全球抗“疫”：用 Python 带你了解世界疫情

王梦圆

2020-02

1 数据的读取及处理

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）。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。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as tkr
confirmed = pd.read_csv('../data/COVID-20/csse_covid_19_data/csse_covid_19_time_series/time_series_19-covid-Confirmed.csv')
recovered = pd.read_csv('../data/COVID-20/csse_covid_19_data/csse_covid_19_time_series/time_series_19-covid-Recovered.csv')
deaths = pd.read_csv('../data/COVID-20/csse_covid_19_data/csse_covid_19_time_series/time_series_19-covid-Deaths.csv')
```

数据已经导入了，让我们来看看数据是啥样的吧。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。

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

Province/State	Country/Region	Lat	long	1/22/20	1/23/20	...
Anhui	Mainland China	31.8257	117.2264	1	9	...
Beijing	Mainland China	40.1824	116.4142	14	22	...
Chongqing	Mainland China	30.0572	107.8740	6	9	...
Fujian	Mainland China	26.0789	117.9874	1	5	...
Gansu	Mainland China	36.0611	103.8343	0	2	...

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

Province/State	Country/Region	Lat	long	1/22/20	1/23/20	...
Anhui	Mainland China	31.8257	117.2264	0	0	...
Beijing	Mainland China	40.1824	116.4142	0	0	...
Chongqing	Mainland China	30.0572	107.8740	0	0	...
Fujian	Mainland China	26.0789	117.9874	0	0	...

Province/State	Country/Region	Lat	long	1/22/20	1/23/20	...
Gansu	Mainland China	36.0611	103.8343	0	0	...

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

Province/State	Country/Region	Lat	long	1/22/20	1/23/20	...
Anhui	Mainland China	31.8257	117.2264	0	0	...
Beijing	Mainland China	40.1824	116.4142	0	0	...
Chongqing	Mainland China	30.0572	107.8740	0	0	...
Fujian	Mainland China	26.0789	117.9874	0	0	...
Gansu	Mainland China	36.0611	103.8343	0	0	...

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

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

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

2 数据可视化

```
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

countries = confirmed['Country/Region'].unique()
print(countries)#可以看出一共75个国家/地区都有新冠状肺炎病例

#计算出每日所有地区新冠肺炎的确诊数，治愈数，死亡数
```

```
fig,ax = plt.subplots()
all_confirmed = np.sum(confirmed.iloc[:,4:])
all_recovered = np.sum(recovered.iloc[:,4:])
all_deaths = np.sum(deaths.iloc[:,4:])
all_confirmed.index = [d[:-3] for d in all_confirmed.index]
all_recovered.index = [d[:-3] for d in all_recovered.index]
all_deaths.index = [d[:-3] for d in all_deaths.index]
ax.plot(all_confirmed,color = 'red',label = '确诊',marker = 'o',
    linewidth=1,markersize=3)
ax.plot(all_recovered,color = 'blue',label = '治愈',marker = 'o',
    linewidth=1,markersize=3)
ax.plot(all_deaths,color = 'lime',label = '死亡',marker = 'o',linewidth
    =1,markersize=3)
ax.xaxis.set_major_locator(tkr.MultipleLocator(2.0))
ax.xaxis.set_minor_locator(tkr.MultipleLocator(1.0))
plt.xticks(rotation = 45)
```

