SeoFulL

목차

1. 목표

목표

서울시 생활인구를 분석해 미래 생활인구 예측 및 덜 붐비는 지역 추천

생활인구란?

생활인구란 통신데이터로 특정 시점에 개인이 위치한 지역을 집계한 '현주인구'를 말합니다.
시간대에 따라 변화하는 인구의 규모로 지역간 특성을 추측해 볼 수 있는 유용한 데이터입니다.

2. 개발 과정

Data analysis

서울 열린데이터 광장
SEOUL OPEN DATA PLAZA

자치구 단위 서울 생활인구

기상청

서울시 1시간 간격 기온, 강수, 바람, 습도 데이터

python

pandas

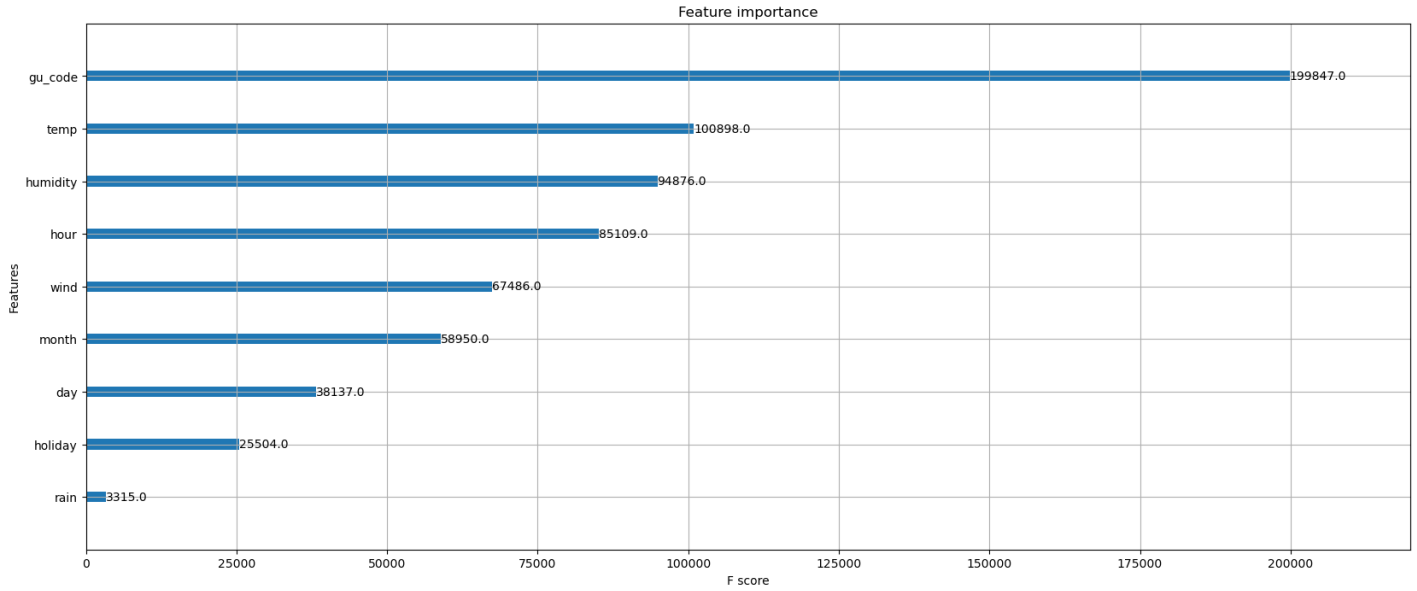
dmlc
XGBoost

Machine Learning

pandas

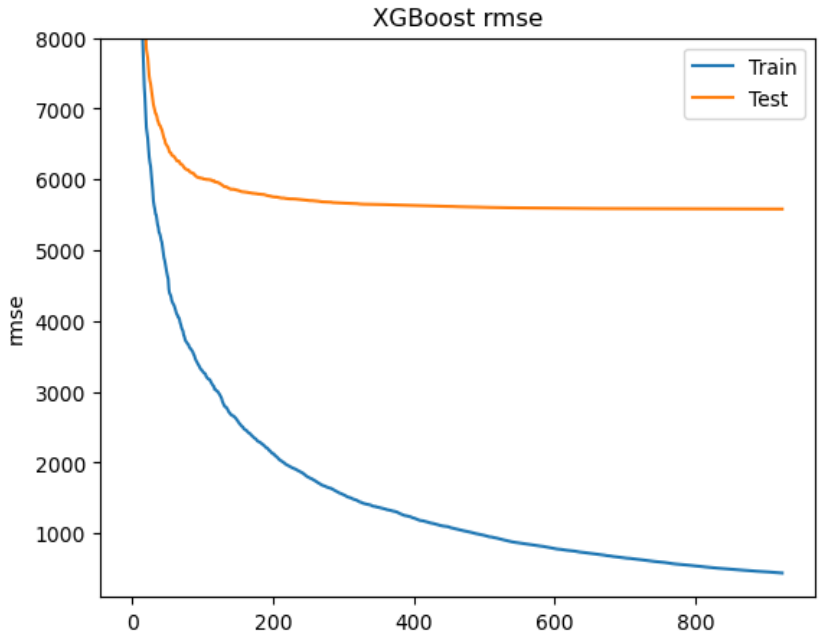
dmlc XGBoost

gu_code	month	holiday	day	hour	temp	wind	rain	humidity	people
11110	1	2	5	0	-9.9	1.0	0	33.2	198013
11140	1	2	5	0	-9.9	1.0	0	33.2	163439
11170	1	2	5	0	-9.9	1.0	0	33.2	252165
11200	1	2	5	0	-9.9	1.0	0	33.2	312343
11215	1	2	5	0	-9.9	1.0	0	33.2	348524
...	...	...	...	...	...	...	...	...	...
11620	12	2	5	23	-1.0	1.0	0	60.9	468533
11650	12	2	5	23	-1.0	1.0	0	60.9	464794
11680	12	2	5	23	-1.0	1.0	0	60.9	626217
11710	12	2	5	23	-1.0	1.0	0	60.9	726058
11740	12	2	5	23	-1.0	1.0	0	60.9	515174



Machine Learning

dmlc
XGBoost



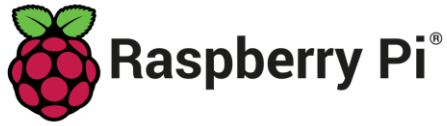

```python
from sklearn.metrics import mean_squared_error
rmse = mean_squared_error(y_test, y_pred, squared=False)
print('Root Mean Squared Error: ', rmse)
```

```
Root Mean Squared Error:  5581.521201213248
```

```python
result['차이(%)'].mean()
```

```
0.006656422695486617 5
```

