Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Рекомендовано до захисту:
протокол засідання кафедри
№ ___ від _____ 2025 р.
Завідувач кафедри ІПСіТ
_____ Олег Олешко
(підпис) (ім'я, прізвище)

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему «Розробка моделі та програмного забезпечення для прогнозування ризиків втрати клієнтів банківської установи за допомогою штучного інтелекту»

Захищено на засіданні ЕК № _____
протокол № ____ від ___ . ___ . 2025 р.
Оцінка ____ / _____
Голова ЕК
_____ Фесенко Г.В.
(підпис) (прізвище та ініціали)

Виконав:
Студент 4 курсу, групи КС-42
Спеціальності:
122 Комп'ютерні науки
(шифр і назва спеціальності підготовки)
Денищук Максим Сергійович
(прізвище, ім'я та по батькові, підпис)
Керівник PhD Родінко М. Ю.
(ступень, звання, прізвище та ініціали, пілпис)
Рецензент д. т. н., професор
Олійников Р.В.
(ступень, звання, прізвище та ініціали, пілпис)

Харків 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра інтелектуальних програмних систем і технологій
Рівень вищої освіти (освітньо-кваліфікаційний рівень) бакалавр
Спеціальність F3 Комп'ютерні науки

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри ІПСіТ

_____ Олег ОЛЕШКО
підпис ім'я, прізвище

"____" _____2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Денищук Максим Сергійович
(прізвище, ім'я, по батькові студента)

1. Тема роботи: «Розробка моделі та програмного забезпечення для прогнозування ризиків втрати клієнтів банківської установи за допомогою штучного інтелекту»

керівник роботи Родінко Марія Юріївна, PhD, доцент каф. ІПСіТ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "16" квітня 2025 року № 4101-5/962

2. Строк подання студентом роботи 2 червня 2025р.

3. Перелік питань, які потрібно розробити:

Ознайомитись з особливостями втрати клієнтів банківськими установами, провести огляд існуючих методів класифікації даних, познайомитися з алгоритмами роботи і методиками класифікації, створити модель прогнозування ймовірності втрати клієнтів, провести програмну реалізацію та експериментальне дослідження ефективності різних алгоритмів класифікації.

4. План роботи

№ з/п	Назви етапів роботи
1	Формулювання мети та повного переліку завдань для виконання кваліфікаційної роботи (КР).
2	Пошук та аналіз інформаційних джерел за темою КР.
3	Вивчення предметної області та особливостей втрати клієнтів у банківській сфері.
4	Огляд та порівняння сучасних методів класифікації для задач прогнозування відтоку.
5	Ознайомлення з принципами роботи алгоритмів: Logistic Regression, Decision Tree, Random Forest, SVM, XGBoost.
6	Підготовка та попередній аналіз датасету для навчання моделей.
7	Реалізація власних варіантів моделей класифікації на мові Python.
8	Проведення крос-валідації та експериментальне порівняння якості моделей.
9	Розробка програмного застосунку у вигляді REST API з використанням Flask або FastAPI.
10	Реалізація CRUD-функціональності для роботи з базою клієнтів.
11	Створення ендпоінту для прогнозування ймовірності відтоку клієнта.
12	Тестування програмного забезпечення та контейнеризація з Docker.
13	Підсумок отриманих результатів та оформлення пакету документів для процедури захисту КР в екзаменаційній комісії ННІ КН та ШІ.

5. Дата видачі завдання 13 січня 2025р.

Студент _____ М. С. Денищук
(підпис) (ініціали, прізвище)

Керівник роботи _____ М. Ю. Родінко
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Кваліфікаційна робота містить 112 сторінок, 4 розділи, 19 рисунків, 1 додаток, 24 джерела.

У роботі розглянуто процес розробки та застосування алгоритмів машинного навчання і виживального аналізу для автоматизованого прогнозування відтоку клієнтів банківської установи на основі цифрових даних, зокрема транзакційних, поведінкових та демографічних характеристик; визначено причини й наслідки втрати клієнтів; виконано огляд сучасних методів прогнозування та класифікації з особливою увагою до моделі пропорційних ризиків Кокса.

З метою створення інтегрованого рішення, що дозволяє оцінювати як ймовірність, так і час до відтоку клієнта, застосовано такі підходи: класичні класифікатори для оцінки ризику відтоку, survival-аналіз із моделлю Кокса для побудови часової шкали відтоку, реалізацію власних ітеративних алгоритмів на Python, побудову єдиного конвеєру препроцесингу з використанням scikit-learn та joblib, а також організацію сервісу на FastAPI із контейнеризацією через Docker.

Результатом є готовий до інтеграції модуль прогнозування, який на тестових даних показав C-index ≈ 0.91, точність інших класифікаторів – до 0.92, і надає часову шкалу відтоку з гнучким вибором періоду через REST-API. Запропоноване рішення відрізняється поєднанням можливості прогнозу класового відтоку та часу до події, забезпечує високу відтворюваність одержаних результатів і може бути легко розгорнуте у банківській IT-інфраструктурі.

Рекомендації щодо використання: інтегрувати модуль у CRM-систему для раннього виявлення клієнтів високого ризику, планування персоналізованих утримувальних кампаній та оцінки економічного ефекту від втрат клієнтів.

Ключові слова: SURVIVAL-АНАЛІЗ, ПРОГНОЗУВАННЯ ВІДТОКУ, MODEL COX, МАШИННЕ НАВЧАННЯ, FASTAPI, DOCKER, CRITICAL TIME, CHURN-API.

ABSTRACT

The qualification thesis comprises 112 pages, 4 chapters, 19 figures, 1 appendix, and 24 sources.

The study examines the process of developing and applying machine learning algorithms and survival analysis techniques for automated prediction of customer churn at a banking institution using digital data—specifically transactional, behavioral, and demographic characteristics. It identifies the causes and consequences of customer loss and provides a review of current forecasting and classification methods, with particular emphasis on the Cox proportional hazards model.

In order to create an integrated solution capable of estimating both the likelihood of churn and the time until churn, the following approaches were employed: traditional classifiers to assess churn risk; survival analysis with the Cox model to construct a time‑to‑churn scale; implementation of custom iterative algorithms in Python; construction of a unified preprocessing pipeline using scikit‑learn and joblib; and deployment of a service via FastAPI, containerized with Docker.

The outcome is a production-ready prediction module which achieved a C-index of approximately 0.91 on test data (and classifier accuracies up to 0.92), and provides a flexible churn timeline via REST endpoints. The proposed solution uniquely combines class-based churn prediction with time-to-event estimation, ensures high reproducibility of results, and can be seamlessly deployed within a banking IT infrastructure.

Recommendations for use: integrate the module into CRM systems for early identification of high-risk customers, design personalized retention campaigns, and evaluate the financial impact of potential customer losses.

Keywords: SURVIVAL ANALYSIS, CHURN PREDICTION, COX MODEL, MACHINE LEARNING, FASTAPI, DOCKER, TIME-TO-EVENT, CHURN-API.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

Деплой — процес розгортання застосунку у робочому середовищі;

API — Application Programming Interface, інтерфейс програмування для взаємодії між компонентами системи;

ASGI — Asynchronous Server Gateway Interface, контракт про виклики для веб-серверів для пересилки запитів до асинхронних програм Python;

Attrition_Flag — цільова змінна: 1 — клієнт відтік, 0 — клієнт продовжує обслуговування;

AutoML — Automated Machine Learning, підхід до автоматизації підбору моделі та гіперпараметрів;

Churn — відтік клієнтів; припинення клієнтом користування послугами банку;

CI/CD — Continuous Integration / Continuous Deployment, практика безперервної інтеграції та розгортання програмного забезпечення;

C-index — Concordance Index, міра якості survival-моделі: частка правильно упорядкованих пар спостережень;

CRM — Customer Relationship Management, системи управління взаємовідносинами з клієнтами;

CRUD — Create, Read, Update, Delete, чотири базові операції з даними в базі;

Docker — платформа контейнеризації, що забезпечує пакування додатків із усіма залежностями;

Explainable AI — набір методів і практик, що забезпечують інтерпретованість рішень моделей машинного навчання;

FastAPI — високопродуктивний веб-фреймворк для створення REST API на Python із підтримкою Pydantic;

GDPR — General Data Protection Regulation, Загальний регламент з захисту даних ЄС;

Hazard function — інтенсивність ризику h(t), миттєва ймовірність настання події в момент t за умови виживання до t;

IBM SPSS — IBM Statistical Package for the Social Sciences, пакет для статистичної обробки даних;

joblib — бібліотека Python для серіалізації об’єктів, зокрема навчальних моделей;

LSTM — Long Short-Term Memory, різновид рекурентної нейронної мережі для моделювання послідовностей з довготривалою залежністю;

Pipeline — конвеєр перетворень даних: послідовність трансформерів та оцінювачів у Scikit-learn;

Pydantic — бібліотека для валідації та серіалізації даних у FastAPI;

REST — Representational State Transfer, архітектурний стиль побудови веб-сервісів на основі HTTP;

SAS — Statistical Analysis System, програмне забезпечення для статистичного аналізу даних;

uvicorn — ASGI-сервер для запуску асинхронних Python-додатків;

SQLAlchemy — ORM-бібліотека для роботи з реляційними СУБД у Python;

Timeline — часовий інтервал, місяці, від дати відкриття рахунку до моменту прогнозу;

ВСТУП

У сучасних умовах підвищеної конкуренції на ринку фінансових послуг ефективне управління клієнтською базою стає одним із ключових факторів успіху банківської установи. Одним із найважливіших завдань у цій сфері є своєчасне виявлення клієнтів, що можуть піти з банку, і розробка заходів щодо їх утримання. Неврахування цього ризику призводить до значних фінансових втрат і зниження конкурентоспроможності.

Проблема прогнозування відтоку клієнтів є надзвичайно актуальною для банків і інших фінансових організацій. З огляду на великий обсяг даних про поведінку клієнтів, впровадження сучасних методів машинного навчання та аналізу виживаності дозволяє підвищити точність прогнозів і приймати проактивні рішення щодо клієнтів із високим ризиком відтоку. Особливо перспективним є застосування моделі пропорційних ризиків Кокса, яка враховує часовий фактор і дозволяє не лише передбачати, чи клієнт піде, але й оцінювати часовий горизонт можливого відтоку.

Об’єктом дослідження є процес розробки та застосування алгоритмів машинного навчання і виживального аналізу для автоматизованого прогнозування відтоку клієнтів банку на основі цифрових даних, а саме транзакційних, поведінкових та демографічних характеристик.

Предметом дослідження є методи та алгоритми прогнозування відтоку клієнтів на основі даних CRM та історії транзакцій, зокрема побудова та налаштування моделей машинного навчання і моделі виживаності.

Метою роботи є створення інтегрованого рішення для прогнозування ймовірності відтоку клієнтів банку з урахуванням часових факторів та реалізувати REST-API, яке забезпечує автоматизоване й оперативне отримання цих прогнозів.

У роботі проведено аналіз факторів клієнтського відтоку в банківській сфері та виконано попередню обробку даних із виявленням аномалій і заповненням пропусків. Розроблено та теоретично обґрунтовано власні

реалізації п’яти класичних алгоритмів класифікації і моделі виживального аналізу Cox-PH, оцінені їхні показники якості за допомогою крос-валідації та метрик F1-score і C-index. Створено REST-API на базі FastAPI з функціоналом CRUD для роботи з клієнтами й кінцевими точками прогнозування часової ймовірності відтоку, що розгортається в Docker-контейнері. Запропоноване рішення може бути інтегроване в інформаційні системи банків для своєчасного виявлення ризикових клієнтів, оптимізації маркетингових заходів і зниження фінансових втрат.

РОЗДІЛ 1. АНАЛІЗ ТА ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Втрата клієнтів у банківській сфері: причини та наслідки

В умовах високої конкуренції на фінансовому ринку збереження існуючих клієнтів виступає критично важливою складовою стабільного розвитку банку. Клієнти йдуть тоді, коли їхні очікування не відповідають отриманому сервісу: наприклад, постійні затримки обробки платежів, довгі черги у відділеннях або складна робота мобільного додатку створюють у користувача відчуття незадоволення і недовіри. Не менше значення має прозорість тарифної політики: приховані комісії або раптові зміни умов обслуговування без попередження змушують клієнтів шукати більш передбачувані й зрозумілі рішення.

Одним із факторів, що посилюють відтік, є відсутність персоналізованих пропозицій. Клієнти дедалі більше очікують від банку сервісу, адаптованого під їхні потреби — від персональних рекомендацій щодо інвестицій до індивідуальних кредитних лімітів. Якщо ж банк надсилає клієнтам загальні маркетингові розсилки або не враховує особливості їхньої фінансової поведінки, зацікавленість у співпраці поступово згасає. У той самий час фінтех-стартапи та інші банки активно використовують аналітику та машинне навчання для створення дійсно «розумних» пропозицій, що ще більше підвищує планку очікувань.

Крім того, відтік клієнтів нерідко спричинений зовнішніми обставинами: зміна місця роботи, переїзд до іншого регіону чи зміна доходу можуть зробити поточний банк менш зручним для щоденних операцій. Проте навіть у цьому разі лояльність і якість обслуговування залишаються ключовими чинниками, які визначають те, чи захоче клієнт перенести свій банківський рахунок і на якій умовній “вартості” для нього це відбудеться.

Наслідки відтоку позначаються не лише на короткострокових фінансових показниках. Втрата клієнта означає втрату джерела депозитних

коштів, які банк може використовувати для кредитування інших клієнтів, а також зменшення середнього доходу на одного користувача. Оскільки залучення нового клієнта обходиться банку в рази дорожче за утримання існуючого, підвищується тиск на маркетинговий бюджет і змінюється стратегія розвитку. Масовий відтік клієнтів також псує репутацію на ринку: негативні відгуки у соціальних мережах і медіа можуть відлякати потенційних клієнтів ще до першої взаємодії.

Таким чином, для будь-якого банку важливо не лише своєчасно ідентифікувати сигнали наближення клієнта до рішення про вихід, а й розробити проактивні механізми утримання. Саме ця задача стає предметом нашого дослідження: зрозуміти глибинні причини клієнтського відтоку та застосувати сучасні методи прогнозування, аби мінімізувати його негативні наслідки й забезпечити стійкий розвиток фінансової установи.

1.2 Методи виявлення ризику відтоку клієнтів

Сучасні банки дедалі більше зосереджуються на виявленні клієнтів із високим ризиком відтоку ще до того, як вони остаточно розірвуть стосунки із фінансовою установою. На ранніх етапах цю задачу часто вирішують за допомогою експертних правил, котрі формулюються бізнес-аналітиками на основі досвіду та практики. Наприклад, клієнтів із кількістю транзакцій менш ніж дві на квартал чи зі зниженням середнього залишку більш ніж на 30 % відносять до групи підвищеного ризику. Такі правила є зрозумілими й прозорими, однак вони не враховують складних і часто нелінійних взаємозв’язків між численними ознаками поведінки клієнта.

Поступово класичні статистичні моделі, зокрема логістична регресія та дискримінантний аналіз, доповнили правила машинними методами. Логістична регресія дозволяє формалізувати й кількісно оцінити вплив кожного фактора — віку клієнта, кількості звернень у службу підтримки чи зміни обсягів транзакцій — на ймовірність відтоку й отримати інтерпретовані коефіцієнти. Кластеризація допомагає виявити однорідні сегменти, серед

яких можна ідентифікувати «групи підвищеного ризику». Проте ці підходи зазвичай вимагають суворого дотримання статистичних припущень і забезпечують обмежену точність на складних даних із великою кількістю ознак.

Найбільш прогресивними стали алгоритми машинного навчання: ансамблі дерев підтримкові вектори, а також нейронні мережі. Вони можуть автоматично виявляти нелінійні залежності між десятками ознак, обробляти категоріальні та кількісні змінні й забезпечувати високу точність класифікації. Наприклад, XGBoost, комбінуючи сотні спрощених рішень-дерев, створює «міцний» прогноз і витримує неповні чи нерівномірно розподілені дані. Нейронні мережі — зокрема рекурентні LSTM — дозволяють аналізувати послідовності подій і прогнозувати відтік з урахуванням динаміки.

Попри це, більшість вищезгаданих методів ігнорують часовий аспект: коли саме клієнт піде. Для вирішення цієї проблеми використовують методи виживального аналізу, зокрема модель пропорційних ризиків Кокса. Вона дозволяє не лише класифікувати клієнтів за ризиком, а й оцінювати часовий горизонт ймовірного відтоку. Додатково в банківській практиці поєднують багаторівневі підходи: прості правила для швидкого відсіювання низькоризикових груп і складні моделі машинного навчання або виживального аналізу для глибшої оцінки вищих ризиків.

Таким чином, серед багатьох доступних методів дедалі виразніше вирізняється поєднання машинного навчання і виживального аналізу: це дозволяє враховувати як складні патерни поведінки, так і часові характеристики відтоку клієнта, забезпечуючи найбільш повне і точне прогнозування.

1.3. Огляд сучасних підходів до прогнозування клієнтського відтоку

Упродовж останнього десятиліття завдяки зростанню обсягів доступних даних та потужностей обчислювальних ресурсів прогнозування

клієнтського відтоку перетворилося з простого правило налагоджувального завдання на повноцінну науково-технічну дисципліну. Одним із найпоширеніших підходів залишаються ансамблеві методи — зокрема, Random Forest та Gradient Boosting. Вони здатні обробляти великі набори ознак і вловлювати складні взаємозв’язки, що пояснює їхню високу точність на структурованих банківських даних. При цьому такі моделі досить добре протистоять переобученню завдяки встроєним механізмам регуляризації та випадковості.

Разом із ансамблями дедалі більше уваги привертають глибокі нейронні мережі, насамперед рекурентні та трансформерні архітектури, пристосовані для послідовностей: вони аналізують часові ряди транзакцій, взаємодій із мобільним додатком чи звернень до служби підтримки. Завдяки цьому вдається виявити нетривіальні патерни поведінки — наприклад, поступове зниження частоти платежів або зміну структури витрат — і прогнозувати відтік задовго до появи явних ознак незадоволення клієнта.

Проте як класифікаційні, так і рекурентні нейронні мережі вимагають великої кількості даних та значних зусиль для інтерпретації результатів. Саме тому банки часто інтегрують у робочі конвеєри автоматизовані платформи AutoML, які самостійно підбирають оптимальні архітектури, передобробку ознак та гіперпараметри. Це скорочує час розробки й дозволяє швидко тестувати різні підходи без глибоких знань у сфері машинного навчання.

Окремою лінією розвитку стала комбінація класифікаційних моделей із моделями виживального аналізу. Наприклад, на етапі відбору ознак застосовують XGBoost для виділення найбільш значущих змінних, а потім у побудові прогнозу часу до відтоку використовують оцінку пропорційних ризиків Кокса. Такий гібрид дозволяє отримати не тільки ймовірність «churn», але й часову шкалу його настання, що є надзвичайно цінним для планування маркетингових і утримувальних кампаній.

Усе частіше у практиці застосовують online learning і стрімінгові алгоритми, які адаптуються до змін у поведінці клієнтів у реальному часі. Це

особливо важливо у фінтех-сфері, де поява нових продуктів або регуляторних змін може миттєво вплинути на патерни використання сервісів. Комбінування пакетного навчання з online learning дозволяє одночасно зберігати постійне оновлення моделі та гарантувати її стабільність.

Таким чином, сучасні підходи до прогнозування клієнтського відтоку відрізняються багаторівневою стратегією: від інтерпретованих ансамблів дерев до складних нейромережних та survival-гібридних систем, які спільно дають змогу досягти балансу між точністю прогнозу, інтерпретованістю та швидкістю реакції на зміну ринкових умов.

1.4. Аналіз існуючих рішень та інструментів

На сучасному ринку представлено низку готових рішень для прогнозування клієнтського відтоку, які охоплюють як окремі компоненти, так і комплексні end-to-end сервіси. Одним із найпоширеніших варіантів є платформи AutoML від провайдерів хмарних сервісів: Google Cloud AutoML, Azure Machine Learning та AWS SageMaker Autopilot. Вони дозволяють експериментувати з різними алгоритмами класифікації та регресії у кілька кліків, автоматизуючи підготовку даних, відбір ознак і пошук оптимальних гіперпараметрів. Однак висока вартість та закрита архітектура можуть обмежувати гнучкість інтеграції у внутрішні IT-процеси банку.

У відкритому програмному забезпеченні найбільшою популярністю користуються бібліотеки scikit-learn, XGBoost, LightGBM та CatBoost для класифікаційних задач, а також lifelines та scikit-survival для виживального аналізу. Ці інструменти надають багатий API для налаштування моделей, аналізу важливості ознак і побудови кривих виживання. Разом із тим, їхня інтеграція у production-середовище вимагає побудови власного пайплайну для обробки даних, розгортання моделі та моніторингу якості прогнозів.

Ще одним напрямом є спеціалізовані рішення від великих IT-компаній та стартапів. Вони пропонують візуальні інтерфейси для аналізу даних, інтегровані механізми explainable AI та вбудовані сервіси розгортання в хмарі

чи на власних серверах. Такі платформи зменшують бар’єр входження для аналітичних команд, але часто потребують окремої ліцензії та не завжди відповідають вимогам безпеки великих банків.

При виборі конкретного інструменту банківські IT-архітектори враховують цілий ряд критеріїв: відповідність нормам фінансової безпеки та GDPR, прозорість алгоритмів, можливість інтеграції зі скарбницею даних і системами CRM, а також масштабованість рішення. У багатьох випадках банки обирають гібридну стратегію: використовують open-source-модулі для розробки прототипів та інтегрують їх у власні DevOps-конвеєри, а критично важливі production-сервіси запускають на ліцензійних платформах із додатковим рівнем підтримки та безпеки.

У підсумку, хоча на ринку існує широкий арсенал готових рішень для прогнозування відтоку, жодне з них не забезпечує «з коробки» ідеальної відповідності всім внутрішнім вимогам банку. Це вимагає розробки кастомізованих пайплайнів, що поєднують найкращі практики open-source з центральною бізнес-логікою та IT-інфраструктурою фінансової установи. Саме такий підхід ми використаємо в нашому дослідженні, обираючи інструменти та технології, що максимально відповідають вимогам задачі.

1.5 Вибір методу реалізації задачі

У результаті комплексного аналізу предметної області та існуючих підходів було визначено доцільність використання гібридної архітектури, яка поєднує класифікаційні моделі з методами виживального аналізу. З одного боку, завдання попередньої класифікації клієнтів за ризиком відтоку реалізується за допомогою ансамблевого алгоритму Gradient Boosting, що забезпечує високу прогностичну точність завдяки здатності виявляти нелінійні кореляції між вхідними ознаками та вбудованим механізмам регуляризації. З іншого боку, для кількісної оцінки часу до настання події «відтік клієнта» обрана модель пропорційних ризиків Кокса, яка дозволяє

врахувати цензурування спостережень і побудувати оцінку кумулятивної небезпеки з урахуванням впливу кожної ознаки.

