МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
“ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ”
Кафедра прикладної математики та інформатики

Методичні вказівки і завдання до практичних занять з дисципліни “Інтелектуальна обробка Web даних (Web-Mining)“

для магістрів, що навчаються на спеціальності
122 «Комп’ютерні науки», усіх форм навчання

ДРОГОБИЧ-2025

УДК 004.4’2:004.912(072)

М 54

Методичні вказівки і завдання до практичних занять з дисципліни «Інтелектуальна обробка Web даних (Web-Mining)» для магістрів, що навчаються на спеціальності 122 «Комп’ютерні науки» усіх форм навчання / уклад. М.О. Александров. – Дрогобич: ДонНТУ, 2025. – 58 с.

Наведено теоретичні зведення, методичні рекомендації, контрольні питання й завдання для виконання практичних занять з наступних розділів інтелектуальної обробки Web даних:

- введення в основи Web Scraping та Web Crawling;
- обробка текстових даних та застосування бібліотек для природної мови (NLTK, SpaCy);
- аналіз структури веб-ресурсів за допомогою графового аналізу;
- використання методів машинного навчання для аналізу Web даних та соціальних сигналів.

Укладач: Микита АЛЕКСАНДРОВ, д.ф., доцент кафедри ПМІ

Рецензент: Сергій КОВАЛЬОВ, к.т.н., доц., зав. каф. ЕТ і КІ

Відповідальний за випуск: Ярослав ДОРОГИЙ, д.т.н., проф., зав. каф. ПМІ

Затверджено навчально-методичним відділом ДВНЗ ДонНТУ,
протокол №5 від 25.03.2025

Розглянуто на засіданні кафедри ПМІ, протокол № 2 від 21.02.2025р.

ЗМІСТ

ВСТУП

У сучасному світі швидкого розвитку інформаційних технологій та глобалізації доступу до онлайн-даних вміння ефективно працювати з веб-ресурсами для отримання, обробки та аналізу інформації стало важливим для фахівців у галузі комп'ютерних наук. Інтернет став джерелом величезних обсягів даних, які необхідно вміти правильно обробляти для прийняття рішень, розвитку технологій та бізнес-процесів. Одним із важливих напрямків є освоєння інструментів і методів веб-скрейпінгу та веб-краулінгу, що дозволяють автоматизувати збір даних з інтернет-ресурсів.

Ці методи використовуються для аналізу трендів, вивчення соціальних сигналів і дослідження взаємодій на веб-платформах, що є важливим для маркетингу, науки, економіки та інших сфер. Зібрані дані дозволяють виявляти популярні теми, прогнозувати сезонні зміни та аналізувати вплив соціальних сигналів на загальні тенденції. Метою цих методичних вказівок є навчити магістрів спеціальності 122 «Комп'ютерні науки» основам роботи з даними веб-ресурсів, використовуючи різноманітні інструменти для збору, обробки і аналізу інформації.

Практичні роботи охоплюють завдання для освоєння веб-скрейпінгу, веб-краулінгу та обробки текстових даних, а також графового аналізу взаємозв'язків між веб-сторінками. Студенти ознайомляться з методами збору та аналізу трендів і соціальних сигналів, використовуючи сучасні платформи для моніторингу актуальних тем на веб-ресурсах. Курс дасть магістрам необхідні знання та навички для роботи з веб-даними, що є важливим компонентом підготовки висококваліфікованих фахівців.

Практичне заняття №1

«Основи Web Scraping»

Мета роботи:

Ознайомитися з основами Web Scraping. Вивчити принципи отримання та обробки HTML-коду веб-сторінок. Освоїти використання бібліотеки BeautifulSoup для парсингу веб-даних. Отримати практичний досвід збору даних із веб-сайтів.

Завдання:

1. Ознайомитися з поняттям Web Scraping, методами отримання HTML-коду та інструментами для його обробки.
2. Встановити необхідні бібліотеки.
3. Написати Python-скрипт, який:
   - Виконує HTTP-запит до заданої веб-сторінки.
   - Аналізує отриманий HTML-код за допомогою BeautifulSoup.
   - Витягує та обробляє потрібну інформацію (заголовки, посилання, ціни, тексти тощо).
   - Виводить отримані дані у зручному форматі (наприклад, у вигляді таблиці).
4. Оформити звіт із поясненням коду та отриманими результатами.

Вимоги до виконання:

- Код повинен бути написаний мовою Python 3.
- Використання бібліотек requests та BeautifulSoup.
- Дотримання етичних норм (парсинг відкритих веб-сторінок, дотримання файлу robots.txt).

- Скрипт має працювати коректно та виводити інформацію у структурованому вигляді.

- У звіті повинні бути описані основні етапи виконання роботи, скріншоти результатів та пояснення коду.

Вимоги до звіту:

1. Мета, завдання та варіант.
2. Теоретичні відомості про Web Scraping.
3. Опис процесу розробки скрипту. (Кожен етап повинен бути описаний)
4. Опис процесу тестування роботи розробленого додатку. (Кожен скріншот повинен містити опис та підпис.)
5. Код програми.
6. Висновки.

Таблиця 1.1. Варіанти завдань до практичного заняття 1

№	Завдання
1	Отримати заголовки статей із головної сторінки новинного сайту (наприклад, https://www.bbc.com/)
2	Зібрати назви, ціни та посилання на товари з сторінки онлайн-магазину (наприклад, https://rozetka.com.ua/)
3	Витягнути всі посилання з головної сторінки будь- кого сайту
4	Отримати список популярних фільмів та їх рейтинги з `https://www.imdb.com/chart/top/`
5	Зібрати курс валют із веб-сторінки банку (наприклад, `https://www.xe.com/`)
6	Витягнути список вакансій із сайту пошуку роботи (наприклад, `https://www.work.ua/`)
7	Отримати прогноз погоди на тиждень з погодного сайту (`https://www.weather.com/`)
8	Витягнути коментарі до останньої статті на новинному сайті
9	Отримати список останніх спортивних результатів із сайту статистики
10	Зібрати дані про криптовалюту (ціни, зміни) з `https://coinmarketcap.com/`

Методичні вказівки до виконання практичного заняття № 1

Приклад завдання. Отримати заголовки новин із веб-сайту example.com.

Загальне пояснення.

- ***Встановлення бібліотек.*** Спочатку ми імпортуємо необхідні бібліотеки: requests для відправки запитів та отримання HTML, і BeautifulSoup для обробки цього HTML.
- ***HTTP-запит.*** Виконується запит до веб-сторінки, щоб отримати її вміст.
- ***Парсинг HTML.*** За допомогою BeautifulSoup ми перетворюємо HTML-код на зручну для обробки структуру.
- ***Збір даних.*** Використовуємо метод find_all() для пошуку конкретних елементів на сторінці (заголовків новин у нашому прикладі).
- ***Виведення результатів.*** Програма виводить знайдені заголовки новин, очищаючи їх від зайвих пробілів.

1. **Встановлення необхідних бібліотек.** Використовуємо PIP для встановлення необхідних бібліотек.

```
pip install requests beautifulsoup4
```

Рисунок 1.1 – Приклад встановлення бібліотеки

2. **Імпорт бібліотеки BeautifulSoup до проєкту.**

```
import requests
from bs4 import BeautifulSoup
```

Рисунок 1.2 – Імпорт бібліотеки

(import requests) - це бібліотека Python, яка дозволяє робити HTTP-запити до веб-сайтів. Вона використовуватиметься для отримання HTML-коду веб-сторінки.

BeautifulSoup — це бібліотека для парсингу HTML та XML-даних. Вона допомагає "розбирати" HTML-код і знаходити в ньому потрібні елементи.

3. Визначення URL сайту для парсингу.

```
# URL сайту для парсингу
url = "https://example.com/news"
```

Рисунок 1.3 – Визначення URL

В змінній **url** зберігається адреса веб-сторінки, яку ми будемо парсити. В даному випадку це просто умовний приклад URL (https://example.com/news). В реальній роботі, ви замінюєте його на актуальний веб-сайт, дані з якого потрібно отримати.

4. Виконання HTTP-запиту.

```
# Виконання HTTP-запиту
response = requests.get(url)
```

Рисунок 1.4 – HTTP-запит

За допомогою функції **requests.get(url)** ми відправляємо HTTP-запит до зазначеного URL. Цей запит отримує HTML-код веб-сторінки. Результат запиту зберігається у змінній response, яка містить:

- **response.text -** текстовий вміст HTML-коду.
- **response.status_code -** код статусу відповіді (наприклад, 200 означає, що сторінка завантажена успішно).

5. Перевірка статусу відповіді.

```
# Перевірка статусу відповіді
if response.status_code == 200:
```

Рисунок .1.5 – Перевірка статусу відповіді

Цей блок перевіряє, чи був запит успішним. Якщо код статусу відповіді — 200, це означає, що запит виконаний успішно, і сторінка доступна для подальшої обробки. Якщо статус не 200 (наприклад, 404 або 500), програма виведе помилку, оскільки сторінка не завантажилась.

6. Створення об'єкта BeautifulSoup.

```
# Створення об'єкта BeautifulSoup
soup = BeautifulSoup(response.text, "html.parser")
```

Рисунок 1.6 – Об'єкт BeautifulSoup

Це команда, яка створює об'єкт BeautifulSoup для обробки HTML-коду веб-сторінки.

- **response.text** - це сам HTML-код, який ми отримали з сайту.
- **"html.parser"** - вказує, що ми використовуємо стандартний парсер Python для аналізу HTML.

Об'єкт **soup** тепер містить структуроване представлення HTML-коду веб-сторінки, з яким ми можемо працювати (шукати елементи, витягувати дані тощо).

7. Знаходження заголовків новин.

```
# Знаходження заголовків новин
news_headlines = soup.find_all("h2", class_="news-title")
```

Рисунок 1.7 – Виокремлення заголовків новин

Цей метод шукає всі елементи на сторінці, які мають тег **<h2>** і клас **"news-title". find_all()** повертає список всіх знайдених елементів, які відповідають зазначеним критеріям. У нашому випадку це можуть бути заголовки новин або статей. Тег **<h2>** часто використовується для заголовків, а клас **"news-title"** — для конкретизації заголовків новин на сайті. Результат зберігається в змінній **news_headlines**, яка міститиме список усіх заголовків.

8. Виведення заголовків.

```
# Виведення заголовків
for i, headline in enumerate(news_headlines, start=1):
    print(f"{i}. {headline.text.strip()}")
```

Рисунок 1.8 – Виведення заголовків новин.

Це цикл, який перебирає всі заголовки новин, які ми знайшли в попередньому кроці. **enumerate()** додає до кожного елемента індекс (починаючи з 1), що дозволяє вивести нумерацію.

headline.text.strip() - для кожного заголовка:

- **.text** - отримує текстову частину елемента (без HTML-тегів).
- **.strip()** - видаляє зайві пробіли на початку і в кінці тексту.

print(f"{i}. {headline.text.strip()}") - виводить номер заголовка і його очищений текст.

```
1. Україна підписала нову угоду про співпрацю
2. Вчені розробили революційну технологію
3. Новий смартфон отримав інноваційний дизайн
```

Рисунок 1.9 – Приклад виведення

Лістинг:

```
import requests
from bs4 import BeautifulSoup

# URL сайту для парсингу
url = "https://example.com/news"
```

```
# Виконання HTTP-запиту
response = requests.get(url)

# Перевірка статусу відповіді
if response.status_code == 200:
    # Створення об'єкта BeautifulSoup
    soup = BeautifulSoup(response.text, "html.parser")

    # Знаходження заголовків новин
    news_headlines = soup.find_all("h2", class_="news-title")

    # Виведення заголовків
    for i, headline in enumerate(news_headlines, start=1):
        print(f"{i}. {headline.text.strip()}")  # Вивід очищеного тексту

else:
    print("Помилка отримання сторінки:", response.status_code)
```

Контрольні питання:

1. Що таке Web Scraping і для чого він використовується?
2. Які бібліотеки Python зазвичай використовуються для Web Scraping і чим вони відрізняються?
3. Що таке HTTP-запит і як ми його виконуємо за допомогою бібліотеки requests?
4. Як перевірити статус HTTP-відповіді за допомогою бібліотеки requests? Що означає код статусу 200?
5. Що таке парсинг HTML і які бібліотеки Python зазвичай використовуються для цього?
6. Які основні методи BeautifulSoup для пошуку елементів на веб-сторінці? Поясніть їх роботу.
7. Чим відрізняються методи find() та find_all() в BeautifulSoup?
8. Як можна витягнути текст з HTML-елемента, використовуючи BeautifulSoup?
9. Що таке CSS-селектори і як вони використовуються для пошуку елементів на веб-сторінці?

