

# 11\_intraday\_model

September 29, 2021

## 1 Intraday Strategy, Part 2: Model Training & Signal Evaluation

In this notebook, we load the high-quality NASDAQ100 minute-bar trade-and-quote data generously provided by [Algoseek](#) (available [here](#)) and use the features engineered in the last notebook to train gradient boosting model that predicts the returns for the NASDAQ100 stocks over the next 1-minute bar.

Note that we will assume throughout that we can always buy (sell) at the first (last) trade price for a given bar at no cost and without market impact. This does certainly not reflect market reality, and is rather due to the challenges of simulating a trading strategy at this much higher intraday frequency in a realistic manner using open-source tools.

Note also that this section has slightly changed from the version published in the book to permit replication using the Algoseek data sample.

### 1.1 Imports & Settings

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

```
[2]: %matplotlib inline

import sys, os
from pathlib import Path
from time import time
from tqdm import tqdm

import numpy as np
import pandas as pd

from scipy.stats import spearmanr
import lightgbm as lgb

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

Ensuring we can import `utils.py` in the repo's root directory:

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

[64]: sns.set_style('whitegrid')
      idx = pd.IndexSlice
      deciles = np.arange(.1, 1, .1)

[5]: # where we stored the features engineered in the previous notebook
      data_store = 'data/algoseek.h5'

[6]: # where we'll store the model results
      result_store = 'data/intra_day.h5'

[7]: # here we save the trained models
      model_path = Path('models/intraday')
      if not model_path.exists():
          model_path.mkdir(parents=True)
```

## 1.2 Load Model Data

```
[8]: data = pd.read_hdf(data_store, 'model_data2')

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

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 30875649 entries, ('AAL', Timestamp('2015-01-02 09:30:00')) to
('YH00', Timestamp('2017-06-16 15:59:00'))
Data columns (total 22 columns):
#   Column                Non-Null Count  Dtype
---  -
0   minute                30875649 non-null  int64
1   ret1min               30612848 non-null  float64
2   ret2min               30302846 non-null  float64
3   ret3min               30220887 non-null  float64
4   ret4min               30141503 non-null  float64
5   ret5min               30063236 non-null  float64
6   ret6min               29983969 non-null  float64
7   ret7min               29903822 non-null  float64
8   ret8min               29824607 non-null  float64
9   ret9min               29745431 non-null  float64
10  ret10min              29666821 non-null  float64
11  fwd1min               30875649 non-null  float64
12  rup                   30083777 non-null  float64
13  rdown                 30083777 non-null  float64
14  BOP                   30612848 non-null  float64
15  CCI                   28517773 non-null  float64
16  MFI                   30873719 non-null  float64
```

```

17 STOCHRSI      30871639 non-null float64
18 slowd         30873302 non-null float64
19 slowk         30873302 non-null float64
20 NATR          30873719 non-null float64
21 trades_bid_ask 30083777 non-null float64
dtypes: float64(21), int64(1)
memory usage: 5.2+ GB

```

```
[10]: data.sample(frac=.1).describe(percentiles=np.arange(.1, 1, .1))
```

```

[10]:
count      minute      ret1min      ret2min      ret3min      ret4min  \
mean      1.944517e+02 -2.933200e-06 -1.869810e-06 -1.598179e-06 -1.981696e-06
std       1.127876e+02  8.522094e-04  1.143481e-03  1.364755e-03  1.554875e-03
min       0.000000e+00 -1.244796e-01 -8.829405e-02 -1.060236e-01 -1.327945e-01
10%       3.800000e+01 -7.494558e-04 -1.048584e-03 -1.268377e-03 -1.454229e-03
20%       7.700000e+01 -4.105090e-04 -5.929791e-04 -7.232152e-04 -8.290892e-04
30%       1.160000e+02 -2.308225e-04 -3.375865e-04 -4.100041e-04 -4.732608e-04
40%       1.550000e+02 -9.680542e-05 -1.555210e-04 -1.871257e-04 -2.161311e-04
50%       1.940000e+02  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
60%       2.340000e+02  9.104980e-05  1.499700e-04  1.805591e-04  2.082899e-04
70%       2.730000e+02  2.255130e-04  3.308337e-04  4.010963e-04  4.651163e-04
80%       3.120000e+02  4.033559e-04  5.854801e-04  7.150892e-04  8.195378e-04
90%       3.510000e+02  7.384155e-04  1.041016e-03  1.264733e-03  1.447178e-03
max       3.890000e+02  9.877805e-02  1.051307e-01  1.034375e-01  1.911828e-01

count      ret5min      ret6min      ret7min      ret8min      ret9min  \
mean     -1.142065e-06  3.964908e-08  8.709935e-07  1.473304e-06  8.600771e-07
std       1.717925e-03  1.863826e-03  1.994273e-03  2.115630e-03  2.235502e-03
min     -1.545420e-01 -1.956444e-01 -2.158158e-01 -1.726063e-01 -2.157102e-01
10%     -1.616089e-03 -1.762741e-03 -1.897606e-03 -2.026686e-03 -2.142857e-03
20%     -9.218753e-04 -1.004307e-03 -1.081334e-03 -1.153403e-03 -1.218621e-03
30%     -5.260389e-04 -5.755245e-04 -6.196361e-04 -6.601532e-04 -6.987242e-04
40%     -2.398082e-04 -2.604845e-04 -2.820079e-04 -3.003003e-04 -3.175107e-04
50%      0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
60%      2.328158e-04  2.537642e-04  2.754062e-04  2.960551e-04  3.132832e-04
70%      5.172414e-04  5.666289e-04  6.132654e-04  6.540222e-04  6.938020e-04
80%      9.140768e-04  9.963062e-04  1.073346e-03  1.147301e-03  1.214083e-03
90%      1.610306e-03  1.759015e-03  1.891895e-03  2.018163e-03  2.133207e-03
max       1.327160e-01  2.006452e-01  1.245827e-01  1.317181e-01  1.121233e-01

count      ...      rup      rdown      BOP      CCI  \
mean      ...      5.109800e-01  5.258520e-01 -2.657472e-03  1.926504e-01
std       ...      7.497976e+00  3.099559e+01  6.268154e-01  1.088541e+02
min       ...      0.000000e+00  0.000000e+00 -1.000000e+00 -4.666667e+02

```

10%	...	0.000000e+00	0.000000e+00	-9.000000e-01	-1.377555e+02
20%	...	9.638554e-02	9.944259e-02	-6.644518e-01	-9.983731e+01
30%	...	1.976744e-01	2.017393e-01	-4.508475e-01	-6.917500e+01
40%	...	2.857143e-01	2.916667e-01	-1.826087e-01	-3.702543e+01
50%	...	3.701016e-01	3.759077e-01	0.000000e+00	4.377638e-01
60%	...	4.578505e-01	4.638802e-01	1.666667e-01	3.785810e+01
70%	...	5.551150e-01	5.599337e-01	4.428571e-01	6.964201e+01
80%	...	6.792615e-01	6.823138e-01	6.538462e-01	9.985441e+01
90%	...	8.656430e-01	8.662695e-01	9.000000e-01	1.375000e+02
max	...	5.500000e+03	5.000100e+04	1.000000e+00	4.666667e+02

