ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Підвищення ефективності прогнозування ціни автомобіля за допомогою машинного навчання»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
(код, найменування спеціальності)
освітньо-професійної програми Інженерія програмного забезпечення
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Сергій ГАЛУЗОВ
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-61

Сергій ГАЛУЗОВ

Керівник: Тимур ДОВЖЕНКО
к.т.н.

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Галузову Сергію Юрійовичу

1. Тема кваліфікаційної роботи: Підвищення ефективності прогнозування ціни автомобіля за допомогою машинного навчання

керівник кваліфікаційної роботи Тимур ДОВЖЕНКО к.т.н.

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» жовтня 2023 р. №145.

2. Строк подання кваліфікаційної роботи «29» грудня 2023 р.

3. Вихідні дані до кваліфікаційної роботи: прогнозування ціни, машинне навчання, підвищення ефективності прогнозування, мова програмування Python, бібліотека scikit-learn, бібліотека lightgbm

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Проаналізувати предметну область прогнозування ціни автомобіля.

2. Виконати аналіз технологій машинного навчання для прогнозування ціни автомобіля.
3. Розробити архітектуру системи прогнозування ціни автомобіля.
4. Навести опис взаємодії складових частин системи прогнозування ціни автомобіля.
5. Продемонструвати наглядне підвищення ефективності прогнозування ціни автомобіля за допомогою методів машинного навчання.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз існуючих методів прогнозування ціни автомобіля
2. Опис математичної моделі системи прогнозування ціни автомобіля
3. Особливості роботи моделі машинного навчання LightGBM
4. Схема архітектури системи_прогнозування ціни автомобіля
5. Метрика прогнозування ціни автомобіля
6. Демонстрація результатів прогнозування ціни автомобіля

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10-05.11.23	
2	Вивчення матеріалів для дослідження методів прогнозування ціни автомобіля	06.11-12.11.23	
3	Дослідження особливостей роботи моделі LightGBM	13.11-19.11.23	
5	Побудова програмного забезпечення для прогнозування ціни автомобіля	20.11-03.12.23	
6	Збір та аналіз результатів прогнозування ціни автомобіля	04.12-10.12.23	
7	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.23	
8	Розробка демонстраційних матеріалів	21.12-29.12.23	

Здобувач вищої освіти _____ Сергій ГАЛУЗОВ
(підпис)

Керівник
кваліфікаційної роботи _____ Тимур ДОВЖЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 82 стор., 31 рис., 20 джерел.

Мета роботи – оптимізація прогнозування ціни автомобіля за рахунок використання моделей машинного навчання

Об'єкт дослідження – процес прогнозування ціни автомобіля.

Предмет дослідження – застосування сучасних алгоритмів машинного навчання для підвищення ефективності прогнозування ціни автомобіля.

Короткий зміст роботи: в роботі проведено дослідження існуючих методів прогнозування ціни автомобіля, виділені їхні переваги та недоліки. Проведено огляд наукової літератури по прогнозуванню ціни автомобіля, проаналізовані підходи по використанню методів машинного навчання в предметній області прогнозування ціни автомобіля. Проаналізовано основні особливості моделі машинного навчання LightGBM. Описана математична модель системи прогнозування ціни автомобіля. Створена архітектура системи прогнозування ціни автомобіля. Наведений опис взаємодії складових частин системи прогнозування ціни автомобіля. Описана архітектура фреймворків lxml та scrapy, що використовуються в парсингу та скрапингу даних. Наведені особливості роботи веб-ресурса з оголошеннями про продаж автомобілей. Описана метрика та обґрунтований її вибір для порівняння результатів прогнозування системи. Наведена демонстрація результатів прогнозування ціни автомобіля, що показує підвищення ефективності прогнозування ціни автомобіля за допомогою машинного навчання.

КЛЮЧОВІ СЛОВА: ПРОГНОЗУВАННЯ ЦІНИ, МАШИННЕ НАВЧАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, НАВЧАННЯ З ВЧИТЕЛЕМ, ПАРСИНГ ДАНИХ, СКРАПИНГ ДАНИХ, PYTHON, МОДЕЛЬ LIGHTGBM.

ABSTRACT

Text part of the master's qualification work:
82 pages, 31 pictures, 20 sources.

The purpose of the work - optimization of car price forecasting through the use of machine learning models.

Object of research – car price forecasting process.

Subject of research – application of modern machine learning algorithms to increase the effectiveness of car price forecasting.

Summary of the work: the research delves into the various methods used for predicting car prices, assessing their strengths and weaknesses. It includes a comprehensive review of the scientific literature on car price prediction and an analysis of how machine learning techniques are applied in this field. The unique features of the LightGBM machine learning model are examined in detail. The paper outlines a mathematical model for a car price prediction system and describes its architecture. It also provides an overview of how the components of the car price prediction system interact. The architecture of the lxml and scrapy frameworks, which are used for data parsing and scraping, is explained. The paper also discusses the characteristics of a web resource for car sale advertisements. A specific metric is chosen for comparing the prediction results of the system, and its selection is justified. The paper concludes with a demonstration of the car price prediction results, highlighting the increased efficiency of car price prediction achieved through machine learning.

KEYWORDS: PRICE FORECATING, MACHINE LEARNING, ARTIFICIAL INTELLIGENCE, SUPERVISED LEARNINIG, DATA PARSING, DATA SCRAPING, PYTHON, LIGHTGBM MODEL.

ЗМІСТ

ВСТУП 9
РОЗДІЛ 1. КОНЦЕПТУАЛЬНІ ЗАСАДИ ДОСЛІДЖЕННЯ 10
1.1 Огляд наукової літератури 10
РОЗДІЛ 2. ОБҐРУНТУВАННЯ ОБРАНОГО МЕТОДУ ДОСЛІДЖЕННЯ 19
2.1 Методологія 19
2.2 Обґрунтування вибору моделі LightGBM 22
2.3 Опис дослідження 25
2.4 Обґрунтування достовірності результатів дослідження 25
РОЗДІЛ 3. АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ .. 27
3.1 Опис програмного забезпечення 27
3.3. Опис процеса парсинга даних 30
3.4 Опис процесів очищення та обробки даних 31
3.4.1 Опис трансформуючого пайплайна 31
3.5 Опис процесів роботи з базою даних 33
3.5.1 Опис особливостей роботи веб-ресурса з оголошеннями 35
3.6 Проведення EDA (Exploratiry Data Analysis) 37
3.6 Опис формування тренувального датасета 49
3.7 Порівняння результатів прогнозування методів 52
ВИСНОВКИ 53
ПЕРЕЛІК ПОСИЛАНЬ 54
ДОДАТОК А 57
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) 86

ВСТУП

Прогнозування ціни автомобіля є важливим завданням, яке має значний вплив на автомобільну промисловість. Воно допомагає визначити вартість вживаного автомобіля, що є критичним для дилерів, страхових компаній та покупців. Однак, це завдання є складним через велику кількість факторів, які впливають на ціну автомобіля, таких як марка, модель, рік випуску, пробіг, стан автомобіля та інше.

Машинне навчання відкриває нові можливості для підвищення ефективності прогнозування ціни автомобіля. Застосування алгоритмів машинного навчання дозволяє обробляти великі обсяги даних та виявляти складні шаблони, які можуть бути недоступні для традиційних статистичних методів. Це може включати в себе використання регресійних моделей, дерев рішень, нейронних мереж та інших методів для створення точних прогнозів.

Однак, використання машинного навчання для прогнозування ціни автомобіля також має свої виклики. Це включає в себе необхідність збору та обробки великих обсягів даних, вибір правильного алгоритму машинного навчання, налаштування параметрів моделі та перевірка її ефективності. Незважаючи на ці виклики, потенціал машинного навчання в області прогнозування ціни автомобіля є великим і продовжує рости.

РОЗДІЛ 1. КОНЦЕПТУАЛЬНІ ЗАСАДИ ДОСЛІДЖЕННЯ

Прогнозування ціни на автомобіль є важливою задачею в галузі автомобільного маркетингу. Воно допомагає встановити оптимальну ціну для автомобіля, що відображає його реальну вартість на основі його характеристик. Це може призвести до збільшення прибутку від продажу автомобілів.

1.1 Огляд наукової літератури

Багато дослідників вже вивчали проблему прогнозування цін на автомобілі. Вони використовували різні методи, включаючи статистичні методи, машинне навчання та глибоке навчання. Однак, більшість цих методів мають певні обмеження, такі як необхідність великої кількості даних для навчання або висока складність моделі.

CS 229 Project Report: Predicting Used Car Prices - Stanford University: цей проект зосереджений на розробці моделей машинного навчання, які можуть точно прогнозувати ціну вживаного автомобіля на основі його характеристик, щоб зробити обґрунтовані покупки. Вони використовують різні методи навчання на датасеті, що складається з цін продажу різних марок і моделей по всіх містах США. Їхні результати показують, що модель Random Forest та K-Means clustering з лінійною регресією дають найкращі результати, але вони вимагають багато обчислювальних ресурсів. Звичайна лінійна регресія також дала задовільні результати, з перевагою значно меншого часу навчання порівняно з вищезгаданими методами.

Prediction of Used Car Prices Using Machine Learning: у цій статті автори мають на меті побудувати модель для прогнозування розумних цін на вживані автомобілі на основі різних аспектів, включаючи пробіг автомобіля, рік виробництва, витрати палива, трансмісію, податок на дорогу, тип палива та розмір двигуна. Модель може бути корисною для продавців, покупців та виробників автомобілів на ринку вживаних

автомобілів. Після завершення вона може виводити відносно точний прогноз ціни на основі введеної користувачами інформації. Процес побудови моделі включає машинне навчання та науку про дані. Датасет, який був використаний, був зібраний з оголошень про продаж вживаних автомобілів. У дослідженні були застосовані різні методи регресії, включаючи лінійну регресію, поліноміальну регресію, регресію опорних векторів, регресію дерева рішень та регресію випадкового лісу, щоб досягти найвищої точності. Перед початком побудови моделі в цьому проекті було візуалізовано дані, щоб краще зрозуміти набір даних. Датасет було розділено та модифіковано, щоб відповідати регресії, тим самим забезпечуючи продуктивність регресії. Для оцінки продуктивності кожної регресії було розраховано R-квадрат. Серед усіх регресій у цьому проекті найвищий R-квадрат, 0.90416, досягла регресія випадкового лісу. У порівнянні з попередніми дослідженнями, отримана модель включає більше аспектів вживаних автомобілів, а також має вищу точність прогнозування.

Predicting Used-Vehicle Resale Value in Developing Markets: Application of Machine Learning Models to the Kazakhstan Car Market: це дослідження зосереджується на розробці моделі для прогнозування вартості вживаних автомобілів на ринку Казахстану. Це дослідження є важливим, оскільки на ринках країн, що розвиваються, таких як Казахстан, велика кількість вживаних автомобілів і обмежена кількість нових автомобілів призводить до переваги вживаних автомобілів. Тому аналіз даних про продаж стає важливим для отримання цінних висновків. У дослідженні було використано різні техніки регресії, такі як лінійна регресія, регресія дерева рішень, машини опорних векторів, нейронна мережа та Bagged Trees, разом з алгоритмами машинного навчання, для прогнозування ціни продажу вживаних автомобілів на основі пов’язаних з ними особливостей. Метрики оцінки були використані для визначення найбільш ефективної моделі, шляхом вивчення продуктивності та помилкового показника кожної моделі. Глибока нейронна мережа показала виняткову продуктивність, як це вказує на її низькі значення RMSE та MSE, що свідчить про

високу ефективність цієї моделі. Крім того, вузькі, середні, двошарові та трьохшарові нейронні мережі показали прекрасну продуктивність при запису кореляцій змінних. Після порівняння різних моделей, Bagged Trees були визнані найбільш економічно ефективним варіантом через їхні переваги в ціноутворенні та продуктивності.

Machine Learning Techniques To Predict The Price Of Used Cars: Predictive Analytics in Retail Business: це дослідження зосереджується на використанні прогностичної аналітики та машинного навчання для підтримки процесу прийняття рішень в бізнесі з продажу вживаних автомобілів. Це дослідження є важливим, оскільки воно допомагає зменшити ризик для продавців та споживачів, надаючи приблизне уявлення про ціну продажу вживаного автомобіля на основі його характеристик. Основна мета цього дослідження полягає в побудові моделі прогнозування, яка може передбачити ціну продажу автомобіля на основі таких характеристик, як модель автомобіля, кількість років, протягом яких автомобіль був у використанні, тип палива, тип продавця, тип трансмісії та кількість кілометрів, які автомобіль проїхав. У дослідженні було використано алгоритми машинного навчання та техніки регресії, такі як лінійна регресія, регресія дерева рішень та регресія випадкового лісу, для досягнення цієї мети. Прогностична аналітика включає використання статистичних методів та технологій для аналізу історичних даних, що допомагає бізнесу отримати нові інсайти та планувати майбутнє. Це дослідження показує, що прогностичні аналітичні моделі можуть бути великим доповненням до бізнесу, особливо для підтримки процесу прийняття рішень. Важливо зазначити, що неправильне прийняття рішень може призвести до великих втрат і навіть до закриття бізнесу. Тому це дослідження пропонує новий підхід до вирішення цієї проблеми.

Predicting used car prices with deep learning - Stanford University: дослідження зосереджується на використанні глибокого навчання для прогнозування цін на вживані автомобілі. У цьому дослідженні автори використовують різні архітектури глибокого навчання, щоб краще прогнозувати ціни на вживані автомобілі, ніж інші моделі, що існують у сучасній літературі. Вони порівнюють продуктивність між

простою лінійною регресією, MLP1 та MLP2 моделями, які є штучними нейронними мережами з різними налаштуваннями та гіперпараметрами. Автори знаходять, що глибші нейронні мережі з більшою кількістю прихованих шарів та меншою кількістю нейронів на шар виконують краще, ніж більш поверхневі, з більшою кількістю нейронів на шар, а також лінійна регресія, на основі середньоквадратичної логарифмічної помилки. Це дослідження підкреслює важливість прозорості та точності ціноутворення на ринку вживаних автомобілів. Ціни на вживані автомобілі не є такими прямими, як при купівлі нового автомобіля, з численними факторами, які, ймовірно, впливають на структуру ціноутворення, включаючи пробіг, попередні аварії тощо. Автори бачать потребу розробити алгоритм глибокого навчання для точного прогнозування цін на вживані автомобілі, використовуючи стандартні особливості, такі як марка, модель, пробіг тощо. Це дослідження вносить важливий внесок у розуміння того, як можна використовувати глибоке навчання для прогнозування цін на вживані автомобілі, і може слугувати основою для подальших досліджень в цій області.

USED CAR PRICE PREDICTION - IRJET Journal: у цьому дослідженні автори розробляють модель, яка ефективно визначає вартість автомобіля на основі різних характеристик. Це важливо, оскільки на глобальному рівні спостерігається зростання продажів вживаних автомобілів через збільшення цін на нові автомобілі та фінансову неспроможність покупців купувати їх. Існуюча система включає процес, в якому продавець випадково визначає ціну, а покупець не має уявлення про автомобіль та його сучасну вартість. Насправді, продавець також не має уявлення про справжню вартість автомобіля або ціну, за яку він повинен продавати автомобіль. Для вирішення цієї проблеми автори розробили модель, яка буде високоефективною. Вони використовують алгоритми регресії, оскільки вони надають нам безперервне значення як вихідне, а не категоризоване значення. Це дозволяє прогнозувати фактичну ціну автомобіля, а не діапазон цін автомобіля. Автори також розробили інтерфейс

користувача, який отримує вхід від будь-якого користувача та відображає ціну автомобіля відповідно до введених користувачем даних.

How much is my car worth? A methodology for predicting used cars prices using Random Forest: дослідження зосереджується на використанні методу машинного навчання, відомого як Random Forest, для прогнозування цін на вживані автомобілі. У цьому дослідженні автори використовують метод навчання з учителем, а саме Random Forest, для прогнозування цін на вживані автомобілі. Модель була обрана після ретельного дослідницького аналізу даних, щоб визначити вплив кожної особливості на ціну. Автори створили Random Forest з 500 деревами рішень для тренування даних. З експериментальних результатів виявлено, що точність тренування становила 95,82%, а точність тестування - 83,63%. Модель може точно прогнозувати ціну автомобілів, вибираючи найбільш корельовані особливості. У дослідженні було використано різні методи регресії, включаючи лінійну регресію, регресію дерева рішень, машини опорних векторів, нейронну мережу та Bagged Trees, щоб досягти найвищої точності. Перед початком побудови моделі в цьому проекті було візуалізовано дані, щоб краще зрозуміти набір даних. Датасет було розділено та модифіковано, щоб відповідати регресії, тим самим забезпечуючи продуктивність регресії. Для оцінки продуктивності кожної регресії було розраховано R-квадрат. Серед усіх регресій у цьому проекті найвищий R-квадрат, 0.90416, досягла регресія випадкового лісу.

Price evaluation model in second-hand car system based on BP neural network theory: дослідження зосереджується на використанні теорії нейронної мережі зворотного поширення помилки (BP) для оцінки цін на вживані автомобілі.У цьому дослідженні автори пропонують модель оцінки цін, яка базується на аналізі великих даних і використовує оптимізований алгоритм BP нейронної мережі. Ця модель була розроблена з метою отримання ціни, яка найкраще відповідає автомобілю. Автори використовують оптимізований алгоритм BP нейронної мережі для вибору оптимальної кількості прихованих нейронів в нейронній мережі. Це покращує

швидкість збіжності топології мережі та точність моделі прогнозування. За допомогою вибіркових моделювальних експериментів автори порівнюють криву прогнозування цін з реальною ціною угоди, отриманою з оптимізованої моделі. В результаті виявлено, що точність оптимізованої моделі вища, а відповідність краща. Це дослідження важливе, оскільки з ростом числа приватних автомобілів та розвитком ринку вживаних автомобілів, вживані автомобілі стають основним вибором при купівлі автомобілів.

Performance Evaluation of Popular Machine Learning Models for Used Car Price Prediction: дослідження зосереджується на оцінці продуктивності популярних моделей машинного навчання для прогнозування цін на вживані автомобілі. У цьому дослідженні автори аналізують п’ять популярних алгоритмів машинного навчання для прогнозування цін на вживані автомобілі: XGBoost, KNN, випадковий ліс, дерево рішень та лінійну регресію. Продуктивність кожного алгоритму оцінюється за допомогою метрик R-квадрат (R2) та середньої абсолютної помилки (MAE). Експерименти проводилися на наборі даних, що містить інформацію про різні вживані автомобілі, включаючи їхній пробіг, модель, вік, тип палива та марку. Результати показали, що XGBoost перевершує інші алгоритми з R2-оцінкою 0.81, а випадковий ліс має R2-оцінку 0.77. Це дослідження підкреслює важливість використання відповідних алгоритмів машинного навчання для прогнозування цін на вживані автомобілі та демонструє перевагу ансамблевих методів над лінійними моделями.

Used car price prediction using K-nearest neighbor based model: у цьому дослідженні автори пропонують модель машинного навчання, яка використовує алгоритм регресії KNN для аналізу цін на вживані автомобілі. Вони навчають свою модель даними про вживані автомобілі, які були зібрані з веб-сайту Kaggle. Автори проводять експерименти з різними співвідношеннями тренувальних та тестових даних. В результаті виявлено, що точність пропонованої моделі становить близько 85%, і вона вважається оптимізованою моделлю. Це дослідження важливе, оскільки з ростом числа приватних автомобілів та розвитком ринку вживаних автомобілів,

вживані автомобілі стають основним вибором при купівлі автомобілів. Тому аналіз даних про продаж стає важливим для отримання цінних висновків. У дослідженні було використано різні методи регресії, включаючи лінійну регресію, регресію дерева рішень, машини опорних векторів, нейронну мережу та Bagged Trees, щоб досягти найвищої точності. Перед початком побудови моделі в цьому проекті було візуалізовано дані, щоб краще зрозуміти набір даних. Датасет було розділено та модифіковано, щоб відповідати регресії, тим самим забезпечуючи продуктивність регресії. Для оцінки продуктивності кожної регресії було розраховано R-квадрат. Серед усіх регресій у цьому проекті найвищий R-квадрат, 0.90416, досягла регресія випадкового лісу.

Vehicle price prediction system using machine learning techniques: дослідження зосереджується на використанні технік машинного навчання для прогнозування цін на вживані автомобілі. У цьому дослідженні автори використовують метод машинного навчання, відомий як лінійна регресія, для прогнозування цін на вживані автомобілі. Вони використовують великий набір даних, зібраний з різних джерел, для тренування та тестування своєї моделі. Автори проводять експерименти з різними співвідношеннями тренувальних та тестових даних. В результаті виявлено, що точність пропонованої моделі становить близько 98%, і вона вважається оптимізованою моделлю. Це дослідження важливе, оскільки з ростом числа приватних автомобілів та розвитком ринку вживаних автомобілів, вживані автомобілі стають основним вибором при купівлі автомобілів. Тому аналіз даних про продаж стає важливим для отримання цінних висновків.

Car price prediction using machine learning techniques: дослідження зосереджується на використанні технік машинного навчання для прогнозування цін на вживані автомобілі. У цьому дослідженні автори використовують три техніки машинного навчання: штучну нейронну мережу, машину опорних векторів та випадковий ліс. Ці техніки були застосовані для роботи як ансамбль. Дані, використані для прогнозування, були зібрані з веб-порталу autopijaca.ba за допомогою

веб-скребка, написаного на мові програмування PHP. Потім були порівняні відповідні продуктивності різних алгоритмів, щоб знайти той, який найкраще підходить для доступного набору даних. Кінцева модель прогнозування була інтегрована в Java-додаток. Крім того, модель була оцінена за допомогою тестових даних, і було отримано точність 87,38%.

