from google.colab import drive

cd My Drive

cd Colab Notebooks/

cd MaupTEST/

import numpy as np

import matplotlib.pyplot as plt

from scipy.spatial import KDTree

import pandas as pd

# Prepare study data

import geopandas as gpd

import sys

from scipy.spatial.distance import cdist

from scipy.cluster.hierarchy import fcluster

import random

import csv

# Specify the path to your shapefile

shapefile\_path =  "lung2019usa.shp"

# Read the shapefile using geopandas

gdf = gpd.read\_file(shapefile\_path, encoding="utf-8")

# Now you can work with the GeoDataFrame 'gdf'

points=pd.DataFrame(gdf[['lat','lon','GEOID','Age\_Adjust']])

def DistanceMatrix(data,isNormal=True):

    data2 = data

    if isNormal :

      # Standardize the data to [[-1,1]]

      meanCor = np.mean(data)

      data2 = (data - meanCor) / np.std(data)

    # Calculate the distance matrix

    distance\_matrix = cdist(data2, data2)

    return distance\_matrix

def WeightsMatrix(data,p=1):

  distance\_matrix = np.array([])

  distance\_matrix = DistanceMatrix(data)

  reciprocal\_matrix2 = np.empty(distance\_matrix.shape)

  i=0

  for row in distance\_matrix:

      # Iterate over columns within each row

      j=0

      for element in row:

          if(element != 0):

            reciprocal\_matrix2[i][j] = 1.0/element#np.power(element,p)

          else:

            reciprocal\_matrix2[i][j] = 0

          j=j+1

      i=i+1

  return reciprocal\_matrix2

def Morans\_I(data, spatial\_weights):

    """

    Calculate Moran's I statistic for spatial autocorrelation.

    Parameters:

        data (array-like): A 1-D array or list containing the data values.

        spatial\_weights (array-like): Spatial weights matrix.

    Returns:

        float: Moran's I statistic value.

    """

    n = len(data)

    mean = np.mean(data)

    # Standardize the data

    std\_data = (data - mean) / np.std(data)

    # Calculate Moran's I numerator

    numerator = np.sum(spatial\_weights \* np.outer(std\_data, std\_data))

    # Calculate Moran's I denominator

    denominator = np.sum(spatial\_weights)

    # Calculate Moran's I

    morans\_i = (n / denominator) \* (numerator / n)

    return morans\_i

def calinski\_harabasz\_score(X, labels):

    """

    Calculate the Calinski-Harabasz index.

    Parameters:

    X : ndarray, shape (n\_samples, n\_features)

        Array of feature vectors.

    labels : ndarray, shape (n\_samples,)

        Cluster labels for each sample.

    Returns:

    score : float

        Calinski-Harabasz index

    """

    # Calculate centroids

    centroids = np.array([np.mean(X[labels == label], axis=0) for label in np.unique(labels)])

    # Calculate global mean

    global\_mean = np.mean(X, axis=0)

    # Calculate within-cluster variance

    within\_cluster\_variance = np.sum([np.sum(np.linalg.norm(X[labels == label] - np.mean(X[labels == label], axis=0)) \*\* 2)

                                      for label in np.unique(labels)])

    # Calculate between-cluster variance

    between\_cluster\_variance = np.sum([len(X[labels == label]) \* np.linalg.norm(np.mean(X[labels == label], axis=0) - global\_mean) \*\* 2

                                       for label in np.unique(labels)])

    # Calculate Calinski-Harabasz index

    score = (between\_cluster\_variance / within\_cluster\_variance) \* ((len(X) - len(np.unique(labels))) / (len(np.unique(labels)) - 1))

    return score

# Visualizing the zoning

def Draw\_zoneFig(dataXY, zoneCentriodXY,zones):

    plt.figure(figsize=(8, 6))

    plt.scatter(dataXY.iloc[:, 1], dataXY.iloc[:, 0], c = zones, cmap='viridis', s=50, alpha=0.6)

    if not zoneCentriodXY:

       if len(zoneCentriodXY)>1 :

           plt.scatter(zoneCentriodXY.iloc[:, 1], zoneCentriodXY.iloc[:,0], c='red', marker='x', s=200, label='Zone Centriod')

    plt.title('KD-tree Zoning')

    plt.xlabel('X')

    plt.ylabel('Y')

    plt.legend()

    plt.show()

def drawLineFig(data):

  # display the trend line between ZoneNum and MoransI

  # get x and y values

  x\_values = data.iloc[:, 0]

  y\_values = data.iloc[:, 1]

  # Create a line plot

  plt.plot(x\_values, y\_values, marker='o', linestyle='-')

  # Add labels and title

  plt.xlabel('zoneNums')

  plt.ylabel('MoransI')

  plt.title('Change Trend of MoransI to zoneNums')

  # Display the plot

  plt.grid(True)

  plt.show()

# Randomly select NumZones  points as References Points to Create a new Zoning

# it return a Zoned Results withs Aggragated AgeAdjuestLcm

def RandomZone2(data,NumZones = 4,zoneIndex = [],stFieldName='Age\_Adjust'):

  #zoneIndex is zone\_assignments

   # n=10

    n = int(NumZones)

    #reference\_points = np.empty((n,2))

    reference\_points = [[0, 0] for \_ in range(NumZones)]

    candidate\_points = [[0, 0] for \_ in range(NumZones)]

    if len(zoneIndex) ==0:

        candidate\_points = data[data['Age\_Adjust'] != 0][['lon','lat']]

