Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «бакалавр»

на тему:

ПРОГРАМНА СИСТЕМА ЗАБЕЗПЕЧЕННЯ ІНЖЕНЕРІЇ ДАНИХ ДЛЯ МАШИННОГО НАВЧАННЯ. ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ТА НЕЙРОННІ МЕРЕЖІ

Виконав:
здобувач вищої освіти 4 курсу
групи ІПЗ-41
спеціальності 121 Інженерія програмного забезпечення
Кушнірук Максим Васильович

Науковий керівник:
к.т.н., доцент Сяський В. А.

Рівне – 2026

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – програмний інтерфейс застосунку.

REST API – інтерфейс взаємодії між frontend- і backend-частинами системи.

ETL – процес вилучення, перетворення та завантаження даних.

ML – машинне навчання.

CSV – формат зберігання табличних даних.

JSON – формат обміну структурованими даними.

Frontend – клієнтська частина програмної системи.

Backend – серверна частина програмної системи.

FastAPI – фреймворк Python для створення REST API.

Docker – платформа для контейнеризації програмних застосунків.

PostgreSQL – система керування реляційними базами даних.

RandomForest – ансамблевий алгоритм машинного навчання на основі дерев рішень.

Neural Network – нейронна мережа.

Accuracy – метрика загальної точності моделі.

Precision – метрика точності позитивних прогнозів.

Recall – метрика повноти виявлення позитивного класу.

F1-score – метрика балансу між precision і recall.

Confusion matrix – матриця помилок класифікації.

ВСТУП

Актуальність теми кваліфікаційної роботи зумовлена потребою в розробленні програмних засобів, які поєднують підготовку даних, навчання моделей машинного навчання, прогнозування та оцінювання якості результатів у межах єдиної системи. На практиці ці процеси часто виконуються окремими інструментами або скриптами, що ускладнює роботу користувача та підвищує ризик помилок. Тому доцільним є створення локального веб-застосунку, який дозволяє запускати ETL-процес, навчати моделі, виконувати прогнозування та переглядати результати через веб-інтерфейс.

У роботі розглядається програмна система забезпечення інженерії даних для машинного навчання. Як демонстраційний набір даних використано датасет Titanic, на основі якого розв’язується задача бінарної класифікації – прогнозування виживання пасажира за набором ознак. Система реалізується як локальний веб-застосунок із backend-частиною на основі Python і FastAPI та frontend-частиною, створеною з використанням HTML, CSS і JavaScript. Програмний продукт передбачає завантаження й оброблення CSV-файлів, виконання ETL-процесу, навчання моделей RandomForest і Neural Network, одиничне та batch-прогнозування, розрахунок метрик якості й побудову confusion matrix.

Метою роботи є розроблення програмної системи забезпечення інженерії даних для машинного навчання, яка дозволяє виконувати підготовку даних, навчання моделей, прогнозування та оцінювання якості результатів у межах єдиного локального веб-застосунку.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. проаналізувати предметну область інженерії даних і машинного навчання;
2. розглянути роль ETL-процесів у підготовці даних;
3. дослідити методи машинного навчання для задачі класифікації;
4. спроєктувати архітектуру локального веб-застосунку;
5. реалізувати backend-частину, REST API, ETL-модуль, ML-модуль і frontend-інтерфейс;

6. виконати тестування системи та оцінити якість її роботи.

Об’єктом дослідження є процеси інженерії даних і машинного навчання в програмних системах інтелектуального аналізу даних.

Предметом дослідження є методи, моделі та програмні засоби реалізації локального веб-застосунку для підготовки даних, навчання моделей, прогнозування та оцінювання якості результатів.

У роботі використано такі методи дослідження: аналіз літературних і технічних джерел, структурний аналіз, модульне програмування, ETL-обробку, методи машинного навчання, тестування програмного забезпечення та оцінювання якості моделей за метриками accuracy, precision, recall і f1-score [5-9].

Практичне значення роботи полягає в розробленні локального веб-застосунку, який демонструє повний цикл роботи з даними в задачах машинного навчання: від завантаження та попередньої обробки CSV-файлу до навчання моделей, прогнозування й оцінювання якості результатів.

Бакалаврська робота складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел і додатків. У першому розділі розглянуто постановку задачі та аналіз предметної області, у другому – проєктування архітектури, у третьому – реалізацію програмного продукту, у четвертому – тестування та оцінку якості системи.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття інженерії даних у системах машинного навчання

Інженерія даних є важливим напрямом розроблення програмних систем, пов’язаних з аналізом, обробленням і використанням інформації. Її основне призначення полягає в організації процесів збирання, очищення, перетворення та підготовки даних для подальшого застосування в аналітичних або інтелектуальних системах. У сфері машинного навчання інженерія даних має особливе значення, оскільки якість підготовленого набору безпосередньо впливає на точність, стабільність і практичну придатність побудованих моделей [1-3; 24].

Процес роботи з даними в системах машинного навчання включає отримання, очищення, перетворення та підготовку набору даних для подальшого навчання моделей і прогнозування результатів.

Загальне місце інженерії даних у цьому процесі подано на рис. 1.1.

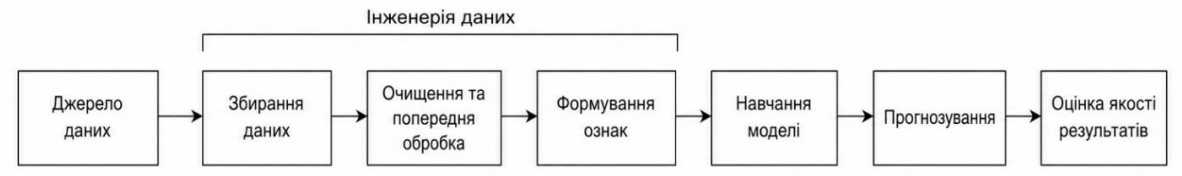


Рисунок 1.1 – Місце інженерії даних у процесі машинного навчання

Окремим елементом інженерії даних є підготовка ознак, або feature engineering. Вона передбачає створення нових змінних на основі наявних даних, що може підвищити інформативність датасету та якість роботи моделі. У задачах класифікації правильно сформовані ознаки допомагають алгоритму точніше розділяти об’єкти за класами [3].

Важливою вимогою до інженерії даних є повторюваність процесу обробки. У програмній системі недостатньо один раз вручну очистити набір даних. Необхідно реалізувати механізм, який дозволяє повторно виконувати однакову

послідовність дій для нових або оновлених даних. Саме тому в практичних рішеннях використовують окремі модулі або пайплайни обробки даних.

Для автоматизації підготовки даних у системах машинного навчання використовують ETL-модулі та механізми формування ознак.

ETL-процеси є важливою складовою інженерії даних і використовуються для послідовної підготовки інформації перед її застосуванням у моделях машинного навчання. Абревіатура ETL означає три основні етапи: Extract, Transform, Load, тобто вилучення, перетворення та завантаження даних. У системах машинного навчання ці етапи дозволяють перетворити початкові дані на структурований набір, придатний для навчання моделей, прогнозування та оцінювання результатів [1-3; 24].

На рис. 1.2 подано узагальнену схему ETL-процесу підготовки даних для машинного навчання

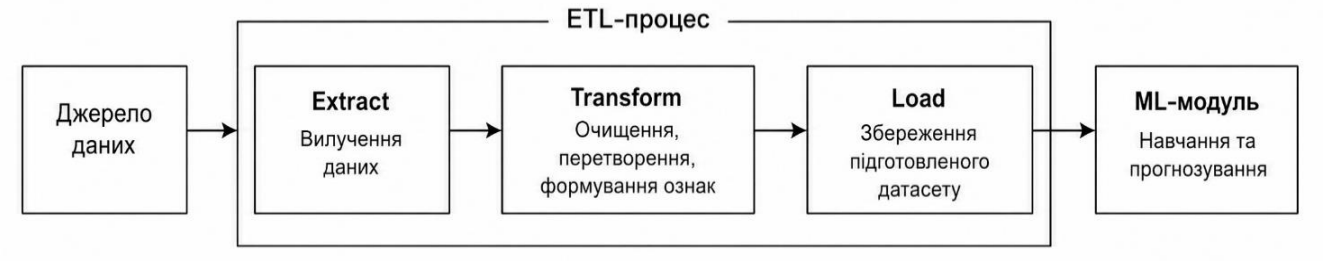


Рисунок 1.2 – Узагальнена схема ETL-процесу

Машинне навчання є напрямом штучного інтелекту, що передбачає створення моделей, здатних виявляти закономірності в даних і використовувати їх для прогнозування або класифікації нових об’єктів. На відміну від класичного програмування, де правила задаються безпосередньо розробником, у машинному навчанні модель формує залежності на основі навчального набору даних [3].

У межах аналізу розглядається задача бінарної класифікації. Її сутність полягає у віднесенні об’єкта до одного з двох наперед визначених класів. У практичній частині роботи використовується датасет Titanic, на основі якого виконується прогнозування виживання пасажира за набором ознак. До таких ознак

належать стать, вік, вартість квитка, розмір сім’ї, титул, вікова група та порт посадки.

Таким чином, інженерія даних і методи машинного навчання є взаємопов’язаними складовими сучасних інформаційних систем аналізу даних. Підготовка та обробка даних забезпечують формування якісного набору ознак, необхідного для ефективного навчання моделей класифікації та прогнозування результатів.

Для реалізації задач машинного навчання використовуються різні програмні платформи та інструменти, які забезпечують роботу з даними, навчання моделей і оцінювання результатів.

1.2 Огляд існуючих програмних рішень для обробки даних і навчання моделей

Для розроблення програмної системи забезпечення інженерії даних для машинного навчання доцільно проаналізувати наявні інструменти, які використовуються для обробки даних, побудови моделей, автоматизації пайплайнів і візуалізації результатів. Такі рішення можуть бути орієнтовані на інтерактивну роботу з кодом, графічне створення аналітичних процесів, автоматизацію обробки даних або керування ML-експериментами [1-3; 24].

Одним із найпоширеніших середовищ для роботи з даними є Jupyter Notebook. Він дозволяє поєднувати програмний код, текстові пояснення, таблиці, графіки та результати виконання команд в одному документі. Google Colab має схоже призначення, але працює у хмарному середовищі та не потребує локального налаштування Python і бібліотек. Водночас ці інструменти більше підходять для дослідження даних і експериментів, ніж для створення повноцінного локального веб-застосунку з REST API та окремим інтерфейсом користувача.

Для візуального створення процесів обробки даних можуть використовуватися KNIME і RapidMiner. Вони надають графічний інтерфейс для побудови аналітичних пайплайнів, очищення даних, навчання моделей і перегляду

результатів. Такі платформи зручні для користувачів без глибоких навичок програмування, однак мають обмежену гнучкість у випадках, коли потрібно реалізувати власну backend-логіку, REST API або спеціалізований веб-інтерфейс.

Для автоматизації складних пайплайнів даних застосовується Apache Airflow. Він дає змогу планувати, запускати та контролювати послідовності задач, однак для невеликого локального навчального застосунку є надмірно складним. Для керування ML-експериментами може використовуватися MLflow, який дозволяє зберігати параметри моделей, метрики та результати навчання. Проте MLflow не замінює систему, у якій користувач може завантажувати дані, запускати ETL-процес, виконувати прогнозування та переглядати результати через веб-інтерфейс.

Порівняння існуючих програмних рішень для інженерії даних і машинного навчання наведено в табл. 1.1.

Таблиця 1.1 – Порівняння існуючих рішень для інженерії даних і машинного навчання

Рішення	Призначення	Переваги	Обмеження
Jupyter Notebook	Інтерактивна робота з Python-кодом і даними	Зручний для аналізу, візуалізації та швидкого тестування моделей	Не є готовим веб-застосунком
Google Colab	Хмарне середовище для запуску Python-коду	Не потребує локального налаштування, зручний для навчання й експериментів	Залежить від інтернету та має обмеження середовища
KNIME	Візуальна платформа для аналітики	Дає змогу будувати пайплайни без активного програмування	Менш гнучкий для власного REST API

Продовження таблиці 1.1

RapidMiner	Платформа для аналізу даних і ML	Зручна для швидкого створення ML-процесів	Обмежена кастомізація під власний застосунок
Apache Airflow	Автоматизація пайплайнів даних	Підходить для складних промислових процесів	Надмірний для невеликої локальної системи
MLflow	Керування ML-експериментами	Зберігає параметри, метрики й моделі	Не забезпечує повний веб-застосунок
Розроблювана система	Локальний веб-застосунок для ETL, ML і прогнозування	Поєднує обробку даних, навчання моделей, прогнозування й оцінювання	Орієнтована на навчальне та демонстраційне використання

Як видно з табл. 1.1, більшість наявних рішень ефективно виконує окремі функції, але не повністю відповідає задачі створення локального веб-застосунку, який поєднує підготовку даних, навчання моделей, прогнозування та оцінювання якості через єдиний інтерфейс [1-3; 11].

