**웹 크롤링 강의 교안**

**개요**

웹 크롤링(Web Crawling)은 웹 페이지의 데이터를 자동으로 수집하는 기술입니다. 웹 크롤링은 주로 웹 데이터 분석, 데이터 수집 및 웹 스크래핑(Web Scraping)으로 활용됩니다. 이 강의에서는 파이썬을 사용하여 정적 페이지와 동적 페이지 크롤링을 다룹니다.

**학습 목표**

1. 웹 크롤링의 기본 개념 이해

**섹션 2. 웹크롤링이란?**

**2.1 웹 크롤링의 기본 개념 이해**

웹 크롤링은 웹 페이지의 데이터를 자동으로 수집하는 기술입니다. 이를 통해 대량의 데이터를 효율적으로 수집하고 분석할 수 있습니다. 웹 크롤링의 기본 개념과 작동 방식을 상세히 설명하겠습니다.

**2.1.1 웹 크롤링의 정의**

* **웹 크롤링(Web Crawling)**: 웹 페이지를 자동으로 방문하여 데이터를 수집하는 기술
* **웹 스크래핑(Web Scraping)**: 웹 페이지에서 특정 데이터를 추출하는 기술

**2.1.2 웹 크롤링의 구성 요소**

웹 크롤링은 몇 가지 주요 구성 요소로 이루어집니다:

1. **크롤러(Crawler)**:
   * 크롤러는 웹 페이지를 방문하고 데이터를 수집하는 프로그램입니다.
   * 특정 웹 페이지의 URL을 입력하면 해당 페이지를 방문하여 HTML 소스를 가져옵니다.
2. **파서(Parser)**:
   * 파서는 가져온 HTML 소스를 분석하여 필요한 데이터를 추출합니다.
   * BeautifulSoup과 같은 파싱 라이브러리를 사용하여 HTML 소스를 구조화된 데이터로 변환합니다.
3. **데이터 저장(Data Storage)**:
   * 추출한 데이터를 구조화하여 저장합니다.
   * CSV, Excel, 데이터베이스 등 다양한 형식으로 저장할 수 있습니다.

**정적 페이지**

**개념**

정적 페이지는 서버에서 클라이언트(웹 브라우저)로 전송될 때 이미 완전히 구성된 HTML 문서입니다. 이러한 페이지는 요청할 때마다 동일한 HTML을 반환하며, 페이지의 콘텐츠는 변경되지 않습니다. 정적 페이지는 일반적으로 서버의 파일 시스템에 저장된 HTML 파일입니다.

**특징**

* **변경되지 않는 콘텐츠**: 정적 페이지는 서버에 저장된 HTML 파일로, 요청할 때마다 동일한 내용을 반환합니다.
* **빠른 로딩 속도**: 정적 페이지는 단순한 HTML 파일이므로 서버와 클라이언트 간의 데이터 전송이 빠릅니다.
* **서버 부하 적음**: 서버에서 별도의 처리를 필요로 하지 않으므로 서버 부하가 적습니다.
* **보안**: 서버 측 스크립트가 없어 보안 취약점이 적습니다.

**예시**

* **개인 블로그**: 간단한 블로그 페이지로, HTML 파일로 작성되어 있으며, 각 글은 별도의 HTML 파일로 제공됩니다.
* **회사 소개 페이지**: 회사의 소개, 연락처, 위치 등과 같은 고정된 정보를 제공하는 웹 페이지.

**동적 페이지**

**개념**

동적 페이지는 서버에서 클라이언트의 요청에 따라 HTML을 생성하여 전송하는 페이지입니다. 이러한 페이지는 서버 측 스크립트(PHP, Python, Node.js 등)를 통해 데이터베이스에서 데이터를 가져오거나 사용자 입력을 처리하여 콘텐츠를 동적으로 생성합니다.

