빅데이터 처리 및 응용

자유 주제 보고서

과목:빅데이터 처리 및 응용

전공:정보융합학부

학번:2017204016

이름:이지환

교수님:박재성 교수님

우선 스레드를 돌릴때마다 selenium으로 455개의 위도 경도를 따오면 너무 시간이 오래걸려 코드를 크게 3가지로 나누어 작성하였습니다. 최종 실행때에는 main code만 실행하면 됩니다.

코드 1 설명

1. 미세먼지 측정소를 beautifulsoup으로 크롤링하고, selenium으로 구글 맵에 접속 해 455개의 측정소의 위도 경도를 따와 하나의 csv파일로 저장하기

   에어코리아에서 제공하는 dustfeel사이트에는 10개씩 46페이지로 455개의 미세먼지 측정소가 저장되어 있다. 이를 beautifulsoup을 이용해 따온다.

1. dustfeel 사이트에서 455개의 주소를 beautifulsoup을 통해 따오기

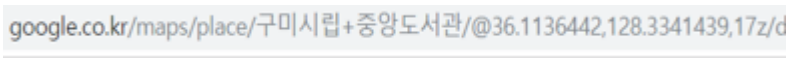
```
#dustfeel 사이트에서 455개의 주소를 beautifulsoup을 통해 따온다
from bs4 import BeautifulSoup
import pandas as pd
from urllib.request import urlopen


addr_list=[] #주소 저장 리스트
for i in range(46):# 455개 정보 가져옴 (한페이지당 10개의 대기정보가 저장되어 있음)

    url='https://dustfeel.com/station?p='+str(i+1)
    page=urlopen(url)
    soup=BeautifulSoup(page,'html.parser')
    addr_text=soup.find_all('p','addr')
    for j in range(len(addr_text)):
        addr_text[j]=addr_text[j].text#주소 문자열만 따옴(10개)

    addr_list.extend(addr_text)#주소가 저장된 리스트


#주소에서 끝에 대기 부분을 전처리 한다.
for i in range(len(addr_list)):
    if '도시대기' in addr_list[i]:
        addr_list[i]=addr_list[i].replace('도시대기','')
    elif '도로변대기' in addr_list[i]:
        addr_list[i]=addr_list[i].replace('도로변대기','')
    elif '옥상' in addr_list[i]:
        addr_list[i]=addr_list[i].replace('옥상','')
    elif '교외대기' in addr_list[i]:
        addr_list[i]=addr_list[i].replace('교외대기','')
```

2. 455개의 주소가 저장된 리스트를 가져와 selenium으로 크롬드라이버에 접속해 455개의 위도와 경도를 따온다.

2. selenium에서 google map에 접속에 455개의 주소를 입력하고 경도 위도를 리스트에 저장한다(크롤링하는데 30분 정도 소요됩니다)

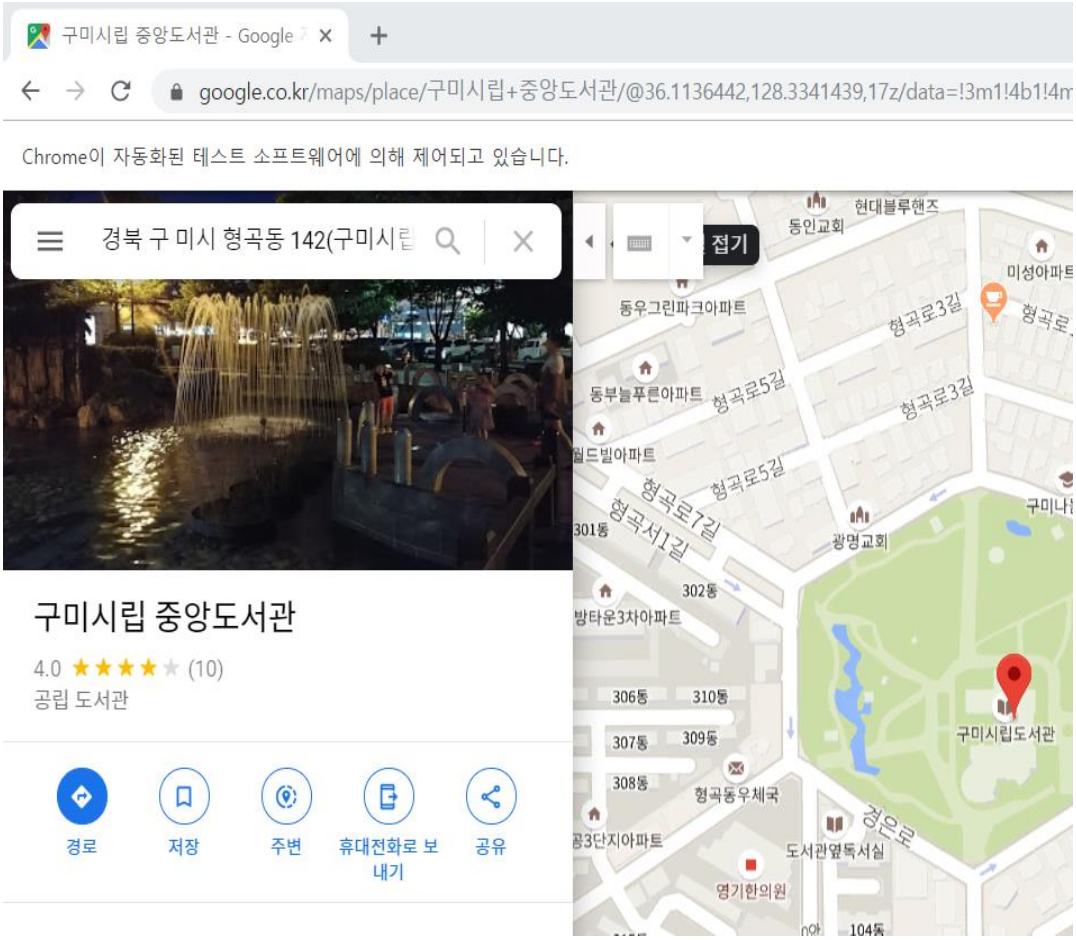
```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys  #내용지우기 위해 필요한 모듈
from time import sleep


latitude=[]
longitude=[]

driver = webdriver.Chrome('../driver/chromedriver')
driver.implicitly_wait(3)
driver.get("https://www.google.co.kr/maps/")

for i in range(len(addr_list)):
    select=driver.find_element_by_name('q')
    select.send_keys(addr_list[i])
    driver.find_element_by_xpath("""//*[@id="searchbox-searchbutton"]""").click()
    sleep(2)
    a=driver.current_url
    local=a.split('@')[1]
    latitude.append(local.split(',')[0]) #위도
    longitude.append(local.split(',')[1]) #경도
    sleep(3)
    for j in range(1,len(addr_list[i])+1):
        select.send_keys(Keys.BACKSPACE)
        #sleep(0.5)
    sleep(1)
driver.close()
```

코드설명: 구글 맵의 상단의 url 주소를 보면,



@ 이후 부분에 위도와 경도 정보가 나와있는 것을 알 수 있다. 따라서 이를 split으로 나누어 좌측값은 위도로, 우측값은 경도로 저장하였다.