Розроблювана програмна система відрізняється тим, що об’єднує основні етапи роботи з даними в межах одного застосунку. Користувач може запустити ETL-процес, навчити модель, виконати одиничне або batch-прогнозування, а також переглянути результати оцінювання якості.

Таким чином, аналіз існуючих рішень показує доцільність розроблення власного локального веб-застосунку. Він має поєднувати ETL-обробку, навчання моделей, прогнозування та оцінювання результатів, що відповідає поставленій задачі бакалаврської роботи.

1.3 Постановка задачі розроблення програмної системи

На основі аналізу предметної області, ETL-процесів, методів машинного навчання та існуючих програмних рішень сформульовано задачу розроблення програмної системи забезпечення інженерії даних для машинного навчання. Необхідність створення такої системи зумовлена тим, що підготовка даних, навчання моделей, прогнозування та оцінювання якості часто виконуються окремо, що ускладнює роботу користувача й підвищує ризик помилок.

Розроблювана система має бути реалізована як локальний веб-застосунок, що забезпечує повний цикл роботи з даними: завантаження CSV-файлу, попередню обробку, формування ознак, навчання моделей машинного навчання, виконання прогнозування та оцінювання якості результатів. Такий підхід дозволяє об’єднати основні етапи ML-процесу в межах єдиного програмного продукту.

У межах практичної реалізації як вхідний набір даних використовується датасет Titanic. Цільовою змінною є ознака виживання пасажира, а вхідними ознаками виступають стать, вік, вартість квитка, розмір сім’ї, ознака самостійної подорожі, титул, вікова група та порт посадки. Такий набір даних дозволяє продемонструвати роботу ETL-модуля, формування ознак, навчання моделей класифікації та виконання прогнозування.

Основною метою розроблюваної системи є надання користувачу можливості виконувати підготовку даних і роботу з моделями машинного навчання через веб-інтерфейс без ручного запуску окремих Python-скриптів. Система має забезпечувати взаємодію між frontend-частиною, backend-сервером, ETL-модулем, ML-модулем, модулем оцінювання якості та базою даних.

До основних функціональних вимог до програмної системи належать:

1. завантаження та обробка CSV-файлу;
2. запуск ETL-процесу через веб-інтерфейс або REST API;
3. очищення даних і формування додаткових ознак;
4. навчання моделей RandomForest і Neural Network;
5. прогнозування для одного запису;
6. batch-прогнозування на основі CSV-файлу;
7. розрахунок метрик accuracy, precision, recall і f1-score;

8. побудова confusion matrix;
9. відображення результатів у веб-інтерфейсі.

До нефункціональних вимог належать простота локального запуску, модульність архітектури, зрозумілість веб-інтерфейсу, можливість тестування окремих компонентів, повторюваність ETL-процесу та використання контейнеризованого середовища для розгортання системи.

Очікуваним результатом розроблення є локальний веб-застосунок, який дозволяє виконувати основні етапи роботи з даними в задачі машинного навчання. Користувач повинен мати можливість запустити ETL-обробку, навчити одну з доступних моделей, виконати одиничне або групове прогнозування, переглянути метрики якості та отримати візуальне подання confusion matrix.

Таким чином, постановка задачі передбачає створення цілісної програмної системи, яка поєднує підготовку даних, машинне навчання, прогнозування, оцінювання якості та веб-інтерфейс користувача. Такий підхід відповідає темі бакалаврської роботи та дозволяє продемонструвати практичне застосування інженерії даних у системах машинного навчання.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМНОЇ СИСТЕМИ

2.1 Загальна архітектура локального веб-застосунку

Проєктування архітектури програмної системи є важливим етапом розроблення, оскільки саме на цьому рівні визначається загальна структура застосунку, взаємодія його компонентів, розподіл функцій між модулями та спосіб обміну даними. Для програмної системи забезпечення інженерії даних для машинного навчання доцільним є використання клієнт-серверної архітектури, яка дозволяє відокремити користувацький інтерфейс від серверної логіки обробки даних і навчання моделей.

Розроблювана система реалізується як локальний веб-застосунок. Користувач взаємодіє із системою через браузер, а основна логіка роботи виконується на локальному сервері. Такий підхід є зручним для навчального й демонстраційного програмного продукту, оскільки не потребує зовнішнього хостингу, але дозволяє реалізувати повноцінну взаємодію між frontend-частиною, backend-частиною, модулями машинного навчання та базою даних.

Загальна архітектура системи передбачає наявність кількох основних компонентів: веб-інтерфейсу користувача, серверної частини на основі REST API, ETL-модуля, модуля машинного навчання, модуля оцінювання якості, бази даних і середовища локального розгортання. Кожен компонент виконує окрему функцію, але разом вони забезпечують повний цикл роботи з даними: від завантаження й обробки CSV-файлу до навчання моделей, прогнозування та аналізу результатів.

Frontend-частина відповідає за взаємодію користувача із системою. Через веб-інтерфейс користувач може запускати ETL-процес, обирати модель для навчання, вводити дані для одиничного прогнозування, виконувати batch-обробку та переглядати результати роботи системи.

Backend-частина виконує роль центрального компонента системи. Вона приймає запити від frontend-частини, перевіряє вхідні дані, запускає відповідні програмні модулі та повертає результати користувачу. Для реалізації backend-

частини використовується FastAPI, який дає змогу створювати REST API, описувати маршрути, виконувати валідацію даних і формувати автоматичну документацію API [11; 12; 13].

ETL-модуль відповідає за підготовку вхідних даних до використання в моделях машинного навчання. На цьому етапі здійснюється завантаження CSV-файлу, очищення даних, обробка пропущених значень, формування додаткових ознак і підготовка датасету до навчання [1-3].

Модуль машинного навчання забезпечує навчання моделей RandomForest і Neural Network. Після отримання підготовленого набору даних система виконує навчання обраної моделі, формує прогнози та передає результати для оцінювання якості [3; 4; 6].

Окремим компонентом є модуль оцінювання якості моделей. Він відповідає за розрахунок метрик accuracy, precision, recall і f1-score, а також за побудову confusion matrix. Завдяки цьому користувач може оцінити не лише факт отримання прогнозу, а й якість роботи моделі [7-9].

База даних PostgreSQL використовується для збереження підготовлених даних після виконання ETL-процесу [17; 18]. Для локального розгортання системи застосовується Docker, який дозволяє ізолювати середовище виконання, налаштувати залежності та забезпечити однаковий запуск системи на різних комп’ютерах [15; 16].

Загальну архітектуру локального веб-застосунку подано на рис. 2.1.

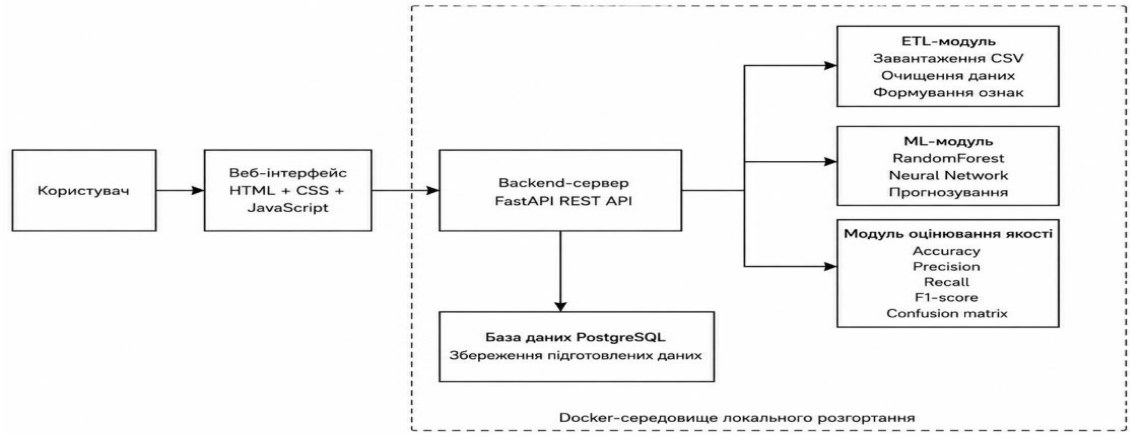


Рисунок 2.1 – Загальна архітектура локального веб-застосунку

Як видно з рис. 2.1, користувач взаємодіє із системою через веб-інтерфейс, який надсилає запити до backend-частини. Серверна частина обробляє запити, запускає ETL-модуль, модулі машинного навчання та оцінювання якості, а також забезпечує обмін даними з базою PostgreSQL. Така структура дозволяє розділити функціональність системи на окремі компоненти й забезпечити зрозумілу логіку взаємодії між ними.

Запропонована архітектура має кілька переваг. По-перше, вона забезпечує модульність, оскільки кожен компонент виконує окрему функцію. По-друге, така структура спрощує тестування, адже можна окремо перевіряти роботу API, ETL-модуля, моделей машинного навчання та веб-інтерфейсу. По-третє, використання REST API дає змогу в майбутньому розширити систему, наприклад додати нові моделі, інші джерела даних або складніший frontend.

Таким чином, загальна архітектура розроблюваної програмної системи базується на клієнт-серверному підході та поєднує frontend-інтерфейс, backend на основі FastAPI, ETL-модуль, модулі машинного навчання, засоби оцінювання якості, базу даних PostgreSQL і Docker-середовище. Така архітектура відповідає поставленій задачі, оскільки дозволяє реалізувати повний цикл роботи з даними в межах одного локального веб-застосунку.

2.2 Обґрунтування вибору Python, FastAPI, HTML, CSS та JavaScript

Вибір технологій розроблення є важливим етапом проєктування програмної системи, оскільки від нього залежать структура застосунку, зручність реалізації функціональних можливостей, підтримка бібліотек машинного навчання та можливість подальшого розширення. Для створення програмної системи забезпечення інженерії даних для машинного навчання було обрано технології, які дозволяють реалізувати обробку даних, навчання моделей, REST API, веб-інтерфейс і локальне розгортання.

Основною мовою програмування для серверної частини системи обрано Python. Такий вибір зумовлений тим, що Python широко використовується в

задачах аналізу даних, машинного навчання та автоматизації обробки інформації[25]. У межах розроблюваної системи Python використовується для реалізації ETL-модуля, навчання моделей RandomForest і Neural Network, обробки API-запитів і формування результатів прогнозування [3; 5; 19].

Для роботи з табличними даними використовується бібліотека pandas, яка забезпечує зчитування CSV-файлів, очищення даних, зміну типів, обробку пропусків і формування підготовлених датасетів [19: 25]. Для реалізації класичної моделі машинного навчання застосовується scikit-learn[25; 28]. Ця бібліотека містить готові алгоритми класифікації, інструменти попередньої обробки даних і засоби оцінювання якості моделей [5; 7]. У роботі scikit-learn використовується для навчання RandomForest, стандартизації ознак і розрахунку метрик accuracy, precision, recall та f1-score [6; 7; 8; 10; 25].

Для реалізації нейронної мережі використовується TensorFlow/Keras[25; 26]. Цей інструментарій дозволяє створювати послідовні моделі нейронних мереж, налаштовувати шари, функції активації, оптимізатори та функції втрат [20]. У системі TensorFlow/Keras застосовується для побудови Neural Network як альтернативного підходу до задачі класифікації.

Для створення backend-частини обрано FastAPI. Цей фреймворк призначений для розроблення веб API мовою Python і підтримує типізацію, автоматичну валідацію даних та формування OpenAPI-документації [11; 12; 13]. FastAPI добре підходить для системи, у якій frontend-частина надсилає HTTP-запити для запуску ETL-процесу, навчання моделей, прогнозування та отримання результатів оцінювання якості.

Для опису структури вхідних даних і валідації запитів використовується Pydantic. Це дозволяє перевіряти відповідність отриманих даних очікуваним типам і зменшує ризик помилок під час обробки API-запитів [23].

Frontend-частина реалізується з використанням HTML, CSS і JavaScript. HTML забезпечує структуру веб-сторінки, форми введення даних, кнопки запуску операцій і блоки відображення результатів. JavaScript використовується для взаємодії з REST API, надсилання асинхронних запитів до backend-сервера та

оновлення результатів без повного перезавантаження сторінки. CSS був використанй для оформлення інтерфейсу користувача, що робить веб-застосунок більш зручним у використанні.

