데이터마이닝 프로젝트 개인보고서

팀 5조

2023년 12월 11일

이름 : 임채윤

학번 : 60191959

목차

1. 개요

- **보고서 설명**: 이 보고서는 제가 팀원에 기여한 부분에 대해 보다 상세하게 설명하고, 저의 관점에서 분석하는 과정과 프로젝트를 진행하며 느낀점을 담은 개인 보고서입니다.
- **팀 구성 및 역할**
  - **임채윤(60191959)** : 데이터 수집 및 전처리, 그래프 시각화, 머신러닝 활용 (K-Means), 코드 정리 (변수명 정리, 코드 균일화, 주석 설명)
  - 박원석(60191920) : 데이터 수집, 지도 데이터 시각화, 자료조사
  - 최민석(60191976) : 그래프 데이터 시각화, PPT제작, 자료조사
- **팀 주제**: 외국인 거주자를 위한 인프라 개선 방안 (데이터분석)

2. 데이터 수집 및 전처리

```
# 필요모듈 가져오기
import numpy as np
import pandas as pd
import csv
import json

# 데이터 추출
country = pd.read_csv('./data/국적별구.csv')
age = pd.read_csv('./data/연령별구.csv')
purpose = pd.read_csv('./data/체류자격별구.csv')
age_dong = pd.read_csv('./data/연령별동.csv')

# GeoJson파일
dong_geo = './data/seoul-dong_win.geojson'
dong_geo = json.load(open(dong_geo))  # 동
geo_str = './data/skorea_municipalities_geo_simple.json'
gu_geo = json.load(open(geo_str, encoding='utf-8')) # 구

# 지하철 노선 위경도
json_str = './data/subway_data.json'
subway = json.load(open(json_str, encoding='utf-8')) # 서브웨이 데이터

# 동 좌표, 외국인 인구 유형
opp = pd.read_csv('./data/동좌표.csv', encoding='euc-kr')
people = pd.read_csv('./data/외국인인구유형.csv', encoding='utf-8')
```

- **수집 데이터 (아래 4개의 데이터 수집 및 전처리)**
  - 서울시 등록외국인 현황 (국적별/구별) 통계 (국적별구.csv)
  - 서울시 등록외국인 현황 (연령별/구별) 통계 (연령별구.csv)

- 서울시 등록외국인 현황 (체류자격별/구별) 통계 (체류자격별구.csv)
- 서울시 등록외국인 현황 (연령별/동별) 통계 (연령별동.csv)

출처 : 서울열린데이터광장 (https://data.seoul.go.kr/)

예시 자료) 국적별/구별 외국인 거주자 데이터

```
country = country.iloc[1:,1:]
cols_to_drop = [col for col in country.columns if '계' in country[col].values]
country = country.drop(cols_to_drop, axis=1)
country = country.replace('-', 0)
new_columns = ['연도', '국적', '성', '인원수', '자치구']
country_2020 = country.filter(regex='2020').T.reset_index(drop=False)
country_2021 = country.filter(regex='2021').T.reset_index(drop=False)
country_2022 = country.filter(regex='2022').T.reset_index(drop=False)
country_2023 = country.filter(regex='2023').T.reset_index(drop=False)
country_2020['index'] = 2020
country_2021['index'] = 2021
country_2022['index'] = 2022
country_2023['index'] = 2023
region = country['자치구별(2)'][2:]
region = region.reset_index(drop=True)
# 2023년도 데이터 활용 (복붙 및 변수명 연도만 바꾸면 여러 연도 데이터 활용 가능)
country_current = pd.DataFrame()
for i in range(3, 28):
    df = country_2023.loc[:, ['index',1,2,i]]
    df['자치구'] = region[i-3]
    df.columns = new_columns
    country_current = pd.concat([country_current, df])
country_current = country_current[['연도', '자치구', '성', '국적', '인원수']]
country_current['인원수'] = country_current['인원수'].astype('int64')
```

← 전처리 코드

country

0.0s

	자치구별(1)	자치구별(2)	2020 3/4	2020 3/4.1	2020 3/4.2	2020 3/4.3	2020 3/4.4	2020 3/4.5	2020 3/4.6	2020 3/4.7	...	2023 3/4.167
0	자치구별(1)	자치구별(2)	합계	합계	합계	합계	합계	합계	합계	합계	...	합계
1	자치구별(1)	자치구별(2)	한국계 중국인	한국계 중국인	한국계 중국인	중국	중국	중국	미국	미국	...	덴마크
2	자치구별(1)	자치구별(2)	계	남자	여자	계	남자	여자	계	남자	...	여자
3	합계	종로구	1464	685	779	3180	1022	2158	409	226	...	10
4	합계	중구	1698	875	823	2305	962	1343	277	179	...	18
5	합계	용산구	1083	532	551	596	221	375	2536	1461	...	30
6	합계	성동구	2014	998	1016	2200	759	1441	201	125	...	2
7	합계	광진구	5385	2746	2639	3874	1259	2615	169	95	...	2
8	합계	동대문구	2358	1182	1176	5682	1740	3942	180	83	...	5
9	합계	중랑구	1565	768	797	637	200	437	58	34	...	1
10	합계	성북구	1002	516	486	3811	1267	2544	263	140	...	9
11	합계	강북구	1020	475	545	697	216	481	78	50	...	-
12	합계	도봉구	442	212	230	384	104	280	74	39	...	-
13	합계	노원구	516	240	276	864	377	487	183	100	...	-
14	합계	은평구	1115	531	584	765	271	494	150	98	...	2
15	합계	서대문구	831	390	441	3421	893	2528	496	265	...	16
16	합계	마포구	1125	535	590	2883	798	2085	1001	463	...	8
17	합계	양천구	1677	893	784	652	232	420	129	66	...	1
18	합계	강서구	2369	1199	1170	990	349	641	206	117	...	2
19	합계	구로구	22541	13857	8684	5645	2356	3289	108	56	...	-
20	합계	금천구	13526	8117	5409	2586	1055	1531	44	28	...	1

