МІНІСТЕРСТВО ОСВІТИ І НАУКИ КРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ОЛІЙНИК БОГДАН СЕРГІЙОВИЧ

Допускається до захисту:

в.о. завідувача кафедри
інформаційних технологій,
канд. техн. наук, доцент
_____ О. В. Зелінська
«_____» _____ 2024 р.

ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ АНАЛІЗУ ДАНИХ ФІНАНСОВИХ РИНКІВ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (магістерська) робота

Науковий керівник:

Н. А. Потапова, доцент кафедри
інформаційних технологій,
канд. екон. наук, доцент
_____
(Підпис)

Оцінка: _____/_____/_____
(бали/ за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(Підпис)

Вінниця – 2024

АНОТАЦІЯ

Олійник Б.С. Інформаційна система для аналізу даних фінансових ринків. Спеціальність 122 «Комп'ютерні науки». Освітня програма: Комп'ютерна обробка даних (Data Science). Донецький національний університет імені Василя Стуса, Вінниця 2024.

У магістерській роботі представлено розробку інформаційної системи для аналізу та прогнозування даних фінансових ринків. Система включає модулі збору, обробки та аналізу часових рядів із застосуванням сучасних математичних моделей, таких як ARIMA, SARIMA, Prophet та Auto ARIMA. Розроблена система дозволяє оцінювати сезонні, трендові та стохастичні компоненти даних, а також виконувати прогнозування для фінансового планування та інвестування.

Робота описує процеси розробки системи: від збору даних до їх візуалізації та аналізу. Використано Python та такі бібліотеки, як Pandas, NumPy, Statsmodels, Scikit-learn, Prophet. У роботі наведено математичні моделі для прогнозування часових рядів, виконано їх тестування, оцінено точність за метриками (MAE, MAPE, RMSE).

Магістерська робота складається зі вступу, трьох розділів, висновків та додатків. У вступі обґрунтовано актуальність теми, визначено мету, завдання дослідження та методологію. У першому розділі розглянуто теоретичні аспекти аналізу часових рядів. Другий розділ присвячено технологіям обробки даних і розробці системи. У третьому розділі наведено результати тестування моделей, їх порівняння та рекомендацій щодо використання в фінансовому прогнозуванні.

Магістерська робота складається зі вступу, 3 розділів, висновків, списку літератури із 40 джерел, включає 74 рисунки і додатки. Загальний обсяг роботи становить 90 сторінок.

ABSTRACT

Oliinyk B.S. Information system for analyzing financial market data. Specialty 122 "Computer Science". Educational Program: Data Science. Vasyl Stus Donetsk National University, Vinnytsia 2024.

This master's thesis presents the development of an information system for analyzing and forecasting financial market data. The system comprises modules for data collection, processing, and analysis of time series using modern mathematical models such as ARIMA, SARIMA, Prophet, and Auto ARIMA. The developed system allows for evaluating seasonal, trend, and stochastic components of data and performing forecasting for financial planning and investment.

The thesis describes the development process of the system, from data collection to visualization and analysis. Python and libraries such as Pandas, NumPy, Statsmodels, Scikit-learn, and Prophet were utilized. Mathematical models for time series forecasting are detailed, and their performance is evaluated using metrics (MAE, MAPE, RMSE).

The master's thesis consists of an introduction, three chapters, conclusions, and appendices. The introduction justifies the relevance of the topic, defines the objectives, and describes the research methodology. The first chapter discusses theoretical aspects of time series analysis. The second chapter focuses on data processing technologies and system development. The third chapter presents the results of model testing and comparison.

The thesis consists of an introduction, 3 chapters, conclusions, a list of references from 40 sources, includes 74 figures and appendices. The total volume of the work is 90 pages.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ARIMA – авторегресійна інтегрована модель на основі ковзного середнього

API – інтерфейс прикладного програмування (Application Programming Interface)

AR – авторегресійна модель

SARIMA – авторегресійна модель з урахуванням сезонності

AWS – хмарні сервіси Amazon Web Services

CSV – формат текстового файлу для зберігання табличних даних (Comma-Separated Values)

EDA – дослідницький аналіз даних (Exploratory Data Analysis)

S3 – сервіс зберігання об’єктів у хмарі Amazon (Simple Storage Service)

Python – мова програмування

MAE – середня абсолютна похибка (Mean Absolute Error)

MSE – середня квадратична похибка

ML – машинне навчання (Machine Learning)

RMSE – квадратний корінь із середньої квадратичної помилки

ВСТУП

Актуальність теми. У сучасному світі фінансові ринки відіграють ключову роль у глобальній економіці, забезпечуючи канали для розподілу капіталу та вплив на рівень економічного розвитку країн. Зростаюча складність фінансових інструментів та швидкість змін на ринку ставлять перед фахівцями складні завдання аналізу та прийняття рішень. Інформаційні технології, зокрема системи аналізу даних, відкривають нові можливості для ефективного управління та прийняття рішень у фінансовій сфері.

Зосередження фінансової інформації по різних джерелах та їх розподіл у часі здійснюють вплив на поточні показників фінансових ринків, а в кінцевому випадку збільшують ризики непередбачуваності отриманих результатів фінансової діяльності. Тому, завдання роботи з даними та розробка систем, здатних проводити аналіз даних на сьогодні є одним із ключових завдань при розробці фінансових рішень, що і обумовлює вибір напряму дослідження.

Метою даної магістерської роботи є розробка інформаційної системи для аналізу даних фінансових ринків на засадах використання машинного навчання, моделей та інструментів, які дозволять фахівцям у прикладній галузі фінансів здійснювати аналіз та прогнозування рухів на фінансових ринках.

Завданнями магістерської роботи було визначено:

1. Визначити місце аналізу даних в процесах оцінки та прийняття рішень на фінансових ринках.

2. Провести аналіз сучасних підходів та методів оцінки даних фінансових ринків із використанням моделей часових рядів.

3. Розробити інформаційну системи для аналізу та прогнозування рухів на фінансових ринках з використанням різноманітних методів аналізу даних.

4. Реалізувати аналітичні модулі для аналізу та прогнозування динаміки

фінансових інструментів на фінансових ринках на основі розроблених моделей та зібраних даних.

5. Провести реалізацію та оцінку моделей часових рядів даних на різних обсягах вибірок даних із використанням методів машинного навчання у середовищі Amazon SageMaker.

Об'єкт дослідження. Інформаційні системи аналізу даних фінансових ринків, а також їх взаємозв'язок з динамікою фінансових інструментів.

Предмет дослідження. Підходи, методи та інструменти аналізу даних для оцінки динаміки фінансових ринків.

В даній роботі були застосовані загально-наукові та емпіричні методи: аналізу та синтезу, абстрагування, моделювання, порівняння, вимірювання, експерименту.

Наукова новизна дослідження полягає в розробці інформаційної системи для аналізу даних фінансових ринків, яка інтегрує функціонування аналітичного апарату із використанням моделей часових рядів, хмарні сервіси та автоматизовані процеси збору даних. Запропонована система дозволяє здійснювати оцінку динаміки вартості та обсягів фінансових інструментів з урахуванням часових лагів.

Структура роботи. Структуру магістерської роботи складають: вступ, три розділи, висновки, список використаних джерел та додатки. Перший розділ присвячений теоретичному аспекту аналізу фінансових даних та вивченню основних понять, що використовуються у фінансовій аналітиці. Другий розділ роботи присвячена розробці та реалізації інформаційної системи для аналізу фінансових даних, яка надає можливість здійснювати аналіз та прогнозування рухів на фінансових ринках з використанням різноманітних методів аналізу даних.

Практичне значення роботи полягає у створенні інформаційної системи, що забезпечує аналіз даних інструментів на фінансових ринків в динаміці. Результати дослідження можуть бути використані фінансовими установами, трейдерами, аналітиками та іншими учасниками фінансових

ринків.

Апробація результатів дослідження:

1. Олійник Б.С., Потапова Н.А. Аналіз даних фінансових ринків на основі ARIMA моделей. Прикладні аспекти сучасних міждисциплінарних досліджень: III Міжнародна науково-практична конференція (м. Вінниця, 1 листопада 2024 р.). Вінниця: ДонНУ імені Василя Стуса, 2024.

2. Олійник Б.С., Волонтир Л.О. Аналіз даних фінансових ринків за допомогою інформаційних систем. Прикладні інформаційні технології: матеріали V Всеукраїнської науково-практичної конференції здобувачів, аспірантів та молодих вчених (м. Вінниця, 24 травня 2024 р.). Вінниця: ДонНУ імені Василя Стуса, 2024. С. 199 - 200.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ЗАСАДИ АНАЛІЗУ ДАНИХ НА ФІНАНСОВИХ РИНКАХ

1.1 Оцінка функціонування фінансових ринків на основі аналізу часових рядів

Функціонування ринкової економіки базується на функціонуванні різноманітних ринків, які можна згрупувати в два класи: ринки продукції (товарів, робіт, послуг) і ринки ресурсів (трудових, фінансових). На ринку фінансових ресурсів зустрічаються, з одного боку, суб'єкти, в яких у процесі господарювання виникає потреба в коштах для розширення діяльності, а з іншого, – суб'єкти, в яких накопичуються заощадження, що можуть бути використані для інвестицій.

Саме на ринку фінансових ресурсів (фінансовому ринку) відбувається претікання коштів, при якому вони переміщуються від тих, хто має їх надлишок, до тих, хто потребує інвестицій. На фінансовому ринку ті, що мають вільні фінансові ресурси, передають їх на різних умовах іншим учасникам ринку, які опосередковано через суб'єктів ринку або безпосередньо використовують залучені ресурси для фінансування різних галузей економіки, забезпечення потреб населення і потреб бюджетів різних рівнів. Фінансові ресурси надаються на умовах позики або на умовах співвласності, коли інвестор набуває прав власності на придбані за інвестовані кошти матеріальні чи нематеріальні активи.

Передача в користування фінансових ресурсів на фінансовому ринку оформляється тим чи іншим фінансовим інструментом. Якщо ресурси передаються на умовах позики, це оформляється відповідними інструментами позики – борговими цінними паперами (облігаціями, векселями, ощадними сертифікатами тощо) або різними видами кредитних ресурсів.

Фінансовий ринок вирішує наступні завдання:

1) надає емітентам можливість мобілізувати внутрішні джерела фінансування і тимчасово вільні кошти для довгострокових інвестицій і задоволення інших потреб;

2) забезпечує перерозподіл капіталу між його учасниками, сприяючи концентрації фінансових ресурсів у найбільш рентабельних сферах економіки;

3) надає інвесторам (юридичним і фізичним особам) можливість сформувати свій інвестиційний портфель найкращим чином з погляду збереження капіталу і з метою отримання додаткового доходу.

3) надає інвесторам (юридичним і фізичним особам) можливість сформувати свій інвестиційний портфель найкращим чином з погляду збереження капіталу і з метою отримання додаткового доходу.

Держава відіграє важливу роль на фінансовому ринку. Свій вплив вона здійснює через керування відсотком, грошовою масою, кредитами, валютним курсом. Суб'єкти фінансового ринку можуть виступати в ролі позичальника й інвестора.

Фінансові інститути відіграють роль посередника.

Позичальники – юридичні й фізичні особи, які залучають кошти інших суб'єктів для розвитку своєї діяльності.

Інвестори – громадяни і юридичні особи, фірми, держави, які приймають рішення щодо вкладання особистих, позикових або залучених коштів в об'єкти інвестування.

Фінансові інститути – це посередники між позичальниками й інвесторами. До них належать банки й спеціальні небанківські установи (страхові й інвестиційні компанії, фінансові, пенсійні, ощадні фонди)

Грошовий ринок – частина фінансового ринку, на якій здійснюються переважно короткострокові (до одного року) депозитно-позичкові операції, що обслуговують головним чином рух оборотного капіталу фірм, короткострокових ресурсів банків, установ, держави й приватних осіб.

Найбільше процес глобалізації поширився у фінансовій сфері.

Інформаційні технології дали можливість поєднати разом фінансові центри світу, скоротити час укладання угод та знизити їх вартість. Хвиля злиття й поглинань у розвинутих країнах і дерегульованість економіки останніми десятиріччями минулого століття призвели до виникнення фінансових холдингів. Фінансовий ринок став розвиватися за рахунок власних джерел, відособлюючись від реальної економіки. Держава втрачає контроль над рухом приватного капіталу, який визначає економічну ситуацію в більшості країн світу. Формується новий глобальний механізм прийняття рішень із новим складом учасників.

Проблеми аналізу фінансової інформації з метою прогнозування майбутнього руху цін та конвертації цих знань у прибутки існують з часів появи фінансових ринків. Питання чи можливо взагалі прогнозувати ціни на фінансові активи як таке не існує, оскільки це можливо. Єдине питання, яке хвилює учасників біржової діяльності – як зробити якісний прогноз та отримати конкурентну перевагу перед іншими. Тому одним із найпотужніших інструментів є аналіз даних часових рядів та інформаційні системи, здатні проводити такий аналіз за даними фінансових ринків.

Часовий ряд – це впорядкована послідовність спостережень, отриманих у певні моменти часу. Аналіз часових рядів є фундаментальним методом у дослідженні динамічних процесів у багатьох галузях: економіці, фінансах, інженерії, природничих науках тощо. Основна мета аналізу часових рядів полягає у виявленні закономірностей, трендів, сезонності та стохастичних компонентів для їх подальшого моделювання і прогнозування.

Часові ряди зазвичай складаються з кількох основних компонентів:

1. Тренд (T_t): це довгострокова зміна рівня часового ряду, яка може бути висхідною, низхідною або стабільною. Формальний опис має вигляд:

$$T_t = f(t) \qquad (1.1)$$

де t – час, а $f(t)$ – функція, яка визначає тренд.

2. Сезонність (S_t) – це періодичні коливання, які повторюються через регулярні інтервали часу (наприклад, місяці, квартали). Для сезонності характерна періодичність, яку можна описати як:

$$S_t = g(t, P) \qquad (1.2)$$

де P – період повторюваності.

3. Стохастична компонента (E_t) – це випадкові флуктуації або "шум", які неможливо пояснити іншими компонентами. Модель шуму часто припускає, що це білий шум із середнім значенням 0:

$$E_t \sim N(0, \sigma^2) \qquad (1.3)$$

де σ^2 – дисперсія.

4. Циклічність (C_t) – це довгострокові коливання, пов'язані з економічними, соціальними чи іншими факторами. Від сезонності циклічність відрізняється нерегулярністю періоду. Часові ряди класифікують за такими ознаками:

1. Детерміновані та стохастичні: детерміновані ряди описуються однозначними закономірностями, тоді як стохастичні мають випадкову природу.

2. Стаціонарні та нестаціонарні:

- Стаціонарні ряди мають сталі середні значення, дисперсію і автокореляцію, що не змінюється з часом.
- Нестаціонарні ряди містять тренди, змінну дисперсію або сезонність.

Основними методами аналізу часових рядів є:

1. Автокореляція та крос-кореляція. Автокореляційна функція (ACF) вимірює залежність між значеннями часового ряду на різних часових лагах:

$$\rho(k) = \frac{\mathrm{Cov}(X_t, X_{t-k})}{\mathrm{Var}(X_t)} \qquad (1.4)$$

де k – лаг.

2. Дисперсійний аналіз: використовується для визначення внеску сезонних та трендових компонент у загальну мінливість ряду.

3. Декомпозиція часових рядів: процес поділу ряду на трендову, сезонну та стохастичну компоненти:

- Адитивна модель:

$$Y_t = T_t + S_t + E_t \quad (1.5)$$

- Мультиплікативна модель:

$$Y_t = T_t \cdot S_t \cdot E_t \quad (1.6)$$

Виклики у роботі з часовими рядами:

1. Нестаціонарність. Багато реальних даних є нестаціонарними, що потребує попередньої обробки, наприклад, диференціювання.

2. Сезонність та гетероскедастичність. Необхідність врахування складних залежностей у даних.

3. Прогнозування. Забезпечення точності передбачень за умов високої мінливості.

Аналіз часових рядів дозволяє зрозуміти структуру даних і підготувати їх до моделювання. Вибір правильної моделі залежить від природи даних і завдань аналізу, що обумовлює необхідність використання сучасних методів, таких як ARIMA, SARIMA, Prophet та Auto ARIMA.

Моделювання часових рядів передбачає використання методів, які дозволяють аналізувати залежності в даних і прогнозувати їх поведінку в майбутньому. У цьому пункті будуть розглянуті найпоширеніші моделі для аналізу та прогнозування: ARIMA, SARIMA, Prophet та Auto ARIMA.

1. ARIMA (Autoregressive Integrated Moving Average). Модель ARIMA є базовим підходом для аналізу нестаціонарних часових рядів. Вона поєднує три основні компоненти:

1. AR (autoregressive, авторегресійна частина): використовує залежність між поточним значенням ряду та його попередніми значеннями. Форма:

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \varepsilon_t, \qquad (1.7)$$

де – $\phi_1, \phi_2, \ \ldots \ , \phi_p$ коефіцієнти авторегресії, p – порядок.

