## 2.1.1 POI数据与描述性统计和正态分布

### 2.1.1.1 单个分类POI数据检索与地理空间点地图

在城市空间分析中，涉及到两个方面的数据，一类是物质空间相关数据，进一步可以分为二维平面数据（例如遥感影像，城市地图等）和三维空间数据（例如雷达数据，含高空扫描和地面扫描，及一般意义上各类格式的城市三维模型等）；另一类则是物质空间承载的各类属性数据，包括反映二维平面数据的属性（与二维地理位置结合，可以划分到二维平面数据当中），例如用地类型、自然资源分布、地址名称等；和反映人类、动物及无形物质各类活动性质的数据，例如人们的活动轨迹（出租车和共享单车车行轨迹、夜间灯光、手机基站定位用户数量等）、动物迁徙路径和各类小气候测量指标变化等。

[百度地图开放平台](http://lbsyun.baidu.com/index.php?title=%E9%A6%96%E9%A1%B5)①提供地图相关的功能与服务，其Web服务API为开发者提供http/https接口。开发者可以通过http/https形式发起检索请求，获取返回JSON（JavaScript Object Notation）或XML（Extensible Markup Language ）格式的检索数据。其中[兴趣点（POI，points of interest）](http://lbsyun.baidu.com/index.php?title=lbscloud/poitags)②数据，目前包括21个大类，和无数小类，是反映物质空间承载的人类活动属性的数据（业态分布），如下：

| 一级行业分类 | 二级行业分类 |
| --- | --- |
| 美食 | 中餐厅、外国餐厅、小吃快餐店、蛋糕甜品店、咖啡厅、茶座、酒吧、其他 |
| 酒店 | 星级酒店、快捷酒店、公寓式酒店、民宿、其他 |
| 购物 | 购物中心、百货商场、超市、便利店、家居建材、家电数码、商铺、市场、其他 |
| 生活服务 | 通讯营业厅、邮局、物流公司、售票处、洗衣店、图文快印店、照相馆、房产中介机构、公用事业、维修点、家政服务、殡葬服务、彩票销售点、宠物服务、报刊亭、公共厕所、步骑行专用道驿站、其他 |
| 丽人 | 美容、美发、美甲、美体、其他 |
| 旅游景点 | 公园、动物园、植物园、游乐园、博物馆、水族馆、海滨浴场、文物古迹、教堂、风景区、景点、寺庙、其他 |
| 休闲娱乐 | 度假村、农家院、电影院、ktv、剧院、歌舞厅、网吧、游戏场所、洗浴按摩、休闲广场、其他 |
| 运动健身 | 体育场馆、极限运动场所、健身中心、其他 |
| 教育培训 | 高等院校、中学、小学、幼儿园、成人教育、亲子教育、特殊教育学校、留学中介机构、科研机构、培训机构、图书馆、科技馆、 其他 |
| 文化传媒 | 新闻出版、广播电视、艺术团体、美术馆、展览馆、文化宫、其他 |
| 医疗 | 综合医院、专科医院、诊所、药店、体检机构、疗养院、急救中心、疾控中心、医疗器械、医疗保健、其他 |
| 汽车服务 | 汽车销售、汽车维修、汽车美容、汽车配件、汽车租赁、汽车检测场、其他 |
| 交通设施 | 飞机场、火车站、地铁站、地铁线路、长途汽车站、公交车站、公交线路、港口、停车场、加油加气站、服务区、收费站、桥、充电站、路侧停车位、普通停车位、接送点、其他 |
| 金融 | 银行、ATM、信用社、投资理财、典当行、其他 |
| 房地产 | 写字楼、住宅区、宿舍、内部楼栋、其他 |
| 公司企业 | 公司、园区、农林园艺、厂矿、其他 |
| 政府机构 | 中央机构、各级政府、行政单位、公检法机构、涉外机构、党派团体、福利机构、政治教育机构、社会团体、民主党派、居民委员会、其他 |
| 出入口 | 高速公路出口、高速公路入口、机场出口、机场入口、车站出口、车站入口、门（备注：建筑物和建筑物群的门）、停车场出入口、自行车高速出口、自行车高速入口、自行车高速出入口、其他 |
| 自然地物 | 岛屿、山峰、水系、其他 |
| 行政地标 | 省、省级城市、地级市、区县、商圈、乡镇、村庄、其他 |
| 门址 | 门址点、其他 |

引自：[百度地图开放平台](http://lbsyun.baidu.com/index.php?title=lbscloud/poitags) 更新于2020年5月11日。

#### 1）单个分类检索

检索数据，需要查看百度提供的检索方法，根据要求来配置对应参数实现下载，具体查看[服务文档](http://lbsyun.baidu.com/index.php?title=webapi/guide/webservice-placeapi)③。本次实验对应配置检索语句为，矩形区域检索:http://api.map.baidu.com/place/v2/search?query=银行&bounds=39.915,116.404,39.975,116.414&output=json&ak={您的密钥} //GET请求，百度地图例举的语句仅包含个别请求参数，实际上提供的行政区划区域检索请求参数约15个左右，可以根据对数据的下载需求来确定使用哪些请求参数。检索服务包括圆形区域检索、地点详情检索和多边形区域检索服务等。此次矩形区域检索（多边形区域检索）的请求参数配置说明如下：

query\_dic={  
 'query':检索关键字。圆形区域检索和矩形区域内检索支持多个关键字并集检索，不同关键字间以“$”符号分隔，最多支持10个关键字检索，如:”银行$酒店”。如果需要按POI分类进行检索，将分类通过query参数进行设置，例如 query='旅游景点',  
 'page\_size': 单次召回POI数量，默认为10条记录，最大返回20条。多关键字检索时，返回的记录数为“关键字个数\*page\_size”，例如 page\_size='20',   
 'page\_num':分页页码，默认为0, 0代表第一页, 1代表第二页，以此类推。常与page\_size搭配使用，例如 page\_num='0',  
 'scope':检索结果详细程度。取值为1或空，则返回基本信息；取值为2，返回检索POI详细信息，例如 scope='2',  
 'bounds':检索矩形区域，为左下角和右上角坐标，多组坐标间以“,”分隔, 例如 str(leftBottomCoordi[1]) + ',' + str(leftBottomCoordi[0]) + ','+str(rightTopCoordi[1]) + ',' + str(rightTopCoordi[0]),  
 'output':输出格式为json或者xml，例如output='json',  
 'ak':开发者的访问密钥，必填项。v2之前该属性为key,   
}

如需下载数据，请求参数需要申请访问密钥’ak’（注意，需要注册登录)。

首先来配置基本参数，最终要实现的目的是将下载后的数据分别存储为CSV（Comma-Separated Values，逗号（字符）分隔值，以纯文本形式存储表格数据）和JSON（JavaScript Object Notation，轻量级数据交换格式，层次结构简洁清晰）数据格式，因此调入json和csv库，辅助读写对应格式的文件。因为要通过网页地址来检索数据，因此调入urllib库实现相应的URL（Uniform Resource Locator，统一资源定位符，即网络地址）处理。对文件路径的管理则可以使用os及pathlib库。

注意，多边形区域检索目前为高级权限，如有需求，需要在百度地图开放平台上提交工单咨询。读者可以从本书提供的数据下载链接下载该数据，无需自行从百度地图开放平台检索下载。

* 多边形区域检索（矩形区域检索）

import os  
  
data\_path='./data' # 配置数据存储位置  
# 定义存储文件名  
poi\_fn\_csv=os.path.join(data\_path,'poi\_csv.csv')  
poi\_fn\_json=os.path.join(data\_path,'poi\_json.json')

配置请求参数，注意其中page\_num参数为页数递增，初始参数时，配置页数范围为page\_num\_range=range(20)。保存的文件类型直接由定义的存储文件名的后缀名确定。由于百度API的限制，检索区域内最多返回的POI数据量有限制，造成下载疏漏，因此如果下载区域很大时，最好是将其切分为数个矩形逐一下载，从而增加一个配置参数partition实现检索区域的切分次数，如果设置为2，则切分矩形为4份检索区域分别下载。

注意在配置坐标时，使用[百度地图坐标拾取系统](http://api.map.baidu.com/lbsapi/getpoint/index.html)④。

bound\_coordinate={'leftBottom':[108.776852,34.186027],'rightTop':[109.129275,34.382171]}   
page\_num\_range=range(20)  
partition=4   
query\_dic={  
 'query':'旅游景点',  
 'page\_size':'20',   
 'scope':2,  
 'ak':'2Zh7jNunzIzKoWx59ucjHLlZ63oI9St0',   
}

通过定义一个函数实现百度地图开放平台下，基于矩形区域检索爬取POI数据，方便日后相关项目的代码迁移或者调用，增加代码融合的力度。因此，需要谨慎确定输入参数，避免在调用时还需调整函数内的变量，造成不必要的错误。由百度地图下载数据，其经纬度坐标为百度坐标系，因此需要对其进行转换，调入转换代码文件coordinate\_transformation.py（从该书配套GitHub仓库下载）。

在坐标转换过程中，调用了两个转换函数，bd09togcj02(bd\_lon, bd\_lat)和gcj02towgs84(lng, lat)。当运行检索函数时，需要将坐标转换文件置于与该.ipynb文件（或者读者新建立包含检索函数的.py文件）同一文件夹下，以备调用。

def baiduPOI\_dataCrawler(query\_dic,bound\_coordinate,partition,page\_num\_range,poi\_fn\_list=False):  
 '''  
 function - 百度地图开放平台POI数据检索——多边形区域检索（矩形区域检索）方式  
   
 Params:  
 query\_dic - 请求参数配置字典，详细参考上文或者百度服务文档；dict  
 bound\_coordinate - 以字典形式配置下载区域；dict  
 partition - 检索区域切分次数；int  
 page\_num\_range - 配置页数范围；range()  
 poi\_fn\_list=False - 定义的存储文件名列表；list  
   
 Returns:  
 None  
 '''  
 import coordinate\_transformation as cc  
 import urllib,json,csv,pathlib  
   
 urlRoot='http://api.map.baidu.com/place/v2/search?' # 数据下载网址，查询百度地图服务文档  
 # 切分检索区域  
 if bound\_coordinate:  
 xDis=(bound\_coordinate['rightTop'][0]-bound\_coordinate['leftBottom'][0])/partition  
 yDis=(bound\_coordinate['rightTop'][1]-bound\_coordinate['leftBottom'][1])/partition   
 # 判断是否要写入文件  
 if poi\_fn\_list:  
 for file\_path in poi\_fn\_list:  
 fP=pathlib.Path(file\_path)  
 if fP.suffix=='.csv':  
 poi\_csv=open(fP,'w',encoding='utf-8')  
 csv\_writer=csv.writer(poi\_csv)   
 elif fP.suffix=='.json':  
 poi\_json=open(fP,'w',encoding='utf-8')  
 num=0  
 jsonDS=[] # 存储读取的数据，用于.json格式数据的保存  
 # 循环切分的检索区域，逐区下载数据  
 print("Start downloading data...")  
 for i in range(partition):  
 for j in range(partition):  
 leftBottomCoordi=[bound\_coordinate['leftBottom'][0]+i\*xDis,bound\_coordinate['leftBottom'][1]+j\*yDis]  
 rightTopCoordi=[bound\_coordinate['leftBottom'][0]+(i+1)\*xDis,bound\_coordinate['leftBottom'][1]+(j+1)\*yDis]  
 for p in page\_num\_range:   
 # 更新请求参数  
 query\_dic.update({'page\_num':str(p),  
 'bounds':str(leftBottomCoordi[1]) + ',' + str(leftBottomCoordi[0]) + ',' +   
 str(rightTopCoordi[1]) + ',' + str(rightTopCoordi[0]),  
 'output':'json',  
 })  
   
 url=urlRoot+urllib.parse.urlencode(query\_dic)  
 data=urllib.request.urlopen(url)  
 responseOfLoad=json.loads(data.read())   
 # print(url,responseOfLoad.get("message"))  
 if responseOfLoad.get("message")=='ok':  
 results=responseOfLoad.get("results")   
 for row in range(len(results)):  
 subData=results[row]  
 baidu\_coordinateSystem=[subData.get('location').get('lng'),subData.get('location').get('lat')] # 获取百度坐标系  
 Mars\_coordinateSystem=cc.bd09togcj02(baidu\_coordinateSystem[0], baidu\_coordinateSystem[1]) # 百度坐标系-->火星坐标系  
 WGS84\_coordinateSystem=cc.gcj02towgs84(Mars\_coordinateSystem[0], Mars\_coordinateSystem[1]) # 火星坐标系-->WGS84  
   
 # 更新坐标  
 subData['location']['lat']=WGS84\_coordinateSystem[1]  
 subData['detail\_info']['lat']=WGS84\_coordinateSystem[1]  
 subData['location']['lng']=WGS84\_coordinateSystem[0]  
 subData['detail\_info']['lng']=WGS84\_coordinateSystem[0]   
 if csv\_writer:  
 csv\_writer.writerow([subData]) # 逐行写入.csv文件  
 jsonDS.append(subData)  
 num+=1   
 print("No."+str(num)+" was written to the .csv file.")  
 if poi\_json:   
 json.dump(jsonDS,poi\_json)  
 poi\_json.write('\n')  
 poi\_json.close()  
 if poi\_csv:  
 poi\_csv.close()  
 print("The download is complete.")  
  
baiduPOI\_dataCrawler(query\_dic,bound\_coordinate,partition,page\_num\_range,poi\_fn\_list=[poi\_fn\_csv,poi\_fn\_json])

Start downloading data...  
No.1 was written to the .csv file.  
No.2 was written to the .csv file.  
No.3 was written to the .csv file.  
No.4 was written to the .csv file.  
No.5 was written to the .csv file.  
No.6 was written to the .csv file.  
No.7 was written to the .csv file.  
No.8 was written to the .csv file.  
No.9 was written to the .csv file.  
No.10 was written to the .csv file.  
No.11 was written to the .csv file.  
No.12 was written to the .csv file.  
No.13 was written to the .csv file.  
No.14 was written to the .csv file.  
No.15 was written to the .csv file.  
No.16 was written to the .csv file.  
The download is complete.

* 圆形区域检索（百度地图开放平台没有限制该方法）

可设置圆心和半径，检索圆形区域内的地点信息。官方给出的检索方式为：https://api.map.baidu.com/place/v2/search?query=银行&location=39.915,116.404&radius=2000&output=xml&ak=您的密钥 //GET请求

def baiduPOI\_dataCrawler\_circle(query\_dic,poi\_save\_path,page\_num\_range):  
 '''  
 function - 百度地图开放平台POI数据检索——圆形区域检索方式  
   
 Params:  
 query\_dic - 请求参数配置字典，详细参考上文或者百度服务文档；dict  
 poi\_save\_path - 存储文件路径；string  
 page\_num\_range - 配置页数范围；range()  
   
 Returns:  
 None  
 '''  
 import coordinate\_transformation as cc  
 import urllib,json,csv,os,pathlib  
 from tqdm import tqdm  
   
 urlRoot='http://api.map.baidu.com/place/v2/search?' #数据下载网址，查询百度地图服务文档  
 poi\_json=open(poi\_save\_path,'w',encoding='utf-8')   
 jsonDS=[] # 存储读取的数据，用于.json格式数据的保存  
 for p in tqdm(page\_num\_range):   
 # 更新请求参数  
 query\_dic.update({'page\_num':str(p)})  
 url=urlRoot+urllib.parse.urlencode(query\_dic)  
 data=urllib.request.urlopen(url)  
 responseOfLoad=json.loads(data.read())   
 if responseOfLoad.get("message")=='ok':  
 results=responseOfLoad.get("results")   
 for row in range(len(results)):  
 subData=results[row]  
 baidu\_coordinateSystem=[subData.get('location').get('lng'),subData.get('location').get('lat')] # 获取百度坐标系  
 Mars\_coordinateSystem=cc.bd09togcj02(baidu\_coordinateSystem[0], baidu\_coordinateSystem[1]) # 百度坐标系-->火星坐标系  
 WGS84\_coordinateSystem=cc.gcj02towgs84(Mars\_coordinateSystem[0],Mars\_coordinateSystem[1]) # 火星坐标系-->WGS84  
  
 # 更新坐标  
 subData['location']['lat']=WGS84\_coordinateSystem[1]  
 subData['detail\_info']['lat']=WGS84\_coordinateSystem[1]  
 subData['location']['lng']=WGS84\_coordinateSystem[0]  
 subData['detail\_info']['lng']=WGS84\_coordinateSystem[0]   
 jsonDS.append(subData)  
 if poi\_json:   
 json.dump(jsonDS,poi\_json)  
 poi\_json.write('\n')  
 poi\_json.close()  
 print("The download is complete.")   
   
page\_num\_range=range(20)  
query\_dic={  
 'location':'34.265708,108.953431',  
 'radius':1000,  
 'query':'旅游景点',   
 'page\_size':'20',  
 'scope':2,   
 'output':'json',  
 'ak':'YuN8HxzYhGNfNLGX0FVo3NU3NOrgSNdF'   
}   
poi\_save\_path='./data/poi\_circle.json'  
baiduPOI\_dataCrawler\_circle(query\_dic,poi\_save\_path,page\_num\_range)

100%|██████████| 20/20 [00:05<00:00, 3.82it/s]  
  
The download is complete.

import json  
  
with open(poi\_save\_path) as f:  
 poi\_circle=json.load(f)  
print(poi\_circle[0])

{'name': '西安城墙', 'location': {'lat': 34.25321785828024, 'lng': 108.94356940023889}, 'address': '陕西省西安市碑林区南大街', 'province': '陕西省', 'city': '西安市', 'area': '碑林区', 'street\_id': '6e7e1320b113d8e3bea82230', 'telephone': '(029)87272792', 'detail': 1, 'uid': '6e7e1320b113d8e3bea82230', 'detail\_info': {'distance': 874, 'tag': '旅游景点;文物古迹', 'navi\_location': {'lng': 108.95260787133, 'lat': 34.259100951615}, 'type': 'scope', 'detail\_url': 'http://api.map.baidu.com/place/detail?uid=6e7e1320b113d8e3bea82230&output=html&source=placeapi\_v2', 'overall\_rating': '4.3', 'comment\_num': '200', 'shop\_hours': '08:00-22:00', 'children': [], 'lat': 34.25321785828024, 'lng': 108.94356940023889}}

