# AlgoPack: Алгоритмический трейдинг | Часть 3: Модель

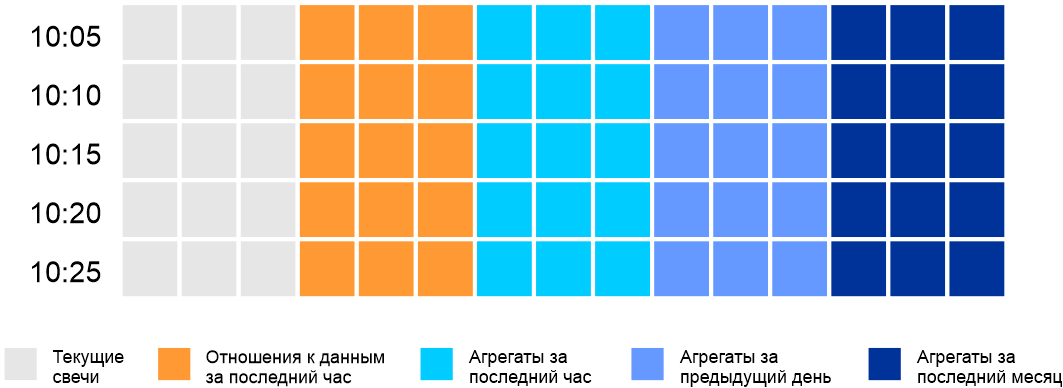
Это третья часть цикла статей, в который мы создаем торгового бота. Сегодня нам предстоит обучить модель машинного обучения, которая будет управлять нашим ботом.

Список статей:

* AlgoPack: Алгоритмический трейдинг | Часть 1: Введение
* AlgoPack: Алгоритмический трейдинг | Часть 2: EDA
* AlgoPack: Алгоритмический трейдинг | Часть 3: Модель
* AlgoPack: Алгоритмический трейдинг | Часть 4: Бот

## Датасет

Прежде всего нам нужно подготовить датасет для обучения.



Датасет, который мы создадим будет иметь следующую структур (по каждой строчке):

1. Торговая статистика за 5 минутный промежуток (возвращаемые методом tradestats).
2. Посчитаем отношение текущей пятиминутки к значениям каждой пятиминутки за последний час
3. Посчитаем агрегаты за последний час, последний день и последний месяц.

### Загрузка данных

Наш бот будет запускаться один раз в 5 минут. И чтобы каждый раз не запрашивать всю необходимую информацию у AlgoPack, будем заранее загружать исторические данные и складировать их локально.

Сначала предварительно загрузим всю историю за последний год:

from moexalgo import Ticker

from datetime import date, timedelta

import pandas as pd

import time

TICKER='YNDX'

loader = Ticker(TICKER)

load\_date = date(2023, 1, 1)

end\_date = date(2023, 12, 8)

while load\_date <= end\_date:

date\_str = load\_date.strftime("%Y-%m-%d")

one\_day\_df = loader.tradestats(date=date\_str, till\_date=date\_str)

#one\_day\_df = pd.DataFrame(one\_day\_df)

if one\_day\_df.shape[0] > 0:

print(date\_str, TICKER, one\_day\_df.shape)

one\_day\_df.to\_pickle(f"data/raw/{date\_str.replace('-','')}\_{TICKER}.pkl")

else:

print(load\_date, TICKER, 'Данных нет')

load\_date += timedelta(days=1)

time.sleep(3)

Код: <https://github.com/slivka83/algopack_simple_bot/blob/main/src/data/get_history.py>

З.Ы. Не стоит спамить API AlgoPack. Более 30 асинхронных запросов в секунду и можно получить бан. Делайте паузу между запросами.

З.Ы. Имейте в виду, что биржа не работает круглосуточно, поэтому за какие-то периоды данных может и не быть.