Для збереження підготовлених даних використовується PostgreSQL. Ця система керування базами даних підтримує роботу зі структурованими даними та може застосовуватися як локальне сховище результатів ETL-обробки [17; 18]. Для локального розгортання системи обрано Docker, який дозволяє ізолювати середовище виконання, зафіксувати залежності та забезпечити однаковий запуск застосунку на різних комп’ютерах [15; 16].

Таблиця 2.1 – Обґрунтування вибору технологій розроблення

Технологія	Призначення в системі	Причина вибору
Python	Реалізація backend-логіки, ETL-модуля та ML-модулів	Має розвинену екосистему бібліотек для аналізу даних, машинного навчання та створення API
pandas	Завантаження й обробка CSV-даних	Забезпечує зручну роботу з табличними даними, очищенням, перетворенням і підготовкою датасетів
scikit-learn	Реалізація RandomForest, попередня обробка та метрики якості	Містить готові алгоритми машинного навчання, засоби стандартизації та оцінювання моделей
TensorFlow/Keras	Реалізація нейронної мережі	Дозволяє створювати й навчати нейронні мережі з використанням послідовної архітектури шарів

Продовження таблиці 2.1

FastAPI	Розроблення REST API backend-частини	Підтримує типізацію, валідацію даних, швидке створення API та автоматичну документацію
Pydantic	Валідація вхідних даних	Дає змогу перевіряти структуру й типи даних, які надходять від користувача через API
HTML	Структура веб-інтерфейсу	Дозволяє створити форми, кнопки та блоки відображення результатів у браузері
CSS	Оформлення інтерфейсу	Використовується для стилізації елементів
JavaScript	Взаємодія frontend-частини з REST API	Забезпечує надсилання асинхронних запитів і динамічне оновлення результатів на сторінці
PostgreSQL	Збереження підготовлених даних	Підтримує роботу зі структурованими даними та може використовуватися як надійне локальне сховище
Docker	Локальне розгортання системи	Ізолює середовище виконання, спрощує запуск і забезпечує однакову роботу застосунку в різних середовищах

Як видно з табл. 2.1, обраний технологічний стек відповідає функціональним потребам системи. Python забезпечує реалізацію обробки даних і машинного навчання, FastAPI відповідає за серверну взаємодію через REST API, HTML, CSS і JavaScript формують веб-інтерфейс, PostgreSQL використовується для збереження структурованих даних, а Docker спрощує локальне розгортання.

Таким чином, вибір даних технологій є обґрунтованим для розроблення програмної системи забезпечення інженерії даних для машинного навчання.

Обраний стек дозволяє реалізувати всі ключові функції застосунку, підтримати роботу з CSV-даними, виконати навчання моделей і надати користувачу веб-інтерфейс для взаємодії з системою.

2.3 Архітектура веб-застосунку та взаємодія компонентів

Архітектура розроблюваної програмної системи побудована за клієнт-серверним принципом[27]. Основними компонентами системи є frontend-частина, backend-частина, REST API, ETL-модуль, модуль машинного навчання та база даних PostgreSQL. Такий підхід дозволяє розділити інтерфейс користувача, логіку обробки запитів, підготовку даних, навчання моделей і збереження результатів.

Backend-частина є центральним компонентом системи, оскільки саме вона забезпечує обробку запитів користувача, запуск функціональних модулів, взаємодію з даними та повернення результатів у веб-інтерфейс. У межах системи backend реалізується на основі FastAPI, що дозволяє створювати REST API, виконувати перевірку вхідних даних і формувати автоматичну документацію API [11-13].

Frontend-частина призначена для взаємодії користувача з основними функціями локального веб-застосунку. Вона реалізується з використанням HTML, CSS і JavaScript. HTML відповідає за структуру сторінки, форми введення даних, кнопки запуску операцій і блоки відображення результатів, CSS стилізує веб-інтерфейс а JavaScript забезпечує надсилання запитів до backend-частини через REST API.

Схему взаємодії frontend, backend та ML-модуля подано на рис. 2.2.

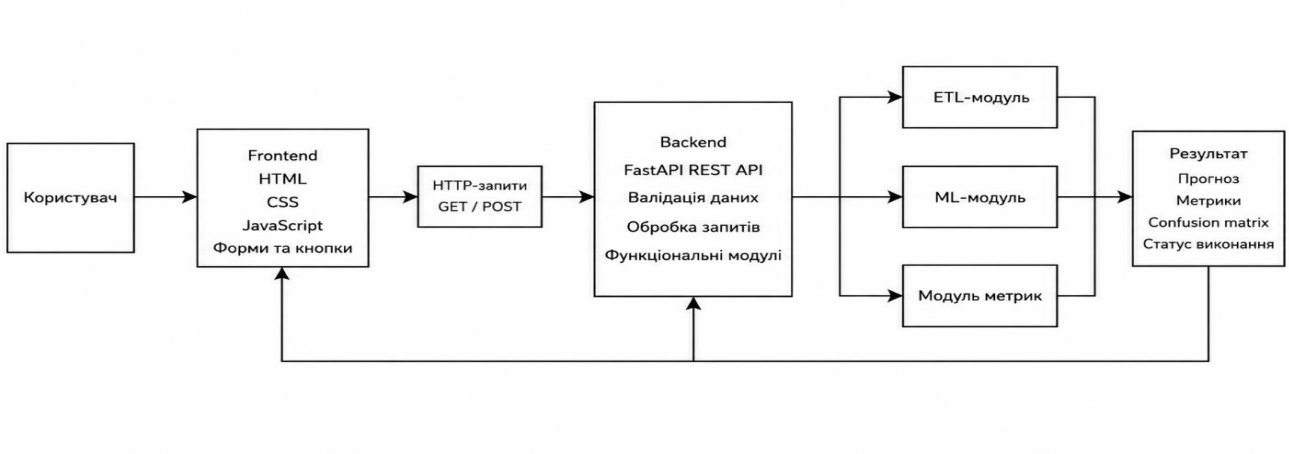


Рисунок 2.2 – Схема взаємодії frontend, backend та ML-модуля

Як видно з рис. 2.2, користувач працює із системою через веб-інтерфейс. Frontend надсилає HTTP-запити до backend-сервера, який викликає відповідний функціональний модуль: ETL-обробку, навчання моделі, прогнозування або формування результатів оцінювання якості. Після виконання операції результат повертається користувачу через веб-інтерфейс.

REST API використовується як основний механізм взаємодії між frontend- і backend-частинами. Через API користувач може запускати підготовку даних, навчання моделей, прогнозування одного запису, batch-прогнозування та перегляд результатів оцінювання. Для передавання даних між клієнтом і сервером використовується формат JSON, а для завантаження наборів даних – CSV-файл через форму веб-інтерфейсу [14].

Основні API-ендпоінти програмної системи наведено в табл. 2.2.

Таблиця 2.2 – Опис основних API-ендпоінтів системи

Ендпоінт	Метод	Призначення
/	GET	Відкриття головної сторінки веб-застосунку
/etl	POST	Запуск ETL-процесу та підготовка даних
/train	POST	Навчання обраної моделі машинного навчання
/predict	POST	Прогнозування для одного запису
/predict_batch_json	POST	Batch-прогнозування на основі JSON-масиву

Продовження таблиці 2.2

/predict_batch_file	POST	Batch-прогнозування на основі CSV-файлу
/metrics	GET	Отримання результатів оцінювання та confusion matrix

Ендпоінт /etl використовується для запуску процесу підготовки даних. Ендпоінт /train відповідає за навчання моделей RandomForest або Neural Network. Через /predict виконується прогнозування для одного запису, а ендпоінти /predict_batch_json і /predict_batch_file забезпечують групову обробку даних. Ендпоінт /metrics використовується для отримання метрик якості та confusion matrix.

Для перевірки структури вхідних запитів використовується Pydantic. Він дозволяє описати очікувані поля, типи значень і перевіряти дані перед передаванням їх до ETL- або ML-модуля [23]. Це зменшує ризик помилок під час навчання моделей і виконання прогнозування.

Веб-інтерфейс системи має односторінкову структуру, оскільки застосунок має навчально-демонстраційне призначення. У межах сторінки передбачено кілька функціональних блоків: запуск ETL-процесу, навчання моделі, прогнозування для одного запису, batch-прогнозування та перегляд метрик якості. Frontend-частина не виконує складної обробки даних самостійно, а лише передає запити до backend-сервера та відображає отримані результати.

Локальне розгортання системи передбачає використання Docker. Ця технологія дозволяє ізолювати середовище виконання, керувати залежностями та забезпечити однаковий запуск застосунку на різних комп’ютерах [15; 16]. У системі Docker використовується для запуску backend-сервера та бази даних PostgreSQL. Backend-сервер обробляє HTTP-запити, запускає ETL-модуль і модуль машинного навчання, а PostgreSQL використовується для збереження підготовлених даних [17; 18].

Схему локального розгортання системи в Docker подано на рис. 2.3.

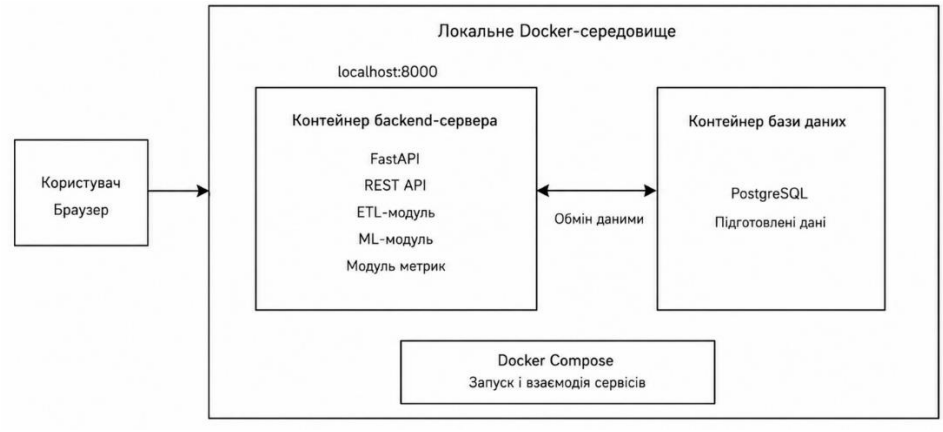


Рисунок 2.3 – Схема локального розгортання системи в Docker

Як видно з рис. 2.3, користувач звертається до локального веб-застосунку через браузер. Запити надходять до контейнера backend-сервера, у якому працює FastAPI та функціональні модулі системи. Окремо запускається контейнер PostgreSQL, який використовується для збереження підготовлених даних. Docker Compose забезпечує одночасний запуск цих сервісів і їхню взаємодію в межах локального середовища.

Таким чином, архітектура системи забезпечує розділення основних компонентів, зручну взаємодію між frontend- і backend-частинами, централізовану обробку запитів через REST API та повторюваний запуск застосунку в локальному середовищі Docker.

2.4 Проєктування ETL-модуля та модуля машинного навчання

ETL-модуль і модуль машинного навчання є основними функціональними частинами розроблюваної системи. ETL-модуль відповідає за завантаження, очищення, перетворення та підготовку даних, а ML-модуль забезпечує навчання моделей, виконання прогнозування та оцінювання якості результатів.

У розроблюваній системі основним форматом вхідних даних є CSV-файл. Такий формат є поширеним для зберігання табличних даних і зручним для навчальних задач. Для його обробки використовується Python, зокрема бібліотека

pandas, яка дозволяє зчитувати дані, аналізувати їхню структуру та виконувати подальші перетворення [19].

Процес роботи ETL-модуля включає кілька етапів. Спочатку система завантажує CSV-файл і перетворює його у внутрішню табличну структуру. Далі виконується перевірка даних: аналізуються наявні стовпці, типи значень, наявність цільової змінної та придатність набору для задачі класифікації.

Після цього здійснюється очищення даних. На цьому етапі видаляються дублікати, обробляються пропущені значення, усуваються некоректні записи та поля, які не мають аналітичної цінності. Якщо ці проблеми не усунути, модель може навчатися на неякісних даних і формувати неточні прогнози [3; 10].

Наступним етапом є перетворення ознак. Категоріальні змінні переводяться у числовий вигляд, числові поля за потреби масштабуються або нормалізуються, а також формуються додаткові ознаки. У межах роботи з датасетом Titanic це можуть бути ознаки, пов’язані з віком, статтю, вартістю квитка, розміром сім’ї, титулом пасажира, віковою групою та портом посадки.

