

AI Programming

데이터 마이닝 / 생성형 AI

데이터 수집

first
coding

데이터 전처리의 개요

데이터 전처리의 중요성

데이터의 유형(정형, 비정형, 반정형 데이터)

데이터 전처리 과정 개요: 데이터 수집, 정제, 변환, 통합

정규 표현식을 이용한 보안 로그 처리 예제

- 파이썬 re 모듈을 활용하여 보안 로그에서 IP 주소, 날짜, 접근 경로, 상태 코드 등을 추출하고, 이를 CSV 파일로 저장
 - 예시 로그 데이터 (sample_logs.log)

```
192.168.1.10 - - [14/Jun/2025:10:15:32 +0900] "GET /admin/login.php HTTP/1.1" 200 532
```

```
10.20.30.40 - - [14/Jun/2025:10:16:01 +0900] "POST /upload.php HTTP/1.1" 403 189
```

```
172.16.0.2 - - [14/Jun/2025:10:17:55 +0900] "GET /index.html HTTP/1.1" 200 1024
```

정규 표현식을 이용한 보안 로그 처리 예제

- 파이썬 re 모듈을 활용하여 보안 로그에서 IP 주소, 날짜, 접근 경로, 상태 코드 등을 추출하고, 이를 CSV 파일로 저장
 - 파이썬 코드: 로그 파싱 및 CSV 저장

```
import re
import csv

# 로그 파일 경로
log_file_path = 'sample_logs.txt'
output_csv_path = 'parsed_logs.csv'
```

정규 표현식을 이용한 보안 로그 처리 예제

- 파이썬 re 모듈을 활용하여 보안 로그에서 IP 주소, 날짜, 접근 경로, 상태 코드 등을 추출하고, 이를 CSV 파일로 저장
 - 파이썬 코드: 로그 파싱 및 CSV 저장

```
# 정규 표현식 패턴 정의
log_pattern = re.compile(
    r'(?P<ip>\d+\.\d+\.\d+\.\d+)\s'           # IP 주소
    r'- - \[(?P<datetime>[^\]]+)\]'         # 날짜 및 시간
    r'"(?P<method>GET|POST|PUT|DELETE|HEAD) ' # HTTP 메서드
    r'(?P<path>[^\s]+).*?'                  # 요청 경로
    r'(?P<status>\d{3})\s'                  # 상태 코드
    r'(?P<size>\d+)'                         # 응답 크기
)

# (?P<이름>패턴) 형태는 "이름이 있는 그룹" 으로, 추출 결과를 딕셔너리처럼
groupdict()로 관리할 수 있게 해줍니다.
```

정규 표현식을 이용한 보안 로그 처리 예제

- 파이썬 re 모듈을 활용하여 보안 로그에서 IP 주소, 날짜, 접근 경로, 상태 코드 등을 추출하고, 이를 CSV 파일로 저장

정규식 부분	설명	예시 추출 값
(?P<ip>Wd+W.Wd+W.Wd+W.Wd+)	IP 주소 형식 (예: 192.168.0.1) (?P<ip>...)는 추출된 값을 "ip"라는 이름으로 저장	"192.168.1.10"
Ws	공백 한 칸	
- -	Apache 로그의 포맷상 사용자 식별자 자리인데 일반적으로 -로 표시	
W[(?P<datetime>[^W])+]W	대괄호 []로 감싼 시간 문자열을 "datetime"이라는 이름으로 추출	14/Jun/2025:10:15:32 +0900
`"(?PGET	POST	PUT
(?P<path>[^]+)	요청한 URL 경로 부분을 추출	"/admin/login.php"
.*?"	나머지 요청 정보(HTTP 버전 등)를 건너뛴	
(?P<status>Wd{3})	응답 상태 코드 (3자리 숫자)	200, 403 등
(?P<size>Wd+)	응답 본문의 크기(바이트 수)	예: 532

정규 표현식을 이용한 보안 로그 처리 예제

- 파이썬 re 모듈을 활용하여 보안 로그에서 IP 주소, 날짜, 접근 경로, 상태 코드 등을 추출하고, 이를 CSV 파일로 저장
 - 파이썬 코드: 로그 파싱 및 CSV 저장

```
# 파싱된 결과 저장 리스트
parsed_logs = []

# 로그 파일 읽기 및 정규식 적용
with open(log_file_path, 'r') as f:
    for line in f:
        match = log_pattern.search(line)
        if match:
            parsed_logs.append(match.groupdict()) #
```

정규 표현식을 이용한 보안 로그 처리 예제

- 파이썬 re 모듈을 활용하여 보안 로그에서 IP 주소, 날짜, 접근 경로, 상태 코드 등을 추출하고, 이를 CSV 파일로 저장
 - 파이썬 코드: 로그 파싱 및 CSV 저장

```
# CSV로 저장
with open(output_csv_path, 'w', newline='') as csvfile:
    fieldnames = ['ip', 'datetime', 'method', 'path', 'status', 'size']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()
    for entry in parsed_logs:
        writer.writerow(entry)

print("보안 로그가 CSV로 저장되었습니다:", output_csv_path)
```


정규 표현식을 이용한 보안 로그 처리 예제

- 파이썬 코드: 로그 파싱 및 CSV로그 파일에서 데이터 추출
→ pandas DataFrame 변환python복사편집 저장

```
import re
import pandas as pd

# 로그 파일 경로
log_file_path = 'sample_logs.log'

# 정규 표현식 패턴 정의
log_pattern = re.compile(
    r'(?P<ip>\d+\.\.\d+\.\.\d+)\s'
    r'- - \[(?P<datetime>[^\]]+)\]'
    r'"(?P<method>GET|POST|PUT|DELETE) '
    r'(?P<path>[^\s]+).*?'
    r'(?P<status>\d{3}) '
    r'(?P<size>\d+)'
)
```