Integrated Linear Regression and Random Forest Framework for E-Commerce Price Prediction of Pre-owned Vehicle: дослідження представлене на Міжнародній конференції з тенденцій в електроніці та інформатиці здоров'я, зосереджується на використанні моделей лінійної регресії та випадкового лісу для прогнозування цін на вживані автомобілі. У цьому дослідженні автори пропонують модель машинного навчання, яка використовує два алгоритми: лінійну регресію та випадковий ліс. Вони порівнюють продуктивність цих двох алгоритмів на основі стандартних показників продуктивності. Особливістю цієї роботи є те, що вона не тільки здатна прогнозувати ціну вживаного автомобіля, але модель також може бути розширена з мінімальними зусиллями для будь-якого виду продукту в різних сферах електронної комерції. Це дослідження важливе, оскільки з ростом числа приватних автомобілів та розвитком ринку вживаних автомобілів, вживані автомобілі стають основним вибором при купівлі автомобілів.

Object detection and used car price predicting analysis system (UCPAS) using machine learning technique: Це дослідження зосереджується на використанні моделей машинного навчання для виявлення об'єктів та прогнозування ціни вживаних автомобілів. Вони розглядають концепцію виявлення об'єктів, таких як автомобілі, та досліджують ціну вживаного автомобіля за допомогою автоматичних методів машинного навчання. Вони також розуміють концепцію категорій виявлення об'єктів. Найбільш складним завданням зараз є визначення ринкової ціни вживаного автомобіля. Є багато факторів, які можуть впливати на ціну вживаного автомобіля. Основною метою цієї статті є розробка моделей машинного навчання, які дозволяють точно прогнозувати ціну вживаного автомобіля відповідно до його параметрів або

характеристик. У цій статті використовуються методи реалізації та оцінювання на наборі даних про автомобілі, який складається з цін продажу різних моделей автомобілів у різних містах Індії. Результати цього експерименту показують, що кластеризація з лінійною регресією та модель випадкового лісу дає найкращий результат точності.

A comprehensive study of machine learning algorithms for predicting car purchase based on customers demands: це дослідження зосереджується на вивченні алгоритмів машинного навчання для прогнозування покупки автомобілів на основі вимог клієнтів. Вони розглядають різні алгоритми машинного навчання та їх застосування для прогнозування покупки автомобілів. Вони також розуміють концепцію використання машинного навчання для аналізу вимог клієнтів та їх впливу на рішення про покупку. Це включає в себе розуміння того, як різні фактори, такі як ціна, марка, модель, рік випуску та інші характеристики автомобіля, можуть впливати на рішення клієнта про покупку. Основною метою цього дослідження є розробка ефективних моделей машинного навчання, які можуть точно прогнозувати покупку автомобілів на основі вимог клієнтів. Це включає в себе розробку та оцінювання різних алгоритмів машинного навчання та їх здатності до прогнозування. Вони також розглядають можливість використання цих моделей в реальному світі та їх потенційну користь для автомобільних дилерів та покупців. Це включає в себе розуміння того, як ці моделі можуть бути використані для покращення процесу прийняття рішень та збільшення продажів.

РОЗДІЛ 2. ОБҐРУНТУВАННЯ ОБРАНОГО МЕТОДУ ДОСЛІДЖЕННЯ

2.1 Методологія

Існує багато способів спрогнозувати ціну продажу автомобіля. Одними з таких способів є:

1. *Лінійний регресійний аналіз* - відносно простий метод моделювання залежності ціни від факторів, в якому передбачається саме лінійна залежність ціни від факторів.
2. *Метод часових рядів* - в основі методу лежить нерозривна та періодична послідовність вимірювань ціни у часі. Це може бути щоденна, щотижнева, місячна або інша періодичність.
3. *Метод експертної оцінки* - оцінка ціни автомобіля передається на експертну сторону, де експерти а галузі маркета автомобілей визначають вартість автомобіля на основі свого досвіду, ситуації на ринку та внутрішнім переконанням.
4. ***Методи машинного навчання*** - будується математична модель на основі історичних даних, яка автоматично визначає паттерни в даних та узагальнює залежність між факторами та ціною автомобіля.

Кожен з цим методів має ті чи інші недоліки:

1. Лінійний регресійний аналіз вимагає певного рівня кореляції між змінними для точних прогнозів. Також він не враховує складні нелинійні залежності між факторами.
2. Метод часових рядів вимагає певного рівня кореляції між змінними для точних прогнозів. Також він не враховує складні нелинійні залежності між факторами.
3. Метод експертної оцінки виносить на перший план суб'єктивність та упередженість оцінок експертів. Також цей метод має обмеженість в кількості факторів для надання оцінки.

4. Методи машинного навчання нівелюють недоліки попередньо описаних методів, проте вимагають великої кількості даних для навчання ефективних моделей.

В даній роботі будуть використовуватися методи машинного навчання для прогнозування цін на автомобілі. Ці методи дозволяють нам врахувати велику кількість факторів, які впливають на ціну автомобіля, і створити модель, яка може досить точно прогнозувати ціну на основі цих факторів.

У цьому дослідженні буде використаний алгоритм градієнтного бустінгу LightGBM для прогнозування цін на автомобілі. LightGBM є ефективним алгоритмом, який використовує дерева, що засновані на градієнтному бустінгу, і відомий своєю швидкістю та точністю.

LightGBM, або Light Gradient Boosting Machine, є алгоритмом градієнтного бустінгу, який використовує дерева рішень.

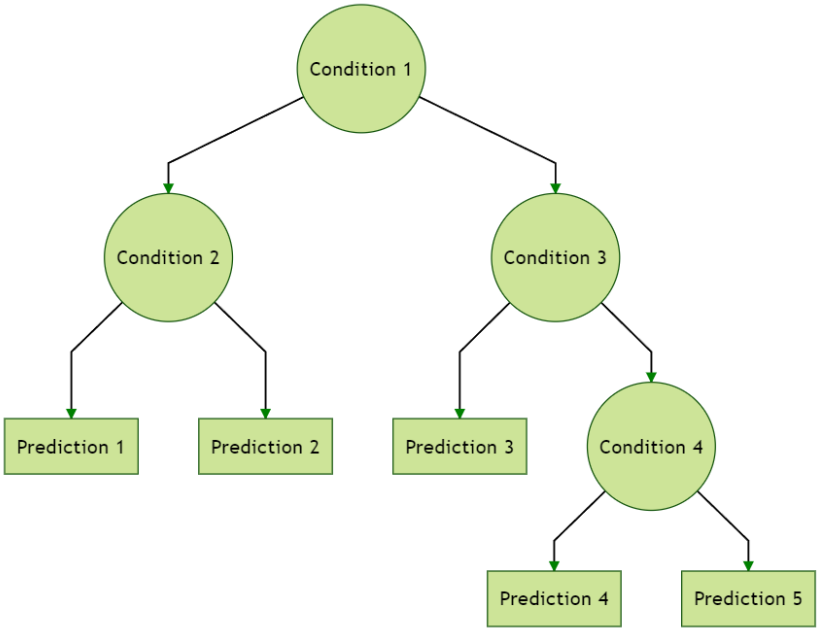


Рис. 2.1. Структура дерева рішень

LightGBM був розроблений Microsoft і відомий своєю швидкістю та ефективністю, особливо на великих наборах даних.

Градієнтний бустінг є методом ансамблю, який будує послідовність слабких моделей, зазвичай дерев рішень, кожна з яких намагається виправити помилки попередньої моделі. Кінцевий прогноз є виваженою сумою прогнозів всіх моделей.

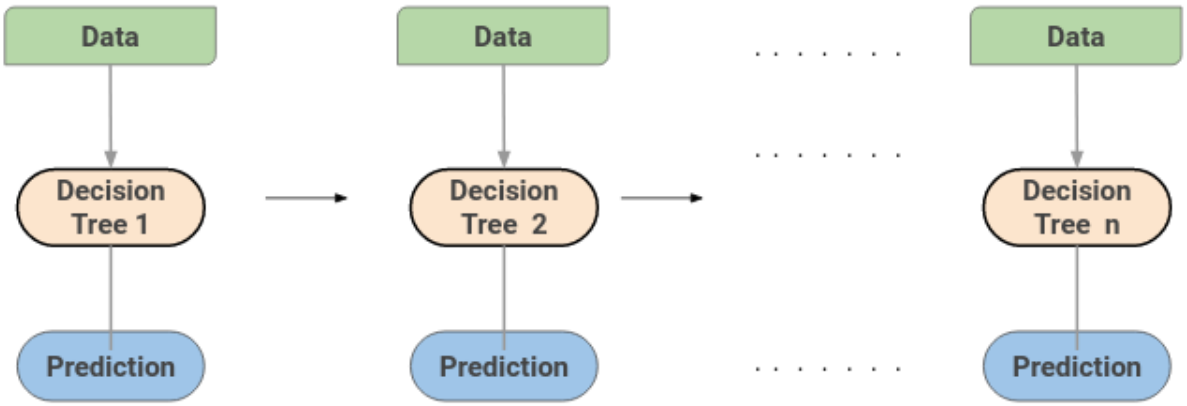


Рис. 2.2. Зображення посилення дерев рішень в LightGBM

2.2 Опис математичної моделі LightGBM

X – набір вхідних ознак (властивості автомобіля, такі як рік випуску, пробіг, бренд тощо).

Y - цільова змінна (ціна автомобіля).

Θ - параметри моделі машинного навчання, які оптимізуються під час тренування моделі.

Функція витрат(Θ)= $L(Y, F(X; \Theta))+\Omega(\Theta)$, де:

$F(X; \Theta)$ - прогноз ціни автомобіля, отриманий від моделі машинного навчання за параметрами Θ.

$L(Y, F(X; \Theta))$ - функція втрат, яка вимірює різницю між прогнозом і фактичною ціною.

Ω(Θ) - регуляризаційна функція для контролю перенавчання моделі.

Мета математичної моделі - мінімізація функції витрат відносно параметрів Θ моделі (2.1):

$$\Theta^* = \text{argmin}_\Theta(L(Y, F(X; \Theta)) + \Omega(\Theta)). \quad (2.1)$$

Функція втрат в LightGBM, як і в більшості алгоритмів градієнтного бустінгу, є диференційованою функцією, яка вимірює різницю між прогнозованими та реальними значеннями. Для задачі регресії, як у нашому випадку з прогнозуванням ціни автомобіля, зазвичай використовується середньоквадратична помилка (MSE):

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \quad (2.2)$$

2.3 Обґрунтування вибору моделі LightGBM

Модель LightGBM був обраний через її здатність ефективно обробляти великі набори даних і високу точність прогнозування. Вона використовує техніку, відому як градієнтний бустінг з використанням дерев рішень, яка дозволяє моделі вчитися з помилок попередніх дерев і поступово покращувати свої прогнози. Крім того, LightGBM має ряд важливих особливостей, таких як підтримка категоріальних ознак, що робить його особливо корисним для нашої задачі прогнозування цін на автомобілі.

LightGBM використовує декілька оптимізацій, які роблять його ефективним на великих наборах даних. Він використовує стратегію розбиття на листя (leaf-wise split), яка дозволяє більш точно контролювати складність моделі, порівняно з традиційними алгоритмами, які використовують стратегію розбиття на рівні (level-wise split). Крім того, LightGBM підтримує обробку категоріальних ознак без необхідності їх попереднього кодування.

Використання інших методів прогнозування ціни на автомобіль може мати декілька недоліків порівняно з LightGBM:

Обчислювальна складність: Деякі алгоритми, такі як випадковий ліс або SVM, можуть бути обчислювально важкими, особливо при роботі з великими наборами даних. LightGBM, з іншого боку, відомий своєю ефективністю та швидкістю, що робить його більш підходящим для великих наборів даних. LightGBM, або Light Gradient Boosting Machine, відомий своєю швидкістю та ефективністю. Це досягається за рахунок декількох ключових особливостей:

- Leaf-wise Growth: Більшість алгоритмів градієнтного бустінгу використовують стратегію розбиття на рівні (level-wise), де дерева ростуть горизонтально. З іншого боку, LightGBM використовує стратегію розбиття на листя (leaf-wise), де дерева ростуть вертикально. Це означає, що LightGBM буде розбивати дерево на місці з найбільшим втратам, що може зменшити більше втрат, ніж рівневий алгоритм.
- Histogram-based Algorithm: LightGBM використовує гістограмний алгоритм, який обчислює відповідні статистики гістограми, а не всіх значень. Це значно зменшує обчислювальну складність.
- Handling of Categorical Features: LightGBM може обробляти категоріальні ознаки безпосередньо, без необхідності їх попереднього кодування. Це може призвести до значного прискорення тренування моделі, особливо коли є багато категоріальних ознак з великою кількістю унікальних значень.
- Parallel Learning: LightGBM підтримує паралельне навчання, що дозволяє йому ефективно використовувати багатоядерні процесори для прискорення процесу навчання.
- GPU Support: LightGBM підтримує навчання на графічних процесорах (GPU), що може забезпечити значне прискорення для великих наборів даних.

Ці особливості роблять LightGBM одним з найшвидших алгоритмів градієнтного бустінгу, доступних сьогодні.

Точність прогнозування: Деякі методи можуть не надавати такої ж точності прогнозування, як LightGBM.

Обробка категоріальних ознак: Деякі алгоритми вимагають попереднього кодування категоріальних ознак, що може бути трудомісним та може вплинути на точність моделі. LightGBM, з іншого боку, може обробляти категоріальні ознаки без необхідності їх попереднього кодування.

LightGBM може безпосередньо обробляти категоріальні ознаки, що робить його ефективним для задач, де категоріальні ознаки грають важливу роль. Категоріальні ознаки, також відомі як дискретні або якісні ознаки, представляють категорії або мітки. Ці ознаки зазвичай приймають обмежену, фіксовану кількість унікальних значень, і вони не мають природного порядку між ними. LightGBM має вбудовану підтримку для обробки категоріальних ознак. Він може ефективно обробляти категоріальні дані високої мірності без необхідності великої попередньої обробки. LightGBM використовує цілочисельне кодування для категоріальних ознак і застосовує метод Fisher (1958) для знаходження оптимального розбиття по категоріях. Це часто працює краще, ніж one-hot кодування. Ви повинні перетворити свої категоріальні ознаки на тип int перед тим, як ви створите Dataset[3]. Ви можете використовувати параметр categorical_feature для вказівки категоріальних ознак. Однією з ключових переваг LightGBM є його здатність ефективно обробляти категоріальні ознаки без необхідності їх попереднього кодування. Це може призвести до значного прискорення тренування моделі, особливо коли є багато категоріальних ознак з великою кількістю унікальних значень. Крім того, LightGBM може надати кращу точність прогнозування, ніж інші методи, які використовують one-hot кодування для категоріальних ознак. Це тому, що LightGBM може врахувати взаємозв’язки між різними категоріями, тоді як one-hot кодування втрачає цю інформацію.

Недостатність даних: Деякі методи можуть мати низьку продуктивність, якщо доступний набір даних недостатній. LightGBM, з іншого боку, відомий своєю здатністю до ефективного навчання на великих наборах даних.

Відсутність ітеративного фреймворку: Деякі методи, такі як випадковий ліс або глибока нейронна мережа, можуть не мати ітеративного фреймворку, який дозволяє поступово покращувати модель. LightGBM, з іншого боку, використовує градієнтний бустінг, який є ітеративним процесом, що дозволяє моделі вчитися з помилок попередніх моделей і поступово покращувати свої прогнози.

2.4 Опис дослідження

Буде проведено ряд експериментів для перевірки ефективності нашої моделі LightGBM. Це включатиме збір та оброку даних для тренування моделі LightGBM на них, налаштування гіперпараметрів моделі для оптимізації її продуктивності та перевірку її прогнозів на тестовому наборі даних.

Також буде продемонстрований відносний вплив кожного з факторів на формування прогнозу моделі.

2.5 Обґрунтування достовірності результатів дослідження

Буде проведений EDA (Exploratoty Data Analysis) на даних, що будуть застосовуватися в навчанні моделі. Це дасть змогу зрозуміти

Буде використана крос-валідація для перевірки достовірності нашої моделі. Це допоможе зрозуміти взаємозв’язки в даних та знайти аномалії в даних, що буде корисним при формуванні навчального датасета для моделі.

Крос-валідація - це метод статистичної оцінки моделі машинного навчання. Вона використовується для оцінки того, як результати статистичного аналізу будуть узагальнюватися на незалежний набір даних.

Це важливо для того, щоб переконатися, що модель не перенавчилася на тренувальних даних. Крос-валідація працює шляхом розділення набору даних на дві частини: тренувальний набір та тестовий набір. Модель навчається на тренувальному наборі даних і тоді тестується на тестовому наборі.

LightGBM може визначити важливість ознак, що допомагає зрозуміти, які ознаки найбільше впливають на прогнози моделі. Це може бути корисним для інтерпретації моделі та її результатів.

Важливість фактора в LightGBM може бути обчислена двома способами:

- Split: Цей метод обчислює важливість фактора на основі кількості разів, коли фактор використовувся для розбиття в моделі.
- Gain: Цей метод обчислює важливість фактора на основі загального поліпшення прогнозу, яке було отримано в результаті розбиття по даному фактору.

Крім того, будуть використані метрики оцінки прогнозування, такі як середня абсолютна помилка (MAE) та середньоквадратична помилка (RMSE), для кількісної оцінки точності прогнозів моделі.

РОЗДІЛ 3. АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

3.1 Опис програмного забезпечення

Було розроблено програмне забезпечення, яке виконує наступні дії:

1. Проходить процес скрапинга даних, в якому дані з веб-сторінок оголошень про продаж автомобілей зберігаються за допомогою скрапера у вигляді html-файлів.
2. Кожен збережений html-файл з оголошенням проходить процедуру парсинга, де витягується інформація з тегів html та зберігається у вигляді json-файлів.
3. Відбувається обробка отриманих з оголошення даних, проводяться операції очистки та трансформації даних.
4. Очищені дані складаються в базу даних.
5. З даних БД формується тренувальний датасет для навчання моделі машинного навчання lightgbm.
6. З цільових даних про автомобілі формується тестовий датасет, на якому відбувається прогноз моделі та видається результат у вигляді прогнозованої оптимальної ціни на автомобіль.

3.2 Опис процесу скрапинга даних

Процес скрапинга даних має наступну послідовність кроків:

1. ПЗ починає виконувати запити зі стартової URL – першої сторінки в ленті пошука машин.
2. ПЗ заходить на кожне оголошення в даній сторінці на ленті.
3. ПЗ зберігає інформацію про оголошення локально у вигляді html-файлів
4. Дойдя до кінця ленти, ПЗ заходить на наступну сторінку з лентою оголошень.
5. ПЗ повторює кроки 2-4 до останньої сторінки з лентою оголошень включно.

Технічний фреймворк, що був застосований – Scrapy. Він має наступну архітектуру:

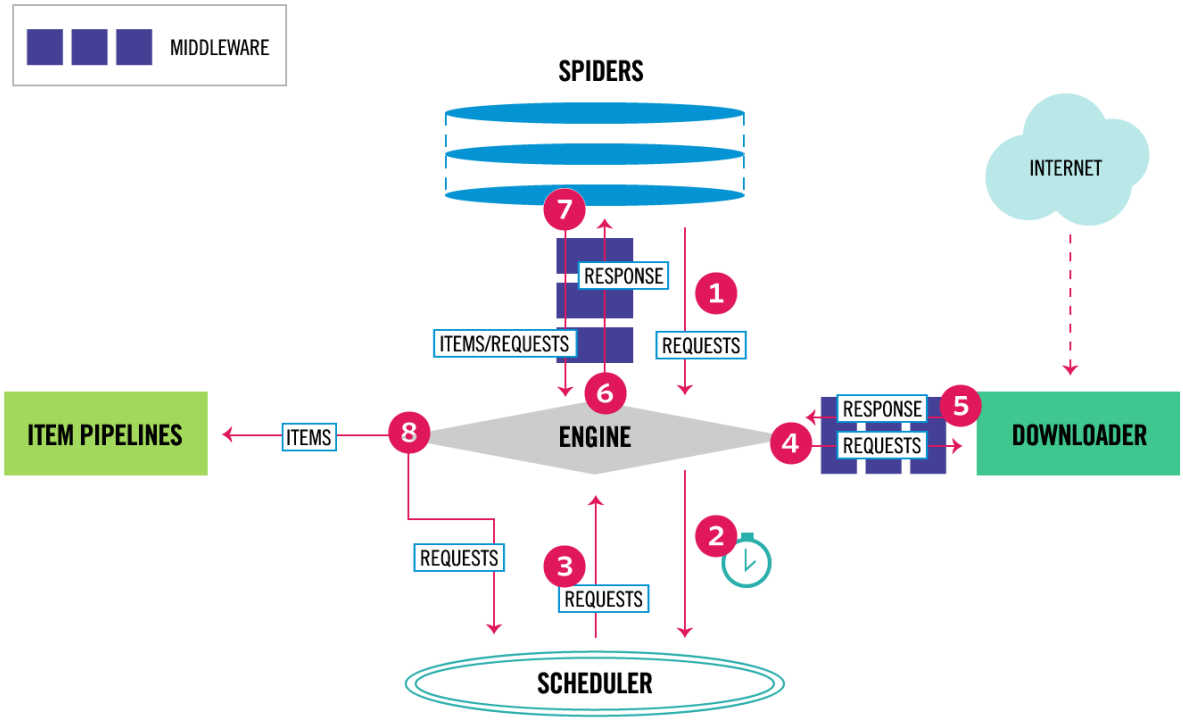


Рис. 3.1. Архітектура фреймворка Scrapy

Потік даних у Scrapy контролюється механізмом виконання та виглядає так:

1. Механізм отримує початкові запити на сканування від Spider.
2. Механізм розкладає запити в планувальнику та запитує наступні запити для сканування.
3. Планувальник повертає наступні запити до механізму.
4. Механізм надсилає запити завантажувачу, проходячи через проміжне програмне забезпечення завантажувача.
5. Після завершення завантаження сторінки Downloader генерує відповідь (з цією сторінкою) і надсилає її до Engine, проходячи через проміжне програмне забезпечення Downloader (див. process_response()).

6. Механізм отримує відповідь від завантажувача та надсилає його Spider для обробки, проходячи через проміжне програмне забезпечення Spider.
7. Spider обробляє відповідь і повертає скопійовані елементи та нові запити (що слідують) до Engine, проходячи через проміжне програмне забезпечення Spider (див. process_spider_output()).
8. Механізм надсилає оброблені елементи до конвеєрів елементів, а потім надсилає оброблені запити до планувальника та запитує можливі наступні запити для сканування.
9. Процес повторюється (з кроку 3), доки більше не буде запитів від Планувальника.