Після завершення ETL-обробки формується підготовлений датасет. Він може бути збережений у вигляді окремого CSV-файлу або переданий до бази даних PostgreSQL. Збереження підготовлених даних дозволяє повторно використовувати їх для навчання різних моделей, тестування системи та аналізу результатів.

Модуль машинного навчання працює з підготовленими даними, отриманими після ETL-процесу. На цьому етапі дані вже мають бути приведені до формату, придатного для алгоритмів машинного навчання: пропущені значення оброблено, категоріальні ознаки перетворено у числовий вигляд, а числові поля за потреби масштабовано [3; 10].

У межах бакалаврської роботи модуль машинного навчання передбачає використання двох моделей: RandomForest і Neural Network. RandomForest є ансамблевим алгоритмом, який складається з множини дерев рішень і формує результат шляхом узагальнення їхніх прогнозів. Такий підхід добре підходить для табличних даних і зменшує ризик перенавчання порівняно з окремим деревом рішень [6].

Нейронна мережа використовується як альтернативний підхід до задачі класифікації. Вона складається з послідовності шарів, які перетворюють вхідні ознаки та формують прогноз. Для задачі бінарної класифікації доцільним є використання вихідного шару із сигмоїдною функцією активації, яка повертає значення в діапазоні від 0 до 1 [4; 20].

Проєктування ML-модуля також передбачає поділ даних на навчальну й тестову вибірки. Навчальна вибірка використовується для побудови моделі, а тестова – для перевірки її роботи на даних, які не використовувалися під час навчання. Це дозволяє оцінити здатність моделі працювати з новими записами [3; 7].

ML-модуль підтримує два основні режими роботи: навчання та прогнозування. У режимі навчання система формує модель на основі підготовленого датасету. У режимі прогнозування модель приймає один запис або групу записів і повертає прогнозований клас. Для групової обробки передбачено batch-прогнозування на основі JSON-масиву або CSV-файлу.

Оцінювання якості моделей виконується за допомогою основних класифікаційних метрик: accuracy, precision, recall і f1-score. Вони дозволяють оцінити загальну точність, якість прогнозування позитивного класу, повноту виявлення позитивних об’єктів і баланс між precision та recall [7; 8]. Додатково формується confusion matrix, яка показує кількість правильних і неправильних класифікацій для кожного класу [9].

Взаємодія ETL- і ML-модулів із веб-застосунком здійснюється через backend-частину. Користувач запускає потрібну дію у веб-інтерфейсі, frontend надсилає запит до backend-сервера, після чого backend викликає відповідний модуль і повертає результат виконання. Така структура дозволяє не запускати окремі скрипти вручну, а керувати всіма процесами через єдиний веб-інтерфейс.

До основних вимог до ETL- і ML-модулів належать:

1. підтримка завантаження CSV-даних;
2. перевірка структури вхідного набору;
3. очищення та перетворення даних;

4. формування додаткових ознак;
5. збереження підготовленого датасету;
6. навчання моделей RandomForest і Neural Network;
7. прогнозування для одного запису;
8. batch-прогнозування для групи записів;
9. розрахунок метрик якості;
10. формування confusion matrix;
11. передавання результатів до backend-частини для відображення у веб-інтерфейсі.

Таким чином, ETL-модуль і модуль машинного навчання забезпечують основну логіку роботи системи. ETL-модуль готує вхідні дані до аналізу, а ML-модуль використовує їх для навчання моделей, прогнозування та оцінювання якості.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Реалізація ETL-модуля та підготовки даних

У межах розробленого веб-застосунку реалізовано ETL-модуль, який відповідає за завантаження вхідного CSV-файлу, попередню обробку даних, формування ознак і створення підготовленого датасету для подальшого навчання моделей RandomForest і Neural Network.

Як демонстраційний набір даних у роботі використано датасет Titanic. Він містить інформацію про пасажирів, на основі якої виконується задача бінарної класифікації – прогнозування виживання пасажира. Перед використанням у моделях машинного навчання початкові дані потребують очищення, перетворення категоріальних значень і формування додаткових ознак.

Структуру вхідного CSV-файлу на основі підготовленого датасету Titanic наведено в табл. 3.1.

Таблиця 3.1 – Структура вхідного CSV-файлу

Назва поля	Тип даних	Опис	Приклад значення
Sex	Числовий	Закодована стать пасажира	0 або 1
Age	Числовий	Вік пасажира	25
Fare	Числовий	Вартість квитка	7,25
FamilySize	Числовий	Кількість членів сім’ї або супутників	1
IsAlone	Числовий	Ознака того, чи подорожує пасажир сам	0 або 1
Title	Числовий	Закодований титул пасажира	1

Продовження таблиці 3.1.

AgeGroup	Числовий	Категорія віку	0–3
Embarked	Числовий	Закодований порт посадки	0–2

Як видно з табл. 3.1, для прогнозування використовуються як початкові, так і сформовані під час обробки ознаки. Це дозволяє надати моделі структурований набір характеристик, придатний для задачі класифікації.

ETL-процес реалізовано як окремий програмний модуль, що викликається через backend-частину системи. Користувач запускає обробку даних через веб-інтерфейс або API-ендпоінт, після чого backend виконує необхідні операції над CSV-файлом. Такий підхід дозволяє не запускати окремі Python-скрипти вручну, а керувати підготовкою даних через єдиний веб-застосунок.

На першому етапі ETL-модуль зчитує CSV-файл і перетворює його у табличну структуру для подальшої обробки. Для цього використовується бібліотека pandas, яка забезпечує роботу з табличними даними та підтримує зчитування CSV-файлів [19]. Після завантаження виконується перевірка структури набору даних і наявності потрібних колонок.

Далі виконується очищення даних: обробляються пропущені значення, усуваються зайві або некоректні записи, а також виконується приведення даних до потрібних типів. Після цього формуються додаткові ознаки, зокрема FamilySize, IsAlone, Title та AgeGroup. Такі ознаки підвищують інформативність датасету та допомагають моделі краще виявляти залежності між характеристиками пасажира і цільовою змінною [3; 10].

Окремо виконується підготовка числових значень до використання в моделях. Для нейронної мережі важливо, щоб числові ознаки мали узгоджений масштаб, тому під час обробки може застосовуватися стандартизація або інші перетворення числових полів [10].

Після завершення обробки формується підготовлений датасет. Він може бути збережений у вигляді окремого CSV-файлу або переданий до бази даних

PostgreSQL для подальшого використання. Надалі цей набір використовується ML-модулем для навчання моделей, прогнозування та оцінювання якості результатів.

Алгоритм роботи ETL-модуля подано на рис. 3.1.

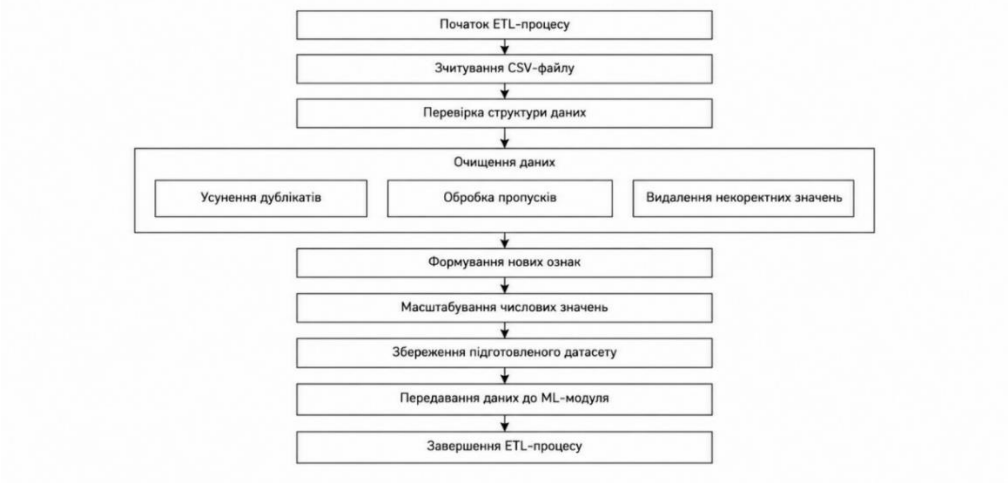


Рисунок 3.1 – Алгоритм роботи ETL-модуля

Як видно з рис. 3.1, робота ETL-модуля починається із завантаження CSV-файлу. Далі система перевіряє структуру даних, виконує очищення, формує нові ознаки, готує числові значення та створює підготовлений датасет.

Таким чином, реалізований ETL-модуль забезпечує повний цикл підготовки вхідного CSV-файлу до використання в моделях машинного навчання. Він виконує зчитування даних, перевірку структури, очищення, формування ознак, підготовку числових значень і створення підготовленого датасету.

3.2 Реалізація модуля машинного навчання, прогнозування та оцінювання якості

Після виконання ETL-обробки підготовлений датасет передається до модуля машинного навчання. У розроблюваній програмній системі реалізовано навчання двох моделей: RandomForest та Neural Network. Це дозволяє порівняти класичний алгоритм машинного навчання з моделлю на основі нейронної мережі та оцінити їхню ефективність для задачі бінарної класифікації.

Перед навчанням дані поділяються на навчальну та тестову вибірки. Навчальна вибірка використовується для побудови моделі, а тестова – для перевірки якості її роботи на нових даних. Такий підхід дозволяє оцінити здатність моделі узагальнювати закономірності, а не лише відтворювати навчальні приклади [3; 7].

Модель RandomForest реалізовано з використанням бібліотеки scikit-learn. Вона використовується для роботи з підготовленими табличними даними та формування прогнозу щодо виживання пасажира [6]. Нейронна мережа реалізується з використанням TensorFlow/Keras і застосовується як альтернативний підхід до задачі класифікації [4; 20].

У системі користувач може обрати тип моделі для навчання. Після вибору frontend-частина надсилає запит до backend-сервера, а backend викликає відповідний ML-модуль. Система завантажує підготовлений датасет, виділяє вхідні ознаки та цільову змінну, виконує поділ даних, навчає модель і формує прогнози для тестової вибірки.

Загальна послідовність навчання моделей у системі має такий вигляд:

1. завантаження підготовленого датасету після ETL-обробки;
2. виділення вхідних ознак і цільової змінної;
3. поділ даних на навчальну та тестову вибірки;
4. вибір моделі RandomForest або Neural Network;
5. навчання моделі на навчальній вибірці;
6. формування прогнозів для тестових даних;
7. розрахунок метрик якості;
8. повернення результатів до backend-частини;
9. відображення результатів у веб-інтерфейсі.

Алгоритм навчання моделей машинного навчання подано на рис. 3.2.

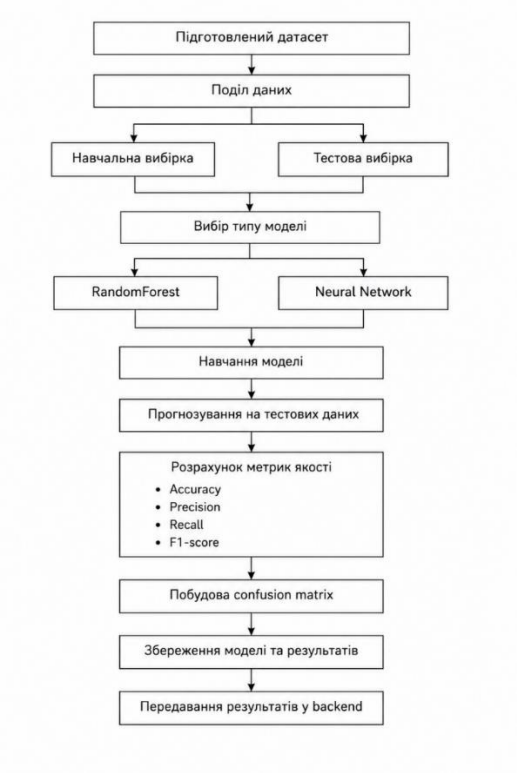


Рисунок 3.2 – Алгоритм навчання моделей машинного навчання

Як видно з рис. 3.2, процес навчання починається з отримання підготовленого датасету. Далі дані поділяються на навчальну й тестову вибірки, після чого виконується навчання RandomForest або Neural Network. Після завершення навчання система формує прогнози, розраховує метрики якості, будує confusion matrix і зберігає результати для подальшого використання.

Основні програмні модулі системи наведено в табл. 3.2.

