## 2.7.6 不平等性和空间隔离

### 2.7.6.1 数据预处理

土地利用来源于芝加哥大都会规划署（Chicago Metropolitan Agency for Planning, CMAP），为[2015年更新数据](https://www.cmap.illinois.gov/data/land-use/inventory)①，由规划署提供的土地使用清单类别[1]，包括一级分类和二级分类。每五年更新一次的区域土地利用信息层数据由德克萨斯州中北部政府委员会（North Central Texas Council of Governments, NCTCOG）的信息部门（Information Services department，RIS）负责开发。

| 一级分类 | 二级分类 |
| --- | --- |
| 住宅（RESIDENTIAL） | █ 单户住宅（Single\_Family\_Residential） |
|  | █ 独栋单户住宅（Single\_Family\_Detached） |
|  | █ 连排单户住宅（Single\_Family\_Attached） |
|  | █ 多户住宅（Multi\_Family） |
|  | █ 托车移动住宅（Mobile\_Home\_Parks\_and\_Trailer\_Courts） |
|  | █ 相关住宅用地（Associated\_Residential\_Land） |
|  | █ 住宅开发区内公共开放空间（Common\_Open\_Space\_in\_a\_Residential\_Development） |
| 商业（COMMERCIAL） | █ 主要零售服务（Primarily\_Retail\_Service） |
|  | █ 购物广场（Shopping\_Malls） |
|  | █ 区域社区零售中心（Regional\_Community\_Retail\_Centers） |
|  | █ 单一大型零售点（Single\_Large\_Site\_Retail） |
|  | █ 城市混合用地（Urban\_Mix） |
|  | █ 含住宅的城市混合用地（Urban\_Mix\_w\_Residential\_Component） |
|  | █ 办公（Office） |
|  | █ 文化娱乐（Cultural\_Entertainment） |
|  | █ 酒店旅馆（Hotel\_Motel） |
| 机构（INSTITUTIONAL） | █ 医疗设施（Medical\_Facilities） |
|  | █ K\_12教育设施（K\_12\_Educational\_Facilities）/含幼儿园，小学和中学 |
|  | █ 大学教育设施（Post\_Secondary\_Educational\_Facilities） |
|  | █ 政府行政和服务（Government\_Administration\_and\_Services） |
|  | █ 监狱和惩教设施（Prison\_and\_Correctional\_Facilities） |
|  | █ 宗教设施（Religious\_Facilities） |
|  | █ 墓地（Cemeteries） |
|  | █ 其他机构（Other\_Institutional） |
|  | █ 国家实验室（National\_Laboratory） |
| 工业（INDUSTRIAL） | █ 矿物提取（Mineral\_Extraction） |
|  | █ 一般工业（General\_Industrial） |
|  | █ 通用电气工业（Industrial\_G\_E） |
|  | █ 制造加工（Manufacturing\_Processing） |
|  | █ 仓储配送（Warehousing\_Distribution） |
|  | █ 不确定用地（Flex\_or\_Indeterminate） |
|  | █ 贮存（Storage） |
| 交通、通讯、公共事业和垃圾处理（TRANS COMM UTIL WASTE） | █ 运输通行权（Transportation\_Right\_of\_Way） |
|  | █ 铁路（Rail\_ROW） |
|  | █ 高速公路（Roadway） |
|  | █ 其他具有相关设施的线性交通（Other\_Linear\_Transportation\_with\_Associated\_Facilities） |
|  | █ 航空运输（Aircraft\_Transportation） |
|  | █ 独立停车场（Independent\_Automobile\_Parking） |
|  | █ 通讯（Communication） |
|  | █ 公用事业和废物处理设施（Utilities\_and\_Waste\_Facilities） |
|  | █ 公用事业通行权（Utility\_Right\_of\_Way） |
|  | █ 废水处理设施（Wastewater\_Treatment\_Facility） |
|  | █ 垃圾填埋场（Landfill） |
|  | █ 其它公用设施废物（Other\_Utility\_Waste） |
|  | █ 雨水管理（Stormwater\_Management） |
|  | █ 联运设施（Intermodal\_Facility） |
| 农业（AGRICULTURE） | █ 农业（AGRICULTURE） |
| 开放空间（OPEN SPACE） | █ 开放空间主要娱乐设施（Open\_Space\_Primarily\_Recreation）/以城市公园为主 |
|  | █ 高尔夫球场（Golf\_Course） |
|  | █ 开放空间主要保护区（Open\_Space\_Primarily\_Conservation） |
|  | █ 非公共开放空间（Non\_Public\_Open\_Space） |
|  | █ 步道或绿道（Trail\_or\_Greenway） |
| 空置（VACANT UNDER CONSTRUCTION） | █ 空置未开发土地（Vacant\_Undeveloped\_Land） |
|  | █ 空置住宅用地（Vacant\_Residential\_Land） |
|  | █ 空置商业用地（Vacant\_Commercial\_Land） |
|  | █ 空置工业用地（Vacant\_Industrial\_Land） |
|  | █ 其它空置（Other\_Vacant） |
| 在建（Under\_Construction） | █ 在建住宅（Under\_Construction\_Residential） |
|  | █ 在建商业（Under\_Construction\_Commercial） |
|  | █ 在建工业（Under\_Construction\_Industrial） |
|  | █ 其他或未知在建（Under\_Construction\_Other\_or\_Unknown） |
| 水体（WATER） | █ 水体（Water） |
| 非包裹区域（NON PARCEL AREAS） | █ 非包裹区域（NON\_PARCEL\_AREAS）/以道路为主 |
| 未分类（NOT CLASSIFIED） | █ 未分类（Not\_Classifiable） |

下载的土地利用数据为矢量SHP格式，大小约900MB。使用土地利用数据的目的为样方统计用于后续分析，因此需要将其转化为栅格数据，建立连续多个Polygon样方进行区域统计，获得各个样方中各土地利用类型的栅格单元数量。土地利用地图数据打印时，通常需要对不同土地利用类型赋予不同的颜色方便观察，因此也需要建立颜色的映射表。为了方便数据的管理和在QGIS等地理信息平台中加载数据，本次实验通过[pgAdmin](https://www.pgadmin.org/)②工具建立PostgreSQL数据库，将主要的数据写入到数据库中，方便统一管理后续调用。

参数管理使用AttrDict()方法（具体查看“Cityscapes数据集——参数管理”一节）。db子属性存储数据库信息；gi子属性存储地理坐标投影信息；data子属性存储数据文件路径。

from database import postSQL2gpd,gpd2postSQL  
from util\_misc import AttrDict  
\_\_C=AttrDict()   
args=\_\_C  
  
\_\_C.db=AttrDict()   
\_\_C.db.UN='postgres'  
\_\_C.db.PW='123456'  
\_\_C.db.DB='inequalityNsegregation'  
\_\_C.db.GC='geometry'   
\_\_C.db.db\_info=dict(geom\_col=args.db.GC,myusername=args.db.UN,mypassword=args.db.PW,mydatabase=args.db.DB)  
  
\_\_C.gi=AttrDict()  
\_\_C.gi.Chicago\_epsg=32616  
\_\_C.gi.epsg\_wgs84=4326  
  
\_\_C.data=AttrDict()  
\_\_C.data.landuse='G:\data\landuse\_Chicago\LUI15\_shapefile\_v1' # 芝加哥2015年土地利用SHP格式文件  
\_\_C.data.landuse\_LB='E:\data\Chicago\_landuse\Chicago\_landuse\_LB\landuse\_LB\_uncropped.shp' # 土地利用分类整数编码后的文件  
\_\_C.data.landuse\_tif='E:\data\Chicago\_landuse\landuse.tif' # 转化为栅格的土地利用文件  
\_\_C.data.Chicago\_boundaries\_city='./data/Chicago\_boundaries\_city/Chicago\_boundaries\_city.shp' # 芝加哥城边界SHP格式文件

#### 1）土地利用数据读取和编码

GeoPandas库提供了read\_file方法读取SHP、GPKG和GeoJson等数据格式文件，且提供了clip方法裁切地理信息数据，为了方便数据操作，进行功能的整合，定义shp2gdf()函数执行SHP格式文件的读取、裁切等数据处理流程。返回的GeoDataFrame格式数据投影，如果不进行裁切则与读取的数据投影同；如果进行裁切，则与裁切对象的投影同。

def shp2gdf(fn,boundary=None,encoding='utf-8'):   
 '''  
 转换.shp地理信息数据为GeoDataFrame(geopandas)数据格式，可以配置投影  
  
 Parameters  
 ----------  
 fn : string  
 SHP文件路径.  
 boundary : .shp, optional  
 配置裁切边界. The default is None.  
 encoding : string, optional  
 配置编码. The default is 'utf-8'.  
  
 Returns  
 -------  
 GeoDataFrame  
 读取SHP格式文件为GeoDataFrame格式返回.  
  
 '''  
 import geopandas as gpd  
 from tqdm import tqdm  
 tqdm.pandas()   
   
 shp\_gdf=gpd.read\_file(fn,encoding=encoding)   
 if boundary is not None:   
 shp\_gdf['mask']=shp\_gdf.geometry.progress\_apply(lambda row:row.is\_valid)  
 shp\_gdf=shp\_gdf[shp\_gdf['mask']==True]  
 shp\_clip\_gdf=gpd.clip(shp\_gdf.to\_crs(boundary.crs),boundary)   
 return shp\_clip\_gdf  
 else:  
 return shp\_gdf

landuse\_gdf=shp2gdf(args.data.landuse)  
landuse\_gdf.to\_crs(args.gi.Chicago\_epsg,inplace=True)  
landuse\_CH\_gdf=landuse\_gdf.copy(deep=True)

读取的数据中LANDUSE列提供了土地利用分类的编码，为了对位名称，根据规划署提供的土地使用清单类别建立映射字典，由土地利用类型编号为键，值为一个列表，表示为’landuse\_CH\_mapping’,‘landuse\_parentClass\_mapping’,‘landuse\_parentClass\_mapping\_2’,’landuse\_EN\_mapping’4个值，对应土地利用二级分类中文名、一级分类中文名、用于地图打印调整的一级分类中文名和英文名。将字典转换为DataFrame格式数据，根据需要提取列，例如landuse\_CH\_mapping为二级分类映射字典；landuse\_parentClass\_mapping为一级分类映射字典。

landuse\_mapping={   
 '1000':['城市化区域','城市化区域','城市化区域','URBANIZED'],# 'URBANIZED',空值，父类  
 '1110':['单户住宅','住宅','住宅','Single\_Family\_Residential'], # 'Single\_Family\_Residential',(SFR)空值，父类  
 '1111':['独栋单户住宅','住宅','住宅','Single\_Family\_Detached'],# 'Single\_Family\_Detached',58787 rows  
 '1112':['连排单户住宅','住宅','住宅','Single\_Family\_Attached'], # 'Single\_Family\_Attached',3874 rows  
 '1130':['多户住宅','住宅','住宅','Multi\_Family'], # 'Multi\_Family',58256 rows  
 '1140':['托车移动住宅','住宅','住宅','Mobile\_Home\_Parks\_and\_Trailer\_Courts'],# 'Mobile\_Home\_Parks\_and\_Trailer\_Courts',2 rows  
 '1150':['相关住宅用地','住宅','住宅','Associated\_Residential\_Land'],# 'Associated\_Residential\_Land',空值  
 '1151':['住宅开发区内公共开放空间','住宅','住宅','Common\_Open\_Space\_in\_a\_Residential\_Development'],# 'Common\_Open\_Space\_in\_a\_Residential\_Development',571 rows  
 '1200':['商业','商业','商业','COMMERCIAL'], # 'COMMERCIAL',空值，父类  
 '1210':['主要零售服务','商业','商业','Primarily\_Retail\_Service'],# 'Primarily\_Retail\_Service',空值，父类  
 '1211':['购物广场','商业','商业','Shopping\_Malls'],# 'Shopping\_Malls',3 rows  
 '1212':['区域社区零售中心','商业','商业','Regional\_Community\_Retail\_Centers'],# 'Regional\_Community\_Retail\_Centers',78 rows  
 '1214':['单一大型零售点','商业','商业','Single\_Large\_Site\_Retail'],# 'Single\_Large\_Site\_Retail',31 rows  
 '1215':['城市混合用地','商业','商业','Urban\_Mix'],# 'Urban\_Mix',10889 rows  
 '1216':['含住宅的城市混合用地','商业','商业','Urban\_Mix\_w\_Residential\_Component'],# 'Urban\_Mix\_w\_Residential\_Component',城市混合用地  
 '1220':['办公','商业','商业','Office'],# 'Office',610 rows  
 '1240':['文化娱乐','商业','商业','Cultural\_Entertainment'],# 'Cultural\_Entertainment',276 rows  
 '1250':['酒店旅馆','商业','商业','Hotel\_Motel'],# 'Hotel\_Motel',166 rows  
 '1300':['机构','机构','机构','INSTITUTIONAL'],# 'INSTITUTIONAL',空值，父类  
 '1310':['医疗设施','机构','机构','Medical\_Facilities'],# 'Medical\_Facilities',319 rows  
 '1320':['教育设施','机构','机构','Educational\_Facilities'],# 'Educational\_Facilities',空值，父类  
 '1321':['K\_12教育设施','机构','机构','K\_12\_Educational\_Facilities'],# 'K\_12\_Educational\_Facilities',幼稚园、小学和中学教育合在一起的统称,1356 rows  
 '1322':['大学教育设施','机构','机构','Post\_Secondary\_Educational\_Facilities'],# 'Post\_Secondary\_Educational\_Facilities',296 rows   
 '1330':['政府行政和服务','机构','机构','Government\_Administration\_and\_Services'],# 'Government\_Administration\_and\_Services',522 rows  
 '1340':['监狱和惩教设施','机构','机构','Prison\_and\_Correctional\_Facilities'],# 'Prison\_and\_Correctional\_Facilities',8 rows   
 '1350':['宗教设施','机构','机构','Religious\_Facilities'],# 'Religious\_Facilities',2799 rows   
 '1360':['墓地','机构','机构','Cemeteries'],# 'Cemeteries',36 rows  
 '1370':['其他机构','机构','机构','Other\_Institutional'],# 'Other\_Institutional',348 rows   
 '1380':['国家实验室','机构','机构','National\_Laboratory'],# 'National\_Laboratory',空值(未裁切文件含值)  
 '1400':['工业','工业','工业','INDUSTRIAL'],# 'INDUSTRIAL',空值，父类  
 '1410':['矿物提取','工业','工业','Mineral\_Extraction'],# 'Mineral\_Extraction',5 rows   
 '1420':['一般工业','工业','工业','General\_Industrial'],# 'General\_Industrial',2746 rows  
 '1430':['通用电气工业','工业','工业','Industrial\_G\_E'],# 'Industrial\_G\_E',General Electric(GE),空值，父类  
 '1431':['制造加工','工业','工业','Manufacturing\_Processing'],# 'Manufacturing\_Processing',253 rows  
 '1432':['仓储配送','工业','工业','Warehousing\_Distribution'],# 'Warehousing\_Distribution',104 rows, A storage warehouse is preferred for industrial or commercial purposes.   
 '1433':['不确定用地','工业','工业','Flex\_or\_Indeterminate'],# 'Flex\_or\_Indeterminate',128 rows   
 '1450':['贮存','工业','工业','Storage'],# 'Storage',165 rows, A storage facility is used for personal and business needs.  
 '1500':['转运与垃圾管理','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','TRANS\_COMM\_UTIL\_WASTE'],# 'TRANS\_COMM\_UTIL\_WASTE',空值，父类  
 '1510':['运输通行权','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','Transportation\_Right\_of\_Way'],# 'Transportation\_Right\_of\_Way',空值  
 '1511':['铁路','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','Rail\_ROW'],# 'Rail\_ROW',1725 rows   
 '1512':['高速公路','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','Roadway'],# 'Roadway',3225 rows   
 '1520':['其他具有相关设施的线性交通','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','Other\_Linear\_Transportation\_with\_Associated\_Facilities'],# 'Other\_Linear\_Transportation\_with\_Associated\_Facilities',438 rows  
 '1530':['航空运输','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','Aircraft\_Transportation'],# 'Aircraft\_Transportation',88 rows  
 '1540':['独立停车场','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','Independent\_Automobile\_Parking'],# 'Independent\_Automobile\_Parking',260 rows   
 '1550':['通讯','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','Communication'],# 'Communication',203 rows  
 '1560':['公用事业和废物处理设施','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','Utilities\_and\_Waste\_Facilities'],# 'Utilities\_and\_Waste\_Facilities',空值，父类  
 '1561':['公用事业通行权','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','Utility\_Right\_of\_Way'],# 'Utility\_Right\_of\_Way',106 rows  
 '1562':['废水处理设施','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','Wastewater\_Treatment\_Facility'],# 'Wastewater\_Treatment\_Facility',8 rows   
 '1563':['垃圾填埋场','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','Landfill'],# 'Landfill',107 rows  
 '1564':['其它公用设施废物','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','Other\_Utility\_Waste'],# 'Other\_Utility\_Waste',177 rows  
 '1565':['雨水管理','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','Stormwater\_Management'],# 'Stormwater\_Management',11 rows  
 '1570':['联运设施','交通\_通讯\_公共事业和垃圾处理','交通\_通讯\_公共事业和垃圾处理','Intermodal\_Facility'],# 'Intermodal\_Facility',72 rows  
 '2000':['农业','农业','其它','AGRICULTURE'],# 'AGRICULTURE',18 rows  
 '3000':['开放空间','开放空间','开放空间','OPEN\_SPACE'],# 'OPEN\_SPACE',空值，父类  
 '3100':['开放空间主要娱乐设施','开放空间','开放空间','Open\_Space\_Primarily\_Recreation'],# 'Open\_Space\_Primarily\_Recreation',978 rows，公园  
 '3200':['高尔夫球场','开放空间','开放空间','Golf\_Course'],# 'Golf\_Course',19 rows  
 '3300':['开放空间主要保护区','开放空间','开放空间','Open\_Space\_Primarily\_Conservation'],#' Open\_Space\_Primarily\_Conservation',163 rows  
 '3400':['非公共开放空间','开放空间','开放空间','Non\_Public\_Open\_Space'],# 'Non\_Public\_Open\_Space',3 rows  
 '3500':['步道或绿道','开放空间','开放空间','Trail\_or\_Greenway'],# 'Trail\_or\_Greenway',31 rows  
 '4000':['空置在建','空置','其它','VACANT\_UNDER\_CONSTRUCTION'],# 'VACANT\_UNDER\_CONSTRUCTION',空值，父类  
 '4100':['空置未开发土地','空置','其它','Vacant\_Undeveloped\_Land'],# 'Vacant\_Undeveloped\_Land',空值  
 '4110':['空置住宅用地','空置','其它','Vacant\_Residential\_Land'],# 'Vacant\_Residential\_Land',18095 rows  
 '4120':['空置商业用地','空置','其它','Vacant\_Commercial\_Land'],# 'Vacant\_Commercial\_Land',3811 rows  
 '4130':['空置工业用地','空置','其它','Vacant\_Industrial\_Land'],# 'Vacant\_Industrial\_Land',1155 rows  
 '4140':['其它空置','空置','其它','Other\_Vacant'],# 'Other\_Vacant',1126 rows   
 '4200':['在建','在建','其它','Under\_Construction'],# 'Under\_Construction',空值，父类  
 '4210':['在建住宅','在建','其它','Under\_Construction\_Residential'],# 'Under\_Construction\_Residential',192 rows  
 '4220':['在建商业','在建','其它','Under\_Construction\_Commercial'],# 'Under\_Construction\_Commercial',35 rows  
 '4230':['在建工业','在建','其它','Under\_Construction\_Industrial'],# 'Under\_Construction\_Industrial',4 rows   
 '4240':['其他或未知在建','在建','其它','Under\_Construction\_Other\_or\_Unknown'],# 'Under\_Construction\_Other\_or\_Unknown',  
 '5000':['水体','水体','水体','Water'],# 'Water',172 rows  
 '6000':['非包裹区域','非包裹区域','非包裹区域','NON\_PARCEL\_AREAS'],# 'NON\_PARCEL\_AREAS',2167 rows，道路为主  
 '9999':['未分类','未分类','其它','Not\_Classifiable'],# 'Not\_Classifiable', 58 rows   
 }