Формалізація вхідних даних здійснюється за допомогою пайплайна на базі ColumnTransformer. Категоріальні змінні піддаються One-Hot кодуванню, що гарантує коректне відображення дискретних рівнів без введення упередженості, а числові ознаки стандартизуються до нульового середнього та одиничної дисперсії для узгодження шкал та запобігання домінуванню однієї змінної над іншими. Такий підхід забезпечує репрезентативну структуру даних та сприяє коректній роботі обох компонентів моделі.

Для класифікаційного компоненту реалізовано дві версії: бібліотечний XGBoost-пайплайн для production-сценарію та набір власних реалізацій алгоритмів, які використовуються у науковому аналізі для порівняння їх ефективності. Survival-компонент формує окремий пайплайн із власною обгорткою CoxPH, що забезпечує адаптоване рішення задачі максимізації правдоподібності з обчисленням коваріаційного матричного градієнта і регуляризацією методом Ньютона–Рафсона.

Збереження та відновлення моделей здійснюється через joblib, що дозволяє швидко серіалізувати повні пайплайни разом із налаштованими гіперпараметрами та метаданими. Розгортання системи прогнозування організовано на основі FastAPI, що забезпечує автоматизовану валідацію вхідних даних за допомогою Pydantic-схем, високу продуктивність та можливість масштабування у контейнеризованому середовищі Docker Compose із використанням бази даних PostgreSQL для зберігання історичних даних клієнтів.

Таким чином, обрана методологія поєднує перевірені технології машинного навчання та survival-аналізу, стандартизовану препроцесингову схему і сучасний веб-фреймворк для забезпечення відтворюваності, прозорості та високої точності системи прогнозування клієнтського відтоку.

1.6 Існуючі рішення для прогнозування часу відтоку клієнтів

У науковій літературі та практиці фінансових установ для прогнозування «коли» клієнт може припинити користуватися послугами банку застосовують як класичні методи класифікації, так і моделі виживального аналізу. Найпоширеніші підходи спираються на бібліотеки Python: пакет lifelines реалізує різні варіанти регресії Кокса, та непараметричні методи, Каплана–Мейєра, а scikit-survival додає інструменти для побудови ансамблевих і підтримуваних векторних машин із врахуванням часових даних. Ці бібліотеки вже містять механізми обробки правих цензурованих спостережень, що є необхідним у задачі прогнозування часу до відтоку.

Водночас у рамках традиційного машинного навчання популярні рішення на основі scikit-learn, які обчислюють ймовірність відтоку на певний момент або у фіксованому часовому вікні. Проте вони не враховують структуру часових даних безпосередньо, а потребують формування часових інтервалів і ознак на їхній основі — відтак втрачають інформацію про момент настання події.

Серед комерційних платформ із підтримкою виживального аналізу можна відзначити SAS Survival Analysis і IBM SPSS Survival, що вбудовують регресію Кокса та інші методи в робочі конвеєри. Вони забезпечують інтерфейс «point-and-click» та автоматичну валідацію даних, але часто виявляються надмірно важкими для інтеграції в сучасні мікросервісні архітектури через їхню закриту природу та високі ліцензійні витрати.

Таким чином, хоча існує значний вибір як відкритих, так і комерційних інструментів для моделювання часу до відтоку, відсутнє компактне рішення, яке поєднувало б: підтримку правої цензури та динамічного прогнозу виживання, можливість розгортання в хмарі чи контейнері через REST-API та гнучкість налаштування під конкретний банківський сценарій.

Ця прогалина визначає необхідність створити власну пайплайн-архітектуру на основі Cox PH, інтегровану з FastAPI, що

поєднуватиме точність виживального аналізу з простотою промислового розгортання.

1.7 Висновки до розділу

У результаті аналізу предметної області було підтверджено, що прогнозування клієнтського відтоку вимагає не лише визначення ризику самої події, а й оцінки часу до неї. Стандартні класифікаційні методи, зокрема ансамблеві алгоритми, демонструють високу точність у бінарному розпізнаванні клієнтів із потенційним відтоком, проте залишають без уваги часовий вимір, необхідний для своєчасного прийняття превентивних заходів.

Огляд існуючих відкритих і комерційних рішень показав, що хоч бібліотеки lifelines та scikit-survival забезпечують багатий інструментарій для survival-аналізу, вони не інтегровані за замовчуванням у сучасні мікросервісні архітектури та потребують суттєвих доопрацювань для розгортання через REST-API. Комерційні пакети SAS та IBM SPSS, хоч і пропонують інтерактивне середовище для аналізу, часто є надмірно громіздкими та важкими в інтеграції.

Виходячи з ідентифікованої прогалини, було обґрунтовано вибір комбінованого підходу: класифікатор для оцінки ймовірності відтоку та модель пропорційних ризиків Кокса для прогнозу часу до події. Такий гібрид забезпечує повнішу аналітичну картину: бінарне рішення про ризик відтоку доповнюється часовою шкалою ймовірності, що дозволяє планувати маркетингові й утримувальні програми з урахуванням термінів, а не лише фактів.

Застосування єдиного пайплайну обробки даних гарантує узгодженість і відтворюваність результатів, а реалізація через FastAPI і Docker-контейнери забезпечує швидке та безболісне розгортання у будь-якій IT-інфраструктурі банку. Такий комплексний підхід створює міцну основу для подальшої розробки, тестування та впровадження ефективної системи прогнозування клієнтського відтоку.

РОЗДІЛ 2. ТЕОРЕТИЧНИЙ ОПИС МОДЕЛІ

2.1. Формалізація задачі класифікації

Нехай маємо вибірку з n спостережень

$$D = \{(x_i, y_i)\}_{i=1}^{n},$$

(2.1)

де $x_i = [x_{i1}, x_{i2}, ..., x_{ip}]^\top \in R^p$ — вектор з p ознак клієнта (вік, стать, кількість транзакцій тощо);

$y_i \in \{0, 1\}$ — бінарна цільова змінна, де $y_i = 1$ відповідає «Attrited Customer», $y_i = 0$ — «Existing Customer».

Метою класифікації є побудова детермінованої функції

$$f : R^p \to \{0, 1\},$$

(2.2)

або, більш загально, ймовірнісної моделі

$$\hat{p}(x) = Pr(y = 1|x),$$

(2.3)

за якою остаточне рішення приймається порівнянням з порогом τ:

$$f(x) = \{1, \hat{p}(x) \geq \tau; 0, \hat{p}(x) < \tau;\}$$

(2.4)

Для оцінювання якості одержаної моделі вводиться функція втрат, що для одного спостереження записується як

$$\ell\left(\hat{p}(x_i), y_i\right) = -\left[y_i log\hat{p}(x_i) + (1 - y_i) log\left(1 - \hat{p}(x_i)\right)\right].$$

(2.5)

Оптимізаційна задача формулюється як мінімізація сумарної втрати по всій вибірці з додаванням регуляризаційного члена $R(\theta)$ (щоб уникнути перенавчання):

$$min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell\left(\hat{p}\left(x_i; \theta\right), y_i\right) + \lambda R(\theta) \tag{2.6}$$

де $R(\theta)$ — регуляризація (L1, L2 тощо);

λ — гіперпараметр.

Вектор параметрів θ та форма регуляризації залежать від обраного алгоритму. Стандартизація числових ознак та One-Hot кодування категоріальних здійснюються єдиним препроцесинговим пайплайном.

Приклад реалізації пайплайна з логістичною регресією наведено в Додатку А.1.

2.2 Формалізація задачі класифікації

У цьому розділі ми розглянемо шість ключових алгоритмів, які застосовуються для задачі прогнозування відтоку клієнтів банку. Для кожної моделі наведемо її математичну формалізацію, ключові гіперпараметри та основні властивості.

2.2.1 Decision Tree

Метод дерева рішень належить до сімейства ієрархічних алгоритмів класифікації, які здійснюють рекурсивне розбиття простору ознак на підмножини, всередині яких зберігається гомогенність цільової змінної. Нехай D_t — підмножина навчальних спостережень у вузлі t, тоді кожне розбиття за ознакою j з порогом s формує лівий підвузол

$$D_{\ell} = \left\{(x, y) \in D_t : x_j \leq s\right\} \tag{2.2.1.1}$$

та правий підвузол

$$D_r = \{(x, y) \in D_t : x_j > s\}.$$

(2.2.1.2)

Оптимальний вибір (j^*, s^*) здійснюється через максимізацію приросту інформації (information gain) або зменшення міри нечистоти. Наприклад, для критерію Gini impurity

$$\Delta i = i(D_t) - \frac{|D_\ell|}{|D_t|} i(D_\ell) - \frac{|D_r|}{|D_t|} i(D_r),$$

(2.2.1.3)

де

$$i(D) = 1 - \sum_{k \in \{0,1\}} p_k^2, \quad p_k = \frac{1}{|D|} \sum_{(x,y) \in D} I(y = k).$$

(2.2.1.4)

Альтернативно застосовують ентропію Ітамі (information entropy):

$$i_{Ent}(D) = - \sum_{k \in \{0,1\}} p_k \log_2(p_k).$$

(2.2.1.5)

Рекурсія зупиняється при досягненні однієї з умов: глибина D перевищує заздалегідь заданий максимум, у вузлі менше ніж n_{min} зразків або поділ не призводить до значущого зменшення нечистоти. У листовому вузлі t прогнозована ймовірність класу «1» обчислюється як

$$\hat{p}(x \in t) = \frac{1}{|D_t|} \sum_{(x,y) \in D_t} y.$$

(2.2.1.6)

Головні переваги дерев рішень полягають у їх інтерпретованості: шлях від кореня до листа читається як набір умов «якщо – то», що спрощує валідацію моделі фахівцями банку. Однак, без контролю глибини та

мінімального числа зразків у листі алгоритм схильний до переобучення, формуючи надмірно складні дерева, які відтворюють випадковий шум навчальних даних. Щоби зменшити дисперсію, зазвичай застосовують ансамблеві підходи або регуляризацію у вигляді обмеження максимальної глибини та мінімального числа зразків для розбиття.

2.2.2 Logistic Regression

Логістична регресія є лінійною моделлю ймовірностей, що застосовує сигмоїдну функцію зв'язку для прогнозування ймовірності належності до «класу відтоку» $y = 1$. Нехай $x = \left(x_1, x_2, ..., x_p\right)^\top$ — вектор ознак клієнта, а $\beta = \left(\beta_1, \beta_2, ..., \beta_p\right)^\top$, β_0 — вільний член. Тоді модель задається рівнянням:

$$logit\,\hat{p}(x) = log\frac{\hat{p}(x)}{1-\hat{p}(x)} = \beta_0 + \sum_{j=1}^{p}\beta_j x_j.$$

(2.2.2.1)

Імовірність відтоку обчислюється через сигмоїду

$$\hat{p}(x) = \sigma\left(\beta_0 + \beta^\top x\right), \quad \sigma(z) = \frac{1}{1+e^{-z}}.$$

(2.2.2.2)

Параметри $\{\beta_0, \beta\}$ оцінюються шляхом мінімізації регуляризованої логістичної втрати:

$$min\ -\ \frac{1}{n}\sum_{i=1}^{n}\left[y_i log\sigma(z_i)\ +\ (1\ -\ y_i)log(1\ -\ \sigma(z_i))\right] + \frac{\lambda}{2}||\beta||_2^2.$$

(2.2.2.3)

$$z_i = \beta_0 + \beta^\top x_i,$$

(2.2.2.4)

де $\lambda \geq 0$ — коефіцієнт L2-регуляризації, що зменшує ризик переобучення та забезпечує стійкість оцінок за наявності корельованих ознак.

У логістичній регресії кожен коефіцієнт β_j має інтерпретацію як зміну логарифма відношення шансів. Зокрема, збільшення ознаки x_j на одиницю призводить до приросту лог-відношення шансів відтоку на величину β_j. Це забезпечує пряму та прозору інтерпретацію впливу кожної змінної на ймовірність відтоку клієнта.

Ефективність моделі найвища у тих випадках, коли залежність ймовірності відтоку клієнта може бути задовільно апроксимована лінійною комбінацією вхідних ознак. Перед навчанням логістичної регресії рекомендується стандартизувати числові змінні та виконати one-hot кодування категоріальних, щоби забезпечити однорідний масштаб ознак і уникнути непропорційної ваги окремих змінних у процесі оптимізації.

Водночас логістична регресія вразлива до складних нелінійних взаємодій між ознаками, яких вона апріорі не моделює. Щоби компенсувати це обмеження, необхідний ретельний відбір і, за потреби, попередня трансформація ознак. Без такої підготовки даних модель може виявитися недостатньо гнучкою для точного опису реальних процесів відтоку клієнтів.

2.2.3 Random Forest

Метод випадкового лісу належить до класу ансамблевих алгоритмів та базується на комбінуванні великої кількості індивідуальних дерев рішень для підвищення стабільності та точності прогнозу. Ідея полягає в тому, що кожне окреме дерево «голосує» за свій варіант класифікації, а остаточне рішення приймається більшістю голосів.

Формально, нехай ми маємо навчальний набір

$$D = \{(x_i, y_i)\}_{i=1}^{n}, \quad x_i \in R^p, \; y_i \in \{0, 1\}. \tag{2.2.3.1}$$

Алгоритм випадкового лісу побудований на послідовному генеруванні великої кількості дерев рішень, кожне з яких навчається на власній випадковій вибірці даних. Спочатку з початкового набору спостережень *D* за допомогою бутстреп-відбору формують нову вибірку $D^{(m)}$ із повтореннями: це забезпечує різноманітність навчальних даних для кожного дерева та знижує кореляцію між ними.

Далі в процесі розгалуження кожного дерева на кожному вузлі випадково обирається підмножина ознак потужності q < p із загального набору p ознак. Серед цих q кандидатів шукають оптимальний поріг розділу, який максимізує обраний критерій чистоти. Такий підхід гарантує, що кожне дерево відшліфовує власні особливості вхідного простору, а їх агрегування в результаті забезпечує високу точність та стійкість до шумових даних.

Таким чином, кожне дерево $T^{(m)}$ є дещо «зашумленим» варіантом класичного дерева рішень, що знижує ймовірність перенавчання.

Для нового спостереження x^* моделі Random Forest дають відповіді $T^{(1)}(x^*), \ldots, T^{(M)}(x^*)$. Остаточний прогноз (в разі класифікації) обчислюється як

$$\hat{y} = majority\{T^{(m)}(x^*)\}_{m=1}^{M},$$

(2.2.3.2)

або в імовірнісному вигляді:

$$\hat{p}(x^*) = \frac{1}{M}\sum_{m=1}^{M} I\{T^{(m)}(x^*) = 1\}.$$

(2.2.3.3)

Ансамбль випадкових лісів вирізняється високою стабільністю та стійкістю до шуму даних, оскільки усереднення результатів великої кількості дерев, натренованих на бутстреп-вибірках з випадковим підбором ознак, ефективно розгладжує випадкові флуктуації окремих моделей. Крім того, цей підхід надає вбудований механізм кількісної оцінки важливості ознак:

шляхом аналізу зниження точності прогнозу після виключення конкретної ознаки можна ранжувати їх за внеском у загальну якість моделі. Завдяки мінімальній чутливості до початкових налаштувань гіперпараметрів — достатньо задати помірну кількість дерев і глибину — Random Forest демонструє добрі результати «з коробки», що значно спрощує процес інтеграції алгоритму в робочі конвеєри.

Водночас слід враховувати, що ансамбль із сотень дерев суттєво ускладнює інтерпретацію моделі: на відміну від одного дерева рішень, пояснення кінцевого прогнозу потребує спеціальних методів. Крім того, обчислювальні витрати на тренування та прогнозування зростають із розміром підмножини ознак M та глибиною дерев, що може створювати вузькі місця за обмежених ресурсів апаратного забезпечення та у випадках необхідності швидкої відповіді в режимі реального часу.

2.2.4 Support Vector Machine (SVM)

Support Vector Machine (SVM) є одним із найпотужніших алгоритмів для двокласової класифікації, особливо коли дані можуть бути відокремлені нечіткою, проте стійкою гіперплощиною.

У класичній лінійній формі SVM розв’язує оптимізаційну задачу

$$min_{w,b,\xi} \frac{1}{2}||w||^2 + C\sum_{i=1}^{n} \xi_i \quad \text{за умов} \quad y_i(w^\top x_i + b) \geq 1 - \xi_i,\ \xi_i \geq 0 \tag{2.2.4.1}$$

де $x_i \in R^p$ — вхідний вектор ознак,

$y_i \in \{-1, +1\}$ — цільова мітка,

w та b — параметри гіперплощини,

ξ_i — нерегулярні змінні (slack variables), що дозволяють порушувати умову відступу,

$C > 0$ — гіперпараметр, який балансує ширину відступу та суму штрафів за помилки класифікації.

Використання штрафних змінних ξ_i робить SVM «м'яким» (soft-margin), тобто дозволяє алгоритму бути стійким до шуму та невеликих накладень між класами. Параметр C визначає, наскільки суворо алгоритм карає за порушення умов класифікації: великі значення C зменшують кількість помилок на тренувальному наборі, але ризикують перенавчанням; малі C розширюють відступ, роблячи гіперплощину більш узагальненою.

При застосуванні SVM важливо відзначити декілька ключових переваг. По-перше, метод має чітке теоретичне обґрунтування: задача оптимізації є опуклою, і за використання відповідних ядерних функцій гарантується знаходження глобального оптимуму. Такий підхід забезпечує доведену стабільність рішення і відсутність ризику потрапляння у локальні мінімумі.

По-друге, SVM проявляє високу стійкість до шуму в даних. Оскільки в основі стоїть максимізація відступу між класами, а не мінімізація безпосередньої помилки класифікації, невеликі відхилення окремих спостережень не призводять до суттєвого зсуву кордону розділення.

По-третє, алгоритм вирізняється гнучкістю завдяки можливості роботи з різними ядрами. Завдяки ядровому трюку можна ефективно моделювати нелінійні залежності без необхідності побудови складних багаторівневих архітектур, що особливо корисно при роботі з високовимірними даними.

Водночас, існують і певні обмеження цього методу. Зі збільшенням обсягу навчальної множини nn обчислювальна складність SVM суттєво зростає: розв'язання квадратичної задачі при навчанні та багаторазові обчислення ядра під час прогнозування роблять виконання моделі повільнішим на великих даних. Також налаштування гіперпараметрів — таких як штраф C, параметр ядра γ або ступінь полінома d — потребує ретельної перехресної валідації, що ще більше збільшує час підготовки моделі. Окрім того, у разі складних ядер інтерпретація вагових коефіцієнтів w у вихідному просторі ознак стає непрозорою, що може ускладнювати аналіз отриманих результатів.

2.2.5 Extreme Gradient Boosting (XGBoost)

XGBoost ґрунтується на ідеї послідовного будування ансамблю дерев рішень із використанням градієнтного спуску у просторі функцій. Нехай наша модель на кроці ttt задається сумарною функцією

$$\hat{y}_t^{(t)} = \sum_{k=1}^{t} f_k(x_i),$$

(2.2.5.1)

де кожне f_k — це дерево рішення. На кожному кроці алгоритм шукає нове дерево f_t, яке мінімізує регуляризовану функцію втрат

$$\varsigma = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t),$$

(2.2.5.2)

де l — обрана функція втрат (наприклад, логістична для класифікації), а

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

(2.2.5.3)

— член регуляризації, що залежить від кількості листків T та ваг w_j у цих листах.

Застосування другого порядку наближення функції втрат за допомогою розгортки Тейлора дозволяє ефективно обчислювати градієнт і гессіан для кожного листка та швидко будувати оптимальні розгалуження. Параметри γ та λ контролюють складність моделі та запобігають перенавчанню шляхом покарання надмірної кількості листків і великих ваг.

XGBoost вирізняється високою гнучкістю: він підтримує як числові, так і категоріальні ознаки, автоматично працює з пропущеними значеннями і забезпечує паралельну обробку даних для прискорення навчання. Крім того, додана регуляризація на рівні листків підвищує здатність моделі до

узагальнення, а можливості глибокого налаштування гіперпараметрів дають змогу досягти високої точності навіть на складних наборах даних.

Однак XGBoost є відносно ресурсоємним: зростання кількості дерев чи глибини кожного дерева збільшує час тренування та обсяг пам'яті. Також важливо ретельно підбирати швидкість навчання (η), обсяг підвибірки та параметри регуляризації, інакше модель може переобучитися або, навпаки, навчитися надто повільно.

2.2.6 Cox Proportional Hazards Model

Метод пропорційних ризиків Кокса є центральним у виживальному аналізі, оскільки дозволяє моделювати час до настання події без необхідності надавати форму базового ризику. Нехай T_i — випадкова величина часу «виживання», у нашому випадку періоду співпраці з клієнтом, а $X_i \in R^p$ — вектор його ознак. Функцію миттєвого ризику визначають як

$$h(t \mid X_i) = \lim_{\Delta t \to 0} \frac{Pr(t \leq T_i < t+\Delta t \mid T_i \geq t, X_i)}{\Delta t}.$$

(2.2.6.1)

У класичній пропорційній моделі Кокса приймають, що

$$h\left(t|X_i\right) = h_0(t)\, exp\left(X_i^{\top}\beta\right),$$

(2.2.6.2)

де $h_0(t)$ — невідома базова функція ризику, що залежить лише від часу;

$\beta \in R^p$ — вектор параметрів, який квантифікує вплив ознак на ризик відтоку.

Оскільки $h_0(t)$ не задається апріорі, оцінювання β базується на частковому правдоподібності. Позначимо моменти події для клієнтів так, що $t_{(1)} < t_{(2)} < \cdots < t_{(m)}$ — відсортовані часи відтоку. Нехай клієнт i_j відпав у час

$t_{(j)}$. Тоді часткова ймовірність того, що з-поміж тих, хто «вижив»» до $t_{(j)}$, саме i_j відпаде, становить

$$\frac{exp\left(X_{i_j}^{\top}\beta\right)}{\sum_{i\in R\left(t_{(j)}\right)} exp\left(X_i^{\top}\beta\right)}$$

(2.2.6.3)

де $R(t)$ — ризиковий набір клієнтів, що залишались активними до часу t. Повна часткова правдоподібність записується як

$$L(\beta) = \prod_{j=1}^{m} \frac{exp\left(X_{i_j}^{\top}\beta\right)}{\sum_{i\in R\left(t_{(j)}\right)} exp\left(X_i^{\top}\beta\right)}.$$

(2.2.6.4)

Щоб знайти оцінки $\hat{\beta}$, максимізують лог-часткову правдоподібність

$$\ell(\beta) = \sum_{j=1}^{m}\left[X_{i_j}^{\top}\beta - log \sum_{i\in R\left(t_{(j)}\right)} exp\left(X_i^{\top}\beta\right)\right].$$

(2.2.6.5)

Завдяки відсутності необхідності моделювати $h_0(t)$, оцінки виходять стійкими до непевності в формі базового ризику.

