# AlgoPack: Алгоритмический трейдинг | Часть 2: EDA

Это вторая часть цикла статей, в который мы создаем торгового бота. В этой части поближе посмотрим на данные, которые предоставляет платформа AlgoPack. А также нарисуем несколько графиков, чтобы визуально представить с чем имеем дело.

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

Ноутбук: <https://github.com/slivka83/algopack_simple_bot/blob/main/notebooks/EDA.ipynb>

## Структура данных

Фондовая биржа (и другие продавцы биржевой информации) по умолчанию предоставляют достаточно ограниченный набор ценовых данных о торгах. Как правило это только OHLC (open-high-low-close), т.е. цена открытия, минимум/минимум и цена закрытия за период. AlgoPack же изначально предоставляет более 50 параметров. В них входят как базовые характеристики стакана (OHLC), так и дополнительные агрегированные показатели.

Структур данных AlgoPack можно предоставить в трех разрезах:

* По охвату: одна компания или все акции
* По содержанию:
  + Японские свечи
  + Торговая статистика на основе сделок
  + Торговая статистика на основе заказов
  + Торговая статистика на основе стакана заявок
* Таймфрейм: 1 мин, 5 мин, 10 мин , 1 час, 1 день   
  З.Ы. Доступные значения зависят от предыдущих двух пунктов.

Полное описание данных: <https://www.moex.com/ru/algopack/description_dataset>

## Подключение

Данные с AlgoPack можно забрать двумя способами: либо через API, либо через Python-библиотеку.

### API

Запросы к API можно отправлять примерно так:

import time

import pandas as pd

date = '2023-11-20'

tradestats = []

# запрос возвращает не более 1000 записей, чтобы получить все записи используем курсор

for cursor in range(5):

url = f'https://iss.moex.com/iss/datashop/algopack/eq/tradestats.csv?date={date}&start={cursor\*1000}&iss.only=data'

df = pd.read\_csv(url, sep=';', skiprows=2)

tradestats.append(df)

if df.shape[0] < 1000:

break

time.sleep(0.5)

tradestats = pd.concat(tradestats)

Описание API: <https://www.moex.com/ru/algopack/description_api>

Еще примеры: <https://github.com/moexalgo/moexalgo/blob/main/samples/download_algopack_data.ipynb>

### Библиотека

Второй способ — использовать специальную Python-библиотеку — [moexalgo](https://github.com/moexalgo/moexalgo):

pip install moexalgo

Библиотека предоставляет три метода для получения данных.

* TradeStats
* OrderStats
* OBStats

Дальше подробнее рассмотрим каждый из этих методов.

Другие примеры: <https://github.com/moexalgo/moexalgo/blob/main/samples/quick_start.ipynb>

З.Ы. Библиотека кажется немного удобнее чем API, но API сейчас предоставляет несколько больше данных.

## Весь рынок

Сначала посмотрим на рынок в целом.

### Торговая статистика (tradestats)

Метод tradestats возвращает метрики, рассчитанные основе потока сделок:

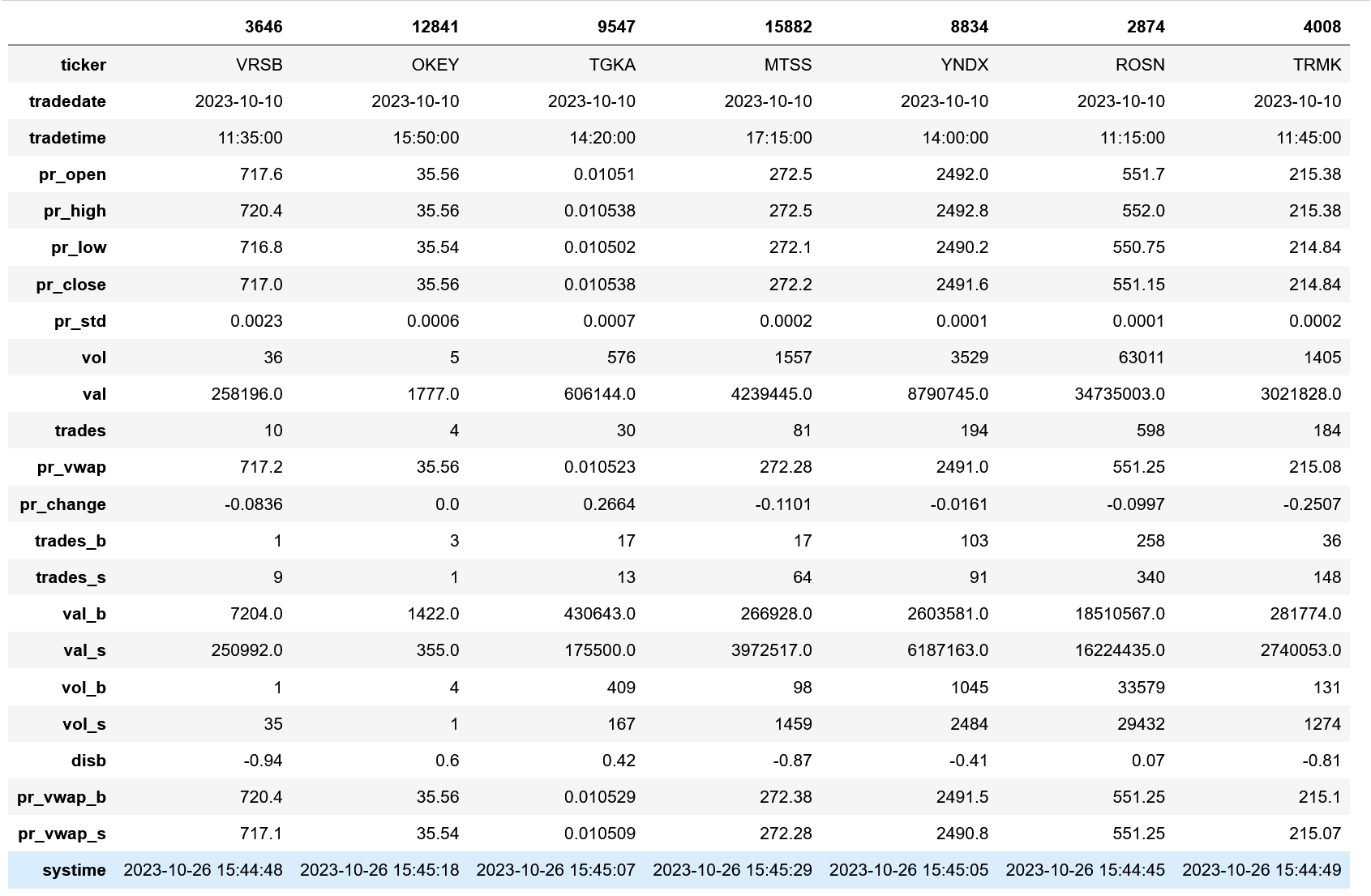
* pr\_open - цена открытия
* pr\_high - максимальная цена за период
* pr\_low - минимальная цена за период
* pr\_close - последняя цена за период
* pr\_std - стандартное отклонение цены
* pr\_vwap - средневзвешенная цена
* pr\_change - изменение цены за период, %
* pr\_vwap\_b - средневзвешенная цена покупки
* pr\_vwap\_s - средневзвешенная цена продажи
* vol - объем в лотах
* vol\_b - объем покупок в лотах
* vol\_s - объем продаж в лотах
* val - объем в рублях
* val\_b - объем покупок в руб
* val\_s - объем продаж в руб
* trades - кол-во сделок
* trades\_b - кол-во сделок на покупку
* trades\_s - кол-во сделок на продажу
* disb - соотношение объема покупок и продаж