10. Що таке .strip() у Python, і як це допомагає обробляти текст після його витягнення з HTML-елемента?
11. Які є можливі проблеми при виконанні Web Scraping і як їх можна уникнути?
12. Які етичні та правові аспекти варто враховувати при здійсненні Web Scraping?
13. Як можна обробити помилки під час отримання даних через HTTP-запит (наприклад, якщо сторінка не завантажилась)?
14. Що таке User-Agent і як він може бути використаний під час виконання запитів до веб-сторінок?
15. Як ви можете використовувати Web Scraping для збору новин чи даних з онлайн-магазинів?

Література: [1-9]

Практичне заняття №2

«Основи Web Crawling»

Мета роботи:

Ознайомитися з основами Web Crawling та його застосуванням. Освоїти використання бібліотеки Scrapy (або альтернатив) для створення веб-краулера. Навчитися витягувати інформацію зі сторінок веб-сайтів та зберігати її у зручному формат.

Завдання:

- Ознайомитися з поняттям Web Crawling та відмінністю від Web Scraping.

- Встановити та налаштувати середовище для роботи з бібліотекою Scrapy або альтернативними інструментами (Selenium, Requests + BeautifulSoup).

- Реалізувати простий веб-краулер для збору даних із вибраного сайту.

- Зберегти отримані дані у форматі CSV або JSON.

- Додати обробку помилок (наприклад, перевірку статусу HTTP, затримки між запитами).

- Перевірити роботу краулера та підготувати звіт.

Вимоги до виконання:

- Використання бібліотеки Scrapy або аналогічних інструментів.
- Реалізація обходу кількох сторінок (не менше 5).
- Витягнення мінімум трьох елементів зі сторінки (наприклад, заголовок, посилання, дата публікації).
- Збереження результатів у CSV або JSON.

- Дотримання правил етичного збору даних (не зловживати запитами, поважати robots.txt).

- Додати коментарі до коду та короткий звіт (мета, підхід, отримані результати).

Вимоги до звіту:

1. Мета, завдання та варіант.
2. Теоретичні відомості про Web Crawling.
3. Опис процесу розробки скрипту. (Кожен етап повинен бути описаний)
4. Опис процесу тестування роботи розробленого додатку. (Кожен скріншот повинен містити опис та підпис.)
5. Код програми та приклад результатів (CSV/JSON).
6. Висновки.

Таблиця 2.1 – Варіанти завдань до практичного заняття 2

№	Тема краулера	Що збирати
1	Краулер для новинного сайту	Заголовки, посилання, дата публікації
2	Аналіз оголошень на OLX або подібних платформах	Назва товару, ціна, місто
3	Збір статей із сайту Wikipedia	Заголовки, короткий опис, посилання
4	Парсинг блогу про технології	Назва статті, автор, дата, URL
5	Аналіз інтернет-магазину	Назва товару, ціна, рейтинг, URL
6	Моніторинг вакансій	Назва вакансії, компанія, зарплата, місто
7	Краулер для сайту з рецензіями на фільми	Назва фільму, рейтинг, короткий опис
8	Збір курсів валют із фінансового сайту	Назва валюти, курс купівлі/продажу, дата
9	Аналіз спортивних новин	Команда, результат матчу, дата
10	Краулер для аналізу продуктів у супермаркетах	Назва продукту, ціна, категорія

Методичні вказівки до виконання практичного заняття № 2

Приклад простого веб-краулера для збору заголовків статей (рис. 2.1).

```
import scrapy

class NewsSpider(scrapy.Spider):
    name = "news"
    start_urls = ["https://example-news-site.com"]  # Змініть на реальний сайт

    def parse(self, response):
        for article in response.css("article"):
            yield {
                "title": article.css("h2 a::text").get(),
                "link": article.css("h2 a::attr(href)").get(),
                "date": article.css(".date::text").get(),
            }

        # Знаходження посилань на наступні сторінки
        next_page = response.css("a.next::attr(href)").get()
        if next_page:
            yield response.follow(next_page, self.parse)
```

Рисунок 2.1 – Збір заголовків статей

Запуск краулера та збереження даних у JSON

```
scrapy runspider news_spider.py -o news.json
```

Рисунок 2.2 – Збереження данних

1. **Імпорт бібліотеки Scrapy до проєкту.** Scrapy - це фреймворк для парсингу та краулінгу веб-сторінок. Ми імпортуємо його, щоб створити клас-павук для обходу сайтів.

```
import scrapy
```

Рисунок 2.3 – Імпорт бібліотеки

2. Створюємо клас краулера.

```
class NewsSpider(scrapy.Spider):
    name = "news"
```

Рисунок 2.4 – Створення класу краулера.

Клас ***NewsSpider*** успадковується від ***scrapy.Spider***, що означає, що ми створюємо спеціальний веб-краулер. ***name = "news"*** - це ім'я павука, яке використовуватиметься для запуску команди Scrapy.

3. Вказуємо стартові URL-адреси.

```
start_urls = ["https://example-news-site.com"]  # Змініть на реальний сайт
```

Рисунок 2.5 – Стартова URL-адреса.

start_urls містить список початкових сторінок, з яких почнеться краулінг. У цьому прикладі використовується фейковий сайт ***"https://example-news-site.com",*** який потрібно замінити на реальний новинний ресурс.

4. Логіка обробки сторінки.

```
def parse(self, response):
```

Рисунок 2.6 – Створення класу краулера.

Цей метод визначає, як Scrapy повинен обробляти кожну веб-сторінку після її завантаження. Аргумент ***response*** містить HTML-код сторінки та допоміжні методи для парсингу.

5. Цикл для збору даних зі сторінки.

```
for article in response.css("article"):
```

Рисунок 2.7 – Цикл для збору даних.

response.css("article") вибирає всі HTML-елементи **<*article*>** на сторінці (якщо сайт містить такі елементи). Цикл ***for*** проходить по всіх знайдених **<*article*>** і обробляє кожен із них.

6. Витягуємо дані (заголовок, посилання, дату публікації).

```
yield {
    "title": article.css("h2 a::text").get(),
    "link": article.css("h2 a::attr(href)").get(),
    "date": article.css(".date::text").get(),
}
```

Рисунок 2.8 – Пошук потрібних елементів.

Використовуємо ***article.css(...)***, щоб знайти потрібні елементи:

- ***"title"*** → `шукає заголовок у` ***h2 a::text*** (текст всередині тегу <a> в <h2>).
- ***"link"*** → `витягує посилання` ***href*** (атрибут <a>).
- ***"date"*** → `шукає дату публікації .date::text (припускаємо, що дата має клас date).`
    - ***.get()*** повертає перший знайдений елемент.
    - ***yield {}*** — Scrapy автоматично зберігає ці дані в JSON або CSV (якщо вказано при запуску).

7. Обхід наступних сторінок.

```
next_page = response.css("a.next::attr(href)").get()
if next_page:
    yield response.follow(next_page, self.parse)
```

Рисунок 2.9 – Обхід сторінок.

response.css("a.next::attr(href)").get() шукає посилання на наступну сторінку (припускаємо, що є кнопка ***"Next"*** із класом **next**). Якщо ***next_page***

знайдено, Scrapy переходить за ним і знову викликає ***parse()*** для нової сторінки.

8. Запуск краулера та збереження результатів у JSON.

```
scrapy runspider news_spider.py -o news.json
```

Рисунок 2.10 – Запуск краулера.

scrapy runspider news_spider.py - команда для запуску Scrapy.

-o news.json - означає, що всі зібрані дані будуть збережені у news.json.

Лістинг:

```
import scrapy
class NewsSpider(scrapy.Spider):
    name = "news"
    start_urls = ["https://example-news-site.com"]  # Змініть на реальний сайт

    def parse(self, response):
        for article in response.css("article"):
            yield {
                "title": article.css("h2 a::text").get(),
                "link": article.css("h2 a::attr(href)").get(),
                "date": article.css(".date::text").get(),
            }
        # Знаходження посилань на наступні сторінки
        next_page = response.css("a.next::attr(href)").get()
        if next_page:
            yield response.follow(next_page, self.parse)
```

Контрольні питання:

1. Що таке Web Crawling, і яка його основна мета?
2. Які основні відмінності між Web Scraping та Web Crawling?
3. Які бібліотеки можна використовувати для реалізації веб-краулінгу в Python?
4. Що таке robots.txt, і як він впливає на роботу веб-краулерів?
5. Як визначити стартові URL-адреси для краулінгу?
6. Як у Scrapy реалізується автоматичний перехід на наступні сторінки?
7. Як витягнути текст із HTML-елемента за допомогою CSS-селекторів у Scrapy?
8. Як можна зберегти отримані дані у форматі JSON або CSV?
9. Які етичні та правові аспекти слід враховувати при створенні веб-краулера?
10. Як обробити помилки, якщо сайт блокує краулінг або видає помилки 403/404?
11. Як налаштувати затримку між запитами, щоб уникнути блокування?
12. Які існують альтернативи Scrapy для краулінгу веб-сайтів?
13. Як обмежити глибину краулінгу, щоб уникнути потрапляння в нескінченні петлі посилань?

Література: [1, 3-6, 10-11]

Практичне заняття №3

«Обробка текстових даних»

Мета роботи:

Ознайомитися з методами обробки текстових даних, включаючи токенізацію, стемінг, лематизацію, виділення ключових слів та аналіз частоти слів. Навчитися використовувати бібліотеки NLTK та SpaCy для обробки природної мови.

Завдання:

1. Завантажити та підготувати текстові дані.
2. Виконати базову очистку тексту (видалення стоп-слів, пунктуації, зайвих пробілів).
3. Провести токенізацію тексту.
4. Виконати стемінг та лематизацію.
5. Підрахувати частоту використання слів у тексті.
6. Визначити ключові слова тексту.
7. Візуалізувати результати (наприклад, побудувати Word Cloud або графік частоти слів).

Вимоги до виконання:

- Використання бібліотек NLTK та/або SpaCy.
- Використання всіх етапів обробки тексту.
- Пояснення отриманих результатів у звіті.
- Надання коду, скріншотів результатів та висновків.
- Додавання коментарів до коду та звіту (мета, підхід, отримані результати).

Вимоги до звіту:

1. Мета, завдання та варіант.
2. Опис підготовки початкового текст.
3. Опис процесу розробки додатку. (Кожен етап повинен бути описаний)
4. Опис процесу тестування роботи розробленого додатку. (Кожен скріншот повинен містити опис та підпис.)
5. Код програми та приклад результатів
6. Висновки.

Таблиця 3.1 – Варіанти завдань до практичного заняття 3

№	Текст для обробки
1	Аналіз тексту з новинного сайту (технології, економіка, спорт).
2	Обробка відгуків користувачів про товар або послугу.
3	Токенізація та аналіз статті з Wikipedia (будь-яка довільна тема).
4	Робота з текстами книг (наприклад, перший розділ "Аліси в країні чудес").
5	Обробка тексту з Twitter (твітів) та аналіз частоти слів.
6	Аналіз коментарів під відео YouTube (наприклад, до популярного відео).
7	Обробка текстів з електронних листів (спам-фільтрація, аналіз ключових слів).
8	Лематизація та аналіз статті з блогу або форуму.
9	Виявлення ключових слів у тексті пісні або поеми.
10	Аналіз текстів веб-сайтів (збір і обробка контенту сторінки).

