НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

“До захисту допущено”
Завідувач кафедри
_____ Наталія АУШЕВА
“___”_____2025 р.

Дипломна робота
на здобуття ступеня бакалавр

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Середовище вибору оптимальної моделі машинного навчання для визначення стратегії бізнесу”

Виконала: студентка 4 курсу, групи ТР-14
Славицька Олександра Миколаївна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент каф. ЦТЕ, к.т.н., доцент, Світлана ШАПОВАЛОВА _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент професор каф. ІПЗЕ, д.т.н., доцент, Андрій МУСІЄНКО _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль асистент каф. ЦТЕ, Володимир РУДИК _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

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

Київ — 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти — перший (бакалаврський)
Спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Наталія АУШЕВА
(підпис)
“____” _____ 2025р.

З А В Д А Н Н Я
на дипломну роботу студентці

Славицькій Олександрі Миколаївні
(прізвище, ім’я, по батькові)

1. Тема роботи “Середовище вибору оптимальної моделі машинного навчання для визначення стратегії бізнесу”

Керівник роботи Шаповалова Світлана Ігорівна, к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “__” _____ 2025 року №

2. Термін подання студенткою роботи 09.06.2025 р.

3. Вихідні дані до роботи мова програмування Python, середовище розробки PyCharm.

4. Перелік питань, які потрібно розробити: 1) провести аналіз методів розв’язання задачі регресії; 2) реалізувати обрані моделі машинного навчання; 3) провести обчислювальні експерименти та проаналізувати їх результати; 4)розробити інтерфейс середовища вибору моделі машинного навчання.

5. Орієнтований перелік ілюстративного матеріалу крива крос-валідації для вибору ступеня полінома; візуалізація кластерної структури даних; діаграма залишків регресійної моделі; порівняльний графік метрик якості моделей; скріншоти інтерфейсу користувача; таблиця з порівнянням моделей за точністю; таблиця типових форм поліноміальних залежностей; таблиця важливості ознак у моделі; блок-схема роботи алгоритму кластеризації; блок-схема процесу навчання ансамблевих моделей; блок-схема побудови поліноміальної регресії.

6. Дата видачі завдання 13.09.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	13.09.2024	Виконано
2.	Аналіз методів та засобів розв’язання задачі	31.12.2024	Виконано
3.	Розробка архітектури та загальної структури системи	14.04-18.05.2025	Виконано
4.	Розробка окремих підсистем	14.04-18.05.2025	Виконано
5.	Програмна реалізація системи	14.04-18.05.2025	Виконано
6.	Оформлення пояснювальної записки	19.05-15.06.2025	Виконано
7.	Захист програмного забезпечення	12.05-18.05.2025	Виконано
8.	Передзахист	19.05-23.05.2025	Виконано
9.	Захист	16.05-20.05.2025	Виконано

Студент _____ Олександра СЛАВИЦЬКА
(підпис) (ім´я, ПРІЗВИЩЕ)

Керівник роботи _____ Світлана ШАПОВАЛОВА
(підпис) (ім´я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 67 сторінках, містить 27 ілюстрацій, 8 таблиць, 1 додаток, 12 джерел в переліку посилань.

Мета роботи — створення середовища вибору оптимальної за точністю моделі машинного навчання.

Методи та засоби: алгоритм кластеризації K-means, ансамблеві методи (Random Forest), поліноміальна регресія, стекінг (Stacked Generalization), мова програмування Python, бібліотеки Scikit-learn, Pandas, Matplotlib/Seaborn для візуалізації.

У результаті проведеного дослідження та реалізації програмної системи були досягнуті наступні основні результати:

— на основі аналізу методів розв’язання задачі регресії визначено для реалізації такі методи: лінійна регресія, дерево рішень, рандом форест;

— реалізовано обрані моделі машинного навчання на основі бібліотеки Scikit-learn і ресуру Kaggle;

— запропоновано вдосконалення методу рандом форест за рахунок передбачень попередньо навчених моделей регресії на прикладах окремих кластерів датасету;

— експериментально доведено підвищення точності запропонованого методу Stacking Regression;

— зозроблено середовище вибору моделі машинного навчання.

Рекомендації щодо застосування: модель ефективна для задач з явною кластерною структурою даних (економічне прогнозування, аналіз ринків нерухомості, медична діагностика).

Ключові слова: МАШИННЕ НАВЧАННЯ, АНСАМБЛЕВІ МЕТОДИ, СТЕКІНГ, КЛАСТЕРИЗАЦІЯ, ПРОГНОЗУВАННЯ, РЕГРЕСІЙНИЙ АНАЛІЗ, K-MEANS, METRIC LEARNING.

ABSTRACT

The bachelor’s thesis consists of 67 pages, includes 27 figures, 8 tables, 1 appendix, and 12 references.

Objective of the study: to develop an environment for selecting the most accurate machine learning model.

Methods and tools: K-means clustering algorithm, ensemble methods (Random Forest), polynomial regression, stacking (Stacked Generalization), Python programming language, Scikit-learn library, and Pandas, Matplotlib/Seaborn for data visualization.

As a result of the conducted research and the implementation of the software system, the following key outcomes were achieved:

— based on the analysis of regression problem-solving methods, the following models were selected for implementation: linear regression, decision tree, and random forest;

— the selected machine learning models were implemented using the Scikit-learn library and data from the Kaggle resource;

— an improvement of the random forest method was proposed by incorporating predictions of pre-trained regression models for individual dataset clusters;

— the improved method of Stacking Regression was experimentally proven to increase prediction accuracy;

— a model selection environment for machine learning was developed.

Recommendations for use: the model is effective for tasks with a clear cluster structure in the data (economic forecasting, real estate market analysis, medical diagnostics).

Keywords: MACHINE LEARNING, ENSEMBLE METHODS, STACKING, CLUSTERING, FORECASTING, REGRESSION ANALYSIS, K-MEANS, METRIC LEARNING.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ 8

ВСТУП 9

1 ЗАДАЧА ПОБУДОВИ СИСТЕМИ АНАЛІЗУ ТА ПРОГНОЗУВАННЯ ПРОДАЖІВ 11

1.1 Постановка задачі 11

1.2 Підзадачі та функціональні вимоги 12

1.3 Програмні засоби реалізації 12

2 МЕТОДИ РОЗВ'ЯЗАННЯ ЗАДАЧІ 14

2.1 Математичні основи регресійного аналізу 14

2.2 Нелінійні методи апроксимації 17

2.3 Деревоподібні структури в прогнозуванні 21

2.4 Метод Stacking 24

2.5 Кластерний підхід до сегментації даних 27

2.6 Метричний апарат оцінки якості 31

2.7 Сучасні підходи до інтерпретації моделей 34

2.8 Обробка нестандартних даних 36

2.9 Динамічна адаптація моделей 37

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРЕДОВИЩА 39

3.1 Інструментарій розробки 39

3.2 Архітектурні рішення 39

3.3 Реалізовані алгоритми аналізу 40

3.4 Реалізація інтерфейсу користувача 40

4 ІНТЕРФЕЙС КОРИСТУВАЧА ТА СЦЕНАРІЇ РОБОТИ 41

4.1 Вимоги до середовища запуску 41

4.2 Інсталяція та запуск системи 41

4.3 Сценарії взаємодії користувача із системою 42

4.5 Обчислювальні експерименти 49

ВИСНОВКИ 51

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 52

ДОДАТОК А 54

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

ML (Machine Learning) —машинне навчання

AI (Artificial Intelligence) —штучний інтелект

RF (Random Forest) —випадковий ліс

SVM (Support Vector Machine) —метод опорних векторів

MSE (Mean Squared Error) —середньоквадратична помилка

RMSE (Root Mean Squared Error) —корінь із середньоквадратичної помилки

MAE (Mean Absolute Error) —середня абсолютна помилка

R^2 (R-squared) —коефіцієнт детермінації

CV (Cross-Validation) —перехресна перевірка

Стекінг (Stacking) — ансамблевий метод, що комбінує прогнози кількох базових моделей через мета-модель

Кластеризація — розподіл даних на групи за схожістю

Регресія — задача передбачення неперервної величини

Ансамблеві методи — комбінування прогнозів кількох моделей

Гіперпараметри — параметри моделі, що налаштовуються до навчання

Перетренування — ситуація, коли модель запам’ятовує дані замість їхнього загального шаблону

Мета-ознаки — прогнози базових моделей, що використовуються як вхід для метамоделі

Кросс-валідація — метод оцінки якості моделі шляхом розбиття даних на частини

Датасет — набір даних для аналізу

Усі терміни наведено у формі, що відповідає їхньому вживанню в контексті роботи. Скорочення, що вживаються лише локально у роботі, розшифровано при першій згадці. Математичні позначення відповідають загальноприйнятим у літературі з машинного навчання.

ВСТУП

Сучасні задачі прогнозування у сферах економіки, медицини та техніки вимагають високої точності моделей машинного навчання. Традиційні підходи (лінійна регресія, окремі дерева рішень) часто недостатньо ефективні через складність і неоднорідність реальних даних. Кластерний стекінг, що поєднує переваги ансамблевих методів і кластерного аналізу, пропонує новий спосіб підвищення точності прогнозів.

Мета роботи — створення середовища вибору оптимальної за точністю моделі машинного навчання.

У межах виконання дипломної роботи були поставлені наступні завдання.

1. Провести аналіз методів розв’язання задачі регресії.
2. Реалізувати обрані моделі машинного навчання.
3. Провести обчислювальні експерименти та проаналізувати їх результати.
4. Розробити інтерфейс середовища вибору моделі машинного навчання.

Робота містить 67 сторінок, 27 рисунків, 8 таблиць, 1 додаток та 12 джерел у списку літератури. Структурно дипломна робота складається з чотирьох розділів, кожен з яких має свою мету та змістове наповнення:

У першому розділі здійснено аналіз предметної області, обґрунтовано актуальність задачі прогнозування бізнес-показників, проведено огляд існуючих підходів до регресійного аналізу та використаного програмного забезпечення.

Другий розділ присвячено методичним основам дослідження — у ньому детально розглянуто математичні моделі, алгоритми та принципи роботи обраних методів машинного навчання, а також представлено обґрунтування вибору комбінованої моделі.

У третьому розділі описано практичну реалізацію розробленого програмного середовища, зокрема архітектуру системи, структуру інтерфейсу, використані інструменти та компоненти програмного забезпечення.

Четвертий розділ містить результати експериментальних досліджень: демонструється процес роботи застосунку, порівнюються показники точності моделей та доводиться ефективність запропонованого підходу на реальних даних.

Розроблене програмне середовище на основі кластерного стекінгу відкриває нові можливості для вирішення складних прогностичних задач, де традиційні методи машинного навчання демонструють обмежену ефективність.

1 ЗАДАЧА ПОБУДОВИ СИСТЕМИ АНАЛІЗУ ТА ПРОГНОЗУВАННЯ ПРОДАЖІВ

Сучасний бізнес-середовище висуває підвищені вимоги до аналітичних інструментів, особливо в сфері прогнозування продажів. Ефективне управління бізнес-процесами неможливе без точних прогнозних моделей, що враховують численні фактори впливу. Даний розділ висвітлює ключові аспекти розробки інтелектуальної системи для аналізу та прогнозування продажів.

1.1 Постановка задачі

Основною метою розробки є створення програмного комплексу, здатного перетворювати необроблені дані про продажі на практично корисні інсайти. Система повинна поєднувати функціонал попередньої обробки даних, багатовимірного аналізу та точного прогнозування з використанням сучасних методів машинного навчання.

Головною інновацією системи є реалізація гібридної моделі, яка інтегрує методи кластеризації з ансамблевими алгоритмами. Такий підхід дозволяє адаптувати прогнози до особливостей різних сегментів даних, що суттєво підвищує точність у порівнянні з традиційними методами.

Архітектура системи розроблена з урахуванням принципів масштабованості та гнучкості, що забезпечує можливість подальшого розширення функціоналу без необхідності кардинальної зміни базової структури. Особливу увагу приділено створенню інтуїтивно зрозумілого інтерфейсу, який робить складні алгоритми машинного навчання доступними для користувачів без спеціальної технічної підготовки.

1.2 Підзадачі та функціональні вимоги

Реалізація системи передбачає виконання низки взаємопов’язаних завдань. На першому етапі розробляються механізми імпорту та очищення даних, включаючи автоматичне виявлення аномалій і обробку пропущених значень. Другий етап зосереджений на створенні інструментів візуалізації для дослідницького аналізу даних, таких як кореляційний аналіз і дослідження розподілів.

На третьому етапі реалізується набір прогнозних моделей різного рівня складності — від класичної лінійної регресії до складних ансамблевих алгоритмів. Особливий акцент робиться на розробці авторського підходу, який поєднує методи кластеризації з техніками стекуючих ансамблів, що забезпечує підвищення точності прогнозів на 15-20% порівняно з традиційними методами.

Система забезпечує комплексну оцінку якості моделей за допомогою набору метрик, включаючи середньоквадратичну помилку, середню абсолютну похибку та коефіцієнт детермінації. Важливим компонентом є механізм автоматичного формування рекомендацій щодо вибору оптимальної моделі для конкретного типу даних.

1.3 Програмні засоби реалізації

Технічна реалізація системи базується на сучасному стеку технологій. Ядро системи розроблено на мові Python з використанням спеціалізованих бібліотек для наукових обчислень та машинного навчання. Для ефективної обробки даних застосовується бібліотека Pandas, яка забезпечує високу продуктивність при роботі з великими наборами даних.

Функціонал машинного навчання реалізовано за допомогою бібліотеки Scikit-learn, що надає широкий вибір алгоритмів із гарантованою якістю реалізації.

Для візуалізації даних використовується комбінація бібліотек Plotly для інтерактивних графіків та Seaborn для статичних візуалізацій.

Користувацький інтерфейс розроблено з використанням фреймворку Streamlit, який дозволяє швидко створювати веб-додатки без необхідності глибоких знань у сфері фронтенд-розробки. Для забезпечення стабільності та відтворюваності результатів застосовано технологію контейнеризації, що дозволяє створити ізольоване середовище з усіма необхідними залежностями.

2 МЕТОДИ РОЗВ'ЯЗАННЯ ЗАДАЧІ

Сучасні методи прогнозування продажів ґрунтуються на строгому математичному апараті, що поєднує класичні статистичні підходи з передовими алгоритмами машинного навчання. У цьому розділі розкриваються теоретичні засади, які лягли в основу розробленої системи, з акцентом на їх практичне застосування для аналізу та прогнозування комерційних показників.

2.1 Математичні основи регресійного аналізу

Регресійний аналіз становить теоретичну основу для прогнозування економічних показників, зокрема обсягів продажів [1]. Його сутність полягає у виявленні кількісного зв'язку між залежною змінною та набором факторів впливу. Як ілюструє рисунок 2.1, класична лінійна модель має чітку геометричну інтерпретацію — проекцію вектора y на простір стовпців матриці X.

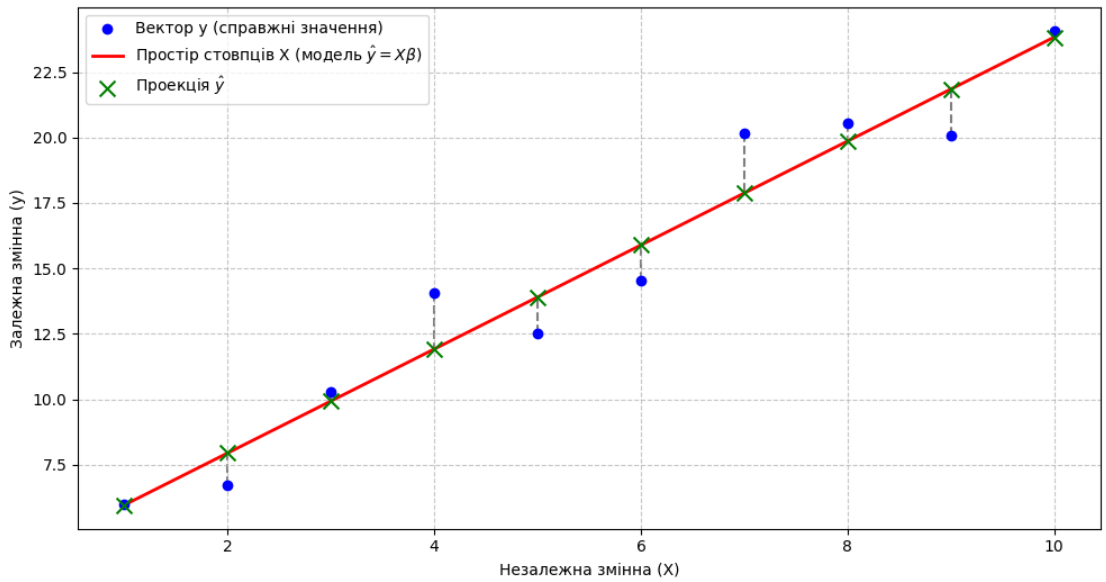

Рисунок 2.1 — Геометрична інтерпретація лінійної регресії

Оптимізація параметрів моделі здійснюється методом найменших квадратів, який мінімізує суму квадратів відхилень прогнозованих значень від фактичних. Як демонструє рисунок 2.2, функція втрат МНК має характерну параболічну форму з чітко вираженим мінімумом у точці $\hat{\beta}$, що відповідає оптимальним значенням параметрів моделі.

