МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра фундаментальної та прикладної математики

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«АВТОМАТИЗАЦІЯ ПРОЦЕСУ МОДЕЛЮВАННЯ ТА ПРОГНОЗУВАННЯ ВЕЛИКИХ ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ»**

Виконав: студент 2 курсу, групи 8.1134
спеціальності 113 прикладна математика
(шифр і назва спеціальності)
освітньої програми прикладна математика
(назва освітньої програми)
Г. Є. Жутовський
(ініціали та прізвище)

Керівник доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Леонтьєва В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент декан математичного факультету, професор, д. т. н. Гоменюк С. І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра фундаментальної та прикладної математики

Рівень вищої освіти магістр

Спеціальність 113 прикладна математика
(шифр і назва)

Освітня програма прикладна математика

ЗАТВЕРДЖУЮ
Завідувач кафедри фундаментальної та прикладної математики, д.т.н., професор

_____ Гребенюк С.М.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Жутовському Глібу Євгеновичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Автоматизація процесу моделювання та прогнозування великих експериментальних даних

керівник роботи Леонтьєва Вікторія Володимирівна, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 28 » лютого 2025 року № 326-с

2. Строк подання студентом роботи 22.10.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Проєктування.
3. Реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 03.03.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	05.03.2025р.	
2.	Збір вихідних даних.	26.03.2025р.	
3.	Обробка методичних та теоретичних	15.05.2025р.	
	джерел.		
4.	Розробка першого та другого розділу.	12.06.2025р.	
5.	Розробка третього розділу.	02.09.2025р.	
6.	Оформлення та нормоконтроль	02.12.2025р.	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	16.12.2025р.	

Студент _____ Г. Є. Жутовський
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В. Леонтьєва
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Автоматизація процесу моделювання та прогнозування великих експериментальних даних»: 68 с., 38 рис., 51 джерело, 1 додаток.

АВТОМАТИЗАЦІЯ, МОДЕЛЮВАННЯ, ПРОГНОЗУВАННЯ, ВЕЛИКІ ДАНІ, PYTHON, LIGHTGBM, PIPELINE, COLUMNTRANSFORMER, HPO, RANDOMIZEDSEARCHCV, CROSS-VALIDATION, MLFLOW, RMSE, MAE, R^2.

Об’єкт дослідження – процес моделювання та прогнозування на великих експериментальних наборах даних.

Мета роботи – розробити та обґрунтувати методику і програмне рішення для автоматизації повного циклу моделювання та прогнозування великих експериментальних даних із забезпеченням відтворюваності та масштабованості.

Методи дослідження – аналітичний, формальний, методи машинного навчання, оптимізація гіперпараметрів, експериментальний.

У роботі виконано аналіз проблематики автоматизації ML-процесу для великих даних, спроєктовано архітектуру рішення на Python із використанням Pipeline та ColumnTransformer, ансамблю LightGBM, журналювання експериментів у MLflow. Реалізовано автоматизований конвеєр: зчитування та економне типізування даних, інженерія ознак, крос-валідаційне порівняння моделей, HPO на підвибірці з подальшим донавчанням і ранньою зупинкою, підсумкова оцінка на holdout-вибірці. Експерименти показали покращення метрик при збільшенні навчальної вибірки до ~50k і плато якості надалі, також виявлено важкохвостий характер цільової змінної (RMSE ≫ MAE).

Таким чином практичнною цінністю є відтворюваний шаблон автоматизації ML-експериментів, придатний до розширення.

SUMMARY

Master’s qualifying paper «Automation of Modeling and Forecasting Process of Big Experimental Data»: 68 pages, 38 figures, 1 table, 51 references, 1 supplement.

AUTOMATION, MODELING, FORECASTING, BIG DATA, PYTHON, LIGHTGBM, PIPELINE, COLUMNTRANSFORMER, HPO, RANDOMIZEDSEARCHCV, CROSS-VALIDATION, MLFLOW, RMSE, MAE, R2.

The object of the study is the process of modeling and forecasting on large experimental data sets.

The aim of the study is to develop and justify a methodology and software solution for automating the full cycle of modeling and forecasting large experimental data with reproducibility and scalability.

The methods of research are analytical, formal, machine learning methods, hyperparameter optimization, experimental.

The paper analyzes the problems of automating the ML process for large data, designs a solution architecture in Python using Pipeline and ColumnTransformer, the LightGBM ensemble, and experiment logging in MLflow. An automated pipeline has been implemented: data reading and economical typing, feature engineering, cross-validation model comparison, HPO on a subsample with subsequent retraining and early stopping, and final evaluation on a holdout sample. Experiments showed improvement in metrics when the training sample was increased to ~50k and a plateau in quality thereafter, as well as the heavy-tailed nature of the target variable (RMSE ≫ MAE).

Thus, the practical value is a reproducible ML experiment automation template that is suitable for expansion.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Зміст 6
Перелік умовних позначень, скорочень, термінів 8
Вступ 10
1 Аналіз проблематики області дослідження 12
1.1 Постановка проблеми 12
1.2 Життєвий цикл даних і процес дослідження 13
1.3 Огляд сучасних підходів до моделювання та прогнозування 15
1.4 Висновки до розділу 16
2 Автоматизація моделювання та прогнозування великих експериментальних даних 18
2.1 Постановка задачі та формальна модель 18
2.2 Характеристики даних, припущення та розбиття вибірки 20
2.3 Метрики та критерії вибору моделі 22
2.4 Підготовка даних 23
2.5 Інженерія ознак 24
2.6 Базові та ансамблеві моделі 25
2.7 Автоматизація налаштування гіперпараметрів 26
2.8 Управління експериментами та відтворюваність 28
2.9 Масштабування під великі дані 29
2.10 Архітектура автоматизованого конвеєра 30
2.11 Перевірка надійності та стабільності 31
2.12 Висновки до розділу 32
3 Рішення для автоматизації процесу моделювання та прогнозування великих

експериментальних даних 34
3.1 Функціональні вимоги до системи 34
3.2 Технічні вимоги 34
3.3 Використані інструменти 35
3.4 Набір даних 35
3.5 Реалізація 36
3.6 Результати роботи 45
3.6.1 Передумови та середовище 45
3.6.2 Запуск сценарію 46
3.6.3 Структура вихідних артефактів 47
3.6.4 Результати експериментів 48
Висновки 52
Перелік посилань 53
Додаток А Код 59

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Система – програмно-технічне рішення для автоматизації процесу моделювання та прогнозування великих експериментальних даних.

Великі експериментальні дані (ВЕД) – сукупність спостережень значного обсягу/швидкості надходження, що потребують спеціальних методів обробки, зберігання та моделювання.

Ознака (feature) – змінна, що характеризує об’єкт спостереження та використовується моделлю.

Інженерія ознак – процес створення похідних ознак (часові атрибути, геодистанції тощо) для підвищення якості моделі.

Пайплайн (Pipeline) – конвеєр з послідовних кроків: перетворення даних - модель.

ColumnTransformer – механізм scikit-learn для паралельної обробки різних підмножин стовпців різними трансформерами.

One-Hot Encoding (OHE) – кодування категоріальних змінних у бінарні індикатори.

Крос-валідація (CV) – метод оцінювання якості моделі шляхом багаторазових розбиттів вибірки.

Holdout – відкладена (незалежна) частина даних для підсумкової перевірки моделі.

HPO (Hyper-Parameter Optimization) – підбір гіперпараметрів моделі.

LightGBM – градієнтний бустинг на основі дерев рішень.

Early stopping – рання зупинка навчання за відсутності поліпшення на валідації.

RMSE / MAE / R^2 – метрики якості регресії: корінь середньоквадратичної помилки, середня абсолютна помилка, коефіцієнт детермінації.

MLflow – платформа для трекінгу експериментів (параметри, метрики, артефакти).

Haversine distance – сферична відстань між двома геоточками за широтою/довготою.

Kaggle – платформа спільних даних та змагань з аналітики/ML.

ВСТУП

У міру зростання обсягу та швидкості накопичення експериментальних даних у науці та промисловості виникає потреба не тільки в побудові точних моделей, але й в автоматизації всього процесу моделювання та прогнозування. Це включає все від підготовки даних та розробки функцій до валідації, вибору гіперпараметрів, оцінки та відтворюваного розгортання. Без стандартизованих конвеєрів зростає ймовірність виникнення помилок, втрачається час, збільшується ймовірність витоку валідації, а масштабування стає більш складним.

Актуальність теми полягає у створенні компактного, відтворюваного та масштабованого рішення, здатного працювати з великими експериментальними наборами даних і забезпечувати стабільні метрики якості при обмежених обчислювальних ресурсах.

З огляду на це, можна виділити наступні цілі і задачі дослідження.

Мета: розробити та обґрунтувати підхід до автоматизації процесу моделювання та прогнозування великих експериментальних даних.

Досягнення поставленої мети передбачає реалізацію наступних завдань:

- виконати аналіз методів моделювання та критеріїв оцінювання для великих даних;
- спроєктувати архітектуру автоматизованого конвеєра (Pipeline, ColumnTransformer, CV, HPO, логування);
- реалізувати програмне рішення з інженерією ознак (часові/геопросторові), підбором гіперпараметрів на підвибірці та ранньою зупинкою;
- провести експерименти на реальному датасеті з підвибірками 20k/50k/100k і порівняти моделі;
- оцінити якість (RMSE/MAE/R^2), інтерпретувати результати,

сформулювати рекомендації щодо подальшого поліпшення та масштабування.

Об’єкт дослідження: процеси побудови та оцінювання моделей для великих експериментальних даних.

Предмет дослідження: методи та засоби автоматизації ML-конвеєра.

Методи дослідження: аналітичний, формальний, методи машинного навчання, оптимізація гіперпараметрів, експериментальний.

Структурно робота містить оглядовий розділ, розділ з конкретизацією та математичними формулами, а також розділ з архітектурою, програмною реалізацією та результатами експериментів.

1 АНАЛІЗ ПРОБЛЕМАТИКИ ОБЛАСТІ ДОСЛІДЖЕННЯ

1.1 Постановка проблеми

У сучасних експериментальних дослідженнях джерела даних, такі як сенсорні мережі, лабораторні установки, вимірювальні стенди, промислові лінії, генерують великі за обсягом, швидкі, різноманітні та мінливі інформаційні потоки. Ці характеристики, що мають назву «V-характеристики» (volume, velocity, variety, variability), відрізняють «великі дані» від традиційних наборів і вказують на необхідність в масштабованих архітектурах зберігання, обробки і автоматизації побудови моделей і прогнозів на всіх етапах – від приймання сирих потоків до відтворюваних результатів [3, 12].

Особливістю великих експериментальних даних є їх походження з фізичних вимірювань. Зазвичай дані отримані із таких вимірювань мають невизначеність і похибки (випадкові та систематичні), які випливають із характеристик датчиків, процедур калібрування та умов експерименту. Згідно з керівництвом [7] невизначеність вимірювання – це параметр, пов’язаний з результатом, який характеризує розсіювання значень, що можуть бути обґрунтовано приписані вимірюваній величині. Ігнорування невизначеності призводить до хибної оцінки точності моделей і ненадійних прогнозів, а тому оцінювання та пропагування мають бути складовою процесу моделювання.

Важливою вимогою до наукових даних є відтворюваність. Для цього необхідно формально фіксувати походження даних, а саме: хто і коли їх створив, які інструменти застосовувалися, які перетворення виконані, з яких джерел вони агреговані. Рекомендація [16] задає технологічно нейтральну модель сутностей, активностей та агентів і відношень між ними (wasGeneratedBy, used, wasDerivedFrom), що забезпечує трасування життєвого циклу експериментальних наборів.

У термінах штучного інтелекту [4] AI-система є інженерною системою,

яка генерує контент, прогнози, рекомендації або рішення для заданих цілей, і тому прогнозування є базовим типом виходу. Стандарт також уніфікує терміни машинного навчання (контрольоване, неконтрольоване, напівконтрольоване, з підкріпленням) і типи задач (регресія, класифікація, кластеризація), що важливою складовою коректної постановки експериментів з моделювання.

Якщо експериментальні дані містять персональну інформацію, то на них поширюються правові обмеження, зокрема вимоги Загального регламенту про захист даних (GDPR) [2]. Вони повинні також бути враховано під час проєктування процесів, бо впливають на способи зберігання, анонімізації та обміну даними.

Отже, «великі експериментальні дані» поєднують технологічні виклики (масштаб і швидкість), методологічні вимоги (облік невизначеності) та процесні аспекти (відтворюваність, відповідність нормативам). Саме тому автоматизація моделювання та прогнозування розглядається як необхідна умова отримання надійних і відтворюваних результатів.

1.2 Життєвий цикл даних і процес дослідження

Практичною рамкою для організації робіт із даними є процес CRISP-DM [10].

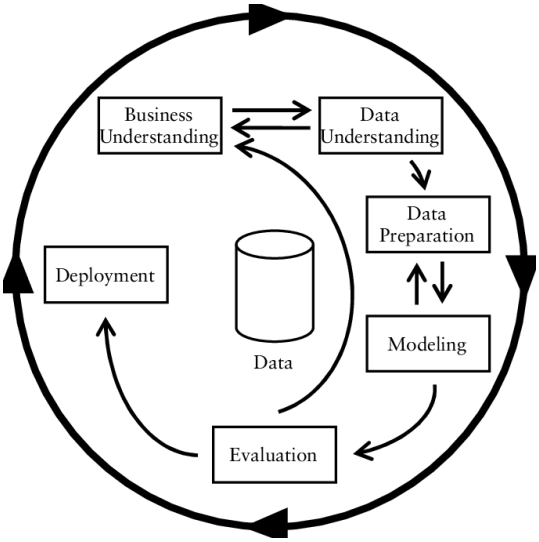


Рисунок 1.1 – CRISP-DM process model [10]

У контексті великих експериментальних даних етап data understanding включає аналіз джерел і умов вимірювань, перевірку протоколів калібрування та наявних метаданих, data preparation – інтеграцію потоків, синхронізацію часових міток, очищення, роботу з пропусками та аномаліями, приведення до узгоджених одиниць. Етап deployment включає інтеграцію моделі в конвеєри обробки та її подальший моніторинг [10].

Для опису життєвого циклу AI/ML-рішень використовується [5], який формалізує компоненти систем на базі ML і їх взаємодії, а саме: керування даними, навчання, валідація, розгортання, експлуатація і перенавчання. Така рамка полегшує автоматизацію конвеєрів, визначення інтерфейсів між підсистемами, а також розподіл ролей у команді.

Якість даних є важливим аспектом і має підтримуватися протягом усього життєвого циклу. Серія [6] встановлює узгоджені терміни, модель якості та вимірники, а також процесний фреймворк керування якістю. Це дозволяє вводити формальні перевірки заповненості, узгодженості, своєчасності, достовірності та відсутності систематичних зміщень і робити ці перевірки частиною автоматизованих конвеєрів.

Для забезпечення відтворюваності всі артефакти (сирі дані, проміжні таблиці, моделі, конфігурації навчання, метрики) мають супроводжуватися описами походження згідно з [16] та версіонуватися. Це дає змогу відтворити повний шлях отримання результату, мінімізувати ризик відхилення налаштувань і пояснювати розбіжності між запусками.

