Михаил Шардин
личный блог

Сегодня в 05:30

Мой первый и неудачный опыт поиска торговой стратегии для Московской биржи

Когда закончил писать механизм своего торгового робота обнаружил, что самое главное всё таки не сам механизм, а стратегия, по которой этот механизм будет работать.

Первый тесты на истории показали что с доходностью и тем более с тем как доходность портфеля компенсирует принимаемый риск (коэффициент Шарпа) проблемы, но неудачный опыт тоже опыт, поэтому решил описать его в статье.

Первый и самый важный вопрос — при помощи чего проводить тесты торговой стратегии на исторических данных? В какой программе или при помощи какой библиотеки создавать стратегию и потом прогонять её на истории?

Раз мой торговый робот создан в среде исполнения JavaScript Node.js, то и тесты в идеале должны проводится на чём-то схожем. Но забегая немного вперёд скажу что получилось по другому.

Windows? macOS? Linux?

Раз сам механизм робота кросс-платформенный, то хотелось чтобы и тесты можно было проводить при помощи кросс-платформенной утилиты. Однако когда рассматривал самые популярные программы, то обнаружилось что все программы из списка только для Windows. Кроме TradingView, который является веб-сервисом и Excel — который есть и для macOS.

Введите текст комментария

построения графиков, автоматизации стратегий и бэктестинга для акций, опционов, фьючерсов и криптовалют.

- **NinjaTrader**: торговое программное обеспечение для фьючерсов и форекс; отлично подходит для расширенного построения графиков, бэктестинга и автоматизированной торговли.
- **MetaStock**: фокусируется на техническом анализе и бэктестинге с обширными инструментами для построения графиков и индикаторов, популярен среди трейдеров акциями.
- **Wealth-Lab**: платформа, известная расширенным бэктестингом и разработкой торговых стратегий с мощной поддержкой портфелей из нескольких активов.
- **TradingView**: удобная в использовании платформа для построения графиков с социальными функциями; отлично подходит для технического анализа, обмена идеями и базового бэктестинга стратегий.
- **RealTest**: легкое программное обеспечение для бэктестинга и разработки стратегий, известное своей скоростью и простотой, ориентированное на системных трейдеров.
- **Neuroshell Trader**: специализируется на прогнозном моделировании и анализе на основе нейронных сетей; идеально подходит для трейдеров, интересующихся машинным обучением.
- **TSLab**: платформа позволяет разрабатывать, тестировать и оптимизировать торговые системы без необходимости глубокого знания программирования.
- **The Zorro Project**: бесплатная, легкая и скриптовая платформа, предназначенная для автоматизированной торговли, бэктестинга и исследований, популярная среди алгоритмических трейдеров.
- и даже **Microsoft Excel**: универсальный инструмент для работы с электронными таблицами, часто используемый для анализа портфеля, пользовательского бэктестинга и организации данных в торговле.

Ни один из этих вариантов мне не приглянулся из-за отсутствия кросс-платформенности или этот вариант был Экселем.

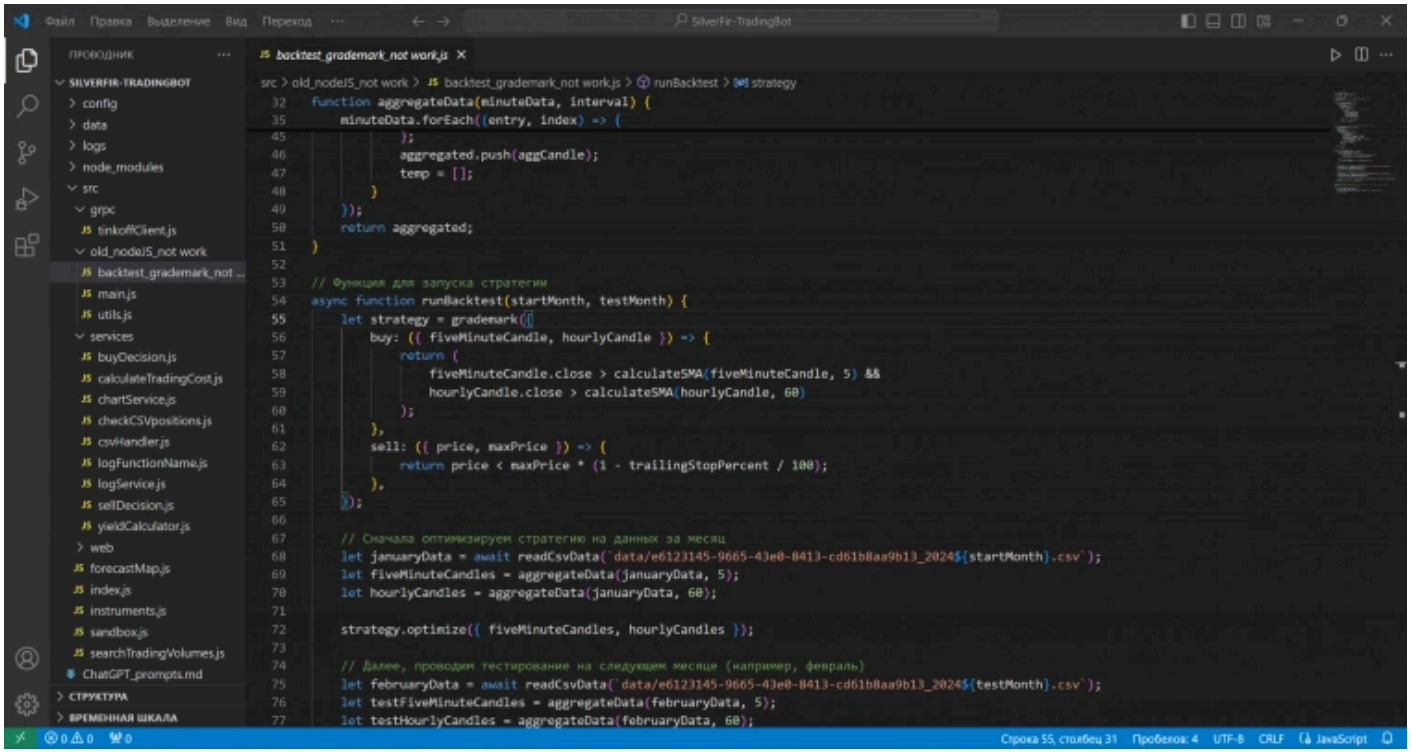
Node.js библиотеки — не смог ❌

После этого стал смотреть библиотеки для Node.js. Выбор оказался небольшой и более-менее живыми мне показались:

grademark: github.com/Grademark/grademark
Библиотека Node.js для бэктестинга торговых стратегий на исторических данных.

