Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента *Шарандо Артема Андрійовича*
(ПІБ)

академічної групи *122М-21-1*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Обгрунтування структури інформаційної системи спостерігання та прогнозу динамікі криптовалют*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи				
розділів:				
спеціальний	*проф. Мещеряков Л.І.*			
Рецензент				
Нормоконтролер	*проф. Лактіонов І.С.*			

Дніпро

2022

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

_____ Алексєєв М.О.
(підпис) (прізвище, ініціали)

« » _____ 20 22 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

студенту *122М-21-1* *Шарандо Артему Андрійовичу*
(група) (прізвище та ініціали)

Тема дипломного проекту *Обгрунтування структури інформаційної системи спостерігання та прогнозу динамікі криптовалют*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 31.10.2022 р. № 1200-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – динаміка зростання або падіння ціни криптовалют.

Предмет досліджень – модель прогнозу та методи оцінки динаміки цін криптовалют.

Мета роботи – дослідження та вдосконалення існуючих методик побудови прогнозуючих моделей та розробці системи підтримки прийняття рішень для короткострокового прогнозування курсу криптовалют з використанням моделей нейронних мереж.

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна результатів кваліфікаційної роботи є відслідковування руху криптовалют та обґрунтування факторів впливу на зміну ціни за допомогою математичних індикаторів та історії динаміки у системі оцінювання та аналізу.

Практична цінність результатів полягає у тому, що розроблено програмне забезпечення для спостерігання та аналізу даних криптовалют, показ зміни динаміки та прогнозування подальшої зміни ціни віртуальної валюти.

4 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз теми та постановка задачі	01.09.2022-30.09.2022
Побудова нечіткої моделі представлення даних для вирішення задачі ідентифікаційної експертизи	01.09.2022-31.11.2022
Створення автоматизованої системи для вирішення задачі побудови моделі аналізу та спостерігання	01.09.2022-15.11.2022

Завдання видав _____ *Мещеряков Л.І.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Шарандо А.А.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *1.09.2022 р.*

Термін подання дипломного проекту до ЕК *20.12.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 96 стор.,18 рис., 5 таблиць, 3 додатка, 40 джерел.

Об'єкт дослідження: динаміка зростання або падіння ціни криптовалют.

Предмет дослідження: модель прогнозу та методи оцінки динаміки цін криптовалют.

Мета магістерської роботи: дослідження та вдосконалення існуючих методик побудови прогнозуючих моделей та розробці системи підтримки прийняття рішень для короткострокового прогнозування курсу криптовалют з використанням моделей нейронних мереж.

Методи дослідження. Для вирішення поставлених задач використані методи: аналізу даних, теорії розпізнавання образів з області обчислювального інтелекту, об'єктно-орієнтоване програмування, модель прогнозування за допомогою нейронної мережі.

Наукова новизна результатів дипломної роботи є відслідковування руху криптовалют та обґрунтування факторів впливу на зміну ціни за допомогою математичних індикаторів та історії динаміки у системі оцінювання та аналізу.

Практична цінність результатів полягає у тому, що розроблено програмне забезпечення для спостерігання та аналізу даних криптовалют, показ зміни динаміки та прогнозування подальшої зміни ціни віртуальної валюти.

Список ключових слів: криптовалюта, методи спостерігання та аналізу, нейронні мережі, біржа, Bitcoin, база даних.

ABSTRACT

Explanatory note: 96 pages, 18 figures, 5 tables, 3 applications, 40 sources.

Object of research: dynamics of the rise or fall of the price of cryptocurrencies.

Subject of research: forecast model and methods of evaluating cryptocurrency price dynamics.

Purpose of Master's thesis: research and improvement of existing methods of building forecasting models and development of a decision support system for short-term forecasting of the exchange rate of cryptocurrencies using neural network models.

Research methods. To solve the set tasks, the following methods were used: data analysis, pattern recognition theories from the field of computational intelligence, object-oriented programming, a prediction model using a neural network.

Originality of research is results of the diploma work include tracking the movement of cryptocurrencies and substantiating the factors influencing price changes using mathematical indicators and the history of dynamics in the evaluation and analysis system.

Practical value of the results is that software has been developed for monitoring and analyzing crypto currency data, showing changes in dynamics and predicting further changes in the price of virtual currency.

Keywords: cryptocurrency, methods of observation and analysis, neural networks, stock exchange, Bitcoin, database.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

LSTM – (англ. Long Short Term Memory) довга короткострокова пам’ять;

RNN – (англ. Recurrent neural network) рекурентні нейронні мережі;

BTC – біткоін;

API – (англ. Application Programming Interface) інтерфейс програмування програм;

OBV – (англ. On Balance Volume) індикатор Балансового обсягу;

SMA – (англ. Simple Moving Average) проста ковзаюча середня;

EMA – (англ. Exponential Moving Average) експоненціальне ковзне середнє;

MSE – (англ. Mean Square Error) середня квадратична помилка;

URL – (англ. Uniform Resource Locator) уніфікований покажчик інформаційного ресурсу;

P2P – одноранговий криптообмін;

USD – долар;

RNN – рекурентні нейронні мережі;

ЗМІСТ

ВСТУП

Актуальність роботи. В світі зараз швидкими темпами розвивається новий ринок криптовалют. Разом з цим, зростає кількість фізичних осіб та компаній, основною задачею яких є торгівля валютою з метою отримання прибутку. Таким чином, розробка та застосування систем прогнозування курсу криптовалют у процесі прийняття рішення щодо здійснення операцій купівлі продажу криптовалюти є актуальною на сьогоднішній день. Можливість отримати достовірну інформацію наперед дає суб'єкту владу над ситуацією, можливість реально впливати на неї і змінювати на свою користь, зводити до мінімуму ймовірність настання несприятливих наслідків і здійснювати ризик-менеджмент.

У рамках кваліфікаційної роботи необхідно вирішити такі задачі:

- Проаналізовати API криптобірж;
- провести огляд та аналіз існуючих математичних методів моделювання і прогнозування криптовалютних котирувань;
- розробити архітектуру системи підтримки прийняття рішень для аналізу, моделювання та прогнозування курсу криптовалюти;
- розробити програмний продукт, в якому реалізувати роботу із фінансовими часовими рядами за допомогою нейронних мереж та моделей експоненційного згладжування;
- використати програмний продукт на реальних даних та провести порівняльний аналіз із обґрунтованим вибором кращої моделі.

Мета роботи полягає у розробці теоретичних положень про криптовалюту за допомогою спостереження та аналізу даних за допомого моделі аналізу з використанням нейронних мереж, дослідженні та вдосконаленні існуючих методів побудови прогнозуючих моделей та розробці системи підтримки

прийняття рішень для короткострокового прогнозування курсу криптовалют на основі інтелектуального аналізу даних з використанням моделей експоненційного згладжування та нейронних мереж.

Об'єкт досліджень - набір фінансових даних щодо операцій купівлі та продажу криптовалюти на торговій онлайн-платформі

Предмет досліджень - математичні методи побудови предикативних моделей, а саме: інформаційна технологія, нейронні мережі довгої короткострокової пам’яті (LSTM) та моделі експоненційного згладжування.

Методи дослідження. Підхід системного аналізу, методи системного аналізу, методи інтелектуального аналізу даних, моделі експоненційного згладжування, нейронні мережі LSTM архітектури

Структура та обсяг роботи. Робота складається з введення, трьох розділів і висновків. Містить 96 сторінку друкарського тексту, у тому числі сторінок тексту основної частини з 18 малюнками, списку використаних джерел з 40 найменуваннями на 4 сторінках.

РОЗДІЛ 1

ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості про криптовалюту

Криптовалюта - це цифрова платіжна система, яка не покладається на банки для перевірки транзакцій. Це однорангова система, яка дозволяє будь-кому будь-де надсилати й отримувати платежі. Замість того, щоб бути фізичними грошима, які переносяться та обмінюються в реальному світі, платежі в криптовалюті існують виключно як цифрові записи в онлайн-базі даних, що описують конкретні транзакції. Коли ви переказуєте кошти в криптовалюті, транзакції реєструються в загальнодоступній книзі. Криптовалюта зберігається в цифрових гаманцях [1].

Свою назву криптовалюта отримала через використання шифрування для перевірки транзакцій. Це означає, що вдосконалене кодування бере участь у зберіганні та передачі даних криптовалюти між гаманцями та публічними книгами. Метою шифрування є забезпечення безпеки та безпеки.

Першою криптовалютою був біткойн (BTC), який був заснований у 2009 році та залишається найвідомішою сьогодні. Значна частина інтересу до криптовалют полягає в торгівлі заради отримання прибутку, а спекулянти часом підвищують ціни до небес. Криптовалюти працюють у розподіленій публічному носії, який називається блокчейн, запис усіх транзакцій, які оновлюються та зберігаються власниками валют [2].

Одиниці криптовалюти створюються за допомогою процесу, що називається майнингом, який передбачає використання потужності комп’ютера для вирішення складних математичних задач, які генерують монети. Користувачі також можуть купувати валюти у брокерів, а потім зберігати та витрачати їх за допомогою криптографічних гаманців.

Якщо ви володієте криптовалютою, ви не володієте нічим матеріальним.

Те, що ви володієте, - це ключ, який дозволяє переміщувати запис або одиницю вимірювання від однієї особи до іншої без довіреної третьої сторони.

Незважаючи на те, що біткойн існує з 2009 року, криптовалюти та застосування технології блокчейн все ще розвиваються у фінансовому плані, і в майбутньому очікується більше використання. Транзакції, включно з облігаціями, акціями та іншими фінансовими активами, зрештою можуть торгуватися за допомогою цієї технології[3].

Переваги криптовалюти:

- Захист від інфляції. Інфляція призвела до того, що вартість багатьох валют з часом знизилася. Майже кожна криптовалюта на момент запуску випускається з фіксованою сумою. Вихідний код визначає кількість будь-якої монети; наприклад, у світі випущено лише 21 мільйон біткойнів. Отже, у міру зростання попиту його вартість зростатиме, що дозволить не відставати від ринку та, у довгостроковій перспективі, запобігатиме інфляції.
- Самоврядний і керований. Управління та підтримка будь-якої валюти є головним фактором її розвитку. Криптовалютні транзакції зберігаються розробниками/майнерами на їх обладнанні, і вони отримують комісію за транзакції як винагороду за це. Оскільки майнери отримують за це гроші, вони зберігають записи транзакцій точними та актуальними, зберігаючи цілісність криптовалюти та децентралізованість записів.
- Безпечний і приватний. Конфіденційність і безпека завжди були головною проблемою для криптовалют. Блокчейн-леджер заснований на різних математичних головоломках, які важко розшифрувати. Це робить криптовалюту більш безпечною, ніж звичайні електронні транзакції. Криптовалюти для кращої безпеки та конфіденційності використовують псевдоніми, які не пов’язані з жодним користувачем, обліковим записом або збереженими даними, які можуть бути пов’язані з профілем.

- Обмін валюти можна легко здійснити. Криптовалюту можна купити за допомогою багатьох валют, таких як долар США, європейський євро, британський фунт, індійська рупія або японська єна. За допомогою різних криптовалютних гаманців і бірж одну валюту можна конвертувати в іншу шляхом торгівлі криптовалютою в різних гаманцях і з мінімальними комісіями за транзакції.
- Децентралізованість. Головною перевагою криптовалюти є те, що вони переважно децентралізовані. Багато криптовалюти контролюються розробниками, які їх використовують, і людьми, які мають значну кількість монет, або організацією, яка розробляє їх до того, як вони будуть випущені на ринок. Децентралізація допомагає зберегти валютну монополію вільною та контрольованою, щоб жодна організація не могла визначити потік і вартість монети, що, у свою чергу, збереже її стабільність і безпеку, на відміну від фіатних валют, які контролюються урядом.
- Рентабельний спосіб транзакції. Одним із основних способів використання криптовалют є пересилання грошей через кордон. За допомогою криптовалюти комісія за транзакцію, яку сплачує користувач, зменшується до незначної або нульової суми. Це робиться завдяки тому, що треті сторони, як-от VISA або PayPal, не потребують перевірки транзакції. Це усуває необхідність сплачувати додаткові комісії за транзакції.
- Швидкий спосіб переказу коштів. Криптовалюти завжди залишалися оптимальним рішенням для транзакцій. Транзакції, як міжнародні, так і внутрішні в криптовалютах, відбуваються блискавично. Це пов’язано з тим, що для перевірки потрібно дуже мало часу, оскільки існує дуже мало перешкод, які потрібно подолати [4].

Недоліки криптовалюти:

- Може використовуватися для незаконних операцій. Оскільки

конфіденційність і безпека транзакцій з криптовалютою є високими, уряду важко відстежити будь-якого користувача за адресою його гаманця або стежити за його даними. У минулому біткойн використовувався як спосіб обміну грошей у багатьох незаконних угодах, таких як покупка наркотиків у темній мережі. Деякі також використовують криптовалюти для конвертації своїх незаконно отриманих грошей через чистого посередника, щоб приховати їх джерело.

- Втрата даних може спричинити фінансові втрати. Розробники хотіли створити практично не відстежуваний вихідний код, сильний захист від злому та непроникні протоколи автентифікації. Це зробить безпечнішим розміщення грошей у криптовалютах, ніж фізичну готівку чи банківські сховища. Але якщо будь-який користувач втратить закритий ключ до свого гаманця, його неможливо повернути. Гаманець залишатиметься заблокованим разом із кількістю монет у ньому. Це призведе до фінансових втрат користувача.
- Децентралізована, але все ще керована певною організацією. Криптовалюти відомі своєю особливістю децентралізації. Але потік і кількість деяких валют на ринку все ще контролюються їх творцями та деякими організаціями. Ці власники можуть маніпулювати монетою для великих коливань її ціни. Навіть монети, які активно торгуються, піддаються таким маніпуляціям, як біткойн, вартість якого подвоїлася в кілька разів у 2017 році та різко знизилась у 2022.
- Деякі монети недоступні в інших фіатних валютах. Деякі криптовалюти можна торгувати лише в одній або кількох фіатних валютах. Це змушує користувача спочатку конвертувати ці валюти в одну з основних валют, як-от біткойн або ефіріум, а потім через інші біржі в бажану валюту. Це стосується лише кількох криптовалют. Завдяки цьому в процесі додаються додаткові комісії за транзакції, що коштує непотрібних грошей.

- Негативний вплив видобутку на навколишнє середовище. Майнінг криптовалют потребує великої обчислювальної потужності та витрат електроенергії, що робить його дуже енергоємним. Найбільшим винуватцем цього є біткойн. Майнінг біткойнів вимагає передових комп’ютерів і багато енергії. На звичайних комп'ютерах це зробити неможливо. Основні майнери біткойнів знаходяться в таких країнах, як Китай, які використовують вугілля для виробництва електроенергії. Це надзвичайно збільшило вуглецевий слід Китаю.
- Сприйнятливий до злому. Незважаючи на те, що криптовалюти дуже безпечні, біржі не настільки безпечні. Більшість бірж зберігають дані гаманця користувачів, щоб належним чином керувати їхнім ідентифікатором користувача. Ці дані можуть бути викрадені хакерами, надаючи їм доступ до багатьох облікових записів. Отримавши доступ, ці хакери можуть легко переказати кошти з цих рахунків. Деякі біржі, такі як Bitfinex або Mt Gox, були зламані в останні роки, і біткойни були вкрадені на тисячі та мільйони доларів США. Більшість бірж сьогодні є дуже безпечними, але завжди є потенціал для нового злому.
- Відсутність політики відшкодування чи скасування. Якщо виникне суперечка між зацікавленими сторонами або якщо хтось помилково надішле кошти на неправильну адресу гаманця, відправник не зможе отримати монету. Цим може скористатися багато людей, щоб обдурити гроші інших. Оскільки відшкодування не передбачено, його можна легко створити для транзакції, продукти чи послуги якої вони не отримували.

Основне призначення майнінгу це створення (видобуток) криптовалюти, логування та підтвердження вже існуючих транзакцій. Емісія криптовалюти – це обчислювальний процес зашифрованих блоків, який має відображення в блокчейні та зберігає в собі транзакції за певний період [5].

Всі учасники ринку окремо збирають свої операції(транзакції) в єдиний

блок, який постійно збільшується. Основною метою цього процесу є прикріплення блоку до загальної бази ланцюга (блокчейну). Учасник ринку, у якого вийшло прикріпити блок, отримує певну криптовалюту, ця операція також оформлюється як транзакція блоку, проте особливого типу.

Кожен транзакційний вузол, що бажає створити блок, працює над обчислювальним завданням, складність якого формується та підбирається самою мережею так, щоб середній час на виконання1 завдання знаходився в межах 10 хвилин. Якщо загальна швидкість створення блоків стрімко збільшується – через кожні N+1(системне значення) блоків завдання ускладнюється, і навпаки. Отже, у окремого учасника знижується шанс вирішити завдання за t хвилин (середній час вирішення). Завдання полягає в підборі відкритого тексту, включаючи блок, такого, щоб застосування до нього хеш-функції SHA256 давало відповідне число, яке менше заданого системного порогу (рис. 1.6). Чим нижчий даний поріг, тим більше часу займе такий перебіг [6].

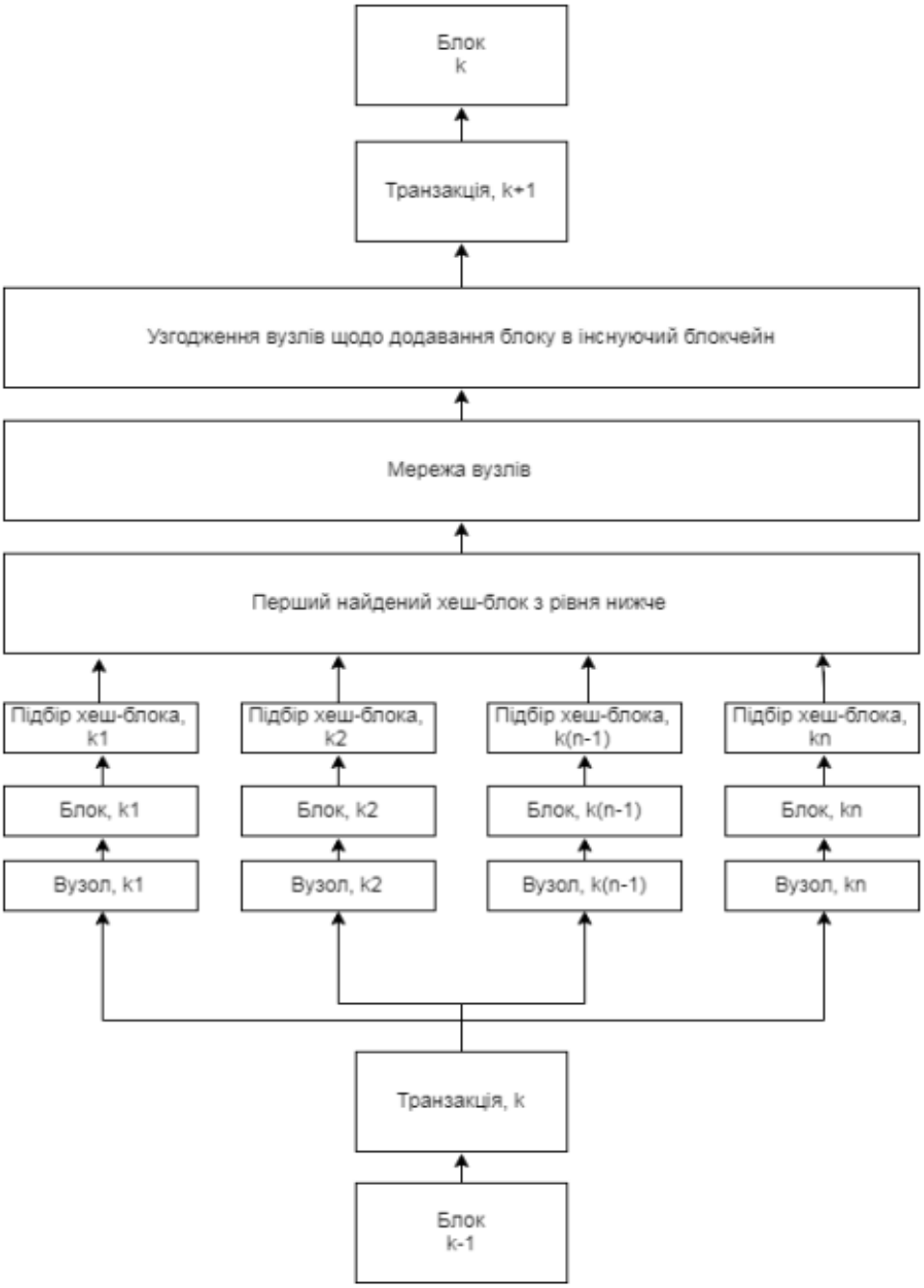

Рис.1. Схематичне зображення роботи блокчейну

1.2. Одноранговий криптообмін

Одноранговий (P2P) криптообмін – це тип криптобірж, який дозволяє трейдерам обмінюватися криптовалютами один з одним без втручання

посередника, наприклад банків чи інших авторитетних організацій.