Получим торговую статистику за один день:

stocks = Market('stocks')

trd\_df = stocks.tradestats(date='2023-12-05')

trd\_df.sample(7).T



Посмотрим верхнеуровнево, что происходило 2023-12-05. В этот день торги шли по 243 акциям. Но основная часть сделок была сосредоточена на небольшом количестве позиций.

temp\_df = pd.pivot\_table(

trd\_df,

index=['ticker'],

columns=['tradetime'],

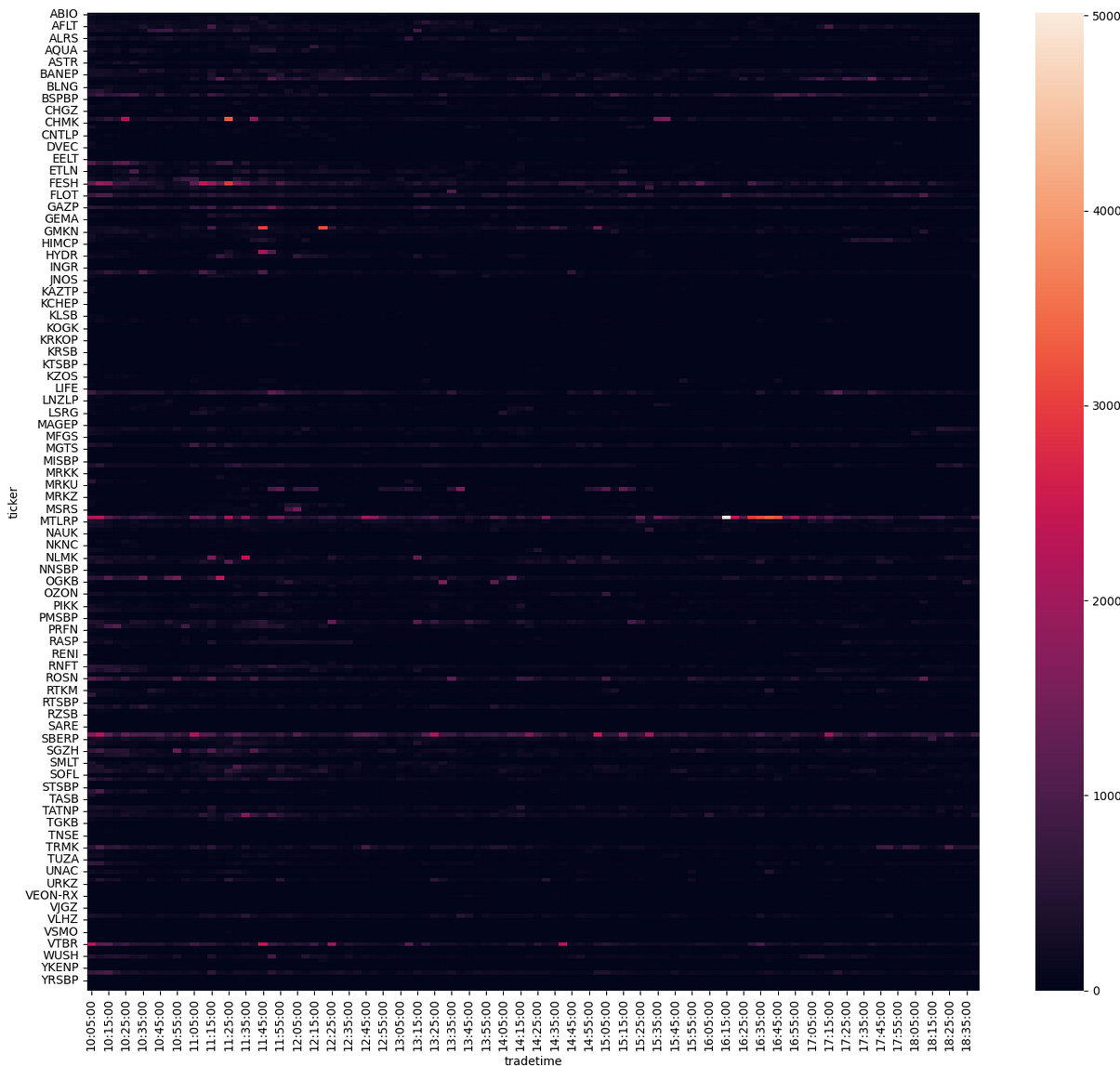
values='trades',

aggfunc='max',

fill\_value=0)

plt.figure(figsize=(17,15))

sns.heatmap(temp\_df);



