서울시 (5대)범죄발생 데이터를 이용한 범죄 분석 모델링

안준호, 조종현

목차

01

연구 및 분석배경

1) 주제 선정 이유 및 목표

서울, 강력 범죄 어디서 가장 많이 발생할까?

범죄 많은데 주민은 느긋... 강남·서초의 '역설'

[한눈에 보는 전국 범죄 지도③] 서울 25개 자치구 범죄 통계

서울 '범죄지도' 첫 공개…강서·구로 '살인·폭력', 강남·서초 '강도·마약' 많다

최초 공개된 서울시 '범죄지도'의 충격적인 결과

서울시 지역구에 따른 범죄 발생건의 차이를 뉴스/매체 등을 통해 접할 수 있음

그렇다면, **지역구별 범죄 발생 수의 차이를 발생시키는 요인**은 어떠한 것이 있을까?

01

연구 및 분석배경

1) 주제 선정 이유 및 목표

기준: 서울시 자치구별 2016~2020년 통계

구분	변수명
종속변수	범죄 발생 수
독립변수	1인당 지방세 납부액
	65세 이상 인구 비율
	공공기관 신뢰도
	기초생활 수급자 비율
	면적 대비 방범용 CCTV 수
	면적 대비 범죄 대응시설 수
	성비
	외국인 비율
	인구 밀도

◆ 데이터 출처

〈종속변수〉
- 범죄 발생건수 -> *서울 열린데이터 광장*

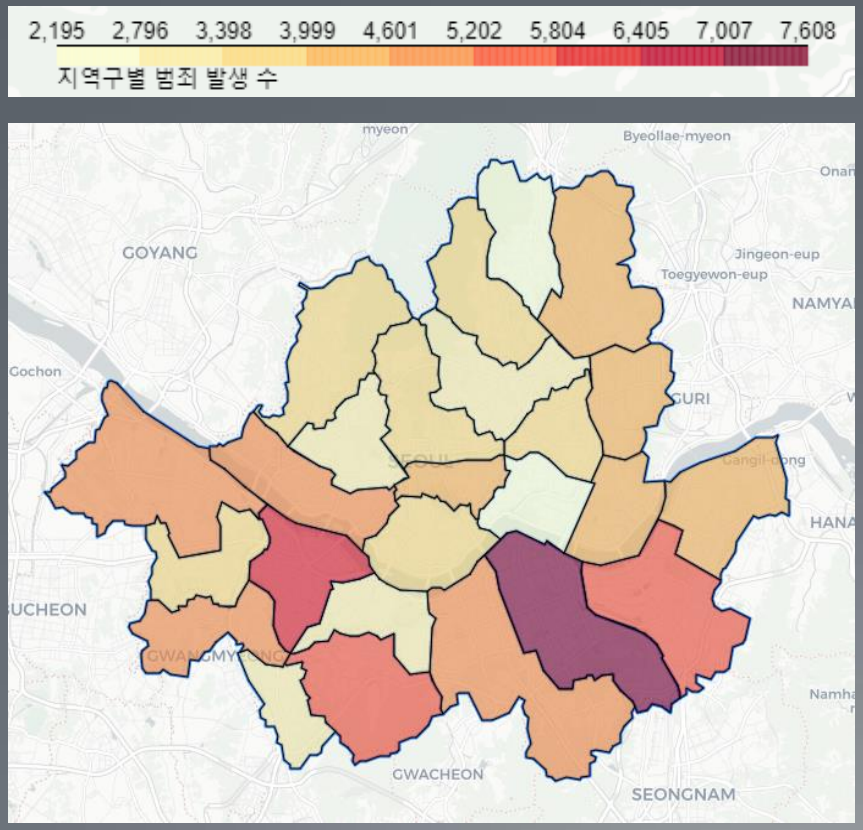
〈독립변수〉
- 범죄대응시설 수, 지방세, 인구밀도, 외국인 비율, 기초생활 수급자, CCTV 수 - > *서울 열린데이터 광장*
- 성비, 고령 인구 비율, 공공기관 신뢰도 -> *kosis*