**특징**

* **변경 가능한 콘텐츠**: 동적 페이지는 사용자의 요청이나 입력에 따라 콘텐츠가 변경됩니다.
* **복잡한 로직 처리**: 서버 측 스크립트를 사용하여 복잡한 로직을 처리하고, 데이터베이스와 상호작용할 수 있습니다.
* **느린 로딩 속도**: 서버 측 스크립트를 실행하고 데이터베이스와 상호작용하는 데 시간이 필요하므로 로딩 속도가 느릴 수 있습니다.
* **서버 부하 큼**: 동적 페이지는 서버 측에서 많은 처리를 필요로 하므로 서버 부하가 큽니다.

**예시**

* **전자 상거래 사이트**: 사용자가 제품을 검색하고, 장바구니에 추가하고, 주문을 처리하는 등의 기능을 제공하는 사이트.
* **소셜 미디어 플랫폼**: 사용자 프로필, 친구 목록, 게시물 작성 및 댓글 기능을 제공하는 사이트.

**정적 페이지와 동적 페이지의 차이**

| **특징** | **정적 페이지** | **동적 페이지** |
| --- | --- | --- |
| 콘텐츠 변경 | 변경되지 않음 | 변경 가능 |
| 로딩 속도 | 빠름 | 느림 |
| 서버 부하 | 적음 | 큼 |
| 복잡한 로직 처리 | 불가능 | 가능 |
| 보안 | 상대적으로 안전 | 보안 취약점 존재 |
| 데이터베이스 사용 | 없음 | 있음 |

정적 페이지와 동적 페이지는 각각의 특성과 용도에 따라 사용됩니다. 정적 페이지는 간단하고 변경되지 않는 콘텐츠를 제공하는 데 적합하며, 동적 페이지는 사용자 상호작용과 데이터베이스를 이용한 복잡한 기능을 제공하는 데 적합합니다.

**2.1.3 웹 크롤링의 동작 과정**

**(**BeautifulSoup**)**

**1. URL 요청**

**원리:**

* HTTP 요청을 보내 웹 페이지의 HTML 소스를 가져옵니다.
* requests 라이브러리를 사용하여 URL에 HTTP 요청을 보내고, 웹 서버로부터 HTML 응답을 받습니다.

**과정:**

1. requests 라이브러리를 사용하여 웹 페이지의 URL에 HTTP 요청을 보냅니다.
2. 웹 서버는 요청된 URL에 해당하는 HTML 문서를 응답으로 반환합니다.
3. 반환된 HTML 소스를 저장합니다.

**2. HTML 파싱**

**원리:**

* 가져온 HTML 소스를 파싱하여 특정 태그나 속성을 찾을 수 있는 구조화된 형식으로 변환합니다.
* BeautifulSoup 라이브러리를 사용하여 HTML 소스를 분석합니다.

**과정:**

1. BeautifulSoup 객체를 생성하고 HTML 소스를 전달하여 파싱합니다.
2. 파싱된 HTML 문서에서 특정 태그나 속성을 찾습니다.
3. 파싱된 문서 구조를 탐색하여 필요한 데이터를 추출할 수 있도록 준비합니다.

**3. 데이터 추출**

**원리:**

* 파싱된 HTML 문서에서 원하는 데이터를 식별하고 추출합니다.
* 특정 태그의 텍스트나 속성 값을 추출합니다.

**과정:**

1. find() 또는 find\_all() 메서드를 사용하여 특정 태그를 찾습니다.
2. 찾은 태그의 텍스트나 속성 값을 추출합니다.
3. 원하는 모든 데이터를 추출할 때까지 반복합니다.

**4. 데이터 저장**

**원리:**

* 추출한 데이터를 원하는 형식으로 저장합니다.
* CSV 파일, JSON 파일, 데이터베이스 등 다양한 형식으로 저장할 수 있습니다.

