## 2.5.1 聚类与城市色彩

### 2.5.1.1 调研图像

#### 1）用手机App记录调研路径

如果区域调研的位置精度要求不是很高，可以在手机的应用（Application，App）中搜索GPS追踪（tracker）用于调研路线的记录。不同的应用存储的文件格式可能不同，例如本例中将调研路线存储为KML格式文件。KML全称Keyhole Markup Language，是基于XML（eXtensible Markup Language，可扩展标记语言）语法标准的一种标记语言（markup language），采用标记结构，含有嵌套的元素和属性。KML通常应用于Google地球相关软件中，例如Google Earth、Google Map、 Google Maps for mobile等，用于显示数据（包括点、线、面、多边形、多面体及模型等）。KML文件可以用文本编辑器打开，例如[Notepad++](https://notepad-plus-plus.org/)。下述例举了GPS跟踪KML文件开头部分内容：

<?xml version="1.0" encoding="UTF-8"?>  
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext/2.2" xmlns:kml="http://www.opengis.net/kml/2.2" xmlns:atom="http://www.w3.org/2005/Atom">  
<Document>  
<name>default\_20170720081441</name>  
<open>1</open>  
<description>线路开始时间：2017-07-20 08:14:41,结束时间：2017-07-20 20:53:03,线路长度：197801。由GPS工具箱导出。</description>  
<Style id="yellowLineGreenPoly" >  
 <LineStyle>  
 <color>7f00ffff</color>  
 <width>4</width>  
 </LineStyle>  
 <PolyStyle>  
 <color>7f00ff00</color>  
 </PolyStyle>  
 </Style>  
<Folder>  
<name>线路标记点</name>  
 <Placemark>  
 <name>线路起点</name>  
 <description><![CDATA[2017-07-20 08:14:41]]></description>  
 <Point>  
 <coordinates>120.132007,30.300508,9.7</coordinates>  
 </Point>  
 <markerStyle>-1</markerStyle>  
 </Placemark>  
 <Placemark>  
 <name>线路追踪路径</name>  
 <visibility>1</visibility>  
 <description>GPS工具箱导出数据</description>  
 <styleUrl>#yellowLineGreenPoly</styleUrl>  
 <LineString>  
 <tessellate>1</tessellate>  
 <coordinates>   
 120.130187,30.211812,18.3  
 120.130298,30.211757,19.5  
 120.130243,30.211673,20.3  
 120.13012,30.211692,20.5  
 120.130095,30.21169,20.5  
 </coordinates>  
 </LineString>  
 </Placemark>  
 <Placemark>  
 <name></name>  
 <description><![CDATA[<img src="20170720091655\_30.21169-120.130095-20.5\_.jpg" width="250"/>2017-07-20 09:16:53]]></description>  
 <Point>  
 <coordinates>120.130095,30.21169,20.5</coordinates>  
 </Point>  
 <markerStyle>0</markerStyle>  
 </Placemark>

该文件中记录有文件名，开始与结束时间，线路长度，地标（placemark）名，描述（description），及点坐标（coordinates）。通常有用的信息为地标点坐标，对坐标的提取在下述的代码中采用了两种方式，一种是自定义函数；再者直接使用GeoPandas库实现。自定义函数可以根据提取数据的要求更为精准的提取，返回的数据格式也更自由。提取的数据保留了地标名与地标点坐标的对应关系。注意上述KML文件中形式为<name></name>的坐标位置，通常为在该位置拍摄有对应的照片。不同的App记录的GPS信息不同，需要根据具体情况调整代码以便提取正确的信息。

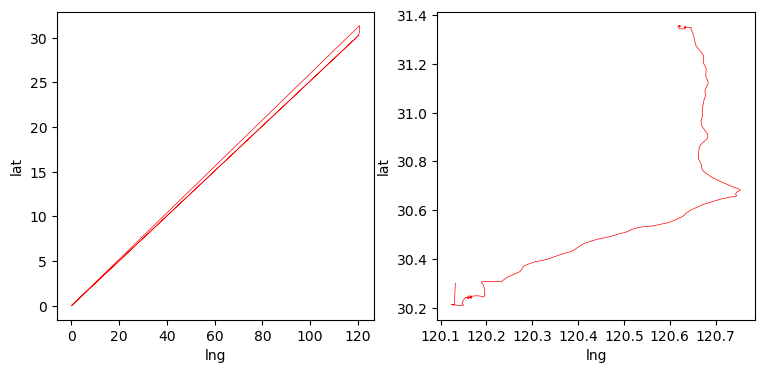
import util\_misc  
import re,os  
surveyPath\_kml\_fn=util\_misc.filePath\_extraction(r'./data/default\_20170720081441',["kml"]) # .kml和.jpg文件在同一文件夹下，读取.kml文件  
  
# A-自定义读取.kml文件坐标信息函数  
def kml\_coordiExtraction(kml\_pathDict):   
 '''  
 function - 提取.kml文件中的坐标信息  
   
 Params:  
 kml\_pathDict - .kml文件路径字典。文件夹名为键，值为包含该文件夹下所有文件名的列表。使用filePath\_extraction()函数提取。  
   
 Returns:  
 kml\_coordi\_dict - 返回坐标信息；dict  
 '''  
 kml\_CoordiInfo={}  
 '''正则表达式函数，将字符串转换为模式对象.号匹配除换行符之外的任何字符串，但只匹配一个字母，增加\*？字符代表匹配前面表达式的0个或多个副本，并匹配尽可能少的副本'''  
 pattern\_coodi=re.compile('<coordinates>(.\*?)</coordinates>')   
 pattern\_name=re.compile('<name>(.\*?)</name>')  
 count=0  
 kml\_coordi\_dict={}  
 for key in kml\_pathDict.keys():  
 temp\_dict={}  
 for val in kml\_pathDict[key]:  
 f=open(os.path.join(key,val),'r',encoding='UTF-8') # .kml文件中含有中文  
 content=f.read().replace('\n',' ') # 移除换行，从而可以根据模式对象提取标识符间的内容，同时忽略换行  
 name\_info=pattern\_name.findall(content)  
 coordi\_info=pattern\_coodi.findall(content)  
 coordi\_info\_processing=[coordi.strip(' ').split('\t\t') for coordi in coordi\_info]  
 print("名称数量：%d,坐标列表数量：%d"%(len(name\_info),len(coordi\_info\_processing))) # 名称中包含了文件名<name>default\_20170720081441</name>和文件夹名<name>线路标记点</name>。位于文头。  
 name\_info\_id=[name\_info[2:][n]+'\_ID\_'+str(n) for n in range(len(name\_info[2:]))] # 名称有重名，用ID标识  
 name\_coordi=dict(zip(name\_info\_id,coordi\_info\_processing))   
 for k in name\_coordi.keys():   
 temp=[]  
 for coordi in name\_coordi[k]:  
 coordi\_split=coordi.split(',')  
 # 提取的坐标值字符，可能不正确，不能转换为浮点数，因此通过异常处理  
 try:   
 one\_coordi=[float(i) for i in coordi\_split]   
 if len(one\_coordi)==3:# 可能提取的坐标值除了经纬度和高程，会出现多余或者少于3的情况，判断后将其忽略  
 temp.append(one\_coordi)  
 except ValueError:  
 count=+1  
 temp\_dict[k]=temp  
   
 kml\_coordi\_dict[os.path.join(key,val)]=temp\_dict  
 print("kml\_坐标字典键：",kml\_coordi\_dict.keys())   
 f.close()  
 return kml\_coordi\_dict  
  
kml\_coordi=kml\_coordiExtraction(surveyPath\_kml\_fn)  
  
# B-使用Geopandas库提取  
import geopandas as gpd  
import fiona   
from fiona.drvsupport import supported\_drivers  
supported\_drivers['KML'] = 'rw' # Enable fiona driver  
surveyPath\_kml\_fn='./data/default\_20170720081441/default\_20170720081441.kml'  
kml\_coordi\_=gpd.read\_file(surveyPath\_kml\_fn, driver='KML')  
util\_misc.print\_html(kml\_coordi\_)

名称数量：81,坐标列表数量：79  
kml\_坐标字典键： dict\_keys(['./data/default\_20170720081441\\default\_20170720081441.kml'])

|  | **Name** | **Description** | **geometry** |
| --- | --- | --- | --- |
| **0** | 线路起点 | 2017-07-20 08:14:41 | POINT Z (120.13201 30.30051 9.70000) |
| **1** | 线路追踪路径 | GPS工具箱导出数据 | LINESTRING Z (120.13019 30.21181 18.30000, 120.13030 30.21176 19.50000, 120.13024 30.21167 20.30000, 120.13012 30.21169 20.50000, 120.13009 30.21169 20.50000) |
| **2** |  | <img src="20170720091655\_30.21169-120.130095-20.5\_.jpg" width="250"/>2017-07-20 09:16:53 | POINT Z (120.13009 30.21169 20.50000) |
| **3** | 线路追踪路径 | GPS工具箱导出数据 | LINESTRING Z (120.13009 30.21169 20.50000, 120.12999 30.21168 19.60000, 120.12991 30.21165 20.30000, 120.12987 30.21166 20.20000, 120.12975 30.21164 20.60000, 120.12963 30.21157 20.20000, 120.12973 30.21163 20.50000) |
| **4** |  | <img src="20170720091848\_30.21163-120.129727-20.5\_.jpg" width="250"/>2017-07-20 09:18:47 | POINT Z (120.12973 30.21163 20.50000) |

