Geekbrains

**ДИПЛОМНАЯ РАБОТА**

**Прогнозирование котировок ценных бумаг с Московской биржи на основе исторических данных и методов машинного обучения.**

Программа: Разработчик-Аналитик

Специализация: Data Science

Сарапаев Роман Павлович

г. Чебоксары

2025

Содержание

[Введение 4](#_Toc198392482)

[Глава 1. Теоретическая часть. 6](#_Toc198392483)

[1.1. Задача и проблемы Data Science. 6](#_Toc198392484)

[Введение в Data Science 6](#_Toc198392485)

[Основные задачи Data Science 6](#_Toc198392486)

[Основные проблемы и вызовы в Data Science 7](#_Toc198392487)

[Data Science в финансовой сфере и прогнозировании цен акций 8](#_Toc198392488)

[1.2. Сбор данных. 9](#_Toc198392489)

[Источники данных 9](#_Toc198392490)

[Методы получения данных 9](#_Toc198392491)

[Проблемы и вызовы при сборе данных 10](#_Toc198392492)

[1.3. Обработка и очистка данных 12](#_Toc198392493)

[Цель обработки данных 12](#_Toc198392494)

[Основные задачи обработки данных 12](#_Toc198392495)

[Трудности и риски 14](#_Toc198392496)

[1.4. Исследовательский анализ данных (Exploratory Data Analysis, EDA) 15](#_Toc198392497)

[Задачи EDA 15](#_Toc198392498)

[Основные методы и инструменты EDA 15](#_Toc198392499)

[Важность EDA 17](#_Toc198392500)

[1.5. Построение и обучение моделей 18](#_Toc198392501)

[Построение и обучение моделей 18](#_Toc198392502)

[Типы задач, решаемые моделями 18](#_Toc198392503)

[Подготовка признаков и выбор переменных 18](#_Toc198392504)

[Выбор модели 19](#_Toc198392505)

[Процесс обучения модели 20](#_Toc198392506)

[Настройка гиперпараметров 21](#_Toc198392507)

[1.6. Оценка и валидация моделей 22](#_Toc198392508)

[Цели оценки моделей 22](#_Toc198392509)

[Разделение данных на выборки 22](#_Toc198392510)

[Метрики оценки качества моделей 23](#_Toc198392511)

[Кросс-валидация (Cross-validation) 24](#_Toc198392512)

[Предотвращение переобучения 24](#_Toc198392513)

[Сравнение моделей 24](#_Toc198392514)

[1.7 Интерпретация и объяснение результатов 26](#_Toc198392515)

[Зачем нужна интерпретация 26](#_Toc198392516)

[Интерпретируемые и не интерпретируемые модели 26](#_Toc198392517)

[Методы интерпретации моделей 27](#_Toc198392518)

[Интерпретация в задачах классификации и регрессии 27](#_Toc198392519)

[Интерпретация ошибок модели 28](#_Toc198392520)

[Практическая значимость интерпретации 28](#_Toc198392521)

[1.8 Внедрение и мониторинг 29](#_Toc198392522)

[Внедрение модели (Model Deployment) 29](#_Toc198392523)

[Мониторинг модели 29](#_Toc198392524)

[Инструменты для внедрения и мониторинга 30](#_Toc198392525)

[Обновление и переобучение модели 31](#_Toc198392526)

[Проблемы на этапе внедрения 31](#_Toc198392527)

[Роль MLOps 31](#_Toc198392528)

[Глава 2. Практическая часть 32](#_Toc198392529)

[2.1. Общая постановка задачи 32](#_Toc198392530)

[2.2. Ресурсы и инструменты 32](#_Toc198392531)

[Программные инструменты и библиотеки: 32](#_Toc198392532)

[Сторонние сервисы и API: 33](#_Toc198392533)

[Среда разработки: 33](#_Toc198392534)

[Машинное обучение: 33](#_Toc198392535)

[2.3. Структура проекта и назначение файлов 33](#_Toc198392536)

[Общий алгоритм работы бота 35](#_Toc198392537)

[2.4. Инициализация и запуск Telegram-бота 35](#_Toc198392538)

[2.5. Обработка пользовательских запросов 36](#_Toc198392539)

[Генерация кнопок с тикерами акций (модуль keyboards.py) 37](#_Toc198392540)

[Обработка выбора акции и полный цикл анализа 37](#_Toc198392541)

[2.6. Загрузка и подготовка данных 40](#_Toc198392542)

[Загрузка исторических данных (stock\_utils.py) 40](#_Toc198392543)

[Преобразование и очистка данных (elt.py) 41](#_Toc198392544)

[Добавление технических индикаторов (расширенное пояснение) 42](#_Toc198392545)

[2.7. Прогнозирование с помощью модели XGBoost 45](#_Toc198392546)

[1. Эффективность на табличных данных 45](#_Toc198392547)

[2. Поддержка регуляризации 45](#_Toc198392548)

[3. Работа с пропущенными значениями и шумом 46](#_Toc198392549)

[4. Гибкость в настройке и высокая точность 46](#_Toc198392550)

[5. Быстрота обучения 46](#_Toc198392551)

[6. Поддержка временных зависимостей 46](#_Toc198392552)

[7. Активное использование в индустрии 46](#_Toc198392553)

[Обучение модели с кросс-валидацией 47](#_Toc198392554)

[Прогнозирование цены 48](#_Toc198392555)

[2.8. Визуализация результатов 49](#_Toc198392556)

[Заключение 51](#_Toc198392557)

[Список использованной литературы: 52](#_Toc198392558)

[*Приложение 1* 53](#_Toc198392559)

[*Приложение 2* 53](#_Toc198392560)

[*Приложение 3* 54](#_Toc198392561)

[*Приложение 4* 54](#_Toc198392562)

[*Приложение 5* 55](#_Toc198392563)

# Введение

**Data Science** (наука о данных) — это междисциплинарная область, которая занимается извлечением знаний и полезной информации из данных с помощью научных методов, алгоритмов, процессов и систем, т.е. это искусство и наука работать с данными, чтобы понимать, **что происходит, почему это происходит, и что, вероятно, произойдёт дальше.**

**Data Scientist** — это специалист, который извлекает ценную информацию и знания из данных с помощью анализа, статистики и алгоритмов машинного обучения. Его задача **— найти закономерности в данных и помочь бизнесу принимать решения на их основе**.

Основные этапы **Data Scientista**

|  |  |
| --- | --- |
| * **Сбор данных** | Получает данные из баз данных, API, логов и т.д. |

|  |  |
| --- | --- |
| * **Очистка данных** | Удаляет ошибки, пропуски, аномалии |

|  |  |
| --- | --- |
| * **Анализ (EDA)** | Ищет закономерности, строит графики |

|  |  |
| --- | --- |
| * **Моделирование** | Применяет ML-модели: регрессия, классификация и т.п. |

|  |  |
| --- | --- |
| * **Оценка моделей** | Смотрит, насколько хорошо работает модель |

|  |  |
| --- | --- |
| * **Интерпретация** | Объясняет бизнесу, что значат результаты |

|  |  |
| --- | --- |
| * **Внедрение** | Помогает интегрировать модель в продукт или бизнес-процесс |

**Тема проект:** Прогнозирование котировок ценных бумаг с Московской биржи на основе исторических данных и методов машинного обучения.

В условиях стремительно развивающегося фондового рынка и роста интереса со стороны частных инвесторов, всё большую актуальность приобретают инструменты, позволяющие проводить технический анализ и прогнозировать поведение финансовых активов. Прогнозирование цен на акции представляет собой сложную задачу, зависящую от множества факторов, включая рыночные тенденции, экономические новости, поведение участников торгов, сезонность и другие. Однако одним из надёжных подходов к анализу временных рядов остаётся технический анализ, основанный на обработке исторических данных о ценах и объёмах торгов. Современные технологии машинного обучения позволяют значительно повысить точность прогноза на основе этих данных.

Целью настоящего дипломного проекта является разработка автоматизированной системы прогнозирования цен акций, реализованной в виде Telegram-бота. Бот предоставляет пользователю удобный интерфейс для получения прогноза цен на основе анализа исторических данных и расчёта технических индикаторов. Telegram был выбран в качестве платформы благодаря своей популярности, простоте интеграции с внешними системами и возможностям интерактивного взаимодействия с пользователями.

Основные функции реализованного Telegram-бота включают:

* предоставление пользователю кнопочного меню для выбора акции из списка (например, SBER, GAZP и др.);
* автоматическую загрузку исторических данных по выбранной акции с Московской биржи (MOEX) через библиотеку moexalgo;
* расчёт технических индикаторов (например, скользящие средние, RSI, MACD и другие) с использованием библиотеки pandas\_ta;
* обучение модели машинного обучения (XGBoost) на основе обработанных данных;
* прогнозирование цены на следующий торговый день;
* визуализация результатов в виде графика с нанесённым прогнозом и отправка пользователю.

В качестве алгоритма прогнозирования была выбрана модель XGBoost — один из наиболее мощных и широко применяемых градиентных бустинговых алгоритмов, показавший высокую эффективность в задачах регрессии и классификации. Обучение модели происходит с использованием кросс-валидации и подбора гиперпараметров, что позволяет минимизировать переобучение и повысить обобщающую способность модели.

Практическая ценность проекта заключается в создании доступного и понятного инструмента, который может быть полезен как начинающим, так и опытным инвесторам для оперативного получения аналитической информации о поведении акций. Проект также демонстрирует применение современных технологий — от анализа временных рядов и расчёта технических индикаторов до машинного обучения и взаимодействия с мессенджер-интерфейсами — в решении прикладных задач в области финансов.

Таким образом, разработанная система представляет собой комплексный инструмент, объединяющий в себе сбор данных, их обработку, интеллектуальный анализ и удобную визуализацию, доступную прямо в мессенджере. Проект может быть расширен в дальнейшем путём добавления новых типов прогнозов, интеграции с другими источниками данных, а также обучения более сложных моделей, включая нейросетевые архитектуры.

# ****Глава 1. Теоретическая часть.****

## Задача и проблемы Data Science****.****

### Введение в Data Science

Data Science (наука о данных) — это область знаний, объединяющая статистику, информатику, математику и бизнес-аналитику для извлечения ценной информации из данных. В современном мире объемы данных растут экспоненциально, и организации сталкиваются с необходимостью эффективно использовать эту информацию для принятия решений, оптимизации процессов и создания новых продуктов.

Data Science включает полный цикл работы с данными — от их сбора и очистки до построения прогностических моделей и визуализации результатов. Главная цель — получение новых знаний и инсайтов, которые могут повысить конкурентоспособность бизнеса или способствовать научным открытиям.

### Основные задачи Data Science

1. **Сбор данных**

Сбор данных — фундаментальный этап, который включает получение информации из различных источников: базы данных, веб-сайты, API, датчики IoT, текстовые документы и др. Важной проблемой является обеспечение полноты, достоверности и актуальности данных.

1. **Обработка и очистка данных**

Сырые данные редко бывают готовы к анализу. Обычно они содержат пропуски, ошибки, дубликаты и разнородные форматы. Этап очистки данных — критически важный, так как качество исходных данных напрямую влияет на качество результатов. Применяются методы заполнения пропусков, фильтрации выбросов, преобразования форматов и нормализации.

