新型疫情数据分析报告

一、 数据选用与预处理

选用 COVID_basedata 与 COVID_extra 两个表格数据，其中 COVID_basedata 为国内的新型冠状病毒感染数据统计，COVID_extra.csv 为全世界的新型冠状病毒感染数据统计

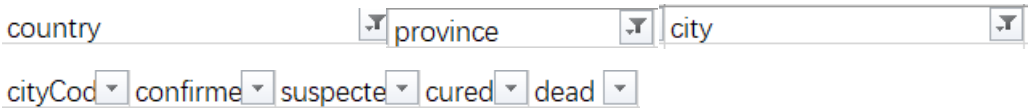
在预处理的部分，统一将这两个表格保存为.csv 格式，采用 UTF-8 编码，解决了先前读取时的编码混乱问题。

在数据清洗的过程中，未发现具有明显异常的数据（比如负数）。这也在后续的绘图描述中得到验证。

在后续的数据处理过程中，我们选择使用了以下列的数据。

COVID-19_extra.csv 中：

COVID_basedata.csv 中：



本次大作业主要基于 numpy 包的 ndarray 二维数组形式，其优点是直观简洁，运用较为灵活，便于绘图对比。

二、 数据检索查询

（一）功能展示与描述

本程序共设计了五种查询功能，通过键盘实现交互：

1、查询某日某地当天的疫情情况，可根据用户需求分别查询累计患病人数、累计死亡人数、新增患病人数、新增死亡人数四个数据

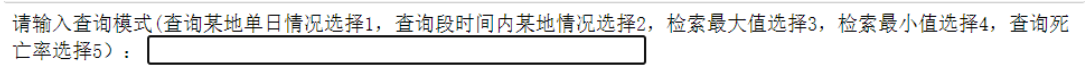
2、查询某段时间内某地的疫情情况，可根据用户需求选择起始时间，可查询内容有段时间内累计患病数、累计死亡数两个数据

3、查询某地新增患病数、新增死亡数最大值（多个相同数值返回第一个）

4、查询某地新增患病数、新增死亡数最小值（多个相同数值返回第一个）

5、查询某地到某日的累计死亡率

在 jupyter noteboo 运行 COV_analysis.py 程序 会提示用户输入查询模式



模式 1——点查询

会要求用户依次输入查询的地区、日期、查询内容，测试情况如图所示：

```
请输入查询模式(查询某地单日情况选择1，查询段时间内某地情况选择2，检索最大值选择3，检索最小值选择4，查询死亡率选择5）：1
当前使用模式：模式 1
地区请输入英文名或拼音，日期标准格式为 XXXX-XX-XX
请输入要查询的地区：China
您要查询的地区是： China
请输入要查询的日期：2020-02-14
您要查询的日期是： 2020-02-14
请输入要查询的信息(当前支持查询的信息有new_cases,new_deaths,total_cases,total_deaths)：new_cases
您要查询的信息是： new_cases
查询新增病例new_cases为： 4156
```

模式 2：区间查询

会要求用户依次输入查询的地区、开始日期、时间段跨度、查询内容，如下图所示：

```
请输入查询模式(查询某地单日情况选择1，查询段时间内某地情况选择2，检索最大值选择3，检索最小值选择4，查询死亡率选择5）：2
当前使用模式：模式 2
地区请输入英文名或拼音，日期标准格式为 XXXX-XX-XX
请输入要查询的地区：China
您要查询的地区是： China
请输入要查询的开始日期：2020-02-01
您要查询的开始日期是： 2020-02-01
请输入要查询的连续天数：30
您要查询的连续天数是： 30
请输入要查询的信息(当前支持查询的信息有new_cases,new_deaths)：new_deaths
您要查询的信息是： new_deaths
该段时间内的新增死亡数new_deaths为： 2701
```

模式 3、4：最值查询

会要求用户依次输入地区、查询内容，如下图所示：

```
请输入查询模式(查询某地单日情况选择1，查询段时间内某地情况选择2，检索最大值选择3，检索最小值选择4，查询死亡率选择5）：3
当前使用模式：模式 3
地区请输入英文名或拼音，日期标准格式为 XXXX-XX-XX
请输入要查询的地区：China
您要查询的地区是： China
请输入要查询的信息(当前支持查询的信息有new_cases,new_deaths)：new_cases
您要查询的信息是： new_cases
该地区最大新增病例数出现在 2020-02-13
```

```
请输入查询模式(查询某地单日情况选择1，查询段时间内某地情况选择2，检索最大值选择3，检索最小值选择4，查询死亡率选择5）：4
当前使用模式：模式 4
地区请输入英文名或拼音，日期标准格式为 XXXX-XX-XX
请输入要查询的地区：China
您要查询的地区是： China
请输入要查询的信息(当前支持查询的信息有new_cases,new_deaths)：new_cases
您要查询的信息是： new_cases
该地区最小新增病例数出现在 2019-12-31
```