Методичні вказівки до виконання практичного заняття № 3

1. Завантаження тексту та його очистка.

```python
import nltk
import spacy
import string
from nltk.corpus import stopwords

# Завантажуємо англійські стоп-слова та токенізатор
nltk.download('stopwords')
nltk.download('punkt')

text = """Natural Language Processing (NLP) is a sub-field of artificial intelligence that deals with
the interaction between computers and humans using natural language. The ultimate goal of NLP is to
enable computers to understand, interpret, and generate human languages in a valuable way."""

# Видалення пунктуації та переведення тексту в нижній регістр
text_clean = text.lower().translate(str.maketrans('', '', string.punctuation))

# Токенізація - розбиття тексту на слова
tokens = nltk.word_tokenize(text_clean)

# Видалення стоп-слів (наприклад, "is", "the", "and")
filtered_tokens = [word for word in tokens if word not in stopwords.words('english')]

print(filtered_tokens)
```

Рисунок 3.1 – Приклад коду завантаження і очистки тексту.

1.1 **Імпорт бібліотек**:

- ***nltk (Natural Language Toolkit)*** – бібліотека для обробки природної мови.
- ***spacy*** – альтернативна бібліотека для NLP.
- ***string*** – для роботи з пунктуацією.
- ***stopwords*** – для отримання списку стоп-слів.

1.2 **Завантаження необхідних ресурсів**:

- ***nltk.download('stopwords')*** – завантажує набір стоп-слів.
- ***nltk.download('punkt')*** – завантажує токенізатор.

1.3 **Очистка тексту**:

- ***text.lower()*** – переводить текст у нижній регістр.
- ***translate(str.maketrans('', '', string.punctuation))*** – видаляє розділові знаки.

1.5 **Токенізація** - ***nltk.word_tokenize(text_clean)*** – розбиває текст на окремі слова.

1.6 **Видалення стоп-слів** - Використовуємо ***stopwords.words('english')*** для фільтрації загальних слів, які не несуть великого сенсу.

2. Стемінг та лематизація (NLTK та SpaCy).

```
from nltk.stem import PorterStemmer, WordNetLemmatizer

# Завантажуємо необхідні ресурси
nltk.download('wordnet')

# Ініціалізація стемера та лематизатора
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Виконання стемінгу та лематизації
stemmed_words = [stemmer.stem(word) for word in filtered_tokens]
lemmatized_words = [lemmatizer.lemmatize(word) for word in filtered_tokens]

print("Стемінг:", stemmed_words)
print("Лематизація:", lemmatized_words)
```

Рисунок 3.2 – Приклад коду стемінгу та лематизації.

2.1 Стемінг (Stemming) - Використовуємо ***PorterStemmer()*** для скорочення слів до їх кореня.

2.2 Лематизація (Lemmatization) - ***WordNetLemmatizer()*** з бібліотеки NLTK повертає слова в їх словникову форму.

2.3 Застосування до списку слів:

- ***[stemmer.stem(word) for word in filtered_tokens]*** – виконує стемінг для кожного слова.
- ***[lemmatizer.lemmatize(word) for word in filtered_tokens]*** – виконує лематизацію.

3. Аналіз частоти слів.

```
from collections import Counter

word_freq = Counter(filtered_tokens)

print("10 найбільш вживаних слів:", word_freq.most_common(10))
```

Рисунок 3.3 – Приклад коду аналізу частоти слів.

3.1 Імпорт Counter - *Counter* з модуля ***collections*** підраховує частоту слів.

3.2 Створення словника з частотами:

- ***word_freq = Counter(filtered_tokens)*** – створює словник, де ключі – це слова, а значення – кількість їх входжень.

3.3 Отримання 10 найчастіших слів - *word_freq.most_common(10)* повертає список із 10 найбільш вживаних слів.

4 Візуалізація Word Cloud

```
from collections import Counter

word_freq = Counter(filtered_tokens)

print("10 найбільш вживаних слів:", word_freq.most_common(10))



from wordcloud import WordCloud
import matplotlib.pyplot as plt

wordcloud = WordCloud(width=800, height=400, background_color='white').generate(" ".join(filtered_tokens))

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Рисунок 3.4 – Приклад коду візуалізації.

4.1 Імпорт бібліотек:

- ***WordCloud*** – для побудови хмари слів.
- ***matplotlib.pyplot*** – для відображення графіку.

4.2 Створення хмари слів:

- ***WordCloud(width=800, height=400, background_color='white')*** – задає розмір та фон.

- ***.generate(" ".join(filtered_tokens))*** – об’єднує слова в один рядок для аналізу.

4.3 Відображення графіку:

- ***plt.imshow(wordcloud, interpolation='bilinear')*** – показує зображення.
- ***plt.axis("off")*** – прибирає осі для кращої читабельності.

5 Результати роботи

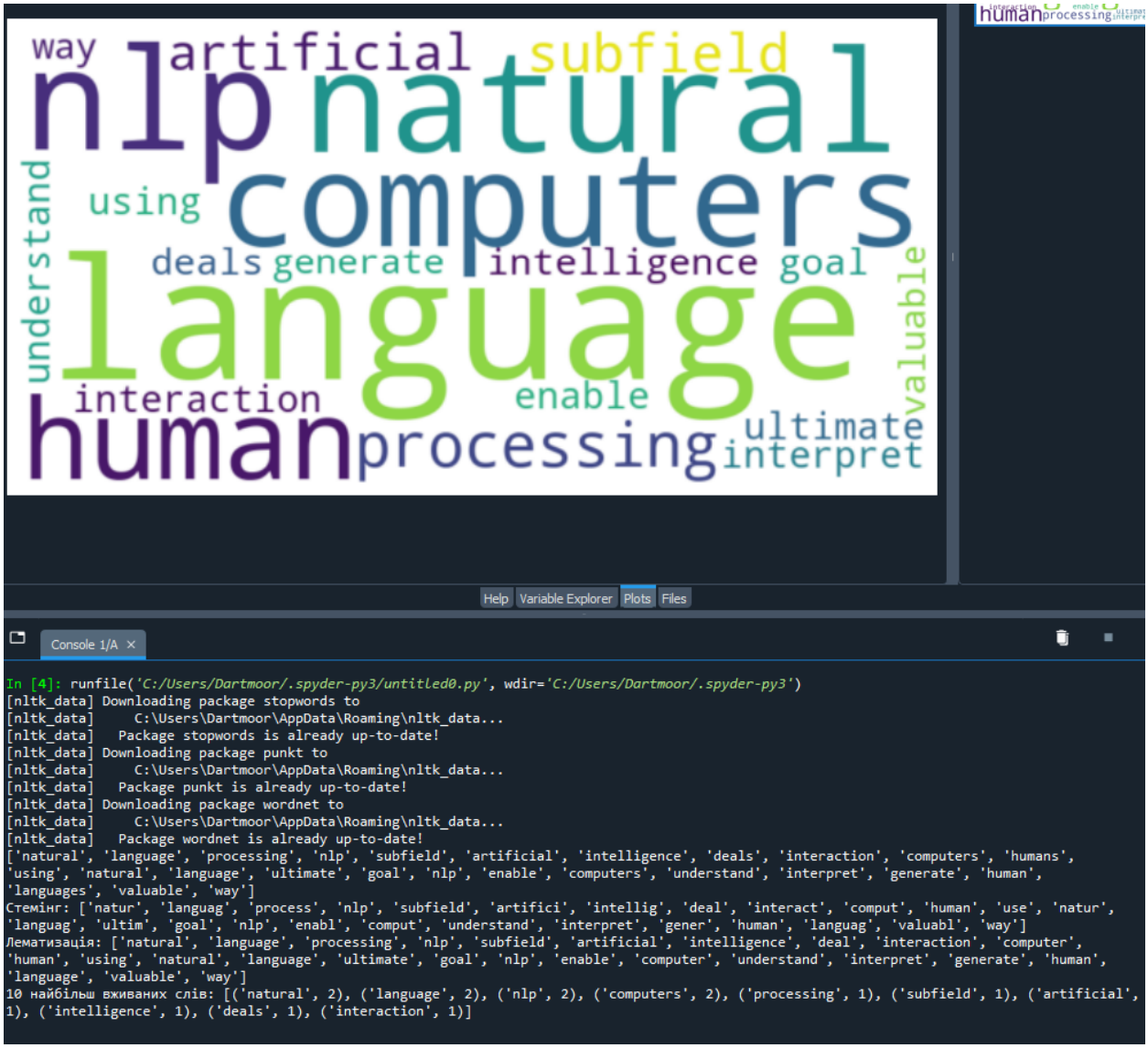


Рисунок 3.5 – Приклад результатів роботи додатку.

Контрольні питання:

1. Що таке токенізація і для чого вона використовується?
2. Які методи очистки тексту використовуються при аналізі даних?
3. Що таке стоп-слова, і чому їх варто видаляти з тексту?
4. Чим відрізняється стемінг від лематизації?
5. Які основні алгоритми стемінгу використовуються в NLP?
6. Чому лематизація вважається більш точною, ніж стемінг?
7. Яким чином можна підрахувати частоту слів у тексті?
8. Які слова найчастіше зустрічаються в тексті після видалення стоп-слів?
9. Як можна використовувати інформацію про частоту слів у практичних задачах?
10. Що таке "хмара слів" (Word Cloud) і як її використовувати?
11. Які бібліотеки Python використовуються для побудови хмари слів?
12. Чому важливо візуалізувати текстові дані перед їх аналізом?
13. Які основні проблеми можуть виникнути при попередній обробці тексту?
14. Чому важливо нормалізувати текст перед його обробкою?
15. Як обробка текстових даних використовується в завданнях машинного навчання та аналізу даних?

Література: [12-21]

Практичне заняття №4

«Аналіз графів Web-структури»

Мета роботи:

Ознайомитися з основами графового аналізу структури веб-сайтів. Навчитися будувати графи зв’язків між веб-сторінками та аналізувати їх за допомогою бібліотеки NetworkX у Python.

Завдання:

1. Зібрати дані про структуру веб-сайту (зв’язки між сторінками).
2. Побудувати граф веб-структури, де вершини — це сторінки, а ребра — це посилання між ними.
3. Візуалізувати отриманий граф.
4. Виконати базовий аналіз:
   a. знайти ступінь кожної вершини;
   b. визначити найважливіші вузли за алгоритмом PageRank;
   c. знайти найбільш зв’язні компоненти графа.

Вимоги до виконання:

- Використати Python та бібліотеку NetworkX для побудови та аналізу графа.

- Застосувати Matplotlib або Plotly для візуалізації графа.

- Використати Web Scraping (наприклад, BeautifulSoup або Scrapy) для отримання структури веб-сторінок або працювати з готовими даними.

- Оформити звіт з поясненням коду та висновками.

Вимоги до звіту:

1. Мета, завдання та варіант.
2. Опис процесу розробки додатку. (Кожен етап повинен бути описаний)
3. Опис процесу тестування роботи розробленого додатку. (Кожен скріншот повинен містити опис та підпис.)
4. Код програми та приклад результатів
5. Висновки.

Таблиця 4.1 – Варіанти завдань до практичного заняття 4

№	Текст для обробки
1	Побудувати граф структури веб-сайту університету та знайти найважливіші сторінки.
2	Проаналізувати веб-структуру новинного порталу (наприклад, BBC, CNN) та знайти найбільш пов’язані сторінки.
3	Дослідити структуру Вікіпедії, обравши одну статтю та аналізуючи її внутрішні посилання.
4	Побудувати граф взаємозв’язку сторінок GitHub-репозиторію та знайти найбільш популярні посилання.
5	Дослідити граф посилань інтернет-магазину (наприклад, Amazon) та знайти найбільш популярні категорії товарів.
6	Виконати аналіз внутрішніх посилань блогу (наприклад, Medium) та визначити найбільш цитовані статті.
7	Побудувати граф сторінок місцевого новинного сайту та визначити його структуру.
8	Використовуючи API соціальних мереж, побудувати граф підписників Twitter-акаунту та знайти найвпливовіших користувачів.
9	Проаналізувати граф посилань сторінок YouTube-каналу (наприклад, через опис відео) та знайти найбільш згадувані сайти.
10	Виконати аналіз сайту, що містить відкриті дані (наприклад, data.gov), та знайти найбільш важливі категорії.

Методичні вказівки до виконання практичного заняття № 4

1. Збір даних про структуру сайту