2. I (integrated, інтегрована частина): відповідає за перетворення нестаціонарного ряду в стаціонарний через операцію диференціювання:

$$Y_t' = Y_t - Y_{t-1} \qquad (1.8)$$

3. MA (moving average, рухоме середнє): моделює залежність поточного значення від попередніх похибок:

$$Y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} \qquad (1.9)$$

де $\theta_1, \theta_2, \ldots, \theta_q$– коефіцієнти, q – порядок.

Загальна форма ARIMA(p, d, q):

$$\Phi(B)(1-B)^d Y_t = \Theta(B)\varepsilon_t \qquad (1.10)$$

де B – оператор зсуву, d – порядок диференціювання.

2. SARIMA (Seasonal ARIMA). SARIMA є розширенням моделі ARIMA, яке враховує сезонність даних. У цій моделі додаються сезонні компоненти авторегресії (SAR), інтеграції (SI) та рухомого середнього (SMA). Загальна форма SARIMA$(p, d, q)(P, D, Q, s)$:

$$\Phi(B)\Phi_s(B^s)(1-B)^d(1-B^s)^D Y_t = \Theta(B)\Theta_s(B^s)\varepsilon_t, \qquad (1.11)$$

де P, D, Q – порядок сезонних компонент;

s – довжина сезонного циклу (наприклад, 12 для місяців).

SARIMA дозволяє моделювати циклічність, яка часто зустрічається у фінансових, економічних чи природничих даних.

3. Prophet. Модель Prophet, розроблена Facebook, є потужним інструментом для прогнозування часових рядів із вираженою сезонністю. Prophet побудований на адитивній моделі:

$$Y(t) = g(t) + s(t) + h(t) + \varepsilon_t \qquad (1.12)$$

де $g(t)$– тренд;

$s(t)$– сезонність;

$h(t)$– свята чи інші події;

ε_t – випадковий шум.

Особливості Prophet:

1. Простота роботи: автоматичне врахування трендів, сезонності та особливих подій.

2. Підтримка нелінійних трендів із змінами:

$$g(t) = \frac{1}{1+e^{-k(t-m)}} \qquad (1.13)$$

де k– швидкість змін, m– точка зміни тренду.

3. Гнучкість у врахуванні свят і аномалій.

4. Auto ARIMA. Auto ARIMA – це автоматизована версія ARIMA, яка використовує алгоритми для визначення оптимальних параметрів p, d, q.

Основні етапи роботи Auto ARIMA:

1. Попередня оцінка стаціонарності.

2. Вибір параметрів за допомогою критеріїв AIC (Akaike Information Criterion) або BIC (Bayesian Information Criterion).

3. Автоматичне побудування та тестування моделі.

Auto ARIMA зменшує ризик помилок, пов'язаних із ручним вибором параметрів, і значно спрощує процес моделювання.

Часові ряди відіграють ключову роль у фінансових системах, оскільки більшість даних у цій галузі є послідовними у часі. Аналіз часових рядів у фінансах дозволяє оцінювати ризики, передбачати майбутні тренди, розробляти стратегії інвестування та управляти активами. У цьому пункті розглянуто специфіку роботи з такими даними.

Основні особливості фінансових часових рядів:

1. Висока мінливість (волатильність). Фінансові дані, такі як ціни акцій чи валют, характеризуються значними коливаннями. Волатильність моделюється через:

- Стандартне відхилення (σ): оцінка міри змінності.
- GARCH-моделі (Generalized Autoregressive Conditional Heteroskedasticity): враховують змінність у часі. Формула волатильності у GARCH(1,1):

$$\sigma_t^2 = \omega + \alpha\varepsilon_{t-1}^2 + \beta\sigma_{t-1}^2 \qquad (1.14)$$

де ω, α, β – параметри моделі.

1. Сезонність. Фінансові дані демонструють характерну сезонність (наприклад, зміни обсягів торгів у різні дні тижня або роки). Для її моделювання використовуються:

- SARIMA: для врахування циклів у даних.
- Періодичні компоненти: додаються до загальної моделі як:

$$S_t = a\sin(\omega t) + b\cos(\omega t) \qquad (1.15)$$

де $\omega = \frac{2\pi}{P}$, P – період.

1. Автокореляція. Ознака, яка визначає залежність між значеннями в часі і аналізується через функцію автокореляції (ACF):

$$\rho(k) = \frac{\sum_{t=k+1}^{n}(Y_t - \overline{Y})(Y_{t-k} - \overline{Y})}{\sum_{t=1}^{n}(Y_t - \overline{Y})^2} \qquad (1.16)$$

2. Тренди та цикли. Довгострокові тренди у фінансових даних відображають економічні цикли або глобальні тенденції, наприклад, зростання фондового ринку.

Методи роботи з фінансовими часовими рядами:

1. Попередня обробка. Для коректного аналізу дані необхідно підготувати дані: видалити пропущені значення та застосувати логарифмічну трансформацію, що зменшує масштабність змін. Логарифмічна трансформація проводиться за наступною формулою:

$$Y_t' = \ln(Y_t) \qquad (1.17)$$

- Виконати нормалізацію:

$$Y_t' = \frac{Y_t - \mu}{\sigma} \qquad (1.18)$$

де μ– середнє значення, σ– стандартне відхилення.

2. Декомпозиція ряду. Для виокремлення компонент ряду (тренд, сезонність, випадкові коливання) використовуються адитивні або мультиплікативні моделі. У фінансових системах прогнози будуються для таких завдань: короткострокові прогнози для трейдингу та довгострокові прогнози для стратегічного планування.

Особливості роботи з фінансовими часовими рядами:

1. Шум у даних. Фінансові ряди містять великий обсяг випадкових коливань, які ускладнюють моделювання. Застосування фільтрів, таких як фільтр Калмана, допомагає згладити дані.

2. Нестаціонарність. Часто фінансові ряди нестаціонарні, тому потрібні

методи стабілізації, такі як диференціювання або перетворення Бокса-Кокса:

$$Y_t' = \frac{Y_t^{\lambda}-1}{\lambda} \qquad (1.19)$$

де λ – параметр трансформації.

3. Раптові зміни. Фінансові дані часто піддаються впливу екзогенних факторів (економічних, політичних), які важко передбачити. У таких випадках використовують моделі із врахуванням аномалій (наприклад, Prophet).

1.2. Бібліотеки Python для аналізу даних часових рядів.

Python є одним із найпопулярніших інструментів для аналізу часових рядів завдяки багатству бібліотек, які забезпечують простоту роботи з даними, моделювання та візуалізацію. У цьому розділі розглянуто ключові бібліотеки, які використовуються у фінансових дослідженнях.

Основні бібліотеки Python:

1. Pandas. Pandas є основною бібліотекою для роботи з табличними даними. Вона дозволяє виконувати операції попередньої обробки, трансформації та аналізу часових рядів. Приклад завантаження та обробки даних:

```
import pandas as pd

# Завантаження даних
df = pd.read_csv('financial_data.csv', parse_dates=['Date'], index_col='Date')

# Нормалізація даних
df['Normalized'] = (df['Close'] - df['Close'].mean()) / df['Close'].std()
```

2. NumPy. Бібліотека для чисельних обчислень, яка використовується для обробки масивів даних і виконання математичних операцій. Наприклад:

```
import numpy as np

# Розрахунок ковзного середнього
df['Moving Average'] = df['Close'].rolling(window=5).mean()
```

3. Statsmodels. Ця бібліотека надає інструменти для створення статистичних моделей, зокрема ARIMA, SARIMA та інших:

```
from statsmodels.tsa.arima.model import ARIMA

# Побудова моделі ARIMA
model = ARIMA(df['Close'], order=(1, 1, 1))
results = model.fit()
print(results.summary())
```

4. Prophet. Інструмент від Facebook для роботи з даними, що мають виражену сезонність і тренди:

```
from prophet import Prophet

# Підготовка даних
prophet_df = df.reset_index()[['Date', 'Close']].rename(columns={'Date': 'ds', 'Close': 'y'})

# Моделювання
model = Prophet()
model.fit(prophet_df)

# Прогнозування
future = model.make_future_dataframe(periods=30)
forecast = model.predict(future)
```

5. Scikit-learn. Бібліотека для машинного навчання, яка дозволяє створювати та тестувати моделі регресії, класифікації та кластеризації. Для фінансових рядів часто використовується для попередньої обробки або побудови гібридних моделей. Для візуалізації фінансових часових рядів застосовують бібліотеки:

1. Matplotlib.

```
import matplotlib.pyplot as plt

# Побудова графіка
df['Close'].plot(title='Stock Prices Over Time')
plt.show()
```

2. Seaborn. Використовується для більш деталізованих графіків і heatmap.

```
import seaborn as sns

# Графік розподілу даних
sns.histplot(df['Close'], kde=True)
```

3. Plotly. Інтерактивна бібліотека для побудови графіків:

```
import plotly.express as px

fig = px.line(df, x=df.index, y='Close', title='Stock Prices')
fig.show()
```

Переваги використання Python для фінансового аналізу:

1. Гнучкість і багатофункціональність: Python дозволяє поєднувати кілька бібліотек для вирішення складних задач.
2. Велика спільнота: численні ресурси та приклади.
3. Автоматизація: можливість створення скриптів для обробки великих обсягів фінансових даних.

Python забезпечує зручний та ефективний інструментарій для роботи з часовими рядами. Завдяки поєднанню бібліотек Pandas, NumPy, Statsmodels, Prophet та Scikit-learn дослідники мають змогу здійснювати повний цикл обробки даних: від попередньої обробки до прогнозування та візуалізації.

1.3 Метрики оцінки точності моделей часових рядів

Оптимізація моделей прогнозування часових рядів є важливим етапом для забезпечення точності прогнозів. Вона передбачає налаштування параметрів моделі, тестування її продуктивності та вибір найкращого варіанта на основі певних метрик.

Методи оптимізації параметрів:

1. Крос-валідація для часових рядів. Крос-валідація дозволяє оцінювати продуктивність моделі, розділяючи дані на навчальний і тестовий набори. У випадку часових рядів використовують послідовну крос-валідацію, коли тестовий набір даних знаходиться після навчального.

Алгоритм:

1. Поділити дані на кілька підмножин.

2. Навчати модель на попередніх підмножинах і перевіряти точність на наступних.

3. Розрахувати середнє значення помилки на всіх ітераціях.

Приклад реалізації в Python:

```
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error

tscv = TimeSeriesSplit(n_splits=5)
for train_index, test_index in tscv.split(date):
    train, test = date[train_index], date[test_index]
    model.fit(train)
    predictions = model.predict(test)
    error = mean_squared_error(test, predictions)
```

2. Підбір параметрів моделі. Для моделей ARIMA та SARIMA підбір параметрів здійснюється вручну або автоматично.

- Ручний підбір: вибір параметрів p, d, q, (ARIMA) або P, D, Q, s (SARIMA) на основі аналізу автокореляційної (ACF) та часткової автокореляційної (PACF) функцій.
- Автоматизований підбір: Auto ARIMA використовує алгоритми пошуку найкращої комбінації параметрів на основі критеріїв AIC чи BIC.

3. Оптимізація в Prophet. У Prophet оптимізація здійснюється шляхом налаштування таких параметрів:

- Гнучкість тренду: параметр change pointpriorscalechangepoint_ prior_scalechangepointpriorscale визначає чутливість до змін тренду.
- Сезонність: параметр seasonalitypriorscaleseasonality_prior_ scaleseasonalitypriorscale регулює вагу сезонної компоненти.

Приклад у Python:

```
from prophet import Prophet

model = Prophet(changepoint_prior_scale=0.1, seasonality_prior_scale=10)
model.fit(date)
```

Метрики оцінки точності моделей:

Для оцінки точності прогнозів використовуються різні метрики, серед яких найпоширенішими є:

1. Mean Absolute Error (MAE): середня абсолютна помилка:

$$MAE = \frac{1}{n}\sum_{t=1}^{n}\left|Y_t - \widehat{Y}_t\right|, \qquad (1.20)$$

де Y^t – фактичне значення, $\widehat{Y}_t$ – прогнозоване значення.

2. Mean Squared Error (MSE): середня квадратична помилка:

$$MSE = \frac{1}{n}\sum_{t=1}^{n}\left(Y_t - \widehat{Y}_t\right)^2. \qquad (1.21)$$

3. Root Mean Squared Error (RMSE): квадратний корінь із середньої квадратичної помилки:

$$RMSE = \sqrt{MSE}. \qquad (1.22)$$

Після оптимізації та оцінки точності кожної моделі здійснюється їх порівняння. Наприклад:

- ARIMA може бути найкращою для короткострокових прогнозів.
- SARIMA підходить для даних із вираженою сезонністю.
- Prophet є ефективним для нелінійних трендів і врахування спеціальних подій.

ВИСНОВКИ З РОЗДІЛУ 1

В даному розділі висвітлюються питання теоретичного піґрунття використання моделей часових рядів в аналізі даних фінансових ринків. Фінансовий ринок є джерелом потоків коштів, з якого відбувається їх перетікання від власників надлишку ресурсів, до тих, хто потребує фінансових інвестицій. З'ясовано, що динамічність розвитку фінансових ринків характеризується непередбачуваністю та високими ризиками, що робиться його залежним від оцінювання та прогнозів поточної статистичної інформації. Короткострокова інформація може бути отримана як за даними бірж, так і з урахуванням фінансових повідомлень та новин. Така інформація потребує детального аналізу даних, особливо з розробкою моделей, які покращують прийняття рішень і можуть застосовуватись в автоматизованому технічному режимі.

Аналіз класу таких моделей показав, що для інтеграції в інформаційну систему аналізу даних можуть бути задіяні моделі типу: ARIMA, SARIMA, Prophet та Auto ARIMA. Вони є потужними інструментами для аналізу часових рядів і прогнозування. Їх використання залежить від структури даних, наявності сезонності та обсягу вхідної інформації. Auto ARIMA забезпечує автоматизацію процесу, тоді як Prophet дозволяє враховувати додаткові аспекти, такі як свята чи нелінійні тренди.

Аналіз фінансових часових рядів вимагає врахування специфічних особливостей, таких як волатильність, сезонність і наявність шуму. Для обробки цих даних застосовуються сучасні методи, які дозволяють розробляти складні прогнози майбутніх значень і створювати стратегії управління ризиками.

РОЗДІЛ 2
ТЕХНОЛОГІЇ АНАЛІЗУ ДАНИХ ФІНАНСОВИХ РИНКІВ

2.1 Етапи підготовки даних для проведення аналізу

Підготовка даних є одним із найважливіших етапів у побудові прогнозних моделей. Вона передбачає трансформацію вихідних даних у зручний для аналізу формат, виявлення трендів, видалення шумів і забезпечення стаціонарності. Підготовка даних складається із наступних етапів:

1. Збирання та очищення даних. Сировинні дані часто містять пропущені значення, викиди або дублікати, які впливають на точність моделей. Обробка пропущених значень: використовується інтерполяція або видалення. Наприклад:

```
df['Close'] = df['Close'].interpolate(method='linear')
```

2. Видалення викидів: викиди визначаються як значення, що виходять за межі $\mu \pm 3\sigma$, де μ – середнє значення, σ – стандартне відхилення.

3. Стаціонаризація часового ряду. Більшість моделей припускає, що дані стаціонарні. Для досягнення стаціонарності використовують:

4. Диференціювання:

$$Y_t' = Y_t - Y_{t-1}.$$

Реалізація в Python:

```
df['Diff'] = df['Close'].diff().dropna()
```

5. Логарифмічна трансформація:

$$Y_t' = \ln(Y_t).$$

```
df['Log'] = np.log(df['Close'])
```

6. Декомпозиція часових рядів. Для виділення компонент (тренд, сезонність, залишки) застосовується адитивна або мультиплікативна модель. Приклад із бібліотеки Statsmodels:

```
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(df['Close'], model='additive')
result.plot()
```

Для забезпечення однакових масштабів даних використовуються:

1. Стандартизація:

$$Y_t' = \frac{Y_t - \mu}{\sigma}.$$

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['Standardized'] = scaler.fit_transform(df[['Close']])
```

2. Мінмакс-нормалізація:

$$Y_t' = \frac{Y_t - Y_{min}}{Y_{max} - Y_{min}}.$$

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['Normalized'] = scaler.fit_transform(df[['Close']])
```

Формування навчального та тестового наборів включає поділ та розширення набору даних:

1. Поділ даних: дані поділяються на навчальний і тестовий набори у співвідношенні 70:30.

```
train_size = int(len(df) * 0.8)
train, test = df[:train_size], df[train_size:]
```

2. Розширення набору даних: у разі недостатнього обсягу даних використовують методи доповнення (augmentation), такі як зсуви чи випадкове додавання шуму.

