

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

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

«На правах рукопису»
УДК 004.006.3

До захисту допущено:

Завідувач кафедри

_____ Сергій СТИПЕНКО

«___» _____ 2024 р.

Магістерська дисертація

на здобуття ступеня магістра

**за освітньо-професійною програмою «Інженерія програмного
забезпечення комп'ютерних систем» зі спеціальності 121 “Інженерія
програмного забезпечення” на тему: «Інтелектуальна система
прогнозування криптовалютних цін на основі статистичних даних»**

Виконав:

студент VI курсу, групи ІМ-21мн

Костін Денис Олександрович _____

Керівник:

доц., к.т.н., доц.

Павлов Валерій Георгієвич _____

Консультант з нормоконтролю:

проф. каф. ОТ, д.т.н., професор

Жабін Валерій Іванович _____

Рецензент:

Доц., к.т.н, доц.,

Шимкович Володимир

Миколайович _____

Засвідчую, що у цій магістерській
дисертації немає запозичень з праць
інших авторів без відповідних
посилань.

Студент _____

(підпис)

Київ – 2024 року

4. Предмет дослідження розробка системи прогнозування цін криптовалют, який включає аналіз методів машинного навчання, обробку та аналіз статистичних даних, а також створення моделей прогнозування для передбачення цінових рухів криптовалют.

5. Перелік завдань, які потрібно розробити:

1. Дослідження методів та технологій прогнозування загалом та в криптовалютному сегменті.
2. Огляд існуючих методів прогнозування криптовалютних цінових рухів.
3. Розробка програмного забезпечення, що включає в себе:
 - 3.1. Збір необхідних статистичних даних про криптовалютний ринок.
 - 3.2. Підготовка та обробка даних для подальшого використання в моделі прогнозування.
 - 3.3. Вибір та налаштування алгоритмів машинного навчання для розробки прогностичної моделі.
 - 3.4. Розробка та навчання прогностичної моделі на основі зібраних даних.
4. Оцінка та валідація результатів моделі за допомогою метрик, здійснення аналізу отриманих результатів та формування висновків стосовно ефективності розробленої системи.

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Проф. Жабін В.І.		

7. Дата видачі завдання_____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1.	Затвердження теми дослідження. Визначення предмету дослідження.		
2.	Проведення огляду існуючих рішень		
3.	Збір необхідних статистичних даних.		
4.	Підготовка та обробка даних для подальшого використання.		

5.	Вибір та налаштування алгоритмів машинного навчання для розробки.		
6.	Розробка та навчання моделей.		
7.	Оцінка та валідація результатів.		
8.	Оформлення пояснювальної записки.		
9.	Захист.		

Студент

Денис КОСТІН

(підпис)

Науковий керівник дисертації

Валерій ПАВЛОВ

(підпис)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Інтелектуальна система прогнозування
криптовалютних цін на основі статистичних даних

студентом: Костіним Денисом Олександровичем

Робота складається із вступу та чотирьох розділів. Загальний обсяг роботи: 86 аркушів основного тексту, 48 ілюстрація, 4 таблиць. При підготовці використовувалася література з 36 різного джерела.

Актуальність. Криптовалюти стають все більш інтегрованими в наше повсякденне життя, привертаючи увагу як індивідуальних користувачів, так і фінансових установ. Зростаюча популярність криптовалют серед трейдерів та звичайних осіб, які обирають цю форму активів, підкреслює важливість розвитку ефективних інструментів аналізу та прогнозування криптовалютних ринків. З врахуванням нестабільності цін на криптовалюти, інтелектуальні системи прогнозування, що базуються на статистичних та аналітичних даних, у поєднанні з технологіями машинного навчання, можуть надати цінні інструменти для прийняття рішень і зменшення фінансового ризику для учасників ринку криптовалют. Таке дослідження стає дуже актуальним в контексті швидкого розвитку криптовалютних ринків та зростання зацікавленості громадськості у цьому сегменті фінансових інструментів.

Проблема. Зараз не існує надійної системи яка би забезпечувала стабільний прогноз хоча б з якістю 51%. Це б сильно вплинуло на трейдерів, які би користувалися такою системою. Нестабільність на криптовалютному ринку ускладнює прийняття рішень для інвесторів, трейдерів та інших учасників ринку, оскільки вони намагаються передбачити цінові рухи для здійснення успішних операцій. Наша система може спробувати допомогти зменшити цю проблему, надаючи користувачам інструмент для прогнозів основуючись на статичних та історичних даних. Таким чином, вирішення цієї проблеми сприятиме підвищенню ефективності управління ризиками та

забезпечить більшу стабільність та впевненість для учасників ринку криптовалют.

Мета і завдання дослідження. Метою магістерської роботи є покращення ефективності управління ризиками та забезпечення стабільності для учасників криптовалютного ринку шляхом розробки та впровадження інтелектуальної системи прогнозування цінових змін.

Для досягнення мети дослідження поставлено і вирішено такі завдання:

- Дослідження методів та технологій прогнозування загалом та в криптовалютному сегменті.
- Огляд існуючих методів прогнозування криптовалютних цінових рухів.
- Розробка програмного забезпечення, що включає в себе:
 - Збір необхідних статистичних даних про криптовалютний ринок.
 - Підготовка та обробка даних для подальшого використання в моделі прогнозування.
 - Вибір та налаштування алгоритмів машинного навчання для розробки прогностичної моделі.
 - Розробка та навчання прогностичної моделі на основі зібраних даних.
- Оцінка та валідація результатів моделі за допомогою метрик, здійснення аналізу отриманих результатів та формування висновків стосовно ефективності розробленої системи.

Об'єкт дослідження. Спосіб прогнозування цінових рухів використовуючи статистичні дані та технології машинного навчання.

Предмет дослідження. Розробка системи прогнозування цін криптовалют, який включає аналіз методів машинного навчання, обробку та аналіз статистичних даних, а також створення моделей прогнозування для передбачення цінових рухів криптовалют.

Наукова новизна. У дисертації запропоновано спосіб прогнозування цінових рухів криптовалют, що включає в себе інтеграцію сучасних методів машинного навчання та обробки великих даних, включаючи обробку поточкових даних у реальному часі. Цей метод враховує можливість постійного донавчання, що дозволяє моделі адаптуватися до нових даних та змін на ринку криптовалют. Проведене дослідження та запропонований підхід може бути використаний для підвищення точності прогнозів та використання одним з компонентів для приймання інвестиційних рішень.

Особистий внесок здобувача. Особистий внесок здобувача у магістерське дослідження полягає у виконанні всіх етапів роботи та розробці концепції інтегрованої системи прогнозування цінових рухів на криптовалютному ринку. Здобувач особисто провів аналіз та обробку великого обсягу даних з криптовалютних ринків, здійснив тестування та валідацію розробленої системи, а також вдосконалив алгоритми та моделі з метою підвищення точності прогнозів. Навички самостійної роботи, творчий підхід та систематичний підхід до вирішення завдань дослідження дозволили здобувачу отримати як теоретичні, так і практичні результати, які відображено у магістерській роботі.

Практичне значення та цінність одержаних результатів. Отримані результати можуть використовуватися у майбутніх дослідженнях за напрямками:

- Розширення області застосування: Дослідження може бути розширене на інші фінансові ринки або ринки активів.
- Оптимізація алгоритмів і моделей: Дослідження може включати подальший розвиток і вдосконалення алгоритмів машинного навчання та моделей прогнозування для підвищення точності та ефективності.
- Дослідження впливу факторів на ринок: Результати дослідження можуть бути використані для подальшого аналізу впливу різних

факторів на цінові рухи на ринку криптовалют та розробки стратегій управління ризиками.

- Аналіз нових типів даних: У майбутніх дослідженнях можна досліджувати використання нових типів даних, таких як соціальні мережі, новинні статті або макроекономічні показники, для поліпшення якості прогнозів цінових змін.

Апробації результатів дисертації. Результати дослідження дисертації були представлені на наступній конференції: «The International Conference on Security, Fault Tolerance, Intelligence» (ICSFTI2024), 7 червня 2024, м. Київ.

Ключові слова. КРИПТОВАЛЮТА, ПРОГНОЗУВАННЯ ЦІН, МАШИННЕ НАВЧАННЯ, АНАЛІЗ ДАНИХ, BIG DATA, ФІНАНСОВИЙ РИНОК, ТОЧНІСТЬ ПРОГНОЗІВ, ТЕХНІЧНИЙ АНАЛІЗ, РОЗРАХУНОК ІНДИКАТОРІВ.

ABSTRACT

The thesis consists of an introduction and four chapters. The total volume of the work is 86 pages of main text, 48 illustrations, and 4 tables. The preparation involved the use of literature from 36 different sources.

Relevance. Cryptocurrencies are becoming increasingly integrated into our daily lives, attracting the attention of both individual users and financial institutions. The growing popularity of cryptocurrencies among traders and ordinary individuals who choose this form of assets underscores the importance of developing effective tools for analyzing and forecasting cryptocurrency markets. Given the volatility of cryptocurrency prices, intelligent forecasting systems based on statistical and analytical data, combined with machine learning technologies, can provide valuable tools for decision-making and reducing financial risk for cryptocurrency market participants. This research is particularly relevant in the context of the rapid development of cryptocurrency markets and the growing public interest in this segment of financial instruments.

Problem. Currently, there is no reliable system that provides stable forecasts with at least 51% accuracy. This would significantly impact traders who could use such a system. Market volatility complicates decision-making for investors, traders, and other market participants as they try to predict price movements to execute successful transactions. Our system aims to help mitigate this problem by providing users with a forecasting tool based on static and historical data. Solving this problem will enhance risk management efficiency and provide greater stability and confidence for cryptocurrency market participants.

Aim and Objectives of the Research. The aim of the master's thesis is to improve risk management efficiency and ensure stability for cryptocurrency market participants by developing and implementing an intelligent system for forecasting price changes.

To achieve the aim of the research, the following objectives were set and accomplished:

- Researching forecasting methods and technologies in general and within the cryptocurrency segment.
- Reviewing existing methods for forecasting cryptocurrency price movements.
- Developing software that includes:
 - Collecting necessary statistical data on the cryptocurrency market.
 - Preparing and processing data for subsequent use in the forecasting model.
 - Selecting and tuning machine learning algorithms to develop the forecasting model.
 - Developing and training the forecasting model based on the collected data.
- Evaluating and validating the model's results using metrics, analyzing the obtained results, and forming conclusions regarding the system's effectiveness.

Object of the Research. The method of forecasting price movements using statistical data and machine learning technologies.

Subject of the Research. The development of a cryptocurrency price forecasting system, which includes the analysis of machine learning methods, the processing and analysis of statistical data, and the creation of forecasting models to predict cryptocurrency price movements.

Scientific Novelty. The dissertation proposes a method for forecasting cryptocurrency price movements that integrates modern machine learning methods and big data processing, including real-time data streams. This method considers the possibility of continuous retraining, allowing the model to adapt to new data and changes in the cryptocurrency market. The conducted research and the proposed approach can be used to enhance the accuracy of forecasts and serve as a component for making investment decisions.

Personal Contribution of the Researcher. The personal contribution of the researcher to the master's thesis includes the completion of all stages of the work and the development of the concept of an integrated system for forecasting price movements in the cryptocurrency market. The researcher personally conducted the analysis and processing of a large volume of cryptocurrency market data, tested and validated the developed system, and improved algorithms and models to enhance forecasting accuracy. The skills of independent work, creative approach, and systematic approach to solving research tasks allowed the researcher to obtain both theoretical and practical results reflected in the master's thesis.

Practical Significance and Value of the Obtained Results. The obtained results can be used in future research in the following areas:

- Expanding the scope of application: The research can be extended to other financial markets or asset markets.
- Optimizing algorithms and models: The research can include further development and improvement of machine learning algorithms and forecasting models to enhance accuracy and efficiency.
- Analyzing the impact of factors on the market: The results can be used for further analysis of the impact of various factors on price movements in the cryptocurrency market and the development of risk management strategies.
- Analyzing new types of data: Future research can explore the use of new types of data, such as social media, news articles, or macroeconomic indicators, to improve the quality of price change forecasts.

Dissemination of the Research Results. The results of the dissertation research were presented at the following conference: "The International Conference on Security, Fault Tolerance, Intelligence" (ICSFTI2024), June 7, 2024, Kyiv.

Keywords. CRYPTOCURRENCY, PRICE FORECASTING, MACHINE LEARNING, DATA ANALYSIS, BIG DATA, FINANCIAL MARKET,

FORECAST ACCURACY, TECHNICAL ANALYSIS, INDICATOR
CALCULATION.

ЗМІСТ

ВСТУП	1
РОЗДІЛ 1	4
АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО ПРОГНОЗУВАННЯ ЦІНОВИХ РУХІВ	4
1.1. Що визначає ціну криптовалют? Що таке прогнозування цін?	4
1.2. Аналіз існуючих методів прогнозування цін криптовалют	7
1.3. Аналіз існуючих готових додатків для передбачення фінансових ринків.....	16
1.4. Дослідження сучасних концепцій обробки великих обсягів даних	21
1.4.1. Збір та зберігання даних	21
1.4.2. Попередня обробка та очищення даних	24
1.4.3. Основні алгоритми для машинного навчання на великих даних ...	28
1.5. Висновки до розділу 1	31
РОЗДІЛ 2	32
СПОСІБ ІНТЕЛЕКТУАЛЬНОГО ПРОГНОЗУВАННЯ ЦІНОВИХ РУХІВ ..	32
2.1. Спосіб збору даних для подальшого використання, відмовостійкість при зборі та збереженні	34
2.2. Аналіз пропусків та артефактів у даних	35
2.3. Аналітичний спосіб розрахунку різних фінансових індикаторів.....	36
2.4. Нормалізація даних та підготовка їх до використання у тренуванні моделі, тренування та тестування моделі	42
2.5. Висновки до розділу 2	44
РОЗДІЛ 3	45
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	45

3.1	Обумовленість вибраних технологій	45
3.2	Опис програмних складових та компонентів	46
3.2.1	<i>Опис бази даних</i>	47
3.2.2	<i>Проект crypto-prices-loader</i>	52
3.2.3	<i>Проект crypto-prices-predictor</i>	60
3.3	Висновки до розділу 3.....	69
РОЗДІЛ 4.....		71
ТЕСТУВАННЯ ПРОГРАМИ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ ПРОЕКТУ.....		71
4.1	Огляд розробленого програмного продукта.....	71
	Висновки до розділу 4	80
ВИСНОВКИ.....		81
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ		83

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ML (Machine Learning) – Машинне навчання

AI (Artificial Intelligence) – Штучний інтелект

BTC (Bitcoin) – Біткоїн

ETH (Ethereum) – Ефір

ROI (Return on Investment) – Показник прибутковості інвестицій

LSTM (Long Short-Term Memory) – Довготермінова короткочасна пам'ять

CNN (Convolutional Neural Network) – Згортова нейронна мережа

API (Application Programming Interface) – Інтерфейс програмування застосунків

ICO (Initial Coin Offering) – Первинна пропозиція монет

NLP (Natural Language Processing) – Обробка природної мови

SVM (Support Vector Machine) – Метод опорних векторів

ANN (Artificial Neural Network) – Штучна нейронна мережа

GPU (Graphics Processing Unit) – Графічний процесор

ВСТУП

У світі, насиченому технологічними новаціями, роль криптовалют стає надзвичайно важливою. За останні декілька років, а особливо місяців, відбулось стрімке зростання криптовалютного ринку та збільшення обсягів торгівлі криптовалютами активами. Вони здобувають все більше популярності серед індивідуальних користувачів та фінансових інституцій, які вбачають у цьому новому класі активів потенційні можливості. Волатильність криптовалютного ринку ускладнює прийняття обґрунтованих фінансових рішень для трейдерів, інвесторів та інших учасників ринку. Постійні зміни в цінових тенденціях та вплив різноманітних факторів вимагають застосування сучасних технологій для ефективного аналізу та прогнозування.

На сьогоднішній день не існує надійної системи прогнозування, яка забезпечила б стабільний прогноз з задовільною точністю. Традиційні методи прогнозування не завжди задовольняють потреби сучасного криптовалютного ринку. Використання машинного навчання (ML) та технологій обробки великих даних (Big Data) дозволяє створити більш точні та адаптивні моделі прогнозування, здатні враховувати безперервні зміни на ринку. Інтелектуальні системи прогнозування, які базуються на статистичних та аналітичних даних, можуть значно покращити процес прийняття рішень та знизити фінансовий ризик.

Сучасний обсяг даних відображає живу динаміку ринку, де кожна зміна може мати значення. Враховуючи це, у даній магістерській роботі проводиться дослідження та розробка інтелектуальної системи прогнозування цін криптовалют, яка базується на використанні методів машинного навчання та обробки великих даних. Для досягнення поставленої мети в роботі буде проведено аналіз існуючих методів прогнозування цінових змін на криптовалютних ринках, досліджено сучасні підходи до аналізу великих

обсягів даних, розроблено та випробувано новий метод прогнозування з використанням машинного навчання.

Основною метою магістерської роботи є покращення ефективності управління ризиками та забезпечення стабільності для учасників криптовалютного ринку шляхом розробки та впровадження інтелектуальної системи прогнозування цінових змін. Для досягнення цієї мети поставлено і вирішено наступні завдання:

- аналіз способів прогнозування використовуючи машинне;
- дослідження та опис технологій;
- розробка способу інтелектуального прогнозування цінових рухів;
- реалізація інтелектуальної системи прогнозування;
- тестування та валідація системи.

Об'єктом дослідження є процес прогнозування змін цін на криптовалютному ринку. Предметом дослідження є процес збору та обробки даних для прогнозування змін цін на криптовалютному ринку.

Наукова новизна одержаних результатів роботи полягає у наступному: запропоновано спосіб прогнозування цінових рухів криптовалют, що включає в себе інтеграцію сучасних методів машинного навчання та обробки великих даних, включаючи обробку потокових даних у реальному часі. Цей метод враховує можливість постійного донавчання, що дозволяє моделі адаптуватися до нових даних та змін на ринку криптовалют. Проведене дослідження та запропонований підхід може бути використаний для підвищення точності прогнозів та використання одним з компонентів для приймання інвестиційних рішень.

Отже, враховуючи тематику роботи та сучасні тенденції, дослідження з передбачення цінових рухів має значення та практичне застосування для трейдерів, інвесторів та фінансових установ, що працюють з криптовалютами. Сучасний ринок вимагає точних та надійних інструментів для прогнозування,

які можуть допомогти учасникам ринку приймати обґрунтовані рішення та зменшувати фінансовий ризик.

РОЗДІЛ 1

АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО ПРОГНОЗУВАННЯ ЦІНОВИХ РУХІВ

I.1. Що визначає ціну криптовалют? Що таке прогнозування цін?

Криптовалютний ринок – це складне середовище, яке постійно розвивається і піддається впливу безлічі факторів. Основні з яких показано в таблиці 1. «Основні фактори, що впливають на ціну криптовалют». Самим важливим та фундаментальним з яких є поняття попиту та пропозиції, далі вже йде рівень використання та впровадження мережі, державне регулювання, засоби масової інформації, технологічні досягнення, настрої ринку, основні події новин, а також дії китів або ринкових маніпуляторів.

Таблиця 1

Основні фактори, що впливають на ціну криптовалют

Фактор	Опис фактора
Попит і пропозиція	Відношення між кількістю доступної криптовалюти та кількістю бажаючих її купити.
Державне регулювання	Закони та нормативні акти, які впливають на ринок криптовалют.
Технологічні досягнення	Інновації в блокчейн-технологіях та криптовалютних проектах.
Засоби масової інформації	Новини та медіа-висвітлення криптовалют.
Настрої ринку	Ставлення та очікування інвесторів та трейдерів.
Дії великих гравців	Активність великих інвесторів та трейдерів, які можуть маніпулювати ринком.
Макроекономічна ситуація в світі, основні події та новини	Важливі події, такі як війни, показники світової економіки, взаємозв'язки між різними країнами, хардфорки, партнерства, регуляторні рішення.

Оцінити перспективи середовища можна за допомогою методів прогнозування. Розрізняють два критерії для класифікації цих методів: рівень об'єктивності прогнозування (суб'єктивні та об'єктивні методи) та рівень аналітичності цього процесу (наївні та причинно-наслідкові методи).

Суб'єктивні методи ґрунтуються на позиції певного експерта щодо процесів, які не виражені в явній формі, але лежать в основі розроблення прогнозу. Застосування об'єктивних методів передбачає, що процеси чітко сформульовані та можуть бути відтворені іншими особами, тобто існує велика ймовірність формулювання ними такого ж прогнозу. Вважають, що при цьому спостерігається протиставлення кількісних методів якісним, в яких переважає інтуїція.

Причинно-наслідкові (казуальні) методи застосовують тоді, коли чинники ідентифіковані та їхні вірогідні майбутні значення спрогнозовані, що дає змогу сформувати ймовірне значення показника за умови реалізації прийнятного сценарію.

Поєднання цих двох підходів до класифікації сприяє утворенню чотирьох типів методів прогнозування (рис. 1).



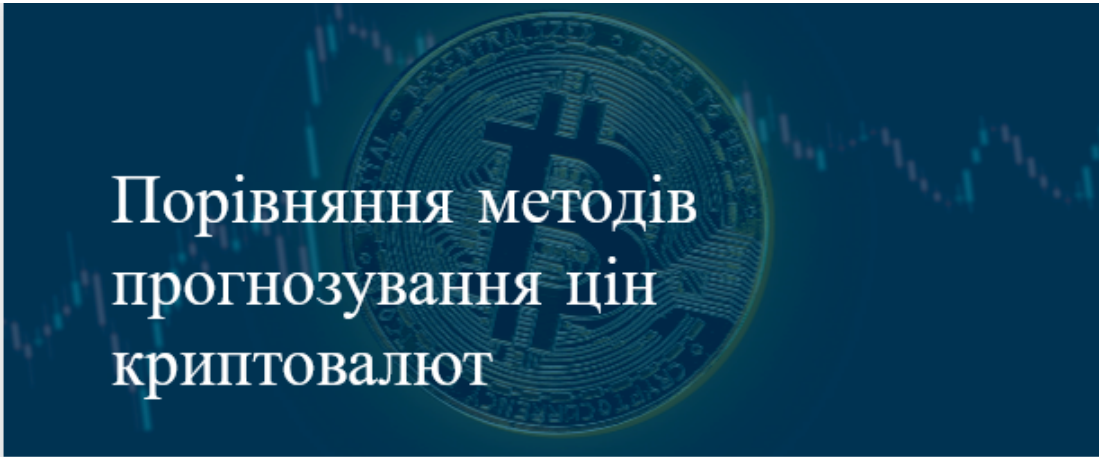
Рис.1. Чотири типи методів прогнозування

Загалом, прогнозуванням цін є передбачення напрямку руху ціни в майбутньому, та абсолютного значення вартості активу. Він включає аналіз багатьох факторів: історичні дані про ціни, аналіз технічних індикаторів, ринкові тенденції, новинні події і технологічні досягнення, щоб визначити потенційну траєкторію ціни конкретної криптовалюти. [1]

Вне залежності від того, який криптоактив досліджується, будь то Bitcoin, Ethereum чи, наприклад, Solana, точне прогнозування є складним завданням через притаманну криптовалютному ринку сильну волатильність. Однак для прогнозування чи передбачення використовується безліч різних методів та підходів. [1]

I.2. Аналіз існуючих методів прогнозування цін криптовалют

Існують декілька методів для прогнозування цін на криптовалюту, основні представлені на (рис. 2).



Костін Денис			
★ Метод	★ Опис	★ Переваги	★ Недоліки
Технічний аналіз	Аналіз історичних цінових даних та індикаторів.	Виявлення короткострокових трендів.	Вразливість до ринкового шуму.
Фундаментальний аналіз	Оцінка внутрішньої цінності через аналіз технології, використання та перспектив.	Довгострокове прогнозування.	Важко оцінити суб'єктивні фактори.
Ончейн-аналітика	Аналіз даних з блокчейну, таких як транзакції та активність адрес.	Прямий доступ до даних мережі.	Вимагає технічних знань для інтерпретації даних.
Аналіз настроїв	Аналіз обговорень в соціальних мережах та новин для оцінки громадської думки.	Виявлення настроїв та їх вплив на ринок.	Вразливість до маніпуляцій та неточностей в обговореннях.
Графічне зображення розробив студент КПІ, ФІОТ, Костін Денис			

Рис.2. Порівняння методів прогнозування цін криптовалют

1. Технічний аналіз:

Цей підхід полягає в аналізі історичних цінових моделей, тенденцій та технічних індикаторів для визначення потенційних рівнів підтримки та опору, прогнозування майбутніх рухів цін та прийняття торгових рішень.

Технічні індикатори є одним з найпоширеніших способів прогнозування руху цін на криптовалюту. Серед них ключове місце посідають ковзні середні. Аналіз базується на порівнянні цін на криптовалюту з ключовими ковзними середніми, такими як 21-, 50- та 200-денні середні. Осцилятори, такі як індекс відносної сили (RSI) та конвергенція/дивергенція ковзних середніх (MACD), використовуються для аналізу короткострокових ринкових тенденцій та точного визначення можливих варіантів торгівлі.

Для прикладу можна привести аналіз графіку історичної вартості біткоїна з сайту tradingview [2], (рис.3.).



Рис.3. Приклад аналізу графіку історичної вартості біткоїна

Для можна завантажити ціни з цікавої біржи та за допомогою будь-якої мови програмування відрендерити графік для аналізу:

Використовуючи python та matplotlib створемо графік з 3-а індикаторами Moving Average (MA21, MA50, MA200) (рис.4.)

```
import matplotlib.pyplot as plt

import pandas as pd

df = pd.read_csv('prices.csv')

df['Date'] = pd.to_datetime(df['Date'])

df.set_index('Date', inplace=True)

df['MA21'] = df['Close'].rolling(window=21).mean()

df['MA50'] = df['Close'].rolling(window=50).mean()

df['MA200'] = df['Close'].rolling(window=200).mean()

plt.figure(figsize=(14, 7))

plt.plot(df['Close'], label='Price', color='blue')

plt.plot(df['MA21'], label='21-day MA', color='orange')

plt.plot(df['MA50'], label='50-day MA', color='green')

plt.plot(df['MA200'], label='200-day MA', color='red')

plt.title('Historical Prices with Moving Averages')

plt.xlabel('Date')

plt.ylabel('Price (USD)')

plt.legend()

plt.grid(True)

plt.show()
```

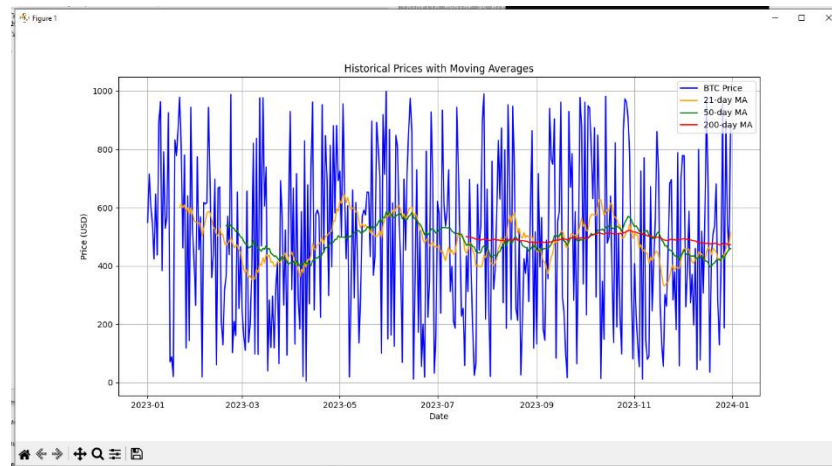


Рис.4. Створений графік для технічного аналізу

2. Фундаментальний аналіз:

Цей метод концентрується на оцінці внутрішньої цінності криптовалюти через аналіз технологічної бази, рівня використання мережі, досвіду команди та майбутніх перспектив.

Для прикладу можна привести аналіз:

- I. Можна провести аналіз та якість білої книги біткоіна, з основними принципами роботи цього криптоактиву (Bitcoin: A Peer-to-Peer Electronic Cash System (Satoshi Nakamoto - satoshin@gmx.com - www.bitcoin.org) [3] (рис.5.).

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model.

Рис.5. Біла книга біткоіна з описом основних принципів роботи криптовалюти Bitcoin

II. Можна проаналізувати код самого криптоактиву. Для прикладу код біткоїна, розташований на github (рис.6.).

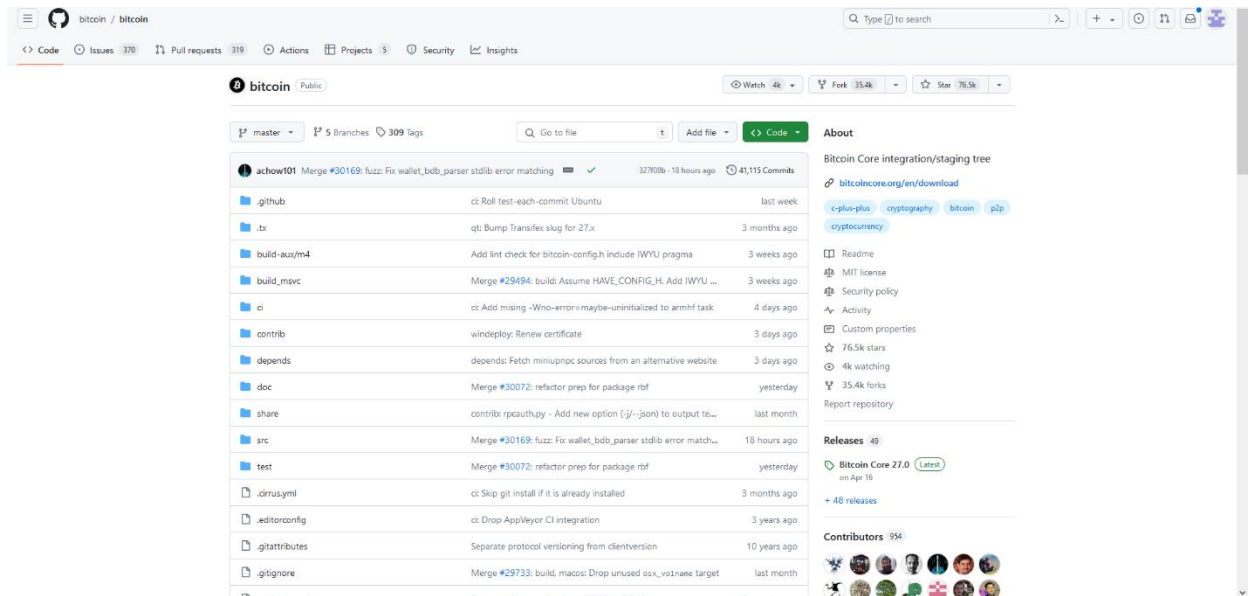


Рис.6. Офіційний github біткоїна.

