Python 带你了解世界疫情

赵凯力

2020-02

1 数据读取和处理

首先当导入必备的包

```
import numpy as np
import pandas as pd
```

然后就是导入数据，数据已经从 github 上下载，共三个文件，分别是疫情的确诊数（confirmed），治愈数（recoved），死亡数（deaths），基本上每日会更新最新疫情数据。

```
confirmed = pd.read_csv(r'.\COVID-19\csse_covid_19_data\csse_covid_19_
    time_series\time_series_19-covid-Confirmed.csv')
recovered = pd.read_csv(r'.\COVID-19\csse_covid_19_data\csse_covid_19_
    time_series\time_series_19-covid-Recovered.csv')
deaths = pd.read_csv(r'.\COVID-19\csse_covid_19_data\csse_covid_19_time_
    series\time_series_19-covid-Deaths.csv')
```

数据已经导入, 接下来查看数据的基本情况。head() 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()
```

表 1　**confirmed 的前五行**

Province/State	Country/Region	Lat	Long	1/22/20	...	3/01/20
Anhui	Mainland China	31.83	117.2	1	...	990
Beijing	Mainland China	40.18	116.4	14	...	413
Chongqing	Mainland China	30.06	107.9	6	...	576
Fujian	Mainland China	26.08	118.0	1	...	296
Gansu	Mainland China	36.06	103.8	0	...	91

```
recovered.head()
```

表 2　**recovered 的前五行**

Province/State	Country/Region	Lat	Long	1/22/20	...	3/01/20
Anhui	Mainland China	31.83	117.2	0	...	873
Beijing	Mainland China	40.18	116.4	0	...	276
Chongqing	Mainland China	30.06	107.9	0	...	450
Fujian	Mainland China	26.08	118.0	0	...	247
Gansu	Mainland China	36.06	103.8	0	...	84

```
deaths.head()
```

表 3　deaths 的前五行

Province/State	Country/Region	Lat	Long	1/22/20	...	3/01/20
Anhui	Mainland China	31.83	117.2	0	...	6
Beijing	Mainland China	40.18	116.4	0	...	8
Chongqing	Mainland China	30.06	107.9	0	...	6
Fujian	Mainland China	26.08	118.0	0	...	1
Gansu	Mainland China	36.06	103.8	0	...	2

2　数据可视化

导入 matplotlib 绘图包画图

```
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 61 个地区都有新冠肺炎病例。

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:])
#np.sum()函数传入数据框默认对每一列求和，结果是series
all_recovered = np.sum(recovered.iloc[:,4:])
all_deaths = np.sum(deaths.iloc[:,4:])
```

下面就可以画出疫情发展趋势图了, 见图1。

```
plt.figure(figsize=(12, 6))
plt.plot(all_confirmed, color='r', label='确诊', marker='o')
plt.plot(all_deaths, color='lime', label='死亡', marker='o')
plt.plot(all_recovered, color='b', label='治愈', marker='o')
plt.xticks(all_confirmed.index[::2],rotation = 45,size = 13)
```

```
plt.yticks(size=20)
```

```
plt.xlabel('时 间',size = 20)
plt.ylabel('数 目',size = 20)
plt.legend(loc = "upper left",fontsize = 20)
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，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, 见图2。

```
death_rate = (all_deaths/all_confirmed)*100
plt.figure(figsize=(12, 6))
```

