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

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Моделі штучного інтелекту в задачах прогнозування курсів акцій та індексів на фондових ринках»

Виконав:
студент IV курсу, групи КІ-02
Трач Євгеній Сергійович _____

Керівник:
доцент, к.ф.-м.н., доцент
Шубенкова Ірина Анатоліївна _____

Консультант з економічного розділу:
професор, д.е.н., проф.кафедкри ЕК ФММ
Шевчук Олена Анатоліївна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри ШІ,
Кравець П.В. _____

Рецензент:
професор, д.т.н., доцент
Зайченко Олена Юріївна _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«31» січня 2024 р.

ЗАВДАННЯ

на дипломну роботу студенту

Трачу Євгенію Сергійовичу

1. Тема роботи «Моделі штучного інтелекту в задачах прогнозування цін акцій та індексів на фондових ринках», керівник роботи доц., к.ф.-м.н. Шубенкова Ірина Анатоліївна, затверджені наказом по університету від «31» травня 2024 р. № 2240-С

2. Термін подання студентом роботи «10» червня 2024 року.

3. Вихідні дані: Історичні дані цін акцій та індексів з онлайн-сервісу Yahoo! Finance

4. Зміст роботи: 1. Дослідження предметної області. 2. Аналіз методів штучного інтелекту в задачах прогнозування курсів акцій та індексів. 3. Практична реалізація роботи. 4. Функціонально-вартісний аналіз програмного продукту

5. Перелік ілюстративного матеріалу: доповнювальні ілюстративні зображення, схематичні зображення прикладів архітектури нейронних мереж, схематичні зображення архітектури реалізованих нейронних мереж, графіки порівняння отриманих прогнозованих значень та реальних значень.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Шевчук Олена Анатоліївна професор, д.е.н., доцент ЕК ФММ		

7. Дата видачі завдання «05» лютого 2024 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формування напряму дослідження	22.04.2024	Виконано
2	Аналіз актуальності задач стосовно напрямку дослідження	29.04.2024	Виконано
3	Аналіз відомих результатів стосовно напрямку дослідження	04.05.2024	Виконано
4	Формування задачі дослідження	05.05.2024	Виконано
5	Уточнення теми дипломної роботи	07.05.2024	Виконано
6	Написання програмного продукту	15.05.2024	Виконано
7	Підготовка розділів дипломної роботи	29.05.2024	Виконано
8	Оформлення розділів згідно вимог нормконтролю	01.06.2024	Виконано
9	Підготовка презентації доповіді	03.06.2024	Виконано
10	Оформлення дипломної роботи у цілому	06.06.2024	Виконано

Студент Євгеній ТРАЧ

Керівник Ірина ШУБЕНКОВА

РЕФЕРАТ

Дипломна робота: 86 с., 21 рис., 12 табл., 16 посилань, 1 додаток.

ПРОГНОЗУВАННЯ ЦІН АКЦІЙ ТА ІНДЕКСІВ МЕТОДАМИ ШТУЧНОГО ІНТЕЛЕКТУ, СТАТИСТИЧНІ МЕТОДИ, НЕЙРОННІ МЕРЕЖІ, ARIMA, SARIMA, RNN, LSTM, TRANSFORMERS

Об’єктом дослідження даної роботи є дослідження, аналіз та порівняння методів прогнозування курсів акцій та індексів на фондових ринках.

Предметом дослідження є моделі статистичного аналізу та моделі нейронних мереж.

Метою роботи є дослідження та оцінка характеристик різних моделей штучного інтелекту в задачах прогнозування цін різних акцій та індексів фондових ринків. Також метою є побудова програмного забезпечення для проведення вище зазначених досліджень та наочної демонстрації результатів цих досліджень.

ABSTRACT

Master's thesis: 86 p., 21 figures, 12 tables, 16 references, 1 appendix.

Artificial Intelligence models in tasks of forecasting stock prices and indices, Statistical Methods, Neural Networks, ARIMA, SARIMA, RNN, LSTM, Transformers

The object of this study is the research, analysis, and comparison of methods for forecasting stock prices and indices in stock markets.

The subject of the study is statistical analysis models and neural network models.

The purpose of the work is to investigate and evaluate the characteristics of various artificial intelligence models in the task of forecasting the prices of different stocks and stock market indices. Another goal is to develop software for conducting the aforementioned research and visually demonstrating the results of these studies.

Зміст

ВСТУП

Штучний інтелект в останні роки став необхідним інструментом у багатьох галузях, в тому числі і в фінансовому секторі. У контексті фондових ринків, прогнозування курсів акцій та індексів має велике значення для інвесторів, трейдерів та фондових аналітиків. Однак, ця задача є складною через велику кількість факторів, які можуть впливати на ціни акцій та індексів.

Ця дипломна робота присвячена дослідженню та порівнянню існуючих моделей штучного інтелекту у задачах прогнозування курсів акцій та індексів на фондових ринках. Метою дослідження є аналіз та порівняння ефективності різних моделей ШІ у прогнозуванні цін на акції та індекси. Для досягнення цієї мети будуть вирішені такі завдання:

- аналіз сучасних підходів до прогнозування цін акцій та індексів;
- розробка різних моделей ШІ для цієї задачі;
- емпіричне дослідження їхньої ефективності на реальних даних фондових ринків;

Це дослідження має велике значення для практичних застосувань у сфері інвестування та фінансового аналізу, а також для подальшого розвитку теорії та методів прогнозування на фондових ринках.

РОЗДІЛ 1 Дослідження предметної області

1.1 Актуальність задачі прогнозування цін

У світлі зростаючої складності ринкових умов та постійної необхідності ефективного стратегічного прийняття рішень, прогнозування цін акцій та індексів стає ключовою складовою фінансового аналізу. Сучасні методи прогнозування цін акцій та індексів мають свої обмеження через складність урахування всіх факторів, які впливають на ринок, та нестабільність ринкових умов. Ці обмеження викликають потребу у нових підходах, які можуть забезпечити більш точні та надійні прогнози.

У цьому контексті моделі штучного інтелекту виявляються особливо перспективними, оскільки вони можуть ефективно аналізувати великі обсяги даних та виявляти складні зв'язки, недосяжні для традиційних методів аналізу. Нинішній швидкий темп розвитку технологій у галузі штучного інтелекту, разом із зростаючою доступністю даних та підвищенням обчислювальної потужності, створює сприятливі умови для використання моделей штучного інтелекту у фінансовому аналізі.

Ураховуючи сучасні виклики та потреби фінансового сектору, дослідження в галузі прогнозування цін акцій та індексів за допомогою моделей штучного інтелекту є актуальним та перспективним напрямом, який має великий практичний потенціал.

1.2 Коротка історія і сучасний вигляд предметної області

Фондовий ринок – частина ринку капіталів, де здійснюється емісія, купівля та продаж цінних паперів. У сучасному світі фондові ринки відіграють ключову роль у глобальній економіці, дозволяючи компаніям залучати капітал

для розвитку та інвесторам здійснювати прибуткові операції. Фондові ринки є важливим інструментом для формування цін на активи, визначення економічних тенденцій та здійснення монетарної політики країн[1].

Початки історії фондових ринків можна відстежити ще у XVII столітті, коли в Європі з'явилися перші акціонерні компанії та фондові біржі. Одним з найвідоміших прикладів є Голландська Східно-Індійська компанія, акції якої торгувалися на Амстердамській біржі у XVII столітті.

У XIX столітті фондові ринки стали більш організованими, і з'явилися перші фондові брокери та банки, що сприяло зростанню обсягів торгівлі та розвитку фінансової системи. У XX столітті фондові ринки пройшли чергову хвилю розвитку, зокрема завдяки широкому впровадженню електронних торгових платформ та іншим технологічним інноваціям. Важливими подіями стали крахи фондових ринків у 1929 та 1987 роках, які викликали серйозні реформи у фінансовій системі та регулюванні фондових ринків, а також у зміні підходів та стратегій торгівлі [2].

Штучний інтелект, як інструмент, з’явився відносно недавно, але за останні десятиліття, з розвитком технологій ШІ в цілому відіграє значну роль в світі фондових ринків. Вже у 1980-х роках виникли перші спроби застосування штучного інтелекту для прогнозування цін акцій та оптимізації інвестиційних портфелів.

Одним з важливих кроків в застосуванні ШІ у фінансах було створення нейромереж, які здатні адаптуватися до змінних умов ринку та швидко аналізувати величезні обсяги даних. Наприклад, в середині 1990-х років нейромережі почали застосовуватися для прогнозування цін акцій та виявлення тенденцій на фондовому ринку.

За останні десятиліття зростаючий обсяг доступних даних та розвиток обчислювальних технологій сприяли швидкому розвитку алгоритмів машинного навчання та глибокого навчання. Це призвело до появи нових методів аналізу даних, таких як алгоритми класифікації, кластеризації та прогнозування, які стали невід'ємною частиною фінансового аналізу.

Сьогоднішній фінансовий світ вже не уявляє себе без штучного інтелекту. Від інвестиційних банків до фондових бірж, від фінансових аналітиків до трейдерів, всі вони активно використовують різноманітні моделі ШІ для прийняття рішень, розробки стратегій та оптимізації портфелів. Використання штучного інтелекту в фінансах стало не лише привілеєм великих компаній, але й ключовим інструментом для багатьох малих та середніх інвесторів, які бажають зробити свої операції більш ефективними та прибутковими.

1.3 Аналіз існуючих підходів

Як видно з історії фондові ринки, як явище, з’явилися вже доволі давно, а отже методів їх прогнозування також є велике різноманіття: від загальних емпіричних підходів до задачі, до строгих сучасних технічних рішень. Для більш детального вивчення питання пропонується розглянути такі методи, проаналізувати їх, та зробити висновки про їх ефективність, плюси та мінуси.

1.3.1 Фундаментальний аналіз

Фундаментальний аналіз є одним найдавніших підходів до аналізу фінансових ринків. В його основі лежить вивчення фінансово-економічної інформації активу, яка, ймовірно, робить вплив на динаміку його ціни. В фундаментальному аналізі фондових ринків використовуються різні фінансові показники компанії, такі як її вартість, виручка, прибуток, грошові зобов’язання, розміри дивідендів, виробничі показники, тощо.

Методами фундаментального аналізу можна вважати аналіз фінансових звітів, різноманітних біржових показників активу, аналіз його підприємств, та

аналіз кадрових рішень. Всі ці методи в купі дають можливість зробити певні емпіричні висновки про теперішній стан активу, та його перспективи в порівнянні з конкурентами. Використовують фундаментальний аналіз в основному для оцінки привабливості на довгострокову перспективу, та для оцінки ризиків пов'язаних з інвестиціями в певний актив.

Хоча аналіз таких базових показників і є доволі об'єктивним, таке велике різноманіття та об'єми інформації робить цей метод прогнозування дуже суб'єктивним, та таким, який важко формалізувати, що і є його головним недоліком порівняно з іншими методами. Також варто зазначити великі часові витрати на аналіз великих об'ємів даних, для того щоб мати уяву про стан активу, та його перспективи, і залежність цього методі від точності та об'єму даних, які можуть бути не завжди достовірними та/або достатніми [3].

1.3.2 Технічний аналіз

Технічних аналіз з'явився дещо пізніше, в кінці XIX століття американський журналіст Чарльз Доу опублікував серію статей про ринок цінних бумаг, що пізніше лягли в основу теорії Доу і, по суті своїй, є початком історії технічного аналізу.

Рисунок 1.1 – Приклад технічного аналізу графіка ціни EUR до USD[1]

Не вдаючись в деталі технічний аналіз можна звести до простого заключення: прогнозування ціни актива в майбутньому можливе на основі його минулої історії цін. В основі цього лежить аналіз часових рядів. Цей метод є набагато більш строгим та формалізованим в порівнянні з фундаментальним аналізом. Його інструменти та методи, ті які описав ще сам Чарльз Доу, та ті які з'явилися з пізніше, добре відомі, та з часом показали свою роботоздатність та ефективність.

Проте і цей метод не є гарантовано ефективним. Ігнорування фундаментальних факторів в технічному аналізі призводить до того що різкі, неочікувані скачки цін, пов'язані з певними подіями в реальному світі, можуть привести до збитків інвестора. Також варто зазначити, що хоча всі методи та показники добре описані та характеризовані, воно все ще носять рекомендаційний характер, та не гарантують що ціна піде в потрібному напрямку. Тому інтерпретація графіків та індикаторів технічного аналізу все ще є роботою інвестора або трейдера, а тому носить суб'єктивний характер [3].

Технічний аналіз ліг в основу майбутнього розвитку інструментів прогнозування з використанням обчислювальної техніки.

---

[1] Джерело зображення https://uk.wikipedia.org/wiki/Технічний_аналіз

1.3.3 Алгоритмічна торгівля

Алгоритмічна торгівля, або алготрейдинг, є сучасним підходом до торгівлі на фінансових ринках, який використовує комп'ютерні програми для автоматичного виконання торгових операцій. Основою алгоритмічної торгівлі є математичні моделі та алгоритми, що базуються на методах та індикаторах вже добре відомого технічного аналізу. Вони аналізують ринкові дані і приймають рішення про купівлю або продаж активів без втручання людини. Цей підхід дозволяє трейдерам швидко реагувати на ринкові зміни, знижуючи людський фактор і мінімізуючи емоційні впливи на процес торгівлі.

Алгоритмічна торгівля має як свої переваги та і недоліки. Однією з основних переваг є висока швидкість виконання торгових операцій, що дозволяє отримати прибуток від найменших змін цін на ринку. Крім того, алгоритмічний трейдинг може одночасно аналізувати величезні обсяги даних за малі проміжки часу, а також виконувати декілька операцій одночасно, що недосяжно для людських трейдерів [4].

Значний розвиток алгоритмічної торгівлі відбувся завдяки прогресу в області комп'ютерних технологій та великих даних. Сучасні алгоритми використовують методи машинного навчання, глибинного навчання і обробки природної мови для аналізу ринкових тенденцій і прогнозування цінових рухів. Вони можуть враховувати як історичні дані, так і поточні новини, соціальні медіа та інші фактори, що впливають на ринок, забезпечуючи більш точні і швидкі рішення.

Алгоритмічна торгівля також включає різні стратегії, такі як арбітраж, маркет-мейкінг, імпульсна торгівля та інші. Кожна з цих стратегій має свої унікальні особливості і підходи до аналізу ринку. Наприклад, арбітражна торгівля шукає невідповідності в цінах одного й того самого активу на різних ринках, тоді як імпульсна торгівля намагається скористатися короткостроковими трендами і коливаннями цін.

1.3.4 Машинне навчання та штучний інтелект

Машинне навчання та штучний інтелект є одними з найбільш інноваційних підходів до аналізу фінансових ринків і прогнозування курсів акцій. Основна ідея машинного навчання полягає у створенні моделей, які можуть навчатися на основі історичних даних і робити передбачення щодо майбутніх ринкових рухів. Використання штучного інтелекту в торгівлі дозволяє автоматизувати складні аналітичні процеси, що значно підвищує точність і швидкість прийняття рішень.

Машинне навчання та штучний інтелект мають широке застосування у фінансовому аналізі. Однією з ключових технологій є глибинне навчання, яке використовує багатошарові нейронні мережі для виявлення складних патернів у великих наборах даних. Наприклад, рекурентні нейронні мережі (RNN) та довготривала короткострокова пам'ять (LSTM) є особливо ефективними для аналізу часових рядів і прогнозування цін на акції. Ці моделі здатні враховувати як довгострокові тенденції, так і короткострокові коливання ринку. Ще одним з напрямків є обробка природної мови (NLP). Використовуючи NLP, моделі можуть аналізувати текстову інформацію з новинних статей, соціальних мереж та фінансових звітів, щоб визначити настрій ринку та його потенційний вплив на ціни акцій. Наприклад, аналіз настроїв у соціальних медіа може допомогти передбачити раптові зміни на ринку, викликані новинами або публічними заявами.

Однак, використання ШІ у фінансових ринках має низку своїх недоліків. Однією з головних проблем є необхідність великих обсягів якісних даних для навчання моделей. Крім того, складність і непрозорість деяких моделей глибинного навчання можуть ускладнювати їх інтерпретацію і пояснення. Також існує ризик перенавчання, коли модель добре працює на тренувальних даних, але показує погані результати на нових, невідомих даних [4].