➢ 상기 독립변수 中 유의한 변수를 추출하고 범죄 발생수를 예측할 수 있는 모델 생성

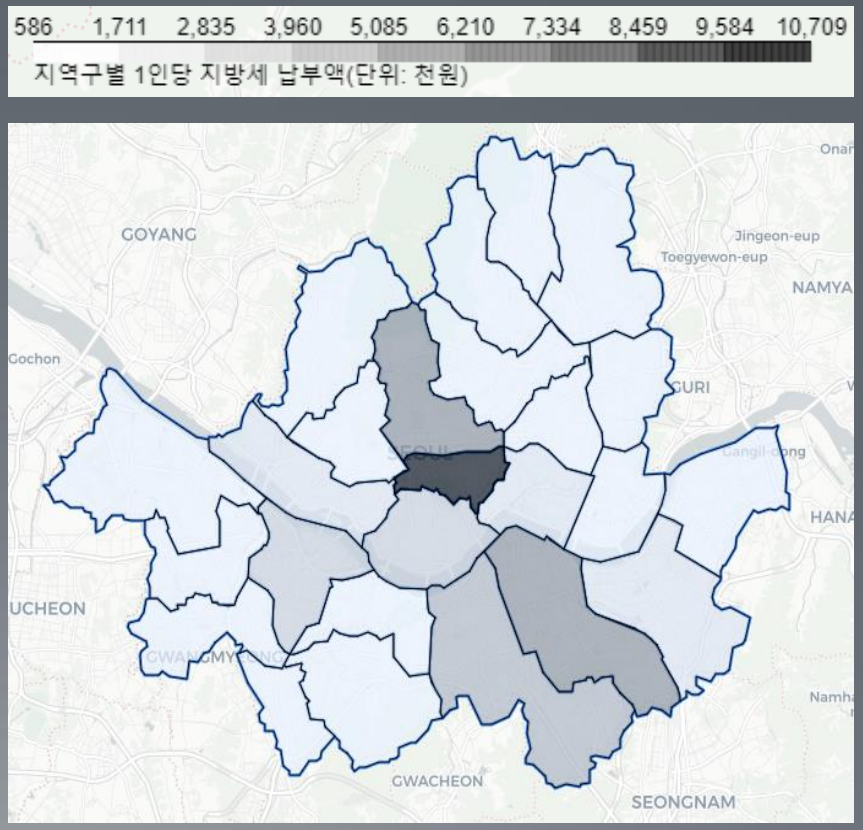
01

연구 및 분석배경

2) Folium 패키지를 이용한 자치구별 변수 시각화



- 강남구, 영등포구, 송파구 순
- 강북 지역에 비해 강남 지역의 범죄 발생수가 높음

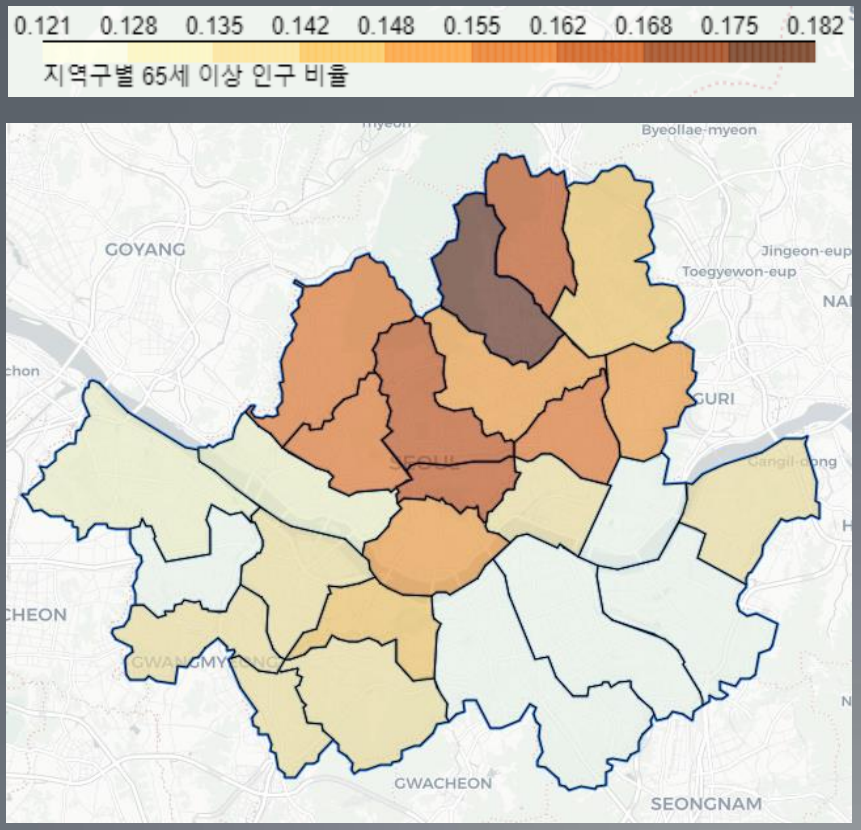


- 중구, 강남구, 종로구 순
- 중구가 매우 높으며 타 지역은 비교적 고른 분포

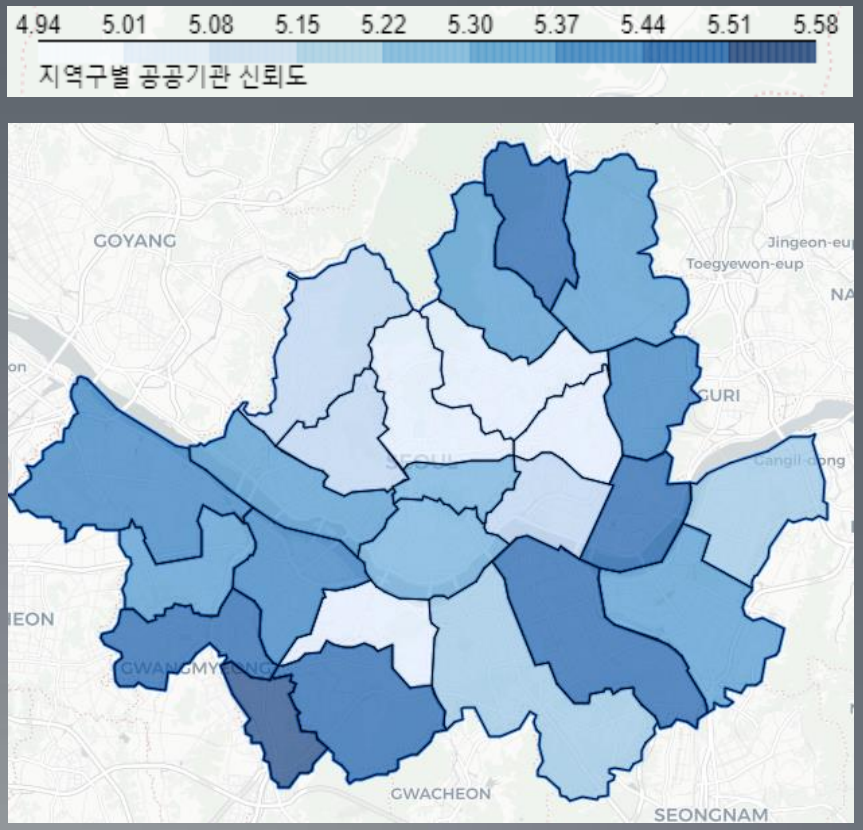
01

연구 및 분석배경

2) Folium 패키지를 이용한 자치구별 변수 시각화



- 강북구, 도봉구, 종로구 순
- 강북지역이 높은 것이 두드러짐

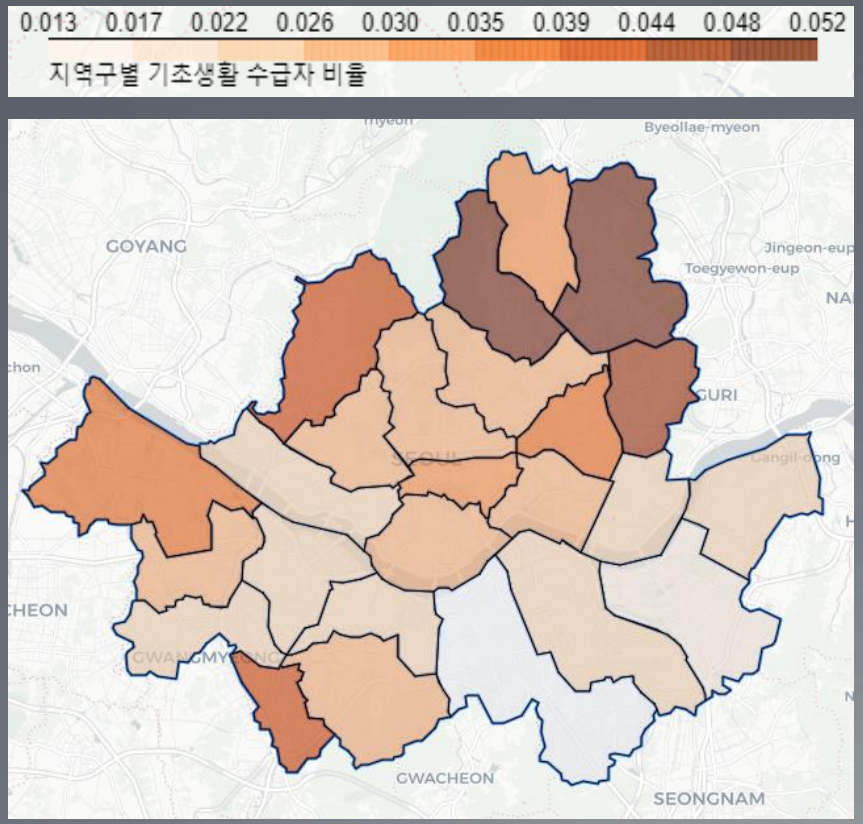


- 금천구, 광진구, 관악구 순
- 대체적으로 고른 분포

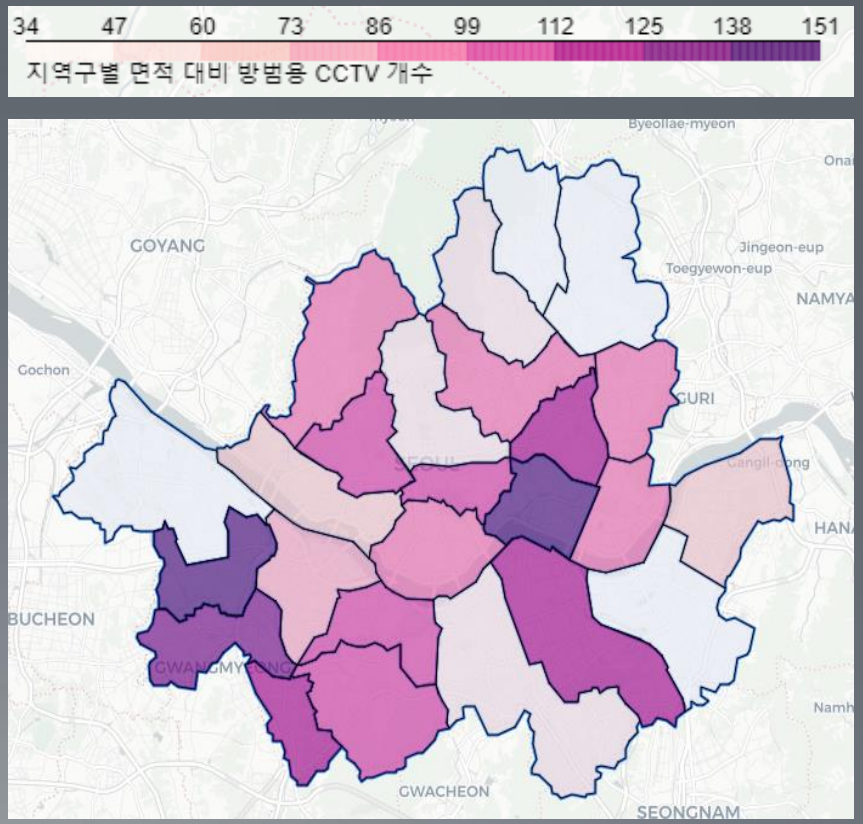
01

연구 및 분석배경

2) Folium 패키지를 이용한 자치구별 변수 시각화



- 강북구, 노원구, 중랑구 순
- 강북지역이 강남 지역에 비해 높음

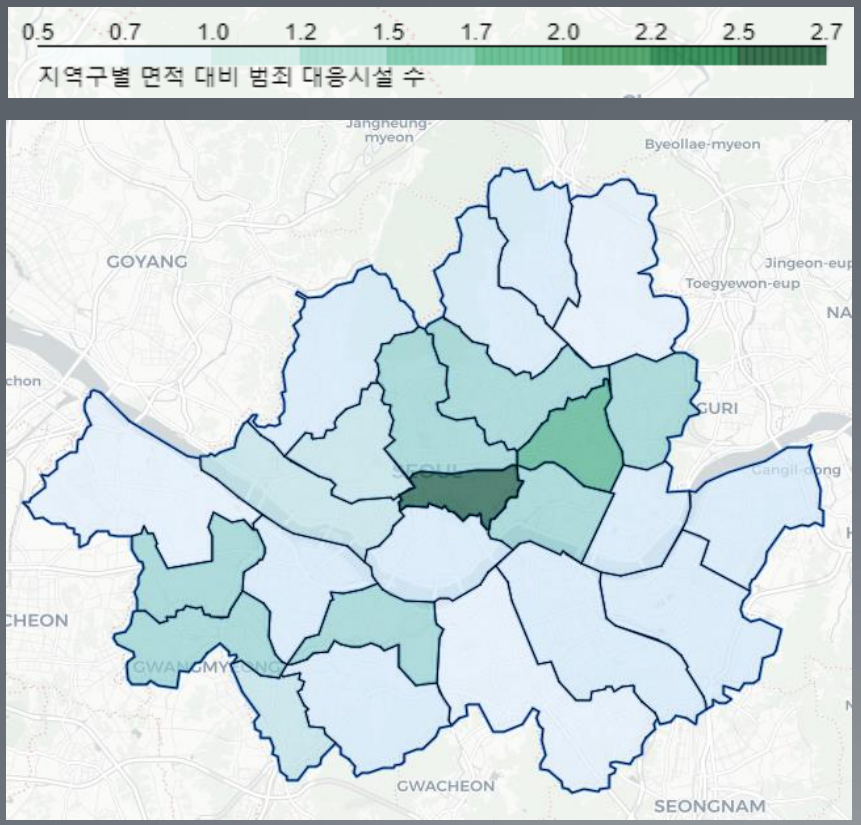


- 양천구, 성동구, 구로구 순
- 중부 지역과 강남 서부지역이 높은 편

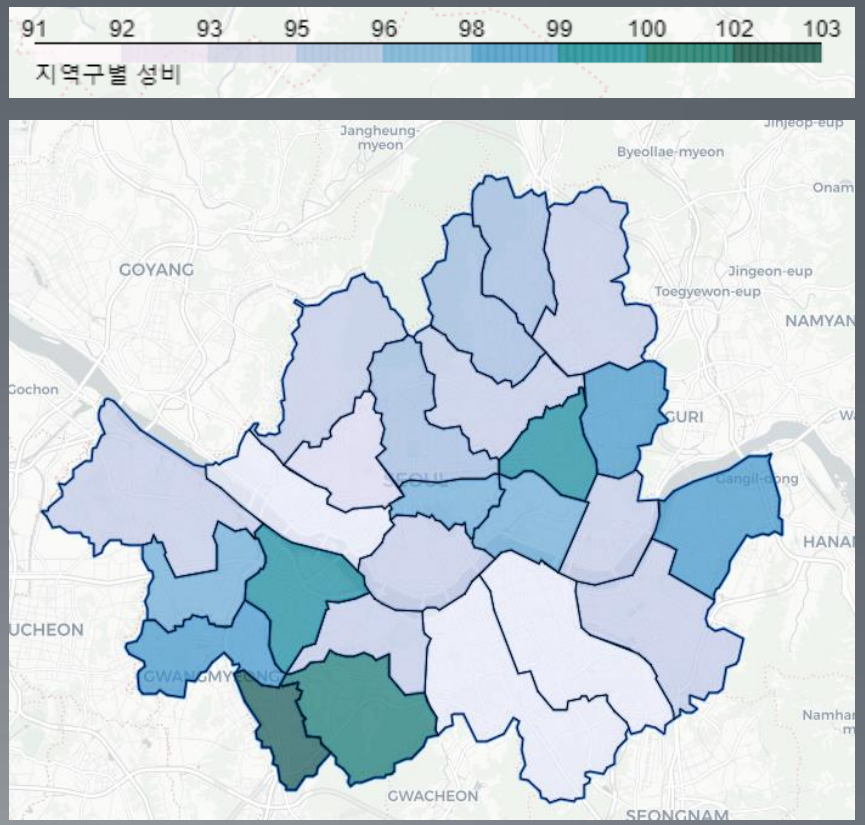
01

연구 및 분석배경

2) Folium 패키지를 이용한 자치구별 변수 시각화



- 중구, 동대문구, 동작구 순
- 대체적으로 고르나 중부 지역이 다소 높음

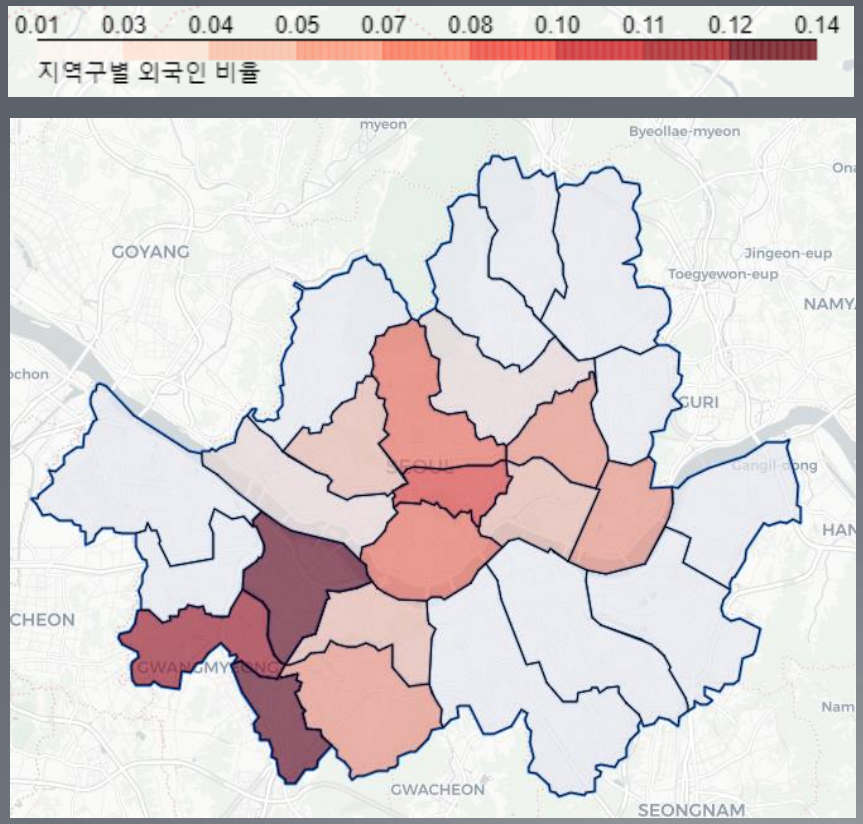


- 금천구, 관악구, 동대문구 순
- 강남 서부지역이 대체적으로 높음

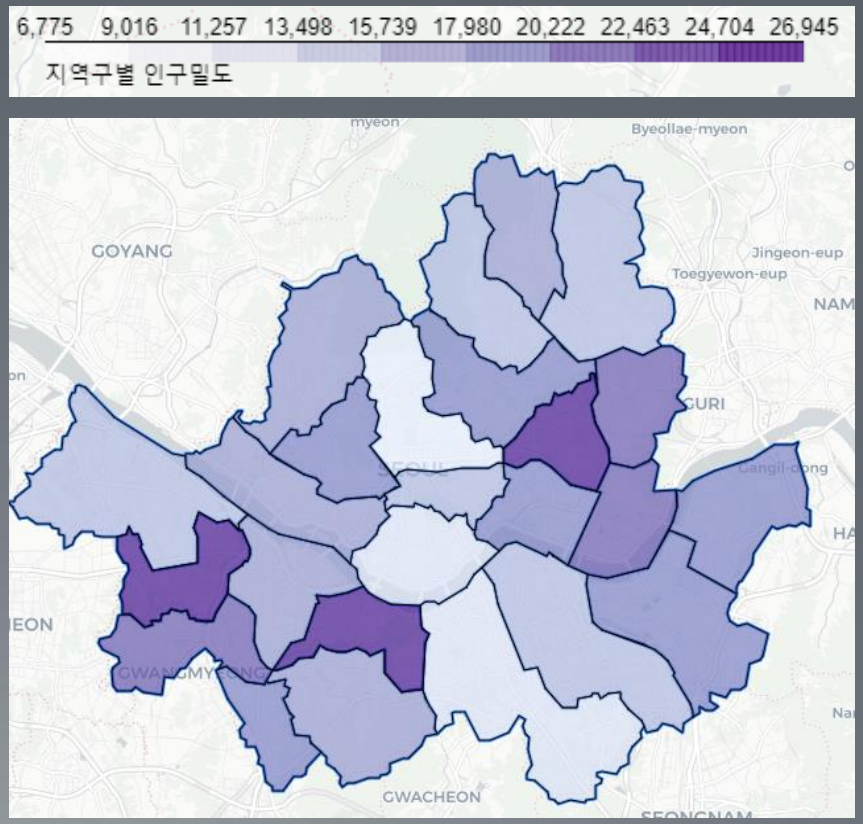
01

연구 및 분석배경

2) Folium 패키지를 이용한 자치구별 변수 시각화



- 영등포구, 금천구, 구로구 순
- 강남 서부지역 및 중부지역에서 높게 나타남



- 양천구, 동대문구, 동작구 순
- 비교적 고른 분포

02

데이터 전처리 및 분석

1) 데이터 전처리

- 총 변수 10개 데이터셋 -> 하나의 Crime 데이터셋으로 통합, 총 125행

- 결측치(N/A) : 도봉구 : 살인 (1개) -> 최근 5년 도봉구 살인의 평균값으로 처리

연도 및 자치구	범죄 발생 합계	면적 대비 범죄 대응시설 수	1인당 평균 지방세 납부액	인구밀도	외국인 비율	성비	65세 이상 인구 비율	공공기관 신뢰도	기초생활 수급자 비율	면적 대비 방범용 CCTV 개수
2016_종로구	4459	1.380176	4369526.432	6771	0.069941	97	0.154087	4.9	0.026241	44.583856
2016_중구	4584	2.811245	9345562.633	13494	0.084161	98.7	0.151686	4.89	0.029552	56.726908
2016_용산구	4137	0.86877	2443665.804	11209	0.069685	94.1	0.145029	4.92	0.02439	77.229081
2016_성동구	3026	1.18624	1291124.033	18218	0.039559	98.3	0.127425	4.73	0.023509	78.766311
2016_광진구	5322	0.87925	859984.6172	21807	0.054592	95.8	0.110969	5.12	0.020392	38.511137
2016_동대문구	4787	1.969058	900147.6323	26050	0.04999	100.1	0.143935	4.79	0.031284	100.21097
