**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Сумський державний університет**
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)

18 грудня 2023 р.

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня магістр**

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційна технологія прогнозування вартості товару на основі його опису»

здобувачки групи ІН.м - 23 Павлун Тетяни Андріївни

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

Тетяна ПАВЛУН
(підпис)

Керівник,
доцент,
кандидат фізико-математичних наук,
доцент
Сергій ШАПОВАЛОВ

(підпис)

**Суми – 2023**

**Сумський державний університет**
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

# ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

**на здобуття освітнього ступеня магістра**

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-23 Павлун Тетяни Андріївни

1. Тема роботи: «Інформаційна технологія прогнозування вартості товару на основі його опису»

затверджую наказом по інституту від "6" грудня 2023 р. № 1412-VI
2. Термін здачі здобувачем кваліфікаційної роботи до 18 грудня 2023 року
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
*1) Аналіз проблеми. Огляд існуючих рішень. Огляд алгоритмів 2) Постановка задачі й формування завдань дослідження. 3) Вибір методів рішення задачі. 4) Моделювання та проєктування системи. 5) Практична реалізація.6)Аналіз результатів.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |

7. Дата видачі завдання «____» _____ 20 ___ р.
Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

# КАЛЕНДАРНИЙ ПЛАН

| № п/п | Назва етапів кваліфікаційної роботи | Термін виконання | Примітка |
|---|---|---|---|
| *1* | *Аналіз проблеми. Огляд існуючих аналогів. Огляд алгоритмів* | *06.11-13.11* | |
| *2* | *Постановка завдання та вибір методів реалізації* | *14.11-19.11* | |
| *3* | *Моделювання та проєктування Telegram бота* | *19.11-26.11* | |
| *4* | *Практична реалізація та аналіз отриманих результатів* | *27.11-11.12* | |
| *5* | *Оформлення пояснювальної записки до кваліфікаційної роботи* | *11.12-17.12* | |

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

# АНОТАЦІЯ

**Записка:** 66 стр., 31 рис., 4 табл., 3 додатки, 20 використаних джерел.

**Обґрунтування актуальності теми роботи –** тема кваліфікаційної роботи актуальна, так як присвячена вирішенню практичної задачі прогнозування вартості товару на основі його опису шляхом розробки та навчання відповідних моделей, методів та інформаційної технології з використанням алгоритмів NLP.

**Об'єкт дослідження** — прогнозування вартості товару на основі його опису.

**Мета роботи —** створення та впровадження інформаційної технології, яка здатна автоматично прогнозувати вартість товару на основі його опису. Це включає в себе розробку та навчання моделі NLP для аналізу текстового опису товарів, інтеграцію з месенджером Telegram для зручного доступу користувачів, а також тестування та оптимізацію цієї технології.

**Методи дослідження** — метод аналітичного огляду, метод порівняння та аналогій, метод моделювання, метод об’єктно-орієнтованого програмування для розробки продукту, метод емпіричного дослідження.

**Результати —** розроблено інформаційну технологію для прогнозування вартості товару на основі його опису. Комп’ютерна реалізація проєкту виконана з використанням алгоритмічної мови Python, а в основі самої технології використано адаптацію алгоритма NLP, що має застосування в ШІ.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, TELEGRAM BOT API, РЕГРЕСІЙНА МОДЕЛЬ, NLP, ML, PYTHON

# ЗМІСТ

# ВСТУП

В останні роки розвиток штучного інтелекту відкрив нові шляхи для інноваційних рішень у різних областях. Однією з таких сфер є сфера електронної комерції, де точне передбачення цін на товари на основі їх описів має першочергове значення. Здатність прогнозувати ціни на товари відіграє вирішальну роль у покращенні взаємодії з користувачами, оптимізації стратегій ціноутворення та підвищенні загальної ефективності платформ електронної комерції. Використання веб-систем на базі сучасних методів машинного навчання стає перспективним підходом до вирішення цієї проблеми.

**Актуальність.** Актуальність цієї проблеми підкреслюється динамічною природою електронної комерції, де ціни часто змінюються через ринкові коливання, коливання попиту та інші зовнішні фактори. Традиційні методи оцінки цін часто виявляються не досить гнучкими в такому стрімкому середовищі. Крім того, споживачі все більше шукають зручності під час здійснення покупок, бажаючи отримати швидку й точну інформацію про ціни на товари без необхідності ручного пошуку. Потреба в рішенні, яке могло б забезпечити актуальні та точні прогнози цін, очевидна. Хоча проблема прогнозування цін на товари на основі їхніх описів не нова, існуючі рішення часто мають обмеження щодо точності, швидкості та зручності для користувача.

Користувачам завжди властиво бажати певності. Коли вам пропонують опис товару та ставлять на це певну ціну, то завжди виникає питання – чи те, що описане так “красиво” відповідає тої вартості, що запитується? Представлена в роботі технологія дозволить в першому наближені відповідпи на це питання.

**Об'єкт дослідження.** Прогнозування вартості товару на основі його опису.

**Предмет дослідження.** Моделі та методи прогнозування ціни товару на основі його опису з використанням алгоритмів NLP.

**Гіпотеза.** Прогнозування вартості товару на основі його опису можна досягнути шляхом застосування інформаційної технології, що включає в себе розробку та навчання моделі NLP для аналізу текстового опису товарів.

**Новизна.** Це дослідження спрямоване на розробку технології, призначеної для прогнозування вартості товару на основі його опису. Використовуючи можливості машинного навчання та аналізу даних, ця технологія намагатиметься надавати користувачам точні та релевантні оцінки цін у реальному часі. Цей підхід не тільки вирішить існуючі проблеми, але й дасть можливість підвищити конкурентоспроможність платформ електронної комерції та сприяти розвитку штучного інтелекту.

**Структура.** Дана робота складається зі вступу, інформаційного огляду, постановки задачі, вибору програмних засобів та алгоритмів розроблення, тестування, висновків, списку використаних джерел та додатків. Загалом, чотири розділи, вісім підрозділів, загальний обсяг роботи 66 сторінок.

# 1 ІНФОРМАЦІЙНИЙ ОГЛЯД

## 1.1 Дослідження предметної області

Штучний інтелект (ШІ) сьогодні є невід'ємною частиною всіх великих компаній електронної комерції. З розвитком інформаційної індустрії та широкими дослідженнями в галузі штучного інтелекту за останні два десятиліття компанії почали досліджувати способи автоматизації різних видів діяльності за допомогою найсучасніших алгоритмів машинного навчання та глибоких нейронних мереж. Багато IT-гігантів і стартапів вже зробили великий стрибок у цій галузі та мають спеціальні команди та ресурси для дослідження та розробки найсучасніших програм ШІ. Платформи онлайн-роздрібної торгівлі сьогодні значною мірою керуються алгоритмами та програмами на базі ШІ. Діяльність, починаючи від управління запасами та перевірки якості на складі до рекомендацій продукту та демографічних показників продажів на веб-сайті, використовує машинне навчання в різних масштабах [1].

Щодо останніх досліджень, з метою створення суспільства, де глобальні ресурси використовуються дбайливо та де кожен може жити заможно, компанія розробила додаток для блошиного ринку «Mercari» в Японії та Сполучених Штатах, який дозволяє людям легко та безпечно купувати та продавати товари. Завдання Mercari полягало у тому, щоб побудувати алгоритм, який автоматично пропонуватиме правильні ціни на продукти продавцям у його додатку [2].

Передбачити ціну продукту є складним завданням, оскільки дуже схожі продукти, які мають незначні відмінності, наприклад різні назви брендів, додаткові характеристики, якість, попит на продукт тощо, можуть мати дуже різні ціни. Наприклад, один із наведених нижче светрів коштував 335 доларів, а інший – 9,99 доларів (див. рис. 1.1). Досить складно вгадати, який з них скільки коштував.

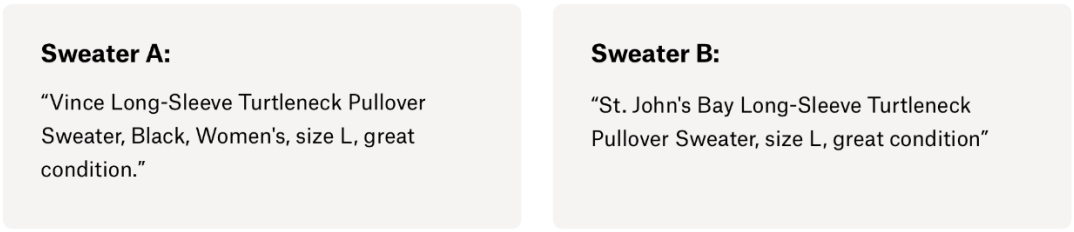


Рисунок 1.1 — Приклад опису товарів

Важко визначити, скільки щось насправді коштує. Маленькі деталі можуть означати великі відмінності в ціні.

Передбачити ціну стає ще складніше, коли існує величезний асортимент товарів, що характерно для більшості платформ онлайн-магазинів. Продавцям Mercari дозволено розміщувати в додатку майже будь-що. Дуже складно передбачити ціну майже всього, що розміщено на онлайн-платформах.

Ця постановка задачі була завантажена на змагання Kaggle. У цьому конкурсі Mercari ставив перед учасниками завдання створити алгоритм, який автоматично пропонує правильні ціни на продукти.

Надані дані складаються з введених користувачем текстових описів своїх продуктів, включаючи такі деталі, як назва категорії продукту, назва бренду, стан товару. Mercari надає введені користувачем текстові описи своїх продуктів, включаючи такі деталі, як назва категорії продукту, назва бренду та стан товару. Використовуючи ці дані, необхідно розробити модель, яка якомога точніше прогнозує ціну продукту, розміщеного на Mercari. Це виглядає як стандартна задача регресії [3].

Отже, створення сервісу, який прогнозує вартість товару за його описом, може бути корисним і цікавим завданням з кількох причин:

1. Покращення користувачного досвіду. Додаток може полегшити процес пошуку та порівняння товарів, дозволяючи користувачам отримувати швидкі та зручні прогнози цін на товари на основі текстового опису. Це полегшує процес прийняття рішень для покупців.

2. Аналіз ринку та конкурентоспроможність. Такий додаток може бути

корисним для бізнесів, які бажають аналізувати ціни на товари на ринку і визначити свою конкурентоспроможність. Він може допомогти визначити оптимальні ціни для продуктів та конкурувати на ринку.

3. Підвищення продажів. Додаток може допомогти продавцям залучати більше покупців, привертаючи їх увагу до конкурентоспроможних цін на товари.

4. Навчання та дослідження. Розробка такого додатку може слугувати навчальною ціллю, де можна вивчати та застосовувати різні методи машинного навчання, NLP (обробки природної мови) та аналізу даних [4].

5. Спрощення електронної комерції. Додаток може бути корисним для онлайн-магазинів та платформ, які бажають автоматизувати процеси фіксації цін та прогнозування для їхніх продуктів.

За останні роки чат-боти набули значної популярності як потужний інструмент для вирішення різноманітних бізнес-завдань за допомогою штучного інтелекту. Їх актуальність і ефективність у покращенні бізнес-операцій підкреслюється кількома ключовими факторами.

Розробка чат-боту для прогнозування вартості товару на основі його опису має численні переваги, які роблять її найкращим вибором для вирішення цієї проблеми. Ось кілька аргументів, що підтверджують цю точку зору:

– Миттєва комунікація: чат-боти забезпечують миттєве спілкування з користувачами. Користувачі можуть одержувати прогнози ціни без очікування, що особливо важливо в галузі електронної комерції, де швидкість і зручність мають першочергове значення.

– Легкий доступ: Телеграм — популярний месенджер, доступний на різних платформах, включаючи мобільні пристрої і веб-версію. Користувачам не потрібно встановлювати додаткові програми або відвідувати веб-сайти для отримання інформації про ціни.

– Зручність спілкування: Телеграм-боти надають зручну платформу для користувачів описувати товари в текстовому форматі, а також отримувати

інформацію про їх вартість за допомогою текстових запитів. Це набагато більш зручно, ніж введення запитів у веб-форми або інші інтерфейси.

– Можливості розширення: Телеграм надає можливість інтеграції з іншими сервісами та додатками через API. Це означає, що можна розширити функціональність бота, додавши нові можливості, такі як перевірка наявності товару або відстеження змін вартості.

– Захист приватності: Телеграм має добре розвинуту систему шифрування та захисту приватності, що важливо для користувачів, які надають особисті дані, такі як опис товару. Це сприяє підвищенню рівня довіри користувачів.

– Спільнота і розповсюдженість: Значна кількість користувачів вже використовує Телеграм, що створює готову аудиторію для впровадження бота. Це сприяє поширенню та прийняттю нового сервісу.

– Аналітика і зворотний зв'язок: Телеграм-боти можуть забезпечити аналітику щодо взаємодії користувачів з ботом і їхніми запитами, що може бути використано для покращення якості обслуговування та розвитку додаткових функцій.

Усі ці фактори роблять розробку Телеграм-боту найкращим рішенням для вирішення проблеми прогнозування вартості товарів на основі їх опису в галузі електронної комерції. Такий бот може покращити зручність користувачів і ефективність платформи, а також сприяти розвитку та вдосконаленню індустрії електронної комерції.

## 1.2 Огляд аналогів розроблюваного програмного продукту

Сервіси, які використовують прогнозування цін на товари, мають свої плюси та мінуси, і вони використовують різні методи для прогнозування цін. Ось загальна інформація про кілька таких сервісів і їхні переваги та недоліки:

– Price.com. Цей додаток дозволяє користувачам знайти найкращу ціну на товари на різних інтернет-магазинах та надає прогнози цін (рис. 1.2). До переваг можна віднести: простий та зручний інтерфейс для знаходження

найкращих цін на товари в різних інтернет-магазинах; надає прогнози цін, які допомагають користувачам визначити оптимальний момент для покупки. Недоліки: додаток зазвичай спирається на історичні дані цін і не надає деталі про те, як саме прогнозуються ціни; модель прогнозування не завжди зовнішнім користувачам доступна.

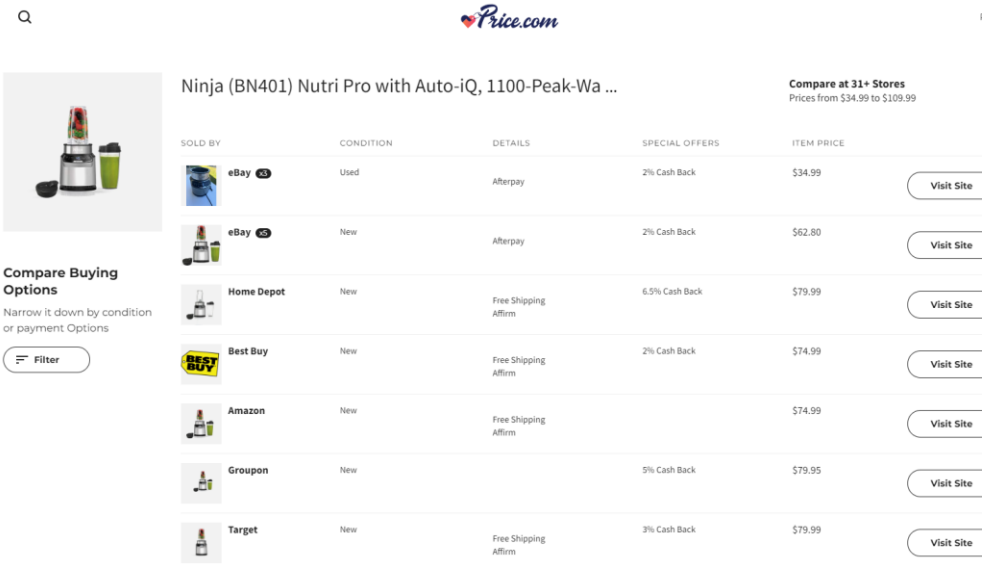


Рисунок 1.2 — Приклад використання Price.com

– Honey. Honey є розширенням браузера та додатком для мобільних пристроїв, яке допомагає знаходити знижки та купони для товарів, а також надає прогнози цін (рис. 1.3). Плюси: допомагає користувачам знаходити знижки та купони для товарів, що може значно зменшити загальну вартість покупок; включає функцію прогнозування цін, яка допомагає користувачам визначити, чекати на знижку чи купувати зараз. Суттєвий мінус: Honey спеціалізується на знаходженні знижок, і його прогнози цін можуть бути обмеженими щодо покриття різних інтернет-магазинів.