行政区划区域检索、地点详情检索等方法可以查看[服务文档](http://lbsyun.baidu.com/index.php?title=webapi/guide/webservice-placeapi)。

#### 2）将CSV格式的POI数据转换为pandas的DataFrame格式数据

读取已经保存的poi\_csv.csv文件时，因为在文件保存时，使用的是csv和json库，因此读取时仍旧使用对应的库。经常使用的pandas库中也有read\_csv()和read\_json()等方法。在将数据存储为CSV格式文件，因为保存的数据格式可能存在差异，因此pandas读取CSV或者JSON文件最好是其自身所存储的文件，数据格式能够对应。如果直接读取上述保存的POI的.csv文件，则会出现错误。只有知道数据格式，才能够有目的的提取数据，读取每一行的数据格式如下：

{  
 "name": "昆明池遗址",  
 "location": {  
 "lat": 34.210991,  
 "lng": 108.779778  
 },  
 "address": "西安市长安区昆明池七夕公园内",  
 "province": "陕西省",  
 "city": "西安市",  
 "area": "长安区",  
 "detail": 1,  
 "uid": "c7332cd7fbcc0d82ebe582d9",  
 "detail\_info": {  
 "tag": "旅游景点;景点",  
 "navi\_location": {  
 "lng": 108.7812626866,  
 "lat": 34.217484892966  
 },  
 "type": "scope",  
 "detail\_url": "http://api.map.baidu.com/place/detail?uid=c7332cd7fbcc0d82ebe582d9&amp;output=html&amp;source=placeapi\_v2",  
 "overall\_rating": "4.3",  
 "comment\_num": "77",  
 "children": []  
 }  
}

读取.csv数据之后，直接使用CSV格式数据来提取和分析数据并不方便。最为常用的数据格式是NumPy库提供的数组（array）和pandas提供的DataFrame及Series。其中NumPy的数据组织形式更倾向于科学计算，为数阵形式，每一数组为同一数据类型；而pandas的DataFrame与地理信息数据中的属性表类似，其列（column）可以理解为属性字段（field），每一列的数据类型相同，因此一个DataFrame可以包含多种数据类型。就POI的CSV格式数据，将其转换为DataFrame的数据格式是最好的。进一步而言，在城市空间数据分析方法中，更多的数据是基于地理空间位置的，因此对于具有地理属性的数据，最好能够以地理信息系统常用的数据格式来处理。在处理为DataFrame数据格式之后，则进一步应用GeoPandas等地理信息库来处理地理空间信息数据。

在数据格式转换时可能会出现错误，例如原始数据可能存在错误的数据格式：

{'name': '励进海升酒店-多功能厅', 'location': {'lat': 34.218525, 'lng': 108.891524}, 'address': '西安市高新区沣惠南路34号励进海升酒店4层', 'province': '陕西省', 'city': '西安市', 'a{'name': '红蚂蚁少儿创意美术馆'", " 'location': {'lat': 34.306666", " 'lng': 108.822465}", " 'address': '陕西省西安市未央区后围寨启航佳苑B区3层商铺'", " 'province': '陕西省'", " 'city': '西安市'", " 'area': '未央区'", " 'telephone': '18209227178", "15229372642'", " 'detail': 1", " 'uid': 'e3fd730bb528b40015c9050c'", " 'detail\_info': {'tag': '文化传媒;美术馆'", " 'type': 'scope'", " 'detail\_url': 'http://api.map.baidu.com/place/detail?uid=e3fd730bb528b40015c9050c&amp;output=html&amp;source=placeapi\_v2'", " 'overall\_rating': '0.0'", ' \'children\': []}}

其中'a{'name': '红蚂蚁少儿创意美术馆'",为多余的部分，并不符合任何格式语法，因此在数据处理时，需要对此做出反应。通常使用try/except语句，最好返回可以检索的信息，加以处理，避免数据损失。

def csv2df(poi\_fn\_csv):  
 '''  
 function-转换CSV格式的POI数据为pandas的DataFrame  
   
 Params:  
 poi\_fn\_csv - 存储有POI数据的CSV格式文件路径   
   
 Returns:  
 poi\_df - DataFrame(pandas)  
 '''  
 import pandas as pd  
 from benedict import benedict # benedict库是dict的子类，支持键列表（keylist）/键路径（keypath）。用该库的flatten方法展平嵌套字典，用于DataFrame数据结构  
 import csv  
   
 n=0  
 with open(poi\_fn\_csv, newline='',encoding='utf-8') as csvfile:  
 poi\_reader=csv.reader(csvfile)  
 poi\_dict={}   
 poiExceptions\_dict={}  
 for row in poi\_reader:   
 if row:  
 try:  
 row\_benedict=benedict(eval(row[0])) # 用eval方法，将字符串字典"{}"转换为字典{}  
 flatten\_dict=row\_benedict.flatten(separator='\_') # 展平嵌套字典  
 poi\_dict[n]=flatten\_dict  
 except:   
 print("incorrect format of data\_row number:%s"%n)   
 poiExceptions\_dict[n]=row  
 n+=1  
 # if n==5:break # 因为循环次数比较多，在调试代码时，可以设置停止的条件，节省时间与方便数据查看  
 poi\_df=pd.concat([pd.DataFrame(poi\_dict[d\_k].values(),index=poi\_dict[d\_k].keys(),columns=[d\_k]).T for d\_k in poi\_dict.keys()], sort=True,axis=0)  
 print("\_"\*50)  
 for col in poi\_df.columns:  
 try:  
 poi\_df[col]=pd.to\_numeric(poi\_df[col])  
 except:  
 print("%s data type is not converted..."%(col))  
 print("\_"\*50)  
 print(".csv to DataFrame completed!")  
 return poi\_df

几乎Python的所有数据类型，列表、字典、集合和类等都可以用pickle来序列化存储，pandas提供了写入pandas.DataFrame.to\_pickle，读取pandas.DataFrame.read\_pickle的方法，因此为了避免每次将CSV转换为DataFrame，可以将pandas类型的文件按pandas提供的方法加以存储，或写入数据库。

poi\_fn\_csv='./data/poi\_csv.csv'  
poi\_df=csv2df(poi\_fn\_csv)  
poi\_df.to\_pickle('./data/poi\_df.pkl')  
  
print("\_"\*50)  
print(poi\_df.columns)  
poi\_df.head()

incorrect format of data\_row number:4  
incorrect format of data\_row number:28  
incorrect format of data\_row number:36  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
address data type is not converted...  
area data type is not converted...  
city data type is not converted...  
detail\_info\_detail\_url data type is not converted...  
detail\_info\_indoor\_floor data type is not converted...  
detail\_info\_tag data type is not converted...  
detail\_info\_type data type is not converted...  
name data type is not converted...  
province data type is not converted...  
street\_id data type is not converted...  
telephone data type is not converted...  
uid data type is not converted...  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Index(['address', 'area', 'city', 'detail', 'detail\_info\_checkin\_num',  
 'detail\_info\_children', 'detail\_info\_comment\_num',  
 'detail\_info\_detail\_url', 'detail\_info\_facility\_rating',  
 'detail\_info\_favorite\_num', 'detail\_info\_hygiene\_rating',  
 'detail\_info\_image\_num', 'detail\_info\_indoor\_floor', 'detail\_info\_lat',  
 'detail\_info\_lng', 'detail\_info\_navi\_location\_lat',  
 'detail\_info\_navi\_location\_lng', 'detail\_info\_overall\_rating',  
 'detail\_info\_price', 'detail\_info\_service\_rating', 'detail\_info\_tag',  
 'detail\_info\_type', 'location\_lat', 'location\_lng', 'name', 'province',  
 'street\_id', 'telephone', 'uid'],  
 dtype='object')

|  | **address** | **area** | **city** | **detail** | **detail\_info\_checkin\_num** | **detail\_info\_children** | **detail\_info\_comment\_num** | **detail\_info\_detail\_url** | **detail\_info\_facility\_rating** | **detail\_info\_favorite\_num** | **...** | **detail\_info\_service\_rating** | **detail\_info\_tag** | **detail\_info\_type** | **location\_lat** | **location\_lng** | **name** | **province** | **street\_id** | **telephone** | **uid** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 西安市长安区昆明池七夕公园内 | 长安区 | 西安市 | 1 | NaN | NaN | 29.0 | http://api.map.baidu.com/place/detail?uid=c733... | NaN | NaN | ... | NaN | 旅游景点;景点 | scope | 34.206767 | 108.768459 | 昆明池遗址 | 陕西省 | NaN | NaN | c7332cd7fbcc0d82ebe582d9 |
| **2** | 西安市高新区云萃路 | 雁塔区 | 西安市 | 1 | NaN | NaN | 34.0 | http://api.map.baidu.com/place/detail?uid=6ccb... | NaN | NaN | ... | NaN | 旅游景点;公园 | scope | 34.215761 | 108.835407 | 云水公园 | 陕西省 | 6ccb87a451f19a27626858b9 | NaN | 6ccb87a451f19a27626858b9 |
| **6** | 西安市长安区昆明池·七夕公园内 | 长安区 | 西安市 | 1 | NaN | NaN | 5.0 | http://api.map.baidu.com/place/detail?uid=66f2... | NaN | NaN | ... | NaN | 旅游景点;景点 | scope | 34.214174 | 108.765677 | 汉武大帝 | 陕西省 | NaN | NaN | 66f2005e22dcafdbc7f50d07 |
| **8** | 西安市长安区沣东新城斗门新型工业园区汇新路北段昆明池·七夕公园内 | 长安区 | 西安市 | 1 | NaN | NaN | 3.0 | http://api.map.baidu.com/place/detail?uid=576a... | NaN | NaN | ... | NaN | 旅游景点;景点 | scope | 34.211592 | 108.765670 | 凤求凰 | 陕西省 | NaN | NaN | 576a6060e0ea8ae21ce999c0 |
| **10** | 陕西省西安市雁塔区西宝疏导线关中驾校南200米 | 雁塔区 | 西安市 | 1 | NaN | NaN | 1.0 | http://api.map.baidu.com/place/detail?uid=8ed2... | NaN | NaN | ... | NaN | 旅游景点;其他 | scope | 34.224172 | 108.815592 | 大明怀远将军园 | 陕西省 | 8ed2e7b406ac382b4bcc4a18 | NaN | 8ed2e7b406ac382b4bcc4a18 |

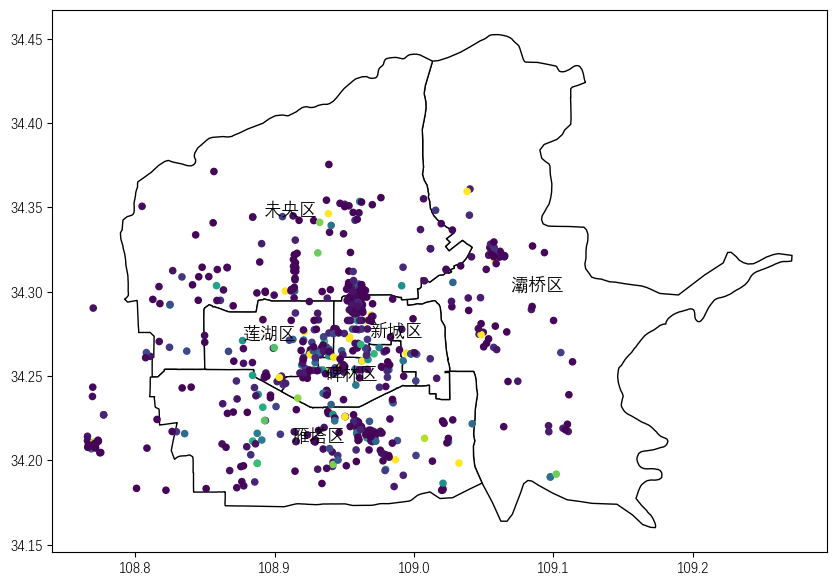
5 rows × 29 columns

#### 3）将数据格式为DataFrame的POI数据转换为GeoPandas地理空间数据GeoDataFrame

可以使用GeoPandas库将pandas的DataFrame和Series数据转换为具有地理意义的GeoDataFrame和GeoSeries地理数据。GeoPandas基于pandas库，因此在数据结构上二者之间最大的区别是GeoPandas有一个列（column）名为’geometry’，用来存储几何数据，例如点数据POINT (163.85316 -17.31631)，单个多边形POLYGON ((33.90371 -0.95000, 31.86617 -1.02736...))，多边形集合MULTIPOLYGON (((120.83390 12.70450, 120.32344 ...)))等。其中对几何数据的表达是使用shapely库实现。大部分SHP地理信息矢量数据，在Python语言中，通常使用该库来建立几何对象。同时GeoDataFrame对象的建立，需要配置坐标系统，这是地理信息数据显著的一个标志，对于坐标系统可以查看[Spatial Reference](https://spatialreference.org/)⑤。

GeoDataFrame可以直接通过.plot()方法显示地理空间信息数据。

import pandas as pd  
import geopandas as gpd  
from shapely.geometry import Point  
  
poi\_df=pd.read\_pickle('./data/poi\_df.pkl') # 读取已经保存的.pkl(pickle)数据格式的POI  
poi\_2gdf=poi\_df.copy(deep=True)  
poi\_2gdf['geometry']=poi\_2gdf.apply(lambda row:Point(row.location\_lng,row.location\_lat),axis=1)   
epsg\_wgs84=4326 # 配置坐标系统，参考：https://spatialreference.org/   
poi\_gdf=gpd.GeoDataFrame(poi\_2gdf,crs=epsg\_wgs84)  
  
xian\_region\_gdf=gpd.read\_file('./data/xian\_region/xian\_region.shp')  
  
import matplotlib.pyplot as plt  
from pylab import mpl  
mpl.rcParams['font.sans-serif']=['DengXian'] # 解决中文字符乱码问题  
  
fig, ax=plt.subplots(figsize=(10,15))  
xian\_region\_plot=xian\_region\_gdf.plot(ax=ax,color='white', edgecolor='black')  
xian\_region\_gdf.apply(lambda row: xian\_region\_plot.annotate(text=row.NAME, xy=row.geometry.centroid.coords[0], ha='center',fontsize=13),axis=1) #标注  
poi\_gdf.plot(ax=ax,column='detail\_info\_comment\_num',markersize=20)  
  
plt.show()

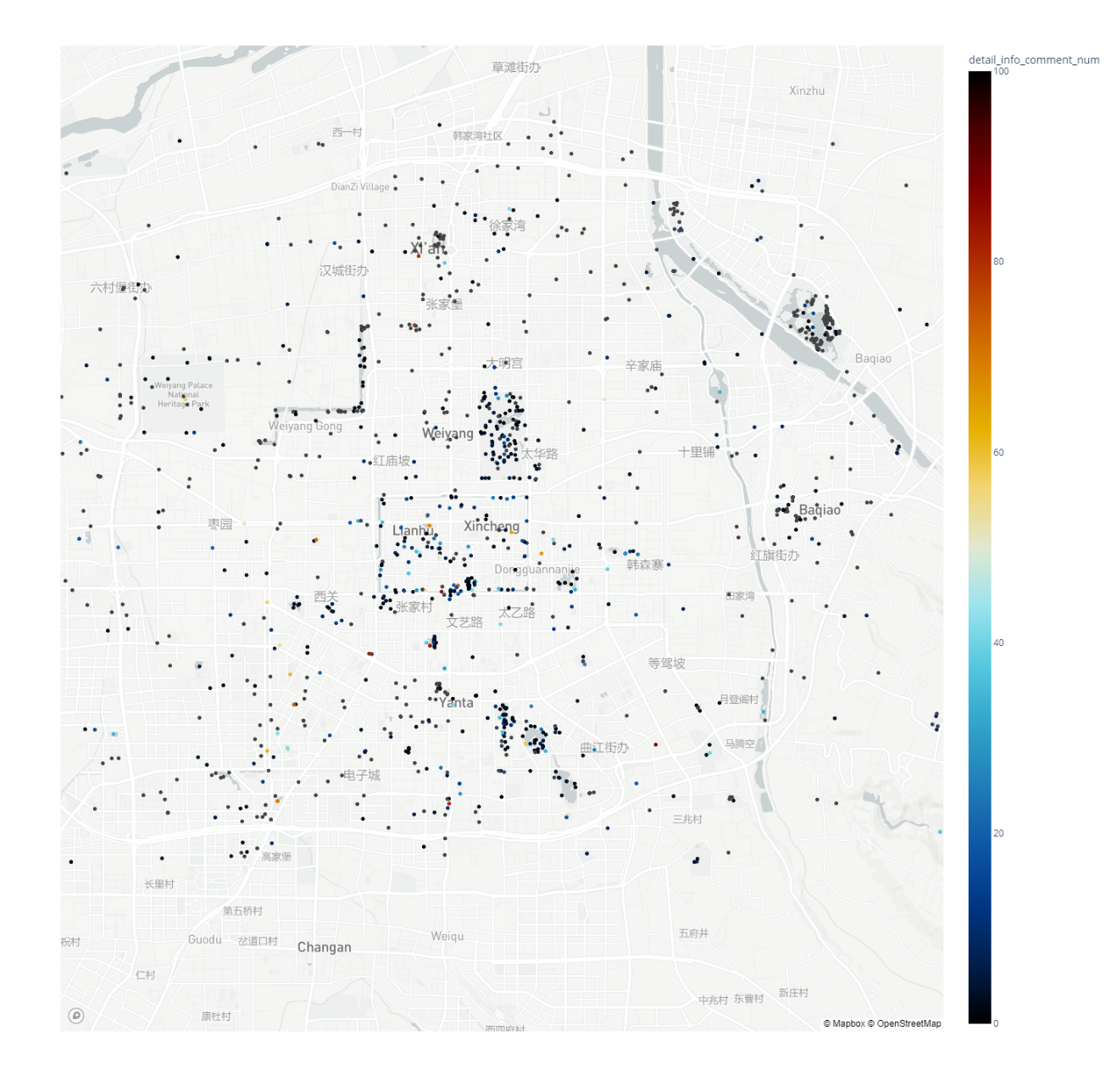


* 使用Plotly库建立地图

GeoPandas库提供的地图显示类型功能有限，但却便捷，通常用于数据的查看。当需要打印的地图具有一定质量，表达更多信息，甚至能够交互操作时，可以使用[Plotly](https://plotly.com/)⑥图表库实现（通常使用[Plotly Python](https://plotly.com/python/)⑦）。下述地图打印结果的底图使用了[mapbox](https://www.mapbox.com/)⑧提供的地图数据。要使用其功能，首先需要注册，并获取访问许可（access token）。