2016_중랑구	5041	1.297297	637381.0699	22474	0.019441	99.4	0.13434	5.05	0.036552	48.540541
2016_성북구	3744	1.261188	717683.6187	18781	0.030863	95.5	0.138379	5.06	0.027919	62.408462
2016_강북구	4229	0.889831	532842.1761	14015	0.018739	96.7	0.163179	5.04	0.046341	35.59322
2016_도봉구	2438	0.774069	531197.1325	16948	0.011377	96.8	0.144077	5.24	0.029494	21.432027
2016_노원구	4543	0.733634	547341.9869	16119	0.01129	94.9	0.12398	5.38	0.043572	35.778781
2016_은평구	4501	0.909091	602248.8986	16696	0.016262	94.8	0.143776	5.67	0.035577	69.86532
2016_서대문구	3665	1.022147	969875.6318	18506	0.041231	94.3	0.145159	5.17	0.023209	66.950596
2016_마포구	5873	1.048658	1573138.291	16394	0.034179	92.1	0.122951	5.26	0.019742	40.687919
2016_양천구	4112	1.378518	877579.2859	27681	0.017055	97.9	0.107829	5.13	0.022123	116.369902
2016_강서구	5450	0.482625	967188.9873	14531	0.019867	95.2	0.117735	5.18	0.034996	23.166023
2016_구로구	5366	1.341948	873479.3439	22347	0.107382	98.9	0.122685	5.13	0.019262	85.437376
2016_금천구	3645	1.152074	1275389.536	19560	0.116472	103.8	0.125566	5.4	0.037482	83.717358
2016_영등포구	6322	0.896861	2511507.101	16583	0.136258	99.9	0.125832	5.17	0.019775	46.473706
2016_동작구	3227	1.345566	862957.1999	25269	0.043637	95.9	0.133218	4.95	0.019337	73.394495
2016_관악구	5678	0.87927	614249.3993	17776	0.055688	102.2	0.126823	4.95	0.024808	69.259385
2016_서초구	4989	0.595998	3451109.563	9610	0.018366	92.3	0.112754	4.69	0.010749	37.994891
2016_강남구	8149	0.810127	4668451.958	14484	0.017062	91.9	0.108054	4.97	0.020156	87.265823
2016_송파구	6090	0.855962	1792257.161	19629	0.018892	95	0.106652	4.92	0.014257	23.553719
2016_강동구	4462	0.976007	1037254.008	18238	0.017968	99.1	0.118449	4.82	0.022307	35.095567
2017_종로구	4057	1.380176	4778723.464	6869	0.072496	96.4	0.158471	5.49	0.026154	51.233793
2017_중구	4184	2.710843	9392201.786	13514	0.084566	98.1	0.157126	5.29	0.029467	84.136546
2017_용산구	4060	0.86877	3054591.154	11179	0.072119	93.7	0.149269	5.05	0.0243	83.721994
