notebook-covidspread

June 24, 2024

1 Estimation of COVID-19 Pandemic

1.1 Loading Data

We will use data on COVID-19 infected individuals, provided by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University. Dataset is available in this GitHub Repository.

[19]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (10,3) # make figures larger
```

We can load the most recent data directly from GitHub using `pd.read_csv`. If for some reason the data is not available, you can always use the copy available locally in the `data` folder - just uncomment the line below that defines `base_url`:

[20]:
```
base_url = "https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/
↪csse_covid_19_data/csse_covid_19_time_series/" # loading from Internet
# base_url = "../../data/COVID/" # loading from disk
infected_dataset_url = base_url + "time_series_covid19_confirmed_global.csv"
recovered_dataset_url = base_url + "time_series_covid19_recovered_global.csv"
deaths_dataset_url = base_url + "time_series_covid19_deaths_global.csv"
countries_dataset_url = base_url + "../UID_ISO_FIPS_LookUp_Table.csv"
```

Let's now load the data for infected individuals and see how the data looks like:

[21]:
```
infected = pd.read_csv(infected_dataset_url)
infected.head()
```

[21]:
```
  Province/State Country/Region       Lat       Long  1/22/20  1/23/20  \
0            NaN    Afghanistan  33.93911  67.709953        0        0
1            NaN        Albania  41.15330  20.168300        0        0
2            NaN        Algeria  28.03390   1.659600        0        0
3            NaN        Andorra  42.50630   1.521800        0        0
4            NaN         Angola -11.20270  17.873900        0        0

   1/24/20  1/25/20  1/26/20  1/27/20  …  2/28/23  3/1/23  3/2/23  3/3/23  \
0        0        0        0        0  …   209322  209340  209358  209362
1        0        0        0        0  …   334391  334408  334408  334427
```

```
2         0         0         0         0  ...   271441  271448  271463  271469
3         0         0         0         0  ...    47866   47875   47875   47875
4         0         0         0         0  ...   105255  105277  105277  105277

   3/4/23  3/5/23  3/6/23  3/7/23  3/8/23  3/9/23
0  209369  209390  209406  209436  209451  209451
1  334427  334427  334427  334427  334443  334457
2  271469  271477  271477  271490  271494  271496
3   47875   47875   47875   47875   47890   47890
4  105277  105277  105277  105277  105288  105288