У результаті перевірені користувачі можуть здійснювати всі операції з криптовалютами без будь-яких проблем. Крім того, криптовалютним біржам P2P не потрібно використовувати книги замовлень для поєднання ордерів на купівлю та продаж і контролю над усіма криптовалютами на їхній платформі.

Натомість ці біржі дають своїм користувачам можливість безпосередньо взаємодіяти один з одним, щоб спростити транзакцію. Таким чином, користувачам не потрібно використовувати посередника для утримання коштів або процесу транзакцій. Однак відсутність посередника значно полегшує користувачам реєстрацію без підтвердження ідентифікатора. Це може послабити безпеку криптобіржі, тому більшість децентралізованих бірж мають багато різних заходів безпеки.

Крім того, криптобіржі P2P дозволяють вам переглядати широкий спектр трейдерів і бачити, що вони можуть вам запропонувати. Вам не потрібно втрачати опіку над своїми коштами, і ви можете переглядати різні способи оплати з різними комісіями за транзакції. Однією з головних причин, чому люди віддають перевагу децентралізованим криптовалютним біржам, а не централізованим, є відсутність єдиної точки відмови. Ці цифрові активи не зберігаються на біржі; кожен трейдер володіє ними самостійно. Платформа просто використовується для транзакцій [7].

Реєструючись на P2P криптообмін, вам потрібно буде лише вказати адресу електронної пошти та пароль. Крім того, вам не потрібно надавати будь-яке підтвердження ідентифікатора для роботи на P2P-криптобіржі більшу частину часу. Потім ви знайдете різні пропозиції купівлі та продажу біткойнів та інших криптовалют . Кожен продавець або покупець вимагатиме різну ставку, спосіб оплати та максимальну або мінімальну суму покупки. Після того, як ви знайдете пропозицію, яка вам найкраще підходить, починається транзакція. Деякі P2P-біржі вимагають, щоб обидві сторони надіслали деяку заставу для договору

умовного депонування, який буде вивільнено після успішної транзакції. Якщо виникне конфлікт, біржа компенсує постраждалій стороні заставою. Інші P2P-біржі вимагають лише від продавця надіслати забезпечення, яке вони знову отримають, коли покупець підтвердить транзакцію.

Ось кілька переваг використання P2P криптообміну:

- Гнучкі способи оплати: Найкраща частина криптовалютних бірж P2P полягає в тому, що вони надають широкий вибір методів оплати, доступність яких залежить від продавця. Це включає подарункову картку, цифрові платежі чи навіть чеки. Простіше кажучи, немає жодних обмежень щодо способів оплати на криптовалютних біржах P2P. Для порівняння, ви побачите, що традиційні централізовані криптовалютні біржі приймають лише платежі кредитною або дебетовою карткою.
- Нижчі комісії: Звичайно, усі криптобіржі P2P вимагають комісії за транзакції для всіх торгів, але вони набагато нижчі, ніж централізовані криптобіржі. Ви також побачите, що ці платформи стягують лише комісію за умовне депонування, оскільки вони не мають посередника.
- Безпека: Незважаючи на поширену думку, більшість криптобірж P2P значно покращили свої системи безпеки. Крім того, вони набагато безпечніші, ніж більшість торгових платформ, оскільки є сторонніми. Система умовного депонування, яка потрібна більшості P2P криптобірж, допомагає захистити кожну сторону, яка бере участь у торгівлі. Це також включає обмін. Оскільки цифрові активи не зберігаються на платформі, користувачам не потрібно турбуватися про втрату своїх криптовалют, оскільки вони ніколи не передають їх на зберігання.
- Доступність: Криптобіржі P2P є більш доступними, ніж інші торгові платформи, оскільки вони не вимагають від користувачів банківських рахунків. Як наслідок, трейдери в регіонах із недостатнім банком можуть легко купувати та продавати криптовалюту. Якщо у вас є доступ до

Інтернету, смартфон і готовність до ризику, ви можете надсилати й отримувати криптовалюту без клопоту.

Ось кілька недоліків використання P2P криптообміну:

- Важко використовувати: Оскільки криптовалютні біржі P2P не мають посередника для регулювання платформи та її операцій, інтерфейс може бути досить складним у використанні. Крім того, платформи не надто інтуїтивно зрозумілі, тому в процесі адаптації доводиться вчитися.
- Швидкість: Відсутність посередника також значно сповільнює роботу платформи, оскільки кожна транзакція займає набагато більше часу. Крім того, неможливо спростити вибір між різними способами оплати без посередників.
- Анонімність: Всупереч поширеній думці, криптобіржі P2P забезпечують меншу анонімність для користувачів, ніж позабіржові криптобіржі. Це пояснюється тим, що кожен користувач торгує безпосередньо з іншим користувачем, і він також може відображати вашу історію торгівлі у вбудованій системі репутації платформи [8].

1.3. Біржі та їх інтерфейс програмування

1.3.1 Біржа CoinBase Pro

Coinbase, є американською публічною компанією, яка керує платформою обміну криптовалют. Coinbase є розподіленою компанією; всі співробітники працюють за допомогою віддаленої роботи, і компанії не вистачає фізичної штаб-квартири. Це найбільша біржа криптовалют у Сполучених Штатах за обсягом торгів. Компанію заснували в 2012 році Браян Армстронг і Фред Ерсам. У травні 2020 року Coinbase оголосила, що закриває свою штаб-квартиру в Сан-

Франциско, штат Каліфорнія, і змінить роботу на дистанційну, що є частиною хвилі кількох великих технологічних компаній, які закривають штаб-квартири в Сан-Франциско через пандемію COVID-19. Coinbase безкоштовно пропонує гаманець доларів США та розміщений гаманець криптовалюти. Це означає, що долар США та криптовалюта безкоштовні для клієнта.

Coinbase не стягує плату за переказ криптовалюти з одного гаманця Coinbase на інший. Також бере та сплачує комісії за мережеві транзакції, такі як комісії майнерів, за транзакції в криптовалютних мережах (тобто перекази криптовалюти з платформи). За ці транзакції Coinbase стягуватиме з клієнта комісію. Ринкова капіталізація складає 75 мільярдів доларів. Обсяг торгів $2,475,615,360 USD (24 години).

Coinbase Pro пропонує два різних API, які мають керування обліковими записами та замовленнями, а також загальнодоступні ринкові дані: REST API та FIX API. REST API містить стандартні кінцеві точки, які використовуються багатьма веб-службами.

FIX API використовує FIX (Financial Information eXchange), стандартний протокол, який можна використовувати для введення замовлень, подання запитів на скасування та отримання заповнень. Користувачі FIX API зазвичай мають наявне програмне забезпечення, яке використовує FIX для керування замовленнями [9].

Можливості:

- Створювати гаманці та адреси біткойн, біткойн кеш, лайткойн і ефіріум
- Купувати та продавати та надсилайте/отримуйте біткойн, біткойн-кеш, лайткойн та ефіріум.
- Надійно зберігати біткойн, біткойн-кеш, лайткойн і ефіріум
- Отримувати інформацію про ціни в реальному часі або за попередні періоди
- Отримуйте сповіщення про надходження платежів

- Різноманітні клієнтські бібліотеки та мобільні SDK.

1.3.2 Біржа Binance

Binance - це криптовалютна біржа, яка є найбільшою біржею в світі за щоденним обсягом торгів криптовалютами. Вона була заснована в 2017 році і зареєстрована на Каймановых островах. Була заснована Чанпеном Чжао, розробником, який раніше створював програмне забезпечення для високочастотної торгівлі. Binance спочатку базувалася в Китаї, але пізніше перенесла свою штаб-квартиру з Китаю незадовго до того, як уряд Китаю ввів правила щодо торгівлі криптовалютою. Ринкова капіталізація складає 80 мільярдів доларів. Обсяг торгів $66,945,744,059 USD (24 години) [10].

1.3.3 Біржа Kraken

Kraken - американська біржа та банк криптовалют, заснований у 2011 році. Це була одна з перших біткойн-бірж, зареєстрованих у Bloomberg Terminal, і, як повідомляється, станом на середину літа 2022 року оцінюється в 10,8 мільярдів доларів США. Станом на 2020 рік дохід Kraken склав 1,1 мільярда доларів США. Kraken доступний для мешканців 48 штатів США та 176 країн і містить список 40 криптовалют, доступних для торгівлі. Обсяг торгів $1,896,370,280 USD (24 години) [11].

1.3.4 Біржа Bitfinex

Bitfinex - це криптовалютна біржа, яка належить і управляється компанією iFinex Inc, зареєстрованою на Британських Віргінських островах. Гроші клієнтів були вкрадені або втрачені в кількох інцидентах, і вони не змогли забезпечити нормальні банківські відносини.

Дослідження показують, що маніпулювання цінами на біткойни на Bitfinex

спричинило приблизно половину зростання ціни на біткойни наприкінці 2017 року. Ринкова капіталізація складає 60 мільярдів доларів. Обсяг торгів $1,044,323,211 USD (24 години).

Bitfinex API пропонує повний набір функцій для взаємодії з нашою платформою, що дозволяє користувачам створювати повністю настроюваний досвід взаємодії з нашою платформою.

Щоб уникнути затримки мережі, користувачі WebSocket API отримують знімок даних кожного разу, коли відкривається з'єднання, і їм потрібно буде оновити ці вихідні дані локально за допомогою оновлень, надісланих через відкритий канал. API постійно оновлюється, щоб включити найновіші функції платформи. Також офіційно підтримує Python, NodeJS, Ruby та Golang, і є офіційні бібліотеки, доступні для кожної з цих мов. Їх можна знайти на сторінці бібліотек з відкритим кодом [12].

1.3.5 Біржа Bitstamp

Bitstamp - це біржа криптовалют, розташована в Люксембурзі. Вона дозволяє торгувати між фіатною валютою, біткойнами та іншими криптовалютами. Він дозволяє вносити та виводити кошти в доларах США, євро, фунтах стерлінгів, біткойнах, ALGO, XRP, ефірі, лайткойнах, біткойнах, XLM, Link, мережі OMG, монетах USD або PAX.

Компанію було засновано як орієнтовану на Європу альтернативу домінуючій на той час біткойн-біржі Mt. Gox. Хоча компанія торгує в доларах США, вона приймає безкоштовні грошові депозити лише через Єдину зону платежів у євро Європейського Союзу, механізм для переказу грошей між європейськими банківськими рахунками. Ринкова капіталізація складає 883 мільйонів доларів. Обсяг торгів $324,380,000 USD (24 години).

Інтерфейс прикладного програмування Bitstamp (API) дозволяє нашим

клієнтам отримувати доступ до своїх облікових записів і контролювати їх за допомогою написаного на замовлення програмного забезпечення. Bitstamp API підтримує такі мови, як Python, Java, C++ [13].

Після проаналізованих даних, було зроблено таблиці з необхідними даними, для того щоб вирішити, кращу API для використання, як джерело даних.

Таблиця 1

Coinbase API обмеженння

CoinBasePro	
MAX_REQ(DAY)	Historical rates should not be polled frequently
MAX_HISTORY(BTC)	From 05.2016
5M	1 req – 300 OHLC
1H	1 req – 300 OHLC
4H	1 req – 300 OHLC
1D	1 req – 300 OHLC

Таблиця 2

Binance API обмеженння

Binance	
MAX_REQ(DAY)	1,728,000 per day
MAX_HISTORY(BTC)	From 08.2017
5M	1 req – 1000 OHLC
1H	1 req – 1000 OHLC
4H	1 req – 1000 OHLC
1D	1 req – 1000 OHLC

Таблиця 3

Kraken API обмеженння

Kraken	
MAX_REQ(DAY)	NO LIMIT
MAX_HISTORY(BTC)	From 09.2019
5M	1 req – last 2 days
1H	1 req – last month
4H	1 req – last 4 month
1D	1 req – last 18 month

Таблиця 4

Bitfinex API обмеженння

Bitfinex	
MAX_REQ(DAY)	7,776,000
MAX_HISTORY(BTC)	From 2013
5M	1 req – 10000 OHLC (last month)
1H	1 req – 10000 OHLC (from last year)
4H	1 req – 10000 OHLC (from 2016)
1D	1 req – 10000 OHLC (from 2013)

Таблиця 5

Bitstamp API обмеженння

Bitstamp	
MAX_REQ(DAY)	1,152,000
MAX_HISTORY(BTC)	From 2015
5M	1 req – 10000 OHLC
1H	1 req – 10000 OHLC
4H	1 req – 10000 OHLC
1D	1 req – 10000 OHLC

РОЗДІЛ 2
МАТЕМАТИЧНІ ОСНОВИ ДЛЯ ЗАДАЧІ ПРОГНОЗУВАННЯ КУРСУ КРИПТОВАЛЮТ

2.1 Класифікація нейронних мереж

Класифікація - це процес класифікації заданого набору даних у класи. Його можна виконувати як на структурованих, так і на неструктурованих даних. Процес починається з прогнозування класу заданих точок даних. Класи часто називають цільовими, мітками або категоріями. Класифікаційне прогностичне моделювання - це завдання апроксимації функції відображення від вхідних змінних до дискретних вихідних змінних. Основна мета - визначити, до якого класу/категорії належатимуть нові дані [27].

Спробуємо зрозуміти це на простому прикладі. Виявлення серцевих захворювань можна ідентифікувати як проблему класифікації, це двійкова класифікація, оскільки може бути лише два класи, тобто є серцева хвороба або не має серцевої хвороби. У цьому випадку класифікатору потрібні навчальні дані, щоб зрозуміти, як ці вхідні змінні пов’язані з класом. І як тільки класифікатор буде навчений точно, його можна буде використовувати для виявлення захворювання серця у конкретного пацієнта чи ні [14].

Класифікаційні термінології в машинному навчанні:

- **Класифікатор** – це алгоритм, який використовується для відображення вхідних даних у певній категорії.
- **Модель класифікації** – модель передбачає або робить висновок щодо вхідних даних, наданих для навчання, вона передбачає клас або категорію даних.
- **Характеристика** – ознака є окремою вимірюваною властивістю явища, яке спостерігається.

- **Бінарна класифікація** – це тип класифікації з двома результатами, наприклад – істинний або хибний.
- **Багатокласова класифікація** – класифікація з більш ніж двома класами, у багатокласовій класифікації кожен зразок присвоюється одній і лише одній мітці або цілі.
- **Класифікація з кількома мітками** – це тип класифікації, де кожному зразку присвоюється набір міток або цілей.
- **Навчання класифікатора** – кожен класифікатор у Sci-kit learn використовує метод fit(X, y) для підгонки моделі для навчання поїзда X і навчання мітки y.
- **Передбачити ціль** – для спостереження X без мітки метод predict(X) повертає передбачену мітку y.
- Оцінити – це в основному означає оцінку моделі, тобто звіт про класифікацію, оцінку точності тощо.

Типи учнів у класифікації:

- **Ледачі учні** – ледачі учні просто зберігають навчальні дані та чекають, поки з’являться дані тестування. Класифікація виконується з використанням найбільш пов’язаних даних у збережених навчальних даних. У них більше часу на передбачення, ніж у тих, хто навчається. Наприклад – k-найближчий сусід, міркування на основі випадків.
- **Охочі** до навчання – охочі до навчання будують модель класифікації на основі наданих навчальних даних, перш ніж отримувати дані для прогнозів. Він повинен бути в змозі прийняти єдину гіпотезу, яка працюватиме для всього простору. Через це вони займають багато часу на навчання і менше часу на прогноз. Наприклад – Decision Tree, Naive Bayes, Штучні нейронні мережі [15].

2.1.1 Нейронна мережа Logistic Regression

Це алгоритм класифікації в машинному навчанні, який використовує одну або кілька незалежних змінних для визначення результату. Результат вимірюється за допомогою дихотомічної змінної, що означає, що він матиме лише два можливі результати. Метою логістичної регресії є знаходження найкращого співвідношення між залежною змінною та набором незалежних змінних. Він кращий за інші алгоритми двійкової класифікації, як найближчий сусід, оскільки він кількісно пояснює фактори, що призводять до класифікації.

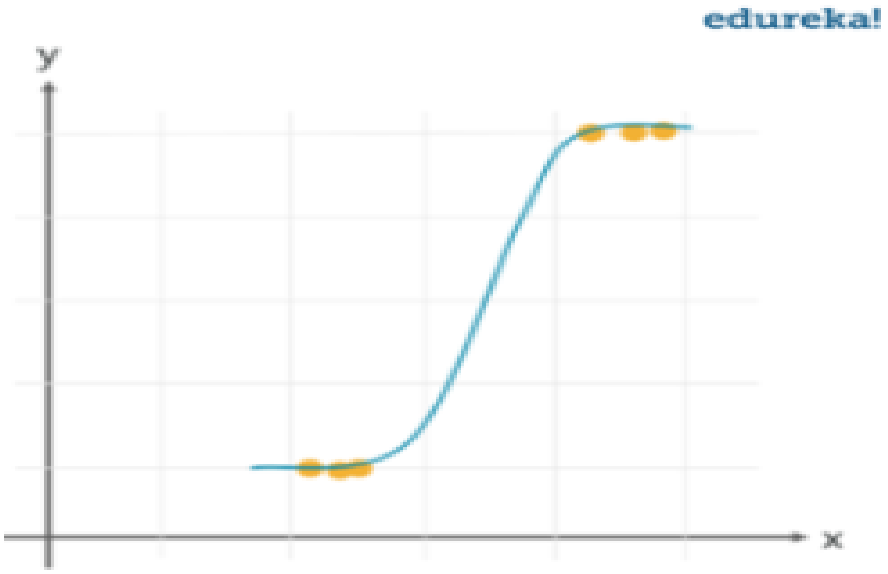

Рис.2. Графік логістчиної регресії

Логістична регресія спеціально призначена для класифікації, вона корисна для розуміння того, як набір незалежних змінних впливає на результат залежної змінної. Основним недоліком алгоритму логістичної регресії є те, що він працює лише тоді, коли прогнозована змінна є двійковою, він припускає, що дані не містять пропущених значень, і передбачає, що предиктори незалежні один від одного.