이를 반복문을 이용해 자동으로 주소들을 입력할 수 있도록 하였고, 너무 빨리 입력하면 검색 버튼을 누르기도 전에 주소 검색창이 지워졌기 때문에 sleep()함수를 이용해 코드가 3초 뒤에 자동으로 지워지도록 코드를 작성하였다.



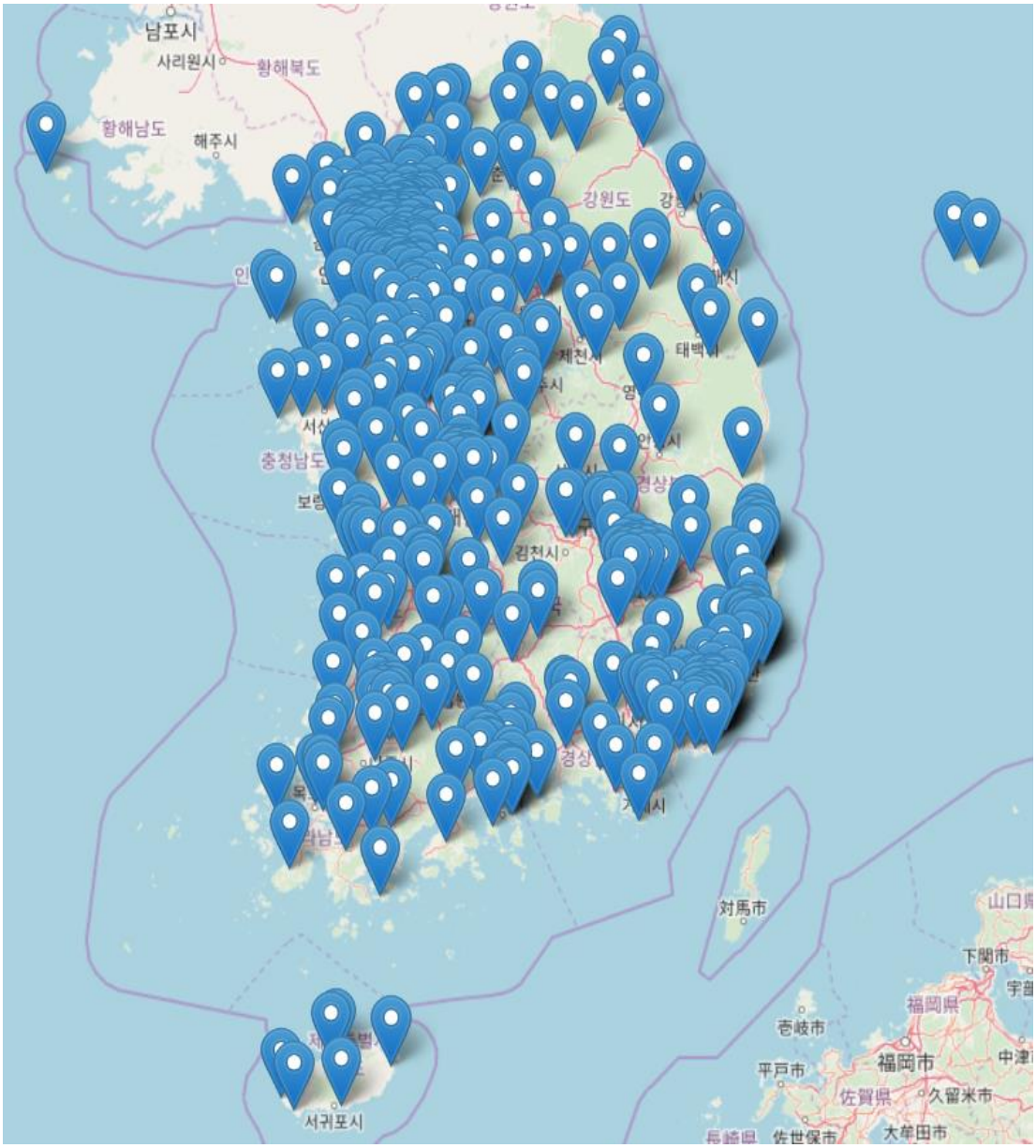
3. 위도 경도를 불러온뒤, 주소 위도 경도 정보를 하나의 dataframe으로 저장해 csv파일에 저장하였다.

4.dataframe에 저장하기

```
addr_all={'주소':addr_list,'위도':latitude,'경도':longitude}
df=pd.DataFrame(addr_all)
df.head()
```

	주소	위도	경도
0	경기 양주시 백석읍 꿈나무로 199꿈나무도서관	37.792424	126.9845236
1	대구 달성군 현풍면 부리 247 (상수도사업본부 달성사업소)(현풍중앙로 144-34...	35.6999534	128.4412756
2	경기 이천시 설성면 신필리산 88-5(전파연구소 입구)	37.1387735	127.5449728
3	경기 포천시 삼육사로 2186번길 11-15선단보건지소	37.8533363	127.1570327
4	충북 음성군 금왕읍 무극로370 (금왕공설운동장내)	36.9627761	127.5755361

4. 추가로, folium을 이용해 전국 미세먼지 측정소 위치를 marker로 표시하고 html 파일로 저장하였다.



코드 2 설명

코드 2는 json을 이용해 지도로 전국의 미세먼지 수치를 표현하기 위해서와 지역의 이름을 간결하지만 정확하게 표현하기 위한 목적으로 코드를 작성하였다.

1. 코드 1에서 저장한 주소,위도,경도,정보를 불러온다.
2. 주소마다 행정구역 위치를 전처리하고, 지역 column을 추가한다.

```
a=[eachAddress.split()[0] for eachAddress in df2['주소']]


#행정구역 전처리 (같은 서울이여도 서울, 서울시, 서울특별시로 기록되는 경우가 있었음)
for i in range(len(a)):
    if '광역시' in a[i]:
        a[i]=a[i].replace('광역시','')
    elif '특별시' in a[i]:
        a[i]=a[i].replace('특별시','')
    elif '강서구' in a[i]:
        a[i]=a[i].replace('강서구','')
    elif '안산시' in a[i]:
        a[i]=a[i].replace('안산시','')
    elif '경기도' in a[i]:
        a[i]=a[i].replace('경기도','경기')
    elif '서울시' in a[i]:
        a[i]=a[i].replace('서울시','서울')

df2['지역']=a  # 17개의 행정구역별로 나눈다.
df2.head()
```

3. 주소마다 시군정보 역시 전처리하고, 시군 column을 추가한다.
4. 주소마다 3번째 split의 길이가 3일 경우에는 두글자만 따오고, 4일 경우에는 세글자만 가져와 상세주소 column을 추가해 dataframe에 저장한다.
5. json에서 지정된 지역 이름과 다른 지역들의 이름을 바꿔준다.