import pandas as pd  
landuse\_mapping\_df=pd.DataFrame.from\_dict(landuse\_mapping,orient='index',columns=['landuse\_CH\_mapping','landuse\_parentClass\_mapping','landuse\_parentClass\_mapping\_2','landuse\_EN\_mapping'])  
# landuse\_mapping\_df.reset\_index(names=['ID'],inplace=True)  
landuse\_mapping\_df

|  | **landuse\_CH\_mapping** | **landuse\_parentClass\_mapping** | **landuse\_parentClass\_mapping\_2** | **landuse\_EN\_mapping** |
| --- | --- | --- | --- | --- |
| **1000** | 城市化区域 | 城市化区域 | 城市化区域 | URBANIZED |
| **1110** | 单户住宅 | 住宅 | 住宅 | Single\_Family\_Residential |
| **1111** | 独栋单户住宅 | 住宅 | 住宅 | Single\_Family\_Detached |
| **1112** | 连排单户住宅 | 住宅 | 住宅 | Single\_Family\_Attached |
| **1130** | 多户住宅 | 住宅 | 住宅 | Multi\_Family |
| **...** | ... | ... | ... | ... |
| **4230** | 在建工业 | 在建 | 其它 | Under\_Construction\_Industrial |
| **4240** | 其他或未知在建 | 在建 | 其它 | Under\_Construction\_Other\_or\_Unknown |
| **5000** | 水体 | 水体 | 水体 | Water |
| **6000** | 非包裹区域 | 非包裹区域 | 非包裹区域 | NON\_PARCEL\_AREAS |
| **9999** | 未分类 | 未分类 | 其它 | Not\_Classifiable |

73 rows × 4 columns

landuse\_CH\_mapping=landuse\_mapping\_df.landuse\_CH\_mapping.to\_dict()  
landuse\_parentClass\_mapping=landuse\_mapping\_df.landuse\_parentClass\_mapping.to\_dict()

将一二级土地利用分类名对位到landuse\_CH\_gdf数据中。

landuse\_CH\_gdf['CHname']=landuse\_CH\_gdf.LANDUSE.apply(lambda row:landuse\_CH\_mapping[row])   
landuse\_CH\_gdf['parentClass']=landuse\_CH\_gdf.LANDUSE.apply(lambda row:landuse\_parentClass\_mapping[row])   
landuse\_CH\_gdf

|  | **FIRST\_COUN** | **LANDUSE** | **LANDUSE2** | **OS\_MGMT** | **FAC\_NAME** | **PLATTED** | **MODIFIER** | **Shape\_Leng** | **Shape\_Area** | **geometry** | **CHname** | **parentClass** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 031 | 1111 | None | None | None | None | None | 517.742137 | 16718.995065 | POLYGON ((405755.086 4666594.715, 405736.553 4... | 独栋单户住宅 | 住宅 |
| **1** | 031 | 1111 | None | None | None | None | None | 477.507596 | 11320.358746 | POLYGON ((406011.595 4666666.401, 406000.000 4... | 独栋单户住宅 | 住宅 |
| **2** | 031 | 1111 | None | None | None | None | None | 484.025949 | 9845.643361 | POLYGON ((406660.786 4666744.982, 406602.816 4... | 独栋单户住宅 | 住宅 |
| **3** | 031 | 1111 | None | None | None | None | None | 700.439764 | 17077.187161 | POLYGON ((406481.048 4666788.526, 406429.637 4... | 独栋单户住宅 | 住宅 |
| **4** | 031 | 1111 | None | None | None | None | None | 340.958375 | 4876.261817 | POLYGON ((406280.699 4666876.317, 406239.891 4... | 独栋单户住宅 | 住宅 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **535173** | 197 | 6000 | None | None | None | None | None | 269.275557 | 3326.588780 | POLYGON ((412872.480 4619999.996, 412836.225 4... | 非包裹区域 | 非包裹区域 |
| **535174** | 197 | 6000 | None | None | None | None | None | 61.452204 | 40.424097 | POLYGON ((412435.632 4620000.000, 412426.321 4... | 非包裹区域 | 非包裹区域 |
| **535175** | 197 | 6000 | None | None | None | None | None | 9155.705032 | 271973.258341 | POLYGON ((413999.999 4619999.999, 413889.511 4... | 非包裹区域 | 非包裹区域 |
| **535176** | 197 | 6000 | None | None | None | None | None | 3895.597432 | 129057.116542 | POLYGON ((413439.307 4620000.000, 413419.194 4... | 非包裹区域 | 非包裹区域 |
| **535177** | 197 | 6000 | None | None | None | None | None | 3315.837148 | 16338.850243 | POLYGON ((414022.299 4620000.000, 413999.999 4... | 非包裹区域 | 非包裹区域 |

535178 rows × 12 columns

将读取并加入一二级土地利用分类名的土地利用数据写入到数据中。

gpd2postSQL(landuse\_CH\_gdf,'landuse\_CH\_uncropped',\*\*args.db.db\_info)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is landuse\_CH\_uncropped.

读取存入到数据库的土地利用数据。

landuse\_CH\_uncropped\_gdf=postSQL2gpd(table\_name='landuse\_CH\_uncropped',\*\*args.db.db\_info)

C:\Users\richi\anaconda3\envs\usda\lib\site-packages\pandas\io\sql.py:580: SAWarning: Did not recognize type 'geometry' of column 'geometry'  
 pandas\_sql.meta.reflect(bind=pandas\_sql.connectable, only=[sql])  
  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is landuse\_CH\_uncropped.

* 编码土地利用类型

使用scikit-learn库提供的LabelEncoder方法编码土地利用类型。定义gdf\_label\_encoder()函数可以直接操作DataFrame格式数据列，并把编码结果作为新列加入。

def gdf\_label\_encoder(gdf,columns):  
 '''  
 给定(Geo)DataFrame数据，指定一个或多个列，转换为整数值编码，增加新列标识符为LB\_  
  
 Parameters  
 ----------  
 gdf : (Geo)DataFrame  
 待处理的数据.  
 columns : List(String)  
 列名列表.  
  
 Returns  
 -------  
 gdf : (Geo)DataFrame  
 返回含编码列的数据.  
 lb\_mapping\_dic : Dict  
 编码映射字典.  
  
 '''  
 from sklearn.preprocessing import LabelEncoder  
   
 lb\_mapping\_dic={}  
 for col in columns:  
 lb\_make=LabelEncoder()  
 gdf['LB\_'+col]=lb\_make.fit\_transform(gdf[col])  
 lb\_mapping=dict(zip(lb\_make.classes\_, lb\_make.transform(lb\_make.classes\_)))  
 lb\_mapping\_dic[col]=lb\_mapping  
 return gdf,lb\_mapping\_dic  
  
landuse\_LB\_uncropped\_gdf,landuse\_LB\_uncropped\_mapping=gdf\_label\_encoder(landuse\_CH\_uncropped\_gdf,columns=['LANDUSE'])  
landuse\_LB\_uncropped\_gdf.head()

|  | **FIRST\_COUN** | **LANDUSE** | **LANDUSE2** | **OS\_MGMT** | **FAC\_NAME** | **PLATTED** | **MODIFIER** | **Shape\_Leng** | **Shape\_Area** | **geometry** | **CHname** | **parentClass** | **LB\_LANDUSE** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 031 | 1111 | None | None | None | None | None | 517.742137 | 16718.995065 | POLYGON ((405755.359 4666594.654, 405736.826 4... | 独栋单户住宅 | 住宅 | 0 |
| **1** | 031 | 1111 | None | None | None | None | None | 477.507596 | 11320.358746 | POLYGON ((406011.869 4666666.341, 406000.274 4... | 独栋单户住宅 | 住宅 | 0 |
| **2** | 031 | 1111 | None | None | None | None | None | 484.025949 | 9845.643361 | POLYGON ((406661.062 4666744.922, 406603.092 4... | 独栋单户住宅 | 住宅 | 0 |
| **3** | 031 | 1111 | None | None | None | None | None | 700.439764 | 17077.187161 | POLYGON ((406481.324 4666788.466, 406429.912 4... | 独栋单户住宅 | 住宅 | 0 |
| **4** | 031 | 1111 | None | None | None | None | None | 340.958375 | 4876.261817 | POLYGON ((406280.974 4666876.257, 406240.166 4... | 独栋单户住宅 | 住宅 | 0 |

对LANDUSE列土地利用类型编号编码，通过landuse\_CH\_mappin映射字典也建立编码和土地利用类型名称的字典映射。

landuse\_LB\_uncropped\_mapping\_chname={landuse\_CH\_mapping[k]:v for k,v in landuse\_LB\_uncropped\_mapping['LANDUSE'].items()}  
landuse\_LB\_uncropped\_mapping.update({'CHname':landuse\_LB\_uncropped\_mapping\_chname})  
landuse\_LB\_uncropped\_mapping  
print(landuse\_LB\_uncropped\_mapping)

{'LANDUSE': {'1111': 0, '1112': 1, '1130': 2, '1140': 3, '1151': 4, '1211': 5, '1212': 6, '1214': 7, '1215': 8, '1216': 9, '1220': 10, '1240': 11, '1250': 12, '1310': 13, '1321': 14, '1322': 15, '1330': 16, '1340': 17, '1350': 18, '1360': 19, '1370': 20, '1380': 21, '1410': 22, '1420': 23, '1431': 24, '1432': 25, '1433': 26, '1450': 27, '1511': 28, '1512': 29, '1520': 30, '1530': 31, '1540': 32, '1550': 33, '1561': 34, '1562': 35, '1563': 36, '1564': 37, '1565': 38, '1570': 39, '2000': 40, '3100': 41, '3200': 42, '3300': 43, '3400': 44, '3500': 45, '4110': 46, '4120': 47, '4130': 48, '4140': 49, '4210': 50, '4220': 51, '4230': 52, '4240': 53, '5000': 54, '6000': 55, '9999': 56}, 'CHname': {'独栋单户住宅': 0, '连排单户住宅': 1, '多户住宅': 2, '托车移动住宅': 3, '住宅开发区内公共开放空间': 4, '购物广场': 5, '区域社区零售中心': 6, '单一大型零售点': 7, '城市混合用地': 8, '含住宅的城市混合用地': 9, '办公': 10, '文化娱乐': 11, '酒店旅馆': 12, '医疗设施': 13, 'K\_12教育设施': 14, '大学教育设施': 15, '政府行政和服务': 16, '监狱和惩教设施': 17, '宗教设施': 18, '墓地': 19, '其他机构': 20, '国家实验室': 21, '矿物提取': 22, '一般工业': 23, '制造加工': 24, '仓储配送': 25, '不确定用地': 26, '贮存': 27, '铁路': 28, '高速公路': 29, '其他具有相关设施的线性交通': 30, '航空运输': 31, '独立停车场': 32, '通讯': 33, '公用事业通行权': 34, '废水处理设施': 35, '垃圾填埋场': 36, '其它公用设施废物': 37, '雨水管理': 38, '联运设施': 39, '农业': 40, '开放空间主要娱乐设施': 41, '高尔夫球场': 42, '开放空间主要保护区': 43, '非公共开放空间': 44, '步道或绿道': 45, '空置住宅用地': 46, '空置商业用地': 47, '空置工业用地': 48, '其它空置': 49, '在建住宅': 50, '在建商业': 51, '在建工业': 52, '其他或未知在建': 53, '水体': 54, '非包裹区域': 55, '未分类': 56}}

将土地利用类型编码映射字典写入到PostgreSQL数据库，先将嵌套字典转化为DataFrame格式数据，然后用pandas提供的to\_sql方法写入。定义df2postSQL()函数将DataFrame类型数据写入至数据库；定义postSQL2df()函数从数据库读取表为DataFrame格式数据。

def df2postSQL(df,table\_name,if\_exists='replace',\*\*kwargs):  
 '''  
 将DataFrame格式数据写入PostgreSQL数据库  
  
 Parameters  
 ----------  
 df : DataFrame  
 DataFrame格式数据.  
 table\_name : string  
 写入数据库中的表名.  
 if\_exists : string, optional  
 包括{‘fail’, ‘replace’, ‘append’}. The default is 'replace'.  
 \*\*kwargs : kwargs  
 接数据库相关信息，包括myusername（数据库的用户名），mypassword（用户密钥），mydatabase（数据库名）.  
  
 Returns  
 -------  
 None.  
  
 '''   
 from sqlalchemy import create\_engine  
 # The URI should start with postgresql:// instead of postgres://. SQLAlchemy used to accept both, but has removed support for the postgres name.  
 engine=create\_engine("postgresql://{myusername}:{mypassword}@localhost:5432/{mydatabase}".format(myusername=kwargs['myusername'],mypassword=kwargs['mypassword'],mydatabase=kwargs['mydatabase']))   
 conn=engine.connect()  
 df.to\_sql(table\_name, con=conn, if\_exists=if\_exists,index=False)   
 # gdf.to\_postgis(table\_name, con=engine, if\_exists='replace', index=False,)   
 print("\_"\*50)   
 print('The GeoDataFrame has been written to the PostgreSQL database.The table name is {}.'.format(table\_name))

import pandas as pd  
  
landuse\_LB\_uncropped\_mapping\_reverse={k:{v\_:k\_ for k\_,v\_ in v.items()} for k,v in landuse\_LB\_uncropped\_mapping.items()}  
landuse\_LB\_uncropped\_mapping\_df=pd.DataFrame.from\_dict(landuse\_LB\_uncropped\_mapping\_reverse)  
landuse\_LB\_uncropped\_mapping\_df.reset\_index(names=['label\_encoder'],inplace=True)  
df2postSQL(landuse\_LB\_uncropped\_mapping\_df,'landuse\_lb',\*\*args.db.db\_info)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is landuse\_lb.

def postSQL2df(table\_name,\*\*kwargs):   
 '''  
 读取PostgreSQL数据库中的表为DataFrame格式数据  
  
 Parameters  
 ----------  
 table\_name : string  
 待读取数据库中的表名.  
 \*\*kwargs : kwargs  
 连接数据库相关信息，包括myusername（数据库的用户名），mypassword（用户密钥），mydatabase（数据库名）.  
  
 Returns  
 -------  
 df : DataFrame  
 从PostgreSQL中读取数据为DataFrame格式返回.  
   
