# 3.6-B 城市空间“语言”试验

## 3.6.1 基于 LULC 嵌入向量空间 Transformers 视觉模型的地表温度预测可行性试验

### 3.6.1.1 数据准备

#### 1) ESA WorldCover（LULC） 数据下载

欧空局（European Space Agency，ESA）基于 Sentinel-1 和 2 发布了 10 m高空分辨率 [2020年](https://worldcover2020.esa.int/)①和[2021年](https://worldcover2021.esa.int/)②的全球土地覆盖产品（WorldCover），该产品包含10个分类类别，为林地（Tree Cover）、灌木丛（Shrubland）、草地（Grassland）、农田（Cropland）、建成区（Built-up）、裸地/稀疏植被（Bare / sparse vegetation）、冰雪（Snow and Ice）、永久水体（Permanent water bodies）、草本湿地（Herbaceous Wetland）、苔藓和地衣（Moss and lichen）等10类，并增加有红树林（Mangrove）类，计有11类，其总体精度为77%（独立验证），使用投影 EPSG 编号为4326，即 WGS1984（World Geodetic System of 1984） 的地理坐标系统。

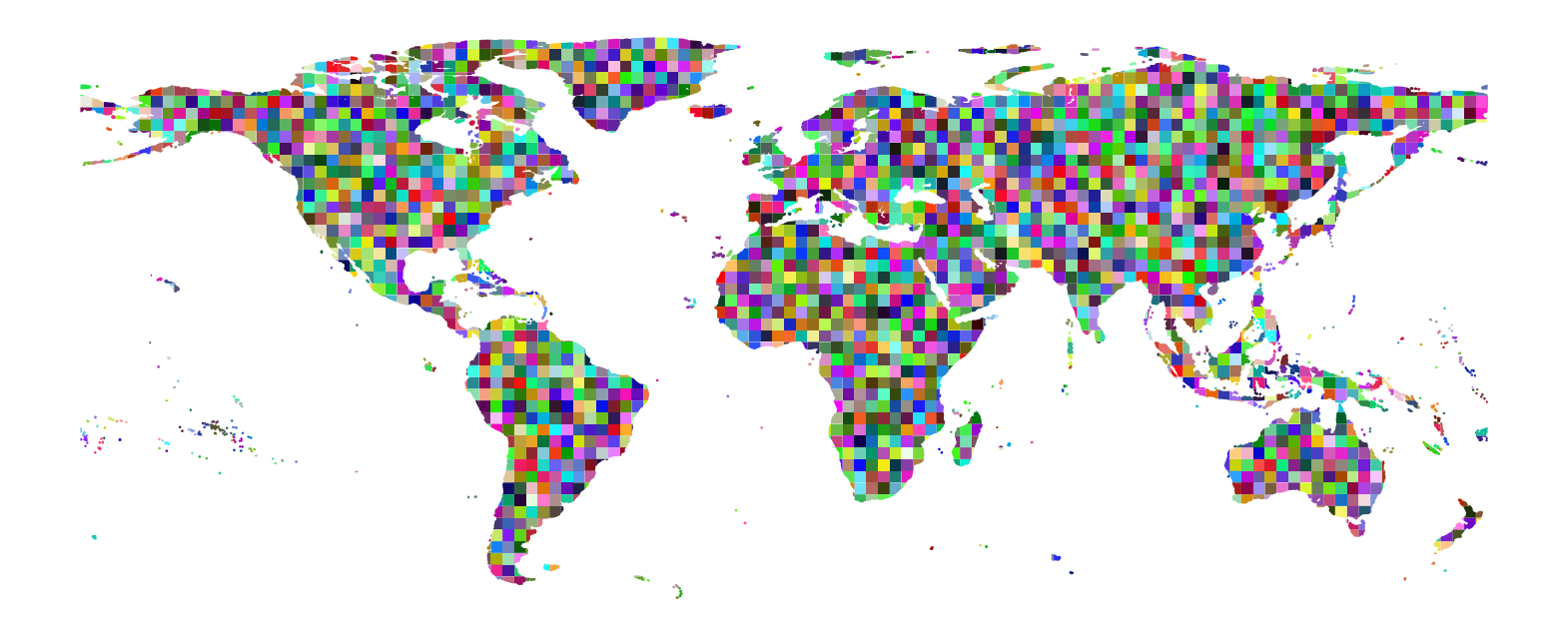
此次试验的研究范围为伦敦区域，为了能够下载指定区域的 WorldCover 数据，需要先下载 WorldCover 的瓦片网格，由指定区域提取对应的瓦片信息后，确定下载文件路径进行下载。 WorldCover数据的详细说明和下载代码示例可以从其[产品使用手册（Product User Manual）](https://github.com/microsoft/PlanetaryComputerExamples/blob/main/datasets/esa-worldcover/esa-worldcover-example.ipynb)③中获取。

%load\_ext autoreload   
%autoreload 2   
%reload\_ext autoreload  
import usda.models as usda\_models  
import usda.geodata\_process as usda\_geodataProcess # conda install gdal; pip install geopandas;  
import usda.datasets as usda\_datasets  
import usda.data\_visual as usda\_datavisual  
  
import matplotlib.pyplot as plt  
import pystac\_client  
import planetary\_computer  
import rich.table  
import rioxarray as rxr  
import os  
import rasterio as rio  
import matplotlib as mpl  
from rasterio.plot import show  
import copy  
from IPython.display import Image,display  
  
import geopandas as gpd  
import mapclassify  
import pickle  
  
from torchvision import transforms  
import torch  
from sklearn.model\_selection import train\_test\_split  
from torch.utils.data import TensorDataset, DataLoader  
import torchvision  
import numpy as np  
  
import pytorch\_lightning as pl  
from pytorch\_lightning.callbacks import LearningRateMonitor, ModelCheckpoint  
from pytorch\_lightning.loggers import TensorBoardLogger

* 从欧空局下载 WorldCover 数据

下载 WorldCover 瓦片网格，并转换为 GeoDataFrame 数据格式。为了直观查看瓦片网格的全球分布情况，定义一个随机的颜色映射打印显示地图。定义esa\_worldcover\_2020\_grid\_downloading()函数实现。

wc\_grid,ranodm\_cmap=usda\_datasets.esa\_worldcover\_2020\_grid\_downloading(show=True)



查看 WorldCover 瓦片网格，其字段包含一个几何对象和一个 ID 索引。

wc\_grid.tail(3)

|  | **geometry** | **ll\_tile** |
| --- | --- | --- |
| **2628** | POLYGON ((-144.00000 -27.66528, -144.00530 -27... | S30W147 |
| **2629** | POLYGON ((-177.00000 -44.08407, -177.02234 -44... | S45W180 |
| **2630** | POLYGON ((-177.00000 -43.54951, -176.99759 -43... | S45W177 |

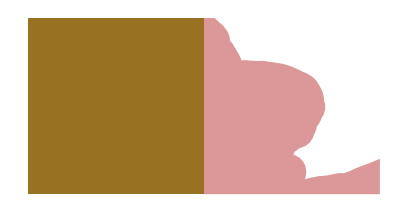
定义esa\_worldcover\_downloading()函数实现根据指定区域(左下角经度，左下角纬度，右上角经度，右上角纬度)下载 WorldCover 数据。

london\_bounds=(-1,51.15,1,52)  
save\_root=r'I:\data\ESA\_London'  
  
url\_lst,fns,tiles=usda\_datasets.esa\_worldcover\_downloading(london\_bounds,save\_root,y='2020')  
print(url\_lst)

100%|█████████████████████████████████████████████████████████████████████████| 2/2 [03:26<00:00, 103.19s/it]  
  
['https://esa-worldcover.s3.eu-central-1.amazonaws.com/v100/2020/map/ESA\_WorldCover\_10m\_2020\_v100\_N51E000\_Map.tif', 'https://esa-worldcover.s3.eu-central-1.amazonaws.com/v100/2020/map/ESA\_WorldCover\_10m\_2020\_v100\_N51W003\_Map.tif']

查看返回的 WorldCover 瓦片区域。

fig, ax=plt.subplots(figsize=(5,5))  
tiles.plot(column='ll\_tile',cmap=ranodm\_cmap,ax=ax)  
fig.patch.set\_visible(False)  
ax.axis('off')  
plt.show()



* 从[Planetary Computer Hub（PcHub）](https://github.com/microsoft/PlanetaryComputerExamples/blob/main/datasets/esa-worldcover/esa-worldcover-example.ipynb)④下载 WorldCover 数据或查看信息[1]

根据指定的范围搜索 WorldCover 数据，返回对应瓦片的 ID。

也可以从提供的数据手册中获得信息。

catalog=pystac\_client.Client.open("https://planetarycomputer.microsoft.com/api/stac/v1",modifier=planetary\_computer.sign\_inplace,)  
search=catalog.search(collections=["esa-worldcover"],bbox=london\_bounds)  
items=list(search.get\_items())  
items

[<Item id=ESA\_WorldCover\_10m\_2021\_v200\_N51W003>,  
 <Item id=ESA\_WorldCover\_10m\_2021\_v200\_N51E000>,  
 <Item id=ESA\_WorldCover\_10m\_2020\_v100\_N51W003>,  
 <Item id=ESA\_WorldCover\_10m\_2020\_v100\_N51E000>]

查看数据属性相关信息。

# Metadata  
t\_metadata = rich.table.Table("Key", "Value")  
for k, v in sorted(items[0].properties.items()):  
 t\_metadata.add\_row(k, str(v))  
t\_metadata

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓

┃ **Key** ┃ **Value** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩

│ created │ 2023-04-06T16:31:45.847838Z │

│ datetime │ None │

│ description │ ESA WorldCover product at 10m resolution │

│ end\_datetime │ 2021-12-31T23:59:59Z │

│ esa\_worldcover:product\_tile │ N51W003 │

│ esa\_worldcover:product\_version │ 2.0.0 │

│ grid:code │ ESAWORLDCOVER-N51W003 │

│ instruments │ ['c-sar', 'msi'] │

│ mission │ sentinel-1, sentinel-2 │

│ platform │ sentinel-1a, sentinel-1b, sentinel-2a, sentinel-2b │

│ proj:epsg │ 4326 │

│ start\_datetime │ 2021-01-01T00:00:00Z │

└────────────────────────────────┴──────────────────────────────────

从数据集中获取分类信息和类别颜色配置值。

class\_list=items[0].assets["map"].extra\_fields["classification:classes"]  
classmap={c["value"]: {"description": c["description"], "hex": c["color-hint"]} for c in class\_list}  
t=rich.table.Table("Value", "Description", "Hex Color")  
for k, v in classmap.items():  
 t.add\_row(str(k), v["description"], v["hex"])  
t

┏━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━┓

┃ **Value** ┃ **Description** ┃ **Hex Color** ┃

┡━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━┩

│ 10 │ Tree cover │ 006400 │

│ 20 │ Shrubland │ FFBB22 │

│ 30 │ Grassland │ FFFF4C │

│ 40 │ Cropland │ F096FF │

│ 50 │ Built-up │ FA0000 │

│ 60 │ Bare / sparse vegetation │ B4B4B4 │

│ 70 │ Snow and ice │ F0F0F0 │

│ 80 │ Permanent water bodies │ 0064C8 │

│ 90 │ Herbaceous wetland │ 0096A0 │

│ 95 │ Mangroves │ 00CF75 │

│ 100 │ Moss and lichen │ FAE6A0 │

└───────┴──────────────────────────┴───────────┘

建立用于土地覆盖地图打印的相关信息，包括分类颜色的映射cmap、用于图例的分类值values、边界boundaries、刻度ticks和标签tick\_labels等。定义worldcover\_cmap4plot()函数完成计算。

cmap,values,boundaries,ticks,tick\_labels=usda\_datavisual.worldcover\_cmap4plot()  
cmap



打开从欧空局下载的 WorldCover 数据，示例中仅打开了一个瓦片。

Fns

['I:\\data\\ESA\_London\\ESA\_WorldCover\_10m\_2020\_v100\_N51E000\_Map.tif',  
 'I:\\data\\ESA\_London\\ESA\_WorldCover\_10m\_2020\_v100\_N51W003\_Map.tif']

london\_esa=rxr.open\_rasterio(fns[1],masked=True).squeeze()  
london\_esa

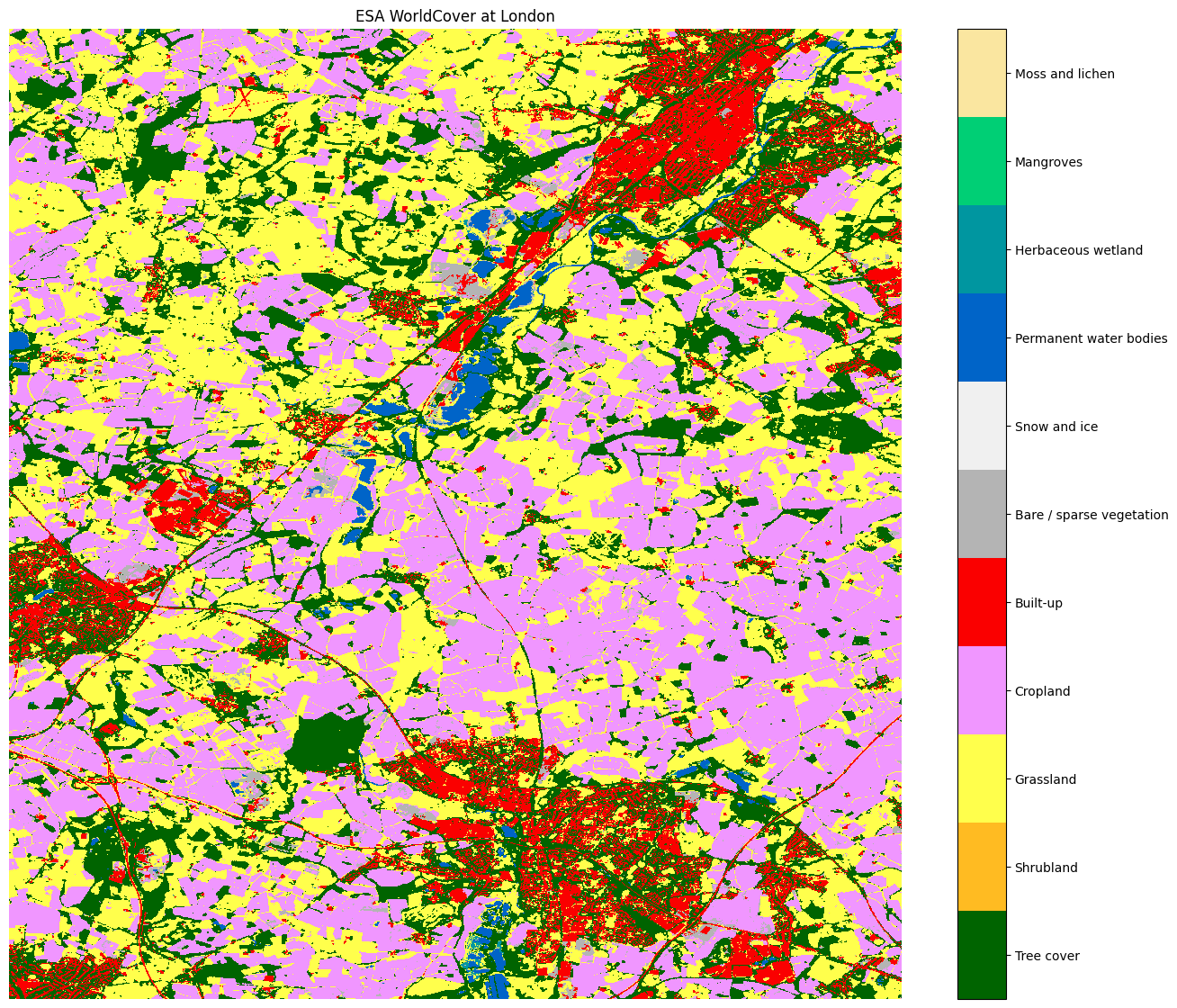
<xarray.DataArray (y: 36000, x: 36000)>  
[1296000000 values with dtype=float32]  
Coordinates:  
 band int32 1  
 \* x (x) float64 -3.0 -3.0 -3.0 ... -0.0002083 -0.000125 -4.167e-05  
 \* y (y) float64 54.0 54.0 54.0 54.0 54.0 ... 51.0 51.0 51.0 51.0  
 spatial\_ref int32 0  
Attributes: (12/18)  
 algorithm\_version: V1.0.0  
 AREA\_OR\_POINT: Area  
 copyright: ESA WorldCover project 2020 / Contains modified Cope...  
 creation\_time: 2021-10-12 16:51:26.705020  
 legend: 10 Tree cover\n 20 Shrubland\n 30 Grassland...  
 license: CC-BY 4.0 - https://creativecommons.org/licenses/by/...  
 ... ...  
 reference: https://esa-worldcover.org  
 time\_end: 2020-12-31T23:59:59Z  
 time\_start: 2020-01-01T00:00:00Z  
 title: ESA WorldCover product at 10m resolution for year 2020  
 scale\_factor: 1.0  
 add\_offset: 0.0

london\_esa.rio.crs

CRS.from\_epsg(4326)

由返回的打印相关信息打印 WorldCover 土地覆盖分类地图。

fig, ax = plt.subplots(figsize=(16, 14))  
normalizer = matplotlib.colors.Normalize(vmin=0, vmax=255)  
  
xy\_0=14000  
delta=3000  
london\_esa.isel(y=slice(xy\_0, xy\_0+delta), x=slice(xy\_0,xy\_0+delta)).plot(ax=ax, cmap=cmap, norm=normalizer)  
  
colorbar=fig.colorbar(  
 cm.ScalarMappable(norm=normalizer, cmap=cmap),  
 boundaries=boundaries,  
 values=values,  
 cax=fig.axes[1].axes,  
)  
colorbar.set\_ticks(ticks, labels=tick\_labels)  
  
ax.set\_axis\_off()  
ax.set\_title("ESA WorldCover at London");



#### 2) 地表温度（Land Surface Temperature，LST）数据

关于地表温度（LST）数据信息可以查看*标记距离*一章，这里从 Planetary Computer Hub 处下载。下载前，通过搜索给定下载区域和数据时段获得满足要求的数据 ID 等相关信息。因为 WorldCover 数据使用的为2020年，因此 LST 选择了同一年份，但仅使用了8月份的数据。

time\_range = "2020-08-07/2020-08-08"  
  
catalog=pystac\_client.Client.open("https://planetarycomputer.microsoft.com/api/stac/v1",modifier=planetary\_computer.sign\_inplace,)  
search=catalog.search(collections=["modis-11A2-061"],bbox=london\_bounds,datetime=time\_range)  
  
items=search.get\_all\_items()  
list(items)

[<Item id=MYD11A2.A2020217.h18v03.061.2021013233159>,  
 <Item id=MYD11A2.A2020217.h17v03.061.2021013233157>,  
 <Item id=MOD11A2.A2020217.h18v03.061.2021013224226>,  
 <Item id=MOD11A2.A2020217.h17v03.061.2021013223600>]

查看 LST 数据属性信息，如下。

items[0].properties

{'created': '2021-01-13T23:31:59Z',  
 'updated': '2021-01-13T17:38:42.615000Z',  
 'datetime': None,  
 'platform': 'aqua',  
 'proj:epsg': None,  
 'proj:wkt2': 'PROJCS["unnamed",GEOGCS["Unknown datum based upon the custom spheroid",DATUM["Not specified (based on custom spheroid)",SPHEROID["Custom spheroid",6371007.181,0]],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]]],PROJECTION["Sinusoidal"],PARAMETER["longitude\_of\_center",0],PARAMETER["false\_easting",0],PARAMETER["false\_northing",0],UNIT["Meter",1],AXIS["Easting",EAST],AXIS["Northing",NORTH]]',  
 'proj:shape': [1200, 1200],  
 'instruments': ['modis'],  
 'end\_datetime': '2020-08-11T23:59:59Z',  
 'modis:tile-id': '51018003',  
 'proj:geometry': {'type': 'Polygon',  
 'coordinates': [[[1111950.519767, 5559752.598833],  
 [1111950.519767, 6671703.118599],  
 [0.0, 6671703.118599],  
 [0.0, 5559752.598833],  
 [1111950.519767, 5559752.598833]]]},  
 'proj:transform': [926.6254331391667,  
 0.0,  
 0.0,  
 0.0,  
 -926.6254331383334,  
 6671703.118599],  
 'start\_datetime': '2020-08-04T00:00:00Z',  
 'modis:vertical-tile': 3,  
 'modis:horizontal-tile': 18}

查看 LST 数据层信息，如下。其中，LST\_Day\_1km为地表温度数据层。

table = rich.table.Table("Asset Key", "Description")  
for asset\_key, asset in items[0].assets.items():  
 table.add\_row(asset\_key, asset.title)  
  
table

┏━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓

┃ **Asset Key** ┃ **Description** ┃

┡━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩

│ hdf │ Source data containing all bands │

│ QC\_Day │ Quality control for daytime LST and emissivity │

│ Emis\_31 │ Band 31 emissivity │

│ Emis\_32 │ Band 32 emissivity │

│ QC\_Night │ Quality control for nighttime LST and emissivity │

│ metadata │ Federal Geographic Data Committee (FGDC) Metadata │

│ LST\_Day\_1km │ 8-day daytime 1km grid Landsurface Temperature │

│ Day\_view\_angl │ Average view zenith angle of daytime Land-surface Temperature │

│ Day\_view\_time │ Average time of daytime Landsurface Temperature observation │

│ LST\_Night\_1km │ 8-day nighttime 1km grid Landsurface Temperature │

│ Clear\_sky\_days │ the days in clear-sky conditions and with validate LSTs │

│ Night\_view\_angl │ View zenith angle of nighttime Land-surface Temperature │

│ Night\_view\_time │ Average view zenith angle of nighttime Land-surface Temperature │