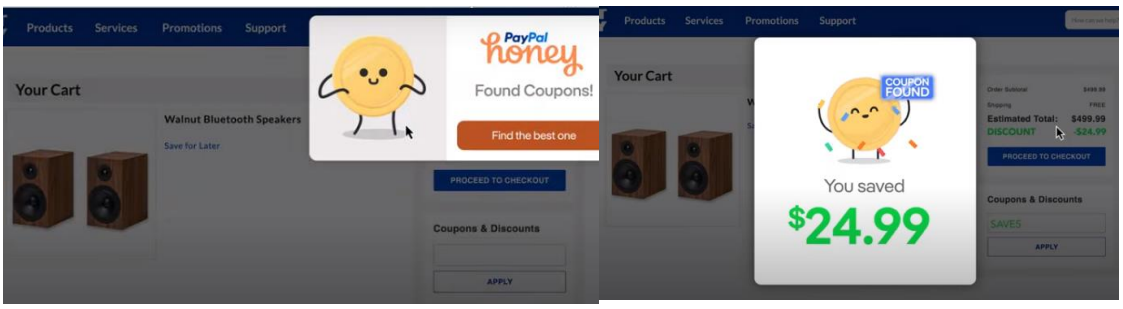


Рисунок 1.3 — Приклад використання Honey

– CamelCamelCamel. Це додаток для відстеження цін на Amazon, який надає графіки цінової динаміки та прогнози цін (рис. 1.4). Плюси: спеціалізується на відстеженні цін на Amazon, і відображає історичні ціни на товари на цій платформі; надає графіки цінової динаміки та прогнози цін на основі історичних даних. Мінуси: обмежений платформою Amazon, тобто прогнозування цін стосується лише цього магазину; користувачам може бути складно користуватися для товарів, які не представлені на Amazon.

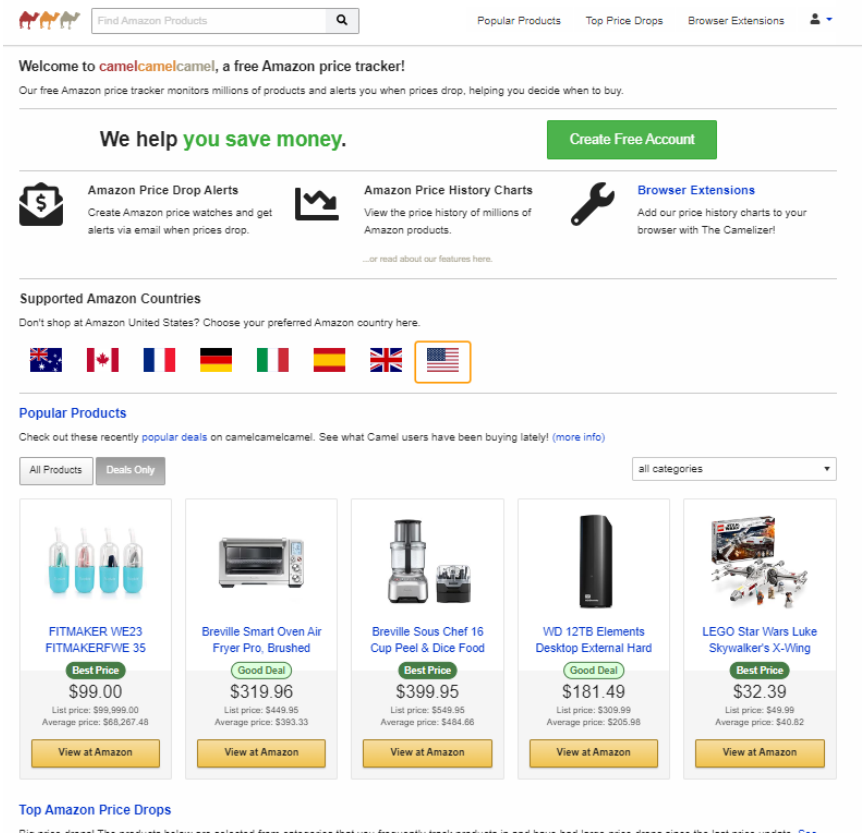


Рисунок 1.4 — Головна сторінка CamelCamelCamel

Якщо розглядати загальний процес прогнозування цін на таких сервісах, він зазвичай базується на аналізі історичних цін на товари, і користувачі зазвичай можуть використовувати наступні кроки:

1. Користувач знаходить товар, який його цікавить, і вводить інформацію про нього (або використовує плагін браузера для автоматичного виявлення товарів). Сервіс аналізує історичні ціни цього товару та визначає зміни цін з часом.

2. На основі аналізу історичних даних і інших факторів, таких як сезонність, знижки, попит тощо, сервіс генерує прогнозну ціну товару в майбутньому.

3. Користувач отримує цю прогнозну ціну та може вирішити, чекати знижки або купувати зараз.

Зважаючи на тему дослідження був проведений аналіз ботів, подібного призначення:

– Auction Price Prediction Bot: Якщо користувач планує взяти участь в інтернет-аукціоні, цей бот може аналізувати опис товару та історію аукціонів, щоб зробити приблизний прогноз ціни, за яку товар може бути проданий. Переваги: Надає користувачам приблизний прогноз ціни на аукціоні. Недоліки: Прогнози можуть бути неточними через змінність аукціонних умов.

– Real Estate Price Estimation Bot: Для нерухомості, бот може приймати опис власності, включаючи параметри, локацію та інші характеристики, і надавати користувачу оцінку ціни на основі аналізу ринкових даних (рис. 1.5). Переваги: Допомагає користувачам отримати загальне уявлення про вартість нерухомості. Недоліки: Складно враховувати унікальні характеристики кожного об'єкта, велика похибка в оцінці.

– Used Car Price Prediction Bot: Даний бот може бути корисним для тих, хто шукає вживані автомобілі. Користувач може надіслати опис авто, і бот надасть інформацію про очікувану вартість, враховуючи марку, модель, рік випуску і стан машини. Переваги: Допомагає покупцям та продавцям

отримати ідею про ціну автомобіля. Недоліки: Точність прогнозів може залежати від точності наданих даних.

– Antique Valuation Bot: Якщо користувач має антикварний товар, він може надіслати опис предмету і одержати оцінку його вартості на основі аналізу подібних антикваріатів, які були продані на аукціонах або в антикварних магазинах. Переваги: Надає інформацію про вартість антикварних товарів, яка може бути корисною для колекціонерів. Недоліки: Важко враховувати унікальність та історичний контекст кожного антикваріату.

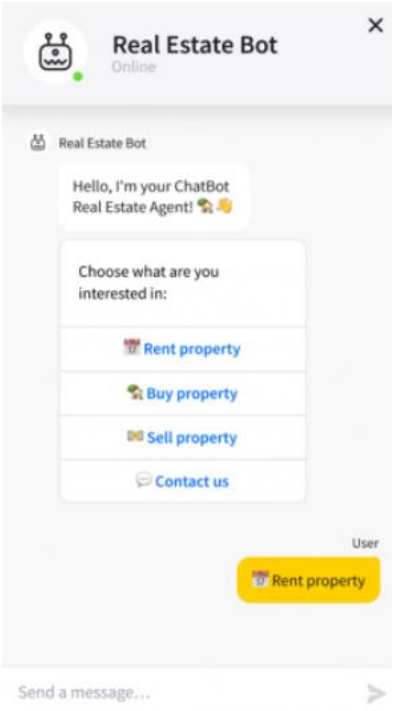


Рисунок 1.5 — Приклад використання Real Estate Price Estimation Bot

Це декілька прикладів вже створених ботів, які здатні вирішувати задачу прогнозування вартості товару на основі його текстового опису. Такі боти можуть бути важливими інструментами для користувачів, які шукають найкращі пропозиції, з точки зору ціна-якість та потребують точної інформації про ціни. Для порівняння вказаних вище ботів, була створена таблиця 1.1, яка дає змогу зрозуміти переваги та недоліки вже створених продуктів та визначитись чи перспективним є провадження власного продукту.

Таблиця 1.1 — Порівняння ботів, подібного призначення

| Бот | Переваги | Недоліки |
|---|---|---|
| Auction Price Prediction Bot | Надавання приблизного прогнозу ціни на аукціоні | Неточність прогнозів через змінність аукціонних умов |
| Real Estate Price Estimation Bot | Надає загальне уявлення про вартість нерухомості | Складність з врахуванням унікальних характеристик об’єкту, велика похибка в оцінці |
| Used Car Price Prediction Bot | Надавання ідеї про ціну авто | Точність прогнозу залежить від точності вхідних даних |
| Antique Valuation Bot | Корисний для колекціонерів | Складно враховувати унікальність та історичний контекст кожного антикваріату |

Отже, розробка власного боту може бути кращим рішенням ніж використання готових з декількох причин:

– Можливість об’єднання різного функціоналу: власний бот може поєднувати функціональність кількох вищевказаних ботів, з додаванням нових можливостей, це дозволить користувачам отримати більше різноманітних послуг.

– Навчання на основі нових даних введених користувачем: у створеного боту може бути можливість навчатися на основі реальних даних та коригувати результати відповідно до зміни тенденцій.

– Зовнішні інтеграції: можна легко інтегрувати бот з іншими джерелами даних, такими як бази даних магазинів, що може зробити його досить поширеним та популярним.

– Інтерфейс користувача: створення інтуїтивно зрозумілого та простого у використанні інтерфейсу, який полегшить введення даних та підвищить рівень інтерактивності з користувачем.

З огляду на ці плюси та здатність бота адаптуватися до специфічних потреб, він може виявитися ідеальним вибором як для індивідуальних користувачів, так і для комерційних організацій, за умови ретельного проектування та ефективного управління.

### 1.3 Постановка задачі

Отже, метою роботи є створення та впровадження інформаційної технології прогнозування вартості товару на основі його опису, що включає в себе навчання моделі NLP для аналізу текстового опису товарів, інтеграцію з користувачем шляхом взаємодії з телеграм-ботом.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. Вибір засобів для функціональної реалізації;
2. Огляд та обрання моделей машинного навчання та моделей глибокого навчання;
3. Структурно-функціональне моделювання;
4. Реалізація та навчання моделі прогнозування:
    - Збір та підготовка даних;
    - Токенізація та векторизація тексту;
    - Оцінка моделі, навчання моделі, розробка моделі.
5. Розробка бота:
    - Створення бота;
    - Реалізація основних команд, логіки бота та інтерфейсу керування;
    - Інтеграція розробленої моделі прогнозування вартості товару.
6. Проведення тестування створеної інформаційної технології.

# 2 ВИБІР ПРОГРАМНИХ ЗАСОБІВ. ПРОЄКТУВАННЯ ТЕХНОЛОГІЇ

## 2.1 Вибір засобів програмування

Python є популярним інструментом для створення телеграм-ботів завдяки багатій екосистемі бібліотек для роботи з Telegram API. Один з найпопулярніших фреймворків для розробки телеграм-ботів на Python — це Telebot. Його можна використовувати для створення інтерфейсу телеграм-боту, який прийматиме описи товарів від користувачів та передаватиме їх до моделі для прогнозування ціни.

Python також має багато бібліотек для машинного навчання і моделей регресії, що можуть досить суттєво спростити розробку моделі для прогнозування ціни на основі його опису [5].

Отже, мова Python є гнучким і потужним вибором для вирішення цієї задачі, особливо коли необхідно поєднати її з API Телеграм-боту для взаємодії з користувачами.

## 2.2 Аналіз та вибір алгоритмів розроблення

Розробка програмного продукту для прогнозування вартості товару на основі інформації, яку користувач надає для цього — це завдання, яке може бути вирішене різними способами, залежно від обраного підходу та технологічного стеку.

Задача прогнозування ціни товару на основі його опису є задачею регресії. В регресійних задачах метою є передбачення числового значення або кількісної змінної (у цьому випадку – ціни) на основі вхідних ознак (у цьому випадку опису товару). Регресія допомагає встановити залежність між вхідними факторами і вихідною змінною для прогнозування числових значень, таких як ціни [6].

Для вирішення задачі регресії на основі опису товару, спершу необхідно перетворити опис товару на фічі (ознаки), щоб мати дані, які можна

використовувати для регресійної моделі.

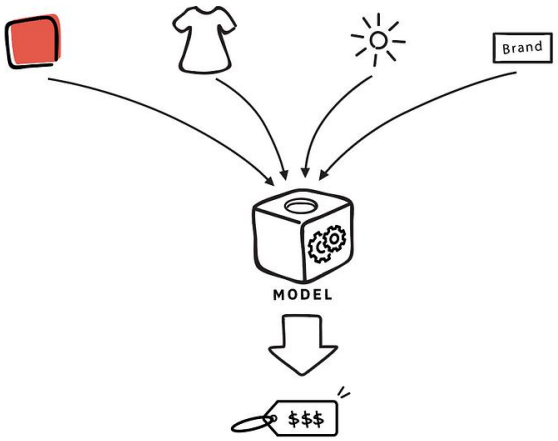


Рисунок 2.1 — Схематичне зображення задачі прогнозування

Процес кодування опису в фічі включає в себе наступні кроки [7]:

1. Токенізація: розділення тексту опису на окремі слова або токени.

2. Векторизація: перетворення слів або токенів на числові представлення.

3. Інженерія ознак: додати додаткові ознаки, які можуть бути корисними для прогнозування ціни, такі як кількість слів у описі, ключові слова, аналіз тональності тощо.

4. Масштабування і нормалізація: зазвичай важливо масштабувати фічі, щоб їхні значення були в одному діапазоні, що може поліпшити роботу регресійної моделі.

Далі будуть розглянуті інструменти для токенізації текстового контенту, такі як NLTK, TextBlob, Spacy, Gensim і Keras.

NLTK (Natural Language Toolkit) — це бібліотека Python з відкритим кодом для обробки природної мови. Він має прості у використанні інтерфейси для більш ніж 50 корпусів та лексичних ресурсів, таких як WordNet, а також набір бібліотек обробки тексту для класифікації, токенізації, стеммінгу та тегування [8].

Можна легко токенізувати речення та слова тексту за допомогою модуля

токенізації NLTK.

Спочатку необхідно імпортувати відповідні функції з бібліотеки NLTK:

```
import nltk
from nltk.tokenize import (word_tokenize,
                           sent_tokenize,
                           TreebankWordTokenizer,
                           wordpunct_tokenize,
                           TweetTokenizer,
                           MWETokenizer)
text="Hope, is the only thing stronger than fear! #Hope #Amal.M"
```

Токенізатор слів та речень:

```
print(word_tokenize(text))
```

```
['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear', '!', '#', 'Hope', '#', 'Amal.M']
```

```
print(sent_tokenize(text))
```

```
['Hope, is the only thing stronger than fear!', '#Hope #Amal.M']
```

Токенізатор на основі пунктуації.

Цей токенізатор розбиває речення на слова на основі пробілів і розділових знаків.

```
print(wordpunct_tokenize(text))
```

```
['Hope', ',', 'is', 'the', 'only', 'thing', 'stronger', 'than', 'fear', '!', '#', 'Hope', '#', 'Amal', '.', 'M']
```

Токенізатор Treebank Word

Цей токенізатор включає безліч загальних правил токенізації англійських слів. Він відокремлює завершальні фразу розділові знаки, наприклад (?!.;,), від сусідніх токенів і зберігає десяткові числа як один токен. Крім того він містить правила для англійських скорочень.

```
text="What you don't want to be done to yourself, don't do to others..."
tokenizer= TreebankWordTokenizer()
print(tokenizer.tokenize(text))
```

```
['What', 'you', 'do', "n't", 'want', 'to', 'be', 'done', 'to', 'yourself', ',', 'do', "n't", 'do', 'to', 'others',
'...']
```

Токенізатор MWET

Токенізатор багатослівних виразів NLTK (MWETokenizer) надає функцію add_mwe(), яка дозволяє користувачеві вводити кілька словесних виразів перед використанням токенізатора в тексті. Простіше кажучи, він може поєднувати вирази з кількох слів в окремі токени [9].

TextBlob — це бібліотека Python для обробки текстових даних. Він надає послідовний API для занурення в загальні завдання обробки природної мови

(NLP), такі як тегування частин мови, виділення іменників, аналіз настроїв, класифікація, переклад тощо.