2.2. Обґрунтування процесу тестування моделей

Аналіз результатів тестування є ключовим етапом у виборі найефективнішої моделі прогнозування. Метою є порівняння моделей за допомогою метрик точності, візуалізації помилок і оцінки прогнозів на

тестовій вибірці. Візуалізація результатів порівняння дозволяє в наочні формі представити оцінку метрик при порівнянні фактичних та прогнозних значень. Розглянемо складові візуалізації даних при оцінці фінансових ринків:

1. Порівняння фактичних та прогнозованих значень використовується для оцінки якості прогнозу. Приклад у Python:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(test.index, test['Close'], label='Фактичні значення')
plt.plot(test.index, predictions, label='Прогноз')
plt.legend()
plt.title('Порівняння прогнозу і фактичних даних')
plt.show()
```

2. Графік помилок відображає розподіл помилок прогнозу:

```
plt.figure(figsize=(10, 6))
plt.hist(test['Close'] - predictions, bins=20, edgecolor='k')
plt.title('Розподіл помилок')
plt.show()
```

3. ACF і PACF залишків. Для перевірки незалежності залишків прогнозу від фактичних значень:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

plot_acf(residue)
plot_pacf(residue)
plt.show()
```

Порівняння моделей проводиться на основі метрик точності (MAE, RMSE, MAPE) та візуалізації прогнозів. Для вибору найкращої моделі важливі такі аспекти:

1. Низькі значення помилок. Модель із найменшими значеннями MAE, RMSE та MAPE зазвичай вважається найкращою.

2. Стаціонарність залишків. Залишки прогнозу повинні бути незалежними та відповідати білому шуму.

3. Сезонність і тренди. Модель повинна точно враховувати ці

компоненти, особливо для фінансових часових рядів.

Аналізуючи моделі часових рядів можна сказати, що вони мають наступні особливості щодо застосування їх в аналітиці даних:

- ARIMA – забезпечує стабільні прогнози для короткострокових періодів із трендами.
- SARIMA – краще справляється з даними із вираженою сезонністю.
- Prophet – ефективний для складних трендів і врахування зовнішніх факторів, таких як свята.
- Auto ARIMA – автоматизований підхід, що забезпечує швидкий вибір параметрів.

2.3. Методи та моделі для покращення прогнозів в аналізі даних

Покращення точності прогнозування часових рядів є важливим завданням у фінансових дослідженнях. Це досягається шляхом вдосконалення підходів до обробки даних, налаштування моделей і інтеграції додаткових факторів. Методи, які застосовують для покращення прогнозів поділяють на:

1. Застосування гібридних моделей. Гібридні моделі поєднують переваги різних методів. Наприклад:

- Комбінація ARIMA і нейронних мереж: ARIMA моделює лінійні тренди, а нейронна мережа враховує нелінійності.

```
from sklearn.neural_network import MLPRegressor
model = MLPRegressor(hidden_layer_sizes=(100,), max_iter=1000)
model.fit(features, residue)
```

2. Використання екзогенних змінних (мультиваріантні моделі). Додавання зовнішніх факторів, таких як макроекономічні показники чи сезонні події, може суттєво покращити прогноз:

- Для SARIMAX можна інтегрувати додаткові змінні:

$$Y_t = \Phi(B)X_t + \Theta(B)\varepsilon_t,$$

Де X_t– екзогенні змінні.

3. Аугментація даних. Розширення обсягу даних шляхом генерації синтетичних даних або використання доповнень, таких як:

- Шум:

```
augmented_data = date + np.random.normal(0, sigma, size=date.shape)
```

- Лінійні перетворення (масштабування, зсув).
- Покращення попередньої обробки. Використання фільтрів, таких як фільтр Ходріка-Прескотта (Hodrick-Prescott Filter), для згладжування трендів:

$$\min_t \sum_t (Y_t - T_t)^2 + \lambda \sum_t [(T_t - T_{t-1}) - (T_{t-1} - T_{t-2})]^2.$$

4. Паралельне прогнозування. Використання ансамблів моделей (наприклад, середнього прогнозу кількох методів):

```
final_forecast = (forecast_arima + forecast_sarima + forecast_prophet) / 3
```

5. Оптимізація параметрів. Використання бібліотек для автоматизації пошуку параметрів, таких як Optuna чи GridSearchCV:

```
from sklearn.model_selection import GridSearchCV
parameters = {'p': [0, 1, 2], 'd': [0, 1], 'q': [0, 1, 2]}
model = ARIMA()
grid = GridSearchCV(model, parameters)
grid.fit(date)
```

6. Застосування машинного навчання. Інструменти автоматизованого машинного навчання (наприклад, H2O AutoML) дозволяють знаходити найкращі моделі для прогнозів із мінімальними витратами часу.

7. Використання додаткових джерел даних:

- Сентимент-аналіз новин. Аналіз настроїв у новинах і соціальних мережах для доповнення моделей. Наприклад, з використанням бібліотек NLP:

```
from transformers import pipeline
sentiment_analysis = pipeline("sentiment-analysis")
sentiment = sentiment_analysis("Stock prices are expected to rise tomorrow")
```

- Історичні дані. Використання довготривалих історичних даних для покращення прогнозів.

ВИСНОВКИ З РОЗДІЛУ 2

В даному розділі висвітлюються питання використання технологій аналізу даних фінансових ринків в інформаційній системі. Одним із основних технологічних приймів роботи з даними є їх попередня підготовка. Підготовка даних є етапом, який забезпечує коректну роботу моделей прогнозування, яка охоплює процес обробки пропущених значень, стаціонаризацію та нормалізацію, що впливає на точність та продуктивність моделей.

Покращення моделей прогнозування є необхідною умовою для досягнення високої точності прогнозів. Використання методів крос-валідації, підбору параметрів та оцінки за метриками дозволяє вибрати модель, яка найкраще відповідає специфіці даних і завданням дослідження.

Аналіз результатів тестування дозволяє оцінити якість моделей і вибрати найкращу для прогнозування. Метрики точності та візуалізація результатів надають можливість об’єктивно порівняти моделі. Для аналізу даних фінансових часових рядів найбільш доцільним для опису сезонних даних є моделі SARIMA і для складних нелінійних трендів – Prophet.

Покращення прогнозів значною мірою залежить від урахування додаткових факторів, процесів фільтрації, автоматизації розрахунку параметрів та використання сучасних гібридних моделей. Поєднання екзогенних змінних, ансамблів моделей і технік обробки даних дозволяє досягати високої точності при дослідженнях та обробці складних часових рядів.

РОЗДІЛ 3
ІНФОРМАЦІЙНА СИСТЕМА АНАЛІЗУ ДАНИХ ФІНАНСОВИХ РИНКІВ НА ОСНОВІ ЧАСОВИХ РЯДІВ

3.1. Розробка та імплементація програмних модулів

Розглянемо основні програмні модулі, що є складовими фнормаційної системи аналізу даних.

data_analysis.py

```
from dash import dcc, html
from dash.dependencies import Input, Output, State
import dash
import plotly.graph_objs as go
from data_collection import fetch_financial_data
from data_processing import clean_data
from financial_data_analysis import apply_arima_model, apply_sarima_model, apply_prophet_model, auto_arima_model
import logging
```

Рисунок 3.1 – Імпорт бібліотек

- dash: Використовується для створення вебдодатків з інтерактивними графіками та елементами інтерфейсу.
- dcc (Dash Core Components) та html: Компоненти Dash для створення елементів інтерфейсу, таких як текстові поля, кнопки та графіки.
- plotly.graph_objs: Модуль для створення інтерактивних графіків.
- logging: Використовується для ведення журналу (логування) інформації про роботу програми.
- Імпортуються функції з інших модулів: fetch_financial_data, clean_data для збору та очищення даних, а також моделі для прогнозування: apply_arima_model, apply_sarima_model, apply_prophet_model, auto_arima_model.

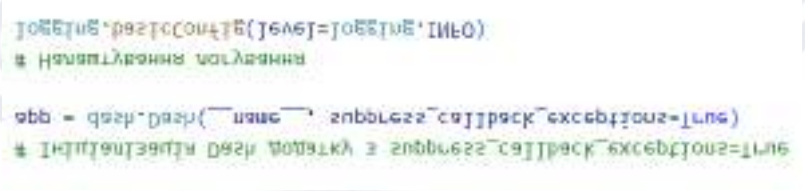

Рисунок 3.2 – Ініціалізація Dash-додатку та логування

- app: Ініціалізує Dash-додаток. Параметр suppress_callback_exceptions=True дозволяє працювати з callback функціями, які залежать від динамічно створених компонентів.
- logging.basicConfig(level=logging.INFO): Налаштовує логування для запису інформаційних повідомлень у лог-файли або консоль.

Рисунок 3.3 – Основний макет (layout) додатку

- app.layout: Описує структуру вебсторінки додатку. Всі елементи інтерфейсу додатку розміщені всередині контейнера html.Div з певними стилями (відступи та шрифти).
- Структура включає заголовки, текстові поля для вводу символу акцій, випадаючі меню для вибору моделей, слайдер для вибору кількості днів прогнозування, а також кнопку для запуску моделі.

Основні елементи інтерфейсу:

1. html.H1: Заголовок сторінки.
2. dcc.Input: Текстове поле для вводу символу акцій (наприклад, AAPL).
3. dcc.Dropdown: Випадаюче меню для вибору моделі прогнозування (ARIMA, SARIMA, Prophet, Auto ARIMA).
4. dcc.Slider: Слайдер для вибору кількості днів прогнозування.
5. html.Button: Кнопка для запуску моделі.
6. callback: Викликається при зміні значення у випадаючому меню для вибору моделі. В залежності від обраної моделі, показуються або приховуються параметри для ARIMA чи SARIMA.

```python
# Контроль видимості параметрів моделі
@app.callback(
    [Output('arima-params', 'style'),
     Output('sarima-params', 'style')],
    Input('model-dropdown', 'value')
)
def update_params_visibility(selected_model):
    if selected_model == 'ARIMA':
        return {'display': 'block'}, {'display': 'none'}
    elif selected_model == 'SARIMA':
        return {'display': 'none'}, {'display': 'block'}
    else:
        return {'display': 'none'}, {'display': 'none'}
```

Рисунок 3.4 – Контроль параметрів моделей

- Якщо користувач вибрав модель ARIMA, блок з параметрами ARIMA (arima-params) стає видимим, а параметри SARIMA приховуються. Якщо вибрано SARIMA, то навпаки.

Рисунок 3.5 – Основна логіка прогнозування та оновлення графіка

- callback: Основна функція, що відповідає за оновлення графіка і відображення прогнозу.
- Вхідні параметри: Значення обраної моделі, кількість днів для прогнозування, символ акцій, параметри моделей ARIMA і SARIMA.
- forecast_days: Кількість днів, на які необхідно зробити прогноз.
- n_clicks: Лічильник кількості кліків на кнопку "Запустити модель". Використовується для запуску процесу прогнозування.

- State: Використовується для збереження параметрів ARIMA і SARIMA.

```
# Завантаження та обробка даних для введеного символу
data = fetch_financial_data(stock_symbol)
if data is None or data.empty:
    return go.Figure(), f"Помилка: Не вдалося завантажити дані для символу {stock_symbol}."

cleaned_data = clean_data(data)
if cleaned_data is None or cleaned_data.empty:
    return go.Figure(), "Помилка: Дані недостатні для аналізу після очищення."
```

Рисунок 3.6 – Обробка даних і прогнозування

- fetch_financial_data: Завантажує історичні ринкові дані для введеного символу акцій (наприклад, AAPL).
- clean_data: Очищає дані, видаляючи відсутні значення та обчислюючи нові параметри, такі як відсоткова зміна ціни.

```
if selected_model == 'ARIMA':
    if None in [arima_p, arima_d, arima_q]:
        raise dash.exceptions.PreventUpdate
    results = apply_arima_model(cleaned_data['Close'], forecast_days, (arima_p, arima_d, arima_q))
elif selected_model == 'SARIMA':
    if None in [sarima_p, sarima_d, sarima_q, sarima_m]:
        raise dash.exceptions.PreventUpdate
    results = apply_sarima_model(
        cleaned_data['Close'],
        order=(sarima_p, sarima_d, sarima_q),
        seasonal_order=(sarima_p, sarima_d, sarima_q, sarima_m),
        forecast_days=forecast_days
    )
elif selected_model == 'Prophet':
    results = apply_prophet_model(cleaned_data['Close'], forecast_days)
elif selected_model == 'AutoARIMA':
    results = auto_arima_model(cleaned_data['Close'], forecast_days)
```

Рисунок 3.7 – Виклик моделі для прогнозування

- apply_arima_model: Якщо обрана модель ARIMA, функція викликає ARIMA модель з вказаними параметрами (p, d, q).
- apply_sarima_model: Якщо обрана модель SARIMA, використовується ця модель з сезонними параметрами (P, D, Q, m), які можна налаштувати.
- apply_prophet_model: Якщо обрана модель Prophet, для прогнозування використовується ця модель.

- auto_arima_model: Якщо обрана модель Auto ARIMA, вона автоматично вибирає найкращі параметри для прогнозування.

```
if results is None:
    return go.Figure(), "Помилка: не вдалося виконати прогнозування."
```

Рисунок 3.8 – Перевірка результатів

- Після запуску моделі перевіряється, чи отримані результати прогнозування. Якщо результатів немає (тобто прогноз не вдалося виконати), виводиться повідомлення про помилку.

```
predicted_data = results['predictions']
mse_value = results.get('MSE', None)
```

Рисунок 3.9 – Візуалізація результатів прогнозу

- predicted_data: Дані, що містять прогнозовані ціни акцій.
- mse_value: Середньоквадратична похибка (MSE), якщо вона доступна для моделі.

```
# Створення фігури
fig = go.Figure()
fig.add_trace(go.Scatter(x=cleaned_data.index, y=cleaned_data['Close'], mode='lines', name='Actual Prices'))
fig.add_trace(go.Scatter(x=predicted_data.index, y=predicted_data, mode='lines', name='Predicted Prices', line=dict(color='red', dash='dash')))
```

Рисунок 3.10 – Створення графіка

- fig = go.Figure(): Створення нового графіка (фігури).
- add_trace: Додає дві лінії на графік: одну для реальних даних (ціни закриття акцій), іншу для прогнозованих даних. Прогнозована лінія відображається червоним кольором і пунктиром для кращої видимості.

```
fig.update_layout(
    title=f'Прогноз цін акцій для {stock_symbol}',
    xaxis_title='Дата',
    yaxis_title='Ціна закриття',
    legend_title='Дані',
    hovermode='x'
)
```

Рисунок 3.11 – Налаштування графіка

- title: Заголовок графіка, що відображає символ акцій.
- xaxis_title, yaxis_title: Підпис для осей X та Y (дата і ціна закриття відповідно).
- hovermode='x': Увімкнення режиму, коли при наведенні курсора на будь-яку точку графіка відображатимуться значення для цієї дати.

```
if mse_value is not None:
    mse_output = f"Середньоквадратична похибка (MSE): {mse_value:.2f}"
else:
    mse_output = "MSE недоступна для обраної моделі."
```

Рисунок 3.12 – Виведення середньоквадратичної похибки (MSE)

- Якщо доступне значення середньоквадратичної похибки **(**MSE**),** воно відображається користувачу. В іншому випадку, виводиться повідомлення про недоступність MSE**.**

```
except Exception as e:
    logging.error(f"Error occurred: {e}")
    return go.Figure(), f"Помилка: Не вдалося виконати прогнозування. Деталі: {str(e)}"
```

Рисунок 3.13 – Обробка помилок

- try-except: Усі можливі помилки обробляються у блоці except. Якщо виникає будь-яка помилка, програма не аварійно завершується, а виводить користувачу відповідне повідомлення та записує помилку у лог.

```
if __name__ == '__main__':
    app.run_server(debug=True, host='0.0.0.0', port=8000)
```

Рисунок 3.14 – Запуск додатку

app.run_server: Запускає сервер для Dash-додатку. Параметри:

- debug=True: Увімкнення режиму налагодження, що дозволяє бачити помилки та змінювати код без перезапуску сервера.

- host='0.0.0.0': Вказує, що сервер доступний для підключень з будь-якої IP-адреси.
- port=8000: Вказує порт, на якому запуститься додаток (у цьому випадку, порт 8000).

