Report

QR Projects Report

Zhao Ziwen

2023.12

Catalog

1 Correlation Coefficient Prediction

1.1 Project Instructions

1.1.1 Project Goal

Predicting Correlation of Selected Stocks with the SPY Index.

1.1.2 Project Objective

Objective: Predict the correlation of GOOG, AMZN, JPM, GME, and XOM with the SPY index for the next two months using only historical prices.

1.1.3 Instructions

Data Collection: Download the historical price data for the stocks GOOG, AMZN, JPM, GME, XOM, and the SPY index from Yahoo Finance or other sources. Ensure to account for corporate adjustments.

Data Preprocessing: Clean the data by handling any missing values, outliers, or anomalies. Check for corporate adjustments and account for them.

Model Selection: Choose an appropriate model to predict the future correlations. Consider models suitable for time series forecasting.

Model Training and Validation: Train the chosen model using historical correlations and validate its performance.

Evaluation: Assess the model's predictive accuracy using suitable metrics.

Report: Present your findings detailing the approach, choice of model, and achieved results. Please attach your code.

1.2 Data Collection

1.2.1 Data Sources

Collection financial data online can be accomplished through various methods, depending on

the specific type of data you're looking for and your intended use. Here are some sources: Financial Websites(Yahoo Finance, Google Finance), Financial News Websites(Bloomberg, CNBC, Reuters), Stock Exchanges(NYSE, NASDAQ), Financial Data APIs(Yahoo Finance API). I chose yfinance, which is a third-party library for Yahoo financial data collection based on Python. The data collected from yfinance is not only complete, but also contains information related to restoration of rights, so I think this is a convenient and fast data source. Table 1 shows the GOOG stock price obtained through yfinance.

Date	Open	High	Low	Close	Adj Close	Volume	Dividends	Stock Splits
2004/9/3 0:00	2.5143261	2.5340021	2.4737279	2.4909129	2.4909129	103538639	0	0
2004/9/1 0:00	2.5579121	2.5646369	2.482445	2.496891	2.496891	183633734	0	0
2004/8/19 0:00	2.490664	2.591785	2.3900421	2.4991331	2.4991331	897427216	0	0
2004/9/2 0:00	2.47049	2.5496931	2.464263	2.5282731	2.5282731	303810504	0	0
2004/9/7 0:00	2.51582	2.540478	2.4809511	2.5300169	2.5300169	117506800	0	0
2004/8/30 0:00	2.6221709	2.6274021	2.5407269	2.5407269	2.5407269	104429967	0	0
2004/9/8 0:00	2.509095	2.5661321	2.503118	2.54795	2.54795	100186120	0	0
2004/9/9 0:00	2.553678	2.558161	2.5155711	2.5481989	2.5481989	81620792	0	0
2004/8/31 0:00	2.54795	2.5830679	2.5444629	2.5496931	2.5496931	98825037	0	0
2004/8/24 0:00	2.7706151	2.7795811	2.579581	2.6119599	2.6119599	306396159	0	0
2004/9/10 0:00	2.530515	2.654052	2.5230429	2.6234169	2.6234169	174804764	0	0
2004/8/25 0:00	2.6142011	2.689918	2.587302	2.6401041	2.6401041	184645512	0	0
2004/8/27 0:00	2.6924081	2.7053599	2.6323831	2.6438401	2.6438401	124826132	0	0
2004/9/13 0:00	2.6557951	2.700129	2.651561	2.677464	2.677464	157628624	0	0
2004/8/26 0:00	2.6139519	2.6886721	2.606729	2.687676	2.687676	142572401	0	0
2004/8/20 0:00	2.51582	2.7168169	2.503118	2.697639	2.697639	458857488	0	0
2004/8/23 0:00	2.7584109	2.826406	2.7160699	2.724787	2.724787	366857939	0	0
2004/9/14 0:00	2.676219	2.7895441	2.65978	2.7768421	2.7768421	217608605	0	0
2004/9/15 0:00	2.753679	2.8450861	2.7447121	2.7895441	2.7895441	215279909	0	0
2004/9/16 0:00	2.798012	2.8841889	2.780827	2.8386099	2.8386099	186207345	0	0

Table 1 GOOG's Stock Price

1.2.2 Data Overview

Figure 1 is a chart of GOOG's adjusted closing price from 2004 to 2023. It has generally been rising year by year, but has shown V-shaped fluctuations in the past three years.

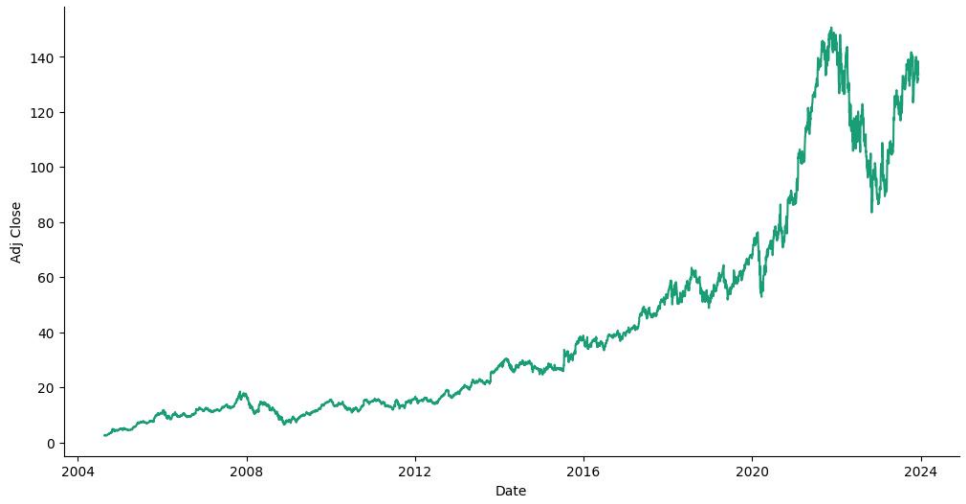


Figure 1 GOOG's Adj Close Price

Meanwhile, Figure 2 is a price chart of multiple stocks and the SPY index. We can see that in 2008, 2015, and around 2020, the SPY index had obvious fluctuations, and some stocks in the picture also had similar fluctuations. Except for GME, other stocks as a whole show an upward trend year by year, similar to the SPY index.

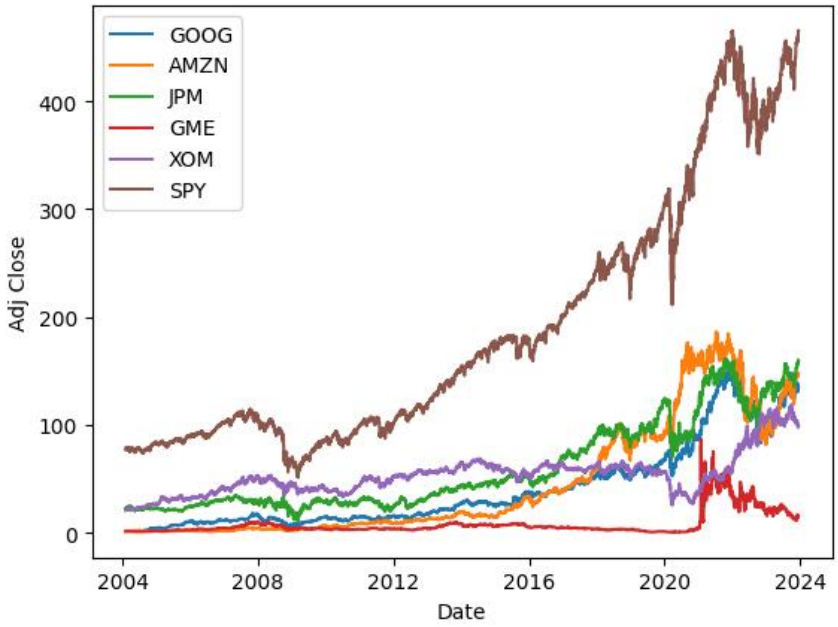


Figure 2 Adj Close Overview

1.3 Data Preprocessing

1.3.1 Data Check

I chose to use data provided by Yahoo Finance, which contains price revision information and has no missing values. Nonetheless, I also added a missing value identification module in the project code to check if the collected data contains missing values.

1.3.2 Data Preprocessing

From the information obtained from Yahoo, I selected the common Open, High, Low, Close and Volume as input data, and calculated the rate of return through the adjusted Close. Because different stocks have different listing dates, in order to ensure the completeness of the training data, I intercepted the data from 2005-01-01 to 2023-12-12. This ensures that all stock and SPY index data are complete.

Since the exchange restricts trading on legal holidays and other days, and the number of days in each month is different, the number of trading days in the two months we intercepted at different time nodes is different. The directly calculated correlation coefficient between the stock

and the SPY index in the next two months will be affected by the different degrees of freedom of the data. Therefore, I optimized the calculation of the correlation coefficient and regarded the correlation coefficient of the return rate in the next 40 trading days as the correlation coefficient in the next two months. Through experiments, it can be found that the difference between the two is less than one thousandth, and can be regarded as equal in practical applications.

The heat map of the stock price correlation coefficient in the entire time period is shown in Figure 3.

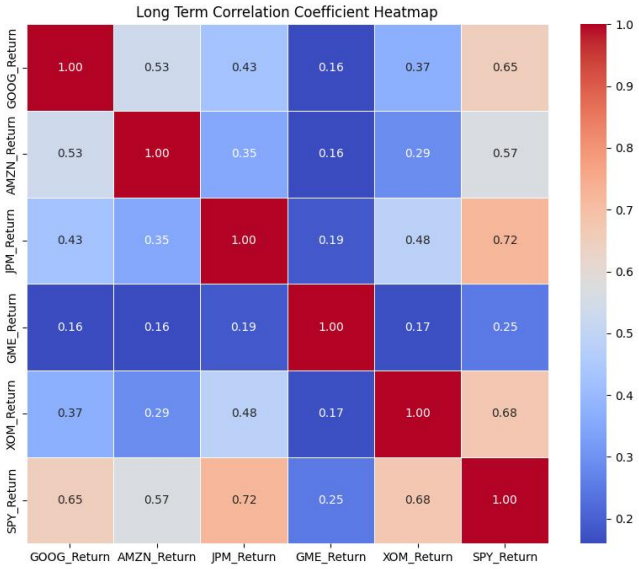


Figure 3 Stock Price Correlation Coefficient in the Entire Time Period

The heat map of the stock price correlation coefficient in 2005 is shown in Figure 4.

Figure 4 Stock Price Correlation Coefficient in 2005

The heat map of the stock price correlation coefficient in 2023 is shown in Figure 5. It can be seen from the figure that whether it is 2005 or 2023, the correlation coefficient between stocks and

SPY is positive. This is also consistent with the laws of economics. Compared with 2005, the correlation coefficients of GOOG and AMZN stocks with SPY have increased in 2023, while the correlation coefficients of JPM, GME and XOM with SPY have decreased.

Figure 5 Stock Price Correlation Coefficient in 2023

1.4 Model Selection

In this prediction project, I mainly used the LSTM model and the LightGBM model, and conducted a series of optimizations.

1.4.1 LSTM

LSTM[1], which stands for Long Short-Term Memory, is a type of recurrent neural network (RNN[2]) architecture designed to overcome the limitations of traditional RNNs in capturing and learning long-term dependencies in sequential data. RNNs are neural networks that can process sequential information by maintaining a hidden state that evolves over time as it processes each element of the sequence. However, standard RNNs often struggle with capturing long-range dependencies due to the vanishing gradient problem, where the gradients diminish as they are backpropagated through time.

LSTMs were introduced to address this issue by incorporating a memory cell and gating mechanisms to control the flow of information within the cell. The key components of an LSTM include:

- Cell State: This is the memory of the LSTM. It can store information over long sequences and allows the network to retain or forget information selectively.
- Hidden State: This is similar to the hidden state in traditional RNNs. It is a function of both the current input and the previous hidden state, but it is modified by the gating

mechanisms.

- Gating Mechanisms: Forget Gate, Determines what information from the cell state should be discarded or kept; Input Gate, Decides what new information should be stored in the cell state. Output Gate, Filters the information from the cell state to produce the output of the LSTM.

The logical structure of its cell is shown in Figure 6.

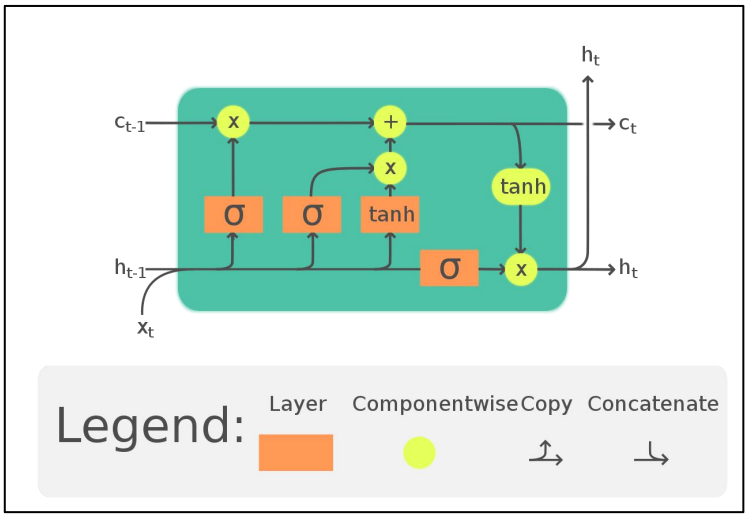


Figure 6 LSTM cell

The gating mechanisms enable LSTMs to selectively learn and remember or forget information at each time step, making them effective for processing sequential data with long-term dependencies. LSTMs have found widespread applications in natural language processing, speech recognition, time series prediction, and other tasks where sequential information is crucial.

Long Short-Term Memory (LSTM) networks and other variants of recurrent neural networks (RNNs) find applications in various fields due to their ability to model sequential data and capture long-term dependencies. They can be used to predict financial market trends, stock prices, and identifying anomalies in financial data.

1.4.2 LightGBM

LightGBM[3], which stands for Light Gradient Boosting Machine, is an open-source, distributed, high-performance gradient boosting framework developed by Microsoft. It is designed for efficient and scalable machine learning tasks, particularly in the context of decision tree-based ensemble models. LightGBM is part of the gradient boosting framework family, similar to XGBoost[4] and CatBoost[5]. Compared with other algorithms, the main features of LightGBM are as follows:

- Gradient Boosting Framework: LightGBM is based on the gradient boosting framework,

where weak learners (typically decision trees) are sequentially added to the ensemble to correct errors made by the previous models.

- Lightweight and Fast: LightGBM is optimized for speed and efficiency. It is particularly efficient for large datasets and high-dimensional data. The framework uses a histogram-based learning method that reduces the memory usage and speeds up the training process.
- Distributed Computing: LightGBM supports distributed training, allowing the utilization of multiple machines to accelerate the model training process. This makes it suitable for handling large-scale datasets.
- Leaf-wise Tree Growth: As shown in Figure 7, Unlike other tree-based models that grow trees level-wise, LightGBM grows trees in a leaf-wise fashion. This approach tends to reduce the number of nodes in the tree, which can result in faster training times.
- Categorical Feature Support: LightGBM can handle categorical features without the need for one-hot encoding, making it more convenient for dealing with datasets that include categorical variables.
- Regularization: LightGBM provides regularization techniques to prevent overfitting, including L1 and L2 regularization on leaf values.

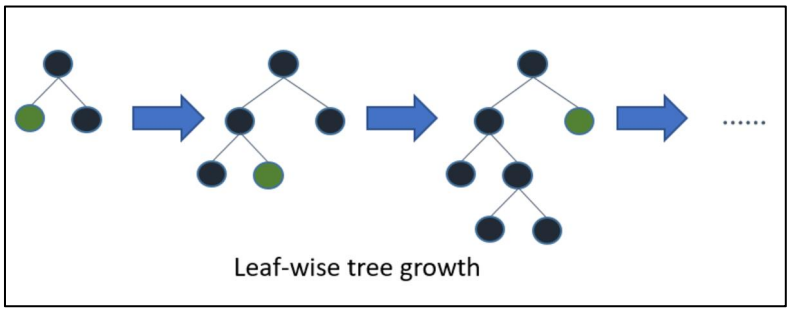


Figure 7 Leaf-wise Tree Growth

Due to its speed, efficiency, and scalability, LightGBM has gained popularity in both research and industry settings. It is commonly used in data science competitions (e.g., Kaggle) and production environments where large datasets and high-dimensional feature spaces are common challenges.

1.5 Model Training and Validation

1.5.1 LSTM solo

Previous training shows that different stocks will adjust the model in different directions. In this way, the trained model does not perform satisfactorily when predicting a single stock because it pursues universality. Therefore, I chose to build five dedicated models for each of the five stocks and train them separately to avoid contamination and interference between the stocks on the models. At the same time, in order to avoid overfitting, I set the backtracking step size to 40 and the size of the hidden layer to 64. I call this optimized method LSTM solo.

1.5.2 LSTM qmimo

Since there are multiple stock and index correlation coefficients that need to be predicted, the format of the input data needs to be discussed. There are two main options here. The first is to input the price series of five stocks to the LSTM model at the same time. The model will output five values, which are the correlation coefficients of the five stocks and the index in the next two months. This method is called multiple input and multiple output, but after preliminary experiments, I found that this method is not good. Different stock data has a negative effect on each other's predictions. The second method is to input only the price series of one stock at a time, and the model will output the predicted value of the correlation coefficient between this stock and the index. The model will traverse all stocks. I named this method pseudo multiple input multiple output, or qmimo for short.

In order to improve the training speed and the timeliness of the prediction results, I selected the price data of the past 8 years for training and testing. It is particularly important to note that the LSTM model is greatly affected by the size of the input value, so before training, the data needs to be normalized. Here I have performed normalization operations on different price sequences of each stock.

1.5.3 LSTM qmlt

After completing the training task of LSTM qmimo, I optimized the model based on the result analysis, added training data and test data, increased the backtracking step size and increased the size of the hidden layer. I call this optimized model LSTM qmlt.

1.5.4 LightGBM org

When using LightGBM for prediction, the process of preprocessing the training data set and the test data set is basically the same as the previous model-related process. It is worth noting that for the LightGBM model, the size of the input value has no impact on the model, so there is no need to normalize the data in advance.

1.6 Evaluation

1.6.1 suitable metrics

The choice of evaluation metrics depends on the specific task I am trying to solve and the nature of my machine learning model. For Classification Tasks, Accuracy((TP + TN) / (TP + TN + FP + FN)), Precision(TP / (TP + FP)), Recall(TP / (TP + FN)) will be used. And as for Regression Tasks, MAE, MSE will be used. Because I treat the prediction of correlation coefficients as a regression problem, I mainly use MSE and correlation matrices to evaluate model performance.

1.6.2 LSTM solo Evaluation

After training, it was found that this solo model was time-consuming and performed poorly. By making a distribution diagram of the true value and predicted value of the correlation coefficient between GOOG stock earnings and the SPY index change rate in the next two months, as well as a scatter diagram of the two, it can be found that the prediction effect is not very obvious. The calculated MSEs are also very large.

I also found through experiments that simply adjusting the size of the neural network cannot significantly improve the performance of the model. Due to the characteristics and forgetting mechanism of LSTM, the data at the end of the input data has a greater weight. Individual stocks will fluctuate in the short term, which will have a negative impact on the training results of a single model. Therefore, I optimized the model and trained the same model using data from five stocks. This also resulted in the new model qmimo.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
import seaborn as sns
import torch
import torch.nn as nn
from torch.autograd import Variable
from sklearn.preprocessing import MinMaxScaler
from tqdm.notebook import tqdm, trange

start_date = '2005-01-01'
end_date = '2023-12-12'
if_max = False
actions = True
stock_list = ['GOOG', 'AMZN', 'JPM', 'GME', 'XOM', 'SPY']
period = 40
train_rate = 0.8
features = ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume',
'Return']
look_back = 40
input_size = len(features)
hidden_layer_size = 64
output_size = 1
epochs = 100
learning_rate = 0.001

# load stock price data
def get_stock_price(stock, if_max=False, start='2022-01-01',
end='2023-01-01'):
    if if_max:
        return yf.download(stock, period='max', actions=True)
    else:
        return yf.download(stock, start=start, end=end, actions=True)


def normalization(stock_df):
    for column in stock_df.columns:
        stock_df[column] = (stock_df[column] -
stock_df[column].mean()) / stock_df[column].std()
    return stock_df


# check if there is any NA value
def check_na_value(stock_dict):
    for stock in stock_dict.keys():
        if len(stock_dict[stock]) != len(stock_dict[stock].dropna()):
            print(stock, 'null!')
            return False
    print('NULL value Checked!')
```

```
    return True


def check_na_value_by_df(stock_df, stock):
    if len(stock_df) != len(stock_df.dropna()):
        print(stock, 'null!')
        return False
    print('NULL value Checked!')
    return True


# get return
def get_return(stock_dict, adj=False):
    label = 'Adj Close' if adj else 'Close'
    for stock in stock_dict.keys():
        stock_dict[stock]['Return'] = stock_dict[stock]
[label].pct_change().fillna(0)
    return stock_dict


def get_return_by_df(stock_df, adj=False):
    label = 'Adj Close' if adj else 'Close'
    stock_df['Return'] = stock_df[label].pct_change().fillna(0)
    return stock_df


# get corr
def corr_cal(stock_dict, n):
    for stock in stock_dict.keys():
        stock_dict[stock]['Corr_p'] = stock_dict[stock]
['Return'].rolling(n).corr(
            stock_dict['SPY']['Return']).shift(-n + 1)
        stock_dict[stock] = stock_dict[stock].iloc[:-n + 1, :]
    return stock_dict

stock_dict = {}

# Data Preparation
for stock in stock_list:
    # data load
    df = get_stock_price(stock, if_max=if_max, start=start_date,
end=end_date)
    # df = normalization(df.loc[:, ['Open', 'High', 'Low', 'Close',
'Adj Close', 'Volume']])
    # null value check
    assert check_na_value_by_df(df, stock)
    # return calculate
    stock_dict[stock] = get_return_by_df(df, adj=actions)

stock_dict = corr_cal(stock_dict, period)
```

```
 [**********100%%**********]  1 of 1 completed
NULL value Checked!
[**********100%%**********]  1 of 1 completed
NULL value Checked!
[**********100%%**********]  1 of 1 completed
NULL value Checked!
[**********100%%**********]  1 of 1 completed
NULL value Checked!
[**********100%%**********]  1 of 1 completed
NULL value Checked!
[**********100%%**********]  1 of 1 completed
NULL value Checked!
```

```
def get_data_Xy_solo(stock_df, rate, features):
    train_size = int(len(stock_dict['SPY']) * rate)
    data_X = stock_df[features]
    data_y = stock_df[['Corr_p']]
    return data_X.iloc[:train_size, :], data_X.iloc[train_size:, :],
data_y.iloc[:train_size, :], data_y.iloc[train_size:, :]

def create_sequences(data, target, look_back):
    sequences, labels = [], []
    for i in range(len(data) - look_back):
        sequence = data[i:i + look_back]
        label = target[i + look_back]
        sequences.append(sequence)
        labels.append(label)
    return torch.stack(sequences), torch.stack(labels)

class CorrelationPredictor(nn.Module):
    def __init__(self, input_size, hidden_layer_size, output_size):
        super().__init__()
        self.hidden_layer_size = hidden_layer_size
        self.lstm = nn.LSTM(input_size, hidden_layer_size)
        self.linear = nn.Linear(hidden_layer_size, output_size)
        self.hidden_cell = (torch.zeros(6, 6, self.hidden_layer_size),
                            torch.zeros(6, 6, self.hidden_layer_size))

    def forward(self, input_seq):
        lstm_out, self.hidden_cell =
self.lstm(input_seq.view(len(input_seq), 1, -1), self.hidden_cell)
        predictions = self.linear(lstm_out.view(len(input_seq), -1))
        return predictions[-1]

def solo_data_predict(stock_df, rate):
  train_X, test_X, train_y, test_y= get_data_Xy_solo(stock_df, rate,
features)
  scaler = MinMaxScaler(feature_range=(-1, 1))
  train_X = torch.tensor(scaler.fit_transform(train_X.values))
  test_X = torch.tensor(scaler.fit_transform(test_X.values))
```

```
  train_y = torch.tensor(scaler.fit_transform(train_y.values))
  test_y = torch.tensor(scaler.fit_transform(test_y.values))

  train_X = train_X.clone().detach().to(torch.float32)
  test_X = test_X.clone().detach().to(torch.float32)
  train_y = train_y.clone().detach().to(torch.float32)
  test_y = test_y.clone().detach().to(torch.float32)

  train_sequences, train_labels = create_sequences(train_X, train_y,
look_back)
  test_sequences, test_labels = create_sequences(test_X, test_y,
look_back)

  model = CorrelationPredictor(input_size, hidden_layer_size,
output_size)
  criterion = nn.MSELoss()
  optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

  total = (len(train_labels))
  for epoch in trange(epochs, desc='Training Progress'):
    count = 0
    for seq, label in zip(train_sequences, train_labels):
        optimizer.zero_grad()
        model.hidden_cell = (torch.zeros(1, 1,
model.hidden_layer_size),
                             torch.zeros(1, 1,
model.hidden_layer_size))

        y_pred = model.forward(seq).reshape(-1, 1)

        loss = criterion(y_pred, label.reshape(-1, 1))
        loss.backward()
        optimizer.step()
        print('\rSequences Training: {:.2%} {}'.format(count/total,
loss), end='')
        count += 1

  model.eval()
  with torch.no_grad():
      test_predictions = torch.zeros_like(test_labels)
      for i, seq in enumerate(test_sequences):
          model.hidden_cell = (torch.zeros(1, 1,
model.hidden_layer_size),
                               torch.zeros(1, 1,
model.hidden_layer_size))
          test_predictions[i] = model(seq)

  predicted_correlations =
scaler.inverse_transform(test_predictions.numpy())
```

```
  actual_correlations = scaler.inverse_transform(test_labels.numpy())
  return predicted_correlations, actual_correlations

predicted_correlations = {}
actual_correlations = {}
for stock in stock_list[:-1]:
    predicted_correlations[stock], actual_correlations[stock] =
solo_data_predict(stock_dict[stock], train_rate)
```

{"model_id":"262e6122ddf440709084874c0b746dab","version_major":2,"vers
ion_minor":0}

```
Sequences Training: 99.97% 0.22082237899303436
```

{"model_id":"0d57c9af374d44b3b758af65160767da","version_major":2,"vers
ion_minor":0}

```
Sequences Training: 99.97% 0.32804417610168457
```

{"model_id":"0b36f44584874456b676db3bfb6d1f84","version_major":2,"vers
ion_minor":0}

```
Sequences Training: 99.97% 0.12734824419021606
```

{"model_id":"82f555480c9345868b030c49d3ba7148","version_major":2,"vers
ion_minor":0}

```
Sequences Training: 99.97% 0.15442687273025513
```

{"model_id":"93d2282c78af4b0cb08dcd99358b7ecb","version_major":2,"vers
ion_minor":0}

```
Sequences Training: 99.97% 0.18867170810699463

predicted_correlations_df = pd.DataFrame(columns=[i + '
Predicted_Correlation' for i in stock_list[:-1]])
actual_correlations_df = pd.DataFrame(columns=[i + '
Actual_Correlation' for i in stock_list[:-1]])
for stock in stock_list[:-1]:
    predicted_correlations_df[stock + ' Predicted_Correlation'] =
predicted_correlations[stock].reshape(-1)
    actual_correlations_df[stock + ' Actual_Correlation'] =
actual_correlations[stock].reshape(-1)

result_df = pd.concat([actual_correlations_df,
predicted_correlations_df], axis=1)
result_df
```

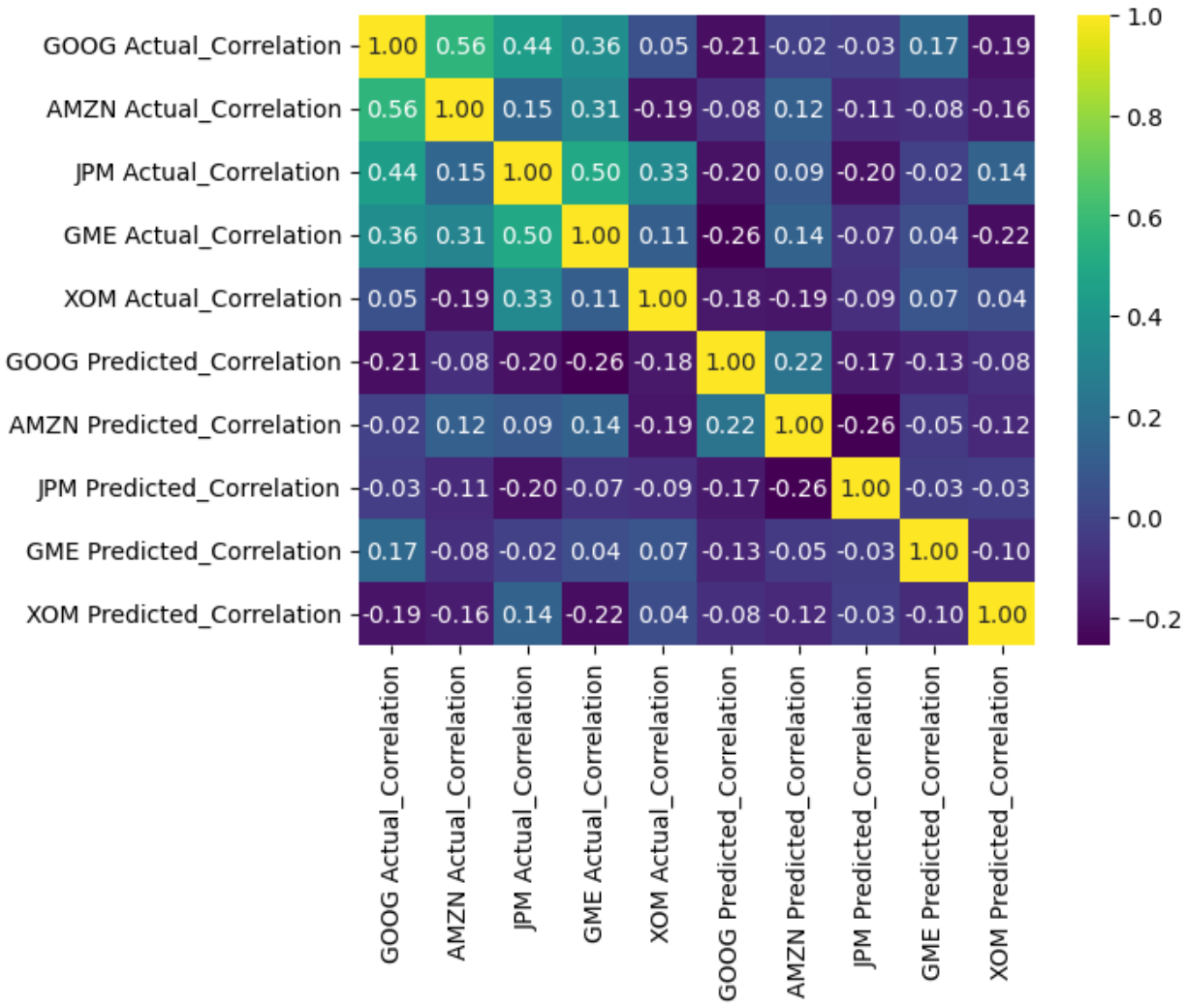
```
     GOOG Actual_Correlation  AMZN Actual_Correlation  JPM
Actual_Correlation  \
0                  0.907475                 0.661051
0.890945
```

```
1                    0.895085                    0.595130
0.896604
2                    0.879429                    0.551002
0.884497
3                    0.839719                    0.480749
0.854981
4                    0.843526                    0.429927
0.850387
..                        ...                         ...
...
901                  0.644389                    0.578024
0.589548
902                  0.647777                    0.582670
0.593460
903                  0.638021                    0.586282
0.590849
904                  0.626643                    0.593000
0.589928
905                  0.614382                    0.575725
0.634735

     GME Actual_Correlation  XOM Actual_Correlation  \
0                  0.393091                0.805181
1                  0.396172                0.773127
2                  0.346803                0.802690
3                  0.360283                0.782983
4                  0.399180                0.774618
..                      ...                     ...
901                0.145419                0.139378
902                0.139249                0.153044
903                0.168262                0.181508
904                0.143280                0.192527
905                0.135489                0.276294

     GOOG Predicted_Correlation  AMZN Predicted_Correlation  \
0                      0.969140                    0.588440
1                      0.994660                    0.807161
2                      1.087950                    0.520981
3                      0.969369                    0.471450
4                      0.854629                    0.283506
..                          ...                         ...
901                    0.876625                    0.473644
902                    0.851115                    0.462595
903                    0.889726                    0.457336
904                    0.827246                    0.469846
905                    0.821964                    0.502077

     JPM Predicted_Correlation  GME Predicted_Correlation  \
0                     0.364765                   0.018391
1                     0.741551                   0.119691
```

```
2                       0.620530                       0.254285
3                       0.808872                       0.334024
4                       0.625015                       0.200255
..                           ...                            ...
901                     0.604241                       0.124590
902                     0.617859                       0.125864
903                     0.612414                       0.159410
904                     0.617715                       0.192639
905                     0.700426                       0.162177

     XOM Predicted_Correlation
0                       0.833884
1                       0.688744
2                       0.894106
3                       0.966962
4                       0.766817
..                           ...
901                     0.831303
902                     0.803529
903                     0.830326
904                     0.716586
905                     0.832659

[906 rows x 10 columns]
```

```
sns.heatmap(result_df.corr(), cmap='viridis', annot=True, fmt=".2f")
```

```
<Axes: >
```




```
for stock in stock_list[:-1]:
  print(f"【{stock}】Mean Squared Error (MSE):", ((result_df[stock + ' Predicted_Correlation'] - result_df[stock + ' Actual_Correlation']) ** 2).mean())
```