정규 표현식을 이용한 보안 로그 처리 예제

- 파이썬 코드: 로그 파싱 및 CSV로그 파일에서 데이터 추출

→ pandas DataFrame 변환python복사편집 저장

```
# 결과 저장
log_entries = []

with open(log_file_path, 'r') as f:
    for line in f:
        match = log_pattern.search(line)
        if match:
            entry = match.groupdict()
            # datetime을 datetime 객체로 변환
            entry['datetime'] = pd.to_datetime(entry['datetime'], format='%d/%b/%Y:%H:%M:%S %z')
            entry['status'] = int(entry['status'])
            entry['size'] = int(entry['size'])
            log_entries.append(entry)

# DataFrame 생성
df = pd.DataFrame(log_entries)

# 결과 확인
print(df.head())
```

정규 표현식을 이용한 보안 로그 처리 예제

- 파이썬 코드: 로그 파싱 및 CSV로그 파일에서 데이터 추출

→ pandas DataFrame 변환python복사편집 저장

- 403 상태 코드(Forbidden) 요청만 추출

```
forbidden_df = df[df['status'] == 403]
print(forbidden_df)
```

- /admin 경로에 접근한 로그만 추출

```
admin_access_df = df[df['path'].str.contains(r'^/admin')]
print(admin_access_df)
```

정규 표현식을 이용한 보안 로그 처리 예제

- 파이썬 코드: 로그 파싱 및 CSV로그 파일에서 데이터 추출

→ pandas DataFrame 변환python복사편집 저장

- 403 상태 코드(Forbidden) 요청만 추출

```
status_counts = df.groupby('status').size().reset_index(name='count')
print(status_counts)
```

- IP별 요청 횟수 Top 5

```
top_ips = df.groupby('ip').size().reset_index(name='count').sort_values(by='count',
ascending=False).head(5)
print(top_ips)
```

- 시간별 요청량 집계 (분 단위)python복사편집

```
df['minute'] = df['datetime'].dt.floor('min')
requests_per_minute = df.groupby('minute').size().reset_index(name='count')
print(requests_per_minute)
```

데이터 수집 개요 및 HTTP 요청

데이터 수집 방법 개요

HTTP 요청과 응답 이해 (GET, POST 등)

requests 모듈을 활용한 데이터 수집

- 데이터 수집 방법 개요

- 데이터 수집은 데이터를 효율적으로 모으고 활용하기 위해 중요한 단계
- 데이터는 다양한 형식과 출처에서 수집될 수 있으며, 목적에 따라 방법도 달라짐
 - 목적과 환경에 따라 적합한 방법을 선택해야 함

- 데이터 수집 방법

- ① 웹 스크래핑(Web Scraping)

- 웹 스크래핑은 웹사이트에서 HTML 데이터를 가져와 필요한 정보를 추출하는 방법
 - 자동화해서 데이터를 수집할 수 있음
 - 예시
 - 목적: 온라인 쇼핑몰의 상품 가격 정보 수집
 - 방법
 1. BeautifulSoup과 같은 Python 라이브러리를 사용
 2. 웹 페이지의 HTML을 파싱
 3. 상품명과 가격을 가져옴.

- 데이터 수집 방법

- ② API 활용

- API(Application Programming Interface)는 데이터를 쉽게 요청하고 응답 받을 수 있도록 제공되는 인터페이스 → 주로 REST API를 통해 JSON 형태의 데이터를 제공
 - 예시
 - 목적: 날씨 데이터를 수집
 - 방법
 - » requests 모듈을 이용하여 OpenWeatherMap API를 호출
 - » 공공데이터 포털 (<https://www.data.go.kr/>)
 - » 지방행정인허가데이터개방: LOCALDATA (<https://www.localdata.go.kr/>)

- 데이터 수집 방법

- ③ 데이터베이스에서 데이터 가져오기

- 이미 수집된 데이터를 데이터베이스에서 쿼리를 통해 가져오는 방법
 - RDBMS(SQL)이나 NoSQL 데이터베이스를 사용
 - 예시
 - 목적: 고객 정보 조회
 - 방법
 - » RDBMS를 이용하여 select를 이용해서 데이터를 조회
 - 데이터베이스 관련 모듈 이용

- 데이터 수집 방법

- ④ 공개된 데이터셋 활용

- 정부나 기업, 연구기관에서 제공하는 공개 데이터셋을 다운로드하거나 API를 통해 사용
→ Kaggle, Google Dataset Search 등이 자주 사용
 - 예시
 - 목적: 인구 통계 데이터 분석
 - 방법
 - » Kaggle에서 데이터를 다운로드한 뒤 Pandas로 읽어 분석

- 데이터 수집 방법

- ⑤ IoT 또는 센서 데이터 수집

- 센서를 이용해 실시간으로 데이터를 수집합니다
 - 온도 센서, 카메라, GPS 장치 등이 사용
 - 예시
 - 목적: 스마트 홈의 온도 데이터를 수집
 - 방법
 - » MQTT 프로토콜을 이용하여 센서 데이터를 수집

- 데이터 수집 방법

- ⑥ 소셜 미디어 데이터 수집

- 소셜 미디어 플랫폼에서 사용자 활동 데이터를 수집하는 방법
 - 해당 플랫폼의 API를 사용
 - 예시
 - 목적: 트위터의 특정 키워드로 트윗 수집
 - 방법
 - » Tweepy 라이브러리를 사용하여 트위터 API 호출