 '''  
 from sqlalchemy import create\_engine  
 import pandas as pd   
   
 engine=create\_engine("postgresql://{myusername}:{mypassword}@localhost:5432/{mydatabase}".format(myusername=kwargs['myusername'],mypassword=kwargs['mypassword'],mydatabase=kwargs['mydatabase']))   
 conn=engine.connect()  
 df=pd.read\_sql('SELECT \* FROM {}'.format(table\_name), conn)  
  
 print("\_"\*50)  
 print('The data has been read from PostgreSQL database. The table name is {}.'.format(table\_name))   
 return df

landuse\_LB\_uncropped\_mapping\_df=postSQL2df('landuse\_lb',\*\*args.db.db\_info)  
landuse\_LB\_uncropped\_mapping\_df.head()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is landuse\_lb.

|  | **label\_encoder** | **LANDUSE** | **CHname** |
| --- | --- | --- | --- |
| **0** | 0 | 1111 | 独栋单户住宅 |
| **1** | 1 | 1112 | 连排单户住宅 |
| **2** | 2 | 1130 | 多户住宅 |
| **3** | 3 | 1140 | 托车移动住宅 |
| **4** | 4 | 1151 | 住宅开发区内公共开放空间 |

#### 2）矢量SHP格式土地利用数据转栅格（raster）

矢量转栅格函数定义时是读入SHP格式文件，转换后直接将栅格写入到指定路径下，因此将预处理后的的GeoDataFrame格式土地利用数据先写入到本地指定路径磁盘中，存储为SHP格式文件。可以直接查看(Geo)DataFrame数据的列名，该列名就为SHP格式数据表（table）的字段（field）名，但是如果希望直接获取给定路径SHP文件的字段名，则定义vectorLayer\_attributeNames()函数，可以直接返回字段名查看，方便矢量转栅格时，确定转换的字段。

landuse\_LB\_uncropped\_gdf.to\_file(args.data.landuse\_LB)

C:\Users\richi\AppData\Local\Temp\ipykernel\_39104\2380882267.py:1: UserWarning: Column names longer than 10 characters will be truncated when saved to ESRI Shapefile.  
 landuse\_LB\_uncropped\_gdf.to\_file(args.data.landuse\_LB)

def vectorLayer\_attributeNames(shp\_fn):  
 '''  
 返回SHP文件的字段（属性）名  
  
 Parameters  
 ----------  
 shp\_fn : string  
 shp格式文件路径.  
  
 Returns  
 -------  
 final\_attribs : list(string)  
 包含的字段名.  
  
 '''  
 from osgeo import ogr  
   
 source\_ds=ogr.Open(shp\_fn)  
 lyr=source\_ds.GetLayer()  
 all\_attribs=[attrib.name for attrib in lyr.schema]  
 final\_attribs=[attribs for attribs in all\_attribs if attribs not in ['list of attributes to exclude']]  
   
 return final\_attribs

landuse\_attribs=vectorLayer\_attributeNames(args.data.landuse\_LB)  
landuse\_attribs

['FIRST\_COUN',  
 'LANDUSE',  
 'LANDUSE2',  
 'OS\_MGMT',  
 'FAC\_NAME',  
 'PLATTED',  
 'MODIFIER',  
 'Shape\_Leng',  
 'Shape\_Area',  
 'CHname',  
 'parentClas',  
 'LB\_LANDUSE',  
 'LB\_CHname']

矢量转栅格主要调用GDAL、OGR底层库（相对GeoPandas、rasterio等库）提供的方法，主程序为create\_multiband\_raster()函数，可为参数attribs提供一个到多个字段名，对应转换为单波段或多波段栅格。然后逐字段的进行计算，先调用rasterize()函数将SHP单个字段转换为单波段栅格数据，再用img\_to\_array()函数提取单波段栅格数据的值，并将值追加到一个空列表img\_array\_list中，当循环完所有字段，将img\_array\_list列表转换为数组，最后调用array\_to\_tif()函数转化为单波段（如果只给了一个字段）或多波段（如果给了多个字段）栅格，并存储到指定文件路径下（文件名通常以.tif为后缀）。

def rasterize(shp, attrib\_name,cellSize=500,dtype='int32'):  
 '''  
 转换单个vector（polygon或points）shp格式数据为栅格数据  
  
 Parameters  
 ----------  
 shp : String  
 shp格式对象文件路径.  
 attrib\_name : String  
 字段（属性）名.  
 cellSize : numerical, optional  
 栅格单元大小. The default is 500.  
 dtype : String, optional  
 数据类型，对应dtype\_mapping = {'byte': gdal.GDT\_Byte, 'uint8': gdal.GDT\_Byte, 'uint16': gdal.GDT\_UInt16, 'int8': gdal.GDT\_Byte, 'int16': gdal.GDT\_Int16, 'int32': gdal.GDT\_Int32, 'uint32': gdal.GDT\_UInt32, 'float32': gdal.GDT\_Float32}. The default is 'int32'.  
  
 Returns  
 -------  
 temp\_out : TIFF  
 栅格临时文件.  
  
 '''  
 import gdal  
 import gdalnumeric as gdn  
 import ogr  
 import numpy as np  
 import tempfile   
   
 # 定义空值（没有数据）的栅格数值 Define NoData value of new raster  
 NoData\_value=-9999  
   
 # 打开.shp点数据，并返回地理区域范围 Open the data source and read in the extent  
 source\_ds=ogr.Open(shp)  
 source\_layer=source\_ds.GetLayer()  
 x\_min, x\_max, y\_min, y\_max=source\_layer.GetExtent()  
   
 # 使用GDAL库建立栅格 Create the destination data source  
 x\_res=int((x\_max - x\_min) / cellSize)  
 y\_res=int((y\_max - y\_min) / cellSize)   
   
 # create empty raster (.tif) as temporary file and set its projection and extent to  
 # that of the reference raster  
 temp\_out = tempfile.NamedTemporaryFile(suffix='.tif').name  
 memory\_driver = gdal.GetDriverByName('GTiff')  
 dtype\_mapping = {'byte': gdal.GDT\_Byte, 'uint8': gdal.GDT\_Byte, 'uint16': gdal.GDT\_UInt16, 'int8': gdal.GDT\_Byte, 'int16': gdal.GDT\_Int16, 'int32': gdal.GDT\_Int32, 'uint32': gdal.GDT\_UInt32, 'float32': gdal.GDT\_Float32}  
 out\_raster\_ds = memory\_driver.Create(temp\_out, x\_res, y\_res, 1,dtype\_mapping[dtype]) # gdal.GDT\_Float64;gdal.GDT\_Byte  
 out\_raster\_ds.SetGeoTransform((x\_min, cellSize, 0, y\_max, 0, -cellSize))  
 outband=out\_raster\_ds.GetRasterBand(1)  
 outband.SetNoDataValue(NoData\_value)  
  
 # open shapefile vector layer to retrieve and burn attribute into the empty raster  
 gdal.RasterizeLayer(out\_raster\_ds, [1], source\_layer, options=["ATTRIBUTE="+attrib\_name])  
 return temp\_out   
  
def img\_to\_array(input\_file, dim\_ordering="channel\_last", dtype="float32"):  
 '''  
 将栅格文件各层栅格值转换为（numpy）数组  
  
 Parameters  
 ----------  
 input\_file : TIFF  
 栅格文件.  
 dim\_ordering : String, optional  
 调整波段位置. The default is "channel\_last".  
 dtype : Strng, optional  
 栅格存储数据类型. The default is "float32".  
  
 Returns  
 -------  
 arr : numpy.ndarray  
 各层的栅格单元值.  
  
 '''  
 import gdal  
 import numpy as np  
 import gdalnumeric as gdn  
 # open input raster, retrieve bands and convert to image array  
 file = gdal.Open(input\_file)  
 bands = [file.GetRasterBand(i) for i in range(1, file.RasterCount +1)]  
 arr = np.array([gdn.BandReadAsArray(band) for band in bands]).astype(dtype)  
   
 # reoder dimensions so that channels/bands are last  
 if dim\_ordering=="channel\_last": arr = np.transpose(arr, [1,2,0])  
 return arr   
  
def array\_to\_tif(array,inVector,dst\_filename,cellSize=500):  
 '''  
 将数组（含多个波段值）写入栅格文件  
  
 Parameters  
 ----------  
 array : numpy.ndarray  
 数组形式各层的栅格单元值.  
 inVector : String  
 shp格式对象文件路径.  
 dst\_filename : String  
 待保存的栅格文件路径.  
 cellSize : numerical, optional  
 栅格单元大小. The default is 500.  
  
 Returns  
 -------  
 None.  
  
 '''  
 import gdal  
 import ogr  
 import os  
  
 # if os.path.exists(dst\_filename):  
 # os.remove(dst\_filename)  
 # print("The file has been deleted successfully")  
 # else:  
 # print("The file does not exist!")   
   
 NoData\_value=-9999  
 source\_ds=ogr.Open(inVector)  
 source\_layer=source\_ds.GetLayer()  
 x\_min, x\_max, y\_min, y\_max=source\_layer.GetExtent()  
 x\_res=int((x\_max - x\_min) / cellSize)  
 y\_res=int((y\_max - y\_min) / cellSize)   
 proj=source\_layer.GetSpatialRef().ExportToWkt()  
   
 # create empty raster (.tif) to which array will be written  
 bands = array.shape[2]  
 dtype = str(array.dtype)  
 dtype\_mapping = {'byte': gdal.GDT\_Byte, 'uint8': gdal.GDT\_Byte, 'uint16': gdal.GDT\_UInt16, 'int8': gdal.GDT\_Byte, 'int16': gdal.GDT\_Int16, 'int32': gdal.GDT\_Int32, 'uint32': gdal.GDT\_UInt32, 'float32': gdal.GDT\_Float32}  
 driver = gdal.GetDriverByName('GTiff')  
 output = driver.Create(dst\_filename, x\_res, y\_res, bands, dtype\_mapping[dtype])  
 # set output image extent and projection  
 output.SetGeoTransform((x\_min, cellSize, 0, y\_max, 0, -cellSize))  
 output.SetProjection(proj)  
   
 # write image array into empty raster  
 for i in range(bands): output.GetRasterBand(i+1).WriteArray(array[:, :, i])  
 output.FlushCache()   
  
def create\_multiband\_raster(attribs,inVector,dst\_filename,cellSize=500,dtype='int32'):  
 '''  
 vector(SHP,.shp)格式文件转栅格，主程序  
 ref\_Rasterize: How to create multiband raster from vector attributes using python https://tkawuah.github.io/Blog1.html  
  
 Parameters  
 ----------  
 attribs : List(String)  
 待存储的.shp属性值列表.  
 inVector : .shp  
 .shp格式文件路径.  
 dst\_filename : Stirng-TIFF(.tiff)  
 栅格保存路径名，通常以.tif为后缀名.  
 cellSize : numercial, optional  
 栅格单元大小. The default is 500.  
 dtype : String, optional  
 栅格单元存储数据的类型。dtype\_mapping = {'byte': gdal.GDT\_Byte, 'uint8': gdal.GDT\_Byte, 'uint16': gdal.GDT\_UInt16, 'int8': gdal.GDT\_Byte, 'int16': gdal.GDT\_Int16, 'int32': gdal.GDT\_Int32, 'uint32': gdal.GDT\_UInt32, 'float32': gdal.GDT\_Float32}. The default is 'int32'.  
  
 Returns  
 -------  
 None.  
  
 '''  
 import numpy as np  
   
 img\_array\_list=[]  
 for i in attribs:   
 fx=rasterize(inVector, i,cellSize=cellSize,dtype=dtype)  
 fx\_array=img\_to\_array(fx,dtype=dtype)  
 img\_array\_list.append(fx\_array)   
  
 fx\_multi=np.concatenate(img\_array\_list, axis=-1)  
 array\_to\_tif(fx\_multi,inVector,dst\_filename,cellSize=cellSize)   
 print('The raster was written successfully!')

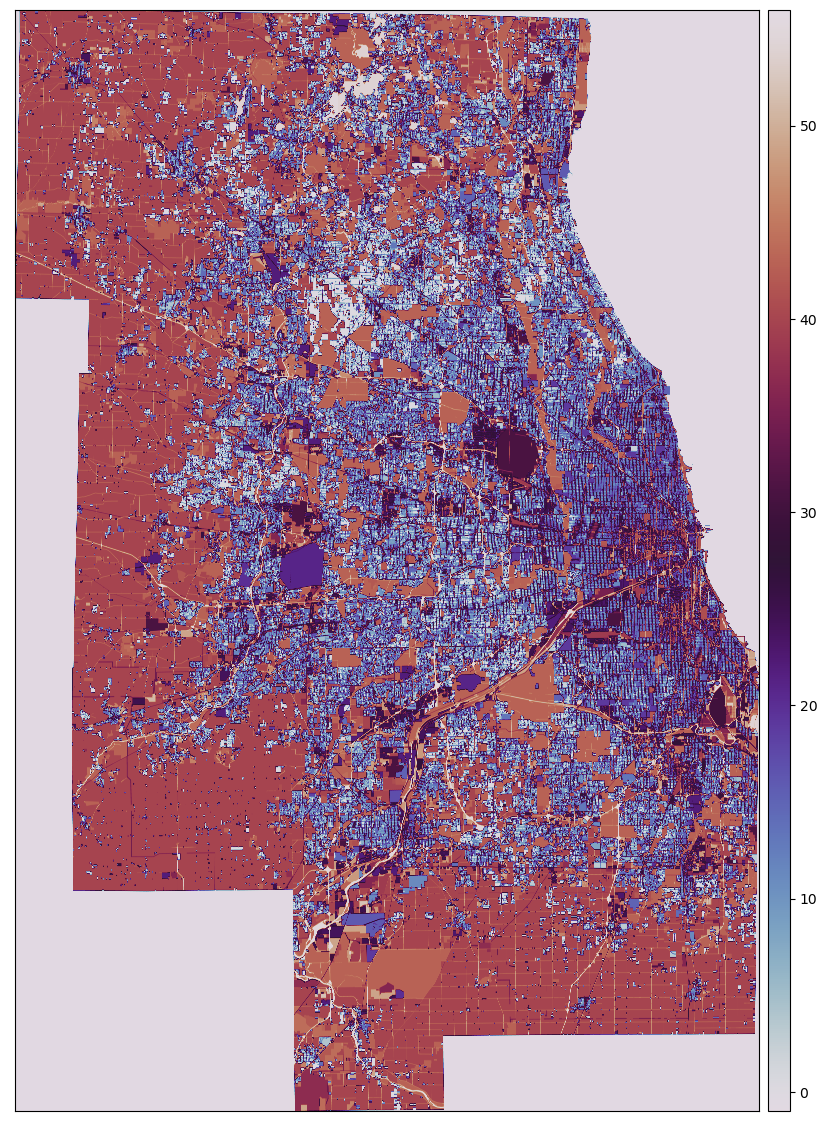
landuse\_attribs\_selection=['lB\_LANDUSE']  
create\_multiband\_raster(landuse\_attribs\_selection,args.data.landuse\_LB,args.data.landuse\_tif,cellSize=5)

The raster was written successfully!

用rasterio库打开栅格文件，用EarthPy库的earthpy.plot.plot\_bands方法打印单波段地图。

import rasterio as rio  
with rio.open(args.data.landuse\_tif) as src:   
 landuse\_array=src.read()

import earthpy.plot as ep  
import copy  
landuse\_array\_copy=copy.deepcopy(landuse\_array,)  
landuse\_array\_copy[landuse\_array\_copy==-9999]=-1  
ep.plot\_bands(landuse\_array\_copy,figsize=(10, 15),cmap='twilight');



#### 3）土地利用地图打印

在2.7.4一节解释MCD12Q1\_v006 土地覆盖类型数据集时，通过cmap\_patch\_build()函数构建颜色的cmap、norm和patches变量，按栅格单元值对应给定的颜色，打印栅格地图。本次则直接打印GeoDataFrame格式地图数据，且将二级土地利用分类按照一级分类分配的色系赋予颜色。定义colorsGrouped\_cmapBased()函数用于生成分组的颜色字典。

landuse\_parentClass\_mapping\_2=landuse\_mapping\_df.landuse\_parentClass\_mapping\_2.to\_dict()  
  
landuse\_parentClass\_mapping\_inverse={}  
for k, v in landuse\_parentClass\_mapping\_2.items():  
 landuse\_parentClass\_mapping\_inverse[v]=landuse\_parentClass\_mapping\_inverse.get(v, [])+[k]   
print(landuse\_parentClass\_mapping\_inverse)

{'城市化区域': ['1000'], '住宅': ['1110', '1111', '1112', '1130', '1140', '1150', '1151'], '商业': ['1200', '1210', '1211', '1212', '1214', '1215', '1216', '1220', '1240', '1250'], '机构': ['1300', '1310', '1320', '1321', '1322', '1330', '1340', '1350', '1360', '1370', '1380'], '工业': ['1400', '1410', '1420', '1430', '1431', '1432', '1433', '1450'], '交通\_通讯\_公共事业和垃圾处理': ['1500', '1510', '1511', '1512', '1520', '1530', '1540', '1550', '1560', '1561', '1562', '1563', '1564', '1565', '1570'], '其它': ['2000', '4000', '4100', '4110', '4120', '4130', '4140', '4200', '4210', '4220', '4230', '4240', '9999'], '开放空间': ['3000', '3100', '3200', '3300', '3400', '3500'], '水体': ['5000'], '非包裹区域': ['6000']}

def colorsGrouped\_cmapBased(palettes,group\_dict):  
 '''  
 指定分组的cmap色系，根据分组字典生成分组颜色  
  
 Parameters  
 ----------  
 palettes : dict  
 指定分组色系，例如{'住宅':"YlOrBr",   
 商业':"Reds"}.  
 group\_dict : dict  
 指定分组字典，例如{'住宅': ['1110', '1111', '1112', '1130', '1140', '1150', '1151'],   
 '商业': ['1200', '1210', '1211', '1212', '1214', '1215', '1216', '1220', '1240', '1250']}.  
  
 Returns  
 -------  
 L2\_pal\_hex\_dict : dict  
 分组成员16进制颜色列表.  
  