使用Plotly库建立地图，不需要将DataFrame转换为GeoDataFrame。

import plotly.express as px  
  
poi\_gdf.detail\_info\_price=poi\_gdf.detail\_info\_price.fillna(0) # pandas库的方法同样适用于GeoPandas库，例如对`nan`位置填充指定数值  
mapbox\_token='pk.eyJ1IjoicmljaGllYmFvIiwiYSI6ImNrYjB3N2NyMzBlMG8yc254dTRzNnMyeHMifQ.QT7MdjQKs9Y6OtaJaJAn0A'  
  
px.set\_mapbox\_access\_token(mapbox\_token)  
fig=px.scatter\_mapbox(poi\_gdf,  
 lat=poi\_gdf.location\_lat,   
 lon=poi\_gdf.location\_lng,  
 color="detail\_info\_comment\_num",  
 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()  
fig.write\_html('./graph/POI\_singlClassi.html') #保存地图



### 2.1.1.2 多个分类POI数据检索

#### 1）多个分类POI检索

前文定义了两个函数，百度地图开放平台POI数据检索baiduPOI\_dataCrawler()和转换CSV格式的POI数据为pandas的DataFrame格式数据csv2df()。为了方便应用建立的函数工具，使用Anaconda的Spyder创建一个新的文件（模块）为util\_A.py，将上述两个函数置于其中，同时包括函数所使用的库。对于所包括的库，为方便日后函数迁移和打包自定义Python库发布，明确每个函数所调用的库，将对应调用库的语句分别置于各个函数内部。util\_A.py与调用该工具的代码文件置于同一文件夹下。调入语句如下：

import util\_A

根据百度地图一级行业分类，建立映射字典，用于多个分类POI的数据检索。可以根据数据分析的需求选择行业分类，因为在进一步分析中不需要的分类包括出入口、自然地物、行政地标和门址等，因此在映射字典中未包含上述分类。

poi\_classificationName={  
 "美食 ":"delicacy",  
 "酒店 ":"hotel",  
 "购物 ":"shopping",  
 "生活服务":"lifeService",  
 "丽人 ":"beauty",  
 "旅游景点":"spot",  
 "休闲娱乐":"entertainment",  
 "运动健身":"sports",  
 "教育培训":"education",  
 "文化传媒":"media",  
 "医疗 ":"medicalTreatment",  
 "汽车服务":"carService",  
 "交通设施":"trafficFacilities",  
 "金融":"finance",  
 "房地产":"realEstate",  
 "公司企业":"corporation",  
 "政府机构":"government"  
 }

配置基本参数。上文配置用到query\_dic。此次批量下载将所有参数在循环函数外以字典形式单独给出，方便调用。而query\_dic字典参数在批量下载函数内配置。

query\_dic={  
 'query':'旅游景点',  
 'page\_size':'20',  
 'scope':2,  
 'ak':'2Zh7jNunzIzKoWx59ucjHLlZ63oI9St0',  
}

poi\_config\_para={  
 'data\_path':'./data/poi\_batchCrawler/', # 配置数据存储位置  
 'bound\_coordinate':{'leftBottom':[108.776852,34.186027],'rightTop':[109.129275,34.382171]}, # 使用百度地图坐标拾取系统 http://api.map.baidu.com/lbsapi/getpoint/index.html  
 'page\_num\_range':range(20),  
 'partition':6,   
 'page\_size':'20',   
 'scope':2,  
 'ak':'2Zh7jNunzIzKoWx59ucjHLlZ63oI9St0', #需要改为读者自行申请的开发者访问密钥  
}

建立批量下载的循环函数，依据给出的poi\_classificationName字典键值逐次调用单个分类POI检索函数baiduPOI\_dataCrawler()下载POI数据。在检索过程中，可以将每一次小批量下载数据存储在同一变量下，待全部下载完后一次性存储。但是这种一次性存储的方式并不推荐，其一，网络有时并不稳定，可能造成下载中断，那么已下载的数据未存储，造成数据丢失，并需要重新下载；其二，有时数据量很大，如果都存储在一个变量下，可能造成内存溢出。

def baiduPOI\_batchCrawler(poi\_config\_para):   
 '''  
 function - 百度地图开放平台POI数据批量爬取，  
 需要调用单个分类POI检索函数 baiduPOI\_dataCrawler(query\_dic,bound\_coordinate,partition,page\_num\_range,poi\_fn\_list=False)  
   
 Params:  
 poi\_config\_para - 参数配置，包含：  
 'data\_path'（配置数据存储位置），  
 'bound\_coordinate'（矩形区域检索坐下、右上经纬度坐标），  
 'page\_num\_range'（配置页数范围），  
 'partition'（检索区域切分次数），  
 'page\_size'（单次召回POI数量），  
 'scope'（检索结果详细程度），  
 'ak'（开发者的访问密钥）  
   
 Returns:  
 None  
 '''  
 import os  
 import util\_A   
   
 for idx,(poi\_ClassiName,poi\_classMapping) in enumerate(poi\_classificationName.items()):  
 print(str(idx+16)+"\_"+poi\_ClassiName)  
 poi\_subFileName="poi\_"+str(idx+16)+"\_"+poi\_classMapping  
 data\_path=poi\_config\_para['data\_path']  
 poi\_fn\_csv=os.path.join(data\_path,poi\_subFileName+'.csv')  
 poi\_fn\_json=os.path.join(data\_path,poi\_subFileName+'.json')  
   
 query\_dic={  
 'query':poi\_ClassiName,  
 'page\_size':poi\_config\_para['page\_size'],  
 'scope':poi\_config\_para['scope'],  
 'ak':poi\_config\_para['ak']   
 }  
 bound\_coordinate=poi\_config\_para['bound\_coordinate']  
 partition=poi\_config\_para['partition']  
 page\_num\_range=poi\_config\_para['page\_num\_range']  
 util\_A.baiduPOI\_dataCrawler(query\_dic,bound\_coordinate,partition,page\_num\_range,poi\_fn\_list=[poi\_fn\_csv,poi\_fn\_json])   
   
baiduPOI\_batchCrawler(poi\_config\_para)

16\_美食   
Start downloading data...  
No.1 was written to the .csv file.  
No.2 was written to the .csv file.  
No.3 was written to the .csv file.  
No.4 was written to the .csv file.  
No.5 was written to the .csv file.  
No.6 was written to the .csv file.  
No.7 was written to the .csv file.  
No.8 was written to the .csv file.  
No.9 was written to the .csv file.  
No.10 was written to the .csv file.  
No.11 was written to the .csv file.  
No.12 was written to the .csv file.  
No.13 was written to the .csv file.  
No.14 was written to the .csv file.  
No.15 was written to the .csv file.  
No.16 was written to the .csv file.  
No.17 was written to the .csv file.  
No.18 was written to the .csv file.  
No.19 was written to the .csv file.  
No.20 was written to the .csv file.  
No.21 was written to the .csv file.  
No.22 was written to the .csv file.  
No.23 was written to the .csv file.  
No.24 was written to the .csv file.  
No.25 was written to the .csv file.  
No.26 was written to the .csv file.  
No.27 was written to the .csv file.  
No.28 was written to the .csv file.  
No.29 was written to the .csv file.  
No.30 was written to the .csv file.  
No.31 was written to the .csv file.  
No.32 was written to the .csv file.  
No.33 was written to the .csv file.  
No.34 was written to the .csv file.  
No.35 was written to the .csv file.  
No.36 was written to the .csv file.  
The download is complete.

#### 2）批量转换CSV格式数据为GeoDataFrame格式

在单个分类部分定义了CSV格式数据到GeoDataFrame数据的转换方法csv2df()。基于已有代码，该部分将达到两个目的，一是定义单独函数实现CSV格式数据批量转换为GeoDataFame格式数据并存储为.pkl文件；二是批量读取存储为.pkl文件的GeoDataFrame格式数据，并根据需要提取信息存储在单一变量下，再存储为.pkl文件。当读取所有数据于单一变量时，内存需要满足要求，如果有内存溢出，则需考虑是否根据内存情况调整每次读取的数据量。

##### 1. 定义提取文件夹下所有文件路径的函数

因为要批量下载POI数据为多个.csv文件及.json文件，因此在批量处理这些数据时，第一件事是要提取所有文件的路径。定义返回所有指定后缀名文件路径的函数为常用的函数之一，在之后的很多实验中，都需要调用该函数，因此可以将其保存在util\_misc.py文件中，方便日后调用。文件夹下通常包括子文件夹，需要使用os.walk()方法遍历目录，给出条件语句判断是否存在子文件夹，如果存在则需要返回该文件夹下的文件路径。

def filePath\_extraction(dirpath,fileType):   
 '''  
 funciton - 以所在文件夹路径为键，值为包含该文件夹下所有文件名的列表。文件类型可以自行定义   
   
 Params:  
 dirpath - 根目录，存储所有待读取的文件；string  
 fileType - 待读取文件的类型；list(string)  
   
 Returns:  
 filePath\_Info - 文件路径字典，文件夹路径为键，文件夹下的文件名列表为值；dict  
 '''  
 import os  
   
 filePath\_Info={}  
 i=0  
 for dirpath,dirNames,fileNames in os.walk(dirpath): # os.walk()遍历目录，使用help(os.walk)查看返回值解释  
 i+=1  
 if fileNames: # 仅当文件夹中有文件时才提取  
 tempList=[f for f in fileNames if f.split('.')[-1] in fileType]  
 if tempList: # 剔除文件名列表为空的情况，即文件夹下存在不为指定文件类型的文件时，上一步列表会返回空列表[]  
 filePath\_Info.setdefault(dirpath,tempList)  
 return filePath\_Info  
  
dirpath='./data/poi\_batchCrawler/'  
fileType=["csv"]  
poi\_paths=filePath\_extraction(dirpath,fileType)  
print(poi\_paths)

{'./data/poi\_batchCrawler/': ['poi\_0\_delicacy.csv', 'poi\_10\_medicalTreatment.csv', 'poi\_11\_carService.csv', 'poi\_12\_trafficFacilities.csv', 'poi\_13\_finance.csv', 'poi\_14\_realEstate.csv', 'poi\_15\_corporation.csv', 'poi\_16\_government.csv', 'poi\_1\_hotel.csv', 'poi\_2\_shopping.csv', 'poi\_3\_lifeService.csv', 'poi\_4\_beauty.csv', 'poi\_5\_spot.csv', 'poi\_6\_entertainment.csv', 'poi\_7\_sports.csv', 'poi\_8\_education.csv', 'poi\_9\_media.csv']}

##### 2. CSV格式POI数据批量转换为GeoDataFrame格式数据

Index(['address', 'area', 'city', 'detail', 'detail\_info\_checkin\_num','detail\_info\_children', 'detail\_info\_comment\_num',  
 'detail\_info\_detail\_url', 'detail\_info\_facility\_rating','detail\_info\_favorite\_num', 'detail\_info\_hygiene\_rating',  
 'detail\_info\_image\_num', 'detail\_info\_indoor\_floor','detail\_info\_navi\_location\_lat', 'detail\_info\_navi\_location\_lng',  
 'detail\_info\_overall\_rating', 'detail\_info\_price','detail\_info\_service\_rating', 'detail\_info\_tag', 'detail\_info\_type',  
 'location\_lat', 'location\_lng', 'name', 'province', 'street\_id','telephone', 'uid'],dtype='object')

上述为POI数据字段，可以用来确定提取的字段名。除了增加循环语句循环POI的.csv文件，逐一转换为pandas的DataFrame格式，再进一步转换为GeoPandas的GeoDataFrame格式之外，其它的条件同前文所述。用GeoDataFrame.plot()方法初步查看地理空间信息数据。

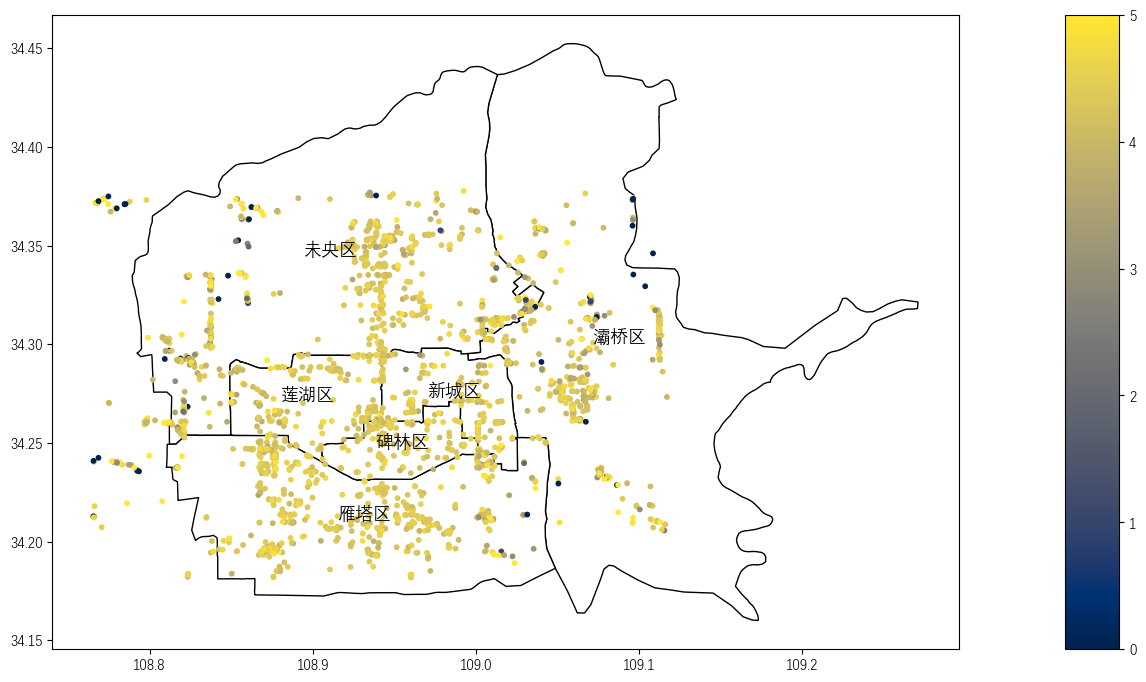
在文件保存部分，可以有多种保存格式，GeoPandas提供了Shapefile、GeoJSON和GeoPackage三种方式，也可以使用pickle库保存数据，或存入数据库。用GeoPandas提供的保存格式再读取后不再包含多重索引，而pickle格式则保持。

转换为SHP格式文件时，在QGIS等桌面GIS平台下打开时会出现两个问题，一是，如果列（字段）名称过长，转化为属性表的字段名会被字段压缩修改，往往不能反映字段的意义，因此需要置换列名称；二是，用POI一级行业分类名作为index时，列中并不包含该字段，转化为SHP文件时也不包含该字段，因此需要将index转换为列，再存储为.shp文件。

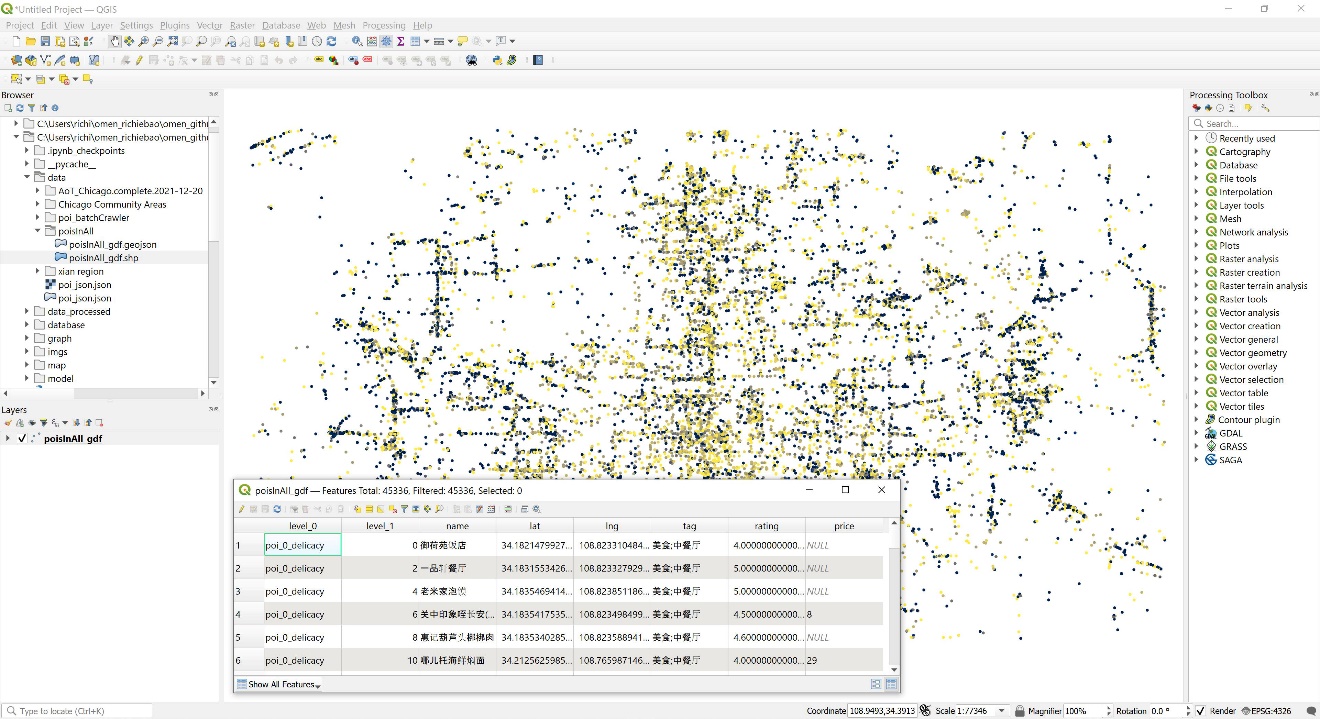
def poi\_csv2GeoDF\_batch(poi\_paths,fields\_extraction,save\_path):   
 '''  
 funciton - CSV格式POI数据批量转换为GeoDataFrame格式数据，需要调用转换CSV格式的POI数据为pandas的DataFrame函数csv2df(poi\_fn\_csv)  
   
