МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-наукової програми «Інформатика»
на тему: «Інтелектуальна технологія автоматизації управління та аналізу криптовалютних портфелів»
здобувача групи ІН.м-24 Рибки Олександра Валерійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Олександр РИБКА
(підпис)

Керівник,
старший викладач кафедри
комп'ютерних наук, к.т.н. Артем КОРОБОВ _____
(підпис)

Суми – 2023

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____

(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-наукової програми

«Інформатика»

здобувача групи ІН.м-24 Рибки Олександра Валерійовича

1. Тема роботи: «Інтелектуальна технологія автоматизації управління та аналізу криптовалютних портфелів»

затверджую наказом по СумДУ від «06» грудня 2023 року № 1412-VI

2. Термін здачі здобувачем кваліфікаційної роботи до 18 грудня 2023 року

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд існуючих методів та рішень для управління та прогнозу вартості криптовалютних активів 3) Розробка інтелектуальної технології автоматизації управління та аналізу криптовалютних портфелів 4) Аналіз результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 2023 р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд існуючих методів та рішень для управління та прогнозу вартості криптовалютних активів*		
3	*Розробка інтелектуальної технології автоматизації управління та аналізу криптовалютних портфелів*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 72 стр., 38 рис., 2 додатки, 2 табл., 22 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки стрімкий розвиток технологій суттєво впливає на криптовалютний ринок, що призводить до зростання популярності цифрових активів серед інвесторів та учасників ринку.

Об’єкт дослідження — процес автоматизації управління та аналізу криптовалютних портфелів за допомогою інтелектуальної технології.

Мета роботи — розробка інтелектуальної технології, яка спрямована на покращення ефективності управління криптовалютними активами та забезпечення більш точного аналізу ринкових умов.

Методи дослідження — у роботі використані методи машинного навчання, зокрема алгоритми XGBoost та ARIMA, для прогнозування ринкових тенденцій. Для розробки мобільного додатку використано відповідні технології та інструментарій для платформи Android.

Результати — розроблено інтелектуальну технологію, яка дозволяє точно прогнозувати значення активів та забезпечує зручний аналіз криптовалютних портфелів. Також реалізований мобільний додаток, що робить процес управління криптовалютними портфелями простим та доступним для користувачів.

КРИПТОВАЛЮТИ, ІНТЕЛЕКТУАЛЬНА СИСТЕМА, МАШИННЕ НАВЧАННЯ, ПРОГНОЗУВАННЯ, ANDROID.

ЗМІСТ

ВСТУП

Актуальність. Актуальність дослідження зумовлена стрімким розвитком технологій та їх впливом на криптовалютний ринок. Висока важливість ефективного управління та аналізу криптовалютних портфелів виникає в контексті постійного зростання популярності цифрових активів серед інвесторів та учасників ринку.

Об’єкт дослідження. Процес автоматизації управління та аналізу криптовалютних портфелів за допомогою інтелектуальної технології.

Предмет дослідження. Методи та підходи, які використовуються для оптимізації управління та аналізу криптовалютних портфелів, включаючи використання інтелектуальних технологій.

Гіпотеза. Застосування інтелектуальних технологій автоматизації може значно покращити ефективність управління криптовалютними активами та забезпечити більш точний аналіз ринкових умов.

Наукова новизна. На відміну від існуючих рішень в сфері управління та аналізу криптовалютних портфелів, розроблена технологія визначається своєю спроможністю точно прогнозувати значення активу та зручністю використання. Розроблений мобільний додаток робить процесс управління криптовалютними портфелями простим та зрозумілим, що відзначає новаторський характер даного дослідження.

Структура. Робота має структуру, яка включає в себе вступ, розділ з аналітичним оглядом, формулювання постановки задачі, вибір методології, розгляд програмного забезпечення інтелектуальної системи, висновки, список використаних джерел та додатків.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Поняття криптовалюти

Криптовалюта, представляючи інноваційний підхід до фінансових операцій, визначається своєрідністю криптографічних принципів та роботою в децентралізованих мережах. Це створює ряд унікальних характеристик, які відрізняють криптовалюти від традиційних фінансових інструментів [1].

Децентралізація, або відсутність центрального контролю, робить криптовалюти незалежними від політичних та економічних впливів. Це створює новий рівень свободи та автономії для користувачів. Анонімність транзакцій додає до цього рівень конфіденційності, хоча варто зауважити, що рівень анонімності може різнитися залежно від конкретної криптовалюти [2].

Незмінність та відкритість блокчейну дозволяють створити безпечне та надійне середовище для фінансових операцій. Історія транзакцій, яка зберігається в блокчейні, стає невіддільною частиною їхньої структури, забезпечуючи неможливість їх зміни чи видалення. Відкритість мережі дозволяє будь-якій особі приєднатися до криптовалютного екосистеми та брати участь у її розвитку.

Однак, незважаючи на переваги, важливо враховувати ризики, пов'язані з високою волатильністю криптовалютних ринків. Різке змінювання їх вартості може викликати значні фінансові втрати для інвесторів та учасників ринку. Також, питання щодо регулювання та легітимності використання криптовалют в різних країнах є актуальним аспектом для уважного розгляду [3,4].

Усі ці аспекти формують комплексне середовище, де криптовалюти відіграють ключову роль у трансформації традиційних підходів до фінансів та створенні нових можливостей для глобальних фінансових взаємодій.

1.2 Поняття криптовалютного портфелю

Криптовалютний портфель представляє собою збалансований набір різних криптовалют або цифрових активів, утримуваних інвестором чи користувачем [5]. Управління таким портфелем спрямоване на досягнення фінансових цілей, мінімізацію ризиків та максимізацію прибутків в умовах волатильного криптовалютного ринку.

Основні характеристики криптовалютних портфелів включають [6]:

- Диверсифікація активів: криптовалютний портфель може включати різні криптовалюти, такі як Bitcoin, Ethereum, альткойни та стейблкоїни. Це дозволяє розподіляти ризики та підвищувати стійкість портфеля до змін на ринку.
- Стратегії управління: Інвестор може використовувати різні стратегії управління, такі як довгострокові інвестиції, активне трейдінг, арбітраж, або копіювання управління іншими успішними трейдерами.
- Ребалансування портфеля: Час від часу інвестор може проводити ребалансування свого портфеля, змінюючи пропорції різних активів відповідно до стратегії та ринкових умов.
- Безпека та захист: Забезпечення безпеки та захисту криптовалютних активів шляхом використання холодних гаманців (cold wallets), двофакторної автентифікації та інших заходів стає важливою складовою управління криптовалютним портфелем.
- Аналіз та прогнозування: Використання аналітичних інструментів та методів прогнозування допомагає приймати обдумані рішення щодо складання та оптимізації портфеля.
- Врахування ризиків: Інвестор повинен бути свідомим ризиків, пов'язаних з волатильністю криптовалют та змінами на ринку. Це включає вивчення ринкових тенденцій, новин та факторів, що впливають на криптовалютні ціни.

Криптовалютні портфелі можуть бути управляні вручну інвесторами або

автоматизовано за допомогою різних інструментів та технологій. Ефективне управління криптовалютним портфелем вимагає не тільки фінансової експертизи, але і розуміння динаміки криптовалютних ринків та стратегій управління ризиками.

1.3 Дослідження актуальності проблеми

У сучасному світі, де технологічний прогрес рухається стрімкими темпами, питання управління та аналізу криптовалютних портфелів стає надзвичайно актуальним. Зростання популярності криптовалют, таких як Bitcoin, Ethereum та інші, призвело до збільшення числа інвесторів, що зацікавлені в цифрових активах.

Наслідком цього є необхідність у розробці та впровадженні інноваційних інструментів управління та аналізу, які враховують високий рівень волатильності криптовалютних ринків. Актуальність дослідження полягає в тому, щоб зрозуміти, як інтелектуальні технології можуть оптимізувати процес управління активами в умовах динамічного й непередбачуваного криптовалютного середовища.

Крім того, проблема прогнозування вартості криптовалютних активів набуває великого значення, оскільки інвестори та учасники ринку шукають ефективні інструменти для прийняття обдуманих фінансових рішень. Дослідження актуальності цієї проблеми дозволяє визначити можливості впровадження передових технологій та підходів для вдосконалення стратегій управління криптовалютними портфелями.

Отже, робота, спрямована на інтеграцію інтелектуальних технологій для автоматизації управління та прогнозування вартості активів у криптовалютних портфелях, не лише відповідає викликам сучасності, але й відкриває нові перспективи для оптимізації інвестиційного процесу в умовах швидкозмінюючогося фінансового ландшафту.

1.4 Аналіз технологій для управління криптовалютними портфелями

У світі криптовалют розширюється роль управління криптовалютними портфелями, і для цього існує різноманітні технології та інструменти. У наступній таблиці представлено порівняльний огляд деяких ключових технологій, які використовуються для управління криптовалютними портфелями, з плюсами та мінусами кожної.

Таблиця 1.1 - Технології для управління криптовалютними портфелями

Технологія	**Плюси**	**Мінуси**
Криптовалютні Гаманці	Висока безпека, особливо для холодних гаманців. Зберігання на довгий термін.	Менша зручність для активного трейдінгу. Втрата доступу до гаманця може призвести до втрати коштів.
Автоматизовані трейдінг боти	Автоматизоване управління портфелем за заданими стратегіями та алгоритмами.	Ризик втрати коштів через несприятливий розвиток програми чи ринку. Складність налаштування та регулярне оновлення.

Продовження таблиці 1.1

Технологія	**Плюси**	**Мінуси**
DeFi Платформи	Доступ до децентралізованих фінансових послуг, можливість здійснювати децентралізовані угоди.	Ризик уразливостей в смарт-контрактах. Обмежені можливості порівняно з традиційними фінансами. Низька ліквідність деяких ринків.
Трейдінгові платформи та біржі	Легкий доступ до ринку та можливість активного трейдінгу. Широкий вибір криптовалют.	Ризик втрати коштів через атаки на біржу, може бути складно вести детальний аналіз динаміки. Вимоги до особистої інформації.

1.5 Аналіз моделей та методів штучного інтелекту для прийняття рішень на основі ринкових даних

В контексті задачі передбачення ціни активу, обираючи модель, важливо враховувати низку факторів, що визначають її ефективність та придатність до даного конкретного завдання. В огляді моделей ШІ в фінансовому аналізі розглядаються різні підходи, такі як нейронні мережі, генетичні алгоритми та методи кластеризації. Аналізуються методи аналізу ринкових даних, такі як технічний та фундаментальний аналіз, а також статистичні методи та індикатори [7].

Далі удосконалюється аналіз алгоритмів для передбачення ринкових

трендів, зокрема XGBoost, Random Forest, ARIMA та інші, використовуючи їх для прогнозування змін на фінансових ринках. Розглянемо найпопулярніші з них.

1.5.1 XGBoost

XGBoost (Extreme Gradient Boosting) - це потужний алгоритм градієнтного бустінгу, який використовується для завдань регресії, класифікації та ранжування. Його ефективність та висока точність роблять його популярним в галузі машинного навчання та аналізу даних [8].

Основні риси XGBoost включають в себе використання градієнтного бустінгу, що дозволяє будувати модель, додаючи до неї слабкі моделі для коригування помилок попередніх. Алгоритм використовує регуляризацію L1 та L2 для контролю перенавчання та поліпшення стійкості моделі.

Важливим аспектом є розподіл роботи на паралельних обчисленнях, що покращує швидкість навчання. XGBoost використовує дерева рішень, що дозволяє виявляти складні нелінійні залежності в даних [9].

Алгоритм підтримує різні функції втрат для вирішення різних завдань. Метрики важливості ознак дозволяють визначити вплив кожної ознаки на прогнози моделі.

XGBoost широко використовується в конкурсах з прогнозування та аналізу даних, а також в різних сферах, таких як фінанси, біологія та технології. Його гнучкість, швидкість та точність роблять його популярним інструментом для вирішення різноманітних завдань машинного навчання. Машинне навчання використовує алгоритми для тренування моделі на знаходження патернів у наборі даних з мітками та ознаками, а потім використовує навчену модель для прогнозування міток на ознаках нового набору даних [10].

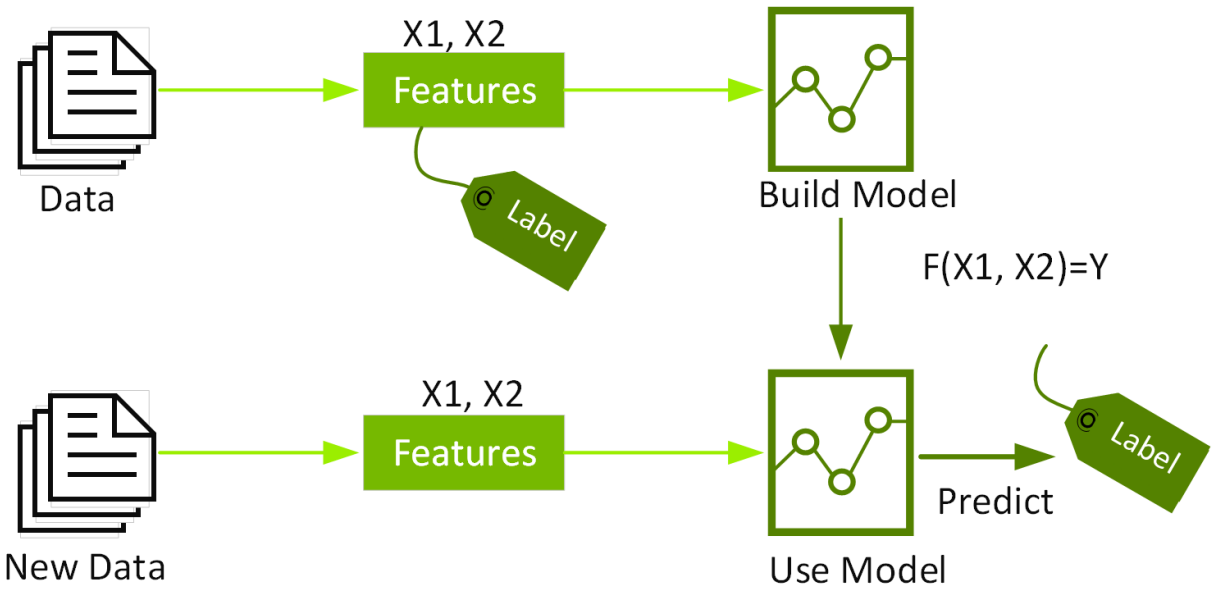

Рисунок 1.1 - Схема роботи XGBoost

XGBoost використовує різні параметри та методи для оптимізації алгоритму та досягнення кращих результатів та продуктивності, такі як [11]:

- Регуляризація: Параметр регуляризації (лямбда) використовується для обчислення показників подібності з метою зменшення чутливості до окремих даних та уникнення перенавчання.
- Скорочення (Gamma): Цей параметр визначається для порівняння виграшу. Гілка, де коефіцієнт зміцнення менший за значення гамми, видаляється. Це запобігає перенавчанню шляхом обрізання зайвих гілок та зменшення глибини дерев.
- Скетч звішених квантилів: Замість перевірки кожного можливого значення як порогу для поділу даних, використовуються лише звішені квантилі. Вибір квантилів виконується за допомогою алгоритму скетча, що оцінює розподіл по кількох системах в мережі.
- Паралельне навчання: Цей метод розділяє дані на блоки, які можна використовувати паралельно для побудови дерев або інших обчислень.
- Пошук розділення з урахуванням розрідженості: XGBoost ефективно обробляє розріджені дані, випробовуючи обидва напрямки поділу

та визначаючи напрямок за замовчуванням за допомогою оцінки схожості.