Інсталяції TextBlob і корпусу NLTK:

```
$pip install -U textblob
$python3 -m textblob.download_corpora
```

У коді нижче зображено токенізацію слів за допомогою бібліотеки TextBlob:

```
from textblob import TextBlob
text= " But I'm glad you'll see me as I am. Above all, I wouldn't want people to think that I want to prove anything

blob_object = TextBlob(text)
# Word tokenization of the text
text_words = blob_object.words
# To see all tokens
print(text_words)
# To count the number of tokens
print(len(text_words))
```

```
['But', 'I', "'m", 'glad', 'you', "'ll", 'see', 'me', 'as', 'I', 'am', 'Above', 'all', 'I', 'would', "n't", 'want',
'people', 'to', 'think', 'that', 'I', 'want', 'to', 'prove', 'anything', 'I', 'do', "n't", 'want', 'to', 'prove', '
anything', 'I', 'just', 'want', 'to', 'live', 'to', 'cause', 'no', 'evil', 'to', 'anyone', 'but', 'myself', 'I', 'h
ave', 'that', 'right', 'have', "n't", 'I', 'Leo', 'Tolstoy']
55
```

SpaCy — це бібліотека Python з відкритим кодом, яка аналізує та розуміє великі обсяги тексту. Завдяки доступним моделям, що обслуговують певні мови (англійська, французька, німецька тощо), він справляється з завданнями НЛП із найефективнішою реалізацією загальних алгоритмів.

SpaCy tokenizer забезпечує гнучкість у визначенні спеціальних токенів, які не потребують сегментації, або мають бути сегментовані за спеціальними правилами для кожної мови [10].

Перед початком роботи зі spaSy, потрібно встановити його, завантажити дані та моделі для англійської мови.

```
$ pip install spacy
$ python3 -m spacy завантажити en_core_web_sm
```

```
text= "All happy families are alike; each unhappy family is unhappy in its own way!!!👌👌 #Leo Tolstoy "
doc = nlp(text)
for token in doc:
    print(token, token.idx)
```

```
All 0
happy 4
families 10
are 19
alike 23
; 28
each 30
unhappy 35
family 43
is 50
unhappy 53
in 61
its 64
own 68
way 72
! 75
! 76
! 77
👌 78
👌 79
# 81
Leo 82
Tolstoy 86
```

Gensim — це бібліотека Python для тематичного моделювання, індексування документів і пошуку подібності з великими корпусами. Цільовою аудиторією є спільнота обробки природної мови (NLP) і пошуку інформації (IR). Він пропонує корисні функції для токенізації.

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.text import text_to_word_sequence
```

```
ntoken= Tokenizer(num_words=20)
```

```
text="All happy families are alike; each unhappy family is unhappy in its own wa
ntoken.fit_on_texts(text)
list_words= text_to_word_sequence(text)
print(list_words)
```

```
['all', 'happy', 'families', 'are', 'alike', 'each', 'unhappy', 'family', 'is',
'unhappy', 'in', 'its', 'own', 'way', 'leo', 'tolstoy', '❤️❤️']
```

Також токенізацію можна здійснити за допомогою великих мовних моделей, таких як, LLaMA (Large Language Model Meta AI) — це найсучасніша базова велика мовна модель, розроблена, щоб допомогти дослідникам просувати свою роботу в підгалузі ШІ (може бути використана для різних завдань NLP, включаючи токенізацію) [13]. LLaMA використовує архітектуру трансформера, стандартну архітектуру для мовного моделювання з 2018 року.

Наступним кроком в процесі кодування опису в фічі є векторизація. Існують різні методи векторизації, такі як TF-IDF (Term Frequency-Inverse Document Frequency), Word Embeddings (наприклад, Word2Vec або GloVe), Bag of Words (BoW) та інші [12].

Term Frequency-Inverse Document Frequency (TF-IDF) — це один з найпоширеніших і найпотужніших методів для отримання ознак з текстових даних. TF-IDF обчислює важливість кожного слова у документі щодо кількості його вживань у цьому документі та у всій колекції текстів. Цей метод дозволяє виділити ключові слова та зрозуміти, які слова мають більшу вагу для певного документа у контексті всієї колекції.

TF (Частота терміна) означає, наскільки часто певне слово з'являється у цьому документі. Таким чином, TF вимірює важливість слова у контексті окремого документа [14].

IDF (Зворотна частота документа) вимірює, наскільки унікальним є слово по всій колекції документів. Слова, які з'являються у більшості документів, мають низький IDF, оскільки вони не вносять великої інформаційної цінності.

Формула TF-IDF комбінує поняття TF та IDF, щоб обчислити важливість кожного слова у кожному документі. Формально, формула виглядає так:

$$TF - IDF(t,d) = TF(t,d) * IDF(t)$$

де:

$TF(t,d)$ — Частота терміна (TF) для слова «t» у документі «d».

$IDF(t)$ — Зворотна частота документа (IDF) для слова «t».

Застосування TF-IDF у задачах аналізу текстів:

1. Виділення ключових слів та термінів. Один із способів зробити це — вибрати топ-N слів із найбільшими значеннями TF-IDF. Приклад на Python:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Приклад текстових даних
documents = [
    "Машинне навчання - це цікава область.",
    "Навчання з вчителем - ключовий аспект машинного навчання.",
    "Область NLP також пов'язана з машинним навчанням."
]

# Створення об'єкту TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()

# Застосування TF-IDF до текстових даних
tfidf_matrix = tfidf_vectorizer.fit_transform(documents)

# Отримання списку ключових слів та їх значення TF-IDF для першого
документу
feature_names = tfidf_vectorizer.get_feature_names_out()
tfidf_scores = tfidf_matrix.toarray()[0]

# Сортування слів за значенням TF-IDF
sorted_keywords = [word for _, word in sorted(zip(tfidf_scores,
feature_names), reverse=True)]

print("Ключові слова:", sorted_keywords)
```

2. Кластеризація та категоризація текстових даних. Кластеризація може допомогти виявити загальні теми та зрозуміти структуру даних. Практичні приклади застосування кластеризації: статті можуть бути автоматично категоризовані за темами з використанням кластеризації на основі TF-IDF; кластеризація повідомлень або постів користувачів у соціальних мережах може допомогти створити персональні стрічки новин. Розглянемо приклад на Python:

```
from sklearn.cluster import KMeans

# Застосування кластеризації KMeans до матриці TF-IDF
num_clusters = 2
kmeans = KMeans(n_clusters=num_clusters, random_state=0)
kmeans.fit(tfidf_matrix)

# Показати приклади документів у кожному кластері
for cluster_id in range(num_clusters):
    cluster_indices = np.where(kmeans.labels_ ==
cluster_id)[0]
    print(f"Кластер {cluster_id + 1}:")
    for idx in cluster_indices:
        print(documents[idx])
    print("--------")
```

GloVe — це алгоритм неконтрольованого навчання, розроблений Стенфордським університетом для створення вбудованих слів шляхом агрегування глобальної матриці спільного використання слів із корпусу [15]. GloVe означає глобальні вектори для представлення слів. Отримані вкладення показують цікаві лінійні підструктури слова у векторному просторі. Приклади лінійних підструктур зображено на рисунку 2.2.

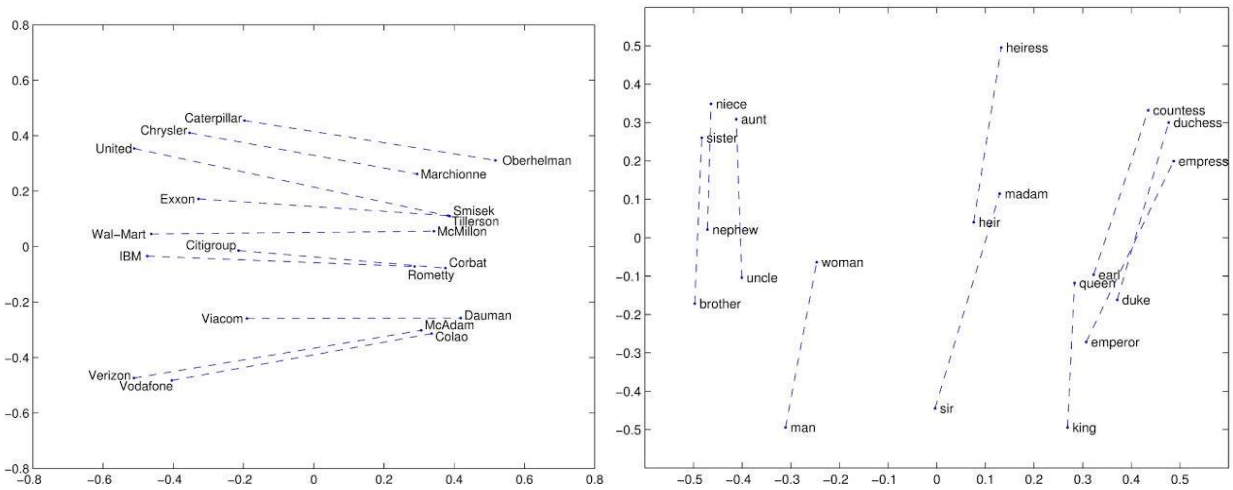


Рисунок 2.2 — Приклади лінійних підструктур

Bag of Words (BOW) — це представлення, яке перетворює довільний текст на вектори фіксованої довжини шляхом підрахунку кількості повторень кожного слова. Приклад роботи можна розглянути далі. Припустімо, необхідно векторизувати наступне [11]:

- the cat sat
- the cat sat in the hat
- the cat with the hat

Кожен із них називатиметься текстовим документом .

Спочатку потрібно визначити словниковий запас, який є набором усіх слів, знайдених у наборі документів. У наведених вище документах є лише слова: the, cat, sat, in, the, hat та with.

Щоб векторизувати наші документи, все, що нам потрібно зробити, це підрахувати, скільки разів з’являється кожне слово (див. табл. 2.1).

Таблиця 2.1 — Приклад візуального представлення розподілення слів

| Документ | the | cat | sat | in | hat | with |
|---|---|---|---|---|---|---|
| the cat sat | 1 | 1 | 1 | 0 | 0 | 0 |
| the cat sat in the hat | 2 | 1 | 1 | 1 | 1 | 0 |
| the cat with the hat | 2 | 1 | 0 | 0 | 1 | 1 |

Тепер у нас є вектори довжини 6 для кожного документа:

– the cat sat: [1, 1, 1, 0, 0, 0]

– the cat sat in the hat: [2, 1, 1, 1, 1, 0]

– the cat with the hat: [2, 1, 0, 0, 1, 1]

BOW як буквальний мішок слів: він лише повідомляє, які слова зустрічаються в документі, а не те, де вони зустрічаються.

Приклад реалізації BOW на Python використовуючи клас Tokenizer Keras:

```
from keras.preprocessing.text import Tokenizer

docs = [
  'the cat sat',
  'the cat sat in the hat',
  'the cat with the hat',
]

## Step 1: Determine the Vocabulary
tokenizer = Tokenizer()
tokenizer.fit_on_texts(docs)
print(f'Vocabulary: {list(tokenizer.word_index.keys())}')

## Step 2: Count
vectors = tokenizer.texts_to_matrix(docs, mode='count')
print(vectors)
```

Запуск цього коду дає нам:

```
Vocabulary: ['the', 'cat', 'sat', 'hat', 'in', 'with']
[[0. 1. 1. 1. 0. 0. 0.]
 [0. 2. 1. 1. 1. 1. 0.]
 [0. 2. 1. 0. 1. 0. 1.]]
```

Вектори тут мають довжину 7 замість 6 через додатковий 0 елемент на початку. Це несуттєва деталь — Keras резервує індекс 0 і ніколи не присвоює його жодному слову.

Незважаючи на відносно базову модель, BOW часто використовується для завдань обробки природної мови (NLP), таких як класифікація тексту. Його сильні сторони полягають у його простоті: обчислення коштує недорого, а іноді простіше краще, коли інформація про позиціонування чи контекст не є актуальною.

Після кодування опису в фічі можна буде використовувати ці фічі як вхідні дані для регресійної моделі, такої як лінійна регресія, дерево рішень, градієнтний бустінг, нейронна мережа тощо. Регресійна модель буде намагатися встановити залежність між цими фічами та ціною товару, щоб прогнозувати числове значення ціни на основі вхідних даних.

Оскільки це проблема регресії, у нас є можливість реалізувати широкий спектр моделей машинного і глибокого навчання для вирішення задачі прогнозування, хоча і мають свої переваги та недоліки [16].

Моделі машинного навчання:

1. Ridge Regressor. Ridge Regressor — це метод регресії, який допомагає прогнозувати значення цільової змінної на основі ознак. Він використовує регуляризацію (L2-норма) для зменшення ваг ознак і запобігання перенавчанню. Це допомагає зробити модель більш стійкою до шуму в даних, але вона все ще враховує всі ознаки. Параметр alpha контролює рівень регуляризації, і його варто налаштовувати для кожного завдання.

- Плюси: Добре підходить для регресійних завдань, допомагає уникнути перенавчання.
- Мінуси: Покращує точність лише при великій кількості ознак.

2. SGD Regressor (Stochastic Gradient Descent Regressor) — це метод регресії, який використовує стохастичний градієнтний спуск для навчання моделі. Він оновлює ваги (коефіцієнти) моделі швидко та ітеративно,

використовуючи невеликі випадкові підвибірки даних. Це допомагає підігнати модель до даних, особливо коли набір даних великий. SGD Regressor підтримує різні функції втрат і регуляризацію, що робить його гнучким для різних завдань прогнозування.

- Плюси: Швидкий і підходить для великих наборів даних.
- Мінуси: Чутливий до початкового значення швидкості навчання.

3. Support Vector Regressor (SVR) — це метод регресії, який використовує опорні вектори для прогнозування значень цільової змінної. Він намагається знайти границю, яка максимально приближається до точок даних, при цьому допускаючи помилки. SVR використовує ядрові функції для перетворення ознак та знаходить границю регресії так, щоб якомога більше точок лежали в межах заданого епсилон-трубки. Цей метод допомагає уникнути перенавчання та створює гнучку модель регресії, яка добре працює в задачах прогнозування.

- Плюси: Добре працює з нелінійними залежностями між ознаками та ціною.
- Мінуси: Вимагає підбору гіперпараметрів.

Структура методу регресії Support Vector Regressor зображена на рис. 2.3.

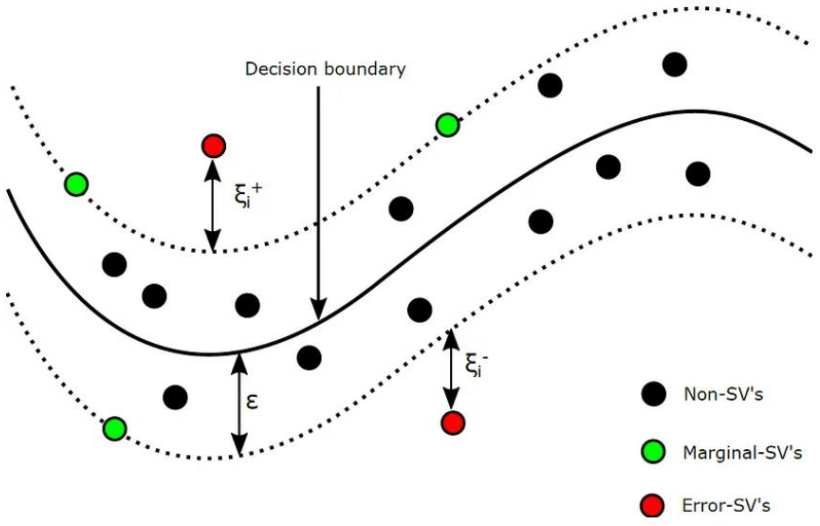


Рисунок 2.3 — Структура методу регресії SVR

4. Decision Tree Regressor — це метод регресії, який використовує дерево рішень для прогнозування значень цільової змінної. Модель розбиває

набір даних на багато рішень на основі ознак, створюючи гілки дерева рішень. Кожний листовий вузол дерева містить прогнозоване значення. Decision Tree Regressor намагається мінімізувати середньоквадратичну помилку (MSE) шляхом оптимального розділення даних на групи.

- Плюси: Простий для інтерпретації, враховує нелінійність в даних.
- Мінуси: Схильний до перенавчання, може бути нестійким.

5. Ensembles (Stacking, Voting):

- Плюси: Поєднують декілька моделей для покращення точності.
- Мінуси: Вимагають більше ресурсів та зусиль для конфігурації.