Даний код створює повноцінний вебдодаток для прогнозування цін акцій за допомогою моделей ARIMA, SARIMA, Prophet та Auto ARIMA. Інтерфейс дозволяє вибирати модель, налаштовувати параметри і відображати графік з реальними і прогнозованими даними.

data_collection.py

```
import yfinance as yf
import pandas as pd
import os
import logging
```

Рисунок 3.15 – Імпорт бібліотек

- yfinance: Бібліотека для завантаження фінансових даних із Yahoo Finance.
- pandas: Використовується для роботи з табличними даними (DataFrame), зокрема для обробки фінансових даних.
- os: Модуль для роботи з файловою системою (наприклад, перевірка наявності файлів або створення файлів).
- logging: Використовується для створення логів, що допомагають відстежувати виконання програми або обробляти помилки.

```
def fetch_financial_data(symbol, period="max", cache=True):
    logging.info(f"Завантаження даних для символу {symbol} за період {period}...")
```

Рисунок 3.16 – Функція fetch_financial_data

fetch_financial_data: Функція завантажує фінансові дані для заданого символу акцій (наприклад, "AAPL") на основі вказаного періоду часу (наприклад, за останній рік, місяць або максимальний доступний період). Функція має три параметри:

- symbol: Символ акцій, для яких потрібно завантажити дані

(наприклад, AAPL для Apple Inc.).

- period: Період, за який потрібно отримати дані. За замовчуванням – максимальний можливий період ("max").
- cache: Прапорець, що вказує, чи потрібно кешувати дані у файл для прискорення повторних завантажень. За замовчуванням – True (використовується кешування).

Рисунок 3.17 – Перевірка кешованих даних

- cache_file: Створює назву файлу для кешу, використовуючи символ акцій та період (наприклад, AAPL_1y.csv для Apple за рік).
- os.path.exists(cache_file): Перевіряє, чи існує файл із кешованими даними для заданого символу і періоду.
- pd.read_csv: Якщо файл існує, дані завантажуються з файлу кешу у форматі CSV.
- parse_dates=True: Вказує Pandas автоматично перетворювати стовпець "Date" на формат дати.
- tz_convert(None): Видаляє інформацію про часові зони з індексу, щоб уникнути помилок у часі.
- yf.Ticker(symbol): Створює об'єкт Ticker для символу акцій, використовуючи бібліотеку yfinance.
- stock.history(period=period): Завантажує історичні фінансові дані для акцій на основі вказаного періоду. Якщо період дорівнює "max", будуть завантажені всі доступні дані.
- if data.empty: Якщо Yahoo Finance не може знайти дані для символу або дані порожні, виводиться попередження, і функція повертає None.
- data.to_csv(cache_file): Якщо кешування увімкнене, завантажені дані записуються у файл CSV для подальшого використання.

```
else:
    # Отримання об'єкта Ticker
    stock = yf.Ticker(symbol)

    # Завантаження історичних даних
    data = stock.history(period=period)

    if data.empty:
        logging.warning(f"Дані для символу {symbol} не знайдено або дані порожні.")
        return None

    # Видаляємо часову зону з індексу
    data.index = data.index.tz_convert(None)

    # Кешування даних
    if cache:
        data.to_csv(cache_file)
        logging.info(f"Дані кешовано у {cache_file}")

    logging.info(f"Успішне завантаження даних для {symbol}.")
```

Рисунок 3.18 – Завантаження даних з Yahoo Finance

```
# Приклад використання
if __name__ == "__main__":
    symbol = 'AAPL'  # Приклад: Apple Inc.
    data = fetch_financial_data(symbol, period="1y", cache=True)

    if data is not None:
        print(data.head())  # Вивести перші рядки даних
```

Рисунок 3.19 – Приклад використання

- symbol = 'AAPL': У прикладі використовується символ акцій Apple Inc.
- fetch_financial_data(symbol, period="1y", cache=True): Завантажуються дані для Apple за останній рік, з використанням кешування.
- print(data.head()): Якщо дані були успішно завантажені, виводяться перші п'ять рядків отриманих даних для перегляду.

Даний код реалізує функціонал для завантаження фінансових даних із Yahoo Finance із можливістю кешування даних для повторного використання та зменшення часу завантаження в майбутньому.

data_processing.py

```
from sklearn.preprocessing import StandardScaler
import pandas as pd
import logging
```

Рисунок 3.20 – Імпорт бібліотек

- StandardScaler: Модель зі бібліотеки scikit-learn (sklearn), яка використовується для нормалізації даних. Вона змінює масштаб даних таким чином, що середнє значення кожного показника дорівнює нулю, а стандартне відхилення дорівнює одиниці.
- pandas: Бібліотека для роботи з даними у форматі таблиць (DataFrame), зокрема для обробки фінансових даних.
- logging: Використовується для ведення журналу (логування) інформації про виконання програми та обробку помилок.

```
def clean_data(data, ma_periods=(50, 200)):
    """
```

Рисунок 3.21 – Функція clean_data

- clean_data: Функція для очищення та попередньої обробки фінансових даних. Видаляє відсутні значення та додає нові показники, такі як ковзні середні (moving averages) та відсоткова зміна ціни закриття.
- data (DataFrame): Вхідні дані, які потрібно очистити (наприклад, дані про ціни акцій).
- ma_periods: Кортеж, який містить два значення для обчислення ковзних середніх (за замовчуванням – 50 та 200 днів).

Основні етапи функції:

1. Перевірка наявності стовпця 'Close':

if 'Close' not in data.columns:

raise ValueError("Стовпець 'Close' відсутній у даних.")

Перевіряється, чи є у вхідних даних стовпець із цінами закриття акцій ("Close"). Якщо стовпець відсутній, функція викликає помилку.

2. Очищення даних від відсутніх значень:

cleaned_data = data.dropna()

Видаляються всі рядки, які містять відсутні значення (NaN).

3. Додавання відсоткової зміни ціни закриття:

cleaned_data['daily_pct_change'] = cleaned_data['Close'].pct_change() * 100

logging.info("Додано стовпець 'daily_pct_change'.")

Обчислюється відсоткова зміна ціни закриття акцій і додається як новий стовпець "daily_pct_change". Це дозволяє аналізувати, на скільки відсотків змінилася ціна порівняно з попереднім днем.

4. Додавання ковзних середніх:

cleaned_data[f'{ma_periods[0]}_day_MA'] = cleaned_data['Close'].rolling(window=ma_periods[0]).mean()

cleaned_data[f'{ma_periods[1]}_day_MA'] = cleaned_data['Close'].rolling(window=ma_periods[1]).mean()

Обчислюються ковзні середні (moving averages) для періодів 50 та 200 днів (можна змінити за необхідності). Ці ковзні середні часто використовуються в технічному аналізі для визначення трендів.

5. Видалення рядків з NaN після обчислення ковзних середніх:

cleaned_data = cleaned_data.dropna()

Після обчислення ковзних середніх видаляються рядки, де значення можуть бути відсутні через недостатню кількість даних на початку періоду.

6. Повернення очищених даних:

return cleaned_data

```
def normalize_data(data, exclude_columns=None):
    """
```

Рисунок 3.22 – Функція normalize_data

- normalize_data: Ця функція нормалізує дані, щоб усі показники мали однаковий масштаб. Нормалізація допомагає у випадках, коли різні ознаки мають різні масштаби, що може вплинути на результати аналізу або моделі машинного навчання.
- data (DataFrame): Дані для нормалізації.
- exclude_columns: Необов'язковий список колонок, які потрібно виключити з процесу нормалізації (наприклад, стовпець з обсягом торгів).

Основні етапи функції:

1. Вибір даних для нормалізації:

```
if exclude_columns:
    data_to_normalize = data.drop(columns=exclude_columns)
    logging.info(f"Колонки {exclude_columns} виключено з нормалізації.")
else:
    data_to_normalize = data
```

Якщо вказані колонки, які не потрібно нормалізувати (наприклад, обсяг торгів "Volume"), вони виключаються з даних для нормалізації.

2. Нормалізація даних:

```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data_to_normalize)
```

Використовується StandardScaler для нормалізації всіх числових значень так, щоб середнє значення було 0, а стандартне відхилення – 1.

3. Створення DataFrame з нормалізованими даними:

```
normalized_data = pd.DataFrame(scaled_data, index=data.index, columns=data_to_normalize.columns)
```

Після нормалізації дані перетворюються назад у DataFrame із збереженням початкових індексів та назв колонок.

4. Додавання виключених колонок**:**

```
if exclude_columns:
    normalized_data = pd.concat([normalized_data, data[exclude_columns]], axis=1)
```

Якщо були виключені колонки (наприклад, "Volume"), вони додаються назад до нормалізованих даних.

Повернення нормалізованих даних:

return normalized_data

```
# Приклад використання
if __name__ == "__main__":
    from data_collection import fetch_financial_data
    data = fetch_financial_data('AAPL')
    cleaned_data = clean_data(data)
    normalized_data = normalize_data(cleaned_data, exclude_columns=['Volume'])
    print(normalized_data.head())
```

Рисунок 3.23 – Приклад використання

- fetch_financial_data: Використовується функція з попереднього файлу data_collection.py для завантаження фінансових даних для Apple Inc. (AAPL).
- clean_data: Викликається функція для очищення даних і додавання ковзних середніх та відсоткових змін.
- normalize_data: Нормалізує дані, виключаючи з процесу нормалізації колонку "Volume" (обсяги торгів).
- print(normalized_data.head()): Виводить перші 5 рядків нормалізованих даних для перегляду результатів.

Даний код забезпечує підготовку фінансових даних, включаючи їх очищення та нормалізацію, що є важливими етапами для подальшого аналізу та використання в моделях машинного навчання.

financial_data_analysis.py

```
import pandas as pd
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error
import logging
import pmdarima as pm
```

Рисунок 3.24 – Імпорт бібліотек

- pandas: Для роботи з даними у вигляді таблиць (DataFrame та Series).
- numpy: Використовується для математичних операцій.
- ARIMA: Модель ARIMA з бібліотеки statsmodels для роботи з часовими рядами.
- SARIMAX: Модель SARIMA (сезонна ARIMA) з statsmodels для аналізу часових рядів з сезонністю.
- mean_squared_error: Функція з scikit-learn для обчислення середньоквадратичної похибки (MSE).
- pmdarima: Бібліотека для автоматичного підбору параметрів моделі ARIMA (auto ARIMA).
- logging: Для ведення журналу (логування) подій і помилок.

```
def apply_arima_model(data, forecast_days=10, order=(1, 1, 1), calculate_mse=True):
    """
```

Рисунок 3.25 – Функція apply_arima_model

- apply_arima_model: Застосовує модель ARIMA до вхідного часового ряду для прогнозування.
- data: Вхідні дані (часовий ряд, наприклад, ціни закриття акцій).
- forecast_days: Кількість днів для прогнозування (за замовчуванням 10 днів).
- order: Параметри (p, d, q) для моделі ARIMA (за замовчуванням (1, 1, 1)).
- calculate_mse: Прапорець для вказівки, чи потрібно обчислювати MSE (за замовчуванням True).

Основні етапи функції:

1. Підготовка даних:

data.index = pd.to_datetime(data.index, utc=True).tz_convert(None)

data = data.asfreq('D').fillna(method='ffill')

- Конвертує індекс в DateTimeIndex і прибирає часову зону.
- Встановлює частоту даних як щоденну і заповнює відсутні значення

методом forward-fill.

2. Створення і тренування моделі ARIMA:

model = ARIMA(data, order=order)

model_fit = model.fit()

3. Прогнозування**:**

predictions = model_fit.forecast(steps=forecast_days)

predicted_index = pd.date_range(start=data.index[-1] + pd.Timedelta(days=1), periods=forecast_days, freq='D')

predictions = pd.Series(predictions, index=predicted_index)

4. Обчислення MSE (середньоквадратична похибка):

if calculate_mse and len(data) >= forecast_days:

mse = mean_squared_error(data[-forecast_days:], predictions[:len(data[-forecast_days:])])

result['MSE'] = mse

5. Повернення результатів**.** Результати містять прогнозовані значення і, за необхідності, MSE.

```
def apply_sarima_model(data, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12), forecast_days=10, calculate_mse=True):
```

Рисунок 3.26 – Функція apply_sarima_model

- apply_sarima_model: Використовує модель SARIMA для аналізу сезонних даних. Параметри:
- data: Вхідний часовий ряд.
- order: Параметри ARIMA (p, d, q).
- seasonal_order: Сезонні параметри (P, D, Q, m).
- forecast_days: Кількість днів для прогнозування.
- calculate_mse: Чи слід обчислювати MSE.

Основні етапи функції:

1. Підготовка даних:

Аналогічно до функції apply_arima_model, встановлюється частота даних і заповнюються відсутні значення.

2. Тренування SARIMA моделі:

model = SARIMAX(data, order=order, seasonal_order=seasonal_order)

model_fit = model.fit(maxiter=1000, method='lbfgs')

3. Прогнозування:

Прогнозується кількість днів, вказана у параметрі forecast_days.

4. Обчислення MSE:

Якщо опція calculate_mse увімкнена і кількість даних достатня, обчислюється MSE для оцінки точності.

```
def apply_prophet_model(data, forecast_days=10):···
```

Рисунок 3. 27 – Функція apply_prophet_model

- apply_prophet_model: Використовує модель Prophet (розроблена Facebook) для прогнозування часових рядів.
- data: Вхідні дані (часовий ряд).
- forecast_days: Кількість днів для прогнозування.

Основні етапи функції:

1. Підготовка даних:

Данні підготовлюються у формат, який потребує Prophet, де стовпець дати називається "ds", а значення часового ряду – "y".

2. Тренування моделі:

model = Prophet()

model.fit(df)

3. Прогнозування:

Створюється майбутній DataFrame для прогнозу і отримуються прогнозовані значення.

```
def auto_arima_model(data, forecast_days=10):···
```

Рисунок 3.28 – Функція auto_arima_model

- auto_arima_model: Автоматично підбирає найкращі параметри для моделі ARIMA за допомогою pmdarima.
- data: Часовий ряд.
- forecast_days: Кількість днів для прогнозування.

Основні етапи функції:

1. Підготовка даних:

Конвертація індексу в DateTimeIndex, встановлення частоти і заповнення відсутніх значень.

Автоматичний підбір параметрів:

model = pm.auto_arima(data, seasonal=False, trace=True, error_action='ignore', suppress_warnings=True)

2. Прогнозування:

Прогноз на вказану кількість днів.

3. Обчислення MSE:

Якщо достатньо даних, обчислюється MSE для оцінки точності прогнозу.

```
# Приклад використання (опціонально)
if __name__ == "__main__":
    from data_collection import fetch_financial_data
    from data_processing import clean_data

    symbol = 'AAPL'
    data = fetch_financial_data(symbol)
    if data is not None:
        cleaned_data = clean_data(data)
        if cleaned_data is not None:
            # Застосування ARIMA моделі
            arima_results = apply_arima_model(cleaned_data['Close'])
            if arima_results:
                print(arima_results['predictions'].head())
        else:
            print("Дані після очищення порожні.")
    else:
        print("Не вдалося завантажити дані.")
```

Рисунок 3.29 – Приклад використання

Завантаження даних для AAPL (Apple Inc.), очищення і застосування моделі ARIMA для прогнозування.

Файл financial_data_analysis.py виконує важливу роль у забезпеченні можливості аналізу та прогнозування часових рядів у проєкті. У ньому реалізовано кілька моделей: ARIMA, SARIMA, Prophet та Auto ARIMA, що дає змогу обирати найкращу модель для прогнозування фінансових даних залежно від типу даних та наявності сезонності. Кожна функція ретельно обробляє дані, адаптує їх для обраної моделі та обчислює показник точності (MSE), якщо це можливо.

Набір інструментів у financial_data_analysis.py дозволяє користувачу швидко й ефективно аналізувати дані за допомогою різних підходів, що є важливим у фінансовому аналізі, де точність прогнозів може мати критичне значення. Завдяки логуванню та автоматизованій обробці помилок файл забезпечує прозорість та надійність роботи, що полегшує інтеграцію в комплексні проєкти з аналітикою часових рядів.

