大作业汇总文档

**图片的展示**

一开始拿到数据,是covid-19 那一大堆的数据,但是他是以时间作为列名的,而且导入之后有一个很难受的点,就是日期有不同的格式,有些是 yyyy-mm-dd 有些是 yyyy/mm/dd 而且我还不会把列名转化成日期的形式.而且想想也是,列名是作为hash的key的,他难道可以转成日期吗,一般是字符串吧,这个也不太清楚.

一开始尝试去写,但是写不出来,经过一段时间的挣扎之后,突然想到,可以把整个数据做一个转置,这样列名就变成数据了,就可以转化了

转置之前的数据

图形用户界面, 表格

描述已自动生成

转置之后的数据,并且把id hasc 去掉了

图形用户界面, 应用程序, 表格, Excel

描述已自动生成

此文件画了堆积柱状图

ame.py

D:\proj\visualization\time\_data\_visual\ame.py

D:\proj\visualization\bigwork\ame.py

我把他的日期稍微旋转了一下,因为正着的话 会挡住

还有一个细节就是高的这些要先画,不然先画短的的话,高的这些会把他们覆盖

图表

描述已自动生成

D:\proj\visualization\bigwork\ame\_sta.py

图表, 直方图

描述已自动生成

美国某几个州的堆积面积图

图表, 直方图

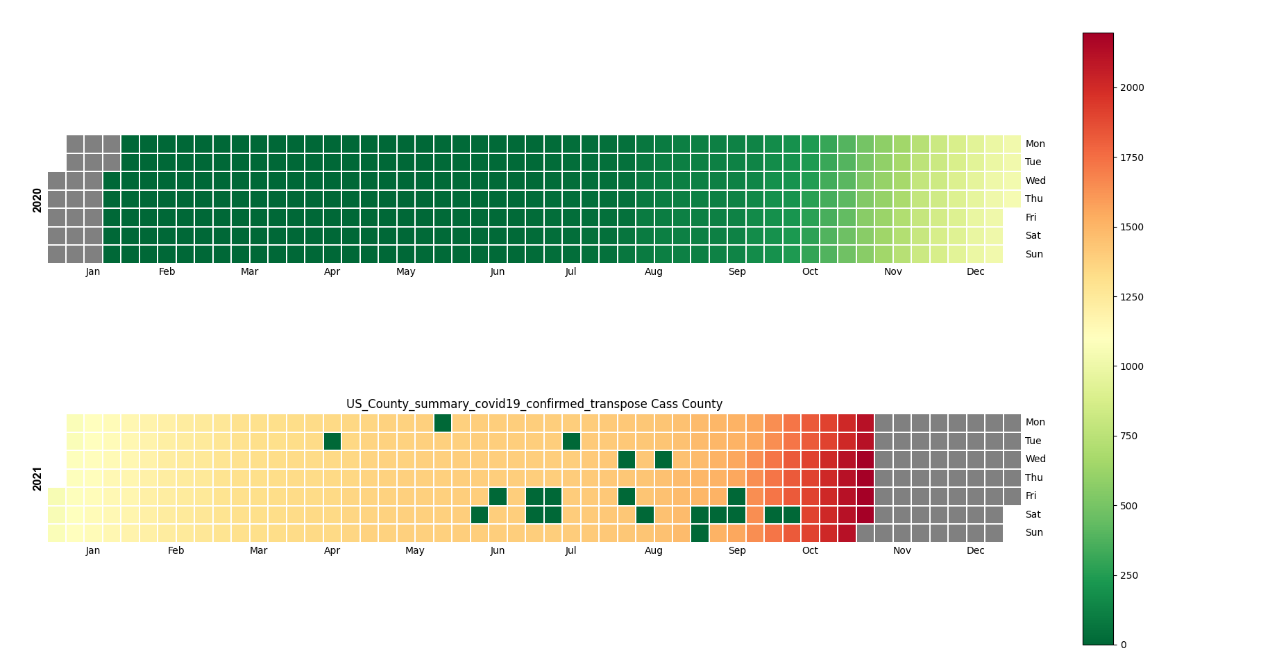
描述已自动生成

美国几个州的百分比堆积图

D:\proj\visualization\bigwork\ame\_sta.py

D:\proj\visualization\bigwork\ame\_calendar.py

这个代码画的图



美国 cass 县 疫情确诊 日历图

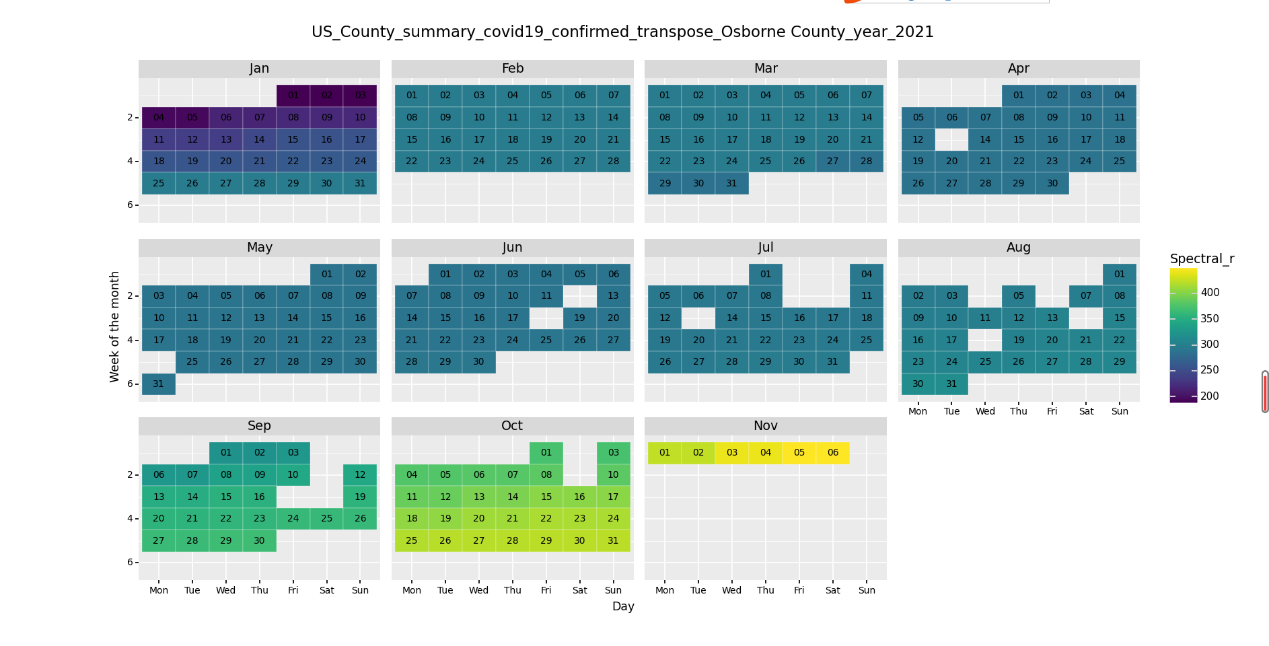
D:\proj\visualization\bigwork\ame\_calen\_month.py