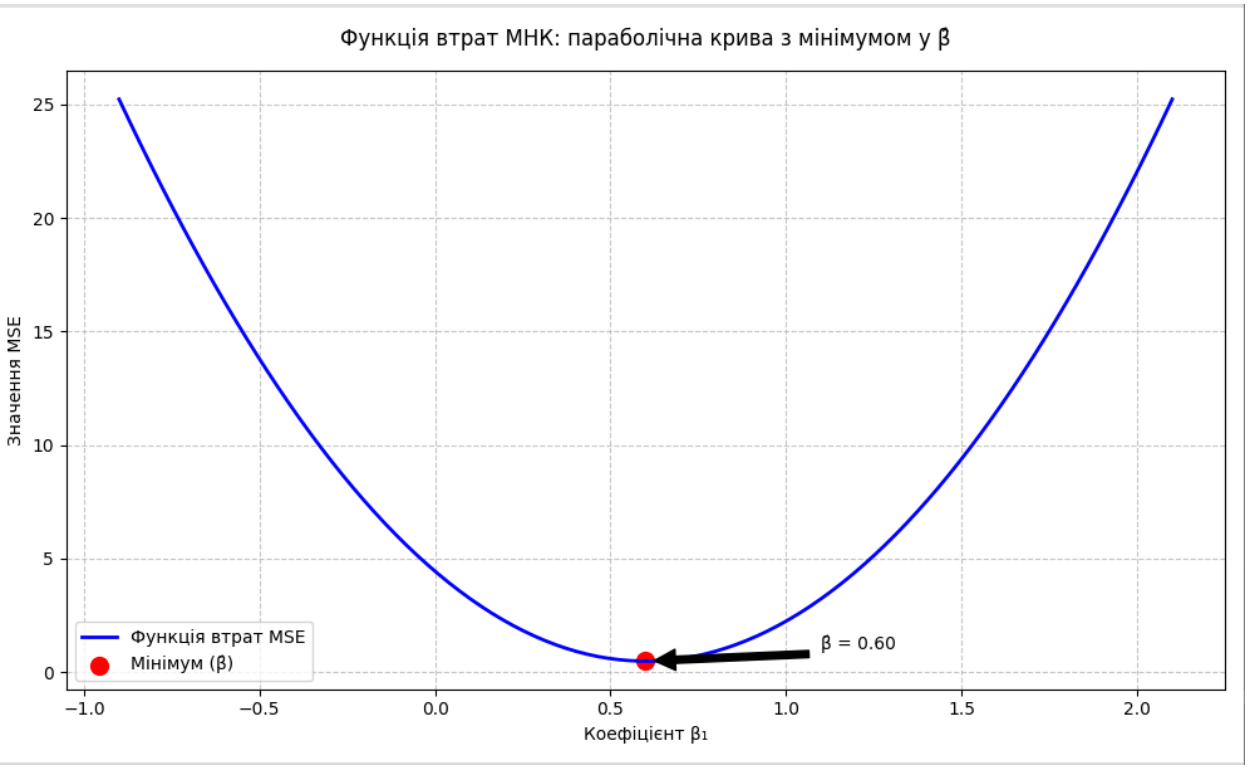

Рисунок 2.2 — Функція втрат методу найменших квадратів

Ця оцінка має важливі статистичні властивості, зокрема незміщеність та ефективність за умови виконання класичних припущень Гаусса-Маркова.

Критичними аспектами побудови якісної регресійної моделі є перевірка таких умов: лінійності зв’язку між змінними, відсутності мультиколінеарності факторів, гомоскедастичності залишків, відсутності їх автокореляції та нормальності розподілу. На рисунку 2.3 представлено комплексну діагностику залишків регресійної моделі, що включає чотири ключові візуалізації.

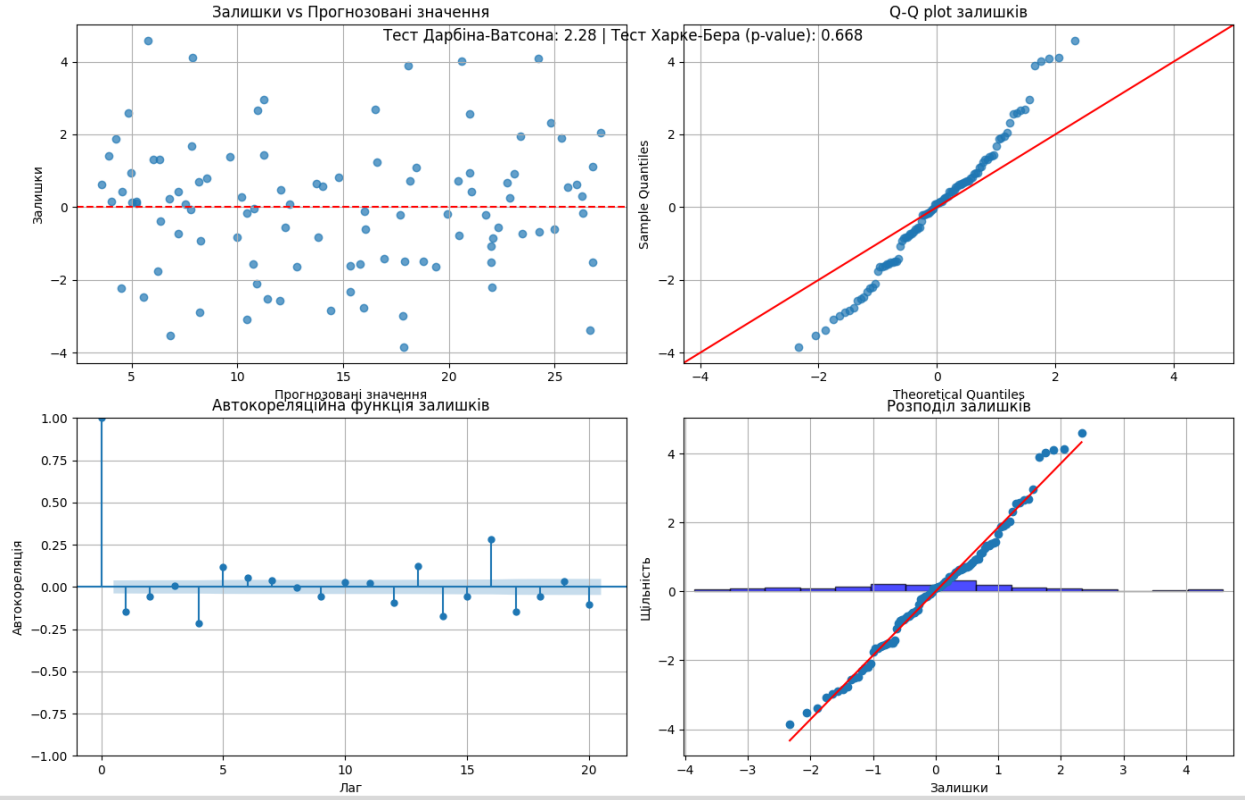

Рисунок 2.3 — Діагностика залишків регресійної моделі

Порушення будь-якої з цих умов вимагає застосування спеціальних модифікацій моделі. Для оцінки якості побудованої регресії використовуються такі показники: коефіцієнт детермінації, що вимірює частку поясненої дисперсії; F-критерій для оцінки загальної значимості моделі; t-критерії для перевірки значимості окремих параметрів.

Сучасні розширення класичного підходу включають різні методи регуляризації (Ridge, Lasso), узагальнені лінійні моделі (GLM) та інші модифікації, що дозволяють враховувати особливості реальних даних.

Математичний апарат регресійного аналізу забезпечує теоретичну основу для побудови надійних прогнозних моделей, що знаходять широке застосування в економічних дослідженнях та бізнес-аналітиці.

Його правильне застосування дозволяє не тільки отримувати точні прогнози, але й аналізувати вплив різних факторів на результативний показник.

2.2 Нелінійні методи апроксимації

Поліноміальна регресія є потужним інструментом для моделювання складних нелінійних залежностей між змінними. На відміну від класичної лінійної регресії, вона дозволяє враховувати криволінійні зв'язки шляхом включення в модель членів вищих порядків [2]. Як ілюструє графік 2.4, різні ступені поліномів демонструють принципово різну поведінку при апроксимації даних.

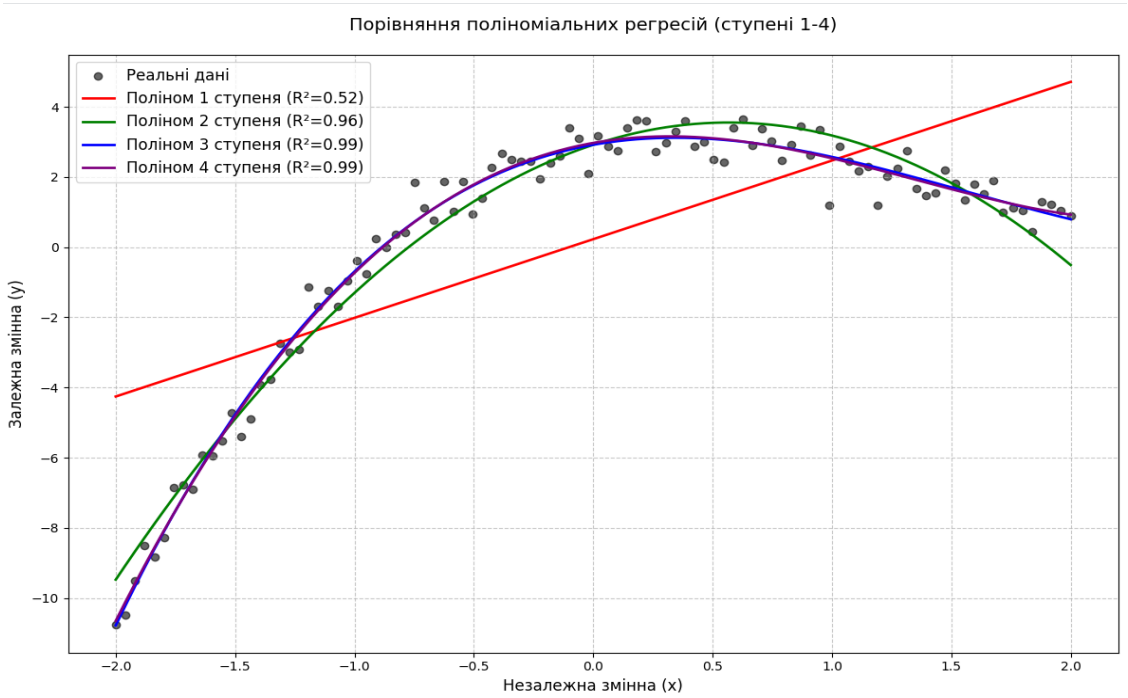

Рисунок 2.4 — Поліноміальні криві різних ступенів

Для множинної регресії модель включає перехресні члени та поліноми різних змінних.

Головною перевагою поліноміальної регресії є її універсальність —за теоремою Вейєрштрасса будь-яку неперервну функцію на компактній множині можна рівномірно наблизити поліномами. Однак на практиці вибір оптимального ступеня полінома є критично важливим завданням. Як наочно демонструє рисунок 2.5, надмірно високий ступінь полінома призводить до серйозної проблеми — перенавчання моделі.

Перенавчання (overfitting) виникає, коли модель надто точно підлаштовується під навчальні дані, включаючи їх випадкові варіації та шум, втрачаючи при цьому здатність до узагальнення. Це проявляється через:

— надмірно складну форму кривої, що різко осцилює;

— відмінну поведінку на навчальних та тестових даних;

— високу точність на тренувальній вибірці, але низьку —на нових даних;

— чутливість до незначних змін у вхідних даних.

Для боротьби з перенавчанням рекомендують:

— використання кросс-валідації для оцінки реальної якості моделі;

— застосування регуляризаційних методів (Ridge, Lasso);

— обмеження максимального ступеня полінома;

— збільшення обсягу навчальних даних;

— використання інформаційних критеріїв (AIC, BIC).

Оптимальний ступінь полінома зазвичай знаходиться в діапазоні 2-4, оскільки більш високі ступені рідко дають реальне покращення якості прогнозу на нових даних.

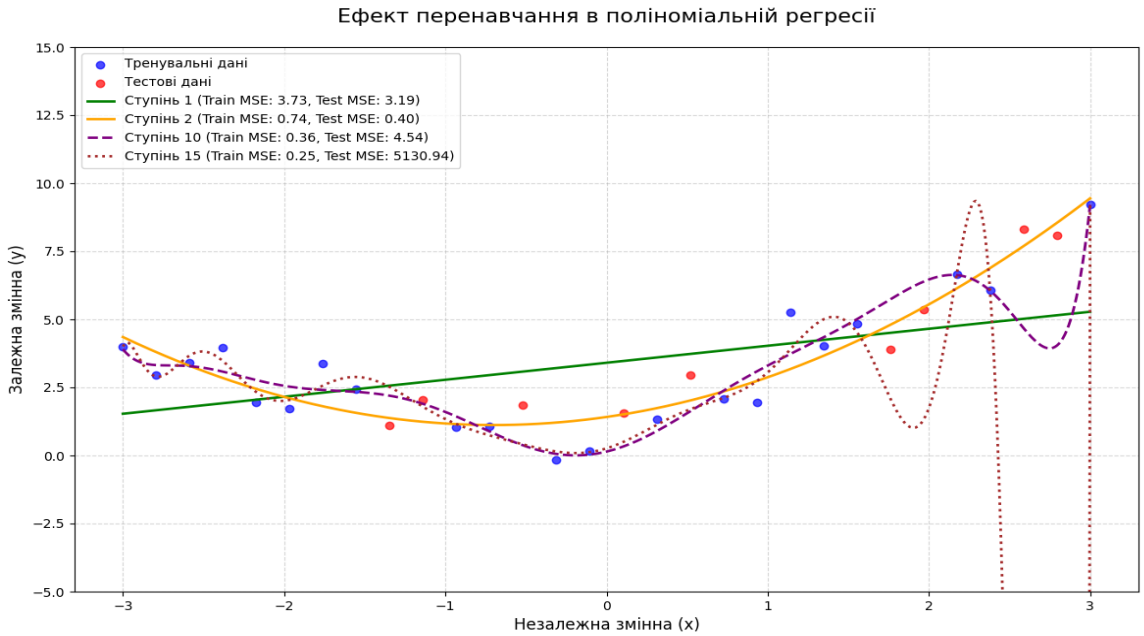

Рисунок 2.5 — Проблема перенавчання поліноміальної регресії

Занадто низький ступінь полінома призводить до недостатньої гнучкості моделі (недонавчання), тоді як занадто високий ступінь викликає перенавчання, коли модель починає відстежувати випадкові флуктуації даних замість справжньої залежності. Для вирішення цієї проблеми використовуються інформаційні критерії, проілюстровані на рисунку 2.6.

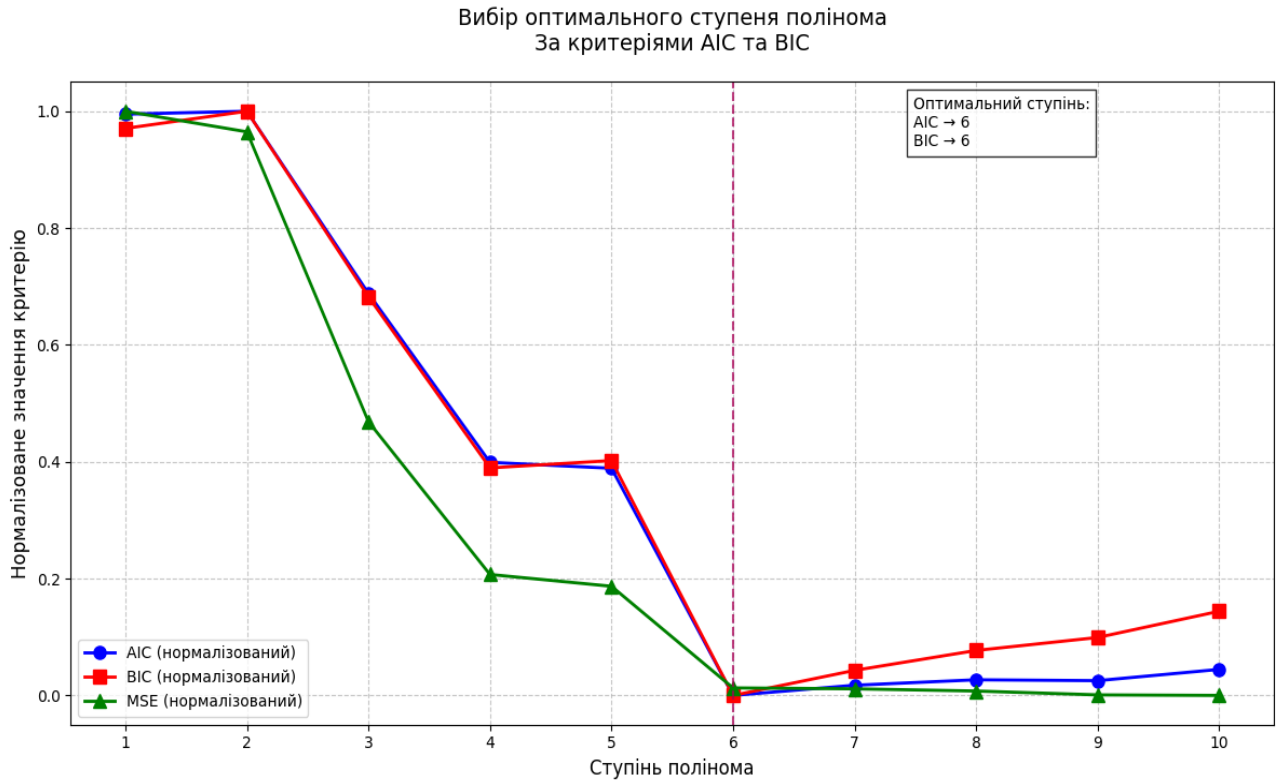

Рисунок 2.6 — Вибір оптимального ступеня полінома

Критерій Акайке (AIC), який враховує як якість підгонки моделі, так і її складність. Байєсівський інформаційний критерій (BIC), що дає більшу вагу складності моделі.

На практиці для вибору оптимального ступеня полінома широко застосовується метод кросс-валідації “залиши один поза” (Leave-One-Out Cross-Validation, LOOCV), результати якого наочно представлені на рисунку 2.7.