- 데이터 수집 방법
 - 웹 스크래핑: HTML 데이터 파싱
 - API 활용: JSON 데이터 가져오기
 - 데이터베이스: 저장된 데이터 검색
 - 공개 데이터셋: Kaggle 등에서 다운로드
 - IoT 센서: 실시간 데이터 수집
 - 소셜 미디어: 트위터 API 활용

- requests 모듈을 활용한 데이터 수집
 - Python의 requests 모듈은 HTTP 요청을 보내고 응답을 처리하는 데 사용되는 라이브러리
 - 웹 서버와 통신하여 데이터를 가져오거나 전송하는 작업을 쉽게 수행
 - 다양한 HTTP 메서드(GET, POST, PUT, DELETE 등)를 지원
 - 데이터를 요청하고 응답을 처리하는 과정을 간단하게 처리

requests 모듈을 활용한 데이터 수집

- requests 모듈을 활용한 데이터 수집
 - requests 모듈 설치

```
pip install requests
```

requests 모듈을 활용한 데이터 수집

- requests 모듈을 활용한 데이터 수집
 - HTTP 요청의 주요 메서드
 - GET 요청
서버에서 데이터를 가져오는 데 사용
 - 목적 : 웹 페이지의 내용을 가져오기

```
import requests
```

```
url = 'https://jsonplaceholder.typicode.com/posts'  
response = requests.get(url)
```

```
if response.status_code == 200:  
    # 상태 코드 200은 성공을 의미  
    print("데이터 가져오기 성공!")  
    print(response.text[:500])  
    # 응답 본문의 일부 출력  
else:  
    print(f"오류 발생: {response.status_code}")
```

```
--
```


requests 모듈을 활용한 데이터 수집

- requests 모듈을 활용한 데이터 수집
 - 응답 데이터 처리
 - response.text
 - 응답 본문을 문자열로 반환
 - response.json()
 - JSON 형식의 응답을 딕셔너리로 변환
 - response.status_code
 - HTTP 상태 코드 반환

```
import requests

url =
'https://jsonplaceholder.typicode.com/posts/1'
response = requests.get(url)

if response.status_code == 200:
    data = response.json()
    # JSON 응답을 딕셔너리로 변환

    print(f"제목: {data['title']}")
    print(f"내용: {data['body']}")
else:
    print(f"오류 발생: {response.status_code}")
```

requests 모듈을 활용한 데이터 수집

- requests 모듈을 활용한 데이터 수집
 - 쿼리 파라미터 사용하기
 - 서버에 데이터를 전달할 때
URL에 파라미터를 포함해서 전달
→ 특정 데이터 가져오기

```
import requests

url = 'https://jsonplaceholder.typicode.com/posts'
params = {'userId': 1}
# userId가 1인 게시글 가져오기

response = requests.get(url, params=params)

if response.status_code == 200:
    print("사용자 게시글 가져오기 성공!")
    print(response.json())
else:
    print(f"오류 발생: {response.status_code}")
```

- requests 모듈을 활용한 데이터 수집
 - 실제 응용 사례
 - 웹 페이지 크롤링: 뉴스 헤드라인 가져오기
 - API 활용: 날씨, 주식 정보, 소셜 미디어 데이터 수집
 - 자동화: 특정 시간에 정기적으로 데이터 요청
 - 주의사항
 - API 호출 시 요청 제한(Request Limit)을 초과하지 않도록 주의
 - 민감한 데이터(API 키, 인증 정보)는 환경 변수로 관리하거나 안전하게 저장
 - 웹 스크래핑 시 사이트의 robots.txt 규칙을 준수

웹 스크래핑 기초

HTML 구조 이해 (태그, 속성, DOM)

BeautifulSoup을 활용한 웹 스크래핑

- HTML이란?
 - HTML (HyperText Markup Language)은 웹 페이지의 구조를 정의하는 마크업 언어
 - HTML 문서는 브라우저가 읽어서 화면에 표시하는 기본 구조를 제공
 - HTML
 - 태그(tag)를 사용하여 요소를 정의
 - 각 요소는 속성(attribute)과 콘텐츠를 포함
 - 웹 페이지의 텍스트, 이미지, 링크, 버튼 등 다양한 요소를 구성

HTML 구조 이해 (태그, 속성, DOM)

- 태그 (Tags)

- 태그는 HTML 요소를 정의하며, 다음과 같은 형식

```
<태그이름 속성="값">내용(Contents)</태그이름>
```

- 시작 태그: <태그이름>
- 종료 태그: </태그이름> (일부 태그는 종료 태그가 없음. 예:)
- 내용: 시작 태그와 종료 태그 사이에 들어가는 콘텐츠 데이터

```
<p>Hello, World!</p>
```

- <p>: 문단(paragraph)을 나타내는 태그
- Hello, World!: 태그 사이의 내용

HTML 구조 이해 (태그, 속성, DOM)

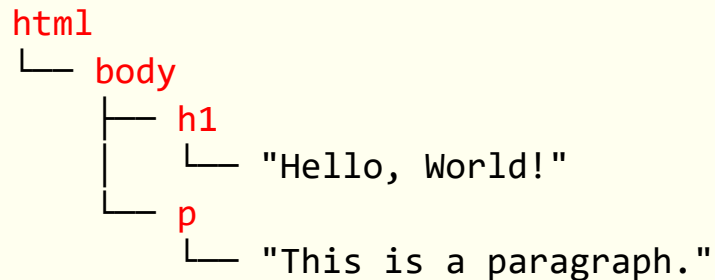
- 속성 (Attributes)
 - 속성은 태그에 추가 정보를 제공
 - 속성="값" 형식으로 작성, 속성은 시작 태그에만 추가
 - id: 요소의 고유 식별자
 - class: CSS 스타일링이나 JavaScript 제어를 위한 클래스 이름
 - href: 링크를 나타냄 (예: <a> 태그에서 사용)
 - src: 이미지나 외부 리소스의 경로 (예: 태그에서 사용)

```
<a href="https://example.com" target="_blank">Click Here</a>
```

