고양시 공공자전거 스테이션 최적 위치 선정

Succez 팀

곽동명, 김준수, 염성현, 이소연

1. 자전거 스테이션 데이터 EDA 과정 설명

자전거 스테이션 데이터 EDA 과정 설명

운영 이력 및 자전거 스테이션 현황 분석.ipynb(1/2)

01.운영이력_ver1(한글컬럼, 이용거리).csv 파일을 만들어서 추후에 월별 이동 횟수를 구할 때 사용을 한다.

로드 파일
01.운영 이력.csv
02.자전거 스테이션.csv

운영 이력 및 bike_Station 현황 분석

자전거 스테이션 이용 시간
•데이터 컬럼의 'RTN_DATE'과 'LEAS_DATE' 차이를 통해 이용 시간을 추축해 'Bike_Time(s)' columns를 새로 만들어 준다.
•to_datatime함수를 이용해서 RTN_DATE 컬럼에서 LEAS_DATE 컬럼의 시간을 빼주어 총 이용 시간을 구한다.
•01.운영 이력(Bike_Time(s).csv 가공된 CSV파일을 만들어 추후에 이용 시간이 포함된 데이터 프레임을 함께 사용한다.

자전거 스테이션 이용 현황 파악
•sort_values를 활용하여 운영 이력, 자전거 스테이션에 포함된 대여, 반납, 바이크 스테이션을 개수를 구해준다.
•중복 데이터를 제거하고 차집합을 사용하는 함수를 만들어 중복된 데이터를 제거하여 각 3개의 스테이션간의 특징이 되는 스테이션을 나타내 준다.
•RTN_STATION의 경우 0,999번 스테이션 데이터 프레임을 확인한다.

회원 및 비회원 구분
•MEMB_NO 칼럼을 통해 회원과 비회원으로 나누어 준 다음 앞서 구한 Bike_Tiem(s) 컬럼을 통해 이용 시간의 평균을 구해 준다.
•평균 이용 시간 : 회원 22분 / 비회원 36분
•set 함수를 사용하여 회원 수를 구해준다 (39126명)

자전거 스테이션 데이터 EDA 과정 설명

운영 이력 및 자전거 스테이션 현황 분석.ipynb(2/2)

운영 이력 데이터 컬럼 변경

- 고양시 데이터 컬럼 목록에 나오는 것과 같이 한글 컬럼 이름으로 변경 후 csv파일 저장

LEAS_NO : 대여 번호

LEAS_DATE : 대여 시간

LEAS_STATION : 대여 스테이션 번호

LEAS_DEF_NO : 대여 거치 대 번호

RTN_DATE : 반납 시간

RTN_STATION : 반납 스테이션 번호

RTN_DEF_NO : 반납 거치 대 번호

TRNV_QTY : 추정 이동 거리

MEMB_DIV : 회원 구분(비회원은 99이며 나머지는 정회원)

MEMB_NO : 회원 번호

TEMP_MEMB_NO : 비회원 번호

BIKE_TAG : 자전거 번호

Bike_Time(s) : 이용 시간(s)

01.운영이력_ver1(한글컬럼, 이용거리).csv 저장

자전거 스테이션 데이터 EDA 과정 설명

자전거 이용 시간 및 시간대 이용 횟수.ipynb

로드 파일
01운영이력.csv

전체 이용 현황
RTN_DATE 시간에서 LEAS_DATE 시간을 빼주어 Bike_Time(s) 컬럼을 만들어 준다.
시간대 별로 나타내기 위해 ex)'00',11' 예제와 같이 LEAS_DATE,RTN_DATE 칼럼의 데이터를 가공하여 준다.

3년간 시간 별 자전거 스테이션 이용 현황

회원 이용 현황
가공한 데이터에서 회원들만 뽑아주는 새로운데이터 프레임을 만들어 준다.

3년간 시간별 자전거 스테이션 이용 현황
전체, 회원을 더해주는 반복 문을 만들어 '시간별 자전거 이용 횟수' 컬럼과 시간별 '회원 자전거 이용 횟수' 컬럼을 만들어 준다.
만들어준 데이터 프레임에 '시간별 자전거 이용 횟수', '회원 자전거 이용 횟수' 컬럼의 수를 빼서 '비회원 자전거 이용 횟수 ' 를 추가하여 하나의 데이터 프레임 만들어 준다.

시간 단위를 기준으로 스테이션 이용 현황

groupby와 sum을 사용하여 시간별 대여와 반납 스테이션의 이용 현황을 만들어 준다.

전체 시간별 이용 횟수 데이터 시각화
시간대별 자전거 이용 횟수를 ggplot을 통해 시각화 해 주고 3/4분위값과 평균값을 라인으로 그어준다.
빨간색 – 평균 / 파란색 – 3/4분위 값

자전거 스테이션 데이터 EDA 과정 설명

회원들의 자전거 이용 분석.ipynb

전체이용, 회원, 비회원들의 월별 자전거 이용 횟수를 나타내고 시각화 한다.

로드 파일
01.운영이력.csv

자전거 운영 이력 csv파일을 불러 온다.
LEAS_DATE,RTN_DATE 컬럼의 apply(lambda)릉 통해 각 년도의 월별로 컬럼 내용을 변경하여 준다.
중요하다고 생각되는 컬럼을 뽑아내서 따로 데이터 프레임을 만들어 준다
MEMB_NO컬럼의 내용에서 0인것은 비회원 0이 아닌 행들은 회원으로 구분하여 준다.

3년간 월별로 일반 전체, 회원, 비회원의 자전거 이용 횟수를 구해 준다.
2017,2018,2019로 나누어 월별로 스테이션 이용 현황을 구해준다.
년도 별로 이용현황을 전체, 회원, 비회원으로 나누어 시각화 해준다.

회원 증가 추세 조사
set 함수를 사용한다.
현재 총 회원수에 저번 달 총 회원수를 빼주어 월별 회원수 증가를 구해 준다.
데이터를 데이터 프레임으로 만들어 준다.
연도별 회원 증가 추세를 시각화 하여 보여 준다.

자전거 스테이션 데이터 EDA 과정 설명

월별 스테이션 이용 횟수.ipynb(1/2)

파일 로드
01.운영이력_ver1(한글컬럼, 이용거리).csv – 운영 이력을 통해 가공한 데이터
02.자전거스테이션.csv

월별 스테이션 이용 횟수
3년간 월별 이용 횟수를 구하기 위해 01.운영이력_ver1(한글컬럼, 이용거리).csv ' 파일을 불러온다.
대여 시간, 반납 시간을 년–월로 데이터를 가공한다.
필요한 컬럼만 가져와서 새롭게 데이터 프레임을 만들어 준다.

스테이션 칼럼 만들 및 대여 반납 이용 횟수 데이터 프레임 만들기
리스트를 데이터 프레임으로 만들어주고 3년간 월별 데이터를 합쳐준다.
전체 이용과 회원들의 월간 대여 및 반납 이용 횟수를 구해준다.
반복문의 경우 3년치의 월로 구분된 날이 들어가 있고(ex.2017–01.2017–02), 다른 하나의 경우 대여 및 반납 스테이션을 카운팅하여 더해주는 반복문을 만들어서 구해준다.
'전체 이용'과 '회원의 이용 횟수'를 구해 준 후에 두개의 컬럼 내용들의 숫자를 빼서 새로운' 비회원 이용 횟수' 컬럼을 만들어 준다.
대여와 반납 같은 방식으로 구해준다.

haversine 함수를 통해 직선거리를 구해준다.
02.자전거스테이션.csv 을 통해 스테이션 간의 거리를 구해준다.
하나의 스테이션이 모든 스테이션의 거리를 구할 수 있도록 하며 harversine 함수를 활용하여 직선거리를 구해주며, 단위는 미터(m)로 설정한다.
각 스테이션별 직선거리를 데이터 프레임으로 만들어 준다.

자전거 스테이션 데이터 EDA 과정 설명

월별 스테이션 이용 횟수.ipynb(2/2)

총 스테이션 이용 횟수

대여 스테이션 이용 횟수
3년간 대여 스테이션의 이용 횟수를 구한다.
groupby,count 함수를 사용하여 '대여 스테이션' 카운팅하여 스테이션별 이용 횟수를 구해준다.
회원들의 이용 횟수도 같은 방식으로 구해준다.
assign,lambda 함수를 통해 '전체 인원 이용횟수 – 회원 이용 횟수' 빼줘 비회원의 이용 횟수를 구해준다.

대여 스테이션 이용 횟수
대여 스테이션 이용 횟수와 같은 방식으로 진행이 되며, 다른 점은 '대여 스테이션 ‘ 이 아닌 '반납 스테이션 ‘ 을 카운팅 해준다.

시각화

대여 스테이션 시각화
ggplot을 통해 3년간 대여, 3/4분위 값 기준, 1/4분위 값 기준으로 대여 스테이션 이용 횟수를 시각화 해 준다.

반납 스테이션 시각화
ggplot을 통해 3년간 반납 스테이션 이용 현황을 구해준다.
그래프 이용 횟수를 '반납 스테이션 0'인 경우를 제외하여 그래프를 다시 시각화 해준다.
'반납 스테이션 0'경우를 제외하고 3년간 반납, 3/4분위 값 기준, 1/4분위 값 기준으로 반납 스테이션 이용 횟수를 시각화 해 준다.

자전거 스테이션 데이터 EDA 과정 설명

자전거 일별 대여 스테이션 이용 현황.ipynb(1/4)

로드 파일
01. 운영이력(Bike_Time(s).csv –이용 시간이 가공된 운영 이력 데이터를 불러 온다.
02.자전거 스테이션.csv 자전거 스테이션을 2번 로드 하여 대여, 반납 기준으로 나누어서 데이터를 정제, 가공하여 준다.
3년간 스테이션 별 자전거 대여 스테이션 이용 현황.csv

운영 이력과 데이터 로드
'LEAS_STAT' 칼럼을 지워준다.
칼럼 목록을 한글로 변환시켜 준다.
바이크 스테이션 대여 및 반납의 칼럼명을 변경 하고 운영 이력과 변경된 칼럼명을 가진 바이크 스테이션을
merge하여 새로운 데이터 프레임을 생성하여 준다.

일별 데이터 구하기(데이터 가공)
대여 시간의 데이터를 년–월–일로 가공하여 '대여 시간 '의 컬럼 내용을 변경한다.

년도별 데이터 구하기(데이터 가공)
대여 시간의 데이터를 년으로 가공하여 '대여 시간 ‘ 의 컬럼 내용을 변경한다.

대여 스테이션 이용 횟수를 기준으로 필요한 feature 들을 구해준다.

일별 대여스테이션 이용 횟수
'대여 시간', '대여 스테이션 번호' 컬럼을 groupby, count를 통해 대여 시간이 '일' 기준으로 대여 스테이션의 이용 횟수를 구해준다.
Pivot 함수를 통해 데이터를 펼쳐준다.

자전거 스테이션 데이터 EDA 과정 설명

자전거 일별 대여 스테이션 이용 현황.ipynb(2/4)

스테이션별 대여 스테이션 최대 이용 횟수 및 최대 이용 날짜
'top ‘ 이라는 이름을 가진 함수를 만들어 데이터 프레임의 오름차순의 가장 높은 값을 뽑아준다.
groupby를 통해 'top'을 apply에 적용 시켜준다.
컬럼 내용을 정제하여 필요한 컬럼을 뽑아 데이터 프레임을 만들어 준다.

스테이션 별 대여 스테이션 최소 이용 횟수 및 최소 이용 날짜
'bottom ‘ 이라는 이름을 가진 함수를 만들어 데이터 프레임의 내림차순의 가장 낮은 값을 뽑아준다.
groupby를 통해 'bottom'을 apply에 적용 시켜준다.
필요한 컬럼을 추출하여 최소 이용일과 최소 스테이션이 이용 횟수가 1개가 나오는 데이터 프레임을 생성하여 준다.
컬럼 내용을 정제하여 필요한 컬럼을 뽑아 데이터 프레임을 만들어 준다.

스테이션 별 median() 중간값
'median'이라는 이름을 가진 함수를 만들어 데이터 프레임의 중간값을 뽑아준다.
groupby를 통해 'median'을 apply에 적용 시켜준다.
필요한 컬럼을 추출하여 중간값의 스테이션 이용 횟수 나오는 데이터 프레임을 생성하여 준다.
컬럼내 용을 정제하여 필요한 컬럼을 뽑아 데이터 프레임을 만들어 준다.

스테이션 별 1/4분위 값
'quantile_25'이라는 이름을 가진 함수를 만들어 데이터 프레임의 $\frac{1}{4}$분위을 뽑아준다.
groupby를 통해 'quantile_25'을 apply에 적용 시켜준다.
필요한 컬럼을 추출하여 quantile(.25) 이용 횟수가 나오는 데이터 프레임을 생성하여 준다.
컬럼 내용을 정제하여 필요한 컬럼을 뽑아 데이터 프레임을 만들어 준다.

자전거 스테이션 데이터 EDA 과정 설명

자전거 일별 대여 스테이션 이용 현황.ipynb(3/4)

스테이션 별 3/4분위 값

'quantile_75'이라는 이름을 가진 함수를 만들어 데이터 프레임의 3/4분위 값을 뽑아준다.

groupby를 통해 'quantile_75'을 apply에 적용 시켜준다.

필요한 컬럼을 추출하여 quantile(.75) 이용 횟수가 나오는 데이터 프레임을 생성하여 준다.

컬럼 내용을 정제하여 필요한 컬럼을 뽑아 데이터 프레임을 만들어 준다.

스테이션 평균

'mean'이라는 이름을 가진 함수를 만들어 데이터 프레임의 평균값을 뽑아준다.

groupby를 통해 'mean'을 apply에 적용 시켜준다.

필요한 컬럼을 추출하여 mean값의 이용 횟수가 나오는 데이터 프레임을 생성하여 준다.

컬럼 내용을 정제하여 필요한 컬럼을 뽑아 데이터 프레임을 만들어 준다.

스테이션 기준으로 3년간 대여 스테이션 이용 횟수

년도별로 정제된 데이터를 통해 3년간 대여 스테이션 이용 횟수를 추출한다.

년도별로 카운팅 한것중에 행을 나누어 주어 2017,2018,2019로 구분 지어준다.

groupby,count를 통해 갯수를 구해준다.

컬럼 내용을 정제하여 2017,2018,2019 데이터를 뽑아 데이터 프레임을 만들어 준다.

자전거 스테이션 데이터 EDA 과정 설명

자전거 일별 대여 스테이션 이용 현황.ipynb(4/4)

스테이션 별 평균 추정 이동 거리 및 평균 이용 시간

추정이동거리를 구하기 위해 3가지 필요한 컬럼을 가져오고 기존 데이터의 NaN값을 제거하여 준다.
데이터중 NaN값을 제외한 0값도 존재하기에 같이 제거해준다.
660148 행을 가진 데이터의 일별 추정이동거리를 groupby,mean을 통해 구해 준다.
'mean' 이라는 평균값을 출력하는 함수를 만들어 groupby를 통해 'mean'을 apply에 적용 시켜준다.
컬럼 내용을 정제하여 필요한 컬럼을 뽑아 평균 추정 이동거리 데이터 프레임을 만들어 준다.
평균 이용 시간을 구하기 위해 '이용 시간(s)'을 활용하여 필요한 칼럼을 가져온다.
평균 이용 시간을 'mean' 이라는 평균값을 출력하는 함수를 만들어 groupby를 통해 'mean'을 apply에 적용 시켜준다.
평균이용시간의 컬럼의 내용을 60으로 나누어 초에서 분 단위로 나타내어지는 컬럼을 만든다.

회원 비회원 비율 현황

3년간 스테이션 별 자전거 대여 스테이션 이용 현황.csv로드하여 자전거의 렌트 스테이션 현황과 data merge해준다.
assign,lambda를 통해 '회원비율', '비회원비율'이 나타나도록 컬럼을 만들어 준다.
컬럼내용을 정제하여 필요한 컬럼을 뽑아 데이터 프레임을 만들어 준다.

데이터 merge

전체적으로 구한 데이터를 merge하여 22columns이 생성된 데이터 프레임을 만들어 준다.

자전거 스테이션 데이터 EDA 과정 설명

자전거 일별 반납 스테이션 이용 현황.ipynb(1/4)

로드 파일
01. 운영이력(Bike_Time(s).csv –이용 시간이 가공된 운영 이력 데이터를 불러 온다.
02.자전거 스테이션.csv 자전거 스테이션을 2번 로드 하여 대여, 반납 기준으로 나누어서 데이터를 정제, 가공하여 준다.
3년간 스테이션 별 자전거 반납 스테이션 이용 현황.csv

운영 이력과 데이터 로드
'LEAS_STAT' 칼럼을 지워준다.
칼럼 목록을 한글로 변환시켜 준다.
바이크 스테이션 대여 및 반납의 칼럼명을 변경 하고 운영 이력과 변경된 칼럼명을 가진 바이크 스테이션을
merge하여 새로운 데이터 프레임을 생성하여 준다.

일별 데이터 구하기(데이터 가공)
대여 시간의 데이터를 년–월–일로 가공하여 '대여 시간 '의 컬럼 내용을 변경한다.

년도별 데이터 구하기(데이터 가공)
대여 시간의 데이터를 년으로 가공하여 '대여 시간 ‘ 의 컬럼 내용을 변경한다.

대여 스테이션 이용 횟수를 기준으로 필요한 feature 들을 구해준다.

일별 대여스테이션 이용 횟수
'대여 시간', ' 반납 스테이션 번호' 컬럼을 groupby, count를 통해 대여 시간이 '일' 기준으로 반납 스테이션의 이용 횟수를 구해준다.
Pivot 함수를 통해 데이터를 펼쳐준다.

자전거 스테이션 데이터 EDA 과정 설명

자전거 일별 반납 스테이션 이용 현황.ipynb(2/4)

스테이션별 반납 스테이션 최대 이용 횟수 및 최대 이용 날짜
' top ' 이라는 이름을 가진 함수를 만들어 데이터 프레임의 오름차순의 가장 높은 값을 뽑아준다.
Groupby를 통해 ' top ' 을 apply에 적용 시켜준다.
컬럼 내용을 정제하여 필요한 컬럼을 뽑아 데이터 프레임을 만들어 준다.

스테이션 별 반납 스테이션 최소 이용 횟수 및 최소 이용 날짜
'bottom ' 이라는 이름을 가진 함수를 만들어 데이터 프레임의 내림차순의 가장 낮은 값을 뽑아준다.
groupby를 통해 'bottom'을 apply에 적용 시켜준다.
필요한 컬럼을 추출하여 최소 이용일과 최소 스테이션이 이용 횟수가 1개가 나오는 데이터 프레임을 생성하여 준다.
컬럼 내용을 정제하여 필요한 컬럼을 뽑아 데이터 프레임을 만들어 준다.

스테이션 별 median() 중간값
'median'이라는 이름을 가진 함수를 만들어 데이터 프레임의 중간값을 뽑아준다.
groupby를 통해 'median'을 apply에 적용 시켜준다.
필요한 컬럼을 추출하여 중간값의 스테이션 이용 횟수 나오는 데이터 프레임을 생성하여 준다.
컬럼내 용을 정제하여 필요한 컬럼을 뽑아 데이터 프레임을 만들어 준다.

스테이션 별 1/4분위 값
'quantile_25'이라는 이름을 가진 함수를 만들어 데이터 프레임의 1/4분위 값을 뽑아준다.
groupby를 통해 'quantile_25'을 apply에 적용 시켜준다.
필요한 컬럼을 추출하여 quantile(.25) 이용 횟수가 나오는 데이터 프레임을 생성하여 준다.
컬럼 내용을 정제하여 필요한 컬럼을 뽑아 데이터 프레임을 만들어 준다.

자전거 스테이션 데이터 EDA 과정 설명

자전거 일별 반납 스테이션 이용 현황.ipynb(3/4)

스테이션 별 3/4분위 값

'quantile_75'이라는 이름을 가진 함수를 만들어 데이터 프레임의 3/4분위 값을 뽑아준다.

groupby를 통해 'quantile_75'을 apply에 적용 시켜준다.

필요한 컬럼을 추출하여 quantile(.75) 이용 횟수가 나오는 데이터 프레임을 생성하여 준다.

컬럼 내용을 정제하여 필요한 컬럼을 뽑아 데이터 프레임을 만들어 준다.

스테이션 평균

'mean'이라는 이름을 가진 함수를 만들어 데이터 프레임의 평균값을 뽑아준다.

groupby를 통해 'mean'을 apply에 적용 시켜준다.

필요한 컬럼을 추출하여 mean값의 이용 횟수가 나오는 데이터 프레임을 생성하여 준다.

컬럼 내용을 정제하여 필요한 컬럼을 뽑아 데이터 프레임을 만들어 준다.

스테이션 기준으로 3년간 대여 스테이션 이용 횟수

년도 별로 정제된 데이터를 통해 3년간 대여 스테이션 이용 횟수를 추출한다.

년도 별로 카운팅 한 것중에 행을 나누어 주어 2017,2018,2019로 구분 지어준다.

groupby,count를 통해 갯수를 구해준다.

컬럼 내용을 정제하여 2017,2018,2019 데이터를 뽑아 데이터 프레임을 만들어 준다.

자전거 스테이션 데이터 EDA 과정 설명

자전거 일별 반납 스테이션 이용 현황.ipynb(4/4)

스테이션 별 평균 추정 이동 거리 및 평균 이용 시간

평균 이용 시간을 구하기 위해 '이용 시간(s)'을 활용하여 필요한 칼럼을 가져온다.

평균 이용 시간을 'mean' 이라는 평균값을 출력하는 함수를 만들어 groupby를 통해 'mean'을 apply에 적용 시켜준다.

