삼정 **KPMG 4**기 **2**차 프로젝트

머신러닝을 활용한
동대문구 밥퍼(무료급식소) 민원 해결 방안

by BOB-UP

I N D E X

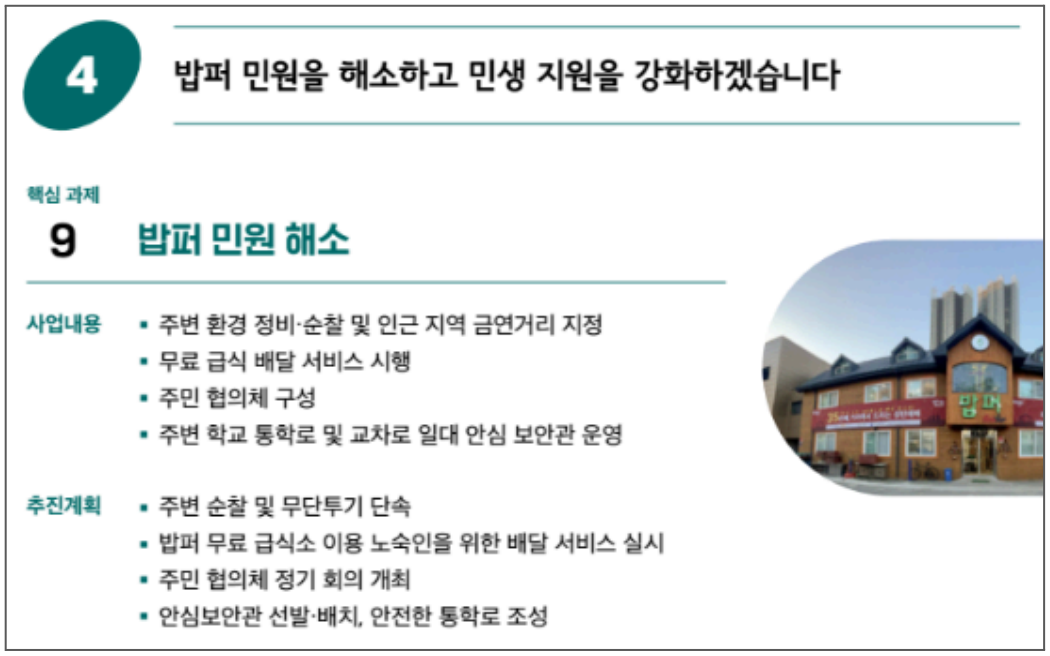
0. 개요

1) 주제 선정 : 동대문구 현안 분석

공공데이터 활용 및 머신러닝 기법 적용을 통한 동대문구에 현존하는 문제 해결방안 도출

2) 문제점 파악 : 무료급식소 '밥퍼'로 인한 민원 속출

동대문구 중장기 주요업무계획인 '비전 **2026**'에 제시된 주요 개선 안건 중 구민 민원사항 해결이 주된 목적인 안건을 본 개발서의 주제로 선정함



청량리역 부근 무료급식소 '밥퍼'에 인근 및 타지 저소득 고령인 / 노숙인이 밀집하여, '치안', '위생', '예산소요', '혼잡' 등 다양한 민원이 속출함.

3) 해결방안 : 대체 무료급식 가능 부지 선정
현재 발생한 민원을 해결하기 위해, '밥퍼'를 대체 할 수 있는 '무료급식'이 가능한 위치 선정

4) 핵심지표 선정 : 민원 사항을 종합적으로 반영하는 대체부지 선정을 위한 핵심지표 선정
- 안전 : 일정 구역의 치안의 정도
- 경쟁 : 무료급식 지원이 가능한 영역 간 침범하지 않는 정도
- 침해 : 무료급식 혜택 비수혜자의 생활이 침해되는 정도
- 위생 : 추가적인 지출 없이 현재의 위생시설을 활용 할 수 있는 정도
- 접근 :

5) 데이터 수집 : 공공 데이터 포털(data.go.kr)에서 OPEN API KEY 등을 활용하여 추출

핵심지표	수집 데이터
안전	서울특별시 경찰서, CCTV 및 가로등 위치 데이터
경쟁	동대문구 자체운영 무료급식소 및 기타 무료급식소 위치 데이터
침해	주거지역, 학교, 아동복시시설 등 교육 관련 시설 위치 데이터
접근	지하철, 버스 등 공공교통시설 위치 데이터
위생	서울특별시 흡연시설, 가로휴지통 위치 데이터
인구밀집도	동대문구 동단위 노인인구 분포

표 1 : 수집데이터

6) 데이터 처리 :

i) 핵심지표 측정용 시설물 데이터 :
동 이름, 주소, 위도 및 경도 등 시설물 위치 파악에 중요한 데이터 반환

ii) 무료급식소 대체 후보지 위치 데이터
동대문구 좌표평면 위 거주지역 분류 위치 데이터를 제외시켜 비거주지역 데이타 집합 추출

7) 머신러닝./알고리즘 활용

i) K-Means 기반 확장형 알고리즘

ii) MCLP(Maximal Covering Location Problem) 기반 사용자 정의 알고리즘 : FULU

1. EDA

1) 라이브러리

i) 기본 라이브러리 (numpy, pandas, matplotlib 등)

- numpy (v. 1.26.4) : 수치 연산과 배열 처리를 위한 필수 라이브러리.
- pandas (v. 2.2.1) : 데이터프레임과 시리즈를 활용한 데이터 분석 및 조작.
- matplotlib (v. 3.8.3) : 데이터 시각화를 위한 기본 플로팅 라이브러리.
- seaborn (v. 0.13.2) : matplotlib 기반으로 더 예쁘고 간단한 시각화 제공.
- scipy (v. 1.12.0) : 과학 계산을 위한 통계, 선형 대수 등 다양한 기능 제공.

ii) 경로 관련 라이브러리

- os (기본 내장, Python v. 3.11 기준) : 운영체제와의 상호작용(파일/디렉토리 관리 등).
- sys (기본 내장, Python v. 3.11 기준)

iii) 시스템 관련 정보 및 파이썬 인터프리터와 상호작용.

