Chapter 03. 데이터 수집, 분석과 시각화

01. 페이스북 API를 활용한 빈도분석

02. 공공 데이터 API를 활용한 데이터 기반 추천

03. 웹서비스 데이터를 활용한 지리정보 기반 시각화

3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

3.1 일반적인 웹 서비스 데이터 수집하기

3.1.1 BeautifulSoup

1) HTML 파싱 라이브러리로 bs4버젼부터 Python 3.x 지원

2) 특징

- 단순한 몇 개의 메소드를 가지고 웹 페이지의 내용 추출이 가능(DOM 탐색이 가능하다)
- HTML 뿐만 아니라 XML도 지원한다.
- UTF-8 형식이 기본이지만 CP949 엔코딩도 지원한다.

3) 설치

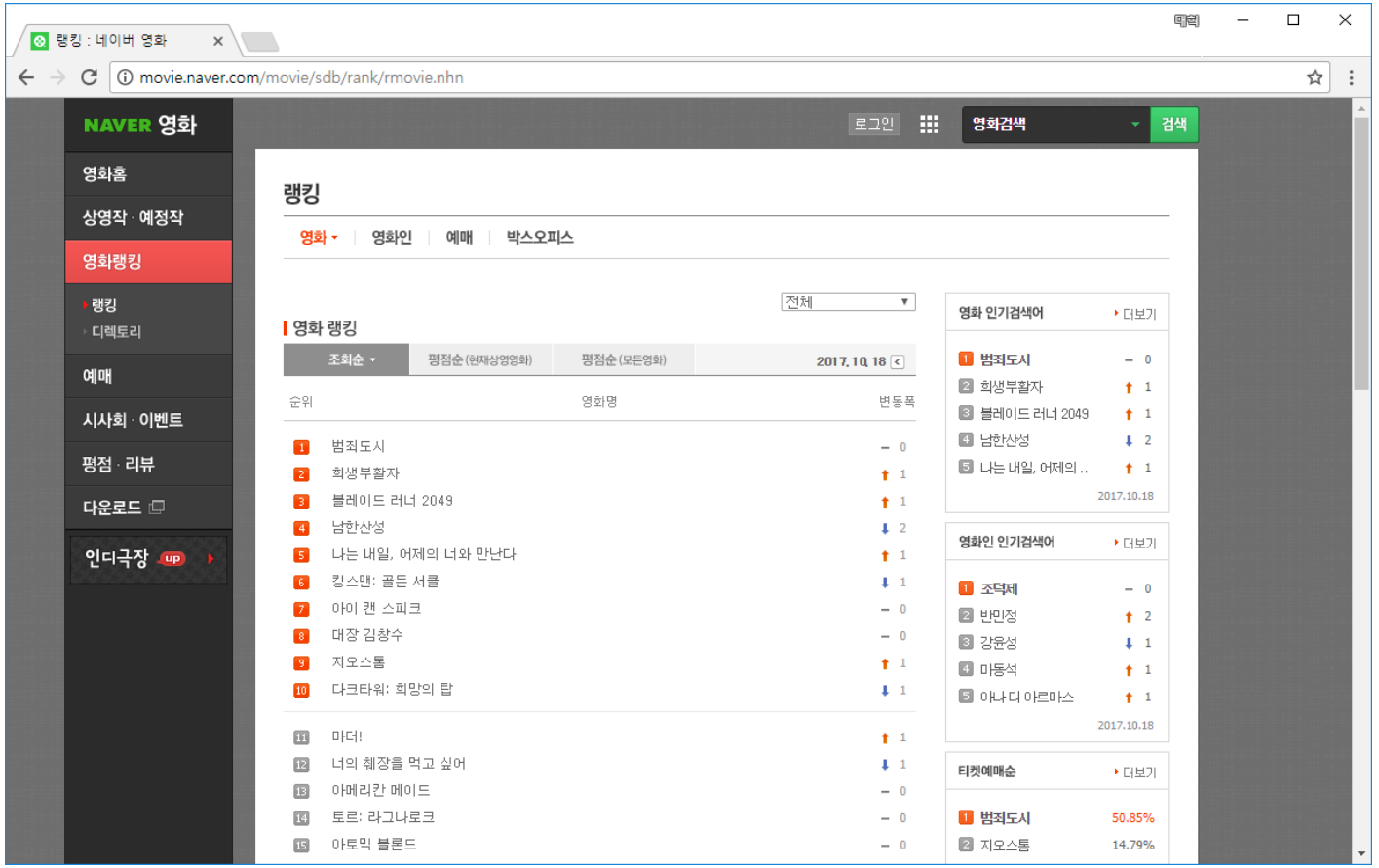
```
>> pip install beautifulsoup4
```

4) 테스트 하기

1. tag 조회
2. 속성값
3. Attributes 조회

3.1.2 예제 Naver 영화랭킹 분석(http://movie.naver.com/movie/sdb/rank/rmovie.nhn)

1) urllib.request 모듈이용 html 크롤링



3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

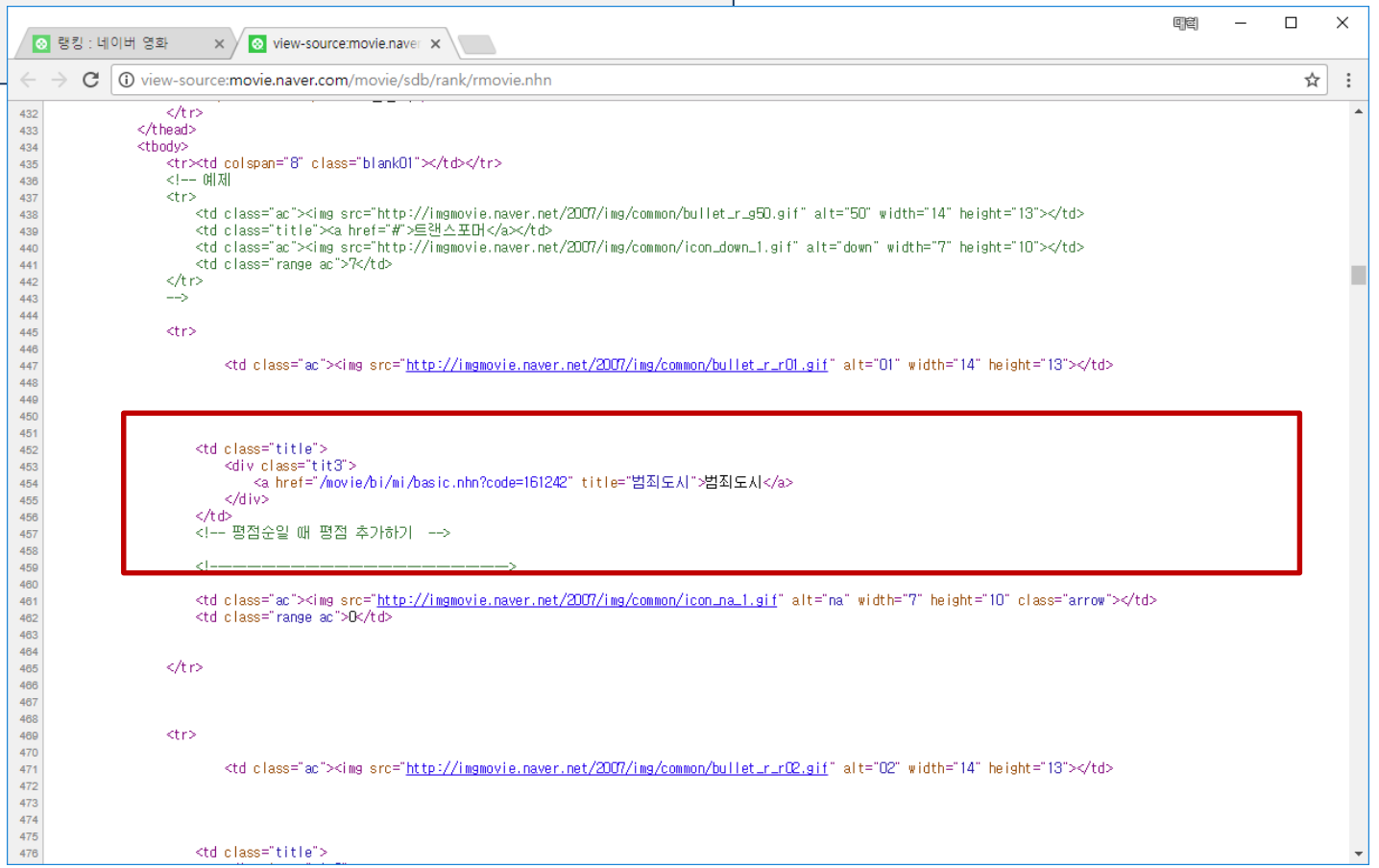
```
from urllib.request import Request, urlopen
from bs4 import BeautifulSoup

request = Request('http://movie.naver.com/movie/sdb/rank/rmovie.nhn')
resp = urlopen(request)
html = resp.read().decode('cp949')

bs = BeautifulSoup(html, 'html.parser')
print(bs.prettify())
```

```
<!DOCTYPE html>
<html>
 <head>
  <meta content="text/html; charset=utf-8" http-equiv="Content-Type"/>
  <meta content="IE=edge" http-equiv="X-UA-Compatible"/>
  <meta content="http://imgmovie.naver.com/today/naverme/naverme_profile.jpg" property="me2:image">
   <meta content="네이버영화 " property="me2:post_tag">
    <meta content="네이버영화" property="me2:category1"/>
    <meta content="http://movie.naver.com/movie/sdb/rank/rmovie.nhn" property="og:url"/>
     .
     .
     .
    </script>
   <!-- //Footer -->
  </div>
 </body>
</html>
```

3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

2) html 구조 확인을 통해 관심 데이터 확인

```
<td class="title">
    <div class="tit3">
        <a href="/movie/bi/mi/basic.nhn?code=161242" title="범죄도시">범죄도시</a>
    </div>
</td>
```