Проте, незважаючи на ці виклики, штучний інтелект має великий потенціал для революціонізації у сфері фінансових ринків. Він може значно підвищити ефективність та прибутковість торгових операцій, забезпечуючи як інвесторів, так і трейдерів, потужними інструментами для аналізу та прийняття рішень. З розвитком технологій і збільшенням доступності даних, застосування цих методів у фінансових ринках продовжує зростати, відкриваючи нові можливості для інвесторів і аналітиків.

1.4 Постановка задачі

Задача прогнозування курсів акцій та індексів на фондових ринках є складною і багатоаспектною проблемою, яка потребує застосування сучасних методів аналізу даних. У цій дипломній роботі ставиться мета дослідити та порівняти існуючі моделі штучного інтелекту які використовуються в задачах прогнозування цін акцій та індексів на фондових ринках.

Результатом цього дослідження стане програма з різними моделями статистичних методів та методів штучного інтелекту, натренованих на різних наборах датасетів, завданням якої буде наглядно показати різницю в точності роботи цих методів на різних наборах даних та різних проміжках часу.

Для досягнення цієї мети буде проведено декілька етапів дослідження, зокрема:

- аналіз існуючих підходів до прогнозування цін акцій та індексів на фондових ринках з застуванням статистичних методів та методів штучного інтелекту. Провести огляд літератури щодо сучасних методів та підходів, визначити їх переваги та недоліки;
- вибір зі знайдених методі ті, які теоретично, найкраще справляються з поставленою задачею прогнозування цін акцій та індексів;

- пошук валідних датасетів з потрібними наборами даних, які будуть включати в себе історію торгівлі акції або індексу, їх торгові дані, та включати в себе інформацію достатню для тренування та оцінки моделей;
- розробка вибраних моделей штучного інтелекту, їх налаштування, оптимізація, тренування та оцінка результативності їх роботи з метою підвищення їх точності;
- дослідження розроблених моделей, проведення емпіричного аналізу ефективності розроблених моделей на основі реальних даних фондових ринків. Здійснення порівняння точності прогнозів різних моделей, їх збіг з реальними рухами цін;
- аналіз та інтерпретація отриманих результатів.

Висновки до розділу 1

У першому розділі даної дипломної роботи було здійснено всебічне дослідження предметної області, що включає аналіз актуальності задачі прогнозування цін на фондових ринках, історичний огляд розвитку цієї галузі, а також детальний розгляд основних підходів та методів, що використовуються для прогнозування цін у фінансовій сфері, а саме на фондових ринках.

У підсумку, цей розділ дозволив глибоко ознайомитися з сучасними тенденціями та методами у сфері прогнозування цін на фондових ринках, а також підготувати основу для подальшого практичного дослідження, що включає розробку та тестування моделей штучного інтелекту для вирішення задачі прогнозування цін акцій та індексів на фондових ринках.

РОЗДІЛ 2 Аналіз методі штучного інтелекту в прогнозуванні курсів акцій та індексів

Методи штучного інтелекту в даній роботі можна поділити на статистичні методи, тобто такі що працюють на основі прийомів та методів математичної статистики, та нейронні мережі – один з методів машинного навчання. Також варто зазначати, що задача прогнозування цін акцій та індексів, по своїй суті, зводиться до задачі прогнозування часових рядів, відповідно методи які використовують для прогнозування часових рядів так само ефективні і в задачах прогнозування цін [8]. Розглянемо їх основні варіації, плюси та мінуси.

1.2 Статистичні методи

1.2.1 Авторегресивні моделі

Модель авторегресії (Auto Regression model, AR) – статистична модель яка використовує попередні значення змінної, які лінійно залежать від поточних, для передбачення майбутніх значень [5]. Модель AR порядку p (AR(p)) записують таким чином:

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-1} + \epsilon_t,$$

де y_t – значення ряду в момент часу t;
c – константа (можу бути 0 в деяких моделях);
ϕ_i – параметри моделі які вказують на вплив попередніх значень на поточне;
ϵ_t – випадкова похибка в момент часу t, яка зазвичай вважається нормально розподіленою з нульовим середнім та сталою дисперсією.

Перевагами цієї моделі є:

– простота у використанні та інтерпретації;
– легко застосовувати для короткострокових прогнозів.

Недоліками можна назвати:

– може бути недостатньо точною для складних та великих часових рядів;
– не враховує багато параметрів, такі як сезонність або зміна середньої, які в певних випадках можуть бути вкрай важливими, в тому числі і в сфері фінансів.

AR-моделі є важливими інструментами в аналізі та прогнозуванні часових рядів, і хоч вона має значні недоліки, вона зазвичай є складовою більш складних моделей.

1.2.2 Ковзаюче середнє

Ковзаючі середні - метод статистичного аналізу, який використовується для згладжування коливань даних та виявлення тенденцій [5]. Існує декілька підвидів MA моделей.

1. Просте ковзаюче середнє (Simple Moving Average, SMA) – середнє арифметичне значення певної кількості попередніх значень.

$$SMA_t = \frac{y_t + y_{t-1} + \cdots + y_{t-(n+1)}}{n},$$

де y_t – значення ряду в момент часу t.

2. Зважене ковзаюче середнє (Weighted Moving Average, WMA) – відрізняється від простого тим, що кожне значення у часовому ряді має різну вагу, зазвичай таку, яка збільшується до кінця ряду.

$$WMA_t = \frac{\sum_{i=0}^{n-1} \omega_i y_{t-i}}{\sum_{i=0}^{n-1} \omega_i},$$

де ω_i – ваги, що надаються кожному значенню y_{t-i}.

3. Експоненціальне ковзаюче середнє (Exponential Moving Average, EMA) – ковзаюче середнє, яке надає більшого значення останнім значенням за допомогою експоненціальної функції, відповідно цей вид ковзаючої середньої є більш чутливим до нових даних.

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

де α – коефіцієнт згладжування, що визначається як $\alpha = \frac{2}{n+1}$.

До переваг MA можна віднести:

– простота у використанні та інтерпретації;
– ефективність у згладжуванні даних та виявленні трендів.

Недоліками можна назвати:

– простота у використанні та інтерпретації;
– затримка сигналу, особливо для SMA, через використання виключно історичних даних.

1.2.3 ARMA

ARMA (Auto Regression Moving Average) – комбінована модель для аналізу та прогнозування часових рядів, яка об’єднує авторегресійну модель AR(p) та модель ковзаючої середньої MA(q). Завдяки такому комбінуванню вона поєднує як залежність маж минулими та поточними значеннями ряду, так і залежності між минулими та поточними похибками.

Модель ARMA порядку записується таким чином:

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-1} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \epsilon_t,$$

де y_t – значення ряду в момент часу t;

c – константа (можу бути 0 в деяких моделях);

ϕ_i – коефіцієнт авторегресивної частини (AR);

θ_j – коефіцієнт частини ковзаючого середнього (MA);

ϵ_t – випадкова похибка в момент часу t.

Перевагами моделі ARMA є:

– гнучкість у моделюванні різних типів часових рядів завдяки об’єднанню AR та MA моделей;
– можливість адекватного опису стаціонарних часових рядів.

До недоліків:

– складність у виборі правильних порядків моделі p та q;
– вимога до стаціонарності ряду.

Загалом модель за рахунок об’єднання двох інших моделей є більш універсальною та точною, проте вона певні недоліки, та все ще не вирішує всіх проблем при прогнозуванні реальних даних [6].

1.2.4 ARIMA

ARIMA (Auto Regressive Integrated Moving Average) – розширення для моделі ARMA, яка додає до AR(p) та MA(q) ще частину інтеграції I(d). Інтеграція – компонент, що використовується для перетворення нестаціонарного ряду в стаціонарний шляхом різниціювання. В іншому ж модель залишилась тією самою.

Модель ARIMA записують таким чином:

$$(1 - \sum_{i=1}^{p} \phi_i L^i)(1 - L)^d y_t = (1 + \sum_{j=1}^{q} \theta_j L^j)\epsilon_t,$$

де y_t – значення ряду в момент часу t;

c – константа (можу бути 0 в деяких моделях);

ϕ_i – коефіцієнт авторегресивної частини (AR);

θ_j – коефіцієнт частини ковзаючого середнього (MA);

L – оператор зсуву (lag operatot), що визначається як $L^k y_t = y_{t-k}$;

d – порядок інтеграції що визначає кількість різниціювання, необхідної для перетворення ряду в стаціонарний;

ϵ_t – випадкова похибка в момент часу t.

Перевагами моделі ARIMA є:

– гнучкість у моделюванні різних типів часових рядів, в тому числі і нестаціонарних;
– можливість повного опису залежностей у даних.

До недоліків:

– складність у виборі правильних порядків моделі p, q та i;
– набагато важча за попередні модулі, потребує більше ресурсів для обчислення великих рядів.

Завдяки можливості працювати з нестаціонарними рядами модель ARIMA можна вважати першою моделлю яка здатна самостійно працювати з реальними даними, аналізувати та прогнозувати їх з адекватною точністю [6,7].

1.2.5 SARIMA

SARIMA (Seasonal ARIMA) – розширення моделі ARIMA яке враховує сезонні компоненти в часових рядах. Модель SARIMA включає в себе всі компоненти моделі ARIMA (авторегресія, інтеграція, ковзаюче середнє), а також додає сезонні компоненти для кожного з них.

Перевагами цієї моделі є:

- гнучкість у моделюванні різних типів часових рядів, як і ARIMA, в тому числі і рядів які мають ознаки сезонності;
- точність прогнозування як короткострокових, так і довгострокових залежностей в рядах.

До недоліків:

- ще більша кількість порядків моделі, які необхідно вибирати;
- ще більша за модель ARIMA обчислювальна складність для великих наборів даних;

Загалом модель, наразі, є найбільш потужним статистичним інструментом прогнозування часових рядів широкої варіативності. Вона враховує стаціонарні та нестаціонарні ряди, добре працює як з сезонними так і не сезонними рядами, відповідно є найбільш універсальним методом [7].

1.3 Нейронні мережі

1.3.1 Згорткові нейронні мережі

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) - один з видів глибоких нейронних мереж. Вперше цей клас нейронних мереж з’явився в кінці 1980-х, початку 1990-х років, був представлений Яном Лекуном, як варіант вирішення проблеми розпізнавання рукописних цифр. Проте з часом виявилася їхня певна ефективність в прогнозуванні часових рядів.

Модель добре себе показує саме в аналізі зображень, вона зазвичай використовується в задачах розпізнавання образів, обробці відео та комп’ютерного зору [9].

CNN складається з деяких основних шарів та їх комбінацій. Можна виділити такі основні шари.

1. Згортковий шар (Convolutional Layer) – ядро або фільтр, яке ковзає по вхідному зображенню і обчислює згортку, генеруючи карту ознак.
2. Шар підвибірки (Pooling Layer) – шар який зменшує кількість ознак зберігаючи важливу інформацію, допомагаючи зменшити обчислювальні витрати та уникати перенавчання.
3. Шар випрямлення (ReLU Layer) – функція що змінює всі від’ємні значення на нуль.
4. Повнозв’язний шар (Fully connected Layer) – в цьому шарі кожен нейрон з’єднаний з усіма нейронами попереднього шару, завдяки чому в ньому об’єднуються ознаки та приймаються високорівневі рішення.
5. Нормалізація, регуляція та інші техніки – додаткові шари які виконують різні ролі, такі як Batch Normalization або Dropout.

До переваг CNN можна віднести:

– ефективність та автоматизованість – ефективно витягує просторові ознаки та не потребує втручання в процесі роботи;
– інваріативна до трансформацій – добре справляється з зміна масштабу, поворотами та зсувами об’єктів.

З недоліків можна назвати:

– необхідність ретельного налаштування гіперпараметрів;
– важка в інтерпретації – не завжди є очевидним, що саме було вивчено моделлю;
– досить високі обчислювальні витрати.

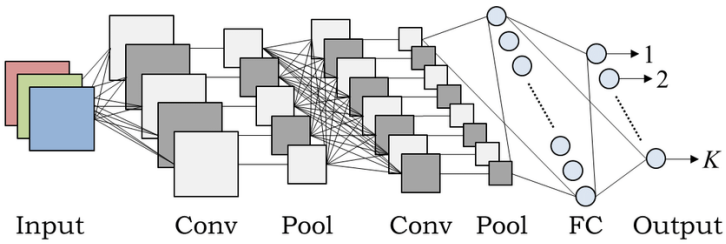

Рисунок 2.1 – Приклад архітектури CNN[2]

Загалом мережа хоч і може застосовуватися в фінансовій сфері, а саме в таких задачах як виявлення сезонності, циклічності, або трендів на графіках, все ж погано працює саме в задачах прогнозування часових рядів.

1.3.2 Рекурентні нейронні мережі

Рекурентні нейронні мережі (Recurrent Neural Networks, RNN) – ще один вид штучних нейронних мереж розроблений для обробки послідовних даних, таких як текст, відео, або часові ряди. Особливістю цього класу нейронних мереж є наявність зворотних зв’язків, що дають їм змогу зберігати інформацію про попередні кроки у послідовності, тобто мати “пам’ять”.

Модель часто використовується в задачах обробки природної мови, розпізнаванні мовлення, аналізі відео, та аналізі часових рядів. Саме через це, вона добре підходить для вирішення задачі прогнозування цін.

Головна ідея RNN полягає в збереженні прихованого стану, який оновлюється на кожному кроці часу на основі поточного значення та попереднього прихованого стану. Навчання RNN здійснюється за допомогою методу зворотного поширення помилки у часі (Backpropagation Through Time,

[2] Джерело зображення https://www.researchgate.net/figure/An-example-of-CNN-architecture_fig1_320748406

BPTT). Цей метод є розширенням стандартного алгоритму зворотного поширення помилки для врахування послідовного характеру даних [10].

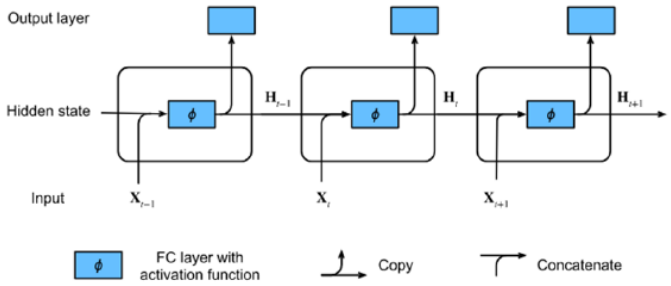

Рисунок 2.2 – Приклад архітектури RNN[3]

Серед переваг моделі RNN можна назвати:

– RNN може обробляти послідовності даних будь-якої довжини;
– звичайна RNN є досить простим алгоритмом, та є основою для розуміння більш складних її варіацій.

Проте вона має ряд серйозних недоліків:

– проблема затухання градієнта – при навчанні довгих послідовностей градієнти можуть затухати (ставати дуже малими), що призводить до повільного або взагалі неможливості подальшого навчання;
– проблема вибуху градієнта – градієнти можуть почати експоненціально рости, що призводить до нестабільності навчання.

Не зважаючи на проблеми RNN все ще є хорошим варіантом для аналізу та прогнозування часових рядів, та може видавати хороші результати, якщо враховувати її недоліки. Модель RNN стала основою для більш складних, кращих моделей, таких як LSTM та GRU, про які піде мова далі.

[3] Джерело зображення https://ml-cheatsheet.readthedocs.io/en/latest/layers.html#rnn

1.3.3 Long Short-Term Memory

LSTM (Long Short-Term Memory) – покращений варіант звичайної RNN, який вирішує проблеми затухання та вибуху градієнтів, характерних для її попередника. Ця варіація може зберігати інформацію протягом довгих часових періодів, що робить її корисною для задач з довгостроковими залежностями в послідовностях даних.

Відмінність LSTM від звичайної RNN є структура її вузлів, що мають спеціальні компоненти – “комірки пам’яті” (memory cells) та декілька різних “шлюзів” (gates). Memory cells зберігають інформацію протягом довгих часових періодів. Gates контролює потік інформації до цих комірок, до прикладу Input Gate контролює яку інформацію додавати до комірки. В іншому ж все залишилось як з звичайною RNN.