Server


Raspberry Pi®


debian
GNU/Linux

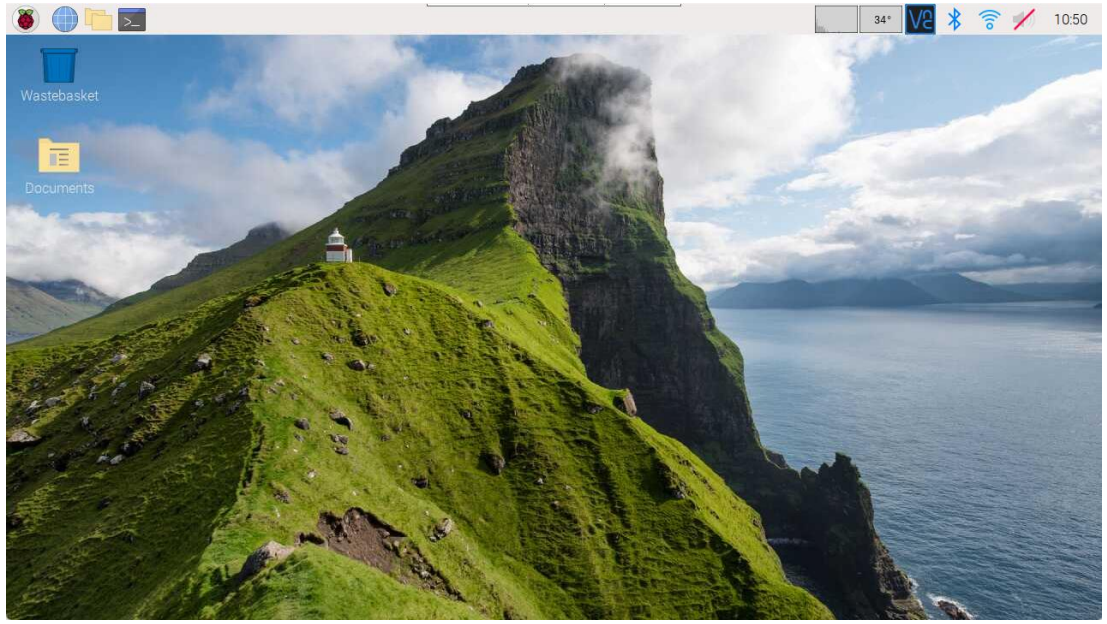

docker


MariaDB


NGINX


DuckDNS

Server



Raspberry Pi® 4B 2gb

debian 11
GNU/Linux

Server

- 컨테이너 생성 (npm, mariaDB)

```
CONTAINER ID   IMAGE                                COMMAND             CREATED       STATUS
12423bca0c6a   jc21/nginx-proxy-manager:latest      "/init"             3 days ago    Up 9 hours
b30886a6cfae   87dc84cdc9be                         "/scripts/run.sh"   3 days ago    Exited (137) 3 days ago
```

```
root        1610    1445  0 10:49 ?        00:00:00 s6-supervise nginx
root        1953    1610  0 10:49 ?        00:00:00 nginx: master process nginx
root        2401    1953  0 10:49 ?        00:00:01 nginx: worker process
root        2402    1953  0 10:49 ?        00:00:00 nginx: worker process
root        2403    1953  0 10:49 ?        00:00:00 nginx: worker process
root        2404    1953  0 10:49 ?        00:00:00 nginx: worker process
root        2405    1953  0 10:49 ?        00:00:00 nginx: cache manager process
pilk       16609   15186  0 11:24 pts/1    00:00:00 grep --color=auto nginx
```

Server

NGINX

DuckDNS

Domain Names *

seofull.duckdns.org

Scheme * | Forward Hostname / IP * | Forward Port *

http | [redacted] | [redacted]

Cache Assets | Block Common Exploits

Websockets Support

Access List

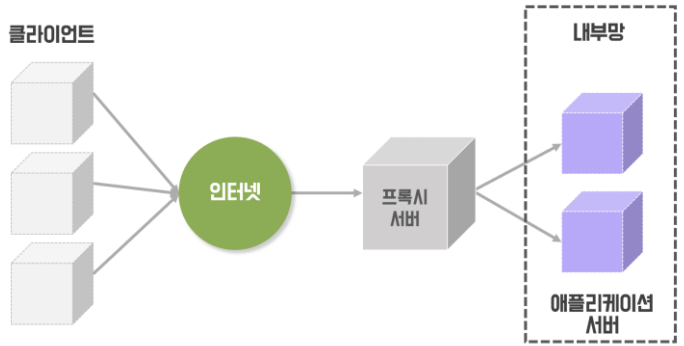
Publicly Accessible

SSL 인증서 발급 ↓

*.seofull.duckdns.org | seofull.duckdns.org | Let's Encrypt - DuckDNS | 6th September 2023, 11:13 am

Created: 8th June 2023

←역방향 Proxy 설정



Back-end

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 32
Server version: 10.5.19-MariaDB-0+deb11u2 Debian 11

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| SeoFulL            |
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
4 rows in set (0.001 sec)

MariaDB [(none)]> use SeoFulL
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [SeoFulL]> show tables;
+-------------------+
| Tables_in_SeoFulL |
+-------------------+
| analyzed_data     |
| weather_data      |
+-------------------+
2 rows in set (0.001 sec)
```

MariaDB

30분 마다
자동 업데이트

One Call API 3.0

API doc | Subscribe

Make an API call to receive access to the various data:

- Current weather
- Minute forecast for 1 hour
- Hourly forecast for 48 hours
- Daily forecast for 8 days
- Government weather alerts
- Historical weather data for any timestamp for 40+ years historical archive
- History daily aggregated weather data for 40+ years historical archive

Read more about this API and subscription plan in the FAQ.

OpenWeather

Back-end

```python
from datetime import datetime
import requests
import schedule
import pymysql
import time
import json

################################################################################
################################################################################

# json 파일 불러오기
with open('json/mariadb.json') as file:
    mariadb = json.load(file)
host = mariadb['host']
port = mariadb['port']
user = mariadb['user']
password = mariadb['password']
database = 'SeoFull'
table = 'weather_data'

with open('json/openweather.json') as file:
    openweather = json.load(file)
api = openweather['api']

# MariaDB 연결
connection = pymysql.connect(host=host, port=port, user=user, password=password, database=database)

################################################################################
################################################################################

# OpenWeatherAPI 링크 설정
openweather_url = 'https://api.openweathermap.org/data/3.0/onecall?lat=37.5642135&lon=127.0016985&exclude=current,minutely,daily,alert&appid='+ api

# 서버로부터 날씨 정보 받아오기
def get_weather_data():
    response = requests.get(openweather_url)
    if response.status_code == 200:
        return response.json()
    else:
        return None

################################################################################
################################################################################

# db에 날씨 정보 저장하기1
def save_weather_data(weather_data):
    # 기존 weather_data 테이블의 데이터 모두 삭제
    with connection.cursor() as cursor:
        delete_query = "DELETE FROM weather_data"
        cursor.execute(delete_query)
        for hour in weather_data['hourly']:
            dt = hour['dt']
            temperature = round(hour['temp'] - 273.15, 1)  # 온도를 섭씨로 변환하고 소수점 한 자리까지 반올림
            humidity = hour['humidity']
            wind_speed = round(hour['wind_speed'], 1)  # 풍속을 소수점 한 자리까지 반올림
            rain = hour.get('rain', {}).get('1h', 0)
            query = "INSERT INTO weather_data (dt, temperature, humidity, wind_speed, rain) VALUES (%s, %s, %s, %s, %s)"
            cursor.execute(query, (dt, temperature, humidity, wind_speed, rain))
    connection.commit()

# db에 날씨 정보 저장하기2
def update_weather_data():
    weather_data = get_weather_data()
    if weather_data:
        save_weather_data(weather_data)
        current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        print(f"Weather information in DB is updated! ({current_time})")
    else:
        print("Failed to fetch weather data.")