3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

2) BeautifulSoup 객체에 html.parser를 이용한 html 파싱

```
tags = bs.findAll('div',attrs={'class': 'tit3'})
print(tags)
```

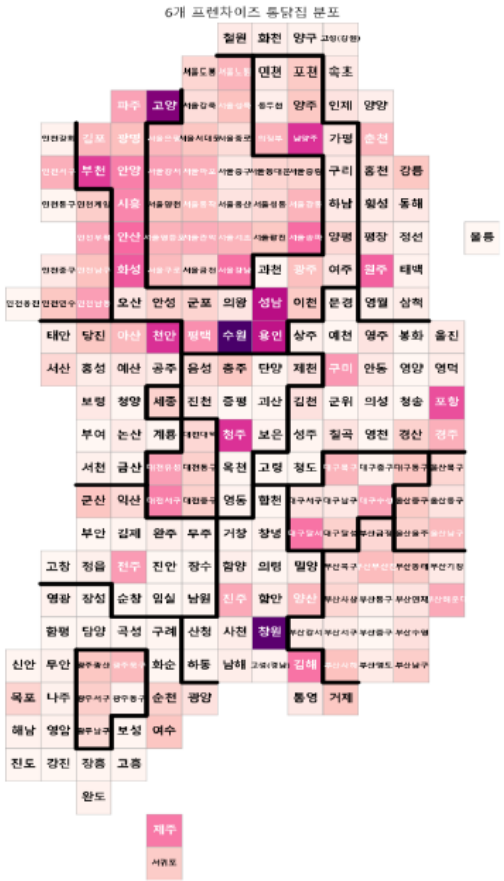
3) 찾아낸 tag 객체 내부의 제목, 링크 추출하기

```
for index, tag in enumerate(tags):
    print(index, tag.a.text, tag.a['href'], sep=' : ')
```

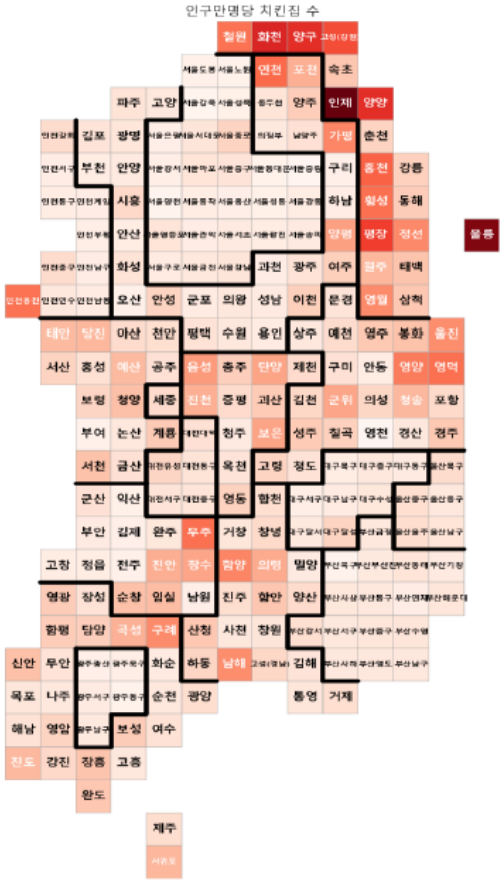
```
0 : 범죄도시 : /movie/bi/mi/basic.nhn?code=161242
1 : 희생부활자 : /movie/bi/mi/basic.nhn?code=140696
2 : 블레이드 러너 2049 : /movie/bi/mi/basic.nhn?code=88227
3 : 남한산성 : /movie/bi/mi/basic.nhn?code=150637
4 : 나는 내일, 어제의 너와 만난다 : /movie/bi/mi/basic.nhn?code=157178
5 : 킹스맨: 골든 서클 : /movie/bi/mi/basic.nhn?code=149747
6 : 아이 캔 스피크 : /movie/bi/mi/basic.nhn?code=161850
7 : 대장 김창수 : /movie/bi/mi/basic.nhn?code=154353
8 : 지오스톰 : /movie/bi/mi/basic.nhn?code=129095
9 : 다크타워: 희망의 탑 : /movie/bi/mi/basic.nhn?code=146407
10 : 마더! : /movie/bi/mi/basic.nhn?code=152650
11 : 너의 췌장을 먹고 싶어 : /movie/bi/mi/basic.nhn?code=159830
12 : 아메리칸 메이드 : /movie/bi/mi/basic.nhn?code=137945
.
.
.
```

3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

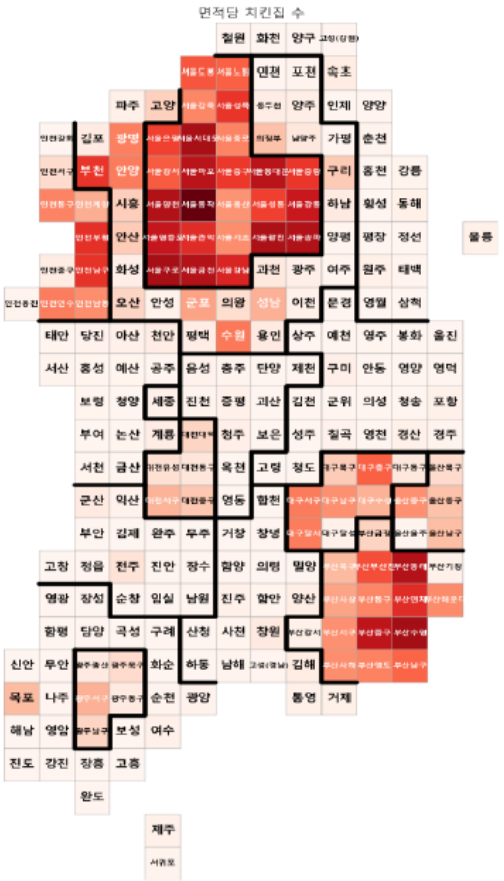
3.1.3 국내 5대 치킨 프랜차이즈 매장 정보 분석



치킨매장 분포도



인구 만명당 치킨집 수



면적대비 치킨집 수

3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

1) BBQ 매장정보 가져오기

url: https://www.bbq.co.kr/shop/shop_ajax.asp?page=1&pagesize=2000&gu=&si=

https://www.bbq.co.kr/sl

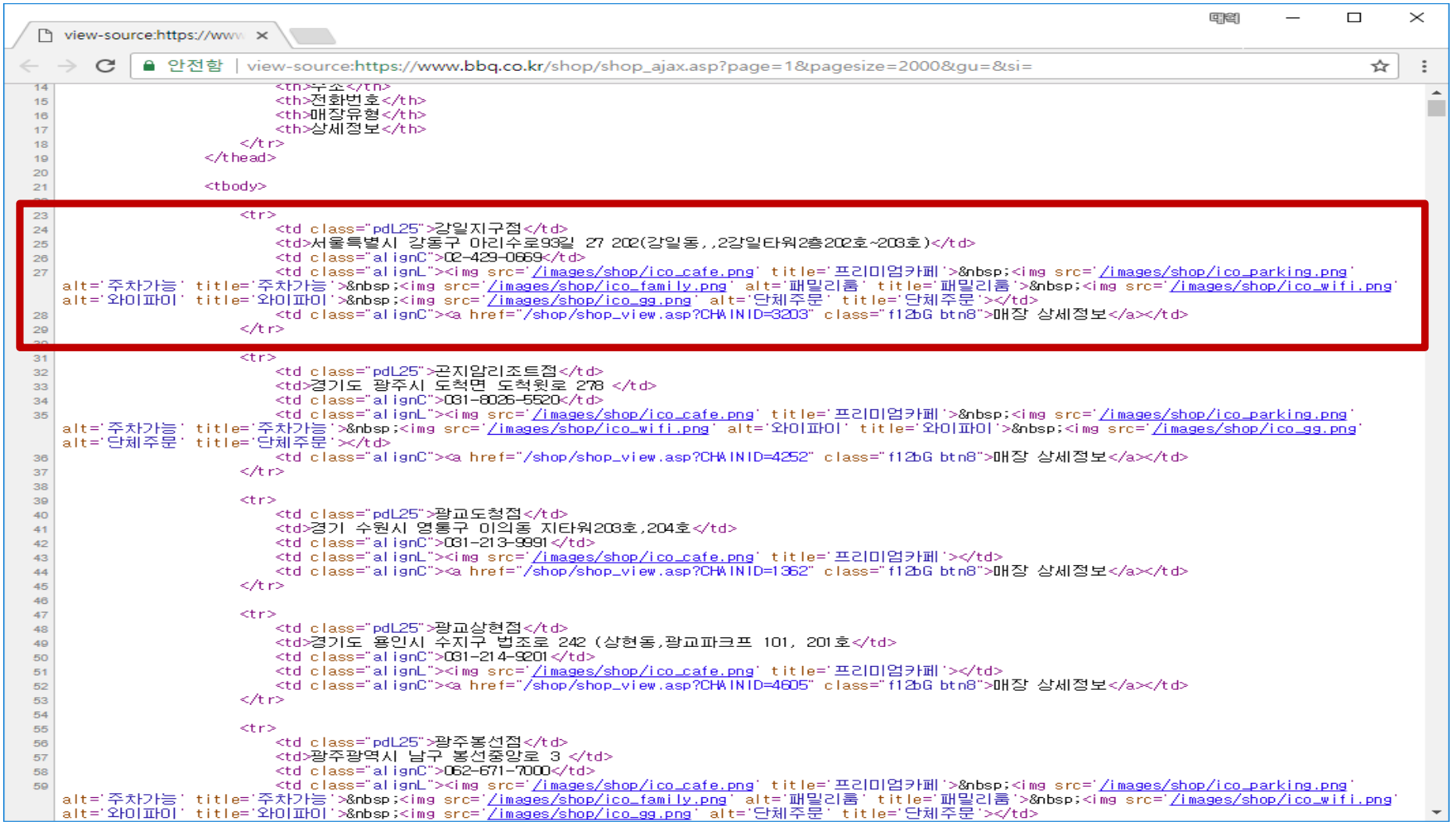
안전함 | https://www.bbq.co.kr/shop/shop_ajax.asp?page=1&pagesize=2000&gu=&si=