평균이용시간의 컬럼의 내용을 60으로 나누어 초에서 분 단위로 나타내어지는 컬럼을 만든다.

추정이동거리를 구하기 위해 3가지 필요한 컬럼을 가져오고 기존 데이터의 NaN값을 제거하여 준다.

데이터중 NaN값을 제외한 0값도 존재하기에 같이 제거해준다.

660148 행을 가진 데이터의 일별 추정이동거리를 groupby,mean을 통해 구해 준다.

'mean' 이라는 평균값을 출력하는 함수를 만들어 groupby를 통해 'mean'을 apply에 적용 시켜준다.

컬럼 내용을 정제하여 필요한 컬럼을 뽑아 평균 추정 이동거리 데이터 프레임을 만들어 준다.

회원 비회원 비율 현황

3년간 스테이션 별 자전거 대여 스테이션 이용 현황.csv로드하여 자전거의 렌트 스테이션 현황과 data merge해준다.

assign,lambda를 통해 '회원비율', '비회원비율'이 나타나도록 컬럼을 만들어 준다.

컬럼내용을 정제하여 필요한 컬럼을 뽑아 데이터 프레임을 만들어 준다.

데이터 merge

전체적으로 구한 데이터를 merge하여 22columns이 생성된 데이터 프레임을 만들어 준다.

2. 도시 계획과 실제 데이터 포함 여부 확인

도시 계획과 실제 데이터 포함 여부 확인

도시계획과 실제 데이터 포함여부 확인.ipynb(1/7)

로드 파일

02.자전거스테이션.csv
08.행정경계(시군구).geojson
10.도시계획(공간시설).geojson
11.도시계획(공공문화체육시설).geojson
12.도시계획(교통시설).geojson
13.용도지역지구(습지보호지역).gsojson
15.도로명주소*건물.geojson*
16.도로명주소_도로.geojson
17.일반건물_분포도(100MX100M)geojson
18.행사장_공간정보.*csv*
19.전철역_공간정보.csv
20.고양시_버스정류소.csv
22.주차장정보.csv
26.고양시 공연장 박물관 정보.csv
27.고양시 체육시설 현황 정보.csv
가공 데이터 : 도로명주소건물.pickle(build_road_plus) – 기존 코드정의서 함께 합친 데이터 프레임을 가져온다.

도시 계획과 실제 데이터 포함 여부 확인

도시계획과 실제 데이터 포함여부 확인.ipynb (2/7)

코드 작성 내용

기존 계획 데이터

10.도시계획(공간시설).geojson
- 새로운 컬럼 생성('분류','UQT_m','라벨') : 파일에 칼럼 생성(여러 데이터들이 겹침을 방지하기 위해)
  - 분류 – '도시계획(공간시설)'로 분류 column 내용 생성
  - UQT_m – 용도지역지구구분(6자리) 를 구분으로 데이터를 lambda함수로 잘라내서 묶어 주었다.
  - 라벨 – 라벨 값은 nan값으로 미리 설정하였고 기존의 데이터들이 계획 데이터에 포함되면 '5', 아니면 '1'로 설정하여 추후에 라벨값을 정한다.
    - 라벨 5 – 현행 공간정보를 기준으로 포함된 도시계획 데이터
    - 라벨 1 – 현행 공간정보를 기준으로 포함되지 않은 도시계획 데이터

11.도시계획(공공문화체육시설).geojson
- 새로운 컬럼 생성('분류','라벨') : 파일에 칼럼 생성(여러 데이터들이 겹침을 방지하기 위해)
  - 분류 – '도시계획(공공문화체육시설)'로 분류 column 내용 생성
  - 라벨 – 라벨 값은 nan값으로 미리 설정하였고 기존의 데이터들이 계획 데이터에 포함되면 '5', 아니면 '1'로 설정하여 추후에 라벨값을 정한다.
    - 라벨 5 – 현행 공간정보를 기준으로 포함된 도시계획 데이터
    - 라벨 1 – 현행 공간정보를 기준으로 포함되지 않은 도시계획 데이터

12.도시계획(교통시설).geojson
- 새로운 컬럼 생성('분류', '라벨') : 파일에 칼럼 생성(여러 데이터들이 겹침을 방지하기 위해)
  - 분류 – '도시계획(교통시설)'로 분류 column 내용 생성
  - 라벨 – 라벨 값은 nan값으로 미리 설정하였고 기존의 데이터들이 계획 데이터에 포함되면 '5', 아니면 '1'로 설정하여 추후에 라벨값을 정한다.
    - 라벨 5 – 현행 공간정보를 기준으로 포함된 도시계획 데이터
    - 라벨 1 – 현행 공간정보를 기준으로 포함되지 않은 도시계획 데이터

좌표계 확인 : epsg:4236인지 계획 데이터의 좌표를 확인하여 준다.

도시 계획과 실제 데이터 포함 여부 확인

도시계획과 실제 데이터 포함여부 확인.ipynb(3/7)

기존 실제 데이터
가공 데이터 로드 : 도로명주소건물.pickle(build_road_plus) ---build_road_plus.ipynb=코드 구현
to_float이라는 함수를 사용
•'X','Y' 좌표를 geometry 데이터로 전환 후 'geometry' 컬럼에 POINT로 좌표 생성을 변경해주는 함수 작성

18.행사장_공간정보.*csv*
•새로운 컬럼 생성('분류','라벨') : 파일에 칼럼 생성(여러 데이터들이 겹침을 방지하기 위해)
    •분류 – ' 행사장_공간정보'로 분류 column 내용 생성
•컬럼 이름 변경
•to_float함수를 사용해서 geometry' 컬럼에 POINT로 좌표 생성

19.전철역_공간정보.csv
•새로운 컬럼 생성('분류','라벨') : 파일에 칼럼 생성(여러 데이터들이 겹침을 방지하기 위해)
    •분류 – '전철역_공간정보'로 분류 column 내용 생성
•'Name'라는 컬럼을 생성
    •'station_nm','rail_nm' 두개의 컬럼 내용을 합쳐준다.
    •기존 station_nm에서 데이터를 편리하게 사용하기 위해서
•to_float함수를 사용해서 geometry' 컬럼에 POINT로 좌표 생성

22.주차장정보.csv
•새로운 컬럼 생성('분류','라벨') : 파일에 칼럼 생성(여러 데이터들이 겹침을 방지하기 위해)
    •분류 – ' 주차장정보 '로 분류 column 내용 생성
•'Name'라는 컬럼을 생성
    •'parking_nm','space' 두개의 컬럼 내용을 합쳐준다.
    •기존 parking_nm에서 데이터를 편리하게 사용하기 위해서
•to_float함수를 사용해서 geometry' 컬럼에 POINT로 좌표 생성

도시 계획과 실제 데이터 포함 여부 확인

도시계획과 실제 데이터 포함여부 확인.ipynb(4/7)

26.고양시 공연장 박물관 정보.csv

•새로운 컬럼 생성('분류','라벨') : 파일에 칼럼 생성(여러 데이터들이 겹침을 방지하기 위해)

- •분류 – 고양시 공연장 박물관 정보 '로 분류 column 내용 생성

•'Name'라는 컬럼을 생성

- •'place_nm','servicetype'두개의 컬럼 내용을 합쳐준다.
- •기존 place_nm에서 데이터를 편리하게 사용하기 위해서

•to_float함수를 사용해서 geometry' 컬럼에 POINT로 좌표 생성

27.고양시 체육시설 현황 정보.csv

•새로운 컬럼 생성('분류','라벨') : 파일에 칼럼 생성(여러 데이터들이 겹침을 방지하기 위해)

- •분류 – 고양시 체육시설 현황 정보 '로 분류 column 내용 생성

•'Name'라는 컬럼을 생성

- •'FAC_NAME','GBN'두개의 컬럼 내용을 합쳐준다.
- •기존 FAC_NAME에서 데이터를 편리하게 사용하기 위해서

•to_float함수를 사용해서 geometry' 컬럼에 POINT로 좌표 생성

계획 데이터와 같이 위에 있는 데이터를 계획시설의 데이터와 같은 좌표계를 사용하기 위해 설정해준다

도시 계획과 실제 데이터 포함 여부 확인

도시계획과 실제 데이터 포함여부 확인.ipynb(5/7)

계획데이터가 실제 데이터에 포함되어 있는지 확인하기
데이터 확인(실제 데이터)
– build_road_plus / physi_curr / fes_space / subway_station / parking_info / theat_muse

build_road_plus 데이터 확인
•실제 데이터가 아래의 도시계획 데이터에 포함 여부를 통해 실제 하는지 계획되어 있는지 확인을한다.
    •space_plan
    •culphy_plan
    •traffic_plan
•도시 계획 데이터에 nan값으로 들어간 라벨들에 1 값을 매겨준다.
    •라벨='1' 값은 도시계획데이터 안에 실제 데이터가 존재한다는 의미로 현재의 계획중이 아닌 존재하는 실존 건물을 나타낸다.

pyhsi_curr 데이터 확인
•실제 데이터가 아래의 도시계획 데이터에 포함 여부를 통해 실제 하는지 계획되어 있는지 확인을한다.
    •space_plan
    •culphy_plan
    •traffic_plan
•도시 계획 데이터에 nan값으로 들어간 라벨들에 1 값을 매겨준다.
    •라벨='1' 값은 도시계획데이터 안에 실제 데이터가 존재한다는 의미로 현재의 계획 중이 아닌 존재하는 실존 건물을 나타낸다.

Fes_space 데이터 확인
•실제 데이터가 아래의 도시계획 데이터에 포함 여부를 통해 실제 하는지 계획되어 있는지 확인을 한다.
    •space_plan
    •culphy_plan
    •traffic_plan
•도시 계획 데이터에 nan값으로 들어간 라벨들에 1 값을 매겨준다.
    •라벨='1' 값은 도시계획데이터 안에 실제 데이터가 존재한다는 의미로 현재의 계획 중이 아닌 존재하는 실존 건물을 나타낸다.

자전거 스테이션 데이터 EDA 과정 설명

도시계획과 실제 데이터 포함여부 확인.ipynb(6/7)

subway_station 데이터 확인

•실제 데이터가 아래의 도시계획 데이터에 포함 여부를 통해 실제 하는지 계획되어 있는지 확인을 한다.

- •space_plan
- •culphy_plan
- •traffic_plan

•도시 계획 데이터에 nan값으로 들어간 라벨들에 1 값을 매겨준다.

- •라벨='1' 값은 도시계획데이터 안에 실제 데이터가 존재한다는 의미로 현재의 계획중이 아닌 존재하는 실존 건물을 나타낸다.

parking_info 데이터 확인

•실제 데이터가 아래의 도시계획 데이터에 포함 여부를 통해 실제 하는지 계획되어 있는지 확인을 한다.

- •space_plan
- •culphy_plan
- •traffic_plan

•도시 계획 데이터에 nan값으로 들어간 라벨들에 1 값을 매겨준다.

- •라벨='1' 값은 도시계획데이터 안에 실제 데이터가 존재한다는 의미로 현재의 계획 중이 아닌 존재하는 실존 건물을 나타낸다.

도시 계획과 실제 데이터 포함 여부 확인

도시계획과 실제 데이터 포함여부 확인.ipynb(7/7)

theat_muse 데이터 확인

- 실제 데이터가 아래의 도시계획 데이터에 포함 여부를 통해 실제 하는지 계획되어 있는지 확인을 한다.
    - space_plan
    - culphy_plan
    - traffic_plan
- 도시 계획 데이터에 nan값으로 들어간 라벨들에 1 값을 매겨준다.
    - 라벨='1' 값은 도시계획데이터 안에 실제 데이터가 존재한다는 의미로 현재의 계획 중이 아닌 존재하는 실존 건물을 나타낸다.

nan값인 라벨 데이터에 5를 붙여준다.(계획도시 데이터에 계획되어 있다.)

- 라벨이 5인 데이터를 확인하기 위해 space_plan / culphy_plan / traffic_plan(공간 계획 데이터)의 라벨 1과 5로 분리해준다.
- geometry.centroid를 통해 geometry의 'X','Y'값을 입력해준다.
    - folium 시각화 자료에 사용하기 위해서
        - 시각화 자료는 도시계획과 실제 데이터 포함여부 확인 폴더에서 확인하면 된다.

도시 계획과 실제 데이터 포함 여부 확인

buide_road_plus.ipynb

도시계획과 실제 데이터 포함여부 확인.ipynb에서 도로명건물.pickle 파일을 사용하기 pickle file을 만들어 주고 첨부해 두었다.

로드 파일

15.도로명주소_건물.geojson

28.코드정의서.xlsx

코드 작성 내용

•28.코드정의서.xlsx에서 BDTYP_CD가 포함된 데이터 프레임을 가져오기 위해 28.코드정의서.xlsx의 구간을 나누어 주었다.(BDTYP_CD_meaning)

•15.도로명주소_건물.geojson는 MULTIPOLYGON공간정보 이기에 좌표를 찍기위해 geometry.centroid를 활용하여 공간의 중간에 POITN좌표를 찍어준다.

•geometry_Point라는 컬럼에 POITN좌표를 찍어준다

•기존에 가공한 코드정의서 데이터와 좌표 가공한 도로명주소_건물 데이터를 합쳐준다.

•합쳐준 데이터에 geometry의 NaN 값의 행을 제거해 주고 마지막에 '코드값 의미'라는 컬럼을 만들어 주었다.

•BDTYP_CD와 코드값 의미의 컬럼의 내용을 합쳐주기 위해 apply(lambda)를 통해 새로운 컬럼을 만들어 주고 기존 컬럼의 내용을 합쳐준다.

•완성된 데이터 프레임은 pickle파일로 '도로명주소_건물'이라는 이름으로 저장후 도시계획과 실제 데이터 포함여부 확인.ipynb에 불러와서 사용하여 준다.

3. 고양시 동 별 자전거 스테이션 현황

고양시 동 별 자전거 스테이션 현황

동별 자전거 스테이션 포함 여부

```
[12]  1 ## 동별 스테이션 포함 여부를 나타낸다.
      2 for i in Bike_station.index:
      3   for j in Koyang_dong.index:
      4     if Bike_station.geometry[i].within(Koyang_dong.geometry[j]):
      5       print('{}는 {}에 속해있다.'.format(Bike_station.Station_ID[i],Koyang_dong.EMD_KOR_NM[j]))
      6
      7
```

```
112는 화정동에 속해있다.
113는 화정동에 속해있다.
114는 화정동에 속해있다.
115는 화정동에 속해있다.
116는 화정동에 속해있다.
118는 화정동에 속해있다.
119는 화정동에 속해있다.
121는 화정동에 속해있다.
123는 토당동에 속해있다.
124는 토당동에 속해있다.
125는 화정동에 속해있다.
126는 화정동에 속해있다.
127는 행신동에 속해있다.
128는 행신동에 속해있다.
129는 행신동에 속해있다.
130는 행신동에 속해있다.
131는 토당동에 속해있다.
133는 토당동에 속해있다.
137는 행신동에 속해있다.
138는 행신동에 속해있다.
```

행정경계(읍면동) 데이터에 자전거 스테이션이 위치가 포함되는지 여부를 구해준다.

고양시 동 별 자전거 스테이션 현황

```
[18]  1 # 동별로 나누기 위해 확인
      2 df_B_in_Ko_merge_Bike['EMD_KOR_NM'].unique()
```

```
array(['성사동', '탄현동', '화정동', '토당동', '행신동', '주교동', '향동동', '덕은동', '원흥동',
       '삼송동', '신원동', '동산동', '도내동', '백석동', '장항동', '마두동', '가좌동', '정발산동',
       '풍동', '식사동', '중산동', '일산동', '대화동', '주엽동', '덕이동'], dtype=object)
```

52개의 동 중에 총 25개의 동에 자전거 스테이션 존재함을 확인

```
[16]  1 ## Bike_station 스테이션을 통해 위도와 경도 값을 merge해 준다.
      2 df_B_in_Ko_merge_Bike = pd.merge(df_B_in_Ko,Bike_station, on=['Station_ID'],how='left')
      3 df_B_in_Ko_merge_Bike['Station_ID'] = df_B_in_Ko_merge_Bike[['Station_ID']].astype('str') # int->str로 변경해 준다
      4 df_B_in_Ko_merge_Bike
```

	EMD_KOR_NM	Station_ID	STATION_NAME	거치대 수량	위도	경도	geometry
0	성사동	101	어울림마을 701동 앞	20	37.654775	126.834584	POINT (126.83458 37.65477)
1	성사동	103	대림e-편한세상106동	20	37.660442	126.840377	POINT (126.84038 37.66044)
2	탄현동	104	탄현마을8단지	25	37.698523	126.766042	POINT (126.76604 37.69852)
3	성사동	105	KT 덕양지사 앞	20	37.655244	126.839261	POINT (126.83926 37.65524)
4	성사동	106	원당역 앞 공영주차장	30	37.653410	126.842530	POINT (126.84253 37.65341)
...	...	...	...	...	...	...	...
159	덕이동	350	★하이파크5단지 502동앞 버스정류장	20	37.697867	126.753089	POINT (126.75309 37.69787)
160	탄현동	351	◆일산에듀포레 푸르지오	40	37.702259	126.767231	POINT (126.76723 37.70226)
161	대화동	352	◆꿈에그린203동앞	40	37.666425	126.749244	POINT (126.74924 37.66642)
162	대화동	353	◆꿈에그린106동앞	40	37.666720	126.750784	POINT (126.75078 37.66672)
163	화정동	992	★피프틴센터	4	37.637529	126.833760	POINT (126.83376 37.63753)

164 rows × 7 columns

각 스테이션이 어떤 법정동에 포함되어 있는지 구분

고양시 동 별 자전거 스테이션 현황

	EMD_KOR_NM	harv(m)_mean	harv(m)_Less than 500	Count	Count_OH
0	성사동	614.160193	383.862977	4	4
1	탄현동	680.163285	378.576435	8	8
2	화정동	790.823164	320.174853	13	13
3	토당동	646.269375	477.422658	4	4
4	행신동	896.254871	401.809122	13	13
5	향동동	870.278696	242.687549	4	0
6	원흥동	669.399417	379.775971	6	6
7	삼송동	1063.294363	NaN	3	3
8	신원동	405.703008	336.502078	4	4
9	동산동	547.685366	NaN	2	2
10	백석동	765.372225	368.239143	13	13
11	장항동	994.477353	360.314420	10	10
12	마두동	971.433428	420.093730	13	13
13	가좌동	228.325286	228.325286	2	2
14	정발산동	737.362559	392.840296	6	6
15	풍동	634.971108	444.478553	4	4
16	식사동	1020.768699	NaN	2	0
17	중산동	821.649348	333.038210	8	8
18	일산동	953.732576	374.051147	12	12
19	대화동	1177.148169	303.793219	14	12
20	주엽동	759.976031	396.743451	10	10
21	덕이동	515.191450	269.674457	6	6
22	주교동	NaN	NaN	1	1
23	덕은동	NaN	NaN	1	0
24	도내동	NaN	NaN	1	1

- 각 법정동 안에 포함된 스테이션의 직선거리를 계산. 이때, 직선거리는 haversine 함수를 통해 m단위로 계산.
- 또한, 동 안의 스테이션 간 거리가 500미터 이내일 때의 평균과 전체 동 안의 스테이션 간 거리를 구분. 추후, 스테이션을 신설/조정할 때 스테이션 간 최소 거리를 500미터로 두기 위함
- ‘Count’ 컬럼은 법정동 내 스테이션 개수
- ‘Count_OH’ 컬럼은 운영이력 데이터에 포함된 법정동내 스테이션 개수
- 동별 자전거 스테이션 개수를 통해 다른 요소와 결합하여 동별 자전거 거치대의 개수까지 계산 가능

4. 고양시 도시화 경계 자전거 스테이션 현황

고양시 도시화 경계 자전거 스테이션 현황

```
[85]  1 ## 경계별 자전거 스테이션 포함 여부
      2 # Station_ID_2 = []
      3 # EMD_KOR_NM_1 = []
      4 for i in Bike_station.index:
      5   for j in city_border.index:
      6     if Bike_station.geometry[i].within(city_border.geometry[j]):
      7       print('{}는 {}에 속해있다.'.format(Bike_station.Station_ID[i],city_border.UA_CD[j]))
      8
```

```
101는 UA311014에 속해있다.
103는 UA311014에 속해있다.
104는 UA311041에 속해있다.
105는 UA311014에 속해있다.
110는 UA311011에 속해있다.
111는 UA311011에 속해있다.
112는 UA311011에 속해있다.
113는 UA311011에 속해있다.
114는 UA311011에 속해있다.
115는 UA311011에 속해있다.
116는 UA311011에 속해있다.
118는 UA311011에 속해있다.
119는 UA311011에 속해있다.
121는 UA311011에 속해있다.
123는 UA311011에 속해있다.
124는 UA311011에 속해있다.
125는 UA311011에 속해있다.
126는 UA311011에 속해있다.
127는 UA311011에 속해있다.
128는 UA311011에 속해있다.
129는 UA311011에 속해있다.
130는 UA311011에 속해있다.
131는 UA311011에 속해있다.
137는 UA311011에 속해있다.
138는 UA311011에 속해있다.
139는 UA311011에 속해있다.
140는 UA311011에 속해있다.
141는 UA311011에 속해있다.
142는 UA311011에 속해있다.
143는 UA311011에 속해있다.
146는 UA311014에 속해있다.
148는 UA311011에 속해있다.
162는 UA311013에 속해있다.
164는 UA311013에 속해있다.
```