[5 rows x 1147 columns]
```

We can see that each row of the table defines the number of infected individuals for each country and/or province, and columns correspond to dates. Similar tables can be loaded for other data, such as number of recovered and number of deaths.

[22]:
```
recovered = pd.read_csv(recovered_dataset_url)
deaths = pd.read_csv(deaths_dataset_url)
```

1.2 Making Sense of the Data

From the table above the role of province column is not clear. Let's see the different values that are present in `Province/State` column:

[23]:
```
infected['Province/State'].value_counts()
```

[23]:
```
Province/State
Australian Capital Territory                    1
New South Wales                                 1
Northern Territory                              1
Queensland                                      1
South Australia                                 1
                                               ..
Jersey                                          1
Montserrat                                      1
Pitcairn Islands                                1
Saint Helena, Ascension and Tristan da Cunha    1
Turks and Caicos Islands                        1
Name: count, Length: 91, dtype: int64
```

From the names we can deduce that countries like Australia and China have more detailed breakdown by provinces. Let's look for information on China to see the example:

[24]:
```
infected[infected['Country/Region']=='China']
```

[24]:
```
    Province/State Country/Region      Lat      Long  1/22/20  1/23/20  \
59           Anhui          China  31.8257  117.2264        1        9
```

60	Beijing	China	40.1824	116.4142	14	22
61	Chongqing	China	30.0572	107.8740	6	9
62	Fujian	China	26.0789	117.9874	1	5
63	Gansu	China	35.7518	104.2861	0	2
64	Guangdong	China	23.3417	113.4244	26	32
65	Guangxi	China	23.8298	108.7881	2	5
66	Guizhou	China	26.8154	106.8748	1	3
67	Hainan	China	19.1959	109.7453	4	5
68	Hebei	China	39.5490	116.1306	1	1
69	Heilongjiang	China	47.8620	127.7615	0	2
70	Henan	China	37.8957	114.9042	5	5
71	Hong Kong	China	22.3000	114.2000	0	2
72	Hubei	China	30.9756	112.2707	444	444
73	Hunan	China	27.6104	111.7088	4	9
74	Inner Mongolia	China	44.0935	113.9448	0	0
75	Jiangsu	China	32.9711	119.4550	1	5
76	Jiangxi	China	27.6140	115.7221	2	7
77	Jilin	China	43.6661	126.1923	0	1
78	Liaoning	China	41.2956	122.6085	2	3
79	Macau	China	22.1667	113.5500	1	2
80	Ningxia	China	37.2692	106.1655	1	1
81	Qinghai	China	35.7452	95.9956	0	0
82	Shaanxi	China	35.1917	108.8701	0	3
83	Shandong	China	36.3427	118.1498	2	6
84	Shanghai	China	31.2020	121.4491	9	16
85	Shanxi	China	37.5777	112.2922	1	1
86	Sichuan	China	30.6171	102.7103	5	8
87	Tianjin	China	39.3054	117.3230	4	4
88	Tibet	China	31.6927	88.0924	0	0
89	Unknown	China	NaN	NaN	0	0
90	Xinjiang	China	41.1129	85.2401	0	2
91	Yunnan	China	24.9740	101.4870	1	2
92	Zhejiang	China	29.1832	120.0934	10	27

	1/24/20	1/25/20	1/26/20	1/27/20	…	2/28/23	3/1/23	3/2/23	\
59	15	39	60	70	…	2275	2275	2275	
60	36	41	68	80	…	40774	40774	40774	
61	27	57	75	110	…	14715	14715	14715	
62	10	18	35	59	…	17122	17122	17122	
63	2	4	7	14	…	1742	1742	1742	
64	53	78	111	151	…	103248	103248	103248	
65	23	23	36	46	…	13371	13371	13371	
66	3	4	5	7	…	2534	2534	2534	
67	8	19	22	33	…	10483	10483	10483	
68	2	8	13	18	…	3292	3292	3292	
69	4	9	15	21	…	6603	6603	6603	
70	9	32	83	128	…	9948	9948	9948	

71	2	5	8	8	…	2876106	2876106	2876106
72	549	761	1058	1423	…	72131	72131	72131
73	24	43	69	100	…	7437	7437	7437
74	1	7	7	11	…	8847	8847	8847
75	9	18	33	47	…	5075	5075	5075
76	18	18	36	72	…	3423	3423	3423
77	3	4	4	6	…	40764	40764	40764
78	4	17	21	27	…	3547	3547	3547
79	2	2	5	6	…	3514	3514	3514
80	2	3	4	7	…	1276	1276	1276
81	0	1	1	6	…	782	782	782
82	5	15	22	35	…	7326	7326	7326
83	15	27	46	75	…	5880	5880	5880
84	20	33	40	53	…	67040	67040	67040
85	1	6	9	13	…	7167	7167	7167
86	15	28	44	69	…	14567	14567	14567
87	8	10	14	23	…	4392	4392	4392
88	0	0	0	0	…	1647	1647	1647
89	0	0	0	0	…	1521816	1521816	1521816
90	2	3	4	5	…	3089	3089	3089
91	5	11	16	26	…	9743	9743	9743
92	43	62	104	128	…	11848	11848	11848

	3/3/23	3/4/23	3/5/23	3/6/23	3/7/23	3/8/23	3/9/23
59	2275	2275	2275	2275	2275	2275	2275
60	40774	40774	40774	40774	40774	40774	40774
61	14715	14715	14715	14715	14715	14715	14715
62	17122	17122	17122	17122	17122	17122	17122
63	1742	1742	1742	1742	1742	1742	1742
64	103248	103248	103248	103248	103248	103248	103248
65	13371	13371	13371	13371	13371	13371	13371
66	2534	2534	2534	2534	2534	2534	2534
67	10483	10483	10483	10483	10483	10483	10483
68	3292	3292	3292	3292	3292	3292	3292
69	6603	6603	6603	6603	6603	6603	6603
70	9948	9948	9948	9948	9948	9948	9948
71	2876106	2876106	2876106	2876106	2876106	2876106	2876106
72	72131	72131	72131	72131	72131	72131	72131
73	7437	7437	7437	7437	7437	7437	7437
74	8847	8847	8847	8847	8847	8847	8847
75	5075	5075	5075	5075	5075	5075	5075
76	3423	3423	3423	3423	3423	3423	3423
77	40764	40764	40764	40764	40764	40764	40764
78	3547	3547	3547	3547	3547	3547	3547
79	3514	3514	3514	3514	3514	3514	3514
80	1276	1276	1276	1276	1276	1276	1276
81	782	782	782	782	782	782	782