```
【GOOG】Mean Squared Error (MSE): 0.06526254
【AMZN】Mean Squared Error (MSE): 0.044888124
【JPM】Mean Squared Error (MSE): 0.09311438
```

1.6.3 LSTM qmimo Evaluation

In this model, like solo, I set the backtracking step size to 40, the size of the LSTM hidden layer to 64, and the learning rate to 0.001. The performance of the optimized model has been significantly improved compared to the solo model. Comparatively looking at the five stocks, we can see that the model's prediction performance for GOOG is better, and its prediction performance for JPM is worse. But all stocks are predicted to perform better than solo. At the same time, the MSEs of the predicted results and the true values have dropped significantly. Among them, for GOOG stock, the MSE is the smallest, which is 0.027; for XOM stock, the MSE is the largest, which is 0.083. The overall MSE is about half smaller than the MSE of the solo model.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
import seaborn as sns
import torch
import torch.nn as nn
from torch.autograd import Variable
from sklearn.preprocessing import MinMaxScaler
from tqdm.notebook import tqdm, trange

start_date = '2015-01-01'
end_date = '2023-12-12'
if_max = False
actions = True
stock_list = ['GOOG', 'AMZN', 'JPM', 'GME', 'XOM', 'SPY']
period = 40
train_rate = 0.8
features = ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume',
'Return']
look_back = 40
input_size = len(features)
hidden_layer_size = 32
output_size = 1
epochs = 100
learning_rate = 0.001

# load stock price data
def get_stock_price(stock, if_max=False, start='2022-01-01',
end='2023-01-01'):
    if if_max:
        return yf.download(stock, period='max', actions=True)
    else:
        return yf.download(stock, start=start, end=end, actions=True)


def normalization(stock_df):
    for column in stock_df.columns:
        stock_df[column] = (stock_df[column] -
stock_df[column].mean()) / stock_df[column].std()
    return stock_df


# check if there is any NA value
def check_na_value(stock_dict):
    for stock in stock_dict.keys():
        if len(stock_dict[stock]) != len(stock_dict[stock].dropna()):
            print(stock, 'null!')
            return False
    print('NULL value Checked!')
```

```
    return True


def check_na_value_by_df(stock_df, stock):
    if len(stock_df) != len(stock_df.dropna()):
        print(stock, 'null!')
        return False
    print('NULL value Checked!')
    return True


# get return
def get_return(stock_dict, adj=False):
    label = 'Adj Close' if adj else 'Close'
    for stock in stock_dict.keys():
        stock_dict[stock]['Return'] = stock_dict[stock]
[label].pct_change().fillna(0)
    return stock_dict


def get_return_by_df(stock_df, adj=False):
    label = 'Adj Close' if adj else 'Close'
    stock_df['Return'] = stock_df[label].pct_change().fillna(0)
    return stock_df


# get corr
def corr_cal(stock_dict, n):
    for stock in stock_dict.keys():
        stock_dict[stock]['Corr_p'] = stock_dict[stock]
['Return'].rolling(n).corr(
            stock_dict['SPY']['Return']).shift(-n + 1)
        stock_dict[stock] = stock_dict[stock].iloc[:-n + 1, :]
    return stock_dict

stock_dict = {}

# Data Preparation
for stock in stock_list:
    # data load
    df = get_stock_price(stock, if_max=if_max, start=start_date,
end=end_date)
    # df = normalization(df.loc[:, ['Open', 'High', 'Low', 'Close',
'Adj Close', 'Volume']])
    # null value check
    assert check_na_value_by_df(df, stock)
    # return calculate
    stock_dict[stock] = get_return_by_df(df, adj=actions)

stock_dict = corr_cal(stock_dict, period)
```

```
[**********************100%%**********************]  1 of 1 completed
NULL value Checked!
[**********************100%%**********************]  1 of 1 completed
NULL value Checked!
[**********************100%%**********************]  1 of 1 completed
NULL value Checked!
[**********************100%%**********************]  1 of 1 completed
NULL value Checked!
[**********************100%%**********************]  1 of 1 completed
NULL value Checked!
[**********************100%%**********************]  1 of 1 completed
NULL value Checked!
```

```
def get_data_Xy(stock_dict, rate, features):
    train_size = int(len(stock_dict['SPY']) * rate)
    data_X = pd.concat(
        [stock_dict[i][features] for i in stock_dict.keys() if i != 
'SPY'],
        axis=1)
    data_y = pd.concat([stock_dict[i]['Corr_p'] for i in 
stock_dict.keys() if i != 'SPY'], axis=1)
    return data_X.iloc[:train_size, :], data_X.iloc[train_size:, :], 
data_y.iloc[:train_size, :], data_y.iloc[train_size:, :]

train_X, test_X, train_y, test_y = get_data_Xy(stock_dict, train_rate, 
features)

scaler = MinMaxScaler(feature_range=(-1, 1))

train_X = torch.tensor(scaler.fit_transform(train_X.values))
test_X = torch.tensor(scaler.fit_transform(test_X.values))
train_y = torch.tensor(scaler.fit_transform(train_y.values))
test_y = torch.tensor(scaler.fit_transform(test_y.values))

train_X = train_X.clone().detach().to(torch.float32)
test_X = test_X.clone().detach().to(torch.float32)
train_y = train_y.clone().detach().to(torch.float32)
test_y = test_y.clone().detach().to(torch.float32)
print(train_X.dtype)
```

```
torch.float32
```

```
test_X.shape,test_y.shape,train_X.shape,train_y.shape
```

```
(torch.Size([443, 35]),
 torch.Size([443, 5]),
 torch.Size([1769, 35]),
 torch.Size([1769, 5]))
```

```
train_X, train_y
```

```
(tensor([[-0.9729, -0.9723, -0.9710,  ...,  0.7629, -0.8419, -0.0185],
         [-0.9774, -0.9778, -0.9797,  ...,  0.6802, -0.6364, -0.2382],
         [-0.9839, -0.9842, -0.9893,  ...,  0.6645, -0.6819, -0.0612],
         ...,
         [ 0.7862,  0.8086,  0.7910,  ...,  0.8336, -0.3345,  0.1703],
         [ 0.7927,  0.7861,  0.7755,  ...,  0.8590, -0.5003,  0.0473],
         [ 0.7483,  0.7922,  0.7333,  ...,  0.8404, -0.5307, -
0.0663]]),
 tensor([[-0.7082, -0.5914,  0.7372,  0.2986,  0.6393],
         [-0.7171, -0.5842,  0.7233,  0.2996,  0.6403],
         [-0.8781, -0.6705,  0.6690,  0.3628,  0.5783],
         ...,
         [ 0.4986,  0.4441, -0.0155,  0.5822, -0.7238],
         [ 0.5561,  0.4992,  0.0801,  0.6677, -0.9256],
         [ 0.5397,  0.4999,  0.0926,  0.6733, -0.9263]]))
```

```
test_X, test_y
```

```
(tensor([[ 0.5876,  0.6409,  0.7184,  ..., -0.9749, -0.2055,  0.6915],
         [ 0.6893,  0.7167,  0.8568,  ..., -0.9822, -0.5101,  0.0619],
         [ 0.6979,  0.7280,  0.7861,  ..., -1.0000, -0.5717,  0.0028],
         ...,
         [ 0.7075,  0.7034,  0.8033,  ...,  0.4906, -0.2595,  0.1005],
         [ 0.6620,  0.6717,  0.7557,  ...,  0.6167, -0.2874,  0.5498],
         [ 0.6343,  0.6584,  0.7991,  ...,  0.6197, -0.6515,
0.1133]]),
 tensor([[ 0.5426,  0.1879, -0.1309,  0.4059, -0.5990],
         [ 0.5401,  0.0791, -0.1328,  0.4156, -0.6878],
         [ 0.5451,  0.0777, -0.1184,  0.4461, -0.6613],
         ...,
         [-0.0732, -0.1312, -0.1103, -0.2895,  0.0123],
         [-0.1131, -0.1103, -0.1137, -0.3477,  0.0339],
         [-0.1560, -0.1639,  0.0537, -0.3659,  0.1979]]))
```

```
def create_sequences_mimo(data, target, look_back, features):
    sequences, labels = [], []
    fn = int(np.shape(data)[1] / len(features))
    fl = len(features)
    for k in range(fn):
        for i in range(len(data) - look_back):
            sequence = data[i:i + look_back, (k * fl): (k * fl + fl)]
            label = target[i + look_back, k]
            sequences.append(sequence)
            labels.append(label)
    return torch.stack(sequences), torch.stack(labels)

train_sequences, train_labels = create_sequences_mimo(train_X,
train_y, look_back, features)
test_sequences, test_labels = create_sequences_mimo(test_X, test_y,
look_back, features)
```

```
train_sequences.shape, train_labels.shape, test_sequences.shape,
test_labels.shape
```

```
(torch.Size([8645, 40, 7]),
 torch.Size([8645]),
 torch.Size([2015, 40, 7]),
 torch.Size([2015]))
```

```
class CorrelationPredictor(nn.Module):
    def __init__(self, input_size, hidden_layer_size, output_size):
        super().__init__()
        self.hidden_layer_size = hidden_layer_size
        self.lstm = nn.LSTM(input_size, hidden_layer_size)
        self.linear = nn.Linear(hidden_layer_size, output_size)
        self.hidden_cell = (torch.zeros(1, 1, self.hidden_layer_size),
                            torch.zeros(1, 1, self.hidden_layer_size))

    def forward(self, input_seq):
        lstm_out, self.hidden_cell =
self.lstm(input_seq.view(len(input_seq), 1, -1), self.hidden_cell)
        predictions = self.linear(lstm_out.view(len(input_seq), -1))
        return predictions[-1]

model = CorrelationPredictor(input_size, hidden_layer_size,
output_size)

criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

total = (len(train_labels))
for epoch in trange(100, desc='Training Progress'):
  count = 0
  for seq, label in zip(train_sequences, train_labels):
      optimizer.zero_grad()
      model.hidden_cell = (torch.zeros(1, 1, model.hidden_layer_size),
                           torch.zeros(1, 1, model.hidden_layer_size))

      y_pred = model.forward(seq).reshape(-1, 1)

      loss = criterion(y_pred, label.reshape(-1, 1))
      loss.backward()
      optimizer.step()
      print('\rSequences Training: {:.2%} {}'.format(count/total,
loss), end='')
      count += 1
```

{"model_id":"a5e3371316aa4f64a344a2d640cc090f","version_major":2,"version_minor":0}

```
Sequences Training: 99.99% 0.957276463508606
```

```
model.eval()
with torch.no_grad():
    test_predictions = torch.zeros_like(test_labels)
    for i, seq in enumerate(test_sequences):
        model.hidden_cell = (torch.zeros(1, 1,
model.hidden_layer_size),
                             torch.zeros(1, 1,
model.hidden_layer_size))
        test_predictions[i] = model(seq)

predicted_correlations =
scaler.inverse_transform(test_predictions.numpy().reshape(5, -1).T)
actual_correlations =
scaler.inverse_transform(test_labels.numpy().reshape(5, -1).T)
predicted_correlations_df = pd.DataFrame(predicted_correlations,
columns=[i + ' Predicted_Correlation' for i in stock_list[:-1]])
actual_correlations_df = pd.DataFrame(actual_correlations, columns=[i
+ ' Actual_Correlation' for i in stock_list[:-1]])

result_df = pd.concat([actual_correlations_df,
predicted_correlations_df], axis=1)
result_df
```

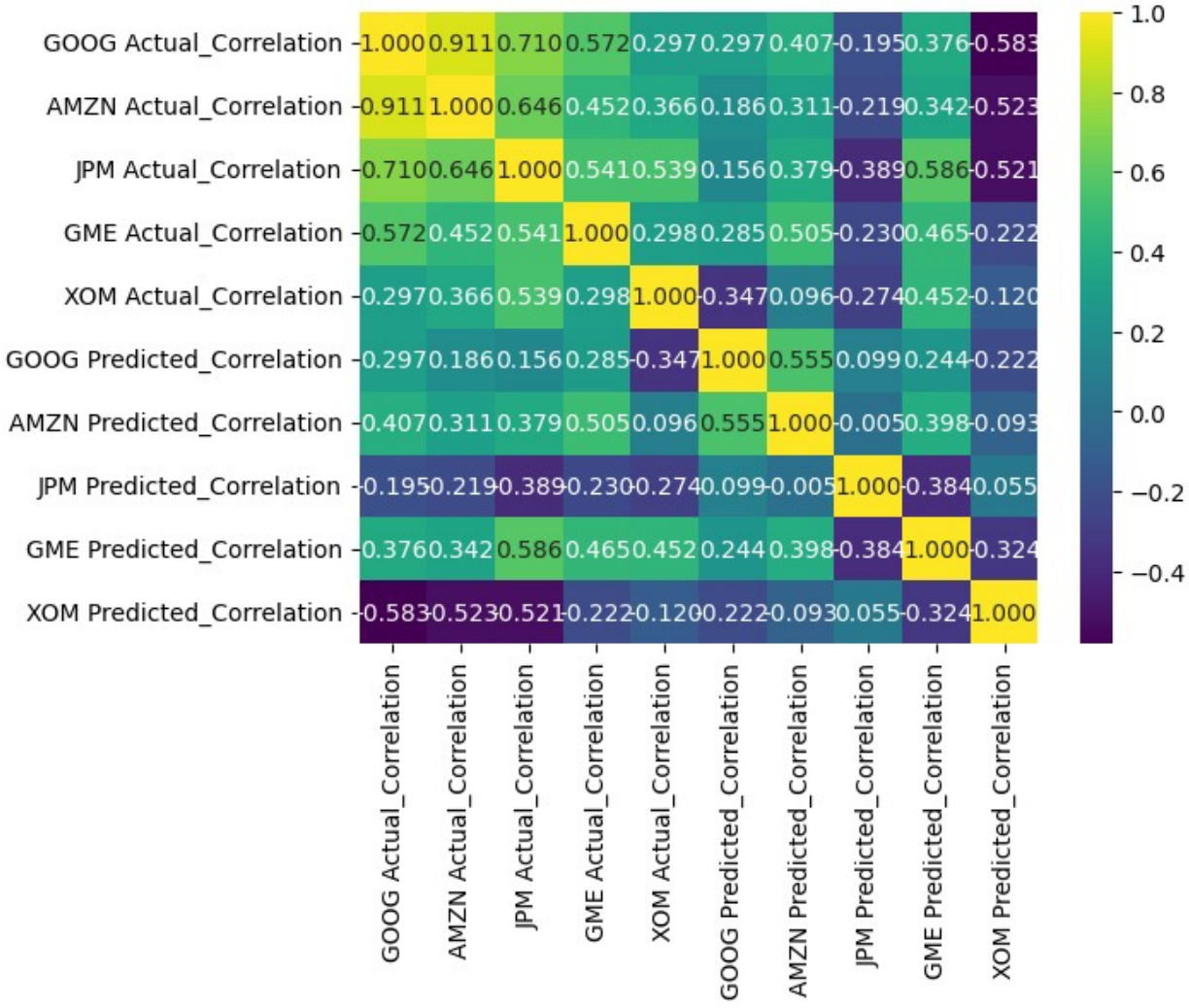
```
     GOOG Actual_Correlation  AMZN Actual_Correlation  JPM
Actual_Correlation  \
0                    0.893765                 0.823192
0.756055
1                    0.893778                 0.865899
0.755206
2                    0.883187                 0.875139
0.747871
3                    0.885538                 0.874900
0.737451
4                    0.875662                 0.870682
0.725114
..                        ...                      ...
...
398                  0.644389                 0.578024
0.589548
399                  0.647777                 0.582670
0.593460
400                  0.638021                 0.586282
0.590849
401                  0.626643                 0.593000
0.589928
402                  0.614382                 0.575725
0.634735

     GME Actual_Correlation  XOM Actual_Correlation  \
0                  0.385137                0.250623
```

```
1                   0.386036                   0.257910
2                   0.425037                   0.361420
3                   0.420744                   0.358414
4                   0.429081                   0.466739
..                       ...                        ...
398                 0.145419                   0.139378
399                 0.139249                   0.153044
400                 0.168262                   0.181508
401                 0.143280                   0.192527
402                 0.135489                   0.276294

     GOOG Predicted_Correlation  AMZN Predicted_Correlation  \
0                      0.737090                    0.810265
1                      0.721149                    0.747873
2                      0.741750                    0.727097
3                      0.789877                    0.736844
4                      0.767553                    0.750998
..                          ...                         ...
398                    0.757723                    0.826581
399                    0.778459                    0.821729
400                    0.764173                    0.806203
401                    0.785112                    0.792673
402                    0.789275                    0.779762

     JPM Predicted_Correlation  GME Predicted_Correlation  \
0                     0.879348                   0.466532
1                     0.881553                   0.468525
2                     0.819104                   0.489341
3                     0.769553                   0.488887
4                     0.828089                   0.505183
..                         ...                        ...
398                   0.750210                   0.292718
399                   0.745441                   0.294204
400                   0.753333                   0.295133
401                   0.776470                   0.284898
402                   0.691821                   0.290662

     XOM Predicted_Correlation
0                    -0.094469
1                     0.000796
2                    -0.129518
3                    -0.091422
4                    -0.041353
..                         ...
398                   0.311978
399                   0.357715
400                   0.425989
401                   0.407804
402                   0.265350
```

```
[403 rows x 10 columns]
```

```
sns.heatmap(result_df.corr(), cmap='viridis', annot=True, fmt=".3f")
```

```
<Axes: >
```




```
for stock in stock_list[:-1]:
  print(f"【{stock}】Mean Squared Error (MSE):", ((result_df[stock + ' Predicted_Correlation'] - result_df[stock + ' Actual_Correlation']) ** 2).mean())
```

```
【GOOG】Mean Squared Error (MSE): 0.027005898
【AMZN】Mean Squared Error (MSE): 0.028874982
【JPM】Mean Squared Error (MSE): 0.031840023
【GME】Mean Squared Error (MSE): 0.030831432
【XOM】Mean Squared Error (MSE): 0.08316811
```

1.6.4 LSTM qmlt Evaluation

In this model, I set the backtracking step size to 80, the size of the LSTM hidden layer to 128, and the learning rate to 0.001. Compared with qmimo, the data and model of this model are more complex, and the time spent in the training process is approximately three times that of qmimo. In this model, the forecast performance of the five stocks is still significantly improved compared to solo, and it is also improved to a certain extent compared to the qmimo model. Among the five stocks, GOOG's forecast performance is still better, XOM's forecast performance has also improved, and GME's forecast performance is the worst. For MSEs, the improvement effect of this model is not very obvious compared with the qmimo model.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
import seaborn as sns
import torch
import torch.nn as nn
from torch.autograd import Variable
from sklearn.preprocessing import MinMaxScaler
from tqdm.notebook import tqdm, trange

start_date = '2005-01-01'
end_date = '2023-12-12'
if_max = False
actions = True
stock_list = ['GOOG', 'AMZN', 'JPM', 'GME', 'XOM', 'SPY']
period = 40
train_rate = 0.8
features = ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume',
'Return']
look_back = 80
input_size = len(features)
hidden_layer_size = 128
output_size = 1
epochs = 100
learning_rate = 0.001

# load stock price data
def get_stock_price(stock, if_max=False, start='2022-01-01',
end='2023-01-01'):
    if if_max:
        return yf.download(stock, period='max', actions=True)
    else:
        return yf.download(stock, start=start, end=end, actions=True)


def normalization(stock_df):
    for column in stock_df.columns:
        stock_df[column] = (stock_df[column] -
stock_df[column].mean()) / stock_df[column].std()
    return stock_df


# check if there is any NA value
def check_na_value(stock_dict):
    for stock in stock_dict.keys():
        if len(stock_dict[stock]) != len(stock_dict[stock].dropna()):
            print(stock, 'null!')
            return False
    print('NULL value Checked!')
```

```
    return True


def check_na_value_by_df(stock_df, stock):
    if len(stock_df) != len(stock_df.dropna()):
        print(stock, 'null!')
        return False
    print('NULL value Checked!')
    return True


# get return
def get_return(stock_dict, adj=False):
    label = 'Adj Close' if adj else 'Close'
    for stock in stock_dict.keys():
        stock_dict[stock]['Return'] = stock_dict[stock]
[label].pct_change().fillna(0)
    return stock_dict


def get_return_by_df(stock_df, adj=False):
    label = 'Adj Close' if adj else 'Close'
    stock_df['Return'] = stock_df[label].pct_change().fillna(0)
    return stock_df


# get corr
def corr_cal(stock_dict, n):
    for stock in stock_dict.keys():
        stock_dict[stock]['Corr_p'] = stock_dict[stock]
['Return'].rolling(n).corr(
            stock_dict['SPY']['Return']).shift(-n + 1)
        stock_dict[stock] = stock_dict[stock].iloc[:-n + 1, :]
    return stock_dict

stock_dict = {}

# Data Preparation
for stock in stock_list:
    # data load
    df = get_stock_price(stock, if_max=if_max, start=start_date,
end=end_date)
    # df = normalization(df.loc[:, ['Open', 'High', 'Low', 'Close',
'Adj Close', 'Volume']])
    # null value check
    assert check_na_value_by_df(df, stock)
    # return calculate
    stock_dict[stock] = get_return_by_df(df, adj=actions)

stock_dict = corr_cal(stock_dict, period)
```

```
[**********100%%**********]  1 of 1 completed
NULL value Checked!
[**********100%%**********]  1 of 1 completed
NULL value Checked!
[**********100%%**********]  1 of 1 completed
NULL value Checked!
[**********100%%**********]  1 of 1 completed
NULL value Checked!
[**********100%%**********]  1 of 1 completed
NULL value Checked!
```

```
def get_data_Xy(stock_dict, rate, features):
    train_size = int(len(stock_dict['SPY']) * rate)
    data_X = pd.concat(
        [stock_dict[i][features] for i in stock_dict.keys() if i != 
'SPY'],
        axis=1)
    data_y = pd.concat([stock_dict[i]['Corr_p'] for i in 
stock_dict.keys() if i != 'SPY'], axis=1)
    return data_X.iloc[:train_size, :], data_X.iloc[train_size:, :], 
data_y.iloc[:train_size, :], data_y.iloc[train_size:, :]

train_X, test_X, train_y, test_y = get_data_Xy(stock_dict, train_rate, 
features)

scaler = MinMaxScaler(feature_range=(-1, 1))

train_X = torch.tensor(scaler.fit_transform(train_X.values))
test_X = torch.tensor(scaler.fit_transform(test_X.values))
train_y = torch.tensor(scaler.fit_transform(train_y.values))
test_y = torch.tensor(scaler.fit_transform(test_y.values))

train_X = train_X.clone().detach().to(torch.float32)
test_X = test_X.clone().detach().to(torch.float32)
train_y = train_y.clone().detach().to(torch.float32)
test_y = test_y.clone().detach().to(torch.float32)
print(train_X.dtype)
```

```
torch.float32
```

```
test_X.shape,test_y.shape,train_X.shape,train_y.shape
```

```
(torch.Size([946, 35]),
 torch.Size([946, 5]),
 torch.Size([3783, 35]),
 torch.Size([3783, 5]))
```

```
train_X, train_y
```

```
(tensor([[-0.9837, -0.9814, -0.9830,  ..., -0.9850, -0.7870, -0.1040],
         [-0.9807, -0.9819, -0.9845,  ..., -0.9935, -0.7746, -0.1476],
         [-0.9866, -0.9864, -0.9854,  ..., -1.0000, -0.8539, -0.1375],
         ...,
         [ 0.9769,  0.9805,  0.9765,  ...,  0.4242, -0.8080, -0.0548],
         [ 0.9873,  0.9917,  0.9886,  ...,  0.4005, -0.8573, -0.1610],
         [ 1.0000,  1.0000,  1.0000,  ...,  0.4257, -0.8931, -
0.0427]]),
 tensor([[-0.1729, -0.2615,  0.5850,  0.2465,  0.3497],
         [-0.1724, -0.2634,  0.5859,  0.2460,  0.3491],
         [-0.2712, -0.3931,  0.5748,  0.2464,  0.3252],
         ...,
         [ 0.8206,  0.4073,  0.7587,  0.1031,  0.7496],
         [ 0.8655,  0.5300,  0.8722,  0.1774,  0.8648],
         [ 0.9017,  0.6140,  0.9103,  0.3769,  0.8695]]))
```

```
test_X, test_y
```

```
(tensor([[-0.6138, -0.6240, -0.5819,  ..., -0.3507, -0.7639, -0.0876],
         [-0.6227, -0.6244, -0.5800,  ..., -0.3526, -0.8899, -0.0313],
         [-0.6053, -0.6137, -0.5692,  ..., -0.3573, -0.8395, -0.0499],
         ...,
         [ 0.8040,  0.8027,  0.7896,  ...,  0.7081, -0.3779, -0.0201],
         [ 0.7736,  0.7816,  0.7598,  ...,  0.7804, -0.4013,  0.2378],
         [ 0.7550,  0.7728,  0.7870,  ...,  0.7821, -0.7072, -
0.0127]]),
 tensor([[ 0.8045,  0.5661,  0.9521,  0.6930,  0.9514],
         [ 0.8713,  0.6925,  0.9439,  0.7579,  1.0000],
         [ 0.9341,  0.7513,  0.9517,  0.9127,  0.9469],
         ...,
         [-0.0898,  0.2032,  0.2365,  0.1920, -0.1629],
         [-0.1289,  0.2179,  0.2345,  0.1556, -0.1450],
         [-0.1711,  0.1801,  0.3320,  0.1442, -0.0094]]))
```

```
def create_sequences_mimo(data, target, look_back, features):
    sequences, labels = [], []
    fn = int(np.shape(data)[1] / len(features))
    fl = len(features)
    for k in range(fn):
        for i in range(len(data) - look_back):
            sequence = data[i:i + look_back, (k * fl): (k * fl + fl)]
            label = target[i + look_back, k]
            sequences.append(sequence)
            labels.append(label)
    return torch.stack(sequences), torch.stack(labels)

train_sequences, train_labels = create_sequences_mimo(train_X,
train_y, look_back, features)
test_sequences, test_labels = create_sequences_mimo(test_X, test_y,
look_back, features)
```

```
train_sequences.shape, train_labels.shape, test_sequences.shape,
test_labels.shape
```

```
(torch.Size([18515, 80, 7]),
 torch.Size([18515]),
 torch.Size([4330, 80, 7]),
 torch.Size([4330]))
```

```
class CorrelationPredictor(nn.Module):
    def __init__(self, input_size, hidden_layer_size, output_size):
        super().__init__()
        self.hidden_layer_size = hidden_layer_size
        self.lstm = nn.LSTM(input_size, hidden_layer_size)
        self.linear = nn.Linear(hidden_layer_size, output_size)
        self.hidden_cell = (torch.zeros(6, 6, self.hidden_layer_size),
                            torch.zeros(6, 6, self.hidden_layer_size))

    def forward(self, input_seq):
        lstm_out, self.hidden_cell =
self.lstm(input_seq.view(len(input_seq), 1, -1), self.hidden_cell)
        predictions = self.linear(lstm_out.view(len(input_seq), -1))
        return predictions[-1]

model = CorrelationPredictor(input_size, hidden_layer_size,
output_size)

criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

total = (len(train_labels))
for epoch in trange(100, desc='Training Progress'):
  count = 0
  for seq, label in zip(train_sequences, train_labels):
      optimizer.zero_grad()
      model.hidden_cell = (torch.zeros(1, 1, model.hidden_layer_size),
                           torch.zeros(1, 1, model.hidden_layer_size))

      y_pred = model.forward(seq).reshape(-1, 1)

      loss = criterion(y_pred, label.reshape(-1, 1))
      loss.backward()
      optimizer.step()
      print('\rSequences Training: {:.2%} {}'.format(count/total,
loss), end='')
      count += 1
```

{"model_id":"c5364dcaad834f25ba61ec3e8d15baf7","version_major":2,"version_minor":0}

```
Sequences Training: 99.99% 0.3633921444416046
```

```
model.eval()
with torch.no_grad():
    test_predictions = torch.zeros_like(test_labels)
    for i, seq in enumerate(test_sequences):
        model.hidden_cell = (torch.zeros(1, 1,
model.hidden_layer_size),
                             torch.zeros(1, 1,
model.hidden_layer_size))
        test_predictions[i] = model(seq)

predicted_correlations =
scaler.inverse_transform(test_predictions.numpy().reshape(5, -1).T)
actual_correlations =
scaler.inverse_transform(test_labels.numpy().reshape(5, -1).T)
predicted_correlations_df = pd.DataFrame(predicted_correlations,
columns=[i + ' Predicted_Correlation' for i in stock_list[:-1]])
actual_correlations_df = pd.DataFrame(actual_correlations, columns=[i
+ ' Actual_Correlation' for i in stock_list[:-1]])

result_df = pd.concat([actual_correlations_df,
predicted_correlations_df], axis=1)
result_df
```

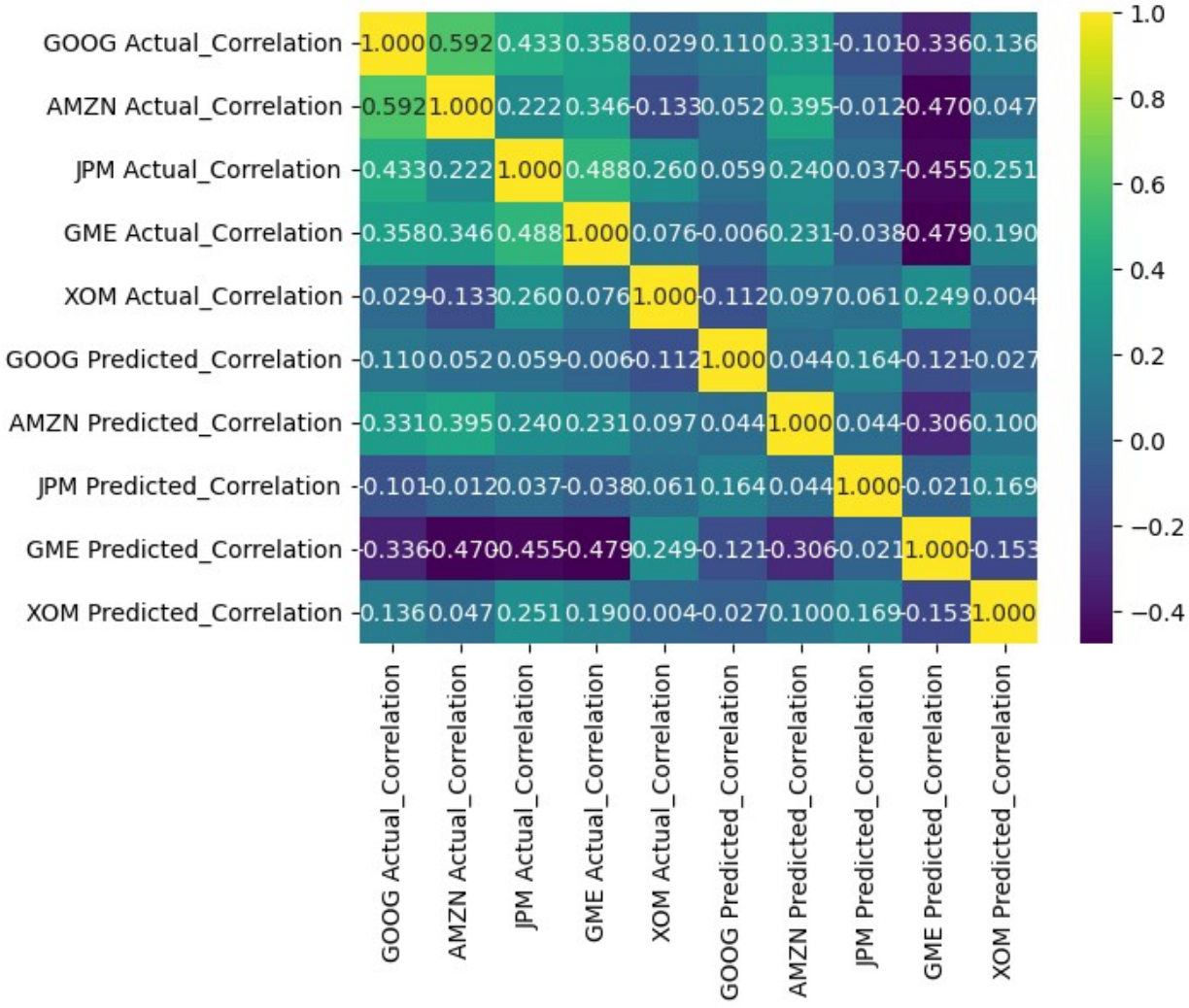
```
     GOOG Actual_Correlation  AMZN Actual_Correlation  JPM
Actual_Correlation  \
0                    0.800326                 0.513397
0.832220
1                    0.800397                 0.531636
0.835464
2                    0.805748                 0.533884
0.842397
3                    0.784627                 0.470211
0.838792
4                    0.773606                 0.472285
0.823098
..                        ...                      ...
...
861                  0.644389                 0.578024
0.589548
862                  0.647777                 0.582670
0.593460
863                  0.638021                 0.586282
0.590849
864                  0.626643                 0.593000
0.589928
865                  0.614382                 0.575725
0.634735

     GME Actual_Correlation  XOM Actual_Correlation  \
0                  0.547392                0.879995
```

```
1                    0.549235                    0.880291
2                    0.552695                    0.878561
3                    0.543046                    0.882672
4                    0.525495                    0.875698
..                        ...                         ...
861                  0.145419                    0.139378
862                  0.139249                    0.153044
863                  0.168262                    0.181508
864                  0.143280                    0.192527
865                  0.135489                    0.276294

     GOOG Predicted_Correlation  AMZN Predicted_Correlation  \
0                      0.748768                    0.766821
1                      0.719357                    0.755483
2                      0.726214                    0.749209
3                      0.738177                    0.719503
4                      0.739333                    0.707378
..                          ...                         ...
861                    0.714407                    0.625163
862                    0.732749                    0.648235
863                    0.697831                    0.633136
864                    0.738582                    0.628946
865                    0.724949                    0.628255

     JPM Predicted_Correlation  GME Predicted_Correlation  \
0                     0.802169                   0.171280
1                     0.793899                   0.181608
2                     0.793242                   0.176225
3                     0.878768                   0.155857
4                     0.819732                   0.167415
..                         ...                        ...
861                   0.627003                  -0.002007
862                   0.614847                   0.013579
863                   0.615112                   0.043584
864                   0.609260                   0.045502
865                   0.590920                   0.043629

     XOM Predicted_Correlation
0                     0.706374
1                     0.798802
2                     0.639722
3                     0.651889
4                     0.675731
..                         ...
861                   0.853191
862                   0.426919
863                   0.699564
864                   0.185815
865                   0.480109
```

```
[866 rows x 10 columns]
```

```
sns.heatmap(result_df.corr(), cmap='viridis', annot=True, fmt=".3f")
```

```
<Axes: >
```




```
for stock in stock_list[:-1]:
  print(f"【{stock}】Mean Squared Error (MSE):", ((result_df[stock + ' Predicted_Correlation'] - result_df[stock + ' Actual_Correlation']) ** 2).mean())
```

```
【GOOG】Mean Squared Error (MSE): 0.022463664
【AMZN】Mean Squared Error (MSE): 0.025668599
【JPM】Mean Squared Error (MSE): 0.039018802
【GME】Mean Squared Error (MSE): 0.1673923
【XOM】Mean Squared Error (MSE): 0.075093
```

1.6.4 LightGBM org Evaluation

In LightGBM, I set the number of leaf nodes to 31 and the learning rate to 0.05. As a tree model, LightGBM greatly improves training efficiency, and its performance is also much better than the LSTM series of models. It can be seen from the correlation matrix that the correlation degree between the predicted sequences of the five stocks and the corresponding true value sequences is the highest, and it is very significant than the others. At the same time, the MSEs of the predicted and true values of the five stocks are also very significantly better than the LSTM series model. The MSEs of all five stocks remained below 0.025.

```
import lightgbm as lgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
import seaborn as sns
from tqdm.notebook import tqdm, trange

start_date = '2005-01-01'
end_date = '2023-12-12'
if_max = False
actions = True
stock_list = ['GOOG', 'AMZN', 'JPM', 'GME', 'XOM', 'SPY']
period = 40
train_rate = 0.8
features = ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume',
'Return']
num_round = 100
params = {'objective': 'regression',
          'metric': 'mse',
          'num_leaves': 31,
          'learning_rate': 0.05,
          'feature_fraction': 0.9}
early_stopping_rounds = 10

# load stock price data
def get_stock_price(stock, if_max=False, start='2022-01-01',
end='2023-01-01'):
    if if_max:
        return yf.download(stock, period='max', actions=True)
    else:
        return yf.download(stock, start=start, end=end, actions=True)


def normalization(stock_df):
    for column in stock_df.columns:
        stock_df[column] = (stock_df[column] -
stock_df[column].mean()) / stock_df[column].std()
    return stock_df


# check if there is any NA value
def check_na_value(stock_dict):
    for stock in stock_dict.keys():
        if len(stock_dict[stock]) != len(stock_dict[stock].dropna()):
            print(stock, 'null!')
            return False
    print('NULL value Checked!')
```

```
    return True


def check_na_value_by_df(stock_df, stock):
    if len(stock_df) != len(stock_df.dropna()):
        print(stock, 'null!')
        return False
    print('NULL value Checked!')
    return True


# get return
def get_return(stock_dict, adj=False):
    label = 'Adj Close' if adj else 'Close'
    for stock in stock_dict.keys():
        stock_dict[stock]['Return'] = stock_dict[stock]
[label].pct_change().fillna(0)
    return stock_dict


def get_return_by_df(stock_df, adj=False):
    label = 'Adj Close' if adj else 'Close'
    stock_df['Return'] = stock_df[label].pct_change().fillna(0)
    return stock_df


# get corr
def corr_cal(stock_dict, n):
    for stock in stock_dict.keys():
        stock_dict[stock]['Corr_p'] = stock_dict[stock]
['Return'].rolling(n).corr(
            stock_dict['SPY']['Return']).shift(-n + 1)
        stock_dict[stock] = stock_dict[stock].iloc[:-n + 1, :]
    return stock_dict

stock_dict = {}

# Data Preparation
for stock in stock_list:
    # data load
    df = get_stock_price(stock, if_max=if_max, start=start_date,
end=end_date)
    # null value check
    assert check_na_value_by_df(df, stock)
    # return calculate
    stock_dict[stock] = get_return_by_df(df, adj=actions)

stock_dict = corr_cal(stock_dict, period)
```

```
 [**********100%%**********]  1 of 1 completed
NULL value Checked!
```

```
[**********************100%%**********************]  1 of 1 completed
NULL value Checked!
[**********************100%%**********************]  1 of 1 completed
NULL value Checked!
[**********************100%%**********************]  1 of 1 completed
NULL value Checked!
[**********************100%%**********************]  1 of 1 completed
NULL value Checked!
[**********************100%%**********************]  1 of 1 completed
NULL value Checked!
```

```
def solo_data_predict(stock_df, rate, features, esr):
  X_train, X_test, y_train, y_test =
train_test_split(stock_df[features], stock_df['Corr_p'], test_size=1-
rate, random_state=42)

  train_data = lgb.Dataset(X_train, label=y_train)
  test_data = lgb.Dataset(X_test, label=y_test, reference=train_data)

  model = lgb.train(params, train_data, num_round,
valid_sets=[train_data, test_data])

  y_pred = model.predict(X_test, num_iteration=model.best_iteration)

  return y_pred, y_test

predicted_correlations = {}
actual_correlations = {}
for stock in stock_list[:-1]:
    predicted_correlations[stock], actual_correlations[stock] =
solo_data_predict(stock_dict[stock], train_rate, features,
early_stopping_rounds)
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.000448 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1785
[LightGBM] [Info] Number of data points in the train set: 3783, number
of used features: 7
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Start training from score 0.625666
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.000177 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1785
```

```
[LightGBM] [Info] Number of data points in the train set: 3783, number
of used features: 7
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Start training from score 0.563858
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.000210 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1785
[LightGBM] [Info] Number of data points in the train set: 3783, number
of used features: 7
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Start training from score 0.703161
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.000162 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1785
[LightGBM] [Info] Number of data points in the train set: 3783, number
of used features: 7
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Start training from score 0.366143
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.000169 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1785
[LightGBM] [Info] Number of data points in the train set: 3783, number
of used features: 7
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Start training from score 0.612682
```