 Params:  
 poi\_paths - 文件夹路径为键，值为包含该文件夹下所有文件名列表的字典；dict  
 fields\_extraction - 配置需要提取的字段；list(string)  
 save\_path - 存储数据格式及保存路径的字典；string  
   
 Returns:  
 poisInAll\_gdf - 提取给定字段的POI数据；GeoDataFrame（GeoPandas）  
 '''  
 import os,pathlib  
 import util\_A  
 import pandas as pd  
 import geopandas as gpd  
 from shapely.geometry import Point  
 from pyproj import CRS  
   
 poi\_df\_dic={}  
 i=0  
 for key in poi\_paths:  
 for val in poi\_paths[key]:  
 poi\_csvPath=os.path.join(key,val)  
 poi\_df=util\_A.csv2df(poi\_csvPath)   
 # 注释掉csv2df() 函数内部的print("%s data type is not converted..."%(col))语句，以pass替代，减少提示内容，避免干扰  
 print(val)  
 poi\_df\_path=pathlib.Path(val)  
 poi\_df\_dic[poi\_df\_path.stem]=poi\_df  
 i+=1  
   
 poi\_df\_concat=pd.concat(poi\_df\_dic.values(),keys=poi\_df\_dic.keys(),sort=True)  
 # print(poi\_df\_concat.loc[['poi\_0\_delicacy'],:]) # 提取index为 'poi\_0\_delicacy'的行，验证结果  
 poi\_fieldsExtraction=poi\_df\_concat.loc[:,fields\_extraction]  
 poi\_geoDF=poi\_fieldsExtraction.copy(deep=True)  
 poi\_geoDF['geometry']=poi\_geoDF.apply(lambda row:Point(row.location\_lng,row.location\_lat),axis=1)   
 crs\_4326=CRS('epsg:4326') # 配置坐标系统，参考：https://spatialreference.org/   
 poisInAll\_gdf=gpd.GeoDataFrame(poi\_geoDF,crs=crs\_4326)  
   
 poisInAll\_gdf.to\_pickle(save\_path['pkl'])  
 poisInAll\_gdf.to\_file(save\_path['geojson'],driver='GeoJSON',encoding='utf-8')  
   
 poisInAll\_gdf2shp=poisInAll\_gdf.reset\_index() # 不指定level参数，例如Level=0，会把多重索引中的所有索引转换为列  
 poisInAll\_gdf2shp.rename(columns={  
 'location\_lat':'lat', 'location\_lng':'lng',  
 'detail\_info\_tag':'tag','detail\_info\_overall\_rating':'rating', 'detail\_info\_price':'price'},inplace=True)  
 poisInAll\_gdf2shp.to\_file(save\_path['shp'],encoding='utf-8')  
   
 return poisInAll\_gdf  
  
fields\_extraction=['name','location\_lat', 'location\_lng',  
 'detail\_info\_tag','detail\_info\_overall\_rating',   
 'detail\_info\_price'] # 配置需要提取的字段，即列（columns）  
# 分别存储为GeoJSON、Shapefile和pickle三种数据格式   
save\_path={'geojson':'./data/poisInAll/poisInAll\_gdf.geojson',  
 'shp':'./data/poisInAll/poisInAll\_gdf.shp',  
 'pkl':'./data/poisInAll/poisInAll\_gdf.pkl'}   
poisInAll\_gdf=poi\_csv2GeoDF\_batch(poi\_paths,fields\_extraction,save\_path)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_0\_delicacy.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_10\_medicalTreatment.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_11\_carService.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_12\_trafficFacilities.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_13\_finance.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_14\_realEstate.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_15\_corporation.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_16\_government.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_1\_hotel.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_2\_shopping.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_3\_lifeService.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_4\_beauty.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_5\_spot.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_6\_entertainment.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_7\_sports.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_8\_education.csv  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
.csv to DataFrame completed!  
poi\_9\_media.csv

import matplotlib.pyplot as plt  
from mpl\_toolkits.axes\_grid1 import make\_axes\_locatable  
import geopandas as gpd  
from pylab import mpl  
  
mpl.rcParams['font.sans-serif']=['DengXian'] # 解决中文字符乱码问题  
  
xian\_region\_gdf=gpd.read\_file('./data/xian\_region/xian\_region.shp')  
  
fig, ax=plt.subplots(figsize=(15,20))  
xian\_region\_plot=xian\_region\_gdf.plot(ax=ax,color='white', edgecolor='black')  
# 标注  
xian\_region\_gdf.apply(lambda row: xian\_region\_plot.annotate(text=row.NAME,   
 xy=row.geometry.centroid.coords[0],   
 ha='center',  
 fontsize=13),  
 axis=1)   
divider=make\_axes\_locatable(ax)  
cax=divider.append\_axes("right", size="5%", pad=0.1)  
poisInAll\_gdf.loc[['poi\_0\_delicacy'],:].plot(ax=ax,  
 column='detail\_info\_overall\_rating',  
 markersize=10,  
 cmap='cividis',  
 legend=True,  
 cax=cax) # 提取index为'poi\_0\_delicacy'的行查看结果  
plt.show()



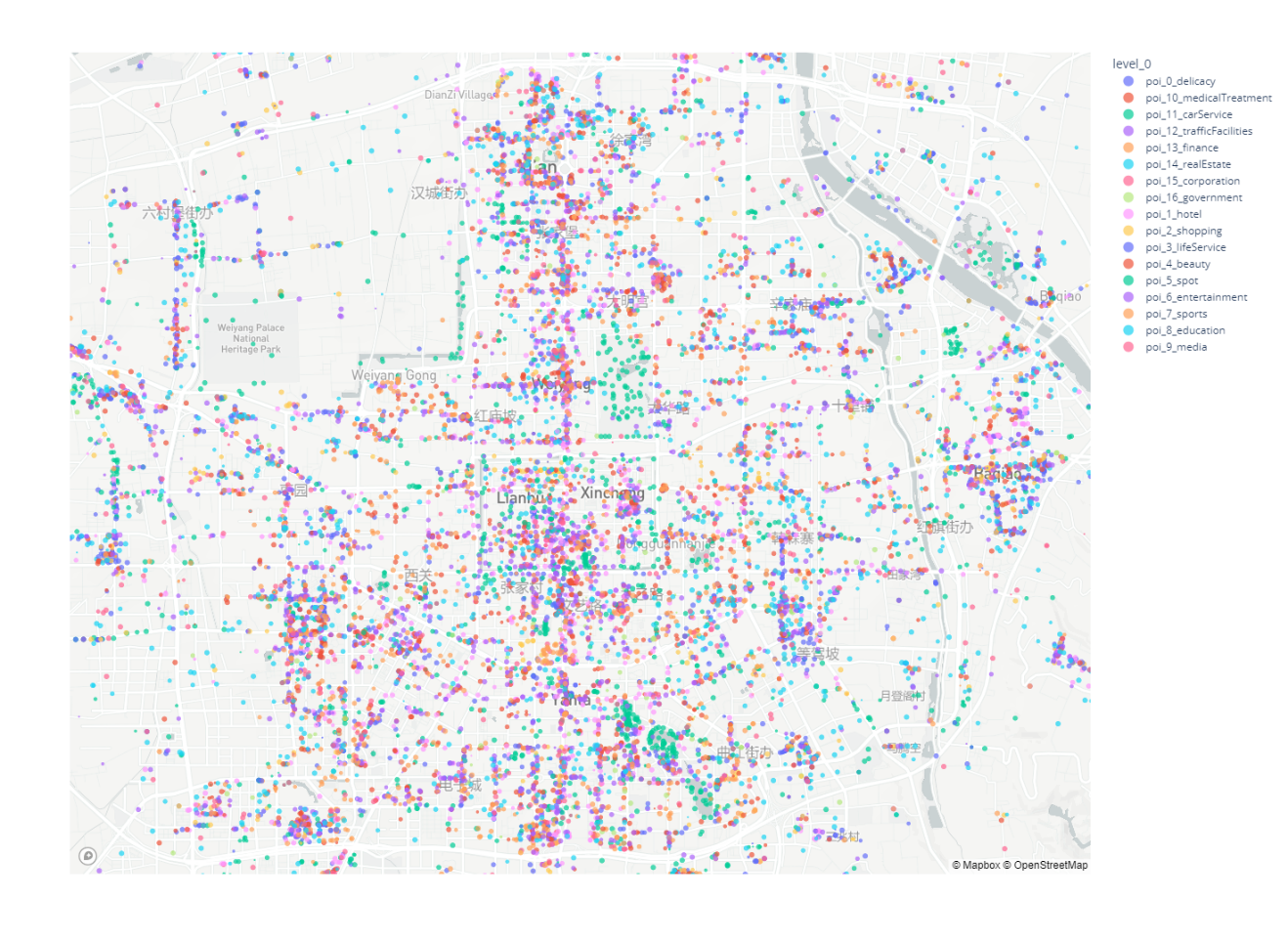
在[QGIS开源桌面GIS平台](https://www.qgis.org/en/site/index.html)下打开保存的.shp数据。虽然所有工作基本都是在Python中完成，但是有些工作与其它平台相互联系，需要以这些平台为辅助。在辅助平台选择上，尽可能使用具有广泛应用的开源软件。在地理信息系统（Geographic Information Systems，GIS）中主要使用的集成式平台有QGIS和ArcGIS。



#### 3）使用Plotly库建立地图

用颜色表示POI一级分类，用大小表示rating字段。

import geopandas as gpd  
import plotly.express as px  
  
poi\_gdf=gpd.read\_file('./data/poisInAll/poisInAll\_gdf.shp') # 读取存储的.shp格式文件  
poi\_gdf.rating=poi\_gdf.rating.fillna(0) # pandas库的方法同样适用于GeoPandas库，例如对`nan`位置填充指定数值  
mapbox\_token='pk.eyJ1IjoicmljaGllYmFvIiwiYSI6ImNrYjB3N2NyMzBlMG8yc254dTRzNnMyeHMifQ.QT7MdjQKs9Y6OtaJaJAn0A'  
  
px.set\_mapbox\_access\_token(mapbox\_token)  
fig=px.scatter\_mapbox(poi\_gdf,lat=poi\_gdf.lat, lon=poi\_gdf.lng,  
 color="level\_0",size='rating',  
 color\_continuous\_scale=px.colors.cyclical.IceFire, size\_max=5, zoom=10) # 亦可以选择列，通过size=""配置增加显示信息  
fig.show()



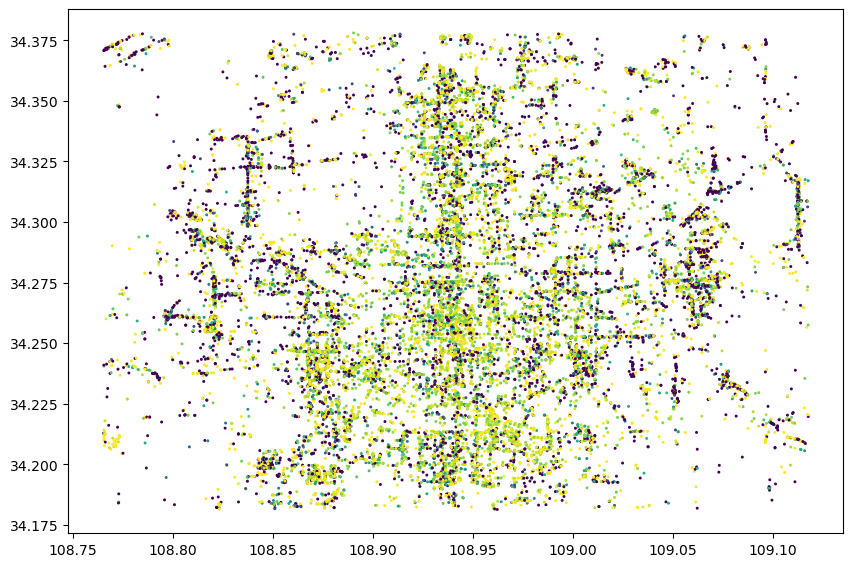
### 2.1.1.3 描述性统计图表

#### 1）读取与查看数据

* 读取已经保存的.pkl数据。通过.plot()确认读取的数据是否正常，或者直接使用poi\_gdf.head()查看数据。

import pandas as pd  
  
poi\_gdf=pd.read\_pickle('./data/poisInAll/poisInAll\_gdf.pkl')  
poi\_gdf.plot(marker=".",markersize=5,column='detail\_info\_overall\_rating',figsize=(10,15)) # 只有不设置columns参数时，可以使用color='green'参数  
print(poi\_gdf.columns) # 查看列名称

Index(['name', 'location\_lat', 'location\_lng', 'detail\_info\_tag',  
 'detail\_info\_overall\_rating', 'detail\_info\_price', 'geometry'],  
 dtype='object')



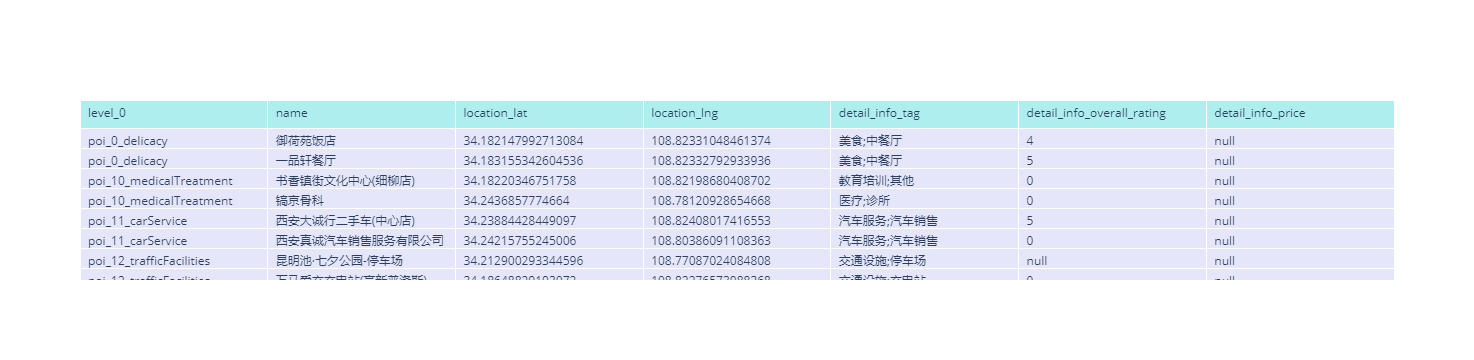
#### 2）用Plotly表格显示DataFrame格式数据

print()是查看数据最为主要的方式，主要用于代码调试。当需要展示数据时，对于DataFrame格式的数据可以使用Plotly转换为表格形式。因为POI数据有万行之多，仅显示每一级行业分类的前两行内容。为方便调用，将该功能定义为一个函数ployly\_table()。提取数据时，因为数据格式是多重索引的DataFrame，因此使用pandas.IndexSlice()函数辅助执行多重索引切分。因为用Plotly显示表格时，如果为多重索引会显示错误，因此需要df.reset\_index()重置索引。Plotly也不能够显示’geometry’几何对象，在列提取时需要移除该列。

如果在Spyder解释器中打印Plotly图表，通常要增加如下代码：

import plotly.io as pio  
pio.renderers.default='browser'

def plotly\_table(df,column\_extraction):  
 '''  
 funciton - 使用Plotly，以表格形式显示DataFrame格式数据  
   
 Params:  
 df - 输入的DataFrame或者GeoDataFrame；[Geo]DataFrame  
 column\_extraction - 提取的字段（列）；list(string)  
   
 Returns:  
 None  
 '''  
 import plotly.graph\_objects as go  
 import pandas as pd   
   
 fig=go.Figure(data=[go.Table(  
 header=dict(values=column\_extraction,  
 fill\_color='paleturquoise',  
 align='left'),  
 cells=dict(values=df[column\_extraction].values.T.tolist(), # values参数值为按列的嵌套列表，因此需要使用参数.T反转数组  
 fill\_color='lavender',  
 align='left'))  
 ])  
 fig.show()   
  
df=poi\_gdf.loc[pd.IndexSlice[:,:2],:]  
df=df.reset\_index()  
column\_extraction=['level\_0','name', 'location\_lat', 'location\_lng', 'detail\_info\_tag','detail\_info\_overall\_rating', 'detail\_info\_price']  
plotly\_table(df,column\_extraction)



#### 3） 描述性统计

本小节内容主要参考《漫画数据库》[1]、《漫画统计学》[2]及维基百科（Wikipedia）。将枯燥的知识以漫画的形式讲出来，并结合实际的案例由简入繁使枯燥的学习变得有趣起来。欧姆社学习漫画系列和众多以漫画和图示的方式讲解知识的优秀图书都值得推荐。但有利有弊，大部分的漫画图书往往以基础知识为主，深入的研究还是要搜索科学文献和相关论著。同时，漫画形式可以引起读者的兴趣，但是因为穿插故事情节，知识点不易定位，核心的知识内容相对分散，阅读上也要花费更多的时间，因此想学习一门知识，以哪种形式入手，需要根据个人的情况确定。

描述性统计分析是对调查总体所有变量的有关数据做统计性描述，了解各变量内的观察值集中与分散的情况。

* 表示集中趋势（集中量数）的有平均数、中位数、众数、几何平均数、调和平均数等；
* 表示离散程度（变异量数）的有极差（全距）、平均差、标准差、相对差、四分差等。数据的次数分配情况，往往会呈现正态分布；
* 数据的频数（次数）分配情况，往往会呈现正态分布。为了表示测量数据与正态分布偏离的情况，会使用偏度、峰度这两种统计数据；
* 为了解个别观察值在整体中所占的位置，会需要将观察值转换为相对量数，如百分等级、标准分数、四分位数等。

通常在描述性统计中将数据图表化，以直观的方式了解整体资料数据的分布情况，例如直方图、散点图、饼图、折现图、箱型图等。

##### 1. 数据种类

通常数据可以分为两类，不可测量的数据称为分类数据；可测量的数据称为数值数据。上述图表中’level\_0’、 ‘detail\_info\_tag’字段均为分类数据，’location\_lat’、‘location\_lng’、‘detail\_info\_overall\_rating’、’detail\_info\_price’均为数值数据，而’name’字段则是数据的索引名称。