III. Можна проаналізувати та дослідити учасників команди творців криптоактиву, основні показники, та що стоїть за ідеєю тієї чи іншої криптовалюти. Для цього чудово підійде сайт coindmarketcap (рис.7.).

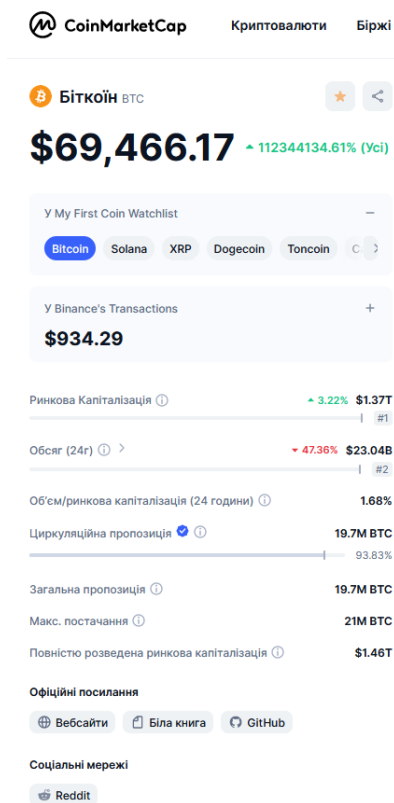


Рис.7. Основні фундаментальні показники біткоїна з сайту coinmarketcap.

3. Ончейн-аналітика:

Цей метод включає аналіз даних безпосередньо з блокчейну, таких як обсяг транзакцій, активні адреси та активність майнерів, для оцінки загального стану та використання криптовалютних мереж. Аналіз дій криптовалютних китів (гравців з вагомих статком, дії яких можуть істотно вплинути на реалії та стан цін активу).

В цей перелік можна віднести наступні індикатори:

- 1) HODL Waves, зображений на (рис.8.). Принцип роботи цього індикатору наступний: всі адреси криптоактиву розподіляться на декілька груп по їх активності, та «вагомості». Потім йде представлення змін по кожній з груп в залежності від планів на зріст чи падіння активу.

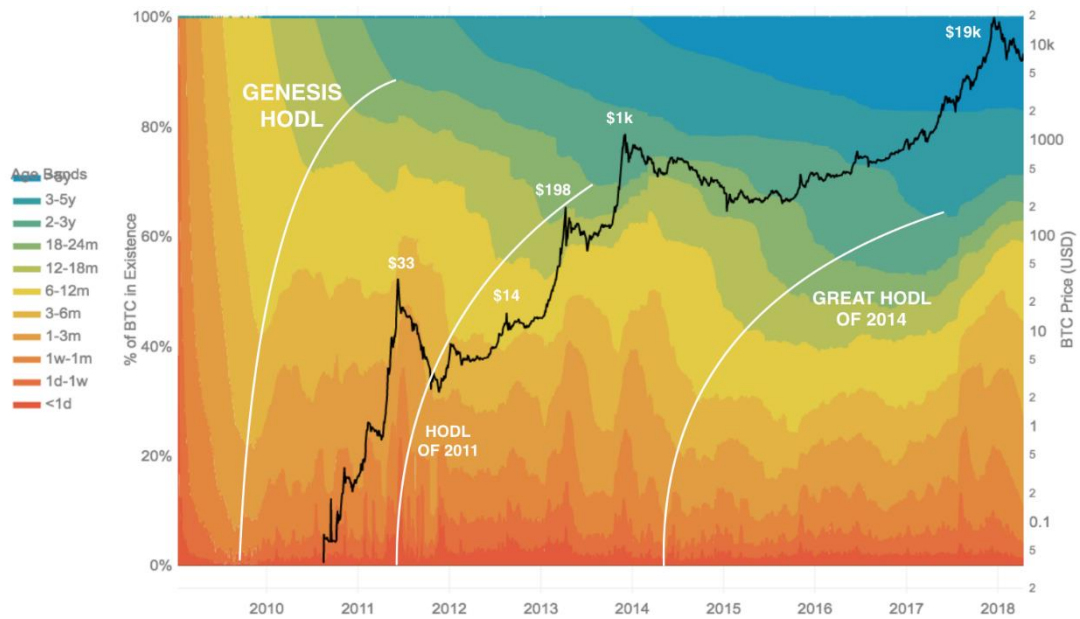


Рис.8. HODL хвилі на прикладі біткоїна за деякий проміжок [6]

- 2) Також можна привести NVT Ratio індикатор, який має в собі сутність відношення ринкової капіталізації до об'єму виконаних транзакцій в USD, цей індикатор позначає чи преоцінений актив. [7]

Для цього типу аналізу зручно використовувати сервіс GlassNode [7] скріншот основних доступних показників якого представлений на (рис.9.) на прикладі Біткоїна.

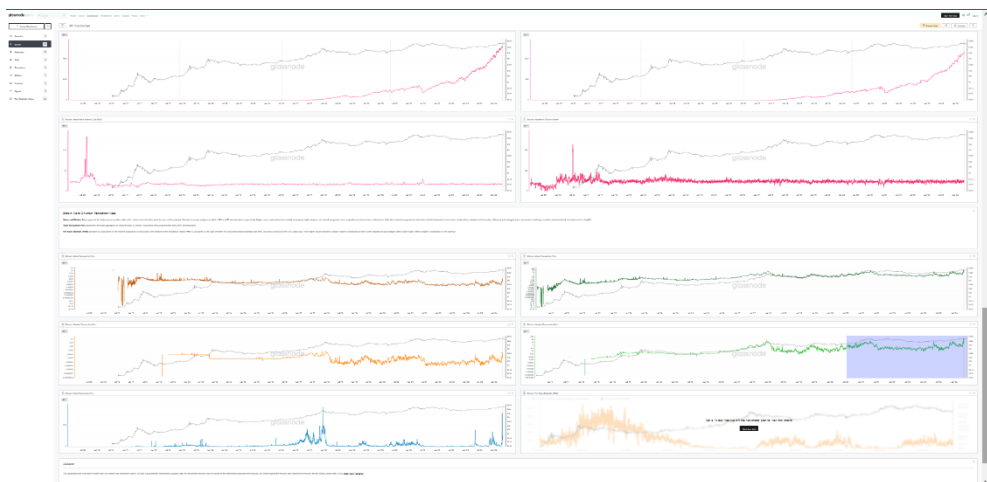


Рис.9. Приклад сервісу для on-chain аналізу, GlassNode

4. Аналіз настроїв:

Цей метод полягає у вивченні обговорень у соціальних мережах, новин, форумів з метою оцінити думку людей про ту чи іншу тему, той чи інший актив. Цей метод поділяється на 2 основних аналіз загальних настроїв широких мас людей, так званий аналіз соціальних мереж, та аналіз більш медіа ресурсів та новин загалом.

I. Аналіз соціальних мереж є відображенням настроїв людей шляхом вивчення та аналізу тем, емоційного забарвлення обговорень на соціальних платформах. Для прикладу, можна використовувати: Twitter, Reddit, Facebook та інші. Цей метод виявляє загальну соціальну думку та її наявний чи майбутній вплив на криптоактив. Позитивні коментарі з більшою вірогідністю позитивно вплинуть на вартість активу, а негативні – навпаки.

Як основні складові цього методу можна привести: збір даних через API соцмереж, аналіз самих настроїв через пропуск текстів через NLP мережі для вивчення тонального забарвлення тексту. Як NLP-мережу можна використовувати будь-який алгоритм обробки природньої мови та наступну класифікацію текстів по настроям. Після цього визначаються основні актуальні теми, які нараз найбільше цікавлять людей. Потім через постійний або періодичний моніторинг відстежуються зміни настроїв та їх вплив на ціну криптовалют.

II. Аналіз новин полягає у відслідковуванні закономірностей між новинами, великими подіями та змінами цін активів. З основних новин можна привести: війни, критичні показники економічних досягнень, зміни з законодавствах

країн, регуляторні дії з боку установ чи навпаки досягнення, відкриття.

Основними складовими є також по-перше – збір даних через обраних ресурсів, аналіз змісту, їх подальша класифікація по тональності та співвідношення з цінами. Після цього проводиться аналіз впливу через співставлення новини та зміни ціни.

5. Машинне навчання (ML)

З розвитком технологій машинного навчання з'явився ще один метод, назовемо його: метод Моделей Прогнозування цін. В ньому використовуються статистичні алгоритми, методи машинного навчання та складні математичні формули для прогнозування майбутніх цін на основі різних показників.

Одним з основних методів машинного навчання, які підходять під прогнозування цін є LSTM (Long Short-Term Memory) та GRU (Gated Recurrent Unit) моделі. Вони можуть знаходити закономірності в різних типах даних.

Однією з переваг цього методу можна вважати високу точність при наявності великих обсягів якісних даних. Якість даних та відсутність пропусків, так званих гепів, може відчутно вплинути на якість результатів предікшену.

З недоліків цього метода можна вважати: Необхідність мати великі обсяги даних та ресурсів. Необхідність аналізувати ці дані, заповнювати пропуски та проводити постійне донавчання моделей.

1.3. Аналіз існуючих готових додатків для передбачення фінансових ринків

В рамках роботи було знайдено та проаналізовано результати декількох готових додатків по аналізу не тільки цін, а і загалом числових рядів. Проведено пошук закономірностей по передбаченню майбутніх значень в цих роботах.

I. Фрактальний аналіз та прогнозування тенденцій фінансового часового ряду. [8]

В цій роботі запропоновано використовувати метод фрактального (рис.10) аналізу часових рядів. Цей метод оснований на показнику Херста та V-статистиці, що є альтернативами гіпотезі ефективності ринків. В цій гіпотезі фрактального ринку враховується як вплив ліквідності на ціни, так і показники інвестиційних горизонтів поведінки інвесторів. При цьому, жодне статистичне обмеження не діє на процес R/S-аналізу. Загалом це представлення набору статистичних методів для аналізу часових рядів, що дозволяють визначити їхні важливі характеристики: наявність неперіодичних циклів, пам'ять, чутливі зміни та інші показники. В рамках цієї роботи було проведено R/S-аналіз вартості акцій компанії IBM і надано прогноз щодо тенденцій цін на фінансових ринках.

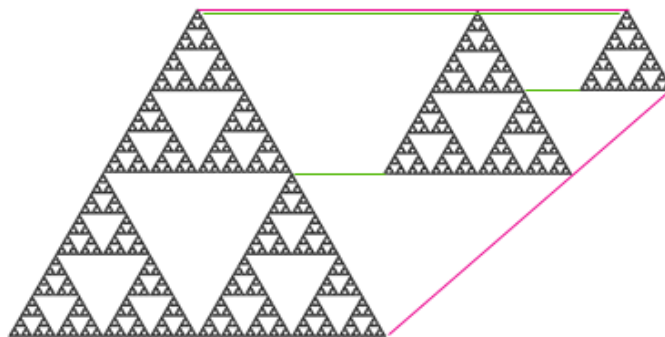


Рис.10. Зображення принципу фрактальності

II. Прогнозування тенденцій фінансових ринків згідно з теорією Еліота та сервіс Crypto forecaster. [9]

В рамках цієї роботи було проаналізовано різні фінансові ринки та була зроблена перевірка можливості прогнозування за допомогою Хвиль Еліота.

Теорія Хвиль Еліота (рис.11) ґрунтується на ідеї, що всі ринки рухаються згідно певних патернів, які є циклічними та повторюються. Згідно з цією теорією слідування цими патернами може допомогти в прогнозуванні цінових рухів.

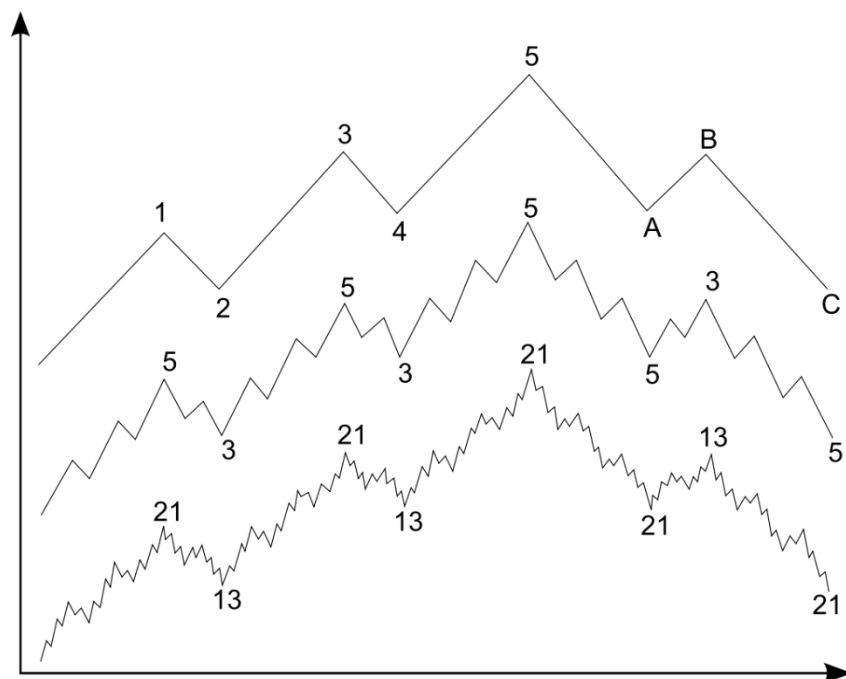


Рис.11. Зображення прикладу хвильового метода Ральфа Еліота

З основних принципів цього метода можна назвати:

- **Наявність імпульсних хвиль.** Згідно з цією теорією ринок рухається згідно головного тренду. Цей тренд складається з 5-ти хвиль, три з яких йдуть згідно основного напрямку, два – протилежно.
- **Наявність корекційних хвиль.** Кожен раз після закінчення імпульсного руху, напрям глобально змінюється на протилежний,

так званий корекційний. Цей рух складається вже з 3-х хвиль, дві з яких йдуть проти тренду, і тільки одна за ним.

- Фрактальний фундамент природи хвиль. Будь-яка окрема взята хвиля в свою чергу може бути поділена на менші хвилі, які мають таку саму структуру.

III. Розробка моделі видобутку знань для прогнозування фінансових ринків з виділенням типових тенденцій з часового ряду та технологія Knowledge Mining. [10]

В рамках цієї статті було проаналізовано та розроблено модель видобутку знань прогнозування залежностей часових рядів шляхом використання Knowledge Mining принципу. Досліджувана модель складається з часових рядів та прихованих тенденцій. Ряди є простою послідовністю зібраних даних в певні моменти часу. Тенденціями є самі патерни, які накладаються та знаходяться з цих рядів. Саме по цим тенденціям і робиться передбачення. Принцип Knowledge Mining складається з двох задач: прямої та зворотної. Згідно з прямою, ми робимо передбачення згідно з існуючих факторів, тоді як зворотна працює задля виявлення самих факторів які мали вплив на минулі, вже історичні, дані.

В рамках цієї роботи було проведено декілька методів оцінювання характеристик винайдених прихованих знань, а саме:

1. Оцінка несуперечності. Проводиться аналіз чи не суперечать винайдені знання один одному та наскільки вони доповнюють одне одне.
2. Оцінка повноти: Наскільки сильно та якісно ці винайдені знання покривають всі існуючі аспекти.

3. Оцінка зв'язності. Наскільки винайдені знання залежать та чи впливають одне з одного.

Загалом, було перевірено та продемонстровано, що оптимізація моделі згідно навчання її на розробленій вибірці дозволяє винайденим знанням набувати семантичну складову. Було доведено зв'язок між ступенями оцінки зв'язності та оцінкою повноти згідно принципу Knowledge Mining.

IV. Методологічний підхід до прогнозування інвестиційної привабливості фінансового ринку України. [11]

В цій роботі досліджувався процес формування інвестиційної привабливості різних країн та фінансових ринків. Основна ідея роботи складалась з дослідження та представлення методичного підходу до побудови динаміки фондових ринків та прогнозування поведінки їх учасників. Пропонується рівноважно-випадковий алгоритм на прикладі дослідження ринку опціонів.

V. Вейвлет-перетворення та векторні авторегресивні моделі на прикладі дослідження тенденцій фондового ринку України відповідно до зовнішніх чинників. [12]

В цьому дослідженні було проведено аналіз залежності зовнішніх чинників на фінансовий ринок України, використовуючи декілька методів вейвлет-перетворень та авторегресивні векторні моделі.

VI. Форсайт ефект для передбачення ринку нафти на світовому рівні. [13]

Згідно з наведеними дослідженнями в рамках цієї статі, використовуються до розкривається термін «форсайт» та його принципи його використання для передбачення. Загалом, цей метод є підходом дослідження можливостей та бажаних сценаріїв в довгострокових стратегіях.

З основних характеристик цього методу можна привести:

1. Періодичність. Прогнози гуртуються на теперішніх тенденціях та реаліях, відповідно до ймовірних майбутніх корекцій.
2. Спрямованість на побудові стратегій. Основними чинниками до прогнозів є сформовані стратегії.
3. Колективні рішення. Експерти та зацікавлені сторони мають вплив на побудову спільної стратегії.
4. Динамічність та інтерактивність. Постійні корекції відповідно до нових даних та чинників.

Згідно зі статтею, з'ясовано що відповідно до використаного методу, нафта залишиться все ще ключовим джерелом енергії в світі. Основними з факторів впливу на ринки нафтопродуктів є геополітичні, фінансові та макроекономічні чинники. Але при цьому, очікуються суттєві трансформації.

Цей метод дозволяє шляхом побудови різних сценаріїв майбутнього передбачати можливі зміни та впливати на майбутнє шляхом формування стратегій та дії відповідно до них.

VII. Прогнозування основних тенденцій демографічних процесів у різних регіонах України на прикладі Львівської області. [14]

Для більшої ясності про можливості використання аналізу методів часових рядів, було проаналізовано роботу, яка не стосується фінансових установ та інвестиційних можливостей.

В даному дослідженні було проведено детальний огляд стану демографії на прикладі Львівської області та показники народжуваності, смертності та міграції в певні моменти часу.

Представлена модель прогнозування кількості населення в майбутньому розраховує основні тенденції до 2050 року. Використовуючи статичне моделювання, було створено 3 можливі варіанти розвитку подій: оптимістичний, оптимальний та песимістичний. Згідно з результатами цього дослідження, було розроблено можливі дії по кожному з варіантів.

1.4. Дослідження сучасних концепцій обробки великих обсягів даних

1.4.1. Збір та зберігання даних

В рамках нашого завдання важливим завданням було проаналізувати та обрати спосіб збереження даних. Для нашого способу дуже важливим є швидкість читання даних для покращення швидкодії навчання моделей на великій кількості даних та і надійність збереження цих даних.

Таблиця порівняння з перевагами та недоліками основних існуючих способів збереження даних:

Таблиця 2

Порівняння основних типів баз даних та способів збереження

Тип Баз даних	Опис	Приклади	Переваги	Недоліки
Реляційні бази даних (SQL)	Збереження та управління строго типізованими та структурованими даними зі збереженням у вигляді таблиць, які мають відношення між собою.	MySQL PostgreSQL Oracle	1) Висока стабільність та надійність 2) Стандартизована мова запитів (SQL) 3) Підтримка транзакцій 4) Використання зовнішніх ключів для цілісності даних	1) Обмежена масштабованість при великих обсягах даних 2) Перевищення меж продуктивності при великій кількості одночасних запитів 3) Попередньо визначена схема даних
NoSQL бази даних	Неструктуровані або напівструктуровані дані..	MongoDB Cassandra Redis	1) Гнучкість у роботі з неструктурованими даними. 2) Здатність до горизонтального масштабування 3) Висока продуктивність	1) Відсутність стандартизованої мови запитів 2) Проблеми з цілісністю даних у розподілених системах

Таблиця 2 (продовження)

Порівняння основних типів баз даних та способів збереження

Тип Баз даних	Опис	Приклади	Переваги	Недоліки
Розподілені файлові системи	Збереження великих обсягів даних, розподілених між кількома серверами.	Hadoop HDFS Apache Cassandra	1) Висока масштабованість 2) Велика пропускна здатність	1) Відсутність структури даних 2) Обмежені можливості запитів
Хмарні сховища	Масштабовані рішення для збереження будь-якої кількості даних на віддалених серверах.	Amazon S3 Azure Blob Storage Google Cloud Storage	1) Легка масштабованість 2) Висока доступність та надійність	1) Змінна вартість використання 2) Потенційні обмеження контролю за безпекою. 3) Необхідність мати постійний стабільний інтернет зв'язок
Бази даних в оперативній пам'яті	Підхід закладається у збереженні всієї або тільки частини даних в кешах	Redis Memcached	1) Неймовірно висока швидкість доступу до даних 2) Підтримка різних структур та типів даних	1) Обмеженість в об'ємі та кількості даних для збереження 2) Нестабільність збереження та постійні ризики

Таблиця 2 (продовження)

Порівняння основних типів баз даних та способів збереження

Тип Баз даних	Опис	Приклади	Переваги	Недоліки
	оперативної пам'яті.			ризик втрати даних при нестабільному живленні оперативної пам'яті, можливість втрати даних
Часові ряди (Time-Series бази даних)	Системи для збереження та управління даними строго призначеними часовими мітками.	InfluxDB Prometheus TimescaleD В OpenTSDB	1) Оптимізовані для часових даних 2) Висока швидкість обробки	1) Обмежені можливості для нечасових даних 2) Потенційні складнощі з підтримкою та використанням складних структур даних

Згідно з проаналізовано інформацією, було з'ясовано що сучасні методи дозволяють якісно обробляти великі дані в будь-якому зручному форматі, тільки необхідно обрати потрібний формат та тип бази даних.

1.4.2. Попередня обробка та очищення даних

Попередній аналіз та обробка даних є критично важливими для покращення якості, точності та надійності результатів роботи моделей та самих моделей.

Представимо основні етапи попередньої обробки та очищення даних в Таблиці 3.

Таблиця 3

Основні етапи попередньої обробки та очищення даних

№	Назва етапу	Опис	Приклади методів
1	Об'єднання даних (Data Integration)	Злиття даних з різних джерел	Вирішення дублювання даних, уніфікація форматів
2	Очищення даних (Data Cleaning)	Виявлення та корекція помилкових, пропущених або неповних даних	Заповнення пропущених значень, видалення дублікатів, корекція аномалій
3	Трансформація даних (Data Transformation)	Зміна формату або структури даних	Нормалізація, агрегація, розбиття на категорії
4	Фільтрація даних (Data Filtering)	Відбір релевантних даних	Відбір даних за умовами, критеріями
5	Форматування даних (Data Formatting)	Перетворення даних у відповідний формат	Перетворення форматів дат, числових значень
6	Перевірка даних (Data Validation)	Перевірка точності та узгодженості даних	Перевірка діапазонів значень, перевірка форматів

Цей процес включає в злиття попередньо зібраних даних для подальшого аналізу. Після цього йде етап очистки даних. Наступним важливим етапом є трансформація їх в сумісний та категоризований вигляд. Їх подальша фільтрація та остаточне форматування для подальшого використання для навчання моделей.

Заповнення відсутніх даних може відбуватися декількома способами:

- Заповнення середніми значеннями

Заповнення середніми значеннями полягає в заміні відсутніх значень середнім значенням для цього атрибуту, при цьому використовується весь проміжок наявних значень.

Формула для обрахунку середнього значення наступна:

$$Mean = \frac{\sum_{i=1}^n x_i}{n} \quad (1.1)$$

Де x_i - значення атрибуту, n – кількість наявних значень

Приклад:

Розглянемо набір даних: [2,3,N,6,8]. Для обчислення середнього значення підставимо все в формулу та отримаємо:

$$Mean = \frac{\sum_{i=1}^n x_i}{n} = \frac{2+4+6+8}{4} = 5$$

Підставивши пропущене значення отримаємо наступний набір: [2,3,5,6,8].

- Заповнення медіаною

Заповнення медіаною полягає в заміні відсутніх значень медіаною для цього атрибуту. Медіаною є значення, що ділить набір на дві рівні частини.

Формула для обрахунку медіани наступна:

$$\begin{cases} Sorted(x) \left[\frac{n+1}{2} \right], \\ \frac{Sorted(x) \left[\frac{n}{2} \right] + Sorted(x) \left[\frac{n}{2} + 1 \right]}{2}, \end{cases} \quad (1.2)$$

Де Sorted(x) – впорядкований набір значень, n – кількість наявних значень

Приклад:

Розглянемо набір даних: [2,3,N,6,8]. Для обчислення медіани підставимо все в формулу та отримаємо:

$$Median = \frac{4 + 6}{2} = 5$$

Підставивши пропущене значення отримаємо наступний набір: [2,3,5,6,8].

- Заповнення модою

Заповнення модо полягає в заміні відсутніх значень найбільш часто зустрічаючим значенням в цьому наборі.

Формула:

$$Mode = \max(Frequency(x_1)) \quad (1.3)$$

Де $Frequency(x_1)$ – частота появи значення в наборі.

Приклад:

Розглянемо набір даних: [2,4,N,4,8]. Для обчислення моди підставимо все в формулу та отримаємо:

$$Mode = \max(Frequency(x_1)) = 4$$

Підставивши пропущене значення отримаємо наступний набір: [2,4,4,4,8].

- Заповнення використовуючи методи передбачення даних

Найчастіше використовується для заповнення «гепів» для подальшого навчання моделей машинного навчання.

- Лінійна регресія

Використовує лінійне співвідношення між атрибутами для подальшого розрахунку відсутніх значень.

Формула:

$$y = a + bx \quad (1.4)$$

Де y – пропущене значення, x – інші атрибути, a і b – коефіцієнти регресії.

- Метод найближчих сусідів

Використовує найближчі значення до пропущеного для обрахунку.

Формула:

$$y = \frac{1}{k} \sum_{i=1}^k y_i \quad (1.5)$$

Де y – пропущене значення, y_i – значення найближчих сусідів, k – кількість сусідів.

Можна зробити висновок, що кожен метод має свої переваги та недоліки та має використовуватись в залежності від характеру та типу даних, вимог до точності. Використання перших трьох методів є простими та швидкими, які підходять для більшості випадків. Методи передбачення дозволяють отримати більш точні результати, але вони потребують більшої кількості ресурсів та обчислень.

Обробка та чищення даних перед використанням є важливими етапами для покращення якості та точності подальших етапів.

1.4.3. Основні алгоритми для машинного навчання на великих даних

Існують велика кількість різних алгоритмів машинного навчання під різні задачі. Основні та найбільш застосовувані з них описано в таблиці 4.

Таблиця 4

Основні алгоритми машинного навчання на великих даних

Алгоритм	Опис	Формула
Гرادієнтний бустинг (Gradient Boosting)	Це комплексний метод для прогнозів ґрунтуючись на декількох менших та слабших моделях, таких як дерева рішень. Він добре переносить великі набори даних та може використовуватись в задачах регресії та класифікації.	Прогноз комбінованої моделі: $F(x) = \sum_{m=1}^M a_m h_m(x),$ Де $h_m(x)$ – слабкі моделі, a_m – ваги моделей

Таблиця 4 (продовження)

Основні алгоритми машинного навчання на великих даних

Алгоритм	Опис	Формула
Випадкові ліси (Random Forest)	Також комплексний метод, який заснований на застосуванні кількох дерев рішень та комбінує їх прогнози. З переваг можна позначити високу стійкість до перенавчання. Підходить під використання як в задачах класифікації, так і в регресії та важливості ознак.	$F(x) = \frac{1}{N} \sum_{i=1}^N h_i(x),$ Де $h_i(x)$- дерева рішень
Глибоке навчання (Deep Learning)	Ці типи моделей показують гарні результати роботи при навчанні на великій кількості даних. Найпоширеніші - нейронні мережі. Для обробки зображень використовуються згорткові нейронні мережі (CNN), для обробки послідовностей – рекурентні нейронні мережі (RNN), для обробки природної мови використовуються трансформаторні моделі.	Формула основної архітектури нейронних мереж: $y = f(Wx + b),$ Де W - ваги, x – вхідні дані, b - зсув, f – результуюча активаційна функція
LSTM (Long Short-Term Memory)	LSTM є покращеним варіантом звичайних рекурентних нейронних мереж. В них вирішено проблеми так званого відсутнього градієнту, яка може виникає при тренуванні RNN. LSTM працюють, отримуючи на вхід контрольований потік інформації	Формули для обчислення стану C та значення виходу h: $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$ $h_t = o_t \cdot \tanh(C_t)$

Таблиця 4 (продовження)

Основні алгоритми машинного навчання на великих даних

Алгоритм	Опис	Формула
	через нейрони. Добре зберігають довготривалі зв'язки між даними.	Де f_t , i_t , o_t – значення нейронів для забування, входу та виходу відповідно $\tilde{C}_t$ -кандидати на стан
GRU (Gated Recurrent Unit)	Це спрощена версія LSTM. Вони також мають нейрони на вході, але вони в собі мають меншу кількість параметрів, що робить їх швидшими.	Формула обчислення стану осередка: $h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t$ Де z_t – нейрони оновлення, $\tilde{h}_t$ – кандидати на стан

Машинне навчання на великих даних є потужним інструментом для аналізу та прогнозування в різних галузях. Воно має в собі велику кількість різних алгоритмів, моделей та технологій для навчання під різні задачі. Під кожен тип задач необхідно обрати саме ті моделі, які підійдуть найкраще.