2017_성동구	2767	1.245552	1609216.491	18551	0.040306	97.4	0.131598	5.24	0.022394	124.733096
2017_광진구	4646	0.937866	979960.1448	21819	0.056987	95.2	0.117621	5.13	0.019906	65.181712
2017_동대문구	3975	1.969058	952720.8554	25748	0.055665	100	0.151763	4.76	0.030873	107.946554
2017_중랑구	4571	1.297297	612209.9956	22318	0.020532	99.2	0.143391	5.25	0.036891	56.594595
2017_성북구	3434	1.261701	763577.8831	18533	0.03368	95.2	0.145191	4.82	0.027303	78.958079
2017_강북구	3393	0.889831	559199.8128	13898	0.019777	96	0.172063	5.45	0.045588	35.635593
2017_도봉구	1999	0.774069	559309.5132	16752	0.011972	96.2	0.154361	5.22	0.030687	26.221577

02

데이터 전처리 및 분석

2) 상관계수 확인 및 시각화

```
# 종속변수, 독립변수 추출
crime = crime.iloc[:,[1,9,18,19,21,22,24,25,27,29]] # 종속변수 1개, 독립변수 9개
crime.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125 entries, 0 to 124
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   범죄 발생 합계           125 non-null    float64
 1   면적 대비 범죄 대응시설 수    125 non-null    float64
 2   1인당 평균 지방세 납부액     125 non-null    float64
 3   인구밀도               125 non-null    float64
 4   외국인 비율             125 non-null    float64
 5   성비                 125 non-null    float64
 6   65세 이상 인구 비율       125 non-null    float64
 7   공공기관 신뢰도           125 non-null    float64
 8   기초생활 수급자 비율        125 non-null    float64
 9   면적 대비 방범용 CCTV 개수  125 non-null    float64
dtypes: float64(10)
memory usage: 9.9 KB
```

```
# 2) 상관계수 확인
COR = crime.corr()

# 종속변수(범죄 발생 합계)와 독립변수간의 상관계수 확인
COR['범죄 발생 합계']
```

```
범죄 발생 합계              1.000000
면적 대비 범죄 대응시설 수     -0.211211
1인당 평균 지방세 납부액        0.257704
인구밀도                -0.146173
외국인 비율               0.012070
성비                  -0.088311
65세 이상 인구 비율         -0.569922
공공기관 신뢰도             0.104935
기초생활 수급자 비율          -0.386618
면적 대비 방범용 CCTV 개수   -0.189366
Name: 범죄 발생 합계, dtype: float64
```

