:34 | 95% | 95% | 1,000 |  2.41% |  1.77% |  1.61%
 12 | 01:00:31 | 50% | 95% |   500 |  2.15% |  1.53% |  1.24%
 13 | 01:05:44 | 95% | 75% |   250 |  2.16% |  1.76% |  2.04%
 14 | 01:10:17 | 75% | 75% |   500 |  1.72% |  1.78% |  2.03%
 15 | 01:15:11 | 75% | 95% |   500 |  2.10% |  1.27% |  1.66%
 16 | 01:20:28 | 95% | 75% | 1,000 |  2.52% |  1.73% |  2.02%
 17 | 01:24:59 | 75% | 75% |   250 |  1.71% |  1.77% |  2.13%
Lookahead: 10 | Train:  63 | Test: 21 | Params:  18
  0 | 00:01:04 | 50% | 75% |   250 | -1.81% | -0.10% | -1.29%
  1 | 00:02:07 | 50% | 75% |   500 | -1.82% |  0.01% | -1.12%
  2 | 00:03:13 | 75% | 75% | 1,000 | -1.86% |  0.43% | -0.78%
  3 | 00:04:11 | 50% | 95% | 1,000 | -0.94% |  0.34% | -0.61%
  4 | 00:05:15 | 50% | 95% |   250 | -2.43% | -0.35% | -0.92%
  5 | 00:06:24 | 95% | 75% |   500 | -1.53% |  0.07% | -0.86%
  6 | 00:07:33 | 95% | 95% |   500 | -2.47% |  0.02% | -0.52%
  7 | 00:08:44 | 95% | 95% |   250 | -2.88% | -0.15% | -0.29%
  8 | 00:09:49 | 75% | 95% | 1,000 | -2.40% |  0.27% | -0.14%
  9 | 00:10:47 | 50% | 75% | 1,000 | -0.81% |  0.51% | -0.80%
 10 | 00:11:54 | 75% | 95% |   250 | -2.38% | -0.18% | -0.47%
 11 | 00:13:02 | 95% | 95% | 1,000 | -2.23% |  0.15% | -0.28%
 12 | 00:14:05 | 50% | 95% |   500 | -2.16% | -0.17% | -0.56%
 13 | 00:15:14 | 95% | 75% |   250 | -1.79% | -0.10% | -0.91%
 14 | 00:16:21 | 75% | 75% |   500 | -1.66% | -0.05% | -1.23%
 15 | 00:17:27 | 75% | 95% |   500 | -2.53% | -0.10% | -0.64%
 16 | 00:18:36 | 95% | 75% | 1,000 | -1.62% |  0.20% | -1.01%
 17 | 00:19:42 | 75% | 75% |   250 | -1.77% | -0.09% | -0.96%
Lookahead: 10 | Train: 252 | Test: 21 | Params:  18
  0 | 00:01:50 | 50% | 75% |   250 | -4.74% |  0.77% |  1.15%
  1 | 00:03:37 | 50% | 75% |   500 | -5.90% |  0.64% |  1.03%
  2 | 00:05:39 | 75% | 75% | 1,000 | -4.93% |  0.70% |  0.61%
  3 | 00:07:27 | 50% | 95% | 1,000 | -5.86% |  0.44% |  0.75%
  4 | 00:09:16 | 50% | 95% |   250 | -3.95% |  0.58% |  1.35%
  5 | 00:11:26 | 95% | 75% |   500 | -2.57% |  0.95% |  1.07%
  6 | 00:13:42 | 95% | 95% |   500 | -2.20% |  0.92% |  1.04%
  7 | 00:15:57 | 95% | 95% |   250 | -2.52% |  0.74% |  0.92%
  8 | 00:17:60 | 75% | 95% | 1,000 | -4.57% |  0.84% |  1.19%
  9 | 00:19:47 | 50% | 75% | 1,000 | -5.78% |  0.54% |  0.52%
 10 | 00:21:51 | 75% | 95% |   250 | -2.87% |  0.85% |  1.42%
 11 | 00:24:06 | 95% | 95% | 1,000 | -4.38% |  0.91% |  0.98%
 12 | 00:25:54 | 50% | 95% |   500 | -6.13% |  0.49% |  1.30%
 13 | 00:28:04 | 95% | 75% |   250 | -2.73% |  0.82% |  0.89%
 14 | 00:30:04 | 75% | 75% |   500 | -3.82% |  0.77% |  1.03%
 15 | 00:32:07 | 75% | 95% |   500 | -2.60% |  0.95% |  1.44%
 16 | 00:34:18 | 95% | 75% | 1,000 | -3.76% |  0.88% |  1.23%
 17 | 00:36:17 | 75% | 75% |   250 | -3.94% |  0.72% |  1.00%
Lookahead: 10 | Train: 756 | Test: 21 | Params:  18
```