        # Randomly select points

        reference\_points = random.sample(candidate\_points.values.tolist(), NumZones)

    else:

      #  print("The array is not empty.")

        zoneName="zones"+ str(NumZones)

        data[zoneName] = pd.DataFrame(zoneIndex)

        #reference\_points = []

        for i in range(NumZones):

          candidate\_points[i] = data[(points['Age\_Adjust'] != 0) & (data[zoneName] == i)][['lon', 'lat']]

          # selected\_indices  = np.random.choice(len(selection\_points), size=1, replace=False)

          reference\_points[i] = random.sample(candidate\_points[i].values.tolist(), 1)[0]

              # Building KD-tree

   # print(reference\_points)

    kdtree = KDTree(reference\_points)

    zone\_assignments = kdtree.query(data[['lon','lat']])[1]

    # Assigning each sample to its nearest reference point

   #zonedLCM = data[data["Age\_Adjust"] !=0 ]["Age\_Adjust"].values

    #labels = data[data["Age\_Adjust"] !=0 ][zoneName].values

    theme\_zones=data["Age\_Adjust"].values

    zonedLCM = theme\_zones[theme\_zones!=0]

    labels = zone\_assignments[theme\_zones!=0]

    #print(len(zonedLCM),len(labels))

    calinski\_harabasz\_score2 = calinski\_harabasz\_score(zonedLCM,labels)

    return calinski\_harabasz\_score2,zone\_assignments

def simulated\_annealingZoning(data,NumZones=4,stFieldName='Age\_Adjust',max\_iterations = 1000):

    # BestZoning for given specific zone numbers.

    # Building a balanced KD-tree as a intial zoning

    current\_cost,current\_zones = RandomZone2(data,NumZones)

    best\_cost = current\_cost

    best\_zones = current\_zones

    t=1

    def temperature\_schedule(t):

      return 1 / np.log(t + 1)

    def AcceptP(delta,t):

      p=0

      if delta<0:

        p = 1

      else:

        p = np.exp(-delta / temperature\_schedule(t))

      return p

   # Perform simulated annealing optimization

    for t in range(max\_iterations):

        # Generate a new candidate configuration by perturbing the current configuration

        new\_cost,new\_zones = RandomZone2(data,NumZones,current\_zones)

        delta\_value = new\_cost-current\_cost

        # Accept the candidate with a probability determined by the Metropolis criterion

        if AcceptP(delta\_value,t ) > np.random.rand() :

            current\_zones = new\_zones

            current\_cost = new\_cost

        # Update the best solution found so far

        if current\_cost < best\_cost:

          #  best\_configuration = new\_configuration

            best\_cost = current\_cost

            best\_zones = current\_zones

            print("zoningNums:",NumZones,"rounds:",t,"calinski\_harabasz\_score:",round(best\_cost,3))

    return best\_cost,best\_zones

def AllBestZoning(data,stFieldName='Age\_Adjust',max\_iterations=1000):

    # BestZoning across any zone numbers.

    zoneNum\_MoransI=[]

    MaxZoneNum=int(len(data[data['Age\_Adjust'] !=0])/2)

    myArrayData=data.values

   # weights = WeightsMatrix(data[['lon','lat']])

   # best\_cost = compute\_autocorrelation(data["Age\_Adjust"].values,weights)

    #zoneNum\_MoransI.append([1, best\_cost])

    #print("zone 1 moransI:",best\_cost)

    for i in range(1000,1300,50):

      calinski\_harabasz\_score1,zones = simulated\_annealingZoning(data,i)

      zoneName="zones"+ str(i)

      data[zoneName]= pd.DataFrame(zones)

      # Grouping by 'Category' column and applying different aggregation methods to different columns

      # Define a custom aggregation function to aggregate string values as a list

      def aggregate\_as\_list(series):

          return list(series)

      # Define custom function to calculate mean excluding zero values

      def mean\_without\_zero(x):

          return x[x != 0].mean()

      agg\_methods1 = {

      'lat': 'mean',

      'lon': 'mean',

      'GEOID': aggregate\_as\_list,

      'Age\_Adjust': mean\_without\_zero

      }

      zoneResults = data.groupby(zoneName).agg(agg\_methods1)

    # zoneResults = pd.concat([grouped\_stats1, grouped\_stats2], axis=1)

      weights = WeightsMatrix(zoneResults[['lon','lat']][zoneResults["Age\_Adjust"].notna()].values)

      zonedLCM = zoneResults["Age\_Adjust"][zoneResults["Age\_Adjust"].notna()].values.flatten()

      MoransI = Morans\_I(zonedLCM,weights)

      zoneNum\_MoransI.append([i, round(MoransI,3)])

      print("BestZoning at Numbers:",i,"MoransI:",round(MoransI,3))

    return zoneNum\_MoransI

import warnings

import time

# Ignore all warnings

warnings.filterwarnings("ignore")

start\_time = time.time()

zoneNum\_MoransI = AllBestZoning(points)

end\_time = time.time()

run\_time = end\_time - start\_time

print("running time：", run\_time, "second")

# define CSV path

csv\_file = 'data.csv'

# write CSVfile

with open(csv\_file, 'w', newline='') as file:

    writer = csv.writer(file)

    writer.writerows(zoneNum\_MoransI)

points.to\_csv('zones2.csv')

drawLineFig(pd.DataFrame(zoneNum\_MoransI))