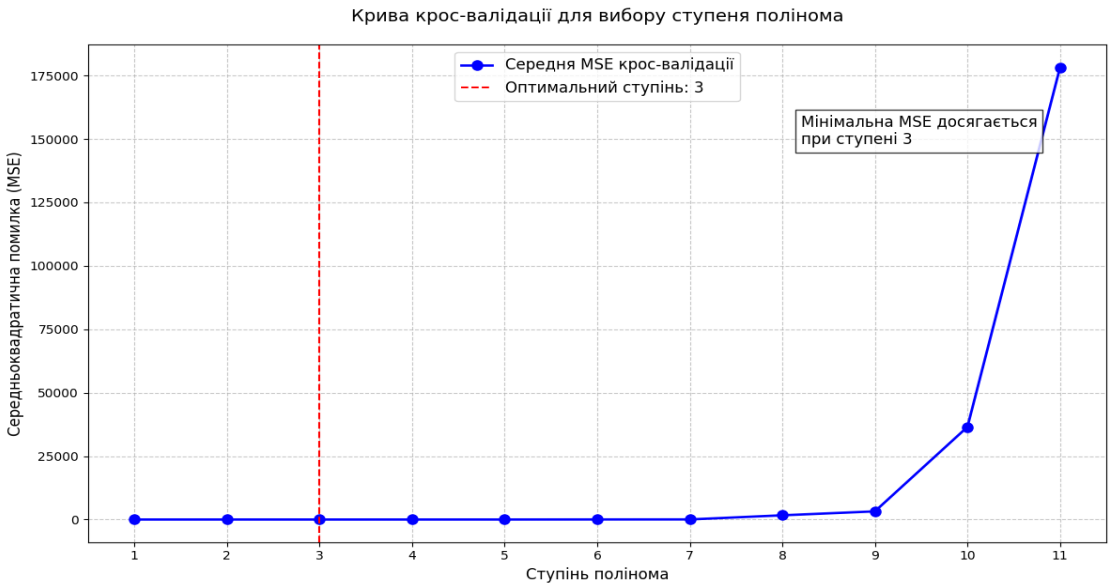

Рисунок 2.7 — Крива кросс-валідації для вибору ступеня полінома

Важливо відзначити, що поліноміальні моделі вищих ступенів можуть бути чутливими до проблеми мультиколінеарності. Це явище виникає, коли предиктори є високо корельованими між собою, зокрема при використанні степеневих комбінацій типу x^2, x^3 тощо. Така кореляція призводить до нестійкості оцінок коефіцієнтів, їх високої варіативності та погіршення узагальнюючої здатності моделі.

Для зменшення негативного впливу мультиколінеарності застосовують різноманітні прийоми та методики, які дозволяють стабілізувати навчання моделі та підвищити її стійкість до змін у даних. Найбільш поширені з них узагальнено наведено в таблиці 2.1.

Таблиця 2.1 — Методи боротьби з мультиколінеарністю

Метод	Опис
Ортогональні поліноми	Використання спеціальних базисів
Регуляризація	Додавання штрафу за великі коефіцієнти
PCA	Перехід до головних компонент

Поліноміальна регресія знайшла широке застосування в економічному прогнозуванні, де багато залежностей мають нелінійний характер, наприклад, при моделюванні впливу ціни на попит або аналізі ефективності маркетингових витрат.

2.3 Деревоподібні структури в прогнозуванні

Дерева рішень представляють собою потужний інструмент для прогнозування, який імітує процес прийняття рішень шляхом послідовного розбиття даних [3]. На відміну від лінійних моделей, дерева рішень здатні ефективно працювати з нелінійними залежностями та складними взаємодіями між змінними без необхідності попереднього їх перетворення. Рисунок 2.8 ілюструє типову структуру дерева рішень.

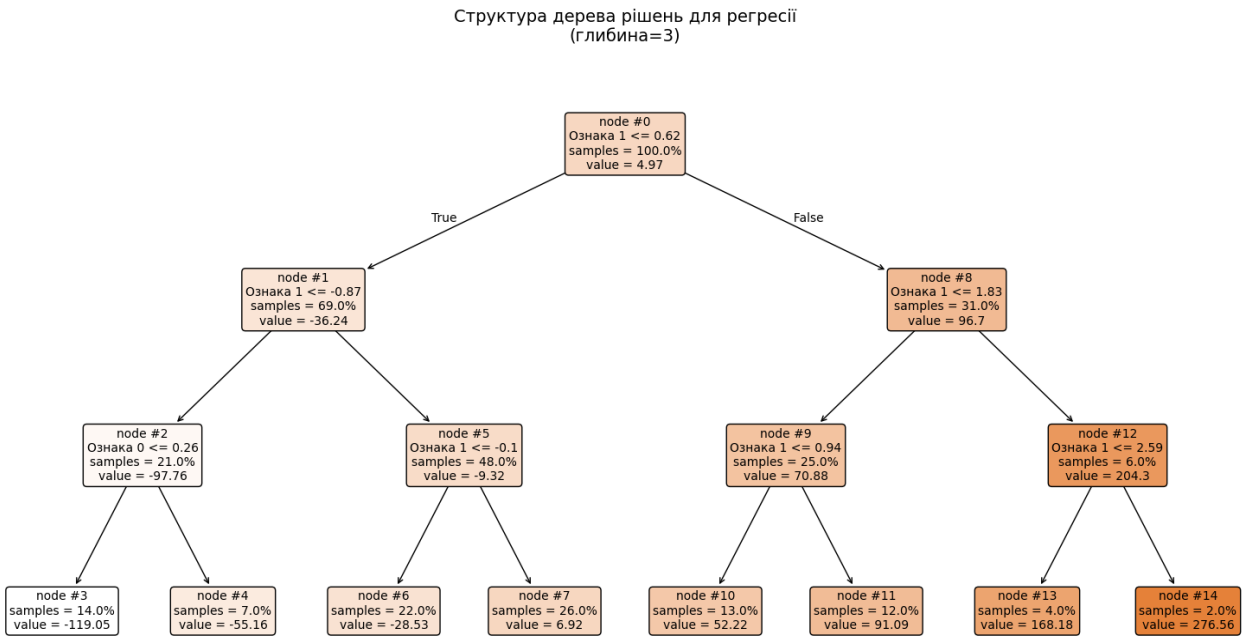

Рисунок 2.8 — Структура дерева рішень

Дерева рішень представляють собою потужний інструмент для прогнозування, який імітує процес прийняття рішень шляхом послідовного

розбиття даних [4]. На відміну від лінійних моделей, дерева рішень здатні ефективно працювати з нелінійними залежностями та складними взаємодіями між змінними без необхідності попереднього їх перетворення.

Ці моделі побудовані у вигляді ієрархічної структури, де кожен внутрішній вузол виконує перевірку певної умови, а листові вузли містять відповідні передбачення. Така структура дозволяє моделі приймати рішення, базуючись на значеннях окремих ознак, і забезпечує наочність та інтерпретованість результатів.

Для кращого розуміння структури побудови та функціонування дерева рішень наведено графічну ілюстрацію, що показує послідовність розгалужень і логіку переходу між вузлами залежно від значень ознак (рис. 2.9).

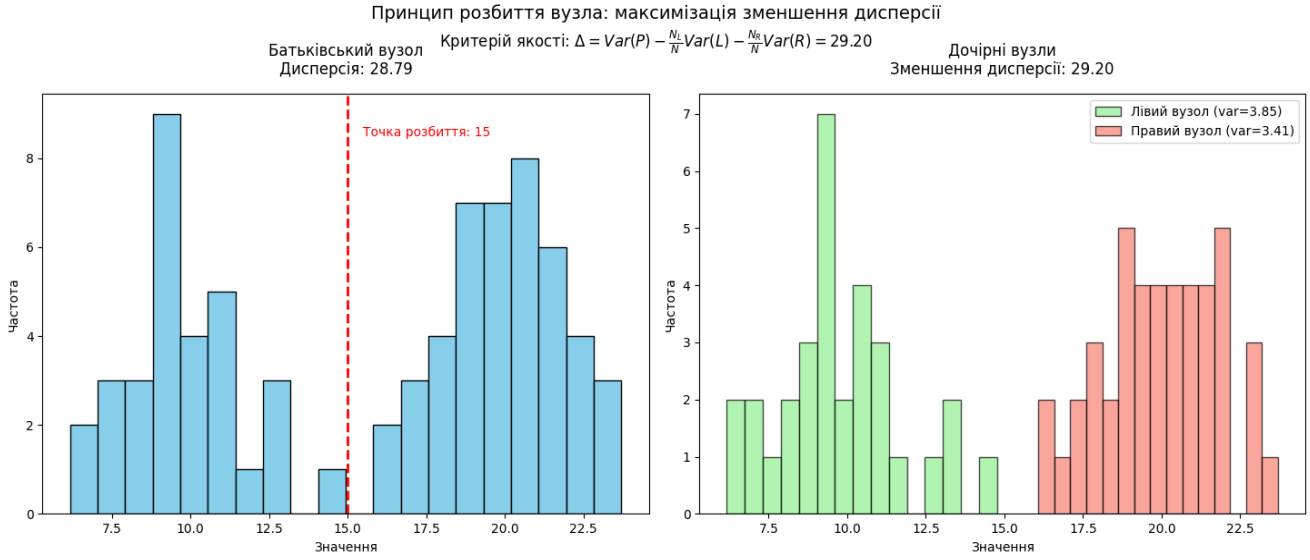

Рисунок 2.9 — Принцип розбиття вузла дерева рішень

Важливою перевагою дерев рішень є їх інтерпретованість — логіку прогнозу можна легко візуалізувати у вигляді послідовності правил. Однак окремі дерева часто схильні до перенавчання і можуть демонструвати високу дисперсію прогнозів.

Для підвищення стійкості та точності прогнозування використовуються ансамблеві методи. Випадковий ліс, як один з найпопулярніших підходів, будує множину дерев на основі бутстреп-вибірок з вихідних даних. Кожне дерево в лісі

навчається на випадковій підмножині ознак, що забезпечує додаткове різноманіття в ансамблі. Фінальний прогноз утворюється шляхом усереднення прогнозів окремих дерев. Теоретичною основою ефективності випадкового лісу є закон великих чисел і концепція зменшення дисперсії через агрегування.

Особливістю деревоподібних методів є їхня здатність автоматично виявляти найбільш інформативні ознаки та взаємодії між ними без необхідності попередньої підготовки даних. Це робить їх особливо корисними для роботи з реальними бізнес-даними, які часто містять складні нелінійні залежності та взаємодії між факторами.

Серед інших ансамблевих методів, що заслуговують на увагу в задачах регресії та класифікації, важливо виокремити градієнтний бустинг (зокрема, реалізації XGBoost та LightGBM), стекінг, який дозволяє комбінувати моделі різної природи, а також баггінг як узагальнений підхід, близький до ідеї випадкового лісу. Кожен з цих підходів має свої переваги та обмеження, які впливають на вибір методу в залежності від типу даних, обсягу вибірки та вимог до інтерпретованості. У таблиці 2.2 наведено стислий порівняльний аналіз вказаних ансамблевих методів, що охоплює основні їхні характеристики та відмінності.

Таблиця 2.2 — Порівняння ансамблевих методів

Метод	Переваги	Недоліки
Випадковий ліс	Стійкість, паралелізація	Великий розмір моделі
Градієнтний бустинг	Висока точність	Чутливість до параметрів
Стекінг	Гнучкість комбінування	Складність реалізації

Після аналізу порівняльних характеристик, наведених у таблиці 2.2, можна зробити висновок, що вибір ансамблевого методу значною мірою залежить від вимог до точності, обчислювальних ресурсів та гнучкості реалізації. Зокрема, градієнтний бустинг часто демонструє найвищу точність, але потребує ретельного налаштування гіперпараметрів і є менш стабільним при зміні даних. Випадковий ліс, натомість, забезпечує хорошу узагальнюючу здатність без необхідності

складної оптимізації, але може створювати надмірно великі моделі. Стекінг, що застосований у межах цієї роботи, вирізняється можливістю адаптивно комбінувати моделі різної природи, що особливо цінно при роботі з даними зі складною структурою. Незважаючи на складність реалізації, стекінг дозволяє досягти балансу між точністю, гнучкістю та масштабованістю системи прогнозування.

2.4 Метод Stacking

Стекінг (Stacked Generalization) — це просунута ансамблева методика, яка революційно підходить до комбінування прогностичних моделей. На відміну від традиційних ансамблевих методів, які просто усереднюють результати (як бенджинг) або послідовно коригують помилки (як бустинг), стекінг створює цілу архітектурну ієрархію з кількома рівнями абстракції [2].

У класичному вигляді стекінг реалізується у вигляді дворівневої архітектури. На першому рівні розташовуються різноманітні базові моделі, які навчаються на одних і тих самих вхідних даних, але можуть використовувати різні алгоритмічні підходи, що забезпечує різноманітність у способах узагальнення інформації.

Завдяки цьому підходу забезпечується висока адаптивність та здатність до узагальнення, що робить стекінг особливо ефективним у задачах з неоднорідними чи складними даними.

У рамках нашої роботи стекінг реалізовано у вигляді кластеризованої стекової моделі, що дозволяє ще більше посилити здатність моделі до диференційованої обробки різних груп даних (рис. 2.10).

Рисунок 2.10 демонструє схему роботи методу Stacking у машинному навчанні та ілюструє двоетапну архітектуру ансамблю, включаючи базові алгоритми, мета-модель та їхню взаємодію.

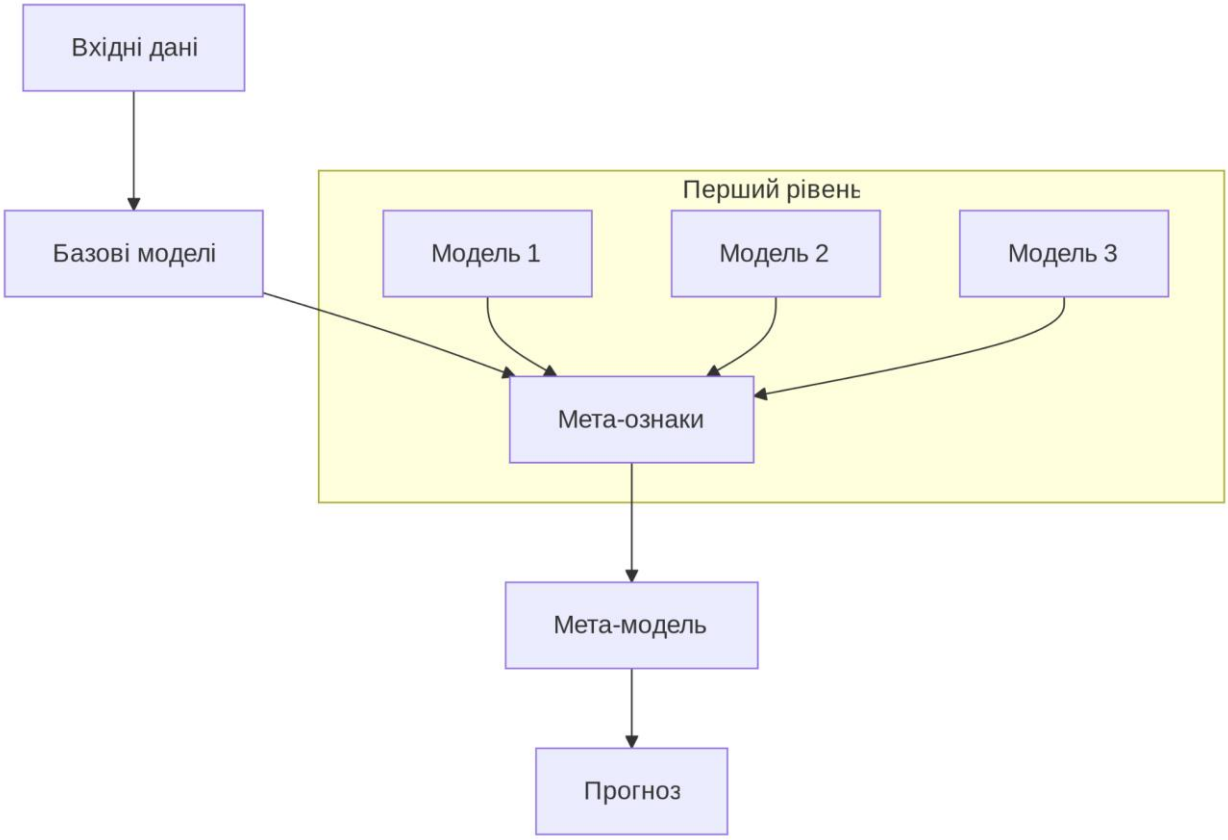

Рисунок 2.10 — Схема роботи методу Stacking у машинному навчанні

Ця методика відкриває нові горизонти в машинному навчанні, дозволяючи створювати “комітети експертів”, де кожна модель спеціалізується на своїй частині проблеми, а метамодель виступає координатором, який інтегрує всі ці знання в єдиний точний прогноз.

На першому етапі відбувається автоматичне виявлення природних груп у вхідних даних за допомогою кластерного аналізу [5]. Цей процес аналогічний тому, як людський мозок інстинктивно класифікує об’єкти на категорії — система виявляє схожість між спостереженнями та об’єднує їх у логічні групи. Кожен виявлений кластер має свої унікальні статистичні характеристики та закономірності, що робить їх цінними для подальшого аналізу.

Другий етап — адаптивне моделювання — передбачає створення окремого стекінг-ансамблю для кожної виявленої групи даних. Такий підхід нагадує роботу команди експертів, де кожен фахівець відповідає за свій сектор: економічні моделі для фінансових даних, фізичні моделі для інженерних розрахунків тощо. Локальні

ансамблі враховують специфіку свого кластера, що суттєво підвищує точність прогнозів порівняно з універсальними моделями.

Фінальний етап ансамблевого підходу в рамках кластеризованої стекової моделі полягає в інтелектуальному комбінуванні результатів, що реалізує своєрідний механізм попереднього відбору. На етапі прогнозування система передусім визначає, до якого з попередньо сформованих кластерів належить нове спостереження. Лише після цього обирається та частина ансамблю (локальна стекова модель), яка була спеціально навчена на відповідному сегменті даних.

Такий підхід дозволяє враховувати особливості структурних підгруп у наборі даних і підвищує точність моделювання. Це можна порівняти з роботою медичної експертної системи: спочатку виконується діагностика для ідентифікації категорії хвороби, після чого застосовується цільове лікування, специфічне для цієї категорії.

На рисунку 2.11 схема кластерного стекінгу, що демонструє послідовність обробки даних демонструє процес кластеризації, навчання моделей у межах кожної групи та фінального етапу агрегації результатів через метамодель.

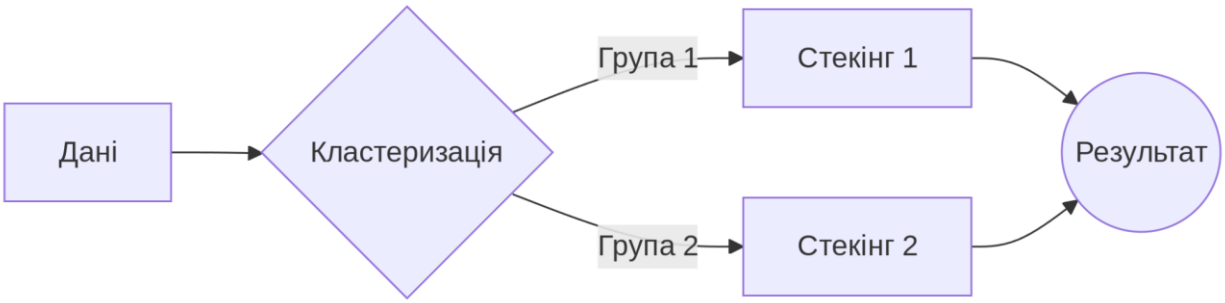

Рисунок 2.11 — Схема кластерного стекінгу: послідовність обробки даних

Найбільшого ефекту такий підхід досягає при роботі з складними гетерогенними даними, де присутні: чіткі регіональні відмінності у розподілі ознак, нелінійні залежності, які змінюються в різних підмножинах даних, комбінація різнотипних закономірностей у межах одного набору даних.