D:\proj\visualization\bigwork\ame\_calendar.py

日历

描述已自动生成

美国osborne 州 在2020年 确诊人数的日历图,颜色分明,看到越到后来越多



美国osborne 州 在2021年 确诊人数的日历图,颜色分明,看到越到后来越多

图表, 箱线图

描述已自动生成

通过箱型图可以看到,2020/7/20 是属于中位数的位置

图表, 箱线图

描述已自动生成

这张图忘记是画的什么了

图表, 气泡图

描述已自动生成

表格

描述已自动生成

图片包含 图形用户界面

描述已自动生成

交互式窗口运行当前文件

图表, 折线图

描述已自动生成

美国的一些州 随着时间的变化 他们的确诊人数的折线图

图表, 直方图

描述已自动生成

美国某个州的新冠确诊随时间变化柱状图

图表, 直方图

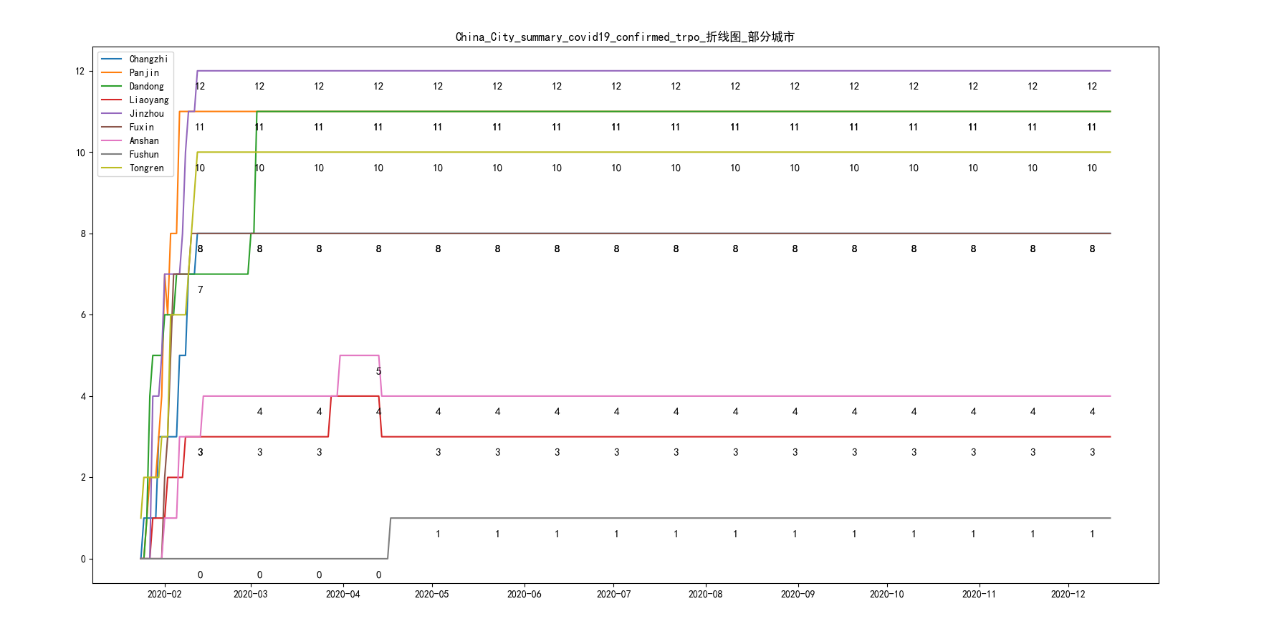
描述已自动生成

美国部分州的新冠确诊随时间变化柱状图,这个是堆积柱状图

图表, 表格, Excel

描述已自动生成

中国部分城市的新冠确诊随时间变化折线图



中国部分城市的新冠确诊随时间变化折线图

表格

描述已自动生成

中国部分城市的新冠确诊随时间变化折线图 从第10个到第20个

图表

描述已自动生成

折线图

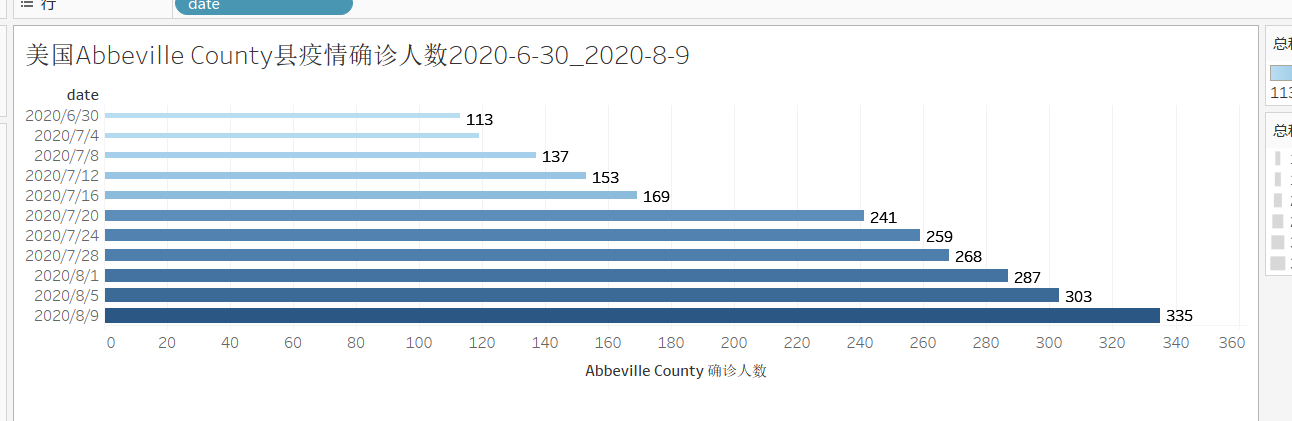
可以知道在 2020-02 --- 2020-12 这段时间里 昭通的确诊人数比较多

