

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

```
1 import numpy as np
2 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，分别是疫情的确证数（confirmed），治愈数（recovered），死亡数（deaths），基本上每日会更新最新疫情数据。

```
1 confirmed = pd.read_csv('./data/time_series_19-covid-Confirmed.csv')
2 recovered = pd.read_csv('./data/time_series_19-covid-Recovered.csv')
3 deaths = pd.read_csv('./data/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；confirmed 表里面包含发生疫情的国家，经纬度，以及从 2020 年 1 月 22 日至今的每日的确证数；recovered 表则记录了治愈数；deaths 表则记录了死亡数。

```
1 confirmed.head()
```

Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	...	2/27/20	2/28/20
Anhui	Mainland China	31.8257	117.2264	1	9	...	989	990
Beijing	Mainland China	40.1824	116.4142	14	22	...	410	410
Chongqing	Mainland China	30.0572	107.8740	6	9	...	576	576
Fujian	Mainland China	26.0789	117.9874	1	5	...	296	296
Gansu	Mainland China	36.0611	103.8343	0	2	...	91	91

```
1 recovered.head()
```

Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	...	2/27/20	2/28/20
Anhui	Mainland China	31.8257	117.2264	0	0	...	792	821
Beijing	Mainland China	40.1824	116.4142	0	0	...	248	257
Chongqing	Mainland China	30.0572	107.8740	0	0	...	401	422
Fujian	Mainland China	26.0789	117.9874	0	0	...	228	235
Gansu	Mainland China	36.0611	103.8343	0	0	...	81	82

```
1 deaths.head()
```

Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	...	2/27/20	2/28/20
Anhui	Mainland China	31.8257	117.2264	0	0	...	6	6
Beijing	Mainland China	40.1824	116.4142	0	0	...	5	7
Chongqing	Mainland China	30.0572	107.8740	0	0	...	6	6
Fujian	Mainland China	26.0789	117.9874	0	0	...	1	1
Gansu	Mainland China	36.0611	103.8343	0	0	...	2	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。

```
1 print(confirmed.shape)
1 print(recovered.shape)
1 print(deaths.shape)
```

2 数据可视化

导入 `matplotlib` 绘图包画图

```
1 import matplotlib.pyplot as plt
2 plt.rcParams['font.sans-serif'] = ['SimHei'] #用来正常显示中文标签
3 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。

```
1 countries = confirmed['Country/Region'].unique()
2 print(countries)
1 countries.shape[0]
```

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。

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

```
1 all_confirmed
```

1/22/20	555
1/23/20	653
1/24/20	941
1/25/20	1434
1/26/20	2118
1/27/20	2927
1/28/20	5578
1/29/20	6166

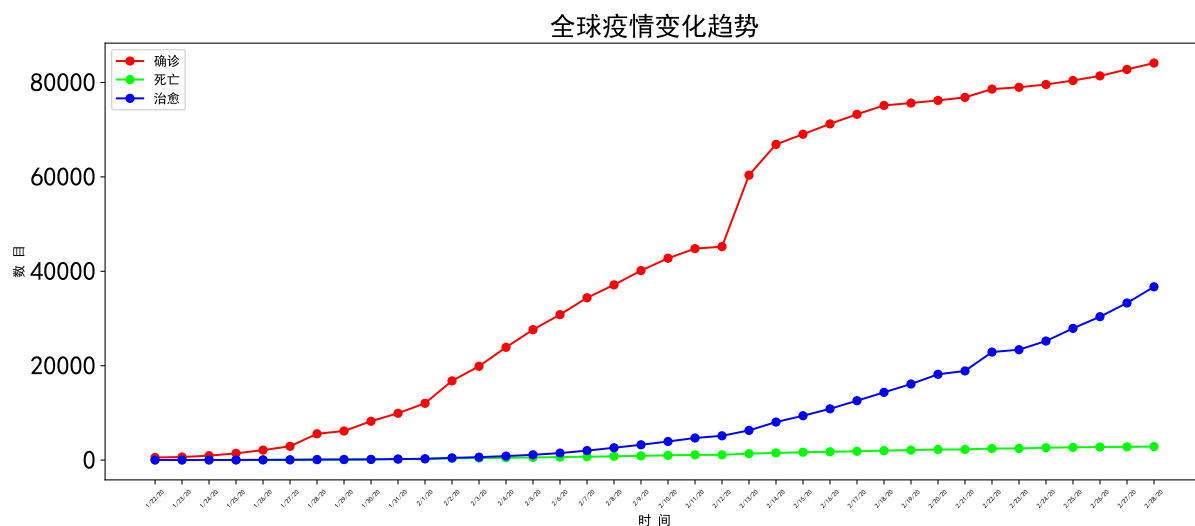
1/22/20	555
1/30/20	8234
1/31/20	9927
2/1/20	12038
2/2/20	16787
2/3/20	19881

下面就可以画出疫情发展趋势图了。