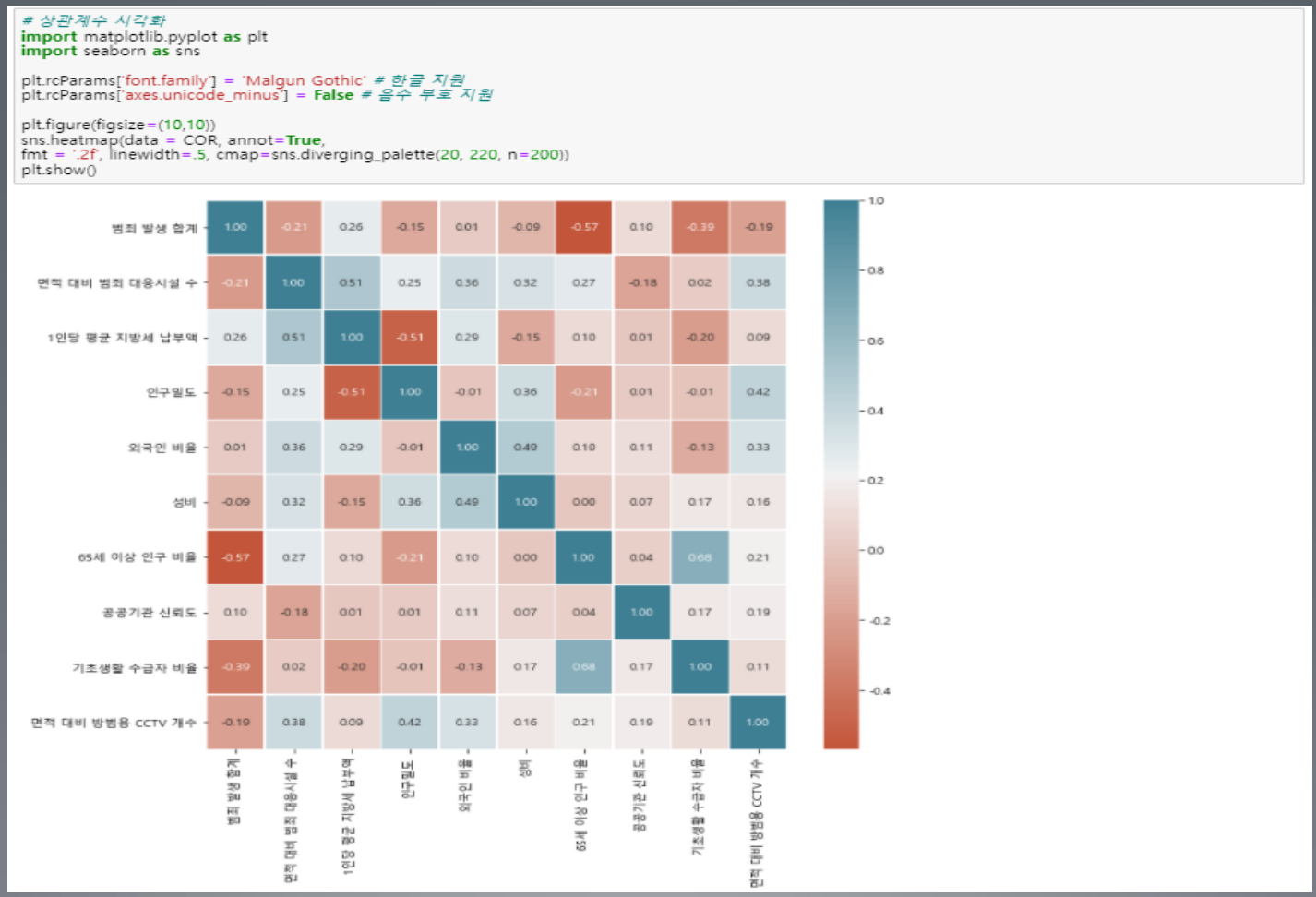
➢ 상관계수의 절댓값이 0.3 이상인 변수 -> 65세 이상 인구 비율(-0.57), 기초생활 수급자 비율(-0.38)

02

데이터 전처리 및 분석

2) 상관계수 확인 및 시각화

```
# 상관계수 시각화
import matplotlib.pyplot as plt
import seaborn as sns

plt.rcParams['font.family'] = 'Malgun Gothic' # 한글 지원
plt.rcParams['axes.unicode_minus'] = False # 음수 부호 지원

plt.figure(figsize=(10,10))
sns.heatmap(data = COR, annot=True,
fmt = '.2f', linewidth=.5, cmap=sns.diverging_palette(20, 220, n=200))
plt.show()
```



➢ 상관계수 행렬 시각화: 붉은색에 가까울수록 부(-), 푸른색에 가까울수록 정(+)의 영향

데이터 전처리 및 분석

3) 회귀분석 - LinearRegression

```
# 3) 회귀분석

# 변수 칼럼명 수정 (한글 -> 영어)
crime.columns = ['total_crime', 'institution', 'local_tax', 'pop_density',
                 'foreigner_ratio','gender_ratio', 'over65_ratio',
                 'pub_reliability', 'basic_life_ratio', 'cctv']
crime.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125 entries, 0 to 124
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   total_crime      125 non-null    float64
 1   institution      125 non-null    float64
 2   local_tax        125 non-null    float64
 3   pop_density      125 non-null    float64
 4   foreigner_ratio  125 non-null    float64
 5   gender_ratio     125 non-null    float64
 6   over65_ratio     125 non-null    float64
 7   pub_reliability  125 non-null    float64
 8   basic_life_ratio 125 non-null    float64
 9   cctv             125 non-null    float64
dtypes: float64(10)
memory usage: 9.9 KB
```

```
# X, y 변수 정의
X = crime.iloc[:,1:]
y = crime.iloc[:,0]
```

➢ column명 영문으로 수정 및 X, y 변수 정의

02

데이터 전처리 및 분석

3) 회귀분석 - LinearRegression

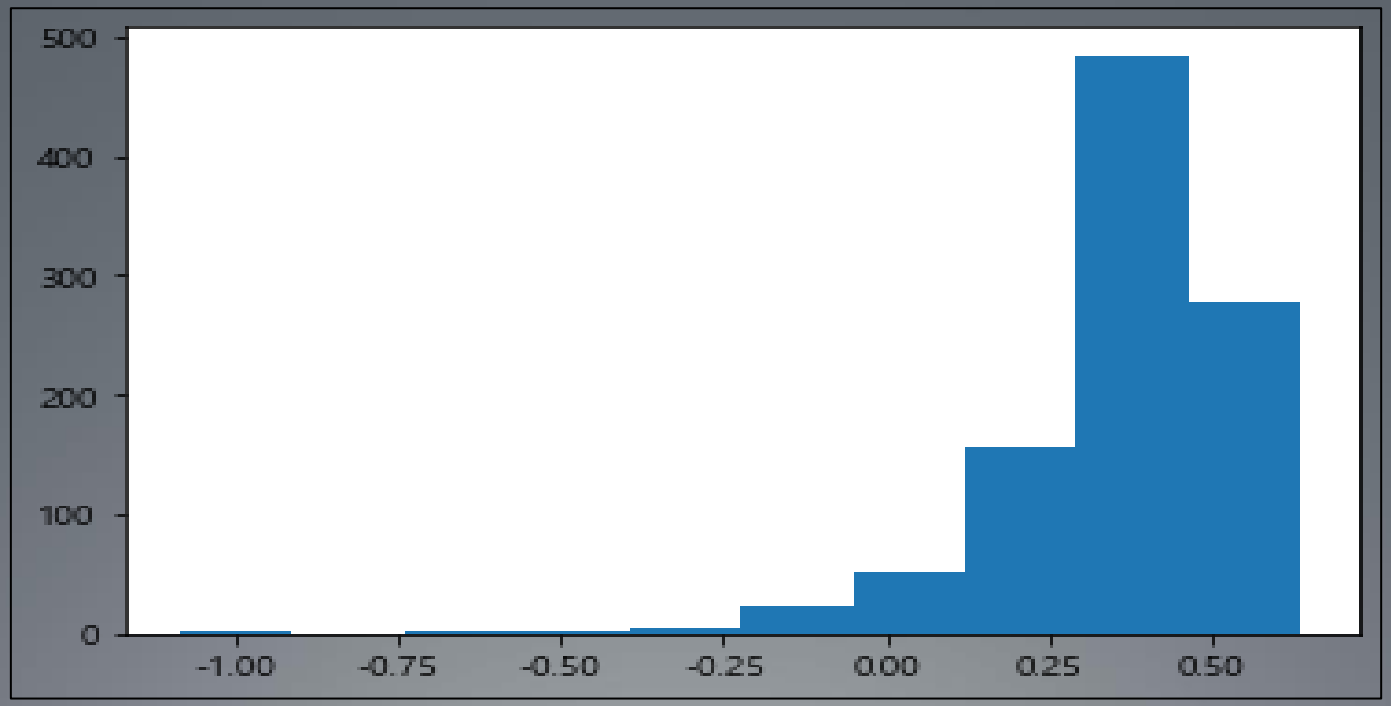
```
# 3-1) Linear Regression(Holdout Method)
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import statistics as st

# random_state 값을 지정하지 않고 train/test split 및 모델 생성 1000회 반복
lr_scores = []
for i in range(1000):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                        test_size = 0.3)
    lr = LinearRegression()
    lr.fit(X_train, y_train) # 모형 학습

    # 학습 마친 모형에 test data 적용 -> 결정계수(R^2) 계산
    r_square = lr.score(X_test, y_test)
    lr_scores.append(r_square)
```

➢ train/test split 및 LinearRegression 모델 생성 1000회 반복 및 model score 저장

데이터 전처리 및 분석

3) 회귀분석 - LinearRegression

```
# score의 분포 확인
plt.hist(lr_scores)
plt.show() # score가 음수인 이상치 확인됨 -> 중앙값으로 score 대표값 채택
```



