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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення для роботи з сирими даними для машинного навчання»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Владислав КРАВЧУК
(підпис)

Виконав: здобувач вищої освіти групи ПД-44

Владислав КРАВЧУК

Керівник: Оксана ЗОЛОТУХІНА
к.т.н., доцент

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кравчуку Владиславу Олеговичу

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення для роботи з сирими даними для машинного навчання»
керівник кваліфікаційної роботи к.т.н, доцент Оксана ЗОЛОТУХІНА,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи обробки сирих даних для машинного навчання, опис методів обробки текстів та числових таблиць, технічна документація з описом бібліотек для обробки сирих даних.

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

1. Аналіз застосунко обробки сирих даних для машинного навчання
2. Визначення засобів реалізації для програмного забезпечення обробки сирих даних для машинного навчання
3. Проектування програмного забезпечення для машинного навчання
4. Реалізація та тестування застосунку.

5. Перелік графічного матеріалу: *презентація*
    1. Аналіз аналогів.
    2. Вимоги до програмного забезпечення.
    3. Програмні засоби реалізації.
    4. Діаграма варіантів використання.
    5. Діаграма класів.
    6. Діаграма послідовності обробки тексту.
    7. Екранні форми.
    8. Апробація результатів дослідження.

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз особливостей обробки сирих даних для машинного навчання	05.03-13.03.2025	
3	Аналіз програмних засобів для обробки сирих даних	14.03-21.03.2025	
4	Визначення засобів реалізації для програмного забезпечення обробки сирих даних для машинного навчання	22.03-25.03.2025	
5	Проектування та моделювання програмного забезпечення обробки сирих даних для машинного навчання	26.03-15.04.2025	
6	Реалізація та тестування застосунку.	16.04-06.05.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Владислав КРАВЧУК
(підпис)

Керівник
кваліфікаційної роботи _____ Оксана ЗОЛОТУХІНА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 1 табл., 30 рис., 23 джерел.

Мета роботи – спрощення процесу обробки та підготовки сирих даних для машинного навчання.

Об'єкт дослідження – процеси обробки та підготовки сирих даних для машинного навчання.

Предмет дослідження – методи, інструменти та технології програмної реалізації обробки сирих даних.

Короткий зміст роботи: В роботі проаналізовано методи для обробки сирих даних для машинного навчання. Проаналізовані існуючі програмні реалізації обробки сирих даних таких програм як: Tableau, Microsoft Power BI, Apache Hadoop. Розроблено алгоритм роботи застосунку та програмно реалізовані ключові функціональні можливості, зокрема: завантаження текстових документів та числових таблиць для обробки, видалення шумів, пропусків та дублікатів у числових таблицях, моделі машинного навчання для аналізу даних, візуалізація результатів через діаграми та гістограми, створення таблиці по діапазону, видалення емоджі та HTML з тексту, лематизація тексту, векторизація тексту. В роботі використано бібліотеки NLTK та gensim для обробки тексту, Tkinter для створення користувацького інтерфейсу, Matplotlib для візуалізації даних, PyTorch та scikit-learn для моделей машинного навчання.

Сферою використання застосунку є обробка та аналіз сирих даних студентами у навчальних цілях.

КЛЮЧОВІ СЛОВА: СИРІ ДАНІ, МАШИННЕ НАВЧАННЯ, ТОКЕНІЗАЦІЯ, ЛЕМАТИЗАЦІЯ, ЧИСЛОВІ ТАБЛИЦІ, МОДЕЛІ, PYTHON, MATPLOTLIB, PYTORCH, GENSIM, SCIKIT-LEARN.

ЗМІСТ

Перелік умовних позначень

ML - Machine Learning

IDE - Integrated Development Environment

UML - Unified Modeling Language

GUI - Graphical User Interface

Tk - Tkinter

ВСТУП

Актуальність: У сучасних умовах інтенсивного розвитку машинного навчання, ключовим етапом створення ефективних моделей є якісна підготовка сирих даних. Саме від якості обробки даних напряму залежить точність, продуктивність і стабільність моделі.

Сирі дані за своєю природою містять велику кількість шуму, пропущених значень, аномалій, нестандартних форматів, повторів і непотрібної інформації. Наприклад, в текстах це можуть бути HTML-теги, стоп-слова, спеціальні символи чи емодзі, у числових таблицях — нульові або відсутні значення. Без попередньої обробки такі дані можуть призвести до некоректних висновків моделі, зниження точності класифікації чи регресії, або навіть до повного провалу обчислень.

Враховуючи активне впровадження ML у різні сфери — медицину, економіку, освіту, транспорт, кібербезпеку — актуальність обробки сирих даних зростає з кожним роком. Вона є базовою умовою створення надійних інтелектуальних систем, що здатні приймати рішення, прогнозувати тенденції або виявляти закономірності. Таким чином, дослідження та реалізація ефективних підходів до підготовки даних є надзвичайно важливим завданням сучасної прикладної інформатики та аналітики даних.

Об’єктом дослідження є процеси обробки та підготовки сирих даних для машинного навчання.

Предметом дослідження є методи, інструменти та технології програмної реалізації обробки сирих даних.

Метою роботи є Спрощення процесу обробки та підготовки сирих даних для машинного навчання.

Для досягнення поставленої мети в бакалаврській роботі було вирішено наступні задачі:

1. Проведений аналіз існуючих аналогів.
2. Проаналізовано інструменти для роботи з сирими даними.
3. Сформульовані функціональні та нефункціональні вимоги до програмного забезпечення.
4. Розроблена архітектура застосунку.
5. Реалізувати програмні модулі застосунку обробки сирих даних.
6. Проведено тестування застосунку.

Практична значущість результатів полягає створенні інструменту для обробки сирих числових та текстових даних різними способами з подальшим збереженням результатів для використання в методах машинного навчання.

1 АНАЛІЗ ЗАСТОСУНКУ ДЛЯ ОБРОБКИ СИРИХ ДАНИХ

1.1 Актуальність теми та особливості обробки сирих даних

У сучасному світі зростає обсяг цифрових даних, а їхнє значення для наукових досліджень, бізнес аналітики та систем автоматизації постійно зростає. Проте більшість даних є неструктуровані, неповні та зашумлені. Без належного очищення даних неможливо досягти високої точності моделей машинного навчання.

До особливостей обробки сирих даних входять:

1.Виявлення та обробка пропущених значень:

Сирі дані часто містять пропущені значення, які необхідно заповнити або видалити ознаку.

2.Видалення шуму та аномалій:

Наявність аномальних значень має сильний вплив на результати аналізу даних. Важливо буде завдяки моделі виявлення аномалій видаляти значення.

3. Форматування та уніфікація даних

Часто сирі дані представлені у різних форматах - числові, текстові, дати тощо. Для забезпечення можливості їхньої обробки алгоритмами, необхідно привести все до єдиного формату.

4.Кодування категоріальних змінних:

Для використання даних для машинного навчання необхідно застосувати техніки кодування змінних.

5.Масштабування та нормалізація:

Нормалізація числових даних дозволить уникнути домінування ознак з великим діапазоном значень.

6.Обробка текстових даних.

Робота з неструктурованими текстовими даними, потребує проведення токенізації, лематизації та векторизації.

Ці характеристики в обробки даних безпосередньо впливають на точність результатів машинного навчання. Надавши користувачу простий інтерфейс та візуалізацію можна можна досягти ефективної взаємодії з додатком.

1.2 Особливості розробки Windows додатку

Специфіка розробки визначається декількома ключовими факторами: цільова аудиторія, функціонал який необхідний для аудиторії, вимоги та особливості інтерфейсу. Розуміючи ці фактори дозволить успішно створити додаток, який надасть користувачам те що вони потребують. Дослідивши предметну область надало корисну інформацію для адаптування вимог до додатку, можливості реалізації необхідного функціоналу та приклади інтерфейсу користувача. Конкретні інструменти та технології мають значний вплив на кінцевий результат та досвід користувача.

Крім функціональності, головним етапом є створення інтерфейсу користувача, оскільки саме він надає змогу взаємодіяти з функціями додатку. Головними вимогами до інтерфейсу є інтуїтивність у користуванні, візуальна привабливість. Інтуїтивність у використанні надають користувачу сприяє швидкому розумінню у навігації по програмі та покращить користувацький досвід з додатком.

Зробивши розрахунок на подальше масштабування надасть суттєві переваги для швидкої реалізації потреб користувачів та збереже від шкоди продуктивності. Масштабованість може гарантувати нам залишення ефективності програми, розширювати та покращувати старі функції та надасть надасть можливість до легкого створення нових що забезпечує додатку підтримку конкурентоспроможності на ринку.

Загальний огляд розділу розробки для Windows має важливе значення для успіху додатку. Виконання простих факторів надасть можливість реалізації успішного додатку та завдяки маштабованості можна досягти простоти у наступних оновленнях додатку. Врахувавши особливість платформи можна

створити безперебійний додаток для будь якого комп’ютера.

1.3 Аналіз аналогів

Для розуміння особливості предметної галузі, необхідно провести аналіз реалізованих програмних забезпечень з виділенням переваг та недоліків. Для аналізу було обрано 3 програмних забезпечення:

1.3.1 Tableau

Переваги:

Інтуїтивно зрозумілий інтерфейс - це надає користувачу легкість у створенні інтерактивних дашбордів та візуалізації даних.

можливість вбудовування візуалізацій - головною перевагою Tableau є візуалізація. Дашборди легко оптимізувати під запити, створюються глибокі діаграми та гістограми.

Розширену підтримку джерел даних - програма підтримує підключення до бази даних та може аналізувати джерела типу Excel, SQL Server, Google BigQuery, Amazon Redshift.

Підтримка різних платформ - програма містить свій варіант використання встановленого на комп’ютер, через веб браузер та на IOS і Android.

Екранні форми програми Tableau буде зображено на рисунку 1.1.

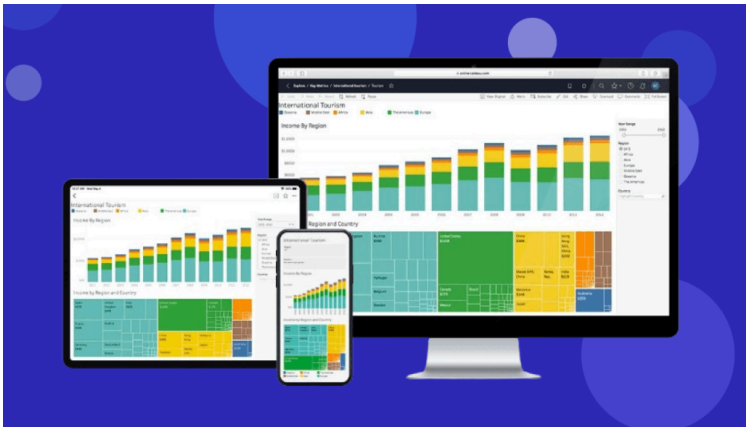


Рис. 1.1. Екранні форми Tableau

Недоліки:

Обмежені можливості обробки сирих даних - програма чудовий інструмент для візуалізації даних, але вона не підтримує глибоку аналітику. Кращим варіантом буде виконання обробки даних за межами програми Tableau та надавати вже оброблені дані.

1.3.2 Microsoft Power BI:

Глибока інтеграція з продуктами Microsoft - Більшість дій у додатку виконується у середовищі Microsoft 365.

Простота використання - Інтерфейс Microsoft Power BI подібний до Excel, що надає змогу швидко адаптуватись до середовища.

Інструменти машинного навчання - Завдяки інтеграції з Azure Machine Learning можливо будувати прогнози моделей класифікації прямо у Microsoft Power BI

Гнучке керування доступом - надає легке керування ролей та прав доступу що забезпечує безпечну корпоративну політику.

Підтримка різних платформ - Microsoft Power BI підтримує декілька платформ що надає зручність у використанні при різних сценаріях. Програма підтримується наступними платформами: Windows, Веб браузер та IOS і Android.

Екранні форми Microsoft Power BI зображено на рисунку 1.2.

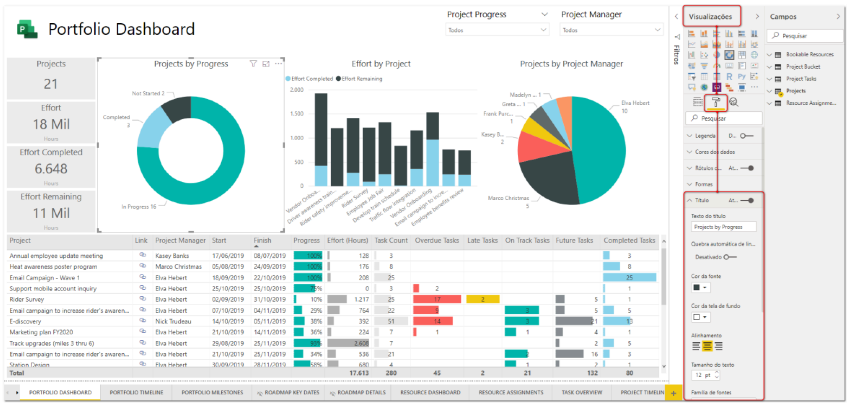


Рис. 1.2. Екранні форми Microsoft Power BI

Недоліки:

Не всі візуалізації адаптовані для мобільного перегляду - частина візуалізації даних недоступне до мобільної версії додатку.

Затримки при обробці дуже великих обсягів даних - при обробці величезних таблиць даних система працює повільно без переходу на хмарну версію.

1.3.3 Apache Hadoop

Переваги:

Висока продуктивність для Big Data - це найкращий варіант для обробки величезної кількості інформації, яке розділяє обчислення на десятки або сотні серверів одночасно.

Масштабованість без обмежень - систему легко розширити додаючи нові вузли, що робить програму ефективною для обробки постійно зростаючих обсягів інформації.

Відкритий код - містить величезну спільноту розробників та доступним кодом.

Гнучкість у використанні супутніх інструментів - Apache Hadoop часто використовує такі інструменти як Apache Hive для Sql запитів, Spark для обробки в реальному часі, HBase для взаємодії з NOSql, що дозволяє адаптувати систему під конкретні потреби.

Ідеальний для архівування й обробки сирих логів, потокових даних, журналів - Завдяки розподіленій файлові системі можна легко зберігати неструктуровані або напів структуровані файли.

Недоліки:

Складна інсталяція та налаштування - потреба досвіду роботи з Linux мережею, безпекою, хмарними системами та адмініструванням. Для корпоративного використання необхідно виділяти окрему хмарну платформу.

Відсутність інтерфейсу для користувачів без технічного досвіду - для роботи з Hadoop потрібні знання програмування, Bash, SQL. Бізнес-користувач не може

використовувати платформу без технічної допомоги. Екосистема Apache Hadoop зображена на рисунку 1.3.

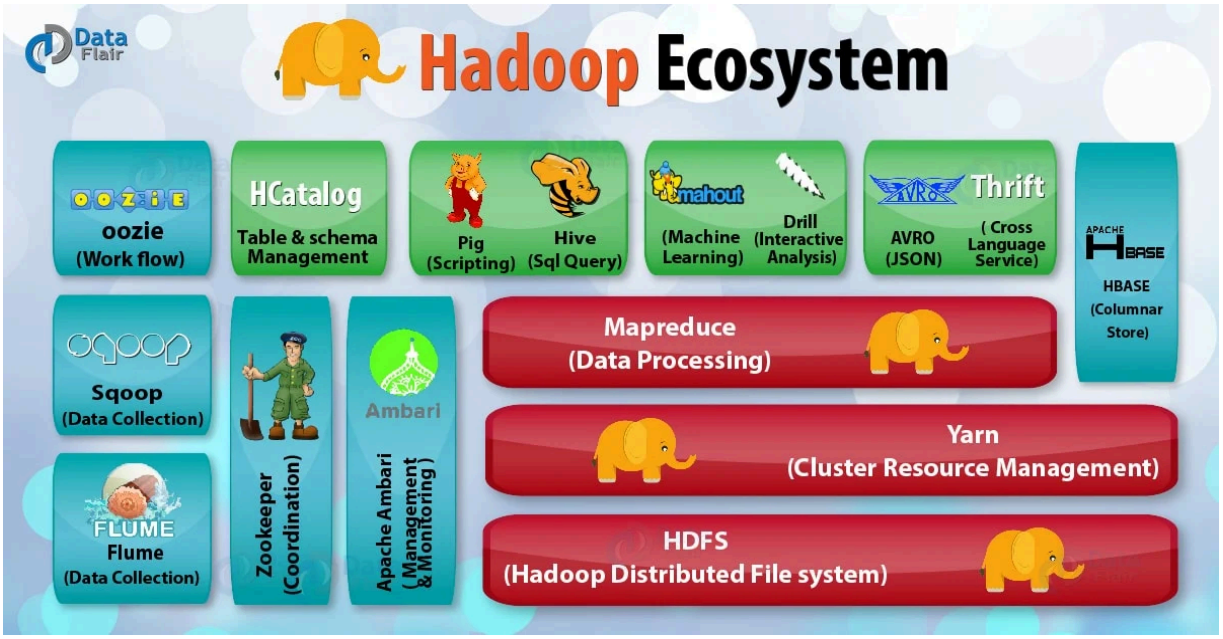


Рис. 1.3. Екосистема Apache Hadoop

Відповідно до описаних переваг та недоліків була створена таблиця.

Таблиця 1.1

Порівняння існуючих аналогів

Показник	**Tableau**	**Microsoft Power BI**	**Apache Hadoop**	**Adaptation**
Видалення дублікатів числових даних	-	+	+	+
Нормалізація та стандартизація числових даних	+	+	+	+
Створення набору чисел в заданому діапазоні	+	+	-	+

Продовження таблиці 1.1

Порівняння існуючих аналогів

Показник	**Tableau**	**Microsoft Power BI**	**Apache Hadoop**	**Adaptation**
Видалення записів таблиці даних з пропущеними значеннями	-	+	+	+
Переведення тексту в нижній регістр	+	+	+	+
Очищення тексту від HTML, емоджі, спецсимволів та стоп-слів	-	+	-	+
Нормалізація слів тексту	-	-	лематизація, стемінг	Лематизація
Статистичні показники тексту	Довжина тексту	Довжина тексту	Довжина тексту, Кількість слів, Частота слів, тд	Загальна кількість токенів та унікальних токенів, середня довжина токена, ТОП-10 слів
Підтримка обробки текстів українською мовою	-	-	+	+
Візуалізація результатів обробки	+	+	-	+
Векторизація тексту	-	-	-	+

З таблиці видно що реалізує необхідні функції користувачу які відсутні в аналогах. Наприклад обробка текстових даних , часткову обробку яку підтримує Microsoft Power BI завдяки інтеграції з Power Query, у моєму додатку виконано напряму та дозволяє повністю очистити, лематизувати, токенізувати та векторизувати Український текст.

1.4 Визначення функціональних та нефункціональних вимог

Визначення функціональних та нефункціональних прямо залежить на успіх виконання розробки додатку. Правильно визначивши вимоги можна встановити чіткі вказівки та очікування від майбутньої програми. Функціональні вимоги програми описують внутрішню роботу системи та що вона повинна виконувати щоб задовольнити аудиторію. Нефункціональні вимоги зосереджені на рисах продуктивності, зручності, безпеки та інструментів.

Проаналізувавши предметну область було встановлені наступні функціональні та нефункціональні вимоги для розробки програмного забезпечення обробки сирих на Windows:

1. Функції для обробки текстових даних:
- переведення тексту в нижній регістр;
- видалення HTML, емоджі, спецсимволів та стоп-слів;
- нормалізація слів тексту;
- векторизація тексту;
- розрахунок загальної кількості токенів та унікальних токенів;
- розрахунок середньої довжини токена;
- формування списку 10 найуживаніших слів.
2. Функції для обробки числових даних:
- стандартизація значень;
- видалення дублікатів;
- видалення записів з пропущеними значеннями;
- візуалізація результатів обробки в графічному вигляді;
- створення набору чисел в заданому діапазоні
3. Імпорт сирих даних.
4. Експорт оброблених даних.
5. Аутентифікація користувача.

Нефункціональні вимоги:

1. Графічне представлення даних повинно бути у вигляді гістограми та

діаграми розсіювання.

2. Імпорт даних для обробки числових таблиць повинен здійснюватись у форматі CSV.

