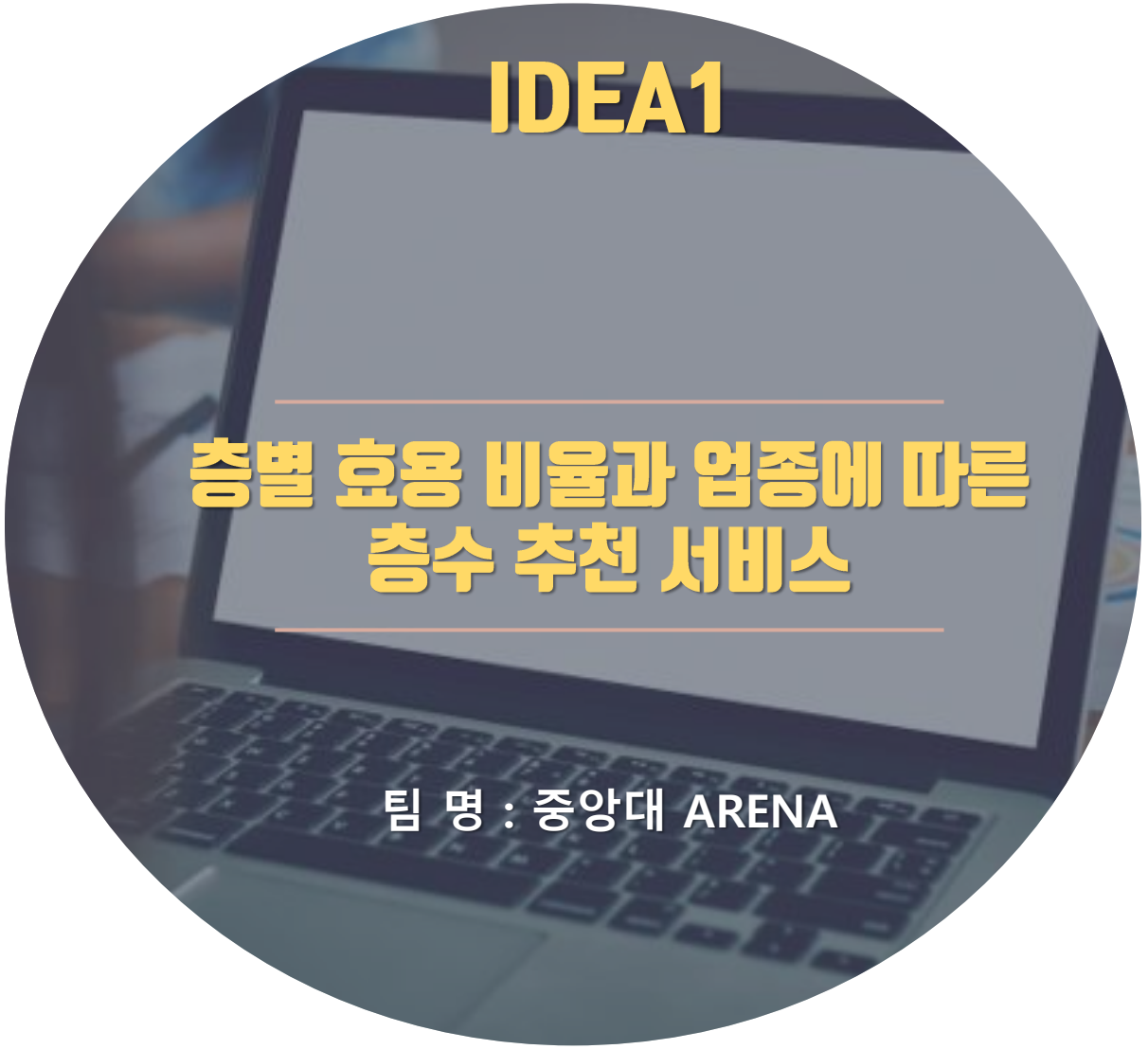
소상공인 지원을 위한 **상권분석 빅데이터 경진대회**

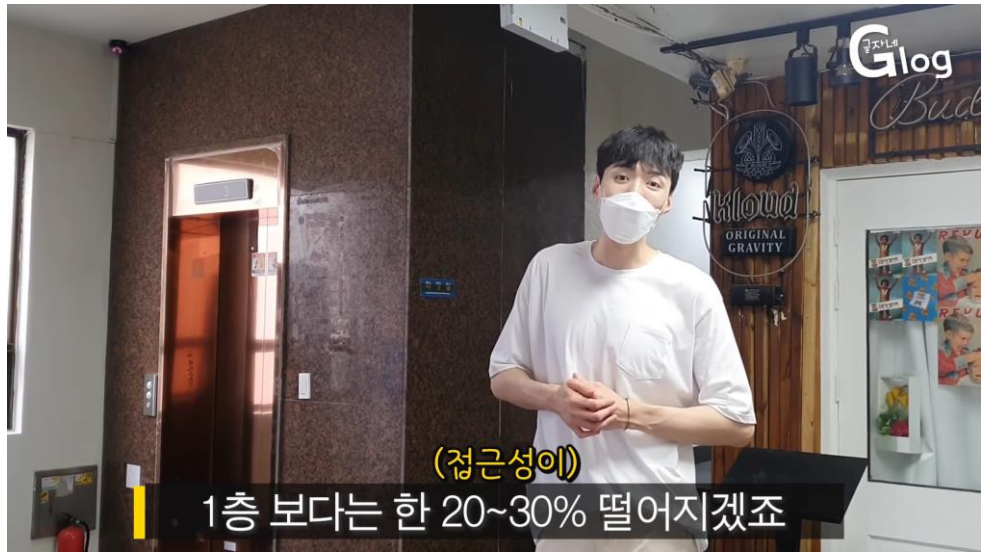


IDEA1

층별 효용 비율과 업종에 따른 층수 추천 서비스

팀 명 : 중앙대 ARENA

1. 분석배경



한 유튜버가 술집 창업을 위해서 어디에 가게를 차릴 것인지 이야기 하고 있다. 1층은 가격이 높기 때문에 접근성이 20~30% 떨어지는 2층을 선택했지만 가성비를 그 이상으로 끌어올린다면 괜찮을 것이라 영상에서 말하고 있다. 비싼 1층에 가게를 차릴지 혹은 저렴한 2층에 가게를 차릴지 많은 고민을 했다고 한다.

이 영상을 보고 이러한 고민이 저 유튜버만의 고민이 아닐 것이라 생각했다. 대부분의 소상공인들이 창업을 할 때 상대적으로 비싸지만 사람들의 접근성이 뛰어난 1층에 가게를 차릴지, 아님 가격이 그나마 저렴하지만 접근성이 떨어지는 그 외의 층에 가게를 차릴지 많은 고민이 있을 것이다.

상세내용 : https://youtu.be/Kha4jwQAI-I

1. 분석배경

소상공인들이 창업을 할 때 중요하게 생각하는 것은 '고객의 접근성' 이다. 아무리 가성비 좋은 곳을 골라 가게를 차려도 사람들의 접근성이 미미한 지역이라면, 가성비는 아무 쓸모가 없어진다.

집객시설이란 상권에 유동인구를 흡수하는 영업시설로 극장 대형마트 관공서 등 말한다

즉 집객시설이 주변에 많을 수록 손님들의 접근성도 높아지게 된다.

집객시설 수

층별 임대시세

집객시설 수 데이터와 층별 임대시세 데이터를 이용한다면 이러한 고민을 해결 할 수 있다.

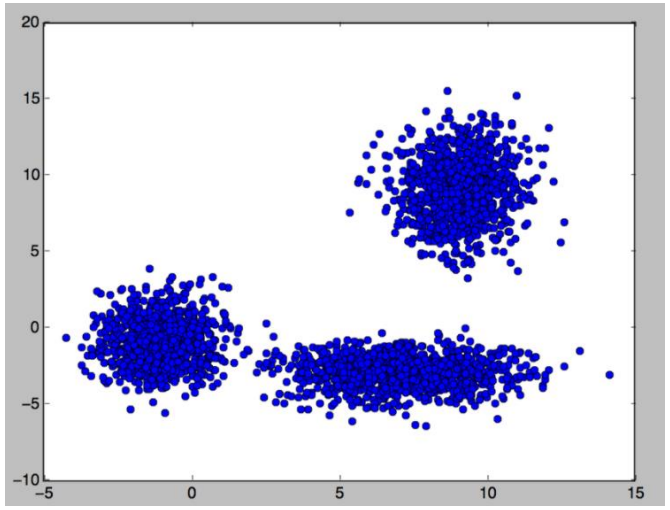
2. 데이터 분석 방법

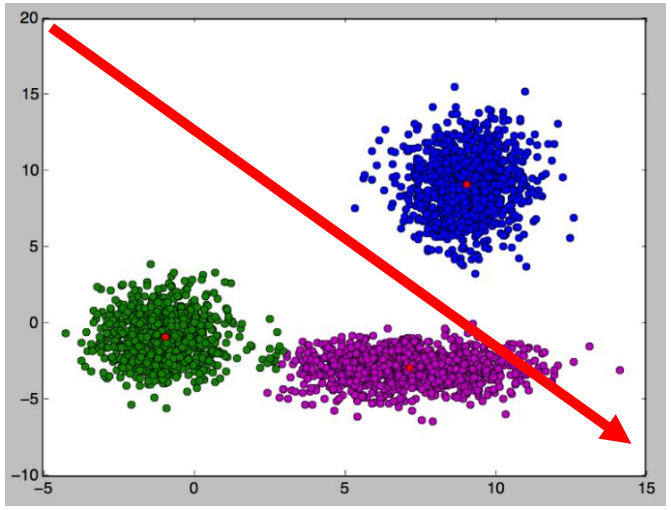
K-means Cluster Analysis

군집분석은 패턴 공간에 주어진 유한 개의 패턴들이 서로 가깝게 모여서 무리를 이루고 있는 패턴 집합을 묶는 과정, K-means, 계층형 밀도기반 군집 등이 포함된다