│ Clear\_sky\_nights │ the nights in clear-sky conditions and with validate LSTs │

│ tilejson │ TileJSON with default rendering │

│ rendered\_preview │ Rendered preview │

└──────────────────┴─────────────────────────────────────────────────────────────

打印 LST 数据，确定为所选区域的数据。

print(items[1].assets.keys())  
im\_01=Image(url=items[1].assets["rendered\_preview"].href,width=500)  
im\_02=Image(url=items[0].assets["rendered\_preview"].href,width=500)  
  
display(im\_01, im\_02)

dict\_keys(['hdf', 'QC\_Day', 'Emis\_31', 'Emis\_32', 'QC\_Night', 'metadata', 'LST\_Day\_1km', 'Day\_view\_angl', 'Day\_view\_time', 'LST\_Night\_1km', 'Clear\_sky\_days', 'Night\_view\_angl', 'Night\_view\_time', 'Clear\_sky\_nights', 'tilejson', 'rendered\_preview'])





通过查看LST\_Day\_1km层数据，可以发现字段href提供了下载地址，根据该地址下载数据。

items[0].assets['LST\_Day\_1km']

href "https://modiseuwest.blob.core.windows.net/modis-061-cogs/MYD11A2/18/03/2020217/MYD11A2.A2020217.h18v03.061.2021013233159\_LST\_Day\_1km.tif?st=2023-07-06T04%3A10%3A39Z&se=2023-07-07T04%3A55%3A39Z&sp=rl&sv=2021-06-08&sr=c&skoid=c85c15d6-d1ae-42d4-af60-e2ca0f81359b&sktid=72f988bf-86f1-41af-91ab-2d7cd011db47&skt=2023-07-07T00%3A49%3A38Z&ske=2023-07-14T00%3A49%3A38Z&sks=b&skv=2021-06-08&sig=AReev5kZKP4CyN3Jj0oQsCU3lvNp%2B8puqGbTreSdDVA%3D"  
 type "image/tiff; application=geotiff; profile=cloud-optimized"  
 title "8-day daytime 1km grid Landsurface Temperature"  
 raster:bands [] 1 items  
 0  
 unit "Kelvin"  
 scale 0.02  
 data\_type "uint16"  
 spatial\_resolution 1000  
 roles [] 2 items  
 0 "data"  
 1 "temperature"

url\_lst=[]  
for item in items:  
 url\_lst.append(item.assets['LST\_Day\_1km'].href)  
save\_root=r'E:\data\LST\LST\_London'  
fns=[]  
i=0  
for url in tqdm(url\_lst):   
 r = requests.get(url, allow\_redirects=True)  
 out\_fn = os.path.join(save\_root,f"LST\_{i}.tif")  
 fns.append(out\_fn)  
 with open(out\_fn, 'wb') as f:  
 f.write(r.content)  
 i+=1

100%|██████████████████████████████████████████████████████████████████████████| 4/4 [02:15<00:00, 33.92s/it]

使用USDA库的raster\_mosaic()函数合并下载的 LST 数据。

fns=['E:\\data\\LST\\LST\_London\\LST\_0.tif',  
 'E:\\data\\LST\\LST\_London\\LST\_1.tif',  
 'E:\\data\\LST\\LST\_London\\LST\_2.tif',  
 'E:\\data\\LST\\LST\_London\\LST\_3.tif']  
  
lst\_mosaic\_fn=os.path.join(save\_root,'lst\_london.tif')  
usda\_geodataProcess.raster\_mosaic(r'E:\data\LST\LST\_London\LST\_london',lst\_mosaic\_fn)

Affine(926.6254331391667, 0.0, -1111950.519767,  
 0.0, -926.6254331383334, 6671703.118599)

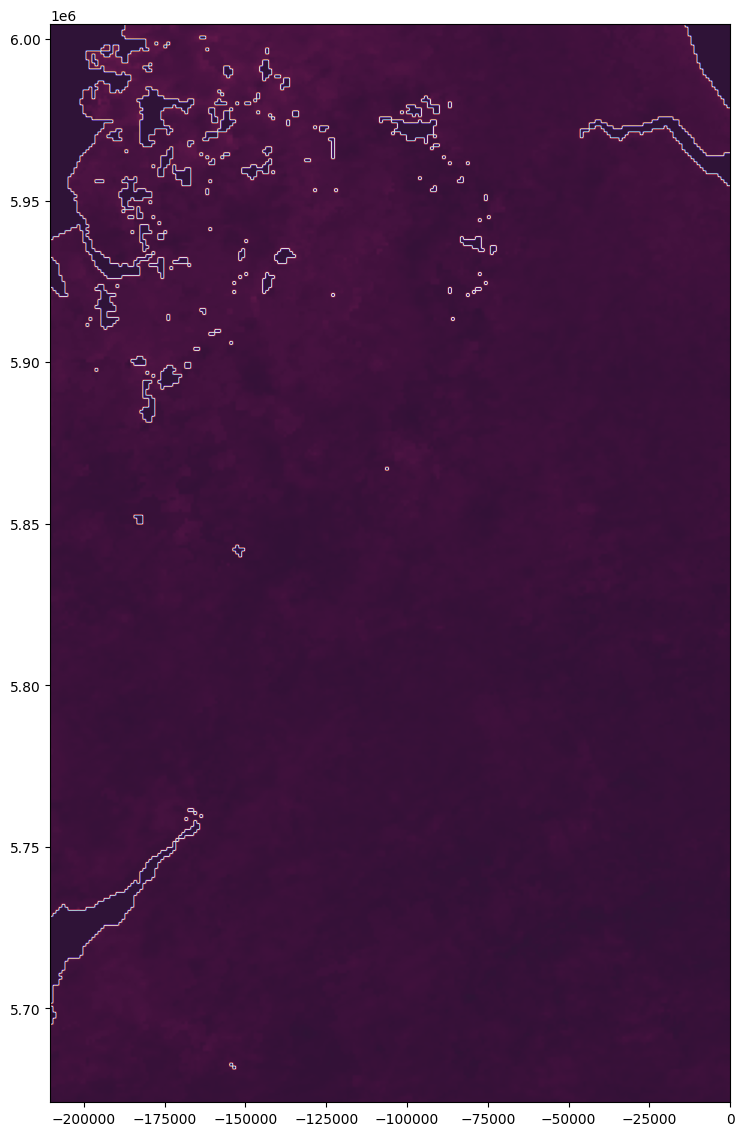
根据 WorldCover 一个瓦片的边界读取合并的 LST 数据。因为 WorldCover 和 LST 的投影不同，因此使用pt\_coordi\_transform方法转换统一边界坐标。

london\_esa\_crs=london\_esa.rio.crs  
london\_esa\_bounds=london\_esa.rio.bounds()  
print(london\_esa\_crs,london\_esa\_bounds)  
  
with rio.open(lst\_mosaic\_fn) as src:   
 epsg\_LST=src.crs   
  
LST\_pt\_leftBottom\_pj=usda\_geodataProcess.pt\_coordi\_transform(london\_esa\_crs,epsg\_LST,london\_esa\_bounds[:2])  
LST\_pt\_rightTop\_pj=usda\_geodataProcess.pt\_coordi\_transform(london\_esa\_crs,epsg\_LST,london\_esa\_bounds[2:])  
  
LST\_temp,LST\_transform,LST\_ras\_meta=usda\_geodataProcess.rio\_read\_subset(lst\_mosaic\_fn,[LST\_pt\_leftBottom\_pj,LST\_pt\_rightTop\_pj])

EPSG:4326 (-3.0, 51.0, -5.0923620788928536e-17, 54.0)

查看 LST 数据。

lst\_cmap=mpl.cm.twilight\_shifted  
f, ax=plt.subplots(figsize=(16,14))  
show(LST\_temp,ax=ax,transform=LST\_transform,cmap=lst\_cmap)   
plt.show()



将按照边界提取的 LST 数据单独保存为一个栅格文件。

LST\_London\_fn=r'I:\data\london\LST\_London.tif'  
LST\_ras\_meta\_=copy.deepcopy(LST\_ras\_meta)  
LST\_ras\_meta\_.update(   
 compress='lzw',  
 )   
with rio.open(LST\_London\_fn,'w',\*\*LST\_ras\_meta\_) as dst:  
 dst.write(LST\_temp)

将 LST 栅格数据转化为以各个栅格单元为多边形对象的矢量文件，定义rastercells2shp()函数实现。

rasterfn =r'I:\data\london\LST\_London.tif'  
aa=rxr.open\_rasterio(rasterfn)  
aa.rio.crs

CRS.from\_wkt('PROJCS["unnamed",GEOGCS["Unknown datum based upon the custom spheroid",DATUM["Not specified (based on custom spheroid)",SPHEROID["Custom spheroid",6371007.181,0]],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]]],PROJECTION["Sinusoidal"],PARAMETER["longitude\_of\_center",0],PARAMETER["false\_easting",0],PARAMETER["false\_northing",0],UNIT["metre",1,AUTHORITY["EPSG","9001"]],AXIS["Easting",EAST],AXIS["Northing",NORTH]]')

rasterfn =r'I:\data\london\LST\_London.tif'  
outSHPfn = r'I:\data\london\LST\_London.shp'   
  
LST\_gpd=usda\_geodataProcess.rastercells2shp(rasterfn,outSHPfn)

LST\_gpd

|  | **geometry** | **vals** | **x** | **y** |
| --- | --- | --- | --- | --- |
| **0** | POLYGON ((-209417.348 6004532.807, -209417.348... | 0 | -209880.660606 | 6.004069e+06 |
| **1** | POLYGON ((-208490.722 6004532.807, -208490.722... | 0 | -208954.035173 | 6.004069e+06 |
| **2** | POLYGON ((-207564.097 6004532.807, -207564.097... | 0 | -208027.409740 | 6.004069e+06 |
| **3** | POLYGON ((-206637.472 6004532.807, -206637.472... | 0 | -207100.784307 | 6.004069e+06 |
| **4** | POLYGON ((-205710.846 6004532.807, -205710.846... | 0 | -206174.158873 | 6.004069e+06 |
| **...** | ... | ... | ... | ... |
| **81715** | POLYGON ((-3706.502 5671874.276, -3706.502 567... | 15367 | -4169.814449 | 5.671411e+06 |
| **81716** | POLYGON ((-2779.876 5671874.276, -2779.876 567... | 15365 | -3243.189016 | 5.671411e+06 |
| **81717** | POLYGON ((-1853.251 5671874.276, -1853.251 567... | 15381 | -2316.563583 | 5.671411e+06 |
| **81718** | POLYGON ((-926.625 5671874.276, -926.625 56709... | 15388 | -1389.938150 | 5.671411e+06 |
| **81719** | POLYGON ((-0.000 5671874.276, -0.000 5670947.6... | 15358 | -463.312717 | 5.671411e+06 |

81720 rows × 4 columns

### 3.6.1.2 构建数据集和训练模型

#### 1) 构建数据集

数据集的构建包括两个大部分，一个是建立地表温度分类标签；另一个是根据 LST 栅格单元将 WorldCover 切分为一个个样方，每个样方为一个土地覆盖类别矩阵，且对应一个 LST 标签。

* 建立地表温度分类标签

LST 为一个连续型数据，将其离散为分类数据，分类的多种方式可以参看*标记距离*一章。下述分别查看了BoxPlot、JenksCaspall和EqualInterval等三种方法，并写入到 LST （gdf）数据中。

BoxPlot分类方法。

outSHPfn = r'I:\data\london\LST\_London.shp'  
LST\_gdf=gpd.read\_file(outSHPfn)  
LST\_gdf

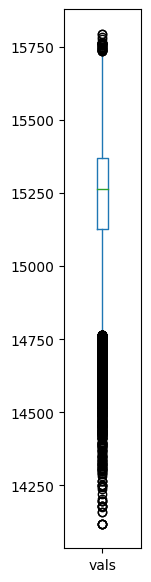
|  | **vals** | **x** | **y** | **geometry** |
| --- | --- | --- | --- | --- |
| **0** | 0 | -209880.660606 | 6.004069e+06 | POLYGON ((-209417.348 6004532.807, -209417.348... |
| **1** | 0 | -208954.035173 | 6.004069e+06 | POLYGON ((-208490.722 6004532.807, -208490.722... |
| **2** | 0 | -208027.409740 | 6.004069e+06 | POLYGON ((-207564.097 6004532.807, -207564.097... |
| **3** | 0 | -207100.784307 | 6.004069e+06 | POLYGON ((-206637.472 6004532.807, -206637.472... |
| **4** | 0 | -206174.158873 | 6.004069e+06 | POLYGON ((-205710.846 6004532.807, -205710.846... |
| **...** | ... | ... | ... | ... |
| **81715** | 15367 | -4169.814449 | 5.671411e+06 | POLYGON ((-3706.502 5671874.276, -3706.502 567... |
| **81716** | 15365 | -3243.189016 | 5.671411e+06 | POLYGON ((-2779.876 5671874.276, -2779.876 567... |
| **81717** | 15381 | -2316.563583 | 5.671411e+06 | POLYGON ((-1853.251 5671874.276, -1853.251 567... |
| **81718** | 15388 | -1389.938150 | 5.671411e+06 | POLYGON ((-926.625 5671874.276, -926.625 56709... |
| **81719** | 15358 | -463.312717 | 5.671411e+06 | POLYGON ((-0.000 5671874.276, -0.000 5670947.6... |

81720 rows × 4 columns

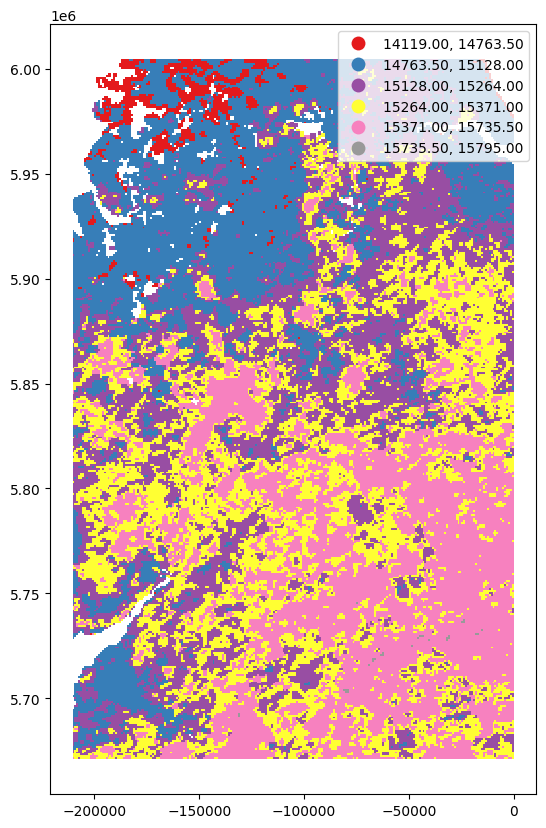
LST\_gdf.drop(LST\_gdf[LST\_gdf['vals']==0].index,inplace=True)  
  
LST\_BoxPlot=mapclassify.BoxPlot(LST\_gdf['vals'])  
LST\_gdf['rank\_boxplot']=LST\_BoxPlot.yb  
LST\_BoxPlot

BoxPlot  
  
 Interval Count  
----------------------------  
[14119.00, 14763.50] | 1711  
(14763.50, 15128.00] | 18031  
(15128.00, 15264.00] | 19775  
(15264.00, 15371.00] | 19589  
(15371.00, 15735.50] | 19573  
(15735.50, 15795.00] | 31

LST\_gdf.boxplot(column=['vals'],grid=False,figsize=(1,7));



LST\_gdf.plot('vals',cmap='Set1',figsize=(10,10),scheme='BoxPlot',legend=True);

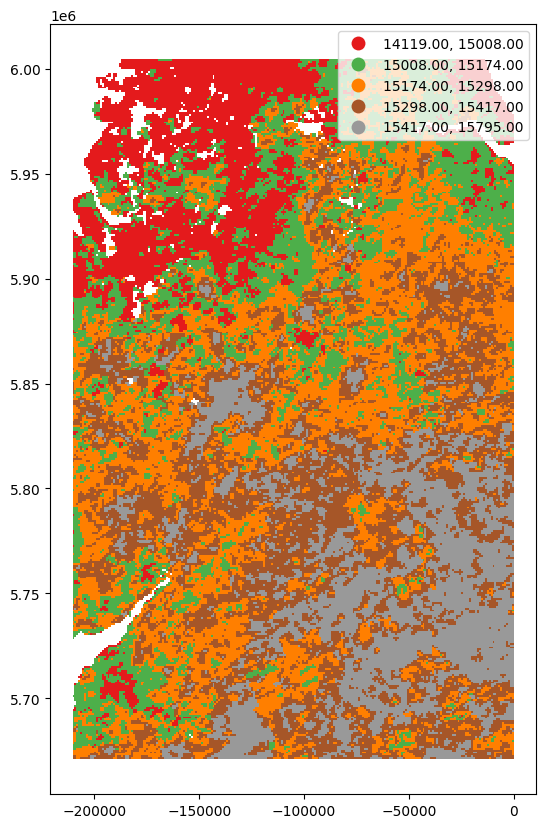


JenksCaspall分类方法。

LST\_JenksCaspall=mapclassify.JenksCaspall(LST\_gdf['vals'],k=5)  
LST\_gdf['rank\_JenksCaspall']=LST\_JenksCaspall.yb  
LST\_JenksCaspall

JenksCaspall  
  
 Interval Count  
----------------------------  
[14119.00, 15008.00] | 10686  
(15008.00, 15174.00] | 14119  
(15174.00, 15298.00] | 21338  
(15298.00, 15417.00] | 19622  
(15417.00, 15795.00] | 12945

LST\_gdf.plot('vals',cmap='Set1',figsize=(10,10),k=5,scheme='jenkscaspall',legend=True);

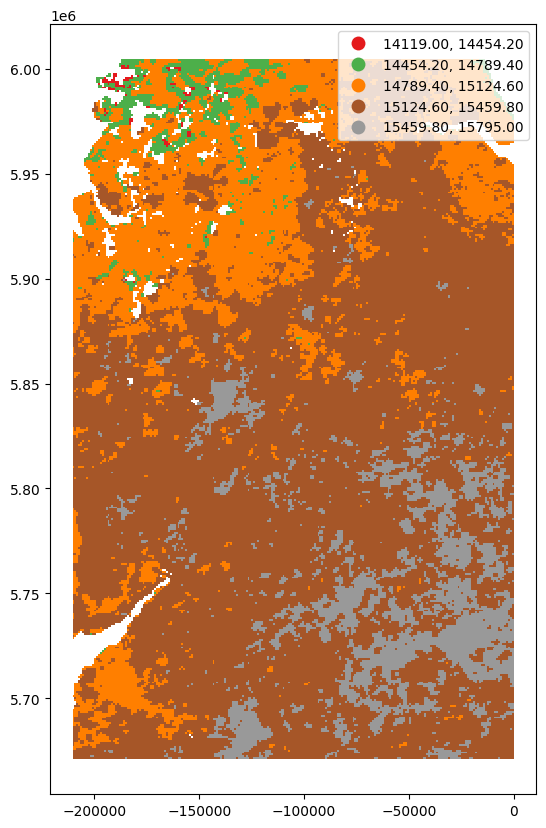


EqualInterval分类方法。

LST\_EqualInterval=mapclassify.EqualInterval(LST\_gdf['vals'],k=5)   
LST\_gdf['rank\_EqualInterval']=LST\_EqualInterval.yb  
LST\_EqualInterval

EqualInterval  
  
 Interval Count  
----------------------------  
[14119.00, 14454.20] | 136  
(14454.20, 14789.40] | 1969  
(14789.40, 15124.60] | 17270  
(15124.60, 15459.80] | 50970  
(15459.80, 15795.00] | 8365

LST\_gdf.plot('vals',cmap='Set1',figsize=(10,10),k=5,scheme='EqualInterval',legend=True);



查看最终数据，确定正确后写入本地磁盘。

LST\_gdf.tail(3)

|  | **vals** | **x** | **y** | **geometry** | **rank\_boxplot** | **rank\_JenksCaspall** | **rank\_EqualInterval** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **81717** | 15381 | -2316.563583 | 5.671411e+06 | POLYGON ((-1853.251 5671874.276, -1853.251 567... | 4 | 3 | 3 |
| **81718** | 15388 | -1389.938150 | 5.671411e+06 | POLYGON ((-926.625 5671874.276, -926.625 56709... | 4 | 3 | 3 |
| **81719** | 15358 | -463.312717 | 5.671411e+06 | POLYGON ((-0.000 5671874.276, -0.000 5670947.6... | 3 | 3 | 3 |

LST\_rank\_fn=r'I:\data\london\LST\_rank.gpkg'  
LST\_gdf.to\_file(LST\_rank\_fn,driver='GPKG',layer='lst\_rank')

* 根据 LST 栅格单元将 WorldCover 切分为一个个样方

用 LST 建立的切分样方单元切分栅格数据时，切分后的样方长宽栅格单元数可能不同，因此先随机采样切分 1000 份后，查看长宽大小的频数再确定最终样方大小。从结果来看，纬度1，即样方的高度基本为101个栅格单元；纬度2，即样方的宽度分布数的变化相对较多，但基本在160个以上。因为切分样方可能与被切分的栅格数据之间并不一定完全水平或垂直吻合，因此切分后的样方单元可能存在空值，所以选择高度大小为90，而宽度大小为150的配置，并通过配置偏移，移除周边可能为空值的栅格单元。上述计算中，定义xy\_size\_elevation()函数查看切分后的样方大小；定义build\_clipped\_raster\_dataset()函数，切分栅格建立数据集。

clipper\_fn=r'I:\data\london\LST\_rank.gpkg'   
raster\_fn=r'I:\\data\\ESA\_London\\ESA\_WorldCover\_10m\_2020\_v100\_N51W003\_Map.tif'  
  
usda\_geodataProcess.xy\_size\_elevation(clipper\_fn,raster\_fn);