Важливим аспектом є врахування невизначеності вимірювань [7] у підготовці й моделюванні, а саме: необхідно описувати моделі вимірювань, оцінювати стандартні невизначеності вхідних величин і пропагувати їх до вихідних показників моделі (похибок, довірчих інтервалів), що особливо важливо для прийняття інженерних рішень.

Таким чином, життєвий цикл даних та ML-систем має бути організований так, щоб забезпечити безперервність якості, простежуваність і відтворюваність,

щоб створити основу для подальшої автоматизації.

1.3 Огляд сучасних підходів до моделювання та прогнозування

Методи, що застосовуються для моделювання та прогнозування на експериментальних даних можна умовно поділяють на три групи: класична статистика, машинне навчання та стрімінгові підходи.

Класична статистика охоплює лінійні та нелінійні регресійні моделі, методи локальної апроксимації (LOESS/LOWESS), а для часових рядів – моделі з урахуванням тренду, сезонності та автокореляції (AR/ARMA/ARIMA тощо). Ці методи добре інтерпретовані, дають чіткі припущення щодо даних і забезпечують прозорість оцінювання похибок (NIST/SEMATECH e-Handbook). Застосовуються при виконані умов стаціонарності або можливості коректної трансформації даних [4, 5].

Машинне навчання [4] включає контрольовані підходи для регресії та класифікації, неконтрольовані – для кластеризації та зниження розмірності, а також напівконтрольовані та методи з підкріпленням. Перевагою ML є здатність моделювати нелінійні залежності у високовимірних і гетерогенних даних. Серед обмежень можна виділити чутливість до якості даних, ризик перенавчання та потреба в ретельній валідації, а також у належній комунікації невизначеності прогнозів [4].

Стрімінгове й пакетне моделювання надає компонент «швидкості», через що частина задач вирішується у пакетному режимі (офлайн-навчання на накопичених вибірках) або в потоку (онлайн-оновлення, адаптація до concept drift) [4, 5]. Концептуальна рамка [5] дозволяє однаково описувати обидві парадигми в межах життєвого циклу.

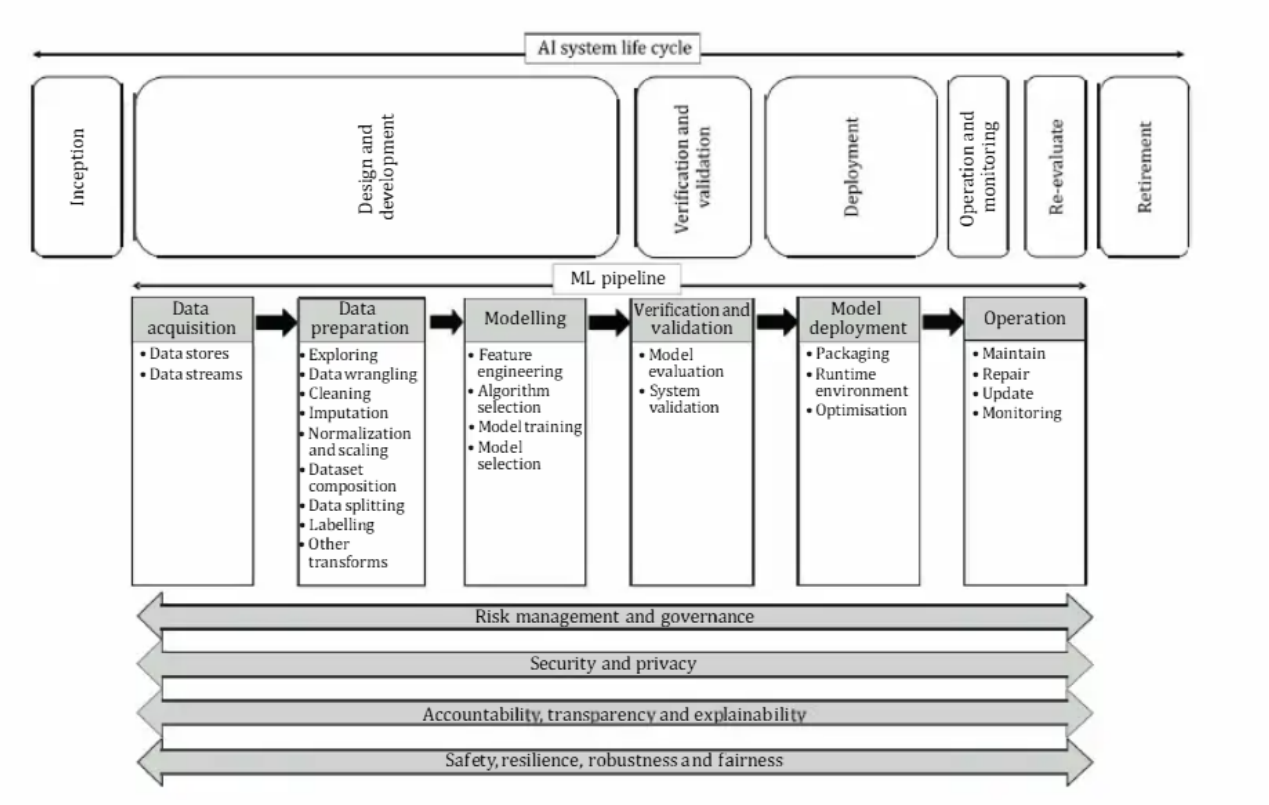


Рисунок 1.2 – ML/AI life circle [5]

Також для оцінювання моделей необхідно спиратися на відповідні метрики: для регресії – MAE, RMSE, MAPE, для класифікації – точність, повнота, F1-міра, ROC-AUC.

Якість і придатність даних є передумовою будь-якого з вищенаведених підходів. Серія [6] пропонує уніфіковані моделі якості та вимірники, які дозволяють прозоро звітувати про придатність наборів до конкретної аналітичної або ML-мети та вбудовувати контроль якості безпосередньо в конвеєри обробки.

1.4 Висновки до розділу

Основуючись на наведених даних можна зробити проміжний висновок, а саме те, що незалежно від вибору підходу (статистичний, ML чи стрімінгоий) надійність результатів визначається не лише алгоритмом, а й самим процесом:

керованою якістю даних, відстеженням походження та явним урахуванням невизначеності. Це задає вимоги до архітектури і прямо підводить до задач автоматизації.

2 АВТОМАТИЗАЦІЯ МОДЕЛЮВАННЯ ТА ПРОГНОЗУВАННЯ ВЕЛИКИХ ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ

2.1 Постановка задачі та формальна модель

Сучасні системи аналітики даних дедалі частіше використовують формалізовані підходи до побудови моделей, що надає можливість автоматизувати процеси навчання та оцінювання якості на великих вибірках. У межах регресійного підходу маємо вибірку $\{(x_i, y_i)\}_{i=1}^{n}$, де $x_i \in \mathbf{R}^d$ – вектор ознак, а $y_i \in \mathbf{R}$ – цільова змінна. Завданням є побудова відображення $f(x;\theta)$, параметри θ якого підбираються шляхом мінімізації середніх втрат на навчальній вибірці [26].

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, f(x_i;\theta)\right).$$

Для кількісної оцінки похибки прогнозу в регресії застосовуються базові метрики [26, 41, 42, 44, 45].

Розглянемо їх.

MSE (середньоквадратична помилка) – міра якості моделі регресії, яка являє собою середнє квадратичне значення помилок, тобто середнє квадратичне значення різниці між прогнозованими та фактичними значеннями [26, 42].

$$\mathbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 .$$

RMSE (корінь із MSE) – міра, яка обчислює квадратний корінь із середнього квадратичного значення помилок моделі. Вона вимірює середню

величину помилок, причому одиниці вимірювання відповідають одиницям цільової змінної [26, 45].

$$\mathbf{RMSE} = \sqrt{\mathbf{MSE}}$$

MAE (середня абсолютна помилка) – метрика ризику, що відповідає очікуваному значенню абсолютної помилки. Вона вимірює середню абсолютну різницю між прогнозованими та фактичними значеннями.

MAE є менш чутливою до викидів порівняно з MSE (середньоквадратичною похибкою), оскільки не підносить різницю до квадрату. Одиниці вимірювання MAE відповідають одиницям цільової змінної [26, 41].

$$\mathbf{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left| y_i - \hat{y}_i \right|$$

Коефіцієнт детермінації R^2 – частка дисперсії залежної змінної, яку можна передбачити з незалежних змінних моделі. Він являє собою міру того, наскільки добре спостережувані результати реплікуються моделлю [26, 45].

$$R^2 = 1 - \frac{\sum\left(y_i - \widehat{y_i}\right)^2}{\sum(y_i - \bar{y})^2}$$

Об’єктивність оцінювання забезпечується перехресною перевіркою (K-Fold cross-validator): вибірка ділиться на K непересічних частин, на кожній ітерації модель навчається на $K-1$ частинах і перевіряється на відкладеній. Середнє значення метрики по всіх фолдах вважають оцінкою узагальнювальної здатності [18, 47]. Схему процесу подано на рисунку 2.1.

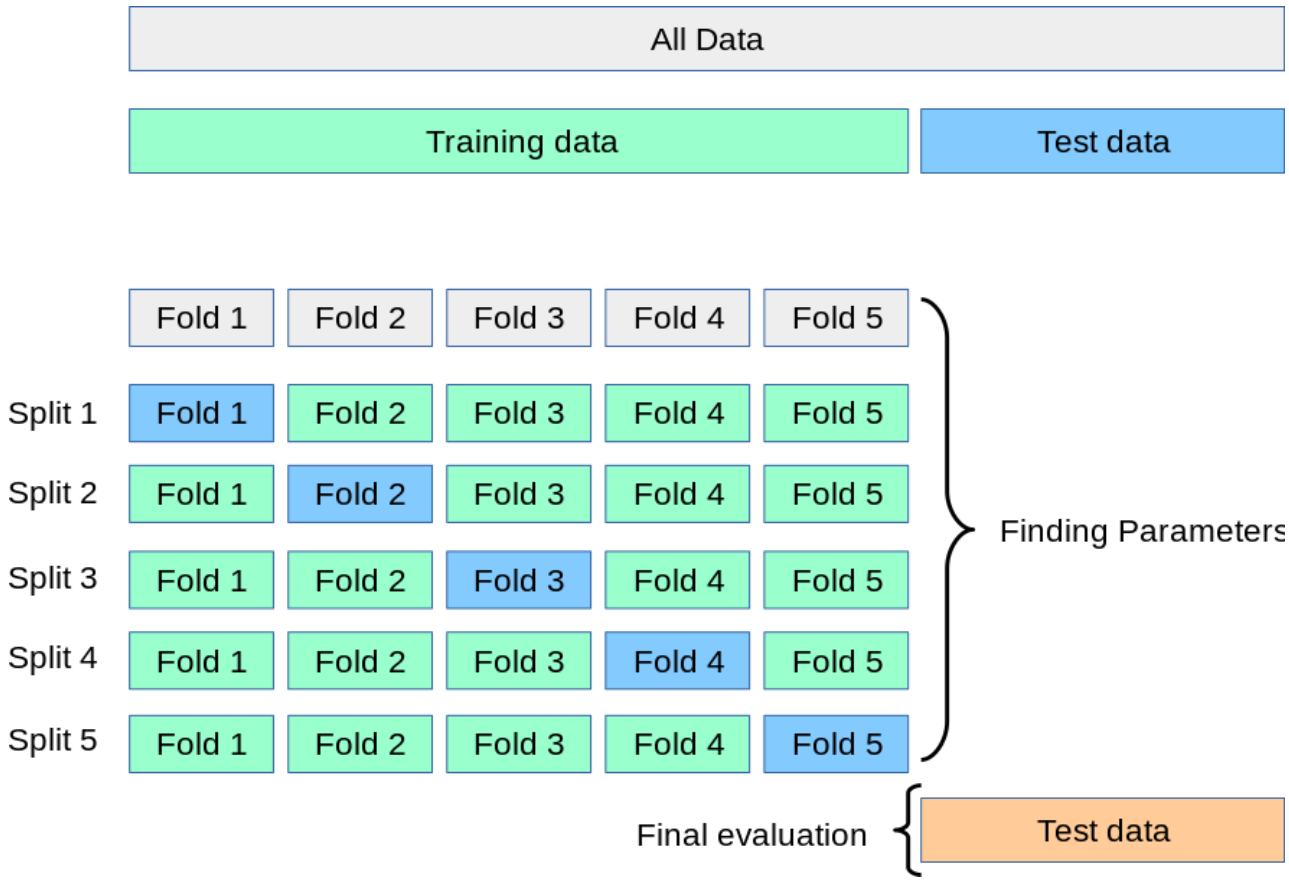


Рисунок 2.1 – Схема K-fold перехресної перевірки [18]

2.2 Характеристики даних, припущення та розбиття вибірки

Для табличних регресійних даних припускають приблизну незалежність спостережень, а також стаціонарність процесу збору даних у межах навчальної/валідаційної вибірок.

На практиці необхідно враховувати пропуски, викиди, відмінності масштабів ознак та можливе відхилення розподілів між тренувальною й тестовою частинами. До типових операцій підготовки відносяться імпутація пропусків, масштабування числових полів, кодування категоріальних змінних, їх необхідно виконувати всередині циклу валідації, аби уникнути витоку інформації.

Рекомендованим механізмом інтеграції підготовки є Pipeline у поєднанні з ColumnTransformer, що забезпечує єдиний об’єкт «дані -> прогноз» і коректну

роботу з cross_val_score/cross_validate [17, 31, 48].

Схематично автоматизований конвеєр подано на рисунку 2.2.

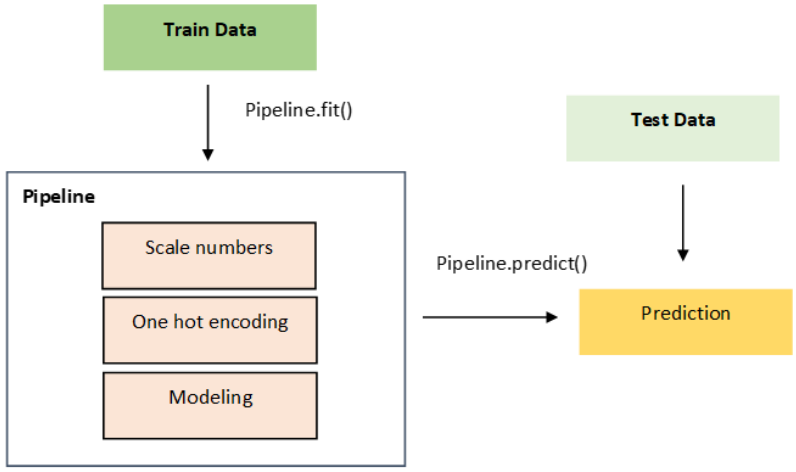


Рисунок 2.2 – Узагальнена схема автоматизованого конвеєра (Pipeline + ColumnTransformer)

Для розбиття вибірки в регресії використовують KFold, який повертає індекси тренувальних і валідаційних підмножин без стратифікації, яка є більш актуальною для класифікації [47]. Якщо дані мають явний часовий порядок, то застосовується TimeSeriesSplit, який унеможливлює «підглядання в майбутнє» за рахунок розсувних вікон тренування-валідації [18]. У цій роботі фокус йде на звичайну табличній регресії, а тому TimeSeriesSplit розглядається як альтернатива для часових рядів.