```
82     7326     7326     7326     7326     7326     7326     7326
83     5880     5880     5880     5880     5880     5880     5880
84    67040    67040    67040    67040    67040    67040    67040
85     7167     7167     7167     7167     7167     7167     7167
86    14567    14567    14567    14567    14567    14567    14567
87     4392     4392     4392     4392     4392     4392     4392
88     1647     1647     1647     1647     1647     1647     1647
89  1521816  1521816  1521816  1521816  1521816  1521816  1521816
90     3089     3089     3089     3089     3089     3089     3089
91     9743     9743     9743     9743     9743     9743     9743
92    11848    11848    11848    11848    11848    11848    11848

[34 rows x 1147 columns]
```

1.3 Pre-processing the Data

We are not interested in breaking countries down to further territories, thus we would first get rid of this breakdown and add information on all territories together, to get info for the whole country. This can be done using `groupby`:

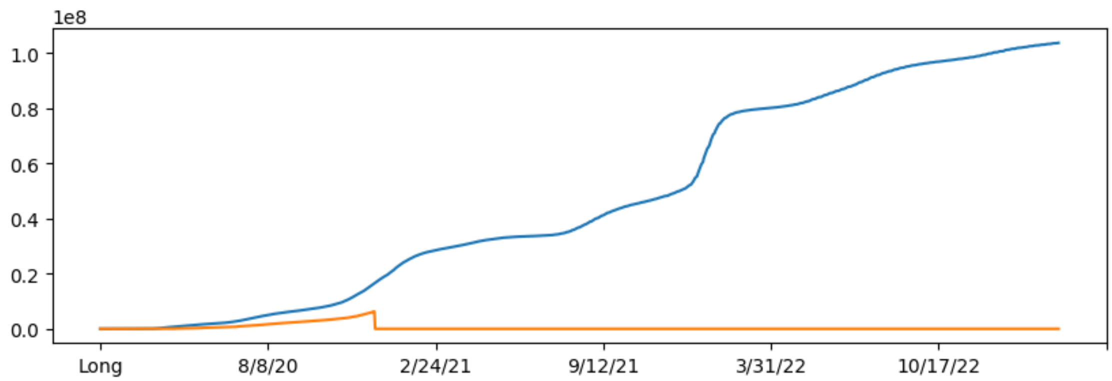
[54]:
```
infected = infected.groupby('Country/Region').sum()
recovered = recovered.groupby('Country/Region').sum()
deaths = deaths.groupby('Country/Region').sum()

infected.head()
```

[54]:
```
               Province/State  1/22/20  1/23/20  1/24/20  1/25/20  1/26/20  \
Country/Region
Afghanistan                 0        0        0        0        0        0
Albania                     0        0        0        0        0        0
Algeria                     0        0        0        0        0        0
Andorra                     0        0        0        0        0        0
Angola                      0        0        0        0        0        0

                1/27/20  1/28/20  1/29/20  1/30/20  …  2/28/23  3/1/23  \
Country/Region                                      …
Afghanistan           0        0        0        0  …   209322  209340
Albania               0        0        0        0  …   334391  334408
Algeria               0        0        0        0  …   271441  271448
Andorra               0        0        0        0  …    47866   47875
Angola                0        0        0        0  …   105255  105277

                3/2/23  3/3/23  3/4/23  3/5/23  3/6/23  3/7/23  3/8/23  3/9/23
Country/Region
Afghanistan     209358  209362  209369  209390  209406  209436  209451  209451
Albania         334408  334427  334427  334427  334427  334427  334443  334457
Algeria         271463  271469  271469  271477  271477  271490  271494  271496
```

```
Andorra            47875   47875   47875   47875   47875   47875   47890   47890
Angola            105277  105277  105277  105277  105277  105277  105288  105288