Dockerfile.

```
# Dockerfile
FROM python:3.9-slim
```

Рисунок 3.30 – Базовий образ

FROM python:3.9-slim**:** Використовується базовий образ Python 3.9 slim**,** який є легшою версією повного образу Python, з мінімальним набором інструментів та залежностей.

```
# Встановлюємо системні залежності
RUN apt-get update && \
    apt-get install -y build-essential libpython3-dev python3-dev && \
    apt-get install -y libgomp1 && \
    apt-get install -y git && \
    apt-get install -y tzdata && \
    apt-get clean
```

Рисунок 3.31 – Встановлення системних залежностей

- apt-get update: Оновлює список пакетів для системи.
- build-essential: Пакет із компіляторами та іншими інструментами, необхідними для збирання програм.
- libpython3-dev та python3-dev: Необхідні для компіляції Python-пакетів із залежностями на C/C++.
- libgomp1: Бібліотека для підтримки багатопотокової роботи OpenMP (потрібна для деяких моделей машинного навчання).
- git: Встановлюється для роботи з репозиторіями Git.
- tzdata: Пакет, який містить інформацію про часові зони.
- apt-get clean: Очищає кеш встановлених пакетів, щоб зменшити розмір контейнера.

```
# Встановлюємо часовий пояс
ENV TZ=Europe/Kyiv

# Налаштовуємо tzdata
RUN echo $TZ > /etc/timezone && \
    ln -fs /usr/share/zoneinfo/$TZ /etc/localtime && \
    dpkg-reconfigure -f noninteractive tzdata
```

Рисунок 3.32 – Налаштування часового поясу

- ENV TZ=Europe/Kyiv: Встановлюється змінна оточення для часового поясу на Київ (Europe/Kyiv).
- echo $TZ > /etc/timezone: Записує часовий пояс у файл /etc/timezone.
- ln -fs /usr/share/zoneinfo/$TZ /etc/localtime: Створює символьне посилання на правильну зону часу.
- dpkg-reconfigure -f noninteractive tzdata: Налаштовує часовий пояс без втручання користувача.

```
# Встановлюємо змінні оточення
ENV PYTHONUNBUFFERED=1
```

Рисунок 3.33 – Змінна оточення для Python

- PYTHONUNBUFFERED=1**:** Забезпечує негайне виведення логів у термінал без кешування (важливо для серверних додатків).

```
# Встановлюємо робочу директорію всередині контейнера
WORKDIR /app
```

Рисунок 3.34 – Встановлення робочої директорії

- WORKDIR /app**:** Встановлює робочу директорію всередині контейнера як /app. Всі наступні команди та операції будуть виконуватись у цій директорії.

```
# Копіюємо файл requirements.txt
COPY requirements.txt /app/

# Встановлюємо необхідні пакети
RUN pip install --no-cache-dir -r requirements.txt
```

Рисунок 3.35 – Копіювання та встановлення залежностей

- COPY requirements.txt /app/: Копіює файл requirements.txt із локальної машини до директорії /app у контейнері.
- RUN pip install --no-cache-dir -r requirements.txt: Встановлює всі залежності, вказані у файлі requirements.txt, без використання кешу, щоб зменшити розмір образу.

```
# Копіюємо решту коду додатку
COPY . /app/
```

Рисунок 3.36 – Копіювання додатку

- COPY . /app/**:** Копіює весь код додатку з локальної директорії до робочої директорії /app у контейнері.

```
# Відкриваємо порт для Dash
EXPOSE 8000
```

Рисунок 3.37 – Відкриття порту для Dash

- EXPOSE 8000: Відкриває порт 8000 для доступу до Dash-додатку, який буде працювати всередині контейнера.

```
# Команда для запуску додатку
CMD ["python", "data_analysis.py"]
```

Рисунок 3.38 – Команда для запуску додатку

- CMD ["python", "data_analysis.py"]**:** Вказує команду для запуску Python-додатку (файл data_analysis.py) при старті контейнера.

Цей Dockerfile створює Docker-контейнер для Python-додатку на базі образу Python 3.9. Він включає встановлення необхідних системних залежностей, налаштування часового поясу та встановлення Python-пакетів, що використовуються у вашому проєкті. Контейнер відкриває порт 8000 для доступу до Dash-додатку і запускає програму при старті контейнера.

3.2. Тестування моделей часових рядів ARIMA та SARIMA

Модель ARIMA.

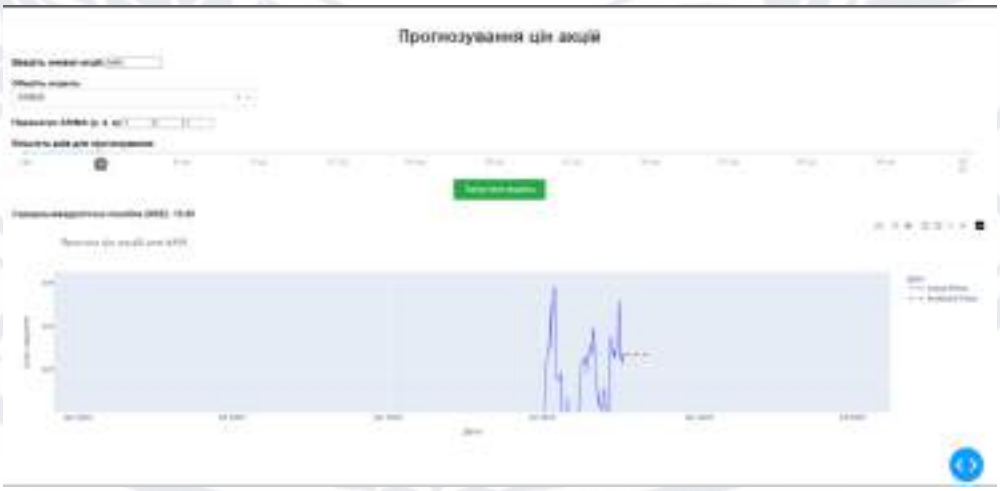

Рисунок 3.39 – модель ARIMA

Для покращення точності прогнозу, було обрано іншу конфігурацію моделі ARIMA для передбачення цін акцій компанії Apple Inc. (AAPL)**.** У цій конфігурації використовуються нові значення параметрів (p, d, q), які можуть впливати на точність прогнозу.

Параметри моделі ARIMA:

- p = 0: Відсутність лагів у моделі.
- d = 1: Використання першого порядку різницювання для усунення тренду.
- q = 1: Використання одного компонента рухомого середнього для зменшення впливу короткочасних коливань.

Кількість днів для прогнозування: Як і в попередньому тестуванні, обрано 30 днів. Така кількість днів дозволяє краще зрозуміти короткострокові зміни та тенденції, що є корисним для фінансових ринків.

Процес підготовки даних: Процес підготовки залишився аналогічним до попередніх тестів:

- Дані завантажуються з кешу (AAPL_max.csv).
- Виконується очищення даних, додавання стовпця щоденної відсоткової зміни (daily_pct_change) та розрахунок ковзних середніх для 50 і 200 днів.

Оцінка моделі:

Значення середньоквадратичної похибки (MSE) для цієї конфігурації складає 19.74, що трохи нижче порівняно з попередньою конфігурацією (1, 1, 1). Це свідчить про незначне покращення точності моделі.

Висновок щодо тестування моделі ARIMA з параметрами (0, 1, 1)

На графіку прогнозування зображені фактичні ціни (Actual Prices) та прогнозовані значення (Predicted Prices) для 30-денного періоду. Червоний пунктир відображає прогнозовані значення. Хоча значення MSE трохи знизилося до 19.74, модель все ще демонструє схожий рівень точності, як і при використанні параметрів (1, 1, 1).

Таким чином, незначне зниження MSE свідчить про потенційне

покращення, але ця конфігурація параметрів не показала суттєвих змін у якості прогнозу. Для подальшого вдосконалення варто експериментувати з іншими параметрами, такими як p, або ж розглянути використання моделі SARIMA, якщо є підозра на сезонність у часовому ряді.

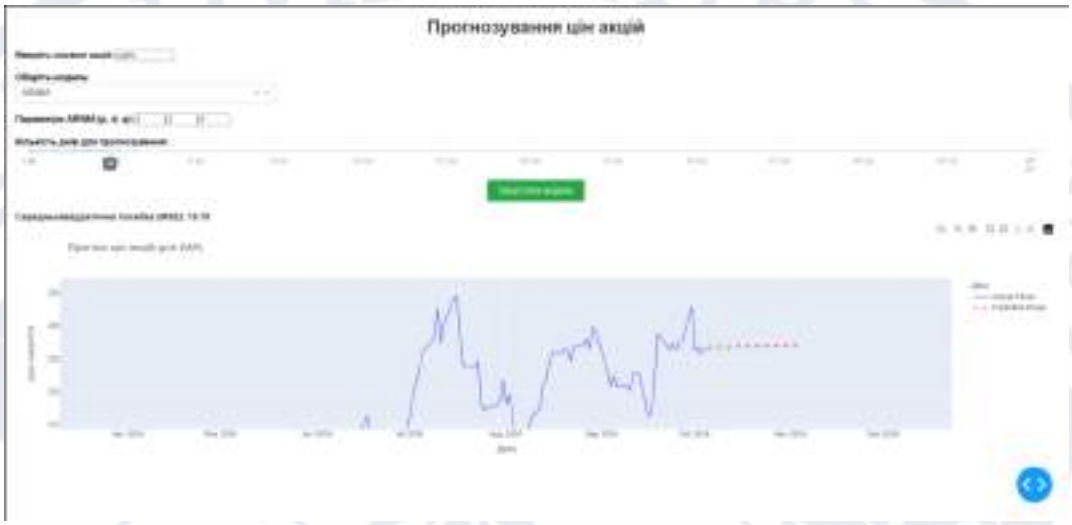

Рисунок 3.40 – модель ARIMA

Щоб оцінити ефективність моделі ARIMA, ми протестували ще одну комбінацію параметрів, яка могла б забезпечити кращу точність прогнозу для цін акцій компанії Apple Inc. (AAPL). Параметри (p, d, q) були налаштовані так:

Параметри моделі ARIMA:

- p = 2: Використання двох лагів у моделі.
- d = 2: Другий порядок різницювання для усунення тренду.
- q = 5: Використання п'яти компонентів рухомого середнього, що дозволяє враховувати складніші короткострокові коливання.

Кількість днів для прогнозування:

Прогнозування було збережено на 30 днів, що дозволяє проаналізувати потенційні зміни ціни на короткому проміжку.

Процес підготовки даних: Процес залишився незмінним і включає:

- Завантаження даних з кешу (AAPL_max.csv).
- Очищення даних з додаванням показника щоденної відсоткової

зміни (daily_pct_change) та ковзних середніх на 50 і 200 днів, що дозволяє краще зрозуміти загальні ринкові тенденції.

Оцінка моделі. Значення середньоквадратичної похибки (MSE) для цієї конфігурації складає 19.78, що дуже близьке до попередніх результатів з іншими параметрами, але трохи вище, ніж для параметрів (0, 1, 1).

Розглянемо тестування моделі ARIMA з параметрами (2, 2, 5). Графік прогнозування демонструє фактичні ціни (Actual Prices) та прогнозовані значення (Predicted Prices) на 30-денний період у майбутньому, де червоний пунктир позначає прогнозовані значення. У порівнянні з попередніми конфігураціями, поточне налаштування з параметрами (2, 2, 5) не показало значного покращення точності, оскільки значення MSE залишилося майже на тому ж рівні.

Отже, ці параметри не забезпечили значної різниці в результатах порівняно з простішими моделями. Це свідчить про те, що додаткове ускладнення моделі не завжди покращує точність прогнозу. Для подальшого вдосконалення варто розглянути інші моделі або перевірити сезонні фактори, можливо, через SARIMA.

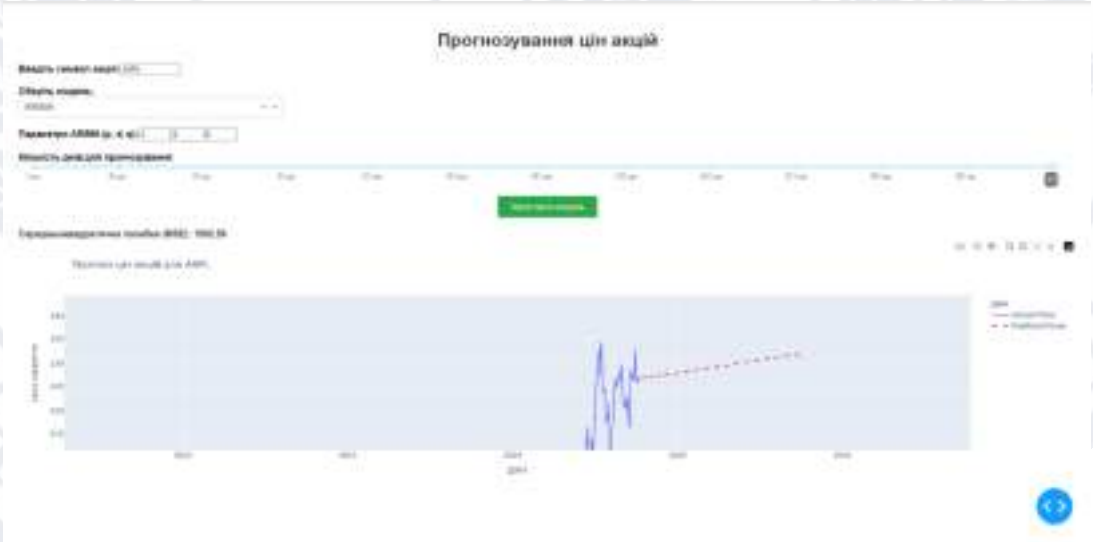

Рисунок 3.41 – модель ARIMA

Для подальшого аналізу моделі ARIMA було вирішено збільшити період прогнозування та перевірити її ефективність на довшому часовому проміжку.

Вибрані параметри (p, d, q) залишилися такими ж, як і в попередньому тесті, проте період прогнозування був встановлений на 361 день.

Параметри моделі ARIMA:

- p = 2: Використання двох лагів.
- d = 2: Другий порядок різницювання.
- q = 5: Використання п'яти компонентів рухомого середнього.

Кількість днів для прогнозування**:** обрано 361 день, що є суттєво довшим терміном, порівняно з попередніми тестами. Такий період дозволяє оцінити можливості моделі для довгострокового прогнозування.

Процес підготовки даних**:**

- Дані завантажено з кешу (AAPL_max.csv).
- Проведено очищення даних, додано стовпець щоденної відсоткової зміни (daily_pct_change) та розраховано ковзні середні для 50 і 200 днів, що покращує аналіз середньострокових і довгострокових трендів.

Оцінка моделі:

- Значення середньоквадратичної похибки (MSE) для цієї конфігурації склало 1902.56, що суттєво більше порівняно з попередніми короткостроковими прогнозами.
- Це значення MSE свідчить про те, що точність моделі зменшується при спробах довгострокового прогнозування, що можна побачити на графіку – червона пунктирна лінія відображає прогнозовані значення, які поступово відхиляються від фактичних даних.

Розглянемо тестування моделі ARIMA з параметрами (2, 2, 5) для 361-денної перспективи. Результати цього тесту демонструють, що модель ARIMA з параметрами (2, 2, 5) має обмежену точність для довгострокового прогнозування цін акцій. Значне збільшення MSE до 1902.56 порівняно з короткостроковими прогнозами підтверджує, що модель краще підходить для короткострокових передбачень, оскільки з часом її точність знижується.

Такий результат вказує на необхідність додаткової оптимізації або

вибору альтернативної моделі для довгострокових прогнозів, оскільки ARIMA, ймовірно, не здатна врахувати всі можливі зміни та коливання ринку на тривалому періоді.

Модель SARIMA.

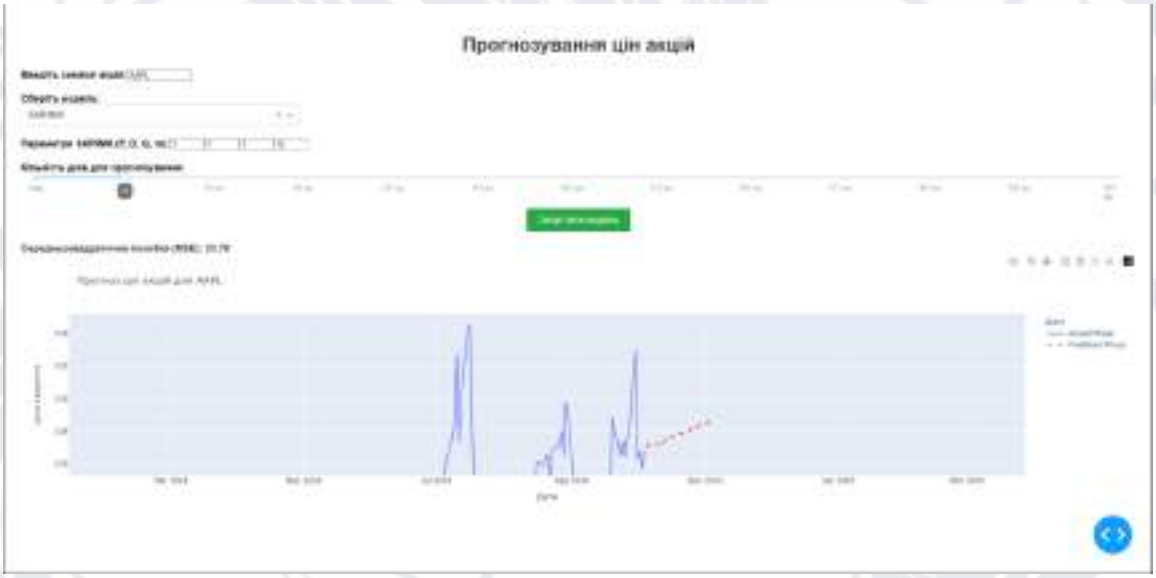

Рисунок 3.42 – модель SARIMA

В даному тестуванні для прогнозування цін акцій моделі було використано SARIMA (сезонна модель ARIMA), що дозволяє врахувати сезонні коливання в даних. Для тесту були обрані такі параметри:

Параметри SARIMA моделі**:**

- P = 1: Сезонний порядок авторегресії.
- D = 1: Сезонний порядок різницювання.
- Q = 1: Сезонний порядок ковзного середнього.
- m = 12: Сезонність у 12 місяців, що підходить для щомісячних або сезонних даних.