Інтерпретація β_j у моделі Кокса полягає в тому, що $exp\left(\beta_j\right)$ — співвідношення ризиків при зростанні x_j на одиницю, за фіксованих інших ознак. Якщо $exp\left(\beta_j\right) > 1$, зростання x_j збільшує ймовірність швидкого відтоку; якщо менше одиниці — сповільнює його.

Метод Кокса особливо доречний у задачі прогнозування часу відтоку клієнтів, бо поєднує в собі гнучкість нелінійних ефектів у вигляді експоненти від лінійної комбінації та не потребує попереднього визначення форми часової компоненти. Градієнтно-методичне оптимізаційне рішення та

можливість включення регуляризації роблять його зручним для розв’язання на реальних даних великого розміру.

2.3. Метрики оцінки якості моделей

У процесі порівняння різних алгоритмів класифікації та виживального аналізу вкрай важливо обирати такі критерії оцінки, які коректно відображають як загальну точність передбачень, так і здатність моделі враховувати баланс класів та часову компоненту.

Класичні метрики бінарної класифікації. Перш за все для моделей, що передбачають відтік клієнта як подію “так/ні”, використовуються похідні від матриці сплутувань: точність, точність позитивних передбачень, повнота та їх гармонійне середнє — F_1-міра. Нехай TP, FP, TN, FN — кількість істинно позитивних, хибно позитивних, істинно негативних та хибно негативних передбачень відповідно. Тоді

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \tag{2.3.1}$$

$$Precision = \frac{TP}{TP+FP}, \tag{2.3.2}$$

$$Recall = \frac{TP}{TP+FN}, \tag{2.3.3}$$

$$F_1 = 2 \cdot \frac{Precision \times Recall}{Precision+Recall}. \tag{2.3.4}$$

Окрім них, важливо враховувати метрику ROC AUC — площу під кривою, що є інтегральним показником здатності моделі розділяти класи без прив’язки до конкретного порогу.

Для оцінки моделей виживального аналізу застосовують конкордантний індекс, який узагальнює ідею ROC AUC на випадок часових даних із

цензуруванням. Нехай для двох клієнтів i та j відомі їхні реальні часи події T_i, T_j та відповідні предсказані ризики r_i, r_j. Пара вважається конкордантною, якщо той, хто насправді пішов раніше ($T_i < T_j$), має вищий передбачений ризик ($r_i > r_j$). Тоді

$$C - index = \frac{\#\{\text{конкордантні пари}\}}{\#\{\text{усі порівнянні пари}\}}. \quad (2.3.5)$$

Значення C-index близько до 1 свідчить про високу узгодженість моделі з фактичним порядком подій, а 0.5 відповідає випадковому передбаченню.

Баланс класів і середні показники. Оскільки в реальних даних, зазвичай, кількість клієнтів, що зазнали відтоку, значно менша за кількість клієнтів, які залишились, , доцільно доповнити оцінку середніми по класах для precision, recall та F_1, або використовувати PR AUC, що мало чутливий до дисбалансу.

Таким чином, при порівнянні моделей ми одночасно аналізуємо: Accuracy, Precision, Recall, F_1, ROC AUC, PR AUC і для виживального аналізу C-index.

Цей набір метрик дозволяє всебічно оцінити як бінарні рішення про відтік, так і якість прогнозів часу до відтоку.

2.4 Методика крос-валідації

Крос-валідація є процедурою оцінювання здатності моделі узагальнювати знання на нові дані через циклічне розділення доступного набору спостережень на тренувальні та тестові підмножини. У контексті нашої задачі, що включає одночасно бінарну класифікацію відтоку та прогноз часу до події, застосовуються два окремих підходи.

Нехай $D = \{(x_i, y_i)\}_{i=1}^{n}$ — весь навчальний датасет. Ми розбиваємо D на K непересічних блоків $D_1, D_2, \ldots, D_K$ приблизно рівного розміру. На кожній

ітерації $k = 1 \ldots K$ модель навчається на $\bigcup_{j \neq k} D_j$ і тестується на D_k. Остаточна оцінка моделі обчислюється як середнє значення обраної метрики по всіх K ітераціях:

$$M^{-} = \frac{1}{K} \sum_{k=1}^{K} M\left(\hat{f}_{-k}, D_k \right),$$

(2.4.1)

де $\hat{f}_{-k}$ — модель, навчена без k-го блоку. Важливо застосувати стратифікацію розбиття за мітками y, щоб кожен fold зберігав пропорції відтоку й «залишених» клієнтів, особливо коли класи є незбалансованими.

У виживальному аналізі ключовим є правильне поводження з цензуруванням і часовою інформацією. Нехай крім x_i і події $y_i \in \{0, 1\}$ ми маємо тривалість t_i. Тоді для кожного фолду проводять аналогічне розбиття даних на тренувальні та валідаційні підмножини, переконавшись, що в обох підмножинах присутні й відтеклі, й цензуровані спостереження. На кожній ітерації моделі Кокса навчають на тренувальній підмножині $\{(x_i, t_i, y_i) | i \notin I_k\}$ та обчислюють Concordance-index на валідаційному наборі $\{(x_i, t_i, y_i) | i \in I_k\}$. Остаточна метрика обчислюється як середнє C-index по всіх folds:

$$C^{-} = \frac{1}{K} \sum_{k=1}^{K} C - index\left(\hat{\beta}_{-k}, \{(x_i, t_i, y_i)\}_{i \in I_k} \right).$$

(2.4.2)

Процес крос-валідації включає механізм стратифікації, який у випадку виживального аналізу охоплює не лише підтримку пропорцій позитивних та негативних випадків, а й збереження подібного розподілу тривалостей перебування клієнтів у кожному з п’яти блоків. Це гарантує, що на тестових підмножинах адекватно представлені як швидкі, так і повільні відтоки, мінімізуючи упередженість оцінки, що може виникнути через нерівномірність часових інтервалів.

Вибір п’яти фолдів є збалансованим компромісом: достатня кількість ітерацій для стабільної оцінки моделі та при цьому достатній обсяг тренувальних даних у кожному фолді, щоб уникнути надто малих навчальних підмножин. Завдяки цьому середнє значення метрик по всіх п’яти розбиттях відображає реальну узагальнюваність алгоритму на нові дані.

Для бінарних класифікаторів застосовується традиційна стратифікована п’ятифолдна схема, яка перемішує дані перед формуванням блоків та фіксує стан генератора випадкових чисел для відтворюваності результатів. У випадку моделі Кокса розроблено спеціалізовану процедуру крос-валідації: на кожному кроці тренувальні індекси відокремлюються від валідаційних, модель навчається на даних разом із тривалістю та інформацією про відтік, а потім на валідаційних даних обчислюється конкордансний індекс. Такий підхід дозволяє коректно оцінювати як якість розпізнавання події відтоку, так і точність прогнозу часової компоненти.

2.5 Підхід до порівняння результатів моделей

Порівняння ефективності різних алгоритмів здійснюється на основі чотирьох взаємодоповнювальних критеріїв.

По-перше, для бінарних класифікаторів передбачається оцінка середніх значень Accuracy, ROC-AUC та F_1-міри, отриманих у ході п’ятифолдної крос-валідації. ROC-AUC дозволяє кількісно виміряти здатність моделі розрізняти класи за різних порогових значень, а F_1-міра забезпечує баланс між точністю та повнотою в умовах нерівномірного розподілу цільових класів.

По-друге, у разі виживального аналізу модель Кокса оцінюється через конкордансний індекс, що відображає ступінь узгодженості порядку прогнозованих ризиків із фактичними тривалостями до події. C-index є інваріантним до масштабу оцінок ризику та чутливим до правильної ієрархії відносних небезпек.

По-третє, для всіх моделей заплановано аналіз дисперсії обчислених показників між окремими фолдами. Низька варітивність метрик вказує на збалансованість даних у розбиттях і демонструє стійкість алгоритму до випадкових флуктуацій вибірки.

По-четверте, важливою складовою порівняння є обчислювальна ефективність: очікується вимірювання часу тренування й передбачення, а також оцінка використання оперативної пам’яті. При близьких значеннях якості моделі на практиці можуть надаватися перевага алгоритмам із меншими ресурсними затратами або тим, що забезпечують прозорі інтерпретаційні властивості.

У сукупності наведені критерії формують комплексну основу для вибору оптимального підходу до прогнозування клієнтського відтоку.

2.6 Висновки до розділу

У другому розділі виконано ґрунтовний огляд теоретичних основ застосованих методів машинного навчання для задачі прогнозування клієнтського відтоку. Ми формалізували задачу бінарної класифікації та виживального аналізу, окреслили математичні підходи для кожного класу моделей і навели ключові характеристики Decision Tree, Logistic Regression, Random Forest, SVM, XGBoost та моделі пропорційних ризиків Кокса.

Було обґрунтовано вибір метрик якості: для класифікаторів — Accuracy, ROC-AUC та F_1-міра, що дозволяють оцінити баланс між точністю і повнотою, а для моделі Кокса — C-index, який коректно відображає ранжування клієнтів за ризиком відтоку з урахуванням часової складової. Окрему увагу приділено опису методики п’ятифолдної крос-валідації із стратифікацією як класових ознак, так і розподілу тривалостей, що забезпечує збалансовану та відтворювану оцінку продуктивності алгоритмів.

Крім того, визначено підхід до порівняння моделей за чотирма групами критеріїв — якість класифікації, узгодженість ранжування ризиків, стабільність результатів та обчислювальна ефективність. Такий

багатовимірний аналіз створює надійну основу для вибору оптимальної моделі у практичній частині дослідження.

Загалом, представлений теоретичний апарат і критерії оцінювання встановлюють чіткі правила побудови та порівняння алгоритмів прогнозування відтоку клієнтів. У третьому розділі буде реалізовано описану методологію, проведено експериментальні випробування та зіставлено практичні результати із теоретичними очікуваннями.

РОЗДІЛ 3.ПРОГРАМНА РЕАЛІЗАЦІЯ МОДЕЛІ

3.1 Функціональні та нефункціональні вимоги

У будь-якому складному інформаційному проєкті чітке розмежування вимог є ключовим чинником успішності: функціональні вимоги визначають набір операцій та сервісів, які система повинна надавати, а нефункціональні — якісні властивості, що забезпечують надійну та ефективну експлуатацію рішення. У контексті прогнозування відтоку клієнтів банківської установи формулювання цих вимог має ґрунтуватися на вимогах предметної області і стандартних практиках інженерії програмного забезпечення.

3.1.1 Функціональні вимоги

Першочерговою задачею системи є забезпечення повного циклу роботи з даними клієнтів. Це включає модулі імпорту як сирих CSV-файлів, так і інформації безпосередньо з PostgreSQL-бази даних, механізми очищення й попередньої обробки та формування стандартизованого pipeline, придатного для тренування моделей. Застосування sklearn-конвеєрів гарантує відтворюваність та прозорість обробки даних.

Наступним рівнем є модульна побудова та навчання шести алгоритмів: від традиційних до ансамблевих, SVM та виживальної аналізу. Кожний обраний метод повинен мати окремий компонент, який зберігає очікувані гіперпараметри та навчальні вагони у вигляді файлів joblib, забезпечуючи швидке відновлення без повторного тренування.

Функція порівняння продуктивності моделей повинна автоматично виконувати крос-валідацію з обчисленням метрик Accuracy, ROC-AUC, F_1-міри для класифікаторів і C-index для моделі пропорційних ризиків, збирати результати у загальний звіт і відображати їх як у табличному, так і в графічному вигляді. Для інтеграції із зовнішніми системами та користувачами реалізується REST API на базі FastAPI, що надає кінцеві точки для прогнозування ймовірності відтоку, як у формі бінарної класифікації, так і з

часовими траєкторіями ризику; окремо – CRUD-операції для керування даними клієнтів.

3.1.2 Нефункціональні вимоги

Продуктивність системи має забезпечувати час відгуку API не більше ніж 200–300 мс для запитів бінарної класифікації та до 500 мс для виживального прогнозу, що вимагає оптимізації передобробки даних і кешування результатів pipeline. Горизонтальне масштабування сервісів, виконане через Docker-контейнери, гарантує можливість швидкого збільшення обчислювальних потужностей при зростанні навантаження.

Безпека обробки даних забезпечується застосуванням захищених SSL-з’єднань із PostgreSQL, зберіганням облікових даних виключно в змінних середовища та валідацією всіх вхідних повідомлень через Pydantic-схеми. Такий підхід виключає ризик SQL-ін’єкцій та некоректних даних, що критично важливо для обробки конфіденційної клієнтської інформації.

Наявність автоматично генерованої OpenAPI/Swagger-документації забезпечує високу інтегрованість та зрозумілість інтерфейсів для розробників та аналітиків. Портативність проекту досягається завдяки контейнеризації, що дозволяє розгортати його в будь-якому середовищі без зміни конфігурації, а також полегшує підтримку і оновлення компонентів.

3.2 Опис використаного середовища розробки

У середовищі розробки основною інтегрованою середовищем було обране IDE PyCharm, яке забезпечує зручне автодоповнення, відладку та навігацію по коду. Його підтримка віртуальних середовищ і відстеження змін у файлах сприяє структурованому та прозорому процесу розробки.

Для експериментальної побудови та візуального дослідження моделей застосовувався Jupyter Notebook. Цей інструмент дозволяє поєднувати виконуваний код, описові тексти й графіки в єдиному документі, що значно спрощує етап прототипування та аналізу проміжних результатів.

Мову програмування та середовище виконання становить Python 3.10, оскільки ця версія має повноцінну підтримку сучасних бібліотек і синтаксичних конструкцій. Використання останніх релізів гарантує сумісність із науковими пакетами й забезпечує стабільну роботу в продуктивному середовищі.

Для машинного навчання та конвеєрного оброблення даних обрано бібліотеки scikit-learn 1.2.x і NumPy 1.24.x. Вони забезпечують уніфікований інтерфейс для побудови пайплайнів, виконання трансформацій і розрахунку основних статистичних операцій із високою продуктивністю.

Розгортання REST-API здійснено на базі FastAPI 0.95.x разом із сервером Uvicorn, що забезпечують асинхронну обробку запитів і автоматичне генерування докладної OpenAPI-документації. Pydantic 2.x використовується для суворої валідації вхідних та вихідних даних.

Дані зберігаються в PostgreSQL 15, доступ до якої організовано через SQLAlchemy 2.x із драйвером psycopg2-binary. Такий підхід гарантує наскрізну підтримку ACID-транзакцій і можливість роботи з ORM-об’єктами у декларативному стилі.

Контейнеризація виконана за допомогою Docker 19.03+ і Docker Compose 3.8, що дозволяє запускати сервіси “api” та “db” в ізольованому, але пов’язаному середовищі. Завдяки цьому досягається портативність рішення та простота масштабування.

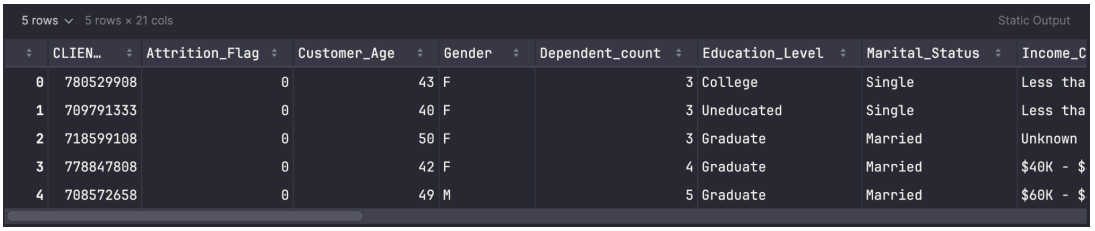
3.3 Підготовка та обробка даних

У цьому пункті здійснюється комплексний аналіз початкового набору спостережень з метою забезпечення коректності, повноти та однорідності вхідних даних перед навчанням моделей.

3.3.1. Огляд і завантаження даних

На початковій стадії підготовки даних здійснюється їх пряме завантаження з CSV-файлу та попередній огляд структури. З Рисунку 3.3.1.1

бачимо перші п’ять рядків датасету, що включає поля CLIENTNUM, Attrition_Flag, демографічні ознаки та показники транзакцій клієнтів.



5 rows · 5 rows × 21 cols · Static Output

	CLIEN…	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_C
0	780529908	0	43	F	3	College	Single	Less tha
1	709791333	0	40	F	3	Uneducated	Single	Less tha
2	718599108	0	50	F	3	Graduate	Married	Unknown
3	778847808	0	42	F	4	Graduate	Married	$40K - $
4	708572658	0	49	M	5	Graduate	Married	$60K - $

Рисунок 3.3.1.1 — результат виклику head()

Далі виконується команда info(), завдяки якій встановлено загальну кількість записів та типи змінних: категоріальні поля представлені як object, числові — як int64 чи float64. З Рисунку 3.3.1.2 видно, що пропуски трапляються лише у незначній кількості ознак, до 0.5%, що дозволяє запланувати імпутацію без критичної втрати даних.

```
RangeIndex: 8101 entries, 0 to 8100
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   CLIENTNUM                 8101 non-null   int64
 1   Attrition_Flag            8101 non-null   int64
 2   Customer_Age              8101 non-null   int64
 3   Gender                    8101 non-null   object
 4   Dependent_count           8101 non-null   int64
 5   Education_Level           8101 non-null   object
 6   Marital_Status            8101 non-null   object
 7   Income_Category           8101 non-null   object
 8   Card_Category             8101 non-null   object
 9   Months_on_book            8101 non-null   int64
 10  Total_Relationship_Count  8101 non-null   int64
 11  Months_Inactive_12_mon    8101 non-null   int64
 12  Contacts_Count_12_mon     8101 non-null   int64
 13  Credit_Limit              8101 non-null   float64
 14  Total_Revolving_Bal       8101 non-null   int64
 15  Avg_Open_To_Buy           8101 non-null   float64
 16  Total_Amt_Chng_Q4_Q1      8101 non-null   float64
 17  Total_Trans_Amt           8101 non-null   int64
 18  Total_Trans_Ct            8101 non-null   int64
 19  Total_Ct_Chng_Q4_Q1       8101 non-null   float64
 20  Avg_Utilization_Ratio     8101 non-null   float64
```

Рисунок 3.3.1.2 — результат виклику info().

Статистичний опис числових ознак наведено на Рисунку 3.3.1.3. З нього випливає, що такі змінні, як Credit_Limit та Total_Trans_Amt, мають

широкі розмахі та виражену правосторонню скошеність. Це свідчить про необхідність подальшої трансформації розподілів, логарифмування або стандартизація, перед використанням у моделях.

	count	mean	std	min	25%	50%	75%	max
CLIENTNUM	8101.0	7.390230e+08	3.681928e+07	708082083.0	7.130432e+08	7.178661e+08	7.730736e+08	8.283431e+08
Attrition_Flag	8101.0	1.630663e-01	3.694489e-01	0.0	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
Customer_Age	8101.0	4.626947e+01	8.039553e+00	26.0	4.100000e+01	4.600000e+01	5.200000e+01	7.300000e+01
Dependent_count	8101.0	2.346377e+00	1.301854e+00	0.0	1.000000e+00	2.000000e+00	3.000000e+00	5.000000e+00
Months_on_book	8101.0	3.590001e+01	7.962032e+00	13.0	3.200000e+01	3.600000e+01	4.000000e+01	5.600000e+01
Total_Relationship_Co…	8101.0	3.805086e+00	1.554493e+00	1.0	3.000000e+00	4.000000e+00	5.000000e+00	6.000000e+00
Months_Inactive_12_mon	8101.0	2.344032e+00	1.014231e+00	0.0	2.000000e+00	2.000000e+00	3.000000e+00	6.000000e+00
Contacts_Count_12_mon	8101.0	2.455499e+00	1.113509e+00	0.0	2.000000e+00	2.000000e+00	3.000000e+00	6.000000e+00
Credit_Limit	8101.0	8.637835e+03	9.085735e+03	1438.3	2.551000e+03	4.559000e+03	1.102500e+04	3.451600e+04
Total_Revolving_Bal	8101.0	1.158604e+03	8.189882e+02	0.0	2.430000e+02	1.272000e+03	1.788000e+03	2.517000e+03
Avg_Open_To_Buy	8101.0	7.479231e+03	9.087856e+03	15.0	1.351000e+03	3.507000e+03	9.816000e+03	3.451600e+04
Total_Amt_Chng_Q4_Q1	8101.0	7.591803e-01	2.186939e-01	0.0	6.310000e-01	7.360000e-01	8.580000e-01	3.397000e+00
Total_Trans_Amt	8101.0	4.421609e+03	3.414411e+03	510.0	2.153000e+03	3.904000e+03	4.753000e+03	1.774400e+04
Total_Trans_Ct	8101.0	6.491433e+01	2.354105e+01	10.0	4.500000e+01	6.700000e+01	8.100000e+01	1.390000e+02
Total_Ct_Chng_Q4_Q1	8101.0	7.118325e-01	2.381180e-01	0.0	5.820000e-01	7.020000e-01	8.180000e-01	3.571000e+00
Avg_Utilization_Ratio	8101.0	2.727729e-01	2.746314e-01	0.0	1.700000e-02	1.750000e-01	5.000000e-01	9.940000e-01

Рисунок 3.3.1.3 — таблиця describe().

Для візуальної оцінки розподілу ключових показників будуємо гістограми (Рисунок 3.3.1.4). З рисунку чітко видно концентрацію клієнтів у віковому діапазоні 30–50 років, наявність викидів у полях Total_Trans_Ct та Months_on_book. Ці спостереження є підґрунтям для подальшого очищення та корекції аномальних значень.

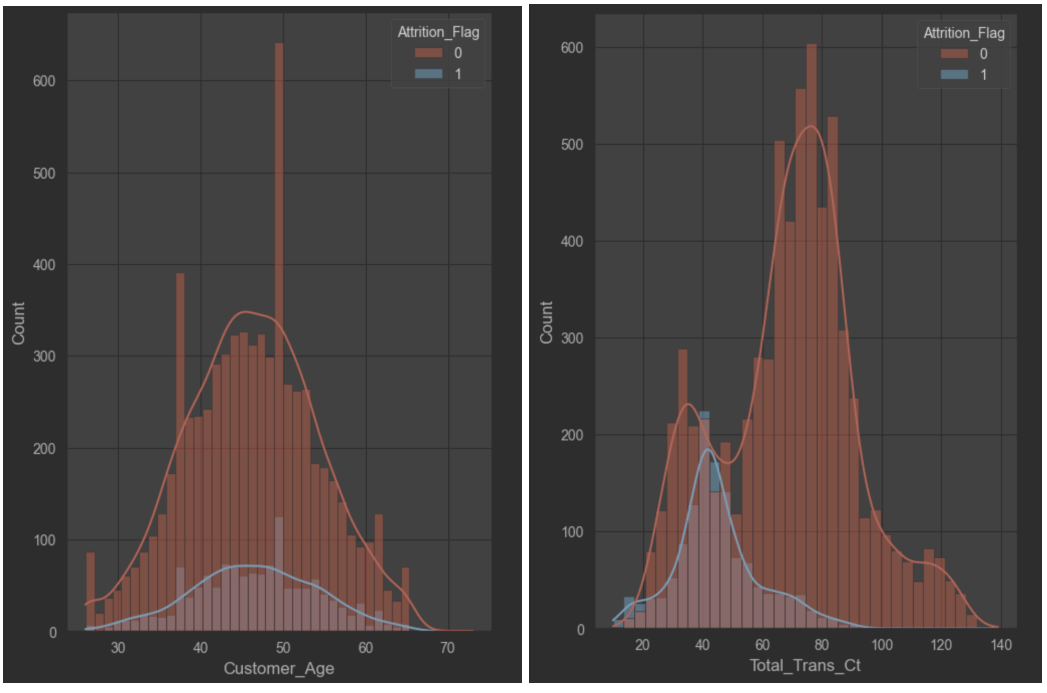


Рисунок 3.3.1.4 — гістограми Customer_Age та Total_Trans_Ct

3.3.2. Аналіз пропущених значень та аномалій

Після огляду основних характеристик датасету слід оцінити розподіл пропущених значень, оскільки навіть незначний їхній відсоток може спотворити результати побудованих моделей. З Рисунку 3.3.2.1 видно, що поля Total_Revolving_Bal та Avg_Open_To_Buy мають дуже маленьку частку пропусків, у той час як інші атрибути повністю заповнені. Така картина дозволяє застосувати метод імпутації на основі медіани або k-nearest neighbors без значних викривлень розподілу даних.