##### 2. 数值数据的描述性统计

* 频数（次数）分布表和直方图

建立简单数据示例，数据来源于《漫画统计学》中“美味拉面畅销前50”上刊载的拉面馆的拉面价格。一般对于只有一组（一个特征、一个自变量/解释变量）的数据通常使用pandas.Series()建立Series格式数据，但是后续分析会加入新的数据，因此仍旧建立DataFrame格式数据。使用df.describe()可以初步查看主要统计值。

ramen\_price=pd.DataFrame([700,850,600,650,980,750,500,890,880,700,890,720,680,650,790,670,680,900,880,720,850,700,780,850,750,  
 780,590,650,580,750,800,550,750,700,600,800,800,880,790,790,780,600,690,680,650,890,930,650,777,700],columns=["price"])  
print(ramen\_price.describe())

price  
count 50.000000  
mean 743.340000  
std 108.261891  
min 500.000000  
25% 672.500000  
50% 750.000000  
75% 800.000000  
max 980.000000

因为有些价格是相同的，一般可以直接使用上述ramen\_price数据直接计算频数，但是很多时候相同的数据并不多，且希望分析内容为数值区段间的比较，分析才更具有意义，因此转换为相对量数，以100间隔为一级。范围则根据数据的最大和最小值来确定。

bins=range(500,1000+100,100) # 配置分割区间（组距）  
ramen\_price['price\_bins']=pd.cut(x=ramen\_price.price,bins=bins,right=False) # 参数right=False指定为包含左边值，不包括右边值  
ramenPrice\_bins=ramen\_price.sort\_values(by=['price']) # 按照分割区间排序  
ramenPrice\_bins.set\_index(['price\_bins',ramenPrice\_bins.index],drop=False, inplace=True) # 以price\_bins和原索引值设置多重索引，同时配置drop=False参数保留原列  
ramenPrice\_bins.head(10)

|  |  | **price** | **price\_bins** |
| --- | --- | --- | --- |
| **price\_bins** |  |  |  |
| **[500, 600)** | **6** | 500 | [500, 600) |
| **31** | 550 | [500, 600) |
| **28** | 580 | [500, 600) |
| **26** | 590 | [500, 600) |
| **[600, 700)** | **34** | 600 | [600, 700) |
| **41** | 600 | [600, 700) |
| **2** | 600 | [600, 700) |
| **44** | 650 | [600, 700) |
| **3** | 650 | [600, 700) |
| **27** | 650 | [600, 700) |

数值区段间的频数计算

ramenPriceBins\_frequency=ramenPrice\_bins.price\_bins.value\_counts()   
ramenPriceBins\_relativeFrequency=ramenPrice\_bins.price\_bins.value\_counts(normalize=True) # 参数normalize=True将计算相对频数(次数)  
ramenPriceBins\_freqANDrelFreq=pd.DataFrame({'fre':ramenPriceBins\_frequency,'relFre':ramenPriceBins\_relativeFrequency})  
ramenPriceBins\_freqANDrelFreq

|  | **fre** | **relFre** |
| --- | --- | --- |
| **[700, 800)** | 18 | 0.36 |
| **[600, 700)** | 13 | 0.26 |
| **[800, 900)** | 12 | 0.24 |
| **[500, 600)** | 4 | 0.08 |
| **[900, 1000)** | 3 | 0.06 |

组中值计算

ramenPriceBins\_median=ramenPrice\_bins.groupby(level=0).median(numeric\_only=True)  
ramenPriceBins\_median.rename(columns={'price':'median'},inplace=True)  
ramenPriceBins\_median

|  | **median** |
| --- | --- |
| **price\_bins** |  |
| **[500, 600)** | 565.0 |
| **[600, 700)** | 650.0 |
| **[700, 800)** | 750.0 |
| **[800, 900)** | 865.0 |
| **[900, 1000)** | 930.0 |

合并分割区间、频数计算和组中值的DataFrame格式数据。

ramen\_fre=ramenPriceBins\_freqANDrelFreq.join(ramenPriceBins\_median).sort\_index().reset\_index() # 在合并时会自动匹配index  
ramen\_fre

|  | **index** | **fre** | **relFre** | **median** |
| --- | --- | --- | --- | --- |
| **0** | [500, 600) | 4 | 0.08 | 565.0 |
| **1** | [600, 700) | 13 | 0.26 | 650.0 |
| **2** | [700, 800) | 18 | 0.36 | 750.0 |
| **3** | [800, 900) | 12 | 0.24 | 865.0 |
| **4** | [900, 1000) | 3 | 0.06 | 930.0 |

计算频数比例，即各个区间频数占总数的百分比，能够更清晰比较之间的差异大小。配合经常使用的df.apply()和lambda匿名函数，能够巧妙的以一种简洁的方式书写代码。

ramen\_fre['fre\_percent%']=ramen\_fre.apply(lambda row:row['fre']/ramen\_fre.fre.sum()\*100,axis=1)  
ramen\_fre

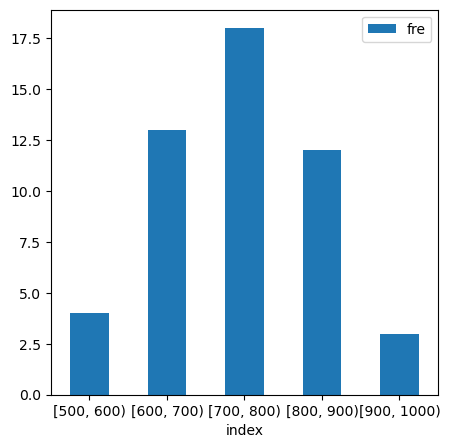
|  | **index** | **fre** | **relFre** | **median** | **fre\_percent%** |
| --- | --- | --- | --- | --- | --- |
| **0** | [500, 600) | 4 | 0.08 | 565.0 | 8.0 |
| **1** | [600, 700) | 13 | 0.26 | 650.0 | 26.0 |
| **2** | [700, 800) | 18 | 0.36 | 750.0 | 36.0 |
| **3** | [800, 900) | 12 | 0.24 | 865.0 | 24.0 |
| **4** | [900, 1000) | 3 | 0.06 | 930.0 | 6.0 |

直方图

Pandas自身就带有不少图表打印的功能（基于Matplotlib库），不必过多的调整数据自身结构，就可迅速的预览。但是不像Plotly具有图表交互功能。

ramen\_fre.loc[:,['fre','index']].plot.bar(x='index',rot=0,figsize=(5,5))

<AxesSubplot:xlabel='index'>



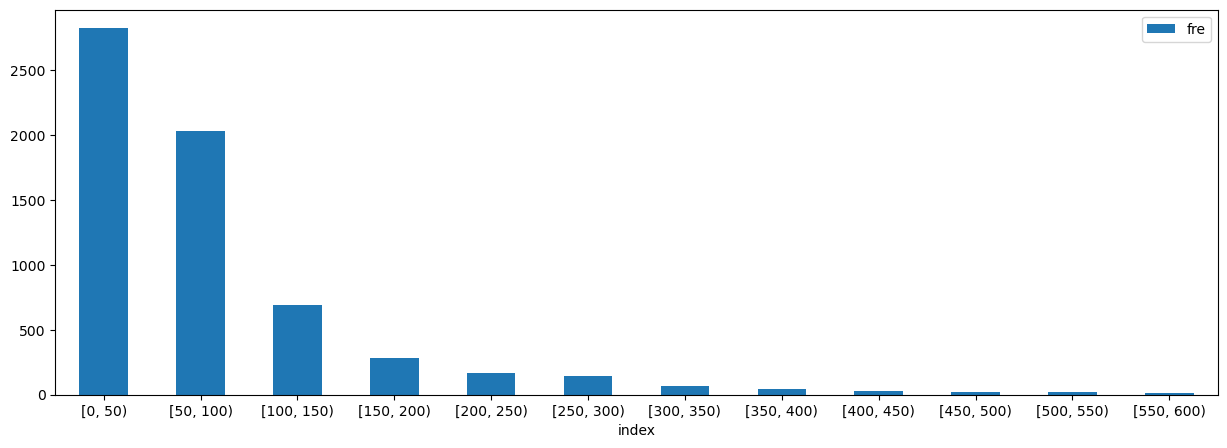
有了上述简单数据的示例，再返回到POI实验数据，可以直接迁移上述代码，略作调整后分析POI一级分类美食’poi\_0\_delicacy’的价格总体分布情况。将上述的所有分析代码置于一个函数中frequency\_bins()，函数的主要功能是计算DataFrame数据格式下，指定组距，一列数据的频数分布。将上述零散的代码纳入到一个函数中，需要注意几点事宜。一是，尽可能让变量名具有普适性，例如原变量名ramenPrice\_bins在函数中更改为df\_bins，因为该函数同样可以计算’detail\_info\_overall\_rating’字段的频数分布情况；二是，公用的常用变量尽量在开始配置，例如column\_name和column\_bins\_name，这样避免不断的用原始的语句，例如重复使用df.columns[0]+'\_bins'，从而导致代码可读性较差；再者，基本不可能直接迁移上述代码于函数中使用，需要逐句或者逐段的迁移测试，例如原代码ramenPrice\_bins.price\_bins，在函数中改为df\_bins[column\_bins\_name]，因为列名是以变量名形式存储，无法直接用.的方式读取数据。最后，需要注意函数返回值的灵活性，例如并未在函数内部定义图表打印，而是返回DataFrame格式的数据，因为打印的方式比较多样化，可以仅打印一列数据的柱状图，或者多列，及图表的形式也会多样化，因此这部分工作放置于函数外处理，增加灵活性。

def frequency\_bins(df,bins,field):   
 '''  
 function - 频数分布计算  
   
 Params:  
 df - 单列（数值类型）的DataFrame数据；DataFrame(Pandas)  
 bins - 配置分割区间（组距）；range()，例如：range(0,600+50,50)  
 field - 字段名；string  
   
 Returns:  
 df\_fre - 统计结果字段包含：index（为bins）、fre、relFre、median和fre\_percent%；DataFrame  
 '''  
 import pandas as pd  
   
 # A-组织数据  
 column\_name=df.columns[0]  
 column\_bins\_name=df.columns[0]+'\_bins'  
 df[column\_bins\_name]=pd.cut(x=df[column\_name],bins=bins,right=False) # 参数right=False指定为包含左边值，不包括右边值。  
 df\_bins=df.sort\_values(by=[column\_name]) # 按照分割区间排序  
 df\_bins.set\_index([column\_bins\_name,df\_bins.index],drop=False,inplace=True) # 以price\_bins和原索引值设置多重索引，同时配置drop=False参数保留原列。  
   
 # B-频数计算  
 dfBins\_frequency=df\_bins[column\_bins\_name].value\_counts() # dropna=False   
 dfBins\_relativeFrequency=df\_bins[column\_bins\_name].value\_counts(normalize=True) # 参数normalize=True将计算相对频数(次数) dividing all values by the sum of values  
 dfBins\_freqANDrelFreq=pd.DataFrame({'fre':dfBins\_frequency,'relFre':dfBins\_relativeFrequency})  
   
 # C-组中值计算  
 df\_bins[field]=df\_bins[field].astype(float)  
 dfBins\_median=df\_bins.groupby(level=0).median(numeric\_only=True)  
 dfBins\_median.rename(columns={column\_name:'median'},inplace=True)  
   
 # D-合并分割区间、频数计算和组中值的DataFrame格式数据。  
 df\_fre=dfBins\_freqANDrelFreq.join(dfBins\_median).sort\_index().reset\_index() # 在合并时会自动匹配index  
   
 # E-计算频数比例  
 df\_fre['fre\_percent%']=df\_fre.apply(lambda row:row['fre']/df\_fre.fre.sum()\*100,axis=1)  
   
 return df\_fre  
  
pd.options.mode.chained\_assignment=None  
bins=range(0,600+50,50) # 配置分割区间（组距）   
delicacy\_price\_df=poi\_gdf[["detail\_info\_price"]]  
delicacy\_price\_df["detail\_info\_price"]=delicacy\_price\_df["detail\_info\_price"].apply(pd.to\_numeric,errors='coerce') # 将str(object)数据类型转换为数值类型  
poiPrice\_fre\_50=frequency\_bins(delicacy\_price\_df.dropna(),bins,"detail\_info\_price")   
poiPrice\_fre\_50

|  | **index** | **fre** | **relFre** | **median** | **fre\_percent%** |
| --- | --- | --- | --- | --- | --- |
| **0** | [0, 50) | 2824 | 0.445918 | 20.0 | 44.591821 |
| **1** | [50, 100) | 2030 | 0.320543 | 73.0 | 32.054319 |
| **2** | [100, 150) | 689 | 0.108795 | 115.0 | 10.879520 |
| **3** | [150, 200) | 286 | 0.045160 | 170.0 | 4.516027 |
| **4** | [200, 250) | 164 | 0.025896 | 223.0 | 2.589610 |
| **5** | [250, 300) | 145 | 0.022896 | 268.0 | 2.289594 |
| **6** | [300, 350) | 71 | 0.011211 | 321.0 | 1.121112 |
| **7** | [350, 400) | 45 | 0.007106 | 373.0 | 0.710564 |
| **8** | [400, 450) | 25 | 0.003948 | 423.0 | 0.394758 |
| **9** | [450, 500) | 21 | 0.003316 | 475.0 | 0.331596 |
| **10** | [500, 550) | 21 | 0.003316 | 522.0 | 0.331596 |
| **11** | [550, 600) | 12 | 0.001895 | 573.5 | 0.189484 |

poiPrice\_fre\_50.loc[:,['fre','index']].plot.bar(x='index',rot=0,figsize=(15,5))

<AxesSubplot:xlabel='index'>



调整组距，查看总体频数分布情况。

import matplotlib.pyplot as plt  
  
bins=list(range(0,300+5,5))+[600] # 通过df.describe()查看数据后，发现72%的价格位于72元之下，结合上图柱状图，重新配置组距，尽可能的显示数据变化的趋势。  
poiPrice\_fre\_5=frequency\_bins(delicacy\_price\_df,bins,"detail\_info\_price")   
poiPrice\_fre\_5.loc[:,['fre','index']].plot.bar(x='index',rot=0,figsize=(30,5))  
\_=plt.xticks(rotation=90)

* 集中量数与变异量数

建立简单数据示例，数据来源于《漫画统计学》保龄球大赛的结果。首先建立嵌套字典，然后将其转换为多重索引的DataFrame数据。

bowlingContest\_scores\_dic={'A\_team':{'Barney':86,'Harold':73,'Chris':124,'Neil':111,'Tony':90,'Simon':38},  
 'B\_team':{'Jo':84,'Dina':71,'Graham':103,'Joe':85,'Alan':90,'Billy':89},  
 'C\_team':{'Gordon':229,'Wade':77,'Cliff':59,'Arthur':95,'David':70,'Charles':88}  
 }  
  
# 可以逐步拆解来查看每一步的数据结构，结合搜索相关方法解释，可以理解每一步的作用。 例如df.stack()是由列返回多重索引的DataFrame,具体解释可以查看官方案例  
bowlingContest\_scores=pd.DataFrame.from\_dict(bowlingContest\_scores\_dic, orient='index').stack().to\_frame(name='score')   
# 使用print()或者直接在每一JupyterLab的Cell最后给出要查看的变量名，都可以查看数据，只是可能显示的模式略有差异。但建议使用print()查看，因为代码迁移时，单独变量的出现可能会造成代码运行错误  
bowlingContest\_scores

|  |  | **score** |
| --- | --- | --- |
| **A\_team** | **Barney** | 86.0 |
| **Harold** | 73.0 |
| **Chris** | 124.0 |
| **Neil** | 111.0 |
| **Tony** | 90.0 |
| **Simon** | 38.0 |
| **B\_team** | **Jo** | 84.0 |
| **Dina** | 71.0 |
| **Graham** | 103.0 |
| **Joe** | 85.0 |
| **Alan** | 90.0 |
| **Billy** | 89.0 |
| **C\_team** | **Gordon** | 229.0 |
| **Wade** | 77.0 |
| **Cliff** | 59.0 |
| **Arthur** | 95.0 |
| **David** | 70.0 |
| **Charles** | 88.0 |

求每一队的均值（算数平均数）

bowlingContest\_mean=bowlingContest\_scores.groupby(level=0).mean()  
bowlingContest\_mean

|  | **score** |
| --- | --- |
| **A\_team** | 87.0 |
| **B\_team** | 87.0 |
| **C\_team** | 103.0 |

求每一队的中位数。C队的均值最高，其原因不是每一个队员的成绩都高，而是Gordon获得了229远超其他队员的得分。因此求中位数更为适合。

bowlingContest\_median=bowlingContest\_scores.groupby(level=0).median()  
bowlingContest\_median

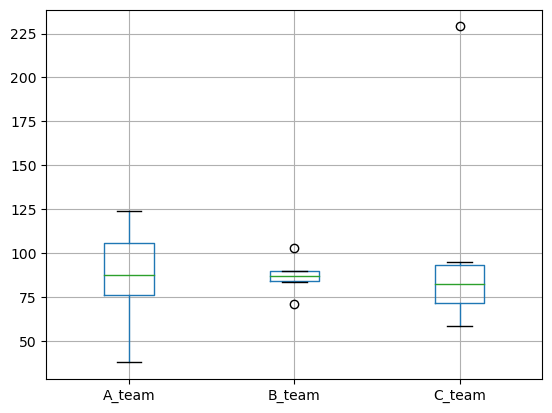
|  | **score** |
| --- | --- |
| **A\_team** | 88.0 |
| **B\_team** | 87.0 |
| **C\_team** | 82.5 |

箱型图（Box plot），又称盒须图，盒式（状）图或箱线图，一种用作显示一组数据分散情况的统计图。显示的一组数据包括最大值、最小值、中位数和上下四分位数，因此使用箱型图较之单一的数值而言可以更清晰的观察数据分布情况。如图（引自：Wikipedia）： +—–+-+  
\* o |——-| + | |—|