매장명	주소	전화번호	매장유형	상세정보
강일지구점	서울특별시 강동구 아리수로93길 27 202(강일동,,2강일타워2층202호~203호)	02-429-0669	P 주차 화장실 와이파이 G	매장 상세정보
곤지암리조트점	경기도 광주시 도척면 도척윗로 278	031-8026-5520	P 주차 와이파이 G	매장 상세정보
광교도청점	경기 수원시 영통구 이의동 지타워203호,204호	031-213-9991	P	매장 상세정보
광교상현점	경기도 용인시 수지구 법조로 242 (상현동,광교파크프 101, 201호	031-214-9201	P	매장 상세정보
광주봉선점	광주광역시 남구 봉선중앙로 3	062-671-7000	P 주차 화장실 와이파이 G	매장 상세정보
광주수완장덕점(P-cafe)(3)	광주광역시 광산구 풍영로 277 (장덕동) 1층	062-952-0088	P	매장 상세정보
광주치평본점	광주광역시 서구 상무자유로 173 치평동,1,2층	062-384-9282	P 주차 화장실 와이파이 G	매장 상세정보
교대본점	서울특별시 서초구 남부순환로 339길 63(서초동 1층)	02-584-9009	P	매장 상세정보
군산수송스타점	전라북도 군산시 수송로 213 (수송동) 2층	063-465-9543	P	매장 상세정보
나주혁신점	전라남도 나주시 상야2길 7 (빛가람동) 114(중흥S-클래스메가티움1차1층)	061-337-9919	P	매장 상세정보
남양주묵현점	경기도 남양주시 화도읍 먹갓로28번길 14	031-593-9285	P	매장 상세정보
논산점	충청남도 논산시 중앙로 271 (취암동)	041-736-7367	P 화장실 와이파이 G	매장 상세정보
논산중앙점	충청남도 논산시 번영로17번길 10-11 (취암동) 1층 101호	041-734-9200	P	매장 상세정보
대전둔산사학연금점	대전광역시 서구 한밭대로 809 (둔산동) 1층	042-483-9282	P 주차 화장실 와이파이 G	매장 상세정보
대전둔산점	대전광역시 서구 둔산남로105번길 18 (둔산동) 102(둔산동)	042-471-9282	P 주차 화장실 와이파이 G	매장 상세정보
대전중리행운점	대전광역시 대덕구 한밭대로 1141 (중리동, 지성빌딩)	042-633-9282	P	매장 상세정보

3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

html 크롤링

1. 데이터 확인



3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

html 크롤링

2. 파싱 처리 (데이터 추출) 함수 작성

```
def proc_bbq(html)
```

3. 데이터 저장 함수 작성

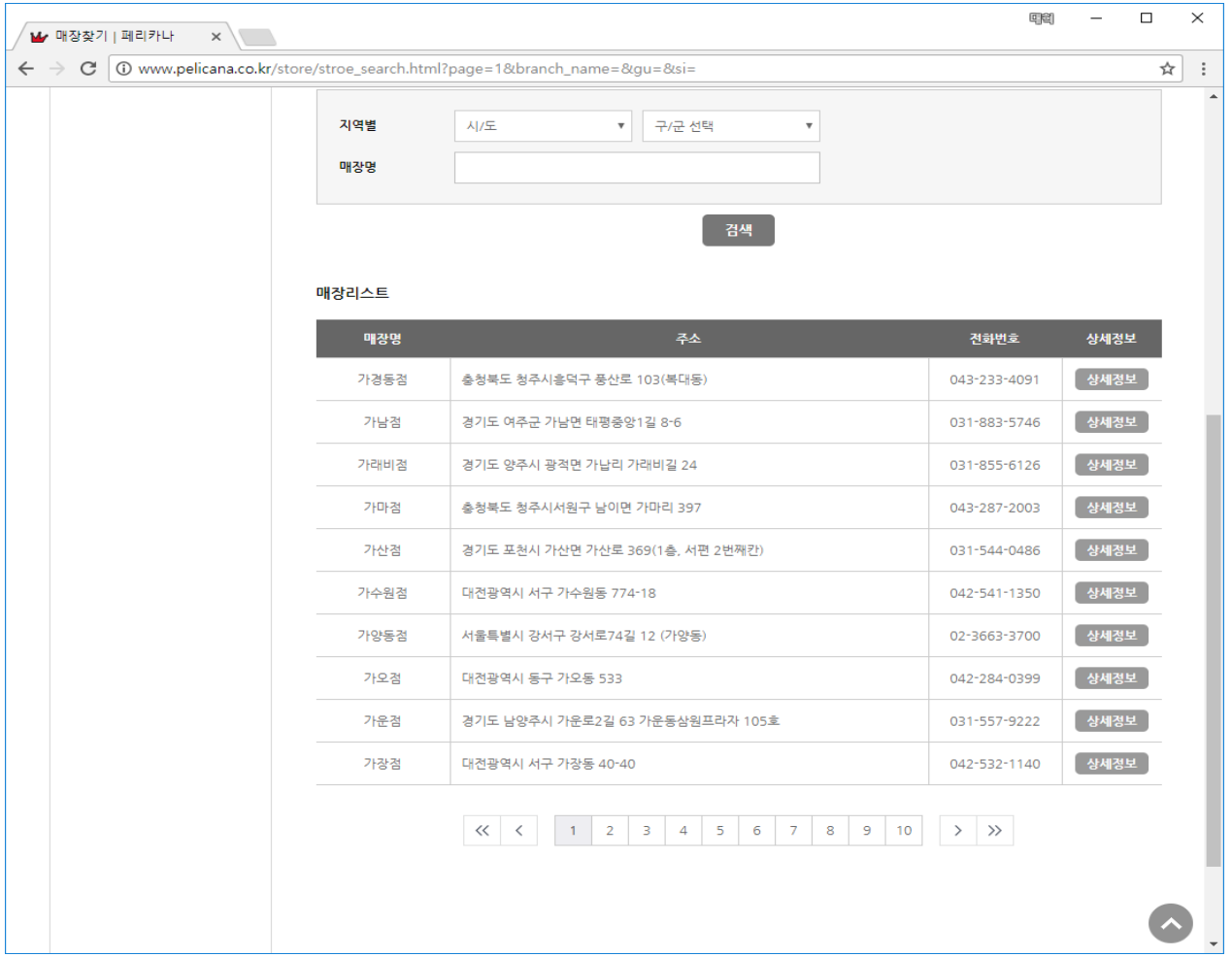
```
def store_bbq(data):
```

- 서울시, 서울시특별시와 같은 여러 표시의 시도명에 대한 하나의 시도명 처리가 필요하다.
- 행정구역 변경에 따른 구군에 대한 처리도 필요하다.
- 데이터 중복을 위한 처리가 필요하다.
- 저장 파일 포맷은 csv 파일(bbq_table.csv)로 Pandas DataFrame에서 제공하는 기능을 사용한다.

3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

2) 페리카나 매장정보 가져오기

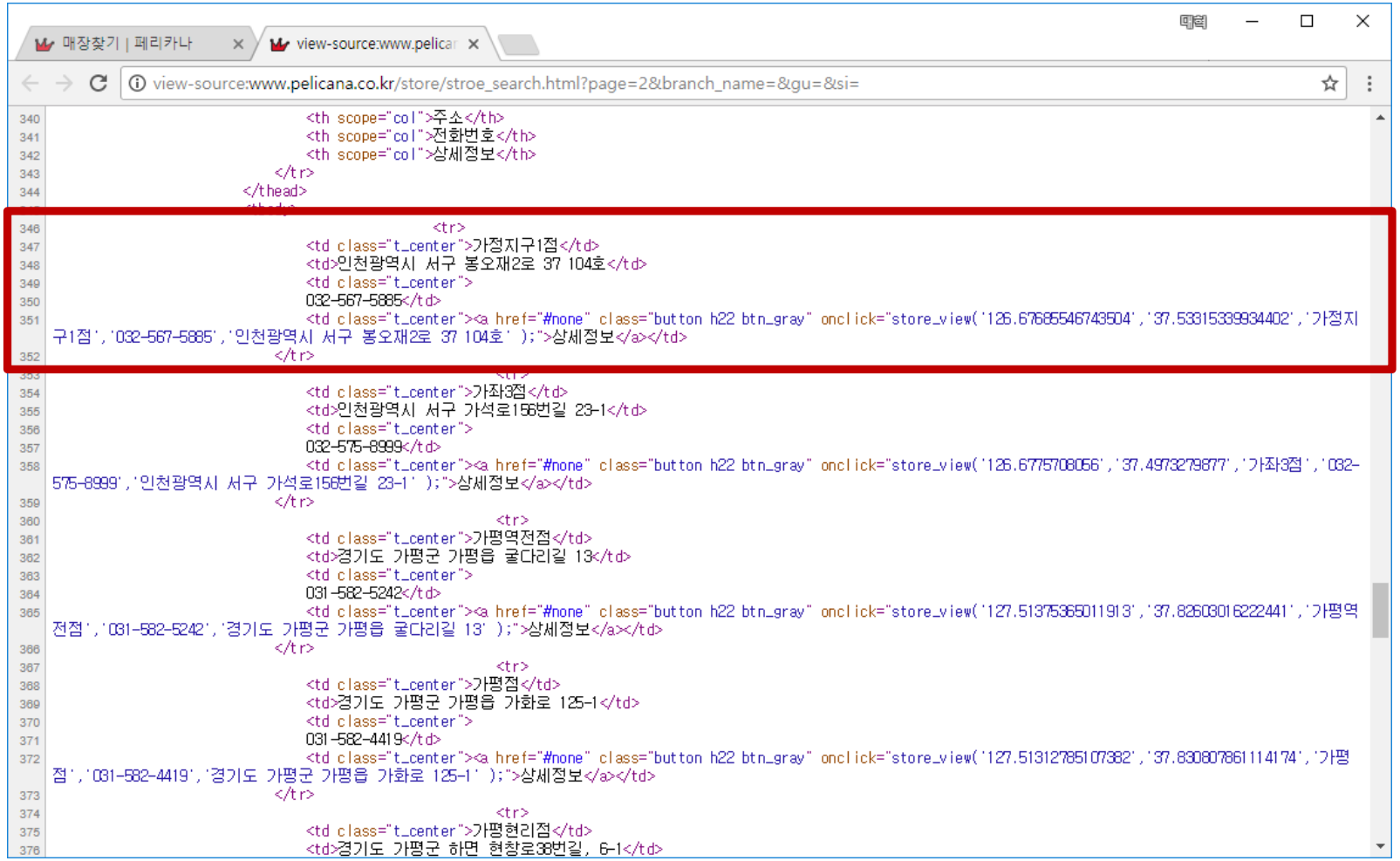
url: http://www.pelicana.co.kr/store/stroe_search.html?branch_name=&gu=&si=&page=1