- requests (v. 2.31.0) : HTTP 요청을 쉽게 처리(예: API 호출).

iv) 위치 변환 관련 라이브러리
- geopandas (v. 0.14.3) : 공간 데이터를 다루기 위한 라이브러리(pandas 확장).
- geopy (v. 2.4.1) : 지오코딩 및 역지오코딩(주소 ↔ 좌표 변환).
- folium (v. 0.15.1) : 인터랙티브 지도 시각화(Leaflet 기반).
- pyproj (v. 3.6.1) : 좌표계 변환 및 지리 정보 투영 처리
- Geokako (v. 3.6.1)

v) 거리 계산 관련 라이브러리

- haversine (v. 2.8.1) : 두 좌표 간의 구면 거리(대권 거리) 계산.
- geopy.distance (v. 2.4.1) : geopy 내의 거리 계산 모듈(빈센티 공식 등 사용).

vI) String 변환 관련 라이브러리 (re 등)

- re (기본 내장, Python v. 3.11 기준) : 정규 표현식을 활용한 문자열 패턴 매칭 및 변환.

v) ML 관련 라이브러리 (k-means 등)

- scikit-learn (v. 1.4.1) : K-means 클러스터링 및 다양한 ML 알고리즘 제공.

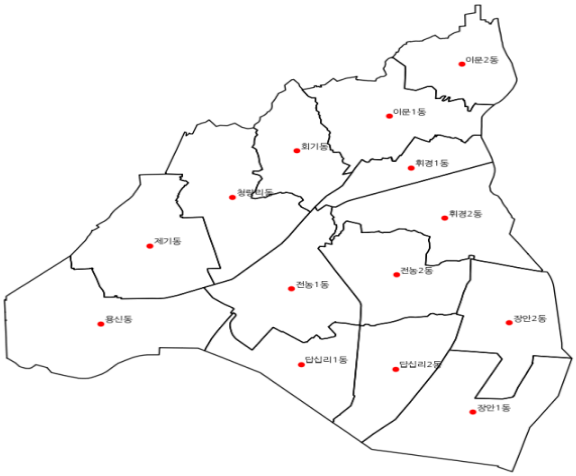
2) 평가지표에 따른 시설물데이터

평가지표 분류	시설분류(대)	시설분류(소)
침해성	교육시설	초중고, 아동복지시설
접근성	교통시설 복지시설	지하철, 버스정류장, 노인복지시설
경쟁성	급식시설	무료급식시설
안전성	방범시설	경찰서,
위생성	흡연시설	흡연시설

표 2 : 평가지표에 따른 데이터 분류

3) 거주지역 제외 대체 후보지 데이터셋

i) 동대문구 그리드화



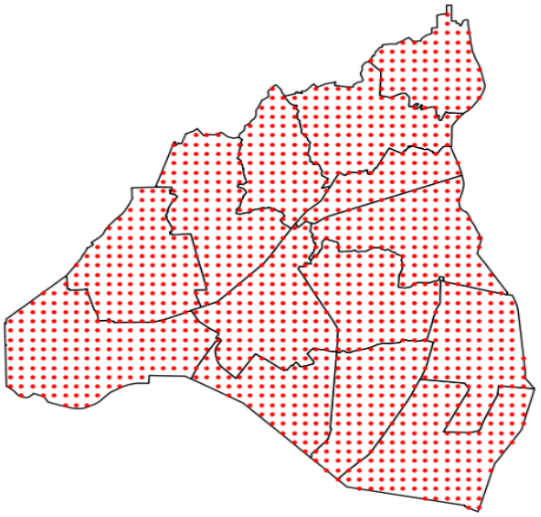
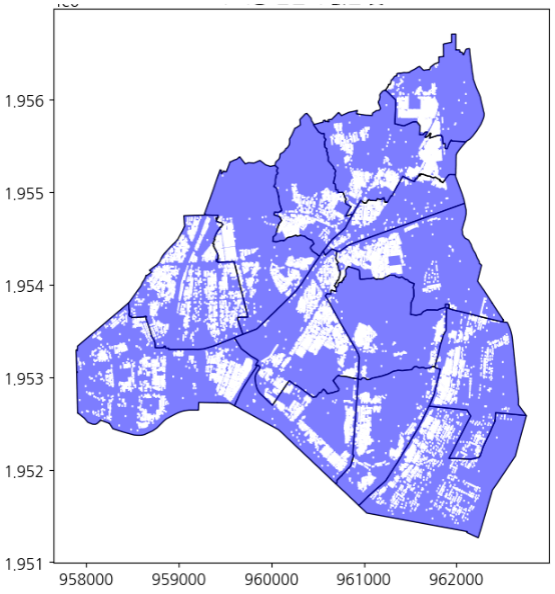
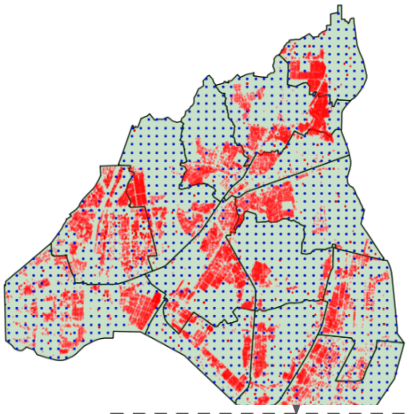


표 3 : 동대문구 동별 그리드

ii) 거주지역 분류 -> 비거주지역 추출





```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 1067 entries, 0 to 1066
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   geometry  1067 non-null   geometry
 1   lon       1067 non-null   float64
 2   lat       1067 non-null   float64
dtypes: float64(2), geometry(1)
memory usage: 25.1 KB
```

표 4 : 동대문구 비거주지역 그리드

2. KMEANS

1) 다기준 의사결정법**(MCDM)** 활용

i) **MCDM**의 정의

- 다수의 기준 또는 목적을 지닌 복잡한 의사결정을 최적화하는 방법론
- 입지선정과 같이 다수의 구성원에게 절대적인 영향을 미치는 정책적 의사결정에 타당한 의사결정 방법론을 활용하는 것이 필수적
- 다속성 의사결정법 : 다기준 의사결정법의 유형 중 하나로, 이미 결정된 유한의 대한 집합에서 대안 간의 우선순위를 평가하는 방법

ii) **KMEANS** 활용

- 현재 보유 데이터 셋이 격자 기반 후보지와 평가지표 5가지라는 점을 고려하였을 때 다양한 지표를 통해 평가를 하며, 유한의 집합에서 우선순위를 도출하는 다속성 의사결정법의 방식을 따를 수 있음
- 다만, 전통적인 MCDM 방법을 활용하지 않고 K-Means를 활용하는 이유는 가중치 부여 등 주관적 판단에 의존하지 않고 데이터에 내재된 패턴을 객관적으로 파악하고 시각화와 해석에 용이한 방법을 사용하기 위해서임