← 기존 데이터

country_current

✓ 0.0s

	연도	자치구	성	국적	인원수
0	2023	종로구	남자	한국계 중국인	415
1	2023	종로구	여자	한국계 중국인	434
2	2023	종로구	남자	중국	1095
3	2023	종로구	여자	중국	2890
4	2023	종로구	남자	미국	526
...	...	...	...	...	...
113	2023	강동구	여자	모 로 코	4
114	2023	강동구	남자	이 라 크	0
115	2023	강동구	여자	이 라 크	0
116	2023	강동구	남자	기 타	17
117	2023	강동구	여자	기 타	22

← 변환 후 데이터

- 통계 자료를 데이터프레임으로 불러와 위와 같이 사용하기 용이한 형태로 전처리함. 그래프 시각화, 지도 시각화, 머신러닝 등에 활용할 데이터를 위와 같은 형식으로 모두 전처리하였음.

그래프 시각화를 위한 데이터전처리

그래프 활용 목적 전처리

```
# 상위 10개국가 데이터 활용
country_2023 = country_current.query('연도 == 2023')
country_count = country_2023.groupby('국적')['인원수'].sum().reset_index()
# display(country_count.sort_values('인원수', ascending=False).nlargest(10, '인원수')) # 인원수 상위 국가 10

# 기타 인원에 3300명 아래의 국가들을 모두 합함
etc = country_count[country_count['인원수']<3300]['인원수'].sum()
country_count.iloc[1, 1] = country_count.iloc[1, 1] + etc
country_top10 = country_count.nlargest(10, '인원수') # 3300명 아래부터 11등..

# 주요 체류자격별 인원수 연도에 따른 추이를 나타내는 그래프를 위한 데이터
purpose_sex_population = purpose.groupby(['연도', '체류자격', '성'])['인원수'].sum().reset_index()
purpose_sex_population = purpose_sex_population[(purpose_sex_population['체류자격']=='결혼이민(F6)')
                                                |(purpose_sex_population['체류자격']=='유학(D2)')
                                                |(purpose_sex_population['체류자격']=='방문취업(H2)')]
```

- 국내 거주 외국인 58개의 국가를 모두 나타내면 국적별 비율 그래프가 매우 난잡해지는 문제 발생. 국내에 많이 거주하는 외국인 대상의 인프라를 우선적으로 개선하고자 상위 10개 국가를 뽑아내어 활용. 10위는 3304명의 인원수를 가진 '프랑스', 그 아래 인원수인 국가(top11 이후)는 '기 타' 국가로 병합.

- 체류자격별 인원수가 연도에 따라 어떻게 변화하는지를 나타내기 위한 데이터

지도 시각화를 위한 데이터전처리

지도 활용 목적 전처리

```
# 자치구별 인원수 데이터프레임
gu_population = country_current.groupby('자치구')['인원수'].sum().reset_index()
# 동별 인원수 데이터프레임
dong_population = age_dong.groupby('동별')['인원수'].sum().reset_index()
```

← 전처리 코드

dong_population

✓ 0.0s

	동별	인원수
0	가락1동	398
1	가락2동	556
2	가락본동	895
3	가리봉동	12847
4	가산동	7985
...	...	...
422	효창동	391
423	후암동	1484
424	휘경1동	6317
425	휘경2동	2398
426	흑석동	6178

427 rows × 2 columns

gu_population

✓ 0.0s

	자치구	인원수
0	강남구	5334
1	강동구	4068
2	강북구	4649
3	강서구	5639
4	관악구	15410
5	광진구	15355
6	구로구	22886
7	금천구	13465
8	노원구	4434
9	도봉구	2393
10	동대문구	17875
11	동작구	10586
12	마포구	11382