Теперь нужно обеспечить регулярное поступление актуальных данных. Для этого будем ежедневно (когда биржа не работает; предположительно в 1 час ночи), запускать следующий код и загружать статистику за прошлый день и сохранять ее локально.

from moexalgo import Ticker

from datetime import datetime, timedelta

import pandas as pd

TICKER = 'YNDX'

yesterday = datetime.today() - timedelta(days=1)

yesterday = yesterday.strftime('%Y-%m-%d')

loader = Ticker(TICKER)

one\_day\_df = loader.tradestats(date=yesterday, till\_date=yesterday)

#one\_day\_df = pd.DataFrame(one\_day\_df)

if one\_day\_df.shape[0] > 0:

print(yesterday, TICKER, one\_day\_df.shape)

one\_day\_df.to\_pickle(f"data/raw/{yesterday.replace('-','')}\_{TICKER}.pkl")

else:

print(yesterday, TICKER, 'Данных нет')

Код: <https://github.com/slivka83/algopack_simple_bot/blob/main/src/data/get_one_day.py>

Посмотрим, что у нас в итоге получилось:

import pandas as pd

import glob

all\_files = glob.glob('../data/raw/\*.pkl')

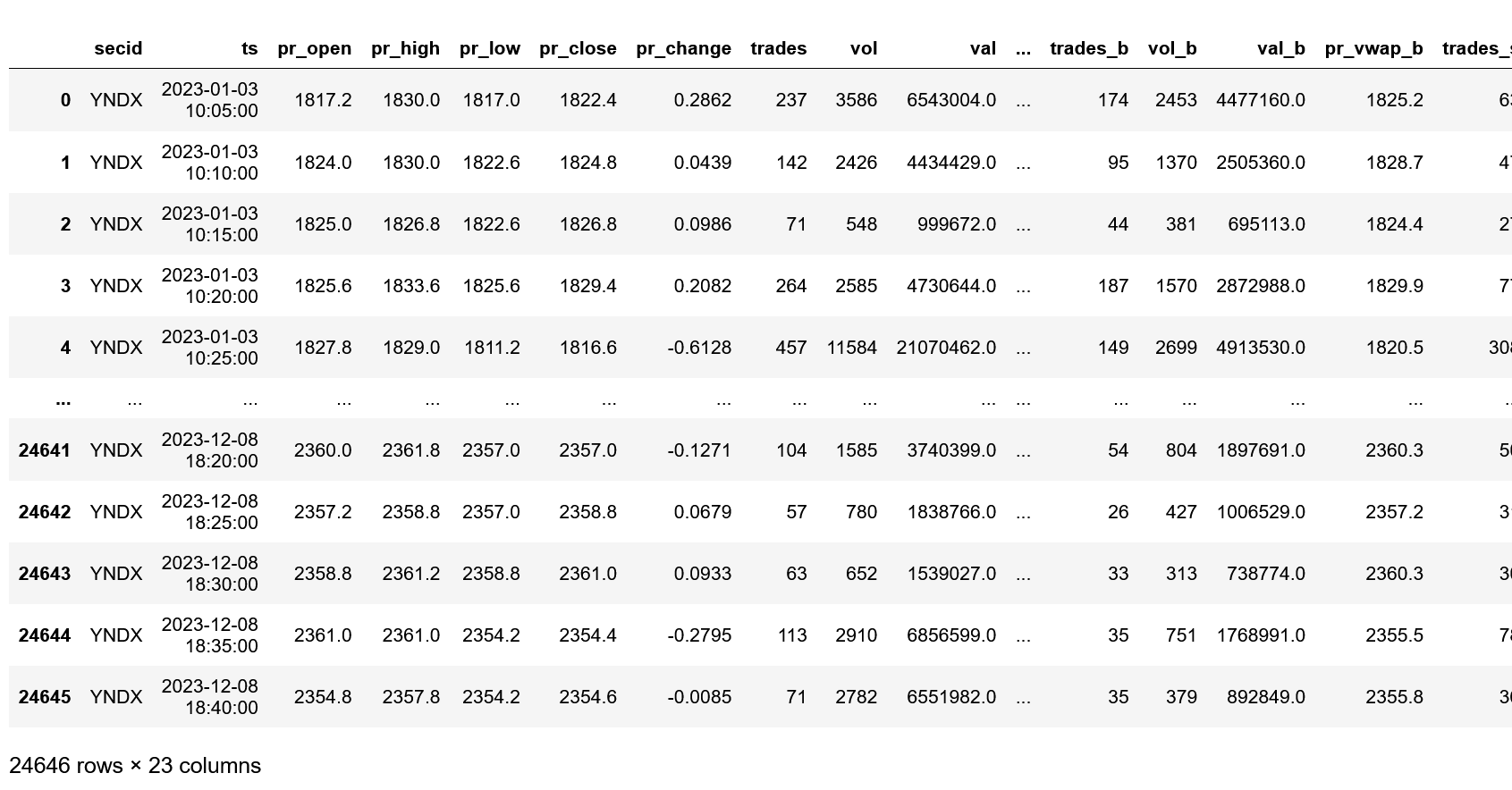
df = []