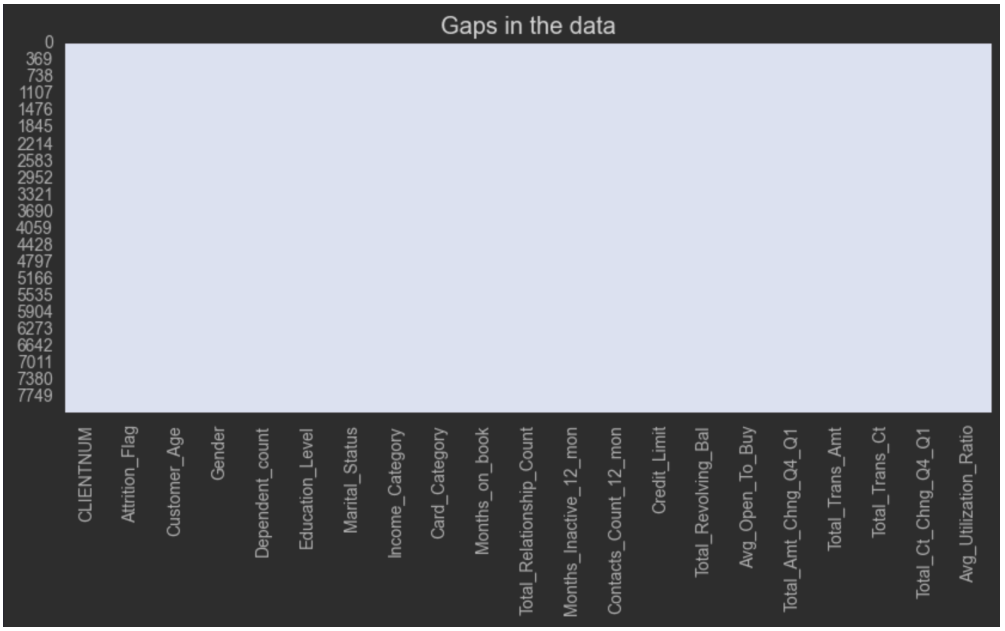


Рисунок 3.3.2.1 — теплова карта пропусків

Паралельно було проведено виявлення аномальних спостережень за допомогою box-plot аналізу. З Рисунку 3.3.2.2 спостерігаються виразні викиди у показниках Credit_Limit та Total_Trans_Amt, які виходять за межі інтерквартильного діапазону більш ніж у півтора раза. Такі аномалії можуть бути обґрунтованими, проте для забезпечення стабільності алгоритмів класифікації та моделі виживання передбачено дві стратегії: або обмежити значення на рівні 99-го перцентиля, або видалити крайні записи за порогом 1.5 IQR.

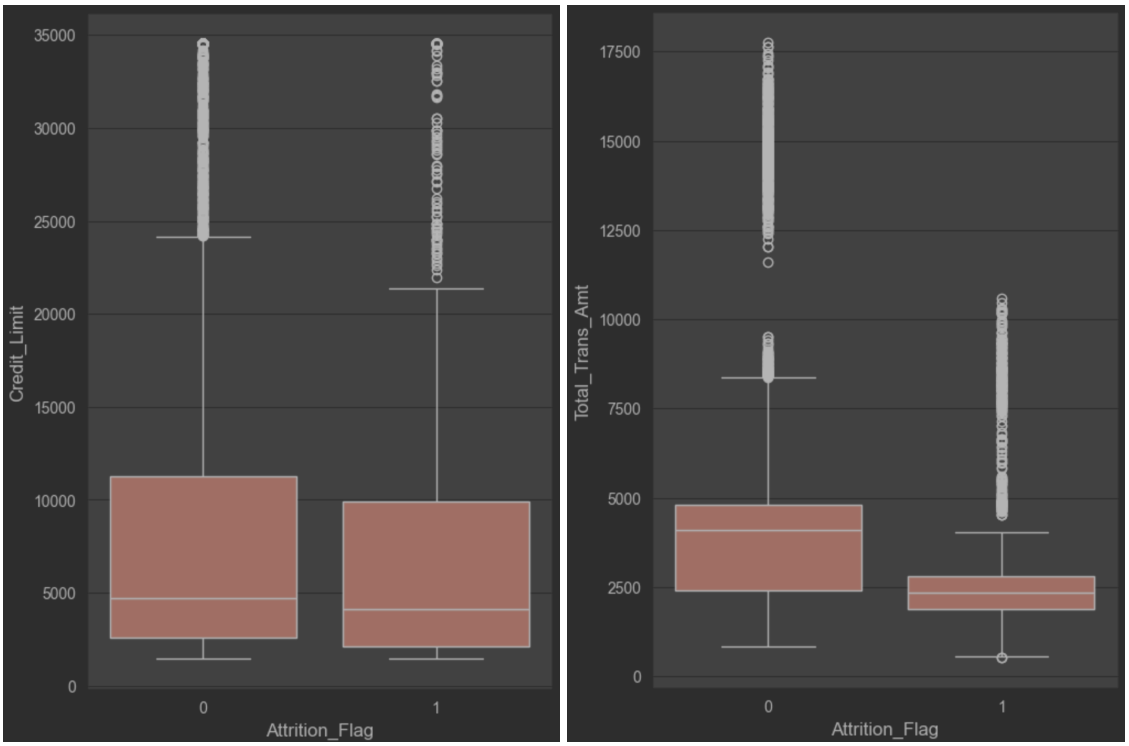


Рисунок 3.3.2.2 — box-plot для Credit_Limit та Total_Trans_Amt.

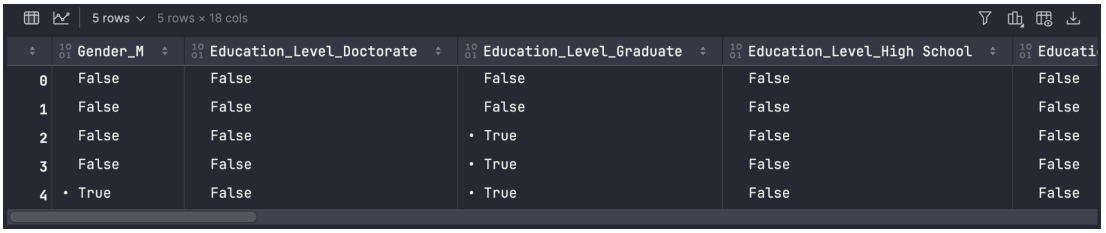
Таким чином, детальний аналіз пропущених даних і аномалій забезпечує обґрунтовану основу для наступних кроків очищення та стабілізації розподілів, що критично важливо для коректного навчання як алгоритмів класифікації, так і регресії Кокса.

3.3.3. Попередня обробка ознак

Перед безпосереднім навчанням моделей був виконаний комплекс операцій із приведення вихідних колонок до формату, зручного для математичних алгоритмів. На початковому етапі було здійснено розділення змінних на числові та категоріальні, що дало змогу застосувати спеціалізовані перетворення для кожного типу.

По-перше, категоріальні ознаки, зокрема Gender, Education_Level, Marital_Status, Income_Category, Card_Category, були перетворені за допомогою одноразового кодування. Такий підхід гарантує відсутність ієрархії між категоріями та дозволяє кожній з них формувати окремий бінарний індикатор. Після кодування кількість ознак зросла до уніфікованого

розміру, що дозволило уникнути втрати інформації про окремі категорії клієнтів.



5 rows × 18 cols

	Gender_M	Education_Level_Doctorate	Education_Level_Graduate	Education_Level_High School	Educati
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	True	False	False
3	False	False	True	False	False
4	True	False	True	False	False

Рисунок 3.3.3.1 — приклад розбиття категоріальної ознаки на бінарні стовпці

По-друге, числові ознаки, зокрема такі, що вимірюються в різних одиницях, було стандартизовано за Z-правилом, відніманням середнього та діленням на стандартне відхилення. Це сприяло виконанню припущень багатьох алгоритмів про однорідність масштабу ознак.

По-третє, було проведено відсіювання рідкісних категорій у тих стовпцях, де кількість унікальних значень перевищувала 10. Категорії, що траплялись у менше ніж 1% записів, об’єднані в одну групу Other, аби зменшити розрідженість one-hot матриці та уникнути перенавчання.

Обчислені проміжні ознаки, такі як відношення Avg_Open_To_Buy до Credit_Limit та нормовані зміни транзакцій, дозволили моделю захопити динаміку активності клієнта.

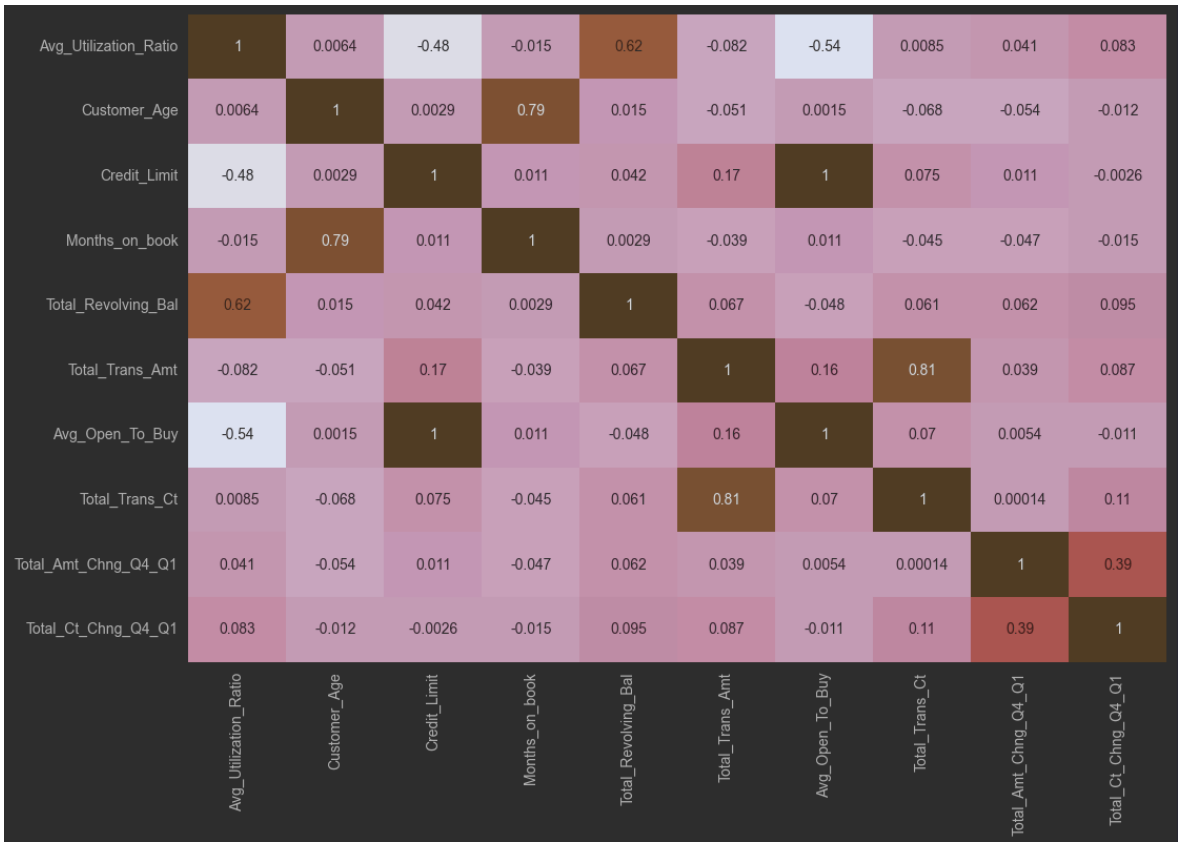


Рисунок 3.3.3.2 — кореляційна матриця інженерних ознак із цільовою змінною

Таким чином, попередня обробка ознак забезпечила уніфікацію даних, зменшення впливу викидів та посилення генералізаційних властивостей моделей.

3.3.4. Аналіз кореляційних зв’язків та взаємодій ознак

У ході детального дослідження міжзв’язків між кількісними ознаками та ймовірністю відтоку клієнтів було виявлено суттєві патерни, які можуть бути використані для покращення якості прогнозної моделі. По–перше, матриця розсіянь (Рисунок 3.3.4.1) демонструє загальну структуру розподілу точок для пари ознак і підкреслює, що класи клієнтів помітно відрізняються за низкою характеристик. Наприклад, скупчення червоних точок у верхньому правому куті діаграми Total_Trans_Amt vs Total_Trans_Ct свідчить про більший обсяг та частоту транзакцій у активних клієнтів, тоді як сині точки розташовані ближче до осі початку координат. Таке розділення класів

створює основу для подальшого відбору ознак і обґрунтування їх включення до моделі.

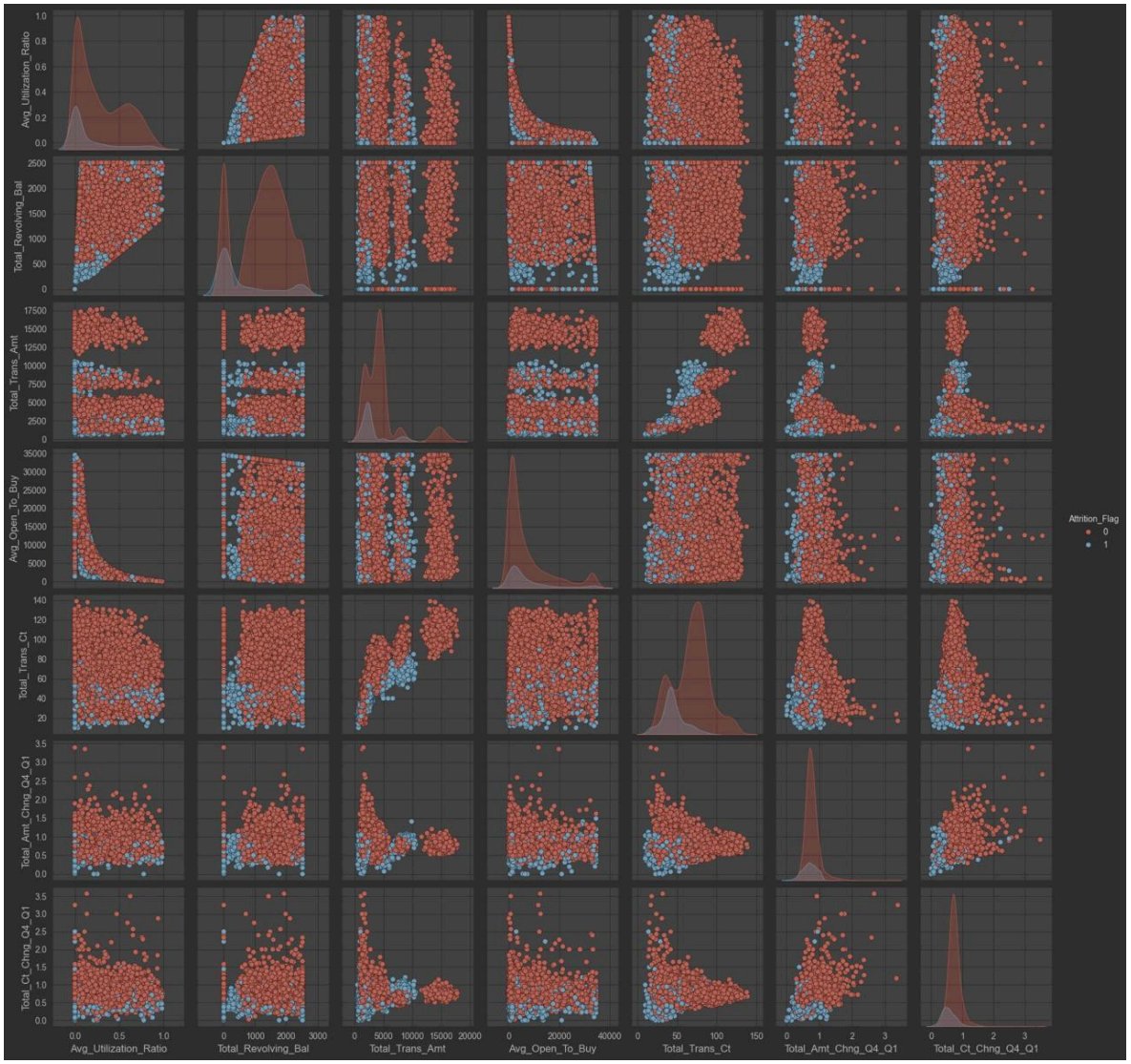


Рисунок 3.3.4.1 — матриця розсіяння ознак із розподілом за Attrition_Flag

Регресійний аналіз окремих пар ознак підтверджує гіпотезу про те, що для клієнтів з високим рівнем транзакційних активностей ймовірність відтоку суттєво знижується. На Рисунку 3.3.4.2 побудовано лінію найменших квадратів для груп Existing та Attrited, що дозволяє побачити різні нахили регресійних ліній: для активних клієнтів залежність Total_Trans_Amt від Total_Trans_Ct є більш крутою, а для тих, хто відтікає, – пологішою. Така відмінність в інтеракції між двома ознаками свідчить про те, що комбінація

цих ознак є інформативною для побудови класифікатора і має бути збережена на стадії інженерії ознак.

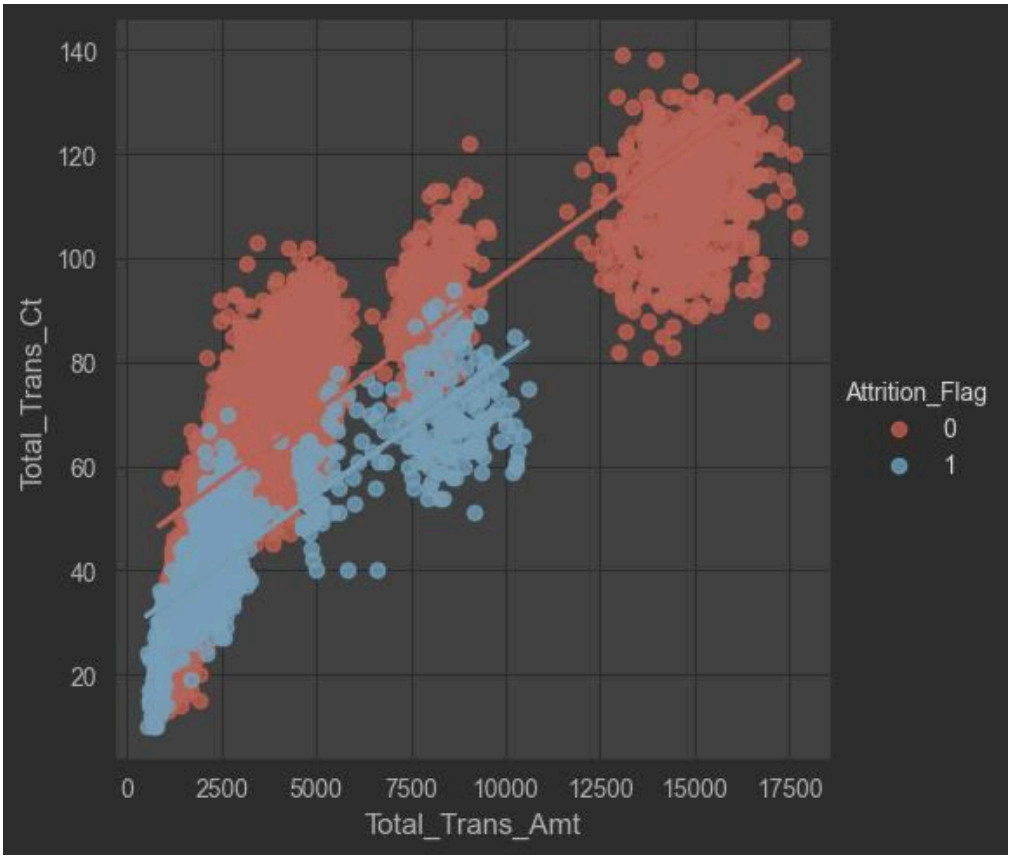


Рисунок 3.3.4.2 — Взаємозв’язок Total_Trans_Amt та Total_Trans_Ct із регресійними лініями

Поглиблений аналіз співвідношення середнього відкритого кредитного ліміту до коефіцієнта його використання (Рисунок 3.3.4.3) виявляє, що клієнти з високим коефіцієнтом використання і водночас помірним чи низьким AVG_Open_To_Buy частіше відходять. Це узгоджується з теоретичними уявленнями про фінансовий стрес, коли високе навантаження на доступні кредитні ресурси є предиктором нездатності обслуговувати рахунок і потенційного відтоку. З огляду на це, у подальшій стадії побудови моделі доцільно вводити не лише початкові величини ознак, але й їхні взаємодії чи відношення як нові композиційні ознаки.