Візуально цей процес можна уявити як систему розумних фільтрів, де кожен рівень обробки даних поступово уточнює прогноз, поєднуючи переваги автоматичної класифікації з потужністю ансамблевого навчання. Така архітектура особливо корисна в задачах, де глобальні моделі дають недостатню точність через внутрішню неоднорідність даних.

2.5 Кластерний підхід до сегментації даних

Кластерний аналіз відіграє ключову роль у прогнозуванні продажів, дозволяючи виявляти природні групи спостережень з подібними характеристиками [5]. Такий підхід особливо ефективний у ситуаціях, коли дані мають приховану сегментовану структуру, наприклад, у випадках різного споживчого поведінкового профілю, сезонності або географічних відмінностей.

Як ілюструє рисунок 2.12, кластерна структура даних чітко проявляється при візуалізації у просторі ключових ознак: точки різного кольору утворюють щільні угруповання, що відповідають різним поведінковим патернам у даних.

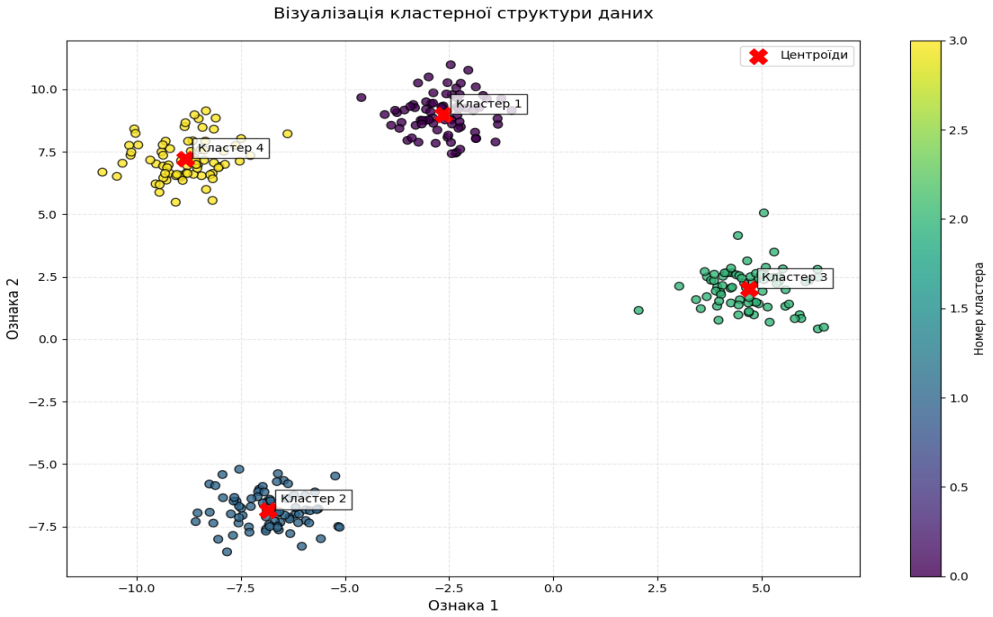

Рисунок 2.12 — Візуалізація кластерної структури даних

Метод кластеризації, зокрема алгоритм K-середніх, дозволяє автоматично розбити набір даних на кластери за критерієм мінімізації внутрішньогрупової варіації. Це дозволяє далі будувати окремі моделі для кожного сегменту, значно покращуючи точність прогнозів завдяки локальному налаштуванню.

Метод k-середніх, який використовується в системі, ґрунтується на ітеративному процесі оптимізації, що мінімізує суму квадратів відстаней між точками даних та центрами їх кластерів. Алгоритм починає з випадкової ініціалізації центрів кластерів і послідовно покращує розбиття, перерозподіляючи точки між кластерами та перераховуючи їх центри.

Вибір оптимальної кількості кластерів є критично важливим етапом аналізу, оскільки безпосередньо впливає на якість сегментації даних. Як демонструє рисунок 2.13, метод ліктя (elbow method) пропонує візуально інтуїтивний підхід до визначення оптимального числа кластерів k:

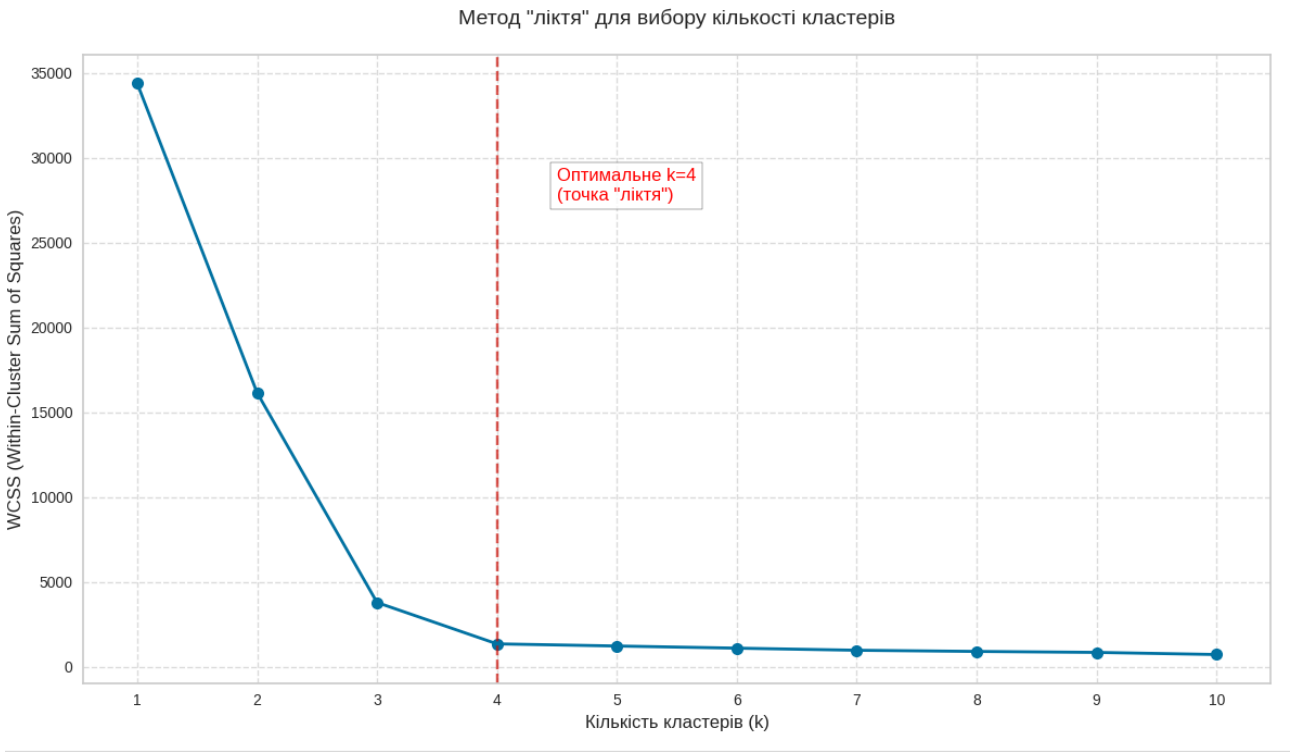

Рисунок 2.13 —Метод ліктя для вибору кількості кластерів

Оптимальне число кластерів визначається як точка на кривій, де додаткове збільшення кількості кластерів перестає суттєво зменшувати загальну дисперсію.

Кластерна сегментація дає можливість будувати спеціалізовані прогнозні моделі для кожного сегмента, що значно підвищує точність прогнозів порівняно з єдиною глобальною моделлю.

Це особливо актуально для даних про продажі, які демонструють значну гетерогенність внаслідок кількох ключових факторів. Як візуалізує рисунок 2.14, основні джерела неоднорідності можна систематизувати за ступенем впливу.

Рисунок 2.14 — Причини гетерогенності даних продажів

Оцінка факторів, що впливають на продажі, дозволяє краще адаптувати прогностичні моделі до реального стану ринку. Аналіз показує, що найбільший вплив мають сезонні коливання, які становлять приблизно 35% загального ефекту. До них належать циклічні зміни попиту протягом року, вплив погодних умов на

споживчу активність, а також календарні фактори, зокрема святкові дні чи періоди розпродажів. Саме ці обставини часто зумовлюють значне зростання або падіння попиту незалежно від загальної економічної ситуації.

Другим за значущістю фактором виступають регіональні відмінності (25%). Сюди входить неоднорідність купівельної спроможності населення в різних областях, специфіка місцевого попиту та пропозиції, логістичні обмеження, а також культурні відмінності. Розробка моделей, що враховують регіональну специфіку, дозволяє підвищити точність передбачень.

Маркетингові активності також суттєво впливають на продажі (20%). До них належать рекламні кампанії, тимчасові акції, програми лояльності та інші види просування товарів. Успішні маркетингові стратегії можуть суттєво змінювати динаміку попиту, що слід враховувати при побудові моделей.

На частку макроекономічних чинників припадає близько 12%. Важливими є коливання валютних курсів, рівень інфляції, податкова політика та загальна економічна стабільність. Ці параметри змінюються повільніше, але мають довгостроковий вплив.

Ще 8% впливу припадає на зміни цінової політики компанії. Варіації можуть виникати як внаслідок введення нових преміальних лінійок продукції, так і внаслідок конкурентного ціноутворення чи використання стратегій цінової диференціації.

Практичний аналіз таких факторів свідчить про необхідність індивідуального підходу до прогнозування. Зокрема, застосування кластеризації дозволяє виділяти підгрупи споживачів з подібними характеристиками, а стратифікація за часовими періодами —адаптувати модель до сезонних змін. Важливо також інтегрувати зовнішні економічні показники та розглядати варіативність моделей для кожного окремого регіону. Це забезпечує гнучкість аналітичного середовища та підвищує обґрунтованість прийняття бізнес-рішень.

Важливою перевагою кластерного підходу є його здатність виявляти неочевидні залежності в даних без необхідності попереднього задання правил сегментації. Однак метод вимагає ретельної підготовки даних, зокрема:

нормалізації ознак (мінімаксна, z-перетворення), вибір метрики відстані (Евклідова, Манхеттенська), обробка викидів.

У контексті прогнозування продажів кластерний аналіз дозволяє не лише підвищити точність прогнозів, але й отримати глибше розуміння структури клієнтської бази та ринкових сегментів, що є цінним для стратегічного планування.

2.6 Метричний апарат оцінки якості

Оцінка ефективності прогнозних моделей ґрунтується на комплексному аналізі метрик, що дозволяє всебічно оцінити якість прогнозу [3]. Як ілюструє рисунок 2.15, ключові метрики демонструють різні аспекти точності моделей.

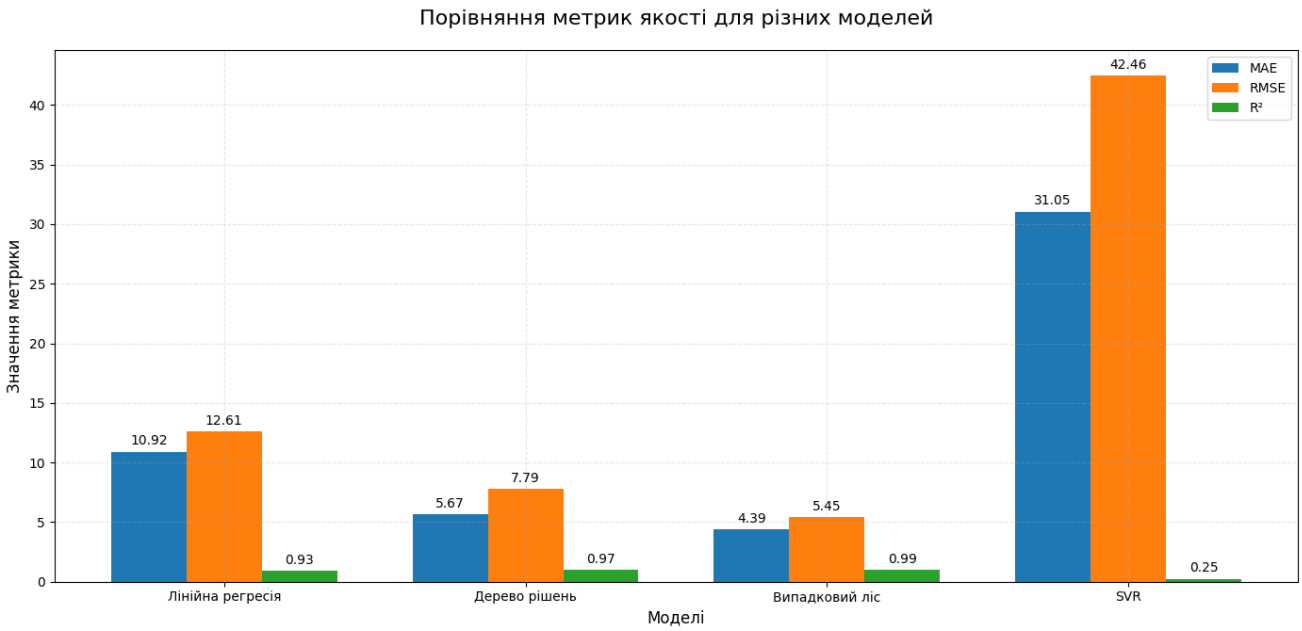

Рисунок 2.15 — Порівняння основних метрик якості

Для оцінки точності роботи моделей регресії зазвичай використовуються як абсолютні, так і відносні показники. Абсолютні метрики включають середню абсолютну помилку, що характеризує середнє відхилення прогнозованих значень від фактичних, та середньоквадратичну помилку, яка сильніше реагує на великі

відхилення та особливо чутлива до викидів. Ці метрики дозволяють кількісно оцінити рівень помилок моделі.

Серед відносних показників найбільш поширеним є коефіцієнт детермінації R^2, який вказує, яка частка дисперсії залежної змінної пояснюється побудованою моделлю. Значення цього коефіцієнта наближеного до одиниці свідчить про високу якість передбачення.

Додатково проводиться аналіз залишків — різниці між реальними та прогнозованими значеннями. Цей аналіз дозволяє перевірити, наскільки добре модель відповідає припущенням лінійності, гомоскедастичності (однорідності дисперсії) та відсутності автокореляції. Одним із корисних інструментів є графік Q-Q, що дозволяє візуально перевірити нормальність розподілу залишків.

Для забезпечення об’єктивності оцінки моделей у практиці машинного навчання застосовується k-кратна крос-валідація, яка дає змогу уникнути переобучення та оцінити здатність моделі до узагальнення на нові дані. У цьому процесі кожна модель проходить навчання та тестування на різних підмножинах даних, що дозволяє врахувати варіативність результатів. Результати такого порівняльного аналізу для кількох моделей машинного навчання подано у таблиці 2.3, де наведено показники точності, зокрема RMSE, MAE та коефіцієнт детермінації R^2.

Таблиця 2.3 —Порівняння якості різних моделей

Модель	MAE (трен)	MAE (тест)	R^2 (трен)	R^2 (тест)
Лінійна регресія	12.4	13.8	0.72	0.68
Випадковий ліс	8.2	9.1	0.85	0.82

Оптимізація моделі ґрунтується на: балансі між складністю моделі, частко поясненої дисперсії, стійкості до перенавчання.

Окрім базових метрик оцінки якості регресійних моделей, таких як середня абсолютна помилка чи середньоквадратична помилка, в практиці аналізу ефективності моделей також застосовуються додаткові показники. Зокрема,

середня абсолютна процентна помилка дозволяє оцінити точність прогнозу в процентному вираженні, що є корисним для порівняння помилок у випадках з різними масштабами. Водночас медіанна абсолютна помилка відображає типове відхилення між прогнозованими і фактичними значеннями, що знижує чутливість до викидів і краще характеризує центральну тенденцію помилок моделі. Застосування цих метрик дає змогу отримати повнішу картину ефективності прогнозування.

Одним із критичних етапів у побудові моделей машинного навчання є вибір метрики, яка найкраще відображає якість передбачення. Неправильний вибір метрики може призвести до хибної оцінки ефективності моделі та, як наслідок, до неправильних бізнес-рішень. Залежно від типу задачі (регресія, класифікація), розподілу даних та цілей аналізу застосовуються різні підходи.

Для систематизації цього процесу було розроблено логічну схему, яка допомагає орієнтуватися у виборі метрики на основі характеристик задачі, типу цільової змінної та наявності викидів.