0 1 2   
1 101 169 100  
 171 95  
 168 88  
 165 85  
 167 81  
 166 79  
 170 78  
 164 76  
 172 68  
 163 48  
 173 46  
 174 40  
 162 29  
 175 25  
 161 22  
 176 15  
 177 7  
 178 2  
 40 1  
 160 1  
 123 1  
 47 1  
 46 1  
 100 173 1  
Name: count, dtype: int64

前文将 LST 栅格数据单元矢量化后写入为 GPKG 数据格式（如果为 SHP 格式数据时，配置的字段较长时，可能会被切断，例如rank\_JenksCaspall字段被切短为rank\_Jenks）。构建数据集的函数使用多线程计算，可以配置ratio\_cpu参数确定 CPU 的使用率；ratio\_split参数影响数据集的批量大小，批量大小为使用的 CPU 数和ratio\_split参数之积。

在定义build\_clipped\_raster\_dataset()函数时，考虑到有些 LST 字段可能被用到，例如纬度列y，其温度与纬度有一定关系。因此参数y\_columns定义为一个列表，可以输入多个字段，以第一个字段为默认的值，其它字段数据写入到extra键下。

dataset\_save\_fn='I:\data\london\LC2LST\_dataset\_rank\_JenksCaspall.pickle'  
usda\_geodataProcess.build\_clipped\_raster\_dataset(clipper\_fn,raster\_fn,dataset\_save\_fn,['rank\_JenksCaspall','y'],90,150,5,10,ratio\_cpu=0.7,ratio\_split=30)

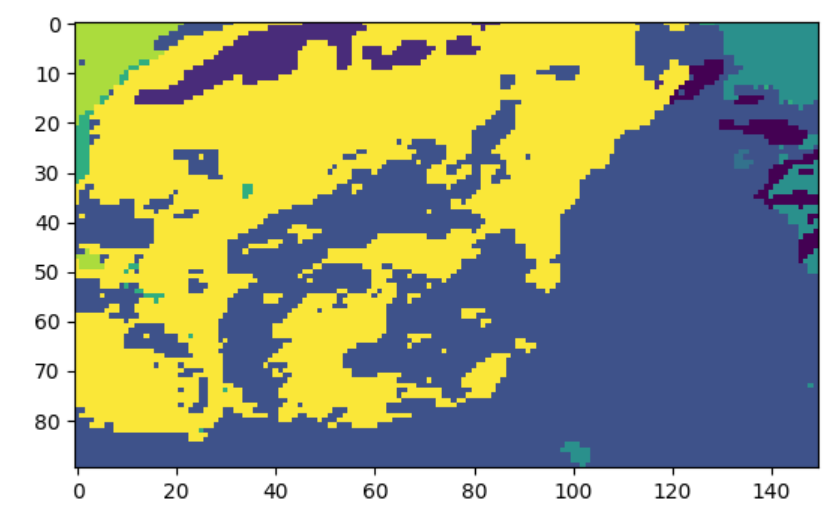
读取查看建立的数据集。

with open(dataset\_save\_fn,'rb') as f:  
 ds=pickle.load(f)

print(ds.data.shape,ds.target.shape,ds.extra.shape)  
sample=ds.data[0]  
print(f'data shape:{sample.shape};\ndata:\n{sample};\ntarget={ds.target[0]}')

(76683, 1, 90, 150) (76683,) (76683, 1)  
data shape:(1, 90, 150);  
data:  
[[[80 80 80 ... 50 50 50]  
 [80 80 80 ... 50 50 50]  
 [80 80 80 ... 50 50 50]  
 ...  
 [30 30 30 ... 30 30 30]  
 [30 30 30 ... 30 30 30]  
 [30 30 30 ... 30 30 30]]];  
target=0

plt.imshow(sample[0]);



同样的方式，建立BoxPlot和EqualInterval分类方式的数据集。

usda\_geodataProcess.build\_clipped\_raster\_dataset(clipper\_fn,raster\_fn,r'I:\data\london\LC2LST\_dataset\_rank\_EqualInterval.pickle',['rank\_EqualInterval','y'],90,150,5,10,ratio\_cpu=0.7,ratio\_split=30)   
usda\_geodataProcess.build\_clipped\_raster\_dataset(clipper\_fn,raster\_fn,r'I:\data\london\LC2LST\_dataset\_rank\_boxplot.pickle',['rank\_boxplot','y'],90,150,5,10,ratio\_cpu=0.7,ratio\_split=30)

* 建立数据集加载器，与查看进一步将样方数据切分为连续的图块矩阵（patch）

以EqualInterval方式为例。

# dataset\_sel\_save\_fn=r'I:\data\london\LC2LST\_dataset\_rank\_JenksCaspall.pickle'  
# dataset\_sel\_save\_fn=r'I:\data\london\LC2LST\_dataset\_rank\_boxplot.pickle'  
dataset\_sel\_save\_fn=r'I:\data\london\LC2LST\_dataset\_rank\_EqualInterval.pickle'  
  
with open(dataset\_sel\_save\_fn,'rb') as f:  
 ds=pickle.load(f)  
ds\_unique\_target=np.unique(ds.target)  
ds\_unique\_data=np.unique(ds.data)  
print(f'ds\_unique\_target:{ds\_unique\_target};\nnum\_classes={len(ds\_unique\_target)};\nds\_unique\_data:{ds\_unique\_data}\nmax\_data={max(ds\_unique\_data)}')

ds\_unique\_target:[0 1 2 3 4];  
num\_classes=5;  
ds\_unique\_data:[ 0 10 20 30 40 50 60 80 90]  
max\_data=90

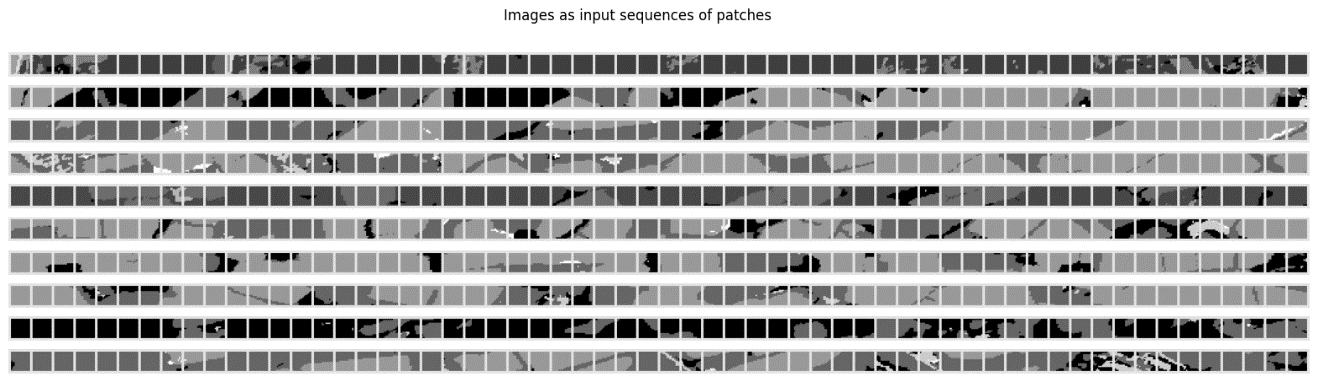
将数据集切分为训练和测试（验证）数据集，并通过除以大于或等于分类最大值将值缩放到[0,1]区间。并建立数据集加载器。

x\_train, x\_test, y\_train, y\_test = train\_test\_split(ds.data, ds.target.astype(int),test\_size=0.2)  
x\_train, x\_test, y\_train, y\_test=torch.Tensor(x\_train), torch.Tensor(x\_test), torch.Tensor(y\_train).type(torch.LongTensor), torch.Tensor(y\_test).type(torch.LongTensor)   
x\_train=x\_train/100.  
x\_test=x\_test/100.  
  
train\_ds=TensorDataset(x\_train,y\_train)  
train\_dl=DataLoader(train\_ds,batch\_size=128, shuffle=True, drop\_last=True, pin\_memory=True, num\_workers=4)  
  
test\_ds=TensorDataset(x\_test,y\_test)  
test\_dl=DataLoader(test\_ds,batch\_size=128, shuffle=False, drop\_last=False, num\_workers=4)

查看将样方数据切分为连续的图块矩阵。切分的单个图块大小需要能够被样方长宽整除，例如样方大小为(90,150)，那么配置patch\_size=15，可以划分为个图块。

img\_patches = usda\_models.img\_to\_patch(x\_train, patch\_size=15, flatten\_channels=False)  
print(img\_patches.shape)  
  
fig, ax = plt.subplots(10, 1, figsize=(20,5))  
fig.suptitle("Images as input sequences of patches")  
for i in range(10):  
 img\_grid = torchvision.utils.make\_grid(img\_patches[i], nrow=60, normalize=True, pad\_value=0.9)  
 img\_grid = img\_grid.permute(1, 2, 0)  
 ax[i].imshow(img\_grid)  
 ax[i].axis('off')  
plt.show()  
plt.close()

torch.Size([61346, 60, 1, 15, 15])



#### 2) 训练视觉模型

分别使用JenksCaspall、boxplot和EqualInterval三种分类数据集训练模型。在配置参数时，boxplot的类别num\_classes有6个，其它两个为5个。可以使用 TensorBoard 日志，方便后续训练参数可视化观察。

每一数据集训练迭代100次，用验证（测试）数据集分别计算各自训练模型的精度，boxplot分类方式精度为0.348；EqualInterval分类方式精度为0.664；JenksCaspall分类方式精度为0.324。基于 WorldCover 数据只能部分的解释 LST 数值，其中其它可能影响因素还包括 WorldCover 的分类类型是否适合；WorldCover 的分类精度目前 2020 年为74.4%，2021 年为76.6%；LST 的数据精度影响；三种数据集分类方式是否适合；切分样方的图块大小是否适合；模型本身的网络结构是否适合，及未考虑加入的其它影响因素等。

在进一步的探索中，除了依据上述影响因素调整参数，增加影响 LST 的其它因素外，可以重新设定问题，例如仅关注高于（或低于）某一 LST 值的区域，这样可以简化分类类别为两类，重点分析高温区域或低温区域的地表覆盖结构组成的向量空间特征等。

model\_kwargs=dict(embed\_dim=256,   
 hidden\_dim=512,   
 num\_channels=1,   
 num\_heads=8,   
 num\_layers=6,   
 num\_classes=5,   
 patch\_size=15,   
 num\_patches=60,   
 dropout=0.2,)  
  
# CHECKPOINT\_PATH=r'I:\model\_ckpts\ViT\_LC\_LST\_JenksCaspall'#r'I:\model\_ckpts\ViT\_LC\_LST'  
# CHECKPOINT\_PATH=r'I:\model\_ckpts\ViT\_LC\_LST\_boxplot'  
CHECKPOINT\_PATH=r'I:\model\_ckpts\ViT\_LC\_LST\_EqualInterval'  
  
device=torch.device("cuda:0") if torch.cuda.is\_available() else torch.device("cpu")  
logger=TensorBoardLogger(os.path.join(CHECKPOINT\_PATH, "tensorboard"), name="lclst")  
trainer=pl.Trainer(default\_root\_dir=os.path.join(CHECKPOINT\_PATH, "ViT"),  
 accelerator="gpu" if str(device).startswith("cuda") else "cpu",  
 #logger=logger,  
 devices=1,  
 max\_epochs=180,  
 callbacks=[ModelCheckpoint(save\_weights\_only=True, mode="max", monitor="val\_acc"),  
 LearningRateMonitor("epoch")])  
  
trainer.logger.\_log\_graph=True # If True, we plot the computation graph in tensorboard  
trainer.logger.\_default\_hp\_metric=None # Optional logging argument that we don't need

GPU available: True (cuda), used: True  
TPU available: False, using: 0 TPU cores  
IPU available: False, using: 0 IPUs  
HPU available: False, using: 0 HPUs

# Check whether pretrained model exists. If yes, load it and skip training  
pretrained\_filename = os.path.join(CHECKPOINT\_PATH, "ViT.ckpt")  
if os.path.isfile(pretrained\_filename):  
 print(f"Found pretrained model at {pretrained\_filename}, loading...")  
 model=usda\_models.ViT.load\_from\_checkpoint(pretrained\_filename) # Automatically loads the model with the saved hyperparameters  
else:  
 pl.seed\_everything(42) # To be reproducable  
 model=usda\_models.ViT(model\_kwargs,lr=3e-4)  
 trainer.fit(model, train\_dl, test\_dl)  
 model=usda\_models.ViT.load\_from\_checkpoint(trainer.checkpoint\_callback.best\_model\_path) # Load best checkpoint after training

Global seed set to 42  
LOCAL\_RANK: 0 - CUDA\_VISIBLE\_DEVICES: [0]  
  
 | Name | Type | Params  
-------------------------------------------------  
0 | model | VisionTransformer | 3.2 M   
1 | valid\_acc | MulticlassAccuracy | 0   
-------------------------------------------------  
3.2 M Trainable params  
0 Non-trainable params  
3.2 M Total params  
12.953 Total estimated model params size (MB)  
Epoch 23: 38%|▍| 184/479 [1:11:44<1:55:00, 23.39s/it, v\_num=0, loss\_step=0.768, val\_loss=0.856, val\_acc=0.67

试验中测试和验证数据集使用了相同的数据。

test\_result=trainer.test(model, test\_dl, verbose=False)  
result={"test": test\_result[0]["test\_acc"]}  
print("ViT results", result)

LOCAL\_RANK: 0 - CUDA\_VISIBLE\_DEVICES: [0]  
  
  
Testing DataLoader 0: 100%|████████████████████████████████████████████████| 120/120 [00:05<00:00, 22.22it/s]  
ViT results {'test': 0.67177414894104}

## 3.6.2 街道视域景观指数的时空特征表述和预测

### 3.6.2.1 数据准备

#### 1) 全景静态图检索下载

分析对象为西安老城区东西大街。

1. 首先用 [Google Earth](https://www.google.com/earth/versions/)⑤沿道路绘制路径，保存为 KML 数据格式。定义kml2gdf\_folder()函数读取 KML 对象为 GeoDataFrame 格式数据。（也可以直接使用即有的矢量（SHP 格式）道路数据文件，用geopandas库读取）；
2. 定义roads\_pts4bsv\_tourLine()函数，由道路线段提取给定距离的采样点，用于全景图采样定位；
3. 定义baidu\_steetview\_crawler()数据检索工具，根据采样点从[百度地图开放平台](https://lbsyun.baidu.com/index.php?title=%E9%A6%96%E9%A1%B5)⑥全景静态图API下载全景图；
4. 定义img\_valid\_copy\_folder()函数，验证下载图像的有效性，并复制有效的图像至新建的文件夹下。总共下载图像583张，有效图像568张。

%load\_ext autoreload   
%autoreload 2   
%reload\_ext autoreload  
import usda.geodata\_process as usda\_geodataProcess   
import usda.datasets as usda\_datasets  
from usda.migrated\_project import pass\_panoseg  
import usda.utils as usda\_utils  
import usda.pano\_projection\_transformation as usda\_pano  
from usda.database import gpd2postSQL,postSQL2gpd  
import usda.utils as usda\_utils  
import usda.models as usda\_models  
  
import matplotlib.pyplot as plt  
import geopandas as gpd  
import pandas as pd  
import os  
from matplotlib import image as mpimg  
import pickle  
import glob  
import rioxarray as rxr  
import seaborn as sb  
import torch.nn as nn  
  
import torch.optim as optim  
import torch.utils.data as data  
import numpy as np  
import torch  
import plotly.express as px

读取 KML 路径。

huiminjie\_fn='../data/dongxistreet.kml'  
  
xian\_epsg=32649  
huiminjie=usda\_geodataProcess.kml2gdf\_folder(huiminjie\_fn,epsg=xian\_epsg)  
huiminjie

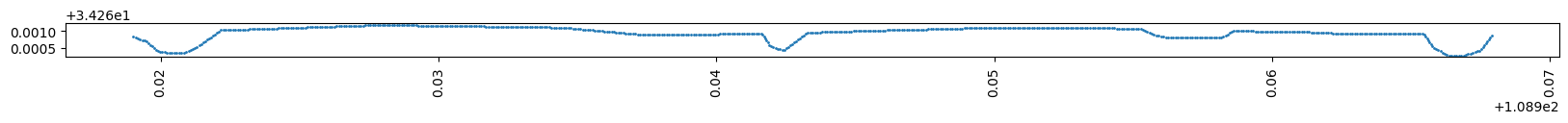
100%|██████████| 1/1 [00:00<00:00, 10.71it/s]

|  | **Name** | **Description** | **geometry** | **group** |
| --- | --- | --- | --- | --- |
| **0** | dongxistreet |  | LINESTRING Z (308400.083 3793037.020 0.000, 30... | dongxistreet.kml |

提取采样点。

sample\_multipts=usda\_datasets.roads\_pts4bsv\_tourLine(huiminjie,8)  
ax=sample\_multipts.plot(figsize=(20,10),markersize=1)  
plt.xticks(rotation=90);

100%|██████████| 1/1 [00:00<00:00, 62.82it/s]  
100%|██████████| 1/1 [00:00<00:00, 13.18it/s]



因为定义的baidu\_steetview\_crawler()方法中，区分了具有不同名称的道路路径，其默认字段为Name，因此增加该字段。

sample\_multipts['Name']='dongxistreet'  
sample\_multipts

|  | **Name** | **group** | **geometry** |
| --- | --- | --- | --- |
| **0** | dongxistreet | huiminjie.kml | MULTIPOINT Z (108.93892 34.26214 0.00000, 108.... |

检索全景图。

百度地图开放平台每天有固定配额，超出配额，需要另行支付费用。

pano\_fn=r'E:\data\pano\_dongxistreet\images'  
pano\_info={'pt\_fns':r'E:\data\pano\_dongxistreet\pt\_fns\_street.pickle',  
 "coords":r'E:\data\pano\_dongxistreet\coords\_street.pickle',  
 "downloadError\_idx":r'E:\data\pano\_dongxistreet\downloadError\_idx\_street.pickle'}  
ak='rSxNX840wLxwVVhs5RDInfPqegZ12G78'  
  
coords,pts\_num=usda\_datasets.baidu\_steetview\_crawler(sample\_multipts,pano\_fn,ak,pano\_info)

pts\_num=583  
  
  
100%|██████████| 1/1 [05:26<00:00, 326.31s/it]

验证图像的有效性。

with open(pano\_info["pt\_fns"],'rb') as f:  
 pt\_fns=pickle.load(f)  
   
pano\_valid\_fn=r'E:\data\pano\_dongxistreet\images\_valid'  
usda\_datasets.img\_valid\_copy\_folder(pano\_fn,pano\_valid\_fn)

100%|██████████| 583/583 [00:00<00:00, 2776.87it/s]  
100%|██████████| 568/568 [00:04<00:00, 135.04it/s]

#### 2) 全景图像素级语义分割（Pixel-wise semantic segmentation）

应用[PASS（ERF-PSPNet）深度学习模型](https://github.com/elnino9ykl/PASS)⑦实现全景图像素级语义分割。Yang, K等人[2]提供了 大小的预训练模型参数，而百度全景图的大小也为，因此可以避免重新训练模型。为了方便使用该模型，将其代码迁移至USDA库方便调用。迁移时为了避免增加库的大小，并未迁移预训练模型参数文件，需要从 PASS 作者的 [GitHub 代码仓库](https://github.com/elnino9ykl/PASS/tree/master/trained_models)⑧获取，作为输入参数传入。全景图分割后保存于两个子文件夹，一个为seg\_img，存储分割图像；另一个为`seg\_label’，存储分割标签数据。

weights\_fn=r'C:\Users\richie\omen\_richiebao\omen\_github\USDA\_special\_study\models\erfpspnet.pth'  
pano\_dir=r'G:\data\pano\_dongxistreet\images\_valid'  
save\_dir=r'G:\data\pano\_dongxistreet\pano\_seg'  
pass\_panoseg.pass\_seg(weights\_fn,pano\_dir,save\_dir)

Model and weights LOADED successfully  
  
  
100%|██████████| 566/566 [01:38<00:00, 5.73it/s]

打印查看分割后结果。

fn=r'dongxistreet\_228'  
fig,axes=plt.subplots(1,2,figsize=(20,10))  
img=mpimg.imread(os.path.join(pano\_dir,fn+'.jpg'))  
axes[0].imshow(img)  
  
seg=mpimg.imread(os.path.join(save\_dir,f'seg\_img\{fn}.jpg'))  
axes[1].imshow(seg)  
plt.show()

查看分割标签文件。

with open(os.path.join(save\_dir,f'seg\_label\{fn}.pkl'),'rb') as f:  
 label=pickle.load(f)  
label

tensor([[22, 22, 22, ..., 22, 22, 22],  
 [22, 22, 22, ..., 22, 22, 22],  
 [22, 22, 22, ..., 22, 22, 22],  
 ...,  
 [27, 27, 27, ..., 27, 27, 27],  
 [27, 27, 27, ..., 27, 27, 27],  
 [27, 27, 27, ..., 27, 27, 27]], dtype=torch.uint8)