```
1 plt.figure(figsize=(15, 6))
2 plt.plot(all_confirmed, color='r', label='确诊', marker='o')
3 plt.plot(all_deaths, color='lime', label='死亡', marker='o')
4 plt.plot(all_recovered, color='b', label='治愈', marker='o')
5 plt.xticks(rotation = 45,size = 5)

1 plt.yticks(size=20)

1 plt.xlabel('时 间',size = 10)
2 plt.ylabel('数 目',size = 10)
3 plt.title('全球疫情变化趋势',size = 20)
4 plt.legend(loc = "upper left",fontsize = 10)
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。

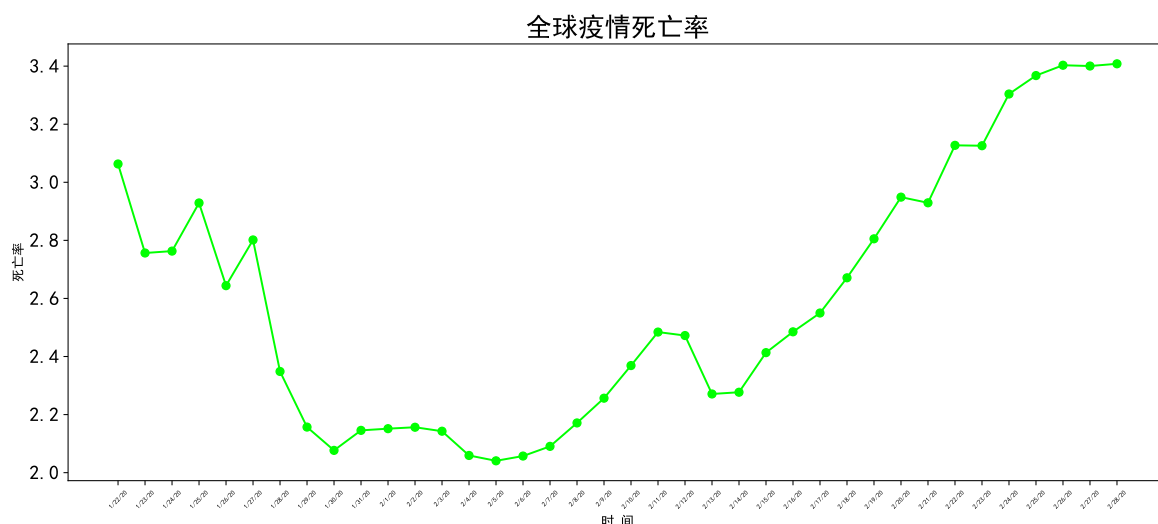
```
1 death_rate = (all_deaths/all_confirmed)*100
2 plt.figure(figsize=(15, 6))
3 plt.plot(death_rate,color = 'lime',label = '死亡',marker = 'o')
4 plt.xticks(rotation = 45,size = 5)

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

```

1 plt.xlabel('时间',size = 10)
2 plt.ylabel('死亡率',size = 10)
3 plt.title('全球疫情死亡率',size = 20)

```



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。