HTML 구조 이해 (태그, 속성, DOM)

- DOM (Document Object Model)
 - DOM은 브라우저가 HTML 문서를 파싱하여 생성한 객체 모델
 - HTML 문서를 트리 구조로 표현하며, 각 요소는 노드(Node)로 구성
 - DOM의 구성 요소
 - HTML 요소 (Element Nodes): 태그들
 - 속성 (Attribute Nodes): 태그의 속성들
 - 텍스트 (Text Nodes): 태그 안의 내용

```
<html>
  <body>
    <h1>Hello, World!</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```



```
html
├── body
│   ├── h1
│   │   └── "Hello, World!"
│   └── p
│       └── "This is a paragraph."
```


HTML 구조 이해 (태그, 속성, DOM)

- DOM (Document Object Model)

```
<!DOCTYPE html>
<html>
<head>
  <title>Sample Page</title>
</head>
<body>
  <h1 id="main-title">
    Welcome to Web Scraping
  </h1>
  <p class="description">
    This is an example paragraph.
  </p>
  <a href="https://example.com" target="_blank">
    Visit Example
  </a>
</body>
</html>
```

```
html
├─ head
│   └─ title
│       └─ "Sample Page"
└─ body
    ├─ h1 (id="main-title")
    │   └─ "Welcome to Web Scraping"
    ├─ p (class="description")
    │   └─ "This is an example paragraph."
    └─ a (href="https://example.com", target="_blank")
        └─ "Visit Example"
```

BeautifulSoup을 활용한 웹 스크래핑

- BeautifulSoup을 활용한 웹 스크래핑
 - Python에서 HTML과 XML 문서를 파싱하고 원하는 데이터를 추출할 때 사용하는 라이브러리
 - HTML의 DOM 구조를 탐색하고 태그, 속성, 텍스트를 쉽게 추출
 - 설치

```
pip install beautifulsoup4
```

BeautifulSoup을 활용한 웹 스크래핑

- BeautifulSoup을 활용한 웹 스크래핑

```
from bs4 import BeautifulSoup

html_doc = """
<html>
  <head><title>Example Page</title></head>
  <body>
    <h1>Welcome to Web Scraping</h1>
    <p class="description">This is an example paragraph.</p>
    <a href="https://example.com">Visit Example</a>
  </body>
</html>
"""

# BeautifulSoup 객체 생성
soup = BeautifulSoup(html_doc, 'html.parser')

# 태그 탐색
print(soup.title.string)
print(soup.h1.string)
print(soup.a['href'])
```

```
Example Page
Welcome to Web Scraping
https://example.com
```

BeautifulSoup을 활용한 웹 스크래핑

- BeautifulSoup을 활용한 웹 스크래핑

- 주요 기능

- 1) 특정 태그 찾기

- 태그 이름으로 요소를 쉽게 찾을 수 있음

```
# 첫 번째 <p> 태그 찾기
print(soup.p)
print(soup.p['class'])
```

```
<p class="description">This is an example paragraph.</p>
['description']
```

BeautifulSoup을 활용한 웹 스크래핑

- BeautifulSoup을 활용한 웹 스크래핑

- 주요 기능

- 2) 여러 태그 찾기

`find_all()`을 사용하면 특정 태그를 모두
찾을 수 있음

```
# 모든 <a> 태그 찾기
links = soup.find_all('a')
for link in links:
    print(link['href'])
```

<https://example.com>

BeautifulSoup을 활용한 웹 스크래핑

- BeautifulSoup을 활용한 웹 스크래핑

- 주요 기능

- 3) CSS 선택자 활용

- `select()`를 사용해

- CSS 선택자**를 기반으로 요소를 찾음

- `select`는 CSS 선택자를 사용하므로
복잡한 선택 조건을 간단하게 표현
 - `find`나 `find_all` 보다
가독성이 좋고 직관적

```
# 클래스가 "description"인 <p> 태그 찾기
description = soup.select_one('.description')
print(description.text)
```

This is an example paragraph.

BeautifulSoup을 활용한 웹 스크래핑

- BeautifulSoup을 활용한 웹 스크래핑

- BeautifulSoup으로 웹 페이지 스크래핑

```
import requests
from bs4 import BeautifulSoup

# URL 요청
url = "https://news.ycombinator.com/"
response = requests.get(url)

# BeautifulSoup 객체 생성
soup = BeautifulSoup(response.text, 'html.parser')

# 제목과 링크 추출 (href 있는 것만 필터링)
for item in soup.find_all('a'):
    href = item.get('href')
    if href: # href가 존재할 때만 출력
        print(f"Title: {item.get_text(strip=True)}")
        print(f"Link: {href}")
```

--

BeautifulSoup을 활용한 웹 스크래핑

- BeautifulSoup을 활용한 웹 스크래핑

```
html_doc = """
<div class="news-item">
  <h2 class="title">Breaking News: AI Takes Over</h2>
  <a href="https://news.com/article123">Read More</a>
</div>
<div class="news-item">
  <h2 class="title">Another News: Python is Awesome</h2>
  <a href="https://news.com/article456">Read More</a>
</div>
"""

# BeautifulSoup 객체 생성
soup = BeautifulSoup(html_doc, 'html.parser')

# 모든 뉴스 아이템 추출
news_items = soup.find_all('div', class_='news-item')

for item in news_items:
    title = item.find('h2', class_='title').text
    link = item.find('a')['href']
    print(f"Title: {title}")
    print(f"Link: {link}")
```