提取的地标GPS坐标有可能存在异常值（离群值），这里剔除异常值的方法采用“异常值处理”部分所定义的is\_outlier(data,threshold=3.5)函数处理，可以选取经度、纬度或高程作为异常值的判断。打印的图表中第一个为原始值，存在异常值；第二个为异常值处理后的图表。

import numpy as np  
import matplotlib.pyplot as plt  
  
kml\_coordi\_lst=[]  
for key\_1 in kml\_coordi.keys():  
 for key\_2 in kml\_coordi[key\_1]:  
 for coordi in kml\_coordi[key\_1][key\_2]:  
 kml\_coordi\_lst.append(coordi)  
kml\_coordi\_array=np.array(kml\_coordi\_lst)  
  
x\_kml=kml\_coordi\_array[:,0]  
is\_outlier\_bool,\_=util\_misc.is\_outlier(x\_kml,threshold=3.5)  
kml\_coordi\_clean=kml\_coordi\_array[~is\_outlier\_bool]  
  
  
fig=plt.figure(figsize=(18/2,8/2))  
ax\_1=fig.add\_subplot(121) # 例如"111"为1x1的格网，第1个子图；"234"为2x3的格网，第4个子图。For example, "111" means "1x1 grid, first subplot" and "234" means "2x3 grid, 4th subplot"。  
ax\_1.plot(kml\_coordi\_array[:,0],kml\_coordi\_array[:,1],'r-',lw=0.5,markersize=5)  
  
ax\_2=fig.add\_subplot(122)  
ax\_2.plot(kml\_coordi\_clean[:,0],kml\_coordi\_clean[:,1],'r-',lw=0.5,markersize=5)  
  
ax\_1.set\_xlabel('lng')  
ax\_1.set\_ylabel('lat')  
ax\_2.set\_xlabel('lng')  
ax\_2.set\_ylabel('lat')  
plt.show()



使用Matplotlib库打印图表可以快速的查看数据信息，但是对于地理信息数据的表达通常不是很清楚。这里仍然使用Plotly库调用地图打印查看信息，色彩标识了高程数据的变化。

import pandas as pd  
import plotly.express as px  
kml\_coordi\_clean\_df=pd.DataFrame(data=kml\_coordi\_clean,columns=["lon","lat","elevation"])  
mapbox\_token='pk.eyJ1IjoicmljaGllYmFvIiwiYSI6ImNrYjB3N2NyMzBlMG8yc254dTRzNnMyeHMifQ.QT7MdjQKs9Y6OtaJaJAn0A'  
px.set\_mapbox\_access\_token(mapbox\_token)  
fig=px.scatter\_mapbox(kml\_coordi\_clean\_df,lat=kml\_coordi\_clean\_df.lat, lon=kml\_coordi\_clean\_df.lon,color="elevation",color\_continuous\_scale=px.colors.cyclical.IceFire, size\_max=15, zoom=10) # 亦可以选择列，通过size=""配置增加显示信息  
fig.show()



* 图像显示

为方便查看图像，或者排布图像用于研究文章的图表说明，定义图像排布显示的函数。图像处理过程，例如打开、调整图像大小均使用PIL图像处理库，在调整大小时要保持图像的R、G、B三个通道不变。

def imgs\_layoutShow(imgs\_root,imgsFn\_lst,columns,scale,figsize=(15,10)):  
 '''  
 function - 显示一个文件夹下所有图片，便于查看。  
   
 Params:  
 imgs\_root - 图像所在根目录；string  
 imgsFn\_lst - 图像名列表；list(string)  
 columns - 列数；int  
   
 Returns:  
 None  
 '''  
 import math,os  
 import matplotlib.pyplot as plt  
 from PIL import Image   
   
 rows=math.ceil(len(imgsFn\_lst)/columns)  
 fig,axes=plt.subplots(rows,columns,sharex=True,sharey=True,figsize=figsize) # 布局多个子图，每个子图显示一幅图像  
 ax=axes.flatten() # 降至1维，便于循环操作子图  
 for i in range(len(imgsFn\_lst)):  
 img\_path=os.path.join(imgs\_root,imgsFn\_lst[i]) # 获取图像的路径  
 img\_array=Image.open(img\_path) # 读取图像为数组，值为RGB格式0-255   
 img\_resize=img\_array.resize([int(scale \* s) for s in img\_array.size] ) # 传入图像的数组，调整图片大小  
 ax[i].imshow(img\_resize) # 显示图像  
 ax[i].set\_title(i+1)  
 fig.tight\_layout() # 自动调整子图参数，使之填充整个图像区域   
 fig.suptitle("images show",fontsize=14,fontweight='bold',y=1.02)  
 plt.show()  
  
imgs\_fn=util\_misc.filePath\_extraction('./data/default\_20170720081441',["jpg"])   
imgs\_root=list(imgs\_fn.keys())[0]  
imgsFn\_lst=imgs\_fn[imgs\_root]  
columns=6  
scale=0.2  
imgs\_layoutShow(imgs\_root,imgsFn\_lst,columns,scale)



上述定义的图像显示函数，存在空白的子图。及图像文件路径处理的参数包括了两个，一个是根目录imgs\_root；一个是文件名列表imgsFn\_lst。下述更新的函数imgs\_layoutShow\_FPList()则解决了上述两个问题。更新的代码块为:

invisible\_num=rows\*columns-len(imgs\_fp\_list)  
if invisible\_num>0:  
 for i in range(invisible\_num):  
 ax.flat[-(i+1)].set\_visible(False)

通过获取空的子图索引，配置不显示该子图达到目的。同时，输入的参数imgs\_fp\_lis为图片文件路径名，可以通过glob库提取。

def imgs\_layoutShow\_FPList(imgs\_fp\_list,columns,scale,figsize=(15,10)):  
 '''  
 function - 显示一个文件夹下所有图片，便于查看。  
  
 Params:  
 imgs\_fp\_list - 图像文件路径名列表；list(string)  
 columns - 显示列数；int  
 scale - 调整图像大小比例因子；float  
 figsize - 打印图表大小。The default is (15,10)；tuple(int)  
   
 Returns:  
 None  
 '''  
 import math,os  
 import matplotlib.pyplot as plt  
 from PIL import Image   
   
 rows=math.ceil(len(imgs\_fp\_list)/columns)  
 fig,axes=plt.subplots(rows,columns,figsize=figsize,) # 布局多个子图，每个子图显示一幅图像  
 ax=axes.flatten() # 降至1维，便于循环操作子图  
 for i in range(len(imgs\_fp\_list)):  
 img\_path=imgs\_fp\_list[i] # 获取图像的路径  
 img\_array=Image.open(img\_path) # 读取图像为数组，值为RGB格式0-255   
 img\_resize=img\_array.resize([int(scale \* s) for s in img\_array.size] ) # 传入图像的数组，调整图片大小  
 ax[i].imshow(img\_resize,) # 显示图像 aspect='auto'  
 ax[i].set\_title(i+1)  
 invisible\_num=rows\*columns-len(imgs\_fp\_list)  
 if invisible\_num>0:  
 for i in range(invisible\_num):  
 ax.flat[-(i+1)].set\_visible(False)  
 fig.tight\_layout() # 自动调整子图参数，使之填充整个图像区域   
 fig.suptitle("images show",fontsize=14,fontweight='bold',y=1.02)  
 plt.show()  
   
import glob  
imgs\_dougong\_fps=glob.glob('./data/default\_20170720081441/\*.jpg')  
columns=6;scale=0.2  
imgs\_layoutShow\_FPList(imgs\_dougong\_fps,columns,scale)



#### 2）Exif（Exchangeable image file format） 可交换图像格式

Exif是专门为数码相机相片设定的档案格式，可以记录照片的属性和拍摄信息。根据设置，用手机拍摄的照片通常包含Exif信息，其中也可能包括GPS位置信息。Exif包括哪些信息内容，可以通过from PIL.ExifTags import TAGS调入TAGS，打印查看，其中相对比较关键用于数据分析的一些信息包括拍摄的时间、图像大小、GPS位置数据和记录时间等。

def img\_exif\_info(img\_fp,printing=True):  
 '''  
 function - 提取数码照片的属性信息和拍摄数据，即可交换图像文件格式（Exchangeable image file format，Exif）  
   
