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: «Система прогнозування попиту для управління запасами»

Виконав:
студент IV курсу, групи КІ-13
Євтушенко Владислав Анатолійович _____

Керівник:
доцент кафедри ШІ, д.ф.,
Гуськова Віра Геннадіївна _____

Консультант з економічного розділу:
доцент кафедри економічної кібернетики, к.е.н., доцент,
Рощина Надія Василівна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри ШІ, к.т.н., доцент,
Комариста Богдана Миколаївна _____

Рецензент:
професор кафедри ММСА, д.т.н., професор,
Бідюк Петро Іванович _____

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

Київ – 2025 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
«15» січня 2025 р.

ЗАВДАННЯ

на дипломну роботу студенту

Євтушенку Владиславу Анатолійовичу

1. Тема роботи «Система прогнозування попиту для управління запасами», керівник роботи Гуськова Віра Геннадіївна, доцент кафедри штучного інтелекту, д.ф., затверджені наказом по НН ІПСА від «26» травня 2025 р. № 1759-с.

2. Термін подання студентом роботи «09» червня 2025 року.

3. Вихідні дані до роботи: історичні дані продажів онлайн магазину.

4. Зміст роботи: дослідження предметної області, вибір та аналіз методів та моделей, огляд та обробка даних, практична реалізація системи, порівняння результатів, функціонально-вартісний аналіз.

5. Перелік ілюстративного матеріалу: презентація.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Надія Василівна, доцент, к. е. н.		

7. Дата видачі завдання «03» лютого 2025 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури за темою	21.04.2025	Виконано
2	Підготовка першого розділу	01.05.2025	Виконано
3	Підготовка другого розділу	10.05.2025	Виконано
4	Розробка програмного продукту	19.05.2025	Виконано
5	Підготовка третього розділу	24.05.2025	Виконано
6	Оформлення дипломної роботи	03.06.2025	Виконано
7	Підготовка презентації доповіді	09.06.2025	Виконано

Студент Владислав ЄВТУШЕНКО

Керівник Віра ГУСЬКОВА

РЕФЕРАТ

Дипломна робота: 104 с., 20 рис., 7 табл., 23 посилань, 1 додаток.

ПОПИТ, ПРОГНОЗУВАННЯ, ЗАПАСИ, ЧАСОВІ РЯДИ, МАШИННЕ НАВЧАННЯ, УПРАВЛІННЯ, ОПТИМІЗАЦІЯ

Об’єктом дослідження є процес управління товарними запасами в умовах змінного попиту.

Предметом дослідження є методи прогнозування попиту та їх вплив на прийняття рішень щодо оптимізації рівня запасів.

Метою роботи є розробка системи прогнозування попиту на основі машинного навчання для підвищення ефективності управління товарними запасами.

Дипломна робота присвячена розробці інтелектуальної системи прогнозування попиту, що дозволяє підприємствам оптимізувати запаси й покращити процес прийняття управлінських рішень. У роботі розглянуто як класичні статистичні методи, так і сучасні алгоритми машинного навчання та глибокого навчання. Проведено порівняльний аналіз їхньої ефективності на основі історичних даних продажів.

Встановлено, що точне прогнозування попиту дозволяє зменшити витрати на зберігання товарів, уникнути дефіциту та підвищити рівень обслуговування клієнтів.

Результати моделювання використано для розробки алгоритмів прийняття рішень з управління запасами, включно з обчисленням економічно обґрунтованого обсягу замовлень та точок повторного замовлення. Запропонована система може бути впроваджена в малому та середньому бізнесі як альтернатива дорогим комерційним рішенням.

ABSTRACT

Bachelor's thesis: 104 p., 20 figures, 7 tables, 23 references, 1 appendix.

DEMAND, FORECASTING, INVENTORY, TIME SERIES, MACHINE LEARNING, MANAGEMENT, OPTIMIZATION

The object of the study is the process of inventory management under variable demand conditions.

The subject of the research is demand forecasting methods and their impact on inventory optimization decisions.

The purpose of the work is to develop a demand forecasting system based on machine learning to improve inventory management efficiency.

This thesis focuses on developing an intelligent demand forecasting system that enables businesses to optimize inventory levels and improve decision-making processes. Both classical statistical approaches and modern machine learning and deep learning models are explored. A comparative analysis of their forecasting accuracy based on historical sales data is conducted.

Accurate demand forecasting is shown to reduce storage costs, prevent stockouts, and increase customer service levels.

The modeling results are used to develop decision-support algorithms for inventory management, including the calculation of economic order quantities and reorder points. The proposed system is suitable for implementation in small and medium-sized businesses as an alternative to costly commercial solutions.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

ANN – Artificial Neural Network

RNN – Recurrent Neural Network

GPU – Graphics Processing Unit

MLP – Multi-Layer Perceptron

ARIMA – AutoRegressive Integrated Moving Average

SARIMA – Seasonal AutoRegressive Integrated Moving Average

XGBoost – eXtreme Gradient Boosting

LSTM – Long Short-Term Memory

N-BEATS – Neural Basis Expansion Analysis for Interpretable Time Series Forecasting

EOQ – Economic Order Quantity

ROP – Reorder Point

AIC – Akaike Information Criterion

BIC – Bayesian Information Criterion

MSE – Mean Squared Error

RMSE – Root Mean Squared Error

MAE – Mean Absolute Error

MAPE – Mean Absolute Percentage Error

ВСТУП

Управління[1] товарними запасами є складним завданням, яке вимагає постійного балансу між наявністю необхідного обсягу продукції для задоволення попиту і мінімізацією витрат, пов’язаних із її зберіганням. У багатьох галузях навіть незначні похибки в оцінці потреб ринку можуть призвести до фінансових втрат, зниження рівня сервісу або надлишкових складських залишків. З огляду на це, підприємства все більше зацікавлені в застосуванні систем, здатних заздалегідь оцінити попит і підтримувати оптимальний рівень запасів.

Один з підходів до вирішення цієї проблеми базується на аналізі історичних даних про продажі та використанні алгоритмів прогнозування, що дозволяють враховувати сезонність, тренди, маркетингові впливи та зовнішні фактори. У цьому контексті дедалі важливішу роль відіграють інструменти штучного інтелекту — зокрема методи машинного навчання, здатні адаптуватися до складних і змінних закономірностей поведінки споживачів.

Розробка спеціалізованої системи, що поєднує прогнозну аналітику з управлінськими рішеннями щодо закупівель, дає змогу компаніям підвищувати точність планування, знижувати витрати і підтримувати гнучкість у взаємодії з ринком. Таким чином, питання інтеграції прогнозування попиту в логістичні процеси набуває не лише прикладного, а й стратегічного значення.

[1]Тут і нижче використані такі інструменти штучного інтелекту як чат-боти з генеративним штучним інтелектом ChatGPT, Claude.ai, виключно для корегування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КПІ ім. Ігоря Сікорського (протокол №11 Вченої ради КПІ ім. Ігоря Сікорського від 11 грудня 2023 р.).

РОЗДІЛ 1 ОГЛЯД СФЕРИ ПРОГНОЗУВАННЯ ПОПИТУ В КОНТЕКСТІ УПРАВЛІННЯ ЗАПАСАМИ

1.1 Актуальність теми та постановка проблеми

У сучасних умовах високої конкуренції, нестабільності постачань та зростаючих очікувань споживачів ефективне управління запасами є одним із ключових чинників забезпечення стабільної діяльності підприємств. Запаси – це не лише сировина чи готова продукція, а й фінансовий ресурс, «заморожений» у матеріальній формі. Від того, наскільки грамотно організовано управління запасами, залежить рівень обслуговування клієнтів, витрати на зберігання, своєчасність виконання замовлень та фінансова стійкість підприємства.

Управління запасами дозволяє досягти балансу між надмірними залишками, які спричиняють додаткові витрати, та дефіцитом товару, що може призвести до втрати продажів і клієнтів.

Неефективне управління запасами тягне за собою низку серйозних проблем. З одного боку, надлишкові запаси призводять до зростання витрат на зберігання, втрат через псування або моральне старіння товару, а також заморожування обігових коштів. З іншого боку, нестача запасів або часте виникнення дефіциту призводить до зривів постачання, незадоволення клієнтів і втрати прибутку. Такі коливання негативно впливають на репутацію підприємства, його конкурентоспроможність та здатність швидко реагувати на зміни ринку.

Одним із ключових інструментів забезпечення ефективного управління запасами є прогнозування попиту. Точне передбачення обсягів продажів або споживання продукції дозволяє підприємствам своєчасно поповнювати запаси, уникати як надлишків, так і дефіциту. Прогнозування попиту стає основою для ухвалення рішень щодо закупівель, виробництва, логістики та стратегічного планування.

Сучасні методи прогнозування базуються на статистичних підходах, машинному навчанні, обробці великих обсягів даних і дозволяють враховувати сезонність, тренди, маркетингові кампанії, зміни у поведінці споживачів. Таким чином, якісне прогнозування дозволяє не лише оптимізувати запаси, а й формувати конкурентні переваги.

1.2 Фактори, що впливають на попит

Попит у роздрібній торгівлі формується під впливом численних чинників. Серед них виокремлюють зовнішні та часові характеристики, що закладені в історії продажів.

До зовнішніх факторів належать економічні фактори. Спад чи зростання економіки загалом впливають на платоспроможність населення. Наприклад, високий рівень інфляції може різко знизити попит на непріоритетні товари, тоді як стабільне зростання доходів навпаки підвищує купівельну активність [1].

Природні цикли й погодні умови генерують стійку сезонність попиту. Зимові холоди підвищують попит на теплий одяг і опалювальне обладнання, тоді як літня спека стимулює продажі пляжного спорядження та кондиціонерів. Сезонні свята (Новорічні, Великодні, тощо) також викликають регулярні стрибки продажів відповідних категорій товарів [12].

Ціноутворення, рекламні кампанії й акційні пропозиції викликають короткострокові зміни попиту. Сезонна чи спеціальна знижка здатна суттєво збільшити обсяг продажів на період акції.

Таким чином, зовнішні чинники створюють контекст, у якому змінюється поведінка споживачів. Аналіз цих факторів дозволяє виявити локальні та глобальні ризики або можливості.

Поряд із зовнішніми регресорами, в сучасних методах прогнозування попиту широко застосовують часові ознаки, які витягують інформацію з самої

послідовності продажів. Вони дозволяють моделі безпосередньо враховувати властивості часового ряду (наприклад, автокореляцію та сезонність) при прогнозуванні.

Лагові ознаки — це значення цільової змінної з попередніх часових періодів. На основі припущення, що «те, що відбулося в минулому, може містити інформацію про майбутнє», лаги корисні для виявлення автокореляції в даних. До цього класу також належать вкладені (nested) лагові ознаки: наприклад, сумарний продаж за останні дві години, за минулі три дні або за попередній тиждень. Лагові ознаки відтворюють довгострокову закономірність та швидку реакцію на неї, якщо використовується малі лаги [13].

Ковзні (віконні) ознаки будують статистики (середнє, максимум, мінімум тощо) по фіксованому інтервалу попередніх спостережень. Наприклад, ковзне середнє за останні 7 днів формується як середнє продажів у вікні шириною 7 днів, що ковзає по кожному моменту часу. Ковзні статистики допомагають згладити випадкові коливання та відстежити локальні тренди в даних. Головна мета їхнього використання — обчислювати статистики зі значень даних, визначаючи інтервал, що включає поточний та кілька попередніх вимірювань.

Сезонні та календарні ознаки відображають повторювані календарні ефекти. Сезонні ознаки зазвичай моделюють щорічні цикли (наприклад, номінальними ознаками можуть бути місяць або квартал року, флаг свят), тижневі цикли (день тижня, вихідні), добові (година, робочий час тощо) і спеціальні події (святкові періоди). Такі ознаки створюються на основі календаря і не залежать від модельованих продажів. Вони дозволяють моделі враховувати закономірні коливання попиту, що повторюються щороку чи щотижня.

Трендові ознаки моделюють плавні (лінійні чи нелінійні) зміни в загальному тренді даних. Наприклад, можна додати числову ознаку «послідовний номер періоду часу» або натренувати поліном, щоб захопити довгостроковий зріст чи спад продажів. Трендові ознаки допомагають моделі відстежувати загальну тенденцію незалежно від сезонних флуктуацій. Хоча

формальне кодування тренду залежить від методу, важливо відокремлювати довгостроковий тренд від циклічних ефектів [13].

Часові ознаки дозволяють безпосередньо моделювати статистичні залежності в даних, які важко відобразити за допомогою чисто зовнішніх змінних. Оскільки вони базуються на самій історії продажів, модель не потребує додаткових зовнішніх прогнозів (на відміну від регресії з екзогенними змінними). Це робить процес прогнозування більш автономним і зменшує залежність від точності передбачень зовнішніх факторів (які можуть бути складними для прорахунку).

У задачах прогнозування продажів використання часових ознак має низку переваг порівняно з категоріальними ознаками. Часові ознаки відображають внутрішню структуру даних, дозволяючи моделі виявляти стабільні залежності, що повторюються з часом.

Категоріальні ознаки, такі як тип товару або магазин, можуть бути інформативними, але часто додають шум до моделі. Це пов'язано з тим, що вони можуть мати багато унікальних значень, що призводить до розрідженості даних і ускладнює навчання моделі. Крім того, вплив категоріальних ознак може змінюватися з часом, що знижує їхню стабільність як предикторів.

Традиційні методи кодування категоріальних ознак, такі як one-hot encoding, не завжди ефективні в задачах прогнозування, особливо коли часові ряди мають виражені тренди або коли горизонт прогнозування є далеким.

1.3 Прогнозування в контексті управління запасами

Управління запасами — це спеціалізована галузь управління бізнесом, яка включає стратегічне планування та контроль за запасами з метою підтримання оптимальної кількості товарів або предметів.

Основні виклики в управлінні запасами пов’язані з необхідністю

досягнення балансу між надмірною кількістю запасів і їх нестачею. Ефективне управління запасами має вирішальне значення, оскільки воно безпосередньо впливає на операційну ефективність, задоволеність клієнтів і фінансові цілі організації [15].

Точне прогнозування попиту є ключовим фактором успіху в ефективному управлінні запасами. Воно охоплює передбачення, проектування або оцінку очікуваного попиту на продукцію у визначений майбутній період.

Прогнозування попиту відіграє вирішальну роль в управлінні запасами, оскільки підвищує конкурентоспроможність організації та сприяє ухваленню обґрунтованих рішень. Це, своєю чергою, створює основу для планування поповнення запасів, розподілу ресурсів та ефективного управління ланцюгом постачання [17].

Прогнозування попиту становить значну складність для різних галузей через дві основні проблеми. По-перше, реальний попит клієнтів може бути нерегулярним. По-друге, часто важко передбачити, коли саме цей попит виникне.

У зв’язку з цим прогнозування попиту є надзвичайно важливим для прийняття обґрунтованих рішень щодо управління запасами та планування виробництва. Завдяки узгодженню рівня запасів із прогнозованим попитом, точне прогнозування дозволяє зменшити витрати на запаси та забезпечити вищий рівень задоволеності клієнтів шляхом своєчасного та стабільного виконання їхніх замовлень.

1.4 Огляд сучасних технік прогнозування попиту

Якісне прогнозування, також відоме як експертне прогнозування, — це поширений метод прогнозування, який ґрунтується на судженнях експертів або думках споживачів, а не на числовому аналізі. Якісне прогнозування є

корисним і часто необхідним у випадках, коли бракує історичних даних, що є основою для будь-яких кількісних методів прогнозування. Воно також є доцільним у ситуаціях, коли історичні значення підозрюються у відсутності суттєвого впливу на майбутні результати.

Хоча це поширена практика, методи якісного прогнозування можуть бути упередженими, оскільки значною мірою залежать від людських думок, які можуть бути під впливом особистих або політичних мотивів. Ефект нещодавності створює додаткові труднощі для експертного прогнозування, оскільки відомо, що експерти або прогнозисти мають тенденцію надавати більшого значення нещодавнім подіям [17]. У результаті прогнози часто виявляються занадто близькими до останніх референтних точок.

Для створення прогнозів на майбутнє кількісне прогнозування спирається на математичні (статистичні) моделі. Завдяки об’єктивності кількісних моделей прогнозування їх рекомендується застосовувати у випадках, коли наявна достатня кількість історичних даних, які мають значний зв’язок із майбутніми значеннями (тобто минулі тенденції можуть продовжитися в майбутньому).

Кількісне прогнозування може використовувати або крос-секційні дані (тобто дані, зібрані в один момент часу), або часові ряди. Останні є найпоширенішим типом даних, що використовуються в прогнозуванні.

Окрім різноманітності типів даних, які можна використовувати, кількісне прогнозування охоплює кілька моделей залежно від того, які предикторні змінні використовуються в моделі. Ці моделі можна класифікувати як моделі часових рядів або як пояснювальні (каузальні) моделі. Пояснювальні моделі прогнозування спрямовані на виявлення базових факторів, що впливають на цільову (прогнозовану) змінну (наприклад, стан економіки, чисельність населення тощо). На відміну від моделей часових рядів, пояснювальні моделі не враховують історії попиту (попередніх значень попиту) [17].

До цієї категорії прогнозування належить багато методів, найвідомішим із яких є регресійний аналіз — узагальнюючий термін для методів, які вивчають вплив однієї або кількох незалежних змінних на залежну змінну.

1.5 Огляд існуючих платформ прогнозування попиту

Lokad — це хмарна платформа, спеціалізована на оптимізації ланцюгів постачання за допомогою машинного навчання та глибокого навчання. Її п’яте покоління прогнозного рушія використовує диференційоване програмування та ймовірнісне прогнозування для точного передбачення попиту навіть за обмежених або нерегулярних даних. Замість одного значення попиту система оцінює розподіл ймовірностей, що дозволяє краще управляти ризиками надлишкових або недостатніх запасів . Платформа використовує кореляції між продуктами для покращення прогнозів, навіть якщо окремі товари мають обмежену історію продажів [21].

Oracle Retail Demand Forecasting Cloud Service є рішенням від Oracle, яке інтегрує штучний інтелект і машинне навчання для точного прогнозування попиту та оптимізації управління запасами. При прогнозуванні враховує сезонність, акції, відсутність товарів та інші фактори для точного передбачення попиту. Система використовує машинне навчання для групування магазинів за схожими закономрностями продажів, що дозволяє краще адаптувати асортимент. Крім цього є автоматичне коригування замовлення на основі прогнозів попиту, поточних запасів та історичних даних [22].

RELEX — це хмарна платформа для планування попиту, яка поєднує машинне навчання з традиційними методами прогнозування для досягнення високої точності. Система дозволяє швидко реагувати на зміни в попиті та оптимізувати запаси. Інструмент використовує часові ряди для довгострокового планування, причинно-наслідкове моделювання для середньострокових впливів (наприклад, акцій) та машинне навчання для врахування зовнішніх факторів, таких як погода [23].

Незважаючи на наявність комерційних рішень, які демонструють високу ефективність у прогнозуванні попиту та оптимізації управління запасами, важливо зазначити, що впровадження подібних систем часто вимагає значних

фінансових та технічних ресурсів. Ці платформи зазвичай орієнтовані на великі корпорації, які мають розгалужену інфраструктуру, значні обсяги даних та високий рівень зрілості бізнес-процесів. Для малих або середніх підприємств, а також для освітніх і наукових цілей, такі рішення можуть виявитися занадто складними або недосяжними через високу вартість ліцензій, обмеження у налаштуванні під специфіку конкретного бізнесу, а також складність інтеграції з наявними IT-системами.

Такі компанії часто стикаються з унікальними викликами: нестабільним попитом, обмеженими ресурсами для зберігання товарів, а також необхідністю швидкого реагування на зміни ринку. Для них важливо мати інструмент, який дозволяє навіть за мінімального набору даних отримувати базові прогнози попиту та приймати обґрунтовані рішення щодо обсягів замовлень чи поповнення запасів. Розробка власної системи може допомогти малим бізнесам уникнути ризику надлишкових запасів, що призводять до замороження капіталу, або, навпаки, нестачі товарів, що шкодить рівню обслуговування клієнтів.

1.6 Формулювання задачі

Одним із ключових елементів управління запасами є точне прогнозування попиту на товари. Недооцінка попиту може призвести до втрат продажів, тоді як його переоцінка — до надлишкових запасів і збільшення витрат на зберігання. Тому розробка моделей прогнозування попиту, які дозволяють покращити процес прийняття рішень щодо управління запасами, є актуальним завданням.

Метою даної роботи є створення системи прогнозування попиту на основі аналізу історичних даних про продажі з подальшим використанням отриманих прогнозів для прийняття рішень щодо управління товарними запасами.

Завдання реалізується в кілька етапів, кожен з яких має важливе значення для загального результату.

Перший етап передбачає попередню обробку та підготовку даних. Особливу увагу приділено формуванню часових ознак, що дозволяють моделі враховувати сезонність, тренди, день тижня, місяць, свята, проміжки між продажами тощо. Такі ознаки є ключовими при роботі з часовими рядами, оскільки вони дозволяють підвищити точність прогнозування за рахунок кращого представлення структурної інформації, яка притаманна часовим даним.