고양시 도시화 경계 지역 데이터에 자전거 스테이션이 어디에 포함되는지 여부 계산

고양시 도시화 경계 자전거 스테이션 현황

동 별로 자전거 스테이션을 개수를 구한 것처럼 도시화 경계 안에서의 자전거 스테이션 개수를 계산. 자전거 스테이션의 위치를 선정하기 위해 거리 및 개수 역시 동일한 방식으로 계산

```
[89] 1 ## Bike_station 스테이션을 통해 위도와 경도 값을 merge해 준다.
     2 df_B_in_ci_merge_Bike = pd.merge(df_B_in_ci,Bike_station, on=['Station_ID'],how='outer')
     3 df_B_in_ci_merge_Bike['Station_ID'] = df_B_in_ci_merge_Bike[['Station_ID']].astype('str') # int->str로 변경해 준다
     4 df_B_in_ci_merge_Bike # 포함되지 않는 것들도 같이 merge시켜 준다.
```

	UA_CD	Station_ID	STATION_NAME	거치대 수량	위도	경도	geometry
0	UA311014	101	어울림마을 701동 앞	20	37.654775	126.834584	POINT (126.83458 37.65477)
1	UA311014	103	대림e-편한세상106동	20	37.660442	126.840377	POINT (126.84038 37.66044)
2	UA311041	104	탄현마을8단지	25	37.698523	126.766042	POINT (126.76604 37.69852)
3	UA311014	105	KT 덕양지사 앞	20	37.655244	126.839261	POINT (126.83926 37.65524)
4	UA311011	110	어울림누리 맞은편	20	37.647972	126.834469	POINT (126.83447 37.64797)
...	...	...	...	...	...	...	...
159	NaN	265	◆원시티 육교	40	37.661296	126.750847	POINT (126.75085 37.66130)
160	NaN	301	고양체육관 부출입구(후문)	20	37.674848	126.741106	POINT (126.74111 37.67485)
161	NaN	331	탄현역 동쪽 출구	20	37.693514	126.761304	POINT (126.76130 37.69351)
162	NaN	349	★고양시 여성창업지원센터 옆	20	37.697838	126.752642	POINT (126.75264 37.69784)
163	NaN	351	◆일산에듀포레 푸르지오	40	37.702259	126.767231	POINT (126.76723 37.70226)

164 rows × 7 columns

	UA_CD	harv(m)_mean	harv(m)_Less than 500	Count
0	UA311014	635.750349	428.676814	4
1	UA311041	1872.029829	368.631440	45
2	UA311011	1423.021035	354.176474	28
3	UA311013	1020.495417	377.318454	9
4	UA311015	405.703008	336.502078	4
5	UA311031	2126.726631	377.663939	47
6	UA311042	228.325286	228.325286	2

5. 자전거 스테이션의 영향권 설정 코드 설명

(1) 가상의 최대 허용거리 내 지표 취합

가상의 최대 허용거리 내 지표 취합

활용 데이터 목록

```
# 데이터
bike_oh=pd.read_csv('/content/drive/My Drive/compas/SBJ_2007_001/01.운영이력.csv')
Bike_Station=pd.read_csv('/content/drive/My Drive/compas/SBJ_2007_001/02.자전거스테이션.csv')
population=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/06.인구(거주)분포도(100M X 100M).geojson')
space_plan=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/10.도시계획(공간시설).geojson')
culphy_plan=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/11.도시계획(공공문화체육시설).geojson')
traffic_plan=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/12.도시계획(교통시설).geojson')
wetland_cons=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/13.용도지역지구(습지보호지역).geojson')
build_road=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/15.도로명주소_건물.geojson')
road_road=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/16.도로명주소_도로.geojson')
build_dist=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/17.일반건물 분포도(100M X 100M).geojson')
fes_space=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/18.행사장_공간정보.csv')
subway_station=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/19.전철역_공간정보.csv')
bus_station=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/20.고양시 버스정류소.csv')
parking_info=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/22.주차장정보.csv')
theat_muse=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/26.고양시 공연장 박물관 정보.csv')
physi_curr=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/27.고양시 체육시설 현황 정보.csv')
duk_godo=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/30.고양시_덕양구_고도.geojson')
ilsansu_godo=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/32.고양시_일산서구_고도.geojson')
ilsando_godo=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/31.고양시_일산동구_고도.geojson')
goyang_road=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/33.고양시_인도.geojson')
```

데이터 좌표 정보의 종류

MULTIPOLYGON – 인구 거주 분포, 도시 계획,
도로명 주소 건물

POLYGON – 고도 데이터

LATITUDE, LONGITUDE – 자전거 스테이션, 행사장 공간정보,
전철역_공간정보, 공연장 박물관 정보,
체육시설 현황 정보, 주차장정보

MULTILINESTRING – 자전거 도로

가상의 최대 허용거리 내 지표 추합

최대 허용 거리를 300으로 설정한 후, MULTIPOLYGON 데이터 추합 함수 작성

Multipolygon

- geometry type이 Multipolygon인 데이터프레임에 대하여 LSCP 데이터프레임 생성

```
[12] import math
```

```
[13] # 300m 기준
     lscp_distance=300
```

```
def geo_dis(Bike_Station, target):
  lscp=[]
  target_nm=target.columns.tolist()[[(t.find('NAME')>=0) | (t.lower().find('dgm_nm')>=0) |
                                     (t.lower().find('bdtyp_cd')>=0) | (t.lower().find('mnum')>=0) for t in target.columns.tolist()].index(True)]
  for sta, lat, long in zip(Bike_Station['Station_ID'],Bike_Station['위도'],Bike_Station['경도']):
    for nm, geometry in zip(target[target_nm],target['geometry']):
      h=[]
      x = (lat, long)
      points = [point for polygon in geometry for point in polygon.exterior.coords[:]]
# Bike_Station 기준으로 해당 멀티폴리곤의 모든 점과의 거리를 구함
      for t_long, t_lat in points:
        y=(t_lat, t_long)
        h.append(haversine(x,y, unit='m'))
# 만약 그 거리가 LSCP 기준 보다 낮으면 반환.
      if min(h)<=lscp_distance:
        lscp.append([sta,nm,min(h)])
  return lscp
```

MULTIPOLYGON을 점으로 반환

가상의 최대 허용거리 내 지표 추합

고도 데이터 추합 함수 작성

```
[112] # 데이터 프레임화
      bike_duk=pd.DataFrame(geo_godo(Bike_Station,duk_godo),columns=['Bike_Station','DN','Harv_dis'])
      bike_ildo=pd.DataFrame(geo_godo(Bike_Station,ilsando_godo),columns=['Bike_Station','DN','Harv_dis'])
      bike_ilsu=pd.DataFrame(geo_godo(Bike_Station,ilsansu_godo),columns=['Bike_Station','DN','Harv_dis'])
```

```
[113] #lscp_base와 병합 이후 차례로 병합
      base_duk=lscp_base.merge(bike_duk.iloc[:,:-1],on='Bike_Station',how='outer')
      duk_ildo=base_duk.merge(bike_ildo.iloc[:,:-1],on='Bike_Station',how='outer')
      ildo_ilsu=duk_ildo.merge(bike_ilsu.iloc[:,:-1],on='Bike_Station',how='outer')
```

```
[114] #nan값 채워주기
      ildo_ilsu=ildo_ilsu.fillna(0)
```

```
[115] #평균을 구하기 위한 물밀작업
      ildo_ilsu['DN_avg']=0.0
```

```
ildo_ilsu
```

	Bike_Station	DN_x	DN_y	DN	DN_avg
0	101	15.0	0.0	0.0	0.0
1	101	14.0	0.0	0.0	0.0
2	101	17.0	0.0	0.0	0.0
3	101	10.0	0.0	0.0	0.0
4	101	14.0	0.0	0.0	0.0
...	...	...	...	...	...
22927	992	13.0	0.0	0.0	0.0
22928	992	15.0	0.0	0.0	0.0
22929	992	17.0	0.0	0.0	0.0
22930	992	16.0	0.0	0.0	0.0
22931	992	12.0	0.0	0.0	0.0

22932 rows × 5 columns

```
[117] # 값이 존재할 경우에만 count를 더해 실질적인 평균값이 나오도록 설계
      for idx,i in enumerate(ildo_ilsu['DN_avg']):
        count=0
        if ildo_ilsu['DN'][idx]!=0:
          count+=1
        if ildo_ilsu['DN_x'][idx]!=0:
          count+=1
        if ildo_ilsu['DN_y'][idx]!=0:
          count+=1
        ildo_ilsu['DN_avg'][idx]=(ildo_ilsu['DN'][idx]+ildo_ilsu['DN_x'][idx]+ildo_ilsu['DN_y'][idx])/count
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.h
  # Remove the CWD from sys.path while we load stuff.
```

```
ildo_ilsu
```

	Bike_Station	DN_x	DN_y	DN	DN_avg
0	101	15.0	0.0	0.0	15.0
1	101	14.0	0.0	0.0	14.0
2	101	17.0	0.0	0.0	17.0
3	101	10.0	0.0	0.0	10.0
4	101	14.0	0.0	0.0	14.0
...	...	...	...	...	...
22927	992	13.0	0.0	0.0	13.0
22928	992	15.0	0.0	0.0	15.0
22929	992	17.0	0.0	0.0	17.0
22930	992	16.0	0.0	0.0	16.0
22931	992	12.0	0.0	0.0	12.0

22932 rows × 5 columns

값이 존재하는 경우만 count를 더해 LSCP_DISTANCE 내 평균값계산

가상의 최대 허용거리 내 지표 추합

위도, 경도로 구성된 데이터 추합 함수 작성

```
[65]  # X,Y 좌표값으로 이루어진 데이터의 LSCP(300m) 내 존재 유무, 개수 카운트 함수
      def harv_dis(Bike_Station, target):
        lscp=[]
        target_nm=target.columns.tolist()[[(t.find('NAME')>=0) | (t.lower().find('station_nm')>=0) |
                                           (t.lower().find('parking_nm')>=0) | (t.lower().find('place_nm')>=0)|
                                           (t.find('명칭')>=0) for t in target.columns.tolist()].index(True)]
        for sta, lat, long in zip(Bike_Station['Station_ID'],Bike_Station['위도'],Bike_Station['경도']):
          try:
            for nm, t_lat, t_long in zip(target[target_nm],target['위도'],target['경도']):
              t_lat=float(t_lat)
              t_long=float(t_long)
              x = (lat, long)
              y = (t_lat, t_long)
              h = haversine(x, y, unit='m')
              if h <=lscp_distance:
                lscp.append([sta,nm,h])
          except:
            try:
              for nm, t_lat, t_long, t_count in zip(target[target_nm],target['Y'],target['X'],target['count']):
                t_lat=float(t_lat)
                t_long=float(t_long)
                x = (lat, long)
                y = (t_lat, t_long)
                h = haversine(x, y, unit='m')
                if h <=lscp_distance:
                  lscp.append([sta,nm,t_lat,t_long,t_count,h])
            except:
              for nm, t_lat, t_long in zip(target[target_nm],target['Y'],target['X']):
                t_lat=float(t_lat)
                t_long=float(t_long)
                x = (lat, long)
                y = (t_lat, t_long)
                h = haversine(x, y, unit='m')
                if h <=lscp_distance:
                  lscp.append([sta,nm,t_lat,t_long,h])
        return lscp
```

위도 경도를 받은 후,

Haversine 함수를 활용하기 위해,

튜플로 묶어주었다.

+) 코드 예외처리를 위해 try, except 활용.

가상의 최대 허용거리 내 지표 추합

자전거 도로 데이터 추합 함수 작성

```
[124] def geo_bike(Bike_Station, target):
        lscp=[]
        for sta, lat, long in zip(Bike_Station['Station_ID'],Bike_Station['위도'],Bike_Station['경도']):
          for nm, geometry in zip(target['자전거도로유/무'],target['geometry']):
            h=[]
            ex=0
            x = (lat, long)
            if nm=='BYC001':
              ex=1
            points = [point for polygon in geometry for point in polygon.coords[:]]
            for t_long, t_lat in points:
              y=(t_lat, t_long)
              h.append(haversine(x,y, unit='m'))
            if min(h)<=lscp_distance:
              lscp.append([sta,ex,min(h)])
        return lscp
```

LINESTRING을 점으로 반환

가상의 최대 허용거리 내 지표 추합

병합된 데이터

```
[203] hosu_park
```

	Bike_Station	Loan_count	Return_count	Population	Sub_pop	Bus_pop	DN_avg	Road_yes	Space_count	Culture_count	Sport_count	KINTEX	Hosu_yes	Park_count
0	101	8366	9207	8692	0	142127	17	1	419	0	0	0	0	0
1	103	4535	5121	14038	0	37794	22	1	0	0	0	0	0	1
2	104	13402	9330	13872	0	53381	35	0	0	0	0	0	0	0
3	105	2806	2696	11255	0	136789	23	1	349	0	0	0	0	0
4	106	4444	2357	4859	8607849	257783	26	1	294	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
159	350	4190	3662	10617	0	29430	25	0	0	0	0	0	0	0
160	351	473	406	12091	0	25007	38	0	0	0	0	0	0	0
161	352	0	0	7293	0	25959	13	1	282	0	0	0	0	1
162	353	0	0	7236	0	65082	14	1	0	1	0	0	0	0
163	992	952	795	9652	0	265812	15	1	685	0	0	0	0	1

164 rows × 14 columns

가상의 최대 허용거리 내 지표 추합

도로명 건물 칼럼들을 반영하기 위해 함수 따로 작성

도로명 주소

```
[133] # 도로명 주소 기반으로 LSCP 내의 드는 값들의 count를 반영한 데이터프레임을 만들어주고자 함.
      def geodis2(Bike_Station, target):
        lscp=[]
        target_nm=target.columns.tolist()[[(t.find('NAME')>=0) | (t.lower().find('dgm_nm')>=0) |
                                           (t.lower().find('bdtyp_cd')>=0) | (t.lower().find('mnum')>=0) for t in target.columns.tolist()].index(True)]
        for sta, lat, long in zip(Bike_Station['Station_ID'],Bike_Station['위도'],Bike_Station['경도']):
          for nm, geometry in zip(target[target_nm],target['geometry']):
            h=[]
            x = (lat, long)
            points = [point for polygon in geometry for point in polygon.exterior.coords[:]]
            for t_long, t_lat in points:
              y=(t_lat, t_long)
              h.append(haversine(x,y, unit='m'))
            if min(h)<=lscp_distance:
              lscp.append([sta,nm,min(h)])
        lscp_pd=pd.DataFrame(lscp,columns=['Bike_Station','BDTYP_CD','Harv_Dis'])
        lscp_pd['BDTYP_CD_count']=1
        lscp_pd=lscp_pd[['Bike_Station','BDTYP_CD_count']].groupby('Bike_Station').agg('sum').reset_index()
        lscp_pd.columns=['Bike_Station','BDTYP_CD_count']
        lscp_pd=lscp_base.merge(lscp_pd[['Bike_Station','BDTYP_CD_count']],on='Bike_Station',how='outer').fillna(0)
        lscp_pd['BDTYP_CD_count']=lscp_pd['BDTYP_CD_count'].astype(int)
        return lscp_pd
```

가상의 최대 허용거리 내 지표 추합

병합된 데이터 (도로명 건물 확장)

[180] build_220

	Bike_Station	Loan_count	Return_count	Population	Sub_pop	Bus_pop	DN_avg	Road_yes	Space_count	Culture_count	Sport_count	KINTEX	Hosu_yes	Park_count	BDTYP_CD_01003	BDTYP_CD_02003	BDTYP_CD_03999	BDTYP_CD_03005
0	101	8366	9207	8692	0	142127	17	1	419	0	0	0	0	0	9	11	37	3
1	103	4535	5121	14038	0	37794	22	1	0	0	0	0	0	1	37	10	5	0
2	104	13402	9330	13872	0	53381	35	0	0	0	0	0	0	0	0	0	1	0
3	105	2806	2696	11255	0	136789	23	1	349	0	0	0	0	0	60	68	15	3
4	106	4444	2357	4859	8607849	257783	26	1	294	0	0	0	0	0	12	54	7	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
159	350	4190	3662	10617	0	29430	25	0	0	0	0	0	0	0	0	0	4	0
160	351	473	406	12091	0	25007	38	0	0	0	0	0	0	0	0	0	0	0
161	352	0	0	7293	0	25959	13	1	282	0	0	0	0	1	0	0	1	0
162	353	0	0	7236	0	65082	14	1	0	1	0	0	0	0	0	0	1	0
163	992	952	795	9652	0	265812	15	1	685	0	0	0	0	1	66	0	2	1

164 rows × 234 columns

5. 자전거 스테이션의 영향권 설정 코드 설명

(2) 가상의 최대 허용거리 내 지표 취합

회귀 모델을 통한 LSCP 최대 허용거리 선정

병합된 데이터의 활용

스테이션 별로 LSCP 최대 허용거리 내로 다른 데이터들을 병합, 취합시켰음.

종속변수를 대여 횟수, 반납 횟수 혹은 대여+ 반납 횟수로 두고 나머지 변수들을 독립변수로 둔다면 임의의 모델에 적합시키는 것도 가능할 것.

LSCP_DISTANCE를 변경해가며 다양한 최대 허용거리의 LSCP 데이터 프레임을 만듬.
(LSCP_500, LSCP_363 외에도 LSCP_450, LSCP_400, LSCP_350, LSCP_300, LSCP_250)

회귀 모델을 통한 LSCP 최대 허용거리 선정

최대 허용 거리를 500으로 두었을 때의 데이터 프레임

```
lscp_500
```

	Bike_Station	Loan_count	Return_count	Population	Sub_pop	Bus_pop	DN_avg	Road_yes	Space_count	Culture_count	Sport_count	KINTEX	Hosu_yes	Park_count	Loan_Return	Sport_yes
0	101	8366	9207	8692	0	228849	21	1	711	2	1	0	0	0	17573	1
1	103	4535	5121	14038	0	81961	25	1	0	0	7	0	0	2	9656	1
2	104	13402	9330	13872	0	78852	35	0	0	0	2	0	0	0	22732	1
3	105	2806	2696	11255	8607849	422520	25	1	349	0	0	0	0	0	5502	0
4	106	4444	2357	4859	8607849	308135	30	1	349	0	0	0	0	0	6801	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
159	350	4190	3662	10617	0	31836	25	0	0	0	4	0	0	1	7852	1
160	351	473	406	12091	0	63520	58	0	0	0	2	0	0	0	879	1
161	352	0	0	7293	0	89081	14	1	282	1	0	1	0	1	0	0
162	353	0	0	7236	0	91515	16	1	282	1	0	1	0	1	0	0
163	992	952	795	9652	14522105	527839	17	1	685	0	2	0	0	3	1747	1

164 rows × 16 columns

회귀 모델을 통한 LSCP 최대 허용거리 선정

최대 허용 거리를 363으로 두었을 때의 데이터 프레임

(근거 : 500m 이내 스테이션 간 평균 거리 363m)

```
[80] lscp_363
```

	Bike_Station	Loan_count	Return_count	Population	Sub_pop	Bus_pop	DN_avg	Road_yes	Space_count	Culture_count	Sport_count	KINTEX	Hosu_yes	Park_count	Loan_Return	Sport_yes
0	101	8366	9207	8692	0	177462	18	1	419	0	0	0	0	0	17573	0
1	103	4535	5121	14038	0	47393	23	1	0	0	7	0	0	1	9656	1
2	104	13402	9330	13872	0	57385	35	0	0	0	0	0	0	0	22732	0
3	105	2806	2696	11255	0	167918	24	1	349	0	0	0	0	0	5502	0
4	106	4444	2357	4859	8607849	257783	27	1	349	0	0	0	0	0	6801	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
159	350	4190	3662	10617	0	31581	26	0	0	0	1	0	0	1	7852	1
160	351	473	406	12091	0	37680	38	0	0	0	2	0	0	0	879	1
161	352	0	0	7293	0	72143	13	1	282	0	0	0	0	1	0	0
162	353	0	0	7236	0	87525	15	1	282	1	0	0	0	0	0	0
163	992	952	795	9652	14522105	518719	16	1	685	0	0	0	0	1	1747	0