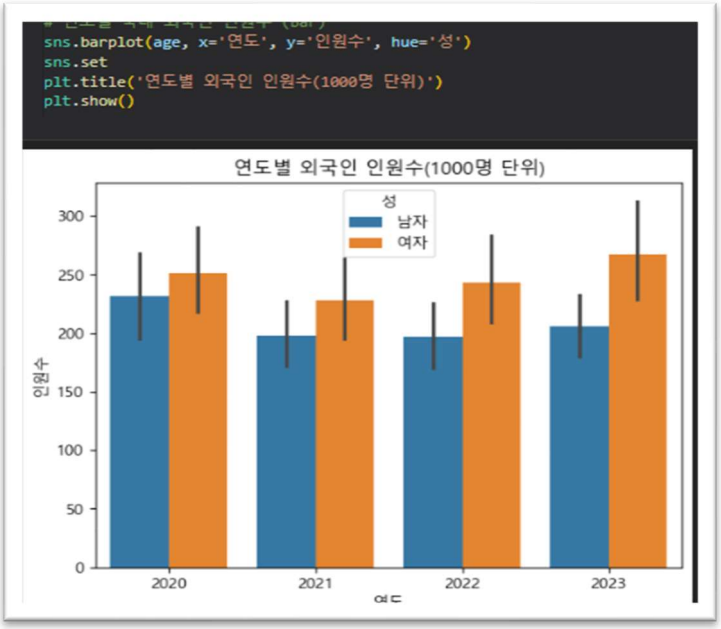
← 동별, 구별 인원수

- 지도 시각화 Choropleth 기능에 사용하기 위한 지역단위 인구수 데이터를 만듦. 전처리데이터를 활용하여 각각 구와 동을 기준으로 groupby를 통해 그룹화하고 인원수에 대한 데이터를 sum()메소드를 적용하여 동별, 구별 인원수를 구함. 동 데이터의 경우 남/여 인원은 있었으나 합계 인원수는 없었기에 age_dong['남자']+age_dong['여자'] 를 통해 age_dong['인원수'] 라는 새로운 column을 생성하여 나타냄.

3. 그래프 시각화 및 머신러닝(K-Means) 활용

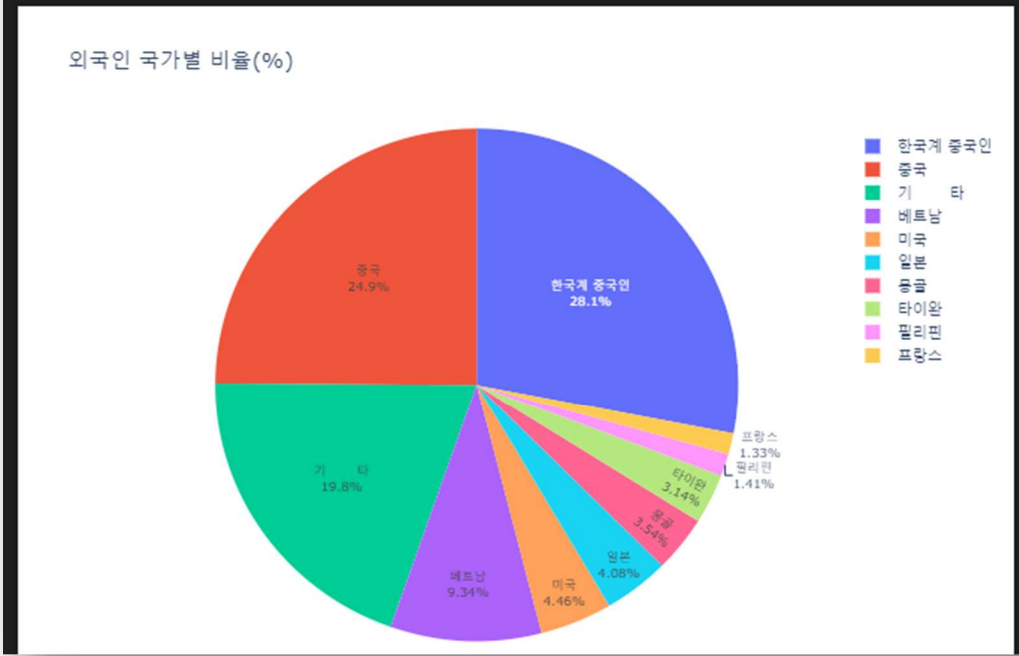
- 위에서 전처리한 데이터를 기반으로 matplotlib, seaborn, plotly를 활용하여 데이터를 그래프로 시각화하였음.

```
sns.barplot(age, x='연도', y='인원수', hue='성')
sns.set
plt.title('연도별 외국인 인원수(1000명 단위)')
plt.show()
```



← 연도별 외국인 인원수 그래프(1)

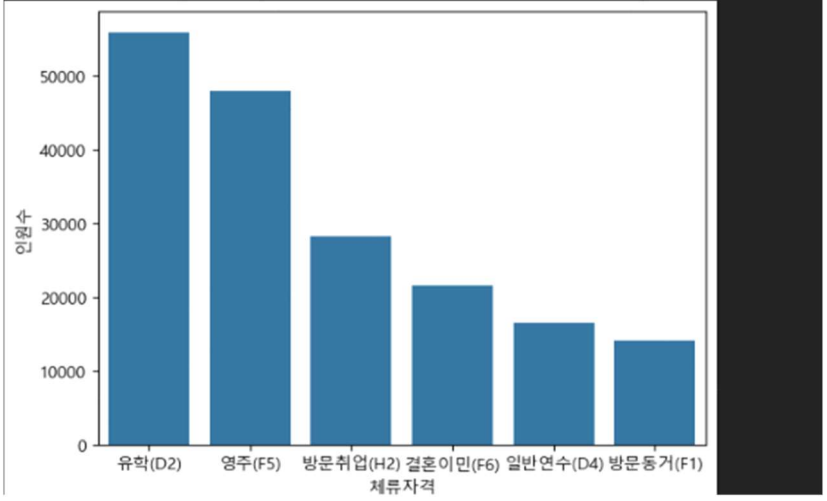
```
# 국내 국적별 외국인 인구 비율 그래프(Pie)
fig = px.pie(country_top10, values=country_top10['인원수'], names=country_top10['국적'], title='외국인 국가별 비율(%)')
fig.update_traces(textinfo = 'label+percent',
                  textfont_size = 10,
                  textposition = 'auto')
fig.update_layout(width=800, height=600)
fig.show()
```