LSTM мережі навчаються аналогічно звичайним RNN за допомогою методу зворотного поширення помилки у часі (Backpropagation Through Time, BPTT). Завдяки спеціальним шлюзам, LSTM мають здатність ефективно передавати градієнти через великі часові інтервали, що дозволяє їм уникати проблем зникнення або вибуху градієнтів[10].

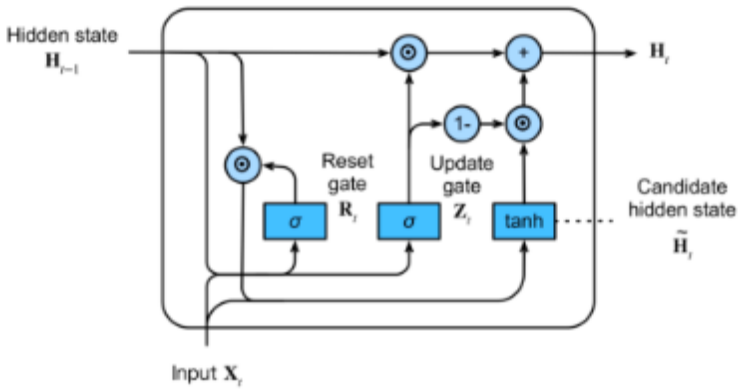

Рисунок 2.3 – Приклад архітектури LSTM[4]

[4]Джерело зображення https://ml-cheatsheet.readthedocs.io/en/latest/layers.html#lstm

Перевагами LSTM є:

– добре працює з довгостроковими часовими рядами завдяки здатності зберігати інформацію;
– стабільне навчання, завдяки вирішенню проблем затухання та вибуху градієнта.

Серед мінусів варто згадати:

– складність налаштування через збільшення кількості параметрів;
– збільшення обчислювальних витрат в порівнянні з RNN.

Загалом LSTM є дуже хорошим варіантом для прогнозування часових рядів, а відповідно і для задачі прогнозування цін. Воно одночасно добре працює як з короткими та довгими часовими рядами, а її недоліків в більшості випадків можна ігнорувати [11].

1.3.4 Gated Recurrent Unit

GRU (Gated Recurrent Unit) – ще одна варіація RNN. Цей тип був запропонований як спрощена версія LSTM, та намагалася вирішити її недоліки. Відмінністю від LSTM було зменшення обчислювальних витрат і певне спрощення моделі, при цьому намагалися зберегти здатність моделювати довгострокові залежності. Цього вдалося досягти завдяки об’єднанню декількох компонентів LSTM в одні шари.

В GRU можна виділити дві основні структури які керують потоками інформації – оновлюючий шлюз (Update gate) який відповідає за те, скільки інформації потрібно перенести з попереднього прихованого стану до нового прихованого стану, та шлюз скидання (Reset gate), який вирішує скільки інформації з попереднього прихованого стану потрібно забути.

В іншому принципи та компоненти залишились тими самими, проте таке спрощення вже дозволило скоротити обчислювальні ресурси.

GRU, як і інші рекурентні нейронні мережі, навчаються за допомогою методу зворотного поширення помилки у часі (Backpropagation Through Time, BPTT). Завдяки своїм шлюзам, GRU здатні ефективно передавати градієнти через великі часові інтервали, що дозволяє їм уникати проблем зникнення або вибуху градієнтів.

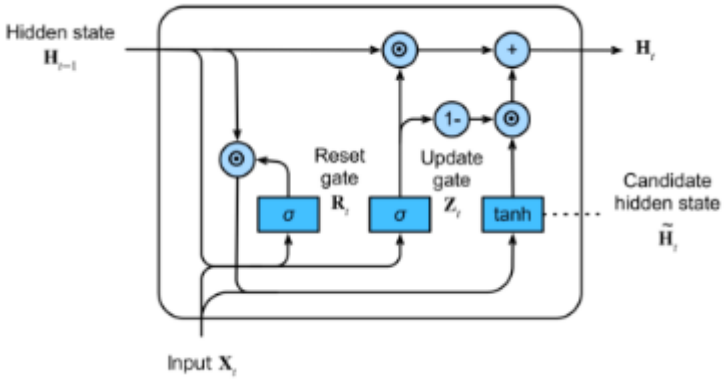

Рисунок 2.4 – Приклад архітектури GRU[5]

Переваги:

– GRU мають менше параметрів у порівнянні з LSTM, що робить їх менш обчислювально затратними.
– менша кількість параметрів та спрощена архітектура дозволяють швидше навчання;
– ефективно зберігають інформацію протягом довгих часових інтервалів.

Недоліки:

– хоча GRU є більш простими, вони можуть бути менш гнучкими у порівнянні з LSTM для деяких специфічних задач;
– хоча GRU мають менше параметрів ніж LSTM, вони все ще вимагають значних обчислювальних ресурсів у порівнянні з ванільними RNN.

[5] Джерело зображення https://ml-cheatsheet.readthedocs.io/en/latest/layers.html#gru

Підсумовуючи можна сказати, що GRU є непоганою альтернативою LTSM, особливо в ситуаціях коли обчислювальні ресурси є обмеженими, і без сумнівів, кращою альтернативою звичайним RNN, проте через свої спрощення він, в середньому, показує дещо менш точні результати [11].

1.3.5 Трансформери

Трансформери (Transformers) — це тип архітектури нейронних мереж, який був запропонований для обробки послідовних даних. Вони використовуються для завдань, що включають обробку природної мови (NLP), таких як машинний переклад, генерація тексту та інші. Основна перевага трансформерів полягає у здатності ефективно обробляти довгі послідовності, уникаючи проблем, властивих рекурентним нейронним мережам. Вперше трансформери були представлені в статті 2017 року "Attention is All You Need" від Google. Їх застосовують в таких відомих нейронних мережах як GPT та BART.

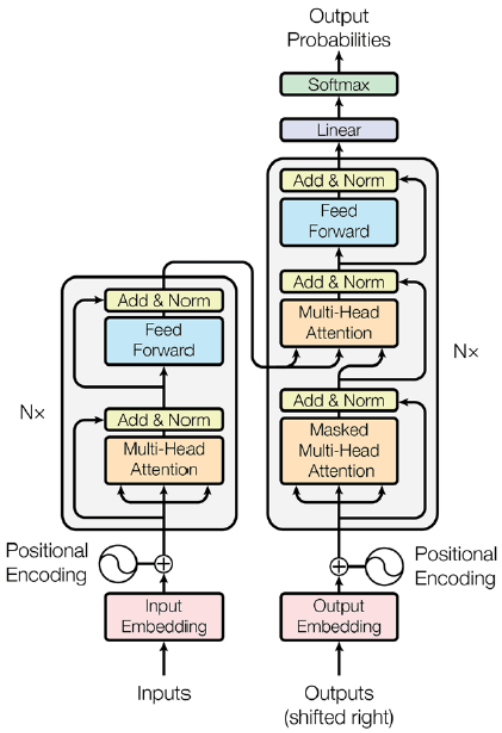

Рисунок 2.5 – Загальна архітектура трансформерів[6]

Основна ідея цього класу нейронних мереж є використання механізму уваги завдяки якому кожен елемент послідовності може звертати увагу на інші елементи, оцінюючи їхню важливість. Також використовуються енкодери та декодери, які працюючи разом обробляють вхідні данні та генерацію вихідних. Це дозволяє моделі ефективно захоплювати залежності між далекими елементами послідовності [12].

Переваги:

– на відміну від рекурентних моделей, трансформери можуть обробляти всі елементи послідовності паралельно;
– механізм уваги дозволяє ефективно враховувати залежності між далекими елементами послідовності;
– легко масштабується для дуже великих моделей.

---

[6] Джерело зображення https://machinelearningmastery.com/wp-content/uploads/2021/08/attention_research_1.png

Недоліки:

- механізм уваги має квадратичну складність щодо довжини послідовності, що може бути проблематичним для дуже довгих послідовностей;
- для ефективного навчання трансформерів часто потрібні великі обсяги даних.

Загалом трансформери є цікавим сучасним рішенням, і хоча вони не є пристосовані для прогнозування часових рядів, вони є доволі універсальними, а тому все ще можуть себе показати з хорошого боку і в цій сфері.

1.4 Підходи до налаштування та тестування методів

1.4.1 Підготовка даних для подальшого аналізу

Першим кроком в підготовці до аналізу та прогнозування стане вибір датасетів, на яких моделі і будуть навчатися. Цей крок є досить важливим, так як не повні датасети не зможуть якісно навчати моделі, а занадто короткі, або навпаки, занадто довгі можуть створити проблеми для певних моделей. Тому до цього питання варто підійти з усією увагою [13].

Перш за все розглянемо що представляє з себе звичайний датасет по певній акції. Зазвичай це своєрідний часовий ряд з вказаними параметрами на певну дату. Цими параметрами може бути ціна на момент відкриття торгів, ціна на момент закриття торгів, максимальна або мінімальна ціна за торговий день.

На мою думку, для якісного порівняння роботи різних моделей, варто сконцентрувати свою увагу на одному з цих параметрів, та прогнозувати саме його, так як кожен з них окремо буде мати свою специфіку, та можуть відрізнятися по своїм характерам росту або падіння. Також варто підбирати датасети різної довжини від декількох років, до декількох десятків років. Це дозволить робити висновки про ефективність моделі на різних наборах даних.

Після вибору датасетів дані варто підготувати для роботи з ними, створити об’єкти в коді з якими можна працювати. Далі дані варто нормалізувати. Через можливий великий діапазон значень сирих даних в певних моделей можуть виникнути складнощі при роботі з ними. Для цього і застосовують нормалізацію. В цій роботі ми скористаємося підходом масштабування ознак (Feature scaling) щоб з зведенням значень даних до діапазону [0, 1].

Після цього датасет можна вважати готовим. Залишилось розбити його на тренувальні та валідаційні частини. Тренувальну частину використовують для навчання модулі, вона повинна бути достатньо великою та різноманітною, щоб навчання моделі було якісним. Валідаційну частину використовують для оцінки ефективності навчання, вона може меншою за тренувальну, проте всерівно має бути достатньо великою щоб включати в себе різноманітні патерни та ситуації з якими модель може зіткнутися при роботі. Зазвичай беруть співвідношення 70% для тренувального, та 30% для валідаціного. Ці самі значення будуть використані і в цій роботі.

1.4.2 Налаштування моделей

Налаштування моделей машинного навчання включає кілька ключових етапів, серед яких налаштування гіперпараметрів, вибір оптимізатора і функції втрат є надзвичайно важливими. Розглянемо ці етапи більш детально.

Гіперпараметри – це параметри, значення яких встановлюються до початку навчання моделі і не змінюються в процесі [14]. Основні гіперпараметри включають:

- кількість шарів і кількість нейронів в кожному шарі;
- розмір векторів (embedding size) для текстових даних;

- розмір вікна (kernel size) та кількість фільтрів у згорткових мережах;
- кількість голов уваги (attention heads) і кількість шарів у трансформерах.

Оптимізатор – це алгоритм, який змінює параметри моделі для мінімізації функції втрат. Найпоширеніші оптимізатори включають SGD, Adam, RMSprop, Adagrad, Momentum та інші. Кожен з оптимізаторів має свої особливості, які ватро враховувати, до прикладу Adagrad, як правило, є дуже повільним в навчанні, а SGD може застрягати в локальних мінімумах. Вибір оптимізатора під конкретну задачу є важливим аспектом налаштування моделі, проте “правилом великого пальця” є використання оптимізатора Adam за його найбільшу універсальність.

Функція втрат (або цільова функція) вимірює, наскільки добре модель робить свої передбачення. Вибір функції втрат залежить від конкретного завдання, це може бути завдання регресії, класифікації або обробки послідовностей. Задача прогнозування цін відноситься до регресивних завдань, а тому для них підходять функції втрат Mean Squared Error (MSE) або Mean Absolute Error (MAE).

Вибір оптимізатора та функції втрат відбувається методом проб та помилок, де тестуючи кожен оптимізатор та кожну функцію втрат ми шукаємо ту, яка дає найточніші результати з найменшою кількістю похибок [15].

Налаштування моделей, вибір оптимізатора і функції втрат є критичними аспектами машинного навчання. Правильний підхід до кожного з цих етапів може суттєво покращити якість і продуктивність моделі, дозволяючи досягати кращих результатів у вирішенні конкретних задач.

1.4.3 Оцінка результативності моделі та інтерпретація результатів

Останнім кроком в процесі тестування програми є оцінка її результативності та інтерпретації отриманих в результаті роботи даних.

Оцінка результативності моделі є критичним етапом у процесі машинного навчання. Вона дозволяє визначити, наскільки добре модель навчається на даних і як ефективно вона передбачає результати на нових, невідомих даних. Для задач прогнозування часових рядів, які належать до завдань регресії, існує декілька метрик, які зазвичай використовуються для оцінки продуктивності моделі. Окрім вже зазначених вище функцій витрат MSE та MAE можна згадати ще Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE) та R-squared (R^2), які також маючи свої особливості можуть краще справлятися з тими чи іншими моделями.

Ще одним важливим кроком в оцінці результативності є візуалізація отриманих даних. Зазвичай це побудова відповідних графіків, будь то графік точності моделі, графік кількості помилок з епохами навчання, чи графік прогнозованих та реальних значень. Графіки дозволяють нам, людям, краще сприймати велику кількість інформації, та візуально оцінювати її, що вкрай важливо при оцінці та демонстрації результативності роботи моделей [16].

Інтерпретація результатів моделі передбачає аналіз ваг і важливості вхідних ознак, що допомагає зрозуміти, які фактори найбільше впливають на прогнозування цін акцій. Наприклад, розглядаючи ваги нейронної мережі або коефіцієнти лінійної регресії, можна визначити, які змінні мають найзначніший вплив на вихід моделі. Це може включати економічні показники, такі як ВВП, рівень безробіття, інфляція тощо, а також інші фактори, такі як новини про компанію, оголошення про прибутки, зміни керівництва чи регуляторні рішення. Аналіз цих факторів дозволяє не лише покращити точність моделі, але й надати більш глибоке розуміння того, які зовнішні та внутрішні чинники керують змінами цін на акції, що в свою чергу може бути корисним для

прийняття інвестиційних рішень або розробки стратегій ризик-менеджменту. Крім того, інтерпретація результатів може допомогти виявити потенційні аномалії або недоліки моделі, які можна врахувати для її подальшого вдосконалення.

Висновки до розділу 2

В розділі 2 цієї роботи було розглянуто теорію практичних методів у контексті теми дипломної роботи. Було проведено аналіз різноманітних аспектів, важливих при написанні практичної частини роботи, а також розглянуто більшу частину теоретичних рішень проблеми прогнозування цін та індексів.

В першій частині було розглянуто низку методів штучного інтелекту які найчастіше використовуються в подібних задачах прогнозування цін. Методи розподіли відповідно на статистичні методи, та нейронні мережі, навели коротку інформацію про кожен, розглянути принципи їх дії, принципи застосування, проаналізували сильні та слабкі сторони. Далі було розглянуто практичні підходи до підготовки даних та налаштування моделей. Було розглянуто такі важливі питання як вибір та підготовка датасетів для навчання та тестування подальших моделей, налаштування самих моделей, зокрема вибір гіперпараметрів, вибір оптимізатора та функції втрат а також те, чому ці кроки є вкрай важливими. На кінець було розглянуто практичні підходи до оцінки результативності моделей, та подальшої інтерпретації отриманих результатів, методи, які найчастіше використовуються для цього, а також важливість даних кроків в процесі аналізу моделей. Загалом розділ 2 надає важливу інформацію для подальшої практичної розробки програми прогнозування курсів акцій та індексів на фондових ринках. Містить найбільш важливі аспекти порівняння та вибору моделей, принципів їх дії, розробки, подальшого аналізу та тестування.

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ РОБОТИ

3.1 Пошук та попередня обробка датасетів

Як вже було зазначено в попередньому розділі, пошук, вибір та обробка даних відіграє важливу роль в тренуванні, та подальшому використанні різних моделей штучного інтелекту.