Під час побудови наборів train/validation/test важливо фіксувати випадкові стани генераторів (random_state) та версії бібліотек для повної відтворюваності експериментів, що особливо критично при великих даних, коли повторні запуски дорогі за ресурсами та часом [17, 18].

2.3 Метрики та критерії вибору моделі

Оцінювання якості здійснюється за єдиною метрикою – RMSE як головної, та додатково – MAE та R^2 [26, 41, 42, 44, 45]. Для чесного порівняння кандидатів використовується однакове розбиття CV і спільний параметр scoring.

Практичним інструментом є model_selection.cross_validate, який повертає розподіл метрик по фолдах і часові витрати, що дозволяє врахувати не лише точність, а також продуктивність [18, 46].

Критеріями селекції є мінімальне середнє значення основної метрики на CV за умови стандартного відхилення між фолдами.

Для орієнтиру обрано базову модель – DummyRegressor, яка відтворює прості евристики (середнє/медіану тощо) та задає нижню межу якості. Модель, що не перевершує «даммі»-базу, потребує перегляду ознак або гіперпараметрів [18, 32].

Узагальнену процедуру вибору моделі подано на рисунку 2.3.

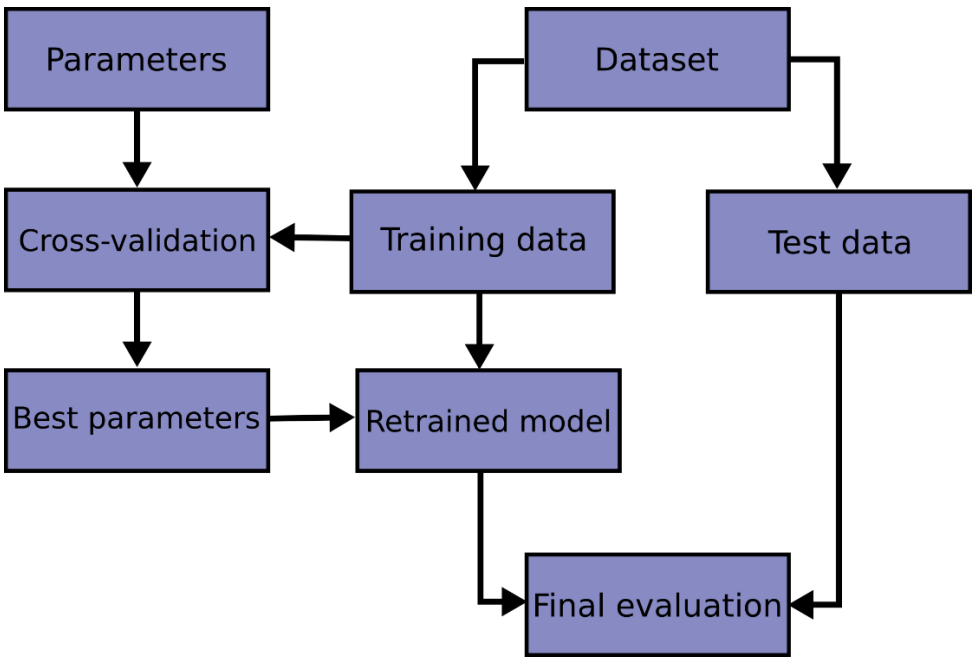


Рисунок 2.3 – Процедура автоматизованого відбору моделі за CV-метрикою (cross_validate) [18]

2.4 Підготовка даних

Автоматизація препроцесингу забезпечує відтворюваність і виключає ручні кроки, що можуть призвести до витоку даних. У типових табличних задачах регресії послідовність включає імпутацію пропусків, масштабування числових ознак і кодування категоріальних змінних, об'єднаних у єдиний конвеєр.

Базовим інструментом є Pipeline, який послідовно застосовує перетворювачі та завершується оцінювачем [48]. Кроки можуть кешуватись через параметр memory, що пришвидшує повторні підбори гіперпараметрів. Для неоднорідних за типами стовпців застосовується ColumnTransformer, який дозволяє паралельно задати різні гілки препроцесингу для числових і категоріальних ознак [31].

Для заповнення відсутніх значень використовується SimpleImputer (середнє/медіана/найчастіше значення/константа) згідно з обраною стратегією на кожному стовпці окремо. Важливим аспектом є те, що імпутація вчиться лише на тренувальних фолдах і застосовується до валідації/тесту в межах того ж конвеєра [23, 40].

Для стандартного масштабування використовується StandardScaler [30]

$$z = \frac{x - \mu}{\sigma},$$

де μ та σ оцінюються на тренувальних даних.

Для даних із викидами доцільним є застосування RobustScaler, що центрує за медіаною і масштабує за інтерквартильним розмахом [30]

$$z = \frac{x - \mathbf{median}}{\mathbf{IQR}}:.$$

Також для розріджених матриць слід уникати центрування у StandardScaler щоб не порушувати розрідженість [30].

Для кодування категоріальних ознак застосовується OneHotEncoder з коректним поводженням із невідомими категоріями (параметр handle_unknown) та можливістю повертати розріджений формат. Цей крок так само інтегрується у конвеєр, щоб статистики кодування не «підглядали» у валідаційні/тестові дані [49].

Для запобігання витоку даних усі перетворення (імпутація, масштабування, кодування) є частиною Pipeline і навчаються лише на тренувальних фолдах, що також є рекомендацією Common pitfalls [17].

2.5 Інженерія ознак

Інженерія ознак підвищує інформативність вхідного простору, але повинна виконуватися всередині конвеєра задля коректної оцінки на CV.

Виділяють наступні дії з ознаками, розглянемо їх.

Оператор PolynomialFeatures генерує поліноміальні комбінації та взаємодії до вказаного степеня, а опція interaction_only=True дозволяє створювати лише добутки без степенів. Застосування виправдане для linear/ridge моделей, коли є вирогідність помітних нелінійностей [50].

Базовий фільтр VarianceThreshold прибирає ознаки з нульовою або низькою дисперсією, підходить як попередній крок, що зменшує виміри [39].

SelectKBest застосовує статистику на кожній ознаці окремо. Для регресії поширені критерії: кореляційний f_regression (лінійна залежність, повертає F-статистику/p-значення) та mutual_info_regression (непараметрична взаємна інформація для виявлення нелінійних залежностей). Відбір відбувається за топ-k балами [36-38].

Функція permutation_importance оцінює зменшення якості моделі при

випадковій перестановці значень окремої ознаки, обчислюється на відкладених даних, а не на тренуванні [29].

Всі вищеописані кроки (генерація, фільтрація, відбір) інтегруються в Pipeline/ColumnTransformer, що гарантує відтворюваність і коректне поєднання з перехресною перевіркою. За потреби обчислювально важкі перетворення кешуються [31, 48].

2.6 Базові та ансамблеві моделі

Розглянемо класичні оцінювачі для табличної регресії, а саме: прості лінійні моделі, ансамблеві методи та бустинги.

Наведемо перелік лінійних моделей.

LinearRegression (OLS) мінімізує суму квадратів залишків і слугує референтною базою [25].

Ridge (L2-регуляризація) стабілізує оцінки ваг і зменшує дисперсію [25]:

Lasso (L1-регуляризація) сприяє розрідженню вектору коефіцієнтів [25]:

ElasticNet поєднує L1/L2, контроль відбувається через l1_ratio [25].

Далі розглянемо та опишемо нелінійні дерева рішень та ансамблі.

DecisionTreeRegressor моделює локальні розбиття простору ознак, схильний до перенавчання без обмежень глибини/листів [19].

RandomForestRegressor усереднює ансамбль дерев, побудованих на бутстреп-підвибірках і підмножинах ознак, що зменшує дисперсію [35].

GradientBoostingRegressor нарощує адитивну модель слабких дерев у напрямку негативного градієнта обраної втрати [33].

Розглянемо зовнішні бустинги.

LightGBM (LGBMRegressor) – градієнтний бустинг на гістограмних деревах, орієнтований на високу швидкість/ефективність і роботу з великими даними. Він підтримує паралелізм і GPU. Саме його будемо брати за основний бустинг при реалізації [9].

XGBoost (XGBRegressor) – популярна альтернатива з широким набором параметрів і підтримкою GPU [51].

2.7 Автоматизація налаштування гіперпараметрів

Автоматизований підбір гіперпараметрів (HPO) забезпечує відтворюваний і об’єктивний вибір конфігурацій моделей у межах єдиного конвеєра. Базові стратегії, такі як вичерпний перебір та випадковий пошук, реалізовано у scikit-learn як GridSearchCV та RandomizedSearchCV [21, 22].

Перший перебирає всі комбінації з решітки параметрів, а другий семплює задану кількість комбінацій із наперед визначених розподілів, що дозволяє корректно керувати незалежно від розмірності простору гіперпараметрів. Обидва підходи інтегруються з крос-перевіркою та підтримують єдиний критерій scoring [21, 22].

Формально завдання HPO визначається як мінімізація оцінки помилки на перехресній перевірці:

$$\theta = \arg\min_{\theta \in \Theta} \hat{R}_{CV}(\theta), \hat{R}_{CV} = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{(\square \mid V_k \mid \square)} \Sigma_{i \in V_k} L\left(y_i, f_\theta^{(-k)}(x_i)\right),$$

де Θ — простір гіперпараметрів, V_k — валідаційний фолд.

На практиці це реалізується за допомогою GridSearchCV/RandomizedSearchCV з заданими param_grid/param_distributions і спільним CV [21, 22].

Для прискорення HPO застосовується паралелізація, а саме параметр n_jobs, який керує кількістю воркерів у бекенді joblib. Водночас потоковий паралелізм у низькорівневих бібліотеках (BLAS/LAPACK, OpenMP) контролюється змінними середовища/threadpoolctl. Це дозволяє узгоджено керувати обчислювальними ресурсами під час сіткового чи випадкового пошуку [28].

У бустингових моделях доцільним є використання раннього зупинення, тобто тренування переривається, якщо метрика на валідаційному наборі не покращується щонайменше на min_delta протягом stopping_rounds. У LightGBM це реалізовано через офіційний колбек lightgbm.early_stopping(...), що додається до списку callbacks при навчанні [9].

Як розширення – байєсівська/послідовна оптимізація (наприклад, Optuna), де ядром є Study.optimize(...), а керування стохастикою та відсікання «слабких» прогонів забезпечується за допомогою samplers і pruners (з медіанним правилом, тощо). Для відтворюваності також важливо фіксувати seed у семплерах і враховувати паралельні запуски [14].

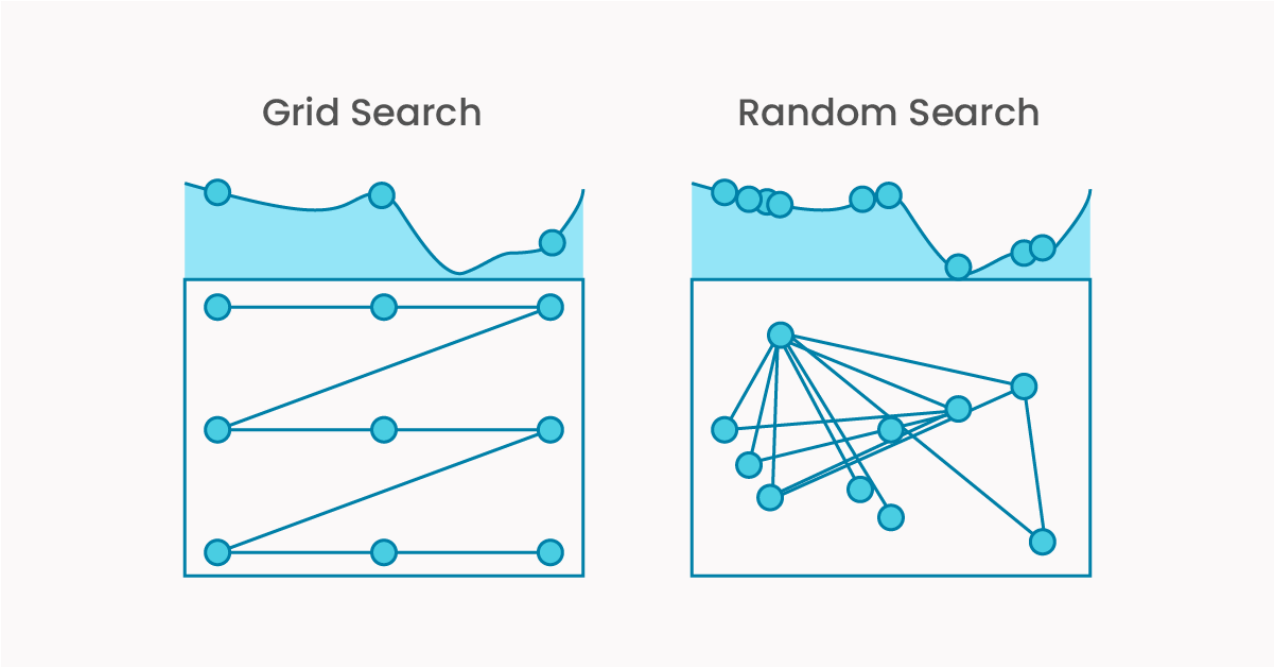


Рисунок 2.4 – GridSearchCV/RandomizedSearchCV

2.8 Управління експериментами та відтворюваність

Систематичний трекінг експериментів спрощує порівняння моделей і забезпечує повну відтворюваність. У роботі застосовуємо MLflow Tracking для логування параметрів, версій коду, метрик та артефактів (графіки, моделі) [11].

Інтерфейс надає API та Web-UI для перегляду/порівняння запусків. Для колективного керування моделями використовується Model Registry (версіонування, стадії «Staging/Production», лінійка походження) [11].

Частина об’єктів scikit-learn є стохастичними (спліттери CV, ліс, тощо), їхня випадковість контролюється параметром random_state. Згідно з рекомендаціями, фіксуємо random_state у всіх релевантних компонентах конвеєра (розбиття, моделі, HPO), щоб забезпечити відтворювані результати [17].

Версії залежностей фіксуються через pip freeze > requirements.txt із подальшим збереженням файлу як артефакта експерименту [8]. Моделі/пайплайни зберігаються згідно до офіційних рекомендацій scikit-learn

(ONNX, skops, pickle/joblib), з урахуванням безпекових та експлуатаційних обмежень кожного способу [27].

2.9 Масштабування під великі дані

Під «великими даними» для табличної регресії розуміємо наступні сценарії: вибірка не вміщується в оперативну пам’ять, обчислення тривають неприйнятно довго без паралелізації, необхідна обробка потоків/пакетів.

Нижче розглянемо практичні стратегії, що сумісні з нашим конвеєром.

Інкрементальне (out-of-core) навчання – оцінювачі з API partial_fit (SGD-моделі, деякі наївні баєсівські, тощо) дозволяють навчатися на міні-пакетах, не завантажуючи весь датасет до пам’яті. Для збереження спільного простору ознак у потоці можуть застосовуватись хешуючі перетворювачі [24].