	MFI	STOCHRSI	slowd	slowk	NATR \
count	3.087377e+06	3.087164e+06	3.087329e+06	3.087329e+06	3.087377e+06
mean	4.995851e+01	4.996170e+01	5.026232e+01	5.025505e+01	9.574717e-02
std	1.969340e+01	3.548760e+01	2.770983e+01	2.860330e+01	7.538252e-02
min	-1.837897e-08	0.000000e+00	-3.910354e-12	-5.092223e-12	2.026288e-07
10%	2.395492e+01	0.000000e+00	1.250572e+01	1.122968e+01	4.014445e-02
20%	3.237073e+01	9.771635e+00	2.094136e+01	1.994344e+01	4.871973e-02
30%	3.883638e+01	2.365167e+01	3.006536e+01	2.936605e+01	5.665948e-02
40%	4.452486e+01	3.687606e+01	3.997666e+01	3.958910e+01	6.520095e-02
50%	4.989353e+01	4.993994e+01	5.029630e+01	5.008741e+01	7.514502e-02
60%	5.529671e+01	6.304218e+01	6.069305e+01	6.111111e+01	8.752835e-02
70%	6.100298e+01	7.622753e+01	7.055556e+01	7.123016e+01	1.040583e-01
80%	6.753297e+01	9.012224e+01	7.961905e+01	8.057127e+01	1.289437e-01
90%	7.610552e+01	1.000000e+02	8.790211e+01	8.905852e+01	1.755765e-01
max	1.000000e+02	1.000000e+02	1.000000e+02	1.000000e+02	3.459402e+01

	trades_bid_ask
count	3.008146e+06
mean	-7.094037e-03
std	2.717903e+01
min	-2.630100e+04
10%	-7.299035e-01
20%	-4.601227e-01
30%	-2.782861e-01
40%	-1.309554e-01
50%	0.000000e+00
60%	1.036810e-01
70%	2.516316e-01
80%	4.365163e-01
90%	7.094017e-01
max	2.500100e+04

[14 rows x 22 columns]

## 1.3 Model Training

### 1.3.1 Helper functions

```
[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,
                  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[self.date_idx] <= days[train_end])].index
            test_idx = dates[(dates[self.date_idx] > days[test_start])
                              & (dates[self.date_idx] <= 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
```

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

### 1.3.2 Categorical Variables

```
[13]: data['stock_id'] = pd.factorize(data.index.get_level_values('ticker'),
    ↪sort=True)[0]
```

```
[14]: categoricals = ['stock_id']
```

### 1.3.3 Custom Metric

```
[15]: def ic_lgbm(preds, train_data):
      """Custom IC eval metric for lightgbm"""
      is_higher_better = True
      return 'ic', spearmanr(preds, train_data.get_label())[0], is_higher_better
```

### 1.3.4 Cross-validation setup

```
[16]: DAY = 390    # number of minute bars in a trading day of 6.5 hrs (9:30 - 15:59)
      MONTH = 21   # trading days
```

```
[17]: def get_cv(n_splits=23):
      return MultipleTimeSeriesCV(n_splits=n_splits,
                                  lookahead=1,
                                  test_period_length=MONTH * DAY,      # test
    ↪for 1 month
                                  train_period_length=12 * MONTH * DAY, # train
    ↪for 1 year
                                  date_idx='date_time')
```

Show train/validation periods:

```
[18]: for i, (train_idx, test_idx) in enumerate(get_cv().split(X=data)):
      train_dates = data.iloc[train_idx].index.unique('date_time')
      test_dates = data.iloc[test_idx].index.unique('date_time')
      print(train_dates.min(), train_dates.max(), test_dates.min(), test_dates.
    ↪max())
```

```
2016-11-29 15:59:00 2017-11-29 15:59:00 2017-11-30 09:30:00 2017-12-29 15:59:00
2016-10-28 15:47:00 2017-10-30 15:58:00 2017-10-30 15:59:00 2017-11-29 15:59:00
2016-09-29 15:47:00 2017-09-29 15:58:00 2017-09-29 15:59:00 2017-10-30 15:58:00
```

```

2016-08-30 15:47:00 2017-08-30 15:58:00 2017-08-30 15:59:00 2017-09-29 15:58:00
2016-08-01 15:47:00 2017-08-01 15:58:00 2017-08-01 15:59:00 2017-08-30 15:58:00
2016-06-30 15:47:00 2017-06-30 15:58:00 2017-06-30 15:59:00 2017-08-01 15:58:00
2016-06-01 15:47:00 2017-06-01 15:58:00 2017-06-01 15:59:00 2017-06-30 15:58:00
2016-05-02 15:47:00 2017-05-02 15:58:00 2017-05-02 15:59:00 2017-06-01 15:58:00
2016-04-01 15:47:00 2017-03-31 15:58:00 2017-03-31 15:59:00 2017-05-02 15:58:00
2016-03-02 15:47:00 2017-03-02 15:58:00 2017-03-02 15:59:00 2017-03-31 15:58:00
2016-02-01 15:47:00 2017-01-31 15:58:00 2017-01-31 15:59:00 2017-03-02 15:58:00
2015-12-30 15:47:00 2016-12-29 15:58:00 2016-12-29 15:59:00 2017-01-31 15:58:00
2015-11-30 15:23:00 2016-11-29 15:58:00 2016-11-29 15:59:00 2016-12-29 15:58:00
2015-10-29 15:09:00 2016-10-28 15:46:00 2016-10-28 15:47:00 2016-11-29 15:58:00
2015-09-30 15:09:00 2016-09-29 15:46:00 2016-09-29 15:47:00 2016-10-28 15:46:00
2015-08-31 15:09:00 2016-08-30 15:46:00 2016-08-30 15:47:00 2016-09-29 15:46:00
2015-07-31 15:09:00 2016-08-01 15:46:00 2016-08-01 15:47:00 2016-08-30 15:46:00
2015-07-01 15:09:00 2016-06-30 15:46:00 2016-06-30 15:47:00 2016-08-01 15:46:00
2015-06-02 15:09:00 2016-06-01 15:46:00 2016-06-01 15:47:00 2016-06-30 15:46:00
2015-05-01 15:09:00 2016-05-02 15:46:00 2016-05-02 15:47:00 2016-06-01 15:46:00
2015-04-01 15:09:00 2016-04-01 15:46:00 2016-04-01 15:47:00 2016-05-02 15:46:00
2015-03-03 15:09:00 2016-03-02 15:46:00 2016-03-02 15:47:00 2016-04-01 15:46:00
2015-01-30 15:09:00 2016-02-01 15:46:00 2016-02-01 15:47:00 2016-03-02 15:46:00

```

### 1.3.5 Train model

```

[19]: label = sorted(data.filter(like='fwd').columns)
      features = data.columns.difference(label).tolist()
      label = label[0]

```

```

[48]: params = dict(objective='regression',
                    metric=['rmse'],
                    device='gpu',
                    max_bin=63,
                    gpu_use_dp=False,
                    num_leaves=16,
                    min_data_in_leaf=500,
                    feature_fraction=.8,
                    verbose=-1)

```

```

[49]: num_boost_round = 250

```

```

[50]: cv = get_cv(n_splits=23) # we have enough data for 23 different test periods

```

```

[51]: def get_scores(result):
      return pd.DataFrame({'train': result['training']['ic'],
                          'valid': result['valid_1']['ic']})

```

The following model-training loop will take more than 10 hours to run and also consumes substantial memory. If you run into resource constraints, you can modify the code, e.g., by: 1. Only loading data required for one iteration. 2. Shortening the training period to require less than one year.