Почнемо з задачі пошуку необхідних даних. Для цього скористаємося сервісом Yahoo Finance, що надає історичні дані фондових ринків та ринку криптовалют. Однією з головних причин вибору сервісу Yahoo Finance та його API для отримання даних про історичні ціни акцій та індексів є висока надійність і точність наданої інформації. Yahoo Finance зарекомендував себе як один з провідних ресурсів для фінансових даних, що забезпечує доступ до великої бази даних, яка охоплює широкий спектр акцій, індексів та інших фінансових інструментів. Цей сервіс надає деталізовані дані, які можуть бути використані для ретельного аналізу ринку, створення фінансових моделей та прийняття обґрунтованих інвестиційних рішень. Крім того, Yahoo Finance пропонує інтерфейс API, який дозволяє автоматизувати процес отримання та обробки даних, що значно спрощує роботу фінансових аналітиків та розробників.

Ще однією важливою перевагою Yahoo Finance API є його доступність та простота використання. Інтерфейс API добре документований, що дозволяє швидко інтегрувати його у різноманітні програмні рішення. Зручність використання API дозволяє користувачам отримувати необхідні дані в реальному часі та швидко адаптуватися до змін на ринку. До того ж, Yahoo Finance надає безкоштовний доступ до основних функцій API, що теж є вкрай важливо. Ці фактори роблять Yahoo Finance API оптимальним вибором для отримання даних про історичні ціни акцій та індексів.

Для завантаження даних скористуємося бібліотекою yfinance, яка працює через вище вказане API. Щодо розміру датасету пропоную використовувати

проміжок в 10 років: з 1 січня 2014 року, до 1 січня 2024 року. Такого проміжку має бути більш ніж достатньо для тренування та подальшого оцінювання моделей. Далі оберемо які саме акції та індекси будуть брати участь в аналізі. Тут я пропоную вибрати для порівняння акції компаній з однієї сфери, так як на них в процесі впливають одні й ті самі події, а тому більша ймовірність знайти сходження в процесу тренування моделей. Вибір в моєму випадку впав на акції Google LCC (GOOG), Microsoft (MSFT) та Apple (AAPL). Що до індексів, для порівняння, пропоную взяти індекс NASDAQ100 як індекс з тієї само сфери що і обрані акції. Завантаживши дані одної з акцій отримуємо такий датасет:

Таблиця 3.1 – Приклад вхідного датасету акції GOOG

	Date	**Open**	**High**	**Low**	**Close**	**Adj Close**	**Volume**
0	1/2/2014	27.78236	27.83940	27.60303	27.72408	27.7240	73129082
1	1/3/2014	27.77090	27.81897	27.52009	27.52184	27.5218	66917888
2	1/6/2014	27.72134	27.86704	27.55770	27.82869	27.8286	71037271
3	1/7/2014	28.01997	28.38585	27.92433	28.36517	28.3651	102486711
4	1/8/2014	28.54301	28.57589	28.22645	28.42421	28.4242	90036218
…	...	...	...	...	...	...	...
2511	12/22/2023	142.1300	143.25	142.0549	142.72	142.72	18494700
2512	12/26/2023	142.98	143.9450	142.5	142.8200	142.820	11170100
2513	12/27/2023	142.83	143.3200	141.051	141.44	141.44	17288400
2514	12/28/2023	141.8500	142.27	140.828	141.28	141.28	12192500
2515	12/29/2023	140.6799	141.435	139.8999	140.9299	140.929	14872700

Розглянувши вхідний датасет отриманий через API Yahoo Finance можемо побачити багато різних даних, а саме дату торгів (Date), ціну відкриття торгів (Open), найвищу ціну за торговий день (High), найнижчу ціну за торговий день (Low), ціну закриття торгів (Close), середню ціну за торговий день (Adj Close) та об’єм торгів за день (Volume). Всі показники цін, а також об’єм торгів підходять для подальшого прогнозування, проте, як вже було сказано вище, для спрощення пропоную обрати ціну відкриття (Open) як основний показник який ми будемо прогнозувати, цього буде достатньо для оцінки моделей в задачах прогнозування цін.

Далі варто впевнитись, що отримані дані є якісними. Потрібно перевірити їх цілісність та наявність пропусків в даних, та за необхідності їх виправлення, для подальшого тренування та використання моделями. В нашому випадку можна сказати що у всіх обраних датасетах нема пропусків, відповідно отримані дані можна вважати якісними.

Далі перейдемо до підготовки цих даних для тренування та подальшого використання. Розпочнемо з розбиття даних та тренувальний та тестовий частини. Це робиться для перевірки натренованої моделі в схожій, проте не тій самій ситуації що і при тренуванні, що дає змогу зрозуміти її ефективність в реальних ситуаціях використання. Пропоную розбити датасет на 80% для тренування моделей, та інші 20% для тестування.

Останнім кроком в підготовці даних є їх нормалізація. Цей процес включає стандартизацію форматів даних, усунення можливих аномалій та приведення всіх значень до єдиного масштабу. Нормалізація даних забезпечує їх коректне порівняння та аналіз, підвищуючи точність і надійність результатів. Вона також допомагає зменшити вплив викривлень і сприяє кращому розумінню трендів і закономірностей. Багато моделей вимагають попередню нормалізацію вхідних даних, так як без цього вони не зможуть показати нормальні результати. В даній роботі ми скористємося методом MinMaxScaling з бібліотеки sklearn.preprocesing, який перенесе всі наші дані в межі значень від 0 до 1, де 0 буде саме низьке вхідне значення, а 1 - саме велике.

Тепер вхідні дані можна вважати готовими, та можемо приступити до наступних кроків.

3.2 Побудова та налаштування моделей

В цьому розділі розглянемо побудову деяких обраних моделей серед розглянутих в розділі 2. Серед всіх моделей описаних в вище зазначеному

розділі пропоную обрати моделі ARIMA, SARIMA, Simple RNN, LSTM та Трансформери. На мою думку саме ці моделі є найбільш перспективними, найкраще підходять для задачі прогнозування цін, та повинні були б показати найкращі результати.

Почнемо з статистичних методів ARIMA та SARIMA. Вони є найбільш простими в побудові, так як вимагають всього лише підібраного порядку моделі (order), що відповідає за певні її компоненти. Порядок підбирається методом проб та помилок на основі значень функції правдоподібності (Log Likelihood) та значень AIC/BIC. Так для ARIMA в ході підбору порядку було визначено порядок ARIMA(3, 3, 3) як найбільш продуктивного за вказаними показниками. Для моделі SARIMA ситуація схожа, проте на цей раз вже окрім звичайного порядку модель також вимагає сезонний порядок. Тож для моделі SARIMA був підібраний сезонний порядок SARIMA(1, 1, 1, 12). Такий порядок був обраний щоб виділити сезонний компонент моделі та спробувати сконцентруватися саме на ньому. Що до звичайного порядку, він залишився тим самим – (3, 3, 3). Обидві моделі присутні готовими в пакеті statmodels, що також спрощує їх імплементацію за застосування.

Тепер приступимо до нейронних моделей. Як вже було зазначено раніше, тут ми сконцентруємося на моделях Simple RNN, LSTM та трансформері. Даній нейронні мережі складаються з різних шарів, можуть мати їх різну кількість, відповідно розглянемо архітектуру кожної з них. Зазначимо, що всі шари з яких ми будемо будувати моделі взяті з бібліотеки Keras.

Розпочнемо з Simple RNN. Архітектуру даної моделі побудуємо з 3 послідовних рівнів, які складаються з шару SimpleRNN та Dropout. В першому такому рівні також зазначимо вхідний формат даних, щоб ініціювати його як вхідний шар. Закінчимо все шаром Dence. Що до гіперпараметрів, то використаємо по 64 вхідних одиниць та стандартним активатором tanh для кожного шару SimpleRNN, та по 0.2 вхідні одиниці для шарів Dropuout. При компіляції моделі застосуємо стандартний оптимізатор Adam, а в якості функції втрат застосуємо MSE. Також для подальшої оцінки моделі додамо метрики, які

будуть відстежуватися при навчанні, а саме: MSE, MAE, MSLE та RMSE. Архітектуру моделі зображено на рис. 3.1.

Рисунок 3.1 – Архітектура моделі Simple RNN

Далі розглянемо Long-Short Term Memory (LSTM) модель. В цьому випадку додамо 2 шари LSTM на 64 одиниці та залишимо стандартні активатори. Також для першого шару додамо вхідний формат до першого шару. Потім 2 шари Dence на 32 та 1 вхідні одиниці. Під час компіляції залишимо ті самі параметри: оптимізатор Adam, функція втрат MSE, та ті самі метрики. Архітектуру моделі зображено на рис. 3.2.

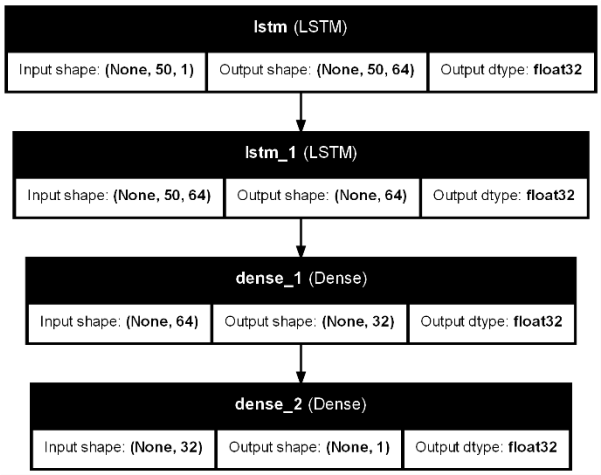

Рисунок 3.2 – Архітектура моделі LSTM

І остання модель – трансформер. В цій роботі пропоную держатися класичної архітектури для трансформера яка включає кодування та декодування. Для частини кодувальника ми побудуємо на основі подвійного зв’язку, який додає вихідні дані з механізму уваги до вхідних даних, що дозволяє легше проходити градієнтам через довгі послідовності шарів. Для цього додамо такі шари: шар нормалізації LayerNormalization, шар скалярно-добуткової уваги MultiHeadAttention та шар Dropout. Результат цих шарів і додамо до вхідних даних. Після друга частина кодувальника – знову шар нормалізації LayerNormalization, шар Dence, шар Dropout та ще один вихідний шар Dence. Після такого кодування отримані дані пропускаються через додаткові шари для отримання кінцевих результатів. Тут використаємо шар GlobalAvaragePooling1D, для зменшення просторових розмірностей даних, шар Dropout, шар Dence, та ще один, вже вихідний шар Dence. Що до компіляції цієї моделі також залишаємо описані в моделі SimpleRNN параметри: оптимізатор Adam, MSE в якості функції втрат, та ті самі метрики для подальшого оцінювання тренування. Архітектуру отриманого трансформера зображено на рис. 3.3.

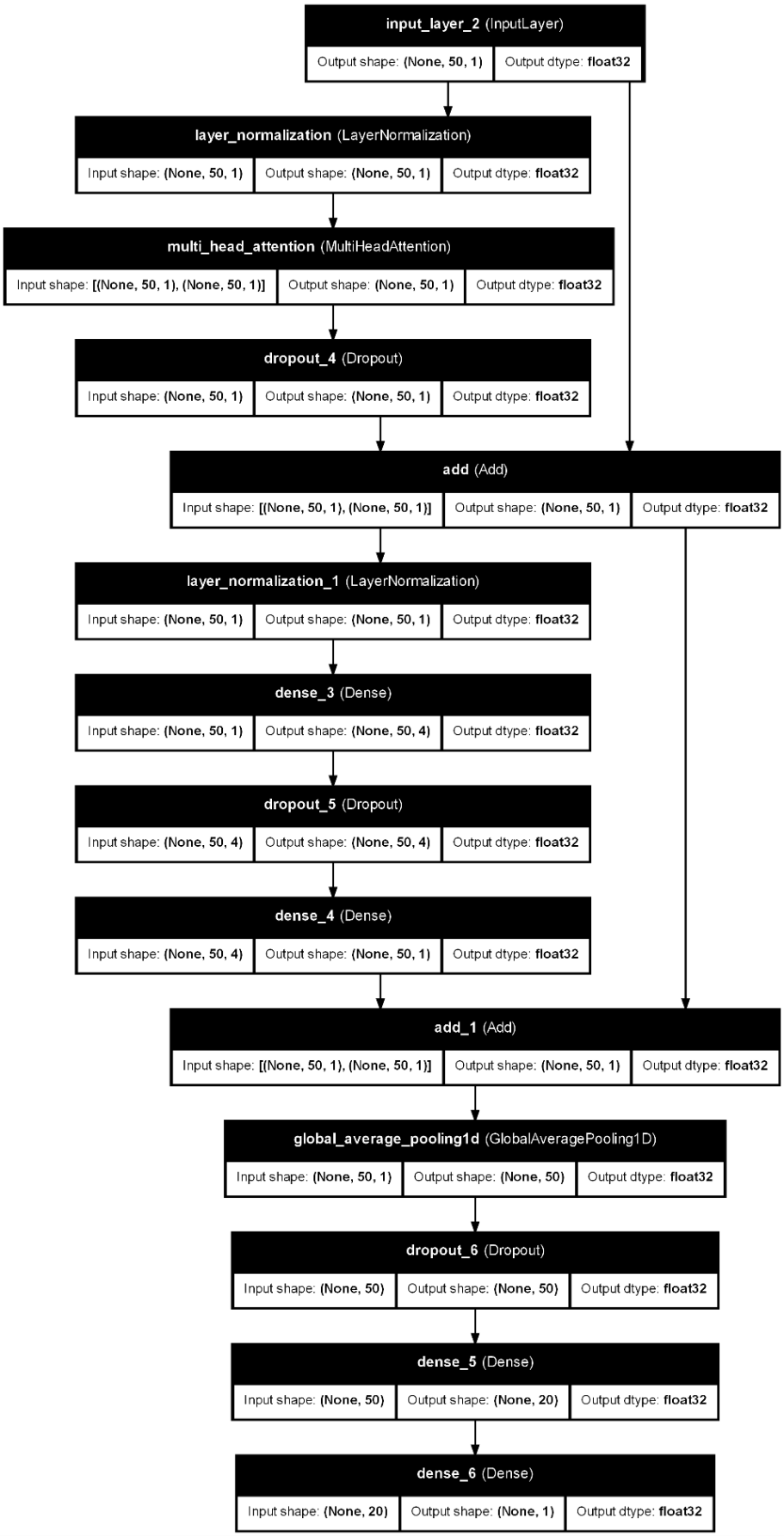

Рисунок 3.3 – Архітектура моделі трансформ ера

3.3 Тренування моделей

В цьому розділі розглянемо процес тренування моделей. Одразу варто зазначити, що для статистичних методів таких як ARIMA та SARIMA процес тренування не подібний до тренування нейронних мереж, або яких небудь інших моделей машинного навчання. Замість тренування статистичні методи використовують ітеративні алгоритми на основі обраних показників для вибору параметрів та безпосереднього прогнозу майбутніх значень. Тому одразу перейдемо до процесу тренування нейронних мереж.

Процес тренування нейронних мереж являє собою її навчання на створеному раніше тренувальному датасеті, який складається з 80% загальної вибірки в нашому випадку, із використанням вибраних раніше оптимізатора та функції втрат. Отримавши вхідні дані модель коригує свої вагові коефіцієнти на основі отриманої помилки з функції втрат, в нашому випадку у всіх трьох моделях це MSE. Процес навчання повторюється протягом обраної кількості епох, де кожна означає проходження цілого тренувального набору. Проте тут пропоную застосовувати callback з бібліотеки keras.utilitis – EarlyStopping, який дозволяє моделі, при відсутності покращення результатів помилки, дочасно завершити своє навчання, не проходячи інші епохи. Також, додатково, пропоную заміряти час навчання моделей, як один з параметрів для подальшого їх порівняння.

По завершенню такого навчання ми отримуємо навчені моделі готові до використання. Результати корекції функції втрат при навчанні по MSE для обраних нейронних моделей зображені на рис. 3.4 - 3.6.