Читання й обробка даних chunk-ми – у pandas.read_csv параметри chunksize/iterator повертають ітератор по частинах файлу, що знижує пікове споживання пам’яті. Для дуже великих масивів використовують numpy.memmap – пам’ять-відображені масиви для доступу до фрагментів без повного завантаження. Ці інструменти органічно поєднуються з partial_fit попередньої стратегії [13, 15].

Паралелізація та розподілення – для локального масштабу застосовується n_jobs (бекенд joblib), для кластерного – бекенд Dask, який дозволяє масштабувати Grid/RandomizedSearch та інші сумісні задачі без істотних змін коду. Для потокового інкрементального навчання існують компоненти Dask-ML [1].

Зниження розмірності на потоках. Для дуже високих значень d застосовується IncrementalPCA, що реалізує покрокову SVD-обробку даних блоками – придатно для out-of-core сценаріїв і як частина Pipeline.

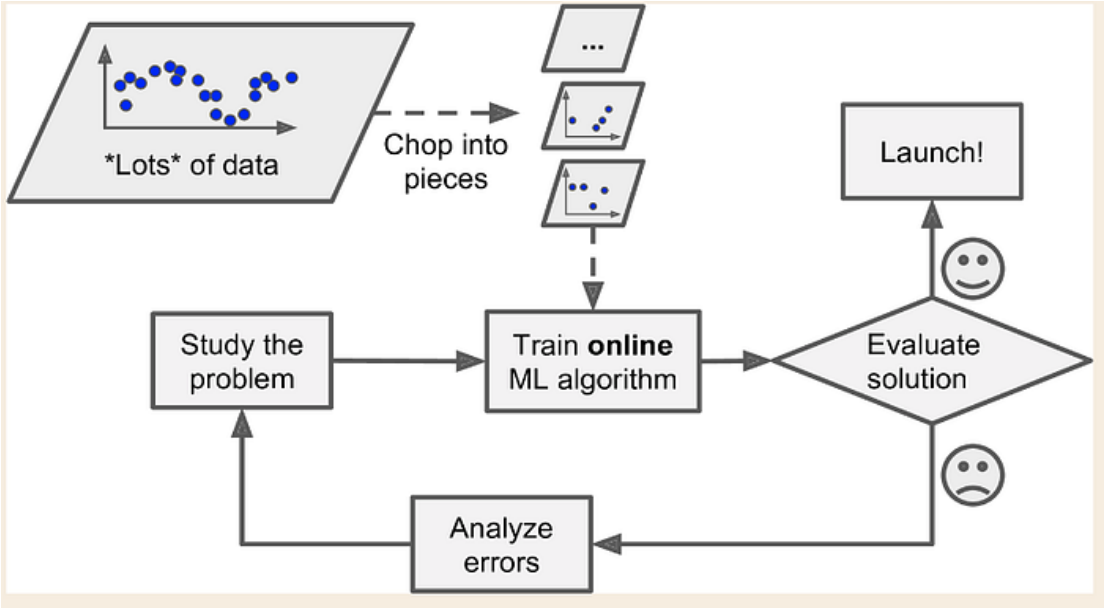


Рисунок 2.5 – Out-of-core

2.10 Архітектура автоматизованого конвеєра

Автоматизований конвеєр забезпечує відтворюваність, контроль якості та скорочення ручних дій у повному циклі «дані -> модель -> інференс».

Узагальнений потік містить наступні етапи:

1) завантаження та валідація даних;
2) препроцесинг;
3) інженерія ознак;
4) навчання та підбір гіперпараметрів;
5) оцінювання на CV;
6) реєстрація та версіонування моделі;
7) інференс (batch/online);
8) моніторинг.

Кожен крок реалізують у складі єдиного об’єкта Pipeline із розгалуженням перетворень через ColumnTransformer, що унеможливлює витоки даних і спрощує повторне використання у тренуванні та на prod [31, 48].

У верхніх гілках ColumnTransformer виконуються імпутація, масштабування числових ознак і кодування категоріальних, після за потреби інженерія ознак і фінальний оцінювач. Конвеєр слугує єдиним «естімейтором» для крос-перевірки та HPO: блок GridSearchCV/RandomizedSearchCV загортає увесь Pipeline, підбираючи як гіперпараметри моделі, так і параметри препроцесингу [21, 22, 31, 48].

Для чесного порівняння моделей використовується cross_validate з єдиним cv і scoring. На кожному прогоні логуються метрики, параметри та артефакти (графіки, конфігурації) у MLflow Tracking. Найкращий пайплайн фіксується у Model Registry зі стадіями «Staging/Production» [11, 26].

Для пакетного передбачення завантажується зареєстрована модель (mlflow.sklearn.load_model) і застосовується як звичайний Pipeline.predict. Online-інференс реалізуюється через сервісування моделі, при цьому артефакт залишається тим самим – серіалізований пайплайн [18, 26].

Усі стохастичні компоненти (розбиття, ансамблі) фіксуються через random_state, а залежності – через requirements.txt, який зберігається як артефакт експерименту [8, 17].

2.11 Перевірка надійності та стабільності

Надійність оцінюється повторними експериментами та додатковими статистичними процедурами.

Для зменшення випадковості розбиття використовується RepeatedKFold, який багаторазово повторює K-fold із різними seed-значеннями, а для випадкового валідаційного контролю – ShuffleSplit [47]. Подача результатів у форматі середнє ± стандартне відхилення по фолдах дає уявлення про стабільність.

Тест permutation_test_score порівнює реальну якість моделі з нульовим

розподілом, отриманим перестановками цілі, якщо значення p-value замале, то модель ловить сигнал, а не шум [47].

learning_curve показує, як змінюється якість зі зростанням обсягу даних (діагностика браку даних/перенавчання), а validation_curve демонструє вплив одного гіперпараметра на якість (пошук балансу bias–variance) [47].

Для оцінки довірчих інтервалів метрик застосовуються бутстреп-ресемплінг CV-скорів через sklearn.utils.resample (повторне вибіркове підсумовування з поверненням), що дає діапазон варіації оцінки якості без додаткових тренувань моделі [47].

Аналіз помилок ґрутується на розподілі залишків $e = y - \hat{y}$, графіках «помилка vs. прогноз/ознака», групуванням помилок за квантилями прогнозу. Фінальна перевірка відбувається на відкладеному тесті з фіксованим train_test_split [46б 47].

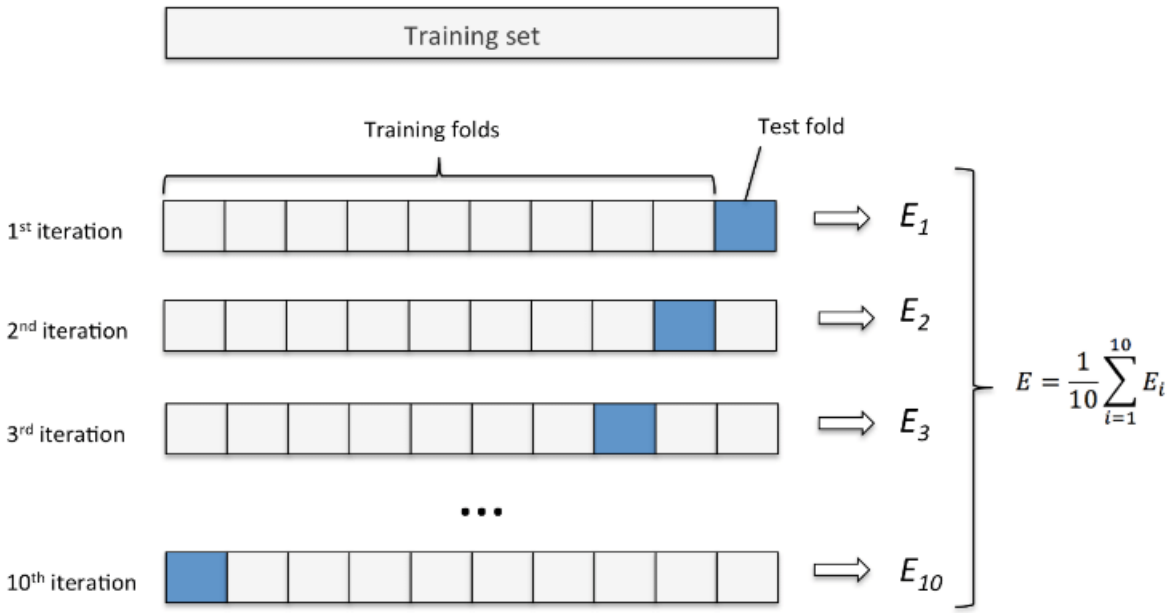


Рисунок 2.6 – RepeatedKFold

2.12 Висновки до розділу

У цьому розділі сформовано повний автоматизований конвеєр моделювання табличної регресії: від уніфікованого препроцесингу (Pipeline + ColumnTransformer) через крос-валідаційне оцінювання та HPO до реєстрації найкращого пайплайна в MLflow. Запропоновано процедури перевірки надійності (RepeatedKFold/ShuffleSplit), статистичної значущості (перестановочний тест), діагностики узагальнювальної здатності (learning/validation curves) та отримання довірчих оцінок метрик (бутстреп).

Сукупність цих практик забезпечує відтворюваність, об’єктивність і масштабованість рішень, що критично важливо для роботи з великими експериментальними даними.

3 РІШЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСУ МОДЕЛЮВАННЯ ТА ПРОГНОЗУВАННЯ ВЕЛИКИХ ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ

3.1 Функціональні вимоги до системи

Система повинна надавати наступні можливості:

- завантажувати сирі CSV-даних та здійснювати первинну валідацію схеми (типи, пропуски, діапазони);
- автоматизовувати препроцесинг у складі конвеєра: імпутація, масштабування числових ознак, кодування категоріальних;
- застосовувати інженерію ознак (часові та геопросторові характеристики) всередині конвеєра;
- порівнянювати базові і ансамблеві регресори під єдиною CV-схемою за головною та другорядними метриками;
- автоматично підбирати гіперпараметри (RandomizedSearchCV/GridSearchCV) із паралелізацією n_jobs;
- логувати параметри/метрики/артефакти і реєструвати найкращі моделі (MLflow Tracking/Model Registry);
- зберігати серіалізований пайплайн та пакетний інференс на нових даних.

3.2 Технічні вимоги

Для розгортання та запуску на пристрої повинні бути встановлені наступні залежності:

- Python 3.x як основна мова, що забезпечує середовище;
- pandas, numpy для роботи з CSV/масивами;

- scikit-learn для конвеєрів, метрик та CV;
- lightgbm як основний бустинг із ранньою зупинкою;
- mlflow для трекінгу і реєстру.

З боку апаратного забезпечення рекомендується мати >= 16 ГБ RAM для «великих» вибірок та багатоядерний CPU. Щодо GPU, то це опційно для LightGBM.

Для відтворюваності використовується фіксація random_state, а також версії залежностей у requirements.txt, що зберігаються як артефакт у MLflow.

3.3 Використані інструменти

При розробці було використано наступні інструменти:

- Pipeline – суцільний об’єкт «препроцесинг -> модель», який можна крос-валідувати як єдине ціле [48];
- ColumnTransformer – паралельні гілки для числових/категоріальних ознак у гетерогенних таблицях [31];
- SimpleImputer, OneHotEncoder – офіційні засоби імпутації та кодування категорій [40, 49];
- haversine_distances – геометрична відстань на сфері [43];
- GridSearchCV/RandomizedSearchCV – автопідбір гіперпараметрів з CV і паралелізацією [21, 22];
- LGBMRegressor + early_stopping – швидкий бустинг на гістограмах для табличних даних [9];
- MLflow Tracking/Model Registry – логування запусків і версіонування prod-моделей [11].

3.4 Набір даних

Обрано відкритий набір New York City Taxi Trip Duration [8]. Він містить записи поїздок таксі з наступними атрибутами: час подачі, кількість пасажирів, геокоординати посадки/висадки, службові маркери тощо. Це реальні операційні дані з великим n , придатні для демонстрації масштабу процесу.

id	vendor_id	pickup_dat...	dropoff_da...	passenger...	pickup_lon...	pickup_lati...	dropoff_lo...	dropoff_lat...	store_and_...
id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.98215484619 1406	40.767936706542 969	-73.96463012695 3125	40.765602111816 406	N
id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.98041534423 8281	40.738563537597 656	-73.99948120117 1875	40.731151580810 547	N
id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.97902679443 3594	40.763938903808 594	-74.00533294677 7344	40.710086822509 766	N
id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.01004028320 3125	40.719970703125	-74.01226806640 625	40.706718444824 219	N
id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.97305297851 5625	40.793209075927 734	-73.97292327880 8594	40.782520294189 453	N
id0801584	2	2016-01-30 22:01:40	2016-01-30 22:09:03	6	-73.98285675048 8281	40.742195129394 531	-73.99208068847 6562	40.749183654785 156	N
id1813257	1	2016-06-17 22:34:59	2016-06-17 22:40:40	4	-73.96901702880 8594	40.757839202880 859	-73.95740509033 2031	40.765895843505 859	N
id1324603	2	2016-05-21 07:54:58	2016-05-21 08:20:49	1	-73.96927642822 2656	40.797779083251 953	-73.92247009277 3438	40.760559082031 25	N

Рисунок 3.1 – Дані [8]

3.5 Реалізація

Для початку необхідно забезпечити контроль середовища та відтворюваність. Робимо це за допомогою фіксації версій бібліотек та SEED (рис. 3.2) [8].

```
import sys
try:
    import sklearn
    import lightgbm as lgb
    import numpy
    import pandas
    print("PY:", sys.executable)
    print("sklearn:", sklearn.__version__, getattr(sklearn, "__file__", ""))
    print("lightgbm:", lgb.__version__)
    print("numpy:", numpy.__version__, "pandas:", pandas.__version__)
except Exception as _e:
    print("Version print warning:", _e)

SEED = 42
```

Рисунок 3.2 – Фіксація версій середовища

Так як основною метрикою виступає RMSE, то необхідно забезпечити уникнення залежності від версій sklearn, тому RMSE рахуємо як sqrt(MSE) і реєструємо як кастомний скорер для CV/HPO (рис. 3.3) [26, 41, 42, 44].

```
def rmse(y_true, y_pred) -> float:
    return float(np.sqrt(mean_squared_error(y_true, y_pred)))

rmse_scorer = make_scorer(rmse, greater_is_better=False)
```