Моделі глибокого навчання:

1. Long Short Term Memory (LSTM) — це тип рекурентних нейронних мереж (RNN), який використовується для задач обробки послідовностей, включаючи прогнозування часових рядів та послідовностей. Основна особливість LSTM полягає в здатності ефективно моделювати довгострокові залежності в даних, уникаючи проблем згасаючого або вибухаючого градієнту, які властиві звичайним RNN.

- Плюси: Добре підходить для обробки послідовних даних, таких як текст або часові ряди.
- Мінуси: Вимагає велику кількість даних для навчання, може бути обчислювально вимогливим.

Схематичне зображення LSTM представлено на рис. 2.4.

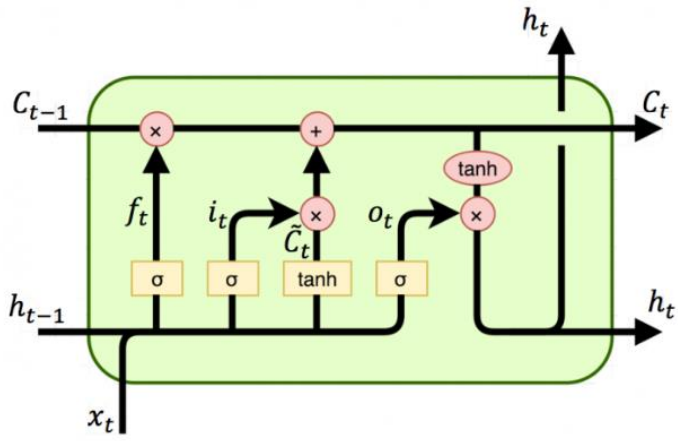


Рисунок 2.4 — Схема для типу LSTM

2. Convolutional Neural Network (CNN):

– Плюси: Відмінно підходить для обробки зображень, враховує локальні особливості.

– Мінуси: Також може бути обчислювально вимогливим, вимагає багато даних.

3. Gated Recurrent Unit (GRU):

– Плюси: Ефективний для обробки послідовностей, спрощений в порівнянні з LSTM.

– Мінуси: Може бути складним для роботи з дуже складними послідовностями.

Схематичне зображення GRU представлено на рис. 2.5.

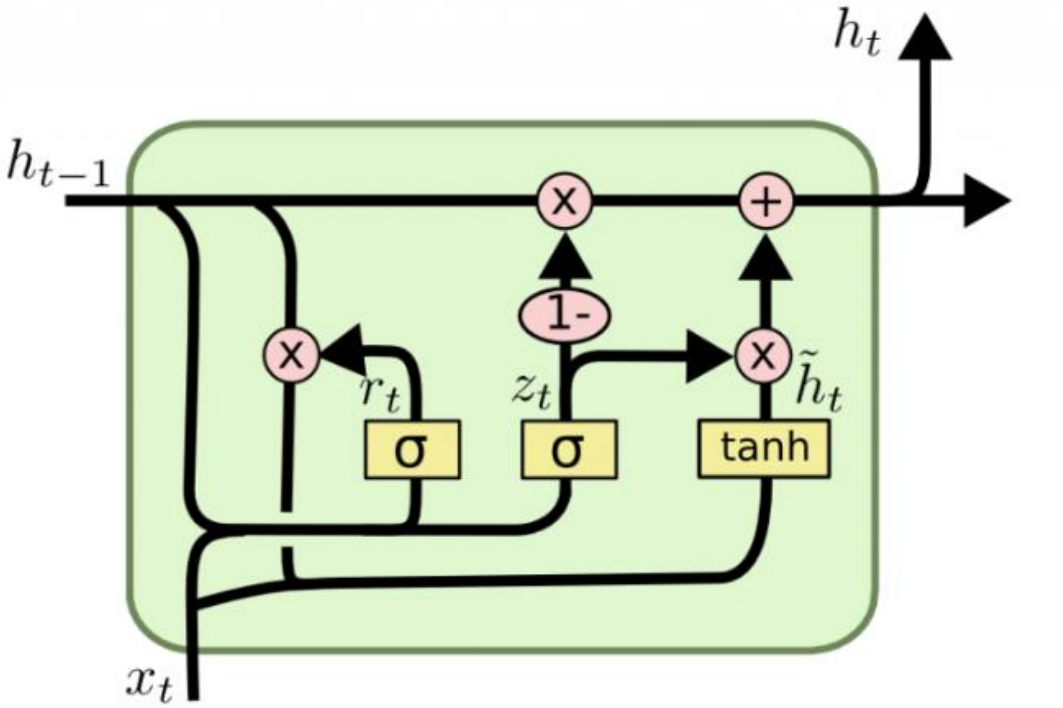


Рисунок 2.5 — Схема для типу GRU

Для порівняння вказаних вище інструментів для токенізації та векторизації тексту, були створені таблиці 2.2-2.3, в яких вказані переваги та недоліки, що допоможуть визначитися з вибором інструменту для реалізації токенізації та векторизації.

Таблиця 2.2 — Порівняння інструментів для токенізації тексту

| Інструмент | Переваги | Недоліки |
|---|---|---|
| NLTK | Гнучка система токенізації та підтримка різних мов. | Вимагає завантаження ресурсів для мов. |
| TextBlob | Зручний API для токенізації та аналізу тональності. | Обмежена функціональність порівняно з іншими інструментами. |
| spaCy | Висока швидкість та продуктивність. | Великий обсяг пам'яті, особливо для моделей. |
| Gensim | Використовується для тематичного моделювання та векторної репрезентації тексту. | Обмежена функціональність для токенізації. |
| Keras | Можливість використовувати різні шари для токенізації та обробки тексту в нейронних мережах. | Спрямована на нейронні мережі, а не на обробку тексту. |
| LLaMA | Має потужні мовні можливості та може використовуватися для різних завдань NLP. | Вимагає великих обчислювальних ресурсів. |

Таблиця 2.3 — Порівняння інструментів для векторизації тексту

| Інструмент | Переваги | Недоліки |
|---|---|---|
| TF-IDF | Враховує важливість термінів у тексті на основі частоти та оберненої частоти документів. | Не враховує семантичний зміст слів. |
| | Застосовується для виділення ключових слів та важливих термінів. | Може вимагати обширного набору даних для ефективної роботи. |

Продовження таблиці 2.3 — Порівняння інструментів для векторизації тексту

| | | |
|---|---|---|
| Word Embeddings | Захоплює семантичні зв'язки між словами. | Може вимагати значного обсягу даних для навчання. |
| GloVe | Здатний враховувати синоніми та відношення між словами. | Витікає з контексту навчальних даних. |
| | Використовується для векторизації слова в просторі низькорозмірних векторів. | Передбачає сталі вбудовані вектори для слів, що може бути обмежено. |
| | Ефективний для врахування семантичного змісту тексту. | Вимагає часу та обчислювальних ресурсів для навчання. |
| BoW | Застосовується для простих моделей та швидких завдань. | Не враховує семантичні зв'язки між словами. |
| | Працює добре для текстових класифікаційних завдань. | Великий розмір векторів у високорозмірних просторах. |
| | Добре підходить для розрізнення унікальних слів у тексті. | |

Зважаючи на проведені аналізи інструментів, найбільш підходящими в розробці будуть для токенізації — NLTK, для векторизації — TF-IDF для регресійної моделі прийнято було рішення спробувати зробити ансамбль із декількох регресійних моделей.

## 2.3 Структурно-функціональне моделювання

IDEF0 (Integration Definition for Function Modeling) та IDEF1 (Integration Definition for Information Modeling) — це стандартизовані методи моделювання для аналізу та проектування функціональних та інформаційних систем. Для даної задачі ми можемо створити діаграми IDEF0 та IDEF1 для технології прогнозування вартості товарів [17].

Діаграма IDEF0 дає узагальнений огляд функціональної структури

системи та показує взаємозв'язки між її основними функціями та підсистемами.

Діаграми IDEF0 та IDEF1 можуть бути досить складними та обширними, тому я створю загальний опис для обох. На рисунку 2.6 наведена спрощена функціональна модель для технології прогнозування вартості товарів.

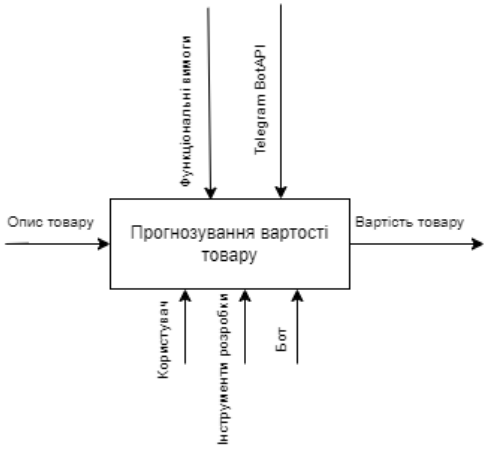


Рисунок 2.6 — IDEF0 діаграма

Діаграма IDEF1 моделює потік інформації через систему та показує, як дані обробляються та передаються між різними компонентами системи.

На діаграмі IDEF1 показано, як інформація пересилається від користувача до чат-бота та моделі для прогнозування, а також як результати прогнозу повертаються назад до користувача.

Ці діаграми IDEF0 та IDEF1 надають візуальне уявлення про функціональну та інформаційну структуру технології для прогнозування вартості товарів та взаємодію з користувачем (див. рис. 2.7).


Рисунок 2.7 — IDEF1 діаграма

# 3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

## 3.1 Реалізація та навчання моделі прогнозування

Прогнозування вартості товару за його текстовим описом — це завдання, яке вимагає використання технік обробки природної мови (NLP) і машинного навчання (ML). Ось загальний план для вирішення цієї задачі:

1.Збір та підготовка даних:

- Зібрати дані про товари, включаючи текстовий опис та вартість.

В якості датасету для навчання моделі використано датасет Mercari Price Suggestion [18]. Завантаження всіх даних та екстракція даних з архіву виконується таким кодом:

```
!curl -o sample_submission.csv.7z https://storage.googleapis.com/kagglesdsdata/co
mpetitions/7559/44327/sample_submission.csv.7z
!curl -o sample_submission_stg2.csv.zip https://storage.googleapis.com/kagglesdsd
ata/competitions/7559/44327/sample_submission_stg2.csv.zip
!curl -o test_stg2.tsv.zip https://storage.googleapis.com/kagglesdsdata/competiti
ons/7559/44327/ test_stg2.tsv.zip

!apt-get install p7zip
!p7zip -d -f -k /train.tsv.7z
!unzip -o /sample_submission_stg2.csv.zip
!unzip -o /test_stg2.tsv.zip
```

- Перевірити та очистити дані від викидів, дублікатів та відсутніх значень.

Далі зчитуємо дані та виконуємо первинну очистку даних, заповнення пропусків:

```
import pandas as pd
train = pd.read_csv('/train.tsv', sep='\t')
test = pd.read_csv('/test.tsv', sep='\t')
train.head()
```

На рисунку 3.1 наведено приклад даних з датасету.

| train_id | name | item_condition_id | category_name | brand_name | price | shipping | item_description |
|---|---|---|---|---|---|---|---|
| 0 | MLB Cincinnati Reds T Shirt Size XL | 3 | Men/Tops/T-shirts | NaN | 10.0 | 1 | No description yet |
| 1 | Razer BlackWidow Chroma Keyboard | 3 | Electronics/Computers & Tablets/Components & P... | Razer | 52.0 | 0 | This keyboard is in great condition and works ... |
| 2 | AVA-VIV Blouse | 1 | Women/Tops & Blouses/Blouse | Target | 10.0 | 1 | Adorable top with a hint of lace and a key hol... |
| 3 | Leather Horse Statues | 1 | Home/Home Décor/Home Décor Accents | NaN | 35.0 | 1 | New with tags. Leather horses. Retail for [rm]... |
| 4 | 24K GOLD plated rose | 1 | Women/Jewelry/Necklaces | NaN | 44.0 | 0 | Complete with certificate of authenticity |

Рисунок 3.1 — Приклад даних з датасету

Дані мають такі колонки:

– name — назва оголошення;

– item_condition_id — стан товару, який надає продавець що відповідають таким значенням : New, Like New, Good, Fair, Poor;

– category_name — категорія оголошення, що має основну категорію та можливі підкатегорії. Значення основних категорій «Men», «Electronics», «Women», «Home», «Sports & Outdoors», «Vintage & Collectibles», «Beauty», «Other», «Kids», «Handmade»;

– brand_name — назва бренду;

– price — ціна, за яку товар був проданий. Це цільова змінна, яку будемо прогнозувати. Ціна вказана в доларах США;

– shipping — 1, якщо вартість доставки оплачує продавець, і 0 - якщо покупець;

– item_description — повний опис товару. Дані вже очищені від можливого знаходженя ціни в описі.

Далі розглянемо безпосередньо самі дані, пропуски, значення і так далі (див. рис. 3.2). Подивимось кількість пропусків по кожній ознаці:

```
plt.figure(figsize=(14, 6))
```

```
missing = train.isna().sum() / len(train) * 100
plots = missing.plot(kind='bar', title='Missing Ratio (%)')
plt.ylim([0, 100])

for bar in plots.patches:
    plots.annotate(format(bar.get_height(), '.2f'),
        (bar.get_x() + bar.get_width() / 2,
        bar.get_height()), ha='center', va='center',
        size=10, xytext=(0, 8),
        textcoords='offset points')
```

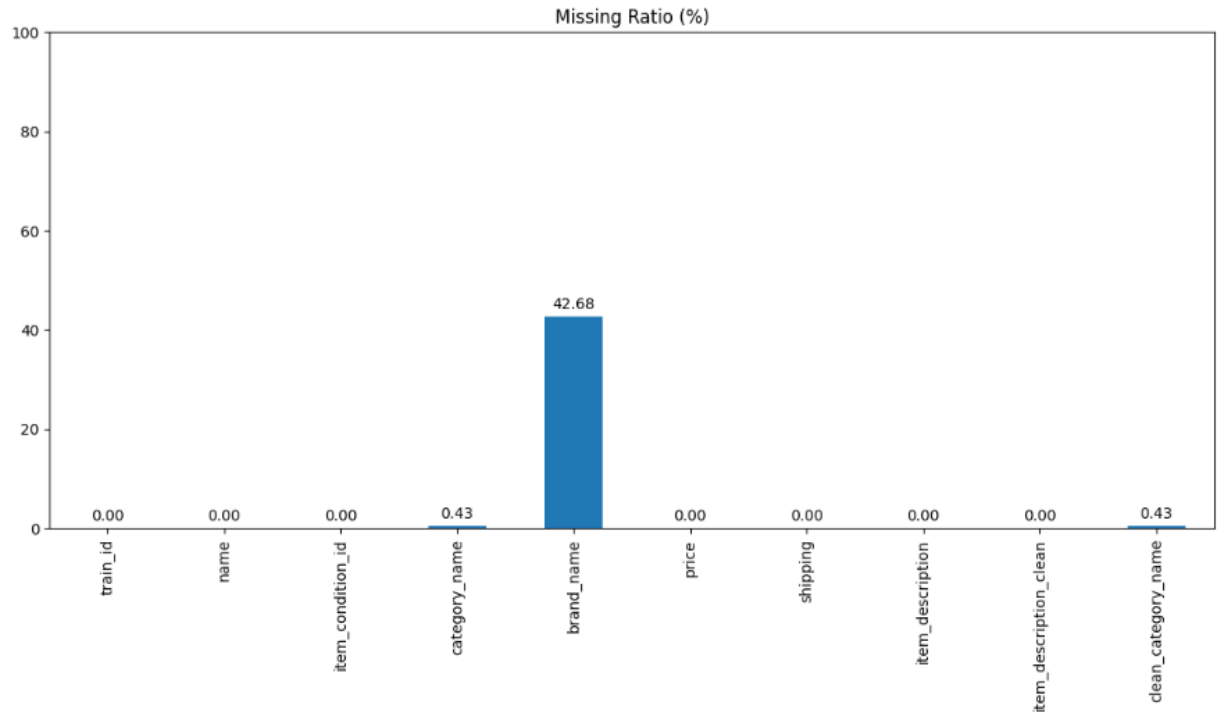


Рисунок 3.2 — Кількість пропусків у відсотках

Як бачимо найбільша кількість пропусків у колонці з назвою бренду. Це досить значна частина тому просто відкинути ці дані — погана ідея, тому необхідно створити метод для заповнення пропусків.

```
def fill_missing_data(data):
    data.category_name.fillna(value = "Others", inplace = True)
    data.brand_name.fillna(value = "Not known", inplace = True)
    data.item_description.fillna(value = "No description given", inplace =
True)
    return data

train = fill_missing_data(train)
print(np.shape(train))
train.head()
train.isnull().any()
```

```
st_words = stopwords.words('english')
def cleanText(x: str):
    if (pd.isna(x)):
        return 'missing'
    x = x.lower()
    x = x.strip()
    return x
def cleanCat(x: str):
    if (pd.isna(x)):
        return x
    x = x.replace("s ", " ")
    x = x.replace("s/", "/")
    x = x.replace('/'," ")
    return x.lower()

train['name'] = train.name.apply(cleanText)
train['item_description_clean'] = train.item_description.apply(cleanText)
train['clean_category_name'] = train.category_name.apply(cleanCat)
train['brand_name'] = train.brand_name.apply(lambda x: x if pd.isna(x)
else x.lower())
```

```
train_id             False
name                 False
item_condition_id    False
category_name        False
brand_name           False
price                False
shipping             False
item_description     False
dtype: bool
```