Fugle Backtest: github.com/fugle-dev/fugle-backtest-node

Библиотека Node.js для торговли криптовалютой, которая предоставляет унифицированный API для подключения и торговли на нескольких криптовалютных биржах, поддерживая как торговлю в реальном времени, так и доступ к историческим данным.



Для Grademark набросал через ChatGPT конкретный пример использования:

```
<code>const fs = require('fs');
const csvParser = require('csv-parser');
const grademark = require('grademark');

// Параметры стратегии
let trailingStopPercent = 1; // Процент падения для трейлинг-стопа
let buyThreshold = 1; // Минимальный процент для покупки

// Функция для чтения данных из CSV-файла
function readCsvData(filePath) {
    return new Promise((resolve, reject) => {
        let data = [];
        fs.createReadStream(filePath)
            .pipe(csvParser())
            .on('data', (row) => {
                // Преобразуем каждую строку CSV в объект данных
                time: new Date(row[1]),
```

```
                    low: parseFloat(row[4]),
                    close: parseFloat(row[5]),
                    volume: parseInt(row[6])
                });
            })
            .on('end', () => resolve(data))
            .on('error', reject);
    });
}

// Функция для агрегирования минутных данных в 5-минутные и часовые свечи
function aggregateData(minuteData, interval) {
    const aggregated = [];
    let temp = [];
    minuteData.forEach((entry, index) => {
        temp.push(entry);
        if ((index + 1) % interval === 0) {
            const aggCandle = {
                open: temp[0].open,
                high: Math.max(...temp.map(t => t.high)),
                low: Math.min(...temp.map(t => t.low)),
                close: temp[temp.length - 1].close,
                volume: temp.reduce((acc, t) => acc + t.volume, 0),
                time: temp[temp.length - 1].time
            };
            aggregated.push(aggCandle);
            temp = [];
        }
    });
    return aggregated;
}

// Функция для запуска стратегии
async function runBacktest(startMonth, testMonth) {
    let strategy = grademark({
        buy: ({ fiveMinuteCandle, hourlyCandle }) => {
```

```
        },
        sell: ({ price, maxPrice }) => {
            return price < maxPrice * (1 - trailingStopPercent / 100);
        },
    });

    // Сначала оптимизируем стратегию на данных за месяц
    let januaryData = await readCsvData(`data/e6123145-9665-43e0-8413-cd61b8aa9b
    let fiveMinuteCandles = aggregateData(januaryData, 5);
    let hourlyCandles = aggregateData(januaryData, 60);

    strategy.optimize({ fiveMinuteCandles, hourlyCandles });

    // Далее, проводим тестирование на следующем месяце (например, февраль)
    let februaryData = await readCsvData(`data/e6123145-9665-43e0-8413-cd61b8aa9
    let testFiveMinuteCandles = aggregateData(februaryData, 5);
    let testHourlyCandles = aggregateData(februaryData, 60);

    let testResults = strategy.run({ fiveMinuteCandles: testFiveMinuteCandles, h

    console.log('Результаты теста:', testResults);
}

// Пример вызова функции для тестирования
runBacktest('09', '10'); // Оптимизация на сентябрьских данных и тестирование на

// Функция для вычисления скользящей средней (SMA)
function calculateSMA(candles, period) {
    if (candles.length < period) return 0;
    const sum = candles.slice(-period).reduce((acc, candle) => acc + candle.clos
    return sum / period;
}</code>
```

При этом криптовалюты мне не подходили, Grademark почему-то не смог установить, а Fugle Backtest не приглянулся.

Python библиотеки — заработало! ✅

популярные индикаторы и метрики.
❌ 4 года не обновлялась.

Backtrader github.com/mementum/backtrader
Одна из самых популярных и многофункциональных библиотек для бэктестинга. Поддерживает несколько активов, таймфреймов, индикаторов и оптимизацию стратегий.

PyAlgoTrade github.com/gbeced/pyalgotrade
Простая библиотека бэктестинга со встроенной поддержкой технических индикаторов и создания базовой стратегии.
❌ Этот репозиторий был заархивирован владельцем 13 ноября 2023 г.

Zipline github.com/quantopian/zipline
Разработанная Quantopian (теперь поддерживаемая сообществом), Zipline — это надежная библиотека бэктестинга, ориентированная на событийно-управляемое бэктестирование, используемая профессионалами.
❌ 4 года не обновлялась.

QuantConnect/Lean github.com/QuantConnect/Lean
Движок с открытым исходным кодом, лежащий в основе QuantConnect; поддерживает бэктестинг и торговлю в реальном времени для нескольких классов активов.

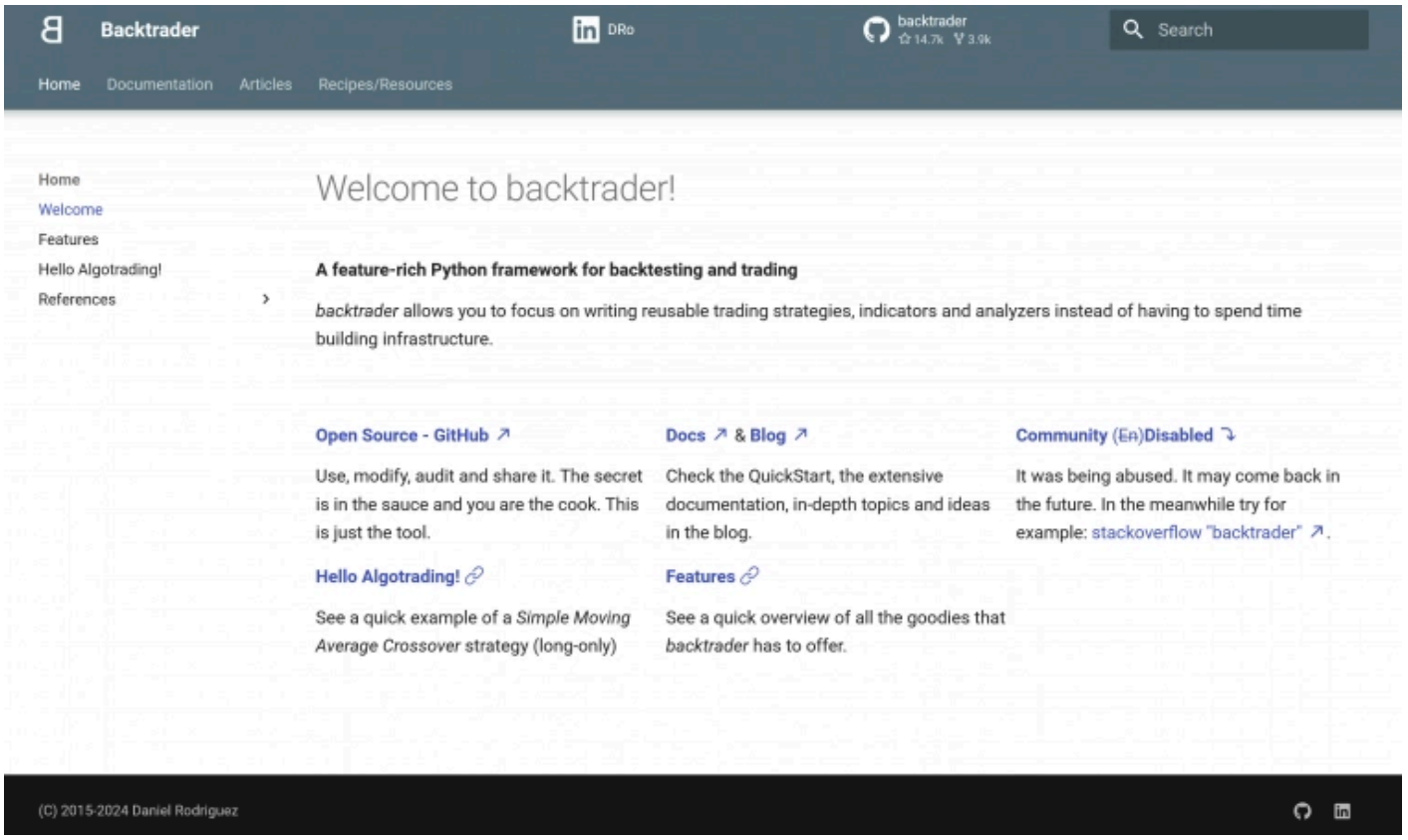
VectorBT github.com/polakowo/vectorbt
Разработан для быстрого векторизованного бэктестинга и анализа стратегий непосредственно на Pandas DataFrames.