[5 rows x 1144 columns]
```

You can see that due to using `groupby` all DataFrames are now indexed by Country/Region. We can thus access the data for a specific country by using `.loc`:|

[26]:
```
infected.loc['US'][2:].plot()
recovered.loc['US'][2:].plot()
plt.show()
```



Note how we use `[2:]` to remove first two elements of a sequence that contain geolocation of a country. We can also drop those two columns altogether:

[27]:
```
infected.drop(columns=['Lat','Long'],inplace=True)
recovered.drop(columns=['Lat','Long'],inplace=True)
deaths.drop(columns=['Lat','Long'],inplace=True)
```

[33]:
```
recovered.loc["US"]
```

[33]:
```
Province/State    0
1/22/20           0
1/23/20           0
1/24/20           0
1/25/20           0
                 ..
3/5/23            0
3/6/23            0
3/7/23            0
3/8/23            0
3/9/23            0
Name: US, Length: 1144, dtype: object
```

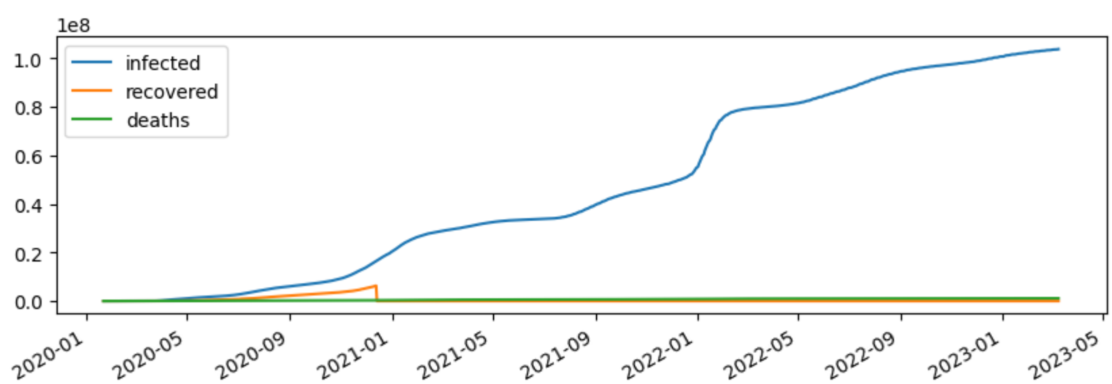
1.4 Investigating the Data

Let's now switch to investigating a specific country. Let's create a frame that contains the data on infections indexed by date:

[88]:
```
def mkframe(country):
    df = pd.DataFrame({ 'infected' : infected.loc[country] ,
                        'recovered' : recovered.loc[country],
                        'deaths' : deaths.loc[country]})
    df.index = pd.to_datetime(df.index, format='%Y-%m-%d',errors = "coerce")
    return df

df = mkframe('US')
df
```

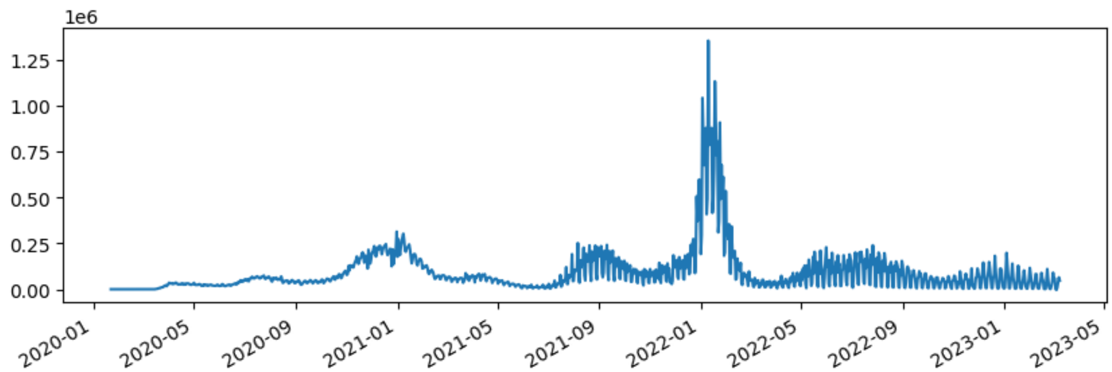
[88]:
```
      infected recovered   deaths
NaT          0         0        0
NaT          1         0        0
NaT          1         0        0
NaT          2         0        0
NaT          2         0        0
..         …         …        …
NaT  103646975         0  1122134
NaT  103655539         0  1122181
NaT  103690910         0  1122516
NaT  103755771         0  1123246
NaT  103802702         0  1123836