```
predicted_correlations_df = pd.DataFrame(columns=[i + '
Predicted_Correlation' for i in stock_list[:-1]])
actual_correlations_df = pd.DataFrame(columns=[i + '
Actual_Correlation' for i in stock_list[:-1]])
for stock in stock_list[:-1]:
    predicted_correlations_df[stock + ' Predicted_Correlation'] =
predicted_correlations[stock]
    actual_correlations_df[stock + ' Actual_Correlation'] =
actual_correlations[stock].values
```

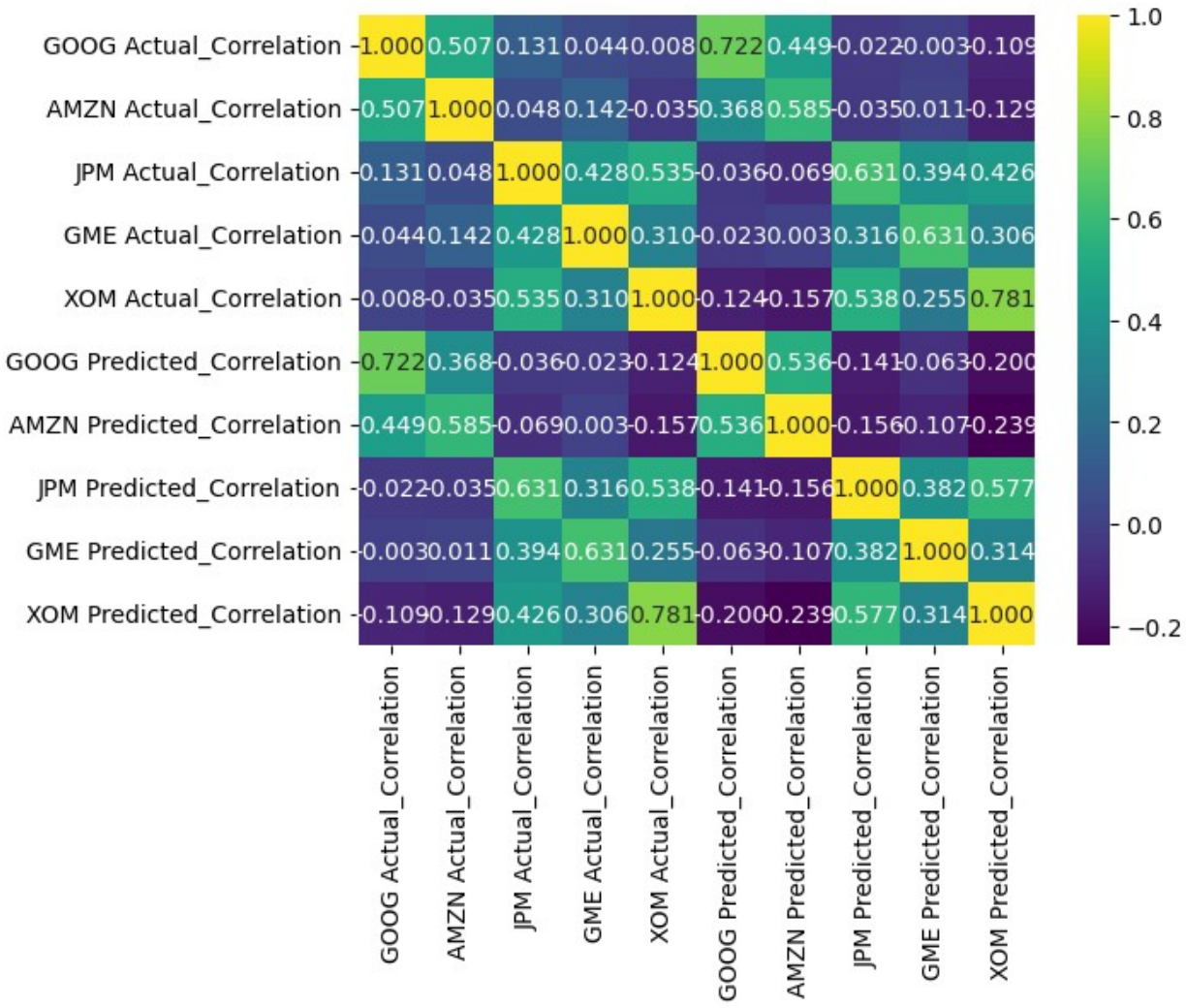
```
result_df = pd.concat([actual_correlations_df,
predicted_correlations_df], axis=1)
result_df
```

```
     GOOG Actual_Correlation  AMZN Actual_Correlation  JPM
Actual_Correlation  \
0                    0.350696                 0.503174
0.765971
1                    0.766777                 0.701655
0.785798
2                    0.743257                 0.448862
0.640816
3                    0.490805                 0.631620
0.395295
4                    0.727478                 0.548553
0.521924
..                        ...                      ...
...
941                  0.598892                 0.721729
0.714869
942                  0.485451                 0.762405
0.702972
943                  0.215713                 0.465185
0.843815
944                  0.716133                 0.614069
0.453927
945                  0.506069                 0.699236
0.721592

     GME Actual_Correlation  XOM Actual_Correlation  \
0                  0.234732                0.719560
1                  0.660197                0.672987
2                  0.210994                0.653368
3                  0.153777                0.146626
4                  0.306599                0.527696
..                      ...                     ...
941                0.035345                0.642374
942                0.732414                0.823043
943                0.551674                0.799734
944                0.282551                0.158553
945                0.425873                0.835022

     GOOG Predicted_Correlation  AMZN Predicted_Correlation  \
0                     0.410031                    0.474876
1                     0.785281                    0.694563
2                     0.756998                    0.644918
3                     0.730011                    0.615275
4                     0.782018                    0.576674
..                         ...                         ...
941                   0.729423                    0.729076
```

```
942                    0.571019                    0.535521
943                    0.535223                    0.474359
944                    0.575575                    0.526972
945                    0.707375                    0.686295

     JPM Predicted_Correlation  GME Predicted_Correlation  \
0                     0.731290                   0.313931
1                     0.780111                   0.554456
2                     0.521895                   0.214049
3                     0.547008                   0.546934
4                     0.565552                   0.243087
..                         ...                        ...
941                   0.716171                   0.082372
942                   0.759455                   0.466644
943                   0.814568                   0.500471
944                   0.655215                   0.296679
945                   0.770201                   0.393389

     XOM Predicted_Correlation
0                     0.637292
1                     0.585362
2                     0.489330
3                     0.194830
4                     0.414478
..                         ...
941                   0.629672
942                   0.688285
943                   0.807996
944                   0.373954
945                   0.734163

[946 rows x 10 columns]
```

```
sns.heatmap(result_df.corr(), cmap='viridis', annot=True, fmt=".3f")
```

```
<Axes: >
```




```
for stock in stock_list[:-1]:
  print(f"【{stock}】Mean Squared Error (MSE):", ((result_df[stock + ' Predicted_Correlation'] - result_df[stock + ' Actual_Correlation']) ** 2).mean())
```