3. Імпорт даних для обробки тексту повинен здійснюватись у форматі Word або txt.

4. Експорт результатів обробки числових таблиць має бути у формат CSV.

5. Експорт результатів обробки текстових даних має бути у формат Word або txt.

6. При обробці текстових даних повинна бути підтримка української мови.

7. Аутентифікація користувачів повинна бути реалізована зі збереженням логіну та паролю на MongoDB, що надасть можливість масштабувати у майбутньому збереження результатів обробки на хмарі.

1.5 Засоби реалізації

Засоби реалізації - це сукупність інструментів та технологій для реалізації програмного продукту. Для створення додатку обробки сирих даних для машинного навчання застосуєм наступні інструменти: Tkinter для створення інтерфейсу користувача, Python як головну мову програмування, PyTorch та Scikit-learn для реалізації машинного навчання, NumPy, Pandas та Gensim для реалізації обробки числових таблиць та тексту, Matplotlib для генерації діаграми розсіювання та гістограми на основі даних обробки числових таблиць та MongoDB для збереження логіну та паролю користувачів.

1.5.1 Python

Python — це одна з найпопулярніших мов програмування у світі, яка завоювала особливу популярність у сфері машинного навчання та аналізу даних завдяки своїй простоті, гнучкості, великій екосистемі бібліотек та активній

спільноті розробників. Вона ідеально підходить для реалізації повного циклу машинного навчання — від обробки сирих даних до побудови, тренування та оцінки моделей.

Однією з головних переваг Python є наявність спеціалізованих бібліотек для обробки сирих даних. Наприклад, pandas забезпечує зручні інструменти для очищення, трансформації та фільтрації табличних даних; NumPy дозволяє ефективно працювати з числовими масивами; OpenCV і Pillow використовуються для обробки зображень, а librosa — для обробки аудіо. Для тексту доступні потужні засоби, зокрема NLTK, spaCy та re, які дозволяють виконувати лематизацію, токенізацію, видалення шуму, роботу з регулярними виразами та інші операції, необхідні для підготовки тексту до аналізу.

Ще однією ключовою перевагою Python є його інтеграція з бібліотеками машинного навчання, такими як scikit-learn, TensorFlow, Keras, PyTorch. Ці бібліотеки забезпечують побудову та тренування моделей різного типу: від простих лінійних класифікаторів до нейронних мереж, здатних працювати з зображеннями, мовою, аудіо й часовими рядами.

Python підтримує структурований та модульний підхід до програмування, що дозволяє розділити проект на логічні частини (наприклад, обробка даних, машинне навчання, візуалізація, збереження результатів). Завдяки цьому Python ідеально підходить не лише для експериментів, а й для створення повноцінних інтелектуальних систем і застосунків.

Таким чином, Python є універсальним інструментом для розробки проектів у сфері машинного навчання, де обробка сирих даних є критично важливою складовою. Його можливості дозволяють розробляти рішення, які є продуктивними, масштабованими та ефективними як у дослідницькому, так і у прикладному середовищі.

1.5.2 PyTorch

PyTorch, є однією з найпотужніших відкритих бібліотек орієнтованих на моделювання нейронних мереж та є однією з найвідоміших платформ глибокого

навчання у світі. PyTorch надає реалізацію функцій обробки даних та можливість створення класифікації, кластеризації та регресії. PyTorch підтримує навчання на GPU, багатопроцесорну обробку, а також має тісну інтеграцію з такими бібліотеками, як NumPy, scikit-learn.

PyTorch широко використовується як у промислових проектах, так і в академічних дослідженнях. Завдяки модульності та прозорості став стандартом для прототипування і реалізації складних моделей глибокого навчання.

Перевагами PyTorch є гнучкість побудови моделей, повна підтримка GPU, автоматичне диференціювання, готові модулі для глибокого навчання та активна спільнота та розвиток . Гнучкість побудови моделей надає легкість у модифікації моделей навчання що на відміну від TensorFlow у якому граф фіксований дає перевагу у масштабованості.

Повна підтримка GPU пришвидшує обчислення приблизно 10 - 100 разів. Це надасть швидкість обробки даних та чудовий користувацький досвід. Автоматичне диференціювання яке спрощує прописування градієнтів.

Активна спільнота надає бібліотеці постійні оновлення та використовується в Meta, Tesla, OpenAI. Приклад реалізації моделі регресії зображено на рисунку 1.4

```
# === Регресія ===
try:
    y = torch.tensor(df[df.columns[-1]].values, dtype=torch.float32).unsqueeze(1)
    X_train, X_test, y_train, y_test = train_test_split(X_tensor, y, test_size=0.2)

    model = nn.Sequential(
        nn.Linear(X_tensor.shape[1], 16),
        nn.ReLU(),
        nn.Linear(16, 1)
    )
    loss_fn = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=0.01)

    for _ in range(100):
        y_pred = model(X_train)
        loss = loss_fn(y_pred, y_train)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    y_pred = model(X_test).detach().numpy()
    mse = mean_squared_error(y_test.numpy(), y_pred)
    regression_result['message'] = f"Регресія MSE: {mse:.2f}"
    regression_result['torch_mse'] = mse
```

Рис. 1.4 приклад реалізації моделі регресії через PyTorch

1.5.3 Matplotlib

Matplotlib - це популярна бібліотека для візуалізації 3D графіків, діаграм, гістограм та інш.. Головні можливості які ми використаєм це побудова діаграм та гістограм, можливість збереження у форматі PNG, та чудову інтеграцію з Tkinter та Pandas.

Головними перевагами Matplotlib які зробили її основною бібліотекою візуалізації в Python це найбільша гнучкість для візуалізації, велика кількість типів графіків, сумісність з іншими бібліотеками та кастомізація стилів. Гнучкість бібліотеки надає повний контроль над всіма елементами графіку такі як розмір шрифту та розташування елементів. Бібліотека має велику кількість типів графіків такі як лінійні, стовпчикові, кругові, точкові, гістограми, boxplots, heatmaps та 3D-поверхні. Чудовим доповненням стає можливість створення анімації та інтерактивних графіків за допомогою FuncAnimation. На візуалізацію теж впливає можливість кастомізувати стилі та готові теми для використання зі зручним збереженням у форматі PNG. Приклад створення гістограми за допомогою MatPlotlib зображено на рисунку 1.5.

```
for name in feature_names:
    if name not in df.columns:
        continue

    data = df[name].dropna().values

    frame = tk.Frame(notebook)
    notebook.add(frame, text=name)

    fig, ax = plt.subplots(figsize=(5, 4))
    n, bins, patches = ax.hist(data, bins=20, color='skyblue', edgecolor='black')
    ax.set_title(f"Гістограма: {name}")
    ax.set_xlabel(name)
    ax.set_ylabel("Кількість")

    for j in range(len(patches)):
        bin_center = 0.5 * (bins[j] + bins[j + 1])
        height = patches[j].get_height()
        if height > 0:
            ax.text(bin_center, height, str(int(height)), ha='center', va='bottom', fontsize=8)

    canvas = FigureCanvasTkAgg(fig, master=frame)
    canvas.draw()
    canvas.get_tk_widget().pack()

    def save_hist(fig=fig, name=name):
        file_path = filedialog.asksaveasfilename(
            defaultextension=".png",
            initialfile=f"{name}.png",
            filetypes=[("PNG", "*.png")]
        )
        if file_path:
            fig.savefig(file_path)
            messagebox.showinfo( title: "Збережено",  message: f"Гістограму збережено:\n{file_path}")

    tk.Button(frame, text="Зберегти", command=save_hist).pack(pady=5)
```

Рис. 1.5 Приклад створення гістограми за допомогою MatPlotlib

1.5.4 Gensim

Gensim - це високорівнева бібліотека Python призначена для векторного представлення тексту. Завдяки сумісності з іншими NLP-інструментами текст можна лематизувати, попередньо очистити та векторизувати. З переваг потрібно виділити оптимізацію для роботи з текстом та підтримку потокової обробки.

Оптимізація для роботи з текстами надає можливість створювати ефективні алгоритми Word2Vec, Doc2Vec, FastText, та підтримка потокової обробки дозволить не завантажувати весь корпус у пам’ять користувача. Завдяки підтримки великих корпусів надає змогу обробляти тексти які не поміщаються в оперативну пам’ять. Підтримка моделювання яке надає можливість розуміння програмою тематики тексту та його виділення .

1.5.5 Pandas

Pandas - це потужна бібліотека яка пропонує широкий спектр можливостей для обробки та аналізу табличних даних. У контексті додатку обробки даних потрібно використовувати для взаємодії з таблицями користувача, очищати їх та аналізувати. Основною структурою даних для бібліотеки є двовимірні та одновимірні таблиці.

За допомогою Pandas та її інтеграцією з бібліотеками NumPy та Matplotlib можна підготувати взаємодію даних з бібліотеками ML, візуалізувати результати та зберігати результати.

Крім того, Pandas має простий синтаксис який надасть швидку та просту попередню обробку таблиць від дублікатів та заповнення пропусків до об’єднання даних за потреби реалізації. З простого синтаксису можна виділити ще просте створення сортування, групування та фільтрацію таблиць.

Підсумовуючи Pandas - це необхідний інструмент для взаємодії з таблицями, який інтегрується з бібліотеками візуалізації та бібліотекою наукових обчислень NumPy, легко створюється видалення дублікатів та заповнення пропусків та

можливе створення функції об’єднання даних у майбутньому. Реалізація зчитування та попередньої очистки зображено на рисунку 1.6

```
try:
        df = pd.read_csv(path, delimiter=",")
except Exception as e:
    messagebox.showerror( title: "Помилка",  message: f"Не вдалося прочитати CSV: {e}")
    return

# Попереднє очищення:
df.drop_duplicates(inplace=True)
df.dropna(inplace=True)

for col in df.select_dtypes(include='number').columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]

numeric = df.select_dtypes(include='number')
if numeric.empty:
    messagebox.showerror( title: "Помилка",  message: "Файл не містить числових даних.")
    return
```

Рис. 1.6 приклад реалізації зчитування CSV таблиці та попередньої очистки

1.5.6 NumPy

NumPy - це фундаментальна бібліотека для наукових обчислень на мові Python, що надає підтримку багатовимірним масивом і матриць з величезними кількостями функції для їх обробки.

NumPy є основою для працювання таких бібліотек як Pandas, Scikit-learn. За допомогою NumPy можна створювати ефективніші масиви які будуть оброблятись, реалізовувати математичні операції яке розширює можливості для створення аналізу.

Крім того, NumPy має високу продуктивність що надається завдяки векторизованим операціям. Це надає пришвидшення обробок та взаємодій з масивами які використовуються у аналізі даних.

Підсумовуючи NumPy - це фундамент для наукових обчислень та підтримки багатовимірних масивів для обробки даних та їхнього аналізу. Бібліотеки Pandas, Scikit-learn неможливо створити ефективними без бібліотеки NumPy.

1.5.7 Scikit-learn

Scikit-learn - це одна з найпопулярніших бібліотек машинного навчання. Бібліотека містить прості та ефективні інструменти для аналізу даних, такі як: моделі, здатність до реалізації масштабування, нормалізації та кодування змінних.

Scikit-learn надає можливість створити поєднання з бібліотекою глибокого навчання PyTorch що створить ряд переваг. Поєднавши створення ML-моделей та створення PyTorch нейромереж та створення гнучкості з GPU-прискоренням.

Підсумовуючи Scikit-learn - це чудова бібліотека яка містить функціонал масштабування, нормалізації та кодуванні даних. Дозволяє створити класичні ML-моделі, та має можливість для інтеграції з бібліотекою глибокого навчання PyTorch.

1.5.8 Tkinter

Tkinter — це вбудована в Python бібліотека для створення графічного інтерфейсу користувача, яка є стандартним інструментом для розробки віконних застосунків. Вона надає прості й ефективні засоби для побудови інтерфейсів будь-якої складності — від найпростіших вікон із кнопками до складних багатовіконних систем з інтерактивними елементами, вкладками, меню, діалогами та валідацією введення.

Tkinter базується на інтерфейсі Tcl/Tk, що був адаптований для Python, і є кросплатформним — тобто застосунки, створені з його допомогою, можуть працювати однаково як у Windows, так і в Linux та mac OS без зміни коду. Основна перевага цієї бібліотеки — її доступність «з коробки» без необхідності встановлення додаткових пакетів, що особливо зручно для студентів, викладачів та початківців у програмуванні.

Основними елементами графічного інтерфейсу в Tkinter є вікна (Tk), фрейми, кнопки, поля введення, текстові поля, мітки, списки, чекбокси, радіокнопки, мен та інші віджети. Всі вони можуть бути зручно комбіновані та налаштовані завдяки гнучкій системі керування розміщенням.

Tkinter підтримує роботу з подіями — кожен елемент інтерфейсу може реагувати на дії користувача: натискання клавіш, кнопок миші, введення даних, вибір зі списку тощо. Це дозволяє створювати динамічні інтерфейси з гнучкою логікою взаємодії.

У межах реалізованого програмного забезпечення Tkinter використовується для створення всіх віконних елементів: вікна входу та реєстрації, головного меню, підменю для обробки тексту, таблиць. Кожне вікно оформлене з урахуванням стилю, зручності користування та адаптації під фіксовану роздільну здатніст.

Оскільки Tkinter легко інтегрується з іншими модулями Python, його можна поєднувати з обчислювальними модулями машинного навчання, обробки тексту, роботи з числовими таблицями, тощо. Завдяки цьому забезпечується повна функціональність системи — користувач завантажує файл, взаємодіє з інтерфейсом, а далі система виконує обробку й відображає результат у тому ж вікні.

Загалом, Tkinter є надійним, зручним та універсальним рішенням для створення GUI в Python-проектах. Його використання дозволяє створювати повноцінні застосунки, придатні як для особистого користування, так і для наукових, навчальних або бізнес-задач, без необхідності залучення сторонніх інструментів.

Створення інтерфейсу за допомогою Tkinter зображено на рисунку 1.7.

```
def main_application():  1 usage
    root = tk.Tk()
    root.title("Обробка сирого тексту (українська)")
    root.geometry("1900x1080")

    raw_text = ""
    cleaned_text = ""
    tokens = []
    vectorization_results = {}

    # Створення основного фрейму
    main_frame = ttk.Frame(root)
    main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

    # Фрейм для кнопок
    button_frame = ttk.Frame(main_frame)
    button_frame.pack(fill=tk.X, pady=5)

    # Кнопки
    ttk.Button(button_frame, text="Завантажити Word файл", command=lambda: load_file()).pack(side=tk.LEFT, padx=5)
    ttk.Button(button_frame, text="Обробити текст", command=lambda: process_text()).pack(side=tk.LEFT, padx=5)
    ttk.Button(button_frame, text="Зберегти результат", command=lambda: save_text()).pack(side=tk.LEFT, padx=5)
    ttk.Button(button_frame, text="Векторизувати", command=lambda: vectorize()).pack(side=tk.LEFT, padx=5)
    ttk.Button(button_frame, text="Статистика", command=lambda: show_stats()).pack(side=tk.LEFT, padx=5)
```

Рис. 1.7 приклад реалізації інтерфейсу через Tkinter та створенню кнопок

1.5.9 MongoDB

База даних - це організована сукупність даних яка описує їх характеристику та взаємодію між елементами. Для простого користувача виділимо 2 види бази даних - SQL та NoSQL. Для проекту було використано NoSQL базу даних - MongoDB.

MongoDB є однією з найпопулярніших NoSQL-систем у світі. Можна виділити наступні характеристики MongoDB:

Документо Орієнтована модель позначається у збереженні колекцій та документів замість таблиць з рядками. Кожний документ містить гнучкість структури даних у вигляді пари «ключ-значення».

Гнучкість схеми не вимагає від кожного документу визначену структуру, кожний документ містить свою особисту структуру.

Масштабованість надає підтримку горизонтального масштабування що надає можливість обробляти великі обсяги даних на різних серверах.

Використовується власна потужна мова запитів яка підтримує фільтрацію, агрегацію, сортування та інші операції.

Використовується для зберіганню логінів та паролів користувачів.

Фрагмент підключення до бази даних зображено на рисунку 1.8

```
client = MongoClient(
    "mongodb+srv://vladmagony:789456vladi@cluster0.ueidgmg.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0")
db = client["user_auth"]
users_collection = db["users"]
```

Рис. 1.8 приклад підключення до бази даних

2 ПРОЕКТУВАННЯ ОБРОБКИ СИРИХ ДАНИХ

2.1 Етапи розробки програмного забезпечення

Розробка програмного забезпечення на Windows містить 4 основних етапа: аналіз вимог, проектування системи, розробка системи та тестування. Від якості виконання кожного етапу залежить якість фінального продукту та його конкурентоспроможність на ринку. Кожний етап базується на результаті виконання попередніх етапів. Розглянемо кожний етап детальніше

Перший етап - це аналіз вимог. До етапу аналізу вимог входить, збір вимог замовника для встановлення функціональних та нефункціональних вимог програми та формування технічного завдання. Аналіз вимог також передбачає аналіз аналогів на ринку для виділення переваг та недоліків конкурентів. Фаза аналізу вимог завершиться після збору вимог замовника, аналізу програм аналогів та встановлення функціональних та нефункціональних вимог.

Другим етапом стає проектування системи на яком створюватиметься візуальне відображення через UML діаграми майбутнього продукту. Етап передбачає визначення структури програми, включаючи діаграму класів, діяльності, варіантів використання. Проектування системи також передбачає розробка макету інтерфейса програми та вибір відповідних технологій, інструментів та фреймворків для повної реалізації функціональних вимог. Другий етап завершується після створення необхідних UML діаграм, макету інтерфейса, визначення структури програми та вибору технологій, інструментів та фреймворків для роботи

Третім етапом є етап розробки системи. На Цьому етапі відбувається повна реалізація функціональних та нефункціональних вимог. Етап передбачає повне створення інтерфейсу користувача, імпортування всіх необхідних бібліотек та фреймворків та повне створення логіки функцій. Завершення третього етапу стає розроблена програма готова перейти на стадію тестування для виявлення

помилок. Після виявлення помилок проект повертається у третю стадію для їхнього виправлення.

Четвертим етапом розробки стане етап тестування. Після завершення етапу розробки потрібно провести перевірку для виявлення помилок та усунути їх. Після знаходження помилки, інформація передається розробникам та вони його виправляють. Фаза тестування завершиться як програма буде повністю відповідати функціональним вимогам та помилки будуть усунуті.

2.2 Засоби UML для проектування застосунку

Головним інструментом у етапі проектування програми є створення UML діаграм. UML - це мова моделювання яка використовується для візуалізації, специфікації, конструюванню та документації програми. Діаграми UML візуально описує архітектуру, логіку та поведінку системи. UML діаграми це важливий інструмент для кращого розуміння складних структур.

Виділимо декілька основних варіантів діаграм UML, кожна з яких має свою роль у процесі проектування:

Діаграма варіантів використання відображає функціональні вимоги з сторони користувача. Діаграма описує дії програми не заглиблюючись у деталі реалізації. Діаграма варіантів використання створює візуальне зображення функціональності системи.

Діаграма діяльності головною задачею є відображення логіки та алгоритмів програми що надасть можливість виявити точки оптимізації або виявити вразливості програми.

Діаграма класів є центральною діаграмою у моделюванні. Діаграма класів демонструє атрибути класів, а також зв'язки між ними.

Діаграма послідовностей відображає взаємодію між об’єктами у системі в часі. Мета діаграми послідовностей відобразити послідовність ініціалізації виклику класу та взаємодії класів.

Підводячи підсумок, діаграми UML - це чудовий інструмент для етапу проектування який надає краще розуміння проекту та зменшує кількість помилок на пізніх етапах розробки. Успіх створення програми залежить від правильності створення UML діаграм.

2.3 Діаграма варіантів використання.

Діаграма варіантів використання забезпечить графічне представлення вимог системи та можливості взаємодії користувача з системою. Візуалізація дозволить швидко зрозуміти повний масштаб системи та створює краще розуміння програми.

З переваг для створення діаграми варіантів використання можна виділити наступні:

Візуальне представлення можливостей взаємодії користувача з системою. Діаграма надає легкий спосіб представлення функціональних вимог до системи.

Простота у комунікації з зацікавленими сторонами. Дозволяє уникнути непорозумінь з функціоналом додатку.