164 rows × 16 columns

회귀 모델을 통한 LSCP 최대 허용거리 선정

LSCP 모델의 칼럼 의미

Bike_Station : 자전거 스테이션 번호

Loan_count : 총 대여 횟수

Return_count : 총 반납 횟수

Population : 주변 거주 인구

Sub_pop : 주변 지하철 탑승 인구

Bus_pop : 주변 버스 탑승 인구

DN_avg : 주변 고도

Road_yes : 자전거 도로 유무

Space_count : 주변 주차장 공간

Culture_count : 주변 문화시설 수

Sport_count : 주변 체육시설 수

KINTEX : 킨텍스 유무

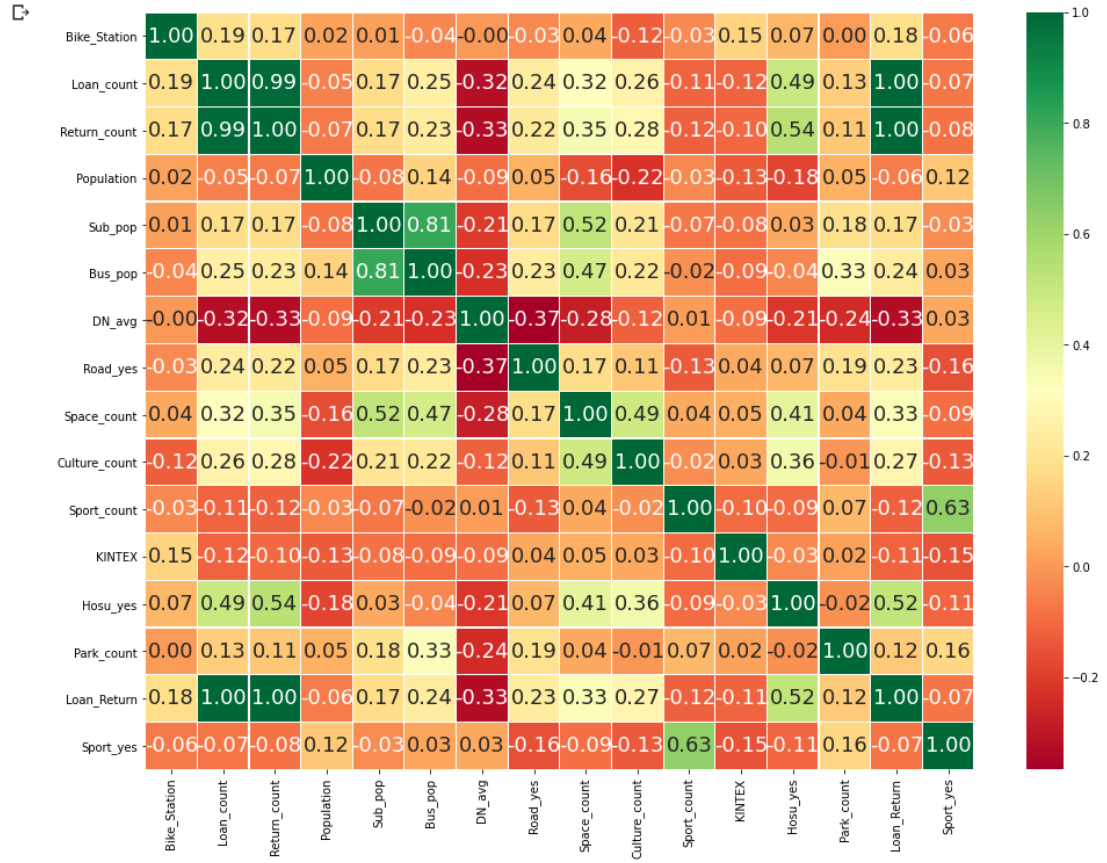
Hosu_yes : 호수 공원 유무

Park_count : 주변 공원 수

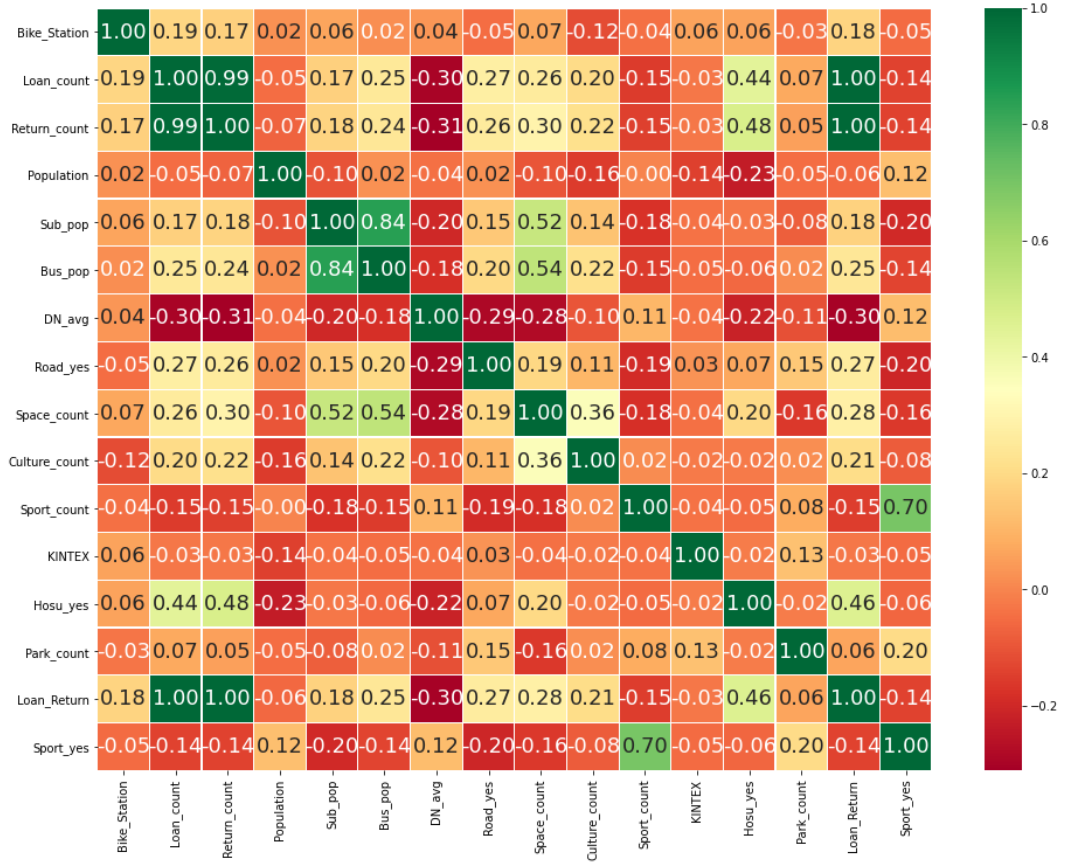
회귀 모델을 통한 LSCP 최대 허용거리 선정

LSCP_500 과 LSCP_300 데이터프레임 상관 그래프

```
plt.figure(figsize=(16,12))
g=sns.heatmap(lscp_500.corr(), cmap='RdYlGn',linewidths=0.2,annot_kws={'size':18},annot=True,fmt='.2f')
```




```
plt.figure(figsize=(16,12))
g=sns.heatmap(lscp_363.corr(), cmap='RdYlGn',linewidths=0.2,annot_kws={'size':18},annot=True,fmt='.2f')
```



회귀 모델을 통한 LSCP 최대 허용거리 선정

Feature Engineering

```
[83]  # trans_pop : 지하철, 버스 유동인구를 합쳐줌
      # all_pop : 주거 인구 + 유동 인구
      # Loan_Return : 대여, 반납 횟수 합
      # Sport_yes : 체육 시설의 경우, 한 장소에 여러 개의 시설이 있는 복합 시설이 있어 유무로 우선 영향을 파악하는 게 낫다고 판단함.

      lscp_500['Loan_Return']=lscp_500['Loan_count']+lscp_500['Return_count']
      lscp_500['Sport_yes']=np.where(lscp_500['Sport_count']>0,1,0)

      lscp_363['Loan_Return']=lscp_363['Loan_count']+lscp_363['Return_count']
      lscp_363['Sport_yes']=np.where(lscp_363['Sport_count']>0,1,0)

      lscp_450['Loan_Return']=lscp_450['Loan_count']+lscp_450['Return_count']
      lscp_450['Sport_yes']=np.where(lscp_450['Sport_count']>0,1,0)

      lscp_400['Loan_Return']=lscp_400['Loan_count']+lscp_400['Return_count']
      lscp_400['Sport_yes']=np.where(lscp_400['Sport_count']>0,1,0)

      lscp_350['Loan_Return']=lscp_350['Loan_count']+lscp_350['Return_count']
      lscp_350['Sport_yes']=np.where(lscp_350['Sport_count']>0,1,0)

      lscp_300['Loan_Return']=lscp_300['Loan_count']+lscp_300['Return_count']
      lscp_300['Sport_yes']=np.where(lscp_300['Sport_count']>0,1,0)

      lscp_250['Loan_Return']=lscp_250['Loan_count']+lscp_250['Return_count']
      lscp_250['Sport_yes']=np.where(lscp_250['Sport_count']>0,1,0)
```

회귀 모델을 통한 LSCP 최대 허용거리 선정

LSCP_500의 다변량 회귀 모델

```
# 대여 + 반납 lscp_500
result = sm.ols(formula = 'Loan_Return ~ Bike_Station+Population+Sub_pop+Bus_pop+DN_avg+Road_yes+Space_count+Culture_count+Sport_yes+KINTEX+Hosu_yes+Park_count', data = lscp_500).fit()
result.summary()
```

OLS Regression Results

Dep. Variable:	Loan_Return	**R-squared:**	0.432
Model:	OLS	**Adj. R-squared:**	0.387
Method:	Least Squares	**F-statistic:**	9.574
Date:	Thu, 10 Sep 2020	**Prob (F-statistic):**	1.15e-13
Time:	18:19:58	**Log-Likelihood:**	-1900.5
No. Observations:	164	**AIC:**	3827.
Df Residuals:	151	**BIC:**	3867.
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.424e+04	1.47e+04	0.966	0.336	-1.49e+04	4.34e+04
Bike_Station	72.9629	22.924	3.183	0.002	27.669	118.257
Population	-0.7148	0.611	-1.169	0.244	-1.923	0.493
Sub_pop	-0.0019	0.001	-2.006	0.047	-0.004	-2.89e-05
Bus_pop	0.0833	0.026	3.153	0.002	0.031	0.135
DN_avg	-616.4892	252.933	-2.437	0.016	-1116.234	-116.745
Road_yes	1.328e+04	9081.464	1.463	0.146	-4658.198	3.12e+04
Space_count	-2.0672	10.514	-0.197	0.844	-22.841	18.707
Culture_count	1977.3562	2472.214	0.800	0.425	-2907.242	6861.954
Sport_yes	-895.3961	4528.299	-0.198	0.844	-9842.404	8051.612
KINTEX	-3.682e+04	1.67e+04	-2.199	0.029	-6.99e+04	-3738.584
Hosu_yes	6.789e+04	1.12e+04	6.054	0.000	4.57e+04	9e+04
Park_count	-280.2923	2421.064	-0.116	0.908	-5063.828	4503.244

Omnibus:	39.589	**Durbin-Watson:**	1.082
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	98.388
Skew:	1.002	**Prob(JB):**	4.32e-22
Kurtosis:	6.222	**Cond. No.**	3.95e+07

회귀 모델을 통한 LSCP 최대 허용거리 선정

LSCP_363의 다변량 회귀 모델

```
# 대여 + 반납 lscp_363
result = sm.ols(formula = 'Loan_Return ~ Bike_Station+Population+Sub_pop+Bus_pop+DN_avg+Road_yes+Space_count+Culture_count+Sport_yes+KINTEX+Hosu_yes+Park_count', data = lscp_363).fit()
result.summary()
```

OLS Regression Results

Dep. Variable:	Loan_Return	**R-squared:**	0.399
Model:	OLS	**Adj. R-squared:**	0.351
Method:	Least Squares	**F-statistic:**	8.350
Date:	Thu, 10 Sep 2020	**Prob (F-statistic):**	5.68e-12
Time:	18:19:58	**Log-Likelihood:**	-1905.2
No. Observations:	164	**AIC:**	3836.
Df Residuals:	151	**BIC:**	3877.
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	3297.8719	1.32e+04	0.250	0.803	-2.27e+04	2.93e+04
Bike_Station	69.9262	23.281	3.004	0.003	23.929	115.924
Population	0.3374	0.615	0.549	0.584	-0.877	1.552
Sub_pop	-0.0011	0.001	-0.956	0.341	-0.003	0.001
Bus_pop	0.0777	0.034	2.287	0.024	0.011	0.145
DN_avg	-516.6047	279.486	-1.848	0.066	-1068.813	35.603
Road_yes	1.548e+04	7475.700	2.071	0.040	710.960	3.03e+04
Space_count	-8.1378	14.581	-0.558	0.578	-36.946	20.671
Culture_count	7507.7667	2876.438	2.610	0.010	1824.504	1.32e+04
Sport_yes	-3365.7473	5079.843	-0.663	0.509	-1.34e+04	6671.001
KINTEX	-1.32e+04	2.89e+04	-0.457	0.648	-7.02e+04	4.38e+04
Hosu_yes	8.137e+04	1.29e+04	6.329	0.000	5.6e+04	1.07e+05
Park_count	1485.0446	3189.737	0.466	0.642	-4817.234	7787.323

Omnibus:	36.372	**Durbin-Watson:**	1.105
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	101.644
Skew:	0.871	**Prob(JB):**	8.48e-23
Kurtosis:	6.441	**Cond. No.**	5.36e+07

회귀 모델을 통한 LSCP 최대 허용거리 선정

LSCP 회귀 분석 모델의 의미

모델 자체를 활용하기엔 R스퀘어 값이나 유의수준 값이 유의미하지 않아 어려움이 있으나
R스퀘어 값을 활용해 더욱 적절한 LSCP 최대 허용 거리를 설정할 수 있을 것이며
유의 수준을 활용해 유의미한 피쳐만 뽑아낼 수 있을 것이라 판단했다.

따라서, LSCP_500, LSCP_363 외에도 LSCP_450, LSCP_400, LSCP_350, LSCP_300, LSCP_250 데이터 프레임들에 대해
다변량 회귀 분석 모델을 적합시켜보았다.

그 결과, LSCP_300 모델의 R스퀘어 값이 가장 높은 것으로 나왔다.

회귀 모델을 통한 LSCP 최대 허용거리 선정

LSCP_300의 다변량 회귀 모델

```
[89] # 대여 + 반납 lscp_300
     result = sm.ols(formula = 'Loan_Return ~ Bike_Station+Population+Sub_pop+Bus_pop+DN_avg+Road_yes+Space_count+Culture_count+Sport_yes+KINTEX+Hosu_yes+Park_count', data = lscp_300).fit()
     result.summary()
```

OLS Regression Results

Dep. Variable:	Loan_Return	**R-squared:**	0.443
Model:	OLS	**Adj. R-squared:**	0.399
Method:	Least Squares	**F-statistic:**	10.02
Date:	Thu, 10 Sep 2020	**Prob (F-statistic):**	2.86e-14
Time:	18:19:58	**Log-Likelihood:**	-1898.8
No. Observations:	164	**AIC:**	3824.
Df Residuals:	151	**BIC:**	3864.
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-9220.6743	1.26e+04	-0.732	0.465	-3.41e+04	1.57e+04
Bike_Station	89.2562	22.791	3.916	0.000	44.225	134.287
Population	0.8812	0.600	1.469	0.144	-0.304	2.066
Sub_pop	0.0028	0.001	2.146	0.033	0.000	0.005
Bus_pop	0.0313	0.037	0.853	0.395	-0.041	0.104
DN_avg	-436.8140	276.833	-1.578	0.117	-983.781	110.153
Road_yes	1.615e+04	6911.421	2.337	0.021	2494.612	2.98e+04
Space_count	-37.8018	16.342	-2.313	0.022	-70.090	-5.514
Culture_count	1.446e+04	3361.525	4.300	0.000	7813.368	2.11e+04
Sport_yes	100.1593	5458.578	0.018	0.985	-1.07e+04	1.09e+04
KINTEX	-1.195e+04	2.78e+04	-0.429	0.668	-6.69e+04	4.3e+04
Hosu_yes	9.045e+04	1.25e+04	7.263	0.000	6.58e+04	1.15e+05
Park_count	3319.3752	3283.082	1.011	0.314	-3167.335	9806.085

Omnibus:	22.983	**Durbin-Watson:**	1.220
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	40.272
Skew:	0.693	**Prob(JB):**	1.80e-09
Kurtosis:	4.992	**Cond. No.**	4.80e+07

회귀 모델을 통한 LSCP 최대 허용거리 선정

STEPWISE 방법론으로 뽑은 LSCP_300의 다변량 회귀 모델

```
# 대여 + 반납 lscp_300
result = sm.ols(formula = 'Loan_Return ~ Bike_Station+Population+Sub_pop+DN_avg+Road_yes+Space_count+Culture_count+Hosu_yes+Park_count', data = lscp_300).fit()
result.summary()
```

OLS Regression Results

Dep. Variable:	Loan_Return	R-squared:	0.440
Model:	OLS	Adj. R-squared:	0.407
Method:	Least Squares	F-statistic:	13.44
Date:	Thu, 10 Sep 2020	Prob (F-statistic):	9.38e-16
Time:	18:19:58	Log-Likelihood:	-1899.4
No. Observations:	164	AIC:	3819.
Df Residuals:	154	BIC:	3850.
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1.058e+04	1.24e+04	-0.856	0.393	-3.5e+04	1.38e+04
Bike_Station	89.2774	22.563	3.957	0.000	44.703	133.851
Population	1.0512	0.567	1.853	0.066	-0.069	2.172
Sub_pop	0.0036	0.001	4.356	0.000	0.002	0.005
DN_avg	-411.0401	273.404	-1.503	0.135	-951.146	129.066
Road_yes	1.635e+04	6760.375	2.419	0.017	2999.685	2.97e+04
Space_count	-34.4392	15.791	-2.181	0.031	-65.633	-3.245
Culture_count	1.488e+04	3308.372	4.498	0.000	8344.994	2.14e+04
Hosu_yes	9.147e+04	1.23e+04	7.439	0.000	6.72e+04	1.16e+05
Park_count	3434.2552	3164.355	1.085	0.279	-2816.891	9685.401

Omnibus:	22.547	Durbin-Watson:	1.242
Prob(Omnibus):	0.000	Jarque-Bera (JB):	37.080
Skew:	0.708	Prob(JB):	8.87e-09
Kurtosis:	4.850	Cond. No.	2.43e+07

이 모델이 가장 높은 adj R2값을 보여주었다. 다른 변수들이(Bus_pop, Sport_yes, KINTEX) 빠졌음에도 좀 더 나은 설명력을 보여주었다.

Park_count 변수의 유의수준이 높은데 이를 제거했을 경우, R 스퀘어 값이 떨어져 그대로 내버려뒀다.

회귀 모델을 통한 LSCP 최대 허용거리 선정

도로명 건물을 반영한 LSCP_300 데이터 프레임

```
[104] lscp_300f
```

	Bike_Station	Loan_count	Return_count	Population	Sub_pop	Bus_pop	DN_avg	Road_yes	Space_count	Culture_count	Sport_count	KINTEX	Hosu_yes	Park_count	BDTYP_CD_01003	BDTYP_CD_02003	BDTYP_CD_03999	BDTYP_CD_03005
0	101	8366	9207	19019	0	142127	17	1	419	0	0	0	0	0	9	11	37	3
1	103	4535	5121	23225	0	37794	22	1	0	0	0	0	0	1	37	10	5	0
2	104	13402	9330	30358	0	53381	35	0	0	0	0	0	0	0	0	0	1	0
3	105	2806	2696	24876	0	136789	23	1	349	0	0	0	0	0	60	68	15	3
4	106	4444	2357	11356	8607849	257783	26	1	294	0	0	0	0	0	12	54	7	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
159	350	4190	3662	16480	0	29430	25	0	0	0	0	0	0	0	0	0	4	0
160	351	473	406	21769	0	25007	38	0	0	0	0	0	0	0	0	0	0	0
161	352	0	0	16939	0	25959	13	1	282	0	0	0	0	1	0	0	1	0
162	353	0	0	16907	0	65082	14	1	0	1	0	0	0	0	0	0	1	0
163	992	952	795	20957	0	265812	15	1	685	0	0	0	0	1	66	0	2	1

164 rows × 234 columns

기본 베이스 모델에 도로명 건물을 반영한다면 모델의 설명력이 높아질 것이라 생각해

도로명 건물을 반영한 전체 칼럼을 통해 다변량 회귀 분석 모델을 설계해보았다.

회귀 모델을 통한 LSCP 최대 허용거리 선정

도로명 건물을 반영한 LSCP_300의 다변량 회귀 모델

```
# 대여 + 반납 lscp_300
result = sm.ols(formula = 'Return_count ~ Bike_Station+Population+Sub_pop+DN_avg+Road_yes+Space_count+Culture_count+Hosu_yes+Park_count+BDTYP_CD_01003+I
result.summary()
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/base/model.py:1294: RuntimeWarning: invalid value encountered in true_divide
  return self.params / self.bse
/usr/local/lib/python3.6/dist-packages/scipy/stats/_distn_infrastructure.py:903: RuntimeWarning: invalid value encountered in greater
  return (a < x) & (x < b)
/usr/local/lib/python3.6/dist-packages/scipy/stats/_distn_infrastructure.py:903: RuntimeWarning: invalid value encountered in less
  return (a < x) & (x < b)
/usr/local/lib/python3.6/dist-packages/scipy/stats/_distn_infrastructure.py:1912: RuntimeWarning: invalid value encountered in less_equal
  cond2 = cond0 & (x <= _a)
```

OLS Regression Results

Dep. Variable:	Return_count	**R-squared:**	0.998
Model:	OLS	**Adj. R-squared:**	0.936
Method:	Least Squares	**F-statistic:**	16.16
Date:	Thu, 10 Sep 2020	**Prob (F-statistic):**	0.00259
Time:	18:20:00	**Log-Likelihood:**	-1318.7
No. Observations:	164	**AIC:**	2955.
Df Residuals:	5	**BIC:**	3448.
Df Model:	158		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2.703e+04	2.07e+04	1.307	0.248	-2.61e+04	8.02e+04
Bike_Station	-63.7516	46.036	-1.385	0.225	-182.092	54.589
Population	0.9767	0.429	2.277	0.072	-0.126	2.079
Sub_pop	0.0150	0.017	0.887	0.416	-0.028	0.058
DN_avg	-536.5526	553.962	-0.969	0.377	-1960.559	887.453
Road_yes	-5612.3111	1.14e+04	-0.494	0.642	-3.48e+04	2.36e+04
Space_count	-117.1177	61.670	-1.899	0.116	-275.645	41.410
Culture_count	-3599.3050	1.05e+04	-0.341	0.747	-3.07e+04	2.35e+04
Hosu_yes	8.363e+04	2.54e+04	3.297	0.022	1.84e+04	1.49e+05
Park_count	-6078.2664	4203.941	-1.446	0.208	-1.69e+04	4728.307
BDTYP_CD_01003	58.5094	209.788	0.279	0.791	-480.768	597.787
BDTYP_CD_02003	-956.5816	544.072	-1.758	0.139	-2355.162	441.999
BDTYP_CD_03999	-1490.4956	829.927	-1.796	0.132	-3623.890	642.899
BDTYP_CD_03005	1175.4900	2248.321	0.523	0.623	-4604.004	6954.984

그 결과, R스퀘어 값이 0.998, adj R스퀘어 값이 0.936으로 높게 나왔다.
그러나, 각각의 변수들의 유의수준이 천차만별이라 이 또한 변수 선택이 필요하다고 판단했다.