```python
import requests
from bs4 import BeautifulSoup
import networkx as nx
import matplotlib.pyplot as plt

def get_links(url, max_links=5):
    """Отримує список перших max_links посилань зі сторінки"""
    try:
        response = requests.get(url, timeout=5)
        response.raise_for_status()  # Перевіряємо статус відповіді
        soup = BeautifulSoup(response.text, 'html.parser')

        links = set()
        for a in soup.find_all('a', href=True):
            link = a['href']
            if link.startswith('http'):  # Враховуємо лише повні URL
                links.add(link)
            if len(links) >= max_links:
                break
        return list(links)
    except requests.RequestException:
        return []
```

Рисунок 4.1 – Приклад збору даних про структуру сайту.

1.1 **Імпорт бібліотек**:

- **requests** – використовується для відправки HTTP-запитів на веб-сайти.
- **BeautifulSoup (з bs4)** – допомагає парсити HTML-код сторінки та знаходити посилання.
- **networkx** – дозволяє створювати, аналізувати та візуалізувати графи.
- **matplotlib.pyplot** – використовується для візуалізації графа.

1.2 **Функція для отримання посилань зі сторінки**:

- Виконує HTTP-запит до сторінки.
- Парсить HTML-код та шукає всі теги **<a>** (гіперпосилання).
- Відбирає лише перші **max_links** посилань (щоб не витрачати багато ресурсів).

- Фільтрує лише абсолютні **URL (href, що починається з "http")**.

(для аналізу внутрішньої структури сайту можна використовувати https://{Назва сайту})

2. Побудова графа у NetworkX

```python
def build_web_graph(start_url, max_depth=2, max_main_links=5, max_sub_links=5):
    """Будує граф структури веб-сайту з обмеженням глибини та кількості посилань"""
    G = nx.DiGraph()
    queue = [(start_url, 0)]  # Черга сторінок для обробки (URL, рівень глибини)
    visited = set()  # Відстежуємо вже відвідані сторінки

    while queue:
        url, depth = queue.pop(0)
        if url in visited or depth > max_depth:
            continue

        visited.add(url)
        G.add_node(url)

        # Визначаємо ліміт посилань залежно від рівня
        link_limit = max_main_links if depth == 0 else max_sub_links
        links = get_links(url, max_links=link_limit)

        for link in links:
            G.add_edge(url, link)
            queue.append((link, depth + 1))

    return G
```

Рисунок 4.2 – Приклад функції побудови графа

- **G = nx.DiGraph()** – створює орієнтований граф, оскільки посилання мають напрямок.
- Використовуємо **queue** для пошуку в ширину (BFS):
  - Спочатку додаємо головну сторінку з рівнем 0.
  - Додаємо у граф тільки 5 перших посилань.
  - Для кожного знайденого посилання переходимо глибше (максимум 2 рівня) та беремо по 5 посилань.
- **G.add_edge(url, link)** – додає зв'язок між вузлами (звідки й куди веде посилання).

3. Аналіз та візуалізація

```
# Головна сторінка сайту для аналізу
start_url = "https://www.rada.gov.ua/"

# Побудова графа з обмеженням: 5 посилань на головній, 5 на кожній наступній сторінці
web_graph = build_web_graph(start_url, max_depth=2, max_main_links=5, max_sub_links=5)

# Візуалізація графа
plt.figure(figsize=(45, 15))
nx.draw(web_graph, with_labels=True, node_color="lightblue", edge_color="gray", font_size=8, node_size=2000)
plt.show()

# Виведення базового аналізу
print("Кількість вузлів:", web_graph.number_of_nodes())
print("Кількість ребер:", web_graph.number_of_edges())
print("PageRank:", nx.pagerank(web_graph))
```

Рисунок 4.3 – Приклад коду аналізу та візуалізації.

3.1 Виконання коду

- Визначаємо стартову сторінку ("https://www.rada.gov.ua/").
- Запускаємо аналіз та побудову графа з обмеженнями:
  - 5 посилань з головної сторінки.
  - 5 посилань з кожної наступної сторінки.
  - Максимальна глибина аналізу = 2 рівні.

3.2 Візуалізація графа:

- Малюємо граф у вигляді мережі сторінок.
- Вузли – сторінки, ребра – посилання між ними.
- Використовуємо **networkx.draw()** для побудови графа.

3.3 Аналіз отриманої структури.

- **web_graph.number_of_nodes()** – кількість знайдених сторінок.
- **web_graph.number_of_edges()** – кількість посилань між сторінками.
- **nx.pagerank(web_graph)** – обчислює PageRank (метрика важливості сторінок, як у Google).

4. Результати роботи

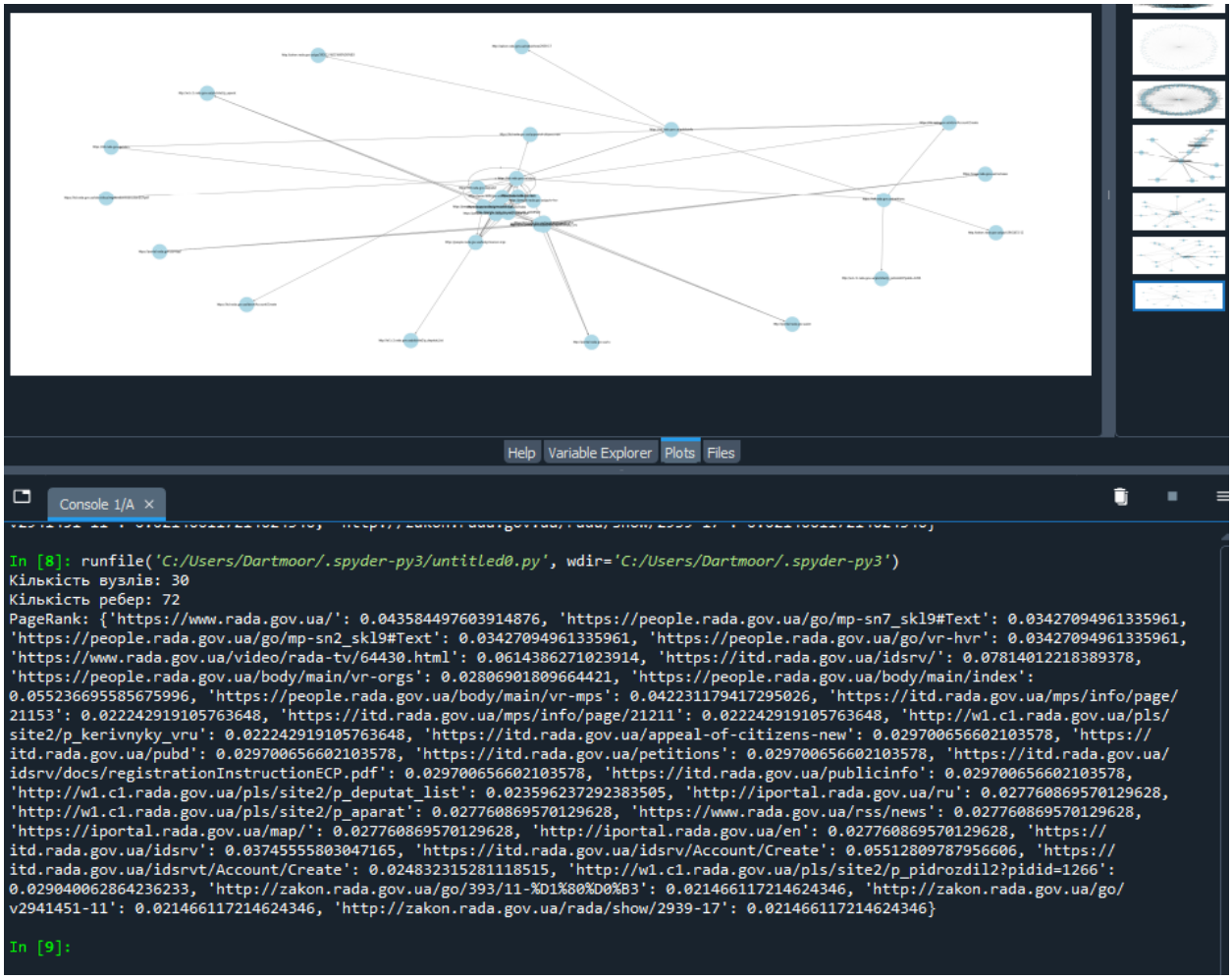


Рисунок 4.4 – Приклад результатів роботи додатку.

Контрольні питання:

1. Що таке граф у контексті аналізу Web-структури?
2. Які основні типи графів використовуються для представлення Web-структури?
3. Що означають вузли та ребра у графі Web-структури?
4. Які методи обходу графів використовуються для аналізу Web-сайтів?
5. Яка роль HTTP-запитів у процесі збору Web-даних?
6. Для чого використовується бібліотека BeautifulSoup у Python?

7. Як можна обмежити кількість отриманих посилань при аналізі сайту?

8. Яку інформацію можна отримати з Web-графа?

9. Що таке PageRank, і як він використовується для аналізу важливості сторінок?

10. Як можна оцінити центральність вузла у Web-графі?

11. Як можна візуалізувати граф Web-структури?

12. Чому важливо виключати дублікатні або некоректні посилання при побудові графа?

13. Які методи можна використати для оптимізації обходу Web-графа?

14. Як можна зробити аналіз Web-структури більш ефективним за допомогою асинхронного програмування?

15. Які є практичні застосування аналізу Web-графів у реальних проектах?

Література: [22-27]

Практичне заняття №5

«Машинне навчання для Web-даних»

Мета роботи:

Ознайомлення з методами машинного навчання для аналізу Web-даних, отриманих із реальних онлайн-сервісів. Використання алгоритмів класифікації, кластеризації та регресії для обробки текстових і числових даних. Розвиток навичок роботи з API та парсингом даних із веб-ресурсів, їхньою попередньою обробкою та підготовкою для моделей машинного навчання.

Завдання:

- Завантажити дані з реального веб-сервісу через API або виконати веб-скрейпінг.

- Попередньо обробити дані, очистити від шуму, нормалізувати, заповнити пропущені значення.

- Перетворити текстові або числові дані у формат, придатний для машинного навчання.

- Використати один із методів машинного навчання (класифікація, кластеризація або регресія) для аналізу даних.

- Оцінити якість моделі за допомогою відповідних метрик (точність, повнота, F1-score, MSE тощо).

- Візуалізувати результати у вигляді графіків або таблиць.

Вимоги до виконання:

1. Мова програмування - Python.
2. Бібліотеки:

- Для роботи з даними: pandas, numpy.

- Для машинного навчання: scikit-learn, tensorflow або pytorch (за бажанням).

- Для текстової обробки: nltk, spaCy, TfidfVectorizer.

- Для отримання даних: requests, BeautifulSoup, tweepy (Twitter API), openai (GPT API).

- Для візуалізації: matplotlib, seaborn, plotly.

3. Джерела даних:

- API сервіси (Twitter API, Reddit API, Google Trends, Hacker News API).

- Парсинг веб-сторінок (BeautifulSoup, Scrapy).

- Готові набори даних із Kaggle, UCI ML Repository, Hugging Face Datasets.

Вимоги до звіту:

1. Мета, завдання та варіант.
2. Опис процесу розробки додатку. (Кожен етап повинен бути описаний)
3. Опис процесу тестування роботи розробленого додатку. (Кожен скріншот повинен містити опис та підпис.)
4. Код програми та приклад результатів
5. Висновки.

Таблиця 5.1 – Варіанти завдань до практичного заняття 5

№	Завдання
1	**Аналіз відгуків про товари на eBay (eBay API).** Зібрати відгуки про популярні товари, знайти основні проблеми покупців. Джерело даних: eBay Finding API. Методи: NLP, Sentiment Analysis, Topic Modeling. Доступ - Безкоштовний API-ключ, до 5000 запитів на день.
2	**Прогнозування популярності статей (Hacker News API).** Отримати заголовки та метрики популярності 2000 статей, передбачити, які з них наберуть понад 500 лайків. Джерело даних: Hacker News API. Методи: Logistic Regression, Random Forest, XGBoost. Доступ - Відкрите API, без обмежень.
3	**Аналіз тональності коментарів на Reddit (Reddit API).** Завантажити коментарі до топових постів у певному subreddit, визначити їхню тональність (позитивна/негативна/нейтральна). Джерело даних: Reddit API (praw). Методи: NLP, Sentiment Analysis (VADER, BERT). Доступ - Потребує безкоштовного ключа, реєстрація в Reddit.
4	**Аналіз коментарів на YouTube (YouTube API).** Зібрати 1000 коментарів під популярними відео та визначити їхню тональність. Джерело даних - YouTube Data API. Методи: NLP, TF-IDF, Sentiment Analysis. Доступ - Потрібен API-ключ Google Cloud, безкоштовні ліміти.
5	**Кластеризація оголошень на OLX (BeautifulSoup, Scrapy).** Отримати дані про 1000 оголошень, згрупувати їх за тематикою. Джерело даних - Парсинг OLX або іншого маркетплейсу. Методи: TF-IDF, K-Means, Latent Dirichlet Allocation (LDA). Доступ - Парсинг без API, потребує requests, BeautifulSoup, Scrapy.
6	**Прогнозування цін на авіаквитки (Alternative Airlines API).** Зібрати дані про ціни на авіаквитки за останній місяць і передбачити зміну цін. Джерело даних - Alternative Airlines API. Методи: Time Series Forecasting (LSTM, XGBoost). Доступ - Безкоштовна версія API з лімітами.

7	**Кластеризація новинних статей (NewsAPI.org).** Завантажити 5000 новинних статей, згрупувати їх за тематикою. Джерело даних - NewsAPI.org. Методи: TF-IDF, K-Means, Latent Dirichlet Allocation (LDA). Доступ - Безкоштовна версія обмежена за кількістю запитів.
8	**Прогнозування трендових тем у Twitter (Twitter API v2, Academic Access).** Зібрати трендові хештеги за останній місяць і передбачити майбутні популярні теми. Джерело даних - Twitter API v2. Методи: Time Series Forecasting (ARIMA, LSTM). Доступ - Безкоштовний рівень має ліміти, для академічного доступу потрібна заявка.
9	**Кластеризація користувачів GitHub за активністю (GitHub API).** Отримати профілі активних користувачів за останні 30 днів і згрупувати їх за активністю. Джерело даних: GitHub API. Методи: K-Means, DBSCAN, PCA. Доступ - Безкоштовний токен GitHub, до 5000 запитів на годину.
10	**Класифікація відгуків про фільми (OMDB API + IMDb Dataset).** Завантажити відгуки про 1000 фільмів, класифікувати їх за тональністю (позитивні/негативні). Джерело даних - OMDB API + IMDb Open Dataset. Методи: NLP, TF-IDF, Logistic Regression, Naïve Bayes. Доступ - OMDB API потребує безкоштовного ключа, IMDb Open Dataset – вільний доступ.

Методичні вказівки до виконання практичного заняття № 5

Завдання - Аналіз тональності коментарів із Twitter (класифікація тексту)

У цьому прикладі:

- Отримаємо твітти за допомогою Twitter API.
- Попередньо обробимо текст: видалимо зайві символи, токенізуємо, приведемо до нижнього регістру.
- Перетворимо дані в числовий формат за допомогою TF-IDF.
- Навчимо модель на готових даних (позитивні/негативні коментарі).
- Оцінимо якість моделі та візуалізуємо результати.

1. Отримання даних з Twitter API

Спочатку потрібно отримати доступ до Twitter API (https://developer.x.com/) та створити облікові дані (API Key, Secret Key, Access Token або використати Bearer Token).

Встановлюємо бібліотеку для роботи з API:

```
In [1]: pip install tweepy
Collecting tweepy
  Downloading tweepy-4.15.0-py3-none-any.whl.metadata (4.1 kB)
Collecting oauthlib<4,>=3.2.0 (from tweepy)
  Downloading oauthlib-3.2.2-py3-none-any.whl.metadata (7.5 kB)
Requirement already satisfied: requests<3,>=2.27.0 in c:
\users\dartmoor\anaconda3\lib\site-packages (from tweepy) (2.32.2)
Collecting requests-oauthlib<3,>=1.2.0 (from tweepy)
  Downloading requests_oauthlib-2.0.0-py2.py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in c:
\users\dartmoor\anaconda3\lib\site-packages (from requests<3,>=2.27.0-
>tweepy) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:
\users\dartmoor\anaconda3\lib\site-packages (from requests<3,>=2.27.0-
>tweepy) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:
\users\dartmoor\anaconda3\lib\site-packages (from requests<3,>=2.27.0-
>tweepy) (2.2.2)
Requirement already satisfied: certifi>=2017.4.17 in c:
\users\dartmoor\anaconda3\lib\site-packages (from requests<3,>=2.27.0-
>tweepy) (2025.1.31)
Downloading tweepy-4.15.0-py3-none-any.whl (99 kB)
Downloading oauthlib-3.2.2-py3-none-any.whl (151 kB)
Downloading requests_oauthlib-2.0.0-py2.py3-none-any.whl (24 kB)
Installing collected packages: oauthlib, requests-oauthlib, tweepy
Successfully installed oauthlib-3.2.2 requests-oauthlib-2.0.0 tweepy-4.15.0
Note: you may need to restart the kernel to use updated packages.
```

Рисунок 5.1 – Приклад встановлення бібліотеки.

Код для завантаження та збереження твітів:

```python
import tweepy
import json
import os

# API-ключі (замініть своїми)
BEARER_TOKEN = "your_bearer_token"

# Авторизація
client = tweepy.Client(bearer_token=BEARER_TOKEN)

# Параметри запиту
QUERY = "machine learning -is:retweet lang:en"
MAX_TWEETS = 100
OUTPUT_FILE = "tweets.json"

# Функція для збереження твітів у файл
def save_to_file(data, filename):
    if os.path.exists(filename):
        with open(filename, "r", encoding="utf-8") as f:
            existing_data = json.load(f)
    else:
        existing_data = []

    existing_data.extend(data)

    with open(filename, "w", encoding="utf-8") as f:
        json.dump(existing_data, f, indent=4, ensure_ascii=False)

# Отримання твітів
tweets = []
response = client.search_recent_tweets(query=QUERY, tweet_fields=["created_at", "text", "author_id"], max_results=MAX_TWEETS)

if response.data:
    for tweet in response.data:
        tweets.append({
            "id": tweet.id,
            "author_id": tweet.author_id,
            "created_at": tweet.created_at.isoformat(),
            "text": tweet.text
        })

    # Збереження у файл
    save_to_file(tweets, OUTPUT_FILE)
    print(f"Збережено {len(tweets)} твітів у {OUTPUT_FILE}")
else:
    print("Не вдалося отримати твіти.")
```

Рисунок 5.2 – Приклад завантаження та збереження твітів.

В результаті маємо файл типу ***Json***, який містить 100 твітів присвячених машинному навчанню. (**X** має обмеження в 100 твітів на день, при використанні безкоштовної версії).

2. Завантаження та перегляд даних.

Спочатку потрібно завантажити файл tweets.json і переглянути його вміст.

```python
import json

# Завантажуємо дані з файлу
with open("tweets.json", "r", encoding="utf-8") as f:
    tweets = json.load(f)

# Виведемо перші 3 твіти для перегляду
for tweet in tweets[:3]:
    print(json.dumps(tweet, indent=4, ensure_ascii=False))
```

Рисунок 5.3 – Приклад завантаження та перегляду даних.

Відкриваємо файл tweets.json та завантажуємо дані. Виводимо три перші твіти у форматі JSON, щоб зрозуміти їхню структуру.

3. Попередня обробка тексту твітів.

Перед обробкою текст необхідно очистити:

- Видалити посилання, згадки (@user), хештеги (#topic).
- Звести всі слова до нижнього регістру.
- Видалити зайві пробіли та спецсимволи.

```python
import re

def clean_tweet(text):
    text = re.sub(r"http\S+|www\S+", "", text)  # Видаляємо URL
    text = re.sub(r"@\w+", "", text)  # Видаляємо згадки (@user)
    text = re.sub(r"#\w+", "", text)  # Видаляємо хештеги (#topic)
    text = re.sub(r"[^\w\s]", "", text)  # Видаляємо спецсимволи
    return text.lower().strip()

# Додаємо очищені тексти у новий список
cleaned_tweets = [clean_tweet(tweet["text"]) for tweet in tweets]

# Виведемо перші 3 очищені твіти
print(cleaned_tweets[:3])
```

Рисунок 5.4 – Приклад коду попередньої обробки тексту твітів.

Видаляємо URL-адреси (http://, www.). Видаляємо згадки (@username). Видаляємо хештеги (#hashtag). Видаляємо спецсимволи (наприклад, !@#$%). Перетворюємо текст у нижній регістр і прибираємо зайві пробіли.

4. Аналіз тональності твітів (Sentiment Analysis)

Визначимо, чи є твіт позитивним, негативним або нейтральним, використовуючи VADER (метод Sentiment Analysis).

```python
from nltk.sentiment import SentimentIntensityAnalyzer
import nltk

nltk.download("vader_lexicon")

# Ініціалізація аналізатора
sia = SentimentIntensityAnalyzer()

# Аналізуємо тональність кожного твіта
sentiments = []
for tweet in cleaned_tweets:
    score = sia.polarity_scores(tweet)
    sentiment = "positive" if score["compound"] > 0.05 else "negative" if score["compound"] < -0.05 else "neutral"
    sentiments.append({"tweet": tweet, "sentiment": sentiment})

# Виведемо результати для перших 5 твітів
for item in sentiments[:5]:
    print(f"Текст: {item['tweet']}\nТональність: {item['sentiment']}\n")
```

Рисунок 5.5 – Приклад коду аналізу тональності твітів.

VADER (від NLTK) визначає емоційний тон тексту (від -1 до +1).

Класифікація:

- `compound > 0.05` → `позитивний.`
- `compound < −0.05` → `негативний.`
- `інакше` → `нейтральний.`

5. Візуалізація результатів.

Побудуємо графік розподілу тональності твітів.

```
import matplotlib.pyplot as plt
from collections import Counter

# Підраховуємо кількість твітів у кожній категорії
sentiment_counts = Counter([item["sentiment"] for item in sentiments])

# Відображаємо графік
plt.bar(sentiment_counts.keys(), sentiment_counts.values(), color=["green", "red", "gray"])
plt.xlabel("Категорія")
plt.ylabel("Кількість твітів")
plt.title("Розподіл тональності твітів")
plt.show()
```

Рисунок 5.6 – Приклад коду аналізу тональності твітів.

Рахуємо кількість позитивних, негативних та нейтральних твітів. Будуємо стовпчикову діаграму.

6. Збереження результатів у новий JSON-файл.

Остаточно збережемо твіт + його тональність у tweets_sentiment.json.

```
# Збереження у файл
with open("tweets_sentiment.json", "w", encoding="utf-8") as f:
    json.dump(sentiments, f, indent=4, ensure_ascii=False)

print("Файл tweets_sentiment.json збережено!")
```

Рисунок 5.7 – Приклад коду збереження результатів.

У tweets_sentiment.json зберігаємо твіти разом із їхньою тональністю. Далі цей файл можна використовувати для подальшого аналізу або навчання моделі.

7. Результати роботи

```
In [2]: runfile('C:/Users/Dartmoor/.spyder-py3/untitled2.py', wdir='C:/Users/Dartmoor/.spyder-py3')
{
    "id": 1906430024791568455,
    "author_id": 1454184482412564481,
    "created_at": "2025-03-30T19:35:12+00:00",
    "text": "@RandallKanna Can machine learning ML and large language modules LLM work without AI...think on it before answering"
}
{
    "id": 1906430024644784515,
    "author_id": 1425384708,
    "created_at": "2025-03-30T19:35:12+00:00",
    "text": "there's nothing about machine learning that should necessitate anyone to know about hugging face\n\nthe end-to-end core concepts and
implementation of ml have nothing to do with HF\n\nfor a college professor to know about HF is simply keeping up with trends and nothing more
https://t.co/Nad2XbCQAo"
}
{
    "id": 1906429247520170080,
    "author_id": 1880379530986356736,
    "created_at": "2025-03-30T19:32:07+00:00",
    "text": "Inferium AI redefines innovation, blending machine learning with intuitive design. Powering industries with smart solutions, it
transforms complexity into simplicity, driving the future of intelligence. Your gateway to tomorrow! @InferiumAI"
}
['can machine learning ml and large language modules llm work without aithink on it before answering', 'theres nothing about machine learning
that should necessitate anyone to know about hugging face\n\nthe endtoend core concepts and implementation of ml have nothing to do with
hf\n\nfor a college professor to know about hf is simply keeping up with trends and nothing more', 'inferium ai redefines innovation blending
machine learning with intuitive design powering industries with smart solutions it transforms complexity into simplicity driving the future of
intelligence your gateway to tomorrow']
Текст: can machine learning ml and large language modules llm work without aithink on it before answering
Тональність: neutral

Текст: theres nothing about machine learning that should necessitate anyone to know about hugging face

the endtoend core concepts and implementation of ml have nothing to do with hf

for a college professor to know about hf is simply keeping up with trends and nothing more
Тональність: positive

Текст: inferium ai redefines innovation blending machine learning with intuitive design powering industries with smart solutions it transforms
complexity into simplicity driving the future of intelligence your gateway to tomorrow
Тональність: positive

Текст: 4 realworld applications  ai amp machine learning  cheaper amp scalable model training  gaming amp metaverse  decentralized rendering amp
compute power
Тональність: neutral

Текст: aidid artificial intelligence  decentralized identity

is a selfsovereign identity framework that integrates aidriven identity verification with decentralized block chains using machine learning and
biometric encryption to ensure security without exposing personal data
Тональність: positive

[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\Dartmoor\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Рисунок 5.8 – Приклад результатів у консолі.