2.1.2 Нейронна мережа Naive Bayes Classifier

Це алгоритм класифікації, заснований на теоремі Байєса, який дає припущення про незалежність предикторів. Простіше кажучи, наївний класифікатор Байєса припускає, що наявність певної ознаки в класі не пов’язана з наявністю будь-якої іншої ознаки. Навіть якщо ознаки залежать одна від одної, усі ці властивості впливають на ймовірність незалежно. Наївну модель Байєса легко створити, і вона особливо корисна для порівняно великих наборів даних. Відомо, що навіть із спрощеним підходом Naive Bayes перевершує більшість методів класифікації в машинному навчанні. Нижче наведено теорему Байєса для реалізації наївної теореми Байєса.

$$P(C_i|\, x_1, x_2 \ldots, x_n) = \frac{P(x_1, x_2 \ldots, x_n|C_i).P(C_i)}{P(x_1, x_2 \ldots, x_n)} \; for\; 1 < i < k$$

Нативний класифікатор Байєса потребує невеликої кількості навчальних даних для оцінки необхідних параметрів для отримання результатів. Вони надзвичайно швидкі за своєю природою порівняно з іншими класифікаторами. Єдиним недоліком є те, що вони, як відомо, погані оцінювачі [16].

2.1.3 Нейронна мережа Stochastic Gradient Descent

Це дуже ефективний і простий підхід для підгонки лінійних моделей. Стохастичний градієнтний спуск особливо корисний, коли вибірка даних є великою. Він підтримує різні функції втрат і штрафи для класифікації. Стохастичний градієнтний спуск стосується обчислення похідної від кожного екземпляра навчальних даних і негайного обчислення оновлення.

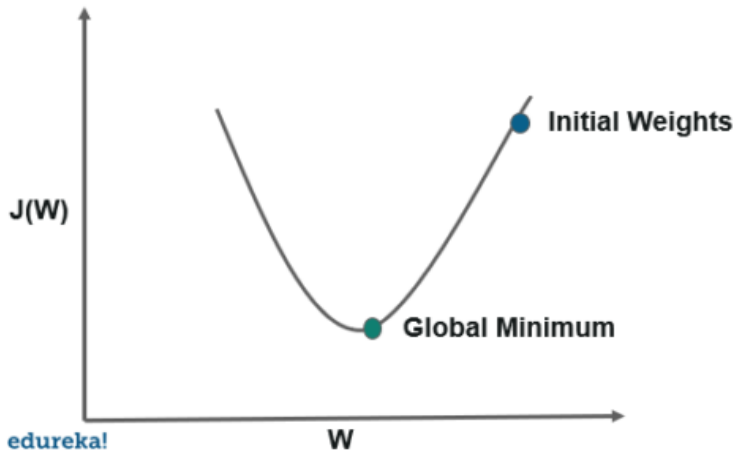

Рис.3. Графік стохатичного градієнтного спуску

Єдиною перевагою є простота впровадження та ефективність, тоді як основним недоліком стохастичного градієнтного спуску є те, що він вимагає ряду гіперпараметрів і чутливий до масштабування функцій [20].

2.1.4 Нейронна мережа K-Nearest Neighbor

Це алгоритм відкладеного навчання, який зберігає всі екземпляри, що відповідають навчальним даним, у n-вимірному просторі. Це ледачий алгоритм навчання, оскільки він не зосереджений на побудові загальної внутрішньої моделі, замість цього він працює над збереженням екземплярів навчальних даних. Класифікація обчислюється простою більшістю голосів k найближчих сусідів кожної точки. Він знаходиться під наглядом і використовує купу позначених точок і використовує їх для позначення інших точок. Щоб позначити нову точку, він переглядає позначені точки, найближчі до цієї нової точки, також відомі як її найближчі сусіди. У ньому голосують ці сусіди, тому мітка для нової точки буде міткою більшості сусідів. «k» - це кількість сусідів, яких він перевіряє.

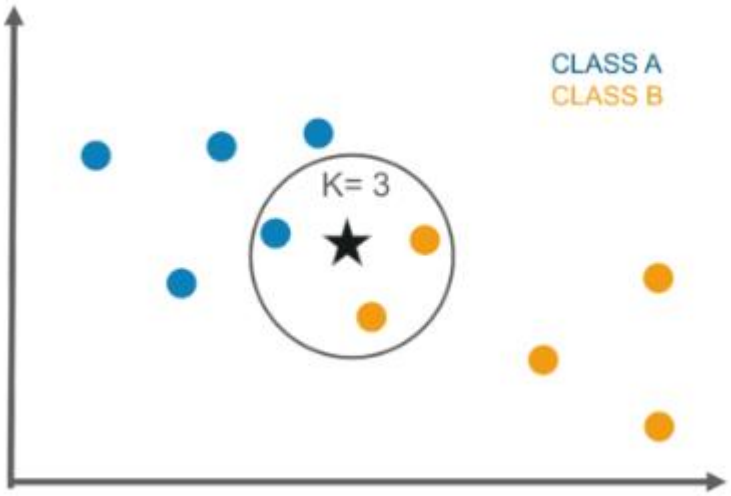

Рис.4. Графік K-Nearest Neighbor класифікації

Цей алгоритм досить простий у своїй реалізації та стійкий до зашумлених навчальних даних. Навіть якщо навчальні дані великі, це досить ефективно. Єдиним недоліком алгоритму KNN є те, що немає необхідності визначати значення K, а вартість обчислень є досить високою порівняно з іншими алгоритмами.

2.1.5 Нейронна мережа Decision Tree

Алгоритм дерева рішень будує модель класифікації у формі деревовидної структури. Він використовує правила якщо-тоді, які однаково вичерпні та взаємовиключні в класифікації. Процес продовжується, розбиваючи дані на менші структури та зрештою пов’язуючи їх із поступовим деревом рішень. Остаточна структура виглядає як дерево з вузлами та листям. Правила вивчаються послідовно з використанням навчальних даних по одному. Кожного разу, коли вивчається правило, кортежі, що покривають правила, видаляються. Процес триває на навчальному наборі, доки не буде досягнуто точки завершення.

Дерево побудовано за рекурсивним підходом «розділяй і володарюй» зверху вниз. Вузол рішення матиме дві або більше гілок, а аркуш представляє класифікацію або рішення. Найвищий вузол у дереві рішень, який відповідає

найкращому предиктору, називається кореневим вузлом, і найкраще в дереві рішень полягає в тому, що воно може обробляти як категоричні і числові дані.

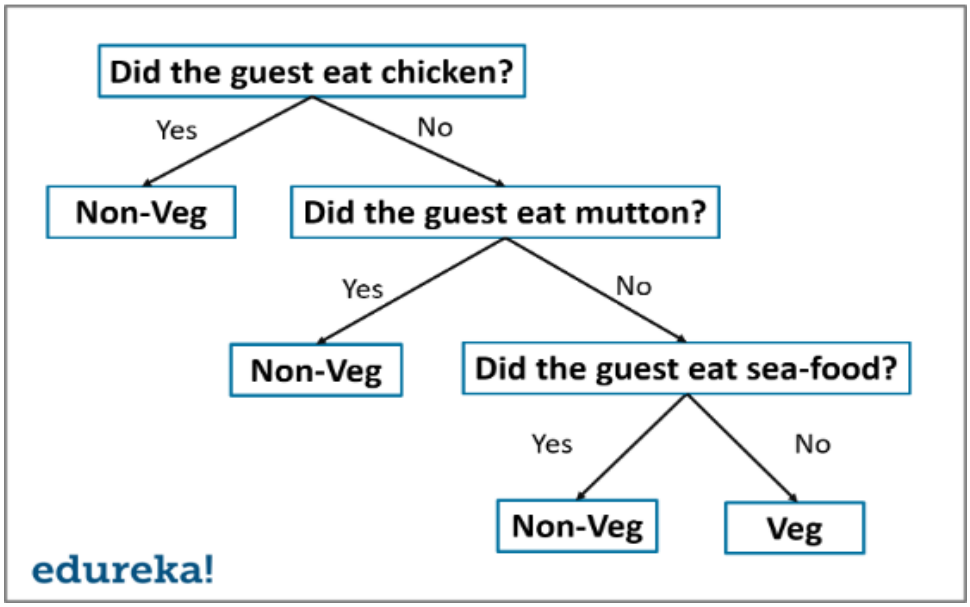

Рис.5. Приклад використання дерева рішень

Дерево рішень дає перевагу простоти для розуміння та візуалізації, воно також вимагає дуже невеликої підготовки даних. Недоліком дерева рішень є те, що воно може створювати складні дерева, які можуть ефективно класифікувати бот. Вони можуть бути досить нестабільними, тому що навіть проста зміна даних може перешкодити всій структурі дерева рішень [17].

2.1.6 Нейронна мережа Random Forest

Random Forest - це метод ансамблевого навчання для класифікації, регресії тощо. Він функціонує шляхом побудови безлічі дерев рішень під час навчання та виводить клас, який є режимом класів або класифікації, або середнього прогнозу (регресії) окремі дерева. Випадковий ліс - це метаоцінювач, який підбирає кілька дерев у різних підвибірках наборів даних, а потім використовує середнє значення для підвищення точності прогностичного характеру моделі. Розмір підвибірки

завжди такий самий, як і вихідний розмір вхідних даних, але вибірки часто складаються із замінами.

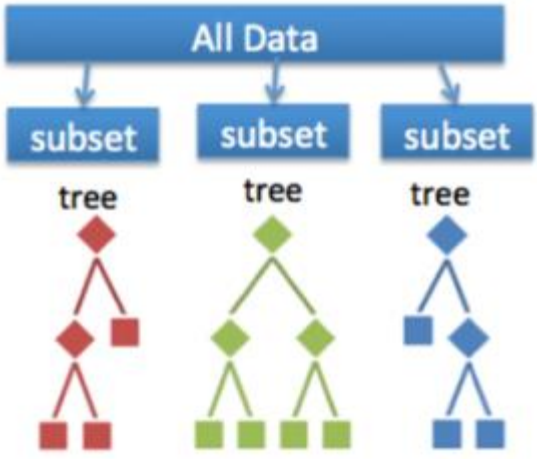

Рис.6. Приклад використання Random Forest

Перевага випадкового лісу полягає в тому, що він більш точний, ніж дерева рішень, завдяки зменшенню надмірної підгонки. Єдиним недоліком класифікаторів випадкового лісу є те, що вони досить складні у реалізації та стають досить повільними в прогнозі в реальному часі.

2.1.7 Нейронна мережа Artificial Neural Networks

Нейронна мережа складається з нейронів, які розташовані шарами, вони приймають деякий вхідний вектор і перетворюють його на вихід. Цей процес передбачає, що кожен нейрон приймає вхідні дані та застосовує до нього функцію, яка часто є нелінійною функцією, а потім передає вихідні дані на наступний рівень. Загалом передбачається, що мережа має прямий зв’язок, що означає, що пристрій або нейрон передає вихідні дані на наступний рівень, але жодного зворотного зв’язку з попереднім рівнем немає. Зважування застосовуються до сигналів, що проходять від одного рівня до іншого, і це зважування, які налаштовуються на етапі навчання, щоб адаптувати нейронну мережу для будь-якої постановки проблеми.

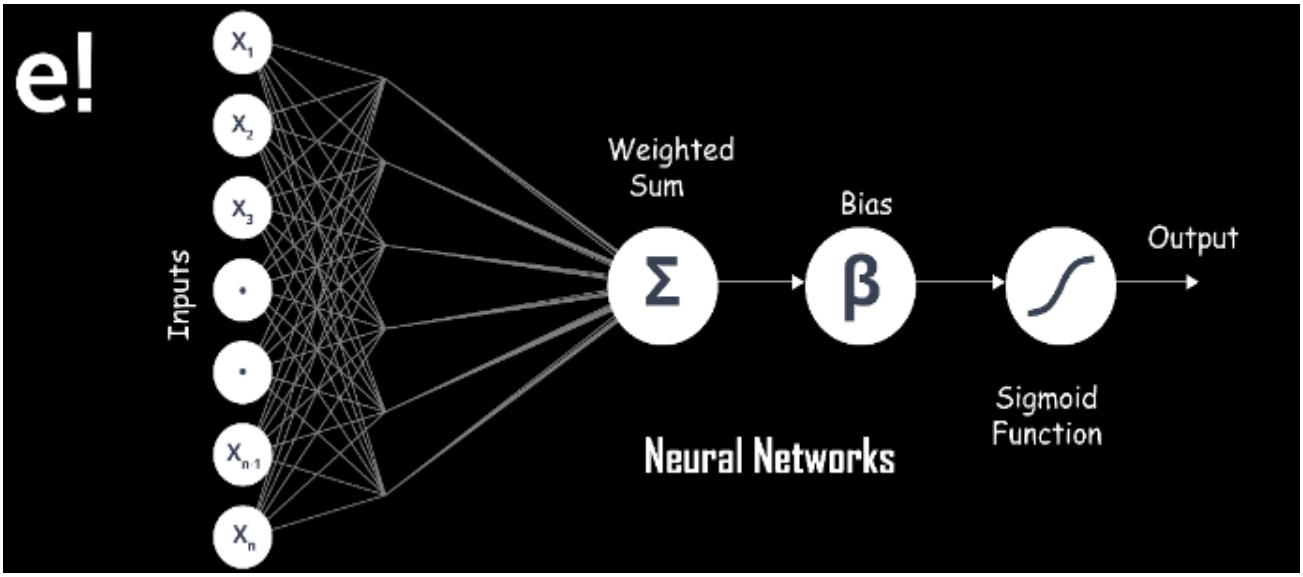

Рис.7. График принципу роботи Artificial Neural Networks

Він має високу толерантність до зашумлених даних і здатний класифікувати ненавчені шаблони, він краще працює з безперервними вхідними та вихідними сигналами. Недоліком штучних нейронних мереж є погана інтерпретація порівняно з іншими моделями.

2.1.8 Нейронна мережа Support Vector Machine

Машина опорних векторів - це класифікатор, який представляє навчальні дані як точки в просторі, розділені на категорії максимально широким проміжком. Потім нові точки додаються до простору, передбачаючи, до якої категорії вони належать і до якого простору належатимуть.

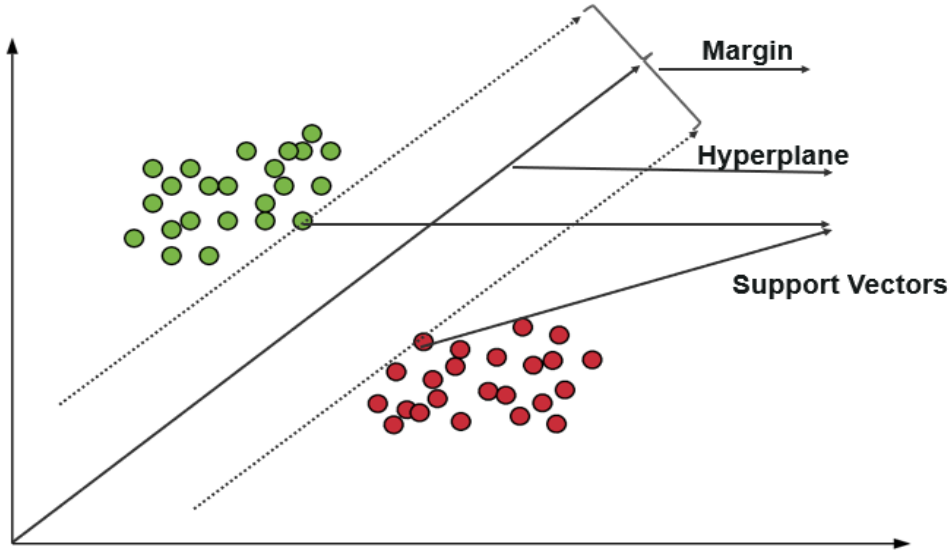

Рис.8. Принцип роботи Support Vector Machine

Він використовує підмножину тренувальних точок у функції прийняття рішень, що робить його ефективним у пам’яті та дуже ефективним у просторах великої розмірності. Єдиним недоліком машини опорних векторів є те, що алгоритм безпосередньо не забезпечує оцінки ймовірності [18].

2.2 Мережа довго короткострокової пам’яті

Нейронна мережа LSTM - це особливий вид мережі RNN, яка часто використовується для обробки довгих вхідних даних. LSTM не обмежується одним входом, але також може обробляти цілі вхідні послідовності. Зазвичай LSTM структурований із чотирма воротами, такими як забуття, вхід, стан комірки та вихід. Кожен шлюз має окрему дію, де стан комірки зберігає повну інформацію про вхідні послідовності, а інші використовуються для керування вхідними та вихідними діями. На рисунку показана структура основного блоку LSTM [19].

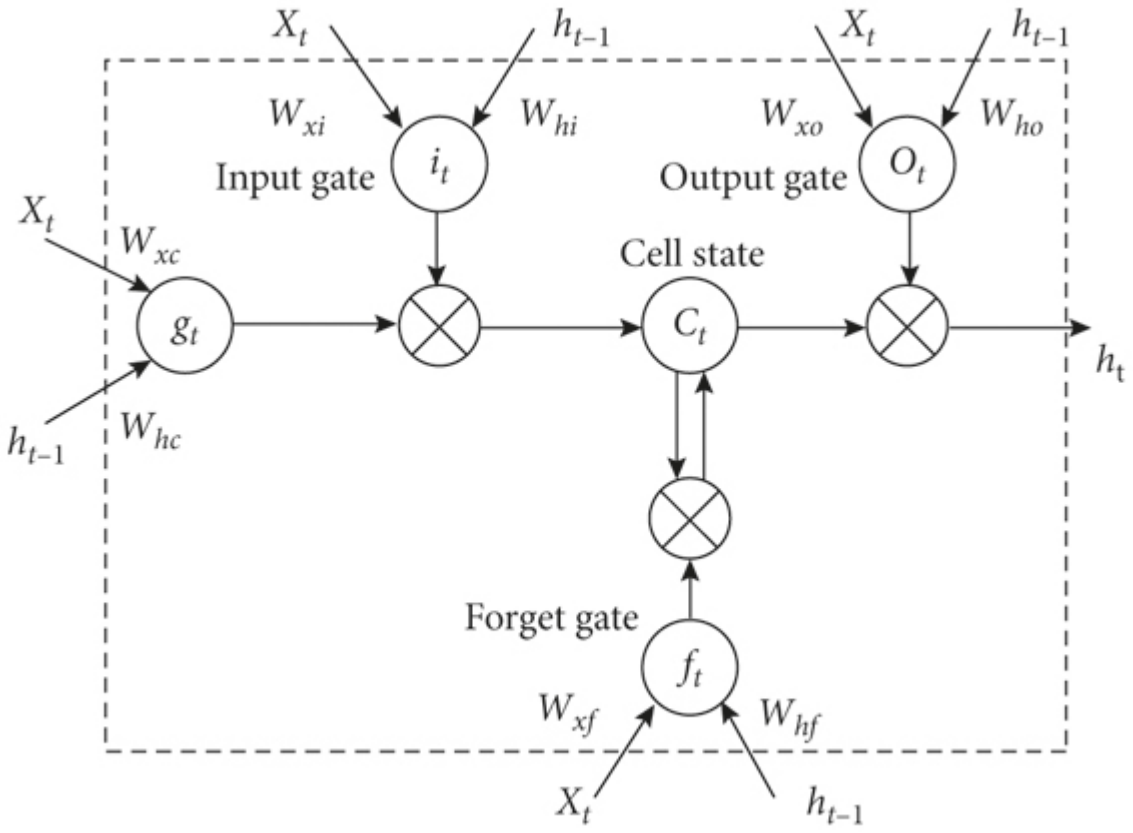

Рис.9. Внутрішня структура блоку LSTM.