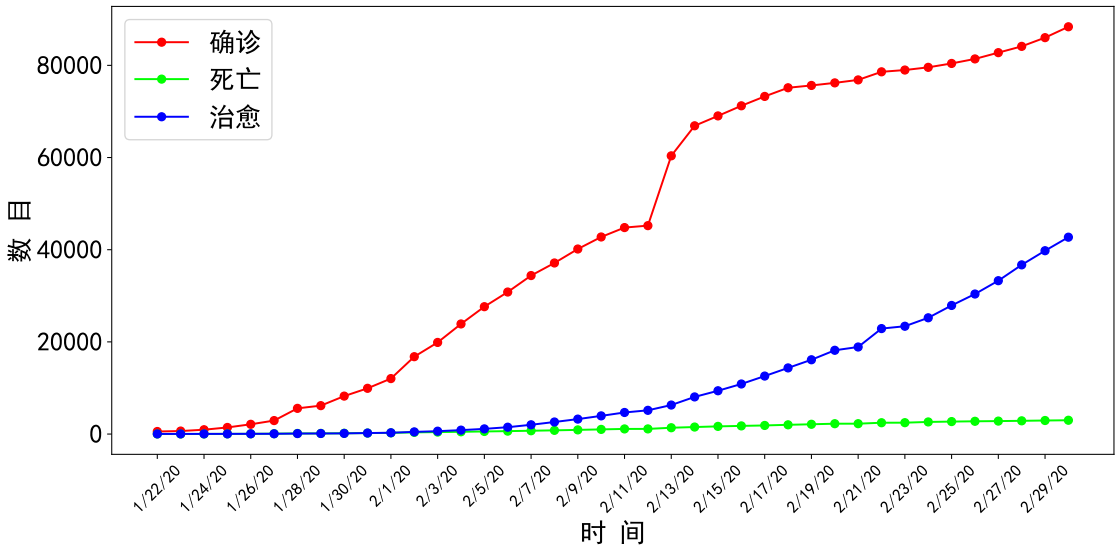

图 1　全球疫情变化趋势

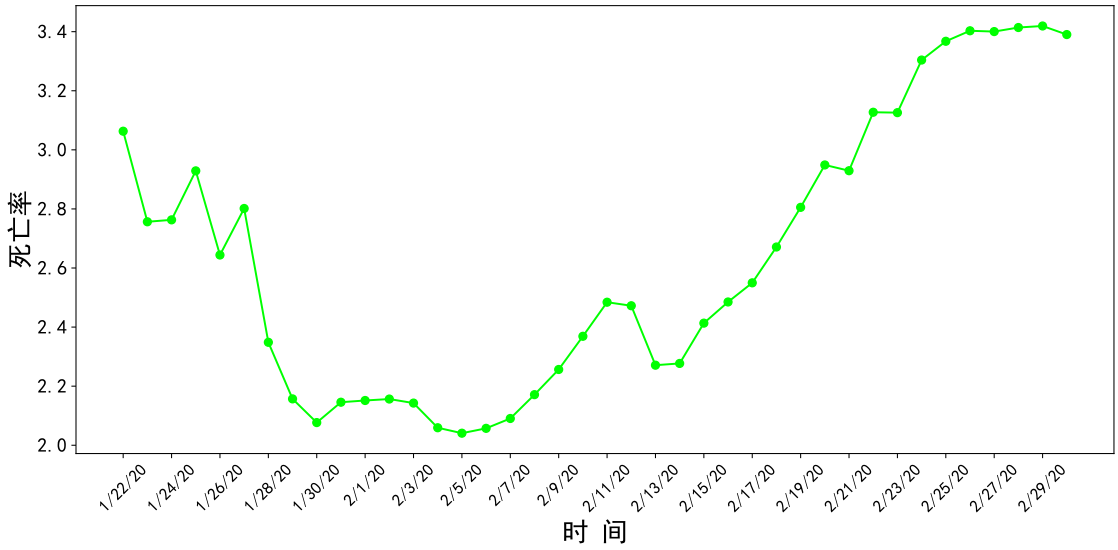

图 2　全球疫情死亡率

```
plt.plot(death_rate,color = 'lime',label = '死亡',marker = 'o')
plt.xticks(all_confirmed.index[::2],rotation = 45,size = 13)
```

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

```
plt.xlabel('时 间',size = 20)
plt.ylabel('死亡率',size = 20)
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 = '3/01/20'  # 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.set_index('Province/State', inplace=True)
China_cases = China_cases.rename(columns = {last_update:'confirmed'})
print(China_cases)
```

下面画出中国大陆每个省份的疫情数量图。(图3)

```
Mainland_china = China_cases.sort_values(by='confirmed',ascending=True)
Mainland_china.plot(kind='barh', figsize=(20,30),color = ['red','blue','
    lime'], width=1, rot=2)
plt.ylabel('省/市',size=40)
plt.xlabel('数量',size=40)
plt.yticks(size=30)
```

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

```
plt.legend(bbox_to_anchor=(0.95,0.95),fontsize = 40)
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
death_rate = (deaths_china/confirmed_china)*100
```