1.5. Висновки до розділу 1

У даному розділі було проведено аналіз існуючих підходів до прогнозування цін криптовалют, а також досліджено сучасні концепції обробки великих обсягів даних.

Щодо аналізу існуючих підходів до прогнозування цін криптовалют, були визначені ключові фактори, що впливають на ціни криптовалют та розглянуті методи прогнозування, такі як технічний аналіз, фундаментальний аналіз, аналіз соціальних медіа тощо. Також було проведено аналіз існуючих готових додатків, які використовуються для прогнозування цін криптовалют.

У рамках дослідження сучасних концепцій обробки великих обсягів даних були розглянуті такі аспекти, як збір та зберігання даних, попередня обробка та очищення даних, машинне навчання на великих даних, обробка потокових даних в реальному часі.

За допомогою аналізу цих підходів та концепцій було виявлено ключові методи та інструменти, які можуть бути застосовані у контексті прогнозування цін криптовалютного ринку з використанням статистичних та аналітичних даних. Попередня обробка та аналіз цих даних має велике значення, оскільки впливає на якість прогнозів та рішень, прийнятих на їх основі.

Отже, цей розділ надає важливі відомості та основи для подальшого дослідження та розробки інтелектуальної системи прогнозування зміни цін криптовалютного ринку. Відповідно до аналізу, важливою частиною процесу буде ефективна обробка даних та вибір оптимальних методів аналізу для досягнення результатів.

РОЗДІЛ 2

СПОСІБ ІНТЕЛЕКТУАЛЬНОГО ПРОГНОЗУВАННЯ ЦІНОВИХ РУХІВ

У даній роботі пропонується спосіб інтелектуального прогнозування цінових рухів на криптовалютному ринку за допомогою технологій машинного навчання з автоматизованим донавчанням. Цей підхід забезпечує повністю автоматичний процес збору, збереження, обробки та аналізу даних, а також розрахунок підтримуваних індикаторів. Процес передбачає створення та використання моделей машинного навчання з різними конфігураціями та наборами шарів-даних, таких як різні цінові показники, значення об'єму торгів та різні торгові індикатори. Запропонований метод підходить для прогнозування цін на всі криптовалюти.

Запропонований підхід до прогнозування складається з:

1. Збору та агрегації даних
 - Використання наданих відкритих даних з основних бірж, доступних в Україні, на прикладі біржі Binance.
 - Автоматизований збір даних з бірж та їх поальше збереження в базу даних.
2. Попередня обробка даних:
 - Забезпечення відмовостійкості системи через перевірку помилок при завантаженні та зберіганні даних, щоб уникнути гепів та пропущених даних.
 - Заповнення відсутніх даних.
3. Розрахунок додаткових показників:
 - Забезпечення розрахунку та збереження різних фінансових індикаторів, які потім будуть використанні в якості додаткових вхідних шарів для навчання моделей.
4. Розробка та навчання моделей машинного навчання:

- Використання різних конфігурацій моделей для обробки даних.
- Навчання моделей на основі зібраних та оброблених даних.
- Тестування моделей для оцінки їхньої точності та ефективності.

5. Використання готових моделей:

- Після навчання моделі готові до використання для прогнозування напрямку руху цін криптовалют.
- Моделі видають прогноз з вірогідністю напрямку руху ціни який відбудеться через певний проміжок часу.

Переваги запропонованого методу:

- Автоматизація процесу збору, обробки та аналізу даних є повністю автоматизованим, що знижує можливість людської помилки в цьому процесі.
- Потенційна гнучкість та можливість задання різних конфігурацій до майбутніх моделей, що потенційно надає можливість адаптувати метод до будь-якої моделі.
- Можливість інтеграції з різними типами вхідних даних та різними джерелами.
- Здатність моделі адаптуватися до нових даних та змін на ринку. Можливість донавчання моделі в залежності від нових даних.

До потенціальних можливостей покращення системи можна віднести:

- Збільшення долі автоматизації процесу.
- Збільшення кількості можливих шарів даних, таких як використання більшої кількості фінансових індикаторів.
- Інтеграція з різними типами даних, такий як аналіз новинних ресурсів, соціальних мереж для більшої та більш повної картини ринку.
- Збільшення швидкості роботи системи шляхом покращення процесів та використання більшої кількості паралельних потоків.

- Збільшення відмовостійкості та стабільності системи.

2.1. Спосіб збору даних для подальшого використання, відмовостійкість при зборі та збереженні

За допомогою сайту CoinMarketCap було знайдено усіх потенційних провайдерів даних та проаналізовано обсяги торгівлі кожного з них. Для збору даних було обрано біржу Binance як основного провайдера даних, це дозволило нам отримувати історичні дані про ціни по великій кількості різноманітних пар та інтервалів, також дало доступ до цін в реальному часі.

Збір даних здійснюється шляхом використання відкритої API Binance.

Реалізація забезпечення відмовостійкості запропоновано наступними кроками (рис. 12).

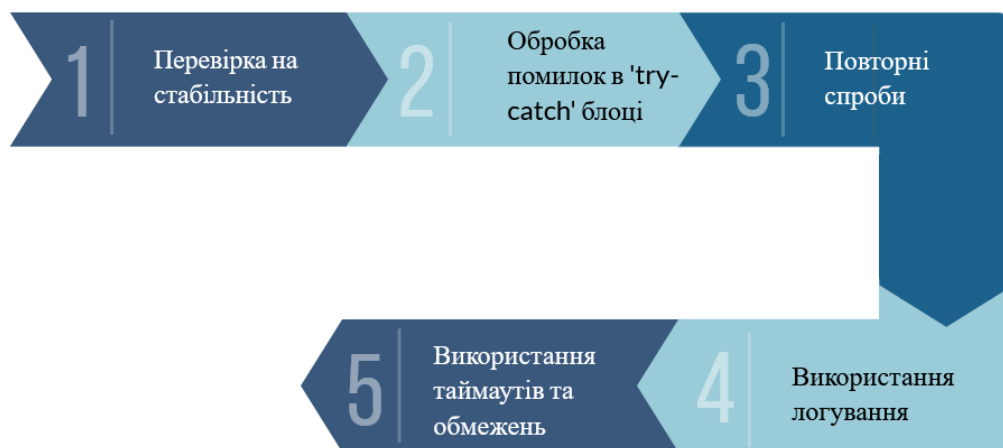


Рис.12. Процес забезпечення відмовостійкості

Алгоритм роботи завантаження даних програми:

1. Перед початком відправки запитів до біржі йде перевірка чи стабільний зв'язок, чи є доступ до АПІ.
2. Проводиться обробка помилок за допомогою блоку 'try-catch' для уникнення непередбачуваного виходу програми.

3. Використовуються повторні спроби запитів для загрузки даних, якщо запит до API не вдається зробити.
4. Використовується логування для реєстрації основних подій.
5. Встановлюються таймаути та обмеження на кількість запитів до API за певний період часу, щоб уникнути блокування.

2.2. Аналіз пропусків та артефактів у даних

Важливим етапом є обробка даних на наявність пропусків та артефактів. У цій роботі пропонується створення додатковий сервіс для аналізу даних, завантажених в базу даних. Цей модуль забезпечить заповнення пропусків та обробку артефактів перед переведенням даних у формат, необхідний для конкретної моделі машинного навчання.

Криптовалютні ринки працюють цілодобово, тому пропуски в даних можуть виникати через технічні збої, відсутність інтернет з'єднання, відсутність світла або інші причини. Пропуски у даних (наприклад, пропущений торговий день або свічка) необхідно заповнювати для забезпечення цілісності даних.

Метод заповнення пропущених даних медіаною полягає в заміні пропущених значень на медіану відповідного набору даних. Медіана є середнім значенням в відсортованому наборі даних. Якщо кількість елементів у наборі даних непарна, медіана - це значення в середині; якщо парна - медіана є середнім значенням двох значень в середині.

Важливою частиною обробки даних є ідентифікація артефактів.

Цінова свічка складається з декількох компонентів: ціна відкриття, ціна закриття, максимальне значення, мінімальне значення, останнє значення, об'єм та час. Артефакти можуть виникати через відсутність одного або кількох цих параметрів. Важливою частиною обробки даних є ідентифікація таких артефактів та їх коректне заповнення.

- Якщо артефакт є в ціні закриття (last), значення буде обиратись як середнє між open, close: $last = \frac{open+close}{2}$, або якщо якогось значення немає, то: між high та low: $last = \frac{high+low}{2}$
- Якщо артефакт є в значеннях open, close, high, low, значення буде обиратися як найближче минуле значення по цьому типу ціни.

2.3. Аналітичний спосіб розрахунку різних фінансових індикаторів

Важливою складовою слоїв моделі є використання фінансових індикаторів. В рамках роботи було обрано розраховувати та використовувати наступні індикатори: OBV, SMA, EMA, AO, SO, MFI, CMO.

Розглянемо кожен з обраних та реалізованих індикаторів.

1) On-Balance Volume (OBV)

OBV – це індикатор, який позначає, що коли індикатор збільшується покупці активно торгують та підіймають ціна активу. Коли індикатор знижується – об’єм продажів більше за об’єм покупок, тому показує спад ціни. Тому OBV підтверджує основний тренд. Приклад побудови індикатора зображено на (рис.13) за допомогою TradingView.

Алгоритм обчислення індикатора:

- Якщо ціна закриття сьогоднішнього дня вище вчорашньої:

$$OBV = OBV_{\text{попередній}} + Volume \quad (2.1)$$

- Якщо ціна закриття сьогоднішнього дня нижче вчорашньої:

$$OBV = OBV_{\text{попередній}} - Volume \quad (2.2)$$

- Якщо ціна закриття сьогоднішнього дня дорівнює вчорашній:

$$OBV = OBV_{\text{попередній}} \quad (2.3)$$



Рис.13. On-Balance Volume індикатор на прикладі Біткоїна за допомогою TradingView

2) Simple Moving Average (SMA)

Індикатор SMA допомагає згладжувати коливання ціни і визначати загальний напрямок тренду. Приклад побудови індикатора зображено на (рис.14) за допомогою TradingView.

Алгоритм обчислення індикатора:

- Розраховується як середнє значення ціни за певний період часу за формулою:

$$SMA = \frac{1}{n} \sum_{i=1}^n P_i, \quad (2.4)$$

Де P_i – ціна закриття за i -й день, n – кількість днів.



Рис.14. Simple Moving Average індикатор на прикладі Біткоїна за допомогою TradingView

3) Exponential Moving Average (EMA)

Це середнє значення ціни з більшим значенням для останніх доступних даних. ЕМА набагато швидше показує зміни в тенденціях за SMA. Приклад побудови індикатора зображено на (рис.15) за допомогою TradingView.

Алгоритм обчислення індикатора:

- Значення залежить від обраної кількості періодів, розраховується за формулою:

$$EMA = \left(\frac{P_t - EMA_{\text{попередній}}}{n+1} \right) + EMA_{\text{попередній}} \quad (2.5)$$

Де P_t – ціна закриття поточного періоду, n – кількість періодів



Рис.15. Exponential Moving Average індикатор на прикладі Біткоїна за допомогою TradingView

4) Awesome Oscillator (AO)

Цей індикатор вимірює різницю між 34-періодним та 5-періодним значеннями SMA, заснованих на середніх цінах: $\frac{high+low}{2}$. Приклад побудови індикатора зображено на (рис.16) за допомогою TradingView.

Формула обчислення індикатора:

$$AO = SMA_5 - SMA_{34} \quad (2.6)$$



Рис.16. Awesome Oscillator індикатор на прикладі Біткоїна за допомогою TradingView

5) Stochastic Oscillator (SO)

Цей індикатор створений для порівняння поточної ціни закриття з діапазоном цін за певний період. Він вимірює імпульс цінового руху. Приклад побудови індикатора зображено на (рис.17) за допомогою TradingView.

Формула обчислення індикатора:

$$SO = \frac{(C - L_{14})}{(H_{14} - L_{14})} \times 100 \quad (2.7)$$

Де C – поточна ціна закриття, L_{14} - найнижча ціна за останні 14 періодів, H_{14} - найвища ціна за останні 14 періодів.



Рис.17. Stochastic Oscillator індикатор на прикладі Біткоїна за допомогою TradingView

6) Money Flow Index (MFI)

Цей індикатор вимірює інтенсивність притоку і відтоку грошей в актив. Він враховує як і значення самої ціни активу, так і об'єми торгівлі. Приклад побудови індикатора зображено на (рис.18) за допомогою TradingView.

Формула обчислення індикатора:

$$MFI = 100 - \left(\frac{100}{1 + MFR} \right) \quad (2.8)$$

Де:

$$MFR (\text{money flow ratio}) = \frac{\text{Positive Money Flow}}{\text{Negative Money Flow}} \quad (2.9)$$

$$\text{Positive/Negative Money Flow} = \text{Typical Price} \times \text{Volume} \quad (2.10)$$

$$\text{Typical Price} = \frac{\text{High} + \text{Low} + \text{Close}}{3} \quad (2.11)$$



Рис.18. Money Flow Index індикатор на прикладі Біткоїна за допомогою TradingView

7) Chande Momentum Oscillator (CMO)

Це індикатор для виміру потужності імпульсів ціни на основі змін між поточними та попередніми цінами. Приклад побудови індикатора зображено на (рис.19) за допомогою TradingView.

Формула обчислення індикатора:

$$\text{CMO} = \frac{\sum_{i=1}^n (P_i - P_{i-1}) - \sum_{i=1}^n (P_{i-1} - P_i)}{\sum_{i=1}^n (P_i - P_{i-1}) + \sum_{i=1}^n (P_{i-1} - P_i)} \times 100 \quad (2.12)$$

Де: P_i – ціна закриття (close) за i -й день, n – кількість періодів.

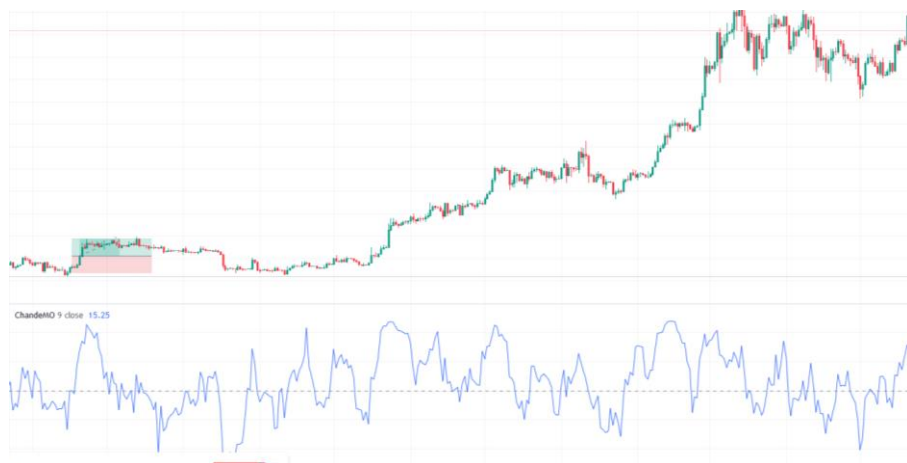


Рис.19. Chande Momentum Oscillator індикатор на прикладі Біткоїна за допомогою TradingView

2.4. Нормалізація даних та підготовка їх до використання у тренуванні моделі, тренування та тестування моделі

Коли дані вже завантажені в базу даних, додаткові індикатори обраховані, наступним важливим етапом є підготовка даних для подальшого використання в навчанні моделі. У цьому підрозділі описані кроки, які необхідні для нормалізації та підготовки даних для використання у тренуванні моделі (рис. 20).

Першим кроком у підготовці даних є їх розбиття на тренувальний, варіаційний та тестовий набори. Це дозволяє оцінити, як модель буде працювати на нових даних, і забезпечує більш об'єктивну оцінку її ефективності.

Нормалізація даних полягає в приведенні їх до одного масштабу. Це необхідно робити для забезпечення стабільної роботи алгоритмів машинного навчання.

Далі вже по вибраній моделі, обраних гіперпараметрах триває процес навчання моделі. Створена модель отримує тестові дані та триває оцінка якості роботи моделі.



Рис.20. Процес від перетворення даних до тестування готової моделі

2.5. Висновки до розділу 2

У даному розділі було розглянуто розробку способу інтелектуального прогнозування цінових рухів на криптовалютному ринку. Для досягнення цієї мети був застосований комплексний підхід, що включав збір та агрегацію різноманітних даних, їх обробку, а також використання методів машинного навчання.

Встановлено, що ефективне прогнозування цін на криптовалюту вимагає збирання різноманітних даних, включаючи інформацію про ціни на криптовалюту, обсяги торгів, індикатори. Дані були зібрані з різних джерел, таких як API біржі Binance і агреговані для подальшого аналізу.

Показано, що обробка даних у реальному часі є критично важливою для криптовалютного ринку через його високу волатильність і швидкість цінових змін. Запропоновано методи забезпечення відмовостійкості.

Розроблено підхід для ідентифікації та заповнення пропущених даних.

Розглянуто та використано аналітичні методи розрахунку фінансових індикаторів, таких як OBV, SMA, EMA, AO, SO, MFI, CMO. Ці індикатори використовуються для підвищення точності прогнозування.

Для прогнозування цінових рухів застосовано процес налаштування гіперпараметрів для підвищення ефективності моделей.

Проведено нормалізацію даних для забезпечення стабільності алгоритмів машинного навчання. Виконано розподіл даних на тренувальний, валідаційний та тестовий набори для об'єктивної оцінки моделей.

Отже, розроблений спосіб інтелектуального прогнозування цінових рухів забезпечує аналіз та прогнозування динаміки цін на криптовалютному ринку, що може бути корисним для трейдерів та інвесторів у прийнятті рішень. Цей підхід дозволяє отримувати важливу інформацію та швидко реагувати на зміни на ринку, підвищуючи ефективність управління ризиками та прийняття інвестиційних рішень.

РОЗДІЛ 3

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Обумовленість вибраних технологій

Програмна частина роботи написана мовою Java, що зумовлено наступними чинниками:

- Об'єктно-орієнтований підхід, що дозволяє створення модульної архітектури.
- Java дозволяє розробляти гнучкі та масштабовані проекти.
- Екосистема мови Java пропонує високу кількість різних бібліотек та фреймворків для роботи з даними, машинного навчання, обробки потоків та багатьох інших рішень.
- Java має вбудовану підтримку багатопоточності, що дозволяє ефективно виконувати багато різних незалежних задач одночасно.
- Платформна незалежність: Java дозволяє запускати розроблене програмне забезпечення на будь-якій операційній платформі, на якій стоїть віртуальна машина Java.
- Простота в використанні та величезна кількість вбудованих рішень, які одразу йдуть з SDK.

Застосовано технологію Docker, що забезпечує ізолюваність, що дозволяє уникнути конфліктів між різними версіями, дає можливість підіймати та запускати контейнери на будь-якому сервері.

Для збереження даних була обрана база даних PostgreSQL яка надає високу продуктивність та підтримку складних запитів. База даних PostgreSQL надає можливість горизонтального та вертикального масштабування, що дозволяє адаптувати сховище до зростаючих потреб системи.

Для створення програмної частини роботи також були використані наступні зовнішні бібліотеки такі як:

- Для підняття всього застосунку, підняттям контексту, впровадження залежностей та управління транзакціями – Spring фреймворк;
- Для роботи з базами даних - Spring JDBC, яка забезпечує простий контракт для виконання SQL-запитів та обробки результатів;
- Для полегшення собі життя та генерації великої кількості коду – Lombok;
- Фреймворк для машинного навчання та глибокого навчання - Deeplearning4j;
- Для роботи з багатовимірними масивами та лінійною алгеброю - ND4J (ND4S);
- Для зручної роботи з CSV файлами - Commons-CSV.

3.2 Опис програмних складових та компонентів

У рамках розробки системи інтелектуального прогнозування цінових рухів на криптовалютному ринку було створено два окремих проекти: crypto-prices-loader та crypto-prices-predictor. Кожен з цих проектів виконує свої унікальні функції, які в сукупності забезпечують повний цикл збору, обробки та аналізу даних для прогнозування цін криптовалют. Вони працюють з базою даних PostgreSQL.

3.2.1 Опис бази даних

Для зберігання даних про криптовалютні ціни, індикатори та інші пов'язані дані було використано базу даних PostgreSQL. Структура бази даних включає таблиці, зображені на (рис. 21).

- >  candles
- >  coin_pair
- >  coin_price_history
- >  coins_by_exchange
- >  databasechangelog
- >  databasechangelock
- >  exchange
- >  exchange_indicator

Рис.21. Створені таблиці для роботи проекту в базі даних

Таблиця candles зберігає типи свічок, що використовуються для аналізу цінових рухів (рис. 22).

	type character (3)	
1	1m	
2	3m	
3	5m	
4	15m	
5	30m	
6	1h	
7	2h	
8	4h	
9	8h	
10	12h	
11	1D	
12	3D	
13	1W	
14	1M	

Рис.22. Підтримувані інтервали свічок

Таблиця `exchange` включає інформацію про різні підтримувані біржі, включаючи назву та статус активності біржі (рис. 23).

```
1 SELECT exchange_name FROM exchange;
```

	exchange_name character varying	
1	Binance	

Рис.23. Підтримувані біржі

Таблиця coin_pair зберігає інформацію про пари криптовалют, включаючи базову та котирувану криптовалюту, а також їхній статус активності, включає наступні колонки (рис. 24).

Columns			
		Name	Data type
		id	integer
		base_coin	character varying
		quote_coin	character varying
		active	boolean

Рис.24. Структура таблиці coin_pair

- Id - Унікальний ідентифікатор валютної пари
- base_coin - Базова криптовалюта
- quote_coin – Відносна криптовалюта
- active - Значення чи торгуються криптовалюта

Актуальний список таблиці coin_pair складається з 116 пар.

Таблиця coins_by_exchange включає наступні колонки (рис. 25) та зберігає зв'язок між парами криптовалют та біржами, а також інформацію про підтримку P2P торгів.

		Name	Data type
		coins_pair_id	integer
		exchange_id	integer
		p2p_supported	boolean

Рис.25. Структура таблиці coins_by_exchange

Таблиця `coin_price_history` включає наступні колонки зображені на (рис. 26) та використовується для збереження самих складових ціни по кожній криптовалютній парі в кожен момент часу.

	Name	Data type
 	coin_pair_id	integer
 	open	numeric
 	high	numeric
 	low	numeric
 	close	numeric
 	volume	numeric
 	number_of_trades	bigint
 	candle_time_open	bigint
 	candle_time_close	bigint
 	candle_type	character varying
 	exchange_id	integer
 	id	bigint

Рис.26. Структура таблиці `coin_price_history`

- `id` - Унікальний ідентифікатор свічки
- `coin_pair_id` - Ідентифікатор валютної пари
- `exchange_id` – Ідентифікатор біржи
- `open` - Ціна відкриття свічки
- `close` - Ціна закриття свічки
- `high` - Максимальна ціна свічки
- `low` - Мінімальна ціна свічки
- `volume` - Обсяг торгів
- `number_of_trades` – кількість транзакцій зроблених протягом «існування» свічки
- `candle_time_open` - Часова мітка відкриття свічки

- `candle_time_close` - Часова мітка закриття свічки

Таблиця `exchange_indicator` включає наступні колонки зображені на (рис. 27) та використовується для збереження всіх розрахованих індикаторів по кожній криптовалютній парі.

	Name	Data type
 	id	bigint
 	calculated_value	numeric
 	indicator_type	character varying
 	coin_price_history_id	bigint
 	period	integer

Рис.27. Структура таблиці `exchange_indicator`

- `id` - Унікальний ідентифікатор свічки
- `coin_price_histoty_id` – колонка зв'язку значення індикатора з свічкою в момент часу
- `calculated_value` – саме значення індикатора
- `indicator_type` – тип індикатора
- `period` – Значення періода індикатора

Діаграма взаємозв'язків таблиць бази даних розташована на (рис. 28). Згідно неї, `coin_pair` пов'язана з `coins_by_exchange` через поле `coin_pair_id`, `exchange` пов'язана з `coins_by_exchange` через поле `exchange_id`, `coin_price_history` пов'язана з `coin_pair` через поле `coin_pair_id`, `coin_price_history` пов'язана з `exchange` через поле `exchange_id`, `exchange_indicator` пов'язана з `coin_price_history` через поле `coin_price_history_id`.

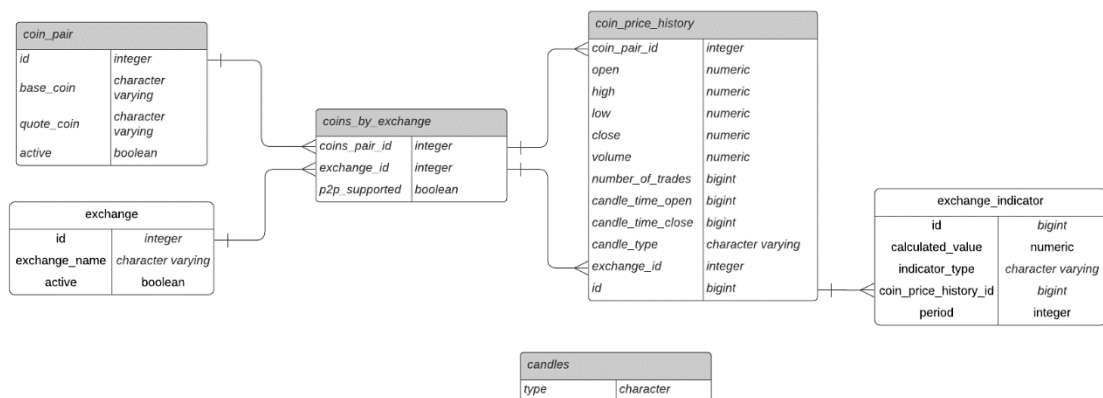


Рис.28. Структура таблиц бази даних

3.2.2 Проект *crypto-prices-loader*

crypto-prices-loader - це проект, призначений для збору та збереження даних про ціни криптовалют, розрахунок обраних індикаторів. Основні функції цього проекту включають:

1. Збір даних:

- Використання API бірж, таких як Binance, для отримання актуальних даних про ціни криптовалют.

2. Валідацію та обробку даних:

- Перевірка валідності отриманих даних.
- Обробка помилок та повторні спроби у випадку невдалих запитів.
- Логування процесів для відстеження та усунення можливих проблем.

3. Збереження даних:

- Зберігання валідних даних у базу даних PostgreSQL.
- Забезпечення відмовостійкості системи шляхом перевірки на помилки та обробки пропусків даних.

Основні компоненти включають (рис. 29):

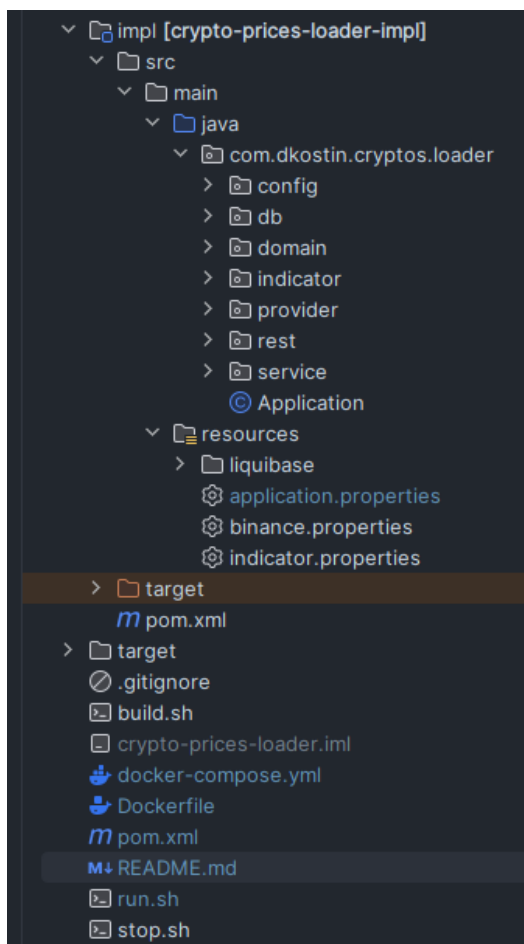


Рис.29. Основні компоненти проекту crypto-prices-loader

Директорія src/main/java/com/dkostin/cryptos/loader

- Пакет config: Містить конфігураційні класи для налаштування програми.
- Пакет db: Містить класи для взаємодії з базою даних.
- Пакет domain: Визначає основні об'єкти домену, такі як моделі даних.
- Пакет indicator: Включає класи для обробки фінансових індикаторів.
- Пакет provider: Відповідає за інтеграцію з провайдерами даних.
- Пакет rest: Містить REST контролери для обробки HTTP запитів.
- Пакет service: Включає сервіси для бізнес-логіки додатка.

Директорія src/main/resources

- liquibase: Конфігурації інструменту для управління версіями баз даних.
- application.properties: Основні конфігурації додатка.
- binance.properties: Конфігурації для інтеграції з біржею Binance.
- indicator.properties: Конфігурації для розрахунку фінансових індикаторів.

Додаткові загальні файли включають в себе:

- Dockerfile, який визначає як збирати Docker-образ проекту.
- docker-compose.yml: Конфігурація для запуску додатка за допомогою Docker Compose.
- pom.xml: Maven конфігураційний файл для управління залежностями та збіркою проекту.
- build.sh: Скрипт для збірки проекту.
- run.sh: Скрипт для запуску проекту.
- stop.sh: Скрипт для зупинки проекту.

Опис REST API проекту crypto-prices-loader:

Сам проект автоматизовано робить завантаження, обробку даних та розрахунок індикаторів одразу після старту та періодично, по заданому значенню, завантажує нові дані та робить додатковий обрахунок індикаторів.

Проте було реалізовані REST API інтерфейси, які забезпечують функціональність збору та аналізу даних про ціни на криптовалюту. Нижче наведено опис інтерфейсів LoaderResource та StatisticsResource, а також їхніх методів (рис. 30).

