世界疫情地图

张元博

2020-02

1 数据读取及处理

首先要导入数据，本次使用的数据是 Github 上一个项目里的，也可以直接用 pandas 包导入，需要注意的是不能直接使用 Github 那个网址，否则会报错，需要将前面部分改成 https://raw.githubusercontent.com/，然后就是加入数据的目录地址。数据主要是是三个文件，包含了疫情的确诊数 (confirmed)，治愈数 (recoved)，死亡数 (deaths)，基本上每日会更新最新疫情数据。

```
import pandas as pd
import numpy as np

confirmed = pd.read_csv('./data/confirmed.csv',index_col=0)
recovered = pd.read_csv('./data/recovered.csv',index_col=0)
deaths = pd.read_csv('./data/deaths.csv',index_col=0)
```

用 head(5) 是查看数据前五行；confirmed 表里面包含发生疫情的国家，经纬度，以及从 2020 年 1 月 22 日至今的每日的确诊数；recovered 表则记录了治愈数；deaths 表则记录了死亡数。(以下表格部分数据作保留两位小数处理)

```
confirmed.head(5)
```

Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	...	2/28/20
Anhui	Mainland China	31.83	117.23	1	9	...	990
Beijing	Mainland China	40.18	116.41	14	22	...	410
Chongqing	Mainland China	30.06	107.87	6	9	...	576
Fujian	Mainland China	26.08	117.99	1	5	...	296
Gansu	Mainland China	36.06	103.83	0	2	...	91

```
recovered.head(5)
```

Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	...	2/28/20
Anhui	Mainland China	31.83	117.23	0	0	...	821
Beijing	Mainland China	40.18	116.41	0	0	...	257
Chongqing	Mainland China	30.06	107.87	0	0	...	422
Fujian	Mainland China	26.08	117.99	0	0	...	235
Gansu	Mainland China	36.06	103.83	0	0	...	82

```
deaths.head(5)
```

Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	...	2/28/20
Anhui	Mainland China	31.83	117.23	0	0	...	6
Beijing	Mainland China	40.19	116.41	0	0	...	7
Chongqing	Mainland China	30.06	107.87	0	0	...	6
Fujian	Mainland China	26.08	117.99	0	0	...	1
Gansu	Mainland China	36.06	103.83	0	0	...	2

使用 shape 则可以打印出三个数据表的维度。每个都是 (114, 42) 维，即 114 行，42 列。(从网站读入数据，数据源会更新，维数可能不一样)

```
print(confirmed.shape)
```

```
print(recovered.shape)
```

```
print(deaths.shape)
```

2 数据可视化

前面我们已经搞清楚了数据，下面进行数据可视化。这部分主要用到 matplotlib 这个绘图包，至于绘制地图，放在第三部分讲。

```
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']  = ['SimHei']    #用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False    #用来正常显示负号
```

首先看看在我们的数据中，哪些地区发生了疫情。可以看出一共 49 个地区都有新冠肺炎病例。

```
countries = confirmed['Country/Region'].unique()
print(countries)
```

接下来看看世界疫情发展趋势，我们的数据还需要再整理下，要计算出每日所有地区新冠肺炎的确诊数，治愈数，死亡数。

```
all_confirmed = np.sum(confirmed.iloc[:,4:])
all_recovered = np.sum(recovered.iloc[:,4:])
all_deaths = np.sum(deaths.iloc[:,4:])
```