```
【GOOG】Mean Squared Error (MSE): 0.01940743596578209
【AMZN】Mean Squared Error (MSE): 0.02430824856626552
【JPM】Mean Squared Error (MSE): 0.012258691865699263
【GME】Mean Squared Error (MSE): 0.02443172548733231
【XOM】Mean Squared Error (MSE): 0.017217325689977674
```

2 Price Prediction Signal

2.1 Project Instructions

2.1.1 Project Goal

Idea Generation for Price Prediction Signal for a Macro Asset or ETF

2.1.2 Project Objective

Objective: Conceptualize a novel price prediction signal for a chosen macro asset or broad-based index ETF. This project focuses on idea generation, understanding the underlying rationale, and potential data-driven insights that could back up the proposed signal. While the main emphasis is on the conceptual framework, implementation or preliminary testing is a valuable addition.

Begin by selecting a major macro asset or broad-based index ETF of interest. Based on your research, conceptualize a potential price prediction signal. This could be rooted in technical analysis, fundamental analysis, or any unconventional sources or patterns you've identified. Articulate the reasoning behind your proposed signal. Why do you believe it might have predictive power? Are there any historical events or data-driven insights that might support your idea? Detail your thought process in conceptualizing the signal, the research conducted, and the rationale behind its potential effectiveness. If you opted for preliminary testing, discuss your data, findings and the implications they might have for the feasibility of your signal.

2.2 project methodology

Here I choose stocks as the research object. Developing a stock price prediction methodology involves combining various techniques and factors to create a model that can forecast future stock prices. It can be a comprehensive methodology that incorporates both fundamental and technical analysis, machine learning, and risk management considerations. The model in Project 1 can also be used in price prediction, so in this project, I want to focus on feature engineering.

2.3 Feature Engineering

Feature engineering is a relatively important research content here. Feature engineering is to extract some data features of current assets through a series of data such as asset price changes, such as overlap features, momentum features, volatility features, morphological features and price features, etc.

2.3.1 factor c_v_corr_fct

Here I list several characteristic factors, the expression of the first one is：
-ts_corr(close, vol, 30)

This factor is relatively simple and is calculated based on the correlation between closing price and trading volume. The higher the correlation, the lower the factor score. This factor can combine price trends and market sentiment. The formula is simple but performs well. When analyzing the factors, in order to avoid using future data, I performed a lag operation on the factors to ensure that there would be no false winning factors in the calculation process.

The performance and detailed analysis of this factor are shown on the next pages:

Quantiles Statistics

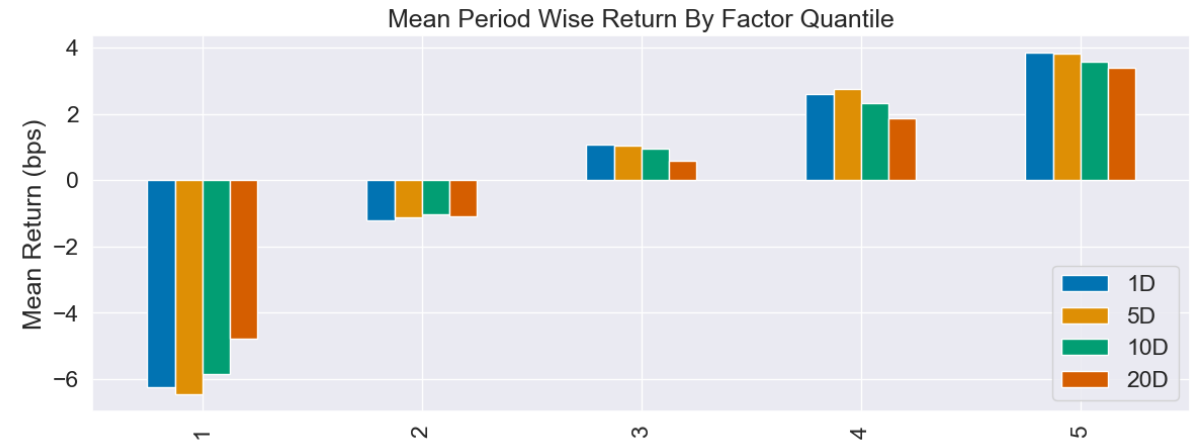
factor_quantile	min	max	mean	std	count	count %
1	-0.994458	-0.446346	-0.793762	0.076319	844125	20.011242
2	-0.885722	-0.162453	-0.636220	0.095352	843410	19.994291
3	-0.831025	0.111042	-0.486920	0.130205	843408	19.994244
4	-0.771042	0.380266	-0.295330	0.166567	843410	19.994291
5	-0.650743	0.952637	0.078368	0.253573	843901	20.005931

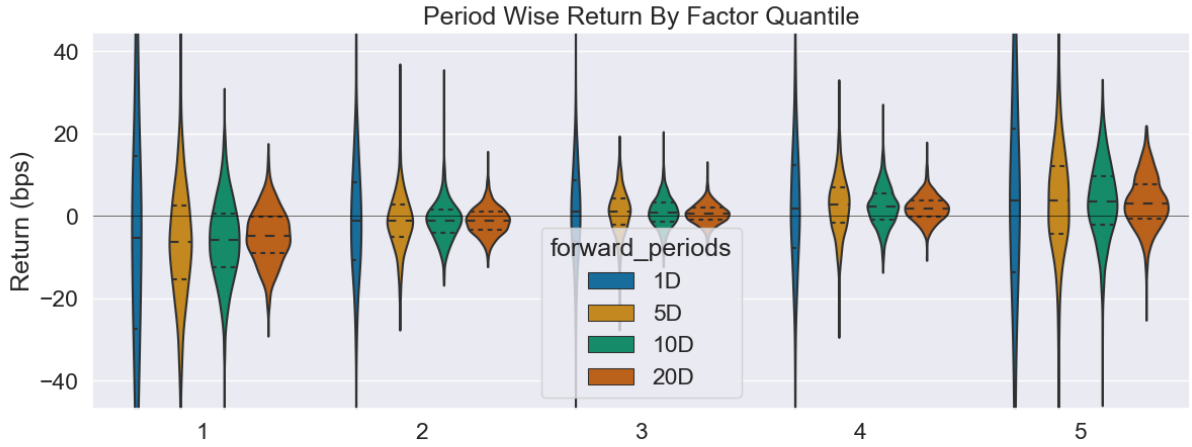
Returns Analysis

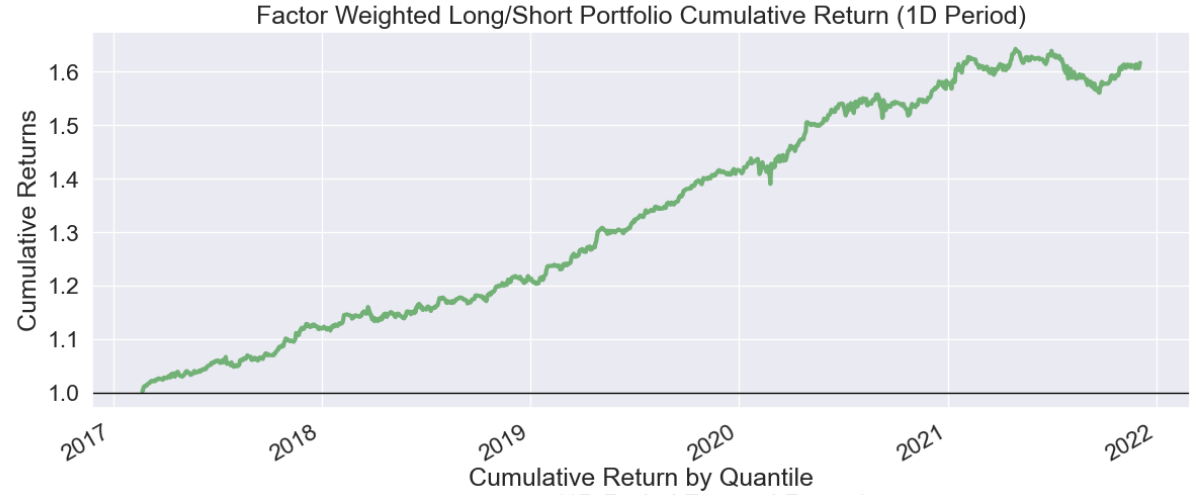
	1D	5D	10D	20D
Ann. alpha	0.116	0.115	0.105	0.091
beta	-0.079	-0.059	-0.058	-0.053
Mean Period Wise Return Top Quantile (bps)	3.851	3.817	3.574	3.400
Mean Period Wise Return Bottom Quantile (bps)	-6.264	-6.475	-5.842	-4.780
Mean Period Wise Spread (bps)	10.115	10.299	9.419	8.182

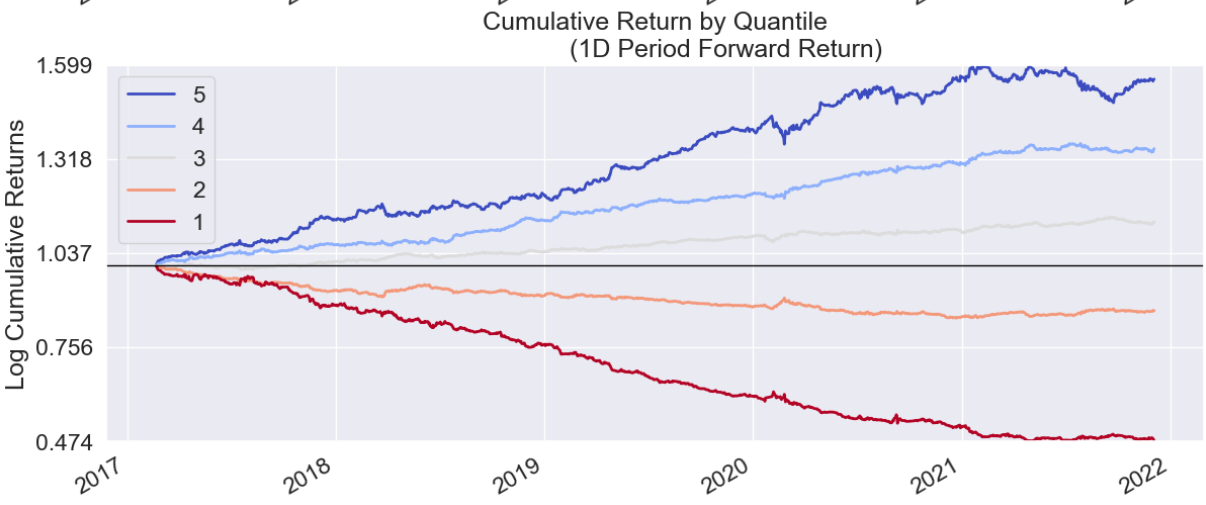
<Figure size 640x480 with 0 Axes>

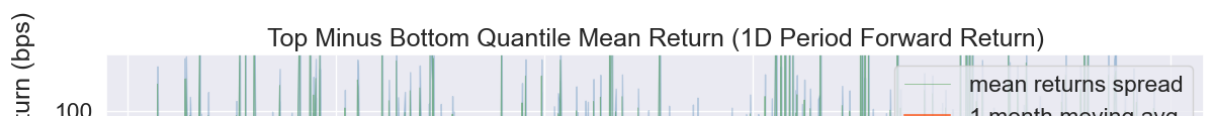
c_v_corr_fct

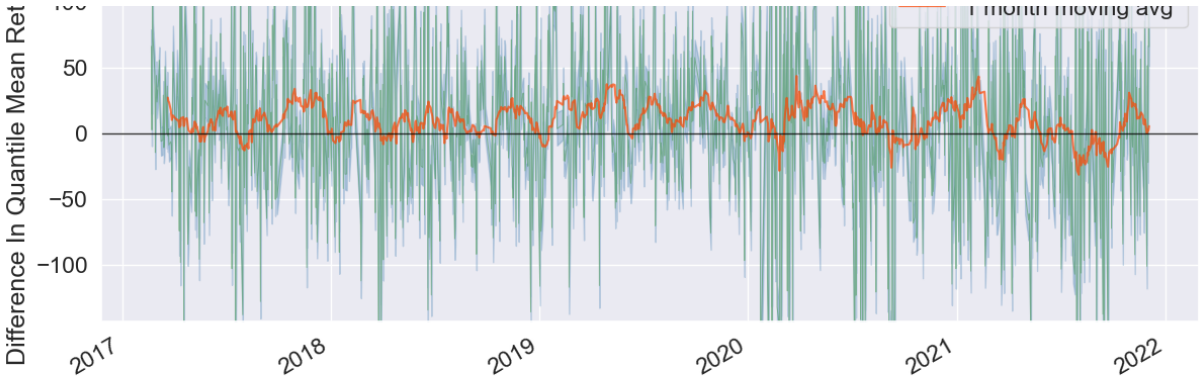

Mean Period Wise Return By Factor Quantile
Mean Return (bps)
4
2
0
−2
−4
−6
1
2
3
4
5
1D
5D
10D
20D

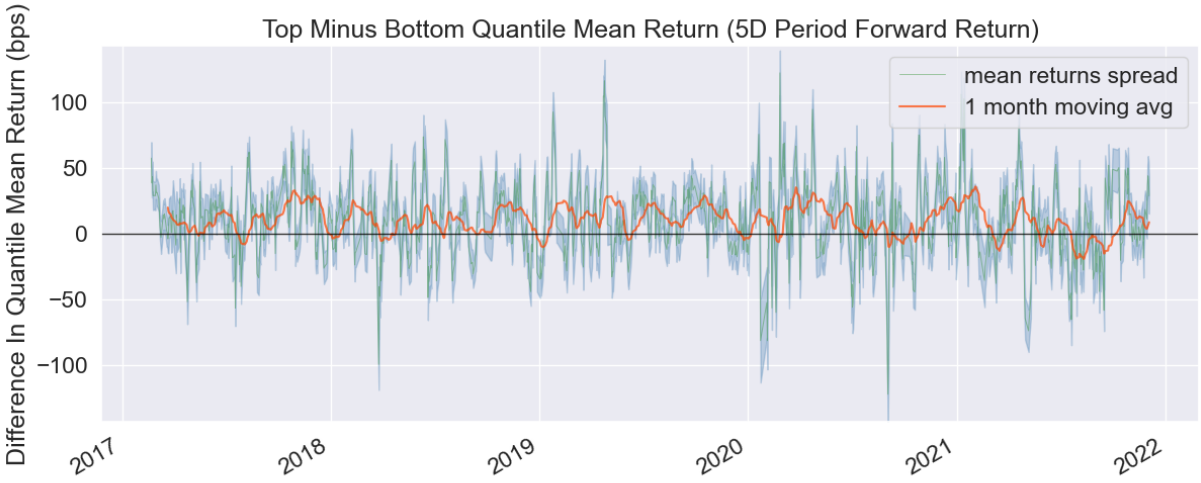

Period Wise Return By Factor Quantile
Return (bps)
40
20
0
−20
−40
1
2
3
4
5
forward_periods
1D
5D
10D
20D

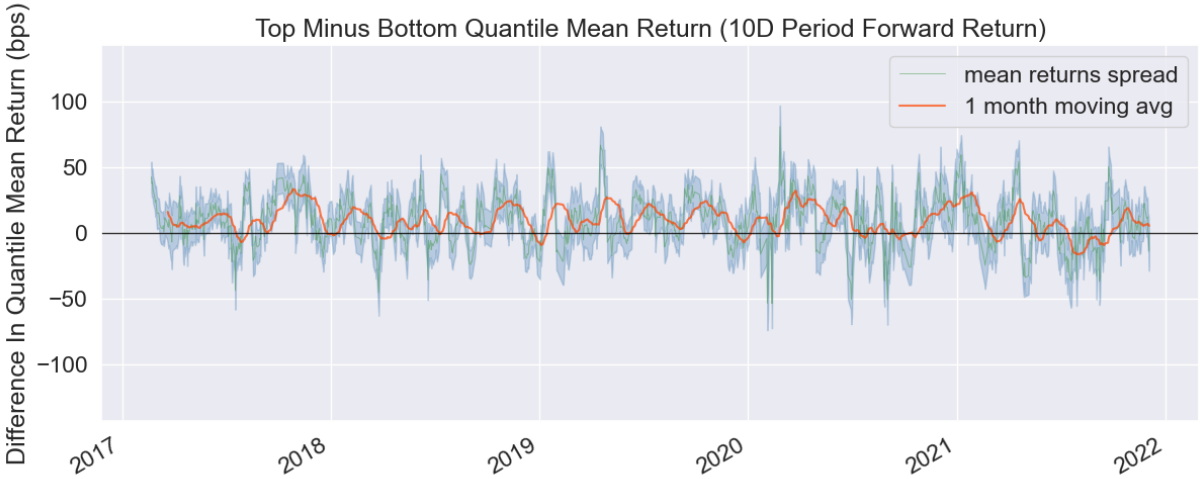

Factor Weighted Long/Short Portfolio Cumulative Return (1D Period)
Cumulative Returns
1.6
1.5
1.4
1.3
1.2
1.1
1.0
2017
2018
2019
2020
2021
2022

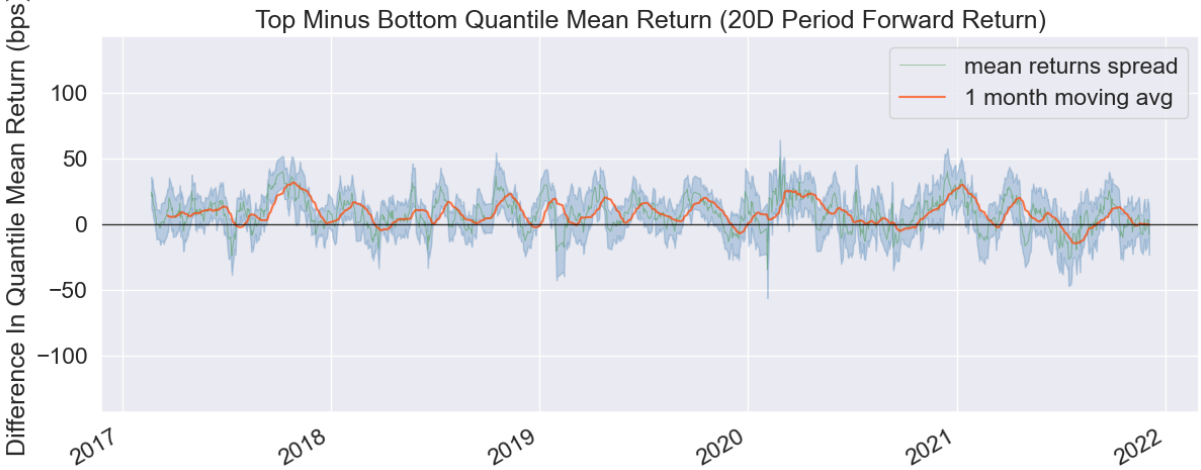

Cumulative Return by Quantile
(1D Period Forward Return)
Log Cumulative Returns
1.599
1.318
1.037
0.756
0.474
5
4
3
2
1
2017
2018
2019
2020
2021
2022


Top Minus Bottom Quantile Mean Return (1D Period Forward Return)
100
mean returns spread



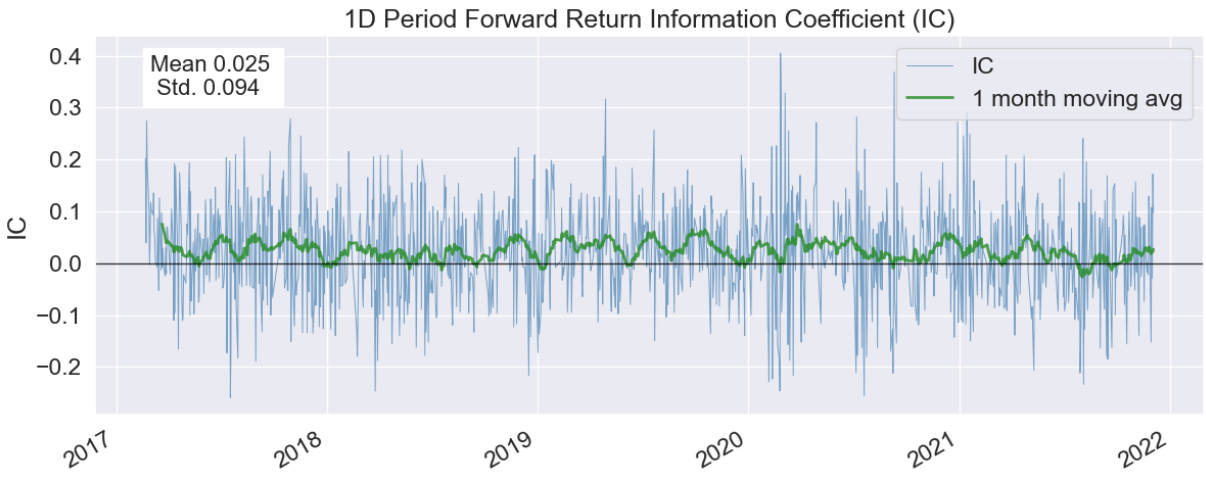


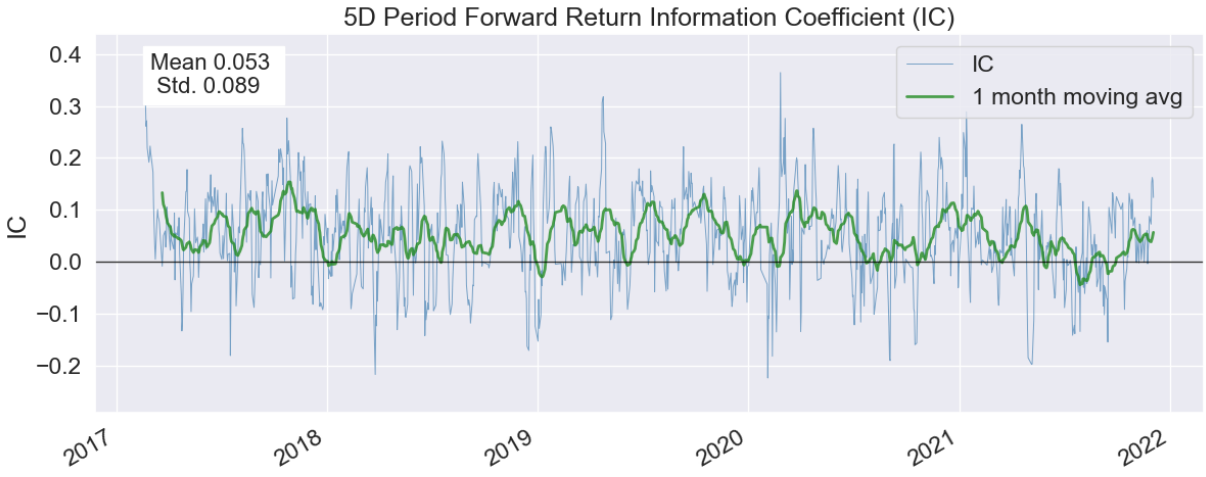


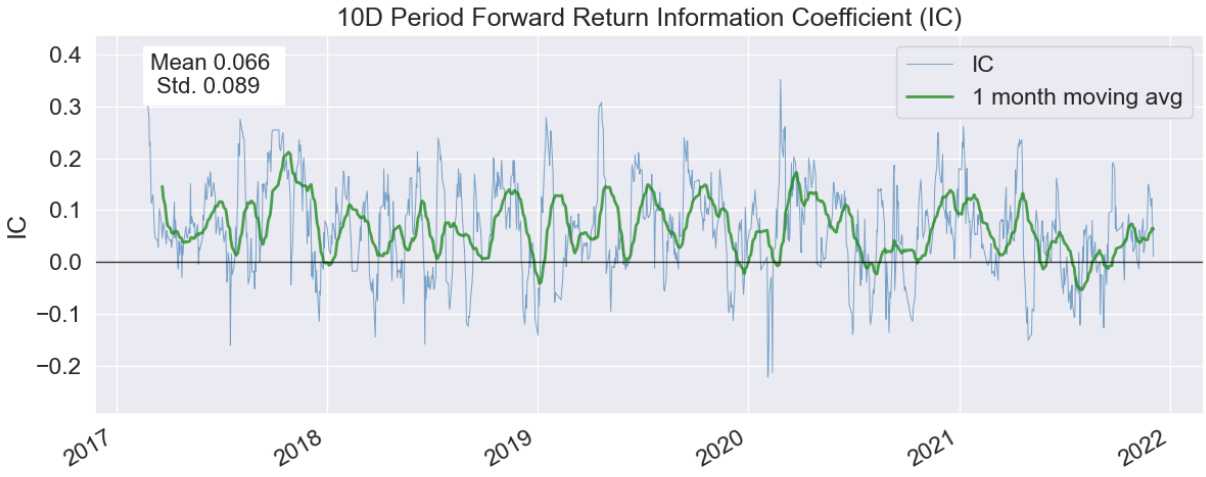


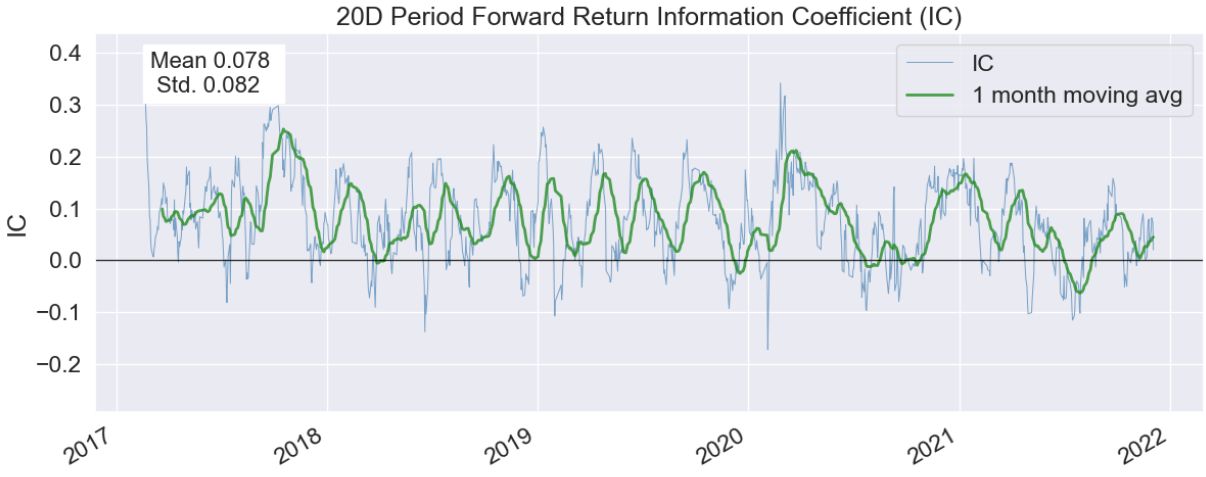
Information Analysis

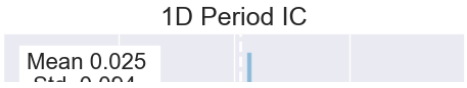
	1D	5D	10D	20D
IC Mean	0.025	0.053	0.066	0.078
IC Std.	0.094	0.089	0.089	0.082
Risk-Adjusted IC	0.265	0.594	0.739	0.950
t-stat(IC)	9.069	20.289	25.245	32.437
p-value(IC)	0.000	0.000	0.000	0.000
IC Skew	0.018	0.010	0.095	0.162
IC Kurtosis	0.437	0.046	0.018	-0.305

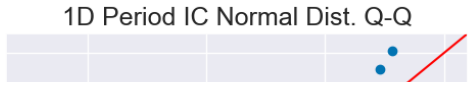

1D Period Forward Return Information Coefficient (IC)
Mean 0.025
Std. 0.094
IC
1 month moving avg
IC
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
2017
2018
2019
2020
2021
2022

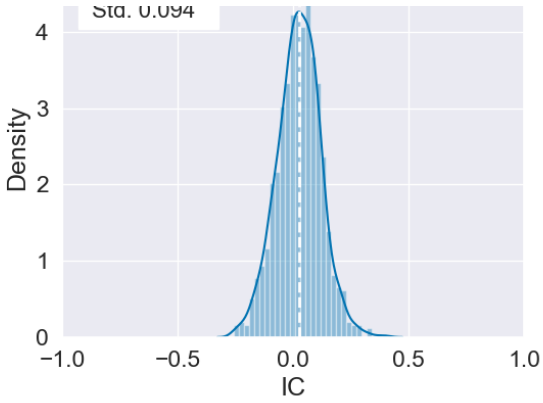

5D Period Forward Return Information Coefficient (IC)
Mean 0.053
Std. 0.089
IC
1 month moving avg
IC
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
2017
2018
2019
2020
2021
2022

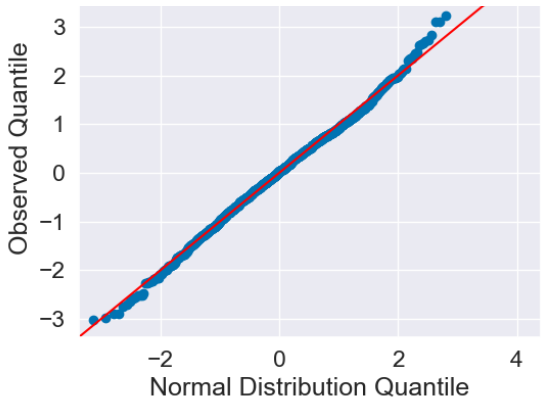

10D Period Forward Return Information Coefficient (IC)
Mean 0.066
Std. 0.089
IC
1 month moving avg
IC
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
2017
2018
2019
2020
2021
2022

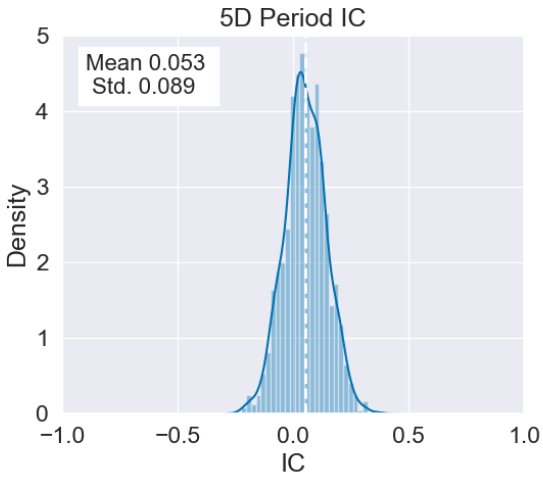

20D Period Forward Return Information Coefficient (IC)
Mean 0.078
Std. 0.082
IC
1 month moving avg
IC
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
2017
2018
2019
2020
2021
2022

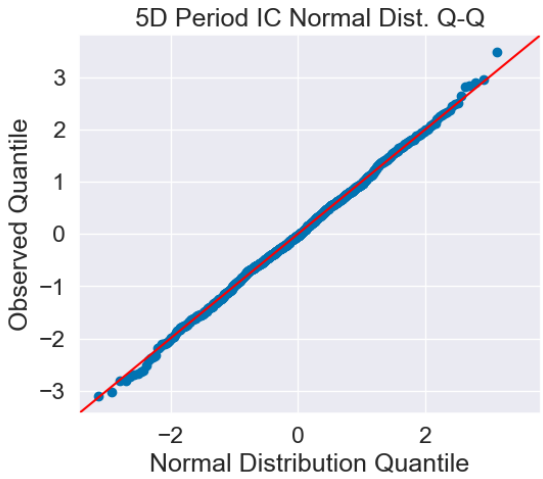

1D Period IC
Mean 0.025

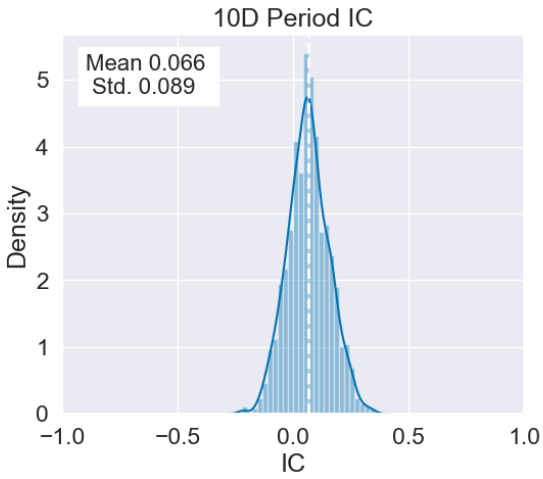

1D Period IC Normal Dist. Q-Q
4

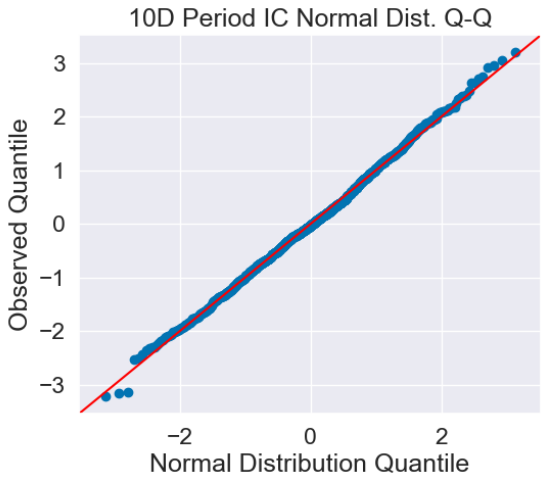

Std. 0.094
Density
IC

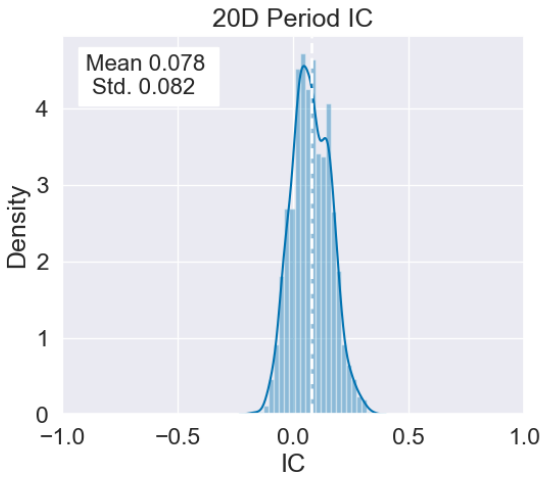

Observed Quantile
Normal Distribution Quantile

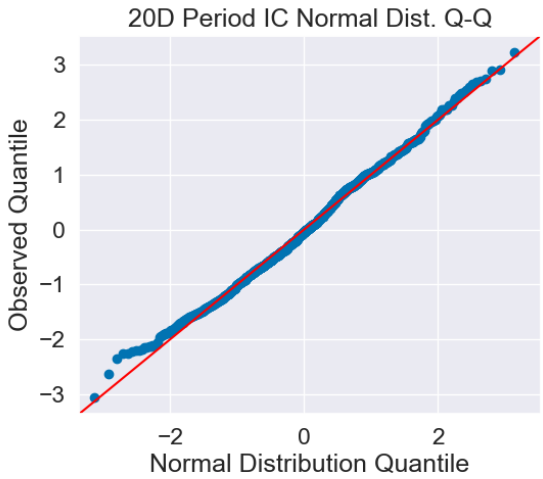

5D Period IC
Mean 0.053
Std. 0.089
Density
IC

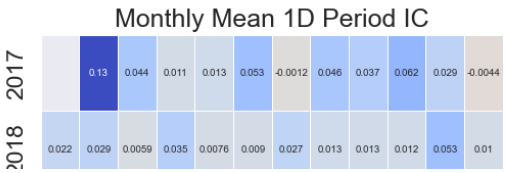

5D Period IC Normal Dist. Q-Q
Observed Quantile
Normal Distribution Quantile

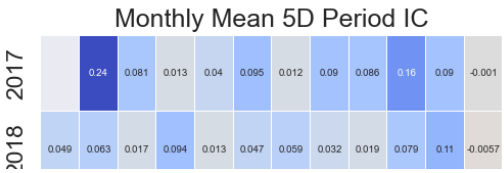

10D Period IC
Mean 0.066
Std. 0.089
Density
IC


10D Period IC Normal Dist. Q-Q
Observed Quantile
Normal Distribution Quantile

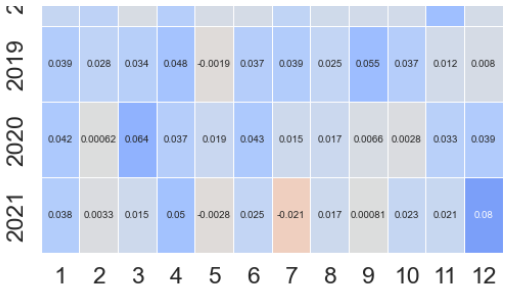

20D Period IC
Mean 0.078
Std. 0.082
Density
IC

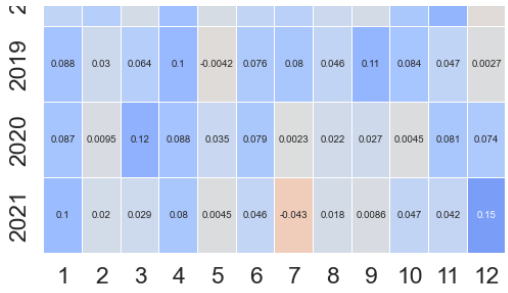

20D Period IC Normal Dist. Q-Q
Observed Quantile
Normal Distribution Quantile


Monthly Mean 1D Period IC
2017
0.13 0.044 0.011 0.013 0.053 -0.0012 0.046 0.037 0.062 0.029 -0.0044
2018