Побудуємо дисперсію цільової змінної, як бачимо вона не має нормального розподілу тому застосуємо логарифмічне перетворення (рис. 3.3).

```
plt.subplot(2,1,1)
train.price.plot.hist(bins=50, edgecolor='white', range=[0,250])
plt.subplot(2,1,2)
np.log(train.price+1).plot.hist(bins=50, edgecolor='white')
train['target_log'] = np.log(train.price+1)
```

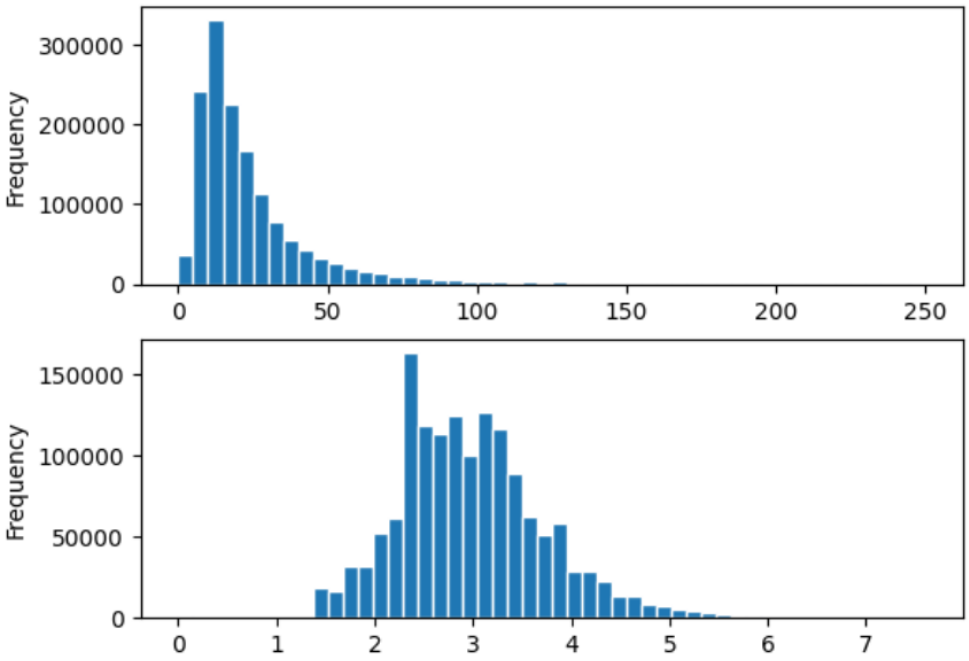


Рисунок 3.3 — Розподіл цільової змінної з і без логарифмічного перетворення

Розглянемо можливі інші ознаки товару (наприклад, категорію, бренд, відгуки тощо) та створемо на їх основі нові ознаки. Подивимося на розподіл товарів за станом (рис. 3.4).

```
train.item_condition_id.value_counts().plot(kind='bar')
```

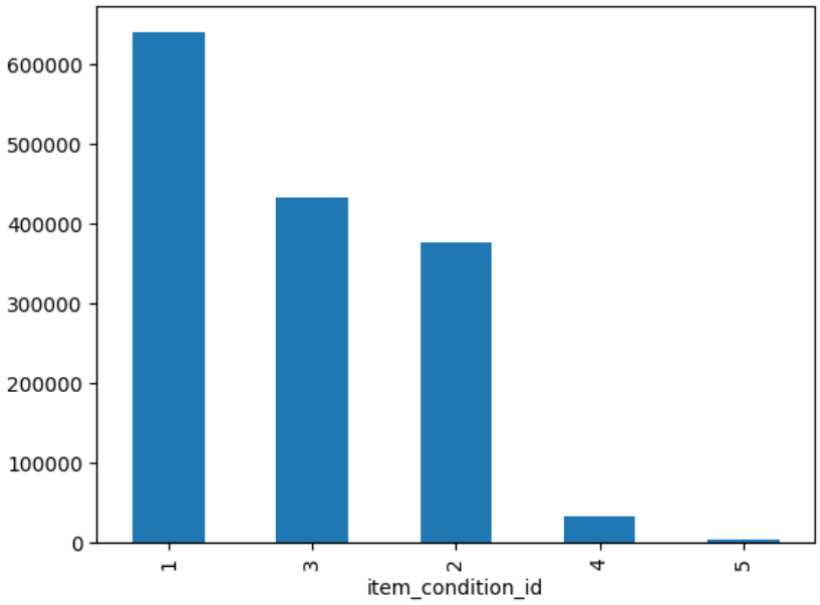


Рисунок 3.4 — Розподіл даних за станом товару

Як бачимо, ознака стану товару досить незбалансована, тому варто це враховувати при подальшому перетворені ознак та навчанні.

Далі розглянемо категорії товарів. Всього 1288 унікальних назв категорій. Розділимо їх на підкатегорії, та подивимось на їх розподіл. Бачимо, що в середньому кожен товар має більше ніж одну підкатегорію, а саме 1.47.

```
len(train.category_name.unique())
train['cat_split_count'] = train.category_name.apply(lambda x: x if
pd.isna(x) else len(x.split('/')))
train['cat_split_count'].describe()
```

```
count    1.476208e+06
mean     3.005045e+00
std      9.573042e-02
min      3.000000e+00
25%      3.000000e+00
50%      3.000000e+00
75%      3.000000e+00
max      5.000000e+00
Name: cat_split_count, dtype: float64
```

Як бачимо, в даних дуже багато унікальних назв категорій category_name, це означає, що ми не можемо обробляти category_name як клас, це потрібно обробляти як природну мову. Обробка як класу призведе до того, що модель не зможе захопити більш узагальнену інформацію. Розділимо фічу category_name на декілька підкатегорій

```
train['general_cat'] = train.category_name.apply(lambda x: x if pd.isna(x)
else (x.split('/')[0]))
train['sub_cat1'] = train.category_name.apply(lambda x: x if pd.isna(x)
else (x.split('/')[1]))
train['sub_cat2'] = train.category_name.apply(lambda x: x if pd.isna(x)
else (x.split('/')[2]))
```

Після розділення на підкатегорії порахуємо кількість унікальних значень в головній категорії (рис. 3.5).

```
len(train.general_cat.unique())
11
train['general_cat'].value_counts().plot(kind='bar')
```

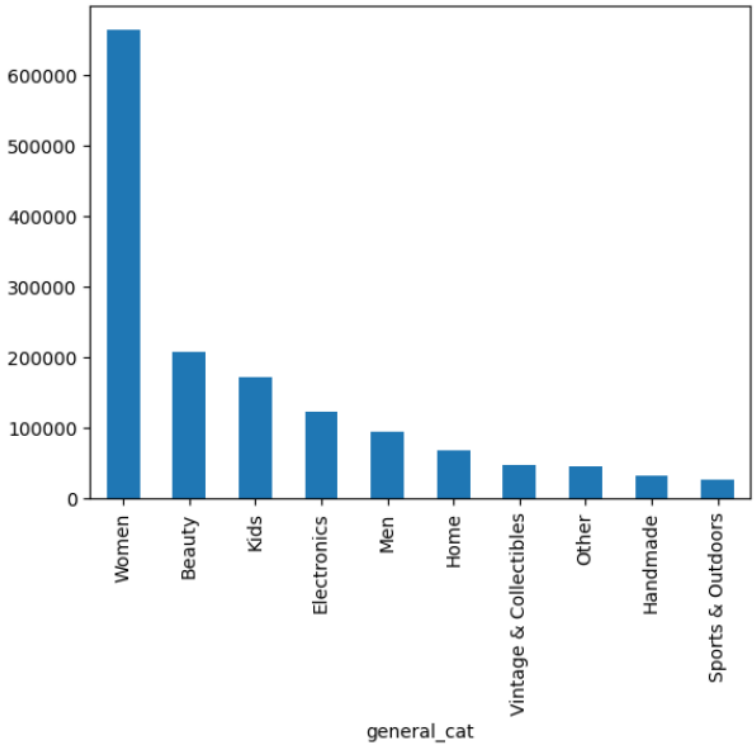


Рисунок 3.5 — Розподіл даних по основній категорії

```
print(f"Len of sub category 1 {len(train.sub_cat1.unique())}")
print(f"Len of sub category 2 {len(train.sub_cat2.unique())}")
Len of sub category 1 114
Len of sub category 2 871
```

Через величезну кількість підкатегорій подивимося лише найбільші та найменші за кількістю (рис. 3.6).

```
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
train.sub_cat1.value_counts().sort_values(ascending=False)[:10].plot(kind=
'bar')
plt.subplot(1,2,2)
train.sub_cat1.value_counts().sort_values(ascending=False)[-
10:].plot(kind='bar')
plt.show()
```

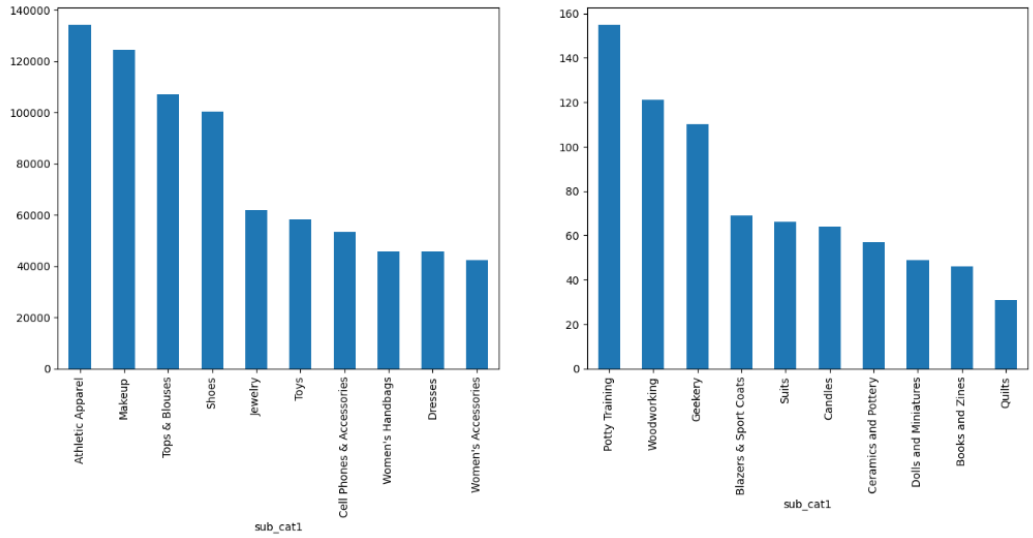


Рисунок 3.6 — Розподіл даних по кількості за субкатегорією

Також розглянемо кількість унікальних брендів та подивимось розподіл за брендами (рис. 3.7).

```
len(train.brand_name.unique())
4810
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
train.brand_name.value_counts().sort_values(ascending=False)[:10].plot(kin
d='bar')
plt.subplot(1,2,2)
train.brand_name.value_counts().sort_values(ascending=False)[-
10:].plot(kind='bar')
plt.show()
```

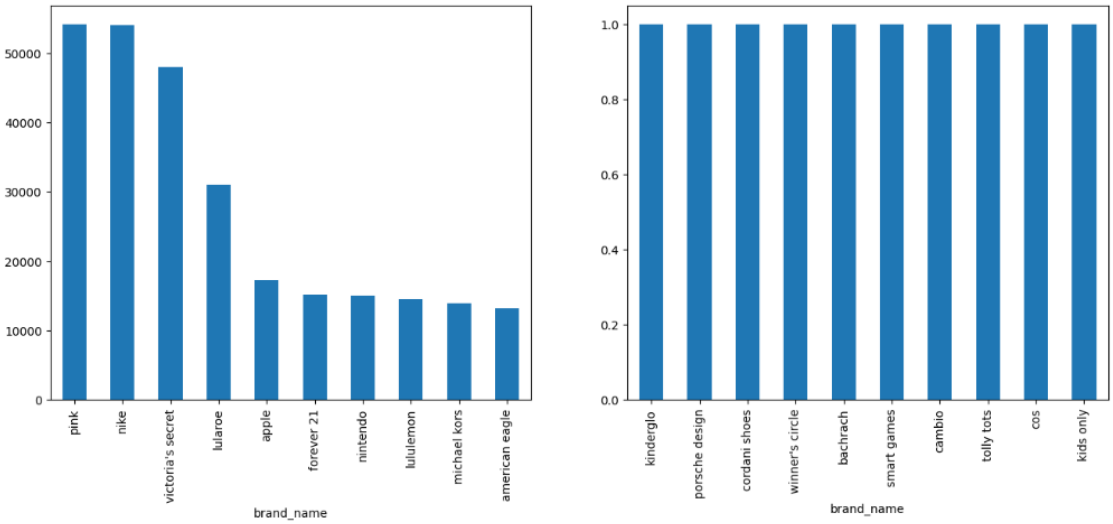


Рисунок 3.7 — Розподіл даних по кількості за брендом

З наведеного вище графіку ми можемо зробити висновок, що найчастіше бренд, який наявний в датасеті це PINK більше ніж 50 000 зразків, і багато брендів, які мають лише по 1 зразку.

