

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___» _____ 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Інженерія програмного
забезпечення мультимедійних та інформаційно-пошукових систем»**

спеціальності 121 Інженерія програмного забезпечення

**на тему: «Вебзастосунок для агрегації та очищення даних біткоїна для
ринкового прогнозування»**

Виконав:

студент IV курсу, групи КП-13
Фадєєв Олексій Олександрович

Керівник:

доцент кафедри ПЗКС, к.т.н., доцент,
Юрчишин Василь Якович

Консультант з нормоконтролю:

доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович

Рецензент:

старший викладач кафедри СПСКС, к.т.н.,
Наливайчук Микола Васильович

Засвідчую, що у цьому дипломному
проєкті немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення
мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___» _____ 2024 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Фадєєву Олексію Олександровичу

1. Тема проєкту «Вебзастосунок для агрегації та очищення даних біткоїна для ринкового прогнозування», керівник проєкту Юрчишин Василь Якович, к.т.н., доцент кафедри ПЗКС, затверджені наказом по університету № 1808-С від «29» травня 2025 р.
2. Термін подання студентом проєкту «13» червня 2025 р.
3. Вихідні дані до проєкту: див. Технічне завдання.
4. Зміст пояснювальної записки:
 - огляд та аналіз існуючих рішень;
 - обґрунтування вибору засобів реалізації;
 - розроблення вебсервісу;
 - аналіз розробленого вебсервісу.
5. Перелік обов'язкового графічного матеріалу:
 - структура бази даних вебзастосунку (креслення);
 - структура класів серверної частини (креслення);

- схема модулів системи вебзастосунку (плакат);
- схема архітектури автентифікації Flask-login (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2024 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	15.11.2024	
2.	Розроблення та узгодження технічного завдання	26.11.2024	
3.	Розроблення структури вебзастосунку	16.12.2024	
4.	Підготовка матеріалів першого розділу дипломного проєкту	17.01.2024	
5.	Розроблення дизайну сторінок та графічних елементів вебзастосунку	12.02.2025	
6.	Підготовка матеріалів другого розділу дипломного проєкту	26.02.2025	
7.	Програмна реалізація вебзастосунку	01.03.2025	
8.	Тестування вебзастосунку	15.04.2025	
9.	Підготовка матеріалів третього розділу дипломного проєкту	28.04.2025	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	15.05.2025	
11.	Підготовка графічної частини дипломного проєкту	22.05.2025	
12.	Оформлення документації дипломного проєкту	23.05.2025	

Студент

Олексій ФАДЄЄВ

Керівник проєкту

Василь ЮРЧИШИН

АНОТАЦІЯ

Дипломний проєкт присвячено розробленню вебзастосунку, призначеного для збору, комплексної обробки та підготовки фінансових даних, зокрема даних криптовалютного ринку (біткоїн), для подальшого використання в задачах машинного навчання. Метою проєкту є надання користувачам гнучкого та потужного інструменту для формування якісних багатофакторних наборів даних.

У рамках проєкту було спроектовано та реалізовано архітектуру вебсервісу, що складається з серверної та клієнтської частин. Серверна частина, розроблена на Python з використанням фреймворку Flask, відповідає за взаємодію із зовнішніми API для збору даних, їх агрегацію, валідацію, очищення, трансформацію (включаючи нормалізацію, кодування, обробку аномалій та пропусків) та збереження результатів. Клієнтська частина надає користувацький вебінтерфейс для конфігурації параметрів збору та обробки даних, моніторингу процесу та завантаження фінальних датасетів і звітів.

Для зберігання сирих та оброблених даних, а також конфігурацій користувача, розроблено структуру бази даних (PostgreSQL), з якою взаємодія відбувається через SQLAlchemy ORM. Це забезпечує надійне зберігання інформації та швидкий доступ до неї.

Ключовою особливістю розробленого вебзастосунку є його здатність створювати кастомізовані багатофакторні набори даних. Користувачі можуть гнучко обирати та комбінувати дані з різних джерел: ринкові дані (ціни, обсяги) з провідних криптовалютних бірж (Binance, Coinbase, Kraken), ончейн-метрики (наприклад, хешрейт, кількість транзакцій), макроекономічні показники та власноруч визначені фактори. Застосунок автоматизує складні етапи попередньої обробки, такі як фільтрація аномалій, інтерполяція пропущених значень, нормалізація числових ознак та кодування категоріальних змінних.

Ще однією важливою функцією є генерація цільових змінних, необхідних для навчання моделей машинного навчання (регресії та класифікації), що значно спрощує підготовку даних для прогнозних завдань. Система також формує детальні звіти про якість зібраних даних та етапи їх обробки, забезпечуючи прозорість процесу.

Цей проєкт має великий потенціал для спрощення роботи аналітиків, дослідників і розробників моделей машинного навчання, пропонуючи ефективний інструмент для підготовки якісних даних, що є ключовим для успішного аналізу та прогнозування майбутніх тенденцій на ринку криптовалют.

ABSTRACT

The diploma project is dedicated to the development of a web application designed for the collection, comprehensive processing, and preparation of financial data, particularly cryptocurrency market data (Bitcoin), for further use in machine learning tasks. The aim of the project is to provide users with a flexible and powerful tool for creating high-quality multifactor datasets.

Within the framework of the project, the architecture of the web service, consisting of server-side and client-side components, was designed and implemented. The server-side, developed in Python using the Flask framework, handles interaction with external APIs for data collection, aggregation, validation, cleansing, transformation (including normalization, encoding, anomaly and missing value handling), and result storage. The client-side provides a user-friendly web interface for configuring data collection and processing parameters, monitoring the process, and downloading final datasets and reports.

For storing raw and processed data, as well as user configurations, a database structure (PostgreSQL) was developed, with interaction facilitated through the SQLAlchemy ORM. This ensures reliable data storage and fast access.

A key feature of the developed web application is its ability to create customized multifactor datasets. Users can flexibly select and combine data from various sources: market data (prices, volumes) from leading cryptocurrency exchanges (Binance, Coinbase, Kraken), on-chain metrics (e.g., hash rate, transaction volume), macroeconomic indicators, and user-defined factors. The application automates complex preprocessing stages, such as anomaly filtering, interpolation of missing values, normalization of numerical features, and encoding of categorical variables.

Another important function is the generation of target variables essential for training machine learning models (regression and classification), significantly simplifying data preparation for forecasting tasks. The system also

generates detailed reports on the quality of collected data and the stages of its processing, ensuring transparency.

This project offers substantial potential to streamline the efforts of analysts, researchers, and machine learning model developers by delivering an effective tool for preparing high-quality data, which is essential for successful analysis and forecasting of upcoming trends in the cryptocurrency market.

ДП.045440-01-90 Вебзастосунок для агрегації та очищення даних біткоїна для
ринкового прогнозування. Відомість проєкту

Позначення	Найменування	Кількість	Примітка
	Документація проєкту		
ДП.045440-02-91	Вебзастосунок для агрегації та очищення даних біткоїна для ринкового прогнозування. Технічне завдання	5	
ДП.045440-03-81	Вебзастосунок для агрегації та очищення даних біткоїна для ринкового прогнозування. Пояснювальна записка	80	
ДП.045440-04-51	Вебзастосунок для агрегації та очищення даних біткоїна для ринкового прогнозування. Програма та методика тестування	4	
ДП.045440-05-34	Вебзастосунок для агрегації та очищення даних біткоїна для ринкового прогнозування. Керівництво користувача	16	
ДП.045440-06-99	Вебзастосунок для агрегації та очищення даних біткоїна для ринкового прогнозування. Структура класів серверної частини. Діаграма компонентів	1	

[illegible]

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“ ____ ” _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ АГРЕГАЦІЇ ТА ОЧИЩЕННЯ ДАНИХ
БІТКОЇНА ДЛЯ РИНКОВОГО ПРОГНОЗУВАННЯ

Технічне завдання

ДП.045440-02-91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Василь ЮРЧИШИН

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Олексій ФАДЄЄВ

2024

ЗМІСТ

1.Найменування та галузь застосування	3
2.Підстава для розроблення.....	3
3.Призначення розробки	3
4.Вимоги до програмного продукту	3
5.Вимоги до проєктної документації.....	5
6.Етапи проєктування.....	5
7.Порядок тестування розробки.....	5

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: вебзастосунок для агрегації та очищення даних біткоїна для ринкового прогнозування.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання аналітиками, дослідниками та розробниками машинного навчання з метою автоматизації процесу збору, агрегації, очищення та попередньої обробки ринкових даних біткоїна. Кінцевою метою є надання користувачам якісних, структурованих наборів даних, готових для використання в моделях ринкового прогнозування та аналітичних дослідженнях.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Вебзастосунок повинен забезпечувати такі основні функції:

- 1)автоматичний щоденний збір даних із джерел: Binance (BTC/USDT), Coinbase (BTC/USD), Kraken (XBT/USD), Blockchain.com (блокчейн-метрики), FRED (макроекономічні показники), Alpha Vantage (технічні індикатори);
- 2)обробка помилок підключення до API з механізмами повторних спроб;

- 3)очищення та валідація даних: видалення дублікатів, заповнення пропусків (інтерполяція), перевірка узгодженості між біржами, корекція аномалій цін та об'ємів (Z-score, IQR);
- 4)розрахунок похідних факторів: спреда цін та об'ємів між біржами, середні ціни, волатильність (3, 7, 14 днів), кореляції з іншими активами;
- 5)вибір та генерація цільової змінної для регресії/класифікації з урахуванням часового горизонту;
- 6)нормалізація/стандартизація (MinMax, Standard, Robust scaler), трансформація (Yeo-Johnson), кодування категоріальних ознак (Label Encoding, One-Hot Encoding);
- 7)вінзоризація даних, балансування класів для класифікації (SMOTE, RandomUnderSampler);
- 8)генерація звіту якості отриманих даних;
- 9)авторизація та реєстрація.

Розробку виконати на основі: Python (серверна логіка з фреймворком Flask, бібліотеки SQLAlchemy для БД), JavaScript (Vanilla JS для клієнтської частини), бібліотеки Pandas, NumPy, Scikit-learn (обробка та аналіз даних), PostgreSQL (система управління базами даних).

Додаткові вимоги:

- 1)система має бути стійкою до збоїв API зовнішніх джерел;
- 2)передбачено механізми логування для діагностики проблем;
- 3)архітектура має підтримувати інтеграцію нових джерел даних та факторів із мінімальними змінами;
- 4)виконання запитів на збір та обробку даних до 3 тис. записів має виконуватися менше ніж за 3 секунди.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

- 1) пояснювальна записка;
- 2) програма та методика тестування;
- 3) керівництво користувача;
- 4) креслення:
 - «Схема бази даних вебзастосунку. ER-діаграма»;
 - «Діаграма класів серверної частини. Діаграма компонентів».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи.....	17.11.2024
Розроблення та узгодження технічного завдання.....	29.11.2024
Розроблення структури вебзастосунку.....	15.12.2024
Розроблення дизайну сторінок та графічних елементів	12.02.2025
Програмна реалізація вебзастосунку.....	12.03.2025
Тестування вебзастосунку	10.04.2025
Підготовка матеріалів текстової частини проєкту.....	27.04.2025
Підготовка матеріалів графічної частини проєкту	23.05.2025
Оформлення технічної документації проєкту	24.05.2025

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“ ” _____ 2025 р.

**ВЕБЗАСТОСУНОК ДЛЯ АГРЕГАЦІЇ ТА ОЧИЩЕННЯ ДАНИХ
БІТКОЇНА ДЛЯ РИНКОВОГО ПРОГНОЗУВАННЯ**

1

записка

ДП.045440-03-81

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Василь ЮРЧИШИН

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Олексій ФАДЄЄВ

2025

3MICT

The table of contents is empty because you haven't selected paragraph styles to appear in it.

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

API (Application Programming Interface) – інтерфейс прикладного програмування.

BTC – тикер біткоїна.

Celery – розподілена черга завдань для Python.

CSS (Cascading Style Sheets) – каскадні таблиці стилів.

CSV (Comma-Separated Values) – формат текстових файлів, де значення розділені комами.

Flask – мікрофреймворк для розробки вебзастосунків на Python.

HTML (HyperText Markup Language) – мова розмітки гіпертексту.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту.

IQR (Interquartile Range) – міжквартильний розмах.

JavaScript (JS) – мова програмування, що використовується для створення інтерактивних вебсторінок.

Jinja2 – шаблонізатор для Python.

JSON (JavaScript Object Notation) – текстовий формат обміну даними.

ML (Machine Learning) – машинне навчання.

MLBit – назва розроблюваного вебзастосунку.

MTV (Model-Template-View) – архітектурний шаблон, варіація MVC.

NumPy – бібліотека Python для наукових обчислень, зокрема роботи з масивами.

OHLCV (Open, High, Low, Close, Volume) – ціни відкриття, максимуму, мінімуму, закриття, обсяг.

ORM (Object-Relational Mapper) – технологія програмування, яка пов'язує бази даних з концепціями об'єктно-орієнтованих мов програмування.

Pandas – бібліотека Python для аналізу та маніпулювання даними.

PostgreSQL – об'єктно-реляційна система управління базами даних.

Python – високорівнева мова програмування загального призначення.

RSI (Relative Strength Index) – індекс відносної сили (технічний індикатор), застосовується для оцінки перекупленості чи перепроданості біткоїна на основі змін ціни.

Render – хмарна платформа розгортання вебзастосунків, бекендів, статичних сайтів та сервісів.

SciPy – бібліотека Python для наукових та інженерних обчислень.

Scikit-learn – бібліотека Python для машинного навчання.

SMOTE (Synthetic Minority Over-sampling Technique) – техніка синтетичного збільшення вибірки для класу меншості (використовується в *imbalanced-learn*).

SQL (Structured Query Language) – мова структурованих запитів.

SQLAlchemy – SQL-інструментарій та ORM для Python.

UI (User Interface) – інтерфейс користувача.

UX (User Experience) – досвід користувача.

Z-score – статистична міра, що показує, на скільки стандартних відхилень значення відрізняється від середнього.

Агрегація даних – процес збору та об'єднання даних з різних джерел в єдиний набір.

Аномалія (викид) – значення в наборі даних, яке суттєво відрізняється від інших значень і може вказувати на помилку вимірювання або нетипову подію.

Біткоїн (Bitcoin, BTC) – децентралізована цифрова валюта (криптовалюта), створена на основі технології блокчейн.

Блокчейн (Blockchain) – розподілена база даних, що складається з ланцюжка блоків, кожен з яких містить інформацію про транзакції та криптографічно пов'язаний з попереднім блоком.

Валідація даних – процес перевірки даних на відповідність певним критеріям якості, точності та повноти.

Вебзастосунок – клієнт-серверний програмний продукт, в якому клієнтом виступає веббраузер, а сервером – вебсервер.

Дані OHLCV – ринкові дані, що включають ціну відкриття (open), найвищу ціну (high), найнижчу ціну (low), ціну закриття (close) та обсяг торгів (volume) за певний період.

Datamset (Dataset) – структурований набір даних, підготовлений для аналізу або використання в моделях машинного навчання.

Інтерполяція – метод знаходження проміжних значень функції за відомим набором її дискретних значень; використовується для заповнення пропусків у даних.

Кодування (Encoding) – процес перетворення категоріальних даних у числовий формат, придатний для обробки алгоритмами машинного навчання.

Криптовалюта – цифрова або віртуальна валюта, захищена криптографією, що ускладнює її підробку.

Макроекономічні показники – індикатори, що відображають стан економіки країни або регіону в цілому (наприклад фондові індекси).

Машинне навчання (Machine Learning, ML) – галузь штучного інтелекту, що вивчає методи побудови алгоритмів, здатних навчатися на основі даних.

Нормалізація даних – процес масштабування числових ознак до певного діапазону (наприклад, $[0, 1]$ або $[-1, 1]$) для покращення роботи алгоритмів машинного навчання.

Ончейн-метрики (On-chain Metrics) – показники, що отримуються безпосередньо з даних блокчейну (наприклад, хешрейт, кількість транзакцій, активні адреси).

Очищення даних – процес виявлення та виправлення (або видалення) помилок, неточностей, пропусків та аномалій у наборі даних.

Пайплайн (Pipeline) – послідовність кроків обробки даних, які виконуються автоматично.

Прогноз ціни (Price Forecasting) – задача передбачення майбутньої ціни активу на основі історичних даних та інших факторів.

Ринкове прогнозування – процес аналізу історичних даних та поточної ринкової ситуації для передбачення майбутніх рухів ринку.

Стандартизація даних – процес перетворення числових ознак таким чином, щоб вони мали середнє значення 0 та стандартне відхилення 1.

Технічні індикатори – математично розраховані показники на основі цін та/або обсягів торгів, що використовуються для аналізу ринкових тенденцій.

Трансформація даних – процес зміни структури або значень даних для їх кращого представлення або підготовки до аналізу.

Хешрейт (Hashrate) – загальна обчислювальна потужність, що використовується для майнінгу та обробки транзакцій у мережі блокчейну (наприклад, Bitcoin).

Цільова змінна (Target Variable) – ознака в наборі даних, значення якої необхідно спрогнозувати за допомогою моделі машинного навчання.

API – абстрактний програмний інтерфейс, набір правил і інструментів, що дозволяє взаємодію між різними програмними компонентами.

SMA (Simple Moving Average) – проста ковзна середня, використовується для визначення тренду ціни біткойна шляхом усереднення за обраний період.

ATR (Average True Range) – середній справжній діапазон, застосовується для вимірювання волатильності біткойна за обраний період.

ADX (Average Directional Index) – індекс напрямного руху, використовується для оцінки сили ринкового тренду біткойна незалежно від його напрямку.

ВСТУП

У сучасному світі криптовалюти, зокрема біткоїн, займають важливе місце на фінансових ринках, викликаючи інтерес аналітиків, інвесторів і розробників машинного навчання. Висока волатильність цін, зростання обсягів торгівлі та складність ринкової динаміки створюють потребу в якісних, структурованих даних для аналізу та прогнозування. Проте підготовка таких даних ускладнена через розбіжності між біржами, аномалії, пропуски, а також необхідність інтеграції різноманітних факторів, включаючи макроекономічні показники, мережеві метрики біткоїна та ринкові характеристики.

Розробка вебзастосунку для автоматизованого збору, очищення та підготовки даних біткоїна є актуальним завданням, оскільки забезпечує аналітиків і дослідників надійними наборами даних, готовими до використання в прогнозних моделях і ринкових дослідженнях. Такі інструменти оптимізують процеси агрегації, валідації та обробки даних, підвищуючи їхню якість і придатність для машинного навчання.

Метою дипломного проєкту є розробка вебзастосунку для збору, агрегації та очищення багатфакторних даних біткоїна з трьох бірж (Binance, Coinbase, Kraken) за період до одного року з можливістю вибору ключових факторів. Застосунок забезпечує зручний доступ до збору, обробки та підготовки даних для аналітичних і прогнозних задач: очищення від аномалій і пропусків, нормалізація числових показників, кодування категоріальних змінних, створення цільових змінних для регресії чи класифікації, а також експорт структурованих наборів у CSV і JSON із прозорим звітом про якість даних.

Розроблений вебзастосунок сприятиме підвищенню ефективності підготовки даних, надаючи аналітикам, розробникам ML і дослідникам достовірні набори для аналізу крипторинку та прогнозування. Він зменшує вплив аномалій і пропусків, автоматизує складні процеси обробки та забезпечує прозорість завдяки звітуванню про якість даних. Готовність даних до ML-пайплайнів, зокрема через нормалізацію, кодування та

цільові змінні, робить застосунок цінним інструментом для створення точних прогнозних моделей.

Використання сучасних технологій, таких як Python, Flask, Pandas, Scikit-learn і публічні API бірж, забезпечує створення зручного, надійного та ефективного інструменту. Цей проєкт має потенціал стати важливим ресурсом для дослідників і практиків у сфері криптовалют, сприяючи розвитку аналітичних і прогнозних підходів до ринку біткоїна.

1.1. CoinGecko

CoinGecko – одна з провідних платформ для агрегації даних про криптовалюту, яка надає детальну ринкову інформацію, включаючи ціни, обсяги торгів, ринкову капіталізацію та історичні дані з численних бірж, таких як Binance, Coinbase і Kraken. Платформа популярна серед трейдерів, аналітиків і розробників завдяки своїй універсальності та доступності [1].

Основні функції CoinGecko включають:

- 1) відстеження цін, обсягів торгів і змін на ринку для сотень криптовалют;
- 2) інформація про ончейнові показники (наприклад, активність мережі), соціальні метрики та активність розробників;
- 3) інструменти для порівняння різних криптовалют за ціною, обсягом чи ринковою капіталізацією;
- 4) надання API для розробників, що дозволяє інтегрувати дані про ціни, обсяги торгів, ринкову капіталізацію та ончейнові метрики у власні застосунки чи платформи, забезпечуючи доступ до інформації про понад 17 тисяч криптовалют.

Функція вибору даних дозволяє користувачам налаштувати період і біржу для отримання історичних даних (рис. 1.1). Наприклад, користувач може обрати біткоїн, вказати Binance як джерело і завантажити дані про ціну закриття за рік [2]. Проте платформа не підтримує вибір додаткових факторів, таких як хешрейт, макроекономічні показники (S&P 500, ціна золота), блокчейн-метрики чи волатильність, що обмежує її можливості для створення багатофакторних наборів даних.

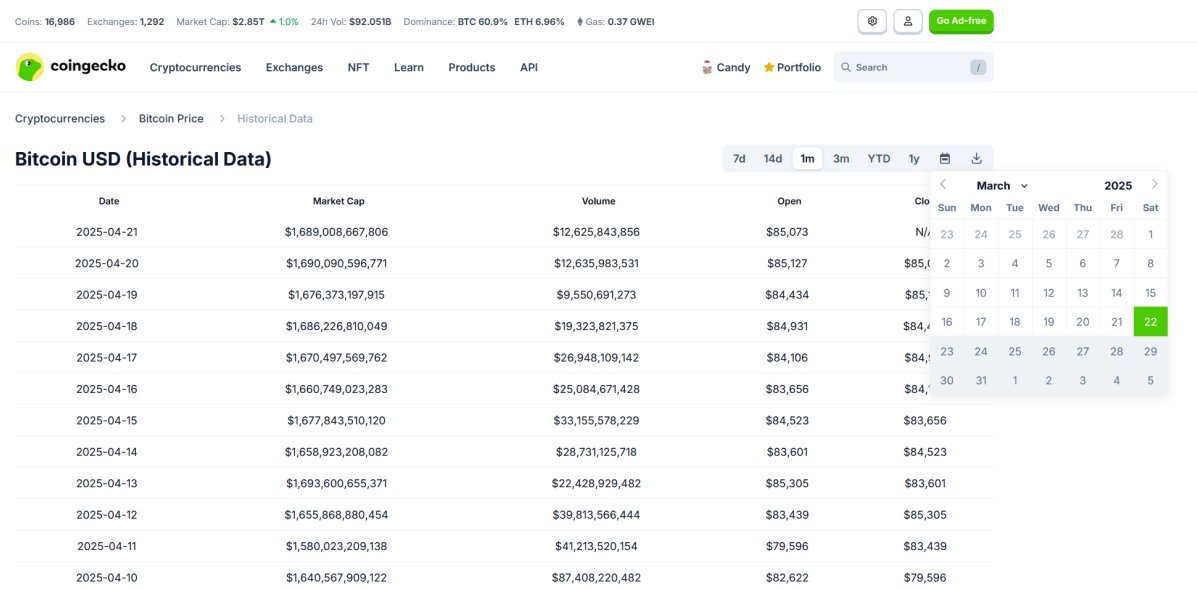


Рис. 1.1. Інтерфейс вибору історичних даних для біткоїна на CoinGecko

CoinGecko підтримує інтеграцію через публічний API, який дозволяє отримувати дані без ключів, хоча з обмеженнями на кількість запитів (приблизно 15 на хвилину). API забезпечує доступ до цін, обсягів і ринкової капіталізації, що спрощує автоматизацію збору даних за допомогою бібліотек, таких як `ccxt` або `requests` у Python (рис. 1.2). Інтеграція з аналітичними інструментами, такими як Excel або Jupyter Notebook, дозволяє користувачам обробляти дані, але відсутність вбудованих функцій для крос-валідації чи фільтрації аномалій вимагає додаткових зусиль від користувача, оскільки аналіз розбіжностей між біржами чи видалення аномальних значень (наприклад, за допомогою Z-score чи IQR) доводиться реалізовувати вручну через зовнішні бібліотеки, такі як `pandas` або `scipy`. Крім того, API не підтримує автоматичне виявлення пропусків чи їх інтерполяцію, що ускладнює роботу з історичними даними, особливо якщо вони мають прогалини, характерні для ранніх періодів діяльності криптовалют.

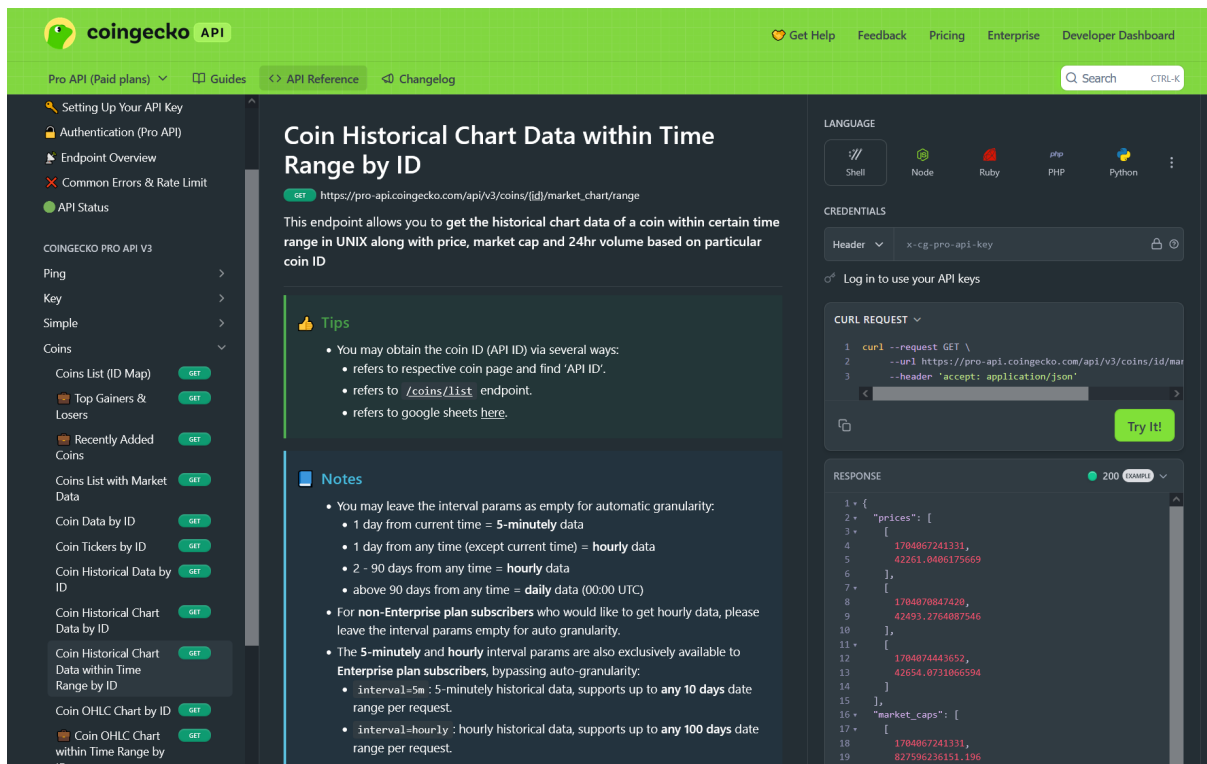


Рис. 1.2. Сторінка документації API CoinGecko, що показує приклади запитів для отримання ринкових даних

Інтерфейс CoinGecko є інтуїтивно зрозумілим і орієнтованим на досвідченого користувача. Головна панель (рис. 1.3) відображає огляд ринку: поточні ціни, обсяги торгів, ринкову капіталізацію, а також графіки для вибраних криптовалют. Дані оновлюються в реальному часі, що дозволяє швидко оцінити ринкову ситуацію. Проте велика кількість інформації може бути складною для новачків, оскільки інтерфейс перевантажений даними, а пояснення до метрик, таких як ончейнові показники чи активність розробників, недостатньо деталізовані для початківців. Також відсутність інструментів для автоматичної обробки пропусків чи виявлення аномалій знижує ефективність підготовки даних для прогнозних моделей, що є критичним для задач машинного навчання. Крім того відсутність вбудованих інструментів для технічного аналізу, таких як розрахунок SMA чи RSI безпосередньо на платформі, обмежує аналітичні можливості для просунутих користувачів.

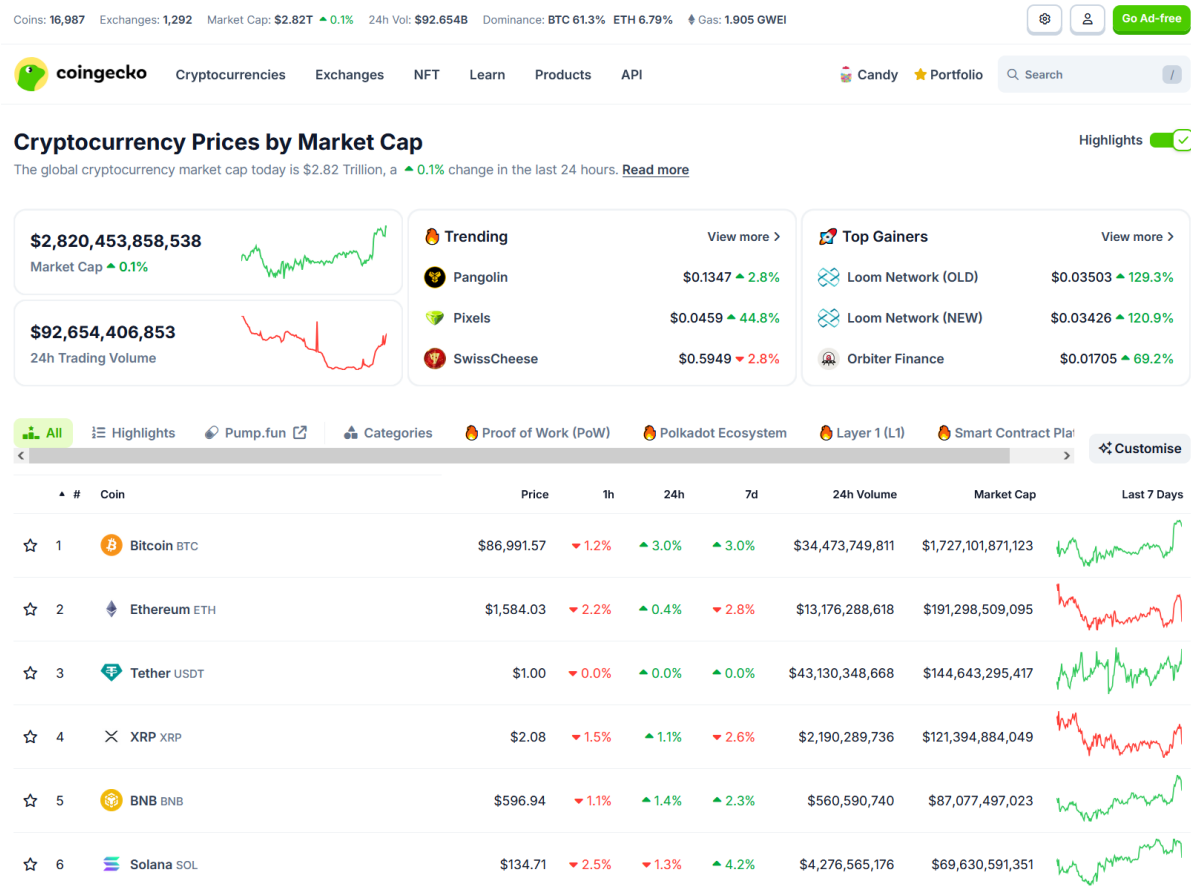


Рис. 1.3. Головна панель CoinGecko з ринковими даними та графіками

CoinGecko надає базові інструменти звітності, такі як інтерактивні графіки та таблиці з історичними даними (рис. 1.4). Користувачі можуть візуалізувати ціни та обсяги за вибраний період, але платформа не генерує детальних звітів про якість даних, таких як відсоток пропусків, кількість аномалій чи середня різниця цін між біржами. Для таких аналізів користувачам доводиться використовувати зовнішні інструменти, наприклад, Python з бібліотеками pandas і numpy. Крім того, CoinGecko не пропонує автоматичних рекомендацій щодо очищення даних чи їхньої нормалізації, що змушує користувачів самотійно розробляти алгоритми для трансформацій, таких як вінзоризація чи застосування MinMaxScaler.

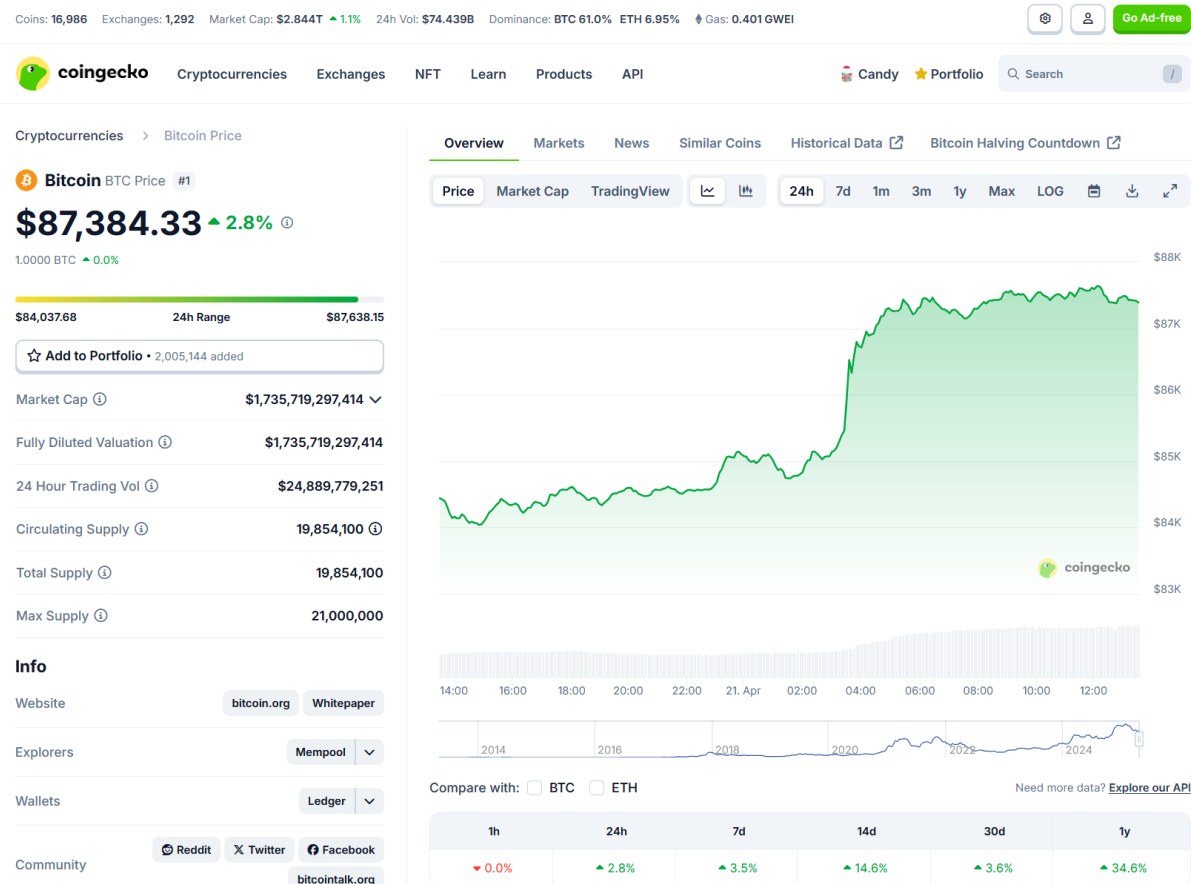


Рис. 1.4. Графік цін біткоіна на CoinGecko за вибраний період із
можливістю експорту

CoinGecko пропонує безкоштовний доступ до базових ринкових даних і API, що робить платформу доступною для широкої аудиторії, включаючи трейдерів, аналітиків і розробників. Вона забезпечує широке покриття бірж і криптовалют, дозволяючи користувачам отримувати дані з таких платформ, як Binance, Coinbase і Kraken. Зручний інтерфейс із підтримкою інтерактивних графіків полегшує візуалізацію цін і обсягів торгів, а регулярне оновлення даних і додавання нових функцій забезпечують актуальність інформації, що є важливим для ринкового аналізу.

Незважаючи на переваги, CoinGecko має обмеження, які впливають на її функціональність. Обмеження API на кількість запитів (від 15 до 20 на хвилину) ускладнює швидкий збір великих обсягів даних, що може бути проблемою для масштабних проєктів. Відсутність вбудованих функцій для фільтрації аномалій чи обробки пропусків змушує користувачів

покладатися на зовнішні інструменти для підготовки даних. Платформа не підтримує вибір додаткових факторів, таких як хешрейт, S&P 500 чи волатильність, що обмежує її для багатфакторного аналізу. Крім того, інтерфейс може бути складним для новачків через велику кількість інформації, що вимагає часу на освоєння.

1.2. CryptoCompare

CryptoCompare – це спеціалізована платформа, яка фокусується на наданні ринкових даних про криптовалюту та їх порівняльному аналізі [3]. Вона агрегує інформацію з провідних бірж, таких як Binance, Coinbase і Kraken, і орієнтована на користувачів, які шукають точні дані для торгівлі та досліджень.

Основні функції CryptoCompare включають:

- 1) відстеження цін, обсягів торгів і змін на ринку для сотень криптовалют;
- 2) доступ до OHLCV-даних за різні періоди – від кількох годин до кількох років;
- 3) аналіз різниці цін і обсягів між платформами, наприклад, між Binance і Kraken;
- 4) експорт ринкової інформації у CSV через вебінтерфейс або API;
- 5) інтеграція новинного контенту для оцінки ринкових настроїв;
- 6) інструмент для моніторингу особистих криптоактивів.

Інтерфейс вибору історичних даних (рис. 1.5), надає користувачу можливість вибрати біржу, наприклад Coinbase, і період для аналізу цін біткоїна [4]. Однак платформа не дозволяє додавати зовнішні фактори, такі як індекси (S&P 500) чи ончейнові метрики (хешрейт), що обмежує її як джерело даних для глибокого аналізу.

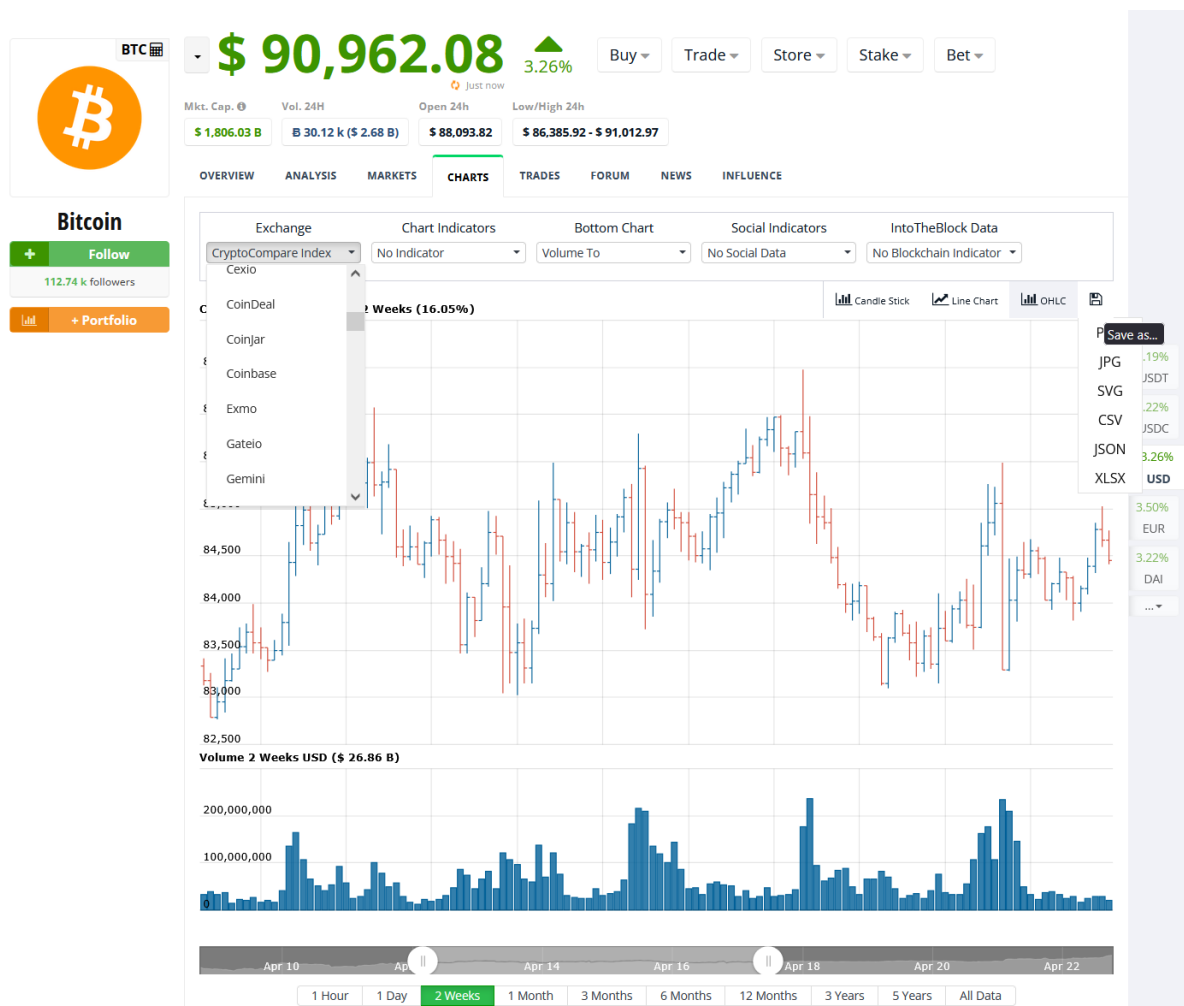


Рис. 1.5. Інтерфейс CryptoCompare для вибору даних із налаштуванням біржі та часового діапазону

CryptoCompare пропонує API без необхідності ключів, що забезпечує базовий доступ до ринкових даних (рис. 1.6). Це дозволяє інтегрувати платформу з мовами програмування, такими як Python чи JavaScript, для автоматизованого збору даних. Проте для складних операцій, таких як крос-валідність чи очищення даних, потрібні додаткові інструменти, наприклад, бібліотеки pandas. Для інтеграції з аналітичними платформами, такими як Jupyter Notebook, API CryptoCompare є ефективним, але потребує додаткових зусиль для реалізації повноцінного ETL-процесу, включаючи нормалізацію та трансформацію даних. Додатково слід враховувати що доступ до API CryptoCompare надається з лімітом 50 запитів на годину.