You can also speed up the process by using fewer `n_splits`, which implies longer test periods.

```
[52]: start = time()
for fold, (train_idx, test_idx) in enumerate(cv.split(X=data), 1):
    # create lgb train set
    train_set = data.iloc[train_idx, :]
    lgb_train = lgb.Dataset(data=train_set.drop(label, axis=1),
                           label=train_set[label],
                           categorical_feature=categoricals)

    # create lgb test set
    test_set = data.iloc[test_idx, :]
    lgb_test = lgb.Dataset(data=test_set.drop(label, axis=1),
                          label=test_set[label],
                          categorical_feature=categoricals,
                          reference=lgb_train)

    # train model
    evals_result = {}
    model = lgb.train(params=params,
                      train_set=lgb_train,
                      valid_sets=[lgb_train, lgb_test],
                      feval=ic_lgbm,
                      num_boost_round=num_boost_round,
                      evals_result=evals_result,
                      verbose_eval=50)

    model.save_model((model_path / f'fold:02.txt').as_posix())

    # get train/valid ic scores
    scores = get_scores(evals_result)
    scores.to_hdf(result_store, f'ic/fold:02')

    # get feature importance
    fi = get_fi(model)
    fi.to_hdf(result_store, f'fi/fold:02')

    # generate validation predictions
    X_test = test_set.loc[:, model.feature_name()]
    y_test = test_set.loc[:, [label]]
    y_test['pred'] = model.predict(X_test)
    y_test.to_hdf(result_store, f'predictions/fold:02')

    # compute average IC per minute
    by_minute = y_test.groupby(test_set.index.get_level_values('date_time'))
    daily_ic = by_minute.apply(lambda x: spearmanr(x[label], x.pred)[0]).mean()
    print(f'\nFold: {fold:02} | {format_time(time()-start)} | IC per minute:␣
    ↳{daily_ic:.2%}\n')
```



[50] training's rmse: 0.0006962 training's ic: 0.038731 valid\_1's rmse:  
0.000816226 valid\_1's ic: 0.0543727  
[100] training's rmse: 0.000695586 training's ic: 0.04416 valid\_1's rmse:  
0.000815993 valid\_1's ic: 0.0552591  
[150] training's rmse: 0.000695027 training's ic: 0.046986 valid\_1's rmse:  
0.000815898 valid\_1's ic: 0.0557145  
[200] training's rmse: 0.000694592 training's ic: 0.04948 valid\_1's rmse:  
0.000815859 valid\_1's ic: 0.0561737  
[250] training's rmse: 0.000694165 training's ic: 0.0517389  
valid\_1's rmse: 0.000815865 valid\_1's ic: 0.0558025

Fold: 01 | 00:17:46 | IC per minute: 5.59%

[50] training's rmse: 0.000699973 training's ic: 0.0376039  
valid\_1's rmse: 0.000847957 valid\_1's ic: 0.0416495  
[100] training's rmse: 0.000699303 training's ic: 0.0426195  
valid\_1's rmse: 0.000847627 valid\_1's ic: 0.043379  
[150] training's rmse: 0.000698748 training's ic: 0.0457404  
valid\_1's rmse: 0.000847548 valid\_1's ic: 0.043617  
[200] training's rmse: 0.000698298 training's ic: 0.0482473  
valid\_1's rmse: 0.000847537 valid\_1's ic: 0.0440953  
[250] training's rmse: 0.000697857 training's ic: 0.0506102  
valid\_1's rmse: 0.000847582 valid\_1's ic: 0.0439462

Fold: 02 | 00:35:29 | IC per minute: 4.45%

[50] training's rmse: 0.000698592 training's ic: 0.0370533  
valid\_1's rmse: 0.000706335 valid\_1's ic: 0.0404773  
[100] training's rmse: 0.000697869 training's ic: 0.0418831  
valid\_1's rmse: 0.000706128 valid\_1's ic: 0.0413394  
[150] training's rmse: 0.000697354 training's ic: 0.0452553  
valid\_1's rmse: 0.000706085 valid\_1's ic: 0.0411713  
[200] training's rmse: 0.000696885 training's ic: 0.0479669  
valid\_1's rmse: 0.000706038 valid\_1's ic: 0.0413983  
[250] training's rmse: 0.000696456 training's ic: 0.0503778  
valid\_1's rmse: 0.000706054 valid\_1's ic: 0.0412612

Fold: 03 | 00:57:48 | IC per minute: 4.45%

[50] training's rmse: 0.000701553 training's ic: 0.0363031  
valid\_1's rmse: 0.000669637 valid\_1's ic: 0.0326244  
[100] training's rmse: 0.000700849 training's ic: 0.0413249  
valid\_1's rmse: 0.000669565 valid\_1's ic: 0.0339486  
[150] training's rmse: 0.000700357 training's ic: 0.0447981  
valid\_1's rmse: 0.000669562 valid\_1's ic: 0.0343703  
[200] training's rmse: 0.000699884 training's ic: 0.0476104  
valid\_1's rmse: 0.000669583 valid\_1's ic: 0.0349983  
[250] training's rmse: 0.000699484 training's ic: 0.0501712

valid\_1's rmse: 0.000669543      valid\_1's ic: 0.0355025

Fold: 04 | 01:24:05 | IC per minute: 3.83%

[50]      training's rmse: 0.000697019      training's ic: 0.0354982  
valid\_1's rmse: 0.000697012      valid\_1's ic: 0.0247309  
[100]      training's rmse: 0.000696274      training's ic: 0.0410205  
valid\_1's rmse: 0.000696904      valid\_1's ic: 0.0271854  
[150]      training's rmse: 0.000695755      training's ic: 0.044584 valid\_1's rmse:  
0.000696912      valid\_1's ic: 0.0276005  
[200]      training's rmse: 0.000695313      training's ic: 0.0474853  
valid\_1's rmse: 0.000696927      valid\_1's ic: 0.0285591  
[250]      training's rmse: 0.000694863      training's ic: 0.0498696  
valid\_1's rmse: 0.000696917      valid\_1's ic: 0.0285991

Fold: 05 | 01:50:23 | IC per minute: 3.13%

[50]      training's rmse: 0.00069678      training's ic: 0.0350113  
valid\_1's rmse: 0.000701348      valid\_1's ic: 0.0275999  
[100]      training's rmse: 0.00069605      training's ic: 0.0406079  
valid\_1's rmse: 0.000701289      valid\_1's ic: 0.0297336  
[150]      training's rmse: 0.000695473      training's ic: 0.0441527  
valid\_1's rmse: 0.000701216      valid\_1's ic: 0.0307175  
[200]      training's rmse: 0.000694997      training's ic: 0.0471703  
valid\_1's rmse: 0.000701244      valid\_1's ic: 0.0314352  
[250]      training's rmse: 0.000694559      training's ic: 0.0492445  
valid\_1's rmse: 0.000701273      valid\_1's ic: 0.0314369

Fold: 06 | 02:16:28 | IC per minute: 3.34%

[50]      training's rmse: 0.000702829      training's ic: 0.0337797  
valid\_1's rmse: 0.000744246      valid\_1's ic: 0.0246692  
[100]      training's rmse: 0.00070212      training's ic: 0.0385954  
valid\_1's rmse: 0.000744224      valid\_1's ic: 0.0264151  
[150]      training's rmse: 0.000701593      training's ic: 0.0430637  
valid\_1's rmse: 0.000744229      valid\_1's ic: 0.0275546  
[200]      training's rmse: 0.000701114      training's ic: 0.0458159  
valid\_1's rmse: 0.000744281      valid\_1's ic: 0.0282104  
[250]      training's rmse: 0.000700721      training's ic: 0.0482636  
valid\_1's rmse: 0.000744313      valid\_1's ic: 0.0283922