- Score 값이 음수인 경우가 확인되어 이상치로 판단
- 이상치의 영향을 최소화할 수 있는 중앙값으로 model score 결정: **설명력 약 41%**
- **모델의 설명력을 보완할 수 있는 다른 회귀 알고리즘 적용 검토**

데이터 전처리 및 분석

3) 회귀분석 - OLS(Ordinary Least Squares) Method

```
# 3-2) OLS 회귀분석
from statsmodels.formula.api import ols

model = ols('total_crime ~ institution + local_tax + pop_density + foreigner_ratio + gender_ratio + over65_ratio + pub_reliability + basic_life_ratio + cctv',
        data = crime)
lm = model.fit()
lm.summary()
```

Dep. Variable:	total_crime	R-squared:	0.521
Model:	OLS	Adj. R-squared:	0.484
Method:	Least Squares	F-statistic:	13.91
Date:	Sat, 29 Jan 2022	Prob (F-statistic):	6.16e-15
Time:	23:59:28	Log-Likelihood:	-1018.0
No. Observations:	125	AIC:	2056.
Df Residuals:	115	BIC:	2084.
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	6487.4074	3917.153	1.656	0.100	-1271.718	1.42e+04
institution	-1369.5760	394.891	-3.468	0.001	-2151.779	-587.373
local_tax	0.0004	8.18e-05	4.760	0.000	0.000	0.001
pop_density	0.0733	0.039	1.902	0.060	-0.003	0.150
foreigner_ratio	2578.4681	3096.934	0.833	0.407	-3555.963	8712.899
gender_ratio	15.1137	37.838	0.399	0.690	-59.837	90.064
over65_ratio	-3.381e+04	7565.259	-4.469	0.000	-4.88e+04	-1.88e+04
pub_reliability	38.2707	254.996	0.150	0.881	-466.827	543.369
basic_life_ratio	1.757e+04	1.22e+04	1.436	0.154	-6673.177	4.18e+04
cctv	-3.7564	2.685	-1.399	0.164	-9.075	1.562

Omnibus:	7.900	Durbin-Watson:	2.335
Prob(Omnibus):	0.019	Jarque-Bera (JB):	3.405
Skew:	0.051	Prob(JB):	0.182
Kurtosis:	2.198	Cond. No.	5.46e+08

- OLS 알고리즘 적용 시 모델의 **수정 결정계수는 약 48.4%**
- 면적 대비 범죄 대응시설 수, 1인당 지방세 납부금액, 65세 이상 인구 비율 등 총 3개의 변수는 통계적으로 유의함

02

데이터 전처리 및 분석

3) 회귀분석 - OLS(Ordinary Least Squares) Method

```
# 다중공선성 확인
from statsmodels.stats.outliers_influence import variance_inflation_factor

# 각 독립변수의 분산팽창계수(VIF) 확인
pd.DataFrame({'독립변수': column, 'VIF': variance_inflation_factor(model.exog, i)}
              for i, column in enumerate(model.exog_names)
              if column != 'Intercept')  # 절편의 VIF는 구할 필요 없음

# VIF가 모두 10 이하이므로 다중공선성 문제는 없다고 판단
```

	독립변수	VIF
0	X[0]	5.229355
1	X[1]	5.821614
2	X[2]	5.533790
3	X[3]	2.110027
4	X[4]	2.209986
5	X[5]	3.353805
6	X[6]	1.321515
7	X[7]	2.751382
8	X[8]	1.998787

- VIF가 10을 초과하는 독립변수가 없으므로 다중공선성 문제는 없는 것으로 판단

데이터 전처리 및 분석

3) 회귀분석 - OLS(Ordinary Least Squares) Method

```
# 변수선택법(단계별 선택법) -> 모델링에 필요한 최적의 변수 갯수 추출
import statsmodels.api as sm

variables = crime.columns[1:].tolist() # 독립 변수 리스트
y = crime['total_crime'] # 종속 변수

selected_variables = [] # 선택된 변수들
sl_enter = 0.05
sl_remove = 0.05

sv_per_step = [] # 각 스텝별로 선택된 변수들
adjusted_r_squared = [] # 각 스텝별 수정 결정계수
steps = [] # 스텝
step = 0
while len(variables) > 0:
    remainder = list(set(variables) - set(selected_variables))
    pval = pd.Series(index=remainder) ## 변수의 p-value
    # 기존에 포함된 변수와 새로운 변수 하나씩 돌아가면서 선형 모형을 적합
    for col in remainder:
        variable = crime[selected_variables+[col]]
        variable = sm.add_constant(variable)
        model = sm.OLS(y,variable).fit()
        pval[col] = model.pvalues[col]

    min_pval = pval.min()
    if min_pval < sl_enter: # 최소 p-value 값이 기준 값보다 작으면 포함
        selected_variables.append(pval.idxmin())
        # 선택된 변수들에 대해서 어떤 변수를 제거할지 선택
        while len(selected_variables) > 0:
            selected_X = crime[selected_variables]
            selected_X = sm.add_constant(selected_X)
            selected_pval = sm.OLS(y,selected_X).fit().pvalues[1:] ## 절편항의 p-value는 뺀다
            max_pval = selected_pval.max()
            if max_pval >= sl_remove: ## 최대 p-value값이 기준값보다 크거나 같으면 제외
                remove_variable = selected_pval.idxmax()
                selected_variables.remove(remove_variable)
            else:
                break

        step += 1
        steps.append(step)
        adj_r_squared = sm.OLS(y,sm.add_constant(crime[selected_variables])).fit().rsquared_adj
        adjusted_r_squared.append(adj_r_squared)
        sv_per_step.append(selected_variables.copy())
    else:
        break
```

```
print(selected_variables) # OLS 회귀모델에서 유의하게 나온 변수 3개와 동일
```

```
['over65_ratio', 'local_tax', 'institution']
```

- 변수선택법을 통해 최적의 변수 조합 탐색 결과 면적 대비 범죄 대응시설 수, 1인당 지방세 납부금액, 65세 이상 인구 비율이 가장 최적의 조합으로 선정됨, 이는 OLS 회귀모델 상에서 유의한 독립변수 3개와 같음

02

데이터 전처리 및 분석

3) 회귀분석 - OLS(Ordinary Least Squares) Method

```
# 변수선택법에서 구한 독립변수 3개만 포함하여 모델 재생성
new_model = ols('total_crime ~ institution + local_tax + over65_ratio', data=crime)
lm2 = new_model.fit()
lm2.summary()
```

Dep. Variable:	total_crime	R-squared:	0.487
Model:	OLS	Adj. R-squared:	0.474
Method:	Least Squares	F-statistic:	38.28
Date:	Sun, 30 Jan 2022	Prob (F-statistic):	1.82e-17
Time:	00:02:58	Log-Likelihood:	-1022.3
No. Observations:	125	AIC:	2053.
Df Residuals:	121	BIC:	2064.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	9482.3812	607.068	15.620	0.000	8280.529	1.07e+04
institution	-811.9034	209.800	-3.870	0.000	-1227.257	-396.550
local_tax	0.0002	3.98e-05	6,116	0.000	0.000	0.000
over65_ratio	-3.408e+04	4337.191	-7.858	0.000	-4.27e+04	-2.55e+04

Omnibus:	11.290	Durbin-Watson:	2.324
Prob(Omnibus):	0.004	Jarque-Bera (JB):	4.118
Skew:	0.054	Prob(JB):	0.128
Kurtosis:	2.117	Cond. No.	1.74e+08

- 유의한 독립변수 3개만 포함하여 OLS 회귀분석 진행
- 수정 결정계수는 약 47.4%로 모든 독립변수를 포함했을 때보다 1% 정도 감소하였으며 이는 독립변수 개수의 감소로 인한 것으로 판단됨
- 설명력 보완을 위한 추가적인 회귀모델 검토 필요

데이터 전처리 및 분석

3) 회귀분석 - Lazy Predict

```
# 3-3) Lazy predict 적용
'''
<LazyPredict 알고리즘>
-> 수 많은 회귀 모델의 결과를 한 번에 도출해주는 모듈(단, Tuning이 없으며 Basic parameter를 기준으로 함)
-> 분석하고자 하는 데이터가 어떠한 모델을 이용했을 때 가장 좋은 결과를 얻을 수 있는지 확인하고자 할 때 사용
'''

from lazypredict.Supervised import LazyRegressor

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3)

reg = LazyRegressor(verbose=0, ignore_warnings=False,
                    custom_metric=None)
models_regression, predictions_regression = reg.fit(
        X_train, X_test, y_train, y_test)

models_regression
```