 Params:  
 img\_fp - 一个图像的文件路径；string  
 printing - 是否打印。The default is True；bool  
   
 Returns:  
 exif\_ - 提取的照片信息结果；dict  
 '''  
 from PIL import Image  
 from PIL.ExifTags import TAGS  
 from datetime import datetime  
 import time   
   
 img=Image.open(img\_fp,)  
 try:  
 img\_exif=img.\_getexif()  
 exif\_={TAGS[k]: v for k, v in img\_exif.items() if k in TAGS}   
 # 由2017:07:20 09:16:58格式时间，转换为时间戳，用于比较时间先后。  
 time\_lst=[int(i) for i in re.split(' |:',exif\_['DateTimeOriginal'])] # DateTimeOriginal;'DateTime'  
 time\_tuple=datetime.timetuple(datetime(time\_lst[0], time\_lst[1], time\_lst[2], time\_lst[3], time\_lst[4], time\_lst[5],))  
 time\_stamp=time.mktime(time\_tuple)  
 exif\_["timestamp"]=time\_stamp  
   
 except ValueError:  
 print("exif not found!")  
 for tag\_id in img\_exif: # 取Exif信息 iterating over all EXIF data fields  
 tag=TAGS.get(tag\_id,tag\_id) # 获取标签名 get the tag name, instead of human unreadable tag id  
 data=img\_exif.get(tag\_id)  
 if isinstance(data,bytes): # 解码 decode bytes   
 try:  
 data=data.decode()  
 except ValueError:  
 data="tag:%s data not found."%tag  
 if printing:   
 print(f"{tag:30}:{data}")  
  
 # 将度转换为浮点数，Decimal Degrees = Degrees + minutes/60 + seconds/3600  
 if 'GPSInfo' in exif\_:   
 GPSInfo=exif\_['GPSInfo']  
 geo\_coodinate={  
 "GPSLatitude":float(GPSInfo[2][0]+GPSInfo[2][1]/60+GPSInfo[2][2]/3600),  
 "GPSLongitude":float(GPSInfo[4][0]+GPSInfo[4][1]/60+GPSInfo[4][2]/3600),  
 "GPSAltitude":GPSInfo[6],  
 "GPSTimeStamp\_str":"%d:%f:%f"%(GPSInfo[7][0],GPSInfo[7][1]/10,GPSInfo[7][2]/100),#字符形式  
 "GPSTimeStamp":float(GPSInfo[7][0]+GPSInfo[7][1]/10+GPSInfo[7][2]/100),#浮点形式  
 "GPSImgDirection":GPSInfo[17],  
 "GPSDestBearing":GPSInfo[24],  
 "GPSDateStamp":GPSInfo[29],  
 "GPSHPositioningError":GPSInfo[31],   
 }   
 if printing:   
 print("\_"\*50)  
 print(geo\_coodinate)  
 return exif\_,geo\_coodinate  
 else:  
 return exif\_  
  
import os  
img\_example\_1=os.path.join(imgs\_root,imgsFn\_lst[0])   
img\_exif\_1=img\_exif\_info(img\_example\_1)

544:0  
 545:0  
 546:0  
 547:0  
 548:1  
 549:  
ResolutionUnit :2  
ExifOffset :414  
ImageDescription :  
Make :HTC  
Model :HTC D830u  
Software :MediaTek Camera Application  
  
Orientation :1  
DateTime :2017:07:20 09:16:58  
YCbCrPositioning :2  
XResolution :72.0  
YResolution :72.0  
ExifVersion :0220  
ComponentsConfiguration :  
FlashPixVersion :0100  
DateTimeOriginal :2017:07:20 09:16:58  
DateTimeDigitized :2017:07:20 09:16:58  
ExposureBiasValue :0.0  
ColorSpace :1  
MeteringMode :2  
LightSource :255  
Flash :0  
FocalLength :3.79  
ExifImageWidth :4160  
ExifImageHeight :2368  
ExifInteroperabilityOffset :1478  
SceneCaptureType :0  
Contrast :0  
SubsecTime :42  
SubsecTimeOriginal :42  
SubsecTimeDigitized :42  
Saturation :0  
Sharpness :0  
ExposureTime :0.002756  
FNumber :2.0  
ExposureProgram :0  
ISOSpeedRatings :100  
ExposureMode :0  
WhiteBalance :0  
DigitalZoomRatio :1.0  
MakerNote :tag:MakerNote data not found.

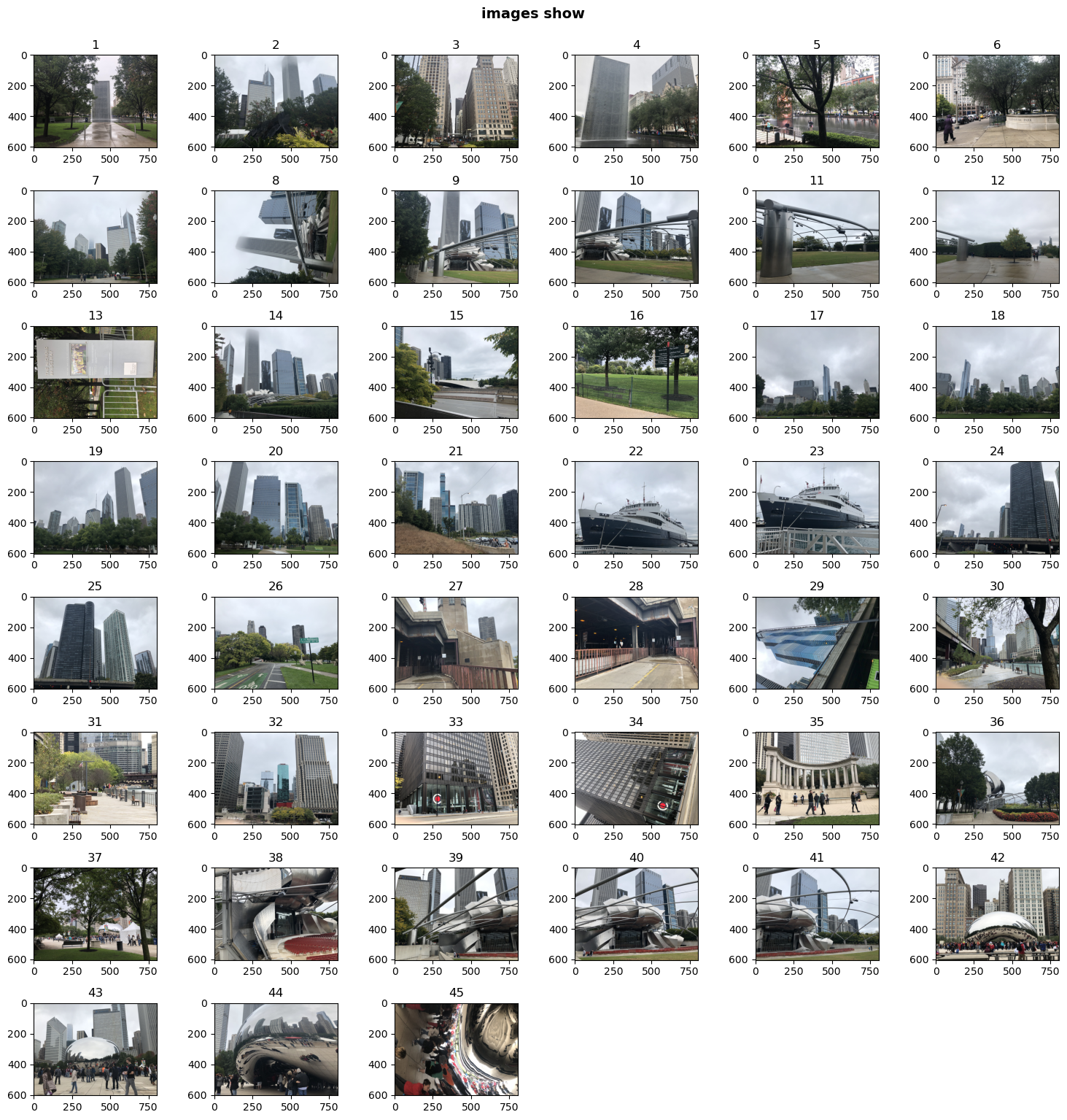
上述图片并未记录有GPS地理位置信息。下述调研的图像则可以查看到GPS信息。GPS信息的记录格式有可能不同，例如下述获取的“GPSInfo”中，2: (41.0, 52.0, 55.38)，但是也有可能为((19, 1), (31, 1), (5139, 100))格式。注意，自定义的Exif数据提取的函数，仅实现了第一种情况。

img\_ChicagoDowntown\_root=r'./data/imgs\_ChicagoDowntown'  
img\_example\_2=os.path.join(img\_ChicagoDowntown\_root,r'2019-10-11\_120110.jpg')  
img\_exif\_2,geo\_coodinate=img\_exif\_info(img\_example\_2,)