```
  0 | 00:03:47 | 50% | 75% |   250 |  3.11% |  1.63% |  1.15%
  1 | 00:07:34 | 50% | 75% |   500 |  2.64% |  1.53% |  0.95%
  2 | 00:12:15 | 75% | 75% | 1,000 |  2.34% |  1.40% |  0.86%
  3 | 00:16:16 | 50% | 95% | 1,000 |  2.17% |  0.92% |  0.54%
  4 | 00:20:17 | 50% | 95% |   250 |  2.84% |  0.92% |  0.63%
  5 | 00:25:44 | 95% | 75% |   500 |  2.58% |  1.44% |  0.14%
  6 | 00:31:44 | 95% | 95% |   500 |  2.66% |  0.72% | -0.10%
  7 | 00:37:41 | 95% | 95% |   250 |  2.66% |  0.80% |  0.02%
  8 | 00:42:46 | 75% | 95% | 1,000 |  2.30% |  0.88% |  0.30%
  9 | 00:46:33 | 50% | 75% | 1,000 |  2.60% |  1.50% |  1.30%
 10 | 00:51:36 | 75% | 95% |   250 |  2.57% |  1.14% |  0.63%
 11 | 00:57:30 | 95% | 95% | 1,000 |  2.64% |  0.69% | -0.03%
 12 | 01:01:30 | 50% | 95% |   500 |  2.34% |  0.87% |  0.73%
 13 | 01:06:56 | 95% | 75% |   250 |  2.57% |  1.44% |  0.17%
 14 | 01:11:34 | 75% | 75% |   500 |  2.69% |  1.46% |  0.94%
 15 | 01:16:41 | 75% | 95% |   500 |  2.56% |  0.98% |  0.45%
 16 | 01:22:26 | 95% | 75% | 1,000 |  2.62% |  1.32% |  0.17%
 17 | 01:27:27 | 75% | 75% |   250 |  2.69% |  1.48% |  0.89%
Lookahead: 21 | Train:  63 | Test: 21 | Params:  18
  0 | 00:01:08 | 50% | 75% |   250 |  0.36% |  2.59% |  1.44%
  1 | 00:02:13 | 50% | 75% |   500 |  0.28% |  2.71% |  1.36%
  2 | 00:03:23 | 75% | 75% | 1,000 |  5.00% |  2.86% |  1.77%
  3 | 00:04:25 | 50% | 95% | 1,000 |  1.51% |  2.59% |  1.61%
  4 | 00:05:32 | 50% | 95% |   250 |  0.49% |  2.68% |  1.44%
  5 | 00:06:45 | 95% | 75% |   500 | 10.07% |  3.04% |  1.83%
  6 | 00:07:57 | 95% | 95% |   500 |  9.87% |  2.87% |  1.82%
  7 | 00:09:11 | 95% | 95% |   250 |  9.79% |  3.18% |  2.49%
  8 | 00:10:22 | 75% | 95% | 1,000 |  5.32% |  2.76% |  2.13%
  9 | 00:11:29 | 50% | 75% | 1,000 |  1.02% |  2.86% |  1.33%
 10 | 00:12:46 | 75% | 95% |   250 |  5.13% |  2.99% |  2.18%
 11 | 00:14:04 | 95% | 95% | 1,000 | 10.18% |  2.64% |  1.84%
 12 | 00:15:14 | 50% | 95% |   500 |  0.47% |  2.80% |  1.54%
 13 | 00:16:33 | 95% | 75% |   250 |  9.76% |  3.26% |  2.74%
 14 | 00:17:43 | 75% | 75% |   500 |  4.92% |  2.90% |  1.54%
 15 | 00:18:52 | 75% | 95% |   500 |  5.12% |  2.87% |  1.78%
 16 | 00:20:04 | 95% | 75% | 1,000 | 10.40% |  2.85% |  1.70%
 17 | 00:21:14 | 75% | 75% |   250 |  4.68% |  2.94% |  1.84%
Lookahead: 21 | Train: 252 | Test: 21 | Params:  18
  0 | 00:01:53 | 50% | 75% |   250 | -5.71% |  1.69% |  0.90%
  1 | 00:03:43 | 50% | 75% |   500 | -5.67% |  1.67% |  0.87%
  2 | 00:05:48 | 75% | 75% | 1,000 | -6.04% |  1.45% |  0.48%
  3 | 00:07:40 | 50% | 95% | 1,000 | -6.44% |  1.56% |  1.38%
  4 | 00:09:34 | 50% | 95% |   250 | -6.56% |  1.54% |  1.31%
  5 | 00:11:51 | 95% | 75% |   500 | -4.51% |  1.45% |  0.32%
  6 | 00:14:50 | 95% | 95% |   500 | -5.06% |  1.46% |  0.98%
  7 | 00:17:42 | 95% | 95% |   250 | -5.10% |  1.51% |  0.90%
  8 | 00:20:22 | 75% | 95% | 1,000 | -6.66% |  1.18% |  0.67%
  9 | 00:22:20 | 50% | 75% | 1,000 | -5.50% |  2.03% |  0.89%
```