Ідентифікація системних вимог. Діаграма полегшує ідентифікації та аналізу системних вимог шляхом моделюванню взаємодії користувача та можливостей системи.

Створена діаграма варіантів використання зображена на рисунку 2.1.

Рис. 2.1 Діаграма варіантів використання

Розглянемо детальніше основний функціонал:

Реєстрація користувача буде перевіряти довжину логіну користувача та валідності пароля для нових користувачів. Після підтвердження правильності логіну та паролю користувача переходить в меню авторизації.

Авторизація перевіряє логін та пароль через MongoDB. Після успішної авторизації користувач переходить до головного меню.

Обробка числових таблиць виконує ряд задач, а саме зчитування файлу, очищенню від дублікатів, пропуску та шумів та масштабуванню.

Обираючи ознаки для обробки запускається виклик методів класифікації, кластеризації та регресії.

Успішне завершення методів дає змогу створювати та зберігати діаграму розсіювання та гістограму.

Створення таблиці за діапазоном надає можливість створення таблиці даних видаляючи та створюючи діапазон чисел у стовпці з подальшою можливістю збереження.

Обробка тексту надає можливість завантажити тексту у програму та викликає методи очищення від HTML, емоджі, пунктуації, стоп-слів, лематизує та токенізує текст.

Оброблений текст користувач може зберегти локально.

Векторизація тексту викликає методи BoW, TF-IDF, Word2Vec, FastText.

Статистика по токенам надасть користувачу загальну кількість токенів у тексті, кількість унікальних токенів, середню довжину та 10 найуживаніших слів в тексті.

2.4 Діаграма діяльності

Моделювання діаграми діяльності дозволяє моделювати логіку процесів та послідовності дій. Створення діаграми діяльності покращить розуміння поведінки системи.

Коректно спроектовані діаграми діяльності мають ряд основних, серед них варто виділити наступні:

Візуалізація процесу. Діаграма повністю відображає можливості системи що забезпечує візуальне представлення роботи системи, зображує послідовність дій та рішень які необхідні системі.

Розуміння поведінки системи. Діаграма діяльності надає візуалізацію роботи системи та створює краще розуміння системи з розумінням які є можливості взаємодії користувача з системою

Забезпечує перевірку вимог. Звертаючи увагу на діаграму діяльності можна перевірити відповідність до вимог очікуваної аудиторії.

Дозволяє виявити потенційні вразливі місця додатка. Діаграма діяльності

допоможе у виявленні потенційних вразливостей додатка, дозволяючи зробити оптимізацію роботи додатку що покращить користувацький досвід. Створена діаграма діяльності для авторизації користувача зображена на рисунку 2.2 та діаграма діяльності обробки тексту зображено на рисунку 2.3.

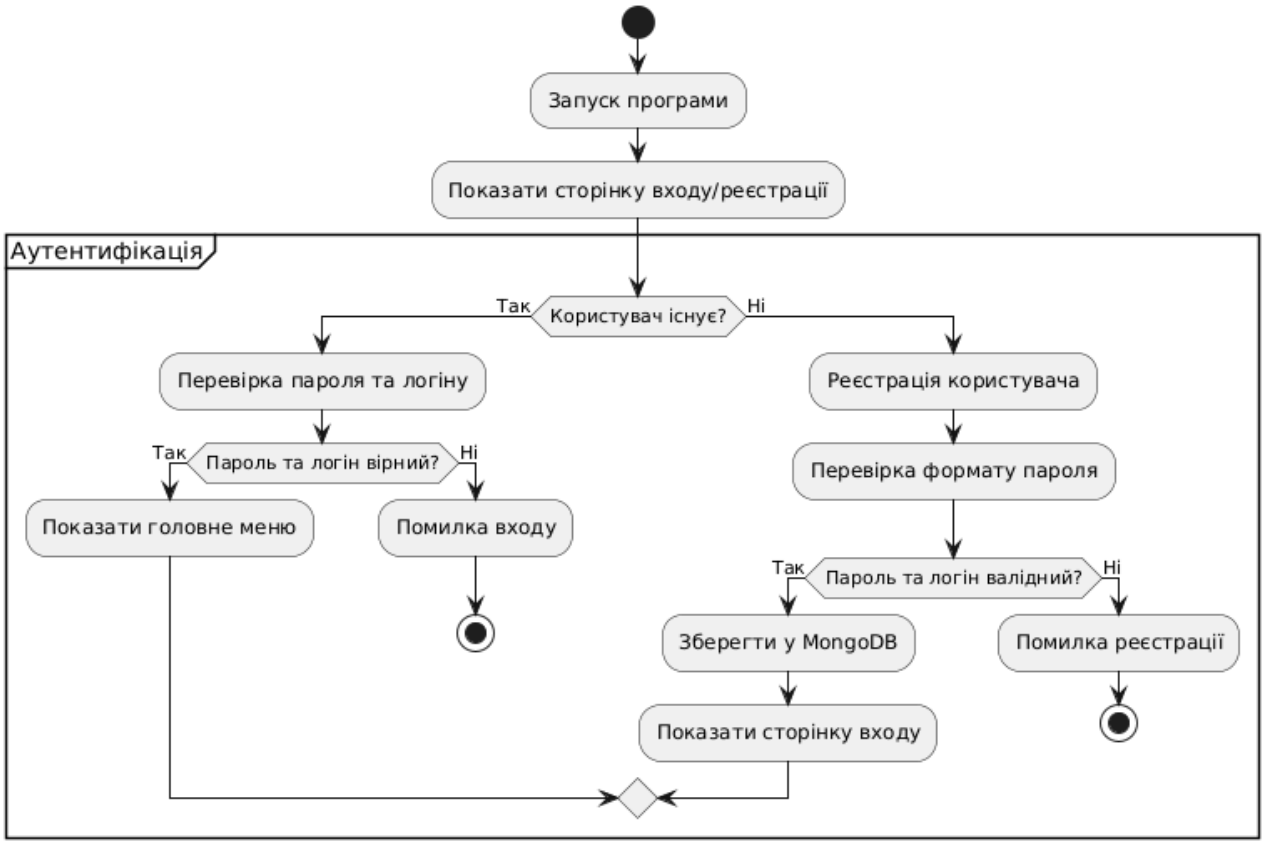


Рис 2.2 Діаграма діяльності авторизації користувача

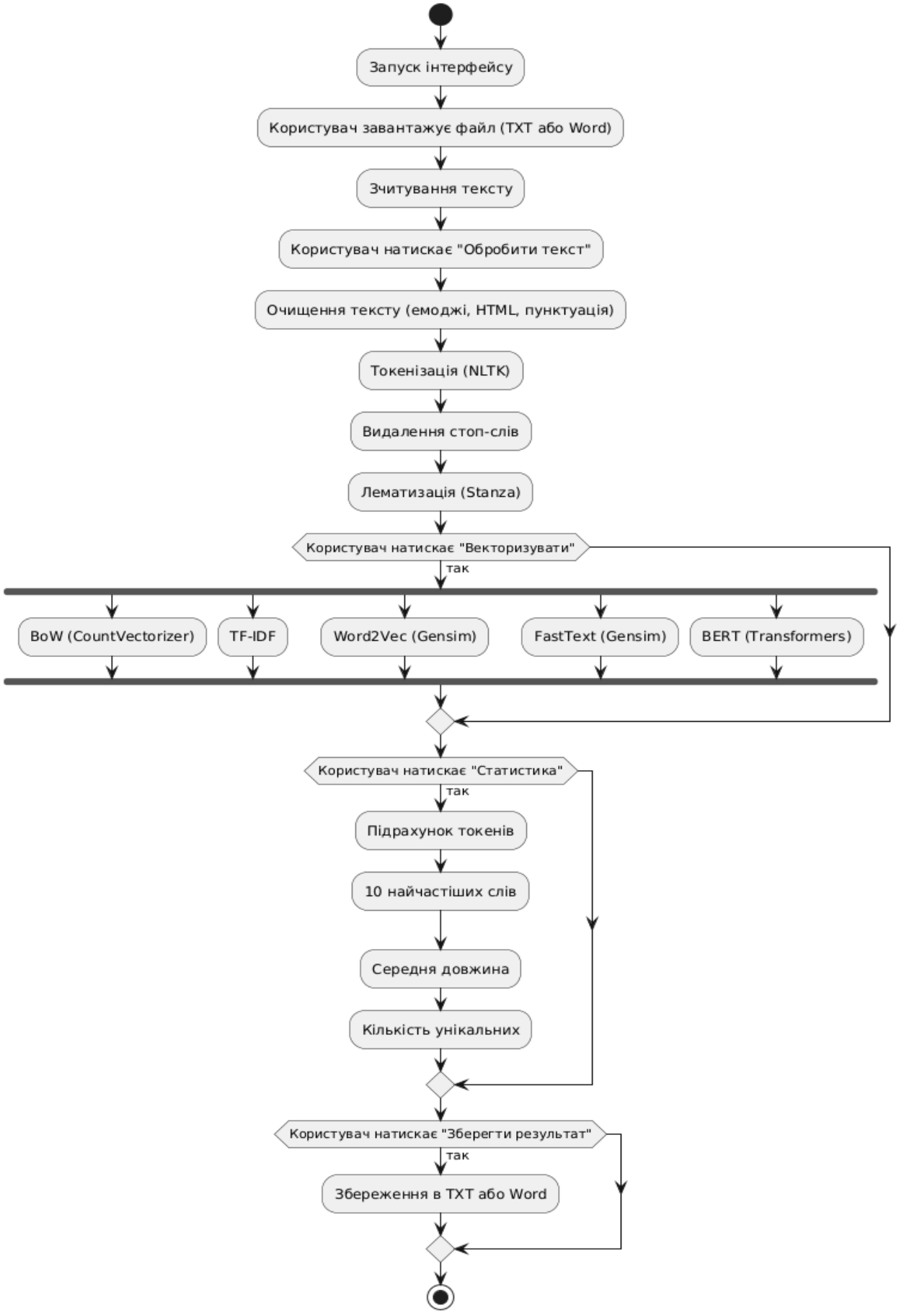


Рис. 2.3 Діаграма діяльності обробки тексту

Розглянемо детальніше діаграму діяльності обробки тексту. Початком дії стає перехід користувача до елемента обробки тексту. Першою дією стає завантаження текстового файлу який автоматично відобразиться у текстовому полі. Наступним кроком користувача стає “обробка тексту” що запускає дії очищення тексту від емоджі, HTML, пунктуацію, видаляє стоп слова, токенізує текст та після лематизації відображає результат. Наступним кроком користувач має можливість векторизувати текст та відобразити результати. Додаткова можливість користувача є створення статистики токенів та зберігання очищеного тексту.

2.5 Діаграма класів

Створення діаграми класів надає можливість провести структурне представлення, відображення зв'язків класів та перевірку дизайну. Діаграма класів містить наступні переваги для проектування системи:

Структурне представлення архітектури та ілюструє класи, їхні атрибути та зв'язки між класами. Це сприяє кращому розумінню вимог програми.

Діаграма класів дозволяє моделювати різні типи зв'язків класів, що надає представлення залежності та взаємодію класів у системі.

Діаграму класів можна використовувати для створення програмного скелету коду. Це прискорює процес реалізації програми та забезпечує узгодженість проекту з реалізацією.

Діаграма класів слугує інструментом для перевірки вимог до програми. Візуалізувавши структури можна виявити недоліки у дизайні, зайві або відсутні функції.

На рисунку 3.6 відображається діаграма класів функції обробки тексту.

Розглянемо її детальніше:

Починаючи зверху діаграми класів продемонстрований клас MainWindow який відображає клас вікна функції обробки тексту. Він виражений окремим вікном з власним стилем та посиланням на інші класи. Класи на які посилається

головне вікно наступні: FileHalder, Statistics, Text Processor та Vectorizer. Розглянемо кожен клас окремо:

FileHalder це клас задачею якого є імпорт та експорт файлів типу Word та txt. збереження файлів відбувається після обробки тексту.

Statistics - клас для створення статистики токенів тексту який буде відображати кількість токенів, середню довжину токена та підраховувати кількість кожного токена окремо для створення списку 10 найчастіших слів у тексті.

Text Processor - клас який містить обробку тексту, видаляючи емоджі, HTML, цифри, лематизовувати текст та токенізовувати його. Токенізація зберігає інформацію яку потім буде використовувати клас Statistics.

Vectorizer - клас задача якого векторизувати текст та відобразити результати векторизації користувачу. Текст векторизується тільки після проведення обробки.

Підводячи підсумок діаграма класів це незамінний інструмент для візуального відображення класів та зв'язків між класами.

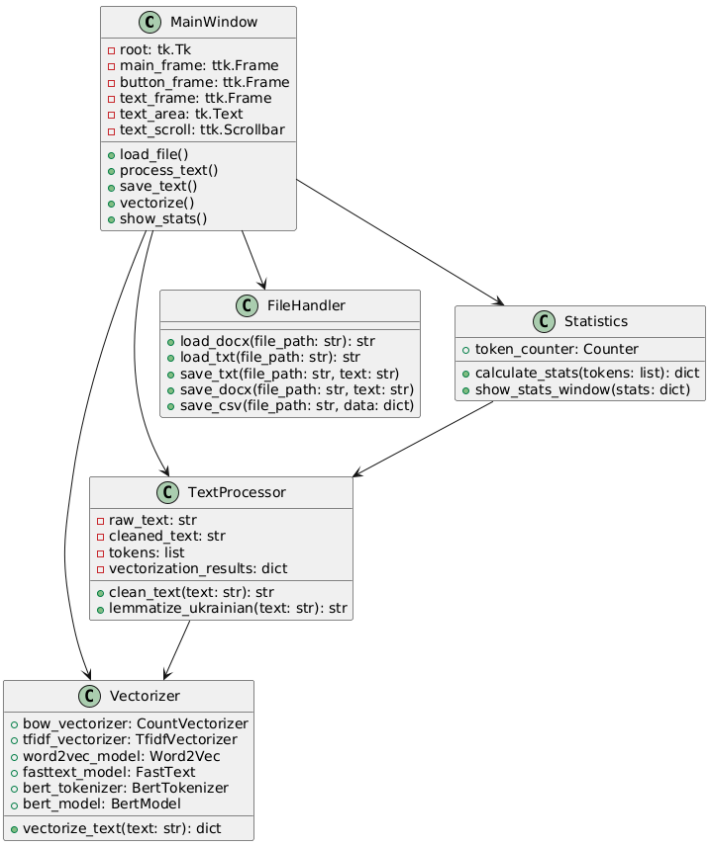


Рис 2.4. Діаграма класів обробки тексту

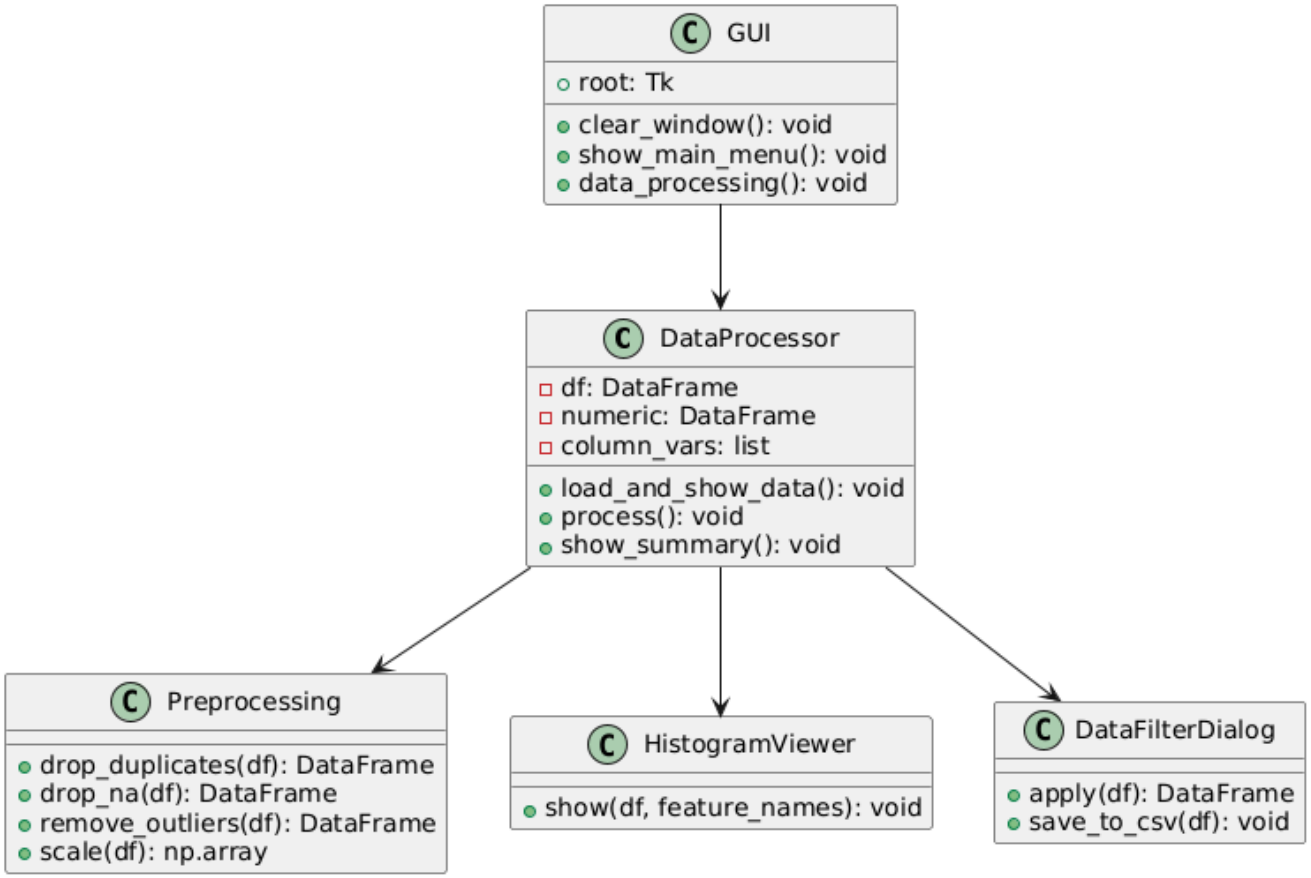


Рис. 2.5 Діаграма класів обробки числових таблиць

2.6 Діаграма послідовності

Для моделювання динаміки взаємодії між об’єктами в рамках сценарію. Діаграма послідовності відображає порядок обміну між компонентами програми. Структуру діаграми послідовності представляється наступним чином: актори та суб’єкти, життєва лінія, активність, умовні блоки та повідомлення. Детальніше опишем елементи діаграми послідовності:

Актори та суб’єкти відображають користувача та систему у верхній частині діаграми горизонтальному виді.

Життєва лінія зображується вертикальною лінією яка починається на акторі та суб’єктах. На життєвій лінії відображають передачу даних або викликів методів.

Активність зображується вертикальним прямокутником на життєвій лінії що відображає час, протягом якого об’єкт виконує операцію.

Умовні блоки зображений у вигляді вертикального прямокутника на життєвій лінії та відображає повторення або альтернативні шляхи взаємодії.

Повідомлення зображаються стрілками поєднуючи життєві лінії та вказуючи дію яку виконує програма після взаємодії з актором.

Для зчитування файлу у діаграмі створено 3 актора та суб’єкта: користувач, візуальна частина системи та частина системи обробки. Щоб розпочати взаємодію користувач повинен натиснути кнопку “завантажити файл” та система відкриє меню локільного вибору файлу. Коли користувач вибере файл, система відправить таблицю на наступну обробку: зчитування таблиці та заповнення її у пам’яті за допомогою масивів, очищення тексту від дублікатів, пропусків та шуму. Після цих дій частина обробки повертає таблицю та відображається користувачу. Створена діаграма послідовності зображена на рисунку 2.6.