1) 비교 거리 변수 생성

i) 밥퍼와 각 시설 간의 거리**(**기준거리**)**

- 우선, 밥퍼 기준점에 대한 위도, 경도를 추출함 (37.5767036234321, 127.045500241255)
- 이후 거리 계산에서도 활용할 수 있도록 Haversine을 활용하여 사용자 정의 함수를 생성하였으며, 세부적인 파악을 위해 시설분류(대)를 시설 분류 기준으로 결정함
- 밥퍼와 시설분류(대)에 따른 시설과의 최소 거리를 기준거리로 산출함. 사용한 코드는 아래와 같음

```
from haversine import haversine

def haversine_distance(lat1, lon1, lat2, lon2):
    # 위도(lat1, lat2), 경도(lon1, lon2)를 받아 haversine 모듈로 거리(km) 계산.
    return haversine((lat1, lon1), (lat2, lon2), unit='km')

baseline_distances = {}
facility_categories = total_ddm_df['시설분류(대)'].unique()

for cat in facility_categories:
    subset = total_ddm_df[total_ddm_df['시설분류(대)'] == cat]

    min_dist = float('inf')
    for _, row in subset.iterrows():
        lat2 = row['위도']
        lon2 = row['경도']
        dist_km = haversine_distance(benchmark_lat, benchmark_lon, lat2, lon2)
        if dist_km < min_dist:
            min_dist = dist_km

    baseline_distances[cat] = min_dist

print("기준 거리(시설분류(대)별 최소):", baseline_distances)
```

ii) 각 후보지 별 시설과의 거리

- 각 후보지 별로 시설과의 최소거리를 산출함. 이때 기준은 앞에서 정한 시설분류(대)에 해당함
- 산출된 값은 후보지 dataframe에 '시설분류(대)_min_dist' (예. '급식지원사업_min_dist')의 칼럼을 만들어서 할당함
- 사용한 코드는 아래와 같음

```
facility_categories = total_ddm_df['시설분류(대)'].unique()

for cat in facility_categories:
    subset = total_ddm_df[total_ddm_df['시설분류(대)'] == cat]

    # 후보지별로 해당 시설분류(대)의 최소 거리를 계산
    col_name = cat + "_min_dist"
    distances = []

    for _, cand in candidate_gdf_4326.iterrows():
        cand_lat = cand['lat']
        cand_lon = cand['lon']

        min_dist_km = float('inf')
        for _, facility in subset.iterrows():
            fac_lat = facility['위도']
            fac_lon = facility['경도']
            dist_km = haversine_distance(cand_lat, cand_lon, fac_lat, fac_lon)
            if dist_km < min_dist_km:
                min_dist_km = dist_km

        distances.append(min_dist_km)

    # 후보지 GeoDataFrame에 새 컬럼 추가
    candidate_gdf_4326[col_name] = distances
```

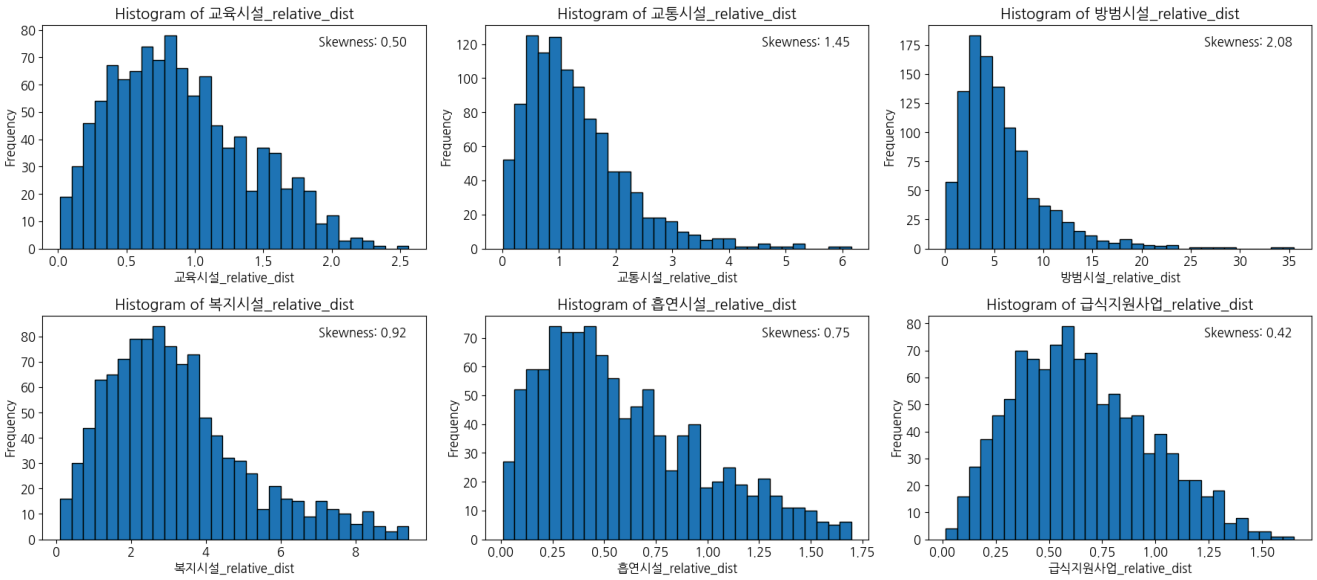
iii) 비교거리 변수 생성

- 비교거리를 구한 공식은 '후보지에서 시설까지의 최소 거리 / 밥퍼 위치에서 시설까지의 최소 거리'임
- 산출된 값은 후보지 dataframe에 '시설분류(대)_relative_distt' (예. '급식지원사업_relative_dist')의 칼럼을 만들어서 할당함
- 1-3)까지의 과정이 끝난 후 후보지들의 dataframe은 아래와 같은 형태를 띔