На самому початку обробка починається з шлюзу забуття, щоб визначити, яку інформацію потрібно відкинути (або зберегти в) стані комірки. Ворота забуття в стані комірки c_{t-1} можна виразити наступним рівнянням (15), де h_{t-1} - прихований стан, а x_t - поточний вхід. Вихід (0 або 1) вентиля забуття створюється за допомогою сигмоїдної функції. Якщо результат шлюзу забуття дорівнює 1, ми зберігаємо дані в стані клітинки; інакше ми відкидаємо дані. Вхідний вентиль визначає, яке значення стану комірки слід оновлювати, коли з'являються нові дані. Тепер за допомогою функції tanh створюється значення-кандидат для стану комірки. Тепер ми оновлюємо старий стан клітинки c_{t-1} на c_t Відфільтрована версія стану комірки буде виведена через функцію sigmoid, і вага також буде оновлено.

Визнаючи переваги LSTM, саме ця модель мережі була вибрана для виявлення, прогнозування, виправлення та класифікації помилок у вихідних кодах [21].

2.3 Рекурентні нейронні мережі

Рекурентні нейронні мережі (RNN) - це клас штучних нейронних мереж, у яких зв’язки між вузлами можуть створювати цикл, дозволяючи виводу з деяких вузлів впливати на наступний вхід до тих самих вузлів. Це дозволяє йому демонструвати тимчасову динамічну поведінку. Похідні від нейронних мереж прямого зв’язку , RNN можуть використовувати свій внутрішній стан (пам’ять) для обробки послідовностей вхідних даних змінної довжини. Це робить їх застосовними для таких завдань, як несегментоване розпізнавання пов’язаного рукописного тексту або розпізнавання мовлення. Теоретично рекурентні нейронні мережі завершений і може запускати довільні програми для обробки довільних послідовностей вхідних даних.

Термін «рекурентна нейронна мережа» використовується для позначення класу мереж із нескінченною імпульсною характеристикою , тоді як « згорточна нейронна мережа » відноситься до класу кінцевої імпульсної характеристики. Обидва класи мереж демонструють часову динамічну поведінку. Скінченна імпульсна рекурентна мережа - це орієнтований ациклічний граф , який можна розгорнути та замінити прямою нейронною мережею, тоді як нескінченна імпульсна рекурентна мережа - це спрямований циклічний граф , який не можна розгорнути [22].

Як кінцеві імпульсні, так і нескінченні імпульсні рекурентні мережі можуть мати додаткові збережені стани, і зберігання може бути під прямим контролем нейронної мережі. Пам’ять також можна замінити іншою мережею чи графіком, якщо це включає часові затримки або має петлі зворотного зв’язку. Такі

контрольовані стани називаються стробованим станом або стробованою пам’яттю і є частиною довготривалих мереж короткочасної пам’яті (LSTM) і стробованих рекурентних блоків. Це також називається нейронною мережею зворотного зв’язку (FNN). Працює за принципом збереження виходу певного шару та передачі його назад на вхід, щоб передбачити вихід шару.

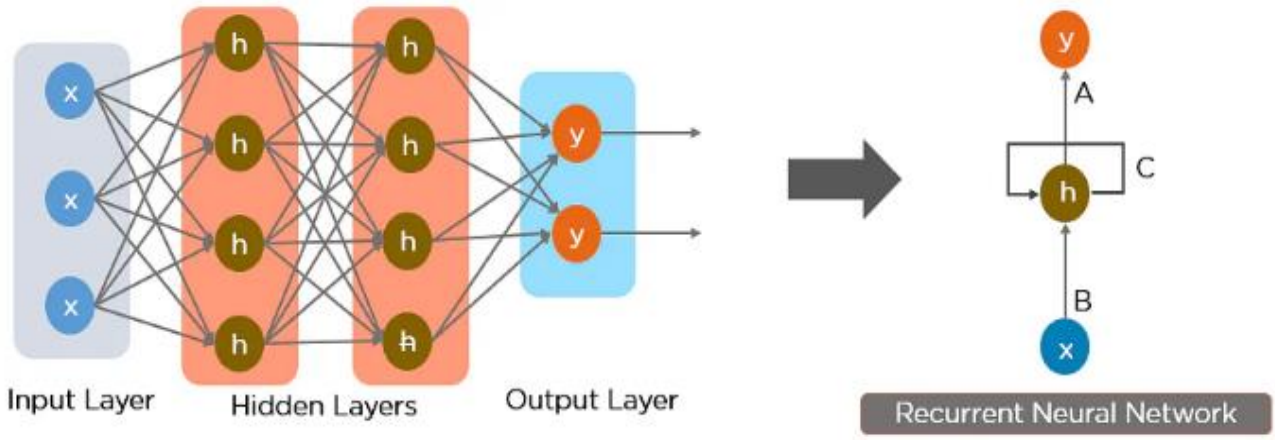

Рис.10. Проста рекурентна нейронна мережа

Вузли на різних рівнях нейронної мережі стискаються, щоб утворити один шар рекурентних нейронних мереж. A, B і C - параметри мережі. Тут «x» - вхідний шар, «h» - прихований шар, а «y» - вихідний шар. A, B і C - параметри мережі, які використовуються для покращення виходу моделі. У будь-який момент часу t поточний вхідний сигнал є комбінацією вхідного сигналу в x(t) і x(t-1). Вихідні дані в будь-який момент часу повертаються в мережу для покращення результатів [23].

2.4 Регресивна модель нейронної мережі

Регресійні моделі використовуються для прогнозування постійного значення. Прогнозування цін на будинок з урахуванням таких характеристик будинку, як розмір, ціна тощо, є одним із поширених прикладів регресії.

Проста лінійна регресія - це один з найпоширеніших і найцікавіших видів техніки регресії. Тут ми прогнозуємо цільову змінну Y на основі вхідної змінної X. Між цільовою змінною та предиктором має існувати лінійний зв'язок, тому й називається лінійна регресія.

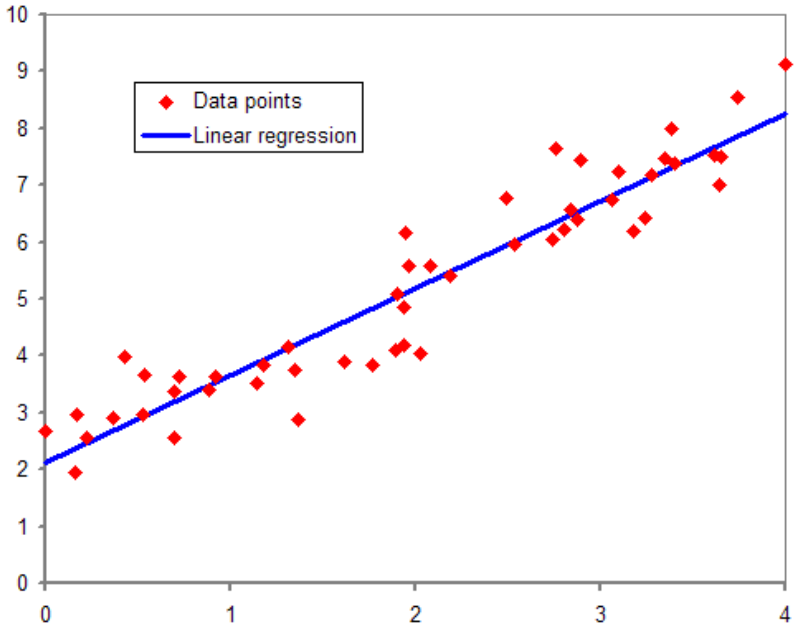

Рис.11. Графік простої лінійної регресії

На рис.15 червоні точки – це точки даних, а синя лінія – прогнозована лінія для даних навчання. Щоб отримати прогнозоване значення, ці точки даних проектуються на лінію.

У поліноміальній регресії ми перетворюємо вихідні ознаки в поліноміальні ознаки заданого ступеня, а потім застосовуємо до них лінійну регресію. Розглянемо лінійну модель, наведену вище, Y = aX + b перетворюється на щось на зразок Y = aX + b + cX^2 . Це все ще лінійна модель, але крива тепер квадратична, а не лінія. Scikit-Learn надає клас PolynomialFeatures для трансформації функцій. Якщо ми збільшимо ступінь до дуже високого значення, крива стане переобладнаною, оскільки вона також дізнається шум у даних.

У SVR ми визначаємо гіперплощину з максимальним запасом так, що максимальна кількість точок даних знаходиться в межах цього запасу. SVR майже подібні до алгоритму класифікації SVM. Ми детально обговоримо алгоритм SVM у моїй наступній статті [24].

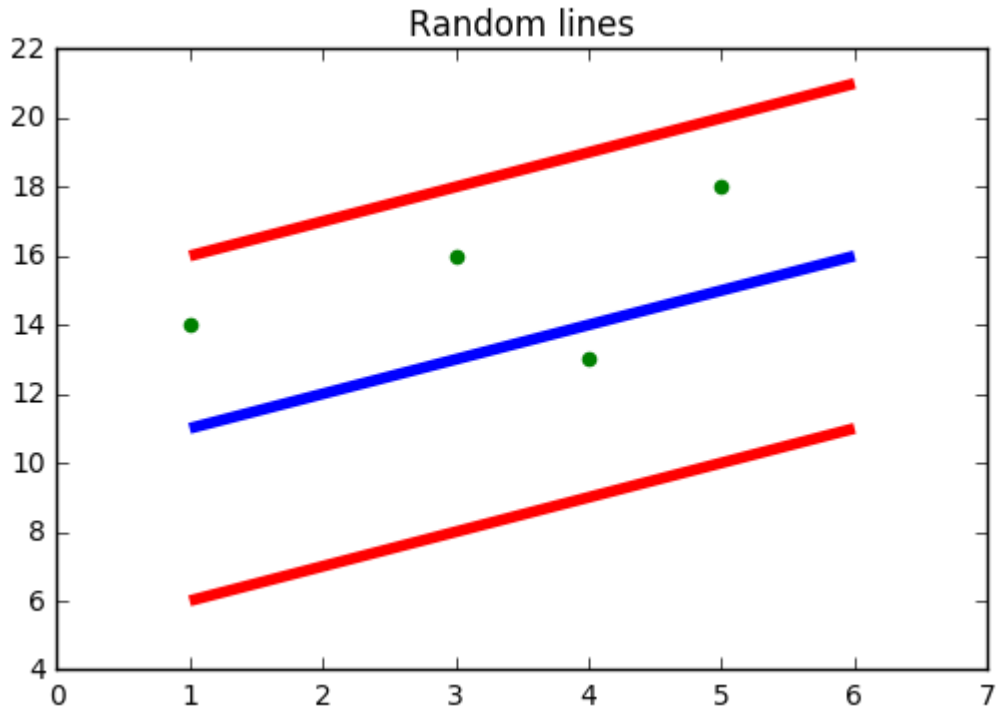

Рис.12. Принцип роботи SVR

Замість мінімізації частоти помилок, як у простій лінійній регресії, ми намагаємося підігнати помилку до певного порогу. Наша мета в SVR полягає в тому, щоб в основному розглянути точки, які знаходяться в межах маржі. Наша найкраща лінія - це гіперплощина, яка має максимальну кількість точок.

Дерева рішень можна використовувати як для класифікації, так і для регресії. У деревах рішень на кожному рівні нам потрібно визначити атрибут розщеплення. У разі регресії алгоритм ID3 може бути використаний для ідентифікації вузла розщеплення шляхом зменшення стандартного відхилення (у класифікації використовується посилення інформації).

Дерево рішень будується шляхом поділу даних на підмножини, що містять екземпляри з подібними значеннями (однорідними). Стандартне відхилення використовується для розрахунку однорідності числової вибірки. Якщо чисельна вибірка повністю однорідна, її стандартне відхилення дорівнює нулю.

Нижче коротко описано кроки для пошуку вузла розщеплення:

1) Обчисліть стандартне відхилення цільової змінної за допомогою наведеної нижче формули.

$$\textbf{Standard Deviation} = S = \sqrt{\frac{\sum(x-\bar{x})^2}{n}}$$

2) Розділіть набір даних на різні атрибути та обчисліть стандартне відхилення для кожної гілки (стандартне відхилення для цілі та предиктора). Це значення віднімається від стандартного відхилення перед поділом. Результатом є зменшення стандартного відхилення.

$$SDR(T,X) = S(T) - S(T,X)$$

3) Атрибут із найбільшим зменшенням стандартного відхилення вибирається як вузол розбиття.

4) Набір даних розділено на основі значень вибраного атрибута. Цей процес виконується рекурсивно на нелистових гілках, доки не буде оброблено всі дані.

Щоб уникнути переобладнання, використовується коефіцієнт відхилення (CV), який визначає, коли припинити розгалуження. Нарешті, середнє значення кожної гілки призначається відповідному листовому вузлу (у регресії береться середнє значення, тоді як у режимі класифікації береться листові вузли) [25].

Random forest - це ансамблевий підхід, у якому ми беремо до уваги передбачення кількох дерев регресії рішень.

1) Виберіть K випадкових точок

2) Визначте n, де n – кількість регресорів дерева рішень, які потрібно створити. Повторіть кроки 1 і 2, щоб створити кілька дерев регресії.

3) Середнє значення кожної гілки призначається листовому вузлу в кожному дереві рішень.

4) Щоб передбачити вихід для змінної, до уваги береться середнє значення всіх прогнозів усіх дерев рішень.

Random forest запобігає переобладнанню (що часто зустрічається в деревах рішень), створюючи випадкові підмножини функцій і будуючи менші дерева, використовуючи ці підмножини [26].

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ ЗБЕРЕЖННЯ ТА АНАЛІЗУ КРИПТОДАННИХ

3.1. Принцип роботи додатку

В результаті дослідження в розділах 1 та 2 було вирішено використовувати серверну архітектуру для функціонування додатку. Додаток вирішено написати на мультиплатформерній мові Java.

Наступним кроком були проаналізовані, які саме бізнес процеси буде виконувати додаток і визначено, що спершу ми маємо отримати історію криптовалютних пар з їх даними з біржі торгівлі за маркетом та p2p. Ці дані додаток трансформує та буде зберігати в базі даних. Для більш точного прогнозування програмую, було обрано різні дані, які будуть використані в ході навчання нейронної мережі.

Перший тип даних має найважливішу вагу для покращення прогнозу моделі це дані продажу через біржу, тобто маркет. Бо саме через продаж маркетом відбувалися найбільше операцій купівлі/продажу криптовалюти. Розділимо їх також за принципом на два типи за часом: архівні дані за всі роки торгівлі на біржі, та поточні дані, які отримуються під час роботи веб додатка.

Другий тип даних - це інформація про купівлю/продаж через сервіс P2P, де люди напряму торгують між собою.

Після збереження даних, вони будуть використані нейронною мережею, для подальшого навчання, та прогнозування динаміки зміни ціни криптовалюти.

Процес отримання даних буде виглядати так, що користувач має зробити запит, з обраними параметрами, а саме: криптовалютну пару, за якої бажає отримати дані аналізу нейронної мережі для передбачення зміни ціни наступної години, вибрану біржу для якої потрібен аналіз.

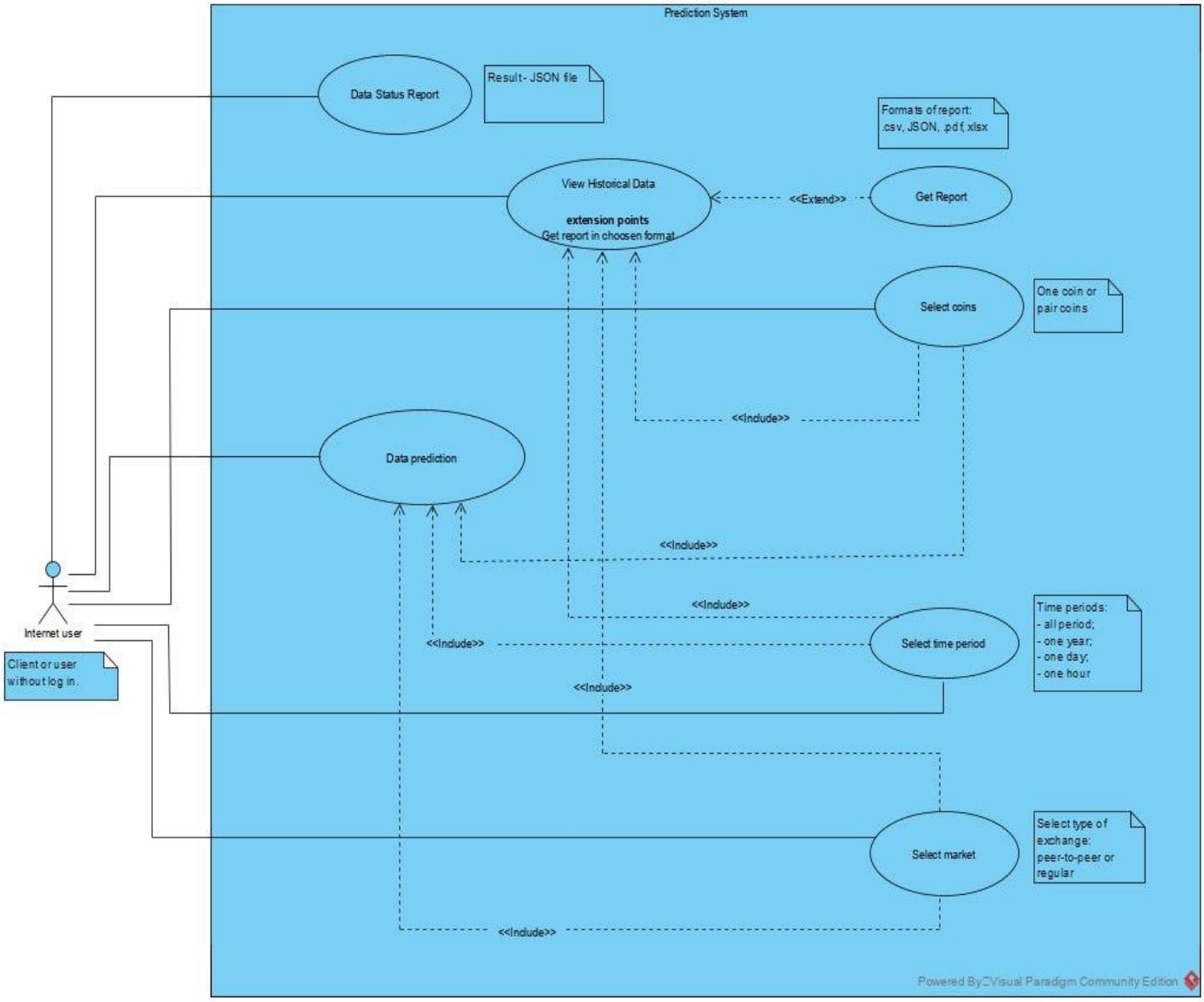

Рис.13. Схематичне зображення принципу роботи додатку

3.2. Архітектура веб-додатку

Для вирішення даної задачі був зроблений аналіз та прийняте рішення, що додаток потрібно розділити на дві частини.

Перша частина веб додатку – Extractor. Ця частина відповідає за отримання та збереження даних які надходять з API біржі. Для цього був імплементован модуль для зв’язку з біржою, URL API якої знаходиться у ресурсних файлах.