Також можна розглянути кількість слів і довжину назви бренду (рис. 3.8).

```
train['name_len'] = train.brand_name.apply(lambda x: x if pd.isna(x) else
len(x))
train['name_split'] = train.brand_name.apply(lambda x: x if pd.isna(x)
else x.split(' '))
train['name_word_count'] = train.brand_name.apply(lambda x: x if
pd.isna(x) else len(x.split(' ')))
train.head()

plt.subplot(2,1,1)
train.name_len.plot.hist(edgecolor='white')
plt.subplot(2,1,2)
(train.name_word_count).plot.hist(edgecolor='white')
```

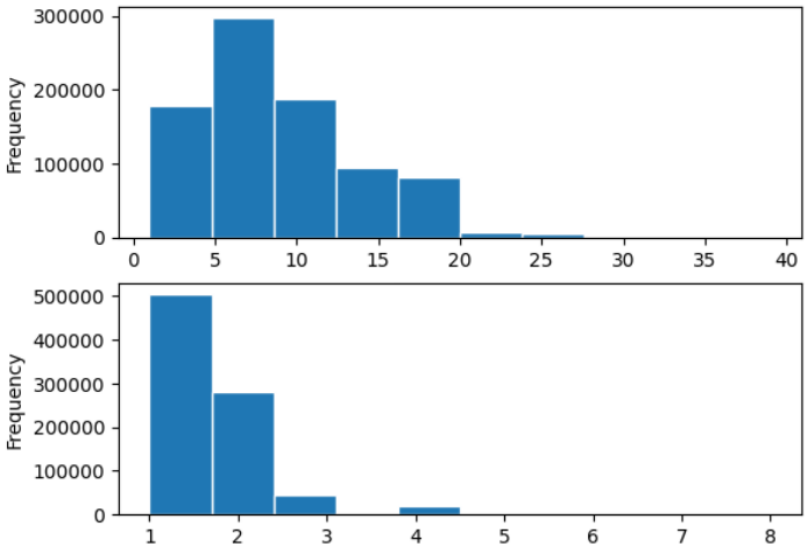


Рисунок 3.8 — Гісторгами розподілу довжини назви бренду та кількості слів у назві бренду

Побудуємо співвідношення довжини та кількості слів до ціни (рис. 3.9-3.10).

```
ax = train.groupby('name_len')['target_log'].median().plot()
ax = train.groupby('name_len')['target_log'].mean().plot()
ax.set_ylabel('log price')
```

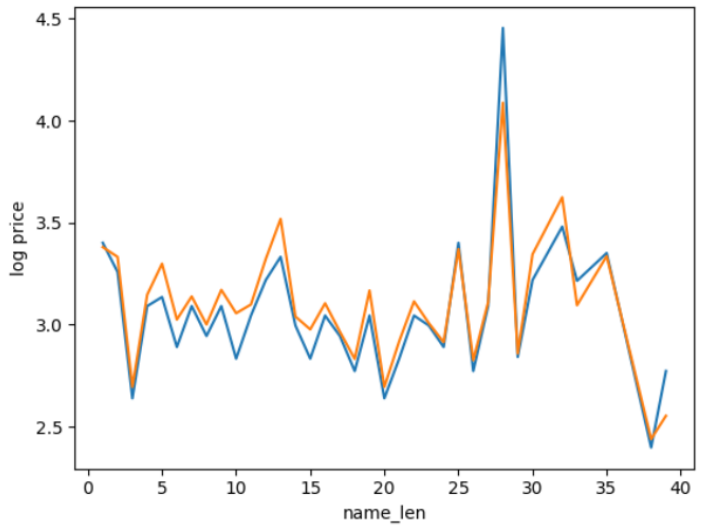


Рисунок 3.9 — Графік залежності ціни товару від довжини назви бренду

```
ax = train.groupby('name_word_count')['target_log'].median().plot()
ax = train.groupby('name_word_count')['target_log'].mean().plot()
ax.set_ylabel('log price')
```

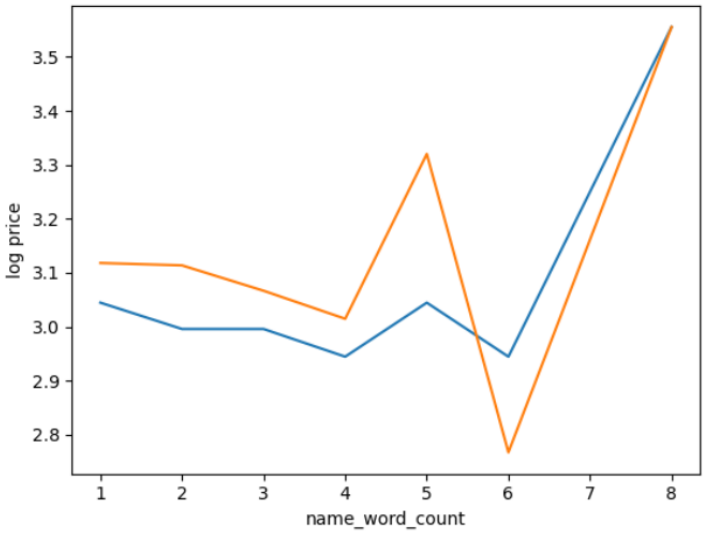


Рисунок 3.10 — Графік залежності ціни товару від кількості слів в назві бренду

Подивимось як вартість доставки впливає на ціну. Побудуємо гістограми залежності ціни від того хто оплачує доставку, покупець чи продавець (рис. 3.11). На графіку показано, що товар, вартість доставки якого сплачував покупець, найімовірніше, буде дорожчим за продукт, вартість доставки якого оплачено продавцем.

```
seller = train[train.shipping==1]
buyer = train[train.shipping==0]
plt.subplot(2,1,1)
seller.price.plot.hist(bins=50, range=[0,200])
buyer.price.plot.hist(alpha=0.7, bins=50,range=[0,200])
plt.subplot(2,1,2)
np.log(seller.price+1).plot.hist(bins=50, )
np.log(buyer.price+1).plot.hist(alpha=0.7, bins=50)
plt.legend(['shipping paid by seller', 'shipping paid by buyer'])
```

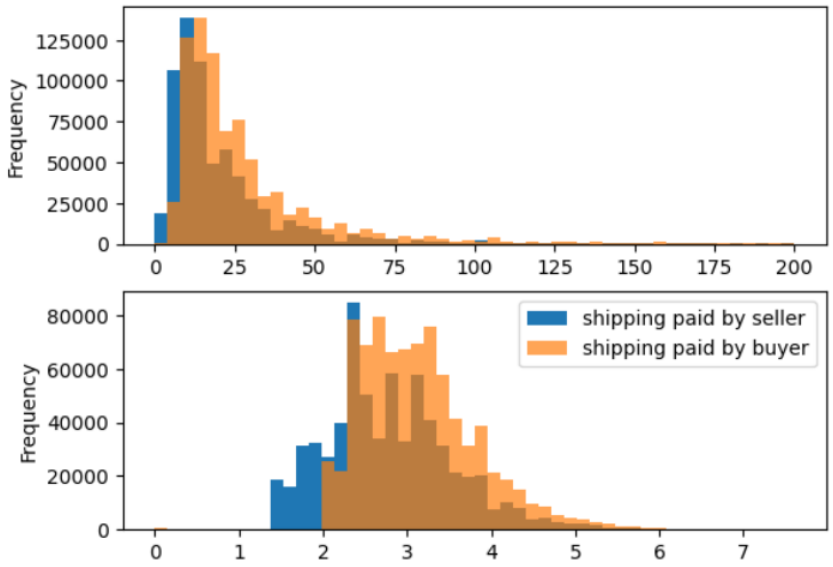


Рисунок 3.11 — Гісторгами розподілу ціни від того хто оплачує доставку

Побудуємо фічі які показують як довжина опису впливає на ціну.

```
train['desc_char_len'] = train.item_description.apply(lambda x: x if
pd.isna(x) else len(x))
train['desc_arr'] = train.item_description.apply(lambda x: x if pd.isna(x)
else x.split(' '))
train['count_desc_word'] = train.item_description.apply(lambda x: x if
pd.isna(x) else len(x.split(' ')))
train
ax = train.groupby('count_desc_word')['target_log'].median().plot()
ax = train.groupby('count_desc_word')['target_log'].mean().plot()
ax.set_ylabel('log price')
```

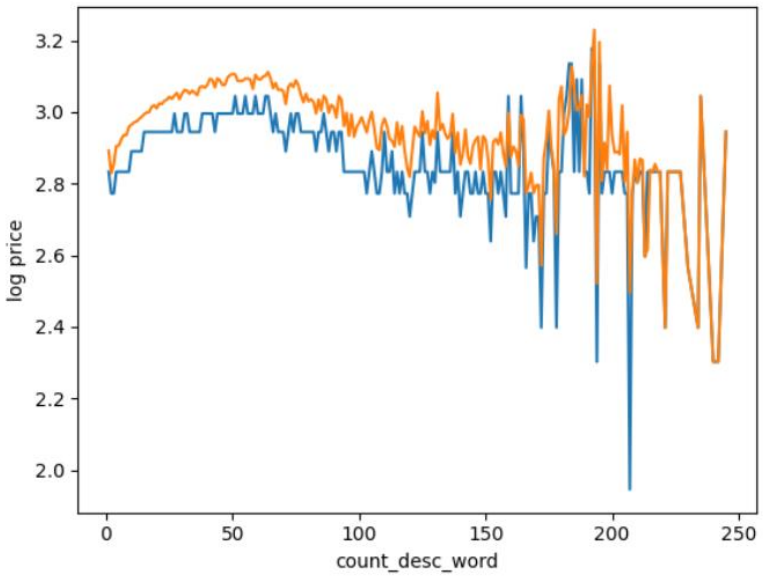


Рисунок 3.12 — Графік залежності вартості товару від кількості слів у його описі

Ціна зросте, якщо продавець надасть більше опису, але є поворотний момент, коли ціна знижується, якщо опис буде занадто довгим. Також можемо зробити висновок, що існує розрив між середнім і медіаною ціни для кожного слова.

За додатковим аналізом ще дізнаємось, що деякі продавці включають назву бренду в назву продукту, тому будемо намагатися приписати назву бренду з назви продукту, якщо ми знайшли існуючу назву бренду в назві продукту.

```
brand_score =
dict(train[train.brand_name.notnull()]["brand_name"].value_counts())
processed_brand_name = []
for index,i in tqdm(train.iterrows()) :

    if  pd.isnull(i.brand_name):
        words = i['name'].split()
        score  = []
        for j in words:
            if j in brand_score.keys():
                score.append(brand_score[j])
            else: #if the word is not a brand name append -1
                score.append(-1)
```

```
        if words == []:
            processed_brand_name.append("missing")
        elif max(score) > 0:
            processed_brand_name.append(words[score.index(max(score))])
        else:
            processed_brand_name.append("missing")
    else:
        processed_brand_name.append((i.brand_name))
train["brand_name_fixed"] = processed_brand_name
```

Додамо до назви товару бренд з опису, якщо він є. До наявного опису додамо

```
train["name"] = train["name"] + train["brand_name_fixed"]
train["text"] = train["name"] + train["brand_name_fixed"] +
train["processed_item_description"] + train["clean_category_name"]
train.loc[train.encoded_cat.isnull(), "encoded_cat"] = -1
```

Також спробуємо додати ці ознаки які відповідають за емоційне забарвлення текстового опису, адже це скоріше за всього також буде впливати на ціну. Якщо продавець дуже емоційно і гпрно описує товар то навряд чи він буде продавати гарну річ яку він описує дуже дешево.

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

nltk.downloader.download('vader_lexicon')
sid = SentimentIntensityAnalyzer()

neg = []
neu = []
pos = []
compound = []
for a in tqdm(train["text"].values) :
    neg_value = sid.polarity_scores(a)['neg']
    neu_value = sid.polarity_scores(a)['neu']
    pos_value = sid.polarity_scores(a)['pos']
    comp_value = sid.polarity_scores(a)['compound']
    neg.append(neg_value)
    neu.append(neu_value)
    pos.append(pos_value)
    compound.append(comp_value)

train['neg'] = neg
train['neu'] = neu
```

```
train['pos'] = pos
train['compound'] = compound
```

2. Токенізація і векторизація тексту:

- Використати токенізацію для розбиття текстового опису на окремі токени. Векторизувати токени в числовий формат, який можна використовувати для навчання моделі. Це може бути TF-IDF векторизація, Word2Vec, GloVe, або вектори, навчені на великих корпусах тексту, але, враховуючи попередній аналіз, було обрано TF-IDF.

Для початку дослідження якості отриманих фіч з текстового опису було вирішено використати більш доступний за ресурсами алгоритм TfidfVectorizer.

```
import torch
import tensorflow as tf
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import hstack
# Tokenisasi dan embedding untuk setiap teks dalam dataset
tfidf_text = TfidfVectorizer(max_features=1000)
tfidf_name = TfidfVectorizer(max_features=100)
tfidf_text = tfidf_text.fit_transform(train['text'])
tfidf_name = tfidf_name.fit_transform(train['name'])
train_numeric = train[['shipping', 'encoded_cat']].values
X_train_gabungan = np.hstack([tfidf_text, tfidf_name, train_numeric])
```

3. Навчання та оцінка моделі:

- Дані необхідно розділити на навчальний і тестовий набори.

```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(
    X_train_gabungan, train["target_log"], test_size=0.2, random_state=42
)
```

- Натренувати модель на навчальних даних і оцінити результати прогнозування на тестових даних. Оцінка буде відбуватися за метриками, це може бути такі як середньоквадратична похибка (RMSE), середня абсолютна похибка (MAE) та інші для оцінки якості прогнозу.

4. Розробка моделі:

– Необхідно створити модель машинного навчання або глибокого навчання для завдання регресії (наприклад, лінійна регресія, Random Forest, глибокі нейронні мережі тощо). Визначити архітектуру моделі і її параметри. Використати векторизовані дані для навчання моделі, використовуючи історичні дані про вартість товарів.

Розглянувши всі переваги та недоліки різних регресійних моделей було вирішено застосувати технологію ансамблювання моделей з використанням класу VotingRegressor з бібліотеки sklearn, щоб враховувати результати з кожної моделі [19].

```
from sklearn.svm import SVR
from sklearn.linear_model import SGDRegressor
from sklearn.ensemble import VotingRegressor
from xgboost import XGBRegressor

model_1 = XGBRegressor(device="cuda",  tree_method= 'gpu_hist')
model_2 = RidgeCV(fit_intercept=True, alphas=[10.0],normalize=False, cv =
3, scoring='neg_mean_squared_error',)
model_3 = SGDRegressor(alpha = 0.00001, loss = 'squared_loss', penalty =
'l2', learning_rate = 'constant')