- Доступ із врахуванням кешу: Цей метод використовує кеш-пам'ять системи для обчислення показників схожості та вихідних значень, що покращує продуктивність моделі.
- Блоки для обчислень поза ядром: Цей метод працює з великими наборами даних, які не поміщаються в кеш або основну пам'ять та повинні зберігатися на жорсткому диску.

Ці параметри та методи роблять XGBoost потужним і ефективним алгоритмом для задач класифікації та регресії з використанням градієнтного бустінгу на основі дерев рішень.

1.5.2 ARIMA

ARIMA (Autoregressive Integrated Moving Average) - це статистичний метод для аналізу та прогнозу часових рядів. Модель ARIMA поєднує в собі три компоненти: авторегресію (AR), інтеграцію (I) та ковзне середнє (MA).

Основні компоненти ARIMA [12]:

- Авторегресія (AR): Ця частина враховує залежність між поточним значенням та попередніми значеннями в часовому ряді. Авторегресійна модель порядку p (позначається AR(p)) включає в себе попередні p значень часового ряду.
- Інтеграція (I): Ця частина враховує необхідність стабілізації часового ряду за допомогою диференціювання. Інтегральна частина позначається як I(d), де d - це кількість диференціацій, необхідних для зроблення ряду стаціонарним (тобто для забезпечення сталої дисперсії та середньої значення).
- Ковзне середнє (MA): Ця частина враховує залежність між поточним значенням та помилками моделі в попередній точці часу. Модель ковзного середнього порядку q позначається MA(q) та включає в себе q попередніх значень помилок.

ARIMA позначається як ARIMA(p, d, q), де p, d та q - це порядки відповідних компонент. Отже, прогнозоване значення у конкретний момент часу представляє собою суму константи, лінійної комбінації попередніх значень часового ряду (авторегресійний компонент), і лінійної комбінації попередніх прогнозованих помилок (компонент ковзного середнього). Це дозволяє моделі ARIMA враховувати залежності в часовому ряді та адаптуватися до його структури для точного прогнозування. ARIMA є ефективним методом для прогнозування часових рядів, особливо у випадках, коли ряд має тренд та сезонні коливання. Цей метод широко використовується в економіці, фінансах, кліматичних дослідженнях та інших галузях для аналізу та прогнозу динаміки часових рядів [13].

У моделі ARIMA, процедура диференціювання використовується для того, щоб зробити дані стаціонарними, тобто усунути будь-які тенденції або сезонні варіації. Стан стаціонарності означає, що статистичні властивості даних, такі як середнє значення та дисперсія, залишаються сталими у часі. Наприклад на Рисунку явно видно, що дані мають тенденцію, відповідно підтверджується нестаціонарність ряду [14].

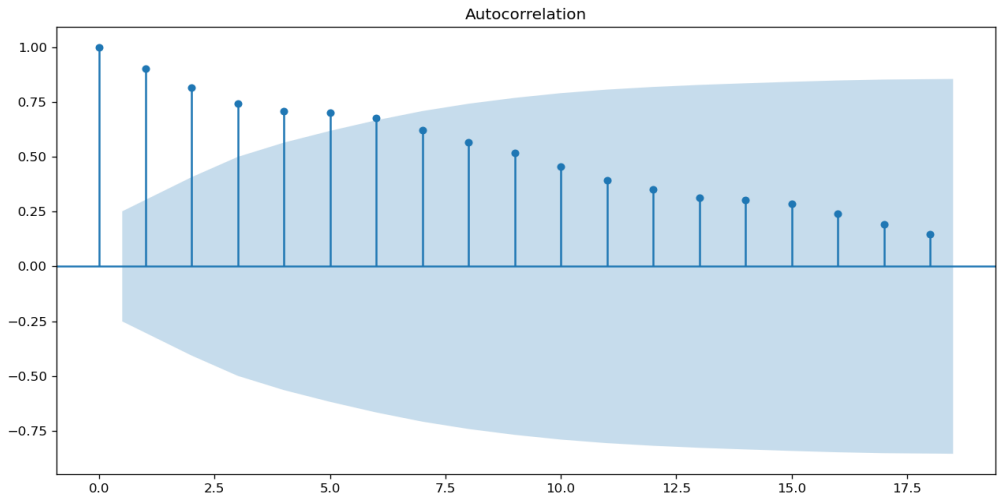

Рисунок 1.2 - Схема графіку з вираженою тенденцією

Багато економічних та ринкових даних демонструють тенденції або повторювані патерни, які можуть негативно вплинути на регресійну модель, особливо якщо присутня сезонність. Сезонність виявляється у регулярних та передбачуваних зразках, що повторюються через календарний період. Під час виявлення тенденцій та відсутності стаціонарності важко ефективно проводити обчислення [15].

Таким чином, введення етапу диференціювання дозволяє виправити ці проблеми. Після процедури диференціювання дані стають менш схильними до тенденцій та сезонних варіацій, що полегшує роботу регресійної моделі та сприяє більш ефективному аналізу. На рисунку ми бачимо, що дотримується стаціонарність ряду.

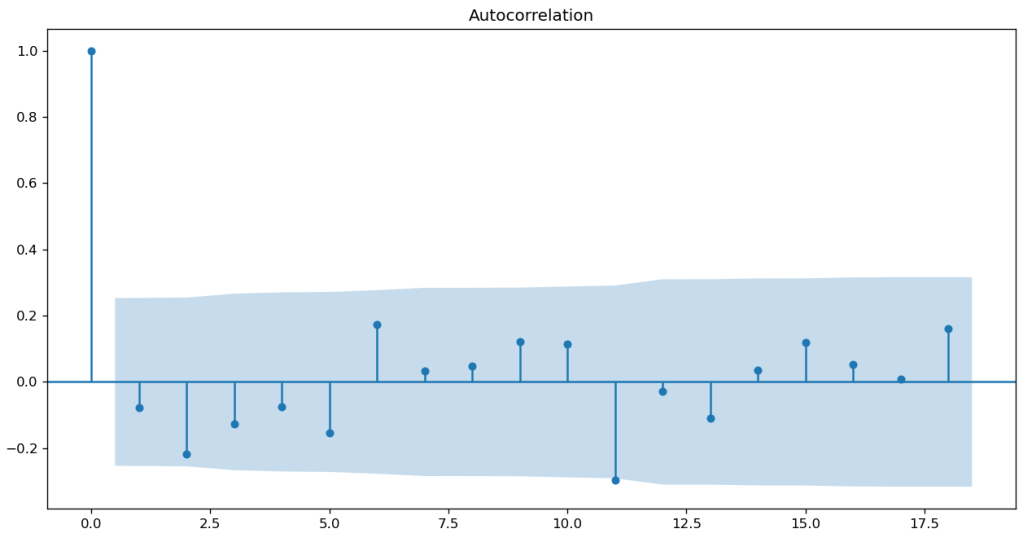

Рисунок 1.3 - Схема графіку зі стаціонарним рядом

1.5.3 LSTM мережа

LSTM (англ. Long short-term memory) – це підтип рекурентних нейронних мереж, який усуває проблему втрати довгострокових зв'язків [22]. Мережі LSTM складаються з аналогічного повторювального модуля, але вони

відрізняються в тому, що замість одного, ця архітектура містить чотири шари, які взаємодіють між собою.

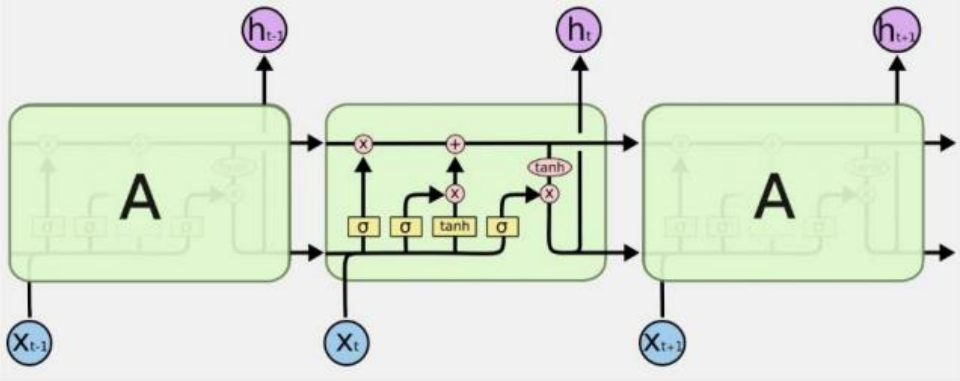

Рисунок 1.4 - Повторювальний модуль LSTM мережі

Однією з ключових особливостей LSTM-мережі є стан комірки, позначений горизонтальною лінією у верхній частині наступного рисунка 1.4. Цей стан проходить операції поступово з невеликими лінійними взаємодіями.

LSTM-мережа має здатність вилучати та додавати інформацію до стану комірки, керовану спеціальними структурами, відомими як Гейти. Гейт - це механізм, що дозволяє вибірково пропускати інформацію, має шар із сигмоїдальною функцією активації і операцією поточечного множення. Вихід цього шару - 0 або 1, де 0 вказує на пропускання інформації, а 1 - на її передачу.

На початковому етапі мережа вирішує, яку інформацію видалити, використовуючи гейт забуття з сигмоїдальним виходом від 0 до 1. Наступним етапом є визначення того, яку інформацію слід зберегти в стані комірки. Цей етап можна умовно розділити на дві частини: вхідний гейт, що вирішує, які значення оновлювати за допомогою сигмоїдальної функції, і шар з гіперболічним тангенсом, який створює новий вектор для додавання до стану комірки.

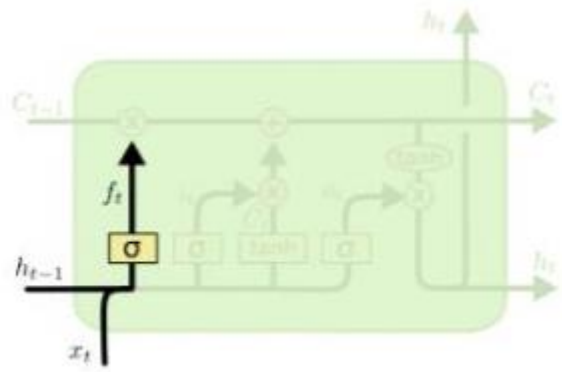

Рисунок 1.5 - Схема гейту забування

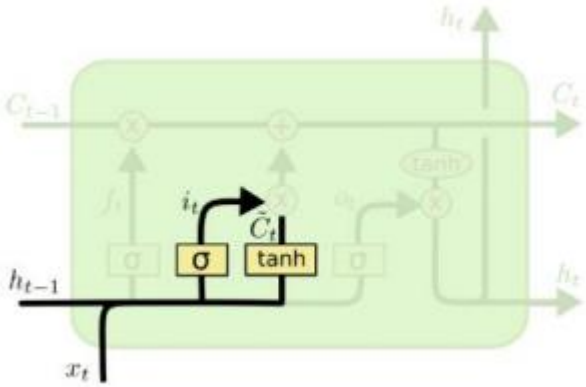

Рисунок 1.6 - Схема гейту записа

Останнім етапом є рішення нейронною мережею щодо вихідних значень. Вихід представляє собою стан комірки після фільтрації. Спочатку сигмоїдальний шар визначає, які елементи стану комірки слід передати на вихід. Потім сам стан комірки перетворюється за допомогою tanh-шару до інтервалу від -1 до 1 і множиться на вихід сигмоїдального шару, щоб вивести лише те, що було вирішено вивести.

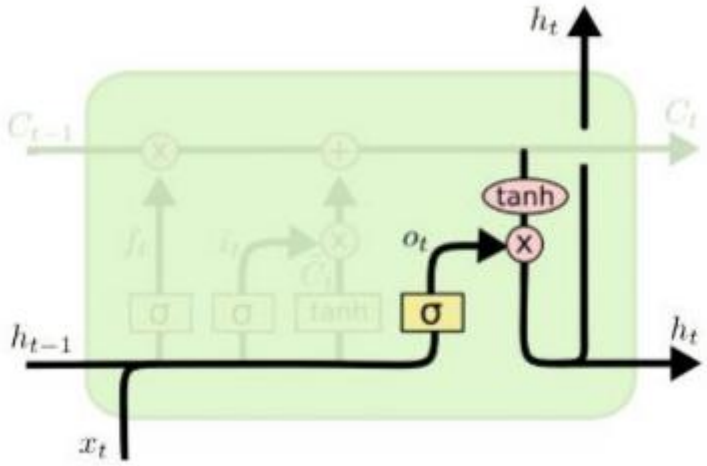

Рисунок 1.7 - Вихідний шар LSTM мережі

1.6 Аналіз індикаторів технічного аналізу

Технічний аналіз (ТА) - це метод аналізу, який передбачає ринкову поведінку на основі попередньої динаміки ціни та торгових об'ємів. Цей підхід широко використовується для акцій та інших активів на традиційних фінансових ринках, а також у торгівлі цифровими валютами на ринку криптовалют.

Однак відмінності від фундаментального аналізу, який враховує різні фактори, пов'язані з ціною активу, ТА сфокусований виключно на історичній динаміці цін. Це робить його інструментом для вивчення коливань цін та обсягів активів, і багато трейдерів використовують його для виявлення трендів та можливостей для торгівлі.

Початкові форми технічного аналізу з'явилися в 17 столітті в Амстердамі та у 18 столітті в Японії. Сучасний аналіз пов'язується з Чарльзом Доу, фінансовим журналістом та засновником The Wall Street Journal. Його робота та теорія Доу визначили подальший розвиток технічного аналізу.

У попередні часи технічний аналіз використовував рукописні таблиці та ручні розрахунки, але з розвитком технологій став широко поширеним. Зараз це важливий інструмент для інвесторів та трейдерів.

1.6.1 RSI

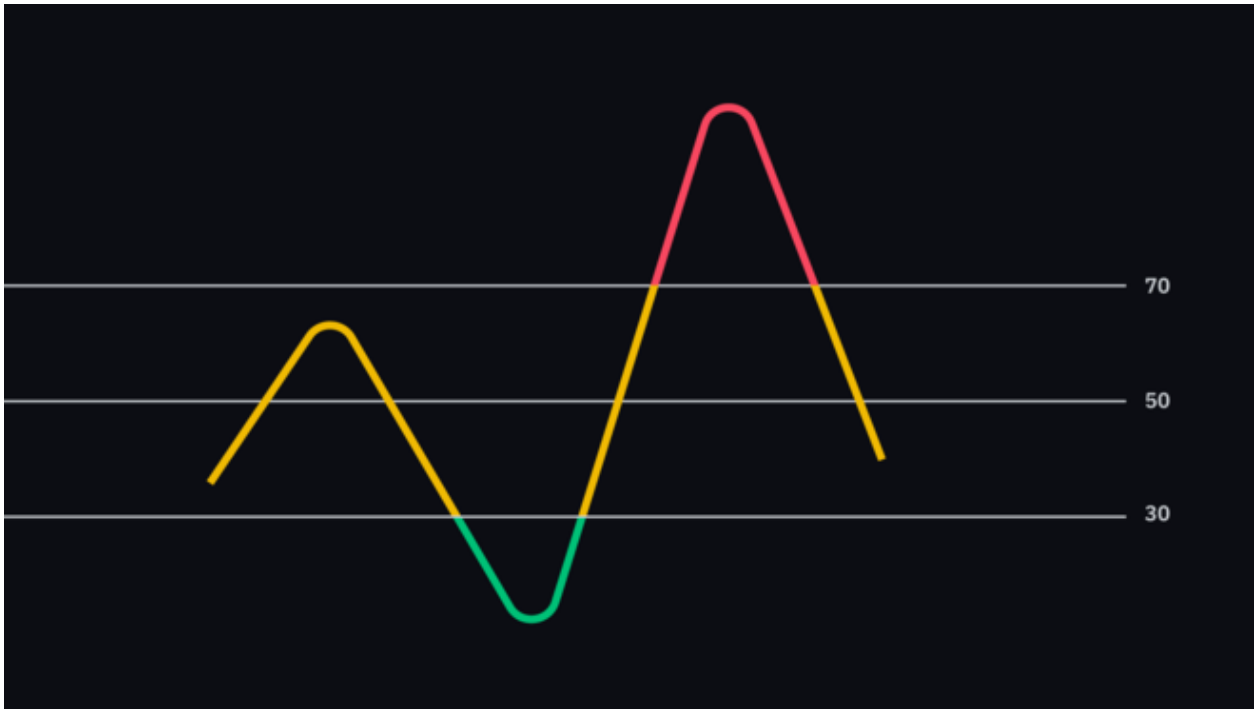

Рисунок 1.8 - Схематичне зображення RSI

RSI - це індикатор імпульсу, який надає показники для оцінки, чи актив є перекупленим чи перепроданим. Для досягнення цього він вимірює розмір недавніх змін ціни за певний період (зазвичай 14 попередніх періодів). Отримані дані представляються у вигляді осцилятора, який може приймати значення від 0 до 100.