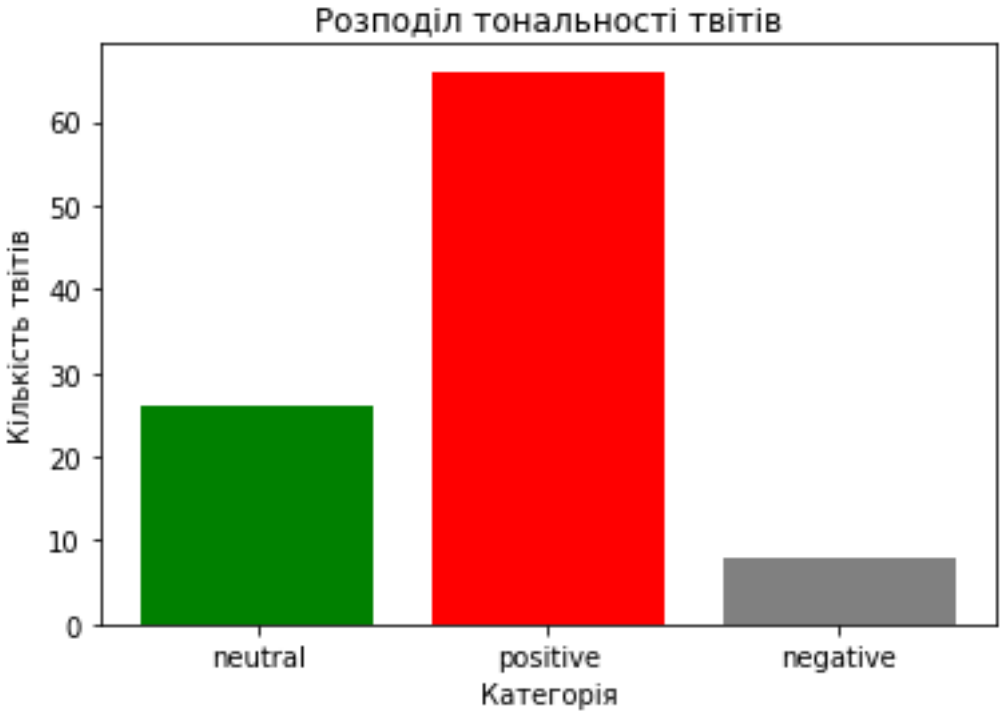
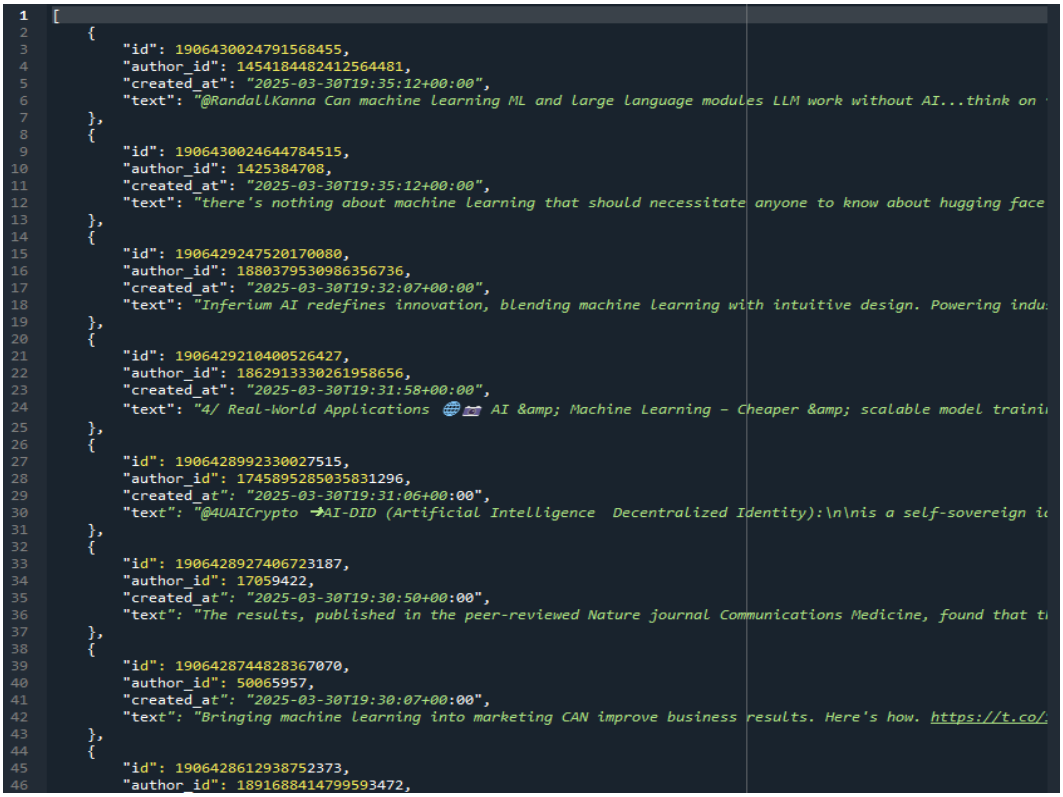
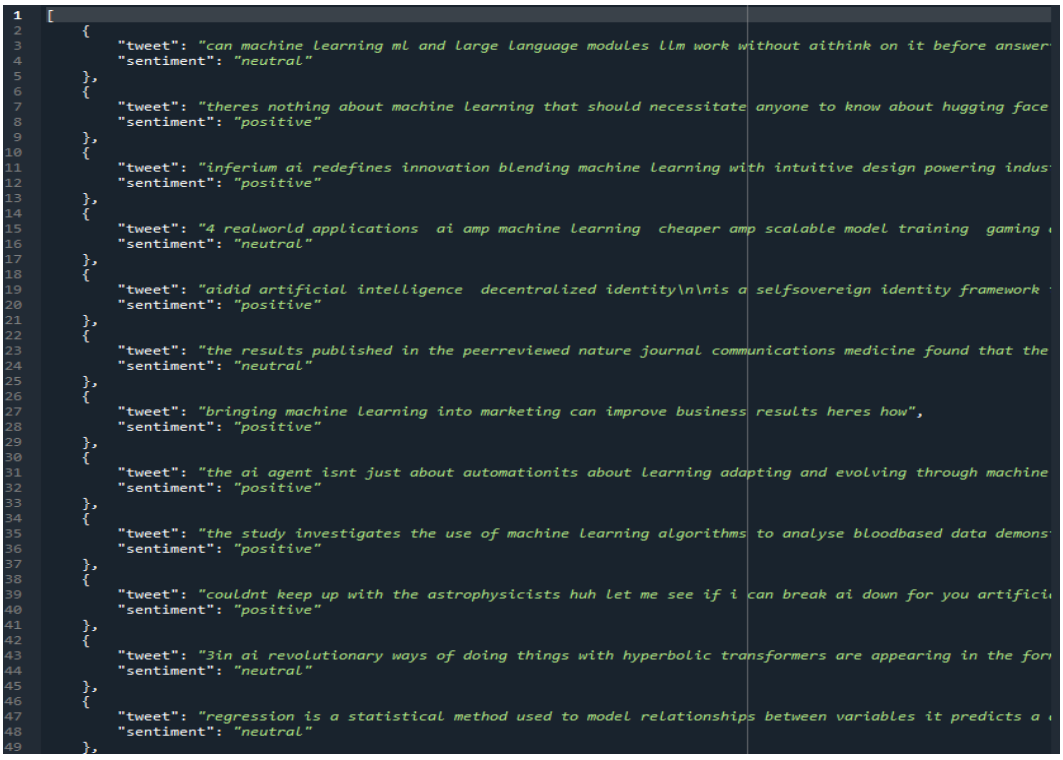


Рисунок 5.9 – Приклад результатів у вигляді графіку.



```
[
    {
        "id": 1906430024791568455,
        "author_id": 1454184482412564481,
        "created_at": "2025-03-30T19:35:12+00:00",
        "text": "@RandallKanna Can machine learning ML and large language modules LLM work without AI...think on
    },
    {
        "id": 1906430024644784515,
        "author_id": 1425384708,
        "created_at": "2025-03-30T19:35:12+00:00",
        "text": "there's nothing about machine learning that should necessitate anyone to know about hugging face
    },
    {
        "id": 1906429247520170080,
        "author_id": 1880379530986356736,
        "created_at": "2025-03-30T19:32:07+00:00",
        "text": "Inferium AI redefines innovation, blending machine learning with intuitive design. Powering indu
    },
    {
        "id": 1906429210400526427,
        "author_id": 1862913330261958656,
        "created_at": "2025-03-30T19:31:58+00:00",
        "text": "4/ Real-World Applications 🌍🏭 AI &amp;amp; Machine Learning – Cheaper &amp;amp; scalable model traini
    },
    {
        "id": 1906428992330027515,
        "author_id": 1745895285035831296,
        "created_at": "2025-03-30T19:31:06+00:00",
        "text": "@4UAICrypto ➔AI-DID (Artificial Intelligence  Decentralized Identity):\n\nis a self-sovereign i
    },
    {
        "id": 1906428927406723187,
        "author_id": 17059422,
        "created_at": "2025-03-30T19:30:50+00:00",
        "text": "The results, published in the peer-reviewed Nature journal Communications Medicine, found that t
    },
    {
        "id": 1906428744828367070,
        "author_id": 50065957,
        "created_at": "2025-03-30T19:30:07+00:00",
        "text": "Bringing machine learning into marketing CAN improve business results. Here's how. https://t.co/
    },
    {
        "id": 1906428612938752373,
        "author_id": 1891688414799593472,
```


Рисунок 5.10 – Приклад файлу tweets.json.



```
[
    {
        "tweet": "can machine learning ml and large language modules llm work without aithink on it before answer
        "sentiment": "neutral"
    },
    {
        "tweet": "theres nothing about machine learning that should necessitate anyone to know about hugging face
        "sentiment": "positive"
    },
    {
        "tweet": "inferium ai redefines innovation blending machine learning with intuitive design powering indus
        "sentiment": "positive"
    },
    {
        "tweet": "4 realworld applications  ai amp machine learning  cheaper amp scalable model training  gaming
        "sentiment": "neutral"
    },
    {
        "tweet": "aidid artificial intelligence  decentralized identity\n\nis a selfsovereign identity framework
        "sentiment": "positive"
    },
    {
        "tweet": "the results published in the peerreviewed nature journal communications medicine found that the
        "sentiment": "neutral"
    },
    {
        "tweet": "bringing machine learning into marketing can improve business results heres how",
        "sentiment": "positive"
    },
    {
        "tweet": "the ai agent isnt just about automationits about learning adapting and evolving through machine
        "sentiment": "positive"
    },
    {
        "tweet": "the study investigates the use of machine learning algorithms to analyse bloodbased data demons
        "sentiment": "positive"
    },
    {
        "tweet": "couldnt keep up with the astrophysicists huh let me see if i can break ai down for you artifici
        "sentiment": "positive"
    },
    {
        "tweet": "3in ai revolutionary ways of doing things with hyperbolic transformers are appearing in the for
        "sentiment": "neutral"
    },
    {
        "tweet": "regression is a statistical method used to model relationships between variables it predicts a
        "sentiment": "neutral"
    },
```


Рисунок 5.11 – Приклад файлу tweets_sentiment.json

Лістинг.