模式 5：累计死亡率查询

会要求用户依次输入地区、查询日期，如下图所示：

```
请输入查询模式(查询某地单日情况选择1，查询段时间内某地情况选择2，检索最大值选择3，检索最小值选择4，查询死亡率选择5）：5
当前使用模式：模式 5
地区请输入英文名或拼音，日期标准格式为 XXXX-XX-XX
请输入要查询的地区：China
您要查询的地区是： China
请输入要查询的日期：2020-05-20
您要查询的日期是： 2020-05-20
当日该地区累计死亡率为： 0.055171593409861414
```

（二）功能实现思路分析：

1、首先导入数据为数组，其中用于计算的数值数据导入为 data（浮点类型），用于查询地点的数据导入为 label（string 类型）：

```
#导入数据，这里分别用label和data存储string和浮点类型的数据
country_label=np.array(np.loadtxt('./COVID_extra.csv',dtype=str,delimiter=',',skiprows=1,usecols=(2,3),encoding='utf-8'))
country_data=np.array(np.loadtxt('./COVID_extra.csv',dtype=float ,delimiter=',',skiprows=1,usecols=(4,5,6,7),encoding='utf-8'))
China_label=np.array(np.loadtxt('./COVID_basedata.csv',dtype=str,delimiter=',',skiprows=1,usecols=(0,1,2,3,4,5,6),encoding='utf-8'))
China_data=np.array(np.loadtxt('./COVID_basedata.csv',dtype=float,delimiter=',',skiprows=1,usecols=(7,8,9,10),encoding='utf-8'))
```

2、用户交互选择，即复现用户的输入要求

```
#选择查询模式,这里增加了用户交互的功能，即给出反馈
select = input("请输入查询模式(查询某地单日情况选择1，查询段时间内某地情况选择2，检索最大值选择3，检索最小值选择4，查询死亡率选择5）：")
print("当前使用模式：模式",select)
print("地区请输入英文名或拼音，日期标准格式为 XXXX-XX-XX")
```

3、模式的核心代码

模式 1 为

```
#模式1
if (select  == "1"):
    area = input("请输入要查询的地区：")
    print("您要查询的地区是：", area)
    date1 = input("请输入要查询的日期：")
    print("您要查询的日期是：",date1)
    need = input("请输入要查询的信息(当前支持查询的信息有new_cases,new_deaths,total_cases,total_deaths)：") #获取用户进一步查询的信息
    print("您要查询的信息是：", need)
    searched_values = np.array([area,date1])  #这里定义查询数组，匹配查询
    for i in range(country_label.shape[0]):    #这里通过对country_label循环对比，找到要查询的内容对应的index
        if ((country_label[i]==searched_values).all()):
            break
    if (need =="new_cases"):                                    #根据用户需求，将index对应的数据输出
        print ("查询新增病例new_cases为：",int(country_data[i,1] ) )
    elif (need == "new_deaths"):
        print("查询新增死亡数new_deaths为：", int(country_data[i, 3]) )
    elif (need =="total_cases"):
        print ("查询累积病例total_cases为：",int (country_data[i,0] ) )
    elif (need =="total_deaths"):
        print ("查询累计死亡数total_deaths为：",int(country_data[i,2] ) )
```

模式 2 为

```
#模式2
elif (select  == "2"):
    area = input("请输入要查询的地区：")
    print("您要查询的地区是：", area)
    date_begin = input("请输入要查询的开始日期：")
    print("您要查询的开始日期是：", date_begin)
    date_end = input("请输入要查询的连续天数：")
    print("您要查询的连续天数是：", date_end)
    need = input("请输入要查询的信息(当前支持查询的信息有new_cases,new_deaths)：")
    print("您要查询的信息是：", need)
    searched_values = np.array([area, date_begin])   #同理，通过输入的地区和日期找到index的起始位置
    for i in range(country_label.shape[0]):
        if ((country_label[i] == searched_values).all()):
            break
    result = 0          #result用于统计累积量
    if (need == "new_cases"):
        for x in range(i, i + int(date_end)+1):        #在该时间段进行累计
            result = result +country_data[x,1]
        print ("该段时间内的新增病例new_cases为：",int(result))
    elif (need == "new_deaths"):
        for x in range(i, i + int(date_end)+1):
            result = result +country_data[x,3]
        print ("该段时间内的新增死亡数new_deaths为：",int(result))
```

模式 3 为:

```
58   #模式3
59   elif (select == "3"):
60       area = input("请输入要查询的地区：")
61       print("您要查询的地区是：", area)
62       need = input("请输入要查询的信息(当前支持查询的信息有new_cases,new_deaths)：")
63       print("您要查询的信息是：", need)
64       count = np.sum(country_label == area)   #统计要查询的地区在label数组里出现的总次数
65       index = np.where(country_label ==area)[0][0]   #找到第一个出现要查询地区的index位置
66       new_data = country_data [index:index+count]
67       max_data = np.max(new_data,axis=0)          #找到拆分数据里每一列中的最大值，输出为一个数组
68       if (need == "new_cases"):
69           t = max_data[1]               #得到最值数据
70           index_t = np.where(country_data == t)[0][0]       #where函数定位最值数据所在的index
71           print ("该地区最大新增病例数出现在",country_label[index_t][1] )        #通过index反输出日期
72       elif (need == "new_deaths"):
73           t = max_data[3]
74           index_t = np.where(country_data == t)[0][0]
75           print ("该地区最大新增死亡数出现在",country_label[index_t][1] )
```