```

1 last_update = '2/28/20' # 设置最新数据日期
2 China_cases = confirmed[['Province/State',last_update]][confirmed['Country/Region'] == 'Mainland China']
3 China_cases['recovered'] = recovered[['last_update']][recovered['Country/Region']=='Mainland China']
4 China_cases['deaths']=deaths[['last_update']][deaths['Country/Region']=='Mainland China']
5 China_cases.set_index('Province/State', inplace=True)
6 China_cases = China_cases.rename(columns = {last_update:'confirmed'})
7 print(China_cases)

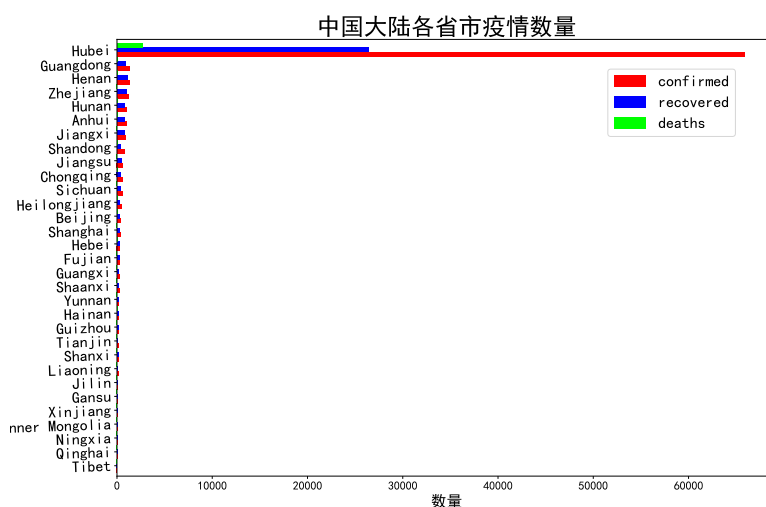
```

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。

```
1 Mainland_china = China_cases.sort_values(by='confirmed',ascending=True)
2 Mainland_china.plot(kind='barh', figsize=(15, 10), color = ['red','blue',
    , 'lime'], width=1, rot=2)
3 plt.title('中国大陆各省市疫情数量', size=30)
4 plt.ylabel('省/市',size=20)
5 plt.xlabel('数量',size = 20)
6 plt.yticks(size=20)

1 plt.xticks(size=15)
1 plt.legend(bbox_to_anchor=(0.95,0.95),fontsize = 20)
```



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。

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

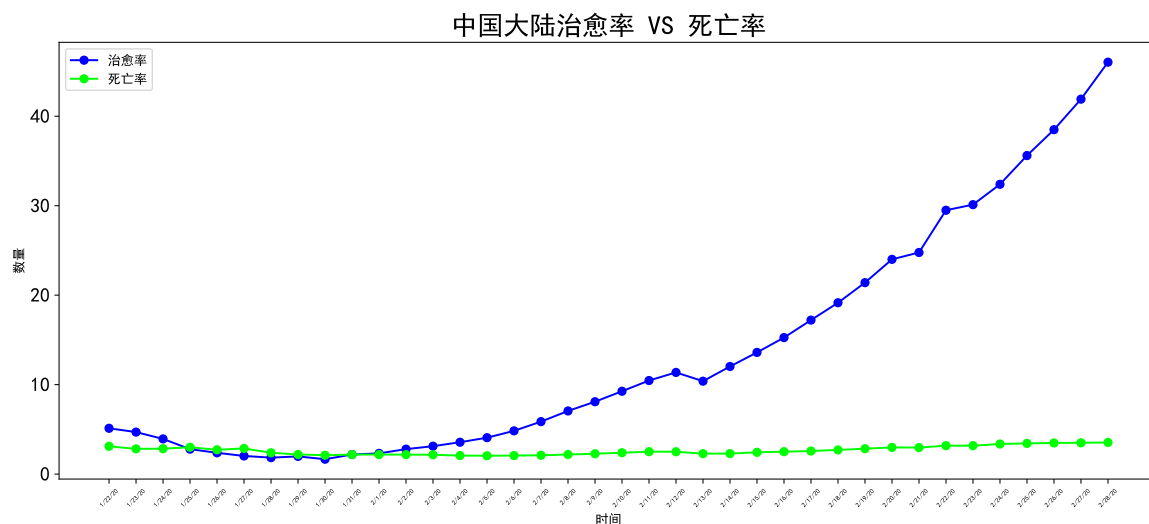
接下来就是画图了