柱状图

图表, 条形图

描述已自动生成

横向柱状图



色块图

图表, 树状图

描述已自动生成

柱状图

图表, 条形图

描述已自动生成

散点图

图表

描述已自动生成

饼图

图表, 饼图

描述已自动生成

点方法- 矩阵散点图\_modi

图表

描述已自动生成

这些都做了和，也就是说，死亡和确诊看起来像是线性关系

图表

描述已自动生成

词云图

todo



关于词云图,因为是 gif,所以不适合放在word里面,我之后会以附件的形式给出,这里可以看一下其中的一帧

中国 关于时间的疫情情况的散点图

图表, 气泡图

描述已自动生成

到2021-01-25 为止,全世界治愈人数的气泡图

地图

描述已自动生成

到2021-01-25 为止,全世界确诊人数的分布图,用黄到棕色的渐变,比较适合表示确诊人数

地图

描述已自动生成

到2021-01-25 为止,全世界治愈人数的分布图,用黄到绿色的渐变,比较适合表示治愈人数.因为美国,印度等确诊人数多,所以相应的,他们的治愈人数自然也多

图片包含 图表

描述已自动生成

到2021-01-25 为止,全世界累计死亡人数的柱状图, 全世界的话 太多了,我们只看前面部分,可以看到,形成了"一超多强"的局面

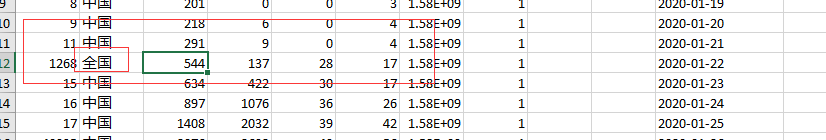


世界到2021-01-25 累计死亡人数 突出显示图.也是同样的情况

表格

描述已自动生成

这里貌似有点问题 怎么有个全国



表格里确实有这样的数据 但是对照上下文 我感觉这个全国 就是中国的意思 所以 把他改掉了

可以看到 随着时间的增长 我国新冠肺炎治愈人数在不断增多,这个tableau 画的 非常长,所以用qq 截长图.这个图还是非常直观的,个数有标记,数量大的话 颜色就深

我这里只给出前面部分,整个的长图会以附件形式给出

图表, 直方图

描述已自动生成

这图虽然挺好看的 但是标记没有出来,虽然点了,但是标记不出来,所以只能这样子,里面的内容是理解不到了

图表

描述已自动生成

这个颜色应该治愈的比较合适

地图

描述已自动生成

**我有画动态柱状图**

但是因为是视频的形式,不适合放在word里,于是我会以附件和链接的形式给出

todo

"G:\video\疫情动态图全世界bandicam 2021-12-03 22-26-06-175.mp4"

这个是 confirmed

"G:\video\疫情动态图全世界 小bandicam 2021-12-03 22-20-43-999.mp4"

confirm

因为我觉得这个数据的格式很奇怪,他是某个城市,然后以时间为列名,来写出他的确诊人数的,这很难去画出折线图什么的,所以我之前是做过转置的.但是在画图的过程中,我突然想到,这种数据是不是用来做b站上经常看到的那种动态图呢,就是每天都会变化的柱状图.

因为b站很多这种的,于是去b站找了,果然找到了一个网站,专门来画这种图.todo. 而且这种格式的数据可以非常简单的放进去,马上就出图.所以这种数据就是用来做这个的. 如果硬是要把这个数据用来做其他的图,那会变得很麻烦,数据的处理非常的痛苦,而且画出来的效果还不好.所以哪种格式的数据,来画哪种图,他的讲究也是挺多的

通过动态柱状图,我们可以看到,中国虽然一开始感染的人多,因为中国是始发地,但是后来慢慢变少了,因为我国的政策好,国家高度重视,严格管理,做好14天的隔离措施,决不放弃每一个人,都尽全力去救治的,而且都不用我们老百姓花钱,之前核酸检测也是免费的,现在也可以免费打疫苗.

**分类任务**

clusters 分类 聚类的模型

首先是跑了一下他原本的代码

D:\proj\visualization\bigwork\covid-19-visualizations-predictions-forecasting.ipynb

表格, 日程表, 日历

描述已自动生成

todo

cpg

地图功能

basemap

python xlsx 转置

<https://blog.csdn.net/zkw_1998/article/details/112616925>

根据此代码画出

D:\proj\visualization\bigwork\covid-19-visualizations-predictions-forecasting copy.ipynb

实际上是这个代码

D:\proj\visualization\bigwork\累计患病的分布(各个县)死亡率.py

*# 层次聚类测试*

**def** HierarchicalClusteringTest():

使用这个方法画出来的 树形图

图表, 直方图, 箱线图

描述已自动生成

*# 肘部法则*

**def** ElbowMethod():

使用*肘部法则 画出来的图*

图表, 折线图

描述已自动生成

显示分成三类 好一点

所有的方法，即肘部法和层次聚类显示K=3将正确的聚类。

表格

描述已自动生成

但是我们发现,有一类是比较奇怪的,他的 治愈率有253%,这是不应该的,治愈率应该最高就100%啊. 所以我们发现了有错误的数据,就是有unkown这样的一列数据, 这个数据自然是无效的数据,我们把他去掉

countrywise=countrywise.drop(index="Unknown")

.....

图表, 折线图

描述已自动生成

去掉 unkown 之后

图表, 箱线图

描述已自动生成

树状图地下的数据看不清楚,但是问题不大,我们只需要知道,根据最后的颜色,他是分成三类的

表格

描述已自动生成

分成了 三类 ,1 --- 死亡率较高,治愈率 一般, 2 -- 治愈率比较低 , 0- 治愈率较高,死亡率较低

可以看到大多数城市还是治愈率较高,死亡率较低,所以中国的治愈率还是比较好.考虑到台湾的治愈率比较低,应该是数据没给的原因,而不是真的治愈率低

当然,由于台湾比较自由,政府采取的政策也是比较放松的,也不会有政府的大量拨款去治疗确诊病人,所以他们的治愈率会比较低

**利用gis的数据画图**

当我看到gis的数据格式的时候,我觉得很开心,因为他终于是比较好作图的格式了,就是类似数据库的格式,每个列都有个列名,这样作图的时候只要 plt.plot(df[列名'],df[另外一个列名']) 就可以了

filename=r"D:\download\GIS疫情地图2020全年-至今数据\【GIS点滴疫情地图·2020年01月02日-2021年01月25日】国内每天疫情统计.xlsx" df = pd.read\_excel(filename) df.plot() plt.show()

x=df[**"day"**]

y=df[**"diagnose"**]