Scrapy - це високорівневий фреймворк для веб-скрапінгу та веб-краулінгу, який використовується для обходу веб-сайтів та витягування структурованих даних з їх сторінок. Він може використовуватися для широкого спектру цілей, від добування даних до моніторингу та автоматизованого тестування.

Основні компоненти Scrapy включають:

1. Spiders: Правила для обходу веб-сайтів.
2. Selectors: Витягування даних з веб-сторінок за допомогою XPath.
3. Items: Визначення даних, які ви хочете витягнути.
4. Item Loaders: Заповнення елементів витягнутими даними.
5. Item Pipeline: Пост-обробка та зберігання витягнутих даних.
6. Requests and Responses: Класи, що використовуються для представлення HTTP-запитів та відповідей.
7. Link Extractors: Зручні класи для витягування посилань для подальшого слідування зі сторінок.

Scrapy також має вбудовані сервіси, такі як:

1. Logging: Використання вбудованого в Python журналування в Scrapy.
2. Stats Collection: Збір статистики про скрапінг-краулер.

3. Sending e-mail: Відправка сповіщень електронною поштою, коли виникають певні події.
4. Telnet Console: Перевірка робочого краулера за допомогою вбудованої консолі Python.

Scrapy підтримується Zyte (раніше Scrapinghub) та багатьма іншими учасниками. Це відкритий та спільний фреймворк, який дозволяє витягувати необхідні дані з веб-сайтів швидко, просто, але при цьому гнучко.

3.3. Опис процеса парсинга даних

Процес парсинга даних має наступну послідовність кроків.

1. В якості вхідних даних цього процеса використовуються збережені після скрапинга html-файли.
2. Відбувається парсинг даних – пошук ключових даних в оголошенні. Це відбувається за допомогою python бібліотеки lxml. Використовуючи CSS-селектори та xpath, ПЗ дістає інформацію у вигляді тексту з html-тегів оголошення та зберігає ці дані у вигляді асоціативного масива, де ключем є назва параметра автомобіля (бренд, модель, рік випуску, тощо), а значенням – відповідне значення параметра.
3. Кожен такий асоціативний словник зберігається у вигляді json-файла.

lxml - це потужний фреймворк для обробки XML та HTML на мові Python. Він є Pythonic обгорткою для бібліотек C libxml2 та libxslt, що поєднує швидкість та повноту функцій XML цих бібліотек з простотою використання рідного Python API.

Основні особливості lxml включають:

1. Обробка XML та HTML: lxml дозволяє легко обробляти XML та HTML документи.
2. XPath та XSLT підтримка: lxml підтримує XPath та XSLT, що дозволяє виконувати складні запити та перетворення.

3. Інтеграція з ElementTree API: lxml є в основному сумісним, але кращим за відомий ElementTree API.
4. Висока продуктивність та ефективність пам’яті: lxml є дуже швидким та ефективним за пам’яттю.

3.4 Опис процесів очищення та обробки даних

Оскільки дані про параметри автомобілів, які були отримані після виконання процесів скрапинга та парсинга даних є необробленим текстом, то ці дані потрібно очистити та обробити.

Це виконується за допомогою об’єднання збережених json-файлів з процеса парсинга даних в Pandas датафрейм.

Отриманий датафрейм передається на обробку трансформуючому пайплайну, який за допомогою трансформаторів перетворює неочищені дані в структуровані та очищені.

3.4.1 Опис трансформуючого пайплайна

1. Успадковується від базових класів BaseEstimator та TransformerMixin, підтримує методи fit та transform.
2. На кожному кроці пайплайн отримує вхідний датафрейм, перетворює його та передає далі на вхід наступному кроку пайплайна. Таким чином, відбувається переміщення датафрейма по трансформуючому пайплайну

Список трансформерів пайплайна:

1. Видалення рядків, в яких відсутня ціна та/або пробіг в кілометрах
2. Очистка зайвих символі \r \n \t в значеннях параметрів автомобіля

3. Заміна значень на NULL у випадках, коли в інформації про наявність авто в ДТП містить “Немає офіційно зареєстрованих” або в імені продавця вказане значення “Ім’я не вказане”.

4. Приведення ціни на автомобіля до значення єдиної валюти – доллар США, оскільки часто буває, що ціна в оголошенні вказана в гривні або євро. Конвертація відбувається через офіційни курс НБУ через використання API НБУ.

5. Проведення додаткового парсинга назви автомобіля на наявність бренда автомобіля, що може складатися з 2 слів. Приклад – Aston Martin.

6. Обробка булевих значень. Заміна значень “Ні” та “Так” на числовий булевий ідентифікатор (0 та 1 відповідно).

7. Обробка значень, що пов'язаних з місцем реєстрації автомобіля.

8. Обробка спарсеного значення моделі бренда автомобіля.

9. Явне приведення типів даних для параметрів автомобіля.

Список трансформаторів пайплайна в коді представлено на наступному рисунку:

```python
import pandas as pd
from sklearn.pipeline import Pipeline
from cleaning.transformers.price_transformer import PriceTransformer
from cleaning.transformers.car_name_transformer import ModelBrandYearExtractor
from cleaning.transformers.car_engine_info_transformer import EngineVolumePowerTypeTransformer
from cleaning.transformers.empty_null_values_handler import EmptyValuesHandler
from cleaning.transformers.boolean_handler import BooleanHandler
from cleaning.transformers.seller_city_handler import SellerCityHandler
from cleaning.transformers.empty_rows_cleaner import EmptyRowsCleaner
from cleaning.transformers.type_transformer import TypeTransformer
from cleaning.transformers.photos_number_extractor import PhotosNumberExtractor
from cleaning.transformers.extra_symbols_cleaner import ExtraSymbolsCleaner
from cleaning.transformers.car_model_transformer import CarModelTransformer


def transform(df: pd.DataFrame) -> pd.DataFrame:
    """The function that transforms the raw\
    dataframe by the help of transformation pipeline

    Args:
        df (pd.DataFrame): input raw dataframe

    Returns:
        pd.DataFrame: output transformed dataframe
    """
    pipeline = Pipeline(
        steps=[
            ("empty_rows_cleaner", EmptyRowsCleaner()),
            ("extra_symbols_cleaner", ExtraSymbolsCleaner()),
            ("empty_null_values_handler", EmptyValuesHandler()),
            ("price_transformer", PriceTransformer()),
            ("model_brand_year_extractor", ModelBrandYearExtractor()),
            ("car_engine_info_transformer", EngineVolumePowerTypeTransformer()),
            ("photos_number_extractor", PhotosNumberExtractor()),
            ("boolean_handler", BooleanHandler()),
            ("seller_city_handler", SellerCityHandler()),
            ("car_model_transformer", CarModelTransformer()),
            ("type_transformer", TypeTransformer()),
        ]
    )
    return pipeline.fit_transform(df)
```

Рис. 3.2. Структура транформуючого пайплайна

3.5 Опис процесів роботи з базою даних

Серед особливостей роботи системи з базою даних варто зазначити, що СУБД є реляційною – MS SQL Server, використовується хмарна база даних в Azure. В ПЗ використовується SQLAlchemy – Python SQL toolkit. В якості коннектора до бази даних використовується pyodbc. Робота з об'єктами бази даних відбувається за допомогою використання ORM структури

```python
from sqlalchemy import ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String, Float, SmallInteger, Date

Base = declarative_base()


class Car(Base):
    """ORM class for the table 'cars' in db

    Args:
        Base: sqalchemy's base class for ORM classes
    """

    __tablename__ = "cars"
    car_unique_id = Column(Integer, primary_key=True, autoincrement=False)
    car_brand = Column(String, nullable=False)
    car_model = Column(String, nullable=False)
    car_specification = Column(String, nullable=True)
    car_year = Column(SmallInteger, nullable=False)
    car_mileage_thousands_km = Column(Float, nullable=False)
    car_type = Column(String, nullable=False)
    car_engine_volume_liters = Column(Float, nullable=True)
    car_engine_type = Column(String, nullable=True)
    car_engine_energy_power_kWh = Column(Float, nullable=True)
    car_model_rating = Column(Float, nullable=True)
    car_gearbox = Column(String, nullable=True)
    car_drive = Column(String, nullable=True)
    car_color = Column(String, nullable=True)
    car_description = Column(String, nullable=True)
    car_state = Column(String, nullable=True)
    car_wanted = Column(SmallInteger, nullable=True)
    car_had_accidents = Column(SmallInteger, nullable=True)
    car_vin_code_verified = Column(SmallInteger, nullable=True)
    car_photos_number = Column(SmallInteger, nullable=True)
    car_is_sold = Column(SmallInteger, nullable=True)
    seller_type = Column(String, nullable=True)
    seller_name = Column(String, nullable=True)
    seller_city = Column(String, nullable=True)
```

Рис. 3.3. Опис структури ORM для таблиці cars

```python
class Ad(Base):
    """ORM class for the table 'ads' in db

    Args:
        Base: sqalchemy's base class for ORM classes
    """

    __tablename__ = "ads"
    car_unique_id = Column(
        Integer, ForeignKey("cars.car_unique_id"), primary_key=True, autoincrement=False
    )
    ad_is_deleted = Column(SmallInteger, nullable=True)


class SellerPhoneNumber(Base):
    """ORM class for the table 'sellers_phone_numbers' in db

    Args:
        Base: sqalchemy's base class for ORM classes
    """

    __tablename__ = "sellers_phone_numbers"
    id = Column(Integer, primary_key=True, autoincrement=True)
    car_unique_id = Column(Integer, ForeignKey("cars.car_unique_id"))
    seller_phone_number = Column(String, nullable=True)


class Timestamp(Base):
    """ORM class for the table 'timestamps' in db

    Args:
        Base: sqalchemy's base class for ORM classes
    """

    __tablename__ = "timestamps"
    car_unique_id = Column(
        Integer, ForeignKey("cars.car_unique_id"), primary_key=True, autoincrement=False
    )
    car_price_usd = Column(Integer, primary_key=True, autoincrement=False)
    timestamp = Column(Date, primary_key=True, autoincrement=False)
```

Рис. 3.4. Опис структури ORM для таблиць ads, timestamps, seller_phone_numbers

Основні характеристики SQLAlchemy включають:

1. ORM: Побудований на основі картки ідентичності, одиниці роботи та шаблонів дата-мапера. Ці шаблони дозволяють прозоро зберігати об’єкти за допомогою декларативної системи конфігурації.
2. Система запитів, орієнтована на відносини: Вона відкриває повний діапазон можливостей SQL, включаючи об’єднання, підзапити, кореляцію та більшість іншого, в термінах об’єктної моделі.

3. Система жадного завантаження для пов’язаних колекцій та об’єктів: Колекції кешуються в межах сесії і можуть бути завантажені при окремому доступі, все одночасно за допомогою об’єднань, або за допомогою запиту на колекцію по всьому набору результатів.
4. Система побудови SQL Core та взаємодії з DBAPI: SQLAlchemy Core - це окремий від ORM і є повним шаром абстракції бази даних, який включає в себе гнучку мову виразів SQL на основі Python, метадані схеми, пул з’єднань, приведення типів та користувацькі типи.
5. Інтроспекція та генерація бази даних: Схеми бази даних можуть бути “відображені” за один крок у структури Python, що представляють метадані бази даних.

3.5.1 Опис особливостей роботи веб-ресурса з оголошеннями

1. Оголошення можуть додаватися, видалятися чи оновлюватися на веб-ресурсі.
2. В оголошенні можуть оновлюватися окремо різні дані (ціна, фотографії, опис, тощо)
3. Коли оголошення видаляється, то воно може бути видаленим самим власником оголошення (коли автомобіль вже є проданим чи за власним бажанням власника оголошення).
4. Веб-ресурс з оголошеннями має можливість самостійно модерувати та видаляти оголошення з сильно заниженими цінами відносно ринкових цін.
5. Після продажу автомобіля оголошення видаляється, проте власник оголошення може відмінити факт продажу авто (наприклад якщо фактичний продаж авто зірвався). В такому випадку оголошення про продаж знову стане активним. З врахуванням перелічених вище особливостей роботи веб-ресурса пропонується наступна структура бази даних:

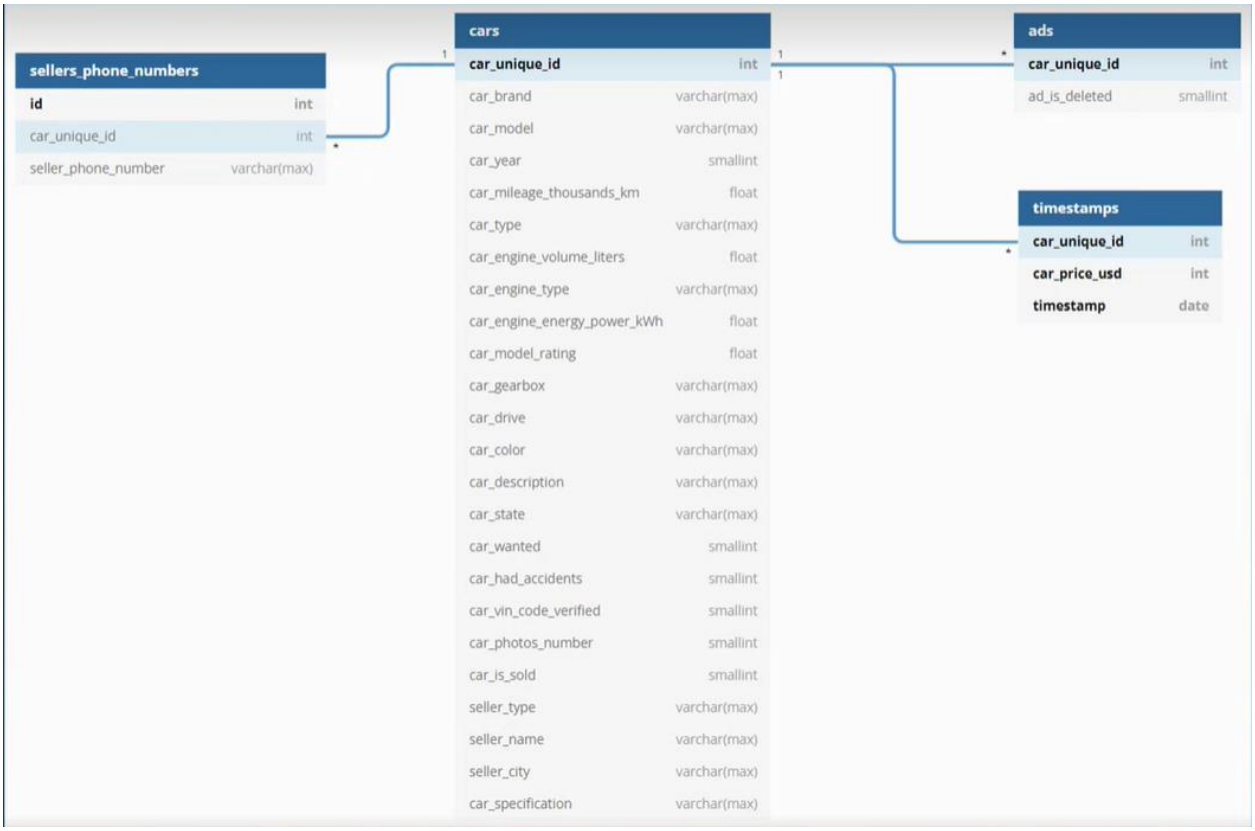


Рис. 3.5. Структура бази даних програмного забезпечення

На даному рисунку зображено 4 таблиці:

1. Таблиця cars – таблиця є основною, містить усі параметри автомобіля, про які є інформація в оголошенні.
2. Таблиця ads – містить інформацію про ідентифікатор автомобіля та чи є його оголошення видаленим.
3. Таблиця timestamps – відображає динаміку зміни ціни на автомобіль в оголошенні з плином часу.
4. Таблиця sellers_phone_numbers – містить інформацію про мобільні номера, що прив’язані до оголошення про продаж автомобіля. Не є обов’язковою для заповнення та використання в моделі машинного навчання для прогнозування ціни.

Існує певний список правил, по яким змінюється зміст даних в цих таблицях:

1. Коли додаються нові оголошення на веб-ресурс – додається інформація до всіх 4 таблиць.
2. Коли змінюється статус про видалення оголошення:
   a. Коли оголошення видаляється – таблиця ads оновлюється
   b. Коли оголошення знову є активним - таблиця ads оновлюється і таблиця timestamps наповнюється новими значеннями.
3. Коли змінилася ціна на автомобіль в оголошенні - таблиця timestamps наповнюється новими значеннями

3.6 Проведення EDA (Exploratiry Data Analysis)

Було отримано дані про 289 тисяч автомобілів. Проведений EDA на цих даних та отримані наступні результати:

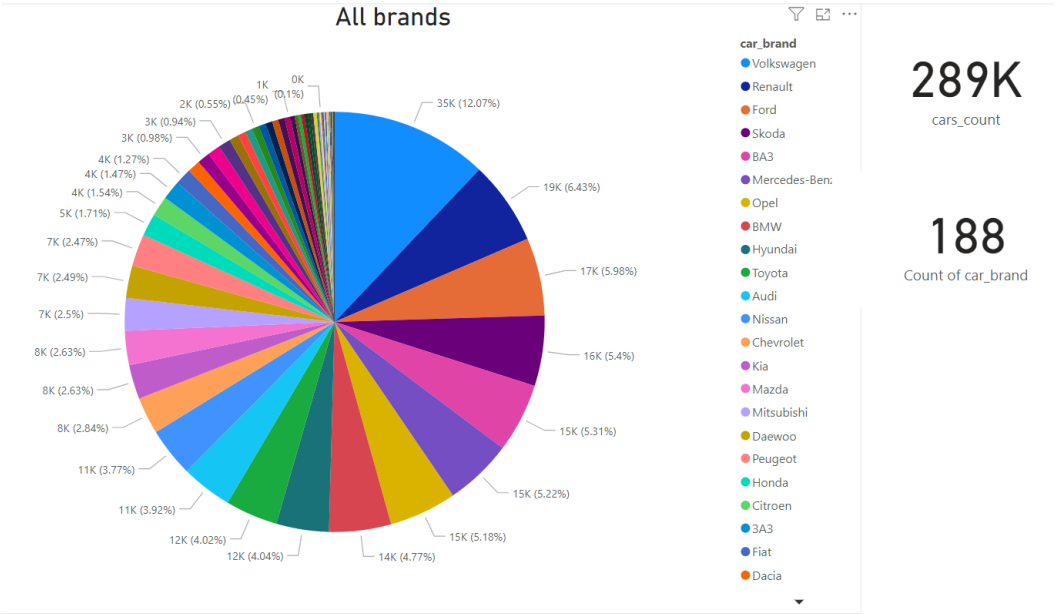


Рис. 3.6. Розподіл кількості автомобілей за належністю до бренда

Як можна побачити, маємо велику кількість різних брендів автомобілей (188), проте лише 18 з них мають досить велику кількість автомобілей, що представлені на веб-ресурсі:

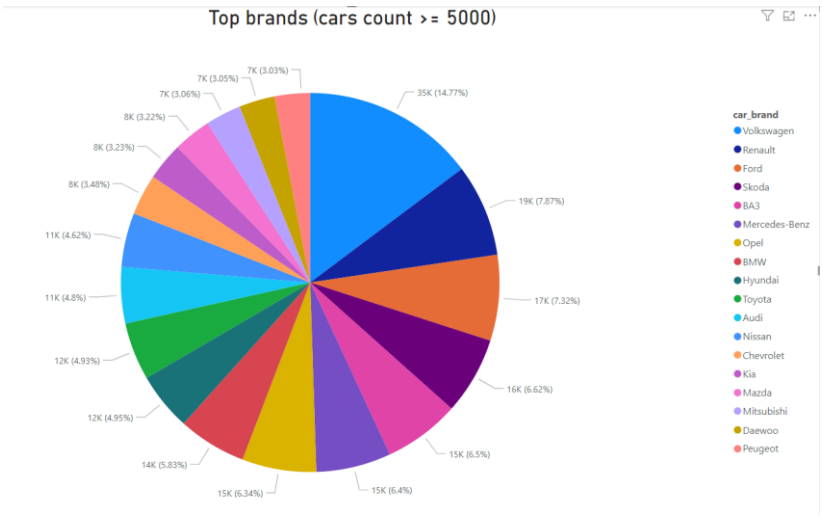


Рис. 3.7. Розподіл кількості автомобілей за топ-брендами

На наступному графіку представлений розподіл кількості унікальних моделей для кожного з брендів автомобілів:

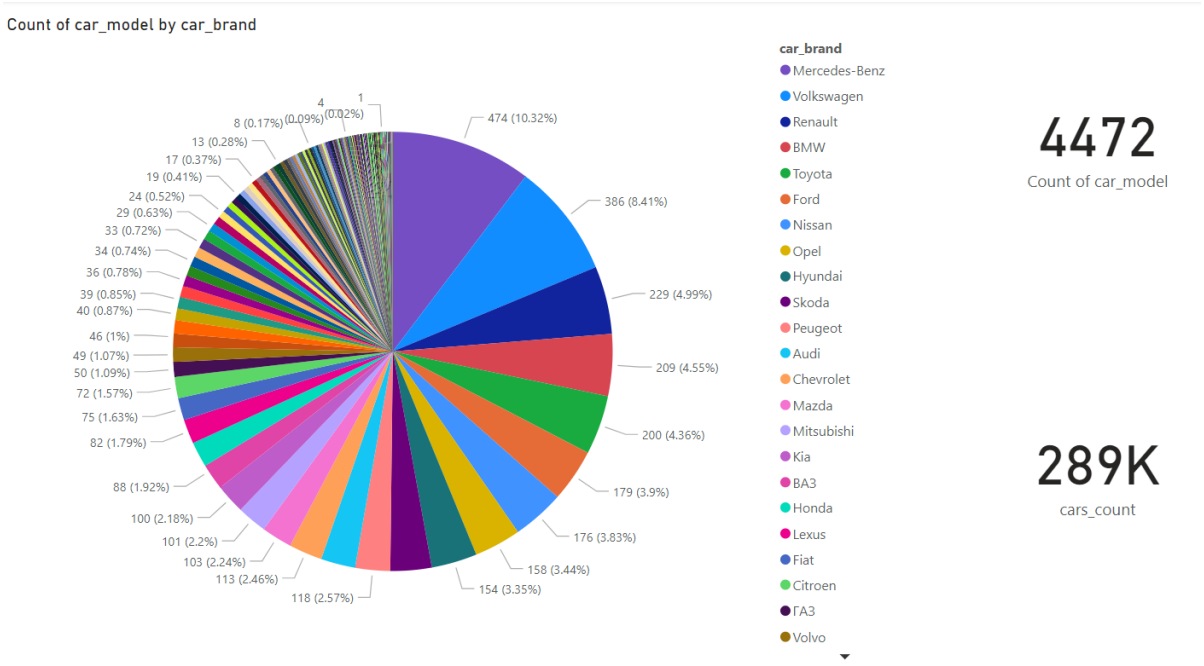


Рис. 3.8. Розподіл кількості унікальних моделей для брендів автомобілів

На наступному графіку представлена інформація про розподіл кількості автомобілів за брендом та моделлю:

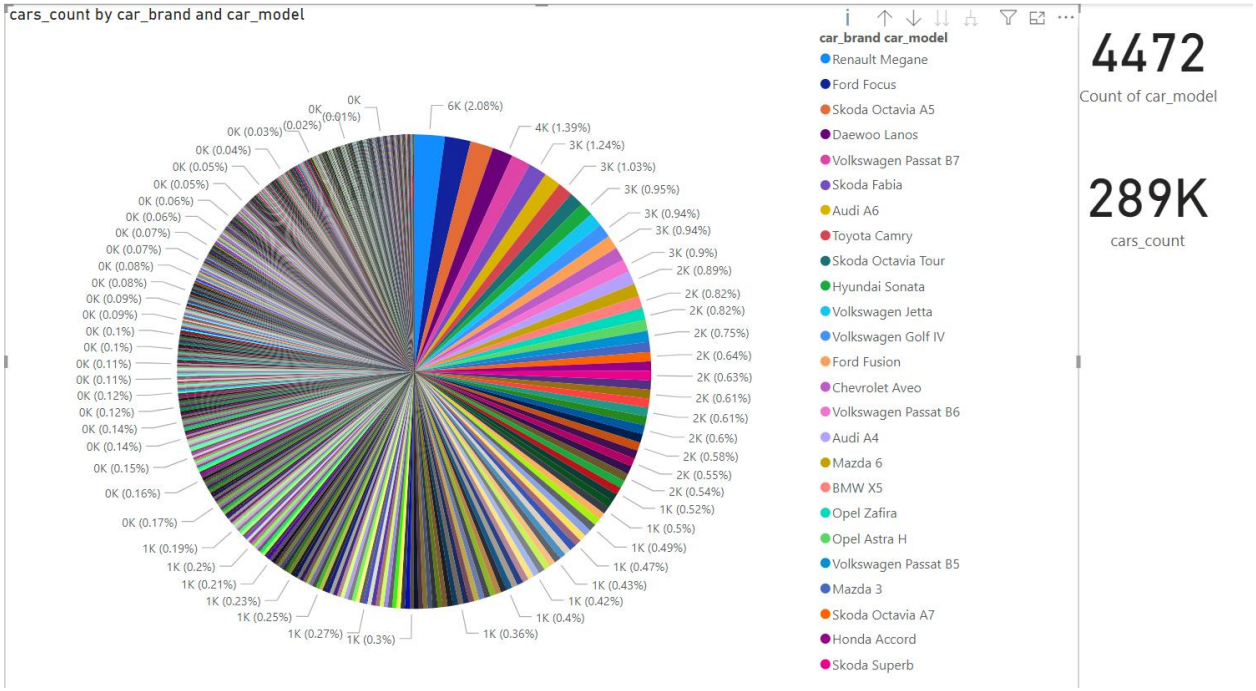


Рис. 3.9 Розподіл кількості автомобілів за брендом та моделлю

Можна побачити, що найпопулярнішою моделлю автомобіля, що виставляється на продажу, є Renault Megane. На наступному графіку можна побачити розподіл кількості автомобілів за роком випуску:

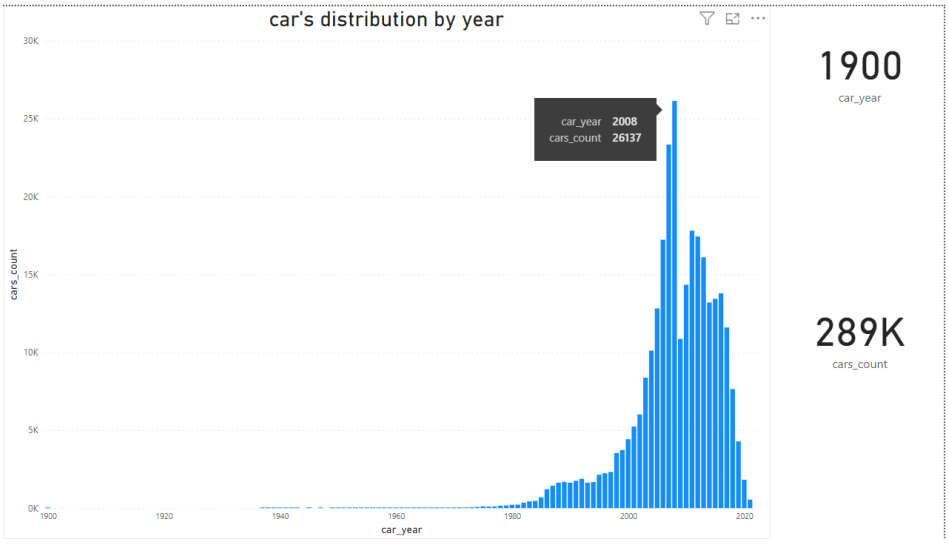


Рис. 3.10. Розподіл кількості автомобілів за роком випуску

Що цікаво – можна помітити значний спад в кількості вироблених авто в 2009 році, це пов'язано зі світовою економічною кризою 2008 року. На наступному графіку

зображений розподіл кількості машин за пройденим ними кілометражем:

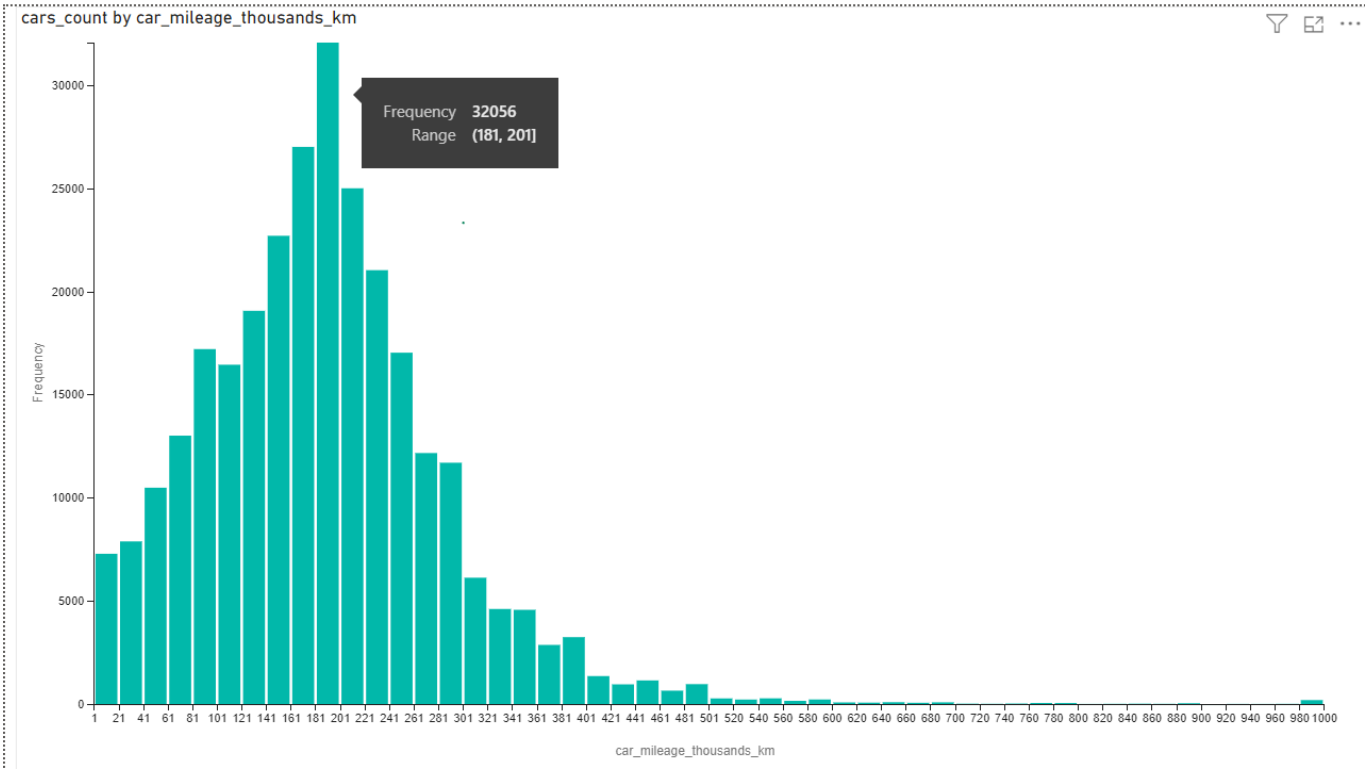


Рис. 3.11. Розподіл кількості автомобілів за пройденим ними кілометражем

Можна побачити аномальні значення між 980 000 та 1 000 000 км, що є аномальним значенням для автомобіля, яке не слід враховувати при навчанні моделі машинного навчання.

На наступному графіку можна побачити розподіл кількості автомобілей за середньою оцінкою моделі від користувача порталу:

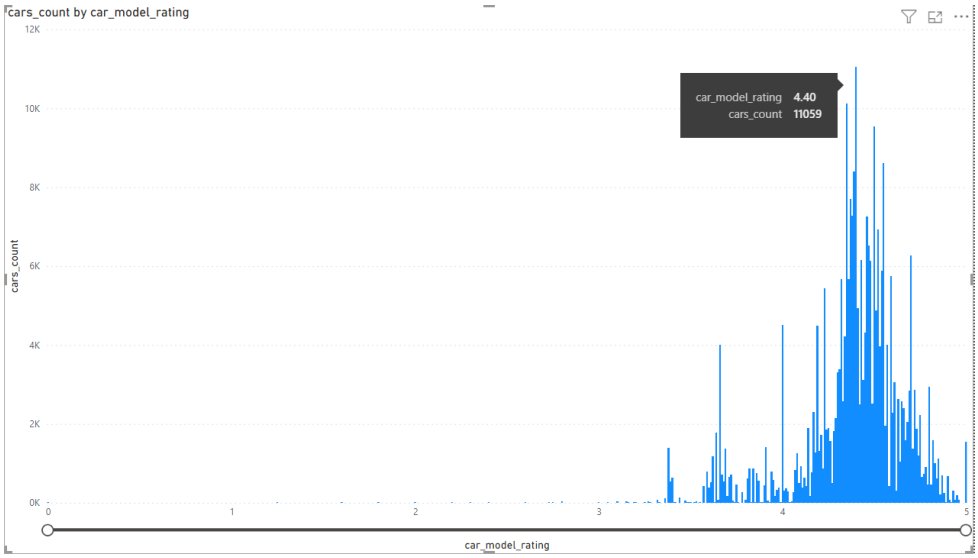


Рис. 3.12. Розподіл кількості автомобілей за середньою оцінкою моделі від користувача порталу

Можна побачити, що найчастіша оцінка, яка ставиться моделі автомобіля – 4.4/5. На наступному графіку можна побачити розподіл оцінок моделей по кожному з брендів автомобілей:

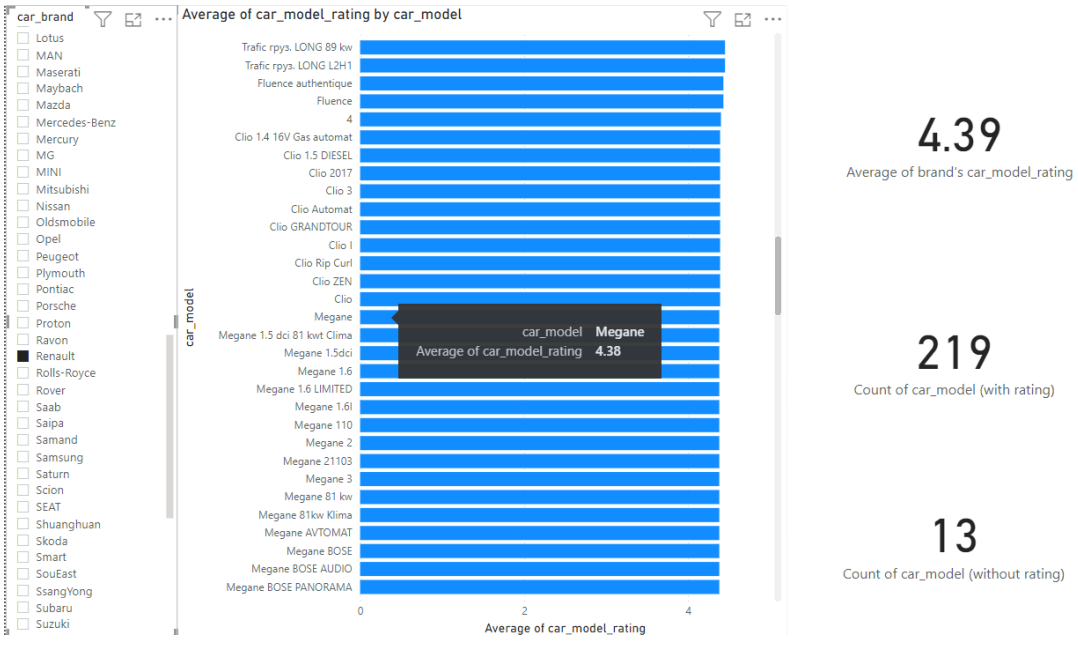


Рис. 3.13. Розподіл оцінок моделей по кожному з брендів автомобілей

На наступному графіку представлений розподіл кількості автомобілів за типом коробки передачі:

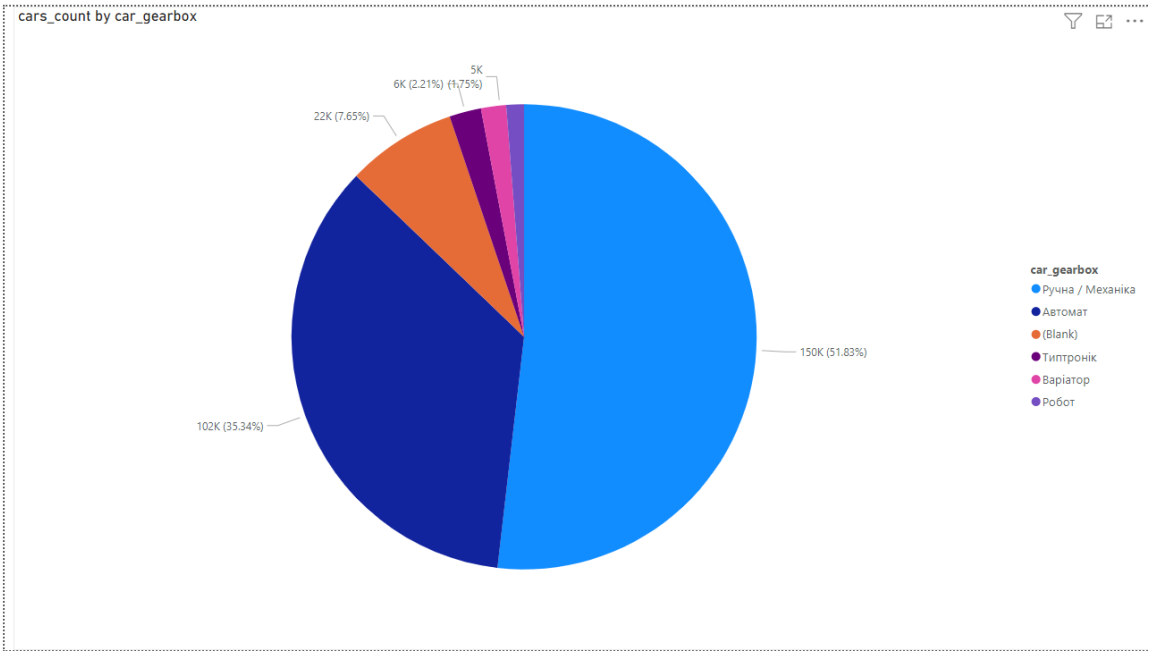


Рис. 3.14. Розподіл кількості автомобілів за типом коробки передачі

На цьому графіці цікаво виглядає той факт, що більше половини усіх представлених на веб-ресурсі автомобілей мають ручну коробку передач. Можна зробити припущення, що скоріше за все це є недорогі автомобілі. На наступному графіку демонструється розподіл кількості автомобілів за ціною:

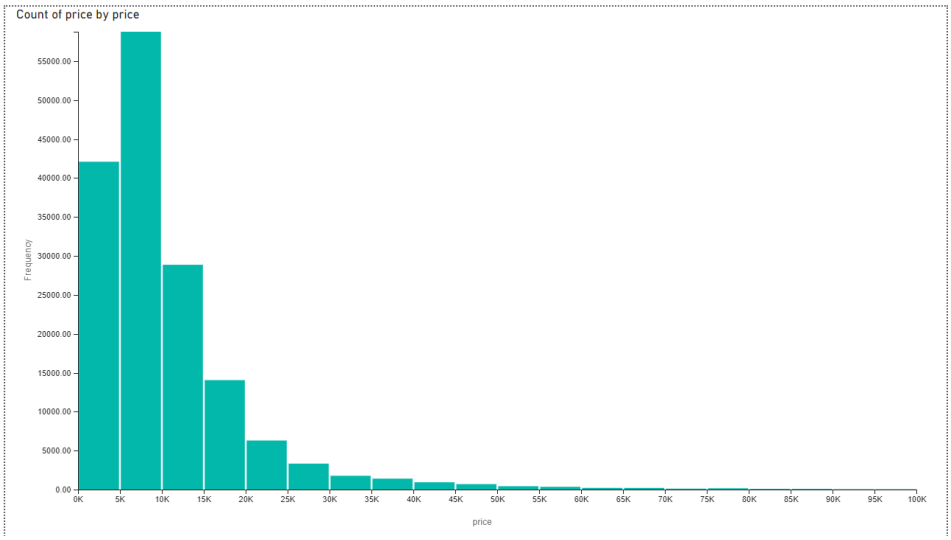


Рис. 3.15. Розподіл кількості автомобілів за ціною

Дійсно підтверджується гіпотеза про те, що більшість машин на ресурсі є недорогими (коштують менше 10000$). На наступному графіку зображений розподіл кількості машин за типом привода:

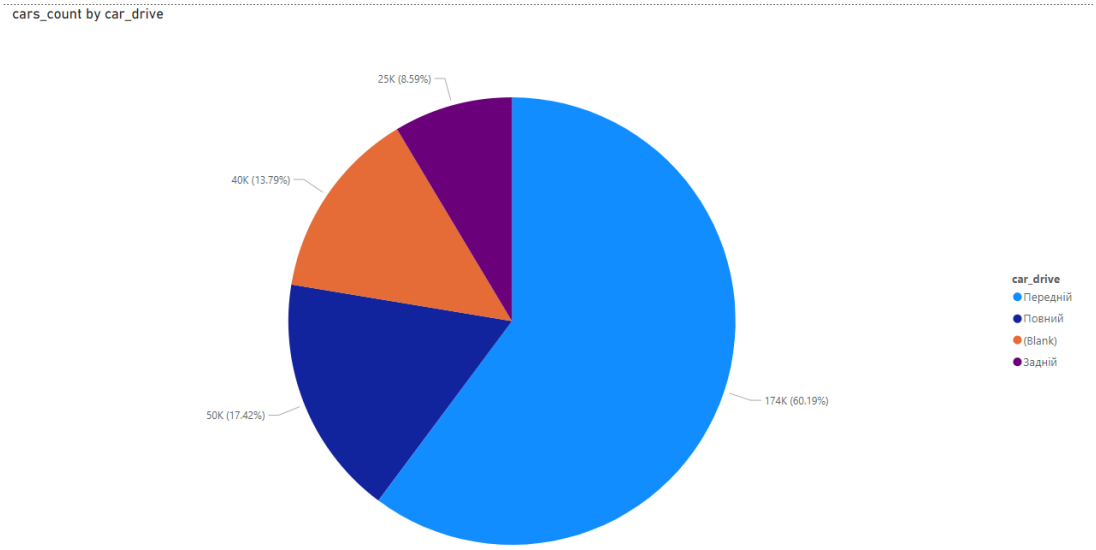


Рис. 3.16. Розподіл кількості автомобілів за типом привода

Цікаво, що основна частина автомобілів має передній тип привода. На наступному графіці зображений розподіл кількості автомобілей за їхнім кольором:

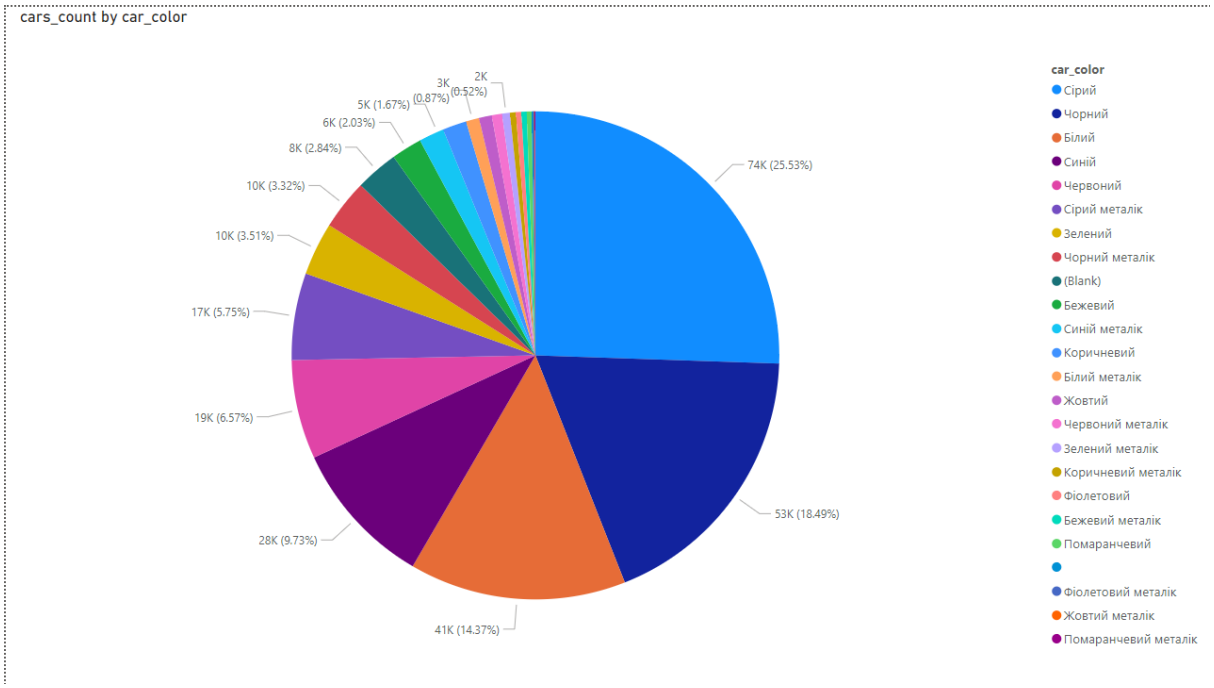


Рис. 3.17. Розподіл кількості автомобілів за кольором

Цікаво те, що більшість автомобілей мають чорний, білий чи сірий кольори. Усі інші кольори для авто є досить рідкісними і потенційно можуть надавати машині ексклюзивності та підвищувати її ціну.