现在数据就变成下面这样，包含每天的总数据

```
all_confirmed
```

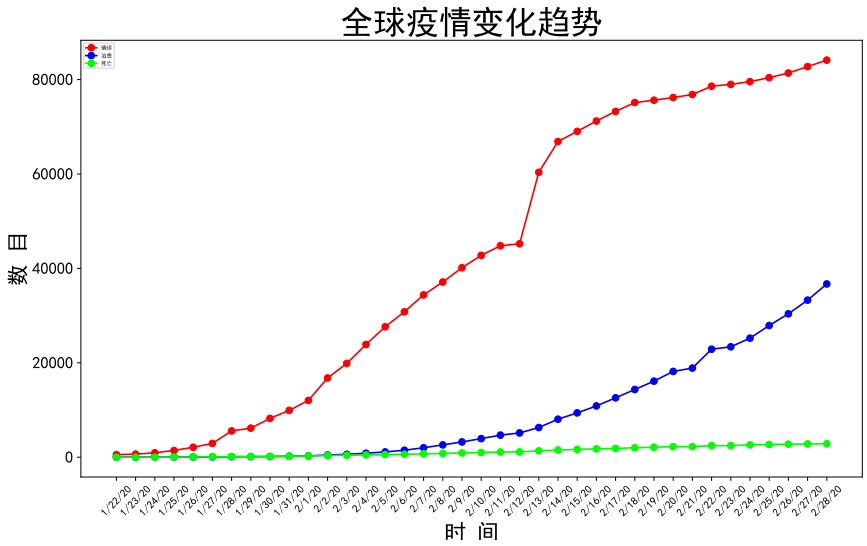
日期	1/22/20	1/23/20	1/24/20	1/25/20	...	2/26/20	2/27/20	2/28/20
确诊数	555	653	941	1434	...	81397	82756	84124

下面就可以画出疫情发展趋势图了。

```
plt.figure(figsize=(12.8,7.2))
plt.plot(all_confirmed,color = 'red',label = '确诊',marker = 'o')
plt.plot(all_recovered,color = 'blue',label = '治愈',marker = 'o')
plt.plot(all_deaths,color = 'lime',label = '死亡',marker = 'o')
plt.xticks(rotation = 45,size = 10)
```

```
plt.yticks(size=15)
```

```
plt.xlabel('时 间',size = 20)
plt.ylabel('数 目',size = 20)
plt.title('全球疫情变化趋势',size = 30)
plt.legend(loc = "upper left",fontsize = 5)
plt.show()
```



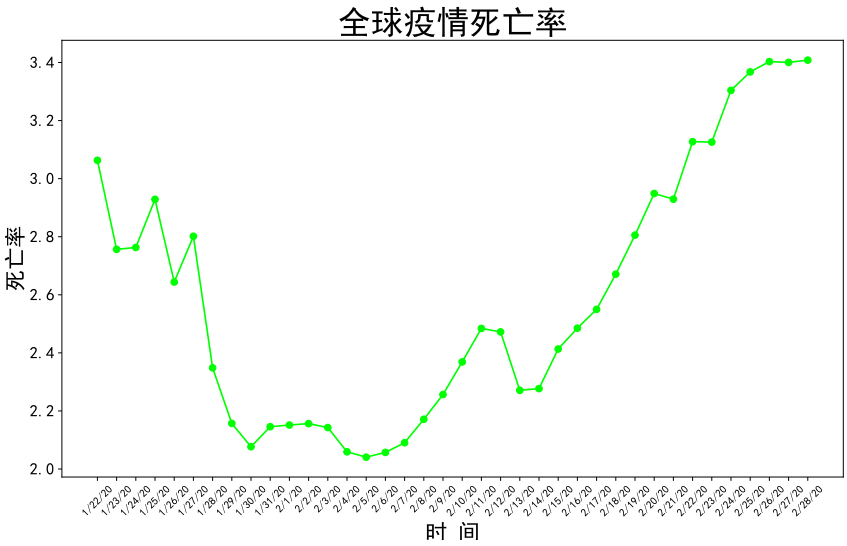
可以看出，目前新冠肺炎确诊病例还在持续增加，不过令人高兴的是治愈数也在持续增长，死亡数很少，希望疫情拐点早日出现，疫情早日结束。

下面看看新冠肺炎的死亡率，首先计算死亡率数据，然后就可以直接画图。

```
death_rate = (all_deaths/all_confirmed)*100
plt.figure(figsize=(12.8,7.2))
plt.plot(death_rate,color = 'lime',label = '死亡',marker = 'o')
plt.xticks(rotation = 45,size = 10)
```

```
plt.yticks(size = 15)
```

```
plt.xlabel('时 间',size = 20)
plt.ylabel('死亡率',size = 20)
plt.title('全球疫情死亡率',size = 30)
```