↑ 외국인 국적별 비율(2)

```
# 체류자격 별 인원수 그래프 (Bar)
purpose_population = purpose_current.groupby('체류자격')['인원수'].sum().reset_index()
purpose_top6 = purpose_population.nlargest(6, '인원수')
sns.barplot(purpose_top6, x='체류자격', y='인원수')
```

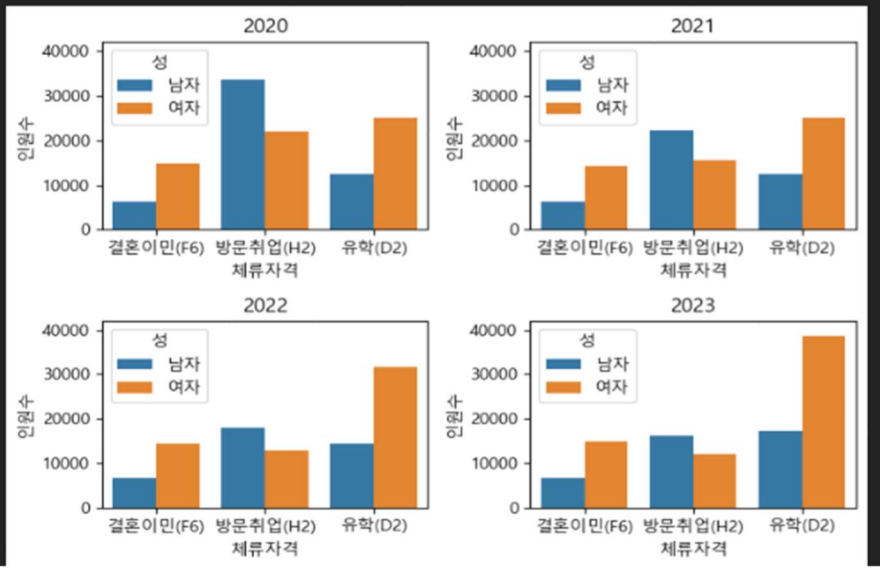
```
Axes: xlabel='체류자격', ylabel='인원수'>
```



← 체류자격별 인원(3)

```
purpose_sex_population = purpose.groupby(['연도', '체류자격', '성'])['인원수'].sum().reset_index()
purpose_optimize = purpose_sex_population[(purpose_sex_population['체류자격']=='결혼이민(F6)')
                                          |(purpose_sex_population['체류자격']=='유학(D2)')
                                          |(purpose_sex_population['체류자격']=='방문취업(H2)')]
plt.figure(figsize=(7,5))
for i in range(2020, 2024):
    plt.subplot(2, 2, i-2019)
    sns.barplot(purpose_optimize.query(f'연도 == {i}'), x='체류자격' , y='인원수', hue='성')
    plt.title(i)
    plt.ylim(0, 42000)
plt.tight_layout()
plt.show()
```

```
✓ 0.3s
```



↑ 연도에 따른 주요 체류자격별 인원수 그래프(4)

← 자치구 연도별 추이(5)

그래프 분석

- (1)2020년부터 코로나로 인하여 외국인 인원수가 점차 감소하는 경향을 보였으나, 코로나 입국제한이 해제된 이후부터 빠르게 외국인 인구수가 증가하고 있음. 여성 외국인 인원수가 남성 외국인 인원수에 비해 항상 높았는데, 점차 그 차이가 커지고 있음.

- (2)국적별 외국인 분포는 1위 '한국계 중국인'(28.1%), 2위 '중국인'(24.9%), 3위 '베트남'(9.34%), 4위 '미국'(4.46%), 5위 '일본'(4.08%), 6위 '몽골'(3.54%), 7위 '타이완'(3.14%), 8위 '필리핀'(1.41%), 9위 '프랑스'(1.33%) 이다. 중국인의 비율이 약 53.0%로 매우 높은 수치를 보였고, 이들은 주로 '영등포구, 구로구'에 대다수 밀집하여 거주하며 이미 인근에 외국인 시설이 많이 설치되어 인프라 개선이 급한 경우는 아니었음.
베트남, 미국, 일본, 몽골 등 인원들을 목적으로 인프라 개선을 진행하는 것을 권장.

- (3)체류자격별 인원수는 유학, 영주, 방문취업, 결혼이민, 일반연수, 방문동거 순으로 많다. 이 중 한국어에 관한 전문성을 가진 '영주' 외에 가장 많은 인원수를 가진 체류자격을 세 가지 분류하면 '유학', '방문취업', '결혼이민'이 나타남.
인원이 많은 부류일수록 관련 외국인 복지시설이 부족할 가능성이 큼. 이에 따라 위 세 가지를 바탕으로 '유학' 인구가 많은 지역에는 서류처리와 같은 문제를 해결할 수 있는 **지원센터**, '방문취업'이 많은 지역에는 일자리를 구할 수 있는 **일자리센터**, '결혼이민'이 많은 지역에는 언어적인 문제해결을 위한 **어학당센터**를 우선 배치하도록 권장.