################################################################################
################################################################################

# 실행 시 바로 db에 날씨 정보 저장
update_weather_data()

# 매 30분마다 db에 새로운 날씨정보 저장
schedule.every().hour.at("00:00").do(update_weather_data)
schedule.every().hour.at("01:00").do(update_weather_data)
schedule.every().hour.at("30:00").do(update_weather_data)
schedule.every().hour.at("31:00").do(update_weather_data)
while True:
    schedule.run_pending()
    time.sleep(1)
```

dt	temperature	humidity	wind_speed	rain
1,686,492,000	20.2	82	0.7	0
1,686,495,600	19.9	86	0.8	0
1,686,499,200	20	83	0.7	0
1,686,502,800	19.9	81	0.9	0
1,686,506,400	19.6	80	1.1	0
1,686,510,000	19.2	80	0.6	0
1,686,513,600	18.7	80	0.7	0
1,686,517,200	18.9	79	0.8	0
1,686,520,800	20	73	0.8	0
1,686,524,400	21.2	66	1	0
1,686,528,000	22.8	59	1.2	0
1,686,531,600	24.5	52	1.5	0
1,686,535,200	25.7	47	2	0
1,686,538,800	26.7	42	2.3	0
1,686,542,400	27.3	39	3.1	0
1,686,546,000	27.5	38	4.3	0
1,686,549,600	27.5	38	4.7	0
1,686,553,200	27.1	39	4.5	0
1,686,556,800	26.4	41	4.2	0
1,686,560,400	25.2	46	4.6	0
1,686,564,000	23.2	52	4.3	0
1,686,567,600	21.9	54	3.2	0
1,686,571,200	21.5	55	1.7	0
1,686,574,800	21.2	56	1.3	0
1,686,578,400	20.8	58	1.9	0
1,686,582,000	20.3	60	1.8	0
1,686,585,600	19.8	62	1.7	0
1,686,589,200	19.4	64	1.8	0
1,686,592,800	19.2	66	1.8	0
1,686,596,400	18.8	69	1.8	0.14
1,686,600,000	17.6	83	2.3	0.95
1,686,603,600	17	90	0.6	1.54
1,686,607,200	17.7	87	1.1	0
1,686,610,800	19.4	78	1.1	0
1,686,614,400	21.4	67	0.6	0
1,686,618,000	23.1	57	1.3	0
1,686,621,600	24.7	47	2	0
1,686,625,200	25.7	39	3.2	0
1,686,628,800	26.3	35	4.2	0
1,686,632,400	26.5	34	4.8	0
1,686,636,000	26.3	34	4.8	0
1,686,639,600	25.8	35	4.9	0
1,686,643,200	25	37	4.6	0
1,686,646,800	23.8	41	4.5	0
1,686,650,400	22.6	46	3.6	0
1,686,654,000	21.6	51	2.5	0
1,686,657,600	21.1	54	1.6	0
1,686,661,200	20.7	56	1.1	0

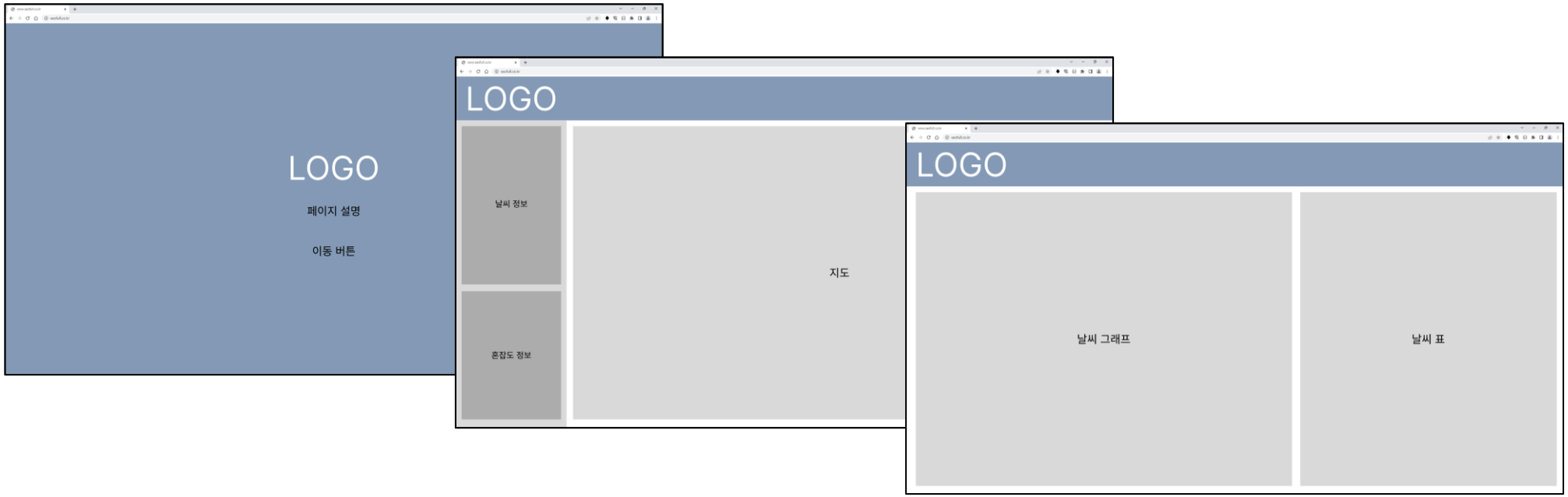
Weather.py 파일 실행 시 업데이트

그 후 정시, 30분 업데이트