由于本次疫情主要发生在中国大陆，下面来具体研究下中国大陆的疫情情况，首先从全部数据中提取出中国大陆的数据。里面包含了省份，以及每个省最新的确诊数，治愈数，死亡数。

```
last_update = '2/28/20'  # 设置最新数据日期
China_cases = confirmed[['Province/State',last_update]][confirmed['
    Country/Region'] == 'Mainland China']
China_cases['recovered']=recovered[[last_update]][recovered['Country/
    Region']=='Mainland China']
China_cases['deaths']=deaths[[last_update]][deaths['Country/Region']=='
    Mainland China']
China_cases = China_cases.set_index('Province/State')
China_cases = China_cases.rename(columns = {last_update:'confirmed'})
```

Province/State	deaths	confirmed	recovered
Anhui	989	744	6
Beijing	400	235	4
Chongqing	576	384	6
Fujian	294	218	1
Gansu	91	81	2
Guangdong	1347	851	7
Guangxi	252	147	2
Guizhou	146	104	2
Hainan	168	129	5
Hebei	312	261	6
Heilongjiang	480	249	12

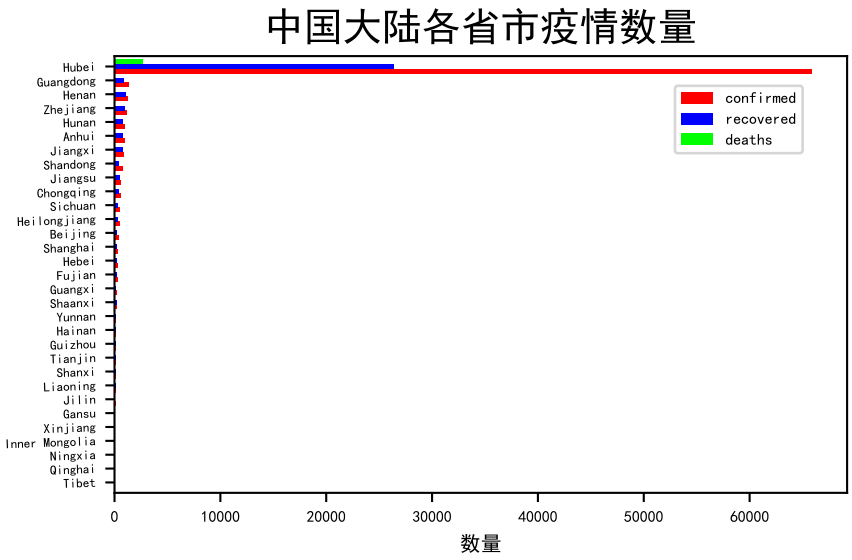
下面画出中国大陆每个省份的疫情数量图。

```
Mainland_china = China_cases.sort_values(by='confirmed',ascending=True)
plt.figure(figsize=(50,90))
```

```
Mainland_china.plot(kind='barh', color = ['red','blue','lime'], width=1,
    rot=2)
plt.title('中国大陆各省市疫情数量', size=15)
plt.ylabel('省/市',size=8)
plt.xlabel('数量',size = 8)
plt.yticks(size=5)
```

```
plt.xticks(size=6)
```

```
plt.legend(bbox_to_anchor=(0.95,0.95),fontsize = 6)
```



可以看到，湖北省三项数据高居第一位，且远远高于其他省份。下面看看中国大陆的治愈率和死亡率数据，数据使用下面的代码即可计算出来，最终结果在 recover_rate 和 death_rate 里。