Fold: 07 | 02:42:44 | IC per minute: 3.28%

[50]      training's rmse: 0.000722509      training's ic: 0.0334184  
valid\_1's rmse: 0.00062052      valid\_1's ic: 0.032487  
[100]      training's rmse: 0.000721876      training's ic: 0.038585 valid\_1's rmse:  
0.000620422      valid\_1's ic: 0.0333264  
[150]      training's rmse: 0.000721342      training's ic: 0.0423346

valid\_1's rmse: 0.000620373      valid\_1's ic: 0.0332792  
[200]    training's rmse: 0.000720854      training's ic: 0.0453648  
valid\_1's rmse: 0.000620391      valid\_1's ic: 0.0344978  
[250]    training's rmse: 0.00072039      training's ic: 0.0475421  
valid\_1's rmse: 0.000620433      valid\_1's ic: 0.0349232

Fold: 08 | 03:08:25 | IC per minute: 3.70%

[50]    training's rmse: 0.000752768      training's ic: 0.0325142  
valid\_1's rmse: 0.0005842      valid\_1's ic: 0.0271741  
[100]    training's rmse: 0.000751985      training's ic: 0.0374633  
valid\_1's rmse: 0.000584136      valid\_1's ic: 0.0283447  
[150]    training's rmse: 0.000751343      training's ic: 0.0407396  
valid\_1's rmse: 0.000584099      valid\_1's ic: 0.0289354  
[200]    training's rmse: 0.000750835      training's ic: 0.0439565  
valid\_1's rmse: 0.000584126      valid\_1's ic: 0.0294128  
[250]    training's rmse: 0.00075033      training's ic: 0.0460732  
valid\_1's rmse: 0.000584183      valid\_1's ic: 0.0293556

Fold: 09 | 03:34:14 | IC per minute: 3.21%

[50]    training's rmse: 0.000772983      training's ic: 0.0315982  
valid\_1's rmse: 0.00063351      valid\_1's ic: 0.0269043  
[100]    training's rmse: 0.000772305      training's ic: 0.0370821  
valid\_1's rmse: 0.000633424      valid\_1's ic: 0.0295316  
[150]    training's rmse: 0.000771751      training's ic: 0.0402892  
valid\_1's rmse: 0.000633369      valid\_1's ic: 0.0301651  
[200]    training's rmse: 0.000771242      training's ic: 0.0432137  
valid\_1's rmse: 0.000633349      valid\_1's ic: 0.0312183  
[250]    training's rmse: 0.000770771      training's ic: 0.0455847  
valid\_1's rmse: 0.000633325      valid\_1's ic: 0.0315627

Fold: 10 | 04:00:30 | IC per minute: 2.98%

[50]    training's rmse: 0.000832092      training's ic: 0.0325253  
valid\_1's rmse: 0.000653653      valid\_1's ic: 0.026781  
[100]    training's rmse: 0.000831323      training's ic: 0.0377314  
valid\_1's rmse: 0.000653568      valid\_1's ic: 0.0289015  
[150]    training's rmse: 0.000830753      training's ic: 0.0411433  
valid\_1's rmse: 0.000653586      valid\_1's ic: 0.0291601  
[200]    training's rmse: 0.000830191      training's ic: 0.043913 valid\_1's rmse:  
0.000653599      valid\_1's ic: 0.0301002  
[250]    training's rmse: 0.000829674      training's ic: 0.0465464  
valid\_1's rmse: 0.000653658      valid\_1's ic: 0.0303744

Fold: 11 | 04:26:17 | IC per minute: 2.94%

[50]    training's rmse: 0.000877395      training's ic: 0.0320049

valid\_1's rmse: 0.000721517      valid\_1's ic: 0.0240198  
[100]    training's rmse: 0.000876658      training's ic: 0.0374841  
valid\_1's rmse: 0.00072146      valid\_1's ic: 0.026157  
[150]    training's rmse: 0.000876046      training's ic: 0.0408182  
valid\_1's rmse: 0.000721393      valid\_1's ic: 0.0272646  
[200]    training's rmse: 0.000875495      training's ic: 0.0441758  
valid\_1's rmse: 0.000721363      valid\_1's ic: 0.0281185  
[250]    training's rmse: 0.000875026      training's ic: 0.0467237  
valid\_1's rmse: 0.00072137      valid\_1's ic: 0.028905

Fold: 12 | 04:52:49 | IC per minute: 3.04%

[50]    training's rmse: 0.000886972      training's ic: 0.0326955  
valid\_1's rmse: 0.000749551      valid\_1's ic: 0.0260998  
[100]    training's rmse: 0.000886233      training's ic: 0.0374855  
valid\_1's rmse: 0.00074944      valid\_1's ic: 0.0283205  
[150]    training's rmse: 0.000885641      training's ic: 0.0409926  
valid\_1's rmse: 0.000749411      valid\_1's ic: 0.029227  
[200]    training's rmse: 0.000885103      training's ic: 0.0439042  
valid\_1's rmse: 0.000749372      valid\_1's ic: 0.0297628  
[250]    training's rmse: 0.000884651      training's ic: 0.0465908  
valid\_1's rmse: 0.000749306      valid\_1's ic: 0.0307105

Fold: 13 | 05:18:51 | IC per minute: 3.01%

[50]    training's rmse: 0.000892264      training's ic: 0.0326621  
valid\_1's rmse: 0.00088496      valid\_1's ic: 0.0215666  
[100]    training's rmse: 0.000891562      training's ic: 0.0366921  
valid\_1's rmse: 0.000884886      valid\_1's ic: 0.0220376  
[150]    training's rmse: 0.000890964      training's ic: 0.0397876  
valid\_1's rmse: 0.000884839      valid\_1's ic: 0.0227016  
[200]    training's rmse: 0.000890451      training's ic: 0.0430167  
valid\_1's rmse: 0.000884803      valid\_1's ic: 0.0235889  
[250]    training's rmse: 0.000889943      training's ic: 0.0452669  
valid\_1's rmse: 0.000884774      valid\_1's ic: 0.0240788

Fold: 14 | 05:45:07 | IC per minute: 2.86%

[50]    training's rmse: 0.000921495      training's ic: 0.0325343  
valid\_1's rmse: 0.000688911      valid\_1's ic: 0.0223877  
[100]    training's rmse: 0.00092084      training's ic: 0.0366749  
valid\_1's rmse: 0.000688793      valid\_1's ic: 0.0239436  
[150]    training's rmse: 0.000920176      training's ic: 0.0401455  
valid\_1's rmse: 0.00068875      valid\_1's ic: 0.0249856  
[200]    training's rmse: 0.000919602      training's ic: 0.0432488  
valid\_1's rmse: 0.000688764      valid\_1's ic: 0.0256182  
[250]    training's rmse: 0.000919108      training's ic: 0.0458315  
valid\_1's rmse: 0.000688732      valid\_1's ic: 0.0265407

Fold: 15 | 06:11:36 | IC per minute: 2.68%

[50] training's rmse: 0.000940675 training's ic: 0.0333497  
valid\_1's rmse: 0.00070608 valid\_1's ic: 0.0200963  
[100] training's rmse: 0.000939891 training's ic: 0.0377662  
valid\_1's rmse: 0.000706092 valid\_1's ic: 0.020633  
[150] training's rmse: 0.000939188 training's ic: 0.0414858  
valid\_1's rmse: 0.000706075 valid\_1's ic: 0.021742  
[200] training's rmse: 0.000938638 training's ic: 0.0441729  
valid\_1's rmse: 0.00070609 valid\_1's ic: 0.0223267  
[250] training's rmse: 0.000938117 training's ic: 0.0468418  
valid\_1's rmse: 0.000706121 valid\_1's ic: 0.0225305