```
plt.yticks()
```

```
plt.xlabel('时间')
plt.ylabel('数目')
plt.legend(loc = 'upper left',fontsize = 10)
```

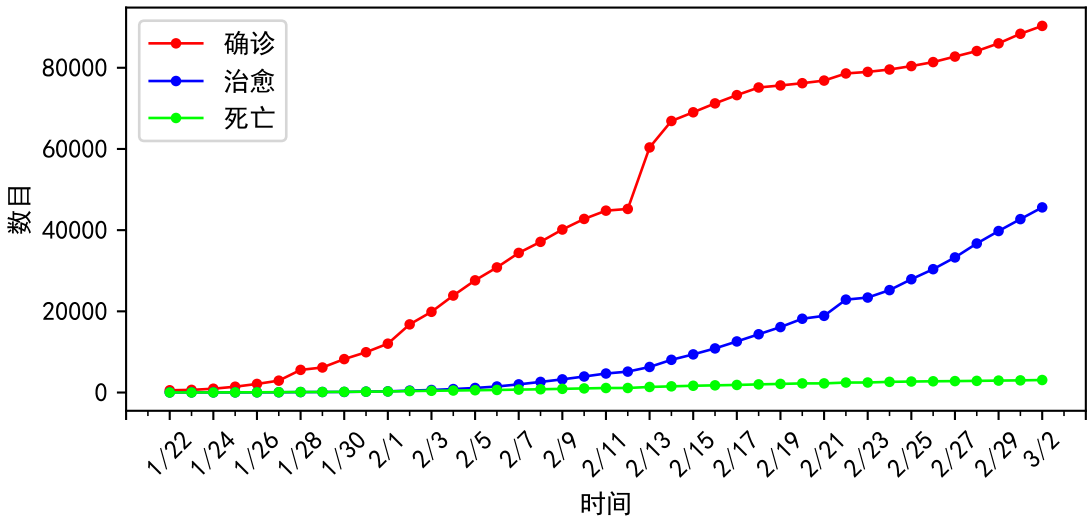

图 1　全球疫情变化趋势

```
plt.tight_layout()
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

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

```
fig,ax = plt.subplots()
death_rate = (all_deaths/all_confirmed)*100
death_rate
```

```
ax.plot(death_rate,color = 'lime',label = '死亡',marker = 'o',linewidth
    =1,markersize=3)
ax.xaxis.set_major_locator(tkr.MultipleLocator(2.0))
ax.xaxis.set_minor_locator(tkr.MultipleLocator(1.0))
plt.xticks(rotation = 45)
```

```
plt.yticks()
```

```
plt.xlabel('时间')
plt.ylabel('死亡率')
plt.tight_layout()
plt.show()
```

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 = '3/2/20' #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')
```

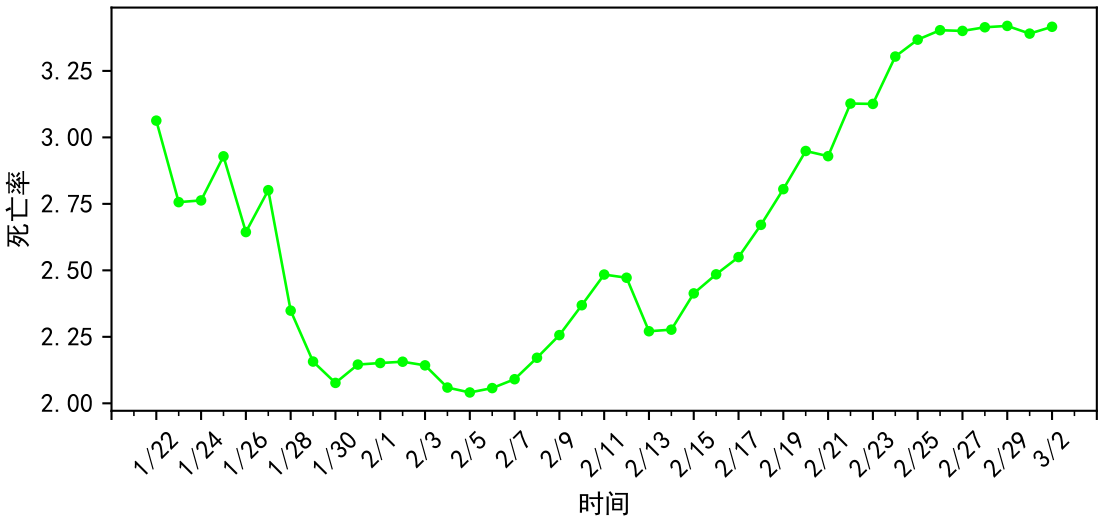

图 2　全球疫情死亡率

```
China_cases = China_cases.rename(columns = {last_update:'confirmed'})
China_cases
```