GPSInfo :{1: 'N', 2: (41.0, 52.0, 55.38), 3: 'W', 4: (87.0, 37.0, 26.43), 5: b'\x00', 6: 182.35323716873532, 7: (17.0, 1.0, 9.99), 12: 'K', 13: 0.0, 16: 'T', 17: 177.5288773523686, 23: 'T', 24: 177.5288773523686, 29: '2019:10:11', 31: 10.0}  
ResolutionUnit :2  
ExifOffset :218  
Make :Apple  
Model :iPhone X  
Software :https://heic.online  
Orientation :1  
DateTime :2019:10:11 12:01:11  
YCbCrPositioning :1  
XResolution :72.0  
YResolution :72.0  
ExifVersion :0221  
ComponentsConfiguration :  
ShutterSpeedValue :6.909027361693629  
DateTimeOriginal :2019:10:11 12:01:11  
DateTimeDigitized :2019:10:11 12:01:11  
ApertureValue :1.6959938128383605  
BrightnessValue :5.888149338229669  
ExposureBiasValue :0.0  
MeteringMode :5  
Flash :24  
FocalLength :4.0  
ColorSpace :65535  
ExifImageWidth :4032  
FocalLengthIn35mmFilm :28  
SceneCaptureType :0  
ExifImageHeight :3024  
SubsecTimeOriginal :201  
SubsecTimeDigitized :201  
SubjectLocation :(2015, 1511, 2217, 1330)  
SensingMethod :2  
ExposureTime :0.008333333333333333  
FNumber :1.8  
SceneType :  
ExposureProgram :2  
CustomRendered :2  
ISOSpeedRatings :25  
ExposureMode :0  
FlashPixVersion :0100  
WhiteBalance :0  
LensSpecification :(4.0, 6.0, 1.8, 2.4)  
LensMake :Apple  
LensModel :iPhone X back dual camera 4mm f/1.8  
MakerNote :tag:MakerNote data not found.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
{'GPSLatitude': 41.88205, 'GPSLongitude': 87.62400833333334, 'GPSAltitude': 182.35323716873532, 'GPSTimeStamp\_str': '17:0.100000:0.099900', 'GPSTimeStamp': 17.1999, 'GPSImgDirection': 177.5288773523686, 'GPSDestBearing': 177.5288773523686, 'GPSDateStamp': '2019:10:11', 'GPSHPositioningError': 10.0}

调用上述定义的imgs\_layoutShow\_FPList()函数，排布显示芝加哥市中心调研图像。

imgs\_ChicagoDowntown\_fps=glob.glob('./data/imgs\_ChicagoDowntown/\*.jpg')  
columns=6;scale=0.2  
imgs\_layoutShow\_FPList(imgs\_ChicagoDowntown\_fps,columns,scale,figsize=(15,15))



在Exif信息提取函数中，根据“DateTimeOriginal:2017:07:20 09:16:58”时间信息，计算时间戳，用于图像按拍摄时间排序，可以绘制图片拍摄的行走路径。此次绘制使用Plotly库提供的go方法实现。该方法在调用地图时，不需提供mapbox地图数据的访问许可（access token）。

import pandas as pd  
import plotly.graph\_objects as go  
  
imgs\_ChicagoDowntown\_coordi=[]  
for fn in imgs\_ChicagoDowntown\_fps:  
 img\_exif\_2,geo\_coodinate=img\_exif\_info(fn,printing=False)   
 imgs\_ChicagoDowntown\_coordi.append((geo\_coodinate['GPSLatitude'],-geo\_coodinate['GPSLongitude'],geo\_coodinate['GPSAltitude'],img\_exif\_2['timestamp'])) #注意经度负号  
  
pd.set\_option('display.float\_format', lambda x: '%.5f' % x)  
imgs\_ChicagoDowntown\_coordi\_df=pd.DataFrame(data=imgs\_ChicagoDowntown\_coordi,columns=["lat","lon","elevation",'timestamp']).sort\_values(by=['timestamp']) #按图片拍摄时间戳排序  
  
fig=go.Figure(go.Scattermapbox(mode = "markers+lines",lat=imgs\_ChicagoDowntown\_coordi\_df.lat, lon=imgs\_ChicagoDowntown\_coordi\_df.lon,marker = {'size': 10})) #亦可以选择列，通过size=""配置增加显示信息  
fig.update\_layout(  
 margin ={'l':0,'t':0,'b':0,'r':0},  
 mapbox = {  
 'style': "stamen-terrain",  
 'center': {'lon': -87.62401, 'lat': 41.88205},  
 'zoom': 14})  
  
fig.show()

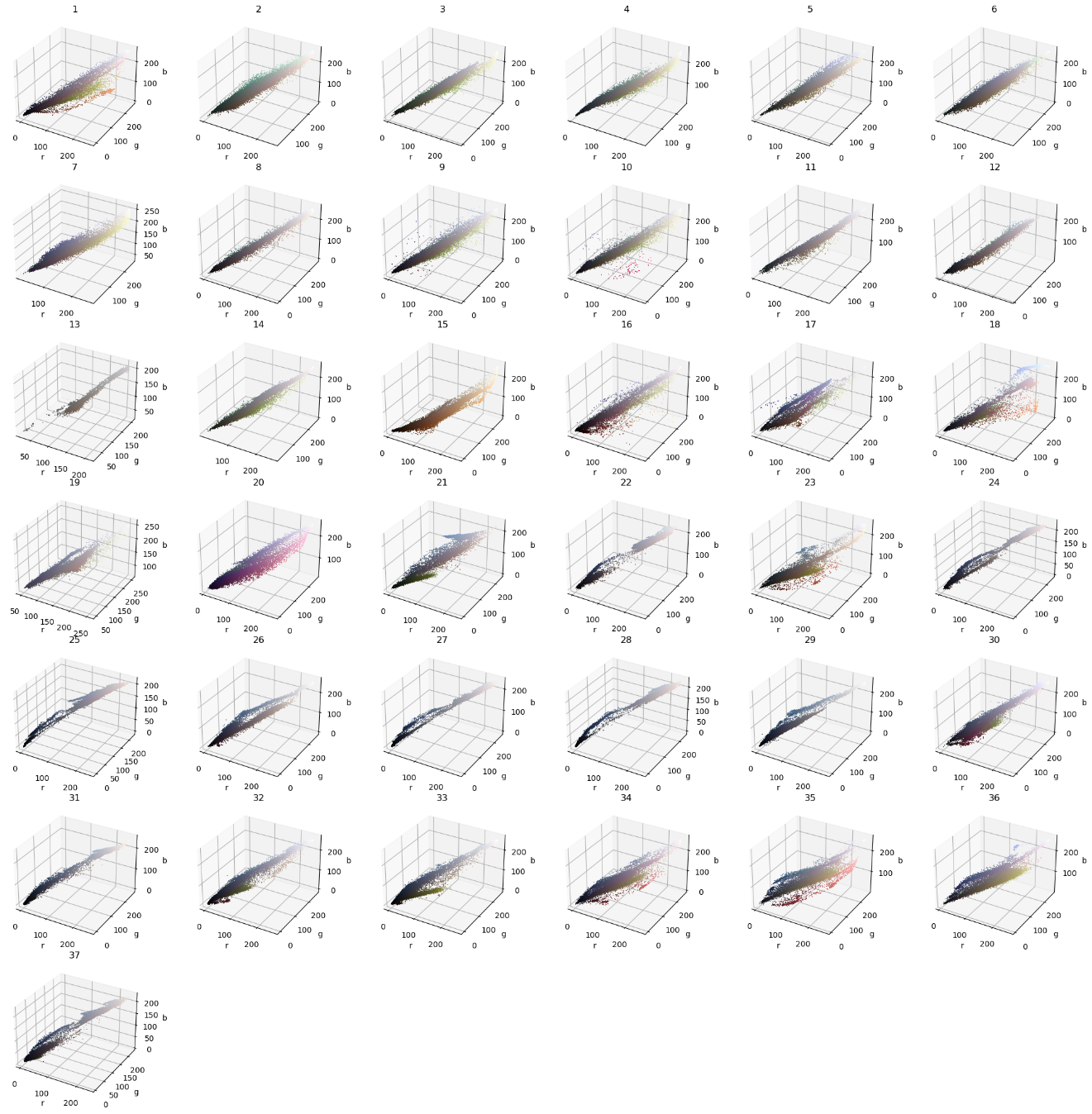


#### 3）RGB色彩的三维图示

包含色彩信息（RGB）的数据投射到三维空间中，可以通过判断区域色彩在三维空间域中的分布情况来把握Red、Green 和Blue 色彩分量的变化情况。为了增加计算速度，同样传入了scale参数，可以根据分析目的调整图像大小。使用PIL库的Image.open()方法读取图像时，返回的数组颜色值位于[0,255]区间，在用matplotlib打印图像时，通常颜色值位于[0,1]区间，因此需要转换，直接除以值255。

def imgs\_colorSpace(imgs\_root,imgsFn\_lst,columns,scale,figsize=(15,10)):  
 '''  
 function - 将像素RGB颜色投射到色彩空间中，直观感受图像颜色的分布。  
   