Fastquant github.com/enzoampil/fastquant
Удобная библиотека бэктестинга, разработанная для быстрого тестирования с минимальной настройкой, вдохновленная Prophet от Facebook.
❌ 3 года не обновлялась.

MibianLib github.com/yassinemaaroufi/MibianLib
Фокусируется на ценообразовании и волатильности опционов, а не на полном бэктестинге, но полезен для стратегий, связанных с опционами.
❌ 11 лет не обновлялась.

Сначала выбрал использовать Backtesting.py, потому что она упоминалась на многих сайтах, но уже на первоначальном этапе использования стали вылазит проблемы. Ошибка возникла из-за несоответствия в том, как новые версии `pandas` обрабатывают метод `get_loc()`. Аргумент `method='nearest'` больше не поддерживается в последних версиях pandas. Эта проблема связана с тем, как библиотека Backtesting.py взаимодействует с

Следующий в списке был **Backtrader** — с ним и продолжил работать.



Идея моей торговой стратегии 💡

Хотя считается что торговая стратегия необязательно должна быть «человекочитаемой» — это вполне может быть результат обучения алгоритма, основанного на интеллектуальных технологиях (нейросети, машинное обучение и т.п.), но я решил начать с простого.

Мои условия:

1. Торговать только в лонг (длинная позиция) — покупать акции с целью их последующей продажи по более высокой цене.
2. Торговать только 15 лучших акций по объему на Московской бирже.
3. Использовать два разных таймфрейма для тестов — это временные интервалы на которых отображается движение цен на графике финансового инструмента. Планирую использовать 5 минут и час. Это из-за того что моё АПИ медленное.

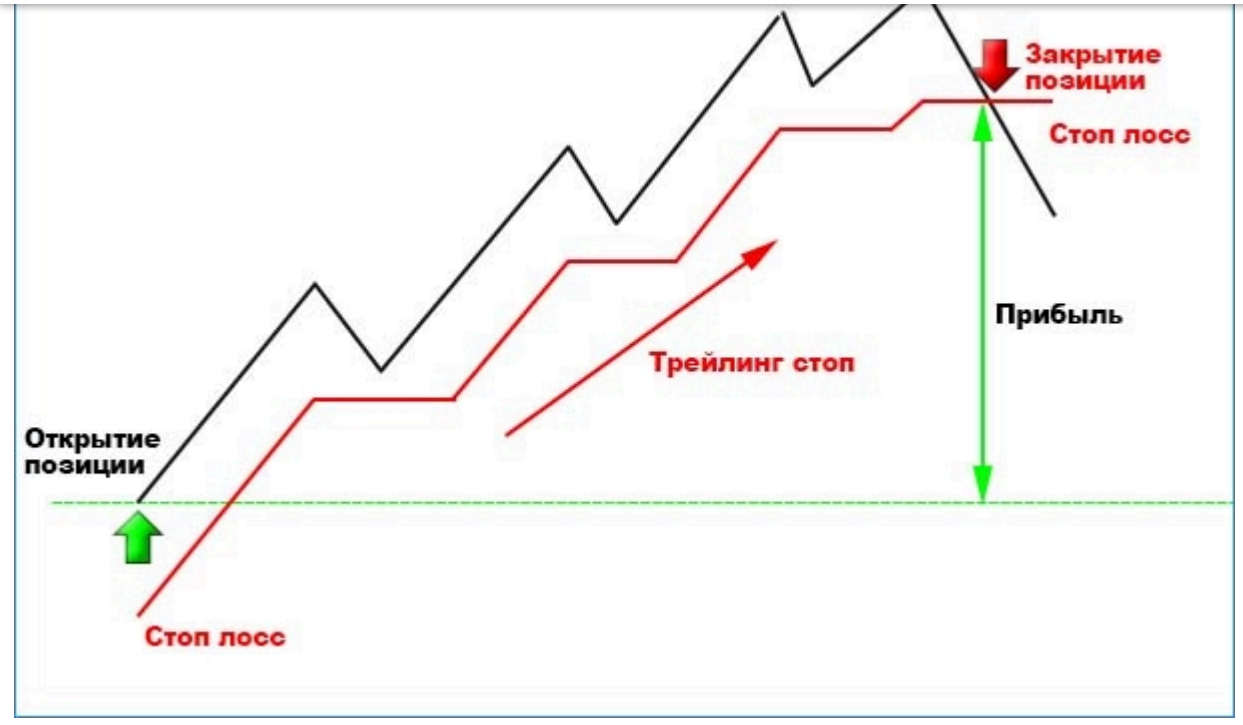
Моя торговая стратегия основана на пересечении скользящих средних двух разных таймфреймов со скользящим стоп-лоссом для продажи.

Условие покупки представляет собой комбинацию двух пересечений скользящих средних:

1. Краткосрочное подтверждение: цена закрытия на пятиминутном интервале выше

Требуя выполнения обоих этих условий, гарантирую что акция будет иметь бычий импульс как на коротких, так и на длинных таймфреймах перед входом в позицию. Такое выравнивание двух таймфреймов помогает избегать покупок во время временного шума или незначительных колебаний на более коротком таймфрейме, отфильтровывая менее стабильные движения.