Model	Adjusted R-Squared	R-Squared	RMSE	Time Taken
ExtraTreesRegressor	0.68	0.76	651.32	0.11
XGBRegressor	0.68	0.76	651.62	0.06
KNeighborsRegressor	0.52	0.64	797.45	0.02
RandomForestRegressor	0.51	0.63	807.95	0.19
BaggingRegressor	0.49	0.62	821.59	0.04
GradientBoostingRegressor	0.42	0.56	877.10	0.04
PoissonRegressor	0.40	0.54	896.33	0.00
AdaBoostRegressor	0.36	0.52	921.91	0.12
LGBMRegressor	0.35	0.51	932.83	0.04
TransformedTargetRegressor	0.33	0.49	947.57	0.01
LinearRegression	0.33	0.49	947.57	0.01
Lasso	0.32	0.49	948.52	0.03
SGDRegressor	0.32	0.49	951.67	0.02
Ridge	0.32	0.48	955.04	0.01
RidgeCV	0.32	0.48	955.04	0.01
LassoLars	0.31	0.48	959.93	0.02
OrthogonalMatchingPursuitCV	0.30	0.47	965.63	0.02
HistGradientBoostingRegressor	0.28	0.45	979.84	0.14
LassoCV	0.26	0.44	990.15	0.07
LassoLarsIC	0.26	0.44	995.67	0.02
LassoLarsCV	0.25	0.43	997.21	0.04
BayesianRidge	0.25	0.43	1002.02	0.01
Lars	0.23	0.42	1012.44	0.02
HuberRegressor	0.23	0.42	1013.95	0.04
LarsCV	0.23	0.42	1014.19	0.03
ElasticNet	0.11	0.33	1086.44	0.02
PassiveAggressiveRegressor	0.11	0.32	1091.69	0.01
ElasticNetCV	0.08	0.31	1105.20	0.06
OrthogonalMatchingPursuit	0.07	0.30	1113.11	0.01
GammaRegressor	0.05	0.28	1122.42	0.02
TweedieRegressor	0.03	0.27	1134.98	0.01
GeneralizedLinearRegressor	0.03	0.27	1134.98	0.01
DecisionTreeRegressor	0.03	0.26	1137.65	0.01
RANSACRegressor	-0.17	0.12	1246.54	0.06
SVR	-0.32	-0.00	1326.82	0.02
NuSVR	-0.33	-0.00	1329.45	0.03
DummyRegressor	-0.33	-0.01	1331.67	0.00
ExtraTreeRegressor	-0.41	-0.06	1368.34	0.03
GaussianProcessRegressor	-1.17	-0.64	1699.33	0.03
LinearSVR	-12.48	-9.20	4236.67	0.01
KernelRidge	-12.54	-9.24	4245.80	0.01
MLPRegressor	-12.92	-9.53	4305.66	0.17

Lazy Predict란?

- 수 많은 회귀 모델의 결과를 한 번에 도출해주는 모듈(단, Tuning이 없으며 Basic parameter를 기준으로 함)
- **분석하고자 하는 데이터가 어떠한 모델을 이용했을 때 가장 좋은 결과를 얻을 수 있는지 확인하고자 할 때 사용**

02

데이터 전처리 및 분석

3) 회귀분석 - Lazy Predict

```
# train/test split + LazyRegressor 100회 반복
reg_list = [] # 설명력이 가장 높은 알고리즘 결과 저장
for i in range(100):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                        test_size=0.3)

    reg = LazyRegressor(verbose=0, ignore_warnings=False,
                        custom_metric=None)
    models_regression, predictions_regression = reg.fit(
        X_train, X_test, y_train, y_test)

    reg_list.append(models_regression.index[0])
    i += 1
```

```
# 설명력이 가장 높은 회귀모델의 빈도 count
wc = {} # 빈 set
for reg in reg_list:
    wc[reg] = wc.get(reg,0) + 1 # 해당하는 값이 있으면 0, 있으면 +1

wc

# 빈도가 가장 높은 ExtraTreesRegressor로 회귀모델 결정
```

```
{'GradientBoostingRegressor': 7,
 'ExtraTreesRegressor': 80,
 'ExtraTreeRegressor': 3,
 'XGBRegressor': 6,
 'KNeighborsRegressor': 3,
 'RandomForestRegressor': 1}
```

- 최적의 회귀모델을 찾아주는 LazyRegressor 100회 반복 후 빈도 확인
- 설명력이 가장 높게 나온 빈도가 제일 큰 ExtraTreesRegressor로 회귀모델 결정

02

데이터 전처리 및 분석

3) 회귀분석 - ExtraTreesRegressor

구분	Boostraping	Split 선택 기준
Random Forest	O	주어진 샘플의 모든 변수에 대한 정보이득 계산 -> 가장 설명력 높은 변수의 Partition을 채택
Extra Trees Regressor	X	무작위로 변수 선정 -> 선정된 변수에 대해 최적의 Partition을 찾아 분할

1) Bias(편향)을 낮출 수 있음

2) Variance(분산)을 줄일 수 있음

3) 연산속도가 Random Forest에 비해 약 3배 빠름

데이터 전처리 및 분석

3) 회귀분석 - ExtraTreesRegressor

```
# 3-4) ExtraTreesRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import GridSearchCV # 최적의 parameter 탐색

# train set/test set 생성
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# 기본 parameter로 회귀모델 생성
extra_model = ExtraTreesRegressor().fit(X_train, y_train)

# GridSearch model -> 최적의 파라미터 빈도수 파악
param_grid = {'n_estimators' : [100, 150, 200],
              'max_features' : ['auto', 'sqrt']
              }
```

```
# GridSearch 100회 반복 후 hyper parameter 결과값 저장
best_param_list = []
for i in range(100):

    grid_model = GridSearchCV(extra_model, param_grid=param_grid,
                              scoring='neg_mean_absolute_error',
                              return_train_score=True,
                              cv=5, n_jobs=-1)

    grid_model.fit(X_train, y_train)
    param_list = [[grid_model.best_params_['n_estimators']
                  ,grid_model.best_params_['max_features']]]
    best_param_list.append(param_list)
```

- Hyper parameter 탐색을 위한 GridSearchCV 활용
- 100회 반복 후 Hyper parameter 탐색 결과 저장

데이터 전처리 및 분석

3) 회귀분석 - ExtraTreesRegressor

```
# hyper parameter 빈도 확인

# list -> DataFrame
param_df = pd.DataFrame(best_param_list)

param_count = param_df[0].value_counts()
param_count # 빈도수가 가장 높은 n_estimators=100, max_features='sqrt'로 hyper parameter 채택
```

```
[100, sqrt]    23
[100, auto]    20
[150, auto]    18
[150, sqrt]    15
[200, sqrt]    13
[200, auto]    11
Name: 0, dtype: int64
```

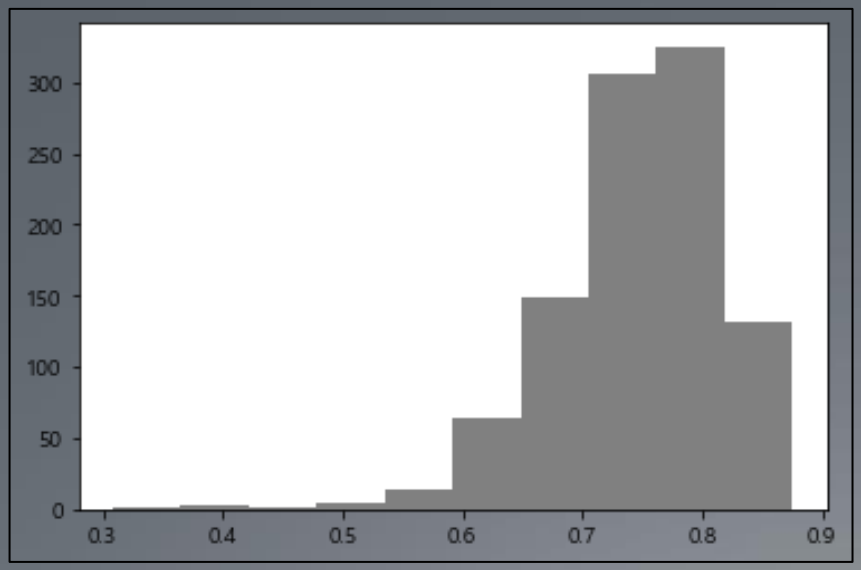
```
# best parameter 적용 후 모델 train/test split + model 생성 1001회 반복
ran_dict = {}
for i in range(1001): # random_state = 0~1000 -> random_state + score를 dictionary 형태로 저장
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
                                                        ,random_state=i)

    new_extra_model = ExtraTreesRegressor(n_estimators=100,
                                          max_features='sqrt',
                                          random_state=i
                                          ).fit(X_train, y_train)
    score = new_extra_model.score(X_test, y_test)
    ran_dict[i] = score
```