Рисунок 3.3 – RMSE

Далі переходимо до інженерії ознак, де додаємо часові ознаки (година/день тижня/місяць/пікові години) та геодистанцію Haversine між точками посадки і висадки. Для цього створюємо клас TimeGeoFeatures (рис. 3.4) [43].

```python
class TimeGeoFeatures(BaseEstimator, TransformerMixin):
    def __init__(self,
                 datetime_col: str = "pickup_datetime",
                 pickup_lat: str = "pickup_latitude",
                 pickup_lon: str = "pickup_longitude",
                 dropoff_lat: str = "dropoff_latitude",
                 dropoff_lon: str = "dropoff_longitude"):
        self.datetime_col = datetime_col
        self.pickup_lat = pickup_lat
        self.pickup_lon = pickup_lon
        self.dropoff_lat = dropoff_lat
        self.dropoff_lon = dropoff_lon

    def fit(self, X: pd.DataFrame, y=None):
        return self

    @staticmethod
    def _haversine_km(lat1, lon1, lat2, lon2) -> np.ndarray:
        R = 6371.0  # кілометри
        lat1 = np.radians(lat1.astype(float))
        lon1 = np.radians(lon1.astype(float))
        lat2 = np.radians(lat2.astype(float))
        lon2 = np.radians(lon2.astype(float))
        dlat = lat2 - lat1
        dlon = lon2 - lon1
        a = np.sin(dlat / 2.0) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2.0) ** 2
        return 2.0 * R * np.arcsin(np.sqrt(a))

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()
        X[self.datetime_col] = pd.to_datetime(X[self.datetime_col], errors="coerce", utc=True)
        X["pickup_hour"] = X[self.datetime_col].dt.hour
        X["pickup_dow"] = X[self.datetime_col].dt.dayofweek
        X["pickup_month"] = X[self.datetime_col].dt.month
        X["is_weekend"] = (X["pickup_dow"] >= 5).astype(int)
        X["is_peak"] = X["pickup_hour"].isin([7, 8, 9, 16, 17, 18, 19]).astype(int)

        X["haversine_km"] = self._haversine_km(
            X[self.pickup_lat], X[self.pickup_lon],
            X[self.dropoff_lat], X[self.dropoff_lon]
        )
        return X
```

Рисунок 3.4 – Клас TimeGeoFeatures

Функція make_preprocessor відповідає за препроцесінг: передані числові ознаки імпутуються медіаною та масштабуються, категоріальні – OneHotEncoder. Повертає загорнуті в ColumnTransformer ознаки (рис. 3.5) [23, 30, 31, 48, 49].

```python
def make_preprocessor(numeric_features: List[str],
                      categorical_features: List[str]) -> ColumnTransformer:
    num = Pipeline([
        ("imputer", SimpleImputer(strategy="median")),
        ("scaler", StandardScaler()),
    ])
    cat = OneHotEncoder(handle_unknown="ignore", sparse_output=True)
    pre = ColumnTransformer(
        transformers=[
            ("num", num, numeric_features),
            ("cat", cat, categorical_features),
        ],
        remainder="drop",
        sparse_threshold=1.0,
    )
    return pre
```

Рисунок 3.5 – Функція make_preprocessor

Далі завантажуємо дані, для цього читаємо лише потрібні стовпці з економними типами, щоб прискорити I/O і зменшити RAM (рис. 3.6) [15].

```
usecols = [
    "vendor_id", "pickup_datetime", "passenger_count",
    "pickup_longitude", "pickup_latitude",
    "dropoff_longitude", "dropoff_latitude",
    "store_and_fwd_flag", "trip_duration"
]
dtype_map = {
    "vendor_id": "category",
    "passenger_count": "int16",
    "pickup_longitude": "float32",
    "pickup_latitude": "float32",
    "dropoff_longitude": "float32",
    "dropoff_latitude": "float32",
    "store_and_fwd_flag": "category",
    "trip_duration": "float32",
}
df = pd.read_csv(train_path, usecols=usecols, dtype=dtype_map)

if "trip_duration" not in df.columns:
    raise ValueError("train.csv не містить 'trip_duration'.")

base_columns = [
    "vendor_id", "pickup_datetime", "passenger_count",
    "pickup_longitude", "pickup_latitude",
    "dropoff_longitude", "dropoff_latitude",
    "store_and_fwd_flag",
]
missing = [c for c in base_columns if c not in df.columns]
if missing:
    raise ValueError(f"Відсутні колонки: {missing}")

X = df[base_columns].copy()
y = df["trip_duration"].astype("float32")
```

Рисунок 3.6 – Завантаження даних

Також застосовуємо чесний holdout-спліт для фінальної оцінки (рис. 3.7) [15].

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=cfg.test_size, random_state=SEED
)
```

Рисунок 3.7 – Holdout-спліт

Далі конструюємо по одному пайплайну для кожної моделі: Dummy, Ridge, RandomForest, LightGBM (рис. 3.8) [9, 25, 32, 35].

```
memory = joblib.Memory(location=cfg.cache_dir, verbose=0) if cfg.cache_dir else None
ridge = Ridge(random_state=SEED)
rf = RandomForestRegressor(n_estimators=300, n_jobs=-1, random_state=SEED)
lgbm = lgb.LGBMRegressor(
    random_state=SEED, n_estimators=500, learning_rate=0.1, num_leaves=63,
    min_child_samples=20, verbose=-1
)

pipe_dummy = Pipeline([("feat", feature_gen),
                       ("preprocess", preprocessor),
                       ("model", DummyRegressor(strategy="median"))],
                      memory=memory)
pipe_ridge = Pipeline([("feat", feature_gen),
                       ("preprocess", preprocessor),
                       ("model", ridge)],
                      memory=memory)
pipe_rf = Pipeline([("feat", feature_gen),
                    ("preprocess", preprocessor),
                    ("model", rf)],
                   memory=memory)
pipe_lgbm = Pipeline([("feat", feature_gen),
                      ("preprocess", preprocessor),
                      ("model", lgbm)],
                     memory=memory)
```

Рисунок 3.8 – Конструювання пайплайнів

Для крос-валідаційного порівняння кандидатів створюємо єдину функцію – evaluate_models_cv, що рахує CV-метрики (RMSE/MAE/R^2) для кількох пайплайнів і формує підсумкову таблицю (рис. 3.9) [46].

```python
def evaluate_models_cv(models: Dict[str, Pipeline],
                       X: pd.DataFrame,
                       y: pd.Series,
                       cv,
                       n_jobs: int = -1) -> pd.DataFrame:
    scoring = {
        "rmse": rmse_scorer,
        "mae": "neg_mean_absolute_error",
        "r2": "r2",
    }
    rows = []
    scores_dict: Dict[str, Dict[str, np.ndarray]] = {}
    for name, pipe in models.items():
        scores = cross_validate(
            pipe, X, y, cv=cv, scoring=scoring,
            n_jobs=n_jobs, return_train_score=False
        )
        scores_dict[name] = scores
        rows.append({
            "model": name,
            "rmse_mean": -np.mean(scores["test_rmse"]),
            "rmse_std": np.std(-scores["test_rmse"]),
            "mae_mean": -np.mean(scores["test_mae"]),
            "mae_std": np.std(-scores["test_mae"]),
            "r2_mean": np.mean(scores["test_r2"]),
            "r2_std": np.std(scores["test_r2"]),
            "fit_time_mean": np.mean(scores["fit_time"]),
            "score_time_mean": np.mean(scores["score_time"]),
        })
    return pd.DataFrame(rows).sort_values("rmse_mean")
```

Рисунок 3.9 – Функція evaluate_models_cv

Щоб уникнути надмірного часу на великому датасеті, HPO виконується на підмножині (задаємо параметром --hpo-sample-size при запуску), після чого найкращий пайплайн refit=True навчається на всьому train. Реалізуємо цю логіку за допомогою функції randomized_search_lgbm (рис. 3.10) [22, 28].

```python
def randomized_search_lgbm(pipe: Pipeline,
                           X_train: pd.DataFrame,
                           y_train: pd.Series,
                           cv,
                           n_iter: int = 20,
                           random_state: int = SEED,
                           n_jobs: int = -1) -> RandomizedSearchCV:
    params = {
        "model__n_estimators": [300, 500, 800],
        "model__num_leaves": [31, 63, 127],
        "model__max_depth": [-1, 8, 12],
        "model__learning_rate": [0.05, 0.1],
        "model__min_child_samples": [10, 20, 50],
        "model__subsample": [0.8, 1.0],
        "model__colsample_bytree": [0.8, 1.0],
        "model__reg_alpha": [0.0, 0.1],
        "model__reg_lambda": [0.0, 0.1, 1.0],
    }
    search = RandomizedSearchCV(
        estimator=pipe,
        param_distributions=params,
        n_iter=n_iter,
        scoring=rmse_scorer,
        cv=cv,
        n_jobs=n_jobs,
        verbose=1,
        random_state=random_state,
        refit=True,
    )
    return search
```

Рисунок 3.10 – Функція randomized_search_lgbm

Так як після HPO модель навчається ще раз на train із внутрішньою валідацією та early_stopping, що стабілізує кількість ітерацій бустингу створимо функцію fit_final_with_early_stopping, яка буде відповідати за цей процес (рис. 3.11) [9].

```python
def fit_final_with_early_stopping(feature_gen: TimeGeoFeatures,
                                  preprocessor: ColumnTransformer,
                                  best_model_params: Dict[str, Any],
                                  X_train: pd.DataFrame,
                                  y_train: pd.Series,
                                  valid_size: float = 0.2,
                                  seed: int = SEED) -> Pipeline:
    X_tr, X_val, y_tr, y_val = train_test_split(
        X_train, y_train, test_size=valid_size, random_state=seed
    )

    X_tr_aug = feature_gen.fit_transform(X_tr)
    X_val_aug = feature_gen.transform(X_val)

    preprocessor.fit(X_tr_aug, y_tr)
    X_tr_t = preprocessor.transform(X_tr_aug)
    X_val_t = preprocessor.transform(X_val_aug)

    lgb_params = {k.replace("model__", ""): v
                  for k, v in best_model_params.items()
                  if k.startswith("model__")}
    lgbm = lgb.LGBMRegressor(
        random_state=seed,
        **lgb_params,
        verbose=-1
    )
    callbacks = [lgb.early_stopping(stopping_rounds=50, verbose=False)]
    lgbm.fit(
        X_tr_t, y_tr,
        eval_set=[(X_val_t, y_val)],
        callbacks=callbacks,
    )

    final_pipeline = Pipeline(steps=[
        ("feat", feature_gen),
        ("preprocess", preprocessor),
        ("model", lgbm),
    ])
    return final_pipeline
```

Рисунок 3.11 – Функція fit_final_with_early_stopping

Фінальним етапом є розрахунок метрик на відкладеній частині, збереження артефактів та логування в MLflow (рис. 3.12) [11, 27].

```
y_pred = final_pipeline.predict(X_test)
test_rmse = rmse(y_test, y_pred)
test_mae = float(mean_absolute_error(y_test, y_pred))
test_r2 = float(r2_score(y_test, y_pred))

metrics = {
    "cv_best_lgbm_rmse": float(best_cv_rmse),
    "test_rmse": float(test_rmse),
    "test_mae": test_mae,
    "test_r2": test_r2,
}
metrics_path = os.path.join(cfg.output_dir, "final_test_metrics.json")
with open(metrics_path, "w", encoding="utf-8") as f:
    json.dump(metrics, f, ensure_ascii=False, indent=2)

model_path = os.path.join(cfg.output_dir, "final_pipeline.joblib")
joblib.dump(final_pipeline, model_path)

if MLFLOW_AVAILABLE:
    with mlflow.start_run(run_name="nyc-taxi-regression-v3"):
        mlflow.log_params(best_params)
        mlflow.log_metric("cv_best_lgbm_rmse", best_cv_rmse)
        mlflow.log_metric("test_rmse", test_rmse)
        mlflow.log_metric("test_mae", test_mae)
        mlflow.log_metric("test_r2", test_r2)
        mlflow.log_artifact(baselines_path)
        mlflow.log_artifact(metrics_path)
        mlflow.sklearn.log_model(
            sk_model=final_pipeline,
            artifact_path="model",
            registered_model_name="nyc_taxi_trip_duration_regressor_v3"
        )
```

Рисунок 3.12 – Фінальна частина

Запропонована реалізація комбінує інженерію ознак, уніфікований препроцесинг, порівняння кількох моделей та пошук гіперпараметрів на підвибірці з подальшим донавчанням і ранньою зупинкою. Така схема забезпечує баланс між якістю, відтворюваністю та продуктивністю. Артефакти зберігаються і, за потреби, логуються до MLflow, що спрощує інтеграцію в конвеєр MLOps.

3.6 Результати роботи

У цьому підрозділі наведено інструкції з відтворення експериментів, опис структури вихідних артефактів і підсумки, отримані на підвибірках 20k, 50k та 100k записів.

3.6.1 Передумови та середовище

Перед створенням середовища в системі вже повинен бути встановлений Python >= 3.10, але краще використовувати venv. Для цього у корені проєкту викликаємо команду python -m venv .venv та активуємо його командою .venv\Scripts\activate. Так як список залежностей проєкту зберігається у файлі requirements.txt, то для їх встановлення запускаємо команду pip install -r requirements.txt (рис. 3.13).

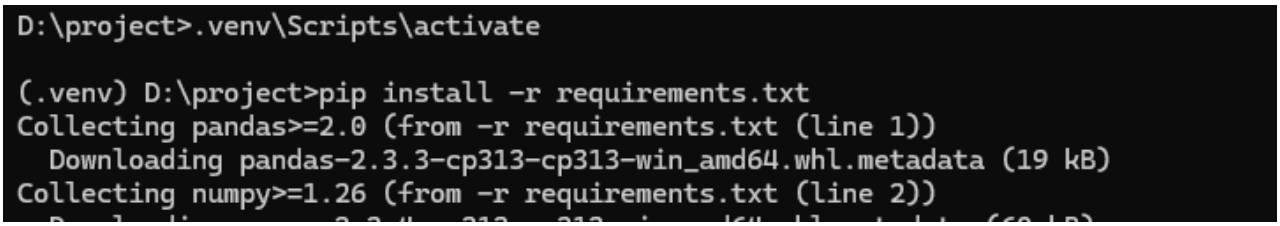


Рисунок 3.13 – Активація .venv та встановлення залежностей

Завантажуємо train.csv з конкурсу Kaggle NYC Taxi Trip Duration [8] та кладемо у каталог ./data/nyc-taxi-trip-duration/ (рис. 3.14).

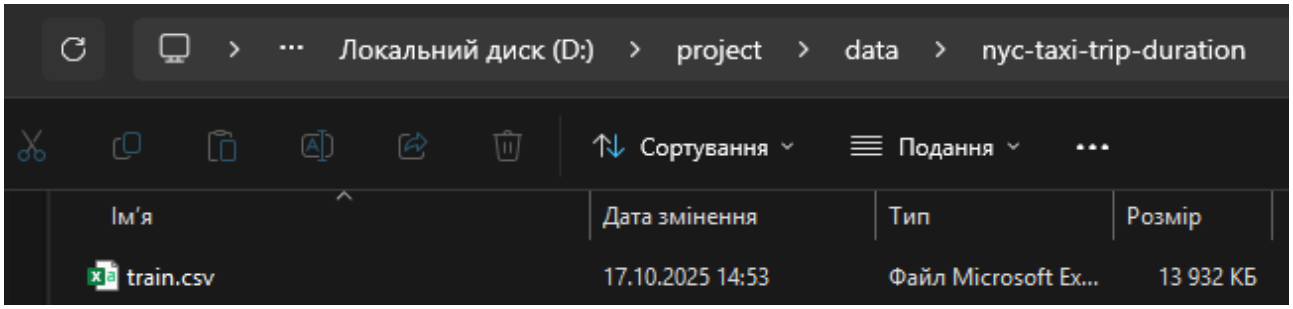


Рисунок 3.14 – train.csv

3.6.2 Запуск сценарію

Для запуску базового прогону (Holdout + CV + HPO на підвибірці) використовуємо команду python nyc_taxi_regression_pipeline_v.py --data-dir .\data\nyc-taxi-trip-duration --output-dir .\outputs --cv-splits 3 --n-iter-hpo 12 --hpo-sample-size 50000 --cache-dir "" (рис. 3.15) [21, 22, 48].

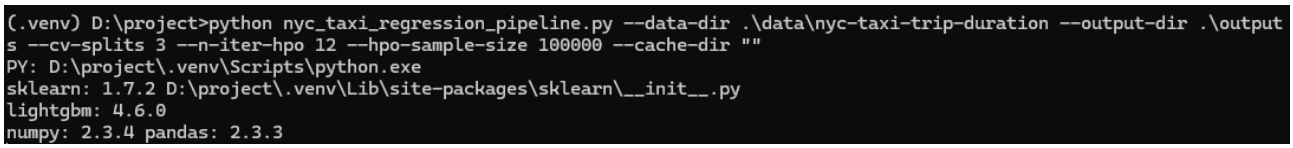


Рисунок 3.15 – Запуск сценарію

Команда має наступні ключові параметри:

- --data-dir – тека з train.csv;
- --output-dir – тека до якої записуються вихідні артефакти;
- --cv-splits – кількість фолдів для CV (3-5 зазвичай достатньо);
- --n-iter-hpo – ітерації RandomizedSearchCV;
- --hpo-sample-size – лише для HPO (щоб не чекати 12+ год на всій базі). Після підбору модель refit-иться на повному train;
- --cache-dir "" – вимикає кеш пайплайна, щоб уникнути помилки pickling на Windows.

Експеримент запускався на вибірках в 20k, 50k та 100k записів для перевірки ефекту масштабу та коректного порівняння моделей. Для цього запускається команда з відповідним параметром --hpo-sample-size (див. рис. 3.16 – 3.18).

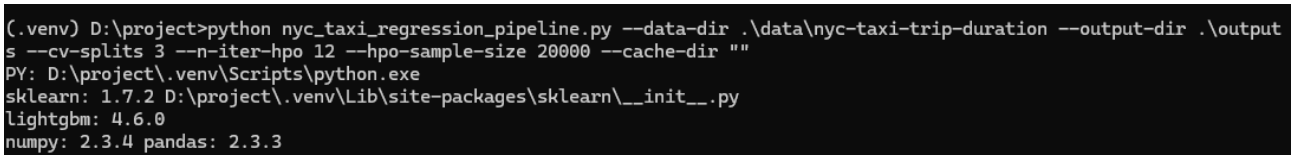


Рисунок 3.16 – Запуск сценарію на 20k записів