模式 4:

```
#模式4
elif (select == "4"):
    area = input("请输入要查询的地区：")
    print("您要查询的地区是：", area)
    need = input("请输入要查询的信息(当前支持查询的信息有new_cases,new_deaths)：")
    print("您要查询的信息是：", need)
    count = np.sum(country_label == area)
    index = np.where(country_label == area)[0][0]
    new_data = country_data[index:index + count]
    max_data = np.min(new_data, axis=0)
    if (need == "new_cases"):
        t = max_data[1]
        index_t = np.where(country_data == t)[0][0]
        print("该地区最小新增病例数出现在",country_label[index_t][1])
    elif (need == "new_deaths"):
        t = max_data[3]
        index_t = np.where(country_data == t)[0][0]
        print("该地区最小新增死亡数出现在",country_label[index_t][1])
```

模式 5:

```
#模式5
elif (select  == "5"):
    area = input("请输入要查询的地区：")
    print("您要查询的地区是：", area)
    date1 = input("请输入要查询的日期：")
    print("您要查询的日期是：",date1)
    searched_values = np.array([area, date1])
    for i in range(country_label.shape[0]):          #循环total_label找到对应的index
        if ((country_label[i] == searched_values).all()):
            break
    print ("当日该地区累计死亡率为：",country_data[i,2]/country_data[i,0] )  #根据index计算死亡率
```