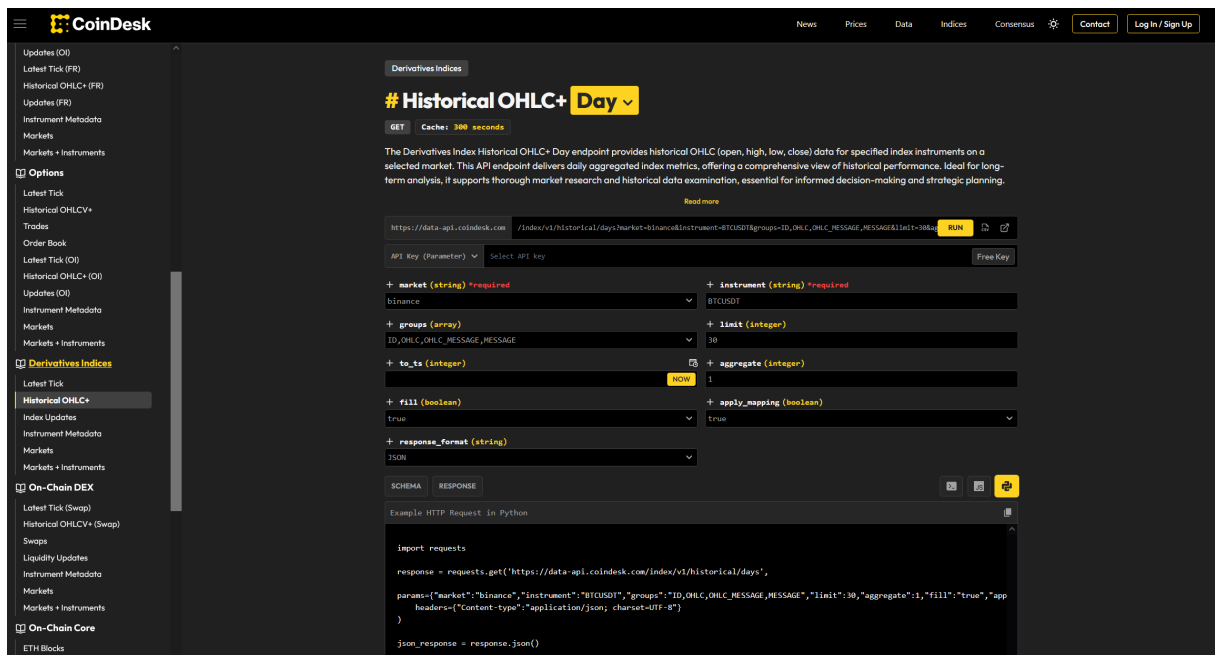


Рис. 1.6. Документація API CryptoCompare (CoinDesk) з прикладами запитів

Дизайн CryptoCompare є практичним і зосередженим на ключовій інформації. На головній сторінці (рис. 1.7) відображаються ціни, графіки та основні ринкові показники в реальному часі. Інтерфейс зручний для швидкого доступу до даних, але через нагромадження функцій може здаватися перевантаженим для тих, хто тільки починає працювати з криптовалютами.

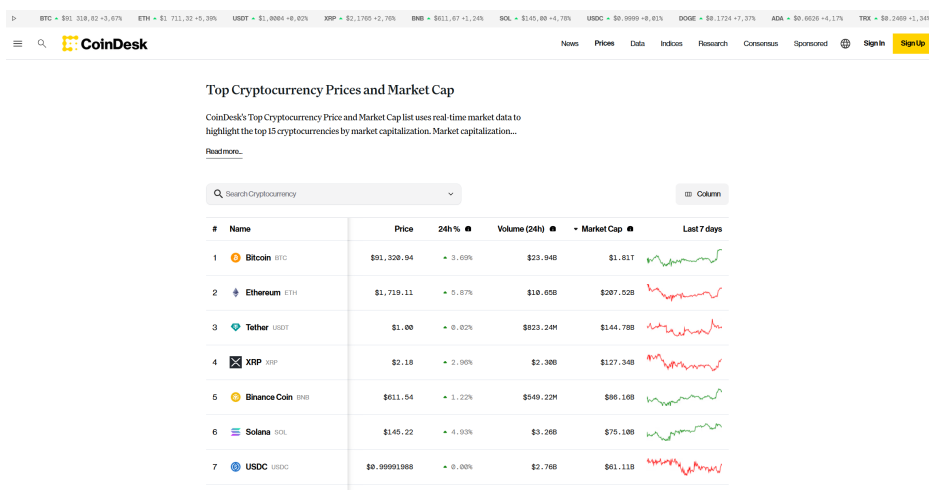


Рис. 1.7. Головна панель CryptoCompare з ринковими даними та графіками

Платформа підтримує базову візуалізацію даних через графіки та

таблиці (рис. 1.8), що дозволяє аналізувати ціни й обсяги за вибраний період. Однак деталізовані звіти, наприклад, про пропуски в даних чи аномальні стрибки цін, відсутні, і користувачам доводиться покладатися на зовнішні засоби обробки.

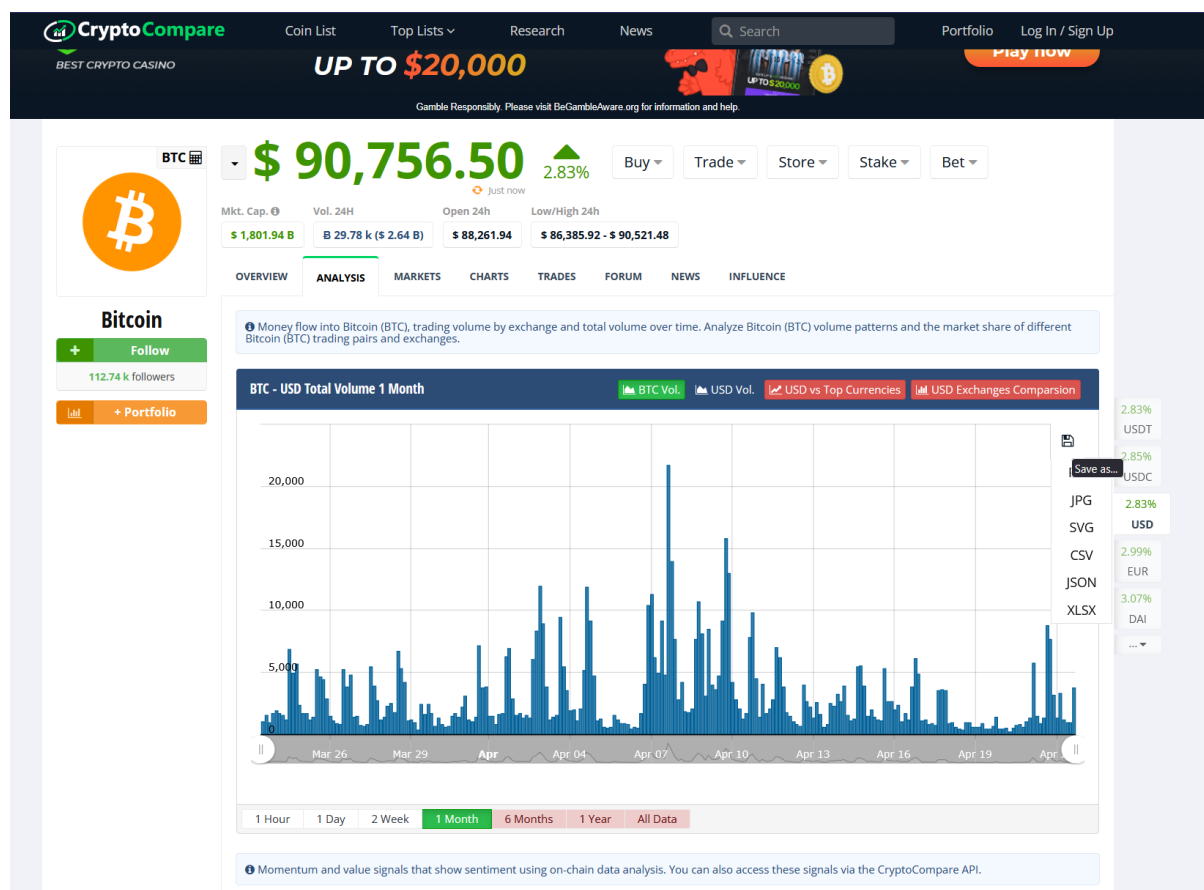


Рис. 1.8. Графік цін біткоїна на CryptoCompare
із функцією експорту

CryptoCompare забезпечує безкоштовний доступ до базових функцій і API, що робить її привабливою для трейдерів, аналітиків і розробників, які шукають ринкові дані без додаткових витрат. Платформа дозволяє порівнювати ціни та обсяги торгів між біржами, такими як Binance і Kraken, що допомагає виявляти ринкові розбіжності. Інтеграція новинного контенту додає цінність, надаючи користувачам контекст для аналізу ринкових тенденцій. Зручний трекер портфеля дозволяє ефективно відстежувати криптоактиви, що особливо корисно для індивідуальних інвесторів.

CryptoCompare має обмеження, які знижують її ефективність для складних аналітичних задач. API дозволяє лише 50 запитів на годину, що ускладнює швидкий збір великих обсягів даних. Відсутність інструментів для автоматичного очищення даних, таких як фільтрація аномалій чи обробка пропусків, змушує користувачів покладатися на зовнішні програми. Платформа не підтримує вибір додаткових факторів, таких як хешрейт чи макроекономічні показники, що обмежує її для багатофакторного аналізу. Крім того, перевантажений інтерфейс із великою кількістю функцій може бути складним для новачків, вимагаючи часу на освоєння.

1.3. Glassnode

Glassnode – це провідна платформа для аналізу блокчейну, яка спеціалізується на ончейнових метриках [5], ринкових даних і аналітиці криптовалют, зокрема біткоїна. Вона ідеально підходить для інституційних інвесторів, аналітиків і розробників, які потребують детальних даних для прогнозування та аналізу.

Основні функції Glassnode включають:

- 1) дані про активність мережі, такі як хешрейт, кількість транзакцій, активні адреси та баланси гаманців;
- 2) ціни, обсяги торгів і ринкова капіталізація з провідних бірж (Binance, Coinbase, Kraken тощо);
- 3) метрики поведінки інвесторів, наприклад, індекси страху та жадібності [6];
- 4) інтерактивні графіки та дашборди для аналізу тенденцій;
- 5) доступ до даних через API у форматах CSV або JSON.

Інтерфейс Glassnode (рис. 1.9) орієнтований на професійних користувачів і зручність роботи з великими обсягами даних. Головна панель включає дашборди з графіками ончейнових метрик (наприклад, обсяг транзакцій) і ринкових показників (наприклад, волатильність цін). Користувачі можуть налаштовувати метрики, змінювати часові рамки та порівнювати кілька показників на одному графіку. Хоча інтерфейс

інтуїтивний для аналітиків із досвідом, новачкам може знадобитися час для освоєння через технічну складність даних.

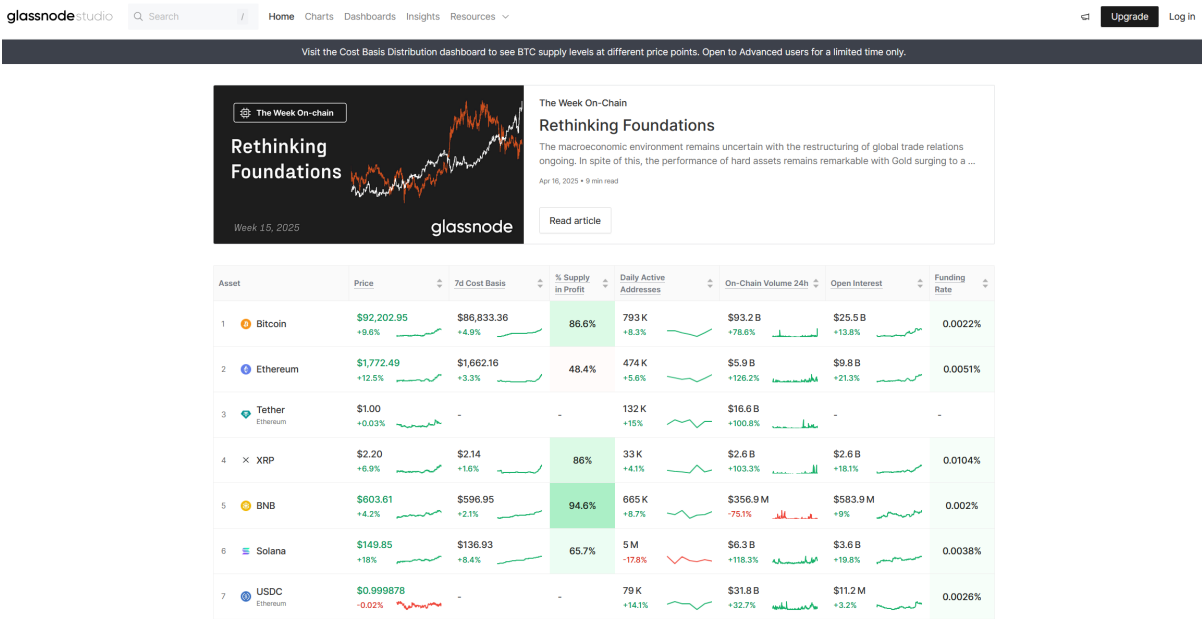


Рис. 1.9. Головна панель Glassnode з дашбордами, що відображають рух біткоїнів і активність мережі

Glassnode підтримує платний API (рис. 1.10), який забезпечує доступ до ончейнових і ринкових даних у реальному часі. Він сумісний із популярними інструментами, такими як Python (з бібліотеками pandas для обробки даних і matplotlib/seaborn для візуалізації), R (для статистичного аналізу) та платформи бізнес-аналітики, як-от Tableau, Power BI або Google Data Studio. У документації API надано приклади запитів у Python і curl, що спрощує інтеграцію в автоматизовані робочі процеси чи підготовку даних для машинного навчання. Така гнучкість робить Glassnode зручним інструментом для аналітиків і розробників.

9 Glassnode Docs

Website Studio Insights Twitter

Search... Ctrl K

API

Blockchain

Breakdowns

DeFi

Derivatives

Distribution

Entities

ETH 2.0

Fees

Indicators

Institutions

Lightning

Market

Mempool

Mining

Point-In-Time

Protocols

Signals

Supply

Transactions

FURTHER INFORMATION

Changelog

Support FAQ

Affiliate FAQs

Exchange Data: Transparency Notice

API > ENDPOINTS

Market

Asia Month-over-Month Price Change

GET https://api.glassnode.com/v1/metrics/market/apac_30d_price_change

This metric shows the 30-day change in the regional price set during Asia working hours, i.e. between 8am and 8pm China Standard Time (00:00-12:00 UTC).

Regional prices are constructed in a two-step process: First, price movements are assigned to regions based on working hours in the US, Europe, and Asia. Regional prices are then determined by calculating the cumulative sum of the price changes over time for each region.

[View in Studio](#)

Query Parameters

Name	Type	Description
a*	string	asset id: BTC, ETH
s	integer	since, unix timestamp
u	integer	until, unix timestamp
i	string	frequency interval: 24h
f	string	format: csv, json
timestamp_format	string	timestamp format: unix or humanized (RFC 3339)

200 OK

400 Bad Request

```
[{"t":1726790400,"v":-397.33830525659505}]
```

Market Cap

Median MVRV

Median Realized Price

MVRV Ratio

MVRV Z-Score

Price

Price Drawdown from ATH

Price OHLC

Realized Cap

Realized Price

Realized Volatility (1 Month)

Realized Volatility (1 Week)

Realized Volatility (1 Year)

Realized Volatility (2 Weeks)

Realized Volatility (3 Months)

Realized Volatility (6 Months)

Realized Volatility (All)

Spot Buying Volume Intraday

Spot Selling Volume Intraday

Spot Volume

Spot Volume (Latest 24h)

Spot Volume (Stacked)

Spot Volume Delta

Spot Volume Intraday

STH-MVRV

US Month-over-Month Price Change

Рис. 1.10. Документація API Glassnode з прикладами запитів у Python і curl

Glassnode вирізняється аналітичною підтримкою та можливостями звітності (рис. 1.11). Платформа пропонує готові щотижневі та щомісячні огляди ринку від команди експертів, які базуються на ончейнових даних і включають прогнози, забезпечуючи користувачів глибоким розумінням ринкових тенденцій. Користувачі можуть створювати власні дашборди, налаштовуючи їх під індивідуальні потреби, і експортувати результати у форматі PDF, CSV або зображення для зручного використання в презентаціях чи звітах. Такі звіти є цінним ресурсом як для інвесторів так і для дослідників. Загалом, аналітичний підхід Glassnode забезпечує міцну основу для стратегічного планування та наукових досліджень, але для повноцінної підготовки даних до машинного навчання чи прогнозування платформа потребує доповнення власними алгоритмами обробки, щоб адаптувати ончейнові дані до специфічних аналітичних задач.

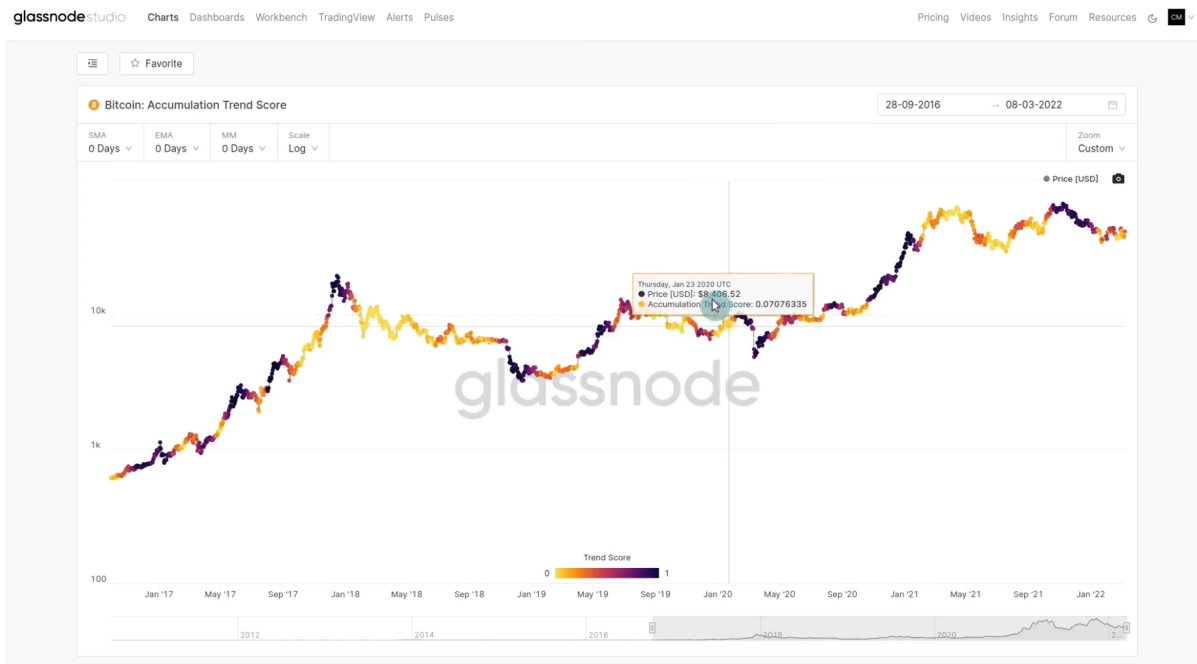


Рис. 1.11. Приклад звіту про ончейнові метрики, що показує накопичення біткоїнів великими гравцями

Glassnode є потужним інструментом для аналізу біткоїна завдяки своїм численним сильним сторонам. Платформа пропонує глибокий аналіз ончейнових даних, що дозволяє здійснювати фундаментальне прогнозування на основі ключових метрик мережі, таких як хешрейт, активність транзакцій і поведінка інвесторів. Висока якість і регулярне оновлення метрик забезпечують актуальність і точність інформації, що є критично важливим для прийняття обґрунтованих рішень. Крім того, інтерактивні інструменти візуалізації, такі як дашборди і графіки, допомагають швидко виявляти ринкові тренди та патерни. Користувачі також отримують аналітичну підтримку у вигляді звітів від експертів, що додає цінних інсайтів для стратегічного планування. Нарешті, гнучкість інтеграції з платформами та інструментами, як-от Python, R і Tableau, робить Glassnode зручним вибором для аналітиків і розробників.

Попри свої переваги, Glassnode має певні обмеження, які можуть вплинути на її використання. Висока вартість доступу до API та просунутих функцій може стати перешкодою для малих команд або індивідуальних користувачів із обмеженим бюджетом. Відсутність високочастотних даних, наприклад щохвилинних цін, робить платформу

менш придатною для короткострокового трейдингу, де швидкість реакції на ринкові зміни є ключовою. Також обмежена можливість порівняння даних між біржами в реальному часі ускладнює аналіз ліквідності чи пошук арбітражних можливостей. Для користувачів без досвіду в криптоаналітиці платформа може мати високий поріг входу через технічну складність даних, що вимагає додаткового навчання. Нарешті, відсутність макроекономічних даних обмежує можливості для комплексного аналізу зовнішніх факторів, які впливають на ринок.

1.4. Висновки

Ринок програмного забезпечення для збору та обробки даних про криптовалюту, зокрема біткоїн, є висококонкурентним і добре розвиненим, із платформами, такими як CoinGecko, CryptoCompare і Glassnode, які пропонують доступ до ринкових даних, історичних цін, обсягів торгів і ончейнових метрик, а також підтримують експорт у формати CSV і JSON. Однак ці сервіси мають обмеження, оскільки не забезпечують комплексного формування структурованих багатфакторних наборів даних із можливістю гнучкого вибору факторів (наприклад, хешрейт [7], транзакції, макроекономічні показники, волатильність), багатометодної фільтрації аномалій (Z-score, IQR, логічні перевірки), обробки пропусків із опціональною лінійною інтерполяцією, нормалізації числових даних (Min-Max, Standard), кодування категоріальних змінних (One-Hot, Label Encoding) чи генерації цільових змінних для задач регресії та класифікації, як передбачено у цьому проєкті. Більшість платформ, таких як Glassnode, орієнтовані на інституційних клієнтів [8] і мають високу вартість, що робить їх менш доступними для індивідуальних аналітиків, дослідників чи невеликих команд розробників машинного навчання.

Для нових розробників у цій сфері ключовим є впровадження інноваційних функцій, які відповідають потребам сучасного аналізу та прогнозування. Розробка вебзастосунку, що автоматизує агрегацію даних із публічних API бірж (Binance, Coinbase, Kraken), забезпечує ретельну

валідацію, очищення та підготовку даних до ML-пайплайнів, а також пропонує зручний доступ до вибору факторів, нормалізації, кодування та створення цільових змінних, має значний потенціал. Такий застосунок може інтегруватися з аналітичними інструментами, такими як Jupyter Notebook чи Python-бібліотеки (наприклад, scikit-learn, TensorFlow), що підвищує його цінність для розробників ML. Доступність через використання відкритих API без ключів і низька вартість порівняно з преміум-сервісами створюють конкурентну перевагу, особливо для індивідуальних користувачів і малого бізнесу. Для залучення аудиторії доцільно застосовувати маркетингові стратегії, такі як безкоштовний пробний доступ, активна взаємодія з криптовалютними спільнотами на платформах, таких як X, і створення навчальних матеріалів для демонстрації можливостей застосунку. Перспективи для нових розробників залишаються сприятливими за умови зосередження на унікальних функціях, високій якості даних і зручності використання, що дозволить задовольнити потреби аналітиків, дослідників і розробників машинного навчання в ефективних інструментах для аналізу ринку біткоїна.

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Мови програмування

2.1.1. *Python*

Python є однією з найпопулярніших мов програмування, відомою своєю універсальністю, простотою та потужною екосистемою бібліотек. Вона широко застосовується в задачах обробки даних, веброзробки, автоматизації та аналізу великих обсягів інформації. Для вебзастосунку, який агрегує дані з криптовалютних бірж, очищає їх від аномалій і готує для прогнозування, Python є оптимальним вибором завдяки своїм численним перевагам.

Python вирізняється чітким і лаконічним синтаксисом, який полегшує написання та читання коду. Це дозволяє розробникам зосередитися на логіці програми, а не на складних конструкціях мови. Для проєкту, який включає складні операції, такі як крос-валідація даних між біржами чи фільтрація аномалій, простота Python значно скорочує час розробки та зменшує ймовірність помилок.

Python пропонує широкий спектр бібліотек, які ідеально підходять для обробки фінансових і ринкових даних `pandas`, `numpy`, `scikit-learn`, `imbalanced-learn`, `scipy`, `fredapi` і `yfinance`. Наприклад, бібліотека `pandas` забезпечує зручну роботу з табличними даними, дозволяючи виконувати операції фільтрації, групування та обчислення статистичних показників, таких як стандартне відхилення для оцінки волатильності. Бібліотека `numpy` оптимізує числові обчислення, необхідні для методів виявлення аномалій, таких як Z-score або IQR. Для роботи з зовнішніми API бірж (наприклад, `Binance`, `Coinbase`, `Kraken`) використовується бібліотека `requests`, яка забезпечує надійне отримання даних. Бібліотека `yfinance` дозволяє отримувати макроекономічні показники, такі як індекс S&P 500 [9] чи ціна золота, що є важливими факторами для аналізу.

Для ефективного збору даних із кількох джерел Python підтримує асинхронне програмування через бібліотеки `asyncio` і `aiohttp`. Це дозволяє одночасно надсилати запити до API різних бірж, зменшуючи час

очікування та підвищуючи продуктивність. Асинхронний підхід є критично важливим для обробки великих обсягів даних у реальному часі, що відповідає потребам проекту.

Python працює на всіх основних операційних системах, включаючи Windows, Linux і macOS, що забезпечує гнучкість у виборі середовища розробки та розгортання. Це дозволяє легко переносити застосунок між різними платформами без необхідності суттєвих змін у коді. Універсальність Python також проявляється в його здатності інтегруватися з іншими технологіями, такими як вебфреймворки для бекенду чи, наприклад, CSS-фреймворки для фронтенду.

Python має одну з найбільших і найактивніших спільнот розробників у світі. Це забезпечує доступ до величезної кількості документації, навчальних матеріалів, форумів і готових рішень на платформах, таких як Stack Overflow і GitHub. Для розробників це означає швидке вирішення проблем, пов'язаних із інтеграцією API, обробкою даних або оптимізацією продуктивності. Спільнота також сприяє регулярному оновленню бібліотек, що гарантує їх актуальність і безпеку.

Python, завдяки бібліотекам pandas і numpy, забезпечує швидку обробку великих наборів даних. Наприклад, операції фільтрації аномалій, крос-валідації цін між біржами чи лінійної інтерполяції пропусків виконуються з мінімальними витратами часу [10]. Це дозволяє обробляти набори даних до 3 тис. записів менше ніж за 3 секунди, що є ключовою вимогою для продуктивності застосунку.

Python пропонує інструменти для забезпечення безпеки даних, такі як валідація вхідних параметрів, обробка помилок API і захист від некоректних даних. Наприклад, бібліотека requests дозволяє обробляти HTTP-помилки, а модуль logging забезпечує детальне логування збоїв, що підвищує надійність застосунку. Python також підтримує шифрування даних для безпечної передачі, що є важливим для роботи з фінансовими даними.

Python підтримує модульне програмування, дозволяючи розділяти код на логічні компоненти, такі як модулі для отримання даних, фільтрації

аномалій чи генерації звітів. Це полегшує тестування, підтримку та масштабування проєкту. Модульність також сприяє повторному використанню коду, що зменшує дублювання та підвищує ефективність розробки.

2.1.2. *JavaScript*

JavaScript є ключовою мовою для створення інтерактивних вебінтерфейсів і клієнтської логіки. У цьому проєкті JavaScript (зокрема vanilla JS) використовується для реалізації фронтенду, що забезпечує зручний і мінімалістичний інтерфейс для вибору параметрів, таких як період даних, джерела чи методи фільтрації. JavaScript ідеально підходить для цих задач завдяки своїм унікальним характеристикам.

JavaScript є основною мовою для створення динамічних і чутливих вебінтерфейсів. У цьому проєкті він використовується для реалізації таких елементів, як календар для вибору дат, чекбокси для вибору факторів, випадаючі меню для форматів експорту та перемикачі для налаштування фільтрів. Використання vanilla JS дозволяє уникнути залежності від складних фреймворків.

Синтаксис JavaScript є відносно простим і доступним для розробників із базовими знаннями програмування. Це дозволяє швидко створювати функціональні елементи інтерфейсу, такі як валідація введених дат чи відображення статусу обробки даних. Велика кількість навчальних ресурсів, включаючи офіційну документацію Mozilla Developer Network, що сприяє швидкому освоєнню мови.

JavaScript підтримується всіма сучасними веббраузерами, такими як Google Chrome, Mozilla Firefox, Safari та Microsoft Edge. Це гарантує, що інтерфейс застосунку буде працювати коректно для всіх користувачів без необхідності встановлення додаткових плагінів чи програмного забезпечення. Така універсальність забезпечує доступність застосунку на різних пристроях.

JavaScript підтримує асинхронне програмування через механізми `async/await`, `Promises` і `fetch API`. Це дозволяє ефективно обробляти запити

до бекенду, наприклад, надсилання POST-запитів із параметрами вибору даних чи отримання файлів для завантаження. Асинхронність забезпечує швидку реакцію інтерфейсу на дії користувача, що підвищує зручність використання.

Сучасний JavaScript (ES6+) підтримує модульність через конструкції `import` і `export`, що дозволяє розділяти код на логічні компоненти, такі як функції для валідації чи обробки подій. Це полегшує підтримку та тестування клієнтської частини, особливо для складних інтерфейсів із багатьма інтерактивними елементами.

JavaScript має одну з найбільших спільнот розробників, що забезпечує доступ до численних ресурсів, бібліотек і готових рішень. Хоча проєкт використовує `vanilla JS`, екосистема JavaScript дозволяє легко використати додаткові бібліотеки, якщо в майбутньому знадобиться розширення функціоналу, наприклад, для покращення візуалізації даних.

JavaScript забезпечує потужні можливості для маніпуляції DOM (Document Object Model), що дозволяє динамічно оновлювати інтерфейс у відповідь на дії користувача. Наприклад, вибір періоду в календарі чи зміна налаштувань фільтрації може миттєво відображатися в інтерфейсі без перезавантаження сторінки.

JavaScript дозволяє створювати адаптивні інтерфейси, які коректно відображаються на різних роздільних здатностях. У поєднанні з Tailwind CSS (CSS-фреймворком), JavaScript забезпечує гнучке керування стилями та поведінкою елементів.

2.2. Бази даних

2.2.1. *PostgreSQL*

Для системи MLBit, розгорнутій на Render, обрано реляційну базу даних PostgreSQL – реляційну базу даних, яка має відкритий код і чудово працює завдяки підтримці PostgreSQL на платформі Render, забезпечуючи зручне зберігання даних та ефективну обробку складних ETL-процесів у хмарі. PostgreSQL забезпечує надійне зберігання сирих, інтерпольованих і похідних даних, включаючи ринкові, блокчейн- і макроекономічні метрики,

а також кастомізовані ознаки, необхідні для аналітики та машинного навчання.

PostgreSQL організує дані в таблиці `market_data` (сирі OHLCV, блокчейн-метрики, макроекономічні показники), `i_market_data` (інтерпольовані дані після обробки пропусків) і `custom_factors` (похідні ознаки, такі як технічні індикатори, крос-біржові спреди, співвідношення хешрейту до складності). Використання первинного ключа `date` (тип `Date`) забезпечує унікальність записів і швидкий доступ до часових рядів. Така структура дозволяє ефективно керувати великими наборами даних.

PostgreSQL оптимізує запити для часових рядів, використовуючи індекси на колонці `date` та підтримуючи швидке виконання операцій вибірки, фільтрації й об'єднання. Наприклад, запити для агрегації щоденних даних або отримання похідних ознак виконуються за мілісекунди для наборів до 500 записів, що відповідає вимогам продуктивності. База підтримує масштабування шляхом додавання нових таблиць або колонок для нових джерел даних (наприклад, Bitfinex) без зміни архітектури.

PostgreSQL забезпечує транзакційну цілісність, що запобігає втраті даних під час збоїв. Функції резервного копіювання та точкового відновлення дозволяють зберігати стан бази й відновлювати дані після перезапуску сервера, що критично для відмовостійкості системи. Логування операцій із базою через SQLAlchemy фіксується в `logs/app.log`, надаючи детальну інформацію для діагностики.

PostgreSQL інтегрується з Python через SQLAlchemy. Моделі дозволяють виконувати запити до бази через об'єктно-орієнтований інтерфейс, зменшуючи складність коду. Наприклад, збереження сирих даних із Binance API чи вибірка похідних ознак для експорту реалізується кількома рядками коду, що підвищує продуктивність розробки.

Розгортання PostgreSQL на Render забезпечує безпеку даних, оскільки платформа пропонує ізольоване хмарне середовище для бази. Паролі та параметри підключення зберігаються в `.env` через `python-dotenv`,

а валідація запитів через SQLAlchemy захищає від SQL-ін'єкцій. Тимчасові дані видаляються після експорту в CSV/JSON.

PostgreSQL підтримує додавання нових таблиць для майбутніх джерел (наприклад, макроекономічні дані з інших API) або колонок для нових ознак, таких як додаткові технічні індикатори чи співвідношення. Схема бази даних дозволяє легко адаптувати структуру до нових вимог без порушення роботи системи.

PostgreSQL ефективно використовує ресурси (8+ GB RAM, 100+ GB диск), що дозволяє обробляти великі набори даних на стандартному апаратному забезпеченні. Використання формату Parquet для тимчасового зберігання зменшує навантаження на диск, а індекси оптимізують запити.

2.3. Фреймворки та бібліотеки

2.3.1. *Flask*

Flask – це мікрофреймворк для Python, який використовується для створення бекенду вебзастосунку. Його легкість, гнучкість і простота роблять Flask ідеальним вибором для проєктів, що потребують швидкої розробки та мінімалістичної архітектури. Для цього проєкту Flask забезпечує обробку запитів, інтеграцію з API і генерацію вихідних файлів.

Flask має простий і зрозумілий API, який дозволяє швидко налаштувати маршрути для обробки запитів, таких як POST-запит для генерації даних. Flask бездоганно інтегрується з усіма Python-бібліотеками, що використовуються в проєкті, забезпечуючи єдину екосистему для обробки даних, роботи з API і генерації звітів. Це відповідає потребам проєкту, де бекенд зосереджений на обробці параметрів і поверненні результатів.

Flask не нав'язує строгих правил щодо структури проєкту чи використовуваних бібліотек, що дозволяє вільно інтегрувати pandas, sklearn, requests і yfinance для обробки даних. Така гнучкість забезпечує точну відповідність специфічним вимогам проєкту.

Flask ідеально підходить для створення RESTful API, що необхідно для взаємодії між фронтендом і бекендом. Наприклад, маршрут /api/

generate обробляє JSON-запити з параметрами, такими як період даних чи методи фільтрації, і повертає файли для завантаження.

Flask підтримує вбудоване логування через модуль logging, що дозволяє фіксувати збої, наприклад, помилки при отриманні даних із API бірж. Це підвищує надійність застосунку та полегшує діагностику проблем.

2.3.2. Tailwind CSS

Tailwind CSS – це утилітарний CSS-фреймворк, який використовується для створення адаптивного та мінімалістичного інтерфейсу. Його підхід до стилізації через класи дозволяє швидко створювати сучасні та функціональні дизайни, що відповідає вимогам проєкту.

Tailwind CSS дозволяє застосовувати стилі безпосередньо в HTML за допомогою утилітарних класів, що значно прискорює розробку інтерфейсу. Наприклад, календар, чекбокси чи випадаючі меню можна стилізувати за лічені хвилини.

Tailwind підтримує адаптивні класи, що дозволяють забезпечити коректне відображення інтерфейсу на різних роздільних здатностях, починаючи від 1280x720. Це гарантує зручність використання для всіх користувачів.

Tailwind використовує інструмент PurgeCSS для видалення невикористовуваних стилів, що зменшує розмір CSS-файлів і підвищує швидкість завантаження сторінок. Це особливо важливо для вебзастосунків із мінімалістичним дизайном.

Tailwind дозволяє легко налаштувати кольори, шрифти та інші стилі під потреби проєкту. Це забезпечує створення унікального дизайну, який відповідає естетичним і функціональним вимогам.

Tailwind ідеально працює з vanilla JS, дозволяючи динамічно змінювати стилі у відповідь на дії користувача, наприклад, підсвічувати вибрані чекбокси чи оновлювати статус обробки.

2.3.3. *Pandas*

Pandas – ключова Python-бібліотека для обробки та аналізу даних, яка є основою для реалізації функціоналу агрегації, очищення, нормалізації та експорту даних. Її потужні можливості роблять її незамінним інструментом для підготовки даних біткоіна до ринкового прогнозування.

Pandas використовує структури DataFrame для ефективної роботи з табличними даними. Це дозволяє легко виконувати операції крос-валідації цін між біржами (Binance, Coinbase, Kraken), обчислення середньої ціни, фільтрацію аномалій (Z-score, IQR), а також додавання факторів, таких як день тижня, волатильність чи хешрейт. Наприклад, `shift()` – створення цільових змінних для ML-задач, а методи `rolling().std()` забезпечують обчислення волатильності за 3-денним вікном [11].

Pandas підтримує гнучке управління пропусками в даних. У проєкті числові пропуски позначаються як NaN. Опціональна лінійна інтерполяція через метод `interpolate(method="linear")` застосовується до числових колонок, таких як ціни, хешрейт чи макроекономічні показники, що забезпечує повноту даних для ML-моделей.

Pandas надає метод `get_dummies()` для реалізації One-Hot Encoding категоріальних змінних, таких як `holidays`. Це дозволяє перетворювати текстові дані в бінарні колонки (наприклад, `"holidays_False"`, `"holidays_True"`), що є необхідним для підготовки даних до машинного навчання. Такий підхід забезпечує сумісність із ML-алгоритмами, які потребують числових вхідних даних.

Pandas гармонійно працює з `numpy` для числових обчислень, `requests` для обробки даних із API, `yfinance` для отримання макроекономічних показників і `sklearn.preprocessing` для нормалізації. Інтеграція з `yfinance` дозволяє додавати дані S&P 500 чи VIX до DataFrame, а з бібліотекою `numpy` – виконувати статистичні обчислення для фільтрації аномалій.

Pandas підтримує експорт у формати CSV і JSON із кодуванням UTF-8 через методи `to_csv()` і `to_json()`. Це забезпечує створення структурованих наборів даних із шаблонами "Базовий" (ціна, обсяг, день тижня) і "Розширений" (усі фактори, включаючи цільові змінні).

Pandas використовує оптимізовані алгоритми для обробки даних, що дозволяє виконувати складні операції, такі як крос-валідація чи обчислення різниці цін, із високою швидкістю. Для наборів даних до 3 тис. записів обробка виконується менше ніж за 3 секунди, що відповідає вимогам продуктивності.

Pandas забезпечує інструменти для додавання факторів, таких як свята чи волатильність. Наприклад, бібліотека дозволяє використовувати модуль `datetime` для визначення часу UTC і обчислення дня тижня, а також інтегруватися з бібліотекою `holidays` для визначення свят США.

Pandas спрощує створення звіту про якість даних шляхом обчислення статистики, такої як відсоток пропусків (`isna().mean()`), кількість аномалій чи середня різниця цін. Це дозволяє генерувати прозорі звіти, які додаються до експортованих файлів.

2.3.4. Scikit-learn

Scikit-learn – це провідна Python-бібліотека для машинного навчання, яка використовується в проєкті для реалізації нормалізації числових даних і кодування категоріальних змінних. Її модуль `sklearn.preprocessing` забезпечує інструменти для підготовки даних до ML-пайплайнів, що є ключовим для створення готових до використання наборів даних.

Scikit-learn надає класи `MinMaxScaler` і `StandardScaler` для нормалізації числових колонок, таких як ціна, обсяг, хешрейт, S&P 500, золото, VIX і різниця цін. `MinMaxScaler` масштабує дані до діапазону $[0, 1]$, що ідеально підходить для алгоритмів, чутливих до масштабу, таких як нейронні мережі. `StandardScaler` стандартизує дані (середнє = 0, стандартне відхилення = 1), що корисно для моделей, таких як лінійна регресія чи SVM. Користувач може обрати "Min-Max", "Standard" або "Вимкнено" через інтерфейс, що забезпечує гнучкість [12].

Scikit-learn пропонує `LabelEncoder` для реалізації Label Encoding, яке перетворює категоріальні змінні (наприклад, свята) у числові значення (наприклад, свято=1, не свято=0). Це корисно для алгоритмів, які приймають порядкові дані. Хоча One-Hot Encoding реалізується через

pandas.get_dummies(), інтеграція з sklearn.preprocessing забезпечує єдиний підхід до обробки даних, що підвищує зручність для ML-розробників.

Scikit-learn бездоганно працює з pandas DataFrame, дозволяючи застосовувати нормалізацію та кодування до вибраних колонок. Наприклад, методи fit_transform() забезпечують швидке масштабування числових даних, а результати легко інтегруються назад у DataFrame для подальшого експорту.

Алгоритми Scikit-learn оптимізовані для швидкої обробки даних, що дозволяє нормалізувати та кодувати набори до 500 записів за частки секунди. Це відповідає вимогам продуктивності проєкту (обробка менше ніж за 3 секунд).

Scikit-learn має інтуїтивний API, що полегшує реалізацію нормалізації та кодування. Наприклад, MinMaxScaler() і LabelEncoder() потребують мінімального налаштування, що прискорює розробку.

Завдяки використанню Scikit-learn дані відповідають стандартам ML-алгоритмів. Нормалізовані та закодовані дані готові до використання в моделях регресії, класифікації чи нейронних мережах без додаткової обробки, що економить час ML-розробників.

Scikit-learn має детальну документацію та активну спільноту, що дозволяє швидко вирішувати проблеми, пов'язані з нормалізацією чи кодуванням, і адаптувати бібліотеку до специфічних потреб проєкту.

2.3.5. Imbalanced-learn

Imbalanced-learn – бібліотека для балансування класів у задачах класифікації, що покращує якість прогнозних моделей.

SMOTE генерує синтетичні зразки для меншості (наприклад, "Up" у binary_price_direction), а RandomUnderSampler зменшує кількість зразків більшості, виправляючи дисбаланс [13].

Працює з pandas і scikit-learn, дозволяючи застосовувати балансування після генерації цільових змінних. Параметр balance_method у

API дозволяє обрати SMOTE, RandomUnderSampler або вимкнути балансування

Балансування виконується швидко, додаючи мінімальне навантаження до конвеєру обробки.

2.3.6. SciPy

SciPy – бібліотека для наукових обчислень, що підтримує складну обробку даних та інженерію ознак.

Модуль `scipy.interpolate` реалізує лінійну, кубічний сплайн і експоненціальну інтерполяцію для числових даних (ціни, хешрейт, S&P 500).

SciPy обчислює скіс, куртозис, кореляції (наприклад, біткоїн із золотом) для звіту про якість через `scipy.stats`. Підтримує розрахунок складних індикаторів, таких як ATR чи ADX, через числові методи [14]. Працює з `pandas` і `numpy`, забезпечуючи точність обчислень.

2.3.7. Інші бібліотеки

Для забезпечення повноцінної функціональності системи MLBit використано набір спеціалізованих бібліотек, які підтримують інтеграцію з API, обробку даних, створення звітів, безпеку конфігурації та тестування. `requests` забезпечує надійні HTTP-запити [15] до публічних API бірж (Binance, Coinbase, Kraken), Blockchain.com і FRED, підтримуючи повторні спроби (до 3 із експоненціальною затримкою) і логування помилок у `logs/app.log` для діагностики збоїв. `yfinance` і `fredapi` надають доступ до макроекономічних даних: `yfinance` отримує ринкові показники (S&P 500, VIX, золото, нафта) із резервними CSV для стабільності [16], а `fredapi` забезпечує економічні метрики (процентні ставки, CPI, держоблігації) через ключ API, що зберігається в `.env`. Бібліотека `pandas-datareader` доповнює `fredapi`, забезпечуючи альтернативний доступ до FRED, що підвищує надійність збору даних при нестабільності API. `holidays` генерує календарні ознаки, такі як державні свята США [17], які додаються до

DataFrame як `us_holiday` або `days_since_last_us_holiday`, підтримуючи інженерію ознак для аналізу ринкової поведінки.