   지역이름이 다른경우

   case1)

```
two_word_list=['서울','부산','인천','광주','대구','울산','대전']
```

   위의 지역들 경우에는 json에 지정된 지역 아이디가 지역이름 column

과 합쳐진 이름으로 저장되어 있다. 또한 지역이름의 이름이 겹치는 경우도 있어 이를 해결하기 위해서도 지역과 시군을 더한 이름을 ID 컬럼에 추가한다.

ex-1> 성동 X-> 서울 성동으로 변경

ex-2> 강서-> 부산 강서. 서울 강서로 겹치지 않게 변경

```
for n in df2.index:
    if df2.loc[n,'지역'] in two_word_list:
        df2.loc[n,'ID']=df2.loc[n,'지역']+' '+df2.loc[n,'시군']
```

case2)

```
more_detail=['성남','수원','안산','전주','창원','포항','청주','용인','고양','부천','
```

위의 지역들은 시군 column과 상세주소 column의 값을 합쳐야 json 아이디와 동일해진다.

```
elif df2.loc[n,'시군'] in more_detail:
    df2.loc[n,'ID']=df2.loc[n,'시군']+' '+df2.loc[n,'상세주소']
```

6. 부천 지역의 경우에만 예외적으로 측정소의 주소가 부천시 ~로(신주소 형식)로 되어 있어 수동으로 변경이 필요하다. (ex>부천시 원미동의 경우 신주소로 부천시 심중로로 저장되어 있다.)

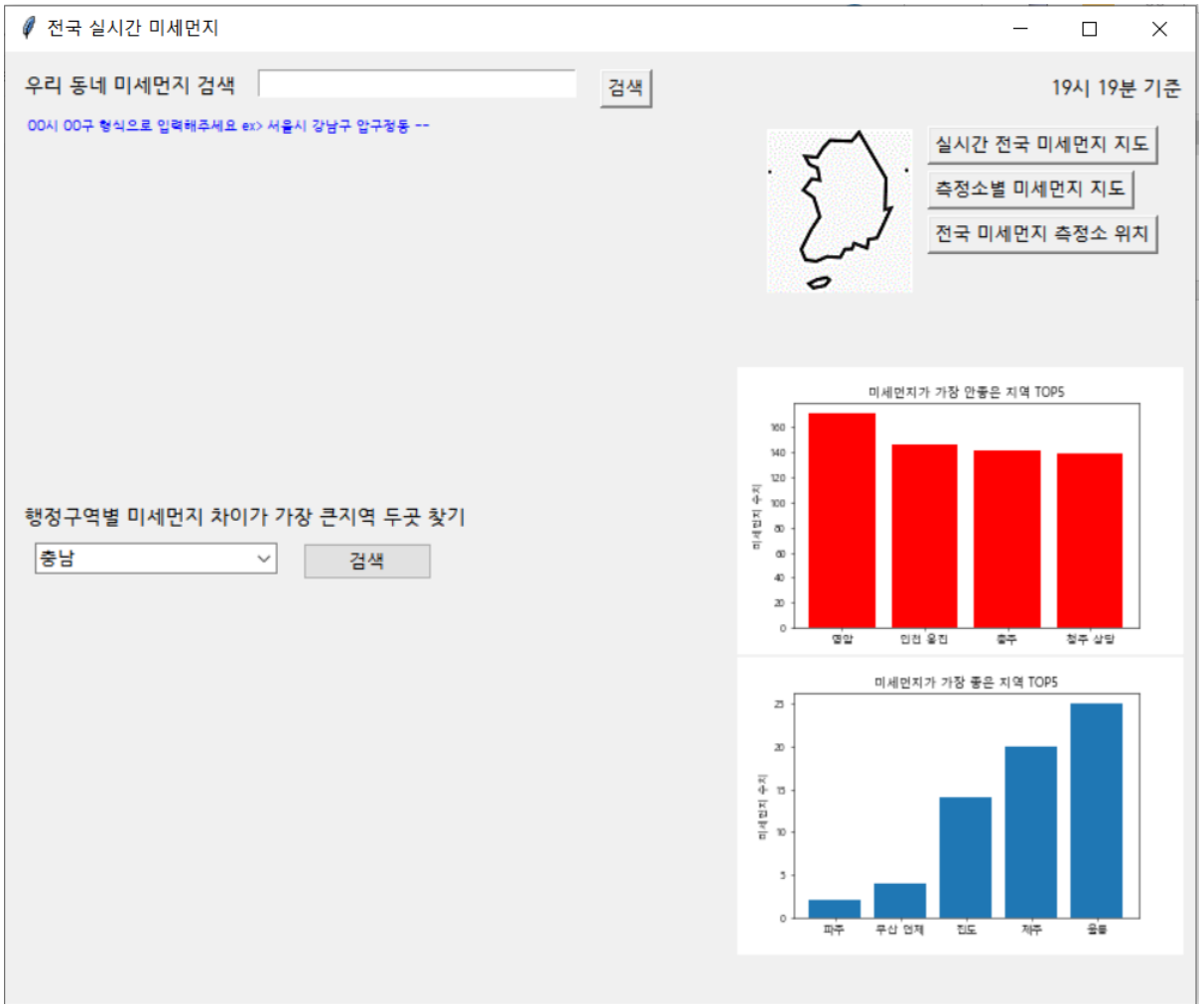
```
#부천지역의 경우에는 예외적으로 측정소가 부천시 ~로로 되어있어 수동으로 변경이 필요하
#부천 경인옛로-> 부천 소사,부천 심중로->부천 원미,부천 성오로->부천 오정으로 변경

for n in df2.index:
    if df2.loc[n,'ID']=='부천 경인옛':
        df2.loc[n,'ID']='부천 소사'
    elif df2.loc[n,'ID']=='부천 심중':
        df2.loc[n,'ID']='부천 원미'
    elif df2.loc[n,'ID']=='부천 성오':
        df2.loc[n,'ID']='부천 오정'
```

7. 최종 dataframe을 plus ID.csv에 저장한다.

	주소	위도	경도	지역	시군	ID
0	경기 양주시 백석읍 꿈나무로 199꿈나무도서관	37.792424	126.984524	경기	양주	양주
1	대구 달성군 현풍면 부리 247 (상수도사업본부 달성사업소)(현풍중앙로 144-34...	35.699953	128.441276	대구	달성	대구달성

코드3 설명(main code)

main 코드에서는 실시간 미세먼지가 저장된 dataframe과 주소 위도 경도가 저장된 dataframe을 결합한 정보들을 가지고 tkinter로 구현하였다.



또한 dustfeel사이트는 미세먼지 데이터를 한시간마다 업데이트하기 때문에 스레드의 주기를 60분으로 설정하였다.