0.022 0.029 0.0059 0.035 0.0076 0.009 0.027 0.013 0.013 0.012 0.053 0.01

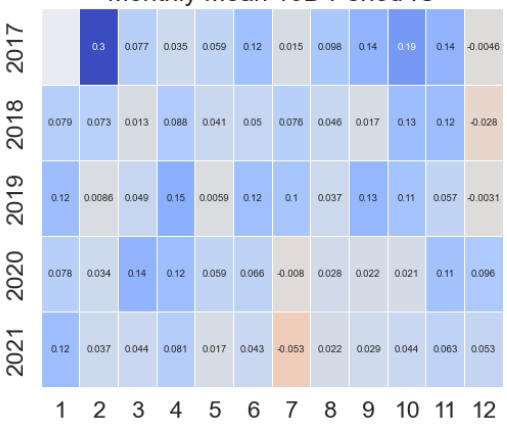

Monthly Mean 5D Period IC
2017
0.24 0.081 0.013 0.04 0.095 0.012 0.09 0.086 0.16 0.09 -0.001
2018
0.049 0.063 0.017 0.094 0.013 0.047 0.059 0.032 0.019 0.079 0.11 -0.0057

c_v_corr_fct

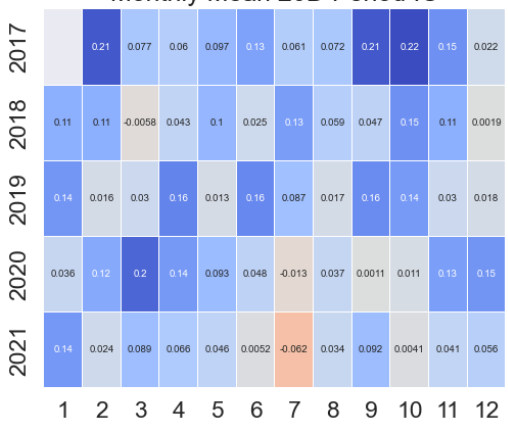




Monthly Mean 10D Period IC



Monthly Mean 20D Period IC



2.3.2 factor m_t_std_fct

The expression of the second factor is:
ts_std(amount, 6)

The factor expression is simpler. It only calculates the volatility of the total transaction amount in the previous 6 days. The greater the volatility, the higher the factor score. The lower the volatility, the lower the factor score. This factor is also intended to tap market sentiment.

The performance and detailed analysis of this factor are shown on the next pages:

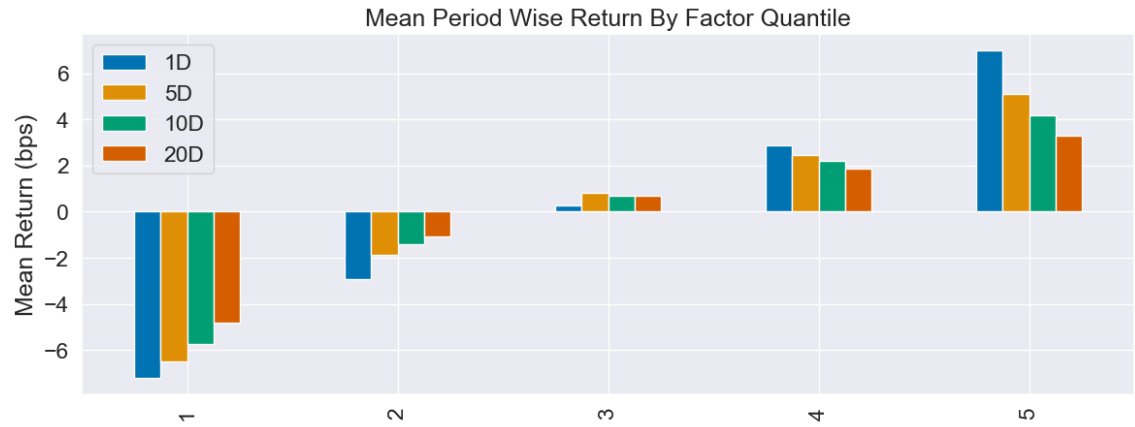
Quantiles Statistics

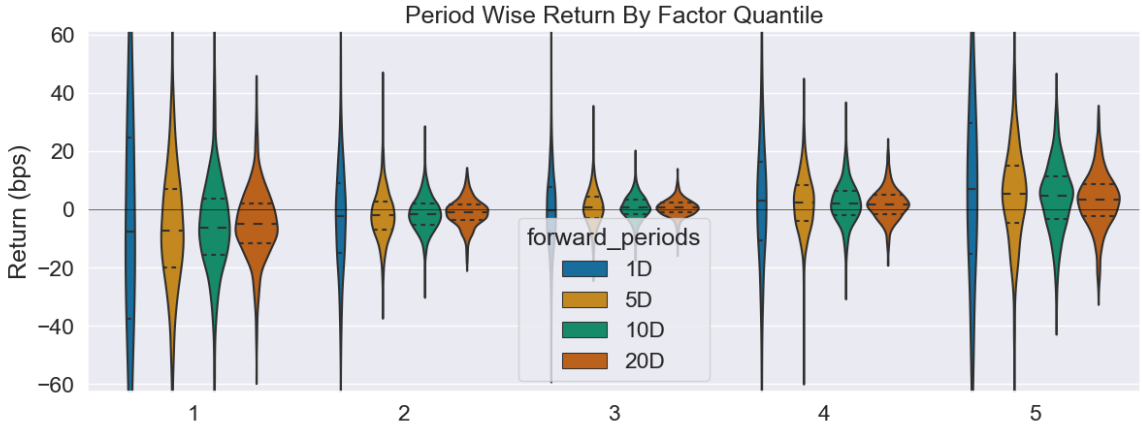
factor_quantile	min	max	mean	std	count	count %
1	-1.442277e+07	-28252.127411	-213936.967866	261882.632775	867129	20.011179
2	-1.578126e+05	-11426.377725	-49518.952992	20971.113029	866395	19.994240
3	-7.461550e+04	-5599.821912	-23464.378565	9651.764929	866412	19.994632
4	-4.029178e+04	-2477.386010	-11653.568645	5148.311021	866395	19.994240
5	-2.126236e+04	-10.332029	-4745.740997	2813.502663	866892	20.005709

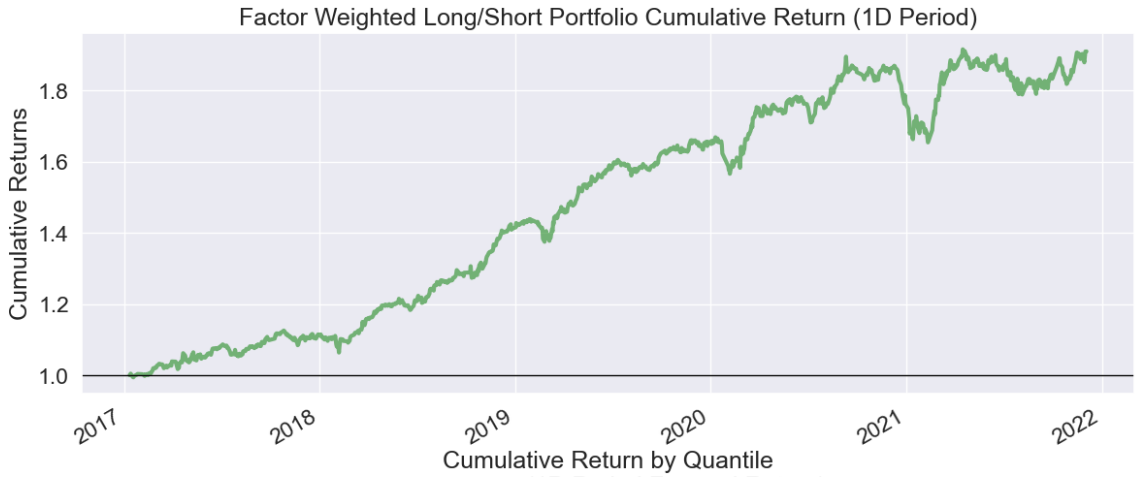
Returns Analysis

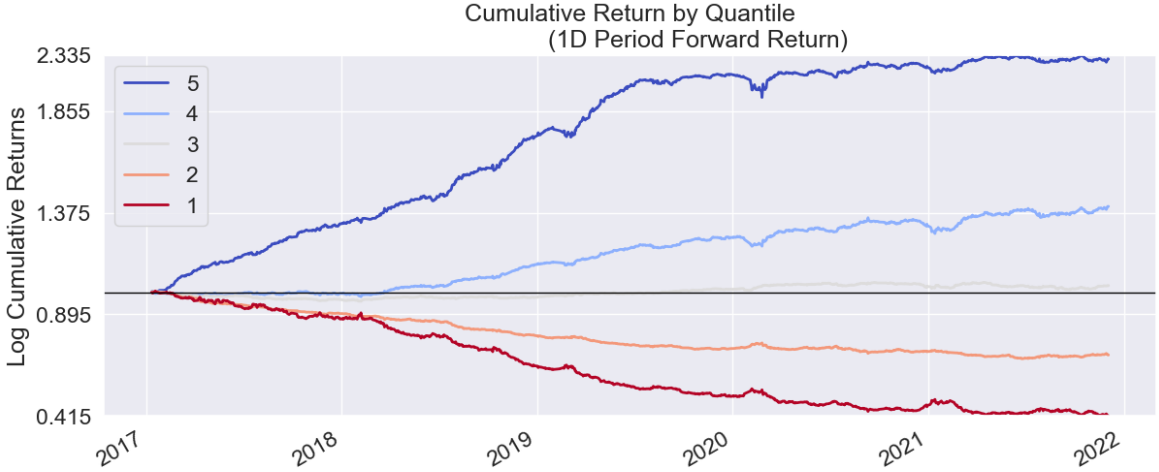
	1D	5D	10D	20D
Ann. alpha	0.154	0.130	0.115	0.097
beta	-0.048	0.009	0.026	0.043
Mean Period Wise Return Top Quantile (bps)	6.996	5.092	4.184	3.277
Mean Period Wise Return Bottom Quantile (bps)	-7.214	-6.488	-5.720	-4.816
Mean Period Wise Spread (bps)	14.210	11.632	9.961	8.154

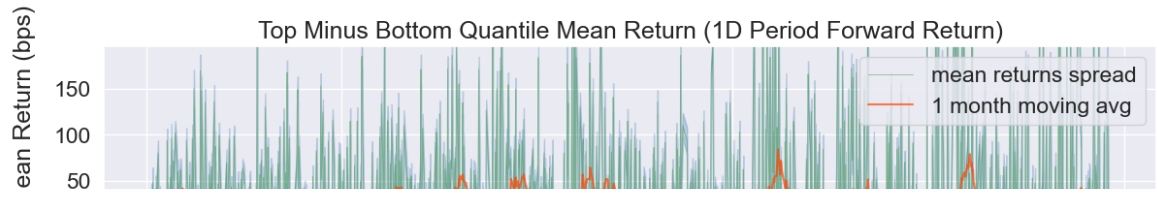
<Figure size 640x480 with 0 Axes>

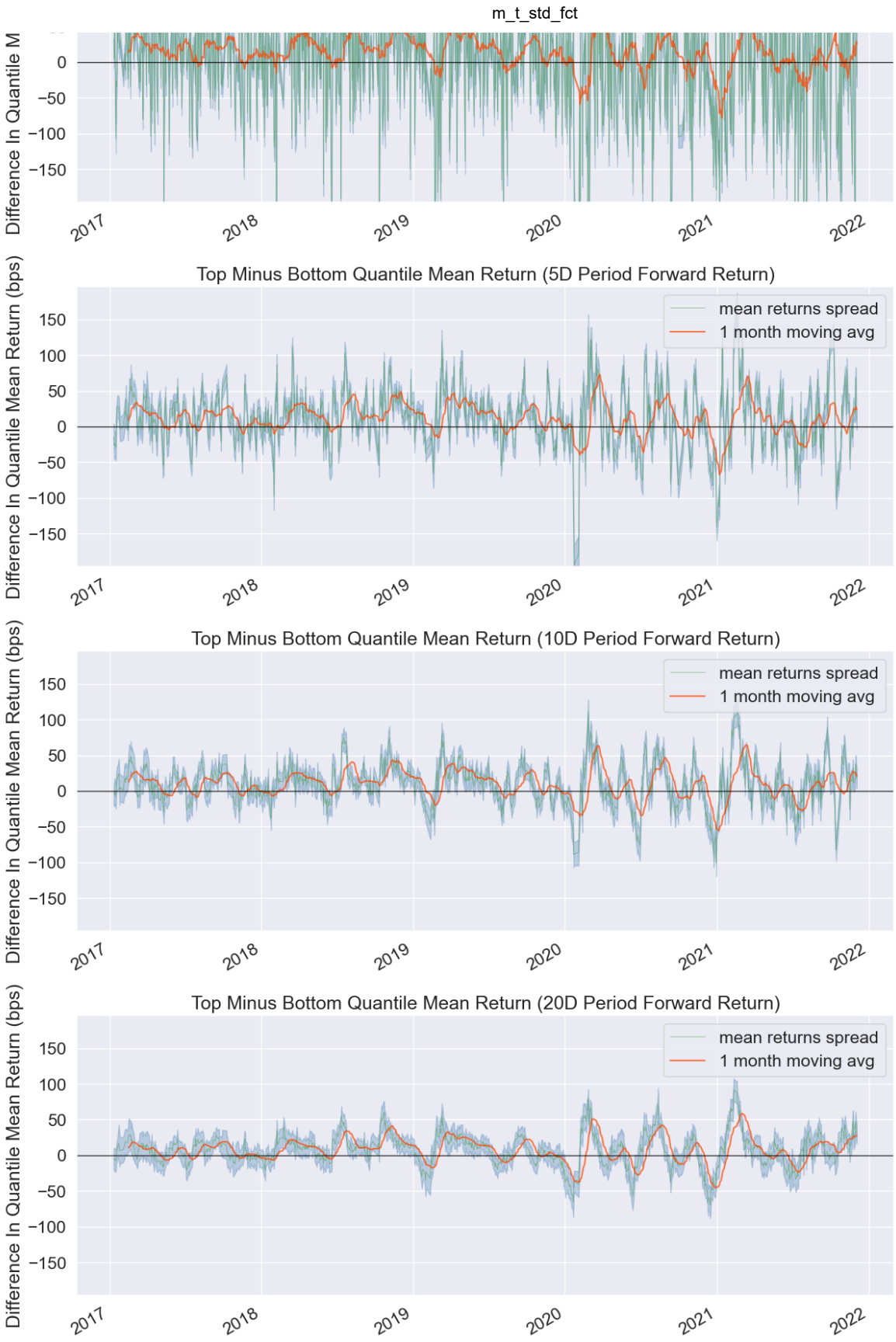

Mean Period Wise Return By Factor Quantile
Mean Return (bps)
1D
5D
10D
20D
6
4
2
0
−2
−4
−6
1
2
3
4
5


Period Wise Return By Factor Quantile
Return (bps)
60
40
20
0
−20
−40
−60
forward_periods
1D
5D
10D
20D
1
2
3
4
5


Factor Weighted Long/Short Portfolio Cumulative Return (1D Period)
Cumulative Returns
1.8
1.6
1.4
1.2
1.0
2017
2018
2019
2020
2021
2022

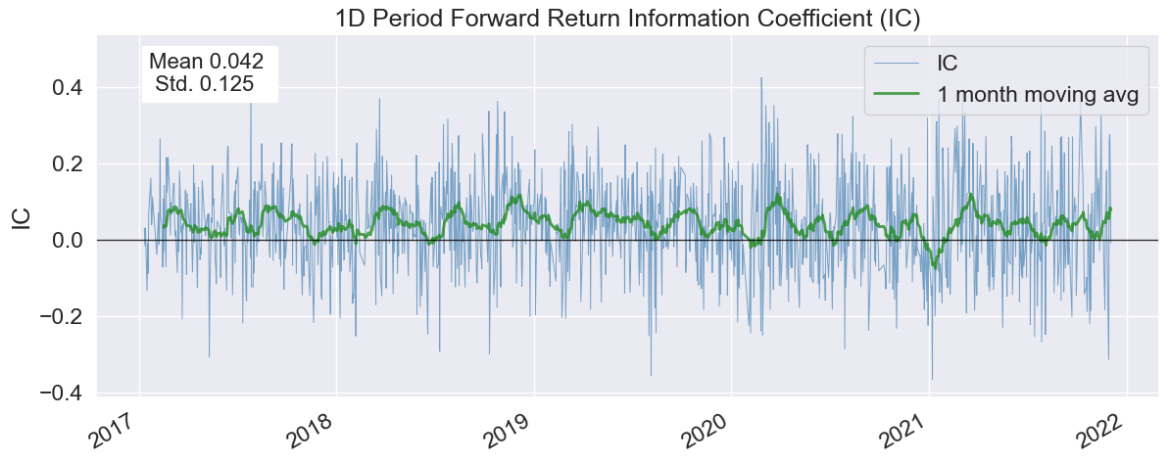

Cumulative Return by Quantile
(1D Period Forward Return)
Log Cumulative Returns
2.335
1.855
1.375
0.895
0.415
5
4
3
2
1
2017
2018
2019
2020
2021
2022

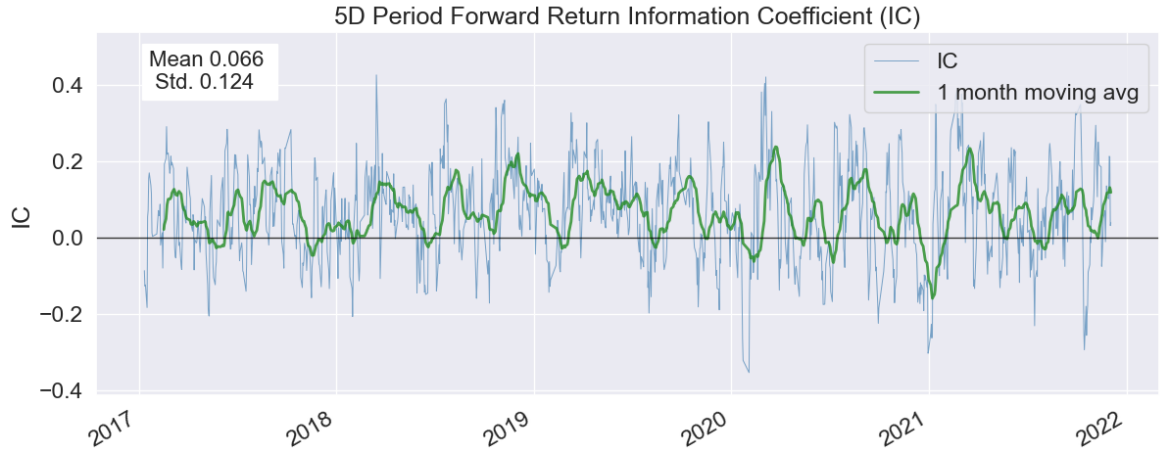

Top Minus Bottom Quantile Mean Return (1D Period Forward Return)
ean Return (bps)
150
100
50
mean returns spread
1 month moving avg

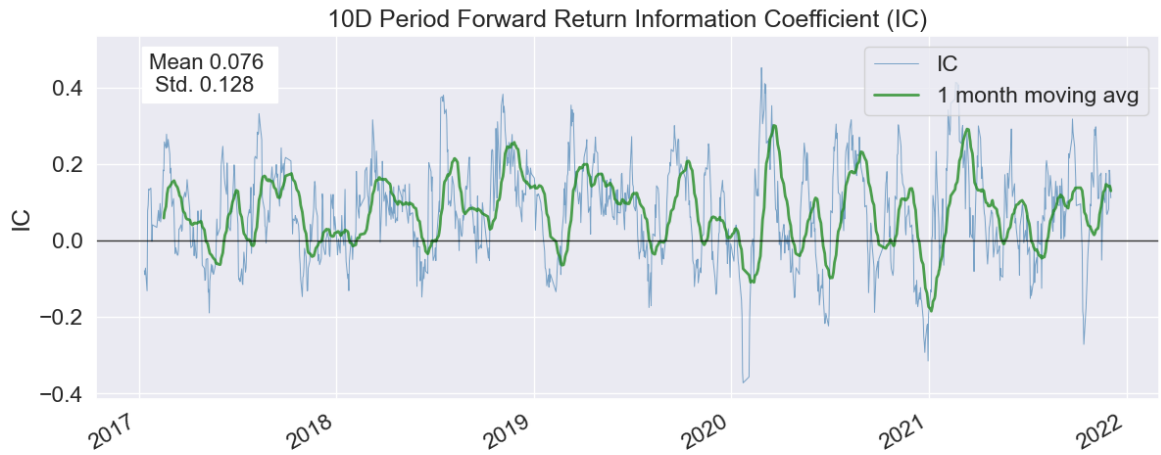


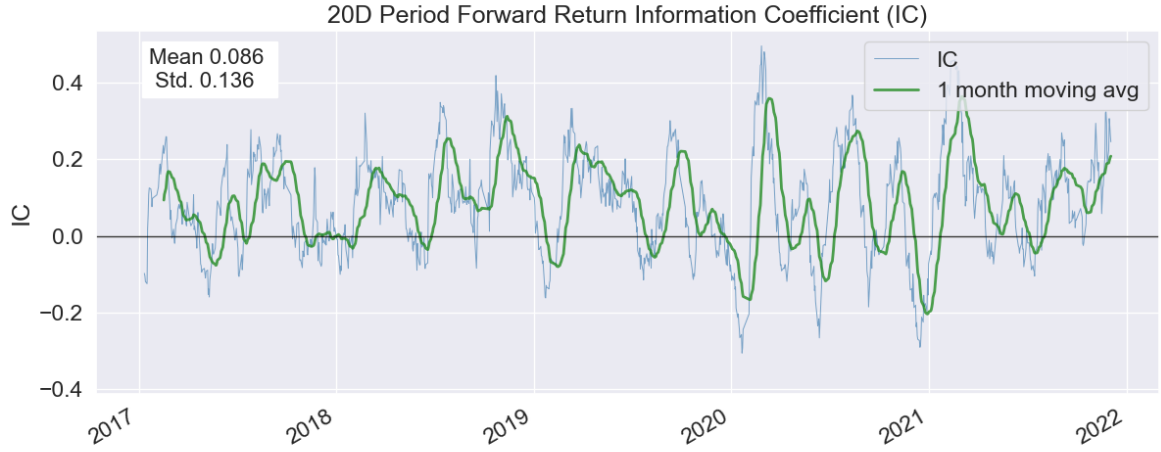
Information Analysis

	1D	5D	10D	20D
IC Mean	0.042	0.066	0.076	0.086
IC Std.	0.125	0.124	0.128	0.136
Risk-Adjusted IC	0.337	0.530	0.595	0.631
t-stat(IC)	11.631	18.287	20.518	21.777
p-value(IC)	0.000	0.000	0.000	0.000
IC Skew	-0.025	-0.031	-0.039	0.051
IC Kurtosis	0.018	-0.101	-0.009	-0.010


1D Period Forward Return Information Coefficient (IC)
Mean 0.042
Std. 0.125
IC
1 month moving avg
IC
0.4
0.2
0.0
−0.2
−0.4
2017
2018
2019
2020
2021
2022


5D Period Forward Return Information Coefficient (IC)
Mean 0.066
Std. 0.124
IC
1 month moving avg
IC
0.4
0.2
0.0
−0.2
−0.4
2017
2018
2019
2020
2021
2022

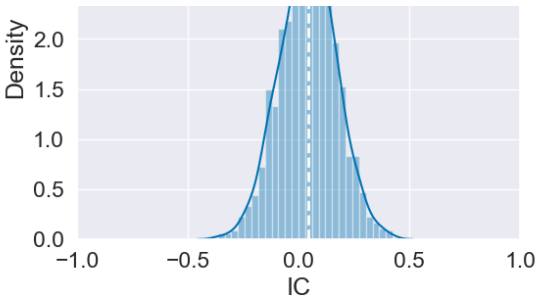

10D Period Forward Return Information Coefficient (IC)
Mean 0.076
Std. 0.128
IC
1 month moving avg
IC
0.4
0.2
0.0
−0.2
−0.4
2017
2018
2019
2020
2021
2022

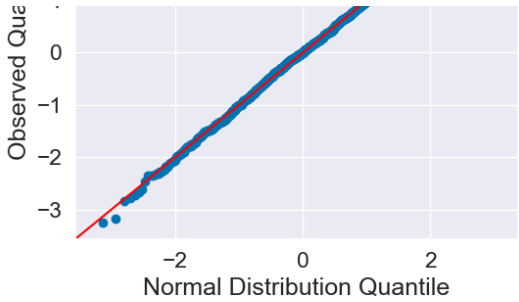

20D Period Forward Return Information Coefficient (IC)
Mean 0.086
Std. 0.136
IC
1 month moving avg
IC
0.4
0.2
0.0
−0.2
−0.4
2017
2018
2019
2020
2021
2022

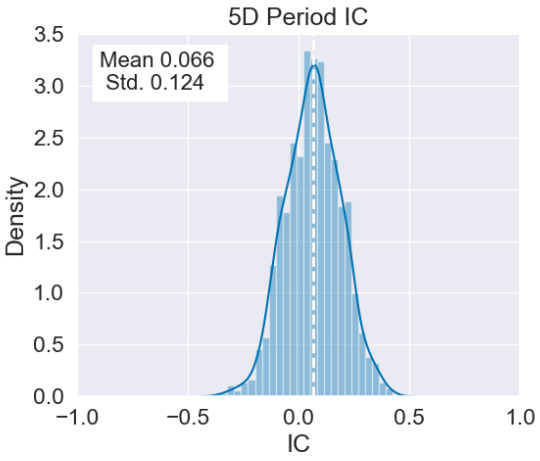

1D Period IC
Mean 0.042
Std. 0.125
3.5
3.0
2.5

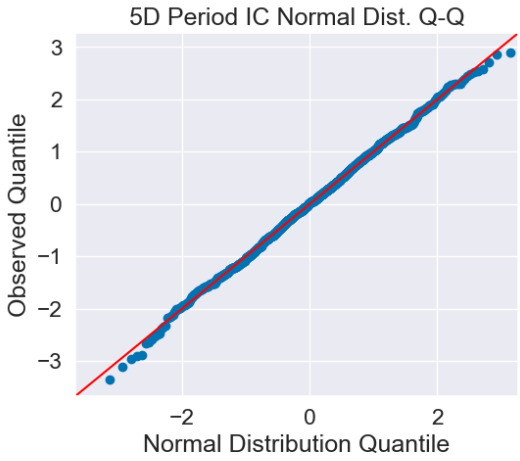
1D Period IC Normal Dist. Q-Q
3
2
1

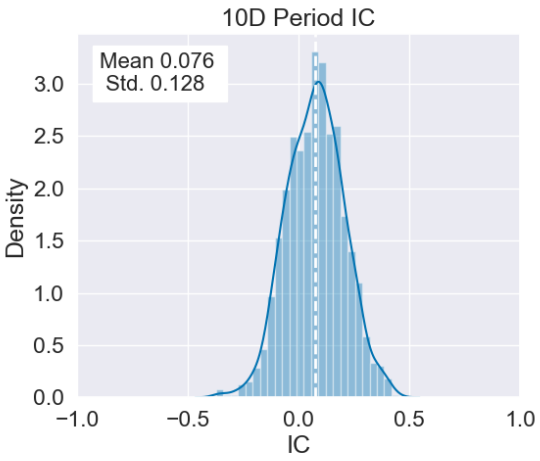
m_t_std_fct

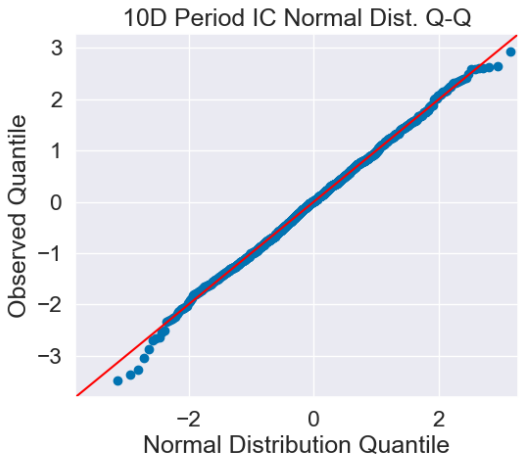

Density
IC

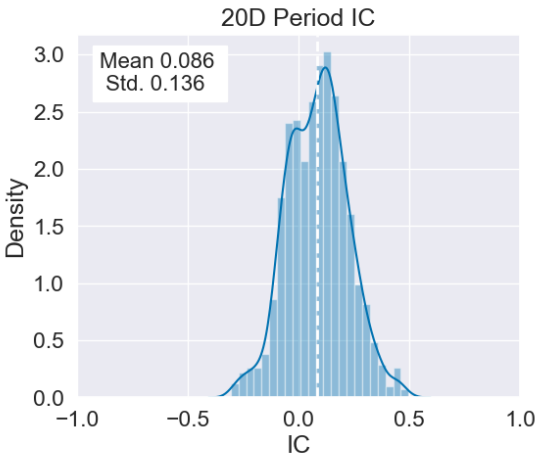

Observed Qua
Normal Distribution Quantile

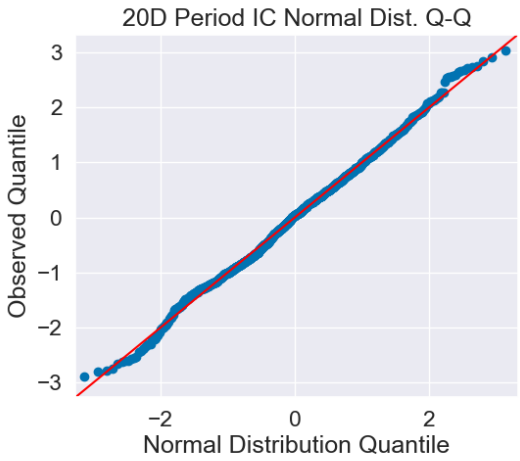

5D Period IC
Mean 0.066
Std. 0.124
Density
IC

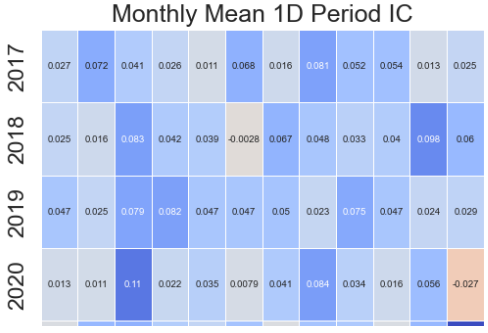

5D Period IC Normal Dist. Q-Q
Observed Quantile
Normal Distribution Quantile

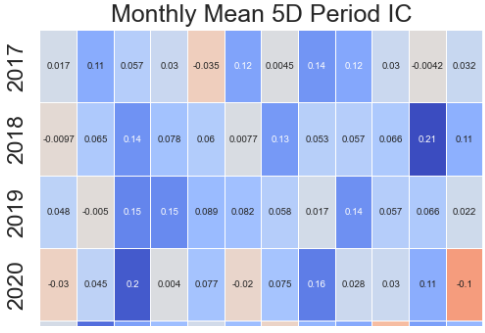

10D Period IC
Mean 0.076
Std. 0.128
Density
IC


10D Period IC Normal Dist. Q-Q
Observed Quantile
Normal Distribution Quantile


20D Period IC
Mean 0.086
Std. 0.136
Density
IC


20D Period IC Normal Dist. Q-Q
Observed Quantile
Normal Distribution Quantile

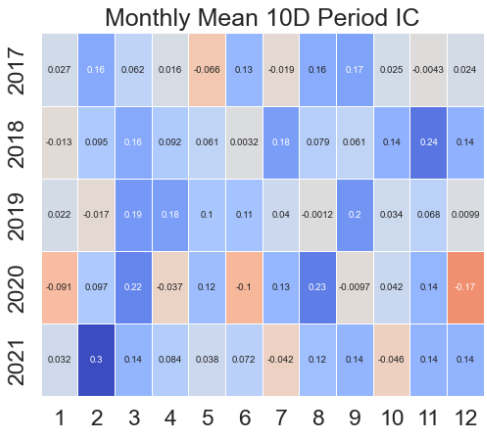

Monthly Mean 1D Period IC
2017
0.027 0.072 0.041 0.026 0.011 0.068 0.016 0.081 0.052 0.054 0.013 0.025
2018
0.025 0.016 0.083 0.042 0.039 -0.0028 0.067 0.048 0.033 0.04 0.098 0.06
2019
0.047 0.025 0.079 0.082 0.047 0.047 0.05 0.023 0.075 0.047 0.024 0.029
2020
0.013 0.011 0.11 0.022 0.035 0.0079 0.041 0.084 0.034 0.016 0.056 -0.027

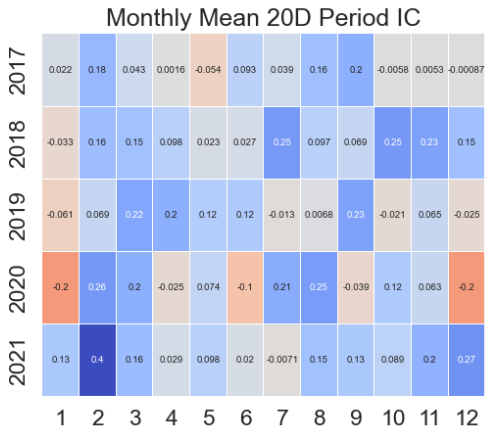

Monthly Mean 5D Period IC
2017
0.017 0.11 0.057 0.03 -0.035 0.12 0.0045 0.14 0.12 0.03 -0.0042 0.032
2018
-0.0097 0.065 0.14 0.078 0.06 0.0077 0.13 0.053 0.057 0.066 0.21 0.11
2019
0.048 -0.005 0.15 0.15 0.089 0.082 0.058 0.017 0.14 0.057 0.066 0.022
2020
-0.03 0.045 0.2 0.004 0.077 -0.02 0.075 0.16 0.028 0.03 0.11 -0.1


2021
0.0041 0.062 0.068 0.039 0.026 0.051 0.0034 0.05 0.044 0.0047 0.061 0.14
1 2 3 4 5 6 7 8 9 10 11 12

m_t_std_fct


2021
0.013 0.18 0.13 0.086 0.019 0.086 -0.026 0.099 0.11 -0.064 0.13 0.094
1 2 3 4 5 6 7 8 9 10 11 12


Monthly Mean 10D Period IC
2017
0.027 0.16 0.062 0.016 -0.066 0.13 -0.019 0.16 0.17 0.025 -0.0043 0.024
2018
-0.013 0.095 0.16 0.092 0.061 0.0032 0.18 0.079 0.061 0.14 0.24 0.14
2019
0.022 -0.017 0.19 0.18 0.1 0.11 0.04 -0.0012 0.2 0.034 0.068 0.0099
2020
-0.091 0.097 0.22 -0.037 0.12 -0.1 0.13 0.23 -0.0097 0.042 0.14 -0.17
2021
0.032 0.3 0.14 0.084 0.038 0.072 -0.042 0.12 0.14 -0.046 0.14 0.14
1 2 3 4 5 6 7 8 9 10 11 12


Monthly Mean 20D Period IC
2017
0.022 0.18 0.043 0.0016 -0.054 0.093 0.039 0.16 0.2 -0.0058 0.0053 -0.00087
2018
-0.033 0.16 0.15 0.098 0.023 0.027 0.25 0.097 0.069 0.25 0.23 0.15
2019
-0.061 0.069 0.22 0.2 0.12 0.12 -0.013 0.0068 0.23 -0.021 0.065 -0.025
2020
-0.2 0.26 0.2 -0.025 0.074 -0.1 0.21 0.25 -0.039 0.12 0.063 -0.2
2021
0.13 0.4 0.16 0.029 0.098 0.02 -0.0071 0.15 0.13 0.089 0.2 0.27
1 2 3 4 5 6 7 8 9 10 11 12

2.3.3 factor h_l_delta_fct

The factor expression is:
-ts_delta(((close - low) - (high - close)) / (high - low), 1)

This factor combines the opening price, closing price, highest price and lowest price of a single day, and predicts future returns by analyzing the shape of the K-line on a single day. If the stock's closing price on that day is closer to the highest price than the previous day, then the factor believes that the stock's price will rise in the future.

The performance and detailed analysis of this factor are shown on the next pages:

Quantiles Statistics

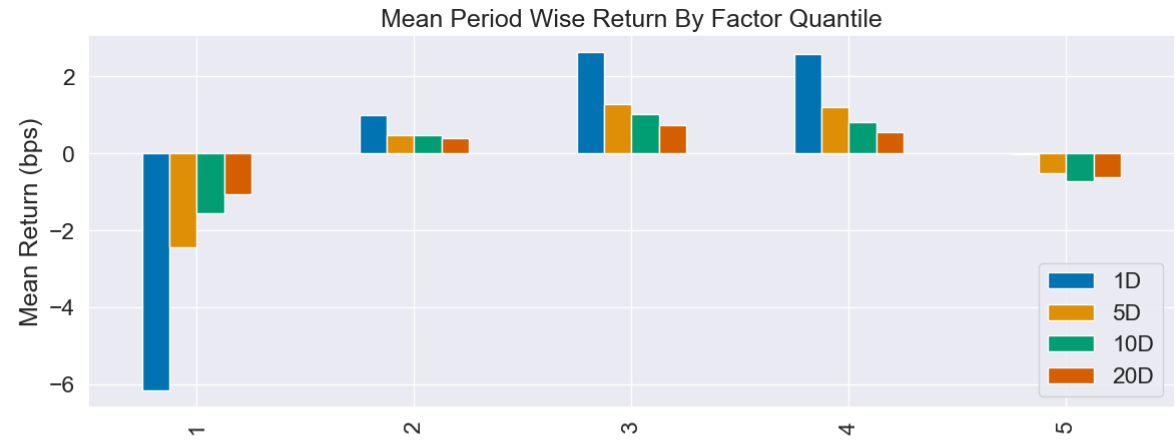
factor_quantile	min	max	mean	std	count	count %
1	-2.000000	1.032273	-0.960605	0.555525	869712	20.018911
2	-1.859338	1.400000	-0.375545	0.569785	868806	19.998057
3	-1.749348	1.646154	-0.009053	0.580128	868463	19.990162