RSI, як індикатор імпульсу, вказує на силу зміни ціни. Зростання індикатора при зростанні ціни свідчить про сильну тенденцію до підвищення та збільшення інтересу покупців. На відміну від цього, зменшення індикатора при рості ціни може свідчити про можливий відхід від покупок та перехід контролю на сторону продавців [16].

Традиційно RSI використовується для визначення, коли актив стає перекупленим (якщо RSI перевищує 70) або перепроданим (якщо RSI менше

30). Проте, важливо розуміти, що ці значення не є прямими сигналами для купівлі чи продажу. Як і інші методи технічного аналізу, RSI може давати неточні сигнали, і рішення про угоду краще приймати, враховуючи інші фактори.

1.6.2 Ковзна середня (MA)

Рисунок 1.9 - Схематичне зображення MA

Ковзна середня використовується для згладжування цінової дії, виключаючи ринковий шум і виділяючи напрямок тренду. Це індикатор запізнювання, оскільки він базується на минулих цінових даних [16].

Проста ковзна середня (SMA або MA) та Експоненціальна ковзна середня (EMA) є двома основними типами ковзних середніх. SMA розраховується на основі цінових даних за певний період і представляє собою середнє значення цих цін. Наприклад, 10-денна SMA визначається середньою ціною за останні 10 днів. З іншого боку, EMA приділяє більше ваги останнім даним, реагуючи швидше на недавні зміни ціни.

Важливо враховувати, що ковзна середня є запізнілим індикатором, і

його затримка залежить від періоду. Трейдери використовують відношення ціни до ковзних середніх для оцінки поточного тренду. Наприклад, якщо ціна тривалий час залишається вище 200-денної SMA, це може вказувати на бичачий ринок.

Трейдери також використовують перетини ковзних середніх як сигнали для угод. Наприклад, перетин 100-денної SMA з 200-денною SMA може служити сигналом на продаж, вказуючи на можливу зміну тренду.

1.6.3 Конвергенція/дивергенція ковзних середніх (MACD)

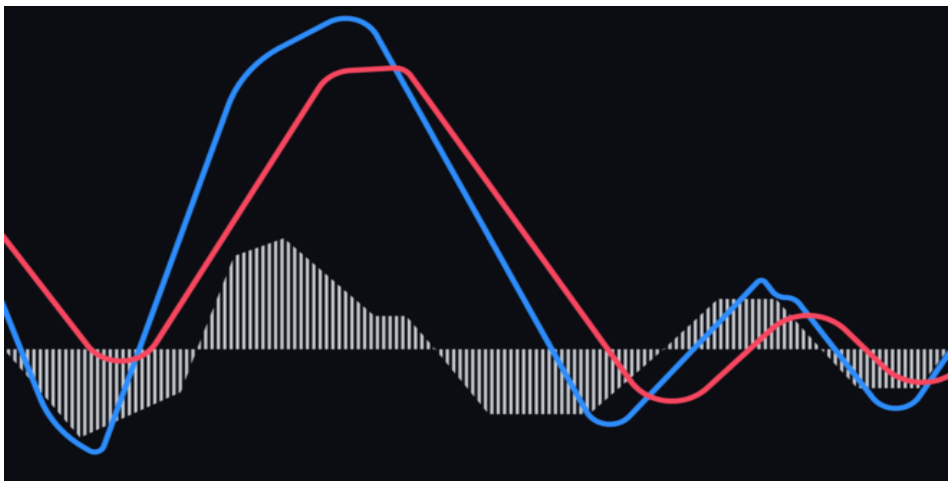

Рисунок 1.10 - Схематичне зображення MACD

MACD - це технічний індикатор, призначений для прогнозу майбутнього цінового руху активу на основі взаємозв'язку двох ковзних середніх. Він складається з лінії MACD та сигнальної лінії. Лінія MACD обчислюється шляхом віднімання 26-денної EMA від 12-денної EMA, і результат накладається на графік 9-денної EMA, що виступає в ролі сигнальної лінії. Деякі інструменти графіків також мають гістограму, яка відображає відстань між цими лініями [16].

Аналізуючи дивергенцію між MACD і ціновим рухом, трейдери отримують уявлення про силу поточного тренду. Наприклад, новий максимум ціни при низьких показниках MACD може свідчити про можливий розворот

ринку. Ці індикатори дозволяють зробити висновок, що висока ціна при слабкому імпульсі може призвести до відкату або розвороту.

Трейдери також використовують MACD для визначення точок перетину між лінією MACD та сигнальною лінією. Зазвичай, перетин знизу вгору вважається сигналом на покупку, а перетин зверху вниз - сигналом на продаж.

MACD часто використовується разом з RSI для отримання більш повного технічного аналізу ринку.

1.7 Постановка задачі

На основі вищезазначеної інформації щодо інтелектуальних технологій та технічного аналізу, які можна використати для аналізу криптовалютних активів, можна зробити висновок, що сучасні інформаційно-інтелектуальні технології можуть значно полегшити прогнозування вартості активів та управління криптовалютними портфелями.

Основною метою цієї роботи є створення інтелектуальної технології для ефективного управління та аналізу криптовалютних портфелів. Технологія буде забезпечувати прогноз фінансового стану активів на основі різноманітних параметрів криптовалютного ринку, в зв’язку з чим оператор зможе більш результативно управляти управляти портфелем.

На вхід технологія отримуватиме дані, такі як цінова динаміка криптовалют, обсяги торгів, ціна відкриття/закриття, максимальна та мінімальна ціна в обраний проміжок часу та інші ключові показники. Додатково будуть вираховуватися інші ознаки, які використовуються для технічного аналізу активів.

Етапи виконання задачі можуть включати:

- Вибір відповідної моделі для аналізу криптовалютних даних та ринкових трендів.
- Підготовка та обробка фінансових даних для тренування та оптимізації моделі.

- Розробка інтерфейсу для взаємодії користувача з технологією та для відображення результатів аналізу активів.
- Тестування роботи системи в режимі реального часу з використанням історичних та поточних даних.

Такий підхід дозволить ефективно використовувати інтелектуальну технологію для оптимізації управління та прийняття рішень в області криптовалютних інвестицій.

2. ОГЛЯД ТА ВИБІР ТЕХНОЛОГІЙ ДЛЯ РІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Огляд технологій розробки для платформи Android

Android, як операційна система з відкритим кодом, популярна на різних мобільних пристроях і планшетах, створена Google на базі ядра Linux та інших відкритих технологій [17].

Ця ОС забезпечує єдиний підхід до розробки додатків для мобільних пристроїв, дозволяючи розробникам створювати застосунки для різних платформ, використовуючи один код.

Android підтримує потужну графіку, аудіо і відео, а також різноманітні можливості підключення. Його інтерфейс розроблений інтуїтивно зрозумілим для користувача.

За допомогою віджетів, тем і шпалер можна налаштовувати інтерфейс, а також користуватися потужними функціями безпеки, такими як ізольоване програмне середовище і безпечне апаратне сховище ключів.

Пристрої Android поставляються з різноманітними попередньо встановленими додатками, багато з яких можна завантажити безкоштовно. Крім того, користувачі можуть отримати доступ до Google Play Store для завантаження тисяч програм і цифрового контенту.

Ця ОС стала однією з найпопулярніших завдяки своїй універсальності

та великому вибору програм.

Мільйони людей використовують Android щодня, роблячи його ідеальним для тих, хто хоче залишатися на зв’язку та працювати продуктивно в дорозі.

Android, завдяки своєму відкритому коду та постійному розвитку функцій, обов'язково залишиться лідером серед мобільних операційних систем у найближчі роки.

Що стосується розробки на платформі Android, одним із найпопулярніших інтегрованих середовищ розробки (IDE) для цієї задачі є IntelliJ IDEA. Обрання Intellij IDEA для розробки на Android має кілька обгрунтованих причин [18].

По-перше, IntelliJ IDEA є потужним інструментом, який надає розширені можливості редагування коду та інтеграцію з системами контролю версій. Він надає зручний інтерфейс для розробників і пропонує широкий набір інструментів для підтримки різних мов програмування.

По-друге, IntelliJ IDEA забезпечує підтримку плагінів, що робить його ідеальним вибором для розробки Android-додатків. Засоби для створення і тестування додатків для Android вбудовані в IntelliJ IDEA, що робить процес розробки ефективнішим.

По-третє, IntelliJ IDEA має велику спільноту користувачів і широкий спектр документації, що полегшує навчання та вирішення проблем. Відмінна підтримка інструментів для відлагодження, автоматичне доповнення коду та інші продуктивні функції роблять його популярним серед розробників.

Таким чином, обираючи IntelliJ IDEA для розробки на Android, розробники отримують надійний інструментарій з багатьма можливостями, що сприяє швидкій і ефективній розробці мобільних додатків.

2.2 Мови програмування додатку

У сфері розробки Android, для написання програмного коду

використовуються дві основні мови програмування: Java і Kotlin. Обидві мови є популярними серед розробників, але останнім часом Kotlin набуває все більшої популярності та визнання в спільноті Android-розробників [19].

Java була основною мовою для розробки Android-додатків протягом тривалого часу. Вона має широку підтримку, обширну документацію та велику кількість сторонніх бібліотек. Багато розробників мають досвід у роботі з Java, і ця мова продовжує бути популярною для Android-розробки.

З іншого боку, Kotlin - це сучасна мова програмування, яка була офіційно підтримана Google для розробки Android-додатків з 2017 року. Kotlin пропонує безліч переваг, таких як компактність коду, уніфікація з Java, безпека та виразність. Вона інтегрується з Java, що дозволяє поступово переходити від Java до Kotlin в проектах.

Однією з основних переваг Kotlin є те, що вона забезпечує більш чітку та ефективну розробку завдяки коротшому коду та покращеній безпеці типів. Kotlin також підтримує функціональну парадигму, що важливо для сучасного програмування.

Отже, обираючи між Java та Kotlin для розробки Android-додатків, я віддаю перевагу Kotlin через його сучасність, ефективність та виразність коду.

2.3 Вибір технології для реалізації машинного навчання

При реалізації проекту з машинного навчання, вибір технології є ключовим етапом, що впливає на успішність та ефективність роботи системи. У нашому випадку, я звертаю увагу на мови програмування та бібліотеки, які найкраще відповідають моїм потребам.

Мови програмування в сфері машинного навчання грають важливу роль у розробці та реалізації алгоритмів, моделей та систем штучного інтелекту. Декілька ключових мов використовуються для реалізації проектів машинного навчання, кожна з яких має свої переваги та особливості.

Python. Python вважається однією з найпопулярніших мов

програмування для машинного навчання. Вона відзначається простотою синтаксису, що полегшує розробку та розвиток проектів. Багато бібліотек, таких як TensorFlow, PyTorch, та scikit-learn, надають підтримку для машинного навчання в Python.

R. R часто використовується в статистичному аналізі та наукових дослідженнях. Він надає велику кількість пакетів для статистичного моделювання та візуалізації даних. R особливо популярний серед статистиків та дослідників, які працюють у галузі машинного навчання.

Java. Java залишається популярним вибором для розробників, які працюють над великими та масштабованими системами машинного навчання. Вона використовується в багатьох корпоративних середовищах та має потужний інструментарій для розробки.

Julia. Julia визначається високою продуктивністю та швидкістю виконання. Вона набирає популярність в галузі наукових обчислень та обробки даних.

C++. C++ використовується для оптимізованих реалізацій алгоритмів машинного навчання, особливо в області вбудованих систем та задач з великим обсягом даних.

Для реалізації поставленої задачі я вирішив надати перевагу на користь мови програмування Python. Це обумовлено численними перевагами, які пропонує Python у сфері аналізу даних та машинного навчання.

Переваги використання Python для машинного навчання:

- Велике співтовариство та екосистема. Python налічує широке та активне співтовариство розробників, що робить його ідеальним вибором для обміну інформацією та вирішення проблем у галузі машинного навчання. Крім того, велика кількість бібліотек, таких як NumPy, Pandas, та Matplotlib, дозволяють ефективно працювати з даними та створювати високоефективні моделі.
- Простота та читабельність коду. Python славиться своєю

простотою та лаконічністю синтаксису. Це робить код читабельним та легким для розуміння, що важливо при роботі з складними алгоритмами машинного навчання.

- Широкий вибір бібліотек для машинного навчання. У Python існує велика кількість бібліотек для машинного навчання, таких як TensorFlow, PyTorch, та Scikit-Learn. Це дозволяє вибирати оптимальний інструментарій для конкретних завдань.

У контексті моєї роботи, я обрав використання бібліотеки XGBoost, яка володіє високою ефективністю у вирішенні завдань класифікації та регресії, і оптимально взаємодіє з мовою програмування Python.

Такий підхід дозволяє поєднати потужність мови програмування та високооптимізованої бібліотеки, забезпечуючи необхідний інструментарій для успішної і розвинутої реалізації завдань машинного навчання.

Також важливим євибір середовища для розробки та експериментів з моделями машинного навчання. Одним зі зручних інструментів, який я обрав, є Google Colab (Colaboratory).

Google Colab - це безкоштовне середовище для виконання коду на мові Python, яке працює у хмарному режимі. Однією з його ключових переваг є можливість використання обчислювальних ресурсів GPU та TPU безкоштовно. Це особливо важливо при великих обчисленнях, таких як тренування складних моделей машинного навчання.

Крім того, Google Colab інтегрований із сервісами Google Drive, що спрощує збереження та обмін даними та кодом між учасниками проекту. Це забезпечує зручну спільну роботу та сприяє легкості обміну результатами експериментів при командній роботі.

Таким чином, вибір Google Colab для моєї роботи обумовлений його зручністю, можливістю використання обчислювальних ресурсів та інтеграцією з іншими інструментами Google, що сприяє ефективній та продуктивній розробці та експериментаціям у галузі машинного навчання.

3. РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНОЇ ТЕХНОЛОГІЇ АВТОМАТИЗАЦІЇ УПРАВЛІННЯ ТА АНАЛІЗУ КРИПТОВАЛЮТНИХ ПОРТФЕЛІВ

3.1 Створення вибірки навчальних та тестових даних

Для створення вибірки даних, я буду використовувати різні бібліотеки даних, такі як NumPy, Pandas, Matplotlib і XGBoost. NumPy і Pandas використовуються для обробки та аналізу даних. Алгоритми машинного навчання, такі як XGBoost, зазвичай використовуються для регресії, класифікації та ранжування. Візуалізації можна створювати за допомогою Matplotlib, бібліотеки графічних зображень.

```
import os
import numpy as np
import pandas as pd
import xgboost as xgb
import matplotlib.pyplot as plt
from xgboost import plot_importance, plot_tree
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV
```

Рисунок 3.1 - Імпорт бібліотек

Також було використано додаткові бібліотеки візуалізації, корисні для створення інтерактивних і динамічних діаграм. Серед них Plotly, Plotly.io та Plotly.graph_objects. Для роботи офлайн використовується офлайн-модуль Plotly, а для завантаження файлу JavaScript для Plotly використовується функція download_plotlyjs.

```
import plotly as py
import plotly.io as pio
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
```

Рисунок 3.2 - Імпорт бібліотек для візуалізації

Підготування історичних даних. Тестові дані було взято для торгової пари BTCUSDT з криптобіржі Binance за допомогою API. Було отримано всі доступні дані. Проте, щоб зменшити обсяг даних для обробки та відсікти той час, коли популярність у криптовалют була занадто мала, було прийнято рішення обрізати період часу та почати з 2019 року.