```
Title: Breaking News: AI Takes Over
Link: https://news.com/article123
Title: Another News: Python is Awesome
Link: https://news.com/article456
```


BeautifulSoup을 활용한 웹 스크래핑

- BeautifulSoup을 활용한 웹 스크래핑

```
from bs4 import BeautifulSoup
```

```
html_doc = """
```

```
<div class="news-item">
```

```
    <h2 class="title">Breaking News: AI Takes Over</h2>
```

```
    <a href="https://news.com/article123">Read More</a>
```

```
</div>
```

```
<div class="news-item">
```

```
    <h2 class="title">Another News: Python is Awesome</h2>
```

```
    <a href="https://news.com/article456">Read More</a>
```

```
</div>
```

```
"""
```

```
# BeautifulSoup 객체 생성
```

```
soup = BeautifulSoup(html_doc, 'html.parser')
```

```
# CSS 선택자로 모든 .news-item 선택
```

```
news_items = soup.select('.news-item')
```

```
for item in news_items:
```

```
    title = item.select_one('.title').text # .title 클래스의 텍스트
```

```
    link = item.select_one('a')['href'] # 첫 번째 <a> 태그의 href 속성
```

```
    print(f"Title: {title}")
```

```
    print(f"Link: {link}")
```

```
Title: Breaking News: AI Takes Over
```

```
Link: https://news.com/article123
```

```
Title: Another News: Python is Awesome
```

```
Link: https://news.com/article456
```

동적 웹 페이지 데이터 수집

Selenium을 활용한 동적 데이터 수집
브라우저 자동화 및 요소 조작

- Selenium이란?
 - Selenium은 웹 브라우저를 자동으로 제어할 수 있게 해주는 라이브러리
 - 주로 웹 애플리케이션 테스트에 사용
 - 동적 웹 페이지에서 데이터를 수집하는 데에도 활용
 - Python을 포함한 여러 프로그래밍 언어에서 사용
 - 실제 브라우저를 통해 페이지를 로드하고 JavaScript 실행 결과까지 확인할 수 있음

Selenium을 활용한 동적 데이터 수집

- 왜 Selenium이 필요한가?
 - 정적 웹 페이지는 HTML 소스 코드만 파싱하면 데이터를 쉽게 가져올 수 있음
 - 동적 웹 페이지는 JavaScript로 콘텐츠를 생성
 - 일반적인 웹 스크래핑 도구(BeautifulSoup 등)로는 데이터 수집이 어려움
 - Selenium은 브라우저를 직접 제어하여 JavaScript가 로드된 이후의 데이터를 가져올 수 있음

Selenium을 활용한 동적 데이터 수집

- Selenium의 기본 구성 요소
 - WebDriver
 - 브라우저를 제어하는 핵심 도구
 - Chrome, Firefox, Edge 등 다양한 브라우저를 지원
 - 예: **ChromeDriver**는 Google Chrome을 제어
 - Browser
 - WebDriver가 제어할 브라우저
 - Selenium을 통해 Chrome 브라우저에서 자동으로 페이지를 열고 데이터를 수집할 수 있음
 - Python 라이브러리
 - Selenium은 Python의 라이브러리로 설치 및 실행

Selenium을 활용한 동적 데이터 수집

- Selenium 설치 및 기본 사용법

- 라이브러리 설치

```
pip install selenium
```

- 드라이버 다운로드

- <https://edgedl.me/gvt1.com/edgedl/chrome/chrome-for-testing/137.0.7151.104/win64/chromedriver-win64.zip>

-

Selenium을 활용한 동적 데이터 수집

- Selenium을 활용한 브라우저 자동화 및 요소 조작

- 브라우저 자동화

- 사람이 직접 브라우저에서 수행하는 작업(페이지 열기, 버튼 클릭, 폼 입력 등)을 Selenium WebDriver를 사용해 프로그램적으로 실행하는 것
 - 주요 기능
 - 브라우저 열기/닫기
 - 특정 URL로 이동
 - 페이지 로딩 상태 확인
 - 브라우저 창 크기 조정 및 탭 제어
 - JavaScript 실행

```
from selenium import webdriver

# Chrome 브라우저 실행
driver = webdriver.Chrome()

# 특정 URL 열기
driver.get('https://www.google.com')

# 브라우저 닫기
driver.quit()
```

Selenium을 활용한 동적 데이터 수집

- Selenium을 활용한 브라우저 자동화 및 요소 조작

- 요소 조작

- 웹 페이지에서 특정 요소(예: 버튼, 텍스트 입력 창, 링크 등)를 찾아 사용자가 수행할 동작(클릭, 텍스트 입력 등)을 자동화하는 것을 의미

- 요소를 찾는 방법

- 웹 요소를 식별 주요 메서드

- find_element: 한 개의 요소를 찾음.

- find_elements: 여러 요소를 찾음.

- 주요 Locator (위치 지정 방법)

- By.ID: 요소의 고유 ID로 찾기

- By.NAME: 요소의 이름 속성으로 찾기

- By.CLASS_NAME: 클래스 이름으로 찾기

- By.TAG_NAME: HTML 태그 이름으로 찾기

- By.XPATH: XPath로 찾기

- By.CSS_SELECTOR: CSS 선택자로 찾기

Selenium을 활용한 동적 데이터 수집

- 브라우저 자동화와 요소 조작 통합

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time

# WebDriver 실행
driver = webdriver.Chrome()
# Google 홈페이지 열기
driver.get('https://www.google.com')
# 검색창 찾기
search_box = driver.find_element(By.NAME, 'q')
# 검색어 입력 및 Enter
search_box.send_keys('Selenium tutorial')
search_box.send_keys(Keys.RETURN)
# 결과 로드 대기
time.sleep(3)
# 검색 결과 출력
results = driver.find_elements(By.XPATH, '//h3')
for index, result in enumerate(results[:5]): # 상위 5개 결과만 출력
    print(f"{index + 1}. {result.text}")
# 브라우저 닫기
driver.quit()
```