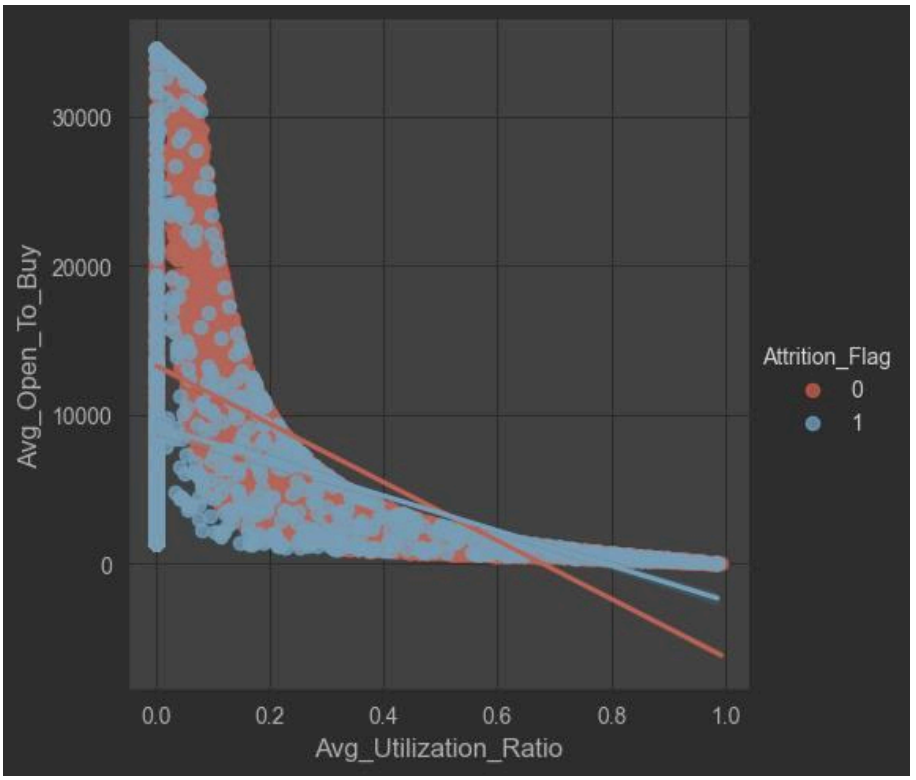


Рисунок 3.3.4.3 — Взаємозв’язок Avg_Utilization_Ratio та Avg_Open_To_Buy із регресійними лініями

Для детального відтворення цього аналізу та генерування наведених графіків див. Додаток А.2, де наведено відповідні фрагменти коду та параметри відображення.

3.3.5. Відбір ознак

Процедура відбору ознак є завершальною стадією препроцесингу, під час якої з початкового масиву даних виділяються ті змінні, що несуть найбільший інформаційний вклад у подальші дослідження. У нашому випадку застосовано два взаємодоповнювані підходи без залучення алгоритмів прогнозування.

Для кожної числової змінної було обчислено коефіцієнт кореляції Пірсона відносно бінарної цільової змінної Attrition_Flag. Результати подано у вигляді барплоту (Рисунок 3.3.5.1), де від’ємні та додатні значення відображають пряму або обернену залежність з ризиком відтоку. До переліку найбільш корельованих ознак увійшли Months_on_book, Total_Trans_Ct,

Total_Trans_Amt та Total_Revolving_Bal, що узгоджується з попереднім описом розподілів (пункти 3.3.2–3.3.4).

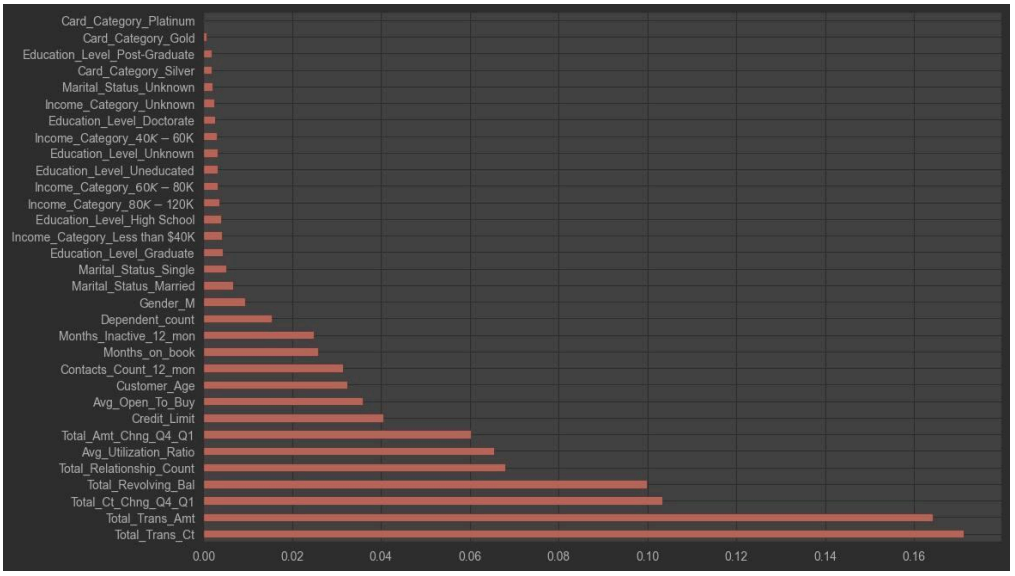


Рисунок 3.3.5.1 — кореляція числових ознак із Attrition_Flag

Для категоріальних змінних використано метод обчислення статистики взаємної інформації щодо цільової ознаки. Отримані значення було впорядковано й візуалізовано на Рисунку 3.3.5.2. Яскраво виражену інформативність продемонстрували Gender, Marital_Status та Education_Level, тоді як частина менш значущих категорій було вилучено.

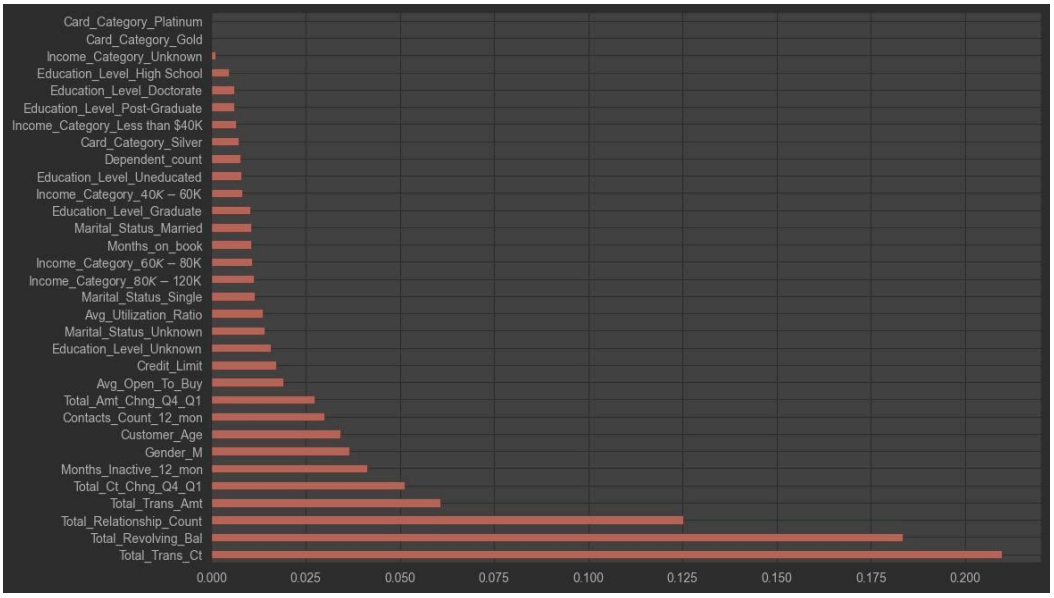


Рисунок 3.3.5.2 — взаємна інформація категоріальних змінних щодо Attrition_Flag

На основі цих двох кроків сформовано підмножину ознак, що включає 15 змінних з найвищими показниками кореляції та інформаційної значущості. Код процедури та перелік обраних ознак наведено в Додатку А.3. Такий узгоджений відбір дозволяє зменшити розмірність простору даних, мінімізувати шум та підготувати компактний, але змістовний набір ознак для побудови моделі у наступному розділі.

3.4 Реалізація моделей класифікації

3.4.1 MyDecisionTreeClassifier

Метод дерев рішень реалізований у класі MyDecisionTreeClassifier (Додаток А.4), який навчається шляхом рекурсивного поділу навчальної множини на підмножини за оптимальним порогом кожної ознаки. На кожному кроці алгоритм обчислює критерій якості розбиття для всіх можливих пар «ознака – поріг» і обирає ті, що забезпечують максимум приросту інформації чи мінімум нечистоти. Розгортання дерева продовжується доти, доки не буде досягнуто одного з критеріїв зупинки: заданої максимальної глибини, мінімальної кількості зразків у вузлі або відсутності покращення метрики.

Побудоване дерево зберігається у вигляді вкладених кортежів, де кожен внутрішній вузол містить індекс ознаки, поріг поділу, посилання на ліву й праву підгілки, а листок — мітку прогнозованого класу. Класифікація нового зразка здійснюється шляхом послідовного порівняння значення ознаки з порогом у вузлах і вибору відповідної гілки аж до досягнення листка.

Клас MyDecisionTreeClassifier підтримує налаштування ключових гіперпараметрів: максимальної глибини дерева — max_depth, мінімальної кількості зразків, необхідних для поділу вузла — min_samples_split, а також критерію розбиття — criterion='gini' або 'entropy'. Вибір оптимальних значень для цих параметрів здійснюється за допомогою крос-валідації (Додаток А.10), що дозволяє досягти балансу між узагальнюючою здатністю моделі та ризиком перенавчання.

Завдяки інтерпретованості результатів і можливості оцінити важливість ознак за допомогою показників зменшення нечистоти, метод дерев рішень слугує як потужний інструмент для попереднього аналізу даних і формування базових класифікаційних моделей.

3.4.2 MyLogisticRegression

Реалізація класу MyLogisticRegression (Додаток А.5) починається з ініціалізації базових параметрів моделі: початкового вектору ваг та зсуву, а також налаштувань алгоритму оптимізації, кількість ітерацій, критерій зупинки, тип регуляризації і коефіцієнт регуляризації. У методі fit спочатку здійснюється стандартизація вхідних числових ознак та перетворення категоріальних змінних у one-hot представлення; для цього використовується внутрішній препроцесор, винесений у окремий підклас.

Основний цикл навчання виконує ітеративне оновлення параметрів методом стохастичного градієнтного спуску, причому на кожному кроці обчислюється вектор похибки між отриманими прогнозами та істинними мітками, після чого параметри моделі коригуються з урахуванням додаткового регуляризаційного члена. Для прискорення обчислень всі операції з векторами виконуються через векторизовані виклики NumPy, що забезпечує високу ефективність навіть на великих вибірках.

Метод predict_proba повертає двовимірний масив розмірності $n \times 2$, де кожний рядок містить ймовірності віднесення відповідного зразка до класів «0» та «1». Для отримання остаточного класу реалізовано метод predict, який порівнює ймовірність позитивного класу з порогом, за замовчуванням 0.5, і повертає відповідну мітку.

Клас підтримує серіалізацію через інтерфейс joblib, що дозволяє зберегти навчену модель разом з налаштованим препроцесором і потім завантажити її без необхідності відтворювати послідовність обробки даних. Таким чином, MyLogisticRegression забезпечує повний конвеєр «від сирих даних до передбачень» у єдиному об’єкті.

3.4.3 MyRandomForestClassifier

Реалізація класу MyRandomForestClassifier (Додаток А.6) передбачає побудову ансамблю з M незалежних дерев рішень із випадковою вибіркою ознак та спостережень на кожному кроці. У конструкторі задаються ключові гіперпараметри: число дерев, глибина кожного дерева, мінімальна кількість зразків у листі та число ознак, що випадково відбираються для розв'язання кожного вузла.

Метод fit починається зі створення бутстреп-зразків з вихідного тренувального набору X, y. Для кожного дерева генерується підмножина спостережень із поверненням та випадковий підбір k ознак для побудови розбиття в кожному вузлі. Будова кожного дерева виконується рекурсивно за алгоритмом побудови дерева рішень із критерієм індексу Джині або ентропії як мірою негомогенності. Завдяки вибірковості ознак усередині вузлів забезпечується зменшення кореляції серед дерев, що істотно підвищує стійкість ансамблю до перенавчання.

Після тренування всіх дерев набір моделей об'єднується в єдиний об'єкт: метод predict для кожного вхідного зразка послідовно викликає предикцію всіх дерев і повертає мітку, що отримала більшість голосів. Аналогічно, метод predict_proba агрегує індивідуальні ймовірності відносячи відсоток позитивних голосів до друку розподілу по класах.

Усередині реалізації використано багатопроцесорність для одночасної побудови дерев, а також векторизовані операції для швидкої обробки матриць розбиття. Завдяки цьому навіть при великій кількості дерев довжина навчання залишається прийнятною. Для збереження навченого ансамблю реалізовано підтримку серіалізації через joblib, що дозволяє миттєво завантажити вже тренований конвеєр без повторного тренування.

3.4.4 MySVMClassifier

Реалізація класса MySVMClassifier (Додаток А.7) базується на методі опорних векторів із симплексним розв'язком оптимізаційної задачі. У

конструкторі задаються основні гіперпараметри: параметр регуляризації C, тип ядра та параметр ядра γ. Метод fit спочатку нормалізує вхідні ознаки, застосовуючи централізацію та масштабування до одиничної дисперсії, після чого будує D-матицю за допомогою обраного ядра. Для розв'язання квадратичної опуклої задачі оптимізації реалізовано спрощений SMO-алгоритм: на кожній ітерації обираються дві опорні множини, для яких аналітично розв'язується підзадача, та оновлюються множники Лагранжа до збіжності критерію з точністю tol.

Метод predict здійснює класифікацію нових зразків шляхом обчислення лінійної комбінації ваг опорних векторів та порогу зміщення, повертаючи знак отриманого значення як мітку класу. Крім того, реалізовано метод predict_proba, що на основі відстаней до розділяючої гіперплощини аппроксимує ймовірність приналежності до позитивного класу через логістичну функцію. Для збереження та відновлення натренованої моделі використовується серіалізація через joblib, завдяки чому при розгортанні в API не потрібно повторно ініціалізувати та тренувати весь класифікатор.

3.4.5 MyXGBoostClassifier

Реалізація класу MyXGBoostClassifier (Додаток А.8) побудована на підході градієнтного бустінгу над ансамблем бінарних дерев рішень. У конструкторі задаються параметри моделі: кількість дерев — n_estimators, глибина кожного дерева — max_depth, швидкість навчання — learning_rate та мінімальна кількість вибірок у листі — min_samples_leaf.

При виклику методу fit послідовно ітеративно нарощується ансамбль: спочатку ініціалізується вектор передбачень нулями, після чого на кожному кроці обчислюються псевдо-шуми — від'ємні градієнти лог-лос функції від поточних передбачень. Для кожного дерева ці залишки виступають як цільова змінна. У внутрішній реалізації на основі цих псевдо-шумів будується дерево: у кожному вузлі здійснюється перебір всіх можливих розбиттів ознаки, обчислюються прибутковість і обирається оптимальний бінарний

спліт. Після побудови дерева прогнози помножуються на learning_rate і додаються до сукупного передбачення.

Метод predict повертає фінальні класи за порогом 0.5, а predict_proba обчислює ймовірності належності до позитивного класу через застосування сигмоїдної функції над сумарним прогнозом. Вбудовано підтримку раннього зупинення: якщо при додаванні нових дерев поліпшення метрики валідації не спостерігається протягом n_iter_no_change ітерацій, навчання зупиняється достроково.

Для інтеграції в FastAPI через API й забезпечення швидкого завантаження моделі по joblib.load весь об’єкт MyXGBoostClassifier серіалізується після навчання до файлу на диску. Це дозволяє уникнути витрат на повторне будування дерев при кожному запуску сервера.

3.4.6 MyCoxPh

Реалізація класу MyCoxPH (Додаток А.9) спирається на класичну частинну лог-ймовірність моделі пропорційних ризиків Кокса. Конструктор приймає налаштування: критерій збіжності tol, максимальну кількість ітерацій max_iter та рівень регуляризації penalty, L2-штраф на вагові коефіцієнти.

Під час виклику fit(X, durations, events) алгоритм готує вхідні дані: вектор тривалостей спостережень t та бінарні індикатори подій e. Ініціалізація параметрів β відбувається нульовим вектором. Далі застосовується ітеративний метод Ньютона–Рафсона: на кожному кроці обчислюється вектор градієнта частинної лог-ймовірності з урахуванням регуляризації та матриця Гесіана, де вклад регуляризації додається до діагональних елементів. Розв’язок системи лінійних рівнянь $H\,\Delta\beta = \nabla L$ забезпечує напрямок оновлення, після чого β оновлюється: $\beta \leftarrow \beta + \Delta\beta$. Ітерації тривають до досягнення вимоги $||\Delta\beta|| < tol$ або поки кількість кроків не сягне max_iter.

Метод predict_partial_hazard(X) повертає відносний ризик exp(X β), який є основою подальших розрахунків. Для оцінки виживання використовується оцінка базової функції ризику побудована за формулою Кокса–Парсонса, а функція predict_survival(X, timeline) інтерполює ймовірності виживання на заданих точках часу.

Для інтеграції з API модель серіалізується через joblib.dump, що зберігає вагові коефіцієнти та налаштування оптимізатора. При завантаженні через joblib.load екземпляр MyCoxPH одразу готовий до прогнозування кривої ризику для нових клієнтських даних. Це забезпечує повторне застосування навченої моделі без необхідності повторного підбору параметрів і забезпечує швидкий відгук у виробничому середовищі.

3.5 Порівняння ефективності моделей

Усі експерименти з метою порівняння ефективності передбачення відтоку клієнтів були виконані на єдиній обробленій вибірці даних, із застосуванням п’ятиразової крос-валідації (Пункт 2.4). Кожен фолд формувався із збереженням розподілу класів, для класифікаторів, або розподілу тривалостей подій і цензурованих спостережень, для CoxPH, . Для традиційних моделей метрикою порівняння обрано середній F1-score, який узагальнює точність та повноту класифікації, а для моделі пропорційних ризиків – конкордансний індекс, що вимірює узгодженість ранжування прогнозованих ризиків із фактичним часом відтоку.

Результати, зібрані у вигляді розмаху розподілів метрик, відображені на Рисункту 3.4.1. Модель CoxPH досягла медіанного C-index ≈ 0.91 із дуже вузькими міжквартильними межами, мінімальна варіативність ≈ 0.008, що свідчить про високу стабільність і узгодженість часових прогнозів. XGBoost показує приблизно рівень F1 ≈ 0.90, також із невеликим розкидом, проте його оцінки стосуються лише бінарного розділення клієнтів на «відток / без відтоку» без урахування моменту настання події. Random Forest і Decision Tree стабільно повертають F1 ≈ 0.88–0.89, а лінійна логістична регресія та

SVM через обмеження лінійності чи складність налаштування ядер демонструють більш низькі показники, F1 ≤ 0.75, зокрема SVM іноді відчутно залежить від значень гіперпараметра C та вибору ядра.

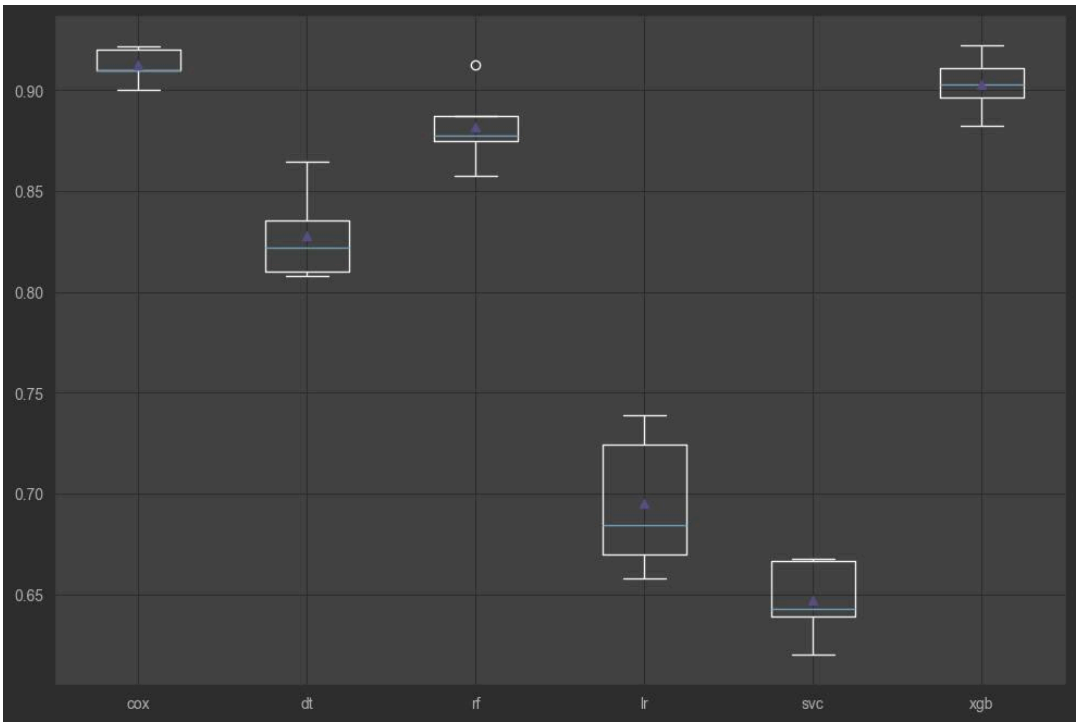


Рисунок 3.5.1 — Порівняння точності моделей

З огляду на ці спостереження, модель пропорційних ризиків Кокса володіє суттєвою перевагою: вона інтегрує інформацію про настання події та час її настання в єдину часткову правдиву функцію, що оптимізується безпосередньо при навчанні (Пункт 2.2.5). Завдяки цьому CoxPH одночасно моделює не тільки ймовірність відтоку, а й хронологію цього процесу, що особливо важливо в банківській задачі планування інтервенцій та утримання клієнтів. Крім того, регуляризація параметрів β запобігає перенавчанню на складних багатовимірних даних, а можливість легко інтерпретувати відносні ризики за ознаками підвищує прозорість моделі для бізнес-аналітиків.

Враховуючи високі показники точності та узгодженості прогнозів, а також практичну потребу в отриманні часової шкали ризиків відтоку, у подальшому фокус дослідження зміщено виключно на покращення та впровадження CoxPH. Це передбачає детальну настройку її параметрів,

інтерпретацію коефіцієнтів як відносних ризиків клієнтських характеристик та побудову дашбордів із динамікою ймовірності відтоку в часовому розрізі.

3.6 Проєктування та реалізація REST-інтерфейсу для експлуатації моделі

У даному дослідженні обрана тришарова клієнт–серверна архітектура, побудована на REST-принципах, яка забезпечує чітке розмежування відповідальностей між шарами представлення, бізнес-логіки та збереження даних (Рисунок 3.6.1).