1. 데이터 병합

```
def getcsv(self):
    #주소,위도,경도,지역,시군,ID 저장된 csv파일 불러오기
    self.df2=pd.read_csv('plus ID.csv',sep=',',encoding='cp949')
    #print(self.df2.head())
    #데이터 병합 열:주소,위도,경도,지역,시군,ID,미세먼지 class,미세먼지
    self.df3=pd.merge(self.df1,self.df2,left_on='address',right_on='주소')
    del self.df3['주소']
    self.df3=self.df3[self.df3['dust value 1']!=-1]#정보없는 데이터 전처리

    self.get_allpic()
```

df1의 columns: 주소, 미세먼지 분류클래스(좋음,보통,나쁨),미세먼지 수치

그리고 정보가 없는(고장) 측정소의 경우에는 분류 클래스열이 '정보없음'을 가지고 미세먼지 수치가 없어 해당값을 -1로 대체하였다(미세먼지 수치를 나중에 string형태에서 int형으로 변환하기 위해)

```
for i in range(len(dust_list)):
    if dust_list[i][0]=='정보없음' and dust_list[i][1]=='정보없음':
        dust_list[i].insert(1,-1) # 빈값에 -1대입 농도가 0보다 작은값을 가질
        dust_list[i].insert(3,-1) # 초미세먼지
    elif dust_list[i][0]=='정보없음' and dust_list[i][1]!='정보없음' and d
        dust_list[i].insert(1,-1) # 미세먼지
    elif dust_list[i][0]!='정보없음' and dust_list[i][2]=='정보없음':
        dust_list[i].insert(3,-1)#초미세먼지
```

df1은 1시간씩 업데이트 될때마다 계속해서 정보가 변경된다.

df2 의 columns: 주소,위도,경도,지역,시군, ID

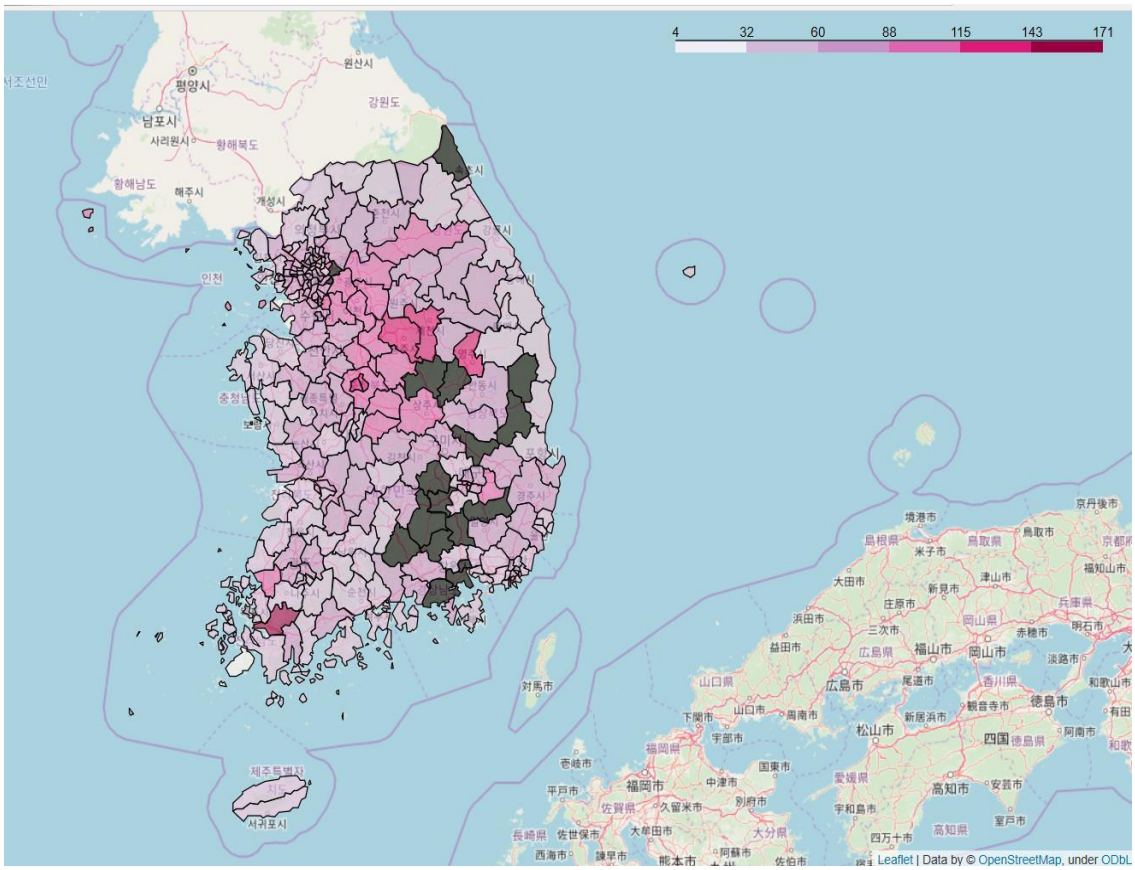
df1과 df2를 merge해서 공통칼럼인 주소를 기준으로 inner join해준다.(값은 동일하나 열이름이 다르기때문에, left_on, right_on을 써준다.)

따라서 df3는 열로 주소 실시간 미세먼지 분류 클래스, 미세먼지 수치, 위도,경도, 지역,시군,ID를 가진다. 그리고 미세먼지가 값이 -1인 경우에는 제외시켰다.

2. 다양한 지도 구현

우측 상단에 버튼 3개를 클릭하면 해당 지도를 html로 열어 보여주는 기능을 구현하였다.

(1) 실시간 전국 미세먼지 지도



코드

```
def get_allpic(self): #실시간 미세먼지 전국 지도 그리기
    self.loc_data=pd.pivot_table(self.df3,index=['ID'],values=['dust value 1'],a

    geo_path='05. skorea_municipalities_geo_simple.json'
    geo_str=json.load(open(geo_path,encoding='utf-8'))
    map=folium.Map(location=[36.2002,127.054],zoom_start=7)
    map.choropleth(geo_data=geo_str,
          data=self.loc_data['dust value 1'],
           columns=[self.loc_data.index,self.loc_data['dust value 1']],
           fill_color='PuRd',
           key_on='feature.id'
          )
    map.save('dust_colormap.html')
```

코드설명

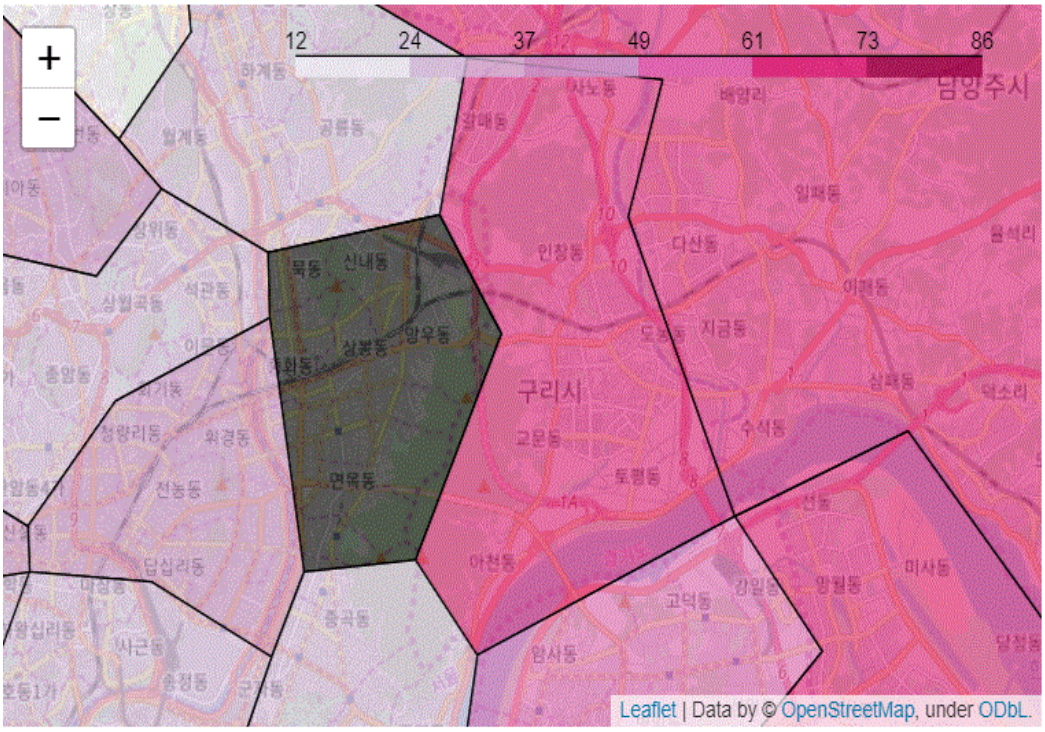
df3의 id column을 이용해 공통 id끼리 pivot table을 통해 평균값으로 미세먼지 값을 가지는 loc_data dataframe 생성