--

Selenium을 활용한 동적 데이터 수집

- 브라우저 자동화와 요소 조작 통합

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
import time
import os

# 크롬 드라이버 경로 지정
driver_path = 'chromedriver.exe' # 예: 'C:/chromedriver.exe'

# 유효성 검사
assert os.path.isfile(driver_path), f"ChromeDriver not found: {driver_path}"

# 웹드라이버 실행
service = Service(driver_path)
driver = webdriver.Chrome(service=service)

# 네이버 IT 뉴스 페이지 접속
url = 'https://news.naver.com/main/list.naver?mode=LSD&mid=sec&sid1=105'
driver.get(url)

# 페이지 로딩 대기
time.sleep(2)

# 뉴스 제목과 링크 가져오기
news_items = driver.find_elements(By.CSS_SELECTOR, 'ul.type06_headline li dt:not(.photo) a')

for item in news_items:
    title = item.text
    link = item.get_attribute('href')
    print(f'Title: {title}')
    print(f'Link: {link}')
```

--

API를 활용한 데이터 수집

REST API의 이해

공공 데이터 포털 또는 오픈 API 활용

- REST API란 무엇인가?
 - REST (Representational State Transfer)
 - 웹 서비스와 클라이언트 간의 통신을 설계하는 아키텍처 스타일
 - REST API는 REST 원칙을 따르는 API(Application Programming Interface)
 - 클라이언트(사용자)와 서버(서비스 제공자) 간 데이터(JSON 또는 XML)를 주고받는 데 사용

REST API의 이해

- REST API란 무엇인가?

- REST API의 구조

- URL 구조

REST API의 자원 접근은 URL로 구성 → 기본 형식: `https://baseurl/resource/{id}`

모든 사용자의 데이터 : `https://api.example.com/users`

특정 사용자 데이터 : `https://api.example.com/users/123`

- HTTP 메서드

- 서버에 요청을 보낼 때 HTTP 메서드와 URL을 조합해 사용

메서드	동작	설명
GET	데이터 조회	서버에서 데이터를 가져옵니다.
POST	데이터 생성	서버에 새 데이터를 추가합니다.
PUT	데이터 수정	기존 데이터를 업데이트합니다.
DELETE	데이터 삭제	서버에서 데이터를 삭제합니다.

- REST API란 무엇인가?
 - REST API의 구조
 - Headers와 Body
 - Headers: 요청 또는 응답의 부가정보를 담음. 예: Content-Type, Authorization.
 - Body: 요청 시 보내는 데이터. 주로 JSON 형식 사용.

```
POST /users HTTP/1.1
Host: api.example.com
Content-Type: application/json
Authorization: Bearer token123

{
  "name": "king",
  "email": "john.doe@example.com"
}
```

- 공공 데이터 포털과 오픈 API
 - 공공 데이터 포털
 - 정부나 공공기관이 제공하는 데이터를 모아둔 플랫폼
 - 대한민국의 대표적인 공공 데이터 포털: 공공데이터포털(data.go.kr)
 - 지방 행정인허가데이터개방: LOCALDATA (www.localdata.go.kr)
 - 다양한 분야의 데이터를 무료로 제공하며, REST API 형식으로 데이터에 접근
 - 주요 데이터
 - 교통 정보 (버스, 지하철 실시간 위치)
 - 기상 정보 (날씨 예보)
 - 인구 통계 (연령별, 지역별 인구)

- 공공 데이터 포털과 오픈 API
 - 오픈 API
 - 기업이나 기관이 데이터를 제공하기 위해 공개한 API
 - 주요 데이터
 - 네이버 검색, 카카오 검색
 - 구글 지도 API (위치 데이터 제공)
 - 트위터 API (트윗 데이터 제공)
 - NASA API (우주 관련 데이터 제공)

- 공공 데이터 포털과 오픈 API 활용 데이터 수집 : 침해사고 공격 IoC 지표

- ① 데이터 탐색 및 API 키 발급
공공 데이터 포털에서 원하는 데이터 검색
- ② API 사용 신청을 통해 인증키(API Key)를 발급
→ 이 키는 사용자가 인증된 요청을 보내기 위해 필요
- ③ API를 사용해 데이터 수집
→ Python의 requests 라이브러리로 API에 요청을 보내 데이터를 수집
→ 데이터는 JSON 형식으로 반환되며, 이를 처리하여 분석에 적합한 형식으로 변환
- ④ 데이터 전처리
→ 결측치 처리, 중복 제거, 데이터 형식 변환 등을 통해 데이터를 정리
- ⑤ 데이터 분석
→ Pandas와 Matplotlib 등을 사용해 데이터 시각화 및 통계 분석을 수행
- ⑥ 머신러닝/딥러닝 모델 학습
→ 데이터에서 유의미한 특징(feature)을 추출하여 머신러닝 및 딥러닝 모델을 훈련합니다.

공공 데이터 포털 또는 오픈 API 활용

- 공공 데이터 포털과 오픈 API 활용 데이터 수집 : 버스 실시간 위치 데이터 수집 및 분석