```
1 plt.figure(figsize=(15, 6))
2 plt.plot(recover_rate, color = 'blue', label = '治愈率', marker = 'o')
3 plt.plot(death_rate, color = 'lime', label = '死亡率', marker = 'o')
4 plt.title('中国大陆治愈率 VS 死亡率',size=20)
5 plt.ylabel('数量',size=10)
6 plt.xlabel('时间',size=10)
7 plt.xticks(rotation=45,size=5)
```

```

1 plt.yticks(size=15)
1 plt.legend(loc = "upper left",fontsize = 10)
2 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。

```

1 confirmed_others = confirmed[confirmed['Country/Region'] != 'Mainland China']
2 confirmed_others = np.sum(confirmed_others.iloc[:,4:])
3 recovered_others = recovered[recovered['Country/Region'] != 'Mainland China']
4 recovered_others = np.sum(recovered_others.iloc[:,4:])
5 deaths_others = deaths[deaths['Country/Region'] != 'Mainland China']
6 deaths_others = np.sum(deaths_others.iloc[:,4:])
7 recover_rate = (recovered_others/confirmed_others)*100
8 death_rate = (deaths_others/confirmed_others)*100

```

然后就是画图了。

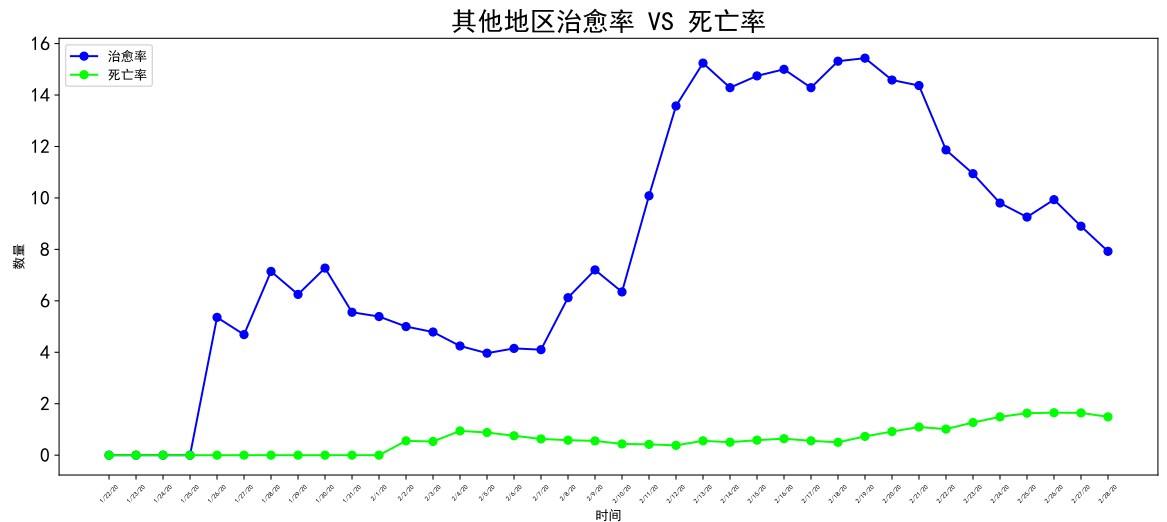
```

1 plt.figure(figsize=(15, 6))
2 plt.plot(recover_rate, color = 'blue', label = '治愈率', marker = 'o')
3 plt.plot(death_rate, color = 'lime', label = '死亡率', marker = 'o')
4 plt.title('其他地区治愈率 VS 死亡率',size=20)
5 plt.ylabel('数量',size=10)
6 plt.xlabel('时间',size=10)
7 plt.xticks(rotation=45,size=5)

1 plt.yticks(size=15)

1 plt.legend(loc = "upper left",fontsize = 10)
2 plt.show()