**과정:**

1. 추출한 데이터를 리스트나 딕셔너리 형태로 정리합니다.
2. csv 라이브러리 등을 사용하여 데이터를 CSV 파일로 저장합니다.
3. 데이터베이스에 저장할 경우, 적절한 데이터베이스 라이브러리를 사용하여 데이터를 삽입합니다.
4. 정적 페이지 크롤링 방법 습득

뉴스 페이지의 본문을 가져오고, 단어 수를 세보기

import requests

from bs4 import BeautifulSoup

# 1. URL 요청

url = 'https://n.news.naver.com/mnews/article/047/0002432943?sid=101'

response = requests.get(url)

# 응답 확인

if response.status\_code == 200:

    html\_content = response.text

    # 2. HTML 파싱

    soup = BeautifulSoup(html\_content, 'html.parser')

    # 3. 기사의 본문 텍스트 추출

    article\_tag = soup.find('div', {'id': 'dic\_area'})  # 기사 본문이 포함된 태그를 지정합니다.

    if article\_tag:

        article\_text = article\_tag.get\_text(separator=' ').strip()

        # 기사 본문 출력

        print("기사 본문:")

        print(article\_text)

    else:

        print("기사를 포함하는 'div' 태그를 찾을 수 없습니다.")

else:

    print(f'HTTP 요청 실패. 상태 코드: {response.status\_code}')

단어 수

import requests

from bs4 import BeautifulSoup

# 1. URL 요청

url = 'https://n.news.naver.com/mnews/article/047/0002432943?sid=101'

response = requests.get(url)

# 응답 확인

if response.status\_code == 200:

    html\_content = response.text

    # 2. HTML 파싱

    soup = BeautifulSoup(html\_content, 'html.parser')

    # 3. 특정 영역의 텍스트 추출

    article\_tag = soup.find('article', {'id': 'dic\_area'})

    if article\_tag:

        article\_text = article\_tag.get\_text()

        # 4. 단어 수 계산

        word\_to\_count = "정부"

        count = article\_text.count(word\_to\_count)

        print(f"'{word\_to\_count}' 단어 수: {count}")

    else:

        print("지정된 'article' 태그를 찾을 수 없습니다.")

else:

    print(f'HTTP 요청 실패. 상태 코드: {response.status\_code}')

2.1.3 셀레니움

Selenium을 사용한 웹 크롤링의 기본 동작 과정과 원리를 자세히 설명해 드리겠습니다.

**1. URL 요청**

**원리:**

* Selenium은 웹 브라우저를 자동으로 제어하여 웹 페이지를 요청합니다.
* 사용자가 수동으로 브라우저를 통해 웹 페이지에 접근하는 것처럼, Selenium도 웹 페이지의 URL을 열어 해당 페이지를 요청합니다.

**과정:**

1. **웹 드라이버 설정**:
   * Selenium은 Chrome, Firefox 등 다양한 브라우저의 드라이버를 사용하여 브라우저를 제어합니다.
   * 각 브라우저에 맞는 드라이버를 설정하고, Selenium WebDriver를 통해 브라우저를 실행합니다.
2. **URL 열기**:
   * driver.get(URL) 메서드를 사용하여 지정된 URL을 엽니다.
   * 브라우저는 해당 URL에 접근하여 웹 페이지를 로드합니다.

**2. HTML 파싱**

**원리:**

* Selenium은 브라우저를 통해 로드된 웹 페이지의 HTML 소스를 가져와 이를 파싱할 수 있습니다.
* JavaScript로 동적으로 생성된 콘텐츠도 브라우저를 통해 완전히 로드된 후 파싱할 수 있습니다.

**과정:**

1. **페이지 로드 대기**:
   * 웹 페이지가 완전히 로드될 때까지 대기합니다.
   * 특정 요소가 로드될 때까지 WebDriverWait을 사용하여 대기할 수 있습니다.
2. **HTML 소스 가져오기**:
   * 로드된 웹 페이지의 HTML 소스를 가져옵니다.
   * driver.page\_source를 사용하여 현재 페이지의 HTML 소스를 가져옵니다.
3. **특정 요소 찾기**:
   * find\_element 및 find\_elements 메서드를 사용하여 특정 태그나 클래스 이름, ID 등을 가진 요소를 찾습니다.
   * 예를 들어, 특정 기사 제목을 찾거나, 링크를 찾을 수 있습니다.