 '''  
 import seaborn as sns  
   
 L2\_pal\_hex\_dict={}  
 for L1,pal in palettes.items():  
 L2=group\_dict[L1]  
 pal=sns.color\_palette(palettes[L1],len(L2))  
 pal\_hex=pal.as\_hex()  
 L1\_L2\_pal\_hex=dict(zip(L2,pal\_hex))  
 L2\_pal\_hex\_dict.update(L1\_L2\_pal\_hex)   
   
 return L2\_pal\_hex\_dict   
   
palettes={'住宅':"YlOrBr",   
 '商业':"Reds",  
 '机构':"GnBu",   
 '工业':"Purples",   
 '交通\_通讯\_公共事业和垃圾处理':"Greys",  
 '其它':"PuBuGn",   
 '开放空间':"YlGn",   
 '非包裹区域':"gray",  
 '水体':'Blues'}  
  
L2\_pal\_hex\_dict=colorsGrouped\_cmapBased(palettes,landuse\_parentClass\_mapping\_inverse)  
print(L2\_pal\_hex\_dict)

{'1110': '#fff7bc', '1111': '#fee390', '1112': '#fec34f', '1130': '#fe9829', '1140': '#eb6f14', '1150': '#cb4b02', '1151': '#983404', '1200': '#fee6da', '1210': '#fdd0bc', '1211': '#fcb499', '1212': '#fc9576', '1214': '#fb7858', '1215': '#f7593f', '1216': '#ec382b', '1220': '#d11e1f', '1240': '#b61319', '1250': '#940b13', '1300': '#e8f6e2', '1310': '#daf0d4', '1320': '#ccebc5', '1321': '#b4e2ba', '1322': '#99d7ba', '1330': '#7accc4', '1340': '#5dbbce', '1350': '#42a6cc', '1360': '#2a8bbe', '1370': '#1373b2', '1380': '#085a9d', '1400': '#f1eff6', '1410': '#dfdfed', '1420': '#c6c7e1', '1430': '#acaad1', '1431': '#908dc2', '1432': '#796eb2', '1433': '#65479e', '1450': '#51228d', '1500': '#f7f7f7', '1510': '#f0f0f0', '1511': '#e4e4e4', '1512': '#d9d9d9', '1520': '#cbcbcb', '1530': '#bdbdbd', '1540': '#a9a9a9', '1550': '#959595', '1560': '#848484', '1561': '#727272', '1562': '#626262', '1563': '#515151', '1564': '#3a3a3a', '1565': '#242424', '1570': '#111111', '2000': '#f4ebf5', '4000': '#e8e0ef', '4100': '#d9d6e9', '4110': '#c4cbe3', '4120': '#acc0dd', '4130': '#8cb5d6', '4140': '#66a9cf', '4200': '#4b9ac6', '4210': '#2e8eb8', '4220': '#11859a', '4230': '#027b7b', '4240': '#016f5f', '9999': '#015b4a', '3000': '#f3fab6', '3100': '#ccea9d', '3200': '#97d385', '3300': '#58b669', '3400': '#2c8f4b', '3500': '#056c39', '6000': '#808080', '5000': '#6aaed6'}

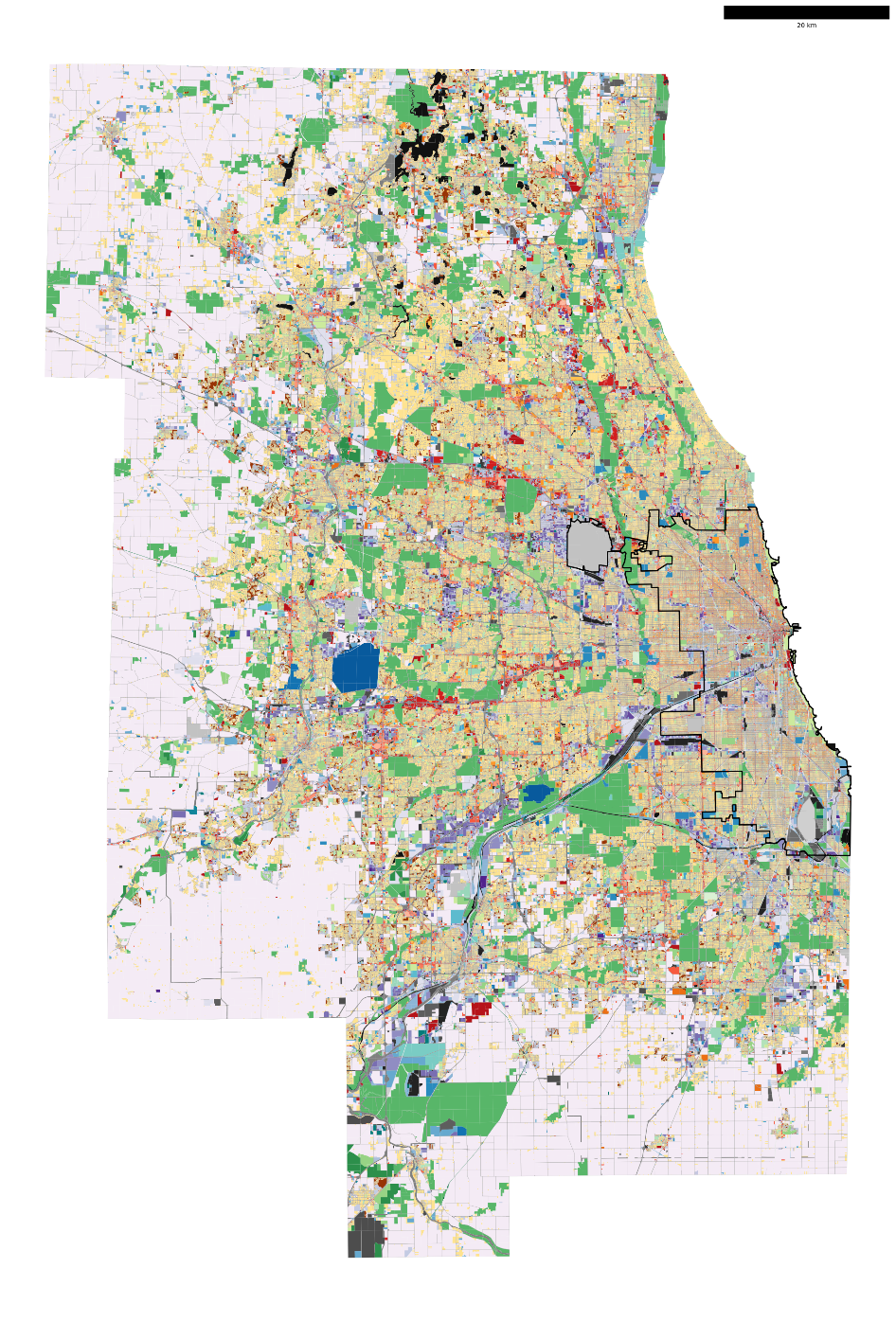
为了定位芝加哥城位置，读取芝加哥城市边界，该数据来源于[Chicago Data Portal，CDP](https://data.cityofchicago.org/)③。将分组颜色字典L2\_pal\_hex\_dict按照土地利用类型编号追加到lu4plot列中，用GeoDataFrame.plot方法直接打印地图。

import geopandas as gpd  
Chicago\_boundary\_gdf=gpd.read\_file(args.data.Chicago\_boundaries\_city)  
Chicago\_boundary\_gdf.to\_crs(lu4plot.crs,inplace=True)

lu4plot=landuse\_CH\_uncropped\_gdf.copy(deep=True)

lu4plot['colors']=lu4plot.LANDUSE.apply(lambda row:L2\_pal\_hex\_dict[row])

from matplotlib\_scalebar.scalebar import ScaleBar  
import matplotlib.pyplot as plt  
  
ax=lu4plot.plot(color=lu4plot['colors'],figsize=(20, 50),legend=True)  
Chicago\_boundary\_gdf.plot(color='none',edgecolor='k',linewidth=2,ax=ax,linestyle='-')  
ax.add\_artist(ScaleBar(1))   
ax.set\_axis\_off()  
plt.tight\_layout()  
plt.show()



上述地图并未打印图例，为了方便图例的排布，定义plot\_colortable\_updated()函数单独打印图例。该函数迁移更新于matplotlib库提供的案例，为了方便参数调整，将主要的参数作为关键字参数传入更新。

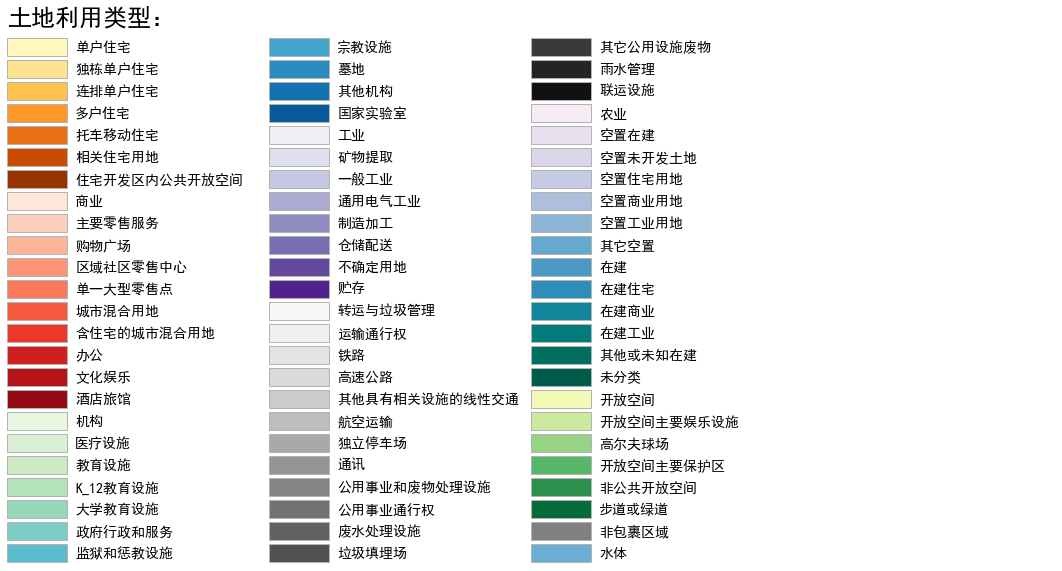
def plot\_colortable\_updated(colors,sort\_colors=True,emptycols=0,\*\*args):  
 '''  
 绘制一个matplotlib支持命名颜色的列表，迁移更新于matplotlib库“List of named colors”，https://matplotlib.org/3.1.1/gallery/color/named\_colors.html  
 查看2.7.4章 plot\_colortable()函数，为该函数的升级版本  
  
 Parameters  
 ----------  
 colors : dict  
 键为图例名，值为16进制颜色.  
 sort\_colors : bool, optional  
 是否排序颜色. The default is True.  
 emptycols : int, optional  
 调整列数量，避免列中出现空值. The default is 0.  
 \*\*args : kargs  
 图表打印参数调整，默认为：  
 setting=dict(  
 cell\_width = 212,  
 cell\_height = 22,  
 swatch\_width = 48,  
 margin = 12,  
 topmargin = 40,  
 title='Base Colors',  
 fig\_width\_fine\_tuning=0,  
 fig\_height\_fine\_tuning=0,  
 ).  
  
 Returns  
 -------  
 fig : matplotlib.figure.Figure  
 matplotlib图表.  
   
 '''  
 from matplotlib.patches import Rectangle  
 import matplotlib.pyplot as plt  
 import matplotlib.colors as mcolors  
 import matplotlib as mpl   
 import numpy as np  
  
 mpl.rcParams['font.sans-serif'] = ['SimHei']  
 mpl.rcParams['font.serif'] = ['SimHei']   
   
 setting=dict(  
 cell\_width = 212,  
 cell\_height = 22,  
 swatch\_width = 48,  
 margin = 12,  
 topmargin = 40,  
 title='Base Colors',  
 fig\_width\_fine\_tuning=0,  
 fig\_height\_fine\_tuning=0,  
 )  
  
 setting.update(args)  
 cell\_width=setting['cell\_width']  
 cell\_height=setting['cell\_height']  
 swatch\_width=setting['swatch\_width']  
 margin=setting['margin']  
 topmargin=setting['topmargin']   
 fig\_width\_fine\_tuning=setting['fig\_width\_fine\_tuning']  
 fig\_height\_fine\_tuning=setting['fig\_height\_fine\_tuning']  
   
 # Sort colors by hue, saturation, value and name.  
 if sort\_colors is True:  
 by\_hsv = sorted((tuple(mcolors.rgb\_to\_hsv(mcolors.to\_rgb(color))),  
 name)  
 for name, color in colors.items())  
 names = [name for hsv, name in by\_hsv]  
 else:  
 names = list(colors)  
  
 n = len(names)  
 ncols = 4 - emptycols  
 nrows = n // ncols + int(n % ncols > 0)  
  
 width = cell\_width \* 4 + 2 \* margin+fig\_width\_fine\_tuning  
 height = cell\_height \* nrows + margin + topmargin+fig\_height\_fine\_tuning  
 dpi = 72  
  
 fig, ax = plt.subplots(figsize=(width / dpi, height / dpi), dpi=dpi)  
 fig.subplots\_adjust(margin/width, margin/height,  
 (width-margin)/width, (height-topmargin)/height)  
 ax.set\_xlim(0, cell\_width \* 4)  
 ax.set\_ylim(cell\_height \* (nrows-0.5), -cell\_height/2.)  
 ax.yaxis.set\_visible(False)  
 ax.xaxis.set\_visible(False)  
 ax.set\_axis\_off()  
 ax.set\_title(setting['title'], fontsize=24, loc="left", pad=10)  
  
 for i, name in enumerate(names):  
 row = i % nrows  
 col = i // nrows  
 y = row \* cell\_height  
  
 swatch\_start\_x = cell\_width \* col  
 text\_pos\_x = cell\_width \* col + swatch\_width + 7  
  
 ax.text(text\_pos\_x, y, name, fontsize=14,  
 horizontalalignment='left',  
 verticalalignment='center')  
 ax.add\_patch(  
 Rectangle(xy=(swatch\_start\_x, y-9), width=swatch\_width,  
 height=18, facecolor=colors[name], edgecolor='0.7')  
 )  
  
 return fig

plot\_colortable\_updated()函数的colors参数为一个字典，键为图例名，值为图例的16进制颜色。

L2\_pal\_hex\_CH\_dict={landuse\_EN2CH\_mapping[landuse\_EN\_mapping[k]]:v for k,v in L2\_pal\_hex\_dict.items()}  
print(L2\_pal\_hex\_CH\_dict)

{'单户住宅': '#fff7bc', '独栋单户住宅': '#fee390', '连排单户住宅': '#fec34f', '多户住宅': '#fe9829', '托车移动住宅': '#eb6f14', '相关住宅用地': '#cb4b02', '住宅开发区内公共开放空间': '#983404', '商业': '#fee6da', '主要零售服务': '#fdd0bc', '购物广场': '#fcb499', '区域社区零售中心': '#fc9576', '单一大型零售点': '#fb7858', '城市混合用地': '#f7593f', '含住宅的城市混合用地': '#ec382b', '办公': '#d11e1f', '文化娱乐': '#b61319', '酒店旅馆': '#940b13', '机构': '#e8f6e2', '医疗设施': '#daf0d4', '教育设施': '#ccebc5', 'K\_12教育设施': '#b4e2ba', '大学教育设施': '#99d7ba', '政府行政和服务': '#7accc4', '监狱和惩教设施': '#5dbbce', '宗教设施': '#42a6cc', '墓地': '#2a8bbe', '其他机构': '#1373b2', '国家实验室': '#085a9d', '工业': '#f1eff6', '矿物提取': '#dfdfed', '一般工业': '#c6c7e1', '通用电气工业': '#acaad1', '制造加工': '#908dc2', '仓储配送': '#796eb2', '不确定用地': '#65479e', '贮存': '#51228d', '转运与垃圾管理': '#f7f7f7', '运输通行权': '#f0f0f0', '铁路': '#e4e4e4', '高速公路': '#d9d9d9', '其他具有相关设施的线性交通': '#cbcbcb', '航空运输': '#bdbdbd', '独立停车场': '#a9a9a9', '通讯': '#959595', '公用事业和废物处理设施': '#848484', '公用事业通行权': '#727272', '废水处理设施': '#626262', '垃圾填埋场': '#515151', '其它公用设施废物': '#3a3a3a', '雨水管理': '#242424', '联运设施': '#111111', '农业': '#f4ebf5', '空置在建': '#e8e0ef', '空置未开发土地': '#d9d6e9', '空置住宅用地': '#c4cbe3', '空置商业用地': '#acc0dd', '空置工业用地': '#8cb5d6', '其它空置': '#66a9cf', '在建': '#4b9ac6', '在建住宅': '#2e8eb8', '在建商业': '#11859a', '在建工业': '#027b7b', '其他或未知在建': '#016f5f', '未分类': '#015b4a', '开放空间': '#f3fab6', '开放空间主要娱乐设施': '#ccea9d', '高尔夫球场': '#97d385', '开放空间主要保护区': '#58b669', '非公共开放空间': '#2c8f4b', '步道或绿道': '#056c39', '非包裹区域': '#808080', '水体': '#6aaed6'}

fig=plot\_colortable\_updated(L2\_pal\_hex\_CH\_dict,sort\_colors=False, emptycols=1,title='土地利用类型：',fig\_width\_fine\_tuning=200)

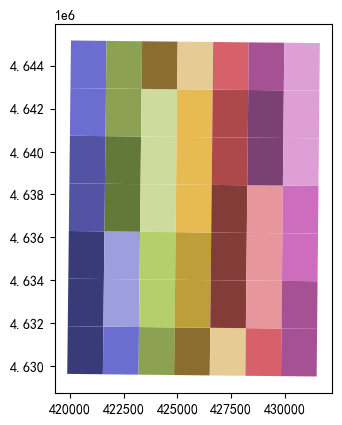


#### 4）样方（区域）统计

样方统计实现的方式是由给定的样方（Polygon对象）对栅格数据采样，因此需要首先构建用于采样的样方，定义rec\_quadrats\_gdf()函数实现网格式样方GeoDataFrame格式数据的构建。在QGIS平台或者在Google Earth下确定待分析区域的左下角和右上角坐标。投影坐标系使用wgs84，然后再根据需要转换为对应栅格的坐标系用于样方统计。后续分析需要不同样方组之间进行相关指数比较，因此定义两组样方，分别标识为a和b。

def rec\_quadrats\_gdf(leftBottom\_coordi,rightTop\_coordi,h\_distance,v\_distance,crs=4326,to\_crs=None):  
 '''  
 构建网格式样方  
  
 Parameters  
 ----------  
 leftBottom\_coordi : list(float)  
 定位左下角坐标.  
 rightTop\_coordi : list(float)  
 定位右上角坐标.  
 h\_distance : float  
 单个样方宽度.  
 v\_distance : float  
 单个样方长度.  
 crs : int, optional  
 投影编号. The default is 4326.  
 to\_crs : int, optional  
 转换投影编号. The default is None.  
  