Приклад отриманого JSON файлу з API Binance для BNBBTC пари:

```
{
"symbol": "BNBBTC",
"priceChange": "-94.99999800",
"priceChangePercent": "-95.960",
"weightedAvgPrice": "0.29628482",
"prevClosePrice": "0.10002000",
"lastPrice": "4.00000200",
"lastQty": "200.00000000",
"bidPrice": "4.00000000",
"askPrice": "4.00000200",
"openPrice": "99.00000000",
"highPrice": "100.00000000",
"lowPrice": "0.10000000",
"volume": "8913.30000000",
"quoteVolume": "15.30000000",
"openTime": 1499783499040,
"closeTime": 1499869899040,
"firstId": 28385, // First tradeId
"lastId": 28460, // Last tradeId
"count": 76 // Trade count
}
```

Для аналізу було визначено, що потрібно лише частина даних, які надаються, а саме такі поля як: priceChange, openPrice, highPrice, lowPrice, openTime, closeTime.

При запуску веб-додатка extractor, програма починає збирати всі архівні дані за вказаним переліком криптовалют, якщо їх нема у БД та запускає процес слідкування торгів. Після того як в біржі з’являться нові дані про трейди, програма у відведений час, подивиться чи є нові дані та добавить у БД також.

Друга частина веб-додатку це – Predictor. Підготовлені дані у БД зчитаються аналізатором, конфігурується нейрона мережа з налаштувань у коді та запускається довгий процес навчання з декількома епохами для всіх криптовалютних пар, що містяться у базі даних. Після того, як процес буде закінчено, програма перейде у стан донавчання з нових даних, що будуть надходити вже при виконані програми.

Якщо модель готова до роботи, користувач може подати запит до серверу з необхідними параметрами, про бажання отримати прогнозування зміни динаміки ціни на ту чи іншу криптовалюту протягом години. Результатом буде дані відсотка точності прогнозування, в якому напрямку буде зміна ціни криптовалюти, та на скільки приблизно буде змінено ціна.

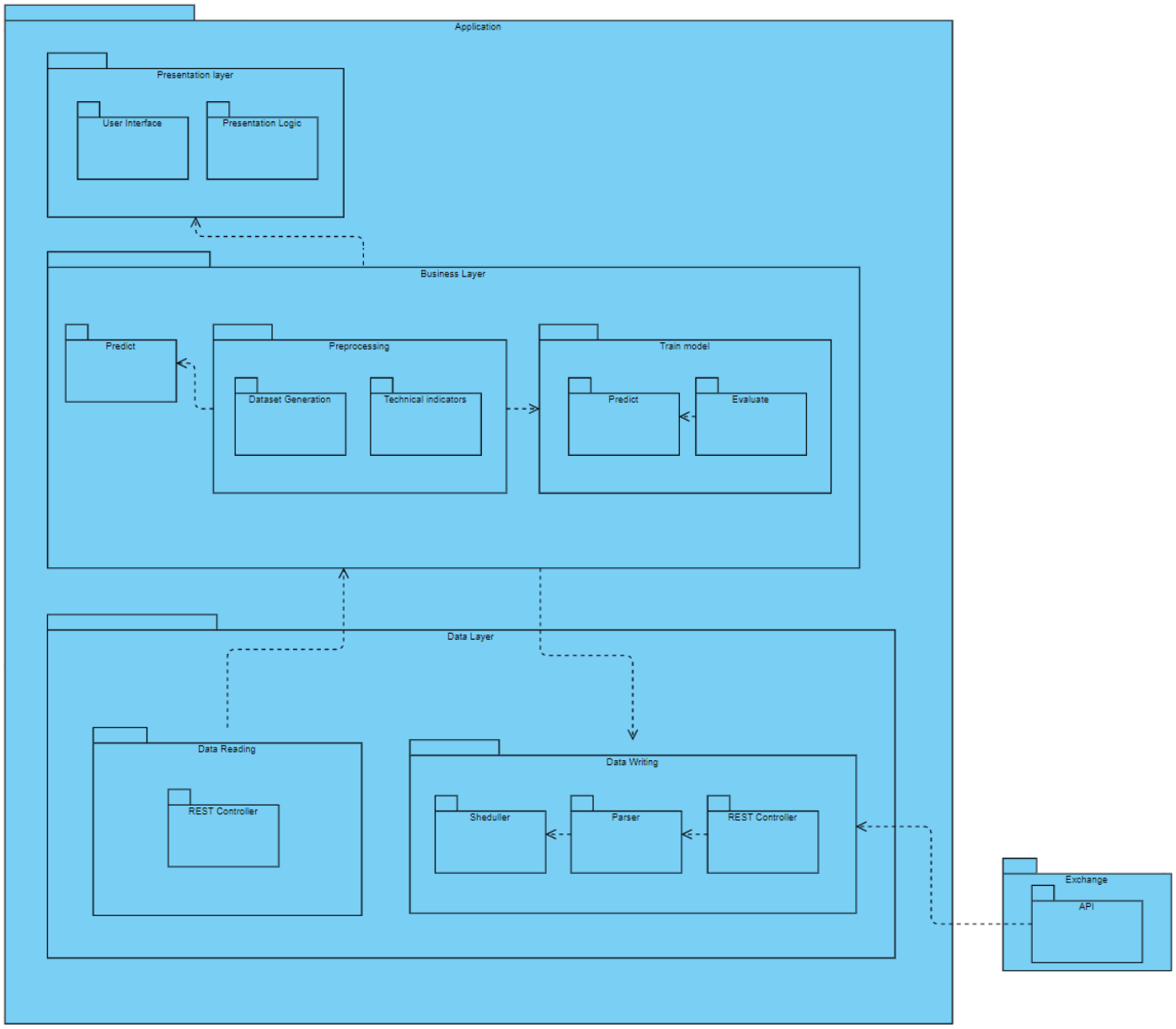

Рис.14. Загальна архітектура веб-додатку

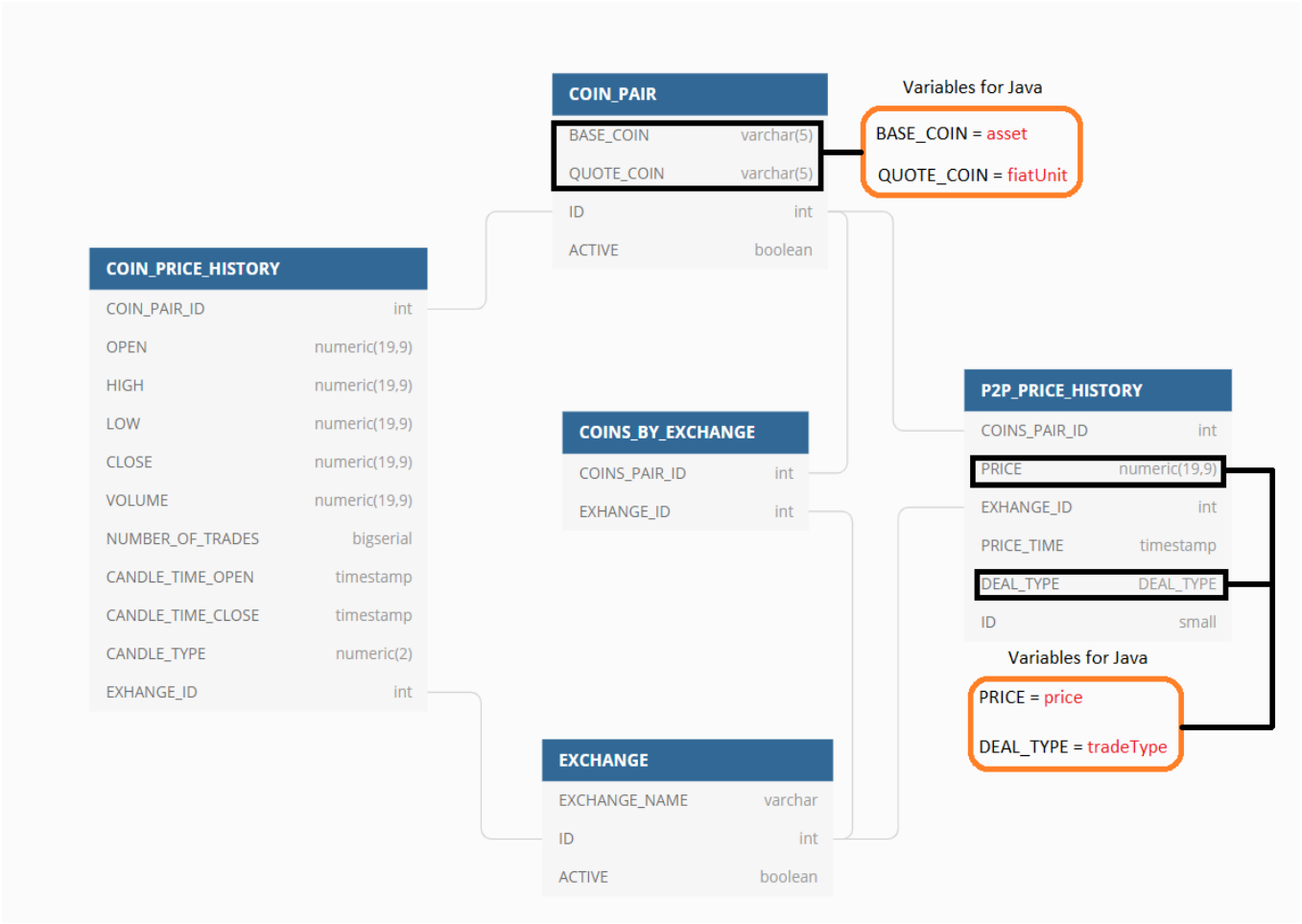

Рис.15. Структура бази даних

3.3 Методи покращення прогнозування

Для покращення результату прогнозування нейронною мережею, було виділено необхідні параметри, які змінювали показник точності моделі до кращого результату. Методи можливо розділити на дві групи, в першу входять ті що будуть впливати на дані, які будуть надходити для навчання моделі. Друга група методів, це ті які покращать саме модель, через налаштування [29].

Індикатор - функція, побудована на значеннях статистичних показників торгів (ціни, обсяг торгів тощо), аналіз поведінки якої покликаний відповісти питанням зміниться чи збережеться поточна тенденція над ринком. На основі

аналізу технічних індикаторів трейдери, прихильники технічного аналізу, приймають рішення про відкриття (розширення) або закриття (скорочення) позицій. У цьому випадку технічні індикатори зазвичай застосовуються у вигляді графіків, накладених або поєднаних з графіками цін/об'ємів інструментів, що торгуються. Крім того, технічні індикатори тією чи іншою мірою використовуються механічними торговими системами при алгоритмічній торгівлі [29].

Нижче представлений список індикаторів, які використані для надання моделі, як дані для навчання:

- OBV - один із найпростіших технічних індикаторів, заснований на ціні та обсязі. Індикатор OBV являє собою кумулятивну ковзну середню обсягу торгів, взятого зі знаком плюс у разі зростаючого ринку і зі знаком мінус у разі падаючого.

$$OBV_t = OBV_{t-i} + \begin{cases} volume_t & \text{if } close_t > close_{t-i} \\ 0 & \text{if } close_t = close_{t-i} \\ -volume_t & \text{if } close_t < close_{t-i} \end{cases},$$

де OBV_t - значення індикатора OBV в момент t, OBV_{t-i} - попереднє значення індикатора, $volume_t$ - обсяг торгів у момент t, $close_t$, $close_{t-i}$ - ціни закриття відповідно до поточного та попереднього періоду. Таким чином, якщо ціна зростає, значення індикатора також зростає, якщо ціна падає, значення індикатора падає [30].

- SMA - це зміна N-денної простої ковзної середньої між учорашнім і сьогоднішнім днем із масштабним коефіцієнтом N+1, тобто:

$$\frac{momentum}{N+1} = SMA_{today} - SMA_{yesterday}$$

$$momentum = close_{today} - close_{N\ days\ ago}$$

Це нахил або крутість лінії SMA, як похідна . Загалом цей зв'язок мало обговорюється, але він цікавий для розуміння сигналів від індикатора.

Коли імпульс перетинає нуль, це відповідає спаду в SMA, а коли він перетинає нуль, це пік. Наскільки високий (чи низький) моментум стає очевидним, наскільки швидко зростає SMA [31].

- EMA - різновид виваженої ковзної середньої, ваги якої зменшуються експоненційно і ніколи не дорівнюють нулю. Визначається наступною формулою:

$$EMA_t = \alpha \cdot p_t + (1 - \alpha) \cdot EMA_{t-1},$$

де EMA_t значення експоненційного ковзного середнього в точці t (останнє значення, у разі часового ряду);

EMA_{t-1} - значення експоненційного ковзного середнього в точці t-1 (попереднє значення у разі часового ряду);

p_t- значення вихідної функції в момент часу t (останнє значення, у разі часового ряду);

α - коефіцієнт, що характеризує швидкість зменшення ваг, приймає значення від 0 і до 1, чим менше його значення тим більше вплив попередніх значень на поточну величину середнього [31].

Перше значення експоненційного ковзного середнього зазвичай приймається рівним першому значенню вихідної функції:

$$EMA_0 = p_0.$$

Коефіцієнт α може бути обраний довільним чином, в межах від 0 до 1.

Наприклад, він може бути виражений через величину вікна усереднення:

$$\alpha = \frac{2}{n+1}.$$

- Money Flow Index – технічний індикатор, покликаний показати інтенсивність, з якою гроші вкладаються в цінний папір і виводяться з нього, аналізуючи обсяги торгів та співвідношення типових цін періодів.

Як ключовий ціновий показник для індексу грошового потоку використовується типова ціна, [33] яка розраховується за такою формулою:

$$typical\ price = \frac{high + low + close}{3}$$

- Aroon Oscillator - технічний індикатор, що стежить за трендом, який використовує аспекти індикатора Aroon (Aroon Up і Aroon Down), щоб оцінити силу поточного тренду та ймовірність його продовження. Показання осцилятора Aroon вище нуля вказують на наявність висхідного тренду , а значення нижче нуля вказують на спадний тренд . Трейдери спостерігають за перетинанням нульової лінії, щоб сигналізувати про можливі зміни тренду. Вони також спостерігають за великими рухами, вище 50 або нижче -50, щоб сигналізувати про сильні рухи ціни [34]. Визначається наступною формулою:

$$\text{Aroon Oscillator} = \text{Aroon Up} - \text{Aroon Down}$$
$$\text{Aroon Up} = 100 * \frac{(25 - \text{Periods Since 25-Period High})}{25}$$
$$\text{Aroon Down} = 100 * \frac{(25 - \text{Periods Since 25-Period Low})}{25}$$

- Stochastic Oscillator - індикатор технічного аналізу, який показує положення поточної ціни щодо діапазону цін за певний період у минулому. Вимірюється у процентах. Згідно з тлумаченням автора індикатора Джорджа Лейна (англ.) рос., основна ідея полягає в тому, що при тенденції зростання ціни (зростаючий тренд) ціна закриття чергового таймфрейму має тенденцію зупинятися поблизу попередніх максимумів. При тенденції зниження ціни (падаючий тренд) ціна закриття чергового таймфрейму має тенденцію зупинятися поблизу попередніх мінімумів. Фактично, індикатор демонструє розбіжність ціни закриття поточного періоду щодо цін попередніх періодів у межах заданого проміжку часу [35].

Індикатор будується із двох ліній:

%K - швидкий стохастик (суцільна лінія, основний графік)

%D - повільний стохастик (пунктирна лінія, додатково усереднений графік)

Формула розрахунку:

$$\%K_t = \frac{C_t - Ln}{Hn - Ln} \cdot 100,$$

Для нейронної мережі було визначені такі методи оптимізації як:

- Line gradient descent - це алгоритм, який знаходить найкращу лінію для заданого навчального набору даних за меншу кількість ітерацій. Якщо ми побудуємо m і c проти MSE, він набуде форму чаші. Для деякої комбінації m і c ми отримаємо найменшу помилку (MSE) [36].

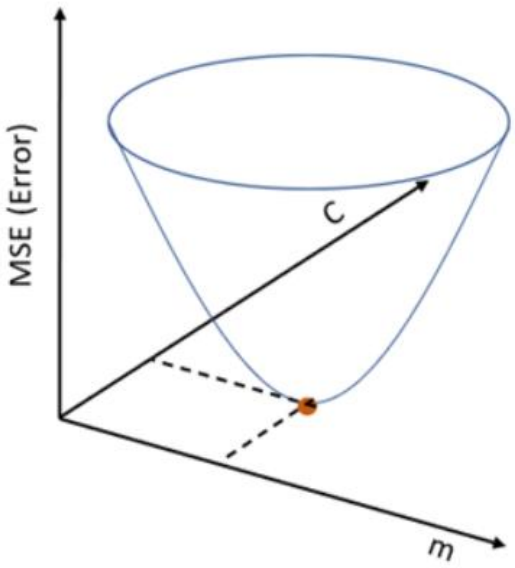

Рис.16. Графічне зображееня MSE

Ця комбінація m і c дасть нам найкращу лінію. Алгоритм починається з деякого значення m і c (зазвичай починається з m=0, c=0). Розраховуємо MSE (вартість) у точці m=0, c=0. Припустимо, MSE (вартість) при m=0, c=0 дорівнює 100. Потім ми зменшуємо значення m і c на деяку величину (Крок навчання). Ми помітимо зниження MSE (вартості). Ми будемо продовжувати робити те саме, доки наша

функція втрат не стане дуже малим значенням або в ідеалі 0 (що означає 0 помилок або 100% точність).

- Stochastic gradient descent - це ітераційний метод оптимізації цільової функції з відповідними властивостями гладкості. Його можна розглядати як стохастичне наближення оптимізації градієнтного спуску , оскільки воно замінює фактичний градієнт (розрахований на основі всього набору даних) його оцінкою (розрахованою на основі випадково вибраної підмножини даних). Особливо в задачах оптимізації великої розмірності це зменшує дуже велике обчислювальне навантаження, досягаючи швидших ітерацій у торгівлі за нижчого рівня конвергенції [37].
- Adam - adaptive moment estimation, це оптимізаційний алгоритм. Він поєднує у собі ідею накопичення руху та ідею слабшого оновлення ваг для типових ознак [38].