```
Weather information in DB is updated! (2023-06-12 11:12:58)
Weather information in DB is updated! (2023-06-12 11:30:02)
Weather information in DB is updated! (2023-06-12 11:31:01)
Weather information in DB is updated! (2023-06-12 12:00:01)
Weather information in DB is updated! (2023-06-12 12:01:05)
```

Front-end

Figma - 스토리 보드 제작



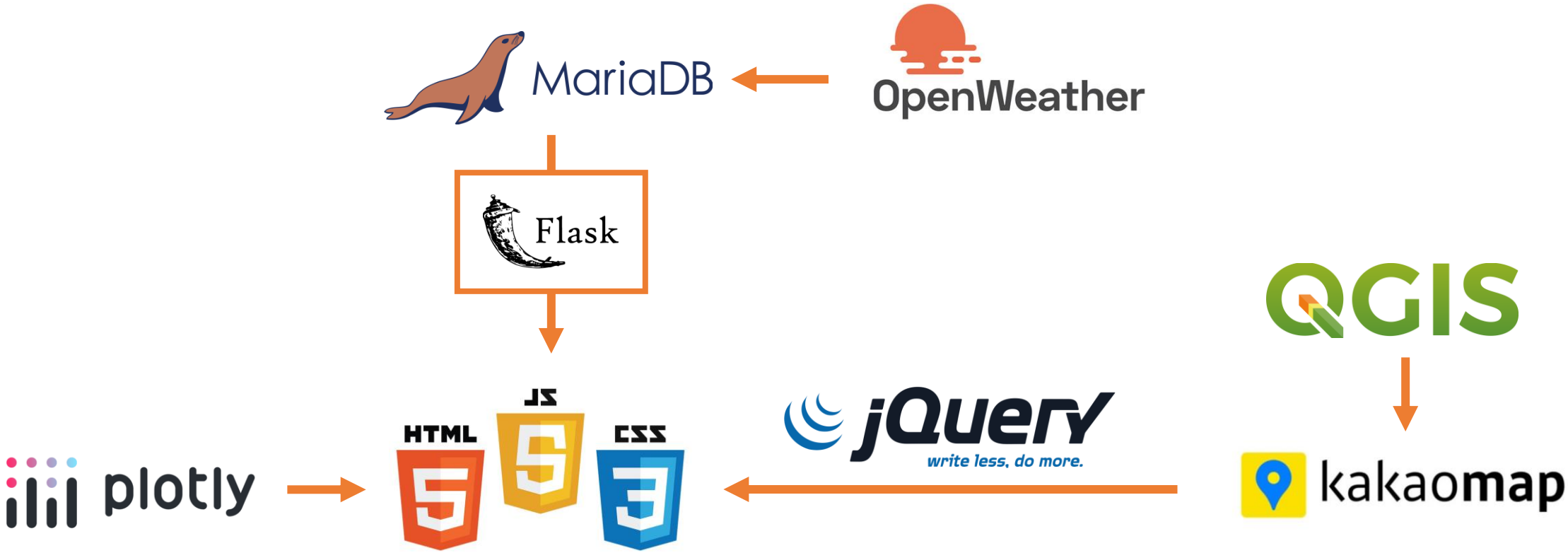
Front-end

Figma - 로고 제작

Seoul + Full = 서울의 밀집을 보여준다.

SeoFulL

Front-end


MariaDB
OpenWeather
Flask
QGIS
plotly
HTML
JS
CSS
jQuery
write less, do more.
kakaomap

Front-end

Flask - DB와 연결

```
52    # MariaDB 연결 정보
53    host = mariadb['host']
54    port = mariadb['port']
55    user = mariadb['user']
56    password = mariadb['password']
57    database = 'SeoFulL'
58    table = 'weather_data'
59
60    # 데이터베이스로부터 날씨 정보를 받는 함수 정의
61    def db_data_all():
62        connection = pymysql.connect(host=host, port=port, user=user, password=password, database=database)
63        query = f"SELECT dt, temperature, humidity, wind_speed, rain FROM {table}"
64        cursor = connection.cursor()
65        cursor.execute(query)
66        row = cursor.fetchall()
67        current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
68        print(f"Weather information from DB is updated! ({current_time})")
69        cursor.close()
70        connection.close()
71        return row
```

Front-end

Flask - html과 연결

```
123     # /main 페이지
124     @app.route("/main")
125     def index():
126         return render_template("main.html")
128     # /congestion 페이지
129     @app.route("/congestion/<int:index>", methods=["GET"])
130     def congestion(index):
                ...
197         return render_template("congestion.html", data1 = data1, new_data = new_data, hansan_gu = hansan_gu, honzap_gu = honzap_gu)
199     # /weather 페이지
200     @app.route("/weather", methods=["GET"])
201     def weather():
                ...
227         return render_template("weather.html", graphJSON=graphJSON, row = row)
229     # 없는 주소 접근 시
230     @app.errorhandler(404)
231     def not_found_error(error):
232         return render_template('error.html')
```

Front-end

Flask - html과 연결

```
123     # /main 페이지
124     @app.route("/main")
125     def index():
126         return render_template("main.html")
128     # /congestion 페이지
129     @app.route("/congestion/<int:index>", methods=["GET"])
130     def congestion(index):
            ...
197         return render_template("congestion.html", data1 = data1, new_data = new_data, hansan_gu = hansan_gu, honzap_gu = honzap_gu)
199     # /weather 페이지
200     @app.route("/weather", methods=["GET"])
201     def weather():
            ...
227         return render_template("weather.html", graphJSON=graphJSON, row = row)
229     # 없는 주소 접근 시
230     @app.errorhandler(404)
231     def not_found_error(error):
232         return render_template('error.html')
```

Front-end

```python
# /congestion 페이지
@app.route("/congestion/<int:index>", methods=["GET"])
def congestion(index):

    # 2일 내 정보 외 접근 시
    if index < 0 or index > 47:
        return render_template("error.html")

    # dataframe 초기화
    new_data = pd.DataFrame({'gu_code': [], 'month': [], 'holiday': [], 'day': [],
                             'hour': [], 'temp': [], 'wind': [], 'rain': [], 'humidity': []})

    # 데이터베이스로부터 날씨 정보를 받아옴
    row = db_data_all()
    row = row[index]

    # 데이터를 딕셔너리로 변환
    data1 = {
        'dt': convert_unix_time1(row[0]),
        'temperature': row[1],
        'humidity': row[2],
        'wind_speed': row[3],
        'rain': row[4],
    }

    # 시간 데이터 생성
    data2 = time_data(row[0])

    # 분석 위한 새로운 data 생성
    new_data['gu_code'] = gu_code_values
    new_data['month'] = data2['month']
    new_data['holiday'] = data2['holiday']
    new_data['day'] = data2['day']
    new_data['hour'] = data2['hour']
    new_data['temp'] = data1['temperature']
    new_data['wind'] = data1['wind_speed']
    new_data['rain'] = data1['rain']
    new_data['humidity'] = data1['humidity']

    # data 타입 변경
    new_data['month'] = new_data['month'].astype('int64')
    new_data['holiday'] = new_data['holiday'].astype('int64')
    new_data['day'] = new_data['day'].astype('int64')
    new_data['hour'] = new_data['hour'].astype('int64')
    new_data['temp'] = new_data['temp'].astype('float64')
    new_data['wind'] = new_data['wind'].astype('int64')
    new_data['rain'] = new_data['rain']
    new_data['humidity'] = new_data['humidity'].astype('float64')

    # xgb_model을 통해 예측
    new_data['people'] = new_xgb_model.predict(new_data).round()
    new_data['people'] = new_data['people'].astype('int64')