for filename in all\_files:

tdf = pd.read\_pickle(filename)

df.append(tdf)

df = pd.concat(df, axis=0, ignore\_index=True)



Итого у нас имеется около 25к строк и 23 столбца. Из них 3 столбца служебных и 20 полезных. С тем, какие именно поля возвращает метод tradestats, вы можете ознакомится в предыдущей статье цикла.

Дополним датасет новыми фичами.

### Фичи

Все фичи, которые мы негенерируем так или иначе будет основаны на лагах временного ряда. А формировать их мы будем на основе одной единственной фичи - среднего между максимальным и минимальным значением 5-минутной свечи.

df['pr\_mean'] = df[['pr\_high','pr\_low']].mean(axis=1)

Посчитаем на сколько изменилась среднее по отношению ко всем пятиминуткам за последний час:

for i in range(1,11+1):

df[f'ratio\_pr\_mean\_{i}'] = df['pr\_mean'] / df['pr\_mean'].shift(i)

Посчитаем четыре статистики за последний час (по отношении к каждой пятиминутке):

for a in ['min','max','mean','std']:

df[f'hour\_{a}'] = df.groupby('tradedate')['pr\_mean'].transform(

lambda s: s.shift(1).rolling(11).agg(a))

Статистики за прошлый день:

agg\_temp = df.groupby('tradedate')['pr\_mean'].agg(['min','max','mean','std'])

agg\_temp.columns = [f'ytd\_{col}' for col in agg\_temp.columns]

agg\_temp = agg\_temp.reset\_index()

agg\_temp['tradedate'] = agg\_temp['tradedate'] + timedelta(days=1)

df = df.merge(agg\_temp, on='tradedate', how='left')

Статистики за прошлый месяц

month\_temp = df['pr\_mean'].rolling(3120).agg(['min','max','mean','std'])

month\_temp.columns = [f'month\_{col}' for col in month\_temp.columns]

month\_temp['tradedate'] = df['tradedate']

month\_temp['tradetime'] = df['tradetime']

month\_temp['tradedate'] = month\_temp['tradedate'].shift(-1)

month\_temp = month\_temp.groupby('tradedate').nth(0)

month\_temp = month\_temp.iloc[:,:-1]

df = df.merge(month\_temp, on='tradedate', how='left')

Код: <https://github.com/slivka83/algopack_simple_bot/blob/main/src/features/create_features.py>

**To-Do**

Получившийся датасет весьма простой. И здесь просто бездонный край для экспериментов. Если посидеть несколько часов, то можно легко нагенерить несколько тысяч фичей:

1. Больше базовых данных. Мы подключили только торговую статистику. Но можно также задействовать статистику по заказам и ODS.
2. Больше статистик: различные квантили, скосы, характеристики временных рядов и т.д.
3. Больше периодов: 3 дня, неделя, месяц, 3 месяца, год и т.д.
4. Больше фичей для агрегатов. Мы использовали только одну фичу — среднее значение торгов. Но только AlgoPack предоставляет около 50 базовых фичей. И на основе их всех можно строить лаги.
5. Можно перейти на минутные свечи (и цеплять к ним 5-минутные статистики).
6. Подключить данные по другим акциям (особенно скоррелированным).

Технические улучшение:

1. Предусмотреть дозагрузку пропущенных периодов.
2. Хранить данные в БД.

