МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра теоретических основ компьютерной безопасности и криптографии

МАШИННОЕ ОБУЧЕНИЕ И ТРЕЙДИНГ

КУРСОВАЯ РАБОТА

студента 3 курса 331 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Никитина Арсения Владимировича

Научный руководитель
доцент _____ Абросимов М. Б.

Заведующий кафедрой
_____ Абросимов М. Б.

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последнее время анализ курсов акций является немаловажной проблемой. Так различают фундаментальный анализ, основанный на финансовой и производственной деятельности компаний, а также на общей экономической обстановке выбранной сферы. Также определен технический анализ, который основан на прогнозировании изменения курсов в зависимости от закономерностей изменений цен в прошлом в аналогичных обстоятельствах.

Таким образом, в данной работе будет рассмотрена концепция технического анализа, его основные методы, инструменты, а также будет разработан детерминированный алгоритм нахождения технических фигур технического анализа.

Более 150 лет торговые залы Чикагской фондовой биржи были местом проведения торговых сделок покупки и продажи.

Вплоть до 1997 года более 10 тысяч людей торговали в общих залах Чикагских бирж (в так называемых "биржевых ямах"). Позднее, в том же году появилась торговля с помощью компьютеров. Сейчас только около 10% трейдеров продолжают торговать в биржевых залах (остальные 90% торгуют с компьютеров).

Несложно догадаться, что большинство трейдеров, использующих в качестве основного инструмента заработка компьютер, пользуются также различными вспомогательными средствами, обеспечивающие им уверенность в своих действиях.

Итак, одним из средств, помогающих выполнять анализ поведения котировок акций, является технический анализ и, в частности, нахождение и анализ технических фигур.

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Перед анализом теоретической составляющей машинного обучения и технического анализа в трейдинге, стоит ввести ряд терминов и определений, являющихся основой для понимания работы алгоритмов поиска технических фигур, а также для определения основных составляющих трейдинга.

Трейдинг — покупка и продажа акций компаний с целью заработка на ежедневном демпинге цен. Трейдеры внимательно следят за краткосрочными колебаниями цен на эти акции, а затем пытаются купить подешевле и продать подороже. Этот краткосрочный подход отличает биржевых трейдеров от традиционных инвесторов фондового рынка, которые склонны работать в долгосрочной перспективе.

Технический анализ — это торговая дисциплина, используемая для оценки инвестиций и выявления торговых возможностей путем анализа статистических тенденций, полученных в результате торговой деятельности, таких как движение цены и объем. В отличие от фундаментального анализа, который пытается оценить стоимость ценной бумаги на основе результатов бизнеса, таких как продажи и прибыль, технический анализ фокусируется на изучении цены и объема.

Фигура технического анализа — определенная закономерность подряд идущих значений курса акции, которая отображаются на графике и позволяют трейдеру получать определенные сигналы о том, как стоимость актива будет строиться в будущем.

Волатильность — финансовый показатель, отражающий то, как сильно меняется цена на актив или товар за короткий промежуток времени.[1]

Искусственный интеллект (англ. Artificial Intelligence) — технология создания алгоритмов, лежащих в основе проектирования интеллектуальных машин и программ, способных имитировать деятельность человека.

Нейронная сеть (нейросеть) (англ. Neural Network) — математическая модель, чаще всего имеющая программную интерпретацию, сутью которой является реализация деятельности, похожей на деятельность биологических нейронных сетей. Нейронная сеть используется при создании какого-либо из алгоритмов искусственного интеллекта и состоит из совокупности нейронов, соединенных между собой связями.

Признак (англ. Feature) — каждый отдельный элемент информации, вклю-

чаемый в представление о каком-либо анализируемом объекте.

Машинное обучение (англ. Machine Learning) — область науки об искусственном интеллекте, которая изучает способы создания алгоритмов, которые могут обучаться (развиваться).

1 Теоретическая часть

1.1 Временной ряд

Временной ряд — это определенные данные, собранные в разные моменты времени, графиком которого является значение времени по оси абсцисс и значение наблюдаемой величины в текущий момент времени по оси ординат. [1]

Временной ряд может иметь детерминированный компонент в определенный момент времени, тогда говорят, что данные временного ряда имеют временной тренд.

Временные тренды в данных временных рядов также имеют значение для тестирования и моделирования. Надежность модели временных рядов зависит от правильного определения и учета временных тенденций.

Определить, наличие временного тренда на графике можно с помощью проведения возрастающей (убывающей) линии и выявления сосредоточения значений наблюдаемой величины в окрестности этой линии.