[1144 rows x 3 columns]
```

[40]:
```
df.plot()
plt.show()
```



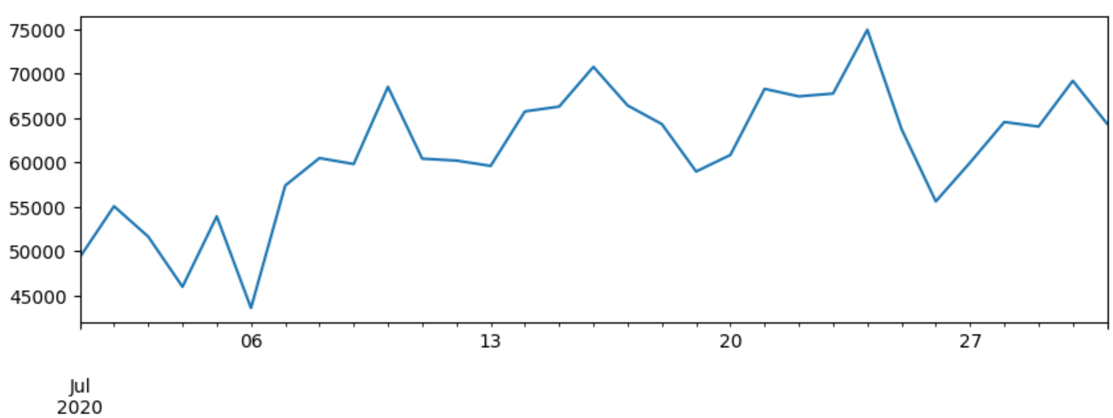
Now let's compute the number of new infected people each day. This will allow us to see the speed at which pandemic progresses. The easiest day to do it is to use `diff`:

[41]:
```
df['ninfected'] = df['infected'].diff()
df['ninfected'].plot()
plt.show()
```



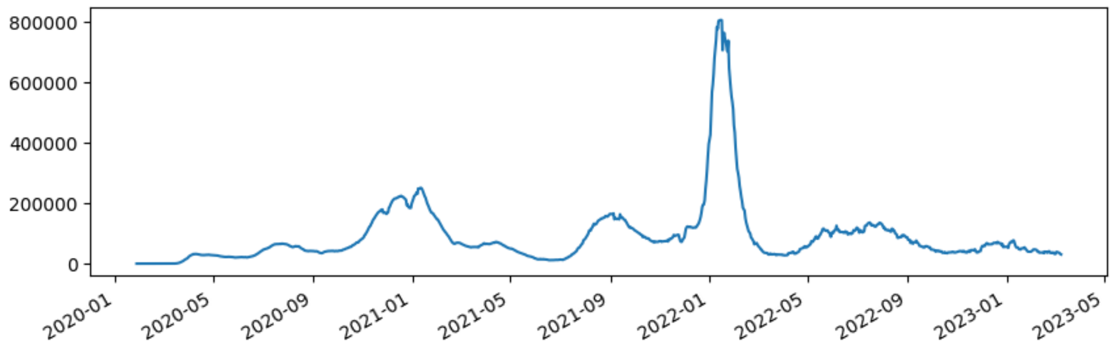
We can see high fluctuations in data. Let's look closer at one of the months:

[42]:
```
df[(df.index.year==2020) & (df.index.month==7)]['ninfected'].plot()
plt.show()
```



It clearly looks like there are weekly fluctuations in data. Because we want to be able to see the trends, it makes sense to smooth out the curve by computing running average (i.e. for each day we will compute the average value of the previous several days):

[43]:
```
df['ninfav'] = df['ninfected'].rolling(window=7).mean()
df['ninfav'].plot()
plt.show()
```



In order to be able to compare several countries, we might want to take the country's population into account, and compare the percentage of infected individuals with respect to country's population. In order to get country's population, let's load the dataset of countries:

[ ]:
```
countries = pd.read_csv(countries_dataset_url)
countries
```

[ ]:
```
            UID iso2 iso3  code3     FIPS      Admin2 Province_State  \
0             4   AF  AFG    4.0      NaN         NaN            NaN
1             8   AL  ALB    8.0      NaN         NaN            NaN
2            12   DZ  DZA   12.0      NaN         NaN            NaN
3            20   AD  AND   20.0      NaN         NaN            NaN
4            24   AO  AGO   24.0      NaN         NaN            NaN
…             …  …   …    …        …         …              …
4191  84056037   US  USA  840.0  56037.0  Sweetwater        Wyoming
4192  84056039   US  USA  840.0  56039.0       Teton        Wyoming
4193  84056041   US  USA  840.0  56041.0       Uinta        Wyoming
4194  84056043   US  USA  840.0  56043.0    Washakie        Wyoming
4195  84056045   US  USA  840.0  56045.0      Weston        Wyoming

     Country_Region        Lat       Long_                Combined_Key  \
0       Afghanistan  33.939110   67.709953                 Afghanistan
1           Albania  41.153300   20.168300                     Albania
2           Algeria  28.033900    1.659600                     Algeria
3           Andorra  42.506300    1.521800                     Andorra
4            Angola -11.202700   17.873900                      Angola
…               …          …           …                         …
4191             US  41.659439 -108.882788  Sweetwater, Wyoming, US
4192             US  43.935225 -110.589080       Teton, Wyoming, US
4193             US  41.287818 -110.547578       Uinta, Wyoming, US
4194             US  43.904516 -107.680187    Washakie, Wyoming, US
4195             US  43.839612 -104.567488      Weston, Wyoming, US