```
 10 | 00:24:35 | 75% | 95% |   250 | -6.65% |  1.37% |  0.81%
 11 | 00:26:56 | 95% | 95% | 1,000 | -5.06% |  1.32% |  0.61%
 12 | 00:28:48 | 50% | 95% |   500 | -6.54% |  1.41% |  1.29%
 13 | 00:31:05 | 95% | 75% |   250 | -4.59% |  1.61% |  0.40%
 14 | 00:33:10 | 75% | 75% |   500 | -6.18% |  1.50% |  0.80%
 15 | 00:35:18 | 75% | 95% |   500 | -6.59% |  1.32% |  0.81%
 16 | 00:37:35 | 95% | 75% | 1,000 | -4.68% |  1.23% | -0.01%
 17 | 00:39:42 | 75% | 75% |   250 | -6.16% |  1.54% |  0.71%
Lookahead: 21 | Train: 756 | Test: 21 | Params:  18
  0 | 00:04:01 | 50% | 75% |   250 | -6.84% |  0.77% |  0.48%
  1 | 00:07:59 | 50% | 75% |   500 | -6.76% |  1.03% |  0.56%
  2 | 00:12:55 | 75% | 75% | 1,000 | -8.84% |  1.10% |  0.81%
  3 | 00:17:52 | 50% | 95% | 1,000 | -9.43% |  1.49% |  1.03%
  4 | 00:22:09 | 50% | 95% |   250 | -9.59% |  1.11% |  0.75%
  5 | 00:27:47 | 95% | 75% |   500 | -10.97% |  1.11% |  0.63%
  6 | 00:33:58 | 95% | 95% |   500 | -12.27% |  0.23% | -0.34%
  7 | 00:40:10 | 95% | 95% |   250 | -12.25% |  0.23% | -0.34%
  8 | 00:45:29 | 75% | 95% | 1,000 | -10.07% |  0.67% |  0.42%
  9 | 00:49:28 | 50% | 75% | 1,000 | -6.73% |  1.22% |  0.63%
 10 | 00:54:46 | 75% | 95% |   250 | -10.31% |  0.37% |  0.16%
 11 | 01:00:59 | 95% | 95% | 1,000 | -12.20% |  0.31% | -0.30%
 12 | 01:05:10 | 50% | 95% |   500 | -9.58% |  1.44% |  1.09%
 13 | 01:10:51 | 95% | 75% |   250 | -10.96% |  1.10% |  0.46%
 14 | 01:15:49 | 75% | 75% |   500 | -8.86% |  0.98% |  0.59%
 15 | 01:21:08 | 75% | 95% |   500 | -10.09% |  0.59% |  0.41%
 16 | 01:26:46 | 95% | 75% | 1,000 | -10.96% |  1.18% |  0.79%
 17 | 01:31:39 | 75% | 75% |   250 | -8.90% |  0.96% |  0.68%
```

1.7 Analyse Cross-Validation Results

1.7.1 Collect Data

We'll now combine the CV results that we stored separately for each fold (to avoid loosing results in case something goes wrong along the way):

[45]:
```python
id_vars = ['train_length',
           'test_length',
           'bagging_fraction',
           'feature_fraction',
           'min_data_in_leaf',
           't', 'date']
```

We'll look at the financial performance in the notebook `alphalens_signal_quality`.

[46]:
```python
daily_ic, ic = [], []
for t in lookaheads:
    print(t)
    with pd.HDFStore(cv_store) as store:
        keys = [k[1:] for k in store.keys() if k.startswith(f'/fi/{t}')]
```

```
        for key in keys:
            train_length, test_length = key.split('/')[2:]
            print(train_length, test_length)
            k = f'{t}/{train_length}/{test_length}'
            cols = {'t': t,
                    'train_length': int(train_length),
                    'test_length': int(test_length)}

            ic.append(pd.melt(store['ic/' + k]
                              .assign(**cols),
                              id_vars=id_vars[:-1],
                              value_name='ic',
                              var_name='rounds')
                      .apply(pd.to_numeric))

            df = store['daily_ic/' + k].assign(**cols).reset_index()
            daily_ic.append(pd.melt(df,
                                    id_vars=id_vars,
                                    value_name='daily_ic',
                                    var_name='rounds')
                            .set_index('date')
                            .apply(pd.to_numeric)
                            .reset_index())
ic = pd.concat(ic, ignore_index=True)
daily_ic = pd.concat(daily_ic, ignore_index=True)
```

```
1
756 21
63 21
252 21
756 21
63 21
252 21
5
756 21
63 21
252 21
10
756 21
63 21
252 21
21
756 21
63 21
252 21
```

1.7.2 Predictive Performance: CV Information Coefficient by Day

We first look at the daily IC, the metric we ultimately care about for a daily trading strategy. The best results for all prediction horizons are typically achieved with three years of training; the shorter horizons work better with 21 day testing period length. More regularization often improves the result but the impact of the bagging and feature fraction parameters are a little less clear cut and likely depend on other parameters.

[47]:
```
group_cols = ['t','train_length', 'test_length',
              'bagging_fraction', 'feature_fraction', 'min_data_in_leaf']
daily_ic_avg = daily_ic.groupby(group_cols + ['rounds']).daily_ic.mean().
 ↪to_frame('ic').reset_index()
daily_ic_avg.groupby('t', group_keys=False).apply(lambda x: x.nlargest(3, 'ic'))
```

[47]:
```
       t  train_length  test_length  bagging_fraction  feature_fraction  \
161    1            63           21              0.75              0.75
160    1            63           21              0.75              0.75
162    1            63           21              0.75              0.75
1942   5           756           21              0.75              0.75
1962   5           756           21              0.75              0.75
1805   5           756           21              0.50              0.75
2886  10           756           21              0.50              0.75
2887  10           756           21              0.50              0.75
2906  10           756           21              0.50              0.75
3481  21            63           21              0.95              0.75
3484  21            63           21              0.95              0.75
3482  21            63           21              0.95              0.75