Бібліотека `jinja2` формує структуровані звіти про якість даних у форматі TXT, використовуючи шаблони для відображення статистик (пропуски, аномалії, кореляції) і опису методів обробки, що підвищує прозорість системи [18]. `python-dotenv` забезпечує безпечне завантаження конфігураційних параметрів (ключі API, порти, шляхи до бази) із файлу `.env`, захищаючи чутливі дані від витоку. `memory_profiler` аналізує використання пам'яті під час обробки великих наборів даних, дозволяючи оптимізувати конвеєр для відповідності вимогам продуктивності. Бібліотека `responses` використовується для мокінгу API під час тестування, імітуючи відповіді від Binance чи Blockchain.com, що спрощує перевірку модулів без реальних запитів. SQLAlchemy, як ORM-бібліотека, забезпечує зручну роботу з PostgreSQL [19] через моделі даних (`market_data`, `i_market_data`, `custom_factors`), інтегрується з `pandas` для збереження й вибірки даних і захищає від SQL-ін'єкцій, відповідаючи вимогам безпеки та модульності.

2.4. Розгортання

2.4.1. Хмарна платформа *Render*

Розгортання вебзастосунку на хмарній платформі `Render` є стратегічним рішенням, яке забезпечує оптимальну продуктивність, безпеку та гнучкість для системи, зосередженої на агрегації та обробці даних у реальному часі. `Render` надає низку реальних переваг, які роблять його ідеальним вибором для цього проєкту, сприяючи стабільній роботі та ефективному управлінню ресурсами у хмарі.

Платформа `Render` легко налаштовується за допомогою Python і Flask, що дозволяє швидко розгорнути застосунок через простий процес конфігурації на стороні платформи. Використання вбудованих інструментів `Render`, таких як автоматичне масштабування та підтримка `gunicorn`, забезпечує стабільну та надійну роботу системи з мінімальними зусиллями. Це дозволяє зосередитися на вдосконаленні функціоналу, а не

на управлінні складною інфраструктурою. Крім того, Render автоматично інтегрується з GitHub, що скорочує час на розгортання та тестування.

Розгортання на Render оптимізує витрати, порівняно з іншими хмарними сервісами, такими як Google Cloud, AWS чи Azure, завдяки доступним тарифам і безкоштовному рівню для невеликих проєктів. Замість складного управління локальним обладнанням, Render пропонує готову інфраструктуру, що економить ресурси та дозволяє ефективно розподіляти бюджет на розробку, тестування та вдосконалення користувацького досвіду.

Render забезпечує надійне управління даними, їх обробкою та безпекою в хмарі. Дані, отримані з API бірж, обробляються в пам'яті Render і експортуються у файли, які видаляються після завантаження користувачем, що мінімізує ризик витоку інформації чи несанкціонованого доступу. Це критично важливо для роботи з фінансовими даними. Крім того, Render дозволяє точно контролювати кожен етап обробки, від отримання даних до генерації звітів, забезпечуючи високу точність і надійність.

Render створює ідеальні умови для динамічної розробки та вдосконалення застосунку. Можна швидко вносити зміни до коду через автоматичне розгортання з GitHub, тестувати нові функції, такі як додаткові методи фільтрації чи формати експорту, і перевіряти їх у хмарному середовищі. Ця гнучкість дозволяє оперативно реагувати на нові вимоги чи виявлені проблеми, забезпечуючи високу якість кінцевого продукту. Render також спрощує інтеграцію нових API чи бібліотек, коли функціонал застосунку розширюється.

Render мінімізує залежність від локальних ресурсів, забезпечуючи стабільність роботи навіть у разі локальних мережеских проблем. Навіть при тимчасовій недоступності однієї з бірж (наприклад, Binance чи Kraken), Render дозволяє продовжити обробку даних із інших джерел завдяки хмарній інфраструктурі. Модуль logging у Python фіксує всі збої та помилки, дозволяючи швидко їх виявляти та усувати через логи Render, що підвищує загальну стабільність системи.

Render забезпечує високу швидкість обробки даних завдяки оптимізованій хмарній інфраструктурі, яка усуває затримки, пов'язані з локальним обладнанням. Це дозволяє обробляти набори даних до 3 тис. записів за лічені секунди, що відповідає вимогам до продуктивності фінансових застосунків. Render також автоматично налаштовує ресурси, наприклад, масштабуючи контейнери для максимальної ефективності.

Хмарна платформа Render дозволяє масштабувати ресурси в межах хмарної інфраструктури. Наприклад, можна збільшити обсяг пам'яті чи потужність обчислень для обробки більших наборів даних або автоматично розподілити навантаження між кількома інстансами. Така масштабованість забезпечує гнучкість для майбутнього розвитку застосунку в хмарі.

Render спрощує інтеграцію з іншими хмарними інструментами, такими як сервіси моніторингу, аналітичні платформи чи інструменти для резервного копіювання. Наприклад, резервні CSV-файли для макроекономічних даних чи API Blockchain.com можуть зберігатися у хмарному сховищі Render, забезпечуючи доступ до даних у разі нестабільності зовнішніх джерел.

2.5. Висновки

Для створення системи MLBit, що агрегує, очищає та готує дані біткоїна для ринкового прогнозування, обрано технології, які забезпечують продуктивність, безпеку та підготовку даних до машинного навчання. Python і JavaScript (vanilla JS) є основними мовами, Flask – бекенд-фреймворк, Tailwind CSS – для опціонального фронтенду, Pandas, Scikit-learn, Imbalanced-learn, SciPy – для обробки даних, PostgreSQL – база даних, а платформа Render із вбудованим сервером Flask – для хмарного розгортання. Додаткові бібліотеки (requests, yfinance, fredapi, pandas-datareader, holidays, jinja2, python-dotenv, memory_profiler, responses, SQLAlchemy) розширюють функціональність.

Python забезпечує агрегацію даних із бірж (Binance, Coinbase, Kraken), Blockchain.com, FRED, крос-валідацію, очищення, інженерію ознак (індикатори, співвідношення, календарні ознаки) і звіти через pandas,

numpy, requests, yfinance, fredapi, scipy, imbalanced-learn. JavaScript створює адаптивний фронтенд із формами, календарями (Flatpickr) і чекбоксами для налаштування параметрів. Flask підтримує REST API, інтегруючись із Python-бібліотеками та логуючи збої в logs/app.log, з обробкою запитів за декілька секунд. Tailwind CSS забезпечує мінімалістичний інтерфейс із темною темою.

Pandas агрегує, очищає (Z-score, IQR, інтерполяція), додає фактори та експортує дані в CSV/JSON, інтегруючись із numpy, scipy, scikit-learn, SQLAlchemy. Scikit-learn нормалізує (MinMaxScaler, StandardScaler, RobustScaler) і кодує (LabelEncoder) дані для ML-моделей. Imbalanced-learn балансує класи (SMOTE, RandomUnderSampler), покращуючи класифікацію. SciPy підтримує інтерполяцію та статистику для звітів. PostgreSQL зберігає сирі, інтерпольовані та похідні дані, інтегруючись із SQLAlchemy для безпеки й продуктивності.

Додаткові бібліотеки забезпечують запити до API (requests), макроекономічні дані (yfinance, fredapi, pandas-datareader), календарні ознаки (holidays), звіти (jinja2), безпеку (python-dotenv), оптимізацію пам'яті (memory_profiler) і тестування (responses). Хмарна платформа Render із Flask забезпечує простоту, економічність, безпеку та продуктивність, відповідаючи потребам аналітиків, трейдерів, дослідників і ML-інженерів у масштабованому середовищі. Render пропонує зручне розгортання через інтеграцію з GitHub, що дозволяє швидко оновлювати код і тестувати нові функції. Завдяки автоматичному масштабуванню та підтримці PostgreSQL, платформа ефективно обробляє великі обсяги даних. Економічна вигода від безкоштовного тарифу робить Render доступним, а безпека даних забезпечується ізольованим хмарним середовищем.

3.1. Загальний опис програми

Вебзастосунок MLBit розроблено для створення комплексного конвеєра обробки даних біткоіна, що охоплює агрегацію, очищення, збагачення та підготовку даних із різноманітних джерел для ринкового аналізу та прогнозування. Система орієнтована на забезпечення високої якості даних, придатних для поглибленого аналізу ринку, розробки торгових стратегій і навчання моделей машинного навчання (ML). Основними джерелами даних є криптовалютні біржі (Binance, Coinbase, Kraken), Blockchain.com (блокчейн-метрики), Yahoo Finance (ринкові індекси) та FRED (макроекономічні показники). Застосунок призначений для аналітиків даних, квантитативних трейдерів і дослідників ML, які потребують структурованих і оброблених наборів даних.

Користувач взаємодіє з системою через вебінтерфейс або API, налаштовуючи параметри обробки: період даних (до 5-ти років), джерела, набір ознак (OHLCV, технічні індикатори, кастомізовані фактори, макроекономічні показники), методи очищення (Z-score, IQR, вінзоризація), трансформації (Йео-Джонсона), нормалізації (MinMaxScaler, StandardScaler, RobustScaler), кодування категорій (LabelEncoder, OneHotEncoder) і балансування класів (SMOTE, RandomUnderSampler). Процес обробки включає збір даних, уніфікацію форматів, крос-валідацію, фільтрацію аномалій, додавання інженерних ознак, генерацію цільової змінної та створення звіту про якість. Результатом є ZIP-архів із файлами даних (CSV або JSON) і текстовим звітом, що містить статистику обробки.

Унікальність системи полягає в розширеній інженерії ознак (міжбіржові спреди, співвідношення, метрики волатильності, кореляції), просунутих методах обробки (вінзоризація, Йео-Джонсона), підтримці ML-функцій (генерація цільової змінної, балансування класів) і деталізованій звітності. Надійність забезпечується резервними файлами CSV, логуванням і обробкою помилок. Система підтримує як синхронну, так і асинхронну обробку з можливістю моніторингу статусу завдань.

3.2. Аналіз функціональних вимог до вебзастосунку

Функціональні вимоги розподілено за пріоритетами: "1" – критичні функції, без яких система не працює; "2" – важливі, але не є критичними; "3" – додаткові для зручності. Розподіл вимог та пріоритетів деталізовано у табл. 3.1.

Таблиця 3.1

Функціональні вимоги до вебзастосунку

Код вимоги	Назва вимоги	Пріоритет вимоги
01	Збір даних із хоча б одного джерела (Binance, Coinbase, Kraken)	1
02	Уніфікація форматів даних (дати, стовпці)	1
03	Експорт даних у CSV або JSON	1
04	API-ендпоінт для ініціації обробки (/api/collect)	1
05	Вибір періоду даних (до 5-ти років)	2
06	Вибір джерел (Blockchain.com, Yahoo Finance, FRED, Alpha Vantage)	2
07	Додавання ознак (OHLCV, технічні індикатори, кастомізовані фактори)	2
08	Крос-валідація цін між біржами (>2%)	2
09	Фільтрація аномалій (Z-score, IQR, логічні перевірки)	2
10	Налаштування порогів Z-score (3.0/4.0)	3

Продовження табл. 3.1

11	Обробка пропусків (лінійна інтерполяція)	2
12	Трансформація даних (вінзоризація, Йео-Джонсона)	2
13	Нормалізація (MinMaxScaler, StandardScaler, RobustScaler)	2

14	Кодування категорій (LabelEncoder, OneHotEncoder)	2
15	Генерація цільової змінної для ML	2
16	Балансування класів (SMOTE, RandomUnderSampler)	2
17	Генерація звіту про якість даних	2
18	Асинхронна обробка з моніторингом статусу	3
19	Логування подій і помилок	2
20	Резервні CSV-файли для збоїв API	2
21	Розрахунок міжбіржових спредів і співвідношень	2
22	Вебінтерфейс для налаштування параметрів	3

3.3. Архітектура системи

3.3.1. Клієнт-серверна архітектура

Система побудована на клієнт-серверній архітектурі, що розділяє функціональність на серверну частину (Flask-додаток) і клієнтську (вебінтерфейс або API-клієнт). Це забезпечує централізовану обробку даних і гнучкість доступу.

Сервер розгорнутий на платформі Render, обробляє HTTP-запити через Flask. Клієнт може бути веббраузером, що відображає сторінки HTML, або будь-яким HTTP-клієнтом (наприклад, cURL, скрипт Python), що взаємодіє з API. Використовується стандартний протокол HTTP із базовими налаштуваннями безпеки, які забезпечує хмарне середовище Render.

3.3.2. Серверна частина

Перед початком обробки даних сервер виконує валідацію введених параметрів, таких як діапазони дат, вибрані джерела та методи обробки. Дані збираються з зовнішніх API (Binance, Coinbase, Kraken, Blockchain.com, Yahoo Finance, FRED) або з резервних CSV-файлів у разі недоступності сервісів.

Після збору даних відбувається етап їх обробки: уніфікація формату, очищення, трансформація, нормалізація, кодування категоріальних змінних та балансування класів у випадку задач класифікації. Для зберігання метаданих використовується база даних PostgreSQL із доступом через SQLAlchemy.

Сервер також відповідає за генерацію звітів і експорт підсумкових даних у формати CSV, JSON або ZIP. Для моніторингу роботи та відстеження помилок використовується модуль logging. Асинхронні операції, такі як обробка великих обсягів даних або формування файлів, управляються через унікальні task_id та систему тимчасових файлів.

3.3.3. Клієнтська частина

Інтерфейс користувача виступає посередником між людиною та системою, надаючи прості інструменти для налаштування, запуску та перегляду результатів обробки даних. Основна його функція – полегшити взаємодію з функціоналом системи без необхідності заглиблюватися в технічні деталі.

Графічна частина реалізована на базі HTML-шаблонів, зокрема сторінок на кшталт index.html та collect.html. Вони містять елементи керування – поля для вибору дат, перемикачі, списки та інші форми введення. Вся логіка динамічної взаємодії реалізована за допомогою JavaScript без сторонніх бібліотек: надсилення асинхронних запитів до сервера, оновлення статусів завдань, обробка відповідей і виведення повідомлень.

Запити до API надсилаються через HTTP методом POST на адресу /api/collect. У відповідь клієнт отримує ідентифікатор завдання (task_id), який потім використовується для отримання результатів із сервера за допомогою запиту до /download/<task_id>. Це забезпечує гнучку та інтерактивну взаємодію без перезавантаження сторінок.

3.3.4. Взаємодія між клієнтом і сервером

Клієнтська та серверна частини системи обмінюються інформацією через мережеві протоколи, що дозволяє відокремити інтерфейс користувача від логіки обробки даних. Завдяки цьому підходу клієнти взаємодіють із централізованим ядром застосунку, яке приймає запити, виконує необхідні обчислення та повертає результати у зручному вигляді. Така модель сприяє стабільності, безпеці та легкому масштабуванню рішення.

Комунікація організована через набір HTTP-ендпоінтів:

1. GET / – ініціалізує завантаження стартової сторінки.
2. GET /collect – відкриває форму для введення налаштувань.
3. POST /api/collect – отримує параметри, запускає обробку даних, повертає task_id або одразу готовий файл.
4. GET /download/<task_id> – дозволяє користувачеві отримати архів з результатами.

Ця модель побудована з урахуванням багатокористувацького середовища – логіка централізована, клієнти легкі, а оновлення системи не вимагає втручання на стороні користувача. Всі обчислення відбуваються на сервері, що також підвищує рівень контролю та безпеки.

Архітектурний шаблон MVT (Model-View-Template) лежить у основі застосунку та поділяє функціональність на три незалежні компоненти:

1. Model відповідає за управління інформацією, бізнес-правилами та обчисленнями. Саме тут відбувається вся основна логіка роботи з даними.
2. Template реалізує візуальний інтерфейс: відображає результати, елементи керування, повідомлення та дозволяє взаємодіяти із системою.
3. View Function приймає дії користувача, ініціює зміни у моделі та оновлення вигляду, виступаючи координатором між логікою та представленням.

Завдяки такій структурі система є гнучкою до змін, підтримуваною та зручною для подальшого розширення. Логіка, зовнішній вигляд і взаємодія між ними можуть змінюватись незалежно одне від одного.

3.4. Шаблон Model-Template-View (MTV)

3.4.1. Компоненти MTV

Система використовує підхід, близький до Model-View-Controller, адаптований для Flask, із чітким розділенням відповідальностей. MTV поділяється на три компоненти: модель, контролер, представлення.

Компонент моделі реалізує рівень доступу до даних та інкапсулює бізнес-логіку, яка визначає правила обробки й трансформації інформації. Структура моделі представлена через ORM-схеми на базі бібліотеки SQLAlchemy, що дозволяє описувати таблиці реляційної бази даних PostgreSQL у вигляді Python-класів. У даному програмному рішенні реалізовано моделі для зберігання первинних ринкових даних, обчислених індикаторів та користувацьких факторів, що є ключовими елементами для подальшого аналізу та прогнозування.

Крім базових схем, логіка моделі включає модулі збору та обробки інформації, зокрема `data_processing/api_clients.py` для роботи з API, та `data_processing/processing.py` для виконання процедур очищення, фільтрації, нормалізації, кодування та балансування вибірки. У реалізації застосовано численні інструменти обробки даних, серед яких – `pandas`, `numpy`, `scikit-learn`, `imbalanced-learn`, `fredapi`. Передбачено підтримку крос-валідації, обробки аномалій, інтерполяції, вінзоризації та трансформацій на основі методу Йео-Джонсона. Завершальним етапом виступає формування звітної інформації у вигляді текстових файлів та структурованих архівів.

Контролер (View Function) виступає посередником між логікою обробки даних та інтерфейсною частиною, керуючи маршрутизацією запитів та реалізацією прикладної логіки у відповідь на дії користувача. Основні контролери розміщено у модулі `app.py`, де визначено функції-обробники маршрутів, які відповідають за завантаження сторінок, прийом параметрів через API, запуск процесів обробки даних, завантаження

сформованих архівів та перевірку статусу завдань. Обробка вхідних даних супроводжується валідацією параметрів, що гарантує коректність і безпечність переданої інформації.

Згідно з шаблоном архітектури Model-Template-View, контролер забезпечує слабе зв'язування між компонентами, сприяючи розширюваності та зручності супроводу системи. У випадку виникнення помилок застосовуються механізми їх централізованої обробки з використанням декораторів `@app.errorhandler`, що дозволяє забезпечити стабільність і надійність під час виконання запитів. Таким чином, контролер координує взаємодію між модельною та представницькою частинами системи, забезпечуючи коректну реалізацію сценаріїв використання програмного засобу.

Шар представлення (Presentation Layer) забезпечує взаємодію між користувачем і програмною системою, реалізуючи виведення інформації та інтерфейсні засоби керування процесом. Вебінтерфейс базується на HTML-шаблوناх, що динамічно рендеряться за допомогою рушія Jinja2. Інтерфейсні компоненти реалізовані у вигляді форм введення, які містять елементи управління, зокрема календарі, чекбокси, випадаючі списки та перемикачі. Це забезпечує користувачеві можливість налаштування параметрів збору й аналізу даних у зручній формі.

Для реалізації клієнтської логіки застосовано JavaScript, що відповідає за надсилання асинхронних HTTP-запитів (AJAX), оновлення елементів інтерфейсу та обробку відповідей сервера у форматі JSON. Такий підхід дозволяє реалізувати безперервну взаємодію користувача з системою без необхідності повного перезавантаження сторінки. Також у межах представлення реалізовано шаблон текстового звіту (`report.txt.j2`), який генерується автоматично на основі отриманих результатів, що робить систему адаптивною до потреб кінцевого користувача в частині відображення аналітичних даних.

3.4.2. Взаємодія між компонентами

У системі MLBit взаємодія між компонентами забезпечує чіткий розподіл обов'язків, що дозволяє ефективно обробляти запити користувачів і повертати результати у зручному форматі. Кожен компонент виконує свою роль, від обробки вхідних даних до генерації звітів, що гарантує стабільність і продуктивність роботи системи.

Важливою частиною взаємодії між компонентами є синхронізація між контролером і моделлю, яка дозволяє оперативно реагувати на запити, а також адаптація представлення до потреб користувача, що полегшує доступ до аналітичних результатів. Ця взаємодія побудована з урахуванням модульності, що сприяє легкому розширенню функціональності системи в майбутньому. Описати взаємодію між компонентами можна наступним прикладом:

- 1.Клієнт надсилає POST-запит до `/api/collect` із параметрами.
- 2.Контролер (`collect_api`) валідує дані та викликає модель (`fetch_data`, `process_data`).
- 3.Модель обробляє дані (збір, очищення, трансформація) і повертає `DataFrame` та звіт.
- 4.Контролер формує ZIP-архів і повертає JSON із `task_id` або файл.
- 5.Шаблон відображає статус або посилання для завантаження.

3.4.3. Реальний приклад

Клієнт: користувач відкриває сторінку `/collect` і вводить параметри: проміжок дат з 01.05.2023 по 01.05.2024, джерело даних Binance, параметри ознаки SMA з періодом 20, поріг очищення Z-score 3.0, цільову змінну з двома класами (-1, 1), нормалізацію `StandardScaler`, формат експорту CSV і архівацію у ZIP. Після цього він надсилає POST-запит до API за адресою `/api/collect`.

Контролер (`app.py`): отримує запит, перевіряє коректність і повноту параметрів. Створює унікальний ідентифікатор завдання `task_id`, викликає функцію `fetch_data` для отримання сирих даних, а потім – `process_data` для їх обробки.

Модель: функція `fetch_data` збирає історичні дані про ціну біткоїна з Binance, уніфікує їх у єдиний `DataFrame`. Функція `process_data` обчислює ковзну середню (SMA) з періодом 20, фільтрує аномалії за методом Z-score з порогом 3.0, формує цільову змінну з двох класів (-1, 1) для прогнозування, а також нормалізує дані за допомогою `StandardScaler`.

Контролер (`app.py`): після обробки контролер зберігає результати у файл `processed_data.csv`, пакує його в ZIP-архів і повертає JSON-відповідь із `task_id` та посиланням для завантаження `download_url`.

Шаблон: після завершення обробки і формування архіву користувач отримує JSON з `task_id` і посиланням для завантаження. Завантаження архіву здійснюється через маршрут `/download/<task_id>`, де клієнт отримує ZIP-файл з CSV.

Діаграму, на якій зображено взаємодію між модулями системи та користувачем, можна побачити на рис. 3.1.

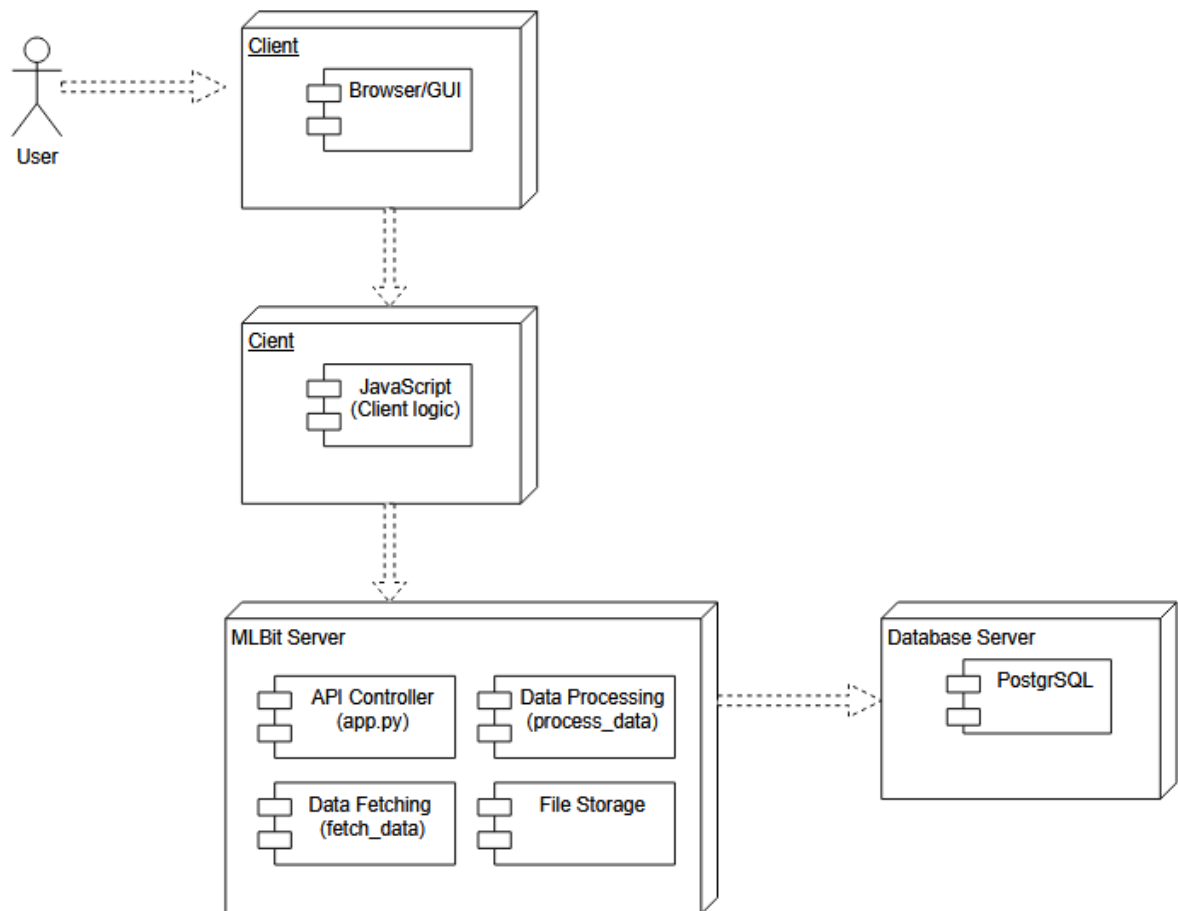


Рис. 3.1. Діаграма модулів системи

Вебзастосунок MLBit реалізований за клієнт-серверною архітектурою, що складається з трьох основних модулів: вебсервер, вебклієнт і модуль роботи з базою даних. Ця архітектура забезпечує ефективну взаємодію користувача із системою та централізоване управління процесами збору, очищення і підготовки даних. Опис нижче базується на описаному прикладі реальної взаємодії.

Вебсервер побудований на Flask, приймає HTTP-запити, зокрема POST-запити до ендпоінта `/api/collect`, проводить валідацію параметрів (наприклад, діапазону дат і джерел даних) та формує відповіді у форматі JSON або CSV. Він працює з базою даних для збереження метаданих завдань, таких як `task_id` і статус обробки, забезпечує хешування паролів користувачів (bcrypt), аутентифікацію та авторизацію для доступу до захищених ресурсів. Крім того, сервер координує асинхронні процеси, генерує тимчасові CSV-файли та пакує їх у ZIP-архіви для завантаження.

Вебклієнт реалізований через HTML-форми з Jinja2 та підтримкою JavaScript для асинхронної взаємодії. Він дозволяє ML-досліднику вибирати період даних, джерела (Binance, Coinbase, Kraken), налаштовувати ознаки (SMA(20)), методи очищення (Z-score 3.0) та нормалізації (StandardScaler), а також ініціювати експорт результатів у CSV з архівацією в ZIP. Вебклієнт відображає форму для введення параметрів, слідкує за статусом завдань і надає посилання для завантаження оброблених даних.

3.4.4. Взаємодія з СУБД

В проєкті MLBit полягає у визначенні та використанні моделей даних за допомогою SQLAlchemy ORM. Наразі система визначає 4 основні моделі для зберігання ринкових, інтерпольованих та кастомізованих факторних та коричтувацьких даних (MarketData, IMarketData, CustomFactors, Users).

Модель Users призначена для базового управління доступом у MLBit. Поле `user_id` – унікальний автоматично згенерований ідентифікатор. `login`

використовується для входу, password – хеш пароля для безпеки. Поле api_limit задає максимальну кількість операцій зі збору та обробки даних за день, контролюючи навантаження, email для відновлення паролю (табл. 3.2).

Таблиця 3.2

Поля моделі Users

Назва поля	Тип (SQLAlchemy)	Опис поля
user_id	Integer, Primary Key	Унікальний ідентифікатор, автогенерація

Продовження табл. 3.2

login	String, Unique, Not Null	Логін користувача
password	String, Not Null	Хеш пароля
api_limit	Integer, Not Null, Default=10	Денний ліміт на кількість API-запитів
email	String, Unique, Not Null	Пошта користувача

Модель MarketData акумулює щоденні ринкові дані Bitcoin, унікально ідентифікуючи записи полем date. Включає OHLCV з бірж Binance, Coinbase, Kraken для порівняльного аналізу, календарні ознаки (us_holiday, day_of_week), диференціали (diff_...) та середні ціни (avg_price_...). Містить блокчейн-метрики, технічні індикатори (sma_20, rsi_14, atr_14, adx_14) та макроекономічні показники (sp500_close, vix_close, us10y, usd_index, gold, oil). Служить джерелом для інтерполяції, фільтрації аномалій та складних ознак (табл. 3.3).

Таблиця 3.3

Поля моделі MarketData

Назва поля	Тип (SQLAlchemy)	Опис поля
date	Date	Дата (первинний ключ)

binance_open, _high, _low, _close, _volume	Float	OHLCV дані Binance
coinbase_open, ...	Float	OHLCV дані Coinbase
kraken_open, ...	Float	OHLCV дані Kraken
us_holiday	Boolean	Свято в США
day_of_week	Integer	День тижня
diff_*	Float	Різниця показників між біржами

Продовження табл. 3.3

avg_price_*	Float	Середні ціни між біржами
hashrate, transactions, unique_addresses, ...	Float/Integer	Метрики Bitcoin блокчейну
sma_20, rsi_14, atr_14, adx_14	Float	Технічні індикатори
sp500_close, vix_close, us10y, usd_index, gold,	Float	Макроекономічні індикатори

Модель IMarketData відображає структуру MarketData з тими ж ринковими, блокчейн- та макроекономічними показниками, але містить інтерпольовані дані. Окрема таблиця MarketData зберігає "сирі" дані для аудиту, надаючи IMarketData готовий набір для аналізу. Що дозволяє користувачу обирати данні у потрібному форматі (табл. 3.4).

Таблиця 3.4

Поля моделі IMarketData

Назва поля	Тип (SQLAlchemy)	Опис поля
date	Date	Дата (первинний ключ)
binance_open, _high, _low, _close, _volume	Float	OHLCV дані Binance
coinbase_open, ...	Float	OHLCV дані
kraken_open, ...	Float	OHLCV дані Kraken

us_holiday	Boolean	Свято в США
day_of_week	Integer	День тижня

Продовження табл. 3.4

diff_*	Float	Різниця показників між біржами
avg_price_*	Float	Середні ціни між біржами
hashrate, transactions, unique_addresses, ...	Float/Integer	Метрики Bitcoin блокчейну
sma_20, rsi_14, atr_14, adx_14	Float	Технічні індикатори
sp500_close, vix_close, us10y, usd_index, gold, oil	Float	Макроекономічні індикатори

Модель CustomFactors агрегує показники для прогнозування в MLBit. Включає волатильність бірж (Binance, Coinbase, Kraken), співвідношення обсягів до цін, різниці між біржами для аномалій. Враховує кореляції Bitcoin із S&P 500, доларом, VIX, золотом, нафтою, блокчейн-даними, технічними індикаторами (RSI, ADX, SMA) та календарними факторами (дні після свят). Створює багатофакторний набір для точних моделей (табл. 3.5).

Таблиця 3.5

Поля моделі CustomFactors

Назва поля	Тип (SQLAlchemy)	Опис поля
volatility_3d/7d/14d	Float	Волатильність за різні періоди
cross_exchange_volatility_diff_*	Float	Різниця волатильності між біржами
volume_to_price_ratio_*	Float	Відношення обсягу до ціни

Продовження табл. 3.5

price_spread_ratio_*	Float	Відношення різниці цін між біржами
bitcoin_sp500_correlation_*	Float	Кореляція Bitcoin з індексом S&P 500
transaction_value_to_volume_ratio	Float	Відношення вартості транзакцій до їх обсягу
hashrate_to_difficulty_ratio	Float	Відношення хешрейту до складності майнінгу
rsi_adx_combo, sma_20_to_price_ratio, adx_strength	Float	Комбіновані технічні індикатори
daily_price_change_pct	Float	Щоденна зміна ціни у відсотках

Для забезпечення швидкого доступу до великих обсягів даних у MLBit реалізовано індексацію ключових полів, таких як date у таблицях MarketData та IMarketData. Це дозволяє значно прискорити запити під час аналізу історичних даних, що є критично важливим для аналітиків і дослідників ринку криптовалют. Крім того, система підтримує періодичне оновлення даних через автоматизовані ETL-процеси, що гарантує актуальність інформації для прогнозних моделей. ETL-процеси також включають автоматичну логіку для обробки помилок, таких як тимчасова недоступність API бірж, із повторними спробами через фіксовані проміжки часу, забезпечуючи безперервність оновлення навіть у разі мережових збоїв. Такий підхід дозволяє системі адаптуватися до змін у джерелах даних, таких як оновлення форматів від Blockchain.com чи FRED, зберігаючи цілісність і актуальність наборів для машинного навчання.

Схему архітектури розробленої бази даних зображено на рис. 3.2.

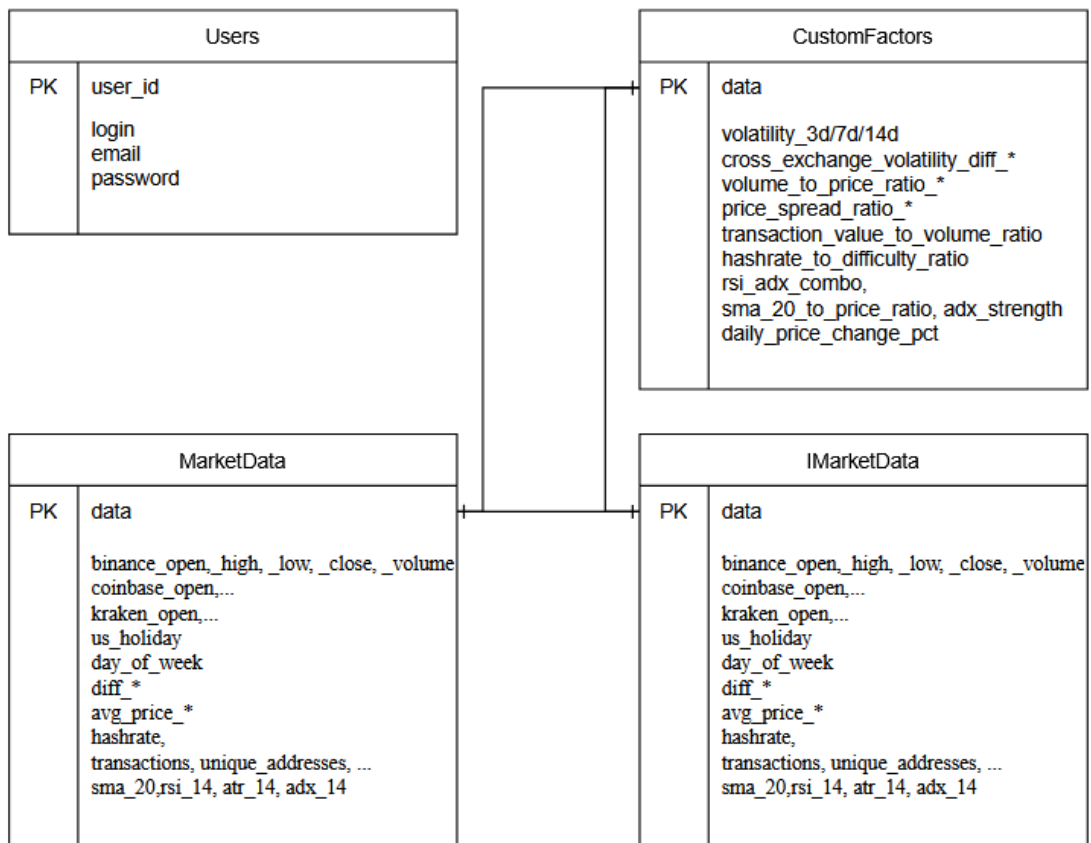


Рис. 3.2. Схема бази даних вебзастосунок

3.5. Особливості реалізації

Вебзастосунок MLBit розроблено з акцентом на модульність, що забезпечує гнучкість і легкість у підтримці коду. Реалізація базується на фреймворку Flask, який дозволяє ефективно обробляти HTTP-запити та координувати роботу всіх компонентів системи. Основна логіка розподілена між трьома ключовими модулями: вебсервером (app.py), вебклієнтом (HTML-шаблони з JavaScript) і модулем для роботи з базою даних (db.py, models.py). Це відповідає клієнт-серверній архітектурі, де кожен модуль має чітко визначені обов'язки, що сприяє масштабованості та зрозумілості коду.

Діаграму класів зображено на рис. 3.3.

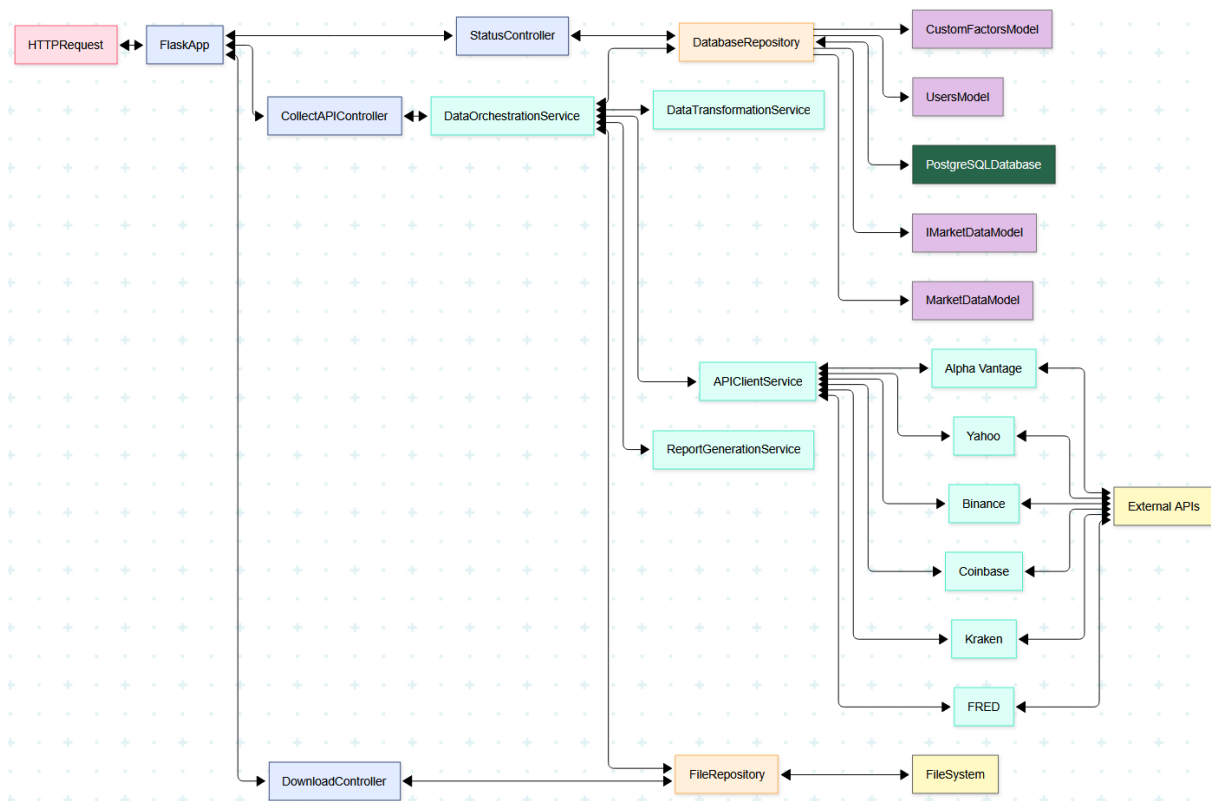


Рис. 3.3. Діаграма класів серверної частини

3.5.1. Використання *Flask* і *MTV-підходу*

Flask є легковаговим вебфреймворком для Python, який використовується в проєкті MLBit для створення серверної частини вебзастосунку, призначеного для агрегації та обробки даних біткоїна. Цей фреймворк забезпечує гнучкість у розробці завдяки мінімалістичному підходу, дозволяючи розробникам самостійно визначати структуру додатку. Flask опирається на архітектуру Model-Template-View (MTV), де модель відповідає за логіку роботи з даними, шаблони – за відображення, а контролери – за обробку запитів [20]. У контексті MLBit Flask слугує основою для маршрутизації запитів, таких як `/api/collect` для ініціювання обробки даних, і забезпечує інтеграцію з модулями для роботи з базою даних і зовнішніми API.

Автентифікація в MLBit реалізується для забезпечення безпечного доступу користувачів, таких як ML-дослідники, до функціоналу обробки даних (наприклад, для періодів із 01.05.2023 до 01.05.2024 із джерел Binance, Coinbase, Kraken, FRED, Alpha Vantage, Yahoo Finance). На відміну від готових рішень, Flask не має вбудованих пакетів для аутентифікації,

тому реалізація базується на бібліотеках, таких як Flask-Login і Werkzeug [21], а також власних механізмах шифрування. Процес включає налаштування середовища, реєстрацію користувачів, перевірку їхніх даних і створення об'єктів автентифікації, що слугують для подальшої авторизації доступу до ресурсів, таких як завантаження результатів у ZIP-архів.

Першим кроком є налаштування робочого середовища. Файл `.env` конфігурується для підключення до PostgreSQL [22], де вказуються параметри хоста, порту, імені бази даних, користувача й пароля. Також визначається секретний ключ (`SECRET_KEY`) для шифрування сесій, який завантажується через бібліотеку `python-dotenv`. Ця конфігурація дозволяє інтегрувати базу даних із моделлю `Users`, де зберігатимуться дані автентифікації, включаючи `user_id`, `login`, `password` (у хешованому вигляді) і `api_limit` (ліміт запитів, наприклад, 10 на день).

Наступний етап передбачає реєстрацію нового користувача. Користувач відкриває сторінку `/register`, заповнює форму з полями `login` і `password`, після чого дані відправляються через POST-запит. Контролер (`app.py`) перевіряє унікальність логіна, звертаючись до таблиці `Users` через SQLAlchemy. Якщо логін вільний, пароль хешується за допомогою `werkzeug.security.generate_password_hash`, а об'єкт користувача з унікальним `user_id` зберігається в базі даних. Успішна операція підтверджується поверненням JSON-повідомлення.

Для входу в систему користувач переходить на сторінку `/login` і вводить `login` та `password`. Контролер приймає POST-запит, звертається до моделі для пошуку відповідного запису в таблиці `Users` і порівнює введений пароль із хешованим значенням за допомогою `check_password_hash`. Якщо збіг виявлено, процес завершується успіхом, інакше повертається помилка. Ця перевірка забезпечує захист від несанкціонованого доступу до даних, таких як OHLCV із Binance.

Після успішної автентифікації Flask-Login [23] створює сесію для користувача, зберігаючи об'єкт із `user_id` і `login` у контексті запиту. Сесія налаштовується через параметр `PERMANENT_SESSION_LIFETIME`, який

визначає час її дії (24 години). Також перевіряється поле `api_limit` у базі даних: якщо ліміт (10 запитів на день) не перевищено, користувачу надається доступ. Це дозволяє контролювати навантаження на систему під час обробки великих обсягів даних.

Доступ до захищених ендпоінтів, таких як `/api/collect` для запуску обробки даних (SMA(20), Z-score 3.0, StandardScaler), обмежується декоратором `@login_required` від Flask-Login. Перед наданням доступу контролер перевіряє статус сесії та наявність прав. Якщо автентифікація пройшла успішно, користувач отримує можливість ініціювати обробку, а результати (наприклад, `processed_data.csv`) стають доступними для завантаження через `/api/download`.

Після автентифікації та доступу до обробки даних система виконує запити до зовнішніх API, обробляє дані й генерує ZIP-архів із результатами. Користувач може завантажити цей архів через `/api/download/<task_id>`, якщо сесія активна й `api_limit` не вичерпано. У разі перевищення ліміту або закінчення терміну сесії повертається JSON із повідомленням про помилку, що забезпечує прозоре інформування.

Процес автентифікації представлений компонентною діаграмою, що відображає взаємодію між вебклієнтом, вебсервером, моделлю, базою даних і модулем обробки даних. Діаграма показує ключові етапи: від заповнення форм до доступу до ресурсів, включаючи хешування паролів, перевірку сесій і генерацію результатів (рис. 3.4).