三、　数据分析与图示对比

（一） 国内疫情对比分析

<1> 数据分析与刻画

对于国内疫情，本项目重点对比了武汉、整个湖北省、全国除了湖北的其他地区的疫情数据。在 COV_analysis1.py 中，绘制如下两幅数据图：

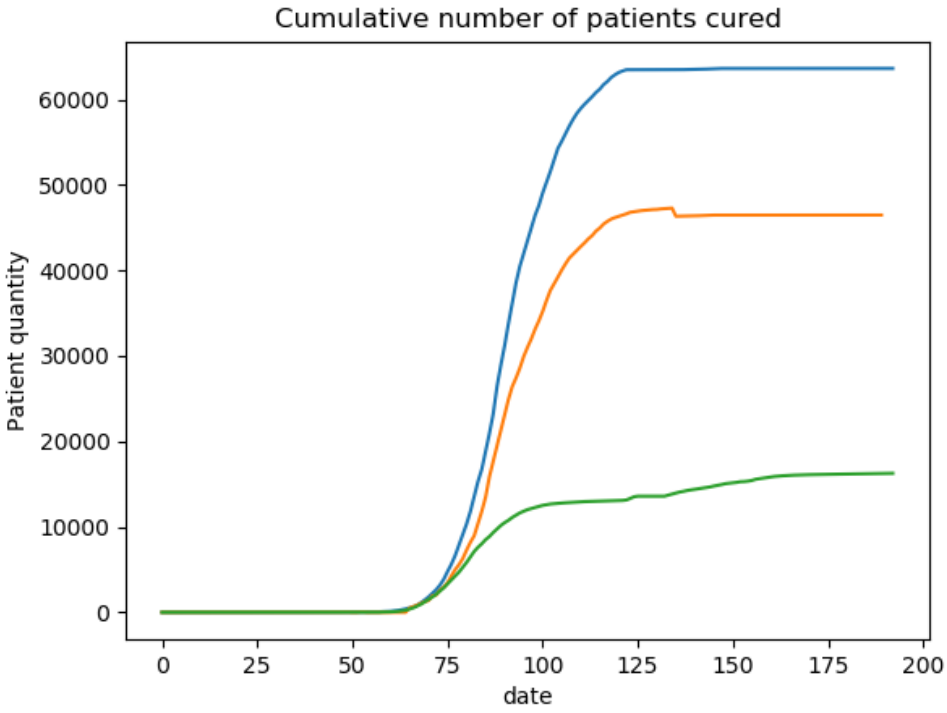


图 3-1：国内地区累计病患对比图

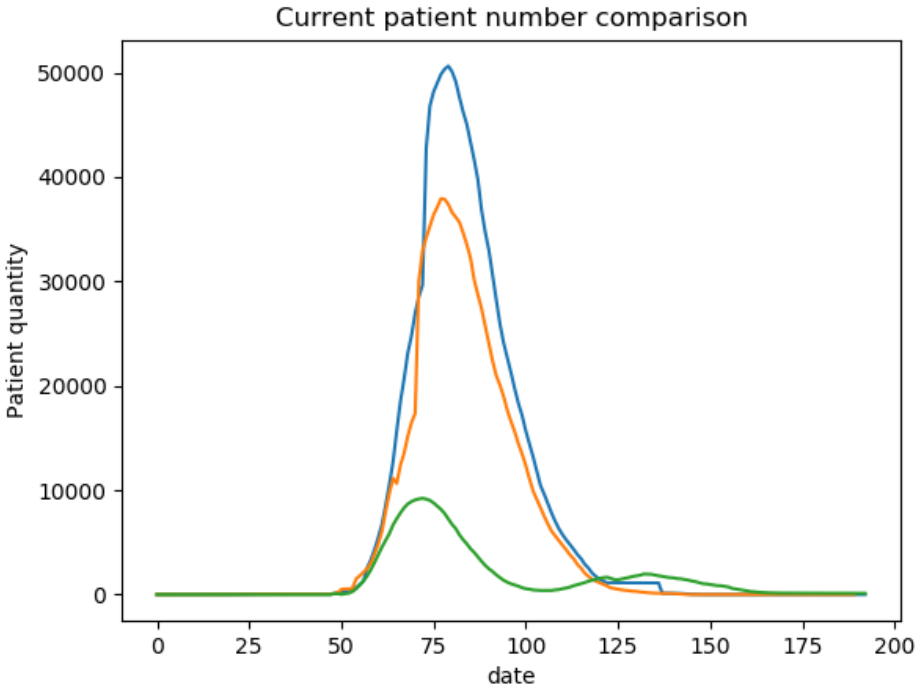


图 3-2：国内当前病患对比图

其中蓝色为湖北省数据，橙色为武汉市数据，绿色为全国其他地区数据。为方便代码书写，Date 标签为以 2019 年 12 月 31 日为基准初始日期，一天为一个单位值。

由图可见，全国病患数量爆发式增长的节点在疫情开始的 55 天左右，即 2 月中旬左右，这符合我们对疫情数据的直观认知，也与政府的新闻资讯较为匹配。在响应了

国家政府号召以及地方政府的努力管控下，疫情爆发后 75 日左右感染人数开始下降，即 3 月初开始减少。自此湖北、武汉的感染人数乘底数小于 1 的指数式下降直到约疫情开始后 125 天趋于平稳，即 4 月中旬开始，湖北的疫情几乎得到解决。

对于全国其他地区，疫情发展的情况趋势大致相似，而其峰值相对有所下降，即疫情的规模相对湖北地区较小。而值得注意的是，全国其他地区的感染人数在疫情爆发后 100 天到 150 天之间有一个小的波动，出现小幅度回升。这一方面可能由于境外输入病例造成疫情波动，也可能由于居民的防疫意识有所下降。随后进一步政策防控下，疫情在爆发后 150 天趋于平稳。

<2>程序实现过程：

（对应 COV_analysis1.py）

1、导入数据

```
#导入数据
China_label=np.array(np.loadtxt('./COVID_basedata.csv',dtype=str,delimiter=',',skiprows=1,usecols=(1,3,5),encoding='utf-8'))
China_data=np.array(np.loadtxt('./COVID_basedata.csv',dtype=float,delimiter=',',skiprows=1,usecols=(7,8,9,10),encoding='utf-8'))
```

2、定义查找数据变量及 index

```
#这里通过searched_XXX来标记查找规律的数据，index列表用于存储对应数据
searched_hubei = np.array(["中国","湖北省",""])
index_hubei = []
searched_wuhan = np.array(["中国","湖北省","武汉市"])
index_wuhan = []
searched_China = np.array(["中国","",""])
index_China = []
```

3、分别预处理湖北数据、武汉数据、全国数据

```
for i in range(China_label.shape[0]):  #这里通过label找到对应的湖北省数据的所有index
    if ((China_label[i] == searched_hubei).all()):
        index_hubei.append(i)
x_hubei = range(len(index_hubei))   #定义横纵坐标
y_hubei_total = []
for i in index_hubei:     #导入数组列表  0: int
    y_hubei_total.append(China_data[i,0]-China_data[i,2]-China_data[i,3] )
    #y_hubei_total.append(China_data[i,0] )
```

```
for i in range(China_label.shape[0]):   #这里通过label找到对应的武汉市数据的所有index
    if ((China_label[i] == searched_wuhan).all()):
        index_wuhan.append(i)
x_wuhan = range(len(index_wuhan))     #定义横纵坐标
y_wuhan_total = []
for i in index_wuhan:       #导入数组列表
    y_wuhan_total.append(China_data[i,0]-China_data[i,2]-China_data[i,3] )
    #y_wuhan_total.append(China_data[i,0] )
```

```
for i in range(China_label.shape[0]):   #这里通过label找到国内的总数据的index
    if ((China_label[i] == searched_China ).all()):
        index_China.append(i)
x_China = range(len(index_China ) )   #定义横纵坐标
y_China_total = []
for i in range(len(index_China )):
    #这里计算的是国内其他地区的数据，方法为用全国的数据减去湖北省的数据
    x = China_data[index_China[i] ,0]-China_data[index_China[i] ,2]-China_data[index_China [i],3]-(China_data[index_hubei[i] ,0]-China_data[index_hubei[i] ,2]-China_data[
    y_China_total.append(x)
    #x = China_data[index_China[i] ,0]-(China_data[index_hubei[i] ,0])
    #y_China_total.append(x)
```