**3. 데이터 추출**

**원리:**

* Selenium을 사용하여 찾은 요소에서 원하는 데이터를 추출합니다.
* 태그의 텍스트나 속성 값을 추출할 수 있습니다.

**과정:**

1. **태그의 텍스트 추출**:
   * 찾은 요소의 텍스트를 추출합니다.
   * 예를 들어, <div> 태그 내부의 텍스트를 추출할 수 있습니다.
2. **속성 값 추출**:
   * 특정 태그의 속성 값을 추출합니다.
   * 예를 들어, <a> 태그의 href 속성 값을 추출하여 링크를 얻을 수 있습니다.
3. **데이터 정리**:
   * 추출한 데이터를 리스트나 딕셔너리 형태로 정리합니다.
   * 필요에 따라 중복을 제거하거나, 정렬할 수 있습니다.

**4. 데이터 저장**

**원리:**

* 추출한 데이터를 원하는 형식으로 저장합니다.
* CSV 파일, JSON 파일, 데이터베이스 등 다양한 형식으로 저장할 수 있습니다.

**과정:**

1. **CSV 파일로 저장**:
   * csv 라이브러리를 사용하여 데이터를 CSV 파일로 저장합니다.
   * 파일을 열고 데이터를 작성한 후 파일을 닫습니다.
2. **데이터베이스에 저장**:
   * 데이터베이스 라이브러리를 사용하여 데이터를 데이터베이스에 삽입합니다.
   * SQLite, MySQL, PostgreSQL 등 다양한 데이터베이스 시스템을 사용할 수 있습니다.
3. **JSON 파일로 저장**:
   * json 라이브러리를 사용하여 데이터를 JSON 파일로 저장합니다.
   * 데이터를 JSON 형식으로 변환하고 파일에 작성합니다.
4. 동적 페이지 크롤링 방법 습득

Html을 검사해서 값을 추가하는 방법을 알아야 함. 우클릭 후 검사

크롬드라이버 설치 방법

<https://googlechromelabs.github.io/chrome-for-testing/>

pip install webdriver\_manager

import time

import csv

from datetime import datetime

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.chrome.service import Service

from webdriver\_manager.chrome import ChromeDriverManager

# Chrome 드라이버 경로 설정

driver\_path = 'C:/chromedriver-win64/chromedriver.exe'

service = Service(driver\_path)

driver = webdriver.Chrome(service=service)

# 웹사이트 열기

url = 'https://search.naver.com/search.naver?where=news&query=%EB%B9%84%ED%8A%B8%EC%BD%94%EC%9D%B8&sm=tab\_opt&sort=1&photo=0&field=0&pd=4&ds=&de=&docid=&related=0&mynews=0&office\_type=0&office\_section\_code=0&news\_office\_checked=&nso=so%3Add%2Cp%3A1d&is\_sug\_officeid=0&office\_category=0&service\_area=0'  # 대상 웹사이트 URL로 변경

driver.get(url)

# 페이지 배율 줄이기 (줌 아웃)

driver.execute\_script("document.body.style.zoom='50%'")

time.sleep(2)  # 페이지 로드 시간을 위해 잠시 대기

# 페이지 스크롤 끝까지 내리기

last\_height = driver.execute\_script("return document.body.scrollHeight")

while True:

    driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")

    time.sleep(2)

    new\_height = driver.execute\_script("return document.body.scrollHeight")

    if new\_height == last\_height:

        break

    last\_height = new\_height

# 데이터 수집

news\_data = []

news\_elements = driver.find\_elements(By.CSS\_SELECTOR, 'li.bx')

for element in news\_elements:

    try:

        link\_element = element.find\_element(By.CSS\_SELECTOR, 'a[href\*="https://n.news.naver.com/"]')

        link = link\_element.get\_attribute('href')