1. **Исследовательский анализ данных (Exploratory Data Analysis, EDA)**

На этом этапе проводится визуализация и статистический анализ, чтобы выявить закономерности, зависимости, тренды и аномалии. EDA помогает понять структуру данных, выбрать признаки для моделей и выдвинуть гипотезы.

1. **Построение и обучение моделей**

Основная задача — разработка моделей машинного обучения или статистических моделей, которые могут решать прикладные задачи: классификация, регрессия, кластеризация, прогнозирование и др. Выбор алгоритмов зависит от задачи, объема и характеристик данных.

1. **Оценка и валидация моделей**

После обучения модели необходимо оценить ее качество с помощью метрик (например, точности, F1-меры, RMSE) и проверить обобщающую способность на новых данных. Валидация позволяет избежать переобучения и повысить надежность прогноза.

1. **Интерпретация и объяснение результатов**

Важной частью является понимание, почему модель делает именно такие прогнозы или классификации. Это особенно важно в бизнесе, медицине и финансах, где от объяснимости зависит доверие к системе.

1. **Внедрение и мониторинг**

Итоговая задача — интегрировать модель в бизнес-процессы или программные решения, обеспечить автоматическое обновление и мониторинг производительности в реальном времени.

### Основные проблемы и вызовы в Data Science

Несмотря на широкие возможности, Data Science сталкивается с рядом серьезных проблем:

* **Качество данных**

Данные часто неполные, имеют ошибки, противоречивы или устарели. Обработка таких данных требует значительных ресурсов и опыта.

* **Объем и разнообразие данных (Big Data)**

Современные данные бывают большими (объем, скорость, разнообразие), что требует масштабируемых решений для хранения и обработки.

* **Выбор подходящих алгоритмов и параметров**

Существует множество методов машинного обучения и статистики. Подбор оптимального алгоритма и гиперпараметров — сложная задача, требующая экспериментов и глубокого понимания предметной области.

* **Интерпретируемость моделей**

Модели типа глубоких нейронных сетей часто считаются «черными ящиками», что затрудняет понимание и объяснение их решений. Для бизнес-приложений и регулирования это может быть критичным.

* **Этические и юридические аспекты**

Использование персональных данных требует соблюдения законодательства (например, GDPR), а также внимательного отношения к вопросам приватности, безопасности и этики.

* **Инфраструктурные ограничения**

Для работы с большими данными и сложными моделями нужны мощные вычислительные ресурсы, которые могут быть дорогими и требуют специальных знаний для настройки.

* **Недостаток квалифицированных специалистов**

Data Science — мультидисциплинарная область, где нужно сочетать навыки программирования, статистики и знания предметной области. Это создает дефицит квалифицированных кадров.

### Data Science в финансовой сфере и прогнозировании цен акций

Финансовый рынок — одна из ключевых сфер применения Data Science. Анализ исторических данных, построение прогностических моделей и автоматизация торговых решений позволяют снизить риски, повысить доходность и оперативно реагировать на изменения.

Однако финансовые данные имеют свои особенности и сложности:

* Высокая волатильность и шумность данных
* Влияние множества факторов (макроэкономических, политических, социальных)
* Неустойчивость и изменчивость рыночных закономерностей
* Ограниченность и задержки данных

В связи с этим задача прогнозирования цен акций — одна из сложных и востребованных в Data Science. Построение качественных моделей требует комплексного подхода, начиная с корректного сбора и обработки данных и заканчивая тщательной валидацией и интерпретацией результатов.

## 1.2. Сбор данных.

Сбор данных — это первый и один из наиболее критически важных этапов в процессе анализа данных. Именно от качества, полноты и репрезентативности собранных данных зависит эффективность последующего анализа и построения моделей. На этапе сбора закладывается основа для всех дальнейших шагов: от очистки и обработки до визуализации и машинного обучения.

### Источники данных

В зависимости от предметной области и поставленной задачи, источники данных могут быть различными. В контексте финансового анализа и прогнозирования цен акций, основными источниками являются:

* **Биржевые платформы и агрегаторы (например, Московская биржа, Yahoo Finance, Alpha Vantage, Bloomberg)** — предоставляют исторические и текущие данные о котировках акций, объемах торгов, индексов и других финансовых инструментов.
* **Открытые API** — позволяют автоматизировать сбор информации и получать данные в машиночитаемом виде. Пример — библиотека moexalgo, с помощью которой можно получать исторические свечи по тикерам с Московской биржи.
* **Финансовая отчетность компаний** — может использоваться как дополнительный источник признаков (например, выручка, прибыль, активы), доступен на сайтах эмитентов или агрегаторах (например, investing.com).
* **Экономические и макроэкономические показатели** — данные о ставках, инфляции, ВВП и других экономических индикаторах, которые могут влиять на стоимость ценных бумаг.
* **Новостные ленты и социальные сети** — всё чаще используются в Data Science проектах для анализа тональности и событий, способных повлиять на поведение рынка.

### Методы получения данных

Существуют различные технические подходы к сбору данных:

1. **Web scraping (парсинг сайтов)**

Метод автоматического извлечения информации с веб-страниц с помощью инструментов вроде BeautifulSoup, Selenium, Scrapy. Он эффективен, когда данные недоступны через API, но требует устойчивости к изменению структуры страниц и соблюдения правил использования сайта.

1. **Использование API**

Более надёжный и структурированный способ. Многие биржи и финансовые платформы предоставляют API-интерфейсы, которые позволяют получать данные в формате JSON или CSV. Например, MOEX предоставляет API для получения информации о торгах и котировках.

1. **Импорт из файлов**

Исторические данные могут быть доступны в виде файлов (CSV, Excel), загружаемых вручную или по расписанию. Это подходяще для офлайн-анализа или при использовании архивов данных.

1. **Сбор в режиме реального времени (real-time streaming)**

Для задач, требующих оперативного реагирования (например, торговые роботы), применяются системы потоковой обработки, такие как Kafka, Spark Streaming или WebSocket-соединения.

### Проблемы и вызовы при сборе данных

Процесс сбора данных сопровождается рядом трудностей:

* **Неполнота и пропущенные значения**

Некоторые источники предоставляют данные не за все даты или с пропусками, особенно в периоды праздников, технических сбоев или по неликвидным инструментам.

* **Разный формат и структура**

Данные из разных источников могут иметь различную структуру, что требует предварительного стандартизирования перед объединением.

* **Ограничения доступа и лицензирование**

Некоторые данные предоставляются на платной основе или имеют ограничения по количеству запросов. Это требует учета юридических и технических аспектов использования данных.

* **Наличие шумов и ошибок**

Биржевые данные могут содержать выбросы, ошибки в котировках или некорректные временные метки, особенно в случае низкой ликвидности или технических сбоев.

* **Задержки в обновлении**

Не все источники обеспечивают данные в режиме реального времени. При прогнозировании или построении торговых стратегий такие задержки могут существенно влиять на результат.

Сбор данных — это не просто механическая задача, а стратегически важный этап, от которого зависит вся последующая аналитика. Надежность, полнота и актуальность данных — основа успешного Data Science проекта. Грамотно организованный процесс сбора позволяет минимизировать ошибки и обеспечить устойчивость аналитической модели к реальным рыночным условиям.

## Обработка и очистка данных

После этапа сбора данных начинается один из самых ответственных процессов в рамках проекта Data Science — **обработка и очистка данных**. Никакая модель машинного обучения не сможет показать адекватные результаты, если обучается на «грязных» или нерепрезентативных данных. Качественная предобработка позволяет устранить ошибки, избавиться от выбросов, привести данные к единому формату и подготовить их к дальнейшему анализу и построению моделей.

### Цель обработки данных

Цель этапа обработки и очистки данных состоит в том, чтобы:

* устранить пропущенные значения;
* выявить и исправить или удалить аномалии и выбросы;
* преобразовать данные к корректным типам;
* нормализовать, масштабировать или агрегировать данные при необходимости;
* подготовить признаки (feature engineering), подходящие для последующего анализа и обучения модели.

### Основные задачи обработки данных

#### 1. Приведение типов данных

При загрузке данных с разных источников часто возникает необходимость привести колонки к нужным типам: числа — к числовым типам, даты — к типу datetime, категориальные признаки — к category или string. Это обеспечивает корректную работу аналитических функций и моделей.

#### 2. Обработка пропущенных значений

Пропущенные значения (NaN, None) могут появляться по множеству причин: отсутствие торгов, ошибки в источниках данных, несовпадения в форматах. Основные стратегии обработки:

* удаление строк или столбцов с большим количеством пропущенных значений;
* заполнение средним/медианным значением, если данные числовые;
* заполнение предыдущим или следующим значением по времени (forward fill, backward fill);
* использование специальных моделей или алгоритмов для иммутации пропусков (например, KNNImputer, IterativeImputer).

#### 3. Удаление или обработка выбросов

Выбросы — это значения, которые существенно отличаются от остальных наблюдений. В финансовых данных это может быть, например, резкий скачок цены за день без экономических причин (техническая ошибка).

Способы обнаружения и обработки:

* межквартильный размах (IQR);
* z-оценка (z-score);
* визуальный анализ (ящик с усами, boxplot);
* ручная фильтрация по логике бизнеса.

#### 4. Масштабирование и нормализация

Некоторые алгоритмы машинного обучения чувствительны к масштабам признаков (например, KNN, SVM, нейронные сети). Поэтому данные часто приводят к одному масштабу:

* **Min-Max Normalization** — приведение значений к диапазону [0, 1];
* **Standard Scaling** — вычитание среднего и деление на стандартное отклонение;
* **Robust Scaling** — масштабирование с учётом медианы и межквартильного размаха.

#### 5. Создание новых признаков

Feature engineering — один из ключевых этапов. В задачах финансового анализа это может быть:

* расчёт скользящих средних (MA, EMA);
* индикаторы (RSI, MACD, Bollinger Bands);
* логарифмическая доходность;
* волатильность;
* признаки календарного времени (день недели, месяц, квартал и т.д.).

#### 6. Объединение данных

Если данные поступают из разных источников (например, биржевые данные + макроэкономика), их необходимо соединить. Это делается с помощью функций объединения (merge, join) по дате, идентификатору актива и другим общим признакам.

### Трудности и риски

* **Потеря информации** — агрессивная фильтрация может привести к удалению полезных данных.
* **Смещение распределений** — заполнение пропусков статистиками может искажать распределения.
* **Переобработка** — излишняя трансформация может усложнить интерпретацию и ухудшить обобщающую способность модели.
* **Ошибки из-за человеческого фактора** — при ручной предобработке возможно нарушение логики данных.

Обработка и очистка данных — это не просто техническая операция, а важнейший этап подготовки, требующий понимания предметной области и внимательности. От того, насколько качественно выполнена предобработка, зависит корректность анализа и точность моделей. Правильная реализация этого этапа позволяет обеспечить устойчивость и надежность аналитических выводов и прогнозов.