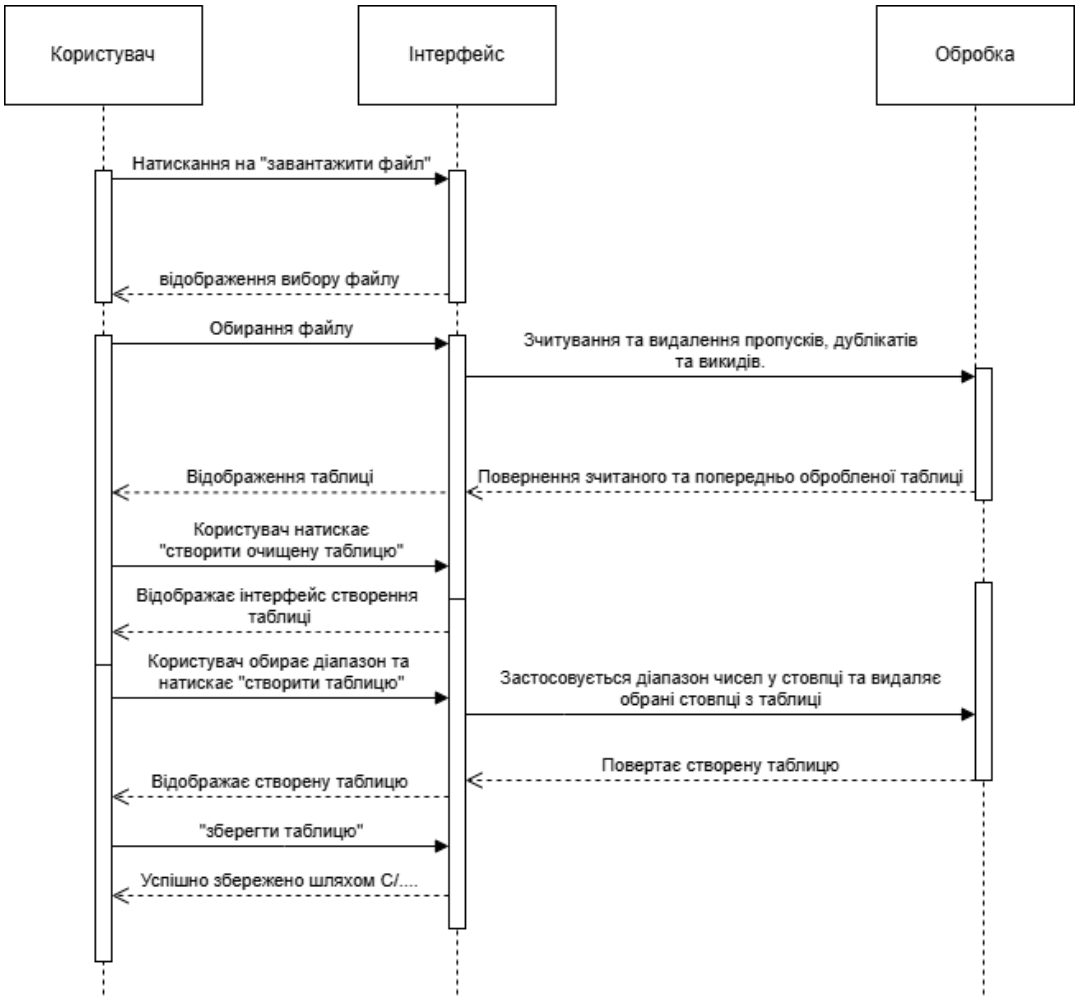


Рис. 2.6 Діаграма послідовності обробки класифікації, кластеризації та регресії

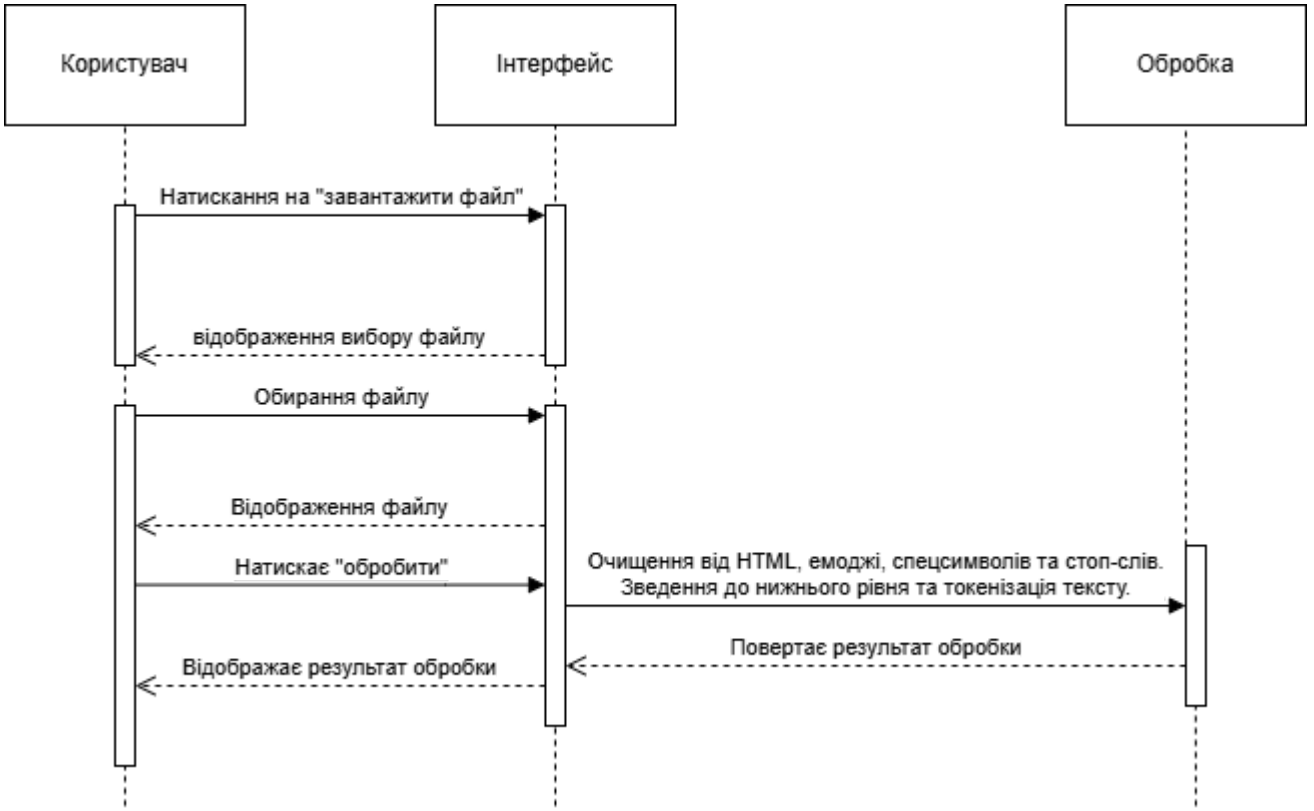


Рис. 2.7 Діаграма послідовності обробки текстових файлів

2.7 Проектування архітектури застосунку

Загальна модель застосунку розбита на 3 частини: користувацька частина GUI, програмна частина та збереження логіну та паролів користувачів на базі даних типу NoSQL MongoDB. Застосування модульного підходу до розробки застосунку дозволить легко оновлювати, виправляти та масштабувати модулі системи без впливу на інші частини системи що спростить розвиток системи у майбутньому.

2.7.1 Архітектура користуватської частини частини

Структура візуалізації Tkinter організована у три основні частини: Логін та реєстрація, меню та обробка даних. Цей підхід забезпечує чітке розділення відповідальностей між різними частинами програми, що сприяє кращій організації коду та полегшує його розуміння, тестування та обслуговування.

Компоненти йдуть один за одним, неможливо з логін сторінки автоматично

перейти до обробки даних. Вони включають логіку управління даними користувацького інтерфейсу та шаблон, який визначає, як виглядає інтерфейс. Кожен компонент крім загальних налаштувань (такі як розширення вікна), має власні кнопки, їхнє розташування та функціонал.

Сторінки складаються з груп компонентів та формують окремі відображення, які користувач бачить у себе на екрані. Наприклад, сторінка пошуку може включати компоненти для пошукової стрічки, фільтрів та результатів пошуку. Сторінки дозволяють агрегувати функціонал, необхідний для певного користувацького досвіду, і спрощують навігацію та взаємодію в рамках додатку.

Сервіси можуть включати взаємодію з сервером, обробку даних та інші повторно використовувані функції. Вони вводяться у компоненти через механізм впровадження залежностей, що забезпечує низьку зв'язність.

Така організація проекту на Tkinter сприяє модульності та повторному використанню коду. Розділення логіки додатку на компоненти, сторінки та сервіси допомагає утримувати код організованим та чистим, а також полегшує розширення та обслуговування додатку. Використання компонентно орієнтованого підходу дозволяє розробникам легко змінювати або оновлювати окремі частини інтерфейсу без впливу на інші, що є важливим для великих та складних додатків. Діаграма пакетів застосунку зображено на рисунку 2.8.

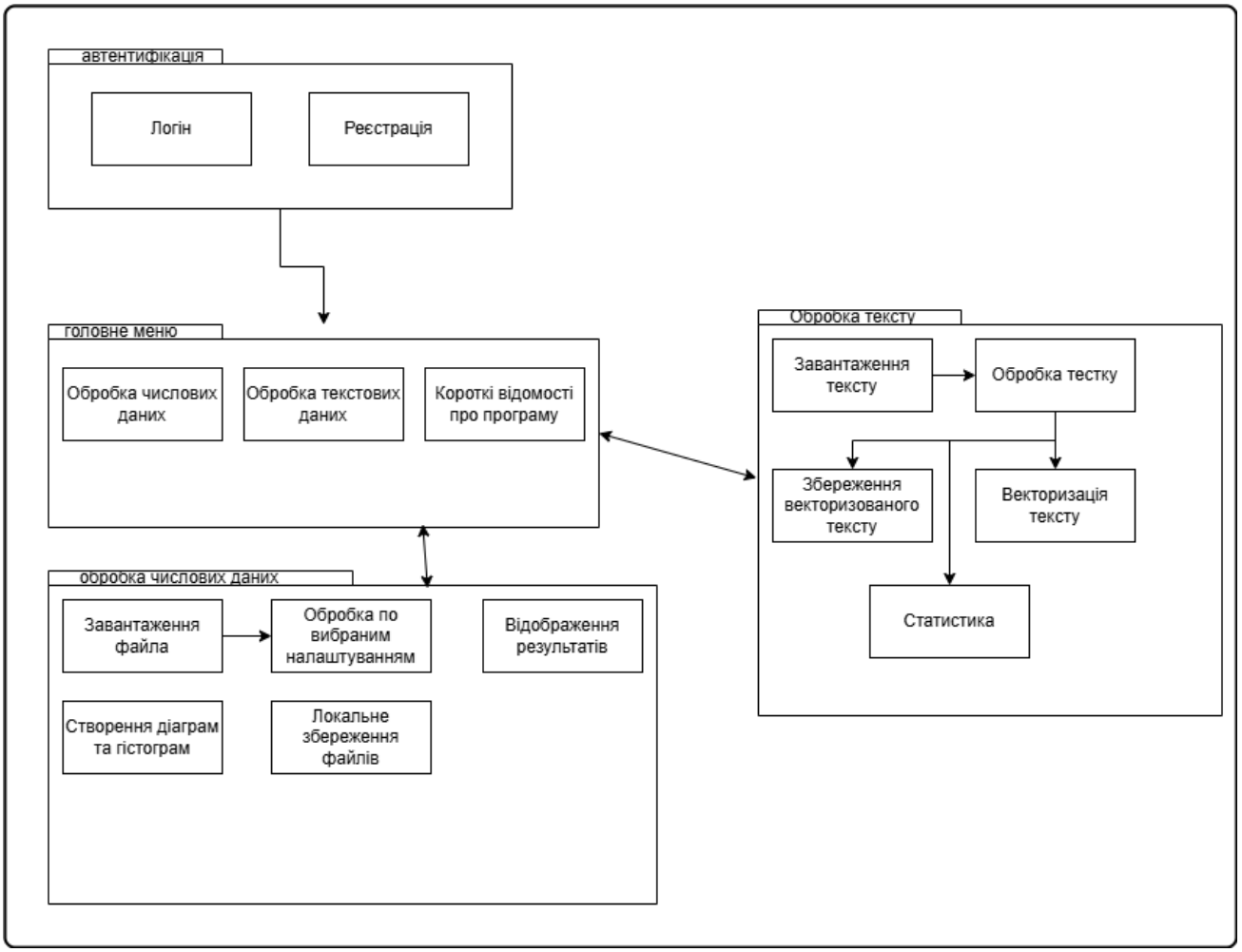


Рис. 2.8 Діаграма пакетів візуальної частини застосунку

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ДОДАТКУ ДЛЯ ОБРОБКИ СИРИХ ДАНИХ

3.1 Дизайн інтерфейсу

Tkinter - це інструмент для створення інтерфейсу користувача методом конвертування звернень Python в звернення Tcl. Для початку потрібно імпортувати Tk у наш проект наступною командою:

– import tkinter as tk

Після цього можна повноцінно створювати та налаштовувати вікно користувача. Програма містить загальні налаштування, такі як розмір вікна, кольорова гама, а також і окремі налаштування розміщення кнопок та їхній функціонал.

1.Меню авторизації

Авторизація містить наступні елементи: 2 текстових поля для логіну та пароля, кнопку яка змінює схований пароль на видимий та кнопок входу та реєстрацією. Після ініціалізації необхідних кнопок та полей, потрібно виставити місце розташування кожного елемента. На рисунку 3.1 та 3.2 зображено інтерфейс логіну та реєстрації в застосунку

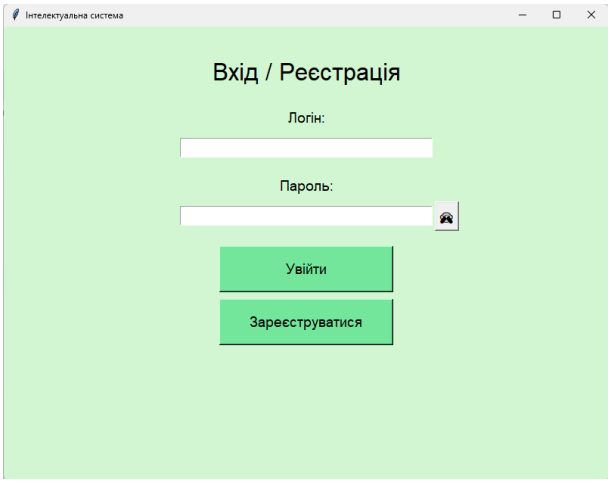


Рис. 3.1. Інтерфейс логіну

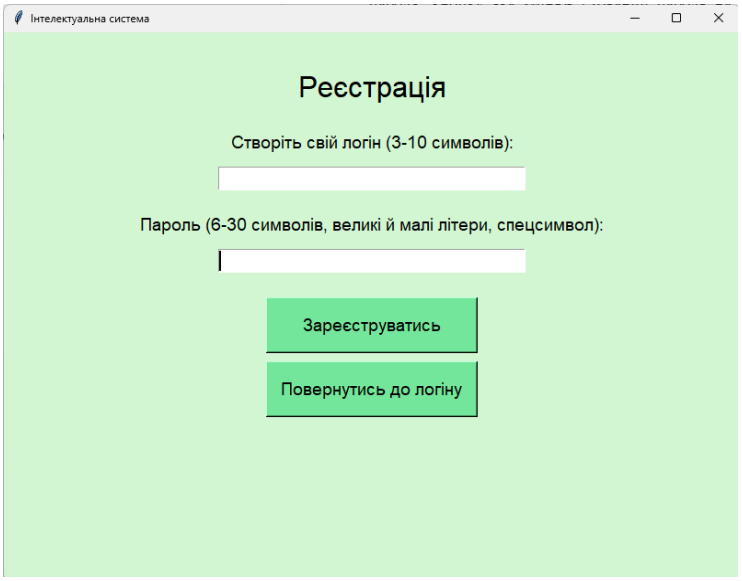


Рис. 3.2. Інтерфейс реєстрації

2. Головне меню

Програма містить невелике головне меню, яке реалізує перехід до обробки числових таблиць та тексту, та для виходу з програми. Важливим елементом буде створення текстового поля та кнопки “короткі відомості” щоб повідомити як працює програма. На рисунку 3.3 зображено інтерфейс головного меню

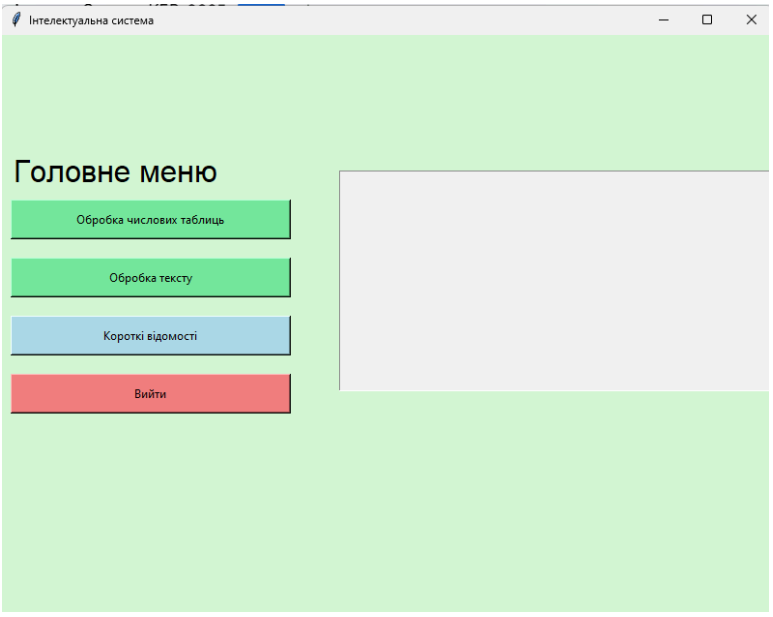


Рис. 3.3. Інтерфейс головного меню

3. Обробка числових таблиць

Для цього пункту необхідно створити 6 сталих кнопок, 1 кнопку яка з’являється коли імпортований файл та чекбокси які будуть залежати від кількості стовпців з числами. Було створено наступні 6 постійних кнопок: “Повернутись до меню”, “ Завантажити та переглянути дані”, “Показати діаграму”, “Показати гістограму”, “Показати результати” та “Створити очищену таблицю”. Створена 1 кнопка яка з'являється це “Запустити обробку”. Кнопки “Показати гістограму”, “Показати діаграму” та “Створити очищену таблицю” створюють нове вікно. Вікно створення очищеної таблиці має динамічний розмір, і залежить від кількості стовпців у таблиці. На рисунку 3.4 зображено інтерфейс обробки числових таблиць. На рисунку 3.5 зображено інтерфейс створення таблиці по діапазону. На рис. 3.6 зображено інтерфейс відображення діаграми