Условие продажи: трейлинг стоп, который предназначен для защиты прибыли и ограничения риска падения. Как работает лучше всего показано на картинке:



Бэктестинг моей торговой стратегии с помощью библиотеки backtrader на Python

Моя, описанная выше стратегия для двух таймфреймов на нескольких бумагах, выглядит в библиотеке backtrader на Python следующим образом:

strategy0_ma_5min_hourly.py:

```
<code>import sys
sys.stdout.reconfigure(encoding='utf-8')

import backtrader as bt

# Стратегия скользящие средние на двух разных временных интервалах
class MovingAveragesOnDifferentTimeIntervalsStrategy(bt.Strategy):
    params = (
        ('ma_period_5min', 30),   # Период для скользящей средней на 5-минутках
        ('ma_period_hourly', 45), # Период для скользящей средней на часовом инт
        ('trailing_stop', 0.03)   # Процент для трейлинг-стопа
    )

    def __init__(self):
        print(f"\nРасчет для параметров: {self.params.ma_period_5min} / {self.pa
```

```
        # Для каждого инструмента добавляем скользящие средние по разным интерва
        for i, data in enumerate(self.datas):
            if i % 2 == 0:  # Четные индексы - 5-минутные данные
                ticker = data._name.replace('_5min', '')
                self.ma_5min[ticker] = bt.indicators.SimpleMovingAverage(data.cl
            else:  # Нечетные индексы - часовые данные
                ticker = data._name.replace('_hourly', '')
                self.ma_hourly[ticker] = bt.indicators.SimpleMovingAverage(data.

        # Переменные для отслеживания максимальной цены после покупки по каждому
        self.buy_price = {}
        self.max_price = {}
        self.order = {}  # Словарь для отслеживания ордеров по каждому инструмен

    def next(self):
        # Для каждого инструмента проверяем условия покупки и продажи
        for i in range(0, len(self.datas), 2):  # Проходим по 5-минутным данным
            ticker = self.datas[i]._name.replace('_5min', '')
            data_5min = self.datas[i]
            data_hourly = self.datas[i + 1]

            # Проверяем, есть ли открытый ордер для этого инструмента
            if ticker in self.order and self.order[ticker]:
                continue  # Пропускаем, если есть открытый ордер

            # Проверяем условия покупки:
            # цена на 5 мин таймфрейме выше скользящей средней на 5 мин + часова
            if not self.getposition(data_5min):  # Открываем сделку только если
                if data_5min.close[0] > self.ma_5min[ticker][0] and data_hourly.
                    self.order[ticker] = self.buy(data=data_5min)
                    self.buy_price[ticker] = data_5min.close[0]
                    self.max_price[ticker] = self.buy_price[ticker]

                    # Получаем текущий тикер и дату покупки
                    buy_date = data_5min.datetime.date(0)
```

```
            elif self.getposition(data_5min):
                current_price = data_5min.close[0]

                # Обновляем максимальную цену, если текущая выше
                if current_price > self.max_price[ticker]:
                    self.max_price[ticker] = current_price

                # Рассчитываем уровень стоп-лосса
                stop_loss_level = self.max_price[ticker] * (1 - self.params.trai

                # Проверяем условие для продажи по трейлинг-стопу
                if current_price < stop_loss_level:
                    self.order[ticker] = self.sell(data=data_5min)
                    sell_date = data_5min.datetime.date(0)
                    sell_time = data_5min.datetime.time(0)
                    print(f"{sell_date} в {sell_time}: продажа за {current_price

    # Обрабатываем уведомления по ордерам
    def notify_order(self, order):
        ticker = order.data._name.replace('_5min', '')

        if order.status in [order.Completed, order.Canceled, order.Margin]:
            self.order[ticker] = None  # Очищаем ордер после завершения</code>
```

Сделал переключатель одиночный тест или оптимизация: `singleTest / optimization` для основного файла запуска: `SingleTestOrOptimization = "optimization"`