	Open time	Open	High	Low	Close	Volume	Close time	Quote asset volume	Number of trades	Taker buy base asset volume	Taker buy quote asset volume	Unused field
0	1502942400000	4261.48	4349.99	4261.32	4349.99	82.088865	1502956799999	3.531943e+05	334	64.013727	275647.421911	0
1	1502956800000	4333.32	4485.39	4333.32	4427.30	63.619882	1502971199999	2.825012e+05	248	58.787633	261054.051154	0
2	1502971200000	4436.06	4485.39	4333.42	4352.34	174.562001	1502985599999	7.742388e+05	858	125.184133	555419.758061	0
3	1502985600000	4352.33	4354.84	4200.74	4325.23	225.109716	1502999999999	9.652911e+05	986	165.036363	707808.200922	0
4	1503000000000	4307.56	4369.69	4258.56	4285.08	249.769913	1503014399999	1.079545e+06	1001	203.226685	878286.968557	0

Рисунок 3.3 - Структура початкових вхідних даних

Діаграма OHLC. Щоб визначити історичні ціни, було намальовано діаграму OHLC (відкрити/максимум/низько/закрити). Також у області під OHLC було намальовано діаграму обсягу, яка показує кількість акцій, що торгуються щодня.

```
fig = make_subplots(rows=2, cols=1)
```

```
fig.add_trace(go.Ohlc(x=data_df.Date,
                      open=data_df.Open,
                      high=data_df.High,
                      low=data_df.Low,
                      close=data_df.Close,
                      name='Price'), row=1, col=1)
fig.add_trace(go.Scatter(x=data_df.Date, y=data_df.Volume, name='Volume'), row=2, col=1)
fig.update(layout_xaxis_rangeslider_visible=False)
fig.show()
```

Рисунок 3.4 - Код для малювання діаграми OHLC

За допомогою функції Make_subplots() було створено фігуру з двох рядків і одного стовпця. Далі за допомогою add_trace() було додано два графіки до фігури: один для даних про курс активу і один для даних про обсяги. Функція go.Ohlc() створює інтерактивну свічкову діаграму на основі

даних про ціну акцій, де параметр x встановлюється у стовпці «Дата», а параметри відкриття, максимуму, мінімуму та закриття — у відповідних стовпцях. Щоб позначити графік, ми встановлюємо параметр імені «Ціна». Функція go.Scatter() використовується для побудови даних об’єму з параметрами x і y, встановленими у стовпцях «Дата» та «Об’єм» відповідно, а параметром імені встановлено значення «Об’єм», щоб позначити графік (див. Рисунок 3.5).

Рисунок 3.5 - Діаграма OHLC та об’єм

Декомпозиція. Розкладаємо дані про ціну активу за допомогою методу декомпозиції STL і малюємо розкладені компоненти.

```
import statsmodels.api as sm
import matplotlib.pyplot as plt

df_close = data_df[['Date', 'Close']].copy()
df_close = df_close.set_index('Date')

# Decompose the time series
decomposition = sm.tsa.seasonal_decompose(df_close, model='additive', period=365)

# Access the decomposed components
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

# Plot the components
plt.figure(figsize=(12, 8))
plt.subplot(411)
plt.plot(df_close, label='Original')
plt.legend()
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend()
plt.subplot(413)
plt.plot(seasonal, label='Seasonal')
plt.legend()
plt.subplot(414)
plt.plot(residual, label='Residual')
plt.legend()
plt.tight_layout()
plt.show()
```

Рисунок 3.6 - Код декомпозиції

Використовуючи синтаксис [[‘Date’, ‘Close’]], в першому рядку коду було створено новий DataFrame під назвою df_close, який містить лише стовпці «Date» і «Close» оригінального DataFrame. Використовуючи функцію copy(), було зроблено копію вибраного DataFrame, не змінюючи оригінал. Потім було використано set_index(), щоб зробити стовпець «Дата» індексом DataFrame.

Змінна під назвою decomp зберігає результат декомпозиції STL у стовпці « Close» DataFrame, використовуючи період у 365 днів.

Функція plot() малює розкладені компоненти часового ряду, включаючи тренд, сезонні та залишкові компоненти. Ділянка встановлюється на 20

дюймів у ширину та 8 дюймів у висоту за допомогою функції set_size_inches(). На графіку показано розкладені компоненти даних про курс активу, що корисно для визначення тенденцій і закономірностей.

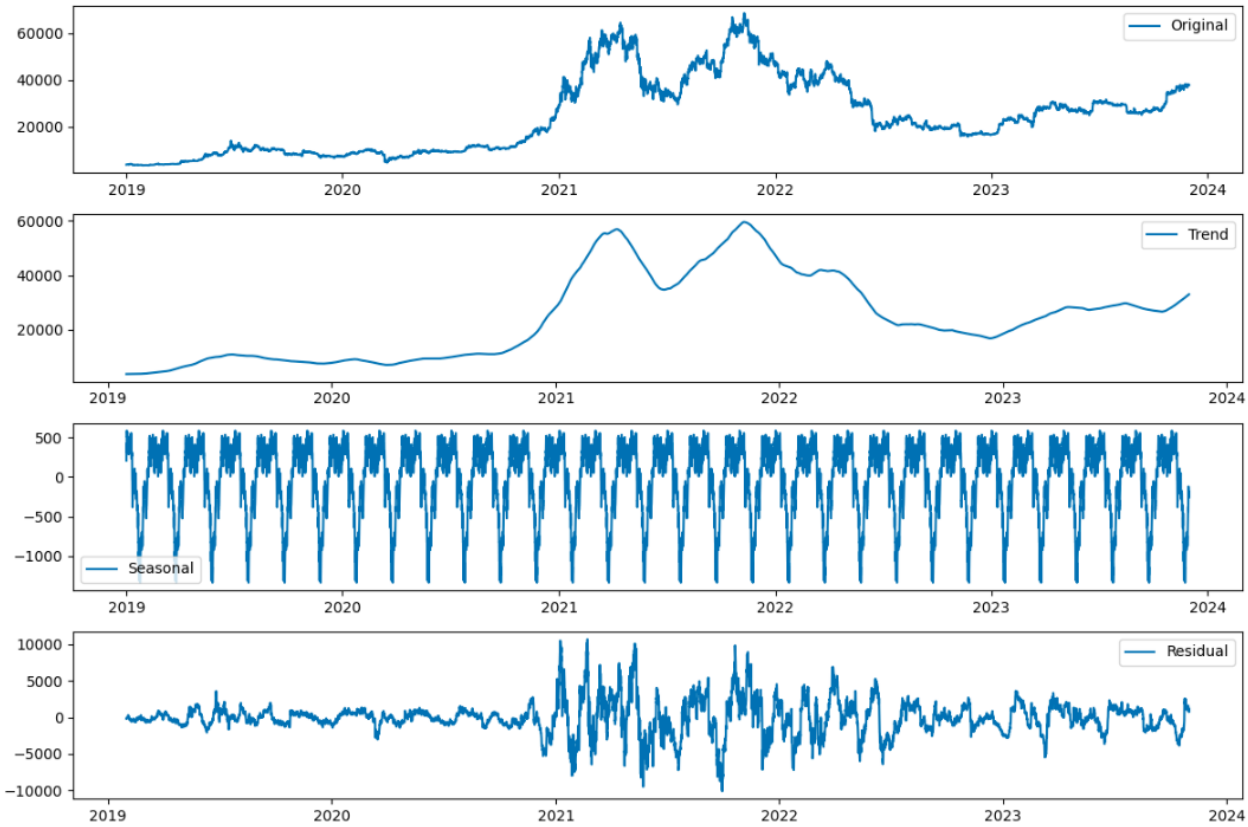

Рисунок 3.7 - Декомпозиція

Індикатори технічного аналізу. Цей фрагмент коду стосується аналізу та візуалізації фінансових часових рядів даних.

Використовуючи функції ewm() і rolling() у стовпці ціни «Close» pandas DataFrame (df), я обчислюю експоненціальне ковзне середнє (EMA) і просте ковзне середнє (SMA) для різних часових вікон (9, 5, 10, 15 і 30). Shift() зсуває обчислені значення на один період назад, щоб вони узгоджувалися з відповідною датою.

```
df['EMA_9'] = df['Close'].ewm(9).mean().shift()
df['SMA_5'] = df['Close'].rolling(5).mean().shift()
df['SMA_10'] = df['Close'].rolling(10).mean().shift()
df['SMA_15'] = df['Close'].rolling(15).mean().shift()
df['SMA_30'] = df['Close'].rolling(30).mean().shift()
```

Рисунок 3.8 - Розрахунок технічних показників

Далі генерую новий графік за допомогою модуля plotly go. За допомогою функції add_trace() до графіка можна додати кілька трас, кожна з яких представляє одне з обчислених ковзних середніх і початкову ціну «Закриття». Кожна траса має іншу назву, яка відображається в легенді.

Функція show() відображає отриманий графік в інтерактивному вікні. Отриманий графік показує різні ковзні середні та ціну «закриття» з плином часу, тому їх легко порівняти.

Рисунок 3.9 - Графік з середніми скользящими

Індекс відносної сили. Щоб передбачити, перекуплена чи перепроданий актив, я додам індикатор RSI.

```
import pandas_ta as ta
data_df['RSI_14'] = data_df.ta.rsi(length = 14)
```

```
fig = go.Figure(go.Scatter(x=data_df.Date, y=data_df.RSI_14, name='RSI_14'))
fig.show()
```

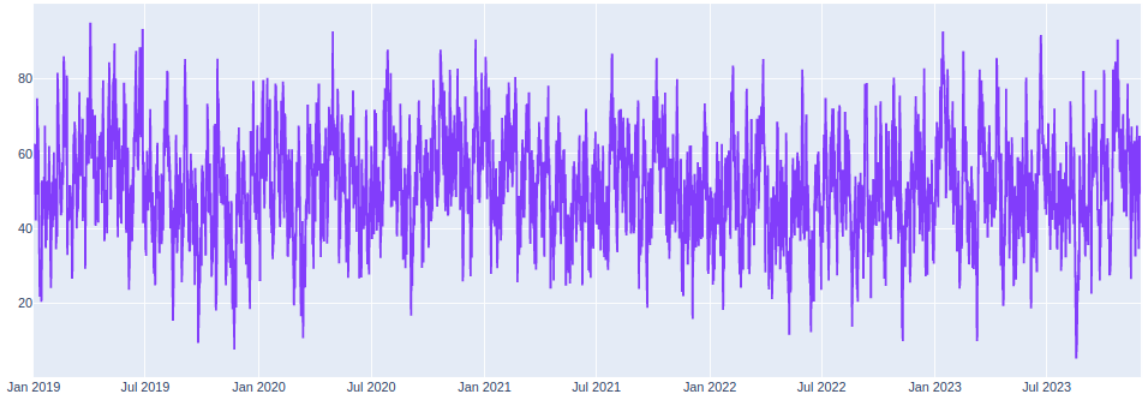

Рисунок 3.10 - Графік індексу RSI

MACD. Цей код обчислює та візуалізує показник дивергенції ковзного середнього (MACD) фінансового часового ряду.

```
EMA_12 = pd.Series(data_df['Close'].ewm(span=12, min_periods=12).mean())
EMA_26 = pd.Series(data_df['Close'].ewm(span=26, min_periods=26).mean())
data_df['MACD'] = pd.Series(EMA_12 - EMA_26)
data_df['MACD_signal'] = pd.Series(data_df.MACD.ewm(span=9, min_periods=9).mean())
```

Рисунок 3.11 - Розрахунок MACD

Перший блок коду обчислює два ряди експоненціальних ковзних середніх (EMA) для стовпця ціни «Закрити» df pandas DataFrame. Функція ewm() обчислює EMA за визначений період часу (12 або 26 у цьому випадку), а параметр min_periods визначає мінімальну кількість спостережень, необхідних для правильного розрахунку EMA. Для зберігання значень EMA для 12 і 26 періодів створено дві серії pandas.

Новий стовпець «MACD» додається до DataFrame df, що містить MACD, розрахований як різниця між двома рядами EMA. Крім того, сигнальна лінія

обчислюється шляхом обчислення 9-періодної EMA MACD і зберігається як новий стовпець «MACD_signal».

Новий графік створюється за допомогою функції make_subplots() з модуля plotly.subplots у другому блоці коду. Він складається з двох рядків і одного стовпця, причому верхній рядок містить ціну «закриття» акції, 12-періодну EMA та 26-періодну EMA, а нижній рядок містить MACD і сигнальну лінію.

Функція add_trace() додає окремі траси на графік, кожна з яких представляє одне з обчислених значень. У легенді ділянки для кожного сліду вказується різна назва. На графіку позиція кожної траси визначається параметрами row і col.

Функція show() відображає отриманий графік в інтерактивному вікні. Трейдери та інвестори можуть використовувати цей графік, щоб оцінити силу та напрямок тренду акцій, дивлячись на різні ковзні середні та індикатор MACD з часом.

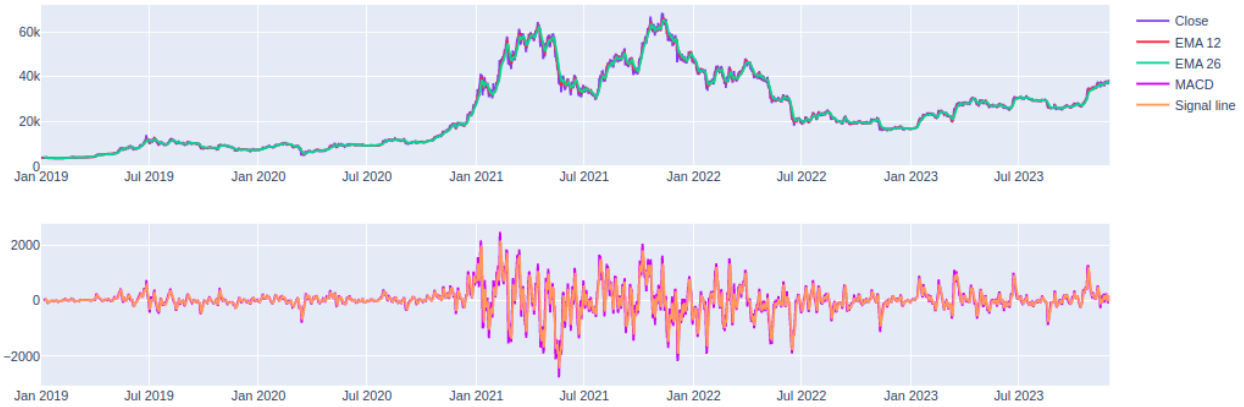

Рисунок 3.12 - Графік MACD

Цей код переміщує всі значення на одну позицію вгору в pandas DataFrame, зсуваючи стовпець «Закрити» на один період у майбутнє.

```
data_df['Close'] = data_df['Close'].shift(-1)
```

З параметром -1 функція shift() застосовується до стовпця Close,

зсуваючи всі значення на одну позицію в негативному напрямку. Він імітує зсув у часі на один період, замінюючи кожне значення в стовпці Close значенням з одного періоду в майбутньому.