## ****1.4.**** Исследовательский анализ данных (Exploratory Data Analysis, EDA)

**Исследовательский анализ данных (Exploratory Data Analysis, EDA)** — это этап, на котором происходит предварительное изучение собранных и очищенных данных. Его цель — выявить основные закономерности, структуры, аномалии и взаимосвязи в данных до применения алгоритмов машинного обучения. Этот этап помогает лучше понять данные, построить гипотезы и принять обоснованные решения по дальнейшей обработке и построению признаков.

### Задачи EDA

К основным задачам EDA относятся:

* Проверка структуры и распределения данных;
* Поиск закономерностей, корреляций и зависимостей между переменными;
* Обнаружение аномалий и выбросов;
* Оценка распределения целевой переменной;
* Выявление скрытых шаблонов;
* Проверка гипотез о влиянии факторов;
* Принятие решений о необходимости трансформации признаков.

EDA может проводиться как в табличной, так и в визуальной форме с использованием графиков и диаграмм.

### Основные методы и инструменты EDA

#### 1. Описательная статистика

Первое, что делается на этапе EDA — это анализ ключевых статистик:

* Среднее значение (mean)
* Медиана (median)
* Мода (mode)
* Минимум и максимум
* Стандартное отклонение (std)
* Межквартильный размах (IQR)

df.describe()

Этот простой вызов позволяет быстро понять масштаб данных, наличие возможных выбросов и асимметрию распределения.

#### 2. Визуализация распределений

Для понимания формы распределения признаков и целевой переменной используют:

* **Гистограммы** — позволяют увидеть, насколько равномерно распределены данные;
* **Boxplot (ящик с усами)** — помогает обнаружить выбросы;
* **Кернельная оценка плотности (KDE)** — сглаженное представление гистограммы.

import seaborn as sns

sns.histplot(df['close'], kde=True)

#### 3. Анализ взаимосвязей

Изучение взаимосвязей между признаками проводится с помощью:

* **Корреляционной матрицы** — показывает силу линейной зависимости между признаками;
* **Диаграмм рассеяния (scatter plots)** — визуализируют парные зависимости;
* **Тепловых карт (heatmaps)** — отображают корреляции в цветовой форме.

import seaborn as sns

import matplotlib.pyplot as plt

corr = df.corr(numeric\_only=True)

plt.figure(figsize=(10, 8))

sns.heatmap(corr, annot=True, cmap='coolwarm')

#### 4. Анализ временных рядов

В задачах, связанных с финансовыми данными (как в этом дипломном проекте), особое внимание уделяется **временным рядам**. Они анализируются на предмет:

* трендов;
* сезонности;
* автокорреляции;
* структурных разрывов (например, экономических кризисов).

plt.plot(df['begin'], df['close'])

plt.title('Динамика закрытия цен')

plt.xlabel('Дата')

plt.ylabel('Цена закрытия')

plt.grid(True)

Конец формы

#### 5. Группировка и агрегация

EDA также включает в себя анализ сгруппированных данных:

* по категориям (например, по дням недели, месяцам);
* по уровням признаков (высокая/низкая цена);
* по временным отрезкам (неделя, месяц).

df['weekday'] = df['begin'].dt.day\_name()

df.groupby('weekday')['close'].mean().sort\_values()

### Важность EDA

EDA позволяет:

* Предотвратить ошибки, связанные с ошибочными предположениями о данных;
* Найти неожиданную структуру, которую можно использовать в моделировании;
* Выявить нерелевантные или избыточные признаки;
* Подтвердить или опровергнуть гипотезы;
* Подготовить основу для выбора моделей машинного обучения и метрик оценки.

EDA — это не разовая операция, а **итеративный процесс**, к которому возвращаются на каждом этапе анализа.

Исследовательский анализ данных играет критически важную роль в проекте Data Science. Он обеспечивает понимание структуры данных, позволяет принимать обоснованные решения на последующих этапах и закладывает основу для качественного построения моделей. Пропуск этого этапа часто приводит к слабым и ошибочным результатам анализа.

## ****1.5. Построение и обучение моделей****

## Построение и обучение моделей

Построение и обучение моделей — один из ключевых этапов процесса анализа данных и разработки решений в рамках Data Science. После того как данные собраны, очищены и исследованы, наступает момент, когда необходимо перейти от анализа к созданию математической модели, которая будет способна выявлять скрытые закономерности в данных и использовать их для предсказаний или принятия решений.

Модель машинного обучения представляет собой математическую структуру, которая «учится» на исторических данных, чтобы затем применять полученные знания к новым, ещё не известным данным. Этот этап требует как глубокого понимания природы данных, так и знания соответствующих алгоритмов и их свойств.

### Типы задач, решаемые моделями

Перед началом построения модели важно четко понимать, какую задачу она должна решать. В зависимости от типа целевой переменной задачи делятся на:

* **Регрессию** — если требуется предсказать числовое значение (например, цену акции на следующий день).
* **Классификацию** — если необходимо определить категорию, к которой относится объект (например, классификация клиентов по уровню риска).
* **Кластеризацию** — если нужно автоматически разбить объекты на группы без заранее заданных меток.
* **Ранжирование** — если требуется упорядочить объекты по степени важности или вероятности события.

Для каждой из этих задач существуют специализированные алгоритмы, подходы и особенности подготовки данных.

### Подготовка признаков и выбор переменных

Перед непосредственным обучением модели данные проходят этап подготовки, называемый **feature engineering** — инженерия признаков. Его цель — преобразовать данные в такой вид, который наиболее точно и эффективно сможет быть интерпретирован алгоритмом обучения.

Сюда входят следующие этапы:

* **Выбор признаков** — отбор только тех переменных, которые имеют потенциальную ценность для построения модели.
* **Создание новых признаков** — на основе имеющихся можно формировать производные переменные, например: технические индикаторы (скользящие средние, индекс относительной силы и т. д.), разности, соотношения между переменными.
* **Нормализация и стандартизация** — преобразование данных к общему масштабу, особенно важно для моделей, чувствительных к величинам признаков (например, линейная регрессия, SVM).
* **Кодирование категориальных признаков** — преобразование текстовых меток в числовую форму с помощью методов one-hot encoding, label encoding и др.
* **Обработка пропущенных значений** — замена недостающих данных средними, медианными значениями или предсказанными значениями на основе других признаков.

Качественная подготовка признаков во многом определяет конечную эффективность модели. На практике часто именно этот этап требует больше всего времени и усилий.

### Выбор модели

На следующем этапе необходимо выбрать алгоритм или набор алгоритмов, которые будут использоваться для обучения модели. Выбор зависит от:

* **Типа задачи** (регрессия, классификация и т.д.),
* **Характеристик данных** (размер выборки, число признаков, наличие категориальных переменных),
* **Необходимости интерпретируемости** (например, линейные модели легко интерпретировать, а нейросети — нет),
* **Скорости обучения и предсказания**, если модель планируется использовать в реальном времени.

Наиболее часто используемые модели включают:

* **Линейные модели** (линейная и логистическая регрессии) — просты и интерпретируемы, хорошо работают с линейными зависимостями.
* **Деревья решений (Decision Trees)** — удобны для понимания, справляются с нелинейными зависимостями.
* **Случайный лес (Random Forest)** — ансамбль деревьев, повышает устойчивость и точность.
* **Градиентный бустинг (например, XGBoost, LightGBM, CatBoost)** — мощный и популярный метод, показывающий высокие результаты на табличных данных.
* **Методы опорных векторов (SVM)** — эффективны для задач с чёткой границей между классами, но плохо масштабируются.
* **Нейронные сети** — особенно эффективны при наличии большого объема данных и сложных зависимостей.

Каждая модель имеет свои преимущества и ограничения. В практике Data Science часто применяется **итеративный подход**, при котором разные модели сравниваются между собой, и выбирается наилучшая.

### Процесс обучения модели

После выбора алгоритма модель обучается на исторических данных — это означает, что алгоритм находит внутренние параметры, которые минимизируют ошибку предсказания на обучающей выборке. Обучение может быть:

* **Супервизируемым** — при наличии целевой переменной, по которой производится обучение (наиболее распространённый подход);
* **Несу­первизируемым** — при отсутствии целевой переменной (например, кластеризация);
* **С подкреплением** — модель учится через взаимодействие с окружающей средой, получая награду за правильные действия.

Цель обучения — **найти закономерности в данных**, которые позволят делать точные предсказания на новых примерах. Модели подбирают коэффициенты, веса или структуры (например, глубину дерева, форму нейронной сети), чтобы минимизировать функцию потерь (loss function), отражающую расхождение между реальными значениями и предсказаниями.

Важно обеспечить **баланс между недообучением и переобучением**:

* **Недообучение** происходит, когда модель слишком простая и не способна уловить сложные закономерности.
* **Переобучение** возникает, когда модель слишком точно запоминает обучающую выборку и плохо обобщает на новые данные.

Чтобы этого избежать, используют регуляризацию, подбор гиперпараметров, увеличение данных, а также разделение на обучающую и тестовую выборки (оценка и валидация рассматриваются отдельно).

### Настройка гиперпараметров

Многие модели содержат параметры, не обучающиеся на данных напрямую, а задаваемые вручную — так называемые **гиперпараметры**. Например, число деревьев в случайном лесе, глубина дерева, скорость обучения в градиентном бустинге, коэффициенты регуляризации в линейных моделях.

Выбор правильных значений гиперпараметров критически важен для достижения наилучших результатов. Для этого применяются:

* **Grid Search** — полный перебор заданных параметров по сетке.
* **Random Search** — случайный перебор параметров.
* **Байесовская оптимизация** — более интеллектуальный подход, учитывающий предыдущие результаты.
* **AutoML** — автоматический подбор моделей и гиперпараметров.

Построение и обучение модели — это сердцевина проекта Data Science. Этот этап требует не только глубоких технических знаний, но и творческого подхода. Грамотный выбор признаков, моделей и параметров способен существенно повлиять на точность прогнозов и успешность всей аналитической системы. Без качественного этапа моделирования даже самые чистые и подробные данные не принесут желаемой пользы.

## 1.6. Оценка и валидация моделей

После того как модель обучена на тренировочных данных, необходимо проверить, насколько хорошо она справляется со своей задачей. Оценка и валидация моделей — важнейший этап в процессе машинного обучения, позволяющий убедиться, что модель не просто «запомнила» обучающую выборку, но способна делать корректные предсказания на новых, ранее не встречавшихся данных. Этот этап помогает выявить переобучение, сравнить несколько моделей между собой и выбрать наилучшее решение.

### Цели оценки моделей

Основные задачи оценки моделей включают:

* **Проверка обобщающей способности** — насколько хорошо модель предсказывает на новых данных;
* **Выбор лучшей модели** — сравнение нескольких алгоритмов или версий одной модели;
* **Настройка гиперпараметров** — оценка качества модели при различных конфигурациях;
* **Определение точности, устойчивости и надёжности модели**.

### Разделение данных на выборки

Для корректной оценки необходимо разделить данные на несколько частей:

* **Обучающая выборка (Training set)** — используется непосредственно для обучения модели;
* **Валидационная выборка (Validation set)** — применяется для настройки гиперпараметров и отбора модели;
* **Тестовая выборка (Test set)** — предназначена для финальной оценки модели после всех настроек.