```



接下来看看其他地区疫情数量。

```

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

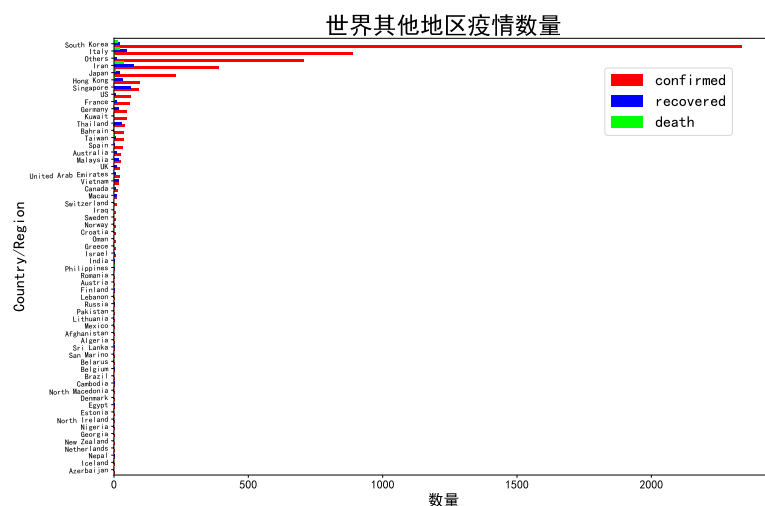
```

然后画图

```

1 sorted = others_countries.sort_values(by = 'confirmed',ascending=True)
2 sorted.plot(kind='barh', figsize=(15, 10), color = ['red','blue','lime'
   ], width=1,rot=2)
3 plt.title('世界其他地区疫情数量', size=30)
4 plt.ylabel('Country/Region',size = 20)
5 plt.xlabel('数量',size = 20)
6 plt.yticks(size=10)
1 plt.xticks(size=15)
1 plt.legend(bbox_to_anchor=(0.95,0.95),fontsize = 20)

```



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。

```
1 others=confirmed[['Country/Region','Lat','Long',last_update]][confirmed[
    'Country/Region'] != 'Mainland China']
2 others['recovered'] = recovered[[last_update]][recovered['Country/Region']
    != 'Mainland China']
3 others['death'] = deaths[[last_update]][deaths['Country/Region'] != '
    Mainland China']
4 others_countries = others.rename(columns = {last_update:'confirmed'})
5 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
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