 Returns  
 -------  
 grids\_gdf : GeoDataFrame  
 Polygon地理几何形式的GeoDataFrame格式样方数据.  
  
 '''   
 import numpy as np  
 from shapely.geometry import MultiLineString  
 from shapely.ops import polygonize  
 import geopandas as gpd  
   
 x=np.arange(leftBottom\_coordi[0], rightTop\_coordi[0], h\_distance)  
 y=np.arange(leftBottom\_coordi[1], rightTop\_coordi[1], v\_distance)  
 hlines=[((x1, yi), (x2, yi)) for x1, x2 in zip(x[:-1], x[1:]) for yi in y]  
 vlines=[((xi, y1), (xi, y2)) for y1, y2 in zip(y[:-1], y[1:]) for xi in x]  
 grids=list(polygonize(MultiLineString(hlines + vlines)))  
   
 grids\_gdf=gpd.GeoDataFrame({'geometry':grids},crs=crs)  
 if to\_crs:  
 grids\_gdf.to\_crs(to\_crs,inplace=True)  
   
 return grids\_gdf  
  
cell\_size=0.02  
pt\_leftBottom\_a=[-87.96487621023101,41.81454769756404]  
pt\_rightTop\_a=[-87.81441541486787,41.96133871743053]   
quadrats\_a\_gdf=rec\_quadrats\_gdf(pt\_leftBottom\_a,pt\_rightTop\_a,cell\_size,cell\_size,args.gi.epsg\_wgs84,args.gi.Chicago\_epsg)  
gpd2postSQL(quadrats\_a\_gdf,'quadrats\_a',\*\*args.db.db\_info)   
  
quadrats\_a\_gdf.plot(cmap='tab20b')

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is quadrats\_a.  
  
  
  
  
  
<AxesSubplot: >



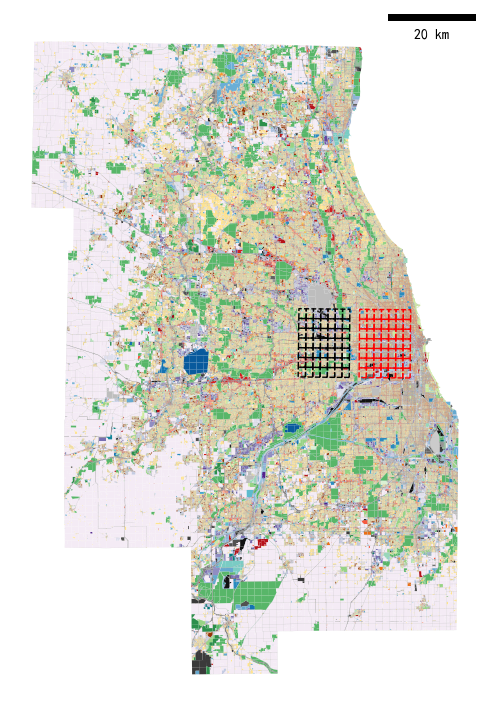
建立b组样方。

pt\_leftBottom\_b=[-87.79790142513288,41.81454769756404]  
pt\_rightTop\_b=[-87.64744062976973,41.96133871743053]   
quadrats\_b\_gdf=rec\_quadrats\_gdf(pt\_leftBottom\_b,pt\_rightTop\_b,cell\_size,cell\_size,args.gi.epsg\_wgs84,args.gi.Chicago\_epsg)  
gpd2postSQL(quadrats\_b\_gdf,'quadrats\_b',\*\*args.db.db\_info)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is quadrats\_b.

为了核实构建的样方是否对位到栅格数据区域，叠加打印栅格数据和样方数据，从地图中确定构建的样方无误后再进行样方统计。

ax=lu4plot.plot(color=lu4plot['colors'],figsize=(20/4, 50/4),legend=True)  
quadrats\_a\_gdf.plot(color='none',edgecolor='k',linewidth=1,ax=ax,linestyle='--')  
quadrats\_b\_gdf.plot(color='none',edgecolor='r',linewidth=1,ax=ax,linestyle='--')  
ax.add\_artist(ScaleBar(1))   
ax.set\_axis\_off()  
plt.tight\_layout()  
plt.show()



样方统计采用rasterstats库提供的zonal\_stats方法实现，其统计量包括’count’单元数量, ’min’最小值, ’max’最大值, ’mean’均值, ’sum’单元值总和, ’std’标准差, ’median’中位数, ’majority’单元数量最多的分类, ’minority’单元数据最少的分类, ’unique’样方包含的分类数量, ’range’栅格单元数值区间, ’nodata’和’nan’为空值分类数的和。默认的zonal\_stats方法并未给出样方类别的频数统计（各个类别的栅格单元数量），因此使用其提供的add\_stats参数传入自定义的frequency()函数计算。

def zonal\_stats\_raster(raster\_fn,sampling\_zone,band=1,stats=['majority'],add\_stats=['frequency'],nodata=-9999):#  
 '''  
 区域统计，包括['count', 'min', 'max', 'mean', 'sum', 'std', 'median', 'majority', 'minority', 'unique', 'range', 'nodata', 'nan']，以及自定义的'frequency'，即频数统计  
  
 Parameters  
 ----------  
 raster\_fn : String  
 待区域统计的栅格数据路径名.  
 sampling\_zone : GeoDataFrame  
 用于栅格区域统计的polygon几何对象.  
 band : int, optional  
 数据波段. The default is 1.  
 stats : List(String), optional  
 默认统计的统计量名称. The default is ['majority'].  
 add\_stats :List(String) , optional  
 自定义统计量名. The default is ['frequency'].  
  
 Returns  
 -------  
 GeoDataFrame  
 返回统计量值.  
  
 '''  
 import rasterio as rio  
 import rasterstats as rst  
 import pandas as pd  
   
 sampling\_zone\_copy=sampling\_zone.copy(deep=True)  
   
 def frequency(x):  
 data=x.data[~x.mask]  
 return pd.value\_counts(data)  
   
 add\_stats\_dict={'frequency':frequency}  
 with rio.open(raster\_fn,'r') as src:  
 band=src.read(band)  
 sampling\_zone\_copy=sampling\_zone\_copy.to\_crs(src.crs)  
 zs\_result=rst.zonal\_stats(sampling\_zone\_copy,band,nodata=nodata,affine=src.transform,stats=stats,add\_stats={i:add\_stats\_dict[i] for i in add\_stats})  
   
 for stat in stats:  
 sampling\_zone\_copy[stat]=[dic[stat] for dic in zs\_result]  
 for stat in add\_stats:  
 if stat=='frequency':  
 fre=pd.concat([dic[stat].to\_frame().T for dic in zs\_result])  
 fre.rename(columns={col:"{}\_{}".format(stat[:3],col) for col in fre.columns},inplace=True)  
 fre.reset\_index(inplace=True)   
 try:   
 zonal\_stats\_gdf=pd.concat([sampling\_zone\_copy,fre],axis=1)   
   
 except:  
 zonal\_stats\_gdf=sampling\_zone\_copy  
 return zonal\_stats\_gdf  
   
landuse\_zonal\_stats\_a\_gdf=zonal\_stats\_raster(args.data.landuse\_tif,quadrats\_a\_gdf,stats=['count'])

print(landuse\_zonal\_stats\_a\_gdf.columns.to\_list())  
landuse\_zonal\_stats\_a\_gdf.head()

['geometry', 'count', 'index', 'fre\_0', 'fre\_43', 'fre\_55', 'fre\_10', 'fre\_46', 'fre\_4', 'fre\_14', 'fre\_29', 'fre\_2', 'fre\_16', 'fre\_1', 'fre\_54', 'fre\_8', 'fre\_12', 'fre\_47', 'fre\_41', 'fre\_5', 'fre\_42', 'fre\_34', 'fre\_7', 'fre\_6', 'fre\_49', 'fre\_37', 'fre\_19', 'fre\_23', 'fre\_27', 'fre\_11', 'fre\_13', 'fre\_24', 'fre\_18', 'fre\_20', 'fre\_50', 'fre\_35', 'fre\_45', 'fre\_28', 'fre\_30', 'fre\_33', 'fre\_32', 'fre\_9', 'fre\_15', 'fre\_38', 'fre\_22', 'fre\_26', 'fre\_25', 'fre\_48', 'fre\_40', 'fre\_56', 'fre\_51', 'fre\_53', 'fre\_31', 'fre\_39', 'fre\_52', 'fre\_36', 'fre\_3', 'fre\_44']

|  | **geometry** | **count** | **index** | **fre\_0** | **fre\_43** | **fre\_55** | **fre\_10** | **fre\_46** | **fre\_4** | **fre\_14** | **...** | **fre\_40** | **fre\_56** | **fre\_51** | **fre\_53** | **fre\_31** | **fre\_39** | **fre\_52** | **fre\_36** | **fre\_3** | **fre\_44** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | POLYGON ((421520.398 4629617.466, 419859.219 4... | 147546 | 0 | 47103 | 23532.0 | 19612 | 18269.0 | 10250.0 | 7814.0 | 5845.0 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **1** | POLYGON ((421544.820 4631838.009, 419884.158 4... | 147500 | 0 | 18338 | NaN | 29150 | 27149.0 | 541.0 | 8627.0 | 1245.0 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **2** | POLYGON ((421569.252 4634058.559, 419909.107 4... | 147453 | 0 | 35973 | 6802.0 | 39109 | 5126.0 | 1261.0 | 4477.0 | 1366.0 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **3** | POLYGON ((421593.693 4636279.118, 419934.066 4... | 147408 | 0 | 72298 | 4878.0 | 32851 | 728.0 | 597.0 | 192.0 | 6550.0 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **4** | POLYGON ((421618.144 4638499.683, 419959.034 4... | 147363 | 0 | 74336 | 2010.0 | 29595 | 489.0 | 550.0 | NaN | 848.0 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

5 rows × 58 columns

为了方便查看DataFrame中一列值对应的另一类值，定义匿名函数colA2colB，例如，为了知道fre\_41列名对应的土地利用类型中文名是什么，传入landuse\_LB\_uncropped\_mapping\_df数据，两个列名，及编号列中的数值获得对应的分类名为开放空间主要娱乐设施。

colA2colB=lambda gdf,colA,colB,v:gdf[gdf[colA]==v][colB]  
colA2colB(landuse\_LB\_uncropped\_mapping\_df,'label\_encoder','CHname',41)

41 开放空间主要娱乐设施  
Name: CHname, dtype: object

不平等性和空间隔离分析主要分析开放空间在样方中的分布情况，因此按照’开放空间’的一级分类合并二级分类，包括’开放空间主要娱乐设施’、‘高尔夫球场’、‘开放空间主要保护区’、‘非公共开放空间’和’步道或绿道’。同时，保存至数据库方便调用；并打印统计样方，查看统计结果。

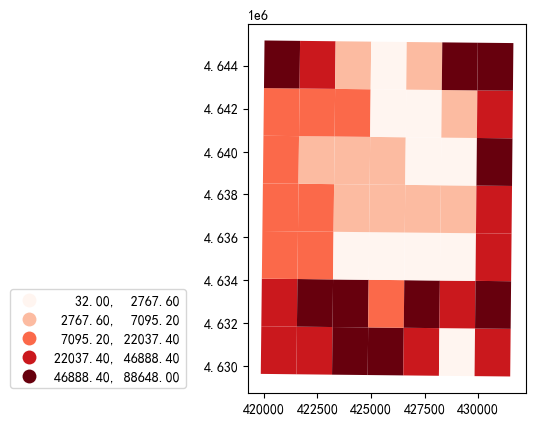
open\_space\_idxes=landuse\_parentClass\_mapping\_inverse['开放空间']   
open\_space\_lbs=[landuse\_LB\_uncropped\_mapping['LANDUSE'][i] for i in open\_space\_idxes if i in landuse\_LB\_uncropped\_mapping['LANDUSE'].keys()]  
open\_space\_cols=[f"fre\_{i}" for i in open\_space\_lbs]  
print(open\_space\_cols)

['fre\_41', 'fre\_42', 'fre\_43', 'fre\_44', 'fre\_45']

开放空间包含的二级分类有5类，但是分析的样方组不一定包括所有二级分类，因此使用集合交集的方法移除未包含的类别。

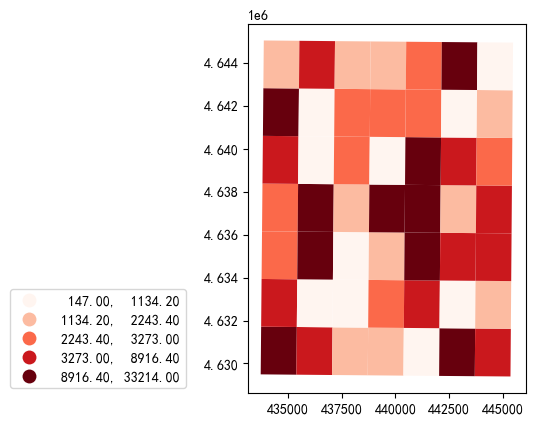
landuse\_zonal\_stats\_a\_columns=set(landuse\_zonal\_stats\_a\_gdf.columns.to\_list())  
open\_space\_cols\_a=landuse\_zonal\_stats\_a\_columns.intersection(set(open\_space\_cols))  
print(open\_space\_cols\_a)  
  
landuse\_zonal\_stats\_a\_gdf['open\_space']=landuse\_zonal\_stats\_a\_gdf[open\_space\_cols\_a].sum(axis=1)  
gpd2postSQL(landuse\_zonal\_stats\_a\_gdf,'landuse\_zonal\_stats\_a',\*\*args.db.db\_info)   
ax=landuse\_zonal\_stats\_a\_gdf.plot('open\_space',k=5,scheme='Quantiles',legend=True,cmap='Reds')  
leg=ax.get\_legend()  
leg.set\_bbox\_to\_anchor((-0.4, 0.1, 0.2, 0.2))

{'fre\_44', 'fre\_45', 'fre\_41', 'fre\_43', 'fre\_42'}  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is landuse\_zonal\_stats\_a.  
  
  
C:\Users\richi\anaconda3\envs\usda\lib\site-packages\geopandas\geodataframe.py:1299: FutureWarning: Passing a set as an indexer is deprecated and will raise in a future version. Use a list instead.  
 result = super(GeoDataFrame, self).\_\_getitem\_\_(key)



landuse\_zonal\_stats\_b\_gdf=zonal\_stats\_raster(args.data.landuse\_tif,quadrats\_b\_gdf,stats=['count'])   
landuse\_zonal\_stats\_b\_columns=set(landuse\_zonal\_stats\_b\_gdf.columns.to\_list())  
open\_space\_cols\_b=landuse\_zonal\_stats\_b\_columns.intersection(set(open\_space\_cols))  
print(open\_space\_cols\_b)  
  
landuse\_zonal\_stats\_b\_gdf['open\_space']=landuse\_zonal\_stats\_b\_gdf[list(open\_space\_cols\_b)].sum(axis=1)  
gpd2postSQL(landuse\_zonal\_stats\_b\_gdf,'landuse\_zonal\_stats\_b',\*\*args.db.db\_info)   
ax=landuse\_zonal\_stats\_b\_gdf.plot('open\_space',k=5,scheme='Quantiles',legend=True,cmap='Reds')  
leg=ax.get\_legend()  
leg.set\_bbox\_to\_anchor((-0.4, 0.1, 0.2, 0.2))

{'fre\_41', 'fre\_42', 'fre\_45'}  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is landuse\_zonal\_stats\_b.



### 2.7.2.2 不平等性

基尼系数（Gini coefficient）是由意大利统计学家Corrado Gini于1912年提出， 作为社会收入不平等的一种测量方法[2]。基尼系数常与Max O. Lorenz于1905年提出的洛伦兹曲线（Lorenz curve）联系在一起。基尼系数是对分析对象一个不平等的衡量标准，定义为所有成对的个体在某种衡量标准下的绝对差异的均值，其公式为：，式中，为观测值，是观测值的数量，是观测值均值。当所有观测值均相等时，最小值为0，观察值之间的差异越大，值越趋近于1，理论上最大值为1。

洛伦兹曲线（Lorenz curve），为对观测值排序后，从低到高每（）数量的观测值之和（累积和）占总观测值之和的比例所绘制的曲线。

下述gini\_coefficient()函数用于计算基尼系数；lorenz\_curve()函数用于计算和绘制洛伦兹曲线。

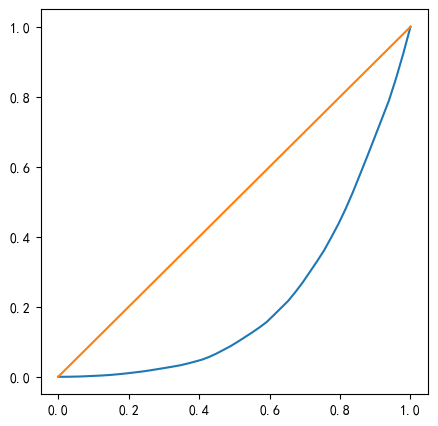
landuse\_zonal\_stats\_a\_gdf=postSQL2gpd(table\_name='landuse\_zonal\_stats\_a',\*\*args.db.db\_info)   
landuse\_zonal\_stats\_a\_gdf.fillna(0,inplace=True)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is landuse\_zonal\_stats\_a.  
  