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 1067 entries, 0 to 1066
Data columns (total 15 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   geometry              1067 non-null   geometry
 1   lon                   1067 non-null   float64
 2   lat                   1067 non-null   float64
 3   교육시설_min_dist         1067 non-null   float64
 4   교통시설_min_dist         1067 non-null   float64
 5   방범시설_min_dist         1067 non-null   float64
 6   복지시설_min_dist         1067 non-null   float64
 7   흡연시설_min_dist         1067 non-null   float64
 8   급식지원사업_min_dist       1067 non-null   float64
 9   교육시설_relative_dist    1067 non-null   float64
 10  교통시설_relative_dist    1067 non-null   float64
 11  방범시설_relative_dist    1067 non-null   float64
 12  복지시설_relative_dist    1067 non-null   float64
 13  흡연시설_relative_dist    1067 non-null   float64
 14  급식지원사업_relative_dist  1067 non-null   float64
dtypes: float64(14), geometry(1)
memory usage: 125.2 KB
```

v) 비교거리 분포 파악

- K-Means와 같은 유클리드 거리를 사용하는 알고리즘은 데이터의 분포에 민감하기 때문에 로그변환 적용이 유리함
- 현재 각 시설분류(대)별로 비교거리의 분포에 대한 히스토그램과 왜도를 출력한 결과 교통시설과 방법시설이 1을 초과하는 왜도를 가진 것으로 파악됨
- 교통시설의 왜도인 1.45의 경우 크게 왜도가 심한 것은 아니지만 보다 정확한 클러스터링을 위해 log 변환을 적용함
- 그 결과 로그변환 후 교통시설은 0.35로, 방법시설은 0.08로 왜도가 조정됨



2) 클러스터링

i) 함수 정의