    # DataFrame 정리
    new_data = new_data[['gu_code', 'people']]
    new_data['gu'] = gu_values
    new_data = new_data.sort_values('people')
    new_data = new_data.reset_index().drop(['index'], axis = 1).reset_index()
    new_data['index'] = (new_data['index']+1)*4
    new_data.columns = ['cong', 'gu_code', 'people', 'gu']
    new_data = new_data.sort_values('gu_code').reset_index().drop(['index'], axis = 1)
    new_data['cong_info'] = pd.cut(new_data['cong'], bins=[4, 24, 44, 64, 84, 101],
                                   labels=['매우 한산', '한산', '보통', '혼잡', '매우 혼잡'], right=False)
    hansan_gu = list(new_data[new_data['cong_info']=='매우 한산']['gu'])
    hansan_gu = ' '.join(hansan_gu)
    honzap_gu = list(new_data[new_data['cong_info']=='매우 혼잡']['gu'])
    honzap_gu = ' '.join(honzap_gu)

    # DataFrame 출력
    # print(new_data)
    return render_template("congestion.html", data1 = data1, new_data = new_data,
                           hansan_gu = hansan_gu, honzap_gu = honzap_gu)
```

seofull.duckdns.org/congestion/<int:index> 접속

해당 시간대 날씨 정보 DB로부터 수신

XGBoost 모델로 생활 인구 예측

HTML로 전달

3. 결과물

결과물

https://seofull.duckdns.org

SeoFulL

https://seofull.duckdns.org/main

InPrivate