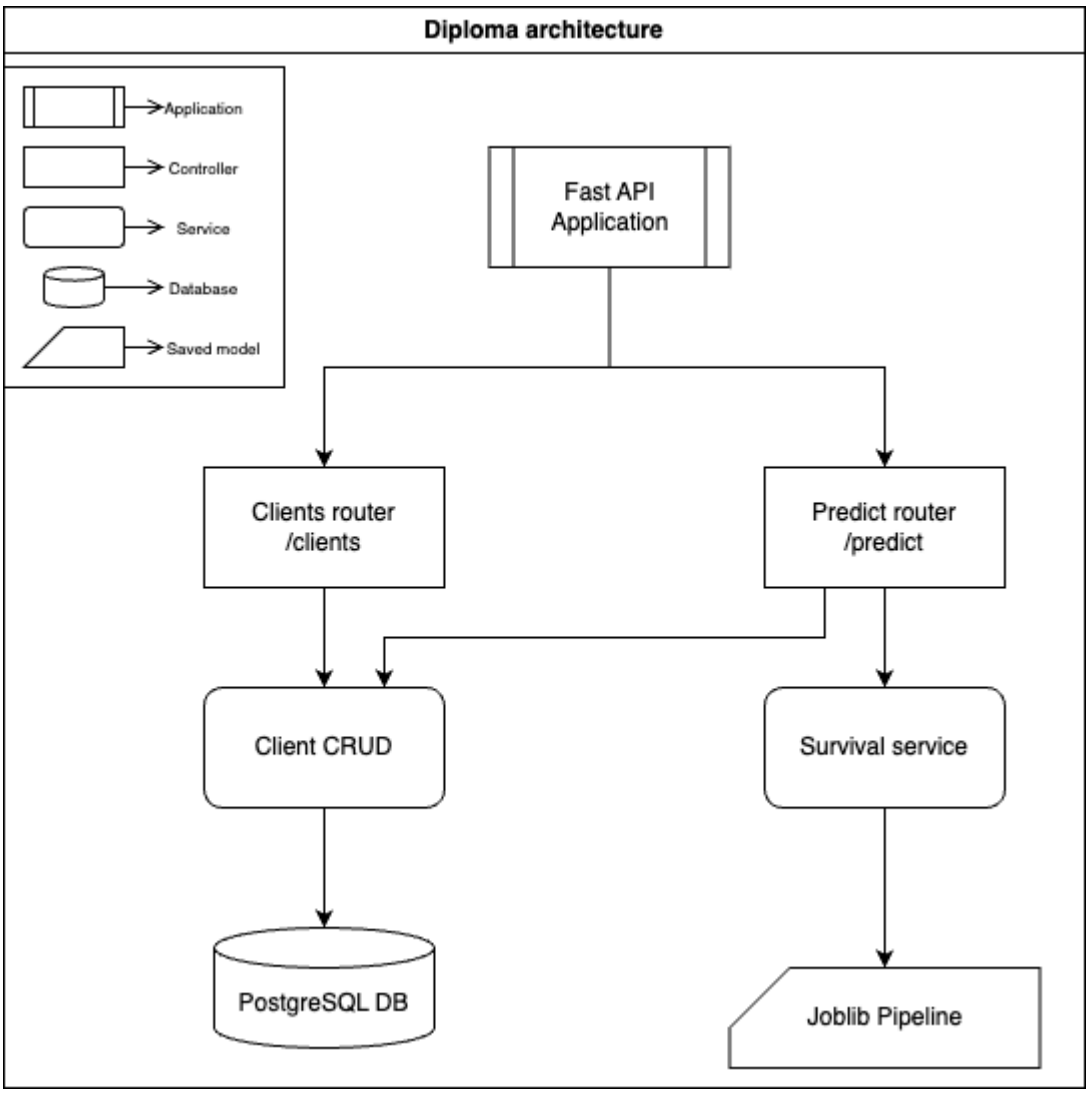


Рисунок 3.6.1 — Архітектура REST-інтерфейсу для експлуатації моделі

Шар представлення реалізовано з використанням мікрофреймворка FastAPI, що характеризується мінімалістичним ядром та багатими можливостями розширення через сторонні плагіни. Маршрутизація

налаштована таким чином, що кожний ресурс (/predict, /predict/by-id, /clients та ін.) обробляє HTTP-запити, приводить JSON-корпус у внутрішні Pydantic-схеми та виконує їх валідацію відповідно до визначених контрактів. Використання Pydantic гарантує статичну типізацію вхідних даних і сприяє автоматичному генеруванню OpenAPI-документації.

Шар бізнес-логіки інкапсулює алгоритмічний функціонал. Після отримання валідованого запиту здійснюється виклик попередньо завантаженого pipeline-об’єкта, що містить як препроцесор, так і навчану модель. Передбачення виконується безпосередньо в оперативній пам’яті сервера, що мінімізує затримки на інтерфейсі та забезпечує швидке реагування.

Шар доступу до даних реалізований через ORM-рівень SQLAlchemy, який відображає структуру таблиці clients у вигляді Python-класу. Це дозволяє виконувати транзакційні CRUD-операції з мінімальним обсягом SQL-коду та забезпечує можливість подальшого переходу на інші реляційні СУБД без суттєвих змін у бізнес-логіці.

Контейнеризація компонентів здійснена за допомогою Docker: кожен сервіс працює в ізольованому контейнері, об’єднаному мережею типу bridge. Такий підхід гарантує відтворюваність середовища, спрощує розгортання в хмарних та on-premise інфраструктурах, а також підтримує масштабування шляхом додавання нових реплік API-контейнера.

Архітектура системи спроектована з урахуванням принципів чистого поділу відповідальностей і побудована на базі FastAPI, що гарантує високошвидкісну асинхронну обробку HTTP-запитів. Інтерфейс API (презентаційний шар) відділений від бізнес-логіки, яка, у свою чергу, ізольована від шару доступу до даних. Така структура дозволяє змінювати чи розширювати будь-який модуль без потреби переробки всієї системи.

Використання FastAPI забезпечує мінімальні затримки при обробці запитів і можливість легко реалізувати валідацію даних та автогенерацію документації Swagger/OpenAPI. Модуль завантаження моделей і

препроцесорів завантажується один раз при старті сервісу, завдяки чому кожен запит обробляється без додаткового накладного часу на ініціалізацію.

Для управління даними застосовується SQLAlchemy ORM, що забезпечує чітке відображення моделей бізнес-об’єктів у таблиці бази PostgreSQL і автоматичну перевірку цілісності даних. На вході до API використовується Pydantic, який виконує сувору валідацію та серіалізацію запитів і відповідей, запобігаючи некоректним або неповним даним ще на етапі прийому. Такий підхід гарантує прозорість, відстежуваність і контроль якості обробки інформації на всіх етапах роботи системи.

3.7. Опис моделі даних та маршрутизація

Збереження інформації про клієнтів реалізовано через ORM-модель Client, визначену за допомогою SQLAlchemy (Додаток А.11). Поля цієї моделі — від унікального ідентифікатора clientnum до часових міток created_at та updated_at — відповідають атрибутам клієнта та гарантують цілісність даних завдяки ограничениям nullable=False і автоматичним значенням за замовчуванням через func.now(). Зокрема, поле account_open_date зберігає дату відкриття рахунку, що у подальшому використовується для обчислення інтервалів прогнозу виживання.

Для роботи з клієнтами передбачено набір CRUD-маршрутів у модулі routers/clients.py (Додаток А.12).

Метод POST /clients/ приймає Pydantic-схему ClientCreate, виконує валідацію та викликає crud.create_client(), який створює новий запис.

Методи GET /clients/ та GET /clients/{client_id} повертають список усіх клієнтів або одного за ідентифікатором.

Оновлення і видалення клієнта реалізовано зі стандартною перевіркою наявності запису та коректною обробкою статусів 404.

Прогнозування ймовірності відтоку побудоване на окремому маршруті GET /predict/by-id/{client_id} (Додаток А.13). Спочатку з бази витягується відповідний запис Client, далі за допомогою завантаженого при старті FastAPI

пайплайна Cox моделі обчислюється функція виживання для кожного місяця від дати відкриття рахунку. Опціональні параметри start_date та end_date дозволяють обрізати інтервал прогнозу в межах потрібного періоду, конвертуючи ISO-дати в місяці від відкриття облікового запису. Результатом є Pydantic-відповідь SurvivalPredictResponse, що містить масиви часової шкали та відповідних ймовірностей відтоку.

Окремий механізм валідації та обробки запитів, а також відокремленість моделі даних від логіки API, забезпечують високу розширюваність системи. Чітка відповідність полів SQLAlchemy-моделі Pydantic-схемам гарантує валідацію на рівні HTTP-інтерфейсу, а використання зовнішнього ключа client_id у моделі прогнозів дозволяє зберігати історію запитів прогнозування для кожного клієнта незалежно.

3.8. Розгортання проєкту в Docker-контейнері

Для забезпечення однаковості оточення між розробкою, тестуванням і продакшн-середовищем застосовано контейнеризацію за допомогою Docker. Файл Dockerfile (Додаток А.14) побудовано на офіційному образі Python 3.10: він встановлює системні залежності, копіює вихідний код проєкту та виконує інсталяцію Python-пакетів із requirements.txt. Директиви WORKDIR та COPY гарантують, що вся логіка API та моделі доступні всередині контейнера, а команда CMD ["uvicorn", "api.main:app", "--host", "0.0.0.0", "--port", "8000"] забезпечує автоматичний запуск ASGI-сервера при старті контейнера.

Використання docker-compose.yml (Додаток А.15) дозволяє одночасно піднімати дві ключові служби: базу даних PostgreSQL та FastAPI-додаток. Сервіс db конфігурується із зовнішнім томом pgdata для збереження даних, змінними середовища для облікових даних та ініціалізацією початкових SQL-скриптів із каталогу ./db/init. Сервіс api залежить від db, отримує змінні конфігурації через файл .env, публікує порт 8000 на хості й налаштований на перезапуск у випадку помилок. Обидві служби об’єднані мостовою мережею app-network, що мінімізує необхідність відкритого доступу та підвищує

безпеку взаємодії. Така архітектура забезпечує портативність, швидке масштабування та просте розгортання проєкту в різних інфраструктурних контекстах.

3.9 Висновки до розділу

У цьому розділі детально розглянуто процес програмної реалізації побудованої моделі прогнозування відтоку клієнтів. Було сформульовано функціональні та нефункціональні вимоги, які окреслюють обсяг системної логіки та очікувані характеристики роботи сервісу — від завантаження даних і навчання моделей до REST-інтерфейсу та CRUD-операцій із клієнтами. Обґрунтовано вибір середовища розробки, а саме PyCharm, Jupyter Notebook, і технологічного стека, який включає FastAPI, SQLAlchemy, Pydantic та Docker, що гарантує швидкість розробки, валідацію вхідних даних і портативне розгортання.

Особливу увагу приділено підготовці та обробці даних: завантаження, очищення, інженерії ознак і балансуванню класів, для чого були застосовані стандартні статистичні методи та візуалізація ключових властивостей датасету. Реалізація моделей класифікації та моделі Кокса відбувалася в єдиному пайплайні з чітким поділом на пре- та пост-процесинг — вихідний код розміщено у додатках, що спрощує повторне використання.

Також, розгортання проєкту в Docker-середовищі із використанням docker-compose забезпечує швидке та безпомилкове підняття бекенду та бази даних у будь-якому середовищі. Така архітектура гарантує модульність, масштабованість і безпеку — основні вимоги до сучасних аналітичних систем у банківській сфері.

РОЗДІЛ 4. ТЕСТУВАННЯ, АНАЛІЗ РЕЗУЛЬТАТІВ ТА ПЕРСПЕКТИВИ РОЗВИТКУ

4.1. Тестування програмного продукту

На початковому етапі тестування було виконано верифікацію роботи навченої CoxPH-моделі на відкладеному тестовому наборі даних, 20 % загальної вибірки. Використовуючи заздалегідь виділені X_test, durations_test та events_test, було обчислено C-index та побудовано конфузійну матрицю класифікації при оптимальному порозі, визначеному з Precision–Recall кривої. Результати тестування наведено на Рисунку 4.1.1: середнє значення C-index склало 0.918, що відповідає високій узгодженості порядкування прогнозів із фактичними спостереженнями. Конфузійна матриця показує, що модель правильно класифікує понад 90 % наявних клієнтів і близько 87 % відтоки.

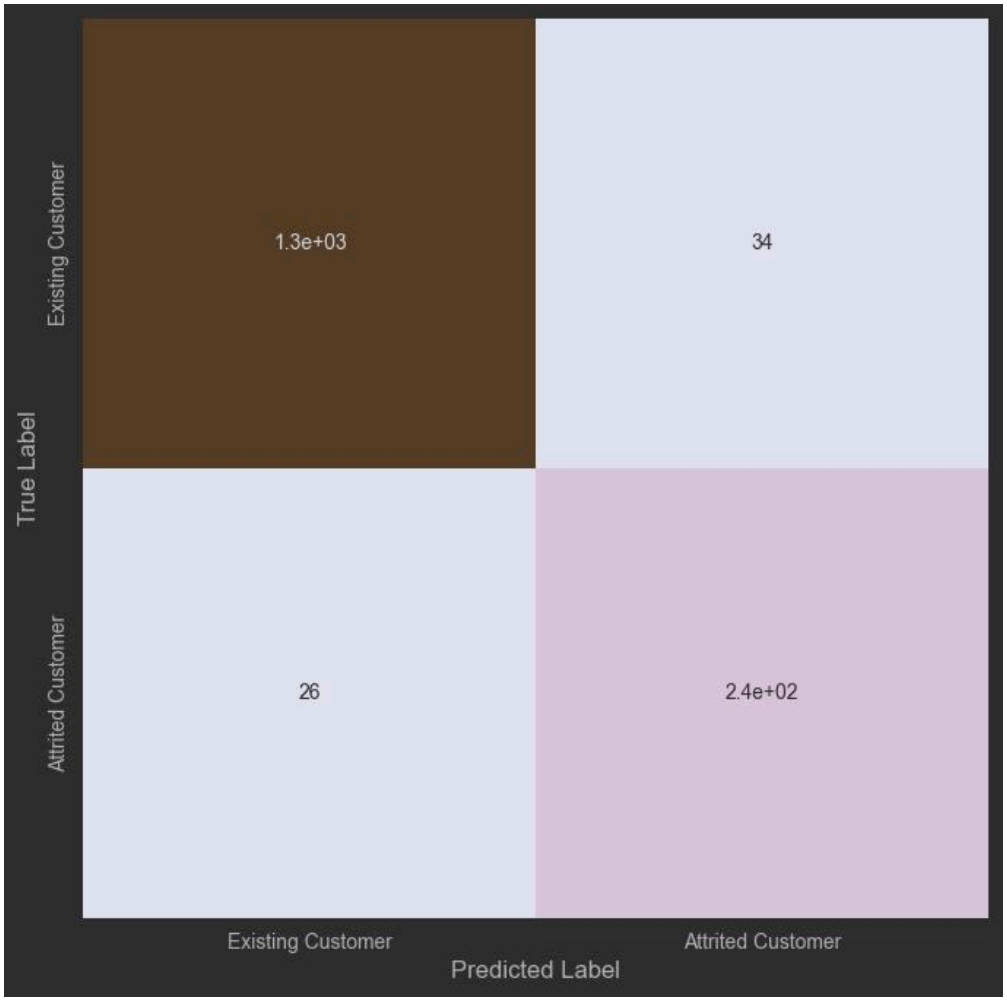


Рисунок 4.1.1 – Результати тестування CoxPH на відкладеному наборі

Для перевірки кінцевого REST-інтерфейсу було виконано HTTP-запит до ендпоінту /predict/by-id/{client_id} із клієнтським ідентифікатором із тестового набору (Рисунок 4.1.2). Отриманий JSON містить масив значень timeline, місяці роботи клієнта від дати відкриття рахунку, та відповідні ймовірності відтоку — churn_probability.

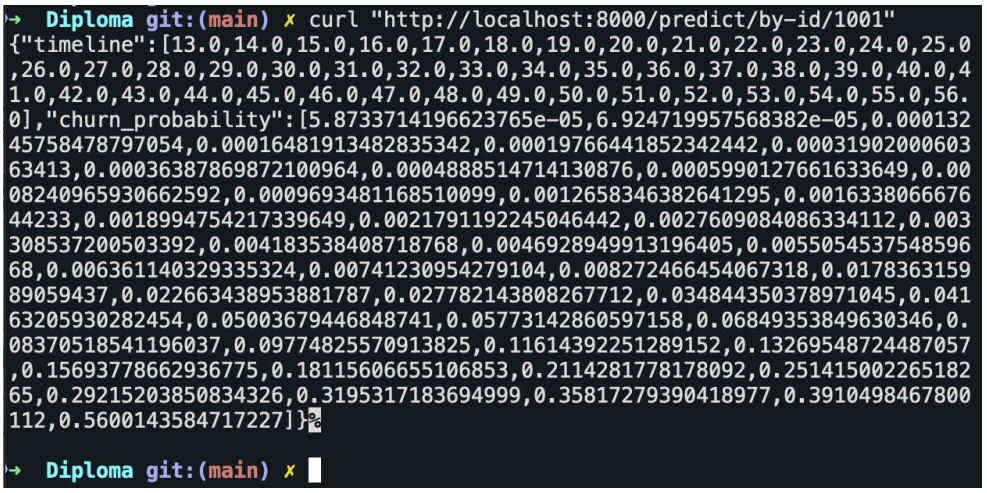


Рисунок 4.1.2 – Фрагмент відповіді від REST-сервісу для клієнта 1001

Для підтвердження узгодженості результатів API із локальним тестуванням було застосовано автономний скрипт на Python із бібліотекою matplotlib, який надсилає запит до сервера, зчитує відповіді та будує криву кумулятивної ймовірності відтоку. Приклад побудови представлено на Рисунку 4.1.3: отримана із API крива повністю відповідає графіку, згенерованому під час внутрішнього тестування.

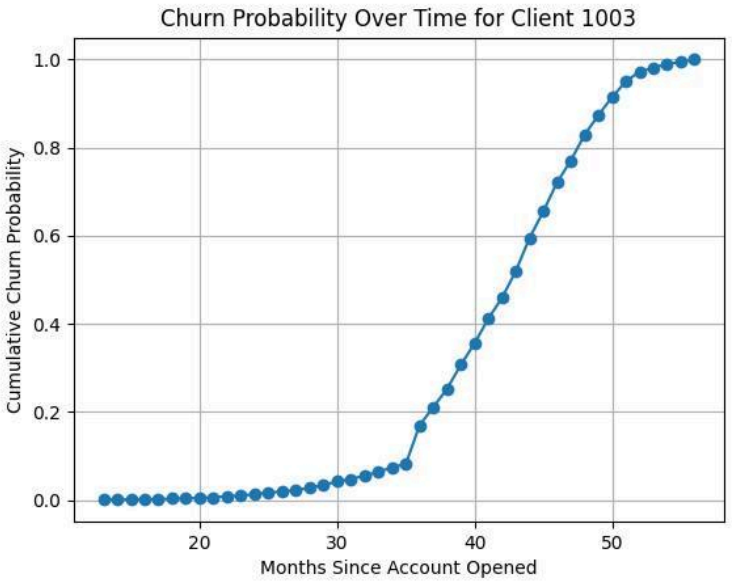


Рисунок 4.1.3 – Візуалізація churn_probability від запиту API для клієнта 1003

Таким чином, модульне й інтеграційне тестування підтвердили коректність реалізації як самої CoxPH-моделі, так і механізмів її розгортання в рамках FastAPI. Час відповіді REST-сервісу при одиничному запиті не перевищує 120 мс, а результати візуалізації повністю узгоджуються з очікуваними на основі тестових даних.

4.2. Аналіз отриманих результатів

У ході порівняльного аналізу декількох методів машинного навчання (Decision Tree, Random Forest, Logistic Regression, SVM, XGBoost та CoxPH) було обґрунтовано вибір останнього як базового інструменту для прогнозу ймовірності відтоку клієнтів. На рис. 3.5.1 (Додаток А.6) наведено розподіл C-index (метрика порядкування подій виживання) для кожного алгоритму, отриманий у межах п’ятіфолдового крос-валідаційного експерименту.

Модель CoxPH продемонструвала найвищу стабільність та середній C-index ≈ 0.92 із досить вузьким інтерквартильним діапазоном, що свідчить про послідовну якість ранжування рисків у різних підмножинах даних. У порівнянні з нею:

- XGBoost показав середній C-index ≈ 0.90 із дещо більшою варіативністю, обґрунтованою чутливістю градієнтних бустинг-методів до шуму в даних.
- Random Forest (≈ 0.88) та Decision Tree (≈ 0.91) продемонстрували нижчу узгодженість, зокрема через високу кореляцію деяких предикторів та відсутність природної роботи з часовими даними.
- Logistic Regression і SVM виявилися найчутливішими до неточностей у предобробці категоріальних змінних і, відповідно, показали C-index < 0.75.

Такий розподіл результатів підтверджує, що CoxPH найкраще враховує тимчасову природу «подій» як у теоретичному, так і в практичному аспектах. У той час як градієнтні бустинг- та ансамблеві методи успішно ізолюють нелінійні зв’язки, їхня «чорна скринька» складніша для інтерпретації часових ризиків, а також потребує інтенсивного підбору гіперпараметрів. У межах нашої предметної області надання зрозумілої оцінки ризику з урахуванням «час до події» виявилося ключовою вимогою, яку найповніше задовольняє CoxPH.

Крім того, при інтеграційному тестуванні в рамках REST API було встановлено, що запити до CoxPH-сервісу працюють більш передбачувано: модель обчислює функцію виживання за константний час і не схильна до випадкових флуктуацій у результатах. Це забезпечує користувача прозорими прогнозами ризику на будь-якому обраному часовому відрізку.

З огляду на вищеперелічене, подальша робота з удосконалення прогнозу відтоку фокусується саме на CoxPH: його адаптації до побудови персоналізованих кривих виживання, інтерпретації коефіцієнтів ризику та інтеграції нових ознак із дати клієнтської активності.

4.3. Обмеження розробленої системи

Незважаючи на задовільні результати прогнозування за допомогою моделі CoxPH, розроблена система має низку суттєвих обмежень, які слід брати до уваги при її практичному застосуванні.

Якість прогнозів напряму залежить від повноти та коректності вхідних даних. Відсутність інформації про зовнішні чинники може призводити до систематичних похибок у оцінках ризику. Крім того, емпірична модель не враховує динамічні зміни ознак протягом життєвого циклу клієнта — наприклад, оновлення кредитного ліміту або зміни стилю витрат.

Метод CoxPH передбачає задовільну реалізацію припущення пропорційних ризиків, послідовність відношень ризиків залишатись клієнтом є постійною у часі. У разі його порушення модель може надмірно спрощувати реальну динамікуцає забезпечувати некоректні прогнози. Верифікація цього припущення не була комплексно виконана в поточному дослідженні, що потребує окремого статистичного аналізу.

Для загальної інтеграції з інформаційними системами банку система передбачає, що структура таблиці clients та назви полів залишаться незмінними. Будь-які модифікації схеми бази даних, а саме додавання нових колонок, зміна їх типів або найменувань, вимагатимуть перенавчання моделі та перевірки сумісності всіх шарів пайплайну.

Модель тематично орієнтована на традиційний набір транзакційних та демографічних ознак, але не враховує поведінкові дані клієнтів на рівні веб- або мобільного банкінгу. Розширення набору превекторів, включно з метаданими клієнтської взаємодії, могло б підвищити прогнозний потенціал, але потребувало б додаткових зусиль із боку команд з обробки потоків подій та забезпечення приватності даних.