4	-1.588089	1.818182	0.367516	0.571129	868568	19.992579
5	-1.230769	2.000000	0.987786	0.554489	868903	20.000290

Returns Analysis

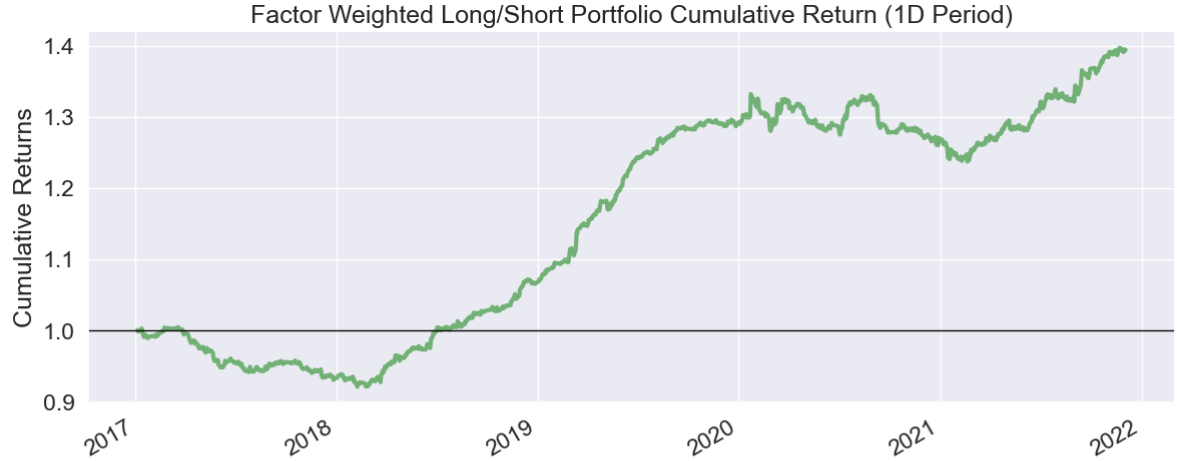
	1D	5D	10D	20D
Ann. alpha	0.073	0.022	0.009	0.005
beta	0.022	0.008	0.005	0.008
Mean Period Wise Return Top Quantile (bps)	-0.022	-0.514	-0.726	-0.630
Mean Period Wise Return Bottom Quantile (bps)	-6.168	-2.438	-1.560	-1.057
Mean Period Wise Spread (bps)	6.145	1.923	0.832	0.427

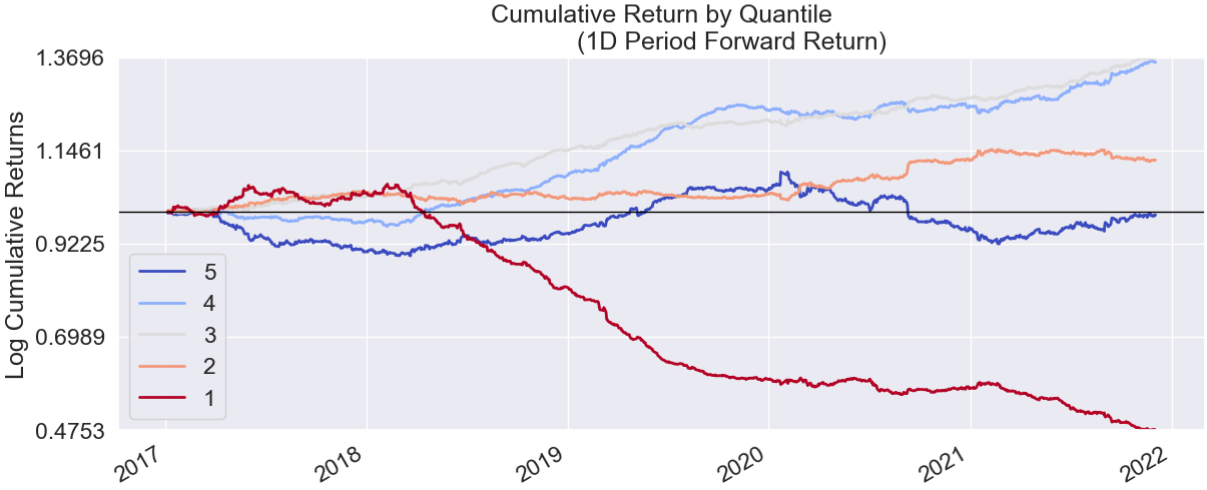
<Figure size 640x480 with 0 Axes>

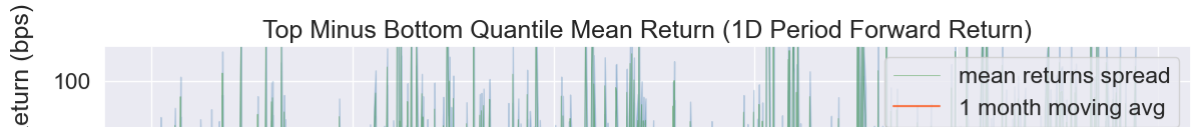
h_l_delta_fct

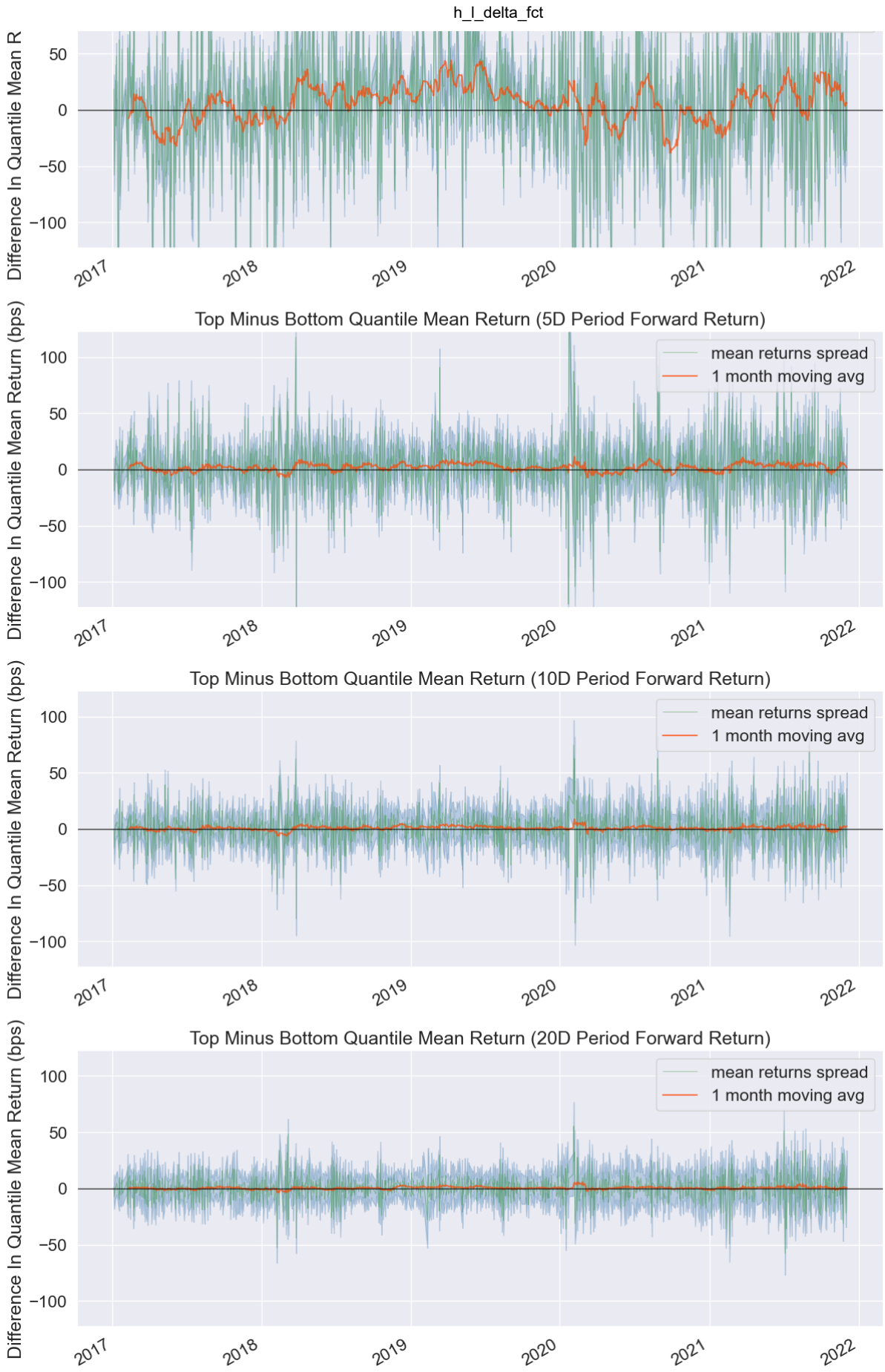

Mean Period Wise Return By Factor Quantile
Mean Return (bps)
2
0
−2
−4
−6
1
2
3
4
5
1D
5D
10D
20D


Period Wise Return By Factor Quantile
Return (bps)
20
0
−20
−40
1
2
3
4
5
forward_periods
1D
5D
10D
20D


Factor Weighted Long/Short Portfolio Cumulative Return (1D Period)
Cumulative Returns
1.4
1.3
1.2
1.1
1.0
0.9
2017
2018
2019
2020
2021
2022

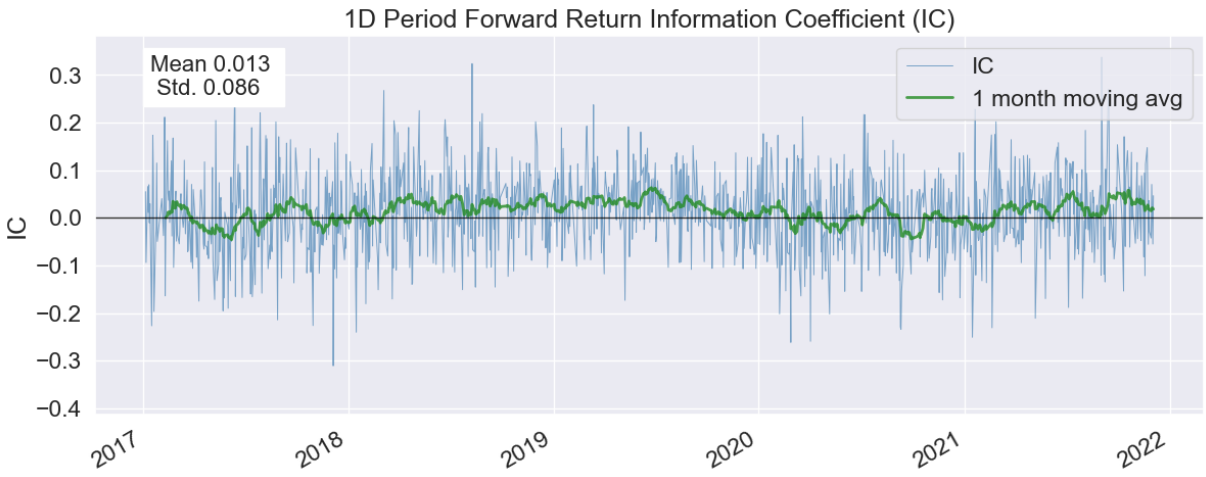

Cumulative Return by Quantile
(1D Period Forward Return)
Log Cumulative Returns
1.3696
1.1461
0.9225
0.6989
0.4753
5
4
3
2
1
2017
2018
2019
2020
2021
2022

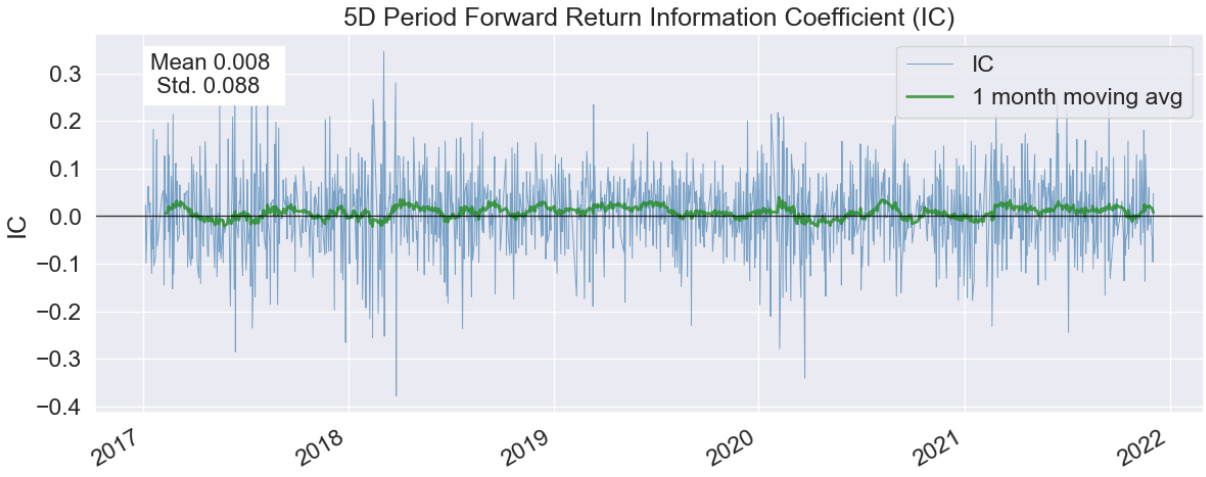

Top Minus Bottom Quantile Mean Return (1D Period Forward Return)
eturn (bps)
100
mean returns spread
1 month moving avg

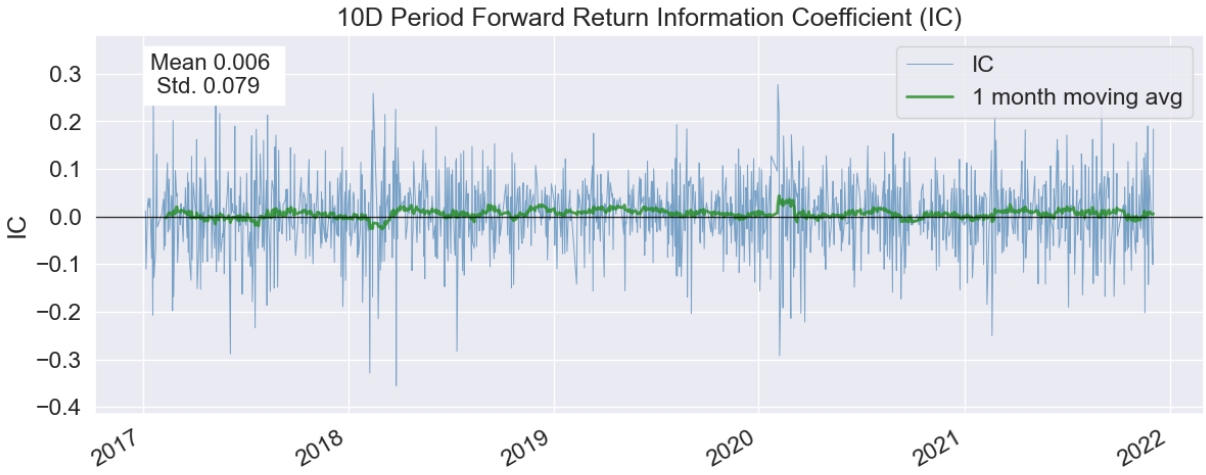


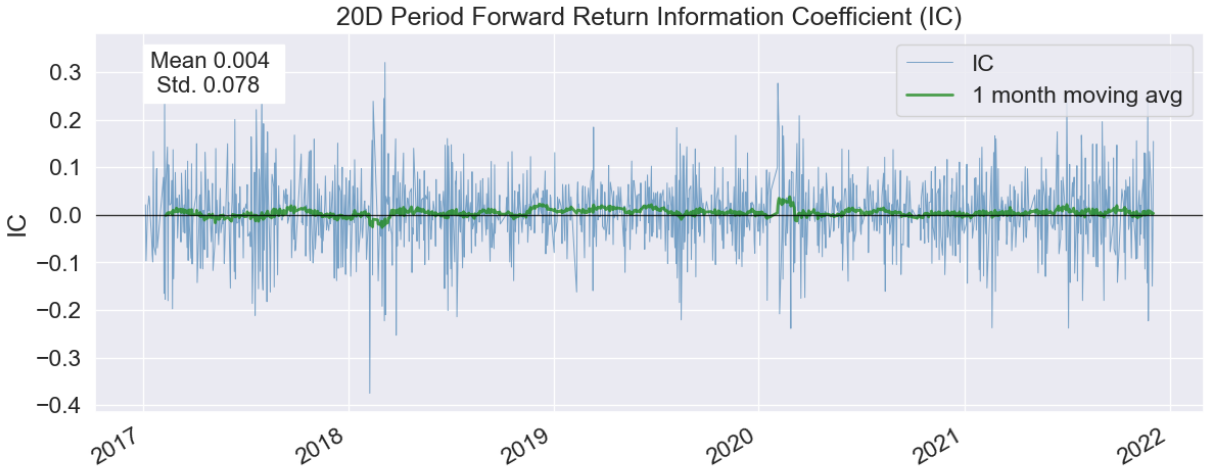
Information Analysis

	1D	5D	10D	20D
IC Mean	0.013	0.008	0.006	0.004
IC Std.	0.086	0.088	0.079	0.078
Risk-Adjusted IC	0.157	0.094	0.074	0.056
t-stat(IC)	5.411	3.244	2.548	1.945
p-value(IC)	0.000	0.001	0.011	0.052
IC Skew	-0.041	-0.130	-0.265	-0.105
IC Kurtosis	0.544	0.841	1.267	1.140

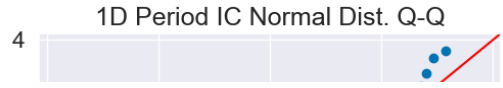

1D Period Forward Return Information Coefficient (IC)
Mean 0.013
Std. 0.086
IC
1 month moving avg
IC
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
2017
2018
2019
2020
2021
2022

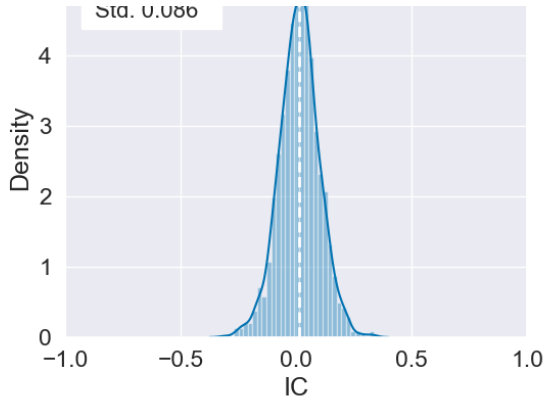

5D Period Forward Return Information Coefficient (IC)
Mean 0.008
Std. 0.088
IC
1 month moving avg
IC
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
2017
2018
2019
2020
2021
2022

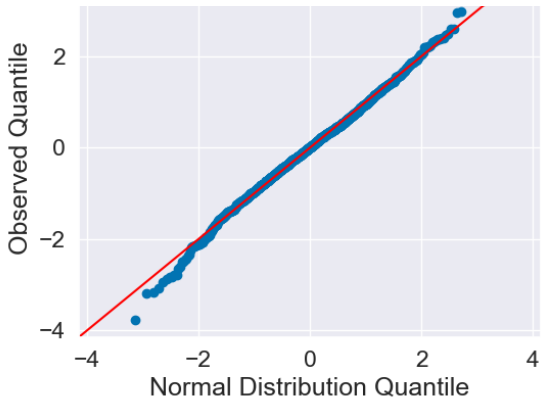

10D Period Forward Return Information Coefficient (IC)
Mean 0.006
Std. 0.079
IC
1 month moving avg
IC
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
2017
2018
2019
2020
2021
2022


20D Period Forward Return Information Coefficient (IC)
Mean 0.004
Std. 0.078
IC
1 month moving avg
IC
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
2017
2018
2019
2020
2021
2022

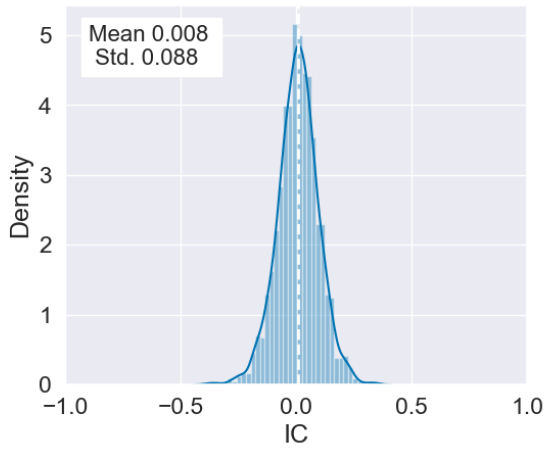

1D Period IC
5
Mean 0.013


1D Period IC Normal Dist. Q-Q
4

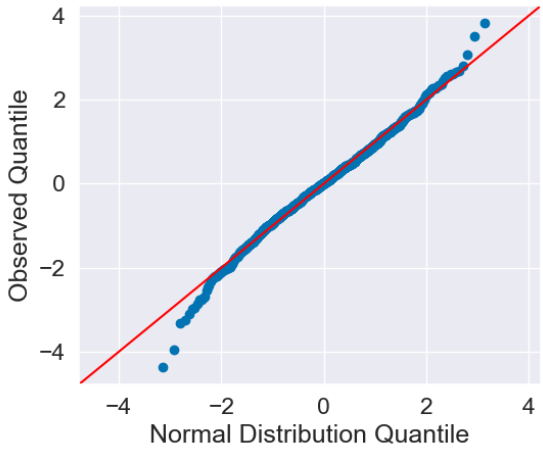




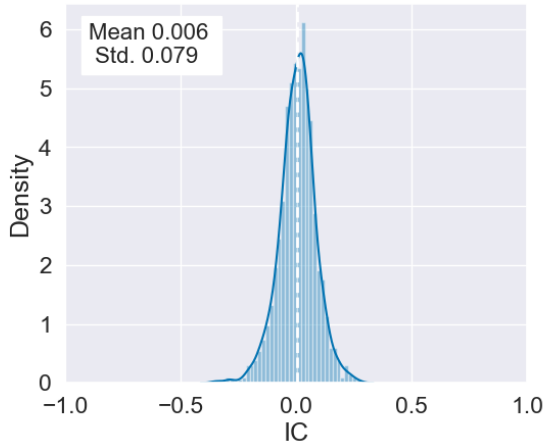
5D Period IC



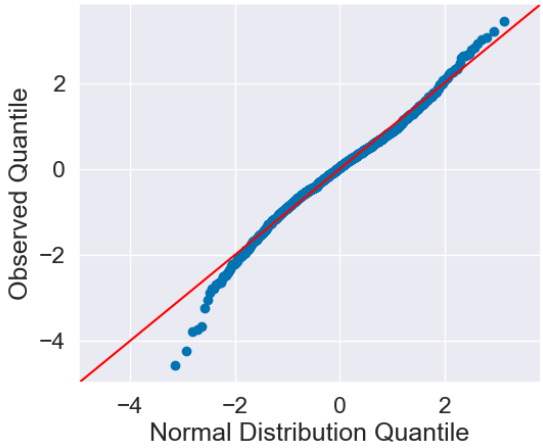
5D Period IC Normal Dist. Q-Q



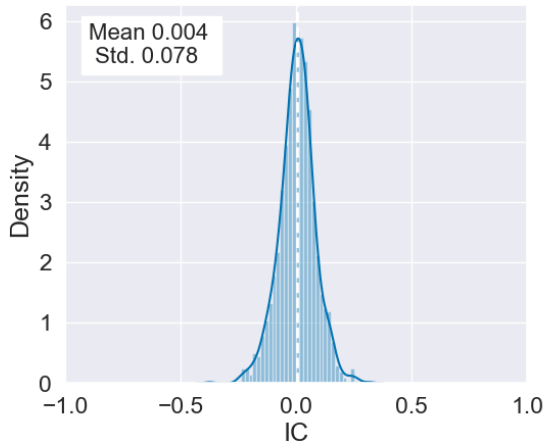
10D Period IC



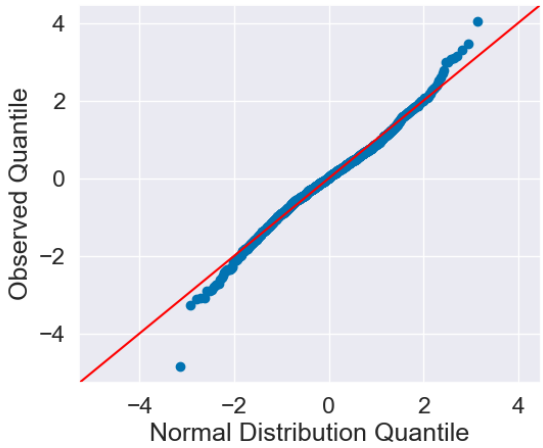
10D Period IC Normal Dist. Q-Q



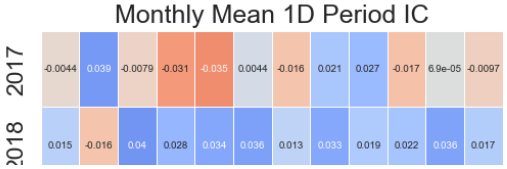
20D Period IC



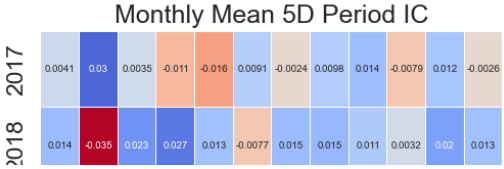
20D Period IC Normal Dist. Q-Q



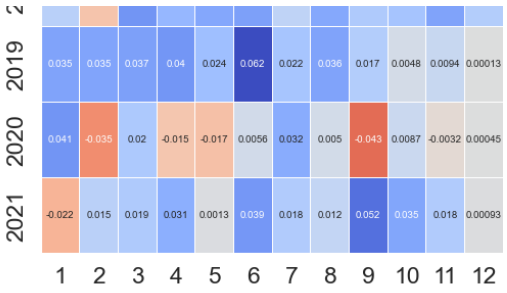
Monthly Mean 1D Period IC

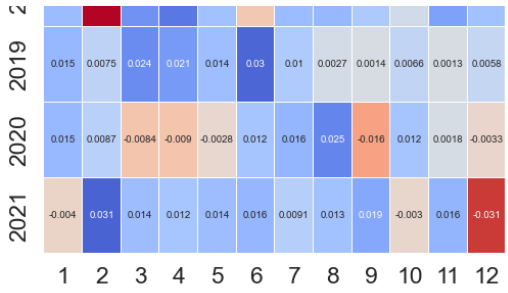


Monthly Mean 5D Period IC

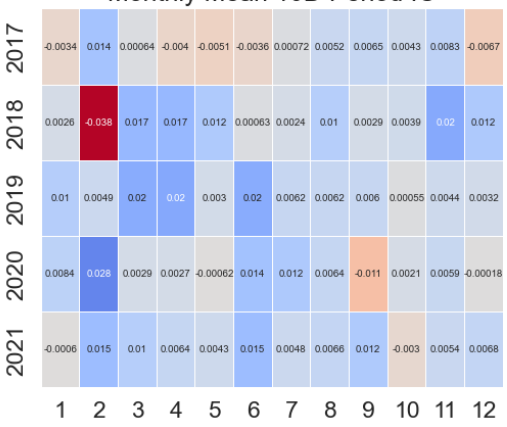


h_l_delta_fct

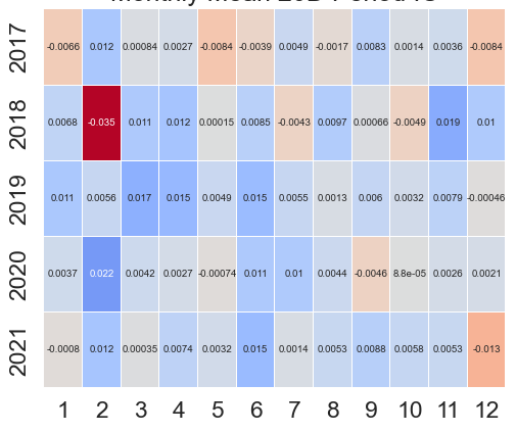




Monthly Mean 10D Period IC



Monthly Mean 20D Period IC



2.3.4 factor c_l_delay_fct

The factor expression is:
ts_delta(close, 5)

This factor reflects the increase or decrease of the current closing price compared to the closing price five days ago. It is one of the most typical momentum factors. The closing market of the momentum factor has a greater impact. If the market reverses, the performance will become worse. In the momentum market, despite the lag, as a single factor, it still performs well.

The performance and detailed analysis of this factor are shown on the next pages:

Quantiles Statistics

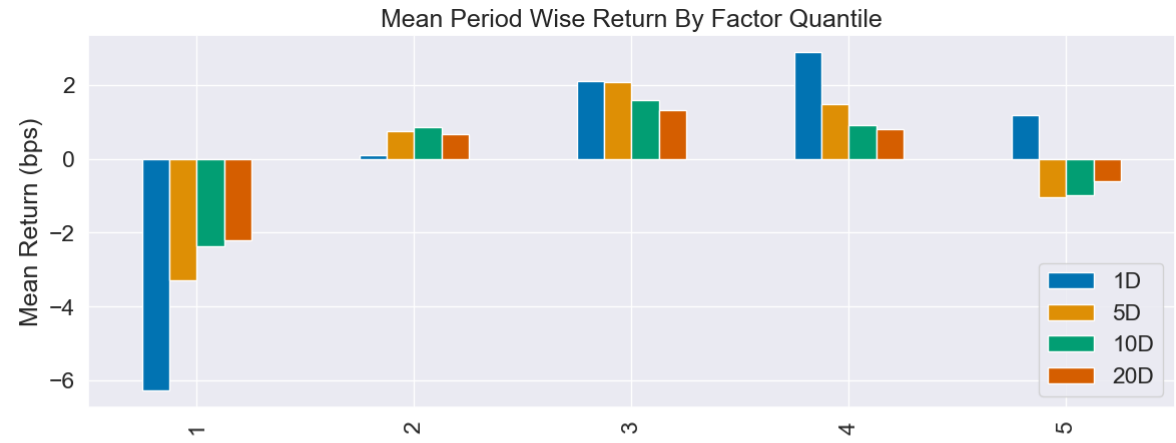
factor_quantile	min	max	mean	std	count	count %
1	-13151.90006	2.28150	-10.694709	70.777299	867239	20.013717
2	-9.95616	4.20257	-0.734818	1.209919	866740	20.002202
3	-4.59684	6.33888	0.153367	1.028234	867126	20.011110
4	-2.50950	11.29024	1.033133	1.451704	865541	19.974532
5	-1.26840	14918.83092	10.308941	86.428337	866577	19.998440

Returns Analysis

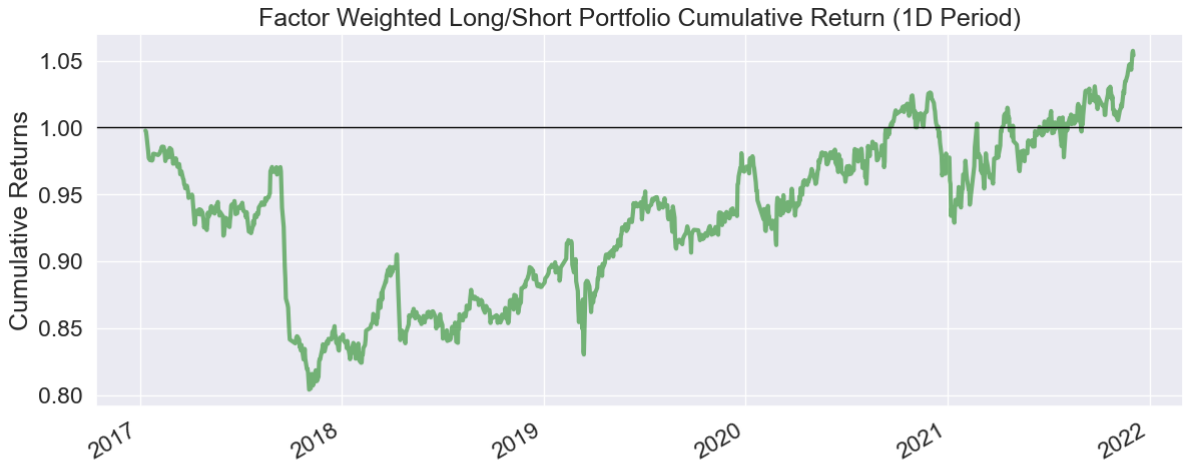
	1D	5D	10D	20D
Ann. alpha	0.014	-0.039	-0.031	-0.003
beta	0.036	0.037	0.020	0.071
Mean Period Wise Return Top Quantile (bps)	1.178	-1.037	-0.996	-0.612
Mean Period Wise Return Bottom Quantile (bps)	-6.281	-3.285	-2.357	-2.193
Mean Period Wise Spread (bps)	7.459	2.290	1.404	1.623

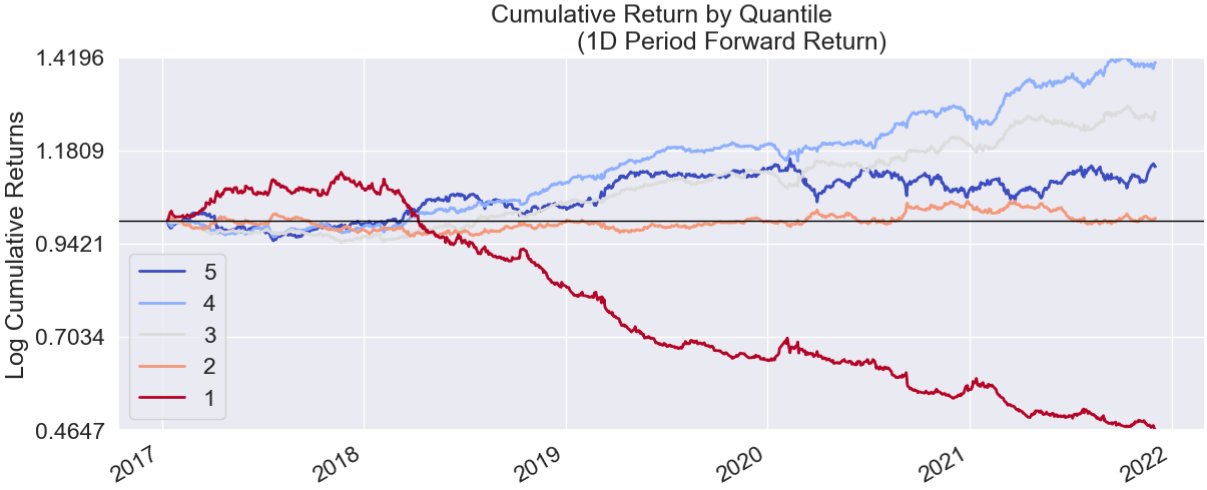
<Figure size 640x480 with 0 Axes>

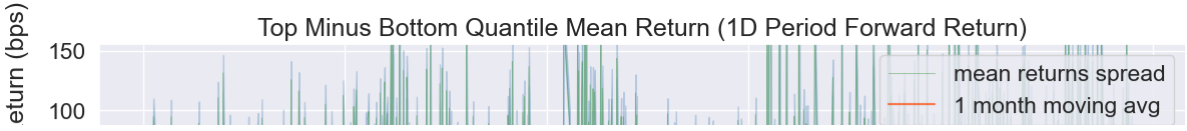
c_l_delay_fct

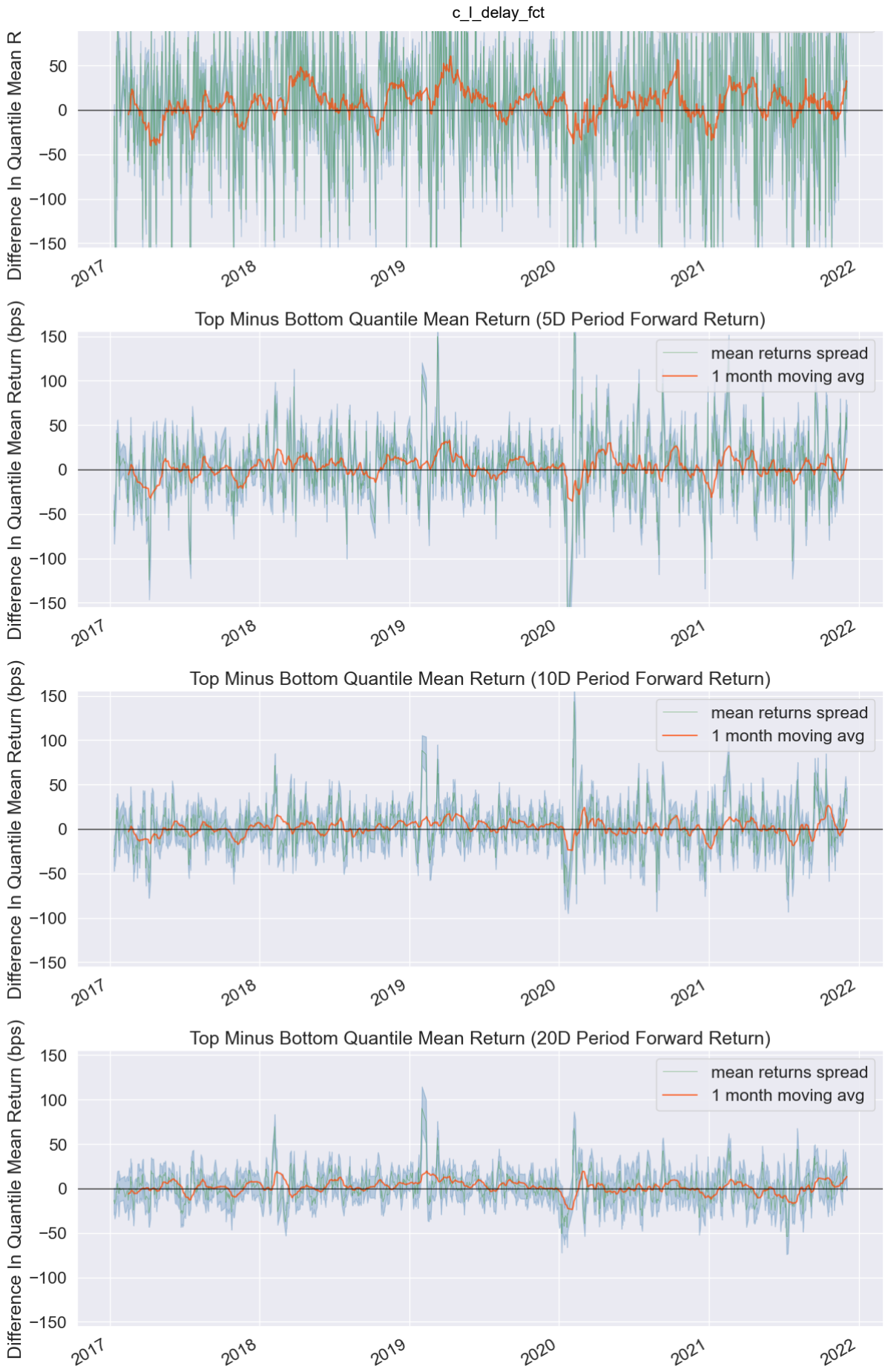

Mean Period Wise Return By Factor Quantile
Mean Return (bps)
2
0
−2
−4
−6
1
2
3
4
5
1D
5D
10D
20D


Period Wise Return By Factor Quantile
Return (bps)
60
40
20
0
−20
−40
−60
1
2
3
4
5
forward_periods
1D
5D
10D
20D


Factor Weighted Long/Short Portfolio Cumulative Return (1D Period)
Cumulative Returns
1.05
1.00
0.95
0.90
0.85
0.80
2017
2018
2019
2020
2021
2022

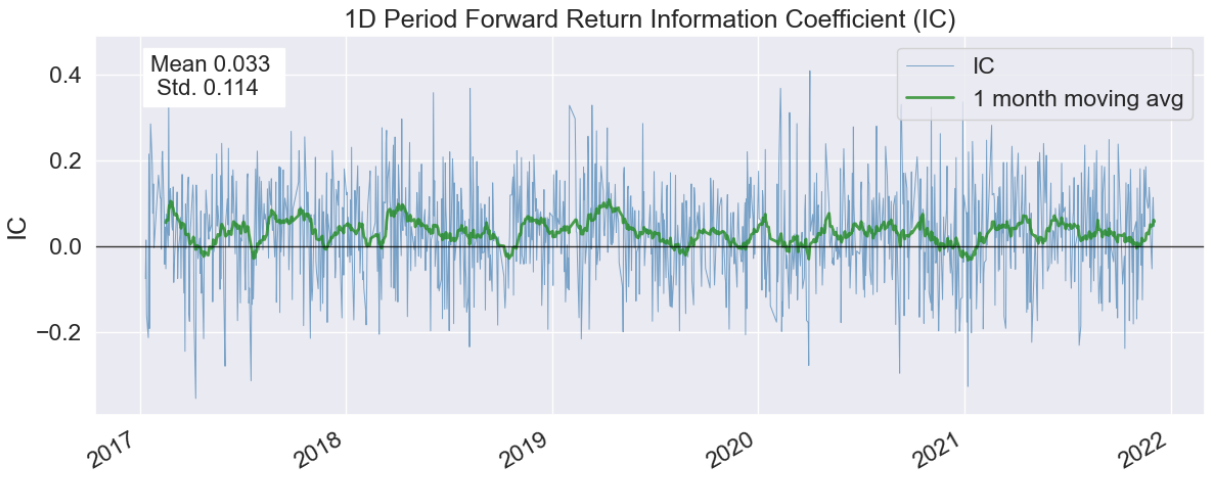

Cumulative Return by Quantile
(1D Period Forward Return)
Log Cumulative Returns
1.4196
1.1809
0.9421
0.7034
0.4647
5
4
3
2
1
2017
2018
2019
2020
2021
2022

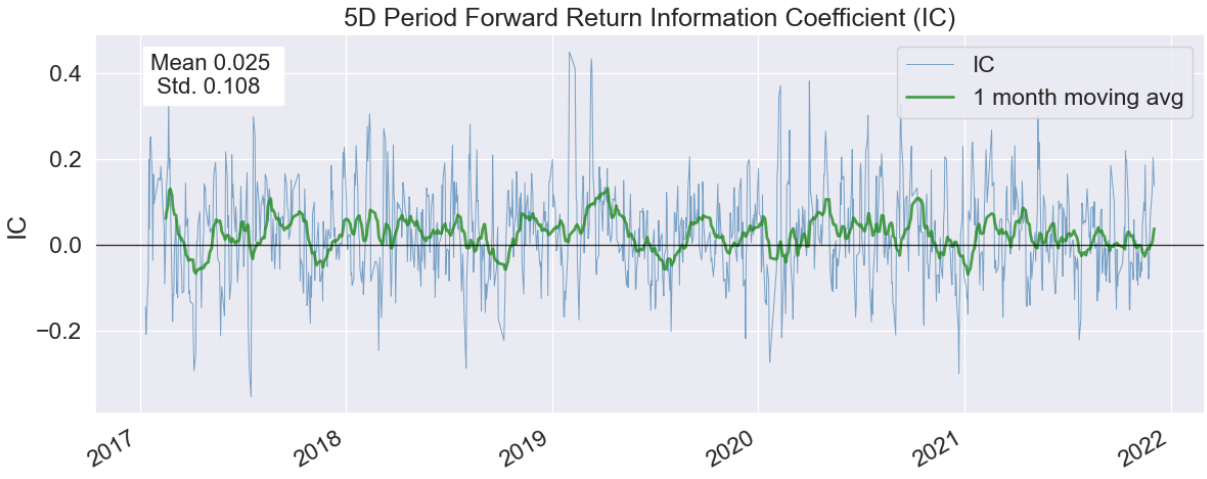

Top Minus Bottom Quantile Mean Return (1D Period Forward Return)
eturn (bps)
150
100
mean returns spread
1 month moving avg

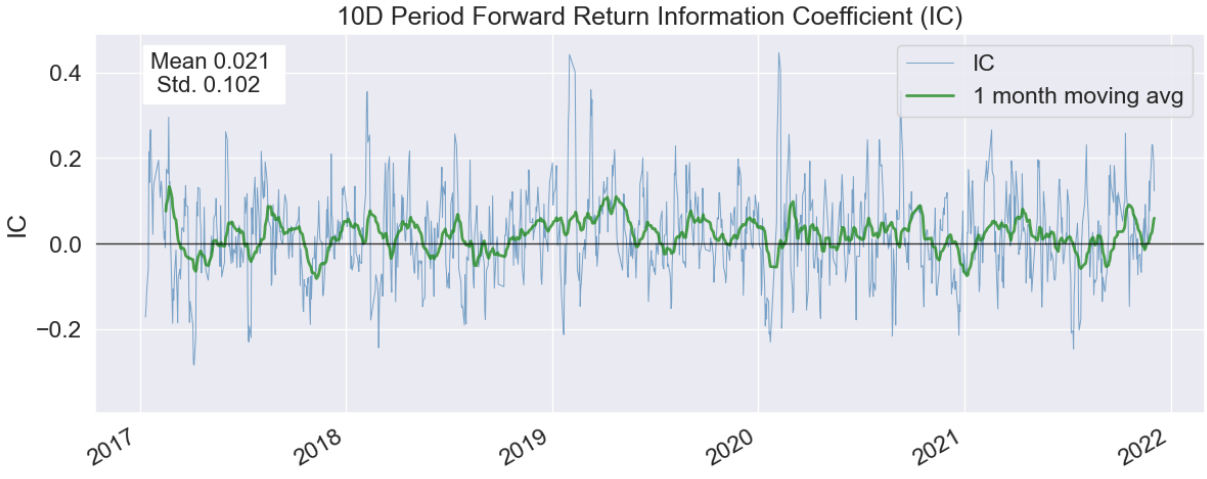


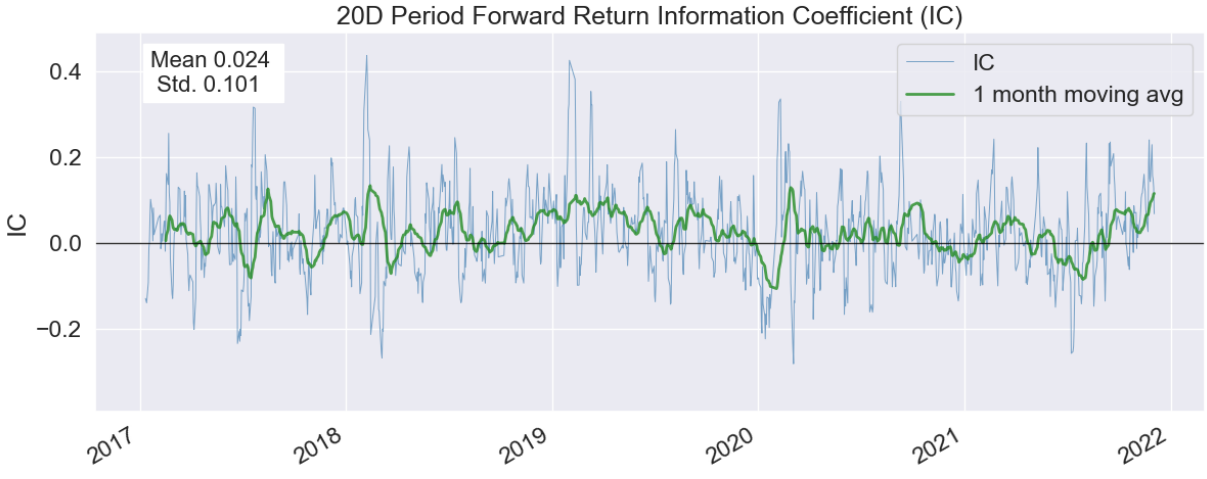
Information Analysis

	1D	5D	10D	20D
IC Mean	0.033	0.025	0.021	0.024
IC Std.	0.114	0.108	0.102	0.101
Risk-Adjusted IC	0.290	0.236	0.205	0.240
t-stat(IC)	10.008	8.136	7.066	8.294
p-value(IC)	0.000	0.000	0.000	0.000
IC Skew	-0.069	0.224	0.368	0.335
IC Kurtosis	0.026	0.704	0.868	1.032

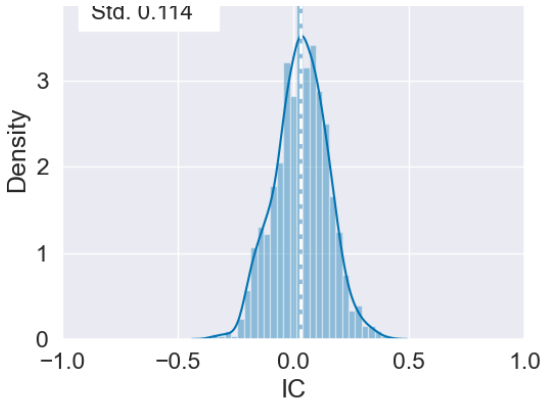

1D Period Forward Return Information Coefficient (IC)
Mean 0.033
Std. 0.114
IC
1 month moving avg
0.4
0.2
0.0
−0.2
IC
2017
2018
2019
2020
2021
2022

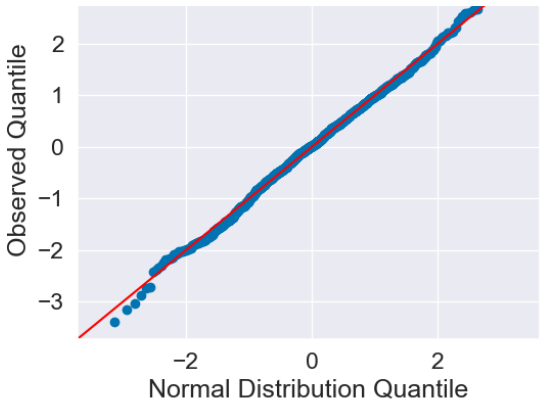

5D Period Forward Return Information Coefficient (IC)
Mean 0.025
Std. 0.108
IC
1 month moving avg
0.4
0.2
0.0
−0.2
IC
2017
2018
2019
2020
2021
2022


10D Period Forward Return Information Coefficient (IC)
Mean 0.021
Std. 0.102
IC
1 month moving avg
0.4
0.2
0.0
−0.2
IC
2017
2018
2019
2020
2021
2022

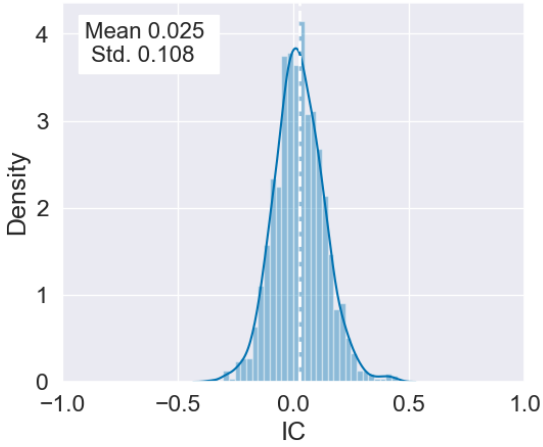

20D Period Forward Return Information Coefficient (IC)
Mean 0.024
Std. 0.101
IC
1 month moving avg
0.4
0.2
0.0
−0.2
IC
2017
2018
2019
2020
2021
2022


1D Period IC
Mean 0.033
4
1D Period IC Normal Dist. Q-Q
3

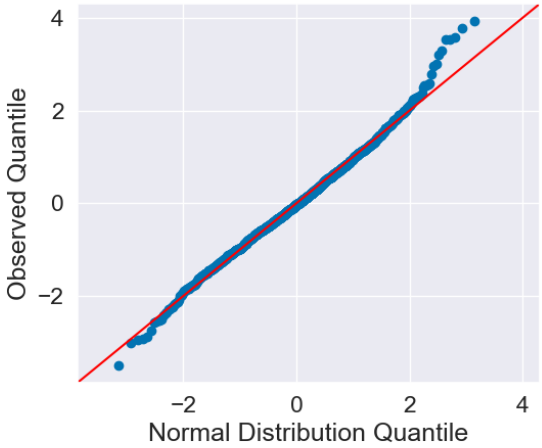




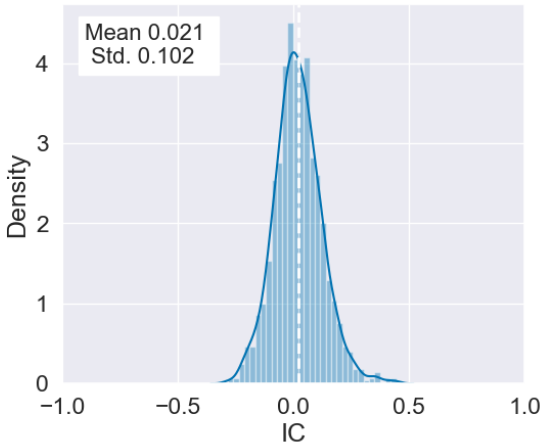
5D Period IC



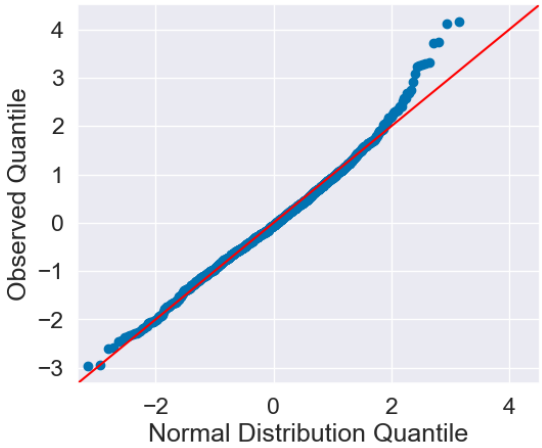
5D Period IC Normal Dist. Q-Q



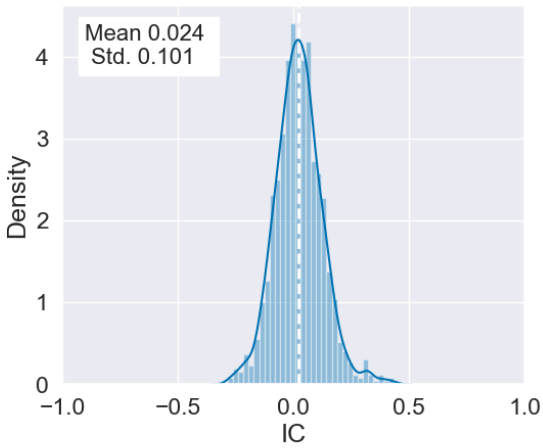
10D Period IC



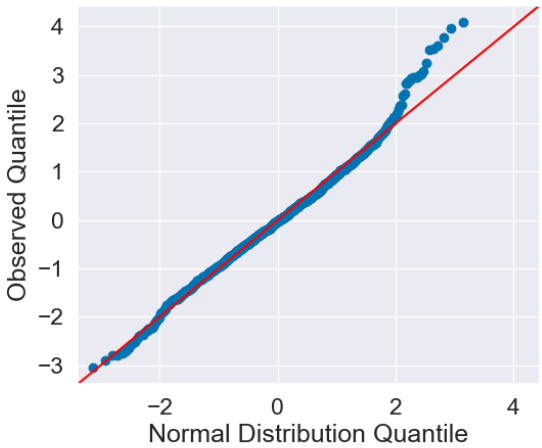
10D Period IC Normal Dist. Q-Q



20D Period IC

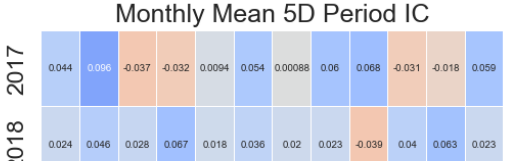


20D Period IC Normal Dist. Q-Q

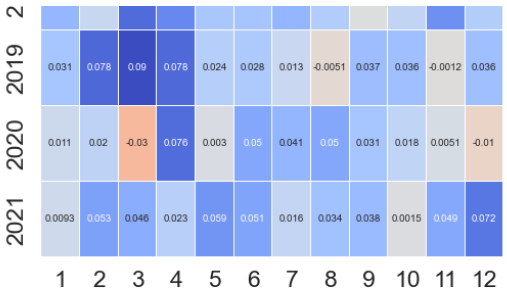