接下来就是画图了 (图4)

```
plt.figure(figsize=(12, 6))
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.ylabel('数量',size=15)
plt.xlabel('时间',size=15)
plt.xticks(all_confirmed.index[::2], rotation=45,size=13)
```

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

```
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，这都得益于全国广大医务人员的不懈努力！然后来看中国大陆以外的其他地区情况。

```
confirmed_others = confirmed[confirmed['Country/Region'] != 'Mainland
    China']
```

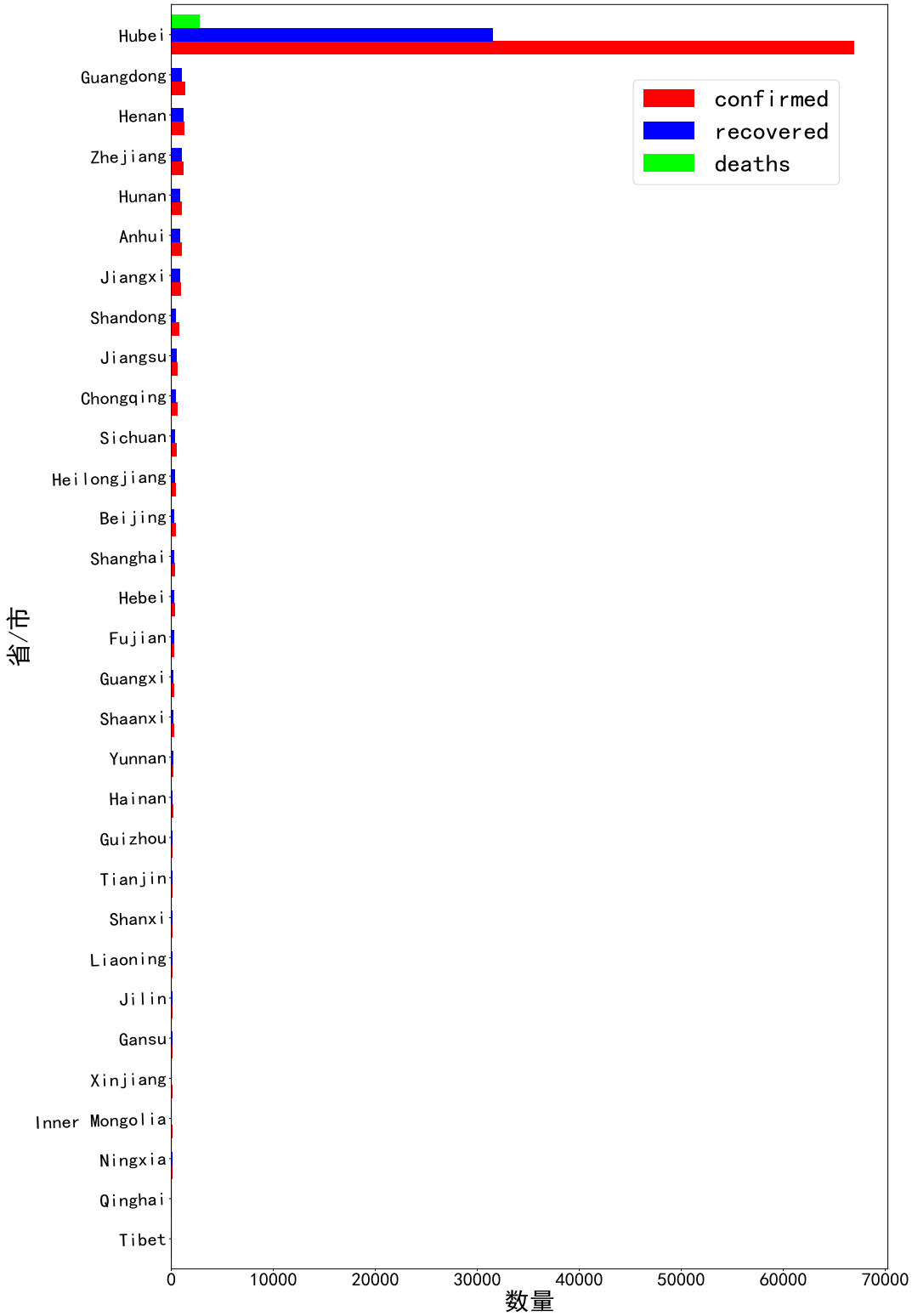

图 3　中国大陆各省市疫情数量

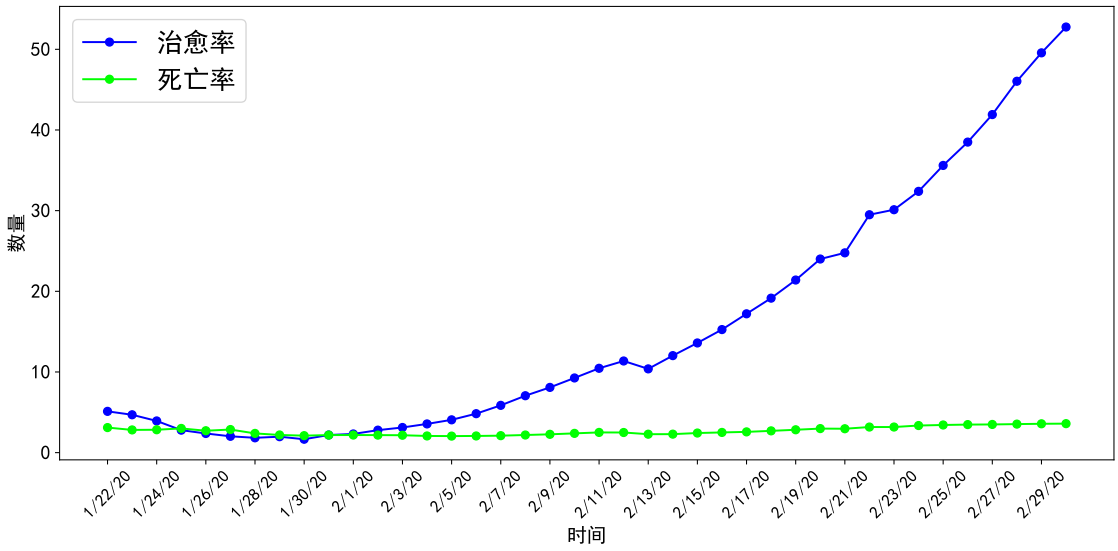

图 4　中国大陆治愈率 VS 死亡率