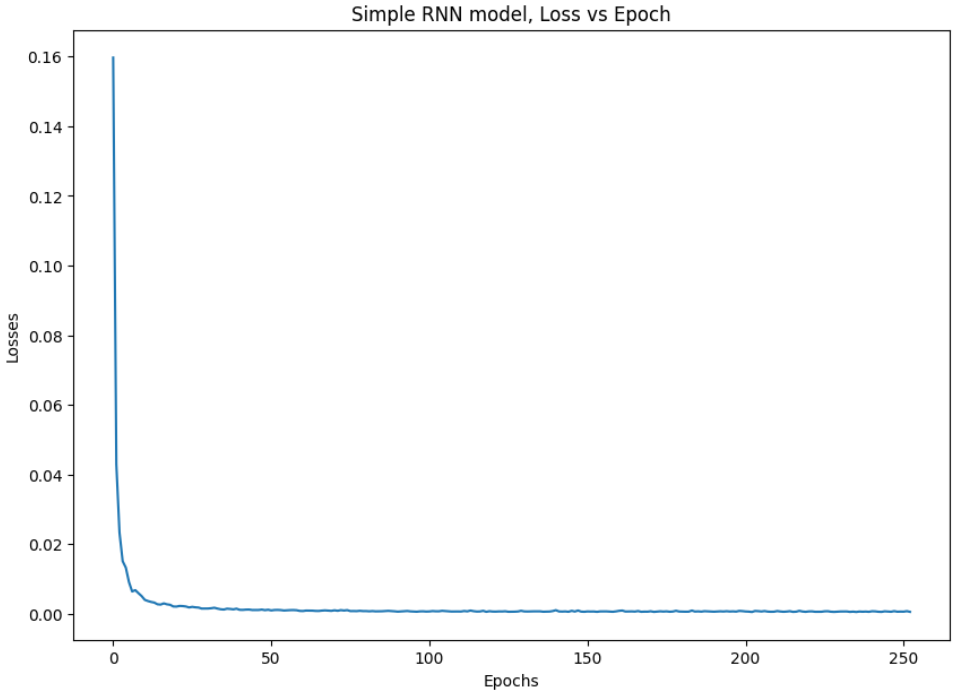

Рисунок 3.4 – Результати функції втрат моделі Simple RNN

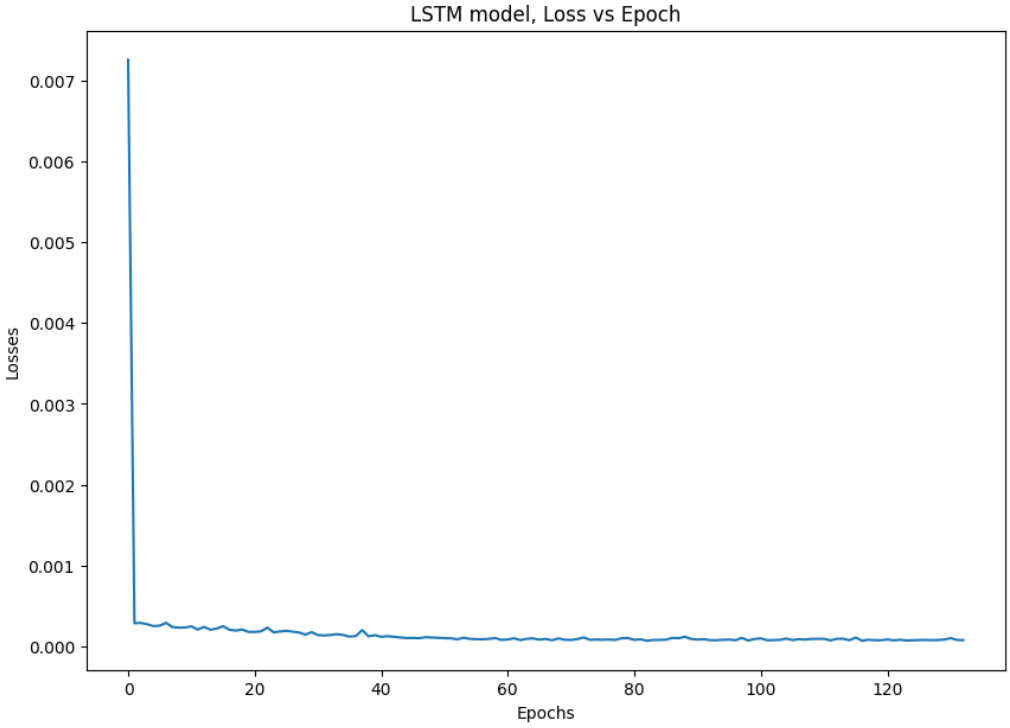

Рисунок 3.5 – Результати функції втрат моделі LSTM

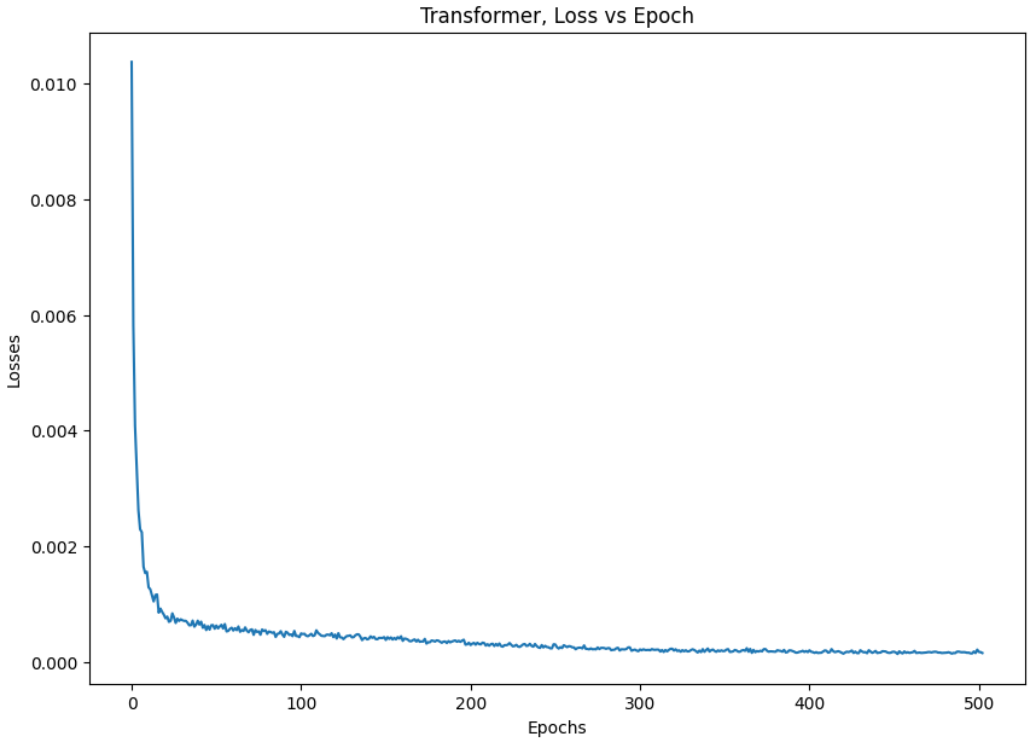

Рисунок 3.6 – Результати функції втрат моделі трансформера

На рисунках можемо побачити, що з епохами значення функції втрат постійно падає, що говорить про те, що процес навчання проходить добре.

3.4 Результати дослідження моделей та їх порівняння

Отримавши натреновані моделі можемо приступити до аналізу їх результативності в задачах прогнозування обраних акцій та індексів. Як було зазначено раніше, перевіряти моделі будемо на тестувальних 20% датасету, щоб перевірити результативність моделі на ще невідомих їх даних. Також додатково для кожної обраного активу візьмемо датасет довжиною в 3 та 10 років, та дослідимо ефективність навчання та прогнозування у обох випадках.

Розпочнемо з тренування моделей, а саме з часу який їм потрібен на цей процес. Як вже було зазначено вище, ми використовуємо callback EarlyStopping,

зі значенням patience 50, що є досить високим значенням, яке означає, що модель зупинить тренування як тільки напротязі 50 епох не буде суттєвих покращень обраної метрики, в нашому випадку функції втрат по MSE. В таблиці 3.1 показані результати по обраним датасетам в секундах.

Таблиця 3.1 – Результати часу навчання моделей нейронних мереж

	NASDAQ100	AAPL	GOOG	MSTF
RNN	112.076	142.5118	166.85	124.3093
LSTM	191.1039	349.7947	292.4977	341.0159
Трансформер	245.9647	211.8061	237.3722	300.8668

З таблиці можемо бачити, що LSTM для акцій традиційно потребує більше часу на навчання, на другому місці можемо виділити модель Трансформера а Simple RNN потребує найменше часу на навчання. Трішки ситуація відрізняється для індексу NASDAQ100, де найбільше часу потребувалося моделі трансформера, а LSTM справилась доволі швидко, Simple RNN залишилась найшвидшою моделлю. Загалом результати є очікуваними, і відображають специфіку роботи моделей. Також варто згадати за статистичні методи, хоча вони і не потребують навчання, проте можемо виділити час їхньої підгонки (model fitting), який складає близько 3-5 секунд для моделі ARIMA, та дещо довше – 7-10 секунд для моделі SARIMA, на датасетах розміром в 10 років. І хоча статистичні методи майже не займають часу на навчання, вони в перспективі займають велику кількість часу для самого прогнозування, на відміну від моделей нейронних мереж, які хоча і навчаються певний час, майже не потребують часу для самого прогнозування, та можуть повертати результати доволі швидко.

Далі розглянемо показники моделей при прогнозуванні. Для цього використаємо функції з sklearn.metrics для відповідних метрик. Розглянемо метрики на прикладі прогнозування тестувального датасету акції GOOG. Результати метрик показані в таблиці 3.2.

Таблиця 3.2 – Результати метрик прогнозування для GOOG 10 років

Model	MSE	MAE	MSLA	RMSE
ARIMA	0.02793543	0.117238453	2.06E-06	0.167138953
SARIMA	0.090634698	0.236001147	7.08E-06	0.301055971
SimpleRNN	8.335444446	2.261351562	0.000635497	2.887116978
LSTM	5.566176655	1.826952263	0.000434457	2.359274604
Transformer	25.99428613	4.297199376	0.001973876	5.098459192

За метрикою MSE, яку ми використовували як функцію втрат при навчанні, чим менше значення, тим краща точність передбачень, відповідно бачимо найменше значення в моделі ARIMA, трішки гірше в SARIMA, моделі RNN та LSTM справилися непогано, а в трансформера – найгірший результат. Метрики MAE та RMSE вказують на середню абсолютну похибку та середню велечину похибки. Вони показують схожі результати з MSE. Так само і MSLA, яка вказує на похибки якщо в даних присутні експоненційні зміни.

Тепер пропоную розглянути дані для датасету GOOG на 3 роки. Результати метрик зображені на таблиці 3.3.

Таблиця 3.3 – Результати метрик прогнозування для GOOG 3 років

Model	MSE	MAE	MSLA	RMSE
ARIMA	0.06191	0.201264	3.58E-06	0.248818
SARIMA	0.130201	0.287673	7.45E-06	0.360833
SimpleRNN	7.602329	2.057754	0.000422	2.757232
LSTM	4.959349	1.729418	0.000273	2.22696
Transformer	11.59388	2.739873	0.000645	3.404979

Бачимо дещо гірші значення показують статистичні методи ARIMA та SARIMA, та дещо кращі результати в нейронних моделей RNN, LSTM та трансформера. Це демонструє дещо кращу роботу нейронних мереж в межах невеликих наборах даних, та кращу роботу з короткостроковими патернами, а статистичні методи – навпаки, краще працюють з великими обсягами даних. Схожу картину бачимо і з іншими наборами даних на таблицях 3.4 та 3.5.

Таблиця 3.4 – Результати метрик прогнозування для AAPL

Model	MSE	MAE	MSLA	RMSE
AAPL 10 років				
ARIMA	0.058973	0.181827	2.32E-06	0.242844
SARIMA	0.091666	0.233634	3.65E-06	0.302764
SimpleRNN	11.39083	2.546402	0.00046	3.37503
LSTM	8.224866	2.189366	0.000336	2.867903
Transformer	44.22786	5.479308	0.001789	6.650403
AAPL 3 роки				
ARIMA	0.044013	0.171557	1.25E-06	0.209792
SARIMA	0.093164	0.255908	2.75E-06	0.305227
SimpleRNN	9.845452	2.395469	0.000302	3.137746
LSTM	5.989899	1.735306	0.000183	2.447427
Transformer	27.79208	4.617122	0.000845	5.271819

Таблиця 3.5 – Результати метрик прогнозування для NASDAQ100

Model	MSE	MAE	MSLA	RMSE
NASDAQ100 10 років				
ARIMA	461.0354	15.94195	2.61E-06	21.47173
SARIMA	364.681	14.78923	1.98E-06	19.09662
SimpleRNN	60821.08	191.0951	0.000363	246.6193
LSTM	42582.81	161.049	0.000265	206.356
Transformer	260723.2	425.9718	0.001513	510.6106
NASDAQ100 3 роки				
ARIMA	293.7271	12.00521	1.29E-06	17.13847
SARIMA	322.8176	15.09201	1.38E-06	17.96713
SimpleRNN	48801.25	180.0302	0.000214	220.91
LSTM	27010.91	128.6777	0.000118	164.35
Transformer	86687.17	251.0611	0.000366	294.4269

Варто зазначити, що сильне зростання значень метрик зумовлено великим діапазоном цін індексу протягом обраних років (до прикладу, від ≈3500 до ≈17000 пунктів в датасеті в 10 років), відповідно реальні значення, та прогнозовані, в певних випадках, могли відрізнятися на десятки та навіть сотні пунктів. Якщо ж дивитись відношення цих метрик одна до одної, то спостерігаємо ту саму картину що і з акціями.

На кінець розглянемо графіки прогнозованих значень та реальних. Для початку знову пропоную подивитись на графіки для акції AAPL, де прогнози

робилися на основі датасету 10 років. На графіку ми можемо бачити ту саму ситуацію що і описану раніше на основі метрик. Статистичні моделі ARIMA та SARIMA залишаються найкращими, після них добре себе показують Simple RNN та LSTM, остання дещо краще за попередню, та на кінець, трансформер, який справляється з поставленою задачею найгірше.

Графік зображено на рис. 3.7.

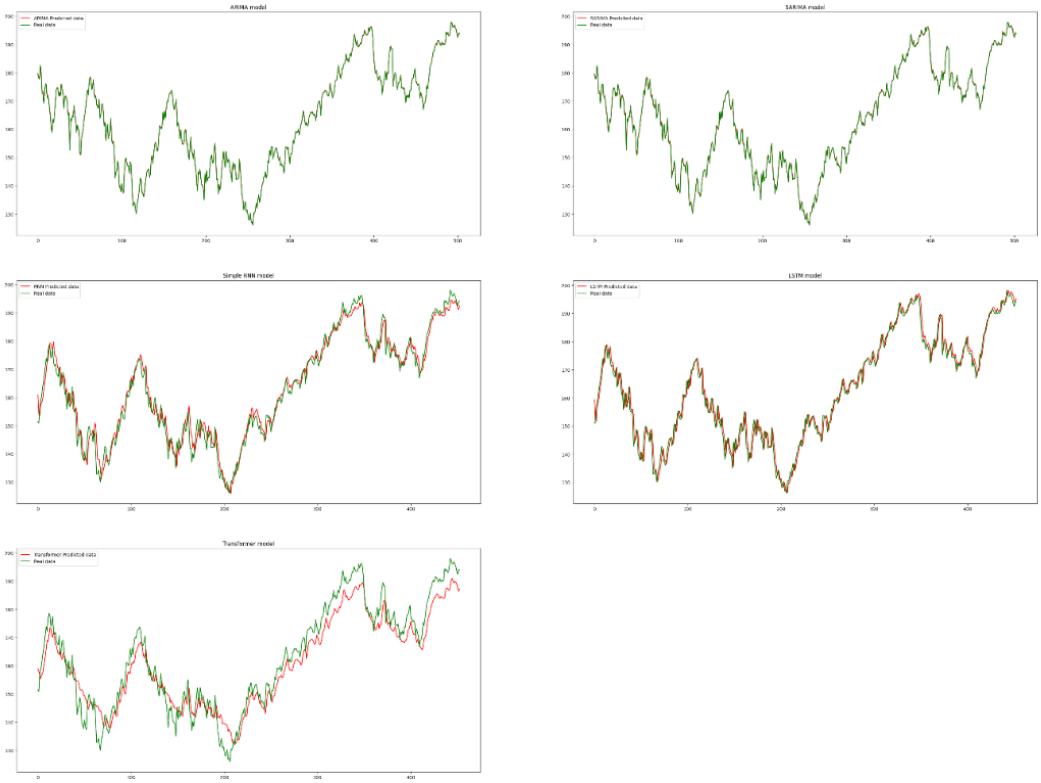

Рисунок 3.7 – Графіки прогнозів та реальних значень акцій AAPL

Так як графіки дають лише можливість візуалізувати процес, але не точно передати інформацію про цифри та дрібні незбіжності, сенсу демонструвати графіки для датасету в 3 роки нема, так як суттєвих відмінності між ними та графіками на 10 років майже нема. Тому вважаю за достатнє продемонструвати інші графіки для акцій та індексів на 10 років, щоб впевнитись в збережені тенеденцій отриманих з попередніх описаних даних. Ці графіки зображені на рис. 3.8 – 3.10.