Наступним етапом є аналіз даних — вивчення загальних закономірностей, трендів, сезонних коливань, виявлення аномалій, пробілів у даних та кореляцій між різними ознаками. Результати аналізу слугують базою для побудови подальших моделей прогнозування.

На етапі прогнозування здійснюється побудова та навчання моделей, які дозволяють передбачити майбутній попит на товари. Для оцінки якості прогнозування варто застосовути різні метрики точності, які дозволяють порівняти ефективність моделей та обрати найбільш придатні для задачі управління запасами.

Останнім етапом є інтеграція результатів прогнозування у процес прийняття рішень щодо управління запасами. На основі отриманих прогнозів потрібно реалізувати підходи до планування обсягів замовлення, оптимізації рівня запасів та мінімізації пов’язаних витрат.

Загалом, задача, яка розв’язується в межах даної роботи, полягає у створенні інтелектуальної системи прогнозування попиту з метою підвищення ефективності управління товарними запасами в умовах невизначеності ринку.

Висновки до розділу 1

Стабільність і конкурентоспроможність сучасних підприємств значною мірою залежать від здатності ефективно управляти запасами в умовах нестабільного попиту та постійних коливань зовнішнього середовища. Управління запасами — це не лише логістична чи складська задача, а стратегічний інструмент, який безпосередньо впливає на рівень обслуговування клієнтів, витрати та фінансову стійкість компанії. Здатність тримати баланс між надлишками та дефіцитом є критичною, оскільки як надлишок, так і нестача призводять до втрат — фінансових, репутаційних, операційних.

Прогнозування попиту — центральний елемент управління запасами, який дозволяє підприємствам приймати проактивні рішення щодо закупівель, виробництва та логістики. Якість прогнозування безпосередньо визначає здатність компанії реагувати на зміну умов ринку, сезонні піки, маркетингові активності, економічні ризики. Сучасні підходи до прогнозування попиту спираються на статистику, машинне навчання, моделі обробки часових рядів та використання зовнішніх і внутрішніх предикторів. Ці підходи не лише забезпечують точніші прогнози, а й формують потенційні конкурентні переваги.

Попит як об'єкт прогнозування залежить від широкого спектра факторів. Зовнішні чинники — економіка, сезонність, погода, маркетинг — створюють контекст, у якому змінюється поведінка споживачів. Їх аналіз дозволяє моделі виявляти закономірності у зовнішньому середовищі та краще адаптуватися до коливань. Часові характеристики (лагові, ковзні, сезонні, трендові ознаки) дозволяють вловлювати повторювані закономірності та довгострокові тенденції, які приховані в самій історії продажів. Їхня перевага — у мінімальній залежності від додаткових джерел даних та здатності зробити прогнозування автономнішим.

Якісні методи є корисними, коли бракує історичних даних або коли

майбутні умови суттєво відрізняються від минулих. Однак вони можуть бути упередженими через суб’єктивність експертних оцінок. Натомість кількісні методи базуються на математичних моделях і забезпечують об'єктивність, проте вимагають достатньо достовірних і репрезентативних даних.

Сучасні програмні рішення, такі як Lokad, Oracle RDF Cloud Service та RELEX, демонструють високий потенціал у прогнозуванні попиту та оптимізації запасів, використовуючи машинне навчання, часові ряди та каузальні моделі. Проте їх впровадження потребує значних ресурсів і зазвичай доступне лише великим компаніям. Малі бізнеси потребують простіших, адаптивних рішень, здатних забезпечити базовий рівень точності навіть за обмеженої кількості даних.

РОЗДІЛ 2 МЕТОДИ ПРОГНОЗУВАННЯ ПОПИТУ ТА УПРАВЛІННЯ ЗАПАСАМИ

2.1 Методи прогнозування попиту

2.1.1 ARIMA

ARIMA — це модель для прогнозування часових рядів, яка працює шляхом виявлення залежностей між попередніми значеннями ряду та похибками прогнозу. Вона перетворює нестійкий ряд у стаціонарний (за потреби), а потім використовує попередні значення і випадкові коливання для побудови прогнозу наступних точок [3]. Вона поєднує три компоненти: авторегресію (AR), яка враховує залежність поточного значення від попередніх; інтегрування (I), що відповідає за перетворення нестабільного ряду у стаціонарний шляхом диференціювання; та компонент ковзного середнього (MA), який моделює залежність поточного значення від попередніх випадкових шумів.

AR — авторегресія. Дана складова являє собою модель часових рядів, в якій поточне значення часового ряду лінійно залежить від попередніх значень цього ж ряду. Формула (2.1) містить обчислення авторегресії порядку p.

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \epsilon_t, \quad (2.1)$$

де Y_t – поточне спостереження;

c – константа;

ϕ_1 до ϕ_p – параметри авторегресії;

ϵ_t – похибка (залишковий термін) у момент часу t.

При використанні моделі авторегресії необхідно визначити порядок p і кількість рівнянь, необхідне для максимально точного прогнозу за допомогою коефіцієнтів авторегресії.

I — інтеграція. При прогнозуванні часового ряду Y_t може виникнути ситуація, що прогноз буде точнішим і простіше при роботі не з самим процесом Y_t, а з його зміною, тобто, в такому випадку, ми отримаємо часовий ряд.

$$\bar{Y}_t = Y_t - Y_{t-1}.$$

MA — ковзне середнє. Дана модель використовує залежність між наглядом і залишкової помилкою від моделі ковзного середнього, застосованої до спостережень, що містить викиди. Під викидами в даному випадку маються на увазі «піки» і «западини», що вибиваються із загального фону тимчасового ряду [2].

Використання моделі ковзного середнього обумовлено як раз згладжуванням подібних подібних ділянок. Формула (2.2) демонструє обчислення простого ковзного середнього порядку n.

$$Y_t = c + \epsilon_t + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + \cdots + \theta_q\epsilon_{t-q}, \quad (2.2)$$

де Y_t – поточне спостереження;

c – константа;

ϵ_t – похибка в момент часу t;

θ_1 до θ_q – параметри ковзного середнього.

Загальна формула моделі представлена нижче.

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \epsilon_t + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + \cdots + \theta_q\epsilon_{t-q}.$$

Порядок моделі ARIMA визначається як ARIMA (p, d, q):

- p – порядок авторегресії (кількість попередніх значень);
- d – порядок інтеграції (кількість необхідних різниць);
- q – порядок ковзного середнього (кількість попередніх помилок).

2.1.2 SARIMA

SARIMA, що розшифровується як Сезонна Авторегресивна Інтегрована Середня Рухома модель, — це універсальна та широко використовувана модель прогнозування часових рядів. Вона є розширенням несезонної моделі ARIMA, розробленої для роботи з даними, що мають сезонні коливання. SARIMA враховує як короткострокові, так і довгострокові залежності в даних, що робить її надійним інструментом для прогнозування [10]. Модель поєднує в собі концепції авторегресії (AR), інтеграції (I) та середньої рухомої (MA) з сезонними компонентами, формула (2.3).

$$(1-\phi_1 B)(1-\Phi_1 B^S)(1-B)(1-B^S)y_t = (1+\theta_1 B)(1+\Theta_1 B^s)\epsilon_t, \quad (2.3)$$

де y_t – спостережуваний часовий ряд у момент часу t;

B – оператор зсуву назад, що представляє лаг-оператор;

ϕ_1 – коефіцієнт не сезонної авторегресії;

Φ_1 – коефіцієнт сезонної авторегресії;

θ_1 – коефіцієнт не сезонного ковзного середнього;

Θ_1 – коефіцієнт сезонного ковзного середнього;

s – сезо нний період;

ϵ_t – похибка білого шуму в момент часу t.

Сезонна авторегресивна компонента позначається як $(1-\Phi_1 B^S)$. Вона відображає зв’язок між поточним спостереженням і певною кількістю попередніх спостережень із сезонною затримкою. Вираз означає оператор зсуву назад, застосований до сезонних лагів.

Компонента сезонного диференціювання позначається як $(1-B^S)$, де D — порядок сезонного диференціювання. Ця компонента використовується для

приведення часового ряду до стаціонарного вигляду шляхом сезонного диференціювання.

Сезонна компонента ковзного середнього позначається як $(1 + \Theta_1 B^s)$. Вона описує зв’язок між поточним спостереженням і залишковими похибками від моделі ковзного середнього, застосованої до сезонно зсунутих спостережень [10].

2.1.3 Random Forest

Random Forest — це алгоритм ансамблевого навчання, який побудований на основі багатьох дерев рішень, кожне з яких тренується на випадковій підмножині даних і ознак. Під час передбачення результати всіх дерев об'єднуються: для класифікації — голосуванням більшості, для регресії — усередненням.

Дерева рішень можуть застосовуватись як до задач регресії, так і до класифікації. Модель дерева рішень — це поєднання логічних правил типу «якщо–тоді–інакше» у структурі даних типу дерево. Внутрішні вузли дерева позначені термінами, а ребра — тестовими умовами. Під час передбачення вага тестового прикладу порівнюється з цими тестовими значеннями. Листя дерева позначає значення класів, які потрібно передбачити [5].

Перевага дерев рішень полягає у можливості візуалізації результатів і легкості інтерпретації отриманих даних при здійсненні передбачення. Однак, оскільки дерева рішень є простою моделлю, вони можуть виявитися недостатніми для розв'язання складних задач на великих наборах даних. Тому на практиці частіше застосовуються ансамблеві методи, побудовані на основі дерев рішень, такі як випадкові ліси та методи бустингу.

Загальна ідея ансамблевих методів полягає в тому, що за допомогою кількох слабких класифікаторів можна побудувати правило, яке дозволить

підвищити точність передбачення, створивши тим самим сильний мета-класифікатор.

Статистичний бутстреп — це метод визначення статистик розподілу ймовірностей на основі багаторазового генерування вибірок методом Монте-Карло з існуючої вибірки. Сам метод бутстрепу полягає в наступному. Нехай є вибірка Z розміру N — набір незалежних однаково розподілених випадкових величин. Рівномірно відбираємо з цієї вибірки N об'єктів із поверненням. Це означає, що щоразу ми обираємо довільний об’єкт вибірки (вважається, що кожен має однакову ймовірність бути обраним), і кожного разу відбір відбувається серед усіх вихідних об’єктів [4].

Дерева рішень зазвичай мають низьку зміщеність, але високу дисперсію (варіативність), тобто вони дуже чутливі до незначних змін у тренувальних даних. Наприклад, якщо розділити тренувальну вибірку на дві частини і побудувати дерева рішень для кожної, результати для однієї й тієї ж тестової вибірки можуть суттєво відрізнятися. Бутстреп виявився дуже корисним для формування ансамблю на основі дерев рішень, оскільки допомагає боротися з проблемою високої варіативності моделі [5].

У беггінгу дерева не обрізаються, тому кожне з них окремо має велику варіативність, але при усередненні їх результатів ця варіативність зменшується. У середньому декількох спостережень можна знизити оцінку дисперсії даних; аналогічно, маючи велику кількість підвибірок загальної сукупності, тренуючи модель на кожній з них та усереднюючи передбачення, можна зменшити варіативність метамоделі [4].

Random Forest є подальшим вдосконалення беггінгу дерев рішень, яке покликане усунути кореляцію між деревами. Як і в беггінгу, тренуються M дерев рішень на бутстреп-вибірках. Але при побудові дерев на кожному розділенні випадковим чином обирається підмножина всіх предикторів, і розділення дозволяється зробити лише за одним з них. Іншими словами, при побудові випадкового лісу на кожному розділенні дереву не дозволяється використовувати більшість наявних предикторів.

Метод випадкового лісу знижує кореляцію між деревами рішень шляхом випадкового відбору зразків і ознак. Спочатку з вихідних навчальних даних випадково вибирається рівна кількість зразків. Крім того, випадковим чином обирається підмножина ознак для побудови кожного дерева рішень. Використання цих двох форм рандомізації зменшує кореляцію між окремими деревами, що, у свою чергу, знижує ймовірність помилки через перенавчання і підвищує точність моделі [5].

Процес роботи випадкового лісу містить такі кроки.

1. Із початкового навчального набору даних випадковим чином за методом бутстрепу вибираються k підмножин (зразків). Зразки, які не були вибрані, формують набір даних out-of-bag (OOB, тобто «поза вибіркою»). Тут k — це ціле число, і ці k підмножин використовуються для побудови k дерев класифікації або регресії.
2. Дерево T_a вирощується на підмножині a, повторюючи рекурсивно наступні дії для кожного не листового вузла до досягнення максимальної глибини дерева: вибирається випадкова підмножина ознак, серед яких за обчисленням інформаційного критерію (наприклад, приросту інформації) обирається найоптимальніша. Потім вузол розділяється на два дочірніх вузли.
3. Повторюючи другий крок k разів, отримують k дерев, які об’єднуються в один випадковий ліс. Остаточний результат визначається методом голосування більшості серед усіх дерев-класифікаторів.

2.1.3 XGBoost

XGBoost — це ітеративний алгоритм побудови дерев рішень, що складається з множини дерев. Кожне наступне дерево навчається на

залишкових помилках попередніх дерев. На відміну від методу Random Forest, де підсумковий результат базується на голосуванні більшості дерев, у XGBoost передбачене значення є сумою результатів усіх дерев (рис. 2.1) [6].

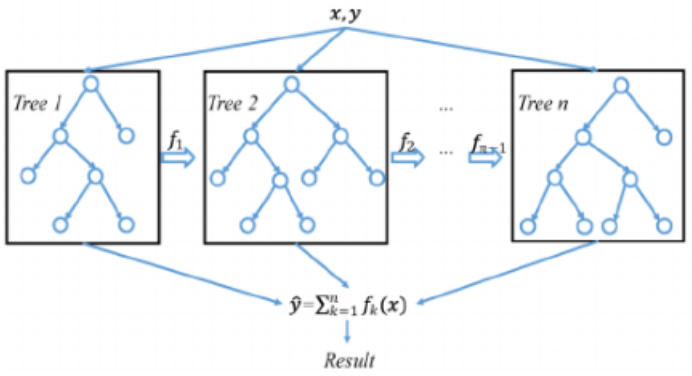

Рисунок 2.1 − Схематичне зображення роботи моделі XGBoost [5]

Алгоритм XGBoost виник із методу, заснованого на деревах рішень, де графічні представлення альтернативних рішень обчислюються на основі певних параметрів. Згодом було розроблено ансамблевий метаалгоритм під назвою «беггінг», який агрегує передбачення з різних дерев рішень, використовуючи стратегію голосування більшості [5].

Шляхом випадкового вибору ознак ця техніка беггінгу була розширена для створення лісу — або ж сукупності дерев рішень.

Продуктивність моделей була покращена шляхом мінімізації помилок, пов’язаних із побудовою послідовних моделей. Крім того, для зменшення цих помилок у послідовній моделі було використано алгоритм градієнтного спуску. В результаті алгоритм XGBoost описується як ефективний підхід для оптимізації градієнтного бустингу завдяки усуненню пропущених значень і зменшенню перенавчання шляхом використання паралельної обробки.

Алгоритм XGBoost оптимізує систему за рахунок паралелізації, обрізки дерев (tree pruning) та апаратної оптимізації [6]. Алгоритм підтримує три типи градієнтного бустингу: градієнтний бустинг на основі машинного навчання,

стохастичний градієнтний бустинг, а також варіанти, що лежать в основі багатьох передових промислових застосувань.

2.1.4 Long Short-Term Memory

Штучні нейронні мережі (ANN) — це тип обчислювальних систем, що імітують та моделюють функціонування людського мозку для аналізу й обробки складних даних адаптивним способом.

Рекурентні нейронні мережі RNN з'явилися як ефективна та масштабована модель штучної нейронної мережі ANN для вирішення низки задач навчання, пов'язаних із послідовними даними. У таких мережах інформація включає петлі (цикли) у прихованому шарі. Ці петлі дозволяють інформації передаватися в різних напрямках, завдяки чому прихований стан відображає інформацію з минулого на певному кроці часу. Таким чином, ці типи мереж мають нескінченну динамічну реакцію на послідовні дані [7].

Однак стандартні та глибокі рекурентні нейронні мережі можуть страждати від проблеми зникнення або вибуху градієнтів. Із додаванням більшої кількості шарів у RNN, що містять функції активації, градієнт функції втрат наближається до нуля. Коли додається більше шарів з функціями активації, градієнт функції втрат може зникати (наближатися до нуля), залишаючи функції без змін. Це зупиняє подальше навчання й передчасно припиняє процес. У результаті параметри моделі відображають лише короткострокові залежності, а інформація з попередніх кроків у часі може бути втрачена. Таким чином, модель сходиться до слабкого (неякісного) розв’язку.

Мережі довготривалої короткочасної пам’яті LSTM вирішують проблему зникнення або вибуху градієнтів і були вперше представлені в. На відміну від звичайної RNN, блок LSTM, окрім прихованого стану, включає комірки пам’яті, які зберігають попередню інформацію, а також вводить низку

спеціальних "вентилів" (вхідний, вихідний та вентиль забування). Ці вентилі дозволяють здійснювати додаткові налаштування (з урахуванням нелінійностей) та запобігають зникненню або вибуху градієнтів [8].

У результаті модель забезпечує точніші передбачення, процес навчання не зупиняється передчасно, а важлива інформація з минулого не втрачається.

Основна відмінність між архітектурами RNN та LSTM полягає в тому, що прихований шар LSTM — це керована одиниця або комірка. Вона складається з чотирьох шарів, які взаємодіють між собою таким чином, щоб сформувати вихід цієї комірки разом із її станом. Ці два елементи потім передаються до наступного прихованого шару.

На відміну від RNN, які мають лише один нейронний шар з активацією tanh, LSTM містить три керувальні вентилі (на основі логістичної сигмоїдної функції) і один шар tanh (рис. 2.2). Вентилі введені для того, щоб обмежити обсяг інформації, яка передається через комірку. Вони визначають, яка частина інформації буде потрібна наступній комірці, а яку слід відкинути. Вихід вентилів зазвичай знаходиться в діапазоні від 0 до 1, де «0» означає «відкинути все», а «1» — «залишити все» [10].

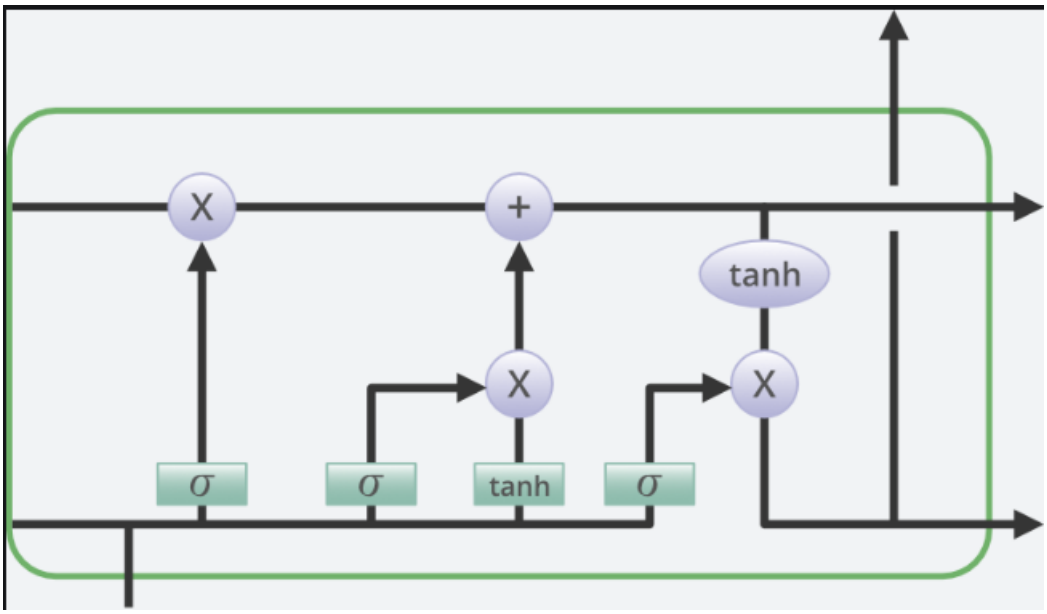

Рисунок 2.2 − Схематичне зображення роботи моделі LSTM [10]

Інформація, яка більше не є корисною у стані комірки, видаляється за допомогою вентиля забування (forget gate). До цього вентиля подаються два входи: x_t (вхід на поточному кроці часу) та h_t-1 (вихід з попередньої комірки) (рис. 2.3). Вони перемножуються з ваговими матрицями, після чого додається зміщення. Отримане значення передається через функцію активації, яка генерує бінарний вихід [10].

Якщо для певного стану комірки вихід дорівнює 0, відповідна частина інформації «забувається». Якщо вихід дорівнює 1 — ця інформація зберігається для подальшого використання.

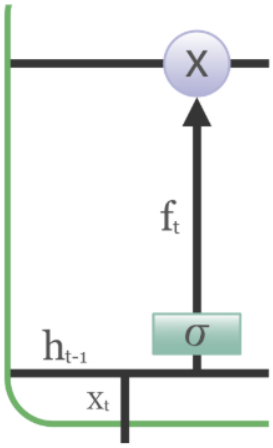

Рисунок 2.3 − Схематичне зображення forget gate [10]

Додавання корисної інформації до стану комірки виконується за допомогою вхідного вентиля (input gate). Спочатку інформація регулюється за допомогою сигмоїдної функції, яка фільтрує значення, що мають бути запам’ятані — подібно до вентиля забування, використовуючи вхідні дані h_t-1 та x_t (рис. 2.4) [10].