회귀 모델을 통한 LSCP 최대 허용거리 선정

STEPWISE로 적합시킨 도로명 건물 LSCP_300의 다변량 회귀 모델

```
# 대여 + 반납 lscp_300
result = sm.ols(formula = 'Return_count ~ Bike_Station+Population+Sub_pop+DN_avg+Road_yes+Space_count+Hosu_yes+Park_count+BDTYP_CD_01003+
result.summary()
```

OLS Regression Results

Dep. Variable:	Return_count	**R-squared:**	0.996
Model:	OLS	**Adj. R-squared:**	0.985
Method:	Least Squares	**F-statistic:**	83.51
Date:	Thu, 10 Sep 2020	**Prob (F-statistic):**	2.25e-28
Time:	18:20:01	**Log-Likelihood:**	-1366.6
No. Observations:	164	**AIC:**	2987.
Df Residuals:	37	**BIC:**	3381.
Df Model:	126		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2.3e+04	3034.392	7.578	0.000	1.68e+04	2.91e+04
Bike_Station	-50.1798	5.924	-8.470	0.000	-62.184	-38.176
Population	0.8898	0.094	9.427	0.000	0.699	1.081
Sub_pop	0.0147	0.001	13.895	0.000	0.013	0.017
DN_avg	-449.0859	51.150	-8.780	0.000	-552.725	-345.447
Road_yes	-6078.1845	1412.640	-4.303	0.000	-8940.466	-3215.903
Space_count	-114.8601	6.994	-16.423	0.000	-129.031	-100.689
Hosu_yes	8.447e+04	3570.152	23.659	0.000	7.72e+04	9.17e+04
Park_count	-5058.8948	625.287	-8.091	0.000	-6325.846	-3791.944
BDTYP_CD_01003	68.3760	17.095	4.000	0.000	33.738	103.014
BDTYP_CD_02003	-920.9403	69.275	-13.294	0.000	-1061.304	-780.576
BDTYP_CD_03999	-1309.4630	207.152	-6.321	0.000	-1729.193	-889.733
BDTYP_CD_01001	42.0622	6.881	6.113	0.000	28.121	56.004
BDTYP_CD_13100	-1.589e+04	1918.501	-8.285	0.000	-1.98e+04	-1.2e+04
BDTYP_CD_16006	8.73e+04	5076.309	17.197	0.000	7.7e+04	9.76e+04
BDTYP_CD_03001	-1893.0664	182.835	-10.354	0.000	-2263.526	-1522.607
BDTYP_CD_17007	3249.9432	669.630	4.853	0.000	1893.145	4606.741
BDTYP_CD_04001	3123.2351	272.855	11.447	0.000	2570.379	3676.092
BDTYP_CD_14999	-952.3046	65.559	-14.526	0.000	-1085.139	-819.470

여기서도 STEPWISE 방법론을 활용하여 변수들을 더하고 빼가며 최적의 모델을 찾았다.

R스퀘어 값이 0.996으로 조금 떨어졌지만 adj R스퀘어 값이 0.985 수준으로 높게 나왔다.

그러나, 각각의 변수들의 유의수준도 일정 수준에서 컨트롤된 모습을 볼 수 있다.

회귀 모델을 통한 LSCP 최대 허용거리 선정

다중 공선성(VIF) 체크

```
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(lscp_300f.values, i) for i in range(lscp_300f.shape[1])]
vif["features"] = lscp_300f.columns
vif
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/regression/linear_model.py:1638: RuntimeWarning: invalid value encountered in double_scalars
  return 1 - self.ssr/self.uncentered_tss
/usr/local/lib/python3.6/dist-packages/statsmodels/stats/outliers_influence.py:185: RuntimeWarning: divide by zero encountered in double_scalars
  vif = 1. / (1. - r_squared_i)
```

	VIF Factor	features
0	3144.984854	Bike_Station
1	30514.249732	Loan_count
2	17237.122032	Return_count
3	3207.491959	Population
4	25572.048662	Sub_pop
...	...	...
229	NaN	BDTYP_CD_08699
230	464.068739	BDTYP_CD_03109
231	NaN	BDTYP_CD_27999
232	2394.211712	BDTYP_CD_90002
233	NaN	BDTYP_CD_05502

234 rows × 2 columns

하지만, 이 모델을 직접 예측에까지 활용하기에는 문제가 존재했다. 바로 다중 공선성 문제인데, 각 변수들의 상호 작용 때문에 각 변수들이 상호 독립적이라는 가설이 무너져 모델을 활용할 수 없게 된 것이다.

하지만, 좌표의 영향권 거리가 300m일 때가 가장 유효했다는 점은 얻어갈 수 있었다.

5. 자전거 스테이션의 영향권 설정 코드 설명

(2) 계획 데이터를 통한 LSCP 데이터 추합

계획 데이터를 통한 LSCP 데이터 추합

일전에 만든 Multipolygon 내 데이터 추합 함수 활용

Multipolygon

- geometry type이 Multipolygon인 데이터프레임에 대하여 LSCP 데이터프레임 생성

[12]
```
import math
```

[13]
```
# 300m 기준
lscp_distance=300
```

```
def geo_dis(Bike_Station, target):
  lscp=[]
  target_nm=target.columns.tolist()[[(t.find('NAME')>=0) | (t.lower().find('dgm_nm')>=0) |
                                     (t.lower().find('bdtyp_cd')>=0) | (t.lower().find('mnum')>=0) for t in target.columns.tolist()].index(True)]
  for sta, lat, long in zip(Bike_Station['Station_ID'],Bike_Station['위도'],Bike_Station['경도']):
    for nm, geometry in zip(target[target_nm],target['geometry']):
      h=[]
      x = (lat, long)
      points = [point for polygon in geometry for point in polygon.exterior.coords[:]]
# Bike_Station 기준으로 해당 멀티폴리곤의 모든 점과의 거리를 구함
      for t_long, t_lat in points:
        y=(t_lat, t_long)
        h.append(haversine(x,y, unit='m'))
# 만약 그 거리가 LSCP 기준 보다 낮으면 반환.
      if min(h)<=lscp_distance:
        lscp.append([sta,nm,min(h)])
  return lscp
```

계획 데이터를 통한 LSCP 데이터 추합

도시 계획(공간 시설)의 분류 세분화

분류 세분화 space_plan

- 단순히 space_plan 만으로 나눌 게 아니라 space_plan 내 분류를 활용한 세분화 작업

도시 계획(공간 시설)의 UQT이하 3자리를 활용한 분류

```
space_label['UQT_m'].unique()
```

```
array(['UQT330', 'UQT210', 'UQT310', 'UQT320', 'UQT250', 'UQT220',
       'UQT260', 'UQT500', 'UQT100', 'UQT290', 'UQT200', 'UQT120',
       'UQT110', 'UQT390', 'UQT520', 'UQT221'], dtype=object)
```

```
[29] # 분류마다 작업을 반복하기 위해 자동화 함수를 짬.
     def space_maker(space_label,UQT):
     # UQT 값을 기준으로 새로운 데이터 프레임 생성
       space_all=pd.DataFrame(geo_dis(Bike_Station,space_label[space_label['UQT_m']=='UQT{}'.format(UQT)]),columns=['Bike_Station','MNUS','Harv_dis'])
       space_all['{}_all_count'.format(UQT)]=1
       space_all=space_all[['Bike_Station','{}_all_count'.format(UQT)]].groupby('Bike_Station').agg('sum').reset_index()
       space_all.columns=['Bike_Station','{}_all_count'.format(UQT)]
       space_all=lscp_base.merge(space_all[['Bike_Station','{}_all_count'.format(UQT)]],on='Bike_Station',how='outer').fillna(0).drop_duplicates().reset_index(drop=True)
       space_all['{}_all_count'.format(UQT)]=space_all['{}_all_count'.format(UQT)].astype(int)

       space_real=pd.DataFrame(geo_dis(Bike_Station,space_label[(space_label['UQT_m']=='UQT{}'.format(UQT))&(space_label['라벨']==1.0)]),columns=['Bike_Station','MNUS','Harv_dis'])
       space_real['{}_real_count'.format(UQT)]=1
       space_real=space_real[['Bike_Station','{}_real_count'.format(UQT)]].groupby('Bike_Station').agg('sum').reset_index()
       space_real.columns=['Bike_Station','{}_real_count'.format(UQT)]
       space_real=lscp_base.merge(space_real[['Bike_Station','{}_real_count'.format(UQT)]],on='Bike_Station',how='outer').fillna(0).drop_duplicates().reset_index(drop=True)
       space_real['{}_real_count'.format(UQT)]=space_real['{}_real_count'.format(UQT)].astype(int)

       return space_all, space_real
```

계획 데이터를 통한 LSCP 데이터 추합

도시 계획(공공문화체육시설)의 분류 세분화

culphy_plan 세분화

```
[35] culphy_plan['DGM_NM'].unique()

array(['기타 사회복지시설', '초등학교', '유치원', '기타 체육시설', '중학교', '기타 도서관시설', '청사(국가)',
       '고등학교', '공공체육시설', '기타 공공청사시설', '특수학교', '각종학교', '기타 공공직업훈련시설', '대학',
       '기타 운동장시설', '지방자치단체', '골프장', '기타 연구시설', '기타청소년수련시설', '기타 문화시설',
       '공관', '박물관', '기타 학교시설', '종합운동장'], dtype=object)
```

도시 계획(공공문화체육시설)의 DGM_NM을 활용한 분류

```
# 분류마다 작업을 반복하기 위해 자동화 함수를 짬.
def culphy_maker(culphy_label,DGM):
# 여기선 DGM_NM 기준
  culphy_all=pd.DataFrame(geo_dis(Bike_Station,culphy_label[culphy_label['DGM_NM']==DGM]),columns=['Bike_Station','MNUS','Harv_dis'])
  culphy_all['{}_all_count'.format(DGM)]=1
  culphy_all=culphy_all[['Bike_Station','{}_all_count'.format(DGM)]].groupby('Bike_Station').agg('sum').reset_index()
  culphy_all.columns=['Bike_Station','{}_all_count'.format(DGM)]
  culphy_all=lscp_base.merge(culphy_all[['Bike_Station','{}_all_count'.format(DGM)]],on='Bike_Station',how='outer').fillna(0).drop_duplicates().reset_index(drop=True)
  culphy_all['{}_all_count'.format(DGM)]=culphy_all['{}_all_count'.format(DGM)].astype(int)

  culphy_real=pd.DataFrame(geo_dis(Bike_Station,culphy_label[(culphy_label['DGM_NM']==DGM)&(culphy_label['라벨']==1.0)]),columns=['Bike_Station','MNUS','Harv_dis'])
  culphy_real['{}_real_count'.format(DGM)]=1
  culphy_real=culphy_real[['Bike_Station','{}_real_count'.format(DGM)]].groupby('Bike_Station').agg('sum').reset_index()
  culphy_real.columns=['Bike_Station','{}_real_count'.format(DGM)]
  culphy_real=lscp_base.merge(culphy_real[['Bike_Station','{}_real_count'.format(DGM)]],on='Bike_Station',how='outer').fillna(0).drop_duplicates().reset_index(drop=True)
  culphy_real['{}_real_count'.format(DGM)]=culphy_real['{}_real_count'.format(DGM)].astype(int)

  return culphy_all, culphy_real
```

계획 데이터를 통한 LSCP 데이터 추합

도시 계획(교통 시설)의 분류 세분화

도시 계획(교통 시설)의

DGM_NM을 활용한 분류

traffic_plan 세부 분류

```
[42]  traffic_plan['DGM_NM'].unique()

      array(['노외주차장', '공영차고지', '기타 주차장시설', '일반철도', '기타철도시설', '도시철도', '여객자동차터미널'],
            dtype=object)
```

```
[43]  # culphy 데이터와 구조가 동일하기 때문에 같은 함수를 사용
      traffic_a_all, traffic_a_real = culphy_maker(traffic_label,'노외주차장')
      traffic_b_all, traffic_b_real = culphy_maker(traffic_label,'공영차고지')
      traffic_c_all, traffic_c_real = culphy_maker(traffic_label,'기타 주차장시설')
      traffic_d_all, traffic_d_real = culphy_maker(traffic_label,'일반철도')
      traffic_e_all, traffic_e_real = culphy_maker(traffic_label,'기타철도시설')
      traffic_f_all, traffic_f_real = culphy_maker(traffic_label,'도시철도')
      traffic_g_all, traffic_g_real = culphy_maker(traffic_label,'여객자동차터미널')
```

계획 데이터를 통한 LSCP 데이터 추합

스테이션 기준 계획 데이터 반영 데이터 프레임(실제)

– 계획 데이터에서 타 데이터 프레임과 겹치는 값들의 수를 반환함

```
[185] build_real_final
```

	Bike_Station	100_real_count	110_real_count	120_real_count	200_real_count	210_real_count	220_real_count	221_real_count	250_real_count	260_real_count	290_real_count
0	101	0	0	0	0	1	1	0	0	0	0
1	103	0	0	0	0	0	0	0	1	0	0
2	104	0	0	0	0	1	1	0	0	0	0
3	105	0	0	0	0	1	1	0	0	0	0
4	106	0	0	0	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
159	350	0	0	0	0	0	1	0	0	0	0
160	351	0	0	0	1	0	1	0	0	0	0
161	352	0	0	0	0	0	2	0	0	0	0
162	353	0	0	0	0	0	2	0	0	0	0
163	992	1	0	0	0	2	1	0	0	0	0

164 rows × 47 columns

계획 데이터를 통한 LSCP 데이터 추합

스테이션 기준 계획 데이터 반영 데이터 프레임(전체)

– 실제 + 계획

```
[184] build_all_final
```

	Bike_Station	100_all_count	110_all_count	120_all_count	200_all_count	210_all_count	220_all_count	221_all_count	250_all_count	260_all_count	290_all_count	310_all_count
0	101	1	0	0	0	2	1	0	0	0	0	2
1	103	0	0	0	1	2	0	0	1	0	0	3
2	104	0	0	0	0	1	1	0	0	0	0	0
3	105	0	0	0	0	1	1	0	0	0	0	1
4	106	0	0	0	0	0	1	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...
159	350	0	0	0	0	3	1	0	0	0	0	3
160	351	0	0	0	1	2	1	0	0	0	0	0
161	352	1	0	0	1	0	4	0	0	0	0	1
162	353	1	0	0	0	0	3	0	0	0	0	2
163	992	2	0	0	0	2	1	0	0	0	0	4

164 rows × 47 columns

모델 적합을 위한 계획 LSCP 데이터 추합

100m 그리드 좌표 데이터를 불러옴

```
[196] lscp_pop_300=pd.read_csv('/content/drive/My Drive/compas/csv 파일 여기로!/lscp_pop_300.csv')
      mapper = lambda x: float(x.strip('₩(').strip('₩)').split(',')[0])
      lscp_pop_300['위도'] = lscp_pop_300['좌표'].map(mapper)

      mapper = lambda x: float(x.strip('₩(').strip('₩)').split(',')[1])
      lscp_pop_300['경도'] = lscp_pop_300['좌표'].map(mapper)
```

```
[197] lscp_pop_300
```

	좌표	주거인구	버스_승하차수	지하철_승하차수	유동인구	위도	경도
0	(37.57961579548528, 126.87506406643347)	12.0	53.0	0.0	53.0	37.579616	126.875064
1	(37.57963617229605, 126.87619583846381)	12.0	0.0	0.0	0.0	37.579636	126.876196
2	(37.57965653826528, 126.87732761163377)	12.0	0.0	0.0	0.0	37.579657	126.877328
3	(37.58049614012486, 126.87390667102693)	12.0	61.0	0.0	61.0	37.580496	126.873907
4	(37.5805165284361, 126.8750384555327)	12.0	61.0	0.0	61.0	37.580517	126.875038
...	...	...	...	...	...	...	...
20832	(37.74702008221923, 126.91341083764837)	0.0	14.0	0.0	14.0	37.747020	126.913411
20833	(37.747040166676285, 126.91454518217792)	0.0	14.0	0.0	14.0	37.747040	126.914545
20834	(37.74706024022562, 126.9156795278339)	0.0	14.0	0.0	14.0	37.747060	126.915680
20835	(37.60951982027619, 126.83456514393055)	0.0	0.0	5568132.0	5568132.0	37.609520	126.834565
20836	(37.64611738424386, 126.91510462769811)	0.0	0.0	1182838.0	1182838.0	37.646117	126.915105

20837 rows × 7 columns

모델 적합을 위한 계획 LSCP 데이터 추합

MULTIPOLYGON을 점으로 치환하기 위한 함수

```
[200] # lscp_distance 기준으로 스테이션과 구분을 위해 필요한 분류명 뽑아주기
      def geo_dis3(Bike_Station, target):
        lscp=[]
        target_nm=target.columns.tolist()[[(t.find('NAME')>=0) | (t.lower().find('dgm_nm')>=0) |
                                           (t.lower().find('bdtyp_cd')>=0) | (t.lower().find('mnum')>=0) for t in target.columns.tolist()].index(True)]
        for sta, lat, long in zip(Bike_Station['좌표'],Bike_Station['위도'],Bike_Station['경도']):
          for nm, geometry in zip(target[target_nm],target['geometry']):
            h=[]
            x = (lat, long)
            points = [point for polygon in geometry for point in polygon.exterior.coords[:]]
            for t_long, t_lat in points:
              y=(t_lat, t_long)
              h.append(haversine(x,y, unit='m'))
            if min(h)<=lscp_distance:
              lscp.append([sta,nm,min(h)])
        return lscp
```

모델 적합을 위한 계획 LSCP 데이터 추합

UQT값을 기반으로 계획 데이터 분류

(공간 시설)

```
[201] # space_plan 에 적용 가능한 자동화 메서드 작성
      def space_maker(space_label,UQT):
        space_all=pd.DataFrame(geo_dis3(lscp_pop_300,space_label[space_label['UQT_m']=='UQT{}'.format(UQT)]),columns=['좌표','MNUS','Harv_dis'])
        space_all['{}_all_count'.format(UQT)]=1
        space_all=space_all[['좌표','{}_all_count'.format(UQT)]].groupby('좌표').agg('sum').reset_index()
        space_all.columns=['좌표','{}_all_count'.format(UQT)]
        space_all=location_base.merge(space_all[['좌표','{}_all_count'.format(UQT)]],on='좌표',how='outer').fillna(0).drop_duplicates().reset_index(drop=True)
        space_all['{}_all_count'.format(UQT)]=space_all['{}_all_count'.format(UQT)].astype(int)

        space_real=pd.DataFrame(geo_dis3(lscp_pop_300,space_label[(space_label['UQT_m']=='UQT{}'.format(UQT))&(space_label['라벨']==1.0)]),columns=['좌표','MNUS','Harv_dis'])
        space_real['{}_real_count'.format(UQT)]=1
        space_real=space_real[['좌표','{}_real_count'.format(UQT)]].groupby('좌표').agg('sum').reset_index()
        space_real.columns=['좌표','{}_real_count'.format(UQT)]
        space_real=location_base.merge(space_real[['좌표','{}_real_count'.format(UQT)]],on='좌표',how='outer').fillna(0).drop_duplicates().reset_index(drop=True)
        space_real['{}_real_count'.format(UQT)]=space_real['{}_real_count'.format(UQT)].astype(int)

        return space_all, space_real
```

모델 적합을 위한 계획 LSCP 데이터 추합

DGM_NM을 기반으로 계획 데이터 분류

(공공문화체육시설, 공공 교통)

```
# culphy_plan 및 traffic_plan에 적용 가능한 자동화 메서드 작성
def culphy_maker(culphy_label,DGM):
  culphy_all=pd.DataFrame(geo_dis3(lscp_pop_300,culphy_label[culphy_label['DGM_NM']==DGM]),columns=['좌표','MNUS','Harv_dis'])
  culphy_all['{}_all_count'.format(DGM)]=1
  culphy_all=culphy_all[['좌표','{}_all_count'.format(DGM)]].groupby('좌표').agg('sum').reset_index()
  culphy_all.columns=['좌표','{}_all_count'.format(DGM)]
  culphy_all=location_base.merge(culphy_all[['좌표','{}_all_count'.format(DGM)]],on='좌표',how='outer').fillna(0).drop_duplicates().reset_index(drop=True)
  culphy_all['{}_all_count'.format(DGM)]=culphy_all['{}_all_count'.format(DGM)].astype(int)

  culphy_real=pd.DataFrame(geo_dis3(lscp_pop_300,culphy_label[(culphy_label['DGM_NM']==DGM)&(culphy_label['라벨']==1.0)]),columns=['좌표','MNUS','Harv_dis'])
  culphy_real['{}_real_count'.format(DGM)]=1
  culphy_real=culphy_real[['좌표','{}_real_count'.format(DGM)]].groupby('좌표').agg('sum').reset_index()
  culphy_real.columns=['좌표','{}_real_count'.format(DGM)]
  culphy_real=location_base.merge(culphy_real[['좌표','{}_real_count'.format(DGM)]],on='좌표',how='outer').fillna(0).drop_duplicates().reset_index(drop=True)
  culphy_real['{}_real_count'.format(DGM)]=culphy_real['{}_real_count'.format(DGM)].astype(int)

  return culphy_all, culphy_real
```