Часто используется схема **train/test split** — например, 80% данных идут на обучение, 20% — на тест. Также часто применяют **k-fold кросс-валидацию**, которая позволяет более устойчиво оценить модель, разделяя данные на K равных частей и последовательно обучая и проверяя модель на каждой из них.

### Метрики оценки качества моделей

Выбор метрики зависит от типа задачи:

#### Для задач классификации:

* **Accuracy (точность классификации)** — доля правильно классифицированных объектов:

где TP — истинно положительные, TN — истинно отрицательные, FP — ложно положительные, FN — ложно отрицательные.

* **Precision (точность)** — доля верных положительных предсказаний среди всех положительных:
* **Recall (полнота)** — доля верных положительных предсказаний среди всех истинных положительных:
* **F1-score** — гармоническое среднее между точностью и полнотой:
* **ROC AUC (площадь под ROC-кривой)** — обобщённая метрика качества бинарной классификации, отражающая способность модели отличать классы.

#### Для задач регрессии:

* **MAE (Mean Absolute Error)** — средняя абсолютная ошибка:

***MAE =***

* **MSE (Mean Squared Error)** — среднеквадратичная ошибка:

***MSE =***

* **RMSE (Root Mean Squared Error)** — корень из среднеквадратичной ошибки:
* **R² (коэффициент детерминации)** — доля дисперсии, объяснённой моделью:

### Кросс-валидация (Cross-validation)

**Кросс-валидация** — это методика, позволяющая получить более устойчивую и надёжную оценку модели. Самый распространённый вариант — **k-fold cross-validation**, где данные разбиваются на k частей (фолдов). На каждом шаге одна часть используется как валидационная, остальные — как обучающие. Процесс повторяется k раз, и финальный результат усредняется.

Преимущества кросс-валидации:

* Устраняет зависимость от случайного разбиения данных;
* Помогает выявить переобучение;
* Повышает точность оценки.

### Предотвращение переобучения

При чрезмерно точном запоминании обучающих данных модель теряет способность обобщать, что снижает её эффективность на тестовых данных. Методы борьбы с переобучением:

* **Регуляризация** (например, L1, L2) — штраф за сложность модели;
* **Уменьшение количества признаков** — удаление нерелевантных или избыточных признаков;
* **Увеличение объема данных** — генерация новых данных или использование данных с дополнениями;
* **Применение кросс-валидации**;
* **Использование простых моделей** — например, линейных моделей вместо сложных нейросетей.

### Сравнение моделей

На этапе оценки можно сравнивать несколько моделей между собой по результатам на одинаковой тестовой или валидационной выборке. Выбор лучшей модели должен учитывать:

* Высокие значения ключевых метрик;
* Стабильность результатов (например, через стандартное отклонение по фолдам кросс-валидации);
* Устойчивость к шуму и пропущенным данным;
* Интерпретируемость (если это важно для прикладной задачи).

Этап оценки и валидации модели критически важен для построения надёжной системы машинного обучения. Он позволяет убедиться, что модель не только хорошо работает на обучающих данных, но и способна адекватно предсказывать на новых. Правильное применение метрик, разбиение данных и кросс-валидация помогают избежать переобучения и выбрать наиболее эффективную модель для дальнейшего использования.

## 1.7 Интерпретация и объяснение результатов

В задачах анализа данных и машинного обучения не всегда достаточно просто построить высокоточное предсказание. Во многих сферах (финансы, здравоохранение, юриспруденция, маркетинг и др.) необходимо **понять**, **почему** модель приняла то или иное решение. Интерпретация модели помогает:

* Повысить доверие пользователей и заказчиков к системе;
* Найти ошибки в данных или логике модели;
* Выявить ключевые факторы, влияющие на результат;
* Соблюдать юридические и этические требования (например, в соответствии с GDPR).

### Зачем нужна интерпретация

Интерпретируемость особенно важна в следующих случаях:

* **Высокая цена ошибки** — например, в медицинской диагностике;
* **Требования прозрачности** — например, при кредитовании или отборе кандидатов;
* **Необходимость принятия бизнес-решений** — руководителям важно понимать, почему рекомендовано то или иное действие;
* **Обучение пользователей** — если ИИ используется как обучающий инструмент.

### Интерпретируемые и не интерпретируемые модели

Модели можно условно разделить на:

| **Тип модели** | **Интерпретируемость** | **Пример** |
| --- | --- | --- |
| Прозрачные (white-box) | Высокая | Линейная регрессия, логистическая регрессия, решающее дерево |
| «Чёрный ящик» (black-box) | Низкая | Случайный лес, XGBoost, нейросети |

Для моделей с высокой интерпретируемостью часто достаточно посмотреть на коэффициенты (например, в логистической регрессии) или структуру дерева. Для сложных моделей требуются специальные методы.

### Методы интерпретации моделей

#### 1. Feature Importance (важность признаков)

Определяет, насколько каждый признак влияет на предсказания модели. Распространённые подходы:

* Встроенная важность признаков в моделях деревьев (например, feature\_importances\_ в RandomForest);
* Анализ коэффициентов в линейных моделях;
* Permutation importance — перестановка значений признака и оценка изменения метрики.

#### 2. SHAP (SHapley Additive exPlanations)

Один из самых популярных современных методов интерпретации. Основан на теории игр и рассчитывает вклад каждого признака в конкретное предсказание. Преимущества:

* Интерпретация индивидуальных предсказаний;
* Работа с любыми моделями;
* Графическая визуализация важности признаков.

#### 3. LIME (Local Interpretable Model-agnostic Explanations)

Строит локально аппроксимирующую интерпретируемую модель (например, линейную) вокруг одного предсказания. Полезен для объяснения конкретных решений модели.

#### 4. Partial Dependence Plots (PDP)

Показывают, как меняется предсказание модели при изменении значения одного признака, при фиксированных остальных. Позволяет оценить нелинейные зависимости.

#### 5. ICE-плоты (Individual Conditional Expectation)

Похожи на PDP, но отображают зависимости для каждого отдельного объекта. Это позволяет увидеть индивидуальные эффекты и выявить взаимодействия.

### Интерпретация в задачах классификации и регрессии

**Для классификации**:

* Можно проанализировать, какие признаки способствовали отнесению объекта к определённому классу.
* Используются вероятности классов, логарифмы шансов (log-odds) и значения функции принятия решений.

**Для регрессии**:

* Изучается влияние признаков на величину прогноза;
* Определяются признаки, увеличивающие или уменьшающие значение целевой переменной.

### Интерпретация ошибок модели

Также важно понимать, где и почему модель ошибается. Это можно сделать с помощью:

* **Анализа confusion matrix (матрицы ошибок)** в классификации;
* **Графиков остатков (residual plots)** в регрессии;
* **Исследования кейсов с наибольшим отклонением** — например, объектов с самой высокой ошибкой предсказания.

### Практическая значимость интерпретации

В реальных проектах интерпретация помогает:

* Улучшить качество модели (за счёт выявления неинформативных признаков или ошибок);
* Подготовить отчёты для заказчиков и руководства;
* Обеспечить юридическую защиту при использовании автоматических решений;
* Повысить прозрачность и этичность решений, принимаемых ИИ.

Интерпретация и объяснение результатов машинного обучения становятся всё более актуальными в эпоху широкого применения «чёрных ящиков». Даже при высоком качестве предсказаний важно понимать, **почему** модель даёт именно такие ответы. Это повышает доверие, способствует улучшению качества решений и облегчает интеграцию моделей в бизнес-процессы. Использование современных интерпретируемых инструментов (SHAP, LIME и др.) позволяет сделать даже сложные модели прозрачными и понятными.

## 1.8 Внедрение и мониторинг

После построения, обучения и оценки модели машинного обучения (ML), критически важным этапом является её внедрение в реальную среду, где она будет использоваться для автоматического принятия решений или поддержки бизнес-процессов. Однако процесс не ограничивается развертыванием — необходимо также организовать мониторинг, поддержку и обновление модели, чтобы обеспечить её стабильную и актуальную работу на протяжении всего жизненного цикла.

### Внедрение модели (Model Deployment)

Внедрение модели — это процесс интеграции обученной ML-модели в производственную (production) среду, где она сможет взаимодействовать с пользователями, другими системами или сервисами. Это может включать:

* **Интеграцию через API**: модель разворачивается как веб-сервис, принимающий запросы и возвращающий предсказания;
* **Интеграцию в мобильное или десктопное приложение**;
* **Интеграцию в систему отчетности или аналитики (например, Power BI)**;
* **Встраивание в автоматизированный рабочий процесс, CRM, ERP и другие бизнес-системы**.

#### Форматы развертывания

* **Онлайн (real-time)** — модель обрабатывает запросы и выдает результат в реальном времени. Важно для приложений, требующих моментального ответа (например, антифрод-систем).
* **Батчевый (batch)** — данные обрабатываются пакетами в определённое время (например, ежедневно формируется список клиентов для маркетинговой рассылки).
* **Edge-развёртывание** — модель запускается на локальных устройствах без подключения к серверу, актуально для IoT и мобильных приложений.

### Мониторинг модели

После внедрения важно постоянно следить за тем, как модель работает в реальной среде. Даже хорошо обученная модель может со временем «устареть» из-за изменений в данных, бизнес-процессах или поведении пользователей. Это явление называется **дрейфом модели** (model drift).

#### Что мониторить

1. **Качество предсказаний**:
   * метрики модели (accuracy, RMSE, AUC и др.);
   * отклонение предсказаний от фактических значений;
   * рост количества ошибок (например, увеличение жалоб от пользователей).
2. **Дрейф данных (Data Drift)**:
   * сравнение текущего распределения данных с обучающим;
   * появление новых значений признаков или их пропадание.
3. **Дрейф концепции (Concept Drift)**:
   * изменения во взаимосвязи между признаками и целевой переменной;
   * например, факторы, определяющие поведение клиента, изменились из-за внешних обстоятельств (кризис, сезон, тренды).
4. **Доступность и отказоустойчивость**:
   * наличие ошибок при обращении к API;
   * стабильность времени ответа модели;
   * отказоустойчивость и масштабируемость системы.

### Инструменты для внедрения и мониторинга

#### Внедрение

* **Docker** — для упаковки модели и всех зависимостей;
* **Flask, FastAPI** — для создания REST API;
* **MLflow** — для управления жизненным циклом моделей;
* **TensorFlow Serving, TorchServe** — специализированные серверы для развёртывания моделей.

#### Мониторинг

* **Prometheus + Grafana** — для визуального мониторинга метрик и логов;
* **Evidently AI** — библиотека для отслеживания дрейфа данных и качества модели;
* **MLflow Tracking** — ведение истории моделей и экспериментов;
* **Kibana + ELK Stack** — для логирования и анализа производственных ошибок.

### Обновление и переобучение модели

Даже лучшие модели со временем теряют точность. Чтобы сохранить актуальность, необходимо:

* Автоматизировать процесс повторного обучения (например, с помощью Airflow);
* Обновлять данные, на которых обучается модель;
* Оценивать необходимость изменений с учётом мониторинга;
* Использовать **A/B-тестирование** новых версий моделей перед полным внедрением;
* Хранить версии моделей и логировать все изменения (model versioning).