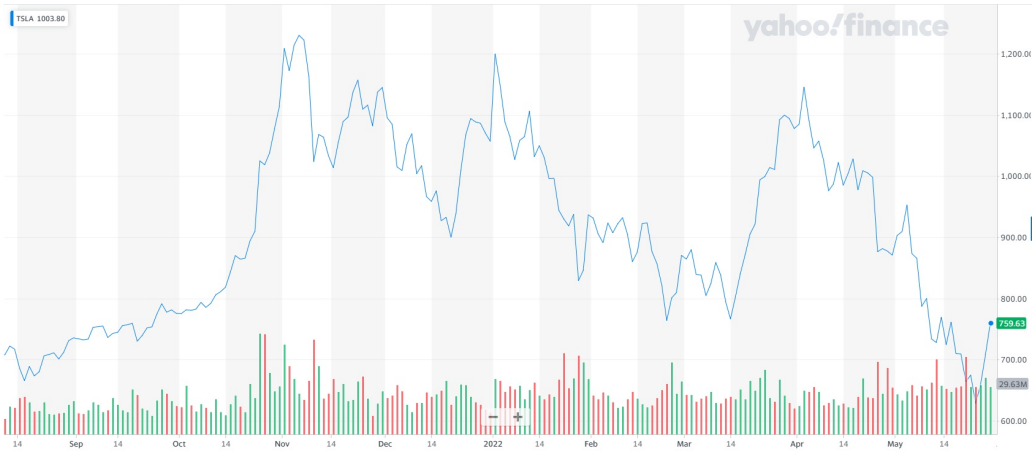

Рисунок 1 – Пример временного ряда

1.2 Основы технического анализа

1.2.1 Японская свеча

Графики свечей — наиболее популярная в Японии и самая древняя форма технического анализа. «Японские свечи» появились раньше, чем штриховые графики и графики «крестики-нолики». Японцы осознали важность технического анализа давным-давно. Они первыми стали торговать фьючерсами. В середине XVII в. они торговали «пустыми» рисовыми контрактами (рисом, которого еше не было, — иначе говоря, рисовыми фьючерсами). [3]

Японская свеча состоит из следующих компонентов:

1. Open (цена открытия) — цена на начало периода.
2. Close (цена закрытия) — цена на конец периода.
3. High () — максимальная цена в течение периода.
4. Close () — минимальная цена в течение периода.

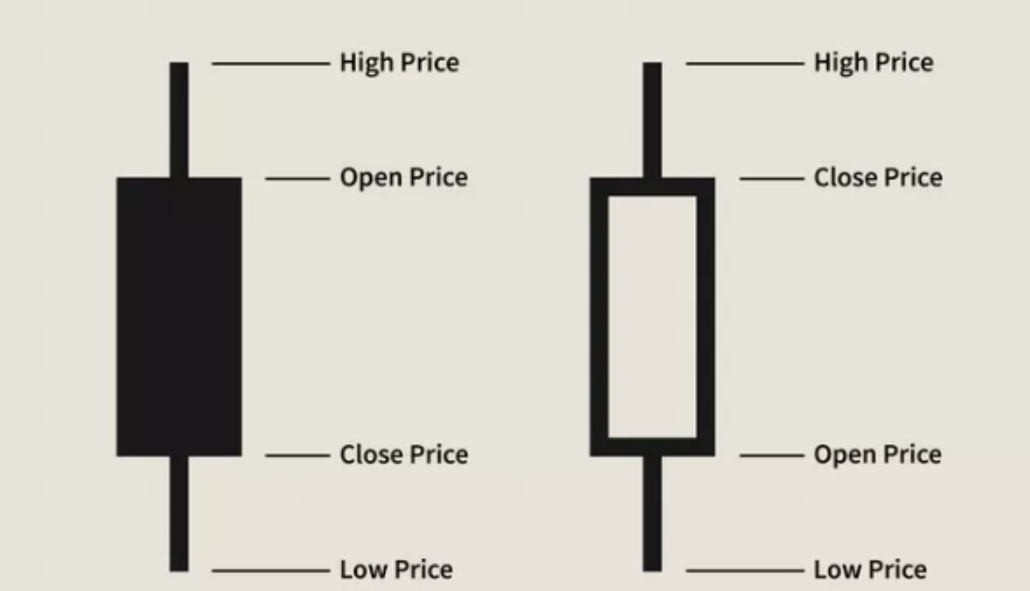

Рисунок 2 – Японская свеча

Для того, чтобы можно было отрисовать график акций, требуется использовать один из показателей японской свечи. В данной работе далее используется показатель $Close$.

1.2.2 Определение тенденции с помощью макисмумов и минимумов

В качестве одного из стандартных методов определения тенденции повышения можно рассмотреть последовательности более высоких максимумов и более низких минимумов.

Итак, тенденция повышения является ненарушенной ровно до тех пор, пока зафиксированный предыдущий относительный минимум не будет пробит, то есть не образуется новый минимум. Если такое происходит, то делается вывод о том, что тенденция закончилась или же в скором времени это произойдет.