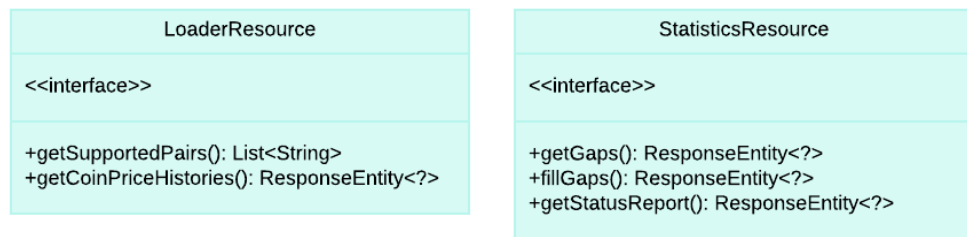


Рис.30. Реалізовані REST API інтерфейси в проекті crypto-prices-loader

Інтерфейс LoaderResource визначає основні операції для збору даних про підтримувані криптовалютні пари та історії цін. Він містить два методи:

- **getSupportedPairs.** Метод повертає список підтримуваних пар криптовалют. Цей метод дозволяє дізнатися, які криптовалютні пари підтримуються системою для збору та аналізу даних.
- **getCoinPriceHistories.** Метод повертає історичні дані про ціни на криптовалюту. Цей метод забезпечує доступ до історичних даних про ціни на криптовалюту, які можуть бути використані для подальшого аналізу або прогнозування.

Інтерфейс StatisticsResource визначає методи для аналізу пропусків у даних, їхнього заповнення та створення звітів про стан системи. Він містить три методи:

- **getGaps.** Метод повертає інформацію про пропуски у даних. Цей метод дозволяє отримати інформацію про наявні пропуски у даних, що необхідно для подальшого їхнього аналізу та заповнення.
- **FillGaps.** Метод заповнює пропуски у даних. Цей метод використовується для автоматичного заповнення пропусків у даних, що забезпечує цілісність та повноту інформації.
- **GetStatusReport.** Метод повертає звіт про стан системи. Цей метод забезпечує генерацію звіту про стан системи, включаючи статус

збору та обробки даних, що необхідно для моніторингу роботи системи.

Опис основних класів в дипломному проекті:

Клас `ExtractorService` є реалізацією сервісу для збору та обробки даних про ціни криптовалют. Він забезпечує автоматичний збір даних від різних провайдерів, обробку цих даних та запуск обчислення фінансових індикаторів. Нижче наведено детальний опис основних компонентів та методів цього класу.

Він складається з основних складових:

- Поля класу
 - `List<PriceBarProvider> barProviders`: Список провайдерів, які забезпечують дані про цінові бари.
 - `List<P2PPriceProvider> p2pProviders`: Список провайдерів, які забезпечують дані про P2P (peer-to-peer) транзакції.
 - `Map<Integer, CompletableFuture> barProvidersState`: Стан обробки даних для кожного провайдера цінових барів.
 - `Map<Integer, CompletableFuture> p2pProvidersState`: Стан обробки даних для кожного провайдера P2P транзакцій.
 - `IndicatorCalculationLauncher indicatorCalculationLauncher`: Лаунчер для обчислення фінансових індикаторів.
- Конструктор, для ініціалізації списків провайдерів та лаунчеру для обчислення індикаторів.
- Методи класу
 - `init()`: Метод ініціалізації, який виконується після створення екземпляра класу. Він налаштовує початковий стан провайдерів та запускає процес збору даних для цінових барів.

- `configureStartingProvidersState()`: Налаштовує початковий стан для всіх провайдерів.
- `regularBarDoExtraction()`: Метод для збору даних від провайдерів цінових барів, який запускається за розкладом. Він перевіряє стан обробки для кожного провайдера та завантажує нові дані, якщо обробка завершена.
- `p2pExtraction()`: Метод для збору даних від провайдерів P2P транзакцій, який запускається за розкладом. Він перевіряє стан обробки для кожного провайдера та завантажує нові дані, якщо обробка завершена.

Клас `ExtractorServiceImpl` є ключовим компонентом в системі збору та обробки даних про криптовалютні ціни. Він забезпечує автоматичний збір та обробку даних, що є необхідним для подальшого аналізу та прогнозування. Методи цього класу забезпечують високу надійність та ефективність процесу збору даних, що дозволяє системі своєчасно реагувати на зміни на ринку криптовалют. Він може бути використаний як основа для розширення функціональності системи шляхом додавання нових провайдерів даних, нових індикаторів та оптимізації процесу обробки даних.

Клас `StatisticsServiceImpl` реалізує інтерфейс `StatisticsService` і забезпечує функціональність для аналізу пропусків у даних, їх заповнення та створення звітів про стан системи.

Він складається з основних складових:

- Поля класу
 - `GapService gapService`: Сервіс для роботи з пропусками у даних.
 - `ExchangeService exchangeService`: Сервіс для роботи з біржами.

- CoinPairService coinPairService: Сервіс для роботи з парами криптовалют.
- StatusReportService statusReportService: Сервіс для створення звітів про стан системи.
- Конструктор, який ініціалізує сервіси для роботи з пропусками, біржами, парами криптовалют та звітами про стан системи.
- Методи класу
 - getGaps(): Метод для отримання інформації про пропуски у даних.
 - fillGaps(): Метод для заповнення пропусків у даних.
 - getReport(): Метод для створення звіту про стан системи.

Клас IndicatorCalculationLauncher забезпечує запуск обчислення фінансових індикаторів для пар криптовалют.

Він складається з основних складових:

- Поля класу
 - List<IndicatorCalculationService> indicatorCalculationServiceList: Список сервісів для обчислення індикаторів.
 - CoinPriceHistoryService coinPriceHistoryService: Сервіс для роботи з історією цін криптовалют.
 - ExchangeIndicatorService exchangeIndicatorService: Сервіс для роботи з індикаторами біржі.
 - CoinPairService coinPairService: Сервіс для роботи з парами криптовалют.
- Конструктор, який ініціалізує сервіси для обчислення індикаторів, роботи з історією цін криптовалют та парами криптовалют.
- Методи класу

- `startCalculations(int idProvider)`: Метод для запуску обчислення індикаторів для заданого провайдера.
- `calculateAndSaveIndicators(int idProvider, int coinPairId, String candleType, IndicatorCalculationService calculator)`: Метод для обчислення та збереження індикаторів.

Класи `StatisticsServiceImpl` та `IndicatorCalculationLauncher` є важливими компонентами системи для аналізу даних та обчислення фінансових індикаторів. Вони забезпечують автоматичний аналіз пропусків у даних, їх заповнення, створення звітів про стан системи та обчислення індикаторів для криптовалютних пар. Ці класи допомагають забезпечити високу надійність та точність системи, що є критично важливим для успішного прогнозування цін на криптовалютному ринку.

Основними структурами даних в рамках роботи є:

- Клас `CoinPrice`, який є представленням історичної свічки (рис. 31).

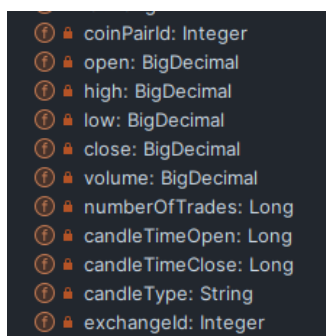


Рис.31. Структура класу та поля класу `CoinPrice`

- Клас `ExchangeIndicator`, який є представленням значення розрахованого індикатора в певний момент часу (рис. 32).

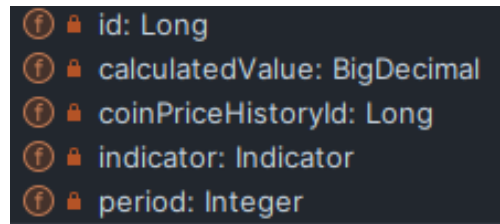


Рис.32. Структура класу та поля класу ExchangeIndicator

Описана структура проекту та основні компоненти забезпечують ефективний збір, обробку та аналіз даних про криптовалютні ринки. Це дозволяє створювати точні прогнози цінових рухів та забезпечувати високий рівень надійності системи.

3.2.3 Проект crypto-prices-predictor

crypto-prices-predictor - це проект, призначений для аналізу та прогнозування цін криптовалют на основі зібраних даних. Основні задачі цього проекту складають:

- Завантаження даних з бази даних PostgreSQL.
- Попередню обробку, включаючи нормалізацію.
- Використання різних алгоритмів машинного навчання для навчання моделей.
- Налаштування гіперпараметрів моделей.
- Тренування моделей на основі зібраних та оброблених даних.
- Використання навчених моделей для прогнозування напрямку руху цін криптовалют.
- Видача прогнозів з вірогідністю напрямку руху ціни через певний проміжок часу.

Основні компоненти включають (рис. 33):

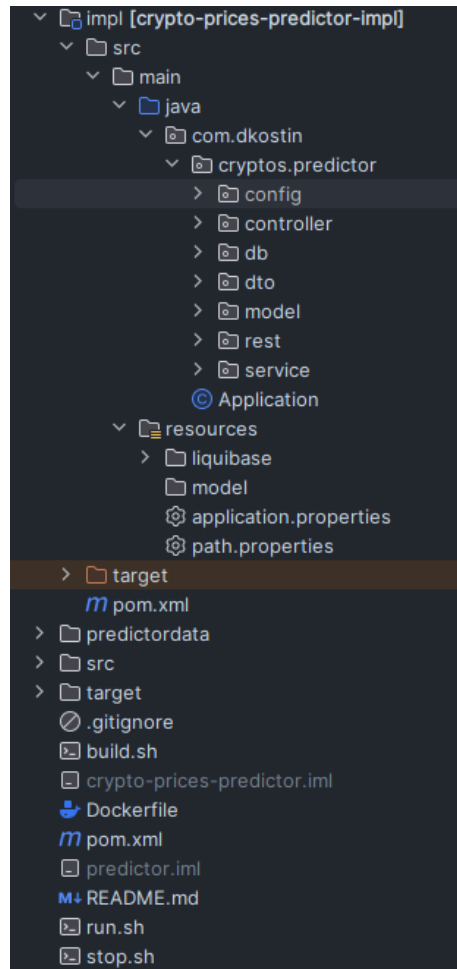


Рис.33. Основні компоненти проекту crypto-prices-predictor

Директорія src/main/java/com/dkoston/cryptos/predictor

- Пакет config: Містить конфігураційні класи для налаштування додатку.
- Пакет controller: Містить класи контролерів, які обробляють HTTP-запити і повертають відповіді на основі обробки даних та прогнозів.
- Пакет db: Містить класи для взаємодії з базою даних.
- Пакет dto: Містить об'єкти передачі даних (DTO), які використовуються для передачі даних між різними компонентами системи.
- Пакет model: Містить моделі машинного навчання для прогнозування цін на криптовалюту.

- Пакет `rest`: Веб-інтерфейси для доступу до функціональності додатку через HTTP-запити.
- Пакет `service`: Включає сервіси для бізнесу додатка.

Директорія `src/main/resources`

- `liquibase`: Конфігурації інструменту для управління версіями баз даних.
- `application.properties`: Основні конфігурації додатка.
- Директорія `model`: Збережені навчені моделі.
- `path.properties`: Конфігурації для шляхами до моделей, даних для навчання.

Додаткові загальні файли включають в себе:

- `Dockerfile`, який визначає як збирати Docker-образ проекту.
- `pom.xml`: Maven конфігураційний файл для управління залежностями та збіркою проекту.
- `build.sh`: Скрипт для збірки проекту.
- `run.sh`: Скрипт для запуску проекту.
- `stop.sh`: Скрипт для зупинки проекту.

Опис REST API проекту `crypto-prices-predictor`:

Проте було реалізовані REST API інтерфейси, які забезпечують функціональність аналізу та використання моделей прогнозування даних про ціни криптовалют. Нижче наведено опис інтерфейсів `LSTMResource` (рис. 34).

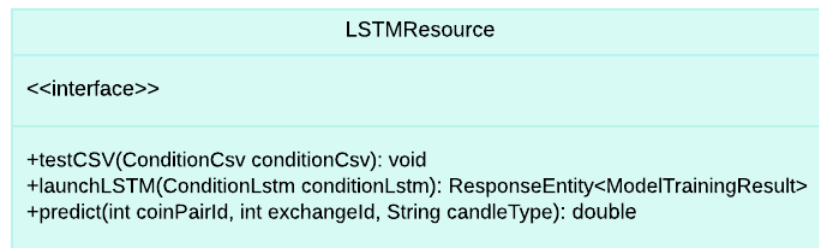


Рис.34. Реалізований REST API інтерфейс в проекті crypto-prices-predictor

Інтерфейс LSTMResource визначає основні операції для збору даних про підтримувані криптовалютні пари та історії цін. Він містить два методи:

- testCSV. Підготовлює дані до використання шляхом поділу на частини, зберігає дані у форматі CSV на основі заданих умов.
- launchLSTM. Запускає тренування LSTM моделі на основі заданих умов.
- predict. Виконує прогнозування ціни на основі заданих параметрів.
- downloadCsv. Завантажує CSV файл з результатами передбачення.

Опис основних класів в дипломному проекті:

Клас NormalizerMinMaxScaler є компонентом для нормалізації наборів даних, що використовується у процесі підготовки даних для машинного навчання. Цей клас дозволяє масштабувати значення ознак (і, за необхідності, міток) до діапазону між мінімальним і максимальним значеннями, за замовчуванням між 0 та 1. Це допомагає підвищити ефективність та точність моделей машинного навчання.

Методи класу:

- setFeatureStats. Встановлює статистику мінімальних та максимальних значень для ознак.
- setLabelStats. Встановлює статистику мінімальних та максимальних значень для міток.
- getTargetMin. Повертає мінімальне значення діапазону нормалізації.

- `getTargetMax`. Повертає максимальне значення діапазону нормалізації.
- `getMin`. Повертає мінімальні значення ознак.
- `getMax`. Повертає максимальні значення ознак.
- `getLabelMin`. Повертає мінімальні значення міток.
- `getLabelMax`. Повертає максимальні значення міток.
- `load`. Завантажує мінімальні та максимальні значення з файлів.
- `save`. Зберігає мінімальні та максимальні значення у файли.

Клас `NormalizerMinMaxScaler` є ефективним інструментом для масштабування значень ознак та міток у наборі даних до заданого діапазону. Використання цього класу допомагає забезпечити більш стабільне та точне навчання моделей машинного навчання за рахунок нормалізації даних. Основні функції класу включають завантаження та збереження статистики, налаштування діапазону нормалізації та обробку значень ознак та міток.

Клас `LSTMServiceImpl` є реалізацією сервісу для навчання і використання моделей LSTM (Long Short-Term Memory) для прогнозування цін на криптовалюту. Клас надає функціональність для навчання моделі на основі вхідних даних, збереження моделі, прогнозування на основі навченої моделі та оцінки її продуктивності.

Він складається з основних складових:

- Константи і поля:
 - Шлях до файлу резервного копіювання моделі.
 - Кількість вхідних параметрів для моделі.
 - Конфігурація для CSV файлів.
 - Об'єкт моделі `ComputationGraph`.

- Сервіс для роботи з історичними даними про ціни на криптовалюту. `StatusReportService statusReportService`: Сервіс для створення звітів про стан системи.
- Методи класу
 - `init ()`: ініціалізує модель, завантажуючи її з файлу, якщо він існує.
 - `saveModel ()`: `saveModel`.
 - `predict ()`: прогнозує ймовірність зростання або падіння ціни на основі останніх даних.
 - `dataForPredictionForm ()`: формує вхідні дані для прогнозування з історичних даних.
 - `launchLSTM ()`: відповідає за навчання моделі LSTM на основі вхідних даних, оцінювання її ефективності та збереження результатів.
 - `initModel ()`: ініціалізує модель LSTM з заданою конфігурацією.

Клас `LSTMServiceImpl` є центральним компонентом для реалізації прогнозування цін на криптовалюту за допомогою моделей LSTM. Він включає в себе функціональність для навчання моделі, збереження моделі, прогнозування цін та оцінки точності моделі. Використання цього сервісу дозволяє автоматизувати процес прогнозування та покращити точність передбачень на основі історичних даних.

Основними структурами даних в рамках роботи є:

- Клас `ConditionCsv` представляє параметри для конфігурації запиту на збереження CSV файлів історичних даних про ціни криптовалют. Ці параметри використовуються для ідентифікації

конкретної пари криптовалют, біржі, типу свічок та періоду часу.
(рис. 35).

```
toString(): String ↑Object
coinPairId: int = 1
exchangeId: int = 1
candleType: String = "1h"
startDate: long = 1420074000000L
```

Рис.35. Структура класу та поля класу ConditionCsv

coinPairId: Ідентифікатор пари криптовалют.

exchangeId: Ідентифікатор біржі.

candleType: Тип свічок (наприклад, "1h" означає свічки з годинними інтервалами).

startDate: Початкова дата в мілісекундах з початку епохи (Unix time).

- Клас ConditionLstm представляє конфігурацію для навчання моделі LSTM. Цей клас містить параметри для визначення умов та гіперпараметрів навчання моделі. (рис. 36).

```
coinPairId: int = 1
exchangeId: int = 1
candleType: String = "1h"
startDate: long = 1420074000000L
trainingPercent: double = 80
numberOfFilesForOneliteration: int = 10
nbEpochs: int = 10
learningRate: double = 0.005
batchSize: int = 64
lstmLayerSize: int = 200
numLabelClasses: int = 2
```

Рис.36. Структура класу та поля класу ConditionLstm

coinPairId: Ідентифікатор пари криптовалют.

exchangeId: Ідентифікатор біржі.

candleType: Тип свічок.

startDate: Початкова дата для навчання моделі.

trainingPercent: Відсоток даних, що використовуються для навчання.

numberOfFilesForOneIteration: Кількість файлів, що використовуються в одній ітерації навчання.

nbEpochs: Кількість epoch навчання.

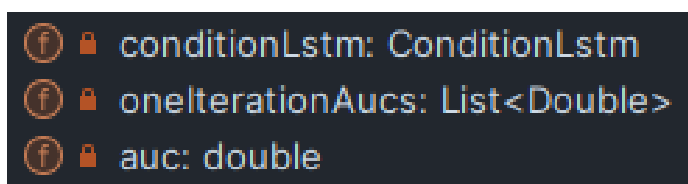
learningRate: Швидкість навчання.

batchSize: Розмір пакета (batch size).

lstmLayerSize: Розмір шару LSTM.

numLabelClasses: Кількість класів вихідного шару моделі.

- Клас `ModelTrainingResult` представляє результати навчання моделі LSTM. Він містить конфігурацію навчання, список значень AUC (Area Under Curve) для кожної ітерації та фінальне значення AUC (рис. 37).



```

(f) conditionLstm: ConditionLstm
(f) oneIterationAucs: List<Double>
(f) auc: double
  
```

Рис.37. Структура класу та поля класу `ModelTrainingResult`

conditionLstm: Конфігурація умов навчання LSTM.

oneIterationAucs: Список значень AUC для кожної ітерації навчання.

auc: Фінальне значення AUC.

- Клас `TestFiles` представляє модельні дані, що використовуються для тестування та оцінки моделі LSTM. Цей клас містить списки різних фінансових показників, які прогнозуються моделлю. (рис.38).

```

(f) openFromModel: List<Float> = new ArrayList<>()
(f) highFromModel: List<Float> = new ArrayList<>()
(f) lowFromModel: List<Float> = new ArrayList<>()
(f) closeFromModel: List<Float> = new ArrayList<>()
(f) volumeFromModel: List<Float> = new ArrayList<>()
(f) mfiFromModel: List<Float> = new ArrayList<>()
(f) obvFromModel: List<Float> = new ArrayList<>()
(f) emaFromModel: List<Float> = new ArrayList<>()
(f) smaFromModel: List<Float> = new ArrayList<>()
(f) cmoFromModel: List<Float> = new ArrayList<>()
(f) aoFromModel: List<Float> = new ArrayList<>()
(f) soFromModel: List<Float> = new ArrayList<>()
(f) riseFromModel: List<Float> = new ArrayList<>()
(f) fallFromModel: List<Float> = new ArrayList<>()
(f) resultRiseFromModel: List<Float> = new ArrayList<>()
(f) resultFallFromModel: List<Float> = new ArrayList<>()

```

Рис.38. Структура класу та поля класу TestFiles

openFromModel: Список прогнозованих значень відкриття.

highFromModel: Список прогнозованих значень максимуму.

lowFromModel: Список прогнозованих значень мінімуму.

closeFromModel: Список прогнозованих значень закриття.

volumeFromModel: Список прогнозованих значень обсягу торгів.

mfiFromModel: Список прогнозованих значень MFI (Money Flow Index).

obvFromModel: Список прогнозованих значень OBV (On-Balance Volume).

emaFromModel: Список прогнозованих значень EMA (Exponential Moving Average).

smaFromModel: Список прогнозованих значень SMA (Simple Moving Average).

cmoFromModel: Список прогнозованих значень CMO (Chande Momentum Oscillator).

aoFromModel: Список прогнозованих значень AO (Awesome Oscillator).

soFromModel: Список прогнозованих значень SO (Stochastic Oscillator).

riseFromModel: Список прогнозованих значень зростання.

fallFromModel: Список прогнозованих значень падіння.

resultRiseFromModel: Список результатів зростання.

resultFallFromModel: Список результатів падіння.

Ці класи забезпечують структуру для конфігурації, навчання та тестування моделей LSTM, що використовуються для прогнозування цін на криптовалюту..

3.3 Висновки до розділу 3

У даному розділі було виконано розробку програмного продукту для інтелектуального прогнозування цінових рухів на криптовалютному ринку. Програмний продукт включає два основні проекти: `crypto-prices-loader` та `crypto-prices-predictor`, які забезпечують збір, обробку, аналіз даних та прогнозування цін на криптовалюту.

Було проведено огляд засобів реалізації програмного додатку, обґрунтовано та наведено перелік необхідних сервісів для роботи додатку. Зазначено мінімальні системні вимоги для коректної роботи розробленої системи, обґрунтовано можливості Java в нашій задачі, а також визначено основні завдання, що мають бути вирішені програмно.

Розглянуто структуру програмного додатку, описано основні функціональні рішення, застосовані при розробці. Надана коротка змістовна інструкція по роботі з системою. Визначено основні кроки роботи розробленої

системи, сформовані та пов'язані між собою головні компоненти програмного рішення. Описано процеси збору даних з криптобірж, їх обробки, збереження.

Виконано моделювання системи прогнозування цінових рухів, продемонстровано роботу основних рівнів програмного рішення.

Особлива майбутня увага приділятиметься покращенню алгоритмів машинного навчання, використанню нових фінансових індикаторів та розширенню підтримуваних криптобірж та варіантів вхідних даних.

РОЗДІЛ 4

ТЕСТУВАННЯ ПРОГРАМИ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ ПРОЕКТУ

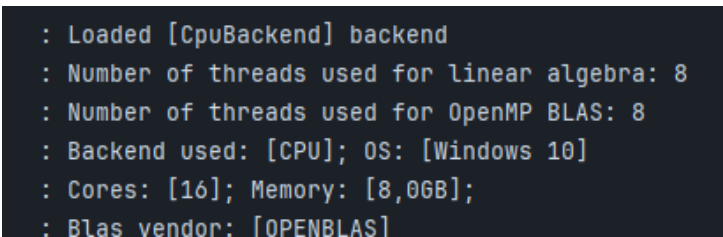
Основаючись на попередніх розділах, у даному розділі планується здійснити огляд інтерфейсу розробленої програми, провести тестування та аналіз розробленого способу інтелектуального прогнозування цінових рухів, зробити висновки. Серед запланованих досліджень передбачаються такі:

- Перевірка точності прогнозування моделей машинного навчання.
- Оцінка надійності та відмовостійкості системи.

4.1 Огляд розробленого програмного продукта

У цьому проекті було розроблено програмний продукт шляхом створення API, використання якого доступне будь-якою системою чи застосунком. Також є можливість розробити веб або мобільний інтерфейс, який використовуватиме нашу систему.

Одразу після запуску програми crypto-prices-loader виконується початкова перевірка та конфігурація відповідно до потужностей обладнання (рис.39).



```
: Loaded [CpuBackend] backend  
: Number of threads used for linear algebra: 8  
: Number of threads used for OpenMP BLAS: 8  
: Backend used: [CPU]; OS: [Windows 10]  
: Cores: [16]; Memory: [8,0GB];  
: Blas vendor: [OPENBLAS]
```

Рис.39. Початкова перевірка потужностей системи

Після цього виконується автоматична перевірка останніх наявних даних у базі даних по кожній підтримуваний парі, а потім триває автоматичне завантаження нових даних з сервісів провайдерів (рис.40).

```

: PriceBar provider loaded new information. Restarting now.
: Url: https://api.binance.com/api/v3/klines?startTime=1714330799999&symbol=BTCUSDT&interval=1h&limit=1000
: Initializing ExecutorService 'applicationTaskExecutor'
: Initializing ExecutorService 'taskScheduler'
: Tomcat started on port(s): 8081 (http) with context path ''
: Started QuotesPredictorApplication in 413.956 seconds (JVM running for 414.745)
: Url: https://api.binance.com/api/v3/klines?startTime=1716771599999&symbol=BTCUSDT&interval=1h&limit=1000
: Url: https://api.binance.com/api/v3/klines?startTime=1714330799999&symbol=ETHUSDT&interval=1h&limit=1000
: Url: https://api.binance.com/api/v3/klines?startTime=1716771599999&symbol=ETHUSDT&interval=1h&limit=1000
: Url: https://api.binance.com/api/v3/klines?startTime=1714330799999&symbol=BNBUSDT&interval=1h&limit=1000
: Url: https://api.binance.com/api/v3/klines?startTime=1716771599999&symbol=BNBUSDT&interval=1h&limit=1000
: Url: https://api.binance.com/api/v3/klines?startTime=1714334399999&symbol=XRPUSDT&interval=1h&limit=1000
: Url: https://api.binance.com/api/v3/klines?startTime=1716771599999&symbol=XRPUSDT&interval=1h&limit=1000
: Url: https://api.binance.com/api/v3/klines?startTime=1714334399999&symbol=ADAUSDT&interval=1h&limit=1000
: Url: https://api.binance.com/api/v3/klines?startTime=1716771599999&symbol=ADAUSDT&interval=1h&limit=1000

```

Рис.40. Процес автоматичного завантаження даних з сервісів провайдерів.

Як усі дані будуть завантажені та збережені почнеться автоматичний процес розрахунку фінансових індикаторів (рис. 41).

```

Calculating indicators of NPXS - ETH
Calculating indicators of IRIS - BTC
Calculating indicators of ZEC - BNB
Calculating indicators of ZEC - BTC
Calculating indicators of ZEC - ETH
Calculating indicators of FTT - BTC
Calculating indicators of XRP - ETH
Calculating indicators of XRP - BNB
Calculating indicators of LTC - ETH

```

Рис.41. Процес автоматичного розрахунку фінансових індикаторів.

На сьогодні, після завантаження та повного розрахунку всіх додаткових значень, наявна кількість даних складає близько 3 мільйонів значень для цінових показників та близько 15 мільйонів значень для фінансових індикаторів (рис. 42).

The image contains two side-by-side screenshots of a database query editor interface. Both screenshots show a 'Query Editor' tab with a SQL query and a 'Data Output' tab with the results of the query.

Left Screenshot:

- Query Editor:** Contains the SQL query: `1 SELECT count(*) FROM coin_price_history;`
- Data Output:** Shows a table with one column named 'count' of type 'bigint'. The first row contains the value '2900881'.

Right Screenshot:

- Query Editor:** Contains the SQL query: `1 SELECT count(*) FROM exchange_indicator;`
- Data Output:** Shows a table with one column named 'count' of type 'bigint'. The first row contains the value '14462460'.

Рис.42. Позначення кількості доступних даних на сьогоднішній день

Опис доступної API по завантаженню та статистиці даних:

1. Можливість ручного старту завантаження нових даних доступна за допомогою наступного ендпоінта:

`curl --location 'http://localhost:8081/extract/exchange'`

Після виклику одразу додається задача в чергу для подальшого завантаження нових даних.

2. Перевірка та отримання всіх гепів:

Перевірка та отримання всіх гепів доступна за допомогою наступного ендпоінта:

`curl --location 'http://localhost:8081/statistics/gaps'`

Після виклику цього ендпоінта буде повернуто всі актуальні гепи в базі даних. Результат після тестового довготривалого вимкнення мережі для перевірки зображено на рис. 43.

```

{
  "exchangeName": "Binance",
  "startDate": 1518049695786,
  "endDate": 1518173999999,
  "baseCoin": "BNB",
  "quoteCoin": "USDT",
  "candleType": "1h"
},
{
  "exchangeName": "Binance",
  "startDate": 1529978399999,
  "endDate": 1530017999999,
  "baseCoin": "BNB",
  "quoteCoin": "USDT",
  "candleType": "1h"
},
{
  "exchangeName": "Binance",
  "startDate": 1530104399999,
  "endDate": 1530111599999
}

```

я

Рис.43. Формат відповіді від ендпоінта по аналізу гепів

3. Заповнення всіх пропусків та гепів:

Заповнення всіх пропусків та гепів доступне за допомогою наступного ендпоінта:

```
curl --location 'http://localhost:8081/statistics/gaps/fill'
```

Одразу після виклику цього ендпоінта відбувається процес заповнення всіх пропусків та видалення зайвих даних з реєстрацією всіх подій в логах (рис. 44).

Після завершення процесу буде повернуто відповідь про успішне завершення процедури. Для перевірки чи все вірно було заповнено, викличемо ще раз ендпоінт з перевіркою та пошуком гепів. Результат показано на рис. 45.

```

: CoinPriceHistory(id=2942835, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942836, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942837, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942838, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942839, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942840, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942841, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942842, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942843, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942844, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942845, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942846, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942847, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942848, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942849, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942850, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942851, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942852, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942853, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942854, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c
: CoinPriceHistory(id=2942855, coinPairId=59, open=0.002500000, high=0.002500000, low=0.002500000, c

```

Рис.44. Реєстрація логів відповідно до процедури заповнення гепів.

```

{
  "1w": [],
  "12h": [],
  "15m": [],
  "30m": [],
  "8h": [],
  "3d": [],
  "1mn": [],
  "1d": [],
  "4h": [],
  "2h": [],
  "1h": [],
  "5m": [],
  "3m": [],
  "1m": []
}

```

Рис.45. Відповідь при відсутності гепів на момент виклику.