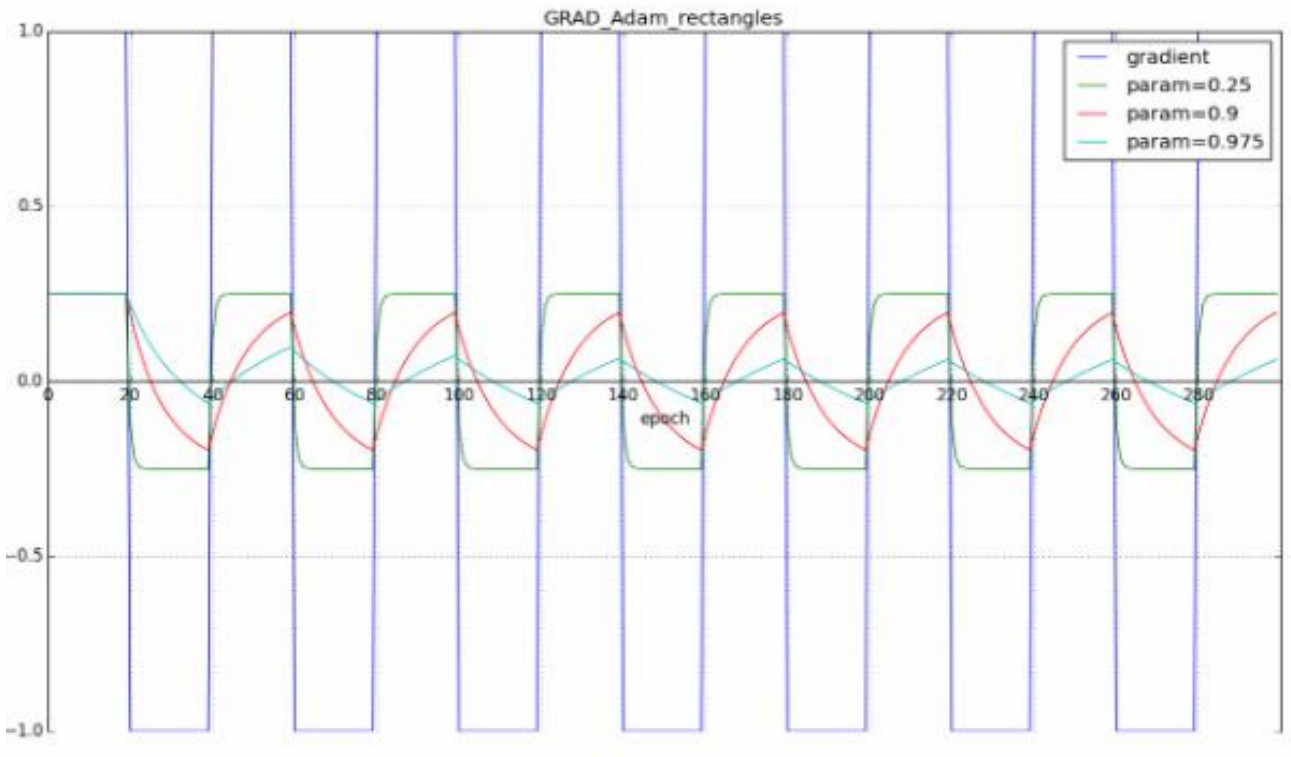

Рис.17. Графічне зображення принципу роботи Adam

В результаті всіх досліджень було визначено необхідні параметри моделі [40], нижче представлена готова модель з використанням бібліотеки deepLearning4J [39]:

```
MultiLayerConfiguration configuration = new NeuralNetConfiguration.Builder()
        .seed(System.currentTimeMillis())
        .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
        .updater(new RmsProp(learningRate))
        .l2(1e-4)
        .list()
        .layer(0, new LSTM.Builder()
            .nIn(NB_INPUTS)
            .nOut(lstmLayer1Size)
            .activation(Activation.TANH)
            .gateActivationFunction(Activation.HARDSIGMOID)
            .dropOut(dropoutRatio)
            .build())
        .layer(1, new LSTM.Builder()
            .nIn(lstmLayer1Size)
            .nOut(lstmLayer2Size)
            .activation(Activation.TANH)
            .gateActivationFunction(Activation.HARDSIGMOID)
            .dropOut(dropoutRatio)
            .build())
        .layer(2, new DenseLayer.Builder()
            .nIn(lstmLayer2Size)
            .nOut(denseLayerSize)
            .activation(Activation.RELU)
            .build())
        .layer(3, new RnnOutputLayer.Builder()
            .nIn(denseLayerSize)
            .nOut(1)
            .activation(Activation.IDENTITY)
            .lossFunction(LossFunctions.LossFunction.MSE)
            .build())
        .backpropType(BackpropType.TruncatedBPTT)
        .tBPTTForwardLength(truncatedBPTTLength)
        .tBPTTBackwardLength(truncatedBPTTLength)
        .build();
```

Середній результат точності даної моделі є 83%. Нижче представлений графік з наглядними даними моделі та реальною ціною, яка змінилися протягом години.

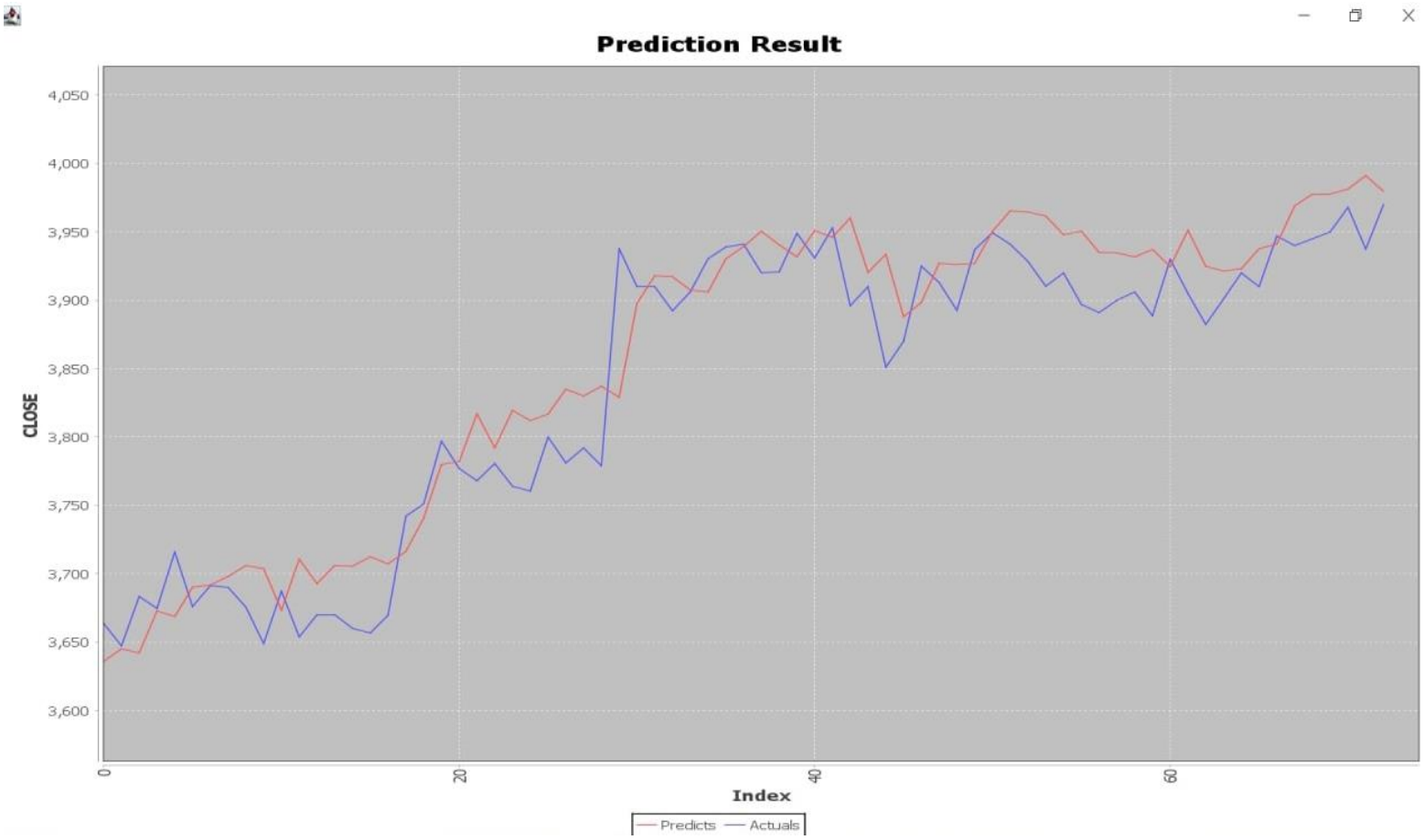

Рис.18. Результат роботи прогнозування

ВИСНОВКИ

Швидкий розвиток ринку крпиптовалют, їх популярність та доступність приводить до значної волатильності їх цін. В даній роботі була проведена робота побудови системи прогнозування курсу криптовалют з використанням аналізу. В результаті роботи отримані наступні результати:

1. Проаналізована сутність ринку криптовалют, платформи для купівлі та продажу. В ході аналізу були виявлено, що ринок криптовалют має велику капіталізацію, тому ціни через це є більш волативні ніж з фіатним чи паперовим ринком. Тому зміна ціни криптовалюті за добу на 3%, це є нормою для цього ринку, на відміну від валютного ринку з десятковими або сотими числами зміни ціни.

2. Аналіз бірж продажу, показав всі переваги користування той чи іншою біржою з точки зору трейдера та розробника. Було виділено найбільш підходящу для даної роботи біржу, API якої є безкоштовною, має найменш обмежень для отримання даних та має найбільшу історію трейдів, що покращую результат моделі аналізу та прогнозування в процесі навчання.

3. Проведено аналіз типових математичних моделей, що використовуються при розв’язанні задачі прогнозу курсу, проведено порівняння цих методів в контексті розв’язку поставленої задачі та виявлено, що кращі результати дає нейронна мережа з достатньо складною топологією і великою кількістю прихованих шарів та нейронів у них.

4. Розроблено веб-додаток, що має змогу зберігати дані з біржі, конвертацію даних до індексів, та прогнозування на основі навченої моделі ціни криптовалюти та тренду курсу. Середній показник даної моделі 83%.

5. На основі проведеного аналізу були запропоновані шляхи до удосконалення побудованої системи для покращення її точності та зручності практичного використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cryptocurrency Explained With Pros and Cons for Investment URL: https://www.investopedia.com/terms/c/cryptocurrency.asp. (дата звернення: 14.09.2022).

2. What is cryptocurrency? URL: https://cointelegraph.com/blockchain-for-beginners/what-is-a-cryptocurrency-a-beginners-guide-to-digital-money (дата звернення: 14.09.2022).

3. Michael Casey, Paul Vigna. The Age of Cryptocurrency: How Bitcoin and the Blockchain Are Challenging the Global Economic Order. 2018. P. 47–59.

4. Why Should You Use Crypto? URL: https://www.fool.com/investing/stock-market/market-sectors/financials/cryptocurrency-stocks/benefits-of-cryptocurrency/ (дата звернення: 15.09.2022).

5. The pros and cons of cryptocurrency URL: https://n26.com/en-eu/blog/pros-and-cons-of-cryptocurrency (дата звернення: 27.09.2022).

6. Harvard Business Review, Catherine Tucker. Blockchain: The Insights You Need from Harvard Business Review. 2019. P. 25–31.

7. Peer-to-Peer Service: Definition, Facts, and Examples. URL: https://www.investopedia.com/terms/p/peertopeer-p2p-service.asp (дата звернення: 01.10.2022).

8. Peer-to-Peer Network Connection. URL: https://www.webroot.com/us/en-resources/glossary/what-is-peer-to-peer-networking (дата звернення: 02.10.2022).

9. Coinbase API URL: https://docs.cloud.coinbase.com-exchange/docs/welcome (дата звернення: 03.10.2022).

10. Binance API URL: https://www.binance.com/en/binance-api (дата звернення: 03.10.2022).

11. Kraken API URL: https://docs.kraken.com/rest/ (дата звернення: 04.10.2022).

12. Bitfinex API URL: https://docs.bitfinex.com/docs (дата звернення: 04.10.2022).

13. Bitstamp API URL: https://www.bitstamp.net/api/ (дата звернення: 05.10.2022).

14. Kathy Warr. Надійність нейронних мереж. Зміцнюємо стійкість II до обману. 2021. С. 37–49.

15. Predicing the Stock Market with Machine Learning.Introduction. URL: https://towardsdatascience.com/predicting-the-stock-market-with-machine-learningintroduction-310cd6069ffa (дата звернення: 05.11.2022).

16. Predicting Stock Returns Using Financial Statement Information URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/1468-5957.t01-1-00205 (дата звернення: 18.10.2022)

17. Breiman L., Friedman J.H., Olshen R. A., Stone C. J. Classification and regression trees. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software. 1984. P. 312.

18. A new correlation coefficient between categorical, ordinal and interval variables with Pearson characteristics URL: https://arxiv.org/pdf/1811.11440.pdf (дата звернення: 19.10.2022)

19. Guide to LSTM’s and GRU’s URL: https://towardsdatascience.com-illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 (дата звернення: 05.10.2022)

20. Stochastic Recurrent Neural Network for Multistep Time Series Forecasting URL : https://arxiv.org/pdf/2104.12311.pdf (дата звернення: 20.10.2022).

21. Mike Bernico. Deep Learning Quick Reference: Useful Hacks for Training and Optimizing Deep Neural Networks. 2021. P. 54-65.

22. Recurrent Neural Networks URL: https://www.ibm.com/cloud/learn-recurrent-neural-networks (дата звернення: 11.10.2022).

23. Glossary recurrent neural network URL: https://www.arm.com/glossary-recurrent-neural-network (дата звернення: 21.10.2022)

24. Налаштування модуля регресії нейронної мережі у конструкторі машинного URL: https://conf.ztu.edu.ua/wp-content/uploads/2021/05/78-4.pdf (дата звернення: 13.10.2022)

25. How powerful can an ensemble of linear models be? URL: https://towardsdatascience.com/how-powerful-can-an-ensemble-of-linear-models-be231824de50e1 (дата звернення: 01.11.2022).

26. How-To Guide on Exploratory Data Analysis for Time Series Data URL: https://medium.com/analytics-vidhya/how-to-guide-on-exploratory-dataanalysis-for-time-series-data-34250ff1d04f (дата звернення: 15.11.2022).

27. Advanced Ensemble Classifiers URL: https://towardsdatascience.com-advanced-ensemble-classifiers-8d7372e74e40 (дата звернення: 16.11.2022).

28. Feature Selection Techinques URL: https://medium.com/analyticsvidhya-feature-selection-techniques-2614b3b7efcd (дата звернення: 17.10.2022).

29. How to Handle Missing Data URL: https://towardsdatascience.com/how-tohandle-missing-data-8646b18db0d4 (дата звернення: 17.09.2022).

30. On-Balance Volume (OBV): Definition, Formula, and Uses as Indicator URL: https://towardsdatascience.com/erroranalysis-to-your-rescue-773b401380ef (дата звернення: 20.11.2022).

31. Simple Moving Average (SMA): What It Is and the Formula URL: https://www.investopedia.com/terms/s/sma.asp (дата звернення: 19.10.2022).

32. How to Use the EMA Indicator URL: https://admiralmarkets.com/education-articles/forex-indicators/exponential-moving-average (дата звернення: 19.10.2022).

33. Money Flow Index - MFI Definition and Uses URL: https://www.investopedia.com/terms/m/mfi.asp (дата звернення: 21.10.2022).

34. Aroon Oscillator: Definition, Calculation Formula, Trade Signals URL: https://www.investopedia.com/terms/a/aroonoscillator.asp (дата звернення: 22.09.2022)

35. Stochastic Oscillator URL: https://corporatefinanceinstitute.com-resources/equities/stochastic-oscillator/ (дата звернення: 23.10.2022).

36. Linear Regression using Gradient Descent URL: https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931 (дата звернення: 23.10.2022).

37. Stochastic Gradient Descent - Clearly Explained URL: https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31 (дата звернення: 24.10.2022).