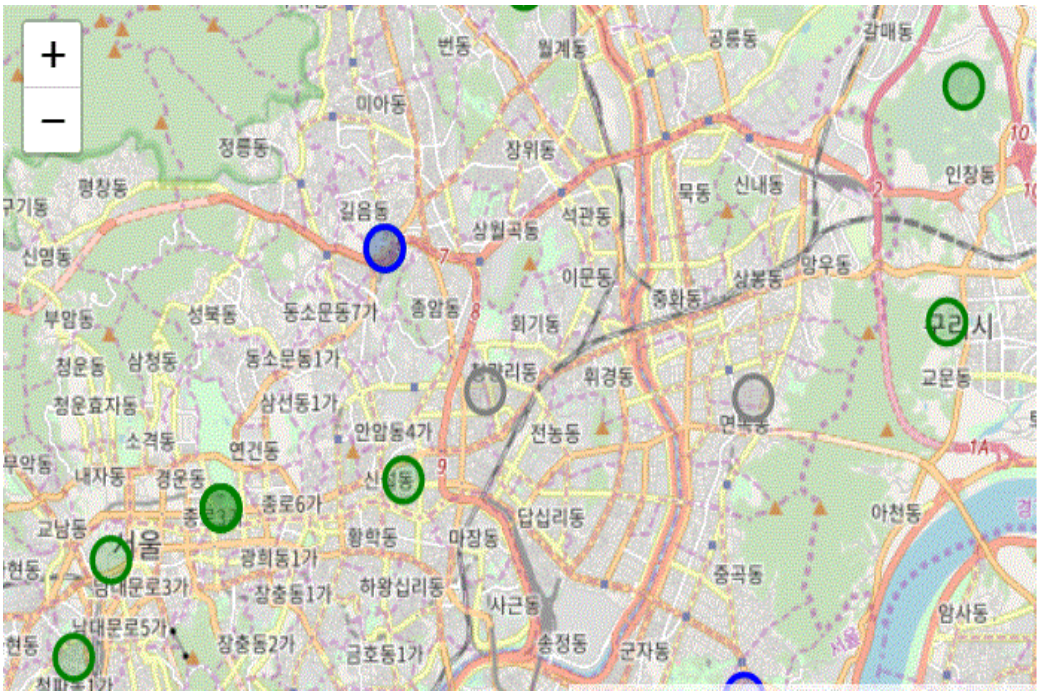
ID	dust value 1
가평	51.000000
강릉	31.000000
강진	24.000000
거제	30.500000
거창	36.500000
경산	53.000000
경주	57.000000
계룡	39.000000
고성	40.000000
고양 덕양	30.000000
고양 일산동	38.500000
고양 일산서	37.000000
고창	28.000000
고흥	20.000000
곡성	39.000000
공주	52.500000

korea.json 파일에 내재된 feature.id와 일치하는 값들을 지도로 표시하도록 함

<-loc_data

그런데 특정 일부지역만 계속해서 업데이트해도 결측값이 발생하는 지역이 있다.





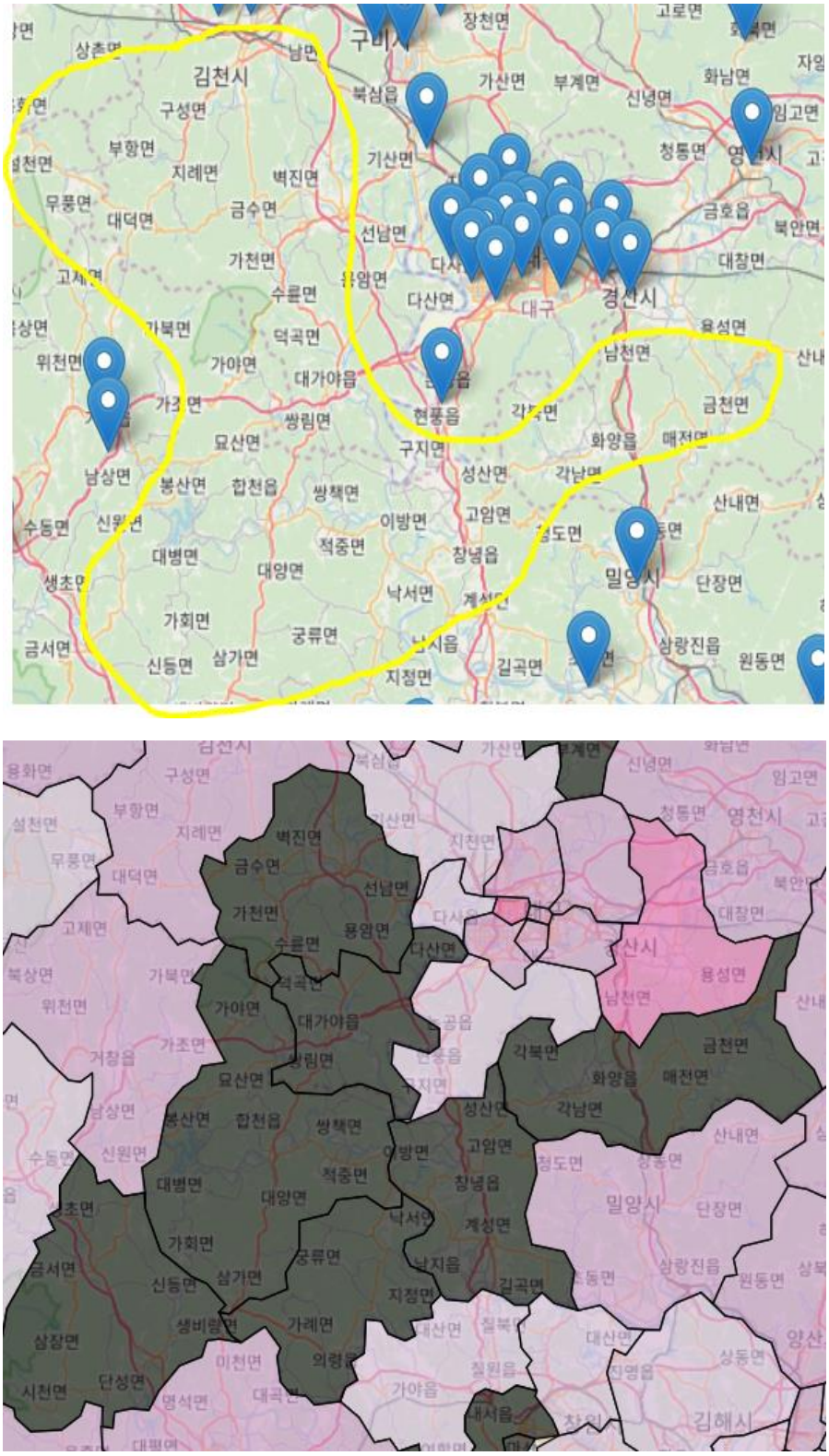
그 이유로는 첫번째로는 미세먼지 관측소가 계속고장난 지역에는

```
self.df3=self.df3[self.df3['dust value 1']!=-1]#정보없는 데이터 전처리
```