4. Є можливість отримати звіт по завантажених даних:

```
curl --location 'http://localhost:8081/statistics/report/'
```

5. Для підготовки даних до розбиття та їх подальшої нормалізації є

наступний ендпоінт:

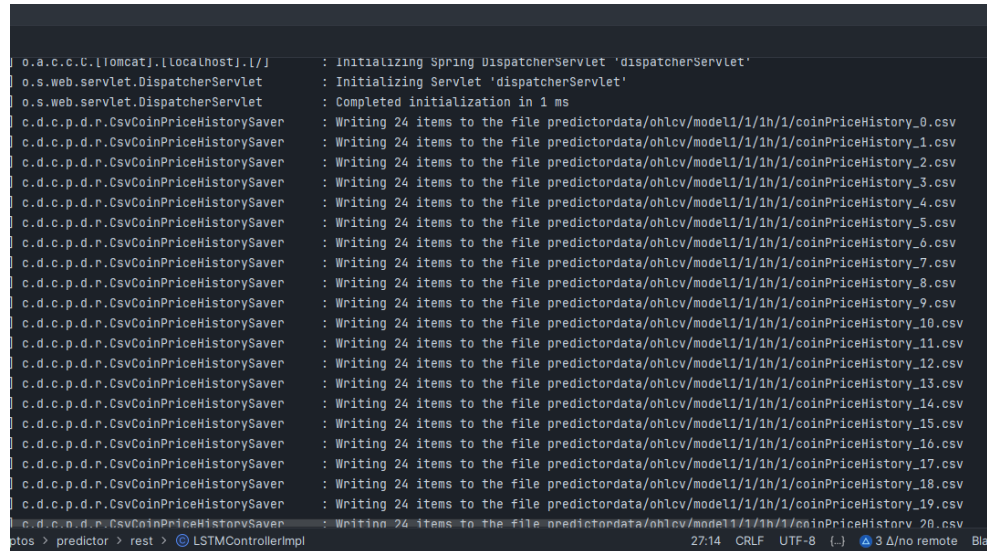
```

curl --location 'http://localhost:8082/testCSV' \
--header 'Content-Type: application/json' \
--data '{
  "coinPairId": 1,
  "exchangeId": 1,

```

```
"candleType": "1h",
"startDate": 1420074000000
}'
```

Після виклику починається процес обробки даних та збереження невеликих готових наборів даних у форматі csv файлів (рис. 46).



```
o.a.c.c.C.[tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_0.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_1.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_2.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_3.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_4.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_5.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_6.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_7.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_8.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_9.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_10.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_11.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_12.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_13.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_14.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_15.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_16.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_17.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_18.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_19.csv
c.d.c.p.d.r.CsvCoinPriceHistorySaver : Writing 24 items to the file predictordata/ohlcv/model1/1/1h/1/coinPriceHistory_20.csv
ptos > predictor > rest > @ LSTMControllerImpl 27:14 CRLF UTF-8 (-) 3 A/no remote Bla
```

Рис.46. Реєстрація логів відповідно етапу підготовки даних до подальшого використання.

6. Для безпосереднього навчання моделі використовується наступний ендпоінт:

```
curl --location 'http://localhost:8082/lstm' \
--header 'Content-Type: application/json' \
--data '{
  "coinPairId":1,
  "exchangeId":1,
  "candleType": "1h",
  "startDate": 1420074000000,
  "trainingPercent": 80,
  "numberOfFilesForOneIteration": 10,
  "nbEpochs": 10,
  "learningRate": 0.005,
  "batchSize": 64,
  "lstmLayerSize": 200,
```

```
"numLabelClasses": 2
}'
```

Під час якого триває навчання моделі згідно заданих параметрів та вхідних шарів даних. Результати зображено на (рис. 47).

```
Many auc for one iteration:
0.6170212765957447
0.6344086021505376
0.6517985611510791
0.6617760617760619
0.6772629310344829
0.6857825567502988
0.681958041958042
0.6877240143369175
0.6845522898154478
0.6829202586206896
0.6821350762527234
0.6864783142776151
0.6852246702992971
0.6844643933811214
0.6877394636015326
0.6867095168981961
0.6894334131343003
0.6882923807557298
```

Рис.47. Результати навчання та результати тестування навченої моделі.

7. Ендпоінт для передбачення застосовуючи навчену модель. У якості параметрів запиту передаються обрана біржа, валютна пара та інтервал на якому буде проводитись передбачення.

```
curl --location 'http://localhost:8082/predict?candleType=1h&coinPairId=1'
```

На рис. 48 можна побачити показники ймовірності у відсотках щодо напрямку руху ціни. Якість такого передбачення є досить високою.

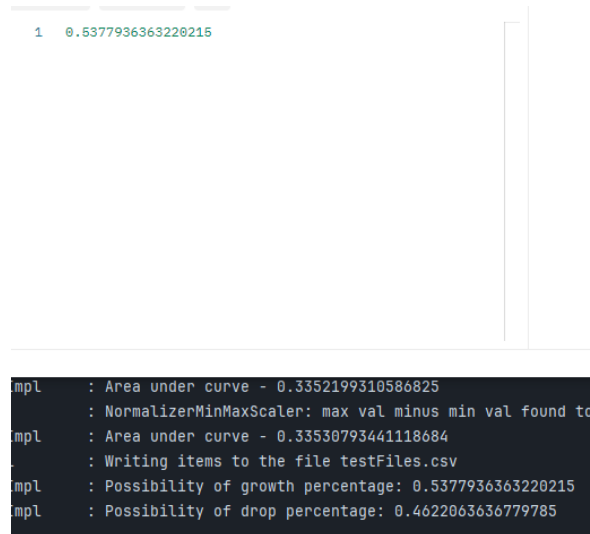


Рис.48. Результати виконання ендпоінту по передбаченню використовуючи обрану біржу, модель та інтервал.

Для вимірювання якості нашої моделі було обрано декілька характеристик, таких як:

AUC для оцінки моделей, які передбачають зростання або падіння цін.

MAE та RMSE для оцінки моделей, які передбачають точні значення цін на криптовалюту.

1. AUC (Area Under the Curve)

AUC (Area Under the Curve) - це метрика, яка використовується для оцінки продуктивності моделей. Вона визначає площу під кривою ROC (Receiver Operating Characteristic). ROC крива будується на основі відношення між чутливістю (True Positive Rate) та специфічністю (1 - False Positive Rate) для різних порогових значень.

Пояснення інтерпретування:

Коли $AUC = 1$, це вважається ідеально якісною моделлю.

Коли $AUC = 0.5$, це вважається, що модель не має дискримінаційної здатності, її точність випадкова.

Коли $AUC < 0.5$, вважається, що модель працює гірше не випадковість.

2. MAE (Mean Absolute Error)

Ця метрика визначає середнє значення абсолютних різниць між фактичними значеннями та передбаченими значеннями.

Пояснення інтерпретування:

Коли $MAE = 0$, це ідеальна модель, яка точно передбачає всі значення.

Чим менше значення MAE, тим краща модель: Низьке значення MAE свідчить про високу точність моделі.

Формула:

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \bar{y}_i| \quad (4.1)$$

Де, y_i - фактичні значення, $\bar{y}_i$ - передбачені значення, n - кількість спостережень.

3. RMSE (Root Mean Squared Error)

Згідно з цією метрикою виконується розрахунок коріня середньоквадратичної помилки, тобто корінь з середнього значення квадратів різниць між фактичними та передбаченими значеннями. Чим менша значення RMSE, тим якісніша модель.

Формула:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \bar{y}_i)^2} \quad (4.2)$$

Висновки до розділу 4

У даному розділі було проведено презентування, тестування та аналіз розробленої системи для інтелектуального прогнозування цінових рухів на криптовалютному ринку. Було розглянуто особливості роботи з розробленим API, програми та її основний функціонал.

1. API забезпечує доступ до основних функцій системи, включаючи запуск збору даних, перевірку пропусків, автоматичне заповнення гепів та аналіз результатів.
2. API дозволяє інтегрувати систему з іншими додатками та системами.

Проведено тестування процесу автоматичного завантаження даних з сервісів провайдерів. Перевірено процес розрахунку фінансових індикаторів.

За допомогою вбудованих функцій було проведено оцінку показників AUC, MAE та RMSE, що дозволило проаналізувати ефективність моделі.

Подальші дослідження можуть бути спрямовані на оптимізацію моделей та розширення функціональних можливостей системи шляхом збільшення кількості вхідних даних, таких як ін-чейн аналіз, аналіз соціальних мереж, новин для кращого розуміння ринку.

ВИСНОВКИ

Магістерська робота присвячена розробці системи інтелектуального прогнозування цінових рухів на криптовалютному ринку з використанням технологій машинного навчання.

Під час виконання кваліфікаційної роботи було проведено дослідження сучасного стану криптовалютного ринку, визначено основні аспекти та вектори розвитку, а також виконано технологічний огляд методів машинного навчання, зокрема, моделей LSTM та інших алгоритмів глибокого навчання. Розглянуто головні види та відкриті задачі прогнозування цін на криптовалютному ринку, виконано огляд існуючих рішень. Виявлено проблему, що актуальна для високоволатильних ринків: швидка зміна цін, великі обсяги даних та потреба в оперативному аналізі. Запропоновано спосіб вирішення даної проблеми із застосуванням сучасних підходів до обробки великих даних та машинного навчання.

Було сформовано основні завдання, які має вирішувати система прогнозування цін. Сформульовано основні етапи збору, обробки та аналізу даних. Визначено процес формування наборів даних для тренування моделей, а також процесів їхньої попередньої обробки та нормалізації. Зазначено аналітичні підходи для опрацювання отриманих даних та реалізації прогнозування цін. Окрім цього, описано основні етапи автоматичного завантаження даних з бірж, їхнього оброблення та збереження.

За проведеною розробкою реалізовано модель системи інтелектуального прогнозування цін. Виконано технологічний огляд засобів реалізації, обґрунтовано вибір мови програмування (Java), архітектури та сервісів. Надано проміжні результати процесів збору, обробки та аналізу даних, а також показано результуючі моделі прогнозування.

Завдяки технологічним та концептуальним особливостям, дана система може бути застосована для прогнозування цін для різних криптовалют. Запропоноване рішення є масштабованим. Окрім цього, методи роботи з великими даними та машинним навчанням дозволяють отримувати прогнози в режимі реального часу, що з практичної точки зору дозволяє трейдерам та інвесторам приймати більш обґрунтовані рішення.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Прогнозування ціни на криптовалюту [Електронний ресурс] // Binance, 2024. – Режим доступу до ресурсу: <https://www.binance.com/uk-UA/price-prediction/>
2. Графік історичних цін біткоїна - tradingview [Електронний ресурс] 2024. Режим доступу до ресурсу: <https://tradingview.com/symbols/BTCUSD/>
3. Біла книга Біткоїна – основний сайт з описом принципів роботи біткоїна від її творця [Електронний ресурс] 2024. Режим доступу до ресурсу: <https://bitcoin.org/bitcoin.pdf>
4. Біткоїн - coinmarketcap [Електронний ресурс] 2024. – Режим доступу до ресурсу: <https://coinmarketcap.com/uk/currencies/bitcoin/>
5. Офіційний ресурс з «сорсами» та кодом біткоїна [Електронний ресурс] 2024. – Режим доступу до ресурсу: <https://github.com/bitcoin/bitcoin>
6. HODL waves - [Електронний ресурс] // Binance, 2024. – Режим доступу до ресурсу: <https://www.binance.com/uk-UA/HODL-waves/>
7. BTC On-chain Activity Analysing - GlassNode - [Електронний ресурс], 2024. – Режим доступу до ресурсу: <https://studio.glassnode.com/dashboards/btc-core-on-chain>
8. Фрактальний аналіз та прогнозування тенденцій фінансового часового ряду, [Електронний ресурс], 2024 – Режим доступу до ресурсу: <http://repository.kpi.kharkov.ua/handle/KhPI-Press/37679>
9. Принцип роботи програм для прогнозування цін активів на фінансовому ринку, [Електронний ресурс], 2024 – Режим доступу до ресурсу: <http://ekmair.ukma.edu.ua/handle/123456789/18997>
10. Розробка моделі видобутку знань для прогнозування фінансових ринків з виділенням типових тенденцій з часового ряду, [Електронний ресурс], 2024 – Режим доступу до ресурсу: https://d1wqtxts1xzle7.cloudfront.net/33276954/business-inform-2014-1_0-

pages-114_117-libre.pdf?1395411868=&response-content-
 disposition=inline%3B+filename%3D33276954.pdf&Expires=1671052110&
 Signature=c87UyyYFriyJSeqrBA-
 x3rKOIsktTI2lgqvll6vz~iHN6ReM0BTGHIInACVdYqPHYIqDUdVKIUHb7
 GR8xavLj2uhZ8H5bQkDkL7xpDdUzjVA6K02YhUW-Yf-
 fYllmIDtCAYrOib3wpfxuKdmbf3cm7hiB-3EGaqymU9ufw0g5-
 mNbtnHet~MOe4lYHz0pnlA1SldmNWqwFX52RSEdmP8nTXkpGT2fYtcZ
 U0RTQt2fiuutJs6X9P1T9zTa-
 hHL1pA7alJt2V6m4JJKFPe02Y7Kpa5gLOMBmhQeZ3Z430cLPPdYDxR8
 ErcSp5i07a66kiTBd7RuKdXjwIbEJKtkQ1NNQ__&Key-Pair-
 Id=APKAJLOHF5GGSLRBV4ZA

11. Методологічний підхід до прогнозування інвестиційної привабливості фінансового ринку України, [Електронний ресурс], 2024 – Режим доступу до ресурсу: <http://ujae.wunu.edu.ua/index.php/ujae/article/view/221>
12. Вейвлет-перетворення та векторні авторегресивні моделі на прикладі дослідження тенденцій фондового ринку України відповідно до зовнішніх чинників, [Електронний ресурс], 2024 – Режим доступу до ресурсу: <http://dspace.nbuv.gov.ua/handle/123456789/54578>
13. Форсайт ефект для передбачення ринку нафти на світовому рівні, [Електронний ресурс], 2024 – Режим доступу до ресурсу: <https://essuir.sumdu.edu.ua/handle/123456789/72586>
14. Прогнозування основних тенденцій демографічних процесів у різних регіонах України на прикладі Львівської області, [Електронний ресурс], 2024 – Режим доступу до ресурсу: <http://dspace.nbuv.gov.ua/handle/123456789/68157>
15. Статистика індикаторів фінансової стійкості, [Електронний ресурс], 2024 – Режим доступу до ресурсу: https://bank.gov.ua/admin_uploads/article/methodological_notes_FSI.pdf

16. СТРУКТУРНІ ФІНАНСОВІ ІНДИКАТОРИ ЯК ОСНОВА ОЦІНКИ ТЕНДЕНЦІЙ НЕСТАБІЛЬНОСТІ, [Електронний ресурс], 2024 – Режим доступу до ресурсу: <http://www.economy.nayka.com.ua/?op=1&z=3356>
17. N. I. Raiterб PhD in economics, assistant professor of the Department of Statistics and Analysis, National Agrarian University, Lviv “THE MAIN INVESTMENT ATTRACTIVENESS INDICATORS”, vol 88, , 2014
18. Індикатори технічного аналізу в трейдингу, [Електронний ресурс], 2024 – Режим доступу до ресурсу: <https://ffin.ua/blog/articles/investopediia/post/indikatori-texnicnogo-analizu-v-treidingu-shho-soboyu-yavlyayut-ta-dlya-cogo-potribni>
19. Майорова Т.В. Інвестиційна діяльність: Навчальний посібник. – К.: «Центр навчальної літератури», 2004. – 376 с.
20. Галицька Е.В., Ковтун Н.В. Фінансова статистика: навчальний посібник. – К.: ВПЦ “Київський університет”, 2004. – 236 с.
21. Пересада А.А. Управління інвестиційним процесом. – К.: Лібра, 2002. – 472 с.
22. Корнеєва Ю.В. ”СИСТЕМА ІНВЕСТИЦІЙНИХ ІНДИКАТОРІВ”, 2012. - 82 с.
23. Державна служба статистики України [Електронний ресурс]. – Режим доступу: <http://www.ukrstat.gov.ua/>
24. Організаційно-методичні підходи до запровадження в НБУ системи оцінки стійкості фінансової системи: Інформаційно-аналітичні матеріали / За редакцією д.е.н., проф. В. І. Міщенко, к.е.н., доц. О. І. Кіреєва і к.е.н. М. М. Шаповалової – Київ : Центр наукових досліджень НБУ, 2005. – 97с.
25. Financial sector / The World Bank [Електронний ресурс]. – Режим доступу: <http://data.worldbank.org/topic/financial-sector>.
26. Karpenko, L., Rachynskyi, A., Dmytrenko, H., Akhlamov, A., & Oliynyk, N.(2020). Innovative entrepreneurship models for information economy

- development. *Journal of Entrepreneurship Education*, London, 23(S1), p.1-9 (Print ISSN: 1098-8394; Online ISSN: 1528-2651). *Digital Entrepreneurship & Education Models*. Available at: <https://www.abacademies.org/articles/Innovative-entrepreneurship-models-for1528-2651-23-S1-556.pdf>. Scopus
27. Karpenko, L.M. & Filyppova, S.V. (2016). Strategic competitive analysis of innovative enterprises development: predictive validity. *Actual problems of the economy*. – 2016. – № 6 (180). – P. 392 – 404. Scopus
28. Karpenko, L.; Zhylinska, O.; Zalizko, V.; Kukhta, P.; Vikulova, A. (2019) Human development in the context of provision of the social safety of society, *Journal of Security and Sustainability Issues* 8(4): 725-734. Available at: [https://doi.org/10.9770/jssi.2019.8.4\(15\)](https://doi.org/10.9770/jssi.2019.8.4(15)). Scopus
29. Стратегія економічного і соціального розвитку України “Шляхом Європейської інтеграції” / А.С.Гальчинській [та ін.]; Нац. ін-т стратег. дослідж., Ін-т екон. прогнозування НАН України, М-во економіки та з питань Європи. Інтер. України. – Офіц. вид. – Київ: [ІВЦ Держкомстату України], 2004. – 416 с
30. Паливода К. В. Проблеми формування привабливого інвестиційного клімату як засобу подолання деструкцій в економіці України К. В. Паливода // *Економіка та держава*. - 2018. - № 3. - С. 4-9.
31. Sebastien Donadio, Sourav Ghosh. *Learn Algorithmic Trading*. Birmingham - Mumbai: Packt Publishing, 2019. 378 p
32. Catania L., Grassi S. *Modelling Crypto-Currencies Financial Time-Series*. 2017. DOI : 10.2139/ssrn.3028486.
33. Kapila Tharanga Rathnayaka R.M., Seneiratna D.M.K.N, Arumawadu H.I. A New Financial Time Series Approach for Volatility Forecasting. // *Symposium on Statistical & Computational Modelling with Applications*. 2016, P. 5–8

34. Brockwell P.J., Davis R.A. Introduction to Time Series and Forecasting. New York : Springer, 2016. 425 p.
35. Barbulescu A. Bautu E. A Hybrid Approach for Modeling Financial Time Series. // The International Arab Journal of Information Technology. 2012, Vol. 9, N. 4, P. 327–335
36. Dingli A., Fournier K.S. Financial Time Series Forecasting – A Machine Learning Approach. // Machine Learning and Applications : An International Journal (MLAIJ). 2017, Vol.4, No.1/2/3.

ДОДАТОК А

Інтелектуальна система прогнозування цінових рухів
криптовалютного ринку

ЛІСТИНГ ПРОГРАМНОГО КОДУ

Аркушів 56

Київ – 2024

```

package com.dkostin.cryptos.loader.config;

import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfig {
}

package com.dkostin.cryptos.loader.db.dao.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.BatchPreparedStatementSetter;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.List;

import com.dkostin.cryptos.loader.db.dao.CoinPairDao;
import com.dkostin.cryptos.loader.db.rowmapper.CoinPairRowMapper;
import com.dkostin.cryptos.loader.domain.CoinPair;

@Repository
public class CoinPairDaoImpl implements CoinPairDao {
    private final String SAVE_QUERY = "INSERT INTO coin_pair (id, base_coin, quote_coin, active) VALUES (?, ?, ?, ?)";
    private final String FIND_MAX_ID_QUERY = "SELECT COALESCE(MAX(id), 0) FROM coin_pair";
    private final String UPDATE_QUERY = "UPDATE coin_pair SET base_coin=?, quote_coin=?, active=? WHERE ID=?";
    private final String FIND_ALL_QUERY = "SELECT * FROM coin_pair";
    private final String FIND_BY_ID_QUERY = "SELECT * FROM coin_pair WHERE ID=?";
    private final String FIND_BY_BASE_COIN_QUERY = "SELECT * FROM coin_pair WHERE base_coin=?";
    private final String FIND_BY_QUOTE_COIN_QUERY = "SELECT * FROM coin_pair WHERE quote_coin=?";
    private final String DELETE_QUERY = "DELETE FROM coin_pair WHERE id=?";
    private final String DELETE_ALL_QUERY = "DELETE FROM coin_pair";
    private final String FIND_ACTIVE_COIN_PAIRS_BY_EXCHANGE = "SELECT coin_pair.id, coin_pair.base_coin, coin_pair.quote_coin, coin_pair.active " +
        "FROM coins_by_exchange JOIN coin_pair ON coins_by_exchange.coins_pair_id = coin_pair.id " +
        "WHERE coins_by_exchange.exchange_id=? AND coin_pair.active='true' AND p2p_supported = ?";

```

```

private JdbcTemplate jdbcTemplate;

@Autowired
public CoinPairDaoImpl(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
}

@Override
public CoinPair save(CoinPair coinPair) {
    int nextId = jdbcTemplate.queryForObject(FIND_MAX_ID_QUERY, Integer.class)
+ 1;
    jdbcTemplate.update(SAVE_QUERY, nextId, coinPair.getBaseCoin(),
coinPair.getQuoteCoin(), coinPair.getActive());
    return coinPair;
}

@Override
public List<CoinPair> saveAll(List<CoinPair> coinPairs) {
    int maxId = jdbcTemplate.queryForObject(FIND_MAX_ID_QUERY, Integer.class);
    for (CoinPair coinPair : coinPairs) {
        coinPair.setId(++maxId);
    }
    this.jdbcTemplate.batchUpdate(
        SAVE_QUERY,
        new BatchPreparedStatementSetter() {

            public void setValues(PreparedStatement ps, int i) throws SQLException {
                ps.setInt(1, coinPairs.get(i).getId());
                ps.setString(2, coinPairs.get(i).getBaseCoin());
                ps.setString(3, coinPairs.get(i).getQuoteCoin());
                ps.setBoolean(4, coinPairs.get(i).getActive());
            }

            public int getBatchSize() {
                return coinPairs.size();
            }
        });
    return coinPairs;
}

@Override
public void deleteById(int id) {
    jdbcTemplate.update(DELETE_QUERY, id);
}

@Override
public void deleteAll() {
    jdbcTemplate.update(DELETE_ALL_QUERY);
}

```

```

@Override
public List<CoinPair> findAll() {
    return jdbcTemplate.query(FIND_ALL_QUERY, new CoinPairRowMapper());
}

@Override
public CoinPair findById(int id) {
    return jdbcTemplate.queryForObject(FIND_BY_ID_QUERY, new
CoinPairRowMapper(), id);
}

@Override
public void update(CoinPair updatedCoinPair) {
    jdbcTemplate.update(UPDATE_QUERY, updatedCoinPair.getBaseCoin(),
updatedCoinPair.getQuoteCoin(), updatedCoinPair.getActive(), updatedCoinPair.getId());
}

@Override
public List<CoinPair> findByBaseCoin(String baseCoin) {
    return jdbcTemplate.query(FIND_BY_BASE_COIN_QUERY, new
CoinPairRowMapper(), baseCoin);
}

@Override
public List<CoinPair> findByQuoteCoin(String quoteCoin) {
    return jdbcTemplate.query(FIND_BY_QUOTE_COIN_QUERY, new
CoinPairRowMapper(), quoteCoin);
}

@Override
public List<CoinPair> getActiveCoinPairsByExchange(int exchangeId, boolean
isPeerToPeerPairs) {
    return jdbcTemplate.query(FIND_ACTIVE_COIN_PAIRS_BY_EXCHANGE, new
CoinPairRowMapper(), exchangeId, isPeerToPeerPairs);
}
}

package com.dkostin.cryptos.loader.db.dao.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.BatchPreparedStatementSetter;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.jdbc.support.KeyHolder;
import org.springframework.stereotype.Repository;

import java.sql.PreparedStatement;

```

```

import java.sql.SQLException;
import java.util.List;
import java.util.Optional;

import com.dkostin.cryptos.loader.db.dao.CoinPriceHistoryDao;
import com.dkostin.cryptos.loader.db.rowmapper.CoinPriceHistoryRowMapper;
import com.dkostin.cryptos.loader.domain.CoinPriceHistory;
import com.dkostin.cryptos.loader.domain.Indicator;
import com.dkostin.cryptos.loader.domain.Interval;

@Repository
public class CoinPriceHistoryDaoImpl implements CoinPriceHistoryDao {

    private final String SAVE_QUERY = "INSERT INTO coin_price_history
(coin_pair_id, open, high, low, close, volume, number_of_trades, " +
    "candle_time_open, candle_time_close, candle_type, exchange_id) VALUES
(?,?,?,?,?,?,?,?,?,?,?)";
    private final String UPDATE_QUERY = "UPDATE coin_price_history SET open=?,
high=?, low=?, close=?, volume=?, number_of_trades=," +
    "candle_time_open=? WHERE coin_pair_id=? AND exchange_id=? AND
candle_time_close=? AND candle_type=?";
    private final String FIND_ALL_QUERY = "SELECT * FROM coin_price_history";
    private final String FIND_BY_ID_QUERY = "SELECT * FROM coin_price_history
WHERE id=?";
    private final String FIND_BY_COIN_PAIR_ID_QUERY = "SELECT * FROM
coin_price_history WHERE coin_pair_id=?";
    private final String FIND_BY_EXCHANGE_ID_QUERY = "SELECT * FROM
coin_price_history WHERE exchange_id=?";
    private final String DELETE_QUERY = "DELETE FROM coin_price_history
WHERE id=?";
    private final String DELETE_ALL_QUERY = "DELETE FROM coin_price_history";
    private final String FIND_LAST_LOADED_HISTORY_DATE = "SELECT
MAX(candle_time_close) FROM coin_price_history WHERE exchange_id = ? AND
coin_pair_id = ? AND candle_type = ?";
    private final String FIND_ALL_CANDLE_TYPES_IN_DATABASE = "SELECT
DISTINCT candle_type FROM coin_price_history cph WHERE coin_pair_id = ? AND
exchange_id = ?";

    private final String FIND_LAST_CALCULATED_HISTORIES_QUERY = "SELECT
* FROM coin_price_history " +
    "INNER JOIN exchange_indicator ON coin_price_history.id =
exchange_indicator.coin_price_history_id " +
    "WHERE exchange_id = ? AND coin_pair_id = ? AND candle_type=? AND
indicator_type = ? AND period = ?" +
    "ORDER BY candle_time_close DESC " +
    "LIMIT ?";

```

```

private final String
FIND_ALL_UN_CALCULATED_COIN_PRICE_HISTORIES_QUERY = "SELECT * "
+
    "FROM coin_price_history " +
    "WHERE candle_time_close > " +
    "(SELECT COALESCE(MAX(candle_time_close), 0) " +
    "FROM coin_price_history cph INNER JOIN exchange_indicator ei " +
    "ON cph.id = ei.coin_price_history_id " +
    "WHERE exchange_id = ? AND coin_pair_id = ? AND candle_type = ? AND
ei.indicator_type = ? AND ei.period = ?) " +
    "AND exchange_id = ? AND coin_pair_id = ? AND candle_type = ?" +
    "ORDER BY candle_time_close " +
    "LIMIT ?; ";

```

```

private final String FIND_BY_EXCHANGE_COIN_PAIR_INTERVAL_QUERY =
"SELECT * FROM coin_price_history WHERE exchange_id = ? AND" +
    " coin_pair_id= ? AND candle_type = ? AND candle_time_close > ? ORDER BY
coin_price_history.candle_time_close ASC LIMIT ?";

```

```

private final String FIND_PREVIOUS_DATA = "SELECT * FROM
coin_price_history " +
    "JOIN exchange_indicator ON coin_price_history.id =
exchange_indicator.coin_price_history_id " +
    "WHERE coin_price_history.exchange_id = ? AND
coin_price_history.coin_pair_id = ? AND coin_price_history.candle_type = ? " +
    "AND exchange_indicator.indicator_type = ? AND
coin_price_history.candle_time_close <= ? AND exchange_indicator.period = ?" +
    "ORDER BY coin_price_history.candle_time_close DESC " +
    "LIMIT ?";

```

```

private final String FIND_LAST_DATE = "SELECT
COALESCE(MAX(candle_time_close), 0) " +
    "FROM coin_price_history cph INNER JOIN exchange_indicator ei " +
    "ON cph.id = ei.coin_price_history_id " +
    "WHERE exchange_id = ? AND coin_pair_id = ? AND candle_type = ? AND
ei.indicator_type = ? AND ei.period = ?";

```

```

private JdbcTemplate jdbcTemplate;

```

```

@Autowired

```

```

public CoinPriceHistoryDaoImpl(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
}

```

```

@Override

```

```

public CoinPriceHistory save(CoinPriceHistory coinPriceHistory) {
    KeyHolder keyHolder = new GeneratedKeyHolder();
    jdbcTemplate.update(connection -> {
        PreparedStatement ps = connection.prepareStatement(SAVE_QUERY, new
String[]{"id"});