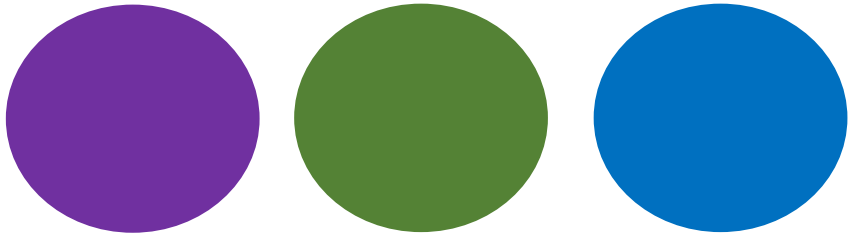
K-means 군집화: 주어진 데이터를 n개의 클러스터로 묶는 알고리즘으로, 각 클러스터와 거리 차이의 분산을 최소화하는 방식으로 동작한다

K-means 라이브러리를 이용하여 집객시설과 층별효용비를 분석할 예정





K-군집화로 분류한 군집들은 블록 근처에 위치한 집객시설이 많을 수록, n층의 층별 효용비가 낮을 수록 그 해당블록의 n층의 임대시세 가성비가 좋다는 것을 의미



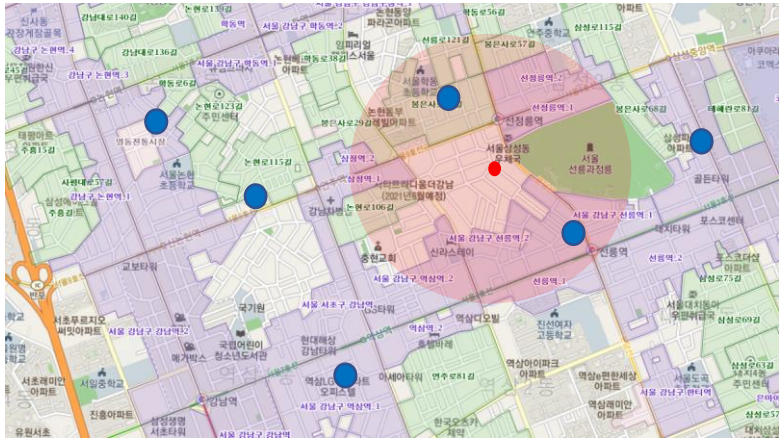
접근성 ↑
가격 ↓

접근성 ↓
가격 ↓

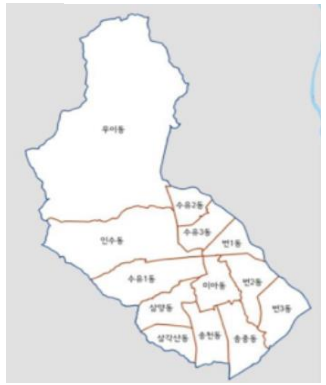
접근성 ↑
가격 ↑

위와 같은 예시로 접근성 대비 가격측면에서 지역을 군집화 하여 소상공인에게 필요한 지역정보를 제공해 줄 수 있다.

2. 데이터 분석 방법

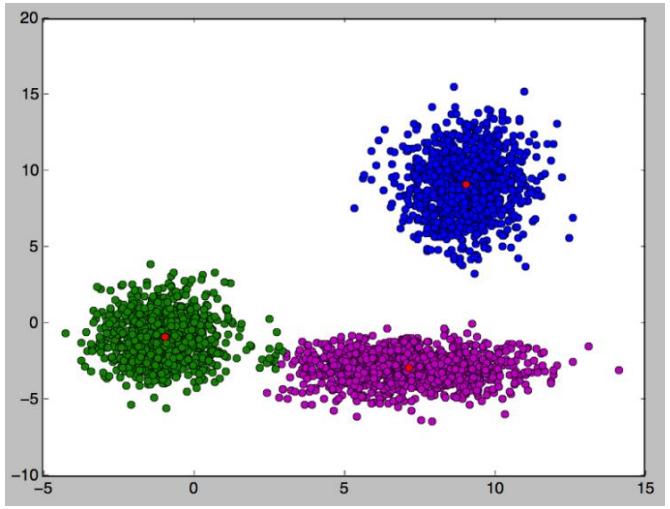


블록에서 일정 거리 이상 위치한 집객 시설 수 데이터 수집



행정동 마다 층별 효용비율 계산 후 데이터화

K-mean 라이브러리를 이용한 K 평균 군집화



군집화 그래프 추출

3. 분석 데이터

9	상권	CSV	행정동집계_임대시세_분기	RENT_CURPRC	1	기준_년_코드
	상권	CSV	행정동집계_임대시세_분기	RENT_CURPRC	2	기준_분기_코드
	상권	CSV	행정동집계_임대시세_분기	RENT_CURPRC	3	행정동_코드
	상권	CSV	행정동집계_임대시세_분기	RENT_CURPRC	4	임대시세_층구분_코드
	상권	CSV	행정동집계_임대시세_분기	RENT_CURPRC	5	보증금_평균
	상권	CSV	행정동집계_임대시세_분기	RENT_CURPRC	6	월임대료_평균
	상권	CSV	행정동집계_임대시세_분기	RENT_CURPRC	7	환산_임대료_평균
	상권	CSV	행정동집계_임대시세_분기	RENT_CURPRC	8	임대건수
23	코드	CSV	블록영역정보	BLCK_DIM	1	블록_코드
	코드	CSV	블록영역정보	BLCK_DIM	2	블록_명
	코드	CSV	블록영역정보	BLCK_DIM	3	행정동_코드
	코드	CSV	블록영역정보	BLCK_DIM	4	엑스좌표_최소_값
	코드	CSV	블록영역정보	BLCK_DIM	5	와이좌표_최소_값
	코드	CSV	블록영역정보	BLCK_DIM	6	엑스좌표_최대_값
	코드	CSV	블록영역정보	BLCK_DIM	7	와이좌표_최대_값
	코드	CSV	블록영역정보	BLCK_DIM	8	엑스좌표_값
	코드	CSV	블록영역정보	BLCK_DIM	9	와이좌표_값
	코드	CSV	블록영역정보	BLCK_DIM	10	집계구_코드
	코드	CSV	블록영역정보	BLCK_DIM	12	블록_대표_지명
	코드	CSV	블록영역정보	BLCK_DIM	13	시군구_코드
	코드	CSV	블록영역정보	BLCK_DIM	14	영역_면적
18	상권	CSV	DW_집객시설정보	VIATR_FCLTY_INFO	1	기준_년월
	상권	CSV	DW_집객시설정보	VIATR_FCLTY_INFO	2	집객_시설_구분_코드
	상권	CSV	DW_집객시설정보	VIATR_FCLTY_INFO	3	집객_시설_코드
	상권	CSV	DW_집객시설정보	VIATR_FCLTY_INFO	4	집객_시설_ID
	상권	CSV	DW_집객시설정보	VIATR_FCLTY_INFO	5	집객_시설_명
	상권	CSV	DW_집객시설정보	VIATR_FCLTY_INFO	6	주소_코드
	상권	CSV	DW_집객시설정보	VIATR_FCLTY_INFO	7	주소_명
	상권	CSV	DW_집객시설정보	VIATR_FCLTY_INFO	8	전화번호
	상권	CSV	DW_집객시설정보	VIATR_FCLTY_INFO	9	엑스좌표_값
	상권	CSV	DW_집객시설정보	VIATR_FCLTY_INFO	10	와이좌표_값
	상권	CSV	DW_집객시설정보	VIATR_FCLTY_INFO	11	주소_정제_구분자
	상권	CSV	DW_집객시설정보	VIATR_FCLTY_INFO	12	블록_코드
	상권	CSV	DW_집객시설정보	VIATR_FCLTY_INFO	13	등록_일자
	상권	CSV	DW_집객시설정보	VIATR_FCLTY_INFO	14	말소_일자