다음과 같이 전처리를 해서 값을 없앴기 때문에, 측정값이 nan값이 발생하게 된다.(회색으로 마크된 곳이 고장난 지역)

두번째 이유는 특정지역에는 미세먼지 측정소가 없어 측정값을 가지지

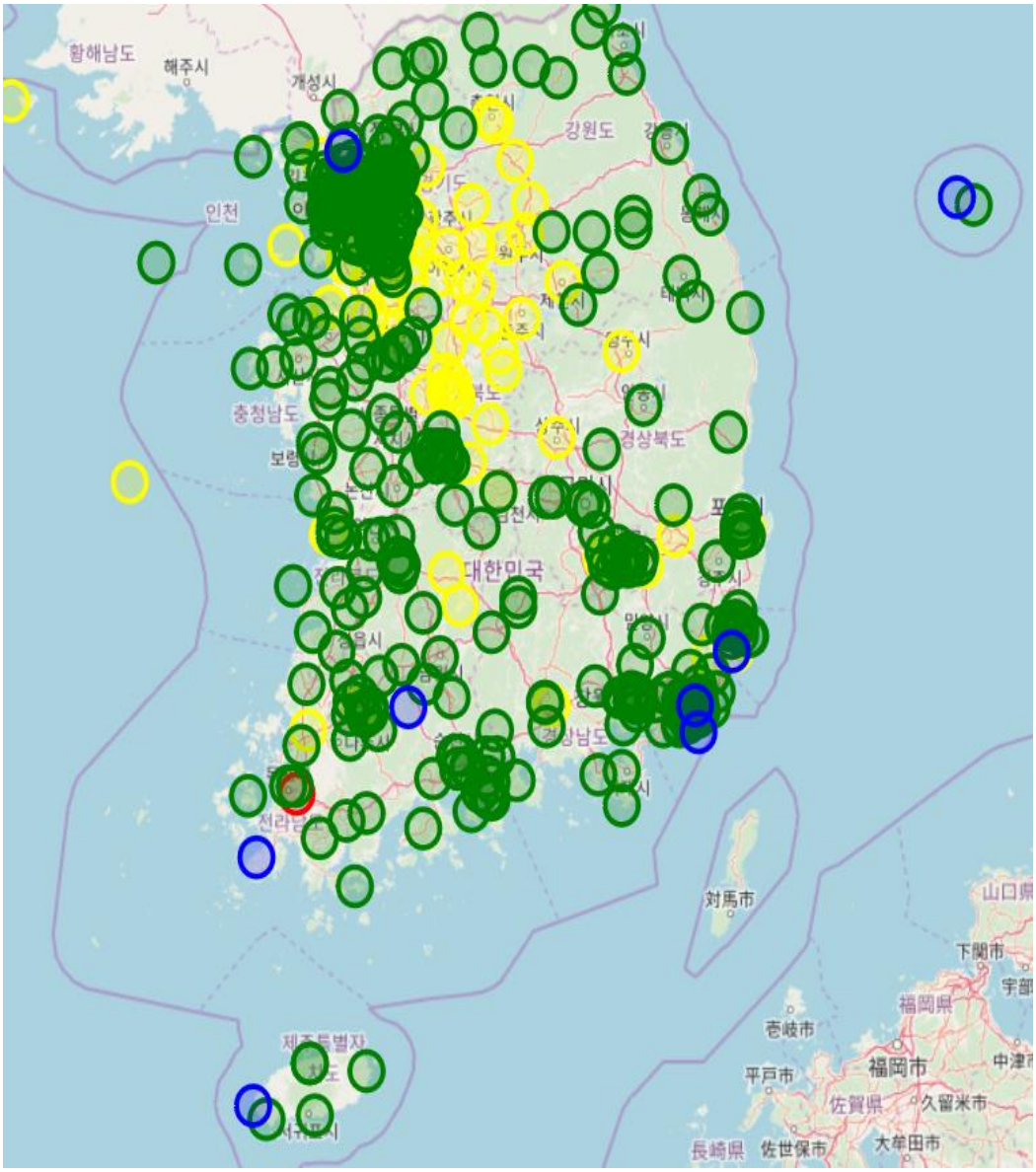
않은 경우가 있었다.



(전국미세먼지 측정소가 저장된 지도와 결측값이 나타난 지역의 비교)

(2) 측정소별 미세먼지 지도

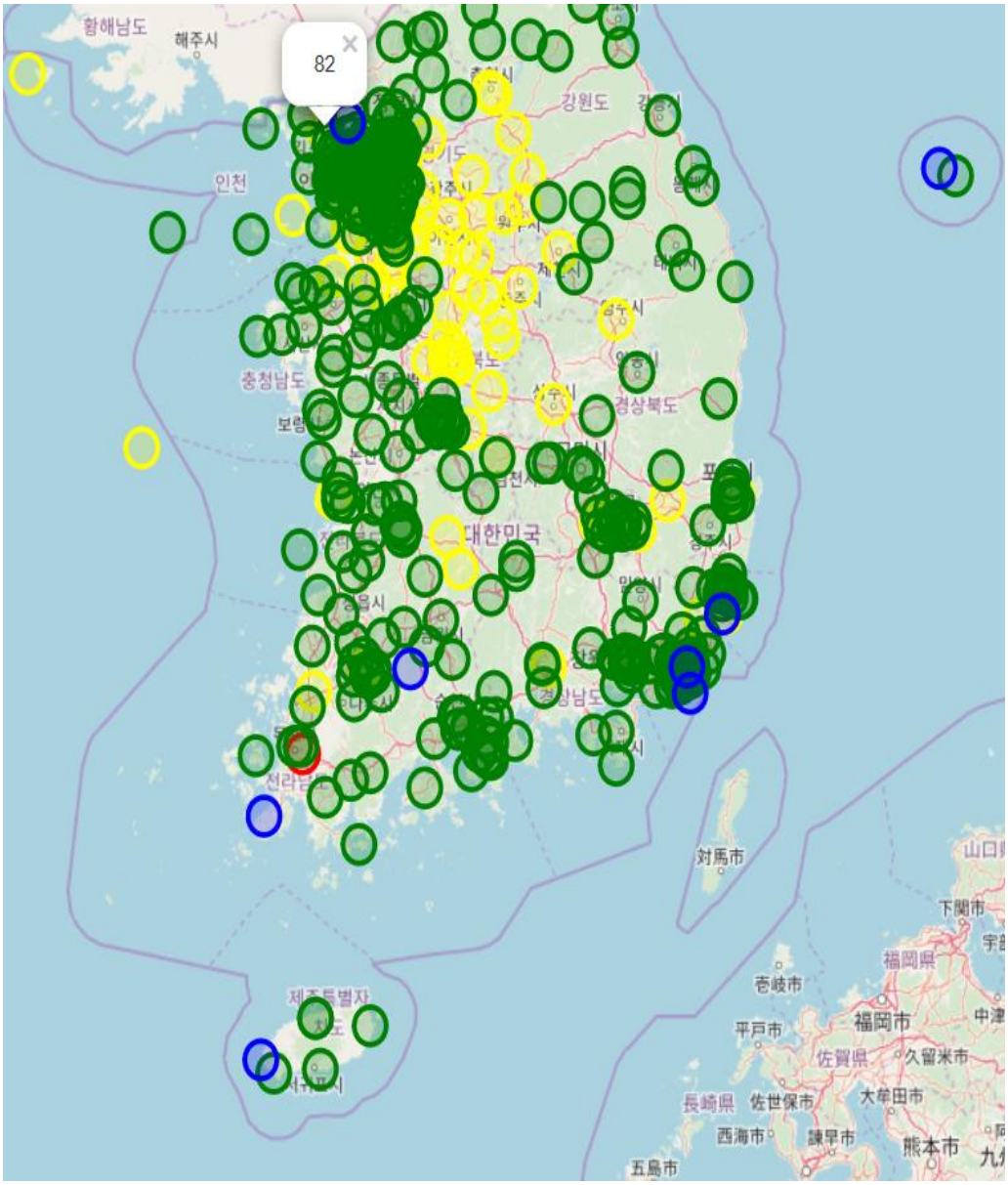
측정소의 위치를 조금 더 자세하게 표현하기 위해 circlemark를 통해 지도로 나타내보았다.