- (4)연도에 따라 체류자격별 인원수의 변화 추이를 살펴보면, 결혼이민의 경우 2020년부터 일정한 수치를 보였음. 하지만 **방문취업과 유학**은 지난 3년 간 매우 큰 변화를 보임. 방문취업 인원은 2020년부터 크게 감소하여 3년 사이에 무려 2배가 넘는 인원이 감소했음. 반면 유학생의 경우 2020년부터 '여학생'을 기준으로 큰 증가폭을 이루었음. **코로나 시대**에 많은 **방문취업 인구가 감소**하였지만, K-POP을 필두로 **한류 문화가 매우 크게 발전**하며 한류 문화에 관심이 많은 **유학(10-20대) 인구가 급증**한 것으로 보임.

```
age.groupby(['자치구', '나이'])['인원수'].sum().reset_index().sort_values('인원수', ascending=False)
✓ 0.0s
```

	자치구	나이	인원수
214	동대문구	20~24세	24532
410	영등포구	55~59세	18890
137	구로구	55~59세	18859
215	동대문구	25~29세	15122
340	성북구	20~24세	15052

← 나이별 주요 거주지

- (5)실제로 자치구 연도별 추이를 통해 구로구, 영등포구와 같은 지역이 방문취업 인구가 많은 지역이며 그 인원수가 감소하고 있음. 또한 동대문구, 성북구와 같은 연령층이 낮은 인구가 많이 거주하는 지역에는 점차 인구가 증가하고 있는데, 이는 곧 유학 인구가 동대문구, 성북구와 같은 지역에 다수 유입되었음을 알 수 있음.

머신러닝 (K-Means) 활용

- 박원석 팀원이 외국인 지원시설, 인구밀집 Choropleth 지도, 서울 지하철역 좌표를 구하고 이를 활용하여 **직접** 최적의 지원시설 좌표를 탐색함.
- 팀 목표 : 직접 최적의 좌표를 구하고, 머신러닝을 통해 최적의 좌표를 마찬가지로 구하여 이 둘이 어느정도 유사한지를 알아보려 함. => 이 과정에서 **머신러닝 구현**을 맡음.

- **구현 과정**

1. 서울시 범위 내에 약 2000개의 랜덤 위경도 좌표 생성
2. 지하철역 위경도 데이터를 활용하여 각 좌표마다 가장 가까운 위경도의 동과 역을 탐색 (유클리디언 거리 사용)
3. 인근 지하철역의 거리, 인근 동의 외국인 인구수를 피처로 삼아 정규화(Scale)
4. 정규화 된 피처들에 가중치를 설정하여 각 좌표의 점수를 계산. 거리가 가까울수록, 인근 동의 외국인 인구수가 많을수록 점수가 높아짐. 이상치를 막기 위해 시그모이드 함수를 사용

5. 해당 점수를 K-Means 군집화의 가중치로 사용. 가중치 값엔 제곱승을 하여 점수가 높을수록 더욱 높은 점수가 되도록 설정.

6. K-Means 군집화 알고리즘을 통해 점수를 가중치로 사용한 군집 15개를 생성하고, 해당 군집의 중앙좌표를 도출.

7. folium 맵에 마커로 표시하여 해당 위치가 적절한 위치인지 파악

```
gdf = gpd.GeoDataFrame.from_features(dong_geo["features"])
def generate_random(number, polygon):
    points = []
    minx, miny, maxx, maxy = polygon.bounds
    while len(points) < number:
        pnt = Point(random.uniform(minx, maxx), random.uniform(miny, maxy))
        if polygon.contains(pnt):
            points.append(pnt)
    return points
# 424개 동에 랜덤좌표 5개씩 뽑기
points = []
for i in range(424):
    points += generate_random(5, gdf.geometry[i])
# 지하철 데이터프레임 생성
subway_df = pd.DataFrame(subway).T.reset_index()
subway_df.columns = ['역', 'x좌표', 'y좌표']
# 랜덤좌표의 데이터프레임
train = pd.DataFrame()
point = []
for i in range(len(points)):
    point.append(points[i].y)
train['x좌표'] = point
point = []
for i in range(len(points)):
    point.append(points[i].x)
train['y좌표'] = point
```