38. Adam: a method for stochastic optimization URL: https://arxiv.org/pdf/1412.6980.pdf (дата звернення: 24.10.2022).

39. Deeplearning4j Suite Overview URL: https://deeplearning4j.konduit.ai/ (дата звернення: 25.10.2022).

40. Xiang-Sun Zhang. Neural Networks in Optimization. 2000. P. 59-62.

ЛІСТИНГ ПРОГРАМИ

```
public interface ExtractorController {
   ResponseEntity<?> getCoinPriceHistories();
}

public interface StatisticsController {
   ResponseEntity<?> getGaps();
}

@RestController
@RequestMapping("/extract")
public class ExtractorControllerImpl implements ExtractorController {
   private ExtractorService extractorService;

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

   @GetMapping
   @Override
   public ResponseEntity<?> getCoinPriceHistories() {
      extractorService.doExtraction();
      return new ResponseEntity<>(HttpStatus.OK);
   }
}

@RestController
@RequestMapping("/statistics")
public class StatisticsControllerImpl implements StatisticsController {
   private StatisticsService statisticsService;

   @Autowired
   public StatisticsControllerImpl(StatisticsService statisticsService) {
      this.statisticsService = statisticsService;
   }

   @GetMapping("/gaps")
   @Override
   public ResponseEntity<?> getGaps() {
      return new ResponseEntity<>(statisticsService.getGaps(), HttpStatus.OK);
   }
}

@Configuration
public class AppConfig {

   /**
    * May be needed in specifying time and testing it
    */
```

```
    @Bean
    public Clock clock() {
        return Clock.systemDefaultZone();
    }
}

@SpringBootApplication(scanBasePackages = {"com.nmu.dlt.quotes.extractor"})
public class QuotesPredictorApplication {

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

}

public interface CoinPairDao {
    List<CoinPair> findByBaseCoin(String baseCoin);

    List<CoinPair> findByQuoteCoin(String quoteCoin);

    CoinPair save(CoinPair pair);

    List<CoinPair> saveAll(List<CoinPair> coinPairList);

    CoinPair findById(int id);

    List<CoinPair> findAll();

    void deleteById(int id);

    void deleteAll();

    void update(CoinPair pair);

    List<CoinPair> getActiveCoinPairsByExchange(int exchangeId);
}

public interface CoinPriceHistoryDao {

    List<CoinPriceHistory> saveAll(List<CoinPriceHistory> prices);

    CoinPriceHistory findByExchangeId(int id);

    CoinPriceHistory save(CoinPriceHistory prices);

    void delete(int coinPairId, int exchangeId, long candleTimeClose, String candleType);

    void deleteAll();

    List<CoinPriceHistory> findAll();

    CoinPriceHistory findById(int coinPairId, int exchangeId, long candleTimeClose, String candleType);
```

```
    List<CoinPriceHistory> findByCoinPairId(int coinPairId);


    void update(CoinPriceHistory prices);

    Optional<Long> findLastHistoryLoadedDate(int coinPairId, int exchangeID, String interval);
}

public interface ExchangeDao {
    Exchange save(Exchange entity);

    Exchange findById(int id);

    Exchange findByName(String exchangeName);

    List<Exchange> findAll();

    void update(Exchange testEntity);

    void deleteById(int id);

    void deleteAll();
}

public interface GapDao {
    List<Gap> findAllByCandleTypeAndExchangeIdAndCoinPairIdWithGaps(Interval candleType, int
exchangeId, int coinPairId);
}

public interface PeerToPeerDao {
    PeerToPeerPriceHistory save(PeerToPeerPriceHistory entity);

    List<PeerToPeerPriceHistory> save(List<PeerToPeerPriceHistory> prices);

    List<PeerToPeerPriceHistory> findAll();

    List<PeerToPeerPriceHistory> findByTimestampAndCoinPairId(Timestamp timestamp, int coinPair);

    void update(PeerToPeerPriceHistory peerToPeerEntity);

    void deleteByTimestampAndCoinPairIdAndExchangeId(Timestamp timestamp, int coinPair, int
exchange);

    void deleteAll();

    Optional<Timestamp> getLastHistoryLoadedDate(int coinPairId, int exchangeId, DealType type);
}

@Repository
public class CoinPairDaoImpl implements CoinPairDao {
```

```
    @Autowired
    private JdbcTemplate jdbcTemplate;

    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'";

    @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;
    }

    //TODO do everything with transactional
    @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) {
        return jdbcTemplate.query(FIND_ACTIVE_COIN_PAIRS_BY_EXCHANGE, new
CoinPairRowMapper(), exchangeId);
    }
}

@Repository
public class CoinPriceHistoryDaoImpl implements CoinPriceHistoryDao {

    @Autowired
    private JdbcTemplate jdbcTemplate;
```

```
    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
coin_pair_id=? AND exchange_id=? AND candle_time_close=? AND candle_type=?";
    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 coin_pair_id=?
AND exchange_id=? AND candle_time_close=? AND candle_type=?";
    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 = ?";

    @Override
    public CoinPriceHistory save(CoinPriceHistory coinPriceHistory) {
        jdbcTemplate.update(SAVE_QUERY,
                coinPriceHistory.getCoinPairId(),
                coinPriceHistory.getOpen(),
                coinPriceHistory.getHigh(),
                coinPriceHistory.getLow(),
                coinPriceHistory.getClose(),
                coinPriceHistory.getVolume(),
                coinPriceHistory.getNumberOfTrades(),
                coinPriceHistory.getCandleTimeOpen(),
                coinPriceHistory.getCandleTimeClose(),
                coinPriceHistory.getCandleType(),
                coinPriceHistory.getExchangeId());
        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(int coinPairId, int exchangeId, long candleTimeClose, String candleType) {
    jdbcTemplate.update(DELETE_QUERY, coinPairId, exchangeId, candleTimeClose, candleType);
}

@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(int coinPairId, int exchangeId, long candleTimeClose, String
candleType) {
    return jdbcTemplate.queryForObject(FIND_BY_ID_QUERY, new CoinPriceHistoryRowMapper(),
coinPairId, exchangeId, candleTimeClose, candleType);
}

@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 coinPairId, int exchangeId, String interval) {
        return Optional.ofNullable(jdbcTemplate.queryForObject(FIND_LAST_LOADED_HISTORY_DATE,
Long.class, coinPairId, exchangeId, interval));

    }

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

@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);
    }
}

@Repository
public class GapDaoImpl implements GapDao {
    private final String
FIND_ALL_BY_CANDLE_TYPE_AND_EXCHANGE_ID_AND_COIN_PAIR_ID_QUERY = "SELECT
exchange.exchange_name, coin_pair.base_coin, coin_pair.quote_coin, coin_price_history.candle_time_close,
coin_price_history.candle_type FROM coin_price_history\n" +
            "JOIN coin_pair ON coin_price_history.coin_pair_id = coin_pair.id\n" +
            "JOIN exchange ON coin_price_history.exchange_id = exchange.id\n" +
            "WHERE LOWER(candle_type) = LOWER(?) AND exchange.id = ? AND coin_pair.id = ?\n" +
            "ORDER BY coin_price_history.candle_time_close";

    @Autowired
    private JdbcTemplate jdbcTemplate;

    @Override
    public List<Gap> findAllByCandleTypeAndExchangeIdAndCoinPairIdWithGaps(Interval candleType, int
exchangeId, int coinPairId) {
        List<Gap> gaps = new ArrayList<>();

jdbcTemplate.query(FIND_ALL_BY_CANDLE_TYPE_AND_EXCHANGE_ID_AND_COIN_PAIR_ID_Q
UERY, new GapCallbackHandler(candleType, gaps), candleType.getType(), exchangeId, coinPairId);
        return gaps;
```

```
    }
}

@Service
public class PeerToPeerDaoImpl implements PeerToPeerDao {

    private static final String SAVE_QUERY = "INSERT INTO p2p_price_history (coin_pair_id,
exchange_id, price, price_time, deal_type, id) VALUES (?,?,?,?,?,?)";
    private static final String FIND_ALL_QUERY = "SELECT coin_pair_id, exchange_id, price, price_time,
deal_type, id FROM p2p_price_history";
    private static final String UPDATE_QUERY = "UPDATE p2p_price_history SET price=?, deal_type=?
WHERE price_time = ? AND coin_pair_id = ? AND exchange_id = ? AND id=?";
    private static final String
DELETE_BY_PRICE_TIME_AND_COIN_PAIR_ID_QUERY_AND_EXCHANGE_ID = "DELETE
FROM p2p_price_history WHERE price_time = ? AND coin_pair_id = ? AND exchange_id = ?";
    private static final String
FIND_BY_PRICE_TIME_AND_COIN_PAIR_ID_QUERY_AND_EXCHANGE_ID = "SELECT
coin_pair_id, exchange_id, price, price_time, deal_type, id FROM p2p_price_history WHERE price_time = ?
AND coin_pair_id = ?";
    private static final String DELETE_ALL_QUERY = "DELETE FROM p2p_price_history";
    private final String FIND_LAST_LOADED_HISTORY_DATE = "SELECT MAX(price_time) FROM
p2p_price_history WHERE exchange_id = ? AND coin_pair_id = ? AND deal_type=?";

    private final JdbcTemplate jdbcTemplate;

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

    @Override
    public PeerToPeerPriceHistory save(PeerToPeerPriceHistory price) {

        price.setId((short)1);

        jdbcTemplate.update(connection ->
        {
            PreparedStatement preparedStatement = connection.prepareStatement(SAVE_QUERY);
            preparedStatement.setInt(1, price.getCoinPairId());
            preparedStatement.setInt(2, price.getExchangeId());
            preparedStatement.setBigDecimal(3, price.getPrice());
            preparedStatement.setTimestamp(4, price.getPriceTime());
            preparedStatement.setInt(5, price.getTradeType().getDbValue());
            preparedStatement.setShort(6, price.getId());

            return preparedStatement;

        });

        return price;
    }
```

```
    @Override
    public List<PeerToPeerPriceHistory> save(List<PeerToPeerPriceHistory> prices) {

        for(short i = 0; i<prices.size(); i++)
            prices.get(i).setId(Short.valueOf((short)(i+1)));

        jdbcTemplate.batchUpdate(SAVE_QUERY, new BatchPreparedStatementSetter() {

            @Override
            public void setValues(PreparedStatement preparedStatement, int i) throws SQLException {
                PeerToPeerPriceHistory price = prices.get(i);
                preparedStatement.setInt(1, price.getCoinPairId());
                preparedStatement.setInt(2, price.getExchangeId());
                preparedStatement.setBigDecimal(3, price.getPrice());
                preparedStatement.setTimestamp(4, price.getPriceTime());
                preparedStatement.setInt(5, price.getTradeType().getDbValue());
                preparedStatement.setShort(6, price.getId());
            }

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

        return prices;
    }

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

    @Override
    public List<PeerToPeerPriceHistory> findByTimestampAndCoinPairId(Timestamp timestamp, int
coinPair) {
        return
jdbcTemplate.query(FIND_BY_PRICE_TIME_AND_COIN_PAIR_ID_QUERY_AND_EXCHANGE_ID, new
PeerToPeerRowMapper(), timestamp, coinPair);
    }

    @Override
    public void update(PeerToPeerPriceHistory price) {
        jdbcTemplate.update(UPDATE_QUERY, price.getPrice(), price.getTradeType().getDbValue(),
price.getPriceTime(), price.getCoinPairId(), price.getExchangeId(), price.getId());
    }

    public void deleteByTimestampAndCoinPairIdAndExchangeId(Timestamp timestamp, int coinPair, int
exchange) {

jdbcTemplate.update(DELETE_BY_PRICE_TIME_AND_COIN_PAIR_ID_QUERY_AND_EXCHANGE_ID,
timestamp, coinPair, exchange);
    }
```

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

    @Override
    public Optional<Timestamp> getLastHistoryLoadedDate(int coinPairId, int exchangeId, DealType type) {
        return Optional.ofNullable(jdbcTemplate.queryForObject(FIND_LAST_LOADED_HISTORY_DATE,
Timestamp.class, coinPairId, exchangeId, type.getDbValue()));
    }
}

@RequiredArgsConstructor
public class GapCallbackHandler implements RowCallbackHandler {
    @NonNull
    private Interval interval;

    @NonNull
    private List<Gap> gaps;

    private long previous;
    private long current;

    @Override
    public void processRow(ResultSet resultSet) throws SQLException {
        if (previous == 0)
            previous = resultSet.getLong("candle_time_close");
        else if (current == 0) {
            current = resultSet.getLong("candle_time_close");
            checkIfGap(resultSet);
        }
        else {
            previous = current;
            current = resultSet.getLong("candle_time_close");
            checkIfGap(resultSet);
        }
    }

    private void checkIfGap(ResultSet resultSet) throws SQLException {
        if (current - previous <= interval.getTime())
            return;
        Gap gap = new Gap();
        gap.setCandleType(resultSet.getString("candle_type"));
        gap.setExchangeName(resultSet.getString("exchange_name"));
        gap.setBaseCoin(resultSet.getString("base_coin"));
        gap.setQuoteCoin(resultSet.getString("quote_coin"));
        gap.setStartDate(previous);
        gap.setEndDate(current);
        gaps.add(gap);
    }
}
```

```
public class CoinPairRowMapper implements RowMapper<CoinPair> {

    @Override
    public CoinPair mapRow(ResultSet resultSet, int i) throws SQLException {
        CoinPair coinPair = new CoinPair();
        coinPair.setId(resultSet.getInt("id"));
        coinPair.setBaseCoin(resultSet.getString("base_coin"));
        coinPair.setQuoteCoin(resultSet.getString("quote_coin"));
        coinPair.setActive(resultSet.getBoolean("active"));
        return coinPair;
    }
}

public class CoinPriceHistoryRowMapper implements RowMapper<CoinPriceHistory> {

    @Override
    public CoinPriceHistory mapRow(ResultSet resultSet, int i) throws SQLException {
        CoinPriceHistory coinPriceHistory = new CoinPriceHistory();
        coinPriceHistory.setCoinPairId(resultSet.getInt("coin_pair_id"));
        coinPriceHistory.setOpen(resultSet.getBigDecimal("open"));
        coinPriceHistory.setHigh(resultSet.getBigDecimal("high"));
        coinPriceHistory.setLow(resultSet.getBigDecimal("low"));
        coinPriceHistory.setClose(resultSet.getBigDecimal("close"));
        coinPriceHistory.setVolume(resultSet.getBigDecimal("volume"));
        coinPriceHistory.setNumberOfTrades(resultSet.getLong("number_of_trades"));
        coinPriceHistory.setCandleTimeOpen(resultSet.getLong("candle_time_open"));
        coinPriceHistory.setCandleTimeClose(resultSet.getLong("candle_time_close"));
        coinPriceHistory.setCandleType(resultSet.getString("candle_type"));
        coinPriceHistory.setExchangeId(resultSet.getInt("exchange_id"));
        return coinPriceHistory;
    }
}

public class ExchangeRowMapper implements RowMapper<Exchange> {
    @Override
    public Exchange mapRow(ResultSet resultSet, int i) throws SQLException {
        Exchange exchange = new Exchange();
        exchange.setId(resultSet.getInt("id"));
        exchange.setExchangeName(resultSet.getString("exchange_name"));
        exchange.setActive(resultSet.getBoolean("active"));
        return exchange;
    }
}

public class PeerToPeerRowMapper implements RowMapper<PeerToPeerPriceHistory> {

    @Override
    public PeerToPeerPriceHistory mapRow(ResultSet resultSet, int i) throws SQLException {
        int deal_type = resultSet.getInt("deal_type");
```

```
        PeerToPeerPriceHistory peerToPeerPrice = new PeerToPeerPriceHistory();
        peerToPeerPrice.setId(resultSet.getShort("id"));
        peerToPeerPrice.setCoinPairId(resultSet.getInt("coin_pair_id"));
        peerToPeerPrice.setExchangeId(resultSet.getInt("exchange_id"));
        peerToPeerPrice.setPrice(resultSet.getBigDecimal("price"));
        peerToPeerPrice.setPriceTime(resultSet.getTimestamp("price_time"));
        peerToPeerPrice.setTradeType(deal_type == 0 ? DealType.BUY : DealType.SELL);

        return peerToPeerPrice;
    }

}

@Data
@AllArgsConstructor
@NoArgsConstructor
public class CoinPair {

    private Integer Id;

    private String baseCoin;

    private String quoteCoin;

    private Boolean active;
}

@Data
@AllArgsConstructor
@NoArgsConstructor
public class CoinPriceHistory {

    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;

}

@Data
@NoArgsConstructor
@AllArgsConstructor
public class CoinsByExchange {

    private Integer coinsPairId;
    private Integer exchangeId;
}

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

    private int dbValue;

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

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Exchange {

    private Integer id;

    private String exchangeName;

    private Boolean active;

}

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Gap {
    private String exchangeName;
    private Long startDate;
    private Long endDate;
    private String baseCoin;
    private String quoteCoin;
    private String candleType;
}

@Getter
@AllArgsConstructor
```

```
public enum Interval {
    ONE_MINUTE("1m", 60_000),
    THREE_MINUTES("3m", 60_000 * 3),
    FIVE_MINUTES("5m", 60_000 * 5),
    FIFTEEN_MINUTES("15m", 60_000 * 15),
    THIRTY_MINUTES("30m", 60_000 * 30),
    ONE_HOUR("1h", 60_000 * 60),
    TWO_HOURS("2h", 60_000 * 60 * 2),
    FOUR_HOURS("4h", 60_000 * 60 * 4),
    EIGHT_HOURS("8h", 60_000 * 60 * 8),
    TWELVE_HOURS("12h", 60_000 * 60 * 12),
    ONE_DAY("1d", 60_000 * 60 * 24),
    THREE_DAYS("3d", 60_000 * 60 * 24 * 3),
    ONE_WEEK("1w", 60_000 * 60 * 24 * 7);
//    ONE_MONTH("1mn");

    private final String type;
    private final long time;
}

@Data
@NoArgsConstructor
@AllArgsConstructor
public class PeerToPeerPriceHistory {

    private Short id;
    private Integer coinPairId;
    private BigDecimal price;
    private Integer exchangeId;
    private Timestamp priceTime;
    private DealType tradeType;

    public PeerToPeerPriceHistory(Integer coinPairId, BigDecimal price, Integer exchangeId, Timestamp
priceTime, DealType tradeType) {
        this.coinPairId = coinPairId;
        this.price = price;
        this.exchangeId = exchangeId;
        this.priceTime = priceTime;
        this.tradeType = tradeType;
    }
}

@Getter
public enum Sources {
    BINANCE(1);

    private int id;

    Sources(int id) {
        this.id = id;
    }
}
```

```
@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 String p2pUrlPrefix;
   private String p2pUrl;
}

@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();
   }
}

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";
}
```

```
@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);
      List<CandleItem> collect = (List<CandleItem>) response.stream()
            .map(o -> CandleConverter.fromArray((List) o)).collect(Collectors.toList());
      return collect;
   }


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

      return UriComponentsBuilder.fromHttpUrl(binanceConnectionConfiguration.getCandleUrlPrefix()
            + binanceConnectionConfiguration.getCandleUrl())
            .queryParam(CandleQueryParams.candleUrlQueryStartTime
                  , begin)
            .queryParam(CandleQueryParams.candleUrlQuerySymbol,
                  buildCoinName(symbol))
            .queryParam(CandleQueryParams.candleUrlQueryInterval,

BinanceInterval.checkIntervals(binanceConnectionConfiguration.getCandleInterval()).getCode())
            .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;
    }
}

public interface BinanceConnector {

    List<CandleItem> requestBarData(long begin, CoinPair symbol);

    List<BinanceDatumDto> requestP2PData(String asset, String fiat, DealType dealType);

    String getInterval();

}

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;
    }

}

@Getter
public enum BinanceInterval {
    ONE_MIN("1"),
    THREE_MIN("3m"),
    FIVE_MIN("5m"),
    FIFTEEN_MIN("15m"),
    THIRTY_MIN("30m"),
    ONE_HOUR("1h"),
    TWO_HOUR("2h"),
    FOUR_HOUR("4h"),
    SIX_HOUR("6h"),
    EIGHT_HOUR("8h"),
    TWELVE_HOUR("12h"),
    ONE_DAY("1d" ),
    THREE_DAY("3d"),
    ONE_WEEK("1w"),
    ONE_MONTH("1M");
    private final String code;

    BinanceInterval(String code) {
        this.code = code;
    }

    public static BinanceInterval checkIntervals(String property){
        BinanceInterval[] binanceIntervals = BinanceInterval.values();
        for (int i = 0; i < binanceIntervals.length; i++) {
            if (property.equals(binanceIntervals[i].getCode())){
                return binanceIntervals[i];}
        }
        return BinanceInterval.ONE_HOUR;
    }
}
```

```
@Data
public class BinanceAdvDetailDto {
   private String asset;
   private String minOrderPrice;
   private String maxOrderPrice;
   private String minOrderAmount;
   private String maxOrderAmount;
   private String fiatUnit;
   private String price;
   private DealType dealType;

}

@Data
public class BinanceDatumDto {

   private BinanceAdvDetailDto advDetail;

}

@Data
public class BinanceP2PRequestBody {
   int page;
   int rows;
   String asset;
   DealType dealType;
   String fiat;
}

@Data
public class BinanceP2PResponseDto {

   private List<BinanceDatumDto> data;
   private int total;
   private boolean success;

}

@Getter
@EqualsAndHashCode
@Setter
public class CandleItem {

   private Long openTime;
   private BigDecimal open;
   private BigDecimal high;
   private BigDecimal low;
   private BigDecimal close;
   private BigDecimal volume;
   private Long closeTime;
```

```
    private BigDecimal quoteAssetVolume;
    private BigDecimal numberOfTrades;
    private BigDecimal takerBuyBaseAssetVolume;
    private BigDecimal takerBuyQuoteAssetVolume;
    private BigDecimal Ignore;

}

@Service
public class BinanceP2PProvider implements P2PPriceProvider {

    private final BinanceConnector binanceConnector;
    private final PeerToPeerPriceService peerToPeerPriceService;
    private final Clock clock;

    //TODO: use available coin pairs here
    private final List<CoinPair> coinPairs = Arrays.asList(
            new CoinPair(1, "BTC", "USD", true)
    );

    @Autowired
    public BinanceP2PProvider(BinanceConnector binanceConnector, PeerToPeerPriceService
peerToPeerPriceService, Clock clock) {
        this.binanceConnector = binanceConnector;
        this.peerToPeerPriceService = peerToPeerPriceService;
        this.clock = clock;
    }

    @Override
    public CompletableFuture<Void> loadData() {
        return CompletableFuture.runAsync(() -> {
            coinPairs.forEach(coinPair -> {
                Arrays.stream(DealType.values()).forEach(
                        dealType -> {
                            loadByDeal(coinPair, dealType);
                        }
                );
            });
        });
    }

    private void loadByDeal(CoinPair coinPair, DealType dealType) {
        Timestamp requestTime = new Timestamp(clock.millis());

        List<BinanceDatumDto> retrievedData = binanceConnector
                .requestP2PData(coinPair.getBaseCoin(), coinPair.getQuoteCoin(), dealType);

        List<PeerToPeerPriceHistory> peerToPeerPriceHistoryList =
                retrievedData.stream().map(binanceDatumDto ->
                        new PeerToPeerPriceHistory(
                                coinPair.getId(),
                                new BigDecimal(binanceDatumDto.getAdvDetail().getPrice()),
```

```
                        1,
                        requestTime,
                        dealType))
                .collect(Collectors.toList());

        peerToPeerPriceService.save(peerToPeerPriceHistoryList);

    }

}

@Service
public class BinancePriceBarProvider implements PriceBarProvider {
    private static final long MAX_DELTA = 1_800_000;

    BinanceConnector binanceConnector;
    CoinPriceHistoryService coinPriceHistoryService;
    CoinPairService coinPairService;

    @Autowired
    public BinancePriceBarProvider(BinanceConnector binanceConnector, CoinPriceHistoryService
coinPriceHistoryService, CoinPairService coinPairService) {
        this.binanceConnector = binanceConnector;
        this.coinPriceHistoryService = coinPriceHistoryService;
        this.coinPairService = coinPairService;