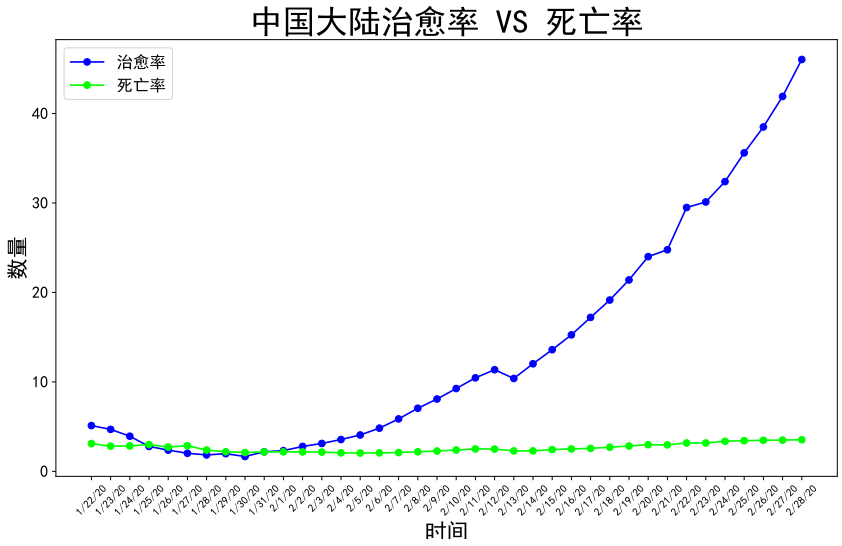
```
confirmed_china = confirmed[confirmed['Country/Region']=='Mainland China
    ']
confirmed_china = np.sum(confirmed_china.iloc[:,4:])
recovered_china = recovered[recovered['Country/Region'] == 'Mainland
    China']
recovered_china = np.sum(recovered_china.iloc[:,4:])
deaths_china = deaths[deaths['Country/Region'] == 'Mainland China']
deaths_china = np.sum(deaths_china.iloc[:,4:])
recover_rate = (recovered_china/confirmed_china)*100
death_rate = (deaths_china/confirmed_china)*100
```

接下来就是画图了

```
plt.figure(figsize=(12.8,7.2))
plt.plot(recover_rate, color = 'blue', label = '治愈率', marker = 'o')
plt.plot(death_rate, color = 'lime', label = '死亡率', marker = 'o')
plt.title('中国大陆治愈率 VS 死亡率',size=30)
plt.ylabel('数量',size=20)
plt.xlabel('时间',size=20)
plt.xticks(rotation=45,size=10)
```

```
plt.yticks(size=15)
```

```
plt.legend(loc = "upper left",fontsize = 15)
```



虽然在 1 月 25 日-1 月 31 日期间死亡率略高于治愈率，但其他时间段，治愈率远远高于死亡率，这都得益于全国广大医务人员的不懈努力!

那中国大陆其他地区这一情况咋样呢？代码大同小异，我们一起来看看，首先还是提取出其他地区的数据。

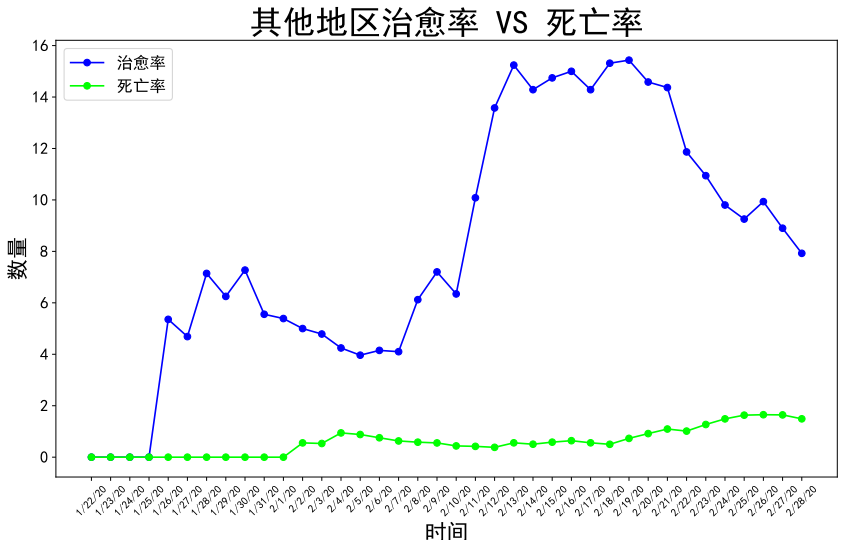
```
confirmed_others = confirmed[confirmed['Country/Region'] != 'Mainland
    China']
confirmed_others = np.sum(confirmed_others.iloc[:,4:])
recovered_others = recovered[recovered['Country/Region'] != 'Mainland
    China']
recovered_others = np.sum(recovered_others.iloc[:,4:])
deaths_others = deaths[deaths['Country/Region'] != 'Mainland China']
deaths_others = np.sum(deaths_others.iloc[:,4:])
recover_rate = (recovered_others/confirmed_others)*100
death_rate = (deaths_others/confirmed_others)*100
```