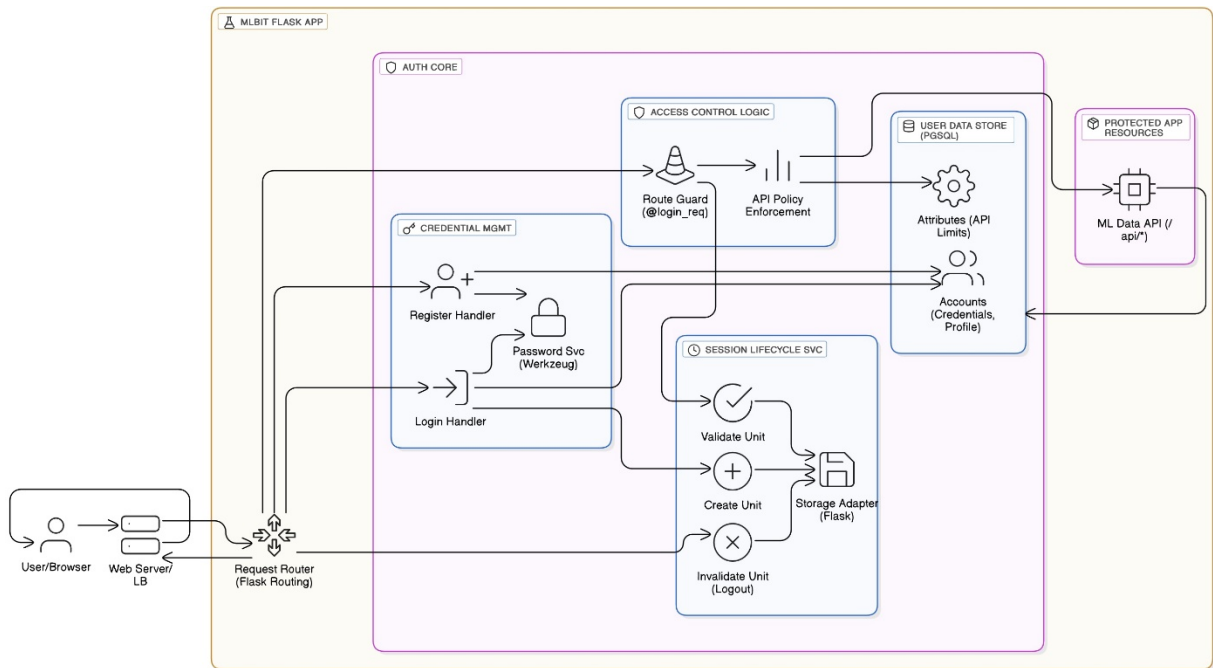


Рис. 3.4. Схема архітектури автентифікації Flask-Login

На клієнтській стороні вебзастосунку MLBit розроблено набір модулів графічного відображення, які забезпечують інтерактивну взаємодію з системою для ML-дослідників. Ці модулі реалізовано через HTML-шаблони, відрендерені за допомогою Jinja2, із підтримкою базового JavaScript для асинхронних дій. Загалом створено 4 ключові сторінки, кожна з яких виконує специфічну функцію, пов'язану з автентифікацією, управлінням даними, отриманням інформації про проєкт та переглядом результатів обробки, що відповідає завданням прогнозування ціни біткоїна.

Було розроблено 6 графічних інтерфейсів:

- 1) сторінка логіну;
- 2) сторінка реєстрації;
- 3) сторінка відновлення паролю;
- 4) головна сторінка;
- 5) сторінка з інформацією стосовно вебзастосунку;
- 6) сторінка підготовки і отримання датасету.

3.5.2. Очищення та нормалізація даних

Етап очищення та нормалізації в системі MLBit розроблено для підготовки сирих даних, отриманих із джерел, таких як Binance і Yahoo

Finance, до аналізу й моделювання. Цей процес включає виявлення аномалій, заповнення пропусків, трансформацію даних і приведення їх до формату, придатного для машинного навчання. Кроки застосовуються послідовно залежно від параметрів, визначених користувачем (наприклад, для обробки даних біткоїна за період із 01.05.2023 до 01.05.2024), що забезпечує гнучкість і адаптивність системи.

Перехресна валідація в MLBit зосереджена на перевірці узгодженості цін між різними джерелами, зокрема між Binance і Coinbase, для виявлення потенційних помилок у даних. Наприклад, якщо різниця цін закриття біткоїна перевищує 2%, запис позначається як "Підозріла_Ціна", визначаючи це на основі порівняння різниці між цінами на двох біржах відносно базової ціни, вираженої у відсотках.

Такий підхід допомагає виявити аномалії, спричинені збоями API або ринковими розбіжностями, і дозволяє логувати ці записи для подальшого аналізу. За потреби система може скоригувати ціни, використовуючи середнє значення між джерелами, або додати булеву ознаку, що сигналізує про розбіжність, для використання в моделях.

Виявлення аномалій у MLBit здійснюється за допомогою кількох методів, щоб усунути значення, які можуть спотворити результати аналізу. Метод Z-score оцінює відхилення від середнього значення, позначаючи як аномалії ті дані, що значно перевищують встановлений поріг, тоді як міжквартильний розмах (IQR) визначає викиди через межі, обчислені на основі першого та третього квартилів із додатковим коефіцієнтом.

Додатково застосовуються логічні перевірки: ціни чи обсяги торгів, менші або рівні нулю, вважаються помилковими й видаляються. Аномалії можуть бути замінені медіаною або граничними значеннями, що дозволяє зберегти цілісність даних для подальшого аналізу, наприклад, при розрахунку SMA(20).

Пропуски в даних, таких як ціни закриття чи обсяги торгів із Binance, обробляються методом лінійної інтерполяції для забезпечення безперервності часових рядів. У бібліотеці pandas використовується функція `interpolate` із параметром `method='linear'`, яка заповнює пропуски,

обчислюючи середнє між сусідніми значеннями, що є ефективним для фінансових даних за період із 01.05.2023 до 01.05.2024.

Для більших пропусків передбачені альтернативні методи, такі як заповнення попереднім значенням (`method='ffill'`) або наступним (`method='bfill'`), залежно від характеру даних. Це дозволяє уникнути втрати важливих записів і зберегти динаміку цін для подальшого прогнозування.

Трансформація даних у MLBit спрямована на зменшення впливу викидів і підготовку до моделювання. Вінзоризація застосовується для обмеження екстремальних значень, замінюючи їх граничними перцентилями (наприклад, 5-й і 95-й), що реалізується через `scipy.stats.mstats.winsorize`, зберігаючи структуру даних без видалення записів.

Додатково використовується трансформація Йео-Джонсона через `PowerTransformer` із `scikit-learn`, яка стабілізує дисперсію й наближає розподіл ознак, таких як обсяги торгів, до нормального. Це покращує продуктивність моделей машинного навчання, особливо тих, що чутливі до розподілу, таких як лінійна регресія.

Нормалізація в MLBit забезпечує єдиний масштаб для числових ознак, що є важливим для моделей машинного навчання. `MinMaxScaler` приводить дані до стандартного діапазону, базуючись на мінімальному та максимальному значеннях, а `StandardScaler` адаптує ознаки до нульового середнього та одиничного стандартного відхилення, що особливо корисно для алгоритмів типу SVM.

Для даних із викидами застосовується `RobustScaler`, який базується на медіані та міжквартильному розмаху, гарантуючи стійкість до аномалій, наприклад, у обсягах торгів біткоїна. Вибір методу залежить від характеру даних і потреб моделі, що допомагає покращити точність прогнозів.

Категоріальні ознаки в MLBit кодуються для використання в моделях машинного навчання, які працюють із числовими даними. Бінарна ознака "US_holidays", що визначається за допомогою пакету `holidays`, обробляється через `LabelEncoder`, який присвоює значення 0 і 1, забезпечуючи просте представлення для моделей.

Ознака "day_of_week" (понеділок – неділя) кодуються за допомогою OneHotEncoder, що створює бінарні стовпці для кожного дня (наприклад, day_of_week_0, day_of_week_1, ..., day_of_week_6), уникаючи помилкової впорядкованості. Це дозволяє коректно враховувати вплив календарних факторів на ціни біткоїна в моделях, таких як логістична регресія.

Вебзастосунок MLBit забезпечує гнучкість у виборі методів очищення та нормалізації, дозволяючи користувачу налаштовувати параметри, такі як поріг Z-score (3.0 чи 4.0) або метод нормалізації (MinMaxScaler чи StandardScaler). Такий підхід дозволяє оптимізувати підготовку даних залежно від цілей аналізу, зменшуючи вплив аномалій і пропусків, а також забезпечуючи сумісність із різними моделями машинного навчання. Користувач може експериментувати з методами, щоб досягти найкращих результатів прогнозування.

3.5.3. Підготовка даних для машинного навчання (ML)

Підготовка даних для машинного навчання в системі MLBit є завершальним етапом після очищення, трансформації та нормалізації, спрямованим на створення умов для ефективного прогнозування різних факторів, в тому числі і ціни біткоїна за певний період. Цей процес включає визначення цільової змінної, яка відображає те, що модель має передбачити, а також усунення за необхідності незбалансованості класів, що часто виникає в фінансових даних.

Система дозволяє адаптувати ці етапи до конкретних завдань, таких як класифікація сигналів купівлі/продажу чи регресія для точного прогнозування цін, базуючись на ознаках із джерел, як-от Binance, Coinbase, Kraken, FRED, Alpha Vantage, Yahoo Finance, що було підготовлено на попередніх кроках.

Цільова змінна в MLBit є ключовим елементом, який визначає мету моделі, наприклад, прогнозування напрямку руху ціни біткоїна. Для класифікаційних завдань, таких як сигнали купівлі чи продажу, розраховується зміна ціни за обраний горизонт прогнозування (1 день), де

значення класифікуються як "Різке зростання" ($>2\%$), "Різке падіння" ($<-2\%$), або "Боковий рух" (між -2% і $+2\%$), що кодується як -1 , 0 , 1 .

Для регресійних завдань цільовою змінною може бути сама майбутня ціна або відсоткова зміна ціни. Вибір горизонту є критичним і залежить від стратегії (короткострокове чи довгострокове прогнозування), а зсув даних у часі (щоб уникнути витоку даних) забезпечується зміщенням цільової змінної відносно ознак. Пороги для класифікації обираються на основі аналізу історичних даних, що підвищує точність моделі.

Незбалансованість класів часто виникає в фінансових даних, де періоди значних коливань цін біткоїна трапляються рідше, ніж стабільні періоди, що може призвести до упередження моделі на користь мажоритарного класу. У MLBit балансування застосовується виключно до тренувального набору після розподілу на тренувальну, валідаційну та тестову вибірки, щоб уникнути впливу тестових даних на навчання. Для цього використовується метод, такий як SMOTE, який генерує синтетичні приклади для міноритарного класу, обираючи найближчих сусідів і створюючи нові точки як комбінацію існуючих зразків, що допомагає зменшити ризик перенавчання. Альтернативою є RandomUnderSampler, який випадково видаляє приклади з мажоритарного класу до збалансованого співвідношення, хоча це може призвести до втрати інформації. Вибір методу залежить від специфіки даних і моделі.

SMOTE в MLBit реалізується через бібліотеку imbalanced-learn із налаштуванням параметрів, таких як `k_neighbors` і `sampling_strategy`, що дозволяє контролювати кількість синтетичних прикладів. Цей метод ефективний, коли міноритарні класи, як-от різке зростання цін, є критичними для прогнозування, але може створювати шум, якщо межі класів розмиті.

Генерація цільової змінної та балансування класів є завершальними етапами підготовки даних у MLBit перед поданням їх у моделі машинного навчання. Ці кроки забезпечують, що модель отримує коректно визначені цілі та збалансований набір для навчання, що критично для точного прогнозування цін біткоїна.

Система дозволяє користувачу налаштувати горизонт прогнозування, пороги класифікації та методи балансування, адаптуючи процес до конкретних потреб. Це сприяє створенню надійних моделей, здатних розпізнавати як типові, так і рідкісні ринкові події, такі як різкі зміни цін.

6.Висновки

Вебзастосунок MLBit розроблено як спеціалізований інструмент для ML-дослідників, що забезпечує агрегацію, обробку та аналіз даних для прогнозування ціни біткоїна на основі різних джерел, таких як Binance, Coinbase, Kraken, FRED, Alpha Vantage, Yahoo Finance. Система побудована на клієнт-серверній архітектурі з використанням адаптованого підходу Model-Template-View (MTV), реалізованого через Flask, що забезпечує чітке розділення логіки обробки, відображення та взаємодії з користувачем.

Архітектура додатку ретельно спроектована для забезпечення стабільності й масштабованості, а гнучка система автентифікації через Flask-Login і Werkzeug із хешуванням паролів (наприклад, bcrypt) гарантує безпечний доступ до функціоналу, включаючи обробку даних (SMA(20), Z-score 3.0, StandardScaler) та експорт результатів у ZIP-архіви.

MLBit охоплює всі необхідні етапи підготовки даних для машинного навчання: від збору й очищення до нормалізації та генерації цільових змінних, що дозволяє користувачам ефективно працювати з фінансовими даними. Модулі графічного відображення, такі як сторінки /login, /register і /collect, забезпечують зручну взаємодію, а асинхронна обробка великих обсягів даних (наприклад, OHLCV із Binance) і логування помилок підвищують надійність системи.

Додаткові можливості, такі як балансування класів (SMOTE, RandomUnderSampler) і гнучкість у виборі методів обробки, дозволяють адаптувати систему до різних сценаріїв прогнозування. Це робить MLBit потужним інструментом для оптимізації аналітичних процесів, підвищення точності прогнозів і забезпечення ефективної роботи з ринковими даними.

4.1. Аналіз реалізованого вебзастосунок

Вебзастосунок MLBit розроблено як спеціалізований інструмент для збору, обробки та підготовки фінансових даних, зокрема ринкових даних біткоїна, для використання в моделях машинного навчання. Система дозволяє користувачам, таким як ML-дослідники, автоматизувати збір даних із криптовалютних бірж (Binance, Coinbase, Kraken) і блокчейну (через API Blockchain.com), а також інтегрувати макроекономічні показники для комплексного аналізу за період із 01.01.2020 до сьогодні. Основна функція включає налаштування конвеєрів обробки з методами, такими як Z-score 3.0, SMA(20) і StandardScaler, а також експорт результатів у ZIP-архівах із CSV або JSON файлами.

Додатково реалізовані інструменти інженерії ознак (RSI, різниці цін), генерація цільових змінних (наприклад, напрямки ціни: -1, 0, 1) і балансування класів (SMOTE, RandomUnderSampler), що забезпечує гнучкість для аналітичних задач. Асинхронна обробка завдань і генерація звітів про якість даних підвищують ефективність роботи з великими наборами даних, роблячи MLBit цінним для прогнозування ринкових трендів.

Дані в MLBit зберігаються в реляційній базі даних PostgreSQL, де моделі, визначені в models.py через SQLAlchemy (db.py), включають Users (з полем api_limit, 10 запитів на день), MarketData, IMarketData і CustomFactors. Оптимізовані запити забезпечують швидкий доступ до метаданих, таких як статус завдань, що підвищує продуктивність у хмарному середовищі Render.

Безпека реалізовано через шифрування паролів за допомогою werkzeug.security (bcrypt) і рекомендацію використання HTTPS для захисту передачі даних між клієнтом і сервером у хмарному середовищі. Механізми автентифікації через Flask-Login із @login_required контролюють доступ до ендпоінтів, таких як /collect, забезпечуючи

конфіденційність оброблених даних і обмеження ресурсів для користувачів.

Вебзастосунок MLBit розгорнуто у хмарному середовищі Render, що забезпечує простий та керований деплой без необхідності власної інфраструктури. Запуск здійснюється через Gunicorn, що відповідає вимогам production-середовища. Для зберігання та обробки даних використовується хмарна база даних PostgreSQL, підключення до якої здійснюється через змінні оточення. Файл requirements.txt містить усі необхідні залежності, зокрема Flask, SQLAlchemy, Gunicorn та драйвер для PostgreSQL, що спрощує розгортання та підтримку проєкту.

Архітектура базується на клієнт-серверній моделі з адаптованою схемою Model-Template-View (MTV), де Flask керує логікою, Jinja2 рендерить шаблони (наприклад, collect.html), а SQLAlchemy забезпечує доступ до даних. Такий підхід сприяє модульності, дозволяючи легко додавати нові методи (наприклад, нові індекси, як RSI) або розширювати систему для хмарного розгортання в майбутньому. Додатково, вебсервіс містить можливість створення списку товарів. Користувач може створювати новий товар та задавати йому певні параметри.

Функціональні вимоги MLBit виконано: реалізований конвеєр збору даних із бірж і блокчейну, модулі обробки (очищення аномалій через Z-score і IQR, лінійна інтерполяція), інженерії ознак (SMA, RSI, ADX) і експорту результатів через /download/<task_id>. Нефункціональні вимоги, такі як модульність і конфігурованість, забезпечені структурою коду (наприклад, api_clients.py, processing.py) і вебформою на /collect.

Обробка даних включає крос-валідацію цін (позначення "Підозрілих цін" при різниці >2%), трансформацію (Вінзоризація, Йео-Джонсон) і нормалізацію (MinMaxScaler, RobustScaler), а також балансування класів для тренувальних наборів. Це робить дані придатними для моделей, таких як SVM, і відповідає поставленим цілям аналітичних досліджень. Важливою складовою розробленого вебсервісу є його інтерфейс. Вебсервіс має практичний дизайн, який було протестовано. Дизайн інтерфейсу було розроблено орієнтованим на швидку навігацію.

Інтерфейс MLBit розроблено як функціонально орієнтований, із сторінками `main.html` (головна з доступом до функцій), `about.html` (інформація про проєкт) і `collect.html` (налаштування обробки даних), що забезпечує інтерактивну взаємодію. Сторінка `/collect` дозволяє задавати параметри (джерела, методи, період) і відстежувати статус завдань, що спрощує роботу з системою.

Дизайн акцентує увагу на чіткому поданні опцій і результатів, використовуючи Jinja2 для шаблонів і JavaScript для асинхронних оновлень. Це забезпечує зручність для технічних користувачів, які інтегрують MLBit у свої аналітичні процеси, із можливістю подальшого вдосконалення візуалізації.

MLBit є спеціалізованим рішенням для збору й підготовки фінансових даних для машинного навчання, що пропонує гнучкі інструменти для аналітиків і дослідників. Система забезпечує автоматизацію процесів, доступ до різноманітних джерел і підтримку інженерії ознак, що робить її цінною для прогнозування ринкових тенденцій.

4.2.Тестування розробленого вебзастосунку

4.2.1. *Процес тестування вебзастосунку*

Тестування вебзастосунку MLBit було проведено для виявлення дефектів, підтвердження відповідності функціональним і нефункціональним вимогам та забезпечення стабільної роботи системи. Комплексний підхід включав модульне, performance, інтеграційне, API-, end-to-end-, валідаційне та безпекове тестування, охоплюючи всі аспекти від окремих функцій до взаємодії з користувачем. Структура тестів організовано в папці `tests` як Python-пакет (завдяки `__init__.py`), із файлами, такими як `test_api.py`, `test_data_transformation.py` і `run_tests.py`, що свідчить про системний підхід до забезпечення якості.

Тестування інтегровано в цикл розробки з частковою автоматизацією за допомогою `pytest`, що забезпечило швидкий зворотний зв'язок. Тестові набори, оновлені відповідно до нових функцій, і структура з `__pytest__`

для прискорення завантаження байт-коду підтвердили високу якість коду. Скрипт `run_tests.py` (680 байт у `tests`, 2338 байт у корені) слугував для запуску тестів, а файли, як-от `test_request.json`, використовувалися для валідних прикладів запитів, що підкреслює деталізований підхід до аналітичних задач ML-дослідників.

4.2.2. Модульне та валідаційне тестування

Модульне тестування MLBit було проведено для перевірки коректності роботи окремих компонентів. У `test_api.py` протестовано функції `fetch_binance_data` і `fetch_blockchain_data` із `api_clients.py` на коректність запитів до API Binance і Blockchain.com. У `test_data_transformation.py` перевірено методи `transformer.py` (SMA(20), StandardScaler, Йео-Джонсон) і `processing.py` (обробка аномалій Z-score 3.0), а в `test_exporter.py` – експорт у CSV та Zip формування.

Валідаційне тестування підтвердило точність і консистентність даних із бірж і Blockchain.com, використовуючи Pandera [24] для схем DataFrame і Great Expectations [25] для статистики. У `test_validator.py` перевірено валідність параметрів API-запитів і типів даних, а в `test_utils.py` – допоміжні функції, як-от форматування дат. Успішно оброблено граничні випадки (нульові ціни) і підтверджено готовність даних для моделей, таких як SVM.

Модульне тестування через `test_data_transformation.py` і `test_validator.py` виявило кілька помилок у крайових випадках (наприклад, при обробці нульових значень), які були оперативно виправлені, що підвищило надійність методів трансформації.

4.2.3. Інтеграційне та end-to-end тестування

Інтеграційне тестування MLBit було проведено для перевірки взаємодії між `app.py` (ендпоінт `/api/collect`), `api_clients.py`, `processing.py` і `transformer.py`, що забезпечило коректний ланцюжок обробки даних. У `test_integration.py` протестовано повний цикл: від запиту до ZIP-архіву й генерації звіту, а також взаємодію з тестовою базою PostgreSQL через SQLAlchemy (`models.py`, `db.py`).

Е2Е-тестування [26] імітувало реальні сценарії: перехід із main.html на collect.html, заповнення форми і запуск із отриманням task_id, а також API-виклики (/api/collect, /api/status/<task_id>, /api/download/<task_id>). Використовувалися Selenium і requests. В результаті було підтверджено успішно інтеграцію компонентів при хмарному розгортанні.

4.2.4. Тестування API, продуктивності та безпеки

Тестування API MLBit включало перевірку ендпоінтів /api/collect, /api/status/<task_id> і /api/download/<task_id> на коректність відповідей (JSON, ZIP) і обробку помилок, виконано з requests і Postman. У test_performance.py виміряно час відповіді /api/collect (менше ніж 3 секунди) і стабільність при обробці великих даних з бази даних, використовуючи Locust для імітації навантаження [27].

Безпекове тестування виявило відсутність вразливостей, таких як SQL-ін'єкції (через SQLAlchemy), і перевірило захист автентифікації (Flask-Login) і валідацію даних у /api/collect. Використовувався OWASP ZAP [28], що підтвердило безпеку сесій і захист ключів API, критично важливих для роботи з біржами.

Невеликі недоліки, як-от затримки при обробці великих наборів даних, були ідентифіковані під час тестування продуктивності й можуть бути вирішені шляхом оптимізації асинхронних задач. Зручність використання для ML-дослідників підтверджена тестами, та мануальним тестуванням декількох залучених користувачів.

4.3. Порівняння з аналогами

У розділі 1 проведено детальний аналіз платформ для збору та аналізу фінансових даних криптовалют, таких як CoinGecko, CryptoCompare і Glassnode, із оцінкою їхніх переваг і недоліків. Дослідження виявило ключові обмеження: обмежену гнучкість у формуванні багатофакторних наборів даних (поєднання ринкових даних із бірж, ончейн-метрик із Blockchain.com і макроекономічних показників), відсутність комплексних інструментів попередньої обробки (наприклад, фільтрація аномалій Z-score 3.0 чи лінійна інтерполяція) і недостатню

підготовку даних для машинного навчання (нормалізація StandardScaler, генерація цільових змінних).

Крім того, висока вартість передплати (особливо для Glassnode) і ліміти на API-запити ускладнюють доступ для індивідуальних дослідників і студентів. Ці аспекти підкреслили потребу в доступному й гнучкому рішенні, що враховує специфіку прогнозування ціни біткоїна за період із 01.01.2020 по сьогоднішній день, із використанням різних джерел таких як Binance, Coinbase, Kraken, Blockchain.com і FRED, що стало основою для розробки MLBit.

MLBit усуває ці недоліки, надаючи користувачам повний контроль над формуванням багатofакторних датасетів, що включають ринкові дані з різних бірж (такі як ціни відкриття, закриття, максимуму, мінімуму та обсяги торгів), ключові ончейн-метрики (наприклад, хешрейт, кількість активних адрес, обсяги транзакцій), макроекономічні показники (як-от індекси S&P 500, ціни на золото), а також власноруч розраховані або інженерно створені фактори (наприклад, різні види волатильності, технічні індикатори, часові ознаки типу "день тижня"). На відміну від аналогів, система інтегрує автоматизовану попередню обробку з можливістю вибору методів (Z-score, IQR, MinMaxScaler, RobustScaler) і генерацію цільових змінних (наприклад, для задач класифікації з визначенням напрямку ціни: -1, 0, 1), що спрощує підготовку даних для ML-моделей.

Доступність забезпечується хмарним розгортанням і використанням публічних API без високих бар'єрів, на відміну від платних рішень Glassnode. MLBit пропонує асинхронну обробку й експорт у ZIP-архіви, що перевершує обмеження за обсягом даних у безкоштовних версіях аналогів, роблячи його конкурентоспроможним для аналітиків і дослідників.

Розроблений вебзастосунок MLBit вигідно відрізняється від CoinGecko, CryptoCompare і Glassnode завдяки спеціалізованому підходу до підготовки даних для машинного навчання, що включає кодування категоріальних ознак (One-Hot Encoding) і балансування класів (SMOTE). Така функціональність, відсутня в аналогах, забезпечує унікальну цінність

для прогнозування ринкових тенденцій, адаптованих до потреб користувачів.

4.4.Рекомендації для подальшого вдосконалення

Розроблений вебзастосунок MLBit забезпечує широкий набір інструментів для збору, обробки та підготовки фінансових даних біткоїна для машинного навчання, відповідаючи потребам ML-дослідників. Однак, з огляду на швидкий розвиток технологій у сфері аналізу криптовалют і високу конкуренцію з платформами, такими як CoinGecko і Glassnode, постійне оновлення системи є необхідним для підтримки її актуальності та конкурентоспроможності. Важливо враховувати сучасні тенденції, які формуються на основі новітніх підходів до обробки великих даних, а також зростання попиту на інноваційні рішення серед користувачів.

Враховуючи сучасні тенденції і потенційний зворотній зв'язок від користувачів, сформульовано рекомендації для подальшого вдосконалення MLBit, спрямовані на розширення функціональності, підвищення зручності та масштабованості, які варто врахувати:

- 1.Розширення джерел даних шляхом інтеграції з новими біржами (наприклад, KuCoin [29], Bitfinex) і додавання альтернативних даних, таких як аналіз настроїв із соціальних мереж (X) і макроекономічні показники (NASDAQ), для створення комплексних наборів даних.
- 2.Впровадження інтерактивних дашбордів із візуалізацією даних (графіки цін, SMA(20), RSI) у вебінтерфейсі (collect.html), що полегшить аналіз якості даних і технічних індикаторів.
- 3.Оптимізація продуктивності через паралельні обчислення (з використанням dask [30]) і кешування API-запитів до Binance для прискорення обробки великих обсягів даних.
- 4.Розробка публічного API для MLBit, що дозволить інтегрувати систему з ML-середовищами, такими як Jupyter Notebook і Google Colab, спрощуючи експорт даних і конфігурацій.

5. Поглиблення обробки даних шляхом упровадження методів заповнення пропусків (KNN-imputation), виявлення аномалій (Isolation Forest) і автоматизації генерації ознак (часові лаги, різниці цін).
6. Створення розширеної документації з прикладами використання (наприклад, прогнозування ціни біткоїна) і форуму для спільноти, що підвищить доступність для студентів і дослідників.
7. Додавання підтримки даних про деривативи (ф'ючерси, опціони) і функції автоматичного відбору ознак (на основі Random Forest), що розширить можливості для складного аналізу ринку.

У перспективі MLBit має потенціал еволюціонувати в інтегроване рішення для аналізу ринку криптовалют, що не лише полегшує підготовку даних для машинного навчання, але й пропонує глибокий фінансовий аналіз, доступний як індивідуальним дослідникам, так і професійним аналітикам. Впровадження зазначених удосконалень розширить функціональні можливості системи, сприяючи її інтеграції з різними аналітичними інструментами та адаптації до нових викликів у прогнозуванні. Це сприятиме зміцненню репутації MLBit як гнучкого й універсального ресурсу для спеціалістів у сфері криптовалютного аналізу.

4.5. Висновки

У цьому розділі проведено детальний аналіз вебзастосунку MLBit, включаючи реалізацію всіх функціональних і нефункціональних вимог. Описано ключові можливості, такі як обробка даних і підготовка для машинного навчання, а також аспекти модульності й безпеки.

Проведено порівняння MLBit з аналогами (CoinGecko, CryptoCompare, Glassnode), яке продемонструвало його конкурентну перевагу завдяки гнучкості в обробці даних і спеціалізованим інструментам для ML, що вигідно відрізняє його від існуючих рішень.

Протягом розробки проводилися тестування, що дозволили виправити виявлені помилки й недоліки. Після завершення система пройшла перевірку користувачами та окремі безпекові тести, які

підтвердили виконання всіх вимог. На основі аналізу тенденцій і зворотного зв'язку сформовано рекомендації для вдосконалення, спрямовані на покращення досвіду та функціональності, які успішно враховано.

ВИСНОВКИ

Даний дипломний проєкт присвячений створенню вебзастосунку MLBit для збору, обробки та підготовки фінансових даних біткоїна для машинного навчання. У першому розділі проаналізовано аналоги (CoinGecko, CryptoCompare, Glassnode), виявлено їхні переваги та недоліки, зокрема обмежену гнучкість у формуванні багатфакторних наборів даних і недостатню підготовку даних для ML. На основі цього сформульовано вимоги до MLBit, акцентуючи на автоматизації обробки та доступності для дослідників.

У другому розділі розглянуто потенційні технології розробки, обґрунтовано вибір інструментів: Python із фреймворком Flask, JavaScript для асинхронних запитів [31], бібліотеки Pandas і Scikit-learn для обробки даних, а також базу даних PostgreSQL із SQLAlchemy.

У третьому та четвертому розділах описано реалізацію MLBit, включаючи структуру бази даних (MarketData, CustomFactors), автентифікацію (Flask-Login), обробку (Z-score 3.0, StandardScaler) і експорт у ZIP. Порівняння з аналогами показало конкурентну перевагу MLBit завдяки гнучкості, спеціалізованим ML-інструментам (SMOTE) і модульності.

У третьому та четвертому розділі роботи було проведено аналіз розробленого сервіса для планування та аналізу статистики, структуру бази даних та візуалізацію. Для забезпечення безпеки та обмеженого доступу до даних, була передбачена система авторизації. Порівняння розробленого програмного забезпечення з аналогами показало, що вебсервіс має ключові функціональні можливості, необхідні для планування та відстеження задач, а також надає додатковий набір графіків для візуалізації зібраних даних [32]. Крім того, додаток пропонує нове рішення, для моніторингу стану виконання поставленої задачі. Це дає йому конкурентну перевагу порівняно з іншими подібними рішеннями на ринку.

У результаті створено вебзастосунок MLBit, який забезпечує ефективний збір, обробку й підготовку даних для прогнозування ринкових

трендів, зручний інтерфейс (collect.html) і підтримку аналітичних задач. Система дозволяє налаштовувати параметри обробки, генерувати звіти й експортувати результати, що сприяє продуктивності ML-дослідників. Тестування підтвердило стабільність, безпеку й відповідність вимогам, а рекомендації (інтеграція нових джерел, дашборди) відкривають перспективи для розвитку MLBit як універсального інструменту для аналізу криптовалютного ринку.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

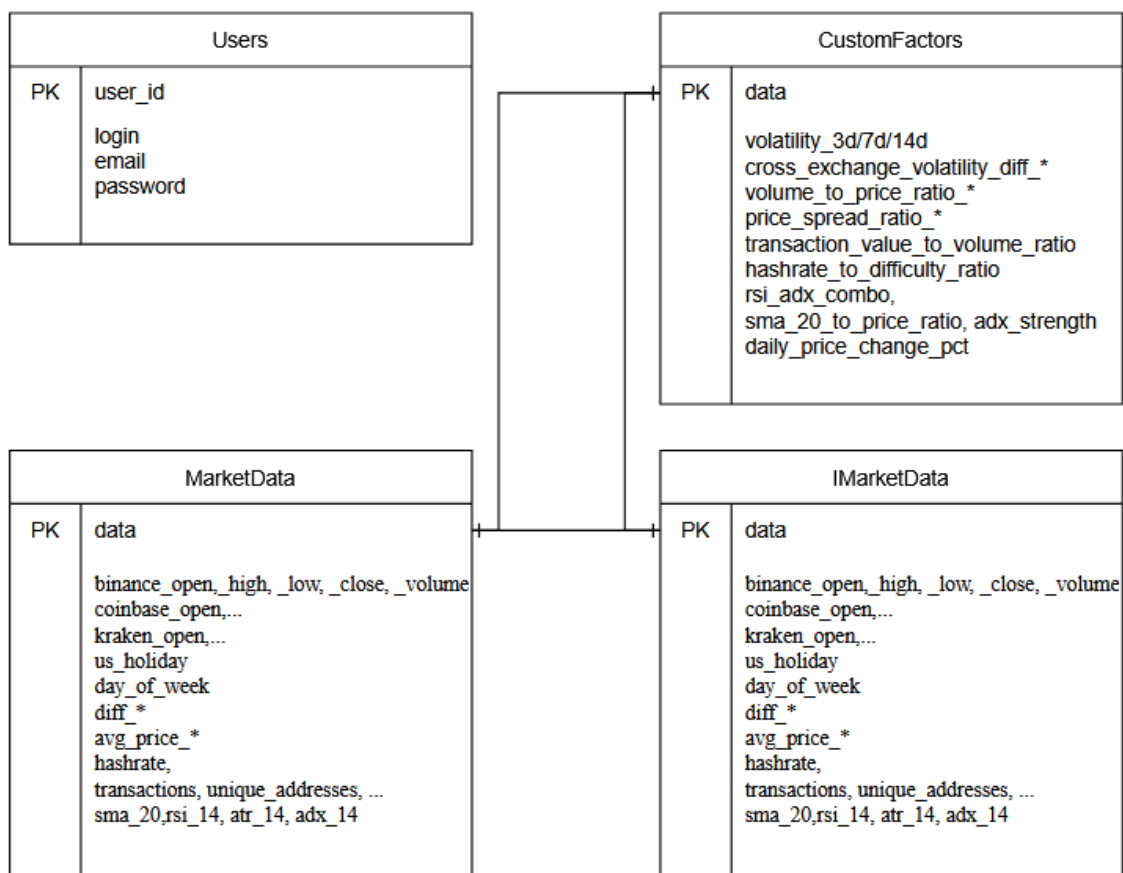
- 1.CoinGecko: офіційний вебсайт CoinGecko [Електронний ресурс]. – Режим доступу: <https://www.coingecko.com>. – Дата доступу: 15.04.2025 р.
- 2.Binance API Documentation [Електронний ресурс]. – Режим доступу: <https://binance-docs.github.io/apidocs/spot/en/>. – Дата доступу: 15.04.2025 р.
- 3.CryptoCompare: офіційний вебсайт CryptoCompare [Електронний ресурс]. – Режим доступу: <https://www.cryptocompare.com>. – Дата доступу: 15.04.2025 р.
- 4.Coinbase API Documentation [Електронний ресурс]. – Режим доступу: <https://docs.cdp.coinbase.com/exchange/docs/welcome>. – Дата доступу: 15.04.2025 р.
- 5.Glassnode: офіційний вебсайт Glassnode [Електронний ресурс]. – Режим доступу: <https://glassnode.com>. – Дата доступу: 18.04.2025 р.
- 6.Kraken API Documentation [Електронний ресурс]. – Режим доступу: <https://docs.kraken.com/rest/>. – Дата доступу: 18.04.2025 р.
- 7.Blockchain.com API Documentation [Електронний ресурс]. – Режим доступу: <https://www.blockchain.com/explorer/api>. – Дата доступу: 18.04.2025 р.
- 8.Yahoo Finance API Guide [Електронний ресурс]. – Режим доступу: <https://finance.yahoo.com>. – Дата доступу: 18.04.2025 р.
- 9.FRED Economic Data API [Електронний ресурс]. – Режим доступу: <https://fred.stlouisfed.org/docs/api/fred/>. – Дата доступу: 22.04.2025 р.
- 10.Python Official Documentation [Електронний ресурс]. – Режим доступу: <https://www.python.org/doc/>. – Дата доступу: 22.04.2025 р.
- 11.Pandas Documentation [Електронний ресурс]. – Режим доступу: <https://pandas.pydata.org/docs/>. – Дата доступу: 22.04.2025 р.
- 12.Scikit-learn Documentation [Електронний ресурс]. – Режим доступу: <https://scikit-learn.org/stable/documentation.html>. – Дата доступу: 22.04.2025 р.

- 13.Imbalanced-learn Documentation [Электронный ресурс]. – Режим доступа: <https://imbalanced-learn.org/stable/>. – Дата доступа: 22.04.2025 г.
- 14.SciPy Documentation [Электронный ресурс]. – Режим доступа: <https://docs.scipy.org/doc/scipy/>. – Дата доступа: 22.04.2025 г.
- 15.Requests Library Documentation [Электронный ресурс]. – Режим доступа: <https://requests.readthedocs.io/en/latest/>. – Дата доступа: 25.04.2025 г.
- 16.Yfinance Library Documentation [Электронный ресурс]. – Режим доступа: <https://github.com/ranaroussi/yfinance>. – Дата доступа: 25.04.2025 г.
- 17.Holidays Library Documentation [Электронный ресурс]. – Режим доступа: <https://pypi.org/project/holidays/>. – Дата доступа: 25.04.2025 г.
- 18.Jinja2 Documentation [Электронный ресурс]. – Режим доступа: <https://jinja.palletsprojects.com/en/3.1.x/>. – Дата доступа: 25.04.2025 г.
- 19.SQLAlchemy Documentation [Электронный ресурс]. – Режим доступа: <https://docs.sqlalchemy.org/en/20/>. – Дата доступа: 25.04.2025 г.
- 20.Flask Official Documentation [Электронный ресурс]. – Режим доступа: <https://flask.palletsprojects.com/en/3.0.x/>. – Дата доступа: 29.04.2025 г.
- 21.Werkzeug Security Documentation [Электронный ресурс]. – Режим доступа: <https://werkzeug.palletsprojects.com/en/3.0.x/>. – Дата доступа: 29.04.2025 г.
- 22.PostgreSQL Official Documentation [Электронный ресурс]. – Режим доступа: <https://www.postgresql.org/docs/>. – Дата доступа: 02.05.2025 г.
- 23.Flask-Login Documentation [Электронный ресурс]. – Режим доступа: <https://flask-login.readthedocs.io/en/latest/>. – Дата доступа: 02.05.2025 г.
- 24.Pandera Documentation [Электронный ресурс]. – Режим доступа: <https://pandera.readthedocs.io/en/stable/>. – Дата доступа: 06.05.2025 г.
- 25.Great Expectations Documentation [Электронный ресурс]. – Режим доступа: <https://docs.greatexpectations.io/docs/>. – Дата доступа: 06.05.2025 г.
- 26.Selenium Documentation [Электронный ресурс]. – Режим доступа: <https://www.selenium.dev/documentation/>. – Дата доступа: 06.05.2025 г.
- 27.Locust Documentation [Электронный ресурс]. – Режим доступа: <https://docs.locust.io/en/stable/>. – Дата доступа: 09.05.2025 г.

- 28.OWASP ZAP Documentation [Электронный ресурс]. – Режим доступа: <https://www.zaproxy.org/docs/>. – Дата доступа: 09.05.2025 г.
- 29.KuCoin API Documentation [Электронный ресурс]. – Режим доступа: <https://docs.kucoin.com>. – Дата доступа: 13.05.2025 г.
- 30.Dask Documentation [Электронный ресурс]. – Режим доступа: <https://docs.dask.org/en/stable/>. – Дата доступа: 13.05.2025 г.
- 31.JavaScript MDN Web Docs [Электронный ресурс]. – Режим доступа: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. – Дата доступа: 16.05.2025 г.
- 32.Tailwind CSS Documentation [Электронный ресурс]. – Режим доступа: <https://tailwindcss.com/docs>. – Дата доступа: 16.05.2025 г.