← (1)랜덤좌표 생성

```
# 역이 어느동에 근접하는지 파악
distances = cdist(dong_info[['위도', '경도']], subway_df[['x좌표', 'y좌표']], metric='euclidean')
closest_dong_index = distances.argmin(axis=0)
dong_name = []
dong_amount = []
for i in range(len(subway_df)):
    dong_name.append(dong_info.iloc[closest_dong_index[i]]['동별'])
    dong_amount.append(dong_info.iloc[closest_dong_index[i]]['인원수'])
subway_df['소속동'] = dong_name
subway_df['동인원수'] = dong_amount
✓ 0.0s

# 각 train 좌표에 대해 가장 가까운 역 찾기
distances = cdist(subway_df[['x좌표', 'y좌표']], train[['x좌표', 'y좌표']], metric='euclidean')
closest_station_index = distances.argmin(axis=0)
closest_station_distance = distances.min(axis=0)
station_distance = []
station_name = []
dong_name = []
dong_amount = []
for i in range(len(train)):
    station_distance.append(closest_station_distance[i])
    station_name.append(subway_df.iloc[closest_station_index[i]]['역'])
    dong_name.append(subway_df.iloc[closest_station_index[i]]['소속동'])
    dong_amount.append(subway_df.iloc[closest_station_index[i]]['동인원수'])
train['인접역 거리'] = station_distance
train['인접역'] = station_name
train['소속동'] = dong_name
train['동인원수'] = dong_amount
# 위도 1도는 약 111km 차이, 경도 1도는 약 88km 차이 거리가 루트2는 약 143km
# 2km의 위경도 거리는 약 0.019779. 따라서 이보다 가까운 역만 생각함
train = train[train['인접역 거리'] < 0.019779]
✓ 0.1s
```

← (2)

```
# 스케일러를 활용한 데이터 정규화
scaler = StandardScaler()
df = pd.DataFrame({'인접역 거리':train['인접역 거리'], '동인원수':train['동인원수']})
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
0.0s
```

↑ (3) 인접역거리 피처와 동인원수 피처는 값의 차이가 너무 크기에 그 규모를 표준스케일러로 정규화

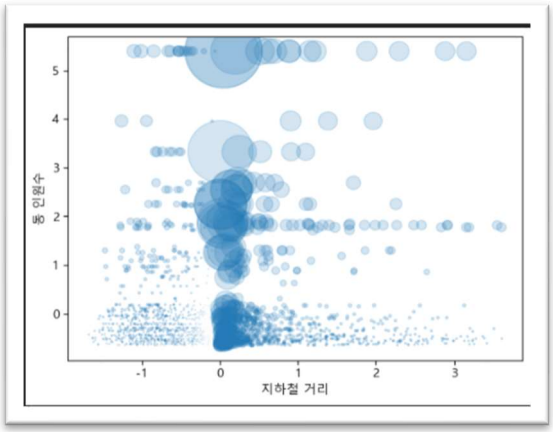
```
def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))
# 가중치 설정
weight_distance = 0.3  # 인접역 거리에 대한 가중치
weight_population = 0.7  # 동인원수에 대한 가중치
# 각 좌표의 점수 계산
df_scaled['score'] = (sigmoid(weight_distance / df_scaled['인접역 거리']) + sigmoid(weight_population * df_scaled['동인원수']))/2
```

↑ (4) 가중치와 시그모이드 함수를 사용하여 각 좌표 index별로 점수를 계산

```
alpha = 2.3
df_scaled['score'] = (df_scaled['score']* alpha) ** 10
# KMeans 알고리즘으로 최적의 좌표 15개 찾기
kmeans = KMeans(n_clusters=15, random_state=34).fit(train[['x좌표', 'y좌표']], sample_weight=df_scaled['score'])
optimal_coordinates = kmeans.cluster_centers_
optimal = pd.DataFrame()
optimal_x = []
optimal_y = []
# 결과 출력
for i, coord in enumerate(optimal_coordinates):
    optimal_x.append(coord[0])
    optimal_y.append(coord[1])
    print(i,'번째 최적의 좌표는', coord[0],',',coord[1],'입니다')
optimal['최적x'] = optimal_x
optimal['최적y'] = optimal_y

plt.scatter(df_scaled.iloc[:, 0], df_scaled.iloc[:, 1], alpha=0.2, s=df_scaled['score'])
plt.xlabel('지하철 거리')
plt.ylabel('동 인원수')
```

```
0 번째 최적의 좌표는 37.58608608461523 , 127.00320567888043 입니다
1 번째 최적의 좌표는 37.49528296399196 , 126.90760224456447 입니다
2 번째 최적의 좌표는 37.54359800804676 , 127.0765073915966 입니다
3 번째 최적의 좌표는 37.50973915849634 , 126.95874972742878 입니다
4 번째 최적의 좌표는 37.59617220706286 , 127.06901486935732 입니다
5 번째 최적의 좌표는 37.57042047824265 , 126.92688099555716 입니다
6 번째 최적의 좌표는 37.5477563311308 , 126.84511707149971 입니다
7 번째 최적의 좌표는 37.49523204627288 , 126.87849787276929 입니다
8 번째 최적의 좌표는 37.65387684013299 , 127.04147056561101 입니다
9 번째 최적의 좌표는 37.52089555866615 , 127.12726443377167 입니다
10 번째 최적의 좌표는 37.486847443561146 , 126.99924349023532 입니다
11 번째 최적의 좌표는 37.52297501736276 , 127.05535888339664 입니다
12 번째 최적의 좌표는 37.473700647056894 , 126.89783874401398 입니다
13 번째 최적의 좌표는 37.54680161690094 , 126.9803741218683 입니다
14 번째 최적의 좌표는 37.60347249178824 , 127.04230902514193 입니다
```