Потім за допомогою функції tanh створюється вектор, який містить усі можливі значення із діапазону від -1 до +1, отримані на основі h_t-1 та x_t.

Нарешті, цей вектор перемножується з відрегульованими значеннями, щоб отримати корисну інформацію, яка буде додана до стану комірки.

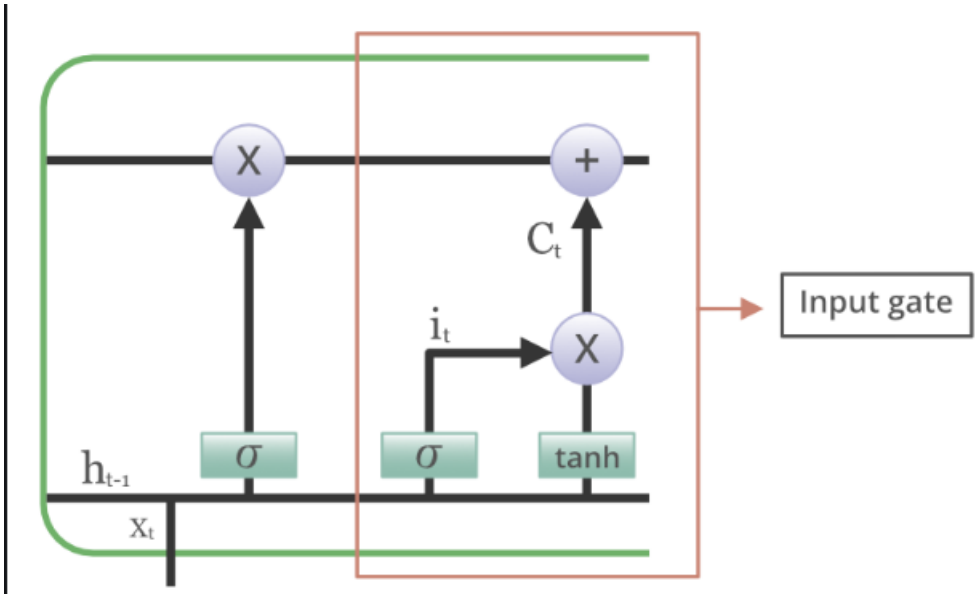

Рисунок 2.4 − Схематичне зображення input gate [10]

Завданням вихідного вентиля (output gate) є вилучення корисної інформації з поточного стану комірки для подальшої передачі як вихід. Спочатку до стану комірки застосовується функція tanh, яка створює вектор із значеннями в межах від -1 до +1. Потім інформація регулюється за допомогою сигмоїдної функції — фільтруються значення, які мають бути збережені, з використанням вхідних даних h_t-1 та x_t [10].

Нарешті, значення цього вектора перемножуються з відрегульованими значеннями, після чого результат надсилається як вихід та як вхід до наступної комірки, як показано на рисунку 2.5.

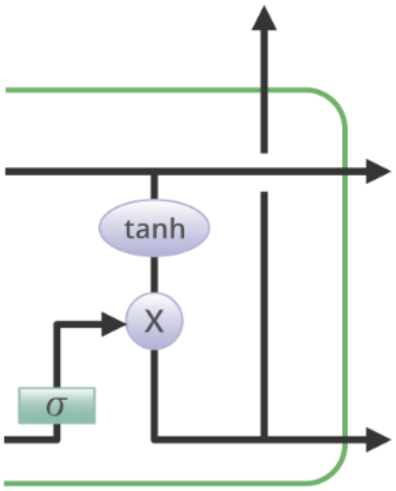

Рисунок 2.5 − Схематичне зображення input gate [10]

2.1.5 N-BEATS

N-BEATS — це глибока нейронна мережа для точкового прогнозування уніваріантних часових рядів.

N-BEATS повністю побудована на основі багатошарових перцептронів MLP і залишкових зв’язків. У ній немає рекурентних або згорткових блоків, що традиційно використовуються для часових рядів. Завдяки цьому модель легко паралелізується й швидко навчається на GPU, оскільки всі шари – повнозв’язні.

Модель застосовує інноваційну архітектуру «подвійних залишкових зв’язків». Кожен блок генерує два виходи – власний прогноз (forward forecast) і зворотний прогноз (backcast), – і видаляє з вхідного сигналу ту його частину, яку зміг апроксимувати. Решта сигналу передається наступному блоку, що полегшує послідовне навчання глибокої моделі (рис. 2.6) [18].

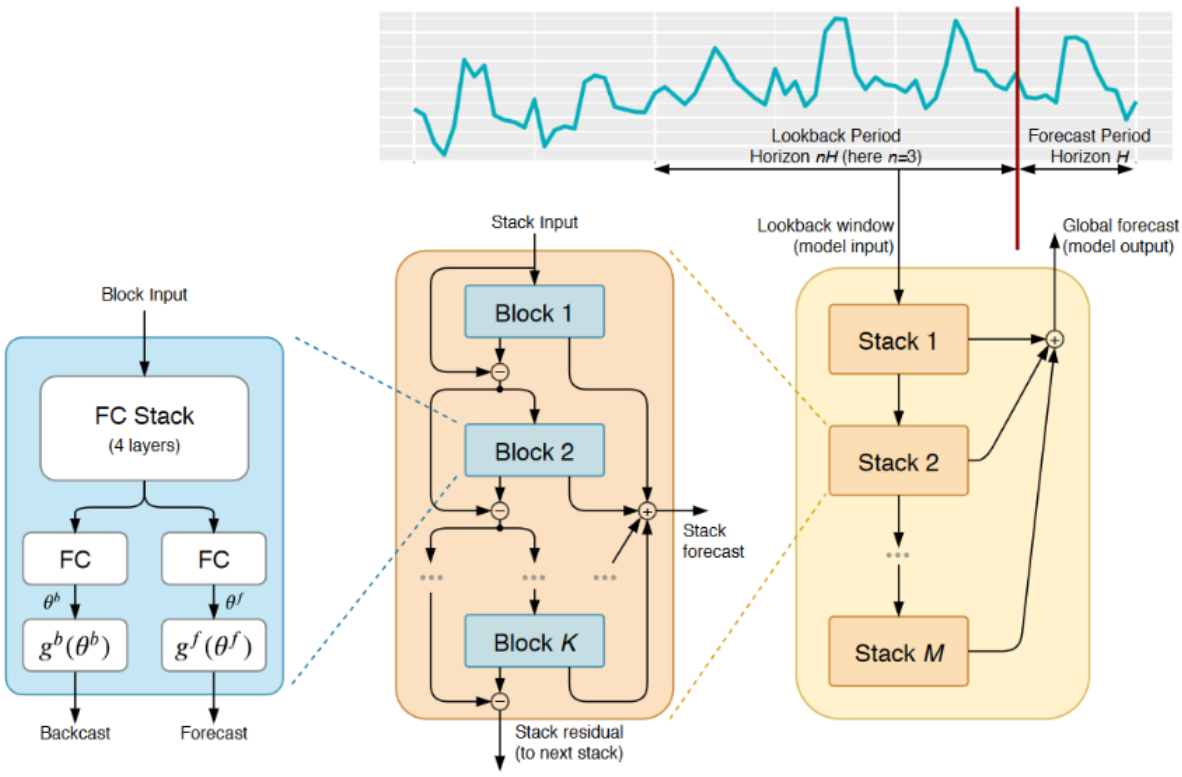

Рисунок 2.6 − Схематичне зображення роботи моделі N-BEATS [18]

Спочатку модель розділяє часовий ряд на період огляду (lookback) і період прогнозу (forecast). Для побудови прогнозу модель використовує період огляду. Довжина періоду огляду є кратною довжині періоду прогнозу. Оптимальне кратне зазвичай знаходиться в межах від двох до шести.

Далі модель N-BEATS складається з багатошарових стеків. Кожен стек, у свою чергу, складається з блоків, розташованих один над одним.

Кожен блок має розгалужену структуру. Один гілка блоку генерує зворотний прогноз, інша — прогноз на основі деяких вхідних даних. Прогноз — це передбачення ще не спостережуваних значень. Зворотний прогноз показує, наскільки добре модель "підлаштовується" під наявні вхідні дані.

Для отримання backcast і forecast, спочатку вхідні дані проходять через повнозв’язну нейромережу з чотирма шарами. Ця мережа генерує коефіцієнти розкладу для backcast і forecast. Ці коефіцієнти надходять у дві гілки — одну для зворотного прогнозу, іншу для прямого. У кожній гілці виконується

базисне розширення — саме тут і відбувається власне "нейронне базисне розширення" [18].

Як видно з рисунку 2.6, N-BEATS з’єднує кілька блоків у стеках. Оскільки кожен блок повертає і зворотний прогноз, і прогноз, відбуваються дві речі.

1. Модель додає частковий прогноз кожного блоку до загального прогнозу стеку.
2. Модель віднімає зворотний прогноз блоку з його вхідних даних. Таким чином, кожен наступний блок отримує лише залишкову частину сигналу, яку попередній блок не зміг пояснити. Тобто кожен блок намагається апроксимувати лише окрему локальну частину сигналу.

Після цього модель накладає один стек на інший. Як і блоки в стеках, кожен наступний стек (окрім першого) тренується на залишках від попереднього. Це дозволяє кожному стеку виявляти глобальні шаблони, які раніше не були захоплені. Остаточний прогноз — це сума прогнозів усіх стеків, що забезпечує ієрархічну декомпозицію.

Таким чином, N-BEATS застосовує подвійний залишковий підхід. Зворотні прогнози формують зворотні залишки, а прямі — прямі залишки. Багатошарова архітектура блоків і стеків приводить до накопичення цих залишків. Завдяки цій структурі N-BEATS здатна відтворювати поведінку класичних статистичних моделей, зберігаючи гнучкість нейронних мереж.

2.2 Методи управління запасами

2.2.1 Top-Down forecast

Top-down forecasting — це метод прогнозування, який починається з аналізу загального ринку або галузі, а потім поступово деталізується до

конкретних підрозділів компанії. Цей підхід дозволяє компаніям встановлювати стратегічні цілі, ґрунтуючись на загальних ринкових тенденціях та економічних показниках.

У контексті продажів, top-down прогнозування часто використовується для швидкої оцінки потенційного доходу. Наприклад, компанія може визначити загальний обсяг ринку та свою очікувану частку, щоб спрогнозувати майбутні продажі. Цей метод особливо корисний для нових компаній або при запуску нових продуктів, коли історичні дані обмежені.

Переваги top-down підходу включають швидкість та простоту у впровадженні, а також можливість узгодження прогнозів з загальними бізнес-цілями. Однак, цей метод може не враховувати специфіку окремих підрозділів або продуктів, що може призвести до менш точних прогнозів.

У порівнянні з bottom-up прогнозуванням, яке базується на деталізованих даних знизу вгору, top-down підхід надає загальну картину, але може бути менш точним у деталях. Тому багато компаній використовують комбінацію обох методів для досягнення більш збалансованого та точного прогнозування.

Загалом, top-down прогнозування є ефективним інструментом для стратегічного планування та встановлення загальних напрямків розвитку компанії, особливо в умовах обмеженої інформації або швидких змін на ринку.

Основна формула top-down прогнозування:

$$\textit{Прогнозований дохід} = \textit{Розмір ринку} \times \textit{Очікувана частка ринку}.$$

2.2.2 XYZ аналіз

XYZ-аналіз — це метод класифікації товарів за рівнем передбачуваності попиту, який широко використовується в управлінні запасами та плануванні попиту. На відміну від ABC-аналізу, який фокусується на вартості товарів,

XYZ-аналіз враховує стабільність та варіативність споживання. Цей підхід дозволяє компаніям краще розуміти поведінку попиту та ефективніше керувати запасами [16].

У рамках XYZ-аналізу товари поділяються на три категорії.

1. Категорія X містить товари зі стабільним і передбачуваним попитом. Їх споживання змінюється незначно, що дозволяє точно прогнозувати потреби та мінімізувати рівень страхових запасів.
2. В категорію Y попадають товари з помірною варіативністю попиту. Зміни в споживанні можуть бути пов'язані з сезонністю, акціями або іншими передбачуваними факторами. Для таких товарів потрібне гнучке управління запасами.
3. Категорія Z для товарів з високою варіативністю та непередбачуваним попитом. Їх споживання важко прогнозувати, що може призводити до надлишкових запасів або дефіциту. Для таких товарів доцільно використовувати стратегії, що мінімізують ризики, наприклад, замовлення за попереднім запитом.

2.2.3 Розрахунок оптимальних параметрів запасів

Оптимальний розмір замовлення (EOQ) — це оптимальна кількість товару, яку слід замовляти, щоб мінімізувати загальні витрати на замовлення та зберігання запасів.

$$EOQ = \sqrt{\frac{2DS}{H}}, \tag{2.4}$$

де D — річний попит (кількість одиниць);

S — вартість одного замовлення;

H — вартість зберігання однієї одиниці товару на рік.

Метою формули (2.4) є визначення оптимальної кількості одиниць товару для замовлення. Якщо це досягнуто, компанія може мінімізувати витрати на закупівлю, доставку та зберігання одиниць товару. Формулу EOQ можна модифікувати для визначення різних рівнів виробництва або інтервалів замовлення, і корпорації з великими ланцюгами постачання та високими змінними витратами використовують алгоритми в програмному забезпеченні для розрахунку EOQ [11].

Для багатьох компаній запаси є найбільшим активом після людських ресурсів, і такі підприємства повинні мати достатню кількість запасів для задоволення потреб клієнтів. Якщо формула EOQ допомагає мінімізувати рівень запасів, зекономлені кошти можуть бути використані для інших бізнесових цілей або інвестицій.

Формула EOQ визначає точку повторного замовлення для компанії. Коли рівень запасів знижується до певного порогу, застосування EOQ у бізнес-процесах автоматично сигналізує про необхідність оформлення нового замовлення на поповнення.

Точка повторного замовлення (ROP) — це рівень запасів, при досягненні якого слід ініціювати нове замовлення, щоб уникнути дефіциту товару.

$$ROP = d \times L + SS,$$

де d — середній щоденний попит;

L — час виконання замовлення (у днях);

SS — страховий запас.

Визначивши точку повторного замовлення, компанія уникає ситуацій нестачі запасів і може продовжувати виконувати замовлення клієнтів. Якщо запаси вичерпуються, виникають витрати через дефіцит — тобто втрачені доходи через неможливість виконати замовлення. Крім того, дефіцит запасів

може призвести до втрати клієнтів або до того, що клієнти надалі робитимуть менше замовлень.

Ця метрика дозволяє бізнесу приймати швидкі, малостресові та обґрунтовані на даних рішення щодо замовлення товарів, без необхідності кожного разу починати з нуля. Простий підхід, заснований на правилах, економить час і зменшує ймовірність дорогих помилок в управлінні запасами [19].

Використання точки повторного замовлення для запуску процесу поповнення запасів допомагає бізнесу працювати ефективніше, балансуючи між двома суперечливими потребами. Якщо компанія замовляє занадто багато і надто рано, вона витрачає кошти раніше, ніж це необхідно, і водночас несе витрати на зберігання зайвих запасів, частина з яких може ніколи не бути проданою (особливо якщо йдеться про товари на завершальному етапі життєвого циклу). З іншого боку, якщо компанія затягує з повторним замовленням або чекає, поки запаси повністю вичерпаються, затримки між моментом замовлення та отриманням товарів призводять до відсутності товару на складі (тобто ситуацій, коли компанія не може обслуговувати клієнтів або не виконує замовлення) [19].

Страховий запас — це середній обсяг запасів, який зберігається на складі для врахування короткострокових невизначеностей, пов’язаних зі змінами попиту та постачання. Простіше кажучи, страховий запас — це резерв товарів, який підтримується для запобігання ситуаціям дефіциту продукції на складі та незаповнених замовлень.

$$SS = Z \times \sigma_{LT} \times \sqrt{L},$$

де Z — коефіцієнт сервісного рівня;

σ_{LT} — стандартне відхилення попиту під час часу виконання замовлення;

L — час виконання замовлення (у днях).

Страховий запас допомагає бізнесу підтримувати високий рівень обслуговування клієнтів, зменшуючи ймовірність виникнення дефіциту товарів. Коли клієнти постійно знаходять потрібні їм товари в наявності, вони з більшою ймовірністю матимуть позитивний досвід і залишатимуться лояльними до бренду. Навпаки, відсутність товару на складі може призвести до втрати продажів, розчарування клієнтів і зіпсованої репутації на ринку [20].

Крім того, страховий запас допомагає компаніям справлятися з природною невизначеністю попиту і пропозиції. На доступність і терміни постачання сировини або готової продукції можуть впливати багато факторів — від несподіваного зростання кількості замовлень клієнтів до перебоїв у ланцюжку постачання, таких як стихійні лиха чи страйки. Додаткові запаси слугують буфером проти таких ризиків, дозволяючи компаніям мінімізувати операційні та фінансові наслідки таких збоїв.

Крім того, страховий запас може допомогти компаніям оптимізувати загальні практики управління запасами. Аналізуючи чинники, що зумовлюють потребу у страховому запасі — такі як мінливість термінів постачання, точність прогнозування попиту і цільові рівні обслуговування — компанії можуть виявити можливості для підвищення ефективності та гнучкості свого ланцюга постачання. Це може включати такі стратегії, як диверсифікація постачальників, вдосконалення процесів планування попиту або інвестування в більш сучасні технології управління запасами.

Висновки до розділу 2

Прогнозування попиту та управління запасами ґрунтується на поєднанні статистичних підходів, машинного навчання й нейромережевих архітектур, кожен із яких має власну сферу ефективного застосування. Статистичні моделі, як-от ARIMA і SARIMA, забезпечують добру точність у стабільних часових

рядах із вираженою сезонністю та тенденціями. Вони відзначаються простотою реалізації й інтерпретованістю результатів, однак мають обмежену гнучкість у випадках складної динаміки або структурних зламів у даних.

Машинне навчання (Random Forest, XGBoost) дає змогу працювати з великою кількістю предикторів і фіксувати складні нелінійні залежності. Random Forest виявляє стійкість до перенавчання за рахунок ансамблевої структури й дозволяє узагальнювати дані навіть за умови високої варіативності. XGBoost підсилює точність прогнозу завдяки ітеративному навчанню моделей на залишкових помилках та потужним засобам регуляризації.

Нейромережеві підходи, зокрема LSTM та N-BEATS, демонструють найвищу адаптивність до змінних структур попиту. LSTM дозволяє враховувати довготривалі залежності в послідовностях завдяки використанню комірок пам’яті та вентилів керування інформаційним потоком. Вона ефективна в ситуаціях, де важливо не втратити контекст попередніх подій при прогнозуванні майбутніх значень. N-BEATS, натомість, пропонує оригінальний підхід без рекурентності, що робить її придатною для паралельного обчислення і дозволяє точно аппроксимувати часові ряди через багаторівневу декомпозицію сигналу.

В управлінні запасами важливу роль відіграють і структурні підходи до організації планування. Top-down прогнозування надає змогу будувати прогнози на основі макрорівня, узгоджуючи їх із бізнес-цілями, що особливо корисно при обмеженій деталізації історичних даних. XYZ-аналіз допомагає класифікувати товари за стабільністю попиту, що дозволяє будувати диференційовану стратегію запасів для різних груп продукції — від передбачуваних до абсолютно нестабільних.

Розрахунок оптимальних параметрів запасів базується на формалізованих підходах, що дозволяють збалансувати витрати на зберігання, замовлення та ризики дефіциту. Концепції EOQ, ROP та страхового запасу забезпечують математично обґрунтовані точки прийняття рішень, дозволяючи мінімізувати витрати й підтримувати високий рівень обслуговування. Умови невизначеності

постачання або попиту враховуються за допомогою введення страхового запасу, що виступає буфером для зниження ризиків втрати продажів.

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ ПРОГНОЗУВАННЯ ПОПИТУ ДЛЯ УПРАВЛІННЯ ЗАПАСАМИ

3.1 Аналіз та обробка даних

У цій роботі використовується набір даних "Superstore Sales", доступний на платформі Kaggle. Набір даних є універсальною та комплексною збіркою даних про продажі. Він включає в себе наступні 21 поле.

1. Row ID – Унікальний ідентифікатор рядка в таблиці.
2. Order ID – Унікальний ідентифікатор замовлення.
3. Order Date – Дата оформлення замовлення.
4. Ship Date – Дата відправлення замовлення
5. Ship Mode – Спосіб доставки.
6. Customer ID – Унікальний ідентифікатор клієнта.
7. Customer Name – Ім’я та прізвище клієнта.
8. Segment – Сегмент ринку, до якого належить клієнт.
9. Country – Країна, в яку доставляється замовлення.
10. City – Місто доставки.
11. State – Штат або область доставки.
12. Postal Code – Поштовий індекс доставки.
13. Region – Регіон (наприклад, Захід, Схід тощо).
14. Product ID – Унікальний ідентифікатор товару.
15. Category – Категорія товару.
16. Sub-Category – Підкатегорія товару.
17. Product Name – Повна назва товару.
18. Sales – Сума продажу.
19. Quantity – Кількість одиниць товару, що були продані.
20. Discount – Знижка, надана на товар.
21. Profit – Прибуток, отриманий від продажу товару.