+—–+-+

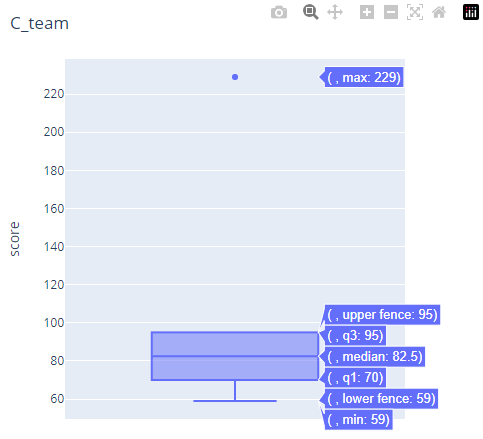
+—+—+—+—+—+—+—+—+—+—+ 分数 0 1 2 3 4 5 6 7 8 9 10 这组数据显示：最小值（minimum）=5；下四分位数（Q1）=7；中位数（Med，即Q2）=8.5；上四分位数（Q3）=9；最大值（maximum）=10；平均值=8；四分位间距(interquartile range)=(Q3-Q2)=2（即ΔQ）。使用Pandas自带的plot功能打印箱型图，查看各队分数的分布情况。

bowlingContest\_scores\_transpose=bowlingContest\_scores.stack().unstack(level=0)  
boxplot=bowlingContest\_scores\_transpose.boxplot(column=['A\_team', 'B\_team', 'C\_team'])



Plotly库提供的箱型图可以互动显示具体的数值，具有相对更强的图示能力。

import plotly.express as px  
  
fig=px.box(bowlingContest\_scores.xs('C\_team',level=0), y="score")  
fig.update\_layout(  
 autosize=False,  
 width=500,  
 height=500,  
 title='C\_team'  
 )  
fig.show()



求标准差，又称准偏差或均方差（Standard Deviation，SD），数学符号通常用σ（sigma），用于测量一组数值的离散程度，公式：$SD= $ 其中 为平均值。虽然A\_team和B\_team具有相同的均值，但是数值的分布情况迥然不同，通过计算标准差比较离散程度，标准差越小，代表数据的离散程度越小；反之，标准差越大，离散程度越大。

上述公式为整体标准差，本书的实验数据通常为全部数据样本，而不是抽样，因此使用整体标准差。计算样本的标准差，公式为：

bowlingContest\_std=bowlingContest\_scores.groupby(level=0).std()  
bowlingContest\_std

|  | **score** |
| --- | --- |
| **A\_team** | 30.172835 |
| **B\_team** | 10.373042 |
| **C\_team** | 63.033325 |

返回到POI实验数据，因为一级行业分类中美食部分的’detial\_info\_tag’包含餐厅的子分类，使用箱型图显示子分类评分’detail\_info\_overall\_rating’的数值分布情况，并计算标准差。

delicacy=poi\_gdf.xs('poi\_0\_delicacy',level=0)  
delicacy\_rating=delicacy[['detail\_info\_tag','detail\_info\_overall\_rating']]   
delicacy\_rating

|  | **detail\_info\_tag** | **detail\_info\_overall\_rating** |
| --- | --- | --- |
| **0** | 美食;中餐厅 | 4.0 |
| **2** | 美食;中餐厅 | 5.0 |
| **4** | 美食;中餐厅 | 5.0 |
| **6** | 美食;中餐厅 | 4.5 |
| **8** | 美食;中餐厅 | 4.6 |
| **...** | ... | ... |
| **6590** | 美食;小吃快餐店 | NaN |
| **6592** | 美食;小吃快餐店 | NaN |
| **6594** | 美食;小吃快餐店 | NaN |
| **6596** | 美食;小吃快餐店 | NaN |
| **6598** | 美食;茶座 | NaN |

3300 rows × 2 columns

查看餐厅类型。并移除错误的分类数据，例如'教育培训;其他'。同时可以调整子分类的名称，例如由’美食;中餐厅’ 修改为’中餐厅’，其中使用了df.applay()方法。最后将其映射为英文字符，在打印时也可以避免显示错误。如果显示中文字符错误，需要增加相应处理代码。

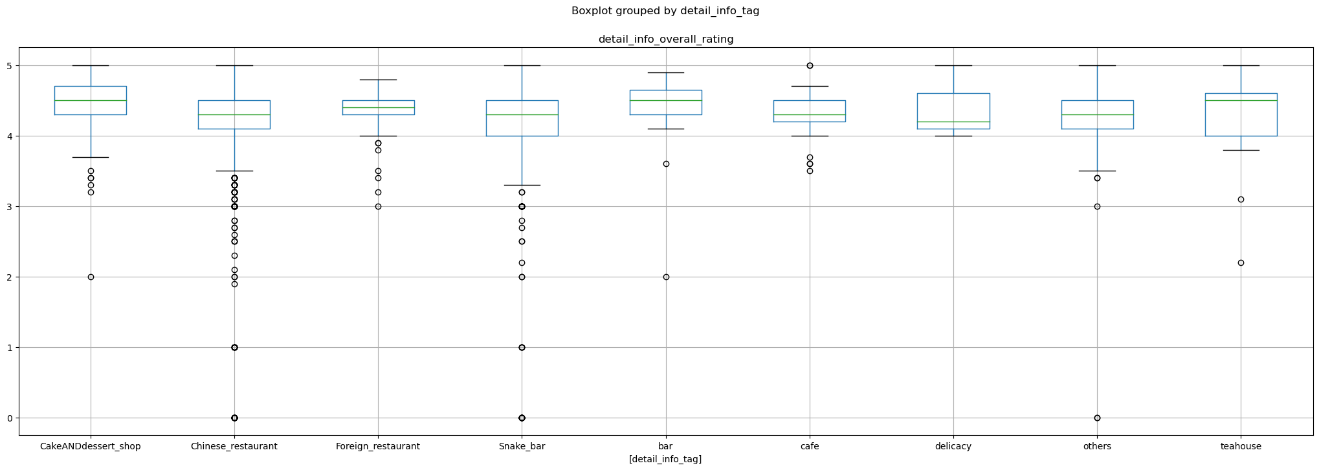
pd.options.mode.chained\_assignment=None方法为Pandas的警告异常处理，选项包括：None，忽略警告；warn，打印警告消息；raise，引发异常。

pd.options.mode.chained\_assignment = None  
print(delicacy\_rating.detail\_info\_tag.unique())  
print("\_"\*50)  
delicacy\_rating["clean\_bool"]=delicacy\_rating.detail\_info\_tag.apply(lambda row:row.split(";")[0]=="美食" if isinstance(row,str) else False)  
delicacy\_rating\_clean=delicacy\_rating[delicacy\_rating.clean\_bool].drop(columns=["clean\_bool"])   
print(delicacy\_rating\_clean.detail\_info\_tag.unique())  
print("\_"\*50)  
  
# 定义一个函数，传入df.apply()函数处理字符串  
def str\_row(row):  
 if type(row)==str:  
 row\_=row.split(';')[-1]  
 else:  
 row\_='nan' # 原数据类型为nan，通过type(row)查看后为float数据类型，此时将其转换为字符串  
 return row\_  
delicacy\_rating\_clean.loc[:,["detail\_info\_tag"]]=delicacy\_rating\_clean["detail\_info\_tag"].apply(str\_row)   
print(delicacy\_rating\_clean.detail\_info\_tag.unique())  
print("\_"\*50)  
  
tag\_mapping={'中餐厅':'Chinese\_restaurant',  
 '小吃快餐店':'Snake\_bar',  
 'nan':'nan',  
 '其他':'others',  
 '外国餐厅':'Foreign\_restaurant',  
 '蛋糕甜品店':'CakeANDdessert\_shop',  
 '咖啡厅':'cafe',  
 '茶座':'teahouse',  
 '酒吧':'bar',  
 '美食':'delicacy',  
 '公司':'company',  
 '商铺':'store',  
 '洗浴按摩':'massage',  
 '超市':'supermarket',  
 '快捷酒店':'budgetHotel',  
 '园区':'Park'}  
delicacy\_rating\_clean.loc[:,["detail\_info\_tag"]]=delicacy\_rating\_clean["detail\_info\_tag"].replace(tag\_mapping)  
print(delicacy\_rating\_clean.detail\_info\_tag.unique())

['美食;中餐厅' '美食;其他' '美食;小吃快餐店' '美食;咖啡厅' '美食;蛋糕甜品店' '美食;茶座' '房地产;其他'  
 '美食;外国餐厅' '美食' '公司企业;公司' '购物;商铺' nan '休闲娱乐;洗浴按摩' '美食;酒吧' '交通设施;其他'  
 '公司企业;其他' '购物;超市' '酒店;快捷酒店' '教育培训;其他' '酒店;其他' '公司企业;园区']  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
['美食;中餐厅' '美食;其他' '美食;小吃快餐店' '美食;咖啡厅' '美食;蛋糕甜品店' '美食;茶座' '美食;外国餐厅' '美食'  
 '美食;酒吧']  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
['中餐厅' '其他' '小吃快餐店' '咖啡厅' '蛋糕甜品店' '茶座' '外国餐厅' '美食' '酒吧']  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
['Chinese\_restaurant' 'others' 'Snake\_bar' 'cafe' 'CakeANDdessert\_shop'  
 'teahouse' 'Foreign\_restaurant' 'delicacy' 'bar']

delicacy\_rating\_clean.boxplot(column=['detail\_info\_overall\_rating'],by=['detail\_info\_tag'],figsize=(25,8))  
delicacy\_rating\_clean\_std=delicacy\_rating\_clean.set\_index(['detail\_info\_tag']).groupby(level=0).std()  
delicacy\_rating\_clean\_std

|  | **detail\_info\_overall\_rating** |
| --- | --- |
| **detail\_info\_tag** |  |
| **CakeANDdessert\_shop** | 0.375599 |
| **Chinese\_restaurant** | 0.703622 |
| **Foreign\_restaurant** | 0.265208 |
| **Snake\_bar** | 0.826895 |
| **bar** | 0.589164 |
| **cafe** | 0.273044 |
| **delicacy** | 0.529150 |
| **others** | 0.751135 |
| **teahouse** | 0.678449 |



* 标准计分（分数）

建立简单数据示例，数据来源于《漫画统计学》中的考试成绩数据。在这个案例中，虽然Mason和Reece分别在English和Chinese科目中，而history和biology科目中具有相同的分数，但是因为标准差，即离散程度不同，所表示的重要程度亦不一样。标准差越小，离散程度越小，则数值每一单位的变化都会影响最终排名，每一分都很重要。也可以理解为标准差小时，落后的同学相对容易追上成绩在前的同学；但是标准差大时，落后的同学相对不容易追上排名靠前的同学。

test\_score\_dic={"English":{"Mason":90,"Reece":81,'A':73,'B':97,'C':85,'D':60,'E':74,'F':64,'G':72,'H':67,'I':87,'J':78,'K':85,'L':96,'M':77,'N':100,'O':92,'P':86},  
 "Chinese":{"Mason":71,"Reece":90,'A':79,'B':70,'C':67,'D':66,'E':60,'F':83,'G':57,'H':85,'I':93,'J':89,'K':78,'L':74,'M':65,'N':78,'O':53,'P':80},  
 "history":{"Mason":73,"Reece":61,'A':74,'B':47,'C':49,'D':87,'E':69,'F':65,'G':36,'H':7,'I':53,'J':100,'K':57,'L':45,'M':56,'N':34,'O':37,'P':70},  
 "biology":{"Mason":59,"Reece":73,'A':47,'B':38,'C':63,'D':56,'E':75,'F':53,'G':80,'H':50,'I':41,'J':62,'K':44,'L':26,'M':91,'N':35,'O':53,'P':68},  
 }  
  
test\_score=pd.DataFrame.from\_dict(test\_score\_dic)  
test\_score.head()

|  | **English** | **Chinese** | **history** | **biology** |
| --- | --- | --- | --- | --- |
| **Mason** | 90 | 71 | 73 | 59 |
| **Reece** | 81 | 90 | 61 | 73 |
| **A** | 73 | 79 | 74 | 47 |
| **B** | 97 | 70 | 47 | 38 |
| **C** | 85 | 67 | 49 | 63 |

求标准计分（Standard Score），又称z-score，即Z-分数，或标准化值。z-score代表原始数值和平均值之间的距离，并以标准差为单位计算，即z-score是从感兴趣的点到均值之间有多少个标准差，这样就可以在不同组数据间比较某一数值的重要程度。公式为：$z=(x- )/ $ ；并是需要被标准化的原始分数；是平均值；是标准差。

标准计分的特征：

1. 无论作为变量的取值范围是多少（例如，科目满分为100还是150分），其标准计分的平均数势必为0， 而其标准差势必为1；
2. 无论作为变量的单位是什么（例如分、公斤和小时等），其标准计分的平均数势必为0， 而其标准差势必为1。

from scipy.stats import zscore  
  
test\_Zscore=test\_score.apply(zscore)  
test\_Zscore.head()

|  | **English** | **Chinese** | **history** | **biology** |
| --- | --- | --- | --- | --- |
| **Mason** | 0.770054 | -0.296174 | 0.780635 | 0.162355 |
| **Reece** | -0.029617 | 1.392020 | 0.207107 | 1.014719 |
| **A** | -0.740436 | 0.414644 | 0.828429 | -0.568242 |
| **B** | 1.392020 | -0.385027 | -0.462008 | -1.116191 |
| **C** | 0.325792 | -0.651584 | -0.366420 | 0.405887 |

其中Mason在English科目中的标准计分为0.77，在整体分布中位于平均分之上0.71个标准差的地位；而Reece在Chinese中的标准计分为1.39，在整体分布中位于平均分之上1.39个标准差的地位，即Reece获得的每一分值价值高于Mason所获取的每一分值。

返回到POI实验数据，分别计算美食部分总体评分’detail\_info\_overall\_rating’和价格 ’detail\_info\_price’的标准计分。

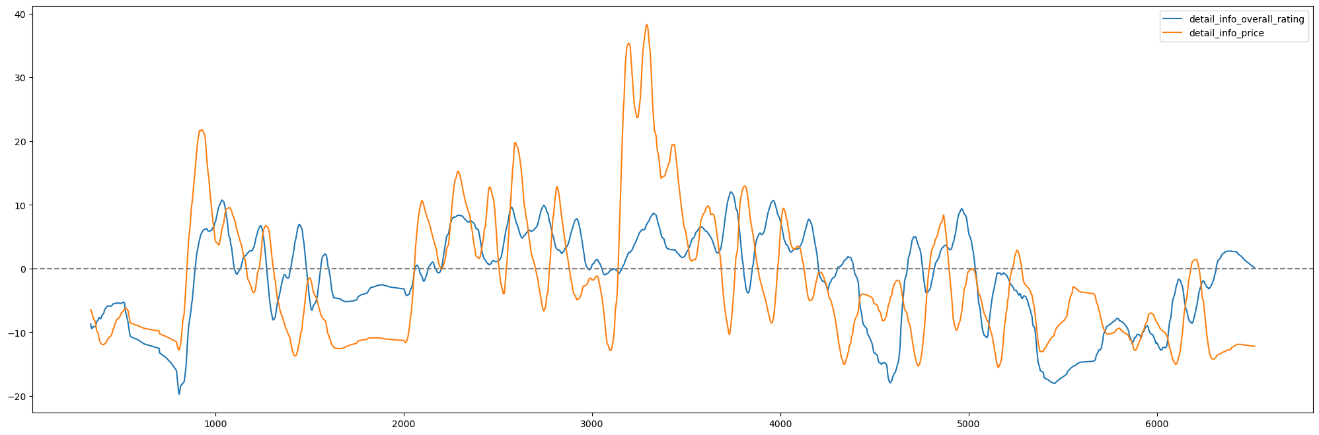
pd.options.mode.chained\_assignment=None  
delicacy=poi\_gdf.xs('poi\_0\_delicacy',level=0)  
delicacy\_dropna=delicacy.dropna(subset=['detail\_info\_overall\_rating', 'detail\_info\_price'])  
delicacy\_dropna[['detail\_info\_overall\_rating', 'detail\_info\_price']]=delicacy\_dropna[['detail\_info\_overall\_rating', 'detail\_info\_price']].astype(float)  
delicacy\_Zscore=delicacy\_dropna[['detail\_info\_overall\_rating', 'detail\_info\_price']].apply(zscore).join(delicacy["name"])  
delicacy\_Zscore

|  | **detail\_info\_overall\_rating** | **detail\_info\_price** | **name** |
| --- | --- | --- | --- |
| **6** | 0.500851 | -0.903030 | 关中印象咥长安(创汇店) |
| **10** | -0.597102 | -0.493965 | 哪儿托海鲜焖面 |
| **22** | 1.598804 | -0.493965 | 赛百味(昆明池店) |
| **26** | 0.720441 | -0.396569 | 太平洋咖啡(昆明池店) |
| **28** | -1.255874 | -0.260214 | 文丰厨8 |
| **...** | ... | ... | ... |
| **6432** | -0.597102 | -0.347870 | 佳美蛋糕(庆华店) |
| **6448** | 1.598804 | -0.786154 | 袁师傅腊汁肉夹馍 |
| **6462** | -7.184819 | -0.864071 | 千里香馄饨王(灞桥堡路店) |
| **6490** | -0.377511 | 0.051455 | 窑村猪蹄坊总店(纺渭路店) |
| **6518** | 0.500851 | -0.727716 | 食膳坊中式快餐 |

2286 rows × 3 columns

通过计算z\_score，比较某一饭店的价格和评分的重要性，其意义是，是否价格越接近均值，对应的评分越高于（低于）均值。但是单独看单一饭店的数据很难判断是否存在这样的一种关系，因此可以打印曲线，观察曲线的变化规律。使用df.rolling()方法平滑数据后绘制曲线观察数据。

import matplotlib.pyplot as plt  
  
delicacy\_Zscore\_selection=delicacy\_Zscore.select\_dtypes(include="number") # 通过.select\_dtypes()方法，提取数值列  
delicacy\_Zscore\_selection.rolling(50, win\_type='triang').sum().plot.line(figsize=(25,8))  
plt.axhline(y=0,color='gray',linestyle='--')  
plt.show()