### Заявки (orderstats)

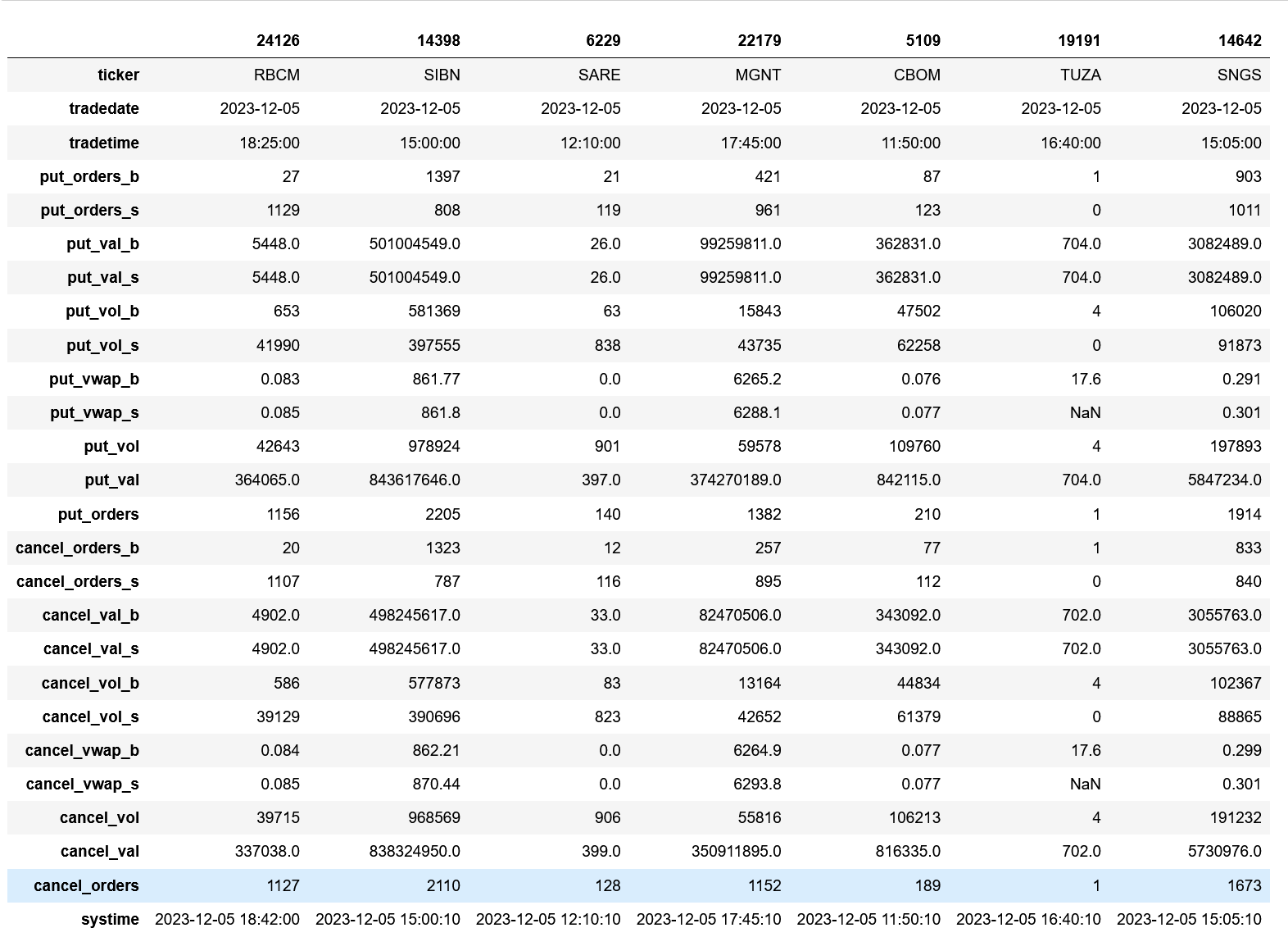
Метод orderstats возвращает метрики, рассчитанные на основе потока заявок:

* put\_orders - кол-во поставленных заявок в стакан
* put\_orders\_b - кол-во поставленных заявок (покупка)
* put\_orders\_s - кол-во поставленных заявок (продажа)
* put\_val\_b - объем заявок поставленных в стакан (покупка, руб)
* put\_val\_s - объем завок поставленных в стакан (продажа, руб)
* put\_vol\_b - объем заявок поставленных в стакан (покупка)
* put\_vol\_s - объем заявок поставленных в стакан (продажа)
* put\_vwap\_b - Средневзвешенная цена заявок (покупка)
* put\_vwap\_s - Средневзвешенная цена заявок (продажа)
* put\_vol - объем заявок, поставленных в стакан
* put\_val - объем заявок, поставленных в стакан (руб)
* cancel\_orders - кол-во снятых заявок
* cancel\_orders\_b - кол-во снятых заявок (покупка)
* cancel\_orders\_s - кол-во снятых заявок (продажа)
* cancel\_val\_b - объем снятых заявок (покупка, руб)
* cancel\_val\_s - объем снятых заявок (продажа, руб)
* cancel\_vol\_b - объем снятых заявок (покупка)
* cancel\_vol\_s - объем снятых заявок (продажа)
* cancel\_vwap\_b - Средневзвешенная цена отмененных заявок (покупка)
* cancel\_vwap\_s - Средневзвешенная цена отмененных заявок (продажа)
* cancel\_vol - объем снятых заявок
* cancel\_val - объем снятых заявок (руб)

Вернем все заявки за один день:

ord\_df = stocks.orderstats(date='2023-12-05')

ord\_df.sample(7).T

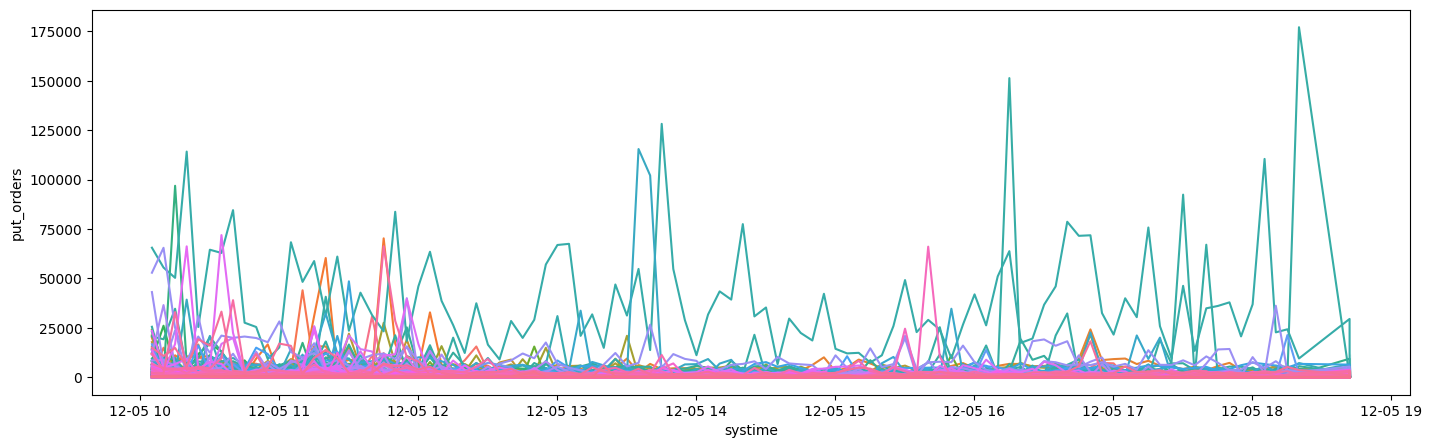


Посмотрим на страшный сон трейдера :) Введем временной ряд по всем акциям сразу :) Как и в предыдущем случае, наблюдается акцент на небольшом количестве позиций. Все остальные торгуются около нуля.