Така структура даних дає змогу не лише виявити закономірності у поведінці покупців, а й побудувати прогнози для оптимізації управління запасами на основі попередніх трендів.

Часовий ряд, представлений на рисунку 3.1 демонструє дані з 2011 по 2015 роки, де спостерігається загальна тенденція зростання кількості продажів. Це підтверджується графіком трендової компоненти, де чітко видно плавне і стійке зростання показників протягом усього періоду. Така тенденція може бути пов’язана з сезонними факторами, зміною попиту або іншими довгостроковими процесами, які впливають на ринок.

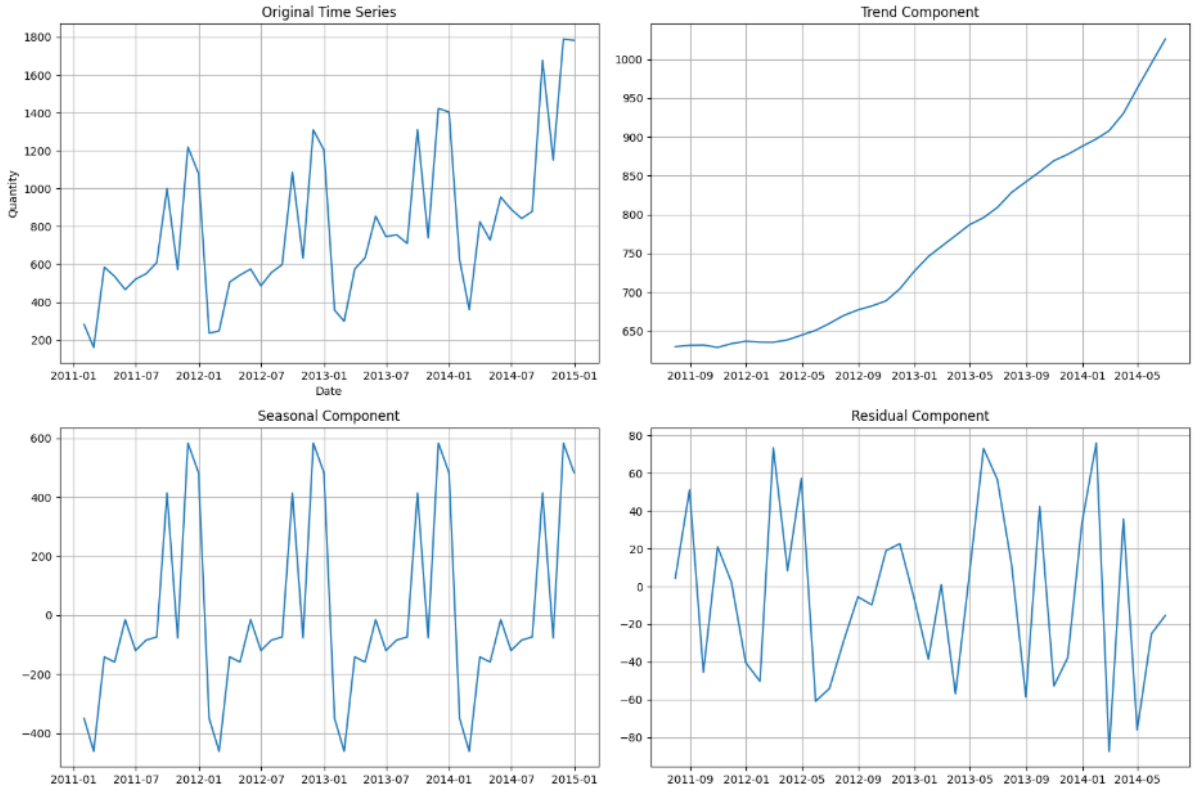

Рисунок 3.1 – Декомпозиція часових рядів

Сезонна компонента (рис. 3.1) демонструє повторюваний характер змін із певною періодичністю: можна помітити виражені піки та спади, що повторюються приблизно раз на рік. Це свідчить про наявність сезонного фактору в даних, наприклад, коливань попиту залежно від часу року, святкових

періодів або інших регулярних подій. Глибина цих коливань досить значна, що вказує на суттєвий вплив сезонності на поведінку часового ряду.

Залишкова компонента (рис. 3.1) відображає випадкові коливання, які залишаються після виділення тренду та сезонності. Вона містить як позитивні, так і негативні відхилення, що можуть бути пов’язані з неочікуваними подіями або випадковими факторами, які не враховані трендовою чи сезонною моделями. Розподіл залишкової компоненти є досить хаотичним, що вказує на відсутність систематичних патернів у цих відхиленнях. Проте деякі періоди демонструють помітні стрибки, які можуть сигналізувати про аномальні зміни, наприклад, вплив зовнішніх факторів, непередбачуваних ситуацій або змін у ринку.

Після імпорту набору даних проведено первинний аналіз структури даних, зокрема перевірка наявності пропущених значень. Результати аналізу засвідчили, що у наборі відсутні пропущені значення, що дає змогу уникнути необхідності їх обробки або заповнення. Наступним етапом є очищення даних від дублікатів.

Для подальшої підготовки даних здійснено перетворення стовпця Order Date у формат datetime, що дало змогу ефективно працювати з часовими мітками. Перетворення дат дозволило здійснити групування записів за місяцями — на основі періоду Year_Month — що забезпечило зручну агрегацію обсягів продажу в розрізі часу.

Для підготовки даних до побудови моделей машинного навчання, таких як Random Forest та XGBoost, реалізовано комплексний підхід до генерації ознак (feature engineering). На основі часових міток (Year_Month) створено набір календарних ознак: місяць, квартал та рік. Ці змінні дає змогу моделі виявити сезонні закономірності та довгострокові тенденції, характерні для циклів продажу.

Оскільки календарні ознаки, як місяць чи квартал, мають циклічну природу, для збереження інформації про їхню повторюваність застосовано синусоїдальні та косинусоїдальні перетворення (наприклад, month_sin,

month_cos, quarter_sin, quarter_cos). Таке представлення дає змогу моделі "відчути", що грудень і січень — близькі у часовому вимірі, хоча їхні числові значення (12 і 1) сильно відрізняються. Додатково були сформовані бінарні індикатори початку і кінця року або кварталу, які можуть бути важливими при прогнозуванні обсягів у періоди звітності або сезонних піків.

Для кращого врахування динаміки часового ряду згенеровані лагові ознаки (quantity_lag_1, ..., quantity_lag_12), що містять значення Quantity за попередні місяці. Вони надають моделі інформацію про історію змін попиту. Також обчислювалися ковзні статистики — середнє, стандартне відхилення, максимум, мінімум і медіана для вікон розміром 2, 3, 6 та 12 місяців. Ці ознаки уможливлюють виявлення локальної тенденції, варіативність і аномалії у динаміці продажів.

Окрім цього, створено набір похідних ознак, що описують темп змін у даних: відносні зміни, абсолютні дельти, імпульс та прискорення. Такі ознаки особливо корисні для моделей, що прогнозують зміни, а не абсолютні значення. Для врахування сезонних ефектів було додано ознаку quantity_yoy, яка вимірює зміну обсягу продажів у порівнянні з тим самим місяцем попереднього року.

У процесі побудови моделі прогнозування загальних помісячних продажів прийнято рішення не використовувати ознаки, що містяться безпосередньо у вихідному датасеті, такі як категорія товару, підкатегорія чи наявність знижок. Натомість акцент зроблено на ознаках, створених на основі самого часового ряду. Такий підхід обумовлено кількома ключовими причинами.

По-перше, завдання моделі полягає у прогнозуванні загальних помісячних продажів, тобто агрегованих показників на рівні всіх категорій товарів. У цьому випадку деталізовані категоріальні ознаки (наприклад, категорія чи підкатегорія товару) втрачають свою інформативність, оскільки підсумовування даних зводить їхню роль до мінімуму або взагалі нівелює.

По-друге, побудова моделі на основі часових ознак дозволяє підвищити її стабільність і стійкість до змін структури даних. Наприклад, у майбутньому

можуть змінитися категорії товарів, ввести нові або об’єднати наявні. Моделі, що залежать від таких змінних, ризикують втратити коректність прогнозу.

Додатково, важливим фактором є ризик витоку даних. Використання ознак, що можуть бути відомі лише після моменту прогнозу (наприклад, наявність знижки у конкретному місяці), може призвести до включення в модель інформації з майбутнього, що недоступна на момент прийняття рішення. Це створює ілюзію високої точності моделі на етапі навчання, але призводить до погіршення якості прогнозів у реальному використанні.

Після цього підготовлений набір даних було розділено на навчальну і тестову вибірки за часовим критерієм. Цільовою змінною для моделей була обрана Quantity, а всі інші ознаки формували матрицю ознак X, яку використовували для навчання і прогнозування.

Процес підготовки даних для моделі LSTM складається з кількох ключових етапів. Першим кроком є формування навчальних послідовностей для LSTM. Часовий ряд розбивається на послідовності фіксованої довжини. Для кожного фрагмента часу створюється вхідний масив ознак і відповідний вихід — цільове значення, яке потрібно передбачити. Таким чином, формується навчальна вибірка у форматі, що відповідає очікуванням рекурентної нейромережі.

Крім цього важливим етапом є нормалізація даних. Застосовується мінімаксне масштабування (Min-Max Scaling) до всіх вхідних ознак та цільової змінної окремо. Це критично важливо для моделей LSTM, оскільки вони чутливі до масштабів вхідних даних.

Підготовка даних для моделі N-BEATS має свою специфіку. В основі підготовки лежить створення послідовностей для навчання, орієнтованих на прогнозування часових рядів з фіксованими вікнами (lookback window) та горизонтом прогнозу (forecast horizon).

На першому етапі формуються вхідні та вихідні послідовності з одномірного масиву часових значень. Для кожної точки часу створюється "вікно" з lookback значень, яке подається як вхід до моделі, а цільовим

значенням виступає наступне (або декілька наступних) значення(нь) відповідно до заданого горизонту прогнозу.

Конструкція самої моделі N-BEATS реалізується через спеціалізовану функцію create_nbeats_block. Кожен блок моделі складається з кількох повнозв'язних шарів (Dense layers), які працюють як базові функції для апроксимації складових часового ряду. У залежності від конфігурації, блок може бути спрямований на моделювання тренду, сезонності або мати загальне призначення. Крім того, модель може використовувати спільні або незалежні параметри (theta) для зворотного відновлення (backcast) і прогнозу (forecast).

3.2 Побудова та тренування моделей

Перша модель, застосована для задачі прогнозування попиту є ARIMA.

Тестування стаціонарності є важливим етапом аналізу часових рядів, оскільки багато методів прогнозування, включаючи моделі ARIMA, вимагають роботи зі стаціонарними даними. Одним із основних інструментів для перевірки стаціонарності є тест Діккі-Фуллера , який дозволяє визначити, чи є ряд стаціонарним, тобто чи має він постійне середнє значення, дисперсію та автоковаріацію протягом часу.

Результати проведених тестів, на рисунку 3.2, показують, що оригінальний ряд є стаціонарним: значення ADF-статистики (-3.44) менше за критичні значення на рівні 5% (-2.925) та навіть 1% (-3.578), а p-value (0.009625) також підтверджує, що ряд стаціонарний.

```
=== Original Series ===
ADF Statistic: -3.441434
p-value: 0.009625
Critical Values:
        1%: -3.578
        5%: -2.925
        10%: -2.601
Series is stationary
```

Рисунок 3.2 – Результати тестування стаціонарності

Ключовий етап побудови моделі ARIMA — визначення параметрів (p, d, q). Для початку проведено побудову графіків автокореляційної функції (ACF) та часткової автокореляційної функції (PACF) (рис. 3.3), що дає змогу візуально оцінити значущі лаги. На графіках помітно, що перші лаги мають статистично значущі автокореляції, що є підказкою для вибору невеликих значень p та q.

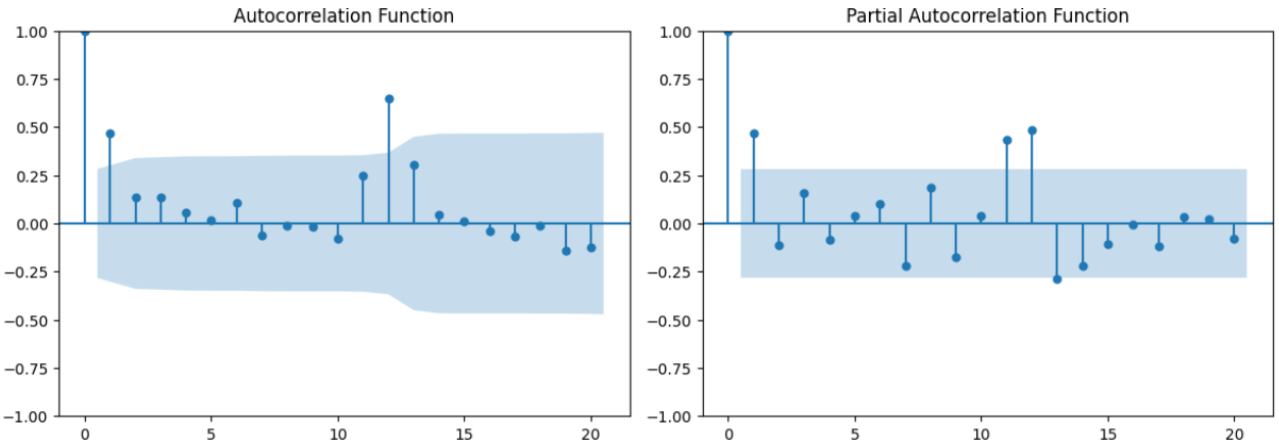

Рисунок 3.3 – Графіки автокореляційної функції та часткової автокореляційної функції

Наступним кроком проведено повний перебір сітки можливих комбінацій параметрів у заданих діапазонах: p = 0..4, d = 0..2, q = 0..4. Для кожної комбінації обчислювались інформаційні критерії AIC та BIC, що дозволяють

порівнювати моделі з різною кількістю параметрів. Модель з найменшим AIC вважається оптимальною, оскільки вона забезпечує компроміс між якістю підгонки та складністю моделі.

Результати пошуку показали, що модель ARIMA(1,2,4) має найменше значення AIC (679.81) серед протестованих варіантів.

Також обрано підхід SARIMA (Seasonal ARIMA), оскільки дані демонструють виражену сезонну компоненту з річною періодичністю. Несезонна частина параметрів для моделі залишиться такою самою як і для ARIMA.

У графіку ACF (рис. 3.3) спостерігається значний пік на лазі 12, що вказує на наявність сезонної залежності з періодом 12 місяців, після чого автокореляція поступово спадає, що є характерною ознакою сезонної нестаціонарності та обґрунтовує необхідність сезонного диференціювання (D=1). У графіку PACF (рис. 3.3) також присутній помітний пік на лазі 12, але відсутні інші значущі піки на сезонних лагах (24, 36 тощо), що свідчить про те, що одного сезонного диференціювання буде достатньо для усунення сезонної нестаціонарності без необхідності додавання сезонних AR або MA компонент (P=0, Q=0). Таким чином, модель SARIMA(0,1,0,12) є оптимальною, оскільки сезонне диференціювання з періодом 12 усуває сезонну залежність, а відсутність додаткових значущих сезонних кореляцій після лагу 12 підтверджує, що додаткові сезонні параметри не потрібні.

Також для вирішення задачі прогнозування використано модель Random Forest Regressor, реалізовану в бібліотеці scikit-learn.

У рамках моєї реалізації модель ініціалізовано з наступними параметрами: n_estimators=100, що означає побудову ста дерев у складі ансамблю, та max_depth=10, що обмежує максимальну глибину дерева для контролю складності моделі та запобігання перенавчанню. Фіксація випадковості (random_state=42) забезпечує відтворюваність результатів, а використання всіх доступних процесорних ядер (n_jobs=-1) дає змогу оптимізувати час навчання моделі.

Процес навчання моделі здійснювався на підготовлених даних: навчальна вибірка (X_train, y_train) використовувалася для побудови ансамблю дерев, а тестова вибірка (X_test) — для оцінки якості прогнозу. Після навчання модель здійснює прогнозування цільової змінної, використовуючи комбінацію результатів окремих дерев, де кожне дерево голосує за свій прогноз, а кінцевий результат визначається шляхом усереднення цих прогнозів.

Крім цього у рамках дослідження застосовано модель XGBoost. Це є потужною реалізацією алгоритму градієнтного бустингу, яка дозволяє будувати ансамблі рішень на основі дерев рішень.

Для побудови моделі обрані ключові параметри, що впливають на її продуктивність та стійкість до перенавчання. Зокрема, параметр n_estimators=100 визначає кількість дерев у ансамблі, а learning_rate=0.1 задає швидкість навчання, що контролює величину внеску кожного дерева в загальний прогноз. Параметри max_depth=4 та min_child_weight=2 впливають на складність кожного дерева: вони обмежують глибину дерева та мінімальну вагу для розбиття вузлів, що запобігає надмірній складності моделі. Додатково використовуються параметри subsample=0.8 та colsample_bytree=0.8, які відповідають за випадкову вибірку рядків та ознак відповідно, що дозволяє зменшити кореляцію між деревами та знижує ризик перенавчання. Для постановки задачі регресії використано цільову функцію objective='reg:squarederror', яка оптимізує середньоквадратичну помилку.

Модель навчалась на тренувальній вибірці, при цьому використовувався механізм раннього зупинення (early stopping) з обмеженням у 20 ітерацій без покращення, що дозволяє запобігти перенавчанню.

Крім того, проведена процедура налаштування гіперпараметрів за допомогою GridSearchCV для підвищення якості прогнозування. Підбір параметрів (n_estimators, learning_rate, max_depth, min_child_weight, subsample, colsample_bytree, gamma) проводився на основі 5-кратної крос-валідації. Такий підхід дозволив знайти оптимальну комбінацію параметрів, що забезпечує мінімальне значення RMSE на тестовій вибірці.

Модель навчалась на тренувальній вибірці, що містить незалежні ознаки (X_train) та відповідні цільові значення (y_train). Під час навчання модель поступово будує ансамбль рішень у вигляді дерев рішень, де кожне наступне дерево намагається мінімізувати помилки попередніх, покращуючи загальний прогноз. Це дає змогу моделі адаптуватися до складних залежностей у даних та формувати більш точні передбачення.

Також для розв’язання задачі прогнозування продажів застосовано рекурентну нейронну мережу на основі довготривалої короткочасної пам’яті LSTM.

Перший шар моделі має 64 нейрони і налаштований з параметром return_sequences=True. Це означає, що цей шар передає повну послідовність вихідних даних далі, а не лише останній часовий крок. Такий підхід важливий, коли наступний шар також повинен працювати з послідовностями, наприклад, ще одним LSTM-шаром.

Другий LSTM-шар містить 32 нейрони і обробляє послідовність, отриману від попереднього шару. Завдяки цьому шар здатний виділяти більш абстрактні та узагальнені патерни у часових рядах, що є критично важливим для завдання прогнозування. Він концентрується на останніх часових кроках, які містять найактуальнішу інформацію для передбачення майбутніх значень.

Щоб запобігти перенавчанню моделі, після кожного LSTM-шару додається Dropout-шар із ймовірністю 0.2. Dropout — це регуляризаційна техніка, яка випадковим чином «вимикає» частину нейронів під час кожної ітерації навчання. Це запобігає занадто сильному пристосуванню моделі до конкретних прикладів навчальних даних, підвищуючи її здатність до узагальнення на нових даних.

Вихідним шаром моделі є щільний шар із одним нейроном. Цей шар відповідає за формування фінального прогнозу, тобто видає єдине числове значення, яке є передбаченням кількості продажів на наступний період. Таким чином, модель будує послідовну обробку даних: від виявлення патернів у часових рядах до формування конкретного прогнозу для бізнес-задачі.

Компільована модель використовує функцію втрат середньоквадратичної помилки (MSE) — стандартний вибір для задач регресії. Оптимізація ваг мережі здійснюється за допомогою алгоритму Adam, що дозволяє ефективно адаптуватися до складної поверхні функції втрат і забезпечує швидку збіжність.

У моделі N-BEATS прогнозування здійснюється шляхом послідовного застосування блоків, кожен із яких складається з кількох повнозв’язних шарів. Ці шари формують приховане представлення даних, що дозволяє моделі захоплювати складні залежності та закономірності в часових рядах.

Реалізація моделі N-BEATS виконана на основі функціонального підходу в бібліотеці Keras. Основним елементом є функція create_nbeats_block(), яка створює окремий блок моделі. Ця функція будує послідовність із кількох повнозв’язних шарів із функцією активації ReLU, що дозволяє моделі формувати приховані представлення вхідних даних. Після формування прихованих шарів функція генерує дві головні компоненти: backcast для уточнення залишків та forecast для формування прогнозу на заданий горизонт. Особливістю реалізації є параметр layer_type, який дозволяє створювати блоки різного призначення — для моделювання трендової компоненти, сезонної складової або загального характеру. Це дає змогу адаптувати модель під особливості часових рядів, що прогнозуються.