### Таргет

Наша цель: продавать акции когда они находятся на пике и покупать когда они находятся в локальном минимуме. Соответственно, нам нужно разметить пики и впадины на временном ряду средней цены. Для этого воспользуемся методом find\_peaks библиотеки ScyPy.

from scipy.signal import find\_peaks

# Дистанция между пикам/впадинами

dist = 5

# Считаем пики

peaks, \_ = find\_peaks(df['pr\_mean'], distance=dist)

temp = pd.DataFrame(index=peaks)

temp['peak'] = 1

df = df.join(temp)

df['peak'] = df['peak'].fillna(0).astype(int)

# Считаем впадины

peaks, \_ = find\_peaks(0 - df['pr\_mean'], distance=dist)

temp = pd.DataFrame(index=peaks)

temp['trough'] = 2

df = df.join(temp)

df['trough'] = df['trough'].fillna(0).astype(int)

# Формируем таргет

df['target'] = df[['trough','peak']].max(axis=1)

Посмотрим как будет выглядеть последний день на рафике.

import matplotlib.pyplot as plt

import seaborn as sns

temp = df[['tradedate','tradetime','target','pr\_mean']][-104:]

temp['dt'] = df['tradedate'].astype(str) + ' ' + df['tradetime'].astype(str)

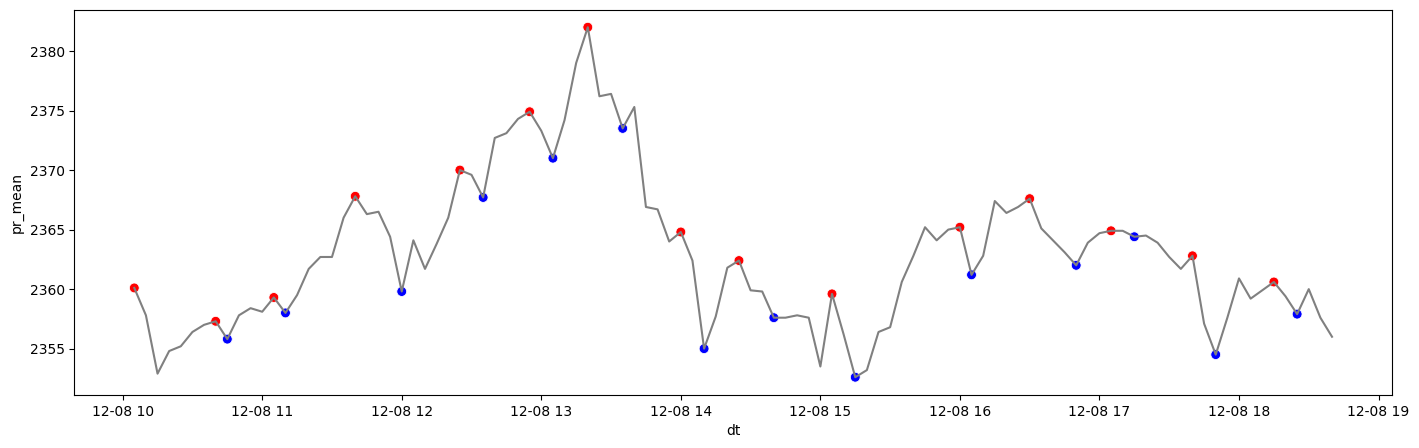
temp['dt'] = pd.to\_datetime(temp['dt'])

plt.figure(figsize=(17,5))

sns.lineplot(x=temp['dt'], y=temp['pr\_mean'], c='grey')

colors = temp.query('target > 0')['target'].map({1:'red',2:'blue'}).to\_list()

sns.scatterplot(x=temp.query('target > 0')['dt'], y=temp.query('target > 0')['pr\_mean'], c=colors, s=50);



Как можно заметить, пики почти всегда выше впадин. И если во впадинах покупать акции, а на пиках продавать — то сможем заработать (в теории :)