*# plt.bar(x, y,label="确诊")*

*# 柱状图*

图表, 条形图, 直方图

描述已自动生成

D:\proj\visualization\bigwork\gis.py

图片包含 图形用户界面

描述已自动生成

plt.plot(x, y,label=**"确诊"**)

plt.plot(x, df[**"suspect"**],label=**"疑似"**)

plt.plot(x, df[**"cure"**],label=**"治愈"**)

plt.plot(x, df[**"death"**],label=**"死亡"**)

plt.plot(x, df[**"seriousCount"**],label=**"严重"**)

plt.plot(x, df[**"currentdiagnose"**],label=**"目前确诊"**)

*# 折线图*

*我们也可以画出各种状况的折线图*

图示

描述已自动生成

图表, 直方图

描述已自动生成

每日新增的状况

**跑SIR模型**

在网上找到一个模型,然后拿来跑了我们的数据,它可以根据目前的患病和治愈人数来推测之后的患病和治愈人数的多少

53

图片包含 应用程序

描述已自动生成

并且模拟了 传播率

图表, 直方图

描述已自动生成

模拟了治愈率,以及他们和真正的治愈率的比较,可以看到是比较准确的

图表, 直方图

描述已自动生成

明天的确诊数量: 100380.0

明天的被感染数量: 2968.0

明天的治愈加上死亡的数量: 97412.0

结束的那天: 101

结束那天会有多少确诊: 102618.0

转折点: 59

The latest transmission rate beta of SIR model: 0.0704020040699093

The latest recovering rate gamma of SIR model: 0.03140794223826715

The latest basic reproduction number R0: 2.241535072110905

这里的工作是做了一些汉化,之前治愈率,传播率的图是没有标题的,我也加了一下

**SIR模型**

SIR模型是是一种[传播模型](https://baike.baidu.com/item/%E4%BC%A0%E6%92%AD%E6%A8%A1%E5%9E%8B/16916507)，是[信息传播](https://baike.baidu.com/item/%E4%BF%A1%E6%81%AF%E4%BC%A0%E6%92%AD/7817547)过程的[抽象](https://baike.baidu.com/item/%E6%8A%BD%E8%B1%A1/9021815)描述。是传染病模型中最经典的模型，其中S表示[易感者](https://baike.baidu.com/item/%E6%98%93%E6%84%9F%E8%80%85/604696)，I表示[感染者](https://baike.baidu.com/item/%E6%84%9F%E6%9F%93%E8%80%85/932259)，R表示移出者。

[SIR模型\_百度百科 (baidu.com)](https://baike.baidu.com/item/SIR%E6%A8%A1%E5%9E%8B/1938321)

这是一个悠久的模型,当然网上就会有很多资料,于是拿了一个拿来用了.因为是边学习边做嘛,当然不可能自己手写算法啦,于是大概去了解了一下就用在自己的数据上了.而且时间紧迫,所以就用前人的智慧了.

# 不过其实有做工作,比如说他的数据是 np.array 传入的, 写死在内存里,我拿自己的xlsx 里面的数据读入的时候,使用的是一列数据,

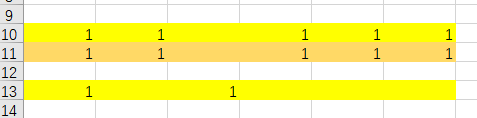
# 比如 df["diagnose"] 是 确诊的人数的一个列 ,于是他在进行 (X[1:] - X[:-1] + R[1:] - R[:-1]) 运算的时候