Рисунок 3.8 – Графіки прогнозів та реальних значень акцій MSFT

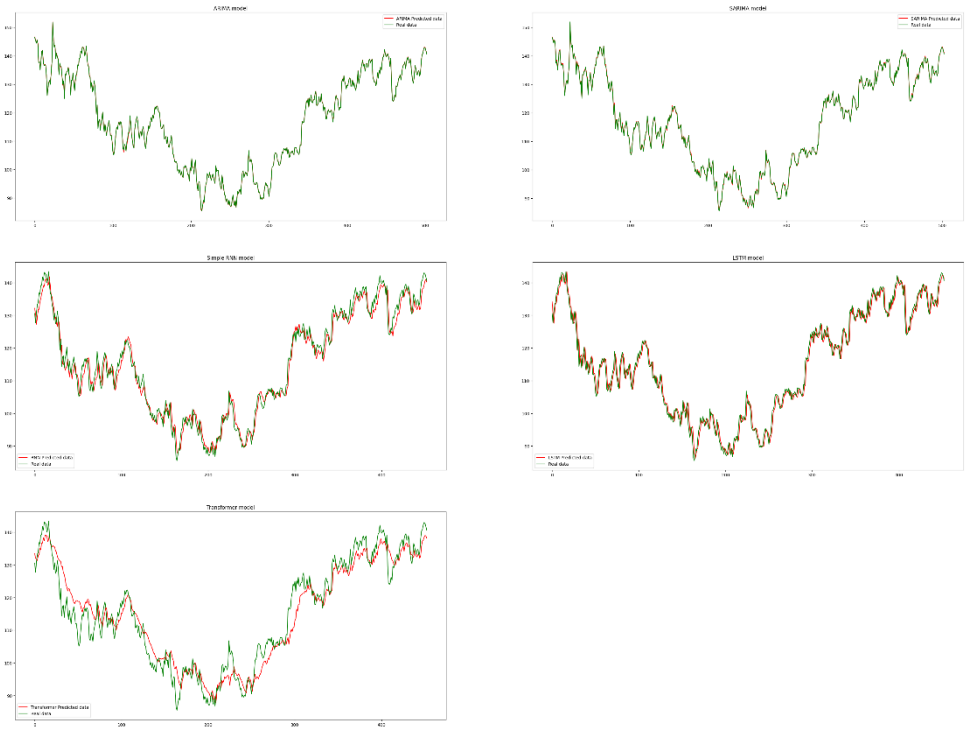

Рисунок 3.9 – Графіки прогнозів та реальних значень акцій GOOG

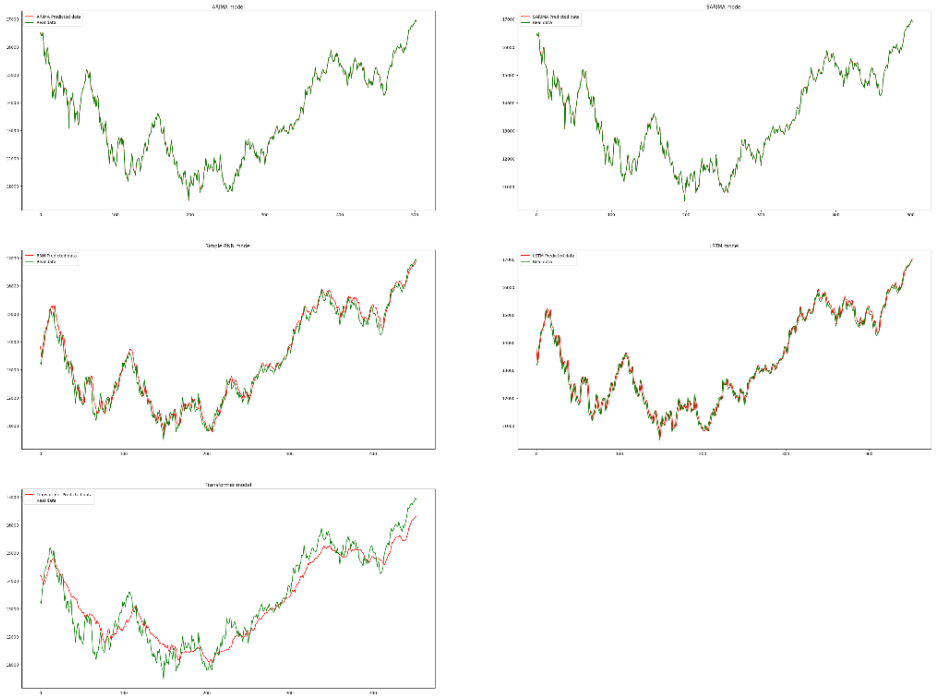

Рисунок 3.10 – Графіки прогнозів та реальних значень акцій NASDAQ100

Загалом, на основі отриманих даних можемо підсумувати, що найкраще з поставленою задачею в даних умовах справилася модель ARIMA, дещо гірше, в силу відсутності помітної сезонності, модель SARIMA. Що до нейронних мереж, то вони справилися гірше, проте їхня результативність все ще помітна. Модель LSTM виявилася найкращою серед нейронних мереж і показала доволі точні результати, далі модель Simple RNN, яка показала дещо гірші, проте все ще непогані результати. Найгіршою виявилося модель трансформера, який виявився найменш результативним у всіх сценаріях. Також варто відмітити вплив розміру вибірки на результати моделей, де більша вибірка давала кращі результати для статистичних методів, та навпаки, менша давала краще себе розкрити нейронним мережам.

Висновки до розділу 3

В цьому розділі було описано реалізацію програмного продукту для дослідження моделей штучного інтелекту в задачах прогнозування курсів акцій та індексів, а саме статистичних методів ARIMA та SARIMA, а також нейронних мереж Simple RNN, LSTM та Трансформера.

В процесі було розглянуто вибір, збір та попередню обробку даних, описано побудову моделі та їх архітектуру, процес їх навчання а також проведено аналіз їх результативності та порівняння між собою.

Загалом програмний продукт справився зі своєю задачею, та продемонстрував всі вищеописані процеси. Можна заключити, що всі моделі так чи інакше справилися з задачею прогнозування курсів акцій та індексів, окрім моделі трансформера, яка виявилася недостатньо точною, по показала погану результативність в даній задачі та поставлених умовах.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО ВАРСТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на дослідженні демографічного стану.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є такими:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити такі.

1. F_1 – вибір самої програми.
2. F_2 – якісний аналіз даних.
3. F_3 – графічні показники.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1.

a) Keras.

b) TensoreFlow.

Функція F_2.

a) Застосування вбудованих функцій.

b) Створення своїх гнуч моделей

Функція F_3.

a) Використання шаблонних графіків.

b) Створення своїх.

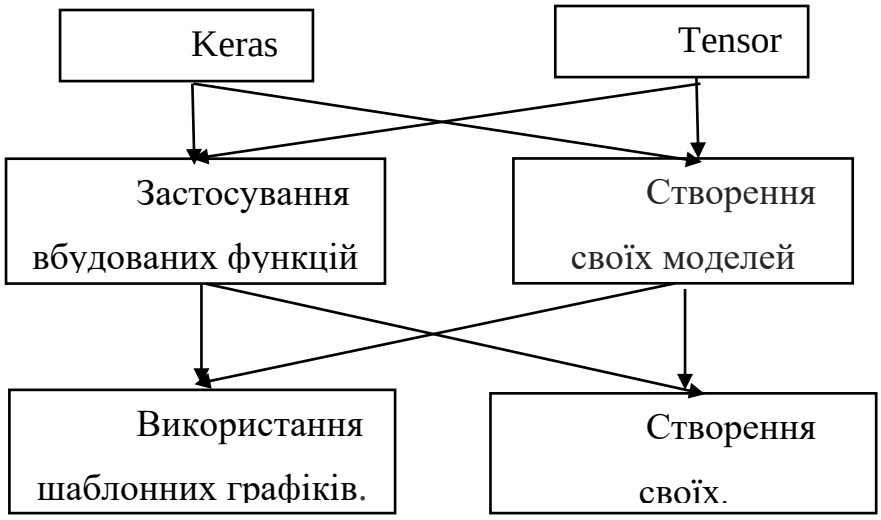

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантиів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 – Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Популярна бібліотека для глибокого навчання з великою спільнотою розробників, багато ресурсів і документації	Можливі певні складнощі у розумінні деяких концепцій і налаштуваннях.
	Б	Бібліотека для глибокого навчання з широкими можливостями та великою кількістю інформаційних ресурсів	Більша складність написання, налаштування, та застосування методів, необхідність знань з різних доточних сфер
F_2	*А*	Доступність та легкість при написанні	Іноді не відповідає задачі яку треба розв’язати
	Б	Гнучкість при описі усіх необхідні характеристики	Достатньо затратно реалізовувати свої алгоритми для подальшої реалізації
F_3	*А*	Загально прийнята реалізація	Іноді не відповідає очікуваним значенням
	Б	При виконанні власних досліджень краще може передавати висновки	Необхідно достатньо багато часу для написання програми для побудови та знаходження всього необхідного в задачі.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1.Перевагу даємо загальнодоступності. Для спрощення роботи по написанню коду варіант Б має бути відкинутий. Функція F_2. Програма допускає обрання обох варіантів. Можливо використати варіанти А чи Б. Функція F_3. Реалізація першого варіанту є сприйнятливою для програми, та пришвидшить написання роботи. Це варіант А.

Таким чином, будемо розглядати такий варіанти реалізації ПП: F_1а – F_2а – F_3а, F_1а – F_2б – F_3а.

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня. Для того, щоб охарактеризувати програмний продукт, будемо використовувати такі параметри:

Для того, щоб охарактеризувати програмний продукт, будемо використовувати такі параметри:

- *X1* – швидкодія мови програмування;
- *X2* – Точність моделі;
- *X3* – час попередньої обробки даних;
- *X4* – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	10000	15000	20000
Точність моделі	X2	%	75	90	97
Час попередньої обробки даних	X3	мс	368	276	159
Потенційний об’єм програмного коду	X4	кількість рядків коду	700	400	200