```

```

        ps.setInt(1, coinPriceHistory.getCoinPairId());
        ps.setBigDecimal(2, coinPriceHistory.getOpen());
        ps.setBigDecimal(3, coinPriceHistory.getHigh());
        ps.setBigDecimal(4, coinPriceHistory.getLow());
        ps.setBigDecimal(5, coinPriceHistory.getClose());
        ps.setBigDecimal(6, coinPriceHistory.getVolume());
        ps.setLong(7, coinPriceHistory.getNumberOfTrades());
        ps.setLong(8, coinPriceHistory.getCandleTimeOpen());
        ps.setLong(9, coinPriceHistory.getCandleTimeClose());
        ps.setString(10, coinPriceHistory.getCandleType());
        ps.setInt(11, coinPriceHistory.getExchangeId());
        return ps;
    }, keyHolder);
    coinPriceHistory.setId((Long) keyHolder.getKey());
    return coinPriceHistory;
}

```

@Override

```
public List<CoinPriceHistory> saveAll(List<CoinPriceHistory> prices) {
```

```
    jdbcTemplate.batchUpdate(SAVE_QUERY, new BatchPreparedStatementSetter() {
```

@Override

```
    public void setValues(PreparedStatement preparedStatement, int i) throws
SQLException {
```

```
        CoinPriceHistory price = prices.get(i);
        preparedStatement.setInt(1, price.getCoinPairId());
        preparedStatement.setBigDecimal(2, price.getOpen());
        preparedStatement.setBigDecimal(3, price.getHigh());
        preparedStatement.setBigDecimal(4, price.getLow());
        preparedStatement.setBigDecimal(5, price.getClose());
        preparedStatement.setBigDecimal(6, price.getVolume());
        preparedStatement.setLong(7, price.getNumberOfTrades());
        preparedStatement.setLong(8, price.getCandleTimeOpen());
        preparedStatement.setLong(9, price.getCandleTimeClose());
        preparedStatement.setString(10, price.getCandleType());
        preparedStatement.setInt(11, price.getExchangeId());
    }
}

```

@Override

```
public int getBatchSize() {
    return prices.size();
}

```

```
});
```

```
return prices;
```

```
}
```

@Override

```

public void delete(long id) {
    jdbcTemplate.update(DELETE_QUERY, id);
}

@Override
public void deleteAll() {
    jdbcTemplate.update(DELETE_ALL_QUERY);
}

@Override
public List<CoinPriceHistory> findAll() {
    return jdbcTemplate.query(FIND_ALL_QUERY, new
CoinPriceHistoryRowMapper());
}

@Override
public CoinPriceHistory findById(long id) {
    return jdbcTemplate.queryForObject(FIND_BY_ID_QUERY, new
CoinPriceHistoryRowMapper(), id);
}

@Override
public List<CoinPriceHistory> findByCoinPairId(int coinPairId) {
    return jdbcTemplate.query(FIND_BY_COIN_PAIR_ID_QUERY, new
CoinPriceHistoryRowMapper(), coinPairId);
}

@Override
public void update(CoinPriceHistory updatedCoinPriceHistory) {
    jdbcTemplate.update(UPDATE_QUERY,
        updatedCoinPriceHistory.getOpen(),
        updatedCoinPriceHistory.getHigh(),
        updatedCoinPriceHistory.getLow(),
        updatedCoinPriceHistory.getClose(),
        updatedCoinPriceHistory.getVolume(),
        updatedCoinPriceHistory.getNumberOfTrades(),
        updatedCoinPriceHistory.getCandleTimeOpen(),

        updatedCoinPriceHistory.getCoinPairId(),
        updatedCoinPriceHistory.getExchangeId(),
        updatedCoinPriceHistory.getCandleTimeClose(),
        updatedCoinPriceHistory.getCandleType());
}

@Override
public Optional<Long> findLastHistoryLoadedDate(int exchangeId, int coinPairId,
String interval) {

```

```

        return
Optional.ofNullable(jdbcTemplate.queryForObject(FIND_LAST_LOADED_HISTORY_
DATE, Long.class, exchangeId, coinPairId, interval));

    }

    @Override
    public CoinPriceHistory findByExchangeId(int exchangeId) {
        return jdbcTemplate.queryForObject(FIND_BY_EXCHANGE_ID_QUERY, new
CoinPriceHistoryRowMapper(), exchangeId);
    }

    @Override
    public List<String> findAllCandleTypes(int coinPairId, int exchangeId){
        return
jdbcTemplate.queryForList(FIND_ALL_CANDLE_TYPES_IN_DATABASE,
String.class, coinPairId, exchangeId);
    }

    @Override
    public List<CoinPriceHistory> findLastCalculatedValues(int exchangeId, int
coinPairId, String interval, Indicator indicator, int period, long limit) {
        return jdbcTemplate.query(FIND_LAST_CALCULATED_HISTORIES_QUERY,
new CoinPriceHistoryRowMapper(), exchangeId, coinPairId, interval, indicator.name(),
period, limit);
    }

    @Override
    public List<CoinPriceHistory> findUnCalculatedCoinPriceHistories(int exchangeId, int
coinPairId, Interval interval, Indicator indicator, int period, long limit) {
        return
jdbcTemplate.query(FIND_ALL_UN_CALCULATED_COIN_PRICE_HISTORIES_QU
ERY,
        new CoinPriceHistoryRowMapper(), exchangeId, coinPairId, interval.getType(),
indicator.name(), period,
        exchangeId, coinPairId, interval.getType(), limit);
    }

    @Override
    public List<CoinPriceHistory>
findHistoryByExchangeCoinPairIntervalCloseTimeLimit(int exchangeId, int coinPairId,
String interval, long closeTime, long limit) {
        return
jdbcTemplate.query(FIND_BY_EXCHANGE_COIN_PAIR_INTERVAL_QUERY, new
CoinPriceHistoryRowMapper(), exchangeId, coinPairId, interval, closeTime, limit);
    }

    @Override

```

```

    public List<CoinPriceHistory> findPreviousData(int exchangeId, int coinPairId, String
interval, String indicatorType, long closeTime, int period, int limit) {
        return jdbcTemplate.query(FIND_PREVIOUS_DATA, new
CoinPriceHistoryRowMapper(), exchangeId, coinPairId, interval, indicatorType,
closeTime, period, limit);
    }

```

```

    @Override
    public Optional<Long> findLastDate(int exchangeId, int coinPairId, String interval,
String indicatorType, int period) {
        return Optional.ofNullable(jdbcTemplate.queryForObject(FIND_LAST_DATE,
Long.class, exchangeId, coinPairId, interval, indicatorType, period));
    }
}

```

```

package com.dkostin.cryptos.loader.db.dao.impl;

```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

```

```

import com.dkostin.cryptos.loader.db.dao.CoinsByExchangeDao;

```

```

@Repository
public class CoinsByExchangeDaoImpl implements CoinsByExchangeDao {

```

```

    private final JdbcTemplate jdbcTemplate;

```

```

    private final String SAVE_QUERY = "INSERT INTO coins_by_exchange
(coins_pair_id,exchange_id, p2p_supported) VALUES (?,?,:)";
    private final String DELETE_ALL_QUERY = "DELETE FROM coins_by_exchange";
    private final String DELETE_QUERY = "DELETE FROM coins_by_exchange
WHERE coins_pair_id=? AND exchange_id=? AND p2p_supported=?";

```

```

    @Autowired
    public CoinsByExchangeDaoImpl(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

```

```

    @Override
    public int save(int coinPairId, int exchangeId, boolean p2pSupport) {
        return jdbcTemplate.update(SAVE_QUERY, coinPairId, exchangeId, p2pSupport);
    }

```

```

    @Override
    public void deleteById(int coin_pair_id, int exchange_id, boolean p2pSupported) {
        jdbcTemplate.update(DELETE_QUERY, coin_pair_id, exchange_id, p2pSupported);
    }

```

```

@Override
public void deleteAll() {
    jdbcTemplate.update(DELETE_ALL_QUERY);
}
}

```

```
package com.dkostin.cryptos.loader.db.dao.impl;
```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

```

```
import java.util.List;
```

```

import com.dkostin.cryptos.loader.db.dao.ExchangeDao;
import com.dkostin.cryptos.loader.db.rowmapper.ExchangeRowMapper;
import com.dkostin.cryptos.loader.domain.Exchange;

```

```

@Repository
public class ExchangeDaoImpl implements ExchangeDao {

```

```

    private final String SAVE_QUERY = "INSERT INTO exchange (id,exchange_name,
active) VALUES (?,?,?)";
    private final String UPDATE_QUERY = "UPDATE exchange SET exchange_name=?,
active=? WHERE id=?";
    private final String FIND_ALL_QUERY = "SELECT * FROM exchange";
    private final String FIND_BY_ID_QUERY = "SELECT * FROM exchange WHERE
id=?";
    private final String FIND_BY_NAME_QUERY = "SELECT * FROM exchange
WHERE exchange_name=?";
    private final String DELETE_QUERY = "DELETE FROM exchange WHERE id=?";
    private final String DELETE_ALL_QUERY = "DELETE FROM exchange";

```

```
    private final JdbcTemplate jdbcTemplate;
```

```

@Autowired
public ExchangeDaoImpl(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
}

```

```

@Override
public Exchange save(Exchange exchange) {
    int index = findAll().size() + 1;
    jdbcTemplate.update(SAVE_QUERY, index, exchange.getExchangeName(),
exchange.getActive());
    return exchange;
}

```

```

    }

    @Override
    public List<Exchange> findAll() {
        return jdbcTemplate.query(FIND_ALL_QUERY, new ExchangeRowMapper());
    }

    @Override
    public Exchange findById(int id) {
        return jdbcTemplate.queryForObject(FIND_BY_ID_QUERY, new
ExchangeRowMapper(), id);
    }

    @Override
    public void deleteById(int id) {
        jdbcTemplate.update(DELETE_QUERY, id);
    }

    @Override
    public void deleteAll() {
        jdbcTemplate.update(DELETE_ALL_QUERY);
    }

    @Override
    public void update(Exchange updatedExchange) {
        jdbcTemplate.update(UPDATE_QUERY, updatedExchange.getExchangeName(),
updatedExchange.getActive(), updatedExchange.getId());
    }

    @Override
    public Exchange findByName(String exchangeName) {
        return jdbcTemplate.queryForObject(FIND_BY_NAME_QUERY, new
ExchangeRowMapper(), exchangeName);
    }
}

```

```
package com.dkostin.cryptos.loader.db.dao.impl;
```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.BatchPreparedStatementSetter;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.jdbc.support.KeyHolder;
import org.springframework.stereotype.Repository;

```

```

import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.List;

```

```
import java.util.Objects;
import java.util.Optional;
```

```
import com.dkostin.cryptos.loader.db.dao.ExchangeIndicatorDao;
import com.dkostin.cryptos.loader.db.rowmapper.ExchangeIndicatorRowMapper;
import com.dkostin.cryptos.loader.domain.ExchangeIndicator;
import com.dkostin.cryptos.loader.domain.Indicator;
```

```
@Repository
```

```
public class ExchangeIndicatorDaoImpl implements ExchangeIndicatorDao {
```

```
    private static final String SAVE_QUERY = "INSERT INTO exchange_indicator " +
        "(calculated_value, coin_price_history_id, indicator_type, period) " +
        "VALUES (?, ?, ?, ?)";
```

```
    private static final String FIND_ALL_QUERY = "SELECT * FROM
exchange_indicator";
```

```
    private static final String FIND_BY_COIN_PRICE_HISTORY_ID_QUERY =
"SELECT * FROM exchange_indicator WHERE id = ?";
```

```
    private final String FIND_LAST_INDICATOR = "SELECT * " +
        "FROM exchange_indicator " +
        "INNER JOIN coin_price_history ON
coin_price_history.id=exchange_indicator.coin_price_history_id " +
        "WHERE coin_price_history.exchange_id = ? AND
coin_price_history.coin_pair_id = ? AND coin_price_history.candle_type = ? AND
indicator_type = ? AND period = ? " +
        "ORDER BY coin_price_history.candle_time_close DESC " +
        "LIMIT ?" ;
```

```
    private final String FIND_ALL_LAST_INDICATORS = "SELECT * FROM
exchange_indicator " +
        "JOIN \n" +
        "(SELECT MAX(exchange_indicator.id), exchange_id, coin_pair_id from
exchange_indicator " +
        "JOIN coin_price_history ON exchange_indicator.coin_price_history_id =
coin_price_history.id " +
        "WHERE indicator_type = ? " +
        "GROUP BY exchange_id, coin_pair_id, candle_type) AS max_ids ON
exchange_indicator.id = max_ids.max";
```

```
    private static final String DELETE_ALL_QUERY = "DELETE FROM
exchange_indicator";
```

```
    private final JdbcTemplate jdbcTemplate;
```

```
@Autowired
```

```
public ExchangeIndicatorDaoImpl(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
```

```

    }

    @Override
    public ExchangeIndicator save(ExchangeIndicator exchangeIndicator) {
        KeyHolder keyHolder = new GeneratedKeyHolder();
        jdbcTemplate.update(connection -> {
            PreparedStatement preparedStatement =
connection.prepareStatement(SAVE_QUERY,
                Statement.RETURN_GENERATED_KEYS);

            setPreparedStatement(exchangeIndicator, preparedStatement);

            return preparedStatement;
        }, keyHolder);

        exchangeIndicator.setId(Objects.requireNonNull(keyHolder.getKey()).longValue());
        return exchangeIndicator;
    }

    @Override
    public List<ExchangeIndicator> saveAll(List<ExchangeIndicator> exchangeIndicators)
    {
        jdbcTemplate.batchUpdate(SAVE_QUERY, new BatchPreparedStatementSetter() {

            @Override
            public void setValues(PreparedStatement preparedStatement, int i) throws
SQLException {
                ExchangeIndicator exchangeIndicator = exchangeIndicators.get(i);
                setPreparedStatement(exchangeIndicator, preparedStatement);
            }

            @Override
            public int getBatchSize() {
                return exchangeIndicators.size();
            }
        });

        return exchangeIndicators;
    }

    @Override
    public List<ExchangeIndicator> findAll() {
        return jdbcTemplate.query(FIND_ALL_QUERY, new
ExchangeIndicatorRowMapper());
    }

    @Override
    public Optional<ExchangeIndicator> findById(Long id) {

```

```

        return
        Optional.ofNullable(jdbcTemplate.queryForObject(FIND_BY_COIN_PRICE_HISTORY
_ID_QUERY,
                new ExchangeIndicatorRowMapper(), id));
    }

    @Override
    public List<ExchangeIndicator> findLastCalculatedIndicators(Integer exchangeId,
Integer coinPairId, String candle_type, Indicator indicator, int period, int limit) {
        List<ExchangeIndicator> lastCalculatedIndicators = jdbcTemplate.query(
FIND_LAST_INDICATOR, new ExchangeIndicatorRowMapper(), exchangeId,
coinPairId, candle_type, indicator.name(), period, limit );
        return lastCalculatedIndicators;
    }

    @Override
    public List<ExchangeIndicator> findAllLastCalculatedIndicators(Indicator
indicator_type) {
        List<ExchangeIndicator> lastCalculatedIndicators = jdbcTemplate.query(
FIND_ALL_LAST_INDICATORS, new ExchangeIndicatorRowMapper(),
indicator_type.name());
        return lastCalculatedIndicators;
    }

    @Override
    public void deleteAll() {
        jdbcTemplate.update(DELETE_ALL_QUERY);
    }

    private void setPreparedStatement(ExchangeIndicator exchangeIndicator,
PreparedStatement preparedStatement)
        throws SQLException {
        preparedStatement.setBigDecimal(1, exchangeIndicator.getCalculatedValue());
        preparedStatement.setLong(2, exchangeIndicator.getCoinPriceHistoryId());
        preparedStatement.setString(3, exchangeIndicator.getIndicator().toString());
        preparedStatement.setInt(4, exchangeIndicator.getPeriod());
    }
}

package com.dkostin.cryptos.loader.domain;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor

```

```

@NoArgsConstructor
public class CoinPair {

    private Integer Id;

    private String baseCoin;

    private String quoteCoin;

    private Boolean active;
}

package com.dkostin.cryptos.loader.domain;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.math.BigDecimal;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class CoinPriceHistory {
    private Long id;

    private Integer coinPairId;

    private BigDecimal open;

    private BigDecimal high;

    private BigDecimal low;

    private BigDecimal close;

    private BigDecimal volume;

    private Long numberOfTrades;

    private Long candleTimeOpen;

    private Long candleTimeClose;

    private String candleType;

    private Integer exchangeId;
}

```

```
package com.dkostin.cryptos.loader.domain;
```

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class CoinsByExchange {

    private Integer coinsPairId;
    private Integer exchangeId;

}
```

```
package com.dkostin.cryptos.loader.domain;
```

```
import lombok.Getter;
```

```
@Getter
public enum DealType {
    BUY(0), SELL(1);

    private int dbValue;

    DealType(int dbValue) {
        this.dbValue = dbValue;
    }

}
```

```
package com.dkostin.cryptos.loader.domain;
```

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Exchange {

    private Integer id;

    private String exchangeName;

    private Boolean active;
```

```
}
```

```
package com.dkostin.cryptos.loader.domain;
```

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.NoArgsConstructor;
```

```
import java.math.BigDecimal;
```

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class ExchangeIndicator {
    @EqualsAndHashCode.Exclude
    private Long id;
    private BigDecimal calculatedValue;
    private Long coinPriceHistoryId;
    private Indicator indicator;
    private Integer period;
}
```

```
package com.dkostin.cryptos.loader.indicator.config;
```

```
import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
```

```
@Data
@Configuration
@PropertySource("classpath:indicator.properties")
@ConfigurationProperties(prefix = "period")
public class PeriodConfiguration {
    // to do update to Map<Name,Integer>

    private Integer sma;
    private Integer ema;
    private Integer mfi;
    private Integer ao;
    private Integer so;
}
```

```
package com.dkostin.cryptos.loader.indicator.service.impl;
```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;
import java.util.stream.Collectors;

import com.dkostin.cryptos.loader.domain.CoinPriceHistory;
import com.dkostin.cryptos.loader.domain.ExchangeIndicator;
import com.dkostin.cryptos.loader.domain.Indicator;
import com.dkostin.cryptos.loader.domain.Interval;
import com.dkostin.cryptos.loader.indicator.config.PeriodConfiguration;
import com.dkostin.cryptos.loader.indicator.service.IndicatorCalculationService;
import com.dkostin.cryptos.loader.service.CoinPriceHistoryService;

@Service
public class CalculateAroonOscillatorImpl implements IndicatorCalculationService {

    private CoinPriceHistoryService coinPriceHistoryService;
    private PeriodConfiguration periodConfiguration;

    @Autowired
    public CalculateAroonOscillatorImpl(CoinPriceHistoryService
coinPriceHistoryService, PeriodConfiguration periodConfiguration) {
        this.coinPriceHistoryService = coinPriceHistoryService;
        this.periodConfiguration = periodConfiguration;
    }

    @Override
    public List<ExchangeIndicator> doCalculate(int idProvider, int coinPairId, Interval
interval, long numberOfCalculations) {
        long closeTime = 0;
        long numberOfCalculationsIndicator = 0;
        int numberAddedElements = periodConfiguration.getAo() - 1;

        closeTime = coinPriceHistoryService.findLastDate(idProvider, coinPairId,
interval.getType(), Indicator.AO.name(), periodConfiguration.getAo());
        if(closeTime > 0){
            numberOfCalculationsIndicator = numberOfCalculations;
        }
        else{
            numberOfCalculationsIndicator = numberOfCalculations +
numberAddedElements;
        }
    }

```

```

        List<CoinPriceHistory> priceHistoriesPrevious =
coinPriceHistoryService.findPreviousData(idProvider, coinPairId, interval.getType(),
Indicator.AO.name(), closeTime, periodConfiguration.getAo(), numberAddedElements);
        List<CoinPriceHistory> priceHistoriesPreviousSorted =
priceHistoriesPrevious.stream()
                .sorted(Comparator.comparing(CoinPriceHistory::getCandleTimeClose))
                .collect(Collectors.toList());

        List<CoinPriceHistory> priceHistoryListNext =
coinPriceHistoryService.findHistoryByExchangeCoinPairIntervalCloseTimeLimit(idProvi
der,
                coinPairId, interval.getType(), closeTime, numberOfCalculationsIndicator);

        List<CoinPriceHistory> priceHistoryList = new ArrayList<>();
        priceHistoryList.addAll(priceHistoriesPreviousSorted);
        priceHistoryList.addAll(priceHistoryListNext);

        return doCalculateForIndicator(priceHistoryList);
    }

    public List<ExchangeIndicator> doCalculateForIndicator(List<CoinPriceHistory>
priceHistoryList){
        List<CoinPriceHistory> priceHistoriesAroonPeriod ;
        BigDecimal aroonOsc = BigDecimal.ZERO;
        long coinPairHistoryId = 0L;
        int numIndicators = priceHistoryList.size();
        List<ExchangeIndicator> exchangeIndicators = new ArrayList<>();

        for (int i = 0; i < numIndicators; i++) {
            priceHistoriesAroonPeriod = priceHistoryList.stream()
                    .limit(periodConfiguration.getAo())
                    .collect(Collectors.toList());

            if(priceHistoriesAroonPeriod.size() < periodConfiguration.getAo()){
                break;
            }
            coinPairHistoryId = priceHistoriesAroonPeriod.get(periodConfiguration.getAo() -
1).getId();
            aroonOsc = getArronOscillator(priceHistoriesAroonPeriod);
            addIndicator(aroonOsc, exchangeIndicators, coinPairHistoryId);

            int j = 0;
            Iterator<CoinPriceHistory> iteratorCoinPriceHistory = priceHistoryList.iterator();
            while (iteratorCoinPriceHistory.hasNext()) {
                iteratorCoinPriceHistory.next();
                iteratorCoinPriceHistory.remove();
                j++;
                if(j == 1){

```

```

        break;
    }
}

}
return exchangeIndicators;
}

public void addIndicator(BigDecimal aroonOsc, List<ExchangeIndicator>
exchangeIndicators, long coinPairHistoryId){
    ExchangeIndicator exchangeIndicator = new ExchangeIndicator();
    exchangeIndicator.setIndicator(Indicator.AO);
    exchangeIndicator.setCalculatedValue(aroonOsc);
    exchangeIndicator.setCoinPriceHistoryId(coinPairHistoryId);
    exchangeIndicator.setPeriod(periodConfiguration.getAo());
    exchangeIndicators.add(exchangeIndicator);
}

public BigDecimal getMaxPeriodHigh(List<CoinPriceHistory> priceHistories25){
    int i = 0;
    int periodHigh = 0;

    CoinPriceHistory coinPriceHistoryMaxHigh = priceHistories25.stream()
        .max(Comparator.comparing(CoinPriceHistory::getHigh)).get();

    for (CoinPriceHistory coinPriceHistory : priceHistories25) {
        if(coinPriceHistory.equals(coinPriceHistoryMaxHigh)){
            periodHigh = i;
            break;
        }
        i++;
    }

    int periodSince= (priceHistories25.size() - 1) - periodHigh;
    return BigDecimal.valueOf(((periodConfiguration.getAo() - periodSince) * 1.0 /
periodConfiguration.getAo()) * 100);
}

public BigDecimal getMinPeriodLow(List<CoinPriceHistory> priceHistories25){
    int i = 0;
    int periodLow = 0;

    CoinPriceHistory coinPriceHistoryMinLow = priceHistories25.stream()
        .min(Comparator.comparing(CoinPriceHistory::getLow)).get();

    for (CoinPriceHistory coinPriceHistory : priceHistories25) {
        if(coinPriceHistory.equals(coinPriceHistoryMinLow)){
            periodLow = i;

```

```

        break;
    }
    i++;
}

int periodSince = (priceHistories25.size() - 1) - periodLow;
return BigDecimal.valueOf(((periodConfiguration.getAo() - periodSince) * 1.0 /
periodConfiguration.getAo()) * 100);
}

public BigDecimal getArronOscillator(List<CoinPriceHistory> priceHistories25) {

    BigDecimal aroonUp = getMaxPeriodHigh(priceHistories25);
    BigDecimal aroonDown = getMinPeriodLow(priceHistories25);

    return aroonUp.subtract(aroonDown);

}
}

```

```

package com.dkostin.cryptos.loader.indicator.service.impl;

import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.annotation.PostConstruct;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.*;

import com.dkostin.cryptos.loader.domain.CoinPriceHistory;
import com.dkostin.cryptos.loader.domain.ExchangeIndicator;
import com.dkostin.cryptos.loader.domain.Indicator;
import com.dkostin.cryptos.loader.domain.Interval;
import com.dkostin.cryptos.loader.indicator.config.PeriodConfiguration;
import com.dkostin.cryptos.loader.indicator.service.IndicatorCalculationService;
import com.dkostin.cryptos.loader.indicator.service.StatableCalculationService;
import com.dkostin.cryptos.loader.service.CoinPairService;
import com.dkostin.cryptos.loader.service.CoinPriceHistoryService;
import com.dkostin.cryptos.loader.service.ExchangeIndicatorService;
import com.dkostin.cryptos.loader.service.ExchangeService;

@Service
@Slf4j
public class ExponentialMovingAverageIndicator implements
IndicatorCalculationService, StatableCalculationService {

```

```

private final PeriodConfiguration periodConfiguration;

private final ExchangeIndicatorService exchangeIndicatorService;
private final CoinPriceHistoryService coinPriceHistoryService;
private final ExchangeService exchangeService;
private final CoinPairService coinPairService;

private Map<Integer, Map<Integer, Map<String, BigDecimal>>> indicatorState;

@Autowired
public ExponentialMovingAverageIndicator(PeriodConfiguration periodConfiguration,
                                         CoinPriceHistoryService coinPriceHistoryService,
                                         ExchangeService exchangeService, CoinPairService
coinPairService,
                                         ExchangeIndicatorService exchangeIndicatorService) {
    this.periodConfiguration = periodConfiguration;
    this.coinPriceHistoryService = coinPriceHistoryService;
    this.exchangeService = exchangeService;
    this.coinPairService = coinPairService;
    this.exchangeIndicatorService = exchangeIndicatorService;
}

@PostConstruct
private void init() {
    initState();
}

@Override
public void initState() {
    indicatorState = new HashMap<>();
    loadState();
}

@Override
public List<ExchangeIndicator> doCalculate(int providerId, int coinPairId, Interval
interval, long limit) {
    List<CoinPriceHistory> coinPriceHistoriesForCalculations =
coinPriceHistoryService.findUnCalculatedCoinPriceHistories( providerId, coinPairId,
interval, Indicator.EMA, periodConfiguration.getEma(), limit);
    return calculateInList( coinPriceHistoriesForCalculations );
}

private List<ExchangeIndicator> calculateInList(List<CoinPriceHistory>
inputCoinsPriceHistories) {
    List<ExchangeIndicator> exchangeIndicators = new ArrayList<>();

    final int period = periodConfiguration.getEma();
    BigDecimal smoothingFactor = new BigDecimal( 2.0 / (1.0 + period) );

```

```

        for (int i = 0; i < inputCoinsPriceHistories.size(); i++) {
            CoinPriceHistory history = inputCoinsPriceHistories.get( i );
            BigDecimal lastCalculatedIndicatorValue = getState( history.getExchangeId(),
            history.getCoinPairId(), history.getCandleType() );
            ExchangeIndicator currentIndicator;

            if (lastCalculatedIndicatorValue.equals( new BigDecimal( "0" ) )) {
                currentIndicator = new ExchangeIndicator();
                currentIndicator.setCoinPriceHistoryId(inputCoinsPriceHistories.get( i
            ).getId());
                currentIndicator.setIndicator(Indicator.EMA);
                currentIndicator.setPeriod(period);
                currentIndicator.setCalculatedValue( inputCoinsPriceHistories.get( i
            ).getClose().setScale( 8, RoundingMode.HALF_UP ) );
            } else {
                currentIndicator = calculateIndicator( lastCalculatedIndicatorValue, history,
            smoothingFactor, period );
            }
            setState( history.getExchangeId(), history.getCoinPairId(),
            history.getCandleType(), currentIndicator.getCalculatedValue() );
            exchangeIndicators.add( currentIndicator );
        }
        return exchangeIndicators;
    }

    private ExchangeIndicator calculateIndicator(BigDecimal lastCalculatedIndicatorValue,
    CoinPriceHistory currentCoinPriceHistory, BigDecimal smoothingFactor, int period) {

        ExchangeIndicator indicator = new ExchangeIndicator();
        indicator.setCoinPriceHistoryId(currentCoinPriceHistory.getId());
        indicator.setIndicator(Indicator.EMA);
        indicator.setPeriod(period);

        double calValue = (currentCoinPriceHistory.getClose().doubleValue() *
        smoothingFactor.doubleValue()) + (lastCalculatedIndicatorValue.doubleValue()*(1-
        smoothingFactor.doubleValue()));

        indicator.setCalculatedValue( new BigDecimal( calValue ).setScale( 8,
        RoundingMode.HALF_UP ) );

        return indicator;
    }

    private BigDecimal getState(int providerId, int coinPairId, String interval) {
        if (indicatorState.containsKey(providerId)) {
            Map<Integer, Map<String, BigDecimal>> exchangeCoinPairMap =
            indicatorState.get(providerId);

            if (exchangeCoinPairMap.containsKey(coinPairId)) {