코드

```
def get_markpic(self):
    map1=folium.Map(location=[36.2002,127.054],zoom_start=7)

    for n in self.df3.index:
        dust_value=self.df3.loc[n,'dust value 1']

        if self.df3.loc[n,'fine dust']=='좋음':
            icon_color='blue'
        elif self.df3.loc[n,'fine dust']=='보통':
            icon_color='green'
        elif self.df3.loc[n,'fine dust']=='나쁨':
            icon_color='yellow'
        elif self.df3.loc[n,'fine dust']=='매우나쁨':
            icon_color='red'


        folium.CircleMarker(
            location=[self.df3.loc[n,'위도'],self.df3.loc[n,'경도']],
            radius=10,
            popup=dust_value,
            color=icon_color,
            fill=True,
            fill_color=icon_color

        ).add_to(map1)

    map1.save('dust_mark.html')
```

df3 column의 fine dust에서 해당 분류 클래스마다 색깔을 다르게하여 지도에 표시하도록 하였다.



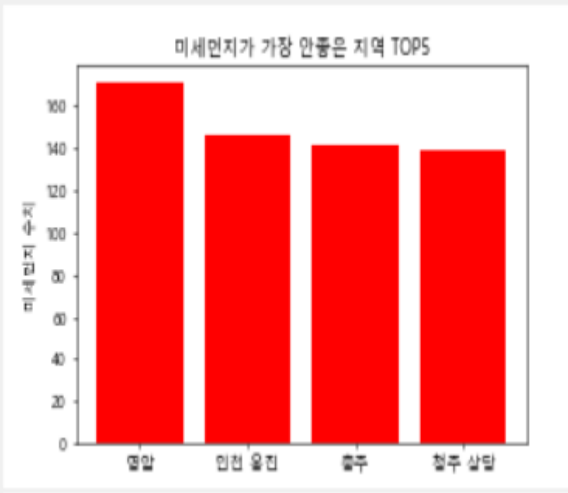
그리고 특정원을 클릭하면, 해당 측정소의 미세먼지값이 표시되도록하였다.

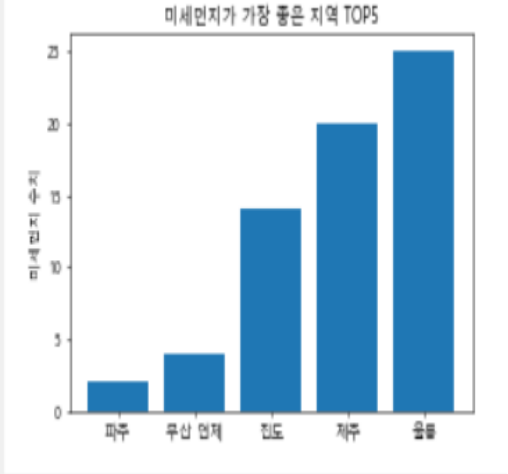
(3) 전국 미세먼지 측정소

미세먼지 측정소의 위도 경도를 따와 marker로 표시해주었다.(csv1 코드에서 실행)

```
import folium

map_mw=folium.Map(location=[36.2002,127.054],zoom_start=7) #html의 시작점


for i in range(len(latitude)):
    folium.Marker([latitude[i],longitude[i]],pop='**').add_to(map_mw)
```