3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

html 크롤링

1. 데이터 확인



3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

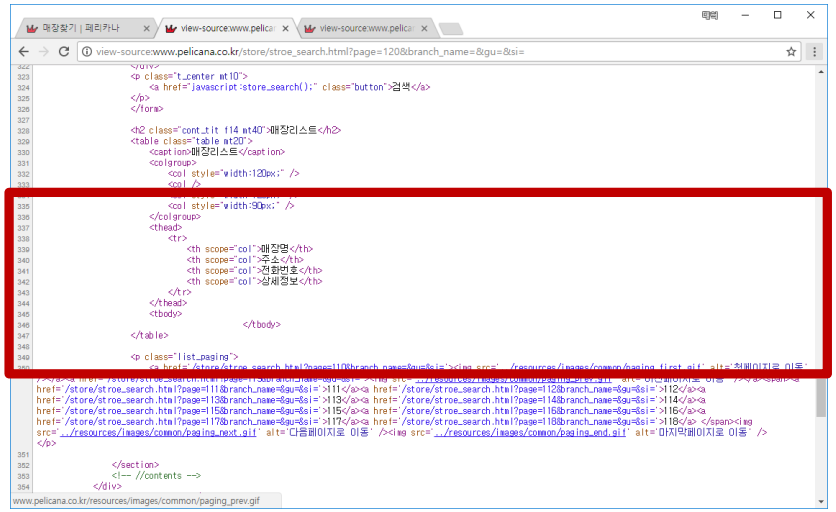
html 크롤링

2. 하나의 함수에 크롤링과 파싱 처리, 저장을 함께 한다.

def crawling_pelicana()

- paging 처리가 되어야 한다.

  무한루프를 사용하고 마지막 페이지(데이터가 없는) html소스를 확인하면 다음과 같다

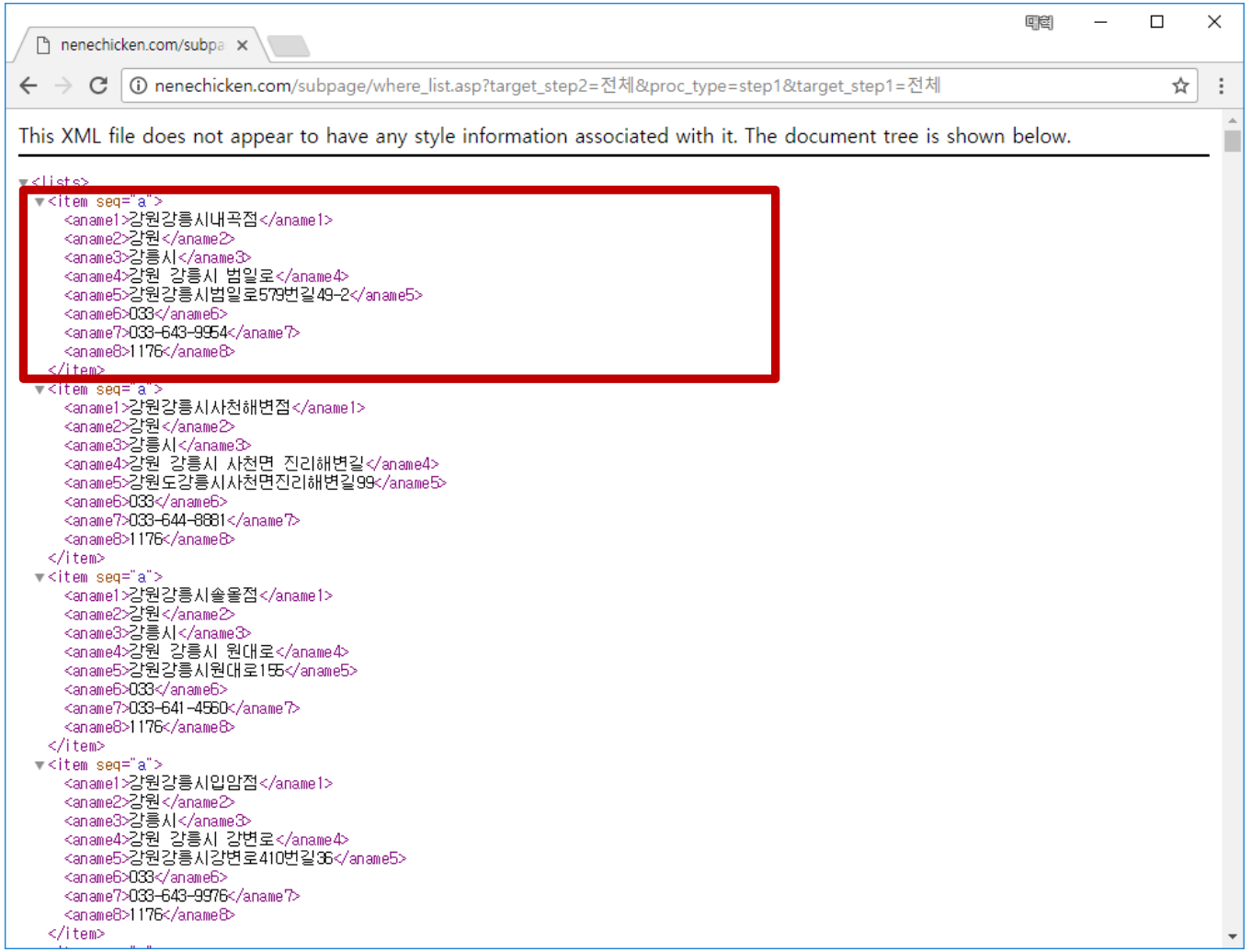


- 서울시, 서울시특별시와 같은 여러 표시의 시도명에 대한 하나의 시도명 처리가 필요하다.
- 행정구역 변경에 따른 구군에 대한 처리도 필요하다.
- 데이터 중복을 위한 처리가 필요하다.
- 저장 파일 포맷은 csv 파일(pelicana_table.csv)로 Pandas DataFrame에서 제공하는 기능을 사용한다.

3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

3) 네네치킨 매장정보 가져오기

url: http://nenechicken.com/subpage/where_list.asp?target_step2=전체&proc_type=step1&target_step1=전체 (XML 응답)



3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

html 크롤링

1. 파싱 처리 (데이터 추출) 함수 작성

```
def proc_nene(xml)
```

- 파싱을 위해 XML Paser ElementTree 를 사용한다.