#### 3) 百度 POI 数据检索下载

百度 POI 数据下载参看*POI数据与描述性统计和正态分布*部分。这里直接使用参看部分的处理结果数据，并将其裁切到分析路径缓冲距离 1000 m 范围内。

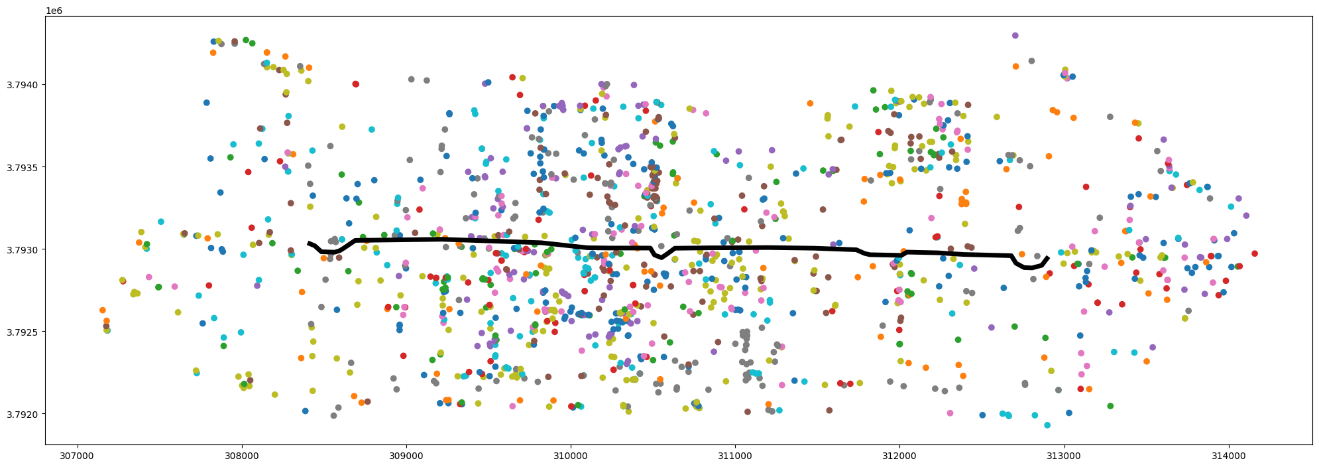
pois\_fn=r'G:\data\POI\_dongxistreet\poisInAll\_gdf.geojson'  
pois=gpd.read\_file(pois\_fn)  
pois.to\_crs(huiminjie.crs,inplace=True)  
  
region=huiminjie.buffer(1000,join\_style=3,cap\_style=3)  
clipped\_pois=gpd.clip(pois,region)  
clipped\_pois

|  | **level\_0** | **level\_1** | **name** | **location\_lat** | **location\_lng** | **detail\_info\_tag** | **detail\_info\_overall\_rating** | **detail\_info\_price** | **geometry** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1237** | poi\_0\_delicacy | 2474 | 韩式自助烤肉店 | 34.252172 | 108.929424 | 美食;外国餐厅 | 4.4 | 77 | POINT (309338.325 3792056.394) |
| **14319** | poi\_14\_realEstate | 2046 | 众生房产 | 34.252177 | 108.929302 | 生活服务;房产中介机构 | NaN | NaN | POINT (309327.161 3792057.198) |
| **1200** | poi\_0\_delicacy | 2400 | 粤珍轩(含光店) | 34.252202 | 108.927973 | 美食;中餐厅 | 3.9 | 115 | POINT (309204.828 3792062.485) |
| **4144** | poi\_10\_medicalTreatment | 1688 | 西安碑林含光门中医诊所 | 34.252214 | 108.928526 | 医疗;诊所 | NaN | NaN | POINT (309255.754 3792062.769) |
| **35110** | poi\_5\_spot | 1090 | 西北联大纪念碑 | 34.251402 | 108.920978 | 旅游景点;文物古迹 | 5.0 | NaN | POINT (308558.776 3791986.915) |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **20867** | poi\_16\_government | 2166 | 中共莲湖区教育局委员会 | 34.268539 | 108.935593 | 政府机构;各级政府 | NaN | NaN | POINT (309943.366 3793860.143) |
| **20877** | poi\_16\_government | 2186 | 陕西省观赏石协会西安观赏石协会联络处 | 34.268617 | 108.935436 | 政府机构;行政单位 | NaN | NaN | POINT (309929.062 3793869.041) |
| **20866** | poi\_16\_government | 2164 | 中国共产党西安市莲湖区建设和住房保障局委员会 | 34.268627 | 108.935667 | 政府机构;其他 | NaN | NaN | POINT (309950.306 3793869.743) |
| **20845** | poi\_16\_government | 2122 | 中共莲湖区城市管理和综合执法局委员会 | 34.268647 | 108.936819 | 政府机构;其他 | NaN | NaN | POINT (310056.438 3793869.855) |
| **20881** | poi\_16\_government | 2194 | 西安市莲湖区教育区 | 34.268770 | 108.935615 | 政府机构;行政单位 | NaN | NaN | POINT (309945.841 3793885.682) |

1389 rows × 9 columns

查看 POI 数据裁切后结果。

f, ax=plt.subplots(1,1,figsize=(20,20))  
clipped\_pois.plot(ax=ax,column='level\_0')  
huiminjie.plot(ax=ax,edgecolor='k',linewidth=5,linestyle='-')  
plt.tight\_layout()  
plt.show()



保存裁切后的 POI 数据。

clipped\_pois\_fn=r'G:\data\POI\_dongxistreet\clipped\_poi\_gdf.gpkg'  
clipped\_pois.to\_file(clipped\_pois\_fn,driver='GPKG', layer='poi')

### 3.6.2.2 全景与语义分割图的投影变换

在城市街道空间的组成结构分析过程中，需要根据不同的分析内容对数据进行预先处理满足分析的要求和改善计算的准确性。极坐标格式全景图(小行星视角360度全景)用于天空相关指数的计算，避免等量矩形投影图未闭合的天空形状影响；等量矩形投影图和极坐标格式全景图因为投影变形，对象像素所占比例不能最大限度的反应实际视觉下对象比例关系。而立方体型全景格式由前，后，左，右，上，下6张透视图组成，用于语义分割对象所占像素百分比的统计相对合理；球面全景，可以反应对象在实际空间中的位置关系，用于语义分割对象空间位置变化的分析。

#### 1) 等量矩形投影图到立方体型全景格式

应用Zhou, C.[3]等人 [Oculus 360 度视频流测量研究成果](https://github.com/bingsyslab/360projection)⑨，实现全景图到立方体格式全景图的投影。迁移代码至USDA库，定义equi2cube()方法，并应用多线程计算。定义方法时，输出结果包含3类数据，在生成的子文件夹img\_seg\_redefined\_color下为重新定义的 PASS 分割结果分类颜色配置，其值如下label\_color\_name属性；在子文件夹cube\_img\_seg下为立方体格式全景图图像；在子文件夹cube\_label\_seg下为立方体格式全景图对应像素标签。

\_\_C=usda\_utils.AttrDict()  
args=\_\_C  
\_\_C.pano\_path=r'G:\data\pano\_dongxistreet\images\_valid'  
\_\_C.label\_seg\_path=r'G:\data\pano\_dongxistreet\pano\_seg\seg\_label'   
\_\_C.face\_size=1000  
\_\_C.equi2cub\_dir=r'G:\data\pano\_dongxistreet\pano\_projection\_transforms'  
  
\_\_C.pano\_redefined\_path=r'G:\data\pano\_dongxistreet\pano\_projection\_transforms\img\_seg\_redefined\_color'  
\_\_C.output\_shape=(1024,1024)  
\_\_C.little\_planet='little\_planet\_1'  
\_\_C.polar\_seg\_img\_dir=r'G:\data\pano\_dongxistreet\polar\_seg\_img'  
  
\_\_C.coords\_street\_fn=r'G:\data\pano\_dongxistreet\coords\_street.pickle'  
  
\_\_C.db=usda\_utils.AttrDict()   
\_\_C.db.UN='postgres'  
\_\_C.db.PW='123456'  
\_\_C.db.DB='visual\_metrics'  
\_\_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.tiff\_sky\_save\_root=r'G:\data\pano\_dongxistreet\tiff\_sky'  
  
\_\_C.colors\_dominant2cluster\_path=r'G:\data\pano\_dongxistreet\colors\_dominant2cluster.pkl'  
  
\_\_C.visual\_BOW\_kmeans\_fn=r'G:\data\pano\_dongxistreet\visual\_BOW\_kmeans.pkl'  
\_\_C.feature\_map\_fn=r'G:\data\pano\_dongxistreet\feature\_map.pkl'

label\_seg\_path=args.label\_seg\_path  
label\_seg\_fns=glob.glob(os.path.join(label\_seg\_path,'\*.pkl'))  
equi2cube\_dir=args.equi2cub\_dir  
face\_size=args.face\_size  
  
usda\_pano.equi2cube(label\_seg\_fns,equi2cube\_dir,face\_size,cpu\_num=8)

100%|██████████| 144/566 [00:10<00:00, 1385.52it/s]

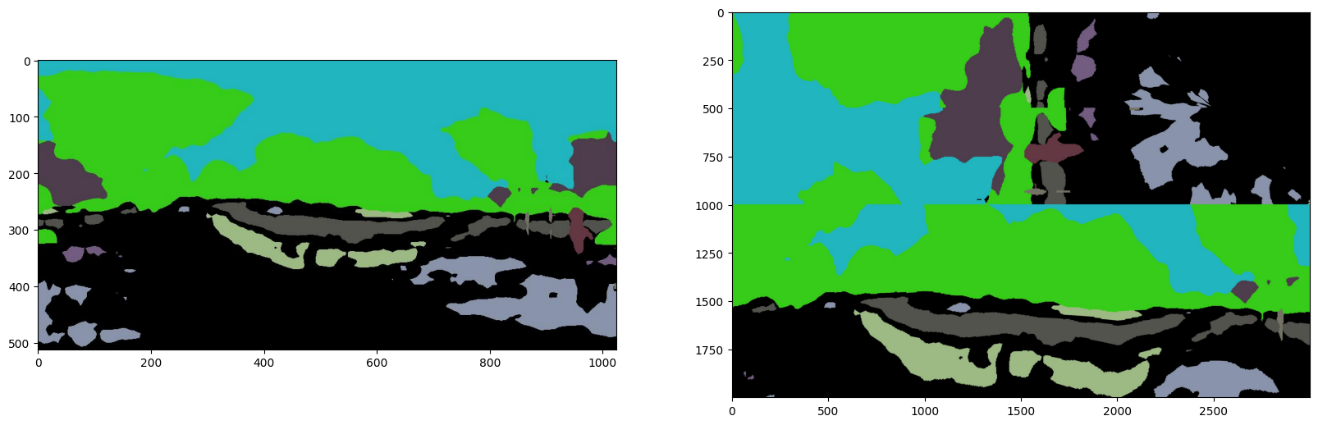
查看重新定义 PASS 全景图分割结果颜色值。

usda\_pano.label\_color\_name

{0: [(117, 115, 102), 'pole'],  
 1: [(212, 209, 156), 'slight'],  
 2: [(224, 9, 9), 'bboard'],  
 3: [(227, 195, 66), 'tlight'],  
 4: [(137, 147, 169), 'car'],  
 5: [(53, 67, 98), 'truck'],  
 6: [(185, 181, 51), 'bicycle'],  
 7: [(238, 108, 91), 'motor'],  
 8: [(247, 5, 5), 'bus'],  
 9: [(127, 154, 82), 'tsignf'],  
 10: [(193, 209, 167), 'tsignb'],  
 11: [(82, 83, 76), 'road'],  
 12: [(141, 142, 133), 'sidewalk'],  
 13: [(208, 212, 188), 'curbcut'],  
 14: [(98, 133, 145), 'crosspln'],  
 15: [(194, 183, 61), 'bikelane'],  
 16: [(141, 139, 115), 'curb'],  
 17: [(157, 186, 133), 'fence'],  
 18: [(114, 92, 127), 'wall'],  
 19: [(78, 61, 76), 'building'],  
 20: [(100, 56, 67), 'person'],  
 21: [(240, 116, 148), 'rider'],  
 22: [(32, 181, 191), 'sky'],  
 23: [(55, 204, 26), 'vege'],  
 24: [(84, 97, 82), 'terrain'],  
 25: [(231, 24, 126), 'markings'],  
 26: [(141, 173, 166), 'crosszeb'],  
 27: [(0, 0, 0), 'Nan']}

查看输出结果。

fn=r'dongxistreet\_228'  
fig,axes=plt.subplots(1,2,figsize=(20,10))  
seg=mpimg.imread(os.path.join(equi2cube\_dir,'img\_seg\_redefined\_color/'+fn+'.jpg'))  
axes[0].imshow(seg)  
cube=mpimg.imread(os.path.join(equi2cube\_dir,'cube\_img\_seg/'+fn+'.jpg'))  
axes[1].imshow(cube)  
plt.show()



#### 2） 等量矩形投影图到极坐标格式全景图

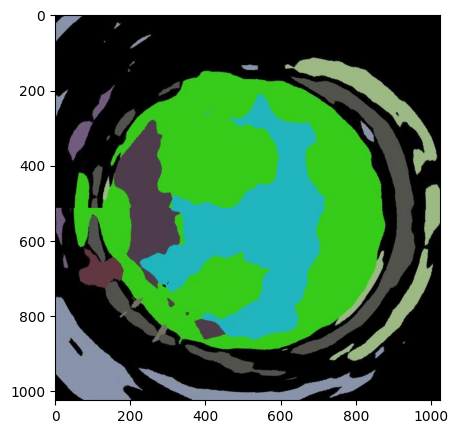
应用[Making a Little Planet Panorma using Python and Scikit Image](https://richwareham.com/little-planet-projection/)⑩阐述的方法，定义equi2polar()函数，将全景图分割图像转化为极坐标格式。

pano\_path=args.pano\_redefined\_path  
output\_shape=args.output\_shape  
little\_planet=args.little\_planet  
polar\_save\_path=args.polar\_seg\_img\_dir   
  
usda\_pano.equi2polar(pano\_path,output\_shape,polar\_save\_path,little\_planet='little\_planet\_1')

100%|██████████| 566/566 [01:41<00:00, 5.60it/s]

查看转化后的结果。

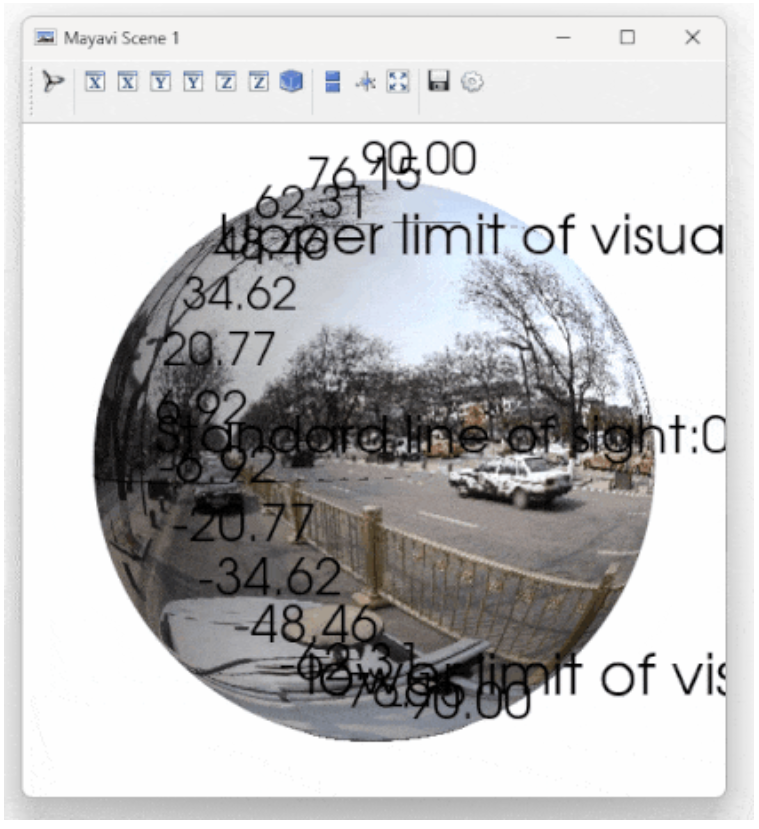
fn=r'dongxistreet\_228'  
fig,ax=plt.subplots(figsize=(5,5))  
polar=mpimg.imread(os.path.join(polar\_save\_path,fn+'.jpg'))  
ax.imshow(polar)  
plt.show()



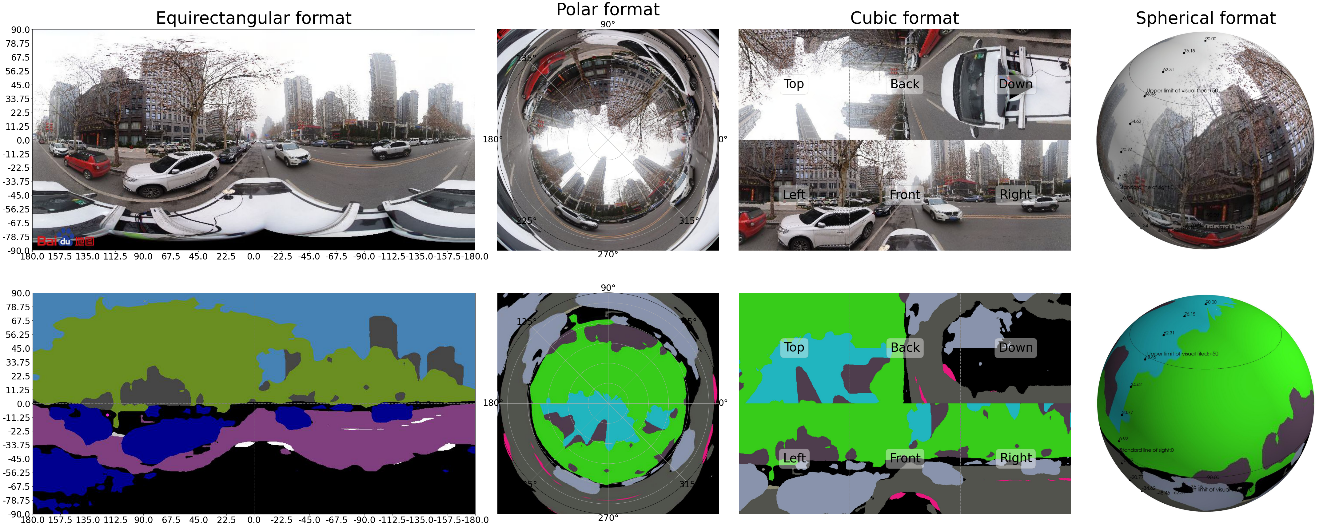
#### 3）等量矩形投影图到球面全景

使用 [mayavi](https://docs.enthought.com/mayavi/mayavi/) ⑪三维科学数据可视化绘图工具，定义sphere\_panorama\_label()方法实现球面全景投影。mlab.show()方法可以交互显示三维图像，也可以通过mlab.savefig方法保存图像。

fn=r'dongxistreet\_228'  
pano\_img\_fn=os.path.join(pano\_dir,fn+'.jpg')   
usda\_pano.sphere\_panorama\_label(pano\_img\_fn)