 Params:  
 imgs\_root - 图像所在根目录；string  
 imgsFn\_lst - 图像名列表；list(string)  
 columns - 列数；int  
   
 Returns:  
 None  
 '''  
 import math,os  
 import matplotlib.pyplot as plt  
 from PIL import Image  
 import numpy as np  
 from tqdm import tqdm   
   
 rows=math.ceil(len(imgsFn\_lst)/columns)  
 fig=plt.figure()   
 for i in tqdm(range(len(imgsFn\_lst))):  
 ax=fig.add\_subplot(rows,columns,i+1, projection='3d') # 不断增加子图，并设置投影为3d模式，可以显示三维坐标空间   
 img=os.path.join(imgs\_root,imgsFn\_lst[i])  
 img\_path=os.path.join(imgs\_root,imgsFn\_lst[i]) # 获取图像的路径  
 img\_array=Image.open(img\_path) # 读取图像为数组，值为RGB格式0-255   
 img\_resize=img\_array.resize([int(scale \* s) for s in img\_array.size] ) # 传入图像的数组，调整图片大小  
 img\_array=np.asarray(img\_resize)  
 ax.scatter(img\_array[:,:,0],img\_array[:,:,1],img\_array[:,:,2], c=(img\_array/255).reshape(-1,3), marker='+',s=0.1) # 用RGB的三个分量值作为颜色的空间坐标，并显示其颜色。设置颜色时，需要将0-255缩放至0-1区间  
 ax.set\_xlabel('r',labelpad=1)  
 ax.set\_ylabel('g')  
 ax.set\_zlabel('b',labelpad=2)  
 ax.set\_title(i+1)  
 fig.set\_figheight(figsize[0])  
 fig.set\_figwidth(figsize[1])  
 print("Ready to show...")  
 fig.tight\_layout()  
 plt.show()   
   
imgs\_fn=util\_misc.filePath\_extraction(r'./data/default\_20170720081441',["jpg"])   
imgs\_root=list(imgs\_fn.keys())[0]  
imgsFn\_lst=imgs\_fn[imgs\_root]  
columns=6  
scale=0.05   
imgs\_colorSpace(imgs\_root,imgsFn\_lst,columns,scale,figsize=(20,20))

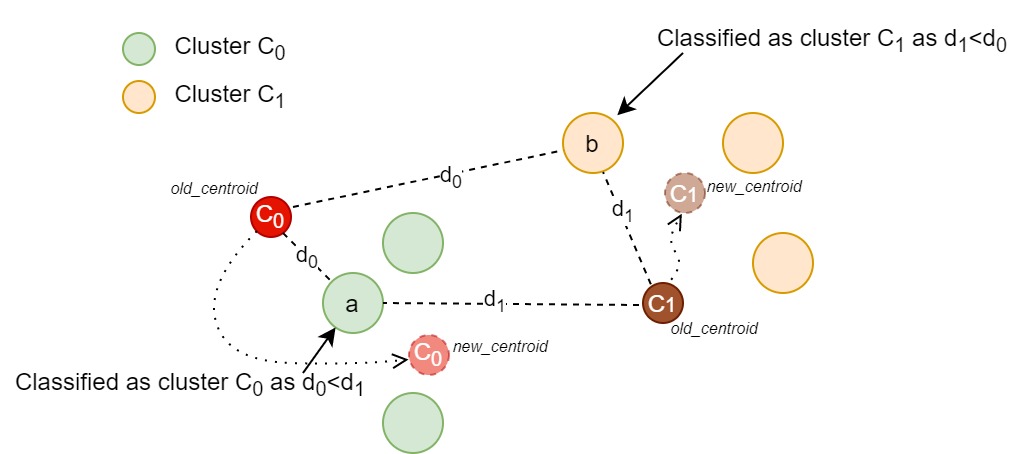
100%|██████████| 37/37 [00:13<00:00, 2.67it/s]  
  
  
Ready to show...



### 2.5.1.2 聚类（Clustering）[1][2]

监督学习（Supervised learning），是机器学习的一种方法，可以学习训练数据集建立学习模型用于预测新的实例。回归属于监督学习，所用到的数据集被标识分类，通常包括特征值（自变量）和目标值（标签，因变量）。非监督学习(Unsupervised learning)，则没有给定事先标记的数据集，自动对数据集进行分类或分群。聚类则属于非监督学习。

聚类是把相似的对象通过静态分类的方法分成不同的组别或者更多的子集（subset），这样让在同一个子集中的成员对象都有相似的一些属性。聚类涉及的算法很多，K-Means是常见算法的一种，通过尝试将样本分离到个方差相等的组中对数据进行聚类，最小化指标（criterion），例如聚类内平方和指标（within-cluster sum-of-squares criterion），其公式可表达为：。*Python Machine Learning* [1]对K-Means算法解释的非常清晰，引用其中的案例加以说明。



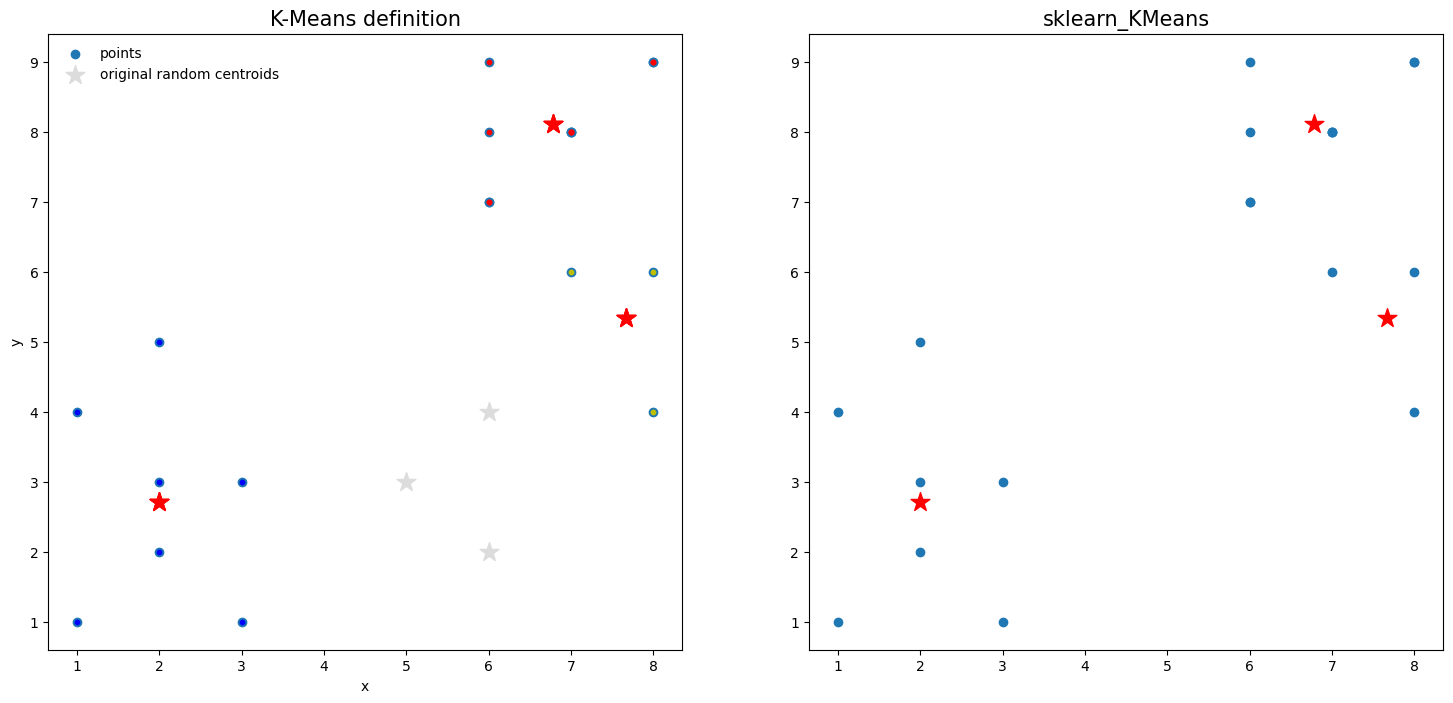
图中假设了两个簇C0C1k=2aC\_{0}$ 距离$d\_{0} C\_{1}ad0所代表的簇。将所有的点根据距离远近归属于不同的质心所代表的类之后，由所归属簇中点的均值作为新的质心，图中用new\_centroid假设标识。分别计算旧质心和新质心的距离，如果所有簇质心的新旧距离值为0，则意味质心没有发生变化，即完成聚类；否则，用新的质心重复上一轮的计算，直至质心新旧距离值为0为止。只要有足够的时间，K-Means总是收敛的，但它可能是一个局部最小值，这高度依赖于质心的初始化。因此，计算通常要进行多次，并对质心进行不同的初始化。