Основной файл запуска main.py:

```
<code>import sys
import time
sys.stdout.reconfigure(encoding='utf-8')

from datetime import datetime
from src.data_loader import load_data_for_ticker, load_ticker_mapping

import pandas as pd
import backtrader as bt
import backtrader.analyzers as btanalyzers
```

```
# отобразить имена всех столбцов в большом фреймворке данных pandas
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)

# Начало времени
start_time = time.perf_counter()

# Путь к JSON файлу с сопоставлениями
mapping_file = "./data/+mappings.json"

# Загрузка сопоставлений тикеров
ticker_mapping = load_ticker_mapping(mapping_file)

# Промежуточное время выполнения
total_end_time = time.perf_counter()
elapsed_time = total_end_time - start_time
print(f"Промежуточное время выполнения: {elapsed_time:.4f} секунд.")

current_time = datetime.now().strftime("%Y-%m-%d %H-%M") # Генерируем текущее вр

# Следующая часть кода запускается только если это основной модуль
if __name__ == '__main__':  # Исправление для работы с multiprocessing

    # Создаем объект Cerebro
    cerebro = bt.Cerebro(optreturn=False)

    # Получаем количество бумаг в ticker_mapping.items()
    num_securities = len(ticker_mapping.items())

    # Рассчитываем процент капитала на одну бумагу
    percent_per_security = 100 / num_securities
    print(f"Процент капитала на одну бумагу: {percent_per_security:.2f}%")

    # Условия капитала
    cerebro.broker.set_cash(100000)  # Устанавливаем стартовый капитал
    cerebro.broker.setcommission(commission=0.005)  # Комиссия 0.5%
```

```
        print(f"Загружаем данные для {ticker}")

        # Загрузка данных с таймфреймами 5 минут и час
        data_5min, data_hourly = load_data_for_ticker(ticker)

        # Пропуск, если данные не были загружены
        if data_5min is None or data_hourly is None:
            continue

        # Добавляем 5-минутные данные в Cerebro
        data_5min_bt = bt.feeds.PandasData(dataname=data_5min, timeframe=bt.Time
        cerebro.adddata(data_5min_bt, name=f"{ticker}_5min")

        # Добавляем часовые данные в Cerebro
        data_hourly_bt = bt.feeds.PandasData(dataname=data_hourly, timeframe=bt.

        # Совмещаем графики 5 минут и часа на одном виде
        data_hourly_bt.plotinfo.plotmaster = data_5min_bt  # Связываем графики
        data_hourly_bt.plotinfo.sameaxis = True            # Отображаем на той ж
        cerebro.adddata(data_hourly_bt, name=f"{ticker}_hourly")


    # Переключатель одиночный тест или оптимизация
    SingleTestOrOptimization = "optimization" # singleTest / optimization

    if SingleTestOrOptimization == "singleTest":
        print(f"{current_time} Проводим одиночный тест стратегии.")

        # Добавляем стратегию для одичного теста MovingAveragesOnDifferentTimeIn
        cerebro.addstrategy(MovingAveragesOnDifferentTimeIntervalsStrategy,
                            ma_period_5min = 30,    # Период для скользящей средней на
                            ma_period_hourly = 45,  # Период для скользящей средней на
                            trailing_stop = 0.03)   # Процент для трейлинг-стопа

        # Writer только для одиночного теста для вывода результатов в CSV-файл
        cerebro.addwriter(bt.WriterFile, csv=True, out=f"./results/{current_time
```

```
        cerebro.addanalyzer(btanalyzers.Returns, _name="returns")
        cerebro.addanalyzer(btanalyzers.SharpeRatio, _name='sharpe_ratio', timef
        cerebro.addanalyzer(btanalyzers.SQN, _name='sqn')
        cerebro.addanalyzer(btanalyzers.PyFolio, _name='PyFolio')

        # Запуск тестирования
        results = cerebro.run(maxcpus=1)  # Ограничение одним ядром для избежани

        # Выводим результаты анализа одиночного теста
        print(f"\nОкончательная стоимость портфеля: {cerebro.broker.getvalue()}"
        returnsAnalyzer = results[0].analyzers.returns.get_analysis()
        print(f"Годовая/нормализованная доходность: {returnsAnalyzer['rnorm100']
        drawdownAnalyzer = results[0].analyzers.drawdown.get_analysis()
        print(f"Максимальное значение просадки: {drawdownAnalyzer['max']['drawdo
        trade_analyzer = results[0].analyzers.trade_analyzer.get_analysis()
        print(f"Всего сделок: {trade_analyzer.total.closed} шт.")
        print(f"Выигрышные сделки: {trade_analyzer.won.total} шт.")
        print(f"Убыточные сделки: {trade_analyzer.lost.total} шт.")
        sharpe_ratio = results[0].analyzers.sharpe_ratio.get_analysis().get('sha
        print(f"Коэффициент Шарпа: {sharpe_ratio}")
        sqnAnalyzer = results[0].analyzers.sqn.get_analysis().get('sqn')
        print(f"Мера доходности с поправкой на риск: {sqnAnalyzer}")

        # Время выполнения
        total_end_time = time.perf_counter()
        elapsed_time = (total_end_time - start_time) / 60
        print(f"\nВремя выполнения: {elapsed_time:.4f} минут.")

        # Построение графика для одиночного теста
        cerebro.plot()


    else:
        print(f"{current_time} Проводим оптимизацию статегии.")

        # Оптимизация стратегии start_date = 2024-10_MovingAveragesOnDifferentTi
```

```
print(f"\nКоличество варинатов оптимизации: {(( (61-10)/10 * (61-15))/2

# Добавляем анализаторы
cerebro.addanalyzer(btanalyzers.TradeAnalyzer, _name='trade_analyzer')
cerebro.addanalyzer(btanalyzers.DrawDown, _name="drawdown")
cerebro.addanalyzer(btanalyzers.Returns, _name="returns")
cerebro.addanalyzer(btanalyzers.SharpeRatio, _name='sharpe_ratio', timef
cerebro.addanalyzer(btanalyzers.SQN, _name='sqn')

# Запуск тестирования
results = cerebro.run(maxcpus=1)  # Ограничение одним ядром для избежани

# Выводим результаты оптимизации
par_list = [ [
            # MovingAveragesOnDifferentTimeIntervalsStrategy
            x[0].params.ma_period_5min,
                x[0].params.ma_period_hourly,
            x[0].params.trailing_stop,

                x[0].analyzers.trade_analyzer.get_analysis().pnl.net.tot
                x[0].analyzers.returns.get_analysis()['rnorm100'],
                x[0].analyzers.drawdown.get_analysis()['max']['drawdown'
                x[0].analyzers.trade_analyzer.get_analysis().total.close
                x[0].analyzers.trade_analyzer.get_analysis().won.total,
                x[0].analyzers.trade_analyzer.get_analysis().lost.total,
                x[0].analyzers.sharpe_ratio.get_analysis()['sharperatio'
                x[0].analyzers.sqn.get_analysis().get('sqn')
        ] for x in results]

# MovingAveragesOnDifferentTimeIntervalsStrategy
par_df = pd.DataFrame(par_list, columns = ['ma_period_5min', 'ma_period_

# Формируем имя файла с текущей датой и временем
filename = f"./results/{current_time}_optimization.csv"
# Сохраняем DataFrame в CSV файл с динамическим именем
par_df.to_csv(filename, index=False)
```