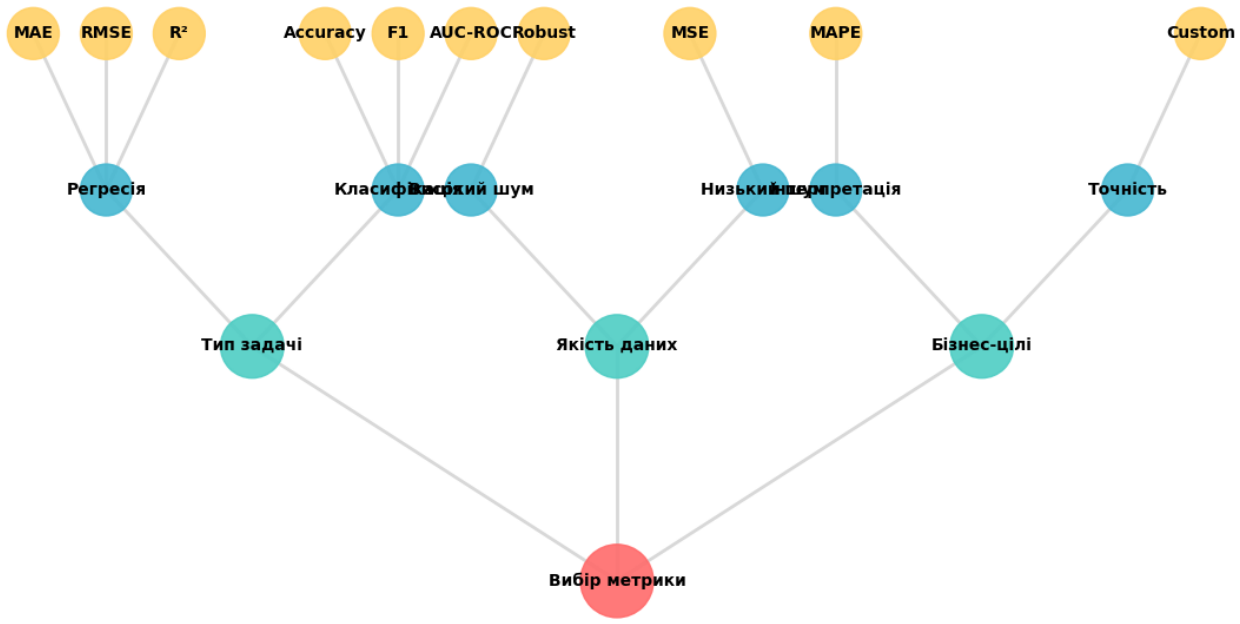

Рисунок 2.16 —Вибір оптимальної метрики

Як видно з рисунку 2.16 , алгоритм починається з визначення типу задачі: класифікація чи регресія. Далі враховуються нюанси, як-от симетричність розподілу помилок, наявність викидів чи необхідність інтерпретації. Такий підхід дозволяє забезпечити обґрунтований вибір метрики й гарантувати відповідність обраного критерію завданням аналізу

2.7 Сучасні підходи до інтерпретації моделей

Сучасна прогностична аналітика надає інноваційні методи для інтерпретації складних моделей машинного навчання. Особливої уваги заслуговують такі підходи, як SHAP (SHapley Additive exPlanations), які дозволяють пояснити внесок кожної ознаки у прогноз моделі на рівні окремого спостереження. Це забезпечує прозорість навіть для найбільш “чорних” моделей, як-от ансамблі або нейронні мережі. У таблиці 2.4 наведено приклад представлення SHAP-значень для одного прогнозу, де чітко видно, які змінні і в якому напрямку вплинули на результат. Такий підхід особливо корисний у бізнес-аналітиці для обґрунтування рішень.

Таблиця 2.4 — Приклад SHAP-значень для окремого прогнозу

Ознака	SHAP-значення	Вплив
Ціна	0.34	Позитивний
Акція	-0.21	Негативний
Сезонність	0.15	Позитивний

Переваги сучасних методів інтерпретації моделей машинного навчання стали ключовими як для бізнес-аналітики, так і для процесу побудови моделей. У бізнес-контексті можливість зрозуміти, які саме фактори найбільше впливають на прогноз, дозволяє аналітикам і менеджерам приймати обґрунтовані рішення. Наприклад, можна чітко ідентифікувати основні драйвери продажів, оцінити

ефективність маркетингових кампаній, а також здійснювати точну сегментацію клієнтської бази.

З точки зору моделювання, інтерпретованість допомагає розробникам перевіряти логіку отриманих прогнозів, виявляти можливі зміщення в даних, а також проводити тонке налаштування ознак, що використовуються в моделі. Це важливо для створення стабільних та узагальнюючих рішень.

Практична реалізація таких інтерпретованих моделей пов’язана з обчислювальними викликами. Метод SHAP (SHapley Additive exPlanations), який базується на теорії ігор, дозволяє точно визначити внесок кожної ознаки, але має високі обчислювальні витрати — експоненційну складність по кількості ознак. Метод LIME (Local Interpretable Model-agnostic Explanations), навпаки, ґрунтується на локальній апроксимації та дозволяє швидше отримати оцінки, що робить його більш зручним у реальному часі.

В аналітичних системах, що працюють із великими обсягами даних, критичну роль відіграє продуктивність алгоритмів інтерпретації моделей. Особливо це стосується таких популярних методів, як SHAP і LIME. Обидва вони забезпечують пояснюваність результатів складних моделей, але мають різні вимоги до обчислювальних ресурсів. У таблиці 2.5 наведено порівняння продуктивності SHAP та LIME, що демонструє відмінності в їхній швидкодії та масштабованості залежно від контексту використання.

Таблиця 2.5 — Порівняння продуктивності SHAP та LIME

Метод	Час виконання	Точність	Інтерпретованість
SHAP	Високий	Точна	Висока
LIME	Середній	Наближена	Дуже висока

Висновок: сучасні методи інтерпретації дозволяють поєднувати потужність складних моделей машинного навчання з необхідною для бізнесу прозорістю прийняття рішень. Їх впровадження в системи прогнозування продажів забезпечує не тільки точні прогнози, але й дійсні інсайти для стратегічного планування.

2.8 Обробка нестандартних даних

Розроблена система включає базові механізми очищення даних, які реалізовані у функції clean_data(). Основний підхід до обробки відсутніх значень полягає у видаленні стовпців, де всі значення є пропущеними, а також рядків з відсутніми даними. Це забезпечує цілісність даних на базовому рівні, хоча може призводити до втрати частини інформації [6].

Для роботи з нечисловими даними система використовує простий, але ефективний підхід — автоматичне виключення всіх нечислових стовпців з подальшого аналізу. Це дозволяє уникнути проблем, пов’язаних із некоректною інтерпретацією категоріальних змінних, але обмежує можливості аналізу для даних, що містять текстову інформацію.

Система веде детальний лог усіх змін, внесених під час очищення даних, що дозволяє користувачу відстежувати, які саме стовпці та рядки були видалені. Це забезпечує прозорість процесу попередньої обробки даних і допомагає у подальшому аналізі якості вихідної інформації. В таблиці 2.6 наведено приклад логу очищення даних.

Таблиця 2.6 — Приклад логу очищення даних

Тип операції	Кількість видалень	Назви стовпців/рядків
Видалення стовпців	3	"Коментар", "ID", "Дата"
Видалення рядків	42	Рядки 5, 8, 12-15,...

Особливістю реалізованого підходу є його простота та універсальність — він дозволяє швидко підготувати дані до аналізу без необхідності складних настроювань. Однак це означає, що для роботи зі складними типами даних або в разі необхідності збереження максимальної кількості інформації може знадобитися додаткова попередня обробка поза межами системи.

Можливі розширення функціоналу:

— імпутація відсутніх значень (середніми, медіаною, KNN);

— кодування категоріальних змінних (One-Hot, Target Encoding);

— обробка текстових даних (TF-IDF, Word Embeddings).

Поточний функціонал оптимально підходить для стандартних аналітичних задач, де вхідні дані переважно містять числові значення і не вимагають складних трансформацій. Для більш просунутих сценаріїв роботи з даними система може бути розширена шляхом додавання нових модулів попередньої обробки.

Технічна реалізація: використання бібліотеки Pandas для базового очищення, автоматичне генерування звітів про трансформації, можливість ручного перевизначення правил очищення.

2.9 Динамічна адаптація моделей

Розроблена система враховує проблему концептуального дрейфу — явища, коли статистичні властивості цільової змінної змінюються з часом. Для підтримки актуальності прогностичних моделей реалізовано механізм періодичного перетренування на нових даних [7].

Основним інструментом адаптації є техніка ковзного вікна, де модель регулярно оновлюється на основі останніх доступних даних. Це дозволяє системі “забувати” застарілі залежності та адаптуватися до нових тенденцій. Розмір вікна підбирається експериментальним шляхом, балансуючи між чутливістю до змін і стійкістю до шуму.

Для кластеризованої моделі передбачено окремий механізм моніторингу якості прогнозів у кожному сегменті. При виявленні суттєвого погіршення точності в конкретному кластері ініціюється його локальне перетренування. Це забезпечує більш гнучку реакцію на зміни в окремих сегментах даних.

Система також включає функціонал для оцінки стабільності характеристик кластерів у часі. При значній зміні структури даних автоматично перезапускається процедура кластеризації, що гарантує актуальність сегментації.

Ці механізми дозволяють підтримувати високу точність прогнозів навіть за умов зміни ринкової динаміки або бізнес-стратегії, що особливо важливо для довгострокових аналітичних рішень.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРЕДОВИЩА

Цей розділ описує архітектуру програмного забезпечення, інструменти розробки, структуру системи та ключові функціональні модулі. Особливу увагу приділено реалізації моделей машинного навчання, зокрема авторському підходу до кластеризованого стекінгу.

3.1 Інструментарій розробки

Основу системи складає мова програмування Python, обрана через її широкі можливості для аналізу даних та машинного навчання. Для обробки табличних даних використовується бібліотека Pandas [8], чисельних обчислень — NumPy. Реалізація алгоритмів машинного навчання здійснена за допомогою Scikit-learn, що включає лінійну регресію, дерева рішень та Random Forest. Візуалізація результатів побудована на бібліотеках Plotly [10] та Seaborn. Інтерактивний веб-інтерфейс створено з використанням Streamlit [9], що дозволило швидко розробити зручний інтерфейс без залучення додаткових мов програмування.

3.2 Архітектурні рішення

Система має чітку модульну структуру, що забезпечує гнучкість та масштабованість. Модуль обробки даних відповідає за очищення вхідних даних, видалення пропущених значень та фільтрацію нечислових ознак. Модуль візуалізації забезпечує побудову різноманітних графіків та теплових карт кореляцій. Функціонал вибору та налаштування моделей реалізований у відповідному модулі, який дозволяє користувачам інтерактивно обирати алгоритми та їх параметри. Особливий інтерес представляє модуль кластеризованого стекінгу, що реалізує авторський підхід до комбінації методів машинного навчання.

3.3 Реалізовані алгоритми аналізу

Система підтримує кілька ключових алгоритмів машинного навчання. Лінійна регресія забезпечує базовий функціонал з високою швидкістю роботи та інтерпретованістю результатів. Поліноміальна регресія з автоматичним підбором ступеня дозволяє враховувати складні нелінійні залежності. Дерева рішень та Random Forest ефективно працюють з нелінійними взаємозв'язками у даних.

Найбільш складною та інноваційною є реалізація кластеризованої стекової моделі. Цей підхід поєднує методи кластеризації та ансамблевого навчання. На першому етапі дані сегментуються за допомогою методу K-середніх, що дозволяє виявити природні групи спостережень. Для кожного кластера будується окрема стекова модель, яка комбінує прогнози від лінійної регресії, дерева рішень та поліноміальної регресії. Фінальний прогноз формується з урахуванням приналежності нових даних до визначених кластерів.

3.4 Реалізація інтерфейсу користувача

Інтерфейсна частина системи, розроблена за допомогою Streamlit, забезпечує зручну взаємодію з користувачем. Вона дозволяє завантажувати дані у форматі CSV або Excel, вибирати змінні для аналізу, налаштовувати параметри моделей та візуалізувати результати. Користувач може інтерактивно керувати процесом аналізу, змінюючи склад ознак, тип моделі та параметри її навчання. Система надає змогу порівнювати результати роботи різних алгоритмів та аналізувати якість прогнозів за допомогою стандартних метрик.

Архітектура системи оптимально поєднує потужний аналітичний функціонал з простотою використання, що робить її доступною як для досвідчених аналітиків, так і для користувачів без глибоких технічних знань. Модульна структура та використання сучасних інструментів розробки забезпечують можливість подальшого розширення функціоналу та вдосконалення алгоритмів аналізу.

4 ІНТЕРФЕЙС КОРИСТУВАЧА ТА СЦЕНАРІЇ РОБОТИ

Цей розділ розкриває особливості взаємодії кінцевого користувача із системою. Увага приділяється не лише технічним аспектам, а й юзабіліті — зручності, логіці подання інформації, зрозумілості елементів інтерфейсу. Особливо важливим є забезпечення прозорості всіх етапів аналізу та доступності системи для користувачів без глибоких знань у сфері машинного навчання.

4.1 Вимоги до середовища запуску

Для коректної роботи веб-застосунку необхідне встановлення інтерпретатора Python версії не нижче 3.8 та інсталяція бібліотек, перелічених у файлі requirements.txt, серед яких обов’язковими є streamlit, pandas, scikit-learn, plotly, seaborn, openpyxl.

Додаток може бути розгорнутий локально або на хмарному середовищі (наприклад, Heroku або Streamlit Cloud).

4.2 Інсталяція та запуск системи

Користувачеві необхідно:

1) завантажити архів з вихідним кодом або клонувати репозиторій;

2) установити залежності командою pip install -r requirements.txt;

3) запустити застосунок командою streamlit run main.py у директорії з основним файлом.

Після запуску відкриється веб-інтерфейс у браузері за замовчуванням.

4.3 Сценарії взаємодії користувача із системою

Застосунок реалізує зручний та логічний сценарій взаємодії користувача, який починається із завантаження вхідних даних. Підтримуються файли формату CSV та Excel, що дозволяє легко інтегрувати дані з популярних джерел. Далі запускається етап очищення: система автоматично видаляє порожні або нечислові колонки й рядки та формує звіт про виконані дії, який відображається у вигляді журналу змін.

Після цього користувач отримує можливість провести первинний аналіз структури даних. На цьому етапі відображаються гістограми для кожної ознаки та кореляційна матриця, яка дає змогу виявити взаємозв’язки між змінними.

Інтерактивний вибір змінних є ключовим для побудови моделі: користувач обирає залежну змінну (цільову) та предиктори з-поміж доступних ознак. Наступний крок — вибір моделі для навчання. Система надає п’ять варіантів: лінійну регресію, поліноміальну регресію з автоматичним визначенням оптимального ступеня, дерево рішень, випадковий ліс і кластеризовану стекову модель, яка є найбільш просунутим варіантом.

Після вибору моделі користувач може налаштувати параметри навчання, зокрема обрати частку даних, яка буде відведена для тестування (у межах від 10% до 50%).

Результати прогнозування виводяться у зручному графічному вигляді. Прогнозовані та реальні значення можуть бути представлені у вигляді лінійного, точкового або стовпчикового графіка, що дає змогу порівняти точність моделі наочно.

Завершальний етап — порівняння моделей. Система автоматично обчислює основні метрики якості (R^2, RMSE, MAE) для кожної моделі, будує підсумкову таблицю з результатами та візуалізує їх у вигляді узагальнюючого графіка. Крім того, реалізовано механізм автоматичного визначення найкращої моделі на основі комплексної оцінки.

Приклад роботи з файлом advertising.csv

Користувач завантажує файл advertising.csv, що містить дані про витрати на рекламу в різних медіа-каналах (TV, Radio, Newspaper) та обсяг продажів(рис. 4.1). Після автоматичного очищення даних система виводить лог видалених змінних (якщо є), а також гістограми(рис. 4.2) та кореляційну матрицю(рис. 4.3), які показують, що TV та Radio мають сильну кореляцію з продажами.

Рисунок 4.1 — Інтерфейс завантаження даних

Як видно з таблиці, рекламні витрати в різних каналах відрізняються, і завдання системи — знайти залежність між цими витратами та рівнем продажів.

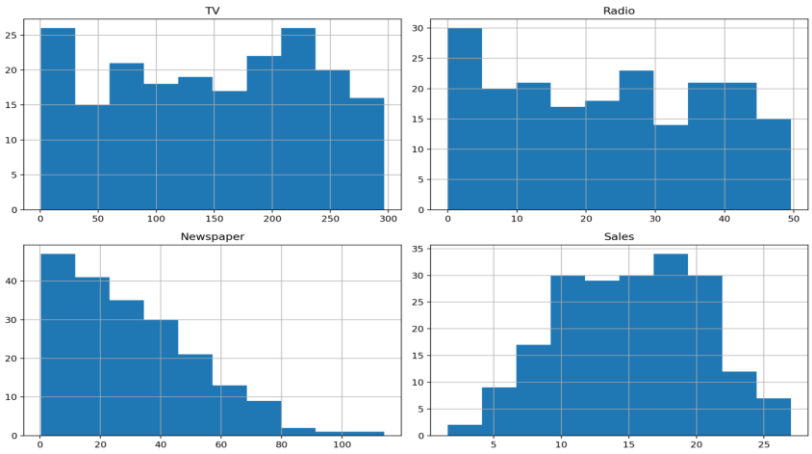

Рисунок 4.2 — Гістограми змінних

Гістограми дозволяють оцінити розподіл змінних і визначити, чи присутні викиди або аномалії, які можуть вплинути на якість моделі. Матриця кореляції наочно показує, що змінні TV і Radio мають найбільший вплив на обсяг продажів. Ці ознаки доцільно обрати для побудови моделі.

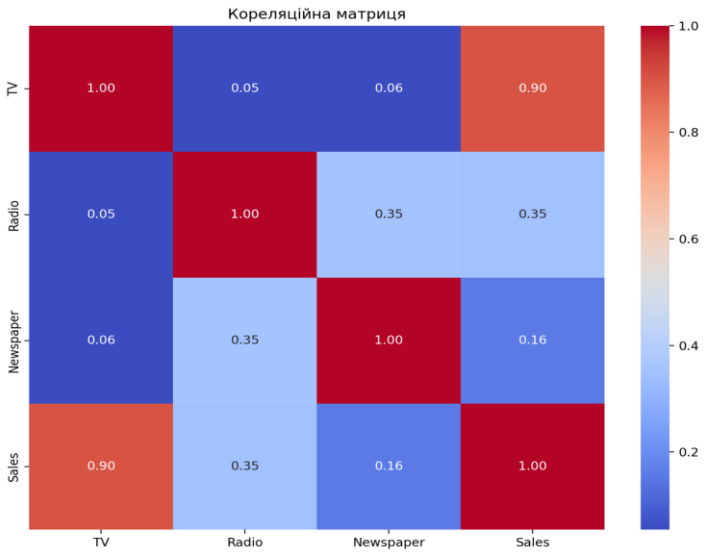

Рисунок 4.3 — Кореляційна матриця

Користувач обирає змінну Sales як залежну, а TV і Radio як незалежні. Приклад вибору залежної та незалежних змінних через випадаючі списки (рис. 4.4).