На наступному графіку зображений розподіл кількості автомобілей за станом:

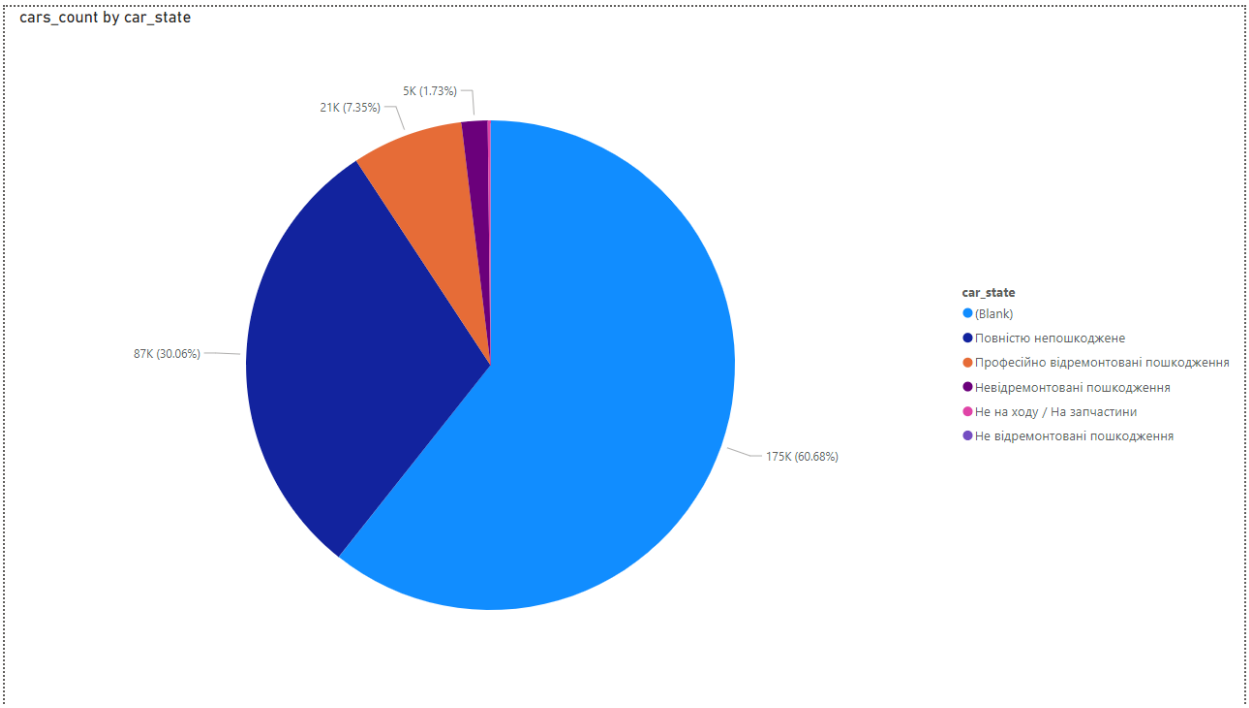


Рис. 3.18. Розподіл кількості автомобілів за станом

Цікаво, що основна частина продавців надає перевагу не вказувати інформацію про стан машини. Це свідчить про те, що продавці досить уважно ставляться до цієї характеристики автомобіля та не хочуть явно вказувати її, сподіваючись, що при особистій зустрічі з потенційним продавцем вдасться видати стан авто за кращий, ніж він є. На наступному графіку зображений розподіл кількості машин за наявністю в них ДТП:

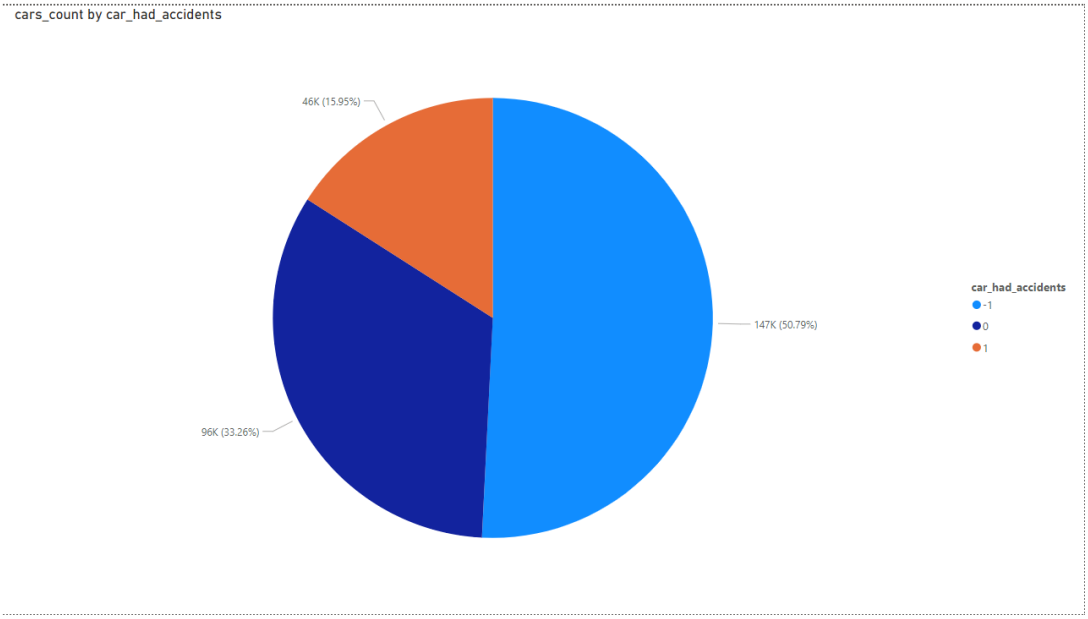


Рис. 3.19. Розподіл кількості автомобілів за наявністю ДТП

Тут можна помітити аналогію зі станом авто, коли продавці надають перевагу не вказувати інформацію про цю характеристику автомобіля. І це є досить логічним, оскільки наявність машини (хоч і в минулому) в ДТП знижує її вартість.

На наступному графіку зображений розподіл автомобілей за наявністю підтвердженого vin-коду (державного номера):

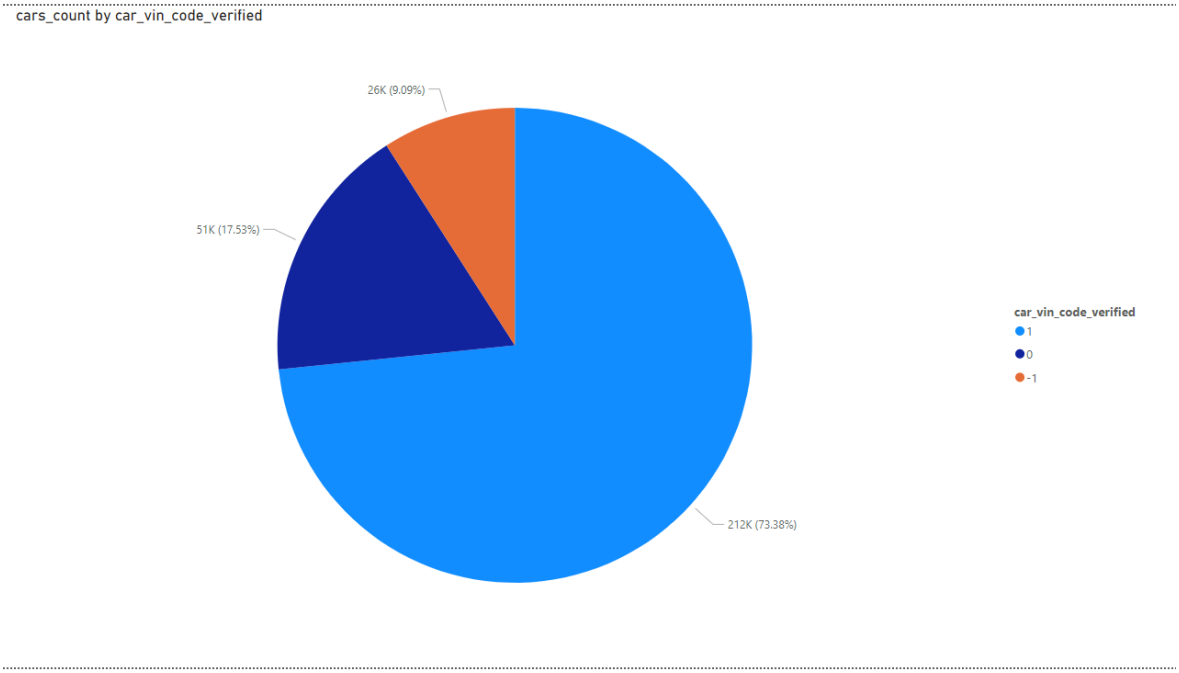


Рис. 3.20. Розподіл автомобілей за наявністю підтвердженого vin-коду

Цікаво, що майже 3 четверті автомобілей на веб-ресурсі мають підтверджений vin-номер, що означає, що на цьому веб-ресурсі достатньо виважено та прискіпливо ставляться до перевірки автомобілей, що виставляються на продаж.

На наступному графіку зображений розподіл кількості автомобілей за кількістю фотографій в оголошенні:

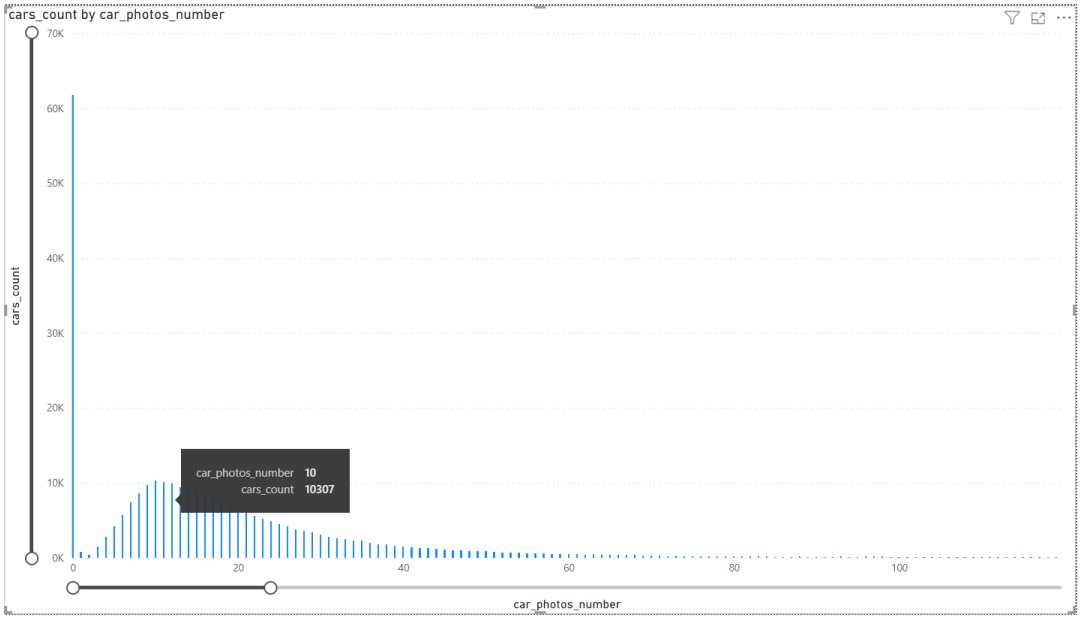


Рис. 3.21. Розподіл автомобілей за наявністю підтвердженого vin-коду

Цей графік чітко демонструє особливості платформи, на якій деактивовані оголошення перестають відображати фотографії автомобіля. А найчастіше продавці вказують саме 10 фотографій автомобіля.

На наступному графіку зображений розподіл кількості автомобілей за станом продажу:

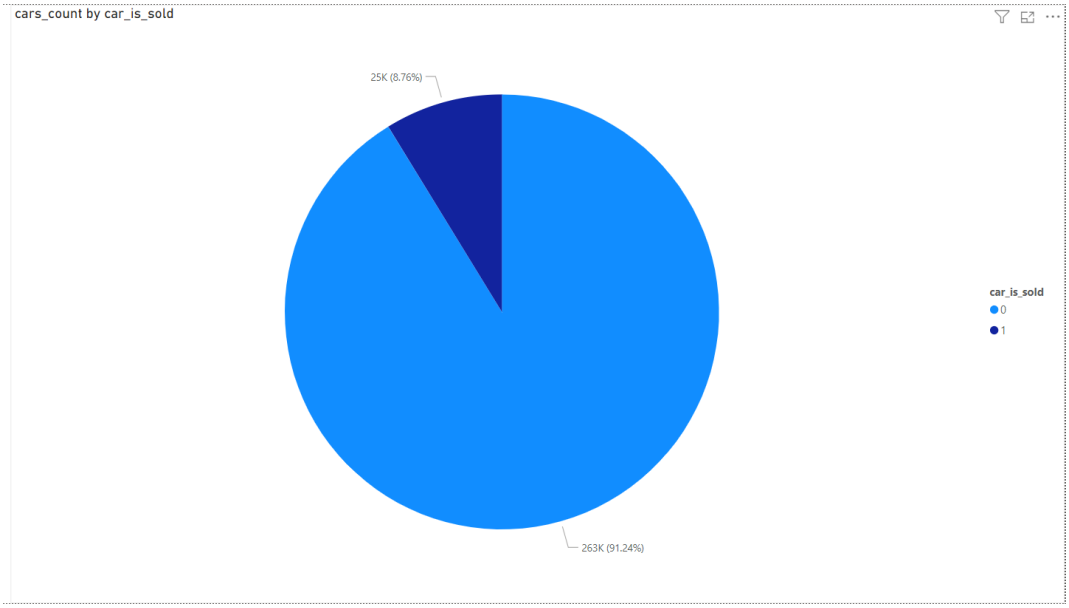


Рис. 3.22. Розподіл кількості автомобілей за станом продажу

На цьому графіку видно, що менше 9% автомобілей вже є проданими. Тобто основна частина оголошень на ресурсі є актуальними.

На наступному графіку зображений розподіл кількості машин за типом продавця:

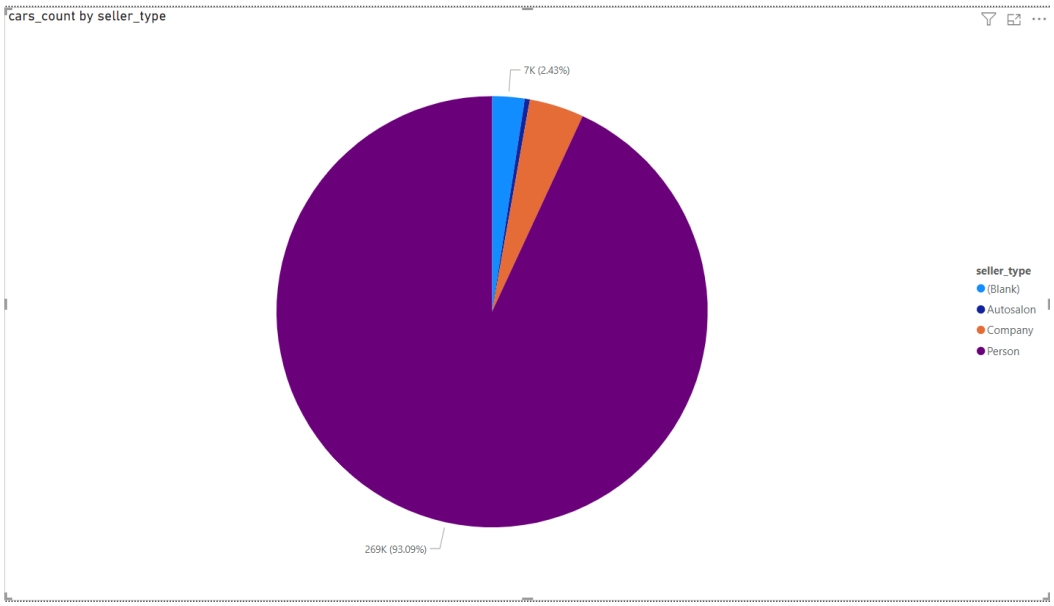


Рис. 3.23. Розподіл кількості машин за типом продавця

Можна побачити, що більше 93% усіх продавців авто – саме фізичні особи, і лише 4% - автосалони.

На наступному графіку показана мапа населених пунктів, де зареєстровані автомобілі:

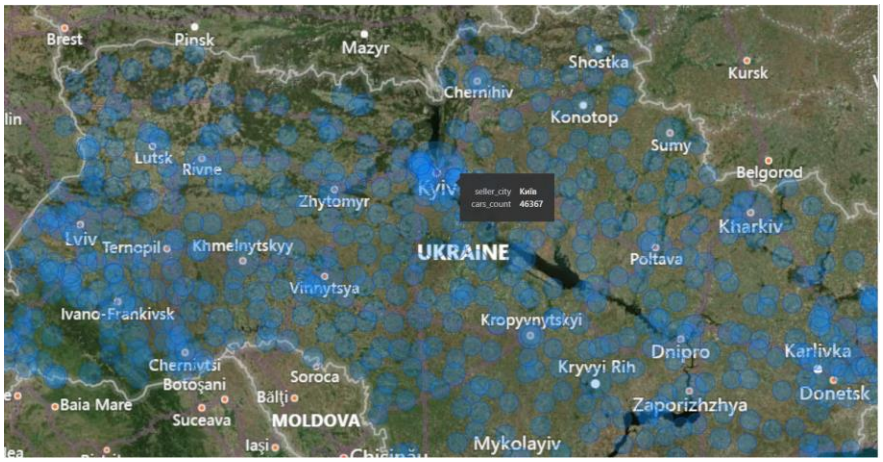


Рис. 3.24. Мапа населених пунктів, де зареєстровані автомобілі

Можна помітити, що найбільше машин зареєстровані в місті Київ, проте це не дивно, враховуючи кількість населення в цьому місті, що має найбільший показник населення в Україні.

На наступному графіку показано розподіл кількості автомобілів за специфікацією (субмоделлю всередині моделі бренда):

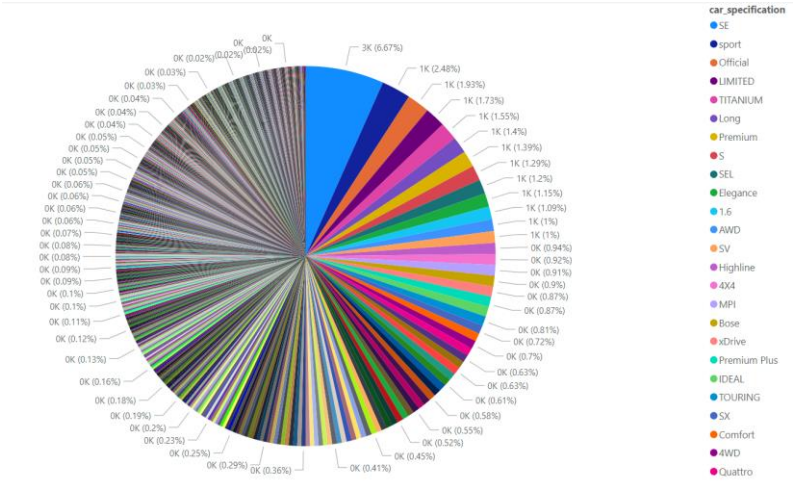


Рис. 3.25. Розподіл кількості автомобілів за специфікацією

Як можна побачити, найпоширенішою специфікацією автомобіля є SE.

Таким чином, проведений EDA по характеристикам автомобілів. Були видвинуті певні гіпотези та знайдені аномальні значення, що буде враховано при побудові тренувального датасета для моделі LightGBM.