Кількість днів для прогнозування: встановлено на 31 день. Такий період дозволяє оцінити ефективність короткострокового прогнозування з урахуванням сезонних впливів.

Процес підготовки даних:

- Дані завантажено з кешу (AAPL_max.csv).
- Очищення даних включало видалення пропусків, додавання стовпця daily_pct_change, та розрахунок ковзних середніх на 50 та 200 днів. Це дозволяє моделі краще враховувати тренди на середньострокову перспективу.

Оцінка моделі:

- Значення середньоквадратичної похибки (MSE) склало 23.78, що є задовільним результатом для короткострокового прогнозування і свідчить про відносно невелике відхилення прогнозованих значень від фактичних.
- На графіку видно, що червона пунктирна лінія (прогнозовані значення) залишається близькою до фактичних даних, що демонструє хорошу точність моделі на обраному періоді.

Розглянемо тестування моделі SARIMA з параметрами (1, 1, 1, 12) для 31 дня. Тестування показало, що модель SARIMA з параметрами (1, 1, 1, 12) забезпечує прийнятну точність для короткострокового прогнозування, зокрема для періоду в 31 день. Значення MSE 23.78 є свідченням того, що модель SARIMA є більш стійкою та точною для прогнозування на короткому проміжку часу порівняно з базовою ARIMA. У майбутньому для покращення прогнозування на довші періоди можна розглянути налаштування інших параметрів сезонності або вибір альтернативної моделі.

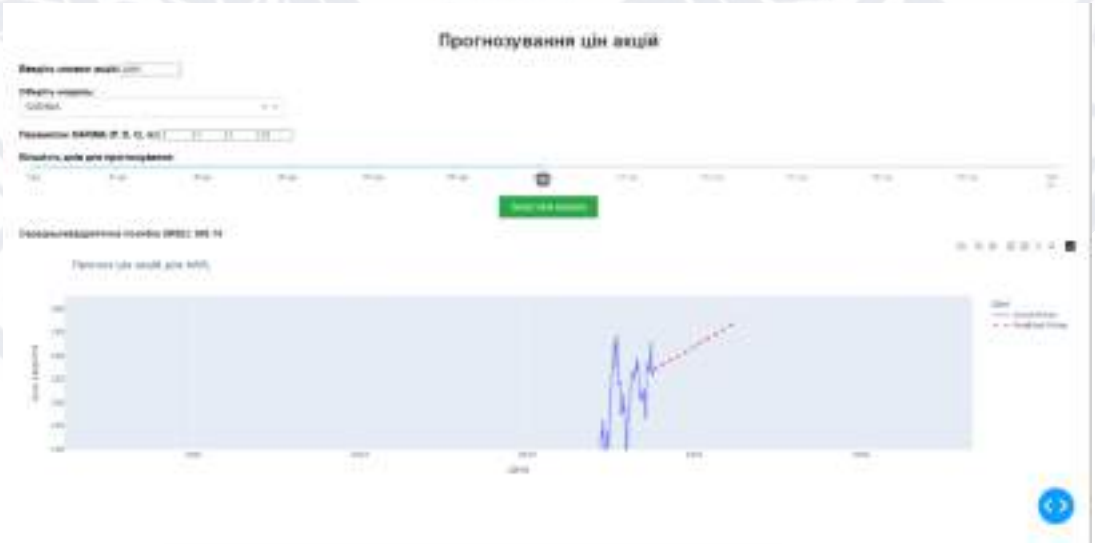

Рисунок 3.43 – модель SARIMA

У цьому тесті використано модель SARIMA для прогнозування цін акцій з такими параметрами:

Параметри SARIMA**:**

- P = 1: Сезонний порядок авторегресії.
- D = 1: Сезонний порядок різницювання.
- Q = 1: Сезонний порядок ковзного середнього.
- m = 12: Сезонність у 12 місяців, яка враховує річний цикл сезонних змін.

Кількість днів для прогнозування**: о**брано період у 181 день. Це дозволяє оцінити можливості моделі SARIMA в довгостроковому прогнозуванні з урахуванням сезонних змін.

Процес обробки даних:

- Дані було завантажено з кешу та очищено. Після очищення було додано нові фінансові індикатори, такі як daily_pct_change (відсоткові зміни) і ковзні середні на 50 і 200 днів. Ці показники надають більше інформації для побудови тренду і сезонності.

Оцінка моделі:

- Середньоквадратична похибка (MSE) склала 966.14, що свідчить про збільшення похибки у порівнянні з короткостроковими прогнозами. Це обумовлено тим, що SARIMA модель показує більшу похибку при довгострокових прогнозах, оскільки сезонні коливання можуть змінюватись з часом і модель не завжди здатна врахувати ці зміни на великому проміжку.
- На графіку видно, що прогнозована червона пунктирна лінія починає відхилятися від фактичних даних, що є свідченням зниження точності моделі при довгостроковому прогнозуванні.

Розглянемо тестування моделі SARIMA з параметрами (1, 1, 1, 12) для 181 дня. Тестування показало, що модель SARIMA з параметрами (1, 1, 1, 12) є ефективною для прогнозування на коротких проміжках часу. При прогнозі на 181 день значення MSE зросло до 966.14, що свідчить про можливі

проблеми з точністю для довгострокових прогнозів. Модель враховує сезонність, однак на тривалих інтервалах прогнозу вона може недооцінювати або переоцінювати коливання, що призводить до високої похибки. Можна розглянути додаткове налаштування параметрів або вибір іншої моделі для довгострокового прогнозування.

Рисунок 3.44 – модель SARIMA

У цьому тесті використано модель SARIMA для прогнозування цін акцій з такими параметрами:

Параметри SARIMA**:**

- P = 1: Сезонний порядок авторегресії.
- D = 1: Сезонний порядок різницювання.
- Q = 1: Сезонний порядок ковзного середнього.
- m = 12: Сезонність у 12 місяців, яка враховує річний цикл сезонних змін.

Кількість днів для прогнозування: 360 днів, що дозволяє оцінити поведінку моделі на річній дистанції.

Результати та оцінка моделі:

Середньоквадратична похибка (MSE): 2437.04.

- Значення MSE є досить великим, що свідчить про високу похибку в довгостроковому прогнозі. Це може бути зумовлено зміною трендів і неврахованими сезонними коливаннями на такому довгому періоді.
- На графіку можна побачити значне відхилення прогнозованих цін від реальних даних у другій половині періоду, що говорить про обмеження моделі SARIMA для точного довгострокового прогнозування.

Для прогнозу на 360 днів модель SARIMA з обраними параметрами продемонструвала обмежену точність, що підтверджується високим значенням MSE (2437.04). Незважаючи на врахування сезонності, модель погано справляється з довгостроковим прогнозуванням, що свідчить про необхідність додаткового налаштування або вибору іншої моделі для подібних завдань.

3.3 Аналіз та тестування моделей Prophet та Auto ARIMA

Модель Prophet.

Рисунок 3.45 – Модель Prophet

Кількість днів для прогнозування: 31 день, що дає місячний прогноз для оцінки короткострокових змін. Особливості**:**

- Модель автоматично деактивувала щоденну сезонність (daily seasonality) та враховувала тільки річну сезонність. Це може обмежити точність у випадку короткострокових прогнозів, де присутні динамічні

щоденні коливання.

Оцінка моделі: середньоквадратична похибка (MSE)**:** не розраховується для моделі Prophet, що обмежує оцінку точності в числовому форматі. На графіку видно, що прогнозовані значення (червона пунктирна лінія) продовжують тенденцію даних, однак рівень невизначеності залишається високим через відсутність щоденної сезонності.

Прогноз на 31 день за допомогою моделі Prophet демонструє адекватне продовження тренду, однак відсутність щоденної сезонності може знижувати точність у короткостроковій перспективі. Модель Prophet корисна для середньо- та довгострокових прогнозів, але для короткотермінових оцінок може вимагати додаткового налаштування або іншого підходу.

Рисунок 3.46 – Модель Prophet

Кількість днів для прогнозування: 181 день (близько шести місяців).

Особливості прогнозу. Модель Prophet автоматично вимкнула щоденну сезонність (daily seasonality), що означає, що прогноз орієнтований на довгострокові тренди та сезонні річні цикли. Щоденні коливання не враховані, що може вплинути на точність прогнозу в короткостроковій перспективі.

Результати та оцінка моделі:

- Середньоквадратична похибка (MSE): відсутня для обраної моделі, тому числова оцінка точності недоступна.

- Графік прогнозу демонструє стабільний тренд, але видно, що модель не захоплює дрібні коливання, які можуть мати місце протягом коротших періодів.

Прогноз моделі Prophet на 181 день відображає загальну тенденцію цін на акції Apple, однак він обмежений через відсутність короткотермінових коливань, що можуть бути значущими в цьому періоді. Ця модель є більш придатною для довгострокових прогнозів з урахуванням сезонних факторів, але для більш точної короткострокової оцінки може знадобитися інша модель або налаштування параметрів.

Рисунок 3.47 – Модель Prophet

Кількість днів для прогнозування. 361 день (близько одного року).

Особливості. Модель Prophet в цьому прогнозі також вимкнула щоденну сезонність (daily seasonality), що робить прогноз орієнтованим на річні тренди без врахування короткострокових коливань.

Результати та оцінка моделі:

- Середньоквадратична похибка (MSE): для моделі Prophet не доступна, тому числова оцінка точності відсутня.
- Графік прогнозу показує поступове зростання цін на акції Apple протягом року, однак через обмеження сезонності модель не захоплює короткотермінові коливання.

Цей прогноз на рік дозволяє оцінити загальний тренд зростання цін, проте модель Prophet з вимкненою щоденною сезонністю не здатна точно

відображати детальні зміни на коротких інтервалах. Модель підходить для довгострокових прогнозів, однак для більш точної оцінки короткотермінових змін можна спробувати включити щоденну сезонність або обрати іншу модель.

Модель Auto ARIMA.

Кількість днів для прогнозування**.** 31 день (близько одного місяця).

Особливості**:** Auto ARIMA автоматично обирає оптимальні параметри моделі ARIMA на основі критерію AIC (інформаційний критерій Акаіке), що допомагає знизити помилку прогнозу.

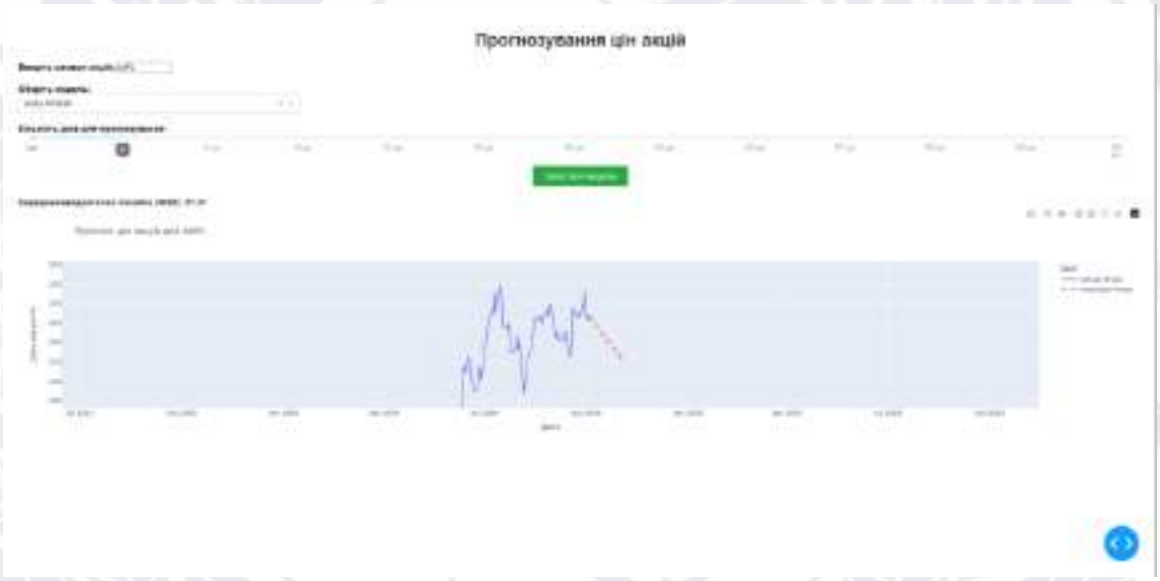

Рисунок 3.48 – Модель Auto ARIMA

Результати та оцінка моделі:

- Середньоквадратична похибка (MSE): 57.01, що вказує на досить високу точність для короткострокового прогнозу.
- Графік прогнозу показує легке зниження цін на акції Apple впродовж наступного місяця, з поступовим падінням, відображеним червоною пунктирною лінією.

Auto ARIMA підходить для короткострокових прогнозів, надаючи точне передбачення з відносно низькою середньоквадратичною похибкою (MSE).

Рисунок 3.49 – Модель Auto ARIMA

Кількість днів для прогнозування. 181 день (близько шести місяців).

Особливості. Auto ARIMA обирає оптимальні параметри на основі AIC, дозволяючи забезпечити точніший прогноз при довгострокових передбаченнях.

Результати та оцінка моделі:

- Середньоквадратична похибка (MSE): 1682.47, що вказує на значну похибку у порівнянні з короткостроковими прогнозами.
- Прогнозована лінія (червоний пунктир) показує значне зниження ціни акцій Apple у довгостроковій перспективі. Такий результат вказує на можливі обмеження моделі Auto ARIMA у довгострокових прогнозах.

Auto ARIMA може показувати значні відхилення у довгострокових передбаченнях, як видно з високої середньоквадратичної похибки (MSE).

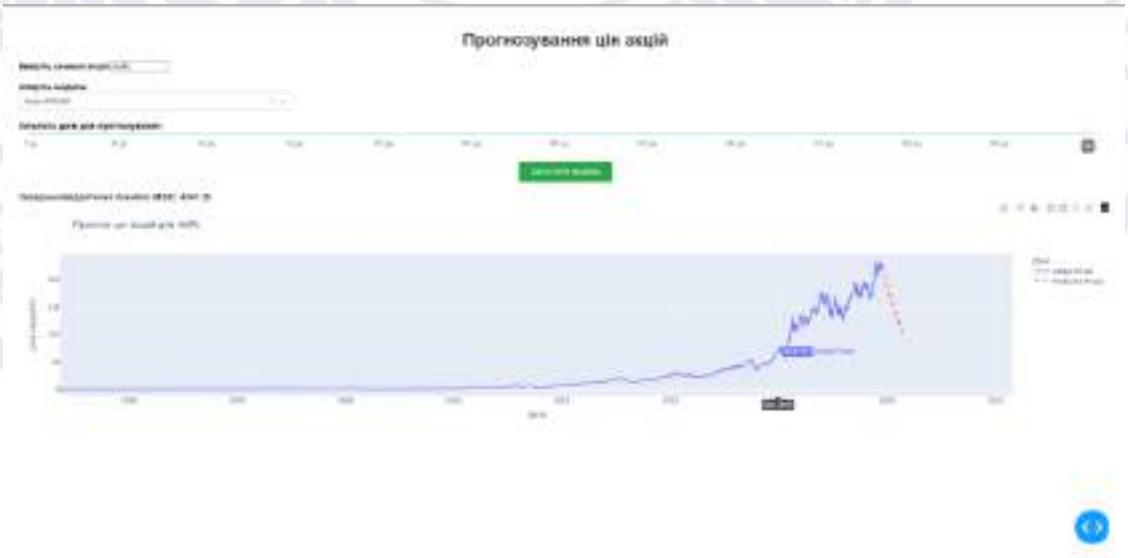

Рисунок 3.50 – Модель Auto ARIMA

Кількість днів для прогнозування: 361 день (майже рік).

Особливості. Auto ARIMA обирає оптимальні параметри моделі для забезпечення передбачень у довгостроковій перспективі, використовуючи AIC як критерій відбору.

Результати та оцінка моделі:

- Середньоквадратична похибка (MSE): 4241.25, що демонструє значну похибку для прогнозів на річну перспективу.
- Прогнозована лінія (червоний пунктир) вказує на можливе зниження цін акцій AAPL. Довгострокове передбачення показує тренд до падіння, але висока похибка свідчить про те, що цей результат може бути менш надійним.

Auto ARIMA добре підходить для короткострокових прогнозів, але з високою похибкою для довгострокових передбачень, що робить її менш точною для прогнозування на один рік вперед.