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

ax = sns.lineplot(data=ord\_df, y='put\_orders', x='systime', hue='ticker')

ax.legend\_.remove()



### Книга заказов (obstats)

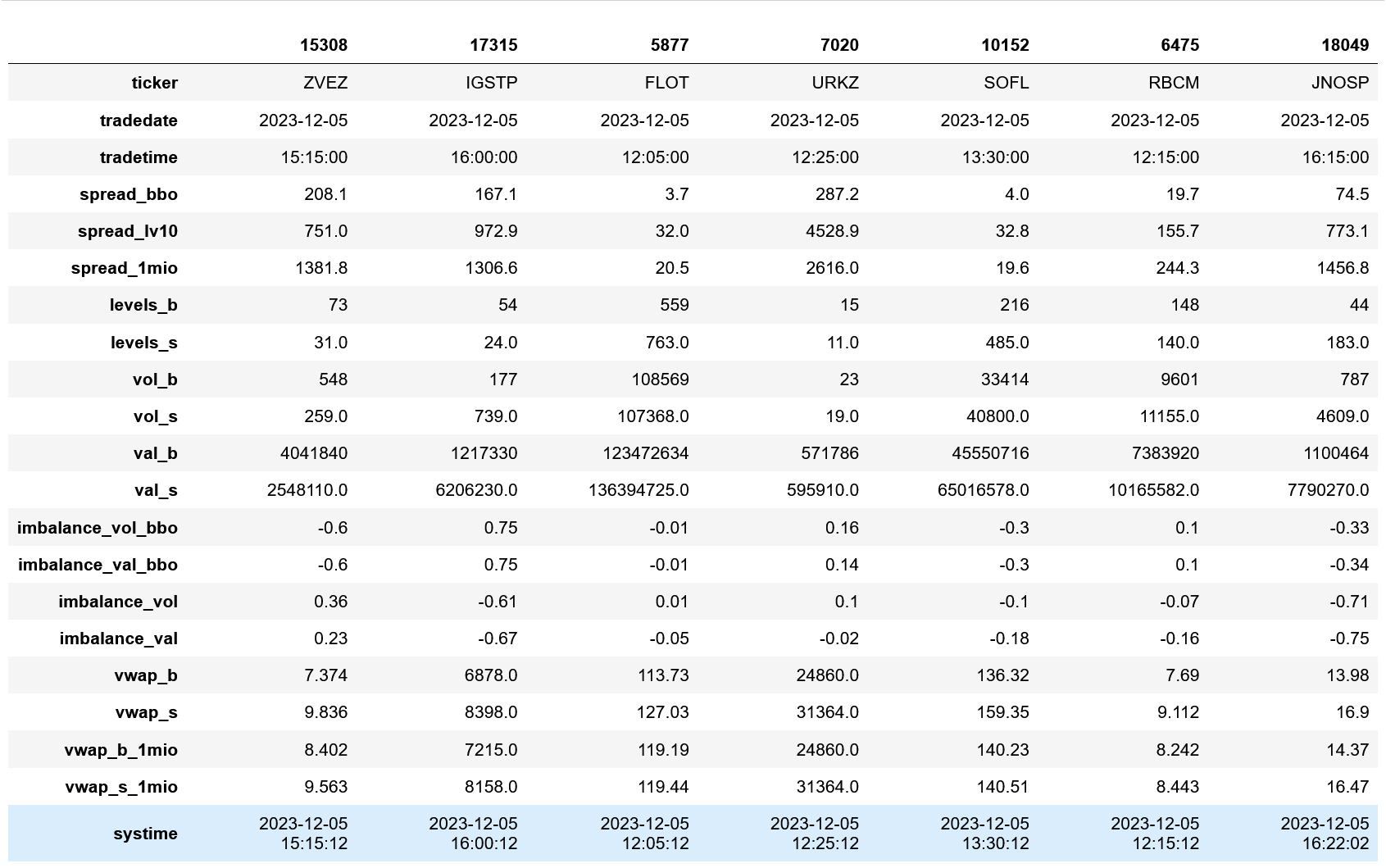
Метод obstats возвращает метрики, рассчитанные на стакане котировок:

* spread\_bbo - спред между лучшей ценой покупки и продажи
* spread\_lv10 - спред между 10ым уровнем цен покупки и продажи
* spread\_1mio - спред на 1 млн руб
* levels\_b - кол-во уровней цен в стакане (покупка)
* levels\_s - кол-во уровней цен в стакане (продажа)
* vol\_b - совокупный объем заявок в стакане на всех уровнях (покупка)
* vol\_s - совокупный объем заявок в стакане на всех уровнях (продажа)
* val\_b - совокупный объем заявок в стакане на всех уровнях (покупка), руб
* val\_s - совокупный объем заявок в стакане на всех уровнях (продажа), руб
* imbalance\_vol\_bbo - дисбаланс объема на лучших ценах
* imbalance\_val\_bbo - дисбаланс объема (руб) по лучшим ценам
* imbalance\_vol - дисбаланс объема на всем стакане (все уровни)
* imbalance\_val - дисбаланс объема (руб) на всем стакане (все уровни)
* vwap\_b - средневзвешенная цена покупки в стакане
* vwap\_s - средневзвешенная цена продажи в стакане
* vwap\_b\_1mio - цена покупки актива на 1 млн руб
* vwap\_s\_1mio - цена продажи актива на 1 млн руб

Запросим один день:

obs\_df = stocks.obstats(date='2023-12-05')

obs\_df.sample(7).T



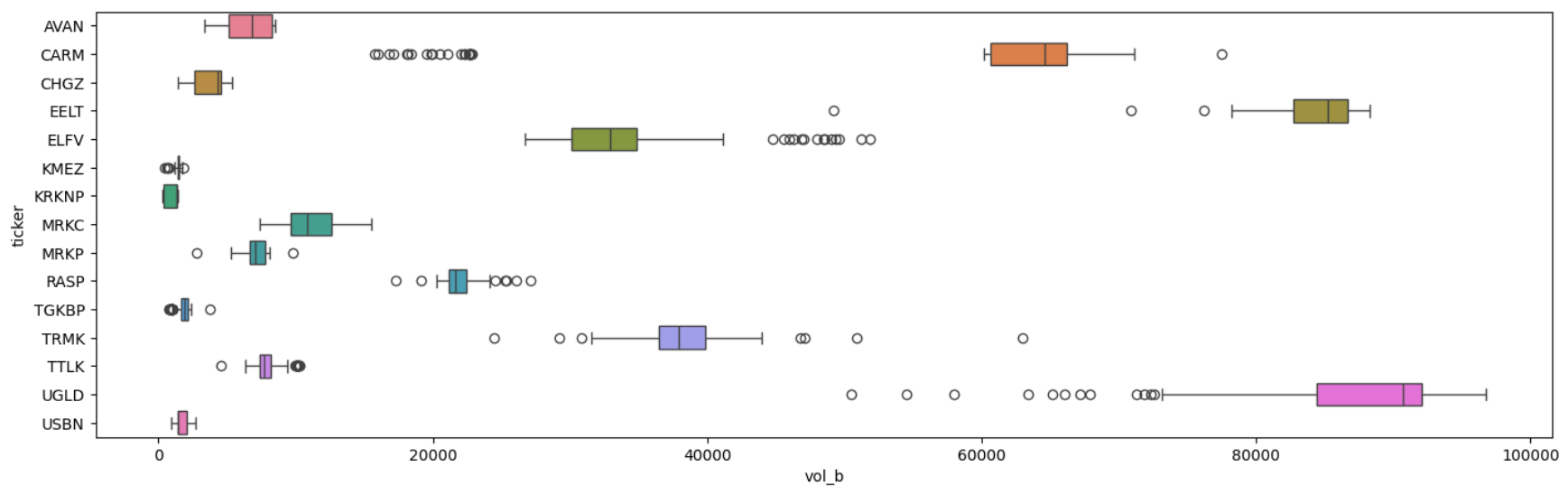
Посмотри на разброс совокупного объем заявок (покупка) в стакане для 15 случайных акций. Для этого построим ящик с усами.