3.6 Опис формування тренувального датасета

Тренувальний датасет формується з таблиці cars бази даних. Проходить процес розділення даних з цієї таблиці на фактори (параметри автомобіля), та цільова колонка у вигляді ціни автомобіля, яке потрібно запрогнозувати. Приклад даних з таблиці cars:

cars

MagicPython

	car_brand	car_model	car_year	car_mileage_thousands_km	car_type	car_engine_volume_liters	car_engine_type	car_engine_energy_power_kWh	car_model_rating	car_gearbox	car_drive	car_color	car_description	car_state	car_want
0	Nissan	X-Trail	2003	215.0	Позашляховик / Кросовер	2.00	Бензин	NaN	4.31	Ручна / Механіка	Повний	NaN	Відмінний стан, не битий і не крашений, один г...	Повністю непошкоджене	
1	Renault	25	1985	330.0	Хетчбек	NaN	Бензин	NaN	4.47	Ручна / Механіка	NaN	Білий	Сигналізація, центральний замок, тонировка. ...	NaN	
2	Chevrolet	Camaro	1993	180.0	Купе	3.40	Бензин	NaN	4.37	Ручна / Механіка	Задній	Сірий	Продам автомобиль, \nк нему есть две пары А...	NaN	
3	Mercedes-Benz	Atego 1218	1990	268.0	Універсал	2.50	Дизель	NaN	NaN	Ручна / Механіка	NaN	Сірий	мерседес-124 універсал-В 2, 5д ABS КПП-5 хутро...	NaN	
4	ГАЗ	69	1963	1.0	Позашляховик / Кросовер	2.12	Бензин	NaN	4.33	Ручна / Механіка	NaN	Зелений	Підсилювач тормозів, сидіння "Опель&...	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
299340	Skoda	Octavia A5	2005	247.0	Хетчбек	1.60	Газ / Бензин	NaN	4.56	NaN	NaN	Бежевий	NaN	NaN	
299341	Ford	Mondeo	1993	309.0	Седан	1.80	Газ / Бензин	NaN	4.39	Ручна / Механіка	Передній	Червоний	Надійний та не прихотливий автомобіль. Заміна м...	NaN	
299342	Peugeot	207	2008	210.0	Універсал	1.60	Газ / Бензин	NaN	4.39	Ручна / Механіка	NaN	NaN	услуги хорошего ухоженный лыжика, не бит не кр...	NaN	
299343	SsangYong	Actyon Sports	2008	229.0	Пікап	2.00	Дизель	107.0	4.24	Автомат	Повний	Сірий	Продам Ssangyong actyon sports в очень хорошем...	Повністю непошкоджене	
299344	Land Rover	Discovery	2019	29.0	Універсал	3.00	Бензин	NaN	4.40	Автомат	NaN	Білий	Універсальний автомобіль для міста і далеких п...	NaN	

299345 rows × 24 columns

Рис. 3.26. Приклад даних з таблиці cars

Для тренувального датасета обирається 90% від даних цієї таблиці. Для моделювання обирається ті автомобілі, які мають менше мільйона кілометрів пробіга, менше 7 літрів двигун та менше 736 кіловатт потужності двигуна (що еквівалентно 1000 кінських сил).

В якості метрики прогнозування була обрана метрика MAPE (3.1):

$$\text{MAPE} = \left(\frac{1}{n}\right) \cdot \sum\left(\left| \frac{\text{Факт} - \text{Прогноз}}{\text{Факт}} \right|\right) \cdot 100, \qquad (3.1)$$

де:

- n - розмір вибірки;
- Факт - фактичне значення даних;
- Прогноз - прогнозоване значення даних

Обгрунтування вибору цієї метрики:

1. Великий розмах в ціні між різними автомобілями
2. Відсутність нульових чи близьконульових значень.

Був отриманий наступний графік розподілу впливу факторів моделі:

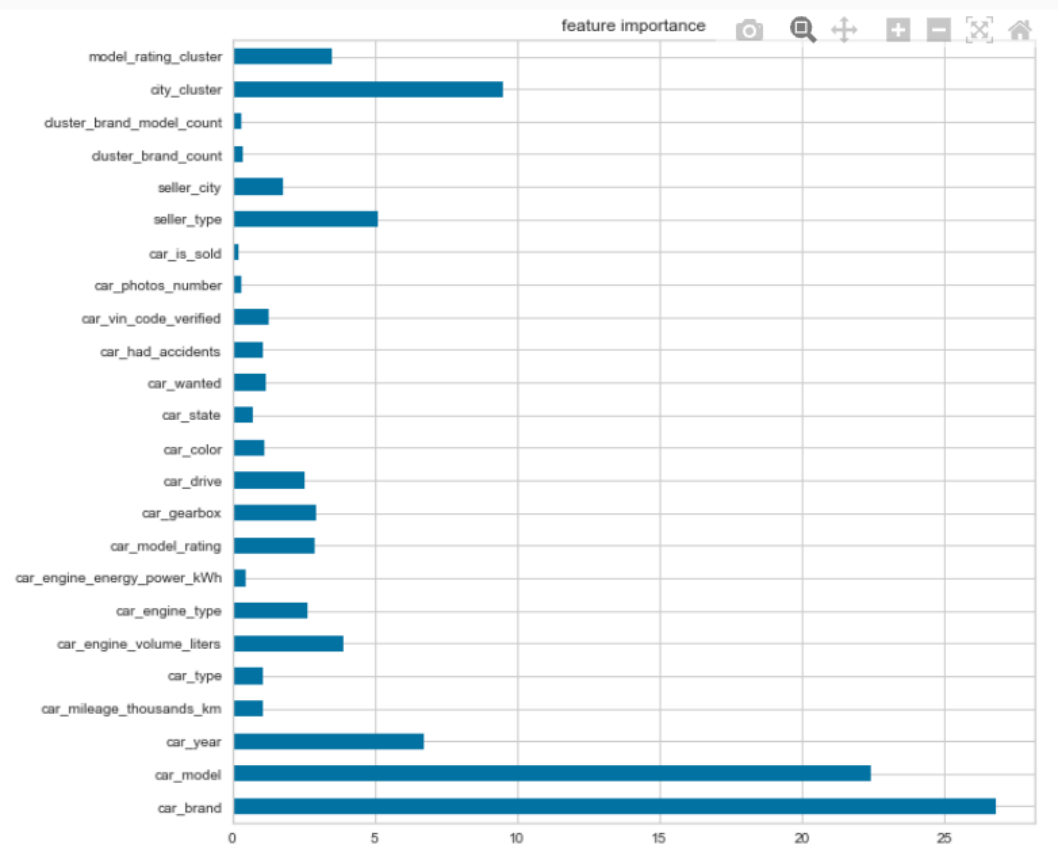


Рис. 3.27. Розподіл впливу факторів на формування прогноза моделі

Як можна побачити, найголовнішими факторами для моделі виявилися бренд, модель, рік випуску та кластер місця реєстрації автомобіля. В цілому модель точно виділила основні фактори, які впливають на прогнозування ціни продажу автомобіля.

В якості оркестратора для моделей був обраний фреймворк MLFlow.

MLflow - це відкритий фреймворк, призначений для управління життєвим циклом моделей машинного навчання. Він включає в себе декілька ключових компонентів:

1. MLflow Tracking: Це система, яка дозволяє записувати та запитувати експерименти: код, дані, конфігурацію та результати.
2. MLflow Projects: Це формат для пакування коду наукових даних, що дозволяє відтворювати запуски на будь-якій платформі.
3. MLflow Models: Це дозволяє розгортати моделі машинного навчання в різних середовищах обслуговування.
4. Model Registry: Це дозволяє зберігати, анотувати, виявляти та керувати моделями в центральному репозиторії.

MLflow дозволяє керувати всім процесом машинного навчання, включаючи експерименти, відтворюваність, розгортання та центральний реєстр моделей. Це допомагає забезпечити прозорість, відтворюваність та співпрацю в процесі розробки моделей машинного навчання.

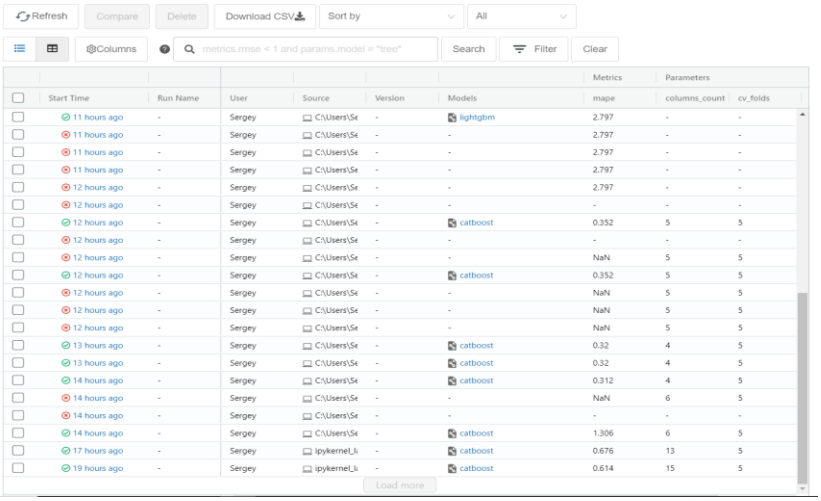


Рис. 3.28. Приклад використання MLFlow в якості оркестратора для моделі

3.7 Порівняння результатів прогнозування методів

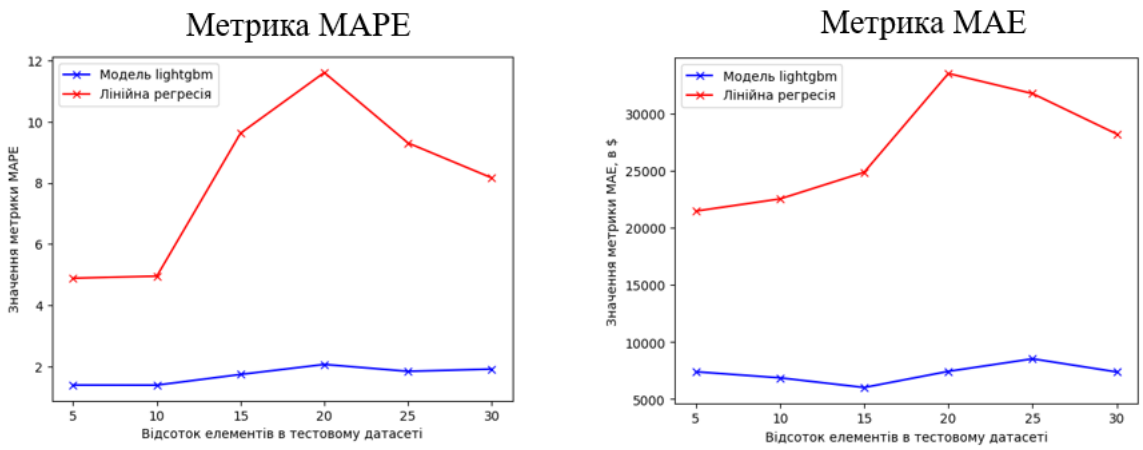


Рис. 3.29. Порівняння результатів прогнозування

Таким чином, маємо зменшення метрики MAPE в середньому в 4.7 рази, а метрики MAE - в 3.72 рази при використанні моделі машинного навчання lightgbm regressor в порівнянні з методом лінійної регресії для прогнозування ціни автомобіля.

ВИСНОВКИ

1. Проведено огляд існуючих методів прогнозування ціни на автомобілі, виділені їхні основні переваги та недоліки. Проаналізовані підходи по використанню методів машинного навчання в предметній області прогнозування ціни автомобіля. Проаналізовано основні особливості моделі машинного навчання LightGBM. Описана математична модель системи прогнозування ціни автомобіля.
2. Створена архітектура системи прогнозування ціни автомобіля. Наведений опис взаємодії складових частин системи прогнозування ціни автомобіля. Описана архітектура фреймворків lxml та scrapy, що використовуються в парсингу та скрапингу даних. Наведені особливості роботи веб-ресурса з оголошеннями про продаж автомобілей.
3. Розроблено програмне забезпеченням, яке за допомогою моделі машинного навчання lightgbm regressor здатне підвищити ефективність прогнозування ціни на автомобіль. Описаний процес виконання скрапинга за допомогою використання бібліотеки Scrapy. Наведена детальна структура трансформуючого пайплайна, що виконує обробку даних. Надано опис структури бази даних, що використовує система для прогнозування ціни автомобіля.
4. Описана метрика та обґрунтований її вибір для порівняння результатів прогнозування системи. Наведена демонстрація результатів прогнозування ціни автомобіля, що показує підвищення ефективності прогнозування ціни автомобіля за допомогою машинного навчання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Pal, N., Arora, P., Kohli, P., Sundararaman, D., Palakurthy, S.S.: How much is my car worth? A methodology for predicting used cars' prices using random forest. In: Future of Information and Communication Conference, pp. 413–422. Springer, Cham (2018)
2. Gegic, E., Isakovic, B., Keco, D., Masetic, Z., Kevric, J.: Car price prediction using machine learning techniques. TEM J. 8(1), 113 (2019)
3. CS 229 Project Report: Predicting Used Car Prices [Електронний ресурс] – https://cs229.stanford.edu/proj2019aut/data/assignment_308832_raw/26612934.pdf
4. K. Noor and S. Jan, "Vehicle price prediction system using machine learning techniques", *International Journal of Computer Applications*
5. Object detection and used car price predicting analysis system (UCPAS) using machine learning technique[Електронний ресурс] – https://www.researchgate.net/publication/355906327_Object_detection_and_used_car_price_predicting_analysis_system_UCPAS_using_machine_learning_technique
6. Performance Evaluation of Popular Machine Learning Models for Used Car Price Prediction [Електронний ресурс] – https://www.researchgate.net/publication/372616822_Performance_Evaluation_of_Popular_Machine_Learning_Models_for_Used_Car_Price_Prediction
7. Used Car Price Prediction using K-Nearest Neighbor Based Model [Електронний ресурс] – https://www.researchgate.net/publication/350094568_Used_Car_Price_Prediction_using_K-Nearest_Neighbor_Based_Model
8. N. Sun, H. Bai, Y. Geng and H. Shi, "Price evaluation model in second-hand car system based on BP neural network theory," 2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Kanazawa, Japan, 2017, pp. 431-436, doi: 10.1109/SNPD.2017.8022758.

9. Barlybayev, A., Sankibayev, A., Kadyr, Y., Amangeldy, N., Sabyrov, T. (2023). Predicting used-vehicle resale value in developing markets: Application of machine learning models to the Kazakhstan car market. Ingénierie des Systèmes d'Information, Vol. 28, No. 5, pp. 1237-1246.
10. C. V. Narayana, C. L. Likhitha, S. Bademiya and K. Kusumanjali, "Machine Learning Techniques To Predict The Price Of Used Cars: Predictive Analytics in Retail Business", *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pp. 1680-1687, 2021.
11. Das Mou A, Saha PK, Nisher SA, Saha A (2021) A comprehensive study of machine learning algorithms for predicting car purchase based on customers demands. In: 2021 International conference on information and communication technology for sustainable development (ICICT4SD, pp 180–184.
12. C. Jin, "Price Prediction of Used Cars Using Machine Learning," 2021 IEEE International Conference on Emergency Science and Information Technology (ICESIT), Chongqing, China, 2021, pp. 223-230, doi: 10.1109/ICESIT53460.2021.9696839.
13. Rane, Praful, Deepak Pandya and Dhawal Kotak. "USED CAR PRICE PREDICTION." (2021).
14. Mishra, Amit & Mallik, Saurav & Sharma, Viney & Paliwal, Shweta & Ray, Kanad. (2023). Integrated Linear Regression and Random Forest Framework for E-Commerce Price Prediction of Pre-owned Vehicle.
15. Kriswantara B, Sadikin R (2022) Machine learning used car price prediction with random forest regressor model. J Inf Syst Inf Comput 6(1):40–49.
16. Reddy A, Kamalraj R (2021) Old/Used cars price prediction using machine learning algorithms. IITM J Manag IT 12(1):32–35
17. Narayana CV, Likhitha CL, Bademiya S, Kusumanjali K (2021) Machine learning techniques to predict the price of used cars: predictive analytics in retail business. In: 2021 second international conference on electronics and sustainable communication systems (ICESC), pp 1680–1687.

18. Q. Yuan, Y. Liu, G. Peng and B. Lv, "A prediction study on the car sales based on web search data", The International Conference on E-Business and E-Government (Index by EI), pp. 5, 2011.
19. L. Chetna, Potharaju, P. Sai and D. Sandhya, "Price Prediction for Pre-Owned Cars Using Ensemble Machine Learning Techniques", *2021 Recent Trends in Intensive Computing*, 2021.
20. D. Adhikary, D.R, R. Sahu, P. Pragyna and S, "Prediction of Used Car Prices Using Machine Learning", *2022 Smart Innovation Systems and Technologies*, vol. 271, 2022.

ДОДАТОК А

Лістинг програмного коду

boolean_handler.py

```
from sklearn.base import BaseEstimator, TransformerMixin
import pandas as pd
class BooleanHandler(BaseEstimator, TransformerMixin):
    def __init__(self) -> None:
        pass
    def fit(self, df: pd.DataFrame, y=None):
        return self
    def transform(self, df: pd.DataFrame) -> pd.DataFrame:
        df["car_wanted"] = (
            df["car_wanted"].fillna(value=-1).replace({r"Ні": 0, r"Так": 1}, regex=True)
        )
        df["car_had_accidents"] = (
            df["car_had_accidents"]
            .fillna(value=-1)
            .replace({r"Ні": 0, r"Зафіксовано ДТП|Був в ДТП": 1}, regex=True)
        )
        df["car_vin_code_verified"] = (
            df["car_vin_code_verified"]
            .fillna(value=-1)
            .replace({"False": 0, "True": 1})
        )
        df["car_is_sold"] = (
            df["car_is_sold"].fillna(value=-1).replace({"False": 0, "True": 1})
        )
        df["ad_is_deleted"] = (
            df["ad_is_deleted"].fillna(value=-1).replace({"False": 0, "True": 1})
        )
        return df
```

car_engine_info_transformer.py

```
import re
import pandas as pd
from sklearn.base import BaseEstimator, TransformerMixin
from typing import Optional
```

```
class EngineVolumePowerTypeTransformer(BaseEstimator, TransformerMixin):
    def __init__(self) -> None:
        pass
    def fit(self, df: pd.DataFrame, y=None):
        return self
    def find_engine_volume(self, row: str) -> Optional[str]:
        row = str(row)
        match = re.search(r"[0-9]+[.]*[0-9]* л", row)
        if match is not None:
            return match.group(0).rstrip(" л")
        else:
            return None
    def find_engine_type(self, row: str) -> str:
        row = str(row)
        gas_and_oil = "Газ / Бензин"
        if gas_and_oil in row:
            return gas_and_oil
        else:
            row = row.replace(gas_and_oil, "")
            match = re.findall(r"[A-Яа-яіІїЇ]{3,}", row)
            result = list(
                filter(
                    lambda elem: bool(re.search(r"[A-Яа-яіІїЇ]", elem))
                    and "кВт" not in elem,
                    match,
                )
            )
            return result[0].strip() if result else None

    def find_engine_horse_power(self, row: str) -> Optional[str]:
        row = str(row)
        match = re.search(r"[1-9][0-9]*[.]*[0-9]*[ ]+к.с.", row)
        if match is not None:
            return match.group(0).rstrip(" к.с.")
        else:
            return None

    def find_engine_energy_power(self, row: str) -> Optional[str]:
```

```
        row = str(row)
        match = re.search(r"[1-9][0-9]*[.]*[0-9]*[ ]+кВт", row)
        if match is not None:
            return match.group(0).rstrip(" кВт")
        else:
            return None
    def transform(self, df: pd.DataFrame) -> pd.DataFrame:
        car_engine_volume = df.car_engine_info.apply(
            lambda x: self.find_engine_volume(x)
        )
        car_engine_type = df.car_engine_info.apply(lambda x: self.find_engine_type(x))
        car_engine_horse_power = df.car_engine_info.apply(
            lambda x: self.find_engine_horse_power(x)
        )
        car_engine_energy_power = df.car_engine_info.apply(
            lambda x: self.find_engine_energy_power(x)
        )
        df.insert(7, "car_engine_energy_power(kWh)", car_engine_energy_power)
        df.insert(7, "car_engine_horse_power", car_engine_horse_power)
        df.insert(7, "car_engine_type", car_engine_type)
        df.insert(7, "car_engine_volume(liters)", car_engine_volume)
        df.drop(columns=["car_engine_info"], inplace=True)
        return df
```

car_model_transformer.py

```
from sklearn.base import BaseEstimator, TransformerMixin
import pandas as pd
import pickle
class CarModelTransformer(BaseEstimator, TransformerMixin):
    def __init__(self) -> None:
        with open("dict.pickle", "rb") as handle:
            data = pickle.load(handle)
        self.cars_brands_models = data
        pass

    def fit(self, y=None):
        return self
```

```
def extract_car_model(self, row: pd.Series, brand: str) -> str:
    if brand in self.cars_brands_models.keys():
        model = row["car_model"]
        possible_models = []
        for dict_model in self.cars_brands_models[brand]:
            if model.startswith(dict_model):
                possible_models.append(dict_model)
        car_model = (
            max(possible_models, key=len) if len(possible_models) > 0 else model
        )
        car_specification = model[len(car_model) :].strip()
        if car_specification == "":
            car_specification = None
        return pd.Series(
            [car_model, car_specification], index=["car_model", "car_specification"]
        )
    return row

def transform(self, df: pd.DataFrame) -> pd.DataFrame:
    df_new = df.copy()
    df_new["car_specification"] = pd.Series(None)
    for brand in df.car_brand.unique():
        sub_df = df_new.loc[
            df_new.car_brand == brand, ["car_model", "car_specification"]
        ]
        sub_df = sub_df.apply(
            self.extract_car_model,
            axis=1,
            args=(brand,),
            result_type="reduce",
        )
        df_new.loc[
            df_new.car_brand == brand, ["car_model", "car_specification"]
        ] = sub_df
    return df_new
```

car_name_transformet.py

```
from sklearn.base import BaseEstimator, TransformerMixin
from typing import List
import pandas as pd
TWO_WORDS_BRANDS = [
    "Alfa Romeo",
    "Land Rover",
    "Great Wall",
    "Mobility Ventures",
    "SMA Maple",
    "Aston Martin",
    "Nysa (Ниса)",
    "Ретро автомобілі",
    "Barkas (Баркас)",
    "Iran Khodro",
    "De Lorean",
    "MPM Motors",
    "Golf Car",
]
class ModelBrandYearExtractor(BaseEstimator, TransformerMixin):
    def __init__(self) -> None:
        pass
    def fit(self, df: pd.DataFrame, y=None):
        return self
    def get_car_brand(
        self, car_name_words: List[str], two_words_brands: List[str]
    ) -> str:
        car_brand = car_name_words[0]
        two_words_brand = " ".join(car_name_words[:2])
        return car_brand if two_words_brand not in two_words_brands else
two_words_brand
    def transform(self, df: pd.DataFrame) -> pd.DataFrame:
        car_name_words = df.car_name.apply(lambda x: x.split())
        car_brand = car_name_words.apply(
            lambda x: self.get_car_brand(x, TWO_WORDS_BRANDS)
        )
        df.insert(1, "car_brand", car_brand)
        car_model_and_year = df.apply(
```

```
            lambda x: x.car_name[len(x.car_brand) + 1 :], axis=1
        )
        car_model = car_model_and_year.apply(lambda x: " ".join(x.split()[:-1]))
        car_year = car_model_and_year.apply(lambda x: x.split()[-1])
        df.insert(2, "car_model", car_model)
        df.insert(3, "car_year", car_year)
        df.drop(columns=["car_name"], inplace=True)
        return df
```

empty_null_values_handler.py

```
from sklearn.base import BaseEstimator, TransformerMixin
import pandas as pd

class EmptyValuesHandler(BaseEstimator, TransformerMixin):
    def __init__(self) -> None:
        pass
    def fit(self, df: pd.DataFrame, y=None):
        return self

    def transform(self, df: pd.DataFrame) -> pd.DataFrame:
        df.replace(r"^\s*$", None, regex=True, inplace=True)
        df["seller_name"].replace(r"Ім’я не вказане", None, regex=True, inplace=True)
        df["car_had_accidents"].replace(
            r"Немає офіційно зареєстрованих", None, regex=True, inplace=True
        )
        return df
```

empty_rows_cleaner.py

```
from sklearn.base import BaseEstimator, TransformerMixin
import pandas as pd

class EmptyRowsCleaner(BaseEstimator, TransformerMixin):
    def __init__(self) -> None:
        pass

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