Fold: 16 | 06:38:11 | IC per minute: 2.44%

[50] training's rmse: 0.000985282 training's ic: 0.0324179  
valid\_1's rmse: 0.000640303 valid\_1's ic: 0.0209769  
[100] training's rmse: 0.00098423 training's ic: 0.0362766  
valid\_1's rmse: 0.000640323 valid\_1's ic: 0.0216562  
[150] training's rmse: 0.000983366 training's ic: 0.0396048  
valid\_1's rmse: 0.000640393 valid\_1's ic: 0.0223887  
[200] training's rmse: 0.000982623 training's ic: 0.042354 valid\_1's rmse:  
0.000640399 valid\_1's ic: 0.0228008  
[250] training's rmse: 0.000981903 training's ic: 0.0447996  
valid\_1's rmse: 0.000640409 valid\_1's ic: 0.0235311

Fold: 17 | 07:04:30 | IC per minute: 2.60%

[50] training's rmse: 0.000992882 training's ic: 0.0330731  
valid\_1's rmse: 0.000698768 valid\_1's ic: 0.0178816  
[100] training's rmse: 0.000991763 training's ic: 0.0369799  
valid\_1's rmse: 0.000698784 valid\_1's ic: 0.0188669  
[150] training's rmse: 0.000990925 training's ic: 0.0401558  
valid\_1's rmse: 0.00069884 valid\_1's ic: 0.0197579  
[200] training's rmse: 0.00099016 training's ic: 0.0430659  
valid\_1's rmse: 0.00069889 valid\_1's ic: 0.0204069  
[250] training's rmse: 0.000989494 training's ic: 0.0454836  
valid\_1's rmse: 0.000698912 valid\_1's ic: 0.021086

Fold: 18 | 07:23:22 | IC per minute: 2.47%

[50] training's rmse: 0.000981605 training's ic: 0.0333102  
valid\_1's rmse: 0.000807922 valid\_1's ic: 0.0192318  
[100] training's rmse: 0.000980441 training's ic: 0.0371727  
valid\_1's rmse: 0.000807994 valid\_1's ic: 0.0198469  
[150] training's rmse: 0.000979597 training's ic: 0.040212 valid\_1's rmse:  
0.000808115 valid\_1's ic: 0.0198447

[200] training's rmse: 0.000978876 training's ic: 0.0429504  
valid\_1's rmse: 0.000808122 valid\_1's ic: 0.0202568  
[250] training's rmse: 0.000978225 training's ic: 0.0454618  
valid\_1's rmse: 0.000808137 valid\_1's ic: 0.0204947

Fold: 19 | 07:42:22 | IC per minute: 2.58%

[50] training's rmse: 0.000971273 training's ic: 0.0343452  
valid\_1's rmse: 0.00084749 valid\_1's ic: 0.0205258  
[100] training's rmse: 0.000970176 training's ic: 0.0383209  
valid\_1's rmse: 0.000847495 valid\_1's ic: 0.0222474  
[150] training's rmse: 0.000969198 training's ic: 0.0409799  
valid\_1's rmse: 0.000847536 valid\_1's ic: 0.0226757  
[200] training's rmse: 0.000968461 training's ic: 0.0437774  
valid\_1's rmse: 0.000847519 valid\_1's ic: 0.0232256  
[250] training's rmse: 0.000967769 training's ic: 0.0463843  
valid\_1's rmse: 0.000847529 valid\_1's ic: 0.0236719

Fold: 20 | 08:00:44 | IC per minute: 2.79%

[50] training's rmse: 0.000956095 training's ic: 0.0343668  
valid\_1's rmse: 0.00093566 valid\_1's ic: 0.0210374  
[100] training's rmse: 0.000955025 training's ic: 0.0392049  
valid\_1's rmse: 0.000935819 valid\_1's ic: 0.022133  
[150] training's rmse: 0.000954102 training's ic: 0.0422933  
valid\_1's rmse: 0.0009359 valid\_1's ic: 0.0228522  
[200] training's rmse: 0.000953454 training's ic: 0.0448814  
valid\_1's rmse: 0.000935966 valid\_1's ic: 0.0233652  
[250] training's rmse: 0.000952775 training's ic: 0.0473471  
valid\_1's rmse: 0.000936005 valid\_1's ic: 0.0231158

Fold: 21 | 08:19:21 | IC per minute: 2.43%

[50] training's rmse: 0.000945276 training's ic: 0.0343428  
valid\_1's rmse: 0.000878341 valid\_1's ic: 0.0227607  
[100] training's rmse: 0.000944164 training's ic: 0.0389748  
valid\_1's rmse: 0.000878351 valid\_1's ic: 0.0246803  
[150] training's rmse: 0.000943245 training's ic: 0.0416026  
valid\_1's rmse: 0.00087842 valid\_1's ic: 0.0257048  
[200] training's rmse: 0.000942459 training's ic: 0.0444224  
valid\_1's rmse: 0.000878479 valid\_1's ic: 0.0260882  
[250] training's rmse: 0.000941729 training's ic: 0.0464706  
valid\_1's rmse: 0.000878522 valid\_1's ic: 0.0260996

Fold: 22 | 08:38:07 | IC per minute: 2.97%

[50] training's rmse: 0.000901678 training's ic: 0.0344405  
valid\_1's rmse: 0.00124889 valid\_1's ic: 0.0247168

```

[100] training's rmse: 0.000900504 training's ic: 0.0387862
valid_1's rmse: 0.00124921 valid_1's ic: 0.0242162
[150] training's rmse: 0.000899561 training's ic: 0.0426923
valid_1's rmse: 0.00124947 valid_1's ic: 0.0241308
[200] training's rmse: 0.00089887 training's ic: 0.045369 valid_1's rmse:
0.00124959 valid_1's ic: 0.0242198
[250] training's rmse: 0.000898202 training's ic: 0.0477219
valid_1's rmse: 0.00124973 valid_1's ic: 0.0247126

```

Fold: 23 | 08:56:41 | IC per minute: 3.17%

## 1.4 Signal Evaluation

```

[112]: with pd.HDFStore(result_store) as store:
        pred_keys = [k[1:] for k in store.keys() if k[1:].startswith('pred')]
        cv_predictions = pd.concat([store[k] for k in pred_keys]).sort_index()

```

```

[113]: cv_predictions.info(null_counts=True)

```

```

<class 'pandas.core.frame.DataFrame'>
MultiIndex: 19648064 entries, ('AAL', Timestamp('2016-02-01 15:47:00')) to
('YH00', Timestamp('2017-06-16 15:59:00'))
Data columns (total 2 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   fwd1min     19648064 non-null  float64
1   pred        19648064 non-null  float64
dtypes: float64(2)
memory usage: 399.0+ MB

```

```

[114]: time_stamp = cv_predictions.index.get_level_values('date_time')
        dates = sorted(np.unique(time_stamp.date))

```

We have out-of-sample predictions for 484 days from February 2016 through December 2017:

```

[116]: print(f'# Days: {len(dates)} | First: {dates[0]} | Last: {dates[-1]}')

```

```
# Days: 484 | First: 2016-02-01 | Last: 2017-12-29
```

We only use minutes with at least 100 predictions:

```

[117]: n = cv_predictions.groupby('date_time').size()