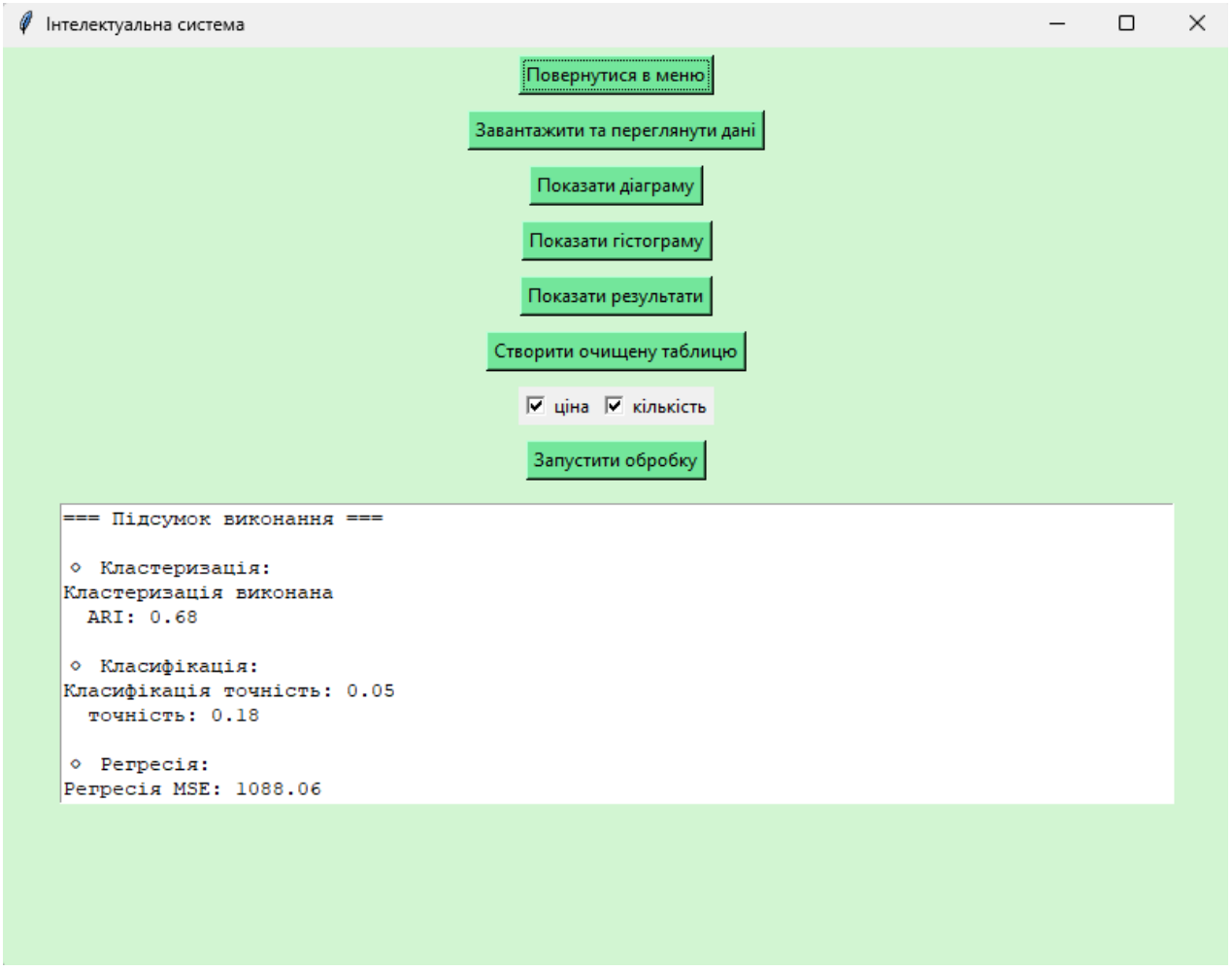


Рис. 3.4. Інтерфейс обробки числових таблицю

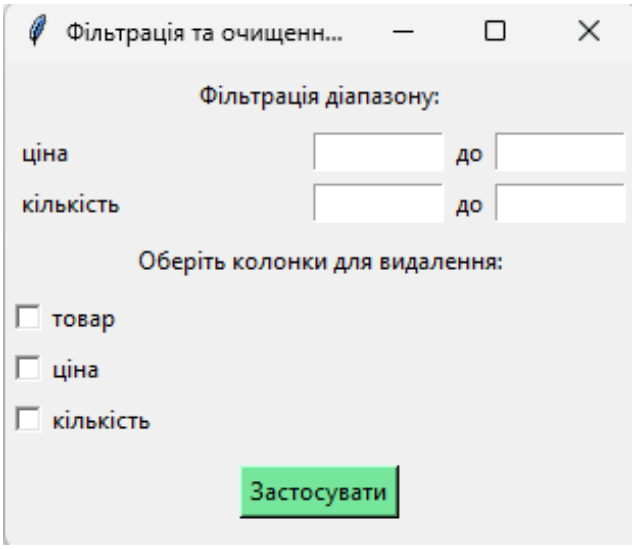


Рис. 3.5. Інтерфейс створення очищеної таблиці

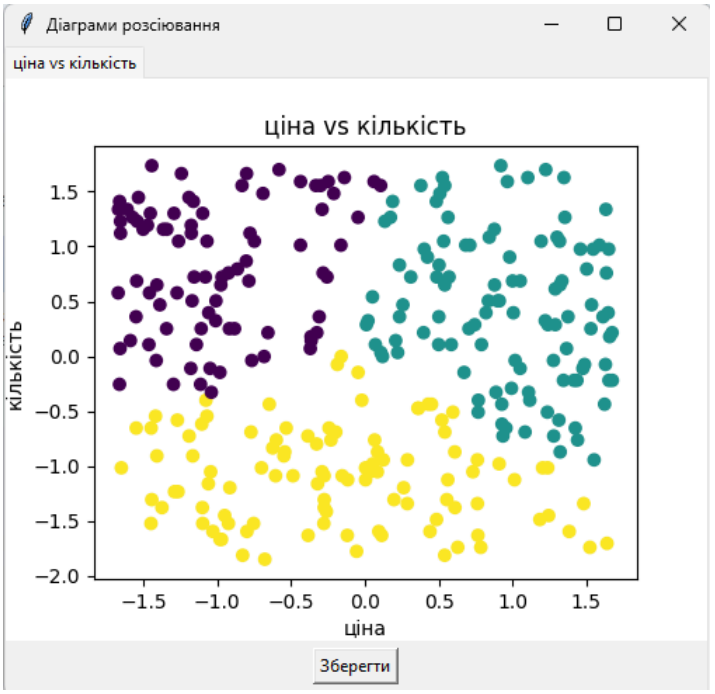


Рис. 3.6. Інтерфейс відображення діаграм

4. Інтерфейс обробки тексту.

Інтерфейс реалізований зі створенням нового вікна, яке має 5 кнопок та текстове поле. Всі кнопки виконують свою функцію та текстове поле забезпечує відображення тексту у поточному стані.

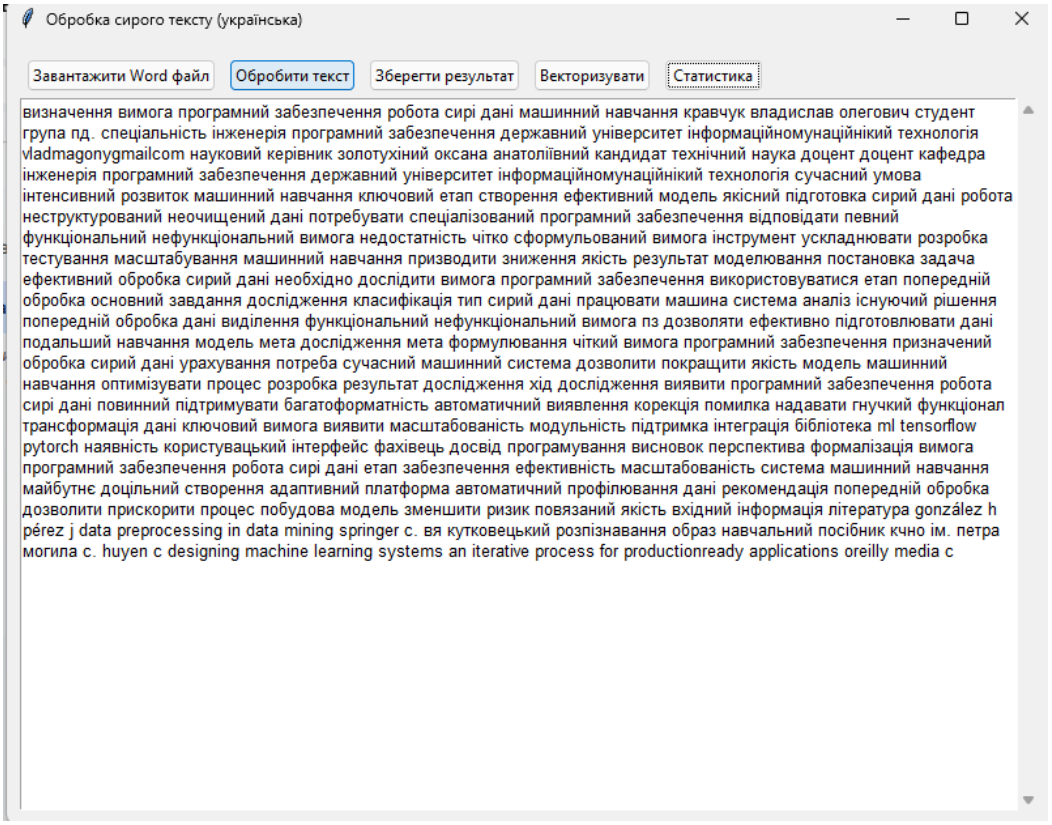


Рис. 3.7. Інтерфейс обробки тексту

3.2 Реалізація обробки числових таблиць

Для реалізації наших функцій нам потрібно наступні бібліотеки: Matplotlib, Pytorch та pandas. тепер ми поступово розглянемо кожну бібліотеку, та її роль у додатку. Pandas - головна задача якого є взаємодія з таблицями, та зчитування таблиці відбувається наступним фрагментом коду:

```
 try:
df = pd.read_csv(path, delimiter=",")
except Exception as e:
messagebox.showerror("Помилка", f"Не вдалося прочитати CSV: {e}")
return
```

Перед тим як показати початкову таблицю користувачу вона проходить через видалення пропусків, дублікатів та шуму. Очищення файлу від дублікатів було реалізовано наступним чином

df.drop_duplicates(inplace=True)

Завдяки бібліотеці Pandas файл автоматично очищається від дублюючих позицій і вносить зміни у завантажений файл. Перевірка видаляє тільки ідентичні рядки, що не призводить до проблем якщо рядки даних різняться 1 значенням з 10 колонок. Очищення від пропусків теж відбувається завдяки рядку

df.dropna(inplace=True)

Після цих дій файл перестає містити дублюючі позиції та видаляє пропуски у таблиці. Після цих етапів та видаленню шуму, користувачу відображають таблицю та дають можливість застосувати класифікацію, кластеризацію та регресію. Не виконуючи ці дії можна створити очищену таблицю по певному діапазону чисел у стовпці та видалити непотрібні стовпці. Вона має динамічний розмір який залежить від загальної кількості стовпців у таблиці та кількості числових стовпців. Після застосування фільтрів діапазону та видаленню стовпців, користувачу відображають нову таблицю та дають можливість зберегти її на комп’ютер. Завдяки наступному фрагменту коду створюється динамічна кількість чек кнопок (Checkbutton) для застосування фільтру:

```
for col in df.columns:
var = tk.IntVar()
cb = tk.Checkbutton(filter_window, text=col, variable=var)
 cb.pack(anchor='w')
 delete_vars.append((col, var))
```

Запустивши наступну обробку файлу, можна буде створити діаграму та гістограму. Обробка застосовує моделі класифікації, кластеризації та регресії. Після завершення обробки результат відображається у автоматично створене текстове поле. При спробі створенню діаграми та гістограми без обробки видає помилку, оскільки необхідні дані відсутні. Для реалізації моделей було використано бібліотеку Pytorch. Це відкрита бібліотека яка спеціалізується на глибокому навчанні моделей, та розроблена Facebook AI. Після використання всіх моделей до таблиці, у текстове поле відображаються результати виділяючи відповідь різних моделей через пропуск рядку.

Після цих дій можна створити діаграму та гістограму. Для створення гістограми необхідно лише один стовпець, коли для діаграми початковий мінімум 2 стовпця. Гістограма відображає загальну кількість значень у невеликому діапазоні. Вікно гістограми містить вкладку на кожну гістограму, та залежить від кількості оброблених стовпців. Наступний фрагмент коду відображає принцип збереження гістограми на комп’ютер користувача:

```
def save_hist(fig=fig, name=name):
file_path = filedialog.asksaveasfilename(
defaultextension=".png",
initialfile=f"{name}.png",
filetypes=[("PNG", "*.png")]
 )
if file_path:
fig.savefig(file_path)
messagebox.showinfo("Збережено", f"Гістограму збережено:\n{file_path}")
```

Створення діаграми розсіювання містить такий же інтерфейс, та відображає кожну діаграму окремо. Результати для діаграми беруться з результатів кластеризації. наступний фрагмент коду відображає реалізацію відображення діаграми за допомогою бібліотеки Matplotlib:

```
frame = tk.Frame(notebook)
notebook.add(frame, text=f"{features[i]} vs {features[i + 1]}")
fig, ax = plt.subplots(figsize=(5, 4))
ax.scatter(X[:, i], X[:, i + 1], c=labels, cmap='viridis')
ax.set_title(f"{features[i]} vs {features[i + 1]}")
ax.set_xlabel(features[i])
ax.set_ylabel(features[i + 1])
canvas = FigureCanvasTkAgg(fig, master=frame)
canvas.draw()
canvas.get_tk_widget().pack()
```

3.3 Реалізація обробки тексту

1. Першим кроком реалізуємо зчитування Word файлу та його відображення у текстовому полі. Для цього ми створимо функцію load_file(), яка буде зчитувати файл та відобразить його у текстове поле. Допоміжним інструментом стане бібліотека python-docx, яка використовується для роботи з Word файлами.

2. Наступним кроком стане реалізація обробки тексту. Кнопка “Обробити текст” запускає наступні дії з текстом: очищення від емоджі, HTML, цифри, лематизує текст, зводить до нижнього рівня всі елементи та токенізує його. Кожний етап обробки використовує унікальну бібліотеку.

```
def clean_text(text):
    text = text.lower()
    text = BeautifulSoup(text, "html.parser").get_text()
    text = emoji.replace_emoji(text, replace='')
    text = re.sub(r'\d+', '', text)
    text = re.sub(r'\n', ' ', text)
    text = re.sub(r'[^\w\sa-яєіїґA-ЯЄІЇҐ]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return lemmatize_ukrainian(text)
```

Фрагмент коду відображає реалізацію очистку тексту. Допоміжними інструментами стають наступні бібліотеки: Emoji для видалення емоджі з тексту, bs4 яке видалятиме HTML з тексту, re яке буде виконувати роботу з регулярними виразами та для лематизації бібліотека stanza.

Бібліотеку stanza містить необхідні дані для лематизації україномовних текстів.. Для цього потрібно у код ввести наступне

```
import stanza
stanza.download('uk')
```

Завдяки 2 рядкам коду ми імпортуєм бібліотеку stanza та встановлюємо українську модель та можемо реалізовувати аналіз тексту. Кінцевим елементом лематизації буде токенізування тексту яке знадобиться в подальшій векторизації

тексту та статистики токенів.

Статистика токенів відображає середню довжину токена, загальну кількість токенів, кількість унікальних токенів та 10 найчастіших слів у тексті. Структура функції статистики проста, Загальний підрахунок кількості токенів, підрахунок унікальних токенів, вираховування середньої довжини токена та відображення слів, токени яких зустрічаються найчастіше.

num_tokens = len(tokens)

num_unique_tokens = len(set(tokens))

token_counts = Counter(tokens)

most_common = token_counts.most_common(10)

Тут відображено такі статистики як кількість токенів, кількість унікальних токенів та 10 найчастіше вживаних токенів. Щоб правильно відобразити результат користувачу необхідно створити звіт де кожен результат буде відображений. На рисунку 3.8 зображено створення звітності токена

```
report = f"""
Статистика тексту:
---------------------------
Загальна кількість токенів: {num_tokens}
Унікальних токенів: {num_unique_tokens}
Середня довжина токена: {np.mean([len(t) for t in tokens]):.2f} символів

10 найчастіших слів:
---------------------------
"""
for word, count in most_common:
    report += f"{word}: {count} разів\n"

stats_text.insert(tk.END, report)
stats_text.config(state=tk.DISABLED)
```

Рис. 3.8. Код реалізації звітності

Застосувавши цей код створюється звіт у завчасно підготовлене вікно та відображається користувачу.

Результуючий текст можна зберегти та векторизувати та побачити

результати векторизації. Для векторизації тексту було використано наступні методи: BoW, TF-IDF, Word2Vec та FastText. BoW - це метод векторизації тексту який рахує частоту кожного слова та у результаті створює матрицю частот. TF-IDF це показник який використовується для оцінки важливості слова у контексті документа. Word2Vec створює модель машинного навчання та навчає модель на основі контексту слів. Натренована модель має змогу виявляти синоніми у тексті. Створює вектор до кожного слова, потім бере середній вектор тексту. FastText подібний до Word2Vec, але опрацьовує частини слів, що надає змогу краще розуміти моделі нові або рідкісні слова. Після цих дій результат відображається окремим вікном та можна переглядати кожний метод окремо.

3.4 Завантаження проекту на GitHub

Для збереження та контролю версій проекту використовується GitHub. Для імпорту на власний репозиторій, необхідно виконати кілька кроків. Відкриваючи Git bash у кореневому файлі проекту. Далі через команду

-git init

потрібно провести ініціалізацію на свій акаунт GitHub. Успішно авторизовавшись потрібно вставити URL посилання на репозиторій куди будем імпортувати проект.

- https://github.com/KODICHOK/-

Далі добавляєм необхідні файли та створюємо коміт повідомлення яке описує внесені зміни у проєкт. Фінальним кроком є завантаження змін на репозиторій.

3.5 Тестування застосунку

Протестуєм частину обробки числових таблиць. Перевірка буде виконуватись на правильність моделей, правильність зчитування файлу та попередньої обробки, створення діаграм та гістограм з подальшим зберіганням.

Для тестування було створено файл з допущеними дублями та пропусками.

Очікується результат який при завантаженні видалить дублікати та пропуски.

Очікування підтвердились, пропуски дублі та шуми видалені.

Перейдем до наступного етапу тестування, а саме чи коректно працює обробка даних, оберемо колонку для обробки “ціна”. Після виконання обробки, користувачу відобразились результати обробки по моделям і є можливість створенні гістограми яка має наступний вигляд. На рисунку 3.9 - 3.11 зображенні тестові дані використані для аналізу, результат зчитування та попередньо обробки таблиці та створенна гістограма

товар	ціна	кількість
Сік	68.68	68
Борошно	267.19	28
Рис	486.92	80
Чай	390.23	40
Масло	120.45	59
Масло	270.46	53
Рис	366.84	38
Сік	68.68	68
Рис	486.92	80
Чай	390.23	40
Масло	120.45	59
Масло	270.46	53
Рис	366.84	38
чайок	163.41	29
Яблука	212.27	23
Яйця	10.39	68
Ковбаса	221.06	31
Кава	429.15	81
Вода	369.43	4
Кава	333.81	33
Масло	441.63	32
Яблука	100.01	63
Олія	65.68	88
Сіль	47.39	37

Рис. 3.9 фрагмент таблиці тестових даних

Борошно	267.19	28.0
Рис	486.92	80.0
Чай	390.23	40.0
Масло	120.45	59.0
Масло	270.46	53.0
Рис	366.84	38.0
чайок	163.41	29.0
Яблука	212.27	23.0
Яйця	10.39	68.0
Ковбаса	221.06	31.0

Рис. 3.10 Результат зчитування даних

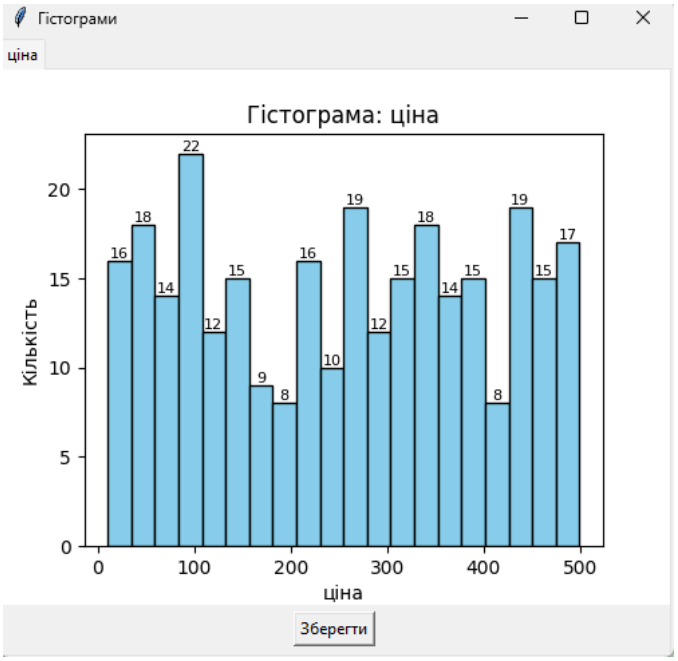


Рис. 3.11 Тестова гістограма

При спробі створити діаграму розсіювання видає помилку про недостачу ознак для створення. Запустивши обробки 2 колонок очікуємо що користувач створить після цього діаграму.