      min_data_in_leaf  rounds        ic
161               1000      50  0.013466
160               1000      25  0.012966
162               1000      75  0.012946
1942               500      75  0.017819
1962              1000      75  0.017804
1805               250     150  0.017762
2886               250     175  0.016297
2887               250     200  0.015740
2906               500     175  0.015297
3481               250      50  0.032590
3484               250     125  0.032105
3482               250      75  0.031893
```

[48]:
```
daily_ic_avg.info(null_counts=True)
```

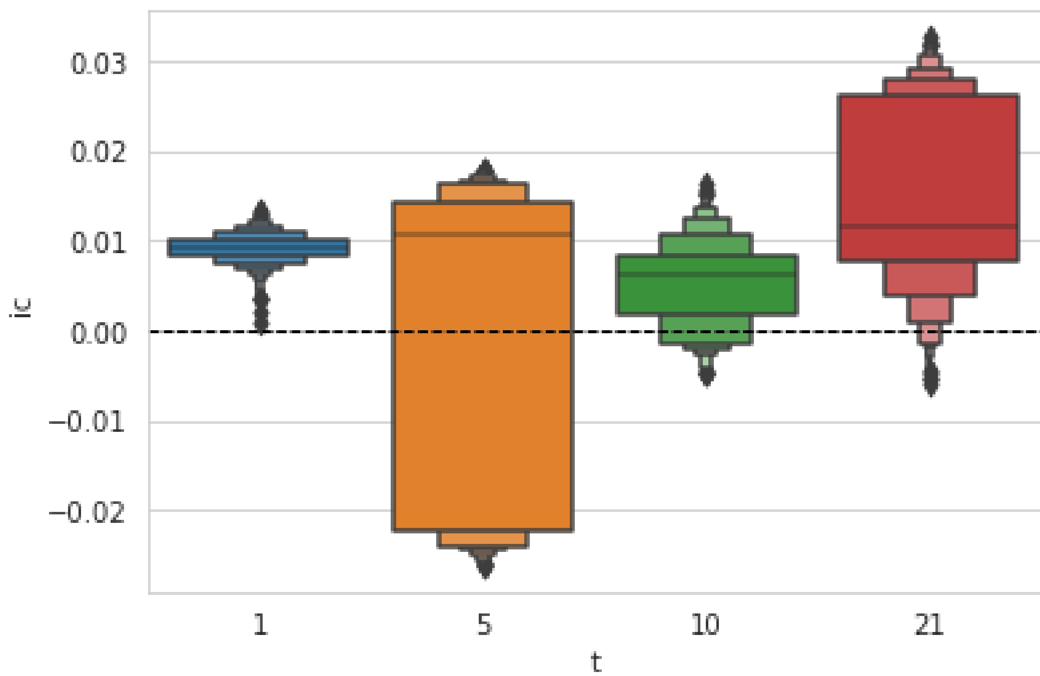
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4320 entries, 0 to 4319
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
```

```
---  ------            --------------  -----
 0   t                 4320 non-null   int64
 1   train_length      4320 non-null   int64
 2   test_length       4320 non-null   int64
 3   bagging_fraction  4320 non-null   float64
 4   feature_fraction  4320 non-null   float64
 5   min_data_in_leaf  4320 non-null   int64
 6   rounds            4320 non-null   int64
 7   ic                4320 non-null   float64
dtypes: float64(3), int64(5)
memory usage: 270.1 KB
```

For a 1-day forecast horizon, over 75% of the predictions yield a positive daily IC; the same is true for 21 days which, unsurprisingly, also shows a wider range.

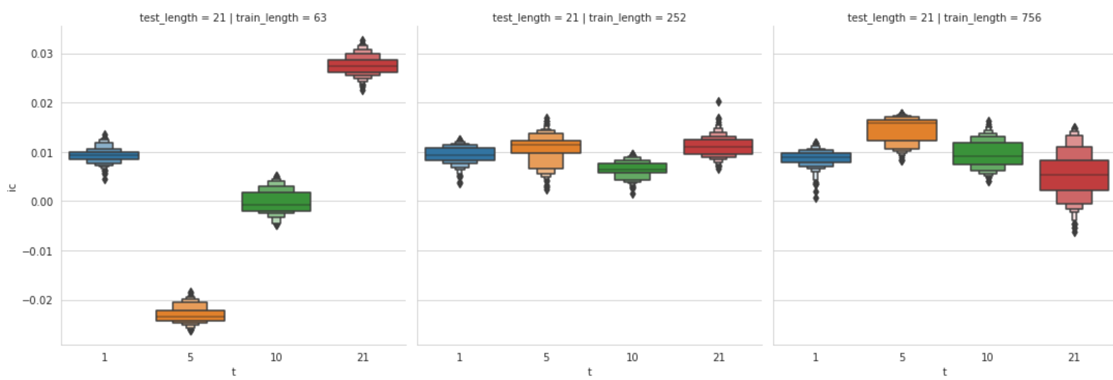
[49]:

```
ax = sns.boxenplot(x='t', y='ic', data=daily_ic_avg)
ax.axhline(0, ls='--', lw=1, c='k');
```



[50]:

```
g = sns.catplot(x='t',
                y='ic',
                col='train_length',
                row='test_length',
                data=daily_ic_avg[(daily_ic_avg.test_length == 21)],
                kind='boxen')
g.savefig(results_path / 'daily_ic_test_21', dpi=300);
```



1.7.3 HyperParameter Impact: Linear Regression

To get a better idea of how the various CV parameters impact the forecast quality, we can run a linear regression with the daily IC as outcome and the one-hot encoded hyperparameters as inputs:

[51]:
```
lin_reg = {}
for t in [1, 5]:
    df_ = daily_ic_avg[(daily_ic_avg.t==t)&(daily_ic_avg.rounds<=250)].dropna()
    y, X = df_.ic, df_.drop(['ic', 't'], axis=1)
    X = sm.add_constant(pd.get_dummies(X, columns=X.columns, drop_first=True))
    model = sm.OLS(endog=y, exog=X)
    lin_reg[t] = model.fit()
    s = lin_reg[t].summary()
    coefs = pd.read_csv(StringIO(s.tables[1].as_csv())).rename(
        columns=lambda x: x.strip())
    coefs.columns = ['variable', 'coef', 'std_err',
                     't', 'p_value', 'ci_low', 'ci_high']
    coefs.to_csv(results_path / f'lr_result_{t:02}.csv', index=False)
```