층별 효용비율 계산 데이터

블록별 근처 집객시설 수 계산 데이터

4. 데이터 전처리 과정

	ADSTRD_CD	XCNTS_VALUE_block	YDNTS_VALUE_block
0	11680660	205139	442688
1	11305555	201925	457763
2	11305555	202139	457628
3	11680565	204256	446790
4	11680630	204471	444947
...	...	...	...
995	11305630	202199	459927
996	11305630	202558	460220
997	11305535	202225	458405
998	11305660	200308	460224
999	11305630	202293	460316

집객시설별 x,y좌표 전처리

	XCNTS_VALUE_attractionP	YDNTS_VALUE_attractionP
0	205885	445740
1	206802	443775
2	202414	459742
3	204162	445980
4	202247	459993
...	...	...
19064	204710	444081
19065	204661	444062
19066	204610	444040
19067	204596	444074
19068	204647	444096

집객시설별 x,y좌표 전처리

	ADSTRD_CD	XCNTS_VALUE_block	YDNTS_VALUE_block	attractionPP_num
0	11680660	205139	442688	1000
1	11305555	201925	457763	1000
2	11305555	202139	457628	1000
3	11680565	204256	446790	1000
4	11680630	204471	444947	1000
...	...	...	...	...
995	11305630	202199	459927	1000
996	11305630	202558	460220	1000
997	11305535	202225	458405	1000
998	11305660	200308	460224	1000
999	11305630	202293	460316	1000

블록별 특정 거리내에 있는 집객시설 수 데이터 프레임

```
number_aP_list=[]
for k in range(block_L):
    attractionPP_list=[]
    block_P_x=block_P.iloc[k,1] #각 블록의 x좌표 값들을 반복문을 통해 순서대로 가져옴
    block_P_y=block_P.iloc[k,2] #각 블록의 y좌표 값들을 반복문을 통해 순서대로 가져옴
    for i in range(attractionPL):
        attractionPP_x=attractionPP.iloc[i,0]#각 집객시설의 x좌표 값들을 반복문을 통해 순서대로 가져옴
        attractionPP_y=attractionPP.iloc[i,1]#각 집객시설의 y좌표 값들을 반복문을 통해 순서대로 가져옴
        number_aP=abs(((block_P_x-attractionPP_x)**2+(block_P_y-attractionPP_y)**2)) #블록 근처에 있는 집객
        if number_aP<=900000000:
            attractionPP_list.append(number_aP) #거리안에 있는 집객시설 데이터 리스트에 추가
    number_aP_dt=len(attractionPP_list) #리스트의 길이를 셈으로 집객시설수 계산, 리스트에 추가
    number_aP_list.append(number_aP_dt) #위의 리스트를 또 다른 리스트에 추가
block_P['attractionPP_num']=number_aP_list #집객시설 수 가 담긴 리스트를 블록들의 좌표가 있는 데이터프레임에
```

이중반복문을 이용해 집객시설의 x,y 좌표값들을 블록좌표값을 이용한 수식에 대입 -> 일정 거리 안에 있는 집객시설과 집객시설의 수를 리스트 형태로 추출-> 데이터프레임과 함침

4. 데이터 전처리 과정

	ADSTRD_CD	RENT_FLOOR_CD	CNVRSN_RNTCHRG
14	11305534	0	75960
82	11305534	1	94350
66	11305534	2	46253
7	11305535	0	75531
70	11305535	1	89508
...	...	...	...
68	11680740	1	0
99	11680740	2	74998
32	11680750	0	0
79	11680750	1	0
58	11680750	2	143553

행정동별 층별 임대시세 데이터 전처리

```
pop=0
for i in rent_valueP_size_list:
    pop=pop+i
    floor_ratio.append(1) #1층의 효용비는 항상 1이기에 1을 리스트에 추가하고 시작
    if i==4: #1,2,3,4 층의 데이터가 있을 때
        _2f_ratio_cal=rent_value_plus.iloc[pop-3,2]/rent_value_plus.iloc[pop-4,2] #2층 효용비 계산
        _3f_ratio_cal=rent_value_plus.iloc[pop-2,2]/rent_value_plus.iloc[pop-4,2] #3층 효용비 계산
        _4f_ratio_cal=rent_value_plus.iloc[pop-1,2]/rent_value_plus.iloc[pop-4,2] #4층 효용비 계산
        floor_ratio.append(_2f_ratio_cal)
        floor_ratio.append(_3f_ratio_cal)
        floor_ratio.append(_4f_ratio_cal)
    elif i==3: #1,2,3 층의 데이터가 있을 때
        _2f_ratio_cal=rent_value_plus.iloc[pop-2,2]/rent_value_plus.iloc[pop-3,2] #2층 효용비 계산
        _3f_ratio_cal=rent_value_plus.iloc[pop-1,2]/rent_value_plus.iloc[pop-3,2] #3층 효용비 계산
        floor_ratio.append(_2f_ratio_cal)
        floor_ratio.append(_3f_ratio_cal)
    elif i==2: #1,2 층의 데이터가 있을 때
        _2f_ratio_cal=rent_value_plus.iloc[pop-1,2]/rent_value_plus.iloc[pop-2,2] #2층 효용비 계산
        floor_ratio.append(_2f_ratio_cal)
    else:
        floor_ratio.append(0) #이외의 값들은 0처리함

rent_value_plus['floor_ratio']=floor_ratio #구한 효용비 데이터를 데이터프레임에 추가
```

전처리한 데이터 프레임을 사용하여 층별 효용비율을 계산(2층)

	ADSTRD_CD	XCNTS_VALUE_block	YDNTS_VALUE_block	attractionPP_num	2floor_ratio
0	11680660	205139	442688	1000	0.000000
1	11305555	201925	457763	1000	0.608151
2	11305555	202139	457628	1000	0.608151
3	11680565	204256	446790	1000	0.885037
4	11680630	204471	444947	1000	0.746010
...	...	...	...	...	...
995	11305630	202199	459927	1000	0.698971
996	11305630	202558	460220	1000	0.698971
997	11305535	202225	458405	1000	0.555878
998	11305660	200308	460224	1000	0.593388
999	11305630	202293	460316	1000	0.698971