Аналогично можно определить и понижающую тенденцию как последовательность более низких минимумов и более низких максимумов. Понижающая тенденция является ненарушенной до тех пор, пока зафиксированный предыдущий максимум не пробит. Если же наблюдается пробитие максимума, то возможно понижающая тенденция подходит к своему завершению.

Рисунок 3 – Пример нисходящего тренда

1.2.3 Линия роста (падения)

Кумулятивная дневная линия роста / падения является одним из наиболее известных индикаторов разброса рынка и часто используется для обнаружения дивергенции (расхождения) с одним из свободных индексов рынка. Как правило, кумулятивная линия роста (падения) вычисляется как скользящая сумма чистого количества растущих акций.

Вычисление значений линии роста (падения) проводится в два этапа:

1. Вычисление разности между количеством растущих и количеством падающих акций с сохранением знака результата. Полученное значение также называется *чистым количеством растущих акций*.
2. Прибавить вычисленную сумму за данный день к значению накопленной суммы ежедневных количеств *растущих акций*.

Также предлагается приступать к вычислению кумулятивной скользящей только после проведения нормализации дневных данных по формуле:

$$N = \frac{A-D}{A+D},$$

где N — отношение чистого количества растущих акций к общему числу акций, показавших изменение цены, A — количество растущих акций, D — количество падающих акций.

1.3 Перцептрон как один из методов реализации нейронных сетей

Типичным примером модели глубокого обучения является глубокая сеть прямого распространения, или многослойный перцептрон (МСП). Многослойный перцептрон — это математическая функция, отображающая множество входных значений на множество выходных. Эта функция является композицией нескольких более простых функций. Каждое применение одной математической функции можно рассматривать как новое представление входных данных.

Глубина позволяет обучать многошаговую компьютерную программу. Каждый слой представления можно мыслить себе как состояние памяти компьютера после параллельного выполнения очередного набора инструкций. Чем больше глубина сети, тем больше инструкций она может выполнить последовательно. Последовательное выполнение инструкций расширяет возможности, поскольку более поздние инструкции могут обращаться к результатам выполнения предыдущих.

Есть два основных способа измерить глубину модели. Первый оценивает архитектуру на основе числа последовательных инструкций, которые необходимо выполнить. Можно считать, что это длина самого длинного пути в графе, описывающем вычисление каждого выхода модели по ее входам. Как у двух эквивалентных компьютерных программ могут быть разные длины пути в зависимости от языка, на котором они написаны, так и одна и та же функциональность может быть изображена графами с разной длиной пути в зависимости от того, какие функции допускаются в качестве шагов.

При другом подходе, используемом в глубоких вероятностных моделях, глубиной модели считается не глубина графа вычислений, а глубина графа, описывающего связи концепций. В этом случае граф вычислений, выполняемых для вычисления представления каждой концепции, может быть гораздо глубже, чем граф самих концепций. Связано это с тем, что понятие системы о простых концепциях можно уточнять, располагая информацией о более сложных. Например, система ИИ, наблюдающая изображение лица, на котором один глаз находится в тени, первоначально может распознать только один глаз. Но, обнаружив присутствие лица, система может заключить, что должен быть и второй глаз. В таком случае граф концепций содержит только два слоя – для глаз и для лиц, тогда как граф вычислений содержит $2n$ слоев, если мы n раз уточняем оценку каждой концепции при известной информации о второй.

Поскольку не всегда ясно, какой из двух подходов – глубина графа вычислений или глубина графа вероятностной модели – более релевантен, и поскольку разные люди по-разному выбирают наборы примитивных элементов, из которых строятся графы, не существует единственно правильного значения глубины архитектуры, как не существует единственно правильной длины компьютерной программы. И нет общего мнения о том, какой должна быть глубина, чтобы модель можно было считать «глубокой». Однако можно все-таки сказать, что глубокое обучение — это наука о моделях, в которых уровень композиции обученных функций или обученных концепций выше, чем в традиционном машинном обучении.

1.4 Использование нейронных сетей для поиска фигур технического анализа

Существует целая теория распознавания изображений образов, являющаяся разделом кибернетики.

Нейронные сети в рассматриваемой задачи могут дать преимущество в связи с тем, что они имеют афинную инвариантность к представляемым данным, в частности, к масштабу и углам смещения фигур технического анализа. Тогда данные отклонения можно считать шумом, с которым нейронная сеть, как правило умеет работать.

Одним из первых вопросов при использовании нейронной сети для нашей задачи является количество входов и подаваемая на них информация. Также важным моментом является выбор типа нейронной сети, ее параметром и методов обучения. Рассматриваемая проблема относится к классу задач классификации, а, значит, необходимо понять, к какой фигуре относится текущее поведение временного ряда.