      Population
```

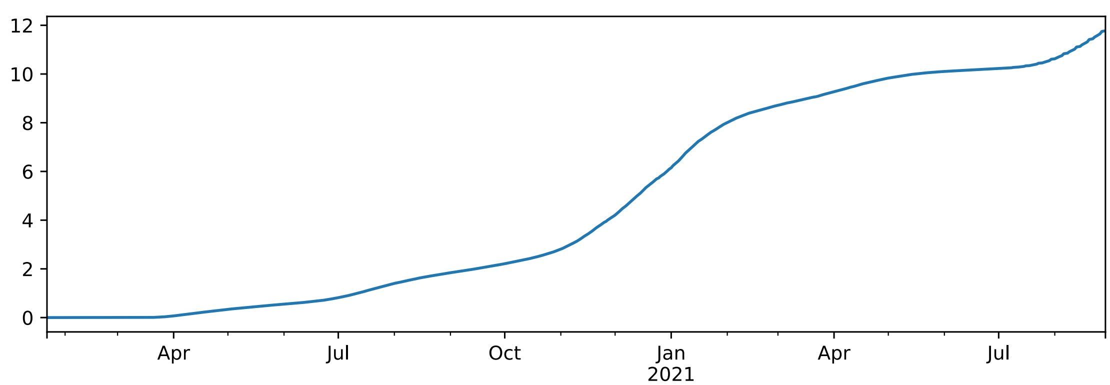
```
0       38928341.0
1        2877800.0
2       43851043.0
3          77265.0
4       32866268.0
…            …
4191       42343.0
4192       23464.0
4193       20226.0
4194        7805.0
4195        6927.0

[4196 rows x 12 columns]
```

Because this dataset contains information on both countries and provinces, to get the population of the whole country we need to be a little bit clever:

```
[ ]: countries[(countries['Country_Region']=='US') & countries['Province_State'].
      ↪isna()]
```

```
[ ]:      UID iso2 iso3  code3  FIPS Admin2 Province_State Country_Region   Lat  \
     790  840   US  USA  840.0   NaN    NaN            NaN             US  40.0