```

There are ~700 periods, equivalent to a bit over a single trading day (0.67% of all periods in the sample), with fewer than 100 predictions over the 23 test months:

```

[120]: incomplete_minutes = n[n<100].index

```

```
[124]: print(f'{len(incomplete_minutes)} ({len(incomplete_minutes)/len(n):.2%})')
```

```
1255 (0.67%)
```

```
[125]: cv_predictions = cv_predictions[~time_stamp.isin(incomplete_minutes)]
```

```
[126]: cv_predictions.info(null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 19571774 entries, ('AAL', Timestamp('2016-02-01 15:47:00')) to
('YH00', Timestamp('2017-06-16 15:59:00'))
Data columns (total 2 columns):
#   Column   Non-Null Count  Dtype
---  -
0   fwd1min  19571774 non-null  float64
1   pred     19571774 non-null  float64
dtypes: float64(2)
memory usage: 397.4+ MB
```

### 1.4.1 Information Coefficient

Across all periods

```
[127]: ic = spearmanr(cv_predictions.fwd1min, cv_predictions.pred)[0]
```

**By minute** We are making new predictions every minute, so it makes sense to look at the average performance across all short-term forecasts:

```
[132]: minutes = cv_predictions.index.get_level_values('date_time')
by_minute = cv_predictions.groupby(minutes)
```

```
[129]: ic_by_minute = by_minute.apply(lambda x: spearmanr(x.fwd1min, x.pred)[0])

minute_ic_mean = ic_by_minute.mean()
minute_ic_median = ic_by_minute.median()

print(f'\nAll periods: {ic:6.2%} | By Minute: {minute_ic_mean: 6.2%} (Median:
↳{minute_ic_median: 6.2%})')
```

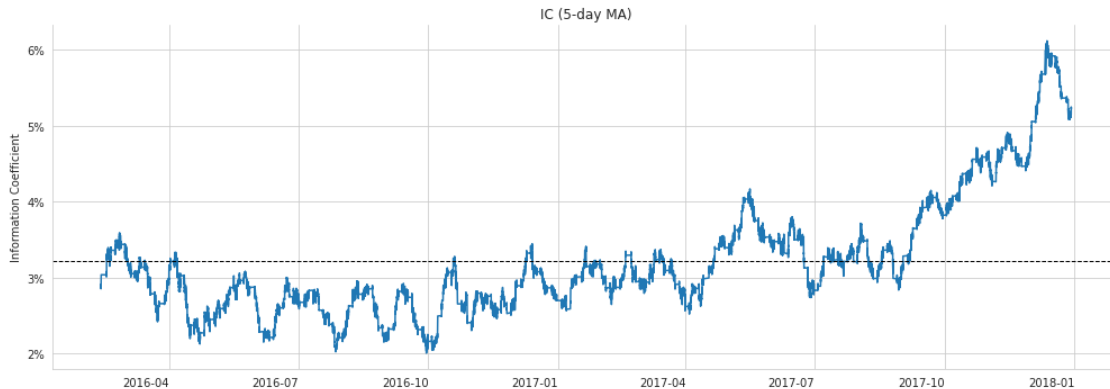
```
All periods: 2.96% | By Minute: 3.21% (Median: 3.23%)
```

Plotted as a five-day rolling average, we see that the IC was mostly below the out-of-sample period mean, and increased during the last quarter of 2017 (as reflected in the validation results we observed while training the model).

```
[279]: ax = ic_by_minute.rolling(5*650).mean().plot(figsize=(14, 5), title='IC (5-day
↳MA)', rot=0)
ax.axhline(minute_ic_mean, ls='--', lw=1, c='k')
```



```
ax.yaxis.set_major_formatter(FuncFormatter(lambda y, _: '{:.0%}'.format(y)))
ax.set_ylabel('Information Coefficient')
ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```



### 1.4.2 Vectorized backtest of a naive strategy: financial performance by signal quantile

Alphalens does not work with minute-data, so we need to compute our own signal performance measures.

Unfortunately, Zipline's Pipeline also doesn't work for minute-data and Backtrader takes a very long time with such a large dataset. Hence, instead of an event-driven backtest of entry/exit rules as in previous examples, we can only create a rough sketch of the financial performance of a naive trading strategy driven by the model's predictions using vectorized backtesting (see Chapter 8 on the [ML4T workflow](#)). As we will see below, this does not produce particularly helpful results.

This naive strategy invests in equal-weighted portfolios of the stocks in each decile under the following assumptions (mentioned at the beginning of this notebook: 1. Based on the predictions using inputs from the current and previous bars, we can enter positions at the first trade price in the following minute bar 2. We exit all positions at the last price in that following minute bar 3. There are no trading cost or market impact (slippage) of our trades (but we can check how sensitive the results would be).

**Average returns by minute bar and signal quantile** To this end, we compute the quintiles and deciles of the model's `fwd1min` predictions for each minute:

```
[133]: by_minute = cv_predictions.groupby(minutes, group_keys=False)
```

```
[134]: labels = list(range(1, 6))
cv_predictions['quintile'] = by_minute.apply(lambda x: pd.qcut(x.pred, q=5,
↪ labels=labels).astype(int))
```

```
[135]: labels = list(range(1, 11))
cv_predictions['decile'] = by_minute.apply(lambda x: pd.qcut(x.pred, q=10,
↳ labels=labels).astype(int))
```

```
[136]: cv_predictions.info(show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 19571774 entries, ('AAL', Timestamp('2016-02-01 15:47:00')) to
('YH00', Timestamp('2017-06-16 15:59:00'))
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   fwd1min     19571774 non-null  float64
1   pred        19571774 non-null  float64
2   quintile    19571774 non-null  int64
3   decile      19571774 non-null  int64
dtypes: float64(2), int64(2)
memory usage: 696.1+ MB
```

### Descriptive statistics of intraday returns by quintile and decile of model predictions

Next, we compute the average one-minute returns for each quintile / decile and minute.

```
[319]: def compute_intraday_returns_by_quantile(predictions, quantile='quintile'):
        by_quantile = cv_predictions.reset_index().groupby(['date_time', quantile])
        return by_quantile.fwd1min.mean().unstack(quantile).sort_index()
```

```
[330]: intraday_returns = {'quintile':
↳ compute_intraday_returns_by_quantile(cv_predictions,
        'decile':
↳ compute_intraday_returns_by_quantile(cv_predictions, quantile='decile')}
```

```
[334]: def summarize_intraday_returns(returns):
        summary = returns.describe(deciles)
        return pd.concat([summary.iloc[:1].applymap(lambda x: f'{x:,.0f}'),
        summary.iloc[1:].applymap(lambda x: f'{x:.4%}')] )
```

The returns per minute, averaged over the 23-months period, increase by quintile/decile and range from -.3 (-.4) to .27 (.37) basis points for the bottom and top quintile (decile), respectively. While this aligns with the finding of a weakly positive rank correlation coefficient, it also suggests that such small gains are unlikely to survive the impact of trading costs.