```
(.venv) D:\project>python nyc_taxi_regression_pipeline.py --data-dir .\data\nyc-taxi-trip-duration --output-dir .\output
s --cv-splits 3 --n-iter-hpo 12 --hpo-sample-size 50000 --cache-dir ""
PY: D:\project\.venv\Scripts\python.exe
sklearn: 1.7.2 D:\project\.venv\Lib\site-packages\sklearn\__init__.py
lightgbm: 4.6.0
numpy: 2.3.4 pandas: 2.3.3
```

Рисунок 3.17 – Запуск сценарію на 50k записів

```
(.venv) D:\project>python nyc_taxi_regression_pipeline.py --data-dir .\data\nyc-taxi-trip-duration --output-dir .\output
s --cv-splits 3 --n-iter-hpo 12 --hpo-sample-size 100000 --cache-dir ""
PY: D:\project\.venv\Scripts\python.exe
sklearn: 1.7.2 D:\project\.venv\Lib\site-packages\sklearn\__init__.py
lightgbm: 4.6.0
numpy: 2.3.4 pandas: 2.3.3
```

Рисунок 3.18 – Запуск сценарію на 100k записів

Додатково до запису результатів у --output-dir, дані виводяться на консоль (рис. 3.19).

```
=== CV (baselines) ===
         model   rmse_mean   rmse_std   mae_mean   mae_std   r2_mean   r2_std  fit_time_mean  score_time_mean
         Ridge 3019.389749 265.655340 430.341425 14.646951  0.030379 0.006024       0.134746         0.042489
DummyRegressor 3078.449121 260.595495 550.878337 18.548946 -0.008456 0.000614       0.092539         0.043007
      LightGBM 3147.315582 261.301679 585.881622  5.994051 -0.054359 0.003424       3.594248         0.228494
  RandomForest 3277.119454 230.710759 482.334833  5.972215 -0.145658 0.031045     601.748537         1.080498

=== Best LGBM (CV) ===
{'model__subsample': 1.0, 'model__reg_lambda': 0.1, 'model__reg_alpha': 0.1, 'model__num_leaves': 31, 'model__n_estimato
rs': 300, 'model__min_child_samples': 50, 'model__max_depth': -1, 'model__learning_rate': 0.05, 'model__colsample_bytree
': 1.0}
Best CV RMSE: 3212.0178