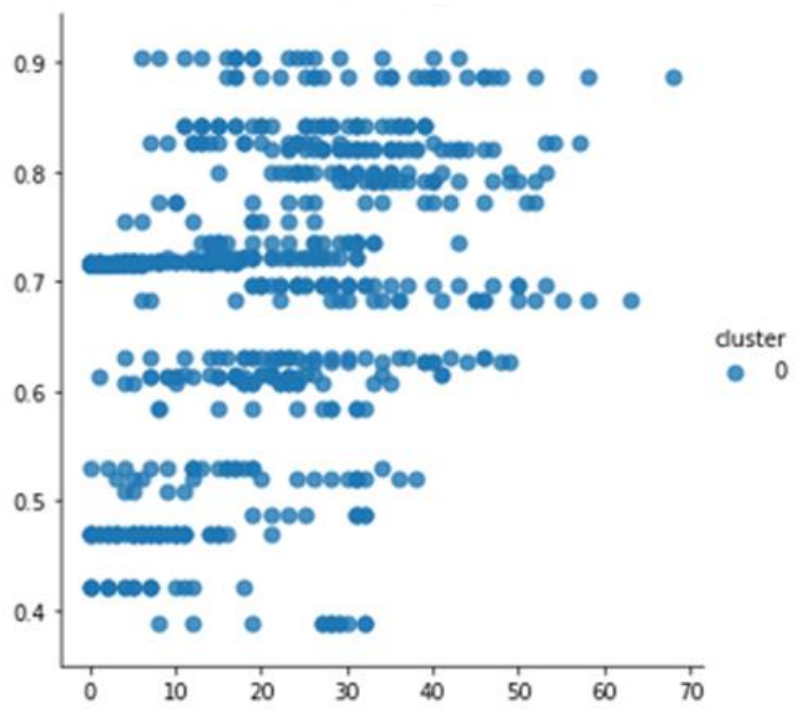
층별효용비율 데이터를 층별로 가로로 전처리

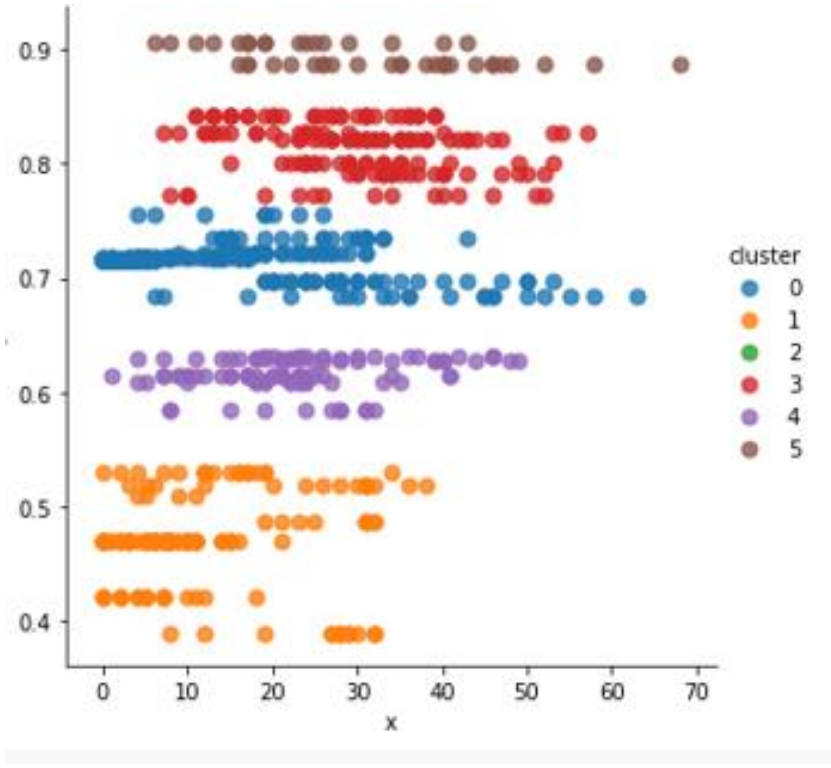
```
points=block_PP.values
kmeans = KMeans(n_clusters=6).fit(points) #클러스터 수 조절
kmeans.cluster_centers_
block_PP['cluster']=kmeans.labels_
sb.lmplot('attractionPP_num', '2floor_ratio', data=block_PP, fit_reg=False, scatter_kws={"s":100}, hue="clus
plt.title("K-means")
plt.xlabel('x')
plt.ylabel('y')
block_PP
```

K-means 라이브러리를 이용한 군집화

5. K-means Cluster Analysis

블럭별 2층 층별효용비율 대비 집객시설 수(접근성) K-means 분석 결과

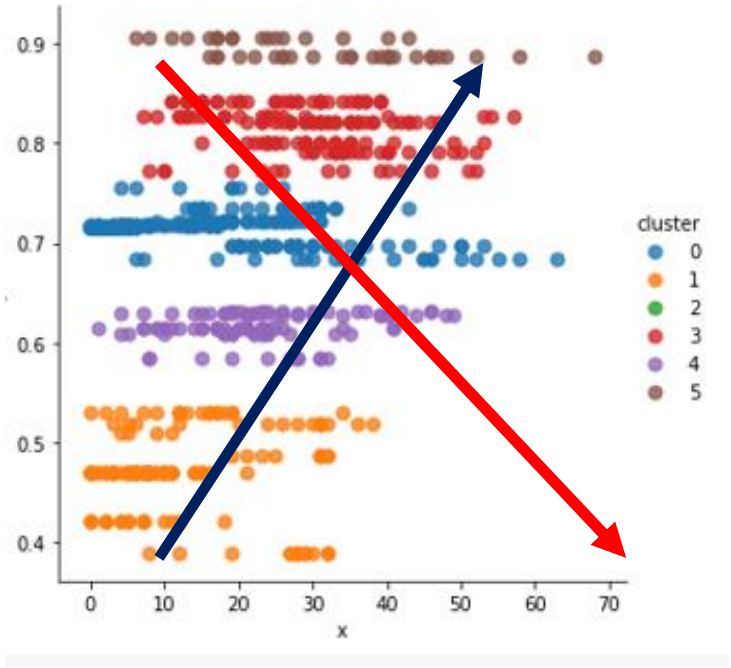


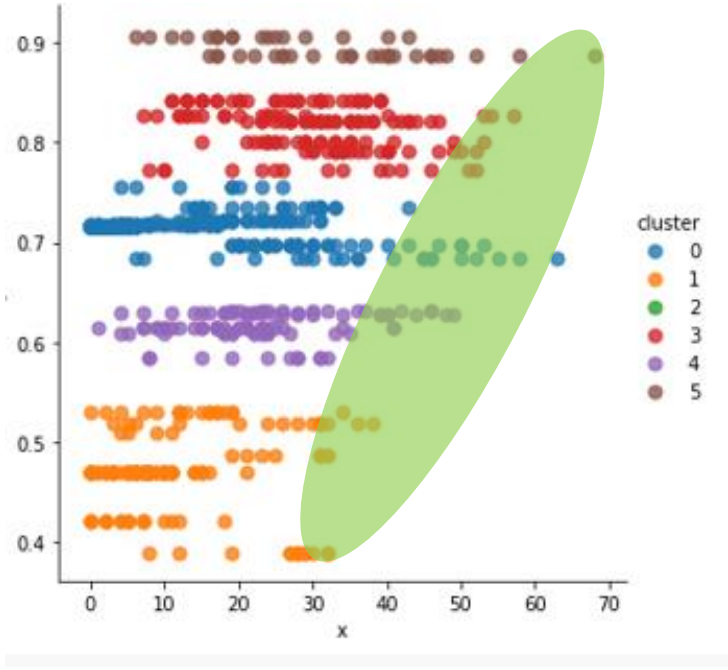


클러스터의 수는 클러스터 변수를 증가시켜도 군집화 되는 블록들은 5개로 일정했다.

6. 결과 해석

블럭별 2층 층별효용비율 대비 집객시설 수(접근성) K-means 분석 결과





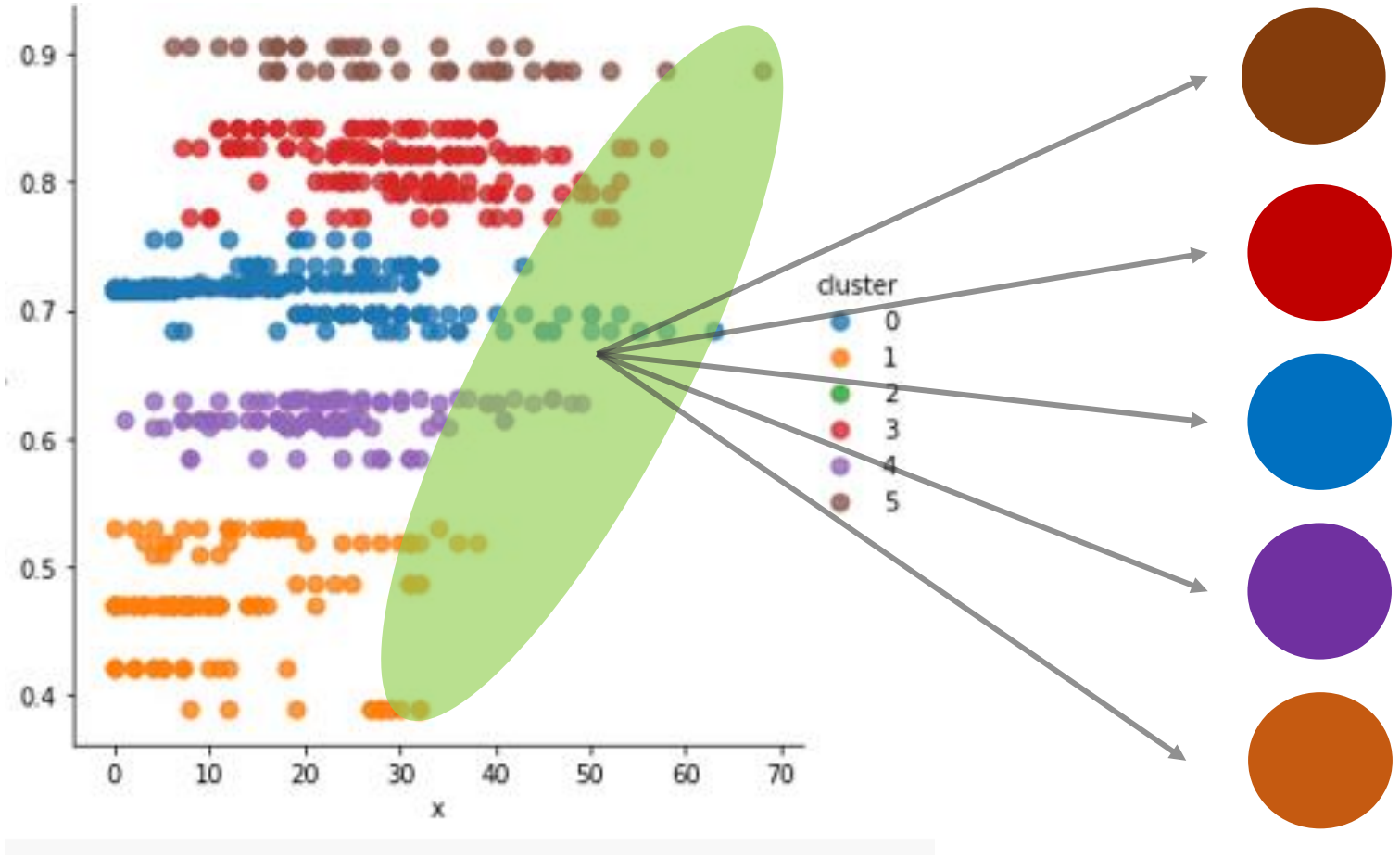
왼쪽의 그래프의 빨간색 화살표처럼 아래로 내려갈 수록 근처에 위치한 집객시설 수는 많고 층별 효용비는 내려감으로 가격대비 사람들의 접근성이 뛰어난 곳이라고 말할 수 있다.