2. 데이터 저장 함수 작성

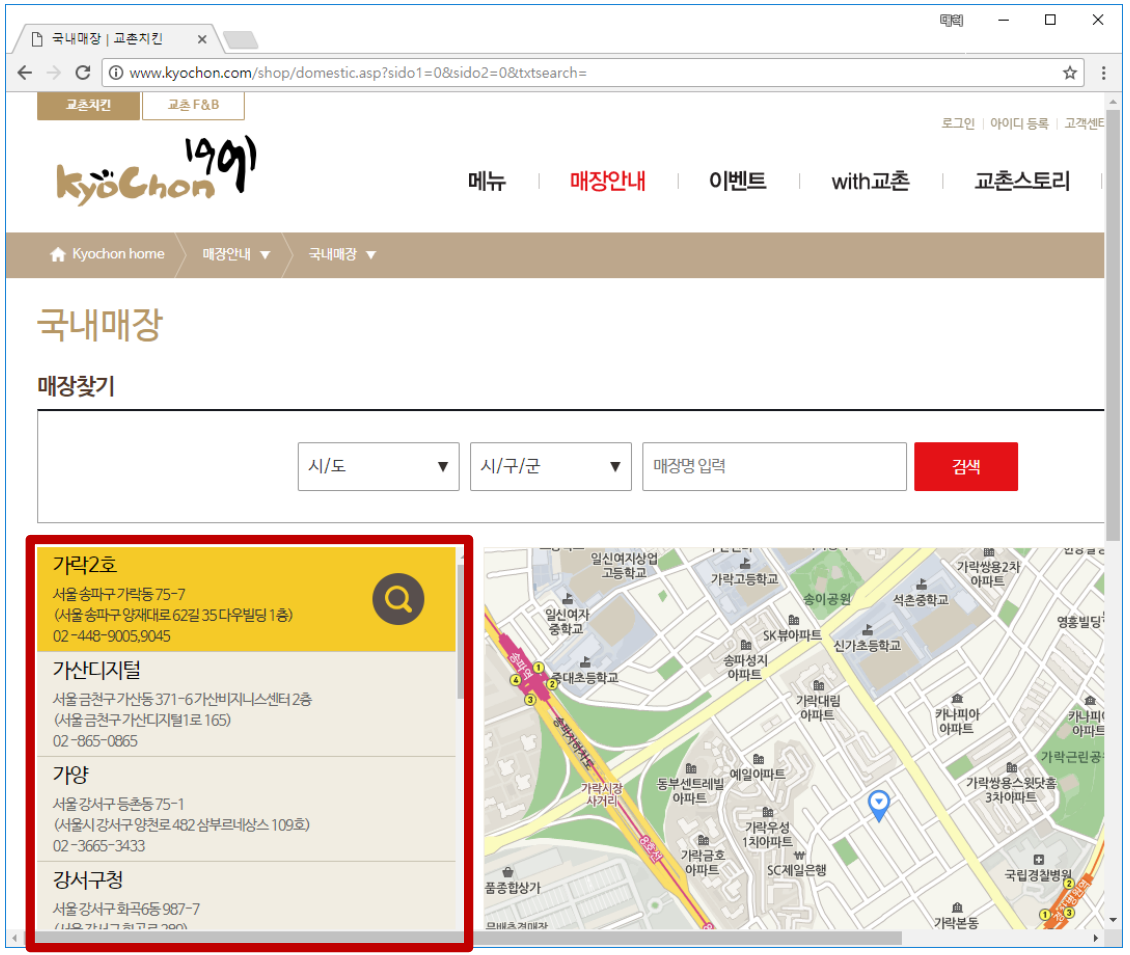
```
def store_nene(data):
```

- 서울시, 서울시특별시와 같은 여러 표시의 시도명에 대한 하나의 시도명 처리가 필요하다.
- 행정구역 변경에 따른 구군에 대한 처리도 필요하다.
- 데이터 중복을 위한 처리가 필요하다.
- 저장 파일 포맷은 csv 파일(nene_table.csv)로 Pandas DataFrame에서 제공하는 기능을 사용한다.

3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

4) 교촌치킨 매장정보 가져오기

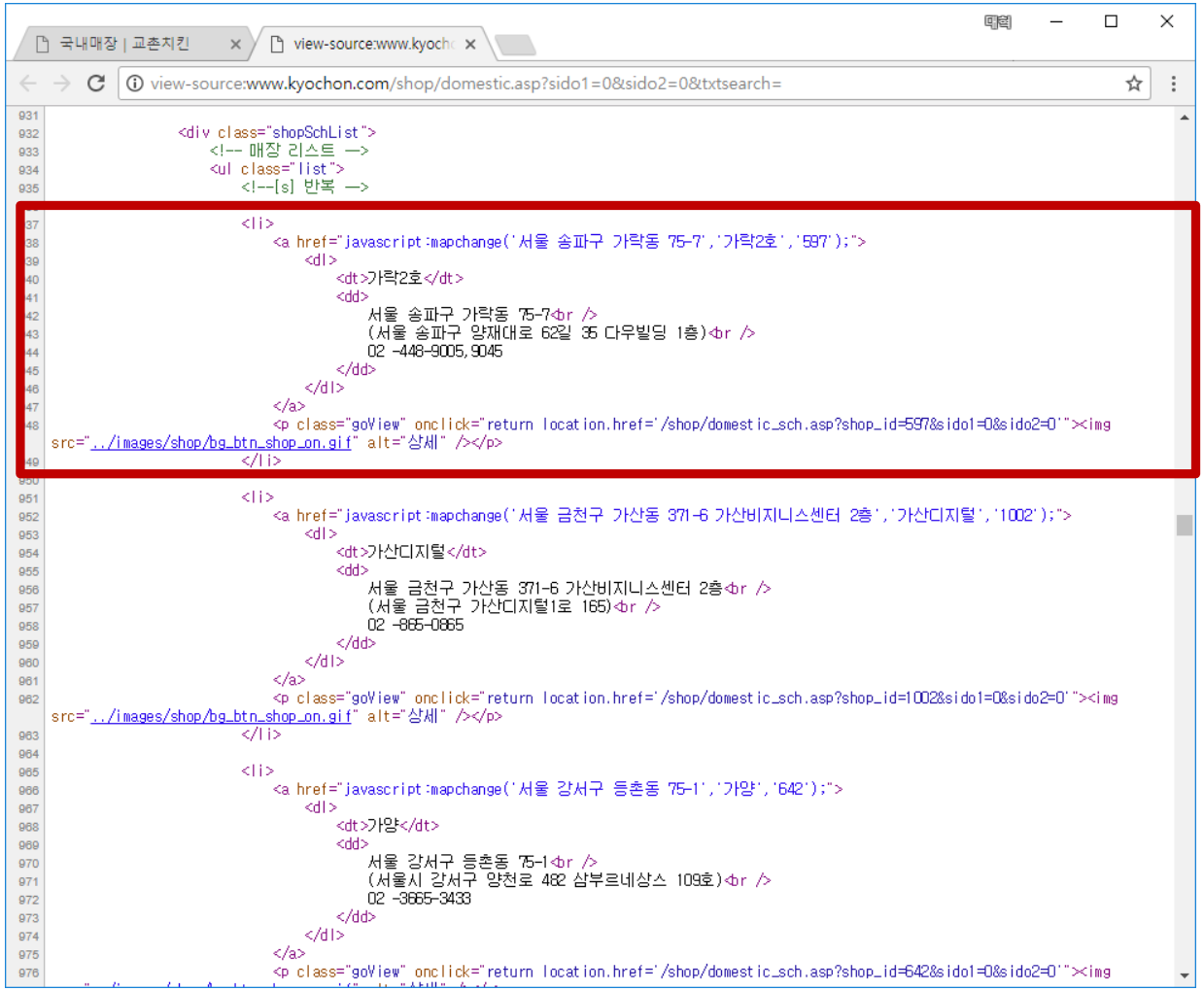
url: http://www.kyochon.com/shop/domestic.asp?sido1=0&sido2=0&txtsearch=



3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

html 크롤링

1. 데이터 확인



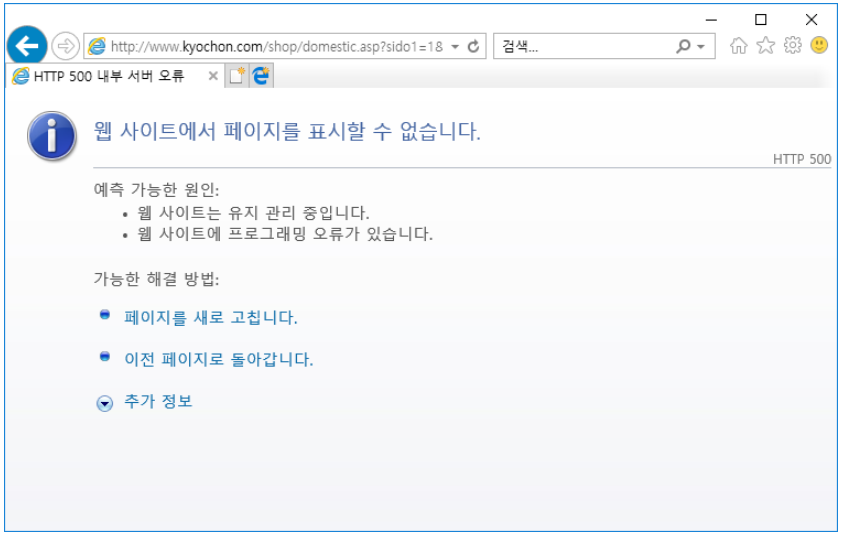
3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

html 크롤링

2. 하나의 함수에 크롤링과 파싱 처리, 저장을 함께 한다.

def crawling_kyouchon()

- paging 처리가 되어야 한다.
  파라미터 중 sido1, sido2 가 정확하지 않으면 500 오류(HttpError) 가 발생 한다.
  따라서 html = crawler.crawling(url=url) 에서 html 이 None인지 확인해서 마지막이 지났는 지 검사한다.