В качестве входной информации для нейронной сети можно использовать координаты начала отрезка, его окончания и угол наклона прямой.

Итак, для задачи о предсказании поведения текущего курса акций разумно в качестве нейросетевого ядра использовать многослойный перцептрон с обучением по методу обратного распространения ошибки. Данный тип нейронной сети хорошо зарекомендовал себя в задачах прогнозирования.

На входы нейронной сети будут подаваться параметры нескольких подряд идущих отрезков. На выходе будут параметры следующего отрезка.

Нейронные сети, безусловно, являются одним из методов поиска фигур

технического анализа и предсказания поведения акций в зависимости от полученных результатов работы. В данной работе же рассматриваются детерминированные подходы которые пытаются выделить технические фигуры на заданном временном ряде и затем провести прогнозирование поведения ряда в зависимости от найденных фигур.

2 Практическая часть

2.1 Описание инструментов и библиотек программной реализации

В виду своей удобности использования, простоты написания кода, а также различных дополнительных библиотек для анализа акций, в данной работе используется язык программирования Python. Простота применения Python для написания кода и возможность его отладки через такие пакетные менеджеры, как conda, упрощают обмен библиотеками и результатами исследований. Его экосистема библиотек для работы в сфере трейдинга, таких как NumPy, yfinance и Matplotlib, упрощают анализ результатов работы алгоритма, получение котировок акций с различной временной градацией.

Yfinance — популярная библиотека с открытым исходным кодом, разработанная Раном Арусси как средство доступа к финансовым данным, расположенным на ресурсе Yahoo Finance.

Yahoo Finance предлагает обширный набор рыночных данных по акциям, облигациям, валютам и криптовалютам. Для удобства использования в данной библиотеке присутствует возможность получения отчетов и анализов данных, а также дополнительные параметры получаемых данных, что отличает его от ряда конкурентов.

Также стоит отметить, что отличительной особенностью yfinance является то, что библиотека позволяет получить градуирование данных вплоть одной минуты. Однако, что данные за текущий день получить нельзя. Если требуется получить данные за день (то есть градуирование составляет по часам и меньше), то минимально возможная разница в днях с текущей датой составляет 60 дней.

Получаемые данные представляют из себя csv-файл, в котором присутствуют вышеописанные элементы свечей: максимальная (минимальная) цена, открытие (закрытие) акции.

Matplotlib — это кроссплатформенная библиотека для визуализации данных и графического построения графиков для Python и NumPy. В данной работе использование библиотеки является необходимым для отображения котировок акций и найденных фигур технического анализа с предполагаемым ростом (падением) курса. По оси абсцисс будем брать градуирование даты, а по оси ординат будем использовать полученные с помощью yfinance котировки, используя поле $close$.

ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены детерминированные методы поиска технических фигур на курсах акций. Проведен анализ работы алгоритма на акциях с высокой волатильностью и низкой, а также на акциях криптовалют. Алгоритм в некоторых случаях может работать некорректно в следствие неправильной подобранных значений допуска eps в фигурах, где это значение используется. Также были рассмотрены теоретические основы нейросетей, описаны основные принципы работы и их роль в решении, основы технического анализа и основные определения трейдинга.

В качестве практической части работы была описана программная реализация алгоритма поиска технических фигур на акциях, получение статистики по фигурам, а также отрисовка данных фигур на графике акции с предполагаемым ростом или падением курса.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Zelun Luo, Boya Peng и т.д., ”Show, Discriminate, and Tell: A Discriminatory Image Captioning Model with Deep Neural Networks”, [Электронный ресурс] : [статья] / URL https://web.stanford.edu/class/cs231a/prev_projects_-2016/cs231a.pdf (дата обращения 8.05.2022) Загл. с экрана. Яз. англ.

2 Ritter F. E., Schooler L. J., ”The learning curve”, [Электронный ресурс] : [статья] / URL http://ritter.ist.psu.edu/papers/ritterS01.pdf (дата обращения 12.05.2022) Загл. с экрана. Яз. англ.