```
        return self
    def transform(self, df: pd.DataFrame) -> pd.DataFrame:
        return df.drop(
            df[
                df.car_mileage_thousands_km.isnull()
                | df.car_unique_id.isnull()
                | df.car_price.isnull()
            ].index
        )
```

extra_symbols_cleaner.py

```
import re
from sklearn.base import BaseEstimator, TransformerMixin
import pandas as pd
class ExtraSymbolsCleaner(BaseEstimator, TransformerMixin):
    def __init__(self) -> None:
        pass
    def fit(self, df: pd.DataFrame, y=None):
        return self
    def transform(self, df: pd.DataFrame) -> pd.DataFrame:
        df = df.applymap(lambda x: str(x).strip("\t\r\n "), na_action="ignore")
        df.car_mileage_thousands_km = df.car_mileage_thousands_km.apply(
            lambda x: re.sub(" тис. км пробіг", "", x)
        )
        df.car_type = df.car_type.apply(
            lambda x: re.sub(
                r" *\(до 1,5т\) *| *[0-9] дверей *| *[0-9] місц[ья] *", "", x
            )
        )
        df.car_type.replace("", None, inplace=True)
        return df
```

photos_number_extractor.py

```
import re
from typing import Optional
import pandas as pd
from sklearn.base import BaseEstimator, TransformerMixin
```

```
class PhotosNumberExtractor(BaseEstimator, TransformerMixin):
    def __init__(self) -> None:
        pass

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

    def extract_photos_number(self, row: Optional[str]) -> str:
        row = str(row)
        if row is not None:
            match = re.search(r"[0-9]+", row)
            if match is not None:
                return match.group(0)
        return "0"

    def transform(self, df: pd.DataFrame) -> pd.DataFrame:
        df.car_photos_number = df.car_photos_number.apply(self.extract_photos_number)
        return df
```

price_transformer.py

```
import re
import requests
from datetime import date
import pandas as pd
from sklearn.base import BaseEstimator, TransformerMixin
class PriceTransformer(BaseEstimator, TransformerMixin):
    def __init__(self) -> None:
        pass

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

    def transform(self, df: pd.DataFrame) -> pd.DataFrame:
        prices = df.car_price
        usd_prices = prices[prices.str.contains("$", regex=False)].apply(
            lambda x: re.sub(r"[ ]+|[$]", "", x)
        )
```

```
    hrn_prices = prices[prices.str.contains("грн", regex=False)].apply(
        lambda x: re.sub(r"[ ]+|[грн]", "", x)
    )
    eur_prices = prices[prices.str.contains("€", regex=False)].apply(
        lambda x: re.sub(r"[ ]+|[€]", "", x)
    )
    today_date = date.today().strftime("%d.%m.%Y")
    exchange_rates = requests.get(
        url=f"https://bank.gov.ua/NBU_Exchange/exchange?json/date={today_date}"
    ).json()
    exchange_rate_usd_to_hrn = [
        elem["Amount"] for elem in exchange_rates if elem["CurrencyCodeL"] == "USD"
    ][0]
    exchange_rate_eur_to_hrn = [
        elem["Amount"] for elem in exchange_rates if elem["CurrencyCodeL"] == "EUR"
    ][0]
    exchange_rate_usd_to_eur = exchange_rate_usd_to_hrn / exchange_rate_eur_to_hrn
    hrn_to_usd_prices = hrn_prices.apply(
        lambda x: round(int(x) / exchange_rate_usd_to_hrn)
    )
    eur_to_usd_prices = eur_prices.apply(
        lambda x: round(int(x) / exchange_rate_usd_to_eur)
    )
    df.car_price = (
        pd.concat([eur_to_usd_prices, hrn_to_usd_prices, usd_prices])
        .sort_index()
        .astype(int)
    )
    df.rename(columns={"car_price": "car_price_usd"}, inplace=True)
    return df
```

seller_city_handler.py

```
import re
import pandas as pd
from sklearn.base import BaseEstimator, TransformerMixin
from typing import Optional
```

```
CITIES = {
    "Киев": "Київ",
    "Харьков": "Харків",
    "Львов": "Львів",
    "Одесса": "Одеса",
    "Днепр": "Дніпро",
    "Запорожье": "Запоріжжя",
    "Кривой Рог": "Кривий Ріг",
    "Николаев": "Миколаїв",
    "Винница": "Вінниця",
    "Франківськ": "Івано-Франківськ",
    "Луцк": "Луцьк",
    "Красноград": "Червоноград",
    "Хмельницкий": "Хмельницький",
    "Франковск": "Івано-Франківськ",
    "Бердичев": "Бердичів",
    "Ровно": "Рівне",
    "Тернополь": "Тернопіль",
    "Коломыя": "Коломия",
    "Знамянка": "Знам'янка",
    "Измаил": "Ізмаїл",
}
class SellerCityHandler(BaseEstimator, TransformerMixin):
    def __init__(self) -> None:
        pass

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

    def handle_city(self, row: str) -> Optional[str]:
        row = str(row)
        if not pd.notnull(row):
            return None
        else:
            match = re.search(r"\w+(?=([,\\]| \(| Зареєстрований))", str(row))
            city = match.group(0) if match else row
            return city if city not in CITIES else CITIES[city]
```

```
    def transform(self, df: pd.DataFrame) -> pd.DataFrame:
        df["seller_city"] = df["seller_city"].apply(lambda x: self.handle_city(x))
        return df
```

type_transformer.py

```
from sklearn.base import BaseEstimator, TransformerMixin
import pandas as pd

class TypeTransformer(BaseEstimator, TransformerMixin):
    def __init__(self) -> None:
        pass

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

    def transform(self, df: pd.DataFrame) -> pd.DataFrame:
        df["car_unique_id"] = df["car_unique_id"].astype(float).astype(int)
        df["car_photos_number"] = df["car_photos_number"].astype(int)
        df = df.rename(
            columns={
                "car_engine_volume(liters)": "car_engine_volume_liters",
                "car_engine_energy_power(kWh)": "car_engine_energy_power_kWh",
            }
        )
        return df
```

dataframe_creating.py

```
import os
import json
import pandas as pd

def get_dataframe_from_json_files(DIR_NAME) -> pd.DataFrame:
    result = []
    if os.path.isdir(DIR_NAME):
        files = os.listdir(DIR_NAME)
        for file in files:
            with open(DIR_NAME + file, encoding="ascii") as f:
                result.append(json.load(f))
```

```
    df = pd.DataFrame(result)
    return df
```

pipeline.py

```
import pandas as pd
from sklearn.pipeline import Pipeline
from cleaning.transformers.price_transformer import PriceTransformer
from cleaning.transformers.car_name_transformer import ModelBrandYearExtractor
from cleaning.transformers.car_engine_info_transformer import
EngineVolumePowerTypeTransformer
from cleaning.transformers.empty_null_values_handler import EmptyValuesHandler
from cleaning.transformers.boolean_handler import BooleanHandler
from cleaning.transformers.seller_city_handler import SellerCityHandler
from cleaning.transformers.empty_rows_cleaner import EmptyRowsCleaner
from cleaning.transformers.type_transformer import TypeTransformer
from cleaning.transformers.photos_number_extractor import PhotosNumberExtractor
from cleaning.transformers.extra_symbols_cleaner import ExtraSymbolsCleaner
from cleaning.transformers.car_model_transformer import CarModelTransformer

def transform(df: pd.DataFrame) -> pd.DataFrame:
    pipeline = Pipeline(
        steps=[
            ("empty_rows_cleaner", EmptyRowsCleaner()),
            ("extra_symbols_cleaner", ExtraSymbolsCleaner()),
            ("empty_null_values_handler", EmptyValuesHandler()),
            ("price_transformer", PriceTransformer()),
            ("model_brand_year_extractor", ModelBrandYearExtractor()),
            ("car_engine_info_transformer", EngineVolumePowerTypeTransformer()),
            ("photos_number_extractor", PhotosNumberExtractor()),
            ("boolean_handler", BooleanHandler()),
            ("seller_city_handler", SellerCityHandler()),
            ("car_model_transformer", CarModelTransformer()),
            ("type_transformer", TypeTransformer()),
        ]
    )
    return pipeline.fit_transform(df)
```

autoria_api_scraper.py

```
from typing import Iterator
from scrapy import Spider
from scrapy.http import Request
from lxml import etree
from data_extraction.helpers.extraction_helper import get_parsed_data
import json
import pandas as pd
import json
import os
from db.filling_db import create_db_engine
from db.orm import Car
from shutil import rmtree
from fake_useragent import UserAgent
class AutoriaAPIScraper(Spider):
    def __init__(self, directory):
        self.output_directory = directory
    name = "autoria_api_scraper"
    def start_requests(self):
        db_engine = create_db_engine()
        db_connection = db_engine.connect()
        cars_remote = pd.read_sql(Car.__tablename__, con=db_connection)
        cars_remote.to_csv("cars_remote.csv", index=False)
        cars_remote = pd.read_csv("cars_remote.csv")
        cars_remote_ids = cars_remote.car_unique_id.iloc[:10000]
        db_connection.close()
        api_credentials = pd.read_csv("api_credentials.csv")
        api_credentials_count = api_credentials.shape[0]
        for index, id in cars_remote_ids.iteritems():
            api_credential = api_credentials.iloc[index % api_credentials_count]
            headers = {
                "Connection": "keep-alive",
                "Cache-Control": "max-age=0",
                "DNT": "1",
                "Upgrade-Insecure-Requests": "1",
                "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.87 Safari/537.36",
                "Sec-Fetch-User": "?1",
```

```
            "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.
8,application/signed-exchange;v=b3",
            "Sec-Fetch-Site": "same-origin",
            "Sec-Fetch-Mode": "navigate",
            "Accept-Encoding": "gzip, deflate, br",
            "Accept-Language": "en-US,en;q=0.9",
        }
        yield Request(

f"https://developers.ria.com/auto/info?api_key={api_credential.api_key}&auto_id={id}",
            headers=headers,
            callback=self.parse_page_with_multiple_cars,
            cb_kwargs={"index": index, "api_key": api_credential.api_key},
        )

    def parse_page_with_multiple_cars(self, response, **kwargs):
        api_key = str(kwargs["api_key"])
        if response.status == 200:
            data = json.loads(response.text)
            with open(
                f"{self.output_directory}/{response.url.split('auto_id=')[-1]}_{api_key}.json",
                "w",
            ) as f:
                json.dump(data, f, ensure_ascii=False)
```

car_model_scraper.py

```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
import time
import pickle
def get_car_brand_models():
    url = "https://auto.ria.com/uk/"
    chrome_options = Options()
    chrome_options.add_argument("--headless")
    driver = webdriver.Chrome(
        "C:\\Users\\Sergey\\Desktop\\chromedriver.exe", options=chrome_options
    )
```

```
driver.get(url)
dictionary = dict()
input_elem_brand = driver.find_element_by_xpath(
    "//label[@for='brandTooltipBrandAutocompleteInput-brand']"
)
driver.execute_script("arguments[0].click();", input_elem_brand)
time.sleep(1)
car_brand_elems = driver.find_elements_by_xpath(
    "//div[@id='brandTooltipBrandAutocomplete-brand']//ul//li"
)
for index, car_brand_elem in enumerate(car_brand_elems):
    if car_brand_elem.text == "Усі марки":
        car_brand_elems = car_brand_elems[index + 1 :]
        break
for car_brand_elem in car_brand_elems:
    car_brand = car_brand_elem.text
    driver.execute_script("arguments[0].click();", car_brand_elem)
    time.sleep(1)
    input_elem_model = driver.find_element_by_xpath(
        "//label[@for='brandTooltipBrandAutocompleteInput-model']"
    )
    driver.execute_script("arguments[0].click();", input_elem_model)
    time.sleep(1)
    car_model_elems = driver.find_elements_by_xpath(
        "//div[@id='brandTooltipBrandAutocomplete-model']//ul//li"
    )
    for i, car_model_elem in enumerate(car_model_elems):
        if car_model_elem.text == "Усі модели":
            car_model_elems = car_model_elems[i + 1 :]
            break
    if "Спочатку оберіть марку" in map(lambda x: x.text, car_model_elems):
        car_models = []
    else:
        car_models = [car_model_elem.text for car_model_elem in car_model_elems]
    print(car_brand, car_models)
    dictionary[car_brand] = car_models
    elem = driver.find_element_by_xpath("//span[@class='ac-clean']")
    driver.execute_script("arguments[0].click();", elem)
```

```
        time.sleep(1)
    driver.close()
    with open("dict.pickle", "wb") as f:
        pickle.dump(dictionary, f)
    return dictionary
```

db_info_scraper.py

```
from typing import Iterator
from scrapy import Spider
from scrapy.http import Request
from lxml import etree
from data_extraction.helpers.extraction_helper import get_parsed_data
import json
class DbInfoScraper(Spider):
    def __init__(self, df, DIR_NAME):
        self.df = df
        self.DIR_NAME = DIR_NAME
    name = "db_info_scraper"
    def start_requests(self) -> Iterator[Request]:
        for index, row in self.df.iterrows():
            yield Request(
                f"https://auto.ria.com/uk/auto___{row.car_unique_id}.html",
                callback=self.parse_page_with_multiple_cars,
                cb_kwargs={"index": index},
            )
    def parse_page_with_multiple_cars(self, response, **kwargs):
        dom = etree.HTML(response.text)
        dictionary = get_parsed_data(dom)
        if dictionary is not None:
            with open(self.DIR_NAME + f"{kwargs['index']}.json", "a") as f:
                json.dump(dictionary, f)
```

html_scraper.py

```
import os
import json
from data_extraction.helpers.extraction_helper import get_parsed_data
from concurrent.futures import ThreadPoolExecutor, as_completed, ProcessPoolExecutor
from lxml import etree
```

```
from typing import Optional

DIR_NAME = "html/"
OUTPUT_DIR = "json/"
PATH = os.path.abspath(os.curdir) + "/html/"

class HtmlParser:
    name = "html_parser"
    def parse(self) -> None:

        if os.path.isdir(DIR_NAME):
            filenames = os.listdir(DIR_NAME)
            if not os.path.exists(os.path.dirname(OUTPUT_DIR)):
                os.makedirs(os.path.dirname(OUTPUT_DIR))

            with ProcessPoolExecutor() as executor:
                jobs = {
                    executor.submit(self.parse_html, filename): filename
                    for filename in filenames
                }

            with ThreadPoolExecutor() as executor2:
                for job in as_completed(jobs):
                    executor2.submit(self.write_json_file, job.result(), jobs[job])

        return DIR_NAME

    def parse_html(self, filename: str) -> Optional[dict]:
        dom = etree.HTML(open(PATH + filename, "r", encoding="utf-8").read())
        if dom is None:
            return None
        return get_parsed_data(dom)

    def write_json_file(self, data: dict, filename: str) -> None:
        if data is not None:
            with open(OUTPUT_DIR + filename.replace("html", "json"), "a") as f:
                json.dump(data, f)
```

web_scraper.py

```
from typing import Iterator, Dict
from scrapy import Spider
from scrapy.http import TextResponse, Request
import logging
import os

DIR_NAME = "html/"

class WebScraper(Spider):
    logger = logging.getLogger("scrapy.spidermiddlewares.httperror")
    logger.setLevel(logging.WARNING)
    name = "web_scraper"
    start_urls = [
        "https://auto.ria.com/uk/search/?indexName=auto&categories.main.id=1& \
        country.import.usa.not=-1&price.currency=1& \
        abroad.not=0&custom.not=1&page=0&size=100"
    ]

    def start_requests(self) -> Iterator[Request]:
        if not os.path.exists(os.path.dirname(DIR_NAME)):
            os.makedirs(os.path.dirname(DIR_NAME))
        for u in self.start_urls:
            yield Request(u, callback=self.parse_page_with_multiple_cars)

    def parse_page_with_multiple_cars(
        self, response: TextResponse
    ) -> Iterator[Request]:
        if response.status == 200:
            cars = response.xpath("//section[contains(@class, 'ticket-item')]")
            if cars:
                page_number = int(response.url.split("page=")[-1].split("&size=")[0])
                for (index, car) in enumerate(cars):
                    product_page = car.css("a.m-link-ticket::attr(href)").get()
                    if product_page is not None:
                        yield response.follow(
                            product_page,
                            callback=self.parse_car_page,
```

```
                cb_kwargs={
                    "page_number": page_number,
                    "product_index": index + 1,
                },
            )

        next_page = (
            self.start_urls[0].split("page=")[0]
            + "page="
            + str(int(page_number) + 1)
            + "&size=100"
        )
        yield response.follow(
            next_page, callback=self.parse_page_with_multiple_cars
        )

def parse_car_page(self, response: TextResponse, **kwargs: Dict[str, int]) -> None:
    if response.status == 200:
        page_number = dict.get(kwargs, "page_number")
        product_number = dict.get(kwargs, "product_index")
        if page_number and product_number:
            file_name = f"page{page_number}-product{product_number}.html"
            with open(DIR_NAME + file_name, "a", encoding="utf-8") as f:
                content = response.css("body div.app-content")[0].css(
                    "div.auto-wrap"
                )
                ad_is_deleted_block = response.xpath("//div[@class='notice_head']")
                car_is_sold = response.xpath(
                    "//div[@class='gallery-order sold-out carousel']"
                )
                car_photos_number = response.xpath(
                    "//span[@class='count']/span[@class='mhide']"
                )
                html = [
                    ad_is_deleted_block.get() if ad_is_deleted_block else "",
                    car_is_sold.get() if car_is_sold else "",
                    car_photos_number.get() if car_is_sold else "",
                    content.css("aside").get(),
```

```
            content.css("div.m-padding").get(),
            content.css("#floatingButtons").get(),
            content.css("#leftPanelMobile").get(),
        ]
        f.writelines(html)
```

filling_db.py

```
import pandas as pd
import re
from sqlalchemy import create_engine
from db.orm import Car, SellerPhoneNumber, Ad, Timestamp, Base
from sqlalchemy.engine import Engine, Connection
from sqlalchemy import inspect

def create_db_engine() -> Engine:
    DBMS = "mssql"
    DB_CONNECTOR = "pyodbc"
    USER_NAME = "Serhii"
    PASSWORD = "Webscraping1"
    SERVER = "autoria.database.windows.net"
    PORT = "1433"
    DATABASE = "database"
    DRIVER = "ODBC+Driver+17+for+SQL+Server"
    connection_string = f"{DBMS}+{DB_CONNECTOR}://{USER_NAME}\
    :{PASSWORD}@{SERVER}:{PORT}/{DATABASE}?driver={DRIVER}"
    engine = create_engine(
        connection_string,
        connect_args={"sslmode": "require"},
        fast_executemany=True,
        echo=True,
    )
    return engine

def get_local_data_tables(df: pd.DataFrame) -> dict:
    cars_columns = list(map(lambda x: x.name, Car.__table__.columns))
    cars = df[cars_columns]
    ads = df[["car_unique_id", "ad_is_deleted"]]
    phones = df[["car_unique_id", "seller_phone_numbers"]]
```

```
    new_df = []
    for row in list(phones.iterrows()):
        phone_list_row = row[1].loc["seller_phone_numbers"]
        phone_numbers = "".join(re.findall(r"[0-9]+|, ", phone_list_row))
        phone_numbers_list = phone_numbers.split(sep=", ")
        if phone_numbers == "":
            new_df.append([row[1].loc["car_unique_id"], None])
        else:
            for phone_number in phone_numbers_list:
                new_df.append([row[1].loc["car_unique_id"], phone_number.strip("'")])
    phones = pd.DataFrame(new_df, columns=["car_unique_id", "seller_phone_number"])
    timestamps = df[["car_unique_id", "car_price_usd"]]
    current_date = pd.to_datetime("today").date()
    timestamps["timestamp"] = current_date
    local_data_tables = {
        Car: cars,
        Ad: ads,
        SellerPhoneNumber: phones,
        Timestamp: timestamps,
    }
    return local_data_tables
def fill_db_with_new_data(
    local_data_tables: dict, db_engine: Engine, db_connection: Connection
) -> None:
    for orm_class in local_data_tables:
        table_name = orm_class.__tablename__
        df_local_data_table = local_data_tables[orm_class]
        if not inspect(db_engine).has_table(table_name):
            Base.metadata.create_all(db_engine, tables=[orm_class.__table__])
            local_data_tables[orm_class].to_sql(
                table_name,
                con=db_connection,
                index=False,
                if_exists="append",
                chunksize=10000,
            )
        else:
            table_from_db = pd.read_sql(table_name, con=db_connection)
```