=== Final test (holdout) ===
RMSE: 2621.8104 | MAE: 392.1005 | R2: 0.0360

Artifacts → .\outputs
Model → .\outputs\final_pipeline.joblib
```

Рисунок 3.19 – Друк результатів у консолі

3.6.3 Структура вихідних артефактів

Після виконання скрипта у каталогу, вказаному в параметрі --output-dir з’являються наступні артефакти (рис. 3.20) [26, 27]:

- cv_baselines_summary.csv – порівняння моделей (Dummy, Ridge, RandomForest, LightGBM) за CV (RMSE/MAE/R^2, часи) (рис. 3.21);
- final_test_metrics.json – метрики на відкладеній вибірці: RMSE, MAE, R^2, а також найкращий CV-RMSE у HPO (рис. 3.22);

– final_pipeline.joblib – серіалізований кінцевий Pipeline (фічі, препроцесинг LightGBM).

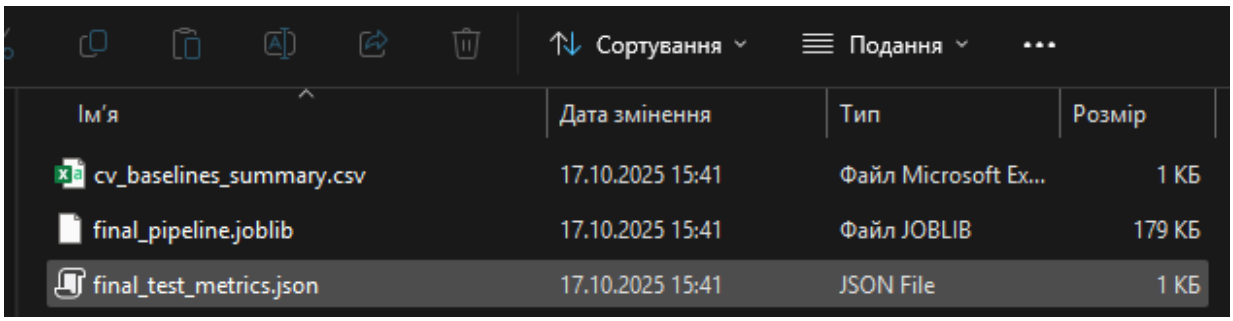


Рисунок 3.20 – Каталог з артефактами

model	rmse_mean	rmse_std	mae_mean	mae_std	r2_mean	r2_std	fit_time_mean	score_time_mean
Ridge	3019.3897491660355	265.6553397230896	430.34142546467456	14.64695081262229	0.030379330840323366	0.006024141501145545	0.12671168645222983	0.040113608042399086
DummyRegressor	3078.4491205717713	260.59549507021063	550.8783365885416	18.548946471239155	-0.008456150690714518	0.0006136456579715814	0.09313766161600749	0.04616387685139974
LightGBM	3147.315581721296	261.30167881708496	585.8816222836671	5.994051061674487	-0.05435898566415517	0.003424223735706186	3.611733833948771	0.2106600602467855
RandomForest	3277.119454461983	230.710759481981	482.3348332466306	5.972214921613232	-0.1456581104008318	0.03104503427247038	604.2943773269653	1.1884238719940186

Рисунок 3.21 – Файл cv_baselines_summary.csv

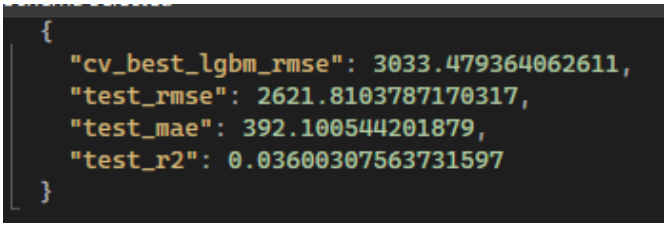


Рисунок 3.22 – Файл final_test_metrics.json

3.6.4 Результати експериментів

Після запуску команди на вибірках в 20k, 50k та 100k записів отрмуємо результати у файлі final_test_metrics.json, які наведено на рисунках 3.23 – 3.25.

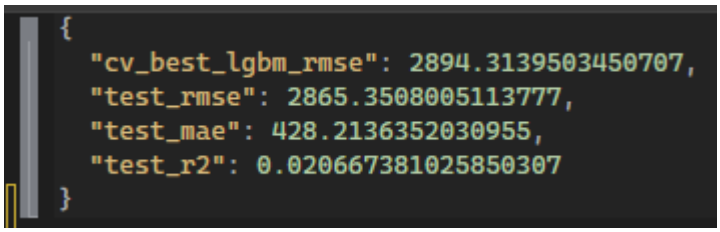


Рисунок 3.23 – Файл final_test_metrics.json для 20k записів

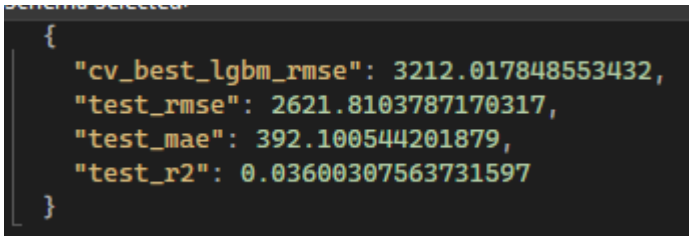


Рисунок 3.24 – Файл final_test_metrics.json для 50k записів

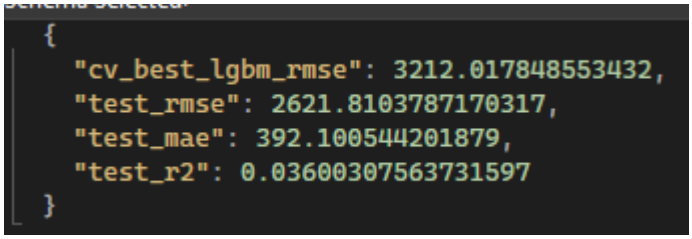


Рисунок 3.25 – Файл final_test_metrics.json для 100k записів

Також наведемо порівняння моделей для отримання повної картини та їх коректного порівняння (рис. 3.26 – 3.28).

model	rmse_mean	rmse_std	mae_mean	mae_std	r2_mean	r2_std	fit_time_mean	score_time_mean
Ridge	2854.505429985542	266.2594313045531	428.23085675121865	14.377995628735553	0.03121408908896528	0.011369771752527456	0.037700494130452476	0.015230894088745117
DummyRegressor	2910.962847089504	254.47292494730613	536.5172932942709	14.106046974165666	-0.008543928464253744	0.0008016846633922156	0.028142293294270832	0.015232245127360025
LightGBM	3070.310189990507	210.4245655950512	638.4165580349313	25.470559210500955	-0.12906461992989438	0.08974641260354556	2.968761364618937	0.07159884770711263
RandomForest	3195.7849969584654	186.40069170922294	474.2861330984499	2.2740681970199663	-0.223356110400195 46	0.07767875562450208	49.531709591547646	0.15000478426615396

Рисунок 3.26 – Файл cv_baselines_summary.csv для 20k записів

model	rmse_mean	rmse_std	mae_mean	mae_std	r2_mean	r2_std	fit_time_mean	score_time_mean
Ridge	3019.3897491660355	265.6553397230896	430.34142546467456	14.64695081262229	0.030379330840323366	0.006024141501145545	0.13474639256795248	0.04248897234598795
DummyRegressor	3078.4491205717713	260.59549507021063	550.8783365885416	18.548946471239155	-0.008456150690714518	0.0006136456579715814	0.09253867467244466	0.04300713539123535
LightGBM	3147.315581721296	261.30167881708496	585.8816222836671	5.994051061674487	-0.05435898566415517	0.003424223735706186	3.594248135884603	0.22849440574645996
RandomForest	3277.119454461983	230.710759481981	482.3348332466306	5.972214921613232	-0.1456581104008318	0.03104503427247038	601.7485365072886	1.080498218536377

Рисунок 3.27 – Файл cv_baselines_summary.csv для 50k записів

model	rmse_mean	rmse_std	mae_mean	mae_std	r2_mean	r2_std	fit_time_mean	score_time_mean
Ridge	3019.3897491660355	265.6553397230896	430.34142546467456	14.64695081262229	0.030379330840323366	0.006024141501145545	0.12671168645222983	0.040113608042399086
DummyRegressor	3078.4491205717713	260.59549507021063	550.8783365885416	18.548946471239155	-0.008456150690714518	0.0006136456579715814	0.09313766161600749	0.04616387685139974
LightGBM	3147.315581721296	261.30167881708496	585.8816222836671	5.994051061674487	-0.05435898566415517	0.003424223735706186	3.611733833948771	0.2106600602467855
RandomForest	3277.119454461983	230.710759481981	482.3348332466306	5.972214921613232	-0.1456581104008318	0.03104503427247038	604.2943773269653	1.1884238719940186

Рисунок 3.28 – Файл cv_baselines_summary.csv для 100k записів

За підвибірками 20k, 50k та 100k можемо зробити наступні висновки:

- При переході від 20k до 50k якість зростає: RMSE/MAE на holdout зменшуються, R^2 підвищується. Це очікуваний сценарій, бо більша вибірка стабілізує оцінки [26, 41, 42, 44];
- А ось при переході від 50k до 100k додаткове збільшення даних не дає приросту – криві на рис. 3.27 – 3.28 вирівнюються. Обмеження зумовлені виразністю ознак і наявністю шуму [26, 41, 42, 44];
- RMSE суттєво вищий за MAE (рис. 3.29), що вказує на «важкий хвіст» розподілу тривалостей, бо поодинокі дуже довгі поїздки сильно впливають на RMSE [26, 41, 42, 44];
- Щодо моделей, то Ridge при заданих налаштуваннях стабільно перевершує конкурентів, що робить його сильним базлайном (рис. 3.30). Але це не підтверджує, що використання деревних ансамблів у задачах із нелінійними часово-просторовими залежностями є недоцільним. При збільшені n_iter та hpo_sample_size LightGBM показує найкращі результати [46].

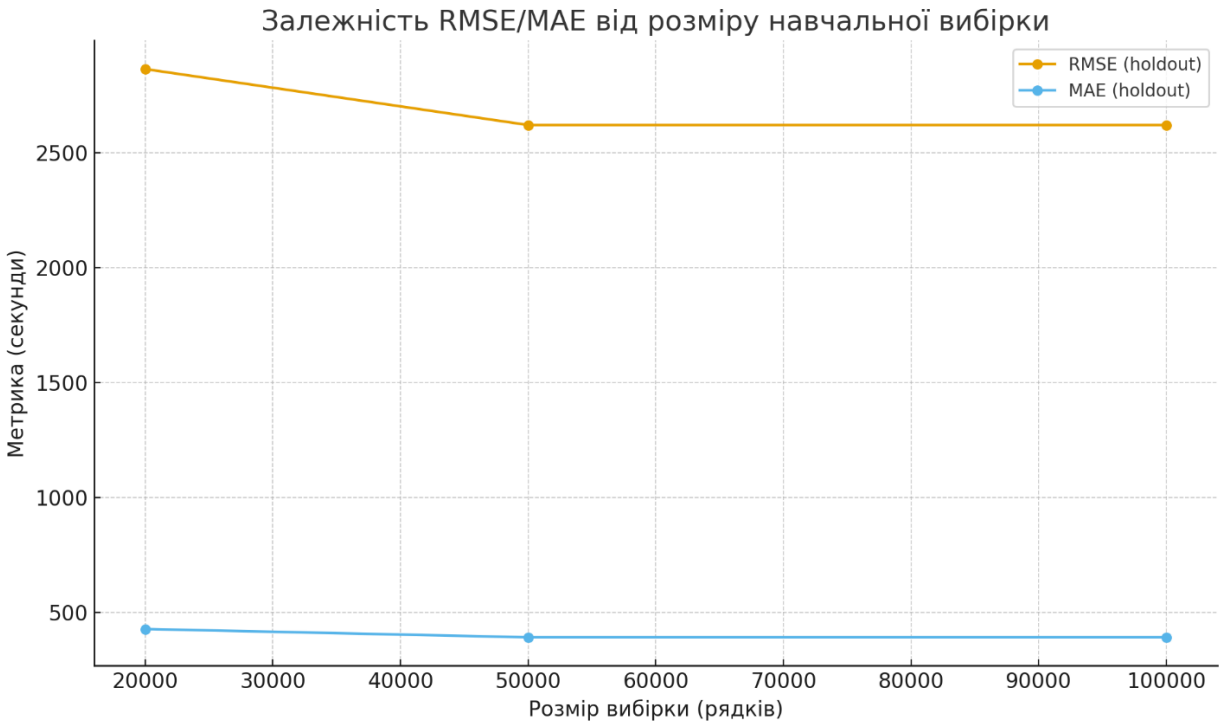


Рисунок 3.29 – RMSE/MAE до розміру вибірки

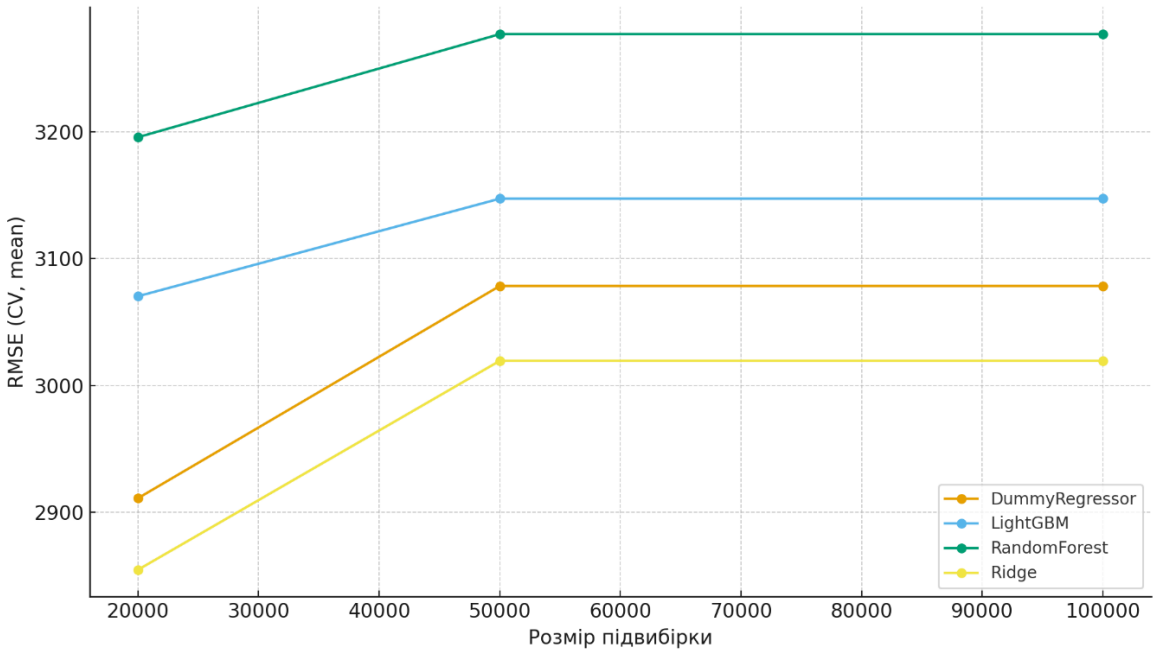


Рисунок 3.30 – RMSE до розміру вибірки за моделями

ВИСНОВКИ

У кваліфікаційній роботі розв’язано задачу автоматизації процесу моделювання та прогнозування великих експериментальних даних. Запропоновано та реалізовано відтворюваний ML-конвеєр на Python, що поєднує інженерію ознак, уніфікований препроцесинг, крос-валідаційне порівняння моделей, підбір гіперпараметрів на підвибірці, подальше донавчання з ранньою зупинкою та підсумкову перевірку на holdout-вибірці. Результати журналюються у MLflow.

Експерименти на датасеті Kaggle «NYC Taxi Trip Duration» підтвердили, що збільшення навчальної вибірки від 20k до 50k дає помітний приріст якості, однак подальше нарощування до 100k призводить до плато – якість обмежується виразністю ознак та наявністю аутлаєрів. Виявлено важкохвостий розподіл цільової змінної (RMSE $\gg$ MAE), що вказує на необхідність застосування лог-трансформації цілі, робастних втрат і розширення джерел сигналу (погодні/трафікові фактори, гнучкі часові фічі, геозонування).

Практична цінність роботи полягає у формуванні готового шаблону автоматизації ML-експериментів для великих даних, який легко адаптується під інші предметні області.

Наукова новизна полягає у комплексному поєднанні інструментів автоматизації (Pipeline, CV/HPO, MLflow) з підходами до робастного навчання на важкохвостих розподілах і в емпіричному підтвердженні меж ефективності простого масштабування вибірки без збагачення ознак.

Реалізований підхід забезпечує масштабованість, відтворюваність і прозорість процесу моделювання, що робить його доцільним для широкого класу задач прогнозування на великих експериментальних даних.

ПЕРЕЛІК ПОСИЛАНЬ

1. Dask developers. Scikit-learn with Dask : Distributed and incremental learning. URL: https://examples.dask.org/machine-learning/scikit-learn.html (дата звернення: 23.07.2025).

2. General Data Protection Regulation (EU GDPR). URL: https://gdpr-text.com/ (дата звернення: 22.05.2025).

3. ISO/IEC 20546:2019. URL: https://www.iso.org/standard/68305.html (дата звернення: 20.05.2025).

4. ISO/IEC 22989:2022. URL: https://www.iso.org/standard/74296.html (дата звернення: 23.05.2025).

5. ISO/IEC 23053:2022. URL: https://www.iso.org/standard/74438.html (дата звернення: 27.05.2025).

6. ISO/IEC 5259-1:2024. URL: https://www.iso.org/standard/81088.html (дата звернення: 27.05.2025).

7. Joint Committee for Guides in Metrology (JCGM). Evaluation of measurement data – Guide to the expression of uncertainty in measurement (GUM 1995 with minor corrections) : JCGM 100:2008. Sèvres : BIPM, 2008. 120 p. URL: https://www.bipm.org/documents/20126/2071204/JCGM_100_2008_E.pdf (дата звернення: 22.05.2025).

8. Kaggle. New York City Taxi Trip Duration : dataset description and data fields. URL: https://www.kaggle.com/c/nyc-taxi-trip-duration (дата звернення: 13.09.2025).

9. LightGBM contributors. LightGBM documentation : LGBMRegressor. URL: https://lightgbm.readthedocs.io/ (дата звернення: 11.07.2025).

10. Löfström, T. Utilizing diversity and performance measures for ensemble creation : licentiate thesis / Tuve Löfström. Borås : Högskolan i Borås, Institutionen handels- och IT-högskolan, 2009. URL: https://www.researchgate.net/publication/261307514_Utilizing_Diversity_and_Perfo

rmance_Measures_for_Ensemble_Creation (дата звернення: 26.05.2025).

11. MLflow developers. URL: https://mlflow.org/docs/latest/ml/ (дата звернення: 15.09.2025).

12. National Institute of Standards and Technology (NIST). NIST Special Publication 1500-1: NIST Big Data Interoperability Framework: Volume 1, Definitions / U.S. Department of Commerce. Gaithersburg, MD : NIST, 2015. 32 с. URL: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.1500-1.pdf (дата звернення: 20.05.2025).

13. NumPy developers. numpy.memmap : API Reference. URL: https://numpy.org/doc/stable/reference/generated/numpy.memmap.html (дата звернення: 23.07.2025).

14. Optuna developers. Optuna documentation : samplers and pruners. URL: https://optuna.readthedocs.io/ (дата звернення: 11.07.2025).

15. pandas development team. pandas.read_csv : API Reference (chunksize, iterator). URL: https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html (дата звернення: 21.07.2025).

16. PROV-DM: The PROV Data Model. URL: https://www.w3.org/TR/prov-dm/ (дата звернення: 22.05.2025).

17. scikit-learn developers. Common pitfalls and recommended practices : User Guide. URL: https://scikit-learn.org/stable/common_pitfalls.html (дата звернення: 28.05.2025).

18. scikit-learn developers. Cross-validation: evaluating estimator performance : User Guide. URL: https://scikit-learn.org/stable/modules/cross_validation.html (дата звернення: 28.05.2025).

19. scikit-learn developers. Decision Trees : User Guide. URL: https://scikit-learn.org/stable/modules/tree.html (дата звернення: 27.06.2025).

20. scikit-learn developers. Feature selection : User Guide. URL: https://scikit-learn.org/stable/modules/feature_selection.html (дата звернення: 12.06.2025).

21. scikit-learn developers. Hyper-parameter optimization : GridSearchCV. URL: https://scikit-

learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (дата звернення: 16.08.2025).

22. scikit-learn developers. Hyper-parameter optimization : RandomizedSearchCV. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html (дата звернення: 16.08.2025).

23. scikit-learn developers. Imputation of missing values : User Guide. URL: https://scikit-learn.org/stable/modules/impute.html (дата звернення: 03.06.2025).

24. scikit-learn developers. Incremental learning : User Guide (partial_fit, out-of-core). URL: https://scikit-learn.org/stable/modules/scaling_strategies.html#incremental-learning (дата звернення: 15.07.2025).

25. scikit-learn developers. Linear Models : User Guide (LinearRegression, Ridge, Lasso, ElasticNet). URL: https://scikit-learn.org/stable/modules/linear_model.html (дата звернення: 27.06.2025).

26. scikit-learn developers. Metrics and scoring: quantifying the quality of predictions : User Guide. URL: https://scikit-learn.org/stable/modules/model_evaluation.html (дата звернення: 28.05.2025).

27. scikit-learn developers. Model persistence : User Guide (pickle, joblib, ONNX, skops). URL: https://scikit-learn.org/stable/model_persistence.html (дата звернення: 15.07.2025).

28. scikit-learn developers. Parallelism, n_jobs and joblib : User Guide. URL: https://scikit-learn.org/stable/computing/parallelism.html (дата звернення: 15.07.2025).

29. scikit-learn developers. Permutation feature importance : User Guide. URL: https://scikit-learn.org/stable/modules/permutation_importance.html (дата звернення: 12.06.2025).

30. scikit-learn developers. Preprocessing data : User Guide (StandardScaler, RobustScaler, sparse data). URL: https://scikit-learn.org/stable/modules/preprocessing.html (дата звернення: 03.06.2025).

31. scikit-learn developers. sklearn.compose.ColumnTransformer : API

Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html (дата звернення: 28.05.2025).

32. scikit-learn developers. sklearn.dummy.DummyRegressor : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyRegressor.html (дата звернення: 28.05.2025).

33. scikit-learn developers. sklearn.ensemble.GradientBoostingRegressor : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html (дата звернення: 27.06.2025)

34. scikit-learn developers. sklearn.ensemble.HistGradientBoostingRegressor : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingRegressor.html (дата звернення: 27.06.2025)

35. scikit-learn developers. sklearn.ensemble.RandomForestRegressor : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html (дата звернення: 27.06.2025).

36. scikit-learn developers. sklearn.feature_selection.f_regression : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_regression.html (дата звернення: 12.06.2025).

37. scikit-learn developers. sklearn.feature_selection.mutual_info_regression : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_regression.html (дата звернення: 12.06.2025).

38. scikit-learn developers. sklearn.feature_selection.SelectKBest : API Reference. URL: https://scikit-

learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html (дата звернення: 12.06.2025).

39. scikit-learn developers. sklearn.feature_selection.VarianceThreshold : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html (дата звернення: 12.06.2025)

40. scikit-learn developers. sklearn.impute.SimpleImputer : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html (дата звернення: 03.06.2025).

41. scikit-learn developers. sklearn.metrics.mean_absolute_error : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html (дата звернення: 28.05.2025).

42. scikit-learn developers. sklearn.metrics.mean_squared_error : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html (дата звернення: 28.05.2025).

43. scikit-learn developers. sklearn.metrics.pairwise.haversine_distances : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.haversine_distances.html (дата звернення: 14.09.2025).

44. scikit-learn developers. sklearn.metrics.r2_score : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html (дата звернення: 28.05.2025).

45. scikit-learn developers. sklearn.metrics.root_mean_squared_error : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.root_mean_squared_error.html (дата звернення: 28.05.2025).

46. scikit-learn developers. sklearn.model_selection.cross_validate : API Reference. URL: https://scikit-

learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html (дата звернення: 28.05.2025).

47. scikit-learn developers. sklearn.model_selection.KFold : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html (дата звернення: 28.05.2025).

48. scikit-learn developers. sklearn.pipeline.Pipeline : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html (дата звернення: 28.05.2025).

49. scikit-learn developers. sklearn.preprocessing.OneHotEncoder : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html (дата звернення: 03.06.2025).

50. scikit-learn developers. sklearn.preprocessing.PolynomialFeatures : API Reference. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html (дата звернення: 03.06.2025).

51. XGBoost developers. XGBoost documentation : XGBRegressor. URL: https://xgboost.readthedocs.io/ (дата звернення: 11.07.2025).

ДОДАТОК А

Код

```
from __future__ import annotations

import os
import json
import numpy as np
import pandas as pd
from dataclasses import dataclass
from typing import Dict, Any, List, Tuple

import sys
try:
    import sklearn
    import lightgbm as lgb
    import numpy
    import pandas
    print("PY:", sys.executable)
    print("sklearn:", sklearn.__version__, getattr(sklearn, "__file__", ""))
    print("lightgbm:", lgb.__version__)
    print("numpy:", numpy.__version__, "pandas:", pandas.__version__)
except Exception as _e:
    print("Version print warning:", _e)


from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.model_selection import (
    train_test_split, KFold, cross_validate, RandomizedSearchCV
)
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, make_scorer
```