4、图像绘制：

```
50  #作图
51  plt.title("Current patient number comparison")
52  plt.xlabel("date")
53  plt.ylabel("Patient quantity")
54  plt.plot(x_hubei ,y_hubei_total )
55  plt.plot(x_wuhan ,y_wuhan_total )
56  plt.plot(x_China ,y_China_total )
57  plt.show()
58
59
60  # #作图
61  # plt.title("Cumulative number of patients cured")
62  # plt.xlabel("date")
63  # plt.ylabel("Patient quantity")
64  # plt.plot(x_hubei ,y_hubei_total )
65  # plt.plot(x_wuhan ,y_wuhan_total )
66  # plt.plot(x_China ,y_China_total )
67  # plt.show()
68
```

（二）中日韩疫情情况对比分析

通过对中日韩三国的疫情数据筛选、作图，得到如下三张图，其中蓝色为中国、橙色为日本、绿色为韩国。对数据进行分别处理，并绘图如下：

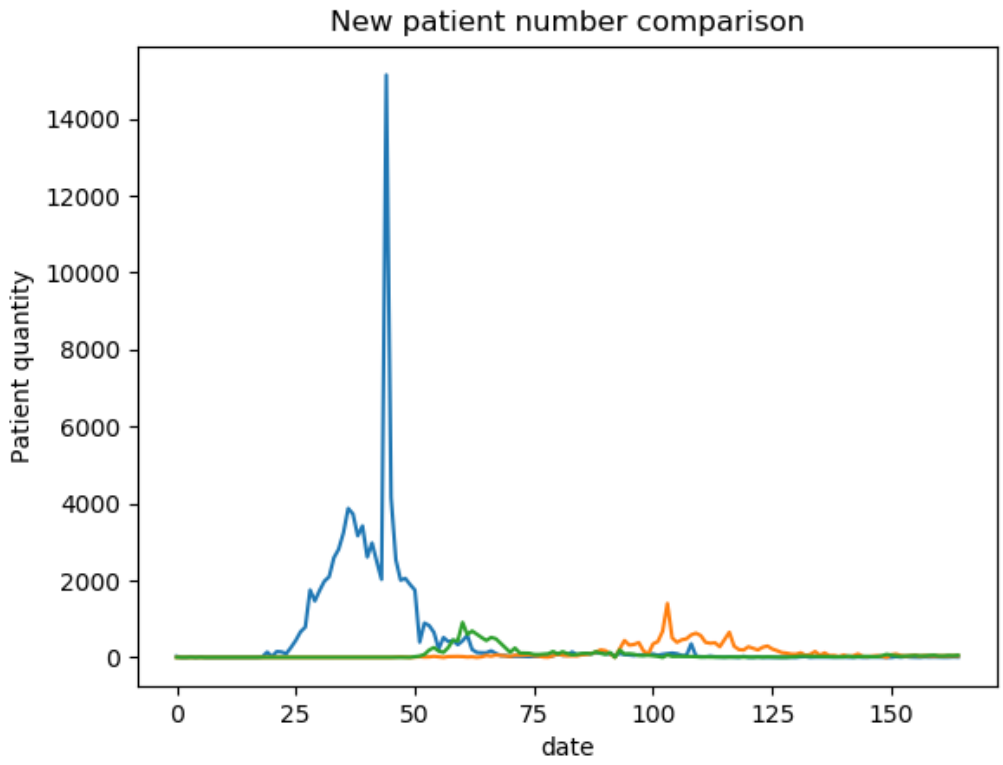


图 3-3：中日韩新感染患者对比图

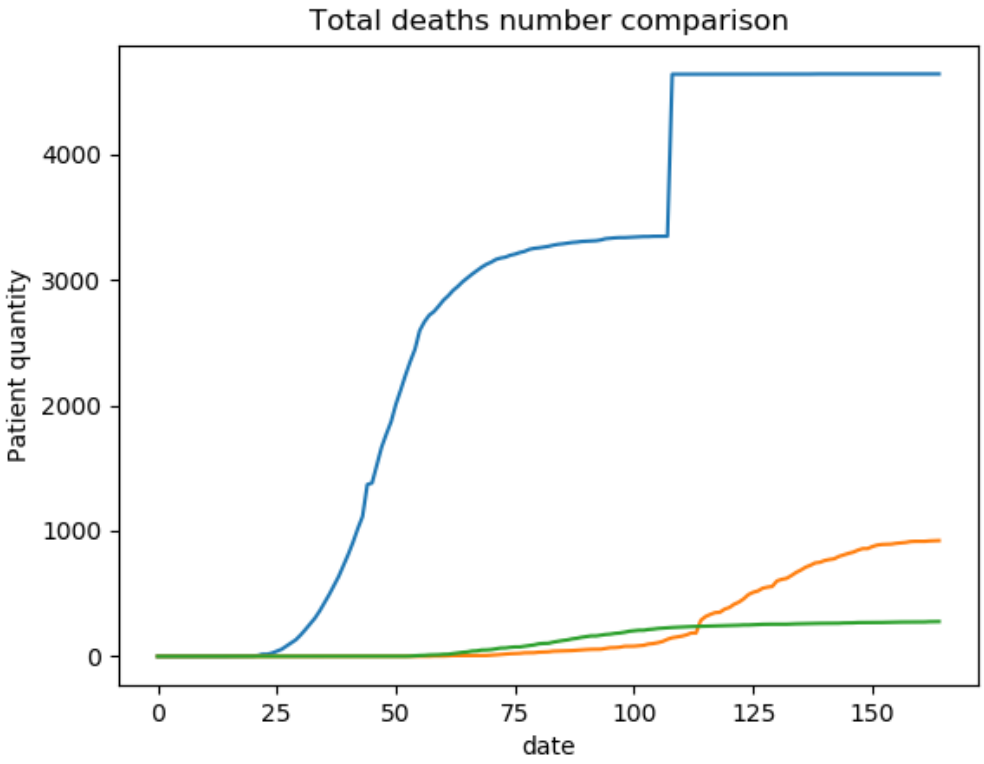