ДОДАТКИ

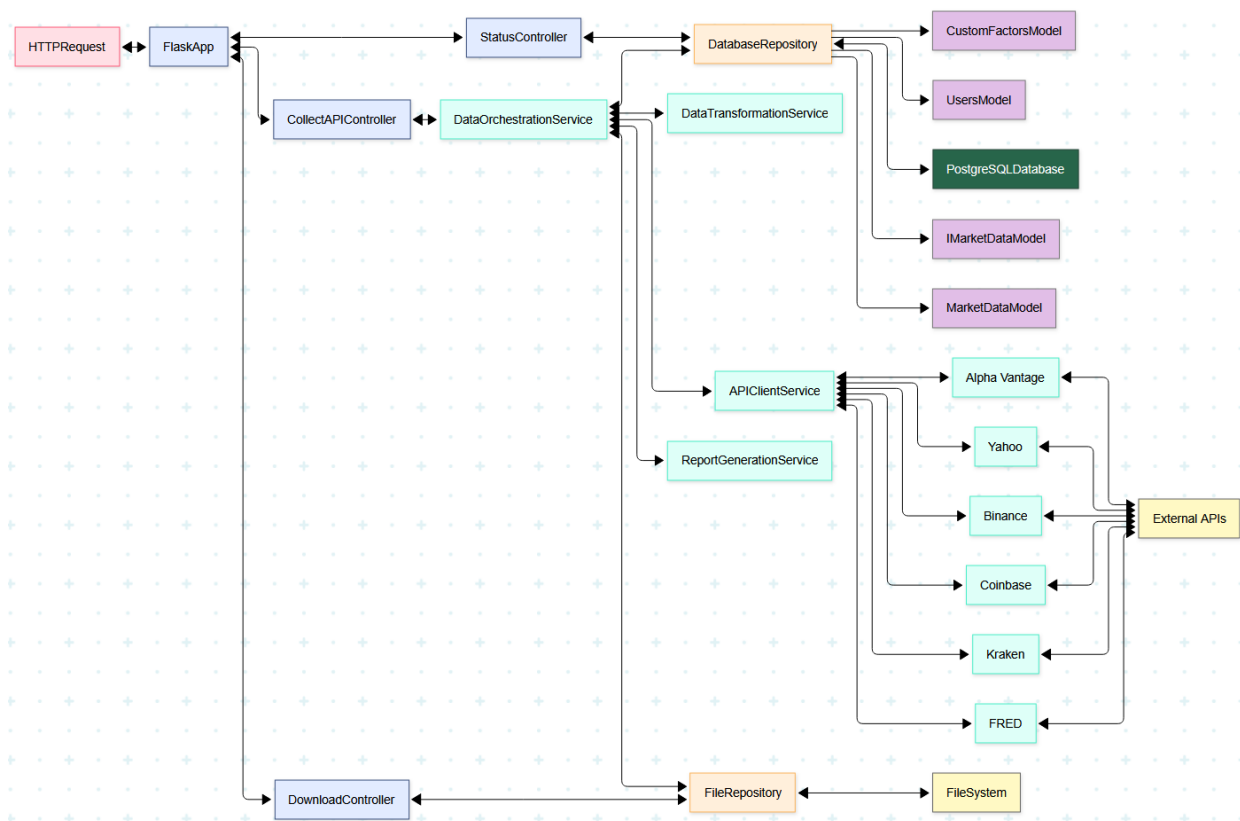
Додаток 1
Копії графічних матеріалів



Д
П.
04
54
40
-0
6-
99.
Be
бз
ас
то
су
но
к
дл
я
аг
ре
га
ції

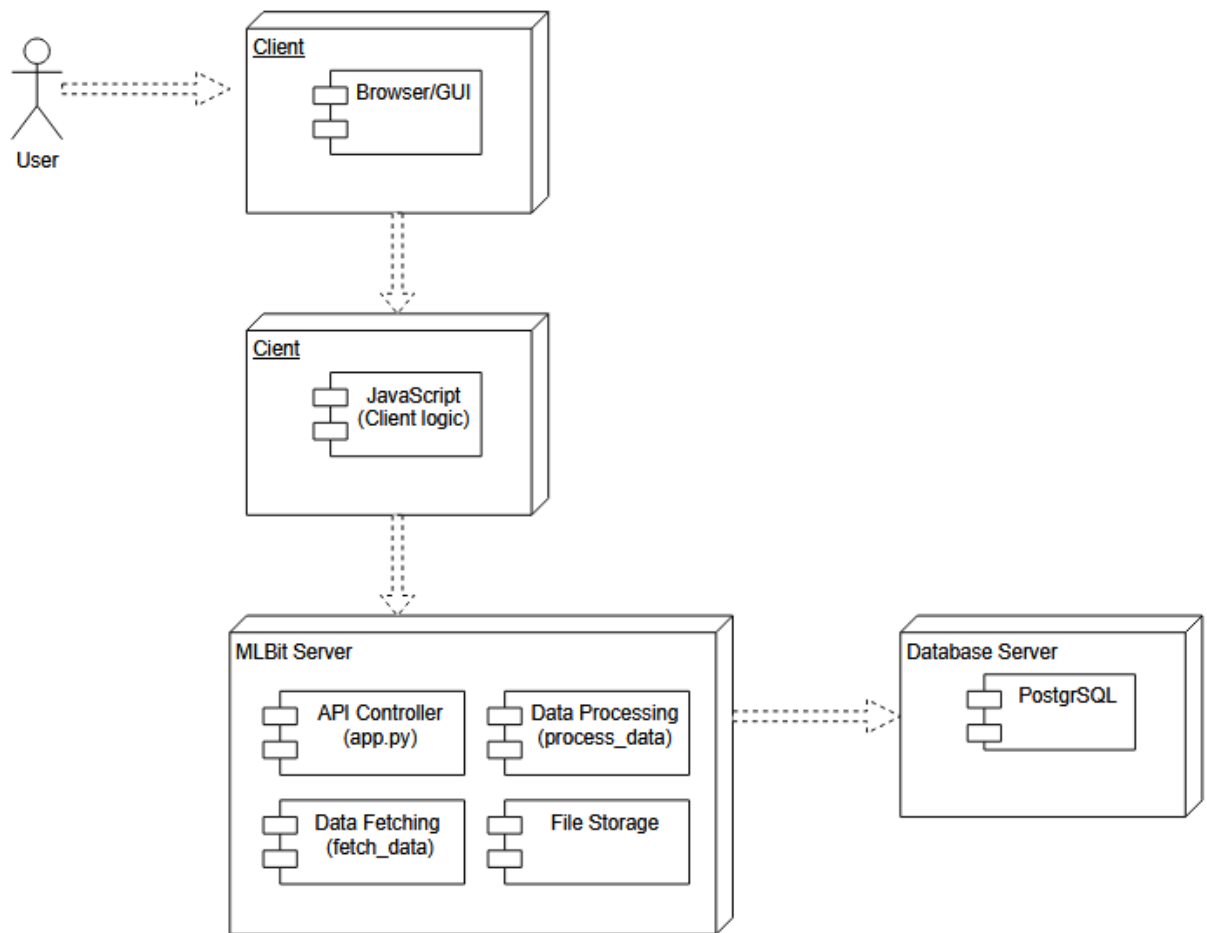
та
оч
и
ще
нн
я
да
ни
х
біт
кої
на
дл
я
ри
нк
ов
ог
о
пр
ог
но
зу
ва
нн
я.
Ст
ру
кт
ур
а
ба
зи
да
ни
х
ве
бз
ас
то
су
нк
у.
ER
D-
діа
гр

am
a

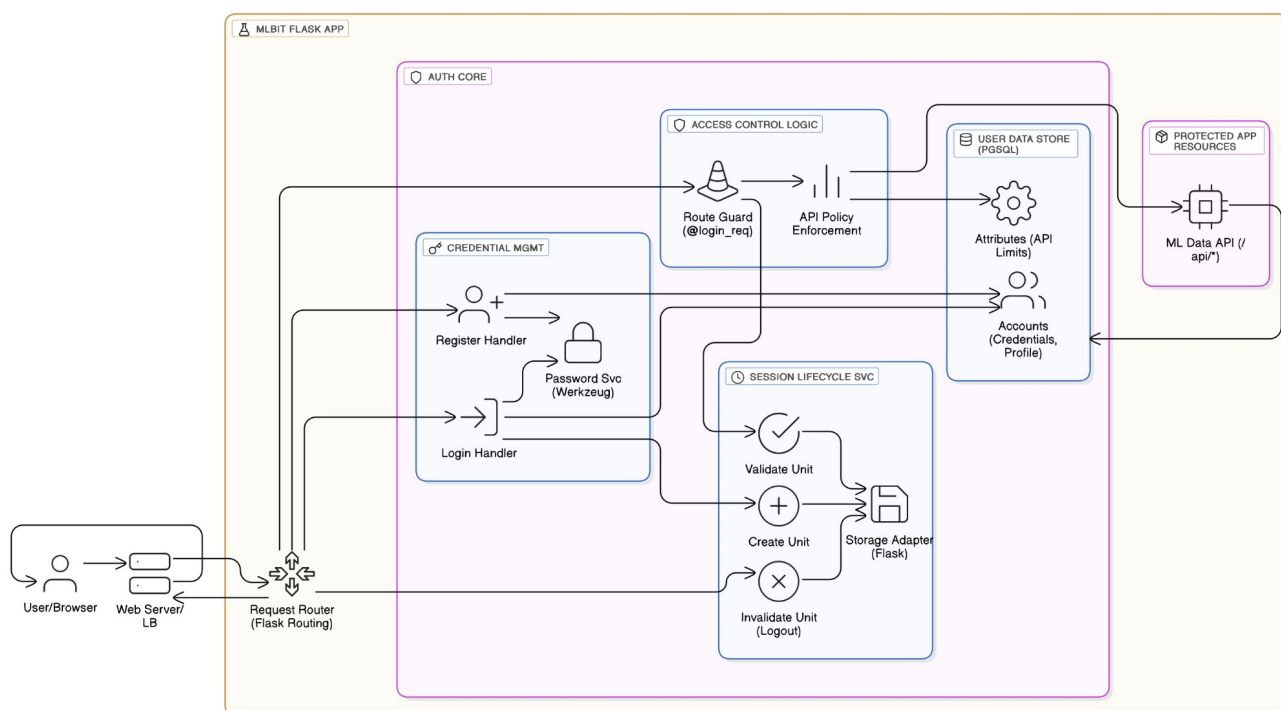


Д
П.
04
54
40
-0
7-
99.
Ве
бз
ас
то
су
но
к
дл
я
аг
ре
га
ції
та
оч
и

ще
нн
я
да
ни
х
біт
кої
на
дл
я
ри
нк
ов
ог
о
пр
ог
но
зу
ва
нн
я.
Ст
ру
кт
ур
а
кл
асі
в
се
рв
ер
но
ї
ча
ст
ин
и.
Ді
аг
ра
ма
ко
мп
он



Діаграма модулів системи вебасотунок у Фадеєв О. О., гр



Сх
ем
а
ар
хіт
ек
ту
ри
ав
те
нті
фі
ка
ції
Fla
sk-
log
in
Фа
де
єв
О.
О.,
гр

уп
а
К
П-
13

Додаток 2
Лістинг програми

```

app.py
import os
import logging
from flask import Flask, request, jsonify, send_file, make_response,
render_template, redirect, url_for, after_this_request
from dotenv import load_dotenv
import tempfile
import zipfile
from datetime import date, datetime
import pandas as pd
import time
import json
from jinja2 import Environment, FileSystemLoader
import shutil
import uuid
from data_processing.processing import process_data, generate_report
import traceback
import numpy as np
from pandas import Timestamp

load_dotenv()

from data_processing.api_clients import fetch_market_data
from data_processing.transformer import process_dataframe
from data_processing.exporter import export_data
from config_settings import Config

app = Flask(__name__)

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %
(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

TEMP_DIR = tempfile.gettempdir()
RESULTS_DIR = os.path.join(TEMP_DIR, 'mlbit_results')
os.makedirs(RESULTS_DIR, exist_ok=True)
REPORT_TEMPLATE_DIR = 'templates'
REPORT_TEMPLATE_NAME = 'data_quality_report.txt'
REPORT_FILENAME = 'quality_report.txt'

def make_json_serializable(obj):
    import pandas as pd
    import numpy as np
    from datetime import date, datetime
    from pandas import Timestamp
    from collections.abc import Mapping

    if isinstance(obj, (str, int, float, bool)) or obj is None:
        return obj
    elif isinstance(obj, (date, datetime, Timestamp)):
        return obj.isoformat()
    elif isinstance(obj, (np.integer, np.floating)):
        return obj.item()
    elif isinstance(obj, pd.DataFrame):
        return [make_json_serializable(row) for row in
obj.to_dict(orient='records')]
    elif isinstance(obj, pd.Series):
        return [make_json_serializable(x) for x in obj]
    elif isinstance(obj, Mapping):
        return {str(make_json_serializable(k)): make_json_serializable(v) for
k, v in obj.items()}
    elif isinstance(obj, (list, tuple, set)):

```

```

        return [make_json_serializable(x) for x in obj]
    else:
        print('[DEBUG] Не серіалізується тип:', type(obj), obj)
        return str(obj)

def convert_df_in_obj(obj):
    if isinstance(obj, (str, int, float, bool)) or obj is None:
        return obj
    elif isinstance(obj, dict):
        return {k: convert_df_in_obj(v) for k, v in obj.items()}
    elif isinstance(obj, (list, tuple, set)):
        return [convert_df_in_obj(x) for x in obj]
    elif isinstance(obj, pd.DataFrame):
        return obj.to_dict(orient='records')
    elif isinstance(obj, pd.Series):
        return obj.apply(convert_df_in_obj).to_list()
    elif isinstance(obj, (date, datetime, Timestamp)):
        return obj.isoformat()
    elif isinstance(obj, (np.integer, np.floating)):
        return obj.item()
    else:
        print('[DEBUG] Не серіалізується тип:', type(obj), obj)
        return str(obj)

@app.errorhandler(Exception)
def handle_exception(e):
    logger.error(f'An error occurred: {str(e)}')
    return jsonify({'error': 'Internal Server Error'}), 500

@app.route('/')
def main_landing():
    logger.info('Accessed the main landing page.')
    return render_template('main.html')

@app.route('/aggregate')
def index():
    logger.info('Accessed the aggregator page.')
    return render_template('index.html')

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/collect')
def collect_page():
    logger.info('Accessed the collect page.')
    return render_template('collect.html')

@app.route('/login')
def login_page():
    logger.info('Accessed the login page.')
    return render_template('login.html')

@app.route('/register')
def register_page():
    logger.info('Accessed the register page.')
    return render_template('register.html')

@app.route('/forgot-password')
def forgot_password_page():
    logger.info('Accessed the forgot password page.')
    return render_template('forgot_password.html')

```

```

@app.route('/api/collect', methods=['POST'])
def collect_api():
    try:
        params = {
            'start_date': request.form.get('start_date'),
            'end_date': request.form.get('end_date'),
            'exchanges': request.form.getlist('exchanges'),
            'ohlcv': request.form.getlist('ohlcv[]'),
            'diffs': request.form.getlist('diffs[]'),
            'avg': request.form.getlist('avg[]'),
            'dt': request.form.getlist('dt[]'),
            'blockchain': request.form.getlist('blockchain[]'),
            'indicators': request.form.getlist('indicators[]'),
            'macro': request.form.getlist('macro[]'),
            'volatility': request.form.getlist('volatility[]'),
            'vpr': request.form.getlist('vpr[]'),
            'crossratios': request.form.getlist('crossratios[]'),
            'macro_adv': request.form.getlist('macro-adv[]'),
            'blockchain_metrics': request.form.getlist('blockchain-
metrics[]'),
            'technical_metrics': request.form.getlist('technical-metrics[]'),
            'aggregated_indexes': request.form.getlist('aggregated-
indexes[]'),
            'additional_factors': request.form.getlist('additional-
factors[]'),
            'interpolation': request.form.get('interpolation-checkbox') ==
'on',
        }
        print('[DEBUG] Params received:', params)

        OHLCV_MAPPING = {
            'open_binance': 'binance_open',
            'high_binance': 'binance_high',
            'low_binance': 'binance_low',
            'close_binance': 'binance_close',
            'volume_binance': 'binance_volume',
            'open_coinbase': 'coinbase_open',
            'high_coinbase': 'coinbase_high',
            'low_coinbase': 'coinbase_low',
            'close_coinbase': 'coinbase_close',
            'volume_coinbase': 'coinbase_volume',
            'open_kraken': 'kraken_open',
            'high_kraken': 'kraken_high',
            'low_kraken': 'kraken_low',
            'close_kraken': 'kraken_close',
            'volume_kraken': 'kraken_volume'
        }

        AVG_PRICE_MAPPING = {
            'avg_avg_price_binance_coinbase': 'avg_price_binance_coinbase',
            'avg_avg_price_binance_kraken': 'avg_price_binance_kraken',
            'avg_avg_price_coinbase_kraken': 'avg_price_coinbase_kraken',
            'avg_avg_price_all': 'avg_price_all'
        }

        def _map_avg_price(factors):
            return [AVG_PRICE_MAPPING.get(f, f) for f in factors]

        def _map_ohlcv(factors):
            return [OHLCV_MAPPING.get(f, f) for f in factors]

        params['factors'] = (
            _map_ohlcv(params['ohlcv']) +

```

```

        params['diffs'] +
        _map_avg_price(params['avg']) +
        params['dt'] +
        params['blockchain'] +
        params['indicators'] +
        params['macro']
    )

    ADDITIONAL_FACTORS_MAPPING = {
        'avg_price_all_minus_close_binance':
'avg_price_all_minus_binance_close',
        'avg_price_all_minus_close_coinbase':
'avg_price_all_minus_coinbase_close',
        'avg_price_all_minus_close_kraken':
'avg_price_all_minus_kraken_close',
        'volatility_3d_to_7d_ratio_binance':
'binance_volatility_3d_to_7d_ratio',
        'volatility_3d_to_7d_ratio_coinbase':
'coinbase_volatility_3d_to_7d_ratio',
        'volatility_3d_to_7d_ratio_kraken':
'kraken_volatility_3d_to_7d_ratio',
    }
    def _map_additional_factors(factors):
        return [
            ADDITIONAL_FACTORS_MAPPING.get(f, f) for f in factors
            if f not in ('avg_price_all_minus_close_full',
'volatility_3d_to_7d_ratio_full')
        ]

    params['additional_factors_group'] = (
        params['volatility'] +
        params['vpr'] +
        params['crossratios'] +
        params['macro_adv'] +
        params['blockchain_metrics'] +
        params['technical_metrics'] +
        params['aggregated_indexes'] +
        _map_additional_factors(params['additional_factors'])
    )
    print('[DEBUG] Params received:', params)
    step4_fields = [
        'anomaly_filter', 'anomaly_method', 'normalization',
'yeo_johnson',
        'target_task', 'target_horizon', 'target_col', 'balance_classes',
'k_neighbors',
        'winsorize', 'winsor_limits', 'encoding', 'show_boxplot_stats',
'download', 'export_format'
    ]
    for field in step4_fields:
        val = request.form.get(field)
        print(f'[DEBUG] {field}: {val}')
        if val is not None:
            params[field] = val
        if val is None and field in ['yeo_johnson', 'winsorize',
'show_boxplot_stats']:
            params[field] = False

    processing = []
    if params.get('interpolation'):
        processing.append('interpolation')
    for field in step4_fields:
        val = params.get(field)
        if val not in [False, 'false', '0', '', None, 'none']:

```

```

        if val == 'on': val = True
        processing.append(f"{field}:{val}")

    params['full_factors'] = params['factors'] +
params['additional_factors_group']
    print ('[DEBUG] Params full factors:', params['full_factors'])
    params = dict(
        interpolate=bool(params.get('interpolation', False)),
        anomaly_filter=str(params.get('anomaly_filter', 'off')),
        anomaly_method=str(params.get('anomaly_method', 'zscore')),
        normalization=str(params.get('normalization', 'none')),
        encoding=str(params.get('encoding', 'none')),
        target_task=str(params.get('target_task', 'none')),
        target_horizon=int(params.get('target_horizon', 1)),
        export_format=str(params.get('export_format', 'csv')),
        factors=list(params.get('full_factors', [])),
        start_date=str(params.get('start_date')),
        end_date=str(params.get('end_date')),
        target_col=str(params.get('target_col')),
        show_boxplot_stats=bool(params.get('show_boxplot_stats', False)),
        yeo_johnson=bool(params.get('yeo_johnson', False)),
        winsorize=bool(params.get('winsorize', False)),
        winsor_limits=tuple(map(float, params.get('winsor_limits',
'0.01,0.99').split(','))) if params.get('winsor_limits') else (0.01, 0.99),
        balance_classes=str(params.get('balance_classes')),
        k_neighbors=int(params.get('k_neighbors', 1))
    )

    processed_df, stats, show_boxplot_stats = process_data(**params)
    processed_df.to_csv('processed.csv', index=False)
    report_text = generate_report(
        stats,
        show_boxplot_stats=show_boxplot_stats,
        yeo_johnson=params.get('yeo_johnson', False),
        winsorize=params.get('winsorize', False),
        balance_classes=params.get('balance_classes', None),
        k_neighbors=params.get('k_neighbors', 1)
    )
    start_date = params.get('start_date') or 'start'
    end_date = params.get('end_date') or 'end'
    export_format = params.get('export_format', 'csv')
    download_type = request.form.get('download', 'zip')
    print('[DEBUG] Download type:', download_type)
    print('[DEBUG] Download param:', params.get('download'))
    base_name = f"bitcoin_data_{start_date}_to_{end_date}"
    report_name = f"bitcoin_report_{start_date}_to_{end_date}.txt"
    data_name = f"bitcoin_data_{start_date}_to_{end_date}.
{export_format}"
    zip_name = f"{base_name}.zip"

    with tempfile.TemporaryDirectory() as tmpdir:
        data_path = os.path.join(tmpdir, data_name)
        if export_format == 'csv':
            processed_df.to_csv(data_path, index=False)
        elif export_format == 'json':
            processed_df.to_json(data_path, orient='records',
force_ascii=False)
        else:
            raise ValueError('Unsupported export_format')

        report_path = os.path.join(tmpdir, 'data_quality_report.txt')
        with open(report_path, 'w', encoding='utf-8') as f:
            f.write(report_text)

```

```

        if download_type == 'zip':
            zip_name = f"bitcoin_data_{start_date}_to_{end_date}.zip"
            tmpdir2 = tempfile.mkdtemp()
            zip_path = os.path.join(tmpdir2, zip_name)
            create_zip_archive(
                data_content=open(data_path, 'rb').read(),
                data_filename=data_name,
                text_report_content=report_text,
                start_date=start_date,
                end_date=end_date,
                zip_path=zip_path
            )
            import uuid
            task_id = str(uuid.uuid4())
            task_dir = os.path.join(RESULTS_DIR, task_id)
            os.makedirs(task_dir, exist_ok=True)
            final_zip_path = os.path.join(task_dir, zip_name)
            shutil.copy2(zip_path, final_zip_path)
            try:
                shutil.rmtree(tmpdir2)
            except Exception as e:
                print(f'[DEBUG] Не вдалося видалити tmpdir: {e}')
            return jsonify({'task_id': task_id, 'download_url': f'/
download/{task_id}'})
        else:
            raise ValueError('Unsupported download type')
    except Exception as e:
        print('Exception in collect_api:', e)
        print(traceback.format_exc())
        return jsonify({'error': str(e), 'traceback':
        traceback.format_exc()}), 500

def create_zip_archive(data_content, data_filename, text_report_content,
start_date, end_date, zip_path=None):
    if zip_path is None:
        zip_filename = f'bitcoin_data_{start_date}_to_{end_date}.zip'
    else:
        zip_filename = zip_path
    with zipfile.ZipFile(zip_filename, 'w') as zipf:
        zipf.writestr(data_filename, data_content)
        zipf.writestr('data_quality_report.txt', text_report_content)
    return zip_filename

@app.route('/download/<task_id>', methods=['GET'])
def download_result(task_id):
    import glob
    task_dir = os.path.join(RESULTS_DIR, task_id)
    zip_files = glob.glob(os.path.join(task_dir, '*.zip'))
    if not zip_files:
        return jsonify({'error': 'ZIP-файл не знайдено'}), 404
    zip_path = zip_files[0]
    zip_name = os.path.basename(zip_path)
    @after_this_request
    def cleanup(response):
        try:
            if os.path.exists(zip_path):
                os.remove(zip_path)
            if os.path.exists(task_dir):
                shutil.rmtree(task_dir)
        except Exception as e:
            print(f'[DEBUG] Не вдалося видалити файл/директорію: {e}')
    return response

```

```

        return send_file(zip_path, as_attachment=True, download_name=zip_name)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=int(os.environ.get('PORT', 5000)))

```

api_clients.py

```

import datetime as dt
import requests
import pandas as pd
import yfinance as yf
from datetime import timedelta, timezone
import time
import logging
from typing import List, Dict, Optional, Union, Any
import numpy as np
import json
import random
import pandas_datareader.data as web
from config_settings import Config
from fredapi import Fred

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

fred_client = Fred(api_key=Config.FRED_API_KEY)

BINANCE_API_URL = "https://api.binance.com/api/v3/klines"
COINBASE_API_URL = "https://api.exchange.coinbase.com/products/BTC-USD/candles"
KRAKEN_API_URL = "https://api.kraken.com/0/public/OHLC"
BLOCKCHAIN_HASH_URL = "https://api.blockchain.info/charts/hash-rate"
BLOCKCHAIN_TX_URL = "https://api.blockchain.info/charts/n-transactions"
FRED_SERIES = {
    'sp500_close': 'SP500',
    'vix_close': 'VIXCLS',
    'us10y': 'DGS10',
    'usd_index': 'DTWEXBGS',
    'gold': 'GOLDAMGBD228NLBM',
    'oil': 'DCOILWTICO',
}

RATE_LIMITS = {
    'binance': 1200,
    'coinbase': 30,
    'kraken': 15,
    'blockchain': 60
}

EXCHANGE_FETCHER = {
    'binance': lambda start, end: fetch_binance_data(start, end),
    'coinbase': lambda start, end: fetch_coinbase_data(start, end),
    'kraken': lambda start, end: fetch_kraken_data(start, end),
}

def _format_date_range(start_date_str: str, end_date_str: str) -> (dt, dt):
    start_dt = dt.strptime(start_date_str, '%Y-%m-%d').replace(tzinfo=timezone.utc)
    end_dt = dt.strptime(end_date_str, '%Y-%m-%d').replace(tzinfo=timezone.utc)

```

```

    return start_dt, end_dt

def _make_request(url: str, params: Dict = None, exchange: str = None) ->
Optional[Union[List, Dict]]:
    max_retries = 3
    retry_delay = 1
    rate_limit = RATE_LIMITS.get(exchange, 60) if exchange else 60

    for attempt in range(max_retries):
        if exchange:
            time.sleep(60 / rate_limit)

        headers = {
            'Accept': 'application/json',
            'Content-Type': 'application/json'
        }

        logger.info(f"Making request to {url} with params: {params}")
        response = requests.get(url, params=params, headers=headers,
timeout=30)
        logger.info(f"Response status code: {response.status_code}")
        logger.info(f"Response headers: {dict(response.headers)}")

        if response.status_code == 429:
            retry_after = int(response.headers.get('Retry-After', 60))
            logger.warning(f"Rate limit exceeded for {exchange}. Waiting
{retry_after} seconds.")
            time.sleep(retry_after)
            continue

        response.raise_for_status()
        data = response.json()
        logger.info(f"Response data (first 2 items): {str(data)[:500]}")
        return data

    logger.error(f"Request failed after {max_retries} attempts")
    return None

def _standardize_df(df: pd.DataFrame, value_col_name: str, date_col_index:
int = 0) -> pd.DataFrame:
    if df.empty or df.iloc[:, date_col_index].isnull().all():
        return pd.DataFrame(columns=[value_col_name])

    try:
        df['Date'] = pd.to_datetime(df.iloc[:, date_col_index], unit='s',
utc=True)
    except (ValueError, TypeError):
        df['Date'] = pd.to_datetime(df.iloc[:, date_col_index], utc=True)

    df = df.set_index('Date')
    df = df[[value_col_name]]
    df = df[~df.index.duplicated(keep='first')]
    df = df.sort_index()
    return df

def _adjust_price_anomalies(df: pd.DataFrame, exchange: str, threshold: float
= 2.0) -> pd.DataFrame:
    price_col = f'price_{exchange}'
    if price_col not in df.columns:
        return df

    df = df.copy()

```

```

window = 5
rolling_mean = df[price_col].rolling(window=window, center=True).mean()
rolling_std = df[price_col].rolling(window=window, center=True).std()

z_scores = (df[price_col] - rolling_mean) / rolling_std

anomalies = abs(z_scores) > threshold

if anomalies.any():
    logger.info(f"Found {anomalies.sum()} potential anomalies in
{exchange} prices")

    for idx in df[anomalies].index:
        start_idx = max(0, df.index.get_loc(idx) - window)
        end_idx = min(len(df), df.index.get_loc(idx) + window + 1)

        surrounding_prices = df[price_col].iloc[start_idx:end_idx].copy()
        surrounding_prices.loc[idx] = np.nan

        median_price = surrounding_prices.median()

        current_price = df.loc[idx, price_col]
        price_diff_pct = abs(current_price - median_price) / median_price
* 100

        if price_diff_pct > threshold * 100:
            logger.warning(f"Adjusting anomalous price for {exchange} at
{idx}: {current_price} -> {median_price}")
            df.loc[idx, price_col] = median_price

    return df

def _validate_exchange_data(df: pd.DataFrame, exchange: str) -> pd.DataFrame:
    if df.empty:
        return df

    price_col = f'price_{exchange}'
    volume_col = f'volume_{exchange}'

    if price_col in df.columns:
        invalid_prices = df[price_col] <= 0
        if invalid_prices.any():
            logger.warning(f"Found {invalid_prices.sum()} invalid prices in
{exchange} data")
            df.loc[invalid_prices, price_col] = np.nan

    if volume_col in df.columns:
        invalid_volumes = df[volume_col] < 0
        if invalid_volumes.any():
            logger.warning(f"Found {invalid_volumes.sum()} invalid volumes in
{exchange} data")
            df.loc[invalid_volumes, volume_col] = 0

    df = _adjust_price_anomalies(df, exchange)

    missing_data = df.isnull().sum()
    if missing_data.any():
        logger.info(f"Missing data in {exchange}: {missing_data.to_dict()}")

    return df

def fetch_binance_data(start_dt: dt, end_dt: dt) -> pd.DataFrame:

```

```

        logger.info(f"Fetching Binance data from {start_dt} to {end_dt -
timedelta(days=1)}")
        if isinstance(start_dt, dt.date) and not isinstance(start_dt,
dt.datetime):
            start_dt = dt.datetime.combine(start_dt, dt.time.min)
        if isinstance(end_dt, dt.date) and not isinstance(end_dt, dt.datetime):
            end_dt = dt.datetime.combine(end_dt, dt.time.min)
        start_ms = int(start_dt.timestamp() * 1000)
        end_ms = int(end_dt.timestamp() * 1000)

        params = {
            'symbol': 'BTCUSDT',
            'interval': '1d',
            'startTime': start_ms,
            'endTime': end_ms,
            'limit': 1000
        }

        max_retries = 3
        retry_delay = 2
        for attempt in range(max_retries):
            try:
                response = requests.get(BINANCE_API_URL, params=params,
timeout=30)
                response.raise_for_status()
                data = response.json()
                if not data:
                    if attempt < max_retries - 1:
                        time.sleep(retry_delay * (attempt + 1))
                        continue
                    return pd.DataFrame(columns=['binance_open', 'binance_high',
'binance_low', 'binance_close', 'binance_volume'])

                df = pd.DataFrame(data, columns=[
                    'Open time', 'Open', 'High', 'Low', 'Close', 'Volume', 'Close
time',
                    'Quote asset volume', 'Number of trades', 'Taker buy base
asset volume',
                    'Taker buy quote asset volume', 'Ignore'
                ])
                df['Date'] = pd.to_datetime(df['Open time'], unit='ms', utc=True)
                df = df.set_index('Date')
                df['binance_open'] = df['Open'].astype(float)
                df['binance_high'] = df['High'].astype(float)
                df['binance_low'] = df['Low'].astype(float)
                df['binance_close'] = df['Close'].astype(float)
                df['binance_volume'] = df['Volume'].astype(float)
                df = df[['binance_open', 'binance_high', 'binance_low',
'binance_close', 'binance_volume']]
                df = df[~df.index.duplicated(keep='first')]
                df = df.sort_index()
                return df
            except requests.exceptions.RequestException as e:
                if attempt < max_retries - 1:
                    time.sleep(retry_delay * (attempt + 1))
                else:
                    raise
        raise Exception(f"Error fetching Binance data after {max_retries}
attempts")

def fetch_coinbase_data(start_dt: dt, end_dt: dt) -> pd.DataFrame:
    logger.info(f"Fetching Coinbase data from {start_dt} to {end_dt -
timedelta(days=1)}")

```

```

        if isinstance(start_dt, dt.date) and not isinstance(start_dt,
dt.datetime):
            start_dt = dt.datetime.combine(start_dt, dt.time.min)
        if isinstance(end_dt, dt.date) and not isinstance(end_dt, dt.datetime):
            end_dt = dt.datetime.combine(end_dt, dt.time.min)
        params = {
            'start': start_dt.isoformat(),
            'end': end_dt.isoformat(),
            'granularity': 86400
        }
        headers = {
            'Accept': 'application/json',
            'Content-Type': 'application/json'
        }
        max_retries = 3
        retry_delay = 2
        for attempt in range(max_retries):
            try:
                response = requests.get(COINBASE_API_URL, params=params,
headers=headers, timeout=30)
                response.raise_for_status()
                data = response.json()
                if not data:
                    if attempt < max_retries - 1:
                        time.sleep(retry_delay * (attempt + 1))
                        continue
                    return pd.DataFrame(columns=['coinbase_open',
'coinbase_high', 'coinbase_low', 'coinbase_close', 'coinbase_volume'])
                df = pd.DataFrame(data, columns=['Time', 'Low', 'High', 'Open',
'Close', 'Volume'])
                df['Date'] = pd.to_datetime(df['Time'], unit='s', utc=True)
                df = df.set_index('Date')
                df['coinbase_open'] = df['Open'].astype(float)
                df['coinbase_high'] = df['High'].astype(float)
                df['coinbase_low'] = df['Low'].astype(float)
                df['coinbase_close'] = df['Close'].astype(float)
                df['coinbase_volume'] = df['Volume'].astype(float)
                df = df[['coinbase_open', 'coinbase_high', 'coinbase_low',
'coinbase_close', 'coinbase_volume']]
                df = df[~df.index.duplicated(keep='first')]
                df = df.sort_index()
                return df
            except requests.exceptions.RequestException as e:
                if attempt < max_retries - 1:
                    time.sleep(retry_delay * (attempt + 1))
                else:
                    raise
        raise Exception(f"Error fetching Coinbase data after {max_retries}
attempts")

def fetch_kraken_data(start_dt: dt, end_dt: dt) -> pd.DataFrame:
    logger.info(f"Fetching Kraken data from {start_dt} to {end_dt -
timedelta(days=1)}")
    if isinstance(start_dt, dt.date) and not isinstance(start_dt,
dt.datetime):
        start_dt = dt.datetime.combine(start_dt, dt.time.min)
    if isinstance(end_dt, dt.date) and not isinstance(end_dt, dt.datetime):
        end_dt = dt.datetime.combine(end_dt, dt.time.min)
    params = {
        'pair': 'XBTUSD',
        'interval': 1440,
        'since': int(start_dt.timestamp())
    }

```

```

max_retries = 3
retry_delay = 5
for attempt in range(max_retries):
    try:
        response = requests.get(KRAKEN_API_URL, params=params,
                                timeout=30)
        response.raise_for_status()
        data = response.json()
        if not data or 'result' not in data:
            if attempt < max_retries - 1:
                time.sleep(retry_delay * (attempt + 1))
                continue
            return pd.DataFrame(columns=['kraken_open', 'kraken_high',
                                         'kraken_low', 'kraken_close', 'kraken_volume'])
        pair_key = next((k for k in data['result'].keys() if k !=
                        'last'), None)
        if not pair_key or not data['result'][pair_key]:
            if attempt < max_retries - 1:
                time.sleep(retry_delay * (attempt + 1))
                continue
            return pd.DataFrame(columns=['kraken_open', 'kraken_high',
                                         'kraken_low', 'kraken_close', 'kraken_volume'])
        ohlc_data = data['result'][pair_key]
        df = pd.DataFrame(ohlc_data, columns=['Time', 'Open', 'High',
                                             'Low', 'Close', 'VWAP', 'Volume', 'Count'])
        df['Date'] = pd.to_datetime(df['Time'], unit='s', utc=True)
        df = df.set_index('Date')
        df['kraken_open'] = df['Open'].astype(float)
        df['kraken_high'] = df['High'].astype(float)
        df['kraken_low'] = df['Low'].astype(float)
        df['kraken_close'] = df['Close'].astype(float)
        df['kraken_volume'] = df['Volume'].astype(float)
        df = df[['kraken_open', 'kraken_high', 'kraken_low',
                  'kraken_close', 'kraken_volume']]
        df = df[~df.index.duplicated(keep='first')]
        df = df.sort_index()
        return df
    except requests.exceptions.RequestException as e:
        if attempt < max_retries - 1:
            time.sleep(retry_delay * (attempt + 1))
        else:
            raise
    raise Exception(f"Error fetching Kraken data after {max_retries}
attempts")

def fetch_blockchain_data(start_dt: dt, end_dt: dt) -> pd.DataFrame:
    logger.info(f"Fetching Blockchain.com data from {start_dt} to {end_dt -
timedelta(days=1)}")
    if isinstance(start_dt, dt.date) and not isinstance(start_dt,
dt.datetime):
        start_dt = dt.datetime.combine(start_dt, dt.time.min)
    if isinstance(end_dt, dt.date) and not isinstance(end_dt, dt.datetime):
        end_dt = dt.datetime.combine(end_dt, dt.time.min)
    dfs_to_merge = []

    duration_days = (end_dt - start_dt).days
    timespan = f"{duration_days}days"

    data_types = {
        'hashrate': {'url': BLOCKCHAIN_HASH_URL, 'min_value': 0, 'max_value':
1e9},
        'transactions': {'url': BLOCKCHAIN_TX_URL, 'min_value': 0,
'max_value': 1e7},

```

```

        'unique_addresses': {'url': 'https://api.blockchain.info/charts/n-unique-addresses', 'min_value': 0, 'max_value': 1e7},
        'usd_transaction_volume': {'url': 'https://api.blockchain.info/charts/output-volume', 'min_value': 0, 'max_value': 1e12},
        'transaction_fees': {'url': 'https://api.blockchain.info/charts/transaction-fees', 'min_value': 0, 'max_value': 1e9},
        'avg_block_size': {'url': 'https://api.blockchain.info/charts/avg-block-size', 'min_value': 0, 'max_value': 10},
        'avg_confirmation_time': {'url': 'https://api.blockchain.info/charts/avg-confirmation-time', 'min_value': 0, 'max_value': 3600},
        'difficulty': {'url': 'https://api.blockchain.info/charts/difficulty', 'min_value': 0, 'max_value': 1e16},
        'blocks_mined': {'url': 'https://api.blockchain.info/charts/blocks-size', 'min_value': 0, 'max_value': 500}
    }

    for data_type, info in data_types.items():
        max_retries = 3
        retry_delay = 2
        for attempt in range(max_retries):
            try:
                params = {'timespan': timespan, 'format': 'json'}
                data = _make_request(info['url'], params=params,
exchange='blockchain')
                if data and 'values' in data and data['values']:
                    blockchain_df = pd.DataFrame(data['values'])
                    if 'x' not in blockchain_df or 'y' not in blockchain_df:
                        break
                    blockchain_df['x'] = pd.to_datetime(blockchain_df['x'],
unit='s', utc=True)
                    blockchain_df.set_index('x', inplace=True)
                    blockchain_df.index =
blockchain_df.index.tz_convert(None)
                    blockchain_df = blockchain_df.rename(columns={'y':
data_type})

                    dfs_to_merge.append(blockchain_df[[data_type]])
                    break
                elif data and 'status' in data and data['status'] == 'ok' and
not data['values']:
                    break
                elif data and 'status' in data and data['status'] == 'error':
                    break
                else:
                    if attempt < max_retries - 1:
                        sleep_time = retry_delay * (attempt + 1) +
random.uniform(0, 2)
                        time.sleep(sleep_time)
                    else:
                        logger.warning(f"Failed to fetch {data_type} data
after {max_retries} attempts")
                    except Exception as e:
                        if hasattr(e, 'response') and getattr(e.response,
'status_code', None) == 404:
                            break
                        elif hasattr(e, 'response') and getattr(e.response,
'status_code', None) == 429:
                            sleep_time = 30 + random.uniform(0, 10)
                            time.sleep(sleep_time)
                        else:
                            if attempt < max_retries - 1:
                                sleep_time = retry_delay * (attempt + 1) +
random.uniform(0, 2)
                                time.sleep(sleep_time)

```

```

        else:
            logger.error(f"Failed to fetch {data_type} data after
all attempts")
        if not dfs_to_merge:
            return pd.DataFrame()
        final_df = pd.concat(dfs_to_merge, axis=1, join='outer')
        final_df.index = pd.to_datetime(final_df.index)
        if hasattr(final_df.index, 'tz') and final_df.index.tz is not None:
            final_df.index = final_df.index.tz_convert(None)
        for data_type in data_types.keys():
            if data_type in final_df.columns:
                if data_type == 'hashrate':
                    final_df[data_type] =
final_df[data_type].interpolate(method='linear', limit=3)
                elif data_type == 'transactions':
                    final_df[data_type] = final_df[data_type].ffill(limit=2)
        return final_df

def fetch_gold_price_metalsapi(api_key: str, date: str) -> float:
    url = f"https://metals-api.com/api/{date}"
    params = {
        "access_key": api_key,
        "base": "USD",
        "symbols": "XAU"
    }
    r = requests.get(url, params=params, timeout=10)
    r.raise_for_status()
    data = r.json()
    if "rates" in data and "XAU" in data["rates"]:
        return data["rates"]["XAU"]
    logger.warning(f"[Metals-API] No gold price for {date}: {data}")
    return None

def fetch_fred_data(start_dt: dt, end_dt: dt) -> pd.DataFrame:
    df = pd.DataFrame()
    for col, series_id in FRED_SERIES.items():
        if col == "gold":
            continue
        s = fred_client.get_series(series_id, start_dt, end_dt)
        df[col] = s.squeeze()
    return df

def _check_cross_exchange_consistency(df: pd.DataFrame) -> pd.DataFrame:
    price_columns = [col for col in df.columns if col.startswith('price_')]

    if len(price_columns) < 2:
        return df

    for i in range(len(price_columns)):
        for j in range(i + 1, len(price_columns)):
            col1, col2 = price_columns[i], price_columns[j]
            diff_col = f'diff_{col1}_{col2}'

            df[diff_col] = ((df[col1] - df[col2]) / df[col2] * 100).abs()

            large_diff = df[diff_col] > 2
            if large_diff.any():
                logger.warning(f"Found {large_diff.sum()} large price
discrepancies between {col1} and {col2}")
                discrepancies = df[large_diff][[col1, col2, diff_col]]
                for date, row in discrepancies.iterrows():
                    logger.warning(f>Date: {date}, {col1}: {row[col1]:.2f},
{col2}: {row[col2]:.2f}, Diff: {row[diff_col]:.2f}%")

```

```

        df.loc[large_diff, col1] = (df.loc[large_diff, col1] +
df.loc[large_diff, col2]) / 2
        df.loc[large_diff, col2] = df.loc[large_diff, col1]
        df.loc[large_diff, diff_col] = 0

    return df

def fetch_market_data(params: dict) -> pd.DataFrame:
    try:
        start_dt, end_dt = _format_date_range(params['start_date'],
params['end_date'])
    except ValueError:
        return pd.DataFrame()

    requested_exchanges = params.get('exchanges', [])
    if not requested_exchanges:
        logger.warning("No exchanges specified in parameters. Fetching only
non-exchange data.")

    all_dataframes = []

    exchange_dfs = []
    for exchange_name in requested_exchanges:
        fetcher = EXCHANGE_FETCHER.get(exchange_name.lower())
        if fetcher:
            try:
                exchange_df = fetcher(start_dt, end_dt)
                if not exchange_df.empty:
                    exchange_df = _validate_exchange_data(exchange_df,
exchange_name)
                    exchange_dfs.append(exchange_df)
            except Exception as e:
                logger.error(f"Error fetching data from {exchange_name}:
{e}")
                continue
        else:
            logger.warning(f"No fetcher implemented for exchange:
{exchange_name}")

    if exchange_dfs:
        date_range = pd.date_range(start=start_dt, end=end_dt -
timedelta(days=1), freq='D', tz='UTC')
        exchange_data = pd.DataFrame(index=date_range)

        for df in exchange_dfs:
            exchange_data = pd.merge(exchange_data, df, left_index=True,
right_index=True, how='left')

        exchange_data = exchange_data.fillna(method='ffill', limit=1)

        price_cols = [col for col in exchange_data.columns if
col.startswith('price_')]
        volume_cols = [col for col in exchange_data.columns if
col.startswith('volume_')]

        if price_cols:
            for col in price_cols:
                exchange_data[col] =
exchange_data[col].fillna(exchange_data[price_cols].mean(axis=1))

        exchange_data['median_price'] =
exchange_data[price_cols].median(axis=1)

```

```

        window = 3
        for col in price_cols:
            rolling_median = exchange_data[col].rolling(window=window,
center=True, min_periods=1).median()
            pct_diff = ((exchange_data[col] - rolling_median) /
rolling_median * 100).abs()
            outliers = pct_diff > 0.75
            if outliers.any():
                exchange_data.loc[outliers, col] =
rolling_median[outliers]

        alpha = 0.3
        for col in price_cols:
            ema = exchange_data[col].ewm(alpha=alpha,
adjust=False).mean()
            pct_diff = ((exchange_data[col] - ema) / ema * 100).abs()
            outliers = pct_diff > 0.75
            if outliers.any():
                exchange_data.loc[outliers, col] = ema[outliers]

        for col in price_cols:
            pct_diff = ((exchange_data[col] -
exchange_data['median_price']) / exchange_data['median_price'] * 100).abs()
            large_diff = pct_diff > 0.75
            if large_diff.any():
                exchange_data.loc[large_diff, col] = (
                    0.25 * exchange_data.loc[large_diff, col] +
                    0.75 * exchange_data.loc[large_diff, 'median_price']
                )

        for i in range(len(price_cols)):
            for j in range(i + 1, len(price_cols)):
                col1, col2 = price_cols[i], price_cols[j]
                pct_diff = ((exchange_data[col1] - exchange_data[col2]) /
exchange_data[col2] * 100).abs()
                large_diff = pct_diff > 0.75
                if large_diff.any():
                    avg_price = (exchange_data.loc[large_diff, col1] +
exchange_data.loc[large_diff, col2]) / 2
                    exchange_data.loc[large_diff, col1] = avg_price
                    exchange_data.loc[large_diff, col2] = avg_price

        exchange_data['median_price'] =
exchange_data[price_cols].median(axis=1)
        for col in price_cols:
            pct_diff = ((exchange_data[col] -
exchange_data['median_price']) / exchange_data['median_price'] * 100).abs()
            large_diff = pct_diff > 0.25
            if large_diff.any():
                exchange_data.loc[large_diff, col] =
exchange_data.loc[large_diff, 'median_price']

        exchange_data = exchange_data.drop('median_price', axis=1)

        for col in volume_cols:
            negative_volumes = exchange_data[col] < 0
            if negative_volumes.any():
                exchange_data.loc[negative_volumes, col] = 0

    all_dataframes.append(exchange_data)

try:

```

```

        blockchain_df = fetch_blockchain_data(start_dt, end_dt)
        if not blockchain_df.empty:
            all_dataframes.append(blockchain_df)
    except Exception as e:
        logger.error(f"Error fetching blockchain data: {e}")

    try:
        fred_df = fetch_fred_data(start_dt, end_dt)
        if not fred_df.empty:
            all_dataframes.append(fred_df)
        else:
            logger.warning("FRED DataFrame is empty!")
    except Exception as e:
        logger.error(f"Error fetching FRED data: {e}")

    if not all_dataframes:
        logger.error("Failed to fetch data from any source")
        return pd.DataFrame()

    if all_dataframes:
        final_df = all_dataframes[0].copy()

        for df in all_dataframes[1:]:
            final_df = pd.merge(final_df, df, left_index=True,
                                right_index=True, how='outer')

        final_df = final_df.sort_index()

        volume_cols = [col for col in final_df.columns if
                        col.startswith('volume_')]
        for col in volume_cols:
            negative_volumes = final_df[col] < 0
            if negative_volumes.any():
                final_df.loc[negative_volumes, col] = 0

        logger.info(f"Final merged DataFrame created with {len(final_df)}
records")
        logger.info(f"Date range: {final_df.index.min()} to
{final_df.index.max()}")
        logger.info(f"Columns: {final_df.columns.tolist()}")

        logger.info("\nData completeness (%):")
        for col in final_df.columns:
            completeness = (final_df[col].notna().sum() / len(final_df)) *
100
            logger.info(f"{col}: {completeness:.1f}%")

        return final_df

class BaseClient:
    def __init__(self, api_key: Optional[str] = None, api_secret:
Optional[str] = None):
        self.api_key = api_key
        self.api_secret = api_secret
        self.session = requests.Session()

    def _make_request(self, method: str, endpoint: str, params:
Optional[Dict[str, Any]] = None) -> Dict[str, Any]:
        response = self.session.request(method, endpoint, params=params)
        response.raise_for_status()
        return response.json()

class BinanceClient(BaseClient):

```

```

    def __init__(self, api_key: Optional[str] = None, api_secret:
Optional[str] = None):
        super().__init__(api_key, api_secret)
        self.base_url = 'https://api.binance.com/api/v3'

    def get_historical_data(self, symbol: str, start_date: str, end_date:
str) -> pd.DataFrame:
        start_ts = int(dt.strptime(start_date, '%Y-%m-%d').timestamp() *
1000)
        end_ts = int(dt.strptime(end_date, '%Y-%m-%d').timestamp() * 1000)

        endpoint = f"{self.base_url}/klines"
        params = {
            'symbol': symbol,
            'interval': '1d',
            'startTime': start_ts,
            'endTime': end_ts
        }

        response = self._make_request('GET', endpoint, params)

        df = pd.DataFrame(response, columns=[
            'timestamp', 'open', 'high', 'low', 'close', 'volume',
            'close_time', 'quote_volume', 'trades', 'taker_buy_base',
            'taker_buy_quote', 'ignore'
        ])

        df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms')
        df = df.set_index('timestamp')

        for col in ['open', 'high', 'low', 'close']:
            df[col] = df[col].astype(float)

        df = df[['close']].rename(columns={'close': f'price_binance'})

        return df

class CoinbaseClient(BaseClient):
    def __init__(self, api_key: Optional[str] = None, api_secret:
Optional[str] = None):
        super().__init__(api_key, api_secret)
        self.base_url = 'https://api.pro.coinbase.com'

    def get_historical_data(self, symbol: str, start_date: str, end_date:
str) -> pd.DataFrame:
        start_iso = dt.strptime(start_date, '%Y-%m-%d').isoformat()
        end_iso = dt.strptime(end_date, '%Y-%m-%d').isoformat()

        endpoint = f"{self.base_url}/products/{symbol}/candles"
        params = {
            'start': start_iso,
            'end': end_iso,
            'granularity': 86400
        }

        response = self._make_request('GET', endpoint, params)

        df = pd.DataFrame(response, columns=[
            'timestamp', 'open', 'high', 'low', 'close', 'volume'
        ])

        df['timestamp'] = pd.to_datetime(df['timestamp'], unit='s')
        df = df.set_index('timestamp')

```

```

        for col in ['open', 'high', 'low', 'close']:
            df[col] = df[col].astype(float)

        df = df[['close']].rename(columns={'close': f'price_coinbase'})

        return df

class KrakenClient(BaseClient):
    def __init__(self, api_key: Optional[str] = None, api_secret:
Optional[str] = None):
        super().__init__(api_key, api_secret)
        self.base_url = 'https://api.kraken.com/0'

    def get_historical_data(self, symbol: str, start_date: str, end_date:
str) -> pd.DataFrame:
        start_ts = int(dt.strptime(start_date, '%Y-%m-%d').timestamp())
        end_ts = int(dt.strptime(end_date, '%Y-%m-%d').timestamp())

        endpoint = f"{self.base_url}/public/OHLC"
        params = {
            'pair': symbol,
            'interval': 1440,
            'since': start_ts
        }

        response = self._make_request('GET', endpoint, params)

        data = response['result'][symbol]
        df = pd.DataFrame(data, columns=[
            'timestamp', 'open', 'high', 'low', 'close', 'vwap',
            'volume', 'trades'
        ])

        df['timestamp'] = pd.to_datetime(df['timestamp'], unit='s')
        df = df.set_index('timestamp')

        for col in ['open', 'high', 'low', 'close']:
            df[col] = df[col].astype(float)

        df = df[df.index <= end_date]

        df = df[['close']].rename(columns={'close': f'price_kraken'})

        return df

def fetch_alpha_indicator(function, symbol, interval="daily", time_period=14,
series_type="close", extra_params=None):
    url = "https://www.alphavantage.co/query"
    params = {
        "function": function,
        "symbol": symbol,
        "interval": interval,
        "apikey": Config.ALPHA_VANTAGE_API_KEY
    }
    if time_period:
        params["time_period"] = time_period
    if series_type:
        params["series_type"] = series_type
    if extra_params:
        params.update(extra_params)
    r = requests.get(url, params=params)
    data = r.json()

```

```

key = [k for k in data.keys() if "Technical Analysis" in k]
if key:
    df = pd.DataFrame.from_dict(data[key[0]], orient="index")
    df.index = pd.to_datetime(df.index)
    df = df.sort_index()
    return df
print(f"Alpha Vantage error: {data}")
return pd.DataFrame()

def fetch_main_indicators(symbol="BTCUSD"):
    sma = fetch_alpha_indicator("SMA", symbol,
time_period=20).rename(columns={"SMA": "sma_20"})
    rsi = fetch_alpha_indicator("RSI", symbol,
time_period=14).rename(columns={"RSI": "rsi_14"})
    atr = fetch_alpha_indicator("ATR", symbol, time_period=14,
series_type=None).rename(columns={"ATR": "atr_14"})
    adx = fetch_alpha_indicator("ADX", symbol, time_period=14,
series_type=None).rename(columns={"ADX": "adx_14"})
    dfs = [sma, rsi, atr, adx]
    dfs = [df for df in dfs if df is not None and not df.empty]
    indicators = reduce(lambda left, right: left.join(right, how="outer"),
dfs)
    return indicators

```

processing.py

```

import math
import pandas as pd
import numpy as np
from sqlalchemy.orm import sessionmaker
from sqlalchemy import create_engine
from typing import Dict, Any, List, Tuple, Optional
import logging
import warnings
from models import MarketData, IMarketData, CustomFactors
from db import engine

logger = logging.getLogger(__name__)
Session = sessionmaker(bind=engine)
session = Session()

ANOMALY_FACTORS = [
    'binance_open', 'binance_high', 'binance_low', 'binance_close',
    'binance_volume',
    'coinbase_open', 'coinbase_high', 'coinbase_low', 'coinbase_close',
    'coinbase_volume',
    'kraken_open', 'kraken_high', 'kraken_low', 'kraken_close',
    'kraken_volume',
    'diff_binance_coinbase_open', 'diff_binance_coinbase_high',
    'diff_binance_coinbase_low', 'diff_binance_coinbase_close',
    'diff_binance_coinbase_volume',
    'diff_binance_kraken_open', 'diff_binance_kraken_high',
    'diff_binance_kraken_low', 'diff_binance_kraken_close',
    'diff_binance_kraken_volume',
    'diff_coinbase_kraken_open', 'diff_coinbase_kraken_high',
    'diff_coinbase_kraken_low', 'diff_coinbase_kraken_close',
    'diff_coinbase_kraken_volume',
    'diff_all_open', 'diff_all_high', 'diff_all_low', 'diff_all_close',
    'diff_all_volume',
    'avg_price_binance_coinbase', 'avg_price_binance_kraken',
    'avg_price_coinbase_kraken', 'avg_price_all',
    'hashrate', 'transactions', 'unique_addresses', 'usd_transaction_volume',
    'transaction_fees',

```

```

    'avg_block_size', 'avg_confirmation_time', 'difficulty', 'blocks_mined',
    'sma_20', 'rsi_14', 'atr_14', 'adx_14',
    'sp500_close', 'vix_close', 'usl0y', 'usd_index', 'gold', 'oil'
]
ENCODE_FACTORS = ['us_holiday', 'day_of_week']
NORMALIZE_FACTORS = ANOMALY_FACTORS.copy()
CUSTOM_NUMERIC_FACTORS = [
    'binance_volatility_3d', 'coinbase_volatility_3d',
    'kraken_volatility_3d',
    'binance_volatility_7d', 'coinbase_volatility_7d',
    'kraken_volatility_7d',
    'binance_volatility_14d', 'coinbase_volatility_14d',
    'kraken_volatility_14d',
    'cross_exchange_volatility_diff_binance_coinbase_3d',
    'cross_exchange_volatility_diff_binance_kraken_3d',
    'cross_exchange_volatility_diff_coinbase_kraken_3d',
    'cross_exchange_volatility_diff_binance_coinbase_7d',
    'cross_exchange_volatility_diff_binance_kraken_7d',
    'cross_exchange_volatility_diff_coinbase_kraken_7d',
    'binance_volume_to_price_ratio', 'coinbase_volume_to_price_ratio',
    'kraken_volume_to_price_ratio',
    'binance_volume_to_avg_price_ratio',
    'coinbase_volume_to_avg_price_ratio', 'kraken_volume_to_avg_price_ratio',
    'price_spread_ratio_binance_coinbase',
    'price_spread_ratio_binance_kraken', 'price_spread_ratio_coinbase_kraken',
    'volume_spread_ratio_binance_coinbase',
    'volume_spread_ratio_binance_kraken', 'volume_spread_ratio_coinbase_kraken',
    'volatility_spread_ratio_binance_coinbase_3d',
    'volatility_spread_ratio_binance_kraken_3d',
    'volatility_spread_ratio_coinbase_kraken_3d',
    'bitcoin_sp500_correlation_7d', 'bitcoin_sp500_correlation_14d',
    'bitcoin_vix_ratio', 'bitcoin_gold_ratio', 'bitcoin_oil_ratio',
    'bitcoin_usd_index_correlation_7d', 'transaction_value_to_volume_ratio',
    'hashrate_to_difficulty_ratio',
    'transaction_fees_to_price_ratio',
    'transactions_to_unique_addresses_ratio',
    'avg_block_size_daily', 'avg_confirmation_time_daily', 'rsi_adx_combo',
    'sma_20_to_price_ratio', 'adx_strength',
    'price_deviation_from_sma_20', 'daily_price_change_pct',
    'blockchain_activity_score', 'cross_market_sentiment_index',
    'exchange_volume_weighted_price', 'avg_price_all_minus_binance_close',
    'avg_price_all_minus_coinbase_close',
    'avg_price_all_minus_kraken_close', 'binance_volatility_3d_to_7d_ratio',
    'coinbase_volatility_3d_to_7d_ratio',
    'kraken_volatility_3d_to_7d_ratio', 'days_since_last_us_holiday',
    'avg_close_5d', 'avg_close_10d'
]
ANOMALY_FACTORS += CUSTOM_NUMERIC_FACTORS
NORMALIZE_FACTORS += CUSTOM_NUMERIC_FACTORS
TARGET_FACTORS = ANOMALY_FACTORS.copy()

def fetch_data(interpolate: bool, factors: List[str], start_date: str = None,
end_date: str = None) -> pd.DataFrame:
    query = None
    if interpolate:
        query = session.query(IMarketData)
        if start_date and end_date:
            query = query.filter(IMarketData.date >= start_date,
IMarketData.date <= end_date)
        df = pd.read_sql(query.statement, session.bind)
        if factors:
            custom_factors = [f for f in factors if hasattr(CustomFactors,
f)]

```

```

        if custom_factors:
            cf_query = session.query(CustomFactors)
            if start_date and end_date:
                cf_query = cf_query.filter(CustomFactors.date >=
start_date, CustomFactors.date <= end_date)
            custom_df = pd.read_sql(cf_query.statement, session.bind)
            df = df.merge(custom_df[custom_factors + ['date']],
on='date', how='left')
        else:
            query = session.query(MarketData)
            if start_date and end_date:
                query = query.filter(MarketData.date >= start_date,
MarketData.date <= end_date)
            df = pd.read_sql(query.statement, session.bind)
            factors = [f for f in factors if not hasattr(CustomFactors, f)]
            keep_cols = ['date'] + [f for f in factors if f != 'date']
            keep_cols = [col for col in keep_cols if col in df.columns]
            df = df[keep_cols]
            return df.sort_values('date')

def smart_select_target(df, task_hint=None):
    priority = [
        'binance_close', 'coinbase_close', 'kraken_close',
        'avg_price_all', 'avg_price_binance_coinbase',
        'avg_price_binance_kraken', 'avg_price_coinbase_kraken',
        'binance_volume', 'coinbase_volume', 'kraken_volume'
    ]
    priority += [col for col in df.columns if any(x in col for x in
['volatility', 'spread', 'ratio'])]
    if task_hint:
        for col in df.columns:
            if task_hint in col:
                return col
    for col in priority:
        if col in df.columns:
            return col
    candidates = [col for col in df.columns if
pd.api.types.is_numeric_dtype(df[col])]
    if candidates:
        candidates = sorted(candidates, key=lambda c: (df[c].isnull().sum(),
-df[c].std()))
        return candidates[0]
    return None

def _winsorize_series(s, lower=0.01, upper=0.99):
    q_low = s.quantile(lower)
    q_high = s.quantile(upper)
    return s.clip(lower=q_low, upper=q_high)

def process_data(
    interpolate: bool = False,
    anomaly_filter: str = 'off',
    anomaly_method: str = 'zscore',
    normalization: str = 'none',
    encoding: str = 'none',
    target_task: str = 'none',
    target_horizon: int = 1,
    export_format: str = 'csv',
    factors: Optional[List[str]] = None,
    start_date: str = None,
    end_date: str = None,
    target_col: str = None,
    show_boxplot_stats: bool = False,

```

```

    yeo_johnson: bool = False,
    winsorize: bool = False,
    winsor_limits: Tuple[float, float] = (0.01, 0.99),
    balance_classes: str = None,
    k_neighbors: int = 1
) -> Tuple[pd.DataFrame, Dict[str, Any], bool]:
    if factors is None:
        factors = []
    df = fetch_data(interpolate, factors, start_date, end_date)
    stats = {'initial_rows': len(df), 'initial_columns': list(df.columns),
'start_date': start_date, 'end_date': end_date}
    stats['interpolate'] = interpolate
    stats['anomaly_filter'] = anomaly_filter
    stats['anomaly_method'] = anomaly_method
    stats['normalization'] = normalization
    stats['encoding'] = encoding
    stats['target_task'] = target_task
    stats['target_horizon'] = target_horizon
    stats['export_format'] = export_format

    if 'df_full' not in stats:
        df_full = fetch_data(interpolate, factors, None, None)
        stats['df_full'] = df_full.copy()

    nan_cols = [col for col in df.columns if df[col].isnull().all()]
    if nan_cols:
        df = df.drop(columns=nan_cols)
    stats['all_nan_columns_dropped'] = nan_cols

    trimmed_dates = None
    target_col_replaced = None
    orig_target_col = target_col
    if target_col and target_col in df.columns:
        if df[target_col].notnull().any():
            notnull_idx = df[df[target_col].notnull()].index
            first_idx, last_idx = notnull_idx[0], notnull_idx[-1]
            trimmed_dates = (str(df.loc[first_idx, 'date']),
str(df.loc[last_idx, 'date']))
            df = df.loc[first_idx:last_idx].reset_index(drop=True)
        else:
            stats['target_col_all_nan'] = True
    else:
        stats['target_col_all_nan'] = False
    stats['trimmed_dates'] = trimmed_dates

    anomaly_counts = {}
    dropped_dates_anomaly = {}
    if anomaly_filter != 'off' and anomaly_method in ('zscore', 'iqr',
'both'):
        factors_to_check = [f for f in ANOMALY_FACTORS if f in df.columns]
        for col in factors_to_check:
            mask = None
            if anomaly_method == 'zscore':
                z = (df[col] - df[col].mean()) / df[col].std()
                mask = np.abs(z) > 3
            elif anomaly_method == 'iqr':
                q1 = df[col].quantile(0.25)
                q3 = df[col].quantile(0.75)
                iqr = q3 - q1
                mask = (df[col] < (q1 - 1.5 * iqr)) | (df[col] > (q3 + 1.5 *
iqr))
            elif anomaly_method == 'both':
                z = (df[col] - df[col].mean()) / df[col].std()

```

```

        mask_z = np.abs(z) > 3
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)
        iqr = q3 - q1
        mask_iqr = (df[col] < (q1 - 1.5 * iqr)) | (df[col] > (q3 +
1.5 * iqr))
        mask = mask_z | mask_iqr
        anomaly_counts[col] = int(mask.sum())
        if anomaly_filter == 'strict':
            dropped_dates_anomaly[col] = df.loc[mask,
'date'].astype(str).tolist()
            df = df[~mask]
        stats['anomaly_counts'] = anomaly_counts
        stats['rows_after_anomaly'] = len(df)
        stats['dropped_dates_anomaly'] = dropped_dates_anomaly

    anomaly_filtering_report = []
    for col, dates in dropped_dates_anomaly.items():
        if dates:
            anomaly_filtering_report.append(f"{col}:")
            month_groups = defaultdict(list)
            for d in dates:
                dt = pd.to_datetime(d)
                key = f"{dt.year}, {dt.strftime('%B')}}"
                month_groups[key].append(dt)
            for key, dts in month_groups.items():
                dts_sorted = sorted(dts)
                count = len(dts_sorted)
                start = dts_sorted[0].strftime('%Y-%m-%d')
                end = dts_sorted[-1].strftime('%Y-%m-%d')
                anomaly_filtering_report.append(f"    {key}: [{count}]")
                anomaly_filtering_report.append(f"        from {start} to {end}")
            stats['anomaly_filtering_report'] = anomaly_filtering_report

    if winsorize:
        numeric_cols = [col for col in df.columns if
pd.api.types.is_numeric_dtype(df[col]) and not
pd.api.types.is_bool_dtype(df[col])]
        for col in numeric_cols:
            df[col] = _winsorize_series(df[col], lower=winsor_limits[0],
upper=winsor_limits[1])
        stats['winsorized_columns'] = numeric_cols
        stats['winsorize'] = winsorize
        stats['winsor_limits'] = winsor_limits

    if yeo_johnson:
        from sklearn.preprocessing import PowerTransformer
        numeric_cols = [col for col in df.columns if
pd.api.types.is_numeric_dtype(df[col]) and not
pd.api.types.is_bool_dtype(df[col])]
        pt = PowerTransformer(method='yeo-johnson')
        df[numeric_cols] = pt.fit_transform(df[numeric_cols])
        stats['yeo_johnson_transformed'] = numeric_cols

    num_cols = [f for f in NORMALIZE_FACTORS if f in df.columns]
    if normalization == 'minmax':
        from sklearn.preprocessing import MinMaxScaler
        scaler = MinMaxScaler()
        df[num_cols] = scaler.fit_transform(df[num_cols])
    elif normalization == 'standard':
        from sklearn.preprocessing import StandardScaler
        scaler = StandardScaler()
        df[num_cols] = scaler.fit_transform(df[num_cols])

```

```

elif normalization == 'robust':
    from sklearn.preprocessing import RobustScaler
    scaler = RobustScaler()
    df[num_cols] = scaler.fit_transform(df[num_cols])
stats['normalization'] = normalization

if encoding == 'label':
    from sklearn.preprocessing import LabelEncoder
    cat_cols = [col for col in ENCODE_FACTORS if col in df.columns]
    for col in cat_cols:
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col].astype(str))
elif encoding == 'onehot':
    cat_cols = [col for col in ENCODE_FACTORS if col in df.columns]
    df = pd.get_dummies(df, columns=cat_cols, drop_first=False,
dtype=float)
    stats['encoding'] = encoding

unified_missing_warning = None
valid_range = None
drop_cols = None
task_success = None
if target_task in ['classification', 'regression']:
    feature_cols = [col for col in df.columns if col not in ['date',
'target']]
    nan_mask = df[feature_cols].isnull().any(axis=1)
    has_nan = df[feature_cols].isnull().any().any()
    if has_nan:
        unified_missing_warning = f"Missing values detected in features.
Most {target_task} models do not support NaN. Fill missing values before
training."
        valid_idx = (~nan_mask).to_numpy().nonzero()[0]
        if len(valid_idx) > 0:
            start_idx, end_idx = valid_idx[0], valid_idx[-1]
            valid_range = (str(df.iloc[start_idx]['date']),
str(df.iloc[end_idx]['date']))
            drop_cols = [col for col in feature_cols if
df[col].isnull().any()]
            task_success = True if target_task == 'regression' else not
has_nan
        stats['unified_missing_warning'] = unified_missing_warning
        stats['unified_valid_range'] = valid_range
        stats['unified_drop_columns'] = drop_cols
        stats['unified_task_success'] = task_success

    if target_task == 'classification':
        feature_cols = [col for col in df.columns if col not in ['date',
'target']]
        nan_mask = df[feature_cols].isnull().any(axis=1)
        if df[feature_cols].isnull().any().any():
            cols_with_nan = [col for col in feature_cols if
df[col].isnull().any()]
            stats['classification_nan_error'] = {'message': "Missing values
detected in features. Classification with class balancing cannot proceed.
Please fill missing values before balancing.", 'cols_with_nan':
cols_with_nan}
            if stats.get('df') is not None:
                df2 = stats['df']
                nan_mask2 = df2[[col for col in df2.columns if col not in
['date', 'target']]].isnull().any(axis=1)
                valid_idx = (~nan_mask2).to_numpy().nonzero()[0]
                if len(valid_idx) > 0:
                    start_idx, end_idx = valid_idx[0], valid_idx[-1]

```

```

        stats['classification_nan_error']['valid_date_range'] =
(str(df2.iloc[start_idx]['date']), str(df2.iloc[end_idx]['date']))
        stats['classification_nan_error']['drop_columns'] = cols_with_nan
    elif df[feature_cols].isnull().any().any():
        cols_with_nan = [col for col in feature_cols if
df[col].isnull().any()]
        stats['classification_nan_warning'] = {'message': "After
interpolation, some features still contain NaN. Class balancing may not work
correctly.", 'cols_with_nan': cols_with_nan}
        stats['classification_nan_warning']['drop_columns'] =
cols_with_nan
    elif target_task == 'regression':
        feature_cols = [col for col in df.columns if col not in ['date',
'target']]
        nan_mask = df[feature_cols].isnull().any(axis=1)
        if df[feature_cols].isnull().any().any():
            stats['regression_warning'] = "Missing values detected in
features. Most regression models do not support NaN. Please fill missing
values before training."
            valid_idx = (~nan_mask).to_numpy().nonzero()[0]
            if len(valid_idx) > 0:
                start_idx, end_idx = valid_idx[0], valid_idx[-1]
                stats['regression_valid_range'] = (str(df.iloc[start_idx]
['date']), str(df.iloc[end_idx]['date']))
                cols_with_nan = [col for col in feature_cols if
df[col].isnull().any()]
                stats['regression_drop_columns_suggestion'] = cols_with_nan

    selected_target = None
    target_col_replaced = None
    if target_task != 'none':
        possible_targets = [f for f in TARGET_FACTORS if f in df.columns]
        if target_col and target_col in possible_targets and not
stats.get('target_col_all_nan', False):
            selected_target = target_col
        elif possible_targets:
            selected_target = smart_select_target(df, task_hint=target_task
if target_task != 'none' else None)
            if orig_target_col and orig_target_col != selected_target:
                target_col_replaced = selected_target
        if selected_target:
            if target_task == 'regression':
                df['target'] = df[selected_target].shift(-target_horizon)
            elif target_task == 'classification':
                df['target'] = (df[selected_target].shift(-target_horizon) >
df[selected_target]).astype(int)
            stats['target_column'] = selected_target
            stats['target_task'] = target_task
            stats['target_horizon'] = target_horizon
            stats['target_col_replaced'] = target_col_replaced
            stats['orig_target_col'] = orig_target_col

    if target_task == 'classification' and balance_classes and 'target' in
df.columns:
        feature_cols = [col for col in df.columns if col not in ['date',
'target']]
        balance = df['target'].value_counts()
        max_count = balance.max()
        min_count = balance.min()
        ratio = min_count / max_count if max_count > 0 else 0
        if ratio > 0.8:
            stats['class_balance_error'] = "Classes are already well balanced
(ratio > 0.8), balancing skipped."

```

```

else:
    try:
        if balance_classes == 'oversample':
            from imblearn.over_sampling import SMOTE
            smote = SMOTE(k_neighbors=k_neighbors if k_neighbors is
not None else 1, random_state=42)
            X = df[feature_cols]
            y = df['target']
            X_res, y_res = smote.fit_resample(X, y)
            df = pd.concat([pd.DataFrame(X_res,
columns=feature_cols), pd.Series(y_res, name='target')], axis=1)
        elif balance_classes == 'undersample':
            from imblearn.under_sampling import RandomUnderSampler
            rus = RandomUnderSampler(random_state=42)
            X = df[feature_cols]
            y = df['target']
            X_res, y_res = rus.fit_resample(X, y)
            df = pd.concat([pd.DataFrame(X_res,
columns=feature_cols), pd.Series(y_res, name='target')], axis=1)
            stats['class_balance'] =
df['target'].value_counts().to_dict()
        except Exception as e:
            stats['class_balance_error'] = str(e)

    stats['feature_stats'] = {}
    for col in df.columns:
        if (col in ENCODE_FACTORS or (stats.get('encoding') == 'onehot' and
col.startswith('day_of_week_') or col.startswith('us_holiday_'))):
            continue
        if pd.api.types.is_numeric_dtype(df[col]):
            col_data = df[col]
            stats['feature_stats'][col] = {
                'mean': float(col_data.mean(skipna=True)) if not
col_data.isnull().all() else None,
                'std': float(col_data.std(skipna=True)) if not
col_data.isnull().all() else None,
                'median': float(col_data.median(skipna=True)) if not
col_data.isnull().all() else None,
                'skew': float(col_data.skew(skipna=True)) if not
col_data.isnull().all() else None,
                'kurtosis': float(col_data.kurtosis(skipna=True)) if not
col_data.isnull().all() else None,
                'missing_pct': float(col_data.isnull().mean() * 100)
            }

    stats['export_format'] = export_format
    stats['final_rows'] = len(df)
    stats['final_columns'] = list(df.columns)
    stats['df'] = df
    return df, stats, show_boxplot_stats

def _format_date_ranges(dates: list) -> list:
    if not dates:
        return []
    dates_sorted = sorted(pd.to_datetime(dates))
    ranges = []
    start = dates_sorted[0]
    end = start
    for curr in dates_sorted[1:]:
        if (curr - end).days == 1:
            end = curr
        else:
            if start == end:

```

```

        ranges.append(start.strftime('%Y-%m-%d'))
    else:
        ranges.append(f"from {start.strftime('%Y-%m-%d')} to
{end.strftime('%Y-%m-%d')}}")
        start = end = curr
    if start == end:
        ranges.append(start.strftime('%Y-%m-%d'))
    else:
        ranges.append(f"from {start.strftime('%Y-%m-%d')} to
{end.strftime('%Y-%m-%d')}}")
    return ranges

def generate_report(stats: Dict[str, Any], show_boxplot_stats: bool = False,
yeo_johnson: bool = False, winsorize: bool = False, balance_classes: str =
None, k_neighbors: int = 1) -> str:
    def format_stat(val):
        if isinstance(val, (float, int)):
            return f"{val:10.4f}"
        elif isinstance(val, list):
            return ', '.join(map(str, val))
        else:
            return f"{str(val):>10}"

    report = []
    report.append("==== DATA PROCESSING REPORT =====")
    report.append('')
    report.append(f"Start Date: {stats.get('start_date')}")
    report.append(f"End Date: {stats.get('end_date')}")
    report.append(f"Initial Rows: {stats.get('initial_rows')}")
    report.append(f"Initial Columns: {' '.join(stats.get('initial_columns',
[]))}")
    report.append(f"Interpolate: {stats.get('interpolate')}")
    report.append(f"Anomaly filter: {stats.get('anomaly_filter')}")
    report.append(f"Anomaly method: {stats.get('anomaly_method')}")
    report.append(f"Rows After Anomaly Filtering:
{stats.get('rows_after_anomaly')}")
    report.append(f"Normalization: {stats.get('normalization')}")
    report.append(f"Encoding: {stats.get('encoding')}")
    report.append(f"Target Task: {stats.get('target_task')}")
    report.append(f"Target Horizon: {stats.get('target_horizon')}")
    report.append(f"Show boxplot stats: {show_boxplot_stats}")
    if stats.get('winsorized_columns'):
        report.append(f"Winsorization: True (limits:
{stats.get('winsor_limits')}")
        report.append(f"Columns winsorized: {' '.join(stats['winsorized_columns'])}")
    else:
        report.append("Winsorization: False")
    report.append(f"Yeo-Johnson transform: {yeo_johnson}")
    if stats.get('yeo_johnson_transformed'):
        report.append(f"Columns Yeo-Johnson transformed: {' '.join(stats['yeo_johnson_transformed'])}")
    report.append(f"Class balancing: {balance_classes}")
    if stats.get('target_task') == 'classification' and balance_classes:
        report.append(f"Number of neighbors for SMOTE: {k_neighbors}")
    if stats.get('target_task') == 'classification' and balance_classes:
        if stats.get('class_balance'):
            balance = stats['class_balance']
            classes = list(balance.keys())
            counts = list(balance.values())
            max_count = max(counts)
            min_count = min(counts)
            ratio = min_count / max_count if max_count > 0 else 0

```

```

        if ratio > 0.8:
            conclusion = "Classes are well balanced. No further balancing
is needed."
            recommendation = "You can proceed with model training."
        elif ratio > 0.5:
            conclusion = "Moderate class imbalance detected."
            recommendation = "Consider using oversampling for improved
performance."
        else:
            conclusion = "Severe class imbalance detected!"
            recommendation = "Oversampling (e.g., SMOTE) is strongly
recommended. Undersampling may result in loss of valuable data."
            report += [' ', "Class balance summary:", f" {dict(balance)}", f"
Conclusion: {conclusion}", f" Recommendation: {recommendation}"]
        else:
            reason = stats.get('class_balance_error', None)
            df_ = stats.get('df', None)
            feature_cols = [col for col in df_.columns if col not in ['date',
'target']] if df_ is not None else []
            min_count = None
            ratio = None
            if df_ is not None and 'target' in df_.columns:
                vc = df_['target'].value_counts()
                if not vc.empty:
                    min_count = vc.min()
                    max_count = vc.max()
                    ratio = min_count / max_count if max_count > 0 else None
            if df_ is not None and any(df_[feature_cols].isnull().any()):
                fail_reason = "Missing values in features prevent class
balancing (e.g. SMOTE)."
            elif ratio is not None and ratio > 0.8:
                fail_reason = "Classes are already well balanced (ratio >
0.8), balancing skipped."
            elif min_count is not None and min_count < 2:
                fail_reason = "Too few samples in the minority class for
SMOTE."
            elif reason:
                fail_reason = reason
            else:
                fail_reason = "Unknown reason. Possibly classes are already
balanced or an internal error occurred."
            logger = logging.getLogger("class_balance")
            logger.warning(f"Class balancing was not performed. Reason:
{fail_reason}")
            target_counts = df_['target'].value_counts().to_dict() if df_ is
not None and 'target' in df_.columns else {}
            recommendation = "Check logs for details, consider collecting
more data, or using undersampling if oversampling is not possible."
            report += [' ', "Class balance summary (before/without
balancing):", f" {target_counts}", f" Conclusion: Class balancing was not
performed. Reason: {fail_reason}", f" Recommendation: {recommendation}"]
            if stats.get('anomaly_counts'):
                report += [' ', 'Anomaly Counts:']
                for col, count in stats['anomaly_counts'].items():
                    report.append(f" {col}: {count}")
            if stats.get('anomaly_filtering_report'):
                report += [' ', 'Detailed anomaly filtering report:']
                for line in stats['anomaly_filtering_report']:
                    report.append(f" {line}")
            report.append(' ')
            if stats.get('target_column'):
                report.append(f"Target column used: {stats['target_column']}")
            report.append(' ')

```

```

if stats.get('unified_missing_warning'):
    report.append(f"[WARNING] {stats['unified_missing_warning']}")
if stats.get('unified_valid_range'):
    start, end = stats['unified_valid_range']
    report.append(f"Valid date subrange with no NaN in features: {start}
to {end}")
if stats.get('unified_drop_columns'):
    drop_cols = stats['unified_drop_columns']
    if drop_cols:
        report.append(f"Alternatively, drop these columns to remove all
NaN: {' , '.join(drop_cols)}")
    if stats.get('target_task') == 'classification':
        if stats.get('unified_task_success') is False:
            report.append(f"[WARNING] Classification task performed with NaN
in features. Fill missing values before training.")
        else:
            report.append("Classification task performed successfully.")
    elif stats.get('target_task') == 'regression':
        report.append("Regression task performed successfully.")
    report += [' ', 'Advanced Feature Statistics:', f" {Column':50}
{'Mean':>10} {'Std':>10} {'Median':>10} {'Skew':>10} {'Kurtosis':>10} {'%
Missing':>10}", ' ' + '-'*110]
    feat_stats = stats.get('feature_stats', {})
    for col, vals in feat_stats.items():
        report.append(f" {col:50} {format_stat(vals['mean'])}
{format_stat(vals['std'])} {format_stat(vals['median'])}
{format_stat(vals['skew'])} {format_stat(vals['kurtosis'])}
{format_stat(vals['missing_pct'])}%")
        report.append(' ')
    if stats.get('initial_columns'):
        if 'us_holiday' and 'day_of_week' in stats.get('initial_columns'):
            report.append('Categorical features (us_holiday and day_of_week)
is not included here.')
        elif 'day_of_week' in stats.get('initial_columns'):
            report.append('Categorical features (day_of_week) is not included
here.')
        else:
            report.append('Categorical features (us_holiday) is not included
here.')

if show_boxplot_stats:
    boxplot_lines = []
    for col, vals in feat_stats.items():
        if vals.get('kurtosis') is not None and vals['kurtosis'] > 1:
            try:
                col_data = stats.get('df', None)[col] if stats.get('df',
None) is not None else None
            except Exception:
                col_data = None
            if col_data is None:
                continue
            desc = col_data.describe(percentiles=[.25, .5, .75])
            q1, median, q3 = desc['25%'], desc['50%'], desc['75%']
            iqr = q3 - q1
            lower_whisker = max(desc['min'], q1 - 1.5 * iqr)
            upper_whisker = min(desc['max'], q3 + 1.5 * iqr)
            outliers = col_data[(col_data < lower_whisker) | (col_data >
upper_whisker)]
            boxplot_lines.append(f"Boxplot summary for '{col}':")
            boxplot_lines.append(f" min: {desc['min']:.4f}, Q1:
{q1:.4f}, median: {median:.4f}, Q3: {q3:.4f}, max: {desc['max']:.4f}")
            boxplot_lines.append(f" IQR: {iqr:.4f}, lower whisker:
{lower_whisker:.4f}, upper whisker: {upper_whisker:.4f}")

```

```

        boxplot_lines.append(f" Outliers: {len(outliers)}
({100*len(outliers)/len(col_data):.2f}%)")
    if boxplot_lines:
        report += ['', 'Boxplot statistics for features with high
kurtosis:', *boxplot_lines]

    recommendations = []
    for col, vals in feat_stats.items():
        if vals['missing_pct'] is not None and col == 'target' and
vals['missing_pct'] > 5:
            recommendations.append(f"Column '{col}' has more than 5%
missing values – your optimal Target Horizon is
{math.floor(stats.get('final_rows') * 0.05)}.")
            if vals['missing_pct'] is not None and vals['missing_pct'] > 10:
                if col == 'target':
                    continue
                else:
                    recommendations.append(f"Column '{col}' has more than 10%
missing values – consider imputation or deletion.")
            if vals['std'] is not None and vals['std'] < 1e-6:
                recommendations.append(f"Column '{col}' is almost constant –
consider deletion.")
            if vals['skew'] is not None and abs(vals['skew']) > 1:
                recommendations.append(f"Column '{col}' has strong skewness
(skew={vals['skew']:.2f}) – consider transformation (Yeo-Johnson).")
            if vals['kurtosis'] is not None and vals['kurtosis'] > 1:
                recommendations.append(f"Column '{col}' has leptokurtic
distribution (kurtosis={vals['kurtosis']:.2f}) – outliers are possible, check
boxplot.")
    if recommendations:
        report += ['', 'Recommendations:']
        for r in recommendations:
            report.append(f" - {r}")

    report += ['', f"Final Rows: {stats.get('final_rows')}"]
    excluded = stats.get('all_nan_columns_dropped', [])
    df_full = stats.get('df_full')
    if excluded:
        report += ['', 'Excluded columns due to missing values:']
        for col in excluded:
            line = f" {col}"
            if df_full is not None and col in df_full.columns:
                notnull_idx = df_full[col].notnull()
                if notnull_idx.any():
                    dates = df_full.loc[notnull_idx, 'date']
                    first, last = dates.iloc[0], dates.iloc[-1]
                    line += f" (set from {first} to {last})"
            else:
                line += f" (set from None to None)"
            report.append(line)
        report.append('')
        report.append("Note: This happened because data is not extrapolated,
so missing values remain at the beginning and end of the series.")
        report += [f"Final Columns: {stats.get('final_columns')}"]
        df = stats.get('df')
        if df is not None:
            report += ['', 'First 5 rows of processed DataFrame:']
            if len(df.columns) > 5:
                cols = list(df.columns[:3]) + list(df.columns[-2:])
                df_preview = df[cols].head()
                report.append(df_preview.to_string(index=False))
                report.append(f"[Showing only the first 3 and last 2 columns out
of {len(df.columns)}]")

```

```

        else:
            report.append(df.head().to_string(index=False))
    return '\n'.join(report)

if __name__ == "__main__":
    try:
        params = dict(
            interpolate=True,
            anomaly_filter='strict',
            anomaly_method='zscore',
            normalization='minmax',
            encoding='none',
            target_task='regression',
            target_horizon=54,
            export_format='csv',
            factors=['binance_close', 'kraken_close', 'coinbase_close',
'gold', 'us_holiday', 'day_of_week'],
            start_date='2020-01-01',
            end_date='2023-01-05',
            target_col='binance_close',
            show_boxplot_stats=True,
            yeo_johnson=False,
            winsorize=True,
            winsor_limits=(0.01, 0.99),
            balance_classes='oversample',
            k_neighbors=1
        )
        df, stats, show_boxplot_stats = process_data(**params)
        print(generate_report(
            stats,
            show_boxplot_stats=show_boxplot_stats,
            yeo_johnson=params.get('yeo_johnson', True),
            winsorize=params.get('winsorize', False),
            balance_classes=params.get('balance_classes', None),
            k_neighbors=params.get('k_neighbors', 5)
        ))
        df.to_csv('processed.csv', index=False)
        df.to_json('processed.json', orient='records', force_ascii=False)
    except Exception as e:
        logger.error(f"Processing failed: {e}")
        print(f"[ERROR]: {e}")

```

custom_factors.py

```

from datetime import date
from typing import Dict, Any
import pandas as pd
import numpy as np

def compute_custom_factors(input_df: pd.DataFrame, date_value: date) ->
Dict[str, Any]:
    result = {k: None for k in [

'binance_volatility_3d', 'coinbase_volatility_3d', 'kraken_volatility_3d',

'binance_volatility_7d', 'coinbase_volatility_7d', 'kraken_volatility_7d',

'binance_volatility_14d', 'coinbase_volatility_14d', 'kraken_volatility_14d',

'cross_exchange_volatility_diff_binance_coinbase_3d', 'cross_exchange_volatili
ty_diff_binance_kraken_3d', 'cross_exchange_volatility_diff_coinbase_kraken_3d
',

```

```

'cross_exchange_volatility_diff_binance_coinbase_7d', 'cross_exchange_volatility_diff_binance_kraken_7d', 'cross_exchange_volatility_diff_coinbase_kraken_7d',
',

'binance_volume_to_price_ratio', 'coinbase_volume_to_price_ratio', 'kraken_volume_to_price_ratio',

'binance_volume_to_avg_price_ratio', 'coinbase_volume_to_avg_price_ratio', 'kraken_volume_to_avg_price_ratio',

'price_spread_ratio_binance_coinbase', 'price_spread_ratio_binance_kraken', 'price_spread_ratio_coinbase_kraken',

'volume_spread_ratio_binance_coinbase', 'volume_spread_ratio_binance_kraken', 'volume_spread_ratio_coinbase_kraken',

'volatility_spread_ratio_binance_coinbase_3d', 'volatility_spread_ratio_binance_kraken_3d', 'volatility_spread_ratio_coinbase_kraken_3d',

'bitcoin_sp500_correlation_7d', 'bitcoin_sp500_correlation_14d', 'bitcoin_vix_ratio', 'bitcoin_gold_ratio', 'bitcoin_oil_ratio', 'bitcoin_usd_index_correlation_7d',

'transaction_value_to_volume_ratio', 'hashrate_to_difficulty_ratio', 'transaction_fees_to_price_ratio', 'transactions_to_unique_addresses_ratio', 'avg_block_size_daily', 'avg_confirmation_time_daily',

'rsi_adx_combo', 'sma_20_to_price_ratio', 'adx_strength', 'price_deviation_from_sma_20', 'daily_price_change_pct',

'blockchain_activity_score', 'cross_market_sentiment_index', 'exchange_volume_weighted_price',

'avg_price_all_minus_binance_close', 'avg_price_all_minus_coinbase_close', 'avg_price_all_minus_kraken_close',

'binance_volatility_3d_to_7d_ratio', 'coinbase_volatility_3d_to_7d_ratio', 'kraken_volatility_3d_to_7d_ratio',
    'days_since_last_us_holiday', 'avg_close_5d', 'avg_close_10d']]
    return result

def compute_custom_factors_from_imarketdata(df: pd.DataFrame) ->
pd.DataFrame:
    result = []
    df = df.copy()
    df = df.sort_values('date').reset_index(drop=True)
    for i, row in df.iterrows():
        record = {'date': row['date']}
        for exch in ['binance', 'coinbase', 'kraken']:
            for window in [3, 7, 14]:
                close_col = f'{exch}_close'
                closes = df.loc[max(0, i-window+1):i, close_col] if close_col in df.columns else pd.Series(dtype=float)
                if closes.isnull().any() or len(closes) < window:
                    record[f'{exch}_volatility_{window}d'] = None
                else:
                    record[f'{exch}_volatility_{window}d'] =
closes.pct_change().std() * np.sqrt(window)
            def cross_vol(ex1, ex2, window):
                v1 = record.get(f'{ex1}_volatility_{window}d')
                v2 = record.get(f'{ex2}_volatility_{window}d')
                if v1 is None or v2 is None:

```

```

        return None
    return abs(v1 - v2)
    for window in [3, 7]:
        record[f'cross_exchange_volatility_diff_binance_coinbase_{window}
d'] = cross_vol('binance', 'coinbase', window)
        record[f'cross_exchange_volatility_diff_binance_kraken_{window}
d'] = cross_vol('binance', 'kraken', window)
        record[f'cross_exchange_volatility_diff_coinbase_kraken_{window}
d'] = cross_vol('coinbase', 'kraken', window)
        for exch in ['binance', 'coinbase', 'kraken']:
            v = row.get(f'{exch}_volume')
            c = row.get(f'{exch}_close')
            record[f'{exch}_volume_to_price_ratio'] = v / c if v is not None
and c not in [None, 0] else None
            price_vals = [row.get(f'{exch}_open'), row.get(f'{exch}_high'),
row.get(f'{exch}_low'), row.get(f'{exch}_close')]
            if all(pd.isna(x) or x is None for x in price_vals):
                avg_price = None
            else:
                avg_price = np.nanmean([x for x in price_vals if x is not
None])
            record[f'{exch}_volume_to_avg_price_ratio'] = v / avg_price if v
is not None and avg_price not in [None, 0, np.nan] else None
        def safe_div(a, b):
            if a in [None, 0, np.nan] or b in [None, 0, np.nan]: return None
            return a / b
        record['price_spread_ratio_binance_coinbase'] =
safe_div(row.get('binance_close'), row.get('coinbase_close'))
        record['price_spread_ratio_binance_kraken'] =
safe_div(row.get('binance_close'), row.get('kraken_close'))
        record['price_spread_ratio_coinbase_kraken'] =
safe_div(row.get('coinbase_close'), row.get('kraken_close'))
        record['volume_spread_ratio_binance_coinbase'] =
safe_div(row.get('binance_volume'), row.get('coinbase_volume'))
        record['volume_spread_ratio_binance_kraken'] =
safe_div(row.get('binance_volume'), row.get('kraken_volume'))
        record['volume_spread_ratio_coinbase_kraken'] =
safe_div(row.get('coinbase_volume'), row.get('kraken_volume'))
        record['volatility_spread_ratio_binance_coinbase_3d'] =
safe_div(record['binance_volatility_3d'], record['coinbase_volatility_3d'])
        record['volatility_spread_ratio_binance_kraken_3d'] =
safe_div(record['binance_volatility_3d'], record['kraken_volatility_3d'])
        record['volatility_spread_ratio_coinbase_kraken_3d'] =
safe_div(record['coinbase_volatility_3d'], record['kraken_volatility_3d'])
        def rolling_corr(col1, col2, window):
            if i < window-1: return None
            s1 = df.loc[i-window+1:i, col1]
            s2 = df.loc[i-window+1:i, col2]
            if s1.isnull().any() or s2.isnull().any(): return None
            return s1.corr(s2)
        record['bitcoin_sp500_correlation_7d'] =
rolling_corr('avg_price_all', 'sp500_close', 7)
        record['bitcoin_sp500_correlation_14d'] =
rolling_corr('avg_price_all', 'sp500_close', 14)
        record['bitcoin_vix_ratio'] = safe_div(row.get('avg_price_all'),
row.get('vix_close'))
        record['bitcoin_gold_ratio'] = safe_div(row.get('avg_price_all'),
row.get('gold'))
        record['bitcoin_oil_ratio'] = safe_div(row.get('avg_price_all'),
row.get('oil'))
        record['bitcoin_usd_index_correlation_7d'] =
rolling_corr('avg_price_all', 'usd_index', 7)

```

```

        record['transaction_value_to_volume_ratio'] =
safe_div(row.get('usd_transaction_volume'), row.get('binance_volume') +
row.get('coinbase_volume') + row.get('kraken_volume'))
        record['hashrate_to_difficulty_ratio'] =
safe_div(row.get('hashrate'), row.get('difficulty'))
        record['transaction_fees_to_price_ratio'] =
safe_div(row.get('transaction_fees'), row.get('avg_price_all'))
        record['transactions_to_unique_addresses_ratio'] =
safe_div(row.get('transactions'), row.get('unique_addresses'))
        record['avg_block_size_daily'] = row.get('avg_block_size')
        record['avg_confirmation_time_daily'] =
row.get('avg_confirmation_time')
        rsi = row.get('rsi_14')
        adx = row.get('adx_14')
        record['rsi_adx_combo'] = rsi * adx / 100 if rsi not in [None,
np.nan] and adx not in [None, np.nan] else None
        sma20 = row.get('sma_20')
        close = row.get('binance_close')
        record['sma_20_to_price_ratio'] = safe_div(sma20, close)
        record['adx_strength'] = adx / 100 if adx not in [None, np.nan] else
None
        record['price_deviation_from_sma_20'] = close - sma20 if close not in
[None, np.nan] and sma20 not in [None, np.nan] else None
        record['daily_price_change_pct'] = safe_div((close -
row.get('binance_open')), row.get('binance_open')) * 100 if close not in
[None, np.nan] and row.get('binance_open') not in [None, np.nan, 0] else None
        vals = [row.get('transactions'), row.get('unique_addresses'),
row.get('usd_transaction_volume')]
        vals_norm = [v for v in vals if v not in [None, np.nan]]
        record['blockchain_activity_score'] = sum(vals_norm) / len(vals_norm)
if vals_norm else None
        vals2 = [row.get('sp500_close'), row.get('vix_close'),
row.get('gold'), row.get('oil')]
        vals2_norm = [v for v in vals2 if v not in [None, np.nan]]
        record['cross_market_sentiment_index'] = sum(vals2_norm) /
len(vals2_norm) if vals2_norm else None
        vols = [row.get('binance_volume'), row.get('coinbase_volume'),
row.get('kraken_volume')]
        closes = [row.get('binance_close'), row.get('coinbase_close'),
row.get('kraken_close')]
        if all(v not in [None, np.nan, 0] for v in vols) and all(c not in
[None, np.nan] for c in closes):
            record['exchange_volume_weighted_price'] = sum([c*v for c, v in
zip(closes, vols)]) / sum(vols)
        else:
            record['exchange_volume_weighted_price'] = None
            avg_all = row.get('avg_price_all')
            record['avg_price_all_minus_binance_close'] = avg_all -
row.get('binance_close') if avg_all not in [None, np.nan] and
row.get('binance_close') not in [None, np.nan] else None
            record['avg_price_all_minus_coinbase_close'] = avg_all -
row.get('coinbase_close') if avg_all not in [None, np.nan] and
row.get('coinbase_close') not in [None, np.nan] else None
            record['avg_price_all_minus_kraken_close'] = avg_all -
row.get('kraken_close') if avg_all not in [None, np.nan] and
row.get('kraken_close') not in [None, np.nan] else None
            for exch in ['binance', 'coinbase', 'kraken']:
                v3 = record.get(f'{exch}_volatility_3d')
                v7 = record.get(f'{exch}_volatility_7d')
                record[f'{exch}_volatility_3d_to_7d_ratio'] = safe_div(v3, v7)
            days_since_holiday = None
            if 'us_holiday' in df.columns:

```

```

        prev_holidays = df.loc[:i, ['date',
'us_holiday']].query('us_holiday == True')
        if not prev_holidays.empty:
            days_since_holiday = (row['date'] - prev_holidays.iloc[-1]
['date']).days
            record['days_since_last_us_holiday'] = days_since_holiday
        def avg_close(window):
            closes = df.loc[max(0, i-window+1):i, 'binance_close']
            if closes.isnull().any() or len(closes) < window:
                return None
            return closes.mean()
        record['avg_close_5d'] = avg_close(5)
        record['avg_close_10d'] = avg_close(10)
        result.append(record)
    return pd.DataFrame(result)

if __name__ == "__main__":
    from db import SessionLocal
    from models import IMarketData, CustomFactors
    import sqlalchemy

    session = SessionLocal()
    imd_rows = session.query(IMarketData).order_by(IMarketData.date).all()
    if not imd_rows:
        exit(1)
    df = pd.DataFrame([{'col': getattr(r, col) for col in
IMarketData.__table__.columns.keys()} for r in imd_rows])
    cf_df = compute_custom_factors_from_imarketdata(df)
    session.query(CustomFactors).delete()
    session.commit()
    for col in cf_df.columns:
        cf_df[col] = cf_df[col].astype(object)
        cf_df[col] = cf_df[col].where(pd.notnull(cf_df[col]), None)
    records = cf_df.to_dict(orient='records')
    objs = [CustomFactors(**rec) for rec in records]
    session.bulk_save_objects(objs)
    session.commit()
    session.close()

```

data_interpolation.py

```

import pandas as pd
import numpy as np
from scipy.interpolate import CubicSpline
from sqlalchemy.orm import sessionmaker
from sqlalchemy import create_engine
from models import MarketData, IMarketData, Base
from db import engine

Session = sessionmaker(bind=engine)
session = Session()

def fetch_market_data():
    query = session.query(MarketData)
    df = pd.read_sql(query.statement, session.bind)
    return df

def exp_interpolate(s):
    s = s.copy()
    s[s <= 0] = np.nan
    mask = s.isnull()
    if s.notnull().sum() < 2:
        return s

```

```

        interp = np.exp(np.interp(np.flatnonzero(mask),
np.flatnonzero(~s.isnull()), np.log(s.dropna()))
        s[mask] = interp
        return s

def cubic_spline_interpolate(series):
    s = series.copy()
    mask = s.notnull()
    if mask.sum() < 2:
        return s
    cs = CubicSpline(np.where(mask)[0], s[mask])
    s[~mask] = cs(np.where(~mask)[0])
    return s

def linear_interpolate(series):
    return series.interpolate(method='linear', limit_direction='both',
limit_area='inside')

def rolling_mean(series, window=3):
    return series.fillna(series.rolling(window, min_periods=1,
center=True).mean())

def forward_fill(series):
    return series.fillna(method='ffill')

def interpolate_dataframe(df):
    df = df.sort_values('date').reset_index(drop=True)
    assert df['date'].is_unique, "Дублікати дат у даних!"
    result = pd.DataFrame()
    result['date'] = df['date']
    exp_cols =
['binance_open', 'binance_high', 'binance_low', 'binance_close', 'coinbase_open',
'coinbase_high', 'coinbase_low', 'coinbase_close', 'kraken_open', 'kraken_high', '
kraken_low', 'kraken_close']
    cubic_cols =
['binance_volume', 'coinbase_volume', 'kraken_volume', 'diff_binance_coinbase_vo
lume', 'diff_binance_kraken_volume', 'diff_coinbase_kraken_volume', 'diff_all_vo
lume', 'transactions', 'blocks_mined', 'usd_transaction_volume']
    linear_cols =
['diff_binance_coinbase_open', 'diff_binance_coinbase_high', 'diff_binance_coin
base_low', 'diff_binance_coinbase_close', 'diff_binance_kraken_open', 'diff_bina
nce_kraken_high', 'diff_binance_kraken_low', 'diff_binance_kraken_close', 'diff_
coinbase_kraken_open', 'diff_coinbase_kraken_high', 'diff_coinbase_kraken_low',
'diff_coinbase_kraken_close', 'diff_all_open', 'diff_all_high', 'diff_all_low',
'diff_all_close', 'avg_price_binance_coinbase', 'avg_price_binance_kraken', 'avg_
price_coinbase_kraken', 'avg_price_all', 'unique_addresses', 'transaction_fees',
'avg_block_size', 'avg_confirmation_time', 'sp500_close', 'vix_close', 'us10y', 'u
sd_index', 'gold', 'oil']
    rolling_cols =
['hashrate', 'difficulty', 'sma_20', 'rsi_14', 'atr_14', 'adx_14']
    ffill_cols = ['us_holiday', 'day_of_week']
    mask_kraken = pd.to_datetime(df['date']) > pd.to_datetime('2023-05-18')
    mask_sma20 = pd.to_datetime(df['date']) > pd.to_datetime('2024-06-10')
    mask_rsi14 = pd.to_datetime(df['date']) > pd.to_datetime('2024-06-05')
    mask_atr14 = pd.to_datetime(df['date']) > pd.to_datetime('2024-06-05')
    mask_adx14 = pd.to_datetime(df['date']) > pd.to_datetime('2024-06-18')
    for col in exp_cols:
        if col.startswith('kraken_'):
            s = df[col].copy()
            s[~mask_kraken] = np.nan
            result[col] = exp_interpolate(s)
        else:
            result[col] = exp_interpolate(df[col])

```

```

for col in cubic_cols:
    if 'kraken' in col:
        s = df[col].copy()
        s[~mask_kraken] = np.nan
        result[col] = cubic_spline_interpolate(s)
    else:
        result[col] = cubic_spline_interpolate(df[col])
for col in linear_cols:
    s = df[col].copy()
    s = s.interpolate(method='linear', limit_direction='both',
limit_area='inside')
    result[col] = s
for col in rolling_cols:
    if col == 'sma_20':
        s = df[col].copy()
        s[~mask_sma20] = np.nan
        result[col] = rolling_mean(s, window=3)
    elif col == 'rsi_14':
        s = df[col].copy()
        s[~mask_rsi14] = np.nan
        result[col] = rolling_mean(s, window=3)
    elif col == 'atr_14':
        s = df[col].copy()
        s[~mask_atr14] = np.nan
        result[col] = rolling_mean(s, window=3)
    elif col == 'adx_14':
        s = df[col].copy()
        s[~mask_adx14] = np.nan
        result[col] = rolling_mean(s, window=3)
    else:
        result[col] = rolling_mean(df[col], window=3)
for col in ffill_cols:
    result[col] = forward_fill(df[col])
for col in exp_cols + cubic_cols + linear_cols:
    if col.startswith('kraken_') or 'diff_binance_kraken_' in col or
'diff_coinbase_kraken_' in col:
        result.loc[~mask_kraken, col] = None
result.loc[~mask_sma20, 'sma_20'] = None
result.loc[~mask_rsi14, 'rsi_14'] = None
result.loc[~mask_atr14, 'atr_14'] = None
result.loc[~mask_adx14, 'adx_14'] = None
for col in ['sp500_close', 'vix_close', 'us10y', 'usd_index', 'gold',
'oil']:
    result[col] = pd.to_numeric(result[col], errors='coerce').round(4)
for col in ['transactions', 'blocks_mined']:
    result[col] = pd.to_numeric(result[col],
errors='coerce').round().astype('Int64')
return result

def save_to_db(df):
    session.query(IMarketData).delete()
    session.commit()
    for col in df.columns:
        df[col] = df[col].astype(object)
        df[col] = df[col].where(pd.notnull(df[col]), None)
    records = df.to_dict(orient='records')
    objs = [IMarketData(**rec) for rec in records]
    session.bulk_save_objects(objs)
    session.commit()

def main():
    df = fetch_market_data()
    df_interp = interpolate_dataframe(df)
    save_to_db(df_interp)

```

```
if __name__ == "__main__":
    main()
```

etl.py

```
import datetime
import sys
import time
from db import SessionLocal
from models import MarketData
from data_processing.api_clients import fetch_binance_data,
fetch_coinbase_data, fetch_kraken_data, fetch_blockchain_data,
fetch_fred_data, fetch_main_indicators
import pandas as pd
import logging
import math
import subprocess
from sqlalchemy import func
import os
import holidays

logger = logging.getLogger(__name__)

def safe_val(val):
    if pd.isna(val) or (isinstance(val, float) and math.isnan(val)):
        return None
    return val

def bulk_fetch_and_upsert(start_date, end_date, chunk_days=300):
    all_dates = pd.date_range(start=start_date, end=end_date, freq='D')
    binance_df = pd.DataFrame()
    d = start_date
    while d <= end_date:
        chunk_end = min(d + datetime.timedelta(days=chunk_days-1), end_date)
        df = fetch_binance_data(d, chunk_end + datetime.timedelta(days=1))
        if not df.empty:
            binance_df = pd.concat([binance_df, df])
            d = chunk_end + datetime.timedelta(days=1)
            time.sleep(0.1)
    binance_df.index = pd.to_datetime(binance_df.index)
    binance_df = binance_df[~binance_df.index.duplicated(keep='first')]

    coinbase_df = pd.DataFrame()
    d = start_date
    while d <= end_date:
        chunk_end = min(d + datetime.timedelta(days=30-1), end_date)
        df = fetch_coinbase_data(d, chunk_end + datetime.timedelta(days=1))
        if not df.empty:
            coinbase_df = pd.concat([coinbase_df, df])
            d = chunk_end + datetime.timedelta(days=1)
            time.sleep(2)
    coinbase_df.index = pd.to_datetime(coinbase_df.index)
    coinbase_df = coinbase_df[~coinbase_df.index.duplicated(keep='first')]

    kraken_df = pd.DataFrame()
    d = start_date
    while d <= end_date:
        chunk_end = min(d + datetime.timedelta(days=30-1), end_date)
        df = fetch_kraken_data(d, chunk_end + datetime.timedelta(days=1))
        if not df.empty:
            kraken_df = pd.concat([kraken_df, df])
            d = chunk_end + datetime.timedelta(days=1)
```

```

        time.sleep(2)
        kraken_df.index = pd.to_datetime(kraken_df.index)
        kraken_df = kraken_df[~kraken_df.index.duplicated(keep='first')]

        blockchain_df = fetch_blockchain_data(start_date, end_date +
datetime.timedelta(days=1))
        blockchain_df.index = pd.to_datetime(blockchain_df.index)
        blockchain_df =
blockchain_df[~blockchain_df.index.duplicated(keep='first')]

        fred_df = fetch_fred_data(start_date, end_date)
        fred_df.index = pd.to_datetime(fred_df.index)
        fred_df = fred_df[~fred_df.index.duplicated(keep='first')]
        fred_df.index = fred_df.index.tz_localize(None)

        for dsource in [binance_df, coinbase_df, kraken_df, blockchain_df,
fred_df]:
            dsource.index = pd.to_datetime(dsource.index).tz_localize(None)

        df = pd.DataFrame(index=all_dates)
        for dsource in [binance_df, coinbase_df, kraken_df, blockchain_df,
fred_df]:
            for col in dsource.columns:
                df[col] = dsource[col]
        df.index = pd.to_datetime(df.index)
        df = df.sort_index()

        df['avg_price_binance_coinbase'] = df[['binance_close',
'coinbase_close']].mean(axis=1, skipna=True)
        df['avg_price_binance_kraken'] = df[['binance_close',
'kraken_close']].mean(axis=1, skipna=True)
        df['avg_price_coinbase_kraken'] = df[['coinbase_close',
'kraken_close']].mean(axis=1, skipna=True)
        df['avg_price_all'] = df[['binance_close', 'coinbase_close',
'kraken_close']].mean(axis=1, skipna=True)

        ohlcv_cols = ['open', 'high', 'low', 'close', 'volume']
        exchanges = ['binance', 'coinbase', 'kraken']
        for i in range(len(exchanges)):
            for j in range(i+1, len(exchanges)):
                ex1, ex2 = exchanges[i], exchanges[j]
                for col in ohlcv_cols:
                    col1 = f'{ex1}_{col}'
                    col2 = f'{ex2}_{col}'
                    diff_col = f'diff_{ex1}_{ex2}_{col}'
                    df[diff_col] = (df[col1] - df[col2]).abs()
        for col in ohlcv_cols:
            cols = [f'{ex}_{col}' for ex in exchanges]
            diff_col = f'diff_all_{col}'
            df[diff_col] = df[cols].max(axis=1, skipna=True) -
df[cols].min(axis=1, skipna=True)

        session = SessionLocal()
        for i, (date, row) in enumerate(df.iterrows()):
            data = {'date': date.date()}
            data['binance_open'] = safe_val(row.get('binance_open', None))
            data['binance_high'] = safe_val(row.get('binance_high', None))
            data['binance_low'] = safe_val(row.get('binance_low', None))
            data['binance_close'] = safe_val(row.get('binance_close', None))
            data['binance_volume'] = safe_val(row.get('binance_volume', None))
            data['coinbase_open'] = safe_val(row.get('coinbase_open', None))
            data['coinbase_high'] = safe_val(row.get('coinbase_high', None))
            data['coinbase_low'] = safe_val(row.get('coinbase_low', None))

```

```

data['coinbase_close'] = safe_val(row.get('coinbase_close', None))
data['coinbase_volume'] = safe_val(row.get('coinbase_volume', None))
data['kraken_open'] = safe_val(row.get('kraken_open', None))
data['kraken_high'] = safe_val(row.get('kraken_high', None))
data['kraken_low'] = safe_val(row.get('kraken_low', None))
data['kraken_close'] = safe_val(row.get('kraken_close', None))
data['kraken_volume'] = safe_val(row.get('kraken_volume', None))
data['hashrate'] = safe_val(row.get('hashrate', None))
data['transactions'] = safe_val(row.get('transactions', None))
data['unique_addresses'] = safe_val(row.get('unique_addresses',
None))
    data['usd_transaction_volume'] =
safe_val(row.get('usd_transaction_volume', None))
    data['transaction_fees'] = safe_val(row.get('transaction_fees',
None))
    data['avg_block_size'] = safe_val(row.get('avg_block_size', None))
    data['avg_confirmation_time'] =
safe_val(row.get('avg_confirmation_time', None))
    data['difficulty'] = safe_val(row.get('difficulty', None))
    data['blocks_mined'] = safe_val(row.get('blocks_mined', None))
    data['avg_price_binance_coinbase'] =
safe_val(row.get('avg_price_binance_coinbase', None))
    data['avg_price_binance_kraken'] =
safe_val(row.get('avg_price_binance_kraken', None))
    data['avg_price_coinbase_kraken'] =
safe_val(row.get('avg_price_coinbase_kraken', None))
    data['avg_price_all'] = safe_val(row.get('avg_price_all', None))
    data['diff_binance_coinbase_open'] =
safe_val(row.get('diff_binance_coinbase_open', None))
    data['diff_binance_coinbase_high'] =
safe_val(row.get('diff_binance_coinbase_high', None))
    data['diff_binance_coinbase_low'] =
safe_val(row.get('diff_binance_coinbase_low', None))
    data['diff_binance_coinbase_close'] =
safe_val(row.get('diff_binance_coinbase_close', None))
    data['diff_binance_coinbase_volume'] =
safe_val(row.get('diff_binance_coinbase_volume', None))
    data['diff_binance_kraken_open'] =
safe_val(row.get('diff_binance_kraken_open', None))
    data['diff_binance_kraken_high'] =
safe_val(row.get('diff_binance_kraken_high', None))
    data['diff_binance_kraken_low'] =
safe_val(row.get('diff_binance_kraken_low', None))
    data['diff_binance_kraken_close'] =
safe_val(row.get('diff_binance_kraken_close', None))
    data['diff_binance_kraken_volume'] =
safe_val(row.get('diff_binance_kraken_volume', None))
    data['diff_coinbase_kraken_open'] =
safe_val(row.get('diff_coinbase_kraken_open', None))
    data['diff_coinbase_kraken_high'] =
safe_val(row.get('diff_coinbase_kraken_high', None))
    data['diff_coinbase_kraken_low'] =
safe_val(row.get('diff_coinbase_kraken_low', None))
    data['diff_coinbase_kraken_close'] =
safe_val(row.get('diff_coinbase_kraken_close', None))
    data['diff_coinbase_kraken_volume'] =
safe_val(row.get('diff_coinbase_kraken_volume', None))
    data['diff_all_open'] = safe_val(row.get('diff_all_open', None))
    data['diff_all_high'] = safe_val(row.get('diff_all_high', None))
    data['diff_all_low'] = safe_val(row.get('diff_all_low', None))
    data['diff_all_close'] = safe_val(row.get('diff_all_close', None))
    data['diff_all_volume'] = safe_val(row.get('diff_all_volume', None))
    data['sp500_close'] = safe_val(row.get('sp500_close', None))

```

```

data['vix_close'] = safe_val(row.get('vix_close', None))
data['usl0y'] = safe_val(row.get('usl0y', None))
data['usd_index'] = safe_val(row.get('usd_index', None))
data['gold'] = safe_val(row.get('gold', None))
data['oil'] = safe_val(row.get('oil', None))
obj = session.query(MarketData).filter_by(date=date.date()).first()
if obj:
    for k, v in data.items():
        setattr(obj, k, v)
else:
    obj = MarketData(**data)
    session.add(obj)
session.commit()
session.close()

if __name__ == "__main__":
    session = SessionLocal()
    last_row =
session.query(MarketData).order_by(MarketData.date.desc()).first()
session.close()
today = datetime.date.today()
if last_row is not None and last_row.date is not None:
    start_date = last_row.date + datetime.timedelta(days=1)
else:
    start_date = datetime.date(2020, 1, 1)
end_date = today - datetime.timedelta(days=1)
if (end_date - start_date).days >= 1:
    bulk_fetch_and_upsert(start_date, end_date)

session = SessionLocal()
last_gold =
session.query(func.max(MarketData.date)).filter(MarketData.gold !=
None).scalar()
session.close()
gold_csv_path = os.path.join(os.path.dirname(__file__),
"gold_h_data.csv")
gold_df = pd.read_csv(gold_csv_path)
gold_df["Date"] = pd.to_datetime(gold_df["Date"], format="%m/%d/%Y")
max_csv_date = gold_df["Date"].max().date()
if not last_gold or max_csv_date > last_gold:
    subprocess.run(["python", "scripts/import_gold_history.py"],
check=True)

session = SessionLocal()
need_update = session.query(MarketData).filter(
    (MarketData.sma_20 == None) |
    (MarketData.rsi_14 == None) |
    (MarketData.atr_14 == None) |
    (MarketData.adx_14 == None)
).count()
session.close()
if need_update > 0:
    btc_indicators = fetch_main_indicators("BTCUSD")
    session = SessionLocal()
    updated = 0
    for idx, row in btc_indicators.iterrows():
        date = idx.date()
        obj = session.query(MarketData).filter_by(date=date).first()
        if obj:
            if "sma_20" in row and pd.notnull(row["sma_20"]): obj.sma_20
= float(row["sma_20"])
            if "rsi_14" in row and pd.notnull(row["rsi_14"]): obj.rsi_14
= float(row["rsi_14"])

```

```

        if "atr_14" in row and pd.notnull(row["atr_14"]): obj.atr_14
= float(row["atr_14"])
        if "adx_14" in row and pd.notnull(row["adx_14"]): obj.adx_14
= float(row["adx_14"])
        updated += 1
        session.commit()
        session.close()

    session = SessionLocal()
    us_holidays = holidays.US(years=range(2020, datetime.date.today().year +
2))
    updated = 0
    for row in session.query(MarketData).all():
        if row.date is not None:
            dow = row.date.weekday()
            if row.day_of_week != dow:
                row.day_of_week = dow
            is_holiday = row.date in us_holidays
            if row.us_holiday != is_holiday:
                row.us_holiday = is_holiday
            updated += 1
    session.commit()
    session.close()

    subprocess.run([sys.executable, "data_interpolation.py"], check=True)
    subprocess.run([sys.executable, "custom_factors.py"], check=True)

```

Додаток 3
Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО”



ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

**Вебзастосунок для агрегації та очищення даних біткоїна для
ринкового прогнозування**

Виконав: студент групи КП-03 Фадєєв Олексій Олександрович

Керівник: доцент кафедри ПЗКС, к.т.н. Юрчишин Василь Якович

Київ – 2025



ПОСТАНОВКА ЗАДАЧІ

Мета проекту: розробити вебзастосунок для агрегації та очищення даних біткоїна для ринкового прогнозування.

Завдання:

1. Проаналізувати існуючі аналоги, та проблематику предметної області.
2. Обрати технологію для розроблення ПЗ, та розробити всі складові програмного забезпечення.
3. Протестувати та проаналізувати розроблене ПЗ.

2



АКТУАЛЬНІСТЬ

1. Зростання популярності криптовалют, зокрема біткоїна, у фінансовому світі.
2. Потреба в адаптивному інструменті для обробки даних.
3. Потреба в якісних даних для прогнозування.
4. Дефіцит гнучких рішень для індивідуальних користувачів.
5. Обмеження наявних рішень:
 - а) Обмежена функціональність.
 - б) Висока вартість і орієнтація на інституції.

3

ІСНУЮЧІ АНАЛОГИ



CoinGecko



CryptoCompare



4

ЗАСОБИ РЕАЛІЗАЦІЇ

1. Мова програмування:

- а) Python
- б) JavaScript



2. Розгортання:

- а) Render



Render

3. Фреймворк:

- а) Flask



Flask

4. База даних:

- а) PostgreSQL (SQLAlchemy)



PostgreSQL



5. Бібліотеки:

- а) Шаблонізація: Jinja2



Jinja



pandas



NumPy

- б) Аналіз даних: Pandas, NumPy

- в) Машинне навчання: Scikit-learn, imbalanced-learn



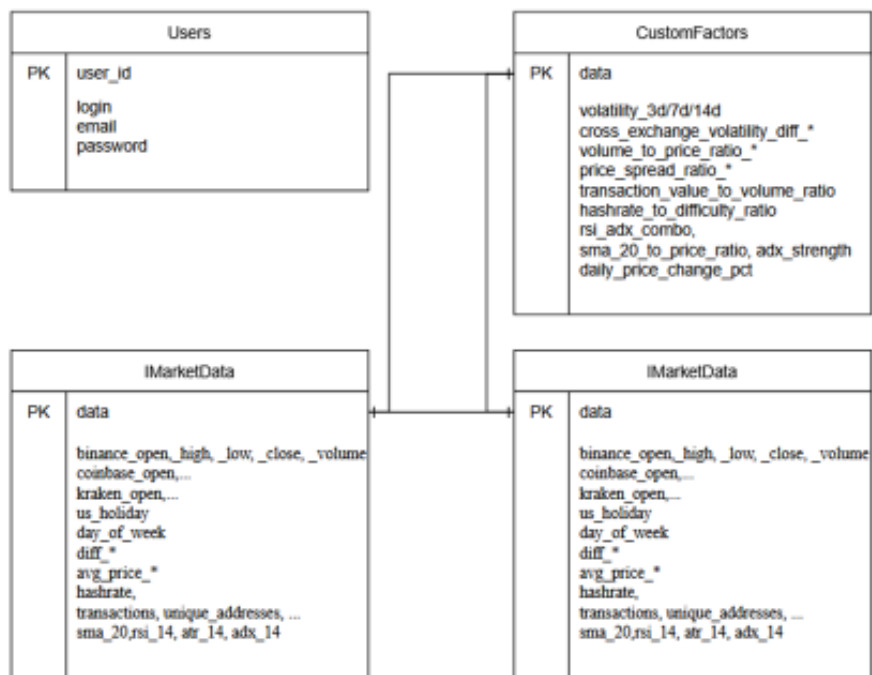
learn



learn

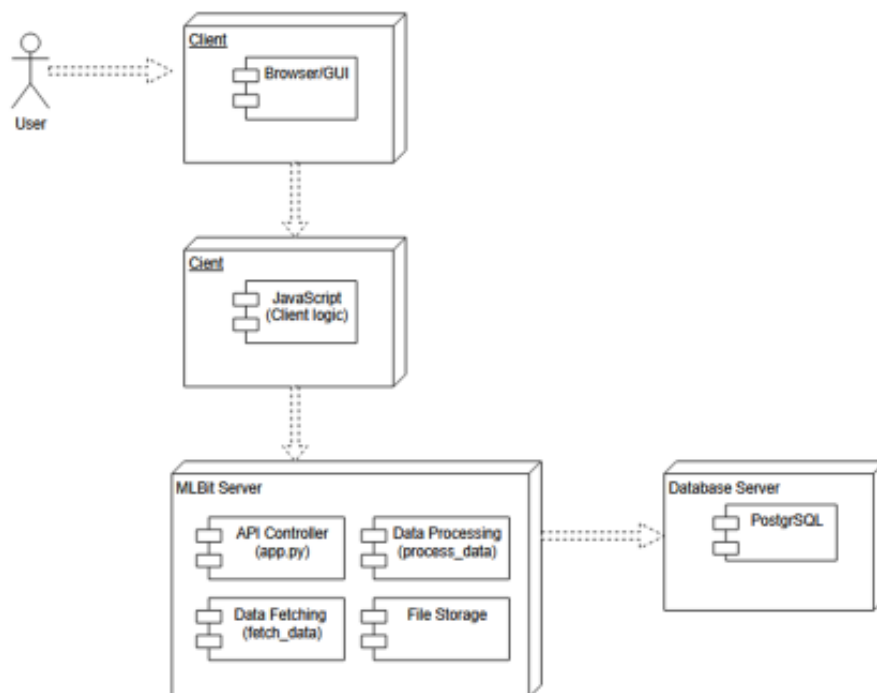
5

СТРУКТУРА БАЗИ ДАНИХ



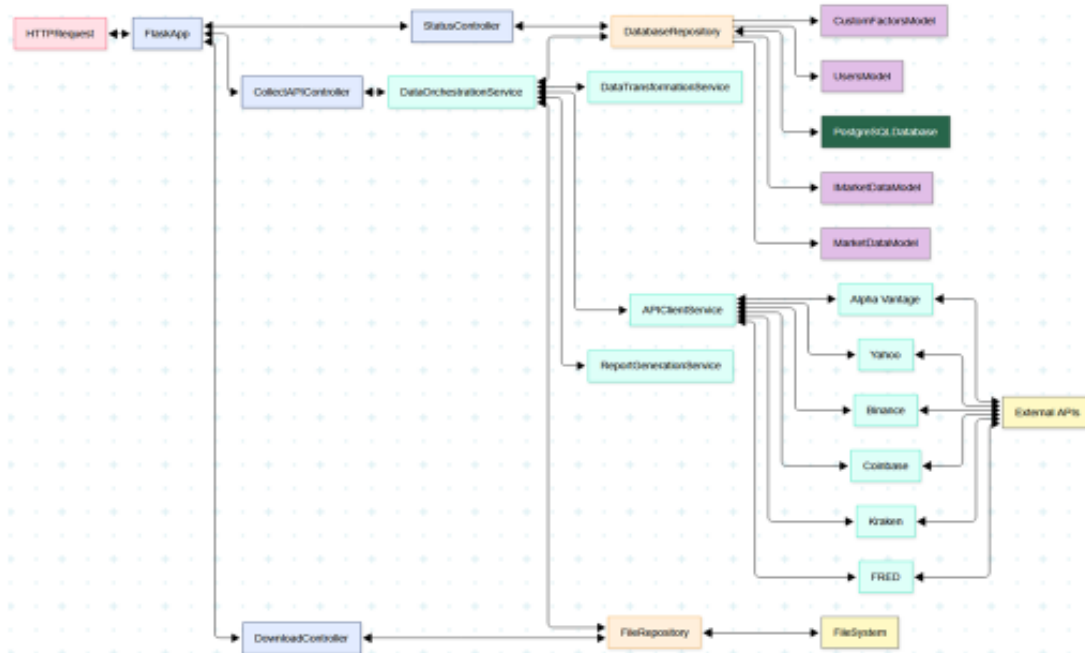
6

ДІАГРАМА МОДУЛІВ СИСТЕМИ



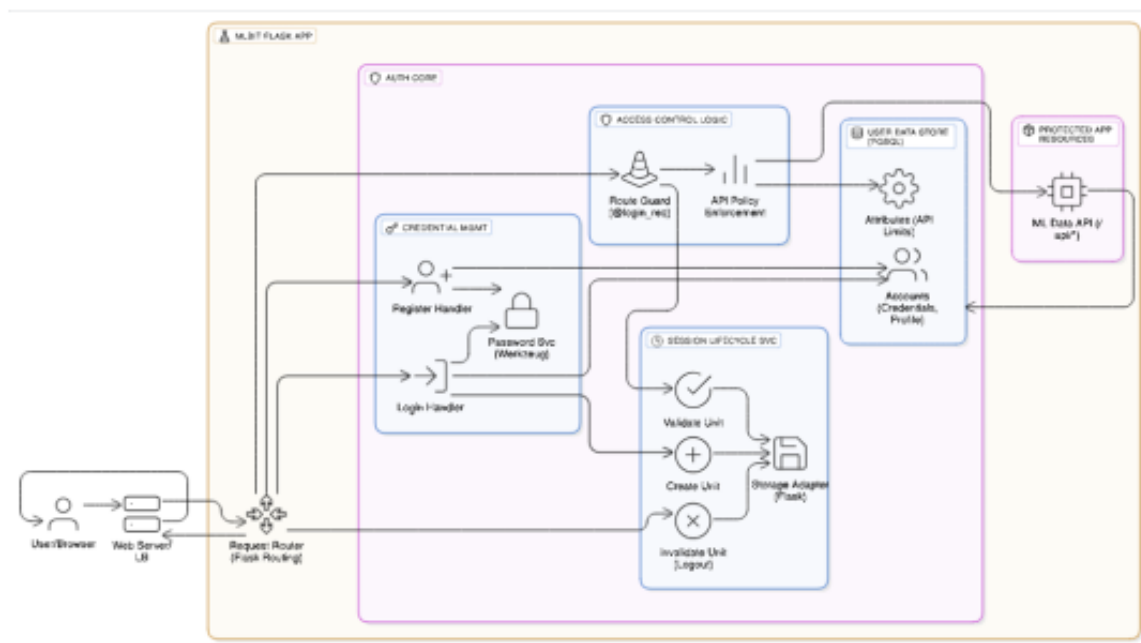
7

ДІАГРАМА КЛАСІВ



8

СХЕМА АРХІТЕКТУРИ АВТЕНТИФІКАЦІЇ FLASK-LOGIN



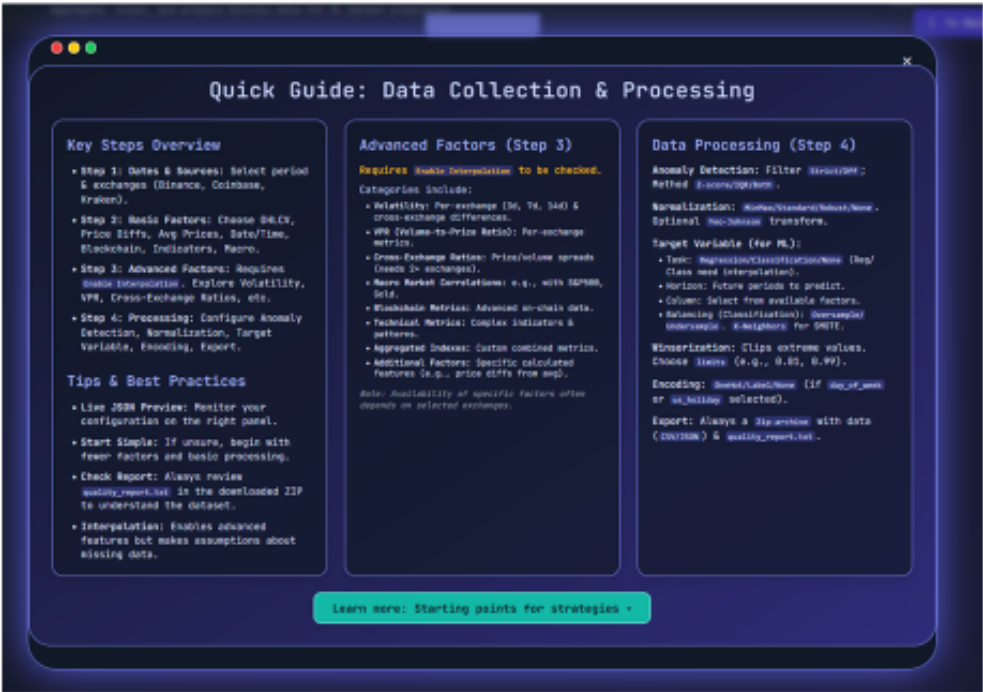
9

ПРИКЛАД РОБОТИ ЗАСТОСУНКУ



10

ПРИКЛАД РОБОТИ ЗАСТОСУНКУ



11

ПРИКЛАД РОБОТИ ЗАСТОСУНКУ



12

ПРИКЛАД РОБОТИ ЗАСТОСУНКУ



13

ПРИКЛАД РОБОТИ ЗАСТОСУНКУ



Bitcoin Data Aggregator
All-in-one tool for Bitcoin data aggregation, cleaning, and ML-ready export

1. Data & Sources 2. Factors 3. Advanced Factors 4. Processing

Factors
Choose additional features to enrich your dataset: temporal, market, network, and external factors.

Differences

☒ Select all

Factor	Between
open	☒ Binance & Kraken
high	☐ Binance & Kraken
low	☐ Binance & Kraken
close	☐ Binance & Kraken
volume	☐ Binance & Kraken

Averages

Date/Time Features

Blockchain

Indicators

Macro/Markets

Back **Next**

14

ПРИКЛАД РОБОТИ ЗАСТОСУНКУ



Bitcoin Data Aggregator
All-in-one tool for Bitcoin data aggregation, cleaning, and ML-ready export

1. Data & Sources 2. Factors 3. Advanced Factors 4. Processing

Advanced Factors
Select advanced features grouped by type. Some require specific exchanges or multiple exchanges.

☒ Enable interpolation (required to select advanced factors)

Volatility

☒ Select all

Full	Reliability Factor	Binance	Coinbase	Kraken
☐	☒	☐	☐	☐
☐	volatility_3d	☐	☐	☐
☐	volatility_7d	☒	☐	☐
☐	volatility_14d	☐	☐	☐

Cross-exchange volatility difference

Binance & Coinbase (3d) ☐

Binance & Coinbase (7d) ☒

Volume-to-Price Ratio

Cross-Exchange Ratios

Macro Market Correlations

Blockchain Metrics

Technical Metrics

Aggregated Indexes

Additional Factors

Back **Next**

15

ПРИКЛАД РОБОТИ ЗАСТОСУНКУ



16

РЕЗУЛЬТАТ РОБОТИ ЗАСТОСУНКУ. ЧАСТИНА ЗВІТУ



Target column used: binance_volatility_3d

Regression task performed successfully.

Advanced Feature Statistics:

Column	Mean	Std	Median	Skew	Kurtosis	% Missing
binance_open	0.3512	0.3036	0.2295	0.6549	-0.8881	0.0000%
coinbase_open	0.3515	0.3036	0.2295	0.6549	-0.8879	0.0000%
binance_volatility_3d	0.2700	0.2300	0.2075	1.1536	0.8316	0.0000%
cross_exchange_volatility_diff_binance_coinbase_7d	0.2644	0.2317	0.1951	1.1278	0.6681	0.0000%
target	0.2701	0.2300	0.2071	1.1417	0.8671	0.1074%

Categorical features (us_holiday) is not included here.

Recommendations:

- Column "binance_volatility_3d" has strong skewness (skew=1.13) - consider transformation (Yeo-Johnson).
- Column "cross_exchange_volatility_diff_binance_coinbase_7d" has strong skewness (skew=1.13) - consider transformation (Yeo-Johnson).
- Column "target" has strong skewness (skew=1.14) - consider transformation (Yeo-Johnson).