# 就会出错, 原因是 他是df ,所以他会错开相减,最后会多出一个数字,举个例子,就是如此

日程表

描述已自动生成

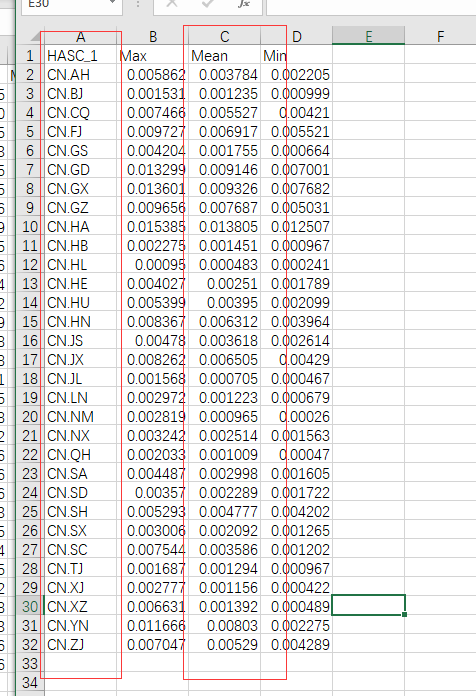
而np.array 的相减是这样的,所以会少一个维度



我在测试中,学习到了这个知识,然后使得模型可以正确运行了

**降维**

拿出 降水的 平均数据,湿度 的平均数据 等等, 每一天, 拼成一行



想法是 按照 感染人数多少 分成 几类

图表, 折线图

描述已自动生成

图表, 箱线图

描述已自动生成

图片包含 图形用户界面

描述已自动生成

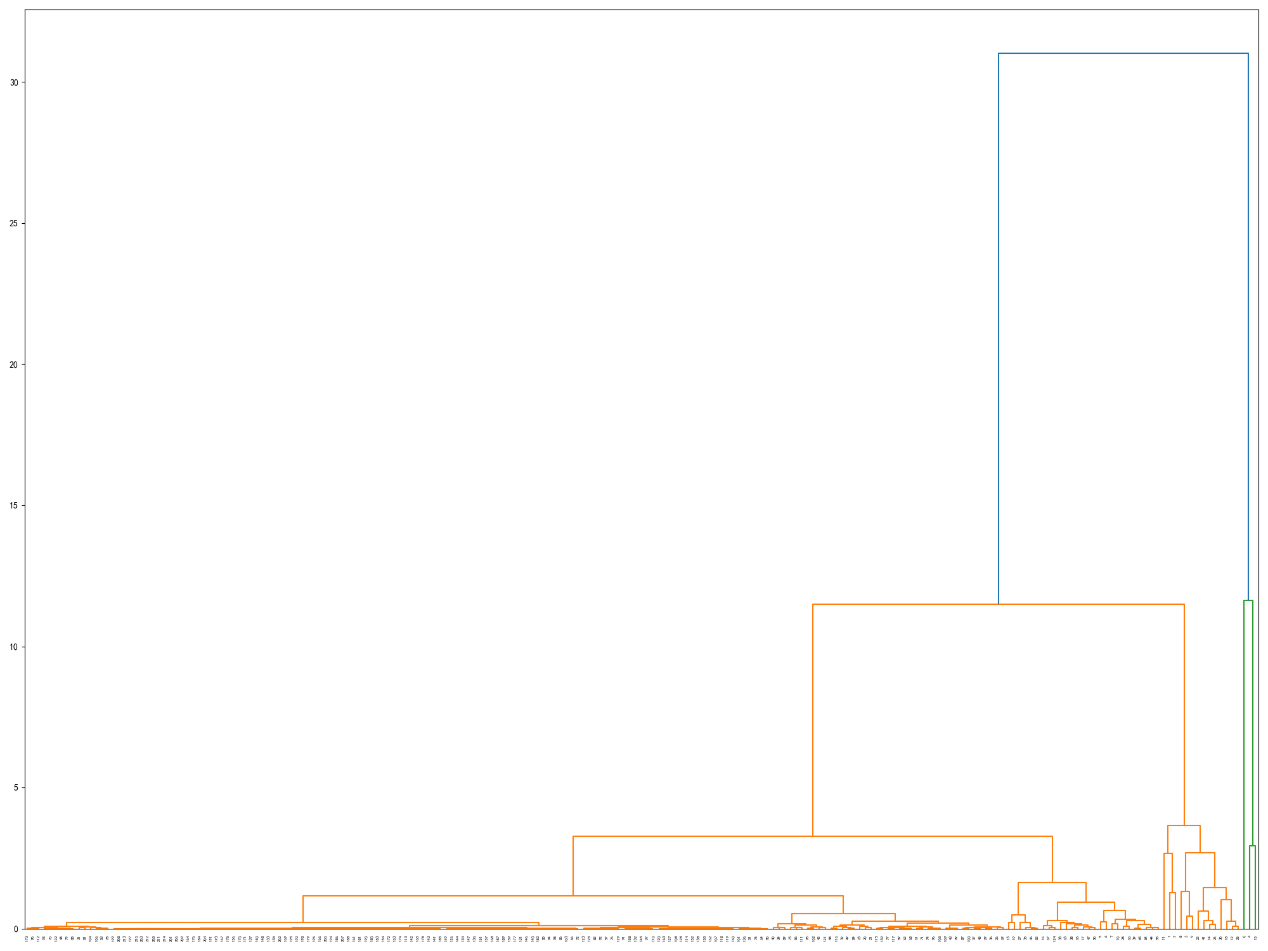
但是这边做错了,这里是按照时间的先后来分了,这样子当然是时间越往后面人数越多了,所以这个不合理,我需要重新整理一下数据

于是用这个代码,重新做了分类,是按照国家来分.分类之前依旧是做了肘部法则和*层次聚类测试,显示是分成2类比较好*

D:\proj\visualization\bigwork\累计患病的分布(各个县)死亡率gis数据.py

图表, 折线图

描述已自动生成



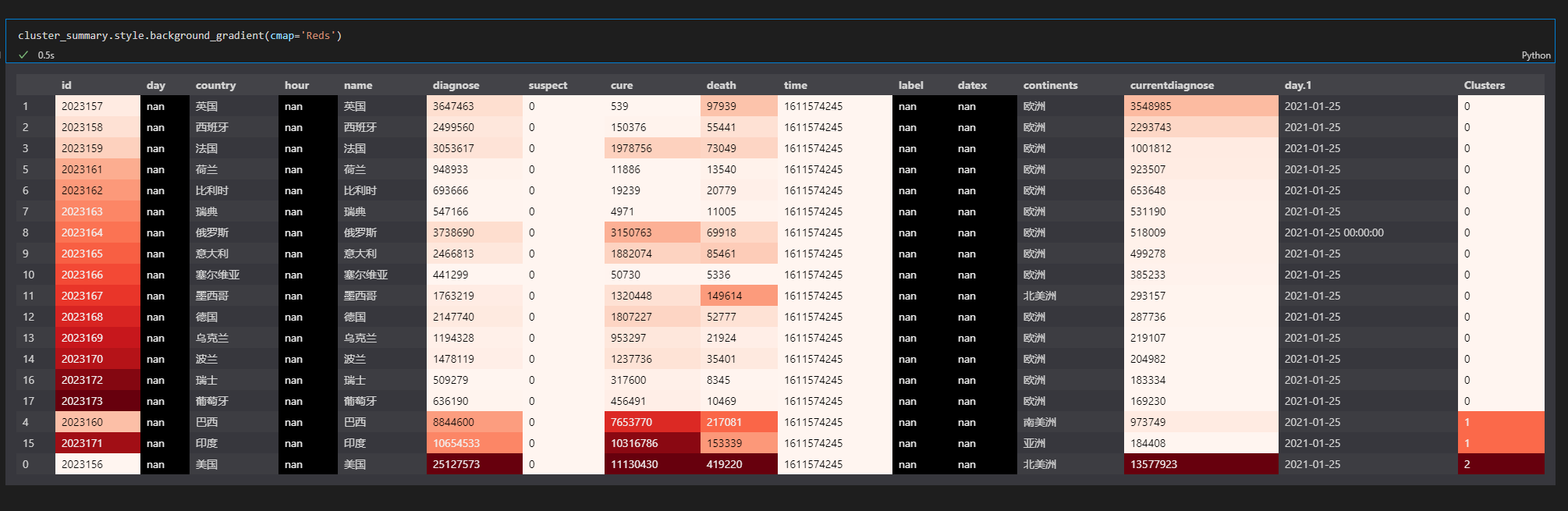
D:\proj\visualization\bigwork\累计患病的分布(各个县)死亡率gis数据.py