图 3-4：中日韩累计死亡人数对比图

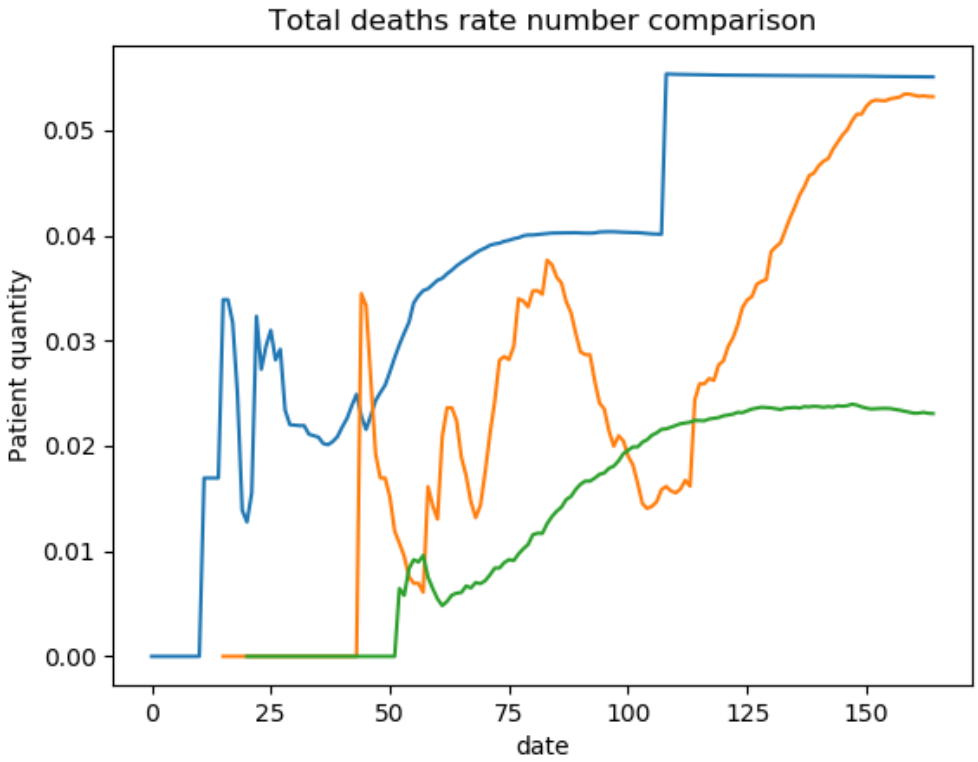


图 3-5：中日韩累计死亡率对比图

由图 3-3，我们可以看出，中日韩疫情爆发的时间节点有不同，中国疫情爆发主要集中在疫情开始后的 25 到 60 天，之后新增病患趋于稳定；日本疫情爆发主要在疫情开始后的 90 天至 125 天，之后新增病患趋于稳定；韩国疫情爆发主要在疫情开始后的 50 到 75 天，之后新增病患趋于稳定。新增病患的数目均较为平稳，因此可以认为，三国在对疫情的管控处理都比较及时。

从图 3-4、图 3-5 可以看出，三国受疫情影响死亡的病患爆发时间顺序和新增病患一致，但是具体时间均稍后移动了约 20 天左右。这可以反映出疫情危重者的生存周期。即确诊之后严重的患者存活的时间约 20 天。三国的死亡人数中，中国最高，这由于中国感染的人口基数较大。当前中韩两国的死亡人数均趋于不变，日本的死亡人数仍在增加，而结合日本新增病患趋于 0 的结论，可以看出日本国内仍有许多早已确诊的患者不治而亡。中日韩三国死亡率大致类似。

值得注意的是，日本和中国的新增人数和死亡率数据波动较大，折线并不平滑。这可以推测应该是由于统计死亡患者的时间间隔周期的缘故，即更新死亡患者不够及时，统计方法具有一定局限性。同时，统计数据一两日内骤增也会直观影响死亡率，这一点在我国体现的尤为明显。2020 年 2 月 13 日，中国新增疫情感染者 13000 余例，这对于图 3-3 中的新增人数造成了极大波动，折线出现极端态势。后续的死亡人数和死亡率也因此产生波动。