rnd\_10 = ob\_df['ticker'].sample(50).unique()[:15]

temp = ob\_df[ob\_df['ticker'].isin(rnd\_10)]

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

sns.boxplot(data=temp, x='vol\_b', y='ticker', hue='ticker');



## Одна акция

Теперь посмотрим все то же самое, но только для одной акции.

### Свечи - это база!

AlgoPack может возвращать и базовые данные OHLC.

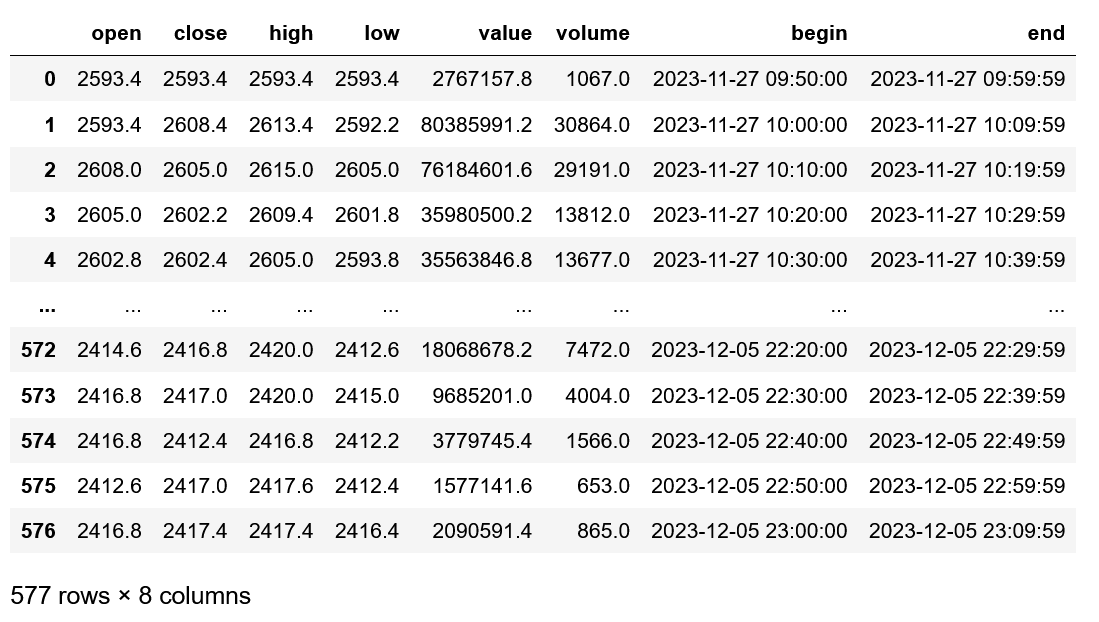
# Акции Яндекса

TICKER = 'YNDX'

one\_tick = Ticker(TICKER)

one\_cndl\_df = one\_tick.candles(date='2023-11-25', till\_date='2023-12-05', period='10m')

one\_cndl\_df



Построим канонический график по свечам:

fig = go.Figure(data=[go.Candlestick(

x=one\_cndl\_df['begin'],

open=one\_cndl\_df['open'],

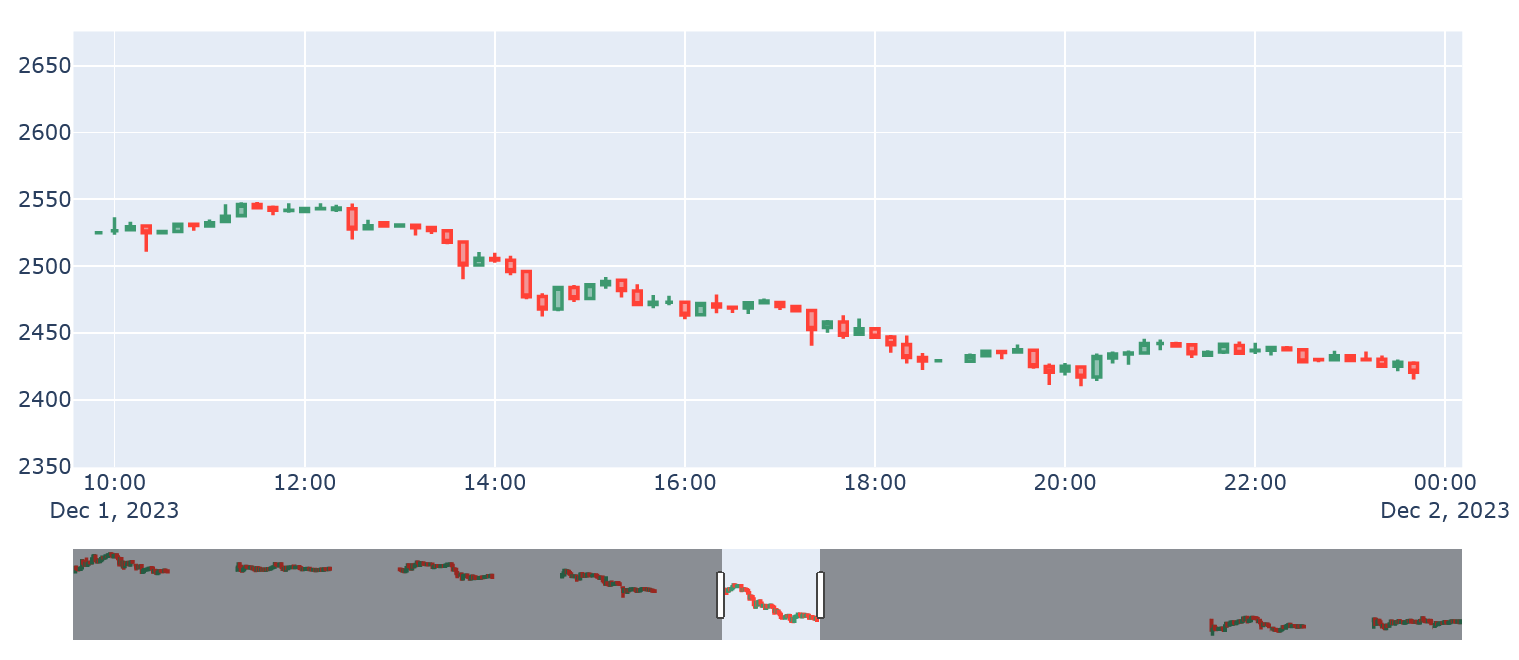
high=one\_cndl\_df['high'],

low=one\_cndl\_df['low'],

close=one\_cndl\_df['close']