Таблиця 3.2 – Опис програмних модулів системи

Модуль	Призначення	Основні функції
Backend-модуль	Організація серверної логіки та REST API	Приймання запитів, запуск ETL, навчання моделей, прогнозування, повернення результатів
ETL-модуль	Завантаження та попередня обробка даних	Зчитування CSV-файлу, очищення даних, обробка пропусків, збереження підготовленого датасету

Продовження таблиці 3.2

Модуль формування ознак	Розширення вхідного набору даних	Створення додаткових ознак на основі наявних полів
Модуль RandomForest	Навчання класичної моделі машинного навчання	Поділ даних, навчання RandomForest, прогнозування, розрахунок метрик
Модуль Neural Network	Навчання нейронної мережі	Створення архітектури нейронної мережі, навчання моделі, прогнозування
Модуль оцінювання якості	Аналіз результатів роботи моделей	Розрахунок accuracy, precision, recall, f1-score, формування confusion matrix
Frontend-модуль	Забезпечення взаємодії користувача із системою	Форми введення, кнопки запуску операцій, відображення прогнозів і результатів
Docker-конфігурація	Локальне розгортання системи	Запуск backend-сервера та бази даних у контейнеризованому середовищі

Після навчання модель може використовуватися для прогнозування. У системі реалізовано два режими прогнозування: для одного запису та для групи записів. У режимі одиничного прогнозування користувач заповнює форму Passenger Prediction у веб-інтерфейсі. JavaScript формує JSON-об’єкт із введених значень і надсилає його до API-ендпоінта. Backend перевіряє структуру запиту, передає дані до навченої моделі та повертає результат у frontend.

Для batch-прогнозування користувач може передати масив записів у форматі JSON або завантажити CSV-файл. Backend-частина приймає ці дані, перевіряє структуру колонок, готує записи до використання моделлю та повертає список прогнозів. Такий режим дозволяє працювати не лише з окремим прикладом, а й з набором даних.

Сценарій batch-прогнозування подано на рис. 3.3.

Рисунок 3.3 – Сценарій batch-прогнозування

Як видно з рис. 3.3, batch-обробка починається з передавання набору записів до системи. Вхідні дані можуть надходити у вигляді JSON-масиву або CSV-файлу. Після цього backend виконує перевірку та підготовку даних, передає їх до навченої моделі, отримує прогнози та повертає результати користувачу.

Оцінювання якості моделей виконується після завершення навчання. Для цього модель формує прогнози на тестовій вибірці, після чого прогнозовані значення порівнюються з фактичними. У системі розраховуються основні метрики класифікації: accuracy, precision, recall і f1-score [7; 8]. Для обчислення метрик використовуються засоби бібліотеки scikit-learn [5; 7].

Крім числових метрик, у системі реалізовано формування confusion matrix. Вона дозволяє проаналізувати кількість правильних і неправильних класифікацій для кожного класу та краще зрозуміти характер помилок моделі [7; 9]. Матриця помилок доповнює числові метрики, оскільки показує не лише загальний рівень якості, а й структуру результатів класифікації.

Таким чином, модуль машинного навчання забезпечує навчання моделей RandomForest і Neural Network, прогнозування для одного запису, batch-прогнозування та оцінювання якості результатів. Це дозволяє реалізувати повний цикл машинного навчання в межах локального веб-застосунку: від підготовки даних до отримання прогнозів і порівняння моделей.

3.3 Реалізація веб-інтерфейсу та локального запуску системи

Веб-інтерфейс користувача є частиною програмної системи, через яку користувач взаємодіє з основними функціями застосунку. Інтерфейс реалізовано як

веб-сторінку, що відкривається в браузері та забезпечує доступ до запуску ETL-процесу, навчання моделей, прогнозування й перегляду результатів.

Frontend-частина системи створена з використанням HTML, CSS і JavaScript. HTML використовується для побудови структури сторінки, форм введення даних, кнопок запуску операцій і блоків відображення результатів. JavaScript забезпечує взаємодію з backend-частиною через REST API, надсилання запитів і оновлення вмісту сторінки без її повного перезавантаження. CSS-файл надає сторінці мінімальної стилізації, завдяки якій користувачу зручніше сприймати продукт.

Основна сторінка веб-застосунку містить кілька функціональних блоків. Блок ETL Pipeline призначений для запуску підготовки даних. Блок Model Training дозволяє обрати модель RandomForest або Neural Network і запустити її навчання. Блок Passenger Prediction використовується для прогнозування одного запису, а Batch Prediction – для групового прогнозування на основі CSV-файлу. Окремо передбачено блок Model Metrics, який використовується для перегляду результатів оцінювання якості моделей.

Під час роботи з інтерфейсом користувач може виконувати основні дії без запуску окремих Python-скриптів. Наприклад, для навчання моделі достатньо обрати потрібний тип моделі та натиснути відповідну кнопку. Після цього frontend надсилає запит до backend-сервера, який викликає потрібний модуль і повертає результат виконання операції.

Приклад веб-інтерфейсу програмної системи подано на рис. 3.4.

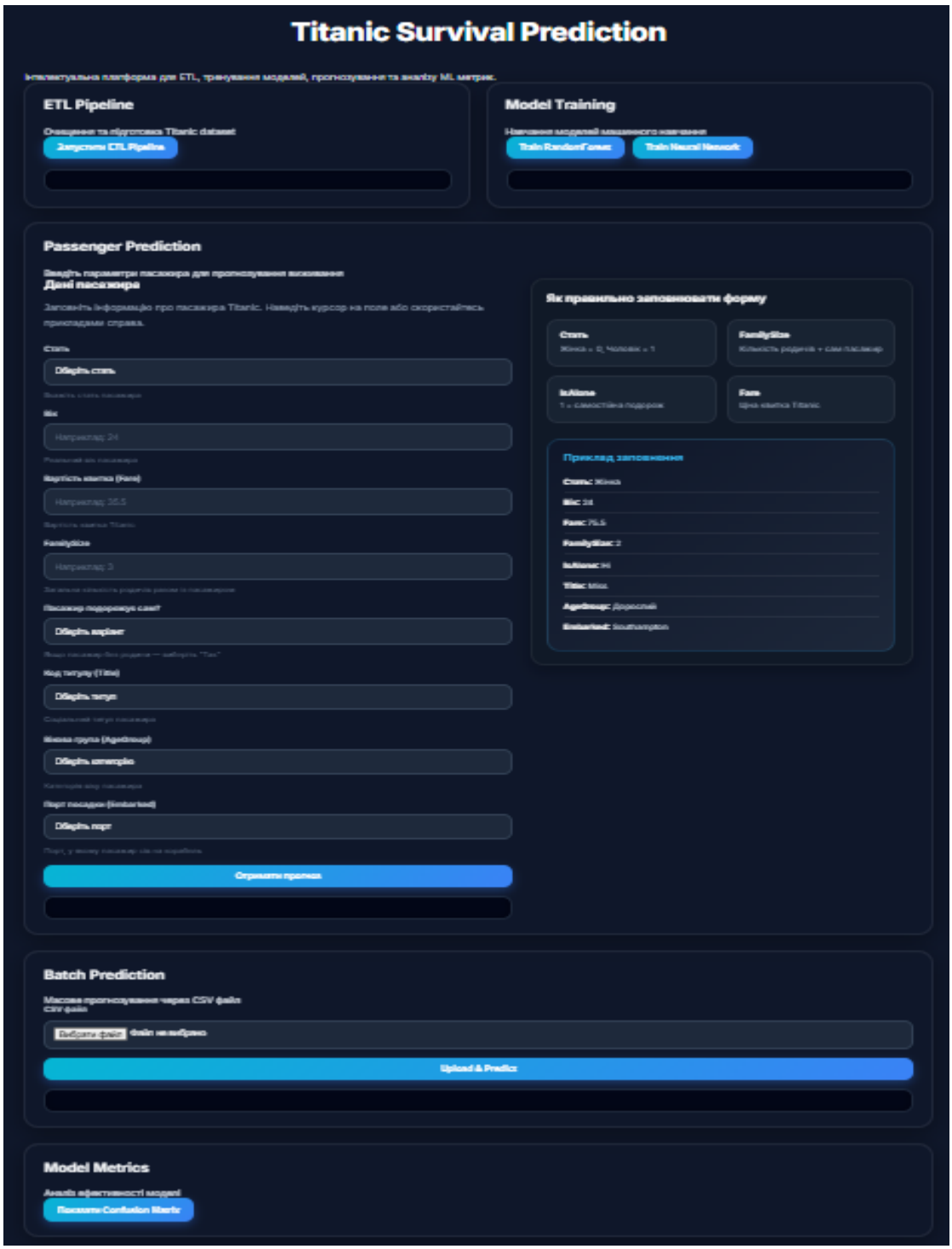


Рисунок 3.4 – Приклад веб-інтерфейсу програмної системи

Як видно з рис. 3.4, інтерфейс має просту структуру та містить основні блоки для роботи із системою. Користувач може послідовно запустити ETL-обробку, навчити модель, виконати прогнозування та переглянути результати. Детальніші приклади окремих функціональних блоків веб-інтерфейсу наведено в додатку Б.

Для забезпечення зручного локального запуску програмної системи використовується Docker. Це дозволяє запускати backend-сервер, базу даних PostgreSQL і необхідне середовище виконання без ручного налаштування всіх залежностей на комп’ютері користувача [15; 16].

У межах реалізації системи використовується Dockerfile, у якому описано середовище для запуску backend-частини. У цьому файлі визначається базовий образ Python, копіюються файли проєкту, встановлюються необхідні залежності та задається команда запуску FastAPI-сервера. Для запуску кількох сервісів використовується файл docker-compose.yml, який описує backend-сервіс і сервіс бази даних PostgreSQL [17; 18].

Локальний запуск системи виконується за допомогою команди:

docker-compose up –build

Після виконання цієї команди Docker Compose збирає необхідні образи, запускає backend-сервер і базу даних. Backend-частина стає доступною через локальний порт, після чого користувач може відкрити веб-інтерфейс у браузері. Також користувач може переглянути автоматично згенеровану документацію REST API через Swagger UI.

Використання Docker спрощує демонстрацію програмного продукту. Користувачу не потрібно вручну встановлювати Python-залежності, налаштовувати PostgreSQL або запускати окремі компоненти системи.

Таким чином, реалізований веб-інтерфейс забезпечує зручний доступ до основних функцій системи, а Docker дозволяє запускати програмний продукт у локальному контейнеризованому середовищі. Це робить застосунок зручним для тестування, демонстрації та подальшого використання.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ОЦІНКА ЯКОСТІ ПРОГРАМНОЇ СИСТЕМИ

4.1 Організація тестування та перевірка функціональних сценаріїв

Тестування програмної системи проводилося для перевірки її працездатності та відповідності поставленим вимогам. У межах роботи перевірялися основні компоненти локального веб-застосунку: REST API, ETL-модуль, модуль машинного навчання, веб-інтерфейс, одиничне прогнозування, batch-обробка та модуль оцінювання якості.

Мета тестування полягала в перевірці повного циклу роботи системи: завантаження CSV-файлу, попередньої обробки даних, навчання моделей RandomForest і Neural Network, формування прогнозів, розрахунку метрик якості та побудови confusion matrix. Окремо перевірялася коректність взаємодії між frontend- і backend-частинами через REST API.

Основні тестові сценарії наведено в табл. 4.1.

Таблиця 4.1 – Тестові сценарії перевірки програмної системи

№	Об’єкт тестування	Очікуваний результат	Статус
1	Запуск локального середовища	Backend-сервер і база даних запускаються без критичних помилок	Виконано
2	Відкриття веб-інтерфейсу	Головна сторінка застосунку відкривається в браузері	Виконано
3	Запуск ETL-процесу	Дані зчитуються, обробляються та зберігаються	Виконано
4	Обробка CSV-файлу	Формується підготовлений датасет	Виконано
5	Навчання RandomForest	Модель навчається, формуються метрики	Виконано

Продовження таблиці 4.1

6	Навчання Neural Network	Нейронна мережа навчається, формуються результати	Виконано
7	Прогнозування для одного запису	Система повертає прогнозований клас	Виконано
8	Batch-прогнозування	Система обробляє набір записів і повертає прогнози	Виконано
9	Розрахунок метрик якості	Обчислюються accuracy, precision, recall і f1-score	Виконано
10	Формування confusion matrix	Створюється матриця помилок	Виконано
11	Відображення результатів у frontend	Результати показуються у веб-інтерфейсі	Виконано

Як видно з табл. 4.1, тестування охоплювало основні функціональні частини системи. Перевірка проводилася як для окремих модулів, так і для сценаріїв, у яких одночасно взаємодіють frontend, backend, ETL-модуль і ML-модуль.