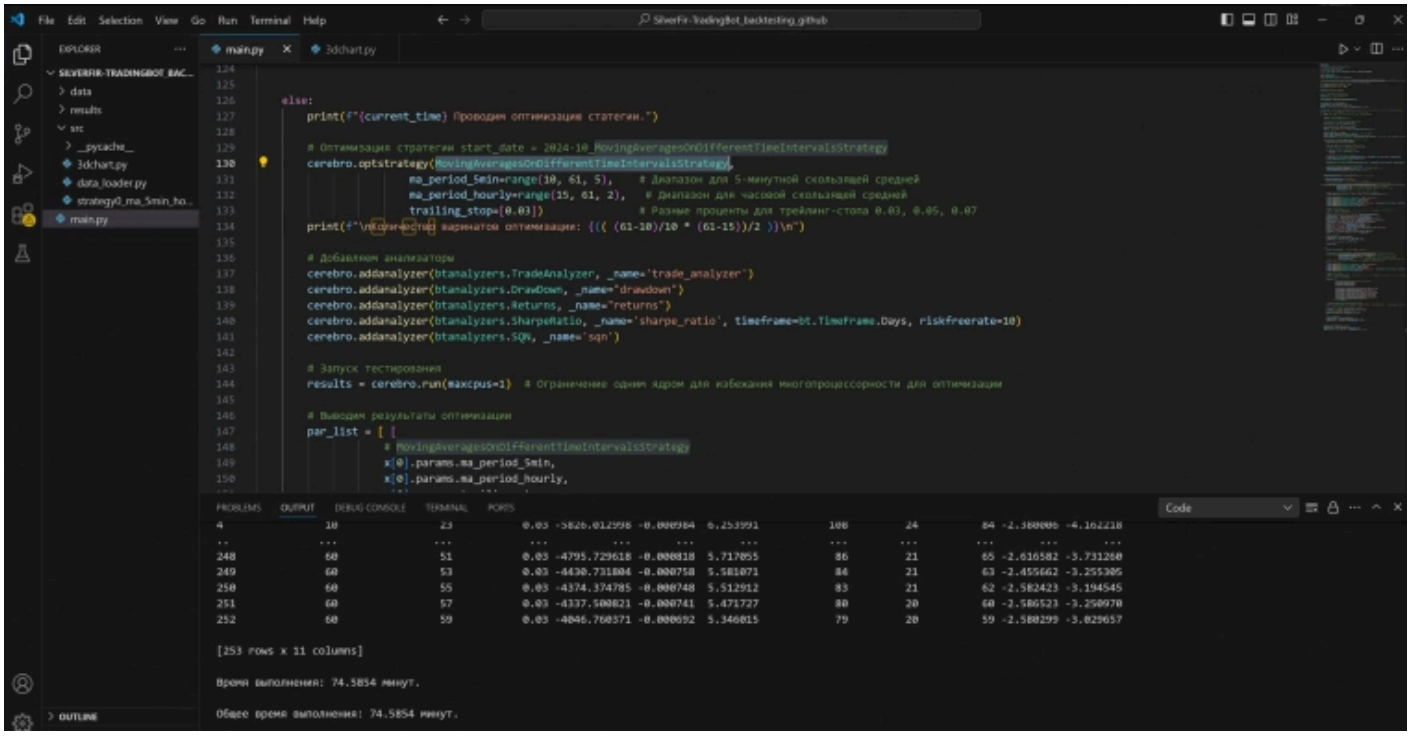
```
        elapsed_time = (total_end_time - start_time) / 60
        print(f"\nВремя выполнения: {elapsed_time:.4f} минут.")


    # Общее время выполнения
    total_end_time = time.perf_counter()
    elapsed_time = (total_end_time - start_time) / 60
    print(f"\nОбщее время выполнения: {elapsed_time:.4f} минут.")</code>
```

В данные загрузил котировки за октябрь 2024:

1. AFLT_1hour.csv
2. AFLT_5min.csv
3. EUTR_1hour.csv
4. EUTR_5min.csv
5. GAZP_1hour.csv
6. GAZP_5min.csv
7. MTLR_1hour.csv
8. MTLR_5min.csv
9. RNFT_1hour.csv
10. RNFT_5min.csv
11. ROSN_1hour.csv
12. ROSN_5min.csv
13. RUAL_1hour.csv
14. RUAL_5min.csv
15. SBER_1hour.csv
16. SBER_5min.csv
17. SGZH_1hour.csv
18. SGZH_5min.csv
19. SNGSP_1hour.csv
20. SNGSP_5min.csv
21. UWGN_1hour.csv
22. UWGN_5min.csv
23. VKCO_1hour.csv
24. VKCO_5min.csv
25. VTBR_1hour.csv
26. VTBR_5min.csv

Время выполнения оптимизации для таких параметров составило 74 минуты:

```
ma_period_5min=range(10, 61, 5),     # Диапазон для 5-минутной ск
ma_period_hourly=range(15, 61, 2),   # Диапазон для часовой скол
trailing_stop=[0.03])                # Разные проценты для трейли
```



Для того чтобы визуально представить результаты оптимизации написал модуль, который строит трехмерный график.

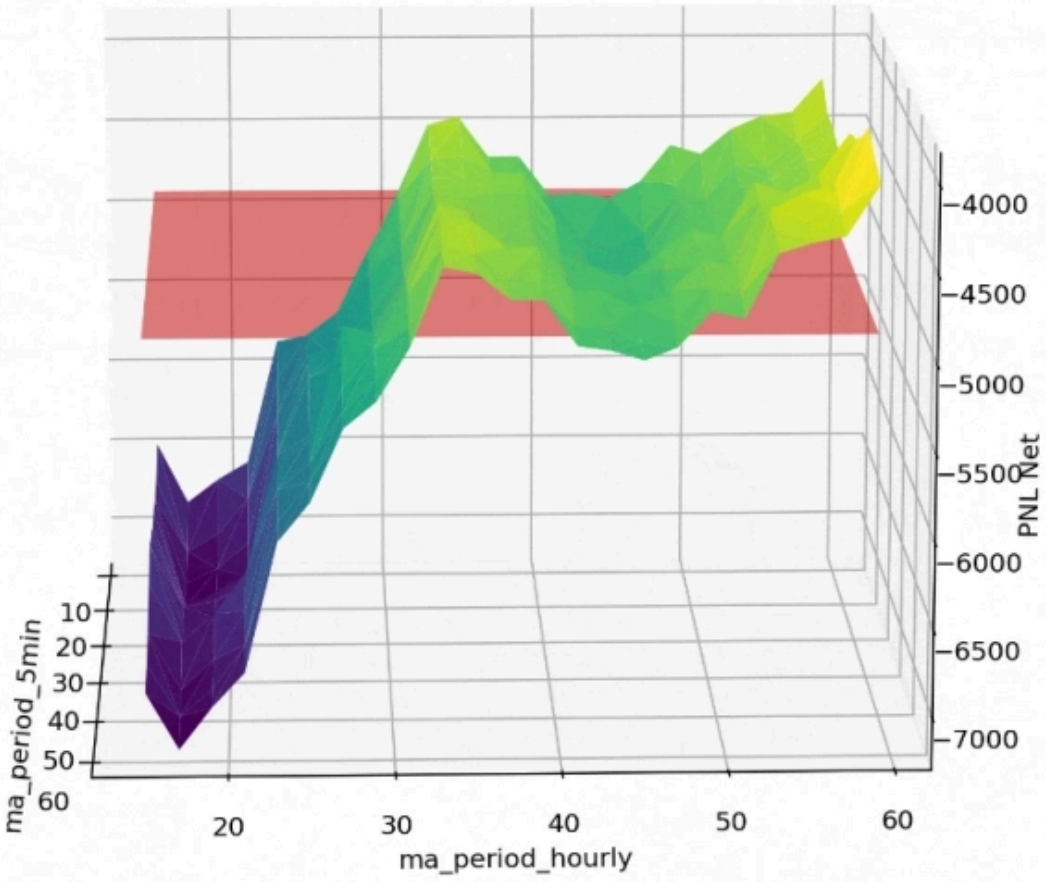
Модуль 3dchart.py:

```
<code>import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
from matplotlib.widgets import Slider
from datetime import datetime

# Чтение данных из CSV файла
data = pd.read_csv('./results/2024-11-12 16-12_optimization_2024-10_MovingAverag

parameter1 = 'ma_period_5min'
parameter2 = 'ma_period_hourly'

# Извлечение необходимых колонок для построения графика
```

```
# Создание 3D-графика
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Построение поверхности с использованием триангуляции
surf = ax.plot_trisurf(x, y, z, cmap='viridis', edgecolor='none')

# Подписи к осям
ax.set_xlabel(parameter1)
ax.set_ylabel(parameter2)
ax.set_zlabel('PNL Net')

# Заголовок графика
current_time = datetime.now().strftime("%Y-%m-%d %H:%M") # Генерируем текущее вр
ax.set_title(f"3D Optimization Chart, {current_time}")

# Добавление плоскости, которая будет двигаться вдоль оси Z
# Начальное значение плоскости по оси Z
z_plane = np.mean(z)

# Плоскость - запоминаем ее как отдельный объект
x_plane = np.array([[min(x), max(x)], [min(x), max(x)]])
y_plane = np.array([[min(y), min(y)], [max(y), max(y)]])
z_plane_values = np.array([[z_plane, z_plane], [z_plane, z_plane]])

# Отображение плоскости
plane = ax.plot_surface(x_plane, y_plane, z_plane_values, color='red', alpha=0.5

# Создание слайдера для управления позицией плоскости по оси Z
ax_slider = plt.axes([0.25, 0.02, 0.50, 0.03], facecolor='lightgoldenrodyellow')
z_slider = Slider(ax_slider, 'Z Plane', min(z), max(z), valinit=z_plane)

# Функция обновления положения плоскости при перемещении слайдера
def update(val):
    new_z_plane = z_slider.val
    z_plane_values[:] = new_z_plane  # Обновляем значения Z для плоскости
```

```
# Привязка слайдера к функции обновления
z_slider.on_changed(update)

# Отображение графика
plt.show()</code>
```