        # 제목 수집

        title\_element = element.find\_element(By.CSS\_SELECTOR, 'a.news\_tit')

        title = title\_element.get\_attribute('title')

        # 시간 수집 - span.info 요소에서 시간 텍스트 찾기

        time\_elements = element.find\_elements(By.CSS\_SELECTOR, 'span.info')

        time\_text = ""

        for time\_element in time\_elements:

            time\_text = time\_element.text

            if "분 전" in time\_text or "시간 전" in time\_text or "일 전" in time\_text:

                break

        news\_data.append({

            'title': title,

            'time': time\_text,

            'link': link

        })

    except Exception as e:

        # 예외 발생 시 무시하고 다음 요소로 넘어감

        continue

# 크롬 드라이버 종료

driver.quit()

# 수집한 데이터를 CSV 파일로 저장

current\_time = datetime.now().strftime('%Y%m%d\_%H%M%S')

file\_name = f'{current\_time}.csv'

with open(file\_name, mode='w', newline='', encoding='utf-8-sig') as file:

    writer = csv.DictWriter(file, fieldnames=['title', 'time', 'link'])

    writer.writeheader()

    writer.writerows(news\_data)

print(f'Data saved to {file\_name}')

위 코드는 Selenium을 사용하여 네이버 뉴스 검색 결과 페이지에서 비트코인 관련 뉴스의 제목, 시간, 링크를 수집한 후, 이를 CSV 파일로 저장하는 작업을 수행합니다. 코드는 다음과 같은 단계를 거칩니다:

1. **크롬 드라이버 설정 및 초기화**:
   * webdriver\_manager를 사용하여 크롬 드라이버를 자동으로 설치하고 설정합니다.
   * driver\_path 변수에 로컬 크롬 드라이버의 경로를 지정합니다.
2. **웹사이트 열기**:
   * 대상 URL(네이버 뉴스 검색 결과 페이지)을 엽니다.
3. **페이지 배율 줄이기 (줌 아웃)**:
   * JavaScript를 사용하여 페이지 배율을 50%로 줄여 화면에 더 많은 내용을 표시합니다.
   * 페이지 로드 시간을 고려하여 잠시 대기합니다.
4. **페이지 스크롤**:
   * 동적 웹페이지에서 모든 내용을 로드하기 위해 페이지의 끝까지 스크롤합니다.
   * 현재 페이지의 높이를 확인하고, 더 이상 높이가 변하지 않을 때까지 스크롤을 계속합니다.
5. **데이터 수집**:
   * li.bx 요소를 찾아 뉴스 항목들을 수집합니다.
   * 각 뉴스 항목에서 다음 정보를 수집합니다:
     + **링크**: a[href\*="https://n.news.naver.com/"] 요소의 href 속성을 통해 링크를 가져옵니다.
     + **제목**: a.news\_tit 요소의 title 속성을 통해 제목을 가져옵니다.
     + **시간**: span.info 요소들 중 "분 전", "시간 전", "일 전" 등의 텍스트를 포함하는 첫 번째 요소의 텍스트를 가져옵니다.
6. **크롬 드라이버 종료**:
   * 작업이 완료된 후 크롬 드라이버를 종료합니다.
7. **수집한 데이터를 CSV 파일로 저장**:
   * 현재 시간을 기반으로 파일 이름을 생성합니다.
   * 수집한 뉴스 데이터를 CSV 파일로 저장합니다. 인코딩은 UTF-8로 설정하여 한글이 깨지지 않도록 합니다.

**이 코드는 기본적으로 다음을 수행합니다**:

* 네이버 뉴스 검색 결과 페이지에서 비트코인 관련 뉴스를 검색합니다.
* 뉴스 제목, 시간, 링크를 수집합니다.
* 수집한 데이터를 CSV 파일로 저장합니다.