### 2.1.1.4 正态分布与概率密度函数，异常值处理

#### 1）正态分布

在开始概率密度函数之前，认识下正态分布，又称常态分布、高斯分布（normal distribution/Gaussian distribution），因为正态分布的形状，正态分布常见的名称为钟形曲线（bell curve）。用NumPy库中的numpy.random.normal()方法生成满足指定平均值、和标准差的一维数据（也可以用SciPy库stats类中提供的norm方法），并打印直方图和对应由概率密度函数计算的分布曲线。下图y轴表示随机生成数值的频数（因为配置了density=Ture，返回的频数为标准化后的结果）；x轴为随机生成的数据集数值分布，因为设置了plt.hist()中bins=30，将数值分为30等分，计算每一等分（频数宽度）的频数。曲线的顶点为均值、中位数，频数为最高，对应的值为0，该值也为众数。可以从新定义平均数mu参数的值，曲线顶点也会随之变动。从众数向两侧移动，曲线高度下降，表示这些值出现的情形逐渐减少，频数降低。在统计学上下图可以文字描述为：x服从平均值为0，标准差为0.1的正态分布。

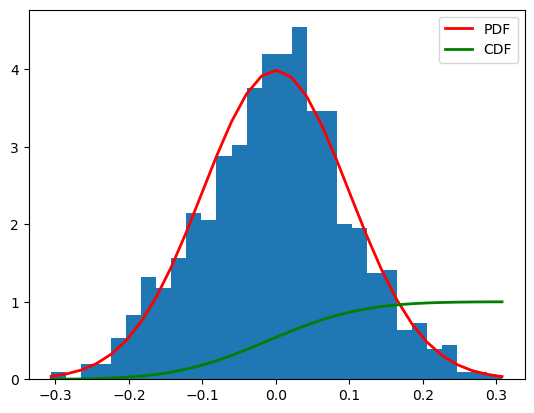
依据正态分布的形状，其三个基本性质为：一，它是对称的，意味左右部分互为镜像；二，均值、中位数和众数处于同一位置，且在分布的中心，即钟形的顶点。表现为曲线中心最高，首尾两端向下倾斜，呈单峰状；三是，正态分布是渐近的，意味分布的左尾和右尾永远不会触及底线，即x轴。

正态分布具有重要意义，自然界与人类社会中经常出现正态分布的各类数据，例如经济中的收入水平，人的智商（IQ）分数和考试成绩，自然界中受大量微小随机扰动影响的事物等。因此能够依据这一现象推断出现某种情形的准确概率。同时，需要注意，正态分布是统计学中所谓的理论分布，很少有数值严格服从正态分布，而是近似于，也有可能相差较远。违背正态分布假设，则依据正态分布假设所计算的概率结果将不再有效[3]。

对于统计学的知识，通常是依据已出版的专著或教材为依据进行解释，并根据阐述的内容，以Python语言为工具，分析数据的变化，从而可以直接应用代码来解决对应的问题，并能够通过具体的数据分析更深入的理解统计学相关的知识。

# 下述案例参考SciPy.org 中`numpy.random.normal`案例  
import math  
import numpy as np  
import matplotlib.pyplot as plt  
  
# 依据分布配置参数生成数据 Draw samples from the distribution  
mu, sigma=0, 0.1 # 配置平均值和标准差 mean and standard deviation  
s=np.random.normal(mu, sigma, 1000)  
# 验证生成的数据是否满足配置参数 Verify the mean and the variance  
print("mean<0.01:",abs(mu-np.mean(s)) < 0.01)  
print("sigmag variance<0.01",abs(sigma - np.std(s, ddof=1)) < 0.01)  
  
# 参数density=True；If True, the first element of the return tuple will be the counts normalized to form a probability density  
count, bins, ignored=plt.hist(s,bins=30,density=True)   
PDF=1/(sigma \* np.sqrt(2 \* np.pi)) \*np.exp( - (bins - mu)\*\*2 / (2 \* sigma\*\*2) ) # y方向的值计算公式即为概率密度函数  
plt.plot(bins,PDF ,linewidth=2, color='r',label='PDF')   
  
CDF=PDF.cumsum() # 计算累积分布函数  
CDF/=CDF[-1] # 通过除以最大值，将CDF数值分布调整在[0,1]的区间  
plt.plot(bins,CDF,linewidth=2, color='g',label='CDF')   
  
plt.legend()  
plt.show()

mean<0.01: True  
sigmag variance<0.01 True



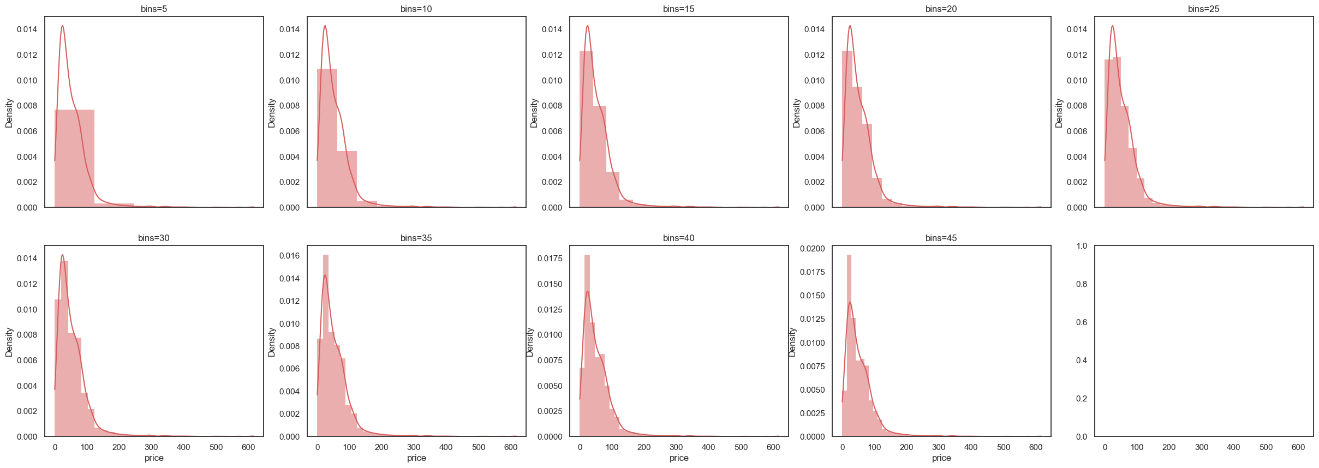
#### 2）概率密度函数（Probability density function，PDF）

当直方图的组距无限缩小至极限后，能够拟合出一条曲线，计算这个分布曲线的公式为概率密度函数： 式中，为标准差；为平均值；为自然对数的底，其值大约为2.7182…。在上述程序中，计算概率密度函数时，并未计算每个数值，而是使用plt.hist()返回值bins替代，为每一组距的左边沿和右边沿，这里划分了30份，因此首位数为第1个频数宽度的左边沿，末位数为最后一个频数宽度的右边沿，而中间的所有为左右边沿重叠。概率密度函数的积分，为累积分布函数（cumulative distribution function，CDF），可以用numpy.cumsum()计算，为给定轴上数组元素的累积和。

图表打印的库主要包括[Matplotlib](https://matplotlib.org/)，[Plotly（含dash）](https://plotly.com/)，[Bokeh](https://docs.bokeh.org/en/latest/index.html)，[seaborn](https://seaborn.pydata.org/)等⑨，具体选择哪个打印图表，没有固定的标准，通常根据数据图表打印的目的、由哪个库能够满足要求，及个人习惯用哪个库来确定。在上述使用Matplotlib库打印概率密度函数曲线时，是自行计算，而Seaborn提供了seaborn.histplot()方法，指定bins参数后可以直接获取上述结果。bins参数的配置，如果为一个整数值，则是划分同宽度频数宽度（bin）的数量，如果是列表，则为频数宽度的边缘，例如[1,2,3,4]，表示[[1,2),[2,3),[3,4]]的频数宽度列表，[代表包含左边沿数据，]代表包含右边沿数据，而(，和)则是分别代表不包含左或右边沿数据。同Pandas的pandas.core.indexes.range.RangeIndex的RangeIndex数据格式。

返回到POI实验数据，直接计算打印POI美食数据的价格概率密度函数的值（纵轴），并连为曲线（分布曲线）。为了进一步观察，组距不断缩小时，与曲线的拟合程度，定义一个循环打印多个连续组距变化的直方图，来观察直方图的变化情况。

import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
import math  
  
poi\_gdf=pd.read\_pickle('./data/poisInAll/poisInAll\_gdf.pkl') # 读取已经存储为.pkl格式的POI数据，其中包括geometry字段，为GeoDataFrame地理信息数据，可以通过poi\_gpd.plot()迅速查看数据。  
delicacy\_price=poi\_gdf.xs('poi\_0\_delicacy',level=0).detail\_info\_price # 提取美食价格数据  
delicacy\_price\_df=delicacy\_price.to\_frame(name='price').astype(float)  
  
sns.set(style="white", palette="muted", color\_codes=True)  
bins=list(range(0,50,5))[1:]  
bin\_num=len(bins)  
  
ncol=5  
nrow=math.ceil(bin\_num/ncol)  
fig, axs = plt.subplots(ncols=ncol,nrows=nrow,figsize=(30,10),sharex=True)  
ax=[(row,col) for row in range(nrow) for col in range(ncol)]  
i=0  
for bin in bins:  
 sns.histplot(delicacy\_price\_df.price, bins=bin, color="r",ax=axs[ax[i]],kde=True, stat="density", linewidth=0).set(title="bins=%d"%bin)   
 i+=1



#### 3）偏度与峰度

描述正态分布的特征（或概率密度函数的分布曲线）有两个概念，一个是偏度（skew），另一个是峰度（kurtosis）。

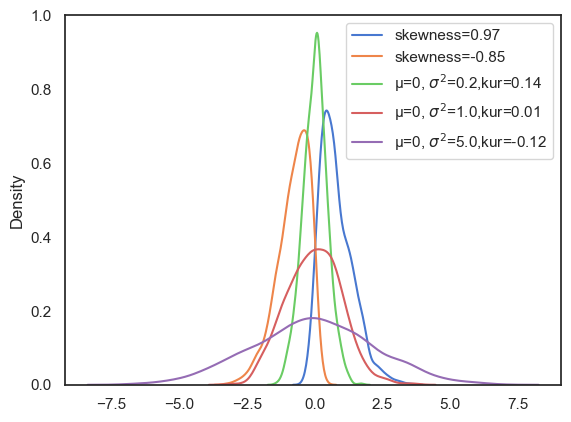
由SciPy库skewnorm方法建立具有正偏态和负偏态属性的数据，并由skew方法计算偏度值，其公式为：$skewness= $ ，式中为均值；为标准差。偏度值为负，即负偏态，则概率密度函数左侧尾部比右侧长；偏度值为正，即正偏态，概率密度函数右侧尾部比左侧长。

由NumPy建立具有尖峰分布（瘦尾）和扁峰分布（厚尾）属性的数据，配置绝对值参数均为0，变化标准差，使用SciPy库kurtosis方法计算峰度值（峰度值计算有多个公式，不同软件平台公式也会有所差异）。当平均值为0，标准差为1时的正态分布为标准正态分布，概率密度函数公式可以简化为：

在用NumPy生成数据集时，间接使用了方差$ ^{2} f(x | , ^{2} )= e^{- } $, 为方差： $^{2} = *{i=1}^N { ( x*{i}- ) ^{2} } $

import math  
import numpy as np  
from scipy.stats import kurtosis  
from scipy.stats import skewnorm  
from scipy.stats import skew  
import matplotlib.pyplot as plt  
  
# 建立具有正偏态和负偏态属性的数据  
skew\_list=[7,-7]  
skewNorm\_list=[skewnorm.rvs(a, scale=1,size=1000,random\_state=None) for a in skew\_list]  
skewness=[skew(d) for d in skewNorm\_list]  
print('skewness for data:',skewness) # 验证偏度值  
  
# 建立具有尖峰分布、和扁峰分布属性的数据  
mu\_list=[0,0,0,]  
variance=[0.2,1.0,5.0,]  
normalDistr\_paras=zip(mu\_list,[math.sqrt(sig2) for sig2 in variance]) # 配置多个平均值和标准差对  
s\_list=[np.random.normal(para[0], para[1], 1000) for para in normalDistr\_paras]  
kur=[kurtosis(s,fisher=True) for s in s\_list]  
print("kurtosis for data:",kur)  
  
i=0  
for skew\_data in skewNorm\_list:  
 sns.kdeplot(skew\_data,label="skewness=%.2f"%skewness[i])  
 i+=1  
  
n=0  
for s in s\_list:  
 sns.kdeplot(s,label="μ=%s, $σ^2$=%s,kur=%.2f"%(mu\_list[n],variance[n],kur[n]))  
 n+=1   
plt.legend()  
\_=plt.plot()

skewness for data: [0.9722605767531536, -0.8485166469879731]  
kurtosis for data: [0.1385811357392006, 0.012989429359109739, -0.11650103191067807]



返回POI实验数据，计算POI美食数据价格的偏度和峰度。偏度计算结果值为正，为正偏态，概率密度函数右侧尾部比左侧长，说明多数价格较低，少数较高价格将分布的尾巴托向另一端。峰度为尖峰分布。

delicacy\_price\_df\_clean=delicacy\_price\_df.dropna()  
print("skewness:%.2f, kurtosis:%.2f"%(skew(delicacy\_price\_df\_clean.price),kurtosis(delicacy\_price\_df\_clean.price,fisher=True)))

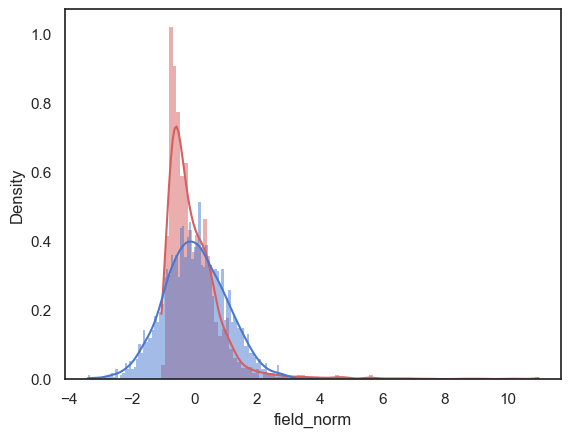
skewness:4.18, kurtosis:29.84

#### 4）检验数据集是否服从正态分布

首先计算标准计分，标准化价格数据集后其平均值为0，标准差为1，绘制概率密度函数分布曲线（理论值）。同时叠加满足上述条件，由NumPy随机生成，满足正态分布数据集的分布曲线（观察值），可以比较二者的差异。因为实验数据（美食价格）是正偏态，可以观察到位于均值左侧的数据有所差异，而右侧趋势基本吻合。同时，具有较高的峰度值，高于标准正态分布曲线。

def comparisonOFdistribution(df,field,bins=100):  
 '''  
 funciton-数据集z-score概率密度函数分布曲线（即观察值/实验值 observed/empirical data）与标准正态分布(即理论值 theoretical set)比较  
   
 Params:  
 df - 包含待分析数据集的DataFrame格式类型数据；DataFrame(Pandas)  
 field - 指定分析数据数据（DataFrame格式）的列名；string  
 bins - 指定频数宽度，为单一整数代表频数宽度（bin）的数量；或者列表，代表多个频数宽度的列表。The default is 100；int;list(int)  
   
 Returns:  
 None  
 '''  
 import pandas as pd  
 import numpy as np  
 import seaborn as sns   
   
 df\_field\_mean=df[field].mean()  
 df\_field\_std=df[field].std()  
 print("mean:%.2f, SD:%.2f"%(df\_field\_mean,df\_field\_std))  
  
 df['field\_norm']=df[field].apply(lambda row: (row-df\_field\_mean)/df\_field\_std) # 标准化价格(标准计分，z-score)，或者使用`from scipy.stats import zscore`方法  
  
 # 验证z-score，标准化后的均值必为0， 标准差必为1.0  
 df\_fieldNorm\_mean=df['field\_norm'].mean()  
 df\_fieldNorm\_std=df['field\_norm'].std()  
 print("norm\_mean:%.2f, norm\_SD:%.2f"%(df\_fieldNorm\_mean,df\_fieldNorm\_std))  
   
 sns.histplot(df['field\_norm'], bins=bins,kde=True,stat="density",linewidth=0,color='r')  
  
 s=np.random.normal(0, 1, len(df[field]))  
 sns.histplot(s, bins=bins,kde=True,stat="density",linewidth=0,color='b')  
   
import pandas as pd  
pd.options.mode.chained\_assignment=None   
  
comparisonOFdistribution(delicacy\_price\_df\_clean,'price',bins=100)

mean:54.19, SD:51.22  
norm\_mean:0.00, norm\_SD:1.00



* 异常值处理

从直方图或者箱型图中可以观察到偏度的差异，有少数部分较高的价格出现，估计为异常值。

处理异常值，先建立简单数据来理解异常值，并找到相应的方法。使用*How to Detect and Handle Outliers*[4]中提供的方法，其公式为：

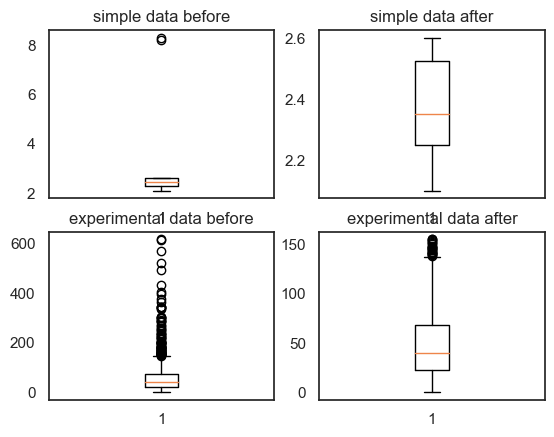
，式中MAD（the median of the absolute deviation about the median /Median Absolute Deviation）为绝对中位差，其中为中位数；