```
1 import folium
2 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。

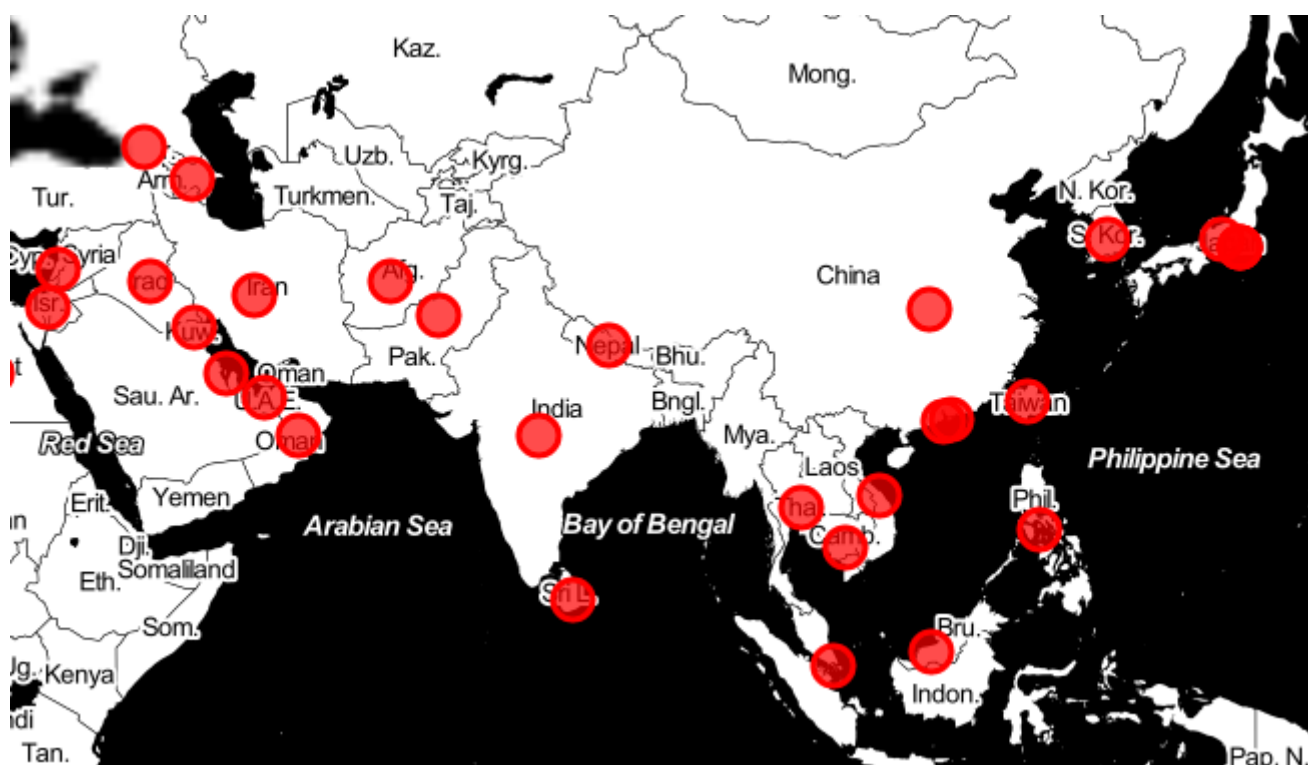
然后往 world_map 里添加其他元素，注意这里的 for 循环和最后的 add_to 是把经纬度点的信息一个一个的加进去

```
1 for lat, lon, value, name in zip(others_countries['Lat'], others_
    countries['Long'], others_countries['confirmed'], others_countries['
    Country/Region']):
2     folium.CircleMarker([lat, lon],
3                           radius=10,
4                           popup = ('<strong>Country</strong>: ' + str(
    name).capitalize() + '<br>'
5                                   '<strong>Confirmed Cases</strong>: ' + str(
    value) + '<br>'),
6                           color='red',
7                           fill_color='red',
8                           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。

```
1 world_map
```



接下来使用 plotly 绘制每日疫情扩散地图，首先是导入包

```
1 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 DataFrame

```
1 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。

```
1 confirmed
```

还需要把 date 列转换成 datetime 格式的数据

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

最终 confirmed 的数据如下格式

```
1 confirmed
```

	province	country	Lat	Long	date	confirmed	date_dt
	Anhui	Mainland China	31.8257	117.2264	2020-01-22	1	2020-01-22
	Beijing	Mainland China	40.1824	116.4142	2020-01-22	14	2020-01-22
	Chongqing	Mainland China	30.0572	107.8740	2020-01-22	6	2020-01-22
	Fujian	Mainland China	26.0789	117.9874	2020-01-22	1	2020-01-22
	Gansu	Mainland China	36.0611	103.8343	2020-01-22	0	2020-01-22

同理整理出治愈数据和死亡数据。

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

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

现在三种数据都有了，我们把它合并一张表里面，主要用到 merge 函数

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