  
C:\Users\richi\anaconda3\envs\usda\lib\site-packages\pandas\io\sql.py:580: SAWarning: Did not recognize type 'geometry' of column 'geometry'  
 pandas\_sql.meta.reflect(bind=pandas\_sql.connectable, only=[sql])

def gini\_coefficient(arr\_):  
 '''  
 计算基尼系数  
  
 Parameters  
 ----------  
 arr\_ : numpy.ndarray(1D)  
 一维数组（观测值）.  
  
 Returns  
 -------  
 float  
 基尼系数.  
  
 '''   
 import numpy as np  
   
 arr=np.sort(arr\_)  
 count=arr.size  
 coefficient=2/count  
 indexes=np.arange(1, count + 1)  
 weighted\_sum=(indexes \* arr).sum()  
 total=arr.sum()  
 constant=(count + 1) / count  
 return coefficient \* weighted\_sum / total - constant  
  
gini\_os\_custom=gini\_coefficient(landuse\_zonal\_stats\_a\_gdf['open\_space'])  
print(gini\_os\_custom)

0.5555786867146546

def lorenz\_curve(arr\_,plot=True,figsize=(5,5)):  
 '''  
 绘制洛伦兹曲线  
  
 Parameters  
 ----------  
 arr\_ : numpy.ndarray(1D)  
 一维数组（观测值）.  
 plot : TYPE, optional  
 DESCRIPTION. The default is True.  
  
 Returns  
 -------  
 lorenz\_curve\_vals : list(float)  
 从低到高累积占比值.  
  
 '''   
 import numpy as np  
 import matplotlib.pyplot as plt  
   
 # this divides the prefix sum by the total sum  
 # this ensures all the values are between 0 and 1.0  
 arr=np.sort(arr\_)  
 scaled\_prefix\_sum = arr.cumsum() / arr.sum()  
 # this prepends the 0 value (because 0% of all people have 0% of all wealth)  
 lorenz\_curve\_vals=np.insert(scaled\_prefix\_sum, 0, 0)  
   
 if plot:   
 fig=plt.figure(figsize=figsize)  
 plt.plot(np.linspace(0.0, 1.0, lorenz\_curve\_vals.size), lorenz\_curve\_vals)  
 plt.plot([0,1], [0,1])  
 plt.show()   
 return lorenz\_curve\_vals  
  
lorenz\_curve(landuse\_zonal\_stats\_a\_gdf['open\_space']);



inequality库提供了inequality.gini.Gini方法可以直接计算基尼系数，其计算结果同上述自定义函数计算结果。参考联合国开发计划署（United Nations Development Programme，UNDP）对收入分配基尼系数等级的划分，0.40是收入分配差距的“警戒线”，0.29小于0.40，表明各个样方开放空间的分布不平等性（不均衡性、不公平性）不明显，即具有一定的平等性。

import inequality  
  
gini\_os=inequality.gini.Gini(landuse\_zonal\_stats\_a\_gdf['open\_space'])  
print(gini\_os.g)

0.5555786867146548

inequality库提供了inequality.gini.Gini\_Spatial方法，在考虑空间权重条件下计算基尼系数，因为分析的各样方均匀排布，邻里单元的空间权重保持一致，因此从计算结果也可以发现，考虑空间权重后计算的基尼系数值同未考虑时计算的结果值。

import libpysal.weights as LW  
import numpy as np  
  
np.random.seed(12345)  
w=LW.Rook.from\_dataframe(landuse\_zonal\_stats\_a\_gdf)  
gini\_os\_w=inequality.gini.Gini\_Spatial(landuse\_zonal\_stats\_a\_gdf['open\_space'],w)  
print(gini\_os\_w.g,gini\_os\_w.p\_sim)

0.5555786867146548 0.01

### 2.7.2.3 空间隔离[4]

#### 1) 空间隔离指数（ aspatial indexes）

* Dissimilarity

Dissimilarity指数由Duncan, O.和 B. Duncan (1955)提出，其公式为：。从计算结果值可以得知有24%的开放空间必须置换到其它样方中才能达到完全平衡。

landuse\_zonal\_stats\_a\_gdf[['open\_space', 'count']]=landuse\_zonal\_stats\_a\_gdf[['open\_space', 'count']].astype('int')  
args\_seg=[landuse\_zonal\_stats\_a\_gdf, 'open\_space', 'count']

import segregation.aspatial as sa  
import tqdm as notebook\_tqdm  
  
dissim\_idx=sa.Dissim(\*args\_seg)  
print(dissim\_idx.statistic)

0.5194584397292389

* Gini（变体）

基尼系数有十多种变体，GiniSeg方法计算的基尼系数公式为： 。计算结果几近为0。

giniSeg\_idx=sa.GiniSeg(\*args\_seg)  
print(giniSeg\_idx.statistic)

0.0078008151095141035

* Entropy

Entropy（信息熵）指数计算公式为：，式中，为全局熵，公式为： ；为空间单元熵，公式为： 。

entropy\_idx=sa.Entropy(\*args\_seg)  
print(entropy\_idx.statistic)

0.12763678307822018

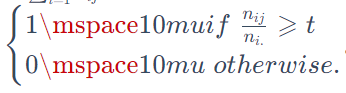
* Atkinson

Atkinson指数公式为： ，式中，为一个形参，决定了如何对洛伦兹曲线的不同部分所带来的隔离增量进行加权。

atkinson\_idx=sa.Atkinson(\*args\_seg)  
print(atkinson\_idx.statistic)

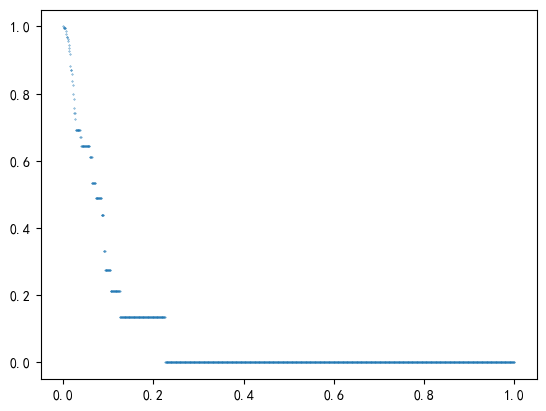
0.2652906513058595

* Concentration Profile

在*Measuring geographic segregation: a graph-based approach*一文中提到Concentration Profile 指数[5]，计算公式为： ，式中，为阈值比例，公式为：，式中为一个逻辑函数，公式为： g(t,i) =该指数提供了plot方法，可以打印图表查看估计值。

ConProf\_idx=sa.ConProf(\*args\_seg)  
print(ConProf\_idx.statistic)  
ConProf\_idx.plot()

0.06110307400271944  
  
  
  
  
  
<matplotlib.collections.PathCollection at 0x2499313caf0>



* Isolation

Isolation指数可以用于测量一个样方中所属一个类别的栅格单元，与另一个栅格单元属于同类别的概率，其公式为：。从下述计算结果可以说明同一样方中，同为开放空间类别的栅格单元的概率为21.41%。

Isolation\_idx=sa.Isolation(\*args\_seg)  
print(Isolation\_idx.statistic)

0.08353886171624425

* Exposure

Exposure指数与Isolation指数互补，可以用于测量一个样方中所属一个类别的栅格单元，与另一个栅格单元不属于同类别的概率，其公式为： 。从计算结果可以得知，

Exposure\_idx=sa.Exposure(os\_sum\_df, 'open\_space', 'count')  
print(Exposure\_idx.statistic)

0.7858889760253966

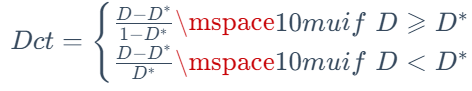
* Correlation Ratio

Correlation Ratio 指数公式为： 。

CorrelationR\_idx=sa.CorrelationR(\*args\_seg)  
print(CorrelationR\_idx.statistic)

0.05149295127792926

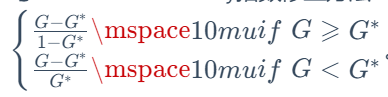
* Modified Dissimilarity

根据*On measuring segregation in samples with small units*论文提出的方法对Dissimilarity指数修正[6]，修正条件公式为： Dct =。

ModifiedDissim\_idx=sa.ModifiedDissim(\*args\_seg,iterations=500)  
print(ModifiedDissim\_idx.statistic)

0.4387485246174006

* Modified Gini

与Modified Dissimilarity指数修正方法一样[6]，Modified Gini 指数的修正条件公式为： Gct =。

ModifiedGiniSeg\_idx=sa.ModifiedGiniSeg(\*args\_seg,iterations=500)  
print(ModifiedGiniSeg\_idx.statistic)

0.006455065486869477

* Bias-Corrected Dissimilarity

Bias-Corrected Dissimilarity指数在*More reliable inference for the dissimilarity index of segregation*一文中提出[7]。

BiasCorrectedDissim\_idx=sa.BiasCorrectedDissim(\*args\_seg,B=500)  
print(BiasCorrectedDissim\_idx.statistic)

0.4419327858776766

* Density-Corrected Dissimilarity

Density-Corrected Dissimilarity指数同样由*More reliable inference for the dissimilarity index of segregation*一文中提出[7]。

DensityCorrectedDissim\_idx=sa.DensityCorrectedDissim(\*args\_seg,xtol = 1e-5)  
print(DensityCorrectedDissim\_idx.statistic)

C:\Users\richi\anaconda3\envs\usda\lib\site-packages\segregation\aspatial\aspatial\_indexes.py:1899: OptimizeWarning: Unknown solver options: xtol  
 res = minimize(fold\_norm,  
  
  
0.0339710074036396

* Minimum-Maximum Index (MM)

MM指数由*A surface‐based approach to measuring spatial segregation*一文提出，公式为：，式中下标是两个相互排斥的群体，例如开放空间和非开放空间。

MinMax\_idx=sa.MinMax(\*args\_seg)  
print(MinMax\_idx.statistic)

0.6129742318156732

#### 2) 组间比较与作用（决定）因素成分分解和贡献度

* 开放空间的构成比例

对样方组a和样方组b均计算开放空间栅格单元数open\_space列占各自样方栅格单元总数count的构成比例，将比值写入compo\_os列。并打印各组开放空间构成比例分布地图。

landuse\_zonal\_stats\_a\_gdf=postSQL2gpd(table\_name='landuse\_zonal\_stats\_a',\*\*args.db.db\_info)   
landuse\_zonal\_stats\_a\_gdf.fillna(0,inplace=True)  
  
landuse\_zonal\_stats\_a\_gdf['compo\_os']=np.where(landuse\_zonal\_stats\_a\_gdf['count']==0, 0, landuse\_zonal\_stats\_a\_gdf['open\_space'] / landuse\_zonal\_stats\_a\_gdf['count'])  
landuse\_zonal\_stats\_a\_gdf.head()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is landuse\_zonal\_stats\_a.  
  
  
C:\Users\richi\anaconda3\envs\usda\lib\site-packages\pandas\io\sql.py:580: SAWarning: Did not recognize type 'geometry' of column 'geometry'  
 pandas\_sql.meta.reflect(bind=pandas\_sql.connectable, only=[sql])

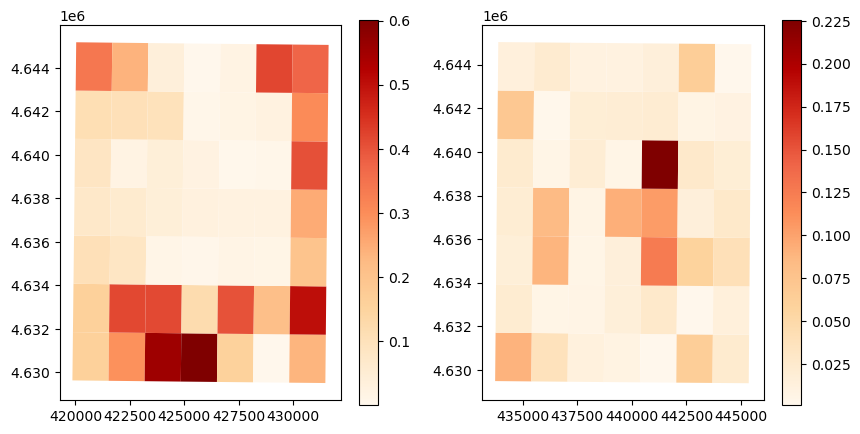
|  | **geometry** | **count** | **index** | **fre\_0** | **fre\_43** | **fre\_55** | **fre\_10** | **fre\_46** | **fre\_4** | **fre\_14** | **...** | **fre\_51** | **fre\_53** | **fre\_31** | **fre\_39** | **fre\_52** | **fre\_36** | **fre\_3** | **fre\_44** | **open\_space** | **compo\_os** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | POLYGON ((421520.398 4629617.466, 419859.219 4... | 147546 | 0 | 47103 | 23532.0 | 19612 | 18269.0 | 10250.0 | 7814.0 | 5845.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 23667.0 | 0.160404 |
| **1** | POLYGON ((421544.820 4631838.009, 419884.158 4... | 147500 | 0 | 18338 | 0.0 | 29150 | 27149.0 | 541.0 | 8627.0 | 1245.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 23656.0 | 0.160380 |
| **2** | POLYGON ((421569.252 4634058.559, 419909.107 4... | 147453 | 0 | 35973 | 6802.0 | 39109 | 5126.0 | 1261.0 | 4477.0 | 1366.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 15376.0 | 0.104277 |
| **3** | POLYGON ((421593.693 4636279.118, 419934.066 4... | 147408 | 0 | 72298 | 4878.0 | 32851 | 728.0 | 597.0 | 192.0 | 6550.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 10945.0 | 0.074250 |
| **4** | POLYGON ((421618.144 4638499.683, 419959.034 4... | 147363 | 0 | 74336 | 2010.0 | 29595 | 489.0 | 550.0 | 0.0 | 848.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 12682.0 | 0.086060 |

5 rows × 60 columns

landuse\_zonal\_stats\_b\_gdf=postSQL2gpd(table\_name='landuse\_zonal\_stats\_b',\*\*args.db.db\_info)   
landuse\_zonal\_stats\_b\_gdf.fillna(0,inplace=True)  
  
landuse\_zonal\_stats\_b\_gdf['compo\_os']=np.where(landuse\_zonal\_stats\_b\_gdf['count'] == 0, 0, landuse\_zonal\_stats\_b\_gdf['open\_space'] / landuse\_zonal\_stats\_b\_gdf['count'])

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is landuse\_zonal\_stats\_b.  
  
  
C:\Users\richi\anaconda3\envs\usda\lib\site-packages\pandas\io\sql.py:580: SAWarning: Did not recognize type 'geometry' of column 'geometry'  
 pandas\_sql.meta.reflect(bind=pandas\_sql.connectable, only=[sql])

fig, (ax1, ax2)=plt.subplots(ncols=2, figsize=(10, 5),)  
landuse\_zonal\_stats\_a\_gdf.plot(column='compo\_os', cmap='OrRd', legend=True,ax=ax1)  
landuse\_zonal\_stats\_b\_gdf.plot(column='compo\_os', cmap='OrRd', legend=True,ax=ax2)  
plt.show()



* 作用因素成分分解和贡献度

根据*Comparative Spatial Segregation Analytics*研究论文[9]，作者采用了Shapley分解（Shapley decomposition）方法[10]研究组间隔离差异，公式为：，结合属性分量（attribute componnet， CA和空间分量（spatial component， ）贡献度，公式为： F(S, A) = CS +CA= ΔG，式中，

。

对于空间分量，获取两个估计，第一个估计是保持a组属性分量A1​不变，而空间分量为a组和b组的空间分量S1​和S2​产生的差异；第二个估计保持b组属性分量A2​不变，空间分量也为a组和b组的空间分量S1​和S2​产生的差异，最后的空间分量被视为这两个估计的均值。在每一估计中都会产生一个反设事实（空间分量2个，属性分量2个），与其真实观测到的隔离系数相对照。对于空间分量，第一个估计的反设事实为G(S2​,A1​)，为组a保持属性分量不变而将空间分量替换为组b的空间分量。这个反设事实的隔离指数与真实a组的G(S1​,A1​)隔离指数之间的差异归因于空间分量的变化，因为空间分量是两组间唯一变化的部分；第二个估计的反设事实为G(S1​,A2​)，为组b保持属性分量不变而将空间分量替换为组a的空间分量。这个反设事实的隔离指数与真实b组的G(S2​,A2​)隔离指数之间的差异同样归因于空间分量的变化。

同样对于属性分量也有两个估计，是保持空间分量而变化不同组的属性分量。即保持各组的空间分量不变而将属性分替换为另一组的属性分类，从而在各个估计中，空间分量是不变量，组间隔离指数的差异归因于属性分量的变化。

对于*Comparative Spatial Segregation Analytics*提出的组间隔离差异分量贡献度，segregation库提供了DecomposeSegregation方法方便直接计算，该方法提供的counterfactual\_approach参数有3种方法，包括”composition”, “share”, “dual\_composition”，是用于生成“counterfactual（反设事实）”对象的方式。

**counterfactual\_approach=“composition”**

以Gini隔离指数为例，a组值为0.661，b组值为0.564，二者存在0.097的差异，那么两组之间的这个差异受到空间结构和属性组成的影响分别是多少？从计算结果得知，空间结构的影响几近为0，而属性组成，即土地利用类型的分布起到主导作用，值约为0.097，这与两组Gini隔离指数差值基本同。

对于”share”, “dual\_composition”的解释可以阅读[decomposition\_wrapper\_example](http://pysal.org/notebooks/explore/segregation/decomposition_wrapper_example.html)④。

import segregation.aspatial as sa  
from tqdm import tqdm  
  
G\_a=sa.GiniSeg(landuse\_zonal\_stats\_a\_gdf, 'open\_space', 'count')  
G\_b=sa.GiniSeg(landuse\_zonal\_stats\_b\_gdf, 'open\_space', 'count')  
print(f"G\_a={G\_a.statistic},\nG\_b={G\_b.statistic},\nG\_a-G\_b={G\_a.statistic-G\_b.statistic}")