```
[335]: summary = summarize_intraday_returns(intraday_returns['quintile'])
summary
```

```
[335]: quintile      1          2          3          4          5
count      187,115    187,115    187,115    187,115    187,115
mean      -0.0030%   -0.0011%   -0.0002%    0.0007%    0.0027%
```

std	0.0368%	0.0308%	0.0300%	0.0306%	0.0366%
min	-0.6316%	-0.4878%	-0.4880%	-0.5088%	-0.5918%
10%	-0.0397%	-0.0325%	-0.0307%	-0.0300%	-0.0326%
20%	-0.0234%	-0.0190%	-0.0178%	-0.0170%	-0.0177%
30%	-0.0145%	-0.0114%	-0.0105%	-0.0097%	-0.0094%
40%	-0.0080%	-0.0057%	-0.0049%	-0.0042%	-0.0032%
50%	-0.0023%	-0.0007%	-0.0000%	0.0008%	0.0024%
60%	0.0033%	0.0042%	0.0048%	0.0057%	0.0079%
70%	0.0096%	0.0098%	0.0104%	0.0113%	0.0144%
80%	0.0179%	0.0171%	0.0176%	0.0186%	0.0230%
90%	0.0328%	0.0300%	0.0302%	0.0317%	0.0383%
max	0.8794%	0.4456%	0.7358%	0.7256%	0.9139%

```
[336]: summary = summarize_intraday_returns(intraday_returns['decile'])
summary
```

```
[336]: decile      1      2      3      4      5      6      7  \
count    187,115    187,115    187,115    187,115    187,115    187,115    187,115
mean     -0.0040%   -0.0020%   -0.0013%   -0.0009%   -0.0003%   -0.0000%    0.0005%
std       0.0448%    0.0369%    0.0342%    0.0333%    0.0328%    0.0329%    0.0332%
min      -0.7406%   -1.2023%   -0.5405%   -0.5460%   -0.9286%   -0.9563%   -0.5948%
10%      -0.0485%   -0.0392%   -0.0365%   -0.0351%   -0.0341%   -0.0338%   -0.0332%
20%      -0.0288%   -0.0233%   -0.0215%   -0.0208%   -0.0200%   -0.0198%   -0.0192%
30%      -0.0180%   -0.0142%   -0.0130%   -0.0124%   -0.0119%   -0.0116%   -0.0111%
40%      -0.0101%   -0.0074%   -0.0066%   -0.0061%   -0.0057%   -0.0054%   -0.0050%
50%      -0.0031%   -0.0015%   -0.0010%   -0.0006%   -0.0001%    0.0002%    0.0005%
60%       0.0038%    0.0045%    0.0047%    0.0049%    0.0053%    0.0056%    0.0060%
70%       0.0116%    0.0110%    0.0110%    0.0111%    0.0115%    0.0118%    0.0123%
80%       0.0218%    0.0196%    0.0193%    0.0192%    0.0195%    0.0198%    0.0204%
90%       0.0395%    0.0348%    0.0334%    0.0332%    0.0331%    0.0337%    0.0343%
max       0.8348%    1.0942%    0.5067%    0.7221%    0.7832%    1.3252%    1.0380%

decile      8      9      10
count    187,115    187,115    187,115
mean      0.0010%    0.0017%    0.0037%
std       0.0339%    0.0365%    0.0448%
min      -0.6565%   -0.5173%   -1.1758%
10%      -0.0331%   -0.0340%   -0.0389%
20%      -0.0191%   -0.0192%   -0.0210%
30%      -0.0109%   -0.0107%   -0.0112%
40%      -0.0046%   -0.0043%   -0.0037%
50%       0.0009%    0.0015%    0.0030%
60%       0.0065%    0.0073%    0.0098%
70%       0.0128%    0.0139%    0.0175%
80%       0.0211%    0.0225%    0.0280%
90%       0.0353%    0.0376%    0.0471%
max       1.3348%    1.3651%    1.1961%
```

**Cumulative Performance by Quantile** To simulate the performance of our naive strategy that trades all available stocks every minute, we simply assume that we can reinvest (including potential gains/losses) every minute. To check for the sensitivity with respect for trading cost, we can assume they are a constant number (fraction) of basis points, and subtract this number from the minute-bar returns.

```
[367]: def plot_cumulative_performance(returns, quantile='quintile',
    ↪trading_costs_bp=0):
    """Plot average return by quantile (in bp) as well as cumulative return,
        both net of trading costs (provided as basis points; 1bp = 0.01%)
        """

    fig, axes = plt.subplots(figsize=(14, 4), ncols=2)

    sns.barplot(y='fwd1min', x=quantile,
                data=returns[quantile].mul(10000).sub(trading_costs_bp).stack().
    ↪to_frame(
                'fwd1min').reset_index(),
                ax=axes[0])
    axes[0].set_title(f'Avg. 1-min Return by Signal {quantile.capitalize()}')
    axes[0].set_ylabel('Return (bps)')
    axes[0].set_xlabel(quantile.capitalize())

    title = f'Cumulative Return by Signal {quantile.capitalize()}'
    (returns[quantile].sort_index().add(1).sub(trading_costs_bp/10000).
    ↪cumprod().sub(1)
    .plot(ax=axes[1], title=title))

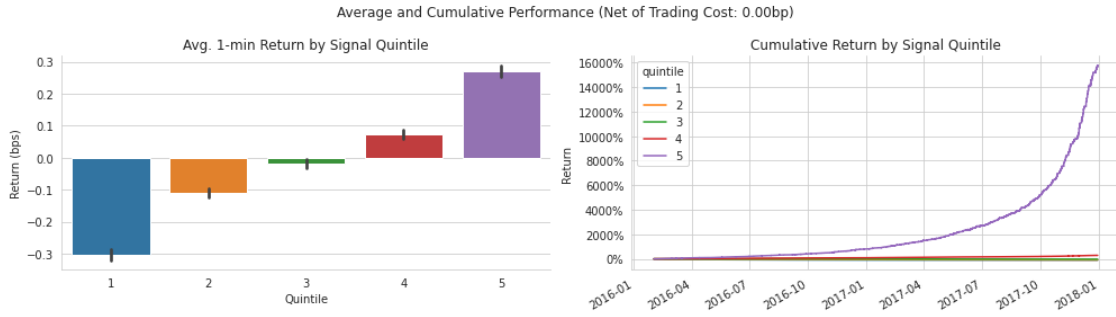
    axes[1].yaxis.set_major_formatter(
        FuncFormatter(lambda y, _: '{:.0%}'.format(y)))
    axes[1].set_xlabel('')
    axes[1].set_ylabel('Return')
    fig.suptitle(f'Average and Cumulative Performance (Net of Trading Cost:
    ↪{trading_costs_bp:.2f}bp)')

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

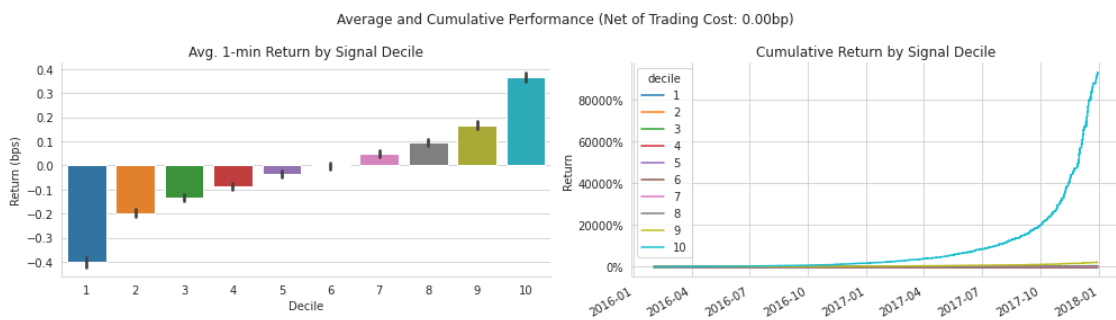
Without trading costs, the compounding of even fairly small gains leads to extremely large cumulative profits for the top quantile. However, these disappear as soon as we allow for minuscule trading costs that reduce the average quantile return close to zero.