하지만 군집화한 블록들을 보면 왼쪽 남색 화살표와 같이 근처 집객시설 수가 늘어날수록 층별 효용비 또한 가파르게 오르고 있는 것을 확인할 수 있다. 이는 접근성이 좋은 지역은 그만큼 가격대비 비싸다는 것을 알 수 있는 지표이다.

연두색 영역은 근처 집객시설 수 대비 층별 효용비가 상대적으로 완만한 기울기를 가진 지역으로 이 부분에 해당하는 블록들은 접근성 대비 가성비가 뛰어난 블록들임을 알 수 있다.

6. 결과 해석

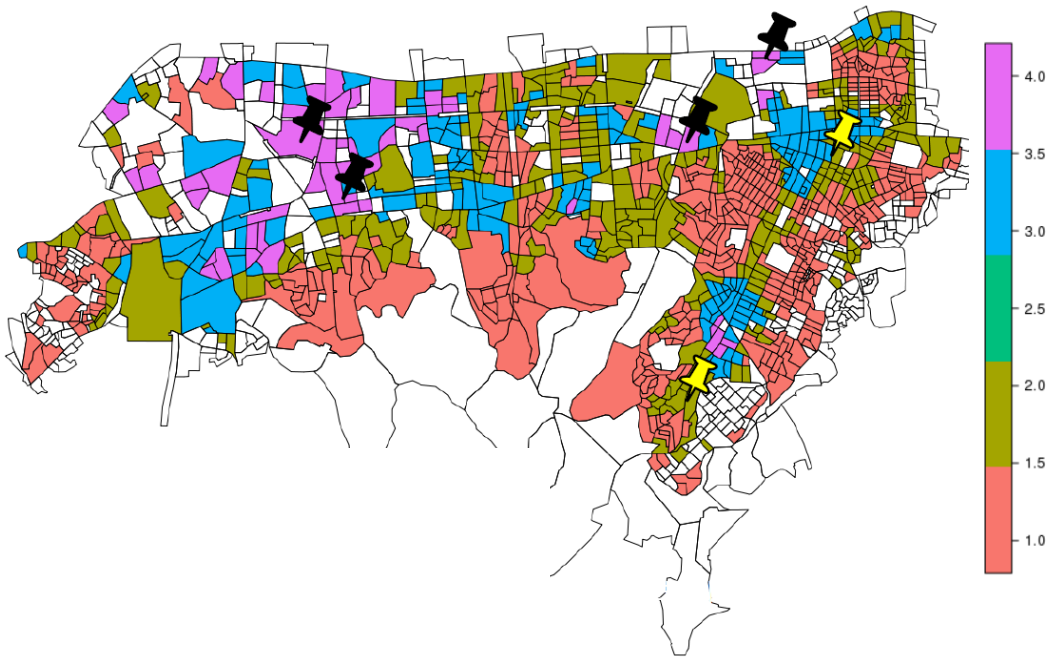
블럭별 2층 층별효용비율 대비 집객시설 수(접근성) K-means 분석 결과

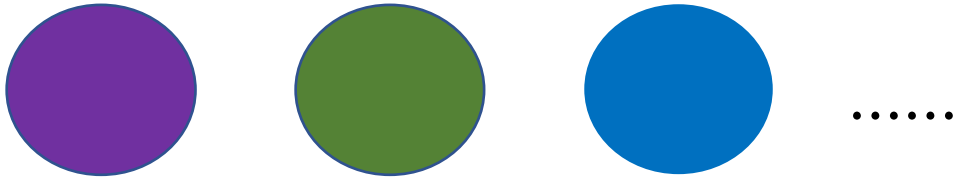


위 연두색의 영역에서 각 군집화된 블록들을 추출해 낼 수 있을 것이고, 추출해낸 블록들을 가지고 소상공인들에게 유의미한 데이터 제공이 가능하다.

(연두색 영역은 접근성대비 가성비가 좋은 지역을 대략 예시로 든 것이지 저 데이터를 가지면 이보다 많고 도움이 되는 데이터들을 뽑아 낼 수 있을 것이다.)

7. 기대효과





접근성 ↑
가격 ↓

접근성 ↓
가격 ↓

접근성 ↑
가격 ↑

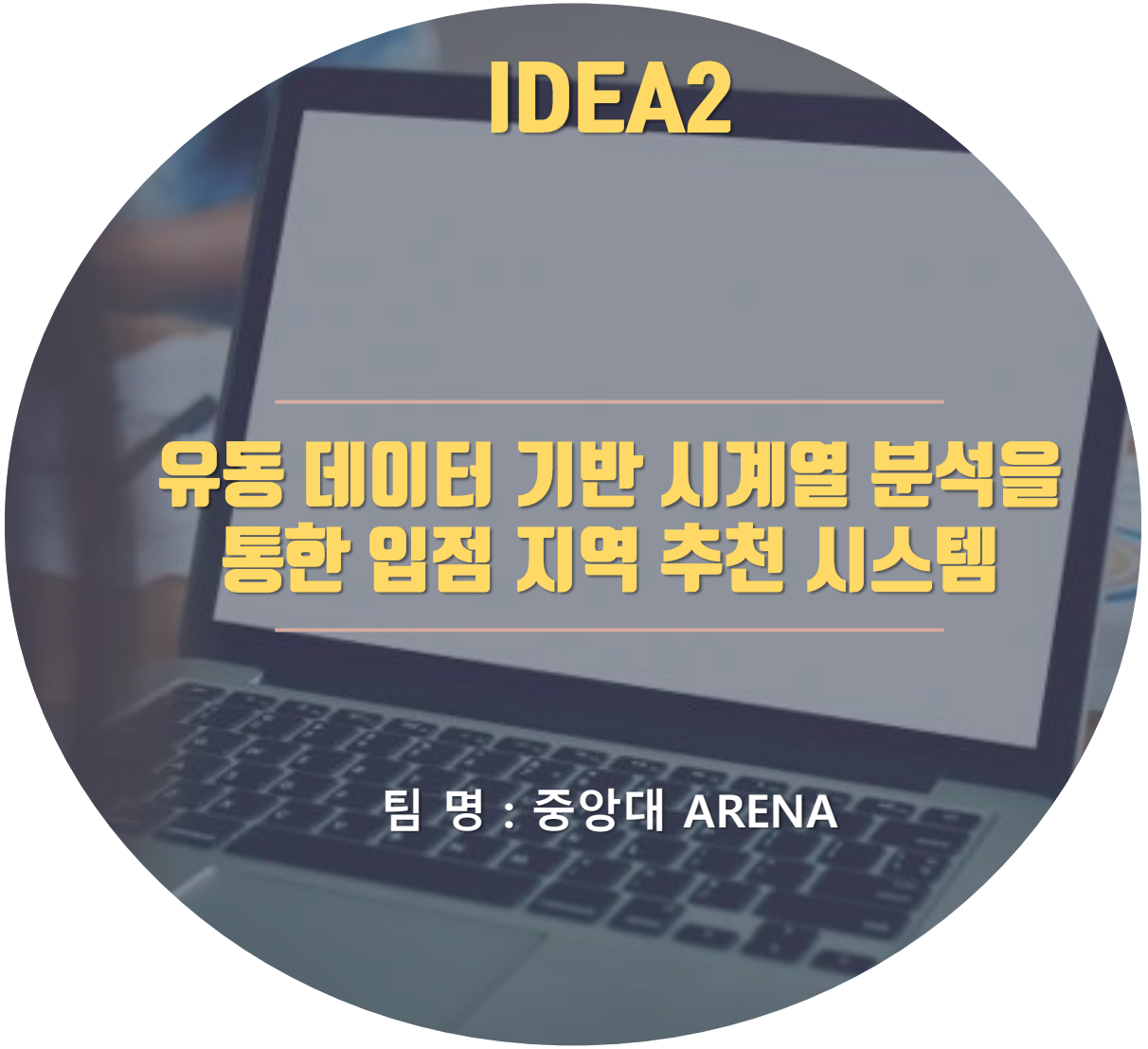
현재 층별 효용비율은 단순히 가격측면에서 측정되고 여러 분야에서 적극적으로 활용되어지지 않는 데이터다. 여러 논문에서도 층별효용비가 여러 분야에서 적극적으로 쓰이지 않는다고 말하고 있다.

즉 이는 가성비가 좋은 지역에 창업을 하고 싶을 때 일일이 찾아다니고 가격 등 여러 요소를 비교하면서 창업 지역을 정해야 하는 소상공인들의 현실을 나타냄과 동시에 가격대비 소상공인들에게 적합한 지역을 추천해주는 서비스가 필요함을 보여주고 있다.