### Проблемы на этапе внедрения

1. **Технические** — несовместимость с инфраструктурой, нехватка вычислительных ресурсов, задержки;
2. **Бизнес-проблемы** — сопротивление пользователей, непонимание работы модели, отсутствие доверия;
3. **Этические** — непрозрачные алгоритмы, дискриминация, неправомерное использование данных;
4. **Организационные** — отсутствие команд DevOps/MLOps, нехватка специалистов для сопровождения.

### Роль MLOps

MLOps (Machine Learning Operations) — это подход к построению инфраструктуры и процессов, обеспечивающих быструю, надёжную и управляемую разработку, внедрение и эксплуатацию моделей машинного обучения. Аналогично DevOps в программной инженерии, MLOps включает:

* автоматизацию обучения и развертывания;
* управление версиями моделей;
* CI/CD (непрерывная интеграция и доставка);
* мониторинг и логирование;
* безопасность и соответствие требованиям.

Внедрение и мониторинг модели являются критически важными этапами жизненного цикла проекта Data Science. Успешная реализация требует не только качественной модели, но и продуманной инфраструктуры, регулярного контроля, оперативного реагирования на изменения в данных и активного взаимодействия с пользователями. Только в этом случае возможно достичь реальной пользы от модели в бизнесе или социальной сфере.

# Глава 2. Практическая часть

## ****2.1. Общая постановка задачи****

В данной главе представлена реализация Telegram-бота, разработанного в рамках дипломной работы. Бот предназначен для анализа выбранной акции, построения прогноза её цены на следующий день и отправки результатов пользователю в виде текста и графика. Для этого используются данные с Московской биржи (MOEX), технические индикаторы и модель машинного обучения XGBoost. Весь проект реализован на языке Python с использованием библиотеки aiogram для взаимодействия с Telegram.

Целью данной практической части является реализация Telegram-бота для анализа и прогнозирования цен акций на основе исторических данных. Бот использует технические индикаторы и модель машинного обучения (XGBoost) для предсказания следующей дневной цены выбранной акции и отправки результата пользователю в виде графика с пояснением.

## ****2.2. Ресурсы и инструменты****

Для реализации данного проекта используются следующие **ресурсы и инструменты:**

### ****Программные инструменты и библиотеки:****

1. **Python** — основной язык программирования проекта.
2. **Telegram Bot API** — для создания бота и взаимодействия с пользователем.
3. **Библиотека aiogram** — для асинхронной обработки команд и событий в Telegram-боте.
4. **moexalgo** — библиотека для получения исторических данных по акциям с Московской биржи (MOEX).
5. **pandas** — для обработки и анализа табличных данных.
6. **numpy** — для численных вычислений.
7. **ta** (technical analysis) — для расчёта технических индикаторов (SMA, RSI, MACD и др.).
8. **matplotlib** — для визуализации графика акций и прогноза.
9. **xgboost** — библиотека градиентного бустинга для обучения модели и предсказания цен.
10. **scikit-learn** — для кросс-валидации, оценки качества модели и масштабирования признаков.

### ****Сторонние сервисы и API:****

* **MOEX (Московская биржа)** — источник исторических данных по тикерам акций.
* **Telegram** — платформа для взаимодействия с пользователем через бота.

### ****Среда разработки:****

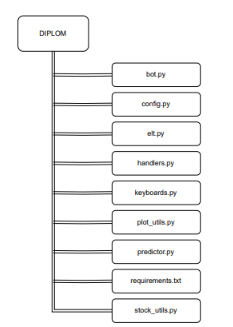
* **Visual Studio Code** или **Jupyter Notebook** — для написания и отладки кода.
* **Git** — система контроля версий (при необходимости).
* **Google Colab** (по желанию) — для тестирования моделей и визуализаций в облаке.

### ****Машинное обучение:****

* **XGBoostRegressor** — модель для прогноза цен.
* **Метрики качества**: RMSE (среднеквадратичная ошибка), MAPE (относительная ошибка).

## 2.3. Структура проекта и назначение файлов

Проект организован в виде модульной структуры, что позволяет упростить сопровождение, масштабирование и повторное использование кода. Каждый модуль выполняет отдельную функцию в рамках общей цепочки обработки запроса пользователя:



Проект состоит из следующих модулей:

1. **bot.py** – точка входа в программу. Запускает Telegram-бота и подключает роутеры

 Создаёт экземпляр Telegram-бота.

 Подключает обработчики (handlers.py).

 Запускает асинхронный цикл polling.

1. **handlers.py** – обработка входящих сообщений и взаимодействие с пользователем.

 Реализует функции-обработчики команд (/start, /predict, и др.).

 Обрабатывает выбор пользователя (тикер).

 Запускает цепочку анализа: elt.py → stock\_utils.py → predictor.py → plot\_utils.py.

 Отправляет пользователю результат.

1. **keyboards.py** – формирование интерфейса кнопок для выбора акций.

 Генерирует inline-кнопки с тикерами акций и командами.

 Используется в handlers.py для взаимодействия.

1. **elt.py** – преобразование и подготовка данных: добавление технических индикаторов и создание признаков/целевой переменной для модели.

 Выполняет начальную очистку и агрегацию данных (например, по дневным свечам).

 Добавляет технические индикаторы (SMA, EMA, RSI, MACD и др.).

 Вычисляет признаки для модели.

 Возвращает DataFrame с признаками и целевой переменной.

* **stock\_utils.py** – загрузка исторических данных акций с MOEX через библиотеку moexalgo.
* Загружает исторические данные с MOEX (через библиотеку moexalgo).
* Передаёт данные для дальнейшей обработки.

1. **predictor.py** – обучение модели XGBoost с кросс-валидацией и прогнозирование цены.

* Делит данные на train/test.
* Обучает модель XGBoost (с параметрами по умолчанию или из config.py).
* Выдаёт прогноз цены на следующий день.
* (Опционально) сохраняет обученную модель.

1. **plot\_utils.py** – генерация графика с прогнозом в виде изображения.

 Строит график на основе исторических данных и прогнозной точки.

 Сохраняет изображение графика.

 Возвращает путь к изображению для отправки пользователю.

1. **requirements.txt –** список зависимостей проекта
2. **config.py** – содержит конфигурационные данные.

## Общий алгоритм работы бота

Работа Telegram-бота начинается с запуска файла bot.py, в котором инициализируется бот и диспетчер (Dispatcher) библиотеки aiogram. Бот регистрирует обработчики команд из модуля handlers.py, а также настраивает клавиатуры из keyboards.py.

После запуска пользователь нажимает на ‘START’ (*Приложение 1*) и в появившемся меню (*Приложение 2*) выбирает интересующую его акцию из списка, что инициирует процесс анализа (*Приложение 3*).

**Получение исторических данных** — используется функция из модуля stock\_utils.py, которая через библиотеку moexalgo получает данные о цене выбранной акции.

**Обработка и агрегация данных** — модуль elt.py проводит очистку, форматирование и добавление технических индикаторов к историческим данным.

**Прогнозирование** — в predictor.py обучается модель XGBoost (или используется предобученная модель), которая предсказывает цену на следующий день.

**Построение графика** — в plot\_utils.py создаётся график с историей и прогнозом.

**Отправка пользователю** — обработчики из handlers.py формируют финальное сообщение с прогнозом и отправляют его через Telegram, (*Приложение 4*).

## 2.4. Инициализация и запуск Telegram-бота

Модули bot.py и config.py отвечают за инициализацию и запуск Telegram-бота, а также за подключение необходимых компонентов проекта.

#### Файл config.py

#### Этот файл содержит конфигурационные данные, в частности токен, полученный при регистрации бота через BotFather. Он необходим для авторизации и взаимодействия с Telegram API.

BOT\_TOKEN = "7909152492:AAGXod..... "

#### Файл bot.py

Файл bot.py является точкой входа в приложение. В нём реализован асинхронный запуск бота с использованием библиотеки aiogram. Код выглядит следующим образом:

Импорт библиотек и модулей

import asyncio

from aiogram import Bot, Dispatcher

from config import BOT\_TOKEN

from handlers import router

Асинхронная функция (main)

async def main():

    bot = Bot(token=BOT\_TOKEN)

    dp = Dispatcher()

    dp.include\_router(router)

    print("✅ Бот запущен.")

    await dp.start\_polling(bot)

if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.run(main())

* async def main(): — основная асинхронная функция, в которой создаются объекты бота и диспетчера (Dispatcher).
* bot = Bot(token=BOT\_TOKEN) — инициализация экземпляра Telegram-бота с использованием токена из файла config.py.
* dp = Dispatcher() — создание диспетчера, отвечающего за обработку входящих сообщений и событий.
* dp.include\_router(router) — подключение маршрутизатора с зарегистрированными обработчиками команд и нажатий кнопок (из модуля handlers.py).
* await dp.start\_polling(bot) — запуск цикла опроса (polling), при котором бот непрерывно слушает входящие сообщения от пользователей.
* asyncio.run(main()) — запуск асинхронной функции main() при старте скрипта.

Этот модуль обеспечивает инициализацию всех компонентов бота и связывает между собой логику проекта через подключённые обработчики. Таким образом, при запуске бота пользователь может сразу начать взаимодействие с ним через Telegram.

## 2.5. Обработка пользовательских запросов

Обработка взаимодействия с пользователем в Telegram-боте осуществляется в модуле handlers.py. В нем реализованы обработчики команды /start и нажатий на кнопки выбора акций. Также именно в этом модуле происходит вызов всех ключевых этапов анализа: от загрузки и подготовки данных до построения прогноза и отправки результата пользователю. Дополнительно используется модуль keyboards.py, отвечающий за формирование inline-клавиатур с доступными тикерами акций.

### Генерация кнопок с тикерами акций (модуль keyboards.py)

В этом модуле содержится словарь TICKERS, в котором перечислены доступные акции и соответствующие им тикеры Московской биржи:

TICKERS = {

    "Сбербанк": "SBER.ME",

    "Газпром": "GAZP.ME",

    "Лукойл": "LKOH.ME",

    "Яндекс": "YNDX.ME",

    "Татнефть": "TATN.ME"

}

Функция get\_stock\_keyboard() создает inline-клавиатуру, в которой каждая кнопка отображает название компании, а при нажатии передает в callback соответствующий тикер:

def get\_stock\_keyboard():

    keyboard = [

        [InlineKeyboardButton(text=name, callback\_data=ticker)]

        for name, ticker in TICKERS.items()

    ]

    return InlineKeyboardMarkup(inline\_keyboard=keyboard)

#### Этот механизм позволяет пользователю выбрать интересующую его акцию для анализа.

#### Модуль handlers.py

Функция start\_command вызывается при получении сообщения /start. Она отправляет приветственное сообщение и клавиатуру с выбором акций (*Приложение 1*):

Обработчик команды /start

@router.message(F.text == "/start")

async def start\_command(message: Message):

    # Отправка приветственного сообщения с кнопками выбора акции

    await message.answer("Привет! Выберите акцию для анализа:", reply\_markup=get\_stock\_keyboard())

Здесь используется фильтр F.text == "/start" для регистрации команды, а функция get\_stock\_keyboard() добавляет клавиатуру с кнопками акций. При запуске команды /start пользователю отображается сообщение с приветствием и клавиатура с кнопками выбора акции. (*Приложение 2*)

### Обработка выбора акции и полный цикл анализа

Основная логика работы бота находится в обработчике stock\_selected, который вызывается, когда пользователь нажимает на одну из кнопок с тикерами.