```
import requests

# 사용자가 쉽게 입력할 수 있는 버스번호
input_bus_number = "100" # 예: '100', '7016', 'N61' 등

# 발급받은 인증키 (URL 인코딩된 형태여야 함)
service_key = '여기에_인증키_입력' # 예: 'abcdefg1234567%2FABCDEFG...'

# 1단계: 버스노선 ID(busRouteId) 조회
search_url = f'http://ws.bus.go.kr/api/rest/busRouteInfo/getBusRouteList'
search_params = {
    'serviceKey': service_key,
    'strSrch': input_bus_number,
    'resultType': 'json'
}

search_response = requests.get(search_url, params=search_params)
search_data = search_response.json()

# 노선 검색 결과 추출
routes = search_data.get('ServiceResult', {}).get('msgBody', {}).get('itemList', [])
```

- 공공 데이터 포털과 오픈 API 활용 데이터 수집 : 버스 실시간 위치 데이터 수집 및 분석

```
if not routes:
    print(f"❌ '{input_bus_number}' 버스 노선을 찾을 수 없습니다.")
else:
    # 첫 번째 결과 사용 (가장 일치하는 노선)
    selected_route = routes[0]
    bus_route_id = selected_route.get('busRouteId')
    bus_route_name = selected_route.get('busRouteNm')

    print(f"✅ 입력한 '{input_bus_number}' 버스는 '{bus_route_name}' 노선이며, busRouteId = {bus_route_id}\n")

    # 2단계: 해당 노선의 실시간 위치정보 조회
    pos_url = f'http://ws.bus.go.kr/api/rest/buspos/getBusPosByRtid'
    pos_params = {
        'serviceKey': service_key,
        'busRouteId': bus_route_id,
        'resultType': 'json'
    }

    pos_response = requests.get(pos_url, params=pos_params)
    pos_data = pos_response.json()

    bus_list = pos_data.get('ServiceResult', {}).get('msgBody', {}).get('itemList', [])
```

- 공공 데이터 포털과 오픈 API 활용 데이터 수집 : 버스 실시간 위치 데이터 수집 및 분석

```
if not bus_list:
    print("🚗 현재 실시간 버스 위치 데이터가 없습니다.")
else:
    for bus in bus_list:
        print("🚌 버스 번호:", bus.get('plainNo'))
        print(" 차량 ID:", bus.get('vehId'))
        print(" 현재 위치 (위도, 경도):", f"{bus.get('tmY')}, {bus.get('tmX')}")
        print(" 도착 여부:", "도착" if bus.get('stopFlag') == "1" else "운행 중")
        print(" 차량 유형:", { "0": "일반", "1": "저상", "2": "굴절" }.get(bus.get('busType', "0"), "알 수 없음"))
        print("-" * 50)
```

데이터 저장 및 관리

CSV, Excel, JSON 파일에 데이터 저장

SQLite를 활용한 간단한 데이터베이스 관리

CSV, Excel, JSON 파일에 데이터 저장

- CSV 파일

- CSV (Comma-Separated Values)

- 데이터를 쉼표로 구분하여 저장하는 텍스트 기반의 파일 형식.
 - 간단한 데이터 저장과 교환에 적합하며, 대부분의 데이터 분석 도구와 호환
 - Python에서 CSV 파일 저장

```
import csv

# 데이터 준비
data = [
    ['Name', 'Age', 'City'],
    ['Alice', 30, 'New York'],
    ['Bob', 25, 'Los Angeles'],
    ['Charlie', 35, 'Chicago']
]

# CSV 파일에 저장
with open('data.csv', mode='w', newline='',
encoding='utf-8') as file:
    writer = csv.writer(file)
    writer.writerows(data)

print("CSV 파일이 저장되었습니다!")
```

CSV, Excel, JSON 파일에 데이터 저장

- CSV 파일

- CSV (Comma-Separated Values)

- CSV 파일 읽기

pandas 라이브러리를 주로 사용

- `pd.read_csv('파일경로')`

: CSV 파일을 읽어 데이터프레임 형태로 반환

- `data.head()`

: 상위 몇 개의 데이터를 미리보기.

- CSV 파일은 단순 텍스트이므로, `open()`을 사용

```
import pandas as pd
```

```
# CSV 파일 읽기
```

```
data = pd.read_csv('data.csv')
```

```
print(data)
```

```
# open()을 사용한 CSV 파일 읽기
```

```
with open('data.csv', 'r') as file:
```

```
    for line in file:
```

```
        print(line.strip()) # 줄바꿈 제거 후 출력
```

CSV, Excel, JSON 파일에 데이터 저장

- Excel 파일

- Excel

- 데이터 시각화와 분석에 강력
 - 표 형태의 데이터를 저장할 수 있음
 - openpyxl 또는 pandas 라이브러리를 주로 사용
 - Excel 파일 쓰기

```
import pandas as pd

# 데이터 준비
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [30, 25, 35],
    'City': ['New York', 'Los Angeles', 'Chicago']
}

# DataFrame 생성
df = pd.DataFrame(data)

# Excel 파일로 저장
df.to_excel('data.xlsx', index=False)

print("Excel 파일이 저장되었습니다!")
```


CSV, Excel, JSON 파일에 데이터 저장

- Excel 파일

- Excel

- Excel 파일 읽기

`pd.read_excel('파일경로')`: 엑셀 파일 읽기.