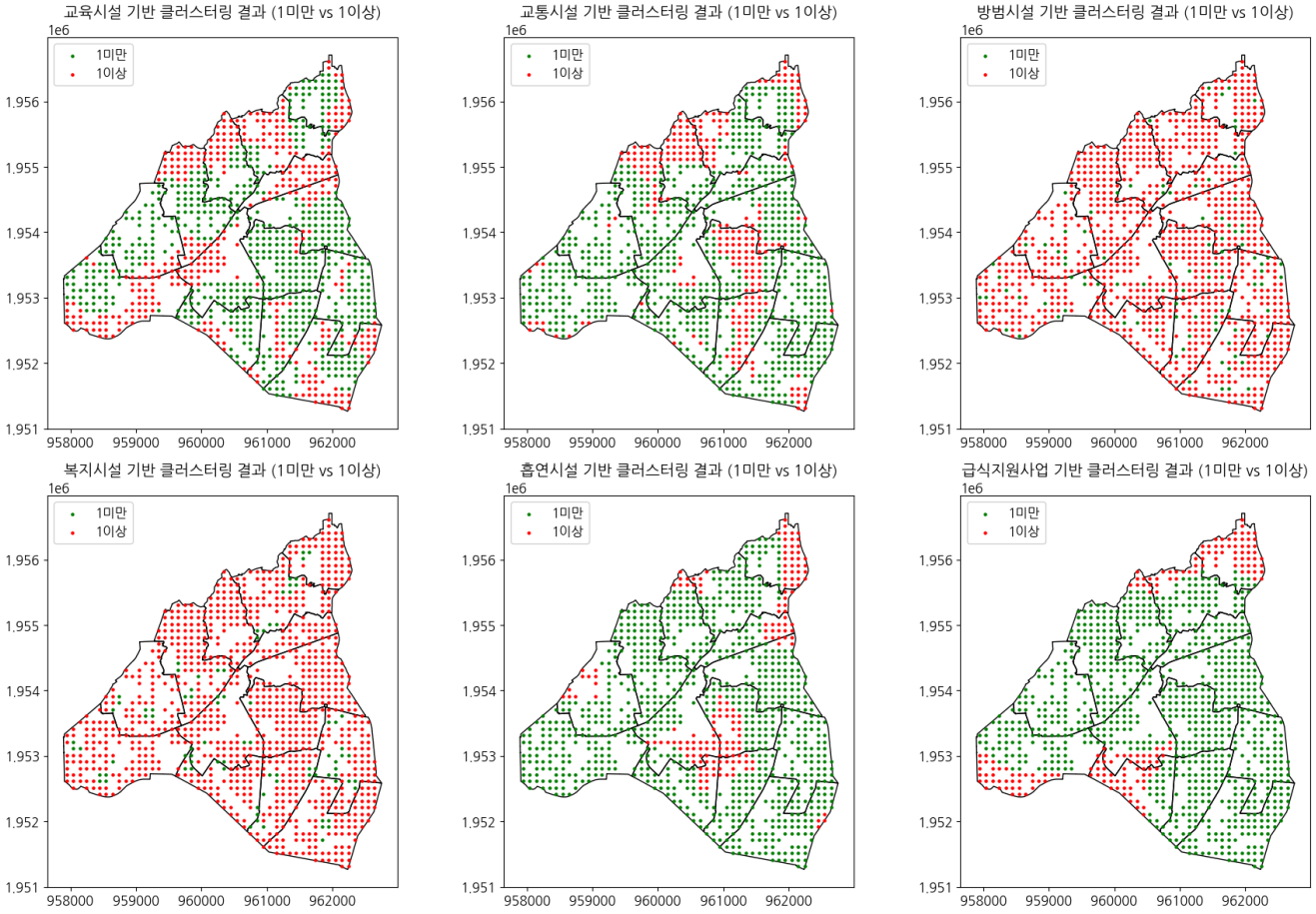
```
def create_cluster(df, column_name, bins, labels):
    df_copy = df.copy()
    df_copy['cluster'] = pd.cut(
        df_copy[column_name],
        bins=bins,
        labels=labels,
        include_lowest=True,
        right=False  # 구간의 오른쪽 경계를 포함하지 않음
    )
    return df_copy
```

ii) 클러스터링

a) 첫번째 시도

- 비교거리가 1미만과 1이상인 것으로 수동 분류함

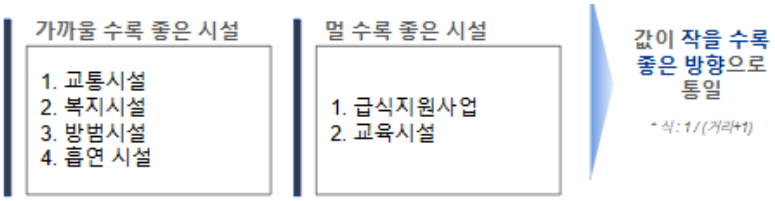
- 그 결과 아래 그림과 같이 시설분류(대)별로 분류된 것의 차이가 과도하게 나타남
- 이는 K-Means를 활용하여 클러스터링을 진행하는 것이 보다 정확한 결과를 도출할 수 있음을 반증함



b) 두 번째 시도**: KMEANS**

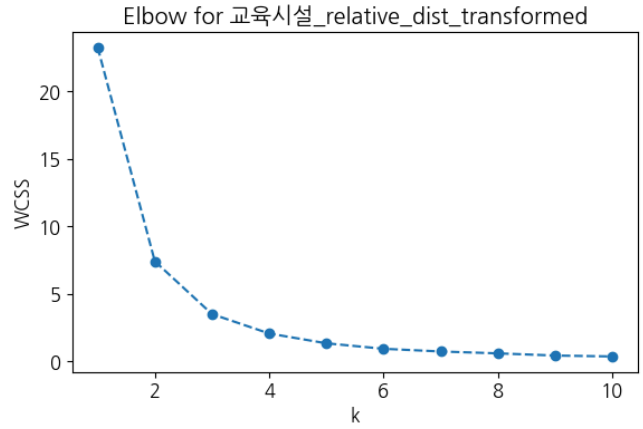
ㄱ) 전처리

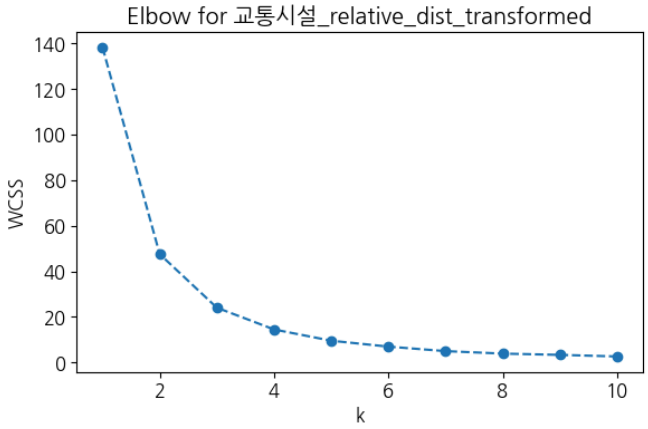
- 현재 변수들은 가까울 수록 좋은 시설과 멀 수록 좋은 시설의 두 가지 방향성을 가지고 있음.
- 값이 작을 수록 좋은 방향으로 변수들의 방향을 통일하여 이후 클러스터링을 진행할 수 있는 형태로 만듦

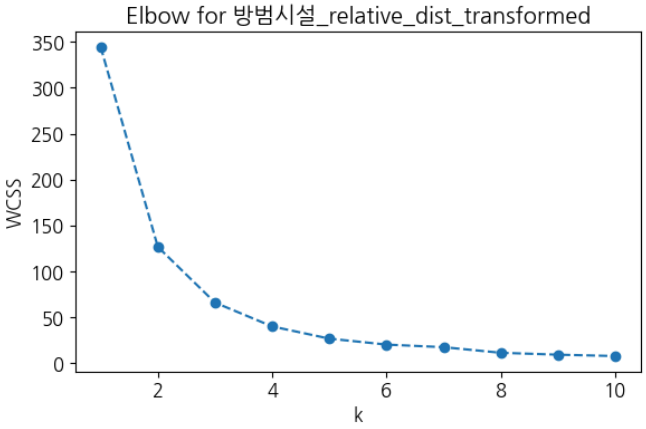


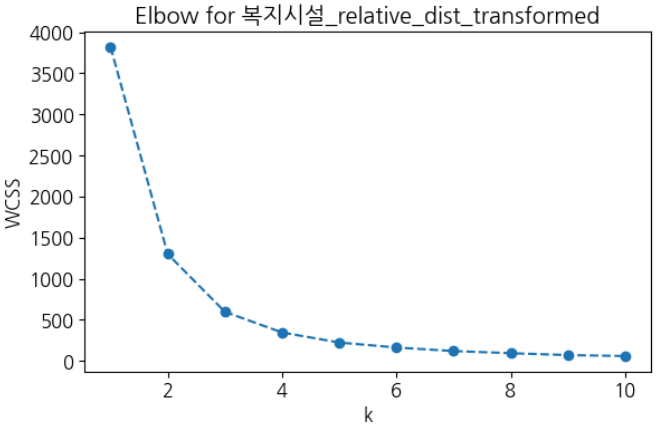
ㄴ) 엘보우 플롯

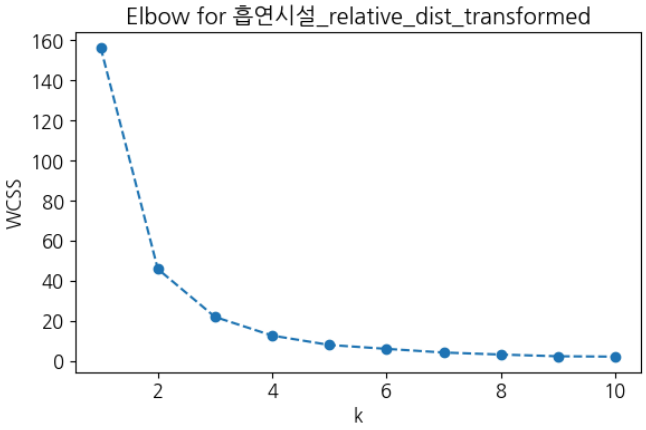
- 엘보우 플롯을 시각화한 결과 아래와 같은 결과를 얻을 수 있음
- K = 2로 설정 시 단순하게 분류되어 세밀한 차이를 보기 어려울 수 있으며, 군집의 다양성을 위해서 K = 3으로 설정 후 우수 후보지들의 겹치는 지점을 확인하기로 함

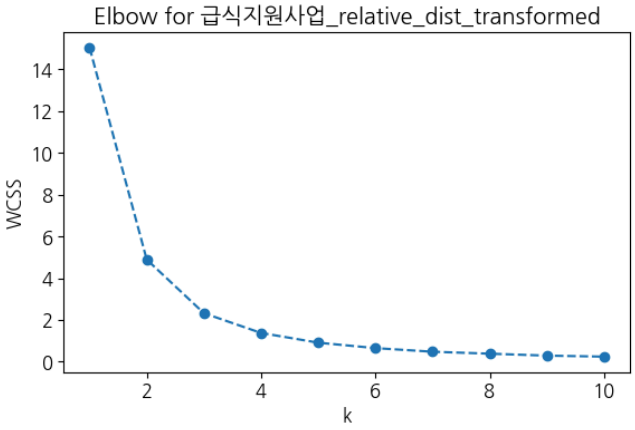










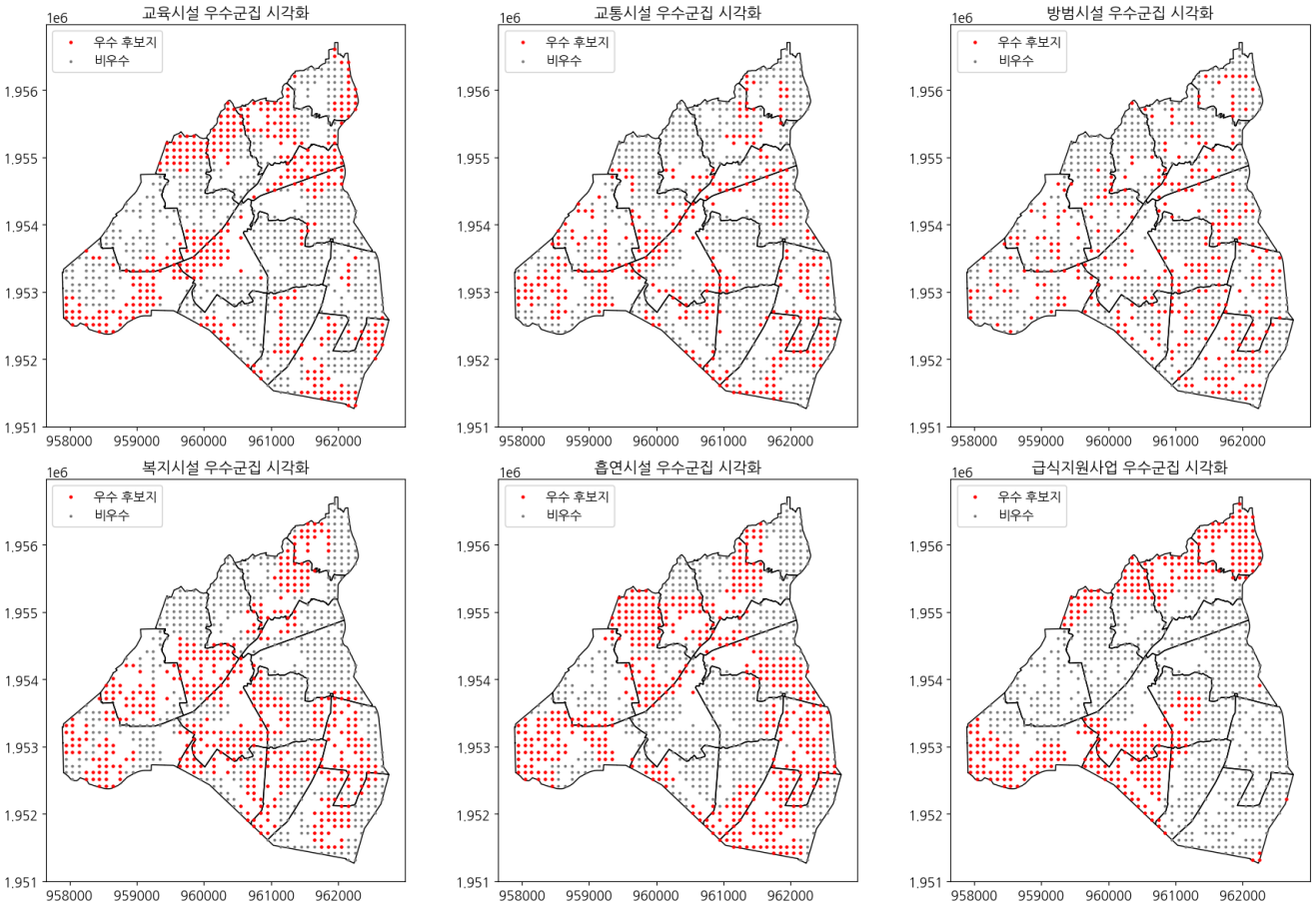


ㄷ) 클러스터링

- 시설 dataframe에 한 번에 적용하여 결과를 추출하기 위하여 클러스터링을 진행할 사용자 지정 함수를 정의함
- 각 변수별로 클러스터링을 진행하여 겹치는 지점을 확인하기 위해 1차원 데이터를 2차원으로 차원 변환하여 비교거리 변수의 데이터 차원을 변환함