reg = VotingRegressor([("xgb", model_1), ("r", model_2), ("sgd",
model_3)], n_jobs = -1)
reg.fit(X_train, Y_train)
```

Також було виконано навчання окремо на кожній моделі, де кожна модель показувала гірший результат ніж при застосуванні ансамблювання (рис. 3.13).

| Model | Alpha | Rmsle Error |
|---|---|---|
| XGBRegressor | 0.1 | 0.4462 |
| RidgeReg Cv | 5 | 0.4592 |
| SGD Reg | 1e-05 | 0.4535 |
| Ensemble Voting Reg | xgb, r, sgd | 0.4391 |

Рисунок 3.13 — Результати

5. Оптимізація моделі:

– У випадку незадовільних результатів прогнозування моделі необхідне подальша оптимізація моделі. Це може бути більш широкий підбір гіперпараметрів моделі та налаштування моделі, збільшення обсягу навчальних даних та інші техніки [20].

У отриманих моделях оптимізація не виконувалась, так як отримані результати цілком задовільні і є на рівні, а в деяких випадках і краще ніж у інших дослідників хто розв’язував дану задачу.

6.Використання моделі:

– Після досягнення задовільних результатів модель може бути використана для прогнозування вартості товарів на нових даних, що надходять в реальному часі. Необхідно інтегрувати цю модель в систему прогнозування вартості товарів для користувачів або системи електронної комерції.

В даному випадку повноцінної платформи для використання системи ще не створено, проте був створений телеграм бот, який примає на вхід всі дані які необхідні для прогнозування вартості.

7.Моніторинг та підтримка:

– Необхідно запезпечити моніторинг і підтримку моделі в реальному часі, оскільки вартість товарів, назви та типи товарів можуть змінюватися з часом, і це скоріше за всього буде приводити до погіршення результатів прогнозування.

8.Постійне вдосконалення:

– Також потрібно продовжувати вдосконалювати модель або перейти до великих мовних моделей, при наявності великого обсягу даних і збирати нові дані для покращення точності передбачень поточної моделі.

## 3.2 Реалізація Telegram-бота

Для простоти інтегрування і взаємодії з користувачем — прогнозування вартості товару буде відбуватися шляхом взаємодії з телеграм-ботом. Реалізація неможлива без чітко встановлених вимог, тому прописуємо що саме є необхідним для коректної роботи.

Функціональні вимоги до боту:

– Взаємодія з користувачем: користувачі можуть відправити опис товару в текстовому форматі через команду бота, додавши також інформацію про бренд, стан товару.

– Попередня обробка тексту: функціонал повинен здійснювати попередню обробку тексту (так званий препроцесінг), включаючи токенізацію та векторизацію текстового опису.

– Прогнозування вартості товару: бот повинен використовувати навчені регресійні моделі, приводити вхідні дані до потрібного для моделі формату, прогнозувати вартість та виводити результат користувачу в зрозумілому для сприйняття вигляді.

Загальний план:

1. Створення Telegram-бота:

Перш за все необхідно створити Telegram-бота за допомогою Telegram @BotFather (рис. 3.14) та отримати токен для доступу до API, написавши команду «/newbot», що дозволить зареєструвати нового бота і вказати ім’я та username бота (рис. 3.15).

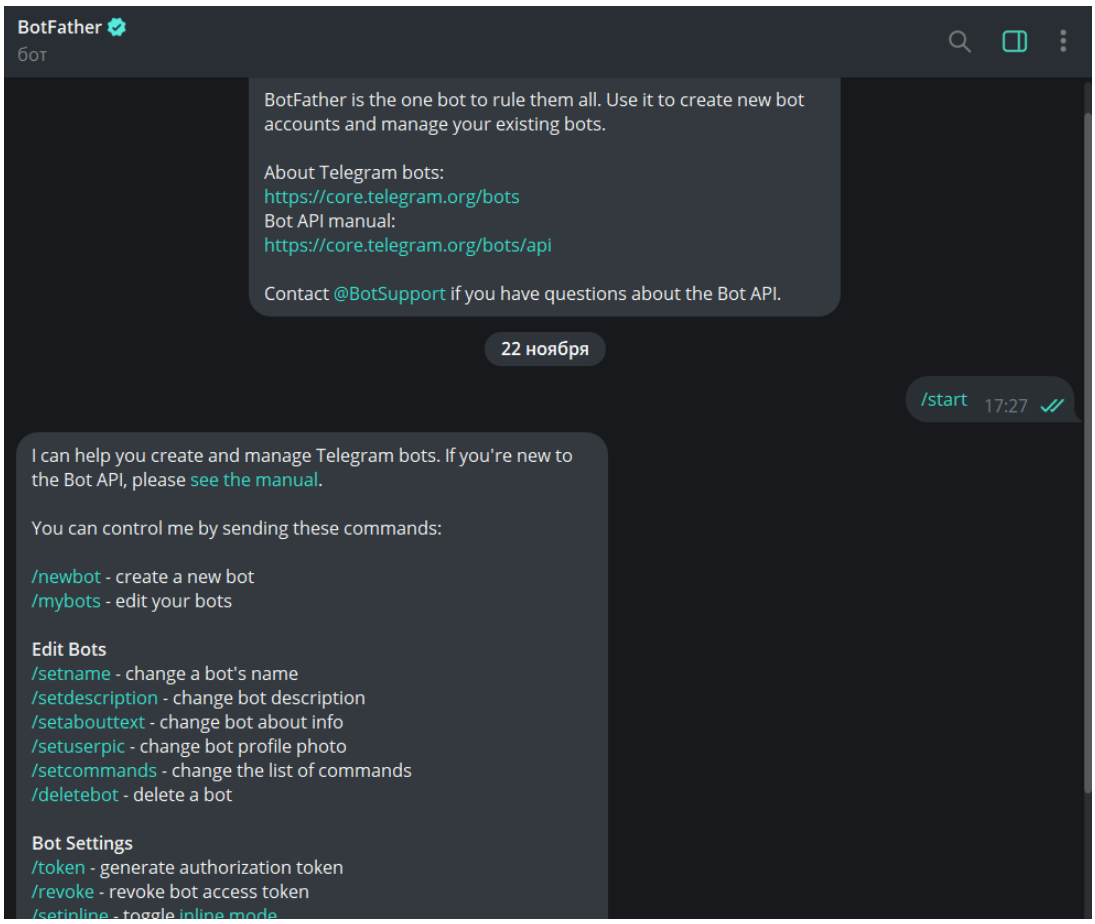


Рисунок 3.14 — Початок реєстрації бота

Після отримання токену Telegram можна розпочати написання коду. Нижче наведено опис деяких ключових елементів структури:

– bot/: Ця директорія містить всі файли та модулі, що пов'язані з ботом.

– main.py: Основний файл для запуску бота.

– utils.py: Допоміжні функції та утиліти, для обробки тексту.

– config.py: Файл конфігурації для зберігання токену та інших налаштувань.

– requirements.txt: Файл, в якому перераховані всі бібліотеки та їх версії, що використовуються в проєкті. Його можна створити за допомогою команди pip freeze > requirements.txt.


Рисунок 3.15 — Команди для створення бота

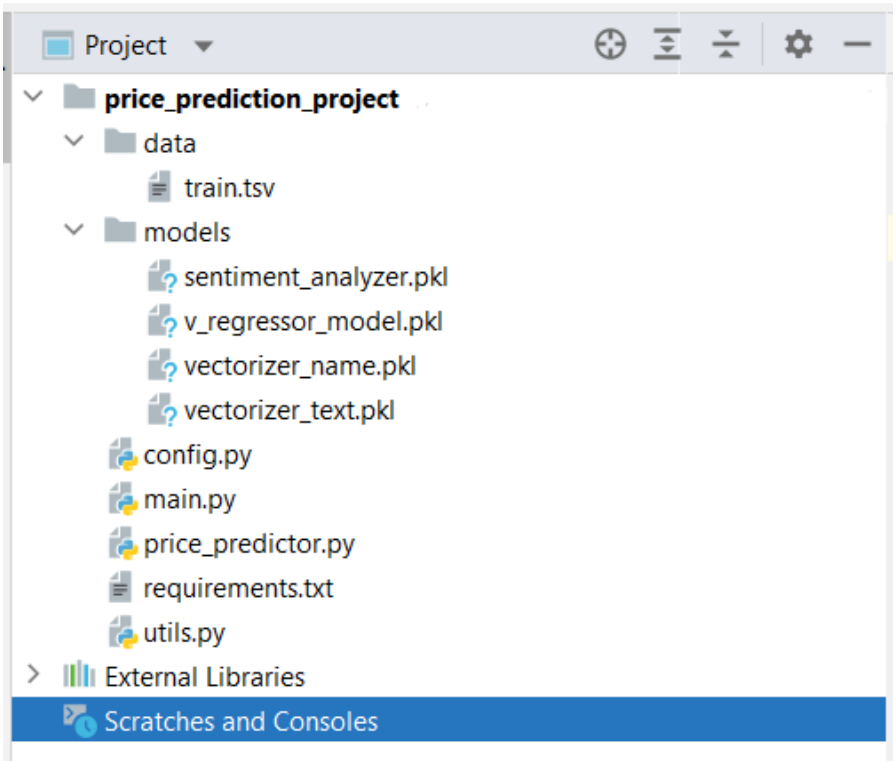


Рисунок 3.16 — Структура проєкту

2. Розробка бота:

– Використовуючи одну із бібліотек Telegram Bot API необхідно розробити основні команди, логіку бота та інтерфейс керування. Мова програмування для реалізації — Python. Основна функція запуску бота, та додавання хендлерів. Повний код боту представлений у додатку А.

```
from telegram import InlineKeyboardButton, InlineKeyboardMarkup
from telegram.ext import Updater, CommandHandler, CallbackQueryHandler,
ConversationHandler, MessageHandler, Filters

def main():
    updater = Updater(TOKEN, use_context=True)
    dp = updater.dispatcher

    conv_handler = ConversationHandler(
        entry_points=[CommandHandler('start', start)],
        states={
            ASK_QUESTION: [MessageHandler(Filters.text & ~Filters.command,
ask_question)],
            GET_RESPONSE: [MessageHandler(Filters.text & ~Filters.command,
get response),
                           CallbackQueryHandler(button)]
        },
        fallbacks=[CommandHandler('cancel', start)]
    )

    dp.add_handler(conv_handler)
    updater.start_polling()
    updater.idle()

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

3. Інтеграція моделі:

– Необхідно додати код для інтеграції розробленої моделі прогнозування вартості товару. Бот повинен приймати текстовий опис товару від користувача, бренд та інші дані які потрібні для моделі.

4. Попередній обробник тексту:

– Після отримання всіх даних від користувача бот повинен мати можливість попереднього оброблення тексту, як це було в описі попередньої задачі. Тобто, токенізацію та векторизацію тексту перед передачею його до моделі, щоб привести дані до потрібного формату, на яких модель навчалась. Повний код представлений у додатку Б

```
def __preprocessing(self, features_list):
    features_df = pd.DataFrame(np.hstack(features_list), columns=[])
    features_df = fill_missing_data(features_df)
    features_df['name'] = features_df.name.apply(cleanText)
    features_df['item_description_clean'] =
features_df.item_description.apply(cleanText)
    features_df['clean_category_name'] =
features_df.category_name.apply(cleanCat)
    features_df['brand_name'] = features_df.brand_name.apply(lambda x: x if
pd.isna(x) else x.lower())

    features_df = self.__add_sentiment(features_df)

    features_df["name"] = features_df["name"] +
features_df["brand_name_fixed"]
    features_df["text"] = features_df["name"] +
features_df["brand_name_fixed"] + features_df["processed_item_description"] +
features_df[
        "clean_category_name"]
    features_df.loc[features_df.encoded_cat.isnull(), "encoded_cat"] = -1

    tfidf_text = self.vectorizer_text.transform(features_df['text'])
    tfidf_name = self.vectorizer_name.fit_transform(features_df['name'])
    f_numeric = features_df[['shipping', 'encoded_cat']].values
    features_processed = np.hstack([tfidf_text, tfidf_name, f_numeric])
    return features_processed
```

5. Прогноз та відповідь користувачу:

- Зробити прогноз за домогою моделі прогнозування вартості товару на основі текстового опису, сформувати відповідь та відправити результат користувачу через Telegram бота.

# 4 ТЕСТУВАННЯ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

Тестування бота для прогнозування вартості товару за його описом важливий етап розробки, який допомагає переконатися, що бот працює коректно та надає точні прогнози. Ось кілька кроків для тестування бота:

1. Підготовка тестових даних:

– Підготуємо набір тестових даних, який включає в себе текстові описи товарів та відомі (правильні) вартості цих товарів. Дані мають бути представлені у форматі, який використовується для взаємодії з ботом.

2. Запуск бота:

Запускаємо чат-бота у тестовому середовищі, та відправляємо перші запити (див. рис. 4.1-4.3).

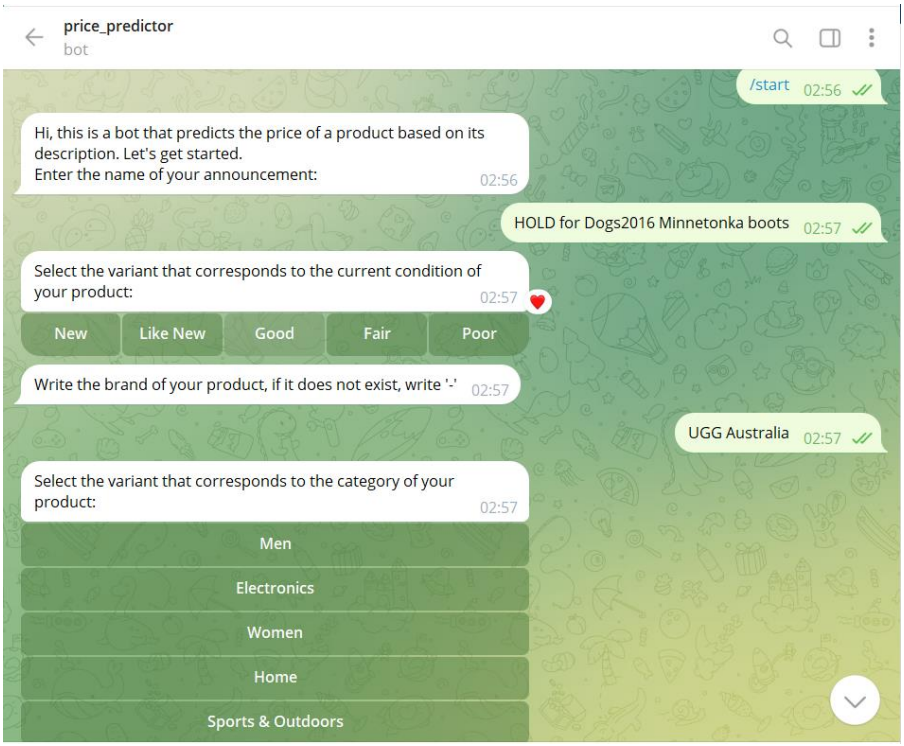


Рисунок 4.1 — Старт бота, та перші запитання повязані з назвою оголошення, станом товару та брендом

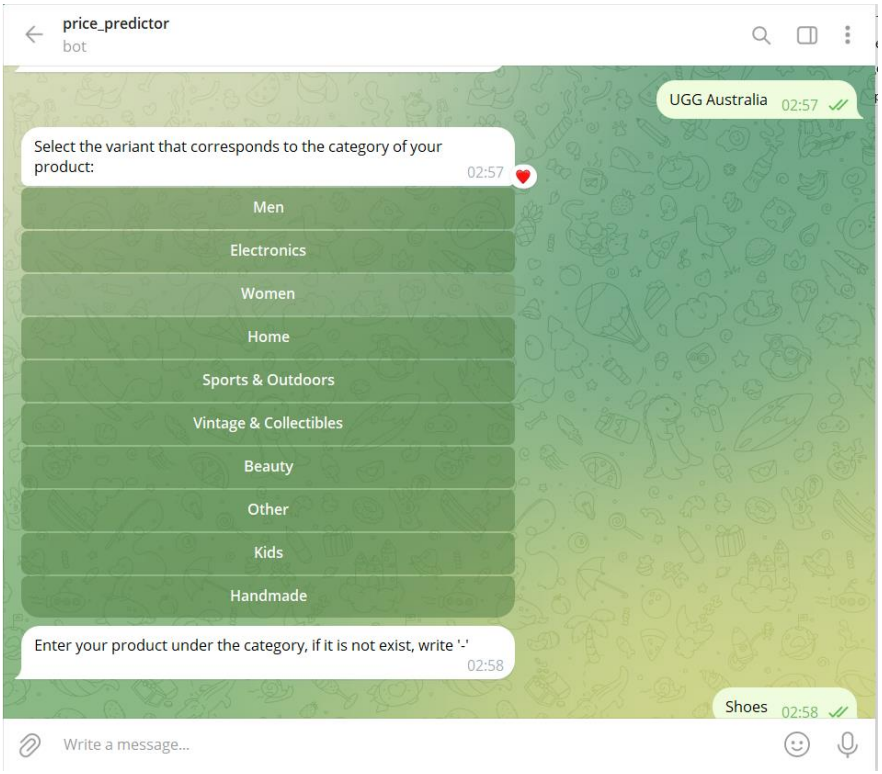


Рисунок 4.2 — Запитання з вибором категорій та підкатегорій

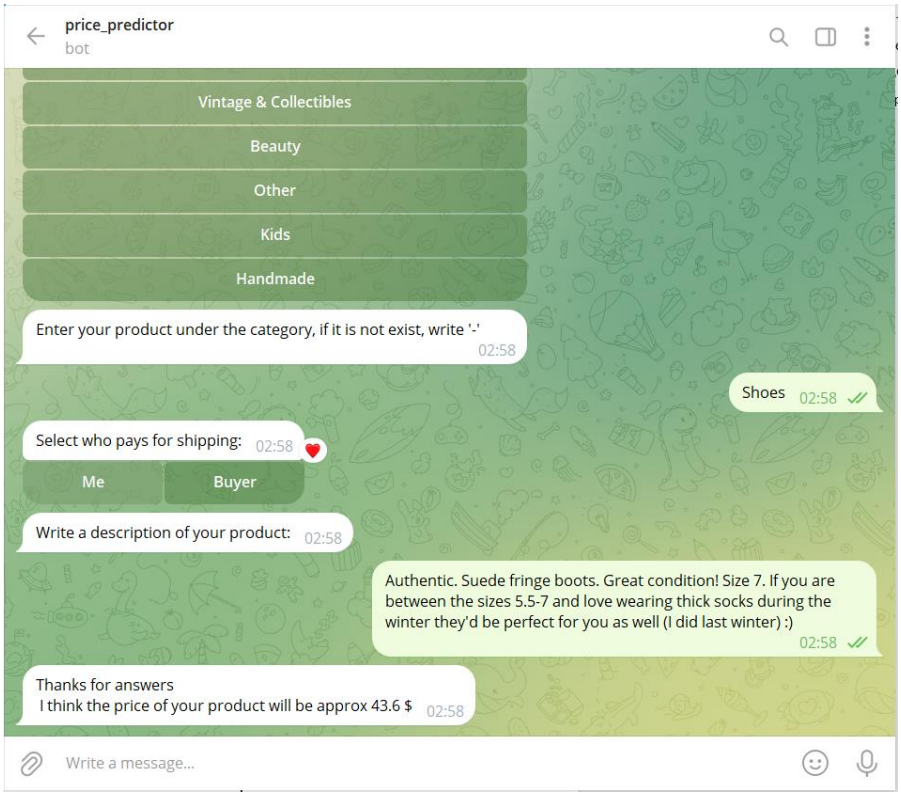


Рисунок 4.3 — Запитання щодо того, хто буде оплачувати доставку, та пропозицією ввести повний опис товару

Як бачимо бот видав ціну в 43.6 $, насправді ж ціна в тестовому датасеті була 43$. Тобто, спрогнозована ціна відрізняється в даному випадку на менше ніж на 1.5%, що є досить прийнятним результатом.

Тестування допомагає впевнитися в якості роботи чат-бота та визначити можливості для його вдосконалення перед впровадженням в реальному середовищі.