将上述完成投影变换计算的立方体型全景格式、极坐标格式全景图、球面全景与原始分割全景和对应的原始全景图像排布如下观察。



### 3.6.2.3 视域景观指数计算

分析城市街道空间特征的指数主要包括基于立方体型360度全景图语义分割对象像素占比，即绿视率、天空视域因子和地面视域占比，及视觉（熵）均衡度和域对象位置变化；基于极坐标格式全景图语义分割天空对象的景观指数，包括周长面积比、形状指数和分维数；基于等量矩形投影全景图颜色的主题色提取和色彩丰富度指数；基于尺度不变特征转换特征关键点信息的关键点邻域尺度(i,j]区间频数、标准差和视域特征匹配消失距离；基于行业分类服务内容兴趣点的POI点数、POI均衡度和POI特征映射，总共16个指数。其中反应视域城市街道物质空间组成结构，基于全景图计算机视觉分析的指数13个；反应居民日常活动，基于具有社会属性，即服务空间的兴趣点指数3个。

| 序号 | 指数名称 | 值个数 | 内容 | 公式 | 描述 |
| --- | --- | --- | --- | --- | --- |
| 1 | 绿视率（Seg object\_Green view index, Seg\_GVI） | 1 | 基于立方体型360度全景图语义分割对象像素占比 |  | 绿视率（）等于立方体型360度全景图语义分割对象中植被像素数 占全部像素数P的百分比。 |
| 2 | 天空视域因子（Seg object Sky view factor, Seg\_SVF） | 1 |  |  | 天空视域因子（ ）等于立方体型360度全景图语义分割对象中天空像素数 占全部像素数 P的百分比。 |
| 3 | 地面视域占比（Seg object Ground view factor, Seg\_GVF） | 1 |  |  | 地面视域占比（ ）等于立方体型360度全景图语义分割对象中地面像素数 占全部像素数P的百分比。 |
| 4 | 视觉（熵）均衡度（Seg object equilibrium degree, Seg\_ED） | 1 |  |  | 视觉（熵）均衡度（）为立方体型360度全景图语义分割对象信息熵与最大熵值之比。 为第个语义分割对象像素数与全部像素数的比值。 |
| 5 | 视域对象位置变化（Variation degree of objects , VDO） | 多个值 |  | $VDO\_{ij}= ( A\_{0} ) *{ij} + ( A*{1} ) *{ij} + + ( A*{k-1} ) *{ij}; (A)* {ij}= | 视域对象位置变化为分析路径下各个地理位置点的球面全景语义分割对象在球面各像素点位置上的变化，即任意球面位置在路径上相邻两位置语义分割对象发生了改变则加1，即否则加0，未发生改变，得球面位置的变化次数。矩阵即为视域对象位置变化。 |
| 6 | 天际线周长面积比（均值）（Skyline perimeter area ratio, Sky\_PARA(mn)） | 1 | 基于极坐标格式全景图语义分割天空对象的景观指数 |  | 天际线周长面积比（均值）（）等于极坐标格式全景图语义分割对象中各个天空斑块周长与面积 比之和除以天空的斑块数。由PyLandStats库（Python）计算。 |
| 7 | 天际线形状指数（均值）（Skyline shape index, Sky\_ SHAPE(mn)） | 1 |  | Sky\_SHAPE\_{(mn)} = | 天际线形状指数（均值）（$ Sky\_SHAPE\_{(mn)}p\_{i}a\_{i}$ 平方根比值乘以调整系数的和除以天空的斑块数 。 由PyLandStats库（Python）计算。 |
| 8 | 天际线分维数（均值）（Skyline fractal dimension, Sky\_ FRAC(mn) ） | 1 |  | Sky\_FRAC\_{(mn)} = | 天际线分维数（均值）（ ）等于极坐标格式全景图语义分割对象中各个天空斑块周长乘以调整系数后取其对数的2倍与面积 的对数比值的和除以天空的斑块数 。 由PyLandStats库（Python）计算。 |
| 9 | 主题色（Theme color, TC） | 多个值 | 基于等量矩形投影全景图城市街道色彩 |  | 主题色（）为等量矩形投影全景图颜色值聚类各簇中心值 的集合，即第个类簇所有颜色点各属性值（R、G、B）的和除以该簇个数。聚类条件为使得所有簇内各颜色点到该簇中心值（经验均值）的平方误差和最小。 由scikit-learn库（Python）的K-Means方法聚类。 |
| 10 | 色彩丰富度指数（Color richness index, CRI） | 1 |  |  | 色彩丰富度指数（CRI）为等量矩形投影全景图主题色聚类的信息熵与最大熵值log⌈S⌉的比值， ⌈S⌉ 即主题色聚类数，全景图各颜色点所属主题色由主题色TC计算时获取。P\_i为第i个主题色聚类簇颜色点数（像素数）⌈S\_i ⌉与全部像素数的比值。 |
| 11 | 关键点邻域尺度(i,j]区间频数（Key point size(i-j] frequency,KPSF\_(i,j] ) | 多个值 | 尺度不变特征转换（Scale Invariant Feature Transform, SIFT） | KPSF\_{(i,j]} ={ x\_{k} :i x\_{k} <j, x\_{k} KPS } | 关键点邻域尺度区间频数（) 为关键点邻域尺度, 即提取的等量矩形投影全景图特征关键点的大小 , 满足条件的集合。 由opencv-python库（Python）StarDetector\_create()方法提取特征关键点。 |
| 12 | 关键点邻域尺度标准差（Key point size std, KPS\_STD） | 1 |  |  | 关键点邻域尺度标准差（）为各个关键点邻域尺度 与均值 的差值平方除以关键点数的平方根。 由opencv-python库（Python）StarDetector\_create()方法提取特征关键点。 |
| 13 | 视域特征匹配消失距离 （Vanishing distance of feature matching, VDFM ） | 1 |  |  | 视域特征匹配消失距离 为分析路径某一位置等量矩形投影全景图与之后所有位置全景图图像匹配特征点数变化趋于平缓的位置与该位置的路径距离。为连续相邻图像匹配特征点数的差值，为差值的差值；为满足条件的第一个值作为参数输入函数，返回其间的路径距离。 由opencv-python库（Python）BFMatcher()方法计算图像匹配。 |
| 14 | 兴趣点数（Number of POI, POI\_NUM） | 1 | POI |  | 兴趣点数为分析路径某一位置给定缓冲区下兴趣点的数量。 |
| 15 | POI均衡度（Equilibrium degree of POI, POI\_ED） | 1 |  | $POI\_ED= $; | 均衡度为分析路径某一位置给定缓冲区下兴趣点一级行业分类的信息熵与最大熵值的比值，为兴趣点一级行业分类数。为第个一级行业分类数与全部一级行业分类数的比值。 |
| 16 | POI特征映射（码本映射）（Codewords of POI, POI\_CW） | 多个值 |  | POI\_CW=[w\_{0},w\_{1},⋯,w\_{k} ] | 特征映射（码本映射）为分析路径某一位置给定缓冲区下兴趣点各一级行业分类的数量或其与总数的比值的矩阵或向量。为一级行业分类的数量，即为矩阵的长度。 |

#### 1）街道空间对象视域占比指数测量

语义分割立方体型全景图，计算植被（绿视率）、天空（天空视域因子）、地面视域所占百分比，以及视觉熵均衡度。将计算结果划分为4个数量级区间，(0,15], (15,25], (25,50]和(50,100]，可以描述为较少，正常，较多，多。街道空间对象视域占比指数与城市街区类型，空间体验，市民活动行为，建筑外热环境（微气候）等内容相关。

* 街道空间对象视域占比

with open(args.coords\_street\_fn,'rb') as f:   
 coords=pickle.load(f)

label\_seg\_path=os.path.join(args.equi2cub\_dir,'cube\_label\_seg')   
img\_Seg\_path=os.path.join(args.equi2cub\_dir,'cube\_img\_seg')   
cube\_object\_percent\_gdf=usda\_pano.metrics\_seg\_pano\_proportion(label\_seg\_path,img\_Seg\_path,coords,)

100%|██████████| 566/566 [03:33<00:00, 2.65it/s]

gpd2postSQL(cube\_object\_percent\_gdf,table\_name='cube\_object\_percent',\*\*args.db.db\_info)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is cube\_object\_percent.

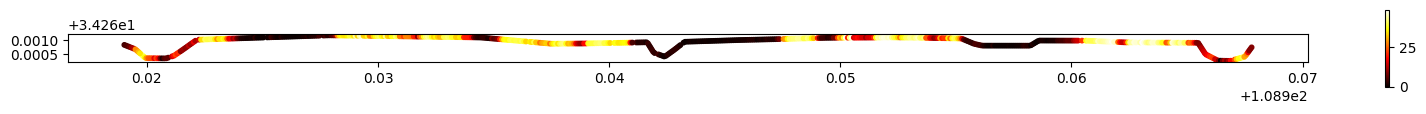
cube\_object\_percent\_gdf=postSQL2gpd(table\_name='cube\_object\_percent',\*\*args.db.db\_info)   
cube\_object\_percent\_gdf.tail(3)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is cube\_object\_percent.

|  | **fn\_stem** | **fn\_key** | **fn\_idx** | **geometry** | **pole** | **slight** | **bboard** | **tlight** | **car** | **truck** | **...** | **person** | **rider** | **sky** | **vege** | **terrain** | **markings** | **crosszeb** | **Nan** | **ground\_diff** | **sky\_vege** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **563** | dongxistreet\_97 | dongxistreet | 97 | POINT (108.92703 34.26116) | 0.006400 | 0.0 | 2.758550 | 0.0 | 3.548533 | 0.0 | ... | 0.011150 | 0.0 | 29.560383 | 0.521933 | 0.0 | 0.0 | 0.0 | 31.350317 | 41.225100 | 30.082317 |
| **564** | dongxistreet\_98 | dongxistreet | 98 | POINT (108.92712 34.26116) | 0.020317 | 0.0 | 1.121667 | 0.0 | 12.247250 | 0.0 | ... | 0.008017 | 0.0 | 29.791083 | 1.563883 | 0.0 | 0.0 | 0.0 | 14.151933 | 50.491133 | 31.354967 |
| **565** | dongxistreet\_99 | dongxistreet | 99 | POINT (108.92720 34.26116) | 0.000000 | 0.0 | 0.982000 | 0.0 | 23.858583 | 0.0 | ... | 0.047583 | 0.0 | 30.658100 | 2.070233 | 0.0 | 0.0 | 0.0 | 9.059367 | 46.557150 | 32.728333 |

3 rows × 34 columns

cube\_object\_percent\_gdf.plot(column='vege',figsize=(20,1),markersize=10,legend=True, cmap='hot');



* 视觉（熵）均衡度

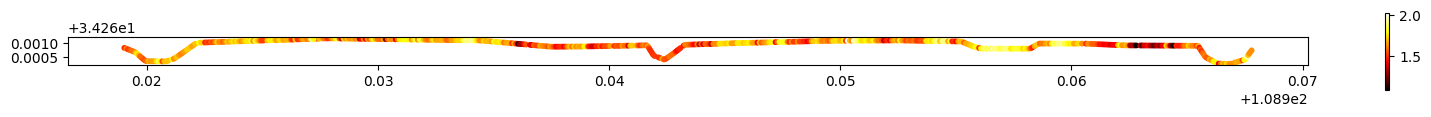
cube\_object\_VE\_gdf=usda\_pano.metrics\_visual\_entropy(cube\_object\_percent\_gdf)  
gpd2postSQL(cube\_object\_VE\_gdf,table\_name='cube\_object\_percent',\*\*args.db.db\_info)   
cube\_object\_VE\_gdf.tail(3)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is cube\_object\_percent.

|  | **fn\_stem** | **fn\_key** | **fn\_idx** | **geometry** | **pole** | **slight** | **bboard** | **tlight** | **car** | **truck** | **...** | **vege** | **terrain** | **markings** | **crosszeb** | **Nan** | **ground\_diff** | **sky\_vege** | **ground** | **ve** | **equilibrium\_degree** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **563** | dongxistreet\_97 | dongxistreet | 97 | POINT (108.92703 34.26116) | 0.006400 | 0.0 | 2.758550 | 0.0 | 3.548533 | 0.0 | ... | 0.521933 | 0.0 | 0.0 | 0.0 | 31.350317 | 41.225100 | 30.082317 | 53.860950 | 1.674268 | 50.245055 |
| **564** | dongxistreet\_98 | dongxistreet | 98 | POINT (108.92712 34.26116) | 0.020317 | 0.0 | 1.121667 | 0.0 | 12.247250 | 0.0 | ... | 1.563883 | 0.0 | 0.0 | 0.0 | 14.151933 | 50.491133 | 31.354967 | 54.351650 | 1.948515 | 58.475244 |
| **565** | dongxistreet\_99 | dongxistreet | 99 | POINT (108.92720 34.26116) | 0.000000 | 0.0 | 0.982000 | 0.0 | 23.858583 | 0.0 | ... | 2.070233 | 0.0 | 0.0 | 0.0 | 9.059367 | 46.557150 | 32.728333 | 51.518967 | 1.787379 | 53.639539 |

3 rows × 37 columns

cube\_object\_percent\_gdf.plot(column='ve',figsize=(20,1),markersize=10,legend=True, cmap='hot');



* 街道空间对象视域占比区间统计

从计算结果的表中可以初步判断分析区域不同视域占比频数比例的分布，绿视率vege\_percentage低于15%的接近一半数量，道路以车行为主，高于25%适宜于步行体验，易于放松身心的占到约30%；天空视域因子sky\_percentage高于50%的基本没有，以25%为数量级划分，高低各占约一半；将绿视率和天空视域因子综合考虑，即自然因素视域占比sky\_vege\_percentage，高于25%的达到96%的数量；地面视域占比ground\_percentage有42.049%的数量高于50%，通常为道路交叉口以及快速路、主干路等路面宽阔的空间。街道空间对象的混合程度equilibrium\_degree\_percentage以50%为数量级划分，上下各占约一半，即约有一半数量的街道空间对象均质性相对较强，对象视域数量分布均衡；而一半数量均质性较弱，以某一类对象占主导。

params={'bins': [0,15,25,50,100],  
 'columns': ['vege','sky','sky\_vege','ground','equilibrium\_degree',],  
 'percentage2excel\_path': '../graph/objects\_percentage.xlsx',  
 'digits': 3}  
  
percentage,column\_names=usda\_pano.percent\_frequency(cube\_object\_percent\_gdf,params['columns'],params['bins'],params['digits'])  
percentage

|  | **vege** | **sky** | **sky\_vege** | **ground** | **equilibrium\_degree** | **vege\_percentage** | **sky\_percentage** | **sky\_vege\_percentage** | **ground\_percentage** | **equilibrium\_degree\_percentage** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **(-0.001, 15.0]** | 265 | 173 | 2 | 0 | 0 | 46.820 | 30.565 | 0.353 | 0.000 | 0.000 |
| **(15.0, 25.0]** | 59 | 132 | 13 | 0 | 0 | 10.424 | 23.322 | 2.297 | 0.000 | 0.000 |
| **(25.0, 50.0]** | 242 | 261 | 393 | 328 | 303 | 42.756 | 46.113 | 69.435 | 57.951 | 53.534 |
| **(50.0, 100.0]** | 0 | 0 | 158 | 238 | 263 | 0.000 | 0.000 | 27.915 | 42.049 | 46.466 |

#### 2）天际线变化指数

语义分割后的等量矩形投影全景图转换为极坐标格式的小行星全景图，提取天空对象计算斑块数量(NP)、周长面积比、形状指数、分维数等4个景观指数。

* 计算天际线变化指数

polar\_seg\_root=args.polar\_seg\_img\_dir  
hsv\_lower=[0,0,200]  
hsv\_upper=[180,255,255]   
tiff\_sky\_save\_root=args.tiff\_sky\_save\_root  
  
metrics\_skyline\_shape\_gdf=usda\_pano.metrics\_skyline\_shape(polar\_seg\_root,coords,hsv\_lower,hsv\_upper,tiff\_sky\_save\_root)

100%|██████████| 566/566 [02:33<00:00, 3.69it/s]

gpd2postSQL(metrics\_skyline\_shape\_gdf,table\_name='metrics\_skyline\_shape',\*\*args.db.db\_info)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is metrics\_skyline\_shape.

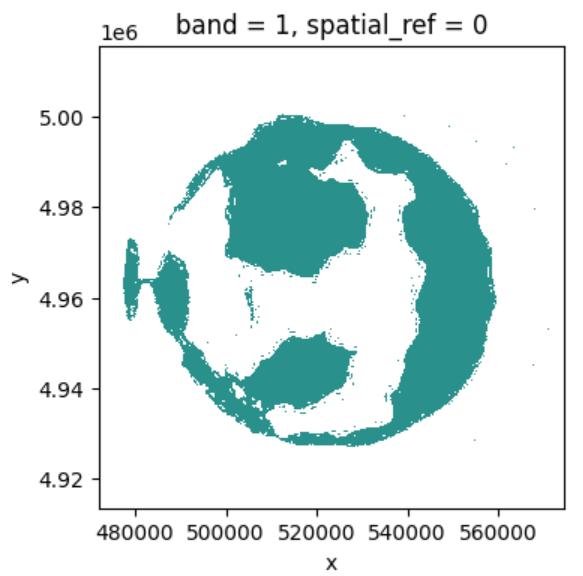
metrics\_skyline\_shape\_gdf=postSQL2gpd(table\_name='metrics\_skyline\_shape',\*\*args.db.db\_info)   
metrics\_skyline\_shape\_gdf.tail(3)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is metrics\_skyline\_shape.

|  | **fn\_stem** | **fn\_key** | **fn\_idx** | **geometry** | **total\_area** | **area\_mn** | **perimeter\_mn** | **perimeter\_area\_ratio\_mn** | **number\_of\_patches** | **landscape\_shape\_index** | **shape\_index\_mn** | **fractal\_dimension\_mn** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **563** | dongxistreet\_97 | dongxistreet | 97 | POINT (108.92703 34.26116) | 37749.0 | 150.996000 | 2205.600000 | 331.851991 | 250.0 | 7.087404 | 1.123028 | 1.018418 |
| **564** | dongxistreet\_98 | dongxistreet | 98 | POINT (108.92712 34.26116) | 140020.0 | 718.051282 | 4803.076923 | 296.252504 | 195.0 | 6.252336 | 1.206710 | 1.025840 |
| **565** | dongxistreet\_99 | dongxistreet | 99 | POINT (108.92720 34.26116) | 41104.0 | 185.990950 | 4474.208145 | 296.170397 | 221.0 | 12.177340 | 1.235508 | 1.028000 |

打印查看天空形状。

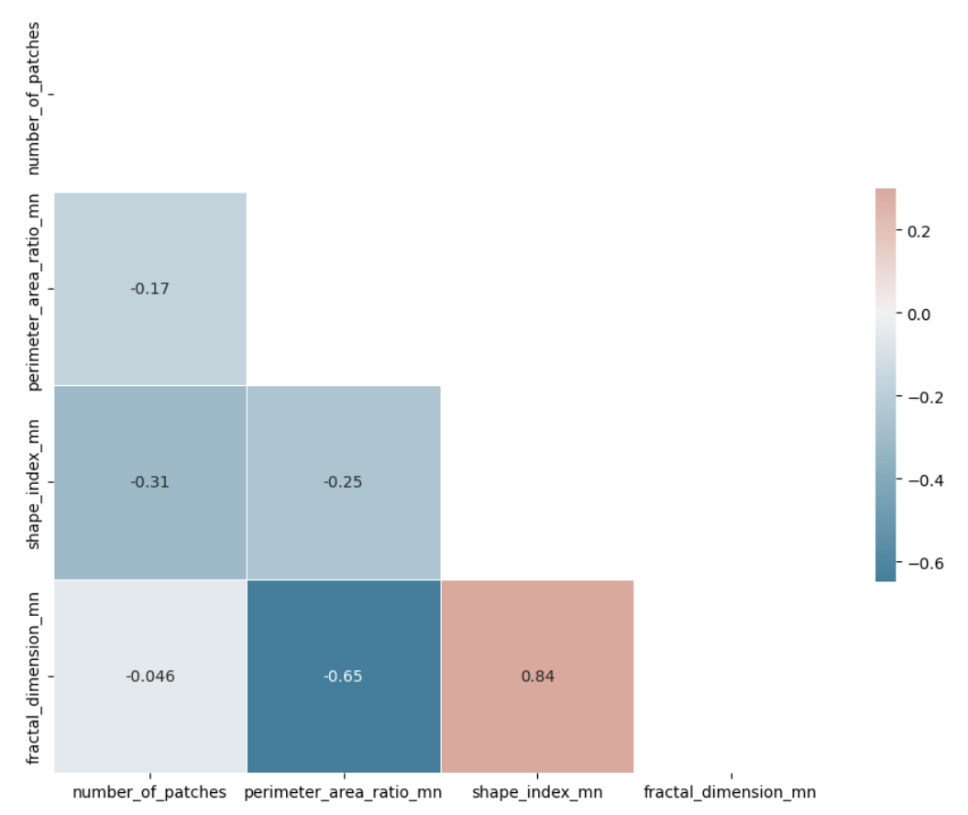
fn=r'dongxistreet\_228'  
sky\_example=rxr.open\_rasterio(os.path.join(args.tiff\_sky\_save\_root,fn+'.tif'),masked=True).squeeze()  
  
f, ax=plt.subplots(1,1,figsize=(4,4))  
sky\_example.plot.imshow(ax=ax,add\_colorbar=False)  
plt.show()



* 天际线景观指数间的相关系数

计算4个指数的相关系数，相关性值高于0.5的两对指数分别为周长面积比和分维数（0.65），及形状指数和分维数（0.84）。使用自然最佳断裂点分级方法（Natural break, Jenks）划分指数，尽量使得划分的聚类簇示例偏离划分界限值观察天际线的形状差异。

correlation\_columns=['number\_of\_patches','perimeter\_area\_ratio\_mn','shape\_index\_mn', 'fractal\_dimension\_mn',]  
corr=usda\_pano.correlation\_df(metrics\_skyline\_shape\_gdf,correlation\_columns,digits=3)



#### 3）街道空间的色彩

色彩属性表征城市街道空间的类型和视觉感受。计算区域尺度全景图主题色，通过主题色分布邻近像素聚类，计算获取每一全景图对应街道空间位置的主题色分布信息熵及均衡度。

* 计算图像主题色（多进程）

img\_path=args.pano\_path  
resize\_scale=0.5  
number\_of\_colors=16  
  
img\_dominant\_color\_gdf=usda\_pano.find\_dominant\_colors\_pool\_main(img\_path,[coords,resize\_scale,number\_of\_colors])

100%|██████████| 566/566 [11:57<00:00, 1.27s/it]

gpd2postSQL(img\_dominant\_color\_gdf,table\_name='img\_dominant\_color',\*\*args.db.db\_info)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is img\_dominant\_color.

img\_dominant\_color\_gdf=postSQL2gpd(table\_name='img\_dominant\_color',\*\*args.db.db\_info)   
img\_dominant\_color\_gdf.tail(3)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is img\_dominant\_color.

|  | **fn\_stem** | **fn\_key** | **fn\_idx** | **geometry** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **563** | dongxistreet\_97 | dongxistreet | 97 | POINT (108.92703 34.26116) | #d5d7d8 | #8ea9d6 | #aac4e8 | #030102 | #a82224 | #332f2e | #747070 | #e6eaef | #b7bbc2 | #1f1816 | #8f8988 | #a3a1a2 | #c2d9f4 | #4a4642 | #f9fbfd | #5b5959 |
| **564** | dongxistreet\_98 | dongxistreet | 98 | POINT (108.92712 34.26116) | #aa2426 | #757170 | #1d1a19 | #b8cde8 | #a7abb0 | #050303 | #888b8f | #85a1c7 | #dbe1ea | #a0b8d6 | #302d2d | #474545 | #f4f8fc | #615f5e | #cfd1d2 | #214ba5 |
| **565** | dongxistreet\_99 | dongxistreet | 99 | POINT (108.92720 34.26116) | #c2c4c6 | #3a3738 | #545252 | #070505 | #f4f8fc | #a62526 | #dcdddf | #6c6968 | #231f1f | #84a0c6 | #9db5d3 | #d2e1f6 | #b5cbe6 | #9ca0a5 | #244ba1 | #838183 |