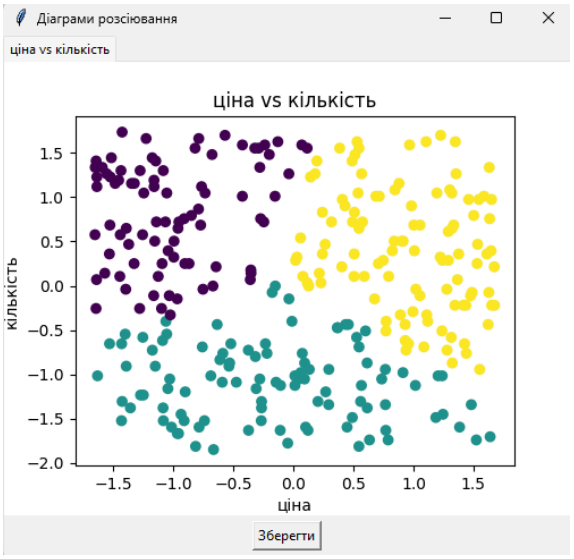


Рис 3.12 Діаграма розсіювання

На рисунку 3.12 зображено створена діаграма. Протестуєм створення очищеної таблиці по діапазону, взявши декілька варіантів створення: діапазон у якому числа відсутні, змінимо мінімальне значення з максимальним та видалення всіх колонок. Обравши відсутні числа у колонці, застосунок видає повідомлення що створена таблиця не містить даних. Результати будуть зображені на рисунках 3.13 та 3.14.

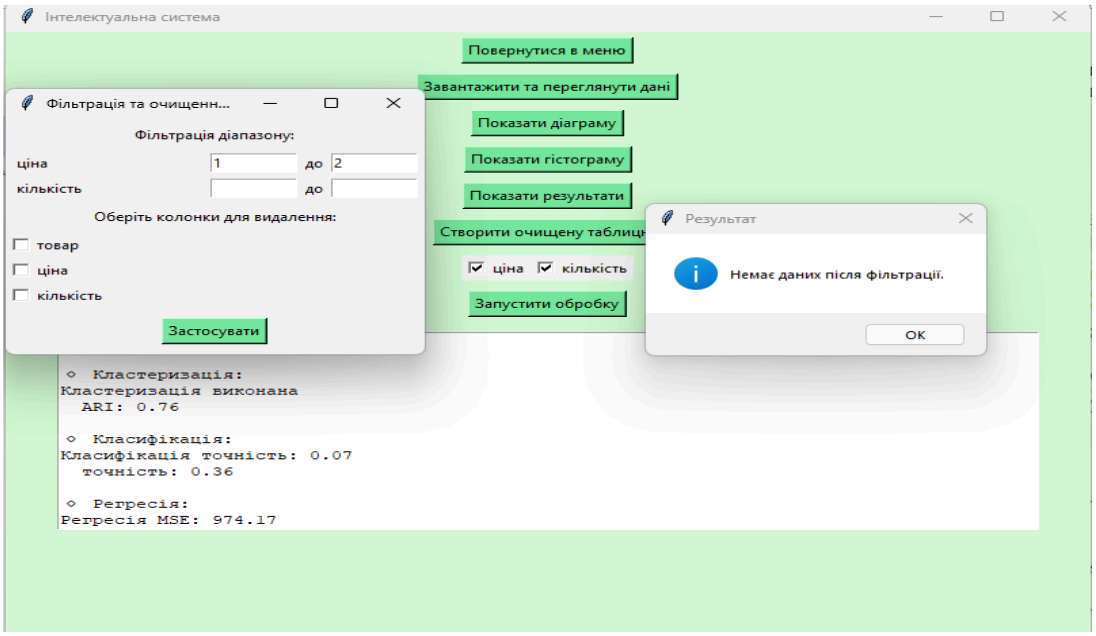


Рис.3.13 Створення по неіснуючим числам у діапазоні

Наступним тестом буде створення діапазону від більшого числа до меншого. Тестовий діапазон містить від 200 до 100.У результаті застосунок визначив мінімальне та максимальне значення і відобразив таблицю за діапазоном. Та фінальний тест буде видалення всіх колонок що в очікуванні видасть повідомлення про відсутність даних. Результат підтвердився.

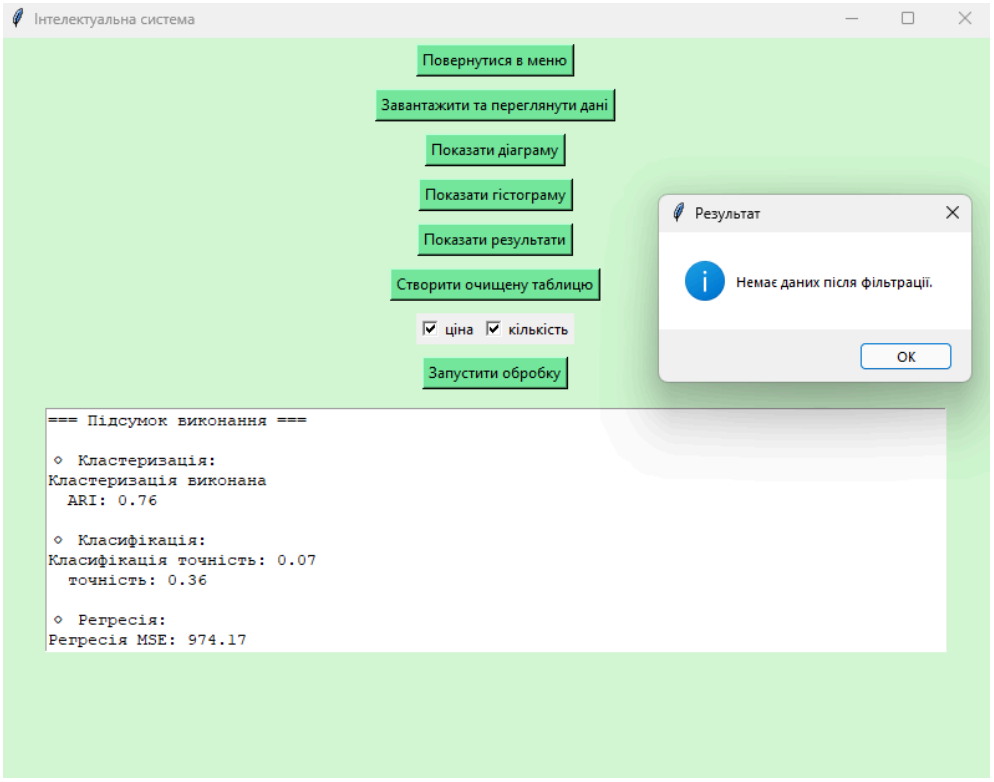


Рис. 3.14 Тест на видалення всіх колонок.

Тепер необхідно протестувати обробку тексту. Тестовими даними для обробки стане моя апробована теза для конференції. Перевіримо роботу статистики, обробки тексту, векторизації та збереження тексту. Очікуваним результатом першого тесту має відображатись очищений текст. Тест виконався успішно, текс очищений, зведений до нижнього рівня та лематизований. Лише після спроби векторизації та статистики ми Дізнаємось чи були створені токени слів.

Текст успішно можна векторизувати, отже обробка тексту повністю зроблена, після завершення векторизації було відображено загальні результати векторизації та є можливість переглянути перші елементи векторів кожного метода. При генеруванні статистики токенів, успішно відображається повна статистика загальної кількості токенів, кількості унікальних токенів, середньої довжини токена та 10 найчастіших слів. На рисунку 3.15 та 3.16 зображено текстові дані до та після обробки.

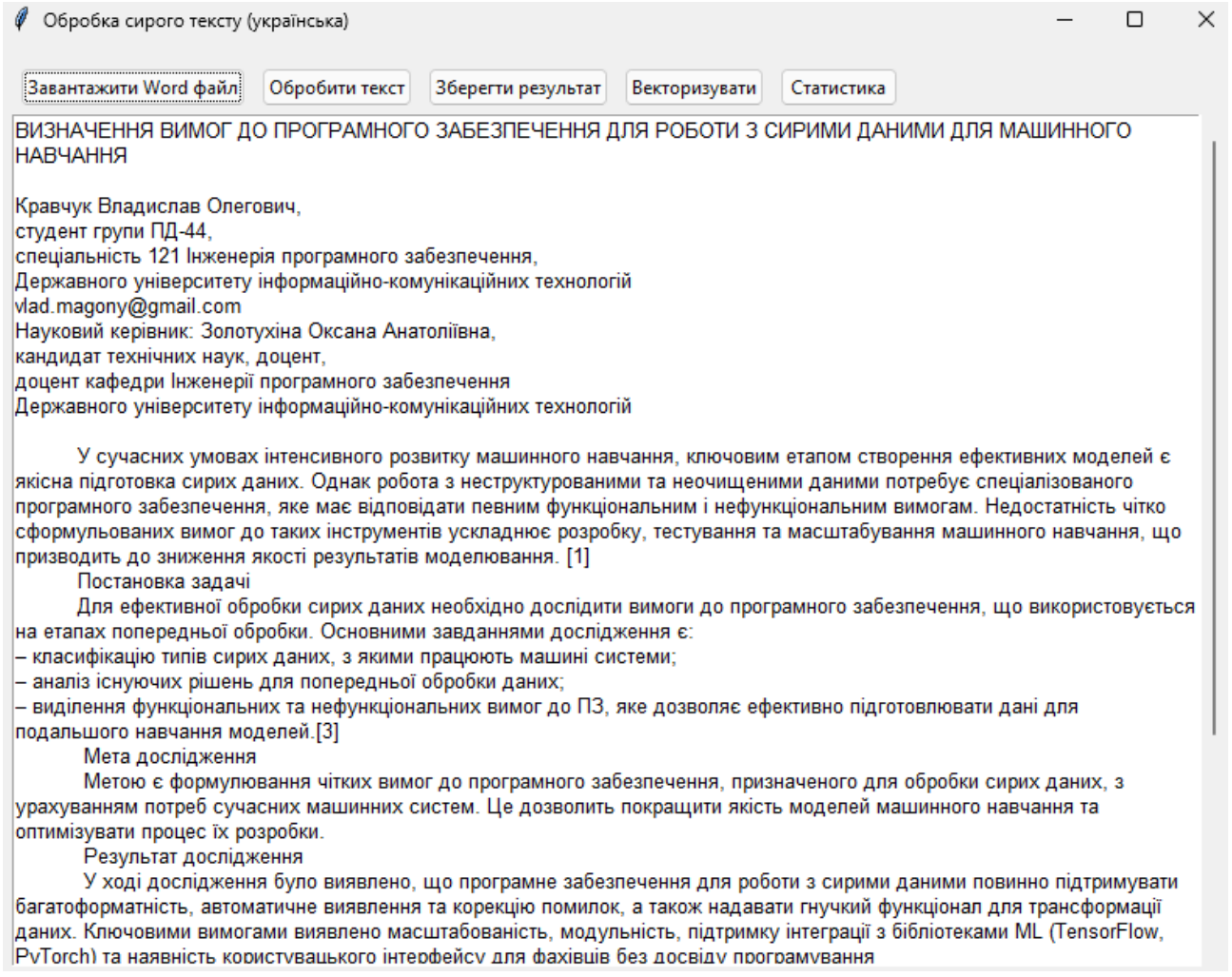


Рис. 3.15 Тестові дані

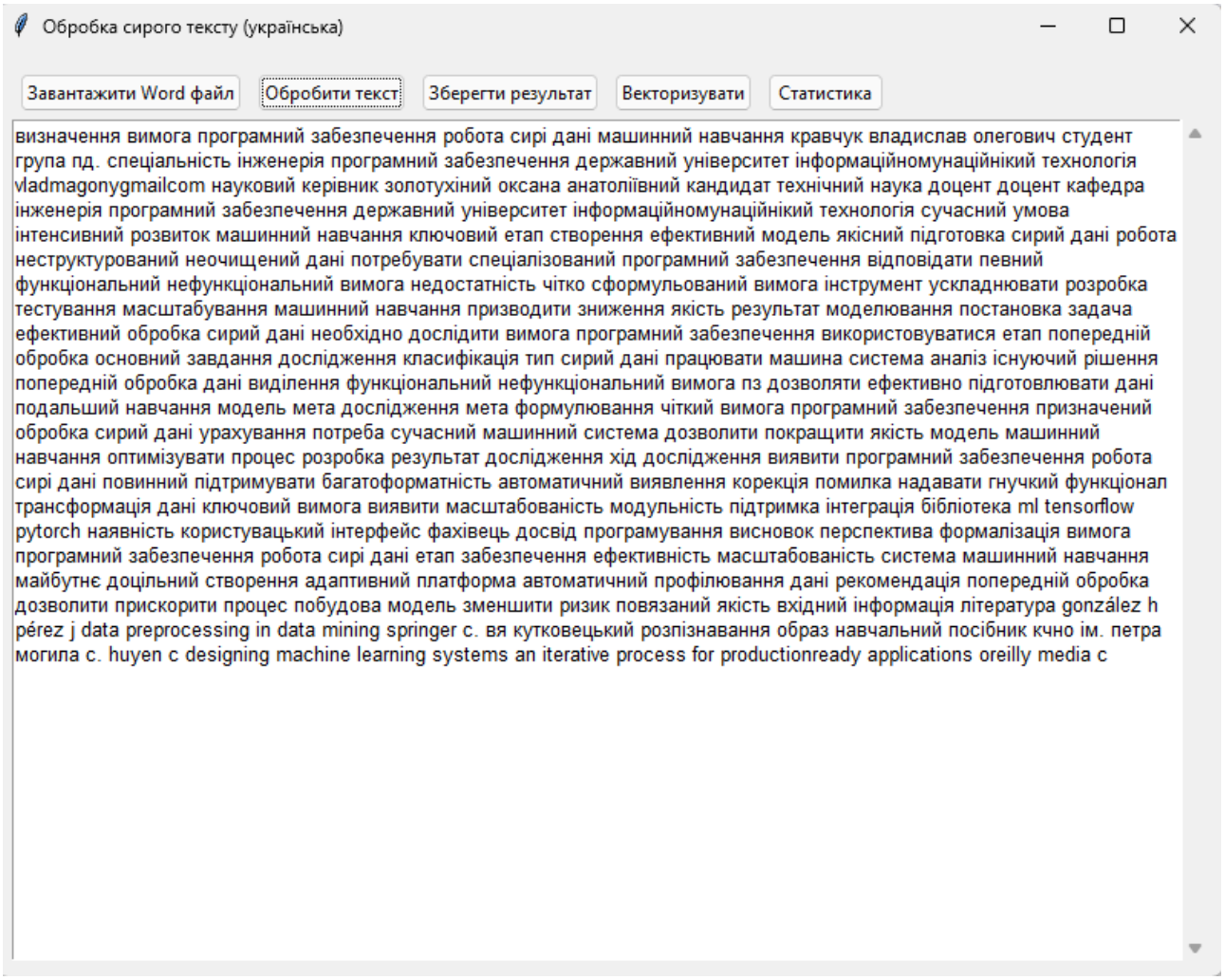


Рис. 3.16 Результат очищення тестових даних

Після текстової обробки текст звівся до нижнього рівня, була проведена лематизація та видалені стоп слова. Після завершення обробки текст успішно можна зберегти, векторизувати та переглянути статистику токенів у тексті.

ВИСНОВКИ

Результатами виконаної роботи став застосунок обробки сирих даних для машинного навчання на мові Python .

Було виконано наступні задачі:

1. Проаналізовано предметну область: визначено актуальність та особливості розробки додатку Windows; Було успішно проаналізовано наступні програми аналоги: Tableau, Microsoft POWER BY, Apache Hadoop. На основі аналізу аналогів було визначено функціональні та нефункціональні вимоги, а також враховано особливості роботи з сирими даними для машинного навчання, зокрема необхідність попереднього очищення, нормалізації, обробки пропусків, викидів та підготовки текстових і числових ознак для подальшої класифікації, кластеризації та регресії.

2. Обрано засоби реалізації: PyCharm як IDE, Python як мову програмування, інструмент Tkinter для створення інтерфейсу користувача, PyTorch та Scikit-learn для реалізації моделей машинного навчання, Matplotlib для створення діаграми та гістограми результатів обробки числових таблиць, Gensim для обробки текстових файлів, MongoDB як базу даних та GitHub як система контролю версій.

3. Виконано проектування системи обробки сирих даних з використанням діаграми варіантів використання, пакетів, діяльності, класів та послідовності.

4. Реалізовано систему, використовуючи обрані технології: зроблений користуватський інтерфейс через Tkinter, зроблені моделі за допомогою PyTorch та Scikit-learn, реалізована візуалізація через діаграми та гістограми, реалізовано обробка текстових файлів, завантажено проект на GitHub та протестовано моделі на коректність роботи.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Кравчук В.О., Золотухіна О.А, Визначення вимог до програмного забезпечення для роботи з сирими даними для машинного навчання // Матеріали Vl Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологій». 24.04.2025, Київ, ДУІКТ, Збірник тез. К.:ДУІКТ, 2025. С. 342-343.

2. Кравчук В.О., Золотухіна О.А, Бібліотека Matplotlib для візуалізації оброблених даних // Матеріали V Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». 15.05.2025, Київ, ДУІКТ (подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (3rd Edition). США, 2022. 1136 с.

2. Huyen, C. Designing Machine Learning Systems: An Iterative Process for Production-Ready Applications. США, 2022. 339 с.

3. Luciano Ramalho. Fluent Python: Clear, Concise, and Effective Programming (2nd Edition). США, 2022. 1022 с.

4. Raschka, S., Mirjalili, V. Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python. Велика Британія, 2022. 724 с.

5. Brownlee, J. Machine Learning Mastery With Python (Updated Edition). Австралія, 2021. 409 с.

6. Hall, M. Data Science Fundamentals for Python and MongoDB. СШАа, 2021. 350 с.

7. Harris, C. Building Machine Learning Powered Applications: Going from Idea to Product. США, 2020. 274 с.

8. Nelli, F. Python for Data Science for Dummies. США, 2022. 464 с.

9. Zaninotto, F. Essential Modern JavaScript for Beginners. Франція, 2021. 278 с.

10. Tiwari, A. Machine Learning for Absolute Beginners: A Plain English Introduction. Індія, 2021. 178 с.

11. Moroney, L. AI and Machine Learning for Coders: A Programmer's Guide to Artificial Intelligence. США, 2020. 390 с.

12. Javed, K. Practical Data Preprocessing. Велика Британія, 2021. 295 с.

13. Shaffer, C.A. Data Structures and Algorithms in Python. США, 2021. 624 с.

14. Valliappan, R. Data Wrangling with Python. Велика Британія, 2020. 350 с.

15. Kiran M., Surajit M., Bhushankumar N. Data pre-processing and data

augmentation techniques - ScienceDirect. Global Transitions Proceedings. 2022. No.3. P. 91–99.

16. Parth Sojitra. The Crucial Role of Data Preprocessing in Machine Learning. [Online Article] 18.10.2023 (дата звернення: 26.04.2025).

17. Документація tkinter. URL: https://docs.python.org/3/library/tkinter.html

18. Стадії циклу розробки ПЗ. URL: https://qalight.ua/baza-znaniy/stadiyi-tsiklu-rozrobki-pz/

19. Діаграма послідовності (UML). URL: https://duikt.edu.ua/ua/news-1-626-7897-zastosuvannya-uml-chastina-2-diagrama-poslidovnosti---sequence-diagram_kafedra-kompyuternih-nauk-ta-informaciynih-tehnologiy

20. Power BI – URL: https://www.microsoft.com/en-us/power-platform/products/power-bi

21. Tableau – URL: https://www.tableau.com/

22. Apache Hadoop – URL: https://hadoop.apache.org/

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – спрощення процесу обробки та підготовки сирих даних для машинного навчання.

Об’єкт дослідження – процеси обробки та підготовки сирих даних для машинного навчання.

Предмет дослідження – методи, інструменти та технології програмної реалізації обробки сирих даних.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз існуючих застосунків
2. Провести аналіз інструментів для роботи з сирими даними.
3. Сформулювати функціональні та нефункціональні вимоги.
4. Розробити архітектуру застосунку обробки сирих даних.
5. Реалізувати застосунок для обробки сирих даних.
6. Провести тестування застосунку.