@router.callback\_query()

async def stock\_selected(callback: CallbackQuery):

    ticker = callback.data

    await callback.message.edit\_text(f"🔄 Загружаю данные по {ticker}...")

#### Получение тикера и сообщение о начале анализа

#### Сначала извлекается значение callback.data, содержащее тикер акции, бот уведомляет пользователя о начале анализа. (*Приложение 3*)

#### Загрузка исторических данных

#### Вызов функции из модуля stock\_utils.py:

# Загрузка данных

    df = load\_stock\_data(ticker)

Если данные успешно загружены (df is not None), начинается их обработка.

if df is not None:

        # Преобразование данных

        processed\_df = transform\_data(df)

        # Подготовка данных для модели

        X, y = prepare\_data\_for\_model(processed\_df)

        # Обучение модели

        model, rmse = train\_model\_with\_cv(X, y)

        # Прогнозирование

        predicted\_price = predict\_price(model, X)  # Передаем модель и данные для прогноза

        predicted\_date = processed\_df['Date'].iloc[-1] + pd.Timedelta(days=1)

        # Относительная ошибка в процентах

        relative\_error\_pct = (rmse / predicted\_price) \* 100

        # Визуализация прогноза

        image\_stream = plot\_forecast(

            processed\_df['Date'],

            processed\_df['Close'],

            predicted\_date,

            predicted\_price

        )

Функция transform\_data из elt.py очищает данные, агрегирует их, рассчитывает технические индикаторы и готовит к анализу.   
Массивы X (признаки) и y (целевая переменная — цена закрытия следующего дня) формируются для обучения модели.  
Функция train\_model\_with\_cv обучает модель XGBoost и возвращает:

* обученную модель,
* среднеквадратичную ошибку (RMSE), полученную в результате кросс-валидации.

Прогноз цены на следующий день формируется на основе обученной модели и последних значений признаков. Также рассчитывается дата прогноза (на день вперед от последнего значения в данных).  
Относительная ошибка прогноза позволяет пользователю оценить уровень доверия к результату.

Функция plot\_forecast строит график с историческими данными и точкой прогноза. Результат — объект BytesIO.

Иначе — пользователю отправляется сообщение об ошибке:

    else:

        await callback.message.answer("⚠️ Не удалось загрузить данные. Попробуйте позже.")

        # Отправка кнопок для выбора акции снова

        await callback.message.answer("Выберите другую акцию для анализа:", reply\_markup=get\_stock\_keyboard())

Подготовка изображения к отправке

 # Создаём BufferedInputFile из потока

        plot\_image\_input = BufferedInputFile(

            file=image\_stream.getvalue(),

            filename="forecast.png"

        )

Из потока создается файл для отправки в Telegram.

Отправка прогноза пользователю

 # Отправка картинки

        await callback.message.answer\_photo(

            photo=plot\_image\_input,

            #caption=f"✅ Прогноз завершён.\n🔮 Прогнозируемая цена на {predicted\_date.date()}: {predicted\_price:.2f}"

             caption=(

                f"✅ Прогноз завершён.\n"

                f"🔮 Прогнозируемая цена на {predicted\_date.date()}: {predicted\_price:.2f}\n"

                f"📉 Ожидаемая средняя ошибка (RMSE): {rmse:.2f}\n"

                f"📊 Относительная ошибка: {relative\_error\_pct:.2f}%"

            )

        )

Пользователь получает изображение графика с прогнозом и текст с ключевыми метриками. (*Приложение 4*)

**Повторный выбор:**  
После отправки результата пользователю снова предлагается выбрать другую акцию для анализа (*Приложение 5*)

 # Отправка кнопок для выбора акции снова

        await callback.message.answer("Выберите другую акцию для анализа:", reply\_markup=get\_stock\_keyboard())

Модуль handlers.py координирует всю бизнес-логику Telegram-бота: от получения тикера до отправки прогноза. В нём задействованы все остальные ключевые модули проекта — stock\_utils.py, elt.py, predictor.py, plot\_utils.py, а также keyboards.py для интерфейса с пользователем. Благодаря модульной архитектуре код остаётся читаемым, масштабируемым и легко расширяемым.

## 2.6. Загрузка и подготовка данных

Для построения прогноза цен акций Telegram-боту необходимо получить достоверные исторические данные и подготовить их для анализа. Эти операции реализованы в двух модулях:

* stock\_utils.py — отвечает за загрузку данных с Московской биржи через библиотеку moexalgo;
* elt.py — выполняет обработку данных, включая расчёт технических индикаторов и подготовку признаков для модели машинного обучения.

### Загрузка исторических данных (stock\_utils.py)

В модуле stock\_utils.py реализована функция load\_stock\_data, которая получает данные по дневным свечам за два года по выбранному тикеру:

Импорт библиотек

from moexalgo import Ticker

import pandas as pd

from datetime import datetime, timedelta

Импортируются необходимые библиотеки: moexalgo — для получения данных с биржи, pandas — для работы с таблицами, datetime — для расчета дат.

Функция load\_stock\_data:

def load\_stock\_data(ticker\_with\_suffix: str) -> pd.DataFrame | None:

    try:

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

        two\_years\_ago = (datetime.now() - timedelta(days=730)).strftime('%Y-%m-%d')

Здесь рассчитываются две даты: текущая (today) и дата два года назад (two\_years\_ago). Данные будут загружены именно за этот диапазон.

 ticker\_symbol = ticker\_with\_suffix.replace(".ME", "")

        ticker = Ticker(ticker\_symbol)

Удаляется суффикс .ME, чтобы получить тикер в нужном формате для moexalgo.

df = ticker.candles(start=two\_years\_ago, end=today, period='1d')

        return df

Метод candles загружает дневные свечи ('1d') — открытие, максимум, минимум, закрытие и объём торгов. Если загрузка прошла успешно, возвращается DataFrame.

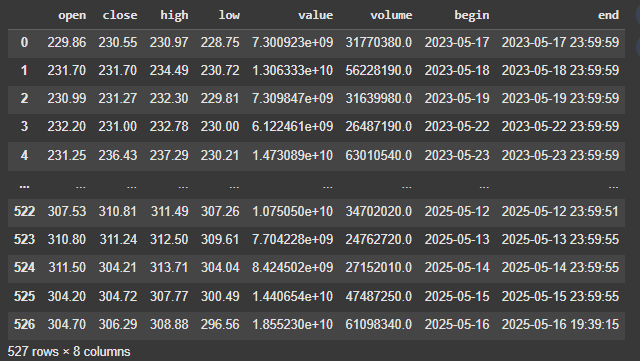
Обработка ошибок

except Exception as e:

        print(f"Ошибка загрузки данных по {ticker\_with\_suffix}: {e}")

        return None

В случае ошибки выводится сообщение в консоль, а функция возвращает None.  
В результате безошибочной работы данного модуля мы получаем DataFrame:



open: Цена открытия периода (например, дневная сессия).

close: Цена закрытия периода.

high: Максимальная цена за период.

low: Минимальная цена за период.

value: Общий объем торгов (суммарная стоимость всех сделок), выраженный в денежных единицах.

volume: Количество акций или контрактов, прошедших за указанный период.

begin: Начало временного интервала (обычно начало торгового дня).

end: Конец временного интервала (конец торгового дня).

### Преобразование и очистка данных (elt.py)

После загрузки данные необходимо привести к удобному виду, добавить технические индикаторы и подготовить признаки для модели.

#### Функция transform\_data

Импорт библиотек

import pandas as pd

import pandas\_ta as ta

Импортируется библиотека pandas и pandas\_ta — для вычисления технических индикаторов.

def transform\_data(df: pd.DataFrame) -> pd.DataFrame:

Функция получает на вход DataFrame, содержащий сырые свечные данные.

 # Переименовываем и выбираем нужные столбцы

    df = df.rename(columns={

        'begin': 'Date',

        'open': 'Open',

        'high': 'High',

        'low': 'Low',

        'close': 'Close',

        'volume': 'Volume'

    })

    df = df[['Date', 'Open', 'High', 'Low', 'Close', 'Volume']]

Столбцы переименовываются в стандартные названия. Это упрощает дальнейшую обработку. Оставляются только нужные поля.

Преобразуем дату в формат datetime, сортируем по дате и сбрасываем индексы для корректной обработки.

# Обеспечиваем корректный тип даты и сортировку

    df['Date'] = pd.to\_datetime(df['Date'])

    df = df.sort\_values('Date').reset\_index(drop=True)

    # Устанавливаем индекс по дате для расчётов индикаторов

    df.set\_index('Date', inplace=True)

Дата устанавливается как индекс — это требуется для корректной работы технических индикаторов.

### Добавление технических индикаторов (расширенное пояснение)

Технические индикаторы — это математические функции, основанные на исторических данных о ценах и объёмах торгов. Их основная цель — **выявление скрытых закономерностей** в движении цены, которые невозможно заметить при обычном визуальном анализе графика. В контексте машинного обучения они служат **признаками (features)**, на которых обучается модель и принимает решения о прогнозе цены.

Добавление индикаторов позволяет:

* определить наличие тренда (направленного движения);
* оценить силу и устойчивость движения;
* выявить точки перекупленности или перепроданности;
* зафиксировать моменты высокой волатильности.

Рассмотрим подробнее каждый из добавляемых индикаторов:

**SMA (Simple Moving Average)** — Простая скользящая средняя

# Добавляем простую скользящую среднюю (SMA) за 13 периодов

    df['SMA\_13'] = ta.sma(df['Close'], length=13)

усреднённое значение цены закрытия за последние 13 дней.

**Зачем нужен:**

* сглаживает краткосрочные колебания;
* показывает общее направление движения (восходящий/нисходящий тренд);
* помогает машине "увидеть", как текущая цена соотносится с недавней динамикой.

. **EMA (Exponential Moving Average)** — Экспоненциальная скользящая средняя

# Добавляем экспоненциальную скользящую среднюю (EMA) за 10 периодов

    df['EMA\_10'] = ta.ema(df['Close'], length=10)

аналог SMA, но с увеличенным весом на более свежие значения.

**Зачем нужен:**

* быстрее реагирует на последние изменения;
* помогает идентифицировать поворотные точки и изменения тренда.

**RSI (Relative Strength Index)** — Индекс относительной силы

    # Добавляем индекс относительной силы (RSI) за 14 периодов

    df['RSI\_14'] = ta.rsi(df['Close'], length=14)

оценивает скорость и изменения ценовых движений за последние 14 дней. Измеряется в диапазоне от 0 до 100.

**Зачем нужен:**

* RSI выше 70 → **перекупленность** (цена может скоро пойти вниз);
* RSI ниже 30 → **перепроданность** (цена может начать рост);
* полезен как сигнал к смене направления.