```
data = pd.read_excel('data.xlsx')
print(data)
```

- JSON 파일

- JSON (JavaScript Object Notation)

- 키-값 쌍으로 데이터를 저장
 - API와 데이터 교환에서 주로 사용
 - 계층적 데이터 구조를 표현할 수 있어 XML의 대안으로 자주 사용
 - Python에서 JSON 파일 저장
 - `json.dump()`
: 데이터를 JSON 형식으로 저장.
 - `indent=4`를 사용하여 사람이 읽기 쉽게 데이터를 정렬

```
import json

# 데이터 준비
data = {
    'people': [
        {'name': 'Alice', 'age': 30, 'city': 'New York'},
        {'name': 'Bob', 'age': 25, 'city': 'Los Angeles'},
        {'name': 'Charlie', 'age': 35, 'city': 'Chicago'}
    ]
}

# JSON 파일로 저장
with open('data.json', mode='w', encoding='utf-8') as file:
    json.dump(data, file, indent=4)

print("JSON 파일이 저장되었습니다!")
```

- JSON 파일

- JSON (JavaScript Object Notation)

- Python에서 JSON 파일 읽기

- json.load()

- : JSON 파일을 파이썬의 딕셔너리로 변환.

```
import json

# JSON 파일 읽기
with open('example.json', 'r') as file:
    data = json.load(file)

print(data)
```

CSV, Excel, JSON 파일에 데이터 저장

- 각 형식의 비교

형식	장점	단점
CSV	간단하고 가볍다. 대부분의 도구와 호환 가능.	계층적 데이터 저장 불가.
Excel	시각화와 다중 시트 지원. 구조화된 데이터 관리.	무겁고 대규모 데이터 교환에 적합하지 않음.
JSON	계층적 데이터 저장 가능. API 데이터 교환에 적합.	사람이 직접 읽기 어렵고 편집 도구가 필요함.

- 활용 팁

- CSV: 대량 데이터 저장 및 분석 초기 단계에 적합
- Excel: 데이터 요약, 보고서 작성, 분석 작업에 유용
- JSON: 웹 및 모바일 애플리케이션과의 데이터 교환에 적합

SQLite를 활용한 간단한 데이터베이스 관리

- SQLite
 - 가볍고 파일 기반의 관계형 데이터베이스 관리 시스템(RDBMS)
 - 서버 설치가 필요 없으며, Python 내장 모듈로 쉽게 사용할 수 있음
 - 데이터 저장, 조회, 관리 등을 SQLite를 활용

SQLite를 활용한 간단한 데이터베이스 관리

- SQLite

- ① SQLite와 Python 연동

- Python의 sqlite3 모듈을 사용하면
SQLite 데이터베이스를 쉽게 사용할 수 있음
 - 데이터베이스 연결
 - **example.db**
SQLite 데이터베이스 파일 이름
파일이 없으면 새로 생성

```
import sqlite3

# SQLite 데이터베이스 연결 (파일 생성)
connection = sqlite3.connect("example.db")
print("데이터베이스에 연결되었습니다!")

# 연결 닫기
connection.close()
```

SQLite를 활용한 간단한 데이터베이스 관리

- SQLite

- ② 테이블 생성

- 데이터를 저장하기 위해 먼저 테이블을 생성
 - 테이블 생성 코드
 - CREATE TABLE IF NOT EXISTS
테이블이 없을 경우에만 생성.
 - PRIMARY KEY AUTOINCREMENT
기본 키로 자동 증가 값 사용.

```
import sqlite3

# 데이터베이스 연결
connection = sqlite3.connect("example.db")
cursor = connection.cursor()

# SQL 명령어로 테이블 생성
cursor.execute("""
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER,
    city TEXT
)
""")
print("테이블이 생성되었습니다!")

# 연결 종료
connection.close()
```

SQLite를 활용한 간단한 데이터베이스 관리

- SQLite

- ③ 데이터 삽입

- Python 코드로 데이터 삽입
 - ?는 값의 자리표시자
SQL 인젝션 공격을 방지
 - executemany()
여러 데이터를 한 번에 삽입할 수 있음

```
# 데이터베이스 연결
connection = sqlite3.connect("example.db")
cursor = connection.cursor()

# 데이터 삽입
cursor.execute("INSERT INTO users (name, age, city)
VALUES (?, ?, ?)",
              ("Alice", 30, "New York"))

# 여러 데이터 삽입
data = [
    ("Bob", 25, "Los Angeles"),
    ("Charlie", 35, "Chicago"),
    ("Diana", 28, "Houston")
]
cursor.executemany("INSERT INTO users (name, age, city)
VALUES (?, ?, ?)", data)

# 변경사항 저장
connection.commit()
print("데이터가 삽입되었습니다!")

# 연결 종료
connection.close()
```


SQLite를 활용한 간단한 데이터베이스 관리

- SQLite

- ④ 데이터 조회

- Python 코드로 데이터 조회
 - fetchall()
모든 조회 결과를 가져옴
 - fetchone()
한 행씩 가져옴

```
# 데이터베이스 연결
connection = sqlite3.connect("example.db")
cursor = connection.cursor()

# 데이터 조회
cursor.execute("SELECT * FROM users")
rows = cursor.fetchall()

# 조회된 데이터 출력
for row in rows:
    print(row)

# 연결 종료
connection.close()
```

SQLite를 활용한 간단한 데이터베이스 관리

- SQLite

- ⑤ 데이터 수정

- Python 코드로 데이터 수정

```
# 데이터베이스 연결
connection = sqlite3.connect("example.db")
cursor = connection.cursor()

# 데이터 수정
cursor.execute("UPDATE users SET age = ? WHERE
name = ?", (31, "Alice"))

# 변경사항 저장
connection.commit()
print("데이터가 수정되었습니다!")

# 연결 종료
connection.close()
```

- SQLite

- ⑥ 데이터 삭제

- Python 코드로 데이터 삭제

```
# 데이터베이스 연결
connection = sqlite3.connect("example.db")
cursor = connection.cursor()

# 데이터 삭제
cursor.execute("DELETE FROM users WHERE name = ?",
               ("Bob",))

# 변경사항 저장
connection.commit()
print("데이터가 삭제되었습니다!")

# 연결 종료
connection.close()
```