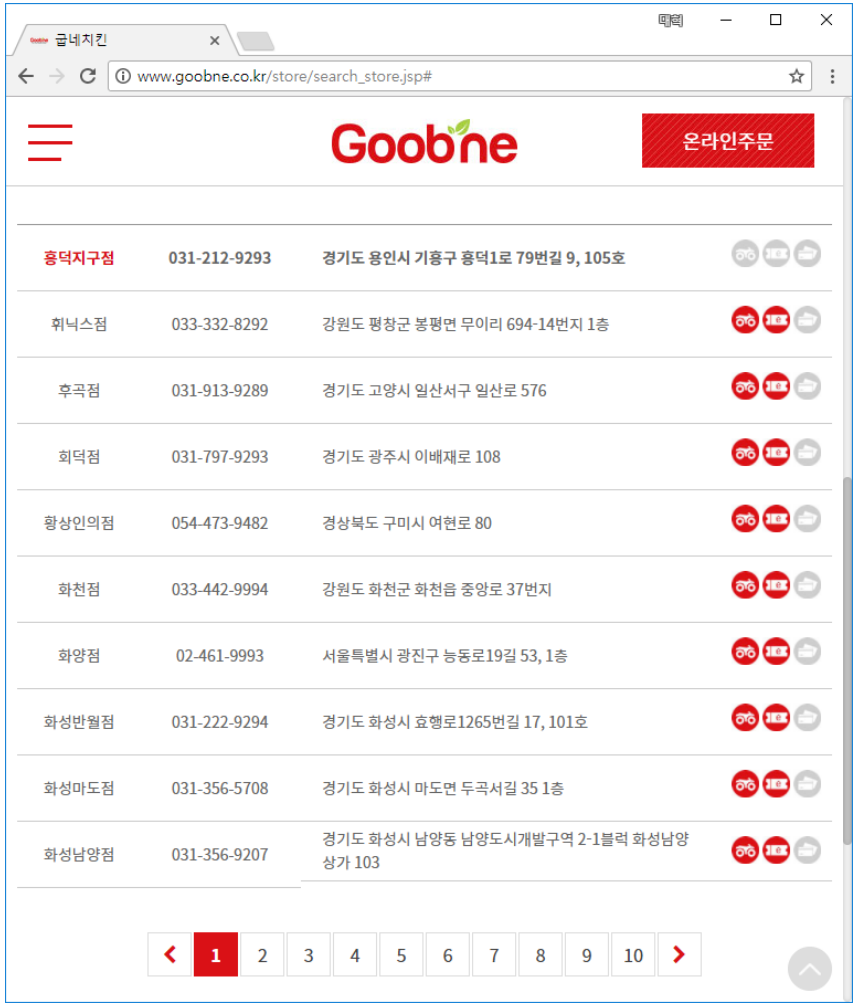


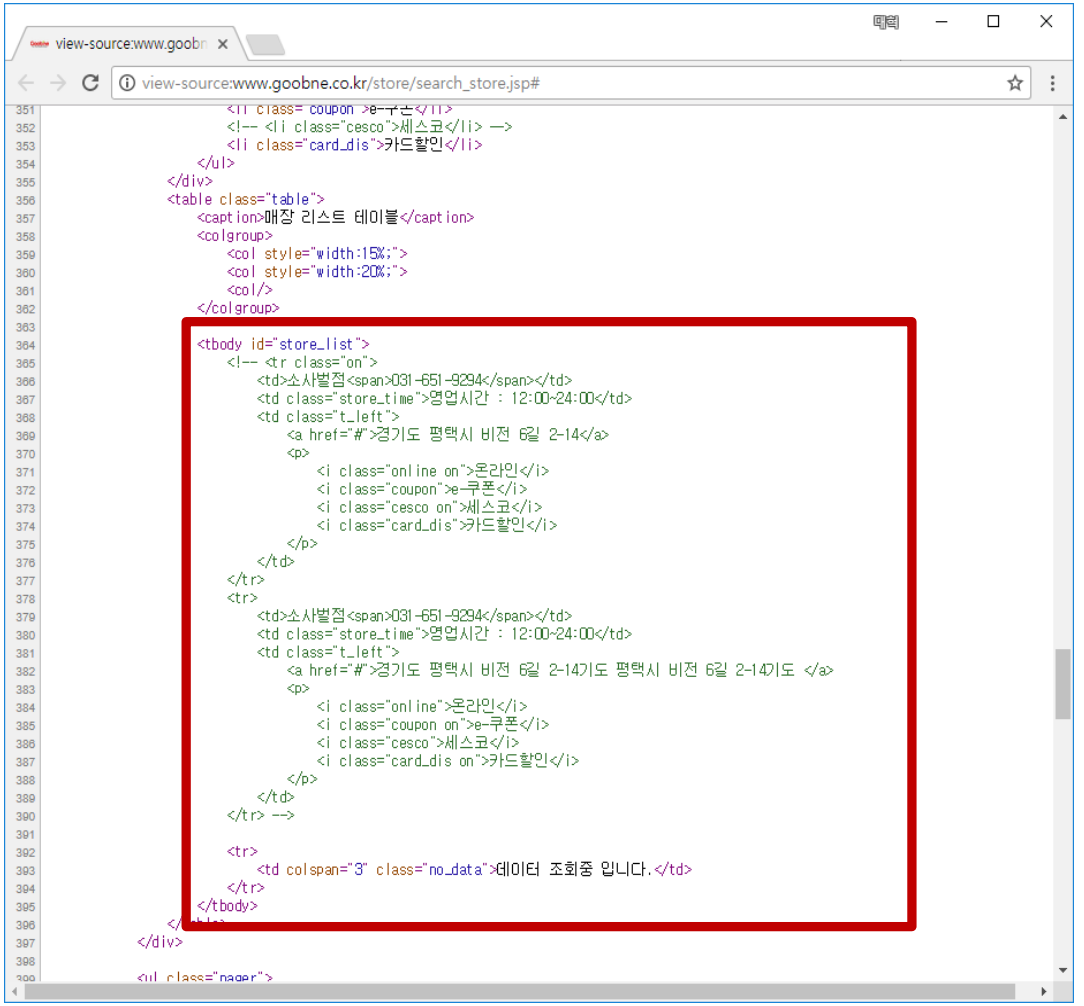
- 서울시, 서울시특별시와 같은 여러 표시의 시도명에 대한 하나의 시도명 처리가 필요하다.
- 행정구역 변경에 따른 구군에 대한 처리도 필요하다.
- 데이터 중복을 위한 처리가 필요하다.
- 저장 파일 포맷은 csv 파일(kyochon_table.csv)로 Pandas DataFrame에서 제공하는 기능을 사용한다.

3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

5) 굽네치킨 매장정보 가져오기

url: http://www.goobne.co.kr/store/search_store.jsp





3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

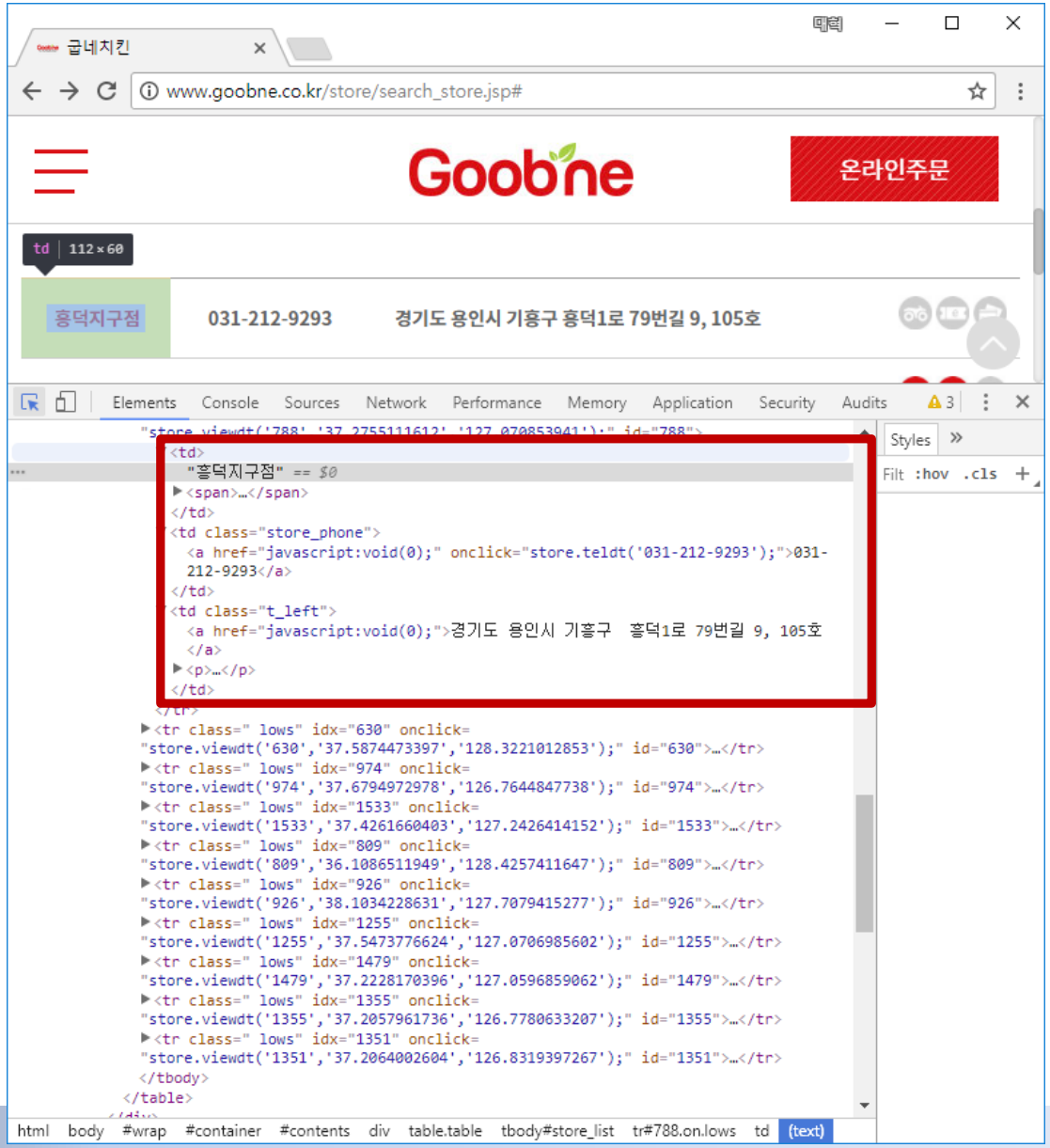
html 크롤링

1. 브라우저 화면에서 보여지는 내용이 소스보기에서 <tbody> 안에 없다.

2. tbody 안의 내용이 동적으로(JS, ajax 통신) 생성됨을 알 수 있다.

3. JS 코드를 분석해서 내부 ajax 통신하는 url 및 파라미터를 맞추어 JSON 데이터를 받아 오는 방법이 있다. url과 파라미터만 찾으면 JSON으로 비교적 복잡하지 않게 크롤링이 가능하다.

4. 파이썬 코드에서 selenium 라이브러리를 사용하여 직접 화면의 자바스크립트를 실행 시키는 방법이 있다.