층별 효용비율 데이터와 사람들의 접근성에 관련이 높은 집객시설 수를 이용하면 접근성 대비 가성비가 좋은 지역을 층별로 추려낼 수 있을 것이고, 군집화한 블록들을 토대로 왼쪽 그림처럼 사용자들이 쉽게 알아볼 수 있도록 시각화 데이터 같은 도움이 되는 정보들을 제공할 수 있을 것이다. 이러한 정보제공은 소상공인들이 쉽게 자신의 조건에서 알맞은 지역에 창업을 할 수 있도록 도와 줄 수 있다.

소상공인들한테 가장 중요한 접근성과 가격에 따라 지역을 군집화하여 소상공인들에게 필요한 지역을 보여주는 이런 분석방법은 많은 도움이 될 것이라 확신한다.

소상공인 지원을 위한 **상권분석 빅데이터 경진대회**



IDEA2

유동 데이터 기반 시계열 분석을 통한 입점 지역 추천 시스템

팀 명 : 중앙대 ARENA

아이디어 소개

창업을 할 때에는 '어떤 업종'인가도 중요하지만, '어떤 위치'인가도 매우 중요하다. 실제로 2018년 소상공인 실태조사에서 '입지 선정'이 창업과정에서의 어려움 2위를 차지하였을 정도로 소상공인들은 '입지 선정'을 중요하게 여기고, 동시에 이에 대해 어려움을 겪는다.

따라서 소상공인들의 이러한 고민 해결에 도움을 주기 위하여 우리 팀은 입점 지역을 추천하는 서비스를 고안하였고, 이를 위하여 지금 현재의 데이터 뿐만 아니라 시계열 분석을 이용하여 데이터를 예측하는 방법을 떠올렸다.

창업과정 중 가장 어려움을 느낀 일은 자금조달임

- 사업체를 운영하면서 느낀 창업 과정의 어려운 일을 100점 만점으로 환산했을 때, 자금조달 58.9점, 입지선정 51.2점, 경영방법 42.5점 순으로 중요하다고 응답함

입지선정의 어려움은 '어려움' 29.1%, 평균 51.2점

아이디어 소개

매출액 **결정 요인**의 분석

결정 요인을 참고로
시계열 분석을 통한 예측

창업자들의 상황을
고려한 **입점 지역 추천**

데이터 전처리

```
In [9]: blck_flpop.columns = ['STDR_YM_CD','BLCK_CD','TOT_FLPOP_CO','ML_FLPOP_CO','FML_FLPOP_CO','AGRDE_10_FLPOP_CO','AGRDE_20_FLPOP_CO','AGRDE_30_F
```

```
In [10]: blck_flpop.drop(blck_flpop.index[30000:],axis='rows',inplace=True) #20만개 데이터에서 3만개만 추출
```

```
In [12]: selng.drop(selng.index[30000:],axis='rows',inplace=True) #20만개데이터에서 3만개만 추출
```

```
In [13]: selng
```

Out[13]:

	BLCK_CD	TA_YM	KSIC_CD	MCT_CNT	AMT	CNT	MIN_AMT	MIN_CNT	MAX_AMT	MAX_CNT	...	RCNT_60	MCT_SALES	NEW_MCT	P1_M6	I
0	10019	201701	47413	1	5000000	1	5000000	1	5000000	1	...	0	25	0	0	
1	10019	201701	47416	1	20158000	15	20158000	15	20158000	15	...	1	130	0	0	
2	10019	201701	47420	1	1429000	3	1429000	3	1429000	3	...	0	19	0	0	
3	10019	201701	47811	1	12668290	706	12668290	706	12668290	706	...	33	101	0	0	
4	10019	201701	47813	1	4000000	1	4000000	1	4000000	1	...	0	13	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
29995	274682	201702	85611	1	1520000	4	1520000	4	1520000	4	...	0	67	0	0	
29996	274682	201702	95213	1	2167000	51	2167000	51	2167000	51	...	0	94	0	0	
29997	274682	201702	96119	1	98000	2	98000	2	98000	2	...	0	24	0	0	
29998	274683	201702	96112	2	4375000	106	174000	5	4201000	101	...	0	55	0	0	
29999	274684	201702	56199	1	281300	36	281300	36	281300	36	...	0	0	1	1	

30000 rows × 71 columns

매출금액(SELNG), 블록집계 상존인구(BLCK_FLPOP), 임대시세(RENT_CURPRC) 테이블의 데이터를 이용하되 상존 인구, 매출 금액 데이터에서는 데이터 처리 시간 단축을 위하여 임의로 3만개를 추출하여 진행함

데이터 전처리

In [14]:
```
data_1=blck_flpop[blck_flpop['BLCK_CD'].isin(seing['BLCK_CD'].values)]
#blck_cd가 겹치는 경우의 코드를 따로 추출
data_2=blck_dim[blck_dim['ADSTRD_CD'].isin(rent_curprc['ADSTRD_CD'].values)]
data_3=rent_curprc.set_index('ADSTRD_CD').join(data_2.set_index('ADSTRD_CD'), how='outer').reset_index()
```

In [15]:
```
#ADSTRD_CD와 BLCK_CD연결해서 RENT_CURPRC 수정
data_5=seing.set_index('BLCK_CD').join(data_1.set_index('BLCK_CD'), how='outer').reset_index()#코드 합치기
```

블록코드(BLCK_CD)를 기준으로 매출금액과 상존인구 데이터를 합침.
또한 임대시세의 경우 블록코드 대신 행정동 코드(ADSTRD_CD)만 존재하므로
행정동 코드와 블록 코드가 모두 있는 블록정보(BLCK_DIM) 테이블의
데이터를 이용하여 행정동 코드와 블록코드를 연결함.

데이터 전처리

```
In [16]: data_5.drop(data_5.index[40000:],axis='rows',inplace=True) #70만개 데이터에서 4만개만 추출
```

```
In [18]: data_3.drop(data_3.index[40000:],axis='rows',inplace=True) #70만개 데이터에서 4만개만 추출
```

역시 데이터 수가 많으므로 매출금액과 상존인구 데이터를 합친 data_5와 임대시세 데이터를 블록 코드를 기준으로 합친 data_3을 각각 데이터 처리 시간 단축을 위하여 4만개만 추출함

데이터 전처리

```
In [20]: data_6=data_5.set_index('BLCK_CD').join(data_3.set_index('BLCK_CD'), how='outer').reset_index()#최종 코드 합치기

In [22]: data_6.drop(data_6.index[40000:],axis='rows',inplace=True) #NaN 이전 값만추출

In [23]: data_6
```

Out[23]:

	BLCK_CD	TA_YM	KSIC_CD	MCT_CNT	AMT	CNT	MIN_AMT	MIN_CNT	MAX_AMT	MAX_CNT	...	XCNTS_MIN_VALUE	YDNTS_MIN_VALU
0	118	201701.0	47122.0	1.0	10334490.0	1867.0	10334490.0	1867.0	10334490.0	1867.0	...	NaN	N
1	118	201701.0	47122.0	1.0	10334490.0	1867.0	10334490.0	1867.0	10334490.0	1867.0	...	NaN	N
2	118	201701.0	47122.0	1.0	10334490.0	1867.0	10334490.0	1867.0	10334490.0	1867.0	...	NaN	N
3	118	201701.0	47122.0	1.0	10334490.0	1867.0	10334490.0	1867.0	10334490.0	1867.0	...	NaN	N
4	118	201701.0	47122.0	1.0	10334490.0	1867.0	10334490.0	1867.0	10334490.0	1867.0	...	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	
39995	33564	201701.0	56219.0	1.0	4428500.0	74.0	4428500.0	74.0	4428500.0	74.0	...	NaN	N
39996	33564	201701.0	56220.0	2.0	8208600.0	1052.0	642200.0	189.0	7566400.0	863.0	...	NaN	N
39997	33564	201701.0	56220.0	2.0	8208600.0	1052.0	642200.0	189.0	7566400.0	863.0	...	NaN	N
39998	33564	201701.0	56220.0	2.0	8208600.0	1052.0	642200.0	189.0	7566400.0	863.0	...	NaN	N
39999	33564	201701.0	56220.0	2.0	8208600.0	1052.0	642200.0	189.0	7566400.0	863.0	...	NaN	N

이후 data_5와 data_3을 블록코드를 기준으로 합쳐서 data_6을 완성함

그 후에 매출값(AMT)이 NaN으로 나오는 데이터들을 드랍하여 제거함

데이터 전처리

In [24]:
```
data_6.sort_values(by=['BLCK_CD','STDR_YM_CD'],axis=0,inplace=True)
data_6['TA_YM']=pd.to_datetime(data_6['TA_YM'],format='%Y%m').dt.to_period('M')
data_6['STDR_YM_CD']=pd.to_datetime(data_6['STDR_YM_CD'],format='%Y%m').dt.to_period('M')
data_6['STDR_YY_CD']=pd.to_datetime(data_6['STDR_YY_CD'],format='%Y').dt.to_period('Y')
data_6=data_6[data_6['TA_YM'].isin(data_6['STDR_YM_CD']) & data_6['STDR_YY_CD'].dt.year.isin(data_6['STDR_YM_CD'].dt.year)]
data_6=data_6.set_index(['BLCK_CD','TA_YM'])
data_6=data_6.set_index(data_6.groupby(level=[0,1]).cumcount(), append=True)
data_6
#데이터 값 확인해보고 쓸데없는 값(00이나 NAN으로 나오는 값들 드랍해야됨)
```