↑ (5, 6) 위에서 점수를 KMeans 알고리즘의 가중치로 하여 점수가 높은(역이 가깝고, 인구수가 많은) 샘플을 가중치를 크게 주어 15개의 군집을 생성, 이후 군집의 중앙좌표들을 최적좌표로 출력함. 오른쪽은 지하철거리와 동인원수에 따른 점수를 제곱승하여 점수가 큰 좌표의 가중치를 높은 것을 보여주는 그래프

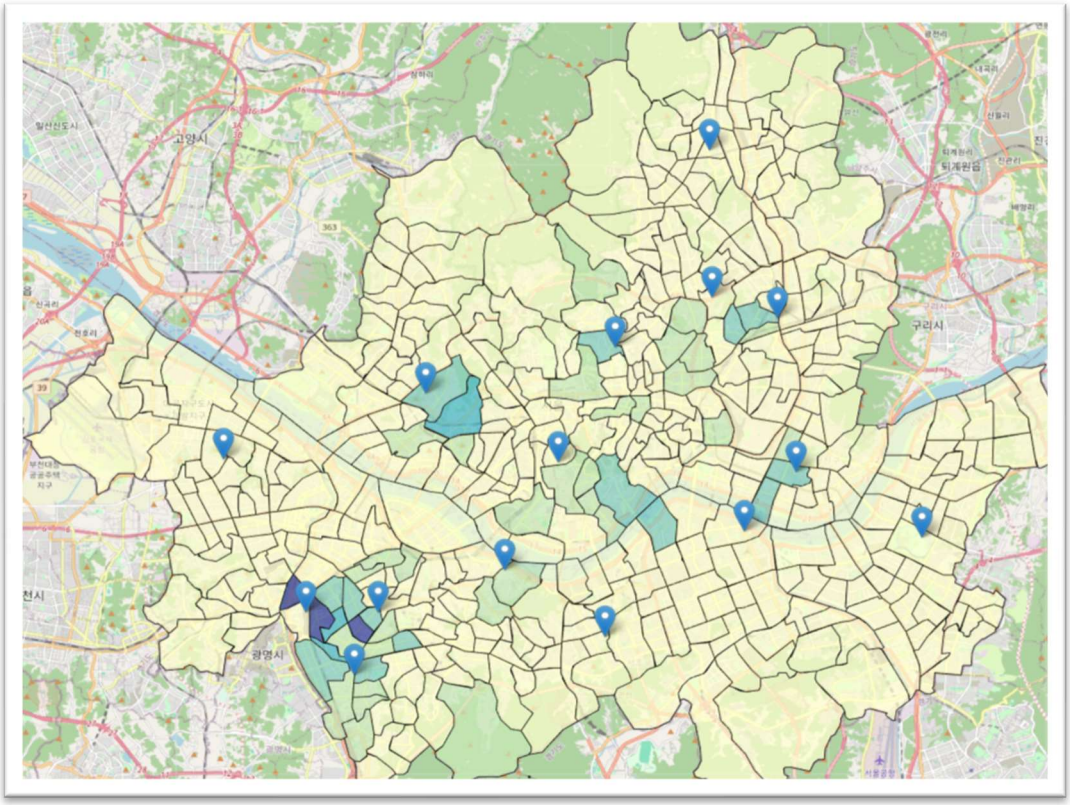
최적의 위치 좌표

```
# 서울의 중심을 기준으로 지도를 생성합니다.
map_osm = folium.Map(location=[37.5665, 126.9780], zoom_start=11)

choropleth_layer = folium.Choropleth( geo_data=dong_geo, data=dong_population, columns=('동별', '인원수'),
                    key_on='feature.properties.동',
                    fill_color='YlGnBu',
                    legend_name='외국인 거주 수', fill_opacity=0.7).add_to(map_osm)
region_data = dong_population.set_index('동별')
region_data = region_data.astype('object')
# geojson내에 인원수 정보 추가 후 툴팁 추가 (채)
for feature in choropleth_layer.geojson.data['features']:
    feature['properties']['인원수'] = region_data.loc[feature['properties']['동'], '인원수']
folium.GeoJsonTooltip(['구','동', '인원수']).add_to(choropleth_layer.geojson)
folium.LayerControl().add_to(map_osm)

""" # 서울 지하철 노선 위치 현황
for station_name, coordinates in subway.items():
    folium.CircleMarker(location=coordinates, popup=station_name, radius=2, color='blue', opacity=0.7).add_t

# 각 좌표를 지도에 표시합니다.
for point in optimal.values:
    folium.Marker(point).add_to(map_osm)
map_osm
```



↑ (7) K-Means를 사용하여 구한 최적의 좌표. 인구가 많고 지하철 역이 가까운 곳을 토대로 군집화가 진행됨. 간혹 인구가 적은 지역임에도 좌표가 찍히는데, 이는 인구가 적은 지역들을 모은 군집의 대푯값(중앙좌표)이다.

머신러닝 분석

- 지하철과의 거리, 인구수를 기준으로 가중치를 만들고 해당 가중치를 기준으로 군집화를 진행하였음. 결과를 도출하고 분석하는 과정에서 아쉬움이 있었다면, K-Means 알고리즘을 사용하면 점수가 높은 지역에 좌표를 얻어낼 순 있지만, 맵 공간에 찍힌 마커들을 보았을 때 점수가 높은 지역이 아니더라도 좌표가 찍히게 됨. K-Means는 비지도학습 군집화 알고리즘이기에 점수를 사용하여 더 좋은 좌표와 좋지 않은 좌표를 나눌 순 있지만, 더 좋은 좌표를 판별하는 알고리즘은 아님.