За даними таблиці 4.3 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

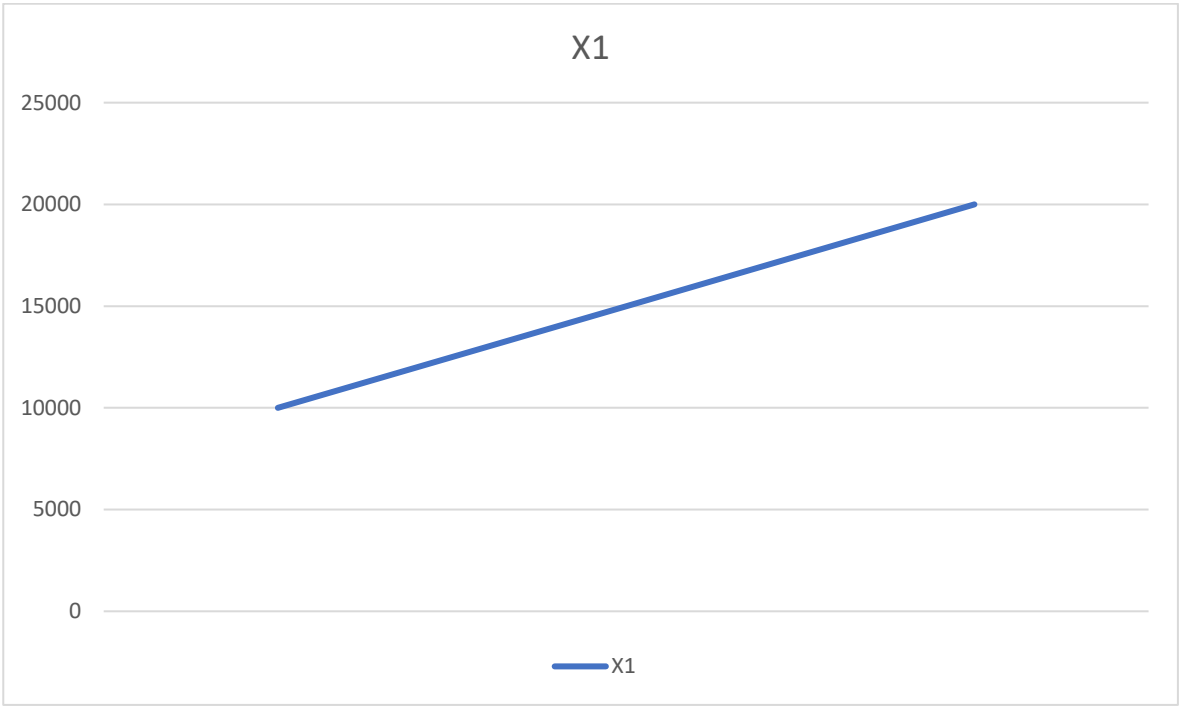

Рисунок 4.2 – X1, швидкодія мови програмування

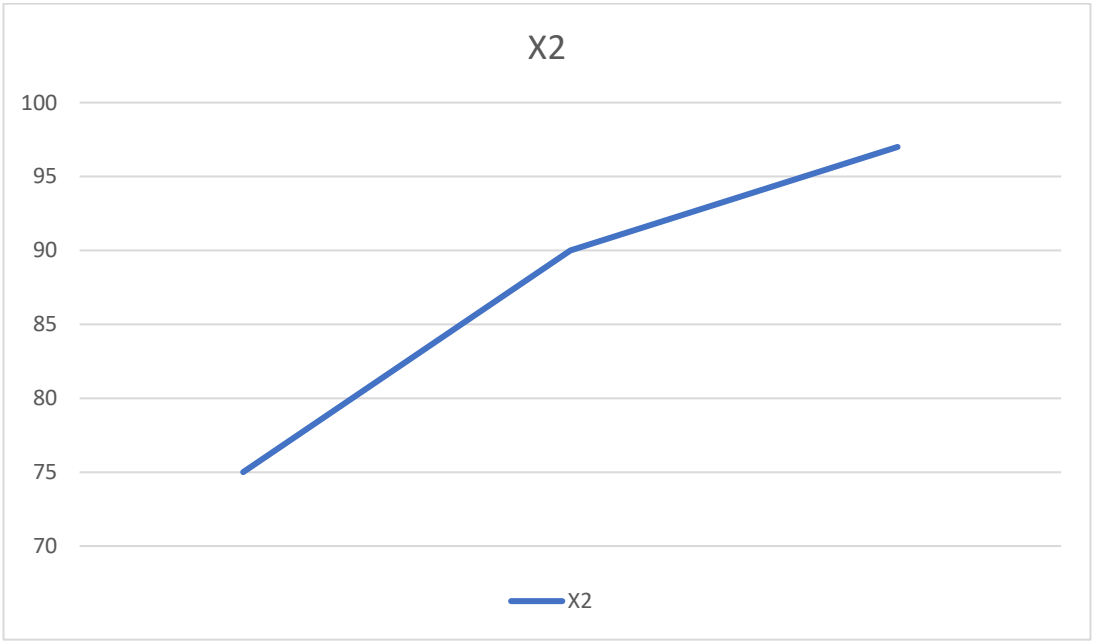

Рисунок 4.3 – X2, Точність моделі

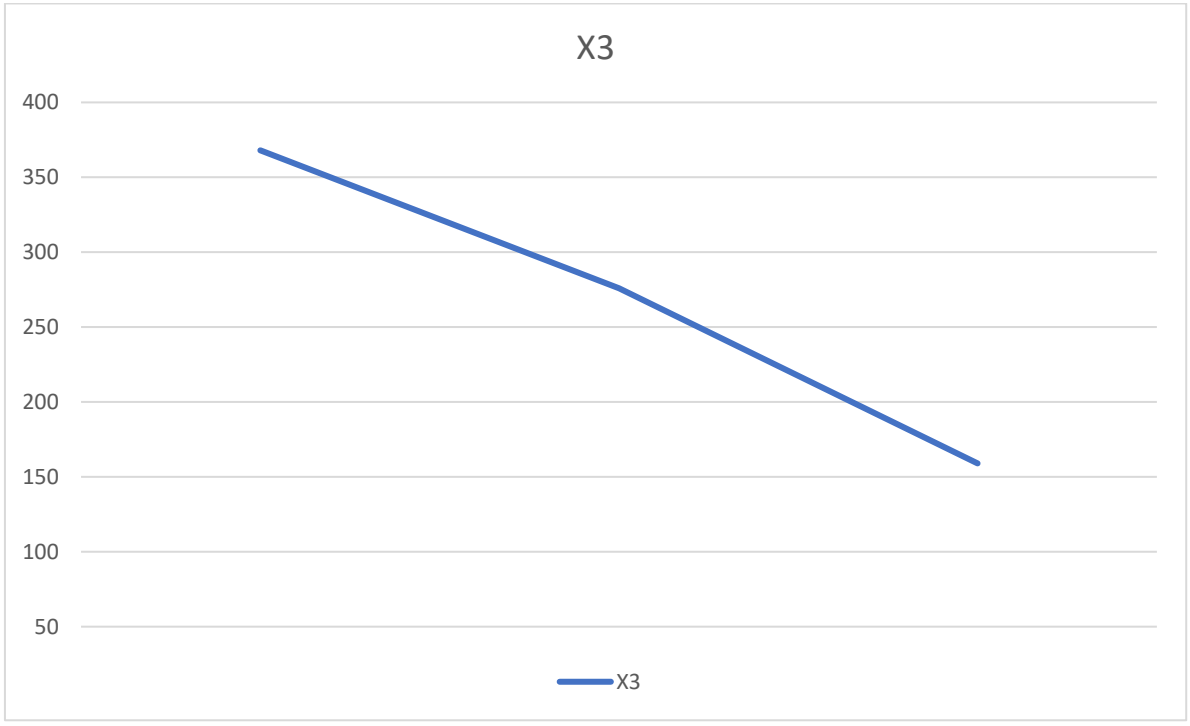

Рисунок 4.4 – X3, час попередньої обробки даних

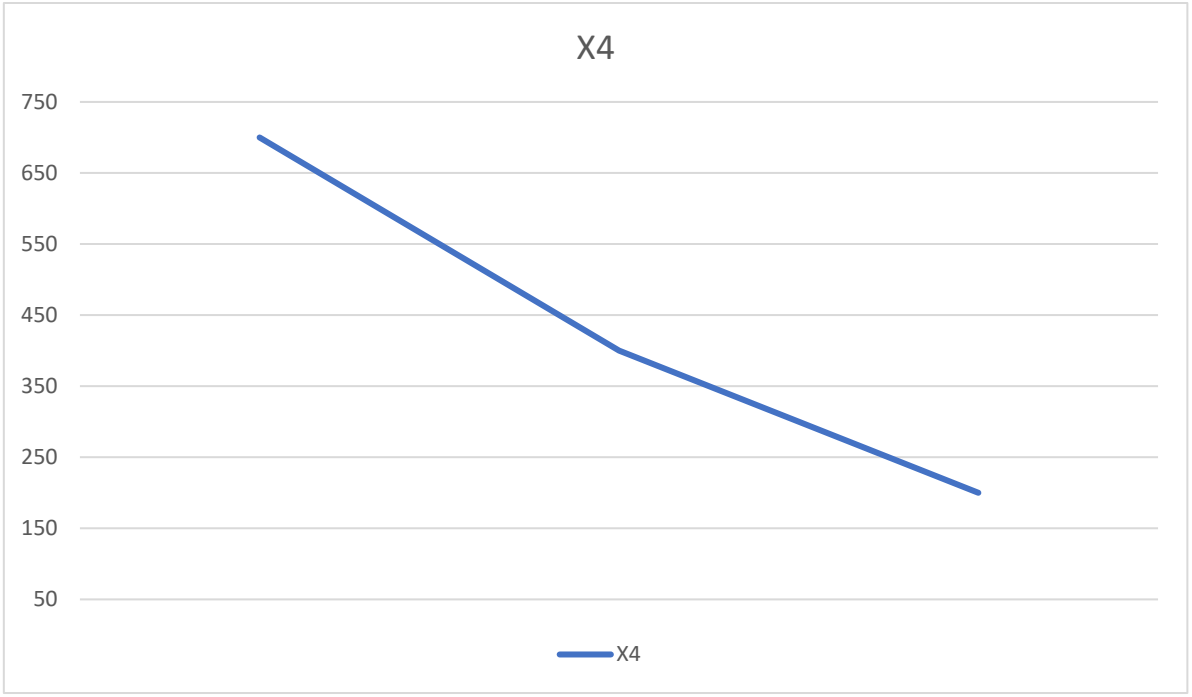

Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	3	4	2	3	2	3	4	21	3.5	12,25
X2	Точність моделі	%	1	2	1	1	1	1	2	9	-8,5	72,25

Закінчення таблиці 4.3

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхи-лення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X3	Час попередньої обробки даних	мс	2	1	3	2	3	2	1	14	-3,5	12,25
X4	Потенційний об'єм програмного коду	Кількість рядків коду	4	3	4	4	4	4		26	8,5	72,25
	Разом		0	0	0	0	0	0	0	70	0	169

Для перевірки степені достовірності експертних оцінок, визначимо такі параметри:

- сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів;

n – кількість параметрів.

- середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5.$$

- відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T.$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

-загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 169.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 169}{7^2(4^3 - 4)} = 0{,}69 > W_k = 0{,}67.$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	>	>	>	>	>	>	1,5
X1 і X3	>	>	<	>	<	>	>	>	1,5
X1 і X4	<	>	<	<	<	<	>	<	0,5
X2 і X3	<	>	<	<	<	<	>	<	0,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}.$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за такими формулами:

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i}.$$

$$b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за такими формулами:

$$K_{\text{вi}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}.$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{вi}}$	b_i^1	$K_{\text{вi}}^1$	b_i^2	$K_{\text{вi}}^2$
X1	1	1,5	1,5	0,5	4,5	16,25	0,28	59,125	0,28	16,25
X2	0,5	1	0,5	0,5	2,5	0,16	9,25	0,16	34,125	0,16
X3	0,5	1,5	1	0,5	3,5	0,22	12,25	0,21	41,875	0,2
X4	1,5	1,5	1,5	1	5,5	0,34	21,25	0,35	77,875	0,36
Всього:					16	1	59	1	213	1

4.5 Аналіз рівня варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (точність моделі), *X3* (час попередньої обробки даних) та *X4* (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} ,$$

де n – кількість параметрів;

$K_{вi}$ – коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	100	25	0,28	7
F3	А	X2	87	29	0,36	10,44
	Б	X3	27	19	0,2	3,8
F4	А	X4	25	23	0,16	3,68

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 7 + 10{,}44 + 3{,}68 = 21{,}12 ;$$

$$K_{K2} = 7 + 3{,}8 + 3{,}68 = 14{,}48 .$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання.

1. Розробка проекту програмного продукту;
2. Розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_О = T_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М},$$

де $T_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_Р$ =37 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.8. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 37 \cdot 1.8 \cdot 0.9 = 59{,}94 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =29 людино-днів, $К_П$ =0.9, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$T_2 = 29 \cdot 0.9 \cdot 0.8 = 20.88 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (59{,}94 + 20.88 + 4.8 + 20.88) \cdot 8 = 852 \text{ людино-годин.}$$

$$T_{II} = (59{,}94 + 20.88 + 6.91 + 20.88) \cdot 8 = 868{,}88 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 17000 грн., один аналітик в області даних з окладом 19000. Визначимо середню зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн.,}$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{17000 + 17000 + 19000}{3 \cdot 21 \cdot 8} = 105{,}16 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_ч \cdot T_i \cdot К_Д ,$$

де $С_ч$– величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 105.16 \cdot 852 \cdot 1.2 = 107515{,}58$ грн.

II. $С_{ЗП} = 105.16 \cdot 868.88 \cdot 1.2 = 109645{,}7$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 107515,58 \cdot 0.22 = 23653,4$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 109645,7 \cdot 0.22 = 24122,06$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 17000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 17000 \cdot 0,2 = 40800 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 40800 \cdot (1 + 0.2) = 48960 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 48960 \cdot 0,22 = 10771,2 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.4 \cdot 0.25 \cdot 10000 = 3500 \text{ грн.},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.4 \cdot 10000 \cdot 0.08 = 1120 \quad \text{грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.35 = 627,2 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;
$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;
$Д_Р$ – кількість днів планових ремонтів устаткування;
t –кількість робочих годин в день;
$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 627,2 \cdot 0,2 \cdot 0,3 \cdot 5,23 = 196,82 \quad \text{грн.},$$

де $N_С$ – середньо-споживча потужність приладу;
$К_З$– коефіцієнтом зайнятості приладу;
$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 10000 \cdot 0,67 = 6700 \quad \text{грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_{А} + С_{Р} + С_{ЕЛ} + С_{Н},$$

$С_{ЕКС}$ = 48960 + 10771,2 + 3500 + 1120 + 196,82+ 6700 = 72048,02 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М\text{-}Г} = С_{ЕКС}/ Т_{ЕФ}$ = 72048,02 / 627,2 = 115,01 грн/год.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_{М} = С_{М-Г} \cdot Т,$$

I. $С_{М}$ = 115,01 · 852 = 97933,52 грн.

II. $С_{М}$ = 115,01 · 868,88 = 99992,57 грн.

Накладні витрати складають 67% від заробітної плати:

$$С_{Н} = С_{ЗП} \cdot 0{,}67,$$

I. $С_{Н}$ = 107515,58 · 0,67 = 72035,45 грн.

II. $С_{Н}$ = 109645,70 · 0,67 = 73462,6 грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_{М} + C_{Н},$$

I. $C_{ПП}$ = 107515,58 + 23653,4 + 97933,52 + 72035,45 = 300138.95 грн.

II. $C_{ПП}$ = 109645,7 + 24122,06 + 99992,57 + 73462,6 = 307222.93 грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{Кj} / C_{Фj},$$

$$К_{ТЕР1} = 21,12 / 300138.95 = 7,038 \cdot 10^{-5},$$

$$К_{ТЕР2} = 14,48 / 307222.93 = 4,711 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}= 7,038 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 7,038 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

– вибір програмного продукту – Keras;

- реалізація важливої постановки з допомогою вбудованих функцій;
- використання стандартного інтерфейсу для побудови значень.

Цей варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

Висновки до розділу 4

В даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У ході виконання дипломної роботи було проведено дослідження та порівняно ефективність різних моделей штучного інтелекту у задачах прогнозування курсів акцій та індексів на фондових ринках.

Було наведено різноманітні статистичні методи та моделі нейронних мереж для вирішення поставленої задачі, проведено їх докладний аналіз та порівняння для виявлення найперспективніших моделей. Після чого, для проведення самого дослідження їх результативності було реалізовано програмний продукт на мові Python, з використанням таких бібліотек як Keras, TensoreFlow, Pandas, NumPy, Sklearn та MatplotLib.

В ході виконання роботи було проаналізовано та порівняно такі моделі як ARIMA та SARIMA зі статистичних методів та моделі Simple RNN, LSTM та трансформер. В результаті дослідження виявили найраші моделі – ARIMA та SARIMA.

Дослідження проводилося на різних датасетах, які включали дані за 3 та 10 років, різні акції, такі як AAPL, GOOG та MSFT, а також індекс NASDAQ100, для порівняння моделей в самих різних умовах. Загалом моделі показали свою ефективність на цих датасетах, та готові для роботи з іншими подібними даними.

ПЕРЕЛІК ПОСИЛАНЬ

1. Hayes A. Financial Markets: Role in the Economy, Importance, Types, and Examples. URL: https://www.investopedia.com/terms/f/financial-market.asp#toc-understanding-the-financial-markets (дата звернення: 22.05.2024).
2. The Evolution of Stock Exchanges. URL: https://www.investopedia.com/articles/07/stock-exchange-history.asp (дата звернення: 22.05.2024).
3. Fundamental vs. Technical Analysis: What's the Difference?. URL: https://www.investopedia.com/ask/answers/difference-between-fundamental-and-technical-analysis/ (дата звернення: 23.05.2024).
4. Artificial intelligence in the stock market: how did it happen?. URL: https://business.fiu.edu/academics/graduate/insights/posts/artificial-intelligence-in-the-stock-market-how-did-it-happen.html (дата звернення: 25.05.2024).
5. What are the differences between a moving average and autoregressive model?. URL: https://www.linkedin.com/advice/0/what-differences-between-moving-average-autoregressive-9yh0e (дата звернення: 25.05.2024).
6. Time Series Models. URL: https://towardsdatascience.com/time-series-models-d9266f8ac7b0 (дата звернення: 25.05.2024).
7. ARIMA & SARIMA: Real-World Time Series Forecasting. URL: https://neptune.ai/blog/arima-sarima-real-world-time-series-forecasting-guide (дата звернення: 26.05.2024).
8. What Is Time-Series Forecasting?. URL: https://www.timescale.com/blog/what-is-time-series-forecasting/ (дата звернення: 26.05.2024).
9. How to Develop Convolutional Neural Network Models for Time Series Forecasting - MachineLearningMastery.com. URL: https://machinelearningmastery.com/how-to-develop-convolutional-

neural-network-models-for-time-series-forecasting/ (дата звернення: 28.05.2024).

10. Foy P. Recurrent Neural Networks (RNNs) and LSTMs for Time Series Forecasting. URL: https://blog.mlq.ai/rnn-lstm-time-series-forecasting-tensorflow/ (дата звернення: 31.05.2024).
11. What are the advantages and challenges of using LSTM or GRU for long-term dependencies in time series?. URL: https://www.linkedin.com/advice/1/what-advantages-challenges-using-lstm-gru-long-term (дата звернення: 31.05.2024).
12. Stock market index prediction using deep Transformer model. URL: https://www.sciencedirect.com/science/article/abs/pii/S0957417422013100 (дата звернення: 31.05.2024).
13. Keylabs. Finding the best training data for your AI model. URL: https://keylabs.ai/blog/finding-the-best-training-data-for-your-ai-model/ (дата звернення: 31.05.2024).
14. What is hyperparameter tuning? | domino data lab. URL: https://domino.ai/data-science-dictionary/hyperparameter-tuning (дата звернення: 01.06.2024).
15. Guidelines for choosing an optimizer and loss functions when training neural networks. URL: https://odsc.medium.com/guidelines-for-choosing-an-optimizer-and-loss-functions-when-training-neural-networks-efcf97b7aa2 (дата звернення: 01.06.2024).
16. Model interpretation strategies. URL: https://towardsdatascience.com/explainable-artificial-intelligence-part-2-model-interpretation-strategies-75d4afa6b739 (дата звернення: 01.06.2024).