[52]:
```
def visualize_lr_result(model, ax):
    ci = model.conf_int()
    errors = ci[1].sub(ci[0]).div(2)

    coefs = (model.params.to_frame('coef').assign(error=errors)
             .reset_index().rename(columns={'index': 'variable'}))
    coefs = coefs[~coefs['variable'].str.startswith(
        'date') & (coefs.variable != 'const')]
    coefs.variable = coefs.variable.str.split('_').str[-1]

    coefs.plot(x='variable', y='coef', kind='bar', ax=ax,
               color='none', capsize=3, yerr='error', legend=False, rot=0)
    ax.set_ylabel('IC')
    ax.set_xlabel('')
```

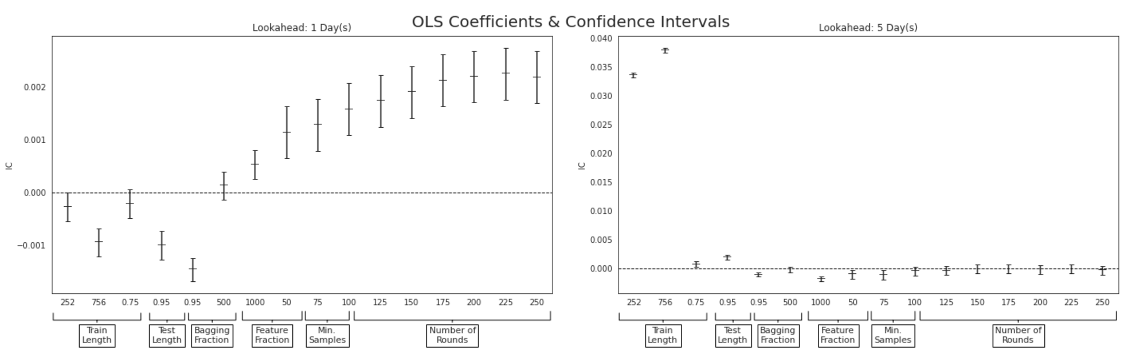
```python
    ax.scatter(x=pd.np.arange(len(coefs)), marker='_', s=120, y=coefs['coef'], color='black')
    ax.axhline(y=0, linestyle='--', color='black', linewidth=1)
    ax.xaxis.set_ticks_position('none')

    ax.annotate('Train\nLength', xy=(.09, -0.1), xytext=(.09, -0.2),
                xycoords='axes fraction',
                textcoords='axes fraction',
                fontsize=11, ha='center', va='bottom',
                bbox=dict(boxstyle='square', fc='white', ec='black'),
                arrowprops=dict(arrowstyle='-[, widthB=5, lengthB=0.8', lw=1.0, color='black'))

    ax.annotate('Test\nLength', xy=(.23, -0.1), xytext=(.23, -0.2),
                xycoords='axes fraction',
                textcoords='axes fraction',
                fontsize=11, ha='center', va='bottom',
                bbox=dict(boxstyle='square', fc='white', ec='black'),
                arrowprops=dict(arrowstyle='-[, widthB=2, lengthB=0.8', lw=1.0, color='black'))

    ax.annotate('Bagging\nFraction', xy=(.32, -0.1), xytext=(.32, -0.2),
                xycoords='axes fraction',
                textcoords='axes fraction',
                fontsize=11, ha='center', va='bottom',
                bbox=dict(boxstyle='square', fc='white', ec='black'),
                arrowprops=dict(arrowstyle='-[, widthB=2.7, lengthB=0.8', lw=1.0, color='black'))


    ax.annotate('Feature\nFraction', xy=(.44, -0.1), xytext=(.44, -0.2),
                xycoords='axes fraction',
                textcoords='axes fraction',
                fontsize=11, ha='center', va='bottom',
                bbox=dict(boxstyle='square', fc='white', ec='black'),
                arrowprops=dict(arrowstyle='-[, widthB=3.4, lengthB=1.0', lw=1.0, color='black'))


    ax.annotate('Min.\nSamples', xy=(.55, -0.1), xytext=(.55, -0.2),
                xycoords='axes fraction',
                textcoords='axes fraction',
                fontsize=11, ha='center', va='bottom',
                bbox=dict(boxstyle='square', fc='white', ec='black'),
                arrowprops=dict(arrowstyle='-[, widthB=2.5, lengthB=1.0', lw=1.0, color='black'))
```

```
    ax.annotate('Number of\nRounds', xy=(.8, -0.1), xytext=(.8, -0.2),
                xycoords='axes fraction',
                textcoords='axes fraction',
                fontsize=11, ha='center', va='bottom',
                bbox=dict(boxstyle='square', fc='white', ec='black'),
                arrowprops=dict(arrowstyle='-[, widthB=11.2, lengthB=1.0', lw=1.
↪0, color='black'))
```

The below plot shows the regression coefficient values and their confidence intervals. The intercept (not shown) has a small positive value and is statistically signifant; it captures the impact of the dropped categories (the smallest value for each parameter).

For 1-day forecasts, some but not all results are insightful: 21-day testing is better, and so is `min_samples_leaf` of 500 or 1,000. 100-200 trees seem to work best, but both shorter and longer training periods are better than intermediate values.