### Without trading costs

```
[368]: plot_cumulative_performance(intraday_returns, 'quintile', trading_costs_bp=0)
```

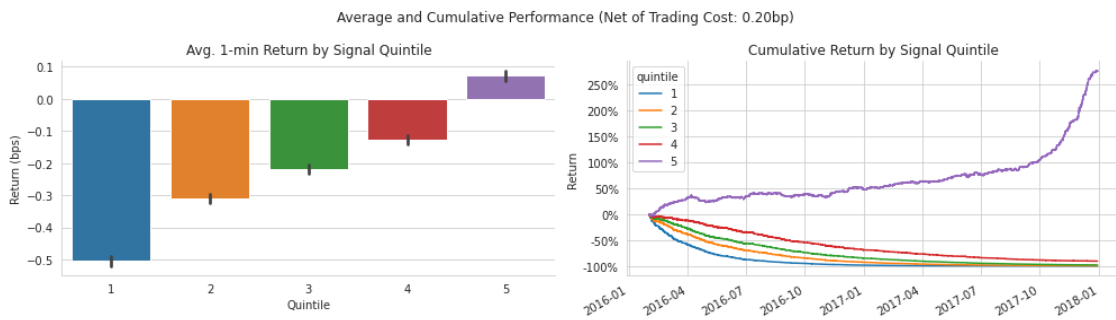


```
[369]: plot_cumulative_performance(intraday_returns, 'decile', trading_costs_bp=0)
```

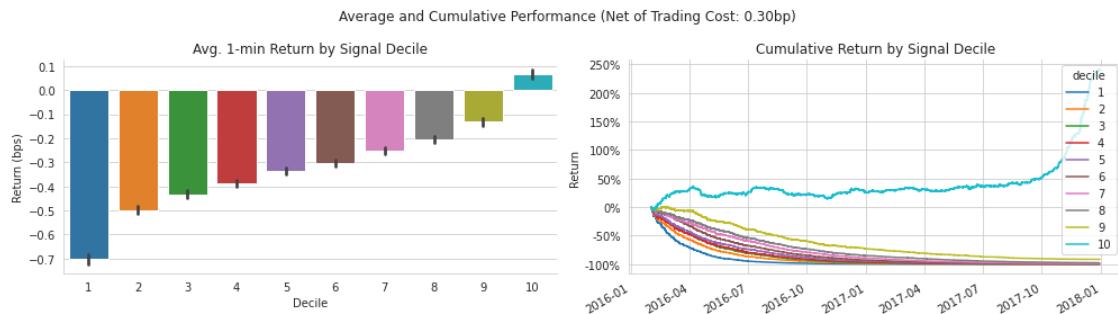


With extremely low trading costs

```
[370]: # assuming costs of a fraction of a basis point, close to the average return of
        ↪ the top quantile
        plot_cumulative_performance(intraday_returns, 'quintile', trading_costs_bp=.2)
```



```
[371]: plot_cumulative_performance(intraday_returns, 'decile', trading_costs_bp=.3)
```



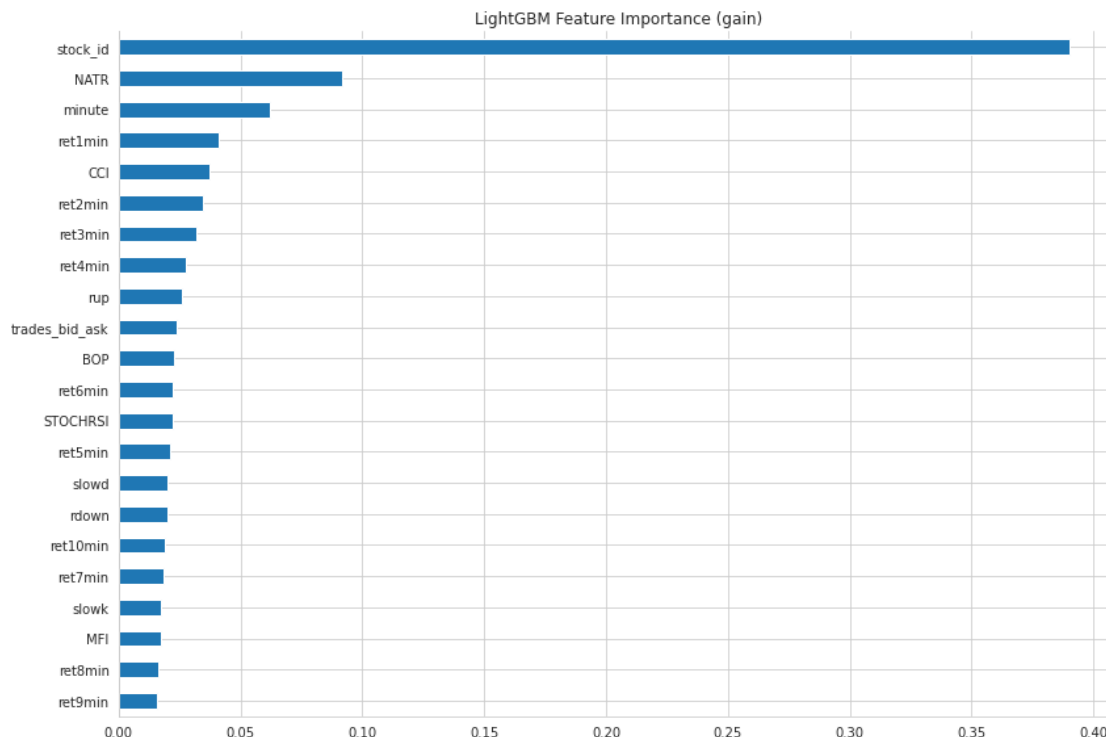
### 1.4.3 Feature Importance

We'll take a quick look at the features that most contributed to improving the IC across the 23 folds:

```
[235]: with pd.HDFStore(result_store) as store:
        fi_keys = [k[1:] for k in store.keys() if k[1:].startswith('fi')]
        fi = pd.concat([store[k].to_frame(i) for i, k in enumerate(fi_keys, 1)],
                        ↪axis=1)
```

The top features from a conventional feature importance perspective are the ticker, followed by NATR, minute of the day, latest 1m return and the CCI:

```
[254]: fi.mean(1).nsmallest(25).plot.barh(figsize=(12, 8), title='LightGBM Feature_
        ↪Importance (gain)')
        sns.despine()
        plt.tight_layout();
```



Explore with greater accuracy and in more detail how feature values affect predictions using SHAP values as demonstrated in various other notebooks in this Chapter and the appendix!

## 1.5 Conclusion

We have seen that a relatively simple gradient boosting model is able to achieve fairly consistent predictive performance that is significantly better than a random guess even on a very short horizon.

However, the resulting economic gains of our naive strategy of frequently buying/(short-)selling the top/bottom quantiles are too small to overcome the inevitable transaction costs. On the one hand, this demonstrates the challenges of extracting value from a predictive signal. On the other hand, it shows that we need a more sophisticated backtesting platform so that we can even begin to design and evaluate a more sophisticated strategy that requires far fewer trades to exploit the signal in our ML predictions.

In addition, we would also want to work on improving the model by adding more informative feature, e.g. based on the quote/trade info contained in the Algoseek data, or by fine-tuning our model architecture and hyperparameter settings.