Функція build_nbeats_model() формує повну архітектуру моделі N-BEATS, об'єднуючи кілька стеків, кожен із яких складається із заданої кількості блоків. Кожен стек може бути налаштований під певний тип даних, наприклад, окремо для трендової та сезонної складових. При побудові моделі використовується принцип залишків: після кожного блоку модель вираховує залишки між фактичним значенням та відновленим backcast, які потім слугують вхідними даними для наступного блоку. Прогнози всіх блоків підсумовуються для формування фінального прогнозу, що дозволяє моделі поступово уточнювати свої передбачення.

У моделі передбачено гнучке налаштування параметрів: кількість стеків, кількість блоків у стеці, кількість нейронів у шарах, кількість прихованих шарів

у кожному блоці, а також варіанти спільного або окремого навчання параметрів. Модель компілюється з використанням оптимізатора Adam і функції втрат середньоквадратичної помилки, що є стандартом для задач прогнозування часових рядів.

Кількість стеків і кількість блоків у кожному стеці визначають архітектуру моделі в цілому. Більша кількість блоків дозволяє моделі послідовно уточнювати прогноз за рахунок багаторазового оновлення залишків, а використання кількох стеків дозволяє розділяти різні компоненти часового ряду (наприклад, тренд і сезонність). Параметр sharing контролює, чи будуть ваги для генерації backcast і forecast спільними чи окремими. Вибір спільного навчання зменшує кількість параметрів моделі та може покращити її здатність до узагальнення, тоді як окреме навчання дає змогу моделі краще адаптуватися до складних структур у даних.

3.3 Аналіз та порівняння результатів

Результати моделювання з використанням ARIMA свідчать про обмежену ефективність цієї моделі для прогнозування наданого часового ряду. Значення середньоквадратичної помилки MSE становить 164,684.77, що є доволі високим показником. Це вказує на наявність суттєвих розбіжностей між фактичними значеннями та прогнозами моделі, особливо при наявності різких змін у ряді. Висока помилка свідчить про те, що модель не змогла належним чином адаптуватися до волатильності в даних.

Корінь з MSE RMSE становить 405.81, що означає, що середня похибка прогнозів дорівнює приблизно 406 одиницям. Це значення є значним, особливо в контексті, де абсолютні обсяги коливань попиту не є надто великими. Середня абсолютна похибка MAE дорівнює 358.96, що додатково підтверджує систематичну невідповідність прогнозів дійсності. Найбільше занепокоєння

викликає значення середня абсолютна відносна похибка MAPE, яке становить 44.42%. Такий високий рівень помилки означає, що в середньому модель відхилялася від реальних значень майже на половину, що є неприйнятним для практичного використання в більшості бізнес-сценаріїв.

Графічна інтерпретація на рисунку 3.4 підтверджує ці висновки. Прогнозна лінія ARIMA виглядає надто згладженою та не враховує різких підйомів і спадів у фактичних тестових даних. Це свідчить про те, що модель не враховує потенційну сезонність або не адаптується до раптових змін у тренді. Хоча прогноз загалом потрапляє в межі 95% довірчого інтервалу, сам інтервал є дуже широким, що сигналізує про високу невизначеність прогнозу та низьку надійність моделі.

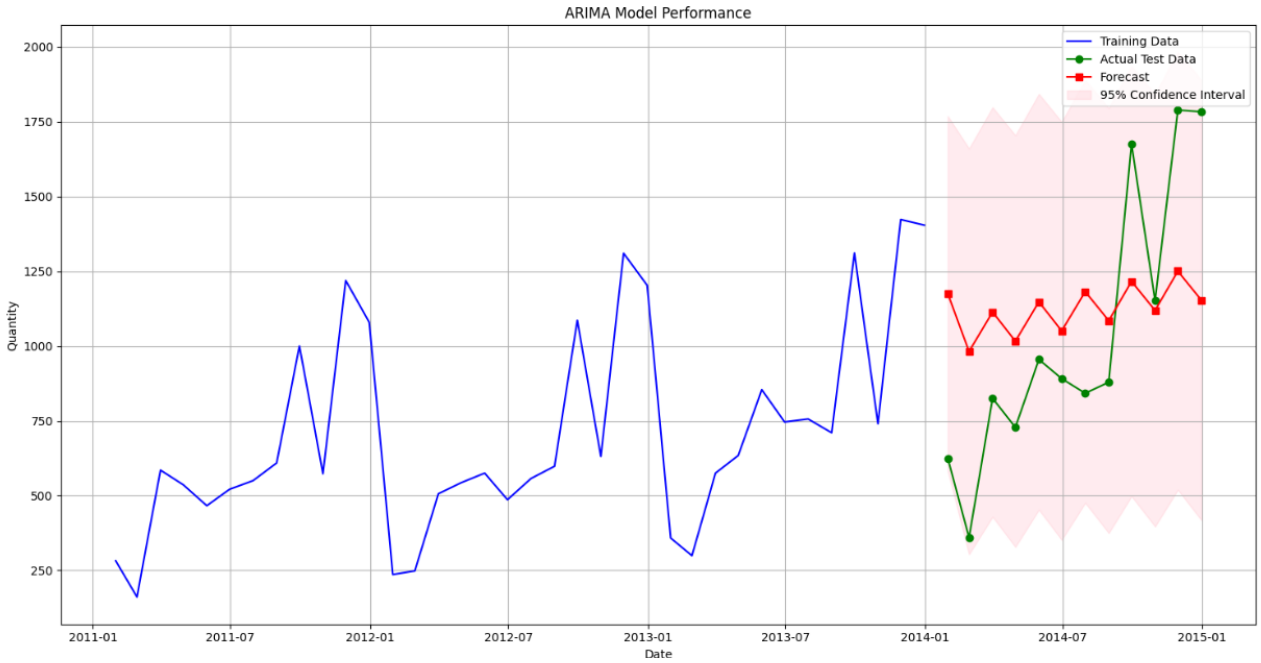

Рисунок 3.4 – Порівняння прогнозу ARIMA і фактичних значень

У підсумку можна стверджувати, що модель ARIMA в даному випадку є слабко адаптованою до природи досліджуваного часового ряду. Це може бути пов’язано з наявністю сезонності, яку проста ARIMA не враховує, або з високою нестабільністю в даних, яку модель не змогла захопити.

Результати моделювання з використанням сезонної моделі SARIMA свідчать про значне покращення якості прогнозування порівняно з попередньою моделлю ARIMA. Значення середньоквадратичної помилки MSE становить 15015.14 — це в кілька разів менше, ніж у попередньому випадку. Така різниця вказує на те, що модель набагато точніше захопила закономірності у даних, особливо з урахуванням сезонності.

Значення RMSE — 122.54, що є прийнятною похибкою для аналізованого ряду. Середня абсолютна похибка MAE дорівнює 103.84, що свідчить про стабільну й невелику похибку по всьому часовому горизонту прогнозу. Найбільш обнадійливим є показник MAPE — 13.69%.

Візуально на рисунку 3.5 видно, що прогноз моделі SARIMA досить чітко слідує за динамікою фактичних даних у тестовому наборі. Модель добре відтворює навіть різкі коливання та зростання попиту в окремі періоди 2014 року. Це свідчить про те, що додавання сезонної компоненти дозволило моделі краще врахувати повторювані патерни, які раніше не були враховані звичайною ARIMA.

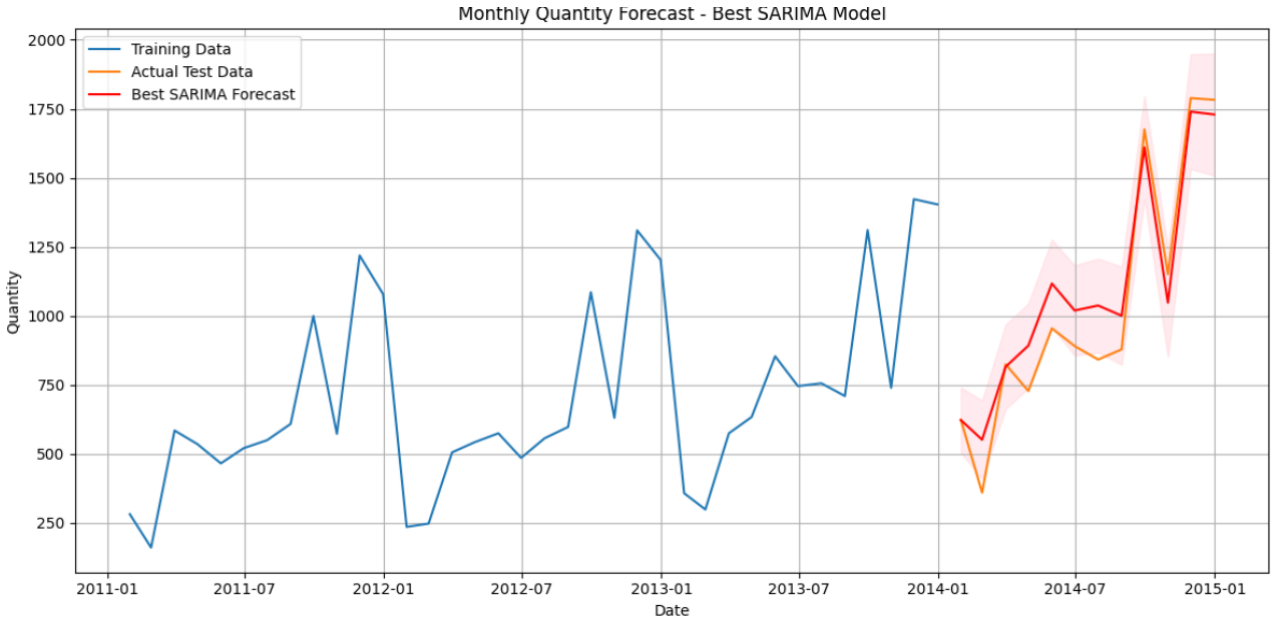

Рисунок 3.5 – Порівняння прогнозу SARIMA і фактичних значень

Результати моделювання з використанням Random Forest демонструють посередню якість прогнозування порівняно з SARIMA, але все ж кращу, ніж у класичної ARIMA. Середньоквадратична помилка MSE становить 58,754.94, що є значно нижчим за ARIMA, однак все ще вищим за SARIMA. Це свідчить про те, що Random Forest може вловлювати нелінійні залежності, проте йому бракує здатності точно передбачати сезонні та короткотермінові сплески, характерні для часових рядів.

Корінь середньоквадратичної помилки RMSE дорівнює 242.39. Хоча цей результат значно кращий, ніж у базової ARIMA-моделі, він все ще суттєво гірший за SARIMA. Середня абсолютна помилка MAE становить 189.32, що вказує на певну стабільність прогнозу, хоча й з меншою точністю. MAPE дорівнює 15.63%, що вже виходить за межі бажаного інтервалу для точних прогнозів, але лишається прийнятним для багатьох бізнес-застосувань.

Графік візуалізації на рисунку 3.6 підтверджує ці висновки: прогнозовані значення в більшості випадків повторюють загальну динаміку фактичних значень, але спостерігається помітне згладжування. Модель не може точно відтворити різкі зростання продажів у другій половині 2014 року. Особливо це помітно у вересні та листопаді, де модель суттєво недооцінює реальні значення.

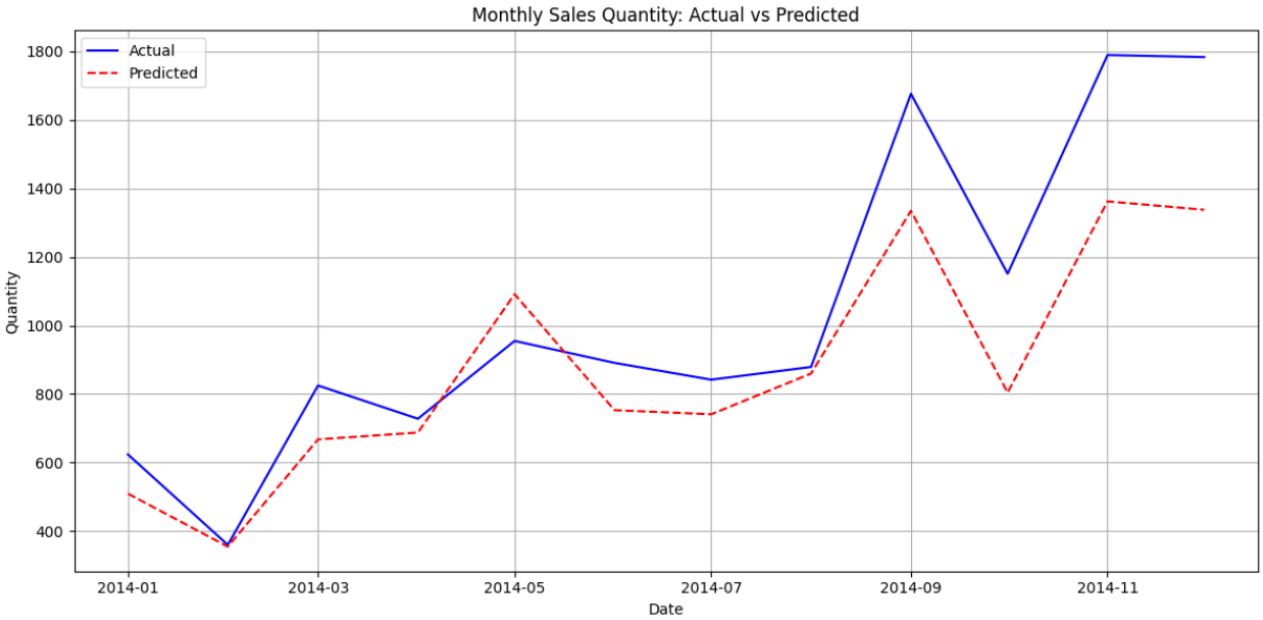

Рисунок 3.6 – Порівняння прогнозу Random Forest і фактичних значень

Загалом, Random Forest показує адекватний рівень прогнозування для задач, де сезонність не є критично важливою, або де необхідно швидко отримати робоче рішення без складного налаштування параметрів. Проте для задач із чіткою сезонною структурою, як у даному випадку, перевага залишається за спеціалізованими часовими моделями, такими як SARIMA.

Модель XGBoost показала кращі результати порівняно з Random Forest, проте дещо поступається SARIMA. Середньоквадратична помилка MSE склала 42,763.72, що свідчить про нижчу варіативність помилок порівняно з Random Forest та значне поліпшення у відтворенні складних патернів. Корінь середньоквадратичної помилки RMSE дорівнює 206.79, що підтверджує помірне середнє відхилення моделі — помітно краще, ніж у Random Forest, але ще не досягає точності SARIMA.

Середня абсолютна помилка MAE у XGBoost дорівнює 162.30. Показник MAPE на рівні 13.42% свідчить про задовільну якість прогнозу — модель загалом добре відтворює тренд і сезонні коливання, але деякі пікові значення залишаються складними для точного передбачення.

На рисунку 3.7 видно, що модель XGBoost досить точно повторює загальну форму реальних значень, хоча дещо згладжує різкі стрибки, зокрема у вересні та листопаді 2014 року. Прогнозована крива має меншу амплітуду, ніж фактична, що є типовою поведінкою для моделей градієнтного бустингу, які часто схильні до "обережного" прогнозування.

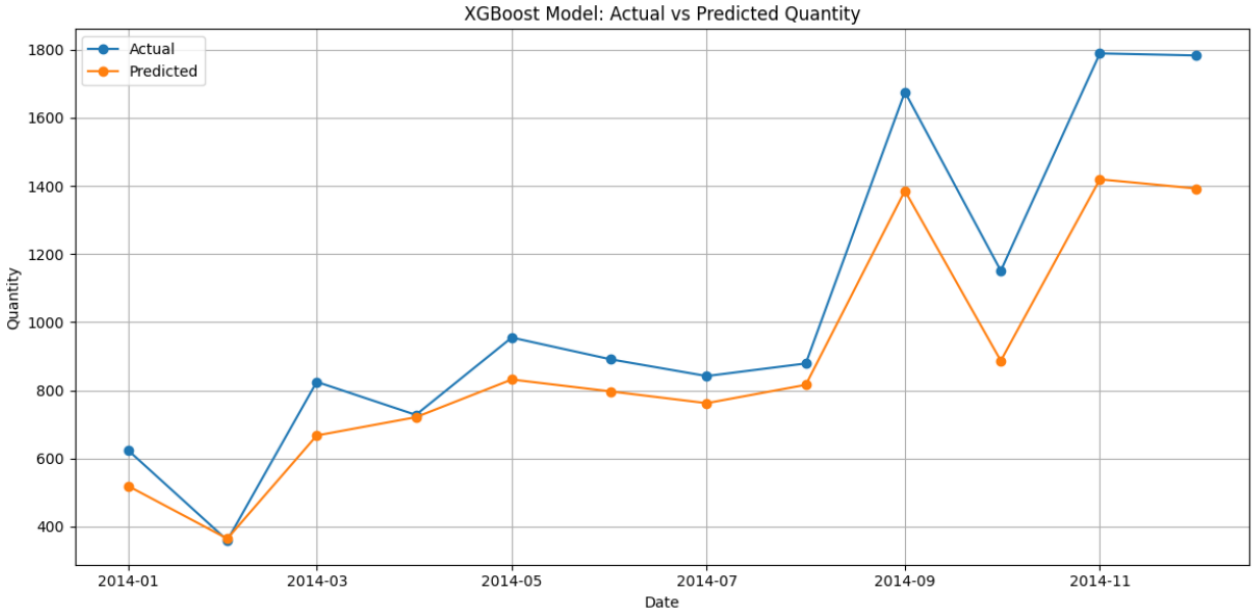

Рисунок 3.7 – Порівняння прогнозу XGBoost і фактичних значень

Аналіз результатів роботи моделі LSTM показує досить високу точність прогнозування з середньою абсолютною процентною похибкою MAPE у 8.42%. Це свідчить про те, що модель в середньому відхиляється від фактичних значень менше ніж на 9%, що є досить гарним результатом для більшості практичних застосувань. Середня абсолютна похибка MAE становить 98.56 одиниць, що при масштабі даних від 200 до 1800 одиниць представляє відносно невелику похибку.

Візуальний аналіз графіка, зображеного на рисунку 3.8, демонструє, що модель успішно відтворює загальні тренди та сезонні коливання у тестових даних. Особливо помітно, що LSTM добре справляється з прогнозуванням періодичних підйомів і спадів, які характерні для тренувальних даних. Модель коректно передбачила значний стрибок у кінці тестового періоду, хоча і з невеликим запізненням - прогноз показує пік на один період пізніше, ніж він фактично відбувся.

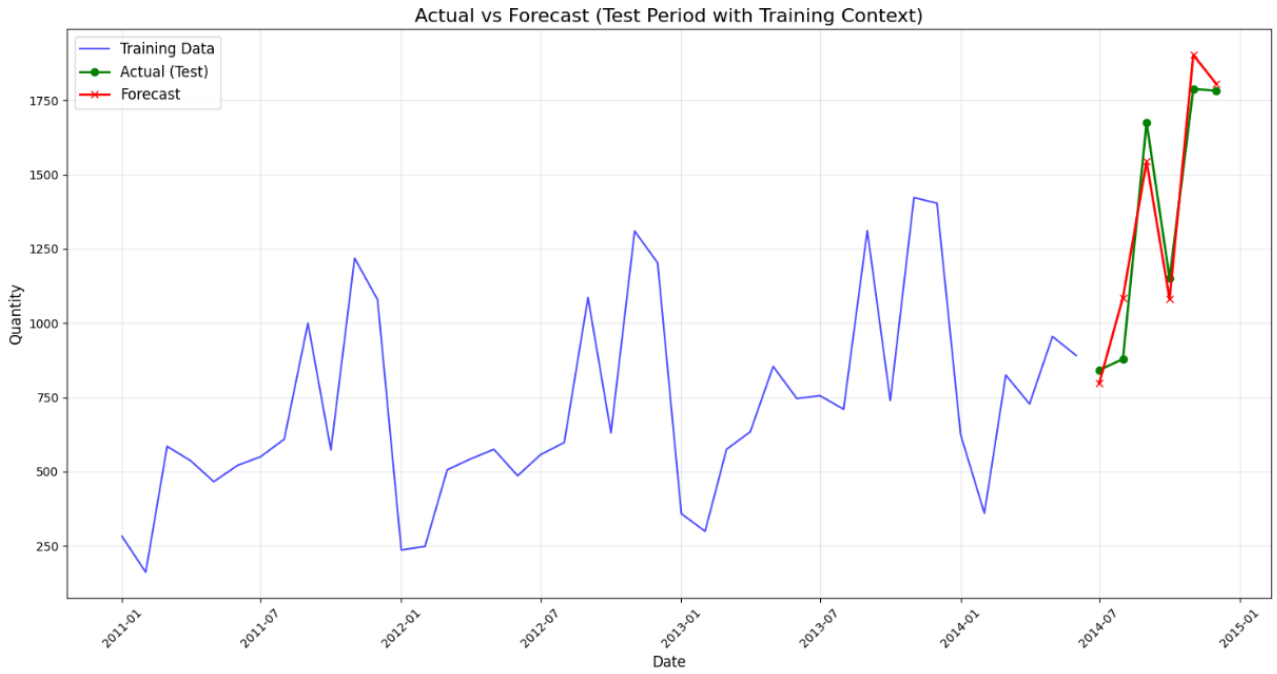

Рисунок 3.8 – Порівняння прогнозу LSTM і фактичних значень

Найбільш складним для моделі виявився момент різкого зростання в кінці тестового періоду, де спостерігається найбільше відхилення між прогнозом і фактичними значеннями. Це типова ситуація для LSTM-моделей, які можуть мати труднощі з передбаченням екстремальних значень, що виходять за межі діапазону тренувальних даних. Тим не менш, модель швидко адаптувалася і наступний прогнозний пункт вже показує значно кращу відповідність фактичним даним, що свідчить про здатність моделі до самокорекції.