)])

fig.show()

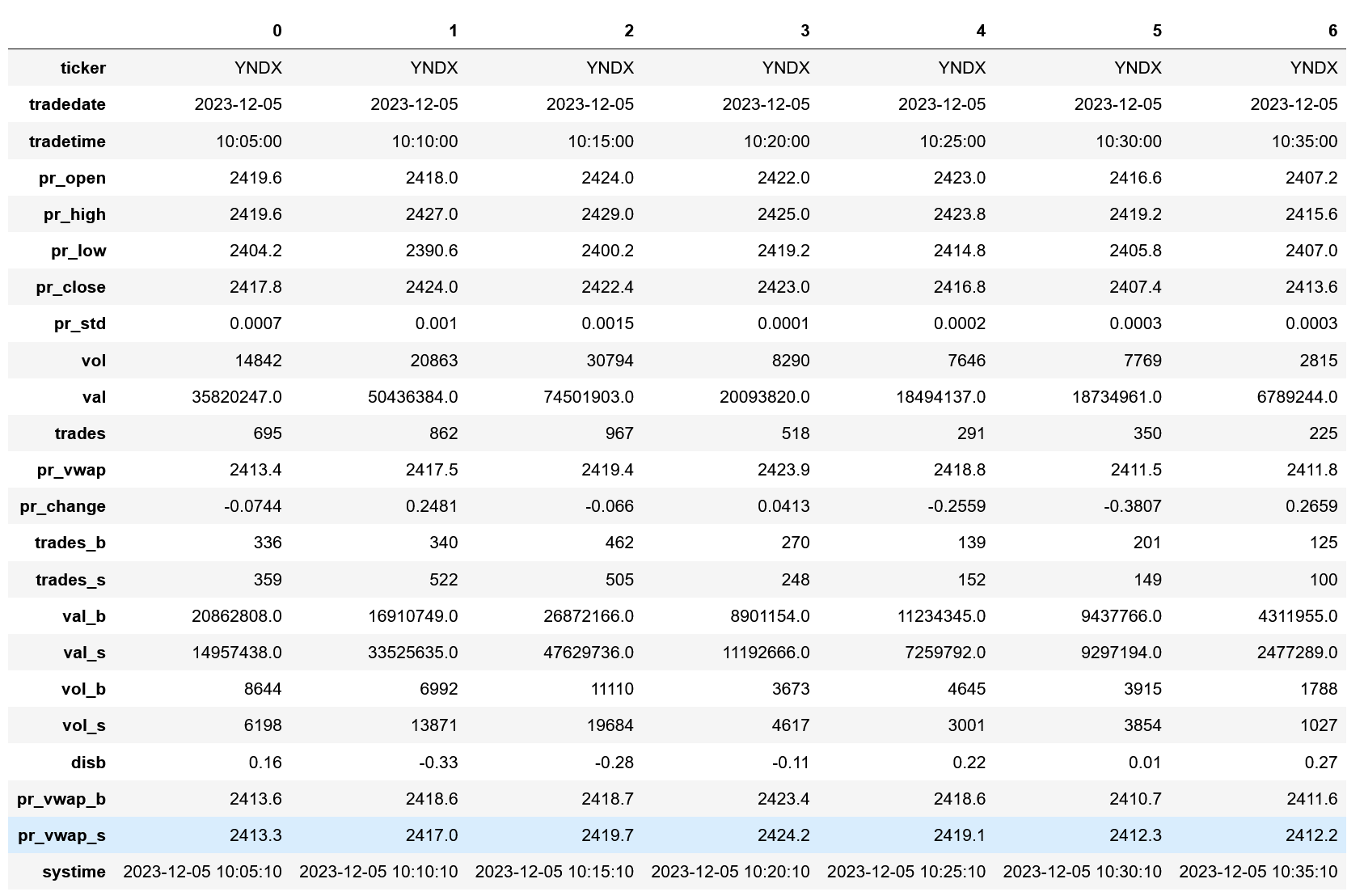


Остальные методы аналогичные рассмотренным выше, просто возвращают агрегаты только по одной акции.

### Торговая статистика

one\_trd\_df = one\_tick.tradestats(date='2023-12-05', till\_date='2023-12-05')

one\_trd\_df.head(7).T

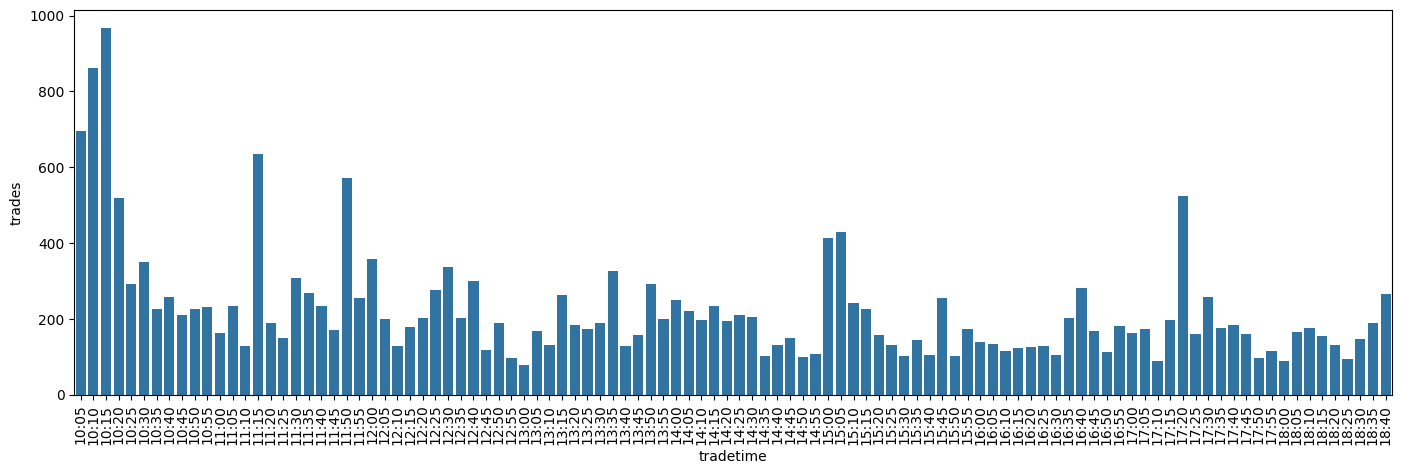


Выведем временной ряд для одной акции:

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

sns.barplot(y=one\_trd\_df['trades'], x=one\_trd\_df['tradetime'].astype(str).str[:-3])

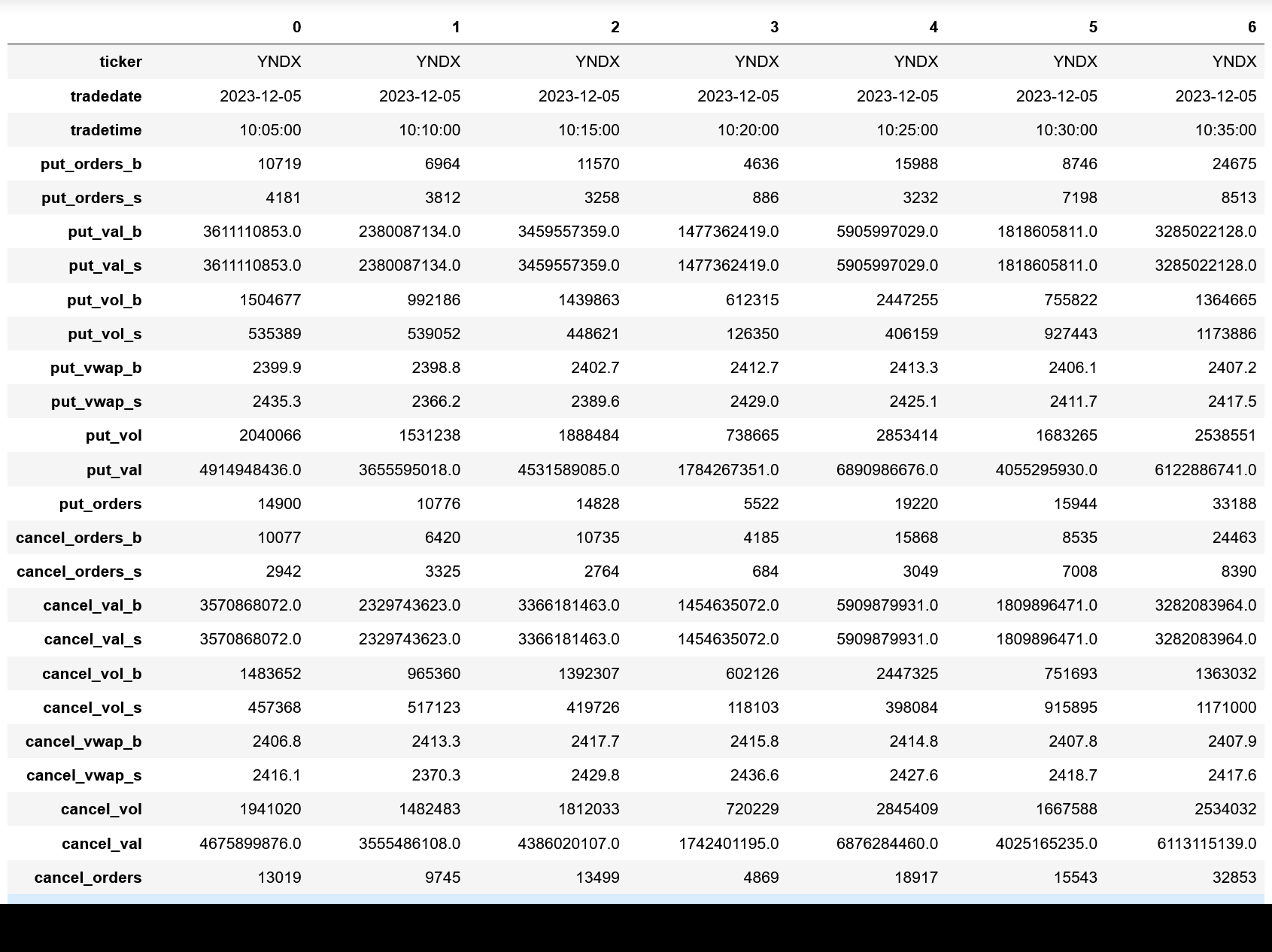
plt.xticks(rotation=90);



### Заявки

one\_ord\_df = one\_tick.orderstats(date='2023-12-05', till\_date='2023-12-05')

one\_ord\_df.head(7).T

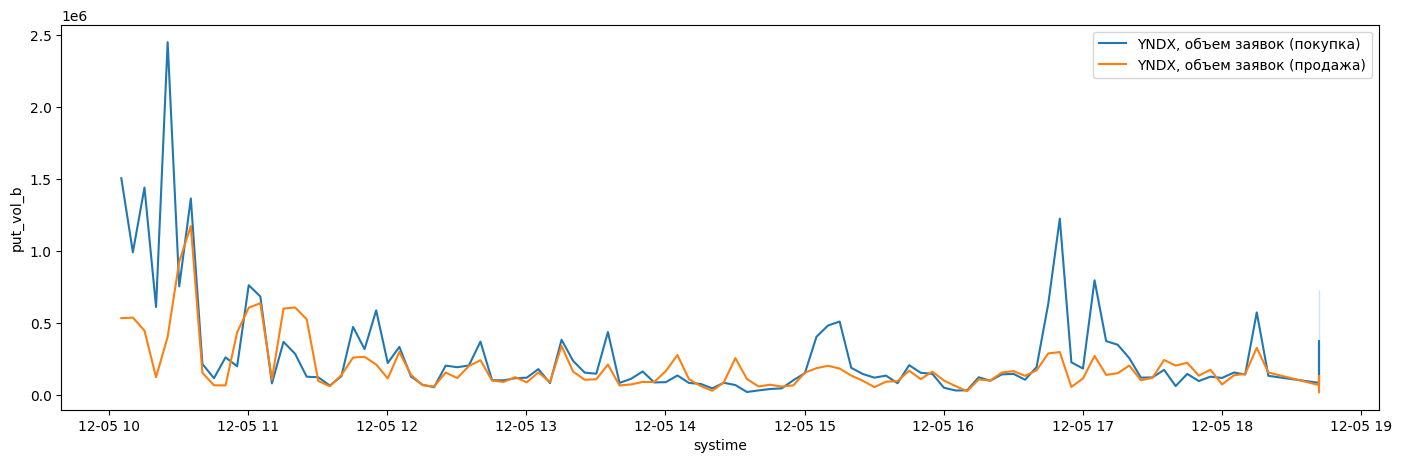


Сравним графики объема заявок по покупке и продаже.