```

```

        Map<String, BigDecimal> candleTypeValueMap =
exchangeCoinPairMap.get(coinPairId);

        if (candleTypeValueMap.containsKey(interval)) {
            return candleTypeValueMap.get(interval);
        }
    }
}

return new BigDecimal( "0" );
}

private void setState(int providerId, int coinPairId, String interval, BigDecimal
newValue) {
    if (indicatorState.containsKey( providerId )) {
        Map<Integer, Map<String, BigDecimal>> exchangeCoinPairMap =
indicatorState.get( providerId );
        if(exchangeCoinPairMap.containsKey( coinPairId )) {
            Map<String, BigDecimal> candleTypeValueMap = exchangeCoinPairMap.get(
coinPairId );
            candleTypeValueMap.put( interval, newValue );
        }
        else {
            Map<String, BigDecimal> candleTypeValueMap = new HashMap<>();
            candleTypeValueMap.put( interval, newValue );
            exchangeCoinPairMap.put( coinPairId, candleTypeValueMap );
        }
    }
    else {
        Map<Integer, Map<String, BigDecimal>> exchangeCoinPairMap = new
HashMap<>();
        Map<String, BigDecimal> candleTypeValueMap = new HashMap<>();
        candleTypeValueMap.put( interval, newValue );
        exchangeCoinPairMap.put( coinPairId, candleTypeValueMap );
        indicatorState.put( providerId, exchangeCoinPairMap );
    }
}

private void loadState() {
    exchangeService.findAll().forEach(
        exchange -> {
            indicatorState.put(exchange.getId(), new HashMap<>());

            coinPairService.getActiveCoinPairsByExchange(exchange.getId(),
false).forEach(
                coinPair -> {
                    indicatorState.get(exchange.getId()).put(coinPair.getId(), new
HashMap<>());

```

```

        coinPriceHistoryService.findAllCandleTypes(coinPair.getId(),
exchange.getId()).forEach(
    candleType -> {
        log.info( "EMA indicator is loading start state now. Exchange
{}, CoinPair {}, CandleType {}", exchange.getId(), coinPair.getId(), candleType);
        List<ExchangeIndicator> lastCalculatedIndicators =
exchangeIndicatorService.findLastCalculatedIndicators(
            exchange.getId(),
            coinPair.getId(),
            candleType,
            Indicator.EMA,
            periodConfiguration.getEma(),
            1 );

        if (lastCalculatedIndicators.isEmpty()) {
            indicatorState.get( exchange.getId() ).get( coinPair.getId()
).put( candleType, new BigDecimal( "0" ));
        } else {
            indicatorState.get( exchange.getId() ).get( coinPair.getId()
).put( candleType, lastCalculatedIndicators.get( 0 ).getCalculatedValue());
        }
    });
}
);
}
);
}
}

```

```
package com.dkostin.cryptos.loader.indicator.service.impl;
```

```
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import javax.annotation.PostConstruct;
import java.math.BigDecimal;
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;
```

```
import com.dkostin.cryptos.loader.domain.CoinPriceHistory;
import com.dkostin.cryptos.loader.domain.ExchangeIndicator;
import com.dkostin.cryptos.loader.domain.Indicator;
import com.dkostin.cryptos.loader.domain.Interval;
import com.dkostin.cryptos.loader.indicator.config.PeriodConfiguration;
```

```

import com.dkostin.cryptos.loader.indicator.service.IndicatorCalculationService;
import com.dkostin.cryptos.loader.indicator.service.StatableCalculationService;
import com.dkostin.cryptos.loader.service.CoinPairService;
import com.dkostin.cryptos.loader.service.CoinPriceHistoryService;
import com.dkostin.cryptos.loader.service.ExchangeService;

@Service
@Slf4j
public class MoneyFlowIndexCalculationService implements
    IndicatorCalculationService, StatableCalculationService {

    private Map<Integer, Map<Integer, Map<String, List<CoinPriceHistory>>>>
indicatorState;

    private final PeriodConfiguration periodConfiguration;

    private final CoinPriceHistoryService coinPriceHistoryService;
    private final ExchangeService exchangeService;
    private final CoinPairService coinPairService;

    @Autowired
    public MoneyFlowIndexCalculationService(PeriodConfiguration periodConfiguration,
        CoinPriceHistoryService coinPriceHistoryService,
        ExchangeService exchangeService, CoinPairService
coinPairService) {
        this.periodConfiguration = periodConfiguration;
        this.coinPriceHistoryService = coinPriceHistoryService;
        this.exchangeService = exchangeService;
        this.coinPairService = coinPairService;
    }

    @PostConstruct
    private void init() {
        initState();
    }

    @Override
    public List<ExchangeIndicator> doCalculate(int providerId, int coinPairId,
        Interval interval, long numberOfCalculations) {
        log.info("Calculating mfi");
        List<ExchangeIndicator> results = new ArrayList<>();

        LinkedList<CoinPriceHistory> state = new LinkedList<>(getState(providerId,
coinPairId, interval));

        List<CoinPriceHistory> coinPriceHistoriesToCalculate = coinPriceHistoryService
            .findUnCalculatedCoinPriceHistories(providerId, coinPairId, interval,
                Indicator.MFI, periodConfiguration.getMfi(), numberOfCalculations);

```

```

        coinPriceHistoriesToCalculate.forEach(
            coinPriceHistory -> {
                if (state.size() == periodConfiguration.getMfi()) {
                    results.add(calculateOnePeriod(
                        Stream.concat(state.stream(),
Stream.of(coinPriceHistory)).collect(Collectors.toList())
                    ));
                    state.removeFirst();
                }
                state.addLast(coinPriceHistory);
            }
        );

        indicatorState.putIfAbsent(providerId, new HashMap<>());
        indicatorState.get(providerId).putIfAbsent(coinPairId, new HashMap<>());
        indicatorState.get(providerId).get(coinPairId).put(interval.getType(), state);

        return results;
    }

    @Override
    public void initState() {
        indicatorState = new HashMap<>();
        loadState();
    }

    private List<CoinPriceHistory> getState(int providerId, int coinPairId, Interval interval)
    {
        if (indicatorState.containsKey(providerId)) {
            Map<Integer, Map<String, List<CoinPriceHistory>>> exchangeCoinPairMap =
indicatorState.get(providerId);

            if (exchangeCoinPairMap.containsKey(coinPairId)) {
                Map<String, List<CoinPriceHistory>> candleTypeValueMap =
exchangeCoinPairMap.get(coinPairId);

                if (candleTypeValueMap.containsKey(interval.getType())) {
                    return candleTypeValueMap.get(interval.getType());
                }
            }
        }

        return new ArrayList<>();
    }

    private ExchangeIndicator calculateOnePeriod(List<CoinPriceHistory> calculationList)
    {

```

```

BigDecimal positiveFlow = new BigDecimal("0");
BigDecimal negativeFlow = new BigDecimal("0");

BigDecimal lastFlow = null;

for (CoinPriceHistory coinPriceHistory : calculationList) {
    BigDecimal currentFlow = calculateRawMoneyFlow(coinPriceHistory);

    if (lastFlow != null) {
        if (currentFlow.compareTo(lastFlow) < 0) {
            negativeFlow = negativeFlow.add(currentFlow);
        } else if (currentFlow.compareTo(lastFlow) > 0) {
            positiveFlow = positiveFlow.add(currentFlow);
        }
    }

    lastFlow = currentFlow;
}

ExchangeIndicator exchangeIndicator = new ExchangeIndicator();
exchangeIndicator.setIndicator(Indicator.MFI);
exchangeIndicator.setCoinPriceHistoryId(calculationList.get(calculationList.size() -
1).getId());
exchangeIndicator.setPeriod(periodConfiguration.getMfi());
exchangeIndicator.setCalculatedValue(negativeFlow.compareTo(BigDecimal.ZERO)
!= 0 ?
    calculateOnePeriodMoneyFlowIndex(positiveFlow, negativeFlow) :
BigDecimal.ZERO);

return exchangeIndicator;
}

private BigDecimal calculateRawMoneyFlow(CoinPriceHistory coinPriceHistory) {
    return coinPriceHistory.getVolume().multiply(

coinPriceHistory.getHigh().add(coinPriceHistory.getLow()).add(coinPriceHistory.getClos
e())
        .divide(BigDecimal.valueOf(3), 4, BigDecimal.ROUND_CEILING)
    );
}

private BigDecimal calculateOnePeriodMoneyFlowIndex(BigDecimal positiveFlow,
BigDecimal negativeFlow) {
    BigDecimal oneHundred = new BigDecimal("100");
    return oneHundred.subtract(
        oneHundred.divide(
            BigDecimal.ONE.add(

```

```

        positiveFlow.divide(negativeFlow, 4,
BigDecimal.ROUND_CEILING)
        ), 4, BigDecimal.ROUND_CEILING
    )
    );
}

private void loadState() {
    exchangeService.findAll().forEach(
        exchange -> {
            indicatorState.put(exchange.getId(), new HashMap<>());

            coinPairService.getActiveCoinPairsByExchange(exchange.getId(),
false).forEach(
                coinPair -> {
                    indicatorState.get(exchange.getId()).put(coinPair.getId(), new
HashMap<>());

                    coinPriceHistoryService
                        .findAllCandleTypes(coinPair.getId(), exchange.getId()).forEach(
                            candleType -> {
                                List<CoinPriceHistory> state =
                                    coinPriceHistoryService.
                                        findLastCalculatedValues(exchange.getId(),
                                            coinPair.getId(), candleType,
                                            Indicator.MFI,
                                            periodConfiguration.getMfi(),
                                            periodConfiguration.getMfi());

                                Collections.reverse(state);
                                indicatorState.get(exchange.getId()).get(coinPair.getId()).put(
                                    candleType, state);
                                log.info("MFI state loading, exchange: " +
                                    "{}", coin pair: {}-{}, candle type {}",
                                    exchange.getExchangeName(), coinPair.getBaseCoin(),
                                    coinPair.getQuoteCoin(), candleType);
                            }
                        );
                }
            );
        }
    );
}

package com.dkostin.cryptos.loader.indicator.service.impl;

import lombok.extern.slf4j.Slf4j;

```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.annotation.PostConstruct;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.*;

import com.dkostin.cryptos.loader.domain.CoinPriceHistory;
import com.dkostin.cryptos.loader.domain.ExchangeIndicator;
import com.dkostin.cryptos.loader.domain.Indicator;
import com.dkostin.cryptos.loader.domain.Interval;
import com.dkostin.cryptos.loader.indicator.service.IndicatorCalculationService;
import com.dkostin.cryptos.loader.indicator.service.StatableCalculationService;
import com.dkostin.cryptos.loader.service.CoinPairService;
import com.dkostin.cryptos.loader.service.CoinPriceHistoryService;
import com.dkostin.cryptos.loader.service.ExchangeIndicatorService;
import com.dkostin.cryptos.loader.service.ExchangeService;

@Service
@Slf4j
public class OnBalanceVolumeIndicatorCalculationService implements
IndicatorCalculationService, StatableCalculationService {

    private Map<Integer, Map<Integer, Map<String, BigDecimal>>> indicatorState;
    private Map<Integer, Map<Integer, Map<String, CoinPriceHistory>>>
coinPriceHistoryState;

    private final CoinPriceHistoryService coinPriceHistoryService;
    private final ExchangeService exchangeService;
    private final CoinPairService coinPairService;
    private final ExchangeIndicatorService exchangeIndicatorService;

    @Autowired
    public OnBalanceVolumeIndicatorCalculationService (CoinPriceHistoryService
coinPriceHistoryService, ExchangeService exchangeService, CoinPairService
coinPairService, ExchangeIndicatorService exchangeIndicatorService) {
        this.coinPriceHistoryService = coinPriceHistoryService;
        this.exchangeService = exchangeService;
        this.coinPairService = coinPairService;
        this.exchangeIndicatorService = exchangeIndicatorService;
    }

    @PostConstruct
    private void init() {
        initState();
    }

    @Override

```

```

public void initState() {
    indicatorState = new HashMap<>();
    coinPriceHistoryState = new HashMap<>();
    loadState();
}

@Override
public List<ExchangeIndicator> doCalculate(int providerId, int coinPairId, Interval
interval, long numberOfCalculations) {
    BigDecimal stateIndicatorValue = getStateFromIndicators(providerId, coinPairId,
interval);
    CoinPriceHistory stateCoinPriceHistory = getStateFromCoinPriceHistory(
providerId, coinPairId, interval );

    List<CoinPriceHistory> coinPriceHistoriesForCalculations =
coinPriceHistoryService.findUnCalculatedCoinPriceHistories( providerId, coinPairId,
interval, Indicator.OBV, 1, numberOfCalculations);

    return calculateInList( coinPriceHistoriesForCalculations, stateIndicatorValue,
stateCoinPriceHistory );
}

private List<ExchangeIndicator> calculateInList(List<CoinPriceHistory>
inputCoinsPriceHistories, BigDecimal inputStateIndicatorValue, CoinPriceHistory
inputStateCoinPriceHistory) {
    List<ExchangeIndicator> results = new LinkedList<>();
    BigDecimal stateIndicatorValue = inputStateIndicatorValue;
    CoinPriceHistory stateCoinPriceHistory = inputStateCoinPriceHistory;

    for (CoinPriceHistory history : inputCoinsPriceHistories) {
        ExchangeIndicator calculatedIndicator = calculateOneIndicator( history,
stateIndicatorValue, stateCoinPriceHistory );
        stateIndicatorValue = calculatedIndicator.getCalculatedValue();
        stateCoinPriceHistory = history;

        setStateToIndicators( history.getExchangeId(), history.getCoinPairId(),
history.getCandleType(), stateIndicatorValue );
        setStateToCoinPriceHistory( history.getExchangeId(), history.getCoinPairId(),
history.getCandleType(), history );

        results.add( calculatedIndicator );
    }

    return results;
}

private ExchangeIndicator calculateOneIndicator(CoinPriceHistory
currentCoinPriceHistory, BigDecimal lastIndicatorValue, CoinPriceHistory
stateCoinPriceHistory) {

```

```

ExchangeIndicator currentIndicator = new ExchangeIndicator();
currentIndicator.setIndicator( Indicator.OBV );
currentIndicator.setCoinPriceHistoryId( currentCoinPriceHistory.getId() );
currentIndicator.setPeriod( 1 );

if(stateCoinPriceHistory == null) {
    currentIndicator.setCalculatedValue( currentCoinPriceHistory.getVolume()
        .setScale( 8, RoundingMode.HALF_UP ) );
}
else {
    if(currentCoinPriceHistory.getClose().doubleValue() >
stateCoinPriceHistory.getClose().doubleValue()) {
        currentIndicator.setCalculatedValue( lastIndicatorValue.add(
currentCoinPriceHistory.getVolume())
            .setScale( 8, RoundingMode.HALF_UP ) );
    }
    else if(currentCoinPriceHistory.getClose().doubleValue() <
stateCoinPriceHistory.getClose().doubleValue()) {
        currentIndicator.setCalculatedValue( lastIndicatorValue.subtract(
currentCoinPriceHistory.getVolume())
            .setScale( 8, RoundingMode.HALF_UP ) );
    }
    else {
        currentIndicator.setCalculatedValue( lastIndicatorValue
            .setScale( 8, RoundingMode.HALF_UP ) );
    }
}
return currentIndicator;
}

private BigDecimal getStateFromIndicators(int providerId, int coinPairId, Interval
interval) {
    return Optional.ofNullable(indicatorState.get(providerId)).map( e ->
e.get(coinPairId)).map( e -> e.get(interval.getType())).orElse(BigDecimal.ZERO);
}

private void setStateToIndicators(int providerId, int coinPairId, String interval,
BigDecimal newValue) {
    if (indicatorState.containsKey( providerId )) {
        Map<Integer, Map<String, BigDecimal>> exchangeCoinPairMap =
indicatorState.get( providerId );
        if(exchangeCoinPairMap.containsKey( coinPairId )) {
            Map<String, BigDecimal> candleTypeValueMap = exchangeCoinPairMap.get(
coinPairId );
            candleTypeValueMap.put( interval, newValue );
        }
        else {
            Map<String, BigDecimal> candleTypeValueMap = new HashMap<>();
            candleTypeValueMap.put( interval, newValue );
        }
    }
}

```

```

        exchangeCoinPairMap.put( coinPairId, candleTypeValueMap );
    }
}
else {
    Map<Integer, Map<String, BigDecimal>> exchangeCoinPairMap = new
HashMap<>();
    Map<String, BigDecimal> candleTypeValueMap = new HashMap<>();
    candleTypeValueMap.put( interval, newValue );
    exchangeCoinPairMap.put( coinPairId, candleTypeValueMap );
    indicatorState.put( providerId, exchangeCoinPairMap );
}

}

private void loadState() {
    exchangeService.findAll().forEach(
        exchange -> {
            coinPriceHistoryState.put(exchange.getId(), new HashMap<>());
            indicatorState.put(exchange.getId(), new HashMap<>());

            coinPairService.getActiveCoinPairsByExchange(exchange.getId(),
false).forEach(
                coinPair -> {
                    coinPriceHistoryState.get(exchange.getId()).put(coinPair.getId(), new
HashMap<>());
                    indicatorState.get(exchange.getId()).put(coinPair.getId(), new
HashMap<>());

                    coinPriceHistoryService.findAllCandleTypes(coinPair.getId(),
exchange.getId()).forEach(
                        candleType -> {
                            loadLastCalculatedIndicators(exchange.getId(),
coinPair.getId(), candleType);

                            loadLastCoinPriceHistory(exchange.getId(), coinPair.getId(),
candleType);

                        });
                });
        });
}

private void loadLastCalculatedIndicators(int exchangeId, int coinPairId, String
candleType) {
    log.info( "OBV indicator. CoinPriceHistory is loading start state now. Exchange {},
CoinPair {}, CandleType {}", exchangeId, coinPairId, candleType);

    List<CoinPriceHistory> lastCalculatedValues =
coinPriceHistoryService.findLastCalculatedValues( exchangeId,

```

```

        coinPairId, candleType,
        Indicator.OBV,
        1,
        1 );

    if(lastCalculatedValues.isEmpty()) {
        coinPriceHistoryState.get(exchangeId).get(coinPairId).put(candleType, null);
    } else {
        coinPriceHistoryState.get(exchangeId).get(coinPairId).put(candleType,
lastCalculatedValues.get( 0 ));
    }
}

private void loadLastCoinPriceHistory(int exchangeId, int coinPairId, String
candleType) {
    log.info( "OBV indicator is loading start state now. Exchange {}, CoinPair {},
CandleType {}", exchangeId, coinPairId, candleType);
    List<ExchangeIndicator> lastCalculatedIndicators =
exchangeIndicatorService.findLastCalculatedIndicators(
        exchangeId,
        coinPairId,
        candleType,
        Indicator.OBV,
        1,
        1 );

    if (lastCalculatedIndicators.isEmpty()) {
        indicatorState.get( exchangeId ).get( coinPairId ).put( candleType, new
BigDecimal( "0" ));
    } else {
        indicatorState.get( exchangeId ).get( coinPairId ).put( candleType,
lastCalculatedIndicators.get( 0 ).getCalculatedValue());
    }
}

private void setStateToCoinPriceHistory(int providerId, int coinPairId, String interval,
CoinPriceHistory newValue) {
    if (coinPriceHistoryState.containsKey( providerId )) {
        Map<Integer, Map<String, CoinPriceHistory>> exchangeCoinPairMap =
coinPriceHistoryState.get( providerId );
        if(exchangeCoinPairMap.containsKey( coinPairId )) {
            Map<String, CoinPriceHistory> candleTypeValueMap =
exchangeCoinPairMap.get( coinPairId );
            candleTypeValueMap.put( interval, newValue );
        }
    } else {
        Map<String, CoinPriceHistory> candleTypeValueMap = new HashMap<>();
        candleTypeValueMap.put( interval, newValue );
        exchangeCoinPairMap.put( coinPairId, candleTypeValueMap );
    }
}

```

```

        }
    }
    else {
        Map<Integer, Map<String, CoinPriceHistory>> exchangeCoinPairMap = new
HashMap<>();
        Map<String, CoinPriceHistory> candleTypeValueMap = new HashMap<>();
        candleTypeValueMap.put( interval, newValue );
        exchangeCoinPairMap.put( coinPairId, candleTypeValueMap );
        coinPriceHistoryState.put( providerId, exchangeCoinPairMap );
    }

}

private CoinPriceHistory getStateFromCoinPriceHistory(int providerId, int coinPairId,
Interval interval) {
    return Optional.ofNullable(coinPriceHistoryState.get(providerId)).map( e ->
e.get(coinPairId)).map( e -> e.get(interval.getType())).orElse(null);
}

}

package com.dkostin.cryptos.loader.indicator.service.impl;

import com.dkostin.cryptos.loader.domain.CoinPriceHistory;
import com.dkostin.cryptos.loader.domain.ExchangeIndicator;
import com.dkostin.cryptos.loader.domain.Indicator;
import com.dkostin.cryptos.loader.domain.Interval;
import com.dkostin.cryptos.loader.indicator.config.PeriodConfiguration;
import com.dkostin.cryptos.loader.indicator.service.IndicatorCalculationService;
import com.dkostin.cryptos.loader.indicator.service.StatableCalculationService;
import com.dkostin.cryptos.loader.service.CoinPairService;
import com.dkostin.cryptos.loader.service.CoinPriceHistoryService;
import com.dkostin.cryptos.loader.service.ExchangeIndicatorService;
import com.dkostin.cryptos.loader.service.ExchangeService;

import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.annotation.PostConstruct;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;

@Service
@Slf4j

```

```

public class StochasticOscillatorCalculation implements IndicatorCalculationService,
StatableCalculationService {

    private Map<Integer, Map<Integer, Map<String, List<CoinPriceHistory>>>>
indicatorState;

    private final PeriodConfiguration configuration;

    private final ExchangeIndicatorService exchangeIndicatorService;
    private final CoinPriceHistoryService coinPriceHistoryService;
    private final ExchangeService exchangeService;
    private final CoinPairService coinPairService;

    @Autowired
    public StochasticOscillatorCalculation(PeriodConfiguration configuration,
                                           CoinPriceHistoryService coinPriceHistoryService,
                                           ExchangeService exchangeService, CoinPairService
coinPairService,
                                           ExchangeIndicatorService exchangeIndicatorService) {
        this.configuration = configuration;
        this.coinPriceHistoryService = coinPriceHistoryService;
        this.exchangeService = exchangeService;
        this.coinPairService = coinPairService;
        this.exchangeIndicatorService = exchangeIndicatorService;
    }

    @PostConstruct
    private void init() {
        initState();
    }

    @Override
    public void initState() {
        indicatorState = new HashMap<>();
        loadState();
    }

    @Override
    public List<ExchangeIndicator> doCalculate(int idProvider, int coinPairId, Interval
interval, long limit) {
        List<CoinPriceHistory> coinPriceHistoriesForCalculations =
coinPriceHistoryService.findUnCalculatedCoinPriceHistories( idProvider, coinPairId,
interval, Indicator.SO, configuration.getSo(), limit );
        return doCalculateToList(coinPriceHistoriesForCalculations, idProvider, coinPairId,
interval);
    }

```

```

    private List<ExchangeIndicator> doCalculateToList(List<CoinPriceHistory>
inputCoinsPriceHistories, int idProvider, int coinPairId, Interval interval) {
        List<ExchangeIndicator> exchangeIndicatorsFroReturn = new ArrayList<>();

        LinkedList<CoinPriceHistory> state = new LinkedList<>((getState(idProvider,
coinPairId, interval)));

        inputCoinsPriceHistories.forEach(
            coinPriceHistory -> {
                if ((state.size()+1) == configuration.getSo()) {

exchangeIndicatorsFroReturn.add(calculateIndicator(Stream.concat(state.stream(),
Stream.of(coinPriceHistory)).collect(Collectors.toList())));
                    state.removeFirst();
                }
                state.addLast(coinPriceHistory);
            }
        );
        return exchangeIndicatorsFroReturn;
    }

    private ExchangeIndicator calculateIndicator(List<CoinPriceHistory>
coinPriceHistories) {
        int size = coinPriceHistories.size();

        ExchangeIndicator indicator = new ExchangeIndicator();
        indicator.setCoinPriceHistoryId(coinPriceHistories.get(size-1).getId());
        indicator.setIndicator(Indicator.SO);
        indicator.setPeriod(size);

        BigDecimal recent =
coinPriceHistories.stream().max(Comparator.comparing(CoinPriceHistory::getCandleTim
eClose)).get().getClose();

        BigDecimal max =
coinPriceHistories.stream().max(Comparator.comparing(CoinPriceHistory::getHigh)).get(
).getHigh();
        BigDecimal min =
coinPriceHistories.stream().min(Comparator.comparing(CoinPriceHistory::getLow)).get().
getLow();

        BigDecimal so = recent.subtract(min);

        BigDecimal fo = max.subtract(min);

        if(fo.compareTo(new BigDecimal("0.0001")) < 0 ){
            indicator.setCalculatedValue(new BigDecimal("1000000"));
            return indicator;
        }
    }

```



```

        Collections.reverse(state);
        indicatorState.get(exchange.getId()).get(coinPair.getId()).put(
            candleType, state);
        log.info("SO state is loading now, exchange: {}, coin pair: {}-
        {}, candle type {}",
            exchange.getExchangeName(), coinPair.getBaseCoin(),
            coinPair.getQuoteCoin(), candleType);
    }
    );
}
);
}
);
}
}
}

```

```
package com.dkostin.cryptos.loader.indicator;
```

```
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import java.util.List;
```

```
import com.dkostin.cryptos.loader.domain.CoinPair;
import com.dkostin.cryptos.loader.domain.ExchangeIndicator;
import com.dkostin.cryptos.loader.domain.Interval;
import com.dkostin.cryptos.loader.indicator.service.IndicatorCalculationService;
import com.dkostin.cryptos.loader.service.CoinPairService;
import com.dkostin.cryptos.loader.service.CoinPriceHistoryService;
import com.dkostin.cryptos.loader.service.ExchangeIndicatorService;
```

```
@Service
```

```
@Slf4j
```

```
public class IndicatorCalculationLauncher {
```

```
    private static final int DEFAULT_NUMBER_OF_CALCULATIONS = 1000;
```

```
    private final List<IndicatorCalculationService> indicatorCalculationServiceList;
```

```
    private final CoinPriceHistoryService coinPriceHistoryService;
```

```
    private final ExchangeIndicatorService exchangeIndicatorService;
```

```
    private final CoinPairService coinPairService;
```

```
    @Autowired
```

```
    public IndicatorCalculationLauncher(List<IndicatorCalculationService>
```

```
indicatorCalculationServiceList,
```

```
        CoinPriceHistoryService coinPriceHistoryService,
```

```
        ExchangeIndicatorService exchangeIndicatorService,
```

```

        CoinPairService coinPairService) {
    this.indicatorCalculationServiceList = indicatorCalculationServiceList;
    this.coinPriceHistoryService = coinPriceHistoryService;
    this.exchangeIndicatorService = exchangeIndicatorService;
    this.coinPairService = coinPairService;
}

public void startCalculations(int idProvider) {
    List<CoinPair> exchangeCoinPairs =
        coinPairService.getActiveCoinPairsByExchange(idProvider, false);

    exchangeCoinPairs.forEach(
        coinPair -> {
            log.info("Calculating indicators of {} - {}",
                coinPair.getBaseCoin(), coinPair.getQuoteCoin());

            coinPriceHistoryService.findAllCandleTypes(coinPair.getId(),
idProvider).forEach(
                candleType -> {
                    log.info("Calculating indicators for interval: {}", candleType);

                    indicatorCalculationServiceList.forEach(
                        calculator ->
                            calculateAndSaveIndicators(idProvider, coinPair.getId(),
                                candleType, calculator));
                });
        });
}

private void calculateAndSaveIndicators(int idProvider, int coinPairId,
    String candleType, IndicatorCalculationService calculator) {
    int size;

    do { //Saves batches of indicators by
        DEFAULT_NUMBER_OF_CALCULATIONS (1000 elements for ex.)