**MACD (Moving Average Convergence Divergence)** — Схождение/расхождение скользящих средних

# Добавляем MACD (схождение-расхождение скользящих средних)

    macd = ta.macd(df['Close'])

    df = pd.concat([df, macd], axis=1)

состоит из разницы между двумя EMA (12 и 26 периодов), сигнальной линии (EMA от этой разницы) и гистограммы.

**Зачем нужен:**

* один из самых популярных индикаторов силы тренда;
* пересечение линий MACD может сигнализировать о начале или завершении тренда;
* используется для предсказания ускорения/замедления движения цены.

**BBANDS (Bollinger Bands)** — Полосы Боллинджера

# Добавляем полосы Боллинджера (BBANDS) за 20 периодов и 2 стандартных отклонения

    bb = ta.bbands(df['Close'], length=20, std=2)

    df = pd.concat([df, bb], axis=1)

три линии, исполнены виде дороги:

* средняя (скользящая за 20 дней),
* верхняя (на 2 стандартных отклонения выше),
* нижняя (на 2 стандартных отклонения ниже).

**Зачем нужны:**

* показывает **волатильность** — когда полосы расширяются, волатильность растёт;
* цена, выходящая за верхнюю/нижнюю полосу, может свидетельствовать о переломе тренда;
* помогает находить точки входа и выхода (перекупленность/перепроданность).

Добавление технических индикаторов в качестве признаков для модели XGBoost позволяет:

* обогатить данные дополнительной смысловой информацией;
* сделать поведение модели более чувствительным к рыночным условиям;
* повысить точность прогноза за счёт комплексного анализа цены и её производных.

Индикаторы как признаки хорошо дополняют друг друга: одни характеризуют направление тренда, другие — его силу, а третьи — рыночную волатильность. Именно их **сочетание** и является сильной стороной в подходе к прогнозированию.

**Подготовка признаков и целевой переменной**

def prepare\_data\_for\_model(df: pd.DataFrame) -> pd.DataFrame:

Функция prepare\_data\_for\_model готовит данные для машинного обучения.

 # Убираем строки с NaN значениями, которые могут быть из-за индикаторов

    df = df.dropna()

Находит и удаляются строки с пропущенными значениями, которые возникают из-за расчётов индикаторов в начале временного ряда.

# Целевая переменная (например, прогнозируемая цена на следующий день)

    df = df.copy()

    df['Target'] = df['Close'].shift(-1)

Формируется целевая переменная — цена закрытия следующего дня. Это делает задачу прогнозирования **одношаговой** регрессией.

 # Убираем строки с NaN значением в целевой переменной

    df = df.dropna(subset=['Target'])

Удаляется последняя строка, у которой целевая переменная (Target) будет NaN.

 # Признаки

    features = ['Open', 'High', 'Low', 'Close', 'Volume', 'SMA\_13', 'EMA\_10', 'RSI\_14', 'MACD\_12\_26\_9', 'BBL\_20\_2.0', 'BBM\_20\_2.0', 'BBU\_20\_2.0']

    X = df[features]

    y = df['Target']

    return X, y

**Делим признаки, и функция** возвращает матрицу признаков X и вектор целевой переменной y для подачи в модель.

Модули stock\_utils.py и elt.py реализуют все необходимые этапы подготовки данных:

* загрузка исторических данных по тикеру с MOEX;
* очистка и стандартизация таблицы;
* расчёт ключевых технических индикаторов;
* формирование признаков и целевой переменной.

Таким образом, этот блок отвечает за качественную подготовку данных — критически важный этап в процессе построения прогноза цен с помощью машинного обучения.

## 2.7. Прогнозирование с помощью модели XGBoost

После этапа загрузки, преобразования и подготовки данных следующим логическим шагом является обучение модели машинного обучения, способной предсказывать будущую цену акции. В данном проекте для решения задачи регрессии используется алгоритм **XGBoost** — одна из самых популярных реализаций градиентного бустинга над решающими деревьями. Выбор этой модели обусловлен рядом причин, связанных с её характеристиками, преимуществами и практической эффективностью в задачах финансового прогнозирования.

### 1. Эффективность на табличных данных

Технические индикаторы, такие как SMA, EMA, RSI, MACD и Bollinger Bands, формируют **табличные числовые признаки**, а XGBoost показывает выдающуюся производительность именно на таких типах данных. В отличие от нейросетей, которым требуется больше данных и ресурсов, XGBoost хорошо работает даже на сравнительно небольших и средних выборках.

### 2. Поддержка регуляризации

Модель XGBoost включает **регуляризацию** по L1 и L2, что помогает избежать **переобучения** — особенно важного в задачах на временных рядах, где модель может «запомнить» закономерности вместо обобщения.

### 3. Работа с пропущенными значениями и шумом

Финансовые данные часто содержат **шум** и пропущенные значения. XGBoost умеет обрабатывать пропуски и автоматически «обходит» такие значения во время построения деревьев. Это снижает необходимость в жёсткой предобработке.

### 4. Гибкость в настройке и высокая точность

XGBoost предоставляет множество **гиперпараметров**, таких как:

* глубина деревьев (max\_depth),
* скорость обучения (learning\_rate),
* количество деревьев (n\_estimators),
* доля подвыборки (subsample),  
  что позволяет точно настроить модель под конкретные данные. Это достигается в проекте через GridSearchCV.

### 5. Быстрота обучения

Модель реализована на C++ и оптимизирована для высокой скорости обучения и предсказания. Даже при большом числе деревьев и параметров XGBoost обучается быстрее многих аналогов.

### 6. Поддержка временных зависимостей

Хотя XGBoost сам по себе не моделирует временные зависимости (как, например, LSTM), он может успешно использовать отложенные или агрегированные признаки, включая технические индикаторы, которые уже учитывают поведение цены во времени.

### 7. Активное использование в индустрии

XGBoost широко применяется в **финансовой аналитике, трейдинге и инвестиционном моделировании**, а также часто используется на соревнованиях по машинному обучению (например, Kaggle) и показывает стабильно высокие результаты в задачах регрессии и классификации.

Функциональность реализована в модуле predictor.py, который содержит две основные функции: train\_model\_with\_cv и predict\_price.

### Обучение модели с кросс-валидацией

Импортируем необходимые модули и библиотеки

from xgboost import XGBRegressor

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.metrics import mean\_squared\_error

import pandas as pd

import numpy as np

def train\_model\_with\_cv(X: pd.DataFrame, y: pd.Series) -> XGBRegressor:

Функция train\_model\_with\_cv отвечает за обучение модели на исторических данных с использованием **кросс-валидации** и **поиска оптимальных гиперпараметров** с помощью GridSearchCV.

Разделяем данные на выборки

 # Разделяем данные на обучающую и тестовую выборки

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

Данные разделяются на обучающую и тестовую выборки в пропорции 80/20.  
Важно: shuffle=False, чтобы сохранить временную последовательность данных — это критично для временных рядов.

# Инициализация модели

    model = XGBRegressor(objective='reg:squarederror', n\_estimators=100, max\_depth=5)

Инициализируется базовая модель XGBoost для задачи регрессии. Параметры могут быть переопределены при подборе.

Для **поиска наилучших гиперпараметров модели XGBoost** с помощью техники подбора параметров — GridSearchCV определяем сетку гиперпараметров для перебора:

   # Параметры для GridSearchCV

    param\_grid = {

        'max\_depth': [3, 5, 7],

        'learning\_rate': [0.01, 0.05, 0.1],

        'n\_estimators': [100, 200],

        'subsample': [0.8, 1.0],

    }

* max\_depth — максимальная глубина дерева (контролирует переобучение),
* learning\_rate — скорость обучения (чем меньше, тем точнее, но медленнее),
* n\_estimators — количество деревьев (итераций бустинга),
* subsample — доля случайных наблюдений для построения каждого дерева.

# Инициализация GridSearchCV

    grid\_search = GridSearchCV(estimator=model, param\_grid=param\_grid, scoring='neg\_mean\_squared\_error', cv=5)

    # Обучение модели с кросс-валидацией

    grid\_search.fit(X\_train, y\_train)

# Лучшая модель после кросс-валидации

    best\_model = grid\_search.best\_estimator\_

GridSearchCV проводит **поиск лучших параметров** по метрике RMSE (с отрицательным знаком, как требует scoring в sklearn) с **5-кратной кросс-валидацией**.  
После завершения поиска берётся лучшая модель.

# Оценка модели на тестовых данных

    y\_pred = best\_model.predict(X\_test)

    mse = mean\_squared\_error(y\_test, y\_pred)

    rmse = np.sqrt(mse)

    print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')

Оценивается качество модели на тестовой выборке с помощью **RMSE** (среднеквадратичная ошибка), что удобно для интерпретации в единицах цены.

Модель XGBoost с оптимальными параметрами обеспечивает точное и устойчивое прогнозирование цены на основе технических индикаторов. Кросс-валидация минимизирует риск переобучения и повышает обобщающую способность модели.

### Прогнозирование цены

Вторая функция predict\_price используется для получения прогноза на основе уже обученной модели:

def predict\_price(model: XGBRegressor, X: pd.DataFrame) -> float:

 latest\_data = X.iloc[-1].values.reshape(1, -1)  # Используем последние данные для прогноза

    predicted\_price = model.predict(latest\_data)

    return predicted\_price[0]

Из набора признаков берётся **последняя строка** (самые свежие технические индикаторы) и подаётся на вход модели.

Результат — это **прогнозируемая цена на следующий день**. Возвращается скалярное значение — предсказанная цена.

Использование алгоритма XGBoost в задаче регрессии позволяет эффективно моделировать сложные зависимости между техническими индикаторами и будущими значениями цены. Высокая интерпретируемость параметров модели и встроенные методы регуляризации делают XGBoost устойчивым к переобучению. Подход, реализованный в модуле predictor.py, обеспечивает надёжную основу для построения прогностической системы в финансовом анализе.

## 2.8. Визуализация результатов

Для эффективного представления прогноза цен на акции в рамках проекта была реализована функция визуализации в модуле plot\_utils.py. Её основная задача — отобразить исторические данные по ценам и наглядно показать предсказанную моделью цену. Это помогает не только проверить корректность расчётов, но и улучшить восприятие результата конечным пользователем.

Импорт необходимых библиотек

from matplotlib import pyplot as plt

import io

Функция plot\_forecast необходима для **визуализации прогноза цены акции**, сделанного моделью машинного обучения, на фоне исторических данных за последние 60 дней. Её основное назначение — **наглядно показать пользователю результат предсказания**, что особенно важно в контексте работы Telegram-бота или любого другого пользовательского интерфейса.

def plot\_forecast(dates, prices, predicted\_date, predicted\_price):

Функция принимает:

* dates — список с датами исторических цен;
* prices — список с соответствующими ценами закрытия;
* predicted\_date — дата, на которую сделан прогноз;
* predicted\_price — прогнозируемое значение цены.

Фильтрация последних 60 дней

# Преобразуем в Series с datetime index для фильтрации последних 60 дней

    data = pd.Series(prices.values, index=pd.to\_datetime(dates))

    last\_date = data.index.max()

    start\_date = last\_date - pd.Timedelta(days=60)

    recent\_data = data[data.index >= start\_date]

Такой подход позволяет отобразить **только последние 2 месяца** данных. Это решение было принято по двум причинам:

1. Избежать перегруженности графика при большом объёме исторических данных (до 2 лет).
2. Сосредоточиться на **актуальной динамике цены**, что более релевантно для краткосрочного прогноза.