```
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。(图5)

```
plt.figure(figsize=(12, 6))
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.ylabel('数量',size=15)
plt.xlabel('时间',size=15)
plt.xticks(all_confirmed.index[::2], rotation=45,size=13)
```

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

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

接下来看看其他地区疫情数量。

```
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()
sorted = others_countries.sort_values(by = 'confirmed',ascending=True)
```

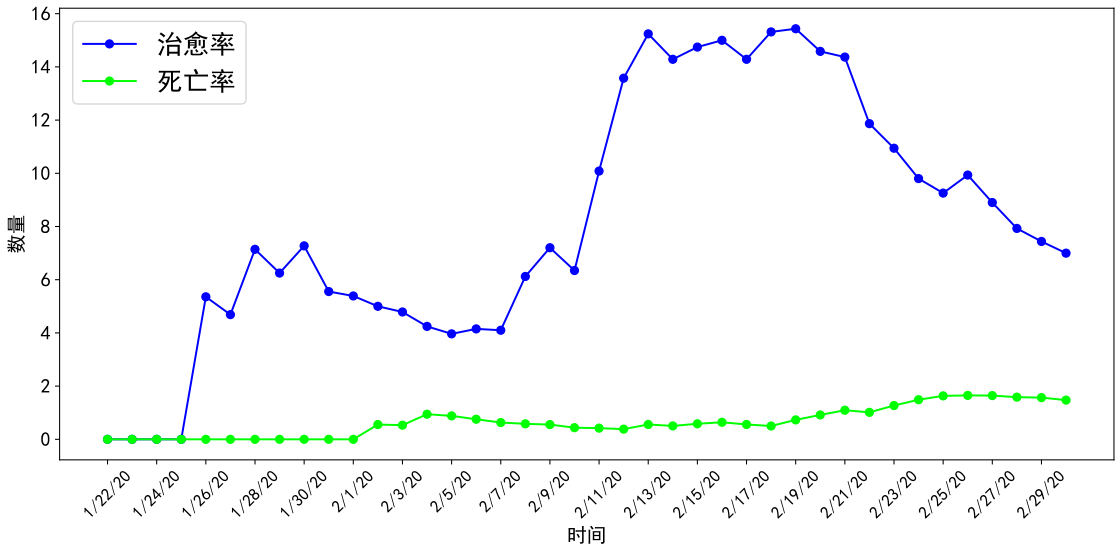

图 5　其他地区治愈率 VS 死亡率

然后画图 (图6)

```
sorted.plot(kind='barh', figsize=(20, 30),color = ['red','blue','lime'],
    width=1,rot=2)
plt.ylabel('Country/Region',size = 30)
plt.xlabel('数量',size = 30)
plt.yticks(size=30)