3

АНАЛІЗ АНАЛОГІВ

Показник	Tableau	Microsoft Power BI	Apache Hadoop	Adaptation
Видалення дублікатів числових даних	-	+	+	+
Нормалізація та стандартизація числових даних	+	+	+	+
Створення набору чисел в заданому діапазоні	+	+	-	+
Видалення записів таблиці даних з пропущеними значеннями	-	+	+	+
Переведення тексту в нижній регістр	+	+	+	+
Очищення тексту від HTML, емоджі, спецсимволів та стоп-слів	-	+	-	+
Нормалізація слів тексту	-	-	лематизація, стемінг	Лематизація
Статистичні показники тексту	Довжина тексту	Довжина тексту	Довжина тексту, Кількість слів, Частота слів, тд	Загальна кількість токенів та унікальних токенів, середня довжина токена, ТОП-10 слів
Підтримка обробки текстів українською мовою	-	-	+	+
Візуалізація результатів обробки	+	+	-	+
Векторизація тексту	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Функції для обробки текстових даних: переведення тексту в нижній регістр, видалення HTML, емоджі, спецсимволів та стоп-слів, нормалізація слів тексту, векторизація тексту, Розрахунок загальної кількості токена та кількості унікальних токенів, розрахунок середньої довжини токена та 10 найуживаніших слів у тексті.
2. Функції для обробки числових даних: стандартизація значень, видалення дублікатів, видалення записів з пропущеними значеннями, візуалізація результатів обробки в графічному вигляді, створення набору чисел в заданому діапазоні.
3. Імпорт сирих даних.
4. Експорт оброблених даних.
5. Аутентифікація користувача.

Нефункціональні вимоги:

1. Графічне представлення даних повинно бути у вигляді гістограми та діаграми розсіювання.
2. Імпорт даних для обробки числових таблиць повинен здійснюватись у форматі CSV.
3. Імпорт даних для обробки тексту повинен здійснюватись у форматі Word або txt.
4. Експорт результатів обробки числових таблиць має бути у формат CSV.
5. Експорт результатів обробки текстових даних має бути у формат Word або txt.
6. При обробці текстових даних повинна бути підтримка української мови.
7. Аутентифікація користувачів повинна бути реалізована зі збереженням логіну та паролю на MongoDB, що надасть можливість масштабувати у майбутньому збереження результатів обробки на хмарі.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

NumPy

GENSIM
topic modelling for humans

python

pandas

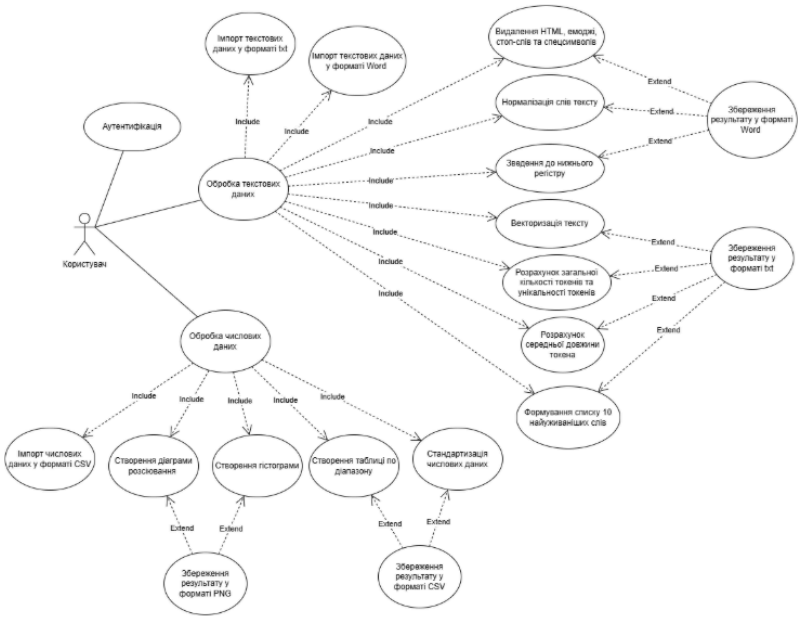
matplotlib

PyTorch

Natural Language Toolkit
(NLTK)

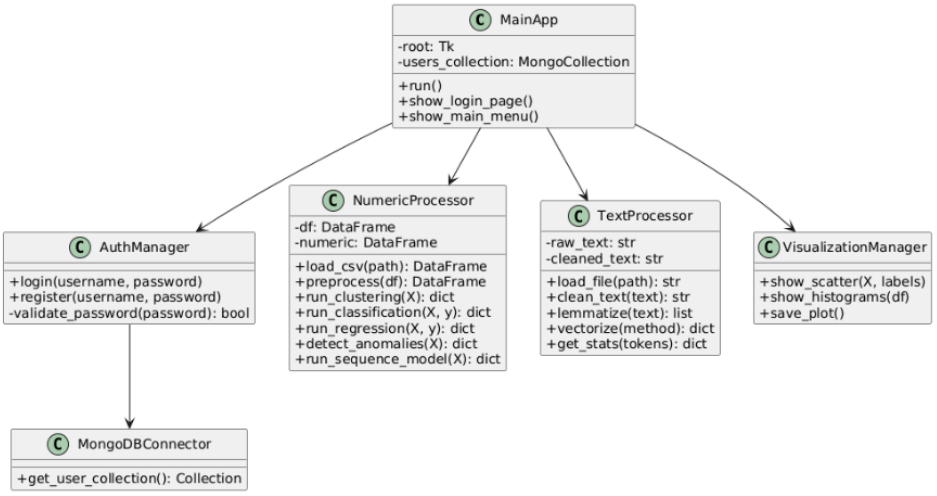
6

Діаграма варіантів використання



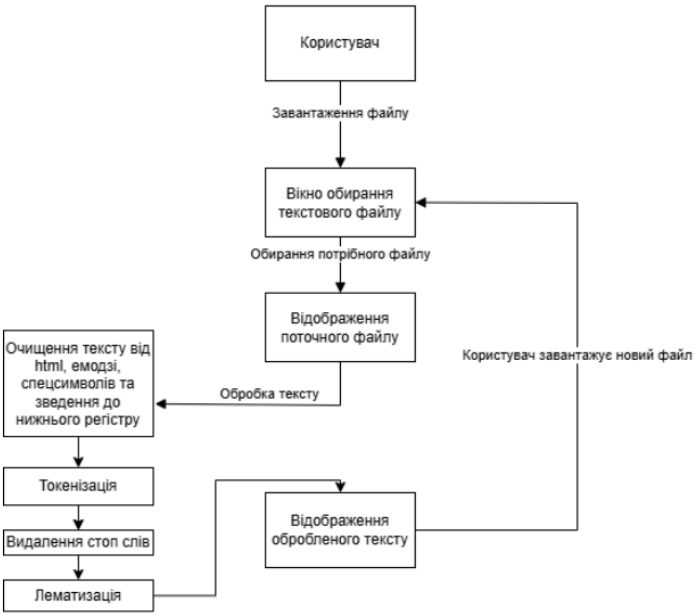
7

Діаграма класів



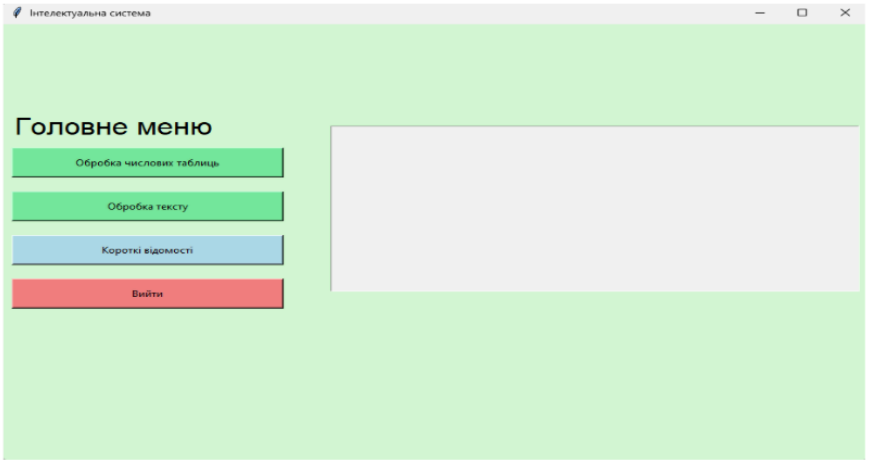
8

Механізм обробки тексту



9

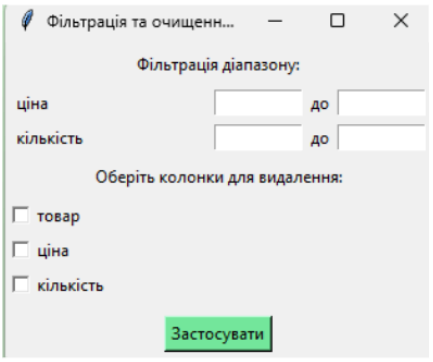
ЕКРАННІ ФОРМИ



Головне меню

10

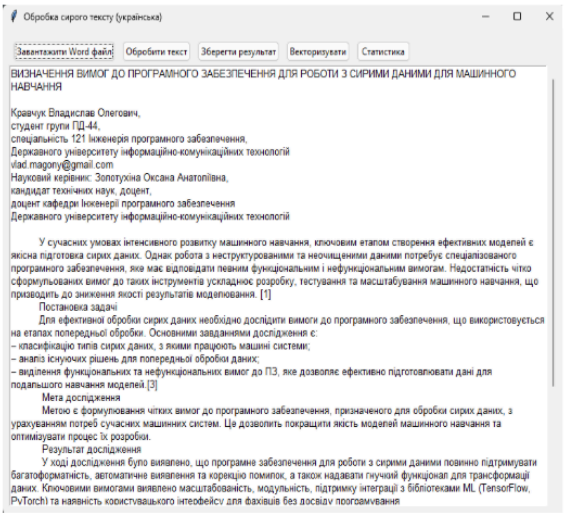
ЕКРАННІ ФОРМИ



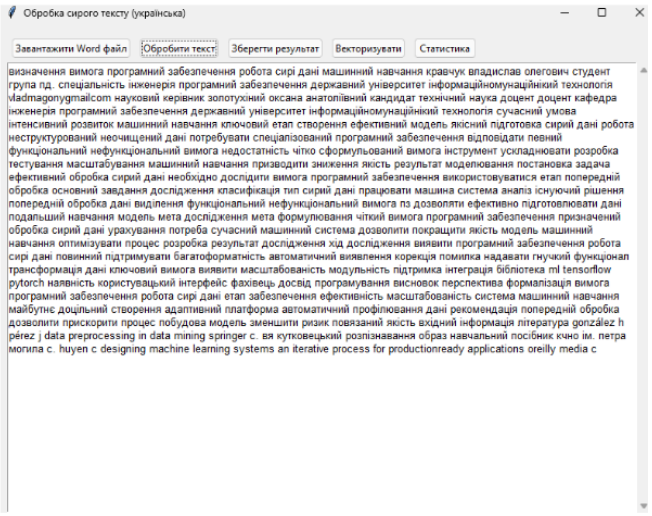
Створення таблиці по діапазону

11

ЕКРАННІ ФОРМИ



Текст до обробки



Текст після обробки

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Кравчук В.О., Золотухіна О.А. Визначення вимог до програмного забезпечення для роботи з сирими даними для машинного навчання: Матеріали всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2025, м. Київ. К.:ДУІКТ, С. 368-369
2. Кравчук В.О., Золотухіна О.А. Бібліотека Matplotlib для візуалізації оброблених даних: Матеріали V Всеукраїнської Науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2025, м. Київ. К.:ДУІКТ, (подано до друку)

13

ВИСНОВКИ

1. Проаналізовано аналоги з метою встановлення функціональних та нефункціональних вимог.
2. Проаналізовані інструменти для обробки сирих даних, на основі чого було обрано мову програмування та бібліотеки для реалізації.
3. Було сформульовано функціональні та нефункціональні вимоги за допомогою яких була створена архітектура застосунку.
4. Розроблена архітектура застосунку обробки сирих даних, для створення створення застосунку.
5. Реалізований застосунок обробки даних, з повною реалізацією функціональних та нефункціональних вимог.
6. Успішно проведені тести застосунку обробки даних, забезпечено спрощення процесу обробки даних.