[53]:
```
with sns.axes_style('white'):
    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20, 6))
    axes = axes.flatten()
    for i, t in enumerate([1, 5]):
        visualize_lr_result(lin_reg[t], axes[i])
        axes[i].set_title(f'Lookahead: {t} Day(s)')
    fig.suptitle('OLS Coefficients & Confidence Intervals', fontsize=20)
    fig.tight_layout()
    fig.subplots_adjust(top=.92)
```



1.7.4 Information Coefficient: Overall

We'll also take a look at the overall IC value, which is often reported but does not necessarily match the goal of a daily trading strategy that uses the model return predictions as well as the daily IC.

[54]:
```
ic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5400 entries, 0 to 5399
```

```
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   train_length       5400 non-null   int64  
 1   test_length        5400 non-null   int64  
 2   bagging_fraction   5400 non-null   float64
 3   feature_fraction   5400 non-null   float64
 4   min_data_in_leaf   5400 non-null   int64  
 5   t                  5400 non-null   int64  
 6   rounds             5400 non-null   int64  
 7   ic                 5400 non-null   float64
dtypes: float64(3), int64(5)
memory usage: 337.6 KB
```

Best Parameters Directionally, and for shorter periods, similar hyperparameter settings work best (while the IC values are higher):

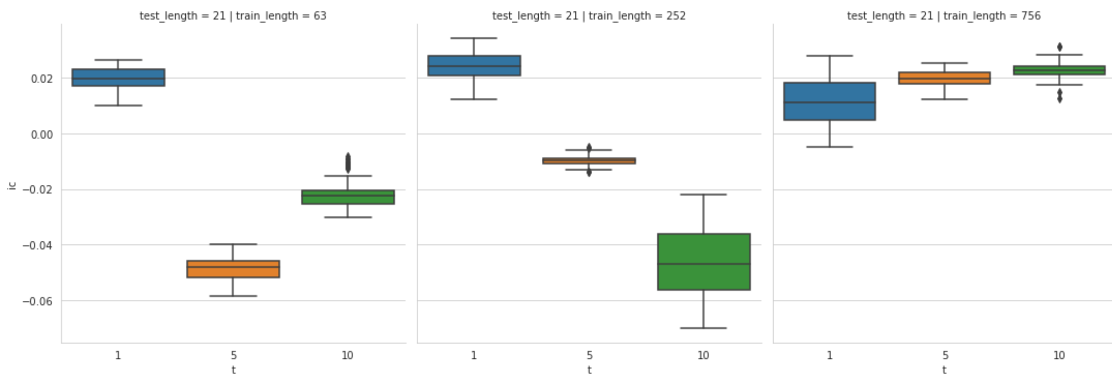
[55]:
```
ic.groupby('t').apply(lambda x: x.nlargest(3, 'ic'))
```

[55]:
```
         train_length  test_length  bagging_fraction  feature_fraction  \
t                                                                        
1  1051           252           21              0.95              0.95   
   2131           252           21              0.95              0.95   
   979            252           21              0.95              0.95   
5  2194           756           21              0.95              0.75   
   2421           756           21              0.50              0.75   
   2295           756           21              0.50              0.75   
10 3258           756           21              0.50              0.75   
   3240           756           21              0.50              0.75   
   3370           756           21              0.50              0.95   
21 4696            63           21              0.95              0.75   
   4691            63           21              0.95              0.95   
   4727            63           21              0.95              0.95   

         min_data_in_leaf   t  rounds        ic  
t                                                
1  1051               250   1     475  0.034049  
   2131               250   1     475  0.034049  
   979                250   1     375  0.034045  
5  2194              1000   5      50  0.025161  
   2421              1000   5     375  0.024792  
   2295              1000   5     200  0.024420  
10 3258               250  10      50  0.031098  
   3240               250  10      25  0.031094  
   3370               250  10     200  0.028374  
21 4696              1000  21      25  0.104042  
   4691              1000  21      25  0.101774  
```