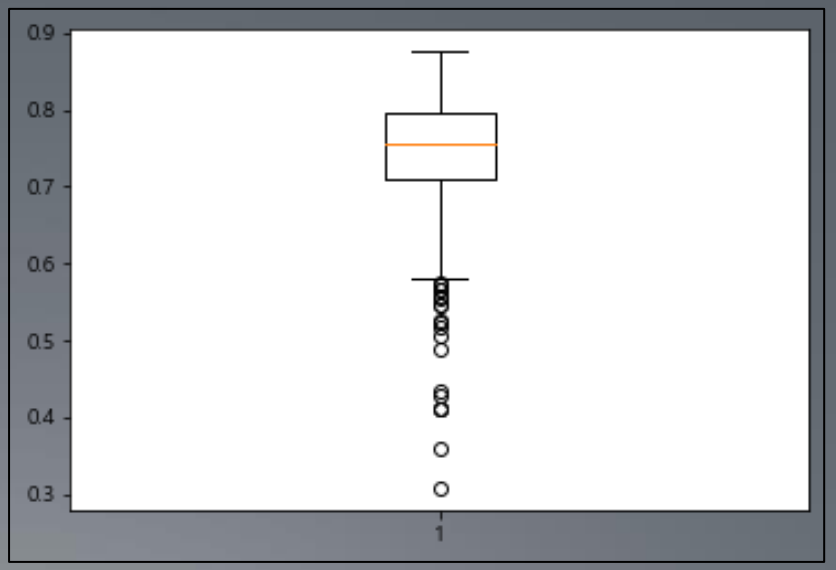
- 빈도 수에 따라 Hyper Parameter 설정: N_estimators = 100, max_features = 'sqrt'
- GridSearch를 통해 채택한 hyper parameter를 적용하여 train/test split + model 생성 1001회 반복 (random_state=0~1000)

02

데이터 전처리 및 분석

3) 회귀분석 - ExtraTreesRegressor





- 히스토그램 및 Boxplot 확인 결과 이상치 발견됨
- 이상치의 영향을 비교적 덜 받는 중위수를 모델의 최종 score로 채택

데이터 전처리 및 분석

3) 회귀분석 - ExtraTreesRegressor

```
# 설명력이 중위수값을 가지는 random state 값 찾기

median_score = st.median(scores)
print(median_score)
```

```
0.7545539536791974
```

```
for i in range(1001):
    if ran_dict[i] == median_score:
        print('random state =', i)
    else:
        continue
```

```
random state = 991
```

```
# 채택한 random_state 값으로 설정 및 모델 재생성
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
                                                    ,random_state=991)

final_extra_model = ExtraTreesRegressor(n_estimators=100,
                                        max_features='sqrt',
                                        random_state=991,
                                        ).fit(X_train, y_train)
score = final_extra_model.score(X_test, y_test)
score # 최종 score : 0.7545539536791974
```

- 중위수의 score를 생성하는 random_state 값 탐색
- random_state 설정 후 최종 모델 생성: **설명력 약 75.4%**

02

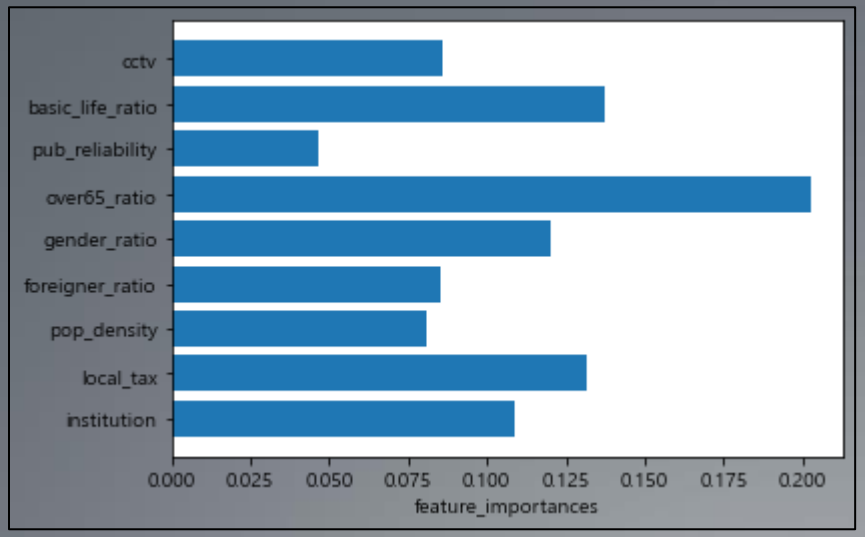
데이터 전처리 및 분석

3) 회귀분석 - ExtraTreesRegressor

```
# 중요변수 확인 및 시각화
final_extra_model.feature_importances_
```

```
array([0.10905671, 0.13163569, 0.08111121, 0.08535313, 0.12009424,
       0.20243017, 0.04684107, 0.1374932 , 0.08598456])
```

```
plt.barh(range(9), width=final_extra_model.feature_importances_)
# y축 눈금 : x변수
plt.yticks(range(9), X.columns)
plt.xlabel('feature_importances')
plt.show() # 65세 이상 인구 비율, 기초생활 수급자 비율, 1인당 지방세 납부액 순
```



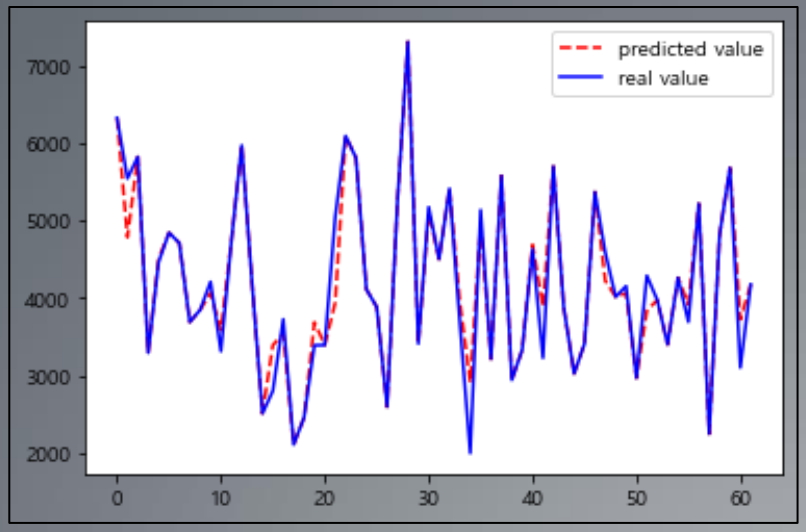
- 65세 이상 인구 비율, 기초생활 수급자 비율, 1인당 지방세 납부액 순으로 유의한 변수로 확인됨

데이터 전처리 및 분석

3) 회귀분석 - ExtraTreesRegressor

```
# validation set 생성 후 모델에 적용
import numpy as np
from sklearn.metrics import r2_score

# dataset의 50% 크기로 validation set 생성 -> index 순서 무작위
idx = np.random.choice(a = len(crime), size = int(len(crime)*0.5),
                       replace=False)

new_crime = crime.iloc[idx]

# 검증용 X, y 변수 생성
new_X = new_crime.iloc[:,1:]
new_Y = new_crime.iloc[:,0]

y_pred = final_extra_model.predict(new_X)
y_true = new_Y

val_score = r2_score(y_true, y_pred)
print(val_score)
```

```
0.9444786865813704
```



- Validation set 생성 후 모델에 적용
- R2 score = 약 0.94

03

분석 결과 및 해석

- LinearRegreesion, OLS, ExtraTreesRegressor로 각각 회귀모델을 생성한 결과 ExtraTressRegressor 회귀모델의 설명력이 약 **75%**로 가장 높게 나타남

- ExtraTressRegressor 모델 기준 65세 이상 인구 비율, 기초생활 수급자 비율, 1인당 평균 지방세 납부액 순으로 범죄 발생에 유의한 변수로 나타남

04

느낀 점 및 한계점

- 데이터셋의 수가 충분하지 않아 정확하고 일관적인 결과를 가져오는데 어려움이 있었음

- 서울이 아닌 타 지역의 데이터셋 수집에 한계 -> '생성한 모델을 타 지역의 데이터셋에 적용해볼 수 있었으면 성능을 보다 객관적으로 검증할 수 있지 않았을까' 에 대한 아쉬움

감사합니다