不过分成两类之后我感觉不太好,于是尝试分成3类和4类

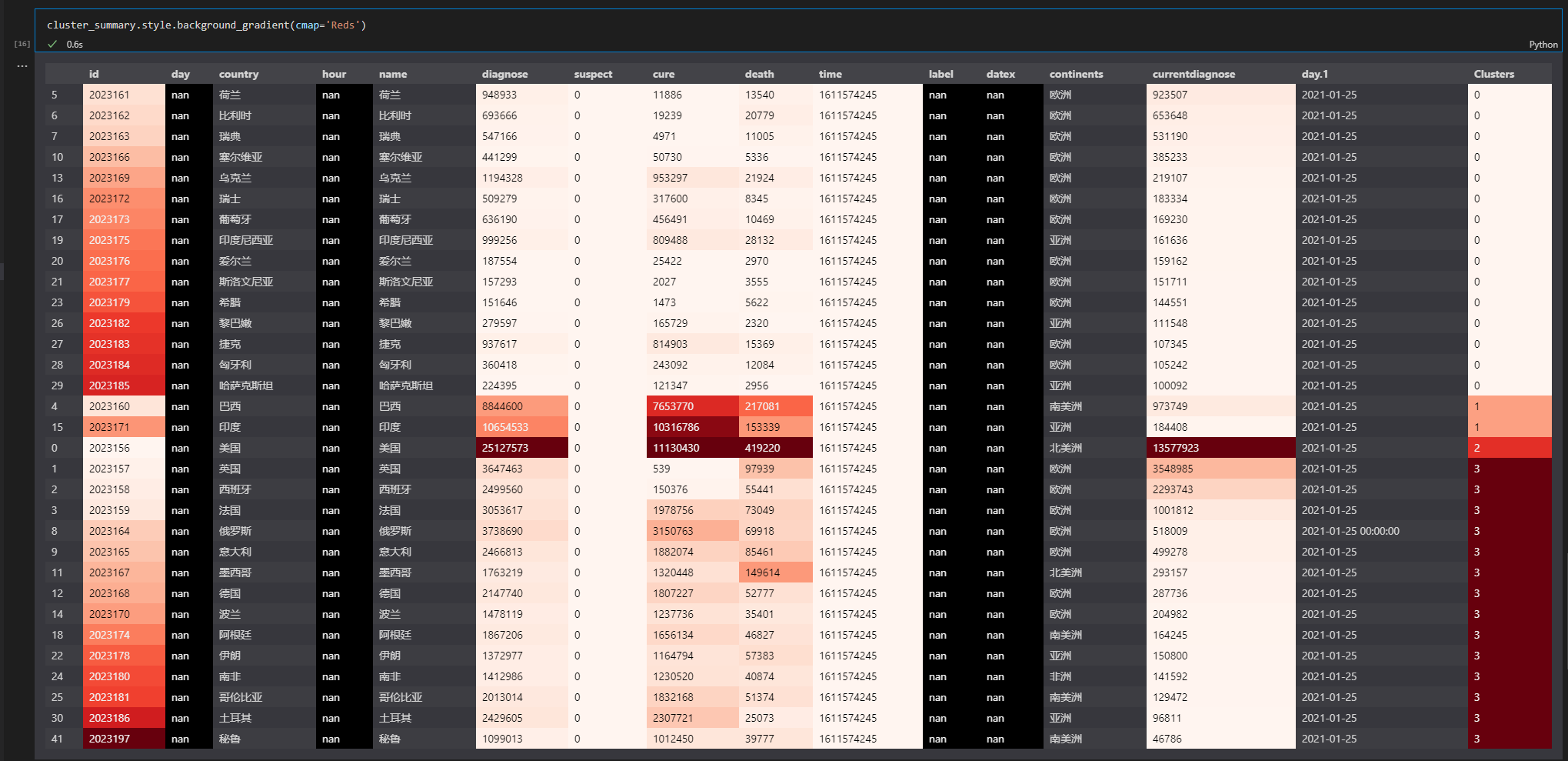
图片包含 应用程序

描述已自动生成

D:\proj\visualization\bigwork\分类gis数据世界2021-1-25.ipynb



cluster\_summary=util.final\_KMeans(X,3,covid)



n\_clusters=4

cluster\_summary=util.final\_KMeans(X,n\_clusters,covid)

分成四类之后 感觉是比较合理的

因为是按照国家来分类,数据给的很多是一个国家的所以地区是什么样子的地理环境,所以我觉得不太合适,于是写了代码把整个国家的地理环境数据整合起来

可以把温度 湿度 求和 来做 作为 国家的所有指标吧. 不过要平均数 或者中位数 因为一个国家的城市可能太多 了

*# 但是因为我们现在是给国家做标签,而给的数据是 一个国家的很多个城市各自的降水量啊什么的*

*# 所以对应不上 于是要自己写代码 把这些数据集合成一个 国家的数据,合成各个国家的表 来比较分类*

*# 于是就是遍历这些文件夹 把一个国家的所有数据拿出来 他们的湿度合成一个加起来的值*

*# 温度也是 .然后写入一个文件, 这整个过程还是有点慢的 因为有好多文件 我觉得至少要1分钟把*

*# 所以 必须写入一个文件 做一个cache*

*# 但是考虑到sum 的话 如果一个国家有很多个 城市 他的sum 肯定会大 觉得用mean 会更加合适 于是又做了一个mean 的表格*

*D:\proj\visualization\bigwork\获得高维数据\_国家合体.py*

图表, 散点图

描述已自动生成

D:\proj\visualization\bigwork\降维- t-SNE图\_世界.py

我用这个代码画出了 各个国家在环境因素上的分类图, 由上图可知,环境因素好像并不是造成疫情严重程度的很大因素,因为一样颜色的点没有聚集在一起.当然也有可能是数据太少了,或者数据处理的方式有问题.因为我是把湿度,降水等用温度

**工作时的报错记录**

电脑萤幕的截图

描述已自动生成

我在尝试下载kaggle的数据的时候, 尝试pip install kaggle 但是下载不了

可能的原因是需要管理员打开 . 也可能是之前路径没有 配好 path

82

**做作业时获得的一些经验**

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

我在尝试用飞桨上传数据集的时候,发现上传zip 的话他自动展开了.也就是说 zip对他是友好的, 总之上传rar 的话,没办法打开,没办法yum 解压rar的包,所以rar是打不开的,zip是原装有的

import pystan.\_api # stanc wrapper

ImportError: DLL load failed: 找不到指定的模块。

这个错误的时候,我查到是要把anaconda 的 library的 bin目录放在path里面,放在path之后要重新打开cmd,他才知道最新的path

262

**具体的代码细节**

D:\proj\visualization\bigwork\ame\_sta.py

front\_cnt=4

*# 具体的碰到的问题,其实是一个小细节,就是怎么取得df的前面几列*

*# 虽然很简单的问题 但是网上查不到. 我这方面也没什么基础,所以其实是查了大量资料,到处找才找到的*

*# 网上虽然 用列索引取出列 这种操作是挺多的 但是用数字取出 却比较少见*

*# 但是最终还是找到了*

*# https://blog.csdn.net/houyanhua1/article/details/87809185*

*# 应该类似这样找*

*# print(df.iloc[1:3,1:3])*

*# 这个东西挺花费时间的*

df= df.iloc[:,:front\_cnt]