모델 적합을 위한 계획 LSCP 데이터 추합

좌표 기준 계획 데이터 프레임 완성

```
location_all_final
```

	좌표	100_all_count	110_all_count	120_all_count	200_all_count	210_all_count	220_all_count	221_all_count	250_all_count	260_all_count	290_all_count
0	(37.57961579548528, 126.87506406643347)	0	0	0	0	0	0	0	0	0	0
1	(37.57963617229605, 126.87619583846381)	0	0	0	0	0	0	0	0	0	0
2	(37.57965653826528, 126.87732761163377)	0	0	0	0	0	0	0	0	0	0
3	(37.58049614012486, 126.87390667102693)	0	0	0	0	0	0	0	0	0	0
4	(37.5805165284361, 126.8750384555327)	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
20832	(37.74702008221923, 126.91341083764837)	0	0	0	0	0	0	0	0	0	0
20833	(37.747040166676285, 126.91454518217792)	0	0	0	0	0	0	0	0	0	0
20834	(37.74706024022562, 126.9156795278339)	0	0	0	0	0	0	0	0	0	0
20835	(37.60951982027619, 126.83456514393055)	0	0	0	0	0	0	0	0	0	0
20836	(37.64611738424386, 126.91510462769811)	2	0	0	0	2	1	0	0	0	0

20837 rows × 47 columns

6. 결과 부분 코드 설명

(1) 스테이션 간 직선거리 확인

스테이션 간 직선거리 확인

특정 스테이션부터 전 스테이션 간 직선거리 확인

```
[6]  # 스테이션 간 직선 거리를 구함 (스테이션 간의 거리에 대한 정보를 얻고자 함)
     station_to_station=[]
     for sta_id, sta_nm, _,lat,long in np.array(bike_station):
       for sta_id2, sta_nm2, _,lat2,long2 in np.array(bike_station):
         station_h=haversine((lat,long),(lat2,long2),unit='m')
         station_to_station.append([sta_id,sta_id2,station_h])
     station_distance=pd.DataFrame(station_to_station,columns=['sta_rent','sta_return','hav_dis'])
```

```
station_distance
```

	sta_rent	sta_return	hav_dis
0	101	101	0.000000
1	101	103	810.593473
2	101	104	7749.401651
3	101	105	414.953376
4	101	106	715.798911
...	...	...	...
26891	992	350	9768.976190
26892	992	351	9278.739241
26893	992	352	8104.699800
26894	992	353	7993.663782
26895	992	992	0.000000

26896 rows × 3 columns

적정 스테이션 거리를 설정해 추후에 함수 활용 시 스테이션들의 좌표가 근접하게 설정되는 것을 방지

+) Haversine 함수 활용

스테이션 간 직선거리 확인

직선거리의 분포 확인

```
[8]  # 거리가 0인 걸 제외하고 describe
     station_distance[station_distance['hav_dis']!=0]['hav_dis'].describe()
```

```
count    26732.000000
mean      5639.034330
std       3579.687975
min         32.629005
25%       2694.246705
50%       4935.346588
75%       8193.268818
max      18515.910859
Name: hav_dis, dtype: float64
```

모든 스테이션 간 직선거리라 값이 높이 뛰는 것을 확인

500m 제한을 둔 분포가 좀 더

실제에 가까울 것으로 판단

```
[9]  # 거리 500m 내 직선 거리 describe
     station_distance[(station_distance['hav_dis']!=0)&(station_distance['hav_dis']<=500)]['hav_dis'].describe()
```

```
count    404.000000
mean     362.559699
std      104.257819
min       32.629005
25%      308.710097
50%      387.163451
75%      445.003856
max      499.876754
Name: hav_dis, dtype: float64
```

평균 362m, 중앙값 387m를 확인.

실제로 362m 보다 가까이 붙은 스테이션들이 있어 데이터를 직접 봐야할 필요성 발생.

스테이션 간 직선거리 확인

실제 데이터 확인

```
# 직선 거리가 작은 것부터 리턴
station_distance[(station_distance['hav_dis']!=0)&(station_distance['hav_dis']<=500)].sort_values('hav_dis')[:50]
```

1817	116	119	147.933379
2143	119	116	147.933379
24910	341	331	160.312901
24095	331	341	160.312901
9241	201	202	167.305705
9404	202	201	167.305705
9734	204	203	167.494818
9571	203	204	167.494818
17821	257	258	172.712591
17984	258	257	172.712591
2306	121	115	183.360442
1654	115	121	183.360442
17323	254	252	216.902861
16997	252	254	216.902861
1977	118	114	217.371165
1488	114	118	217.371165
26741	992	114	217.832058
1639	114	992	217.832058
991	111	112	218.160733
1154	112	111	218.160733
26233	350	348	223.112547
25907	348	350	223.112547
25741	347	348	224.753038
25904	348	347	224.753038
22403	320	248	225.381386
16372	248	320	225.381386

최저값부터 50개를 뽑아본 결과

보통 200m내외로 수렴함을 확인

6. 결과 부분 코드 설명

(2) 동 단위 데이터 소팅

동 단위 데이터 소팅

행정경계(읍면동) 데이터 활용

동 단위로 데이터 소팅

- 각 동에 필요한 자전거 스테이션 개수를 파악하고 그에 따라 스테이션 위치를 선정하기 위함.

```
[11] border=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/09.행정경계(읍면동).geojson')
```

```
border
```

	EMD_CD	EMD_KOR_NM	geometry
0	41281101	주교동	MULTIPOLYGON (((126.81068 37.65820, 126.81069 ...
1	41281102	원당동	MULTIPOLYGON (((126.83321 37.68013, 126.83340 ...
2	41281103	신원동	MULTIPOLYGON (((126.86362 37.67729, 126.86364 ...
3	41281104	원흥동	MULTIPOLYGON (((126.85975 37.65081, 126.85990 ...
4	41281105	도내동	MULTIPOLYGON (((126.84806 37.62348, 126.84835 ...
5	41281106	성사동	MULTIPOLYGON (((126.83047 37.64860, 126.83053 ...
6	41281107	북한동	MULTIPOLYGON (((126.95320 37.65582, 126.95334 ...
7	41281108	효자동	MULTIPOLYGON (((126.94756 37.65922, 126.94777 ...
8	41281109	지축동	MULTIPOLYGON (((126.89619 37.65389, 126.89627 ...
9	41281110	오금동	MULTIPOLYGON (((126.89197 37.67066, 126.89222 ...
10	41281111	삼송동	MULTIPOLYGON (((126.87369 37.64345, 126.87385 ...

동 단위로 적정 자전거 스테이션 수를 설정하는 것이 좌표 만으로 적정 자전거 스테이션 수를 설정하는 것보다 전반적인 자전거 서비스 수요를 높이는데 유효할 것이라 판단.

동 단위 데이터 소팅

실제 좌표 기준 수요값을 불러옴

```
[14] model_real_demand
```

	좌표	대여_월평균	반납_월평균	총 이용_월평균	스테이션	위도	경도	Hosu_yes	CD_10299_기타일반업무시설	CD_11299_기타관광숙박시설	지하철_승하차수	CD_90001	Road_yes	CD_05501_박물관	CD_05999_기타문화및집회시설	CD_03004_일반목욕장
0	(37.59052639593674, 126.88041634904563)	0.00	0.000000	0.000000	[155]	37.590526	126.880416	0	0	0	0.0	0	1	0	0	0
1	(37.590546726423185, 126.88154829019997)	0.00	0.000000	0.000000	[155]	37.590547	126.881548	0	0	0	0.0	0	1	0	0	0
2	(37.59142713034578, 126.88039079257342)	0.00	0.000000	0.000000	[155]	37.591427	126.880391	0	0	0	0.0	0	1	0	0	0
3	(37.59144746148929, 126.88152274735107)	0.00	0.000000	0.000000	[155]	37.591447	126.881523	0	0	0	0.0	0	1	0	0	0
4	(37.59146778178655, 126.88265470326604)	0.00	0.000000	0.000000	[155]	37.591468	126.882655	0	0	0	0.0	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3228	(37.70445091519225, 126.75467703178417)	169.25	127.277778	296.527778	[346]	37.704451	126.754677	0	0	0	0.0	0	0	0	0	0
3229	(37.704472506609626, 126.75581060819373)	169.25	127.277778	296.527778	[346]	37.704473	126.755811	0	0	0	0.0	0	0	0	0	0
3230	(37.70466633943072, 126.7660128502553)	118.25	101.500000	219.750000	[351]	37.704666	126.766013	0	0	0	0.0	0	0	0	0	0
3231	(37.70468782197291, 126.76714643873707)	118.25	101.500000	219.750000	[351]	37.704688	126.767146	0	0	0	0.0	0	0	0	0	0
3232	(37.70470929362739, 126.76828002842278)	118.25	101.500000	219.750000	[351]	37.704709	126.768280	0	0	0	0.0	0	0	0	0	0

3233 rows × 57 columns

동 단위 데이터 소팅

모델 예측 수요값을 불러옴

```
model_expected_demand
```

	좌표	예상 수요	위도	경도	Hosu_yes	CD_10299_기타일반업무시설	CD_11299_기타관광숙박시설	지하철_승하차수	CD_90001	Road_yes	CD_05501_박물관	CD_05999_기타문화및집회시설	CD_03004_일반목욕장	CD_02003_다세대주택
0	(37.57961579548528, 126.87506406643347)	319.321900	37.579616	126.875064	0	0	0	0.0	0	1	0	0	0	0
1	(37.57963617229605, 126.87619583846381)	268.645420	37.579636	126.876196	0	0	0	0.0	0	1	0	0	0	0
2	(37.57965653826528, 126.87732761163377)	268.645420	37.579657	126.877328	0	0	0	0.0	0	1	0	0	0	0
3	(37.58049614012486, 126.87390667102693)	373.882050	37.580496	126.873907	0	0	0	0.0	0	1	0	0	0	0
4	(37.5805165284361, 126.8750384555327)	474.046330	37.580517	126.875038	0	0	0	0.0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17599	(37.74702008221923, 126.91341083764837)	49.397846	37.747020	126.913411	0	0	0	0.0	0	0	0	0	0	0
17600	(37.747040166676285, 126.91454518217792)	49.397846	37.747040	126.914545	0	0	0	0.0	0	0	0	0	0	0
17601	(37.74706024022562, 126.9156795278339)	54.330900	37.747060	126.915680	0	0	0	0.0	0	0	0	0	0	0
17602	(37.60951982027619, 126.83456514393055)	356.763920	37.609520	126.834565	0	0	0	5568132.0	0	0	0	0	0	0
17603	(37.64611738424386, 126.91510462769811)	461.099730	37.646117	126.915105	0	1	0	1182838.0	0	0	0	0	0	0

17604 rows × 54 columns

동 단위 데이터 소팅

실제 수요와 예측 수요값을 합쳐준 후, 자전거 스테이션 인접 유무를 표시

```
[16] # 모델에 따른 수요를 보기 위해 실제 스테이션 수요값과 예측 수요값 데이터를 하나로 합쳐주기 위해 각 수요를 나타낸 칼럼들을 demand로 통일.
     model_real_demand['demand']=model_real_demand['총 이용_월평균']
     model_expected_demand['demand']=model_expected_demand['예상 수요']
     # 해당 좌표 가까이(300m)에 자전거 스테이션이 있다면 1 아니면 0으로 표시
     model_real_demand['bike_station']=1
     model_expected_demand['bike_station']=0
```

```
[17] # 데이터 병합(실제 수요 + 예측 수요)
     model_demand_all=pd.concat([model_real_demand[['좌표','demand','bike_station']],model_expected_demand[['좌표','demand','bike_station']]]).reset_index(drop=True)
```

```
[18] # 모델을 불러올 때 들어있는 좌표가 str 타입이라 활용하기 편하게 Point로 바꿔줌.
     model_demand_all['좌표']=model_demand_all['좌표'].apply(lambda x : Point(float(x[(x.find(',')+1):-1]),float(x[(x.find('(')+1):x.find(',')])))
```

```
[19] # 결과
     model_demand_all
```

	좌표	demand	bike_station
0	POINT (126.8804163490456 37.59052639593674)	0.000000	1
1	POINT (126.8815482902 37.59054672642318)	0.000000	1
2	POINT (126.8803907925734 37.59142713034578)	0.000000	1
3	POINT (126.8815227473511 37.59144746148929)	0.000000	1
4	POINT (126.882654703266 37.59146778178655)	0.000000	1
...	...	...	...
20832	POINT (126.9134108376484 37.74702008221923)	49.397846	0
20833	POINT (126.9145451821779 37.74704016667629)	49.397846	0
20834	POINT (126.9156795278339 37.74706024022562)	54.330900	0
20835	POINT (126.8345651439305 37.60951982027619)	356.763920	0
20836	POINT (126.9151046276981 37.64611738424386)	461.099730	0

20837 rows × 3 columns

동 단위 데이터 소팅

동 데이터 기준 병합을 위해 각 좌표가 어느 동에 위치하는지 표시

```
# 좌표 값을 동 데이터 기준으로 병합하기 위해 EMD_KOR_NM을 붙여줌.
#demand_to_border
model_demand_all['EMD_KOR_NM']=''
for i in tqdm(model_demand_all.index):
    for j in border.index:
# 해당 좌표가 특정 동 geometry 안에 있으면 특정 동 이름을 입력
        if model_demand_all['좌표'][i].within(border.geometry[j]):
          model_demand_all['EMD_KOR_NM'][i]=border['EMD_KOR_NM'][j]
```

```
100% |████████████████████████████████| 20837/20837 [01:51<00:00, 187.26it/s]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
model_demand_all
```

	좌표	demand	bike_station	EMD_KOR_NM
0	POINT (126.8804163490456 37.59052639593674)	0.000000	1	덕은동
1	POINT (126.8815482902 37.59054672642318)	0.000000	1	덕은동
2	POINT (126.8803907925734 37.59142713034578)	0.000000	1	덕은동
3	POINT (126.8815227473511 37.59144746148929)	0.000000	1	덕은동
4	POINT (126.882654703266 37.59146778178655)	0.000000	1	덕은동
...	...	...	...	...
20832	POINT (126.9134108376484 37.74702008221923)	49.397846	0	벽제동
20833	POINT (126.9145451821779 37.74704016667629)	49.397846	0	벽제동
20834	POINT (126.9156795278339 37.74706024022562)	54.330900	0	벽제동
20835	POINT (126.8345651439305 37.60951982027619)	356.763920	0	강매동
20836	POINT (126.9151046276981 37.64611738424386)	461.099730	0	지축동

20837 rows × 4 columns

동 단위 데이터 소팅

동 데이터 기준 병합을 위해 스테이션 데이터프레임에도 동을 표시

```
# station_to_border
# 좌표 값을 동 데이터 기준으로 병합하기 위해 EMD_KOR_NM을 붙여줌.
bike_station['EMD_KOR_NM']=''
for i in tqdm(bike_station.index):
  bike_location=Point(bike_station['경도'][i],bike_station['위도'][i])
  for j in border.index:
# 해당 좌표가 특정 동 geometry 안에 있으면 특정 동 이름을 입력
    if bike_location.within(border.geometry[j]):
      bike_station['EMD_KOR_NM'][i]=border['EMD_KOR_NM'][j]
```

```
100% |████████████████████████████████| 164/164 [00:00<00:00, 333.59it/s]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  if __name__ == '__main__':
```

```
bike_station
```

	Station_ID	STATION_NAME	거치대 수량	위도	경도	EMD_KOR_NM
0	101	어울림마을 701동 앞	20	37.654775	126.834584	성사동
1	103	대림e-편한세상106동	20	37.660442	126.840377	성사동
2	104	탄현마을8단지	25	37.698523	126.766042	탄현동
3	105	KT 덕양지사 앞	20	37.655244	126.839261	성사동
4	106	원당역 앞 공영주차장	30	37.653410	126.842530	성사동
...	...	...	...	...	...	...
159	350	★하이파크5단지 502동앞 버스정류장	20	37.697867	126.753089	덕이동
160	351	◆일산에듀포레 푸르지오	40	37.702259	126.767231	탄현동
161	352	◆꿈에그린203동앞	40	37.666425	126.749244	대화동
162	353	◆꿈에그린106동앞	40	37.666720	126.750784	대화동
163	992	★피프틴센터	4	37.637529	126.833760	화정동

164 rows × 6 columns

6. 최적 스테이션 선정 코드 설명

(3) 동 별 적정 스테이션 수 산출

동 별 적정 스테이션 수 산출

운영 이력 데이터를 불러온 후 월별로 자료를 정리

bike_oh

	LEAS_NO	LEAS_STAT	LEAS_DATE	LEAS_STATION	LEAS_DEF_NO	RTN_DATE	RTN_STATION	RTN_DEF_NO	TRNV_QTY	MEMB_DIV	MEMB_NO	TEMP_MEMB_NO
0	15945541	2	2017-01-01 00:00:41	213	18	2017-01-01 00:13:52	260	17	0.0	6	164203	0.0
1	15945542	2	2017-01-01 00:01:03	231	4	2017-01-01 00:50:24	231	17	31039.0	1	187551	0.0
2	15945543	2	2017-01-01 00:01:50	119	14	2017-01-01 01:01:50	0	0	NaN	12	168994	0.0
3	15945544	2	2017-01-01 00:02:09	121	17	2017-01-01 00:15:58	133	14	15490.0	12	183971	0.0
4	15945545	2	2017-01-01 00:03:32	320	29	2017-01-01 00:18:44	259	27	0.0	12	167475	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
3142683	21718608	2	2019-12-31 23:52:39	222	16	2019-12-31 23:58:56	219	24	0.0	3	236957	0.0
3142684	21718610	2	2019-12-31 23:53:59	343	14	2019-12-31 23:57:59	311	15	0.0	99	0	4124665.0
3142685	21718611	2	2019-12-31 23:55:27	333	17	2020-01-01 00:06:24	322	5	0.0	99	0	4124667.0
3142686	21718613	2	2019-12-31 23:59:14	264	4	2020-01-01 00:10:39	210	13	0.0	12	180671	0.0
3142687	21718614	2	2019-12-31 23:59:30	214	18	2020-01-01 00:10:08	222	3	33169.0	12	215409	0.0

3142688 rows × 14 columns

```
# 대여, 반납 횟수를 파악하기 위해 각각의 데이터 프레임으로 정렬
bike_leas=bike_oh[['LEAS_STATION','LEAS_DATE']]
bike_rtn=bike_oh[['RTN_STATION','RTN_DATE']]
```

```
# 월별 이용횟수를 뽑기 위해 DATE의 월까지만 리턴
bike_leas['LEAS_DATE']=bike_leas['LEAS_DATE'].apply(lambda x: x[:7])
bike_rtn['RTN_DATE']=bike_rtn['RTN_DATE'].apply(lambda x: x[:7])
# 이용횟수를 구하기 위해 각 row에 1을 부여
bike_leas['leas_count']=1
bike_rtn['rtn_count']=1
# groupby sum
bike_leas=bike_leas.groupby(['LEAS_STATION','LEAS_DATE']).agg('sum').reset_index()
bike_rtn=bike_rtn.groupby(['RTN_STATION','RTN_DATE']).agg('sum').reset_index()
# 필요한 데이터만 뽑아보기
bike_leas=bike_leas[['LEAS_STATION','leas_count']]
bike_rtn=bike_rtn[['RTN_STATION','rtn_count']]
```

lambda x[:7]로 날짜 데이터의 연-월만 뽑아서 활용

월별 이용 횟수를 뽑아 적정 거치대 수를 산출하기 위함

동 별 적정 스테이션 수 산출

결과

```
[29] bike_leas
```

	LEAS_STATION	leas_count
0	101	80
1	101	74
2	101	201
3	101	306
4	101	437
...	...	...
5382	992	36
5383	992	31
5384	992	19
5385	998	23
5386	998	2

5387 rows × 2 columns

```
[30] bike_rtn
```

	RTN_STATION	rtn_count
0	0	3823
1	0	3421
2	0	6382
3	0	9783
4	0	12295
...	...	...
5431	999	1
5432	999	1
5433	999	2
5434	999	2
5435	999	1

5436 rows × 2 columns

원활한 계산을 위해

날짜 정보는 제거

동 별 적정 스테이션 수 산출

GROUP_BY를 통해 월 총합, 평균 데이터 산출

```
[31] # 합과 평균을 구해줌.
     bike_leas=bike_leas.groupby(['LEAS_STATION']).agg(['sum','mean']).reset_index()
     bike_rtn=bike_rtn.groupby(['RTN_STATION']).agg(['sum','mean']).reset_index()
     bike_leas.columns=['Station_ID','LEAS_SUM','LEAS_MONTH_MEAN']
     bike_rtn.columns=['Station_ID','RTN_SUM','RTN_MONTH_MEAN']
```

```
[32] bike_leas
```

	Station_ID	LEAS_SUM	LEAS_MONTH_MEAN
0	101	8366	232.388889
1	103	4535	125.972222
2	104	13402	372.277778
3	105	2806	77.944444
4	106	4444	123.444444
...	...	...	...
151	349	7532	209.222222
152	350	4190	116.388889
153	351	473	118.250000
154	992	952	26.444444
155	998	25	12.500000

156 rows × 3 columns

월별 이용 횟수를 뽑아

적정 거치대 수를 산출하기 위함

```
[33] bike_rtn
```

	Station_ID	RTN_SUM	RTN_MONTH_MEAN
0	0	398234	10763.081081
1	101	9207	255.750000
2	103	5121	142.250000
3	104	9330	259.166667
4	105	2696	74.888889
...	...	...	...
153	350	3662	101.722222
154	351	406	101.500000
155	992	795	22.083333
156	998	21	10.500000
157	999	10	1.250000