          Long_ Combined_Key   Population
     790 -100.0           US  329466283.0
```

```
[ ]: pop = countries[(countries['Country_Region']=='US') &␣
      ↪countries['Province_State'].isna()]['Population'].iloc[0]
     df['pinfected'] = df['infected']*100 / pop
     df['pinfected'].plot(figsize=(10,3))
     plt.show()
```



1.5 Computing R_t

To see how infectious is the disease, we look at the **basic reproduction number** R_0, which indicated the number of people that an infected person would further infect. When R_0 is more than 1, the epidemic is likely to spread.

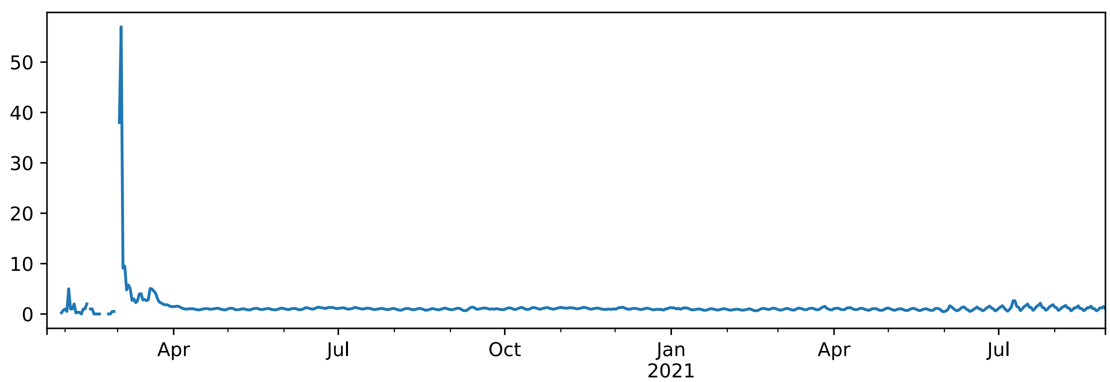
R_0 is a property of the disease itself, and does not take into account some protective measures that people may take to slow down the pandemic. During the pandemic progression, we can estimate the reproduction number R_t at any given time t. It has been shown that this number can be roughly estimated by taking a window of 8 days, and computing

$$R_t = \frac{I_{t-7} + I_{t-6} + I_{t-5} + I_{t-4}}{I_{t-3} + I_{t-2} + I_{t-1} + I_t}$$

where I_t is the number of newly infected individuals on day t.

Let's compute R_t for our pandemic data. To do this, we will take a rolling window of 8 `ninfected` values, and apply the function to compute the ratio above:

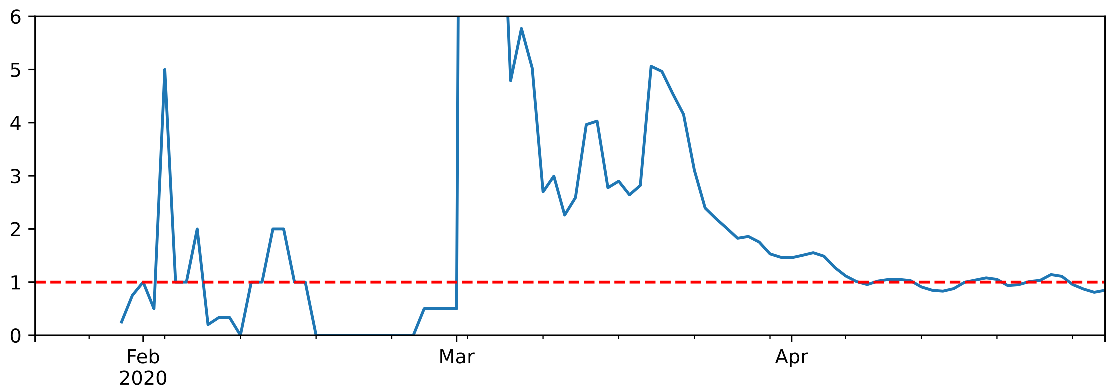
```
[ ]: df['Rt'] = df['ninfected'].rolling(8).apply(lambda x: x[4:].sum()/x[:4].sum())
     df['Rt'].plot()
     plt.show()
```



You can see that there are some gaps in the graph. Those can be caused by either `NaN`, if `inf` values being present in the dataset. `inf` may be caused by division by 0, and `NaN` can indicate missing data, or no data available to compute the result (like in the very beginning of our frame, where rolling window of width 8 is not yet available). To make the graph nicer, we need to fill those values using `replace` and `fillna` function.

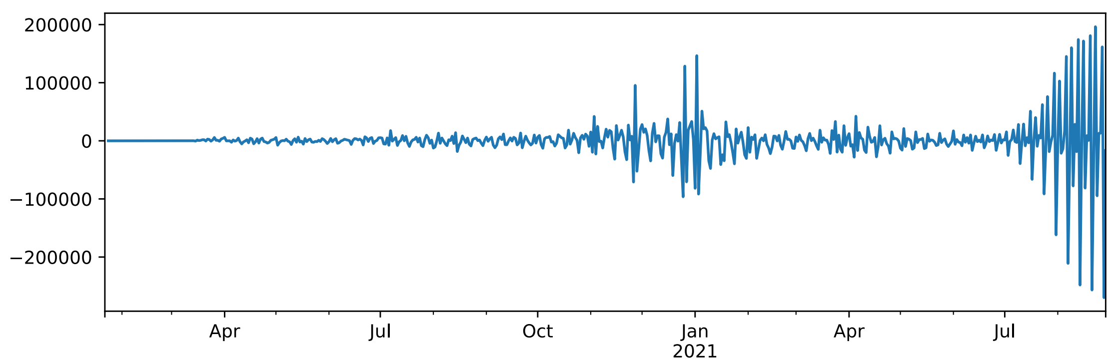
Let's further look at the beginning of the pandemic. We will also limit the y-axis values to show only values below 6, in order to see better, and draw horizontal line at 1.