Результат оптимизации в виде графика:



Гифка здесь. Сюда не смог вставить.

Выводы из этой оптимизации

Цифры по шкале Z показывают лишь степень убытков в рублях. Они со знаком минус.

Вы можете сами полностью повторить мой опыт потому что код загружен на GitHub:https://github.com/empenoso/SilverFir-TradingBot_backtesting

Тем не менее:

Например, стратегии следования за трендом, как правило, хорошо работают на

избежать. Переобучение прошлыми данными: если стратегия хорошо работает на исторических данных, но плохо на будущих данных в режиме скользящего окна, она может быть слишком адаптирована к историческим моделям, которые не будут повторяться.
3. Транзакционные затраты: хорошо, если тестирование учитывает реалистичное проскальзывание, комиссии и спреды.

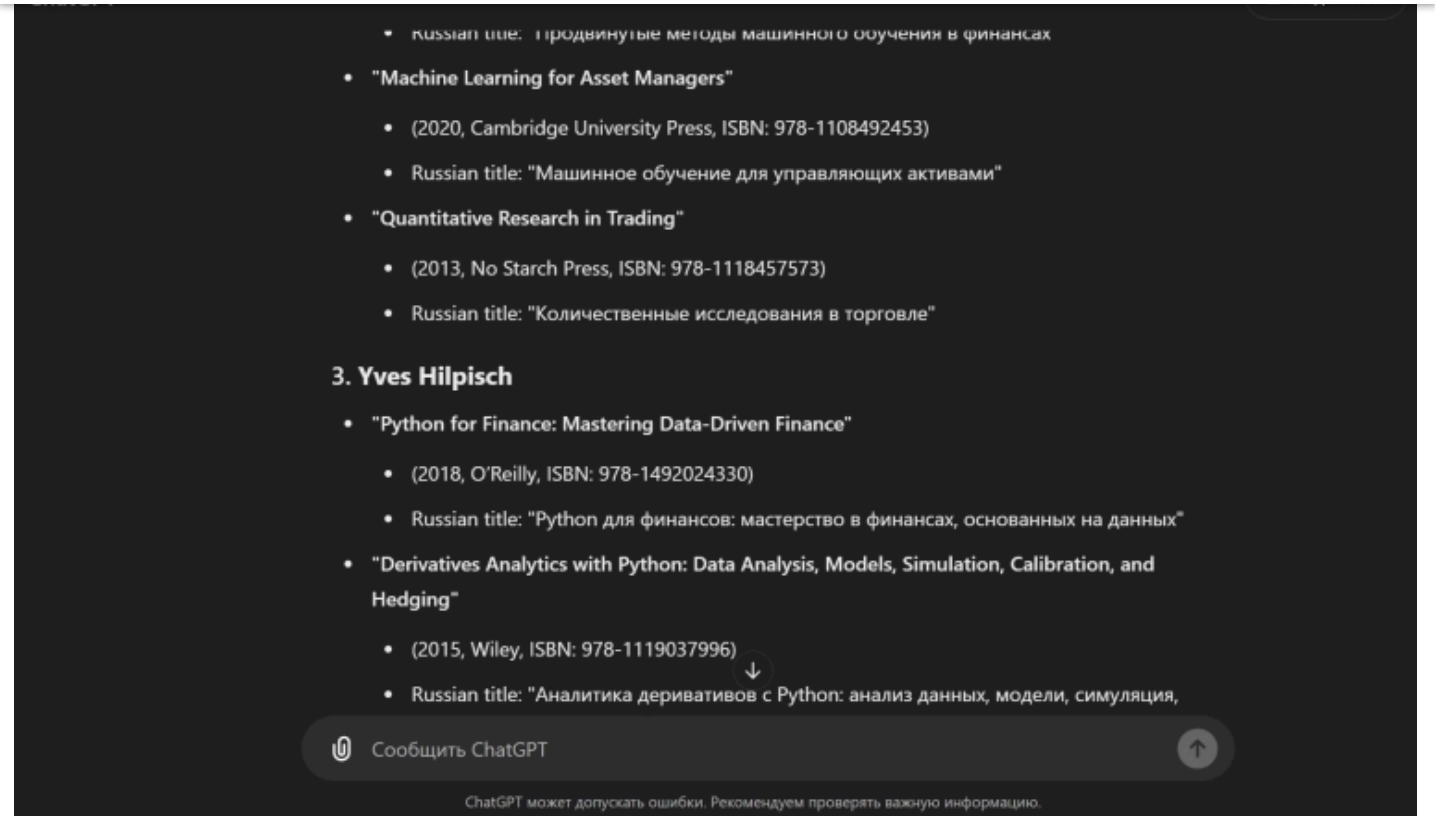
Будущие шаги — где искать прибыльные торговые стратегии 📝

Я хочу использовать подход скользящего окна — когда данные разбиваются на более мелкие последовательные периоды например по месяцам, за которым следует период тестирования вне этой выборки. Например, оптимизация идёт на месячных данных, а тестировать уже на следующем месяце. То есть происходит сдвиг вперед: после каждого периода тестирования окно «скользит» вперед на указанный интервал, и процесс повторяется. Таким образом, каждый сегмент данных используется как для обучения, так и для тестирования с течением времени, но никогда одновременно. Это помогает проверить, что стратегия работает стабильно в меняющихся рыночных условиях.