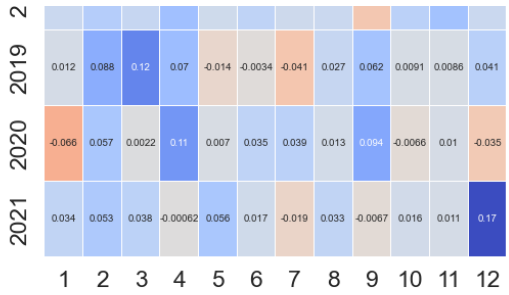
Monthly Mean 1D Period IC

Monthly Mean 5D Period IC

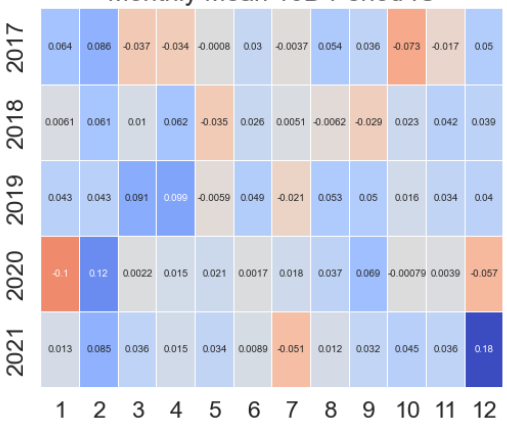


c_l_delay_fct

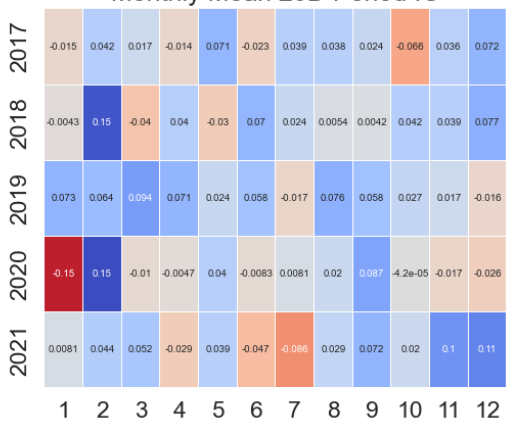




Monthly Mean 10D Period IC



Monthly Mean 20D Period IC



2.3.5 factor a_s_div_fct

The factor expression is:
assets/sales*ts_mean(vol)/ts_std(vol)

This factor consists of two parts. First, assets divided by revenue can be used to express the company's asset conversion rate. At the same time, because the data is updated slowly, a time-sensitive volatility inverse is added. The higher the company's asset conversion rate and the less volatile the stock trading volume, the higher the value of the factor. In fact, this factor itself is also a combination of the volatility relationship, and gains are obtained by screening stocks with lower volatility.

The performance and detailed analysis of this factor are shown on the next pages:

Quantiles Statistics

factor_quantile	min	max	mean	std	count	count %
1	-989426.929689	38.632162	-802.996074	10157.379914	834350	20.011705
2	-148.905952	94.685378	35.348176	24.383827	833621	19.994220
3	30.588444	173.874411	74.569691	31.096906	833648	19.994867
4	49.891799	412.145928	144.528627	67.942014	833621	19.994220
5	98.162662	758165.732041	1240.642558	8927.030536	834070	20.004989

Returns Analysis

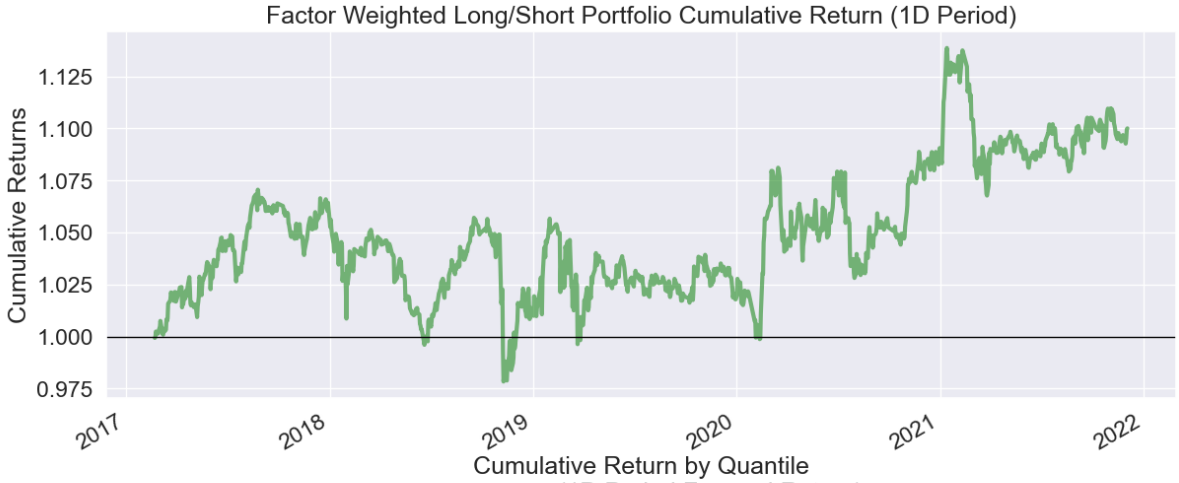
	1D	5D	10D	20D
Ann. alpha	0.023	0.026	0.026	0.040
beta	0.006	0.001	-0.030	-0.041
Mean Period Wise Return Top Quantile (bps)	0.134	0.201	0.127	-0.027
Mean Period Wise Return Bottom Quantile (bps)	-1.829	-1.369	-1.093	-0.718
Mean Period Wise Spread (bps)	1.963	1.568	1.215	0.687

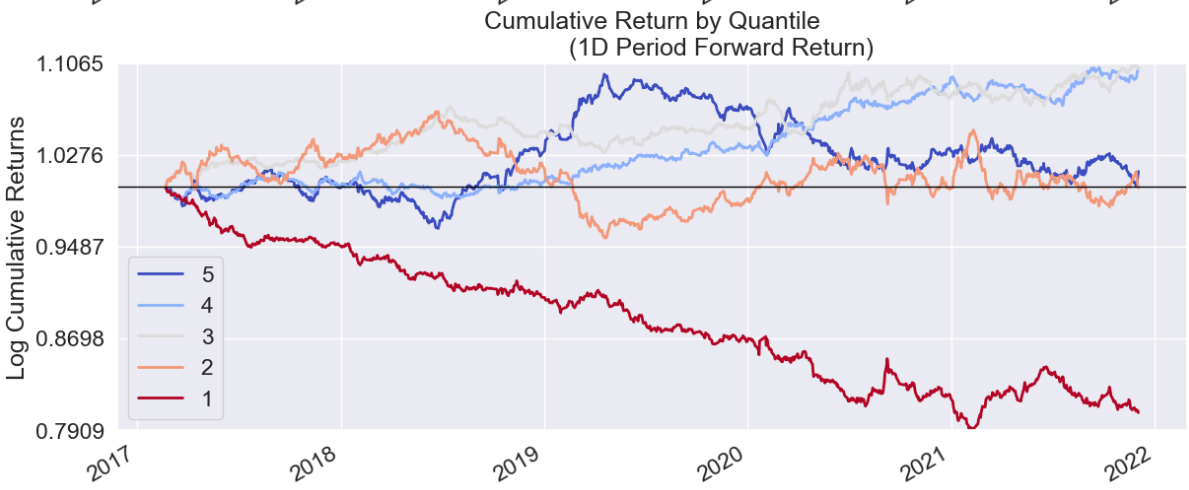
<Figure size 640x480 with 0 Axes>

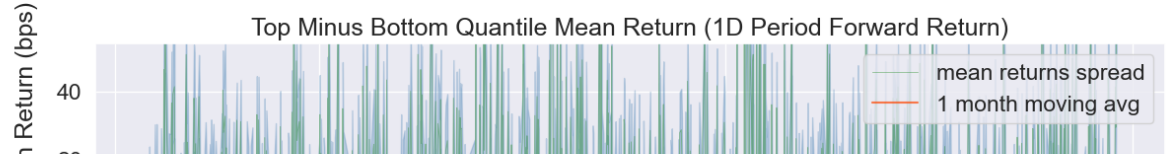
a_s_div_fct

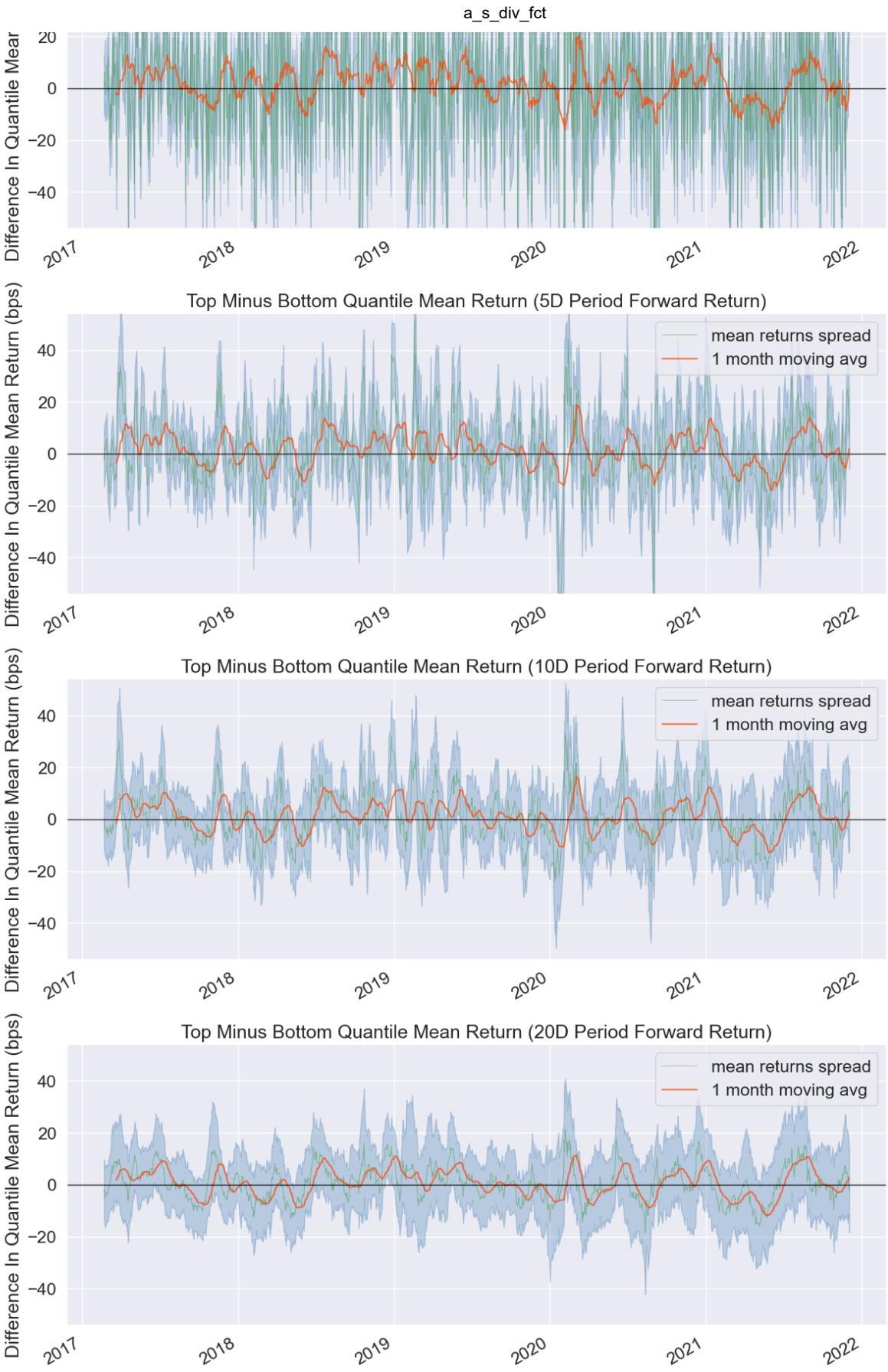

Mean Period Wise Return By Factor Quantile
Mean Return (bps)
0.5
0.0
−0.5
−1.0
−1.5
1D
5D
10D
20D
1
2
3
4
5


Period Wise Return By Factor Quantile
Return (bps)
30
20
10
0
−10
−20
−30
forward_periods
1D
5D
10D
20D
1
2
3
4
5


Factor Weighted Long/Short Portfolio Cumulative Return (1D Period)
Cumulative Returns
1.125
1.100
1.075
1.050
1.025
1.000
0.975
2017
2018
2019
2020
2021
2022

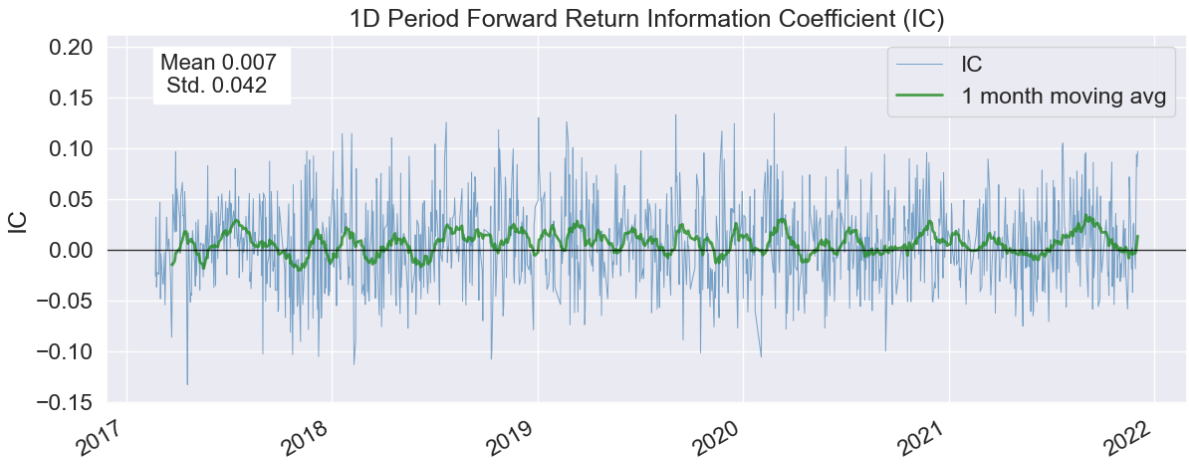

Cumulative Return by Quantile
(1D Period Forward Return)
Log Cumulative Returns
1.1065
1.0276
0.9487
0.8698
0.7909
5
4
3
2
1
2017
2018
2019
2020
2021
2022

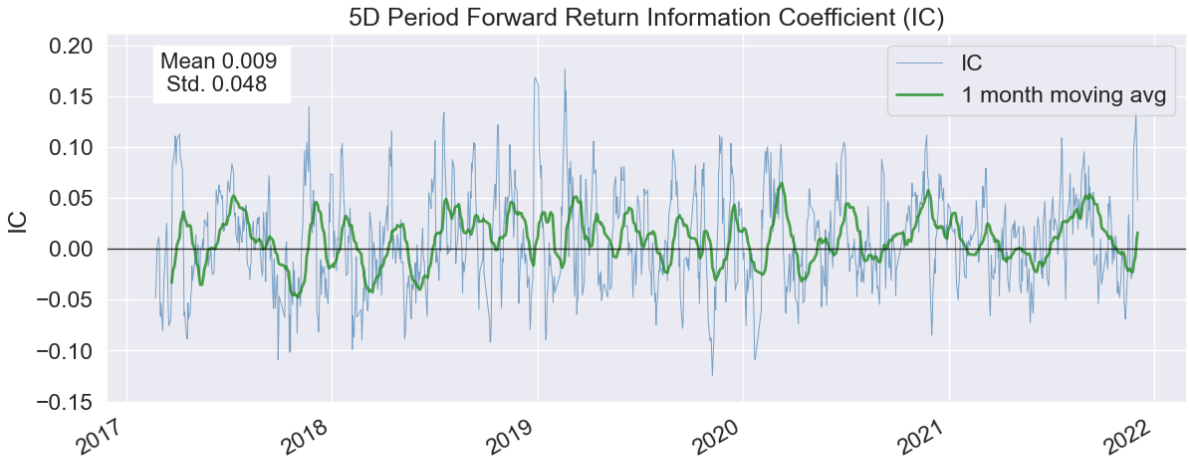

Top Minus Bottom Quantile Mean Return (1D Period Forward Return)
Return (bps)
40
mean returns spread
1 month moving avg

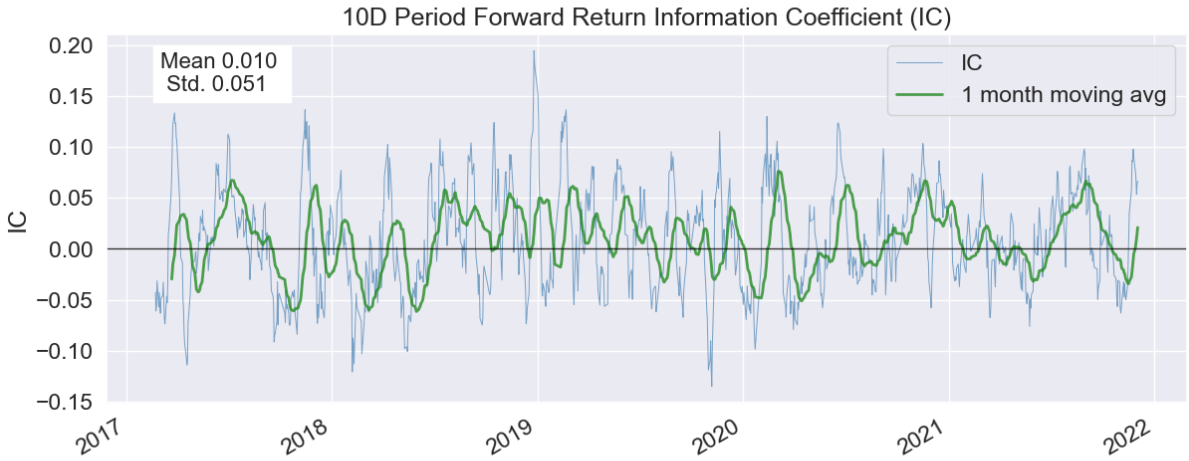


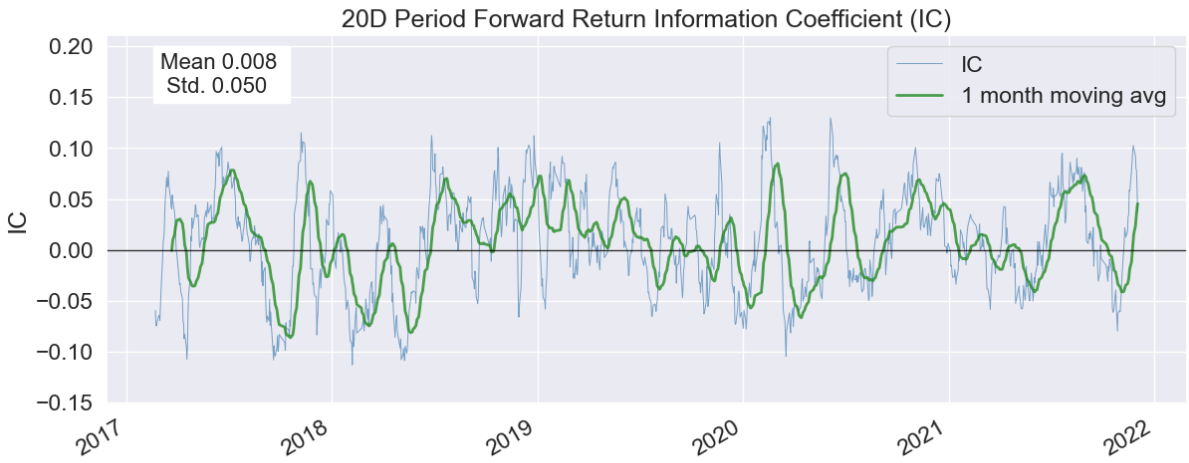
Information Analysis

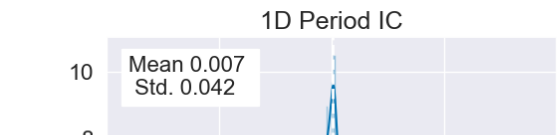
	1D	5D	10D	20D
IC Mean	0.007	0.009	0.010	0.008
IC Std.	0.042	0.048	0.051	0.050
Risk-Adjusted IC	0.163	0.198	0.188	0.166
t-stat(IC)	5.563	6.766	6.421	5.661
p-value(IC)	0.000	0.000	0.000	0.000
IC Skew	0.143	0.282	0.219	-0.047
IC Kurtosis	-0.012	-0.021	-0.233	-0.642

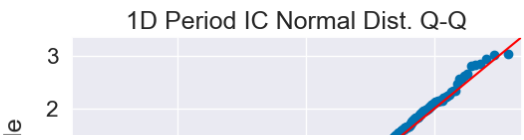

1D Period Forward Return Information Coefficient (IC)
Mean 0.007
Std. 0.042
IC
1 month moving avg
IC
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
2017
2018
2019
2020
2021
2022


5D Period Forward Return Information Coefficient (IC)
Mean 0.009
Std. 0.048
IC
1 month moving avg
IC
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
2017
2018
2019
2020
2021
2022

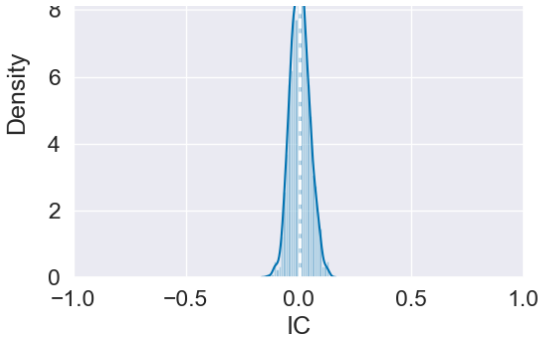

10D Period Forward Return Information Coefficient (IC)
Mean 0.010
Std. 0.051
IC
1 month moving avg
IC
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
2017
2018
2019
2020
2021
2022

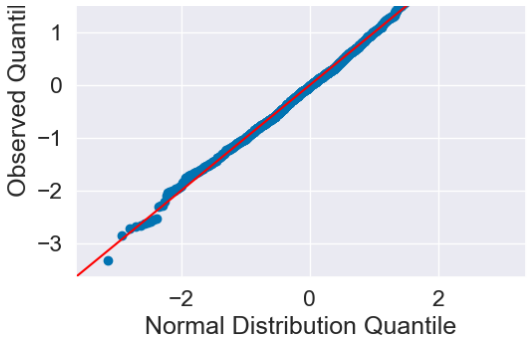

20D Period Forward Return Information Coefficient (IC)
Mean 0.008
Std. 0.050
IC
1 month moving avg
IC
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
2017
2018
2019
2020
2021
2022


1D Period IC
Mean 0.007
Std. 0.042
10

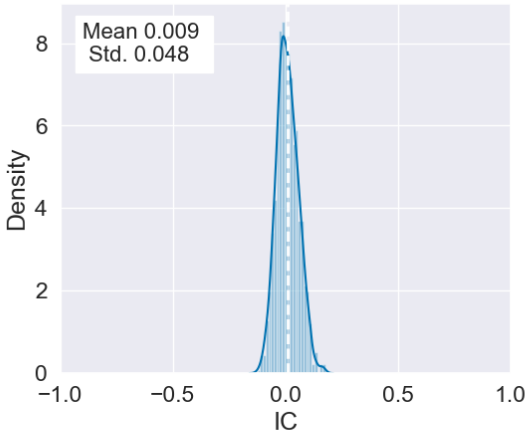

1D Period IC Normal Dist. Q-Q
3
2

a_s_div_fct

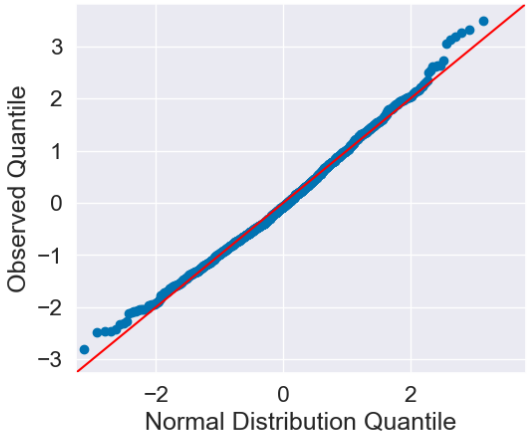




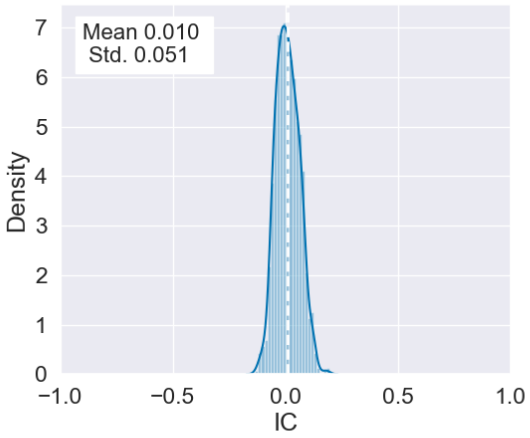
5D Period IC



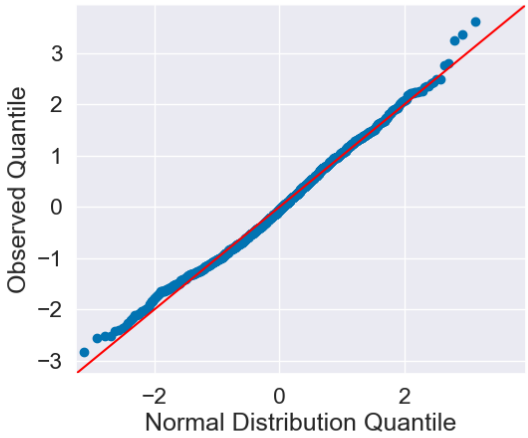
5D Period IC Normal Dist. Q-Q



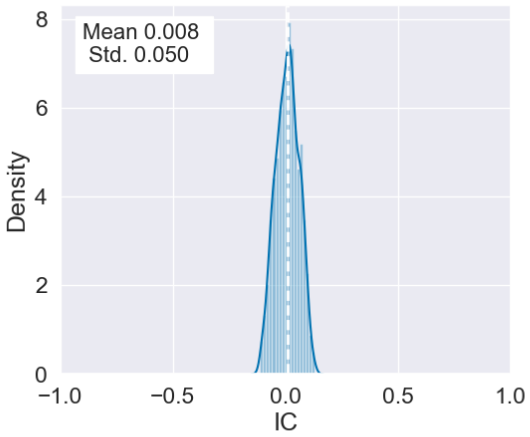
10D Period IC



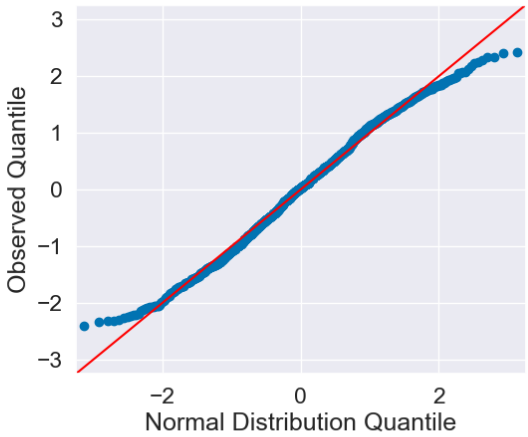
10D Period IC Normal Dist. Q-Q



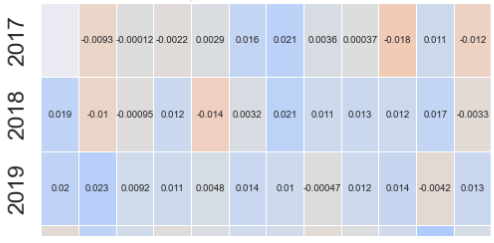
20D Period IC



20D Period IC Normal Dist. Q-Q

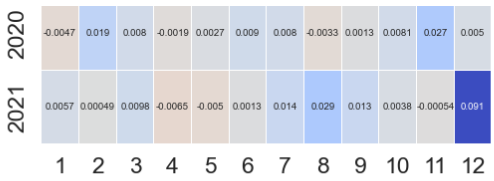


Monthly Mean 1D Period IC

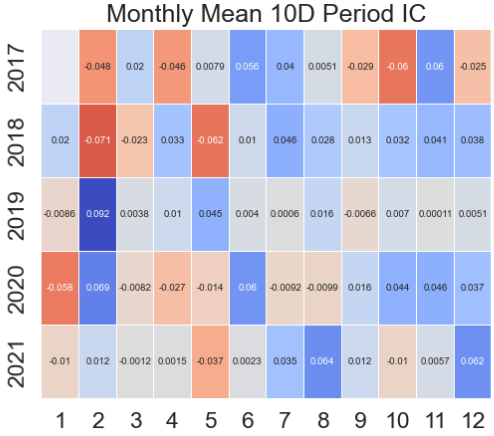


Monthly Mean 5D Period IC

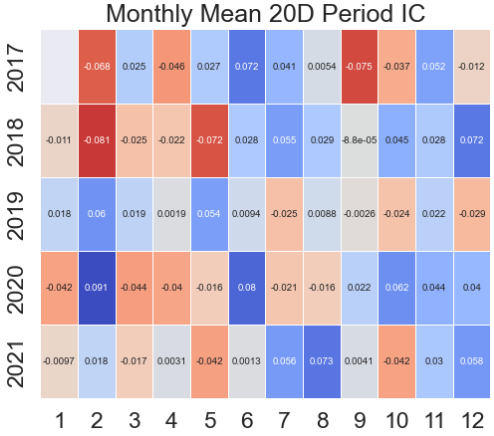
a_s_div_fct



Monthly Mean 10D Period IC



Monthly Mean 20D Period IC



2.3.6 factor k_u_ret_fct

The factor expression is:
open/pre_close - 1

This factor directly represents the increase or decrease between the opening price and yesterday's closing price, thereby predicting the future price trend. It is a momentum factor that changes quickly. Although the data used are the opening price and the closing price of the previous day, theoretically, this factor can also be used for day trading, but during the analysis process, I still lagged the factor by one day, but its performance is still better.

The performance and detailed analysis of this factor are shown on the next pages:

Quantiles Statistics

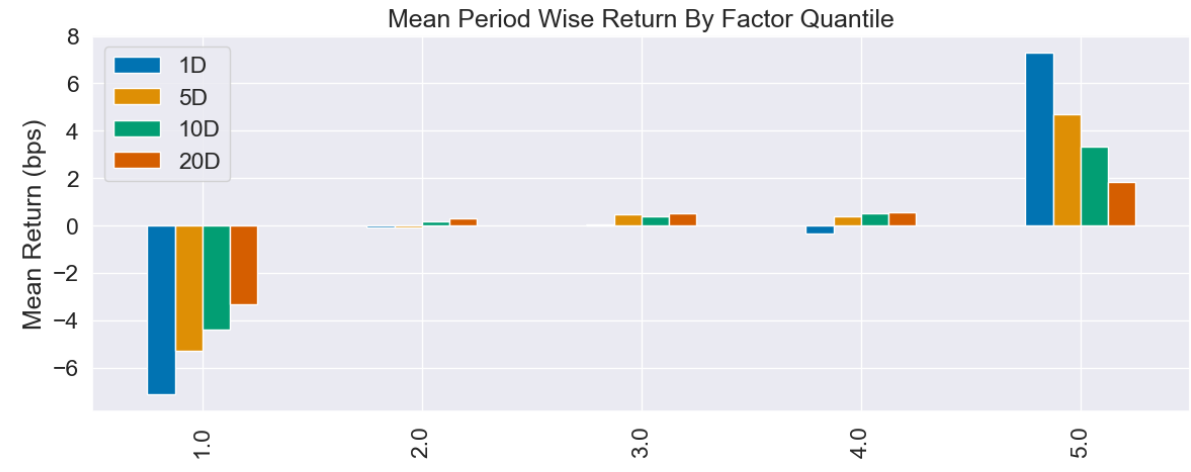
	min	max	mean	std	count	count %
factor_quantile						
1.0	-0.323051	0.014641	-0.013814	0.014314	879521	20.307991
2.0	-0.100196	0.019608	-0.003736	0.006523	948790	21.907400
3.0	-0.100000	0.024057	-0.000911	0.005690	910373	21.020358
4.0	-0.099775	0.030137	0.001967	0.006048	732751	16.919096
5.0	-0.098876	3.436652	0.012962	0.018314	859476	19.845155

Returns Analysis

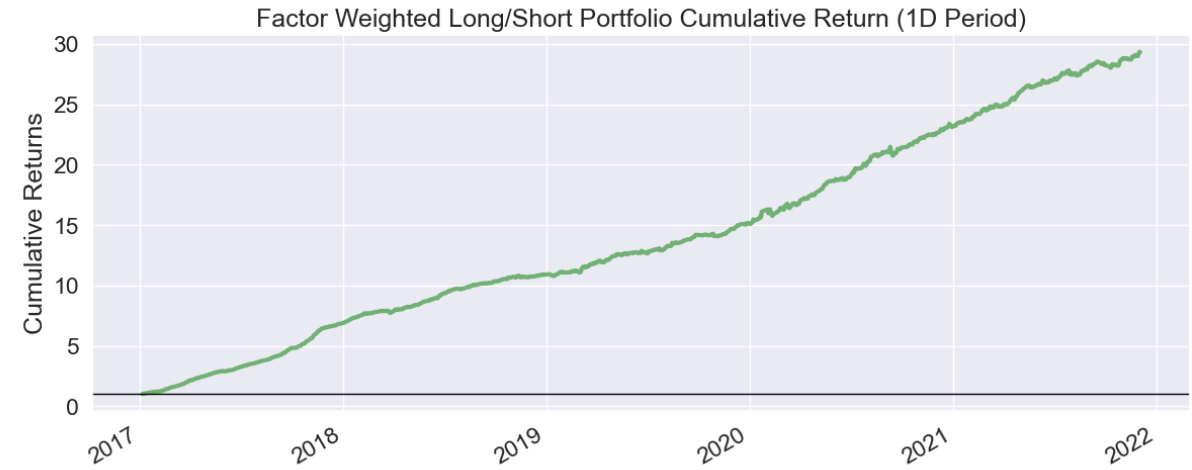
	1D	5D	10D	20D
Ann. alpha	1.062	0.720	0.479	0.240
beta	-0.054	-0.031	-0.016	-0.007
Mean Period Wise Return Top Quantile (bps)	7.308	4.690	3.349	1.859
Mean Period Wise Return Bottom Quantile (bps)	-7.127	-5.261	-4.383	-3.300
Mean Period Wise Spread (bps)	14.435	9.947	7.731	5.160

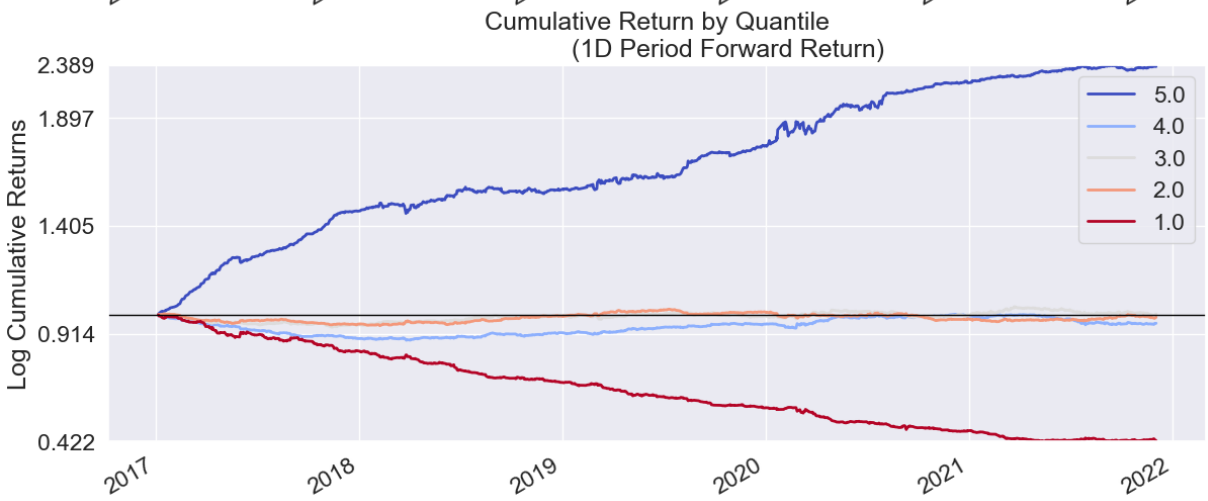
<Figure size 640x480 with 0 Axes>

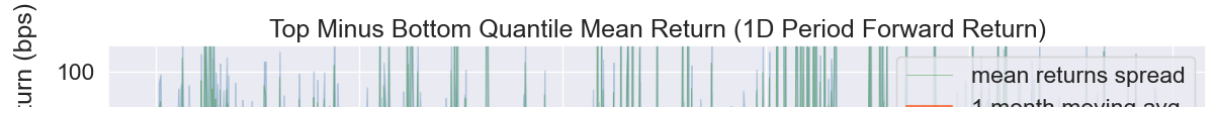
k_u_ret_fct

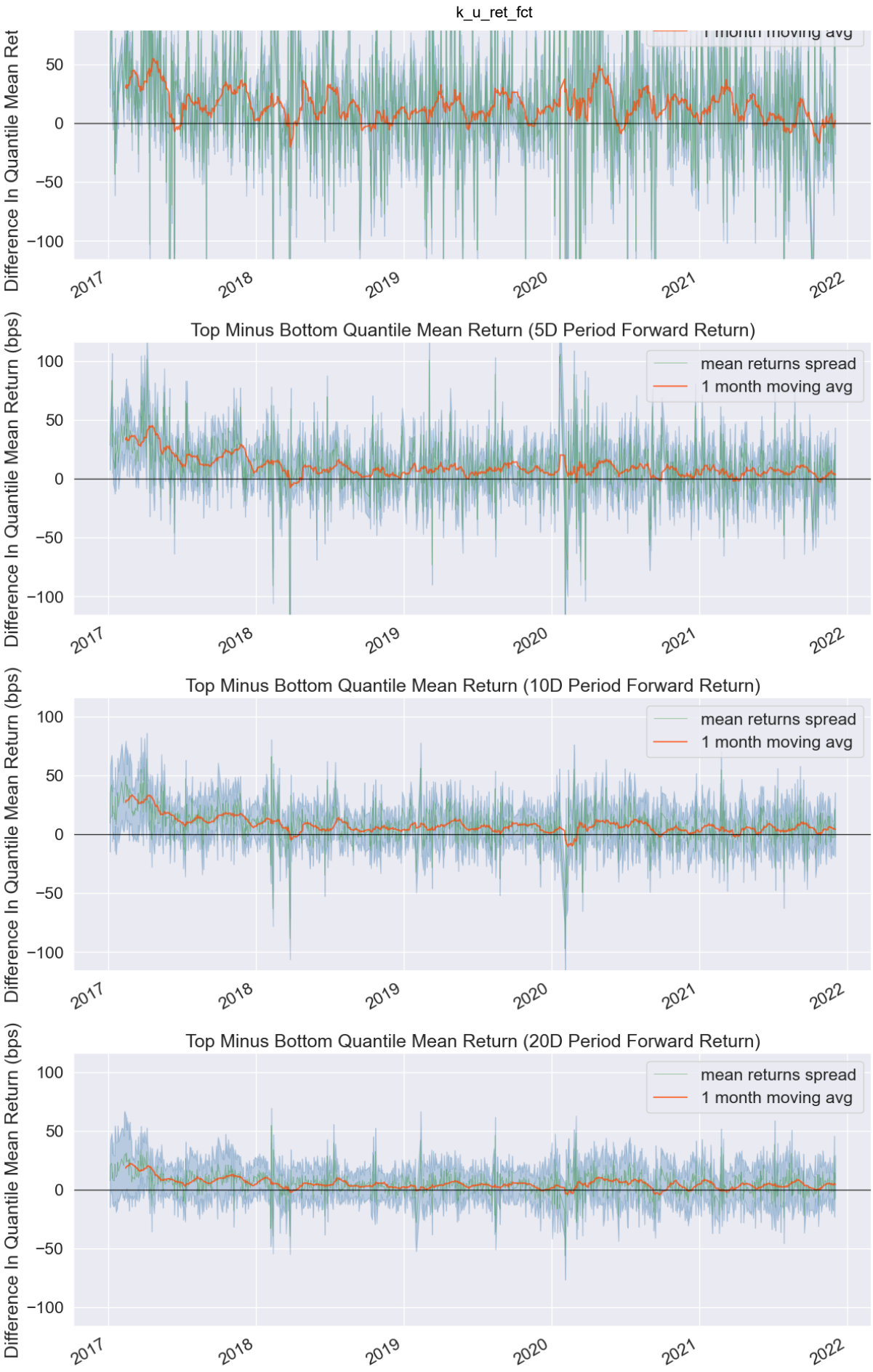

Mean Period Wise Return By Factor Quantile
Mean Return (bps)
1D
5D
10D
20D
1.0
2.0
3.0
4.0
5.0


Period Wise Return By Factor Quantile
Return (bps)
forward_periods
1D
5D
10D
20D
1.0
2.0
3.0
4.0
5.0


Factor Weighted Long/Short Portfolio Cumulative Return (1D Period)
Cumulative Returns
2017
2018
2019
2020
2021
2022

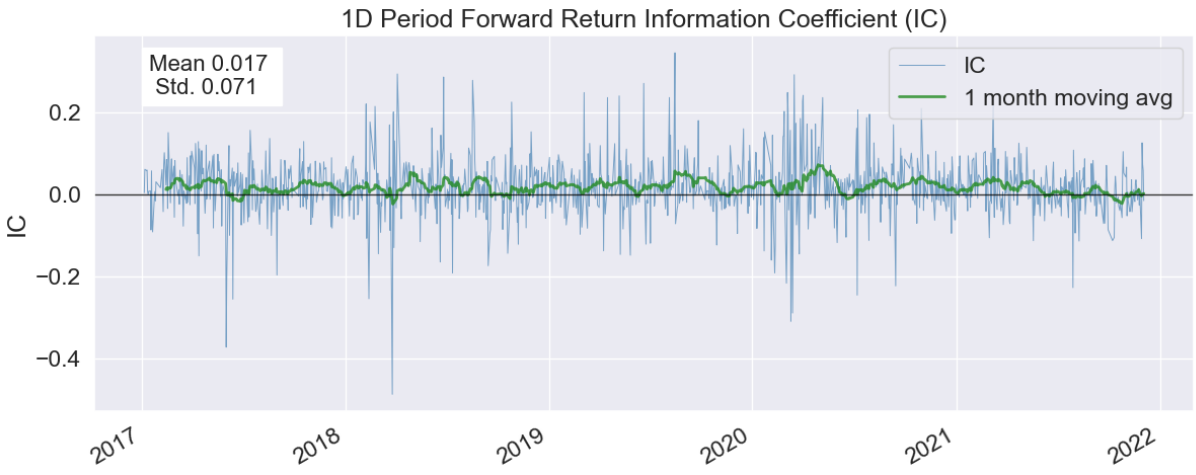

Cumulative Return by Quantile
(1D Period Forward Return)
Log Cumulative Returns
2.389
1.897
1.405
0.914
0.422
5.0
4.0
3.0
2.0
1.0
2017
2018
2019
2020
2021
2022

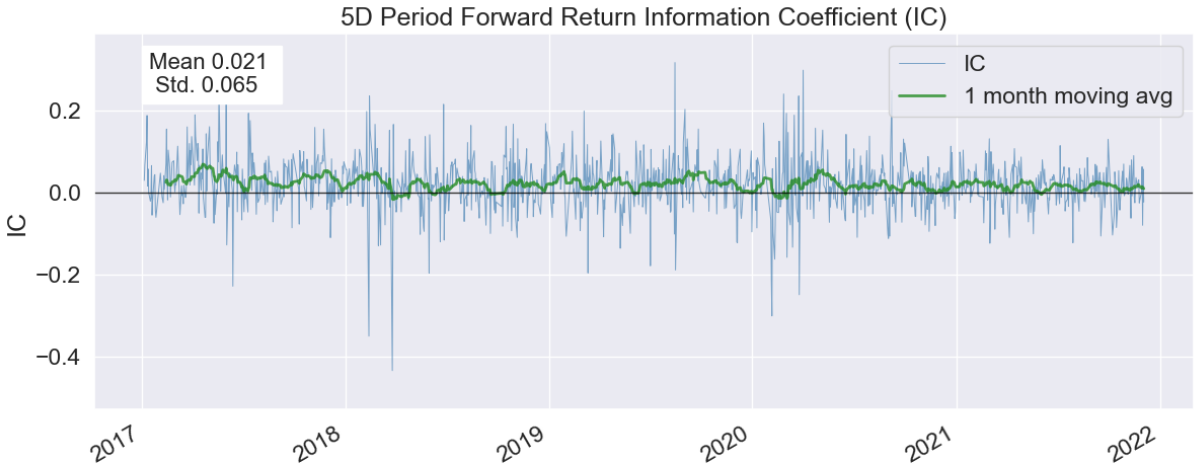

Top Minus Bottom Quantile Mean Return (1D Period Forward Return)
100
mean returns spread

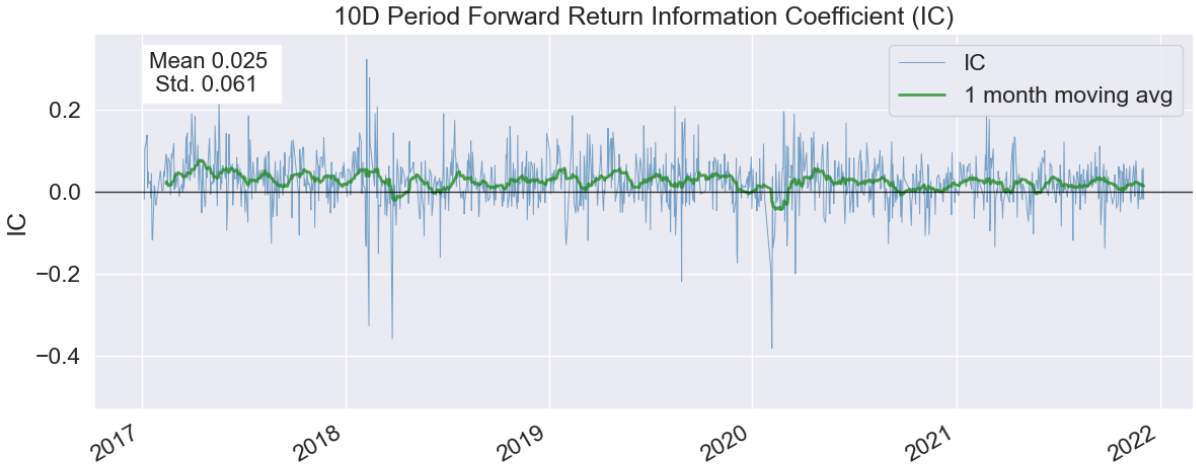


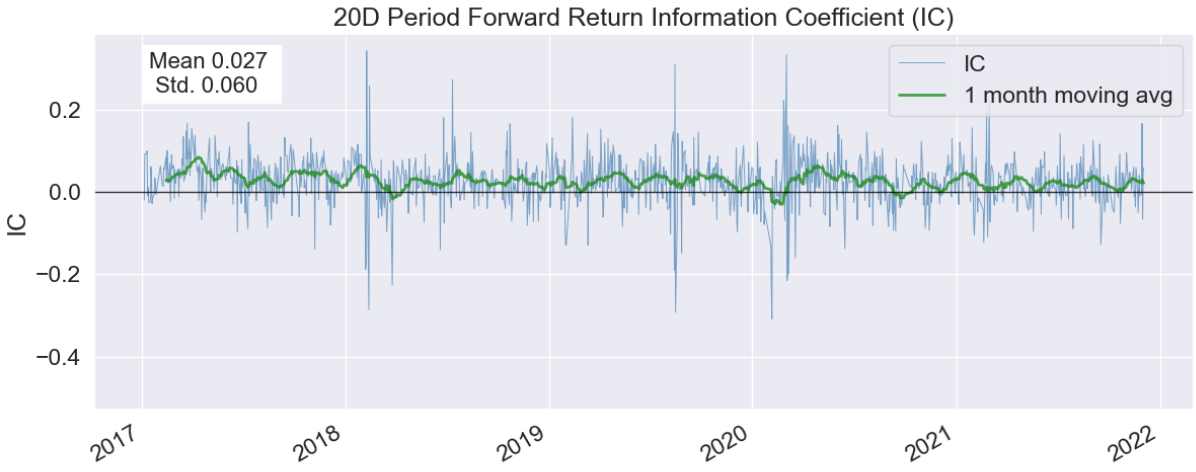
Information Analysis

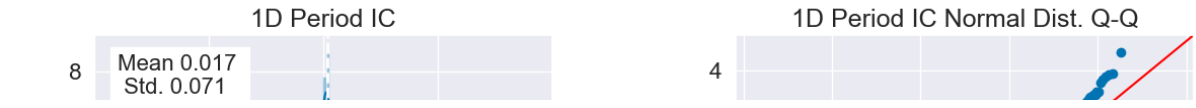
	1D	5D	10D	20D
IC Mean	0.017	0.021	0.025	0.027
IC Std.	0.071	0.065	0.061	0.060
Risk-Adjusted IC	0.247	0.324	0.408	0.453
t-stat(IC)	8.511	11.167	14.054	15.631