* 主题色聚类（多进程）

colors\_dominant2cluster\_path=args.colors\_dominant2cluster\_path  
resize\_scale\_cluster=0.1  
\_=usda\_pano.dominant2cluster\_colors\_pool\_main(img\_path,colors\_dominant2cluster\_path,[coords,resize\_scale\_cluster,number\_of\_colors])

100%|██████████| 566/566 [01:13<00:00, 7.70it/s]

with open(colors\_dominant2cluster\_path,'rb') as f:  
 color\_dic\_list=pickle.load(f)

print(color\_dic\_list[0])

{'fn\_stem': 'dongxistreet\_0', 'fn\_key': 'dongxistreet', 'fn\_idx': 0, 'geometry': <POINT (108.919 34.261)>, 'counter': {1: 23, 0: 310, 2: 337, 3: 91, 4: 83, 5: 112, 6: 357, 7: 79, 8: 297, 9: 308, 10: 33, 11: 1, 12: 1, 13: 1, 14: 1, 15: 1, 16: 3, 17: 1, 18: 3, 19: 1, 20: 2, 21: 1, 22: 1, 23: 1, 24: 1, 25: 2, 26: 1, 27: 1, 28: 2, 29: 22, 30: 1, 31: 2, 32: 7, 33: 1, 34: 1, 35: 1, 36: 1, 37: 1, 38: 1, 39: 1, 40: 1, 41: 1, 42: 2, 43: 1, 44: 1, 45: 2, 46: 1, 47: 1, 48: 2, 49: 1, 50: 2, 51: 1, 52: 1, 53: 1, 54: 1, 55: 1, 56: 1, 57: 4, 58: 1, 59: 6, 60: 13, 61: 3, 62: 1, 63: 2, 64: 1, 65: 2, 66: 2, 67: 1, 68: 1, 69: 7, 70: 1, 71: 10, 72: 5, 73: 9, 74: 1, 75: 1, 76: 1, 77: 1, 78: 2, 79: 1, 80: 1, 81: 1, 82: 3, 83: 4, 84: 1, 85: 1, 86: 2, 87: 8, 88: 1, 89: 1, 90: 8, 91: 1, 92: 1, 93: 1, 94: 1, 95: 1, 96: 1, 97: 3, 98: 6, 99: 2, 100: 1, 101: 1, 102: 5, 103: 1, 104: 4, 105: 1, 106: 1, 107: 1, 108: 1, 109: 1, 110: 1, 111: 5, 112: 2, 113: 1, 114: 2, 115: 1, 116: 3, 117: 1, 118: 1, 119: 1, 120: 1, 121: 1, 122: 1, 123: 2, 124: 5, 125: 6, 126: 1, 127: 22, 128: 2, 129: 2, 130: 1, 131: 1, 132: 2, 133: 25, 134: 1, 135: 4, 136: 1, 137: 1, 138: 1, 139: 3, 140: 2, 141: 1, 142: 1, 143: 2, 144: 4, 145: 1, 146: 1, 147: 3, 148: 1, 149: 1, 150: 1, 151: 5, 152: 4, 153: 23, 154: 1, 155: 1, 156: 1, 157: 4, 158: 4, 159: 20, 160: 2, 161: 1, 162: 2, 163: 3, 164: 1, 165: 2, 166: 5, 167: 1, 168: 2, 169: 2, 170: 1, 171: 1, 172: 1, 173: 1, 174: 3, 175: 1, 176: 1, 177: 2, 178: 1, 179: 1, 180: 1, 181: 1, 182: 3, 183: 1, 184: 1, 185: 1, 186: 6, 187: 7, 188: 1, 189: 1, 190: 1, 191: 1, 192: 1, 193: 3, 194: 1, 195: 1, 196: 6, 197: 1, 198: 1, 199: 1, 200: 6, 201: 2, 202: 1, 203: 5, 204: 1, 205: 2, 206: 1, 207: 1, 208: 1, 209: 1, 210: 1, 211: 2, 212: 46, 213: 2, 214: 1, 215: 3, 216: 1, 217: 1, 218: 1, 219: 3, 220: 2, 221: 2, 222: 5, 223: 4, 224: 1, 225: 1, 226: 1, 227: 1, 228: 1, 229: 1, 230: 1, 231: 1, 232: 1, 233: 1, 234: 1, 235: 1, 236: 1, 237: 1, 238: 1, 239: 1, 240: 1, 241: 4, 242: 1, 243: 1, 244: 1, 245: 1, 246: 1, 247: 1, 248: 1, 249: 2, 250: 105, 251: 1, 252: 1, 253: 1, 254: 2, 255: 1, 256: 1, 257: 1, 258: 2, 259: 2, 260: 1, 261: 2, 262: 1, 263: 1, 264: 15, 265: 1, 266: 1, 267: 4, 268: 5, 269: 2, 270: 1, 271: 1, 272: 1, 273: 1, 274: 4, 275: 1, 276: 94, 277: 1, 278: 1, 279: 1, 280: 363, 281: 3, 282: 1, 283: 3, 284: 49, 285: 2, 286: 1, 287: 1, 288: 6, 289: 3, 290: 1, 291: 3, 292: 1, 293: 27, 294: 1, 295: 26, 296: 2, 297: 3, 298: 3, 299: 8, 300: 1, 301: 4, 302: 1, 303: 2, 304: 3, 305: 18, 306: 1, 307: 2, 308: 2, 309: 2, 310: 3, 311: 1, 312: 3, 313: 3, 314: 6, 315: 2, 316: 1, 317: 3, 318: 1, 319: 1, 320: 2, 321: 22, 322: 2, 323: 10, 324: 2, 325: 2, 326: 3, 327: 1, 328: 2, 329: 1, 330: 1, 331: 1, 332: 10, 333: 3, 334: 1, 335: 2, 336: 2, 337: 1, 338: 1, 339: 1, 340: 3, 341: 2, 342: 1, 343: 1, 344: 1, 345: 1, 346: 1, 347: 1, 348: 1, 349: 1, 350: 1, 351: 1, 352: 1, 353: 1, 354: 1, 355: 3, 356: 1, 357: 2, 358: 1, 359: 1, 360: 2, 361: 7, 362: 3, 363: 1, 364: 1}}

* 色彩丰富度指数计算

colors\_dominant\_entropy\_gdf=usda\_pano.colors\_entropy(colors\_dominant2cluster\_path)

3570  
  
  
100%|██████████| 566/566 [00:03<00:00, 183.03it/s]

gpd2postSQL(colors\_dominant\_entropy\_gdf,table\_name='colors\_dominant\_entropy',\*\*args.db.db\_info)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is colors\_dominant\_entropy.

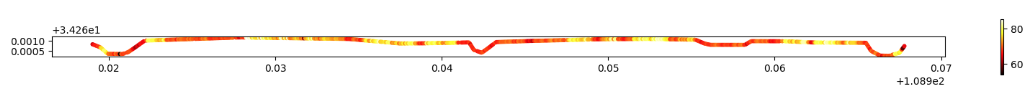
colors\_dominant\_entropy\_gdf=postSQL2gpd(table\_name='colors\_dominant\_entropy',\*\*args.db.db\_info)   
colors\_dominant\_entropy\_gdf.tail(3)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is colors\_dominant\_entropy.

|  | **fn\_stem** | **fn\_key** | **fn\_idx** | **geometry** | **counter** |
| --- | --- | --- | --- | --- | --- |
| **563** | dongxistreet\_97 | dongxistreet | 97 | POINT (108.92703 34.26116) | 69.367080 |
| **564** | dongxistreet\_98 | dongxistreet | 98 | POINT (108.92712 34.26116) | 69.022090 |
| **565** | dongxistreet\_99 | dongxistreet | 99 | POINT (108.92720 34.26116) | 68.598123 |

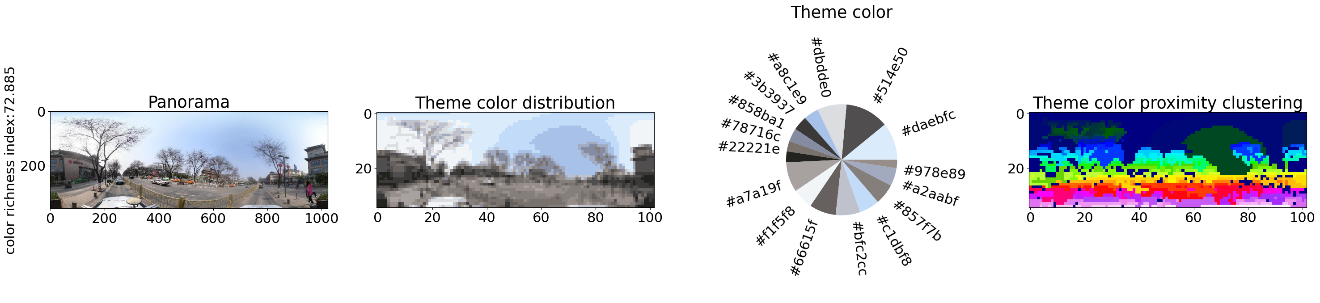
查看色彩丰富度指数的空间分布。

colors\_dominant\_entropy\_gdf.plot(column='counter',figsize=(20,1),legend=True,cmap='hot',markersize=10);



打印原始图像、主题色的分布、主题色比例和主题色的聚类分布。

fn=r'dongxistreet\_228'  
usda\_pano.dominant2cluster\_colors\_imshow(os.path.join(args.pano\_path,fn+'.jpg'),coords,colors\_dominant\_entropy\_gdf,resize\_scale=0.1,number\_of\_colors=16)



#### 4）关键点邻域大小

尺度不变特征变换（SIFT）在不同的尺度上寻找关键点。关键点邻域大小反映了对象具有的尺度特征。图中两幅全景影像提取了图像的关键点，对关键点的邻域大小划分为 (0,10], (10,20], (20,30] 和 (30,40] 等4个区间，可以描述为小尺度，中尺度，较大尺度和大尺度。当图像内的对象较小时，例如窗户的角点、招牌的边缘等，通常是实际当中较为细碎的对象；当对象如天空，成片的墙面时，所提取的边缘特征的邻域尺度则较大。因此尺度不变特征变换关键点邻域尺度大小一定程度上反应了街道视域空间繁复的程度。分别计算所有全景图像给定4个区间的频数，如果图像关键点某一区间的邻域尺度频数接近时，则这些图像在该区间具有相似尺度的对象。

该部分代码及解释可以参看*图像分类器、识别器与机器学习模型网络实验应用平台部署*部分。

* 提取图像特征，返回所有图像关键点聚类视觉词袋

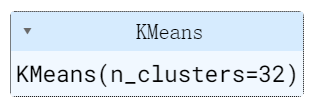
num\_cluster=32  
img\_fp\_list=glob.glob(os.path.join(args.pano\_path,'\*.jpg'))  
kmeans=usda\_pano.feature\_builder\_BOW(num\_cluster).get\_visual\_BOW(img\_fp\_list) # pip install opencv-contrib-python (pip uninstall opencv-python)

100%|██████████| 566/566 [00:15<00:00, 35.74it/s]  
  
  
start KMean...  
end KMean...

Kmeans

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

KMeans



visual\_BOW\_kmeans\_fn=args.visual\_BOW\_kmeans\_fn  
with open(visual\_BOW\_kmeans\_fn,'wb') as f:   
 pickle.dump(kmeans,f) # 存储kmeans聚类模型

* 图像的特征映射（码本映射）

with open(visual\_BOW\_kmeans\_fn,'rb') as f:  
 kmeans=pickle.load(f)   
   
feature\_map=usda\_pano.feature\_builder\_BOW(num\_cluster).get\_feature\_map(img\_fp\_list,kmeans)

100%|██████████| 566/566 [00:19<00:00, 29.23it/s]

feature\_map\_fn=args.feature\_map\_fn  
with open(feature\_map\_fn,'wb') as f:   
 pickle.dump(feature\_map,f)

* 码本映射转换为GeoDataFrame，写入数据库

with open(feature\_map\_fn,'rb') as f:  
 feature\_map=pickle.load(f)   
featureMap\_gdf=usda\_pano.kps\_desciptors\_BOW\_feature(feature\_map,coords,num\_cluster)

100%|██████████| 566/566 [00:00<00:00, 41992.75it/s]

gpd2postSQL(featureMap\_gdf,table\_name='featuremap',\*\*args.db.db\_info)

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

featureMap\_gdf=postSQL2gpd(table\_name='featuremap',\*\*args.db.db\_info)

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

featureMap\_gdf.tail(3)

|  | **fn\_stem** | **fn\_key** | **fn\_idx** | **geometry** | **0** | **1** | **2** | **3** | **4** | **5** | **...** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **563** | dongxistreet\_97 | dongxistreet | 97 | POINT (108.92703 34.26116) | 24.0 | 22.0 | 13.0 | 24.0 | 22.0 | 16.0 | ... | 10.0 | 13.0 | 19.0 | 3.0 | 10.0 | 7.0 | 14.0 | 9.0 | 26.0 | 23.0 |
| **564** | dongxistreet\_98 | dongxistreet | 98 | POINT (108.92712 34.26116) | 26.0 | 14.0 | 12.0 | 21.0 | 9.0 | 9.0 | ... | 9.0 | 11.0 | 12.0 | 3.0 | 12.0 | 15.0 | 9.0 | 14.0 | 14.0 | 17.0 |
| **565** | dongxistreet\_99 | dongxistreet | 99 | POINT (108.92720 34.26116) | 34.0 | 16.0 | 13.0 | 28.0 | 27.0 | 12.0 | ... | 11.0 | 18.0 | 16.0 | 10.0 | 7.0 | 5.0 | 9.0 | 9.0 | 13.0 | 12.0 |

3 rows × 36 columns

* 给定分割区间，统计关键点邻域尺度

bins=[0,10,20,30,40]  
kp\_size\_stats\_gdf=usda\_pano.kp\_stats(feature\_map,coords,bins)

100%|██████████| 566/566 [00:04<00:00, 139.08it/s]

gpd2postSQL(kp\_size\_stats\_gdf,table\_name='kp\_size\_stats',\*\*args.db.db\_info)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is kp\_size\_stats.

kp\_size\_stats\_gdf=postSQL2gpd(table\_name='kp\_size\_stats',\*\*args.db.db\_info)

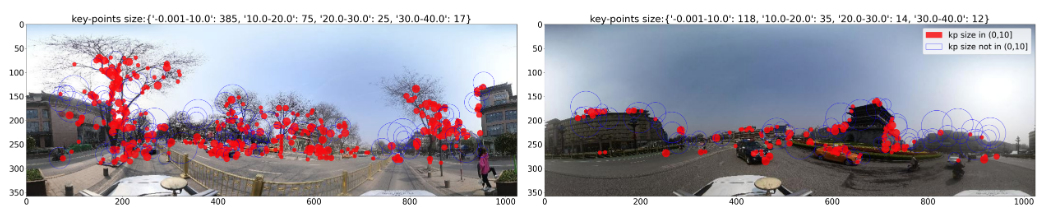
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is kp\_size\_stats.

kp\_size\_stats\_gdf.tail(3)

|  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** | **num** | **fn\_stem** | **fn\_key** | **fn\_idx** | **geometry** | **-0.001\_10.0** | **10.0\_20.0** | **30.0\_40.0** | **20.0\_30.0** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **563** | 445.0 | 11.051685 | 8.464527 | 4.0 | 4.0 | 8.0 | 16.0 | 32.0 | 445 | dongxistreet\_97 | dongxistreet | 97 | POINT (108.92703 34.26116) | 261 | 104 | 37 | 43 |
| **564** | 395.0 | 10.881013 | 8.517138 | 4.0 | 4.0 | 8.0 | 16.0 | 32.0 | 395 | dongxistreet\_98 | dongxistreet | 98 | POINT (108.92712 34.26116) | 249 | 77 | 35 | 34 |
| **565** | 464.0 | 10.741379 | 8.445604 | 4.0 | 4.0 | 8.0 | 12.0 | 32.0 | 464 | dongxistreet\_99 | dongxistreet | 99 | POINT (108.92720 34.26116) | 292 | 98 | 43 | 31 |

特征点邻域大小比较示例

kp\_stats\_example\_fn=['dongxistreet\_228','dongxistreet\_269']  
img\_path=args.pano\_path  
usda\_pano.kp\_show(feature\_map,kp\_stats\_example\_fn,img\_path)



### 3.6.2.4 不同邻里尺度最优簇数选择和指数贡献度

AgglomerativeClustering层次聚类算法中connectivity参数可以为每个样本指定连接矩阵（Connectivity matrix）。该连接矩阵可以通过kneighbors\_graph方法，给定每个样本的邻元数计算。越少的邻元数，在聚类中所考虑的邻里关系相对越少，即邻里尺度趋于小范围；反之则越多，邻里尺度趋于大范围。不同邻里尺度下对给定多个街道空间景观指数聚类，计算指数特征的贡献度不仅可以发现主导街区特征的景观指数和对街区特征形成影响较小的景观指数，还可以发现指数作用不同邻里尺度的变化。

AgglomerativeClustering方法需要指定聚类的数量，给定KElbowVisualizer方法计算的拐点作为最有簇数

panorama\_object\_percent\_gdf=postSQL2gpd(table\_name='cube\_object\_percent',\*\*args.db.db\_info)  
sky\_class\_level\_metrics\_gdf=postSQL2gpd(table\_name='metrics\_skyline\_shape',\*\*args.db.db\_info)  
colors\_dominant\_entropy\_gdf=postSQL2gpd(table\_name='colors\_dominant\_entropy',\*\*args.db.db\_info)  
kp\_size\_stats\_gdf=postSQL2gpd(table\_name='kp\_size\_stats',\*\*args.db.db\_info)  
  
metrics\_panorama\_object\_percent=['vege','sky','ground','equilibrium\_degree']  
metrics\_sky\_class\_level\_metrics=['number\_of\_patches','perimeter\_area\_ratio\_mn','shape\_index\_mn','fractal\_dimension\_mn']  
metrics\_colors\_dominant\_entropy=['counter']  
metrics\_kp\_size\_stats=['-0.001\_10.0', '10.0\_20.0','30.0\_40.0', '20.0\_30.0']   
metrics\_auxiliary=['fn\_stem', 'fn\_key', 'fn\_idx', 'geometry']  
  
metrics\_merge=pd.concat([panorama\_object\_percent\_gdf[metrics\_panorama\_object\_percent],  
 sky\_class\_level\_metrics\_gdf[metrics\_sky\_class\_level\_metrics],  
 colors\_dominant\_entropy\_gdf[metrics\_colors\_dominant\_entropy],  
 kp\_size\_stats\_gdf[metrics\_kp\_size\_stats],  
 panorama\_object\_percent\_gdf[metrics\_auxiliary]],axis=1)   
fields\_mapping= {  
 'vege':'Green view index',   
 'sky':'Sky view factor',   
 'ground':'Ground view index',   
 'equilibrium\_degree':'Equilibrium degree',  
 'number\_of\_patches':'number of patches',   
 'perimeter\_area\_ratio\_mn':'Perimeter area ratio(mn)',   
 'shape\_index\_mn':'Shape index(mn)',  
 'fractal\_dimension\_mn':'Fractal dimension(mn)',   
 'counter':'Color richness index',   
 '-0.001\_10.0':'Key point size(0-10]',   
 '10.0\_20.0':'Key point size(10-20]',  
 '30.0\_40.0':'Key point size(30-40]',   
 '20.0\_30.0':'Key point size(20-30]'}   
metrics\_merge=metrics\_merge.rename(columns=fields\_mapping)  
metrics\_merge\_gdf=gpd.GeoDataFrame(metrics\_merge,crs=panorama\_object\_percent\_gdf.crs,geometry='geometry')  
  
gpd2postSQL(metrics\_merge\_gdf,table\_name='metrics\_merge',\*\*args.db.db\_info)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is cube\_object\_percent.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is metrics\_skyline\_shape.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is colors\_dominant\_entropy.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is kp\_size\_stats.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is metrics\_merge.

metrics\_merge\_gdf=postSQL2gpd(table\_name='metrics\_merge',\*\*args.db.db\_info)  
metrics\_merge\_gdf.tail(3)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is metrics\_merge.

|  | **Green view index** | **Sky view factor** | **Ground view index** | **Equilibrium degree** | **number of patches** | **Perimeter area ratio(mn)** | **Shape index(mn)** | **Fractal dimension(mn)** | **Color richness index** | **Key point size(0-10]** | **Key point size(10-20]** | **Key point size(30-40]** | **Key point size(20-30]** | **fn\_stem** | **fn\_key** | **fn\_idx** | **geometry** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **563** | 0.521933 | 29.560383 | 53.860950 | 50.245055 | 250.0 | 331.851991 | 1.123028 | 1.018418 | 69.367080 | 261 | 104 | 37 | 43 | dongxistreet\_97 | dongxistreet | 97 | POINT (108.92703 34.26116) |
| **564** | 1.563883 | 29.791083 | 54.351650 | 58.475244 | 195.0 | 296.252504 | 1.206710 | 1.025840 | 69.022090 | 249 | 77 | 35 | 34 | dongxistreet\_98 | dongxistreet | 98 | POINT (108.92712 34.26116) |
| **565** | 2.070233 | 30.658100 | 51.518967 | 53.639539 | 221.0 | 296.170397 | 1.235508 | 1.028000 | 68.598123 | 292 | 98 | 43 | 31 | dongxistreet\_99 | dongxistreet | 99 | POINT (108.92720 34.26116) |