Аналіз результатів моделі N-BEATS демонструє схожу з LSTM точність прогнозування з середньою абсолютною процентною похибкою MAPE у 8.50%. Це свідчить про стабільну продуктивність моделі з відхиленням від фактичних значень менше ніж на 9%, що є конкурентоспроможним результатом. Середня абсолютна похибка MAE становить 98.75 одиниць, що практично ідентично до результатів LSTM, однак RMSE у 119.44 вказує на дещо більшу чутливість до екстремальних відхилень.

Візуальний аналіз графіка на рисунку 3.9 показує, що N-BEATS модель демонструє більш консервативний підхід до прогнозування порівняно з LSTM.

Модель успішно відтворює загальний тренд зростання протягом тестового періоду, але має тенденцію до згладжування екстремальних піків і спадів. Особливо це помітно у вересні 2014 року, де фактичне значення досягає піку близько 1680 одиниць, тоді як прогноз N-BEATS показує більш помірне зростання до 1500 одиниць.

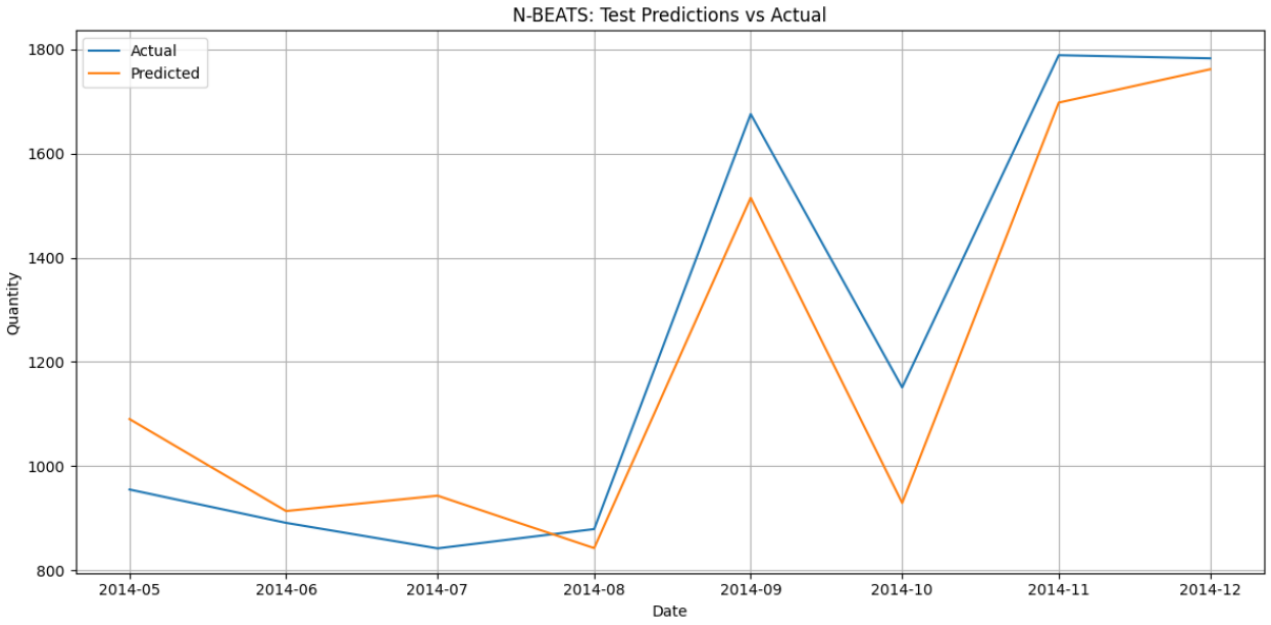

Рисунок 3.9 – Порівняння прогнозу N-BEATS і фактичних значень

Найбільш показовою особливістю роботи N-BEATS є її здатність до стабільного відстеження довгострокового тренду, що особливо добре видно в кінці тестового періоду. Модель коректно передбачила загальне зростання до рівня близько 1750 одиниць, хоча і не змогла точно відтворити різкі коливання. Це характерна риса N-BEATS архітектури, яка розкладає часовий ряд на трендові та сезонні компоненти, що робить її особливо ефективною для захоплення довгострокових патернів, але іноді призводить до недооцінки короткострокової волатильності. Загалом, модель показує надійну та передбачувану поведінку, що може бути перевагою в сценаріях, де стабільність прогнозів важливіша за точне відтворення всіх флуктуацій.

З таблиці 3.1 видно, що найкращий результат показують моделі LSTM та N-BEATS.

Таблиця 3.1 – Порівняння метрик прогнозування всіх моделей

Метрики	*ARIMA*	*SARIMA*	*Random Forest*	*XGBoost*	*LSTM*	*N-BEATS*
MSE	164684.77	15015.14	58754.94	42763.72	13458.74	14266.34
RMSE	405.81	122.54	242.39	206.79	116.0118	119.44
MAE	358.96	103.84	189.32	162.30	98.5638	98.75
MAPE	44.42%	13.69%	15.63%	13.42%	8.42%	8.50%

Порівнюючи ці дві моделі, LSTM демонструє незначно кращі результати за більшістю ключових метрик. LSTM має нижчі значення MSE (13458.74 проти 14266.34), RMSE (116.01 проти 119.44) та MAPE (8.42% проти 8.50%), що свідчить про дещо вищу точність прогнозування. У порівнянні з N-NEATS різниця невелика, але стабільна у всіх показниках, що вказує на системну перевагу LSTM для цього конкретного набору даних.

Для практичного застосування в управлінні запасами варто враховувати не лише точність, а й характер помилок. LSTM краще відтворює різкі коливання та екстремальні значення, що видно з графіків (рис. 3.7) — модель успішно передбачила різкий стрибок у кінці періоду, хоча і з невеликим запізненням. N-BEATS, навпаки, демонструє більш консервативний підхід зі згладжуванням піків, що може призводити до недооцінки попиту під час різких зростань.

3.4 Застосування результатів прогнозування для управління запасами

Основу прогнозування складає ієрархічний підхід, при якому загальний прогноз попиту поступово деталізується від вищих рівнів до нижчих. Спочатку система розраховує історичні частки кожної категорії у загальному обсязі продажів, потім аналогічно визначає частки підкатегорій всередині категорій та, нарешті, частки окремих SKU всередині підкатегорій. Цей метод забезпечує консистентність прогнозів на всіх рівнях та дозволяє контролювати відповідність суми деталізованих прогнозів загальному прогнозу.

Ключовою особливістю системи є інтеграція XYZ-аналізу для класифікації товарів за рівнем варіативності попиту. Система автоматично розраховує коефіцієнт варіації для кожного SKU на основі історичних даних продажів та класифікує їх. Ця класифікація використовується для коригування прогнозних часток: для товарів з високою варіативністю система застосовує більш консервативні оцінки, зменшуючи їх прогнозні частки для мінімізації ризику надлишкових запасів.

Фінальним етапом є розрахунок ключових метрик управління запасами для кожного SKU. Система обчислює оптимальний розмір замовлення EOQ на основі економічної моделі, що балансує витрати на замовлення та утримання запасів. Розрахунок точки повторного замовлення ROP враховує очікуваний попит протягом часу поставки та страховий запас, який визначається на основі заданого рівня сервісу та варіативності попиту. Система також передбачає диференційовані параметри для різних XYZ-класів, включаючи різні рівні сервісу, витрати на замовлення та утримання запасів.

Практична реалізація включає гнучку систему параметрів, що дозволяє легко налаштовувати ключові змінні (час поставки, рівень сервісу, витрати) для різних категорій товарів. Результати представляються у вигляді структурованих

звітів з можливістю фільтрації та експорту у CSV-формат, що забезпечує зручність використання для прийняття управлінських рішень.

Аналіз розподілу попиту показав, що товари Y-класу генерують найвищий сукупний річний попит (10,580 одиниць), що становить 58,1% від загального попиту, при середньому попиті 11,00 одиниць на SKU. Товари X-класу, незважаючи на більшу кількість позицій, мають дещо нижчий середній попит (9,12 одиниць на SKU) та сукупний попит 6,542 одиниці. Товари Z-класу характеризуються найнижчими показниками як за сукупним (1,023 одиниці), так і за середнім попитом (5,59 одиниць на SKU).

Розрахунок оптимальних розмірів замовлення EOQ виявив диференційований підхід: найвищі значення притаманні Y-класу (14,40 одиниць в середньому), що обумовлено балансом між вищими витратами на утримання запасів та помірним рівнем попиту. X-клас демонструє середні значення EOQ (12,93 одиниці) завдяки стабільності попиту, тоді як Z-клас має найнижчі показники (9,20 одиниць) через високу непередбачуваність та консервативну стратегію управління.

Страхові запаси та точки повторного замовлення ROP відображають ризик-орієнтований підхід: Y-клас вимагає найвищих рівнів страхового запасу (12,81 одиниць) та ROP (21,99 одиниць) для забезпечення 90% рівня сервісу. Товари X-класу, завдяки стабільності, потребують мінімальних страхових запасів (5,30 одиниць) при найвищому рівні сервісу 95%.

Топ-20 найбільш затребуваних SKU демонструють диверсифікацію між категоріями Technology, Office Supplies та Furniture, з річним попитом від 24,99 до 35,62 одиниць. Переважання Y-класу серед лідерів (13 з 20 позицій) підтверджує їх ключову роль у загальній структурі попиту.

Висновки до розділу 3

У межах третього розділу реалізовано практичну частину системи прогнозування попиту для управління запасами. Початковим етапом став аналіз і попередня обробка вхідних даних, що включала очищення, нормалізацію та формування часових, категоріальних і трендових ознак. Цей етап заклав основу для подальшого моделювання та забезпечив високу якість вхідного датасету.

Наступним кроком є навчання моделей прогнозування попиту. У роботі протестовано та порівняно різні алгоритми, серед яких ARIMA, SARIMA, Random Forest, XGBoost, LSTM і N-BEATS. Кожна з моделей показала різну ефективність залежно від характеристик даних. Зокрема, нейромережеві підходи, такі як LSTM та N-BEATS, продемонстрували найвищу точність при роботі з нерівномірними та сезонними рядами. Натомість моделі на основі ансамблевого навчання, як-от Random Forest і XGBoost, виявилися більш стійкими до шуму та краще справлялися з короткотерміновими прогнозами.

Результати прогнозування були інтегровані в процес управління запасами. Продемонстровано, як використання прогнозних значень дозволяє визначати економічно оптимальні розміри замовлення EOQ, розраховувати точку повторного замовлення ROP та величину страхового запасу. Це забезпечує зменшення ризиків дефіциту чи надлишку товарів та дозволяє ефективніше управляти складськими ресурсами.

Загалом, реалізована система підтвердила свою практичну цінність, продемонструвавши здатність адаптуватися до змін ринку, підвищити точність прийняття рішень та знизити витрати, пов’язані з логістикою та зберіганням. Практична реалізація довела ефективність застосування сучасних методів прогнозування у реальних бізнес-сценаріях.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ СИСТЕМИ ПРОГНОЗУВАННЯ ПОПИТИ ДЛЯ УПРАВЛІННЯ ЗАПАСАМИ

У цьому розділі буде проведено оцінку основних характеристик майбутнього вбудованого програмного продукту. Також досліджуються різні варіанти реалізації, що дозволяють обрати найбільш ефективну та оптимальну стратегію, враховуючи економічні аспекти та сумісність із програмним продуктом. Для цього використовується метод функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це методологія, яка дозволяє оцінити реальну вартість продукту чи послуги незалежно від структури компанії. Метою ФВА є пошук можливостей для зниження витрат за рахунок ефективніших виробничих рішень та кращого балансу між споживчою цінністю продукту і витратами на його створення. Для аналізу залучається економічна, технічна та конструкторська інформація.

Алгоритм ФВА передбачає: визначення етапів розробки продукту, розрахунок річних витрат і кількості робочого часу, ідентифікацію джерел витрат, а також остаточний 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 наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- швидке опрацювання даних, побудова прогнозів та відображення результатів;
- можливість масштабування системи для обробки великих обсягів даних без необхідності суттєвого доопрацювання системи;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 відповідає за створення програми, яка вирішує задачу сегментації внутрішніх органів людини зі зручним інтерфейсом. Таким чином можна вирізнити наступні підфункції:

- F_1 – вибір найбільш підходящої мови програмування;
- F_2 – вибір засобів розробки;
- F_3 – вибір середовища розробки.

Кожна з цих функцій має кілька можливих варіантів реалізації:

Функція F_1:

а) Python;

б) R.

Функція F_2:

а) застосування готових бібліотек для побудови моделей;

б) написання алгоритмів власноруч.

Функція F_3

а) Kaggle Notebook;

б) Jupiter Notebook;

в) PyCharm.

Морфологічна карта системи на рисунку 4.1 відображає варіанти реалізацій функцій описаних вище.

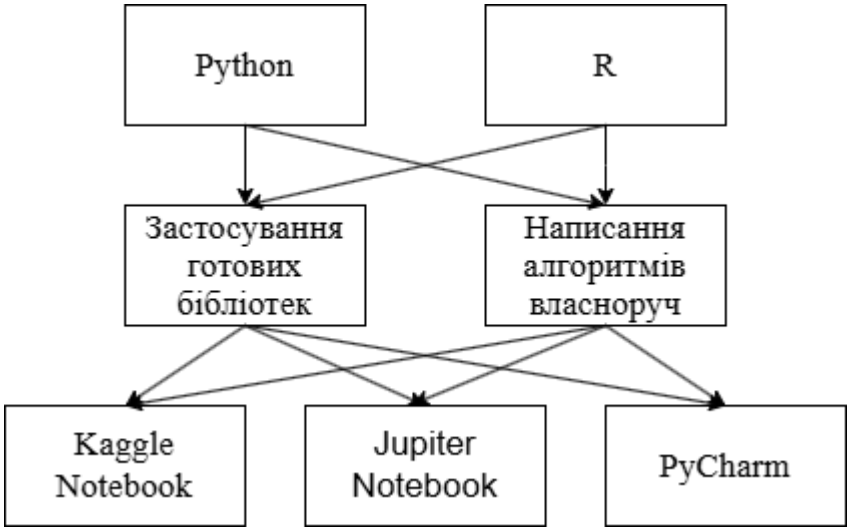

Рисунок 4.1 – Морфологічна карта

За допомогою морфологічної карти є можливість побудувати позитивно-негативну матрицю з відображенням усіх варіантів основних функцій (табл. 4.1).

Таблиця 4.1 – Позитивно-негативна матриця

Функції	*Варіанти реалізації*	*Переваги*	*Недоліки*
F_1	*А*	Велика кількість бібліотек для машинного навчання, проста для використання.	Може працювати повільніше при обробці великих обсягів даних.
	Б	Сильний інструмент для статистичного аналізу і роботи з часовими рядами.	Обмежені можливості для складних моделей машинного навчання.

Продовження таблиці 4.1

Функції	*Варіанти реалізації*	*Переваги*	*Недоліки*
F_2	*А*	Швидкий доступ до сучасних методів прогнозування, простота у використанні та перевіреність рішень. Зменшує час розробки, оскільки реалізації вже оптимізовані.	Обмежена гнучкість, залежність від сторонніх рішень.
	Б	Повний контроль над логікою моделі, можливість адаптації під конкретні бізнес-потреби. Допомагає повністю освоїти та зрозуміти принципи роботи алгоритму.	Вимагає більше часу, складності та досвіду у розробці, висока ймовірність помилок.
F_3	*А*	Потужне середовище розробки IDE, ефективний функціонал для розробки, налагодження і тестування.	Низька інтерактивність, нижча ефективність для дослідження даних і побудови візуалізацій.
	Б	Інтерактивне середовище, зручне для експериментів та візуалізації результатів, підтримує markdown та вбудовані графіки, зручний для швидкого прототипування.	Менш зручний для великих проектів, важче підтримувати складну структуру коду, не підходить для командної роботи в масштабі.

Кінець таблиці 4.1

Функції	*Варіанти реалізації*	*Переваги*	*Недоліки*
F_3	B	Повноцінне середовище розробки з підтримкою великих проектів, інтеграція систем контролю версій (Git), потужний відлагоджувач і автодоповнення коду.	Потрібна установка на комп'ютер, велика вага IDE, менше можливостей для інтерактивної роботи.

Згідно з аналізом позитивно-негативної матриці, деякі варіанти реалізації функцій програмного продукту не відповідають поставленим перед ним задачам. Ці варіанти слід виключити з розробки.

Функція F_1:

У порівнянні мова програмування Python має перевагу за рахунок своєї читабельності та зрозумілості та великої кількості доступних бібліотек як засобів реалізації.

Функція F_2:

Варіант А дозволяє швидко та ефективно реалізувати моделі без необхідності заново розробляти складні алгоритми. Бібліотеки забезпечують оптимізовану та протестовану реалізацію, що знижує ризик помилок і економить час розробки.

Функція F_3:

Варіант Б не розглядається через обмежену інтерактивність і складнощі у роботі з великими проектами. Таким чином, залишаються варіанти А та В. PyCharm забезпечує повноцінне середовище розробки з потужними інструментами для кодування, відлагодження та тестування, тоді як Kaggle Notebook ідеально підходить для ефективної розробки та експериментів. Маємо наступні варіанти реалізації ПП:

$$F_1\text{а} - F_2\text{а} - F_3\text{а},$$

$$F_1\text{а} - F_2\text{а} - F_3\text{в}.$$

Оцінювання якості згаданих функцій виконується з допомогою параметрів нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі аналізу представлених даних визначено основні характеристики, що використовуються для розрахунку коефіцієнта технічного рівня програмного продукту. Для оцінки ефективності системи обрано такі ключові параметри:

X_1 – потенційний обсяг програмного коду;

X_2 – час, необхідний для навчання моделі;

X_3 – час витрачений на виконання поставленого завдання;

X_4 – точність моделі.

Параметри поділяються на гірші, середні і кращі. Значення підібрані з урахуванням практичних вимог до продуктивності, ресурсів і якості розробки. Усі значення наведено в таблиці 4.2.

Таблиця 4.2 – Основні параметри програмного продукту

Назва Параметра	*Умовні позна-чення*	*Одиниці виміру*	*Значення параметра*		
			гірші	*середні*	*кращі*
Гіпотетичний об’єм коду програми	X1	кількість рядків коду	900	700	500
Час навчання моделей	X2	хвилини	47	30	20

Кінець таблиці 4.2

Назва Параметра	*Умовні позна-чення*	*Одиниці виміру*	*Значення параметра*		
			гірші	*середні*	*кращі*
Час витрачений на виконання поставленого завдання	X3	дні	60	35	20
Точність моделі	X4	відсотки	50	85	92

Дані в таблиці 4.2 дозволяють побудувати графічні характеристики для оцінки та порівняння параметрів на рисунках 4.2 – 4.5.