48시간 이내 서울의 날씨 예보와 예측 생활 인구수을 알아보세요!

생활인구란 통신데이터로 특정 시점에 개인이 위치한 지역을 집계한 '현주인구'를 말합니다.

0~47(시간)을 작성해주세요: 제출

전체 날씨 보기

SeoFulL

https://seofull.duckdns.org/main

InPrivate

seofull.duckdns.org의 메시지

0~47 사이의 정수를 입력해주세요!

확인

48시간 이내 서울의 날씨 예보와 예측 생활 인구수을 알아보세요!

생활인구란 통신데이터로 특정 시점에 개인이 위치한 지역을 집계한 '현주인구'를 말합니다.

0~47(시간)을 작성해주세요: 48 제출

전체 날씨 보기

SeoFulL

https://seofull.duckdns.org/congestion/48

InPrivate

죄송합니다. 해당 페이지를 찾을 수 없습니다.

로고를 누르면 홈페이지로 이동합니다.

SeoFulL

https://seofull.duckdns.org/congestion/0

InPrivate

SeoFulL

2023년 06월 11일 (일) 오후 11시 기준

기상 예보

전체 날씨 보기

기온: 20.2°C
습도: 82%
풍속: 0.7m/s
강우량: 0.0mm

다른 시간도 확인해보세요!

0~47(시간)을 작성해주세요 제출

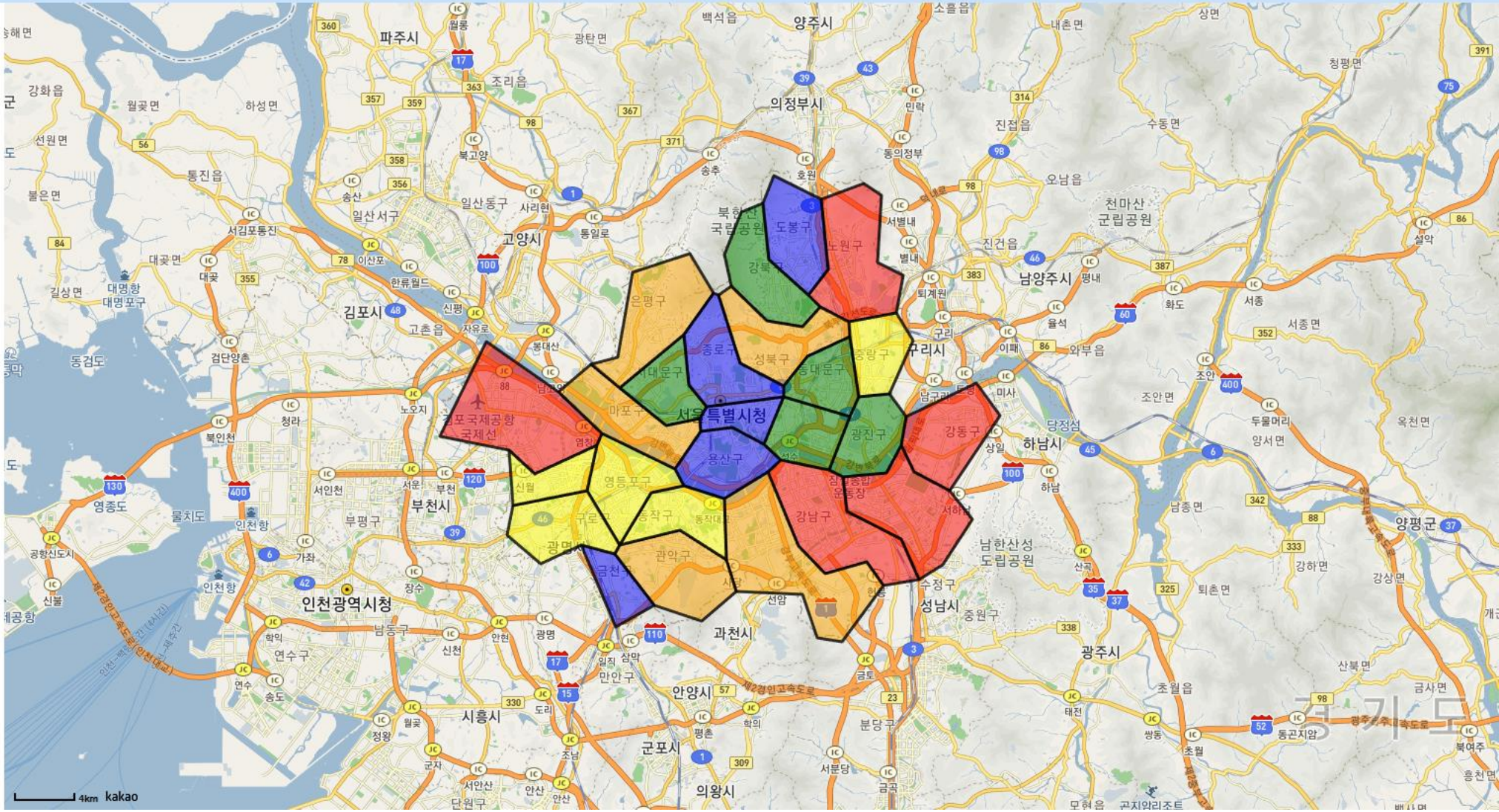
쾌적한 행정구

종로구 중구 용산구 도봉구 금천구

혼잡한 행정구

노원구 강서구 강남구 송파구 강동구

매우 혼잡 | 혼잡 | 보통 | 쾌적 | 매우 쾌적



SeoFulL

https://seofull.duckdns.org/congestion/0

InPrivate

seofull.duckdns.org의 메시지

0~47 사이의 정수를 입력해주세요!

확인

SeoFulL

2023년 06월 11일 (일) 오후 11시 기준

기상 예보

전체 날씨 보기

기온: 20.2°C

습도: 82%

풍속: 0.7m/s

강우량: 0.0mm

다른 시간도 확인해보세요!

48

제출

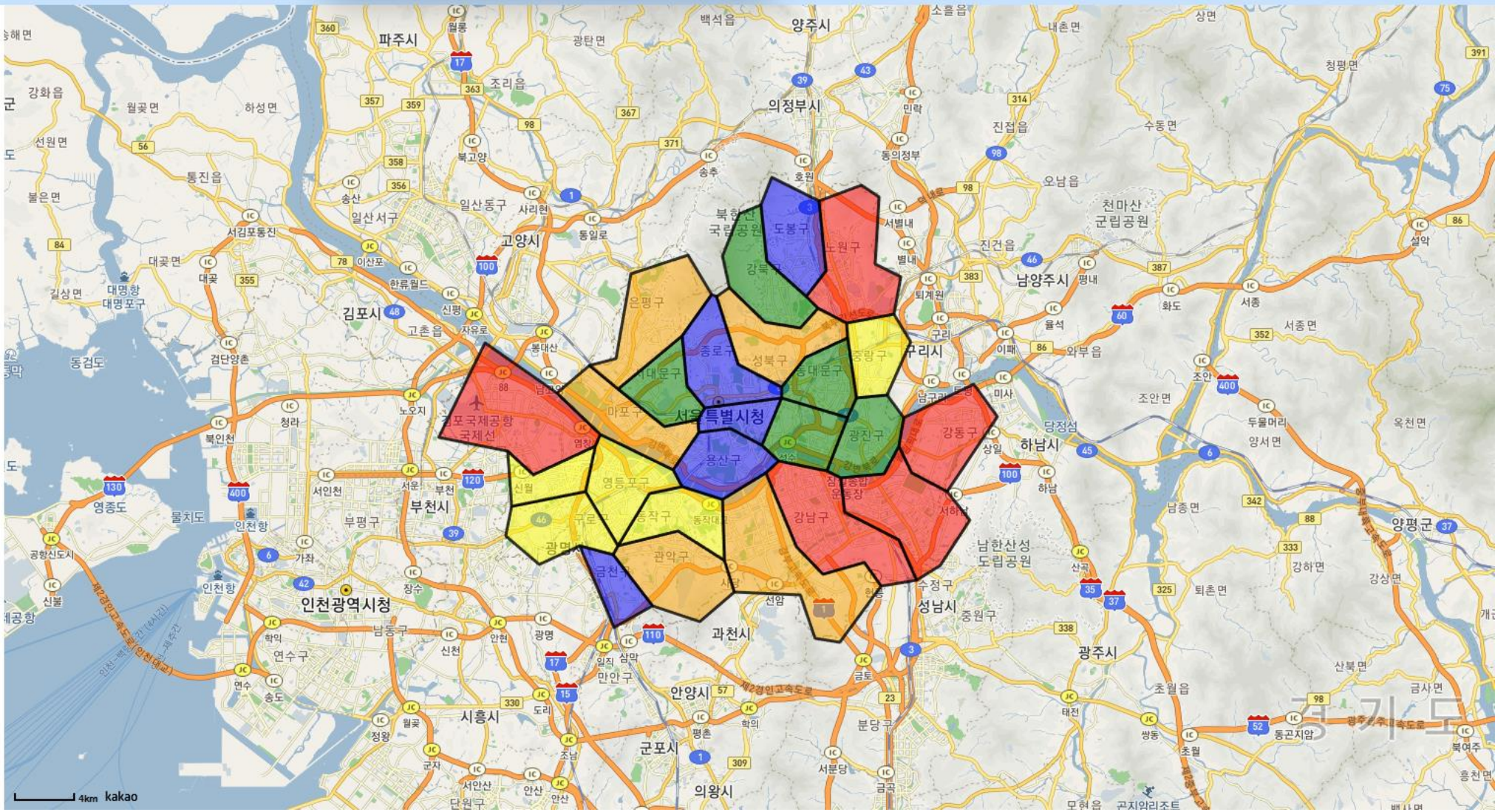
쾌적한 행정구

종로구 중구 용산구 도봉구 금천구

혼잡한 행정구

노원구 강서구 강남구 송파구 강동구

매우 혼잡 | 혼잡 | 보통 | 쾌적 | 매우 쾌적

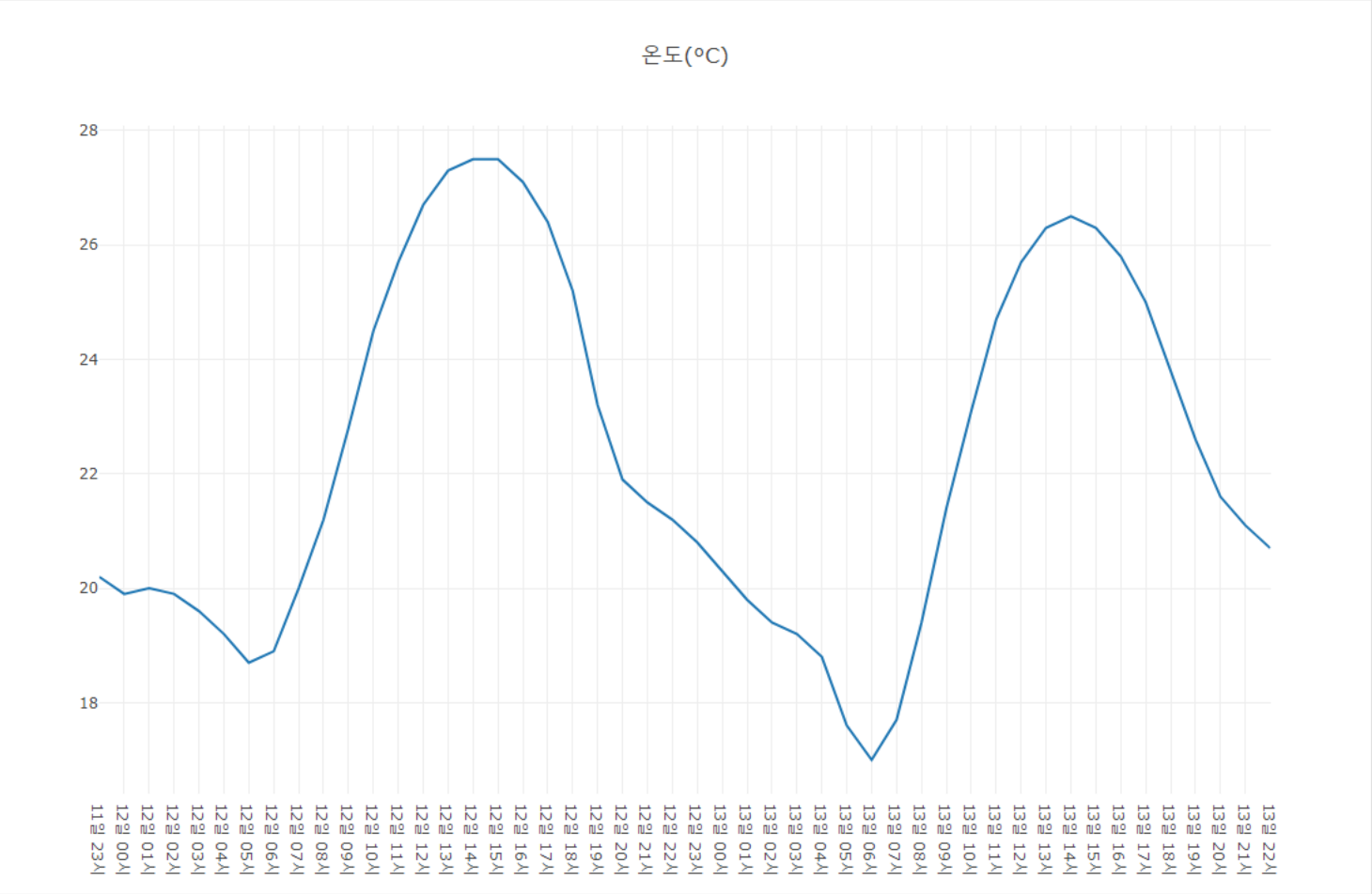


SeoFullL