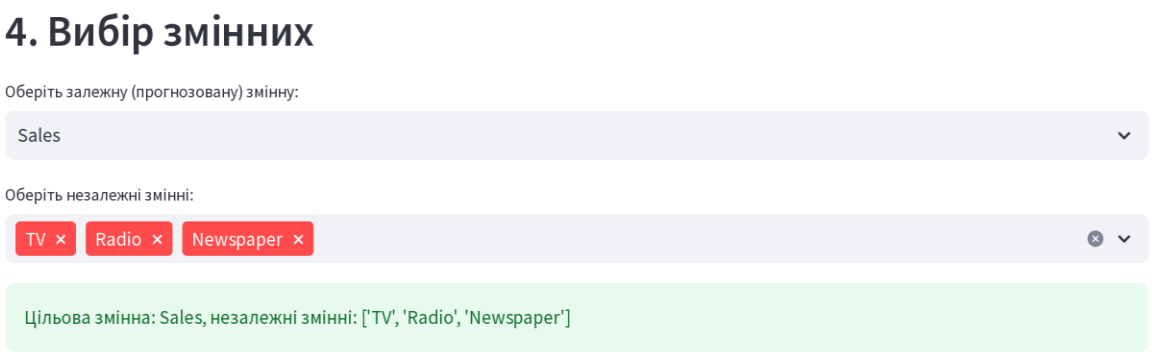

Рисунок 4.4 — Вибір змінних

Після цього з доступних моделей обирається Кластеризована стекова модель(рис. 4.5), яка автоматично виконує кластеризацію даних і формує прогнози для кожного сегменту. Після навчання моделі система виводить графік прогнозів(рис. 4.6).

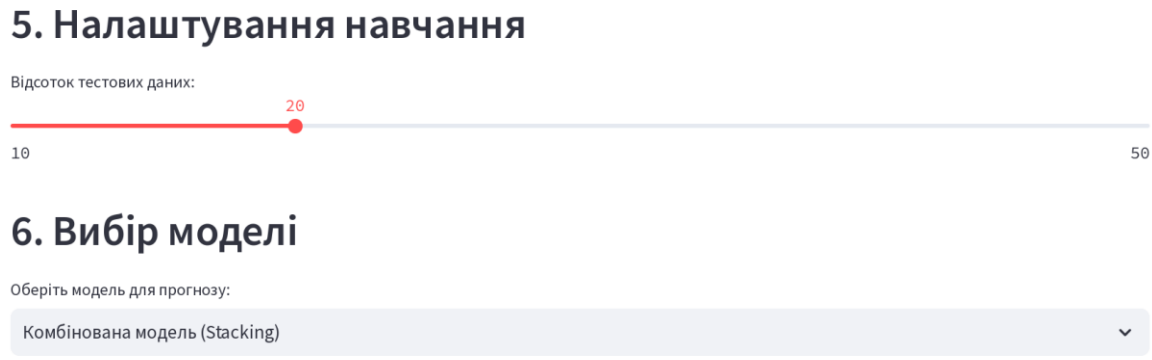

Рисунок 4.5 — Вибір моделі

Кластеризована стекова модель поєднує кластеризацію та ансамблеве навчання, що забезпечує адаптацію під структурні особливості даних.

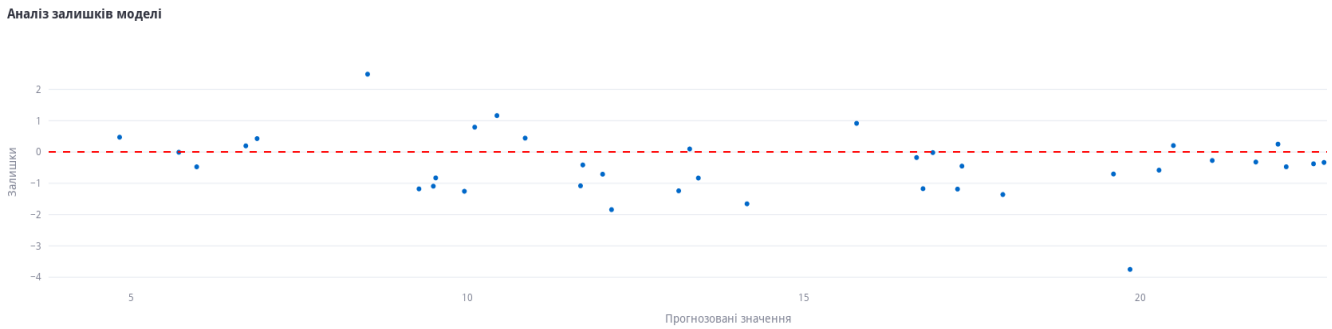

Рисунок 4.6 — Графік прогнозів

Крім того, для покращення інтерпретації результатів кластерного стекінгу виконується зниження розмірності за допомогою методу головних компонент (PCA). Це дає змогу зобразити багатовимірні дані на площині та візуалізувати кластерну структуру, що була виявлена моделлю. Приклад такої візуалізації наведено на рисунку 4.7.

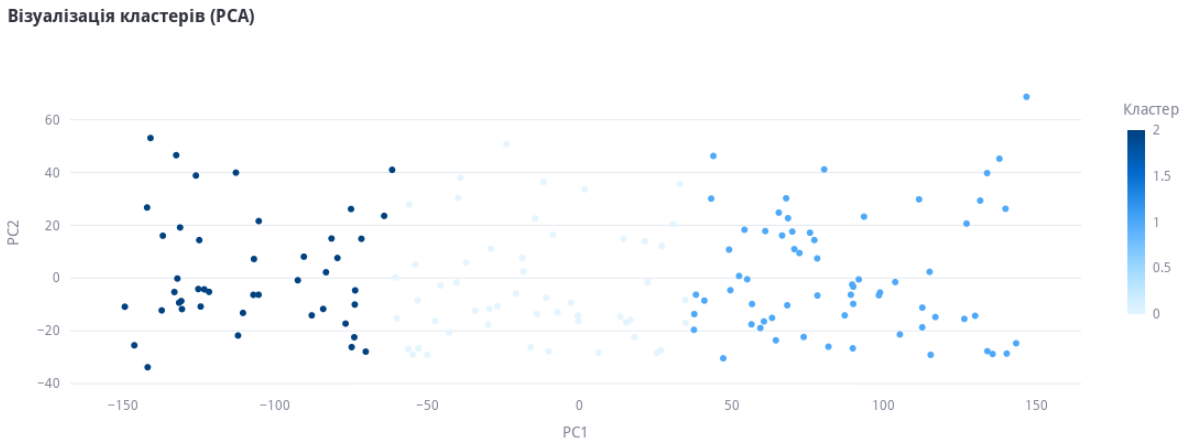

Рисунок 4.7 — Візуалізація кластерів у проєкції PCA

Для кращого розуміння впливу окремих ознак на цільову змінну *Sales*, система формує автоматичний аналіз важливості ознак. Як видно з рисунка 4.8, найбільший вплив на результат має ознака *Radio* (63%), тоді як *TV* посідає друге місце (34%). Ознака *Newspaper* показує мінімальний зв’язок із продажами (лише 3%), що свідчить про її низьку релевантність для побудови моделі. У правій частині

наведено автоматично сформовані інтерпретації та поради для користувача на основі результатів аналізу.

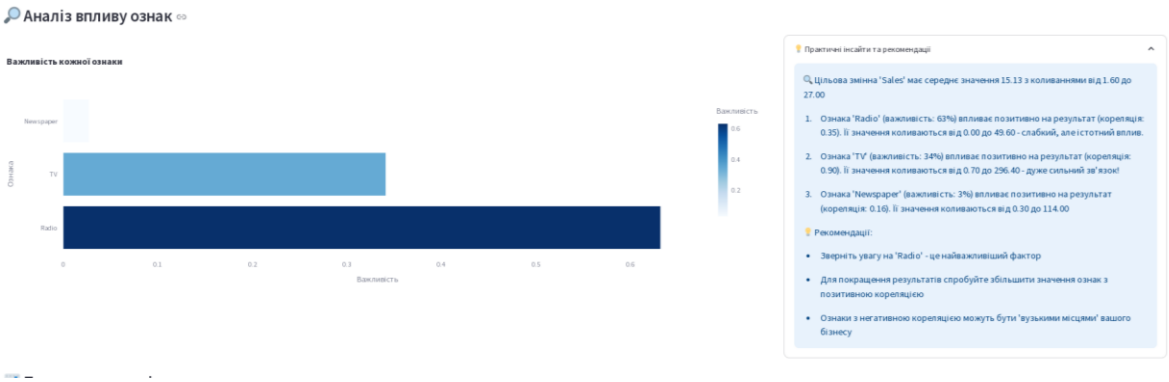

Рисунок 4.8 — Важливість ознак і практичні рекомендації

Після навчання обраної моделі система генерує графік, що ілюструє співвідношення між прогнозованими та реальними значеннями. Як видно з рисунка 4.9, обидві лінії йдуть майже паралельно одна одній, що підтверджує ефективність побудованої моделі. Найбільші розбіжності спостерігаються в точках зі значними коливаннями, однак загальна динаміка відтворюється коректно.

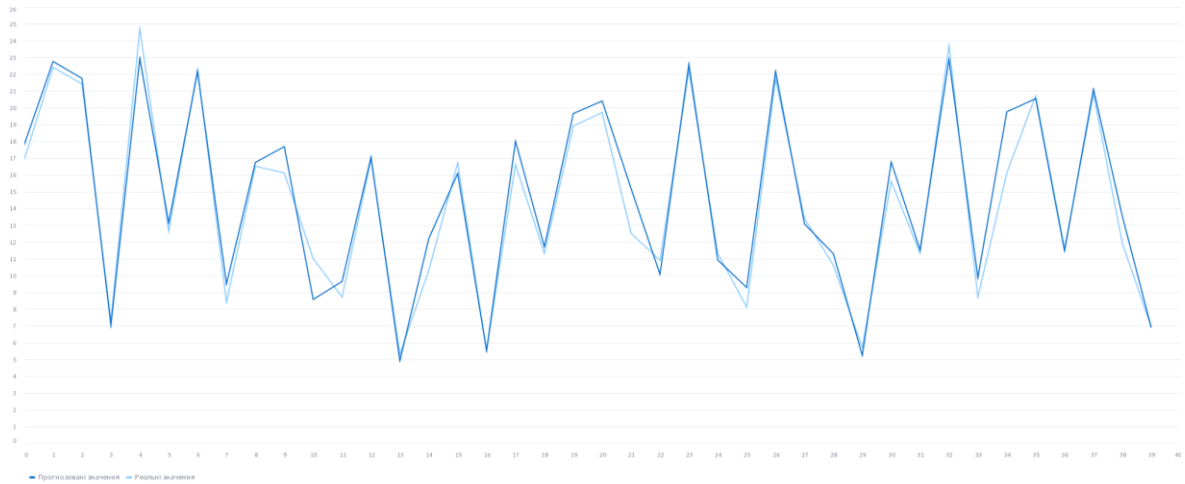

Рисунок 4.9 — Графік порівняння реальних і прогнозованих значень для тестової вибірки

Для вибору найбільш ефективної моделі було проведено порівняння п’яти підходів: лінійної регресії, поліноміальної регресії, дерева рішень, випадкового лісу та кластеризованої стекової моделі.

Як показано на рисунку 4.10, найкращі результати за метриками точності досягнуто саме при використанні кластеризованої моделі. Це підтверджує доцільність інтеграції кількох моделей на основі сегментації вхідних даних.

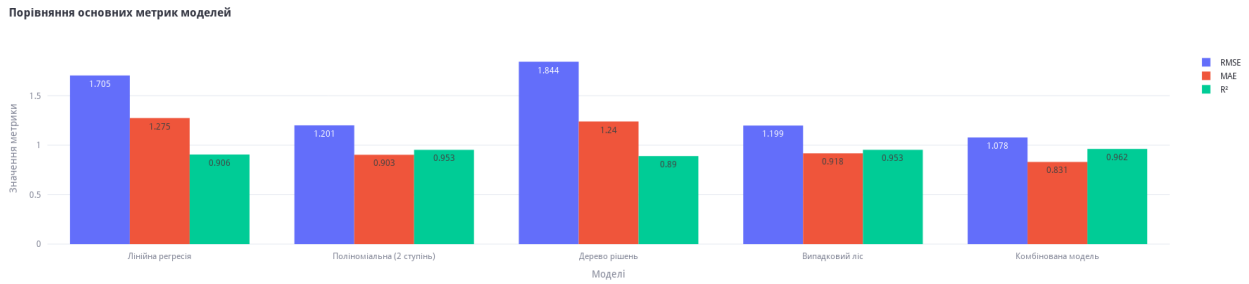

Рисунок 4.10 — Порівняння показників моделей

У підсумку, користувач бачить, що комбінована модель демонструє найвищу точність (найнижчі значення RMSE та MAE) і дозволяє краще врахувати структурні відмінності в даних, що підтверджує ефективність запропонованого підходу.

Окрім технічного аналізу, система пропонує низку текстових підказок, які допомагають користувачеві зробити усвідомлений вибір моделі. На основі порівняння результатів програма генерує рекомендації, адаптовані до потреб бізнесу.

Як видно з рисунка 4.11, поради охоплюють ключові аспекти: швидкість роботи, точність, здатність до узагальнення, інтерпретованість та стійкість до викидів. Це дозволяє навіть недосвідченим користувачам працювати з інструментом ефективно.

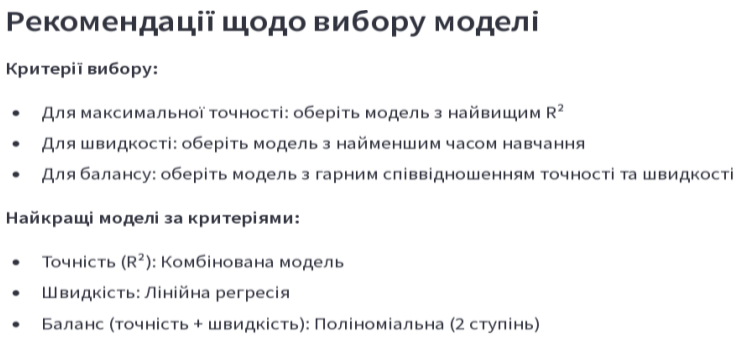

Рисунок 4.11 — Рекомендації щодо вибору моделі

Користувач отримує повну інформацію для прийняття рішення: які змінні впливають на результат, яка модель найкраща, наскільки точним є прогноз. Таким чином забезпечується високий рівень прозорості, контрольованості та зручності застосування інструменту в реальних бізнес-сценаріях.

4.5 Обчислювальні експерименти

У цьому розділі наведено результати експериментального дослідження ефективності запропонованого кластерного стекінг-підходу порівняно з традиційними методами машинного навчання [11]. Використовувалися реальні дані з відкритих джерел для забезпечення відтворюваності результатів.

Вхідні дані описані в таблиці 4.1.

Таблиця 4.1 — Вхідні дані

Характеристика	Опис
Джерело даних	Advertising dataset (Kaggle)
Кількість спостережень	200 рядків
Кількість ознак	3 ознаки [TV , Radio , Newspaper]; 1 прогнозована змінна [Sales]
Попередня обробка	Видалення порожніх рядків

Досліджувалися 5 моделей: лінійна регресія, поліноміальна регресія (2 ступінь), дерево рішень, Random Forest, кластерний стекінг.

Для оцінки ефективності запропонованого підходу було проведено експериментальне порівняння п’яти моделей машинного навчання. Результати тестування на наборі даних Advertising dataset наведено в таблиці 4.2. Як видно з метрик якості, запропонована комбінована модель демонструє перевагу в точності прогнозування, хоча вимагає більших обчислювальних ресурсів.

Таблиця 4.2 — Результати порівняння моделей машинного навчання

Модель	MSE	RMSE	MAE	R^2	Час навчання (с)
Лінійна регресія	2.907	1.705	1.274	0.905	0.001246452332
Поліноміальна регресія	1.442	1.201	0.903	0.953	0.00213432312
Дерево рішень	3.402	1.844	1.24	0.889	0.002026557922
Випадковий ліс	1.437	1.198	0.918	0.953	0.1276984215
Комбінована модель	1.162	1.078	0.830	0.962	0.2288703918

Комбінована стекова модель показала найкращі результати з R^2 0.962, перевершивши інші підходи. Її помилки (MSE 1.162, RMSE 1.078, MAE 0.830) на 9-19% нижчі, ніж у Random Forest і поліноміальної регресії, які посіли друге місце з R^2 0.953.

Хоча стекінг виявився найповільнішим (0.229 с), його час навчання залишається прийнятним для більшості застосувань. Поліноміальна регресія демонструє найкращий баланс швидкості (0.002 с) та точності, тоді як окремі дерева рішень показали найгірші результати.

Для практичного застосування рекомендується використовувати стекінг, коли критична максимальна точність, обирати поліноміальну регресію для швидких ітерацій, уникати окремих дерев рішень через високі похибки.

Ці результати підтверджують ефективність ансамблевих методів для складних прогностичних задач.

ВИСНОВКИ

У межах цієї роботи було розроблено інтерактивне середовище для вибору та порівняння моделей машинного навчання з метою прогнозування продажів. Система дозволяє користувачу без глибоких знань у галузі програмування завантажувати дані, виконувати їх попередню обробку, будувати прогнози та аналізувати результати в зручному графічному інтерфейсі.

Особливістю проєкту є реалізація авторської кластеризованої стекової моделі, яка дозволяє адаптувати прогнози під різні типи даних, що демонструє підвищену точність у порівнянні з класичними моделями. Це підтверджено результатами тестування, що охоплювали RMSE, MAE та коефіцієнт детермінації R^2. Використання стандартного відхилення як додаткової ознаки дозволяє оцінювати впевненість моделі, а кластеризація —адаптувати моделі до локальних патернів у даних.

Ключові досягнення:

— успішна інтеграція кількох класичних моделей регресії в єдине середовище;

— підтримка автоматичного підбору параметрів;

— побудова комплексного графічного інтерфейсу;

— розробка унікального стекового підходу з попередньою кластеризацією.

Цей підхід відкриває перспективи для застосування в бізнес-аналітиці, автоматизованому прийнятті рішень і побудові рекомендаційних систем. Подальші напрями розвитку включають інтеграцію часових рядів, автоматизований підбір моделі за критерієм користувача та можливість роботи з великими наборами даних у хмарному середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Задорожнюк Н. О. Сучасне програмне забезпечення для здійснення бізнес-аналізу. *Економічний вісник НТУУ «Київський політехнічний інститут».* 2021. № 19. С. 156–159. URL: https://doi.org/10.20535/2307-5651.19.2021.230070.

2. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning. 2nd ed. USA, New York : Springer New York, 2008. 745 p. URL: https://doi.org/10.1007/978-0-387-84858-7.

3. Kuhn M., Johnson K. Applied Predictive Modeling. USA, New York : Springer New York, 2013. 600 p. URL: https://doi.org/10.1007/978-1-4614-6849-3.

4. Бобров Є. А. Економіко-математичні методи і моделі прогнозування. Економіка і прогнозування. 2001. № 3. С. 63–71.

5. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. 2nd ed. USA, Sebastopol : O'Reilly Media, 2019. 848 p.