```
China_cases.to_csv('./Chinacases.csv')
```

表 4　中国大陆各省疫情数据

省份	确诊数	治愈数	死亡数
Anhui	990	917	6
Beijing	414	282	8
Chongqing	576	469	6
Fujian	296	255	1
Gansu	91	85	2
Guangdong	1350	1059	7
Guangxi	252	192	2
Guizhou	146	114	2
Hainan	168	151	5
Hebei	318	296	6
Heilongjiang	480	356	13
Henan	1272	1205	22
Hubei	67103	33934	2803
Hunan	1018	887	4
Inner Mongolia	75	54	0
Jiangsu	631	543	0
Jiangxi	935	850	1
Jilin	93	83	1
Liaoning	122	103	1
Ningxia	74	69	0
Qinghai	18	18	0

表 4　中国大陆各 (续)

省份	确诊数	治愈数	死亡数
Shaanxi	245	216	1
Shandong	758	460	6
Shanghai	337	292	3
Shanxi	133	119	0
Sichuan	538	386	3
Tianjin	136	111	3
Tibet	1	1	0
Xinjiang	76	66	3
Yunnan	174	168	2
Zhejiang	1206	1069	1

下面画出中国大陆每个省份的疫情数量图

```
Mianland_China = China_cases.sort_values(by='confirmed',ascending=True)
Mianland_China.plot(kind='barh',figsize=(20,30),color = ['red','blue','
    lime'],width = 1)
plt.xlabel('省/市',size = 30)
plt.ylabel('数量',size = 30)
plt.xticks(size = 25)
```

```
plt.yticks(size = 30)
```

```
plt.legend(bbox_to_anchor = (0.95,0.95),fontsize = 20)
plt.tight_layout()
plt.show()
```

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  #中国地区的治愈率
deaths_rate = (deaths_China/confirmed_China)*100#中国各地区的死亡率
deaths_rate.index = [d[:-3] for d in deaths_rate.index]
deaths_rate
```

```
recover_rate.index = [d[:-3] for d in recover_rate.index]
recover_rate
#接下来画图
```

```
fig,ax = plt.subplots()
```