下述代码根据上述计算原理自定义聚类函数，同时使用sklearn库的KMeans方法直接实现，比较二者之间的差异。自定义的K-Means所计算的结果有多种可能，通过多次运行会获得与sklearn库一致的结果。sklearn库的KMeans方法通过init='k-means++'使得质心彼此之间保持距离，解决了局部最小值的问题，证明比随机初始化得到更好的结果。

不配置忽略warning警告信息，将提示：KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP\_NUM\_THREADS=1.

import numpy as np  
import warnings  
warnings.filterwarnings('ignore') #  
  
class K\_Means:  
 '''  
 class - 定义K-Means算法  
   
 Params:  
 X - 待分簇的数据（数组）  
 k - 分簇数量  
 figsize - Matplotlib图表大小  
 '''   
 def \_\_init\_\_(self,X,k,figsize):   
 self.X=X  
 self.k=k  
 self.figsize=figsize   
  
 def euclidean\_distance(self,a,b,ax=1):   
 '''  
 function - 计算两点距离。To calculate the distance between two points  
   
 Params:  
 a - 2维度数组，例如[[3,4]  
 [5,6]  
 [1,4]]  
 b - 2维度数组  
 ax - 计算轴   
 '''  
 import numpy as np  
 return np.linalg.norm(a-b, axis=ax)  
   
 def update(self,ax):   
 '''  
 function - K-Means算法  
 '''  
 from copy import deepcopy  
 import numpy as np   
   
 # 生产随机质心 generate k random points (centroids)  
 Cx=np.random.randint(np.min(X[:,0]), np.max(X[:,0]), size=self.k)  
 Cy=np.random.randint(np.min(X[:,1]), np.max(X[:,1]), size=self.k)  
 ax.scatter(Cx, Cy,label="original random centroids",marker='\*',c='gainsboro',s=200)   
  
 C=np.array(list(zip(Cx, Cy)), dtype=np.float64) # 质心数组 -represent the k centroids as a matrix  
 C\_prev=np.zeros(C.shape) # 建立同质心数组形状，值为0的数组-create a matrix of 0 with same dimension as C (centroids)  
 clusters=np.zeros(len(X))# 存储每个点所属子群-to store the cluster each point belongs to   
 distance\_differences=self.euclidean\_distance(C, C\_prev)# 计算质心与C\_prev之间的距离-measure the distance between the centroids and C\_prev   
   
 # 循环计算，缩小前一步和后一步质心距离的差异 -loop as long as there is still a difference in distance between the previous and current centroids  
 count=0  
 while distance\_differences.any() != 0:  
 print("epoch:%d"%count)  
 # 将每个值分配到最近的簇-assign each value to its closest cluster  
 for i in range(len(self.X)):  
 distances=self.euclidean\_distance(self.X[i], C)  
 cluster=np.argmin(distances) # 延着一个轴，返回最小值索引-returns the indices of the minimum values along an axis  
 clusters[i]=cluster  
   
 C\_prev=deepcopy(C) # 存储前一质心-store the prev centroids  
  
 # 通过取均值寻找新的质心-find the new centroids by taking the average value  
 for i in range(k):  
 points=[X[j] for j in range(len(X)) if clusters[j]==i] #取簇i中的所有点-take all the points in cluster i  
 if len(points)!=0:  
 C[i]=np.mean(points,axis=0)  
  
 distance\_differences=self.euclidean\_distance(C, C\_prev) # 计算前一与后一质心的距离-find the distances between the old centroids and the new centroids  
 print("distance\_differences:",distance\_differences)  
 count+=1  
  
 # 打印散点图-plot the scatter plot  
 colors=['b','r','y','g','c','m']  
 for i in range(k):  
 points=np.array([X[j] for j in range(len(X)) if clusters[j] == i])  
 if len(points) > 0:  
 ax.scatter(points[:, 0], points[:, 1], s=10, c=colors[i])  
 else:  
 print("Plesae regenerate your centroids again.")# 这意味着其中一个簇没有点 this means that one of the clusters has no points  
 #ax.scatter(points[:, 0], points[:, 1], s=10, c=colors[i])  
 ax.scatter(C[:, 0], C[:, 1], marker='\*', s=200, c='red')   
   
 def sklearn\_KMeans(self,ax):   
 '''  
 function - 使用Sklearn库的KMeans算法聚类  
 '''  
 from sklearn.cluster import KMeans  
   
 kmeans=KMeans(n\_clusters=self.k)  
 kmeans=kmeans.fit(self.X)  
 labels=kmeans.predict(self.X)  
 centroids = kmeans.cluster\_centers\_  
   
 c = ['b','r','y','g','c','m']  
 colors = [c[i] for i in labels]   
 ax.scatter(centroids[:, 0], centroids[:, 1], marker='\*', s=200, c='red')  
   
 print("预测(7,5)的簇为：%d"%kmeans.predict([[7,5]])[0])  
   
 def execution(self):  
 '''  
 function - 执行  
 '''  
 %matplotlib inline  
 import matplotlib.pyplot as plt  
   
 fig, axs=plt.subplots(1,2,figsize=self.figsize)   
 axs[0].scatter(self.X[:,0], self.X[:,1],label="points")  
 axs[0].set\_title(r'K-Means definition', fontsize=15)  
 self.update(axs[0])  
   
 axs[1].scatter(self.X[:,0], self.X[:,1],label="points")  
 axs[1].set\_title(r'sklearn\_KMeans', fontsize=15)  
 self.sklearn\_KMeans(axs[1])  
   
 axs[0].set\_xlabel('x')  
 axs[0].set\_ylabel('y')  
 axs[0].legend(loc='upper left', frameon=False)  
 plt.show()  
   
kmeans\_dataset=[(1,1),(2,2),(2,3),(1,4),(3,3),(6,7),(7,8),(6,8),(7,6),(6,9),(2,5),(7,8),(8,9),(6,7),(7,8),(3,1),(8,4),(8,6),(8,9)]   
X=np.array(kmeans\_dataset)   
k=3 # 配置分组的数量（亦随机生成中心的数量）  
figsize=(18,8)  
K=K\_Means(X,k,figsize)  
K.execution()

epoch:0  
distance\_differences: [3.01357473 3.56000156 0. ]  
epoch:1  
distance\_differences: [0. 0.32363651 2.82842712]  
epoch:2  
distance\_differences: [0. 0.20429277 1. ]  
epoch:3  
distance\_differences: [0. 0.21227748 0.47140452]  
epoch:4  
distance\_differences: [0. 0. 0.]  
预测(7,5)的簇为：2