ВИСНОВКИ З РОЗДІЛУ 3

Даний розділ присвячений питанню практичної розробки інформаційної системи аналізу даних фінансових ринків. Наведено програмний код, на основі якого розроблено інформаційну систему для аналізу даних та прогнозування таких фінансових інструментів, як цін акцій за допомогою моделей ARIMA, SARIMA, Prophet та Auto ARIMA. Програмно система функціонує у вигляді вебдодатка, інтерфейс якого дозволяє обирати вид моделі, налаштовувати параметри і відображати графік з реальними і прогнозованими даними. Реалізовано функціонал для завантаження фінансових даних із Yahoo Finance із можливістю кешування даних для повторного використання та зменшення часу завантаження в майбутньому.

Програмно забезпечено підготовку фінансових даних, включаючи їх очищення та нормалізацію, що є важливими етапами для подальшого аналізу та використання в моделях машинного навчання.

Реалізовано моделі типу: ARIMA**,** SARIMA**,** Prophet та Auto ARIMA. При обробці даних існує їх адаптація до обраної моделі та обчислюється метрика точності (MSE). Завдяки логуванню та автоматизованій обробці помилок забезпечується прозорість та надійність роботи, що полегшує інтеграцію в комплексні проєкти з аналітикою часових рядів.

ВИСНОВКИ

В процесі виконання магістерської роботи була розроблена інформаційна система для аналізу даних фінансових ринків. В основі роботи з даними в даній системі покладено підходи, методи та моделі аналізу часових рядів з використанням технологій машинного навчання. За результатами дослідження можна зробити наступні висновки:

1. На основі аналізу теоретичних засад аналізу даних фінансових ринків визначено, що технології аналізу даних займають провідне місце в отриманні якісних та точних характеристик фінансових інструментів і надають можливість зпрогнозувати динаміку руху потоків фінансових ресурсів в короткостроковій та середньостроковій перспективі. Визначено, що одним із підходів для проведення аналізу даних на фінансових ринків є застосування технологій аналізу часових рядів. Виділено основні класифікаційні ознаки часових рядів і методи їх моделювання.

2. Встановлено, що основними чинниками, які впливають на інформаційні потоки про процеси руху фінансових ресурсів є компоненти часових рядів, такі як-от: тренд, сезонність, циклічнісь та шум.

3. Аналіз теоретичних основ роботи з даними на фінансових ринках дозволив виділити основні типи математичних моделей, які можливо використовувати в результаті навчання для подальшої розробки прогнозних рішень. Такими моделями є: ARIMA, SARIMA, Prophet та Auto ARIMA. Дослідження із використанням цих моделей побудоване не тільки на визначені їх параметрів, але й на застосуванні відповідних метрик вимірювання якості та точності.

4. На основі теоретичних засад було розроблено інформаційну систему для аналізу часових рядів. Система включає в себе модулі для збору, обробки та аналізу даних, яка була протестована на основі реальних фінансових даних, що підтвердило її ефективність і адаптивність до змінних умов.

5. Оцінку моделей проведено в процесі тестування на основі

сформованих тестових вибірок даних. Визначено, що кожна з моделей має свої переваги залежно від типу даних та завдання. Зокрема, модель ARIMA добре справляється з короткостроковими прогнозами. Модель SARIMA ефективна для даних із вираженою сезонністю. Модель Prophet дозволяє враховувати складні тренди та зовнішні фактори. Модель Auto ARIMA спрощує вибір параметрів моделі.

6. Для підвищення точності прогнозів доцільно використовувати гібридні моделі, інтегрувати екзогенні змінні, такі як макроекономічні індикатори чи дані з новин, а також застосовувати методи автоматизованого підбору параметрів.

7. Програмна реалізація інформаційної системи впроваджена у розробленому вебдодатку, призначеного для прогнозування цін акцій за допомогою моделей ARIMA, SARIMA, Prophet та Auto ARIMA. Інтерфейс дозволяє вибирати модель, налаштовувати параметри і відображати графік з реальними і прогнозованими даними. Реалізований функціонал для завантаження фінансових даних із Yahoo Finance із можливістю кешування даних для повторного використання та зменшення часу завантаження в майбутньому.

8. Використаний набір інструментів у financial_data_analysis.py дозволяє користувачу швидко й ефективно аналізувати дані за допомогою різних підходів, що є важливим у фінансовому аналізі, де точність прогнозів може мати критичне значення. Завдяки логуванню та автоматизованій обробці помилок файл забезпечує прозорість та надійність роботи, що полегшує інтеграцію в комплексні проєкти з аналітикою часових рядів

9. Для роботи з даними передбачений Dockerfile, який створює Docker-контейнер для Python-додатку на базі образу Python 3.9. Він включає встановлення необхідних системних залежностей, налаштування часового поясу та встановлення Python-пакетів, що використовуються у вашому проєкті. Контейнер відкриває порт 8000 для доступу до Dash-додатку і запускає програму при старті контейнера.

10. Результати тестового прогону моделей показали, що для прогнозу на 360 днів модель SARIMA з обраними параметрами продемонструвала обмежену точність, що підтверджується високим значенням MSE (2437.04). Незважаючи на врахування сезонності, модель погано справляється з довгостроковим прогнозуванням, що свідчить про необхідність додаткового налаштування або вибору іншої моделі для подібних завдань.

11. Прогноз на 31 день за допомогою моделі Prophet демонструє адекватне продовження тренду, однак відсутність щоденної сезонності може знижувати точність у короткостроковій перспективі. Модель Prophet корисна для середньо- та довгострокових прогнозів, але для короткотермінових оцінок може вимагати додаткового налаштування або іншого підходу. Прогноз моделі Prophet на 181 день відображає загальну тенденцію цін і є більш придатною для довгострокових прогнозів з урахуванням сезонних факторів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Ткачук В.М. Алгоритми і структура даних. Івано-Франківськ: Видавництво Прикарпатського національного університету імені Василя Стефаника, 2016. 286 с.

2. Волонтир Л.О., Зелінська О.В., Потапова Н.А., Чіков І.А. Чисельні методи. Навчальний посібник. Вінниця: ВНАУ, 2020. 322 с.

3. Потапова Н.А., Денисюк В.О., Крохмалюк В.В. Парсинг як метод обробки даних при оцінці споживчої цінності товарів. Наука і техніка сьогодні, 2023. №13(27). С. 819-827.

4. Потапова Н.А., Волонтир Л.О., Зелінська О.В. Математичне та комп'ютерне моделювання функціонування логістичних процесів та систем. Вісник Хмельницького національного університету. Технічні науки, 2022. № 2. С. 73-80. URL: http://journals.khnu.km.ua/vestnik/?cat=65.

5. Нікітенко К.С., Осадчий А.А. Упровадження хмарних технологій у діяльність сучасних підприємств. Підприємництво і торгівля, 2020. № 27. С. 53-57. URL: http://nbuv.gov.ua/UJRN/Torg_2020_27_11.

6. Палійчук О.В. Прогнозування часових рядів: методологія та застосування. Київ: НАН України, 2019. 310 с.

7. Теслюк В.М. Інтелектуальний аналіз даних у прогнозуванні економічних показників. Львів: Львівська політехніка, 2018. 240 с.

8. Гончарук В.О. Часові ряди: основи аналізу та прогнозування. Харків: ХНЕУ ім. С. Кузнеця, 2020. 280 с.

9. Ляшенко В.І., Руденко І.В. Сезонні коливання в економічних даних та їхній вплив на прогнозування. Економічний простір, 2022. № 174. С. 112-120.

10. Семенюк А.І. Методи стаціонаризації часових рядів у фінансовому прогнозуванні. Київ: КНЕУ, 2018. 198 с.

11. Офіційний сайт Верховної Ради України. Закон України «Про хмарні послуги». URL: https://zakon.rada.gov.ua/laws/show/2075-20#Text.

12. Іваненко О.С., Петренко А.В. Прогнозування економічних показників із використанням ARIMA-моделей. Фінанси України, 2021. № 7. С. 15-22.

13. Романюк М.В., Коваленко П.М. Моделі часових рядів у задачах прогнозування. Статистика України, 2020. № 2. С. 45-52.

14. Богданович О.О. Інтегровані моделі прогнозування для економічного аналізу. Київ: Наукова думка, 2017. 290 с.

15. Бойко Т.М. Використання нейронних мереж у прогнозуванні часових рядів. Комп'ютерні системи і мережі, 2022. № 1. С. 78-85.

16. Гуменюк Л.С. Сезонні компоненти в моделюванні економічних процесів. Одеса: ОНУ ім. І.І. Мечникова, 2020. 210 с.

17. Кравець І.М. Прогнозування продажів із використанням SARIMA. Економічний аналіз, 2021. № 3. С. 34-42.

18. Литвиненко А.П. Моделювання часових рядів для управління запасами. Харків: НАН України, 2019. 230 с.

19. Сидоренко В.П., Трофименко О.В. Аналіз випадкових процесів у часових рядах. Прикладна статистика, 2020. № 5. С. 89-96.

20. Андрійчук О.П. Сучасні підходи до прогнозування з використанням машинного навчання. Львів: Львівський університет, 2022. 265 с.

21. Лазаренко О.М. Економетричний аналіз часових рядів. Дніпро: ДНУ, 2020. 215 с.

22. Малий І.С. Основи економетрії. Київ: КНЕУ, 2019. 350 с.

23. Мороз О.В. Детерміновані моделі в аналізі часових рядів. Харків: ХНЕУ, 2018. 240 с.

24. Шевчук С.В. Прогнозування економічних циклів. Львів: Український католицький університет, 2021. 200 с.

25. Тимошенко І.М. Часові ряди: аналітичні та прогнозні підходи. Київ: ІЕП НАН України, 2019. 190 с.

26. Кушніренко А.В. Моделі коригування помилки для економічного прогнозування. Одеса: ОНУ ім. І.І. Мечникова, 2021. 215 с.

27. Деркач О.В. Моделі машинного навчання в аналізі часових рядів. Київ: КПІ ім. Ігоря Сікорського, 2020. 230 с.

28. Олійник В.М. Методи прогнозування часових рядів у фінансовій аналітиці. Харків: НАН України, 2018. 200 с.

29. Захарченко В.В. Гібридні методи прогнозування. Київ: Інститут кібернетики НАН України, 2022. 250 с.

30. Яковенко О.С. Сезонність у часових рядах: економічні аспекти. Одеса: ОНУ ім. І.І. Мечникова, 2020. 180 с.

31. Литвин С.В. Автоматизація аналізу часових рядів. Київ: КПІ ім. Ігоря Сікорського, 2021. 275 с.

32. Марчук О.М. Прогнозування економічних процесів за допомогою нейронних мереж. Львів: ЛНУ ім. Івана Франка, 2020. 220 с.

33. Савчук В.А. Економічна динаміка та аналіз часових рядів. Харків: ХНЕУ, 2021. 260 с.

34. Гришко П.В. Прогнозування фінансових показників у бізнес-аналітиці. Київ: ІЕП НАН України, 2022. 210 с.

35. Демченко І.О. Моделювання часових рядів у маркетингових дослідженнях. Одеса: ОНУ ім. І.І. Мечникова, 2021. 200 с.

36. Іванов С.В. Прогнозування продажів із використанням машинного навчання. Київ: КПІ ім. Ігоря Сікорського, 2020. 230 с.

37. Поліщук Н.О. Прогнозування валютних курсів із використанням ARIMA. Львів: ЛНУ ім. Івана Франка, 2021. 195 с.

38. Кириченко Л.О. Моделювання сезонних процесів у часових рядах. Дніпро: ДНУ, 2022. 210 с.

39. Гончаренко Т.М. Моделі часових рядів у фінансових ринках. Київ: Наукова думка, 2018. 300 с.

40. Коваленко О.В. Економетричні методи прогнозування. Харків: ХНЕУ, 2020. 220 с.

ДОДАТКИ

Додаток А

Лістинг коду data_analysis.py

```
# data_analysis.py
from dash import dcc, html
from dash.dependencies import Input, Output, State
import dash
import plotly.graph_objs as go
from data_collection import fetch_financial_data
from data_processing import clean_data
from  financial_data_analysis  import  apply_arima_model,  apply_sarima_model,
apply_prophet_model, auto_arima_model
import logging

# Ініціалізація Dash додатку з suppress_callback_exceptions=True
app = dash.Dash(__name__, suppress_callback_exceptions=True)

# Налаштування логування
logging.basicConfig(level=logging.INFO)

# Основний макет
app.layout  =  html.Div(style={'padding':  '20px',  'font-family':  'Arial,  sans-
serif'}, children=[
    html.H1("Прогнозування цін акцій", style={'text-align': 'center', 'color':
'#333'}),

    html.Div([
        html.Label("Введіть символ акцій:", style={'font-weight': 'bold'}),
        dcc.Input(id='stock-symbol', type='text', value='AAPL', style={'width':
'100px'}),
    ], style={'margin-bottom': '20px'}),

    html.Div([
        html.Label("Оберіть модель:", style={'font-weight': 'bold'}),
        dcc.Dropdown(
            id='model-dropdown',
            options=[
                {'label': 'ARIMA', 'value': 'ARIMA'},
                {'label': 'SARIMA', 'value': 'SARIMA'},
                {'label': 'Prophet', 'value': 'Prophet'},
                {'label': 'Auto ARIMA', 'value': 'AutoARIMA'}
            ],
            value='ARIMA',
            style={'width': '50%'}
        ),
    ], style={'margin-bottom': '20px'}),

    html.Div([
        html.Div([
```

```
            html.Label("Параметри ARIMA (p, d, q):", style={'font-weight':
'bold'}),
             dcc.Input(id='arima-p', type='number', value=1, min=0, max=5,
style={'width': '50px'}),
             dcc.Input(id='arima-d', type='number', value=1, min=0, max=2,
style={'width': '50px'}),
             dcc.Input(id='arima-q', type='number', value=1, min=0, max=5,
style={'width': '50px'}),
        ], id='arima-params', style={'display': 'block'}),

        html.Div([
            html.Label("Параметри SARIMA (P, D, Q, m):", style={'font-weight':
'bold'}),
             dcc.Input(id='sarima-p', type='number', value=1, min=0, max=5,
style={'width': '50px'}),
             dcc.Input(id='sarima-d', type='number', value=1, min=0, max=2,
style={'width': '50px'}),
             dcc.Input(id='sarima-q', type='number', value=1, min=0, max=5,
style={'width': '50px'}),
            dcc.Input(id='sarima-m', type='number', value=12, min=1, max=12,
style={'width': '50px'}),
        ], id='sarima-params', style={'display': 'none'}),

        # Додаткові параметри моделей можна додати аналогічно
    ], id='model-params', style={'margin-bottom': '20px'}),

    html.Div([
         html.Label("Кількість днів для прогнозування:", style={'font-weight':
'bold'}),
        dcc.Slider(
            id='forecast-slider',
            min=1,
            max=365,
            step=1,
            value=30,
            marks={i: f'{i} дн.' for i in range(1, 366, 30)},
            tooltip={"placement": "bottom", "always_visible": True},
        ),
    ], style={'margin-bottom': '20px'}),

    html.Div([
        html.Button('Запустити модель', id='run-model', n_clicks=0,
                    style={'background-color': '#28a745', 'color': 'white',
'border': 'none',
                           'padding': '10px 20px', 'font-size': '16px', 'cursor':
'pointer'}),
    ], style={'text-align': 'center', 'margin-bottom': '20px'}),

      html.Div(id='mse-output',  style={'font-weight':  'bold',  'margin-top':
'20px', 'color': '#333'}),
```

```
    dcc.Graph(id='price-prediction-graph', config={'displayModeBar': True}),
])

# Контроль видимості параметрів моделі
@app.callback(
    [Output('arima-params', 'style'),
     Output('sarima-params', 'style')],
    Input('model-dropdown', 'value')
)
def update_params_visibility(selected_model):
    if selected_model == 'ARIMA':
        return {'display': 'block'}, {'display': 'none'}
    elif selected_model == 'SARIMA':
        return {'display': 'none'}, {'display': 'block'}
    else:
        return {'display': 'none'}, {'display': 'none'}

# Оновлення графіка та обробка натискання кнопки
@app.callback(
         [Output('price-prediction-graph',    'figure'),    Output('mse-output',
'children')],
    [Input('model-dropdown', 'value'),
     Input('forecast-slider', 'value'),
     Input('run-model', 'n_clicks'),
     Input('stock-symbol', 'value')],
    [State('arima-p', 'value'),
     State('arima-d', 'value'),
     State('arima-q', 'value'),
     State('sarima-p', 'value'),
     State('sarima-d', 'value'),
     State('sarima-q', 'value'),
     State('sarima-m', 'value')],
    prevent_initial_call=True
)
def update_graph(selected_model, forecast_days, n_clicks, stock_symbol, arima_p,
arima_d, arima_q, sarima_p, sarima_d, sarima_q, sarima_m):
    if n_clicks == 0:
        raise dash.exceptions.PreventUpdate

    try:
        # Завантаження та обробка даних для введеного символу
        data = fetch_financial_data(stock_symbol)
        if data is None or data.empty:
           return go.Figure(), f"Помилка: Не вдалося завантажити дані для символу
{stock_symbol}."

        cleaned_data = clean_data(data)
        if cleaned_data is None or cleaned_data.empty:
```

```
            return go.Figure(), "Помилка: Дані недостатні для аналізу після
очищення."

        if selected_model == 'ARIMA':
            if None in [arima_p, arima_d, arima_q]:
                raise dash.exceptions.PreventUpdate
             results = apply_arima_model(cleaned_data['Close'], forecast_days,
(arima_p, arima_d, arima_q))
        elif selected_model == 'SARIMA':
            if None in [sarima_p, sarima_d, sarima_q, sarima_m]:
                raise dash.exceptions.PreventUpdate
            results = apply_sarima_model(
                cleaned_data['Close'],
                order=(sarima_p, sarima_d, sarima_q),
                seasonal_order=(sarima_p, sarima_d, sarima_q, sarima_m),
                forecast_days=forecast_days
            )
        elif selected_model == 'Prophet':
            results = apply_prophet_model(cleaned_data['Close'], forecast_days)
        elif selected_model == 'AutoARIMA':
            results = auto_arima_model(cleaned_data['Close'], forecast_days)
        else:
            return go.Figure(), "Помилка: Невідома модель."

        if results is None:
            return go.Figure(), "Помилка: не вдалося виконати прогнозування."

        predicted_data = results['predictions']
        mse_value = results.get('MSE', None)

        # Створюємо фігуру
        fig = go.Figure()
         fig.add_trace(go.Scatter(x=cleaned_data.index, y=cleaned_data['Close'],
mode='lines', name='Actual Prices'))
             fig.add_trace(go.Scatter(x=predicted_data.index,  y=predicted_data,
mode='lines', name='Predicted Prices', line=dict(color='red', dash='dash')))

        fig.update_layout(
            title=f'Прогноз цін акцій для {stock_symbol}',
            xaxis_title='Дата',
            yaxis_title='Ціна закриття',
            legend_title='Дані',
            hovermode='x'
        )

        if mse_value is not None:
            mse_output = f"Середньоквадратична похибка (MSE): {mse_value:.2f}"
        else:
            mse_output = "MSE недоступна для обраної моделі."
```

```
            return fig, mse_output

        except Exception as e:
            logging.error(f"Error occurred: {e}")
            return go.Figure(), f"Помилка: Не вдалося виконати прогнозування. Деталі:
{str(e)}"

if __name__ == '__main__':
    app.run_server(debug=True, host='0.0.0.0', port=8000)
```