ПРИЛОЖЕНИЕ А

Код drawing.py

```
def draw_pennants_rising_flags(plt, indexes, prices, date, delta):
    for index in indexes:
        x_upper_line = [date[index], date[index + 4]]
        y_upper_line = [prices[index], prices[index + 4]]
        plt.plot(x_upper_line, y_upper_line)
        x_lower_line = [date[index + 1], date[index + 5]]
        y_lower_line = [prices[index + 1], prices[index + 5]]
        plt.plot(x_lower_line, y_lower_line, color='red')
        if prices[index - 1] < prices[index]:
            x_rising_line = [date[index + 5], date[index + 8]]
            y_rising_line = [prices[index + 5], prices[index + 5] + delta]
            plt.plot(x_rising_line, y_rising_line, color='red')


def draw_descending_flags(plt, indexes, prices, date, delta):
    for index in indexes:
        x_upper_line = [date[index], date[index + 4]]
        y_upper_line = [prices[index], prices[index + 4]]
        plt.plot(x_upper_line, y_upper_line)
        x_lower_line = [date[index + 1], date[index + 5]]
        y_lower_line = [prices[index + 1], prices[index + 5]]
        plt.plot(x_lower_line, y_lower_line, color='red')
        if prices[index - 1] > prices[index]:
            x_rising_line = [date[index + 5], date[index + 8]]
            y_rising_line = [prices[index + 5], prices[index + 5] + delta]
            plt.plot(x_rising_line, y_rising_line, color='red')


def draw_rectangles(plt, indexes, prices, date, delta):
    for index in indexes:
        x_upper_line = [date[index + 1], date[index + 7]]
        y_upper_line = [prices[index + 1], prices[index + 7]]
        plt.plot(x_upper_line, y_upper_line, color='red')
        x_lower_line = [date[index], date[index + 6]]
        y_lower_line = [prices[index], prices[index + 6]]
        plt.plot(x_lower_line, y_lower_line, color='red')
        x_line = [date[index + 7], date[index + 10]]
        y_line_2nd_arg = prices[index + 7] - delta if \
             prices[index] < prices[index - 1] else \
                  prices[index + 7] + delta
        y_line = [prices[index + 7], y_line_2nd_arg]
        plt.plot(x_line, y_line, color='red')


def draw_rhombuses(plt, indexes, prices, date, delta):
    for index in indexes:
```

```
        y_middle_value = (prices[index + 4] + prices[index + 3]) / 2

        x_1st_line = [date[index - 1], date[index + 4]]
        y_1st_line = [y_middle_value, prices[index + 4]]
        plt.plot(x_1st_line, y_1st_line, color='red')

        x_2nd_line = [date[index + 4], date[index + 9]]
        y_2nd_line = [prices[index + 4], y_middle_value]
        plt.plot(x_2nd_line, y_2nd_line, color='red')

        x_3rd_line = [date[index + 9], date[index + 4]]
        y_3rd_line = [y_middle_value, prices[index + 4]]
        plt.plot(x_3rd_line, y_3rd_line, color='red')

        x_4th_line = [date[index + 4], date[index - 1]]
        y_4th_line = [prices[index + 4], y_middle_value]
        plt.plot(x_4th_line, y_4th_line, color='red')

        y_value_line = prices[index + 8] + delta if \
            prices[index] < prices[index - 1] else \
                prices[index + 8] - delta
        y_values_line = [prices[index + 8], y_value_line]
        x_values_line = [date[index + 9], date[index + 13]]
        plt.plot(x_values_line, y_values_line, color='red')


def draw_double_tops(plt, indexes, prices, date, delta):
    for index in indexes:
        x_upper_line = [date[index - 2], date[index + 4]]
        y_upper_line = [prices[index], prices[index]]
        plt.plot(x_upper_line, y_upper_line)
        x_lower_line = [date[index - 2], date[index + 4]]
        y_lower_line = [prices[index + 1], prices[index + 1]]
        plt.plot(x_lower_line, y_lower_line, color='red')
        if prices[index] > prices[index - 1]:
            y_line = [prices[index + 2], prices[index + 2] - delta]
            x_line = [date[index + 2], date[index + 5]]
            plt.plot(x_line, y_line)


def draw_triple_bottoms(plt, indexes, prices, date, delta):
    for index in indexes:
        x_upper_line = [date[index + 1], date[index + 3]]
        y_upper_line = [prices[index + 1], prices[index + 1]]
        plt.plot(x_upper_line, y_upper_line)
        x_lower_line = [date[index], date[index + 4]]
        y_lower_line = [prices[index], prices[index + 4]]
        plt.plot(x_lower_line, y_lower_line, color='red')
```

```
        if prices[index] < prices[index - 1]:
            x_line = [date[index + 4], date[index + 7]]
            y_line = [prices[index + 4], prices[index + 4] + delta]
            plt.plot(x_line, y_line)


def draw_head_and_shoulders(plt, indexes, prices, date, delta):
    for index in indexes:
        x_line = [date[index - 1], date[index + 5]]
        y_line = [prices[index + 1], prices[index + 1]]
        plt.plot(x_line, y_line, color='red')
        x_line = [date[index + 4], date[index + 7]]
        y_value = prices[index + 4] - delta if \
            prices[index] > prices[index - 1] else prices[index + 4] + delta
        y_line = [prices[index + 7], y_value]
        plt.plot(x_line, y_line)


def draw_wedges(plt, indexes, prices, date, flag, delta_y):
    # Флаг нужен для отрисовки нисходящих / восходящих
    delta = 4 if flag else 6
    for index in indexes:
        x_upper_line = [date[index], date[index + delta]]
        y_upper_line = [prices[index + 1], prices[index + delta]]
        plt.plot(x_upper_line, y_upper_line)
        x_lower_line = [date[index + 1], date[index + 5]]
        y_lower_line = [prices[index + 1], prices[index + 5]]
        plt.plot(x_lower_line, y_lower_line, color='red')
        x_line = [date[index + delta], date[index + delta + 3]]
        y_line_1st = prices[index + delta] + delta_y if delta == 5 else \
             prices[index + delta] - delta_y
        y_line_2nd = prices[index + delta]
        y_line = [y_line_2nd, y_line_1st]
        plt.plot(x_line, y_line)
```