3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

Selenium

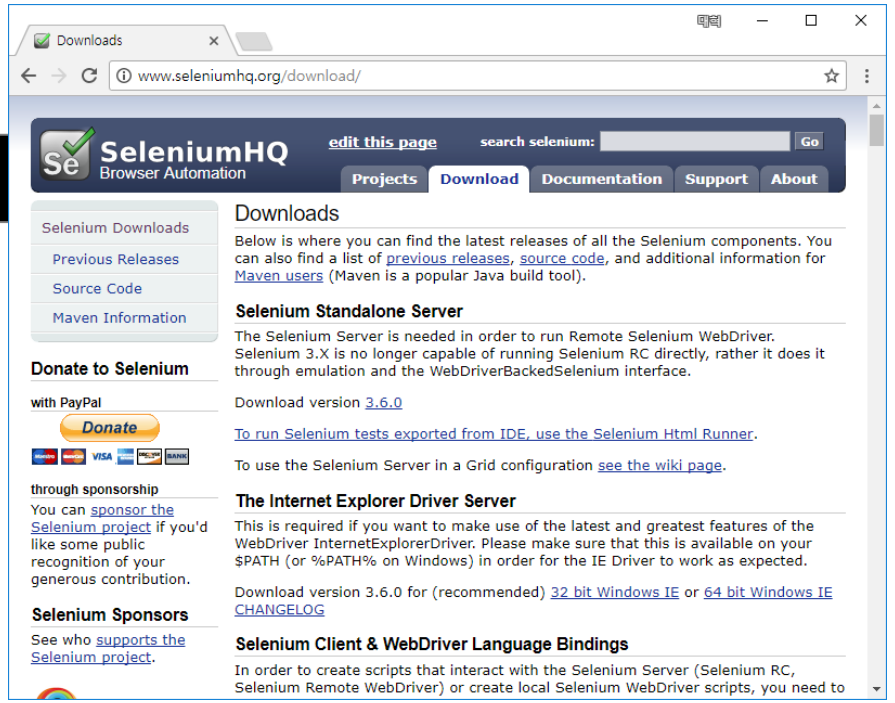
1. 원래 웹 사이트 UI(화면) 테스트 목적으로 제작된 라이브러리
2. 테스트 절차에 따라 개발한 웹 화면들을 브라우저에서 직접 동작시키고 화면을 캡쳐하여 확인하는 테스트 용도로 활용된다.
3. 브라우저에서 동작 시킬 때 자바스크립트 코드 동작도 가능하기 때문에 동적 웹페이지 테스트에 필수 도구 이다.
4. Selenium 파이썬 라이브러리 자체는 브라우저를 포함하고 있지 않다.
5. WebDriver 라는 인터페이스와 함께 구동된다.
6. WebDriver는 운영체제 및 브라우저에 맞게 다운로드 받아 설치해야 한다.

Selenium 설치

```
>> pip install selenium
```

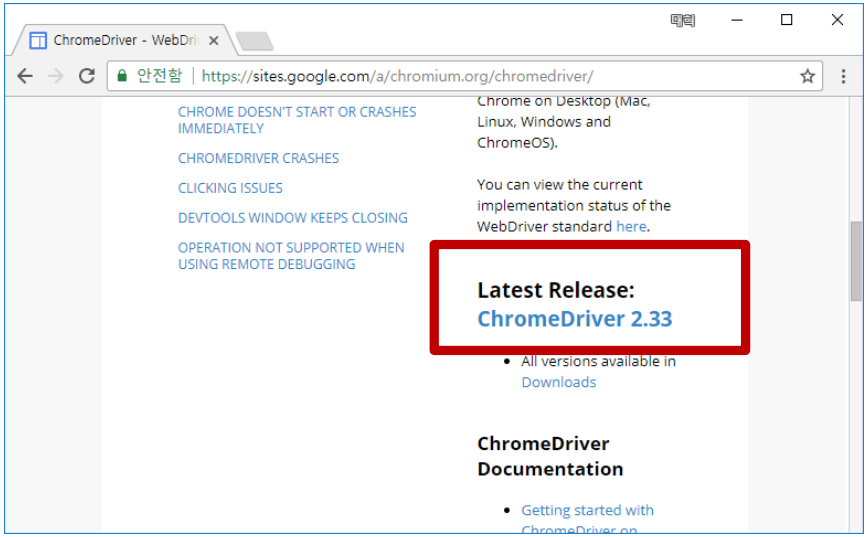
설치가 완료되면 사용할 브라우저에 맞는 WebDriver 다운로드 한다.

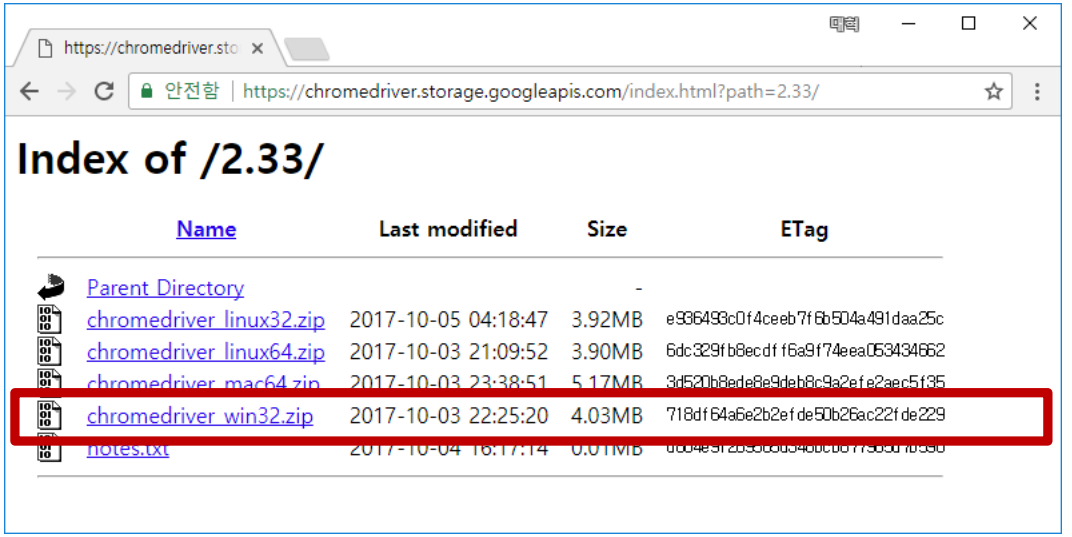
http://www.seleniumhq.org/download 로 이동

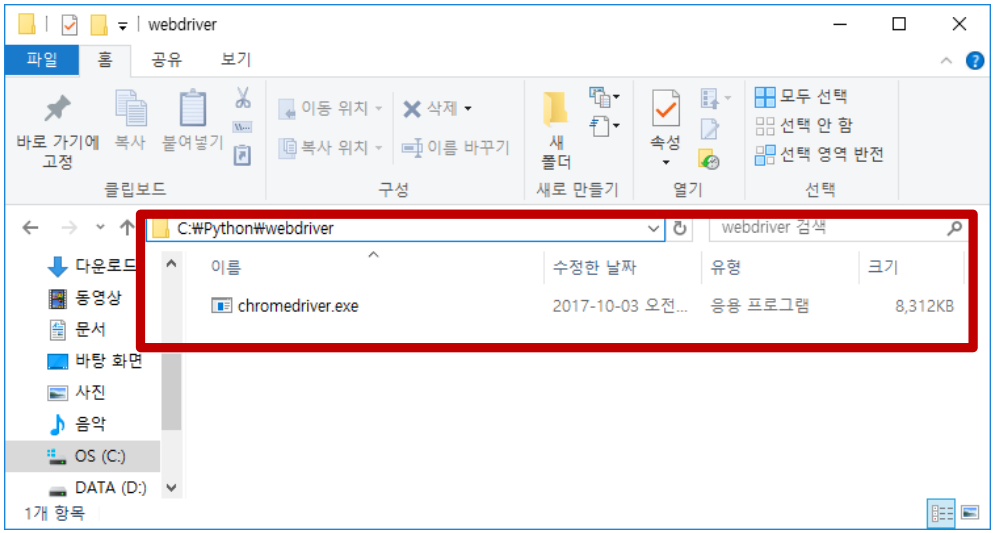


3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

ChromeDriver 다운로드 및 설치



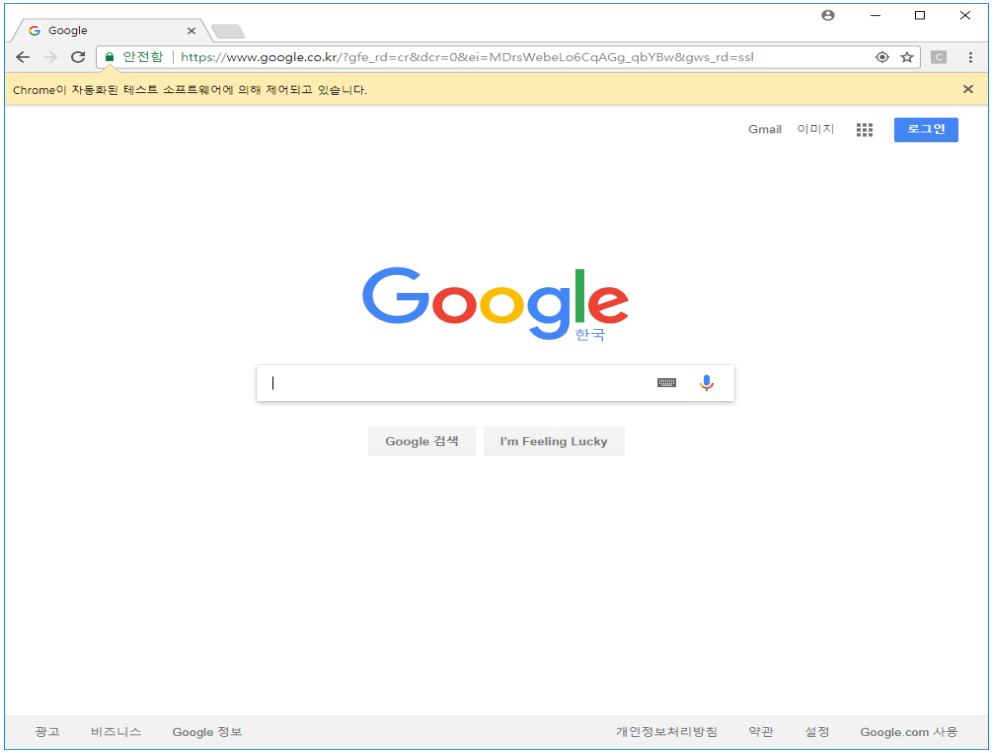




3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

Selenium 테스트

```
from selenium import webdriver

wd = webdriver.Chrome('c:/python/webdriver/chromedriver.exe')
wd.get('http://www.google.com')
```



3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

html 크롤링

5. 하나의 함수에 크롤링과 파싱 처리, 저장을 함께 한다.

def crawling_goobne()

- 매장 검색 첫 페이지를 로딩하고 store_pageList(nPage) 자바스크립트 코드를 실행 시킨다(nPage=1 부터)
- paging 처리가 되어야 한다.
  마지막 페이지는 tbody 안에 tr 태그들이 존재하지 않는다. 이 것으로 마지막 페이지를 검출한다.
- 서울시, 서울시특별시와 같은 여러 표시의 시도명에 대한 하나의 시도명 처리가 필요하다.
- 행정구역 변경에 따른 구군에 대한 처리도 필요하다.
- 데이터 중복을 위한 처리가 필요하다.
- 저장 파일 포맷은 csv 파일(goobne_table.csv)로 Pandas DataFrame에서 제공하는 기능을 사용한다.