Файл – Tweets_parsing.py

```
import tweepy
import json
import os

# API-ключі (замініть своїми)
BEARER_TOKEN = "your_bearer_token"

# Авторизація
client = tweepy.Client(bearer_token=BEARER_TOKEN)

# Параметри запиту
QUERY = "machine learning -is:retweet lang:en"
MAX_TWEETS = 100
OUTPUT_FILE = "tweets.json"

# Функція для збереження твітів у файл
def save_to_file(data, filename):
    if os.path.exists(filename):
        with open(filename, "r", encoding="utf-8") as f:
            existing_data = json.load(f)
    else:
        existing_data = []

    existing_data.extend(data)

    with open(filename, "w", encoding="utf-8") as f:
        json.dump(existing_data, f, indent=4, ensure_ascii=False)

# Отримання твітів
tweets = []
response = client.search_recent_tweets(query=QUERY, tweet_fields=["created_at", "text",
"author_id"], max_results=MAX_TWEETS)

if response.data:
    for tweet in response.data:
        tweets.append({
            "id": tweet.id,
            "author_id": tweet.author_id,
            "created_at": tweet.created_at.isoformat(),
            "text": tweet.text
        })

    # Збереження у файл
    save_to_file(tweets, OUTPUT_FILE)
    print(f"Збережено {len(tweets)} твітів у {OUTPUT_FILE}")
else:
    print("Не вдалося отримати твіти.")
```

Файл – Tweets_ analysis.py

```
import json

# Завантажуємо дані з файлу
with open("tweets.json", "r", encoding="utf-8") as f:
    tweets = json.load(f)

# Виведемо перші 3 твіти для перегляду
for tweet in tweets[:3]:
    print(json.dumps(tweet, indent=4, ensure_ascii=False))

import re

def clean_tweet(text):
    text = re.sub(r"http\S+|www\S+", "", text)  # Видаляємо URL
    text = re.sub(r"@\w+", "", text)  # Видаляємо згадки (@user)
    text = re.sub(r"#\w+", "", text)  # Видаляємо хештеги (#topic)
    text = re.sub(r"[^\w\s]", "", text)  # Видаляємо спецсимволи
    return text.lower().strip()

# Додаємо очищені тексти у новий список
cleaned_tweets = [clean_tweet(tweet["text"]) for tweet in tweets]

# Виведемо перші 3 очищені твіти
print(cleaned_tweets[:3])

from nltk.sentiment import SentimentIntensityAnalyzer
import nltk

nltk.download("vader_lexicon")

# Ініціалізація аналізатора
sia = SentimentIntensityAnalyzer()

# Аналізуємо тональність кожного твіта
sentiments = []
for tweet in cleaned_tweets:
    score = sia.polarity_scores(tweet)
    sentiment = "positive" if score["compound"] > 0.05 else "negative" if score["compound"]
< -0.05 else "neutral"
    sentiments.append({"tweet": tweet, "sentiment": sentiment})

# Виведемо результати для перших 5 твітів
for item in sentiments[:5]:
    print(f"Текст: {item['tweet']}\nТональність: {item['sentiment']}\n")

import matplotlib.pyplot as plt
from collections import Counter

# Підраховуємо кількість твітів у кожній категорії
sentiment_counts = Counter([item["sentiment"] for item in sentiments])
```

```
# Відображаємо графік
plt.bar(sentiment_counts.keys(), sentiment_counts.values(), color=["green", "red", "gray"])
plt.xlabel("Категорія")
plt.ylabel("Кількість твітів")
plt.title("Розподіл тональності твітів")
plt.show()

# Збереження у файл
with open("tweets_sentiment.json", "w", encoding="utf-8") as f:
    json.dump(sentiments, f, indent=4, ensure_ascii=False)

print("Файл tweets_sentiment.json збережено!")
```

Контрольні питання:

1. Які API можна використовувати для отримання реальних Web-даних?
2. Які обмеження має безкоштовний доступ до Twitter API v2?
3. Які основні кроки передбачає попередня обробка тексту твітів?
4. Чому важливо видаляти URL-адреси, хештеги та згадки з тексту перед аналізом?
5. Що таке Sentiment Analysis і для чого він використовується?
6. Який метод аналізу тональності тексту було використано в роботі?
7. Яким чином бібліотека VADER визначає тональність тексту?
8. Які порогові значення використовуються для класифікації тональності у VADER?
9. Як можна візуалізувати результати аналізу тональності твітів?
10. Для чого використовують бібліотеку matplotlib у цьому завданні?
11. Чому важливо зберігати зібрані дані у файл перед наступним запуском?
12. Які можливі обмеження підходу до Sentiment Analysis, використаного у роботі?
13. Як можна покращити точність аналізу тональності твітів?

14. Які ще методи машинного навчання можна застосувати для аналізу Web-даних?

15. Як можна використовувати результати аналізу тональності для реальних застосунків?

Література: [1–2, 4–8, 10, 12, 15–20, 22]

Практичне заняття №6

«Робота з API сервісів»

Мета роботи: Ознайомлення з методами збору та аналізу трендів та соціальних сигналів із відкритих джерел, таких як Google Trends, Twitter Trends, YouTube Trending. Закріплення навичок виявлення популярних тем, сезонності та залежності між трендами.

Завдання:

Отримати дані про популярні запити, хештеги або відео з відкритих платформ, обробити їх, візуалізувати та зробити висновки щодо їхньої динаміки та значення. Завдання включає такі етапи:

- Збір даних. Отримання інформації про популярні запити/хештеги/відео за певний період.
- Обробка даних. Перетворення отриманих даних у зручний для аналізу формат (таблиця, графік, список).
- Аналіз трендів. Визначення закономірностей, піків популярності, сезонності, кореляції між трендами.
- Візуалізація результатів. Побудова графіків або діаграм для демонстрації динаміки змін.
- Формування висновків. Аналіз отриманих результатів та їхнє можливе практичне застосування.

Вимоги до виконання:

1. Мова програмування - Python.
2. Джерела даних: Дані мають бути отримані з відкритих джерел, таких як:

- Google Trends (популярні пошукові запити)

- Twitter Trends (популярні теги та теми)
- YouTube Trending (популярні відео)
- Інші джерела відкритих даних

3. Методи збору даних:
   - API сервіси (Google Trends API, Twitter API)
   - Scraping (збір інформації зі сторінок Google Trends, Twitter, YouTube за допомогою BeautifulSoup, Selenium)

Вимоги до звіту:

1. Мета, завдання та варіант.
2. Опис процесу розробки додатку. (Кожен етап повинен бути описаний)
3. Опис процесу тестування роботи розробленого додатку. (Кожен скріншот повинен містити опис та підпис.)
4. Код програми та приклад результатів
5. Висновки.

Таблиця 6.1 – Варіанти завдань до практичного заняття 6

№	Завдання
1	**Порівняння трендів у різних країнах.** Виберіть 3-5 популярних запитів (наприклад, "Bitcoin", "ChatGPT", "Tesla"). Використовуючи Google Trends, проаналізуйте їхню популярність у різних країнах. Побудуйте графік популярності цих запитів для кожної країни. Визначте, які регіони мають найбільший інтерес до обраної тематики.
2	**Аналіз сезонності пошукових запитів.** Виберіть 3-5 запитів, які можуть мати сезонний характер (наприклад, "Black Friday", "Summer Vacation", "Christmas Gifts"). Проаналізуйте їхню популярність за останні 5 років. Побудуйте графік, який показує сезонні піки популярності. Опишіть отримані закономірності.
3	**Аналіз впливу подій на популярність запитів.** Оберіть глобальну подію (наприклад, вихід нового iPhone, запуск ChatGPT 4.0, Олімпійські ігри). Визначте ключові пошукові запити, пов’язані з цією подією.

	Проаналізуйте, як змінилась їхня популярність до та після події. Побудуйте графік тренду та визначте, як довго тривав "ефект популярності".
4	**Порівняння технологічних трендів.** Оберіть 5 технологій для аналізу (наприклад, "Artificial Intelligence", "Blockchain", "Quantum Computing", "5G", "Cloud Computing"). Визначте, як змінювалась їхня популярність у пошукових запитах за останні 5 років. Побудуйте графіки змін популярності. Визначте, які технології є "зростаючими", а які – "спадаючими" трендами.
5	**Порівняння популярності брендів.** Оберіть 5 брендів для аналізу (наприклад, "Apple", "Samsung", "Tesla", "Microsoft", "Google"). Проаналізуйте їхню популярність у пошукових запитах за останні 5 років. Побудуйте графіки змін популярності брендів у часі. Визначте, які бренди демонструють зростання інтересу, а які – спад.
6	**Аналіз запитів у сфері здоров'я та спорту.** Виберіть 5 запитів, пов'язаних зі здоров'ям чи спортом (наприклад, "Yoga", "Meditation", "Intermittent Fasting", "Gym Workout", "Running"). Проаналізуйте, як змінювалась їхня популярність за останні 5 років. Визначте, які з них мають сезонний характер (наприклад, тренажерні зали можуть ставати популярнішими в січні). Побудуйте відповідні графіки.
7	**Аналіз трендів у сфері кіно та розваг.** Оберіть 5 фільмів, серіалів або франшиз (наприклад, "Marvel", "Harry Potter", "Stranger Things", "Barbie movie"). Проаналізуйте їхню популярність у Google Trends. Побудуйте графік змін популярності у часі. Визначте, які події (вихід нового фільму, серіалу) впливали на тренди.
8	**Порівняння трендів у різних мовах.** Виберіть один популярний запит (наприклад, "Artificial Intelligence"). Дослідіть його популярність у Google Trends на різних мовах (наприклад, "Штучний інтелект", "Inteligencia artificial", "Intelligence artificielle"). Побудуйте графіки популярності для кожної мовної версії. Зробіть висновки про популярність запиту у різних мовних спільнотах.
9	**Аналіз запитів у сфері фінансів**. Виберіть 5 фінансових запитів (наприклад, "Bitcoin", "Stock Market", "Inflation", "Recession", "Interest Rates"). Проаналізуйте їхню популярність за останні 5 років. Визначте, чи є зв'язок між популярністю цих запитів та глобальними економічними подіями. Побудуйте відповідні графіки.
10	**Аналіз трендів серед професій та навичок.** Виберіть 5 популярних професій або навичок для аналізу (наприклад, "Data Scientist", "Software Engineer", "AI Engineer", "Cybersecurity Specialist", "DevOps"). Використовуючи Google Trends, проаналізуйте їхню популярність за останні 5 років. Побудуйте графік трендів. Визначте, які навички стають більш затребуваними.

Методичні вказівки до виконання практичного заняття № 6

Завдання - Аналіз популярності запиту "AI" за останній рік

1. Отримання даних з Google Trends

Google Trends дозволяє отримувати інформацію про динаміку популярності певного пошукового запиту. Для автоматизованого збору даних використаємо бібліотеку pytrends.

2. Встановлюємо бібліотеку для роботи з Google Trends:

```
In [1]: pip install pytrends
Collecting pytrends
  Downloading pytrends-4.9.2-py3-none-any.whl.metadata (13 kB)
Requirement already satisfied: requests>=2.0 in c:\users\dartmoor\anaconda3\lib\site-packages (from pytrends) (2.32.2)
Requirement already satisfied: pandas>=0.25 in c:\users\dartmoor\anaconda3\lib\site-packages (from pytrends) (2.2.2)
Requirement already satisfied: lxml in c:\users\dartmoor\anaconda3\lib\site-packages (from pytrends) (5.2.1)
Requirement already satisfied: numpy>=1.26.0 in c:\users\dartmoor\anaconda3\lib\site-packages (from pandas>=0.25->pytrends) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\dartmoor\anaconda3\lib\site-packages (from pandas>=0.25->pytrends)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\dartmoor\anaconda3\lib\site-packages (from pandas>=0.25->pytrends) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\dartmoor\anaconda3\lib\site-packages (from pandas>=0.25->pytrends) (2023.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\dartmoor\anaconda3\lib\site-packages (from requests>=2.0->pytrends)
(2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\dartmoor\anaconda3\lib\site-packages (from requests>=2.0->pytrends) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\dartmoor\anaconda3\lib\site-packages (from requests>=2.0->pytrends) (2.2.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\dartmoor\anaconda3\lib\site-packages (from requests>=2.0->pytrends) (2025.1.31)
Requirement already satisfied: six>=1.5 in c:\users\dartmoor\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas>=0.25-
>pytrends) (1.16.0)
Downloading pytrends-4.9.2-py3-none-any.whl (15 kB)
Installing collected packages: pytrends
Successfully installed pytrends-4.9.2
Note: you may need to restart the kernel to use updated packages.
```

Рисунок 6.1 – Приклад встановлення бібліотеки.