ПРИЛОЖЕНИЕ Б

Код figures.py

```
def make_bolting(indexes, delta):
    res = []
    if indexes:
        res.append(indexes[0])
    if len(indexes) > 1:
        for i in range(1, len(indexes) - 1):
            if indexes[i] - indexes[i + 1] <= delta and \
                indexes[i] - res[-1] <= delta:
                continue
            else:
                res.append(indexes[i])
    return res


def get_pennants(prices):
    res = []
    col = 0
    for i in range(0, len(prices) - 6):
        if prices[i] > prices[i + 2] > prices[i + 4] > prices[i + 5] > \
            prices[i + 3] > prices[i + 1]:
            res.append(i)
            col += 1
    return col, make_bolting(res, 6)


def get_descending_flags(prices):
    res = []
    col = 0
    eps = 1.25
    for i in range(0, len(prices) - 6):
        if prices[i] > prices[i + 2] > prices[i + 4] > prices[i + 5] > \
            prices[i + 3] > prices[i + 1] and \
                abs(prices[i] - prices[i + 4] - prices[i + 1] \
                    + prices[i + 5]) < eps:
            res.append(i)
            col += 1
    return col, make_bolting(res, 6)


def get_rising_flags(prices):
    res = []
    col = 0
    eps = 0.5
    for i in range(0, len(prices) - 6):
        if prices[i] < prices[i + 2] < prices[i + 4] < prices[i + 5] and \
            prices[i + 1] < prices[i + 3] < prices[i + 5] \
```

```
                and abs(prices[i] - prices[i + 4] - prices[i + 1] + \
                    prices[i + 5]) < eps:
            res.append(i)
            col += 1
    return col, make_bolting(res, 6)


def get_rectangles(prices):
    res = []
    col = 0
    eps = 0.15
    for i in range(len(prices) - 8):
        if abs(prices[i] - prices[i + 2]) < eps and \
            abs(prices[i + 2] - prices[i + 4]) < eps and \
                abs(prices[i + 4] - prices[i + 6]) < eps and \
                    abs(prices[i + 1] - prices[i + 3]) < eps and \
                abs(prices[i + 3] - prices[i + 5]) < eps and \
                    abs(prices[i + 5] - prices[i + 7]) < eps and \
                prices[i + 7] > prices[i + 6]:
            res.append(i)
            col += 1
    return col, make_bolting(res, 8)


def get_descending_rhombuses(prices):
    res = []
    col = 0
    eps = 0.40
    for i in range(len(prices) - 9):
        if prices[i + 2] < prices[i] < prices[i + 1] and \
            prices[i + 6] < prices[i + 8] < prices[i + 7] and \
                abs(prices[i + 3] - prices[i + 5]) < eps and \
                    abs(prices[i + 2] - prices[i + 6]) < eps and \
                prices[i + 4] < prices[i + 2] and \
                    abs(prices[i + 1] - prices[i + 7]) < eps and \
                abs(prices[i] - prices[i + 8]) < eps and \
                    prices[i + 3] > prices[i + 8]:
            res.append(i)
            col += 1
    return col, make_bolting(res, 9)


def get_rising_rhombuses(prices):
    res = []
    col = 0
    eps = 0.15
    for i in range(len(prices) - 9):
        if prices[i + 2] > prices[i] > prices[i + 1] and \
```

```
            prices[i + 6] > prices[i + 8] > prices[i + 7] and \
                abs(prices[i + 3] - prices[i + 5]) < eps and \
                    abs(prices[i + 2] - prices[i + 6]) < eps and \
                prices[i + 4] > prices[i + 2] and \
                    abs(prices[i + 1] - prices[i + 7]) < eps and \