，式中为修正的z-score（modified z-score），为中位数。

由计算结果，确定该公式能够有效的判断异常值。

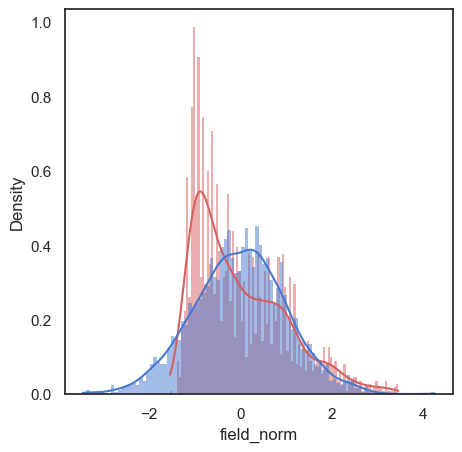
import numpy as np  
import matplotlib.pyplot as plt  
  
outlier\_data=np.array([2.1,2.6,2.4,2.5,2.3,2.1,2.3,2.6,8.2,8.3]) # 建立简单的数据，便于观察  
ax1=plt.subplot(221)  
ax1.margins(0.05)   
ax1.boxplot(outlier\_data)  
ax1.set\_title('simple data before')  
  
def is\_outlier(data,threshold=3.5):  
 '''  
 function-判断异常值  
   
 Params:  
 data - 待分析的数据，列表或者一维数组；list/array  
 threshold - 判断是否为异常值的边界条件, The default is 3.5；float  
   
 Returns  
 is\_outlier\_bool - 判断异常值后的布尔值列表；list(bool)  
 data[~is\_outlier\_bool] - 移除异常值后的数值列表；list  
 '''  
 import numpy as np   
   
 MAD=np.median(abs(data-np.median(data)))  
 modified\_ZScore=0.6745\*(data-np.median(data))/MAD  
 is\_outlier\_bool=abs(modified\_ZScore)>threshold   
 return is\_outlier\_bool, data[~is\_outlier\_bool]  
   
is\_outlier\_bool,data\_clean=is\_outlier(outlier\_data,threshold=3.5)   
print(is\_outlier\_bool,data\_clean)  
  
plt.rcParams["figure.figsize"]=(10,10)  
ax2=plt.subplot(222)  
ax2.margins(0.05)   
ax2.boxplot(data\_clean)  
ax2.set\_title('simple data after')  
  
\_,delicacyPrice\_outliersDrop=is\_outlier(delicacy\_price\_df\_clean.price,threshold=3.5)  
print("原始数据描述:",delicacy\_price\_df\_clean.price.describe(),"\n")  
print("-"\*50)  
print("异常值处理后数据描述:", delicacyPrice\_outliersDrop.describe(), "\n")  
  
ax3=plt.subplot(223)  
ax3.margins(0.05)   
ax3.boxplot(delicacy\_price\_df\_clean.price)  
ax3.set\_title('experimental data before')  
  
ax3=plt.subplot(224)  
ax3.margins(0.05)   
ax3.boxplot(delicacyPrice\_outliersDrop)  
ax3.set\_title('experimental data after')  
  
plt.show()

[False False False False False False False False True True] [2.1 2.6 2.4 2.5 2.3 2.1 2.3 2.6]  
原始数据描述: count 2303.000000  
mean 54.186062  
std 51.215379  
min 0.000000  
25% 22.000000  
50% 41.000000  
75% 72.000000  
max 617.000000  
Name: price, dtype: float64   
  
--------------------------------------------------  
异常值处理后数据描述: count 2232.000000  
mean 47.695116  
std 30.958886  
min 0.000000  
25% 22.000000  
50% 40.000000  
75% 68.000000  
max 155.000000  
Name: price, dtype: float64



plt.rcParams["figure.figsize"]=(5,5)  
comparisonOFdistribution(pd.DataFrame(delicacyPrice\_outliersDrop,columns=['price']),'price',bins=100)

mean:47.70, SD:30.96  
norm\_mean:0.00, norm\_SD:1.00



如果将较高的价格视为异常值移除，调用定义的comparisonOFdistribution(df,field,bins=100)函数打印概率密度函数，比较标准正态分布可以发现右侧较高值的部分发生了变化，其它部分未发生明显变化。那么如何检验数据集是否服从正态分布？SciPy库提供有多种正态性检验的方法，分别为kstest，shapiro，normaltest和anderson。计算结果显示的p-value基本为0，即p-value<0.05，拒绝原假设，表明清理异常值后的美食价格数据集仍不服从正态分布，为非正态数据集。只有服从正态分布的数据可以计算从一个总体中选取特定值或区间的概率，本次美食价格数据集为正偏和尖峰分布，为非正态数据集，正态分布的概率不能很好适用于该类数据集。

SciPy检验数据集是否服从正态分布的返回值，第1个为统计量，该值越接近0表明数据和标准正态分布拟合的越好；第2个为p-value值，如果该值大于0.05的显著性水平，接受原假设，判断样本的总体服从正态分布。

from scipy import stats  
  
kstest\_test=stats.kstest(delicacyPrice\_outliersDrop,cdf='norm')  
print("original data:",kstest\_test)  
z\_score\_kstest\_test=stats.kstest(stats.zscore(delicacyPrice\_outliersDrop),cdf='norm')  
print('z\_score:',z\_score\_kstest\_test)  
  
# 官方文档注释，当N>5000时，只有统计量正确，但是p-value值不一定正确。本次实验`len(delicacyPrice\_outliersDrop)`数据量为770，可以使用  
shapiro\_test=stats.shapiro(delicacyPrice\_outliersDrop)   
print("original data - shapiroResults(statistic=%f,pvalue=%f)"%(shapiro\_test))  
  
normaltest\_test=stats.normaltest(delicacyPrice\_outliersDrop,axis=None)  
print("original data:",normaltest\_test)  
  
anderson\_test=stats.anderson(delicacyPrice\_outliersDrop,dist='norm')  
print("original data:",anderson\_test)

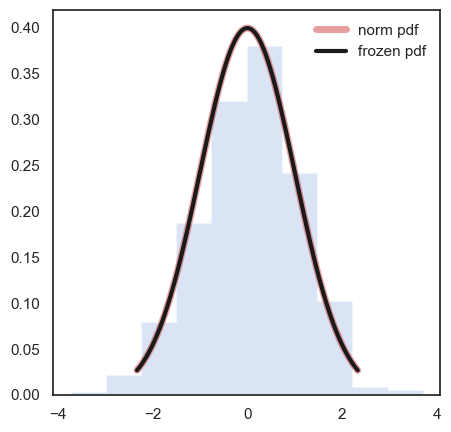
original data: KstestResult(statistic=0.999551684674593, pvalue=0.0)  
z\_score: KstestResult(statistic=0.11464960111854455, pvalue=5.153632014592587e-26)  
original data - shapiroResults(statistic=0.918487,pvalue=0.000000)  
original data: NormaltestResult(statistic=232.93794469002853, pvalue=2.6191959927510468e-51)  
original data: AndersonResult(statistic=53.19820994109705, critical\_values=array([0.575, 0.655, 0.786, 0.916, 1.09 ]), significance\_level=array([15. , 10. , 5. , 2.5, 1. ]))

#### 5）给定特定值计算概率，及找到给定概率的值

如果数据集服从正态分布，则可以直接使用SciPy库stats类中提供的norm方法进行计算。下述案例使用norm.cdf()、norm.sf()和norm.ppf()分别计算小于或等于特定值、大于或等于特定值、或找到给定概率的值进行计算。

from scipy.stats import norm  
import matplotlib.pyplot as plt  
import numpy as np  
  
fig, ax=plt.subplots(1, 1)  
mean, var, skew, kurt = norm.stats(moments='mvsk')   
print('mean, var, skew, kurt=',(mean, var, skew, kurt)) # 验证符合标准正态分布的相关统计量  
x=np.linspace(norm.ppf(0.01),norm.ppf(0.99), 100) # norm.ppf 百分比点函数 - Percent point function (inverse of cdf — percentiles)  
ax.plot(x, norm.pdf(x), 'r-', lw=5, alpha=0.6, label='norm pdf') # norm.pdf为概率密度函数  
rv=norm() # 固定形状（偏度和峰度）、位置loc（平均值）和比例scale（标准差）参数，即指定固定值  
ax.plot(x, rv.pdf(x), 'k-', lw=3, label='frozen pdf') # 固定/“冻结”分布（frozen distribution）  
  
vals = norm.ppf([0.001, 0.5, 0.999]) # 返回概率为0.1%、50%和99.9%的值，默认loc=0,scale=1  
print("验证累积分布函数CDF返回值与其PPF返回值是否相等或近似：",np.allclose([0.001, 0.5, 0.999], norm.cdf(vals)))  
  
r=norm.rvs(size=1000) # 指定数据集大小  
ax.hist(r, density=True, histtype='stepfilled', alpha=0.2)  
ax.legend(loc='best', frameon=False)  
plt.show()

mean, var, skew, kurt= (array(0.), array(1.), array(0.), array(0.))  
验证累积分布函数CDF返回值与其PPF返回值是否相等或近似： True

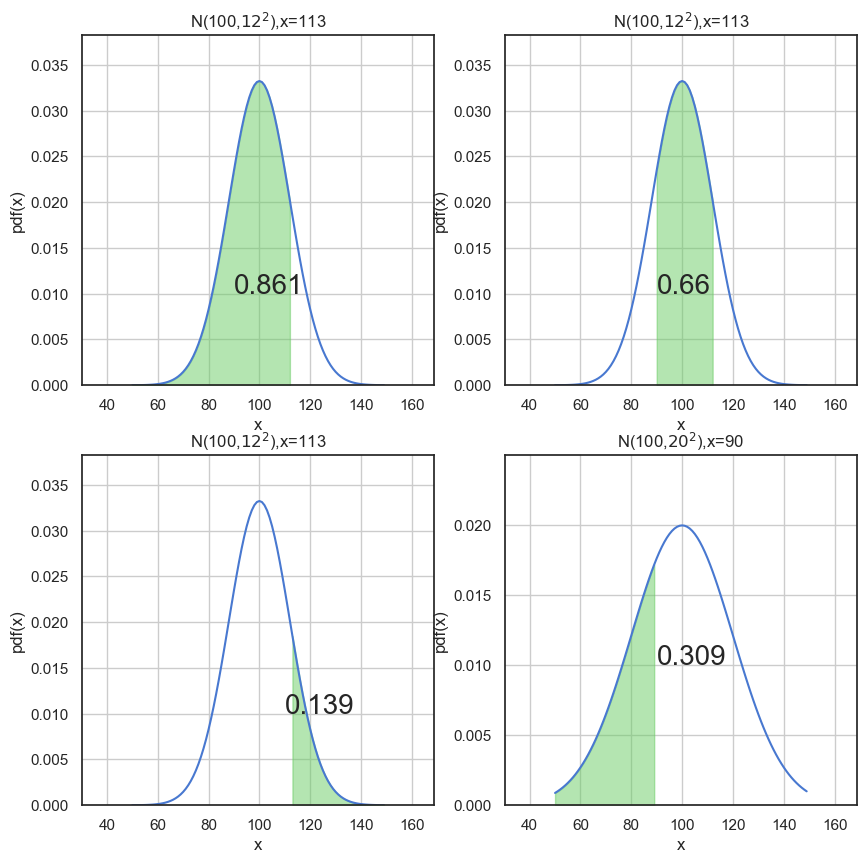


# 如果需要计算概率则定义固定分布的数据集。  
print("用.cdf计算值小于或等于113的概率为：",norm.cdf(113,100,12)) # pdf(x, loc=0, scale=1) 配置Loc(均值)和scale(标准差)  
print("用.sf计算值大于或等于113待概率为：",norm.sf(113,100,12))   
print("可以观察到.cdf（<=113）概率结果+.sf(>=113)概率结果为：",norm.cdf(113,100,12)+norm.sf(113,100,12))  
print("用.ppf找到给定概率值为98%的数值为：",norm.ppf(0.86066975,100,12))

用.cdf计算值小于或等于113的概率为： 0.8606697525503779  
用.sf计算值大于或等于113待概率为： 0.13933024744962208  
可以观察到.cdf（<=113）概率结果+.sf(>=113)概率结果为： 1.0  
用.ppf找到给定概率值为98%的数值为： 112.99999986204986

在传统的概率计算中，已知z-score（标准计分），可以通过查表的方式来获取对应的概率值，这种方式已经很少使用。为了直观的观察概率值，由曲线、横轴和通过对应概率密度函数值的垂直线围合的面积即为概率值，通过绘制图表可以更清晰的观察。

def probability\_graph(x\_i,x\_min,x\_max,x\_s=-9999,left=True,step=0.001,subplot\_num=221,loc=0,scale=1):   
 '''  
 function - 正态分布概率计算及图形表述  
   
 Paras:  
 x\_i - 待预测概率的值；float  
 x\_min - 数据集区间最小值；float  
 x\_max - 数据集区间最大值；float  
 x\_s - 第2个带预测概率的值，其值大于x\_i值。The default is -9999；float  
 left - 是否计算小于或等于，或者大于或等于指定值的概率。The default is True；bool  
 step - 数据集区间的步幅。The default is 0.001；float  
 subplot\_num - 打印子图的序号，例如221中，第一个2代表列，第二个2代表行，第三个是子图的序号，即总共2行2列总共4个子图，1为第一个子图。The default is 221；int  
 loc - 即均值。The default is 0；float  
 scale - 标准差。The default is 1；float  
   
 Returns:  
 None  
 '''  
 import math  
 import numpy as np  
 from scipy.stats import norm  
   
 x=np.arange(x\_min,x\_max,step)  
 ax=plt.subplot(subplot\_num)  
 ax.margins(0.2)   
 ax.plot(x,norm.pdf(x,loc=loc,scale=scale))  
 ax.set\_title('N(%s,$%s^2$),x=%s'%(loc,scale,x\_i))  
 ax.set\_xlabel('x')  
 ax.set\_ylabel('pdf(x)')  
 ax.grid(True)  
   
 if x\_s==-9999:  
 if left:  
 px=np.arange(x\_min,x\_i,step)  
 ax.text(loc-loc/10,0.01,round(norm.cdf(x\_i,loc=loc,scale=scale),3), fontsize=20)  
 else:  
 px=np.arange(x\_i,x\_max,step)  
 ax.text(loc+loc/10,0.01,round(1-norm.cdf(x\_i,loc=loc,scale=scale),3), fontsize=20)  
   
 else:  
 px=np.arange(x\_s,x\_i,step)  
 ax.text(loc-loc/10,0.01,round(norm.cdf(x\_i,loc=loc,scale=scale)-norm.cdf(x\_s,loc=loc,scale=scale),2), fontsize=20)  
 ax.set\_ylim(0,norm.pdf(loc,loc=loc,scale=scale)+0.005)  
 ax.fill\_between(px,norm.pdf(px,loc=loc,scale=scale),alpha=0.5, color='g')  
   
plt.figure(figsize=(10,10))  
probability\_graph(x\_i=113,x\_min=50,x\_max=150,step=1,subplot\_num=221,loc=100,scale=12)  
probability\_graph(x\_i=113,x\_min=50,x\_max=150,step=1,left=False,subplot\_num=223,loc=100,scale=12)  
probability\_graph(x\_i=113,x\_min=50,x\_max=150,x\_s=90,step=1,subplot\_num=222,loc=100,scale=12)  
probability\_graph(x\_i=90,x\_min=50,x\_max=150,step=1,subplot\_num=224,loc=100,scale=20)  
plt.show()



注释（Notes）：

① 百度地图开放平台（API），是为开发者免费提供的一套基于百度地图服务的应用接口，包括JavaScript API、Web服务API、Android SDK、iOS SDK、定位SDK、车联网API、LBS云等多种开发工具与服务，提供基本地图展现、搜索、定位、逆/地理编码、路线规划、LBS云存储与检索等功能，适用于PC端、移动端、服务器等多种设备，多种操作系统下的地图应用开发（<https://lbsyun.baidu.com/index.php?title=%E9%A6%96%E9%A1%B5>）。

② 兴趣点（POI，points of interest），是一个具有一定意义的特定点位置，例如地球上代表菲尔铁塔（Eiffel Tower）的位置，或是火星上代表最高峰奥林匹斯山（Olympus Mons）的位置。在提及酒店、露营地、加油站或汽车导航系统中任何其它类别都会使用这个术语（<>）。

③ 服务文档，接口功能介绍，含参数说明（<https://lbsyun.baidu.com/index.php?title=webapi/guide/webservice-placeapi>）。

④ 百度地图坐标拾取系统，支持POI点坐标显示、复制；地址精确/模糊查询；坐标鼠标跟随显示；支持坐标查询（<http://api.map.baidu.com/lbsapi/getpoint/index.html>）。

⑤ Spatial Reference， 空间参考系统（spatial reference system，SRS)或坐标参考系统（ coordinate reference system，CRS）是用于精确测量地球表面位置，用坐标表示的一个体系。是坐标系和解析几何的抽象数学在地理空间的应以。例如一个CRS坐标表示为：Universal Transverse Mercator WGS 84 Zone 16，包括选择地球椭圆体、水平基准、地图投影，原点和计量单位等（<https://spatialreference.org/>）。

⑥ Plotly，是一个通过数据应用实现数据驱动决策的实践者，构建有支持多种编程语言的图表库（<https://plotly.com/>）。

⑦ Plotly Python，Plotly的Python图形库可以制作互动的、具有出版质量的图表。例如制作线图（line plots）、散点图（scatter plots）、面积图（area charts）、条形图（bar charts）、误差条（error bars）、箱形图（box plots）、柱状图（histograms）、热图（heatmaps）、子图（subplots）、多轴图（multiple-axes）、极地图（polar charts）和气泡图（bubble charts）等（<https://plotly.com/python/>）。

⑧ mapbox，用于移动和网络应用的位置数据平台，提供定制在线地图的供应商（<https://www.mapbox.com/>）。

⑨ Matplotlib，<https://matplotlib.org/>；Plotly（含dash），<https://plotly.com/>；Bokeh，<https://docs.bokeh.org/en/latest/index.html>；seaborn，<https://seaborn.pydata.org/>。

参考文献（References）:

[1] (日)高桥麻奈著,崔建锋译.株式会社TREND-PRO漫画制作.漫画数据库[M].科学出版社.北京,2010.5.

[2] (日)高桥 信著,陈刚译.株式会社TREND-PRO漫画制作.漫画统计学[M].科学出版社.北京,2019.8.

[3] Timothy C.Urdan(蒂莫西•C•厄丹),彭志文译.白话统计学[M].中国人民大学出版社.2013,12.第3版. <Timothy C. Urdan.Statistics in plain English[M]. New York: Routledge, 2010>.

[4] Boris Iglewicz and David Hoaglin (1993), “Volume 16: How to Detect and Handle Outliers”, The ASQC Basic References in Quality Control: Statistical Techniques, Edward F. Mykytka, Ph.D., Editor.