У результаті такого зміщення цін Close значення ціни в кожен момент часу тепер пов’язані з майбутнім, а не з теперішнім, що дозволяє тестувати моделі на даних, які були недоступні на момент навчання , що може бути корисним у задачах моделювання часових рядів і прогнозування.

3.2 Навчання моделі

Було створено три підмножини базових кадрів даних: навчання (70%), перевірка (15%) і тестування (15%). Щоб створити три окремі кадри (train_df, valid_df, test_df), було розраховано розділені індекси. На діаграмі нижче нанесено всі три кадри.

Рисунок 3.13 - Графік розподілу даних на тренувальні, валідаційні та тестові

Щоб навчити, перевірити та протестувати модель машинного навчання, було використано pandas DataFrame df, що містить фінансові дані, поділені на три частини.

У перших двох рядках коду змінні test_size і valid_size визначають частку даних, які використовуються для тестування та перевірки.

DataFrame df розділений на підмножини навчання, перевірки та тестування на основі змінних test_split_idx і valid_split_idx. Частка довжини DataFrame округляється до найближчого цілого числа за допомогою функції int().

За допомогою функції loc[] df DataFrame розбивається на підмножини навчання, перевірки та тестування. Метод .copy() використовується, щоб уникнути зміни оригінального DataFrame.

Потім створюється новий графік за допомогою модуля go бібліотеки plotly. Ділянка показує окремі сліди для кожної з трьох підмножин. Візуальна перевірка даних і перевірка того, що підмножини були правильно розділені, можлива за допомогою цього графіка цін на закриття з часом для кожної підмножини.

Модель машинного навчання можна навчити та оцінити за допомогою підмножин навчання, перевірки та тестування; набір для навчання навчає модель, набір для перевірки налаштовує гіперпараметри моделі, а набір для тестування перевіряє продуктивність моделі.

Під час навчання, перевірки та тестування певні стовпці видаляються з pandas DataFrame, що містить фінансові дані.

Точне налаштування XGBoostRegressor. Для налаштування гіперпараметрів моделі машинного навчання використовуються XGBoost і GridSearchCV.

Потім створюється словник гіперпараметрів, включаючи кількість оцінювачів, швидкість навчання, максимальну глибину, гамму та випадковий стан. Щоб шукати модель XGBoost, словник гіперпараметрів і набір оцінок, необхідно викликати функцію GridSearchCV. Доповнюючи модель різними комбінаціями гіперпараметрів, GridSearchCV виконує вичерпний пошук у сітці параметрів.

Найкращі гіперпараметри отримують за допомогою атрибута best_params_ об’єкта GridSearchCV, а найкращу оцінку перевірки – за

допомогою атрибута best_score_. За допомогою функції print() результати відображаються на консолі.

```
import xgboost as xgb
from sklearn.model_selection import train_test_split, GridSearchCV


parameters = {
    'n_estimators': [100, 200, 300, 400],
    'learning_rate': [0.001, 0.005, 0.01, 0.05],
    'max_depth': [8, 10, 12, 15],
    'gamma': [0.001, 0.005, 0.01, 0.02],
    'random_state': [42]
}

eval_set = [(X_train, y_train), (X_valid, y_valid)]
model = xgb.XGBRegressor(objective='reg:squarederror')
clf = GridSearchCV(model, parameters)
clf.fit(X_train, y_train)

print(f'Best params: {clf.best_params_}')
print(f'Best validation score = {clf.best_score_}')
```

```
Best params: {'gamma': 0.02, 'learning_rate': 0.05, 'max_depth': 15, 'n_estimators': 200, 'random_state': 42}
Best validation score = 0.7934252550011942
```

Рисунок 3.14 - Код виклику GridSearchCV

Цей код шукає найкращу комбінацію гіперпараметрів для моделі машинного навчання XGBoost за допомогою GridSearchCV для виконання систематичного пошуку в сітці вказаних параметрів. Потім моделі машинного навчання можна навчити та перевірити за допомогою найкращих гіперпараметрів.

Використовуючи найкращі гіперпараметри, отримані під час налаштування гіперпараметрів за допомогою GridSearchCV, навчається модель машинного навчання.

```
model = xgb.XGBRegressor(**clf.best_params_, objective='reg:squarederror')
model.fit(X_train, y_train, eval_set=eval_set, verbose=False)
```

Рисунок 3.15 - Код виклику XGBRegressor з кращими параметрами

Найкращі гіперпараметри з попереднього кроку використовуються для створення нової моделі XGBoost. Функція XGBRegressor викликається за допомогою синтаксису **clf.best_params_, який розпаковує найкращі

гіперпараметри, отримані з об’єкта GridSearchCV, і передає їх як аргументи функції XGBRegressor. Установивши цільову функцію на «reg:squarederror», моделі навчаються мінімізувати середню квадратичну помилку.

Далі викликаю команду fit() у моделі XGBoost із навчальними даними та набором оцінок. Під час навчання вихідні дані не друкуватимуться, якщо для параметра verbose встановлено значення False. Функція Fit() навчає модель XGBoost за допомогою гіперпараметрів і цільової функції, указаних у даних навчання. Під час навчання продуктивність моделі розраховується за допомогою набору оцінок, щоб уникнути переобладнання.

Використовуючи кроки налаштування гіперпараметрів з попереднього кроку налаштування гіперпараметрів, цей код навчає модель XGBoost за допомогою цих гіперпараметрів. Тепер можна робити прогнози, використовуючи нові дані, або оцінювати та тестувати модель на основі нових даних.

Для навченої моделі машинного навчання XGBoost створює графічне представлення важливості функції.

У бібліотеці XGBoost plot_importance() створює графік відносної важливості кожної функції. На осі абсцис гістограми нанесено назви об’єктів, а на осі у – оцінки важливості.

Побудова графіка відносної важливості кожної функції в моделі корисна для розуміння того, які характеристики є найбільш важливими для прогнозування цільової змінної, а також для виявлення нерелевантних або шумних.

Навчену модель XGBoost можна проаналізувати, візуалізувавши важливість її функцій за допомогою цієї функції.

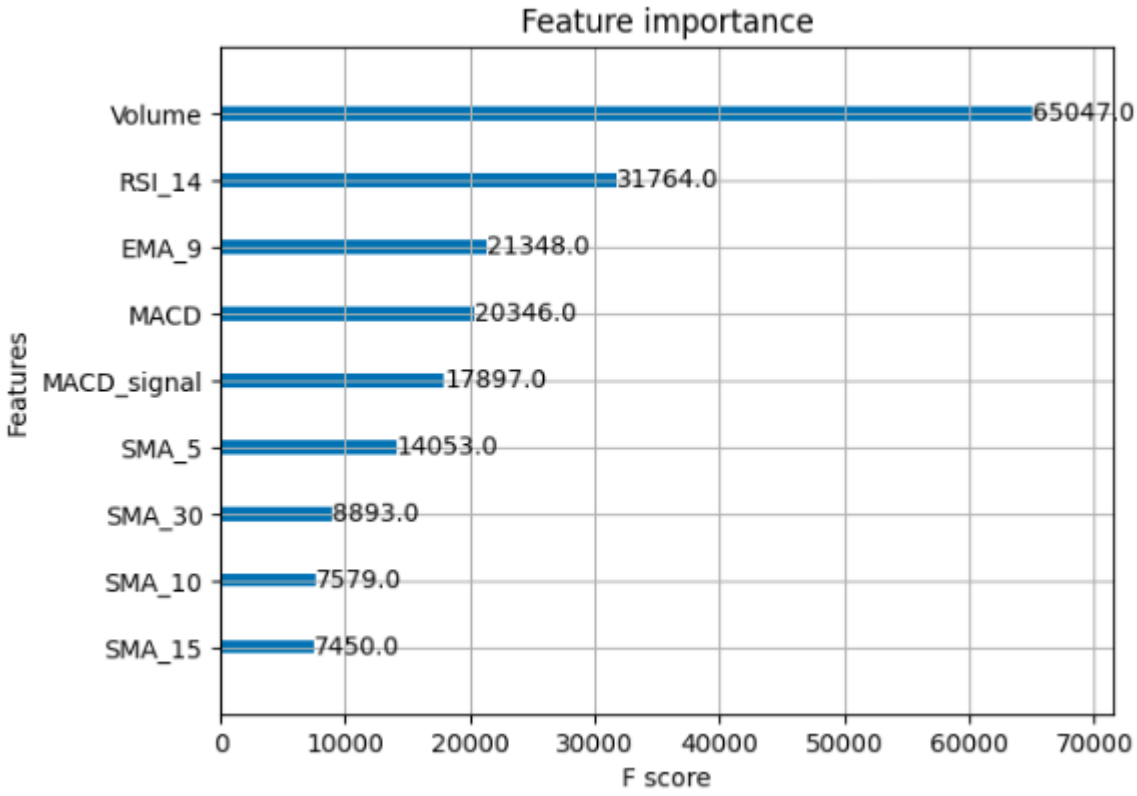

Рисунок 3.16 - Важливіть показників

Використовуючи навчену модель XGBoost, робимо прогнози на основі даних тестування та відображаємо фактичні та прогнозовані значення цільової змінної для перших п’яти екземплярів.

За допомогою змінних функції тестування X_test як вхідних даних функція predict() викликається на навченій об’єктній моделі моделі XGBoost. Прогнозовані значення для цільової змінної зберігаються в y_pred.

У змінній y_test зберігаються фактичні значення цільової змінної. Змінна y_pred також відображає прогнозовані значення цільової змінної для тих самих екземплярів у даних тестування.

Загалом цей код порівнює фактичні та прогнозовані значення цільової змінної, щоб оцінити продуктивність навченої моделі XGBoost. Функція predict() використовується для створення прогнозів для даних тестування за допомогою навченої моделі, а функція print() використовується для відображення фактичних і прогнозованих значень для перших п’яти

екземплярів у даних тестування, щоб модель могла бути візуалізовано.

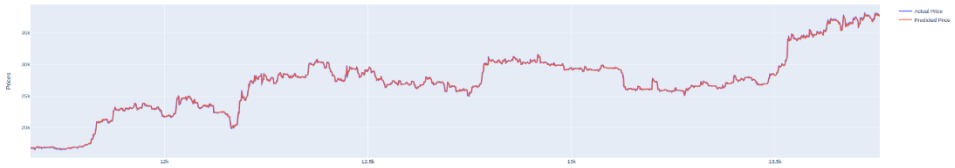

Рисунок 3.17 - Результат прогнозування XGBoost

Середньоквадратична помилка (MSE) обчислюється та відображається між фактичними та прогнозованими значеннями цільової змінної для даних тестування за допомогою наведеного нижче коду.

Для обчислення MSE між фактичними значеннями цільової змінної y_test і прогнозованими значеннями y_pred, згенерованими моделлю XGBoost, використовується функція mean_squared_error() із бібліотеки scikit-learn. Нижчий MSE вказує на кращу відповідність між прогнозами моделі та фактичними значеннями. Значення MSE відображаються як числа з плаваючою комою, що дозволяє легко оцінити продуктивність моделі на основі даних тестування.

Ця функція використовує середню квадратичну помилку як показник продуктивності для оцінки продуктивності навченої моделі XGBoost на основі даних тестування. У результаті MSE можна порівнювати з іншими моделями або використовувати як еталон для майбутніх удосконалень. Код надано в Додатку А.

ARIMA. Для використання цієї моделі наступний фрагмент коду було використано.

```
train_ar = train_df['Close'].values
test_ar = test_df['Close'].values
history = [x for x in train_ar]
print(type(history))
predictions = list()
for t in range(len(test_ar)):
    model = ARIMA(history, order=(5, 1, 0))
    model_fit = model.fit()
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test_ar[t]
    history.append(obs)
error = mean_squared_error(test_ar, predictions)
print('Testing Mean Squared Error: %.3f' % error)
```

Рисунок 3.18 - Код виклику ARIMA

У вихідному коді реалізовано прогнозування цін закриття (Close) за допомогою моделі ARIMA. Навчальний набір даних (train_df) використовується для тренування моделі, тоді як тестовий набір (test_df) використовується для оцінки її продуктивності.

Починаємо із створення масивів train_ar і test_ar, які містять значення цін закриття для відповідних наборів. Набір даних history ініціалізується значеннями з train_ar та використовується як історія для тренування ARIMA.

У циклі для кожного елементу тестового набору реалізовано наступні дії:

a. Створення моделі ARIMA з параметрами (p, d, q) = (5, 1, 0).
b. Проходження моделі ARIMA на навчальних даних.
c. Прогнозування на один крок вперед.
d. Додавання прогнозованого значення до списку predictions.
e. Отримання фактичного значення з тестового набору.
f. Додавання фактичного значення до набору даних history для подальшого використання.

Наприкінці розраховується середньоквадратична помилка (MSE) між фактичними та прогнозованими значеннями на тестовому наборі, і це значення виводиться на екран. Реалізація моделі ARIMA дозволяє проводити прогнози

на основі історії цін закриття для подальшої аналізу їх точності та ефективності.

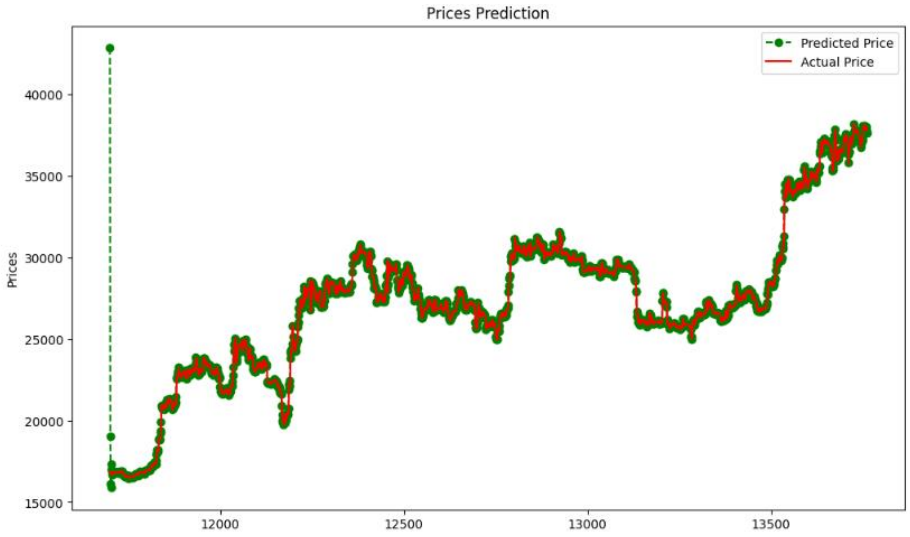

Рисунок 3.19 - Результат прогнозування ARIMA

LSTM. Спочатку дані навчального та тестового наборів масштабуються до інтервалу від 0 до 1. Потім визначається функція для створення пари вхідних та вихідних даних для LSTM моделі з використанням певної кількості попередніх часових кроків. Модель LSTM будується з двома шарами LSTM та одним шаром Dense, після чого компілюється для тренування. Тренування проводиться протягом 50 епох. Після тренування модель використовується для прогнозування та оцінки, а отримані прогнозні значення повертаються до вихідного масштабу. Завершально обчислюється середньоквадратична помилка між фактичними та прогнозованими значеннями, яка виводиться на екран для оцінки точності моделі. Код реалізації наведено в додатку А.