https://seofull.duckdns.org/weather

InPrivate

SeoFullL



온도 | 습도 | 풍속 | 강우량

날짜	온도(°C)	습도(%)	풍속(m/s)	강우량(mm)
2023년 06월 11일 (일) 오후 11시	20.2	82	0.7	0.0
2023년 06월 12일 (월) 오전 12시	19.9	86	0.8	0.0
2023년 06월 12일 (월) 오전 01시	20.0	83	0.7	0.0
2023년 06월 12일 (월) 오전 02시	19.9	81	0.9	0.0
2023년 06월 12일 (월) 오전 03시	19.6	80	1.1	0.0
2023년 06월 12일 (월) 오전 04시	19.2	80	0.6	0.0
2023년 06월 12일 (월) 오전 05시	18.7	80	0.7	0.0
2023년 06월 12일 (월) 오전 06시	18.9	79	0.8	0.0
2023년 06월 12일 (월) 오전 07시	20.0	73	0.8	0.0
2023년 06월 12일 (월) 오전 08시	21.2	66	1.0	0.0
2023년 06월 12일 (월) 오전 09시	22.8	59	1.2	0.0
2023년 06월 12일 (월) 오전 10시	24.5	52	1.5	0.0
2023년 06월 12일 (월) 오전 11시	25.7	47	2.0	0.0
2023년 06월 12일 (월) 오후 12시	26.7	42	2.3	0.0
2023년 06월 12일 (월) 오후 01시	27.3	39	3.1	0.0
2023년 06월 12일 (월) 오후 02시	27.5	38	4.3	0.0

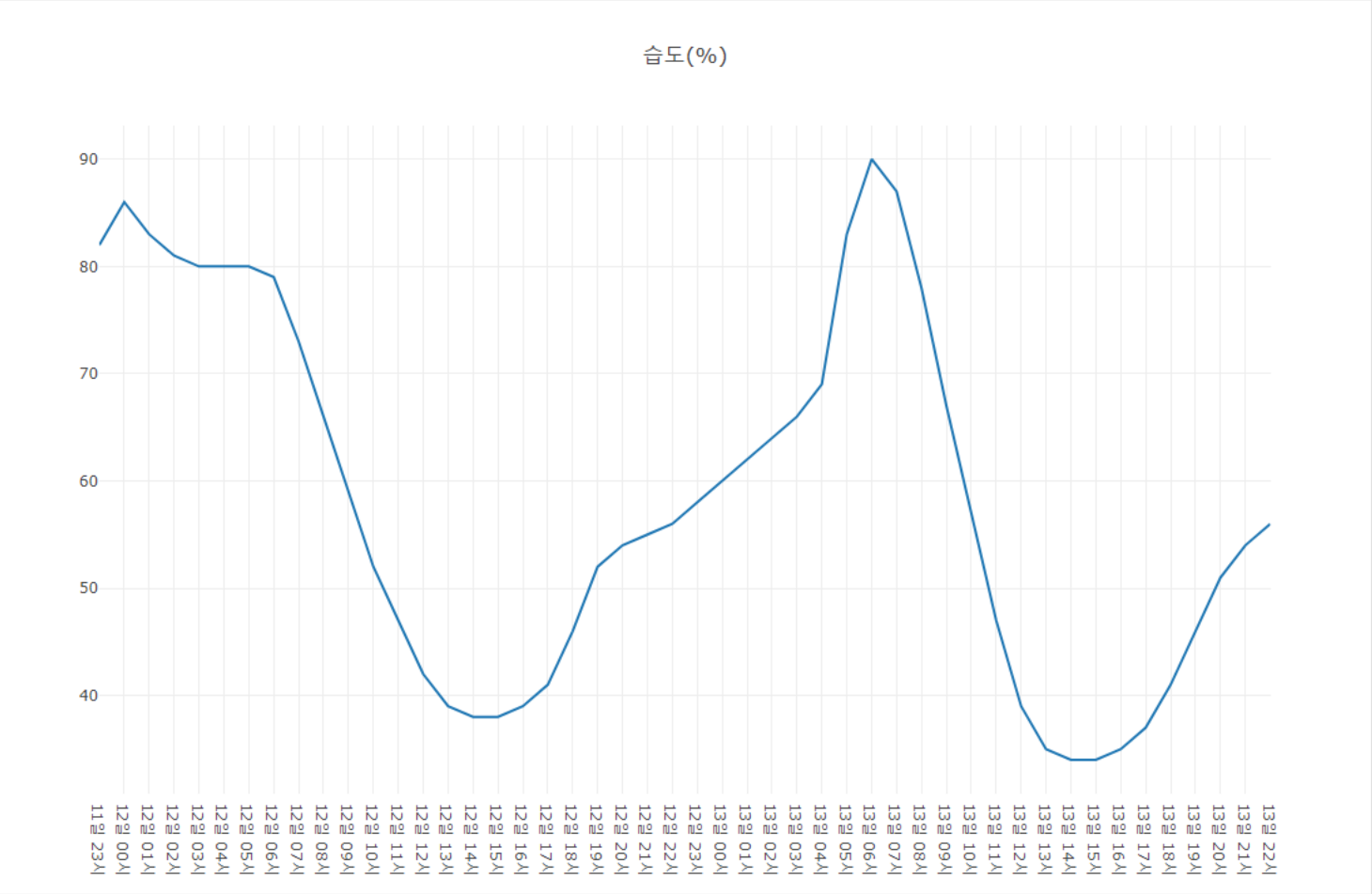
1 | 2 | 3

SeoFull

https://seofull.duckdns.org/weather

InPrivate

SeoFull



온도 습도 풍속 강우량

날짜	온도(°C)	습도(%)	풍속(m/s)	강우량(mm)
2023년 06월 12일 (월) 오후 03시	27.5	38	4.7	0.0
2023년 06월 12일 (월) 오후 04시	27.1	39	4.5	0.0
2023년 06월 12일 (월) 오후 05시	26.4	41	4.2	0.0
2023년 06월 12일 (월) 오후 06시	25.2	46	4.6	0.0
2023년 06월 12일 (월) 오후 07시	23.2	52	4.3	0.0
2023년 06월 12일 (월) 오후 08시	21.9	54	3.2	0.0
2023년 06월 12일 (월) 오후 09시	21.5	55	1.7	0.0