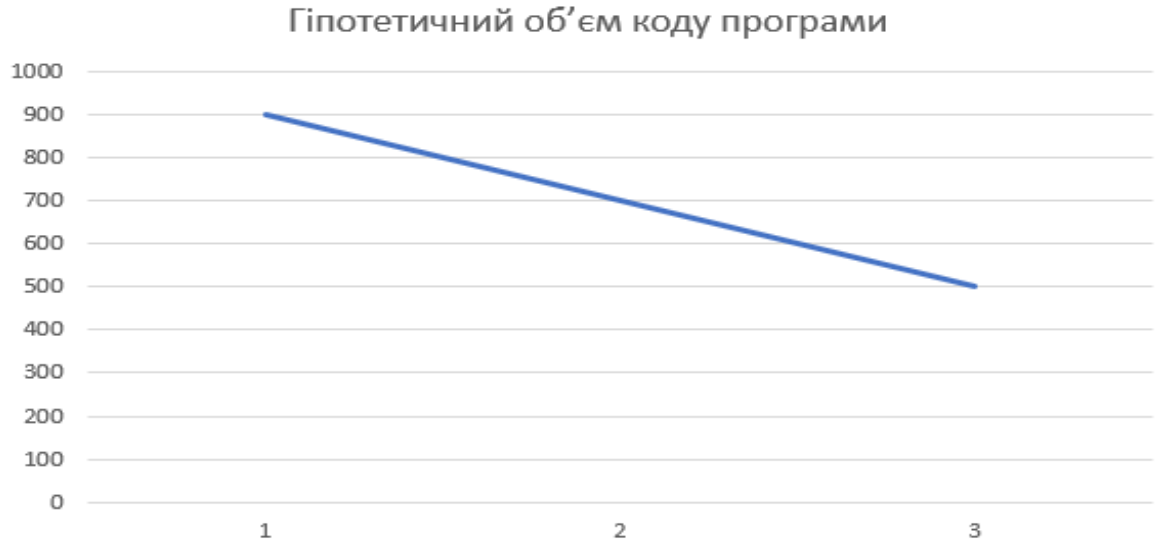

Рисунок 4.2 – X1, Гіпотетичний об’єм коду програми

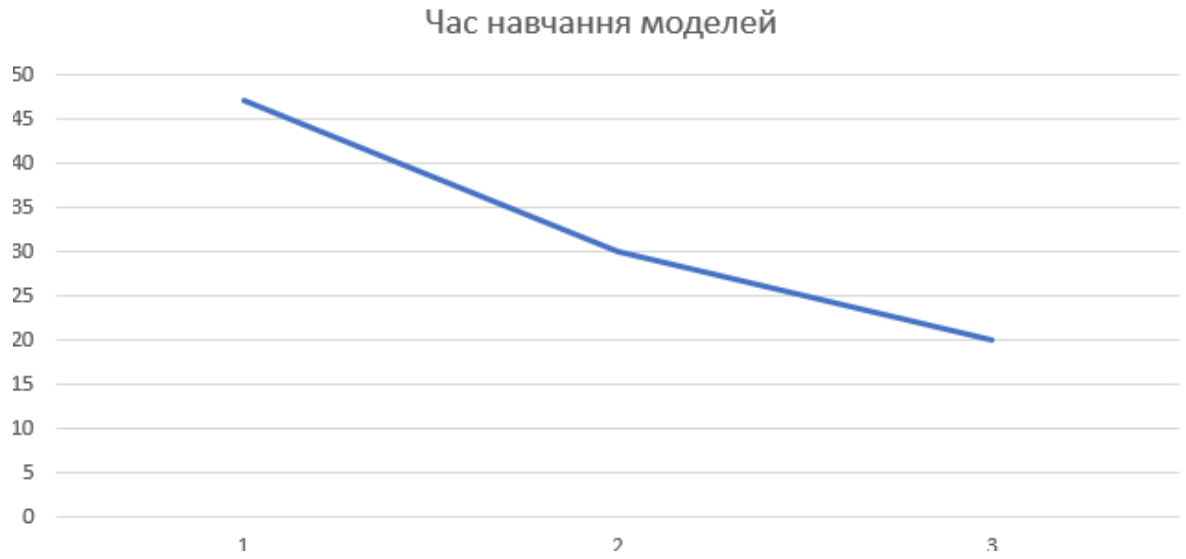

Рисунок 4.3 – X2, Час навчання моделей

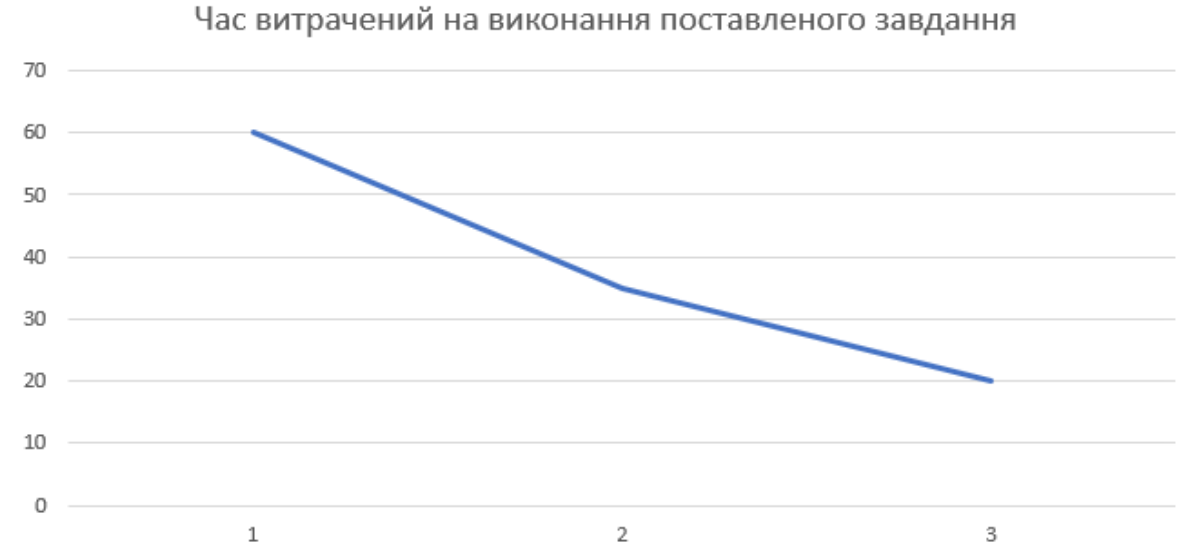

Рисунок 4.4 – X3, Час витрачений на виконання поставленого завдання

Рисунок 4.5 – X4, Точність моделі

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення та комплексного аналізу кожен експерт уважно оцінює значимість кожного параметра з урахуванням основної мети – створення програмного продукту, здатного забезпечити високу точність прогнозування параметрів моделей та розрахунку прогнозних значень.

Вагу кожного параметра визначають за допомогою методу попарного порівняння, що здійснюється групою з чотирьох експертів. Процес визначення коефіцієнтів значимості включає наступні етапи:

- присвоєння параметрам рівні значимості після виконання ранжування;
- оцінка придатності експертних оцінок для затвердження їх можливого наступного використання;
- порівняння параметрів за їх важливістю;
- проведення глибокого аналізу даних та визначення вагових коефіцієнтів параметрів.

Результати рангової оцінки експертами 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	Гіпотетичний обсяг програмного коду	кількість рядків коду	1	2	2	1	1	1	2	10	-7,5	56,25
X2	Час навчання моделей	хвилини	3	4	3	3	4	3	4	24	6,5	42,25
X3	Час витрачений на виконання поставленого завдання	дні	2	1	1	2	2	2	1	11	-6,5	42,25
X4	Точність моделі	%	4	3	4	4	3	4	3	25	7,5	56,25
	Разом		10	10	10	10	10	10	10	70	0	197

Щоб оцінити достовірність експертних рейтингів, визначаємо такі параметри:

а) сумарний ранг кожного показника та загальний ранг усіх показників:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.1)$$

де N – кількість експертів,

n – число параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5; \tag{4.2}$$

в) різниця між рангами кожного параметра та середньої суми рангів:

$$\Delta_i = R_i - T. \tag{4.3}$$

Сума, що визначає відхилення кожного параметру повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 137. \tag{4.4}$$

Коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 197}{7^2(4^3-4)} = 0{,}754 > W_k = 0{,}56. \tag{4.5}$$

Отримані результати ранжування можна вважати надійними, адже розрахований коефіцієнт узгодженості перевищує нормативний показник, значення якого дорівнює 0,56. Таким чином утворимо таблицю 4.4, де попарно порівняємо всі параметри.

Таблиця 4.4 – Попарне порівняння параметрів.

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

Кінець таблиці 4.4

Параме-три	*Експерти*							*Кінцева оцінка*	*Числове значення*
	1	2	3	4	5	6	7		
X2 і X3	>	>	<	>	>	>	>	>	1,5
X2 і X4	<	>	<	<	<	>	<	<	0,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

Ступінь переваги *i*–го параметра над *j*–м визначається числовим параметром a_{ij} і визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j. \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.6)$$

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

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij}. \quad (4.8)$$

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 наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \quad (4.9)$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j. \quad (4.10)$$

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.*	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$
X1	1	0,5	1,5	0,5	3,5	0,21875	12,25	0,207627
X2	1,5	1	1,5	0,5	4,5	0,28125	16,25	0,275424
X3	0,5	0,5	1	0,5	2,5	0,15625	9,25	0,15678
X4	1,5	1,5	1,5	1	5,5	0,34375	21,25	0,360169
Всього:					16	1	59	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X1* (Гіпотетичний об’єм коду програми), *X2* (Час навчання моделей) та *X4* (Точність моделі) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3* (Час витрачений на виконання поставленого завдання) обрано не найгіршим.

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

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (4.11)$$

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	A	X1	700	17	0.21	3.57
F2	A	X4	91	33	0.36	11.88
F3	A	X3	50	25	0.15	3.75
	B	X3	57	20	0.15	3

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 3.57 + 11.88 + 3.75 = 19.2 ;$$

$$K_{K2} = 3.57 + 11.88 + 3 = 18.45 .$$

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

4.6 Економічний аналіз варіантів розробки ПП

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 Б, завдання 2 до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 2.

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:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

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

$K_М$ – коефіцієнт рівня мови програмування;

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

$K_{СТ.М}$ – 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_P = 28 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.3. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ}$ = 0.5. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 64 \cdot 1.3 \cdot 1 \cdot 0.5 = 41.6 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =25 людино-днів, $К_П$ =1.08, $К_{СК}$ = 1, $К_{СТ}$ =0.7:

$$T_2 = 25 \cdot \ 1.08 \cdot 1 \cdot 0.7 = 18.9 \text{ людино-днів.}$$

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

$$T_I = (41.6 \ + 18.9 + 19.2 + 18.9) \cdot 8 = 788.8 \text{ людино-годин,}$$
$$T_{II} = (41.6 \ + 18.9 + 18.45 + 18.9) \cdot 8 = 782.8 \text{ людино-годин}$$

В розробці беруть участь один програміст з окладом 27200 грн., аналітик даних з окладом 25000. Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{грн,} \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів на місяць;

t – кількість робочих годин в день.

$$С_Ч = \frac{27200+25000}{2 \cdot 18 \cdot 8} = 181.85 \ \text{грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_ч \cdot T_i \cdot К_Д, \quad (4.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

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

Зарплата розробників за варіантами становить:

$$I.\ С_{ЗП} = 181.85 \cdot 788.8 \cdot 1.125 = 161373.69 \text{ грн.}$$

$$II.\ С_{ЗП} = 181.85 \cdot 782.8 \cdot 1.125 = 160146.2025 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$I.\ С_{ВІД} = С_{ЗП} \cdot 0.22 = 161373.69 \cdot 0.22 = 35{,}502.2118 \text{ грн}$$

$$II.\ С_{ВІД} = С_{ЗП} \cdot 0.22 = 160146.2025 \cdot 0.22 = 35{,}232.16455 \text{ грн.}$$

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

Так як одна ЕОМ обслуговує одного програміста з окладом 27200 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 27200 \cdot 0{,}2 = 65280 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 65280 \cdot (1 + 0.2) = 78336 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 78336 \cdot 0.22 = 17233.92 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 37000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.23 \cdot 0.25 \cdot 37000 = 11377.5 \text{ грн},$$

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:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.23 \cdot 37000 \cdot 0.03 = 1365.3 \text{ грн},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 105 – 21 – 16) \cdot 8 \cdot 0.8 = 1408 \text{ години},$$

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:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1408 \cdot 0.3 \cdot 0.5 \cdot 9.43 = 1991.616 \text{ грн.},$$

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:

$$С_Н = Ц_{ПР} \cdot 0.67 = 37000 \cdot 0,67 = 24790 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \quad (4.17)$$

$С_{ЕКС}$ =78336 + 17233.92 + 11377.5 + 1365.3 + 1991.616 + 24790 = 135094.336 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 135094.336 / 1408 = 95.95 \text{ грн/год.}$$

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:

$$С_М = С_{М-Г} \cdot Т, \quad (4.18)$$

I. $С_М = 95.95 \cdot 788.8 = 75685.36$ грн.

II. $С_М = 95.95 \cdot 782.8 = 75109.66$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_М = С_{М-Г} \cdot Т, \quad (4.19)$$

I. $С_Н = 161373.69 \cdot 0.67 = 108120.3723$ грн.

II. $С_Н = 160146.2025 \cdot 0.67 = 107297.955675$ грн

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_{М} + C_{Н}, \quad (4.20)$$

I. $C_{ПП}$ = 161373.69 + 35502.2118 + 75685.36 + 108120.3723 = 380681.6341 грн.
II. $C_{ПП}$ = 160146.2025 + 35,232.16455 + 75109.66 + 107297.955675 = 377785.982725 грн

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{Кj} / C_{Фj}, \quad (4.21)$$

$К_{ТЕР1}$ = 19.2 / 380681.6341 = 5.04 ·10^{-5}.
$К_{ТЕР2}$ = 18.45 / 377785.982725 = 4.88 ·10^{-5}.

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}$= 5.04 ·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. Цей варіант включає використання мови Python, розробка системи з допомогою бібліотек та використання Kaggle Notebook, як середовище розробки.

Висновки до розділу 4

У цьому розділі дипломної роботи проведено докладний аналіз функціональних та вартісних аспектів програмного продукту. Крім цього оцінено основні функції цього програмного продукту.

В результаті аналізу функціональності та вартості програмного комплексу, який розробляється, було визначено та проаналізовано його ключові функції, а також ідентифіковано параметри, що характеризують його. На основі проведеного аналізу був обраний варіант реалізації програмного продукту.

ВИСНОВКИ

У рамках дипломній роботі розроблено інтелектуальну систему прогнозування попиту для оптимізації управління товарними запасами, що базується на сучасних методах машинного навчання. Проведене дослідження охоплює повний цикл вирішення прикладної задачі — від аналізу предметної області до практичної реалізації моделі та застосування її результатів у контексті прийняття управлінських рішень.

На першому етапі здійснено глибокий аналіз факторів, що впливають на формування попиту, включно з часовими, економічними та маркетинговими аспектами. Доведено, що точне прогнозування попиту є ключовим чинником ефективного управління запасами, особливо в умовах нестабільного середовища та обмежених ресурсів.

У роботі проведено огляд сучасних методів прогнозування, серед яких ARIMA, SARIMA, Random Forest, XGBoost, LSTM і N-BEATS. Для кожної з моделей проаналізовано переваги, недоліки та сферу застосування. Особливу увагу приділено особливостям часових рядів і формуванню ознак, що враховують сезонність, тренди та лагові залежності.

У процесі реалізації системи зібрані й оброблені історичні дані продажів. Проведено експерименти з тренування моделей і порівняння точності прогнозування за ключовими метриками. Найкращі результати показали моделі LSTM та N-BEATS, що забезпечили високу точність передбачення майбутнього попиту.

На завершальному етапі розроблену систему інтегровано з модулем прийняття рішень щодо управління запасами, зокрема розраховано оптимальні обсяги замовлень EOQ та точки повторного замовлення ROP. Система дозволяє зменшити витрати на зберігання, знизити ризики дефіциту та забезпечити стабільне обслуговування клієнтів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Factors That Affect the Retail Industry Nowadays – Retail Minded. URL: https://retailminded.com/factors-that-affect-the-retail-industry-nowadays/ (дата звернення: 01.05.2025).
2. Широкопетлєва М. С., Пономаренко О. А., Дудар З. В. Порівняння методів прогнозування часових рядів. Біоніка інтелекту. 2018. №4.
3. Марчук Д. К., Кравченко С. М., Левченко А. Ю., Лежньов І. Я. Використання часових рядів при прогнозуванні грошової вартості автомобілів. Вчені записки ТНУ імені В. І. Вернадського. Серія: Технічні науки. 2023. Т. 34, №1.
4. Salman H. A., Kalakech A., Steiti A. Random Forest Algorithm Overview. Babylonian Journal of Machine Learning. 2024.
5. Hasan M. M., Mahmud M. A hybrid ensemble method for pulsar candidate classification. URL: https://www.researchgate.net/publication/335483097_A_hybrid_ensemble_method_for_pulsar_candidate_classification (дата звернення: 03.05.2025).
6. Kumar A., Singh R. Exploring the Power of eXtreme Gradient Boosting Algorithm in Machine Learning: A Review. URL: https://www.researchgate.net/publication/371202498_Exploring_the_Power_of_eXtreme_Gradient_Boosting_Algorithm_in_Machine_Learning_a_Review (дата звернення: 02.05.2025).
7. Kumar S., Sharma R. Deep Learning Long-Short Term Memory. URL: https://www.researchgate.net/publication/352383391_Deep_Learning_Long-Short_Term_Memory (дата звернення: 08.05.2025).
8. Olah C. Understanding LSTM Networks. URL: https://web.stanford.edu/class/cs379c/archive/2018/class_messages_listing/content/Artificial_Neural_Network_Technology_Tutorials/OlahLST

M-NEURAL-NETWORK-TUTORIAL-15.pdf (дата звернення: 05.05.2025).

9. GeeksforGeeks. SARIMA (Seasonal Autoregressive Integrated Moving Average). URL: https://www.geeksforgeeks.org/sarima-seasonal-autoregressive-integrated-moving-average/ (дата звернення: 03.05.2025).
10. GeeksforGeeks. Understanding of LSTM Networks. URL: https://www.geeksforgeeks.org/understanding-of-lstm-networks/ (дата звернення: 03.05.2025).
11. Investopedia. Economic Order Quantity (EOQ). URL: https://www.investopedia.com/terms/e/economicorderquantity.asp (дата звернення: 09.05.2025).
12. ADW Service. Сезонність попиту. URL: https://adwservice.com.ua/uk/sezonnist-popytu (дата звернення: 08.05.2025).
13. Medium. Introduction to Feature Engineering for Time Series Forecasting. URL: https://medium.com/data-science-at-microsoft/introduction-to-feature-engineering-for-time-series-forecasting-620aa55fcab0 (дата звернення: 07.05.2025).
14. Sertis. A Practical Time Series Forecasting Guideline for Machine Learning – Part II. URL: https://sertiscorp.medium.com/a-practical-time-series-forecasting-guideline-for-machine-learning-part-ii-aea360b06ce2 (дата звернення: 30.04.2025).
15. Ahmed A. E. M. Inventory Management. У: Operations Management – Recent Advances and New Perspectives. IntechOpen, 2024.
16. Луценко І. С. Логістичне управління запасами: навчально-методичний комплекс дисципліни. Київ: КПІ ім. Ігоря Сікорського, 2021.
17. Demand Forecasting for Improved Inventory Management in Small and Medium-Sized Businesses. URL:

https://www.researchgate.net/publication/372086184_Demand_Forecasting_for_Improved_Inventory_Management_in_Small_and_Medium-Sized_Businesses (дата звернення: 07.05.2025).

18. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. URL: https://www.researchgate.net/publication/333418084_N-BEATS_Neural_basis_expansion_analysis_for_interpretable_time_series_forecasting (дата звернення: 04.05.2025).
19. What Is a Reorder Point (ROP)? URL: https://www.netsuite.com/portal/resource/articles/inventory-management/reorder-point-rop.shtml (дата звернення: 08.05.2025).
20. Safety Stock. URL: https://www.netsuite.com/portal/resource/articles/inventory-management/safety-stock.shtml (дата звернення: 08.05.2025).
21. Lokad. Deep Learning. URL: https://www.lokad.com/deep-learning/?utm_source (дата звернення: 29.04.2025).
22. Oracle. AI & Analytics for Retail. URL: https://www.oracle.com/cis/retail/ai-analytics/?utm_source (дата звернення: 29.04.2025).
23. Relex Solutions. Demand Planning Software. URL: https://www.relexsolutions.com/solutions/demand-planning-software/?utm_source (дата звернення: 29.04.2025).