        List<ExchangeIndicator> exchangeIndicators =
            calculator.doCalculate(idProvider, coinPairId,
                Interval.checkIntervals(candleType),
        DEFAULT_NUMBER_OF_CALCULATIONS);
        size = exchangeIndicators.size();

        exchangeIndicatorService.saveAll(exchangeIndicators);

    } while (size > 1);
}
}

```

```

package com.dkostin.cryptos.loader.provider.binance.config;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

@Builder
@Data
@NoArgsConstructor
@AllArgsConstructor
@Configuration
@PropertySource("classpath:binance.properties")
@ConfigurationProperties
public class BinanceConnectionConfiguration {
    private String candleUrlPrefix;
    private String candleUrl;
    private Integer candleLimit;
    private String candleInterval;

    private Integer binanceLimit;
    private Integer secondsToSleepBeforeNextRequest;

    private String p2pUrlPrefix;
    private String p2pUrl;
}

package com.dkostin.cryptos.loader.provider.binance.config;

import lombok.extern.slf4j.Slf4j;
import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpStatus;
import org.springframework.http.client.ClientHttpResponse;
import org.springframework.web.client.ResponseErrorHandler;
import org.springframework.web.client.RestTemplate;

import java.io.IOException;
import java.time.Duration;

@Configuration

```

```

@Slf4j
public class BinanceProviderConfig {
    @Bean
    public RestTemplate binanceRestTemplate(RestTemplateBuilder builder) {

        return builder
            .setConnectTimeout(Duration.ofMillis(3000))
            .setReadTimeout(Duration.ofMillis(3000))
            .errorHandler(new ResponseErrorHandler() {
                @Override
                public boolean hasError(ClientHttpResponse clientHttpResponse) throws
IOException {
                    return !clientHttpResponse.getStatusCode().equals(HttpStatus.OK);
                }

                @Override
                public void handleError(ClientHttpResponse clientHttpResponse) throws
IOException {
                    log.error("Error occurred while attempt to connect to Binance {}",
clientHttpResponse.getStatusCode());
                }
            })
            .build();
    }
}

package com.dkostin.cryptos.loader.provider.binance.config;

public class CandleQueryParams {
    public static final String candleUrlQuerySymbol = "symbol";
    public static final String candleUrlQueryInterval = "interval";
    public static final String candleUrlQueryStartTime = "startTime";
    public static final String candleUrlQueryEndTime = "endTime";
    public static final String candleUrlQueryLimit = "limit";
}

package com.dkostin.cryptos.loader.provider.binance.connector.impl;

import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;
import org.springframework.web.util.UriComponentsBuilder;

import java.net.URI;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

```

```

import com.dkostin.cryptos.loader.domain.CoinPair;
import com.dkostin.cryptos.loader.domain.DealType;
import com.dkostin.cryptos.loader.domain.Interval;
import
com.dkostin.cryptos.loader.provider.binance.config.BinanceConnectionConfiguration;
import com.dkostin.cryptos.loader.provider.binance.config.CandleQueryParams;
import com.dkostin.cryptos.loader.provider.binance.connector.BinanceConnector;
import com.dkostin.cryptos.loader.provider.binance.convert.CandleConverter;
import com.dkostin.cryptos.loader.provider.binance.dto.BinanceDatumDto;
import com.dkostin.cryptos.loader.provider.binance.dto.BinanceP2PRequestBody;
import com.dkostin.cryptos.loader.provider.binance.dto.BinanceP2PResponseDto;
import com.dkostin.cryptos.loader.provider.binance.dto.CandleItem;

@Slf4j
@Service
public class BinanceConnectorImpl implements BinanceConnector {

    private final RestTemplate restTemplate;
    private final BinanceConnectionConfiguration binanceConnectionConfiguration;

    @Autowired
    public BinanceConnectorImpl(RestTemplate rt, BinanceConnectionConfiguration cf) {
        this.restTemplate = rt;
        this.binanceConnectionConfiguration = cf;
    }

    @Override
    public List<CandleItem> requestBarData(long begin, CoinPair symbol) {

        URI url = buildUrl(begin, symbol,
binanceConnectionConfiguration.getCandleLimit());

        log.info("Url: {}", url);
        List response = restTemplate.getForObject(url, List.class);

        //noinspection unchecked
        return (List<CandleItem>) response.stream()
            .map(o -> CandleConverter.fromArray((List) o)).collect(Collectors.toList());
    }

    private URI buildUrl(long begin, CoinPair symbol, int limit) {

        return
UriComponentsBuilder.fromHttpUrl(binanceConnectionConfiguration.getCandleUrlPrefi
x()
            + binanceConnectionConfiguration.getCandleUrl())
            .queryParams(CandleQueryParams.candleUrlQueryStartTime

```

```

        , begin)
        .queryParam(CandleQueryParams.candleUrlQuerySymbol,
            buildCoinName(symbol))
        .queryParam(CandleQueryParams.candleUrlQueryInterval,
            Interval.checkIntervals(binanceConnectionConfiguration.getCandleInterval()).getType())
        .queryParam(CandleQueryParams.candleUrlQueryLimit, limit)
        .build().toUri();
    }

    private String buildCoinName(CoinPair symbol) {
        return symbol.getBaseCoin() + symbol.getQuoteCoin();
    }

    @Override
    public String getInterval() {
        return binanceConnectionConfiguration.getCandleInterval();
    }

    @Override
    public List<BinanceDatumDto> requestP2PData(String asset, String fiat, DealType
dealType) {
        URI p2pUrl = URI.create(binanceConnectionConfiguration.getP2pUrlPrefix()
            + binanceConnectionConfiguration.getP2pUrl());

        return getDataList(p2pUrl, asset, fiat, dealType);
    }

    private List<BinanceDatumDto> getDataList(URI p2pUrl, String asset, String fiat,
DealType dealType) {
        List<BinanceDatumDto> p2pDataOnPage;
        int maxRowsOnPage = 20;
        BinanceP2PRequestBody binanceP2PRequestBody;
        binanceP2PRequestBody = requestFactory(1, maxRowsOnPage, asset, fiat,
dealType);
        BinanceP2PResponseDto binanceP2PResponseDto =
restTemplate.postForObject(p2pUrl, binanceP2PRequestBody,
BinanceP2PResponseDto.class);
        p2pDataOnPage = binanceP2PResponseDto.getData();
        int totalPricesAmount = binanceP2PResponseDto.getTotal();
        List<BinanceDatumDto> p2pData = new ArrayList<>(p2pDataOnPage);
        int totalRequests = (int) Math.ceil((double) totalPricesAmount / maxRowsOnPage);

        log.info("Total amount of prices of selected coin pair {}", totalPricesAmount);

        for (int page = 2; page <= totalRequests; page++) {
            binanceP2PRequestBody = requestFactory(page, maxRowsOnPage, asset, fiat,
dealType);

```

```

        p2pDataOnPage = restTemplate.postForObject(p2pUrl, binanceP2PRequestBody,
        BinanceP2PResponseDto.class).getData();
        p2pData.addAll(p2pDataOnPage);
    }
    return p2pData;
}

```

```

    private BinanceP2PRequestBody requestFactory(int page, int rows, String asset, String
    fiat, DealType dealType) {
        BinanceP2PRequestBody binanceP2PRequestBody = new
        BinanceP2PRequestBody();
        binanceP2PRequestBody.setAsset(asset);
        binanceP2PRequestBody.setFiat(fiat);
        binanceP2PRequestBody.setDealType(dealType);
        binanceP2PRequestBody.setPage(page);
        binanceP2PRequestBody.setRows(rows);
        return binanceP2PRequestBody;
    }
}

```

```

package com.dkostin.cryptos.loader.provider.binance.convert;

```

```

import java.math.BigDecimal;
import java.util.List;

```

```

import com.dkostin.cryptos.loader.provider.binance.dto.CandleItem;

```

```

public class CandleConverter {

```

```

    public static CandleItem fromArray(List<Object> fields) {
        int i = 0;
        CandleItem ci = new CandleItem();
        ci.setOpenTime((Long) fields.get(i++));
        ci.setOpen(new BigDecimal(fields.get(i++).toString()));
        ci.setHigh(new BigDecimal(fields.get(i++).toString()));
        ci.setLow(new BigDecimal(fields.get(i++).toString()));
        ci.setClose(new BigDecimal(fields.get(i++).toString()));
        ci.setVolume(new BigDecimal(fields.get(i++).toString()));
        ci.setCloseTime((Long) fields.get(i++));
        ci.setQuoteAssetVolume(new BigDecimal(fields.get(i++).toString()));
        ci.setNumberOfTrades(new BigDecimal(fields.get(i++).toString()));
        ci.setTakerBuyBaseAssetVolume(new BigDecimal(fields.get(i++).toString()));
        ci.setTakerBuyQuoteAssetVolume(new BigDecimal(fields.get(i++).toString()));
        ci.setIgnore(new BigDecimal(fields.get(i).toString()));
        return ci;
    }
}

```

```
}
```

```
package com.dkostin.cryptos.loader.provider.binance;
```

```
import lombok.extern.slf4j.Slf4j;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.Marker;
import org.slf4j.MarkerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.stereotype.Service;
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CompletableFuture;
```

```
import com.dkostin.cryptos.loader.domain.CoinPair;
import com.dkostin.cryptos.loader.domain.CoinPriceHistory;
import com.dkostin.cryptos.loader.domain.Sources;
import com.dkostin.cryptos.loader.provider.PriceBarProvider;
import
com.dkostin.cryptos.loader.provider.binance.config.BinanceConnectionConfiguration;
import com.dkostin.cryptos.loader.provider.binance.connector.BinanceConnector;
import com.dkostin.cryptos.loader.provider.binance.dto.CandleItem;
import com.dkostin.cryptos.loader.provider.binance.service.BinanceRestriction;
import com.dkostin.cryptos.loader.service.CoinPairService;
import com.dkostin.cryptos.loader.service.CoinPriceHistoryService;
```

```
//TODO think about move all beans related to the certain provider into separate .xml
spring configuration file
```

```
// and maybe into separate mvn module
```

```
// to be able to disable it in one action
```

```
@Service
```

```
@Slf4j
```

```
public class BinancePriceBarProvider implements PriceBarProvider {
    private static final long MAX_DELTA = 1_800_000;
    private Logger logger = LoggerFactory.getLogger(BinancePriceBarProvider.class);
```

```
    BinanceConnector binanceConnector;
    CoinPriceHistoryService coinPriceHistoryService;
    CoinPairService coinPairService;
    BinanceRestriction binanceRestriction;
    BinanceConnectionConfiguration connectionConfiguration;
    Long firstTime;
```

[illegible]

```

        CoinPriceHistory coinPriceHistory =
mapCandleItemToCoinPriceHistory(candleItem, coinPair, Sources.BINANCE.getId(),
binanceConnector.getInterval());
        coinPriceHistories.add(coinPriceHistory);
    }
    try {
        coinPriceHistoryService.saveAll(coinPriceHistories);
    } catch (DataIntegrityViolationException dive){
        //log.error("Attempt to insert or update data results in violation of an
integrity constraint", dive);
        Marker fatal = MarkerFactory.getMarker("FATAL");
        logger.error(fatal, "this is a fatal message");
    }
    candleItems = binanceConnector.requestBarData(begin, coinPair);
    numberOfCalls++;

    if (numberOfCalls >= connectionConfiguration.getBinanceLimit()) {
        secondTime = System.currentTimeMillis();
        log.info("In the period between {} and {} : {} requests were done to
request price from binance", firstTime, secondTime, numberOfCalls);
        if (secondTime - firstTime < 500000) {
            binanceRestriction.doThreadSleep();
        }
        numberOfCalls = 0;
        firstTime = System.currentTimeMillis();
    }
}
}
} catch (DataAccessException access) {
    log.error("Cannot get access to data in database ", access);
} catch (Exception exception) {
    log.error("Following exception occurred: ", exception);
}
});
}
}

```

```

@Override
public CoinPriceHistory mapCandleItemToCoinPriceHistory(CandleItem candleItem,
CoinPair coinPair, int exchangeId, String candleType) {
    CoinPriceHistory coinPriceHistory = new CoinPriceHistory();
    coinPriceHistory.setExchangeId(exchangeId);
    coinPriceHistory.setOpen(candleItem.getOpen());
    coinPriceHistory.setHigh(candleItem.getHigh());
    coinPriceHistory.setLow(candleItem.getLow());
    coinPriceHistory.setClose(candleItem.getClose());
    coinPriceHistory.setVolume(candleItem.getVolume());
}

```

```

coinPriceHistory.setNumberOfTrades(candleItem.getNumberOfTrades().longValueExact(
));
    coinPriceHistory.setCandleTimeOpen(candleItem.getOpenTime());
    coinPriceHistory.setCandleTimeClose(candleItem.getCloseTime());
    coinPriceHistory.setCoinPairId(coinPair.getId());
    coinPriceHistory.setCandleType(candleType);
    return coinPriceHistory;
}

@Override
public int getId() {
    return Sources.BINANCE.getId();
}
}

```

```

package com.dkostin.cryptos.loader.provider.binance.service;

import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.concurrent.TimeUnit;

import
com.dkostin.cryptos.loader.provider.binance.config.BinanceConnectionConfiguration;

@Service
@Slf4j
public class BinanceRestriction {
    BinanceConnectionConfiguration binanceConnectionConfiguration;

    @Autowired
    public BinanceRestriction(BinanceConnectionConfiguration
binanceConnectionConfiguration){
        this.binanceConnectionConfiguration = binanceConnectionConfiguration;
    }

    public void doThreadSleep(){
        try {
            log.info("Thread go to sleep for minutes: {} ",
binanceConnectionConfiguration.getSecondsToSleepBeforeNextRequest()/60);

TimeUnit.SECONDS.sleep(binanceConnectionConfiguration.getSecondsToSleepBeforeN
extRequest());
        } catch (Exception e){
            e.printStackTrace();
        }
    }
}

```

```
    }
}
```

```
package com.dkostin.cryptos.loader.rest;
```

```
import com.dkostin.cryptos.loader.resources.LoaderResource;
import com.dkostin.cryptos.loader.service.ExtractorService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
import java.util.List;
```

```
@RestController
@RequestMapping("/extract")
public class LoaderControllerImpl implements LoaderResource {
    private ExtractorService extractorService;

    @Autowired
    public LoaderControllerImpl(ExtractorService extractorService) {
        this.extractorService = extractorService;
    }

    @Override
    public List<String> getSupportedPairs() {
        return null;
    }

    @RequestMapping("/exchange")
    @GetMapping
    @Override
    public ResponseEntity<?> getCoinPriceHistories() {
        extractorService.barDoExtraction();
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

```
package com.dkostin.cryptos.loader;
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;
import org.springframework.retry.annotation.EnableRetry;
```

```

@SpringBootApplication(scanBasePackages = {"com.dkostin.dlt.cryptos.loader"})
@EnableScheduling
@EnableRetry
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

}

```

```

package com.dkostin.cryptos.predictor.config;

```

```

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

import lombok.Data;

```

```

@Data
@Configuration
@PropertySource("classpath:path.properties")
@ConfigurationProperties(prefix = "csv")
public class CsvConfiguration {
    private String input;
    private String expected;
    private int batchSize;
}

```

```

package com.dkostin.cryptos.predictor.controller;

```

```

import java.io.IOException;

```

```

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestBody;

```

```

import com.dkostin.cryptos.predictor.model.ConditionCsv;
import com.dkostin.cryptos.predictor.model.ConditionLstm;
import com.dkostin.cryptos.predictor.model.ModelTrainingResult;

```

```

public interface LSTMController {
    ResponseEntity<ModelTrainingResult> launchLSTM(ConditionLstm conditionLstm)
    throws IOException, InterruptedException;

    void testCSV(@RequestBody ConditionCsv conditionCsv);

    ResponseEntity<?> downloadCsv();
}

```

```

}

package com.dkostin.cryptos.predictor.model;

import lombok.Data;

@Data
public class ConditionCsv {
    private int coinPairId = 1;
    private int exchangeId = 1;
    private String candleType = "1h";
    private long startDate = 1420074000000L;
}

package com.dkostin.cryptos.predictor.model;

import org.springframework.context.annotation.Configuration;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Configuration
public class ConditionLstm {
    private int coinPairId;
    private int exchangeId;
    private String candleType;
    private long startDate;
    private double trainingPercent;
    private int numberOfFilesForOneIteration;
    private int nbEpochs;
    private double learningRate;;
    private int batchSize;
    private int lstmLayerSize;
    private int numLabelClasses;
}

package com.dkostin.cryptos.predictor.model;

import java.util.List;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data

```

```

@AllArgsConstructor
@NoArgsConstructor
public class ModelTrainingResult {
    private ConditionLstm conditionLstm;
    private List<Double> oneIterationAucs;
    private double auc;
}

package com.dkostin.cryptos.predictor.service.impl;

import java.io.File;
import java.io.IOException;
import java.math.BigDecimal;
import java.util.LinkedList;
import java.util.List;
import java.util.Objects;
import java.util.Optional;

import javax.annotation.PostConstruct;

import org.datavec.api.records.reader.SequenceRecordReader;
import org.datavec.api.records.reader.impl.csv.CSVSequenceRecordReader;
import org.datavec.api.split.NumberedFileInputSplit;
import org.deeplearning4j.datasets.datavec.SequenceRecordReaderDataSetIterator;
import org.deeplearning4j.nn.api.OptimizationAlgorithm;
import org.deeplearning4j.nn.conf.ComputationGraphConfiguration;
import org.deeplearning4j.nn.conf.NeuralNetConfiguration;
import org.deeplearning4j.nn.conf.layers.LSTM;
import org.deeplearning4j.nn.conf.layers.RnnOutputLayer;
import org.deeplearning4j.nn.graph.ComputationGraph;
import org.deeplearning4j.nn.weights.WeightInit;
import org.deeplearning4j.util.ModelSerializer;
import org.nd4j.evaluation.classification.ROC;
import org.nd4j.linalg.activations.Activation;
import org.nd4j.linalg.api.ndarray.INDArray;
import org.nd4j.linalg.dataset.api.DataSet;
import org.nd4j.linalg.dataset.api.iterator.DataSetIterator;
import org.nd4j.linalg.dataset.api.preprocessor.DataNormalization;
import org.nd4j.linalg.dataset.api.preprocessor.NormalizerMinMaxScaler;
import org.nd4j.linalg.factory.Nd4j;
import org.nd4j.linalg.learning.config.Adam;
import org.nd4j.linalg.lossfunctions.LossFunctions;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.dkostin.cryptos.predictor.config.CsvConfiguration;
import com.dkostin.cryptos.predictor.dto.CsvCoinPriceHistory;
import com.dkostin.cryptos.predictor.model.ConditionLstm;
import com.dkostin.cryptos.predictor.model.ModelTrainingResult;

```

```

import com.dkostin.cryptos.predictor.model.TestFiles;
import com.dkostin.cryptos.predictor.service.CoinPriceHistoryService;
import com.dkostin.cryptos.predictor.service.LSTMService;
import com.dkostin.cryptos.predictor.service.TestData;

import lombok.extern.slf4j.Slf4j;

@Slf4j
@Service
public class LSTMServiceImpl implements LSTMService {
    private static final String MODEL_BACKUP_PATH =
        "src/main/resources/model/model_backup.zip";
    private static final int NB_INPUTS = 12;

    private final CsvConfiguration csvConfiguration;

    private ComputationGraph model;

    private final CoinPriceHistoryService coinPriceHistoryService;

    @Autowired
    public LSTMServiceImpl(CsvConfiguration csvConfiguration, CoinPriceHistoryService
        coinPriceHistoryService) {
        this.csvConfiguration = csvConfiguration;
        this.coinPriceHistoryService = coinPriceHistoryService;
    }

    @PostConstruct
    private void init() throws IOException {
        File modelFile = new File(MODEL_BACKUP_PATH);

        if (modelFile.exists()) {
            model = ModelSerializer.restoreComputationGraph(modelFile);
        }
    }

    private void saveModel() throws IOException {
        File modelFile = new File(MODEL_BACKUP_PATH);
        modelFile.getParentFile().mkdirs();
        modelFile.createNewFile();

        ModelSerializer.writeModel(model, modelFile, true);
    }

    @Override
    public double predict(int coinPairId, int exchangeId, String candleType) {
        List<CsvCoinPriceHistory> coinPriceHistories =
            coinPriceHistoryService.getLastBatchForPrediction(coinPairId, exchangeId, candleType);
        INDArray input = Nd4j.create(dataForPredictionForm(coinPriceHistories));
    }

```

```

INDArray[] output = model.output(input.reshape(new int[] {64, 12, 1})); // maybe:
model.output(input.reshape(1, 12, 64));
double growResult = output[0].getDouble(63, 1, 1);
double dropResult = output[0].getDouble(63, 0, 1);
log.info("Possibility of growth percentage: " + growResult);
log.info("Possibility of drop percentage: " + dropResult);
return growResult;
}

```

```

private double[][] dataForPredictionForm(List<CsvCoinPriceHistory> coinPriceHistories)
{
    double[][] dataInput = new double[64][12];
    // todo why we don't have date???
    for (int i = 0; i < coinPriceHistories.size(); i++) {
        CsvCoinPriceHistory coinPriceHistory = coinPriceHistories.get(i);
        dataInput[i][0] = coinPriceHistory.getOpen().doubleValue();
        dataInput[i][1] = coinPriceHistory.getHigh().doubleValue();
        dataInput[i][2] = coinPriceHistory.getLow().doubleValue();
        dataInput[i][3] = coinPriceHistory.getClose().doubleValue();
        dataInput[i][4] = coinPriceHistory.getVolume().doubleValue();
        // todo indicators: next, just uncomment me
        dataInput[i][5] =
            Optional.ofNullable(coinPriceHistory.getMfi()).map(BigDecimal::doubleValue).orElse(0.0);
        dataInput[i][6] =
            Optional.ofNullable(coinPriceHistory.getObv()).map(BigDecimal::doubleValue).orElse(0.0);
        dataInput[i][7] =
            Optional.ofNullable(coinPriceHistory.getEma()).map(BigDecimal::doubleValue).orElse(0.0);
        dataInput[i][8] =
            Optional.ofNullable(coinPriceHistory.getSma()).map(BigDecimal::doubleValue).orElse(0.0);
        dataInput[i][9] =
            Optional.ofNullable(coinPriceHistory.getCmo()).map(BigDecimal::doubleValue).orElse(0.0);
        dataInput[i][10] =
            Optional.ofNullable(coinPriceHistory.getAo()).map(BigDecimal::doubleValue).orElse(0.0);
        dataInput[i][11] =
            Optional.ofNullable(coinPriceHistory.getSo()).map(BigDecimal::doubleValue).orElse(0.0);
    }

    return dataInput;
}

```

@Override

```

public ModelTrainingResult launchLSTM(ConditionLstm conditionLstm) throws
IOException, InterruptedException {
    List<Double> aucs = new LinkedList<>();
    ROC roc = new ROC();
    TestFiles testFiles = new TestFiles();
    TestData testDataToFile = new TestDataImpl(testFiles);

    String featureBaseDir =
        csvConfiguration.getInput() + "/model1/" + conditionLstm.getExchangeId() + "/" +
        conditionLstm.getCandleType() + "/"
        + conditionLstm.getCoinPairId();
    String closeBaseDir =
        csvConfiguration.getExpected() + "/model1/" + conditionLstm.getExchangeId() + "/" +
        conditionLstm.getCandleType() + "/"
        + conditionLstm.getCoinPairId();

    int filesNumber = Objects.requireNonNull(new File(closeBaseDir).list()).length; // 95
    double trainSetRatio = conditionLstm.getTrainingPercent() / 100; // 0.8
    int trainFilesNumber = (int)
        Math.round(conditionLstm.getNumberOfFilesForOneIteration() * trainSetRatio);

    //train
    int currentBatch;
    String featuresFileNameTemplate = featureBaseDir + "/coinPriceHistory_%d.csv";
    String labelsFileNameTemplate = closeBaseDir + "/closePrice_%d.csv";

    for (currentBatch = 0; currentBatch + trainFilesNumber < filesNumber; currentBatch +=
        conditionLstm.getNumberOfFilesForOneIteration()) {
        SequenceRecordReader trainFeatures = new CSVSequenceRecordReader(1, ",");
        SequenceRecordReader trainLabels = new CSVSequenceRecordReader();
        DataNormalization normalizer = new NormalizerMinMaxScaler();
        trainFeatures.initialize(new NumberedFileInputSplit(featuresFileNameTemplate,
            currentBatch, currentBatch + trainFilesNumber - 1));
        trainLabels.initialize(new NumberedFileInputSplit(labelsFileNameTemplate,
            currentBatch, currentBatch + trainFilesNumber - 1));
        DataSetIterator trainData = new SequenceRecordReaderDataSetIterator(trainFeatures,
            trainLabels,
            filesNumber, 2, false,
            SequenceRecordReaderDataSetIterator.AlignmentMode.ALIGN_END);

        while (trainData.hasNext()) {
            DataSet batch = trainData.next();
            normalizer.fit(batch);
            normalizer.transform(batch);
        }

        initModel(conditionLstm);
        model.fit(trainData, conditionLstm.getNbEpochs());
    }
}

```

```

currentBatch -= conditionLstm.getNumberOfFilesForOneIteration();

int rest = filesNumber % conditionLstm.getNumberOfFilesForOneIteration();
int restTrainFilesNumber = (int) (rest * conditionLstm.getTrainingPercent() / 100);

if (restTrainFilesNumber != 0) {
    SequenceRecordReader trainFeatures = new CSVSequenceRecordReader(1, ",");
    SequenceRecordReader trainLabels = new CSVSequenceRecordReader();
    DataNormalization normalizer = new NormalizerMinMaxScaler();

    trainFeatures.initialize(
        new NumberedFileInputSplit(featuresFileNameTemplate, currentBatch, currentBatch +
            restTrainFilesNumber - 1));
    trainLabels.initialize(new NumberedFileInputSplit(labelsFileNameTemplate,
        currentBatch, currentBatch + restTrainFilesNumber - 1));
    DataSetIterator trainData = new SequenceRecordReaderDataSetIterator(trainFeatures,
        trainLabels,
        filesNumber, 2, false,
        SequenceRecordReaderDataSetIterator.AlignmentMode.ALIGN_END);

    while (trainData.hasNext()) {
        DataSet batch = trainData.next();
        normalizer.fit(batch);
        normalizer.transform(batch);
    }

    model.fit(trainData, conditionLstm.getNbEpochs());
}

//test
int testFilesNumber = conditionLstm.getNumberOfFilesForOneIteration() -
    trainFilesNumber;

for (
    currentBatch = 0;
    currentBatch + trainFilesNumber + testFilesNumber < filesNumber;
    currentBatch += conditionLstm.getNumberOfFilesForOneIteration()) {
    SequenceRecordReader testFeatures = new CSVSequenceRecordReader(1, ",");
    SequenceRecordReader testLabels = new CSVSequenceRecordReader();
    DataNormalization normalizer = new NormalizerMinMaxScaler();
    testFeatures.initialize(new NumberedFileInputSplit(featuresFileNameTemplate,
        currentBatch + trainFilesNumber + 1,
        currentBatch + trainFilesNumber + testFilesNumber));
    testLabels.initialize(new NumberedFileInputSplit(labelsFileNameTemplate, currentBatch
        + trainFilesNumber + 1,
        currentBatch + trainFilesNumber + testFilesNumber));
    DataSetIterator testData = new SequenceRecordReaderDataSetIterator(testFeatures,
        testLabels, conditionLstm.getBatchSize(), 2, false,
        SequenceRecordReaderDataSetIterator.AlignmentMode.ALIGN_END);

```

```

while (testData.hasNext()) {
    DataSet batch = testData.next();
    INDArray[] out = model.output(batch.getFeatures());
    testDataToFile.addData(batch.getFeatures(), batch.getLabels(), out[0]);
    normalizer.fit(batch);
    normalizer.transform(batch);
    INDArray[] output = model.output(batch.getFeatures());
    roc.evalTimeSeries(batch.getLabels(), output[0]);
}

double auc = roc.calculateAUC();
log.info("Area under curve - {}", auc);
aucs.add(auc);
}
currentBatch -= conditionLstm.getNumberOfFilesForOneIteration();

int restTestFilesNumber = rest - restTrainFilesNumber;

if (restTrainFilesNumber != 0) {
    SequenceRecordReader testFeatures = new CSVSequenceRecordReader(1, ",");
    SequenceRecordReader testLabels = new CSVSequenceRecordReader();
    DataNormalization normalizer = new NormalizerMinMaxScaler();
    testFeatures.initialize(new NumberedFileInputSplit(featuresFileNameTemplate,
        currentBatch + restTrainFilesNumber,
        currentBatch + restTrainFilesNumber + restTestFilesNumber - 1));
    testLabels.initialize(new NumberedFileInputSplit(labelsFileNameTemplate, currentBatch
        + restTrainFilesNumber,
        currentBatch + restTrainFilesNumber + restTestFilesNumber - 1));
    DataSetIterator testData = new SequenceRecordReaderDataSetIterator(testFeatures,
        testLabels, conditionLstm.getBatchSize(), 2, false,
        SequenceRecordReaderDataSetIterator.AlignmentMode.ALIGN_END);

    while (testData.hasNext()) {
        DataSet batch = testData.next();
        INDArray[] out = model.output(batch.getFeatures());
        testDataToFile.addData(batch.getFeatures(), batch.getLabels(), out[0]);
        normalizer.fit(batch);
        normalizer.transform(batch);
        INDArray[] output = model.output(batch.getFeatures());
        roc.evalTimeSeries(batch.getLabels(), output[0]);
    }

    double auc = roc.calculateAUC();
    log.info("Area under curve - {}", auc);
    aucs.add(auc);
}

saveModel();

```

```

testDataToFile.saveToCsv(conditionLstm, aucs, aucs.get(aucs.size() - 1));
ModelTrainingResult modelTrainingResult = new ModelTrainingResult(conditionLstm,
aucs, aucs.get(aucs.size() - 1));
return modelTrainingResult;

}

private void initModel(ConditionLstm conditionLstm) {
if (model == null) {
ComputationGraphConfiguration conf = new NeuralNetConfiguration.Builder()

.seed(System.currentTimeMillis())
.optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
.updater(new Adam(conditionLstm.getLearningRate()))
.weightInit(WeightInit.XAVIER)
.dropOut(0.25)
.graphBuilder()
.addInputs("ohlcv")
.setOutputs("closePrice")
.addLayer("L1",
new
LSTM.Builder().nIn(NB_INPUTS).nOut(conditionLstm.getLstmLayerSize()).dropOut(0.2
).forgetGateBiasInit(1).build(),
"ohlcv")
.addLayer("L2",
new
LSTM.Builder().nIn(conditionLstm.getLstmLayerSize()).nOut(conditionLstm.getLstmLa
yerSize()).forgetGateBiasInit(1)
.dropOut(0.2).build(), "L1")
.addLayer("L3",
new
LSTM.Builder().nIn(conditionLstm.getLstmLayerSize()).nOut(conditionLstm.getLstmLa
yerSize()).forgetGateBiasInit(1)
.dropOut(0.2).build(), "L2")
.addLayer("L4",
new
LSTM.Builder().nIn(conditionLstm.getLstmLayerSize()).nOut(conditionLstm.getLstmLa
yerSize()).forgetGateBiasInit(1)
.dropOut(0.2).build(), "L3")
.addLayer("closePrice", new
RnnOutputLayer.Builder(LossFunctions.LossFunction.MCXENT).activation(Activation.S
OFTMAX)
.nIn(conditionLstm.getLstmLayerSize()).nOut(conditionLstm.getNumLabelClasses()).buil
d(), "L4")
.build());

model = new ComputationGraph(conf);
}
}

```

```
# Use an official OpenJDK runtime as a parent image
FROM openjdk:17-jdk-slim

# Set the working directory to /app
WORKDIR /app

# Copy the JAR file into the container at /app
COPY impl/target/crypto-prices-predictor-impl-1.0-SNAPSHOT.jar /app/app.jar

# Expose the port that your Spring Boot application will run on
EXPOSE 8082

# Specify the command to run on container startup
CMD ["java", "-jar", "app.jar"]

# Use an official OpenJDK runtime as a parent image
FROM openjdk:17-jdk-slim

# Set the working directory to /app
WORKDIR /app

# Copy the JAR file into the container at /app
COPY impl/target/crypto-prices-loader-impl-1.0-SNAPSHOT.jar /app/app.jar

# Expose the port that your Spring Boot application will run on
EXPOSE 8052

# Specify the command to run on container startup
CMD ["java", "-jar", "app.jar"]
```