G\_a=0.6619845430595112,  
G\_b=0.5649299326429257,  
G\_a-G\_b=0.09705461041658547

对于DecomposeSegregation方法的理解可以通过help(DecomposeSegregation)查看。

from segregation.decomposition import DecomposeSegregation  
help(DecomposeSegregation)

Help on class DecomposeSegregation in module segregation.decomposition.decompose\_segregation:  
  
class DecomposeSegregation(builtins.object)  
 | DecomposeSegregation(index1, index2, counterfactual\_approach='composition')  
 |   
 | Decompose segregation differences into spatial and attribute components.  
 |   
 | Given two segregation indices of the same type, use Shapley decomposition  
 | to measure whether the differences between index measures arise from  
 | differences in spatial structure or population structure  
 |   
 | Parameters  
 | ----------  
 | index1 : segregation.SegIndex class  
 | First SegIndex class to compare.  
 | index2 : segregation.SegIndex class  
 | Second SegIndex class to compare.  
 | counterfactual\_approach : str, one of  
 | ["composition", "share", "dual\_composition"]  
 | The technique used to generate the counterfactual population  
 | distributions.  
 |   
 | Attributes  
 | ----------  
 |   
 | c\_s : float  
 | Shapley's Spatial Component of the decomposition  
 |   
 | c\_a : float  
 | Shapley's Attribute Component of the decomposition  
 |   
 | Methods  
 | ----------  
 |   
 | plot : Visualize features of the Decomposition performed  
 | plot\_type : str, one of ['cdfs', 'maps']  
 |   
 | 'cdfs' : visualize the cumulative distribution functions of the compositions/shares  
 | 'maps' : visualize the spatial distributions for original data and counterfactuals generated and Shapley's components (only available for GeoDataFrames)  
 |   
 | Examples  
 | --------  
 | Several examples can be found at https://github.com/pysal/segregation/blob/master/notebooks/decomposition\_wrapper\_example.ipynb.  
 |   
 | Methods defined here:  
 |   
 | \_\_init\_\_(self, index1, index2, counterfactual\_approach='composition')  
 | Initialize self. See help(type(self)) for accurate signature.  
 |   
 | plot(self, plot\_type='cdfs')  
 | Plot the Segregation Decomposition Profile  
 |   
 | ----------------------------------------------------------------------  
 | Data descriptors defined here:  
 |   
 | \_\_dict\_\_  
 | dictionary for instance variables (if defined)  
 |   
 | \_\_weakref\_\_  
 | list of weak references to the object (if defined)

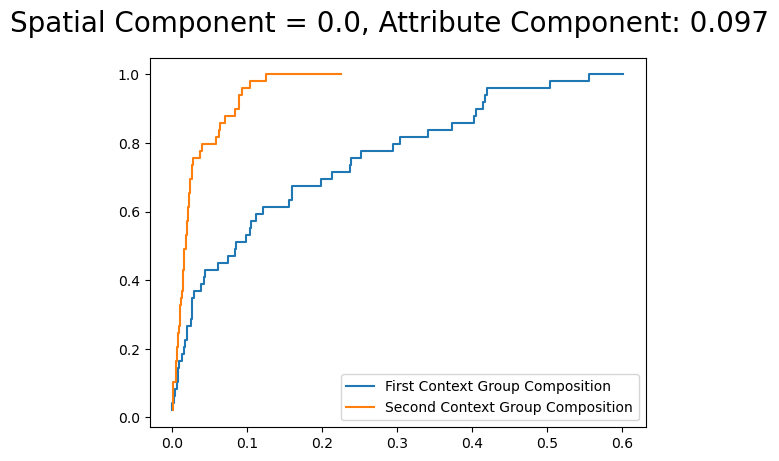
counterfactual\_approach参数默认为composition方式。

DS\_composition=DecomposeSegregation(G\_a, G\_b)  
print(f"Spatial component: {DS\_composition.c\_s}\nattribute component: {DS\_composition.c\_a}")

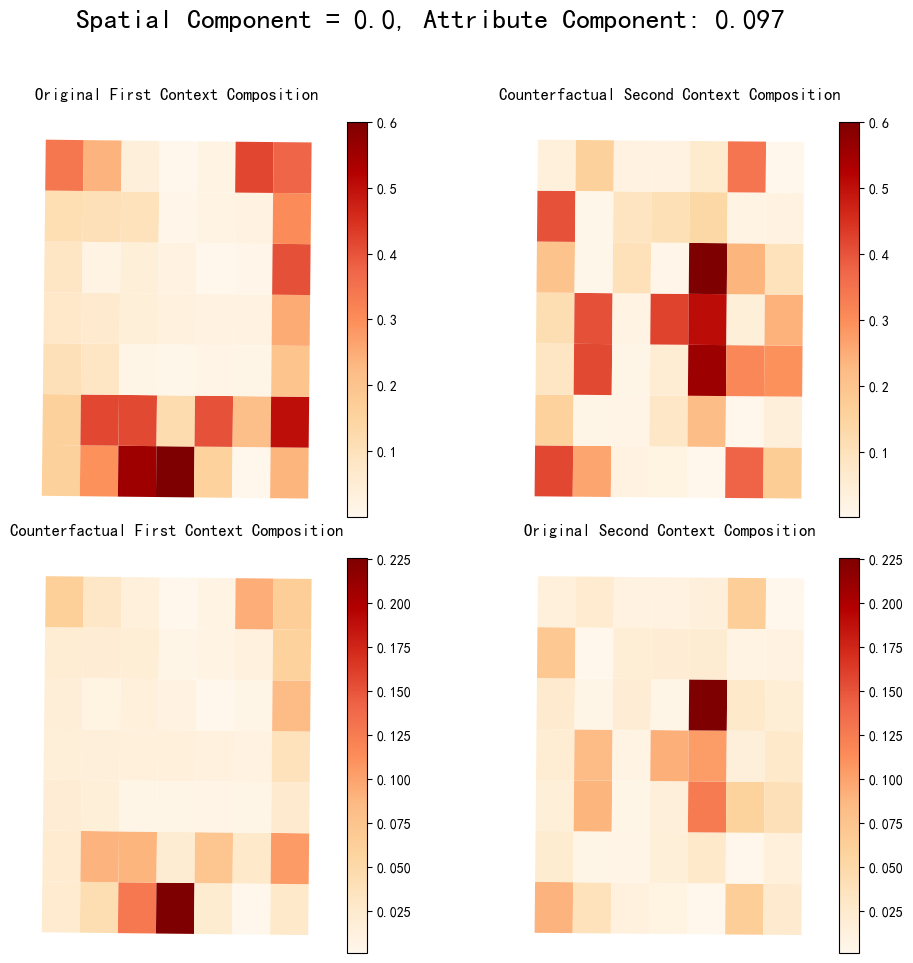
Spatial component: 8.982971253457261e-06  
attribute component: 0.09704562744533202

DecomposeSegregation方法的返回值具有plot打印图表方法，提供的plot\_type参数包括cdfs和maps，其中cdfs为各组成（compositions）或份额（share）的累积量分布，本例为开放空间的栅格单元数（compositions）或者其占对应组开放空间总栅格单元数的比例（share） ；maps为原始数据的空间分布（同上文中的开放空间构成比例），以及counterfactuals反设事实的开放空间构成比例分布。

DS\_composition.plot(plot\_type='cdfs')



DS\_composition.plot(plot\_type='maps')



#### 3) 空间隔离指数推断

空间隔离指数返回的类通常具有statistic属性，但是没有p-values值，不能进行统计推断确定指数返回值是否具有统计学意义，而segregation.inference模块提供了统计显著性检验的两种方法，一种是对单值的检验，为SingleValueTest；另一种是对二值的检验，为TwoValueTest。检验方法接收的参数为空间隔离指数返回的指数类，并由null\_approach参数确定无效假设的类型，由iterations\_under\_null参数确定在无效假设下迭代的次数。需要注意，检验方法对null\_approach参数提供了多个无效假设类型，不同类型选择直接影响统计推断的结果。

* 非空间单指数统计显著性

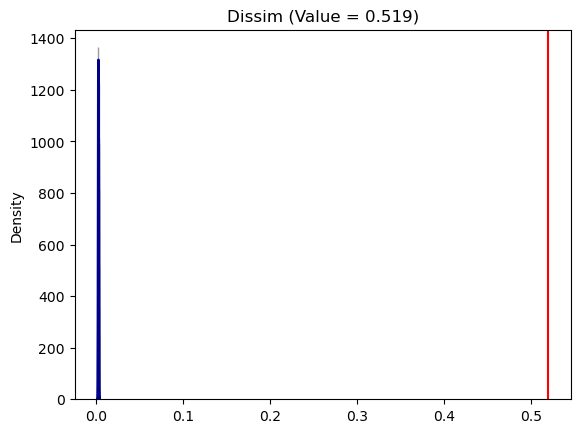
以Dissimilarity指数为例，计算结果约为0.519。定义无效假设类型为evenness，即假设每一空间单元从固定总体的某类对象中抽取元素的全局概率相同（二项分布）。以无效假设数据计算Dissimilarity指数为iterations\_under\_null给定的1000次，打印分布，发现0.519位于右侧，远偏离无效假设分布区域，且p-value值为0，因此拒绝原假设，Dissimilarity指数表明样方组区域各个样方之间存在开放空间明显隔离现象，部分样方有较多的开放空间栅格单元，但部分则很少或没有，且样方之间开放空间的栅格单元连通较少。

dissim\_idx=sa.Dissim(\*args\_seg)  
print(dissim\_idx.statistic)

0.5194584397292389

from segregation.inference import SingleValueTest, TwoValueTest  
  
infer\_D\_eve=SingleValueTest(dissim\_idx, iterations\_under\_null=1000, null\_approach="evenness", two\_tailed=True)  
print(f"The mean value of the distribution={infer\_D\_eve.est\_sim.mean()}\np-value={infer\_D\_eve.p\_value}")  
infer\_D\_eve.plot()

Processed 1000 iterations out of 1000: 100%|██████████| 1000/1000 [00:03<00:00, 263.03it/s]  
C:\Users\richi\anaconda3\envs\usda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).  
 warnings.warn(msg, FutureWarning)  
  
  
The mean value of the distribution=0.002802174544920078  
p-value=0.0  
  
  
  
  
  
<AxesSubplot: title={'center': 'Dissim (Value = 0.519)'}, ylabel='Density'>



* 空间单指数统计显著性

**Relative Concentration**

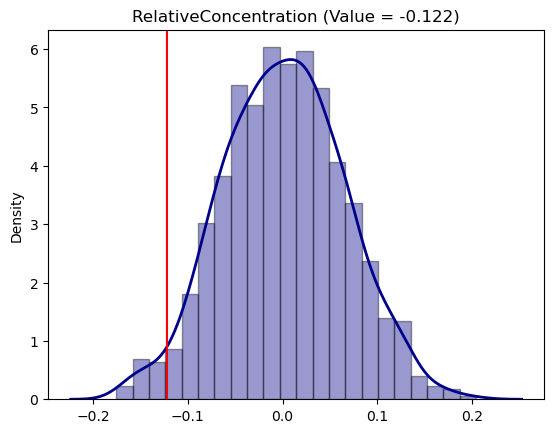
RelativeConcentration指数为一个空间指数（受空间结构影响），因此无效假设类型选择为permutation，即在空间上随机分配单元，且保持原始值不变。计算无效假设的RelativeConcentration指数，打印分布，其p-value值为0.05，表明RelativeConcentration指数统计显著性不明显。但是，如果将无效假设类型选择为even\_permutation，综合evenness和permutation类型，则计算结果具有统计显著性。

from segregation.spatial import RelativeConcentration  
RCO=RelativeConcentration(\*args\_seg)  
RCO.statistic

-0.12219593013769346

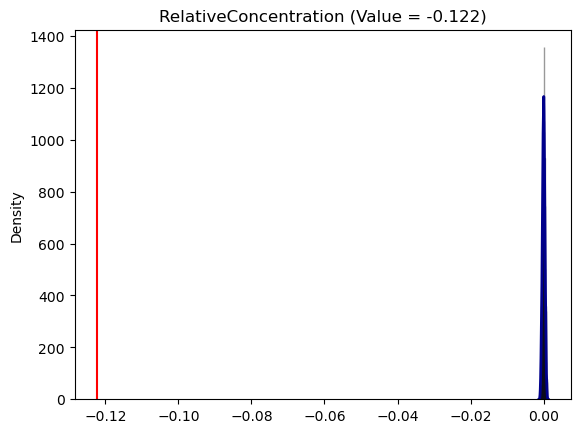
infer\_RCO\_per=SingleValueTest(RCO, iterations\_under\_null=1000, null\_approach="permutation", two\_tailed=True)  
print(f"p-value={infer\_RCO\_per.p\_value}")  
infer\_RCO\_per.plot()

Processed 1000 iterations out of 1000: 100%|██████████| 1000/1000 [00:06<00:00, 165.50it/s]  
C:\Users\richi\anaconda3\envs\usda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).  
 warnings.warn(msg, FutureWarning)  
  
  
p-value=0.054  
  
  
  
  
  
<AxesSubplot: title={'center': 'RelativeConcentration (Value = -0.122)'}, ylabel='Density'>



infer\_RCO\_eve\_per=SingleValueTest(RCO, iterations\_under\_null = 1000, null\_approach = "even\_permutation", two\_tailed = True)  
infer\_RCO\_eve\_per.plot()

Processed 1000 iterations out of 1000: 100%|██████████| 1000/1000 [00:06<00:00, 146.16it/s]  
C:\Users\richi\anaconda3\envs\usda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).  
 warnings.warn(msg, FutureWarning)  
  
  
  
  
  
<AxesSubplot: title={'center': 'RelativeConcentration (Value = -0.122)'}, ylabel='Density'>

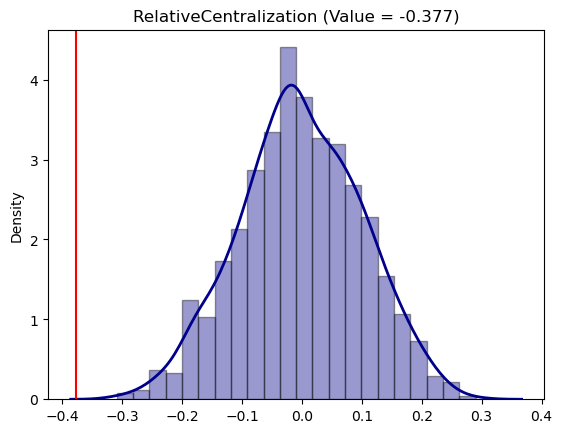


**Relative Centralization**

同样检验Relative Centralization指数，无效假设类型为permutation。因为结果值-0.376位于分布左侧末尾，表明开放空间分布的集中化方面有明显的副作用（负值），即开放空间栅格单元分布通常偏离样方组的中心区域。

from segregation.spatial import RelativeCentralization  
RCE=RelativeCentralization(\*args\_seg)  
print(f"RCE={RCE.statistic}")  
infer\_RCE\_per=SingleValueTest(RCE, iterations\_under\_null=1000, null\_approach="permutation", two\_tailed=True)  
infer\_RCE\_per.plot()

RCE=-0.3768144764624637  
  
  
Processed 1000 iterations out of 1000: 100%|██████████| 1000/1000 [00:14<00:00, 70.54it/s]  
C:\Users\richi\anaconda3\envs\usda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).  
 warnings.warn(msg, FutureWarning)  
  
  
  
  
  
<AxesSubplot: title={'center': 'RelativeCentralization (Value = -0.377)'}, ylabel='Density'>

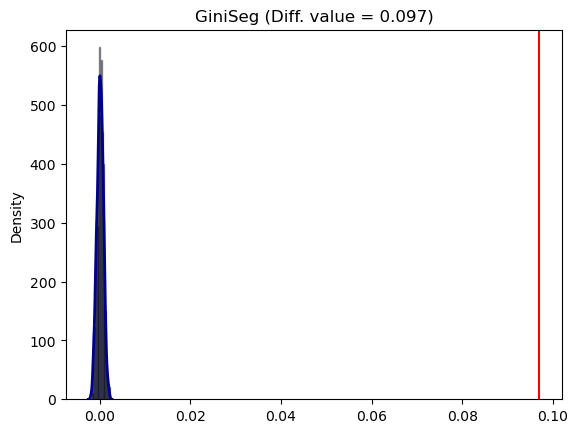


* 非空间双指数（二值）统计显著性

二值方法基本同单值方法，但需要传入两组同一隔离指数计算的返回类，例如传入样方组a和b计算的Gini系数。

compare\_G\_fit=TwoValueTest(G\_a, G\_b, iterations\_under\_null=1000, null\_approach="random\_label")  
compare\_G\_fit.plot()

Processed 1000 iterations out of 1000: 100%|██████████| 1000/1000 [00:10<00:00, 94.37it/s]  
C:\Users\richi\anaconda3\envs\usda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).  
 warnings.warn(msg, FutureWarning)  
  
  
  
  
  
<AxesSubplot: title={'center': 'GiniSeg (Diff. value = 0.097)'}, ylabel='Density'>

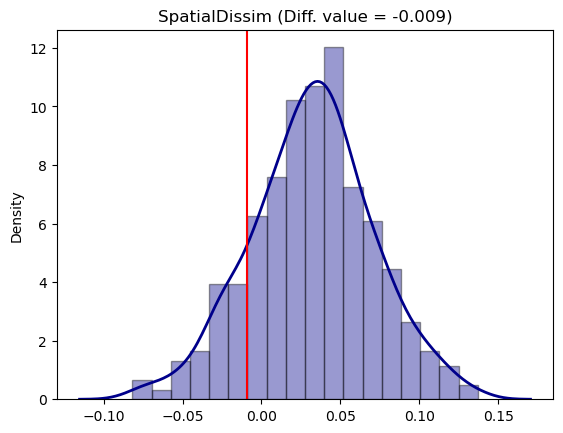


* 空间双指数（二值）统计显著性

二值方法同样使用于空间指数。

from segregation.spatial import SpatialDissim  
SD\_a=SpatialDissim(landuse\_zonal\_stats\_a\_gdf, 'open\_space', 'count')  
SD\_b=SpatialDissim(landuse\_zonal\_stats\_b\_gdf, 'open\_space', 'count')  
  
compare\_SD\_fit=TwoValueTest(SD\_a, SD\_b, iterations\_under\_null = 500, null\_approach = "counterfactual\_composition")  
compare\_SD\_fit.plot()