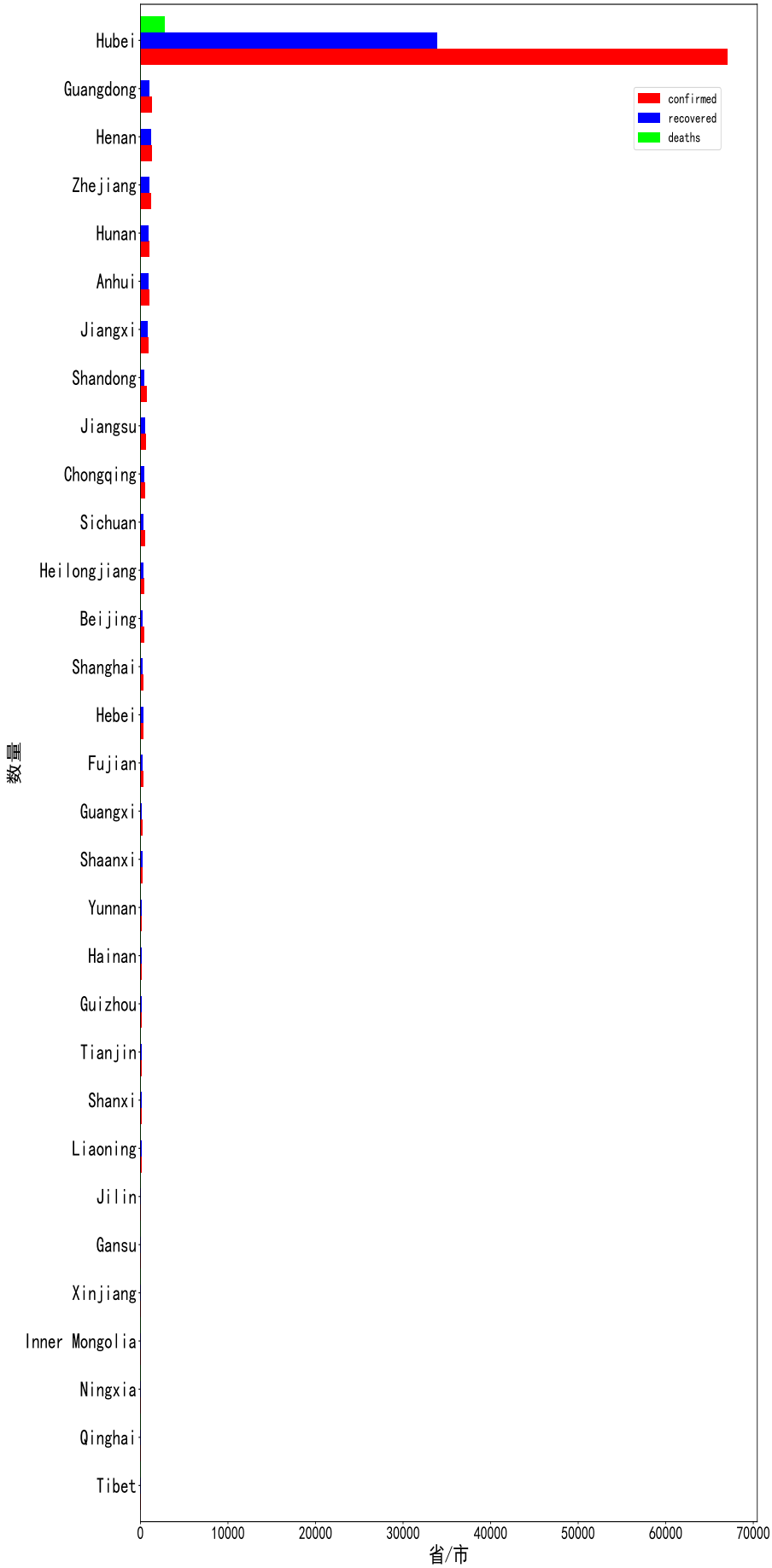

图 3　中国大陆各省市疫情数量

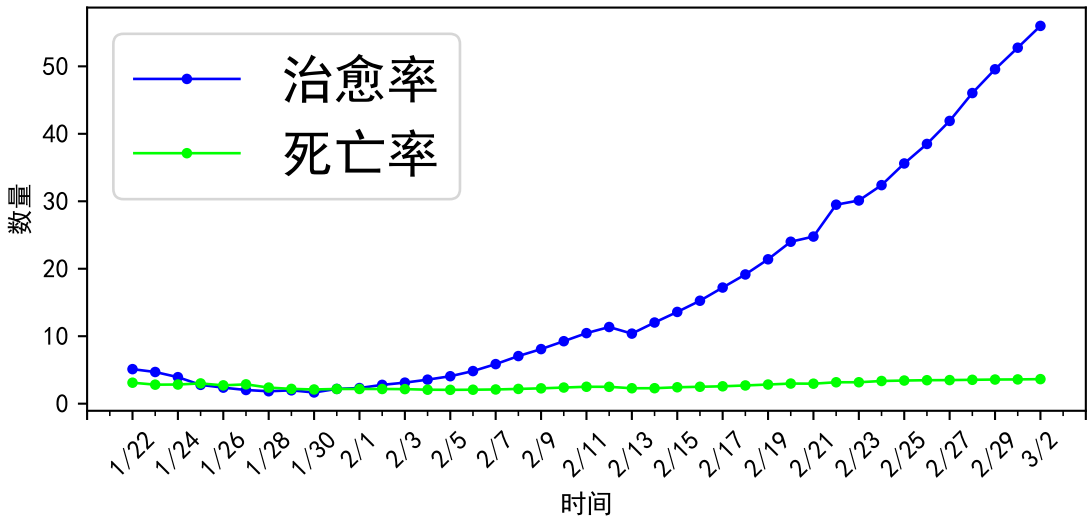

图 4　中国大陆治愈率 VS 死亡率

```
ax.plot(recover_rate,color = 'blue',label = '治愈率',marker = 'o',
    linewidth=1,markersize=3)
ax.plot(deaths_rate,color = 'lime',label = '死亡率',marker = 'o',
    linewidth=1,markersize=3)
ax.xaxis.set_major_locator(tkr.MultipleLocator(2.0))
ax.xaxis.set_minor_locator(tkr.MultipleLocator(1.0))
plt.xlabel('时间')
plt.ylabel('数量')
plt.xticks(rotation = 45)
```

```
plt.yticks()
```

```
plt.legend(loc = 'upper left',fontsize = 20)
plt.tight_layout()
plt.show()
```