3. Імпорт необхідних модулів:

```
import pandas as pd
import matplotlib.pyplot as plt
from pytrends.request import TrendReq
```

Рисунок 6.2 – Приклад підключення необхідних модулів.

4. Підключення до Google Trends API та отримання даних:

```
# Ініціалізація з'єднання
pytrends = TrendReq(hl='en-US', tz=360)

# Вибір ключових слів для аналізу
keywords = ["AI"]

# Завантаження даних за останній рік
for keyword in keywords:
    pytrends.build_payload([keyword], timeframe='today 12-m', geo='US')
    data = pytrends.interest_over_time()
    print(data)
    time.sleep(5+random.randrange(0, 10, 1))  # Затримка між запитами

# Видаляємо зайві колонки
data = data.drop(columns=['isPartial'])

# Вивід перших рядків
print(data.head())
```

Рисунок 6.3 – Приклад завантаження даних.

На цьому етапі ми отримали таблицю зі змінами популярності запиту AI за останні 12 місяців.

5. Побудова графіку змін популярності:

```
plt.figure(figsize=(12, 6))
plt.plot(data.index, data["AI"], color='blue', linestyle='-', marker='o', label="AI")
plt.xlabel("Дата")
plt.ylabel("Популярність")
plt.title("Динаміка популярності запиту 'AI' за останній рік")
plt.legend()
plt.grid()
plt.xticks(rotation=45)
plt.show()
```

Рисунок 6.4 – Приклад побудови графіку.

Отримуємо графік, на якому видно коливання популярності запиту AI протягом року (рис. 6.5). Наприклад, можна помітити, що популярність різко зросла у певні періоди (наприклад, коли з'явилися нові моделі штучного інтелекту).

Рисунок 6.5 – Графік коливання популярності запиту AI протягом року.

Після візуалізації можна зробити висновки:

- Запит "AI" мав піки популярності у періоди активного обговорення в ЗМІ.
- Найвищий рівень інтересу спостерігався у березні 2025, що може збігатися з виходом нового релізу ChatGPT або іншого AI-продукту.
- У літній період популярність запиту знизилася, що може бути пов’язано зі зменшенням активності користувачів у літній сезон.

Якщо потрібно порівняти кілька трендів (наприклад, "AI" vs. "Machine Learning"), можна розширити код (рис. 6).

```python
import pandas as pd
import time
import random
import matplotlib.pyplot as plt
from pytrends.request import TrendReq


pytrends = TrendReq(hl='en-US', tz=360)
keywords = ["AI", "Machine Learning"]

for keyword in keywords:
    pytrends.build_payload([keyword], timeframe='today 12-m', geo='US')
    data = pytrends.interest_over_time()
    if data.empty:
        print(f"Немає даних для {keyword}")
        continue
    print(data.columns)  # Перевіряємо назви колонок
    plt.plot(data.index, data.iloc[:, 0], marker='o', label=data.columns[0])
    time.sleep(5+random.randrange(0, 10, 1))  # Затримка між запитами

plt.legend()
plt.xticks(rotation=45)
plt.xlabel("Дата")
plt.ylabel("Інтерес")
plt.title("Тренди за 12 місяців")
plt.show()
```

Рисунок 6.6 – Приклад коду порівняння трендів "AI" vs. "Machine Learning".

Таким чином, ми можемо порівнювати популярність різних запитів, визначати взаємозв'язки та робити прогнози щодо майбутніх трендів (рис. 7).

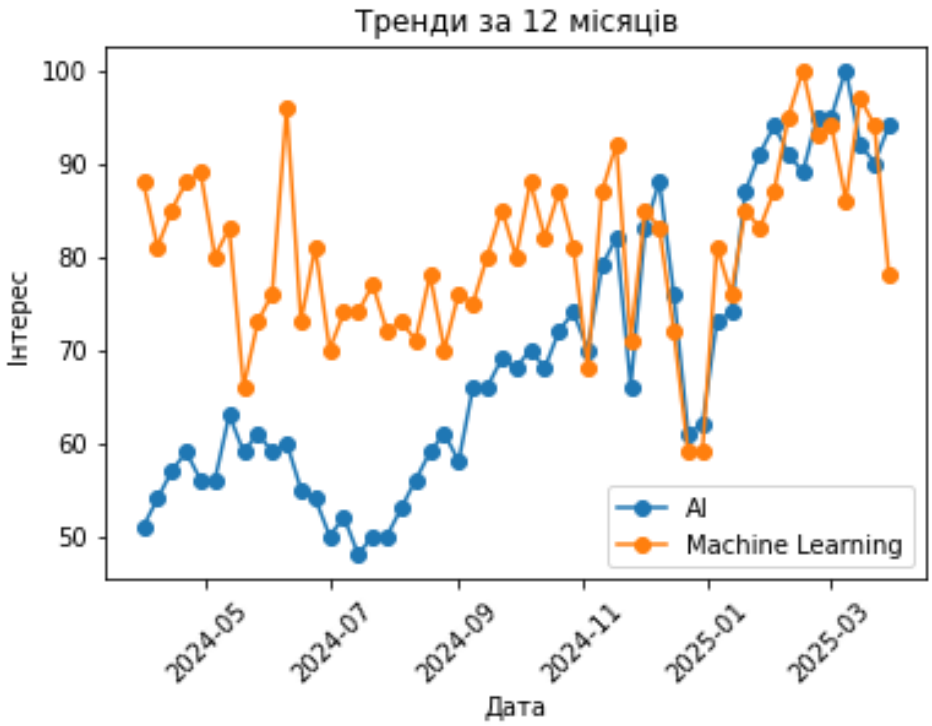


Рисунок 6.7 – Графік коливання трендів "AI" vs. "Machine Learning".

Лістинг.

```
import pandas as pd
import time
import random
import matplotlib.pyplot as plt
from pytrends.request import TrendReq

pytrends = TrendReq(hl='en-US', tz=360)
keywords = ["AI", "Machine Learning"]

for keyword in keywords:
    pytrends.build_payload([keyword], timeframe='today 12-m', geo='US')
    data = pytrends.interest_over_time()
    if data.empty:
        print(f"Немає даних для {keyword}")
        continue
    print(data.columns)  # Перевіряємо назви колонок
    plt.plot(data.index, data.iloc[:, 0], marker='o', label=data.columns[0])
    time.sleep(5+random.randrange(0, 10, 1))  # Затримка між запитами

plt.legend()
plt.xticks(rotation=45)
plt.xlabel("Дата")
plt.ylabel("Інтерес")
plt.title("Тренди за 12 місяців")
plt.show()
```

Контрольні питання:

1. Що таке веб-тренди? Які фактори впливають на їхнє формування?
2. Які основні метрики використовуються для аналізу популярності пошукових запитів у Google Trends?
3. Як визначити сезонність пошукових запитів? Які методи використовуються для її виявлення?
4. Як глобальні події можуть впливати на популярність певних запитів у пошукових системах?
5. Які переваги та обмеження використання Google Trends для аналізу веб-трендів?
6. Що таке порівняльний аналіз трендів? Як його можна використати в бізнесі чи маркетингу?
7. Як можна визначити "зростаючі" та "спадаючі" тренди за допомогою аналізу пошукових запитів?
8. Які існують альтернативи Google Trends для збору даних про популярність пошукових запитів?
9. Чому важливо враховувати географічний фактор при аналізі веб-трендів?
10. Як можна застосувати аналіз трендів для прогнозування майбутніх змін у певній галузі?
11. Що таке веб-тренди? Які фактори впливають на їхнє формування?
12. Які основні метрики використовуються для аналізу популярності пошукових запитів у Google Trends?
13. Як визначити сезонність пошукових запитів? Які методи використовуються для її виявлення?
14. Як глобальні події можуть впливати на популярність певних запитів у пошукових системах?

Література: [1–2, 6, 8–11, 15, 20, 22–25]

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Mitchell R. Web scraping with Python: Collecting data from the modern web / R. Mitchell. - 2nd ed. - O'Reilly Media, Inc., 2018.

2. Jarmul K. Python web scraping / K. Jarmul, R. Lawson. - 2nd ed. - Packt Publishing, 2017.

3. Sweigart A. Automate the boring stuff with Python: Practical programming for total beginners /A. Sweigart. - 2nd ed. - No Starch Press, 2019.

4. BeautifulSoup Documentation [Електронний ресурс]. – Режим доступу : https://www.crummy.com/software/BeautifulSoup/bs4/doc. - Назва з екрана.

5. Real Python – Web Scraping with BeautifulSoup [Електронний ресурс]. – Режим доступу : https://realpython.com/beautiful-soup-web-scraper-python. - Назва з екрана.

6. Scrapy Documentation [Електронний ресурс]. – Режим доступу : https://docs.scrapy.org/en/latest. - Назва з екрана.

7. Python Requests Documentation [Електронний ресурс]. – Режим доступу : https://requests.readthedocs.io. - Назва з екрана.

8. Towards Data Science – Web Scraping with Python [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com. - Назва з екрана.

9. Web Scraping Hub – A Beginner's Guide [Електронний ресурс]. – Режим доступу : https://www.webscrapinghub.com. - Назва з екрана.

10. Russell M. A. Mining the social web: Data mining Facebook, Twitter, LinkedIn, Instagram, GitHub, and more / M. A. Russell . - 3rd ed. - O'Reilly Media, 2019. - 568 p.

11. Kouzis-Loukas D. Mastering Scrapy / D. Kouzis-Loukas. - Packt Publishing, 2016. - 306 p.

12. Bird S. Natural language processing with Python: Analyzing text with the natural language toolkit / S. Bird, E. Klein, E. Loper, O'Reilly. - Media, 2009. 504 p.

13. Manning C. D. Foundations of statistical natural language processing / C. D. Manning, H. Schütze. - MIT Press, 1999. - 680 p.

14. Jurafsky D. Speech and language processing / D. Jurafsky, J. H. Martin. - 3rd ed. - Draft, 2023.

15. Kukich K. Text data mining and analysis: Practical methods, examples, and case studies using Python / K. Kukich, C. C. Aggarwal. - Springer, 2022. - 452 p.

16. Goldberg Y. Neural network methods for natural language processing / Y. Goldberg. - Morgan & Claypool Publishers, 2017. - 309 p.

17. NLTK Documentation [Електронний ресурс]. – Режим доступу: https://www.nltk.org. - Назва з екрана.

18. spaCy Documentation [Електронний ресурс]. – Режим доступу : https://spacy.io/api/doc. - Назва з екрана.

19. Fast.ai – NLP Course [Електронний ресурс]. – Режим доступу : https://course.fast.ai. - Назва з екрана.

20. Kaggle – Natural Language Processing [Електронний ресурс]. – Режим доступу : https://www.kaggle.com/models. - Назва з екрана.

21. Stanford NLP Group [Електронний ресурс]. – Режим доступу : https://nlp.stanford.edu. - Назва з екрана.

22. Leskovec J. Mining of massive datasets / J. Leskovec, A. Rajaraman, J. Ullman. - 2nd ed. - Cambridge University Press, 2014. - 580 p.

23. Barabási, A.-L. Network science / A.-L. Barabási. - Cambridge University Press, 2016. - 475 p.

24. Easley D. Networks, crowds, and markets: Reasoning about a highly connected world / D. Easley, J. Kleinberg. - Cambridge University Press, 2010. - 727 p.

25. Newman M. Networks: An introduction / M. Newman. - Oxford University Press, 2010. - 772 p.