# ВИСНОВКИ

В магістерській кваліфікаційній роботі розроблено та впроваджено інформаційну технологію для прогнозування вартості товару на основі його опису. Створено програмний продукт з використанням мови програмування Python та реалізацією його в Telegram-боті. В основі самої технології використано адаптацію алгоритму NLP, що є часто вживаним в штучному інтелекті.

Представлений проєкт надасть додатковий інструментарій користувачам, яким необхідне попереднє передбачення ціни на товар для більш конкурентноспроможного оголошення продажу. Враховуючи динамічний характер електронної комерції, де ціни швидко змінюються під впливом різних факторів, технологія може успішно прогнозувати цінові тенденції.

Перший розділ роботи присвячений огляду таких питань: загальне дослідження предметної області, огляд існуючих рішень, переваги та недоліки, які дають змогу оцінити необхідність та актуальність створення власного продукту. Другий розділ висвітлює питання з вибору програмних засобів, алгоритмів розроблення, методів для оброблення NLP, характеристики процесів токенізації та векторизації, приділено увагу характеристикам моделей машинного та глибокого навчання. Наступна глава виокремлює саме реалізацію та навчання моделей прогнозування, і реалізацію боту. У четвертому розділі протестовано продукт на відповідність встановленим вимогам та, власне, правильність отриманих результатів.

Загалом, під час виконання завдання були розв'язані наступні питання:

- визначено об'єкт та предмет дослідження;
- виявлено проблему та актуальність розробки;
- проведено аналіз продуктів-аналогів, внаслідок чого виявлені недоліки та переваги;
- визначені функції та завдання, які повинна виконувати система;

- обрано програмні засоби для реалізації;
- обрано методи вирішення поставленої задачі.

Реалізація чат-боту була виконана за допомогою мови програмування Python з використанням TF-IDF для векторизації тексту, та для прийняття рішень — ансамбль із декількох регресійних моделей.

Однією з ключових переваг цієї розробки є можливість використання у реальному часі, що дозволяє користувачам отримувати актуальну та точну інформацію про ціни без зайвих затримок.

Результати дослідження свідчать про те, що впровадження даної технології може значно покращити взаємодію з користувачами, оптимізувати стратегії ціноутворення та підвищити ефективність платформ електронної комерції.

Розроблена технологія має перспективи для впровадження в реальні умови та внесення позитивного вкладу у розвиток сфери електронної комерції.

У процесі налагодження та тестування програмного продукту досліждено його працездатність для передбачення цін різних товарів, він працює стабільно та видає правильні результати.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Machine learning algorithms for teaching AI chat bots / Evgeny Tebenkov — Tokyo: Procedia Computer Science, 2021. — 191 с.

2. Mercari Price Suggestion Challenge [Електронний ресурс]. — Режим доступу: https://www.kaggle.com/c/mercari-price-suggestion-challenge/overview – (дата звернення 21.11.2023) — Назва з титулу екрана.

3. Mercari Price Suggestion Challenge using ML and DL models [Електронний ресурс]. — Режим доступу: https://medium.com/@sridatta0808/mercari-price-suggestion-challenge-using-ml-and-dl-models-505fe11d8fde – (дата звернення 21.11.2023) — Назва з титулу екрана.

4. TextCL: A Python package for NLP preprocessing tasks / Alina Petukhova — London: Richmond House, 2020. — 332 с.

5. The Python Book: The ultimate guide to coding with Python / Alex Hoskins — London: Richmond House, 2017. — 272 с.

6. Regression analysis/ Kanishka Tyagi — London: Academic Press, 2022. — 395 с.

7. Comparative study of regressor and classifier with decision tree using modern tools/ Jitendra Singh Kushwah — London: Proceedings, 2022. — 195 с.

8. Natural Language Processing With Python's NLTK Package [Електронний ресурс]. — Режим доступу: https://realpython.com/nltk-nlp-python/ – (дата звернення 21.11.2023) — Назва з титулу екрана.

9. Tokenization in NLP: Types, Challenges, Examples, Tools [Електронний ресурс]. — Режим доступу: https://neptune.ai/blog/tokenization-in-nlp – (дата звернення 21.11.2023) — Назва з титулу екрана.

10. Natural Language Processing With spaCy in Python [Електронний ресурс]. — Режим доступу: https://realpython.com/natural-language-processing-spacy-python/ – (дата звернення 21.11.2023) — Назва з титулу екрана.

11. A Simple Explanation of the Bag-of-Words Model [Електронний

ресурс]. — Режим доступу: https://victorzhou.com/blog/bag-of-words/ – (дата звернення 21.11.2023) — Назва з титулу екрана.

12. 4 методи векторизації текстів [Електронний ресурс]. — Режим доступу: https://medium.com/@bigdataschool/4f8ac90e4175a – (дата звернення 21.11.2023) — Назва з титулу екрана.

13. LLaMA (Large Language Model Meta AI) [Електронний ресурс]. — Режим доступу: https://lablab.ai/tech/meta/llama – (дата звернення 21.11.2023) — Назва з титулу екрана.

14. Understanding TF-IDF for Machine Learning [Електронний ресурс]. — Режим доступу: https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/ – (дата звернення 21.11.2023) — Назва з титулу екрана.

15. Семантика і технологія Word2Vec [Електронний ресурс]. — Режим доступу: https://habr.com/ru/articles/585838/ – (дата звернення 21.11.2023) — Назва з титулу екрана.

16. Top 5 Predictive Analytics Models and Algorithms [Електронний ресурс]. — Режим доступу: https://insightsoftware.com/blog/top-5-predictive-analytics-models-and-algorithms/ – (дата звернення 21.11.2023) — Назва з титулу екрана.

17. Компоненти синтаксису IDEF0 [Електронний ресурс]. — Режим доступу: https://elib.lntu.edu.ua/sites/default/files/elib_upload/page9.html – (дата звернення 21.11.2023) — Назва з титулу екрана.

18. Use of chat bots in Learning Management Systems / Eugeny Bezverhny — Tokyo: Procedia Computer Science, 2020. — 169 с.

19. Machine learning algorithms for teaching AI chat bots / Evgeny Tebenkov — Tokyo: Procedia Computer Science, 2021. — 191 с.

20. How to Create a Telegram Bot using Python[Електронний ресурс]. — Режим доступу: https://www.freecodecamp.org/news/how-to-create-a-telegram-bot-using-python/ – (дата звернення 21.11.2023) — Назва з титулу екрана.

# ДОДАТКИ

## Додаток А

## Код телеграм боту, та обробників подій

```
from telegram import InlineKeyboardButton, InlineKeyboardMarkup
from telegram.ext import Updater, CommandHandler, CallbackQueryHandler,
ConversationHandler, MessageHandler, Filters
import logging
from config import *
from price_predictor import PricePredictor

price_predictor = PricePredictor()


# Визначення станів бесіди
ASK_QUESTION, GET_RESPONSE = range(2)

# Логування для відстеження помилок
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s -
%(message)s', level=logging.INFO)
logger = logging.getLogger(__name__)

# Список питань
questions = ["""Hi, this is a bot that predicts the price of a product based
on its description. Let's get started.\nEnter the name of your
announcement:""",
             """Select the variant that corresponds to the current condition
of your product: """,
            "Write the brand of your product, if it does not exist, write '-
'",
             """Select the variant that corresponds to the category of your
product:""",
             "Enter your product under the category, if it is not exist,
write '-'",
             "Select who pays for shipping:",
             "Write a description of your product:"
             ]

answers = []

def start(update, context):
    context.user_data['index'] = 0
    return ask_question(update, context)

def ask_question(update, context):
    index = context.user_data['index']

    if index < len(questions):
        # Перевіряємо, чи поточне питання має відображатися з кнопками
        if index == 1:  # Додано умову для 'Питання 3'
            keyboard = [
                [InlineKeyboardButton("New", callback_data='1'),
                 InlineKeyboardButton("Like New", callback_data='2'),
                 InlineKeyboardButton("Good", callback_data='3'),
                 InlineKeyboardButton("Fair", callback_data='4'),
                 InlineKeyboardButton("Poor", callback_data='5')]
            ]
            reply_markup = InlineKeyboardMarkup(keyboard)
            update.message.reply_text(questions[index],
```

```
reply_markup=reply_markup)
            return GET_RESPONSE
        if index == 3:
            keyboard = [
                [InlineKeyboardButton("Men", callback_data='1')],
                 [InlineKeyboardButton("Electronics", callback_data='2')],
                 [InlineKeyboardButton("Women", callback_data='3')],
                 [InlineKeyboardButton("Home", callback_data='4')],
                 [InlineKeyboardButton("Sports & Outdoors",
callback_data='5')],
                 [InlineKeyboardButton("Vintage & Collectibles",
callback_data='6')],
                 [InlineKeyboardButton("Beauty", callback_data='7')],
                 [InlineKeyboardButton("Other", callback_data='8')],
                 [InlineKeyboardButton("Kids", callback_data='9')],
                 [InlineKeyboardButton("Handmade", callback_data='10')]
            ]
            reply_markup = InlineKeyboardMarkup(keyboard)
            update.message.reply_text(questions[index],
reply_markup=reply_markup)
            return GET_RESPONSE
        if index == 5:
            keyboard = [[InlineKeyboardButton("Me", callback_data='0'),
                 InlineKeyboardButton("Buyer", callback_data='1'),
               ]]

            reply_markup = InlineKeyboardMarkup(keyboard)
            update.message.reply_text(questions[index],
reply_markup=reply_markup)
            return GET_RESPONSE
        else:
            update.message.reply_text(questions[index])
            return GET_RESPONSE
    else:
        res = price_predictor.predict(answers)
        context.bot.send_message(chat_id=update.effective_chat.id,
                                 text=f"Thanks for answers\n I think the
price of your product will be approx {res} $")

        return ConversationHandler.END



def get_response(update, context):
    user_response = update.message.text
    answers.append(user_response)
    context.user_data['index'] += 1
    return ask_question(update, context)


def button(update, context):
    query = update.callback_query
    query.answer()

    # Записуємо відповідь у список відповідей
    answers.append(query.data)
    context.user_data['index'] += 1

    # Відправляємо наступне питання
    index = context.user_data['index']
    if index < len(questions):
        context.bot.send_message(chat_id=update.effective_chat.id,
```

```
text=questions[index])
        return GET_RESPONSE
    else:
        return ConversationHandler.END

def clear_answers():
    global answers
    answers = []

def main():
    updater = Updater(TOKEN, use_context=True)
    dp = updater.dispatcher

    conv_handler = ConversationHandler(
        entry_points=[CommandHandler('start', start)],
        states={
            ASK_QUESTION: [MessageHandler(Filters.text & ~Filters.command,
ask_question)],
            GET_RESPONSE: [MessageHandler(Filters.text & ~Filters.command,
get_response),
                           CallbackQueryHandler(button)]
        },
        fallbacks=[CommandHandler('cancel', start)]
    )

    dp.add_handler(conv_handler)
    updater.start_polling()
    updater.idle()

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

# Додаток Б

## Код класу для препроцесінгу текстового опису та прогнозування ціни

```
import pickle
import numpy as np
from tqdm import tqdm
from utils import *
class PricePredictor():
    def __init__(self):
        with open('./models/v_regressor_model.pkl', 'rb') as file:
            self.v_regressor_model = pickle.load(file)
        with open('./models/vectorizer_text.pkl', 'rb') as file:
            self.vectorizer_text = pickle.load(file)
        with open('./models/vectorizer_name.pkl', 'rb') as file:
            self.vectorizer_name = pickle.load(file)
        with open('./models/sentiment_analyzer.pkl', 'rb') as file:
            self.sentiment_analyzer = pickle.load(file)

    def predict(self, features):
        features_for_predict = self.__preprocessing(features)
        result = self.v_regressor_model.predict(features_for_predict)
        return result

    def __preprocessing(self, features_list):
        features_df = pd.DataFrame(np.hstack(features_list), columns=[])
        features_df = fill_missing_data(features_df)
        features_df['name'] = features_df.name.apply(cleanText)
        features_df['item_description_clean'] =
features_df.item_description.apply(cleanText)
        features_df['clean_category_name'] =
features_df.category_name.apply(cleanCat)
        features_df['brand_name'] = features_df.brand_name.apply(lambda x: x
if pd.isna(x) else x.lower())

        features_df = self.__add_sentiment(features_df)

        features_df["name"] = features_df["name"] +
features_df["brand_name_fixed"]
        features_df["text"] = features_df["name"] +
features_df["brand_name_fixed"] + features_df["processed_item_description"] +
features_df[
            "clean_category_name"]
        features_df.loc[features_df.encoded_cat.isnull(), "encoded_cat"] = -1

        tfidf_text = self.vectorizer_text.transform(features_df['text'])
        tfidf_name = self.vectorizer_name.fit_transform(features_df['name'])
        f_numeric = features_df[['shipping', 'encoded_cat']].values
        features_processed = np.hstack([tfidf_text, tfidf_name, f_numeric])
        return features_processed

    def __add_sentiment(self, features_df):
        neg = []
        neu = []
        pos = []
        compound = []
        for a in tqdm(features_df["text"].values):
            neg_value = self.sentiment_analyzer.polarity_scores(a)['neg']
            neu_value = self.sentiment_analyzer.polarity_scores(a)['neu']
            pos_value = self.sentiment_analyzer.polarity_scores(a)['pos']
            comp_value =
self.sentiment_analyzer.polarity_scores(a)['compound']
```

```
            neg.append(neg_value)
            neu.append(neu_value)
            pos.append(pos_value)
            compound.append(comp_value)

        features_df['neg'] = neg
        features_df['neu'] = neu
        features_df['pos'] = pos
        features_df['compound'] = compound
        return features_df

    def __split_categories_words(self, features_df):
        features_df['cat_split_count'] = features_df.category_name.apply(
            lambda x: x if pd.isna(x) else len(x.split('/')))
        features_df['general_cat'] = features_df.category_name.apply(lambda
x: x if pd.isna(x) else (x.split('/')[0]))
        features_df['sub_cat1'] = features_df.category_name.apply(lambda x: x
if pd.isna(x) else (x.split('/')[1]))
        features_df['sub_cat2'] = features_df.category_name.apply(lambda x: x
if pd.isna(x) else (x.split('/')[2]))

        features_df['name_len'] = features_df.brand_name.apply(lambda x: x if
pd.isna(x) else len(x))
        features_df['name_split'] = features_df.brand_name.apply(lambda x: x
if pd.isna(x) else x.split(' '))
        features_df['name_word_count'] = features_df.brand_name.apply(lambda
x: x if pd.isna(x) else len(x.split(' ')))

        features_df['desc_char_len'] =
features_df.item_description.apply(lambda x: x if pd.isna(x) else len(x))
        features_df['desc_arr'] = features_df.item_description.apply(lambda
x: x if pd.isna(x) else x.split(' '))
        features_df['count_desc_word'] = features_df.item_description.apply(
            lambda x: x if pd.isna(x) else len(x.split(' ')))

        brand_score =
dict(features_df[features_df.brand_name.notnull()]["brand_name"].value_counts
())
        processed_brand_name = []
        for index, i in tqdm(features_df.iterrows()):

            if pd.isnull(i.brand_name):
                words = i['name'].split()
                score = []
                for j in words:
                    if j in brand_score.keys():
                        score.append(brand_score[j])
                    else:  # if the word is not a brand name append -1
                        score.append(-1)

                if words == []:
                    processed_brand_name.append("missing")
                elif max(score) > 0:

processed_brand_name.append(words[score.index(max(score))])
                else:
                    processed_brand_name.append("missing")
            else:
                processed_brand_name.append((i.brand_name))
        features_df["brand_name_fixed"] = processed_brand_name
        return features_df
```

# Додаток В

Код файлу utils.py з додатковими допоміжними функціями.

```
import pandas as pd

def fill_missing_data(data):
    data.category_name.fillna(value = "Others", inplace = True)
    data.brand_name.fillna(value = "Not known", inplace = True)
    data.item_description.fillna(value = "No description given", inplace =
True)
    return data

def cleanText(x: str):
    if (pd.isna(x)):
        return 'missing'
    x = x.lower()
    x = x.strip()
    return x

def cleanCat(x: str):
    if (pd.isna(x)):
        return x
    x = x.replace("s ", " ")
    x = x.replace("s/", "/")
    x = x.replace('/'," ")
    return x.lower()
```