6) 중복 데이터 확인 및 제거

다음 코드에서 중복 데이터를 확인해 보자

```
table = pd.DataFrame.from_csv('../__result__/crawling/bbq_table.csv', encoding='utf-8', index_col=0, header=0)

# 중복 row 확인
print(table)
table = table.drop_duplicates(subset='name', keep='first')
print(table.count())
```

5개 크롤링 작업에 반영한다.

7) 시/도 와 구/군 정리

데이터 딕션너리 파일을 가지고 반영 한다.

3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

8) 다음과 같은 머지 테이블을 만든다.

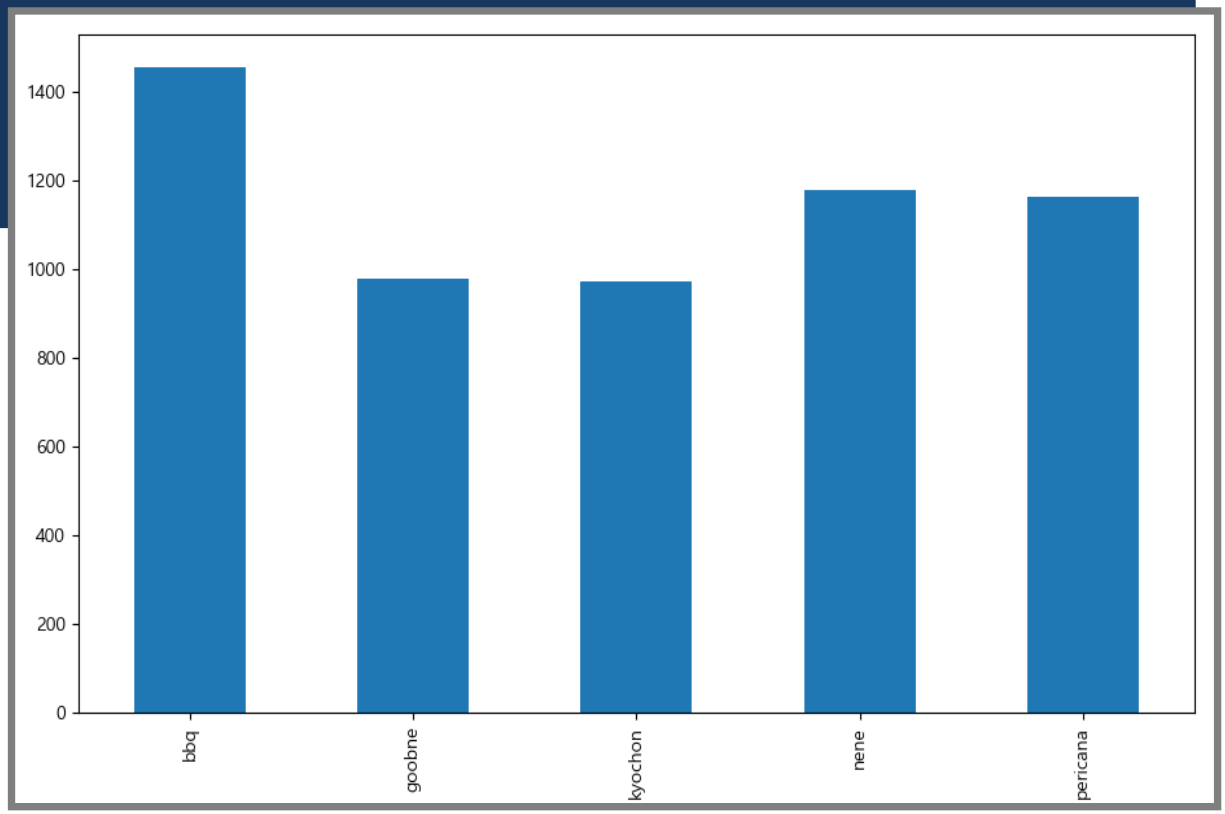
	bbq	goobne	kyochon	nene	pericana
강원도 강릉시	3.0	5.0	5.0	8.0	13.0
강원도 고성군	3.0	2.0	1.0	0.0	3.0
강원도 동해시	3.0	2.0	2.0	4.0	6.0
강원도 삼척시	2.0	1.0	1.0	1.0	2.0
강원도 속초시	2.0	2.0	3.0	3.0	4.0
강원도 양구군	2.0	1.0	1.0	3.0	0.0
강원도 양양군	4.0	2.0	1.0	2.0	1.0
강원도 영월군	1.0	0.0	1.0	1.0	3.0
강원도 원주시	15.0	6.0	9.0	9.0	27.0
강원도 인제군	4.0	2.0	1.0	4.0	3.0
강원도 정선군	1.0	1.0	1.0	1.0	5.0
강원도 철원군	2.0	1.0	0.0	6.0	4.0
.	.	.	.	.	.
.	.	.	.	.	.

- sido, gungu 가 비어 있는 경우 처리
- 결과를 보고 비정상적인 데이터는 삭제해야 한다.

3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

9) 다음 코드를 참고해서 머지 테이블 기반의 5개 치킨 프랜차이즈 매장 바그래프를 그려보자.

```
chicken_table = pd.DataFrame({'bbq': bbq, 'pericana': pericana, 'nene': nene, 'kyochon': kyochon, 'goobne': goobne}).fillna(0)
chicken_table = chicken_table.₩
    drop(chicken_table[chicken_table.index == '00 18'].index).₩
    drop(chicken_table[chicken_table.index == '테스트 테스트구'].index)
chicken_table.to_csv('../__result__/crawling/chicken_table.csv', encoding="utf-8", mode='w', index=True)

plt.figure()
chicken_table.sum(axis=0).iloc[:5].plot(kind='bar')
plt.show()
```



3. 웹서비스 데이터를 활용한 지리정보 기반 시각화

9) 다음 코드를 참고해서 우리나라 지역별 인구수, 면적과 앞의 치킨 매장 정보와 머지 한다.

```
data_draw_korea = pd.read_csv('data_draw_korea.csv', index_col=0, encoding='UTF-8')
data_draw_korea.index = data_draw_korea.apply(lambda r: r['광역시도'] + ' ' + r['행정구역'], axis=1)

chicken = pd.merge(data_draw_korea, chicken_table, how='outer', left_index=True, right_index=True)
chicken = chicken[~np.isnan(chicken['면적'])].fillna(0)

chicken['total'] = chicken_table.sum(axis=1)
chicken = chicken[~np.isnan(chicken['total'])].fillna(0)
```

10) 다음 코드는 블록맵을 그리는 showmap 함수다.

```
def showmap(blockedmap, targetdata, title, color):

    BORDER_LINES = [
        [(3, 2), (5, 2), (5, 3), (9, 3), (9, 1)],  # 인천
        [(2, 5), (3, 5), (3, 4), (8, 4), (8, 7), (7, 7), (7, 9), (4, 9), (4, 7), (1, 7)],  # 서울
        [(1, 6), (1, 9), (3, 9), (3, 10), (8, 10), (8, 9),
         (9, 9), (9, 8), (10, 8), (10, 5), (9, 5), (9, 3)],  # 경기도
        [(9, 12), (9, 10), (8, 10)],  # 강원도
        [(10, 5), (11, 5), (11, 4), (12, 4), (12, 5), (13, 5),
         (13, 4), (14, 4), (14, 2)],  # 충청남도
        [(11, 5), (12, 5), (12, 6), (15, 6), (15, 7), (13, 7),
         (13, 8), (11, 8), (11, 9), (10, 9), (10, 8)],  # 충청북도
        [(14, 4), (15, 4), (15, 6)],  # 대전시
        [(14, 7), (14, 9), (13, 9), (13, 11), (13, 13)],  # 경상북도
        [(14, 8), (16, 8), (16, 10), (15, 10),
         (15, 11), (14, 11), (14, 12), (13, 12)],  # 대구시
        [(15, 11), (16, 11), (16, 13)],  # 울산시
        [(17, 1), (17, 3), (18, 3), (18, 6), (15, 6)],  # 전라북도
        [(19, 2), (19, 4), (21, 4), (21, 3), (22, 3), (22, 2), (19, 2)],  # 광주시
        [(18, 5), (20, 5), (20, 6)],  # 전라남도
        [(16, 9), (18, 9), (18, 8), (19, 8), (19, 9), (20, 9), (20, 10)],  # 부산시
    ]

    whitelabelmin = (max(blockedmap[targetdata]) - min(blockedmap[targetdata])) * 0.25 + min(blockedmap[targetdata])

    vmin = min(blockedmap[targetdata])
    vmax = max(blockedmap[targetdata])
    mapdata = blockedmap.pivot(index='y', columns='x', values=targetdata)
    masked_mapdata = np.ma.masked_where(np.isnan(mapdata), mapdata)
    cmapname = color
    plt.figure(figsize=(8, 13))
```