```

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

```
plt.legend(bbox_to_anchor=(0.95,0.95),fontsize = 30)
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

使用 folium 包绘制地图，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]]
others_countries.to_csv("./mydata/others_countries.csv")
```

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')
```

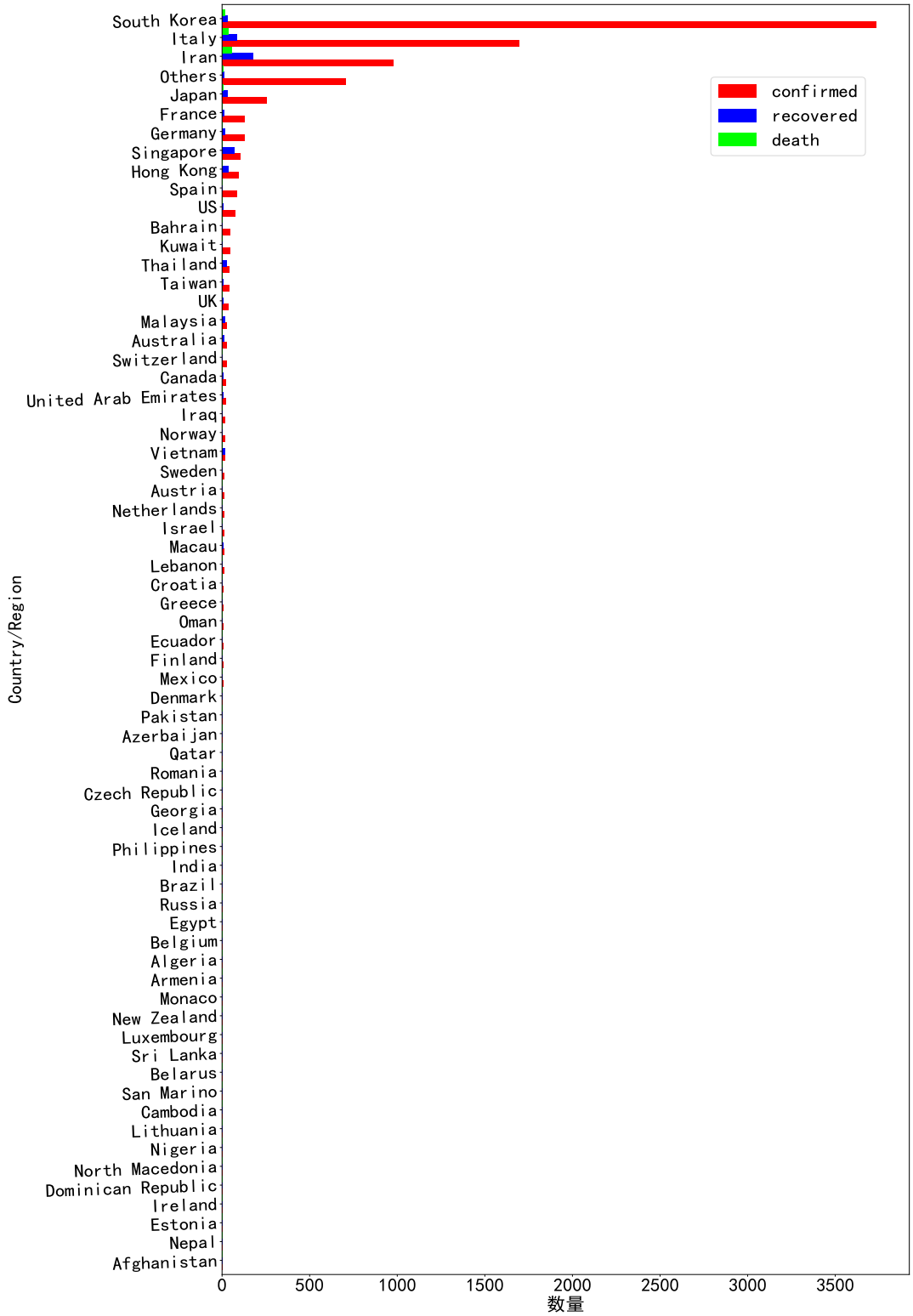

图 6　世界其他地区疫情数量

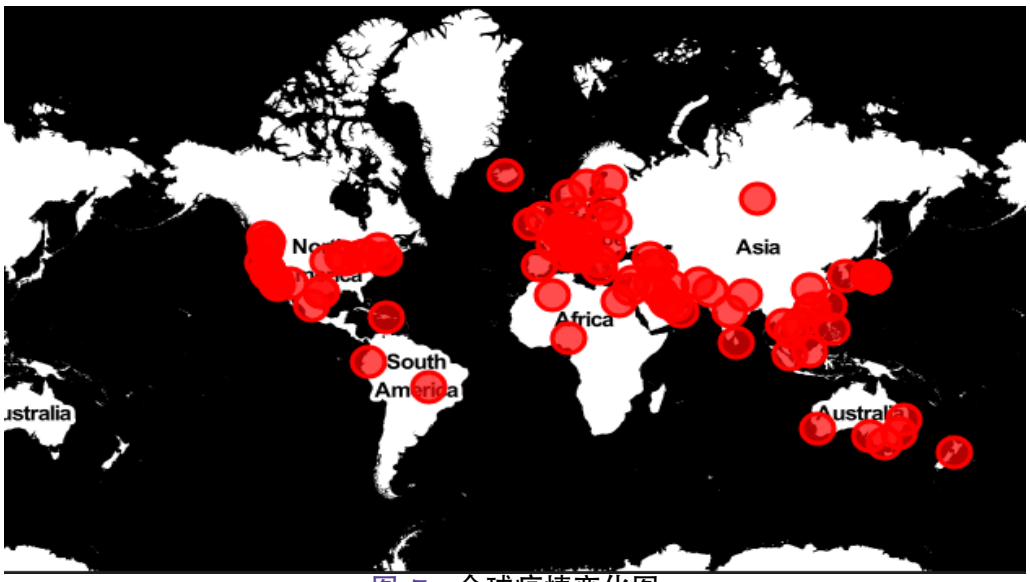

图 7　全球疫情变化图

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，即可出现如图图7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。

接下来使用 plotly 绘制每日疫情扩散地图，首先是导入包

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

想绘制每日疫情扩散地图，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 mydataFrame