```
=merge_on)
```

最终我们得到的数据就是下面这种

date_dt	province	confirmed	deaths	recovered	Lat	Long	size
2020-01-22	Montreal, QC	0	0	0	45.5017	-73.5673	0.000000
2020-01-22	Anhui	1	0	0	31.8257	117.2264	1.000000
2020-01-22	Beijing	14	0	0	40.1824	116.4142	3.741657
2020-01-22	Boston, MA	0	0	0	42.3601	-71.0589	0.000000
2020-01-22	British Columbia	0	0	0	49.2827	-123.1207	0.000000

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 confirmed 数据的二分之一一次方来表示实心圆的大小。

```
1 Coronavirus_map = all_data.groupby(['date_dt', 'province'])['confirmed',  
    'deaths', 'recovered', 'Lat', 'Long'].max().reset_index()  
2 Coronavirus_map['size'] = Coronavirus_map.confirmed.pow(0.5) # 创建实心圆  
    大小  
3 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。

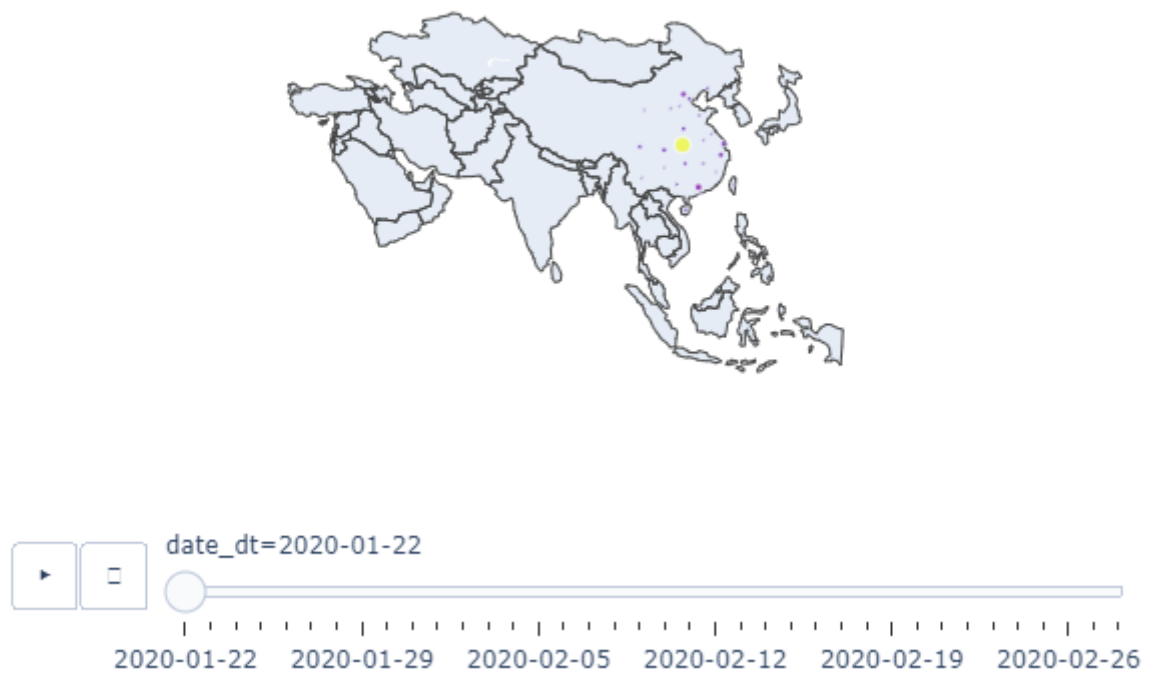
date_dt	province	confirmed	deaths	recovered	Lat	Long	size
2020-01-22	Montreal, QC	0	0	0	45.5017	-73.5673	0.000000
2020-01-22	Anhui	1	0	0	31.8257	117.2264	1.000000
2020-01-22	Beijing	14	0	0	40.1824	116.4142	3.741657
2020-01-22	Boston, MA	0	0	0	42.3601	-71.0589	0.000000
2020-01-22	British Columbia	0	0	0	49.2827	-123.1207	0.000000

最后就是绘图部分

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

好了地图部分就画好了，代码运行完就会出现

亚洲地区疫情扩散图



，点击播放按钮，就会动态变化