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

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_others = (recovered_others/confirmed_others)*100  #其他地区的
    治愈率
deaths_rate_others = (deaths_others/confirmed_others)*100#其他各地区的死亡率
#接下来画图
fig,ax = plt.subplots()
recover_rate_others.index = [d[:-3] for d in recover_rate_others.index]
```

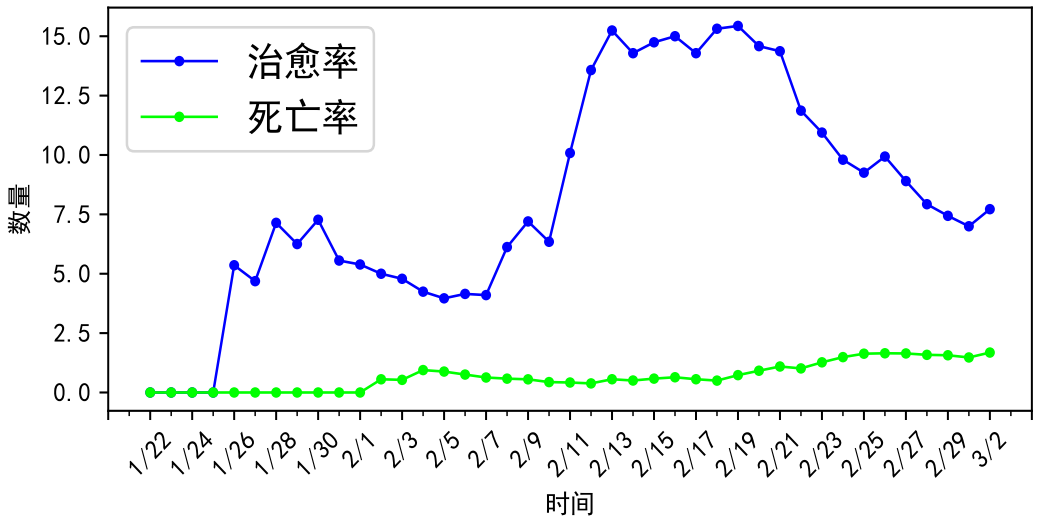

图 5　其他地区治愈率 VS 死亡率

```
deaths_rate_others.index = [d[:-3] for d in deaths_rate_others.index]
ax.plot(recover_rate_others,color = 'blue',label = '治愈率',marker = 'o'
    ,linewidth=1,markersize=3)
ax.plot(deaths_rate_others,color = 'lime',label = '死亡率',marker = 'o',
    linewidth=1,markersize=3)
ax.xaxis.set_major_locator(tkr.MultipleLocator(2.0))
ax.xaxis.set_minor_locator(tkr.MultipleLocator(1.0))
plt.xlabel('时间')
plt.ylabel('数量')
plt.xticks(rotation = 45)
```

```
plt.yticks()
```

```
plt.legend(loc = 'upper left',fontsize=15)
plt.tight_layout()
plt.show()
```

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['deaths'] = 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()
#接着画图
others_countries.sort_values(by = 'confirmed',ascending = True).plot(
    kind='barh',figsize=(20,30),color = ['red','blue','lime'], width=1)
plt.ylabel('Country/Region')
plt.xlabel('数量')
plt.yticks()
```

```
plt.xticks()
```

```
plt.legend(bbox_to_anchor=(0.95,0.95),fontsize = 20)
plt.tight_layout()
plt.show()
```

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

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。

```
import folium
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[others_countries.index==94]
```

```
others_countries.loc['94'] = ['Mainland China',30.9756,112.2707,
    confirmed_China[-1],recovered_China[-1],deaths_China[-1]]
others_countries
```

```
others_countries.to_csv("./otherscountries.csv")
```