Final Rows: 1067

Final Columns: ["date", "binance_open", "coinbase_open", "binance_volatility_3d", "cross_exchange_volatility_diff_binance_coinbase_7d", "target"]

First 5 rows of processed DataFrame:

date	binance_open	coinbase_open	cross_exchange_volatility_diff_binance_coinbase_7d	target
2022-05-04	0.243013	0.243052		0.272026 0.000000
2022-05-06	0.229522	0.229486		0.222259 0.257755
2022-05-07	0.223337	0.223311		0.211411 0.778730
2022-05-08	0.217126	0.217102		0.177277 0.953677
2022-05-09	0.200676	0.200757		0.267978 0.605250

[Showing only the first 3 and last 2 columns out of 6]

17

РЕЗУЛЬТАТ РОБОТИ ЗАСТОСУНКУ. ДАНІ У ФОРМАТІ CSV



1876	2025-04-26,0.895937644578676,0.8971511847362025,0.28316236912485715,0.1127664299605976
1877	2025-04-27,0.8958151261899576,0.8966789604716917,0.29667793232829653,0.13668889492988012
1878	2025-04-28,0.8857347699631775,0.886510589237237,0.1127664299605976,0.1387144367265697
1879	2025-04-29,0.9002108776870685,0.9009928983942902,0.13668889492988012,0.2019747067434563
1880	2025-04-30,0.891557035733654,0.8921396401783764,0.1387144367265697,0.17957340037328462
1881	2025-05-01,0.8905840004792811,0.8911236981580325,0.2019747067434563,0.2715598879806616
1882	2025-05-02,0.9171744085918796,0.9180046771110876,0.17957340037328462,0.25447905176785857
1883	2025-05-03,0.9217313384916026,0.9226503769653268,0.2715598879806616,0.24805882153277714
1884	2025-05-04,0.9099072738012637,0.9104363508732366,0.25447905176785857,0.2330308073557799
1885	2025-05-05,0.8917955337640526,0.8921563946723298,0.24805882153277714,0.1643696820625402
1886	2025-05-06,0.8970274624184003,0.897451847572902,0.2330308073557799,0.10342769616956186
1887	2025-05-07,0.9211220729603631,0.9216638126878055,0.1643696820625402,0.22129292460121797
1888	2025-05-08,0.9233760456882474,0.924125231460465,0.10342769616956186,0.30157668456315473
1889	2025-05-09,0.9948578861017654,0.9952041064355807,0.22129292460121797,
1890	2025-05-10,0.9915354330202819,0.9921600673492098,0.30157668456315473,
1891	Ctrl+L to Cascade, Ctrl+I to Command

18

ШЛЯХИ РОЗВИТКУ



1. Розширення джерел даних



2. Впровадження інтерактивних дашбордів



3. Оптимізація продуктивності



4. Розробка публічного API



5. Поглиблення обробки даних



6. Створення документації та спільноти



7. Додавання даних про деривативи



19



ВИСНОВКИ

1. Проаналізовано аналіз програмних рішень.
2. Спроектовано архітектуру системи та схему бази даних.
3. Розроблено серверну та клієнтську частини застосунку.
4. Розроблено та протестовано ПЗ для агрегації та очищення даних біткоїна для ринкового прогнозування.
5. Представлені напрямки подальшого вдосконалення.

Розробка виконана у повному обсязі згідно з положенням Технічного завдання, тестування продукту проведено відповідно до затвердженої програми та методики тестування.



Дякую за увагу!

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“ ____ ” _____ 2024 р.

ВЕБЗАСТОСУНОК ДЛЯ АГРЕГАЦІЇ ТА ОЧИЩЕННЯ ДАНИХ
БІТКОЇНА ДЛЯ РИНКОВОГО ПРОГНОЗУВАННЯ

Програма та методика тестування

ДП.045440-04-51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Василь ЮРЧИШИН

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Олексій ФАДЄЄВ

2024

ЗМІСТ

1.Об'єкт випробувань.....	3
2.Мета тестування	3
3.Методи тестування	4
4.Засоби та порядок тестування	4

1.

ОБ'ЄКТ ВИПРОБУВАНЬ

Об'єктом випробувань є вебзастосунок MLBit, розроблений для збору та очистки ринкових даних Bitcoin з аналізом якості отриманих даних для задач машинного навчання в напрямку прогнозування ринкових трендів. Система розроблена з використанням мови програмування Python, вебфреймворку Flask, бібліотек Pandas і NumPy для обробки даних, SQLAlchemy для взаємодії з базою даних PostgreSQL, а також спеціалізованих бібліотек (yfinance, fredapi, requests) для роботи з API та обробки фінансових і блокчейн-даних.

2.

МЕТА ТЕСТУВАННЯ

Метою тестування є забезпечення високої якості, надійності та відповідності вебзастосунку MLBit технічним вимогам. У процесі тестування перевіряється:

- 1) коректність роботи всіх функціональних модулів (збір, обробка, зберігання даних);
- 2) відповідність функціональним вимогам (точність обробки даних, коректність API-ендпоінтів) і нефункціональним вимогам (продуктивність, безпека);
- 3) коректність окремих компонентів (модулі збору даних, обробки, експорту), швидкодія при обробці великих обсягів даних;
- 4) точність і консистентність даних із зовнішніх джерел (Binance, Blockchain.com, FRED);
- 5) коректність інтеграції між компонентами (API, база даних, модулі обробки);
- 6) повнота сценаріїв користувача (збір, обробка, завантаження звітів);
- 7) функціональність API-ендпоінтів (/api/collect, /api/status, /api/download);
- 8) рівень захисту даних і системи від вразливостей (SQL-ін'єкції, XSS).

3.

МЕТОДИ ТЕСТУВАННЯ

Тестування проводиться методом Gray Box Testing, що передбачає аналіз як коду, так і функціональності продукту. Використовуються:

- 1) функціональне тестування (Critical path test для перевірки основних сценаріїв);
- 2) модульне тестування (перевірка окремих функцій і компонентів);
- 3) інтеграційне тестування (взаємодія між модулями);
- 4) тестування продуктивності (Stability testing, Load testing);
- 5) тестування безпеки (перевірка на вразливості);
- 6) тестування зручності інтерфейсу (E2E-сценарії).

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується з використанням інструментів pytest, Pandera, Great Expectations, Selenium, Locust, OWASP ZAP, Postman і Requests. Процедура тестування включає:

- 1) модульне тестування функцій збору (api_clients.py), трансформації (transformer.py, processing.py) та експорту (exporter.py) через pytest;
- 2) валідація даних за допомогою Pandera (схеми DataFrame) і Great Expectations (статистика даних);
- 3) інтеграційне тестування взаємодії між app.py, api_clients.py, processing.py та PostgreSQL (SQLAlchemy);
- 4) тестування E2E-сценаріїв користувача через Selenium (перехід між main.html і collect.html, заповнення форм);
- 5) тестування API-ендпоінтів через Requests і Postman (обробка помилок, HTTP-статуси, JSON-відповіді);
- 6) тестування продуктивності (Locust, час відповіді менше ніж 3 секунди), безпеки (OWASP ZAP, SQL-ін'єкції, XSS, захист API-ключів).

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“ ____ ” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ АГРЕГАЦІЇ ТА ОЧИЩЕННЯ ДАНИХ
БІТКОЇНА ДЛЯ РИНКОВОГО ПРОГНОЗУВАННЯ

Керівництво користувача

ДП.045440-05-34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Василь ЮРЧИШИН

Нормоконтроль:

_____ Микола ОНАЙ

____ ” _____ 2011 р.

Виконавець:

_____ Олексій ФАДЄЄВ

2025

ЗМІСТ

1.Опис структури вебзастосунку	3
2.Сторінка з інформацією про проєкт	4
3.Головна сторінка вебзастосунку	5
4.Процедура авторизації та реєстрації користувача	6
5.Процедура створення набору даних	8
6.Рор-уп повідомлення для новачків системи.....	14

1. Опис структури вебзастосунку

Вебзастосунок MLBit призначений для збору, обробки та аналізу ринкових даних для досліджень у сфері машинного навчання. Він реалізований з метою надання зручного інтерфейсу для взаємодії з даними та їх підготовки. Структура вебзастосунку включає наступні основні сторінки:

- 1.Сторінка входу (Login Page): призначена для авторизації існуючих користувачів у системі.
- 2.Сторінка реєстрації (Registration Page): дозволяє новим користувачам створити обліковий запис у системі.
- 3.Головна сторінка (Main Page/Dashboard): це центральна сторінка застосунку, яка відображається після успішної авторизації. Звідси користувач може отримати доступ до основних функцій, таких як ініціювання збору даних. Деталі сторінки описані в розділі 3.
- 4.Сторінка збору даних (Collect Data Page): на цій сторінці користувач може налаштувати параметри для збору та обробки даних. Це включає вибір джерел даних, часового діапазону, факторів для аналізу та параметрів попередньої обробки. Деталі цієї сторінки будуть описані в розділі 5.
- 5.Сторінка "Про продукт" (About Product): містить загальну інформацію про проєкт MLBit, його мету, основні можливості та технології, що використовуються. Деталі цієї сторінки будуть описані в розділі 2.
- 6.Сторінка відновлення паролю (Password recovery page): призначена для відновлення паролю для існуючих користувачів у системі.

2. Сторінка з інформацією про проєкт

Сторінка "Про продукт" (About Product) (рис. 1) існує для того, щоб надати користувачам стислу, але вичерпну інформацію про вебзастосунок MLBit. Вона виконує роль інформаційного центру, де можна швидко ознайомитися з основною концепцією, цілями та можливостями платформи.

На сторінці "Про продукт" користувач дізнається про MLBit як платформу для агрегації, трансформації та експорту Bitcoin-даних, її переваги (AI/ML-готовність, модульність, хмарне розгортання), основні функції (збір, обробка, експорт), а також про автора та місію.

Таким чином, сторінка "Про продукт" допомагає користувачеві швидко зорієнтуватися, зрозуміти цінність MLBit та його відповідність своїм потребам, особливо якщо він взаємодіє з системою вперше.

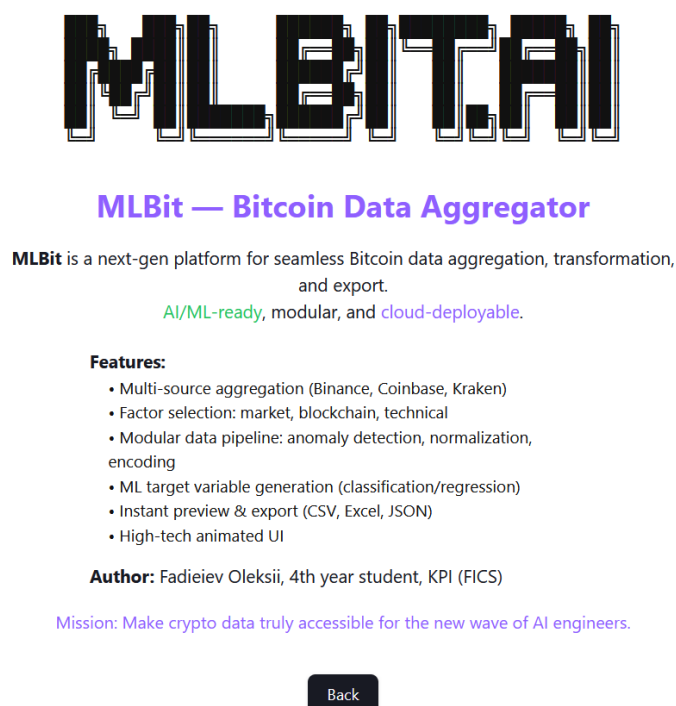


Рис. 1. Сторінка "Про продукт" (About Product)

3. Головна сторінка вебзастосунку

Головна сторінка (Main Page/Dashboard) (рис. 2) є ключовою точкою входу для користувачів, пропонуючи сучасний технологічний дизайн, що підкреслює інноваційність інструментів аналізу даних. Вона оформлена з використанням темної теми, фону "Матриця" з ефектом "матричного дощу" та CRT-ефекту з тонкими горизонтальними лініями, які імітують старий ЕПТ-монітор, створюючи атмосферу роботи з передовими технологіями.

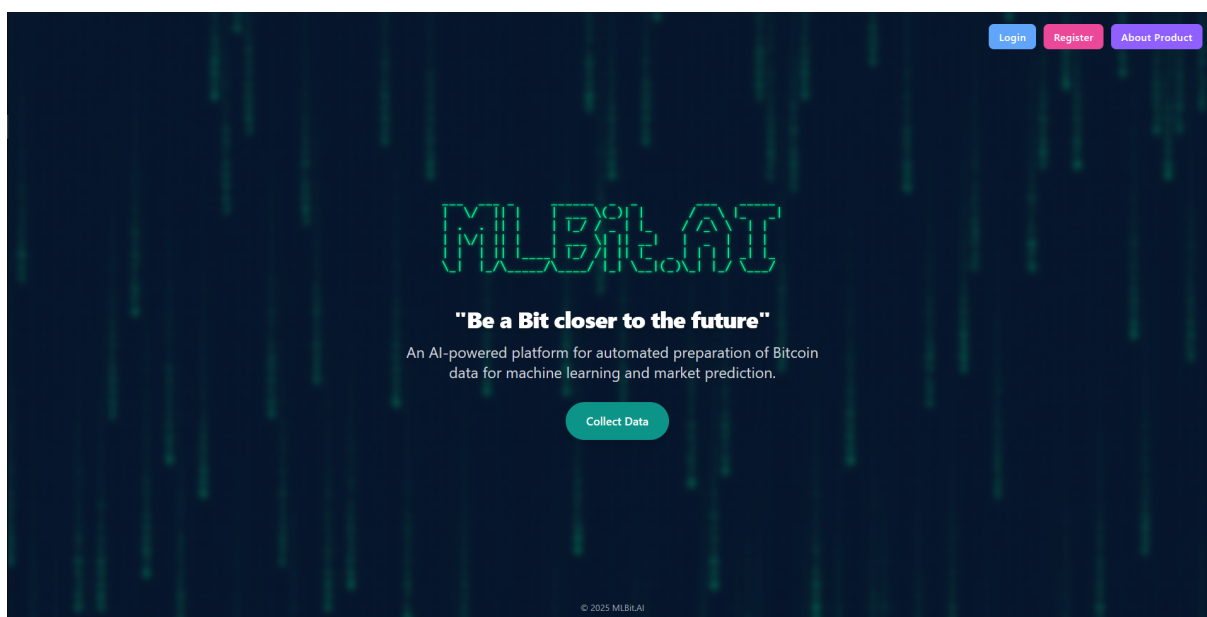


Рис. 2. Головна сторінка (Main Page/Dashboard)

У центрі сторінки розташований ASCII-арт логотип "MLBIT.AI", що привертає увагу, разом із гаслом "Be a Bit closer to the future" та описом платформи. Основна кнопка "Collect Data" веде до функціоналу збору та обробки даних (/collect), забезпечуючи швидкий доступ до ключових можливостей системи. У футері розміщено копірайт "© 2025 MLBit.AI".

Навігаційні елементи включають блок кнопок у верхньому правому куті: "Login" (/login), "Register" (/register) та "About Product".

4. Процедура авторизації та реєстрації користувача

Для доступу до розширених функцій вебзастосунку MLBit, зокрема до можливості збору та обробки даних, користувачу необхідно мати обліковий запис та увійти в систему. Якщо облікового запису немає, його можна створити через процедуру реєстрації.

Сторінка входу (Login Page) (рис. 3) призначена для авторизованих користувачів із зареєстрованими акаунтами. Користувач переходить на неї через кнопку "Login" на Головній сторінці, де заповнює поля: "Ім'я користувача або Email" (Username or Email) та "Пароль" (Password). Натиснувши "Увійти" (Login), при успішному введенні даних його перенаправляють на Головну сторінку. Якщо пароль забуто, доступне посилання "Забули пароль?" (Forgot your password?), а для нових користувачів – "Немає акаунту? Зареєструватися" (Don't have an account? Register).

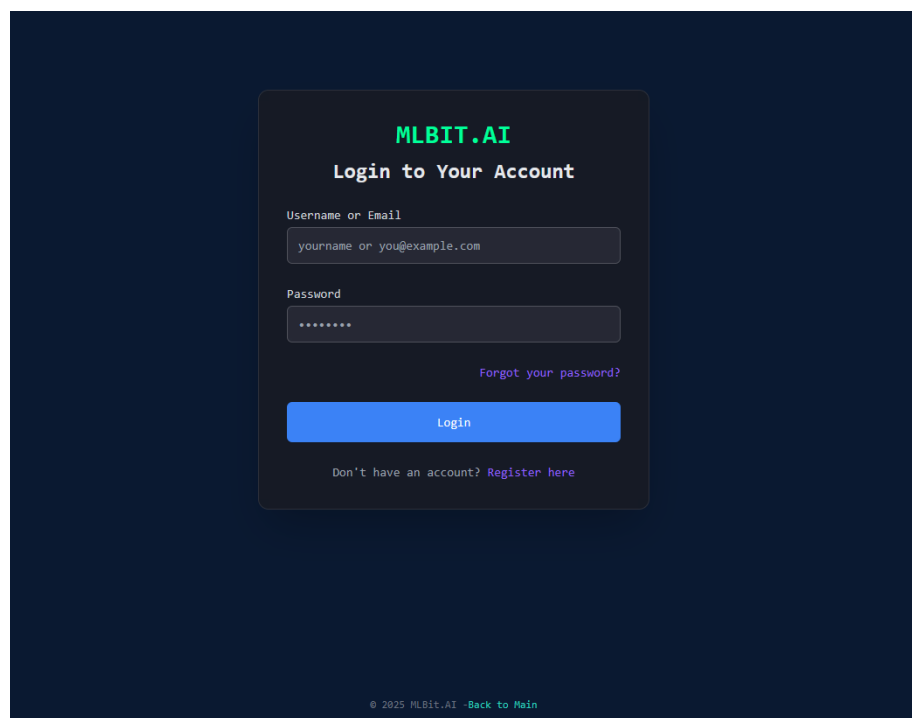


Рис. 3. Сторінка входу (Login Page)

Для нових користувачів системи MLBit створення облікового запису є обов'язковим кроком для доступу до функціоналу. Сторінка реєстрації (Registration Page) (рис. 4) відкривається кнопкою "Register" на Головній сторінці або посиланням на сторінці Входу. На сторінці користувач

заповнює поля: "Ім'я користувача" (Username), "Адреса електронної пошти" (Email), "Пароль" (Password) та "Підтвердіть пароль" (Confirm Password). Після введення даних і валідації (перевірка складності пароля, збігу паролів, унікальності імені та email) натискають "Зареєструватися" (Register). У разі успіху користувача автоматично авторизують і перенаправляють на Головну сторінку. Для тих, хто вже має акаунт, доступне посилання "Вже є акаунт? Увійти" (Already have an account? Login).

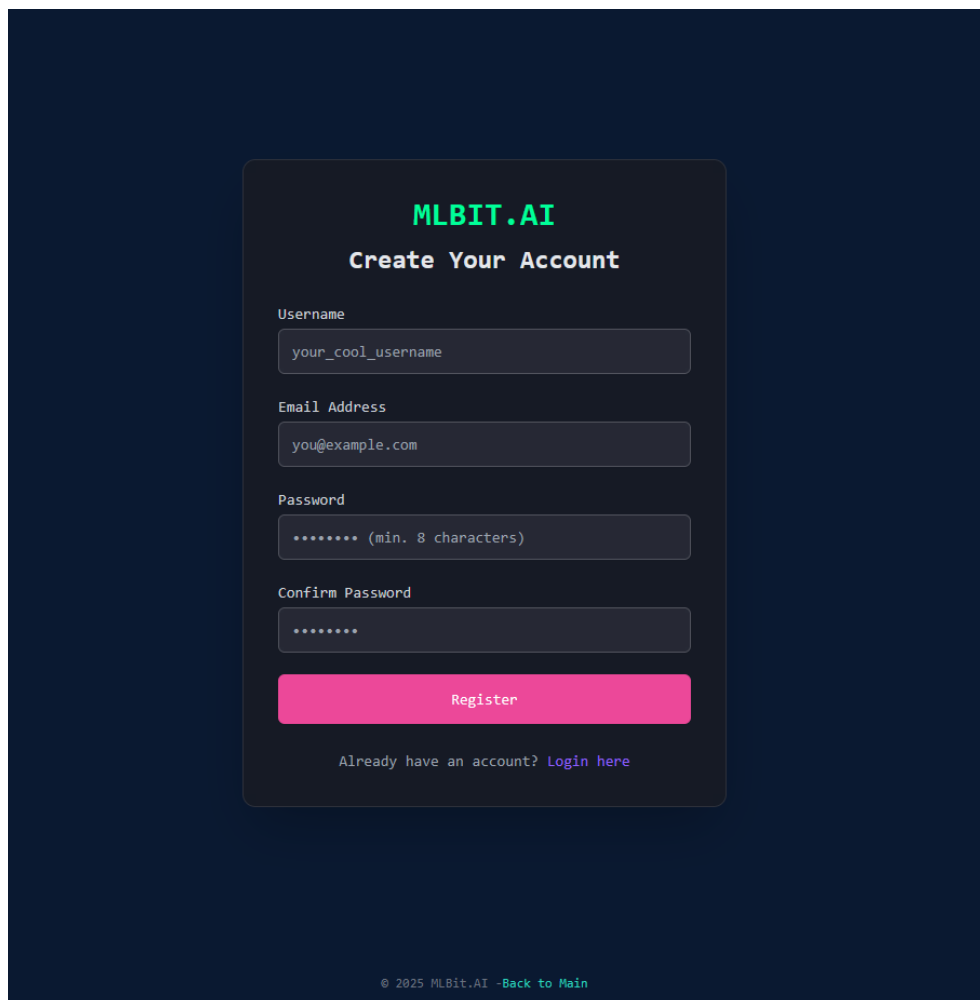
The image shows a registration form for MLBIT.AI. The form is centered on a dark blue background. It has a title "MLBIT.AI Create Your Account" in green and white. Below the title are four input fields: "Username" with the placeholder "your_cool_username", "Email Address" with the placeholder "you@example.com", "Password" with a strength indicator "(min. 8 characters)", and "Confirm Password" with a placeholder "*****". A bright pink "Register" button is positioned below the input fields. At the bottom of the form, there is a link "Already have an account? Login here" in purple. The footer of the page contains the text "© 2025 MLBit.AI -Back to Main" in small green font.

Рис. 4. Сторінка реєстрації (Registration Page)

Якщо користувач забув пароль, функція відновлення дозволяє повернути доступ. Сторінка відновлення паролю (Password recovery page) відкривається через кнопку "Забули пароль?" (Forgot your password?) що знаходиться на сторінці Входу (рис. 5), що перенаправляє на, де вводиться "Адреса електронної пошти" (Email), пов'язана з акаунтом. Натиснувши

"Надіслати інструкції" (Send Reset Link), система відправляє лист з інструкціями для скидання паролю.

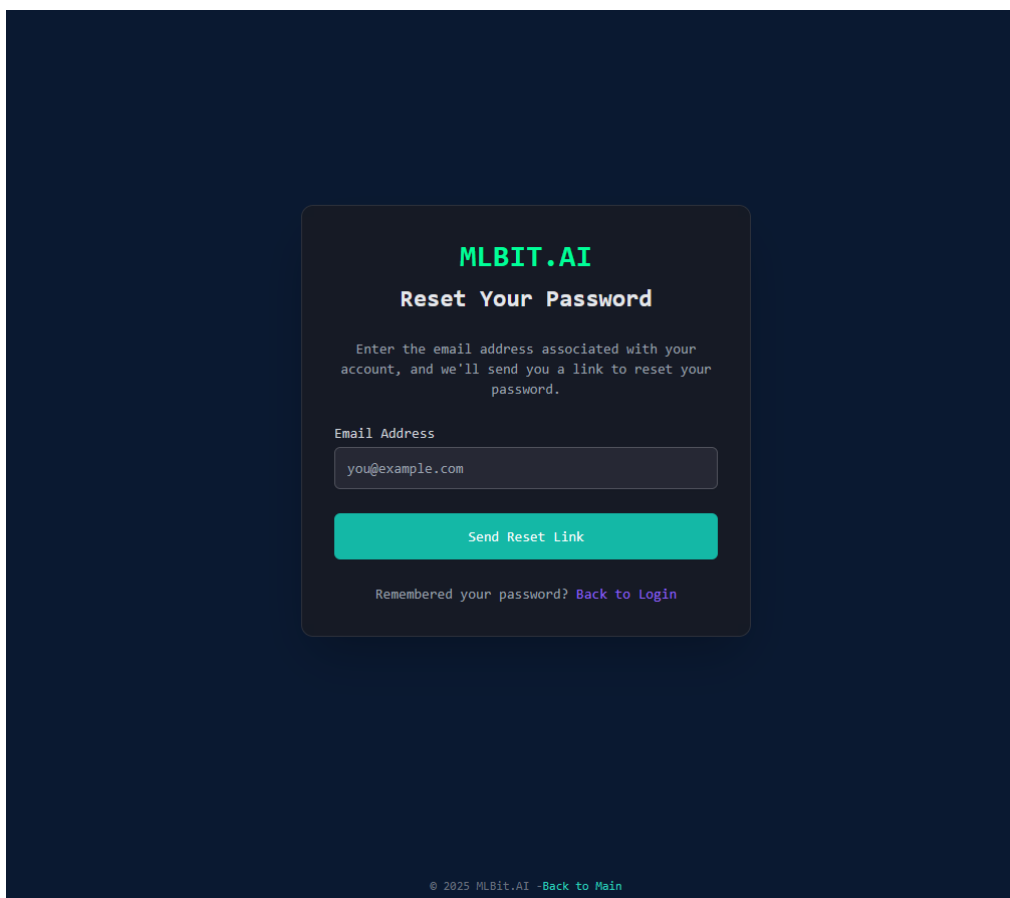


Рис. 5. Сторінка відновлення паролю (Password recovery page)

5. Процедура створення набору даних

Сторінка збору даних у вебзастосунку MLBit є центральним інструментом для роботи з ринковими, макроекономічними та даними, призначеними для задач машинного навчання. Вона розроблена у форматі покрокового майстра (wizard), що дозволяє користувачу послідовно налаштувати параметри збору, обробки та експорту даних. Інтерфейс користувача виконаний у темному "технологічному" стилі, узгодженому з дизайном усього застосунку, з акцентом на зручність і функціональність.

На першому кроці (рис. 6) користувач визначає базові параметри збору даних. Діапазон дат задається через поля "Start Date" (дата початку) та "End Date" (дата кінця), які обираються за допомогою інтерактивного

календаря. Наприклад, можна обрати період із 01.01.2025 по 30.04.2025 для аналізу історичних даних Bitcoin. Далі користувач обирає джерела даних (Exchanges), такі як Binance, Coinbase і Kraken, із можливістю вибору кількох бірж одночасно. Вибір бірж впливає на доступність факторів на наступних етапах: наприклад, якщо обрано лише Binance, то крос-біржові різниці (Binance-Coinbase) будуть недоступні. Після налаштування користувач натискає "Next" для переходу до наступного кроку.

The screenshot shows the 'Bitcoin Data Aggregator' web application. At the top, the title 'Bitcoin Data Aggregator' is displayed with the subtitle 'Aggregate, clean, and prepare Bitcoin data for ML market prediction'. There are two buttons in the top right: 'User Guide' and '< To Main'. The main content area has a dark blue background with a white border. It features a progress bar with four steps: '1. Date & Sources', '2. Factors', '3. Advanced Factors', and '4. Processing'. The first step, 'Date & Sources', is active. Below the progress bar, there is a section titled 'Date & Sources' with the instruction 'Select the date range and exchanges to aggregate data from.' This section contains two input fields for 'Start Date' (2025-01-01) and 'End Date' (2025-04-30). Below these fields is a section titled 'Data Sources' with three radio button options: 'Binance' (selected, with the description 'High liquidity exchange'), 'Coinbase' (with the description 'US-based exchange'), and 'Kraken' (with the description 'European exchange'). At the bottom right of the main content area is a 'Next >' button.

Рис. 6. Сторінка збору даних. Крок 1

Другий крок (рис. 7) дозволяє обрати основні типи даних (фактори) для включення до датасету, згруповані в акордеони для зручності. Групи включають: ohlcv (open, high, low, close, volume) для кожної біржі (наприклад, binance_open, coinbase_close), differences (різниці цін між біржами, наприклад, binance_coinbase_close_diff, доступні при виборі ≥ 2 бірж), average prices (середні ціни, наприклад, avg_price_all), date/time features (часові ознаки: day_of_week, us_holiday, що впливають на кодування на кроці 4), blockchain data (hashrate, transaction_count), indicators

(технічні індикатори: sma_20, rsi_14), macro/markets (макроекономічні дані: s&p500_close, gold_price). Кожна група має опцію "select all" для швидкого вибору всіх пунктів. Попередження відображаються, якщо певні фактори недоступні через вибір на кроці 1.

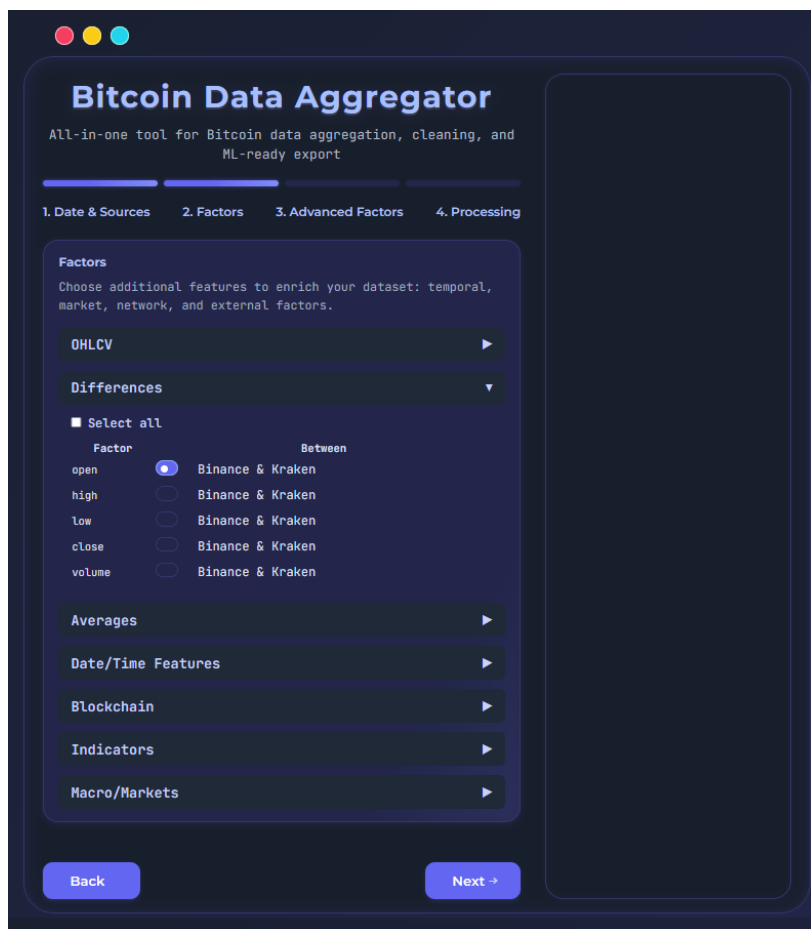


Рис. 7. Сторінка збору даних. Крок 2

Третій крок (рис. 8) призначений для додавання складніших факторів, доступних після активації чекбокса "enable interpolation". Без нього всі опції цього кроку неактивні. Розширені фактори згруповані в акордеони: volatility (наприклад binance_volatility_3d), volume-to-price ratio (співвідношення обсягу до ціни), cross-exchange ratios (крос-біржові співвідношення), macro market correlations (кореляція bitcoin із s&p500), blockchain metrics (похідні блокчейн-метрики), technical metrics (складніші індикатори), aggregated indexes (індекси, наприклад, страху та жадібності), additional factors (наприклад, avg_price_all_minus_binance_close). Доступність факторів залежить від вибору бірж і увімкнення інтерполяції,

з попередженнями, якщо умови не виконані (наприклад, недостатньо бірж для крос-біржових розрахунків).

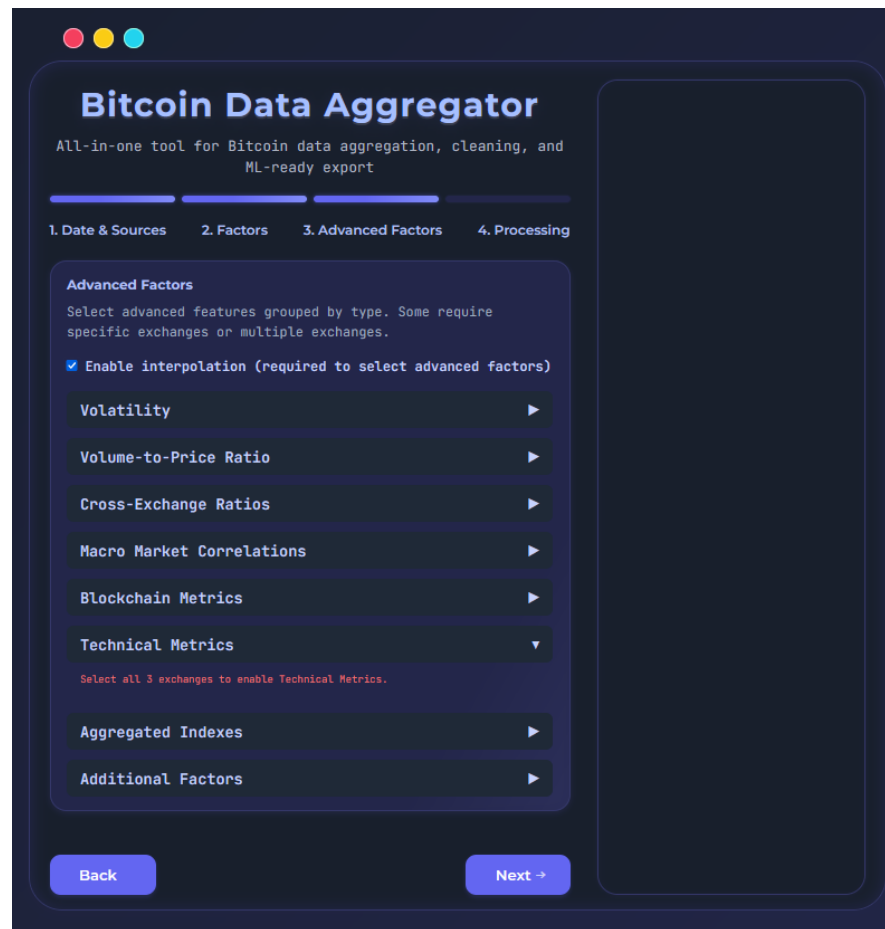


Рис. 8. Сторінка збору даних. Крок 3

На четвертому кроці (рис. 9) користувач налаштовує параметри обробки даних. Доступні опції: Anomaly Filter (Strict/Off, методи: Z-score, IQR, Both), Normalization (None/MinMax/Standard/Robust), Yeo-Johnson transform (перетворення для ненормальних розподілів), Target Task (Regression/Classification/None, залежить від інтерполяції), Target Horizon (горизонт прогнозування, якщо Target Task активний), Target Column (вибір цільової змінної із доступних колонок), Balance Classes (None/Oversample/Undersample, для Classification), K Neighbors (для Oversample, наприклад, SMOTE), Winsorize (вінсоризація) із Winsor Limits (квантилі, наприклад, (0.01, 0.99)), Encoding (OneHot/Label/None, для us_holiday, day_of_week), Show Boxplot Stats (статистика у звіті), Download (Zip-archive), Export Format (CSV/JSON). Опції динамічно активуються залежно від попередніх виборів.

Bitcoin Data Aggregator
All-in-one tool for Bitcoin data aggregation, cleaning, and ML-ready export

1. Date & Sources 2. Factors 3. Advanced Factors 4. Processing

Processing
Configure data processing options for your dataset.

Anomaly Filter: Strict Method: Z-score

Normalization: MinMax ☒ Yeo-Johnson transform

Target Task: Regression Target Horizon: 2

Target Column: diff_binance_kraken_ope

Balance Classes: None K Neighbors: 1

☐ Winsorize Winser Limits: (0.01, 0.99) ☒

Encoding: None ☒ Show Boxplot Stats

Download: Zip-archive Export Format: CSV

Back Submit

Рис. 9. Сторінка збору даних. Крок 4

Після налаштування всіх параметрів користувач натискає "Collect Data". Запит валідується: якщо є помилки (наприклад, не обрано дати чи біржі), відображається повідомлення. Під час виконання показується спінер і прогрес-бар. Після завершення формується ZIP-архів (рис. 10) що включає оброблений датасет (рис. 11) із вибраними факторами та трансформаціями, готовий для аналізу, візуалізації чи моделювання (формат залежить від вибору "Export Format" на Кроці 4), та звіт про якість і обробку даних (quality_report.txt) (рис. 12), який детально документує процес, надає статистику та відображає застосовані зміни, сприяючи розумінню формування кінцевого датасету.

Додатково є кнопки "Previous" і "Next" для навігації між кроками та "User Guide Modal" із інструкціями щодо розширених опцій, що робить сторінку потужним інструментом для створення датасетів для аналізу та прогнозування.

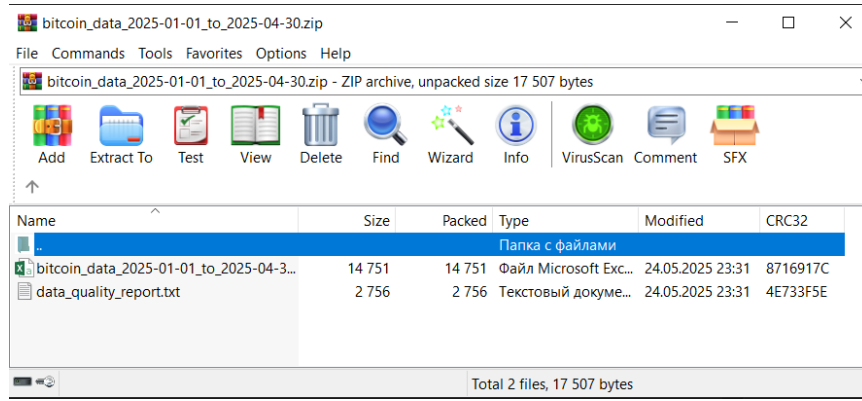


Рис. 10. Вміст сформованого ZIP-архіву

100	2025-04-16,0.2736687549887667,0.2776880435417054,-0.3953053763356827,0.31792542939192997,0.9879182882765717,0.5429156362885657	✓
104	2025-04-17,0.28751958917208154,0.10893906418106847,0.10400716081280297,0.20907607380968335,0.914941529641551,0.08419202384113622	
105	2025-04-18,0.3201879882816441,0.5429156362885657,0.5653629242452702,0.5751368314578992,0.921888832922679,0.5837973464965042	
109	2025-04-19,0.3033781592093233,0.08419202384113622,0.9984098851100798,0.6177637032654886,0.9287622452885402,0.48055401355685823	
107	2025-04-20,0.3247589422665335,0.5837973464965042,1.4092477754158603,0.4296600881176812,0.9355637246124997,0.34717240958142415	
108	2025-04-21,0.3283751297548255,0.48055401355685823,-1.6076956695330633,0.8619953731546284,0.9422951481913353,0.47871853122663855	
109	2025-04-22,0.41001259914248407,0.34717240958142415,-0.9515290995659739,0.6276539925227891,0.9489583155567203,0.32888968692898035	
110	2025-04-23,0.6087858462889413,0.47871853122663055,-0.3953053763356827,0.8549488602039303,0.9555549531723619,0.3924263790410937	
111	2025-04-24,0.6168623940918886,0.32888968692898035,-0.10400716081280297,0.027439537568276273,0.9620867184579656,0.749448284185714	
112	2025-04-25,0.6262605009213316,0.3924263790410937,0.5653629242452702,0.08949765705518015,0.9685552035351022,0.5271042348681069	
113	2025-04-26,0.6475413724157404,0.749448284185714,0.9984098851100798,0.16774947020705883,0.9749619387179883,0.7037830991101197	
114	2025-04-27,0.647197110021113,0.5271042348681069,1.4092477754158603,0.16256456431056449,0.9813083957699813,0.31567873973978955	
115	2025-04-28,0.6187551002172221,0.7037830991101197,-1.6076956695330633,0.9677749927444744,0.987595990946619,0.31293056648661655	
116	2025-04-29,0.6595273880349399,0.31567873973978955,-0.9515290995659739,0.8534509126681085,0.9938260878282847,	
117	2025-04-30,0.6352111592718446,0.31293056648661655,-0.3953053763356827,0.39686205679319875,1.0,	

Рис. 11. Вміст сформованого ZIP-архіву. Дані у форматі CSV

data_quality_report.txt — Блокнот

ФайлПравкаФорматВидСправка

===== DATA PROCESSING REPORT =====

Start Date: 2025-01-01
End Date: 2025-04-30
Initial Rows: 120
Initial Columns: date, binance_open, diff_binance_kraken_open, day_of_week, kraken_volatility_3d_to_7d_ratio, days_since_last_us_holiday
Interpolate: True
Anomaly filter: strict
Anomaly method: zscore
Rows After Anomaly Filtering: 116
Normalization: minmax
Encoding: none
Target Task: regression
Target Horizon: 2
Show boxplot stats: True
Winsorization: False
Yeo-Johnson transform: True
Columns Yeo-Johnson transformed: binance_open, diff_binance_kraken_open, day_of_week, kraken_volatility_3d_to_7d_ratio, days_since_last_us_holiday
Class balancing: None

Anomaly Counts:
 binance_open: 0
 diff_binance_kraken_open: 4
 kraken_volatility_3d_to_7d_ratio: 0
 days_since_last_us_holiday: 0

Detailed anomaly filtering report:
 diff_binance_kraken_open:
 2025, January: [4]
 from 2025-01-01 to 2025-01-19

|

Target column used: diff_binance_kraken_open

Regression task performed successfully.

Advanced Feature Statistics:

Column	Mean	Std	Median	Skew	Kurtosis	% Missing
binance_open	0.5348	0.2536	0.6128	-0.0043	-1.2436	0.0000%
diff_binance_kraken_open	0.4587	0.2117	0.4790	-0.0502	-0.3663	0.0000%
kraken_volatility_3d_to_7d_ratio	0.4943	0.2738	0.4765	0.0249	-0.9530	0.0000%
days_since_last_us_holiday	0.5707	0.2575	0.5600	-0.0985	-0.8800	0.0000%
target	0.4582	0.2119	0.4790	-0.0498	-0.3464	1.7241%

Categorical features (us_holiday and day_of_week) is not included here.

Final Rows: 116
Final Columns: ['date', 'binance_open', 'diff_binance_kraken_open', 'day_of_week', 'kraken_volatility_3d_to_7d_ratio', 'days_since_last_us_holiday', 'target']

First 5 rows of processed DataFrame:

date	binance_open	diff_binance_kraken_open	days_since_last_us_holiday	target
2025-01-04	0.759676	0.682612	0.179752	0.654762
2025-01-05	0.761122	0.294791	0.217670	0.322697
2025-01-06	0.765584	0.654762	0.250912	0.421093
2025-01-07	0.884202	0.322697	0.280710	0.000000
2025-01-08	0.721401	0.421093	0.307845	0.478558

[Showing only the first 3 and last 2 columns out of 7]

6. **Pop-up повідомлення для новачків системи**

Посібник користувача (рис. 13), доступний на сторінці збору даних вебзастосунку MLBit, є основним довідковим інструментом, що допомагає користувачам налаштувати процес збору та обробки даних для аналізу криптовалют. Він виконаний у вигляді модального вікна з трьома колонками, надаючи структуровану інформацію про етапи роботи.

Перша колонка ("Key Steps Overview" і "Tips & Best Practices") коротко описує чотири кроки: "Дати та Джерела" (вибір діапазону дат і бірж), "Базові Фактори" (OHLCV, блокчейн-метрики), "Розширені Фактори" (волатильність, кореляції), "Обробка" (фільтри, нормалізація), а також містить поради, наприклад, використовувати "Live JSON Preview" для перевірки параметрів, починати з простих конфігурацій, аналізувати звіт quality_report.txt і враховувати вплив інтерполяції.

Друга колонка ("Advanced Factors (Step 3)") деталізує розширені фактори, доступні після увімкнення інтерполяції: волатильність, VPR (Volume-to-Price Ratio), міжбіржові співвідношення, макроекономічні кореляції, розширені блокчейн-метрики, технічні метрики (комбінації індикаторів) та агреговані індекси (наприклад, індекс страху).

Третя колонка ("Data Processing (Step 4)") пояснює опції обробки: методи виявлення аномалій, нормалізацію, налаштування цільової змінної, вінзоризацію, кодування та параметри експорту.



Рис. 13. Модальне вікно Quick Guide

Інтерактивний розділ "Сценарії", доступний через кнопку "Дізнатися більше: Стартові точки для стратегій" у посібнику і пропонує три приклади аналітичних підходів для роботи з даними Bitcoin. Кожен сценарій надає мету, рекомендовані фактори та налаштування, слугуючи відправною точкою для досліджень.