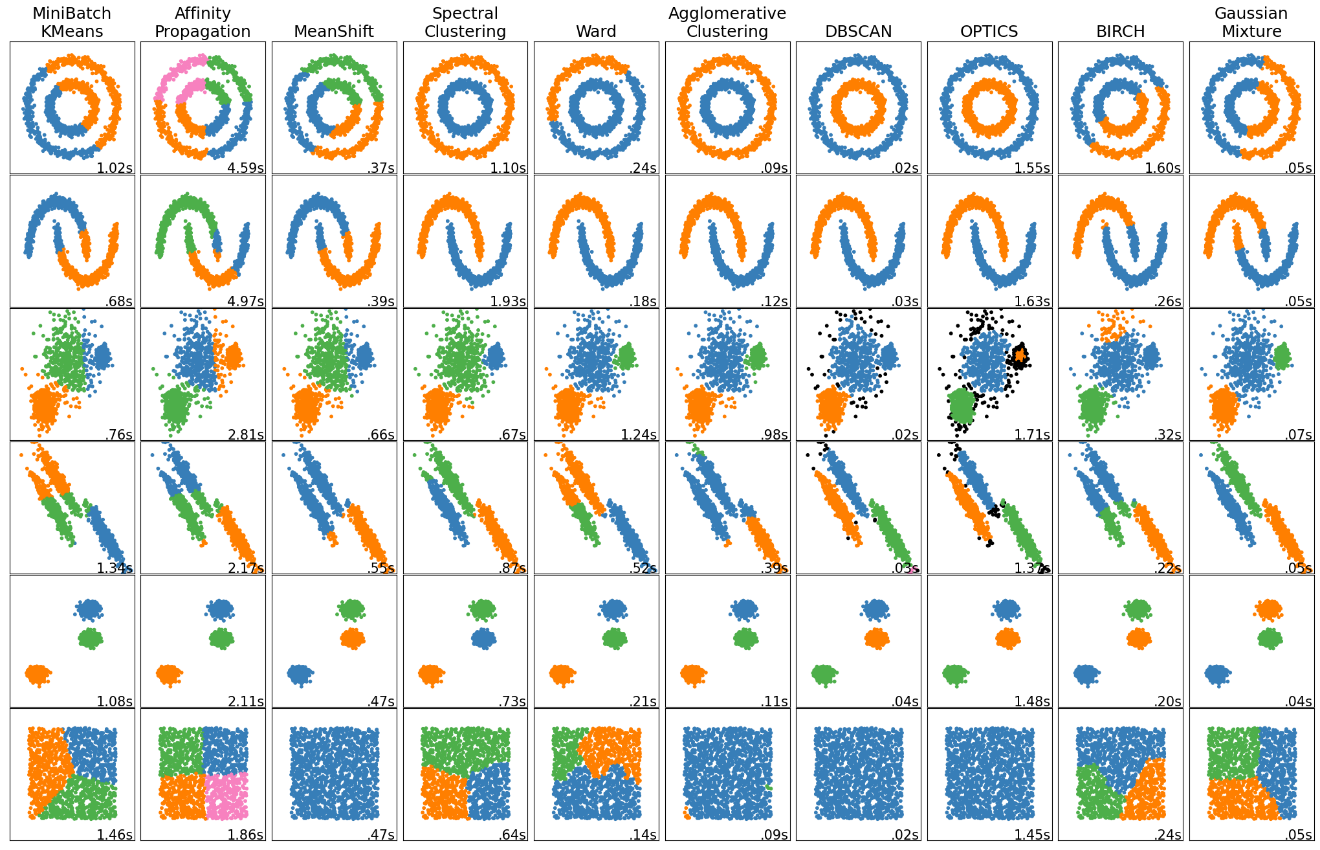


* 聚类算法比较

[sklearn官网聚类部分](https://scikit-learn.org/stable/modules/clustering.html#clustering)①提供了一组代码，比较多个不同聚类算法，其归结如下：

| 方法名称 Method name | 参数 Parameters | 扩展性 Scalability | 用例 Usecase | 几何 Geometry (metric used) |
| --- | --- | --- | --- | --- |
| K-Means | number of clusters | Very large n\_samples, medium n\_clusters with MiniBatch code | General-purpose, even cluster size, flat geometry, not too many clusters | Distances between points |
| Affinity propagation | damping, sample preference | Not scalable with n\_samples | Many clusters, uneven cluster size, non-flat geometry | Graph distance (e.g. nearest-neighbor graph) |
| Mean-shift | bandwidth | Not scalable with n\_samples | Many clusters, uneven cluster size, non-flat geometry | Distances between points |
| Spectral clustering | number of clusters | Medium n\_samples, small n\_clusters | Few clusters, even cluster size, non-flat geometry | Graph distance (e.g. nearest-neighbor graph) |
| Ward hierarchical clustering | number of clusters or distance threshold | Large n\_samples and n\_clusters | Many clusters, possibly connectivity constraints | Distances between points |
| Agglomerative clustering | number of clusters or distance threshold, linkage type, distance | Large n\_samples and n\_clusters | Many clusters, possibly connectivity constraints, non Euclidean distances | Any pairwise distance |
| DBSCAN | neighborhood size | Very large n\_samples, medium n\_clusters | Non-flat geometry, uneven cluster sizes | Distances between nearest points |
| OPTICS | minimum cluster membership | Very large n\_samples, large n\_clusters | Non-flat geometry, uneven cluster sizes, variable cluster density | Distances between points |
| Gaussian mixtures | many | Not scalable | Flat geometry, good for density estimation | Mahalanobis distances to centers |
| Birch | branching factor, threshold, optional global clusterer. | Large n\_clusters and n\_samples | Large dataset, outlier removal, data reduction. | Euclidean distance between points |

官网代码计算结果如下：



### 2.5.1.3 聚类图像主题色

城市色彩，也称“城市环境色彩”，泛指城市各个构成要素公共空间部分所呈现出的色彩面貌总和。城市色彩包含大量复杂多变的元素，因此必须科学地调查与分析，才能实现有效引导和规划发展。在城市色彩相关的研究上，主要有：分析城市色彩特征，调查与定量分析，更新与保护机制研究；景观环境色彩构成，及利用 MATLAB 计算插值与应用回归算法实现城市色彩主色调意向图的自动填充，得到城市色彩主色调理想色彩地图的研究。部分传统的研究由于受到数据分析技术的限制，对于批量的城市影像提取主题色时偏重手工提取，不仅增加时间成本，也影响分析的精度，同时在数据分析方法和数据信息表达上亦受到限制。因此有必要借助机器学习中的聚类等方法，自主聚类主题色，通过Python的matplotlib标准库实现数据可视化增强表达。

首先在Python程序语言中批量读取图像，因为所拍摄的图像分辨率较高，而色彩分析不需要这样的高精度，因此通过压缩图像降低图像大小来节约分析的时间。然后设置色彩主题色聚类的数量为7，获取每幅图像的7个主题色。采用KMeans聚类算法分类色彩提取主题色，提取所有图像的主题色之后汇总于一个数组中。在数据增强可视化方面设计了散点形式打印主题色，直观反映城市色彩印象。通过城市主题色的提取、色彩印象感官的呈现来研究城市色彩，可以针对不同的城市空间、不同的调研时间，分析色彩的变化。

聚类部分，参考了上述sklearn提供的”Comparing different clustering algorithms on toy datasets”代码，从下面代码中也可以观察到代码迁移的痕迹。

def img\_rescale(img\_path,scale):  
 '''  
 function - 读取与压缩图像，返回2维度数组  
   
 Params:  
 img\_path - 待处理图像路径；lstring  
 scale - 图像缩放比例因子；float  
   
 Returns:  
 img\_3d - 返回三维图像数组；ndarray   
 img\_2d - 返回二维图像数组；ndarray  
 '''  
 from PIL import Image  
 import numpy as np   
   
 img=Image.open(img\_path) # 读取图像为数组，值为RGB格式0-255   
 img\_resize=img.resize([int(scale \* s) for s in img.size] ) # 传入图像的数组，调整图片大小  
 img\_3d=np.array(img\_resize)  
 h, w, d=img\_3d.shape  
 img\_2d=np.reshape(img\_3d, (h\*w, d)) # 调整数组形状为2维  
  
 return img\_3d,img\_2d  
  
def img\_theme\_color(imgs\_root,imgsFn\_lst,columns,scale,):   
 '''  
 function - 聚类的方法提取图像主题色，并打印图像、聚类预测类的二维显示和主题色带  
   
 Params:  
 imgs\_root - 图像所在根目录；string  
 imgsFn\_lst - 图像名列表；list(string)  
 columns - 列数；int   
   
 Returns:  
 themes - 图像主题色；array  
 pred - 预测的类标；array  
 '''  
 import os,time,warnings  
 import numpy as np  
 import matplotlib.pyplot as plt  
 from tqdm import tqdm  
 from sklearn.preprocessing import StandardScaler  
 from sklearn import cluster, datasets, mixture  
 from itertools import cycle, islice   
   