Processed 500 iterations out of 500: 100%|██████████| 500/500 [00:11<00:00, 43.53it/s]  
C:\Users\richi\anaconda3\envs\usda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).  
 warnings.warn(msg, FutureWarning)  
  
  
  
  
  
<AxesSubplot: title={'center': 'SpatialDissim (Diff. value = -0.009)'}, ylabel='Density'>



#### 4) 局部空间隔离指数

segregation.local模块提供有多个局部空间隔离指数，包括有MultiLocationQuotient为Location Quotient (LQ), MultiLocalDiversity为Local Diversity, MultiLocalEntropy为Local Entropy, MultiLocalSimpsonInteraction为Local Simpson Interaction, MultiLocalSimpsonConcentration为Local Simpson Concentration , LocalRelativeCentralization为Local Centralization等。

from segregation.local import MultiLocationQuotient, MultiLocalDiversity, MultiLocalEntropy, MultiLocalSimpsonInteraction, MultiLocalSimpsonConcentration, LocalRelativeCentralization  
  
groups\_list=['fre\_41','fre\_42','fre\_43','fre\_44','fre\_45'] # '开放空间主要娱乐设施': 41, '高尔夫球场': 42, '开放空间主要保护区': 43, '非公共开放空间': 44, '步道或绿道':45  
M\_location\_quotient=MultiLocationQuotient(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
print(M\_location\_quotient.statistics.shape)  
M\_location\_quotient.statistics[:3,:]

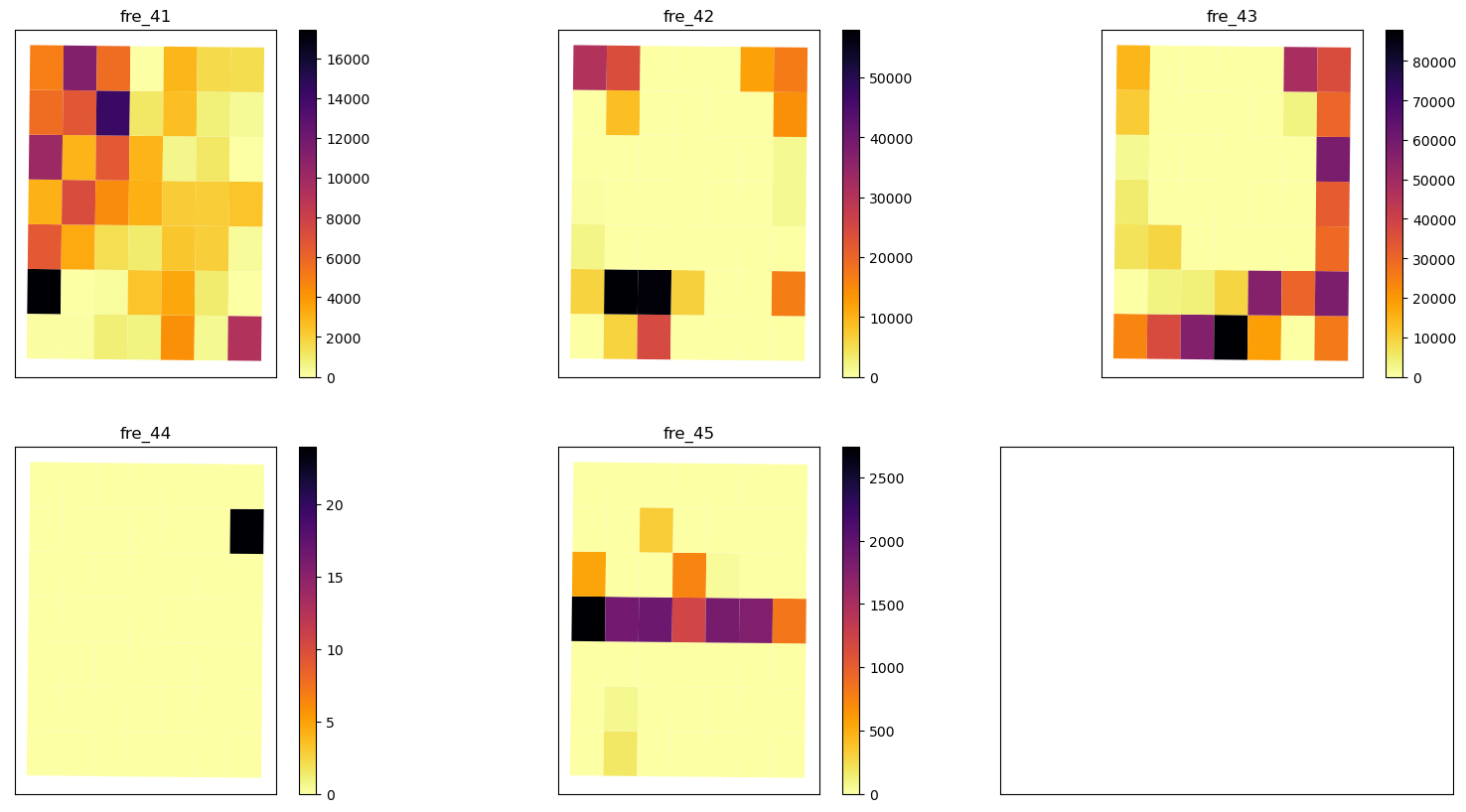
(49, 5)  
  
  
  
  
  
array([[0.04016389, 0. , 1.65965646, 0. , 0. ],  
 [5.19992145, 1.0597044 , 0. , 0. , 0. ],  
 [3.00953483, 0.52763894, 0.73840704, 0. , 0. ]])

for i in range(len(groups\_list)):  
 landuse\_zonal\_stats\_a\_gdf['LQ\_' + groups\_list[i]]=M\_location\_quotient.statistics[:,i]  
landuse\_zonal\_stats\_a\_gdf.head()

|  | **geometry** | **count** | **index** | **fre\_0** | **fre\_43** | **fre\_55** | **fre\_10** | **fre\_46** | **fre\_4** | **fre\_14** | **...** | **fre\_36** | **fre\_3** | **fre\_44** | **open\_space** | **compo\_os** | **LQ\_fre\_41** | **LQ\_fre\_42** | **LQ\_fre\_43** | **LQ\_fre\_44** | **LQ\_fre\_45** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | POLYGON ((421520.398 4629617.466, 419859.219 4... | 147546 | 0 | 47103 | 23532.0 | 19612 | 18269.0 | 10250.0 | 7814.0 | 5845.0 | ... | 0.0 | 0.0 | 0.0 | 23667 | 0.160404 | 0.040164 | 0.000000 | 1.659656 | 0.0 | 0.000000 |
| **1** | POLYGON ((421544.820 4631838.009, 419884.158 4... | 147500 | 0 | 18338 | 0.0 | 29150 | 27149.0 | 541.0 | 8627.0 | 1245.0 | ... | 0.0 | 0.0 | 0.0 | 23656 | 0.160380 | 5.199921 | 1.059704 | 0.000000 | 0.0 | 0.000000 |
| **2** | POLYGON ((421569.252 4634058.559, 419909.107 4... | 147453 | 0 | 35973 | 6802.0 | 39109 | 5126.0 | 1261.0 | 4477.0 | 1366.0 | ... | 0.0 | 0.0 | 0.0 | 15376 | 0.104277 | 3.009535 | 0.527639 | 0.738407 | 0.0 | 0.000000 |
| **3** | POLYGON ((421593.693 4636279.118, 419934.066 4... | 147408 | 0 | 72298 | 4878.0 | 32851 | 728.0 | 597.0 | 192.0 | 6550.0 | ... | 0.0 | 0.0 | 0.0 | 10945 | 0.074250 | 1.945410 | 0.109595 | 0.743924 | 0.0 | 20.750819 |
| **4** | POLYGON ((421618.144 4638499.683, 419959.034 4... | 147363 | 0 | 74336 | 2010.0 | 29595 | 489.0 | 550.0 | 0.0 | 848.0 | ... | 0.0 | 0.0 | 0.0 | 12682 | 0.086060 | 5.620948 | 0.000000 | 0.264552 | 0.0 | 3.572606 |

5 rows × 65 columns

def gdf\_layoutPlot\_simple(gdf,cols,fig\_cols,figsize=(10,10)):  
 '''  
 同时打印GeoDataFrame多个列于子图  
  
 Parameters  
 ----------  
 gdf : GeoDataFrame  
 GeoDataFrame格式数据.  
 cols : list(string)  
 待打印的列名列表.  
 fig\_cols : int  
 打印子图列数.  
 figsize : tuple(int), optional  
 图表大小. The default is (10,10).  
  
 Returns  
 -------  
 None.  
  
 '''   
 import math  
 import matplotlib.pyplot as plt  
   
 fig\_rows=math.ceil(len(cols)/fig\_cols)  
 fig,axes=plt.subplots(fig\_rows,fig\_cols,sharex=True,sharey=True,figsize=figsize)   
 ax=axes.flatten()   
 for i in range(len(cols)):  
 gdf.plot(column=cols[i],cmap='inferno\_r',legend=True,ax=ax[i])  
 ax[i].set\_title(cols[i])  
 ax[i].set\_xticks([])  
 ax[i].set\_yticks([])  
 ax[i].set\_facecolor('white')   
   
gdf\_layoutPlot\_simple(landuse\_zonal\_stats\_a\_gdf,groups\_list,3,figsize=(20,10))



M\_local\_diversity=MultiLocalDiversity(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
M\_local\_entropy=MultiLocalEntropy(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
M\_local\_simpson\_intersaction=MultiLocalSimpsonInteraction(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
M\_local\_simpson\_concentration=MultiLocalSimpsonConcentration(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
local\_centralization=LocalRelativeCentralization(landuse\_zonal\_stats\_a\_gdf, 'open\_space', 'count')  
  
M\_local\_idxes={"M\_local\_diversity":M\_local\_diversity.statistics,  
 "M\_local\_entropy":M\_local\_entropy.statistics,  
 "M\_local\_simpson\_intersaction":M\_local\_simpson\_intersaction.statistics,  
 "M\_local\_simpson\_concentration":M\_local\_simpson\_concentration.statistics,  
 "local\_centralization":local\_centralization.statistics}  
for k,v in M\_local\_idxes.items():  
 landuse\_zonal\_stats\_a\_gdf[k]=v  
   
landuse\_zonal\_stats\_a\_gdf.head(3)

|  | **geometry** | **count** | **index** | **fre\_0** | **fre\_43** | **fre\_55** | **fre\_10** | **fre\_46** | **fre\_4** | **fre\_14** | **...** | **LQ\_fre\_41** | **LQ\_fre\_42** | **LQ\_fre\_43** | **LQ\_fre\_44** | **LQ\_fre\_45** | **M\_local\_diversity** | **M\_local\_entropy** | **M\_local\_simpson\_intersaction** | **M\_local\_simpson\_concentration** | **local\_centralization** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | POLYGON ((421520.398 4629617.466, 419859.219 4... | 147546 | 0 | 47103 | 23532.0 | 19612 | 18269.0 | 10250.0 | 7814.0 | 5845.0 | ... | 0.040164 | 0.000000 | 1.659656 | 0.0 | 0.0 | 0.035159 | 0.021845 | 0.011343 | 0.988657 | -0.289302 |
| **1** | POLYGON ((421544.820 4631838.009, 419884.158 4... | 147500 | 0 | 18338 | 0.0 | 29150 | 27149.0 | 541.0 | 8627.0 | 1245.0 | ... | 5.199921 | 1.059704 | 0.000000 | 0.0 | 0.0 | 0.574618 | 0.357030 | 0.386234 | 0.613766 | 0.065970 |
| **2** | POLYGON ((421569.252 4634058.559, 419909.107 4... | 147453 | 0 | 35973 | 6802.0 | 39109 | 5126.0 | 1261.0 | 4477.0 | 1366.0 | ... | 3.009535 | 0.527639 | 0.738407 | 0.0 | 0.0 | 0.989541 | 0.614836 | 0.604662 | 0.395338 | -0.306343 |

3 rows × 70 columns

gdf\_layoutPlot\_simple(landuse\_zonal\_stats\_a\_gdf,list(M\_local\_idxes.keys()),3,figsize=(20,10))



#### 5) 多类空间隔离指数

segregation.aspatial模块提供有多类空间隔离指数，包括有MultiDissim为Multigroup Dissimilarity Index，MultiGiniSeg为Multigroup Gini Index，MultiNormalizedExposure为Multigroup Normalized Exposure Index，MultiInformationTheory为Multigroup Information Theory Index，MultiRelativeDiversity为Multigroup Relative Diversity Index，MultiSquaredCoefficientVariation为Multigroup Squared Coefficient of Variation Index，MultiDiversity为Multigroup Diversity Index，SimpsonsConcentration为Simpson’s Concentration Index (lambda)，SimpsonsInteraction为Simpson’s Interaction Index (I)，MultiDivergence为Multigroup Divergence Index等。

import segregation.aspatial as sa  
  
M\_dissimilarity=sa.MultiDissim(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
M\_gini=sa.MultiGiniSeg(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
M\_normalized\_exposure=sa.MultiNormalizedExposure(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
M\_information\_theory=sa.MultiInformationTheory(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
M\_relative\_diversity=sa.MultiRelativeDiversity(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
M\_squared\_coefficient\_of\_variation=sa.MultiSquaredCoefficientVariation(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
M\_diversity=sa.MultiDiversity(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
M\_simpsons\_concentration=sa.SimpsonsConcentration(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
M\_simpsons\_interaction=sa.SimpsonsInteraction(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
M\_divergence=sa.MultiDivergence(landuse\_zonal\_stats\_a\_gdf, groups\_list)  
  
M\_aspatial\_idxes={"M\_dissimilarity":M\_dissimilarity.statistic,  
 "M\_gini":M\_gini.statistic,  
 "M\_normalized\_exposure":M\_normalized\_exposure.statistic,  
 "M\_information\_theory":M\_information\_theory.statistic,  
 "M\_relative\_diversity":M\_relative\_diversity.statistic,  
 "M\_squared\_coefficient\_of\_variation":M\_squared\_coefficient\_of\_variation.statistic,  
 "M\_diversity":M\_diversity.statistic,  
 "M\_simpsons\_concentration":M\_simpsons\_concentration.statistic,  
 "M\_simpsons\_interaction":M\_simpsons\_interaction.statistic,  
 "M\_divergence":M\_divergence.statistic}  
  
print(M\_aspatial\_idxes)

{'M\_dissimilarity': 0.6732150430715581, 'M\_gini': 0.8434789124832778, 'M\_normalized\_exposure': 0.5278542428716425, 'M\_information\_theory': 0.5498565988739315, 'M\_relative\_diversity': 0.5191008310172185, 'M\_squared\_coefficient\_of\_variation': 0.32708599440620156, 'M\_diversity': 0.9830556615079412, 'M\_simpsons\_concentration': 0.44012719596442745, 'M\_simpsons\_interaction': 0.5598728040355725, 'M\_divergence': 0.5405396425405196}

注释（Notes）：

① 芝加哥土地利用2015年更新数据（已更新至2018年）， （<https://www.cmap.illinois.gov/data/land-use/inventory>）。

② pgAdmin，为最流行和功能丰富的PostgreSQL开源管理和开发平台（<https://www.pgadmin.org/>）。

③ Chicago Data Portal，CDP，为芝加哥城开放数据门户，可免费下载数据用于相关分析，其中许多数据集每天至少更新一次或数次（<https://data.cityofchicago.org/>）。

④ decomposition\_wrapper\_example，（<http://pysal.org/notebooks/explore/segregation/decomposition_wrapper_example.html>）。

参考文献（References）:

[1] Chicago Metropolitan Agency for Planning. Chicago Metropolitan Agency for Planning 2015 Parcel-Based Land Use Inventory Categories September 2020, https://www.cmap.illinois.gov/data/land-use/inventory.

[2] Gini C. “Variabilitá e mutabilita” 1912 reprinted in Memorie di metodologica statistica (Ed. Pizetti E, Salvemini, T). Rome: Libreria Eredi Virgilio Veschi 1955.

[3] Lorenz MO. Methods for measuring the concentration of wealth. Journal of the American Statistical Association 1905;9:209-219.

[4] PySAL Notebooks. Segregation, <http://pysal.org/notebooks/explore/segregation/intro.html>.

[5] Hong, Seong-Yun, and Yukio Sadahiro. “Measuring geographic segregation: a graph-based approach.” Journal of Geographical Systems 16.2 (2014): 211-231.

[6] Carrington, W. J., Troske, K. R., 1997. On measuring segregation in samples with small units. Journal of Business & Economic Statistics 15 (4), 402–409.

[7] Allen, R., Burgess, S., Davidson, R., Windmeijer, F., 2015. More reliable inference for the dissimilarity index of segregation. The econometrics journal 18 (1), 40–66.

[8] O’Sullivan, David, and David WS Wong. “A surface‐based approach to measuring spatial segregation.” Geographical Analysis 39.2 (2007): 147-168.

[9] Rey, S. J., Cortes, R. & Knaap, E. Comparative Spatial Segregation Analytics. Spat Demogr 9, 31–56 (2021).

[10] Shorrocks, A. F. (2013). Decomposition procedures for distributional analysis: A unified framework based on the Shapley value. The Journal of Economic Inequality, 11(1), 99–126.