Project Documentation: Ophthalmology Data Extraction

Contents

1 Overview

This project involves extracting question-answer pairs related to ophthalmology from multiple web pages and saving them in a CSV file. The script uses Python's `requests` library for web scraping and `BeautifulSoup` for parsing HTML content. It includes error handling, data validation, and ensures the extracted data is relevant to ophthalmology.

2 Requirements

- Python 3.x
- aiohttp library
- `requests` library
- `beautifulsoup4` library
- `csv` module (part of Python's standard library)

3 Setup Instructions

1. **Install Python**: Ensure Python 3.x is installed on your system. You can download it from python.org.
2. **Install Required Libraries**: Install the necessary Python libraries using pip:

```
pip install aiohttp beautifulsoup4 requests
```

4 Script Explanation

4.1 Fetching Web Pages

The `fetch_page` function retrieves the content of a given URL and handles potential request errors.

```
import requests
from requests.exceptions import RequestException

def fetch_page(url):
    headers = {
```

```
        "User-Agent": "Mozilla/5.0␣(Windows␣NT␣10.0;␣
            ↪ Win64;␣x64)␣AppleWebKit/537.36␣(KHTML,␣
            ↪ like␣Gecko)␣Chrome/58.0.3029.110␣
            ↪ Safari/537.3"
    }
    try:
        response = requests.get(url, headers=headers)
        response.raise_for_status()
        return response.content
    except RequestException as e:
        print(f"Request␣failed␣for␣{url}:␣{e}")
        return None
```

4.2 Parsing Question-Answer Pairs

The `parse_questions_answers` function parses the HTML content to extract question-answer pairs. It ensures that only relevant questions (related to ophthalmology) are included.

```
from bs4 import BeautifulSoup

def parse_questions_answers(soup):
    qa_pairs = []
    questions = soup.find_all('h1')
    for question in questions:
        answer = question.find_next('p')
        question_text = question.get_text(strip=True)
        answer_text = answer.get_text(strip=True) if
            ↪ answer else 'No␣answer␣found'
        if
            ↪ is_valid_ophthalmology_question(question_text):
             qa_pairs.append((question_text,
                ↪ answer_text))
    return qa_pairs

def is_valid_ophthalmology_question(question):
    ophthalmology_keywords = ['eye', 'vision',
        ↪ 'optometrist', 'ophthalmologist', 'eye␣
        ↪ health']
    return any(keyword.lower() in question.lower() for
        ↪ keyword in ophthalmology_keywords)
```

4.3 Saving Data to CSV

The `save_to_csv` function saves the extracted question-answer pairs to a CSV file.

```
import csv

def save_to_csv(data, filename):
    with open(filename, mode='w', newline='',
       ↪ encoding='utf-8') as file:
        writer = csv.writer(file)
        writer.writerow(["Question", "Answer"])
        writer.writerows(data)
```

4.4 Main Function

The `main` function coordinates the scraping process, iterating over multiple URLs and aggregating the results.

```
def main():
    urls = [
        "https://www.healthline.com/health/eye-health/optometrist-vs-oph
        # Add more URLs here
    ]

    all_qa_pairs = []

    for url in urls:
        page_content = fetch_page(url)
        if page_content:
            soup = BeautifulSoup(page_content,
               ↪ 'html.parser')
            qa_pairs = parse_questions_answers(soup)
            all_qa_pairs.extend(qa_pairs)

    save_to_csv(all_qa_pairs, "vision_health_qa.csv")
    print("Data␣extraction␣and␣saving␣completed.")

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

4.5 Main Function

Listing 1: Main Function

```
async def main(api_key=None):
    '''Main function to coordinate the scraping process
       ↪ and save the results.'''
    urls = [
        "https://www.healthline.com/health/eye-health/optometrist-vs-oph
        "https://www.healthdirect.gov.au/ophthalmologist",
        "https://www.webmd.com/eye-health/default.htm",
        "https://www.aao.org/eye-health",
        "https://www.med.unc.edu/ophth/for-patients/eye-diseases-and-dis
        "https://healthcare.utah.edu/moran/services",
        "https://www.cdc.gov/vision-health/about-eye-disorders/index.htm
        "https://medlineplus.gov/eyesandvision.html",
        "https://www.news-medical.net/condition/Eye-Health"
    ]

    async with aiohttp.ClientSession() as session:
        tasks = [fetch_page(session, url) for url in
           ↪ urls]
        pages_content = await asyncio.gather(*tasks)

        all_qa_pairs = []
        for content in pages_content:
            if content:
                soup = BeautifulSoup(content,
                   ↪ 'html.parser')
                qa_pairs =
                   ↪ parse_questions_answers(soup,
                   ↪ api_key=api_key)
                all_qa_pairs.extend(qa_pairs)

        save_to_csv(all_qa_pairs,
           ↪ "ophthalmology_qa.csv")
        print("Data␣extraction␣and␣saving␣completed.")

# Run the main function
api_key = 'your_openai_api_key_here'  # Replace with
   ↪ your actual OpenAI API key if available
asyncio.run(main(api_key=api_key))
```

5 Instructions for the Intern

1. **Understand the Current Code**: Thoroughly read and understand the existing code. Pay special attention to the functions and how they interact.

2. **Add More URLs**: Expand the `urls` list with additional URLs that contain relevant ophthalmology information.

3. **Improve Validation**:
   Enhance the `is_valid_ophthalmology_question` function to better filter relevant questions. Consider using more advanced NLP techniques if necessary.

4. **Error Handling**: Improve error handling to make the script more robust. Consider logging errors to a file for better traceability.

5. **Documentation**: Maintain detailed documentation of any changes made. Ensure comments in the code are clear and descriptive.

6. **Code Testing**: Test the code with various URLs to ensure it works correctly and handles edge cases.

7. **Version Control**: Use a version control system (e.g., Git) to track changes and collaborate effectively.

8. **Optimization**: Optimize the code for performance, especially if dealing with a large number of URLs or large HTML content.

9. **User Interface**: If time permits, consider creating a simple user interface (CLI or GUI) to allow non-technical users to add URLs and run the script.

10. **Regular Updates**: Regularly update the script to adapt to changes in the HTML structure of target websites.

6 Future Enhancements

- **Advanced Data Processing**: Implement advanced data processing techniques to refine the extracted data.

- **Machine Learning Integration**: Use machine learning models to automatically classify and validate question-answer pairs.

- **Scalability**: Make the script scalable to handle a larger volume of data and concurrent requests.