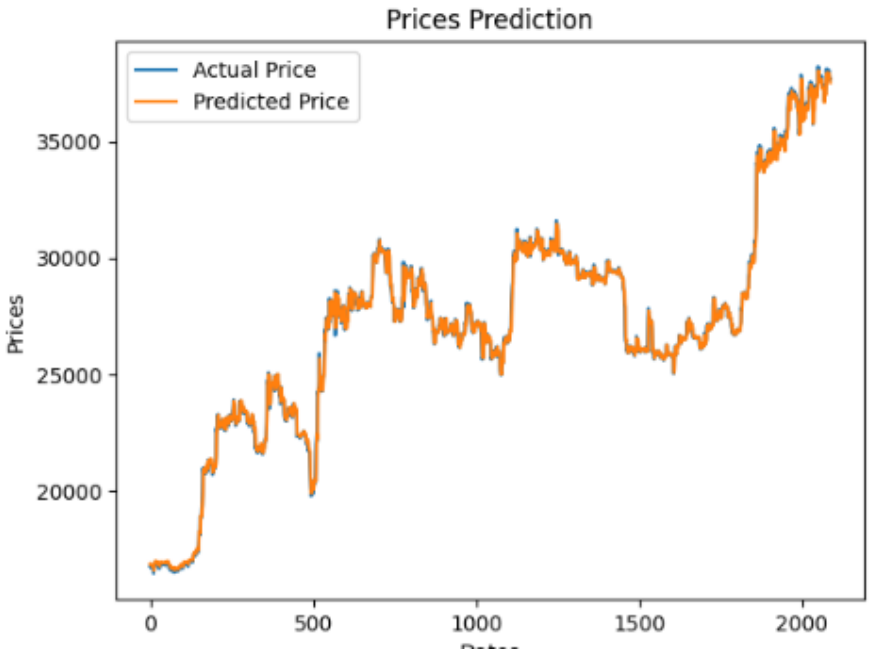

Рисунок 3.20 - Результат прогнозування LSTM

3.3 Порівняння результатів

Таблиця 3.1 - порівняння MSE

Модель	Середньоквадратична помилка (MSE)
XGBoost	61268.430
ARIMA	807171.538
LSTM	90245.726

Загалом нижчий MSE свідчить про кращий результат, оскільки це означає, що прогнози моделі ближчі до фактичних значень.

XGBoost має найнижчий MSE серед трьох моделей, що свідчить про те, що, на підставі наданих метрик, XGBoost найкраще працює у відношенні мінімізації квадратичних відмінностей між прогнозованими та фактичними значеннями.

ARIMA має найвищий MSE, що вказує на те, що, на підставі цих метрик, ARIMA не працює так ефективно, як XGBoost або LSTM на даному наборі даних.

LSTM має нижчий MSE порівняно з ARIMA, але вищий, ніж у XGBoost. Він веде себе краще, ніж ARIMA, але не так ефективно, як XGBoost.

На підставі отриманих значень MSE, XGBoost, є найефективнішою моделлю серед трьох для заданого набору даних та метрики оцінки.

3.4 Реалізація мобільного додатку

Мобільний додаток спрямований на управління фінансовим портфелем користувача та надає зручні інструменти для додавання, відстеження та аналізу фінансових активів. Одним з ключових факторів є прогнозування курсу активу. Після завантаження користувачу показуються його портфелі (див. Рисунок 3.18) або повідомлення що портфелів немає (див. Рисунок 3.17).

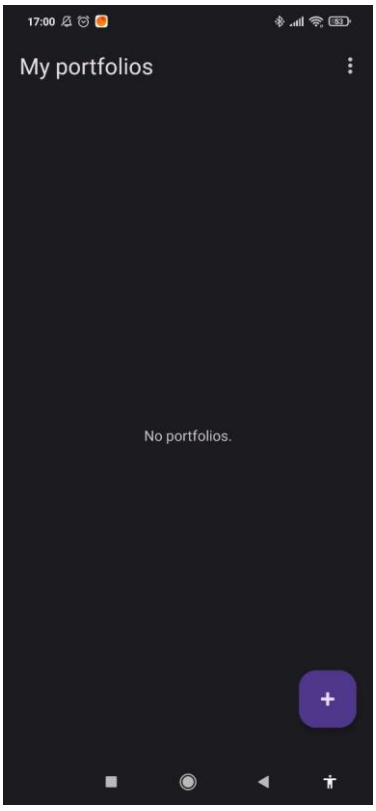

Рисунок 3.21 - Екран додатку без портфелів

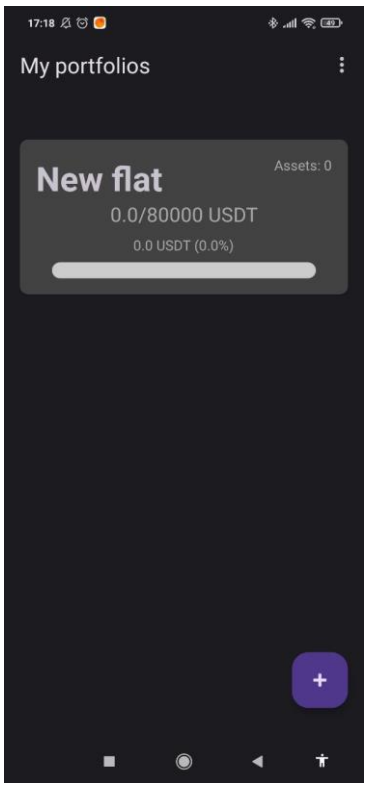

Рисунок 3.22 - Екран додатку з активними портфелями

Для перегляду деталей портфелю та його управління користувач натискає на потрібний портфель. Екран деталей показано на рисунку 3.19.

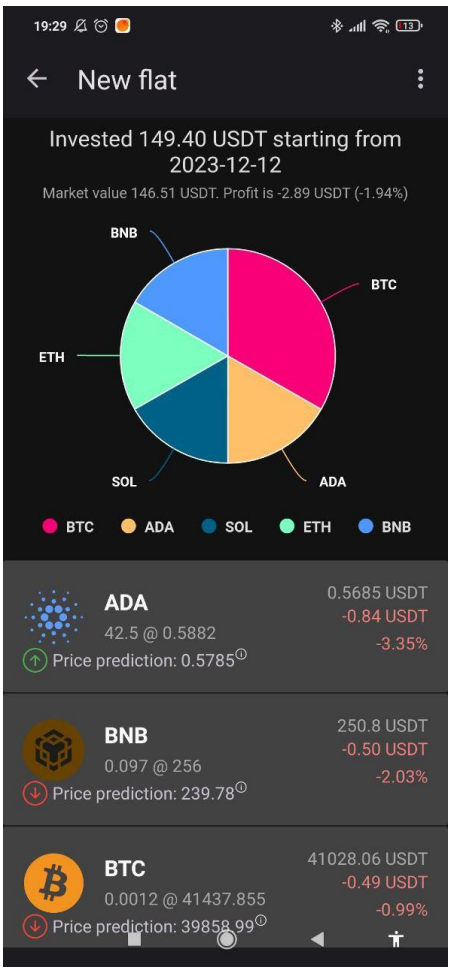

Рисунок 3.23 - Екран додатку з деталізацією портфелю

Також було реалізовано можливість додавати нові транзакції як прямою купівлею на біржі, так і внесенням даних вручну та автоматичним імпортом історії з біржі (див. Рисунок 3.20, 3.21, 3.22).

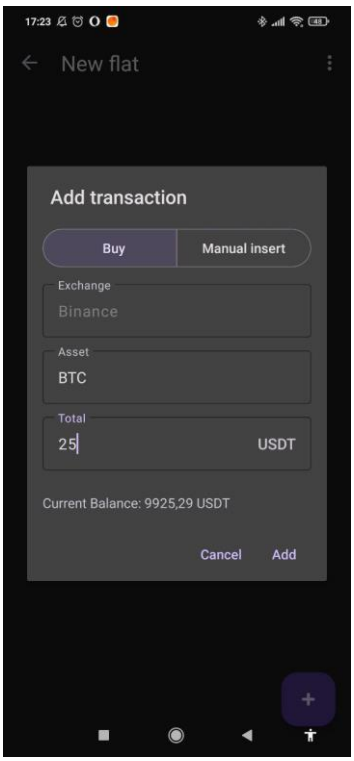

Рисунок 3.24 - Діалогове вікно купівлі активу напряму на біржі

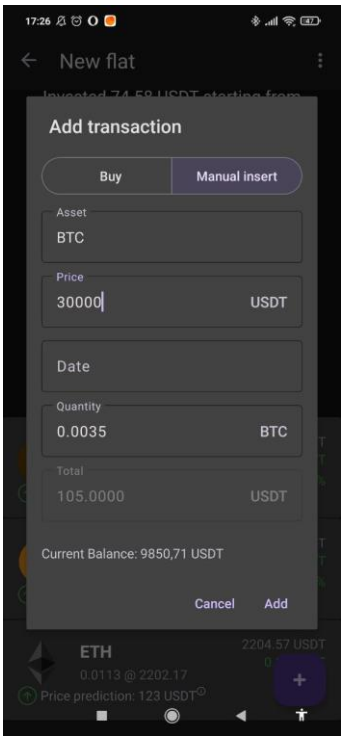

Рисунок 3.25 - Діалогове вікно ручного додавання транзакції

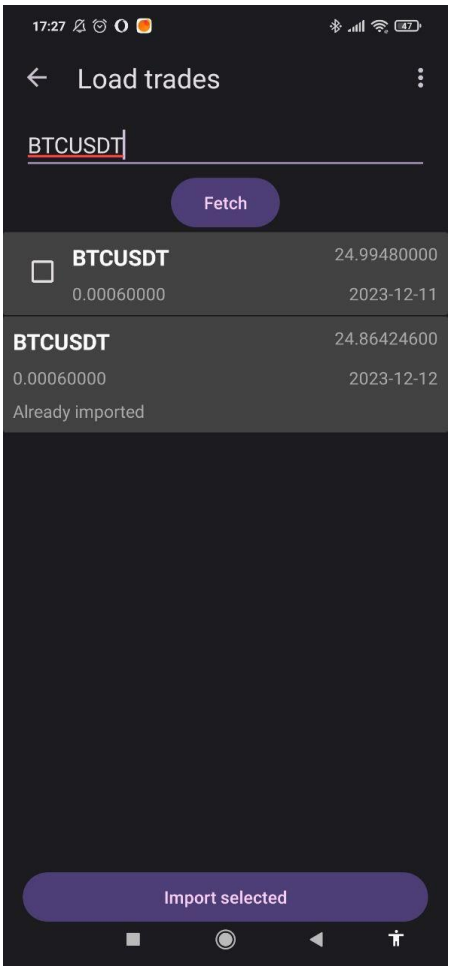

Рисунок 3.26 - Екран автоматичного завантаження транзакцій з біржі

Для активів в портфелі можна подивитись як деталі всіх транзакцій (див. Рисунок 3.23), так і прогнозовану ціну яка отримана за допомогою розробленої моделі. Також, користувач інформується про те, що ціна розрахована ШІ (див. Рисунок 3.24). Програмний код ключових моментів наведено в Додатку Б.

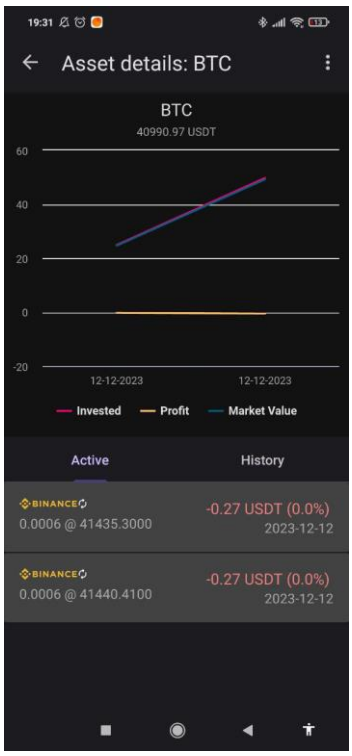

Рисунок 3.27 - Екран деталей по конкретному активу

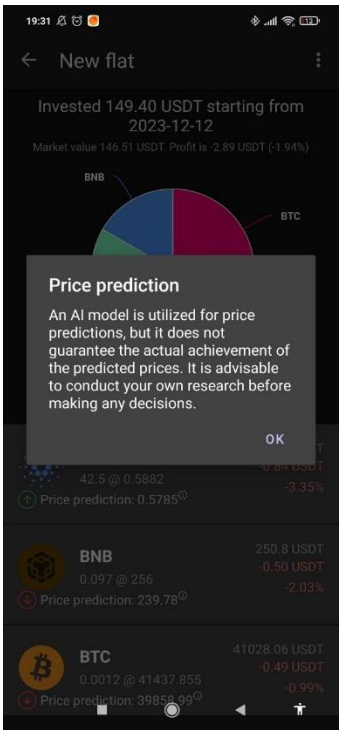

Рисунок 3.28 - Інформавання користувача про прогноз ціни ШІ

ВИСНОВКИ

У ході виконання даної роботи було проведено аналіз предметної області управління криптовалютними портфелями та визначена актуальність проблеми в умовах сучасного фінансового ринку. Визначено поняття криптовалюти та криптовалютного портфеля, які є важливими елементами сучасного інвестиційного середовища.

Проведений аналіз технологій для управління криптовалютними портфелями виявив різноманітні підходи та інструменти, що використовуються у сфері фінансів.

Було проведено детальний огляд моделей та методів штучного інтелекту, зокрема XGBoost та ARIMA, дозволив визначити їхню потенційну ефективність у прогнозуванні ринкових тенденцій. Також висвітлені ключові індикатори технічного аналізу, такі як RSI, ковзна середня та MACD.

У другому розділі проведено огляд та вибір технологій для розробки додатку на платформі Android. Визначено мови програмування та інструментарій для оптимальної розробкидодатку.

Реалізація інтелектуальної технології автоматизації управління та аналізу криптовалютних активів була проведена через створення вибірки навчальних та тестових даних, навчання моделі та оцінку її точності. Отримані результати вказують на потенційну ефективність застосованих методів у розв'язанні задачі прогнозування ринкових динамік.

Однією з основних перспектив для подальшого розвитку є порівняльний аналіз різних алгоритмів для прогнозування ринкових тенденцій криптовалют. Дослідження ефективності інших методів машинного навчання та статистичного аналізу може допомогти визначити оптимальний підхід для прогнозування змін вартості криптовалют.