Также планирую использовать Technical Analysis of STOCKS & COMMODITIES для поиска новых идей. Их советы трейдерам доступны в открытом доступе.

А ещё планирую использовать ChatGPT, отправляя запросы вроде:

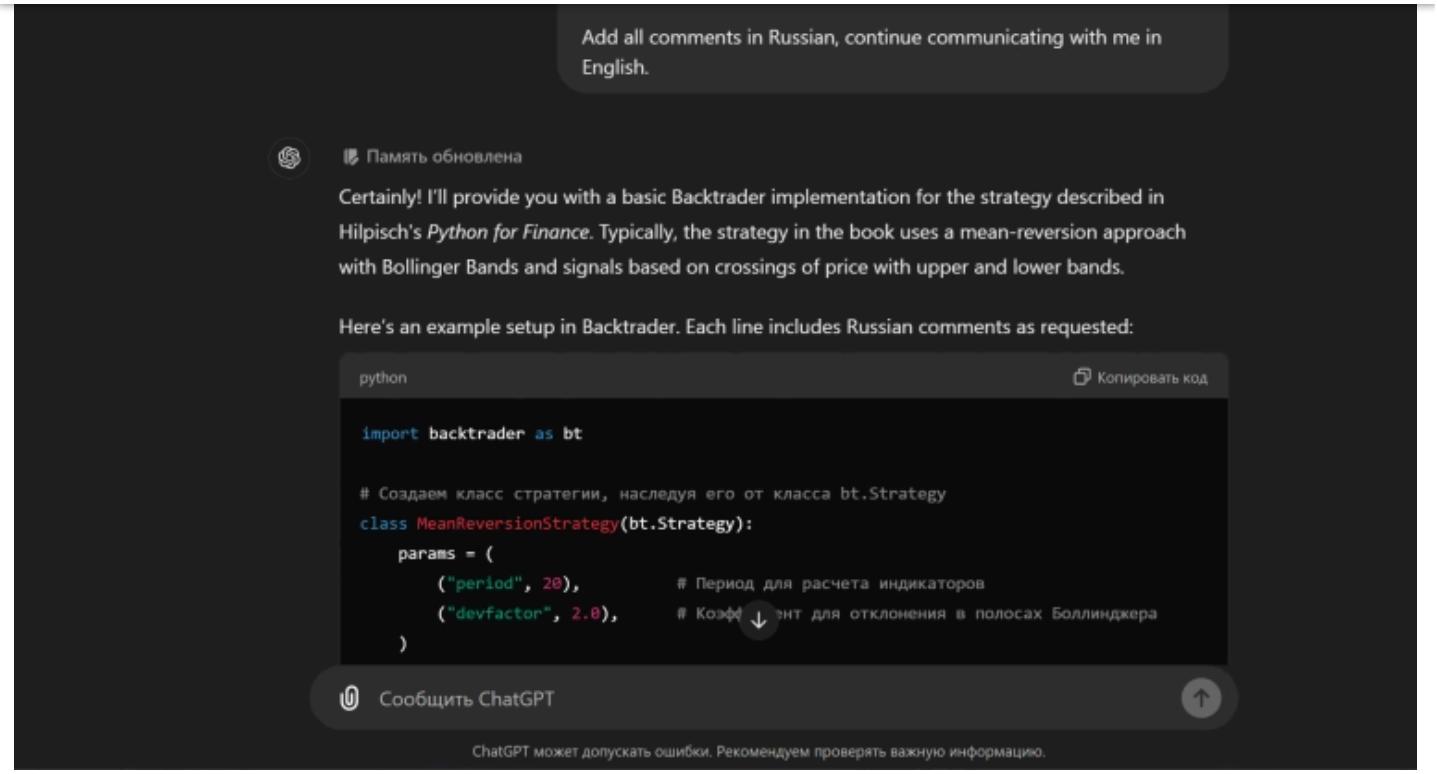
Действуй как опытный издатель. Отобрази 10 ведущих авторов в области алгоритмической торговли на рынке Америки. Для каждого автора перечисли три самые популярные книги, включая сведения о книге (дату публикации, издателя и ISBN), и предоставь русские переводы для каждого названия книги.



и дальше после ответа:

Действуй как опытный пользователь библиотеки backtrader на Python.Хочу использовать торговую стратегию из книги Yves Hilpisch «Python for Finance: Mastering Data-Driven Finance» для тестов.

Добавляй все комментарии на русском языке, продолжай со мной общение на английском.



И дальше подобные промты.

Итоги

Несмотря на то, что первоначальный выбор стратегии на двух разных таймфреймах и сразу для 15 активов был не самый удачный — впереди ещё очень большое поле исследований и тестов.

Автор:Михаил Шардин

18 ноября 2024 г.

трейдинг торговые роботы

806 5 5 6

Михаил Шар...

Пермь

34 402

с 23 января 2019

Ramha
Сегодня в 06:00

сочинять и тестировать модели удобно в Амиброкер www.amibroker.com/

+1

Михаил Шардин
Сегодня в 07:14

> Ramha, раньше я им пользовался, но сейчас я не хотел использовать Windows.

0

SergeyJu
Сегодня в 09:04

Зачем Вами все эти костыли. Вы умеете программировать, ну и пишите все сами на удобном для Вас языке. Питоне, Си, без разницы. Да, и базу данных надо прикрутить желательно тоже стандартную.

+1

Михаил Шардин
Сегодня в 09:17

> SergeyJu, использование готовых библиотек программирования имеет преимущества: экономия времени, качество кода, совместимость и стандартизация, обучение и обмен опытом.
>
> На мой взгляд использование готовых библиотек позволяет сосредоточиться на сути, а не на реализации базовых функций, что делает разработку быстрее, надежнее и эффективнее.

0

SergeyJu
Сегодня в 09:27

Михаил Шардин, то, что должно считаться доли секунды у Вас вылезло в 74 минуты. А вся та маргинальщина, что Вы перечислили, в любой момент сгинет без поддержки. Я бы понял что-то действительно распространенное, хотя бы MT-5.
Я несколько лет был вынужден писать под чужие платформы. 2 разные. Много мучений доставляют скрытые ошибки в чужих прогах, непонятные и неприятные ограничения, медленная работа. При том, что собственно тестирование — процесс с точки зрения программиста очень простой и вполне реализуемый самостоятельно. Ну, кроме, возможно, тяжелых библиотек типа дип леарнинг. Которые есть и стандартно подключаются.

ОТПРАВИТЬ