**To-Do**

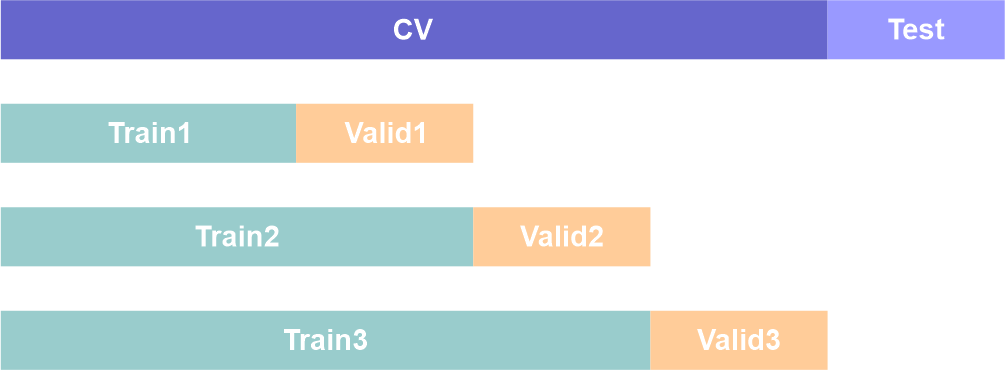
1. Попробовать другие расстояния между пиками/впадинами.
2. Посчитать скользящую среднюю и использовать ее для определения пиков и впадин, чтобы шум не влиял таргет.
3. Постобработка таргета, чтобы:
   1. Пики и впадины всегда чередовались.
   2. Пики всегда были выше впадин.
   3. Между пиками и впадина была определенная разница в цене.
4. Сместить таргет в прошлое, чтобы предсказывать его на будущее.

## Модель

### Датасеты

Для обучения будем проводить классическую кросс-валидацию на временных рядах.

Для теста будем использовать один последний исторический день. Целый год до него — для обучения.



today = datetime.today().strftime('%Y-%m-%d')

df = pd.read\_pickle(f"../data/processed/{today.replace('-','')}\_{ticker}.pkl")

df = df.fillna(0)

drop\_cols = ['ticker','secid','ts','tradedate','tradetime','systime','peak','trough','target']

f\_cols = [col for col in df.columns if col not in drop\_cols]

last\_day = str(df['tradedate'].max())

train = df[df['tradedate'].astype('str') < last\_day]

test = df[df['tradedate'].astype('str') == last\_day]

X\_train, y\_train = train[f\_cols], train['target']

X\_test, y\_test = test[f\_cols], test['target']

print(X\_train.shape, X\_test.shape)

print(y\_train.mean() \* 100, y\_test.mean() \* 100)

### Обучение

Сделаем простой поиск гиперпараметров по сетка. А в качестве базовой модели будем использовать LGBM:

params = {

'n\_estimators ': [100],

'learning\_rate ': [0.9, 0.8, 0.7, 0.5, 0.4, 0.3, 0.2, 0.1],

#'max\_depth ': [1,2,3,4,5,6,7,8],

'subsample ': [0.3],

'colsample\_bytree ': [0.3]

}

clf = lgb.LGBMClassifier(verbosity=-1)

model = GridSearchCV(

clf,

params,

scoring='precision\_macro',

cv=TimeSeriesSplit(n\_splits=5))

model.fit(X\_train, y\_train)

print(model.best\_score\_)

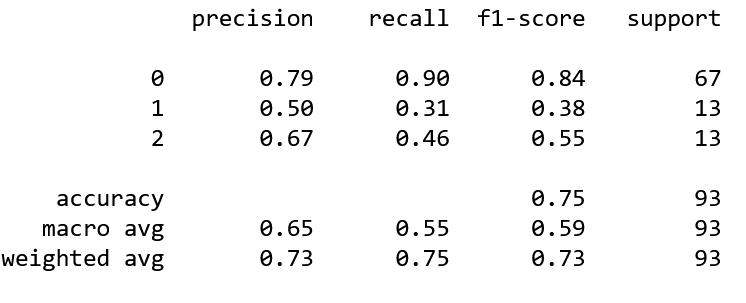
print(model.best\_params\_)

### Оценка

Посмотрим чему мы научились. В данной задаче нам важна точность, поэтому в качестве метрики будем использовать Precision:

y\_pred = model.predict(X\_test)

print(classification\_report(y\_test, y\_pred)) # , labels=[1, 2, 3]