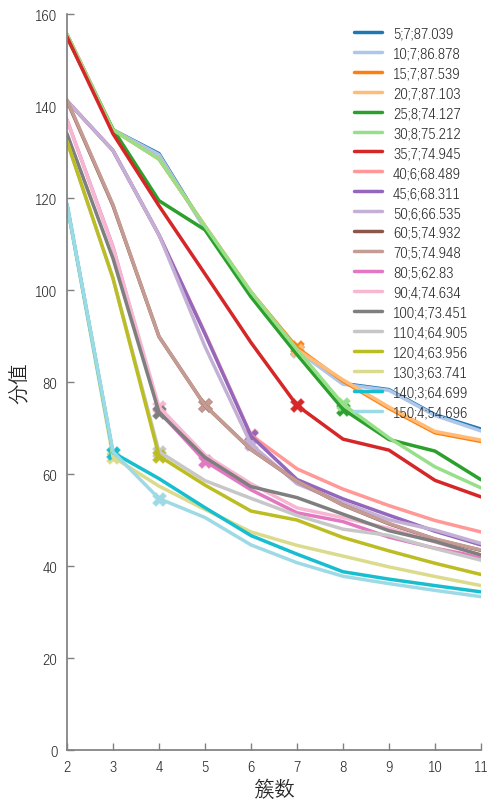
* 不同邻里尺度（邻元数）最优簇数计算-多指数

通过给定多个簇数（聚类数）分别聚类，计算每个点到其所属聚类中心的平方距离之和(distortion score)，曲线拐点即为最优的簇数选择。从下述计算结果来看，各个不同邻里尺度的最优簇数不尽相同。

fields=['Green view index','Sky view factor', 'Ground view index','Equilibrium degree', 'Perimeter area ratio(mn)','Shape index(mn)', 'Fractal dimension(mn)', 'Color richness index','Key point size(0-10]', 'Key point size(10-20]','Key point size(30-40]', 'Key point size(20-30]',]  
kneighbors\_graph\_n\_neighbors\_list=[5,10,15,20,25,30,35,40,45,50,60,70,80,90,100,110,120,130,140,150]  
distortion\_score\_save\_path='../graph'  
k=(2,12)  
  
elbow\_score\_dict=usda\_pano.distortion\_score\_elbow\_kneighbors(metrics\_merge\_gdf,fields,distortion\_score\_save\_path,kneighbors\_graph\_n\_neighbors\_list,k)

100%|██████████| 20/20 [00:44<00:00, 2.22s/it]  
  
  
[118.69348062107632, 64.84605180802042, 54.69607692661462, 50.580944380733044, 44.62354172738738, 40.754705891336904, 37.80621351461414, 36.2090554105516, 34.73579086012199, 33.37959313559141] 4 54.69607692661462

usda\_pano.elbow\_score\_plot(elbow\_score\_dict,k,figsize=(6, 8),ylim=(0,160),legend\_loc='upper right')



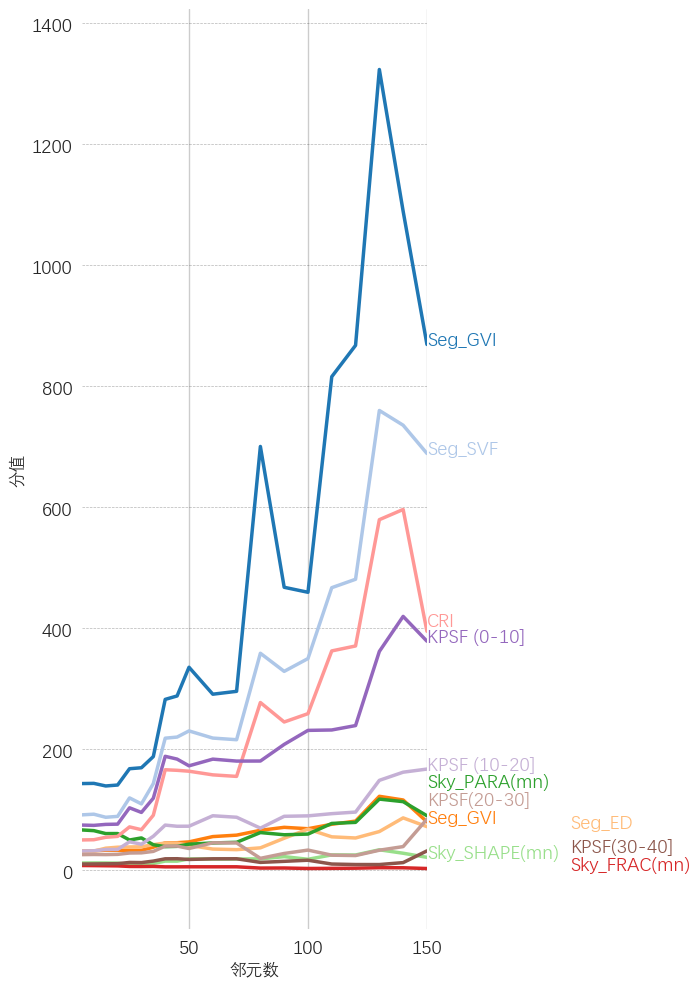
* 不同邻里尺度景观指数贡献度

给定不同邻里尺度的连接矩阵，和对应的最优簇数，通过特征选择（SelectKBest）方法，计算给定连续邻元数下景观指数贡献度及其变化。计算结果显示，不同邻里尺度下Seg\_GVI基本占绝对优势，其次为Seg\_SVF、CRI和KPSF(0-10]。

featureScores\_dict=usda\_pano.idxes\_clustering\_contribution\_kneighbors(metrics\_merge\_gdf,fields,elbow\_score\_dict)

100%|██████████| 20/20 [00:06<00:00, 3.18it/s]

x\_offsets={ 'Fractal dimension(mn)':60,'Key point size(30-40]':60,'Equilibrium degree':60}  
y\_offsets={'Perimeter area ratio(mn)':50,'Key point size(20-30]':25,'Color richness index':10}  
usda\_pano.idxes\_clustering\_contribution\_kneighbors\_plot(featureScores\_dict,figsize=(5, 10),x\_offsets=x\_offsets,y\_offsets=y\_offsets)



### 3.6.2.5 街道行业分类服务空间组成结构

每一位置点建立 500m 的缓冲区域，提取落于缓冲区内的POI点数据。计算每一位置点对应的POI点数量，信息熵和均衡度。

clipped\_pois\_fn=r'G:\data\POI\_dongxistreet\clipped\_poi\_gdf.gpkg'  
poi\_gdf=gpd.read\_file(clipped\_pois\_fn)  
print(poi\_gdf.crs)

PROJCS["WGS 84 / UTM zone 49N",GEOGCS["WGS 84",DATUM["World Geodetic System 1984",SPHEROID["WGS 84",6378137,298.257223563]],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433],AUTHORITY["EPSG","4326"]],PROJECTION["Transverse\_Mercator"],PARAMETER["latitude\_of\_origin",0],PARAMETER["central\_meridian",111],PARAMETER["scale\_factor",0.9996],PARAMETER["false\_easting",500000],PARAMETER["false\_northing",0],UNIT["metre",1],AXIS["Easting",EAST],AXIS["Northing",NORTH],AUTHORITY["EPSG","32649"]]

coordi\_df=metrics\_merge\_gdf.to\_crs(poi\_gdf.crs)

pos\_poi\_idxes\_gdf,pos\_poi\_feature\_vector\_gdf=usda\_pano.street\_poi\_structure(poi=poi\_gdf,position=coordi\_df,distance=500)

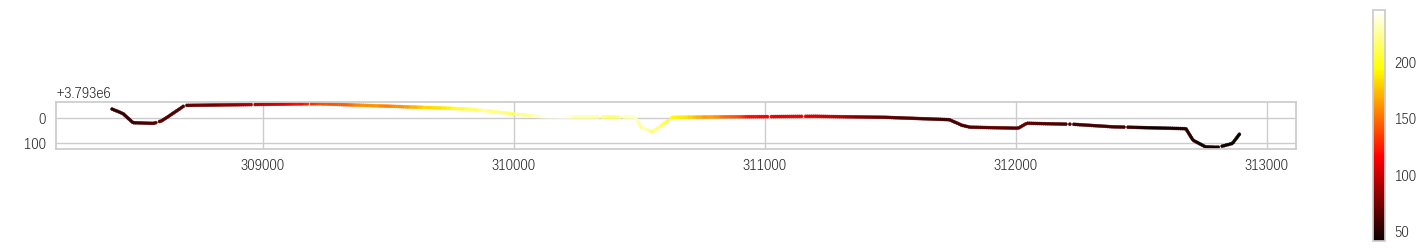
100%|██████████| 566/566 [01:31<00:00, 6.17it/s]

pos\_poi\_idxes\_gdf.tail(3)

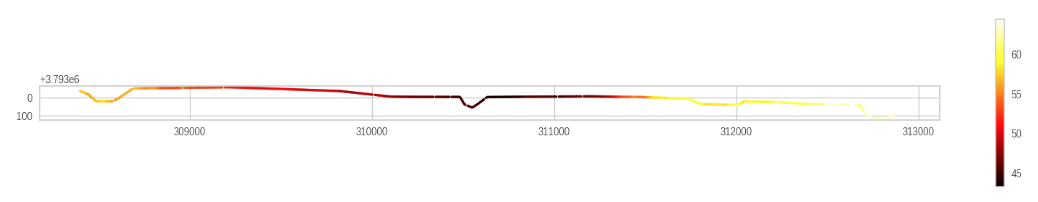
|  | **geometry** | **frank\_e** | **num** | **num\_diff** |
| --- | --- | --- | --- | --- |
| **563** | POINT (309138.206 3793057.251) | 52.488255 | 121 | 1.0 |
| **564** | POINT (309146.205 3793057.353) | 52.861526 | 121 | 0.0 |
| **565** | POINT (309154.205 3793057.456) | 52.838794 | 119 | -2.0 |

打印 POI 数量列num和信息熵列frank\_e，可以看到以钟楼区域及其西侧（回民街）数量最多，街道空间提供的服务越多。而均衡度最大的区域在POI数量最少的区域，是因为主要的行业分类，例如购物、生活服务、美食和酒店等在该区域分布较少，减少了不同行业数量的差异。

pos\_poi\_idxes\_gdf.plot(column='num',cmap='hot',figsize=(20,3),markersize=5,legend=True);



pos\_poi\_idxes\_gdf.plot(column='frank\_e',cmap='hot',figsize=(20,3),markersize=5,legend=True);



pos\_poi\_feature\_vector\_gdf为每一位置点给定范围内， POI 一级分类频数。

pos\_poi\_feature\_vector\_gdf.tail(3)

|  | **geometry** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **...** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **563** | POINT (309138.206 3793057.251) | 7.0 | 4.0 | 1.0 | 1.0 | 13.0 | 15.0 | 11.0 | 4.0 | 6.0 | ... | 1.0 | 10.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **564** | POINT (309146.205 3793057.353) | 7.0 | 5.0 | 1.0 | 1.0 | 13.0 | 15.0 | 10.0 | 4.0 | 6.0 | ... | 2.0 | 10.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **565** | POINT (309154.205 3793057.456) | 7.0 | 5.0 | 1.0 | 1.0 | 13.0 | 14.0 | 10.0 | 4.0 | 6.0 | ... | 2.0 | 11.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

3 rows × 26 columns

gpd2postSQL(pos\_poi\_idxes\_gdf,table\_name='pos\_poi\_idxes',\*\*args.db.db\_info)   
gpd2postSQL(pos\_poi\_feature\_vector\_gdf,table\_name='pos\_poi\_feature\_vector',\*\*args.db.db\_info)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is pos\_poi\_idxes.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is pos\_poi\_feature\_vector.

合并全景图视域景观指数和 POI 相关指数，并写入数据库。

cols=['frank\_e','num','num\_diff']  
metrics\_merge\_gdf[cols]=pos\_poi\_idxes\_gdf[cols]

gpd2postSQL(metrics\_merge\_gdf,table\_name='metrics\_inall',\*\*args.db.db\_info)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The GeoDataFrame has been written to the PostgreSQL database.The table name is metrics\_inall.

### 3.6.2.6 建立特征指数的 LSTM 时空预测模型

使用 LSTM 网络结构构建预测模型。首先读取前文处理的视域空间指数和 POI 指数数据，移除可能存在的空值，用fn\_idx字段按照东西大街空间顺序排序。

metrics\_inall=postSQL2gpd(table\_name='metrics\_inall',\*\*args.db.db\_info)  
metrics\_inall.sort\_values(by=['fn\_idx'],inplace=True)  
metrics\_inall.reset\_index(drop=True,inplace=True)  
metrics\_inall.dropna(inplace=True)  
metrics\_inall.head(3)

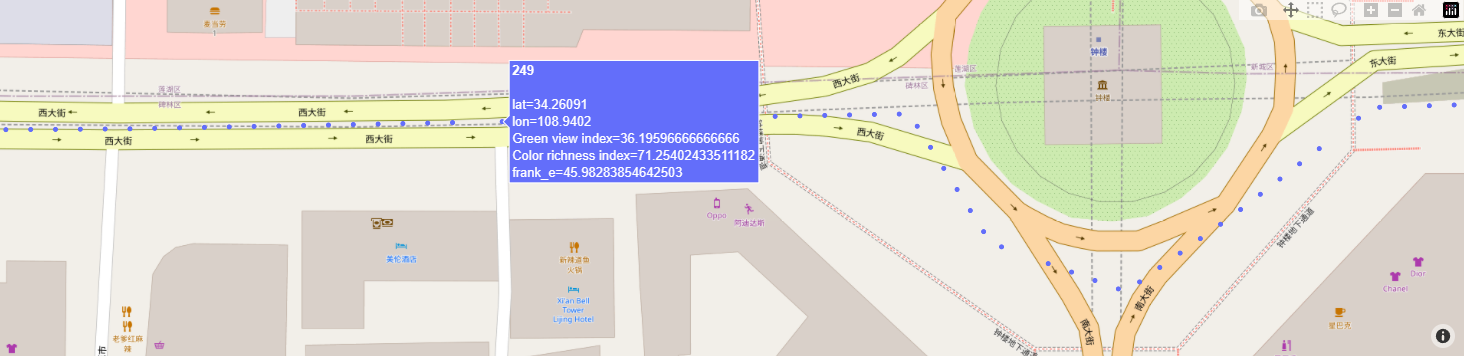
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The data has been read from PostgreSQL database. The table name is metrics\_inall.

|  | **Green view index** | **Sky view factor** | **Ground view index** | **Equilibrium degree** | **number of patches** | **Perimeter area ratio(mn)** | **Shape index(mn)** | **Fractal dimension(mn)** | **Color richness index** | **Key point size(0-10]** | **Key point size(10-20]** | **Key point size(30-40]** | **Key point size(20-30]** | **fn\_stem** | **fn\_key** | **fn\_idx** | **geometry** | **frank\_e** | **num** | **num\_diff** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 4.988783 | 39.721033 | 50.634883 | 48.372925 | 282.0 | 261.505874 | 1.228234 | 1.033018 | 65.470208 | 159 | 44 | 22 | 22 | dongxistreet\_1 | dongxistreet | 1 | POINT (108.91910 34.26081) | 57.426584 | 55 | 1.0 |
| **2** | 5.789183 | 39.835183 | 50.661900 | 45.381630 | 295.0 | 266.110274 | 1.199734 | 1.031069 | 67.015989 | 144 | 44 | 17 | 32 | dongxistreet\_2 | dongxistreet | 2 | POINT (108.91918 34.26078) | 57.426584 | 55 | -164.0 |
| **3** | 7.378683 | 39.334867 | 49.886200 | 46.898469 | 286.0 | 260.221003 | 1.191649 | 1.029841 | 66.440458 | 157 | 45 | 14 | 19 | dongxistreet\_3 | dongxistreet | 3 | POINT (108.91927 34.26076) | 57.426584 | 55 | -128.0 |

使用Plotly库打印交互式图表，查看具体细节信息。

metrics\_inall\_copy=metrics\_inall.copy(deep=True)  
metrics\_inall\_copy[['lat','lon']]=metrics\_inall\_copy.apply(lambda row:[row.geometry.y,row.geometry.x],axis=1,result\_type='expand')

fig = px.scatter\_mapbox(metrics\_inall\_copy, lat="lat", lon="lon", hover\_name="fn\_idx", hover\_data=['Green view index','Color richness index','frank\_e'],zoom=14, height=300)  
fig.update\_layout(mapbox\_style="open-street-map")  
fig.update\_layout(margin={"r":0,"t":0,"l":0,"b":0})  
fig.show()



定义normalize\_by\_meanNstd()函数标准化数据，公式为 vμσ为数据的标准差。打印标准化后的数据，查看各列（指数）曲线变化。

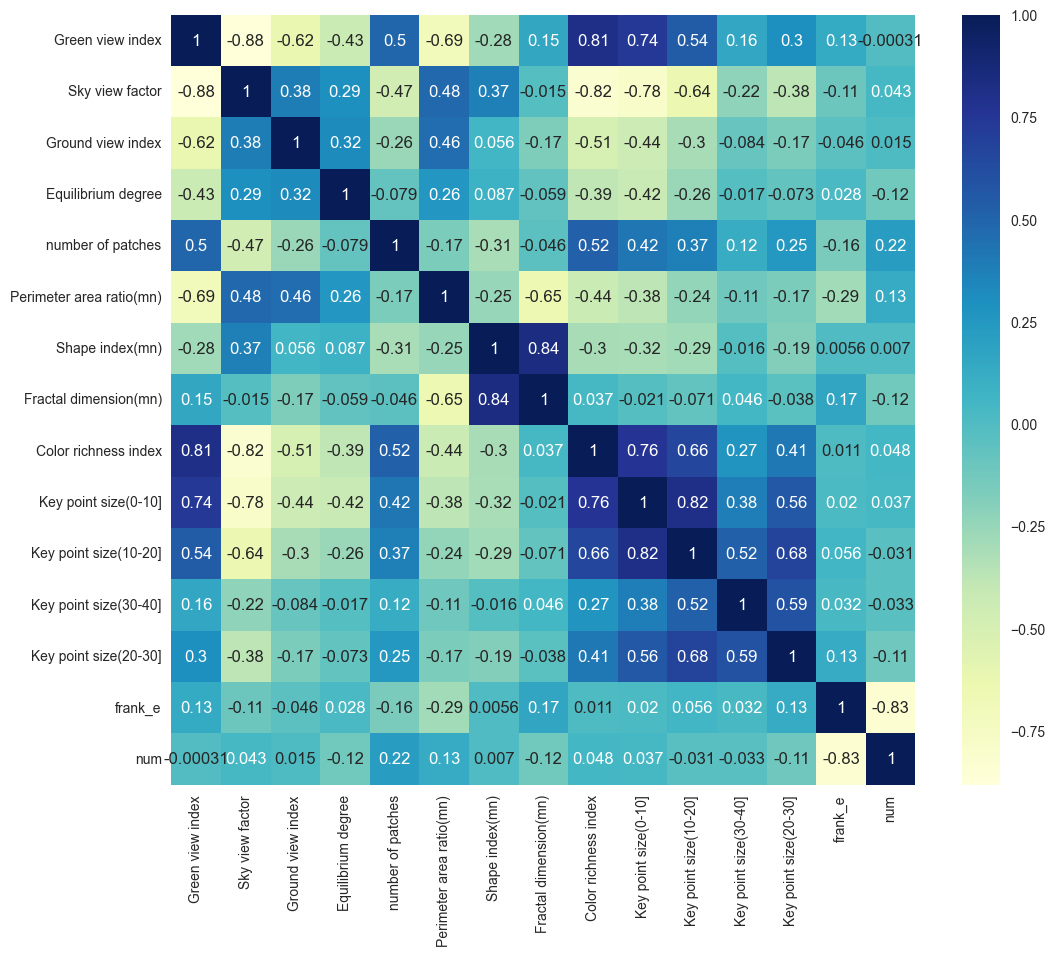
cols=['Green view index', 'Sky view factor', 'Ground view index','Equilibrium degree', 'number of patches', 'Perimeter area ratio(mn)','Shape index(mn)', 'Fractal dimension(mn)', 'Color richness index',  
 'Key point size(0-10]', 'Key point size(10-20]','Key point size(30-40]', 'Key point size(20-30]','frank\_e', 'num']  
  
norm\_df,mean\_std\_dict=usda\_utils.normalize\_by\_meanNstd(metrics\_inall[cols])  
print(mean\_std\_dict)  
norm\_df.plot.line(figsize=(20,20),subplots=True);

{'Green view index': [19.923202684365783, 16.43173153981058], 'Sky view factor': [22.42920268436578, 9.705913947185682], 'Ground view index': [49.61750383480826, 3.458661094856522], 'Equilibrium degree': [49.2884185074578, 4.362935945874506], 'number of patches': [250.99469026548672, 67.83167767775927], 'Perimeter area ratio(mn)': [270.536192974786, 29.428301484843914], 'Shape index(mn)': [1.1646643107968098, 0.04053803041743256], 'Fractal dimension(mn)': [1.0259775695134745, 0.00417126374694389], 'Color richness index': [72.11344856579554, 5.1726015268900865], 'Key point size(0-10]': [296.6725663716814, 111.91842920496616], 'Key point size(10-20]': [67.7575221238938, 23.25421415827134], 'Key point size(30-40]': [21.08495575221239, 7.3114505216344154], 'Key point size(20-30]': [27.833628318584072, 9.241476911181776], 'frank\_e': [53.119824785230435, 6.098244214407541], 'num': [118.43362831858407, 65.38189450866699]}



查看各指数间的相关系数，为指数的进一步删选提供依据。

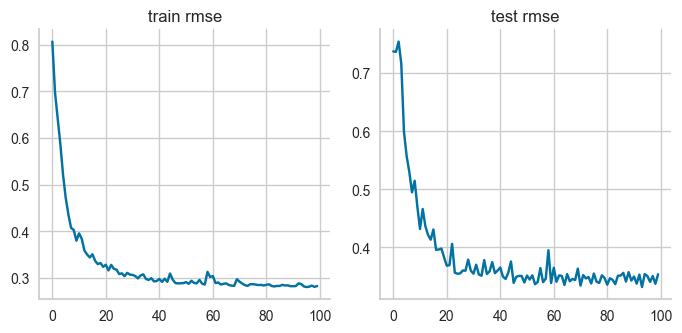
fig, ax=plt.subplots(figsize=(12, 10))  
sb.heatmap(norm\_df.corr(),cmap='YlGnBu',annot=True,ax=ax)  
plt.show()