然后开始正式构建地图。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

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

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,
```

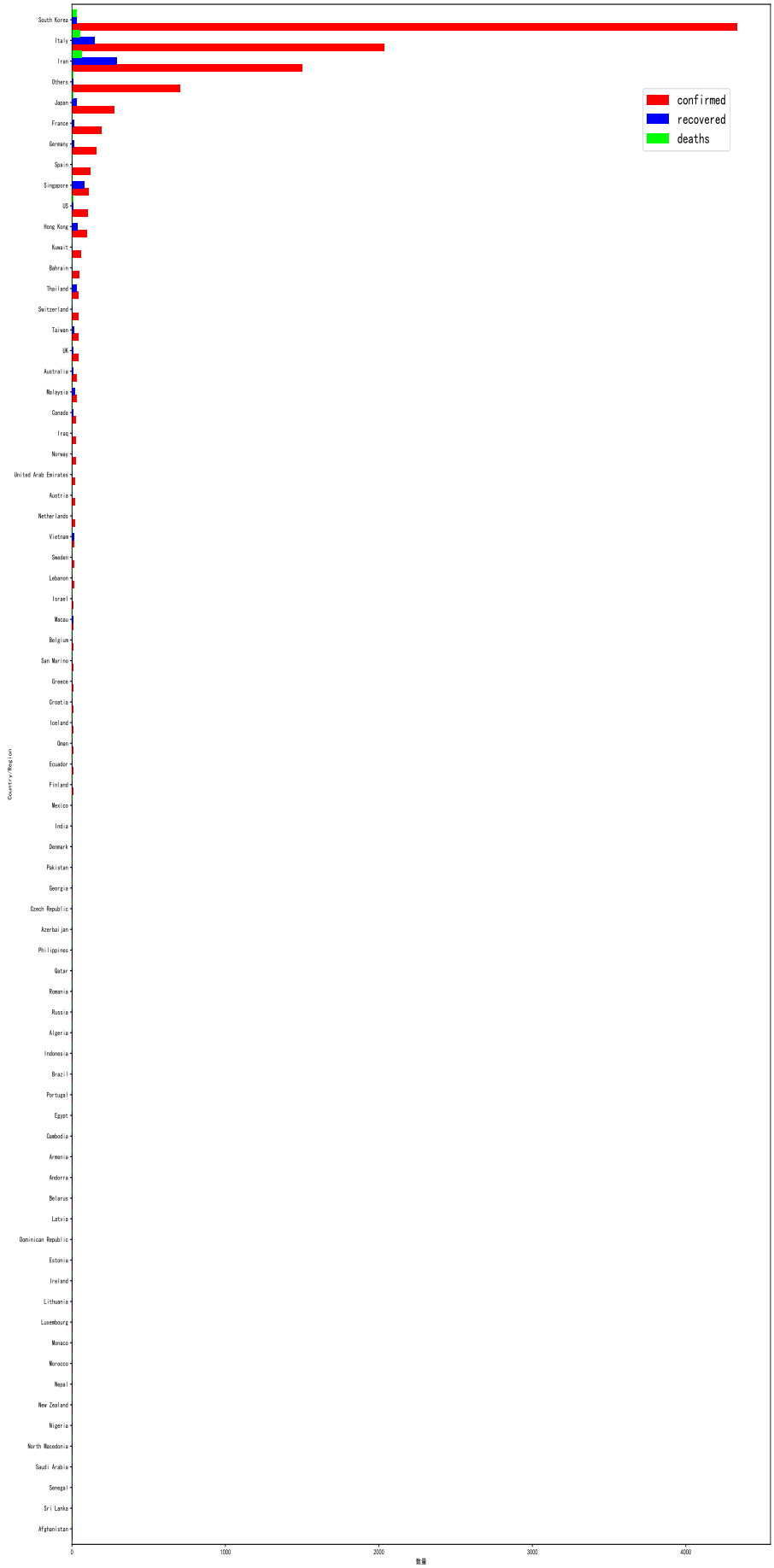

图 6　世界其他地区疫情数量

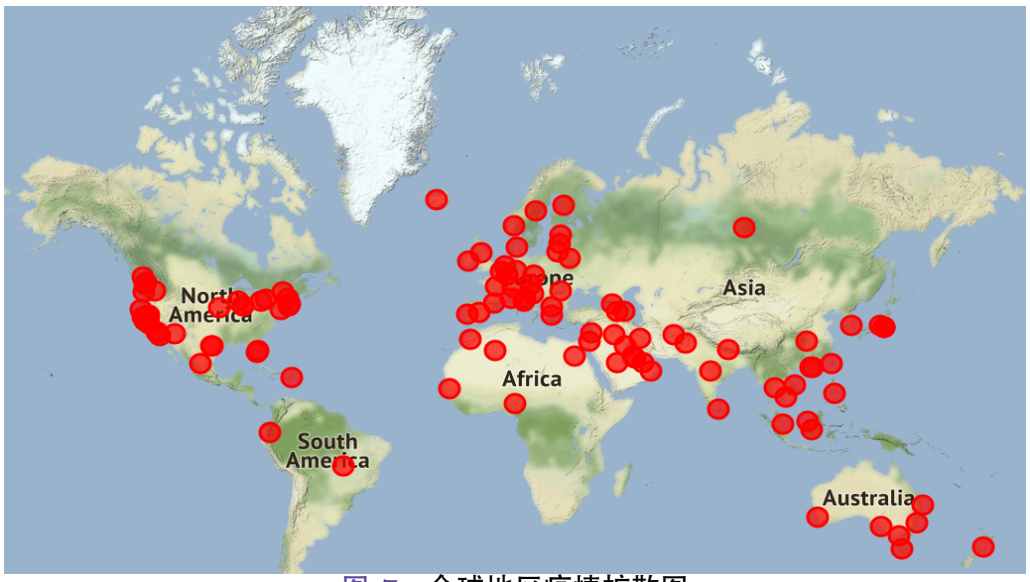

图 7　全球地区疫情扩散图