- 이후, 비교 거리 값이 가장 작은 데이터를 포함하는 클러스터를 우수 클러스터로 지정하는 코드를 작성함
- 각 시설분류에 대한 우수 후보지를 우수 후보지에 해당하면 True, 해당하지 않으면 False로 지정하여 후보지 dataframe에 칼럼을 생성하여 값을 할당함

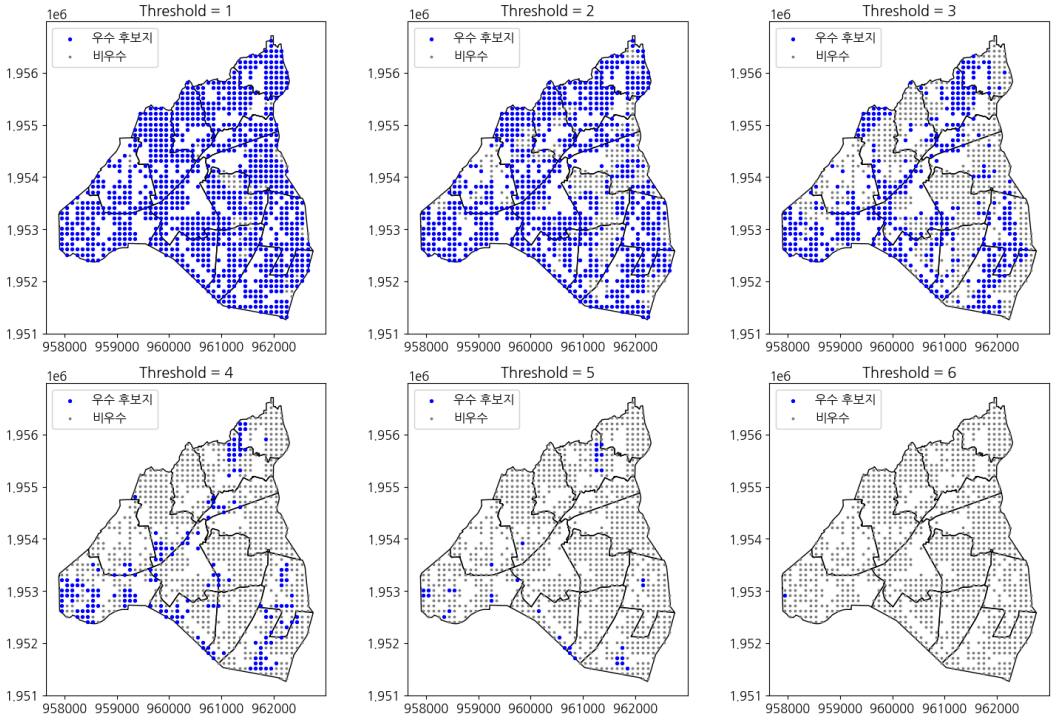
ㄹ) 우수군집 시각화

- 아래 그림은 시설분류(대)에 따라 우수후보지에 해당하는 후보지를 빨간색으로 시각화한 것임



ㅁ) 각 클러스터별 우수 후보지 겹치는 개수 산출

- b-ㅁ)의 각 시설분류(대)의 우수후보지가 겹치는 개수를 더하여 1에서부터 6까지의 값으로 각각 후보지에 'best_count' 컬럼을 만들어서 할당함
- 6개가 모두 겹치는 후보지는 하나만 존재하였기에, 5개 이상 겹치는 후보지들을 최종 우수 후보지로 선정함
- 아래 그림은 후보지가 겹치게 나오는 개수에 따라서 지도를 시각화한 것이며 dataframe 형태의 그림은 현 단계까지의 과정에 따라 최종 산출된 후보지 dataframe의 일부임



교육시설 _relative_dist	...	흡연시설 _relative_dist_transformed	급식지원사업 _relative_dist_transformed	교육 시설 _best	교통 시설 _best	방범 시설 _best	복지 시설 _best	흡연 시설 _best	급식 지원 사업 _best	best_count	final_best
1.710491	...	0.147664	0.443306	True	True	True	False	True	True	5	False
1.465147	...	0.009307	0.456269	True	True	True	False	True	True	5	False
1.181006	...	0.041028	0.469523	True	True	True	True	True	True	6	True
0.907462	...	0.053825	0.482961	False	True	True	True	True	True	5	False
1.102126	...	0.090319	0.481342	True	True	True	True	True	True	6	True
...	...	...	...	...	...	...	...	...	...	...	...
1.109174	...	0.139837	0.742234	True	True	True	True	True	False	5	False

3. FULO

1) 개요

: 기존의 위치로부터 negative 시설물은 적게 있으며, positive시설물은 더 많이 있는 새로운 위치를 찾는 알고리즘 모델이다.

2) 모델 코드