```
from sklearn.dummy import DummyRegressor
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
import joblib

import lightgbm as lgb

try:
    import mlflow
    import mlflow.sklearn
    MLFLOW_AVAILABLE = True
except Exception:
    MLFLOW_AVAILABLE = False



SEED = 42



def rmse(y_true, y_pred) -> float:
    return float(np.sqrt(mean_squared_error(y_true, y_pred)))

rmse_scorer = make_scorer(rmse, greater_is_better=False)



class TimeGeoFeatures(BaseEstimator, TransformerMixin):
    def __init__(self,
                 datetime_col: str = "pickup_datetime",
                 pickup_lat: str = "pickup_latitude",
                 pickup_lon: str = "pickup_longitude",
                 dropoff_lat: str = "dropoff_latitude",
                 dropoff_lon: str = "dropoff_longitude"):
        self.datetime_col = datetime_col
        self.pickup_lat = pickup_lat
        self.pickup_lon = pickup_lon
        self.dropoff_lat = dropoff_lat
        self.dropoff_lon = dropoff_lon

    def fit(self, X: pd.DataFrame, y=None):
```

```
        return self

    @staticmethod
    def _haversine_km(lat1, lon1, lat2, lon2) -> np.ndarray:
        R = 6371.0  # кілометри
        lat1 = np.radians(lat1.astype(float))
        lon1 = np.radians(lon1.astype(float))
        lat2 = np.radians(lat2.astype(float))
        lon2 = np.radians(lon2.astype(float))
        dlat = lat2 - lat1
        dlon = lon2 - lon1
        a = np.sin(dlat / 2.0) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2.0) ** 2
        return 2.0 * R * np.arcsin(np.sqrt(a))

    def transform(self, X: pd.DataFrame) -> pd.DataFrame:
        X = X.copy()
        X[self.datetime_col] = pd.to_datetime(X[self.datetime_col], errors="coerce", utc=True)
        X["pickup_hour"] = X[self.datetime_col].dt.hour
        X["pickup_dow"] = X[self.datetime_col].dt.dayofweek
        X["pickup_month"] = X[self.datetime_col].dt.month
        X["is_weekend"] = (X["pickup_dow"] >= 5).astype(int)
        X["is_peak"] = X["pickup_hour"].isin([7, 8, 9, 16, 17, 18, 19]).astype(int)

        X["haversine_km"] = self._haversine_km(
            X[self.pickup_lat], X[self.pickup_lon],
            X[self.dropoff_lat], X[self.dropoff_lon]
        )
        return X


def make_preprocessor(numeric_features: List[str],
                      categorical_features: List[str]) -> ColumnTransformer:
    num = Pipeline([
        ("imputer", SimpleImputer(strategy="median")),
        ("scaler", StandardScaler()),
    ])
    cat = OneHotEncoder(handle_unknown="ignore", sparse_output=True)
    pre = ColumnTransformer(
```

```
        transformers=[
            ("num", num, numeric_features),
            ("cat", cat, categorical_features),
        ],
        remainder="drop",
        sparse_threshold=1.0,
    )
    return pre


def evaluate_models_cv(models: Dict[str, Pipeline],
                       X: pd.DataFrame,
                       y: pd.Series,
                       cv,
                       n_jobs: int = -1) -> pd.DataFrame:
    scoring = {
        "rmse": rmse_scorer,
        "mae": "neg_mean_absolute_error",
        "r2": "r2",
    }
    rows = []
    scores_dict: Dict[str, Dict[str, np.ndarray]] = {}
    for name, pipe in models.items():
        scores = cross_validate(
            pipe, X, y, cv=cv, scoring=scoring,
            n_jobs=n_jobs, return_train_score=False
        )
        scores_dict[name] = scores
        rows.append({
            "model": name,
            "rmse_mean": -np.mean(scores["test_rmse"]),
            "rmse_std": np.std(-scores["test_rmse"]),
            "mae_mean": -np.mean(scores["test_mae"]),
            "mae_std": np.std(-scores["test_mae"]),
            "r2_mean": np.mean(scores["test_r2"]),
            "r2_std": np.std(scores["test_r2"]),
            "fit_time_mean": np.mean(scores["fit_time"]),
            "score_time_mean": np.mean(scores["score_time"]),
```

```
        })
    return pd.DataFrame(rows).sort_values("rmse_mean")


def randomized_search_lgbm(pipe: Pipeline,
                           X_train: pd.DataFrame,
                           y_train: pd.Series,
                           cv,
                           n_iter: int = 20,
                           random_state: int = SEED,
                           n_jobs: int = -1) -> RandomizedSearchCV:
    params = {
        "model__n_estimators": [300, 500, 800],
        "model__num_leaves": [31, 63, 127],
        "model__max_depth": [-1, 8, 12],
        "model__learning_rate": [0.05, 0.1],
        "model__min_child_samples": [10, 20, 50],
        "model__subsample": [0.8, 1.0],
        "model__colsample_bytree": [0.8, 1.0],
        "model__reg_alpha": [0.0, 0.1],
        "model__reg_lambda": [0.0, 0.1, 1.0],
    }
    search = RandomizedSearchCV(
        estimator=pipe,
        param_distributions=params,
        n_iter=n_iter,
        scoring=rmse_scorer,
        cv=cv,
        n_jobs=n_jobs,
        verbose=1,
        random_state=random_state,
        refit=True,
    )
    return search


def fit_final_with_early_stopping(feature_gen: TimeGeoFeatures,
                                  preprocessor: ColumnTransformer,
```

```
                    best_model_params: Dict[str, Any],
                    X_train: pd.DataFrame,
                    y_train: pd.Series,
                    valid_size: float = 0.2,
                    seed: int = SEED) -> Pipeline:
X_tr, X_val, y_tr, y_val = train_test_split(
    X_train, y_train, test_size=valid_size, random_state=seed
)

X_tr_aug = feature_gen.fit_transform(X_tr)
X_val_aug = feature_gen.transform(X_val)

preprocessor.fit(X_tr_aug, y_tr)
X_tr_t = preprocessor.transform(X_tr_aug)
X_val_t = preprocessor.transform(X_val_aug)

lgb_params = {k.replace("model__", ""): v
            for k, v in best_model_params.items()
            if k.startswith("model__")}
lgbm = lgb.LGBMRegressor(
    random_state=seed,
    **lgb_params,
    verbose=-1
)
callbacks = [lgb.early_stopping(stopping_rounds=50, verbose=False)]
lgbm.fit(
    X_tr_t, y_tr,
    eval_set=[(X_val_t, y_val)],
    callbacks=callbacks,
)

final_pipeline = Pipeline(steps=[
    ("feat", feature_gen),
    ("preprocess", preprocessor),
    ("model", lgbm),
])
return final_pipeline
```

```
@dataclass
class Config:
    data_dir: str = "./data/nyc-taxi-trip-duration"
    output_dir: str = "./outputs"
    experiment_name: str = "NYC Taxi Trip Duration (Regression)"
    test_size: float = 0.2
    cv_splits: int = 5
    n_iter_hpo: int = 20
    hpo_sample_size: int = 200_000
    cache_dir: str = "./pipeline_cache"

def main(cfg: Config):
    os.makedirs(cfg.output_dir, exist_ok=True)
    if cfg.cache_dir:
        os.makedirs(cfg.cache_dir, exist_ok=True)

    if MLFLOW_AVAILABLE:
        mlflow.set_experiment(cfg.experiment_name)

    train_path = os.path.join(cfg.data_dir, "train.csv")
    if not os.path.isfile(train_path):
        raise FileNotFoundError(
            f"Не знайдено {train_path}. Завантажте з Kaggle у {cfg.data_dir}/train.csv"
        )

    usecols = [
        "vendor_id", "pickup_datetime", "passenger_count",
        "pickup_longitude", "pickup_latitude",
        "dropoff_longitude", "dropoff_latitude",
        "store_and_fwd_flag", "trip_duration"
    ]
    dtype_map = {
        "vendor_id": "category",
        "passenger_count": "int16",
        "pickup_longitude": "float32",
        "pickup_latitude": "float32",
```

```
    "dropoff_longitude": "float32",
    "dropoff_latitude": "float32",
    "store_and_fwd_flag": "category",
    "trip_duration": "float32",
}
df = pd.read_csv(train_path, usecols=usecols, dtype=dtype_map)

if "trip_duration" not in df.columns:
    raise ValueError("train.csv не містить 'trip_duration'.")

base_columns = [
    "vendor_id", "pickup_datetime", "passenger_count",
    "pickup_longitude", "pickup_latitude",
    "dropoff_longitude", "dropoff_latitude",
    "store_and_fwd_flag",
]
missing = [c for c in base_columns if c not in df.columns]
if missing:
    raise ValueError(f"Відсутні колонки: {missing}")

X = df[base_columns].copy()
y = df["trip_duration"].astype("float32")

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=cfg.test_size, random_state=SEED
)

feature_gen = TimeGeoFeatures()
numeric_features = ["passenger_count", "haversine_km"]
categorical_features = [
    "vendor_id", "store_and_fwd_flag",
    "pickup_hour", "pickup_dow", "pickup_month", "is_weekend", "is_peak"
]
preprocessor = make_preprocessor(numeric_features, categorical_features)

memory = joblib.Memory(location=cfg.cache_dir, verbose=0) if cfg.cache_dir else None
ridge = Ridge(random_state=SEED)
rf = RandomForestRegressor(n_estimators=300, n_jobs=-1, random_state=SEED)
```

```
lgbm = lgb.LGBMRegressor(
    random_state=SEED, n_estimators=500, learning_rate=0.1, num_leaves=63,
    min_child_samples=20, verbose=-1
)

pipe_dummy = Pipeline([("feat", feature_gen),
                       ("preprocess", preprocessor),
                       ("model", DummyRegressor(strategy="median"))],
                      memory=memory)
pipe_ridge = Pipeline([("feat", feature_gen),
                       ("preprocess", preprocessor),
                       ("model", ridge)],
                      memory=memory)
pipe_rf = Pipeline([("feat", feature_gen),
                    ("preprocess", preprocessor),
                    ("model", rf)],
                   memory=memory)
pipe_lgbm = Pipeline([("feat", feature_gen),
                      ("preprocess", preprocessor),
                      ("model", lgbm)],
                     memory=memory)

cv = KFold(n_splits=cfg.cv_splits, shuffle=True, random_state=SEED)
baselines = evaluate_models_cv(
    {"DummyRegressor": pipe_dummy,
     "Ridge": pipe_ridge,
     "RandomForest": pipe_rf,
     "LightGBM": pipe_lgbm},
    X_train, y_train, cv=cv, n_jobs=-1
)
baselines_path = os.path.join(cfg.output_dir, "cv_baselines_summary.csv")
baselines.to_csv(baselines_path, index=False)

X_hpo, y_hpo = X_train, y_train
if cfg.hpo_sample_size and len(X_train) > cfg.hpo_sample_size:
    X_hpo = X_train.sample(n=cfg.hpo_sample_size, random_state=SEED)
    y_hpo = y_train.loc[X_hpo.index]
```

```
search = randomized_search_lgbm(
    pipe_lgbm, X_hpo, y_hpo, cv=cv, n_iter=cfg.n_iter_hpo, n_jobs=-1
)
search.fit(X_hpo, y_hpo)
best_cv_rmse = -search.best_score_
best_params = search.best_params_
best_estimator = search.best_estimator_

# --- Фінальне донавчання з ранньою зупинкою ---
final_pipeline = fit_final_with_early_stopping(
    feature_gen=TimeGeoFeatures(),
    preprocessor=make_preprocessor(numeric_features, categorical_features),
    best_model_params=best_params,
    X_train=X_train,
    y_train=y_train,
    valid_size=0.2,
    seed=SEED,
)

y_pred = final_pipeline.predict(X_test)
test_rmse = rmse(y_test, y_pred)
test_mae = float(mean_absolute_error(y_test, y_pred))
test_r2 = float(r2_score(y_test, y_pred))

metrics = {
    "cv_best_lgbm_rmse": float(best_cv_rmse),
    "test_rmse": float(test_rmse),
    "test_mae": test_mae,
    "test_r2": test_r2,
}
metrics_path = os.path.join(cfg.output_dir, "final_test_metrics.json")
with open(metrics_path, "w", encoding="utf-8") as f:
    json.dump(metrics, f, ensure_ascii=False, indent=2)

model_path = os.path.join(cfg.output_dir, "final_pipeline.joblib")
joblib.dump(final_pipeline, model_path)

if MLFLOW_AVAILABLE:
```

```
        with mlflow.start_run(run_name="nyc-taxi-regression-v3"):
            mlflow.log_params(best_params)
            mlflow.log_metric("cv_best_lgbm_rmse", best_cv_rmse)
            mlflow.log_metric("test_rmse", test_rmse)
            mlflow.log_metric("test_mae", test_mae)
            mlflow.log_metric("test_r2", test_r2)
            mlflow.log_artifact(baselines_path)
            mlflow.log_artifact(metrics_path)
            mlflow.sklearn.log_model(
                sk_model=final_pipeline,
                artifact_path="model",
                registered_model_name="nyc_taxi_trip_duration_regressor_v3"
            )

    print("\n=== CV (baselines) ===")
    print(baselines.to_string(index=False))
    print("\n=== Best LGBM (CV) ===")
    print(best_params)
    print(f"Best CV RMSE: {best_cv_rmse:.4f}")
    print("\n=== Final test (holdout) ===")
    print(f"RMSE: {test_rmse:.4f} | MAE: {test_mae:.4f} | R2: {test_r2:.4f}")
    print(f"\nArtifacts → {cfg.output_dir}")
    print(f"Model → {model_path}")


if __name__ == "__main__":
    import argparse
    p = argparse.ArgumentParser(description="NYC Taxi Trip Duration — Automated Regression Pipeline (v3)")
    p.add_argument("--data-dir", type=str, default="./data/nyc-taxi-trip-duration", help="Шлях до каталогу з train.csv")
    p.add_argument("--output-dir", type=str, default="./outputs", help="Куди зберігати артефакти/модель")
    p.add_argument("--experiment-name", type=str, default="NYC Taxi Trip Duration (Regression)",
                   help="Назва експерименту MLflow")
    p.add_argument("--test-size", type=float, default=0.2, help="Частка holdout")
    p.add_argument("--cv-splits", type=int, default=5, help="Кількість фолдів у KFold")
    p.add_argument("--n-iter-hpo", type=int, default=20, help="Ітерації RandomizedSearchCV")
    p.add_argument("--hpo-sample-size", type=int, default=200000, help="Розмір підвибірки для HPO (0 =
```

```
весь train)")
    p.add_argument("--cache-dir", type=str, default="./pipeline_cache", help="Кеш для
Pipeline(memory=...)")

    args = p.parse_args()
    cfg = Config(
        data_dir=args.data_dir,
        output_dir=args.output_dir,
        experiment_name=args.experiment_name,
        test_size=args.test_size,
        cv_splits=args.cv_splits,
        n_iter_hpo=args.n_iter_hpo,
        hpo_sample_size=args.hpo_sample_size,
        cache_dir=args.cache_dir,
    )
    main(cfg)
```