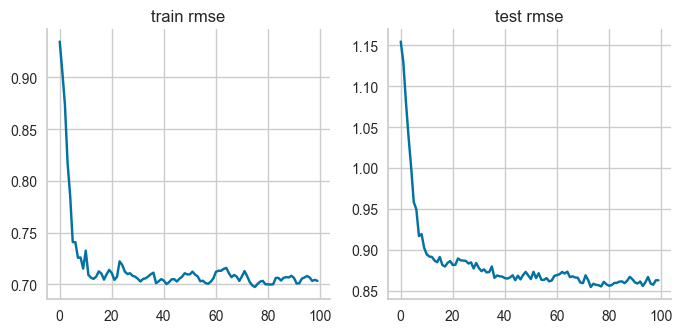
提取测试部分指数，并逐一训练各个指数的 LSTM 网络模型，且将训练的各模型保存至本地磁盘。在训练时，根据前文指数变化曲线，和训练均方误差曲线变化，及打印的预测曲线来调整配置窗口window\_size的大小。

cols\_selection=['Green view index','Equilibrium degree','Perimeter area ratio(mn)', 'Color richness index','Key point size(0-10]','frank\_e' ]  
test\_size=100  
batch\_size=64  
  
window\_size\_dict={'Green view index':10,'Equilibrium degree':1,'Perimeter area ratio(mn)':5, 'Color richness index':10,'Key point size(0-10]':1,'frank\_e':5}  
save\_path=r'G:\model\_ckpts\spatial\_metrics\_lstm\spatial\_metrics\_lst.pt'  
model\_state\_dict={}  
for col in cols\_selection:   
 print('-'\*10,col)  
 window\_size=window\_size\_dict[col]  
 train\_data\_dict,test\_data\_dict,train\_loader\_dict,test\_loader\_dict=usda\_utils.create\_sequence2series\_datasetNdataloader\_from\_df(norm\_df,cols\_selection,window\_size=window\_size)   
 model=usda\_models.Simple\_LSTM(input\_size=window\_size,output\_size=1,hidden\_size=128)  
 train\_rmse\_dict, test\_rmse\_dict=usda\_models.spatial\_metrics\_train(model,train\_loader\_dict[col],train\_data\_dict[col],test\_data\_dict[col],n\_epochs=100,verbose=False)   
 usda\_models.train\_test\_rmse\_plot(train\_rmse\_dict,test\_rmse\_dict)  
 model\_state\_dict[col]=model.state\_dict()  
   
torch.save(model\_state\_dict,save\_path)

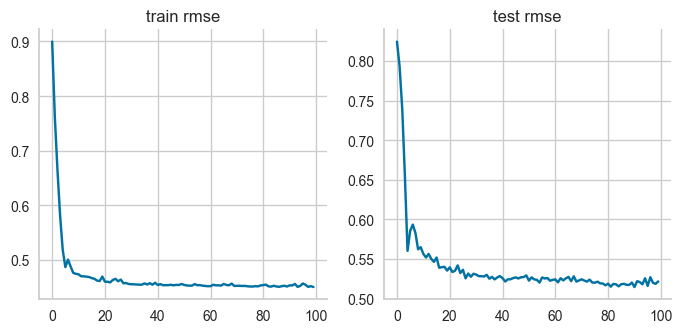
---------- Green view index  
Complete training.



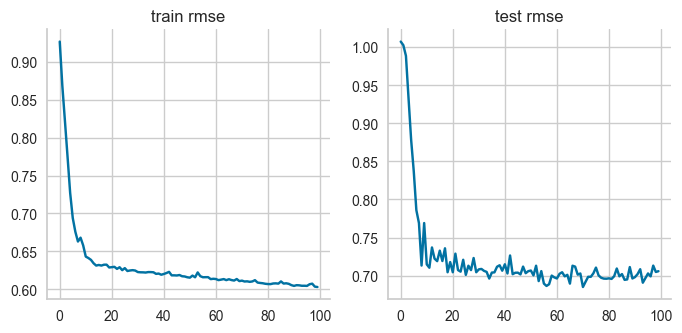
---------- Equilibrium degree  
Complete training.



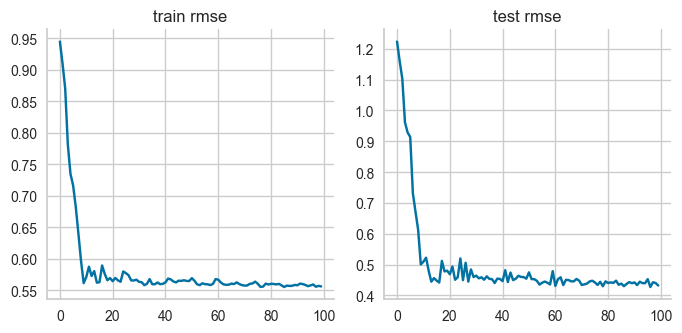
---------- Perimeter area ratio(mn)  
Complete training.



---------- Color richness index  
Complete training.



---------- Key point size(0-10]  
Complete training.



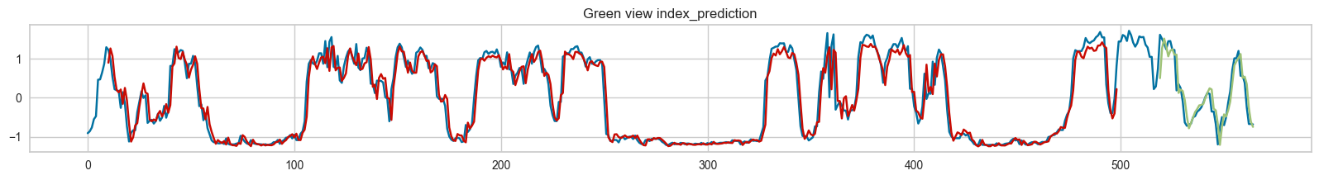
---------- frank\_e  
Complete training.



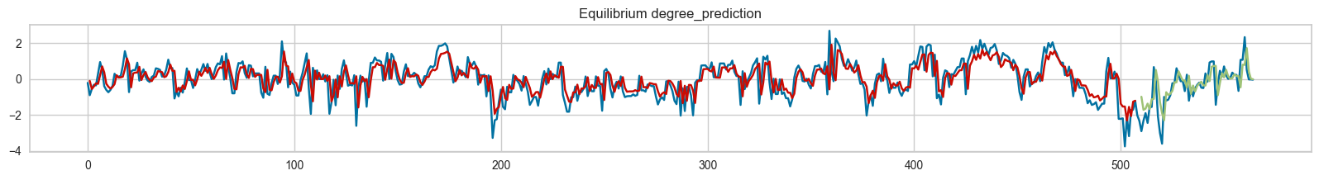
打印指数真实曲线和预测曲线，观察训练的模型预测能力。

spatial\_metrics\_lst=torch.load(save\_path)  
for col in cols\_selection:  
 window\_size=window\_size\_dict[col]  
 train\_data\_dict,test\_data\_dict,train\_loader\_dict,test\_loader\_dict=usda\_utils.create\_sequence2series\_datasetNdataloader\_from\_df(norm\_df,cols\_selection,window\_size=window\_size)   
 trained\_model=usda\_models.Simple\_LSTM(input\_size=window\_size,output\_size=1,hidden\_size=128)  
 trained\_model.load\_state\_dict(spatial\_metrics\_lst[col])  
 print(trained\_model)  
 usda\_models.timeseries\_pred\_plot(trained\_model,norm\_df[col],train\_data\_dict[col],test\_data\_dict[col],window\_size,title=f'{col}\_prediction')

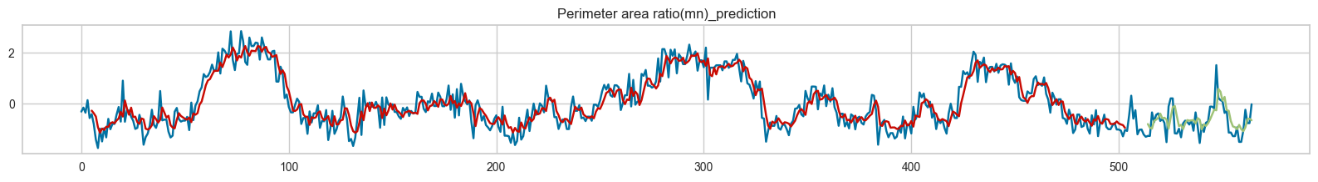
Simple\_LSTM(  
 (lstm): LSTM(10, 128, batch\_first=True)  
 (linear): Linear(in\_features=128, out\_features=1, bias=True)  
)



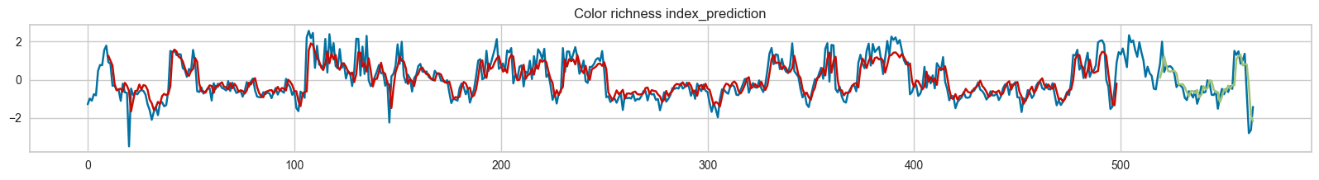
Simple\_LSTM(  
 (lstm): LSTM(1, 128, batch\_first=True)  
 (linear): Linear(in\_features=128, out\_features=1, bias=True)  
)



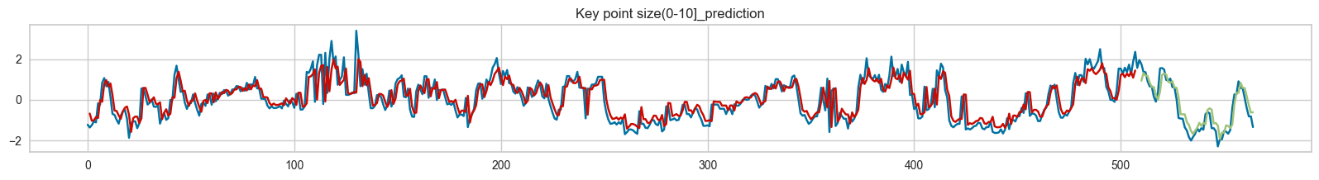
Simple\_LSTM(  
 (lstm): LSTM(5, 128, batch\_first=True)  
 (linear): Linear(in\_features=128, out\_features=1, bias=True)  
)



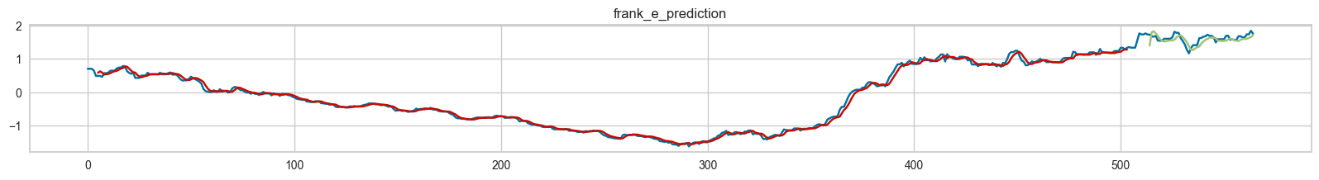
Simple\_LSTM(  
 (lstm): LSTM(10, 128, batch\_first=True)  
 (linear): Linear(in\_features=128, out\_features=1, bias=True)  
)



Simple\_LSTM(  
 (lstm): LSTM(1, 128, batch\_first=True)  
 (linear): Linear(in\_features=128, out\_features=1, bias=True)  
)



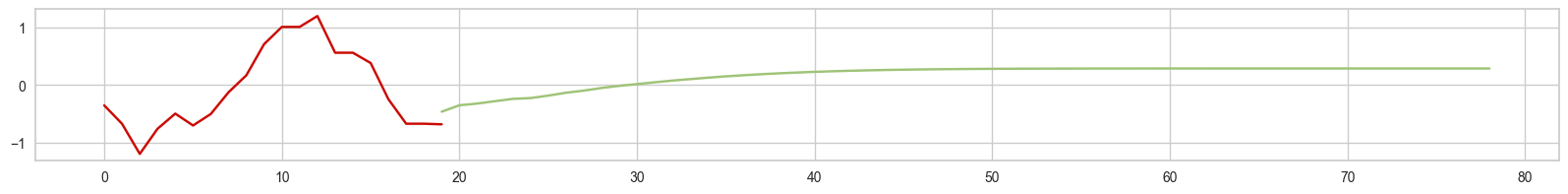
Simple\_LSTM(  
 (lstm): LSTM(5, 128, batch\_first=True)  
 (linear): Linear(in\_features=128, out\_features=1, bias=True)  
)



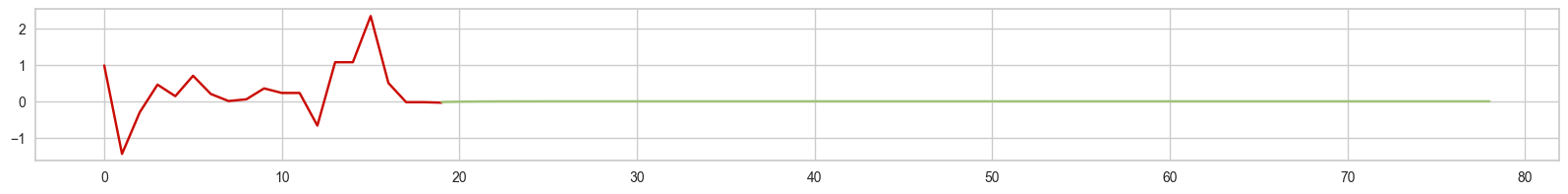
模型训练时，均方误差的计算是用连续切分的指数值片段（窗口）预测紧邻的下一个空间的指数值，且每一指数片段均采样自指数的真实值。如果构建一个类似文本生成的试验，则把每一个新预测的下一空间的指数值追加到已有指数序列下，用于再下一空间指数值的预测，周而复始，不断生成连续空间的指数值。从下述计算打印结果来看，通常临近真实值小部分空间区域指数顺势变化，但随着空间的延长，模型已经不再有变化预测能力，曲线变成直线。这与本次试验有限的样本量、简单架构的 LSTM 网络结构有关。

spatial\_metrics\_lst=torch.load(save\_path)  
generation\_dict={}  
for col in cols\_selection:  
 print('-'\*10,col)  
 window\_size=window\_size\_dict[col]  
 trained\_model=usda\_models.Simple\_LSTM(input\_size=window\_size,output\_size=1,hidden\_size=128)  
 trained\_model.load\_state\_dict(spatial\_metrics\_lst[col])  
 pred\_lst,X=usda\_models.timeseries\_pred\_generation(trained\_model,metrics\_inall[col][-20:],window\_size,gen\_length=60,norm=mean\_std\_dict[col])   
 generation\_dict[col]=[pred\_lst,X]  
   
 usda\_models.timeseries\_pred\_generation\_plot(X,pred\_lst)

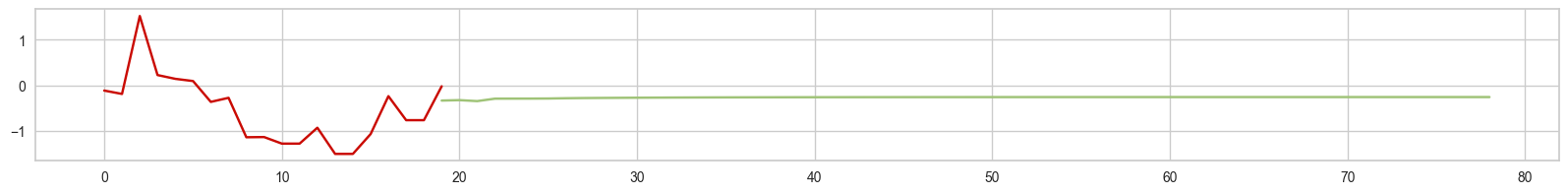
---------- Green view index



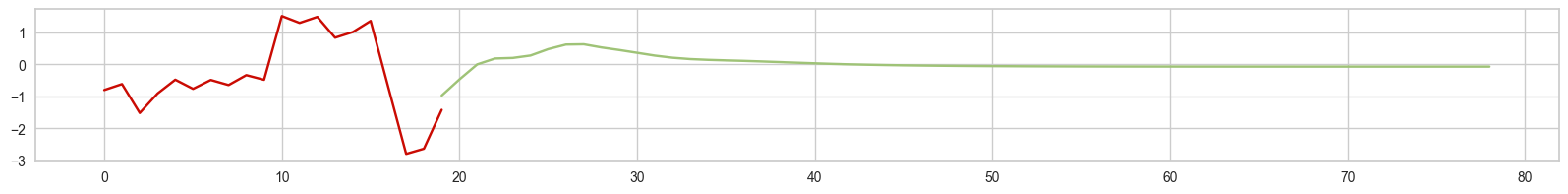
---------- Equilibrium degree



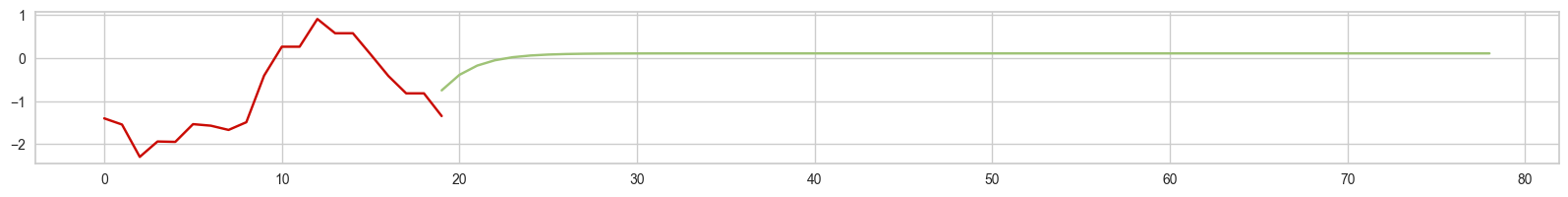
---------- Perimeter area ratio(mn)



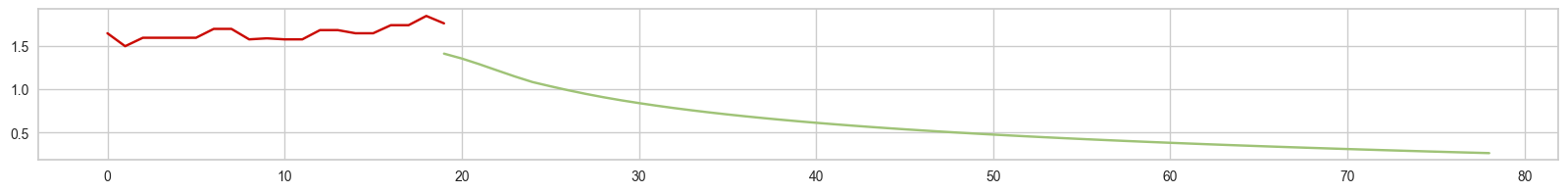
---------- Color richness index



---------- Key point size(0-10]



---------- frank\_e



注释（Notes）：

① 全球土地覆盖产品（WorldCover） 2020年，（<https://worldcover2020.esa.int/>）。

② 全球土地覆盖产品（WorldCover） 2021年，（<https://worldcover2021.esa.int/>）。

③ WorldCover数据产品使用手册（Product User Manual），（<https://github.com/microsoft/PlanetaryComputerExamples/blob/main/datasets/esa-worldcover/esa-worldcover-example.ipynb>）。

④ Planetary Computer Hub（PcHub），（<https://github.com/microsoft/PlanetaryComputerExamples/blob/main/datasets/esa-worldcover/esa-worldcover-example.ipynb>）。

⑤ Google Earth，（<https://www.google.com/earth/versions/>）。

⑥ 百度地图开放平台，（<https://lbsyun.baidu.com/index.php?title=%E9%A6%96%E9%A1%B5>）。

⑦ PASS（ERF-PSPNet）深度学习模型，（<https://github.com/elnino9ykl/PASS>）。

⑧ PASS GitHub 代码仓库，（<https://github.com/elnino9ykl/PASS/tree/master/trained_models>）。

⑨ Oculus 360 度视频流测量研究成果，（<https://github.com/bingsyslab/360projection>）。

⑩ Making a Little Planet Panorma using Python and Scikit Image，（<https://richwareham.com/little-planet-projection/>）。

⑪ mayavi 三维科学数据可视化绘图工具，（<https://docs.enthought.com/mayavi/mayavi/>）。

参考文献（References）:

[1] Accessing ESA WorldCover classification data with the Planetary Computer STAC API,<https://github.com/microsoft/PlanetaryComputerExamples/blob/main/datasets/esa-worldcover/esa-worldcover-example.ipynb>

[2] Yang, K., Hu, X., Bergasa, L. M., Romera, E., & Wang, K. (2020). PASS: Panoramic Annular Semantic Segmentation. IEEE Transactions on Intelligent Transportation Systems, 21(10), 4171–4185. doi:10.1109/tits.2019.2938965

[3] Zhou, C., Li, Z., & Liu, Y. (2017). A Measurement Study of Oculus 360 Degree Video Streaming. Proceedings of the 8th ACM on Multimedia Systems Conference - MMSys’17. doi:10.1145/3083187.3083190