Out[24]:

BLCK_CD	TA_YM		KSIC_CD	MCT_CNT	AMT	CNT	MIN_AMT	MIN_CNT	MAX_AMT	MAX_CNT	AMT_P	CNT_P	...	XCNTS_MIN_VALUE	Y
118	2017-01	0	47122.0	1.0	10334490.0	1867.0	10334490.0	1867.0	10334490.0	1867.0	10042630.0	1853.0	...	NaN	
		1	56111.0	1.0	1287000.0	33.0	1287000.0	33.0	1287000.0	33.0	1287000.0	33.0	...	NaN	
		2	56191.0	1.0	1983000.0	9.0	1983000.0	9.0	1983000.0	9.0	1983000.0	9.0	...	NaN	
		3	56219.0	1.0	1990400.0	82.0	1990400.0	82.0	1990400.0	82.0	1990400.0	82.0	...	NaN	
		4	56220.0	1.0	2383000.0	218.0	2383000.0	218.0	2383000.0	218.0	2213200.0	209.0	...	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
33564	2017-01	29	47420.0	1.0	470000.0	5.0	470000.0	5.0	470000.0	5.0	470000.0	5.0	...	NaN	
		30	47851.0	2.0	1817500.0	6.0	500.0	1.0	1817000.0	5.0	1817500.0	6.0	...	NaN	
		31	56194.0	1.0	4579600.0	448.0	4579600.0	448.0	4579600.0	448.0	4033800.0	417.0	...	NaN	
		32	56199.0	1.0	561000.0	13.0	561000.0	13.0	561000.0	13.0	462000.0	11.0	...	NaN	
		33	56219.0	1.0	4428500.0	74.0	4428500.0	74.0	4428500.0	74.0	4268000.0	71.0	...	NaN	

블록코드의 수에 따른 값들이 매우 많으므로 블록코드에 따라 데이터를 묶어 정렬한 이후,
시계열 분석을 위하여 블록코드에 따라 묶인 데이터를 다시 한 번 기준년도(TA_YM)을 기준으로
당 해/분기별로 묶어서 정렬함.

데이터 분석

In [25]:
```
result=smf.ols(formula='AMT ~ TOT_FLPOP_CO', data=data_6).fit()
result.summary()
#...~다중회귀분석해서 상관성 파악. 매출액 결정 요인 분석
#TOT_FLPOP_CO,GTN_AVRG, CNVRSN_RNTCHRG <- 변수 나중에 넣어볼것
#회귀분석
```

In [26]:
```
y, X = dmatrices("AMT ~ TOT_FLPOP_CO", data=data_6, return_type = "dataframe")
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["features"] = X.columns
```

In [27]:
```
#시계열분석1
columns=['KSIC_CD', 'MCT_CNT', 'CNT', 'MIN_AMT', 'MIN_CNT', 'MAX_AMT',
       'MAX_CNT', 'STDR_YM_CD', 'ML_FLPOP_CO', 'FML_FLPOP_CO',
       'AGRDE_10_FLPOP_CO', 'AGRDE_20_FLPOP_CO', 'AGRDE_30_FLPOP_CO',
       'AGRDE_40_FLPOP_CO', 'AGRDE_50_FLPOP_CO', 'AGRDE_60_ABOVE_FLPOP_CO',
       'TMZON_1_FLPOP_CO', 'TMZON_2_FLPOP_CO', 'TMZON_3_FLPOP_CO',
       'TMZON_4_FLPOP_CO', 'TMZON_5_FLPOP_CO', 'TMZON_6_FLPOP_CO', 'ADSTRD_CD',
       'STDR_YY_CD', 'GTN_AVRG', 'MT_RNTCHRG_AVRG', 'RENT_CO',
        'BLCK_NM'] #연관있는것 제외하고 드랍
data_6.drop(columns, axis=1, inplace=True)
```

회귀 분석을 통해서 매출액과 연관이 있는 데이터를 찾아냄. 이후 매출액과 관련성이 낮은 데이터를 가지는 칼럼들을 모두 드랍함.

데이터 분석

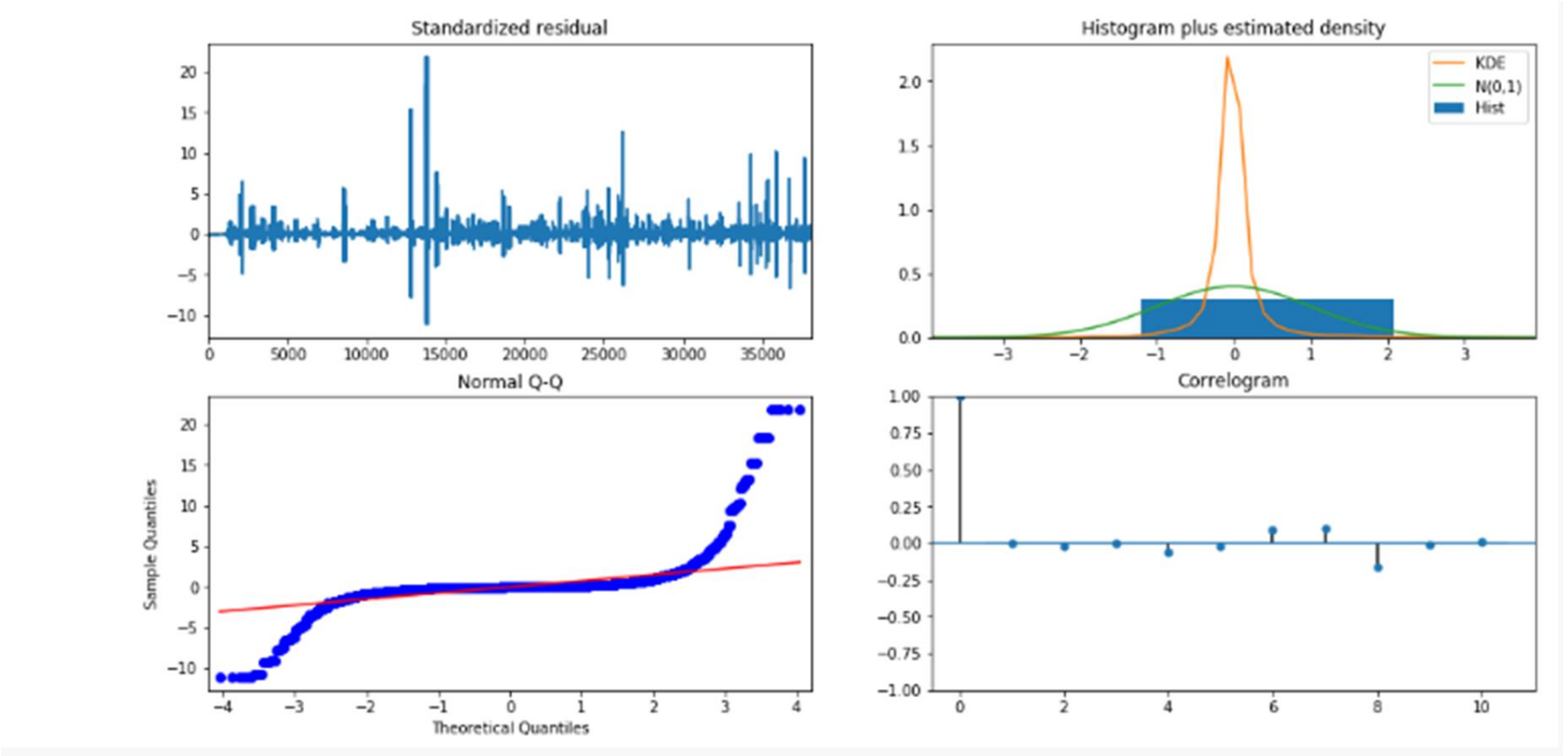
In [45]:
```
#시계열분석2
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
```

In [48]:
```
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(data_6['AMT'],
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)
            results = mod.fit()
            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
        except:
            continue
```

In [49]:
```
mod = sm.tsa.statespace.SARIMAX(data_6['AMT'],
                                order=(1, 1, 1),
                                seasonal_order=(1, 1, 0, 12), #위의 블럭에서 최저 AIC값인 것 찾기
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results = mod.fit()
```