14

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНИХ МОДУЛІВ

```
def show_login_page():
    clear_window()
    root.update_idletasks()
    window_width = root.winfo_width()
    center_x = window_width // 2
    tk.Label(root, text="Вхід / Реєстрація", font=("Helvetica",
24), bg="#d2f8d2").place(x=center_x, y=60, anchor="center")
    tk.Label(root, text="Логін:", bg="#d2f8d2",
font=("Helvetica", 14)).place(x=center_x, y=120,
anchor="center")
    global entry_user, entry_pass, toggle_btn, password_visible
    entry_user = tk.Entry(root, font=("Helvetica", 14),
width=30)
    entry_user.place(x=center_x, y=160, anchor="center")
    tk.Label(root, text="Пароль:", bg="#d2f8d2",
font=("Helvetica", 14)).place(x=center_x, y=210,
anchor="center")
    entry_pass = tk.Entry(root, show="*", font=("Helvetica",
14), width=30)
    entry_pass.place(x=center_x, y=250, anchor="center")
    password_visible = False
    toggle_btn = tk.Button(root, text="🙈", font=("Helvetica",
14), command=toggle_password, width=2, height=1)
    toggle_btn.place(x=center_x + 170, y=250, anchor="w")
    tk.Button(root, text="Увійти", font=("Helvetica", 14),
width=20, height=2, bg="#75e69b",
command=login).place(x=center_x, y=320, anchor="center")
    tk.Button(root, text="Зареєструватися", font=("Helvetica",
14), width=20, height=2, bg="#75e69b",
command=show_register_page).place(x=center_x, y=390,
anchor="center")
def show_register_page():
    clear_window()
    root.update_idletasks()
    window_width = root.winfo_width()
    center_x = window_width // 2

    tk.Label(root, text="Реєстрація", font=("Helvetica", 24),
bg="#d2f8d2").place(x=center_x, y=60, anchor="center")
    tk.Label(root, text="Створіть свій логін (3-10 символів):",
bg="#d2f8d2", font=("Helvetica", 14)).place(x=center_x,
y=120, anchor="center")
    global reg_user, reg_pass, reg_toggle_btn,
reg_password_visible
    reg_user = tk.Entry(root, font=("Helvetica", 14), width=30)
    reg_user.place(x=center_x, y=160, anchor="center")
    tk.Label(root, text="Пароль (6-30 символів, великі й малі
літери, спецсимвол):", bg="#d2f8d2", font=("Helvetica",
14)).place(x=center_x, y=210, anchor="center")
    reg_pass = tk.Entry(root, show="*", font=("Helvetica", 14),
width=30)
    reg_pass.place(x=center_x, y=250, anchor="center")
    reg_password_visible = False
    tk.Button(root, text="Зареєструватись", font=("Helvetica",
14), width=20, height=2, bg="#75e69b",
command=register).place(x=center_x, y=320, anchor="center")
    tk.Button(root, text="Повернутись до логіну",
font=("Helvetica", 14), width=20, height=2, bg="#75e69b",
command=show_login_page).place(x=center_x, y=390,
anchor="center")
def login():
    username = entry_user.get()
    password = entry_pass.get()
    user = users_collection.find_one({"username": username,
"password": password})
    if user:
        show_main_menu()
    else:
        messagebox.showerror("Помилка", "Невірний логін або
пароль")

def register():
    username = reg_user.get()
    password = reg_pass.get()
    if not (3 <= len(username) <= 10):
        messagebox.showerror("Помилка", "Логін має містити
від 3 до 10 символів")
        return

    if not
re.match(r"^(?=.*[a-z])(?=.*[A-Z])(?=.*[\W_]).{6,30}$",
password):
        messagebox.showerror("Помилка", "Пароль має містити
від 6 до 30 символів, великі й малі літери та спецсимвол")
        return
    if users_collection.find_one({"username": username}):
        messagebox.showerror("Помилка", "Користувач з таким
логіном вже існує")
    else:
        users_collection.insert_one({"username": username,
"password": password})
        messagebox.showinfo("Успішно", "Реєстрація
завершена")
        show_login_page()
def data_processing():
    clear_window()
    global df, numeric, column_vars, checkbox_frame,
process_button
    result_box = None
    df = None
    numeric = None
    column_vars = None
    process_button = None

    clustering_result = {'X': None, 'labels': None, 'message': "",
'feature_names': []}
    classification_result = {'pred': None, 'n_classes': 0, 'message':
""}
    regression_result = {'y_pred': None, 'message': ""}
    anomaly_result = {'count': 0, 'message': ""}
    sequence_result = {'message': ""}

    column_vars = []

    def load_and_show_data():
        global df, numeric, column_vars, checkbox_frame,
process_button
        path = filedialog.askopenfilename(filetypes=[("CSV
файли", "*.csv")])
        if not path:
            return

        try:
            df = pd.read_csv(path, delimiter=",")
```

```
        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося
прочитати CSV: {e}")
            return

        df.drop_duplicates(inplace=True)
        df.dropna(inplace=True)

        for col in df.select_dtypes(include='number').columns:
            Q1 = df[col].quantile(0.25)
            Q3 = df[col].quantile(0.75)
            IQR = Q3 - Q1
            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR
            df = df[(df[col] >= lower_bound) & (df[col] <=
upper_bound)]

        numeric = df.select_dtypes(include='number')
        if numeric.empty:
            messagebox.showerror("Помилка", "Файл не містить
числових даних.")
            return
        preview_window = tk.Toplevel(root)
        preview_window.title("Попередній перегляд даних")
        text = tk.Text(preview_window, height=10, width=180)
        text.insert(tk.END, df.to_string(index=False))
        text.pack()

        if checkbox_frame is not None:
            for widget in checkbox_frame.winfo_children():
                widget.destroy()
        else:
            checkbox_frame = tk.Frame(root)
            checkbox_frame.pack(pady=5)
        column_vars.clear()
        for col in numeric.columns:
            var = tk.IntVar()
            tk.Checkbutton(checkbox_frame, text=col,
variable=var).pack(side=tk.LEFT)
            column_vars.append((col, var))
        if process_button is not None:
            process_button.destroy()
        process_button = tk.Button(root, text="Запустити
обробку", bg="#75e69b", command=process)
        process_button.pack(pady=5)
    def process():
        selected_cols = [col for col, var in column_vars if var.get()
== 1]
        if not selected_cols or df is None:
            messagebox.showerror("Помилка", "Оберіть хоча б
одну ознаку.")
            return
        selected_data = df[selected_cols]
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(selected_data)
        X_tensor = torch.tensor(X_scaled, dtype=torch.float32)
        clustering_result['feature_names'] = selected_cols
        try:
            class KMeansTorch:
                def __init__(self, n_clusters, n_iter=100):
                    self.n_clusters = n_clusters
                    self.n_iter = n_iter

                def fit_predict(self, X):
                    centroids =
X[torch.randperm(X.size(0))[:self.n_clusters]]
                    for _ in range(self.n_iter):
                        dist = torch.cdist(X, centroids)
                        labels = dist.argmin(dim=1)
                        for i in range(self.n_clusters):
                            if (labels == i).sum() > 0:
                                centroids[i] = X[labels == i].mean(dim=0)
                    return labels

            kmeans = KMeansTorch(n_clusters=3)
            labels = kmeans.fit_predict(X_tensor)
            clustering_result['X'] = X_tensor.numpy()
            clustering_result['torch_labels'] = labels.numpy()
            clustering_result['labels'] = labels.numpy()
            clustering_result['message'] = "Кластеризація
виконана"

            kmeans_sklearn = KMeans(n_clusters=3,
random_state=42)
            sklearn_labels =
kmeans_sklearn.fit_predict(X_tensor.numpy())
            clustering_result['sklearn_ari'] =
adjusted_rand_score(labels.numpy(), sklearn_labels)
        except Exception as e:
            clustering_result['message'] = f"Помилка
кластеризації: {str(e)}"
        try:
            y =
torch.tensor(df[df.columns[-1]].astype('category').cat.codes.val
ues, dtype=torch.long)
            X_train, X_test, y_train, y_test =
train_test_split(X_tensor, y, test_size=0.2)

            model = nn.Sequential(
                nn.Linear(X_tensor.shape[1], 16),
                nn.ReLU(),
                nn.Linear(16, len(y.unique()))
            )
            loss_fn = nn.CrossEntropyLoss()
            optimizer = optim.Adam(model.parameters(), lr=0.01)
            for _ in range(100):
                y_pred = model(X_train)
                loss = loss_fn(y_pred, y_train)
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
            pred = model(X_test).argmax(dim=1)
            acc = (pred == y_test).float().mean().item()
            classification_result['message'] = f"Класифікація
точність: {acc:.2f}"
            classification_result['torch_acc'] = acc
            clf = RandomForestClassifier()
            clf.fit(X_train.numpy(), y_train.numpy())
            y_pred_sklearn = clf.predict(X_test.numpy())
            acc_sklearn = accuracy_score(y_test.numpy(),
y_pred_sklearn)
            classification_result['sklearn_acc'] = acc_sklearn
        except Exception as e:
            classification_result['message'] = f"Помилка
класифікації: {str(e)}"
        try:
            y = torch.tensor(df[df.columns[-1]].values,
dtype=torch.float32).unsqueeze(1)
            X_train, X_test, y_train, y_test =
train_test_split(X_tensor, y, test_size=0.2)
            model = nn.Sequential(
                nn.Linear(X_tensor.shape[1], 16),
                nn.ReLU(),
                nn.Linear(16, 1)
            )
            loss_fn = nn.MSELoss()
            optimizer = optim.Adam(model.parameters(), lr=0.01)
            for _ in range(100):
                y_pred = model(X_train)
                loss = loss_fn(y_pred, y_train)
```

```
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        y_pred = model(X_test).detach().numpy()
        mse = mean_squared_error(y_test.numpy(), y_pred)
        regression_result['message'] = f"Регресія MSE:
{mse:.2f}"
        regression_result['torch_mse'] = mse
        reg = LinearRegression()
        reg.fit(X_train.numpy(), y_train.numpy())
        y_pred_sklearn = reg.predict(X_test.numpy())
        mse_sklearn = mean_squared_error(y_test.numpy(),
y_pred_sklearn)
        regression_result['sklearn_mse'] = mse_sklearn
    except Exception as e:
        regression_result['message'] = f"Помилка регресії:
{str(e)}"
    try:
        class Autoencoder(nn.Module):
            def __init__(self, input_dim):
                super().__init__()
                self.encoder = nn.Sequential(
                    nn.Linear(input_dim, 16),
                    nn.ReLU(),
                    nn.Linear(16, 8)
                )
                self.decoder = nn.Sequential(
                    nn.Linear(8, 16),
                    nn.ReLU(),
                    nn.Linear(16, input_dim)
                )
            def forward(self, x):
                encoded = self.encoder(x)
                decoded = self.decoder(encoded)
                return decoded
        model = Autoencoder(X_tensor.shape[1])
        loss_fn = nn.MSELoss()
        optimizer = optim.Adam(model.parameters(), lr=0.01)
        for _ in range(100):
            output = model(X_tensor)
            loss = loss_fn(output, X_tensor)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        with torch.no_grad():
            reconstructed = model(X_tensor)
            mse = torch.mean((X_tensor - reconstructed) ** 2,
dim=1)
            threshold = mse.mean() + 2 * mse.std()
            anomalies = mse > threshold
        anomaly_result['count'] = anomalies.sum().item()
        anomaly_result['message'] = f"Аномалії виявлено:
{anomalies.sum().item()} з {len(anomalies)} записів"
    except Exception as e:
        anomaly_result['message'] = f"Помилка виявлення
аномалій: {str(e)}"
    try:
        if X_tensor.shape[0] >= 10:
            sequence_length = 10
            X_seq = []
            for i in range(len(X_tensor) - sequence_length):
                X_seq.append(X_tensor[i:i + sequence_length])
            X_seq = torch.stack(X_seq)
            class LSTMNet(nn.Module):
                def __init__(self, input_size, hidden_size,
output_size):
                    super().__init__()
                    self.lstm = nn.LSTM(input_size, hidden_size,
batch_first=True)
                    self.fc = nn.Linear(hidden_size, output_size)
                def forward(self, x):
                    _, (h_n, _) = self.lstm(x)
                    out = self.fc(h_n[-1])
                    return out
            lstm_model = LSTMNet(X_tensor.shape[1], 32,
X_tensor.shape[1])
            loss_fn = nn.MSELoss()
            optimizer = optim.Adam(lstm_model.parameters(),
lr=0.01)
            for _ in range(50):
                output = lstm_model(X_seq)
                target = X_tensor[sequence_length:]
                loss = loss_fn(output, target)
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
            sequence_result['message'] = f"Обробка
послідовностей виконана. Loss: {loss.item():.4f}"
        else:
            sequence_result['message'] = "Недостатньо даних
для послідовностей"
    except Exception as e:
        sequence_result['message'] = f"Помилка обробки
послідовностей: {str(e)}"

    show_summary()
def show_summary():
    nonlocal result_box
    if result_box is None:
        result_box = tk.Text(root, height=12, width=90)
        result_box.pack(pady=10)
    result_box.delete("1.0", tk.END)
    result_box.insert(tk.END, "=== Підсумок виконання
===\n\n")
    if clustering_result.get('message'):
        result_box.insert(tk.END, " 🔹 Кластеризація:\n")
        result_box.insert(tk.END, clustering_result['message'] +
"\n")
        if 'sklearn_ari' in clustering_result:
            result_box.insert(tk.END, f"  ARI:
{clustering_result['sklearn_ari']:.2f}\n\n")
    if classification_result.get('message'):
        result_box.insert(tk.END, " 🔹 Класифікація:\n")
        result_box.insert(tk.END,
classification_result['message'] + "\n")
        if 'torch_acc' in classification_result and 'sklearn_acc' in
classification_result:
            result_box.insert(tk.END, f"  точність:
{classification_result['sklearn_acc']:.2f}\n\n")
    if regression_result.get('message'):
        result_box.insert(tk.END, " 🔹 Регресія:\n")
        result_box.insert(tk.END, regression_result['message']
+ "\n")
        if 'torch_mse' in regression_result and 'sklearn_mse' in
regression_result:
            result_box.insert(tk.END, f"   MSE:
{regression_result['sklearn_mse']:.2f}\n\n")
    if anomaly_result.get('message'):
        result_box.insert(tk.END, anomaly_result['message'] +
"\n\n")
    if sequence_result.get('message'):
        result_box.insert(tk.END, " 🔹 Обробка
послідовностей:\n")
        result_box.insert(tk.END, sequence_result['message'] +
"\n\n")
def show_scatter_plot():
    if clustering_result['X'] is None or
clustering_result['labels'] is None:
        messagebox.showwarning("Увага", "Спочатку
виконайте кластеризацію.")
```

```
            return
        X = clustering_result['X']
        labels = clustering_result['labels']
        features = clustering_result['feature_names']
        n_features = X.shape[1]
        if n_features < 2:
            messagebox.showerror("Помилка", "Недостатньо
ознак для побудови графіка.")
            return
        window = tk.Toplevel(root)
        window.title("Діаграми розсіювання")
        notebook = ttk.Notebook(window)
        notebook.pack(expand=True, fill='both')
        for i in range(min(n_features - 1, 3)):
            frame = tk.Frame(notebook)
            notebook.add(frame, text=f"{features[i]} vs {features[i
+ 1]}")
            fig, ax = plt.subplots(figsize=(5, 4))
            ax.scatter(X[:, i], X[:, i + 1], c=labels, cmap='viridis')
            ax.set_title(f"{features[i]} vs {features[i + 1]}")
            ax.set_xlabel(features[i])
            ax.set_ylabel(features[i + 1])
            canvas = FigureCanvasTkAgg(fig, master=frame)
            canvas.draw()
            canvas.get_tk_widget().pack()
            def save_plot(fig=fig,
name=f"{features[i]}_vs_{features[i + 1]}"):
                file_path = filedialog.asksaveasfilename(
                    defaultextension=".png",
                    initialfile=f"{name}.png",
                    filetypes=[("PNG", "*.png")]
                )
                if file_path:
                    fig.savefig(file_path)
                    messagebox.showinfo("Збережено", f"Діаграму
збережено:\n{file_path}")
            tk.Button(frame, text="Зберегти",
command=save_plot).pack(pady=5)

    def show_histogram():
        if clustering_result['X'] is None or df is None:
            messagebox.showwarning("Увага", "Спочатку
виконайте обробку.")
            return
        feature_names = clustering_result['feature_names']
        window = tk.Toplevel(root)
        window.title("Гістограми")
        notebook = ttk.Notebook(window)
        notebook.pack(expand=True, fill='both')
        for name in feature_names:
            if name not in df.columns:
                continue
            data = df[name].dropna().values
            frame = tk.Frame(notebook)
            notebook.add(frame, text=name)
            fig, ax = plt.subplots(figsize=(5, 4))
            n, bins, patches = ax.hist(data, bins=20, color='skyblue',
edgecolor='black')
            ax.set_title(f"Гістограма: {name}")
            ax.set_xlabel(name)
            ax.set_ylabel("Кількість")
            for j in range(len(patches)):
                bin_center = 0.5 * (bins[j] + bins[j + 1])
                height = patches[j].get_height()
                if height > 0:
                    ax.text(bin_center, height, str(int(height)),
ha='center', va='bottom', fontsize=8)
            canvas = FigureCanvasTkAgg(fig, master=frame)
            canvas.draw()
            canvas.get_tk_widget().pack()
            def save_hist(fig=fig, name=name):
                file_path = filedialog.asksaveasfilename(
                    defaultextension=".png",
                    initialfile=f"{name}.png",
                    filetypes=[("PNG", "*.png")]
                )
                if file_path:
                    fig.savefig(file_path)
                    messagebox.showinfo("Збережено", f"Гістограму
збережено:\n{file_path}")
            tk.Button(frame, text="Зберегти",
command=save_hist).pack(pady=5)
    def filter_and_show_cleaned_data():
        if df is None:
            messagebox.showwarning("Увага", "Спочатку
завантажте дані.")
            return
        filter_window = tk.Toplevel(root)
        filter_window.title("Фільтрація та очищення даних")
        range_entries = []
        delete_vars = []
        tk.Label(filter_window, text="Фільтрація
діапазону:").pack(pady=5)
        for col in numeric.columns:
            frame = tk.Frame(filter_window)
            frame.pack(padx=5, pady=2, fill='x')
            tk.Label(frame, text=col, width=20,
anchor='w').pack(side='left')
            min_entry = tk.Entry(frame, width=10)
            min_entry.pack(side='left')
            tk.Label(frame, text=" до ").pack(side='left')
            max_entry = tk.Entry(frame, width=10)
            max_entry.pack(side='left')
            range_entries.append((col, min_entry, max_entry))

        tk.Label(filter_window, text="Оберіть колонки для
видалення:").pack(pady=5)
        for col in df.columns:
            var = tk.IntVar()
            cb = tk.Checkbutton(filter_window, text=col,
variable=var)
            cb.pack(anchor='w')
            delete_vars.append((col, var))
        def apply_filter():
            filtered = df.copy()
            for col, min_entry, max_entry in range_entries:
                try:
                    min_val = float(min_entry.get()) if min_entry.get()
else -float('inf')
                    max_val = float(max_entry.get()) if
max_entry.get() else float('inf')
                    if min_val > max_val:
                        min_val, max_val = max_val, min_val
                    filtered = filtered[(filtered[col] >= min_val) &
(filtered[col] <= max_val)]
                except ValueError:
                    messagebox.showerror("Помилка",
f"Некоректний діапазон для {col}")
                    return
            cols_to_delete = [col for col, var in delete_vars if
var.get() == 1]
            filtered = filtered.drop(columns=cols_to_delete,
errors='ignore')
            if filtered.empty:
                messagebox.showinfo("Результат", "Немає даних
після фільтрації.")
                return
            result_window = tk.Toplevel(filter_window)
            result_window.title("Очищені дані")
            text = tk.Text(result_window, height=10, width=180)
```

```
        text.insert(tk.END,
filtered.head(100).to_string(index=False))
        text.pack()
        def save_filtered():
            file_path = filedialog.asksaveasfilename(
                defaultextension=".csv",
                filetypes=[("CSV файли", "*.csv")],
                initialfile="filtered_data.csv"
            )
            if file_path:
                filtered.to_csv(file_path, index=False)
                messagebox.showinfo("Збережено", f"Дані
збережено:\n{file_path}")
        tk.Button(result_window, text="Зберегти у CSV",
command=save_filtered).pack(pady=5)
    tk.Button(filter_window, text="Застосувати",
command=apply_filter, bg="#75e69b").pack(pady=10)
  tk.Button(root, text="Повернутися в меню", bg="#75e69b",
command=show_main_menu).pack(pady=5)
  tk.Button(root, text="Завантажити та переглянути дані",
bg="#75e69b", command=load_and_show_data).pack(pady=5)
  tk.Button(root, text="Показати діаграму", bg="#75e69b",
command=show_scatter_plot).pack(pady=5)
  tk.Button(root, text="Показати гістограму", bg="#75e69b",
command=show_histogram).pack(pady=5)
  tk.Button(root, text="Показати результати", bg="#75e69b",
command=show_summary).pack(pady=5)
  tk.Button(root, text="Створити очищену таблицю",
bg="#75e69b",
command=filter_and_show_cleaned_data).pack(pady=5)
 def load_file():
    nonlocal raw_text
    file_path = filedialog.askopenfilename(
        filetypes=[("Word файли", "*.docx"), ("Текстові
файли", "*.txt"), ("Всі файли", "*.*")]
    )
    if file_path:
        try:
            if file_path.endswith('.docx'):
                doc = Document(file_path)
                raw_text = "\n".join([para.text for para in
doc.paragraphs])
            else:
                with open(file_path, 'r', encoding='utf-8') as f:
                    raw_text = f.read()
            text_area.delete(1.0, tk.END)
            text_area.insert(tk.END, raw_text)
            nonlocal cleaned_text, tokens
            cleaned_text = ""
            tokens = []
        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося
завантажити файл: {e}")
  def lemmatize_ukrainian(text):
    doc = nlp_uk(text)
    lemmas = [word.lemma for sent in doc.sentences for word
in sent.words]
    filtered = [lemma for lemma in lemmas if lemma not in
uk_stopwords and lemma not in string.punctuation]
    nonlocal tokens
    tokens = filtered
    return ' '.join(filtered)
  def clean_text(text):
    text = text.lower()
    text = BeautifulSoup(text, "html.parser").get_text()
    text = emoji.replace_emoji(text, replace='')
    text = re.sub(r'\d+', '', text)
    text = re.sub(r'\n', ' ', text)
    text = re.sub(r'[^\w\sa-яєіїґA-ЯЄІЇҐ]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return lemmatize_ukrainian(text)

  def process_text():
    nonlocal cleaned_text
    if not raw_text:
        messagebox.showwarning("Увага", "Спочатку
завантажте файл.")
        return
    try:
        cleaned_text = clean_text(raw_text)
        text_area.delete(1.0, tk.END)
        text_area.insert(tk.END, cleaned_text)
    except Exception as e:
        messagebox.showerror("Помилка обробки", f"Не
вдалося обробити текст: {e}")
  def save_text():
    if not cleaned_text:
        messagebox.showwarning("Увага", "Немає тексту для
збереження.")
        return
    file_path = filedialog.asksaveasfilename(
        defaultextension=".txt",
        filetypes=[("Текстові файли", "*.txt"), ("Word файли",
"*.docx"), ("CSV файли", "*.csv")]
    )
    if file_path:
        try:
            if file_path.endswith(".txt"):
                with open(file_path, 'w', encoding='utf-8') as f:
                    f.write(cleaned_text)
            elif file_path.endswith(".docx"):
                doc = Document()
                doc.add_paragraph(cleaned_text)
                doc.save(file_path)
            elif file_path.endswith(".csv"):
                import pandas as pd
                df = pd.DataFrame({
                    'original_text': [raw_text],
                    'cleaned_text': [cleaned_text],
                    'tokens': [tokens]
                })
                df.to_csv(file_path, index=False, encoding='utf-8')
            messagebox.showinfo("Успіх", "Файл збережено.")
        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося
зберегти файл: {e}")
  def vectorize():
    nonlocal vectorization_results
    if not cleaned_text:
        messagebox.showwarning("Увага", "Обробіть текст
перед векторизацією.")
        return
    try:
        text = cleaned_text
        tokens_list = [text.split()]
        bow = CountVectorizer()
        bow_matrix = bow.fit_transform([text])
        vectorization_results['bow'] = bow_matrix.toarray()
        tfidf = TfidfVectorizer()
        tfidf_matrix = tfidf.fit_transform([text])
        vectorization_results['tfidf'] = tfidf_matrix.toarray()
        w2v = Word2Vec(sentences=tokens_list,
vector_size=100, window=5, min_count=1, workers=4)
        vectorization_results['w2v'] = np.mean([w2v.wv[word]
for word in tokens_list[0]], axis=0)
        ft = FastText(sentences=tokens_list, vector_size=100,
window=5, min_count=1, workers=4)
        vectorization_results['ft'] = np.mean([ft.wv[word] for
word in tokens_list[0]], axis=0)
```

```
            tokenizer =
BertTokenizer.from_pretrained("bert-base-multilingual-cased")
            model =
BertModel.from_pretrained("bert-base-multilingual-cased")
            inputs = tokenizer(text, return_tensors="pt",
truncation=True, padding=True, max_length=512)
            with torch.no_grad():
                outputs = model(**inputs)
            vectorization_results['bert'] =
outputs.last_hidden_state.mean(dim=1).numpy()
            show_vectorization_info()
        except Exception as e:
            messagebox.showerror("Помилка векторизації",
str(e))
    def show_vectorization_info():
        info_window = tk.Toplevel(root)
        info_window.title("Результати векторизації")
        info_window.geometry("600x400")
        notebook = ttk.Notebook(info_window)
        notebook.pack(fill=tk.BOTH, expand=True)
        # Загальна інформація
        general_frame = ttk.Frame(notebook)
        notebook.add(general_frame, text="Загальна
інформація")
        info_text = tk.Text(general_frame, wrap=tk.WORD)
        info_scroll = ttk.Scrollbar(general_frame,
command=info_text.yview)
        info_text.config(yscrollcommand=info_scroll.set)
        info_scroll.pack(side=tk.RIGHT, fill=tk.Y)
        info_text.pack(fill=tk.BOTH, expand=True)
        info = "Розмірності векторів:\n\n"
        for method, vec in vectorization_results.items():
            info += f"{method.upper()}: {vec.shape}\n"
        info_text.insert(tk.END, info)
        info_text.config(state=tk.DISABLED)
        # Деталі для кожного методу
        for method, vec in vectorization_results.items():
            frame = ttk.Frame(notebook)
            notebook.add(frame, text=method.upper())
            text = tk.Text(frame, wrap=tk.WORD)
            scroll = ttk.Scrollbar(frame, command=text.yview)
            text.config(yscrollcommand=scroll.set)

            scroll.pack(side=tk.RIGHT, fill=tk.Y)
            text.pack(fill=tk.BOTH, expand=True)
            display_data = vec
            if len(vec.shape) > 1 and vec.shape[1] > 20:
                display_data = vec[:, :20]
            text.insert(tk.END, f"Перші елементи
вектора:\n\n{str(display_data)}")
            text.config(state=tk.DISABLED)
    def show_stats():
        nonlocal tokens
        if not tokens:
            messagebox.showwarning("Увага", "Спочатку
обробіть текст.")
            return
        stats_window = tk.Toplevel(root)
        stats_window.title("Статистика тексту")
        stats_window.geometry("500x400")
        # Основні статистики
        num_tokens = len(tokens)
        num_unique_tokens = len(set(tokens))
        token_counts = Counter(tokens)
        most_common = token_counts.most_common(10)

        stats_text = tk.Text(stats_window, wrap=tk.WORD)
        scroll = ttk.Scrollbar(stats_window,
command=stats_text.yview)
        stats_text.config(yscrollcommand=scroll.set)
        scroll.pack(side=tk.RIGHT, fill=tk.Y)
        stats_text.pack(fill=tk.BOTH, expand=True)
        report = f"""
        Статистика тексту:
        ----------------------------
        Загальна кількість токенів: {num_tokens}
        Унікальних токенів: {num_unique_tokens}
        Середня довжина токена: {np.mean([len(t) for t in
tokens]):.2f} символів

        10 найчастіших слів:
        ----------------------------
        """
        for word, count in most_common:
            report += f"{word}: {count} разів\n"

        stats_text.insert(tk.END, report)
        stats_text.config(state=tk.DISABLED)
```