                abs(prices[i] - prices[i + 8]) < eps and \
                    prices[i + 3] < prices[i + 8]:
            res.append(i)
            col += 1
    return col, make_bolting(res, 9)


def get_double_tops(prices):
    res = []
    col = 0
    eps = 0.1
    for i in range(1, len(prices) - 3):
        if prices[i - 1] < prices[i + 1] < prices[i] and \
            abs(prices[i] - prices[i + 2]) < eps:
            res.append(i)
            col += 1
    return col, make_bolting(res, 3)


def get_triple_bottoms(prices):
    res = []
    col = 0
    eps = 0.1
    for i in range(1, len(prices) - 5):
        if abs(prices[i] - prices[i + 2]) < eps and \
            abs(prices[i + 2] - prices[i + 4]) < eps and \
                abs(prices[i + 1] - prices[i + 3]) < eps and \
                    prices[i] < prices[i + 1] and prices[i] < prices[i - 1]:
            res.append(i)
            col += 1
    return col, make_bolting(res, 5)


def get_reversed_head_and_shoulders(prices):
    res = []
    col = 0
    eps = 0.1
    for i in range(1, len(prices) - 5):
        if prices[i] < prices[i - 1] and \
            abs(prices[i + 1] - prices[i + 3]) < eps and \
                abs(prices[i] - prices[i + 4]) < eps and \
                    prices[i + 1] > prices[i] > prices[i + 2]:
            res.append(i)
```

```
            col += 1
    return col, make_bolting(res, 5)


def get_head_and_shoulders(prices):
    res = []
    col = 0
    eps = 0.1
    for i in range(1, len(prices) - 5):
        if prices[i] > prices[i - 1] and \
            abs(prices[i + 1] - prices[i + 3]) < eps and \
                abs(prices[i] - prices[i + 4]) < eps and \
                    prices[i + 1] < prices[i] < prices[i + 2]:
            res.append(i)
            col += 1
    return col, make_bolting(res, 5)


def get_falling_wedges(prices):
    res = []
    col = 0
    for i in range(len(prices) - 6):
        if prices[i] > prices[i + 2] > prices[i + 4] > prices[i + 5] and \
            prices[i + 1] > prices[i + 3] > prices[i + 5]:
            res.append(i)
            col += 1
    return col, make_bolting(res, 6)


def get_rising_wedges(prices):
    res = []
    col = 0
    for i in range(len(prices) - 7):
        if prices[i] < prices[i + 2] < prices[i + 4] < prices[i + 6] and \
                prices[i + 1] < prices[i + 3] < prices[i + 5] < prices[i + 6]:
            res.append(i)
            col += 1
    return col, make_bolting(res, 7)


def get_rising_triangles(prices):
    res = []
    col = 0
    eps = 0.25
    for i in range(len(prices) - 7):
        if abs(prices[i + 1] - prices[i + 3]) < eps and \
            abs(prices[i + 3] - prices[i + 5]) < eps and \
                prices[i] < prices[i + 2] < prices[i + 4] < \
```

```
                    prices[i + 6] < prices[i + 5]:
            res.append(i)
            col += 1
        return col, make_bolting(res, 7)


def get_falling_triangles(prices):
    res = []
    col = 0
    eps = 0.25
    for i in range(len(prices) - 8):
        if abs(prices[i] - prices[i + 2]) < eps and \
            abs(prices[i + 2] - prices[i + 4]) < eps and \
                abs(prices[i + 4] - prices[i + 6]) < eps and \
                    prices[i + 6] < prices[i + 7] < prices[i + 5] < \
                prices[i + 3] < prices[i + 1]:
            res.append(i)
            col += 1
        return col, make_bolting(res, 8)
```