Построение графика

 # Генерируем график только за последние 60 дней

    plt.figure(figsize=(10, 5))

    plt.plot(recent\_data.index, recent\_data.values, label="Историческая цена")

    plt.scatter(predicted\_date, predicted\_price, color='red', label="Прогноз", zorder=5)

    plt.title("Прогноз цены (последние 2 месяца)")

    plt.xlabel("Дата")

    plt.ylabel("Цена")

    plt.legend()

    plt.grid(True)

    plt.tight\_layout()

Синяя линия — историческое движение цены. Красная точка — предсказанная цена, выделенная с помощью plt.scatter(...)

buf = io.BytesIO()

    plt.savefig(buf, format='png')

    buf.seek(0)

    plt.close()

    return buf

Вместо сохранения графика в файл, он сохраняется в байтовый буфер (BytesIO), который:

* легко передать в Telegram-бот;
* не требует сохранения на диск;
* удобен для отправки изображения в ответ пользователю.

Визуализация прогноза в рамках проекта играет важную роль:

* делает результат машинного обучения наглядным;
* позволяет быстро оценить точность прогноза в контексте последних рыночных движений;
* упрощает интеграцию с ботом или веб-интерфейсом.

Таким образом, модуль plot\_utils.py закрывает важную задачу — превращение численного результата в удобную для пользователя визуальную форму.

# Заключение

В рамках практической части дипломного проекта была реализована полностью автономная система прогнозирования цен акций, интегрированная в интерфейс Telegram-бота. Бот предоставляет пользователю простой и интуитивно понятный способ получения прогноза котировок российских акций на следующий торговый день, опираясь на исторические данные и алгоритмы машинного обучения.

Реализация охватывает полный цикл Data Engineering и Data Science:

* **Сбор данных**: автоматизированная загрузка исторических котировок с Московской биржи с помощью библиотеки moexalgo.
* **Обработка данных**: очистка, стандартизация и обогащение исторических данных техническими индикаторами (SMA, EMA, RSI, MACD, Bollinger Bands).
* **Формирование признаков и целевой переменной**: подготовка данных для регрессионной модели.
* **Обучение модели**: использование модели XGBoost с кросс-валидацией и подбором параметров.
* **Прогнозирование и визуализация**: предсказание цены акции на следующий день и отображение результата на графике.
* **Интерфейс взаимодействия**: через Telegram с помощью библиотеки aiogram.

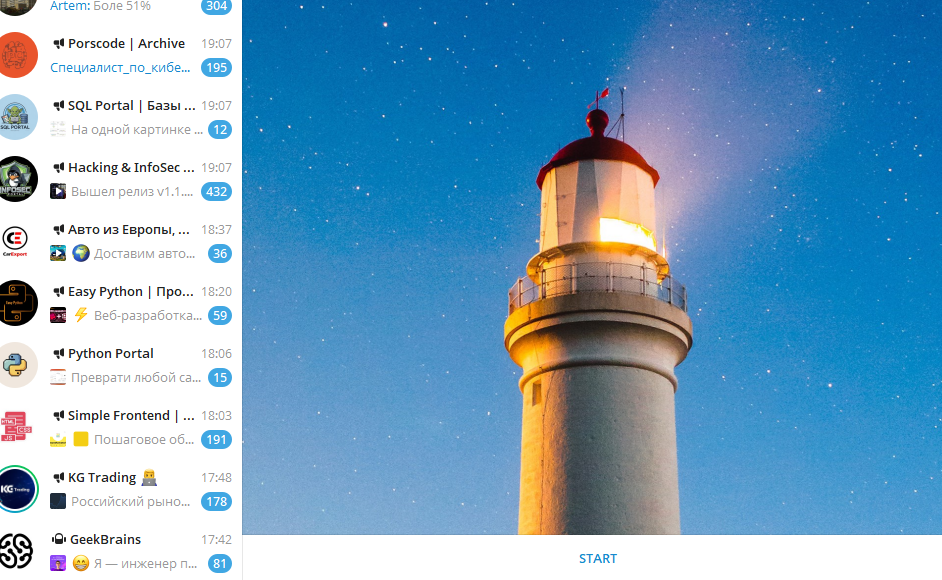
Каждый модуль системы (от stock\_utils.py до predictor.py и plot\_utils.py) выполняет строго определённую функцию, что обеспечивает модульность, читаемость и лёгкость в сопровождении и масштабировании кода. Архитектура позволяет расширить функционал: например, добавить новые алгоритмы прогнозирования, расширить список доступных тикеров, или подключить дополнительные источники данных.

Реализованная система успешно демонстрирует, как современные инструменты Python-разработки, аналитики и машинного обучения могут быть объединены в практическое приложение, решающее прикладную задачу финансового анализа. Проект обладает потенциальной ценностью как для образовательных целей, так и для начального анализа на рынке ценных бумаг.

# Список использованной литературы:

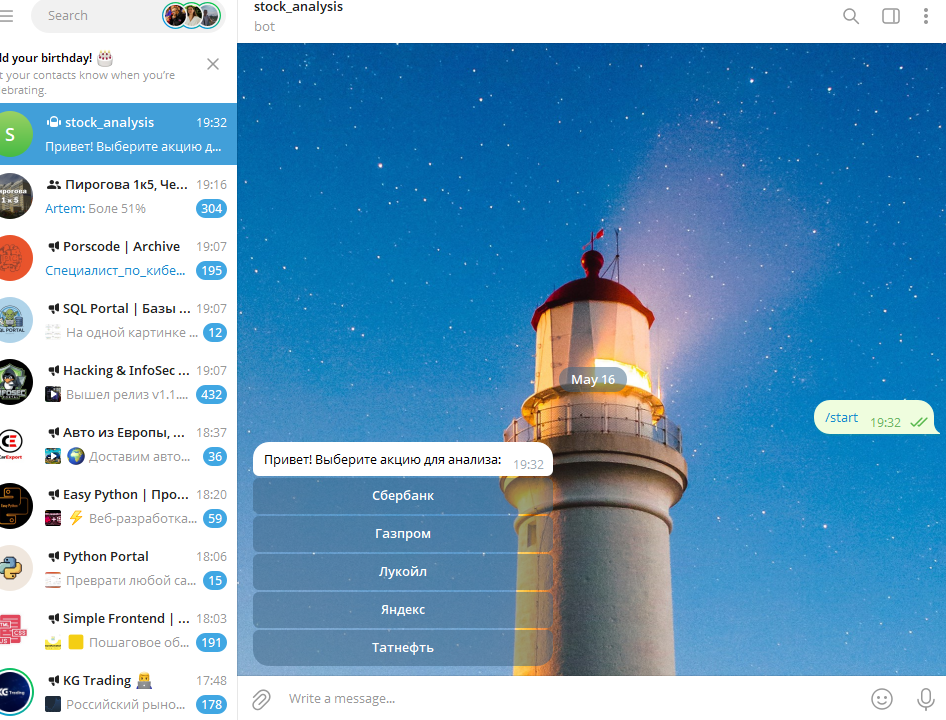
* Герасимов, А. В. Машинное обучение: практический курс на Python. — СПб.: Питер, 2022. — 384 с.
* Чжан Ю., Матиас К. XGBoost: алгоритмы, теория и практика. — Москва: Диалектика, 2021. — 296 с.
* Броунинг М. Программирование Telegram-ботов с использованием Python и Aiogram. — СПб.: БХВ-Петербург, 2022.
* Документация библиотеки aiogram — https://docs.aiogram.dev
* Документация библиотеки xgboost — <https://xgboost.readthedocs.io>
* Документация библиотеки pandas — https://pandas.pydata.org
* Документация библиотеки pandas-ta (технические индикаторы) — <https://github.com/twopirllc/pandas-ta>
* Документация matplotlib — https://matplotlib.org/stable/contents.html
* MOEX Algo SDK (библиотека moexalgo) — <https://github.com/Lu-Yi-Hsun/moexalgo>
* Официальная документация Telegram Bot API — https://core.telegram.org/bots/api

# *Приложение 1*

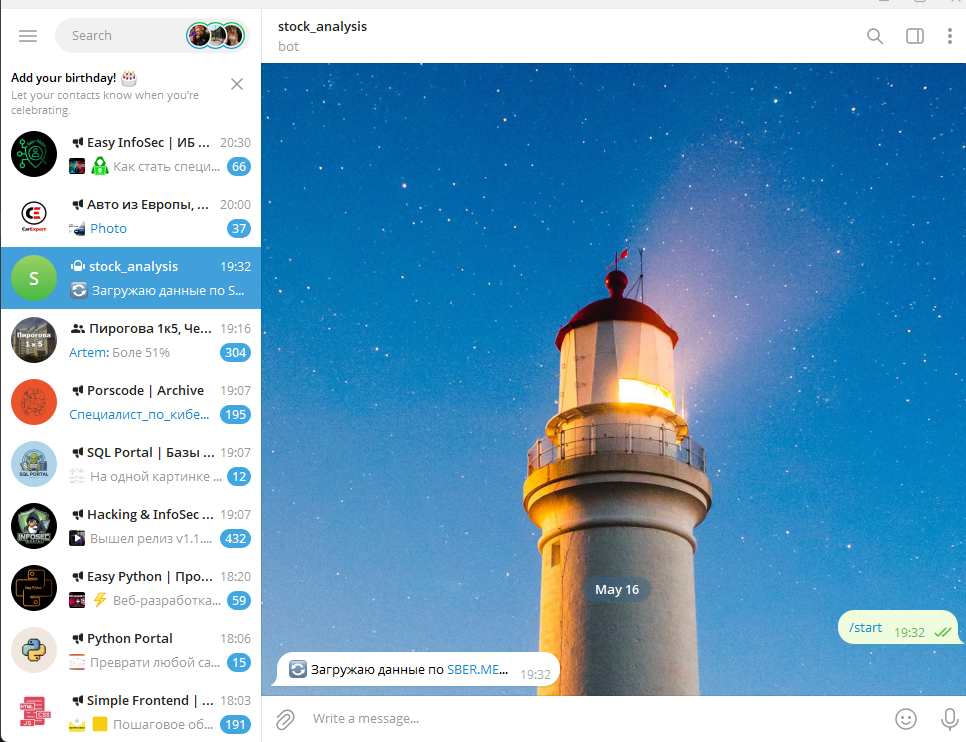


# 

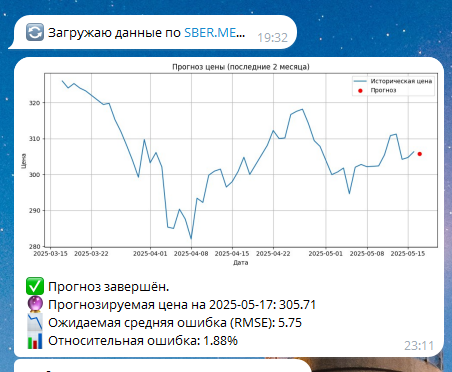
# *Приложение 2*



# *Приложение 3*



# *Приложение 4*



# *Приложение 5*