*# df 取出前面4列*

*# df 取出第一列*

*# 至于为什么要取出前面几列 是因为 数据太多了 要是所有都拿来作图的话 图会有很多 这样大家之间都不好分辨*

*# 因此是分开几个作图*

*autumn 的效果*

图表

描述已自动生成

*# 这个东西是设置日历图的渐变颜色,但是一开始不知道去哪里找这个参数*

*# 网上找到的好像也没有合适这张图的,因为他是确诊人数,所以我觉得他从红色变到绿色有点不好*

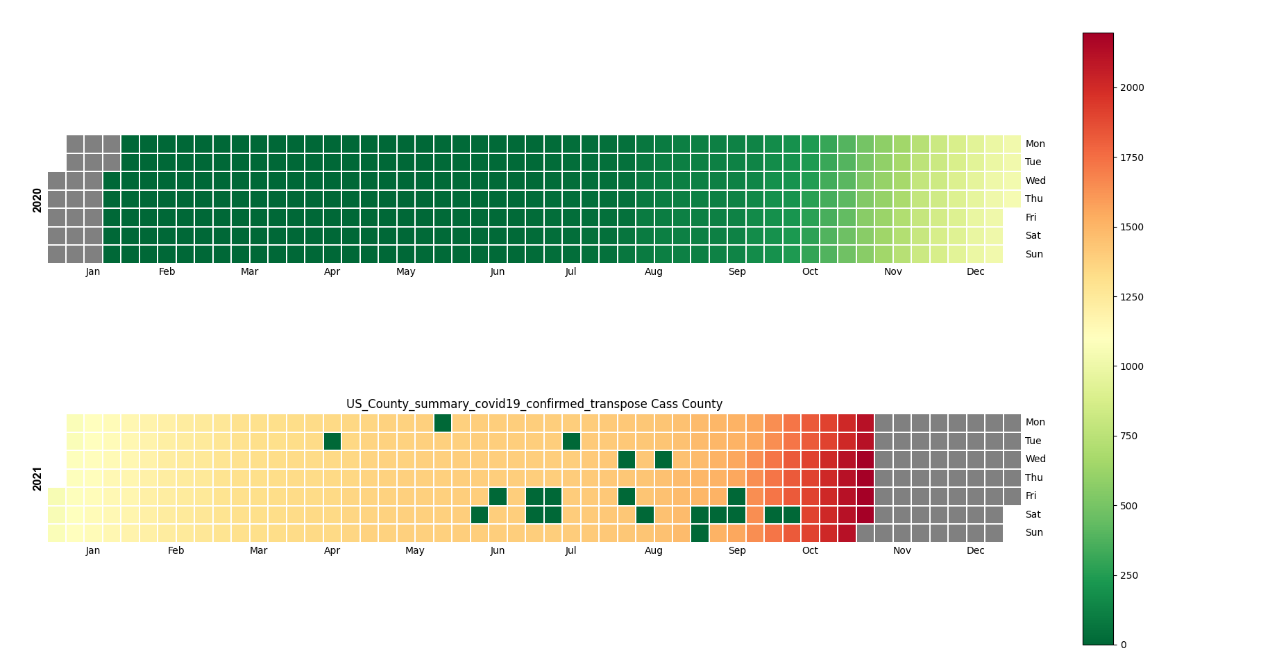
*# 网上查到有autumn,试了一下,效果也不是太好*

*# 最后在写了一个错误的参数的时候,他报错了*

*# f"{val!r} is not a valid value for {key}; "*

*# ValueError: 'reds' is not a valid value for name; supported values are 'Accent', 'Accent\_r', 'Blues', 'Blues\_r', 'BrBG', 'BrBG\_r', 'BuGn', 'BuGn\_r', 'BuPu', 'BuPu\_r', 'CMRmap', 'CMRmap\_r', 'Dark2', 'Dark2\_r', 'GnBu', 'GnBu\_r', 'Greens', 'Greens\_r', 'Greys', 'Greys\_r', 'OrRd', 'OrRd\_r', 'Oranges', 'Oranges\_r', 'PRGn', 'PRGn\_r', 'Paired', 'Paired\_r', 'Pastel1', 'Pastel1\_r', 'Pastel2', 'Pastel2\_r', 'PiYG', 'PiYG\_r', 'PuBu', 'PuBuGn', 'PuBuGn\_r', 'PuBu\_r', 'PuOr', 'PuOr\_r', 'PuRd', 'PuRd\_r', 'Purples', 'Purples\_r', 'RdBu', 'RdBu\_r', 'RdGy', 'RdGy\_r', 'RdPu', 'RdPu\_r', 'RdYlBu', 'RdYlBu\_r', 'RdYlGn', 'RdYlGn\_r', 'Reds', 'Reds\_r', 'Set1', 'Set1\_r', 'Set2', 'Set2\_r', 'Set3', 'Set3\_r', 'Spectral', 'Spectral\_r', 'Wistia', 'Wistia\_r', 'YlGn', 'YlGnBu', 'YlGnBu\_r', 'YlGn\_r', 'YlOrBr', 'YlOrBr\_r', 'YlOrRd', 'YlOrRd\_r', 'afmhot', 'afmhot\_r', 'autumn', 'autumn\_r', 'binary', 'binary\_r', 'bone', 'bone\_r', 'brg', 'brg\_r', 'bwr', 'bwr\_r', 'cividis', 'cividis\_r', 'cool', 'cool\_r', 'coolwarm', 'coolwarm\_r', 'copper', 'copper\_r', 'cubehelix', 'cubehelix\_r', 'flag', 'flag\_r', 'gist\_earth', 'gist\_earth\_r', 'gist\_gray', 'gist\_gray\_r', 'gist\_heat', 'gist\_heat\_r', 'gist\_ncar', 'gist\_ncar\_r', 'gist\_rainbow', 'gist\_rainbow\_r', 'gist\_stern', 'gist\_stern\_r', 'gist\_yarg', 'gist\_yarg\_r', 'gnuplot', 'gnuplot2', 'gnuplot2\_r', 'gnuplot\_r', 'gray', 'gray\_r', 'hot', 'hot\_r', 'hsv', 'hsv\_r', 'inferno', 'inferno\_r', 'jet', 'jet\_r', 'magma', 'magma\_r', 'nipy\_spectral', 'nipy\_spectral\_r', 'ocean', 'ocean\_r', 'pink', 'pink\_r', 'plasma', 'plasma\_r', 'prism', 'prism\_r', 'rainbow', 'rainbow\_r', 'seismic', 'seismic\_r', 'spring', 'spring\_r', 'summer', 'summer\_r', 'tab10', 'tab10\_r', 'tab20', 'tab20\_r', 'tab20b', 'tab20b\_r', 'tab20c', 'tab20c\_r', 'terrain', 'terrain\_r', 'turbo', 'turbo\_r', 'twilight', 'twilight\_r', 'twilight\_shifted', 'twilight\_shifted\_r', 'viridis', 'viridis\_r', 'winter', 'winter\_r'*