具体的 Python 实现过程与第一部分相近，详见 COV_analysis2.py

（三）欧美各国的疫情数据对比

为了分析欧美疫情发展情况，本文选择了欧洲的英国、德国、挪威，美洲的美国、巴西，以及当前疫情控制趋于良好的中国作为参照对比。作图如下，其中蓝色为中国，橙色为美国，绿色为英国，红色为德国，紫色为巴西，褐色为挪威。对数据进行分别预处理，与前文类似。

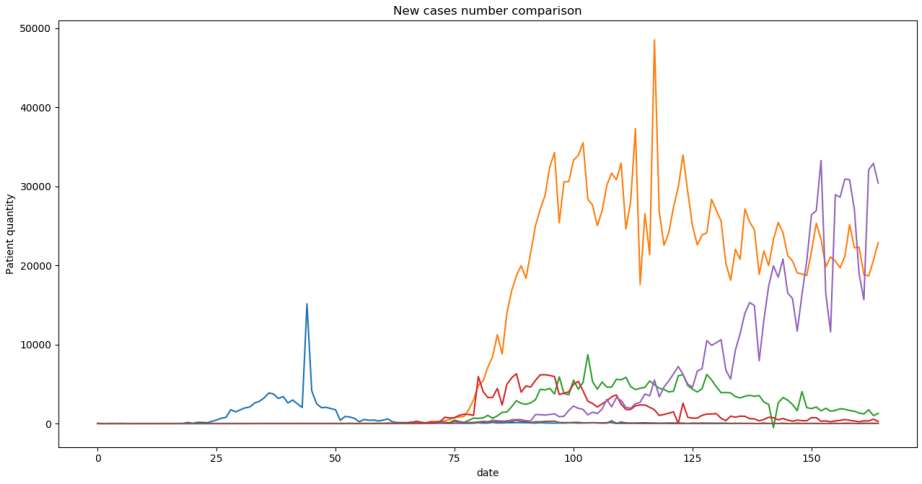


图 3-6：欧美各国新感染患者对比图

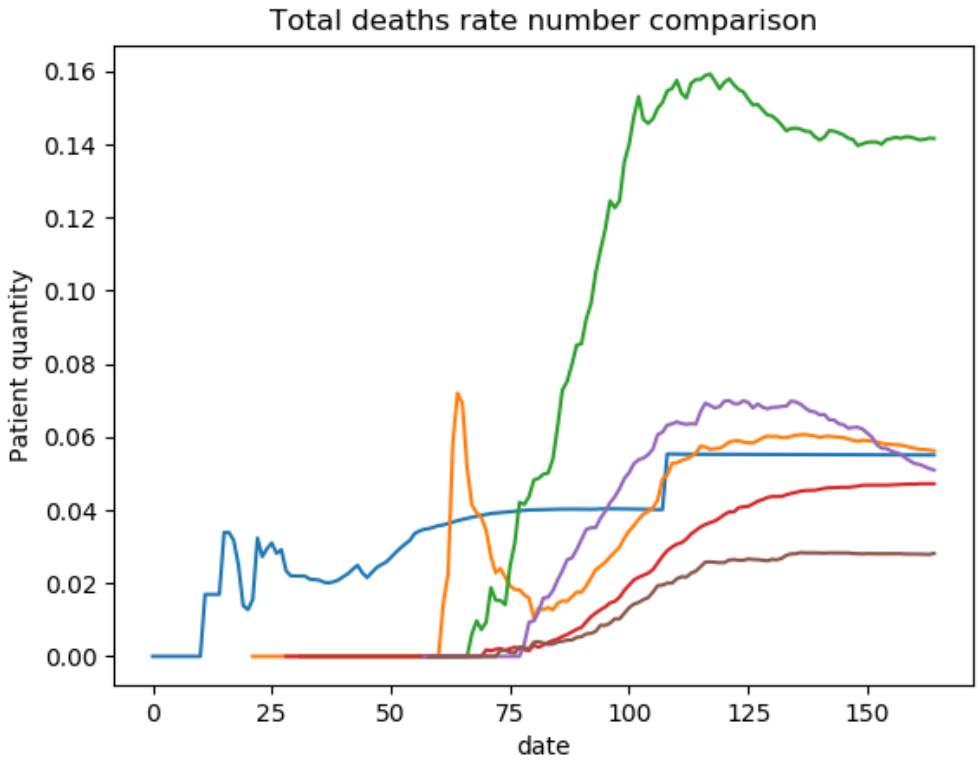


图 3-7：欧美各国累计死亡率对比图

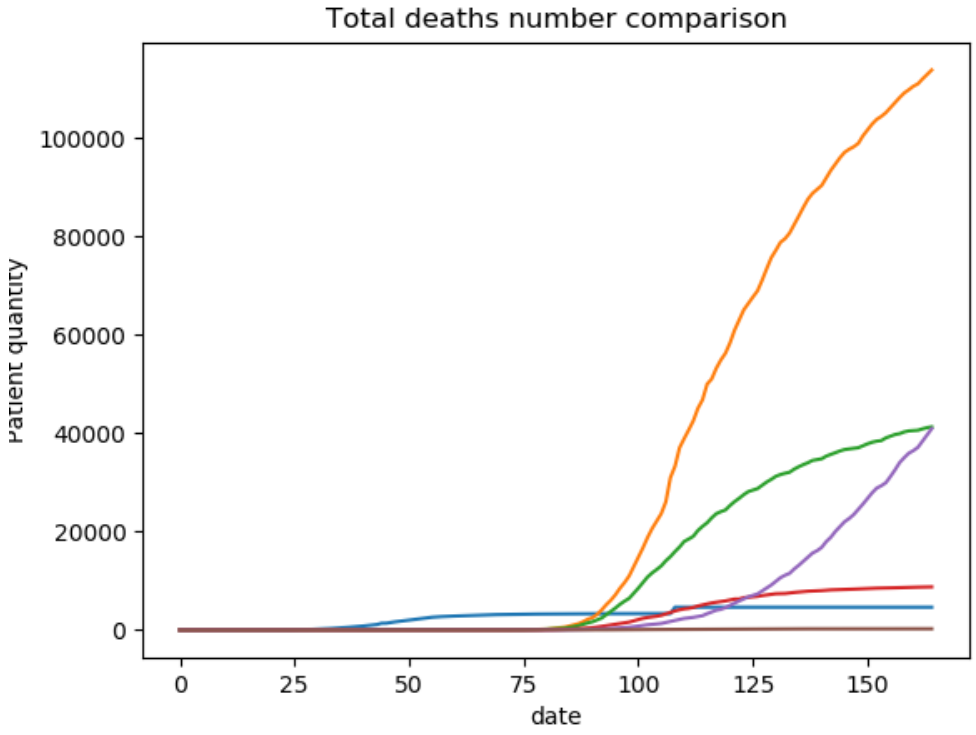


图 3-8：欧美各国累计死亡患者对比图

通过对比数据，结合各国疫情应对政策。欧洲国家中，英国的疫情管控最差，这很可能是由于英国政府对疫情不够重视“无为而治”的态度，英国的死亡率高居六国之首，所幸的是从新感染患者数目和累计死亡患者数目可以看出，英国疫情正在得到控制，应该会在未来缓慢下降；德国属于疫情管控一般的欧洲国家，得赖于良好的医疗条件，虽然感染的患者数目不小，但是死亡率排六国倒数第二；挪威是欧洲疫情控制比较好的国家之一，地处北欧人口流动相对其他欧洲国家来说较小，政府管理也比较严格，在这次疫情中有着很好的表现，不管是累计患者数、死亡率、死亡数都排在六国中最少的一位。

在美洲国家中，美国的疫情可以看出呈失控状态，结合近期美国国内的政治形势，一是种族运动集会不断增大了交叉感染的几率，二是美国国内复工的呼声较大，居民流动没有得到很好的管控，重灾区人口密集，三是美国国内医疗系统存在一定问题，医生工资普遍较高看病费用较高很多病患没能得到及时的救治，从数据上看疫情短时间内似乎难以得到控制，三项数据都还在不断上升并居高不下。巴西的疫情与美国类似，从时间点上看爆发比美国晚了约一个多月，当前爆发势头比美国更甚，这可能是由于巴西当地医疗条件较差，当地气候给病毒滋生感染以更好的空间。

具体的 Python 实现过程与第一部分相近，详见 COV_analysis3.py