 # 设置聚类参数，本实验中仅使用了KMeans算法，其它算法可以自行尝试  
 kmeans\_paras={'quantile': .3,  
 'eps': .3,  
 'damping': .9,  
 'preference': -200,  
 'n\_neighbors': 10,  
 'n\_clusters': 7}   
   
 imgsPath\_lst=[os.path.join(imgs\_root,p) for p in imgsFn\_lst]  
 imgs\_rescale=[(img\_rescale(img,scale)) for img in imgsPath\_lst]   
 datasets=[((i[1],None),{}) for i in imgs\_rescale] # 基于img\_2d的图像数据，用于聚类计算  
 img\_lst=[i[0] for i in imgs\_rescale] # 基于img\_3d的图像数据，用于图像显示  
   
 themes=np.zeros((kmeans\_paras['n\_clusters'], 3)) # 建立0占位的数组，用于后面主题数据的追加。'n\_clusters'为提取主题色的聚类数量，此处为7，轴2为3，是色彩的RGB数值  
 (img\_3d,img\_2d)=imgs\_rescale[0] # 可以1次性提取元组索引值相同的值，img就是img\_3d，而pix是img\_2d  
 img2d\_V,img2d\_H=img\_2d.shape # 获取img\_2d数据的形状，用于pred预测初始数组的建立  
 pred=np.zeros((img2d\_V)) # 建立0占位的pred预测数组，用于后面预测结果数据的追加，即图像中每一个像素点属于设置的7个聚类中的哪一组，预测给定类标  
   
 plt.figure(figsize=(6\*3+3, len(imgsPath\_lst)\*2)) # 图表大小的设置，根据图像的数量来设置高度，宽度为3组9个子图，每组包括图像、预测值散点图和主题色  
 plt.subplots\_adjust(left=.02, right=.98, bottom=.001, top=.96, wspace=.3,hspace=.3) # 调整图，避免横纵向坐标重叠   
 subplot\_num=1 # 子图的计数   
   
 for i\_dataset, (dataset, algo\_params) in tqdm(enumerate(datasets)): # 循环pixData数据，即待预测的每个图像数据。enumerate()函数将可迭代对象组成一个索引序列，可以同时获取索引和值，其中i\_dataset为索引，从整数0开始  
 X, y=dataset # 用于机器学习的数据一般包括特征值和类标，此次实验为无监督分类的聚类实验，没有类标，并将其在前文中设置为None对象  
 Xstd=StandardScaler().fit\_transform(X) # 标准化数据仅用于二维图表的散点，可视化预测值，而不用于聚类，聚类数据保持色彩的0-255值范围  
 # 此次实验使用KMeans算法，参数为'n\_clusters'一项。不同算法计算效率不同，例如MiniBatchKMeans和KMeans算法计算较快  
 km=cluster.KMeans(n\_clusters=kmeans\_paras['n\_clusters'])  
 clustering\_algorithms=(('KMeans',km),)  
 for name, algorithm in clustering\_algorithms:   
 t0=time.time()   
 # 警告错误，使用warning库  
 with warnings.catch\_warnings():  
 warnings.filterwarnings(  
 "ignore",  
 message="the number of connected components of the " +"connectivity matrix is [0-9]{1,2}" +" > 1. Completing it to avoid stopping the tree early.",  
 category=UserWarning)  
 warnings.filterwarnings(  
 "ignore",  
 message="Graph is not fully connected, spectral embedding" +" may not work as expected.",  
 category=UserWarning)  
 algorithm.fit(X) # 通过fit函数执行聚类算法   
   
 quantize=np.array(algorithm.cluster\_centers\_, dtype=np.uint8) # 返回聚类的中心，为主题色  
 themes=np.vstack((themes,quantize)) # 将计算获取的每一图像主题色追加到themes数组中  
 t1=time.time() # 计算聚类算法所需时间  
 '''获取预测值/分类类标'''   
 if hasattr(algorithm, 'labels\_'):  
 y\_pred=algorithm.labels\_.astype(int)  
 else:  
 y\_pred=algorithm.predict(X)   
 pred=np.hstack((pred,y\_pred)) # 将计算获取的每一图像聚类预测结果追加到pred数组中  
 fig\_width=(len(clustering\_algorithms)+2)\*3 # 水平向子图数  
 plt.subplot(len(datasets), fig\_width,subplot\_num)  
 plt.imshow(img\_lst[i\_dataset]) # 图像显示子图  
   
 plt.subplot(len(datasets),fig\_width, subplot\_num+1)  
 if i\_dataset == 0:  
 plt.title(name, size=18)  
 colors = np.array(list(islice(cycle(['#377eb8', '#ff7f00', '#4daf4a','#f781bf', '#a65628', '#984ea3','#999999', '#e41a1c', '#dede00']),int(max(y\_pred) + 1)))) # 设置预测类标分类颜色  
 plt.scatter(Xstd[:, 0], Xstd[:, 1], s=10, color=colors[y\_pred]) # 预测类标子图  
 plt.xlim(-2.5, 2.5)  
 plt.ylim(-2.5, 2.5)  
 plt.xticks(())  
 plt.yticks(())  
 plt.text(.99, .01, ('%.2fs' % (t1 - t0)).lstrip('0'),transform=plt.gca().transAxes, size=15,horizontalalignment='right') # 子图中显示聚类计算时间长度，   
 # 图像主题色子图参数配置  
 plt.subplot(len(datasets), fig\_width,subplot\_num+2)  
 t=1  
 pale=np.zeros(img\_lst[i\_dataset].shape, dtype=np.uint8)  
 h, w,\_=pale.shape  
 ph=h/len(quantize)  
 for y in range(h):  
 pale[y,::] = np.array(quantize[int(y/ph)], dtype=np.uint8)  
 plt.imshow(pale)   
 t+=1   
 subplot\_num+=3   
 plt.show()   
 return themes,pred  
  
import util\_misc  
imgs\_fn=util\_misc.filePath\_extraction(r'./data/default\_20170720081441',["jpg"])   
imgs\_root=list(imgs\_fn.keys())[0]  
imgsFn\_lst=imgs\_fn[imgs\_root]  
columns=6  
scale=0.2  
themes,pred=img\_theme\_color(imgs\_root,imgsFn\_lst,columns,scale,)

37it [06:46, 10.99s/it]

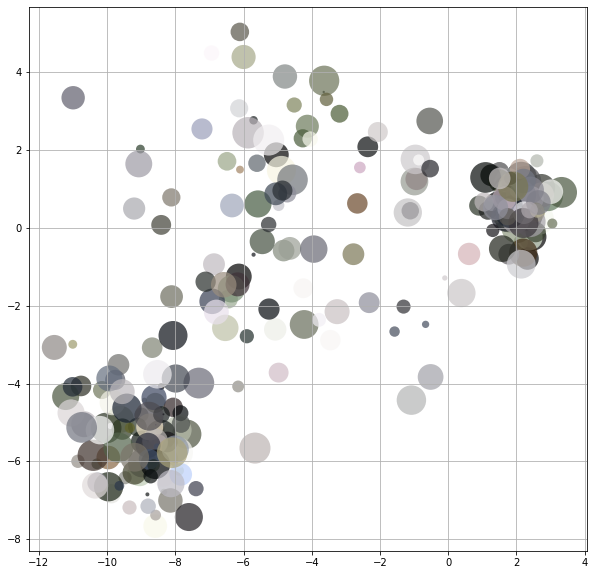


聚类所有图像获取主题色后，可以将计算的结果包括主题色，及预测的类标保存在硬盘中，避免重复计算。

def save\_as\_json(array,save\_root,fn):   
 '''  
 function - 保存文件,将文件存储为json数据格式  
   
 Params:  
 array - 待保存的数组；array  
 save\_root - 文件保存的根目录 ；string  
 fn - 保存的文件名；string  
   
 Returns:  
 None  
 '''  
 import time,os,json  
   
 json\_file=open(os.path.join(save\_root,r'%s\_'%fn+str(time.time()))+'.json','w')  
 json.dump(array.tolist(),json\_file) # 将numpy数组转换为列表后存储为json数据格式  
 json\_file.close()  
   
save\_root='./data\_processed'   
save\_as\_json(themes,save\_root,'themes')   
save\_as\_json(pred,save\_root,'themes\_pred')

以随机散点图的形式显示色彩。

def themeColor\_impression(theme\_color):  
 '''  
 function - 显示所有图像主题色，获取总体印象  
   
 Params:  
 theme\_color - 主题色数组；array  
   
 Returns:  
 None  
 '''  
 from sklearn import datasets  
 from numpy.random import rand  
 import matplotlib.pyplot as plt   
   
 n\_samples=theme\_color.shape[0]  
 random\_state=170 #可 为默认，不设置该参数，获得随机图形  
 # 利用scikit的datasets数据集构建有差异变化的斑点  
 varied=datasets.make\_blobs(n\_samples=n\_samples,cluster\_std=[1.0, 2.5, 0.5],random\_state=random\_state)  
 (x,y)=varied   
 fig, ax=plt.subplots(figsize=(10,10))  
 scale=1000.0\*rand(n\_samples) # 设置斑点随机大小  
 ax.scatter(x[...,0], x[...,1], c=theme\_color/255,s=scale,alpha=0.7, edgecolors='none') # 将主题色赋予斑点  
 ax.grid(True)   
 plt.show()   
  
import json  
import numpy as np  
with open("./data\_processed/themes\_1641623409.4860697.json") as f:  
 themes=json.load(f)  
themeColor\_impression(np.array(themes))



注释（Notes）：

① sklearn官网聚类部分，聚类算法比较（<https://scikit-learn.org/stable/modules/clustering.html#clustering>）。

参考文献（References）:

[1] Wei-Meng Lee.Python Machine Learning[M].US:Wiley.April, 2019.

[2] Giuseppe Bonaccorso.Mastering Machine Learning Algorithms: Expert techniques for implementing popular machine learning algorithms, fine-tuning your models, and understanding how they work[M].Birmingham:Packt Publishing.January, 2020.