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

sns.lineplot(y=one\_ord\_df['put\_vol\_b'], x=one\_ord\_df['systime'], label=f'{TICKER}, объем заявок (покупка)')

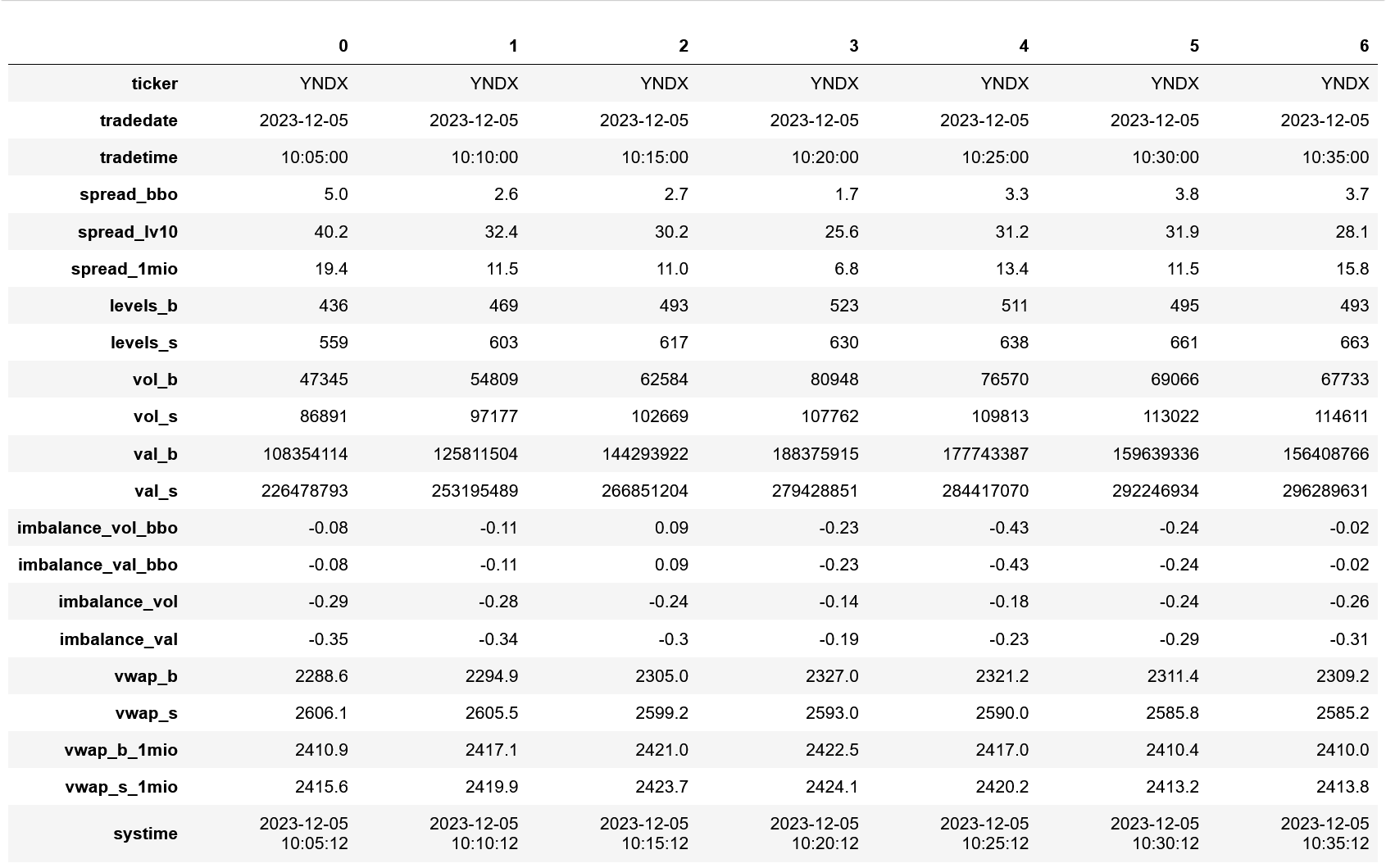
sns.lineplot(y=one\_ord\_df['put\_vol\_s'], x=one\_ord\_df['systime'], label=f'{TICKER}, объем заявок (продажа)');



### Книга заказов

one\_obs\_df = one\_tick.obstats(date='2023-12-05', till\_date='2023-12-05')

one\_obs\_df.head(7).T

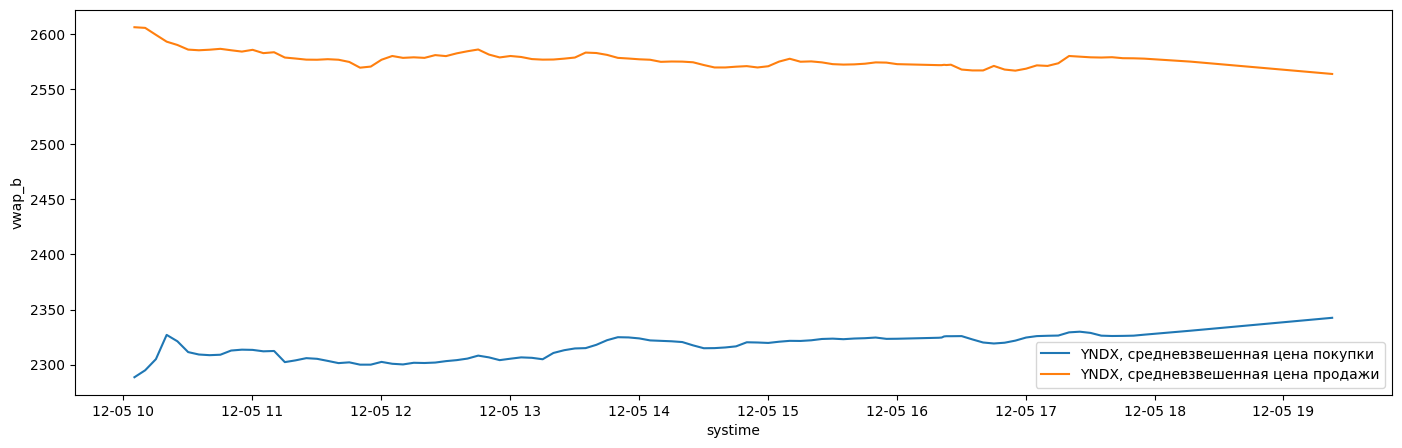


Выведем графики средневзвешенной цены по покупке и продаже.

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

sns.lineplot(y=one\_ob\_df['vwap\_b'], x=one\_ob\_df['systime'], label=f'{TICKER}, средневзвешенная цена покупки');

sns.lineplot(y=one\_ob\_df['vwap\_s'], x=one\_ob\_df['systime'], label=f'{TICKER}, средневзвешенная цена продажи');



Посмотрели немного на данные — теперь пора писать код...