*于是我就知道了,这些参数就是他可以用的,然后看到很多后面都有个\_r ,猜测是 reversed 的意思,也就是反转,那么就简单了,一开始是 红色到绿色,现在我给他来个反转* cmap=**"RdYlGn\_r" , 不就是从绿色变成红色了吗**



plt.title(title)

plt.show()

*# 一个小细节是 plt.title 放在 plt.show()后面的话 title 就不显示了*

*# plt.title(title)*

**try**:

*# https://stackoverflow.com/questions/57105747/modulenotfounderror-no-module-named-plotly-graph-objects/57112843*

*# import plotly.graph\_objects as go*

*# import plotly.express as px*

*# 这里的一个小问题是 版本问题 plotly.graph\_objects 可能不能直接引入 ,需要 from plotly import graph\_objs as go, 这都是特定版本的引入方法*

*# 虽然说是小问题,但是一下子也没找到解决方法,是多方查找,才找到stackoverflow的一个回答,我个人认为一个比较好的方案是用try except 来尝试import*

*# 这样以后这个代码如果给别人跑了 或者我换了个环境,他至少可以尝试两种导入方法,导入的成功率会增高很多*

*# 假如我直接换成 from plotly import graph\_objs as go 这样子导入了,但是下次我换了个环境,我又需要重新写成import plotly.graph\_objects as go了*

*# 这是比较麻烦的,特别是我可能会忘记要这么修改 那我需要重新再去查各种资料,效率就低了*

*# 这一点我是觉得脚本语言比较好的一点 他可以尝试各种import ,因为他的类没有固定,他import了之后,这些类是可以在代码里跑的,但是java这种就过不了编译*

*# 对于java 我就不太清楚 怎么去try catch import 以解决依赖的问题,不过java里好像比较少碰到这种问题*

**import** plotly.express **as** px

**import** plotly.graph\_objects **as** go

**except** ImportError **as** e:

**from** plotly **import** graph\_objs **as** go

**from** plotly **import** express **as** px

在这个文件 写了几个工具

D:\proj\visualization\bigwork\util.py

*# 因为x轴如果数据太多的话 他们都会挤在一起,所以写了个函数,间隔几个再显示x轴上的标记*

**def** show\_lbls(ax,lbl\_show\_step):

i=0

**for** label **in** ax.get\_xticklabels():

**if** i==lbl\_show\_step:

i=0

**continue**

label.set\_visible(**False**)

i+=1

*# 同样的 折线图上面标记的数字 也是要隔几个显示一下 不然就叠在一起,不过具体要隔几个 还是要慢慢调出来*

*# 这里写了一个函数 方便调试和调用*

**def** show\_txt(x,y,show\_txt\_step,plt):

i=0

down=0.5

*# down=0.03*

**for** a,b **in** zip(x,y):

**if** i==show\_txt\_step:

*# 可以考虑 在相近的 x y 的地方 不要再重复放置 text*

plt.text(a,b-down,b,ha=**'center'**,va=**'bottom'**,fontsize=11)

i=0

i+=1

*# 因为作图可能是做很多个折线图 就可能他们的x都是一样的,然后分别是好几个列来和这个x做图*

*# 于是写了一个函数 传入cols 是列名的列表,可以让x 和这些列来做折线图,同时可以在折线图上显示数字*

**def** plot\_lst(x,cols,df,show\_txt\_step,plt):

**for** col **in** cols:

*# df.iloc*

plt.plot(x, df[col],label=col)

*# plt.plot(x, df[col],label=col)*

y=df[col]

show\_txt(x,y,show\_txt\_step,plt)

D:\proj\visualization\bigwork\全球3.py

*# 关于肘部法则*

**def** ElbowMethod(X):

wcss = []

sil = []

**for** i **in** range(2, 11):

*# 分类的个数 去尝试 每种尝试 发现 2 3 会好一些*

*# 再根据层次聚类图 我们认为 选择分成3类比较好*

clf = KMeans(n\_clusters=i, init=**'k-means++'**, random\_state=42)

*# 分类的个数是2..11 我们都去尝试一下,fit 一下,看一下的出来的结果,画个图,直观一点*

*# 看看那种方案好点*

clf.fit(X)

labels = clf.labels\_

centroids = clf.cluster\_centers\_

sil.append(silhouette\_score(X, labels, metric=**'euclidean'**))

wcss.append(clf.inertia\_)

x = np.arange(2, 11)

plt.figure(figsize=(10, 5))

plt.plot(x, wcss, marker=**'o'**)

plt.xlabel(**"Number of Clusters 集群的个数 "**)

*# 集群;群集;*

plt.ylabel(**"Within Cluster Sum of Squares (WCSS) 簇内平方和"**)

*# 簇内平方和（WCSS）*

plt.title(**"Elbow Method 肘部法则"**)

*# –Elbow Method和轮廓...*

*# 肘部法则*

plt.show()

*# https://blog.csdn.net/qq\_41185868/article/details/89226991*

*# No handles with labels found to put in legend.*

*# 在放legend 的时候 首先画图要有label*

*D:\proj\visualization\bigwork\获得高维数据.py*

*# 因为数据集给的数据都是一天天的 而且只有一个特定的属性 比如降水量*

*# 这样不是很合适数据分析 所以我写了一个代码 用来合并这些数据为一个表格*

*# 过程也挺艰辛的 花了一上午 主要就是df 的 append ,会返回一个df*

*# 要把这个df 重新复制 ,他不是替换的*

*# 在处理国家地理数据的合并的时候 比较恶心的地方是nan merge 之后有些数据是nan的 需要去掉,但是直接dropna去不掉*

*# df\_merged.dropna(thresh=3,axis="index",replace=True)*

*# 就是这样 没有什么用处*

*# 于是只能自己写 dropna 函数*

*# 但是他nan的数据多种多样,可能是空字符串 可能是np.nan ,np里面的nan 有也有多种 np.nan np.NaN,np.NAN ,*

*# 也不知道都是些什么,这边在drop的时候调试了很久, 弄了快一个下午,最后没办法一个函数drop掉*

*# 只能先过滤到一部分,然后再 df\_merged.dropna(subset=['Clusters'],inplace=True) ,于是这样是成功的*

**def** is\_nan(val):

*# np.nan*

*# np.nan*

**return** val == np.nan **or** val **is None or** val **is** np.nan \

**or** val **is** np.NAN **or** val == np.NAN **or** val **is** np.nan \

**or** val == np.nan **or** val == **"NaN" or** val == **"nan" or** val==**""** \

**or** val ==np.NaN **or** val **is** np.NaN