전체적으로 이 코드는 동적 웹페이지에서 특정 정보를 자동으로 수집하고 이를 구조화된 데이터 형식(CSV)으로 저장하는 작업을 자동화하는 데 유용합니다.

import time

import csv

from datetime import datetime

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.chrome.service import Service

from selenium.webdriver.chrome.options import Options

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.common.exceptions import NoSuchElementException, TimeoutException, StaleElementReferenceException, ElementClickInterceptedException

from webdriver\_manager.chrome import ChromeDriverManager

# Chrome 드라이버 설정

chrome\_options = Options()

chrome\_options.add\_argument("--start-maximized")

driver\_path = 'C:/chromedriver-win64/chromedriver.exe'

service = Service(driver\_path)

driver = webdriver.Chrome(service=service, options=chrome\_options)

# 링크 목록을 저장한 CSV 파일 경로

links\_file = '0303.csv'  # 'links.csv' 파일은 현재 작업 디렉토리에 위치해야 합니다.

# 링크 목록 읽기

with open(links\_file, mode='r', encoding='utf-8-sig') as file:

    reader = csv.DictReader(file)

    links = [{'title': row['title'], 'time': row['time'], 'link': row['link']} for row in reader]

# 수집한 데이터를 저장할 리스트

all\_news\_data = []

for link\_info in links:

    driver.get(link\_info['link'])

    time.sleep(2)  # 페이지 로드 시간을 위해 잠시 대기

    # 기사 제목 수집

    title\_element = driver.find\_element(By.CSS\_SELECTOR, 'h2#title\_area span')

    title = title\_element.text

    # 기사 입력 시간 수집

    time\_element = driver.find\_element(By.CSS\_SELECTOR, 'span.media\_end\_head\_info\_datestamp\_time.\_ARTICLE\_DATE\_TIME')

    article\_time = time\_element.get\_attribute('data-date-time')

    # 댓글 수 수집

    comment\_count\_element = driver.find\_element(By.CSS\_SELECTOR, 'a#comment\_count')

    comment\_count\_text = comment\_count\_element.text

    comment\_count = 0 if 'is\_zero' in comment\_count\_element.get\_attribute('class') else int(comment\_count\_text)

    if comment\_count > 0:

        # 댓글 페이지로 이동

        comment\_page\_url = comment\_count\_element.get\_attribute('href')

        driver.get(comment\_page\_url)

        # 페이지 배율 줄이기 (줌 아웃)

        driver.execute\_script("document.body.style.zoom='50%'")

        time.sleep(2)  # 페이지 로드 시간을 위해 잠시 대기

        # 클린봇 해제하여 모든 댓글을 볼 수 있도록 설정

        try:

            cleanbot = driver.find\_element(By.CSS\_SELECTOR, 'a.u\_cbox\_cleanbot\_setbutton')

            cleanbot.click()

            time.sleep(1)

            cleanbot\_disable = driver.find\_element(By.XPATH, "//input[@id='cleanbot\_dialog\_checkbox\_cbox\_module']")

            cleanbot\_disable.click()

            time.sleep(1)

            cleanbot\_confirm = driver.find\_element(By.CSS\_SELECTOR, 'div.u\_cbox\_layer\_cleanbot2\_extra a')

            cleanbot\_confirm.click()

            time.sleep(1)

        except:

            pass

        # "더보기" 버튼 클릭을 통한 댓글 로딩

        while True:

            try:

                more\_button = WebDriverWait(driver, 5).until(

                    EC.element\_to\_be\_clickable((By.CSS\_SELECTOR, "a.u\_cbox\_btn\_more"))

                )

                driver.execute\_script("arguments[0].scrollIntoView(true);", more\_button)

                time.sleep(1)

                more\_button.click()

                time.sleep(2)

            except (NoSuchElementException, TimeoutException):

                break

            except ElementClickInterceptedException:

                driver.execute\_script("arguments[0].click();", more\_button)

                time.sleep(2)