3. 미세먼지가 가장 좋은지역/ 안 좋은지역 상위 5개 지역 뽑아내기



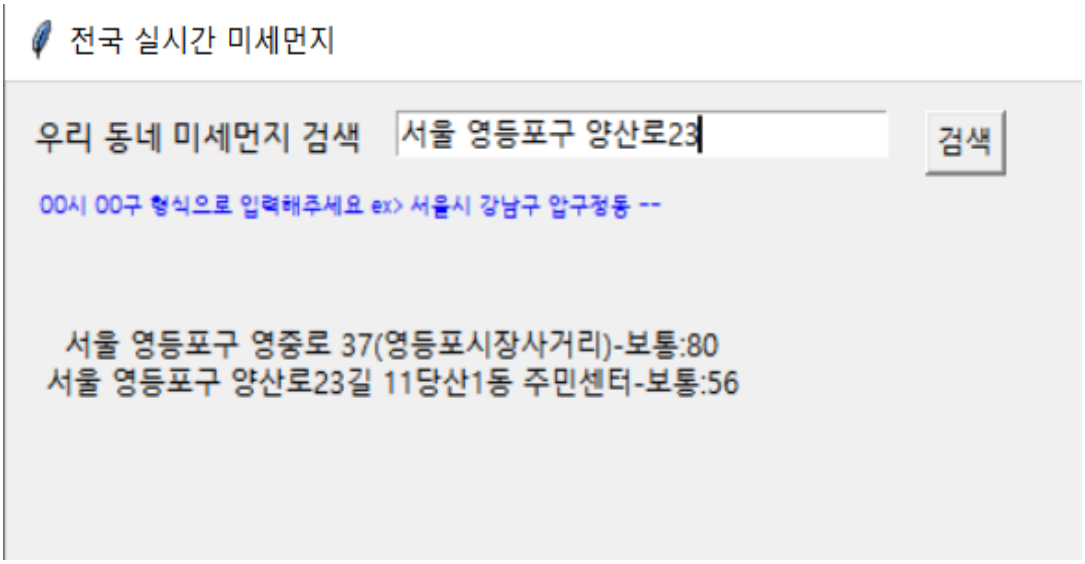


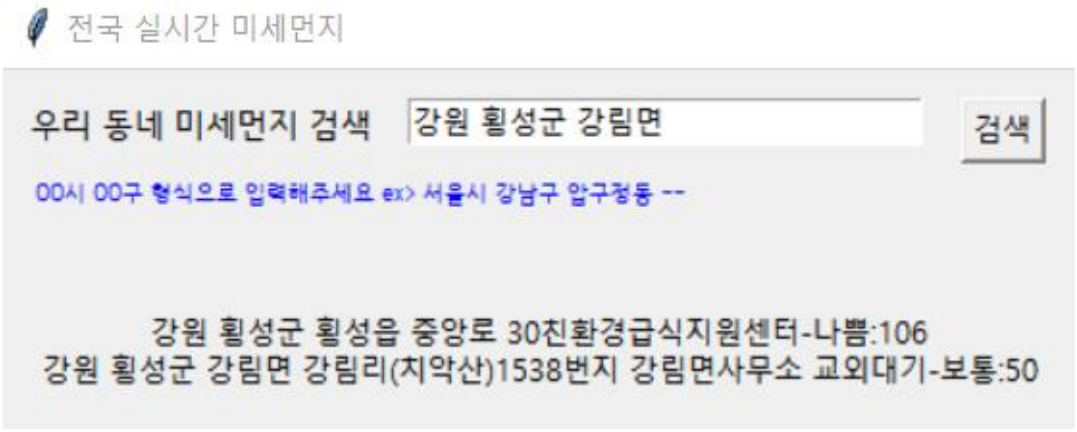
코드

```
def top_bad(self): #가장 안좋은 지역
    df4=self.df3.head()
    plt.rc('font', family='Malgun Gothic')
    plt.rc('xtick', labelsize=11)
    plt.title('미세먼지가 가장 안좋은 지역 TOP5')
    plt.ylabel('미세먼지 수치')
    plt.bar(df4['ID'],df4['dust value 1'],color='red')

    fig1=plt.gcf()
    fig1.savefig('bad_area.png',dpi=50)

    plt.clf()

    self.top_good()
```

df3이 미세먼지 값 기준으로 내림차순으로 정렬되어 있기 때문에 상위의 경우에는 head()로 하위의 경우에는 tail()로 데이터 프레임을 저장해 막대 그래프로 표시하였다.

4. 사용자가 입력하면 해당 동네의 미세먼지 주소와 수치를 알려주는 기능



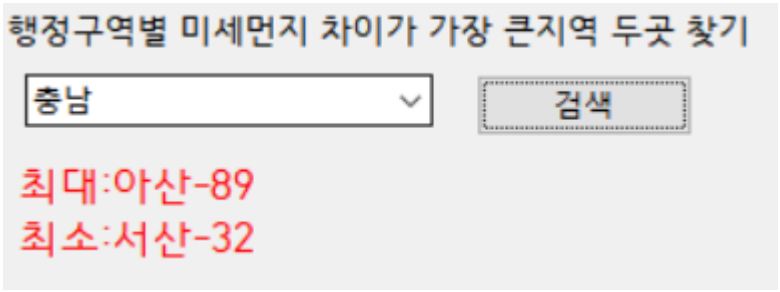


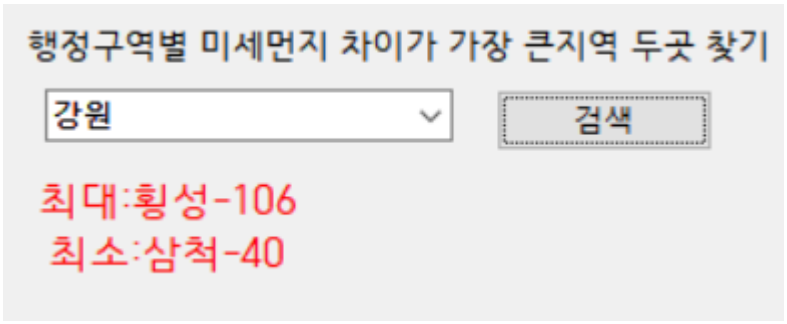

```
def click1(self):# 우리동네 입력하면 찾아주는 기능
    string=self.entry1.get()
    b=string.split(' ')[1]
    if len(b)>=3:
        if b[2]=='시' or  b[2]=='군' or  b[2]=='구':
            b=b[0]+b[1]
    if len(b)>=4:
        if b[3]=='시' or  b[3]=='군' or  b[3]=='구':
            b=b[0]+b[1]+b[2]


    dust_info=[[],[],[]]
    for n in self.df3.index:
        if b in self.df3.loc[n,'시군']:
            dust_info[0].append(self.df3.loc[n,'address'])
            dust_info[1].append(self.df3.loc[n,'fine dust'])
            dust_info[2].append(self.df3.loc[n,'dust value 1'])

    string=''
    for i in range(len(dust_info[0])):
        string=string+'₩n'+dust_info[0][i]+'-'+dust_info[1][i]+':'+str(dust_inf

    self.var2.set(string)
```

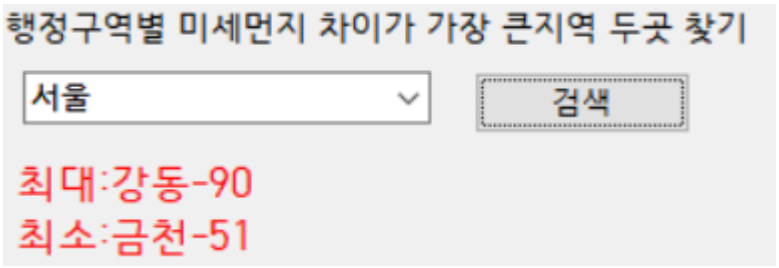
사용자가 입력한 주소를 split한뒤 시군과 일치하는 미세먼지 측정소를 label에 표시하도록 함

5. 행정 구역별 미세먼지 차이가 가장 큰 지역 두곳 찾기

같은 서울이여도 노원구나 강남구의 미세먼지 수치가 다른 것처럼 행정구역별로 미세먼지 값이 가장 큰 곳과 작은 곳을 표시하도록 하였다.







-코드

```
def action(self): #결과 알려주기

    ax=[[],[]]
    for n in self.df3.index:
        if self.df3.loc[n,'지역']==self.str.get():
            ax[0].append(self.df3.loc[n,'dust value 1'])
            ax[1].append(self.df3.loc[n,'시군'])
    result=''
    max_value=0
    min_value=200
    for i in range(len(ax[0])):
        if max_value<=ax[0][i]:
            max_value=ax[0][i]
            max_local=ax[1][i]
        if min_value>=ax[0][i]:
            min_value=ax[0][i]
            min_local=ax[1][i]
    result='최대:'+max_local+'-'+str(max_value)+'₩n'+'최소:'+min_local+'-'+str(min_value)
    self.var.set(result)
```

combobox에 행정구역의 위치를 저장하여 사용자가 선택한 행정구역과 df3의 '지역' 칼럼과 일치하는 지역의 최대 최소를 구하면 된다.