Додаток Б

Лістинг коду data_collection.py

```
# data_collection.py
import yfinance as yf
import pandas as pd
import os
import logging

# Налаштування логування
logging.basicConfig(level=logging.INFO)

def fetch_financial_data(symbol, period="max", cache=True):
       logging.info(f"Завантаження даних для символу {symbol} за період {period}...")

    cache_file = f"{symbol}_{period}.csv"
    try:
        if cache and os.path.exists(cache_file):
             data = pd.read_csv(cache_file, index_col='Date', parse_dates=True, infer_datetime_format=True)
            data.index = pd.to_datetime(data.index, utc=True).tz_convert(None)
            logging.info(f"Дані завантажено з кешу {cache_file}.")
        else:
            # Отримання об'єкта Ticker
            stock = yf.Ticker(symbol)

            # Завантаження історичних даних
            data = stock.history(period=period)

            if data.empty:
                logging.warning(f"Дані для символу {symbol} не знайдено або дані порожні.")
                 return None

            # Видаляємо часову зону з індексу
            data.index = data.index.tz_convert(None)

            # Кешування даних
            if cache:
                data.to_csv(cache_file)
                logging.info(f"Дані кешовано у {cache_file}")

            logging.info(f"Успішне завантаження даних для {symbol}.")

        return data

    except Exception as e:
```

```
            logging.error(f"Помилка при завантаженні даних для символу {symbol}:
{e}")
        return None

# Приклад використання
if __name__ == "__main__":
    symbol = 'AAPL'  # Приклад: Apple Inc.
    data = fetch_financial_data(symbol, period="1y", cache=True)

    if data is not None:
        print(data.head())  # Вивести перші рядки даних
```

Додаток В

Лістинг коду data_processing.py

```
# data_processing.py
from sklearn.preprocessing import StandardScaler
import pandas as pd
import logging

# Налаштування логування
logging.basicConfig(level=logging.INFO)

def clean_data(data, ma_periods=(50, 200)):
    """
    Очищення та підготовка даних для аналізу. Видалення null-значень,
    додавання нових показників (ковзні середні та відсоткові зміни).

    Args:
    data (DataFrame): Дані для очищення.
    ma_periods (tuple): Періоди для обчислення ковзних середніх (50, 200 днів за
замовчуванням).

    Returns:
    DataFrame: Очищені дані з доданими показниками.
    """
    try:
        # Перевірка наявності стовпця 'Close' у даних
        if 'Close' not in data.columns:
            raise ValueError("Стовпець 'Close' відсутній у даних.")

        logging.info("Початок очищення даних...")

        # Видаляємо будь-які рядки з відсутніми даними
        cleaned_data = data.dropna()

        # Додаємо нові фінансові індикатори: відсоткова зміна та ковзні середні
        cleaned_data['daily_pct_change'] = cleaned_data['Close'].pct_change() *
100
        logging.info("Додано стовпець 'daily_pct_change'.")

        # Додаємо ковзні середні з налаштуванням періодів
                                cleaned_data[f'{ma_periods[0]}_day_MA']       =
cleaned_data['Close'].rolling(window=ma_periods[0]).mean()
                                cleaned_data[f'{ma_periods[1]}_day_MA']       =
cleaned_data['Close'].rolling(window=ma_periods[1]).mean()
             logging.info(f"Додано  ковзні  середні  за  {ma_periods[0]}  та
{ma_periods[1]} днів.")

        # Видаляємо рядки з NaN після обчислення ковзних середніх
```

```
        cleaned_data = cleaned_data.dropna()
        logging.info("Очищені дані без NaN повернено.")

        return cleaned_data

    except Exception as e:
        logging.error(f"Помилка під час очищення даних: {e}")
        return None

def normalize_data(data, exclude_columns=None):
    """
    Нормалізація даних для того, щоб усі ознаки мали однаковий масштаб.

    Args:
    data (DataFrame): Дані для нормалізації.
     exclude_columns (list): Список колонок, які слід виключити з нормалізації
(за замовчуванням None).

    Returns:
    DataFrame: Нормалізовані дані.
    """
    try:
        logging.info("Початок нормалізації даних...")

        if exclude_columns:
            data_to_normalize = data.drop(columns=exclude_columns)
           logging.info(f"Колонки {exclude_columns} виключено з нормалізації.")
        else:
            data_to_normalize = data

        # Використовуємо StandardScaler для нормалізації даних
        scaler = StandardScaler()
        scaled_data = scaler.fit_transform(data_to_normalize)

          # Повертаємо нормалізовані дані як DataFrame з тими ж індексами та
колонками
             normalized_data   =   pd.DataFrame(scaled_data,   index=data.index,
columns=data_to_normalize.columns)

        # Додаємо колонки, які не нормалізували, якщо були виключені
        if exclude_columns:
           normalized_data = pd.concat([normalized_data, data[exclude_columns]],
axis=1)

        logging.info("Нормалізація даних завершена.")
        return normalized_data

    except Exception as e:
        logging.error(f"Помилка під час нормалізації даних: {e}")
        return None
```

```
# Приклад використання
if __name__ == "__main__":
    from data_collection import fetch_financial_data
    data = fetch_financial_data('AAPL')
    cleaned_data = clean_data(data)
    normalized_data = normalize_data(cleaned_data, exclude_columns=['Volume'])
    print(normalized_data.head())
```

Додаток Г

Лістинг коду financial_data_analysis.py

```
# financial_data_analysis.py
import pandas as pd
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error
import logging
import pmdarima as pm

# Налаштування логування
logging.basicConfig(level=logging.INFO)

def apply_arima_model(data, forecast_days=10, order=(1, 1, 1), calculate_mse=True):
    """
    Застосування ARIMA моделі з обраними параметрами.
'
    Args:
    data (pd.Series): Часовий ряд (наприклад, ціни закриття).
    forecast_days (int): Кількість днів для прогнозування.
    order (tuple): Параметри (p, d, q) для ARIMA моделі.
    calculate_mse (bool): Чи слід обчислювати MSE (за замовчуванням True).

    Returns:
    dict: Результати ARIMA з прогнозами та MSE (якщо обрано).
    """
    try:
        data = data.copy()

        # Конвертуємо індекс у DatetimeIndex з UTC, потім видаляємо часову зону
        data.index = pd.to_datetime(data.index, utc=True).tz_convert(None)

        # Встановлюємо частоту на щоденну
        data = data.asfreq('D')

        # Заповнюємо пропущені значення
        data = data.fillna(method='ffill')

        logging.info(f"Тренуємо ARIMA модель з параметрами: {order}")

        # Створюємо і тренуємо модель ARIMA
        model = ARIMA(data, order=order)
        model_fit = model.fit()

        # Прогноз на вказану кількість днів
```

```
        predictions = model_fit.forecast(steps=forecast_days)

        # Встановлюємо індекс для прогнозованих даних
                predicted_index    =    pd.date_range(start=data.index[-1]    +
pd.Timedelta(days=1), periods=forecast_days, freq='D')
        predictions = pd.Series(predictions, index=predicted_index)

        result = {'predictions': predictions}

        # Обчислення середньоквадратичної похибки (MSE), якщо можливо
        if calculate_mse and len(data) >= forecast_days:
                        mse    =    mean_squared_error(data[-forecast_days:],
predictions[:len(data[-forecast_days:])])
            result['MSE'] = mse
            logging.info(f"MSE для ARIMA: {mse}")

        return result

    except Exception as e:
        logging.error(f"Помилка при застосуванні ARIMA моделі: {e}")
        return None

def apply_sarima_model(data, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12),
forecast_days=10, calculate_mse=True):
    """
    Застосування SARIMA моделі з обраними параметрами.

    Args:
    data (pd.Series): Часовий ряд (наприклад, ціни закриття).
    order (tuple): Параметри (p, d, q) для ARIMA моделі.
    seasonal_order (tuple): Сезонні параметри (P, D, Q, m) для SARIMA моделі.
    forecast_days (int): Кількість днів для прогнозування.
    calculate_mse (bool): Чи слід обчислювати MSE (за замовчуванням True).

    Returns:
    dict: Результати SARIMA з прогнозами та MSE (якщо обрано).
    """
    try:
        data = data.copy()

        # Конвертуємо індекс у DatetimeIndex з UTC, потім видаляємо часову зону
        data.index = pd.to_datetime(data.index, utc=True).tz_convert(None)

        # Встановлюємо частоту на щоденну
        data = data.asfreq('D')

        # Заповнюємо пропущені значення
        data = data.fillna(method='ffill')
```

```
        logging.info(f"Тренуємо SARIMA модель з параметрами: {order}, сезонні параметри: {seasonal_order}")

        # Створюємо і тренуємо модель SARIMA
        model = SARIMAX(data, order=order, seasonal_order=seasonal_order)
        model_fit = model.fit(maxiter=1000, method='lbfgs')

        # Прогноз на вказану кількість днів
        predictions = model_fit.forecast(steps=forecast_days)

        # Встановлюємо індекс для прогнозованих даних
        predicted_index = pd.date_range(start=data.index[-1] + pd.Timedelta(days=1), periods=forecast_days, freq='D')
        predictions = pd.Series(predictions, index=predicted_index)

        result = {'predictions': predictions}

        # Обчислення середньоквадратичної похибки (MSE), якщо можливо
        if calculate_mse and len(data) >= forecast_days:
            mse = mean_squared_error(data[-forecast_days:], predictions[:len(data[-forecast_days:])])
            result['MSE'] = mse
            logging.info(f"MSE для SARIMA: {mse}")

        return result

    except Exception as e:
        logging.error(f"Помилка при застосуванні SARIMA моделі: {e}")
        return None

def apply_prophet_model(data, forecast_days=10):
    """
    Застосування моделі Facebook Prophet для прогнозування.

    Args:
    data (pd.Series): Часовий ряд (наприклад, ціни закриття).
    forecast_days (int): Кількість днів для прогнозування.

    Returns:
    dict: Прогнозовані значення з датами.
    """
    try:
        from prophet import Prophet

        data = data.copy()

        # Конвертуємо індекс у DatetimeIndex з UTC, потім видаляємо часову зону
        data.index = pd.to_datetime(data.index, utc=True).tz_convert(None)

        # Підготовка даних для Prophet
```

```
        df = data.reset_index().rename(columns={'Date': 'ds', data.name: 'y'})

        # Створюємо та тренуємо модель Prophet
        model = Prophet()
        model.fit(df)

        # Створюємо майбутній DataFrame
        future = model.make_future_dataframe(periods=forecast_days)

        # Прогнозуємо
        forecast = model.predict(future)

        # Отримуємо прогнозовані значення
        predictions = forecast[['ds', 'yhat']].set_index('ds')[-forecast_days:]

        return {'predictions': predictions['yhat']}

    except Exception as e:
        logging.error(f"Помилка при застосуванні Prophet моделі: {e}")
        return None

def auto_arima_model(data, forecast_days=10):
    """
    Автоматичний підбір найкращих параметрів ARIMA моделі.

    Args:
    data (pd.Series): Часовий ряд (наприклад, ціни закриття).
    forecast_days (int): Кількість днів для прогнозування.

    Returns:
    dict: Результати ARIMA з прогнозами та MSE (якщо обрано).
    """
    try:
        data = data.copy()

        # Конвертуємо індекс у DatetimeIndex з UTC, потім видаляємо часову зону
        data.index = pd.to_datetime(data.index, utc=True).tz_convert(None)

        # Встановлюємо частоту на щоденну
        data = data.asfreq('D')

        # Заповнюємо пропущені значення
        data = data.fillna(method='ffill')

        logging.info("Автоматичний підбір параметрів ARIMA моделі")

        # Автоматичний підбір параметрів
                model = pm.auto_arima(data, seasonal=False, trace=True,
error_action='ignore', suppress_warnings=True)
        model_fit = model
```

```
        # Прогноз на вказану кількість днів
        predictions = model_fit.predict(n_periods=forecast_days)

        # Встановлюємо індекс для прогнозованих даних
                  predicted_index    =    pd.date_range(start=data.index[-1]    +
pd.Timedelta(days=1), periods=forecast_days, freq='D')
        predictions = pd.Series(predictions, index=predicted_index)

        result = {'predictions': predictions}

        # Обчислення середньоквадратичної похибки (MSE), якщо можливо
        if len(data) >= forecast_days:
                           mse    =    mean_squared_error(data[-forecast_days:],
predictions[:len(data[-forecast_days:])])
            result['MSE'] = mse
            logging.info(f"MSE для автоматичної ARIMA: {mse}")

        return result

    except Exception as e:
        logging.error(f"Помилка при застосуванні автоматичної ARIMA моделі: {e}")
        return None

# Приклад використання (опціонально)
if __name__ == "__main__":
    from data_collection import fetch_financial_data
    from data_processing import clean_data

    symbol = 'AAPL'
    data = fetch_financial_data(symbol)
    if data is not None:
        cleaned_data = clean_data(data)
        if cleaned_data is not None:
            # Застосування ARIMA моделі
            arima_results = apply_arima_model(cleaned_data['Close'])
            if arima_results:
                print(arima_results['predictions'].head())
        else:
            print("Дані після очищення порожні.")
    else:
        print("Не вдалося завантажити дані.")
```

Додаток Д

Лістинг коду Dockerfile

```
# Dockerfile
FROM python:3.9-slim

# Встановлюємо системні залежності
RUN apt-get update && \
    apt-get install -y build-essential libpython3-dev python3-dev && \
    apt-get install -y libgomp1 && \
    apt-get install -y git && \
    apt-get install -y tzdata && \
    apt-get clean

# Встановлюємо часовий пояс
ENV TZ=Europe/Kyiv

# Налаштовуємо tzdata
RUN echo $TZ > /etc/timezone && \
    ln -fs /usr/share/zoneinfo/$TZ /etc/localtime && \
    dpkg-reconfigure -f noninteractive tzdata

# Встановлюємо змінні оточення
ENV PYTHONUNBUFFERED=1

# Встановлюємо робочу директорію всередині контейнера
WORKDIR /app

# Копіюємо файл requirements.txt
COPY requirements.txt /app/

# Встановлюємо необхідні пакети
RUN pip install --no-cache-dir -r requirements.txt

# Копіюємо решту коду додатку
COPY . /app/

# Відкриваємо порт для Dash
EXPOSE 8000

# Команда для запуску додатку
CMD ["python", "data_analysis.py"]
```

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____
_____
_____
_____
_____

Повністю вказується ПІБ та статус (освітня (освітньо-наукова) програма – для здобувачів вищої освіти, назва кваліфікаційної роботи)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)