시계열 분석에서 ARIMA 모델을 사용하고, 이때의 파라미터 p(계절성),d(추세),q(노이즈) 값을 구한 후에 모형을 구축함

데이터 분석

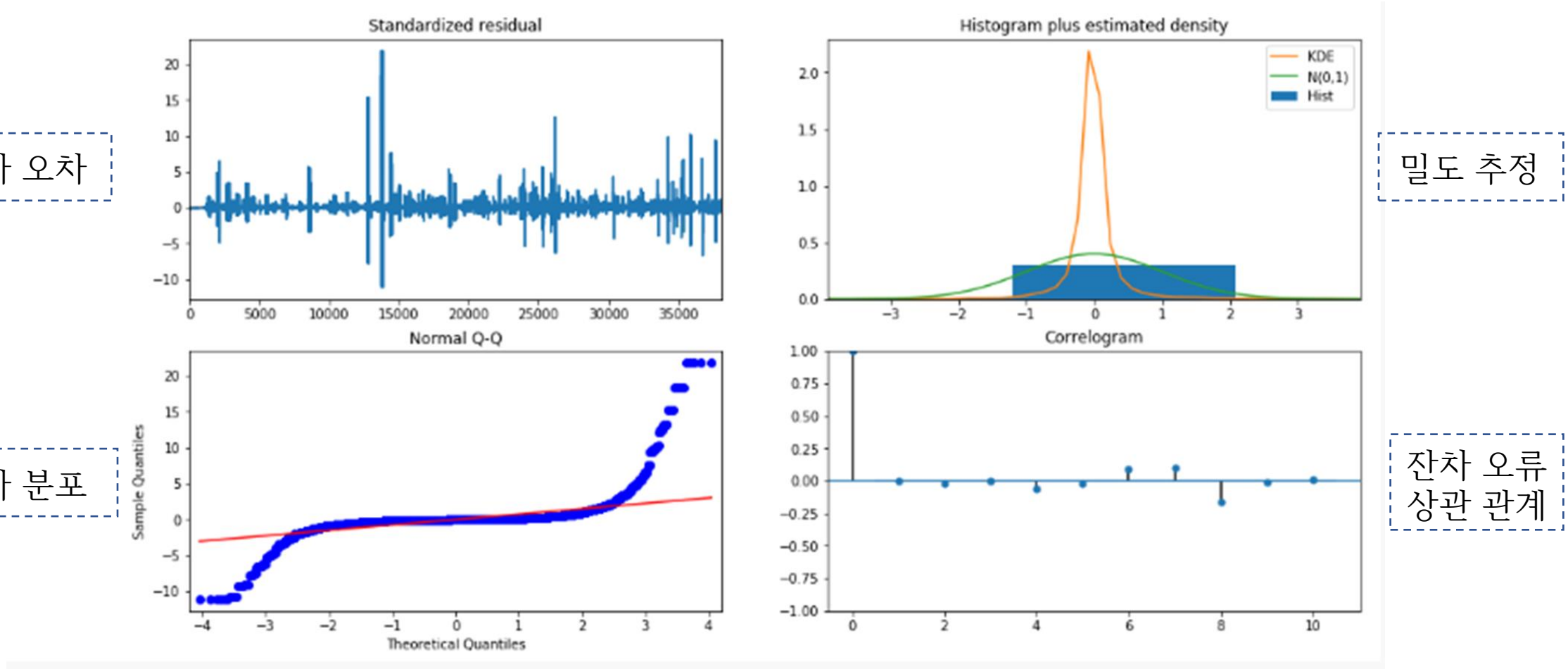


잔차 오차, 오류 등에 대한 그래프들로 오차의 정도에 대해 알 수 있음

데이터 분석

잔차 오차

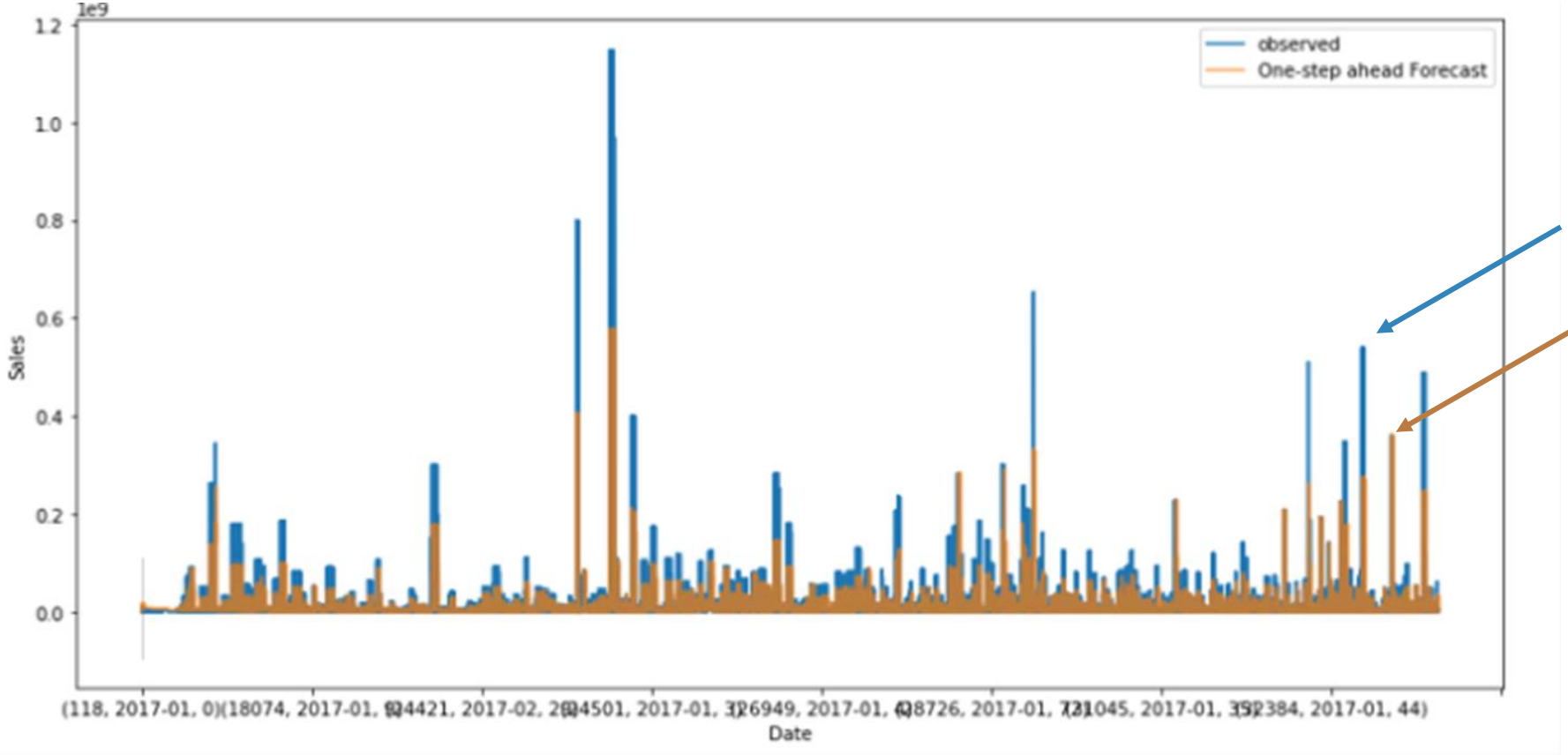
편차 분포



밀도 추정

잔차 오류
상관 관계

데이터 분석



실제 그래프

예측 그래프

매출액의 예측값과 실제값을 함께 나타낸 그래프

결론

- 회귀 분석을 바탕으로 매출액과 연관이 있는 데이터를 찾아내고, 이후 시계열 분석을 통하여 매출액을 예측하고 그를 실제 값과 비교해보는 작업을 진행하였다. 본 분석에서는 약간의 오차가 발생하였으나, 좀 더 세밀한 전처리 과정들을 거치면 충분히 유의미한 결과값이 나올 것으로 생각된다.

- 이미 있는 데이터들을 바탕으로 분석하는 것을 넘어서서 시계열 분석을 통하여 값을 예측하는 새로운 관점을 제시한 것에서 본 분석은 의미 있을 것으로 보인다. 이를 활용하여 다양한 서비스를 제공할 수 있을 것이다.

제안

- 데이터 처리 시간의 문제로 많은 양의 데이터를 다루지 못했으나 전체의 데이터를 다루면 좀 더 유의미한 값이 나올 것으로 예상된다. 또한 본 분석에서는 블록코드를 기준으로 값을 정렬하여 데이터를 분석하였으나 업종분류에 따라 업종별 매출 예측을 진행하는 것도 소상공인들에게 도움이 될 것으로 보인다.

- 시계열 분석을 이용하여 예측한 매출을 바탕으로 임대료, 업종 등의 고려사항을 바탕으로 분류까지 진행하면 소상공인이 입지 선정을 할 때에 도움을 줄 수 있을 것이다. 특히 소상공인의 창업 예산, 원하는 지역, 업종 등을 입력 받아 최적의 위치를 추천해주는 맞춤 서비스를 제공하는 방법으로의 발전도 가능할 것으로 보인다.