```
                                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)
```

```
world_map
#world_map.save("worldmap.html")
#import webbrowser
#webbrowser.open("worldmap.html")
```

用 plotly 绘制每日疫情扩散地图

```
import plotly.express as px
confirmed = confirmed.melt(id_vars = ['Province/State', 'Country/Region'
    , 'Lat', 'Long'], var_name='date',value_name = 'confirmed')
confirmed.to_csv("./confirmedmelt.csv")
```

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。

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)
confirmed
```

治愈数据

```
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
```

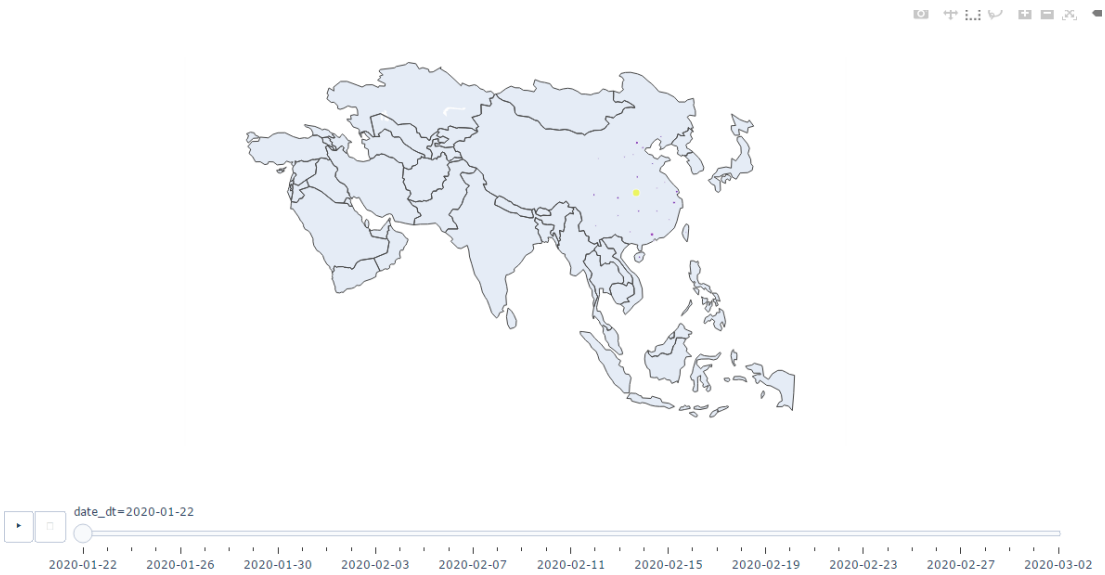

图 8　亚洲地区疫情扩散图

```
recovered.rename(columns={'Country/Region': 'country', 'Province/State':
    'province'}, inplace=True)
```

死亡数据

```
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)
```

合并数据

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

```
Coronavirus_map = all_data.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')
```

```
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
    ")
fig.update(layout_coloraxis_showscale=False)
```

```
fig.show()
```