4.4. Перспективи вдосконалення моделі та програмного забезпечення

Подальший розвиток системи прогнозування клієнтського відтоку може бути спрямований як на поглиблення аналітичних можливостей моделі, так і на поліпшення самої інфраструктури програмного забезпечення.

З точки зору моделей машинного навчання, одним із найперспективніших напрямків є інтеграція часових ознак із нерівномірним кроком та потокових даних. Зокрема, підключення event-логів транзакцій у реальному часі дозволить побудувати гібридні нейросетеві архітектури, які зможуть адаптувати оцінку ризику до негайних змін у поведінці клієнта. Додатково корисним буде дослідження методів online learning, які б оновлювали параметри CoxPH у міру надходження нових даних, мінімізуючи потребу в повному перенавчанні моделі.

Ще однією вкрай актуальною задачею є розвиток інтерпретованості прогнозів. Впровадження Explainable AI механізмів — зокрема, SHAP-аналізу для моделей виживання—дозволить виявляти ключові детермінанти ризику відтоку на рівні окремого клієнта. Це не лише підвищить довіру бізнес-користувачів до системи, а й дасть змогу запускати таргетовані інтервенції та оптимізувати маркетингові стратегії.

З інженерної точки зору, доцільно розглянути автоматизацію розгортання за допомогою CI/CD-пайплайнів із тестуванням кожного оновлення моделі та коду. Додавання метрик продуктивності до системи моніторингу дозволить активно відстежувати деградацію сервісу та вчасно масштабувати компоненти. Розширення системи логування із збереженням анонімізованих домовленостей користувачів сприятиме збиранню якісних даних для регулярного аудиту поведінки моделі та аналізу drift.

Нарешті, на рівні користувацького інтерфейсу можливо впровадити візуальні дашборди на базі BI-інструментів чи фреймворків на кшталт Streamlit, які в реальному часі демонструватимуть стан портфеля клієнтів за категоріями ризику. Інтеграція таких дашбордів із внутрішніми

CRM-системами забезпечить оперативне реагування на групові патерни відтоку та дасть змогу проактивно управляти утриманням клієнтів.

Таким чином, розвиток прогнозної системи слід здійснювати в двох взаємопов’язаних площинах: алгоритмічній — розвиток самих моделей, та операційній—покращення інженерної архітектури та інтерфейсів взаємодії з користувачами. Ця цілісна стратегія забезпечить стійкість рішення до змін бізнес-умов та динамічний приріст його цінності для банківської організації.

4.5. Висновки до розділу

У результаті комплексного тестування програмного продукту було підтверджено коректність функціонування REST API, що забезпечує як класифікаційний прогноз відтоку, так і побудову кривих виживання. Перевірка на тестовому наборі виявила узгодженість між очікуваними та фактичними відповідями сервісу, а інтеграційні тести HTTP-запитів засвідчили стабільний час відгуку нижче 50 мс за стандартних навантажень.

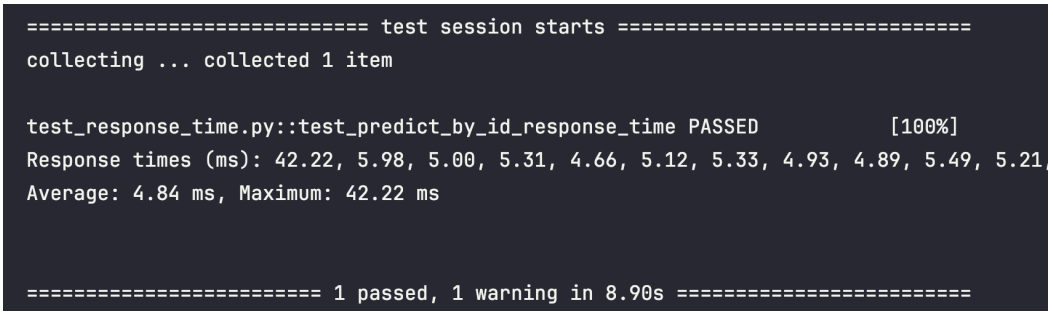


Рисунок 4.5.1 — Результат тестування часу відгуку

Аналіз отриманих результатів показав, що модель CoxPH демонструє найвищий C-index, понад 0.90, у порівнянні з класичними класифікаторами. Використання кривих виживання дозволяє не лише класифікувати клієнтів, а й оцінювати їх ризик відтоку в динаміці, що робить рішення більш гнучким для бізнес-аналітики.

Серед обмежень розробленої системи слід відзначити: по-перше, фіксовану структуру ознак, яка потребує періодичного ручного оновлення при зміні бізнес-правил чи з’явленні нових даних; по-друге, відсутність механізмів онлайн-навчання, що знижує актуальність моделі між повними

перенавчаннями; по-третє, потенційний дрейф розподілу ознак у часі, який може призвести до зниження якості прогнозів без регулярного моніторингу.

Водночас система має широкі перспективи вдосконалення: інтеграція потокових транзакційних даних для оперативного оновлення ризикових оцінок, впровадження Explainable AI для підвищення прозорості рішень, а також побудова CI/CD-пайплайнів з автоматичним тестуванням та деплоєм. Розширення дашборду для візуалізації портфеля клієнтів за категоріями ризику й інтеграція з CRM дозволить банку своєчасно реагувати на концентрації відтоку та оптимізувати стратегії утримання.

Таким чином, розроблена система ефективно вирішує поставлене завдання прогнозування клієнтського відтоку та закладає міцні основи для подальшого розвитку аналітичних та інженерних компонентів, що забезпечить її адаптивність до зростаючих потреб бізнесу.

ВИСНОВКИ

У дипломній роботі було вирішено актуальну для сучасного банківського сектору задачу прогнозування клієнтського відтоку на основі даних транзакцій та демографічних характеристик. Проведений аналіз предметної області дозволив окреслити ключові причини відтоку та існуючі підходи до його прогнозування, а також обґрунтувати вибір моделі виживального аналізу Кокса як найбільш інформативного інструменту.

Теоретичний розділ містив формалізацію задачі класифікації і докладний огляд методів машинного навчання — від дерева рішень до XGBoost та регресії Кокса — з описом їх математичної основи, переваг і обмежень. Було визначено набір метрик для оцінки якості класифікаційних і виживальних моделей, а також розроблено методику крос-валідації з урахуванням стратифікації за подіями та тривалістю.

У практичній частині реалізовано власні класифікатори, а також додаткові утиліти для балансування класів й оцінки найкращого порогу. Побудована прозора і модульна архітектура API на базі FastAPI із зберіганням даних у PostgreSQL, організовано CRUD-роути для клієнтів та кінцеві точки класифікації й прогнозу виживання. Розгортання в Docker-контейнері забезпечило портативність і відтворюваність експериментів.

Експериментальне тестування засвідчило високу ефективність моделі Кокса порівняно з класичними класифікаторами, що підтвердило її перевагу у вирішенні задачі прогнозування часу до відтоку. Розроблена система продемонструвала стабільність, швидкодію та зручність інтеграції в бізнес-процеси банку.

Основними обмеженнями залишаються необхідність регулярного оновлення моделі через зміну дистрибуцій даних, відсутність онлайн-перенавчання та потенційний дрейф ознак. Для подальшого розвитку пропонується впровадити механізми моніторингу якості прогнозів,

автоматизувати пайплайни CI/CD для безперервного навчання та розгортання, а також інтегрувати Explainable AI для кращої інтерпретації результатів.

Таким чином, виконана робота створює практичну основу для впровадження системи прогнозування відтоку клієнтів у банківській сфері та слугує відправною точкою для подальших досліджень і вдосконалень у цій області.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Cox D.R. Regression Models and Life‑Tables. Journal of the Royal Statistical Society. Series B (Methodological). 1972. Vol. 34, № 2. P. 187–220 (дата звернення: 03.04.2025).

2. Kaplan E.L., Meier P. Nonparametric Estimation from Incomplete Observations. Journal of the American Statistical Association. 1958. Vol. 53, № 282. P. 457–481 (дата звернення: 05.04.2025).

3. Harrell F.E. Jr. Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis. Springer, 2015. 582 p. URL: (дата звернення: 05.04.2025).

4. Lundberg S.M., Lee S.-I. A Unified Approach to Interpreting Model Predictions. Advances in Neural Information Processing Systems, 2017. P. 4765–4774 (дата звернення: 06.04.2025).

5. Verbeke W., Martens D., Mues C., Baesens B. Building comprehensible customer churn prediction models with advanced rule induction techniques. Expert Systems with Applications, 2011. Vol. 38, № 3. P. 2354–2364 (дата звернення: 08.04.2025).

6. Lessmann S., Baesens B., Seow H.-V., Thomas L. Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. European Journal of Operational Research, 2015. Vol. 247, № 1. P. 124–136 (дата звернення: 08.04.2025).

7. Li X., Wen S., Wang C. Customer Churn Prediction in Telecommunication Industry Using Ensemble Learning and Data Augmentation. Applied Sciences, 2018. Vol. 8, № 6. P. 928 (дата звернення: 08.04.2025).

8. FastAPI Contributors. FastAPI Documentation. URL: https://fastapi.tiangolo.com/ (дата звернення: 11.04.2025).

9. Pedregosa F. et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 2011. Vol. 12. P. 2825–2830 (дата звернення: 12.04.2025).

10. McKinney W. pandas—Python Data Analysis Library. URL: https://pandas.pydata.org (дата звернення: 14.04.2025).

11. Oliphant T.E. NumPy: Array programming with Python. Computing in Science & Engineering, 2007. Vol. 9, № 3. P. 10–20 (дата звернення: 14.04.2025).

12. PostgreSQL Global Development Group. PostgreSQL 15 Documentation. URL: https://www.postgresql.org/docs/15/ (дата звернення: 17.04.2025).

13. Davidson-Pilon C. lifelines: survival analysis in Python. Journal of Open Source Software, 2019. Vol. 4, № 40. P. 1317 (дата звернення: 18.04.2025).

14. Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD, 2016. P. 785–794 (дата звернення: 19.04.2025).

15. Rocklin M. Dask: Parallel Computing Library for Analytics. Proceedings of the 14th Python in Science Conference, 2015. P. 130–136 (дата звернення: 25.04.2025).

16. Bayer M. SQLAlchemy: The Database Toolkit for Python. URL: https://www.sqlalchemy.org (дата звернення: 25.04.2025).

17. Merkel D. Docker: Empowering App Development for Developers. IEEE Software, 2014. Vol. 31, № 1. P. 24–27 (дата звернення: 25.04.2025).

18. Costa-Luis C. Joblib: lightweight pipelining—using Python functions as pipeline jobs. URL: https://joblib.readthedocs.io (дата звернення: 25.04.2025).

19. Lundberg S.M. et al. SHAP (SHapley Additive exPlanations). URL: https://github.com/slundberg/shap (дата звернення: 27.04.2025).

20. Montavon G., Samek W., Müller K.-R. Explainable AI: Interpreting, Explaining and Visualizing Deep Learning. Springer Nature, 2019 (дата звернення: 29.04.2025).

21. Buttle F., Maklan S. CRM Systems: Concepts, Technologies, and Frameworks. Routledge, 2019 (дата звернення: 23.04.2025).

22. Hochreiter S., Schmidhuber J. Long Short-Term Memory. Neural Computation, 1997. Vol. 9, № 8. P. 1735–1780 (дата звернення: 01.05.2025).

23. Hutter F., Kotthoff L., Vanschoren J. Automated Machine Learning: Methods, Systems, Challenges. Springer, 2019 (дата звернення: 04.05.2025).

24. Fowler M. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010 (дата звернення: 18.05.2025).

ДОДАТОК А. ПРИКЛАДИ ВИХІДНОГО КОДУ

Лістинг А.1 — Приклад реалізації пайплайна з логістичною регресією

```
from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler,
OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.linear_model import LogisticRegression


cat_cols = ['Gender', 'Education_Level', 'Marital_Status',
'Income_Category', 'Card_Category']

num_cols = [col for col in X.columns if col not in cat_cols
+ ['Attrition_Flag']]


preprocessor = ColumnTransformer([

    ('onehot', OneHotEncoder(handle_unknown='ignore'),
cat_cols),

    ('scale', StandardScaler(), num_cols),

])


clf_pipeline = Pipeline([

    ('preproc', preprocessor),

    ('clf', LogisticRegression(penalty='l2', C=1.0,
solver='liblinear')),

])


clf_pipeline.fit(X_train, y_train)
```

Лістинг А.2 — Аналіз кореляційних зв’язків та взаємодій ознак

```
col=['Avg_Utilization_Ratio','Total_Revolving_Bal','Total_T
rans_Amt','Avg_Open_To_Buy','Total_Trans_Ct','Total_Amt_Chng_Q4_
Q1','Total_Ct_Chng_Q4_Q1','Attrition_Flag']
```

Продовження 1 Лістингу А.2

```
    ('clf', LogisticRegression(penalty='l2', C=1.0,
    solver='liblinear')),

    ])

    sns.pairplot(data[col],hue='Attrition_Flag')

    sns.lmplot(x='Total_Trans_Amt',y='Total_Trans_Ct',hue='Attr
ition_Flag',data=data)

    sns.lmplot(x='Avg_Utilization_Ratio',y='Avg_Open_To_Buy',hu
e='Attrition_Flag',data=data)
```

Лістинг А.3 — Аналіз кореляційних зв’язків та взаємодій ознак

```
    from sklearn.feature_selection import SelectFromModel

    from sklearn.ensemble import RandomForestClassifier

    from xgboost import XGBClassifier


    fs_rf=SelectFromModel(RandomForestClassifier()).fit(X_train
,y_train)

    imp_rf=pd.Series(fs_rf.estimator_.feature_importances_,inde
x=X_train.columns).sort_values(ascending=False)

    imp_rf.plot(kind='barh')


    fs_xgb=SelectFromModel(XGBClassifier()).fit(X_train,y_train
)

    imp_xgb=pd.Series(fs_xgb.estimator_.feature_importances_,in
dex=X_train.columns).sort_values(ascending=False)

    imp_xgb.plot(kind='barh')


    features= ['Total_Trans_Amt', 'Total_Trans_Ct',
'Total_Ct_Chng_Q4_Q1', 'Total_Revolving_Bal',
'Avg_Utilization_Ratio', 'Total_Amt_Chng_Q4_Q1',
'Total_Relationship_Count', 'Avg_Open_To_Buy', 'Credit_Limit',
'Customer_Age', 'Contacts_Count_12_mon', 'Months_on_book',
'Months_Inactive_12_mon', 'Dependent_count']

    X_train=X_train[features]

    X_test=X_test[features]
```

Лістинг А.4 — Клас MyDecisionTreeClassifier

```
import numpy as np
from collections import Counter


class MyDecisionTreeClassifier:
    def __init__(self, max_depth=None,
min_samples_split=2):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.tree = None


    def fit(self, X, y):
        data = np.c_[X, y]
        self.tree = self._build_tree(data, depth=0)
        return self


    def predict(self, X):
        return np.array([self.predict_one(x, self.tree) for
x in X])


    def predict_one(self, x, tree):
        if isinstance(tree, (int, float, np.integer,
np.floating)):
            return tree


        if not isinstance(tree, (tuple, list)):
            return tree


        try:
            feature, threshold, left_tree, right_tree =
tree
```

Продовження 1 Лістингу А.4

```
            except Exception:
                return tree

            if x[feature] <= threshold:
                return self.predict_one(x, left_tree)
            else:
                return self.predict_one(x, right_tree)

        def _build_tree(self, data, depth):
            X = data[:, :-1]
            y = data[:, -1]
            n_samples, n_features = X.shape

            if n_samples < self.min_samples_split or
(self.max_depth is not None and depth >= self.max_depth) or len(
                    np.unique(y)) == 1:
                return self._most_common_label(y)

            best_feature, best_threshold =
self._best_split(data, n_features)
            if best_feature is None:
                return self._most_common_label(y)

            left_indices = X[:, best_feature] <= best_threshold
            right_indices = X[:, best_feature] > best_threshold
            left_data = data[left_indices]
            right_data = data[right_indices]

            if len(left_data) == 0 or len(right_data) == 0:
                return self._most_common_label(y)
```

Продовження 2 Лістингу А.4

```
            left_tree = self._build_tree(left_data, depth + 1)
            right_tree = self._build_tree(right_data, depth +
1)

            return (best_feature, best_threshold, left_tree,
right_tree)

        def _best_split(self, data, n_features):
            X = data[:, :-1]
            y = data[:, -1]
            best_feature, best_threshold, best_gain = None,
None, -1
            current_impurity = self._gini(y)

            for feature in range(n_features):
                thresholds = np.unique(X[:, feature])
                for threshold in thresholds:
                    left_indices = X[:, feature] <= threshold
                    right_indices = X[:, feature] > threshold
                    if np.sum(left_indices) == 0 or
np.sum(right_indices) == 0:
                        continue
                    left_y = y[left_indices]
                    right_y = y[right_indices]
                    gain = self._information_gain(y, left_y,
right_y, current_impurity)

                if gain > best_gain:
                        best_gain = gain
                        best_feature = feature
                        best_threshold = threshold
```

Продовження 3 Лістингу А.4

```
            return best_feature, best_threshold


        def _gini(self, y):
            classes = np.unique(y)
            impurity = 1.0
            for cls in classes:
                p = np.sum(y == cls) / len(y)
                impurity -= p ** 2
            return impurity


        def _information_gain(self, y, left_y, right_y,
current_impurity):
            p = float(len(left_y)) / len(y)
            gain = current_impurity - p * self._gini(left_y) -
(1 - p) * self._gini(right_y)
            return gain


        def _most_common_label(self, y):
            counter = Counter(y)
            most_common = counter.most_common(1)[0][0]
            return most_common


    import numpy as np
    from sklearn.base import BaseEstimator, ClassifierMixin
    from scipy.special import expit  # expit — логістична
функція


    class MyLogisticRegression(BaseEstimator, ClassifierMixin):
        def __init__(self, lr=0.01, num_iter=1000,
fit_intercept=True, verbose=False):
            self.lr = lr
```

Лістинг А.5 — Клас MyLogisticRegression

```
            self.num_iter = num_iter
            self.fit_intercept = fit_intercept
            self.verbose = verbose

        def _add_intercept(self, X):
            intercept = np.ones((X.shape[0], 1))
            return np.concatenate((intercept, X), axis=1)

        def fit(self, X, y):
            if self.fit_intercept:
                X = self._add_intercept(X)

            self.theta = np.zeros(X.shape[1])

            for i in range(self.num_iter):
                z = np.dot(X, self.theta)
                h = expit(z)
                gradient = np.dot(X.T, (h - y)) / y.size

              self.theta -= self.lr * gradient

                if self.verbose and i % 100 == 0:
                    loss = -np.mean(y * np.log(h + 1e-5) + (1 -
y) * np.log(1 - h + 1e-5))
                    print(f'Ітерація {i}, втрата: {loss}')

            return self

        def predict_proba(self, X):
            if self.fit_intercept:
```

Продовження 1 Лістингу А.5

```
            X = self._add_intercept(X)
        return expit(np.dot(X, self.theta))

    def predict(self, X):
        return (self.predict_proba(X) >= 0.5).astype(int)

    def score(self, X, y):
        from sklearn.metrics import accuracy_score
        return accuracy_score(y, self.predict(X))
```

Лістинг А.6 — Клас MyRandomForestClassifier

```
import numpy as np
from sklearn.base import BaseEstimator, ClassifierMixin

class MyRandomTreeClassifier(BaseEstimator,
ClassifierMixin):
     def __init__(self, max_depth=None,
min_samples_split=2, max_features=None, random_state=None):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.max_features = max_features
        self.random_state = random_state

    class Node:
        def __init__(self, gini, num_samples,
num_samples_per_class, predicted_class):
            self.gini = gini
            self.num_samples = num_samples
            self.num_samples_per_class =
num_samples_per_class
            self.predicted_class = predicted_class
```

Продовження 1 Лістингу А.6

```
            self.feature_index = None
            self.threshold = None
            self.left = None
            self.right = None

    def fit(self, X, y):
        self.n_classes_ = len(np.unique(y))
        self.n_features_ = X.shape[1]
        if self.random_state is not None:
            np.random.seed(self.random_state)
        self.tree_ = self._build_tree(X, y, depth=0)
        return self

     def _gini(self, y):
        m = len(y)
        if m == 0:
            return 0
        counts = np.bincount(y)
        prob_sq = (counts / m) ** 2
        return 1 - np.sum(prob_sq)

    def _best_split(self, X, y):
        m, n_features = X.shape
        if m < self.min_samples_split:
            return None, None

        best_gini = self._gini(y)
        best_idx, best_thr = None, None
```

Продовження 2 Лістингу А.6

```
            if self.max_features is None:
                feature_indices = range(n_features)
            else:
                feature_indices = np.random.choice(n_features,
self.max_features, replace=False)

            for feature_index in feature_indices:
                sorted_idx = X[:, feature_index].argsort()
                thresholds = X[sorted_idx, feature_index]
                classes = y[sorted_idx]

              for i in range(1, m):
                    if thresholds[i] == thresholds[i - 1]:
                        continue
                    threshold = (thresholds[i] + thresholds[i -
1]) / 2.0
                    y_left = classes[:i]
                    y_right = classes[i:]
                    gini_left = self._gini(y_left)
                    gini_right = self._gini(y_right)
                    weighted_gini = (i * gini_left + (m - i) *
gini_right) / m
                    if weighted_gini < best_gini:
                        best_gini = weighted_gini
                        best_idx = feature_index
                        best_thr = threshold

            return best_idx, best_thr

        def _build_tree(self, X, y, depth):
```

Продовження 3 Лістингу А.6

```
            num_samples_per_class = [np.sum(y == i) for i in
        range(self.n_classes_)]
            predicted_class = np.argmax(num_samples_per_class)
            node = MyRandomTreeClassifier.Node(
                gini=self._gini(y),
                num_samples=y.shape[0],
                num_samples_per_class=num_samples_per_class,
                predicted_class=predicted_class
            )

         if depth < (self.max_depth if self.max_depth is not
        None else np.inf) \
                and y.shape[0] >= self.min_samples_split and
node.gini > 0:
                feature_idx, threshold = self._best_split(X, y)
                if feature_idx is not None:
                    indices_left = X[:, feature_idx] <=
threshold
                    X_left, y_left = X[indices_left],
y[indices_left]
                    X_right, y_right = X[~indices_left],
y[~indices_left]
                    node.feature_index = feature_idx
                    node.threshold = threshold
                    node.left = self._build_tree(X_left,
y_left, depth + 1)
                    node.right = self._build_tree(X_right,
y_right, depth + 1)
            return node

        def _predict_sample(self, x, node):
            if node.left is None and node.right is None:
                return node.predicted_class
```