ДОДАТОК А ОСНОВНІ СКЛАДНИКИ ЛІСТИНГУ ПРОГРАМИ

Повний лістинг коду розміщено за посиланням:

https://github.com/yevtush/diplom.git

```
import pandas as pd
import numpy as np
import math
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
import tensorflow as tf
from tensorflow.keras.layers import Dense, Concatenate, Input, Subtract, Add, LSTM, Dropout
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
import xgboost as xgb
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller

# Завантаження та підготовка даних
df = pd.read_csv('Superstore.csv', encoding="latin1")
df = df.drop_duplicates()

df['Order Date'] = pd.to_datetime(df['Order Date'], dayfirst=True)

# Агрегація даних по місяцях
monthly_quantity = df.groupby(pd.Grouper(key='Order Date',
freq='M'))['Quantity'].sum().reset_index()
monthly_quantity.set_index('Order Date', inplace=True)
monthly_quantity = monthly_quantity[monthly_quantity['Quantity'] > 0]

# Пошук оптимальних параметрів ARIMA
p_values = range(0, 4)
d_values = range(0, 3)
q_values = range(0, 4)

results = []
```

```
for p in p_values:
    for d in d_values:
        for q in q_values:
            aic, model_fit = evaluate_arima_model(train['Quantity'], (p, d, q))
            if model_fit is not None:
                results.append((p, d, q, aic))

results.sort(key=lambda x: x[3])
best_arima_params = results[0][:3]
print(f"Найкращі параметри ARIMA: {best_arima_params}")

# Побудова ARIMA моделі
arima_model = ARIMA(train['Quantity'], order=best_arima_params)
arima_fitted = arima_model.fit()

# Прогнозування ARIMA
arima_forecast = arima_fitted.forecast(steps=len(test))

# Сезонна декомпозиція
decomposition = seasonal_decompose(monthly_quantity['Quantity'], model='multiplicative',
period=12)
seasonal_strength = np.std(decomposition.seasonal) / np.std(monthly_quantity['Quantity'])

# Функція для оцінки SARIMA моделі
def evaluate_sarima_model(data, order, seasonal_order):
    train_data = data[:-12]
    test_data = data[-12:]
    try:
        model = SARIMAX(train_data, order=order, seasonal_order=seasonal_order,
                        enforce_stationarity=False, enforce_invertibility=False)
        model_fit = model.fit(disp=False)
        forecast = model_fit.get_forecast(steps=len(test_data))
        mse = mean_squared_error(test_data, forecast.predicted_mean)
        return mse
    except:
        return float("inf")

# Пошук оптимальних параметрів SARIMA
candidate_params = [
    ((0,1,1), (0,1,0,12)),
    ((1,2,4), (0,1,0,12)),
    ((1,0,0), (1,1,1,12)),
    ((0,0,1), (1,1,1,12)),
    ((2,0,1), (1,1,1,12))
]
```

```
best_score = float("inf")
best_sarima_params = None

for order, seasonal_order in candidate_params:
    mse = evaluate_sarima_model(monthly_quantity['Quantity'], order, seasonal_order)
    if mse < best_score:
        best_score = mse
        best_sarima_params = (order, seasonal_order)


# Побудова SARIMA моделі
order, seasonal_order = best_sarima_params
sarima_model = SARIMAX(train, order=order, seasonal_order=seasonal_order,
                       enforce_stationarity=False, enforce_invertibility=False)
sarima_fitted = sarima_model.fit(disp=False)

def prepare_time_series_data(df):
    """Підготовка часових рядів"""
    df['Order Date'] = pd.to_datetime(df['Order Date'], dayfirst=True)
    df['Year_Month'] = df['Order Date'].dt.to_period('M')

    monthly_data = df.groupby(['Year_Month']).agg({
        'Quantity': 'sum'
    }).reset_index()

    monthly_data['Year_Month'] = monthly_data['Year_Month'].dt.to_timestamp()
    monthly_data = monthly_data.sort_values('Year_Month')

    return monthly_data

def create_features(df):
    """Створення ознак для машинного навчання"""
    df = df.copy()

    # Часові ознаки
    df['month'] = df['Year_Month'].dt.month
    df['quarter'] = df['Year_Month'].dt.quarter
    df['year'] = df['Year_Month'].dt.year

    # Циклічні ознаки
    df['month_sin'] = np.sin(2 * np.pi * df['month']/12)
    df['month_cos'] = np.cos(2 * np.pi * df['month']/12)
    df['quarter_sin'] = np.sin(2 * np.pi * df['quarter']/4)
    df['quarter_cos'] = np.cos(2 * np.pi * df['quarter']/4)
```

```
    # Індикатори періодів
    df['is_year_end'] = (df['month'] == 12).astype(int)
    df['is_quarter_end'] = (df['month'].isin([3, 6, 9, 12])).astype(int)

    # Лагові ознаки
    for lag in range(1, 7):
        df[f'quantity_lag_{lag}'] = df['Quantity'].shift(lag)

    # Ковзні вікна
    for window in [3, 6, 12]:
        df[f'quantity_ma_{window}'] = df['Quantity'].rolling(window=window).mean()
        df[f'quantity_std_{window}'] = df['Quantity'].rolling(window=window).std()

    # Різницеві ознаки
    for lag in [1, 3, 6]:
        df[f'quantity_diff_{lag}'] = df['Quantity'].diff(lag)
        df[f'quantity_pct_change_{lag}'] = df['Quantity'].pct_change(lag)

    # Річна зміна
    df['quantity_yoy'] = df['Quantity'] / df['Quantity'].shift(12) - 1

    # Заповнення пропущених значень
    df = df.fillna(method='bfill').fillna(method='ffill')

    return df

def train_test_split_time_series(df, test_months=12):
    """Розділення на тренувальну та тестову вибірки"""
    split_date = df['Year_Month'].max() - pd.DateOffset(months=test_months)
    train = df[df['Year_Month'] <= split_date]
    test = df[df['Year_Month'] > split_date]
    return train, test

def prepare_features_target(train, test):
    """Підготовка ознак та цільової змінної"""
    target = 'Quantity'
    drop_features = ['Year_Month', target]

    X_train = train.drop(drop_features, axis=1)
    y_train = train[target]
    X_test = test.drop(drop_features, axis=1)
    y_test = test[target]

    return X_train, y_train, X_test, y_test
```

```
def train_evaluate_model(X_train, y_train, X_test, y_test):
    """Навчання та оцінка моделі Random Forest"""
    model = RandomForestRegressor(
        n_estimators=100,
        max_depth=10,
        random_state=42,
        n_jobs=-1
    )

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Розрахунок метрик
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, y_pred)
    mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

    return model, y_pred

# XGBoost модель для прогнозування

def train_evaluate_xgboost(X_train, y_train, X_test, y_test):
    """Навчання та оцінка XGBoost моделі"""
    # Ініціалізація XGBoost регресора з параметрами для часових рядів
    model = xgb.XGBRegressor(
        n_estimators=100,
        learning_rate=0.1,
        max_depth=4,
        min_child_weight=2,
        subsample=0.8,
        colsample_bytree=0.8,
        gamma=0,
        objective='reg:squarederror',
        random_state=42
    )

    # Навчання моделі
    print("Навчання XGBoost моделі...")
    model.fit(
        X_train, y_train,
        eval_set=[(X_train, y_train), (X_test, y_test)],
        eval_metric=['rmse', 'mae'],
        early_stopping_rounds=20,
```

```
        verbose=False
    )

    # Прогнозування
    y_pred = model.predict(X_test)

    # Оцінка моделі
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # MAPE (якщо можливо)
    if np.any(y_test <= 0):
        mape = None
    else:
        mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

    # SMAPE
    smape = 100 * np.mean(2 * np.abs(y_pred - y_test) / (np.abs(y_pred) + np.abs(y_test)))

    return model, y_pred


def tune_xgboost_hyperparameters(X_train, y_train):
    """Налаштування гіперпараметрів XGBoost за допомогою GridSearchCV"""
    # Визначення сітки параметрів
    param_grid = {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.05, 0.1],
        'max_depth': [3, 4, 5, 6],
        'min_child_weight': [1, 2, 3],
        'subsample': [0.7, 0.8, 0.9],
        'colsample_bytree': [0.7, 0.8, 0.9],
        'gamma': [0, 0.1, 0.2]
    }

    # Ініціалізація XGBoost регресора
    xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

    # Виконання пошуку з перехресною валідацією
    print("Налаштування гіперпараметрів...")
    grid_search = GridSearchCV(
        estimator=xgb_model,
        param_grid=param_grid,
```

```
        cv=5,
        scoring='neg_mean_squared_error',
        n_jobs=-1,
        verbose=0
    )

    grid_search.fit(X_train, y_train)

    best_params = grid_search.best_params_
    best_score = np.sqrt(-grid_search.best_score_)

    print(f"Найкращі параметри: {best_params}")
    print(f"Найкращий RMSE: {best_score:.2f}")

    return best_params
# ---- Підготовка багатовимірних даних для LSTM ----

def prepare_multivariate_lstm_data(df, feature_columns=None):
    """Підготовка багатовимірних часових рядів для LSTM"""
    monthly_data = prepare_time_series_data(df)
    feature_df = create_features(monthly_data)

    if feature_columns is None:
        feature_columns = [col for col in feature_df.columns if col != 'Year_Month']

    return feature_df, feature_columns

def create_multivariate_sequences(data, target_col, feature_cols, n_steps_in, n_steps_out=1):
    """Створення багатовимірних послідовностей для LSTM моделі"""
    X, y = [], []

    target_data = data[target_col].values
    feature_data = data[feature_cols].values

    for i in range(len(data) - n_steps_in - n_steps_out + 1):
        seq_x = feature_data[i:(i + n_steps_in)]
        seq_y = target_data[(i + n_steps_in):(i + n_steps_in + n_steps_out)]

        X.append(seq_x)
        y.append(seq_y)

    return np.array(X), np.array(y)

def scale_multivariate_data(train_df, test_df, feature_cols, target_col='Quantity'):
    """Масштабування багатовимірних даних за допомогою Min-Max"""
```

```
    feature_scaler = MinMaxScaler(feature_range=(0, 1))
    target_scaler = MinMaxScaler(feature_range=(0, 1))

    feature_scaler.fit(train_df[feature_cols])
    target_scaler.fit(train_df[[target_col]])

    train_features_scaled = feature_scaler.transform(train_df[feature_cols])
    test_features_scaled = feature_scaler.transform(test_df[feature_cols])

    train_target_scaled = target_scaler.transform(train_df[[target_col]]).flatten()
    test_target_scaled = target_scaler.transform(test_df[[target_col]]).flatten()

    return (train_features_scaled, train_target_scaled,
            test_features_scaled, test_target_scaled,
            feature_scaler, target_scaler)

# ---- Побудова багатовимірної LSTM моделі ----

def build_multivariate_lstm_model(n_steps_in, n_features):
    """Побудова багатовимірної LSTM моделі"""
    model = Sequential()

    # Перший LSTM шар з поверненням послідовностей
    model.add(LSTM(units=64,
                   return_sequences=True,
                   input_shape=(n_steps_in, n_features)))
    model.add(Dropout(0.2))

    # Другий LSTM шар
    model.add(LSTM(units=32))
    model.add(Dropout(0.2))

    # Вихідний шар
    model.add(Dense(units=1))

    # Компіляція моделі
    model.compile(optimizer='adam', loss='mse')

    return model
#N-BEATS модель для прогнозування

# Підготовка часових рядів для прогнозування
def prepare_time_series_data(df):
    """Агрегація даних до місячного рівня"""
    df['Order Date'] = pd.to_datetime(df['Order Date'], dayfirst=True)
```

```
    df['Year_Month'] = df['Order Date'].dt.to_period('M')

    monthly_data = df.groupby(['Year_Month']).agg({
        'Quantity': 'sum',
    }).reset_index()

    monthly_data['Year_Month'] = monthly_data['Year_Month'].dt.to_timestamp()
    return monthly_data.sort_values('Year_Month')

# Створення послідовностей для прогнозування
def create_sequences(data, lookback, forecast_horizon):
    """Формування вхідних та цільових послідовностей"""
    X, y = [], []
    for i in range(len(data) - lookback - forecast_horizon + 1):
        X.append(data[i:(i + lookback)])
        y.append(data[(i + lookback):(i + lookback + forecast_horizon)])

    if forecast_horizon == 1:
        return np.array(X), np.array(y).reshape(-1, forecast_horizon)
    return np.array(X), np.array(y)

# Обчислення MAPE
def mean_absolute_percentage_error(y_true, y_pred):
    """Розрахунок середньої абсолютної відсоткової похибки"""
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    mask = y_true != 0
    return np.mean(np.abs((y_true[mask] - y_pred[mask]) / y_true[mask])) * 100

# Створення блоку N-BEATS
def create_nbeats_block(x, units, layers, sharing, layer_type, name):
    """Створення базового блоку N-BEATS"""
    # Приховані шари блоку
    for i in range(layers):
        x = Dense(units, activation='relu', name=f'{name}_FC_{i}')(x)

    # Вихідні шари блоку
    if sharing == 'shared':
        theta = Dense(units, activation='linear', name=f'{name}_theta')(x)
        backcast = Dense(units, activation='linear', name=f'{name}_backcast')(theta)
        forecast = Dense(units, activation='linear', name=f'{name}_forecast')(theta)
    else:
        theta_b = Dense(units, activation='linear', name=f'{name}_theta_b')(x)
        theta_f = Dense(units, activation='linear', name=f'{name}_theta_f')(x)
        backcast = Dense(units, activation='linear', name=f'{name}_backcast')(theta_b)
        forecast = Dense(units, activation='linear', name=f'{name}_forecast')(theta_f)
```

```
    # Спеціалізовані вихідні шари залежно від типу блоку
    if layer_type == 'trend':
        backcast = Dense(1, activation='linear', name=f'{name}_backcast_trend')(backcast)
        forecast = Dense(1, activation='linear', name=f'{name}_forecast_trend')(forecast)
    elif layer_type == 'seasonality':
        backcast = Dense(1, activation='linear', name=f'{name}_backcast_seasonal')(backcast)
        forecast = Dense(1, activation='linear', name=f'{name}_forecast_seasonal')(forecast)
    else:  # 'generic'
        backcast = Dense(1, activation='linear', name=f'{name}_backcast_generic')(backcast)
        forecast = Dense(1, activation='linear', name=f'{name}_forecast_generic')(forecast)

    return backcast, forecast

# Побудова моделі N-BEATS
def build_nbeats_model(lookback, forecast_horizon, stacks=2, blocks=3, units=128,
                       layers=4, sharing='shared', stack_types=None):
    """Побудова спрощеної моделі N-BEATS"""
    if stack_types is None:
        stack_types = ['trend', 'seasonality']

    # Вхідний шар
    input_layer = Input(shape=(lookback,), name='input')
    residuals = input_layer
    forecast_outputs = []

    # Створення стеків
    for stack_id in range(stacks):
        stack_type = stack_types[stack_id]
        stack_input = residuals

        # Створення блоків для стеку
        for block_id in range(blocks):
            backcast, forecast = create_nbeats_block(
                stack_input, units, layers, sharing, stack_type,
                f'Stack{stack_id}_Block{block_id}'
            )

            # Оновлення залишків
            residuals = Subtract(name=f'Subtract_S{stack_id}_B{block_id}')([stack_input,
backcast])
            forecast_outputs.append(forecast)
            stack_input = residuals

    # Підсумовування всіх прогнозів
```

```
    if len(forecast_outputs) > 1:
        final_forecast = Add(name='Sum_Forecast')(forecast_outputs)
    else:
        final_forecast = forecast_outputs[0]

    final_forecast = Dense(forecast_horizon, activation='linear',
                           name='Forecast_Output')(final_forecast)

    model = Model(inputs=input_layer, outputs=final_forecast)
    model.compile(optimizer=Adam(), loss='mse')

    return model

data = {
    'Date': [
        '2015-01-01', '2015-02-01', '2015-03-01', '2015-04-01',
        '2015-05-01', '2015-06-01', '2015-07-01', '2015-08-01',
        '2015-09-01', '2015-10-01', '2015-11-01', '2015-12-01'
    ],
    'Predicted_Quantity': [
        615.464966, 930.078796, 1367.803345, 1517.312378,
        1674.555176, 1665.634766, 1633.679932, 1712.018066,
        1753.078613, 1740.651367, 1768.306763, 1767.003540
    ]
}

# Підрахунок історичних часток продажів по категоріях
category_shares = df.groupby('Category')['Quantity'].sum()
category_shares = category_shares / category_shares.sum()

# Створимо новий датафрейм із категоріями, який будемо заповнювати
category_forecast = []

for _, row in sales_pred.iterrows():
    total = row['Predicted_Quantity']
    date = row['Date']
    for cat, share in category_shares.items():
        category_forecast.append({
            'Date': date,
            'Category': cat,
            'Forecast_Quantity': total * share
        })

category_forecast_df = pd.DataFrame(category_forecast)
```

```
# Групування і обчислення частки підкатегорії в межах своєї категорії
subcategory_shares = (
    df.groupby(['Category', 'Sub-Category'])['Quantity']
    .sum()
    .groupby(level=0)
    .apply(lambda x: x / x.sum())
)

subcategory_forecast = []

for _, row in category_forecast_df.iterrows():
    date = row['Date']
    category = row['Category']
    cat_quantity = row['Forecast_Quantity']

    subcats = subcategory_shares.loc[category]

    for subcat, share in subcats.items():
        subcategory_forecast.append({
            'Date': date,
            'Category': category,
            'Sub-Category': subcat,
            'Forecast_Quantity': cat_quantity * share
        })

subcategory_forecast_df = pd.DataFrame(subcategory_forecast)

# Сума продажів за місяцями для кожного SKU
sku_monthly = df.groupby(['Sub-Category', 'Product ID', 'Month'])['Quantity'].sum().reset_index()

def classify_xyz(cv):
    if cv <= 0.5:
        return 'X'
    elif cv <= 1.0:
        return 'Y'
    else:
        return 'Z'

sku_cv['XYZ_Class'] = sku_cv['cv'].apply(classify_xyz)

xyz_result = sku_cv[['Sub-Category', 'Product ID', 'mean', 'std', 'cv', 'XYZ_Class']]

# Обчислення історичних часток SKU в межах кожної підкатегорії
sku_totals = df.groupby(['Sub-Category', 'Product ID'])['Quantity'].sum().reset_index()
```

```
# Розрахунок загальної кількості по підкатегорії
subcategory_totals = sku_totals.groupby('Sub-Category')['Quantity'].transform('sum')

# Розрахунок частки кожного SKU
sku_totals['Historical_Share'] = sku_totals['Quantity'] / subcategory_totals

# Залишаємо тільки потрібні стовпці
sku_shares = sku_totals[['Sub-Category', 'Product ID', 'Historical_Share']]

# Об'єднання з XYZ аналізом
sku_shares_xyz = sku_shares.merge(
    xyz_result[['Sub-Category', 'Product ID', 'XYZ_Class']],
    on=['Sub-Category', 'Product ID'],
    how='left'
)

print("SKU частки з XYZ класифікацією:")
print(sku_shares_xyz.head())

# Функція для коригування прогнозу на основі XYZ класу
def adjust_forecast_by_xyz(forecast_qty, xyz_class, base_share):
    if xyz_class == 'X':
        # Для стабільних товарів використовуємо історичну частку з мінімальними змінами
        adjustment_factor = 1.0
    elif xyz_class == 'Y':
        # Для помірно варіативних товарів трохи зменшуємо частку
        adjustment_factor = 0.95
    else:  # Z класс
        # Для високоваріативних товарів значно зменшуємо частку (консервативний підхід)
        adjustment_factor = 0.85

    return forecast_qty * base_share * adjustment_factor

# Створення фінального прогнозу по SKU
sku_forecast = []

for _, subcat_row in subcategory_forecast_df.iterrows():
    date = subcat_row['Date']
    category = subcat_row['Category']
    subcategory = subcat_row['Sub-Category']
    subcat_quantity = subcat_row['Forecast_Quantity']

    # Отримуємо всі SKU для цієї підкатегорії
    subcat_skus = sku_shares_xyz[sku_shares_xyz['Sub-Category'] == subcategory].copy()
```

```
    if len(subcat_skus) > 0:
        # Розраховуємо скориговані частки
        subcat_skus['Adjusted_Share'] = subcat_skus.apply(
            lambda row: adjust_forecast_by_xyz(1.0, row['XYZ_Class'], row['Historical_Share']),
            axis=1
        )

        # Нормалізуємо частки, щоб їх сума дорівнювала 1
        total_adjusted_share = subcat_skus['Adjusted_Share'].sum()
        subcat_skus['Normalized_Share'] = subcat_skus['Adjusted_Share'] / total_adjusted_share

        # Розподіляємо прогноз
        for _, sku_row in subcat_skus.iterrows():
            sku_forecast.append({
                'Date': date,
                'Category': category,
                'Sub-Category': subcategory,
                'Product_ID': sku_row['Product ID'],
                'XYZ_Class': sku_row['XYZ_Class'],
                'Historical_Share': sku_row['Historical_Share'],
                'Adjusted_Share': sku_row['Normalized_Share'],
                'Forecast_Quantity': subcat_quantity * sku_row['Normalized_Share']
            })

sku_forecast_df = pd.DataFrame(sku_forecast)

def calculate_inventory_metrics(row):
    # Дані з рядка
    annual_demand = row['Annual_Demand']
    daily_demand_mean = row['Daily_Demand_Mean']
    daily_demand_std = row['Daily_Demand_Std']
    lead_time = row['Lead_Time_Days']
    ordering_cost = row['Ordering_Cost']
    holding_cost_rate = row['Holding_Cost_Rate']
    unit_cost = row['Unit_Cost']
    z_score = row['Z_Score']
    safety_days = row['Safety_Stock_Days']

    # 1. EOQ (Economic Order Quantity)
    holding_cost_per_unit = unit_cost * holding_cost_rate
    if holding_cost_per_unit > 0:
        eoq = (2 * annual_demand * ordering_cost / holding_cost_per_unit) ** 0.5
    else:
        eoq = 0
```

```
# 2. Страховий запас (Safety Stock)
# Метод 1: На основі рівня сервісу та варіації попиту
if daily_demand_std > 0:
    safety_stock_service = z_score * daily_demand_std * (lead_time ** 0.5)
else:
    safety_stock_service = 0

# Метод 2: На основі днів покриття
safety_stock_days = daily_demand_mean * safety_days

# Використовуємо максимум з двох методів для більшої надійності
safety_stock = max(safety_stock_service, safety_stock_days)

# 3. ROP (Reorder Point)
lead_time_demand = daily_demand_mean * lead_time
rop = lead_time_demand + safety_stock

# 4. Максимальний запас
max_stock = rop + eoq

# 5. Кількість замовлень на рік
orders_per_year = annual_demand / eoq if eoq > 0 else 0

# 6. Цикл замовлення (в днях)
order_cycle_days = 365 / orders_per_year if orders_per_year > 0 else 0

return pd.Series({
    'EOQ': round(eoq, 0),
    'Safety_Stock': round(safety_stock, 0),
    'ROP': round(rop, 0),
    'Max_Stock': round(max_stock, 0),
    'Lead_Time_Demand': round(lead_time_demand, 0),
    'Orders_Per_Year': round(orders_per_year, 1),
    'Order_Cycle_Days': round(order_cycle_days, 1),
    'Annual_Holding_Cost': round(safety_stock * unit_cost * holding_cost_rate, 2),
    'Annual_Ordering_Cost': round(orders_per_year * ordering_cost, 2)
})
```