    }


    /**
     * TODO
     * this method should load and save data using @binanceConnector
     * it should handle all provider restrictions (calls per minute etc)
     * loading can be scheduled and result CompletableFuture can be completed after all data is loaded
     */
    @Override
    public CompletableFuture<Void> loadData() {

        return CompletableFuture.runAsync(() -> {
            List<CoinPair> supportedCoinPairs =
coinPairService.getActiveCoinPairsByExchange(Sources.BINANCE.getId());
            for (CoinPair coinPair : supportedCoinPairs) {
                long begin = coinPriceHistoryService.findLastHistoryLoadedDate(coinPair.getId(),
Sources.BINANCE.getId(), binanceConnector.getInterval());
                List<CandleItem> candleItems = binanceConnector.requestBarData(begin, coinPair);
                while (System.currentTimeMillis() - begin >= MAX_DELTA && candleItems.size() != 0) {
                    List<CoinPriceHistory> coinPriceHistories = new ArrayList<>(candleItems.size());
                    begin = candleItems.get(0).getCloseTime();
                    for (CandleItem candleItem : candleItems) {
                        if (begin < candleItem.getCloseTime())
                            begin = candleItem.getCloseTime();
                        CoinPriceHistory coinPriceHistory = mapCandleItemToCoinPriceHistory(candleItem,
```

```
coinPair, Sources.BINANCE.getId(), binanceConnector.getInterval());
                    coinPriceHistories.add(coinPriceHistory);
                }
                coinPriceHistoryService.saveAll(coinPriceHistories);
                candleItems = binanceConnector.requestBarData(begin, coinPair);
            }
        }
    });
  }

  private void updateState(CoinPair coinPair, BinanceState state) {
    //Store state
  }

  @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;
  }
}

public interface P2PPriceProvider {
  CompletableFuture<Void> loadData();
}

public interface PriceBarProvider {

  CompletableFuture<Void> loadData();

  CoinPriceHistory mapCandleItemToCoinPriceHistory(CandleItem candleItem, CoinPair coinPair, int
exchangeId, String candleType);
}

public interface CoinPairService {

  CoinPair save(CoinPair coinPair);

  List<CoinPair> saveAll(List<CoinPair> prices);

  void deleteById(int id);
```

```
    void deleteAll();
    void update(CoinPair updatedCoinPair);
    List<CoinPair> findAll();
    CoinPair findById(int id);
    List<CoinPair> findByBaseCoin(String baseCoin);
    List<CoinPair> findByQuoteCoin(String quoteCoin);
    List<CoinPair> getActiveCoinPairsByExchange(int exchangeId);
}
public interface CoinPriceHistoryService {
    CoinPriceHistory save(CoinPriceHistory coinPriceHistory);
    List<CoinPriceHistory> saveAll(List<CoinPriceHistory> coinPriceHistoryList);
    void delete(int coinPairId, int exchangeId, long candleTimeClose, String candleType);
    void deleteAll();
    void update(CoinPriceHistory updatedCoinPriceHistory);
    List<CoinPriceHistory> findAll();
    CoinPriceHistory findById(int coinPairId, int exchangeId, long candleTimeClose, String candleType);
    Long findLastHistoryLoadedDate(int coinPairId, int exchangeId, String interval);
}
public interface ExchangeService {
    Exchange save(Exchange exchange);
    void deleteById(int id);
    void deleteAll();
    void update(Exchange  updatedExchange);
    List<Exchange> findAll();
    Exchange findById(int id);
    Exchange findByName(String exchangeName);
}
```

```
public interface ExtractorService {
   void doExtraction();
}
public interface GapService {
   List<Gap> findAllByCandleTypeAndExchangeIdAndCoinPairIdWithGaps(Interval candleType, int
exchangeId, int coinPairId);
}

public interface PeerToPeerPriceService {
   PeerToPeerPriceHistory save(PeerToPeerPriceHistory entity);

   List<PeerToPeerPriceHistory> save(List<PeerToPeerPriceHistory> prices);

   void deleteAll();

   void deleteByTimestampAndCoinPairIdAndExchangeId(Timestamp timestamp, int coinPair, int
exchange);

   void update(PeerToPeerPriceHistory updatedExchange);

   List<PeerToPeerPriceHistory> findAll();

   List<PeerToPeerPriceHistory> findByTimestampAndCoinPairId(Timestamp timestamp, int coinPair);

   Timestamp getLastHistoryLoadedDate(int coinPairId, int exchangeId, DealType type);
}

public interface StatisticsService {
   Map<String, List<Gap>> getGaps();
}

@Service
public class CoinPairServiceImpl implements CoinPairService {

   private final CoinPairDao coinPairDao;

   @Autowired
   public CoinPairServiceImpl(CoinPairDao coinPairDao) {
      this.coinPairDao = coinPairDao;
   }

   @Override
   public CoinPair save(CoinPair coinPair) {
      return coinPairDao.save(coinPair);
   }

   @Override
   public List<CoinPair> saveAll(List<CoinPair> prices) {
      return coinPairDao.saveAll(prices);
   }
```

```
    @Override
    public void deleteById(int id) {
        coinPairDao.deleteById(id);
    }

    @Override
    public void deleteAll() {
        coinPairDao.deleteAll();
    }

    @Override
    public void update(CoinPair updatedCoinPair) {
        coinPairDao.update(updatedCoinPair);
    }

    @Override
    public List<CoinPair> findAll() {
        return coinPairDao.findAll();
    }

    @Override
    public CoinPair findById(int id) {
        return coinPairDao.findById(id);
    }

    @Override
    public List<CoinPair> findByBaseCoin(String baseCoin) {
        return coinPairDao.findByBaseCoin(baseCoin);
    }

    @Override
    public List<CoinPair> findByQuoteCoin(String quoteCoin) {
        return coinPairDao.findByQuoteCoin(quoteCoin);
    }

    @Override
    public List<CoinPair> getActiveCoinPairsByExchange(int exchangeId) {
        return coinPairDao.getActiveCoinPairsByExchange(exchangeId);
    }
}

@Service
public class CoinPriceHistoryServiceImpl implements CoinPriceHistoryService {
    private static final long START_POINT = 1617715246000L;
    private static final Timestamp DEFAULT_BEGIN = new Timestamp(START_POINT);

    private final CoinPriceHistoryDao coinPriceHistoryDao;

    @Autowired
    public CoinPriceHistoryServiceImpl(CoinPriceHistoryDao coinPriceHistoryDao) {
        this.coinPriceHistoryDao = coinPriceHistoryDao;
```

```
    }

    @Override
    public CoinPriceHistory save(CoinPriceHistory coinPriceHistory) {
        return coinPriceHistoryDao.save(coinPriceHistory);
    }

    @Override
    public List<CoinPriceHistory> saveAll(List<CoinPriceHistory> coinPriceHistoryList) {
        return coinPriceHistoryDao.saveAll(coinPriceHistoryList);
    }

    @Override
    public void delete(int coinPairId, int exchangeId, long candleTimeClose, String candleType) {
        coinPriceHistoryDao.delete(coinPairId, exchangeId, candleTimeClose, candleType);
    }

    @Override
    public void deleteAll() {
        coinPriceHistoryDao.deleteAll();
    }

    @Override
    public void update(CoinPriceHistory updatedCoinPriceHistory) {
        coinPriceHistoryDao.update(updatedCoinPriceHistory);
    }

    @Override
    public List<CoinPriceHistory> findAll() {
        return coinPriceHistoryDao.findAll();
    }

    @Override
    public CoinPriceHistory findById(int coinPairId, int exchangeId, long candleTimeClose, String
candleType) {
        return coinPriceHistoryDao.findById(coinPairId,exchangeId, candleTimeClose, candleType);
    }

    @Override
    public Long findLastHistoryLoadedDate(int coinPairId, int exchangeId, String interval) {
        return coinPriceHistoryDao.findLastHistoryLoadedDate(coinPairId, exchangeId,
interval).orElse(Timestamp.valueOf(LocalDateTime.of(2015, 1,1,10,0,0)).getTime());

    }
}

@Service
public class ExchangeServiceImpl implements ExchangeService {

    private final ExchangeDao exchangeDao;

    @Autowired
```

```
    public ExchangeServiceImpl(ExchangeDao exchangeDao) {
        this.exchangeDao = exchangeDao;
    }

    @Override
    public Exchange save(Exchange exchange) {
        return exchangeDao.save(exchange);
    }

    @Override
    public void deleteById(int id) {
        exchangeDao.deleteById(id);
    }

    @Override
    public void deleteAll() {
        exchangeDao.deleteAll();
    }

    @Override
    public void update(Exchange updatedExchange) {
        exchangeDao.update(updatedExchange);
    }

    @Override
    public List<Exchange> findAll() {
        return exchangeDao.findAll();
    }

    @Override
    public Exchange findById(int id) {
        return exchangeDao.findById(id);
    }

    @Override
    public Exchange findByName(String exchangeName) {
        return exchangeDao.findByName(exchangeName);
    }
}

@Service
public class ExtractorServiceImpl implements ExtractorService {

    private final List<PriceBarProvider> barProviders;
    private final List<P2PPriceProvider> p2pProviders;

    @Autowired
    public ExtractorServiceImpl(List<PriceBarProvider> barProviders, List<P2PPriceProvider> p2pProviders)
{
        this.barProviders = barProviders;
        this.p2pProviders = p2pProviders;
    }
```

```
    @PostConstruct
    public void init() {
        //TODO initialization here
    }

    /**
     * This method will be invoked periodically.
     * It collects data from all providers and store it
     */
    @Override
    public void doExtraction() {

        for (PriceBarProvider provider : barProviders) {
            //TODO check provider active
            CompletableFuture<Void> longCompletableFuture = provider.loadData();
        }
        for (P2PPriceProvider provider : p2pProviders) {
            //TODO check provider active
            CompletableFuture<Void> longCompletableFuture = provider.loadData();
        }
        //TODO we can wait all futures is completed (CompletableFuture.allOf(...).get())
    }

}

@Service
public class GapServiceImpl implements GapService {
    private GapDao gapDao;

    @Autowired
    public GapServiceImpl(GapDao gapDao) {
        this.gapDao = gapDao;
    }

    @Override
    public List<Gap> findAllByCandleTypeAndExchangeIdAndCoinPairIdWithGaps(Interval candleType, int
exchangeId, int coinPairId) {
        return gapDao.findAllByCandleTypeAndExchangeIdAndCoinPairIdWithGaps(candleType, exchangeId,
coinPairId);
    }
}

@Service
public class PeerToPeerPriceServiceImpl implements PeerToPeerPriceService {

    private final PeerToPeerDao peerToPeerDao;

    @Autowired
    public PeerToPeerPriceServiceImpl(PeerToPeerDao peerToPeerDao) {
        this.peerToPeerDao = peerToPeerDao;
    }
```

```
    @Override
    public PeerToPeerPriceHistory save(PeerToPeerPriceHistory entity) {
        return peerToPeerDao.save(entity);
    }

    @Override
    public List<PeerToPeerPriceHistory> save(List<PeerToPeerPriceHistory> prices) {

        peerToPeerDao.save(prices);

        return prices;
    }

    @Override
    public void deleteAll() {
        peerToPeerDao.deleteAll();
    }

    @Override
    public void deleteByTimestampAndCoinPairIdAndExchangeId(Timestamp timestamp, int coinPair, int
exchange) {
        peerToPeerDao.deleteByTimestampAndCoinPairIdAndExchangeId(timestamp, coinPair, exchange);
    }

    @Override
    public void update(PeerToPeerPriceHistory updatedExchange) {
        peerToPeerDao.update(updatedExchange);
    }

    @Override
    public List<PeerToPeerPriceHistory> findAll() {
        return peerToPeerDao.findAll();
    }

    @Override
    public List<PeerToPeerPriceHistory> findByTimestampAndCoinPairId(Timestamp timestamp, int
coinPair) {
        return peerToPeerDao.findByTimestampAndCoinPairId(timestamp, coinPair);
    }

    public Timestamp getLastHistoryLoadedDate(int coinPairId, int exchangeId, DealType type) {
        return peerToPeerDao.getLastHistoryLoadedDate(coinPairId, exchangeId, type).get();
    }
}

@Service
public class StatisticsServiceImpl implements StatisticsService {
    private GapService gapService;
    private ExchangeService exchangeService;
    private CoinPairService coinPairService;
```

```
    @Autowired
    public StatisticsServiceImpl(GapService gapService, ExchangeService exchangeService, CoinPairService
coinPairService) {
        this.gapService = gapService;
        this.exchangeService = exchangeService;
        this.coinPairService = coinPairService;
    }

    @Override
    public Map<String, List<Gap>> getGaps() {
        Interval[] intervals = Interval.values();
        Map<String, List<Gap>> gaps = new HashMap<>(intervals.length);
        for (Interval interval : intervals)
            gaps.put(interval.getType(), new ArrayList<>());
        List<Exchange> exchanges = exchangeService.findAll();
        List<CoinPair> coinPairs = coinPairService.findAll();
        for (Exchange exchange : exchanges)
            for (CoinPair coinPair : coinPairs)
                for (Interval interval : intervals)

gaps.get(interval.getType()).addAll(gapService.findAllByCandleTypeAndExchangeIdAndCoinPairIdWithGa
ps(interval, exchange.getId(), coinPair.getId()));
        return gaps;
    }
}

MultiLayerConfiguration configuration = new NeuralNetConfiguration.Builder()
            .seed(System.currentTimeMillis())
            .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
            .updater(new RmsProp(learningRate))
            .l2(1e-4)
            .list()
            .layer(0, new LSTM.Builder()
                  .nIn(NB_INPUTS)
                  .nOut(lstmLayer1Size)
                  .activation(Activation.TANH)
                  .gateActivationFunction(Activation.HARDSIGMOID)
                  .dropOut(dropoutRatio)
                  .build())
            .layer(1, new LSTM.Builder()
                  .nIn(lstmLayer1Size)
                  .nOut(lstmLayer2Size)
                  .activation(Activation.TANH)
                  .gateActivationFunction(Activation.HARDSIGMOID)
                  .dropOut(dropoutRatio)
                  .build())
            .layer(2, new DenseLayer.Builder()
                  .nIn(lstmLayer2Size)
                  .nOut(denseLayerSize)
                  .activation(Activation.RELU)
                  .build())
            .layer(3, new RnnOutputLayer.Builder()
                  .nIn(denseLayerSize)
```

```
        .nOut(1)
        .activation(Activation.IDENTITY)
        .lossFunction(LossFunctions.LossFunction.MSE)
        .build())
.backpropType(BackpropType.TruncatedBPTT)
.tBPTTForwardLength(truncatedBPTTLength)
.tBPTTBackwardLength(truncatedBPTTLength)
.build();
```

ВІДГУК КЕРІВНИКА

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

Факультет інформаційних технологій
Кафедра програмного забезпечення комп'ютерних систем

ВІДГУК

Наукового керівника Мещерякова Леоніда Івановича, д.т.н., проф. каф. ПЗКС
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада, місце роботи)

на магістерську роботу

студента Шарандо Артема Андрійовича
(прізвище, ім'я, по батькові)

курсу II групи 122м-21-1
спеціальності 122 Комп'ютерні науки
освітньої програми
на тему Обгрунтування структури інформаціїної системи спостерігання та прогнозу динамікі криптовалют

Актуальність теми Представлена магістерська кваліфікаційна робота присвячена обгрунтування структури інформаціїної системи спостерігання та прогнозу динамікі криптовалют. На сьогоднішній момент це є однією з передових галузей розвитку інформаційних технологій, тому з огляду на це, магістерська робота характеризується актуальністю та своєчасністю.

Мета досліджень Полягає у дослідженні та вдосконаленні існуючих методик побудови прогнозуючих моделей та розробці системи підтримки прийняття рішень для короткострокового прогнозування курсу криптовалют з використанням моделей нейронних мереж.

Коротка характеристика розділів роботи аналітичний огляд літературних джерел

за темою магістерської роботи. Другий розділ присвячено огляду проблем, що вирішуються в ході виконання роботи, а також представлені шляхи вирішення цих проблем. В третьому розділі розглянута задача програмної реалізації методу побудови прогнозуючих моделей та розробці системи підтримки прийняття рішень для короткострокового прогнозування курсу криптовалют з використанням моделей нейронних мереж.

Практичне значення роботи результатів полягає у тому, що розроблено програмне забезпечення для спостерігання та аналізу даних криптовалют, показ зміни динаміки та прогнозування подальшої зміни ціни віртуальної валюти.

Зауваження та недоліки В роботі відсутній більш детальний огляд проблем, що постають під час програмної реалізації інформаційної системи аналізу та прогнозування динаміки зміни ціни криптовалют, їх причини та залежність від мови програмування.

Висновки та оцінка Магістром було проведено аналіз та порівняння можливих методів розв'язання поставленої задачі та обрано оптимальний варіант. Під час виконання магістерської кваліфікаційної роботи студент Шарандо А.А. проявив себе грамотним, кваліфікованим спеціалістом здатним приймати самостійно складні технічні рішення. Вважаю, що магістерська кваліфікаційна робота заслуговує оцінку «добре», а Шарандо А. А. – присвоєння кваліфікації «магістра» з комп'ютерних наук.

Науковий керівник Мещеряков Л.І., док. техн. наук, проф., проф. каф. ПЗКС

(прізвище, ім'я, по батькові, посада, місце роботи)

«___»_____20__ р. _____

(підпис)

РЕЦЕНЗІЯ
на магістерську роботу

студента Шарандо Артема Андрійовича

(прізвище, ім'я, по батькові)

курсу II групи _122м-21-1___

кафедри програмного забезпечення комп'ютерних систем

спеціальності 122 Комп'ютерні науки

освітньої програми

Тема роботи Обгрунтування структури інформаціїної системи спостерігання та прогнозу динамікі криптовалют

Стисла характеристика розділів роботи В першому розділі надано загальний опис наукової галузі, в якій ведеться дослідження, та наведені теоретичні дані. Другий розділ містить короткий огляд проблеми та потенційні шляхи її вирішення. Третій розділ містить в собі програмну реалізацію інформаціїної системи для спостерігання та аналізу динаміки криптовалют, аналіз розглянутих варіантів вирішення проблеми, а також результати застосування.

Пропозиції, внесені студентом, рівень їх наукового обґрунтування В даній кваліфікаційній роботі студентом надано декілька пропозицій щодо вирішення поставлених задач. Кожна з пропозицій була обґрунтована та підкріплена науковими даними.

Практичне значення роботи Результати роботи можуть бути застосовані для подальших наукових досліджень в даній сфері, а також вони можуть бути корисними для практичного використання.

Якість оформлення роботи Магістерська кваліфікаційна робота, яку подано на рецензію, виконана у повному обсязі у встановлений термін. Робота є добре структурованою та достатньо проілюстрованою. Викладена основна суть проблеми, що вирішується в ході виконання роботи, і шляхів її вирішення.

Недоліки в роботі відсутність примітивного користувацького інтерфейсу програми. Проте вказаний недолік не впливає на позитивне враження від роботи.

Загальний висновок Магістерська кваліфікаційна робота виконана у

(підготовленість студента до самостійної роботи як спеціаліста)

відповідності з завданням із дотриманням всіх вимог.

Оцінка магістерської роботи Робота заслуговує оцінки «добре», а студент Шарандо А.А. – присвоєння кваліфікації «магістра» з комп'ютерних наук.

Рецензент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада, місце роботи)

«___»_____20__ р. _____

(підпис)

Додаток В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім'я файла	**Опис**
Пояснювальні документи	
Диплом_ Шарандо.doc	Пояснювальна записка до магістерської роботи. Документ Word.
Диплом_ Шарандо.pdf	Пояснювальна записка до до магістерської роботи в форматі PDF
Програма	
quote-predictor.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_ Шарандо.ppt	Презентація до магістерської роботи