```
class FULO:
    '''
    data: 데이터프레임 형태로 제공됨
    positive_data: 긍정적인 특징 데이터 (많을수록 좋음)
    negative_data: 부정적인 특징 데이터 (적을수록 좋음)
    radius_km: 반지름 파라미터 (단위: km, 추후 정의)
    area: 영역 파라미터 (추후 정의)
```

```
    '''
    def __init__(self, init_location, positive_data, negative_data, radius_km,
area):
        self.init_location = init_location  # 초기 위치 (위치 데이터 예상)
        self.positive_data = positive_data  # 긍정적 데이터프레임
        self.negative_data = negative_data  # 부정적 데이터프레임
        self.radius_km = radius_km          # 반지름 (km 단위)
        self.area = area                    # 영역

    def df_copy(self, df):
        return df.copy()

    def positive_distance(self, standard_locate, data=None):
        if data is None:
            data = self.positive_data
        positive_df = self.df_copy(data)
        if '위도' not in positive_df.columns or '경도' not in positive_df.columns:
            raise ValueError("데이터프레임에 '위도'와 '경도' 열이 필요합니다.")
        positive_df['distance(km)'] = positive_df.apply(
            lambda row: haversine(standard_locate, (row['위도'], row['경도'])), axis=1
        )
        return positive_df[positive_df['distance(km)'] < self.radius_km]

    def negative_distance(self, standard_locate, data=None):
        if data is None:
            data = self.negative_data
        negative_df = self.df_copy(data)
        if '위도' not in negative_df.columns or '경도' not in negative_df.columns:
            raise ValueError("데이터프레임에 '위도'와 '경도' 열이 필요합니다.")
        negative_df['distance(km)'] = negative_df.apply(
            lambda row: haversine(standard_locate, (row['위도'], row['경도'])), axis=1
        )
        return negative_df[negative_df['distance(km)'] < self.radius_km]

    def grid_cell(self):
        # dataframe으로 lat, lon열로 여러 개 존재함.
        grid = pd.read_csv("/content/drive/MyDrive/DDM/candidate_gdf_original.csv")
        return grid[['lon', 'lat']]

    def evaluate_location(self, location):
        pos_df = self.positive_distance(location)
        neg_df = self.negative_distance(location)
        return len(pos_df), len(neg_df)  # 긍정 개수, 부정 개수 반환

    def candidates_info_filtering(self, current_pos, current_neg, candidates_info):
        return [info for info in candidates_info if info['positive'] > current_pos and
info['negative'] < current_neg]

    def recommend_better_location(self):
        # 현재 위치 평가
        current_pos, current_neg = self.evaluate_location(self.init_location)
        print( '현재 밥퍼의 긍정/부정 건물 수', current_pos, current_neg)
        grid = self.grid_cell()