- 그렇기에 또다른 머신러닝 이용기법을 제시하면, 로지스틱회귀와 같은 '분류' 모델을 사용하여 각 좌표에서의 지하철역과의 거리, 소속동의 인구수를 구하는 식을 사용함. 이후 해당 피처들을 사용하여 확률을 구하는 가설함수를 만들고, 교차엔트로피를 통해 좋은 모델인지 확인함.
좋은 모델이 만들어졌다면, 해당 로지스틱회귀에 랜덤 서울 좌표값들을 넣고 자동으로 기입되는 지하철역과의 거리, 소속동의 인구수를 통해 '적합' 혹은 '부적합'으로 분류하는 확률을 얻어내는 방법을 사용.

4. 느낀점

이번 팀 프로젝트를 진행하며 데이터분석을 위해 많은 과정이 필요한 것을 알게 되었습니다. 통계자료, 기사자료 조사부터 시작하여 데이터수집, 데이터전처리, 분석 및 적절한 그래프 선택, 그래프 및 지도 시각화, 머신러닝 모델 선택 및 구현, 발표자료 제작 및 발표와 같이 많은 과정을 수행해야 했었고, 혼자서는 꽤 버거운 업무였을 것입니다.

초기에는 팀원들이 함께 모든 과정에 참여하려 했으나, 오히려 한 과정에 집중하지 못하고 되려 비효율적인 상황이 나타났습니다. 의견이 많아질수록 이 의견들을 전부 수용하기에는 한계가 있었고, 여러 개의 방안 중 하나를 택해야 하는 상황이 생겼습니다. 사소한 것 하나하나 선택이 필요한 상황이 계속되니 오히려 실제 코드를 작성하는 시간보다 회의에 쏟는 시간이 지나치게 많이 필요했고, 이런 비효율을 개선하고자 **역할을 나누어 각자 임무에 집중**하고자 하였습니다. 그리고 실제로 **이전보다 더 매끄럽게 프로젝트를 진행**할 수 있었습니다.

저는 이번 프로젝트에서 **데이터 수집 및 전처리**, **그래프 시각화 및 머신러닝 활용**, 코드 정리를 주로 맡았습니다. 데이터 전처리가 기반 되어야 이후 지도, 그래프, 머신러닝에 사용할 수 있었기에 이 과정이 매우 중요했습니다.

강의 시간에 배운 것들을 활용하며 원시데이터를 활용가능한 데이터로 변환하고, 이와 동시에 다른 팀원들은 기사, 통계와 같은 자료들을 조사했습니다. 이후 전처리한 데이터를 토대로 그래프 시각화를 진행했습니다. 팀 회의를 통해 데이터 분석 발표를 위해 필요한 자료를 함께 논의하며 체크리스트에 적었고, 저는 그래프 시각화를, 박원석 팀원은 지도 시각화를, 최민석 팀원은 시각화 한 자료들을 토대로 분석 및 발표자료를 제작했습니다. 그리고 각자 의논할 것이 생기면 해당 역할을 맡은 팀원에게 직접 질문하여 좀 더 효율적인 체계를 구성했습니다.

이번 프로젝트에서 아쉬운 점이 있다면, 우리가 분석한 내용들을 **실제로** 국가에 적용하기 위해 제안하려면 범위를 '서울시'가 아닌 '수도권'으로 잡았어야 했다고 생각합니다. 하지만 경기도까지 데이터를 탐색했을 때, 최근 정보이면서 균일한 내용을 담고 있는 데이터를 찾을 수 없었기에 어쩔 수 없이 범위를 '서울시'로 제한했지만, 공장단지가 많은 경기도를 포함시킨다면 결과가 사뭇 다르게 나오지 않았을까 생각했습니다.

또한 머신러닝 구현에서도 아쉬움이 있었습니다. 팀 발표 전까지는 머신러닝을 구현하여 적용하는데 중점을 두어서 그 과정과 내용을 완벽히 이해하지 못했지만, 발표 이후 제가 만든 모델의 성능 향상을 위해 K-Means Clustering을 포함한 sklearn 라이브러리를 활용하는 법을 찾아보았고, 이후 성능이 더욱 개선된 머신러닝 결과를 얻어낼 수 있었습니다. 하지만 팀 발표에 이러한 과정 및 분석 내용을 담지 못하여 아쉬움만 남았습니다.

저는 데이터분석 분야뿐만 아니라 모든 팀 프로젝트에서 '팀원과의 의사소통'이 얼마나 중요한지 깨닫고 있습니다. 단지 구현을 잘 하는 것이 최고가 아니라, 팀원과의 의사소통을 통해 함께 나아가는 능력이 있는 인재가 되어야 할 것입니다. 또한 이번 프로젝트를 통해 데이터 활용 기술에 대한 이해를 얻을 수 있었고, 이번 경험을 통해 차후 데이터를 활용하는 공모전이 있다면 참여해 볼 용기를 얻을 수 있었습니다.