然后就是画图了，我们一起来看看吧。

```
plt.figure(figsize=(12.8,7.2))
plt.plot(recover_rate, color = 'blue', label = '治愈率', marker = 'o')
plt.plot(death_rate, color = 'lime', label = '死亡率', marker = 'o')
plt.title('其他地区治愈率 VS 死亡率',size=30)
plt.ylabel('数量',size=20)
plt.xlabel('时间',size=20)
plt.xticks(rotation=45,size=10)
```

```
plt.yticks(size=15)
```

```
plt.legend(loc = "upper left",fontsize = 15)
```



接下来看看其他地区疫情数量。首先还是提出其他地区的数据

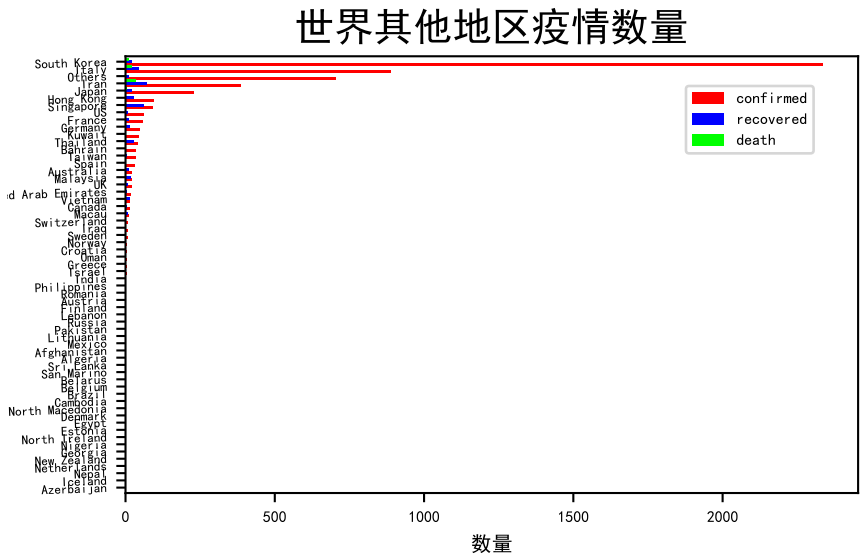
```
others = confirmed[['Country/Region',last_update]][confirmed['Country/
    Region'] != 'Mainland China']
others['recovered'] = recovered[[last_update]][recovered['Country/Region
    '] != 'Mainland China']
others['death'] = deaths[[last_update]][deaths['Country/Region'] != '
    Mainland China']
others_countries = others.rename(columns = {last_update:'confirmed'})
others_countries = others_countries.set_index('Country/Region')
others_countries = others_countries.groupby('Country/Region').sum()
```

然后画图

```
plt.figure(figsize=(50,90))
others_countries.sort_values(by = 'confirmed',ascending = True).plot(
    kind='barh',  color = ['red','blue','lime'], width=1, rot=2)
plt.title('世界其他地区疫情数量', size=15)
plt.ylabel('Country/Region',size = 8)
plt.xlabel('数量',size = 8)
plt.yticks(size=5)
```

```
plt.xticks(size=6)
```

```
plt.legend(bbox_to_anchor=(0.95,0.95),fontsize =6)
```



从图可以看到，韩国，意大利，日本这些地区也有很多新冠肺炎患者。

3 绘制疫情地图

想必看到这，大家已经厌烦了前面的折线图，柱状图了，下面笔者将教大家如何画出疫情地图，将每个数据对应到地图上听起来是不是很好玩呢？

这里主要用到两个 python 包，一个是 folium 包，这个包也是笔者最近才发现的绘图包，类似于 R 语言绘图里的 ggplot2，可以添加图层来定义一个 Map 对象，最后以几种方式将 Map 对象展现出来。这里有一个详细教程，感兴趣的可以看看 https://python-visualization.github.io/folium/。另一个包就是 plotly 了，这也是一个强大的绘图包，详细教程请看这里 https://plot.ly/python/plotly-express/。