Наша точность на одном (последнем) дне составила 65%. Не знаю хорошо это или плохо, но для первого подхода к снаряду — сойдет :)

Код обучения: <https://github.com/slivka83/algopack_simple_bot/blob/main/src/models/train_model.py>

Выведем на графики реальный и предсказанный таргет:

temp = test[['tradedate','tradetime','target','pr\_mean']]

temp['pred'] = y\_pred

temp['dt'] = temp[['tradedate','tradetime']].astype(str).apply(

lambda row: row['tradedate'] + ' ' + row['tradetime'], axis=1)

temp['dt'] = pd.to\_datetime(temp['dt'])

plt.figure(figsize=(17,5))

sns.lineplot(x=temp['dt'], y=temp['pr\_mean'], c='grey')

colors = temp.query('target > 0')['target'].map({1:'red',2:'blue'}).to\_list()

sns.scatterplot(x=temp.query('target > 0')['dt'], y=temp.query('target > 0')['pr\_mean'], c=colors, s=50);

plt.show()

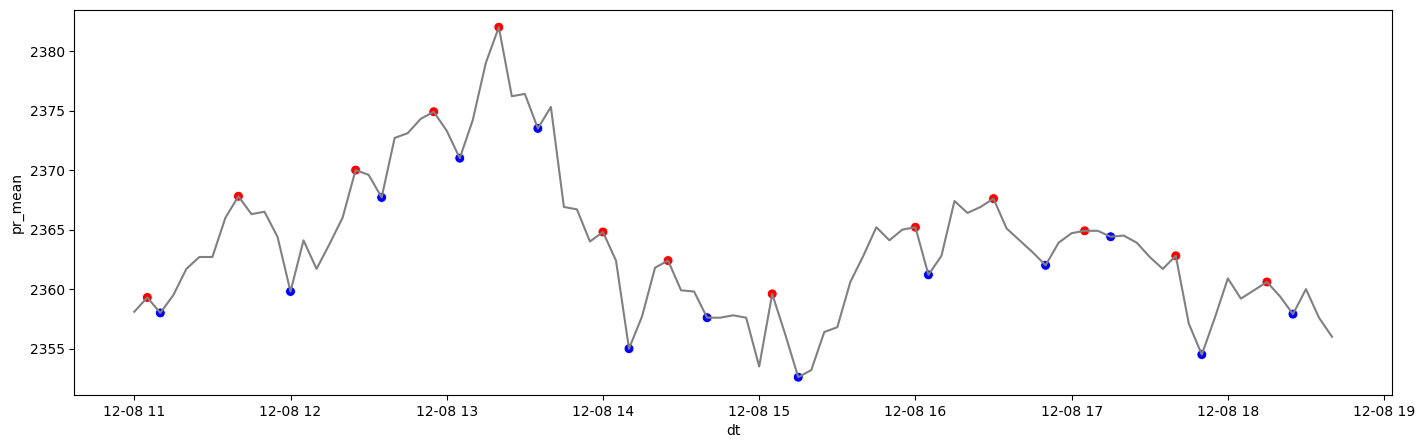
plt.figure(figsize=(17,5))