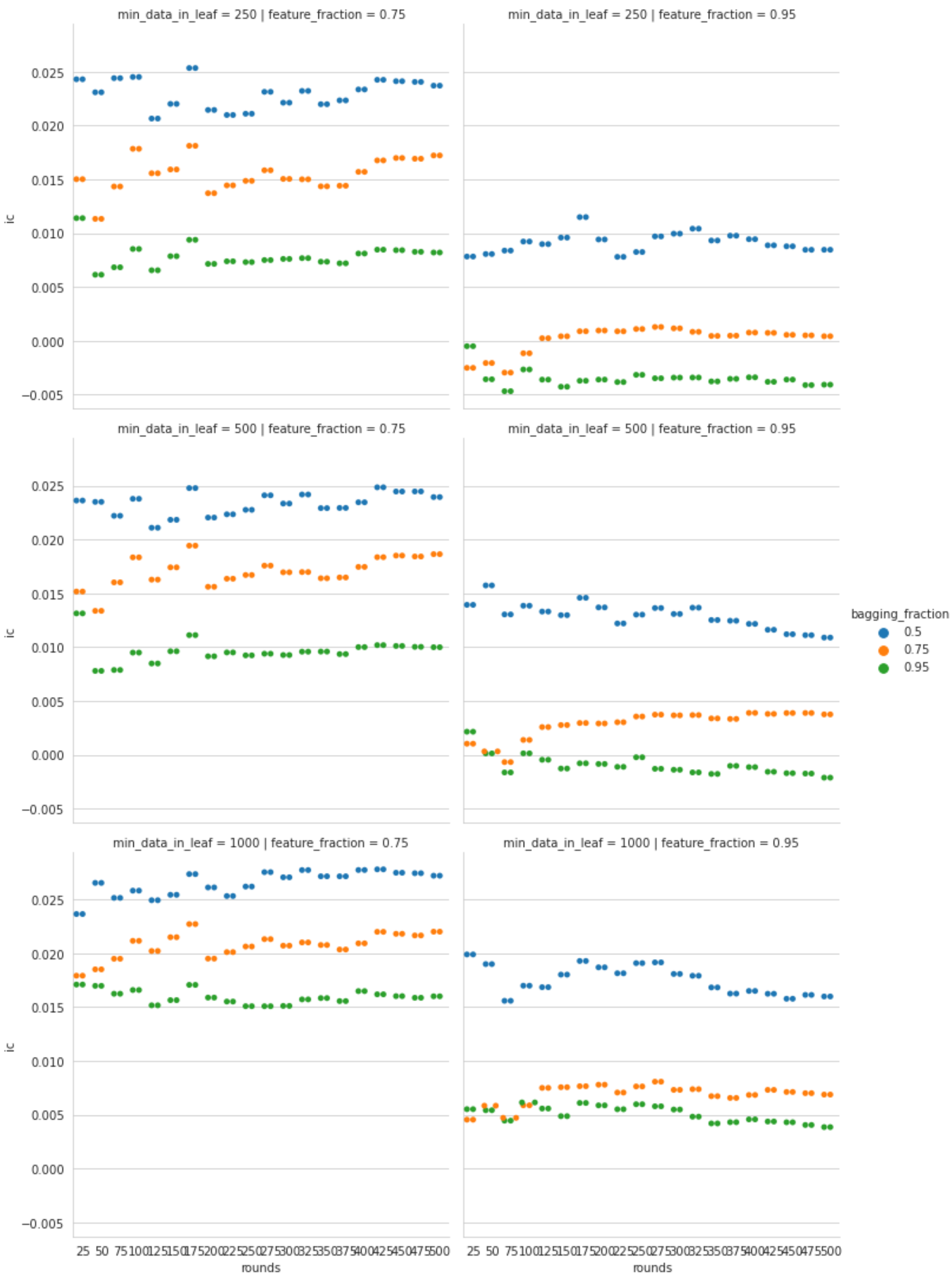
```
4727                 1000   21       75   0.101336
```

Visualiztion

[56]:
```
g = sns.catplot(x='t',
                y='ic',
                col='train_length',
                row='test_length',
                data=ic[(ic.test_length == 21) & (ic.t < 21)],
                kind='box')
```



[57]:
```
t = 1
train_length = 756
test_length = 21
g = sns.catplot(x='rounds',
    y='ic',
    col='feature_fraction',
    hue='bagging_fraction',
    row='min_data_in_leaf',
    data=ic[(ic.t == t) &
            (ic.train_length == train_length) &
            (ic.test_length == test_length)],
    kind='swarm');
```


min_data_in_leaf = 250 | feature_fraction = 0.75
min_data_in_leaf = 250 | feature_fraction = 0.95
min_data_in_leaf = 500 | feature_fraction = 0.75
min_data_in_leaf = 500 | feature_fraction = 0.95
min_data_in_leaf = 1000 | feature_fraction = 0.75
min_data_in_leaf = 1000 | feature_fraction = 0.95
ic
rounds
bagging_fraction
0.5
0.75
0.95

1.7.5 Random Forest vs Linear Regression

Let's compare the best-performing (in-sample) random forest models to our linear regression baseline:

[59]:
```
lr_metrics = pd.read_csv(results_path / 'lin_reg_performance.csv')
lr_metrics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40 entries, 0 to 39
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   lookahead     40 non-null     int64  
 1   train_length  40 non-null     int64  
 2   test_length   40 non-null     int64  
 3   ic_by_day     40 non-null     float64
 4   ic            40 non-null     float64
dtypes: float64(2), int64(3)
memory usage: 1.7 KB
```

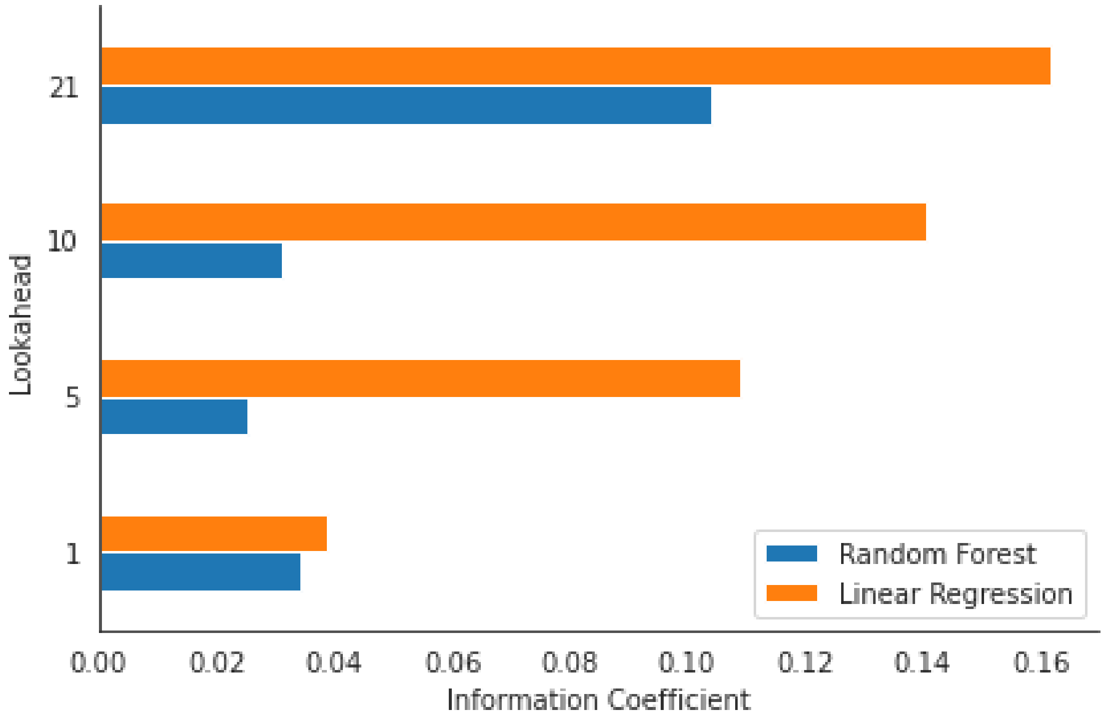
[60]:
```
daily_ic_avg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4320 entries, 0 to 4319
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   t                 4320 non-null   int64  
 1   train_length      4320 non-null   int64  
 2   test_length       4320 non-null   int64  
 3   bagging_fraction  4320 non-null   float64
 4   feature_fraction  4320 non-null   float64
 5   min_data_in_leaf  4320 non-null   int64  
 6   rounds            4320 non-null   int64  
 7   ic                4320 non-null   float64
dtypes: float64(3), int64(5)
memory usage: 270.1 KB
```