        # 데이터 수집

        while True:

            try:

                comments = driver.find\_elements(By.CSS\_SELECTOR, ".u\_cbox\_contents")

                comment\_dates = driver.find\_elements(By.CSS\_SELECTOR, ".u\_cbox\_date")

                recommend\_counts = driver.find\_elements(By.CSS\_SELECTOR, ".u\_cbox\_cnt\_recomm")

                unrecommend\_counts = driver.find\_elements(By.CSS\_SELECTOR, ".u\_cbox\_cnt\_unrecomm")

                break

            except StaleElementReferenceException:

                time.sleep(0.5)

        for i in range(len(comments)):

            try:

                comment = comments[i].text

                comment\_date = comment\_dates[i].text

                recommend\_count = recommend\_counts[i].text

                unrecommend\_count = unrecommend\_counts[i].text

                all\_news\_data.append({

                    "title": title,

                    "time": article\_time,

                    "comment\_count": comment\_count,

                    "comment": comment,

                    "comment\_date": comment\_date,

                    "recommend\_count": recommend\_count,

                    "unrecommend\_count": unrecommend\_count

                })

            except StaleElementReferenceException:

                continue

# 크롬 드라이버 종료

driver.quit()

# 수집한 데이터를 CSV 파일로 저장

current\_time = datetime.now().strftime('%Y%m%d\_%H%M%S')

file\_name = f'{current\_time}.csv'  # 파일을 현재 작업 디렉토리에 저장

with open(file\_name, mode='w', newline='', encoding='utf-8-sig') as file:

    fieldnames = ['title', 'time', 'comment\_count', 'comment', 'comment\_date', 'recommend\_count', 'unrecommend\_count']

    writer = csv.DictWriter(file, fieldnames=fieldnames)

    writer.writeheader()

    writer.writerows(all\_news\_data)

print(f'Data saved to {file\_name}')

위 코드는 주어진 링크 목록에 있는 각 뉴스 기사의 제목, 시간, 댓글 수, 각 댓글의 내용, 댓글 작성 시간, 추천 수, 비추천 수를 수집하여 CSV 파일로 저장합니다. 이 코드는 다음과 같은 단계를 수행합니다:

1. **Chrome 드라이버 설정**:
   * webdriver\_manager를 사용하여 Chrome 드라이버를 자동으로 설치하고 설정합니다.
   * 드라이버를 최대화된 창 모드로 실행합니다.
2. **링크 목록 읽기**:
   * CSV 파일(0303.csv)에서 뉴스 기사 링크 목록을 읽습니다. 이 파일은 현재 작업 디렉토리에 위치해야 합니다.
3. **각 뉴스 기사 페이지 방문 및 데이터 수집**:
   * 각 링크를 방문하여 뉴스 기사 제목, 시간, 댓글 수를 수집합니다.
   * 댓글 수가 0보다 큰 경우, 댓글 페이지로 이동하여 모든 댓글을 로딩하고 수집합니다.
4. **댓글 페이지에서 추가 작업 수행**:
   * 페이지 배율을 50%로 줄여서 더 많은 내용을 한 화면에 표시합니다.
   * 클린봇 설정을 비활성화하여 모든 댓글을 볼 수 있도록 합니다.
   * "더보기" 버튼을 반복해서 클릭하여 모든 댓글을 로딩합니다.
5. **댓글 데이터 수집**:
   * 각 댓글의 내용, 작성 시간, 추천 수, 비추천 수를 수집합니다.
6. **크롬 드라이버 종료**:
   * 모든 작업이 완료된 후, 크롬 드라이버를 종료합니다.
7. **수집한 데이터를 CSV 파일로 저장**:
   * 현재 시간을 기반으로 파일 이름을 생성합니다.
   * 수집한 모든 데이터를 CSV 파일로 저장합니다. 인코딩은 UTF-8로 설정하여 한글이 깨지지 않도록 합니다.