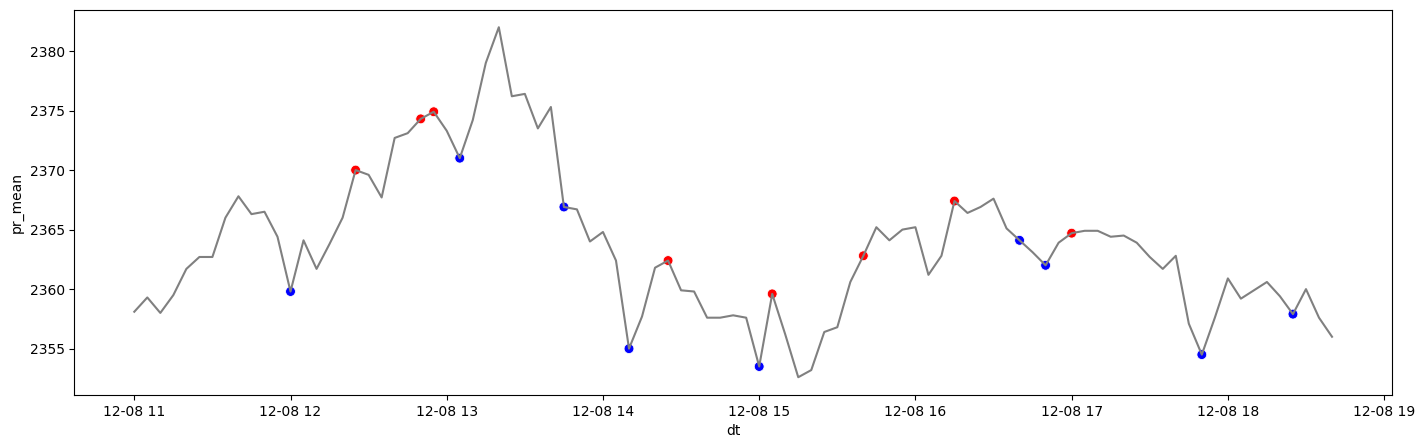
sns.lineplot(x=temp['dt'], y=temp['pr\_mean'], c='grey')

colors = temp.query('pred > 0')['pred'].map({1:'red',2:'blue'}).to\_list()

sns.scatterplot(x=temp.query('pred > 0')['dt'], y=temp.query('pred > 0')['pr\_mean'], c=colors, s=50);

plt.show()





Сейчас на предсказание модели больше всего влияют отношения текущий пятиминутки к значениям предыдущего часа. Довольно логично, если учесть как считается таргет.

fi\_df = pd.DataFrame({

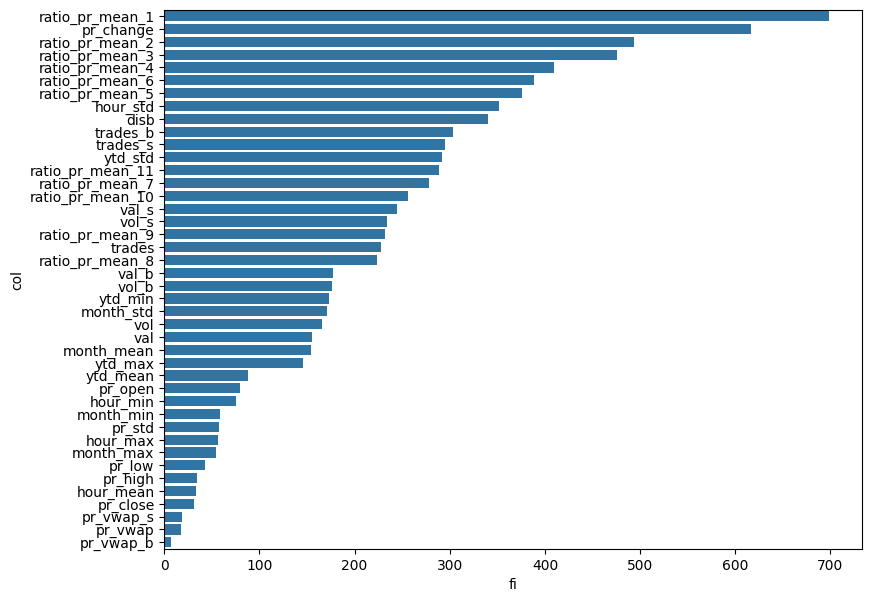
'col': X\_train.columns,

'fi': model.best\_estimator\_.feature\_importances\_

}).sort\_values(by='fi', ascending=False)

plt.figure(figsize=(9,7))

sns.barplot(data=fi\_df, y='col', x='fi', orient='h');



**To-Do**

* Попробовать другие модели: линейные (деревянные все таки не умеют экстраполировать), нервные сетки и т.д..
* Подбор гиперпараметров с помощью Optuna.
* Можно перейти на минутные свечи и цеплять к ним 5-минутные статистики.
* Отдельные модели для пиков и впадин.
* Попробовать изменить трешхолд для определения класса.
* Постпроцессинг: не покупать, если в прошлый раз покупали.
* Более точная оценка модели посредством бэктестинга.

Сердце нашего робота готово, пора заставить его делать что-то полезное...