```
            if table_from_db.empty:
                df_local_data_table.to_sql(
                    table_name,
                    con=db_connection,
                    index=False,
                    if_exists="append",
                    chunksize=10000,
                )
            else:
                new_ads = df_local_data_table[
                    ~df_local_data_table.car_unique_id.isin(table_from_db.car_unique_id)
                ]
                if not new_ads.empty and orm_class is not Timestamp:
                    new_ads.to_sql(
                        table_name,
                        con=db_connection,
                        index=False,
                        if_exists="append",
                        chunksize=10000,
                    )
def update_ads_and_return_deleted_ads(
    local_data_tables: dict, db_connection: Connection
) -> pd.DataFrame:
    ads_from_local_data = local_data_tables[Ad]
    ads_from_db = pd.read_sql(Ad.__tablename__, con=db_connection)
    joined_ads = pd.merge(
        ads_from_db,
        ads_from_local_data,
        how="inner",
        on="car_unique_id",
        suffixes=("_db", "_local"),
    )
    deleted_ads = joined_ads[
        (joined_ads.ad_is_deleted_db == 0) & (joined_ads.ad_is_deleted_local == 1)
    ]
    restored_ads = joined_ads[
        (joined_ads.ad_is_deleted_db == 1) & (joined_ads.ad_is_deleted_local == 0)
    ]
```

```
    if not deleted_ads.empty or not restored_ads.empty:
        newly_deleted_ads_in_db = ads_from_db.loc[
            ads_from_db.car_unique_id.isin(deleted_ads.car_unique_id), "ad_is_deleted"
        ]
        newly_restored_ads_in_db = ads_from_db.loc[
            ads_from_db.car_unique_id.isin(restored_ads.car_unique_id), "ad_is_deleted"
        ]
        transaction = db_connection.begin()
        try:
            newly_deleted_ads_in_db.to_sql(
                "deleted_ads",
                con=db_connection,
                index=False,
                if_exists="replace",
                chunksize=10000,
            )
            newly_restored_ads_in_db.to_sql(
                "restored_ads",
                con=db_connection,
                index=False,
                if_exists="replace",
                chunksize=10000,
            )
            db_connection.execute(
                "UPDATE ads SET ad_is_deleted = 1\
                    WHERE ads.car_unique_id IN (SELECT car_unique_id FROM deleted_ads)"
            )
            db_connection.execute(
                "UPDATE ads SET ad_is_deleted = 0\
                    WHERE ads.car_unique_id IN (SELECT car_unique_id FROM
restored_ads)"
            )
            db_connection.execute("DROP TABLE deleted_ads")
            db_connection.execute("DROP TABLE restored_ads")
            transaction.commit()
        except Exception:
            transaction.rollback()
```

```
    return deleted_ads


def fill_timestamps(
    df: pd.DataFrame, deleted_ads: pd.DataFrame, db_connection: Connection
) -> None:
    new_timestamps = df[(~df.car_unique_id.isin(deleted_ads.car_unique_id))][
        ["car_unique_id", "car_price_usd"]
    ]
    new_timestamps["timestamp"] = pd.to_datetime("today").date()
    new_timestamps.timestamp = new_timestamps.timestamp.astype("datetime64")
    new_timestamps = new_timestamps.drop_duplicates()

    if not new_timestamps.empty:
        new_timestamps.to_sql(
            "timestamps",
            con=db_connection,
            index=False,
            if_exists="append",
            chunksize=10000,
        )

def fill_db(df: pd.DataFrame) -> None:
    db_engine = create_db_engine()
    db_connection = db_engine.connect()
    local_data_tables = get_local_data_tables(df)
    fill_db_with_new_data(local_data_tables, db_engine, db_connection)
    deleted_ads = update_ads_and_return_deleted_ads(local_data_tables, db_connection)
    fill_timestamps(df, deleted_ads, db_connection)
    db_connection.close()
```

main.py

```
from scrapy.utils.project import get_project_settings
from scrapy.crawler import CrawlerProcess
from data_extraction.spiders.html_parser import HtmlParser
from cleaning.dataframe_creating import get_dataframe_from_json_files
from cleaning import pipeline
```

```
from db.filling_db import fill_db
from shutil import rmtree
import pandas as pd
from cleaning.transformers.car_model_transformer import CarModelTransformer
from tools.update_db_info import update_db_info
from tools.add_car_specification import add_car_specification
import json
import os
from data_extraction import settings
from db.filling_db import create_db_engine
from db.orm import Car, Timestamp
from shutil import rmtree


def main() -> None:
    """Main function to run the program"""
    process = CrawlerProcess(get_project_settings())
    process.crawl("web_scraper")
    process.start()
    OUTPUT_DIR_NAME = HtmlParser().parse()
    raw_df = get_dataframe_from_json_files(OUTPUT_DIR_NAME)
    cleaned_df = pipeline.transform(raw_df)
    fill_db(cleaned_df)
    rmtree("html")
    rmtree("json")
    print("FINISH")

    update_db_info(
        [
            "car_brand",
            "car_model",
            "car_specification",
            "car_photos_number",
            "car_is_sold",
        ]
    )
    print("FINISH")
    add_car_specification()
```

```
    output_folder = "json2"
    os.makedirs(output_folder, exist_ok=True)
    process = CrawlerProcess(get_project_settings())
    process.crawl("autoria_api_scraper", output_folder)
    process.start()
    df = pd.DataFrame()

    if os.path.isdir(output_folder):
        files = os.listdir(output_folder)
        for file in files:
            path = output_folder + "/" + file
            with open(path, "r") as f:
                df = df.append(pd.json_normalize(json.load(f)))
    rmtree(settings.JOBDIR)
    rmtree(output_folder)
    df.to_csv("output_df.csv", index=False)

    db_engine = create_db_engine()
    db_connection = db_engine.connect()
    cars = pd.read_sql(Car.__tablename__, con=db_connection)
    timestamps = pd.read_sql(Timestamp.__tablename__, con=db_connection)
    cars.to_csv("cars.csv", index=False)
    timestamps.to_csv("timestamps.csv", index=False)

    db_connection.close()

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

orm.py

```
from sqlalchemy import ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String, Float, SmallInteger, Date

Base = declarative_base()
```

```
class Car(Base):
    __tablename__ = "cars"
    car_unique_id = Column(Integer, primary_key=True, autoincrement=False)
    car_brand = Column(String, nullable=False)
    car_model = Column(String, nullable=False)
    car_specification = Column(String, nullable=True)
    car_year = Column(SmallInteger, nullable=False)
    car_mileage_thousands_km = Column(Float, nullable=False)
    car_type = Column(String, nullable=False)
    car_engine_volume_liters = Column(Float, nullable=True)
    car_engine_type = Column(String, nullable=True)
    car_engine_energy_power_kWh = Column(Float, nullable=True)
    car_model_rating = Column(Float, nullable=True)
    car_gearbox = Column(String, nullable=True)
    car_drive = Column(String, nullable=True)
    car_color = Column(String, nullable=True)
    car_description = Column(String, nullable=True)
    car_state = Column(String, nullable=True)
    car_wanted = Column(SmallInteger, nullable=True)
    car_had_accidents = Column(SmallInteger, nullable=True)
    car_vin_code_verified = Column(SmallInteger, nullable=True)
    car_photos_number = Column(SmallInteger, nullable=True)
    car_is_sold = Column(SmallInteger, nullable=True)
    seller_type = Column(String, nullable=True)
    seller_name = Column(String, nullable=True)
    seller_city = Column(String, nullable=True)

class Ad(Base):
    __tablename__ = "ads"
    car_unique_id = Column(
        Integer, ForeignKey("cars.car_unique_id"), primary_key=True, autoincrement=False
    )
    ad_is_deleted = Column(SmallInteger, nullable=True)

class SellerPhoneNumber(Base):
    id = Column(Integer, primary_key=True, autoincrement=True)
    car_unique_id = Column(Integer, ForeignKey("cars.car_unique_id"))
    seller_phone_number = Column(String, nullable=True)
```

```
class Timestamp(Base):
    __tablename__ = "timestamps"
    car_unique_id = Column(
        Integer, ForeignKey("cars.car_unique_id"), primary_key=True, autoincrement=False
    )
    car_price_usd = Column(Integer, primary_key=True, autoincrement=False)
    timestamp = Column(Date, primary_key=True, autoincrement=False)
```

update_db_info.py

```
from scrapy.utils.project import get_project_settings
from scrapy.crawler import CrawlerProcess
import pandas as pd
from db.orm import Car
from db.filling_db import create_db_engine
from cleaning import pipeline
import os
from shutil import rmtree
from cleaning.dataframe_creating import get_dataframe_from_json_files
from cleaning.transformers.car_model_transformer import CarModelTransformer
from data_extraction import settings
from sqlalchemy import create_engine

def update_db_info(columns_list):
    db_engine = create_db_engine()
    db_engine = create_engine(
        "mssql+pyodbc://WINDOWS-
IECQDVR/database?driver=ODBC+Driver+17+for+SQL+Server?Trusted_Connection=ye
s",
        connect_args={"sslmode": "require"},
        fast_executemany=True,
        echo=True,
    )
    db_connection = db_engine.connect()
    cars_remote = pd.read_sql(Car.__tablename__, con=db_connection)
    cars_remote.to_csv("cars_remote.csv", index=False)
    cars_remote = pd.read_csv("cars_remote.csv")
```

```
DIR_NAME = "tools/scraped_items/"
os.mkdir(DIR_NAME)

process = CrawlerProcess(get_project_settings())
process.crawl("db_info_scraper", cars_remote, DIR_NAME)
process.start()
rmtree(settings.JOBDIR)
temp_df = get_dataframe_from_json_files(DIR_NAME)
temp_df.to_csv("temp_df.csv", index=False)
cars_scraped = pipeline.transform(temp_df)[["car_unique_id"] + columns_list]
cars_scraped.to_csv("cars_scraped.csv", index=False)
joined_cars = pd.merge(
    cars_remote,
    cars_scraped,
    how="inner",
    on="car_unique_id",
    suffixes=("_db", "_scraped"),
)
joined_cars.to_csv("joined_cars.csv", index=False)
joined_cars = pd.read_csv("joined_cars.csv")
joined_columns_list = [column + "_db" for column in columns_list] + [
    column + "_scraped" for column in columns_list
]
boolean_list = []
joined_columns_list = [column + "_db" for column in columns_list] + [
    column + "_scraped" for column in columns_list
]
for index, row in joined_cars.iterrows():
    boolean_list.append(
        any(
            row[column + "_db"] != row[column + "_scraped"]
            and not (
                pd.isnull(row[column + "_db"])
                and pd.isnull(row[column + "_scraped"])
            )
            for column in columns_list
        )
    )
```

```
joined_cars = joined_cars[["car_unique_id"] + joined_columns_list][boolean_list]
updated_cars = joined_cars.drop(
    columns=[column + "_db" for column in columns_list]
).rename(columns={column + "_scraped": column for column in columns_list})
updated_cars.to_csv("updated_cars.csv", index=False)
updated_cars = pd.read_csv("updated_cars.csv")
# rmtree(DIR_NAME)
transaction = db_connection.begin()
try:
    updated_cars.to_sql(
        "updated_cars",
        con=db_connection,
        index=False,
        if_exists="replace",
        chunksize=10000,
    )
    table1 = "cars"
    table2 = "updated_cars"
    unique_id = "car_unique_id"
    updated_values = ", ".join(
        f"{table1}.{column} = {table2}.{column}" for column in columns_list
    )
    query = f"UPDATE {table1} SET {updated_values} FROM {table2}\
        WHERE {table1}.{unique_id} = {table2}.{unique_id}"
    db_connection.execute(query)
    db_connection.execute(f"DROP TABLE {table2}")
    transaction.commit()
except Exception:
    transaction.rollback()
db_connection.close()
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Підвищення ефективності прогнозування ціни автомобіля за допомогою машинного навчання»

Виконав: студент групи ПДМ-61 Галузов Сергій Юрійович

Керівник: к.т.н., доц., доцент кафедри ІПЗ Довженко Тимур Павлович

Київ - 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: оптимізація прогнозування за рахунок використання моделей машинного навчання.

Об'єкт дослідження: процес прогнозування ціни автомобіля.

Предмет дослідження: застосування сучасних алгоритмів машинного навчання для підвищення ефективності прогнозування ціни автомобіля.

АКТУАЛЬНІСТЬ РОБОТИ

Аналіз існуючих методів прогнозування ціни автомобіля

Метод	Особливість роботи методу	Недоліки
Лінійний регресійний аналіз	Відносно простий метод моделювання залежності ціни від факторів, в якому передбачається саме лінійна залежність ціни від факторів.	• вимагає певного рівня кореляції між змінними для точних прогнозів. • не враховує складні нелінійні залежності між факторами.
Метод часових рядів	В основі методу лежить нерозривна та періодична послідовність вимірювань ціни у часі. Це може бути щоденна, щотижнева, місячна або інша періодичність.	• вимагає стабільних та регулярних часових даних • не є ефективним для прогнозування в умовах значних змін на ринку автомобілей
Метод експертної оцінки	Оцінка ціни автомобіля передається на експертну сторону, де експерти а галузі маркета автомобілей визначають вартість автомобіля на основі свого досвіду, ситуації на ринку та внутрішнім переконанням.	• суб'єктивність та упередженість оцінок експертів • обмеженість в кількості факторів для надання оцінки
Методи машинного навчання	Будується математична модель на основі історичних даних, яка автоматично визначає паттерни в даних та узагальнює залежність між факторами та ціною автомобіля.	• вимагають великої кількості даних для навчання ефективних моделей

3

ОПИС МАТЕМАТИЧНОЇ МОДЕЛІ

X – набір вхідних ознак (властивості автомобіля, такі як рік випуску, пробіг, бренд тощо).
Y - цільова змінна (ціна автомобіля).
Θ - параметри моделі машинного навчання, які оптимізуються під час тренування моделі.

Функція витрат(Θ)= $L(Y, F(X; Θ))+Ω(Θ)$, де:

$F(X; Θ)$ - прогноз ціни автомобіля, отриманий від моделі машинного навчання за параметрами Θ.

$L(Y, F(X; Θ))$ - функція втрат, яка вимірює різницю між прогнозом і фактичною ціною.

$Ω(Θ)$ - регуляризаційна функція для контролю перенавчання моделі.

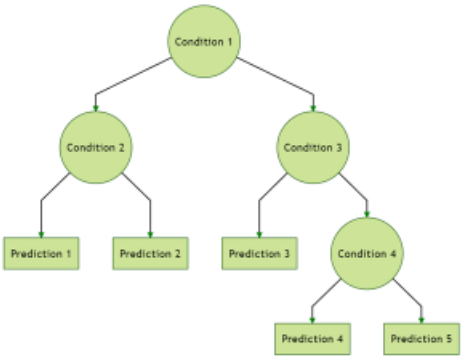
Мета математичної моделі - мінімізація функції витрат відносно параметрів Θ моделі:

$$Θ^* = \text{argmin}_Θ(L(Y, F(X; Θ))+Ω(Θ)).$$

4

ОСОБЛИВОСТІ РОБОТИ МОДЕЛІ LIGHTGBM REGRESSOR

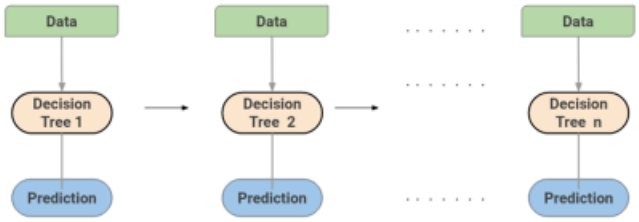
- Належить до класу машинного навчання з учителем (supervised learning)
- В основі роботи моделі машинного навчання lightgbm regressor лежить модель дерево рішень (decision tree):



5

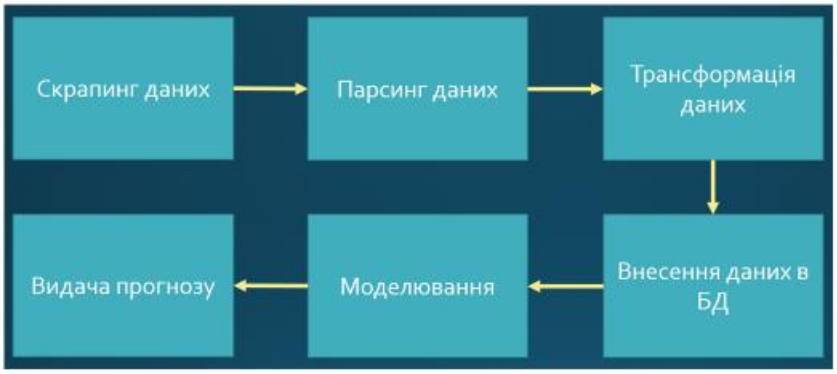
ОСОБЛИВОСТІ РОБОТИ МОДЕЛІ LIGHTGBM REGRESSOR (продовження)

- Використовує посилення (boosting) дерев рішень для поступового зменшення помилки прогнозування моделі:



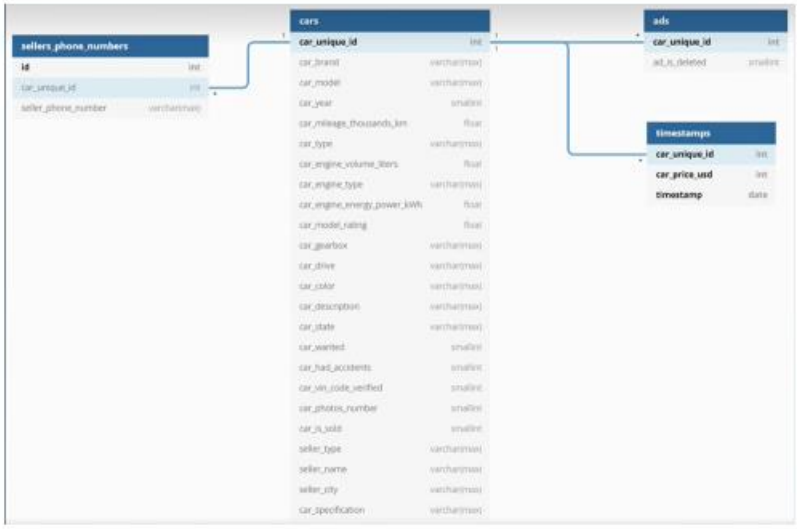
6

СХЕМА АРХІТЕКТУРИ СИСТЕМИ ПРОГНОЗУВАННЯ



7

СТРУКТУРА БАЗИ ДАНИХ СИСТЕМИ ПРОГНОЗУВАННЯ



8

МЕТРИКА РЕЗУЛЬТАТІВ ПРОГНОЗУВАННЯ МОДЕЛІ

Метрика результатів прогнозування ціни – Mean Absolute Percentage Error (MAPE).

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|A_i - F_i|}{A_i}$$

A_i is the actual value

F_i is the forecast value

n is total number of observations

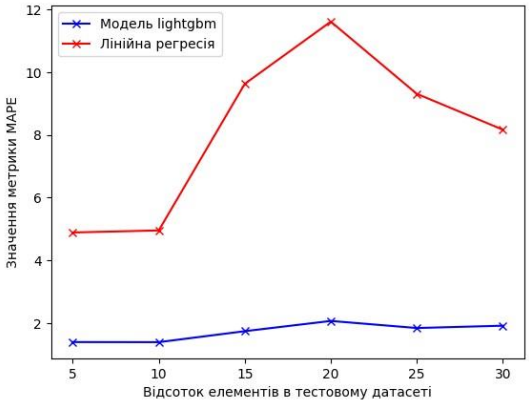
Обгрунтування використання метрики MAPE:

1. Великий розмах в ціні між різними автомобілями.
2. Відсутність нульових чи близьконульових значень ціни на автомобілі.

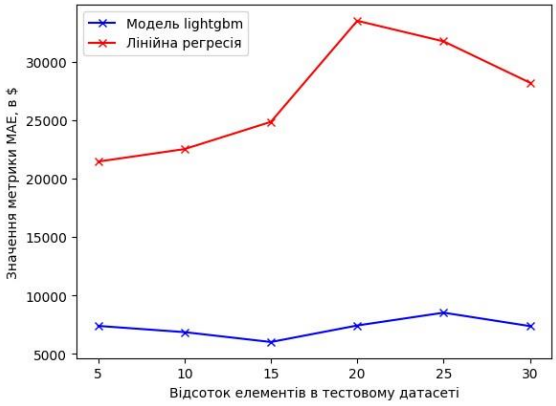
9

РЕЗУЛЬТАТ ПРОГНОЗУВАННЯ ЦІНИ

Метрика MAPE



Метрика MAE



Таким чином, маємо зменшення метрики MAPE в середньому в 4.7 рази, а метрики MAE - в 3.72 рази при використанні моделі машинного навчання lightgbm regressor в порівнянні з методом лінійної регресії для прогнозування ціни автомобіля.

ВИСНОВКИ

1. Проведено огляд існуючих методів прогнозування ціни на автомобілі, виділені їхні основні переваги та недоліки.

2. Спроектована архітектура програмного забезпечення, яке використовує методи машинного навчання для прогнозування ціни на автомобілі за їхніми характеристиками.

3. Розроблено програмне забезпечення, яке за допомогою моделі машинного навчання lightgbm regressor здатне підвищити ефективність прогнозування ціни на автомобіль.

4. Проведено аналіз ефективності розробленого рішення та підтверджено підвищення точності прогнозування ціни на автомобіль в порівнянні з існуючими методами.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Галузов С.Ю. Дослідження прогнозування ціни на автомобіль за допомогою методів машинного навчання // Конференція для країн Європи «Перспективні технології цифрової трансформації і проблеми їх впровадження в світі та в Україні» - подано до друку

Статті:

1. Галузов С.Ю. ЗАСТОСУВАННЯ МЕТОДІВ DATA SCIENCE ДЛЯ ПРОГНОЗУВАННЯ ПОПИТУ В РИТЕЙЛІ // ТЕЛЕКОМУНІКАЦІЙНІ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ. №3, 2023.