        # 모든 후보 위치의 정보를 저장할 리스트
        candidates_info = []
        best_locations = []
```

```
best_score = -float('inf')

# 모든 후보 위치 평가
for idx, row in grid.iterrows():
    candidate_loc = (row['lat'], row['lon'])
    pos_count, neg_count = self.evaluate_location(candidate_loc)

    # 후보 정보 저장
    candidates_info.append({
        'location': candidate_loc,
        'positive': pos_count,
        'negative': neg_count
    })

return self.candidates_info_filtering(current_pos, current_neg, candidates_info)
```

3) 모델 평가

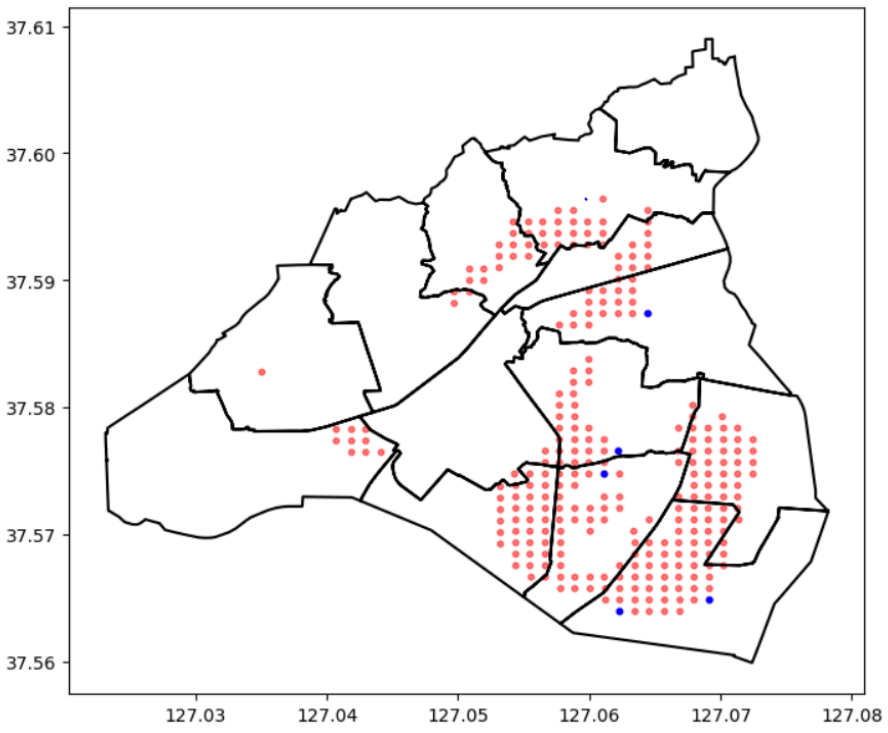
$$1-\left(\frac{\text{모델 추천 위치의 } 550m \text{ 반경이내 } negative\text{개수}}{\text{밥퍼 위치 } 550m \text{ 반경이내 } negative\text{개수}}\right)$$

모델 결과로 나온 후보지들이 실제 밴치마킹 되는 위치보다 얼만큼 향상되었는지에 대한 지표.

후보지들에 대한 향상성이 높은 상위 5개 위치와 최저 향상성을 가진 위치 선정

```
rank 1: [37.56490575716029, 127.06908049830815], 정확도: 46.424%
rank 2: [37.56397943392325, 127.06229199126444], 정확도: 44.815%
rank 3: [37.57479118510581, 127.06109624561525], 정확도: 43.266%
rank 4: [37.58742236150174, 127.06442007446036], 정확도: 43.206%
rank 5: [37.57659804749267, 127.0622181384529], 정확도: 43.147%

최저 점수는 location:[37.58280339643234, 127.03499960706336], 정확도: 0.06%
```



4. 결론

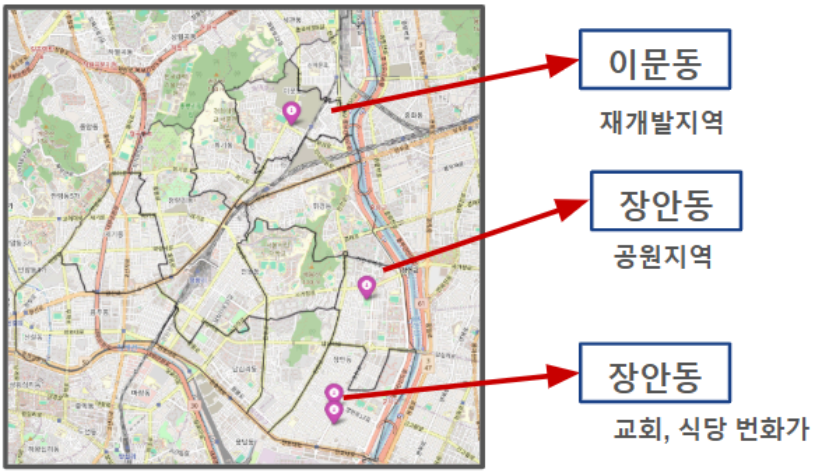
1. 민원 최소화 측정 방법과 민원최대반영 방법을 결합하여 두 조건을 모두 만족하는 위치를 선정함

Shapely	Geopandas
mapping() : Geometry 객체를 GeoJSON 형식으로 반환하여 시각화할 수 있는 형태로 바꿈	**공간 조인 (sjoin)** : 두 dataset의 객체들이 공간적으로 어떻게 연관되어있는지에 따라 데이터 연결
행정동 경계 데이터 => GeoJSON 형식으로 변환	**K-Means 최종 후보지 & FULO 최종 후보지 => intersect**

final_best	index_right	location	positive	negative	lat_right	lon_right
False	148	[37.59642303426622, 127.06096921713821]	225	1661	37.596423	127.060969

1. 이문동 및 장안동 후보지 선정

Folium



2. 각 지역별 특성 확인