首先是 folium 包绘制地图，import folium，只需要导入包就可以了，没下载这个包的记得下载才能使用。我们在前面数据里加入中国大陆的数据，并使用武汉的经纬度。

```
others=confirmed[['Country/Region','Lat','Long',last_update]][confirmed[
    'Country/Region'] != 'Mainland China']
others['recovered'] = recovered[[last_update]][recovered['Country/Region
    '] != 'Mainland China']
others['death'] = deaths[[last_update]][deaths['Country/Region'] != '
    Mainland China']
others_countries = others.rename(columns = {last_update:'confirmed'})
others_countries.loc['94'] = ['Mainland China',30.9756,112.2707,
    confirmed_china[-1],recovered_china[-1],deaths_china[-1]]
```

下面是我们用于画地图的数据格式

Country/Region	Lat	Long	confirmed	recovered	death
Thailand	15.0000	101.0000	40	22	0
Japan	36.0000	138.0000	189	22	2
South Korea	36.0000	128.0000	1261	22	12

Country/Region	Lat	Long	confirmed	recovered	death
Taiwan	23.7000	121.0000	32	5	1
US	47.6062	-122.3321	1	1	0

然后开始正式构建地图

```
import folium
world_map = folium.Map(location=[10, -20], zoom_start=2.3,tiles='Stamen
    Toner')
```

上面一行是定义一个 world_map 对象；location 的格式为 [纬度, 经度]；zoom_start 表示初始地图的缩放尺寸，数值越大放大程度越大；tiles 为地图类型，用于控制绘图调用的地图样式，默认为'OpenStreetMap'，也有一些其他的内建地图样式，如'Stamen Terrain'、'Stamen Toner'、'Mapbox Bright'、'Mapbox Control Room' 等；也可以传入'None' 来绘制一个没有风格的朴素地图，或传入一个 URL 来使用其它的自选 osm。

然后往 world_map 里添加其他元素，注意这里的 for 循环和最后的 add_to 是把经纬度点的信息一个一个的加进去

```
for lat, lon, value, name in zip(others_countries['Lat'], others_
    countries['Long'], others_countries['confirmed'], others_countries['
    Country/Region']):
    folium.CircleMarker([lat, lon],
                        radius=10,
                        popup = ('<strong>Country</strong>: ' + str(
    name).capitalize() + '<br>'
                        '<strong>Confirmed Cases</strong>: ' + str(
    value) + '<br>'),
                        color='red',
                        fill_color='red',
                        fill_opacity=0.7 ).add_to(world_map)
```

这里主要说下 popup 参数 popup：str 型或 folium.Popup() 对象输入，用于控制标记部件的具体样式（folium 内部自建了许多样式），默认为 None，即不显示部件。代码使用的是自定义的网页样式，其中表示加粗，表示换行，以便将各个数据显示出来。

然后再运行 world_map，即可出现下面的地图样式，这是一种可交互的地图，可以随意移动缩放，鼠标点击地图上红点，即可出现地区的疫情信息。

```
world_map
```

下面我再教大家用 plotly 绘制每日疫情扩散地图，首先是导入包

```
import plotly.express as px
```

在这里，我想绘制每日疫情扩散地图，我需要增加一列，里面记录了每天的日期，因此我们的数据还需要再重新整理下，这里需要用的 melt 函数，它将列名转换为列数据 (columns name → column values)，重构 DataFrame，

```
confirmed = confirmed.melt(id_vars = ['Province/State', 'Country/Region'
    , 'Lat', 'Long'], var_name='date',value_name = 'confirmed')
```

主要参数说明：- id_vars: 不需要被转换的列名。- value_vars: 需要转换的列名，如果剩下的列全部都要转换，就不用写了。- var_name 和 value_name 是自定义设置对应的列名。

重新得到的数据如下，新增了 date 一列，记录时间。

```
confirmed
```

还需要把 date 列转换成 datetime 格式的数据

```
confirmed['date_dt'] = pd.to_datetime(confirmed.date, format="%m/%d/%y")
confirmed.date = confirmed.date_dt.dt.date
confirmed.rename(columns={'Country/Region': 'country', 'Province/State':
    'province'}, inplace=True)
```