Працездатність REST API перевірялася через локальний запуск backend-сервера та автоматично згенеровану документацію Swagger UI. Після запуску застосунку користувач може відкрити головну сторінку за локальною адресою, а також переглянути доступні API-ендпоінти через Swagger UI.

Автоматично згенеровану документацію REST API подано на рис. 4.1.

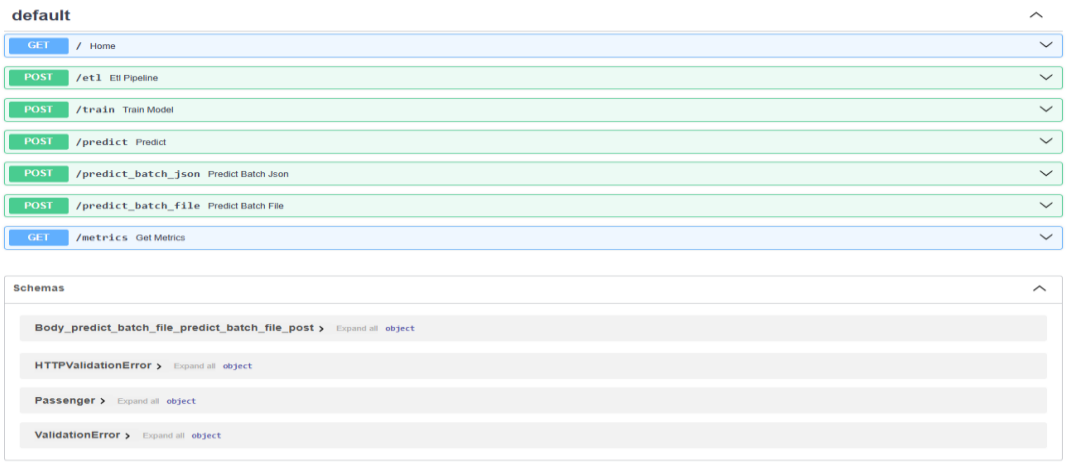


Рисунок 4.1 – Автоматично згенерована документація REST API у Swagger UI

Під час тестування API було перевірено доступність основних ендпоінтів, запуск ETL-процесу, навчання моделей, приймання даних для прогнозування, batch-обробку та повернення результатів оцінювання якості. Успішне виконання цих сценаріїв підтвердило, що backend-частина коректно взаємодіє з функціональними модулями системи.

Окремо перевірялося завантаження та обробка CSV-файлу. Система мала зчитати файл, перевірити структуру даних, обробити пропущені значення, сформувати додаткові ознаки та створити підготовлений датасет для навчання моделей. Приклад успішного виконання ETL-процесу наведено в додатку В.1.

Також було протестовано навчання моделей RandomForest і Neural Network. Перед навчанням система перевіряє наявність підготовленого датасету, виконує поділ даних на навчальну й тестову вибірки, запускає обрану модель і формує результати для подальшого оцінювання. Результати навчання RandomForest наведено в додатку В.2, а результати навчання Neural Network – у додатку В.3.

Таким чином, функціональне тестування підтвердило, що система коректно виконує основні сценарії роботи: запуск локального середовища, обробку CSV-файлу, навчання моделей, прогнозування та передавання результатів у веб-інтерфейс.

4.2 Оцінка якості моделей машинного навчання

Оцінка якості моделей виконувалася після навчання RandomForest і Neural Network на підготовленому датасеті. Для перевірки результатів використовувалися метрики accuracy, precision, recall і f1-score, які дозволяють оцінити загальну точність класифікації, якість визначення позитивного класу, повноту його виявлення та баланс між precision і recall [7; 8].

Метрики якості моделей наведено в табл. 4.2.

Таблиця 4.2 – Метрики якості моделей машинного навчання

Модель	Accuracy	Precision	Recall	F1-score
RandomForest	0,8101	0,7778	0,7568	0,7671
Neural Network	0,7821	0,7612	0,6892	0,7234

Як видно з табл. 4.2, модель RandomForest показала вищі значення за всіма основними метриками порівняно з Neural Network. Accuracy для RandomForest становить 0,8101, тоді як для Neural Network – 0,7821. Це свідчить про кращу загальну точність класифікації ансамблевої моделі на тестовій вибірці.

За показниками precision, recall і f1-score RandomForest також має кращі результати. Найбільша різниця спостерігається за recall, що вказує на кращу здатність цієї моделі виявляти об’єкти позитивного класу. Отже, для використаного набору даних RandomForest є більш ефективною та збалансованою моделлю.

Для наочного порівняння результатів побудовано стовпчикову діаграму, подану на рис. 4.2.

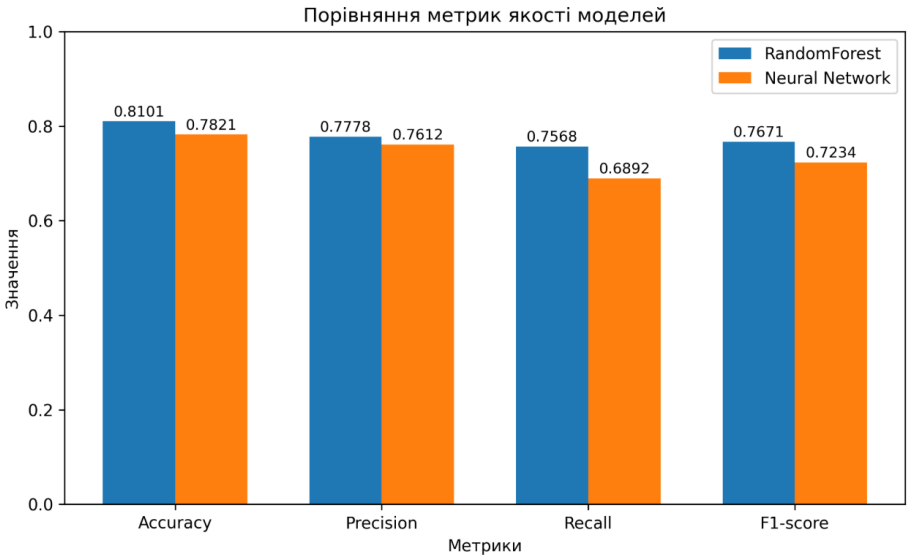


Рисунок 4.2 – Порівняння метрик якості моделей

Крім числових метрик, у системі використовується confusion matrix. Вона дозволяє оцінити структуру правильних і помилкових прогнозів, тобто показує, які класи модель визначає коректно, а де допускає помилки [7; 9].

Приклад сформованої confusion matrix подано на рис. 4.3.

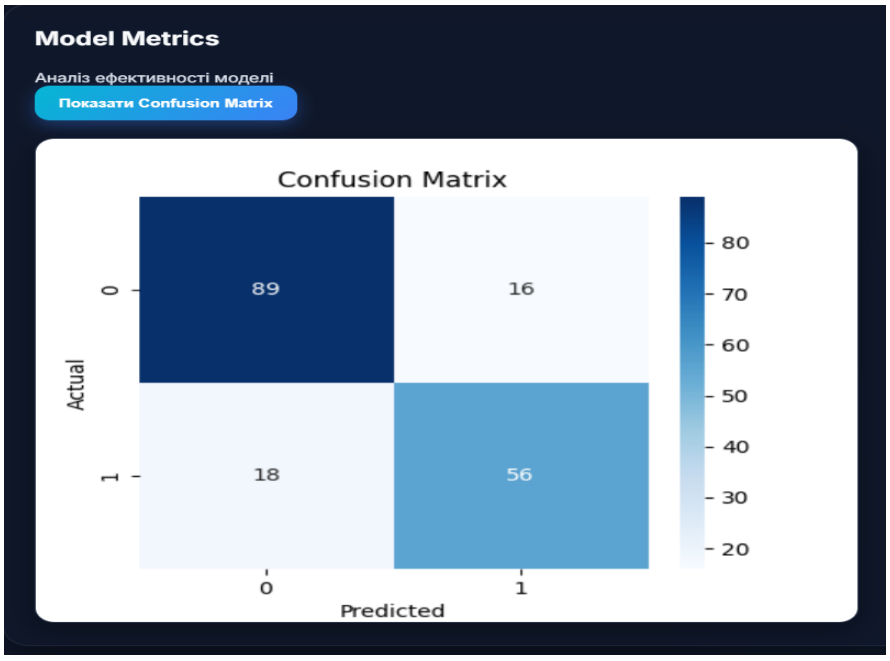


Рисунок 4.3 – Приклад confusion matrix

Як видно з рис. 4.3, значення на головній діагоналі відповідають правильним прогнозам моделі, а значення поза головною діагоналлю показують помилки

класифікації. Такий аналіз доповнює числові метрики та дозволяє краще оцінити поведінку моделі на тестових даних.

Таким чином, оцінювання якості моделей показало, що RandomForest продемонструвала кращі результати порівняно з Neural Network. Це підтверджується як числовими метриками, так і графічним порівнянням результатів.

4.3 Перевірка веб-інтерфейсу та загальна оцінка працездатності системи

Окремо було перевірено роботу веб-інтерфейсу, оскільки саме через нього користувач взаємодіє з основними функціями програмної системи. Інтерфейс має забезпечувати запуск ETL-процесу, навчання моделей, прогнозування для одного запису, batch-обробку та перегляд результатів.

Під час перевірки веб-інтерфейсу оцінювалася доступність основних функціональних блоків, коректність роботи кнопок і форм, передавання запитів до backend-частини та відображення результатів виконання операцій. Користувач може виконувати основні дії без ручного запуску Python-скриптів, що робить систему зручнішою для демонстрації та тестування.

Для одиничного прогнозування користувач заповнює форму Passenger Prediction, після чого введені дані передаються до backend-сервера, проходять перевірку та використовуються навченою моделлю для формування прогнозу. Приклад такого сценарію наведено в додатку В.4.

Також було перевірено batch-прогнозування через CSV-файл. У цьому режимі користувач завантажує файл із кількома записами, система обробляє їх і повертає список прогнозів. Приклад batch-прогнозування наведено в додатку В.5.

У результаті тестування встановлено, що frontend-частина коректно взаємодіє з backend-сервером, а результати операцій відображаються у веб-інтерфейсі. Інтерфейс має просту структуру та містить основні функціональні блоки, необхідні для роботи із застосунком.

Загалом тестування підтвердило, що розроблена система відповідає поставленій задачі. Вона забезпечує повний цикл роботи з даними в задачі машинного навчання: завантаження CSV-файлу, ETL-обробку, навчання моделей, одиничне й batch-прогнозування, розрахунок метрик і візуальний аналіз результатів за допомогою confusion matrix.

Разом із тим, система має напрями для подальшого вдосконалення. У майбутньому доцільно додати підтримку інших форматів даних, розширити набір моделей машинного навчання, реалізувати збереження історії експериментів, покращити візуалізацію результатів і розгорнути застосунок на сервері або хмарній платформі.

Таким чином, результати тестування підтвердили працездатність програмної системи. Працездатність було перевірено через Swagger UI, виконання ETL-процесу, навчання моделей RandomForest і Neural Network, одиничне прогнозування, batch-прогнозування, розрахунок метрик якості та побудову confusion matrix. Відповідні результати наведено на рис. 4.1–4.3 та в додатку В.

ВИСНОВКИ

У бакалаврській роботі було розглянуто розроблення програмної системи забезпечення інженерії даних для машинного навчання. Актуальність теми зумовлена потребою в програмних рішеннях, які поєднують підготовку даних, навчання моделей, прогнозування та оцінювання якості результатів у межах одного застосунку.

У першому розділі було проаналізовано предметну область інженерії даних і машинного навчання. Розглянуто роль ETL-процесів, особливості підготовки CSV-даних, формування ознак, використання моделей RandomForest і Neural Network, а також основні метрики оцінювання якості класифікації. Також було виконано огляд існуючих рішень для обробки даних і навчання моделей та обґрунтовано доцільність створення власного локального веб-застосунку.

У другому розділі було спроєктовано архітектуру програмної системи. Визначено клієнт-серверну структуру застосунку, у якій frontend-частина забезпечує взаємодію з користувачем, а backend-частина відповідає за обробку запитів, запуск ETL-процесу, навчання моделей і повернення результатів. Для реалізації системи обрано Python, FastAPI, HTML, JavaScript, CSS PostgreSQL і Docker.