p-value(IC)	0.000	0.000	0.000	0.000
IC Skew	-0.285	-0.365	-0.585	-0.342
IC Kurtosis	5.512	4.878	5.471	4.972

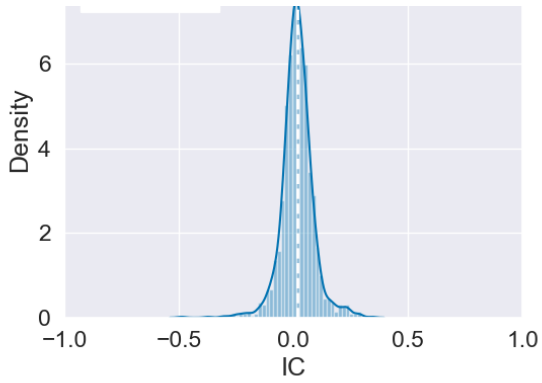

1D Period Forward Return Information Coefficient (IC)
Mean 0.017
Std. 0.071
IC
1 month moving avg
0.2
0.0
−0.2
−0.4
IC
2017
2018
2019
2020
2021
2022

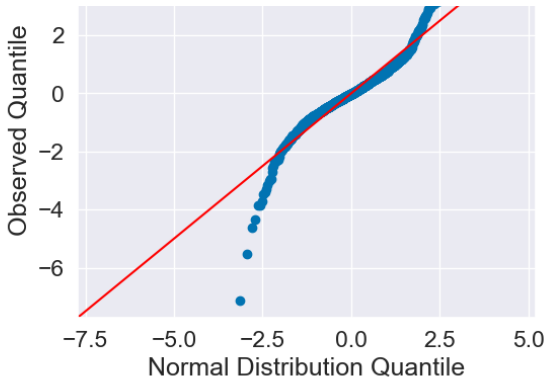

5D Period Forward Return Information Coefficient (IC)
Mean 0.021
Std. 0.065
IC
1 month moving avg
0.2
0.0
−0.2
−0.4
IC
2017
2018
2019
2020
2021
2022

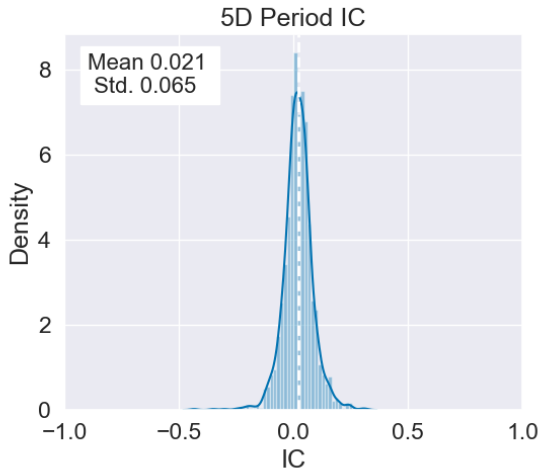

10D Period Forward Return Information Coefficient (IC)
Mean 0.025
Std. 0.061
IC
1 month moving avg
0.2
0.0
−0.2
−0.4
IC
2017
2018
2019
2020
2021
2022

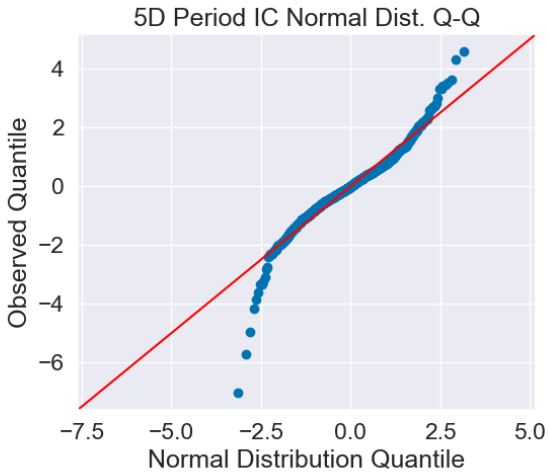

20D Period Forward Return Information Coefficient (IC)
Mean 0.027
Std. 0.060
IC
1 month moving avg
0.2
0.0
−0.2
−0.4
IC
2017
2018
2019
2020
2021
2022

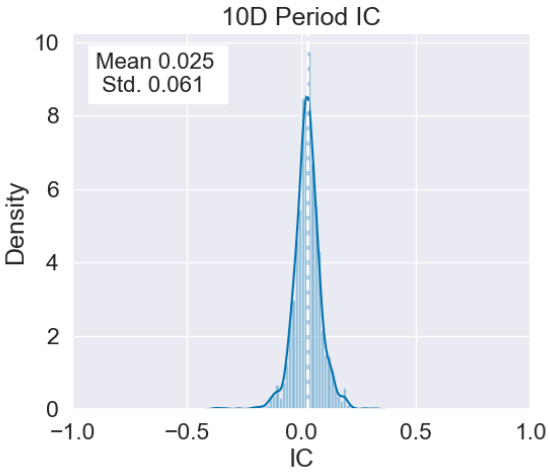

1D Period IC
Mean 0.017
Std. 0.071
8
1D Period IC Normal Dist. Q-Q
4

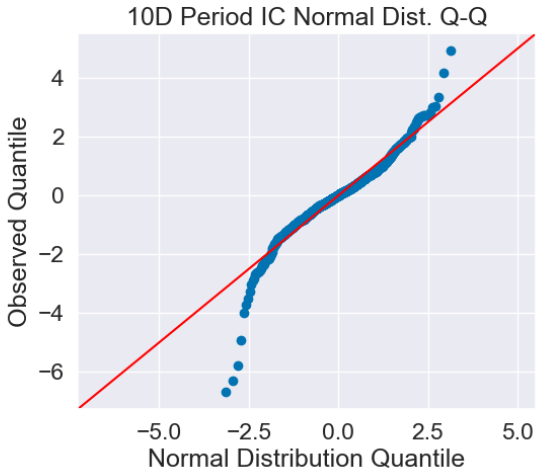

Density
IC

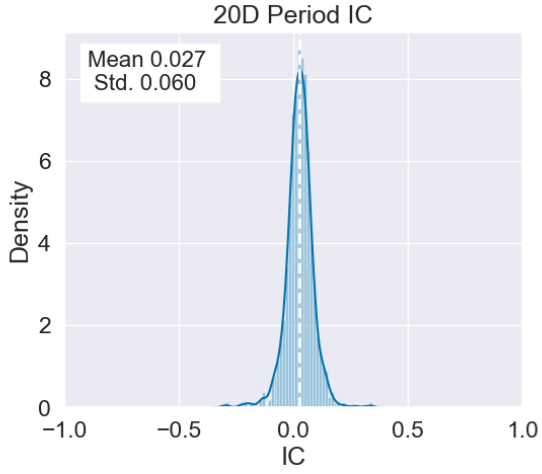

Observed Quantile
Normal Distribution Quantile

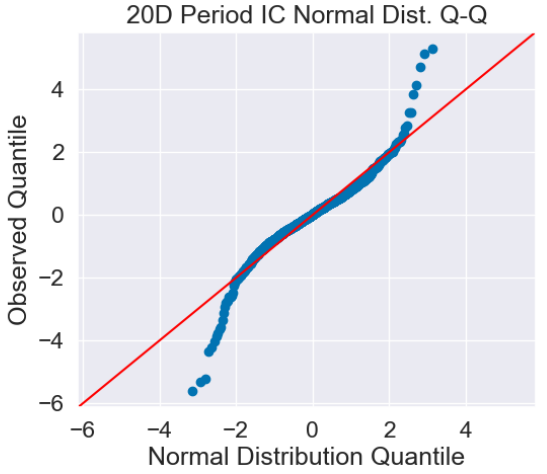

5D Period IC
Mean 0.021
Std. 0.065
Density
IC

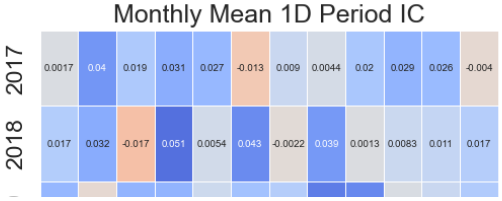

5D Period IC Normal Dist. Q-Q
Observed Quantile
Normal Distribution Quantile

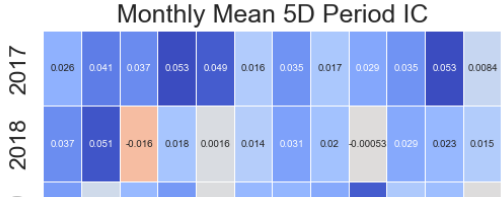

10D Period IC
Mean 0.025
Std. 0.061
Density
IC


10D Period IC Normal Dist. Q-Q
Observed Quantile
Normal Distribution Quantile

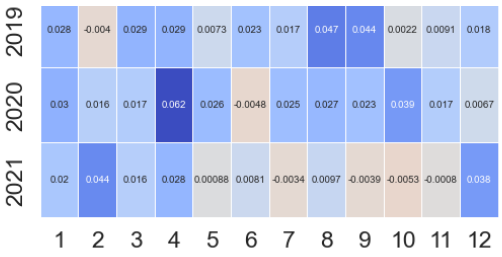

20D Period IC
Mean 0.027
Std. 0.060
Density
IC

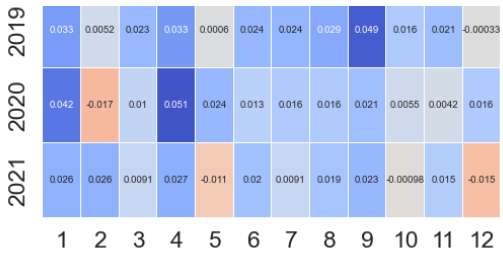

20D Period IC Normal Dist. Q-Q
Observed Quantile
Normal Distribution Quantile


Monthly Mean 1D Period IC
2017
0.0017 0.04 0.019 0.031 0.027 -0.013 0.009 0.0044 0.02 0.029 0.026 -0.004
2018
0.017 0.032 -0.017 0.051 0.0054 0.043 -0.0022 0.039 0.0013 0.0083 0.011 0.017

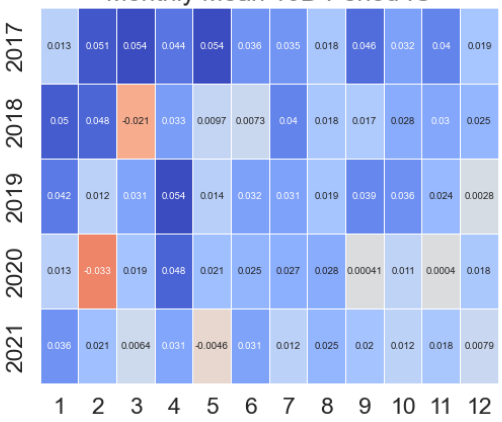

Monthly Mean 5D Period IC
2017
0.026 0.041 0.037 0.053 0.049 0.016 0.035 0.017 0.029 0.035 0.053 0.0084
2018
0.037 0.051 -0.016 0.018 0.0016 0.014 0.031 0.02 -0.00053 0.029 0.023 0.015

k_u_ret_fct

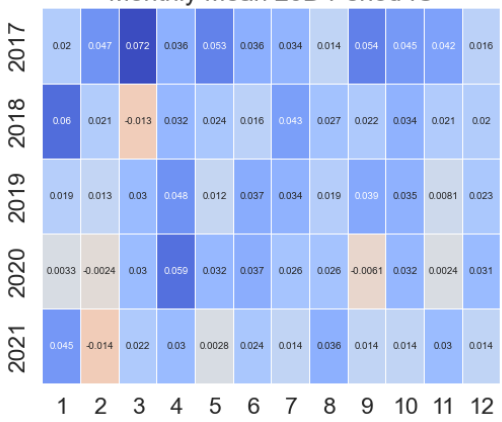




Monthly Mean 10D Period IC



Monthly Mean 20D Period IC



2.3.7 factor w_p_max_fct

The factor expression is:
-ts_max(ts_delta(vwap, 3), 5)

This factor calculates the maximum change rate of three-day vwap in the past five days. Using vwap increases the robustness of the factor and can be regarded as a special reversal factor.

The performance and detailed analysis of this factor are shown on the next pages:

Quantiles Statistics

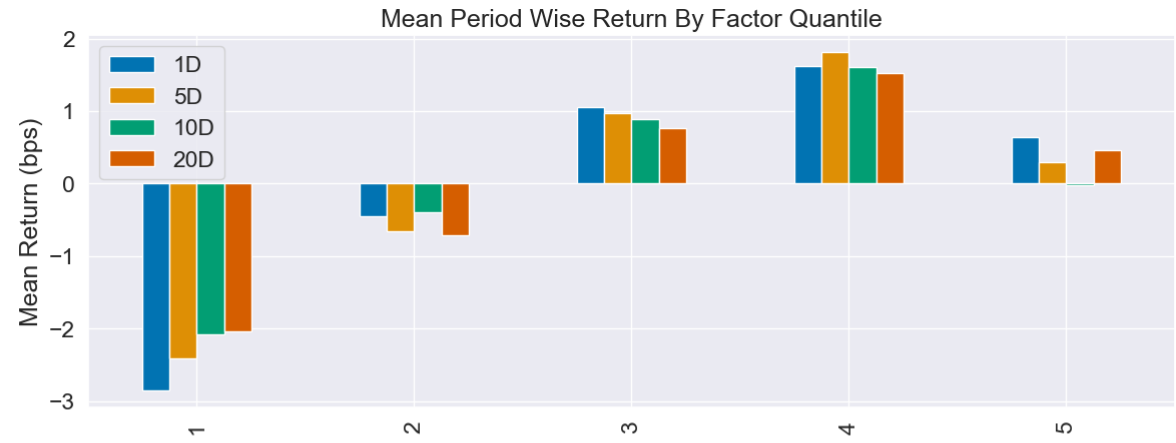
factor_quantile	min	max	mean	std	count	count %
1	-311.516767	-0.048856	-3.246868	5.586049	865255	20.011291
2	-2.564659	0.209523	-0.622288	0.334215	864509	19.994038
3	-1.162494	0.399279	-0.269045	0.181423	864546	19.994893
4	-0.645647	0.836162	-0.104635	0.135153	864509	19.994038
5	-0.366273	169.930153	0.195594	0.860569	865015	20.005740

Returns Analysis

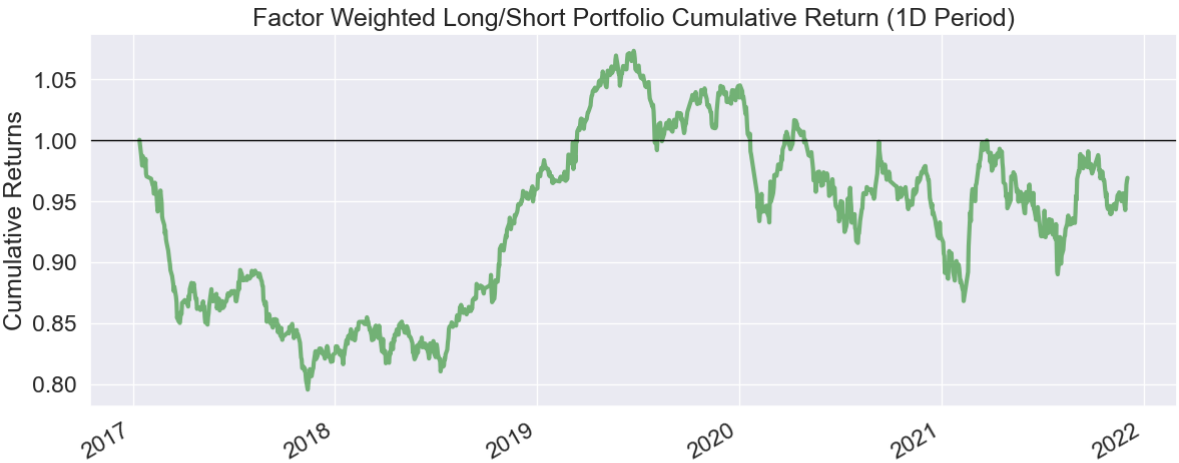
	1D	5D	10D	20D
Ann. alpha	0.001	0.016	0.023	0.032
beta	-0.039	-0.001	0.023	0.019
Mean Period Wise Return Top Quantile (bps)	0.637	0.291	-0.022	0.462
Mean Period Wise Return Bottom Quantile (bps)	-2.858	-2.415	-2.083	-2.043
Mean Period Wise Spread (bps)	3.494	2.797	2.186	2.664

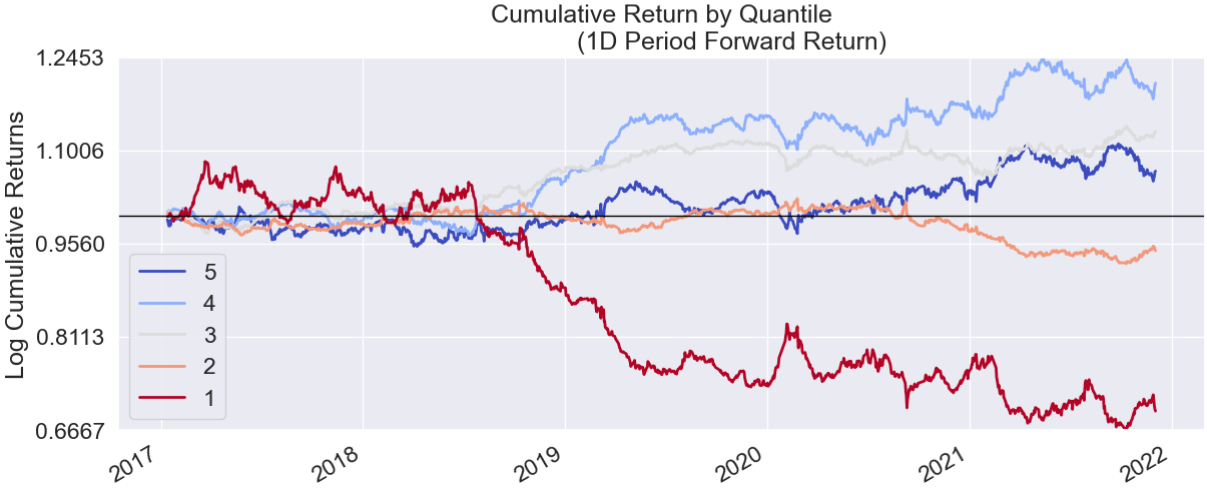
<Figure size 640x480 with 0 Axes>

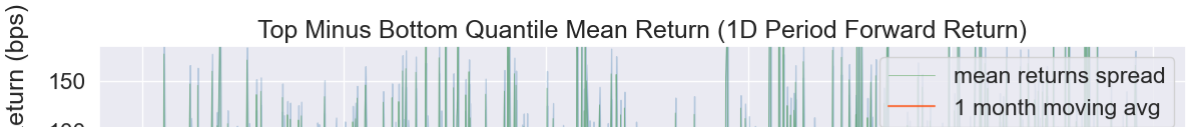
w_p_max_fct

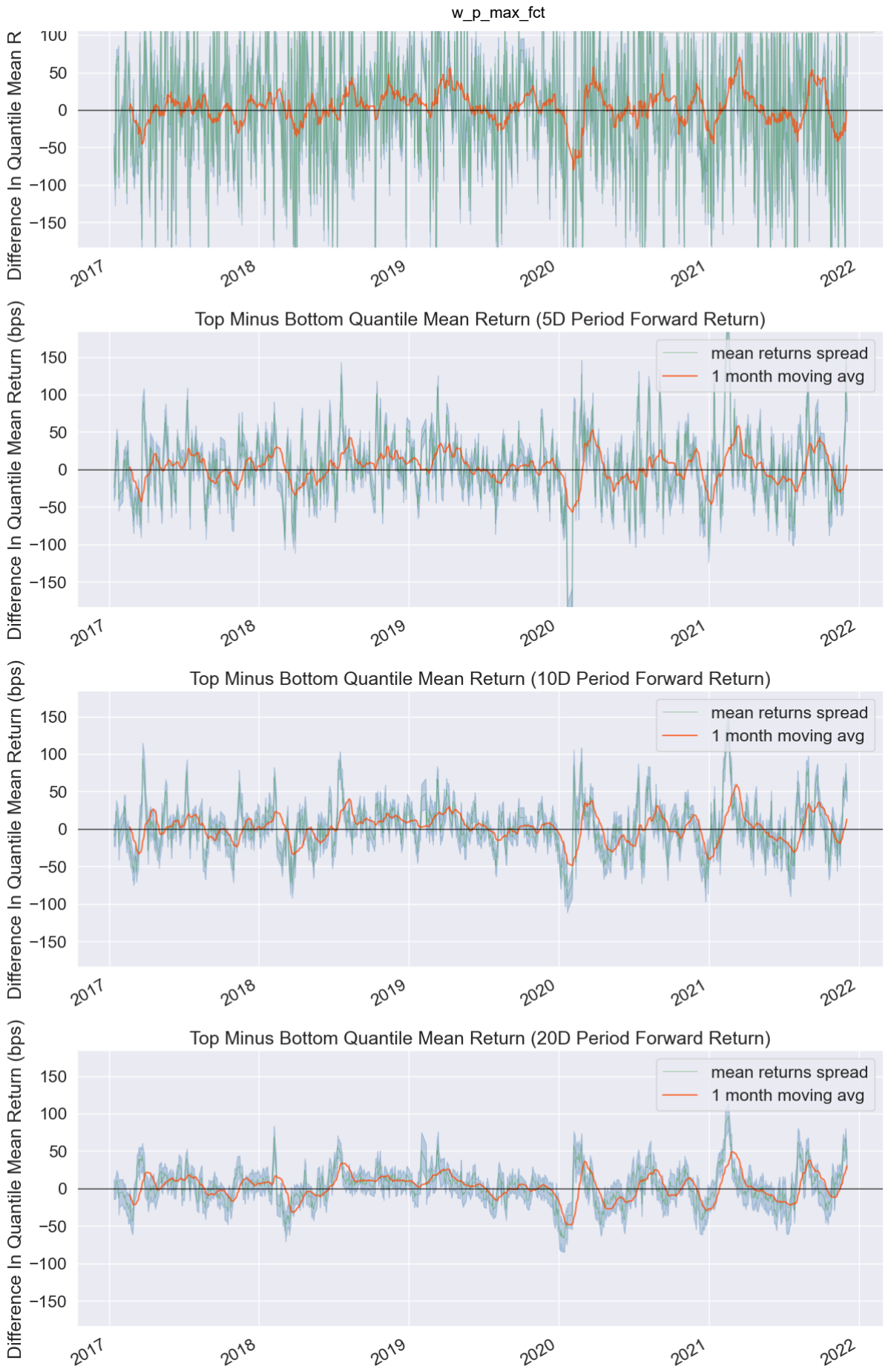

Mean Period Wise Return By Factor Quantile
Mean Return (bps)
1D
5D
10D
20D
2
1
0
−1
−2
−3
1
2
3
4
5


Period Wise Return By Factor Quantile
Return (bps)
60
40
20
0
−20
−40
−60
forward_periods
1D
5D
10D
20D
1
2
3
4
5


Factor Weighted Long/Short Portfolio Cumulative Return (1D Period)
Cumulative Returns
1.05
1.00
0.95
0.90
0.85
0.80
2017
2018
2019
2020
2021
2022

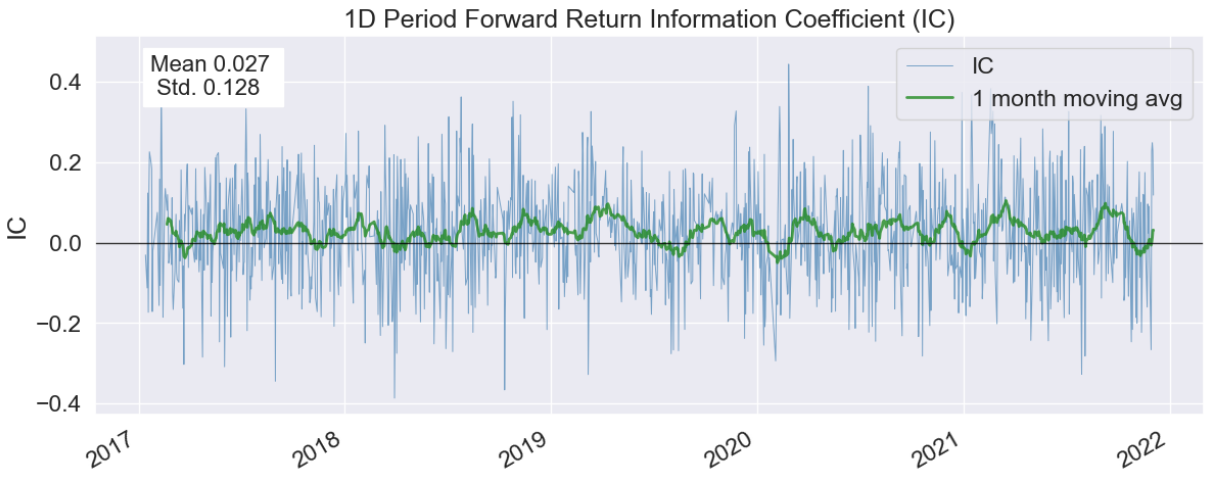

Cumulative Return by Quantile
(1D Period Forward Return)
Log Cumulative Returns
1.2453
1.1006
0.9560
0.8113
0.6667
5
4
3
2
1
2017
2018
2019
2020
2021
2022

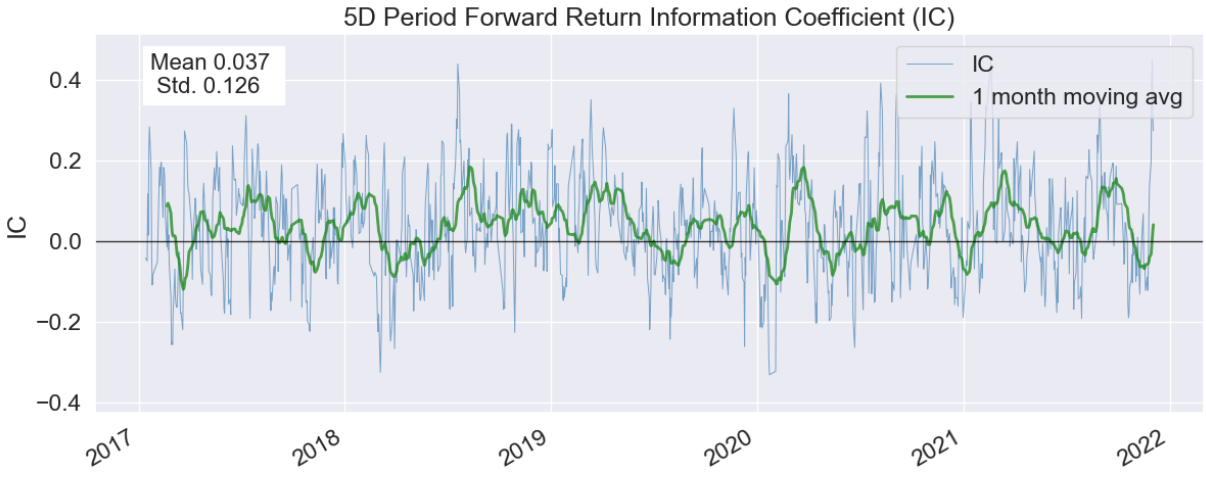

Top Minus Bottom Quantile Mean Return (1D Period Forward Return)
eturn (bps)
150
mean returns spread
1 month moving avg

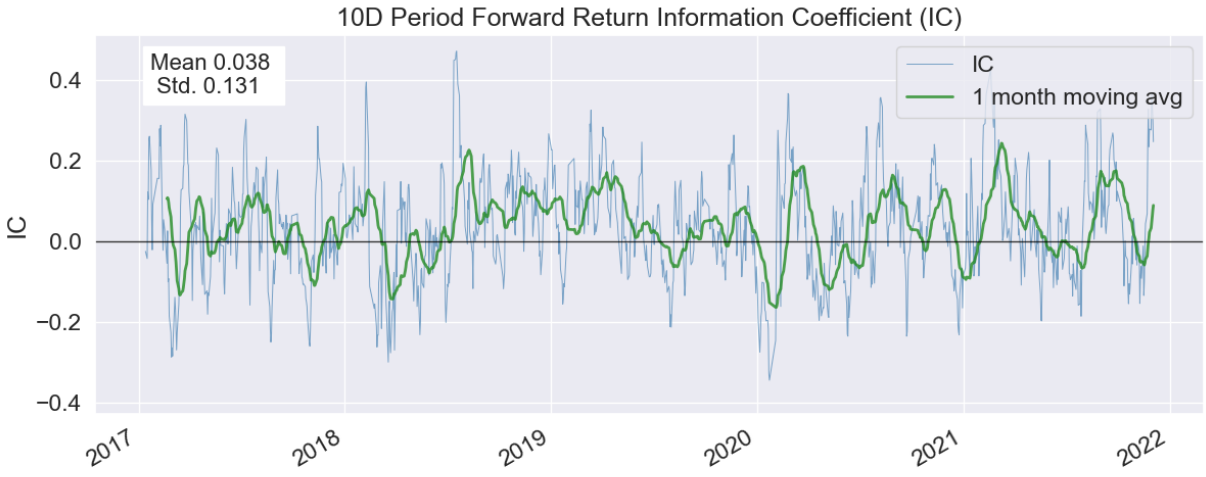


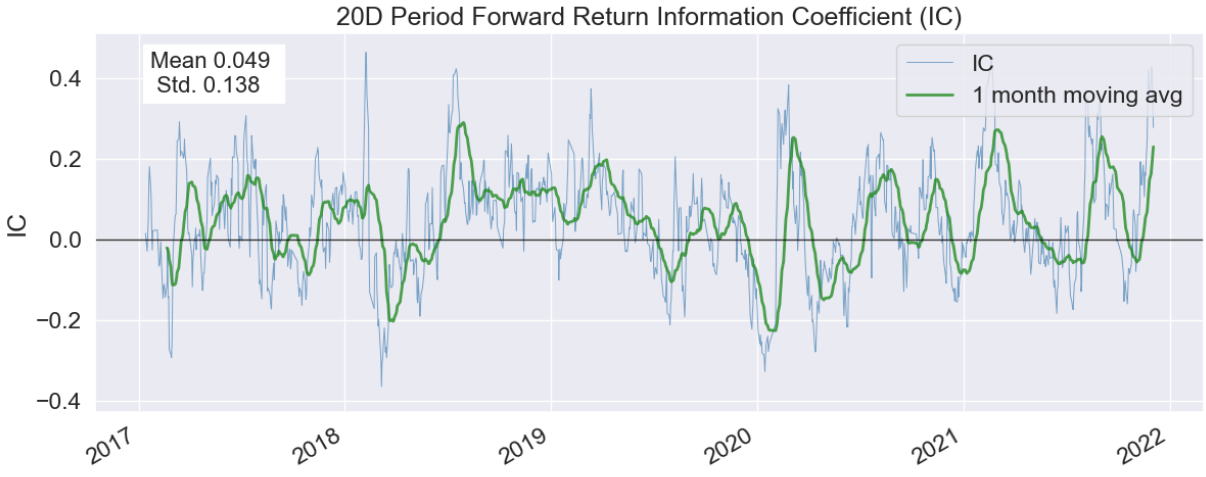
Information Analysis

	1D	5D	10D	20D
IC Mean	0.027	0.037	0.038	0.049
IC Std.	0.128	0.126	0.131	0.138
Risk-Adjusted IC	0.214	0.294	0.293	0.355
t-stat(IC)	7.387	10.146	10.087	12.254
p-value(IC)	0.000	0.000	0.000	0.000
IC Skew	-0.047	0.138	0.168	0.094
IC Kurtosis	0.024	-0.051	0.146	0.165

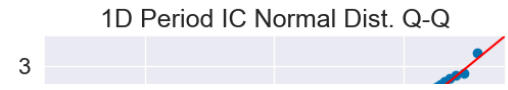

1D Period Forward Return Information Coefficient (IC)
Mean 0.027
Std. 0.128
IC
1 month moving avg
IC
0.4
0.2
0.0
−0.2
−0.4
2017
2018
2019
2020
2021
2022

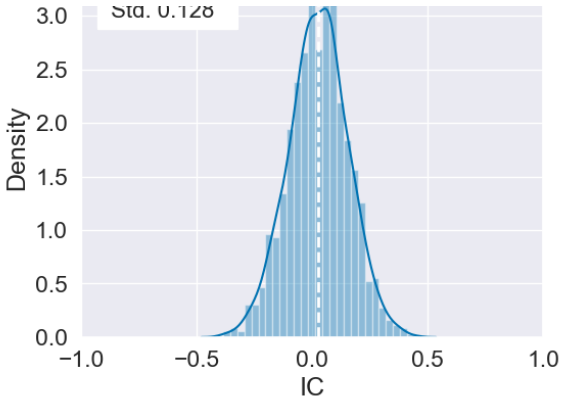

5D Period Forward Return Information Coefficient (IC)
Mean 0.037
Std. 0.126
IC
1 month moving avg
IC
0.4
0.2
0.0
−0.2
−0.4
2017
2018
2019
2020
2021
2022

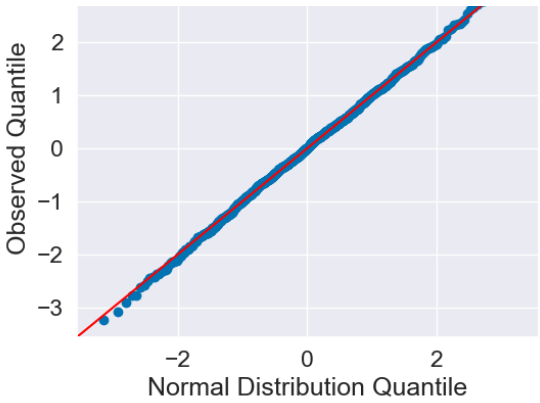

10D Period Forward Return Information Coefficient (IC)
Mean 0.038
Std. 0.131
IC
1 month moving avg
IC
0.4
0.2
0.0
−0.2
−0.4
2017
2018
2019
2020
2021
2022


20D Period Forward Return Information Coefficient (IC)
Mean 0.049
Std. 0.138
IC
1 month moving avg
IC
0.4
0.2
0.0
−0.2
−0.4
2017
2018
2019
2020
2021
2022

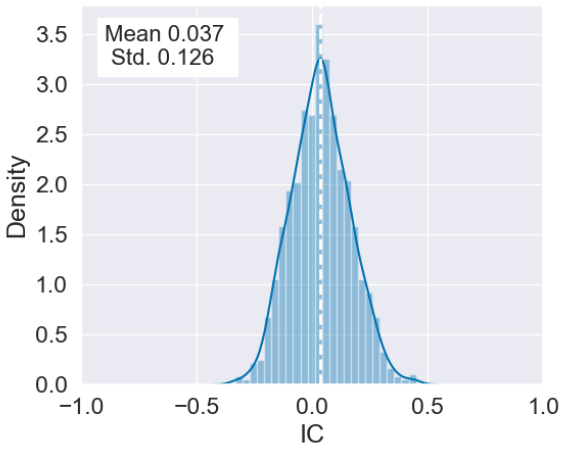

1D Period IC
Mean 0.027
3.5


1D Period IC Normal Dist. Q-Q
3

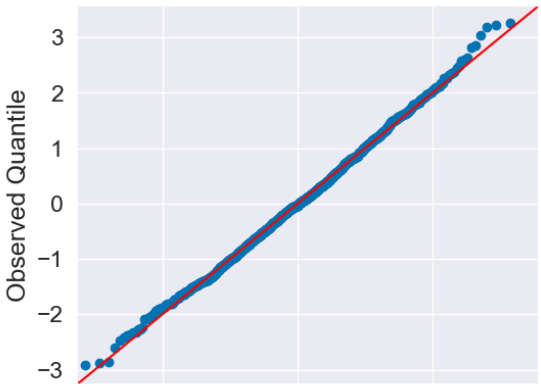




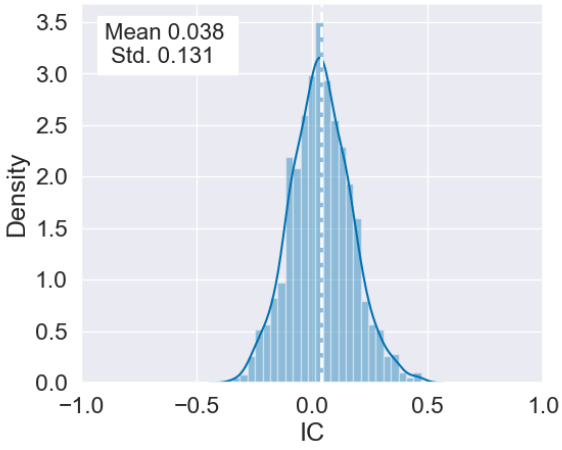
5D Period IC



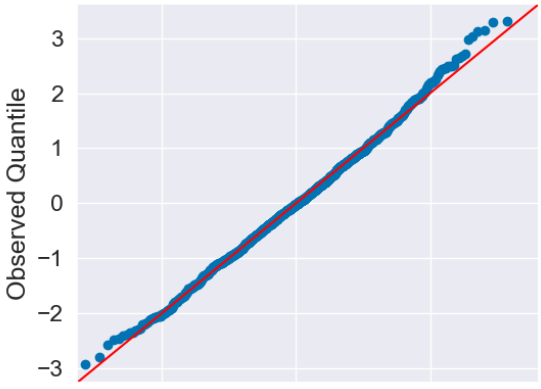
5D Period IC Normal Dist. Q-Q



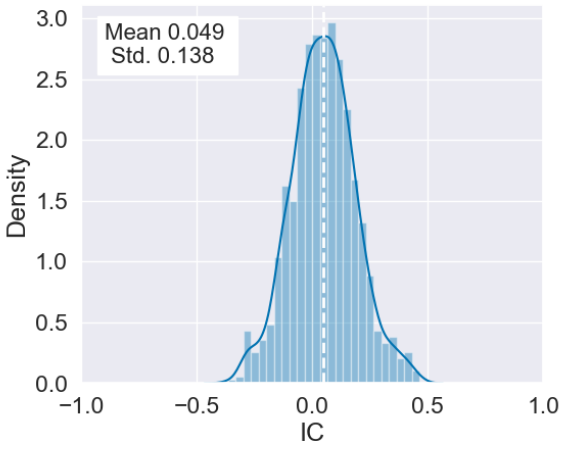
10D Period IC



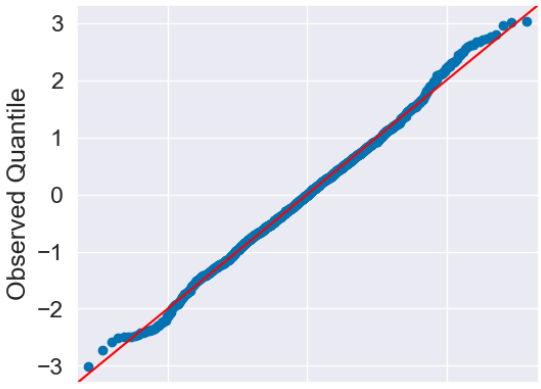
10D Period IC Normal Dist. Q-Q



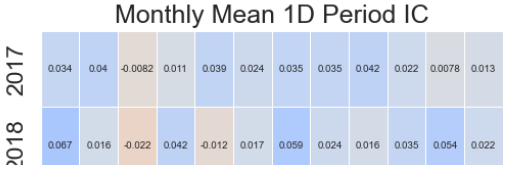
20D Period IC



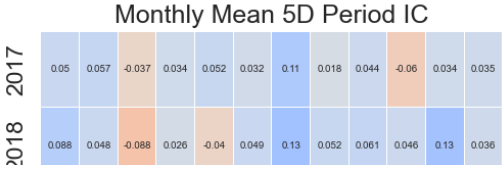
20D Period IC Normal Dist. Q-Q



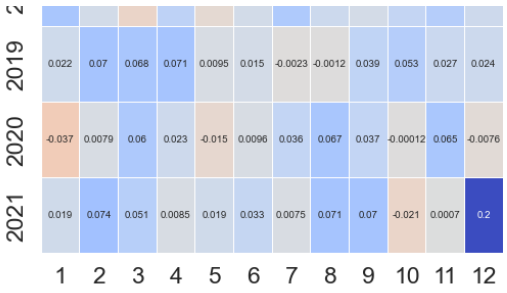
Monthly Mean 1D Period IC

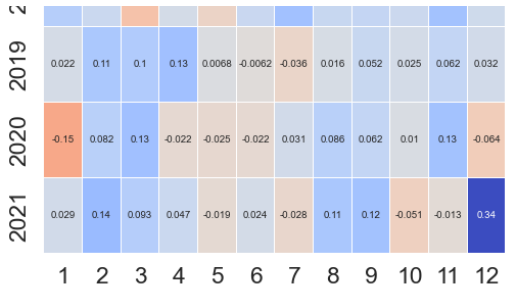


Monthly Mean 5D Period IC

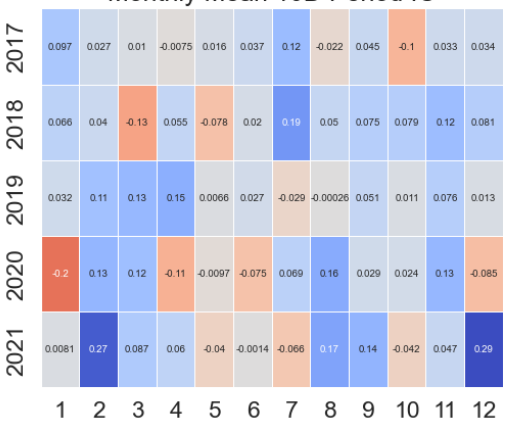


w_p_max_fct

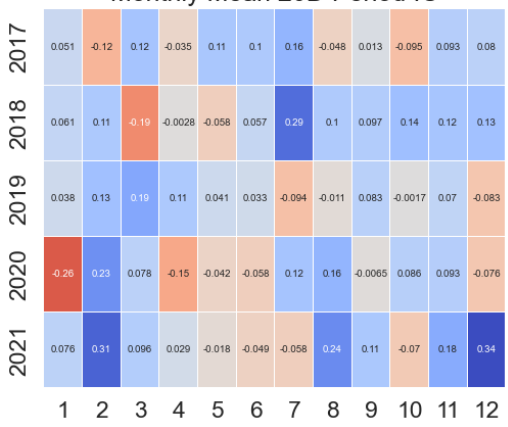




Monthly Mean 10D Period IC



Monthly Mean 20D Period IC



2.4 Genetic Programming

Genetic programming (GP) is a powerful algorithmic approach to factor digging in quantitative finance. Factor digging involves searching for and identifying mathematical relationships or factors that have predictive power for asset returns. Genetic programming can be applied in this context to automatically evolve mathematical expressions or factors that maximize a chosen objective function, such as risk-adjusted returns or information ratio.

The process of discovering high-quality single factors is like an adventure, requiring more thinking and observation. The process of optimizing a single factor is also interesting. Through hard work, it is gratifying to see satisfactory results, soaring returns, high IC, and insignificant retracement. And the process itself is also a learning experience. Technology is constantly changing, and as a human being, we need to constantly learn and expand our knowledge and skills to keep up with the times.