ДОДАТОК А КОД ПРОГРАМНОГО ПРОДУКТУ

```
# %%
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import yfinance as yf

# %%
# Loading Data
DATA_SET = "^NDX"
YEARS = 10

data = yf.download(DATA_SET, start=f'20{24-YEARS}-01-01', end='2024-01-
01').reset_index()
data

# %%
# Spliting Data as Train and Validation
SPLIT_RATION = 0.8

#data['Date'] = pd.to_datetime(data['Date']) # converting to date time object

length_train = round(len(data) * SPLIT_RATION)
length_validation = len(data) - length_train

print("Data length :", len(data))
print("Train data length :", length_train)
print("Validation data lenth :", length_validation)

train_data = data[:length_train].iloc[:,:2]
train_data['Date'] = pd.to_datetime(train_data['Date'])  # converting to date time
object

validation_data = data[length_train:].iloc[:,:2]
validation_data['Date'] = pd.to_datetime(validation_data['Date'])  # converting to
date time object

dataset_train = np.reshape(train_data.Open.values, (-1, 1))

# %%
# Normalization & Feature scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range = (0, 1))

dataset_train = np.reshape(train_data.Open.values, (-1, 1))
dataset_train_scaled = scaler.fit_transform(dataset_train)

dataset_validation = validation_data.Open.values
dataset_validation = np.reshape(dataset_validation, (-1,1))
scaled_dataset_validation =  scaler.fit_transform(dataset_validation)
```

```
print("Shape of scaled validation dataset :", scaled_dataset_validation.shape)

dataset_train

# %%
plt.subplots(figsize = (15, 6))
plt.plot(dataset_train)
plt.xlabel("Days")
plt.ylabel("Open Price")
plt.show()

# %%
# Creating X_train and Y_train
TIME_STEP = 50

X_train = []
y_train = []

for i in range(TIME_STEP, length_train):
    X_train.append(dataset_train_scaled[i-TIME_STEP:i,0])
    y_train.append(dataset_train_scaled[i,0])

# convert list to array
X_train, y_train = np.array(X_train), np.array(y_train)

# %%
# Reshaping
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1],1))
y_train = np.reshape(y_train, (y_train.shape[0], 1))

# %%
# Dataframe for training evaluation history saves
train_res_df = pd.DataFrame([], columns=['Model', 'MSE', 'MAE', 'MSLA', 'RMSE',
'Train/predict time'])

# %% [markdown]
# # ARIMA

# %%
# ARIMA imports
import time
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error, mean_absolute_error,
mean_squared_log_error

import warnings
from statsmodels.tools.sm_exceptions import ConvergenceWarning
warnings.simplefilter('ignore', ConvergenceWarning)
```

```
# %%
model_arima = sm.tsa.arima.ARIMA(dataset_train, order=(3, 3, 3))
model_arima_res = model_arima.fit()

print(model_arima_res.summary())

# %%
arima_predictions = []

arima_start = time.time()

print(f'Steps {len(dataset_validation)} to run')
for i in range(len(dataset_validation)):
    print(f'Step {i} running...')
    arima_model_fit_new = sm.tsa.arima.ARIMA(
        np.append(dataset_train, dataset_validation[:i+1]), order=(3,3,3)
    )
    arima_model_fit_new.initialize_approximate_diffuse() # this line
    arima_model_fit_new = arima_model_fit_new.fit()
    forecast = arima_model_fit_new.forecast(steps=1)
    arima_predictions.append(forecast[0])

mse = mean_squared_error(dataset_validation, arima_predictions)
mae = mean_absolute_error(dataset_validation, arima_predictions)
msle = mean_squared_log_error(dataset_validation, arima_predictions)
rmse = np.sqrt(mse)

arima_end = time.time()

train_res_df.loc[len(train_res_df.index)] = ['ARIMA', mse, mae, msle, rmse,
(arima_end-arima_start)]

# %%
model_arima_res.plot_diagnostics(figsize=(8,8))

# %% [markdown]
# # SARIMA

# %%
sarima_model = sm.tsa.SARIMAX(dataset_train, order = (1, 1, 1), seasonal_order =
(1, 1, 1, 12),  trend='c')
sarima_res = sarima_model.fit(maxiter=len(dataset_train), disp=False, method =
'nm')

print(sarima_res.summary())

# %%
sarima_predictions = []

sarima_start = time.time()
```

```
print(f'Steps {len(dataset_validation)} to run')
for i in range(len(dataset_validation)):
    print(f'Step {i} running...')
    sarima_model_fit_new = sm.tsa.SARIMAX(
        np.append(dataset_train, dataset_validation[:i+1]), order = (1, 1, 1),
seasonal_order = (1, 1, 1, 12),  trend='c'
    )
    sarima_model_fit_new = sarima_model_fit_new.fit()
    forecast = sarima_model_fit_new.forecast(steps=1)
    sarima_predictions.append(forecast[0])

mse = mean_squared_error(dataset_validation, sarima_predictions)
mae = mean_absolute_error(dataset_validation, sarima_predictions)
msle = mean_squared_log_error(dataset_validation, sarima_predictions)
rmse = np.sqrt(mse)

sarima_end = time.time()

train_res_df.loc[len(train_res_df.index)] = ['SARIMA', mse, mae, msle, rmse,
(sarima_end-sarima_start)]

# %%
sarima_res.plot_diagnostics(figsize=(8,8))

# %% [markdown]
# # RNN

# %%
from tensorflow.keras import models as kerasModels

from keras.models import Model, Sequential
from keras.layers import SimpleRNN, Dropout, Dense, Input, LayerNormalization,
MultiHeadAttention, GlobalAveragePooling1D
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam
from keras.utils import plot_model

# %%
model_RNN = Sequential()

model_RNN.add(SimpleRNN(units=64, activation="tanh", return_sequences=True,
input_shape=(X_train.shape[1], 1)))
model_RNN.add(Dropout(0.2))

model_RNN.add(SimpleRNN(units=64, activation="tanh", return_sequences=True))
model_RNN.add(Dropout(0.2))

model_RNN.add(SimpleRNN(units=64, activation="tanh", return_sequences=False))
model_RNN.add(Dropout(0.2))
```

```
model_RNN.add(Dense(units=1))

model_RNN.compile(optimizer=Adam(), loss="mean_squared_error",
metrics=["mean_squared_error", "mean_absolute_error", 'root_mean_squared_error',
'mean_squared_logarithmic_error'])

try:
    kerasModels.load_model('MODELS/_Trained_models/rnn.keras')
except ValueError:
    rnn_start = time.time()
    history_rnn = model_RNN.fit(X_train, y_train, epochs=500, batch_size=32,
callbacks=[EarlyStopping(monitor='loss', patience=50)])
    model_RNN.save('MODELS/_Trained_models/rnn.keras')
    rnn_end = time.time()
    train_res_df.loc[len(train_res_df.index)] = [
        'RNN',
        min(history_rnn.history['mean_squared_error']),
        min(history_rnn.history['mean_absolute_error']),
        min(history_rnn.history['mean_squared_logarithmic_error']),
        min(history_rnn.history['root_mean_squared_error']),
        (rnn_end-rnn_start)
    ]
else:
    model_RNN = kerasModels.load_model('MODELS/_Trained_models/rnn.keras')

model_RNN.summary()
# plot_model(model_RNN, show_dtype=True, show_layer_names=True, show_shapes=True,
to_file='model_RNN.png')

# %% [markdown]
# # LSTM

# %%
y_train = scaler.fit_transform(y_train)

# %%
from keras.layers import LSTM

model_lstm = Sequential()
model_lstm.add(
    LSTM(64,return_sequences=True,input_shape = (X_train.shape[1],1))) #64 lstm
neuron block
model_lstm.add(
    LSTM(64, return_sequences= False))
model_lstm.add(Dense(32))
model_lstm.add(Dense(1))
model_lstm.compile(loss = "mean_squared_error", optimizer = Adam(), metrics =
["mean_squared_error", "mean_absolute_error", 'root_mean_squared_error',
'mean_squared_logarithmic_error'])
```

```
try:
    kerasModels.load_model('MODELS/_Trained_models/lstm.keras')
except ValueError:
    lstm_start = time.time()
    history_lstm = model_lstm.fit(X_train, y_train, epochs=1000, batch_size=32,
callbacks=[EarlyStopping(monitor='loss', patience=50)])
    model_lstm.save('MODELS/_Trained_models/lstm.keras')
    lstm_end = time.time()
    train_res_df.loc[len(train_res_df.index)] = [
        'LSTM',
        min(history_lstm.history['mean_squared_error']),
        min(history_lstm.history['mean_absolute_error']),
        min(history_lstm.history['mean_squared_logarithmic_error']),
        min(history_lstm.history['root_mean_squared_error']),
        (lstm_end-lstm_start)
    ]
else:
    model_lstm = kerasModels.load_model('MODELS/_Trained_models/lstm.keras')

# plot_model(model_lstm, show_dtype=True, show_layer_names=True, show_shapes=True,
to_file='model_lstm.png')

# %% [markdown]
# # Transformers

# %%
# Transformer Block
def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout=0):
    x = LayerNormalization(epsilon=1e-6)(inputs)
    x = MultiHeadAttention(key_dim=head_size, num_heads=num_heads,
dropout=dropout)(x, x)
    x = Dropout(dropout)(x)
    res = x + inputs

    x = LayerNormalization(epsilon=1e-6)(res)
    x = Dense(ff_dim, activation="relu")(x)
    x = Dropout(dropout)(x)
    x = Dense(inputs.shape[-1])(x)
    return x + res

# %%
inputs = Input(shape=(X_train.shape[1], 1))
x = transformer_encoder(inputs, head_size=256, num_heads=4, ff_dim=4, dropout=0.1)
x = GlobalAveragePooling1D(data_format='channels_first')(x)
x = Dropout(0.1)(x)
x = Dense(20, activation="relu")(x)
outputs = Dense(1, activation="linear")(x)

model_transf = Model(inputs=inputs, outputs=outputs)
```

```
model_transf.compile(optimizer = Adam(), loss = "mean_squared_error", metrics =
["mean_squared_error", "mean_absolute_error", 'root_mean_squared_error',
'mean_squared_logarithmic_error'])

try:
    kerasModels.load_model('MODELS/_Trained_models/transf.keras')
except ValueError:
    transf_start = time.time()
    history_transf = model_transf.fit(X_train, y_train, epochs=2000, batch_size=32,
callbacks=[EarlyStopping(monitor='loss', patience=50)])
    model_transf.save('MODELS/_Trained_models/transf.keras')
    transf_end = time.time()
else:
    model_transf = kerasModels.load_model('MODELS/_Trained_models/transf.keras')

# plot_model(model_transf, show_dtype=True, show_layer_names=True,
show_shapes=True, to_file='model_transf.png')

# %% [markdown]
# # Evalueting models

# %%
plt.figure(figsize =(10,7))
plt.plot(history_rnn.history["loss"])
plt.xlabel("Epochs")
plt.ylabel("Losses")
plt.title("Simple RNN model, Loss vs Epoch")
plt.show()

plt.figure(figsize =(10,7))
plt.plot(history_lstm.history["loss"])
plt.xlabel("Epochs")
plt.ylabel("Losses")
plt.title("LSTM model, Loss vs Epoch")
plt.show()

plt.figure(figsize =(10,7))
plt.plot(history_transf.history["loss"])
plt.xlabel("Epochs")
plt.ylabel("Losses")
plt.title("Transformer, Loss vs Epoch")
plt.show()

# %%
dataset_validation = validation_data.Open.values  # getting "open" column and
converting to array
dataset_validation = np.reshape(dataset_validation, (-1,1))  # converting 1D to 2D
array
scaled_dataset_validation =  scaler.fit_transform(dataset_validation)  # scaling
open values to between 0 and 1
```

```
print("Shape of scaled validation dataset :",scaled_dataset_validation.shape)

# %%
# Creating X_test and y_test
X_test = []
y_test = []

for i in range(TIME_STEP, length_validation):
    X_test.append(scaled_dataset_validation[i-TIME_STEP:i,0])
    y_test.append(scaled_dataset_validation[i,0])

# Converting to array
X_test, y_test = np.array(X_test), np.array(y_test)
print("Shape of X_test before reshape :", X_test.shape)
print("Shape of y_test before reshape :", y_test.shape)

X_test.shape[1]

X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))  # reshape to 3D
array
y_test = np.reshape(y_test, (-1, 1))  # reshape to 2D array
print("Shape of X_test after reshape :", X_test.shape)
print("Shape of y_test after reshape :", y_test.shape)

# %%
# predictions with X_test data
y_pred_of_test_RNN = model_RNN.predict(X_test)
y_pred_of_test_LSTM = model_lstm.predict(X_test)
y_pred_of_test_Transf = model_transf.predict(X_test)

# y_pred_of_test_ARIMA = np.reshape(arima_predictions, (-1, 1))
# y_pred_of_test_SARIMA = np.reshape(sarima_predictions, (-1, 1))


# %%

mean_squared_error(y_test, model_transf.predict(X_test))

# %%
valid_data = scaler.inverse_transform(y_test)

mse = mean_squared_error(y_test, y_pred_of_test_RNN)
mae = mean_absolute_error(y_test, y_pred_of_test_RNN)
msle = mean_squared_log_error(y_test, y_pred_of_test_RNN)
rmse = np.sqrt(mse)

train_res_df.loc[len(train_res_df.index)] = ['SimpleRNN', mse, mae, msle, rmse, '']

mse = mean_squared_error(y_test, y_pred_of_test_LSTM)
mae = mean_absolute_error(y_test, y_pred_of_test_LSTM)
```

```
msle = mean_squared_log_error(y_test, y_pred_of_test_LSTM)
rmse = np.sqrt(mse)

train_res_df.loc[len(train_res_df.index)] = ['LSTM', mse, mae, msle, rmse, '']

mse = mean_squared_error(y_test, y_pred_of_test_Transf)
mae = mean_absolute_error(y_test, y_pred_of_test_Transf)
msle = mean_squared_log_error(y_test, y_pred_of_test_Transf)
rmse = np.sqrt(mse)

train_res_df.loc[len(train_res_df.index)] = ['Transformer', mse, mae, msle, rmse,
'']

# %%
# scaling back from 0-1 to original
y_pred_of_test_RNN = scaler.inverse_transform(y_pred_of_test_RNN)
y_pred_of_test_LSTM = scaler.inverse_transform(y_pred_of_test_LSTM)
y_pred_of_test_Transf = scaler.inverse_transform(y_pred_of_test_Transf)

figure, axis = plt.subplots(3, 2, figsize=(40, 30))
figure.suptitle(f'{DATA_SET} of {YEARS} years prdeictions with different models')

axis[0,0].plot(y_pred_of_test_ARIMA, label = "ARIMA Predicted data", c = "r")
axis[0,0].plot(dataset_validation, label = "Real data", c = "g")
axis[0,0].set_title("ARIMA model")
axis[0,0].legend()

axis[0,1].plot(y_pred_of_test_SARIMA, label = "SARIMA Predicted data", c = "r")
axis[0,1].plot(dataset_validation, label = "Real data", c = "g")
axis[0,1].set_title("SARIMA model")
axis[0,1].legend()

axis[1,0].plot(y_pred_of_test_RNN, label = "RNN Predicted data", c = "r")
axis[1,0].plot(scaler.inverse_transform(y_test), label = "Real data", c = "g")
axis[1,0].set_title("Simple RNN model")
axis[1,0].legend()

axis[1,1].plot(y_pred_of_test_LSTM, label = "LSTM Predicted data", c = "r")
axis[1,1].plot(scaler.inverse_transform(y_test), label = "Real data", c = "g")
axis[1,1].set_title("LSTM model")
axis[1,1].legend()

axis[2,0].plot(y_pred_of_test_Transf, label = "Transformer Predicted data", c =
"r")
axis[2,0].plot(scaler.inverse_transform(y_test), label = "Real data", c = "g")
axis[2,0].set_title("Transformer model")
axis[2,0].legend()

plt.show()
```

```
# %%
try:
    pd.read_csv(f'MODELS/train_res_{DATA_SET}{YEARS}.csv')
except FileNotFoundError:
    train_res_df.to_csv(f'MODELS/train_res_{DATA_SET}{YEARS}.csv')
else:
    train_res_df1 = pd.read_csv(f'MODELS/train_res_{DATA_SET}{YEARS}.csv')
    train_res_df = pd.concat([train_res_df, train_res_df1])
    train_res_df.drop_duplicates(subset='Model', keep='first')
    train_res_df.to_csv(f'MODELS/train_res_{DATA_SET}{YEARS}.csv')

train_res_df
```