In [1]:

```
import pandas as pd
import statsmodels.api as sm
```

다중회귀모델

In [2]:

```
class MLR:

    def __init__(self, X, y):
        self.X = X
        self.y = y
        self.model = self._model()

    def _model(self):
        Xc = sm.add_constant(self.X)
        lr = sm.OLS(self.y,Xc)
        lr_result = lr.fit()

        return lr_result

    def predict(self, x_test):
#         Xp = sm.add_constant(x_test, has_constant='add')
        Xp = sm.add_constant(x_test)
        y_predict = self.model.predict(Xp)

        return y_predict

    def summary(self):
        return self.model.summary()

    def coef(self):
        return self.model.params
```

MCLP

[Hyperparameter]

설명: 후보지 r(500m) 내에 수요지(x가 얼만큼 있는지를 파악하여 상위

- K = 20
- r = 500m (도보로 6분 거리 기준)
- x_j: 후보지 중 K에 선정된 지역은 1, 미선정 지역은 0
- y_i: 적어도 하나의 설비로 그 포인트가 커버가 되면 1, 그렇지 않으면 0
    - 반경 안에 있는지를 의미

[후보지(J) 선정]

목적: 전기차 충전소에 대한 사람들의 실질적인 수요와 공급(충전소 보급 현황)을 모두 고려하여 후보지를 선정 하고자 함.

$$s = (1 - \frac{500m\,\text{안 충전기 수}}{\text{용인시 전체 충전기 수}})$$

- $CntCust * s$ 상위 1000개 값을 후보지로 선정

[수요지(I) 선정]

- 100*100 기준 cnt_cust 값이 있는 지역을 수요지로 선정

In [ ]:

```python
## 필요 패키지 로드
import numpy as np
from scipy.spatial import distance_matrix
from gurobipy import *
from scipy.spatial import ConvexHull
from shapely.geometry import Polygon, Point
from numpy import random
```

In [1]:

```python
from scipy.spatial import distance_matrix
def mclp(K, radius, demand, candidate, Weight):

    """
    [Maximum Covering Location Problem]
    입력값:
        K: 최적 입지 개수
        radius: 입지 반경
        demand(I): 수요지 (input points, Numpy array in shape of [N,2])
        candidate(J): 후보지
    출력값:
        opt_sites: locations K optimal sites, Numpy array in shape of [K,2]
        f: the optimal value of the objective function
    """
    print('----- Configurations -----')
    print('  수요지 수 %g' % demand.shape[0])
    print('  후보지 수 %g' % candidate.shape[0])
    print('  K %g' % K)
    print('  Radius %g' % radius)
    import time
    start = time.time()
    J = candidate.shape[0]
    I = demand.shape[0]
    D = distance_matrix(demand, candidate)
    mask1 = D<=radius
    D[mask1]=1
    D[~mask1]=0
    w = Weight
    from mip import Model, xsum, maximize, BINARY

    # Build model
    m = Model("mclp")
    # Add variables

    x = [m.add_var(name = "x%d" % j, var_type = BINARY) for j in range(J)]
    y = [m.add_var(name = "y%d" % i, var_type = BINARY) for i in range(I)]


    m.objective = maximize(xsum(w[i]*y[i] for i in range (I)))

    m += xsum(x[j] for j in range(J)) == K

    for i in range(I):
        m += xsum(x[j] for j in np.where(D[i]==1)[0]) >= y[i]

    m.optimize()

    end = time.time()
    print('----- Output -----')
    print('  런타임 : %s seconds' % float(end-start))
    print('  Optimal coverage points: %g' % m.objective_value)
    solution = []
    for i in range(J):
        if x[i].x ==1:
            solution.append(int(x[i].name[1:]))
    opt_sites = candidate[solution]

    return opt_sites,m.objective_value
```

In [ ]:

```
## 시각화
def plot_input(points):
    '''
    Plot the result
    Input:
        points: input points, Numpy array in shape of [N,2]
        opt_sites: locations K optimal sites, Numpy array in shape of [K,2]
        radius: the radius of circle
    '''
    from matplotlib import pyplot as plt
    fig = plt.figure(figsize=(8,8))
    plt.scatter(points[:,0],points[:,1],c='C0')
    ax = plt.gca()
    ax.axis('equal')
    ax.tick_params(axis='both',left=False, top=False, right=False,
                   bottom=False, labelleft=False, labeltop=False,
                   labelright=False, labelbottom=False)

def plot_result(points,opt_sites,radius):
    '''
    Plot the result
    Input:
        points: input points, Numpy array in shape of [N,2]
        opt_sites: locations K optimal sites, Numpy array in shape of [K,2]
        radius: the radius of circle
    '''
    from matplotlib import pyplot as plt
    fig = plt.figure(figsize=(8,8))
    plt.scatter(points[:,0],points[:,1],c='C0')
    ax = plt.gca()
    plt.scatter(opt_sites[:,0],opt_sites[:,1],c='C1',marker='+')
    for site in opt_sites:
        circle = plt.Circle(site, radius, color='C1',fill=False,lw=2)
        ax.add_artist(circle)
    ax.axis('equal')
    ax.tick_params(axis='both',left=False, top=False, right=False,
                   bottom=False, labelleft=False, labeltop=False,
                   labelright=False, labelbottom=False)
```