四、 研究展望与结语

通过 4 组程序，本项目实现了数据的交互查询，国内疫情分析，中日韩对比分析以及欧美疫情状况的对比研究。这些程序的实现较为成熟全面，并结合疫情的研究进行了相关的逻辑分析，对大作业要求进行了比较完整的实现。

事实上，笔者也尝试了进一步的分析方法，希望通过传染病的常用数学模型——SIR 模型进行数据拟合分析，以诠释疫情的发展情况。其基本原理是微分方程模型，如下：

$$\frac{dS}{dt} = -\beta SI,$$

$$\frac{dI}{dt} = \beta SI - \gamma I,$$

$$\frac{dR}{dt} = \gamma I.$$

其中，S 代表易感者，I 表示感染者，R 表示恢复者，β代表基础传染系数，γ代表基础恢复系数。结合近日来，北京的疫情出现反弹情况，对这个模型的参数估计和检验，很可能预测北京的疫情进展情况，具有一定的研究意义。通过 scipy.integrate 包，我们可以求解相关的微分方程，核心代码如下：

```
def ode_sirs(data_in,t):
    Y=np.zeros((3))
    V = data_in
    Y[0] = - beta * V[0] * V[1]
    Y[1] = beta * V[0] * V[1] - gamma * V[1]
    Y[2] = gamma * V[1]
    return Y
result = spi.odeint(ode_sirs,index_hubei,t_range)
```

实际操作中，β，γ的系数需要给出比较精准的拟合，之前初步拟合的精确度不高，而且国内疫情波动复杂，直接应用数据模型匹配度有所不够，故本文还未列出这些进阶研究的最终结果，将在后期进一步开发，打算调整数据的切片结构，得出进阶结论。（其实是最近通宵三天了，实在熬不动了 QAQ）

本文已列出结果的项目还是相对完整的，并已经上传至 github，链接为

https://github.com/Kaiyu-Zhang/COV_analysis.git

感谢您的阅读。