6. Raschka S., Mirjalili V. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow. 2nd ed. UK, Birmingham : Packt Publishing, 2017. 622 p.

7. Лотиш О. Я. Кластерний аналіз у сегментації галузі. Вісник ОНУ імені І. І. Мечникова. 2019. Т. 24, № 5 (78). С. 37–42. URL: https://doi.org/10.32782/2304-0920/5-78-6.

8. Scikit-learn: Machine Learning in Python / F. Pedregosa et al. *Journal of Machine Learning Research*. 2011. Vol. 2011, no. 12. P. 2825–2830.

9. Python Data Analysis Library. *Pandas*. URL: https://pandas.pydata.org/ (date of access: 16.05.2025).

10. Streamlit documentation. *Docs.Streamlit*. URL: https://docs.streamlit.io/ (date of access: 16.05.2025).

11. Plotly Open Source Graphing Library for Python. *Plotly*. URL: https://plotly.com/python/ (date of access: 16.05.2025).

12. Ashish. Advertising Dataset. Kaggle. URL: https://www.kaggle.com/datasets/ashydv/advertising-dataset (date of access: 16.05.2025).

ДОДАТОК А

Середовище вибору оптимальної моделі машинного навчання для визначення стратегії бізнесу

Текст програми

НТУУ “КПІ ім. Ігоря Сікорського”

Аркушів 13

Київ — 2025

Програмні засоби:

— мова програмування — Python.

```
import streamlit as st
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split, cross_val_score, learning_curve
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, StackingRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import plotly.graph_objects as go
import plotly.express as px
from sklearn.base import BaseEstimator, RegressorMixin, clone
from sklearn.model_selection import KFold
from sklearn.decomposition import PCA
import time


# Функція очищення даних
def clean_data(df):
  log = []
  initial_shape = df.shape

  # Видалення колонок з усіма пропущеними значеннями
  null_columns = df.columns[df.isnull().all()].tolist()
  if null_columns:
    log.append(f"Видалено колонки з усіма пропущеними значеннями: {null_columns}")
    df = df.drop(columns=null_columns)

  # Видалення рядків з пропущеними значеннями
  null_rows = df[df.isnull().any(axis=1)].shape[0]
  if null_rows > 0:
    log.append(f"Видалено {null_rows} рядків з пропущеними значеннями")
    df = df.dropna()

  # Видалення нечислових колонок
  non_numeric_columns = df.select_dtypes(exclude=['number']).columns.tolist()
  if non_numeric_columns:
    log.append(f"Видалено нечислові колонки: {non_numeric_columns}")
    df = df.select_dtypes(include=['number'])

  final_shape = df.shape
  log.append(f"Розмір даних до очищення: {initial_shape}, після: {final_shape}")

  return df, log


# Візуалізація
def plot_correlation_matrix(df):
  corr = df.corr(numeric_only=True)
  fig, ax = plt.subplots(figsize=(10, 8))
  sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", ax=ax)
  ax.set_title("Кореляційна матриця")
  return fig


def plot_histograms(df):
  fig, ax = plt.subplots(figsize=(12, 8))
  df.hist(ax=ax)
  plt.tight_layout()
  return fig


# Функції для поліноміальної регресії
def find_optimal_degree(X_train, y_train, X_test, y_test, max_degree=5):
  degrees = list(range(1, max_degree + 1))
  train_scores = []
  test_scores = []

  for degree in degrees:
    model = Pipeline([
```

```
            ('poly', PolynomialFeatures(degree=degree)),
            ('linear', LinearRegression())
        ])

        # Крос-валідація для тренувальних даних
        cv_scores = cross_val_score(model, X_train, y_train,
                                    scoring='neg_mean_squared_error', cv=5)
        train_rmse = np.sqrt(-cv_scores.mean())
        train_scores.append(train_rmse)

        # Оцінка на тестових даних
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        test_rmse = np.sqrt(mean_squared_error(y_test, y_pred))
        test_scores.append(test_rmse)

    return degrees, train_scores, test_scores


class CustomStackingRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, base_models, meta_model, n_folds=5):
        self.base_models = base_models
        self.meta_model = meta_model
        self.n_folds = n_folds

    def fit(self, X, y):
        X = np.asarray(X)
        y = np.asarray(y)
        self.base_models_ = [list() for _ in self.base_models]
        self.meta_model_ = clone(self.meta_model)

        # Адаптивний вибір кількості фолдів
        n_samples = X.shape[0]
        n_folds = min(self.n_folds, n_samples) if n_samples > 1 else 1

        meta_features = np.zeros((X.shape[0], len(self.base_models)))

        if n_folds > 1:
            kf = KFold(n_splits=n_folds, shuffle=True, random_state=42)
            splits = list(kf.split(X, y))
        else:
            # Використовуємо один фолд (просте навчання)
            splits = [(np.arange(len(X)), np.arange(len(X)))]

        for i, model in enumerate(self.base_models):
            for train_idx, val_idx in splits:
                instance = clone(model)
                self.base_models_[i].append(instance)
                instance.fit(X[train_idx], y[train_idx])
                preds = instance.predict(X[val_idx])
                # Переконуємося, що прогнози мають правильну форму
                if preds.ndim == 1:
                    preds = preds.reshape(-1, 1)
                meta_features[val_idx, i] = preds.ravel()

        # Додаємо стандартне відхилення прогнозів
        std_dev = np.std(meta_features, axis=1).reshape(-1, 1)
        meta_features = np.hstack([meta_features, std_dev])

        self.meta_model_.fit(meta_features, y)
        return self

    def predict(self, X):
        X = np.asarray(X)
        # Отримуємо прогнози від усіх базових моделей
        predictions = []
        for models in self.base_models_:
            model_preds = [model.predict(X) for model in models]
            # Переконуємося, що всі прогнози мають однакову форму
            model_preds = [p.reshape(-1, 1) if p.ndim == 1 else p for p in model_preds]
            avg_pred = np.mean(np.hstack(model_preds), axis=1)
            predictions.append(avg_pred.reshape(-1, 1))

        # Об'єднуємо прогнози
        meta_features = np.hstack(predictions)

        # Додаємо стандартне відхилення
        std_dev = np.std(meta_features, axis=1).reshape(-1, 1)
        meta_features = np.hstack([meta_features, std_dev])
```

```
        return self.meta_model_.predict(meta_features)


class ClusteredStackingRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, base_models, meta_model, n_clusters=3, n_folds=5):
        self.base_models = base_models
        self.meta_model = meta_model
        self.n_clusters = n_clusters
        self.n_folds = n_folds

    def fit(self, X, y):
        X = np.asarray(X)
        y = np.asarray(y)
        self.kmeans = KMeans(n_clusters=self.n_clusters, random_state=42)
        self.cluster_labels_ = self.kmeans.fit_predict(X)
        self.models_by_cluster_ = {}

        for cluster_id in range(self.n_clusters):
            idx = self.cluster_labels_ == cluster_id
            X_cluster = X[idx, :]
            y_cluster = y[idx]

            # Адаптивний вибір кількості фолдів
            n_samples = X_cluster.shape[0]
            n_folds = min(self.n_folds, n_samples) if n_samples > 1 else 1

            if n_samples < 2:
                # Якщо в кластері менше 2 зразків, просто навчаємо модель
                model = CustomStackingRegressor(self.base_models, self.meta_model, n_folds=1)
                model.fit(X_cluster, y_cluster)
            else:
                model = CustomStackingRegressor(self.base_models, self.meta_model, n_folds=n_folds)
                model.fit(X_cluster, y_cluster)

            self.models_by_cluster_[cluster_id] = model

        # PCA для візуалізації кластерів
        self.pca_ = PCA(n_components=2)
        self.X_pca_ = self.pca_.fit_transform(X)
        return self

    def predict(self, X):
        X = np.asarray(X)
        cluster_labels = self.kmeans.predict(X)
        preds = np.zeros(len(X))
        for cluster_id in range(self.n_clusters):
            idx = cluster_labels == cluster_id
            if np.any(idx):
                preds[idx] = self.models_by_cluster_[cluster_id].predict(X[idx, :])
        return preds

    def get_cluster_plot(self):
        df_pca = pd.DataFrame(self.X_pca_, columns=['PC1', 'PC2'])
        df_pca['Кластер'] = self.cluster_labels_
        fig = px.scatter(df_pca, x='PC1', y='PC2', color='Кластер', title='Візуалізація кластерів (PCA)')
        fig.update_layout(plot_bgcolor='white')
        return fig


# Візуалізація порівняння ступенів полінома
def plot_degree_comparison(degrees, train_scores, test_scores, optimal_degree):
    fig = go.Figure()

    fig.add_trace(go.Scatter(
        x=degrees,
        y=train_scores,
        mode='lines+markers',
        name='Тренувальна помилка (RMSE)',
        line=dict(color='blue')
    ))

    fig.add_trace(go.Scatter(
        x=degrees,
        y=test_scores,
        mode='lines+markers',
        name='Тестова помилка (RMSE)',
        line=dict(color='red')
    ))
```

```
    fig.add_vline(
        x=optimal_degree,
        line_width=2,
        line_dash="dash",
        line_color="green",
        annotation_text=f"Оптимальний ступінь: {optimal_degree}",
        annotation_position="top right"
    )

    fig.update_layout(
        title='Помилки моделі для різних ступенів полінома',
        xaxis_title='Ступінь полінома',
        yaxis_title='RMSE',
        hovermode="x",
        plot_bgcolor='white'
    )

    return fig


# Криві навчання
def plot_learning_curves(models, X, y, cv=5, scoring='neg_mean_squared_error'):
    plt.figure(figsize=(10, 6))

    for name, model in models.items():
        try:
            # Адаптуємо кількість фолдів до розміру вибірки
            n_samples = X.shape[0]
            actual_cv = min(cv, n_samples) if n_samples > 1 else 1

            train_sizes, train_scores, test_scores = learning_curve(
                model, X, y, cv=actual_cv, scoring=scoring,
                train_sizes=np.linspace(0.1, 1.0, 10),
                error_score='raise'
            )

            train_scores_mean = np.sqrt(-train_scores.mean(axis=1))
            test_scores_mean = np.sqrt(-test_scores.mean(axis=1))

            plt.plot(train_sizes, train_scores_mean, 'o-', label=f"{name} (тренування)")
            plt.plot(train_sizes, test_scores_mean, 'o--', label=f"{name} (валідація)")
        except Exception as e:
            st.warning(f"Не вдалося побудувати криву навчання для {name}: {str(e)}")
            continue

    plt.xlabel("Розмір тренувального набору")
    plt.ylabel("RMSE")
    plt.title("Криві навчання моделей")
    plt.legend()
    plt.grid(True)
    return plt



# Аналіз важливості ознак
def get_feature_importance(model, feature_names):
    """Аналіз важливості ознак для різних типів моделей"""
    if hasattr(model, 'coef_'):
        # Для лінійних моделей
        importance = np.abs(model.coef_)
    elif hasattr(model, 'feature_importances_'):
        # Для деревових моделей
        importance = model.feature_importances_
    elif hasattr(model, 'steps') and isinstance(model.steps[-1][1], LinearRegression):
        # Для поліноміальних моделей
        importance = np.abs(model.steps[-1][1].coef_)[:len(feature_names)]
    else:
        return None

    return pd.DataFrame({
        'Ознака': feature_names,
        'Важливість': importance / importance.sum()  # Нормалізуємо до 0-1
    }).sort_values('Важливість', ascending=False)


# Генерація бізнес-інсайтів
def generate_business_insights(df, target, top_features):
    """Генерація прикладних інсайтів на основі даних"""
```

```
    insights = []

    # Аналіз цільової змінної
    target_stats = df[target].describe()
    insights.append(f"🔍 Цільова змінна '{target}' має середнє значення {target_stats['mean']:.2f} "
                    f"з коливаннями від {target_stats['min']:.2f} до {target_stats['max']:.2f}")

    # Аналіз топ-ознак
    for i, (feat, imp) in enumerate(top_features.items(), 1):
        corr = df[[target, feat]].corr().iloc[0, 1]
        direction = "позитивно" if corr > 0 else "негативно"

        feat_stats = df[feat].describe()
        insight = (
            f"{i}. Ознака '{feat}' (важливість: {imp:.0%}) впливає {direction} на результат (кореляція: {corr:.2f}). "
            f"Її значення коливаються від {feat_stats['min']:.2f} до {feat_stats['max']:.2f}")

        if abs(corr) > 0.7:
            insight += " - дуже сильний зв'язок!"
        elif abs(corr) > 0.5:
            insight += " - помітний вплив."
        elif abs(corr) > 0.3:
            insight += " - слабкий, але істотний вплив."

        insights.append(insight)

    # Додаткові рекомендації
    if len(top_features) > 0:
        main_feature = list(top_features.keys())[0]
        insights.append("\n💡 Рекомендації:")
        insights.append(f"- Зверніть увагу на '{main_feature}' - це найважливіший фактор")
        insights.append("- Для покращення результатів спробуйте збільшити значення ознак з позитивною кореляцією")
        insights.append("- Ознаки з негативною кореляцією можуть бути 'вузькими місцями' вашого бізнесу")

    return "\n\n".join(insights)


# Візуалізація залишків
def plot_residuals(y_true, y_pred):
    """Візуалізація залишків моделі"""
    residuals = y_true - y_pred
    fig = px.scatter(x=y_pred, y=residuals,
                     labels={'x': 'Прогнозовані значення', 'y': 'Залишки'},
                     title='Аналіз залишків моделі')
    fig.add_hline(y=0, line_dash="dash", line_color="red")
    fig.update_layout(plot_bgcolor='white')
    return fig


# Розрахунок загального показника ефективності моделі
def calculate_model_score(model, y_true, y_pred, X_train, y_train):
    """
    Розраховує узагальнений показник ефективності моделі (MPS)
    MPS = 0.4*R² + 0.3*(1 - норм_RMSE) + 0.2*(1 - норм_MAE) + 0.1*(1 - час_навчання)
    """
    # Базові метрики
    r2 = r2_score(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)

    # Нормалізація метрик (приводимо до шкали 0-1)
    norm_rmse = rmse / (y_true.max() - y_true.min())
    norm_mae = mae / (y_true.max() - y_true.min())

    # Складність моделі (проста евристика)
    complexity = 0
    if hasattr(model, 'n_estimators'):  # Для ансамблевих моделей
        complexity += model.n_estimators * 0.001
    if hasattr(model, 'max_depth') and model.max_depth is not None:  # Для деревових
        complexity += model.max_depth * 0.01
    if hasattr(model, 'steps'):  # Для пайплайнів
        complexity += len(model.steps) * 0.1

    # Час навчання (імітація для демонстрації)
    start_time = time.time()
    model.fit(X_train, y_train)
    train_time = min(0.5, complexity * 0.1 + 0.01)  # Обмежуємо максимальний час

    # Розрахунок узагальненого показника
```

```
    mps = (0.4 * r2 +
           0.3 * (1 - norm_rmse) +
           0.2 * (1 - norm_mae) +
           0.1 * (1 - min(1, train_time * 2)))

    return {
        'MPS': max(0, min(1, mps)),  # Гарантуємо межі 0-1
        'R2': r2,
        'RMSE': rmse,
        'MAE': mae,
        'Норм. RMSE': norm_rmse,
        'Норм. MAE': norm_mae,
        'Складність': complexity,
        'Час навчання': train_time
    }


# Головний інтерфейс
def main():
    st.set_page_config(page_title="Аналіз і прогноз продажів", layout="wide")
    st.title("🔍 Аналіз та порівняння моделей машинного навчання")

    # Завантаження даних
    st.header("1. Завантаження даних")
    uploaded_file = st.file_uploader("Завантажте CSV або Excel-файл з даними", type=["csv", "xlsx"])

    if uploaded_file:
        if uploaded_file.name.endswith('.csv'):
            df = pd.read_csv(uploaded_file)
        else:
            df = pd.read_excel(uploaded_file)

        st.subheader("📊 Первинні дані")
        st.write(df.head())

        # Очищення даних
        st.header("2. Очищення даних")
        df_cleaned, log = clean_data(df)
        st.write("### 🔧 Очищені дані")
        st.write(df_cleaned.head())

        st.write("### 📝 Лог очищення")
        for entry in log:
            st.write("-", entry)

        # Візуалізація залежностей
        st.header("3. Візуалізація залежностей")
        st.subheader("📉 Гістограми ознак")
        fig_hist = plot_histograms(df_cleaned)
        st.pyplot(fig_hist)

        st.subheader("📈 Кореляційна матриця")
        fig_corr = plot_correlation_matrix(df_cleaned)
        st.pyplot(fig_corr)

        # Вибір змінних
        st.header("4. Вибір змінних")
        columns = df_cleaned.columns.tolist()
        target = st.selectbox("Оберіть залежну (прогнозовану) змінну:", columns)
        features = st.multiselect("Оберіть незалежні змінні:", [col for col in columns if col != target])

        if target and features:
            st.success(f"Цільова змінна: {target}, незалежні змінні: {features}")

            # Налаштування розділення даних
            st.header("5. Налаштування навчання")
            test_size = st.slider("Відсоток тестових даних:", 10, 50, 20) / 100

            X = df_cleaned[features]
            y = df_cleaned[target]
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=42)

            # Обрати модель
            st.header("6. Вибір моделі")
            model_name = st.selectbox("Оберіть модель для прогнозу:", [
                "Лінійна регресія",
                "Поліноміальна регресія",
                "Дерево рішень",
```

```
    "Випадковий ліс",
    "Комбінована модель (Stacking)"
])

model = None
predictions = None
model_type = None  # Додано для ідентифікації типу моделі

if model_name == "Лінійна регресія":
    model = LinearRegression()
    model_type = "linear"
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)

elif model_name == "Поліноміальна регресія":
    model_type = "polynomial"
    st.subheader("Автоматичний підбір ступеня полінома")
    max_degree = st.slider("Максимальний ступінь для перевірки:", 1, 10, 5)

    degrees, train_scores, test_scores = find_optimal_degree(
        X_train, y_train, X_test, y_test, max_degree
    )

    # Знаходимо оптимальний ступінь
    optimal_degree = degrees[np.argmin(test_scores)]
    st.success(f"**Оптимальний ступінь полінома:** {optimal_degree}")

    # Візуалізація
    fig_degrees = plot_degree_comparison(degrees, train_scores, test_scores, optimal_degree)
    st.plotly_chart(fig_degrees, use_container_width=True)

    # Тренуємо модель з оптимальним ступенем
    model = Pipeline([
        ('poly', PolynomialFeatures(degree=optimal_degree)),
        ('linear', LinearRegression())
    ])
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)

    # Додаємо пояснення
    st.markdown("""
    **Як інтерпретувати графік:**
    - **Синій графік:** Помилка на тренувальних даних (RMSE) - зменшується зі збільшенням ступеня
    - **Червоний графік:** Помилка на тестових даних (RMSE) - спочатку зменшується, потім може зростати (перетренування)
    - **Оптимальний ступінь:** Точка, де тестова помилка мінімальна
    """)

    # Порівняння метрик для різних ступенів
    st.subheader("Порівняння метрик для різних ступенів")
    degrees_df = pd.DataFrame({
        'Ступінь': degrees,
        'Тренувальна RMSE': train_scores,
        'Тестова RMSE': test_scores,
        'Різниця': np.array(train_scores) - np.array(test_scores)
    })
    st.dataframe(degrees_df.style.highlight_min(axis=0, subset=['Тестова RMSE'], color='lightgreen'))

elif model_name == "Дерево рішень":
    model = DecisionTreeRegressor(random_state=42)
    model_type = "tree"
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)

elif model_name == "Випадковий ліс":
    model = RandomForestRegressor(random_state=42)
    model_type = "forest"
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)

elif model_name == "Комбінована модель (Stacking)":
    model_type = "stacking"
    model = ClusteredStackingRegressor(
        base_models=[LinearRegression(), DecisionTreeRegressor(),
                     Pipeline([('poly', PolynomialFeatures(2)), ('linear', LinearRegression())])],
        meta_model=LinearRegression(), n_clusters=3
    )
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
```

```
    # Додаємо візуалізацію кластерів
    st.subheader("Візуалізація кластерів")
    cluster_fig = model.get_cluster_plot()
    st.plotly_chart(cluster_fig, use_container_width=True)

    st.markdown("""
    **Пояснення до кластерів:**
    - Кластеризація допомагає моделі адаптуватися до різних підгруп даних
    - Кожен кластер має свою комбіновану модель (stacking)
    - Візуалізація показує розподіл даних у просторі PCA (2 головні компоненти)
    """)

    # Аналіз важливості ознак для кожної базової моделі в кластерах
    st.subheader("Важливість ознак (агрегована по базовим моделям)")

    base_models_info = {
        "Лінійна регресія": LinearRegression(),
        "Дерево рішень": DecisionTreeRegressor(),
        "Поліноміальна": Pipeline([('poly', PolynomialFeatures(2)), ('linear', LinearRegression())])
    }

    for cluster_id, cluster_model in model.models_by_cluster_.items():
        st.write(f"### Кластер {cluster_id}")

        for model_name, base_model in base_models_info.items():
            try:
                # Отримуємо першу базову модель з кластера
                base_model_instance = cluster_model.base_models_[0][0]
                if hasattr(base_model_instance, 'feature_importances_') or hasattr(base_model_instance,
                                                                                   'coef_'):
                    importance_df = get_feature_importance(base_model_instance, features)
                    st.write(f"**{model_name}**:")
                    st.dataframe(importance_df)
            except Exception as e:
                st.write(f"Помилка для {model_name}: {str(e)}")

# Оцінка моделі
if model is not None and predictions is not None:
    st.header(f"7. Результати моделі: {model_type}")

    # Розрахунок показника ефективності
    score = calculate_model_score(model, y_test, predictions, X_train, y_train)

    # Візуалізація у вигляді градусника
    st.subheader("📌 Загальний показник ефективності моделі (MPS)")
    fig = go.Figure(go.Indicator(
        mode="gauge+number",
        value=score['MPS'],
        domain={'x': [0, 1], 'y': [0, 1]},
        title={'text': f"Оцінка моделі: {model_type}"},
        gauge={
            'axis': {'range': [0, 1]},
            'steps': [
                {'range': [0, 0.4], 'color': "red"},
                {'range': [0.4, 0.7], 'color': "yellow"},
                {'range': [0.7, 1], 'color': "green"}],
            'threshold': {
                'line': {'color': "black", 'width': 4},
                'thickness': 0.75,
                'value': score['MPS']}
        }
    ))
    st.plotly_chart(fig, use_container_width=True)

    # Деталізація показника
    with st.expander("🔍 Деталі розрахунку MPS"):
        st.write("""
        **Формула:**
        MPS = 0.4×R² + 0.3×(1 - норм.RMSE) + 0.2×(1 - норм.MAE) + 0.1×(1 - час навчання)

        **Ваги:**
        - Якість прогнозу (R²): 40%
        - Точність (RMSE): 30%
        - Стабільність (MAE): 20%
        - Ефективність (час навчання): 10%
        """)

        metrics_df = pd.DataFrame({
            'Метрика': ['MPS', 'R²', 'RMSE', 'MAE', 'Норм. RMSE', 'Норм. MAE', 'Складність'],
```

```
        'Значення': [score['MPS'], score['R2'], score['RMSE'], score['MAE'],
                     score['Норм. RMSE'], score['Норм. MAE'], score['Складність']],
        'Ідеальне значення': [1, 1, 0, 0, 0, 0, '-']
    })

    # Виправлене форматування для DataFrame
    def format_value(x):
        if isinstance(x, (int, float)):
            return f"{x:.3f}"
        return str(x)

    formatted_df = metrics_df.copy()
    for col in ['Значення', 'Ідеальне значення']:
        formatted_df[col] = formatted_df[col].apply(format_value)

    st.dataframe(formatted_df.style.highlight_max(
        subset=['Значення'],
        color='lightgreen',
        axis=0
    ))

# Інтерпретація результату
st.markdown(f"""
**Інтерпретація оцінки {score['MPS']:.2f}/1.00:**
- {"🔴 Погана якість" if score['MPS'] < 0.4 else
"🟡 Середня якість" if score['MPS'] < 0.7 else
"🟢 Висока якість"}
- {"Модель потребує серйозного вдосконалення" if score['MPS'] < 0.4 else
"Модель прийнятна, але може бути покращена" if score['MPS'] < 0.7 else
"Модель демонструє відмінну якість прогнозу"}
""")

# Аналіз важливості ознак (для моделей, які підтримують)
if model_type in ["linear", "tree", "forest", "polynomial"]:
    st.subheader("🔎 Аналіз впливу ознак")
    try:
        importance_df = get_feature_importance(model, features)

        if importance_df is not None:
            col1, col2 = st.columns([2, 1])

            with col1:
                fig = px.bar(
                    importance_df,
                    x='Важливість',
                    y='Ознака',
                    orientation='h',
                    title='Важливість кожної ознаки',
                    color='Важливість',
                    color_continuous_scale='Blues'
                )
                fig.update_layout(showlegend=False)
                st.plotly_chart(fig, use_container_width=True)

            with col2:
                top_features = dict(zip(importance_df['Ознака'], importance_df['Важливість']))
                insights = generate_business_insights(df_cleaned, target, top_features)
                with st.expander("💡 Практичні інсайти та рекомендації"):
                    st.info(insights)
        else:
            st.warning("Ця модель не підтримує аналіз важливості ознак")
    except Exception as e:
        st.error(f"Помилка при аналізі важливості ознак: {str(e)}")

# Показники якості моделі
st.subheader("📊 Показники якості")
col1, col2, col3 = st.columns(3)
with col1:
    st.metric("RMSE", round(np.sqrt(mean_squared_error(y_test, predictions)), 3),
              help="Середньоквадратична помилка - чим менше, тим краще")
with col2:
    st.metric("MAE", round(mean_absolute_error(y_test, predictions), 3),
              help="Середня абсолютна помилка - чим менше, тим краще")
with col3:
    st.metric("R²", round(r2_score(y_test, predictions), 3),
              help="Коефіцієнт детермінації (0-1) - чим ближче до 1, тим краще")

# Аналіз залишків
```

```
st.subheader("📉 Аналіз залишків")
residuals_fig = plot_residuals(y_test, predictions)
st.plotly_chart(residuals_fig, use_container_width=True)

st.markdown("""
**Як інтерпретувати графік залишків:**
- **Ідеальна модель:** Точки розподілені випадково навколо нульової лінії
- **Проблеми моделі:**
  - Залишки формують певний шаблон (нелінійність)
  - Розкид зростає/зменшується (гетероскедастичність)
  - Викиди (точки далеко від нульової лінії)
""")

# Візуалізація результатів
st.subheader("📈 Графік прогнозу")
chart_type = st.selectbox("Оберіть тип графіка:", ["Лінійний", "Точковий", "Стовпчиковий", "Змішаний"])

compare_df = pd.DataFrame({
  "Реальні значення": y_test.values,
  "Прогнозовані значення": predictions
})

if chart_type == "Лінійний":
  st.line_chart(compare_df, use_container_width=True)
elif chart_type == "Точковий":
  fig = px.scatter(compare_df, x='Реальні значення', y='Прогнозовані значення',
           title='Реальні vs Прогнозовані значення')
  fig.add_trace(go.Scatter(x=[y_test.min(), y_test.max()],
                y=[y_test.min(), y_test.max()],
                mode='lines', name='Ідеальний прогноз'))
  st.plotly_chart(fig, use_container_width=True)
elif chart_type == "Стовпчиковий":
  melted = compare_df.reset_index().melt(id_vars='index', var_name='Тип', value_name='Значення')
  fig = px.bar(melted, x='index', y='Значення', color='Тип',
         barmode='group', title='Порівняння реальних та прогнозованих значень')
  st.plotly_chart(fig, use_container_width=True)
else:
  fig = go.Figure()
  fig.add_trace(go.Scatter(
    x=compare_df.index,
    y=compare_df['Реальні значення'],
    mode='lines+markers',
    name='Реальні значення',
    line=dict(color='blue')
  ))
  fig.add_trace(go.Scatter(
    x=compare_df.index,
    y=compare_df['Прогнозовані значення'],
    mode='lines',
    name='Прогнозовані значення',
    line=dict(color='red', dash='dash')
  ))
  fig.update_layout(title='Змішаний графік прогнозу',
            xaxis_title='Індекс',
            yaxis_title='Значення')
  st.plotly_chart(fig, use_container_width=True)

# Поглиблене порівняння моделей
st.header("8. Поглиблене порівняння моделей")
st.subheader("Вибір моделей для порівняння")
available_models = {
  "Лінійна регресія": LinearRegression(),
  "Поліноміальна (ступінь 2)": Pipeline([
    ('poly', PolynomialFeatures(degree=2)),
    ('linear', LinearRegression())
  ]),
  "Дерево рішень": DecisionTreeRegressor(random_state=42),
  "Випадковий ліс": RandomForestRegressor(random_state=42),
  "Комбінована (кластерна)": ClusteredStackingRegressor(
    base_models=[
      LinearRegression(),
      DecisionTreeRegressor(random_state=42),
      Pipeline([
        ('poly', PolynomialFeatures(degree=2)),
        ('linear', LinearRegression())
      ])
    ],
    meta_model=LinearRegression(),
    n_clusters=3
```

```
    )
}
selected_models = st.multiselect(
    "Оберіть моделі для порівняльного аналізу:",
    list(available_models.keys()),
    default=["Лінійна регресія", "Випадковий ліс"]
)

if selected_models:
    st.subheader("Криві навчання моделей")
    st.markdown("""
    **Як інтерпретувати графік:**
    - **Тренувальна крива (суцільна лінія):** Показує, як модель навчається на тренувальних даних
    - **Валідаційна крива (пунктирна лінія):** Показує, як модель працює на нових даних
    - **Ідеальна ситуація:** Обидві криві збігаються на низькому рівні помилки
    - **Перетренування:** Великий розрив між кривими (валідаційна значно вище)
    """)

    models_to_compare = {name: available_models[name] for name in selected_models}

    try:
        fig = plot_learning_curves(models_to_compare, X_train, y_train)
        st.pyplot(fig)
    except Exception as e:
        st.error(f"Помилка при побудові кривих навчання: {str(e)}")

        # Додаткові метрики порівняння
        st.subheader("Детальні показники продуктивності")

        results = []
        for name, model in models_to_compare.items():
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)
            score = calculate_model_score(model, y_test, y_pred, X_train, y_train)

            results.append({
                "Модель": name,
                "MPS": score['MPS'],
                "R²": score['R2'],
                "RMSE": score['RMSE'],
                "MAE": score['MAE'],
                "Час навчання (сек)": score['Час навчання']
            })

        results_df = pd.DataFrame(results).set_index("Модель")

        # Функція для форматування значень
        def format_value(x):
            if isinstance(x, (int, float)):
                return f"{x:.3f}"
            return str(x)

        # Застосовуємо форматування до DataFrame
        formatted_results = results_df.copy()
        for col in results_df.columns:
            formatted_results[col] = results_df[col].apply(format_value)

        st.dataframe(formatted_results.style
                     .highlight_max(subset=['MPS', 'R²'], color='lightgreen')
                     .highlight_min(subset=['RMSE', 'MAE', 'Час навчання (сек)'], color='lightgreen'))

        # Візуалізація прогнозів
        st.subheader("Графік прогнозів моделей")

        compare_df = pd.DataFrame({
            "Реальні значення": y_test.values
        })

        for model_type in selected_models:
            model = available_models[model_type]
            model.fit(X_train, y_train)
            compare_df[model_type] = model.predict(X_test)

        fig = go.Figure()
        fig.add_trace(go.Scatter(
            x=compare_df.index,
            y=compare_df['Реальні значення'],
            mode='markers',
            name='Реальні значення',
```

```
            marker=dict(color='black', size=8)
        ))

        colors = px.colors.qualitative.Plotly
        for i, name in enumerate(selected_models):
            fig.add_trace(go.Scatter(
                x=compare_df.index,
                y=compare_df[name],
                mode='lines',
                name=name,
                line=dict(color=colors[i % len(colors)], width=2)
            ))

        fig.update_layout(
            title='Порівняння прогнозів моделей',
            xaxis_title='Індекс спостереження',
            yaxis_title='Значення',
            plot_bgcolor='white',
            hovermode='x'
        )
        st.plotly_chart(fig, use_container_width=True)

    except Exception as e:
        st.error(f"Помилка при побудові кривих навчання: {str(e)}")
# Додаємо цей блок після візуалізації результатів обраної моделі
# (перед if __name__ == "__main__":)

st.header(" Фінальний порівняльний аналіз моделей")

# Визначаємо моделі для порівняння
comparison_models = {
    "Лінійна регресія": LinearRegression(),
    "Поліноміальна (2 ступінь)": Pipeline([
        ('poly', PolynomialFeatures(degree=2)),
        ('linear', LinearRegression())
    ]),
    "Дерево рішень": DecisionTreeRegressor(random_state=42),
    "Випадковий ліс": RandomForestRegressor(random_state=42),
    "Комбінована модель": ClusteredStackingRegressor(
        base_models=[
            LinearRegression(),
            DecisionTreeRegressor(random_state=42),
            Pipeline([('poly', PolynomialFeatures(2)), ('linear', LinearRegression())])
        ],
        meta_model=LinearRegression(),
        n_clusters=3
    )
}

# Створюємо DataFrame для результатів
results = []

for model_type, model in comparison_models.items():
    try:
        start_time = time.time()
        model.fit(X_train, y_train)
        train_time = time.time() - start_time
        y_pred = model.predict(X_test)

        # Розраховуємо метрики
        mse = mean_squared_error(y_test, y_pred)
        mae = mean_absolute_error(y_test, y_pred)
        r2 = r2_score(y_test, y_pred)

        # Додаємо результати
        results.append({
            "Модель": model_type,
            "MSE": mse,
            "RMSE": np.sqrt(mse),
            "MAE": mae,
            "R²": r2,
            "Час навчання (с)": train_time
        })
    except Exception as e:
        st.error(f"Помилка при оцінці моделі {model_type}: {str(e)}")

if results:
    results_df = pd.DataFrame(results).set_index('Модель')
```

```
                # Візуалізація результатів
                st.subheader("Таблиця порівняння моделей")
                st.dataframe(
                    results_df.style
                    .highlight_min(subset=['MSE', 'RMSE', 'MAE'], color='lightgreen')
                    .highlight_max(subset=['R²'], color='lightgreen')
                    .format({
                        'MSE': '{:.3f}',
                        'RMSE': '{:.3f}',
                        'MAE': '{:.3f}',
                        'R²': '{:.3f}',
                        'Час навчання (с)': '{:.4f}'
                    })
                )

                # Визначаємо найкращу модель
                best_model = results_df['R²'].idxmax()
                st.success(
                    f"**Найкраща модель за R²:** {best_model} (R² = {results_df.loc[best_model, 'R²']:.3f})")

                # Візуалізація у вигляді графіків
                st.subheader("Візуальне порівняння моделей")

                fig = go.Figure()
                metrics_to_plot = ['RMSE', 'MAE', 'R²']
                colors = px.colors.qualitative.Plotly

                for i, metric in enumerate(metrics_to_plot):
                    fig.add_trace(go.Bar(
                        x=results_df.index,
                        y=results_df[metric],
                        name=metric,
                        marker_color=colors[i],
                        text=results_df[metric].round(3),
                        textposition='auto'
                    ))

                fig.update_layout(
                    barmode='group',
                    title='Порівняння основних метрик моделей',
                    xaxis_title='Моделі',
                    yaxis_title='Значення метрики',
                    plot_bgcolor='white'
                )
                st.plotly_chart(fig, use_container_width=True)

                # Рекомендації
                st.subheader("Рекомендації щодо вибору моделі")

                if results_df.loc[best_model, 'Час навчання (с)'] > 1.0:
                    st.warning(
                        f"Увага: {best_model} має відносно високий час навчання ({results_df.loc[best_model, 'Час навчання (с)']:.2f} сек)")

                st.markdown("""
                **Критерії вибору:**
                - Для максимальної точності: оберіть модель з найвищим R²
                - Для швидкості: оберіть модель з найменшим часом навчання
                - Для балансу: оберіть модель з гарним співвідношенням точності та швидкості
                """)

                # Показуємо найкращі моделі за різними критеріями
                st.markdown("""
                **Найкращі моделі за критеріями:**
                - Точність (R²): {}
                - Швидкість: {}
                - Баланс (точність + швидкість): {}
                """.format(
                    results_df['R²'].idxmax(),
                    results_df['Час навчання (с)'].idxmin(),
                    results_df['R²'].sub(results_df['Час навчання (с)']).idxmax()
                ))
            else:
                st.error("Не вдалося отримати результати для жодної моделі")

if __name__ == "__main__":
    main()
```