```
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。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

表 4　**confirmed 前五行**

province	country	Lat	Long	date	confirmed	date_dt
Anhui	Mainland China	31.83	117.2	2020-01-22	1	2020-01-22
Beijing	Mainland China	40.18	116.4	2020-01-22	14	2020-01-22
Chongqing	Mainland China	30.06	107.9	2020-01-22	6	2020-01-22
Fujian	Mainland China	26.08	118.0	2020-01-22	1	2020-01-22
Gansu	Mainland China	36.06	103.8	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_data = confirmed.merge(deaths[merge_on + ['deaths']], how='left', on
    =merge_on).merge(recovered[merge_on + ['recovered']], how='left', on
    =merge_on)
```

由于要演示的是疫情扩散地图，因此使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

图 8　亚洲地区疫情扩散图

confirmed 数据的二分之一次方来表示实心圆的大小。

```
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')
```

最后就是绘图部分

```
fig = px.scatter_geo(Coronavirus_map, lat='Lat', lon='Long', scope='asia
    ',
                     color="size", size='size', hover_name='province',
                     hover_mydata=['confirmed', 'deaths', 'recovered'],
                     projection="natural earth",animation_frame="date_dt
    ")
fig.update(layout_coloraxis_showscale=False)
```

得到的图形如图8所示。