У третьому розділі було описано реалізацію програмного продукту. Реалізовано модуль завантаження та попередньої обробки CSV-файлів, ETL-пайплайн, навчання моделей RandomForest і Neural Network, прогнозування для одного запису, batch-обробку, розрахунок метрик якості, формування confusion matrix, веб-інтерфейс користувача та Docker-контейнер для локального запуску системи. Також описано порядок локального запуску застосунку через Docker Compose.

У четвертому розділі було проведено тестування та оцінку якості програмної системи. Перевірено працездатність REST API, коректність завантаження й обробки CSV-файлів, запуск навчання моделей, одиничне й batch-прогнозування, розрахунок метрик accuracy, precision, recall і f1-score, а також формування матриці

помилок. Працездатність REST API підтверджено через Swagger UI, результати тестування наведено в додатках, а порівняння моделей подано у вигляді таблиці та діаграми.

Мету бакалаврської роботи досягнуто. У результаті було розроблено локальний веб-застосунок, який забезпечує повний цикл роботи з даними для задачі машинного навчання: від завантаження та попередньої обробки CSV-файлу до навчання моделей, прогнозування й оцінювання якості результатів.

За результатами оцінювання якості моделей встановлено, що RandomForest продемонструвала кращі значення основних метрик порівняно з Neural Network. Для RandomForest отримано accuracy 0,8101, precision 0,7778, recall 0,7568 та f1-score 0,7671. Для Neural Network отримано accuracy 0,7821, precision 0,7612, recall 0,6892 та f1-score 0,7234. Це свідчить про кращу збалансованість RandomForest для використаного набору даних.

Практичне значення роботи полягає в тому, що розроблена система може бути використана як навчальний і демонстраційний інструмент для вивчення принципів інженерії даних, побудови REST API, реалізації ML-модулів і контейнеризованого локального розгортання. Система демонструє не лише навчання моделей, а й повний процес підготовки даних, прогнозування, batch-обробки та аналізу якості результатів.

У подальшому систему можна вдосконалити шляхом додавання нових моделей машинного навчання, підтримки інших форматів даних, розширення візуалізації результатів, збереження історії експериментів, реалізації авторизації користувачів і розгортання застосунку на сервері або хмарній платформі.

Таким чином, розроблена програмна система відповідає поставленій меті, має модульну структуру та забезпечує основні етапи роботи з даними в задачах машинного навчання. Її працездатність підтверджено тестовими сценаріями, результатами навчання моделей, прикладами прогнозування, графічним порівнянням метрик і побудовою confusion matrix.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol: O'Reilly Media, 2017. 616 p. URL: https://0-lucas.github.io/digital-garden/99.-Books/Martin-Kleppmann---Designing-Data-Intensive-Applications_-O%E2%80%99Reilly-Media-(2017).pdf
2. Reis J., Housley M. Fundamentals of Data Engineering: Plan and Build Robust Data Systems. Sebastopol: O'Reilly Media, 2022. 446 p. URL: https://soclibrary.futa.edu.ng/books/Fundamentals%20of%20Data%20Engineering%20(Reis,%20JoeHousley,%20Matt)%20(Z-Library).pdf
3. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. 3rd ed. Sebastopol: O'Reilly Media, 2022. 864 p. URL: https://www.rasa-ai.com/wp-content/uploads/2022/02/Aur%C3%A9lien-G%C3%A9ron-Hands-On-Machine-Learning-with-Scikit-Learn-Keras-and-Tensorflow_-Concepts-Tools-and-Techniques-to-Build-Intelligent-Systems-O%E2%80%99Reilly-Media-2019.pdf
4. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge: MIT Press, 2016. 800 p. URL: https://www.researchgate.net/publication/320703571_Ian_Goodfellow_Yoshua_Bengio_and_Aaron_Courville_Deep_learning_The_MIT_Press_2016_800_pp_ISBN_0262035618
5. scikit-learn. User Guide. URL: https://scikit-learn.org/stable/user_guide.html (дата звернення: 09.05.2026).
6. scikit-learn. RandomForestClassifier. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (дата звернення: 09.05.2026).
7. scikit-learn. Model evaluation: quantifying the quality of predictions. URL: https://scikit-learn.org/stable/modules/model_evaluation.html (дата звернення: 09.05.2026).

8. scikit-learn. f1_score. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html (дата звернення: 09.05.2026).
9. scikit-learn. confusion_matrix. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html (дата звернення: 09.05.2026).
10. scikit-learn. StandardScaler. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (дата звернення: 09.05.2026).
11. FastAPI. FastAPI Documentation. URL: https://fastapi.tiangolo.com/ (дата звернення: 09.05.2026).
12. FastAPI. Tutorial – User Guide. URL: https://fastapi.tiangolo.com/tutorial/ (дата звернення: 09.05.2026).
13. FastAPI. First Steps. URL: https://fastapi.tiangolo.com/tutorial/first-steps/ (дата звернення: 09.05.2026).
14. OpenAPI Initiative. OpenAPI Specification. URL: https://spec.openapis.org/oas/v3.2.0.html (дата звернення: 09.05.2026).
15. Docker. Docker Compose Documentation. URL: https://docs.docker.com/compose/ (дата звернення: 09.05.2026).
16. Docker. Compose file reference. URL: https://docs.docker.com/reference/compose-file/ (дата звернення: 09.05.2026).
17. PostgreSQL. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/current/index.html (дата звернення: 09.05.2026).
18. PostgreSQL. Data Types. URL: https://www.postgresql.org/docs/current/datatype.html (дата звернення: 09.05.2026).
19. pandas. pandas.read_csv. URL: https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html (дата звернення: 09.05.2026).

20. TensorFlow. tf.keras.Sequential. URL: https://www.tensorflow.org/api_docs/python/tf/keras/Sequential (дата звернення: 09.05.2026).
21. NumPy. NumPy Documentation. URL: https://numpy.org/doc/stable/ (дата звернення: 09.05.2026).
22. SQLAlchemy. SQLAlchemy Documentation. URL: https://docs.sqlalchemy.org/ (дата звернення: 09.05.2026).
23. Pydantic. Models. URL: https://docs.pydantic.dev/latest/concepts/models/ (дата звернення: 09.05.2026).
24. Provost F., Fawcett T. *Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking*. Hoboken : Wiley, 2013. 414 p.
25. Géron A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 3rd ed. Sebastopol : O'Reilly Media, 2022. 851 p.
26. Goodfellow I., Bengio Y., Courville A. *Deep Learning*. Cambridge : MIT Press, 2016. 775 p.
27. Kleppmann M. *Designing Data-Intensive Applications*. Sebastopol : O'Reilly Media, 2017. 616 p.
28. Albon C. *Machine Learning with Python Cookbook*. Sebastopol : O'Reilly Media, 2018. 366 p.

ДОДАТКИ

ДОДАТОК А

Фрагменти програмного коду

Додаток А.1 – Фрагмент реалізації REST API

```
from fastapi import FastAPI, Query, UploadFile, File
from pydantic import BaseModel, Field
from typing import List
from pathlib import Path
from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from fastapi.responses import FileResponse
import joblib
import pandas as pd
import logging
import numpy as np
from etl import run_etl
from ml_model import train_rf
from nn_model import train_nn

logging.basicConfig(level=logging.INFO)

app = FastAPI(title="ML Платформа Titanic")
model = None
BASE_DIR = Path(__file__).resolve().parent

app.mount(
    "/static",
    StaticFiles(directory=BASE_DIR / "static"),
    name="static"
)

class Passenger(BaseModel):
    Sex: int = Field(..., ge=0, le=1, description="0=Female, 1=Male")
    Age: float = Field(..., ge=0, description="Age must be non-negative")
    Fare: float = Field(..., ge=0, description="Fare must be non-negative")
    FamilySize: int = Field(..., ge=1, description="At least 1 (the passenger)")
    IsAlone: int = Field(..., ge=0, le=1)
```

```
    Title: int
    AgeGroup: int = Field(..., ge=0, le=3)
    Embarked: int

@app.get("/")
def home():
    return FileResponse("web/index.html")

@app.post("/etl")
def etl_pipeline(
    input_file: str = Query("titanic.csv", description="Шлях до вхідного CSV"),
    output_file: str = Query("titanic_prepared.csv", description="Шлях до вихідного
CSV")
):
    try:
        df = run_etl(input_file, output_file)
        logging.info(f"ETL завершено: {len(df)} рядків")
        return {
            "status": "ETL complete",
            "rows": len(df),
            "input_file": input_file,
            "output_file": output_file
        }
    except Exception as e:
        logging.error(f"ETL error: {e}")
        return {"error": str(e)}

@app.post("/train")
def train_model(model_type: str = Query("rf", description="Тип моделі: rf або nn")):
    try:
        df = pd.read_csv("titanic_prepared.csv")
        global model

        if model_type == "rf":
            model, metrics = train_rf(df)
        elif model_type == "nn":
            model, history = train_nn(df)
            metrics = {"accuracy": history.history["val_accuracy"][-1]}
        else:
            return {"error": f"Unknown model type: {model_type}"}

        joblib.dump(model, "model.pkl")
        logging.info(f"Model {model_type} trained")
```

```
        return {"status": "Model trained", "model_type": model_type, "metrics":
metrics}
    except Exception as e:
        logging.error(f"Train error: {e}")
        return {"error": str(e)}

@app.post("/predict")
def predict(passenger: Passenger):
    global model
    if model is None:
        try:
            model = joblib.load("model.pkl")
        except Exception:
            return {"error": "Model not trained yet"}

    data = [[
        passenger.Sex, passenger.Age, passenger.Fare, passenger.FamilySize,
        passenger.IsAlone, passenger.Title, passenger.AgeGroup, passenger.Embarked
    ]]

    X = np.array(data).reshape(1, -1)

    try:
        prediction = model.predict(X)[0]
    except Exception as e:
        return {"error": f"Prediction failed: {str(e)}"}

    if isinstance(prediction, (np.ndarray, list)):
        prediction = prediction.item()
    return {"prediction": int(prediction)}
```

Додаток А.2 – Фрагмент реалізації ETL-модуля

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sqlalchemy import create_engine
import joblib

def add_features(df: pd.DataFrame) -> pd.DataFrame:
    df["FamilySize"] = df["SibSp"] + df["Parch"] + 1
    df["IsAlone"] = (df["FamilySize"] == 1).astype(int)
```

```
    df["Title"] = df["Name"].str.extract(r' ([A-Za-z]+)\.', expand=False)
    df["Title"] = df["Title"].replace(['Mlle', 'Ms'], 'Miss')
    df["Title"] = df["Title"].replace(['Mme'], 'Mrs')
    df["Title"]   =   df["Title"].replace(['Don',   'Sir',   'Countess',   'Lady',
'Jonkheer'], 'Rare')
    df["Title"] = df["Title"].astype('category').cat.codes

    df["AgeGroup"] = pd.cut(df["Age"], bins=[0, 12, 18, 50, 80], labels=[0, 1, 2,
3])
    return df

def run_etl(input_file="titanic.csv", output_file="titanic_prepared.csv"):
    df = pd.read_csv(input_file)

    df.drop_duplicates(inplace=True)
    df.fillna(df.mean(numeric_only=True), inplace=True)
    for col in df.select_dtypes(include=['object']).columns:
        df[col].fillna(df[col].mode()[0], inplace=True)

    df = add_features(df)
    df['Sex'] = df['Sex'].astype('category').cat.codes
    df['Embarked'] = df['Embarked'].astype('category').cat.codes

    scaler = StandardScaler()
    df[['Age', 'Fare']] = scaler.fit_transform(df[['Age', 'Fare']])
    joblib.dump(scaler, "scaler.pkl")

    df.to_csv(output_file, index=False)

    engine = create_engine("postgresql://<user>:<password>@db:5432/mydb")
    df.to_sql("titanic_dataset", engine, if_exists="replace", index=False)

    return df
```

Додаток А.3 – Фрагмент реалізації моделей машинного навчання

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
```

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

def train_nn(df: pd.DataFrame):

    features = ["Sex", "Age", "Fare", "FamilySize", "IsAlone", "Title", "AgeGroup",
"Embarked"]
    X = df[features]
    y = df["Survived"]

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    joblib.dump(scaler, "scaler.pkl")

    X_train, X_test, y_train, y_test = train_test_split(
        X_scaled, y, test_size=0.2, random_state=42
    )

    model = tf.keras.Sequential([
                                tf.keras.layers.Dense(64,       activation='relu',
input_shape=(X_train.shape[1],)),
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])

                model.compile(optimizer='adam',       loss='binary_crossentropy',
metrics=['accuracy'])

    history = model.fit(
        X_train, y_train,
        epochs=20,
        batch_size=32,
        validation_data=(X_test, y_test),
        verbose=1
    )

    model.save("nn_model.h5")

    return model, history
```

Додаток А.4 – Фрагмент конфігурації Docker

Dockerfile (Збірка Python-сервісу):