158 rows × 3 columns

동 별 적정 스테이션 수 산출

자전거 스테이션 데이터프레임에 병합

```
[35] # bike_station 데이터프레임에 대여, 반납 데이터 합쳐줌. 대여, 반납 이용횟수를 합친 TOTAL_MONTH_MEAN 칼럼 정의
     # 즉, TOTAL_MONTH_MEAN 은 총 이용 횟수의 월 평균값
     bike_station=bike_station.merge(bike_leas,on='Station_ID',how='outer').fillna(0)
     bike_station=bike_station.merge(bike_rtn,on='Station_ID',how='outer').fillna(0)
     bike_station['TOTAL_MONTH_MEAN']=bike_station['LEAS_MONTH_MEAN']+bike_station['RTN_MONTH_MEAN']
```

```
[36] bike_station
```

	Station_ID	STATION_NAME	거치대 수량	위도	경도	EMD_KOR_NM	LEAS_SUM	LEAS_MONTH_MEAN	RTN_SUM	RTN_MONTH_MEAN	TOTAL_MONTH_MEAN
0	101	어울림마을 701동 앞	20.0	37.654775	126.834584	성사동	8366.0	232.388889	9207.0	255.750000	488.138889
1	103	대림e-편한세상106동	20.0	37.660442	126.840377	성사동	4535.0	125.972222	5121.0	142.250000	268.222222
2	104	탄현마을8단지	25.0	37.698523	126.766042	탄현동	13402.0	372.277778	9330.0	259.166667	631.444444
3	105	KT 덕양지사 앞	20.0	37.655244	126.839261	성사동	2806.0	77.944444	2696.0	74.888889	152.833333
4	106	원당역 앞 공영주차장	30.0	37.653410	126.842530	성사동	4444.0	123.444444	2357.0	65.472222	188.916667
...	...	...	...	...	...	...	...	...	...	...	...
162	353	◆꿈에그린106동앞	40.0	37.666720	126.750784	대화동	0.0	0.000000	0.0	0.000000	0.000000
163	992	★피프틴센터	4.0	37.637529	126.833760	화정동	952.0	26.444444	795.0	22.083333	48.527778
164	998	0	0.0	0.000000	0.000000	0	25.0	12.500000	21.0	10.500000	23.000000
165	0	0	0.0	0.000000	0.000000	0	0.0	0.000000	398234.0	10763.081081	10763.081081
166	999	0	0.0	0.000000	0.000000	0	0.0	0.000000	10.0	1.250000	1.250000

167 rows × 11 columns

동 별 적정 스테이션 수 산출

행정동 기준으로 병합

행정동 기준으로 자전거 스테이션 데이터 병합

- 각 동에 필요한 스테이션 수요 개수를 구하기 위한 실질적인 데이터프레임 만들기

```
# border에 필요한 bike_station 데이터 병합
# bike_station + border
# station_count : 모든 자전거 스테이션 수
# station_count_OH : 운영 이력이 있는 자전거 스테이션 수
# bike_count : 모든 자전거 스테이션의 거치대 수 합
# bike_count_OH : 운영 이력이 있는 자전거 스테이션의 거치대 수 합
border['station_count']=0.0
border['station_count_OH']=0.0
border['bike_count']=0.0
border['bike_count_OH']=0.0
border['TOTAL_MONTH_MEAN']=0.0
for i in tqdm(bike_station.index):
  location=Point(bike_station['경도'][i],bike_station['위도'][i])
  for j in border.index:
    if location.within(border.geometry[j]):
      border['station_count'][j]+=1
      border['bike_count'][j]+=bike_station['거치대 수량'][i]
      border['TOTAL_MONTH_MEAN'][j]+=bike_station['TOTAL_MONTH_MEAN'][i]
      if bike_station['TOTAL_MONTH_MEAN'][i]!=0.0:
        border['station_count_OH'][j]+=1
        border['bike_count_OH'][j]+=bike_station['거치대 수량'][i]
# 거치대 수를 뽑기 위해 MONTH_MEAN_STATION_RATIO, MONTH_MEAN_BIKE_RATIO 설정
# MONTH_MEAN_STATION_RATIO : 월 평균 자전거 스테이션 이용 횟수
# MONTH_MEAN_BIKE_RATIO : 월 평균 자전거(거치대) 이용 횟수
border['MONTH_MEAN_STATION_RATIO']=border['TOTAL_MONTH_MEAN']/border['station_count_OH']
border['MONTH_MEAN_BIKE_RATIO']=border['TOTAL_MONTH_MEAN']/border['bike_count']
```

동별로 적정 스테이션 개수를 뽑고

적정 거치대 개수의 기준을 선정하기 위함

동 별 적정 스테이션 수 산출

병합된 데이터 확인

```
# 데이터 확인
border
```

	EMD_CD	EMD_KOR_NM	geometry	station_count	station_count_OH	bike_count	bike_count_OH	TOTAL_MONTH_MEAN	MONTH_MEAN_STATION_RATIO	MONTH_MEAN_BIKE_RATIO
0	41281101	주교동	MULTIPOLYGON (((126.81068 37.65820, 126.81069 ...	1.0	1.0	17.0	17.0	183.916667	183.916667	10.818627
1	41281102	원당동	MULTIPOLYGON (((126.83321 37.68013, 126.83340 ...	0.0	0.0	0.0	0.0	0.000000	NaN	NaN
2	41281103	신원동	MULTIPOLYGON (((126.86362 37.67729, 126.86364 ...	4.0	4.0	95.0	95.0	638.000000	159.500000	6.715789
3	41281104	원흥동	MULTIPOLYGON (((126.85975 37.65081, 126.85990 ...	6.0	6.0	165.0	165.0	2381.250000	396.875000	14.431818
4	41281105	도내동	MULTIPOLYGON (((126.84806 37.62348, 126.84835 ...	1.0	1.0	40.0	40.0	234.777778	234.777778	5.869444
5	41281106	성사동	MULTIPOLYGON (((126.83047 37.64860, 126.83053 ...	4.0	4.0	90.0	90.0	1098.111111	274.527778	12.201235
6	41281107	북한동	MULTIPOLYGON (((126.95320 37.65582, 126.95334 ...	0.0	0.0	0.0	0.0	0.000000	NaN	NaN
7	41281108	효자동	MULTIPOLYGON (((126.94756 37.65922, 126.94777 ...	0.0	0.0	0.0	0.0	0.000000	NaN	NaN
8	41281109	지축동	MULTIPOLYGON (((126.89619 37.65389, 126.89627 ...	0.0	0.0	0.0	0.0	0.000000	NaN	NaN
9	41281110	오금동	MULTIPOLYGON (((126.89197 37.67066, 126.89222 ...	0.0	0.0	0.0	0.0	0.000000	NaN	NaN
10	41281111	삼송동	MULTIPOLYGON (((126.87369 37.64345, 126.87385 ...	3.0	3.0	75.0	75.0	1197.361111	399.120370	15.964815
11	41281112	동산동	MULTIPOLYGON (((126.88013 37.64166, 126.88019 ...	2.0	2.0	80.0	80.0	743.413793	371.706897	9.292672
12	41281113	용두동	MULTIPOLYGON (((126.86983 37.61649, 126.86992 ...	0.0	0.0	0.0	0.0	0.000000	NaN	NaN

동 별 적정 스테이션 수 산출

계획 모델 데이터 불러오기

```
[39] # 계획 모델의 수요 데이터 불러오기
     plan_expected_demand_real=pd.read_csv('/content/drive/My Drive/compas/csv 파일 여기로!/도시계획_예상수요_REALplan.csv')
     plan_expected_demand_all=pd.read_csv('/content/drive/My Drive/compas/csv 파일 여기로!/도시계획_예상수요_ALLplan.csv')
```

```
[40] # 데이터프레임 좌표들을 str에서 Point로 치환 (추후 within 함수 활용 목적)
     model_real_demand['좌표']=model_real_demand['좌표'].apply(lambda x : Point(float(x[(x.find(',')+1):-1]),float(x[(x.find('(')+1):x.find(',')])))
     model_expected_demand['좌표']=model_expected_demand['좌표'].apply(lambda x : Point(float(x[(x.find(',')+1):-1]),float(x[(x.find('(')+1):x.find(',')])))
     plan_expected_demand_real['좌표']=plan_expected_demand_real['좌표'].apply(lambda x : Point(float(x[(x.find(',')+1):-1]),float(x[(x.find('(')+1):x.find(',')])))
     plan_expected_demand_all['좌표']=plan_expected_demand_all['좌표'].apply(lambda x : Point(float(x[(x.find(',')+1):-1]),float(x[(x.find('(')+1):x.find(',')])))
```

```
# 데이터 확인
model_real_demand
```

	좌표	대여_월평균	반납_월평균	총 이용_월평균	스테이션	위도	경도	Hosu_yes	CD_10299_기타일반업무시설	CD_11299_기타관광숙박시설	지하철_승하차수	CD_90001	Road_yes	CD_05501_박물관	CD_05999_기타문화및집회시설	CD_03004_일반목욕장	CD_02003_다세대주택
0	POINT (126.8804163490456 37.59052639593674)	0.00	0.000000	0.000000	[155]	37.590526	126.880416	0	0	0	0.0	0	1	0	0	0	0
1	POINT (126.8815482902 37.59054672642318)	0.00	0.000000	0.000000	[155]	37.590547	126.881548	0	0	0	0.0	0	1	0	0	0	0
2	POINT (126.8803907925734 37.59142713034578)	0.00	0.000000	0.000000	[155]	37.591427	126.880391	0	0	0	0.0	0	1	0	0	0	0
3	POINT (126.8815227473511 37.59144746148929)	0.00	0.000000	0.000000	[155]	37.591447	126.881523	0	0	0	0.0	0	1	0	0	0	0
4	POINT (126.882654703266 37.59146778178655)	0.00	0.000000	0.000000	[155]	37.591468	126.882655	0	0	0	0.0	0	1	0	0	0	0

동 별 적정 스테이션 수 산출

도로명 건물 모델과 계획 모델의 수요값을 동 별 데이터에 추합

[45]
```
# model_real_demand
# 월 평균 수요 데이터 병합
border['model_real_demand']=0.0
for i in tqdm(model_real_demand.index):
    for j in border.index:
        if model_real_demand['좌표'][i].within(border.geometry[j]):
            border['model_real_demand'][j]+=model_real_demand['총 이용_월평균'][i]
```

```
100% 3233/3233 [28:28<00:00, 1.89it/s]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable
  import sys
```

[46]
```
#model_expected_demand
border['model_demand']=0.0
for i in tqdm(model_expected_demand.index):
    for j in border.index:
        if model_expected_demand['좌표'][i].within(border.geometry[j]):
            border['model_demand'][j]+=model_expected_demand['예상 수요'][i]
```

```
100% 17604/17604 [01:11<00:00, 244.94it/s]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable
```

[47]
```
border['model_total_demand']=border['model_real_demand']+border['model_demand']
```

[48]
```
#plan_real_demand
border['plan_real_demand']=0.0
for i in tqdm(plan_expected_demand_real.index):
    for j in border.index:
        if plan_expected_demand_real['좌표'][i].within(border.geometry[j]):
            border['plan_real_demand'][j]+=plan_expected_demand_real['예상 수요'][i]
```

```
100% 17604/17604 [27:01<00:00, 10.86it/s]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
```

[49]
```
border['plan_real_total']=border['model_real_demand']+border['plan_real_demand']
```

[50]
```
# 비교를 위해 계획에서 현 자전거 스테이션 주변 좌표와 일치하는 좌표를 뽑음
count = 0
bike_index = []
for idx, loc in tqdm(enumerate(np.array(plan_expected_demand_all['좌표']))):
  for y_loc in np.array(model_real_demand['좌표']):
    if loc == y_loc:
      count += 1
      bike_index.append(idx)
```

```
20837/? [24:12<00:00, 14.35it/s]
```

```
# plan_all_bike_demand
# 비교를 위해 현 자전거 스테이션 주변 좌표의 계획 데이터 반영 수요를 뽑음
border['plan_all_bike_demand']=0.0
for i in tqdm(bike_index):
    for j in border.index:
      if plan_expected_demand_all['좌표'][i].within(border.geometry[j]):
        border['plan_all_bike_demand'][j]+=plan_expected_demand_all['예상 수요'][i]
```

```
#plan_all_demand
border['plan_all_demand']=0.0
for i in tqdm(plan_expected_demand_all.index):
    for j in border.index:
        if plan_expected_demand_all['좌표'][i].within(border.geometry[j]):
            border['plan_all_demand'][j]+=plan_expected_demand_all['예상 수요'][i]
```

```
100% 20837/20837 [01:24<00:00, 247.67it/s]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
```

동 별 적정 스테이션 수 산출

적정 스테이션 개수 선정의 기준

```
# 실제 자전거 수요 값을 자전거 스테이션 이용횟수로 나누어 스테이션 당 수요 분포 추출
(border['model_real_demand']/border['station_count_OH']).describe()
```

```
count    2.800000e+01
mean              inf
std               NaN
min      2.909669e+03
25%      7.434965e+03
50%      1.728218e+04
75%      4.173719e+04
max               inf
dtype: float64
```

실제 자전거 수요값(운영 이력을 활용)을 실제 운영 이력이 있는 자전거 스테이션의 이용횟수로 나누어 스테이션 당 수요 분포 추출 이를 활용해 적정 스테이션 개수를 산출해낸 것.

+) 수요 분포가 오른쪽으로 치우쳐짐을 확인.

동 별 적정 스테이션 수 산출

필요한 데이터만 활용하기 위해 border_brief 데이터프레임 정의

```
# 필요한 데이터만 보기 위해 border_brief 정의
# station_count : 실제 스테이션 수
# bike_count : 실제 거치대 수 합
# station_demand_least : 도로명 모델 수요를 스테이션 당 수요의 삼사분위 값으로 나눈 것
# station_demand_median : 도로명 모델 수요를 스테이션 당 수요의 중앙값으로 나눈 것
# station_demand_most : 도로명 모델 수요를 스테이션 당 수요의 일사분위 값으로 나눈 것
# plan_demand_least : 계획 모델 수요를 스테이션 당 수요의 삼사분위 값으로 나눈 것
# plan_demand_median : 계획 모델 수요를 스테이션 당 수요의 중앙값으로 나눈 것
# plan_demand_most : 계획 모델 수요를 스테이션 당 수요의 일사분위 값으로 나눈 것
# month_mean_bike_ratio : 거치대 수를 구하기 위해 필요한 월 평균 자전거 이용량
border_brief=border[['EMD_KOR_NM','station_count','bike_count']]
border_brief['station_count']=border_brief['station_count'].astype(int)
border_brief['bike_count']=border_brief['bike_count'].astype(int)
border_brief['station_demand_least']=round(border['model_total_demand']/(border['model_real_demand']/border['station_count_OH']).quantile(0.75),0).astype(int)
border_brief['station_demand_median']=round(border['model_total_demand']/(border['model_real_demand']/border['station_count_OH']).quantile(0.5),0).astype(int)
border_brief['station_demand_most']=round(border['model_total_demand']/(border['model_real_demand']/border['station_count_OH']).quantile(0.25),0).astype(int)
border_brief['plan_demand_least']=round(border['plan_all_demand']/(border['model_real_demand']/border['station_count_OH']).quantile(0.75),0).astype(int)
border_brief['plan_demand_median']=round(border['plan_all_demand']/(border['model_real_demand']/border['station_count_OH']).quantile(0.5),0).astype(int)
border_brief['plan_demand_most']=round(border['plan_all_demand']/(border['model_real_demand']/border['station_count_OH']).quantile(0.25),0).astype(int)
border_brief['month_mean_bike_ratio']=border['MONTH_MEAN_BIKE_RATIO'].fillna(0)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

스테이션 당 수요 분포를 활용해 스테이션 선정 기준값들을 도출

(일사분위, 중앙값, 삼사분위 활용)

동 별 적정 스테이션 수 산출

결과 확인

```
border_brief
```

	EMD_KOR_NM	station_count	bike_count	station_demand_least	station_demand_median	station_demand_most	plan_demand_least	plan_demand_median	plan_demand_most	month_mean_bike_ratio
0	주교동	1	17	3	6	15	2	4	10	10.818627
1	원당동	0	0	3	7	15	2	4	10	0.000000
2	신원동	4	95	2	5	11	2	6	13	6.715789
3	원흥동	6	165	3	7	16	3	8	18	14.431818
4	도내동	1	40	3	7	16	4	11	25	5.869444
5	성사동	4	90	3	8	20	4	9	20	12.201235
6	북한동	0	0	0	0	0	0	0	1	0.000000
7	효자동	0	0	1	3	6	1	2	4	0.000000
8	지축동	0	0	3	7	17	2	5	12	0.000000
9	오금동	0	0	1	4	8	1	3	7	0.000000
10	삼송동	3	75	2	4	10	3	6	15	15.964815
11	동산동	2	80	2	4	10	2	5	11	9.292672
12	용두동	0	0	3	8	18	3	7	16	0.000000
13	벽제동	0	0	3	7	16	1	2	6	0.000000
14	선유동	0	0	2	4	9	0	1	2	0.000000
15	고양동	0	0	2	5	12	1	2	5	0.000000
16	대자동	0	0	5	12	27	3	7	16	0.000000
17	관산동	0	0	4	9	20	2	5	12	0.000000
18	내유동	0	0	3	7	17	1	3	8	0.000000
19	토당동	4	90	3	6	14	8	20	46	16.714815

동 별 적정 스테이션 수 산출

DEMAND_LEAST, PLAN_LEAST 활용 데이터프레임 간략화

```
[63] # station_demand_least와 plan_demand_least를 사용하기로 했기 때문에 각 칼럼들을 뽑고 그에 더해 편의를 위해 칼럼명들을 단순화 시켜줌.
     border_brief=border_brief[['EMD_KOR_NM','station_count','station_demand_least','plan_demand_least','month_mean_bike_ratio']]
     border_brief.columns=['EMD_KOR_NM','station_count','station_demand','plan_demand','bike_ratio']
```

```
border_brief
```

	EMD_KOR_NM	station_count	station_demand	plan_demand	bike_ratio
0	주교동	1	3	2	10.818627
1	원당동	0	3	2	0.000000
2	신원동	4	2	2	6.715789
3	원흥동	6	3	3	14.431818
4	도내동	1	3	4	5.869444
5	성사동	4	3	4	12.201235
6	북한동	0	0	0	0.000000
7	효자동	0	1	1	0.000000
8	지축동	0	3	2	0.000000
9	오금동	0	1	1	0.000000
10	삼송동	3	2	3	15.964815
11	동산동	2	2	2	9.292672
12	용두동	0	3	3	0.000000
13	벽제동	0	3	1	0.000000
14	선유동	0	2	0	0.000000
15	고양동	0	2	1	0.000000
16	대자동	0	5	3	0.000000

수요 그래프가 오른쪽으로 치우쳐져 있기 때문에 중앙값보다 삼사분위 값으로 나눈 station_demand_least와 plan_demand_least를 행정동 당 스테이션 개수 설정에 사용하기로 함.

또한, station_demand_least와 plan_demand_least의 스테이션 개수가 현행 스테이션 개수와 가장 유사해 이를 활용하는 것이 스테이션의 실질적인 운영에 더 적합하고 도움이 된다고 판단했음.

동 별 적정 스테이션 수 산출

추가 고려 사항 1. 19년도 하반기 이후 설치된 스테이션들

```
# 19년도 하반기 이후 설치된 스테이션들
bike_station['STATION_NAME']=bike_station['STATION_NAME'].astype(str)
new_station=bike_station[(bike_station['STATION_NAME'].apply(lambda x : x[0])=='◆')]
new_station
```

운영 이력이 적거나 전무함

	Station_ID	STATION_NAME	거치대 수량	위도	경도	EMD_KOR_NM	LEAS_SUM	LEAS_MONTH_MEAN	RTN_SUM	RTN_MONTH_MEAN	TOTAL_MONTH_MEAN
35	151	◆향동지구 10단지 151	40.0	37.607051	126.894917	향동동	0.0	0.000000	0.0	0.000000	0.000000
36	152	◆향동지구 2단지 152	40.0	37.601162	126.895028	향동동	0.0	0.000000	0.0	0.000000	0.000000
37	153	◆향동지구 4단지 153	40.0	37.595718	126.892157	향동동	0.0	0.000000	0.0	0.000000	0.000000
38	154	◆향동지구 1단지 154	40.0	37.594102	126.890306	향동동	0.0	0.000000	0.0	0.000000	0.000000
39	155	◆향동 덕은교 155	40.0	37.591856	126.882963	덕은동	0.0	0.000000	0.0	0.000000	0.000000
94	241	◆동국대병원 사거리	37.0	37.674051	126.806167	식사동	0.0	0.000000	0.0	0.000000	0.000000
95	242	◆식사 3단지 분수대 옆	43.0	37.681718	126.812546	식사동	0.0	0.000000	0.0	0.000000	0.000000
114	263	◆원시티 정문	40.0	37.661579	126.747736	대화동	253.0	84.333333	425.0	141.666667	226.000000
115	264	◆킨텍스 사거리	40.0	37.664403	126.748019	대화동	688.0	229.333333	759.0	253.000000	482.333333
116	265	◆원시티 육교	40.0	37.661296	126.750847	장항동	767.0	255.666667	973.0	324.333333	580.000000
160	351	◆일산에듀포레 푸르지오	40.0	37.702259	126.767231	탄현동	473.0	118.250000	406.0	101.500000	219.750000
161	352	◆꿈에그린203동앞	40.0	37.666425	126.749244	대화동	0.0	0.000000	0.0	0.000000	0.000000
162	353	◆꿈에그린106동앞	40.0	37.666720	126.750784	대화동	0.0	0.000000	0.0	0.000000	0.000000

동 별 적정 스테이션 수 산출

추가 고려 사항 2. 도시화 지역 경계 데이터(2020년 기준 인구)

```
[73] city_border['pop_growth']=round(city_border['INGU_CNT']/city_border['population'],2)
```