ПРИЛОЖЕНИЕ В

Код main.py

```
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt
import figures
import drawing as d
plt.figure(figsize=(30, 20))


def print_statistics(figure_name, indexes, col, prices, flag, c):
    print(f'Стастистика по фигуре {figure_name}:')
    print(f'Всего найдено: {col}, что составляет {round(col / c, 3)} \
        от общего числа фигур')
    res = []
    # pennants
    if flag == 1:
        for index in indexes:
            res.append([1 if prices[index] > prices[index - 1] and \
                prices[index + value] > prices[index + 5] else 0
                        for value in range(6, 11)])
    #flags
    if flag == 2:
        for index in indexes:
            res.append([1 if prices[index] < prices[index - 1] and \
                prices[index + value] > prices[index + 5] or
                            prices[index] > prices[index - 1] and \
                                prices[index + 6] < prices[index + value] else 0
                        for value in range(6, 11)])
    #rectangles
    if flag == 3:
        for index in indexes:
            res.append([1 if prices[index] > prices[index - 1] and \
                prices[index + 7] < prices[index + value] or
                            prices[index] < prices[index - 1] and \
                                prices[index + 7] > prices[index + value] else 0
                        for value in range(8, 13)])
    #rhombuses
    if flag == 4:
        for index in indexes:
            res.append([1 if prices[index] > prices[index - 1] and \
                prices[index + 8] > prices[index + value] or
                            prices[index] < prices[index - 1] and \
                                prices[index + 8] < prices[index + value] else 0
                        for value in range(9, 14)])
    #head_and_shoulders
    else:
        for index in indexes:
```

```
            res.append([1 if prices[index] > prices[index - 1] and \
                 prices[index + 4] < prices[index + value] or
                             prices[index] < prices[index - 1] and \
                                 prices[index + 4] > prices[index + value] \
                                      else 0
                         for value in range(5, 10)])

    for subres in res:
        print(subres)


def get_extrema(date, prices):
    indexes = []
    for i in range(1, len(prices) - 1):
        if prices[i] > prices[i - 1] and prices[i] > prices[i + 1] or \
                prices[i] < prices[i - 1] and prices[i] < prices[i + 1]:
           indexes.append(i)

    return [date[index] for index in indexes], \
        [prices[index] for index in indexes]


data = yf.Ticker("AAPL").history(period='max')
prices = np.array(data['High'])
data.reset_index(inplace=True)
date = np.array(data['Date'], dtype='datetime64[D]')

delta_y = (max(prices) - min(prices)) / 10
date_extrema, prices_extrema = get_extrema(date, prices)
# plt.plot(date_extrema, prices_extrema, color='red')
plt.plot(date, prices)
count_pennants, res_pennants = figures.get_pennants(prices_extrema)
d.draw_pennants_rising_flags(plt, res_pennants, prices_extrema, \
    date_extrema, delta_y)

count_descending_flags, res_descending_flags = figures.get_descending_flags(\
    prices_extrema)
d.draw_descending_flags(plt, res_descending_flags, prices_extrema, \
    date_extrema, delta_y)

count_rising_flags, res_rising_flags = figures.get_rising_flags(prices_extrema)
d.draw_pennants_rising_flags(plt, res_rising_flags, prices_extrema, \
    date_extrema, delta_y)

count_rectangles, res_rectangles = figures.get_rectangles(prices_extrema)
d.draw_rectangles(plt, res_rectangles, prices_extrema, date_extrema, delta_y)

count_descending_rhombuses, res_descending_rhombuses = figures.get_descending_rhombuses(\
```

```
    prices_extrema)
d.draw_rhombuses(plt, res_descending_rhombuses, prices_extrema, \
    date_extrema, delta_y)

count_rising_rhombuses, res_rising_rhombuses = figures.get_rising_rhombuses(\
    prices_extrema)
d.draw_rhombuses(plt, res_rising_rhombuses, prices_extrema, \
    date_extrema, delta_y)

count_double_tops, res_double_tops = figures.get_double_tops(prices_extrema)
d.draw_double_tops(plt, res_double_tops, prices_extrema, date_extrema, delta_y)

count_triple_bottoms, res_triple_bottoms = figures.get_triple_bottoms(prices_extrema)
d.draw_triple_bottoms(plt, res_triple_bottoms, prices_extrema, \
    date_extrema, delta_y)

count_head_and_shoulders, res_head_and_shoulders = figures.get_head_and_shoulders(\
    prices_extrema)
d.draw_head_and_shoulders(plt, res_head_and_shoulders, prices_extrema, \
    date_extrema, delta_y)

count_reversed_head_and_shoulders, res_reversed_head_and_shoulders = \
    figures.get_reversed_head_and_shoulders(prices_extrema)
d.draw_head_and_shoulders(plt, res_reversed_head_and_shoulders, \
    prices_extrema, date_extrema, delta_y)

count_falling_wedges, res_falling_wedges = figures.get_falling_wedges(prices_extrema)
d.draw_wedges(plt, res_falling_wedges, prices_extrema, \
    date_extrema, True, delta_y)

count_rising_wedges, res_rising_wedges = figures.get_rising_wedges(prices_extrema)
d.draw_wedges(plt, res_rising_wedges, prices_extrema, \
    date_extrema, False, delta_y)

plt.show()
c = count_pennants + count_descending_flags + count_rectangles + \
    count_descending_rhombuses + count_rising_rhombuses \
      + count_double_tops + count_head_and_shoulders + \
          count_reversed_head_and_shoulders + count_falling_wedges \
      + count_rising_wedges

print(f'Общее количество найденных фигур: {c}')

# print_statistics('вымпел', res_pennants, count_pennants, prices_extrema, 1, c)
# pennants, \
# descending_flags, \
# rising_flags, \
# rectangles, \
```

```
# descending_rhombuses, \
# rising_rhombuses, \
# head_and_shoulders, \
# reversed_head_and_shoulders, \


# double_tops, \
# triple_bottoms, \
# falling_wedges, \
# rising_wedges, \

# rising_triangles, \
# falling_triangles
```