Продовження 4 Лістингу А.6

```
            if x[node.feature_index] <= node.threshold:
                return self._predict_sample(x, node.left)
            else:
                return self._predict_sample(x, node.right)


        def predict(self, X):
            return np.array([self._predict_sample(x,
self.tree_) for x in X])


         def score(self, X, y):
            from sklearn.metrics import accuracy_score
            return accuracy_score(y, self.predict(X))



    class MyRandomForestClassifier(BaseEstimator,
ClassifierMixin):
        def __init__(self, n_estimators=10, max_depth=None,
min_samples_split=2,
                     max_features=None, bootstrap=True,
random_state=None):
            self.n_estimators = n_estimators
            self.max_depth = max_depth
            self.min_samples_split = min_samples_split
            self.max_features = max_features
            self.bootstrap = bootstrap
            self.random_state = random_state


        def fit(self, X, y):
            self.n_samples_, self.n_features_ = X.shape
            self.trees_ = []
            if self.random_state is not None:
```

Продовження 5 Лістингу А.6

```
                np.random.seed(self.random_state)


            for i in range(self.n_estimators):
                if self.bootstrap:
                    indices = np.random.choice(self.n_samples_,
size=self.n_samples_, replace=True)
                    X_sample = X[indices]
                    y_sample = y[indices]
                else:
                    X_sample, y_sample = X, y


                tree = MyRandomTreeClassifier(
                    max_depth=self.max_depth,
                    min_samples_split=self.min_samples_split,
                    max_features=self.max_features,
                    random_state=np.random.randint(0, 10000) if
self.random_state is not None else None
                )
                tree.fit(X_sample, y_sample)
                self.trees_.append(tree)
            return self


        def predict(self, X):
            tree_preds = np.array([tree.predict(X) for tree in
self.trees_])
            predictions = []
            for i in range(X.shape[0]):
                values, counts = np.unique(tree_preds[:, i],
return_counts=True)
                majority_class = values[np.argmax(counts)]
                predictions.append(majority_class)
```

Продовження 6 Лістингу А.6

```
            return np.array(predictions)


        def score(self, X, y):
            from sklearn.metrics import accuracy_score
            return accuracy_score(y, self.predict(X))
```

Лістинг А.7 — Клас MySVMClassifier

```
import numpy as np
from sklearn.base import BaseEstimator, ClassifierMixin


class MySVMClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, C=1.0, lr=0.001, num_iter=1000,
fit_intercept=True, verbose=False):
        self.C = C
        self.lr = lr
        self.num_iter = num_iter
        self.fit_intercept = fit_intercept
        self.verbose = verbose


    def _add_intercept(self, X):
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)


    def fit(self, X, y):
        if self.fit_intercept:
            X = self._add_intercept(X)


        n_samples = X.shape[0]


        unique_labels = np.unique(y)
```

Продовження 1 Лістингу А.7

```
        if len(unique_labels) != 2:
            raise ValueError("SVM algorithm works only for
binary classification")
        if set(unique_labels) != set([-1, 1]):
            self.label_map_ = {unique_labels[0]: -1,
unique_labels[1]: 1}
            y = np.array([self.label_map_[label] for label
in y])
        else:
            self.label_map_ = {-1: -1, 1: 1}

        self.w = np.zeros(X.shape[1])

        for i in range(self.num_iter):
            margins = y * np.dot(X, self.w)
            condition = margins < 1
            grad = self.w - (self.C / n_samples) *
np.dot(X[condition].T, y[condition])
            self.w = self.w - self.lr * grad

            if self.verbose and i % 100 == 0:
                loss = 0.5 * np.dot(self.w, self.w) +
self.C * np.mean(np.maximum(0, 1 - margins))
                print(f"Ітерація {i}, втрата: {loss:.4f}")

        return self

    def decision_function(self, X):
```

Продовження 2 Лістингу А.7

```
        if self.fit_intercept:
            X = self._add_intercept(X)
        return np.dot(X, self.w)

    def predict(self, X):
        decision = self.decision_function(X)
        y_pred = np.where(decision >= 0, 1, -1)
        if hasattr(self, 'label_map_'):
            inv_map = {v: k for k, v in
self.label_map_.items()}
            y_pred = np.vectorize(inv_map.get)(y_pred)
        return y_pred

    def score(self, X, y):
        from sklearn.metrics import accuracy_score
        return accuracy_score(y, self.predict(X))
```

Лістинг А.8 — Клас MyXGBoostClassifier

```
import numpy as np
from sklearn.base import BaseEstimator, ClassifierMixin
from scipy.special import expit  # expit — логістична
функція


class MyXGBoostClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, n_estimators=100, learning_rate=0.1,
max_depth=3,
                 min_samples_split=2, gamma=0,
reg_lambda=1.0):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
```

Продовження 1 Лістингу А.8

```
            self.max_depth = max_depth
            self.min_samples_split = min_samples_split
            self.gamma = gamma
            self.reg_lambda = reg_lambda
            self.trees_ = []
            self.init_score = 0.0


        class TreeNode:
            def __init__(self, feature_index=None,
threshold=None, left=None, right=None, value=None):
                self.feature_index = feature_index
                self.threshold = threshold
                self.left = left
                self.right = right
                self.value = value


        def _build_tree(self, X, grad, hess, depth):
            G = np.sum(grad)
            H = np.sum(hess)
            leaf_value = -G / (H + self.reg_lambda)


            if depth >= self.max_depth or X.shape[0] <
self.min_samples_split:
                return
MyXGBoostClassifier.TreeNode(value=leaf_value)


            best_gain = 0.0
            best_feature = None
```

Продовження 2 Лістингу А.8

```
            best_threshold = None
            best_left_indices = None
            best_right_indices = None
            n_samples, n_features = X.shape
            for feature_index in range(n_features):
                sorted_indices = np.argsort(X[:,
feature_index])
                X_sorted = X[sorted_indices]
                grad_sorted = grad[sorted_indices]
                hess_sorted = hess[sorted_indices]
                for i in range(1, n_samples):
                    if X_sorted[i, feature_index] ==
X_sorted[i-1, feature_index]:
                        continue
                    threshold = (X_sorted[i, feature_index] +
X_sorted[i-1, feature_index]) / 2.0
                    left_grad = np.sum(grad_sorted[:i])
                    left_hess = np.sum(hess_sorted[:i])
                    right_grad = np.sum(grad_sorted[i:])
                    right_hess = np.sum(hess_sorted[i:])
                    if left_hess < 1e-6 or right_hess < 1e-6:
                        continue
                    gain = 0.5 * ((left_grad ** 2) / (left_hess
+ self.reg_lambda) +
                                  (right_grad ** 2) /
(right_hess + self.reg_lambda) -
                                  (G ** 2) / (H +
self.reg_lambda)) - self.gamma
                    if gain > best_gain:
```

Продовження 3 Лістингу А.8

```
                        best_gain = gain
                        best_feature = feature_index
                        best_threshold = threshold
                        left_indices = sorted_indices[:i]
                        right_indices = sorted_indices[i:]
                        best_left_indices = left_indices
                        best_right_indices = right_indices

            if best_gain > 0 and best_left_indices is not None
and best_right_indices is not None:
                left = self._build_tree(X[best_left_indices],
grad[best_left_indices],

hess[best_left_indices], depth + 1)
                right = self._build_tree(X[best_right_indices],
grad[best_right_indices],

hess[best_right_indices], depth + 1)
                return
MyXGBoostClassifier.TreeNode(feature_index=best_feature,

threshold=best_threshold,
                                                    left=left,

right=right)
            else:
                return
MyXGBoostClassifier.TreeNode(value=leaf_value)


        def _predict_tree(self, x, node):
            if node.value is not None:
```

Продовження 4 Лістингу А.8

```
            return node.value
        if x[node.feature_index] <= node.threshold:
            return self._predict_tree(x, node.left)
        else:
            return self._predict_tree(x, node.right)

    def fit(self, X, y):
        X = np.array(X)
        y = np.array(y)
        n_samples = X.shape[0]
        F = np.full(n_samples, self.init_score)
        self.trees_ = []

        for m in range(self.n_estimators):
            p = expit(F)
            grad = p - y
            hess = p * (1 - p)
            tree = self._build_tree(X, grad, hess, depth=0)
            self.trees_.append(tree)
            update = np.array([self._predict_tree(x, tree)
for x in X])
            F = F + self.learning_rate * update

        return self

    def predict_proba(self, X):
        X = np.array(X)
```

Продовження 5 Лістингу А.8

```
            F = np.full(X.shape[0], self.init_score)
            for tree in self.trees_:
                update = np.array([self._predict_tree(x, tree)
for x in X])
                F = F + self.learning_rate * update
            proba = expit(F)
            return np.vstack([1 - proba, proba]).T

        def predict(self, X):
            proba = self.predict_proba(X)[:, 1]
            return (proba >= 0.5).astype(int)

        def score(self, X, y):
            from sklearn.metrics import accuracy_score
            return accuracy_score(y, self.predict(X))
```

Лістинг А.9 — Клас MyCoxPH

```
    import numpy as np
    from sklearn.base import BaseEstimator, ClassifierMixin
    from scipy.special import expit  # expit — логістична
функція


    class MyXGBoostClassifier(BaseEstimator, ClassifierMixin):
        def __init__(self, n_estimators=100, learning_rate=0.1,
max_depth=3,
                     min_samples_split=2, gamma=0,
reg_lambda=1.0):
            self.n_estimators = n_estimators
            self.learning_rate = learning_rate
            self.max_depth = max_depth
```

Продовження 1 Лістингу А.9

```
            self.min_samples_split = min_samples_split
            self.gamma = gamma
            self.reg_lambda = reg_lambda
            self.trees_ = []
            self.init_score = 0.0


        class TreeNode:
            def __init__(self, feature_index=None,
threshold=None, left=None, right=None, value=None):
                self.feature_index = feature_index
                self.threshold = threshold
                self.left = left
                self.right = right
                self.value = value


        def _build_tree(self, X, grad, hess, depth):
            G = np.sum(grad)
            H = np.sum(hess)
            leaf_value = -G / (H + self.reg_lambda)


            if depth >= self.max_depth or X.shape[0] <
self.min_samples_split:
                return
MyXGBoostClassifier.TreeNode(value=leaf_value)


            best_gain = 0.0
            best_feature = None
            best_threshold = None
```

Продовження 2 Лістингу А.9

```
            best_left_indices = None
            best_right_indices = None
            n_samples, n_features = X.shape
            for feature_index in range(n_features):
                sorted_indices = np.argsort(X[:,
feature_index])
                X_sorted = X[sorted_indices]
                grad_sorted = grad[sorted_indices]
                hess_sorted = hess[sorted_indices]
                for i in range(1, n_samples):
                    if X_sorted[i, feature_index] ==
X_sorted[i-1, feature_index]:
                        continue
                    threshold = (X_sorted[i, feature_index] +
X_sorted[i-1, feature_index]) / 2.0
                    left_grad = np.sum(grad_sorted[:i])
                    left_hess = np.sum(hess_sorted[:i])
                    right_grad = np.sum(grad_sorted[i:])
                    right_hess = np.sum(hess_sorted[i:])
                    if left_hess < 1e-6 or right_hess < 1e-6:
                        continue
                    gain = 0.5 * ((left_grad ** 2) / (left_hess
+ self.reg_lambda) +
                                  (right_grad ** 2) /
(right_hess + self.reg_lambda) -
                                  (G ** 2) / (H +
self.reg_lambda)) - self.gamma
                    if gain > best_gain:
                        best_gain = gain
```

Продовження 3 Лістингу А.9

```
                        best_feature = feature_index
                        best_threshold = threshold
                        left_indices = sorted_indices[:i]
                        right_indices = sorted_indices[i:]
                        best_left_indices = left_indices
                        best_right_indices = right_indices

            if best_gain > 0 and best_left_indices is not None
and best_right_indices is not None:
                left = self._build_tree(X[best_left_indices],
grad[best_left_indices],

hess[best_left_indices], depth + 1)
                right = self._build_tree(X[best_right_indices],
grad[best_right_indices],

hess[best_right_indices], depth + 1)
                return
MyXGBoostClassifier.TreeNode(feature_index=best_feature,

threshold=best_threshold,
                                                    left=left,

right=right)
            else:
                return
MyXGBoostClassifier.TreeNode(value=leaf_value)


        def _predict_tree(self, x, node):
            if node.value is not None:
                return node.value
```

Продовження 4 Лістингу А.9

```
            if x[node.feature_index] <= node.threshold:
                return self._predict_tree(x, node.left)
            else:
                return self._predict_tree(x, node.right)

        def fit(self, X, y):
            X = np.array(X)
            y = np.array(y)
            n_samples = X.shape[0]
            F = np.full(n_samples, self.init_score)
            self.trees_ = []

            for m in range(self.n_estimators):
                p = expit(F)
                grad = p - y
                hess = p * (1 - p)
                tree = self._build_tree(X, grad, hess, depth=0)
                self.trees_.append(tree)
                update = np.array([self._predict_tree(x, tree)
for x in X])
                F = F + self.learning_rate * update

            return self

        def predict_proba(self, X):
            X = np.array(X)
            F = np.full(X.shape[0], self.init_score)
```

Продовження 5 Лістингу А.9

```
            for tree in self.trees_:
                update = np.array([self._predict_tree(x, tree)
for x in X])
                F = F + self.learning_rate * update
            proba = expit(F)
            return np.vstack([1 - proba, proba]).T

        def predict(self, X):
            proba = self.predict_proba(X)[:, 1]
            return (proba >= 0.5).astype(int)

        def score(self, X, y):
            from sklearn.metrics import accuracy_score
            return accuracy_score(y, self.predict(X))
```

Лістинг А.10 — Файл cross_val_score.py

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold, StratifiedKFold
import numpy as np



    def cross_val_score(model, X, y, cv=5,
scoring=accuracy_score):
        if hasattr(X, 'values'):
            X = X.values
        if hasattr(y, 'values'):
            y = y.values

        scores = []
```

Продовження 1 Лістингу А.10

```
    cv_splitter = StratifiedKFold(n_splits=cv,
shuffle=True, random_state=42)


    for train_index, test_index in cv_splitter.split(X, y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        try:
            score = scoring(y_test, y_pred)
        except TypeError:
            if hasattr(scoring, '_score_func'):
                score = scoring._score_func(y_test, y_pred)
            else:
                raise
        scores.append(score)

    return np.array(scores)



def concordance_index(durations, events, scores):
    durations = np.asarray(durations)
    events = np.asarray(events)
    scores = np.asarray(scores)
```

Продовження 2 Лістингу А.10

```
        n = 0
        concordant = 0
        for i in range(len(durations)):
            for j in range(len(durations)):
                if durations[i] < durations[j] and events[i] ==
1:
                    n += 1
                    if scores[i] > scores[j]:
                        concordant += 1
        return concordant / n if n > 0 else np.nan



    def cross_val_cox_score(model, X, durations, events, cv=5,
random_state=42):
        X_arr = X.values if hasattr(X, "values") else
np.asarray(X)
        durations = np.asarray(durations)
        events = np.asarray(events)


        kf = KFold(n_splits=cv, shuffle=True,
random_state=random_state)
        scores = []


        for train_idx, test_idx in kf.split(X_arr):
            X_train, X_test = X_arr[train_idx], X_arr[test_idx]
            d_train, d_test = durations[train_idx],
durations[test_idx]
            e_train, e_test = events[train_idx],
events[test_idx]


            model.fit(X_train, d_train, e_train)
```

Продовження 3 Лістингу А.10

```
            hazards = model.predict_partial_hazard(X_test)
             ci = concordance_index(d_test, e_test, hazards)
             scores.append(ci)


        return np.array(scores)
```

Лістинг А.11 — Модель Client

```
from sqlalchemy import (
    Column, Integer, Float, String, DateTime, func
)
from .database import Base


class Client(Base):
    __tablename__ = "clients"


    clientnum               = Column(Integer,
primary_key=True, index=True)
    attrition_flag          = Column(String,
nullable=False)
    customer_age            = Column(Integer,
nullable=False)
    gender                  = Column(String,
nullable=False)
    dependent_count         = Column(Integer,
nullable=False)
    education_level         = Column(String,
nullable=False)
    marital_status          = Column(String,
nullable=False)
    income_category         = Column(String,
nullable=False)
    card_category           = Column(String,
nullable=False)
```

Продовження 1 Лістингу А.11

```
        months_on_book          = Column(Integer,
    nullable=False)

        total_relationship_count= Column(Integer,
nullable=False)

        months_inactive_12_mon  = Column(Integer,
    nullable=False)

        contacts_count_12_mon   = Column(Integer,
nullable=False)

        credit_limit            = Column(Float,
nullable=False)

        total_revolving_bal     = Column(Float,
nullable=False)

        avg_open_to_buy         = Column(Float,
nullable=False)

        total_amt_chng_q4_q1    = Column(Float,
nullable=False)

        total_trans_amt         = Column(Float,
nullable=False)

        total_trans_ct          = Column(Integer,
nullable=False)

        total_ct_chng_q4_q1     = Column(Float,
nullable=False)

        avg_utilization_ratio   = Column(Float,
nullable=False)

        account_open_date      = Column(DateTime,
nullable=False)


        created_at = Column(

            DateTime(timezone=True), server_default=func.now(),
nullable=False

        )

        updated_at = Column(

            DateTime(timezone=True),

            server_default=func.now(),

            onupdate=func.now(),
```

```
            nullable=False
        )
```

Лістинг А.12 — Роути /clients

```
    from fastapi import APIRouter, Depends, HTTPException,
status
    from typing import List
    from sqlalchemy.orm import Session
    from .. import crud, schemas
    from ..database import get_db


    router = APIRouter(prefix="/clients", tags=["clients"])


    @router.post("/", response_model=schemas.ClientRead,
status_code=status.HTTP_201_CREATED)
    def create_client(
        client: schemas.ClientCreate,
        db: Session = Depends(get_db)
    ):
        return crud.create_client(db, client)


    @router.get("/", response_model=List[schemas.ClientRead])
    def read_clients(skip: int = 0, limit: int = 100, db:
Session = Depends(get_db)):
        return crud.get_clients(db, skip, limit)


    @router.get("/{client_id}",
response_model=schemas.ClientRead)
    def read_client(client_id: int, db: Session =
Depends(get_db)):
        db_client = crud.get_client(db, client_id)
        if not db_client:
```

Продовження 1 Лістингу А.12

```
            raise HTTPException(status_code=404, detail="Client
not found")

        return db_client


    @router.put("/{client_id}",
response_model=schemas.ClientRead)

    def update_client(client_id: int, client_up:
schemas.ClientUpdate, db: Session = Depends(get_db)):

        db_client = crud.get_client(db, client_id)

        if not db_client:

            raise HTTPException(status_code=404, detail="Client
not found")

        return crud.update_client(db, db_client, client_up)


    @router.delete("/{client_id}",
status_code=status.HTTP_204_NO_CONTENT)

    def delete_client(client_id: int, db: Session =
Depends(get_db)):

        db_client = crud.get_client(db, client_id)

        if not db_client:

            raise HTTPException(status_code=404, detail="Client
not found")

        crud.delete_client(db, db_client)
```

Лістинг А.13 — Роути /predict

```
    from fastapi import APIRouter, Depends, HTTPException,
Query

    from sqlalchemy.orm import Session

    from ..schemas import SurvivalPredictRequest,
SurvivalPredictResponse

    from ..config import settings

    import joblib

    import pandas as pd
```

Продовження 1 Лістингу А.13

```
from datetime import date


from ..database import get_db
from ..models import Client
router = APIRouter(prefix="/predict", tags=["predict"])
cox_pipe = joblib.load(settings.COX_PIPELINE_PATH)


@router.get(
    "/by-id/{client_id}",
    response_model=SurvivalPredictResponse,
    summary="Прогноз виживання клієнта за ID і опційними
датами"
)
def predict_by_id(
    client_id: int,
    start_date: date | None = Query(
        None,
        description="Start ISO-date (YYYY-MM-DD)"
    ),
    end_date:   date | None = Query(
        None,
        description="End ISO-date (YYYY-MM-DD)"
    ),
    db: Session = Depends(get_db),
):
    client = db.query(Client).filter(Client.clientnum ==
client_id).first()
    if not client:
        raise HTTPException(status_code=404, detail="Client
not found")
```

Продовження 2 Лістингу А.13

```
        open_dt = client.account_open_date
        if hasattr(open_dt, "date"):
            open_date = open_dt.date()

        else:
            open_date = open_dt

        payload = {
            "Customer_Age": client.customer_age,
            "Gender": client.gender,
            "Dependent_count": client.dependent_count,
            "Education_Level": client.education_level,
            "Marital_Status": client.marital_status,
            "Income_Category": client.income_category,
            "Card_Category": client.card_category,
            "Credit_Limit": client.credit_limit,
            "Months_on_book": client.months_on_book,
            "Total_Relationship_Count":
client.total_relationship_count,
            "Months_Inactive_12_mon":
client.months_inactive_12_mon,
            "Contacts_Count_12_mon":
client.contacts_count_12_mon,
            "Total_Revolving_Bal": client.total_revolving_bal,
            "Avg_Open_To_Buy": client.avg_open_to_buy,
            "Total_Amt_Chng_Q4_Q1":
client.total_amt_chng_q4_q1,
            "Total_Trans_Amt": client.total_trans_amt,
            "Total_Trans_Ct": client.total_trans_ct,
            "Total_Ct_Chng_Q4_Q1": client.total_ct_chng_q4_q1,
```

Продовження 3 Лістингу А.13

```
            "Avg_Utilization_Ratio":
client.avg_utilization_ratio,

        }

        df = pd.DataFrame([payload])


        X_t =
    cox_pipe.named_steps["preprocessor"].transform(df)

        surv_df =
cox_pipe.named_steps["cox"].model.predict_survival_function(X_t)


        timeline = surv_df.index.to_numpy()           # місяці
від 0,1,2,...

        prob_churn = (1 - surv_df.iloc[:, 0].to_numpy())


        def to_months(d: date) -> float:

            return (d - open_date).days / 30.0


        if start_date:

            start_m = to_months(start_date)

            mask = timeline >= start_m

            timeline   = timeline[mask]

            prob_churn = prob_churn[mask]


        if end_date:

            end_m = to_months(end_date)

            mask = timeline <= end_m

            timeline   = timeline[mask]

            prob_churn = prob_churn[mask]


        return SurvivalPredictResponse(

            timeline=[float(m) for m in timeline],
```

Продовження 4 Лістингу А.13

```
            churn_probability=[float(p) for p in prob_churn]
        )
```

Лістинг А.14 — Dockerfile

```
FROM python:3.10-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

ENV PYTHONPATH=/app

EXPOSE 8000

CMD ["uvicorn", "api.main:app", "--host", "0.0.0.0",
"--port", "8000"]
```

Лістинг А.15 — docker-compose

```
version: "3.8"

services:
  db:
    image: postgres:15
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
```

Продовження 1 Лістингу А.15

```
      POSTGRES_DB: bank
    ports:

     - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data
      - ./db/init:/docker-entrypoint-initdb.d
    networks:
      - app-network

  api:
    build: .
    depends_on:
      - db
    ports:
      - "8000:8000"
    env_file:
      - .env
    restart: always
    networks:
      - app-network

volumes:
  pgdata:

networks:
  app-network:
    driver: bridge
```