Також, було розроблено мобільний додаток для Android, який використовує розроблену модель.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. [White paper]
2. Криптовалюта — що це таке, як вона працює і навіщо потрібна [Електронний ресурс]. URL: https://termin.in.ua/kryptovaliuta/
3. Цікаво знати: Ризики та переваги використання крипто валюти. [Електронний ресурс] - http://cikavoznaty.com.ua/2018/01/03/ryzyky-ta-perevagykryptovaljuty/
4. Роз'яснення щодо правомірності використання в Україні «віртуальної валюти/криптовалюти» Bitcoin // Офіційне Інтернет-представництво НБУ.– 10.11.2014
5. Криптовалютний портфель: що необхідно знати новачкам [Електронний ресурс]. URL: https://www.nta.ua/kryptovalyutnyj-portfel-shho-neobhidno-znaty-novachkam/
6. Портфель криптовалют: стратегії формування та вибір активів [Електронний ресурс]. URL: https://hub.obozrevatel.com/ukr/portfel-kriptovalyut-strategii-formuvannya-ta-vibir-aktiviv.htm
7. Kirkpatrick, C. D., & Dahlquist, J. R. (2010). Technical Analysis: The Complete Resource for Financial Market Technicians. FT Press.
8. Raschka, S. (2018). Python Machine Learning (3rd ed.). Packt Publishing.
9. Raschka, S., & Mirjalili, V. (2019). Python Machine Learning (2nd ed.). Packt Publishing.
10. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830. [Електронний ресурс] URL: https://scikit-learn.org/stable/
11. XGBoost Documentation on Regularization Parameters [Електронний ресурс] URL:

https://xgboost.readthedocs.io/en/latest/tutorials/regression.html#regularization

12. Box, G. E. P., Jenkins, G. M., & Reinsel, G. C. (2015). "Time Series Analysis: Forecasting and Control." Hoboken, NJ: John Wiley & Sons.
13. Hyndman, R. J., & Athanasopoulos, G. (2018). "Forecasting: Principles and Practice." OTexts.
14. Enders, W. (2014). "Applied Econometric Time Series." Wiley.
15. Cryer, J. D., & Chan, K. S. (2008). "Time Series Analysis: With Applications in R." Springer.
16. 5 основних індикаторів, що використовуються в технічному аналізі [Електронний ресурс]. URL: https://academy.binance.com/uk/articles/5-essential-indicators-used-in-technical-analysis
17. Android – платформа для всіх [Електронний ресурс]. URL: https://www.android.com/.
18. Android Studio vs. IntelliJ IDEA [Електронний ресурс]. URL: https://www.trustradius.com/compare-products/android-studio-vs-intellij-idea
19. Kotlin VS Java – What's the Difference? [Електронний ресурс]. URL: https://www.freecodecamp.org/news/kotlin-vs-java-whats-the-difference/
20. Puneet Singh, Prem Kumar Yadav, Shashank Chaurasia and Shubham Bhardwaj, "Stock Price Predictions With ML Using Python", International Journal of Scientific Research in Engineering and Management (IJSREM), 2020.
21. A. Sharma, D. Bhuriya and U. Singh, "Survey of Stock Market Prediction Using Machine Learning Approach", pp. 506-509, 2017.
22. Andrej Karpathy The Unreasonable Effectiveness of Recurrent Neural Networks [Електронний ресурс] - URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

ДОДАТОК А

Код декомпозиції.

```
import statsmodels.api as sm
import matplotlib.pyplot as plt

df_close = data_df[['Date', 'Close']].copy()
df_close = df_close.set_index('Date')

# Decompose the time series
decomposition = sm.tsa.seasonal_decompose(df_close, model='additive', period=365)

# Access the decomposed components
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

# Plot the components
plt.figure(figsize=(12, 8))
plt.subplot(411)
plt.plot(df_close, label='Original')
plt.legend()
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend()
plt.subplot(413)
plt.plot(seasonal, label='Seasonal')
plt.legend()
plt.subplot(414)
plt.plot(residual, label='Residual')
plt.legend()
plt.tight_layout()
plt.show()
```

Код розрахунку значень технічних індикаторів.

```
data_df['EMA_9'] = data_df['Close'].ewm(9).mean().shift()
data_df['SMA_5'] = data_df['Close'].rolling(5).mean().shift()
data_df['SMA_10'] = data_df['Close'].rolling(10).mean().shift()
data_df['SMA_15'] = data_df['Close'].rolling(15).mean().shift()
data_df['SMA_30'] = data_df['Close'].rolling(30).mean().shift()
data_df['RSI_14'] = data_df.ta.rsi(length = 14)
EMA_12 = pd.Series(data_df['Close'].ewm(span=12, min_periods=12).mean())
EMA_26 = pd.Series(data_df['Close'].ewm(span=26, min_periods=26).mean())
data_df['MACD'] = pd.Series(EMA_12 - EMA_26)
data_df['MACD_signal'] = pd.Series(data_df.MACD.ewm(span=9, min_periods=9).mean())
```

Код підготовки тренувальних/валідаційних та тестових даних.

```
data_df['Close'] = data_df['Close'].shift(-1)

df = data_df.iloc[33:] # Because of moving averages and MACD line
df = data_df[:-1] # Because of shifting close price

data_df.index = range(len(data_df))

test_size = 0.15
valid_size = 0.15

test_split_idx = int(df.shape[0] * (1-test_size))
valid_split_idx = int(df.shape[0] * (1-(valid_size+test_size)))

train_df = df.loc[:valid_split_idx].copy()
valid_df = df.loc[valid_split_idx+1:test_split_idx].copy()
test_df = df.loc[test_split_idx+1:].copy()

fig = go.Figure()
fig.add_trace(go.Scatter(x=train_df.Date, y=train_df.Close, name='Training'))
fig.add_trace(go.Scatter(x=valid_df.Date, y=valid_df.Close, name='Validation'))
fig.add_trace(go.Scatter(x=test_df.Date, y=test_df.Close, name='Test'))
fig.show()
```

Код навчання XGBBoost.

```
import xgboost as xgb
from sklearn.model_selection import train_test_split, GridSearchCV


parameters = {
'n_estimators': [100, 200, 300, 400],
'learning_rate': [0.001, 0.005, 0.01, 0.05],
'max_depth': [8, 10, 12, 15],
'gamma': [0.001, 0.005, 0.01, 0.02],
'random_state': [42]
}

eval_set = [(X_train, y_train), (X_valid, y_valid)]
model = xgb.XGBRegressor(objective='reg:squarederror')
clf = GridSearchCV(model, parameters)
clf.fit(X_train, y_train)

print(f'Best params: {clf.best_params_}')
print(f'Best validation score = {clf.best_score_}')

import joblib # Add this line for the joblib module
```

```
# Save the best model
model = xgb.XGBRegressor(**clf.best_params_, objective='reg:squarederror')
model.fit(X_train, y_train, eval_set=eval_set, verbose=False)
```

Код навчання ARIMA.

```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error

train_ar = train_df['Close'].values
test_ar = test_df['Close'].values
history = [x for x in train_ar]
print(type(history))
predictions = list()
for t in range(len(test_ar)):
model = ARIMA(history, order=(5, 1, 0))
model_fit = model.fit()
output = model_fit.forecast()
yhat = output[0]
predictions.append(yhat)
obs = test_ar[t]
history.append(obs)
error = mean_squared_error(test_ar, predictions)
print('Testing Mean Squared Error: %.3f' % error)
```

ДОДАТОК Б

Код головного Activity.

```
class MainActivity : AppCompatActivity() {
    private lateinit var navController: NavController

    override fun onCreate(savedInstanceState: Bundle?) {
        PreferenceManager.setDefaultValues(this, R.xml.root_preferences, false)
        WindowCompat.setDecorFitsSystemWindows(window, false)
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        setSupportActionBar(findViewById(R.id.toolbar))

        navController = findNavController(R.id.nav_host_fragment_content_main)
        val appBarConfiguration = AppBarConfiguration(navController.graph)
        setupActionBarWithNavController(navController, appBarConfiguration)
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        menuInflater.inflate(R.menu.menu_main, menu)
        return true
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        return when (item.itemId) {
            R.id.action_settings -> {
                startActivity(Intent(this, SettingsActivity::class.java))
                true
            }
            else -> super.onOptionsItemSelected(item)
        }
    }

    override fun onSupportNavigateUp(): Boolean {
        return navController.navigateUp() || super.onSupportNavigateUp()
    }
}
```

Код фрагменту для додавання портфелю.

```
class AddPortfolioDialogFragment(private val parent: Fragment) : DialogFragment() {
    private val sharedViewModel: SharedViewModel by activityViewModels()
    interface AddPortfolioDialogListener {
        fun onPortfolioAdded(tradingPair: String, amount: Double)
    }

    private lateinit var listener: AddPortfolioDialogListener
```

```
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        return activity?.let {
            val builder = AlertDialog.Builder(it)
            val inflater = requireActivity().layoutInflater

            val binding = DialogAddPortfolioBinding.inflate(inflater)
            binding.portfolioTarget.suffixText = sharedViewModel.basicAsset.value

            builder.setView(binding.root)
                .setTitle(R.string.add_portfolio_dialog_title)
                .setPositiveButton(R.string.add) { _, _ ->
                    val nameStr = binding.portfolioName.editText?.text.toString()
                    if (nameStr.isNotEmpty()) {
                        val targetStr = binding.portfolioTarget.editText?.text.toString().toDouble()
                        listener.onPortfolioAdded(nameStr.replaceFirstChar{it.uppercase()}, targetStr)
                    }
                }
                .setNegativeButton(R.string.cancel) { dialog, _ ->
                    dialog.cancel()
                }

            builder.create()
        } ?: throw IllegalStateException("Activity cannot be null")
    }

    override fun onAttach(context: Context) {
        super.onAttach(context)
        try {
            listener = parent as AddPortfolioDialogListener
        } catch (e: ClassCastException) {
            throw ClassCastException("Parent fragment must implement AddPortfolioDialogListener")
        }
    }

    companion object {
        fun newInstance(parentFragment: Fragment): AddPortfolioDialogFragment {
            return AddPortfolioDialogFragment(parentFragment)
        }
    }
}
```

Код фрагменту з додавання транзакції:

```
class AddTransactionDialogFragment(private val parent: Fragment) : DialogFragment() {
    interface AddPositionDialogListener {
        fun onExchangePositionAdded(tradingPair: String, asset: String, amount: BigDecimal)
        fun onManualPositionAdded(asset: String, date: Date, price: BigDecimal, quantity: BigDecimal, commission:
BigDecimal)
    }

    private lateinit var listener: AddPositionDialogListener
    private var tradingPairs = mutableListOf<BinanceSymbol>()
    private var availableBalance: BigDecimal? = null
```

```
override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
    return activity?.let {
        val builder = AlertDialog.Builder(it)
        val inflater = requireActivity().layoutInflater

        val binding = DialogAddTransactionBinding.inflate(inflater)

        val dataRepository = DataRepositoryImpl(requireContext())

        val sharedPreferences = PreferenceManager.getDefaultSharedPreferences(requireContext())
        val basicAsset = sharedPreferences.getString("binance_basic_asset", "") ?: "USDT"

        val addButton = dialog?.findViewById<Button>(Dialog.BUTTON_POSITIVE)

        binding.exchange.editText?.setText("Binance")
        binding.exchange.editText?.isEnabled = false
        // Coroutine Scope
        val scope = CoroutineScope(Dispatchers.Main)

        // Get Trading Pairs
        scope.launch {
            try {
                tradingPairs.addAll(withContext(Dispatchers.IO) {
                    dataRepository.getTradingPairs()
                })
            } catch (e: Exception) {
                Log.e("AddPositionDialogFragment", "Error in coroutine", e)
            }
            Log.d("MY DEBUG", tradingPairs.toString())
            val adapter = ArrayAdapter(
                requireContext(),
                android.R.layout.simple_spinner_dropdown_item,
                tradingPairs.map { it.baseAsset }
            )
            (binding.assetDropdown.editText as? AutoCompleteTextView)?.setAdapter(adapter)
        }

        // Disable the add button initially
        addButton?.isClickable = false

        binding.total.suffixText = basicAsset
        binding.price.suffixText = basicAsset

        scope.launch {
            try {
                val accountInfo = dataRepository.getAccountInfo()
                availableBalance = accountInfo.balances.find { it.asset == basicAsset }?.free ?: BigDecimal.valueOf(0.0)
                binding.currentBalance.text = String.format("Current Balance: %.2f %s", availableBalance, basicAsset)
            } catch (e: Exception) {
                Toast.makeText(requireContext(), e.toString(), Toast.LENGTH_SHORT).show()
            }
        }
```

```
        val amountDefault = sharedPreferences.getString("binance_weekly_amount", "")
        binding.total.editText?.text = Editable.Factory.getInstance().newEditable(amountDefault)


        val textWatcher = object : TextWatcher {
            override fun afterTextChanged(s: Editable?) {
                addButton?.isClickable = binding.assetDropdown.editText?.text!!.isNotEmpty() &&
binding.total.editText?.text!!.isNotEmpty() && tradingPairs.map
{ it.symbol }.contains(binding.assetDropdown.editText?.text!!.toString())
                if (binding.assetDropdown.editText?.text!!.isNotEmpty() && !tradingPairs.map
{ it.baseAsset }.contains(binding.assetDropdown.editText?.text!!.toString())) {
                    binding.assetDropdown.error = "Invalid trading pair"
                } else {
                    binding.quantity.suffixText = binding.assetDropdown.editText?.text!!
                    binding.assetDropdown.error = null
                    binding.assetDropdown.isErrorEnabled = false
                }

                if ( binding.total.editText?.text!!.isNotEmpty() ) {
                    val amountValue = binding.total.editText?.text!!.toString().toBigDecimal()
                    if ( amountValue <  BigDecimal.valueOf(11.0) ) {
                        binding.total.error = "Minimum amount is 11"
                    } else if ( availableBalance != null && amountValue > availableBalance ) {
                        binding.total.error = "Not enough available funds"
                    } else {
                        binding.total.error = null
                        binding.total.isErrorEnabled = false
                    }
                }
            }

            override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {
                // No implementation needed
            }

            override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
                // No implementation needed
            }
        }

        binding.dateLayout.editText?.setOnFocusChangeListener { _, hasFocus ->
            if (hasFocus) {
                showDatePicker(binding.dateLayout)
            }
        }

        binding.dateLayout.editText?.showSoftInputOnFocus = false
        // Set an OnClickListener on the TextInputLayout
        binding.dateLayout.editText?.setOnClickListener {
            showDatePicker(binding.dateLayout)
        }

        binding.assetDropdown.editText?.addTextChangedListener(textWatcher)
        binding.total.editText?.addTextChangedListener(textWatcher)
```

```
// Set up a listener for the toggle group
binding.toggleGroup.addOnButtonCheckedListener { group, checkedId, isChecked ->
    // Toggle visibility of additional fields based on the selected button
    MyDebug.print("group = $group, checkedId = $checkedId, isChecked = $isChecked")
    if(isChecked) {
        when (checkedId) {
            R.id.buyOnExchangeButton -> {

                // Hide additional fields for "Buy on exchange"
                binding.price.visibility = View.GONE
                binding.dateLayout.visibility = View.GONE
                binding.quantity.visibility = View.GONE
                binding.exchange.visibility = View.VISIBLE
                binding.total.editText?.setText(amountDefault.toString())
                binding.total.isEnabled = true
            }
            R.id.manualInsertButton -> {
                // Show additional fields for "Manual insert"
                binding.price.visibility = View.VISIBLE
                binding.dateLayout.visibility = View.VISIBLE
                binding.quantity.visibility = View.VISIBLE
                binding.exchange.visibility = View.GONE
                binding.total.editText?.setText("0")
                binding.total.isEnabled = false
                updateTotal(binding.price, binding.quantity, binding.total)
            }
        }
    }
}

// Set up TextWatchers
binding.price.editText?.addTextChangedListener(object : TextWatcher {
    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {
        // Not needed in this case
    }

    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
        updateTotal(binding.price, binding.quantity, binding.total)
    }

    override fun afterTextChanged(s: Editable?) {
        // Not needed in this case
    }
})

binding.quantity.editText?.addTextChangedListener(object : TextWatcher {
    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {
        // Not needed in this case
    }

    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
        updateTotal(binding.price, binding.quantity, binding.total)
    }
```

```
            override fun afterTextChanged(s: Editable?) {
                // Not needed in this case
            }
        })

        builder.setView(binding.root)
            .setTitle(R.string.add_transaction_dialog_title)
            .setPositiveButton(R.string.add) { _, _ ->
                val checkedButtonId = binding.toggleGroup.checkedButtonId
                val tradingAsset = binding.assetDropdown.editText?.text!!.toString()
                val totalStr = binding.total.editText?.text!!.toString()

                if (tradingAsset.isEmpty()) {
                    binding.assetDropdown.error = "Trading pair is required"
                    return@setPositiveButton
                }

                if (totalStr.isEmpty()) {
                    binding.total.error = "Amount is required"
                    return@setPositiveButton
                }

                if (!tradingPairs.map { it.baseAsset }.contains(tradingAsset)) {
                    binding.assetDropdown.error = "Invalid trading pair"
                    return@setPositiveButton
                }

                if ( checkedButtonId == R.id.manualInsertButton ) {
                    if (binding.price.editText?.text!!.isEmpty()) {
                        binding.price.error = "Amount is required"
                        return@setPositiveButton
                    }
                }

                val total = totalStr.toBigDecimal()
                val binanceSymbol = tradingPairs.find { it.baseAsset == tradingAsset }

                // Check which button is checked

                when (checkedButtonId) {
                    R.id.buyOnExchangeButton -> {
                        listener.onExchangePositionAdded(binanceSymbol?.symbol!!, binanceSymbol?.baseAsset ?: "N/A",
total)

                        dialog?.dismiss()
                    }
                    R.id.manualInsertButton -> {
                        val price = binding.price.editText?.text!!.toString().toBigDecimal()
                        val quantity = binding.quantity.editText?.text!!.toString().toBigDecimal()
                        val selectedDateText: String? = binding.dateLayout.editText?.text!!.toString()

                        val selectedDate: Date? = try {
                            SimpleDateFormat("yyyy-MM-dd", Locale.getDefault()).parse(selectedDateText.orEmpty())
                        } catch (e: ParseException) {
```

```
                    null // Handle the parsing error, e.g., invalid date format
                }

                listener.onManualPositionAdded(tradingAsset, selectedDate!!, price, quantity, BigDecimal.ZERO)
                // Process logic for "Manual insert"
                dialog?.dismiss()
            }
        }
    }
    .setNegativeButton(R.string.cancel) { dialog, _ ->
        dialog.cancel()
    }
    // Disable the add button initially
    addButton?.isClickable = false

    builder.create()
} ?: throw IllegalStateException("Activity cannot be null")
}

override fun onStart() {
    super.onStart()
    // Disable the add button initially
    dialog?.findViewById<Button>(Dialog.BUTTON_POSITIVE)?.isClickable = false
    dialog?.findViewById<Button>(Dialog.BUTTON_POSITIVE)?.isEnabled = false
    dialog?.findViewById<Button>(Dialog.BUTTON_POSITIVE)?.isActivated = false
    dialog?.findViewById<Button>(Dialog.BUTTON_POSITIVE)?.isFocusable = false
}

override fun onAttach(context: Context) {
    super.onAttach(context)
    try {
        listener = parent as AddPositionDialogListener
    } catch (e: ClassCastException) {
        throw ClassCastException("Parent fragment must implement AddPositionDialogListener")
    }
}

companion object {
    fun newInstance(parentFragment: Fragment): AddTransactionDialogFragment {
        return AddTransactionDialogFragment(parentFragment)
    }
}

private fun showDatePicker(dateTextInputEditText: TextInputLayout) {
    val calendar = Calendar.getInstance()
    val year = calendar.get(Calendar.YEAR)
    val month = calendar.get(Calendar.MONTH)
    val day = calendar.get(Calendar.DAY_OF_MONTH)

    val datePickerDialog = DatePickerDialog(
        requireContext(),
        { _, selectedYear, selectedMonth, selectedDay ->
            // Handle the selected date
            val selectedDate = "$selectedYear-${selectedMonth + 1}-$selectedDay"
```

```
            dateTextInputEditText.editText?.setText(selectedDate)
        },
        year,
        month,
        day
    )
    datePickerDialog.datePicker.maxDate = System.currentTimeMillis()

    datePickerDialog.show()
}

private fun updateTotal (price: TextInputLayout, amount: TextInputLayout, total: TextInputLayout) {
    try {
        val value = BigDecimal(price.editText?.text.toString()) * BigDecimal(amount.editText?.text.toString())
        total.editText?.setText(value.toString())
    } catch (e: Exception) {

    }
}
```