The results are mixed: for the shortest and longest horizons, the random forest outperforms (slightly for 1 day), while linear regression is competitive for the intermediate horizons:

[61]:
```
with sns.axes_style("white"):
    ax = (ic.groupby('t').ic.max().to_frame('Random Forest')
     .join(lr_metrics.groupby('lookahead').ic.max().to_frame('Linear␣
↪Regression')).plot.barh())
    ax.set_ylabel('Lookahead')
    ax.set_xlabel('Information Coefficient')
    sns.despine()
```

```
    plt.tight_layout();
```



1.8 Generate predictions

To build and evaluate a trading strategy, we create predictions for the 2018-19 period using the 10 best models that we then ensemble:

[62]:
```
param_cols = ['train_length', 'test_length', 'bagging_fraction',
              'feature_fraction', 'min_data_in_leaf', 'rounds']
```

[63]:
```
def get_params(data, t=5, best=0):
    df = data[data.t == t].sort_values('ic', ascending=False).iloc[best]
    df = df.loc[param_cols]
    rounds = int(df.rounds)
    params = pd.to_numeric(df.drop('rounds'))
    return params, rounds
```

[64]:
```
base_params = dict(boosting_type='rf',
                   objective='regression',
                   bagging_freq=1,
                   verbose=-1)

store = Path(results_path / 'predictions.h5')
```

[81]:

```
for lookahead in [1, 5, 10, 21]:
    if lookahead > 1:
        continue
    print(f'\nLookahead: {lookahead:02}')
    data = (pd.read_hdf('data.h5', 'stooq/japan/equities'))
    labels = sorted(data.filter(like='fwd').columns)
    features = data.columns.difference(labels).tolist()
    label = f'fwd_ret_{lookahead:02}'
    data = data.loc[:, features + [label]].dropna()

    categoricals = ['year', 'weekday', 'month']
    for feature in categoricals:
        data[feature] = pd.factorize(data[feature], sort=True)[0]

    lgb_data = lgb.Dataset(data=data[features],
                           label=data[label],
                           categorical_feature=categoricals,
                           free_raw_data=False)

    for position in range(10):
        params, num_boost_round = get_params(daily_ic_avg,
                                             t=lookahead,
                                             best=position)
        params = params.to_dict()
        params['min_data_in_leaf'] = int(params['min_data_in_leaf'])
        train_length = int(params.pop('train_length'))
        test_length = int(params.pop('test_length'))
        params.update(base_params)

        print(f'\tPosition: {position:02}')

        n_splits = int(2 * YEAR / test_length)
        cv = MultipleTimeSeriesCV(n_splits=n_splits,
                                  test_period_length=test_length,
                                  lookahead=lookahead,
                                  train_period_length=train_length)

        predictions = []
        start = time()
        for i, (train_idx, test_idx) in enumerate(cv.split(X=data), 1):
            train_set = lgb_data.subset(used_indices=train_idx.tolist(),
                                        params=params).construct()

            model = lgb.train(params=params,
                              train_set=train_set,
                              num_boost_round=num_boost_round,
                              verbose_eval=False)
```

```
            test_set = data.iloc[test_idx, :]
            y_test = test_set.loc[:, label].to_frame('y_test')
            y_pred = model.predict(test_set.loc[:, model.feature_name()])
            predictions.append(y_test.assign(prediction=y_pred))

        if position == 0:
            test_predictions = (pd.concat(predictions)
                                .rename(columns={'prediction': position}))
        else:
            test_predictions[position] = pd.concat(predictions).prediction


    by_day = test_predictions.groupby(level='date')
    for position in range(10):
        if position == 0:
            ic_by_day = by_day.apply(lambda x: spearmanr(x.y_test,␣
↪x[position])[0]).to_frame()
        else:
            ic_by_day[position] = by_day.apply(lambda x: spearmanr(x.y_test,␣
↪x[position])[0])

    test_predictions.to_hdf(store, f'test/{lookahead:02}')
```

```
Lookahead: 01
        Position: 00
        Position: 01
        Position: 02
        Position: 03
        Position: 04
        Position: 05
        Position: 06
        Position: 07
        Position: 08
        Position: 09
```

[ ]:

```

```