```
FROM python:3.11-slim

WORKDIR /app

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

COPY . .
EXPOSE 8000

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

docker-compose.yml (Оркестрація сервісів):

```
services:
  api:
    build: .
    container_name: ml_api
    ports:
      - "8000:8000"
    depends_on:
      - db
    environment:
      - DATABASE_URL=postgresql://user:password@db:5432/mydb

  db:
    image: postgres:15
    container_name: postgres_db
    restart: always
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: mydb
    ports:
      - "5432:5432"
    volumes:
```

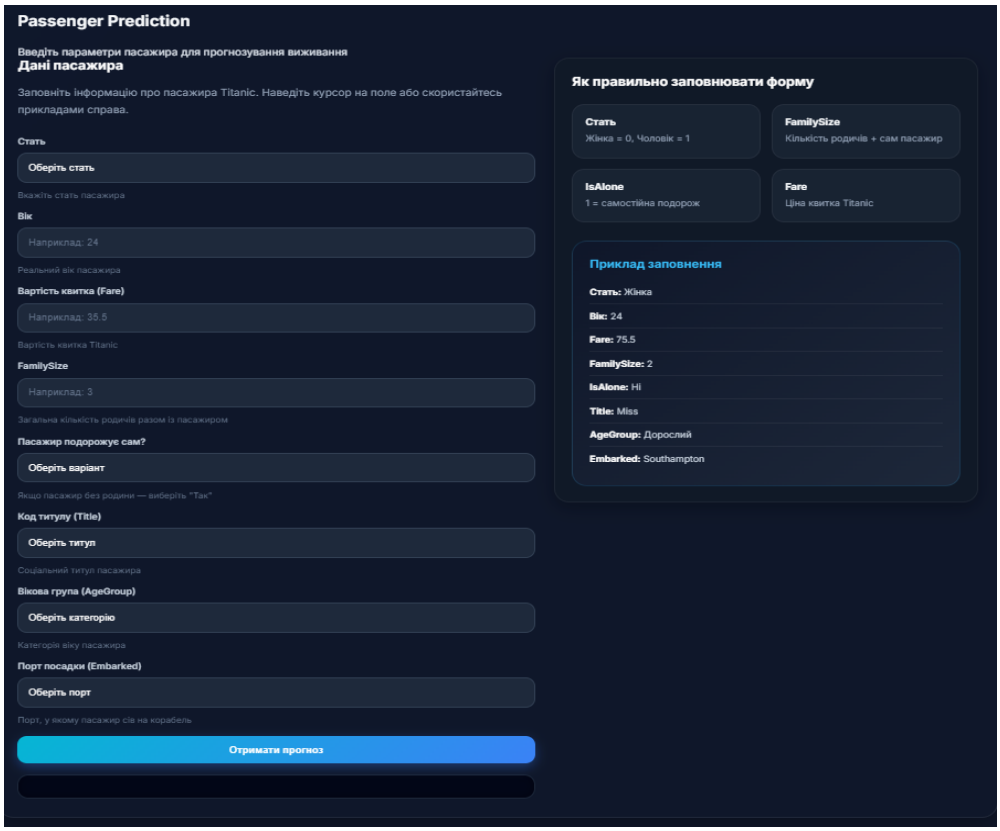
```
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

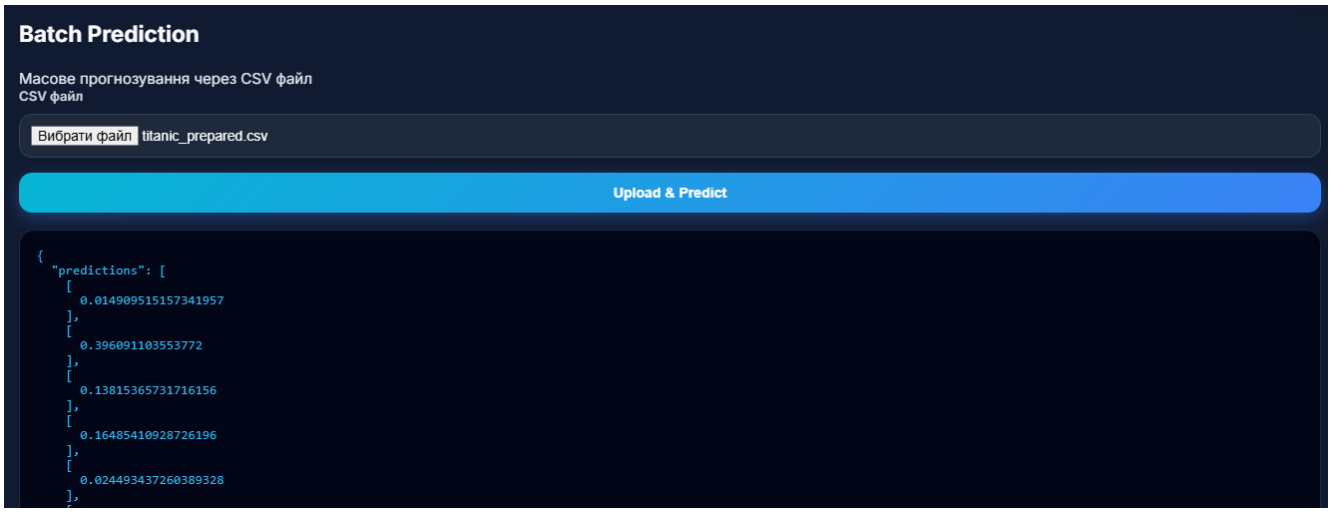
ДОДАТОК Б

Функціональні блоки веб-інтерфейсу

Додаток Б.1 – Форма прогнозування Passenger Prediction



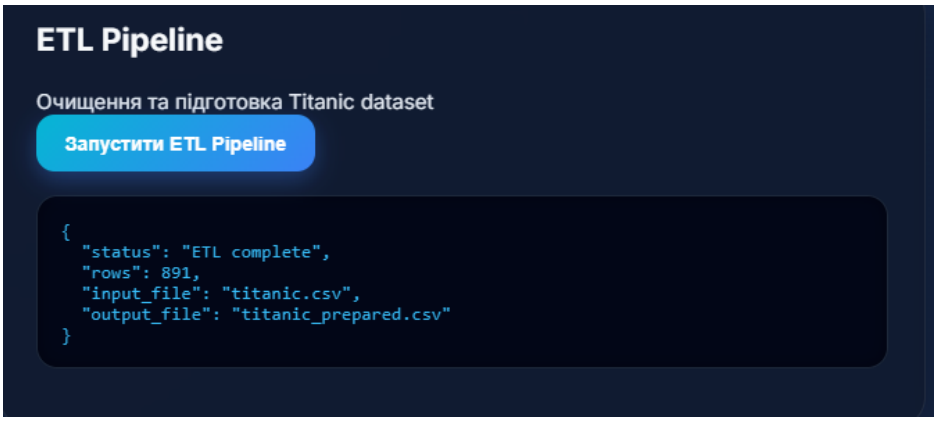
Додаток Б.2 – Блок batch-прогнозування



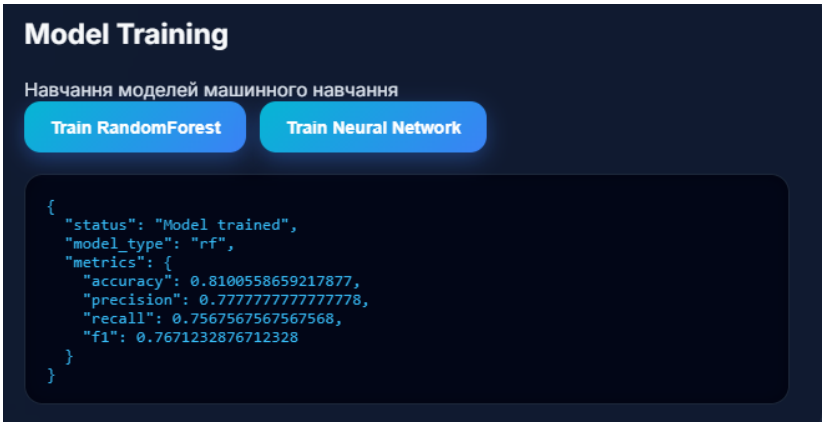
ДОДАТОК В

Результати тестування програмної системи

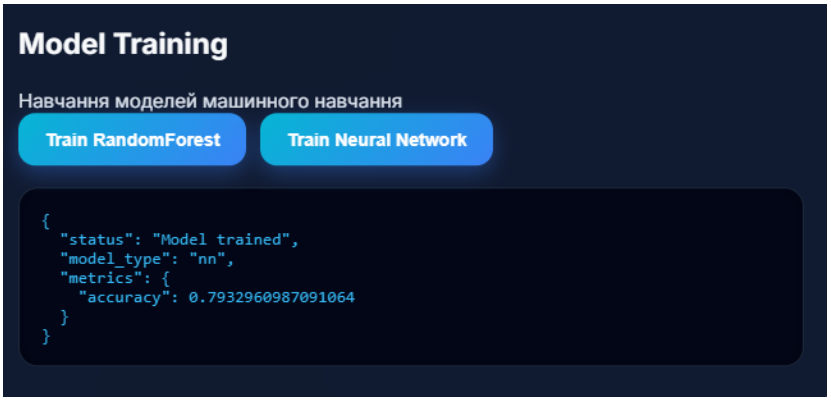
Додаток В.1 – Результат виконання ETL-процесу



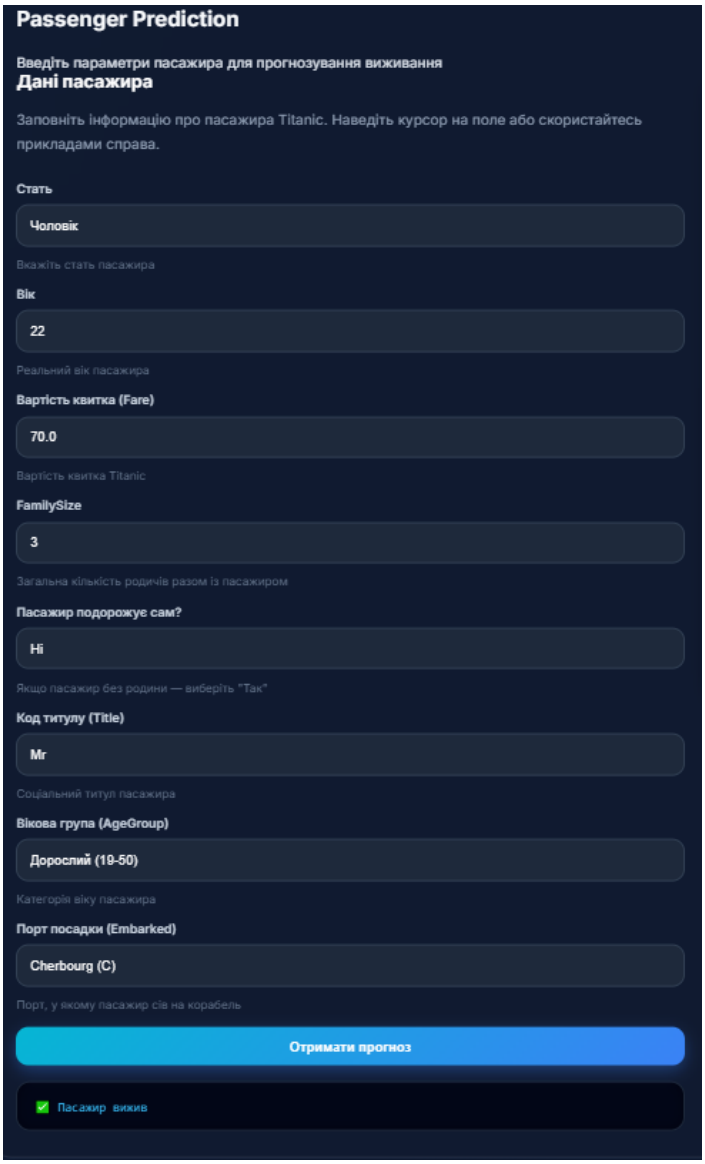
Додаток В.2 – Результат навчання моделі RandomForest



Додаток В.3 – Результат навчання моделі Neural Network



Додаток В.4 – Результат прогнозування для одного запису



Додаток В.5 – Результат batch-прогнозування