Код ViewModel:

```
class SharedViewModel : ViewModel() {
    private lateinit var db: DataRepositoryImpl
    private val _combinedLiveData: MediatorLiveData<CombinedResult> = MediatorLiveData()

    val combinedLiveData: LiveData<CombinedResult>
        get() = _combinedLiveData

    private val _basicAsset = MutableLiveData<String>()
    val basicAsset: LiveData<String>
        get() = _basicAsset

    private val _portfolios = MutableLiveData<List<Portfolio>?>()
    val portfolios: LiveData<List<Portfolio>?>
        get() = _portfolios


    private val _portfoliosDataList = MutableLiveData<List<PortfolioDetails>?>()
    val portfoliosDataList: LiveData<List<PortfolioDetails>?>
        get() = _portfoliosDataList

    private val _currentPricesList = MutableLiveData<List<BinanceLatestPrice>>()
    val currentPricesList: LiveData<List<BinanceLatestPrice>>
        get() = _currentPricesList

    private val _currentPortfolioDetails = MutableLiveData<PortfolioDetails?>()
    val currentPortfolioDetails: LiveData<PortfolioDetails?>
        get() = _currentPortfolioDetails

    private val _tradingPairs = MutableLiveData<List<BinanceSymbol>>()
    val tradingPairs: LiveData<List<BinanceSymbol>> get() = _tradingPairs

    private val _assets = MutableLiveData<List<String>>()
    val assets: LiveData<List<String>> get() = _assets
```

```
private val _transactions = MutableLiveData<List<Transaction>>()
val transactions: LiveData<List<Transaction>> get() = _transactions

private var _activePortfolioId: Int? = null
var activePortfolioId: Int?
    get() = _activePortfolioId
    set(value) {
        _activePortfolioId = value
    }

private var _activeAsset: String? = null
var activeAsset: String?
    get() = _activeAsset
    set(value) {
        _activeAsset = value
    }

private var _accountInfo: BinanceAccount? = null
val accountInfo: BinanceAccount?
    get() = _accountInfo



fun init(context: Context) {
    val sharedPreferences = PreferenceManager.getDefaultSharedPreferences(context)
    _basicAsset.value = sharedPreferences.getString("binance_basic_asset", "") ?: "USDT"
    db = DataRepositoryImpl(context)

    _combinedLiveData.addSource(currentPricesList) { updateCombinedResult() }
    _combinedLiveData.addSource(portfolios) { updateCombinedResult() }
}

fun resetCurrentPortfolio() {
    activePortfolioId = null
    _currentPortfolioDetails.value = null
}

private fun updateCombinedResult() {
    val result = CombinedResult(portfolios.value, currentPricesList.value)
    _combinedLiveData.value = result
}

private suspend fun loadAssets() {
    _assets.value = db.getAllAssets()
}

private suspend fun loadPortfolios() {
    _portfolios.value = db.getPortfolios()
}

suspend fun loadTransactions(asset: String) {
    if (currentPortfolioDetails.value != null) {
        val assetTransactions = db.getTransactions(currentPortfolioDetails.value!!.portfolio.id, asset)
```

```
            MyDebug.print(assetTransactions.toString())
            _transactions.postValue(assetTransactions)
            _activeAsset = asset
        }
    }

    private suspend fun loadPortfoliosData() {
        viewModelScope.launch {
            async { _accountInfo = db.getAccountInfo() }.await()
            async { loadAssets() }.await()
            async { loadAssetsCurrentPrice() }.await()
            val newList = db.getPortfolios().map { portfolio ->
                val positionSummaries = db.getAllPositionsInfo(portfolio.id).map { position ->
                    val currentPrice = getPriceForAsset(position.asset)
                    val invested = position.quantity * position.avgPrice
                    val marketValue = position.quantity * currentPrice!!
                    val profit = marketValue - invested
                    val profitPercent = if (invested.toDouble() == 0.0 ) BigDecimal.ZERO else (profit / invested) *
BigDecimal.valueOf(100)
                    val availableBalance = getAvailableAssetBalance(position.asset)
                    PositionSummary(
                        count = 0,
                        quantitySum = position.quantity,
                        avgPrice = position.avgPrice,
                        asset = position.asset,
                        currentPrice = currentPrice,
                        profit = profit,
                        profitPercent = profitPercent,
                        invested = invested,
                        availableBalance = availableBalance
                    )
                }
                PortfolioDetails(
                    portfolio = portfolio,
                    positions = positionSummaries
                )
            }

            _portfoliosDataList.postValue(newList)

            if (_activePortfolioId != null) {
                val currentData = newList.find { it.portfolio.id == activePortfolioId }
                _currentPortfolioDetails.postValue(currentData!!)
            }
        }
    }

    fun getAvailableAssetBalance(asset: String): BigDecimal {
        return accountInfo?.balances?.find { it.asset == asset }?.free ?: BigDecimal.valueOf(0.0)
    }

    suspend fun getPriceForAsset(asset: String): BigDecimal? {
        val symbol = "$asset${basicAsset.value}"
        if (currentPricesList.value != null) {
            val exchangePrice = currentPricesList.value?.find { it.symbol == symbol }
```

```
        if (exchangePrice != null) {
            return exchangePrice.price
        }
    }
    return viewModelScope.async {
        val exchangePrice = db.getLatestPrice(symbol)
        if (exchangePrice != null) {
            exchangePrice.price
        } else {
            BigDecimal.ZERO
        }
    }.await()
}

suspend fun getHistoryTransactions(): List<HistoryTransaction> {
    return db.getHistoryTransactionsList(activePortfolioId!!, activeAsset!!)
}

suspend fun addPortfolio(name: String, target: Double) {
    db.insertPortfolio(name, target)
    val currentList = _portfoliosDataList.value
    _portfoliosDataList.value = currentList
    loadPortfoliosData()
}

private suspend fun loadAssetsCurrentPrice() {
    val prices = viewModelScope.async {
        _assets.value?.map { asset ->
            val symbol = "$asset${basicAsset.value}"
            async {
                try {
                    db.getLatestPrice(symbol)
                } catch (e: Exception) {
                    MyDebug.print("Error in coroutine: $e")
                    null
                }
            }
        }?.awaitAll()
    }

    val result = prices.await()
    if ( result != null ) {
        _currentPricesList.value = result as List<BinanceLatestPrice>?
    }
}

suspend fun refreshData() {
    loadPortfoliosData()
}

fun getPortfolioData(portfolioId: Int): PortfolioDetails? {
    return _portfoliosDataList.value?.find { it.portfolio.id == portfolioId }
}
```

```
    fun sell(context: Context, asset: String, amount: BigDecimal) {
        if ( currentPortfolioDetails != null ) {
            viewModelScope.launch {
                val buyTransactionsList = db.getTransactions(currentPortfolioDetails.value!!.portfolio.id, asset,
TransactionStatus.ACTIVE.value)
                try {
                    val order = db.sell(asset, amount)
                    val priceAvg = order.fills.fold(BigDecimal.ZERO) { acc, fill ->
                        acc.add(fill.price * fill.qty)
                    } / order.fills.fold(BigDecimal.ZERO) { acc, fill ->
                        acc.add(fill.qty)
                    }
                    val newTransaction = Transaction(
                        timestamp = order.transactTime,
                        portfolioId = currentPortfolioDetails.value?.portfolio!!.id,
                        asset = asset,
                        orderId = order.orderId,
                        quantity = order.executedQty,
                        price = priceAvg,
                        commission = order.fills.fold(BigDecimal.ZERO) { acc, fill ->
                            acc.add(fill.commission)
                        },
                        type = TransactionType.AUTO,
                        direction = TransactionDirections.SELL.value,
                        status = TransactionStatus.HISTORY.value
                    )
                    val sellTransactionId = db.insertTransaction(newTransaction).toInt()
                    db.closeBuyTransactions(buyTransactionsList)
                    db.insertSellHistory(sellTransactionId, buyTransactionsList)
                    refreshData()
                    println(order)
                    Toast.makeText(context, "$asset has been sold.", Toast.LENGTH_SHORT).show()
                } catch (e: BinanceException) {

                    Log.e("sell", "Error in coroutine", e)
                    val jsonParser = JsonParser()
                    val jsonObject = jsonParser.parse(e.toString()).asJsonObject
                    val errorMsg = jsonObject.get("msg")?.asString ?: "Unknown error"

                    Toast.makeText(context, errorMsg, Toast.LENGTH_SHORT).show()
                }
            }
        }
    }

    fun addExchangeTransaction(context: Context, tradingPair: String, asset: String, amount: BigDecimal) {
        viewModelScope.launch {
            try {
                val order = db.buy(tradingPair, amount)
                val priceAvg = order.fills.fold(BigDecimal.ZERO) { acc, fill ->
                    acc.add(fill.price * fill.qty)
                } / order.fills.fold(BigDecimal.ZERO) { acc, fill ->
                    acc.add(fill.qty)
                }
                val newTransaction = Transaction(
```

```
                timestamp = order.transactTime,
                portfolioId = currentPortfolioDetails.value?.portfolio!!.id,
                asset = asset,
                orderId = order.orderId,
                quantity = order.executedQty,
                price = priceAvg,
                commission = order.fills.fold(BigDecimal.ZERO) { acc, fill ->
                    acc.add(fill.commission)
                },
                type = TransactionType.AUTO
            )
            db.insertTransaction(newTransaction)
            refreshData()
            Toast.makeText(context, "$asset has been bought.", Toast.LENGTH_SHORT).show()
        } catch (e: Exception) {
            Log.e("onPositionAdded", "Error in coroutine", e)
            Toast.makeText(context, "Failed to bought coin. Response:\n${e}", Toast.LENGTH_SHORT).show()
        }
    }
}

fun addManualTransaction(
    context: Context,
    asset: String,
    date: Date,
    price: BigDecimal,
    quantity: BigDecimal,
    commission: BigDecimal
) {
    viewModelScope.launch {
        try {
            val newTransaction = Transaction(
                timestamp = date.time,
                portfolioId = currentPortfolioDetails.value?.portfolio!!.id,
                asset = asset,
                orderId = 0,
                quantity = quantity,
                price = price,
                commission = commission,
                type = TransactionType.MANUAL
            )
            Log.d("MY DEBUG", newTransaction.toString())
            db.insertTransaction(newTransaction)
            refreshData()
        } catch (e: Exception) {
            Log.e("onPositionAdded", "Error in coroutine", e)
        }
    }
    Toast.makeText(context, "$asset has been added.", Toast.LENGTH_SHORT).show()
}

fun importTrades(trades: Set<BinanceHistoryOrder>): Int {
    var errors = 0
    trades.forEach { trade ->
        viewModelScope.launch {
```

```
            val currentPrice = db.getLatestPrice(trade.symbol).price
            val position = Transaction(
                timestamp = trade.time,
                portfolioId = currentPortfolioDetails.value!!.portfolio.id,
                asset = trade.symbol.replace(basicAsset.value!!, ""),
                orderId = trade.orderId,
                quantity = trade.executedQty.toBigDecimal(),
                price = if (trade.price != "0.00000000") trade.price.toBigDecimal() else currentPrice,
                commission = BigDecimal.ZERO,//trade.commission.toBigDecimal(),
                type = TransactionType.IMPORTED
            )
            try {
                db.insertTransaction(position)
            } catch (e: Exception) {
                Log.e("ERROR", "Failed to insert position: ${position}\nError: $e")
                errors++
            }
            refreshData()
        }
    }

    return errors
}

fun fetchTradingPairs() {
    viewModelScope.launch {
        try {
            val pairs = withContext(Dispatchers.IO) {
                db.getTradingPairs()
            }
            _tradingPairs.value = pairs
        } catch (e: Exception) {
            Log.e("MY DEBUG", "Error fetching trading pairs", e)
        }
    }
}

fun getAllOrders(symbol: String): LiveData<List<BinanceHistoryOrder>> {
    val result = MutableLiveData<List<BinanceHistoryOrder>>()

    viewModelScope.launch {
        try {
            withContext(Dispatchers.IO) {
                val orders = db.getAllOrders(symbol)
                result.postValue(orders)
            }
        } catch (e: Exception) {
            // Handle exceptions as needed
            Log.e("ViewModel", "Error getting order IDs", e)
        }
    }

    return result
}
```

```
    fun getAllExistingOrdersIs(symbol: String): LiveData<List<Long>> {
        val result = MutableLiveData<List<Long>>()
        viewModelScope.launch {
            withContext(Dispatchers.IO) {
                val orders = db.getAllExistingOrderIds(symbol.replace(basicAsset.value!!, ""))
                result.postValue(orders)
            }
        }
        return result
    }
}
```