最终 confirmed 的数据如下格式

province	country	Lat	Long	date	confirmed	date_dt
Anhui	Mainland China	31.83	117.23	2020-01-22	1	2020-01-22
Beijing	Mainland China	40.18	116.41	2020-01-22	14	2020-01-22
Chongqing	Mainland China	30.06	107.87	2020-01-22	6	2020-01-22
Fujian	Mainland China	26.08	117.99	2020-01-22	1	2020-01-22
Gansu	Mainland China	36.06	103.83	2020-01-22	0	2020-01-22

同理整理出治愈数据和死亡数据。

```
recovered = recovered.melt(id_vars = ['Province/State', 'Country/Region'
```

```
    , 'Lat', 'Long'], var_name='date',value_name = 'recovered')
recovered['date_dt'] = pd.to_datetime(recovered.date, format="%m/%d/%y")
recovered.date = recovered.date_dt.dt.date
recovered.rename(columns={'Country/Region': 'country', 'Province/State':
     'province'}, inplace=True)
deaths = deaths.melt(id_vars = ['Province/State', 'Country/Region', 'Lat
    ', 'Long'], var_name='date', value_name = 'deaths')
deaths['date_dt'] = pd.to_datetime(deaths.date, format="%m/%d/%y")
deaths.date = deaths.date_dt.dt.date
deaths.rename(columns={'Country/Region': 'country', 'Province/State': '
    province'}, inplace=True)
```

现在三种数据都有了，我们把它们合并在一张表里面，主要用到 merge 函数

```
merge_on = ['province', 'country', 'date']
all_date = confirmed.merge(deaths[merge_on + ['deaths']], how='left', on
    =merge_on). \
                        merge(recovered[merge_on + ['recovered']], how='
    left', on=merge_on)

del all_date['date']
```

最终我们得到的数据就是下面这种

province	country	Lat	Long	confirmed	date_dt	deaths	recovered
Anhui	Mainland China	31.83	117.23	1	2020-01-22	0	0
Beijing	Mainland China	40.18	116.41	14	2020-01-22	0	0
Fujian	Mainland China	26.08	117.99	1	2020-01-22	0	0
Gansu	Mainland China	36.06	103.83	0	2020-01-22	0	0

由于要演示的是疫情扩散地图，因此笔者用实心圆来表示每个地区的疫情变化，而实心圆的大小则代表了三种数据的大小，所以在我们的数据里要加一列，使用 confirmed 数据的二分之一次方来表示实心圆的大小。

```
Coronavirus_map = all_date.groupby(['date_dt', 'province'])['confirmed',
     'deaths','recovered', 'Lat', 'Long'].max().reset_index()
Coronavirus_map['size'] = Coronavirus_map.confirmed.pow(0.5)  # 创建实心圆
    大小
Coronavirus_map['date_dt'] = Coronavirus_map['date_dt'].dt.strftime('%Y
    -%m-%d')
```

最终数据就变成下面这样啦。

date_dt	province	confirmed	deaths	recovered	Lat	Long	size
2020-01-22	Montreal, QC	0	0	0	45.50	-73.57	0.00
2020-01-22	Anhui	1	0	0	31.83	117.23	1.00
2020-01-22	Beijing	14	0	0	40.18	116.41	3.74
2020-01-22	Boston, MA	0	0	0	42.36	-71.06	0.00

最后就是绘图部分，代码也很简单，如果有不懂得参数可以使用help(px.scatter_geo) 来查看每个参数用法

```
fig = px.scatter_geo(Coronavirus_map, lat='Lat', lon='Long', scope='asia
    ',
                     color="size", size='size', hover_name='province',
                     hover_data=['confirmed', 'deaths', 'recovered'],
                     projection="natural earth",animation_frame="date_dt
    ",title='亚洲地区疫情扩散图')
fig.update(layout_coloraxis_showscale=False)
```

```
fig.show()
```

好了地图就画好了，代码运行完就会出现picture的样式，点击播放按钮，就会动态变化。

本文到这里就结束了，希望这次疫情也能早早结束，武汉加油！中国加油！人类加油！