2023년 06월 12일 (월) 오후 10시	21.2	56	1.3	0.0
2023년 06월 12일 (월) 오후 11시	20.8	58	1.9	0.0
2023년 06월 13일 (화) 오전 12시	20.3	60	1.8	0.0
2023년 06월 13일 (화) 오전 01시	19.8	62	1.7	0.0
2023년 06월 13일 (화) 오전 02시	19.4	64	1.8	0.0
2023년 06월 13일 (화) 오전 03시	19.2	66	1.8	0.0
2023년 06월 13일 (화) 오전 04시	18.8	69	1.8	0.14
2023년 06월 13일 (화) 오전 05시	17.6	83	2.3	0.95
2023년 06월 13일 (화) 오전 06시	17.0	90	0.6	1.54

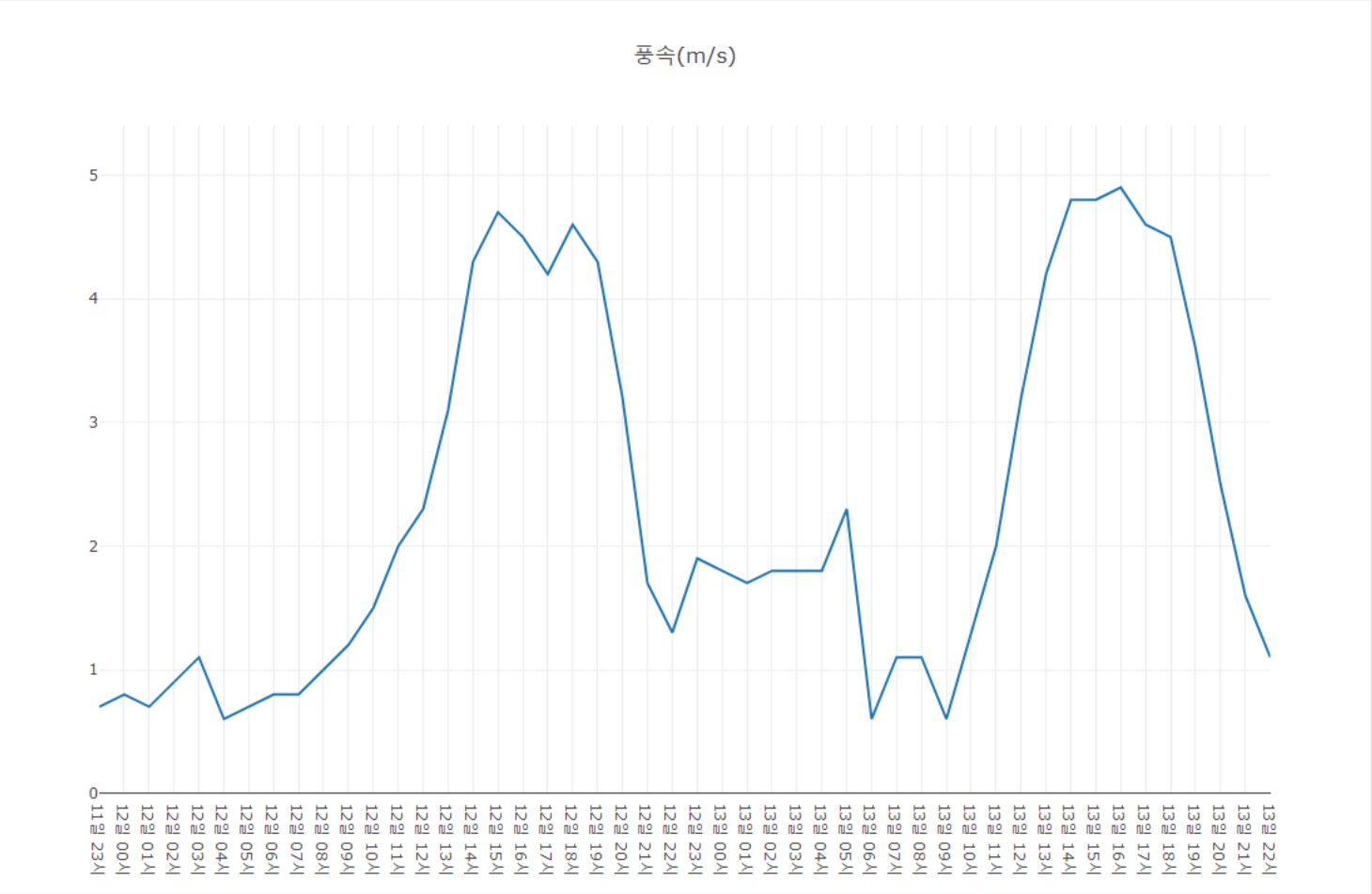
1 2 3

SeoFullL

https://seofull.duckdns.org/weather

InPrivate

SeoFullL



날짜	온도(°C)	습도(%)	풍속(m/s)	강우량(mm)
2023년 06월 13일 (화) 오전 07시	17.7	87	1.1	0.0
2023년 06월 13일 (화) 오전 08시	19.4	78	1.1	0.0
2023년 06월 13일 (화) 오전 09시	21.4	67	0.6	0.0
2023년 06월 13일 (화) 오전 10시	23.1	57	1.3	0.0
2023년 06월 13일 (화) 오전 11시	24.7	47	2.0	0.0
2023년 06월 13일 (화) 오후 12시	25.7	39	3.2	0.0
2023년 06월 13일 (화) 오후 01시	26.3	35	4.2	0.0
2023년 06월 13일 (화) 오후 02시	26.5	34	4.8	0.0
2023년 06월 13일 (화) 오후 03시	26.3	34	4.8	0.0
2023년 06월 13일 (화) 오후 04시	25.8	35	4.9	0.0
2023년 06월 13일 (화) 오후 05시	25.0	37	4.6	0.0
2023년 06월 13일 (화) 오후 06시	23.8	41	4.5	0.0
2023년 06월 13일 (화) 오후 07시	22.6	46	3.6	0.0
2023년 06월 13일 (화) 오후 08시	21.6	51	2.5	0.0
2023년 06월 13일 (화) 오후 09시	21.1	54	1.6	0.0
2023년 06월 13일 (화) 오후 10시	20.7	56	1.1	0.0

온도 습도 풍속 강우량

1 2 3