```
[74] city_border
```

	SIGUNGU_CD	UA_CD	UA_NM	INGU_CNT	UA_AREA	BAS_CNT	geometry	population	pop_growth
0	31101	UA311011	도시화지역 1	200330.0	5493213.78	1105	MULTIPOLYGON (((126.83119 37.64876, 126.83131 ...	177104	1.13
1	31101	UA311012	도시화지역 10	3553.0	308050.11	17	MULTIPOLYGON (((126.87061 37.60478, 126.87064 ...	1047	3.39
2	31101	UA311013	도시화지역 2	64906.0	2906659.22	381	MULTIPOLYGON (((126.87958 37.64411, 126.87942 ...	52216	1.24
3	31101	UA311014	도시화지역 3	47864.0	1347246.48	338	MULTIPOLYGON (((126.83641 37.66600, 126.83655 ...	33156	1.44
4	31101	UA311015	도시화지역 4	21652.0	734702.03	107	MULTIPOLYGON (((126.89562 37.66369, 126.89570 ...	14335	1.51
5	31101	UA311016	도시화지역 5	20735.0	512190.96	123	MULTIPOLYGON (((126.90673 37.71124, 126.90669 ...	8758	2.37
6	31101	UA311017	도시화지역 6	12266.0	362798.92	82	MULTIPOLYGON (((126.86483 37.69379, 126.86505 ...	6621	1.85
7	31101	UA311018	도시화지역 7	10318.0	628378.85	58	MULTIPOLYGON (((126.85891 37.71724, 126.85898 ...	6998	1.47
8	31101	UA311019	도시화지역 8	7770.0	414097.95	29	MULTIPOLYGON (((126.84777 37.73355, 126.84729 ...	5428	1.43
9	31101	UA3110110	도시화지역 9	4288.0	130148.22	39	MULTIPOLYGON (((126.90758 37.72327, 126.90771 ...	1821	2.35
10	31103	UA311031	도시화지역 1	250302.0	9690207.23	1526	MULTIPOLYGON (((126.81401 37.67574, 126.81438 ...	215634	1.16
11	31103	UA311032	도시화지역 2	4401.0	312777.94	24	MULTIPOLYGON (((126.84998 37.69648, 126.84991 ...	2930	1.50
12	31103	UA311033	도시화지역 3	3571.0	662231.53	15	MULTIPOLYGON (((126.79362 37.71660, 126.79366 ...	1890	1.89
13	31104	UA311041	도시화지역 1	244326.0	9071094.48	1382	MULTIPOLYGON (((126.76751 37.70556, 126.76766 ...	238998	1.02
14	31104	UA311042	도시화지역 2	15772.0	711032.23	80	MULTIPOLYGON (((126.72422 37.69686, 126.72419 ...	12206	1.29
15	31104	UA311043	도시화지역 3	7854.0	799972.33	37	MULTIPOLYGON (((126.73456 37.70497, 126.73467 ...	5635	1.39

population은 비교를 위해 100m 그리드 인구 분포에서 데이터를 뽑아 도시화 지역 경계 데이터에 취합

pop_growth(인구 성장률)을 통해 전 지역에 전반적으로 인구가 많아졌음을 볼 수 있음.

동 별 적정 스테이션 수 산출

기준에 따른 함수 작성

- 동별 스테이션 개수 산출 함수 작성

```
border_brief['station_final']=0
for i in border_brief.index:
  if border_brief['EMD_KOR_NM'][i] in ['지축동', '원흥동', '도내동', '향동동', '신원동', '동산동','식사동']:
    border_brief['station_final'][i]=border_brief['station_count'][i]+2
  else:
    if (border_brief['station_demand'][i]>=border_brief['plan_demand'][i])|(border_brief['EMD_KOR_NM'][i]=='장항동'):
      border_brief['station_final'][i]=border_brief['station_demand'][i]
    else:
      border_brief['station_final'][i]=border_brief['plan_demand'][i]
```

신도시 지역에 현행 대비 자전거 스테이션 2개소 추가 설치
(미래 입주 인구, 도시 계획, 컴플레인 등 고려)

도로명 건물 모델 적정 스테이션 수와 계획 모델 적정 스테이션 수 중 더 높은 값을 차용. 이는 도시 계획 모델을 반영하면서도 도시 계획 모델로 추산한 적정 스테이션 수가 비현실적이 되는 것을 방지하기 위함.

동 별 적정 스테이션 수 산출

동 별 적정 스테이션 개수 산출

산출된 동별 적정 스테이션 개수

```
border_brief
```

	EMD_KOR_NM	station_count	station_demand	plan_demand	bike_ratio	station_final
0	주교동	1	3	2	10.818627	3
1	원당동	0	3	2	0.000000	3
2	신원동	4	2	2	6.715789	6
3	원흥동	6	3	3	14.431818	8
4	도내동	1	3	4	5.869444	3
5	성사동	4	3	4	12.201235	4
6	북한동	0	0	0	0.000000	0
7	효자동	0	1	1	0.000000	1
8	지축동	0	3	2	0.000000	2
9	오금동	0	1	1	0.000000	1
10	삼송동	3	2	3	15.964815	3
11	동산동	2	2	2	9.292672	4
12	용두동	0	3	3	0.000000	3
13	벽제동	0	3	1	0.000000	3
14	선유동	0	2	0	0.000000	2
15	고양동	0	2	1	0.000000	2
16	대자동	0	5	3	0.000000	5
17	관산동	0	4	2	0.000000	4
18	내유동	0	3	1	0.000000	3
19	토당동	4	3	8	16.714815	8

station_final 칼럼이 최종 동별 적정 스테이션 개수

거치대 개수 설정을 위해

월별 자전거 한 대당 이용(bike_ratio)의 분포 확인

```
[78]  # bike_ratio의 describe
      # 거치대 개수를 설정하기 위해 describe로 분포 확인
      border_brief[border_brief['bike_ratio']!=0]['bike_ratio'].describe()
```

```
count     22.000000
mean      32.268378
std       27.473682
min        5.869444
25%       12.321759
50%       21.534180
75%       36.565391
max      105.633323
Name: bike_ratio, dtype: float64
```

6. 최적 스테이션 선정 코드 설명

(4) 최적 스테이션 선정

최적 스테이션 선정

도로명 도로 데이터 불러오기

100m 그리드로 스테이션 좌표를 찍을 경우,
도로변에서 벗어날 위험 존재
가까운 도로 좌표를 활용해 위치를 조정하기 위해
도로 데이터를 불러옴

```
[79] # 모델 좌표값으로 좌표를 설정해주면 도로에서 많이 벗어난 곳에 좌표를 찍을 수 있기 때문에 도로 데이터를 가져옴.
     road=gpd.read_file('/content/drive/My Drive/compas/SBJ_2007_001/16.도로명주소_도로.geojson')
```

```
road
```

	BSI_INT	RBP_CN	RDS_DPN_SE	RDS_MAN_NO	REP_CN	RN	ROAD_BT	ROAD_LT	ROA_CLS_SE	WDR_RD_CD	geometry
0	2000	선유동 520-12	0	21930	신평동 1-1	서울외곽순환고속도로	50.0	128020.000	1	1	MULTILINESTRING ((126.91306 37.67694, 126.9128...
1	2000	현천동 792-5	0	21932	강매동 604-29	인천국제공항고속도로	50.0	38550.000	1	1	MULTILINESTRING ((126.82662 37.58822, 126.8266...
2	2000	경기도 화성시 봉담읍 봉담IC	0	23767	경기도 파주시 문산읍 내포IC	수원문산고속도로	30.0	78600.000	1	1	MULTILINESTRING ((126.82665 37.58824, 126.8271...
3	20	법곳동 307-1전	1	935	동산동 27-31전	고양대로	2.0	198.820	2	3	MULTILINESTRING ((126.85633 37.64793, 126.8563...
4	20	법곳동 307-1전	2	1906	동산동 27-31전	고양대로	3.0	91.000	2	3	MULTILINESTRING ((126.84622 37.65368, 126.8462...
...	...	...	...	...	...	...	...	...	...	...	...
4539	21	덕이동361-38	0	3984	덕이동336-32	하이파크2로59번길	14.0	451.000	4	3	MULTILINESTRING ((126.75590 37.69932, 126.7558...
4540	20	덕이동 산143-11	0	4524	덕이동 303-21	하이파크3로78번길	14.0	406.707	4	3	MULTILINESTRING ((126.75289 37.69852, 126.7559...
4541	20	가좌동 619-17	1	5527	가좌동 619	송파로151번길	3.0	345.637	4	3	MULTILINESTRING ((126.71417 37.69893, 126.7148...
4542	10	가좌동 619-17	0	5506	가좌동 619	송파로151번길	3.0	345.637	4	3	MULTILINESTRING ((126.71424 37.69702, 126.7140...
4543	10	가좌동 576-8	0	5507	가좌동 609-1	송파로202번길	3.0	265.924	4	3	MULTILINESTRING ((126.71757 37.70062, 126.7175...

4544 rows × 11 columns

최적 스테이션 선정

적정 스테이션 위치 반환 함수

최종 좌표값 뽑기

```
[231] # station demand 좌표 찍기
      # dong_list는 동별 좌표값을 받기 위함, sta_list는 전체 좌표값 리턴
      # full_list는 중복 방지를 위한 리스트
      dong_list=[]
      full_list=[]
      sta_list=[]
      # border_brief 좌표들을 하나씩 찍음
      # final_demand만 사용.
      for sta_emd,_,_,_,_,final_demand in tqdm(np.array(border_brief)):
      # 중간 결과값을 받기 위한 loc_list
        loc_list=[]
      # 신설 스테이션 고려
        if sta_emd in new_station['EMD_KOR_NM'].drop_duplicates().tolist():
          for _, _, _, new_lat, new_long,_,_,_,_,_,_ in np.array(new_station[new_station['EMD_KOR_NM']==sta_emd]):
            new_sta_loc=Point(new_long,new_lat)
            loc_list.append(new_sta_loc)
            full_list.append(new_sta_loc)
      # 신설 스테이션의 경우 거치대 수가 30이 넘기 때문에 상한선인 30으로 설정
            sta_list.append((new_sta_loc,30))
      # border_brief에서 받은 동을 기준으로 수요가 높은 순으로 model_demand_all을 정렬 후 좌표들을 하나씩 뽑음
        for loc,loc_demand,station_ex,loc_emd in np.array(model_demand_all[model_demand_all['EMD_KOR_NM']==sta_emd].sort_values('demand',ascending=False)):
      # model_demand_all의 좌표값을 받음
          loc_point=(loc.x,loc.y)
      # hav 거리를 넣을 h 와 새로운 location 값을 넣을 new_loc 리스트 정의
          h=[]
          new_loc=[]
      # 100m 이내의 기존 스테이션이 존재한다면(100m 그리드를 사용했기 때문에 오차 허용 범위라 가정) 스테이션 좌표값을 리턴하기 위한 반복문
          if len(loc_list)<final_demand:
            bike_h=[]
            bike_point=[]
            for j in bike_station.index:
      # 기존 스테이션의 값들을 z로 하나씩 받아줌
              z=(bike_station['경도'][j],bike_station['위도'][j])
      # 좌표값과 스테이션 좌표값의 직선거리 산출
              bike_hav=haversine(loc_point,z, unit='m')
      # 만약 직선거리가 100m 이내라면 리스트의 0번째에 스테이션 좌표값을 리턴
              if bike_hav<=100:
                if bike_h==[]:
                  bike_h.append(bike_hav)
                  bike_point.append(Point(z))
      # 혹시 기존에 받은 스테이션 좌표값이 있다면 최소값만을 반환.
                else:
                  if bike_hav<min(bike_h):
                    bike_h.insert(0,bike_hav)
                    bike_point.insert(0,Point(z))
      # 가장 가까운 스테이션 값을 list에 삽입
            if bike_point != []:
              if (full_list!=[])&(bike_point[0] not in full_list):
      # 좌표의 예측 수요 값을 bike_ratio의 평균값으로 나눠 적정 거치대 개수를 산출, 최대값을 30으로 설정
                bike_demand=int(round((loc_demand/border_brief[border_brief['bike_ratio']!=0]['bike_ratio'].mean()),0))
                if bike_demand>30:
                  bike_demand=30
                loc_list.append(bike_point[0])
                full_list.append(bike_point[0])
                sta_list.append((bike_point[0],bike_demand))
      # 스테이션 값을 넣어줬을 경우 다음 iteration으로
                continue
      # 만약 해당 동의 수요가 있음에도 초기값이 존재하지 않는다면 초기값을 넣어줌.
          if (len(loc_list)==0)&(final_demand!=0):
      # 찍힌 좌표를 가까운 도로변으로 옮겨주기 위한 코드
            for i in road.index:
      # 선으로 표현된 도로를 점으로 치환
              points = [point for polygon in road.geometry[i] for point in polygon.coords[:]]
      # 점으로 치환된 도로들을 하나씩 비교
              for t_long, t_lat in points:
      # y는 점으로 치환된 도로 값 하나, loc_point는 100m 그리드의 좌표값, mid_point는 좌표가 도로에 찍히는 걸 방지하는 좌표값.
                y=(t_long, t_lat)
      # 직선거리를 구해줌
                mid_point=((t_long+0.75+loc.x+0.25),(t_lat+0.75+loc.y+0.25))
                new_h=haversine(loc_point,y, unit='m')
      # 여기서 직선거리의 초기값이 없다면 h라는 리스트에 넣어줌, 좌표의 초기값도 new_loc에 넣어줌.
                if h==[]:
                  h.append(new_h)
                  new_loc.append(Point(mid_point))
```

최적 스테이션 선정

적정 스테이션 위치 반환 함수

```
# 여기서 직선거리의 초기값이 없다면 h라는 리스트에 넣어줌, 좌표의 초기값도 new_loc에 넣어줌.
          if h==[]:
            h.append(new_h)
            new_loc.append(Point(mid_point))
# 초기값이 존재한다면 아래의 조건문을 실행
          else:
# 새로운 직선거리가 기존 직선거리의 최소값보다 작은 경우
            if new_h<min(h):
# 각 리스트의 0번째에 직선거리값과 좌표값을 넣어줌
              h.insert(0,new_h)
              new_loc.insert(0,Point(mid_point))
# 거치대수 리턴, 최대값을 30으로 설정
      bike_demand=int(round((loc_demand/border_brief[border_brief['bike_ratio']!=0]['bike_ratio'].mean()),0))
      if bike_demand>30:
        bike_demand=30
# road 반복문이 끝난 후 loc_list, sta_list에 각각 값을 입력
      loc_list.append(new_loc[0])
      full_list.append(new_loc[0])
      sta_list.append((new_loc[0],bike_demand))
# 동별 스테이션 수요 개수보다 반환된 좌표값의 개수가 적을 때
    elif len(loc_list)<final_demand:
# 기존에 존재하는 loc_list의 값과 새로운 값을 비교해주기 위함. 기존과 가까운 좌표에 새로운 좌표가 찍히는 것을 방지.
      haversine_list=[]
      for locs in loc_list:
        locs_point=(locs.x,locs.y)
        haversine_list.append(haversine(locs_point,loc_point,unit='m'))
# loc_list에 존재하는 모든 좌표와의 직선 거리의 최소값이 200이상
# 좌표값이 너무 근접하게 되는 것을 방지
      if min(haversine_list)>=200:
        h=[]
        new_loc=[]
        for i in road.index:
          points = [point for polygon in road.geometry[i] for point in polygon.coords[:]]
          for t_long, t_lat in points:
            y=(t_long, t_lat)
          points = [point for polygon in road.geometry[i] for point in polygon.coords[:]]
          for t_long, t_lat in points:
            y=(t_long, t_lat)
# mid_point는 도로에 0.75, 100m 그리드 좌표값에 0.25의 가중치를 준 보간 위치값
            mid_point=((t_long*0.75+loc.x*0.25),(t_lat*0.75+loc.y*0.25))
# 직선 거리를 구함.
            new_h=haversine(loc_point,y, unit='m')
            if h==[]:
              h.append(new_h)
              new_loc.append(Point(mid_point))
            else:
# 만약 도로와의 직선 거리값이 이전 좌표들과의 직선 거리 값보다 작다면 각 리스트의 0번째에 넣음
              if new_h<min(h):
                h.insert(0,new_h)
                new_loc.insert(0,Point(mid_point))
# 거치대수 리턴, 최대값을 30으로 설정
        if new_loc[0] not in full_list:
          bike_demand=int(round((loc_demand/border_brief[border_brief['bike_ratio']!=0]['bike_ratio'].mean()),0))
          if bike_demand>30:
            bike_demand=30
# road 반복문이 끝난 후 loc_list, sta_list에 각각 값을 입력
          loc_list.append(new_loc[0])
          full_list.append(new_loc[0])
          sta_list.append((new_loc[0],bike_demand))
# 반환된 좌표 개수가 수요 개수와 같거나 많다면 예외 처리를 위해 NaN값 부여 (동별 데이터프레임 만들 때를 위함)
    else:
      loc_list.append(np.nan)
# 반환된 좌표 값이 전체 자전거 스테이션 수요의 최대보다 많다면 리스트 반환 후 반복문 탈출
    if len(loc_list)>border_brief['station_demand'].max():
      loc_list.insert(0,sta_emd)
      dong_list.append(loc_list)
      break
# 결과 확인 용 리스트들은 바로 데이터프레임 변환
dong_pd=pd.DataFrame(dong_list)
# 제출 형식에 맞춰 결과 데이터프레임 산출
sta_final=[]
for loc, bike in sta_list:
  sta_final.append((bike,loc.y,loc.x))
sta_final_pd=pd.DataFrame(sta_final,columns=['거치대수량','X좌표(위도)','Y좌표(경도)'])
sta_final_pd.index.name='스테이션 번호'
```

```
100% 53/53 [13:53<00:00, 15.73s/it]
```

최적 스테이션 선정

결과 확인

```
[243] sta_final_pd
```

스테이션 번호	거치대수량	X좌표(위도)	Y좌표(경도)
0	20	37.651140	126.821031
1	20	37.652548	126.832309
2	18	37.654066	126.820734
3	18	37.679189	126.850636
4	17	37.676696	126.848491
...	...	...	...
261	16	37.667612	126.727548
262	15	37.663052	126.722723
263	15	37.657185	126.731232
264	14	37.661195	126.729438
265	13	37.657259	126.729210

266 rows × 3 columns

총 266 스테이션

최적 스테이션 선정

결과 도출 - 유지되는 스테이션

```
still_station=[]
for final_bike,final_lat, final_long in np.array(sta_final_pd):
  for i in bike_station.index:
    if (final_lat,final_long) == (bike_station['위도'][i],bike_station['경도'][i]):
      still_station.append((bike_station['Station_ID'][i],bike_station['STATION_NAME'][i],int(bike_station['거치대 수량'][i]),int(final_bike)))
```

```
still_pd=pd.DataFrame(still_station,columns=['Station_ID','STATION_NAME','기존 거치대 수량','추천 거치대 수량'])
```

```
still_pd
```

최적 스테이션 위치로 뽑은 좌표값이 기존 스테이션과 100m 이내 인접하면 기존 스테이션 위치를 반환하도록 함수 설계 (100m 그리드 좌표를 통해 예측 모델링)

좌표값이 일치하면 리턴하는 함수를 설계

최적 스테이션 선정

결과 도출 - 유지되는 스테이션

결과적으로 현행 스테이션들 중 아래 총 47개의 스테이션 유지

still_pd

	Station_ID	STATION_NAME	기존 거치대 수량	추천 거치대 수량
0	110	어울림누리 맞은편	20	24
1	125	꽃우물 공원 앞	30	14
2	256	고양백석체육센터	20	26
3	155	◆향동 덕은교 155	40	30
4	151	◆향동지구 10단지 151	40	30
5	152	◆향동지구 2단지 152	40	30
6	153	◆향동지구 4단지 153	40	30
7	154	◆향동지구 1단지 154	40	30
8	241	◆동국대병원 사거리	37	30
9	242	◆식사 3단지 분수대 옆	43	30
10	245	★하늘마을4단지 403동 앞	20	30
11	229	★풍산역	15	30
12	225	양지마을 405동 앞	20	30
13	265	◆원시티 육교	40	30
14	253	★호수공원 (강선17단지 삼거리)	30	30
15	254	★호수공원 제3주차장 (MBC맞은편)	30	30
16	210	★남정씨티프라자 옆	20	30
17	259	★웨스턴돔 입구	30	30
18	252	★일산문화공원 홈플러스 앞	30	30
19	215	마두역 7번 출구	25	30
20	214	★마두역 교보생명 옆	20	30
21	212	★정발산역 4번 출구 일산동구청 방면	30	30
22	211	★정발산역 2번 출구 (롯데백화점)	30	30
23	232	★뉴코아아울렛 앞	20	30
24	204	백석역 6번 출구 (신선설농탕)	30	30
25	203	★백석역 3번 출구 앞	20	30
26	235	백마마을 308동 앞	30	30
27	222	백석고 옆	30	30
28	221	백석1동주민센터	30	30
29	233	백마역	25	30
30	342	★문화공원	30	30
31	316	후곡성당 앞	30	30
32	343	문화초교 옆	25	30
33	317	★일산3동 주민센터 앞	20	30
34	310	★주엽역 2번 출구	30	30
35	309	★주엽역 6번출구 앞	30	30
36	351	◆일산에듀포레 푸르지오	40	30
37	330	동문1차 101동 맞은편	25	30
38	263	◆원시티 정문	40	30
39	264	◆킨텍스 사거리	40	30
40	352	◆꿈에그린203동앞	40	30
41	353	◆꿈에그린106동앞	40	30
42	305	★대화역 6번 출구	20	30
43	306	대화역 1번 출구	30	30
44	307	장성공원	20	30
45	341	탄현역 서쪽 현충공원앞	30	30
46	331	탄현역 동쪽 출구	20	30