```
[ ]: ax = df[df.index<"2020-05-01"]['Rt'].replace(np.inf,np.nan).
       ↪fillna(method='pad').plot(figsize=(10,3))
     ax.set_ylim([0,6])
     ax.axhline(1,linestyle='--',color='red')
     plt.show()
```



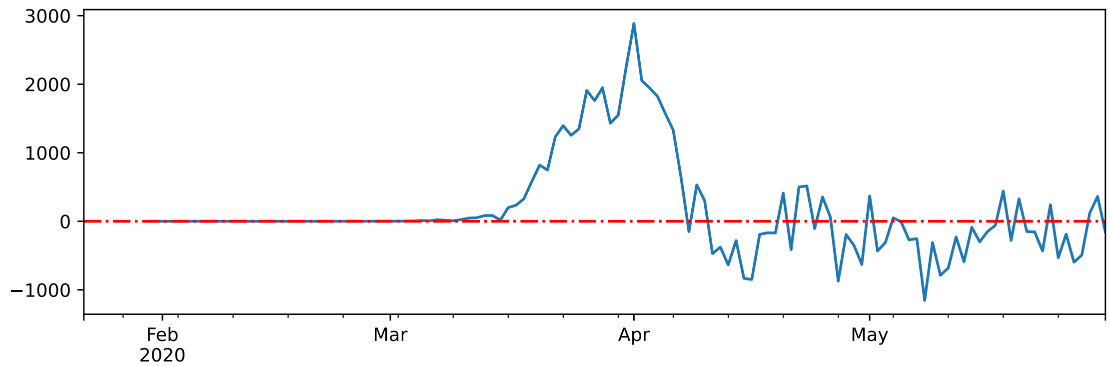
Another interesting indicator of the pandemic is the **derivative**, or **daily difference** in new cases. It allows us to see clearly when pandemic is increasing or declining.

[ ]:
```
df['ninfected'].diff().plot()
plt.show()
```



Given the fact that there are a lot of fluctuations in data caused by reporting, it makes sense to smooth the curve by running rolling average to get the overall picture. Let's again focus on the first months of the pandemic:

[ ]:
```
ax=df[df.index<"2020-06-01"]['ninfected'].diff().rolling(7).mean().plot()
ax.axhline(0,linestyle='-.',color='red')
plt.show()
```



1.6 Challenge

Now it is time for you to play more with the code and data! Here are a few suggestions you can experiment with: * See the spread of the pandemic in different countries. * Plot R_t graphs for several countries on one plot for comparison, or make several plots side-by-side * See how the number of deaths and recoveries correlate with number of infected cases. * Try to find out how long a typical disease lasts by visually correlating infection rate and deaths rate and looking for some anomalies. You may need to look at different countries to find that out. * Calculate the fatality rate and how it changes over time. You may want to take into account the length of the disease in days to shift one time series before doing calculations

1.7 References

You may look at further studies of COVID epidemic spread in the following publications: * Sliding SIR Model for Rt Estimation during COVID Pandemic, blog post by Dmitry Soshnikov * T.Petrova, D.Soshnikov, A.Grunin. Estimation of Time-Dependent Reproduction Number for Global COVID-19 Outbreak. *Preprints* **2020**, 2020060289 (doi: 10.20944/preprints202006.0289.v1) * Code for the above paper on GitHub

2 Observing the Spread of the Pandemic in different countries

Here, we will analyze covid spread in 6 countries, namely **Philippines**, **India**, **China**, **Italy**, **South Korea**, and **Japan**.

[99]:
```
infected.head()
```

[99]:
```
                Province/State  1/22/20  1/23/20  1/24/20  1/25/20  1/26/20  \
Country/Region
Afghanistan                  0        0        0        0        0        0
Albania                      0        0        0        0        0        0
Algeria                      0        0        0        0        0        0
Andorra                      0        0        0        0        0        0
Angola                       0        0        0        0        0        0
```

```
                1/27/20  1/28/20  1/29/20  1/30/20  …  2/28/23  3/1/23  \
Country/Region                                      …
Afghanistan           0        0        0        0  …   209322  209340
Albania               0        0        0        0  …   334391  334408
Algeria               0        0        0        0  …   271441  271448
Andorra               0        0        0        0  …    47866   47875
Angola                0        0        0        0  …   105255  105277

                3/2/23  3/3/23  3/4/23  3/5/23  3/6/23  3/7/23  3/8/23  3/9/23
Country/Region
Afghanistan     209358  209362  209369  209390  209406  209436  209451  209451
Albania         334408  334427  334427  334427  334427  334427  334443  334457
Algeria         271463  271469  271469  271477  271477  271490  271494  271496
Andorra          47875   47875   47875   47875   47875   47875   47890   47890
Angola          105277  105277  105277  105277  105277  105277  105288  105288

[5 rows x 1144 columns]
```

[100]:
```python
def mkframe(country):
    df = pd.DataFrame({ 'infected' : infected.loc[country] ,
                        'recovered' : recovered.loc[country],
                        'deaths' : deaths.loc[country]})
    df.index = pd.to_datetime(df.index, format='%m/%d/%y',errors = "coerce")
    return df

df = mkframe('US')
df
```

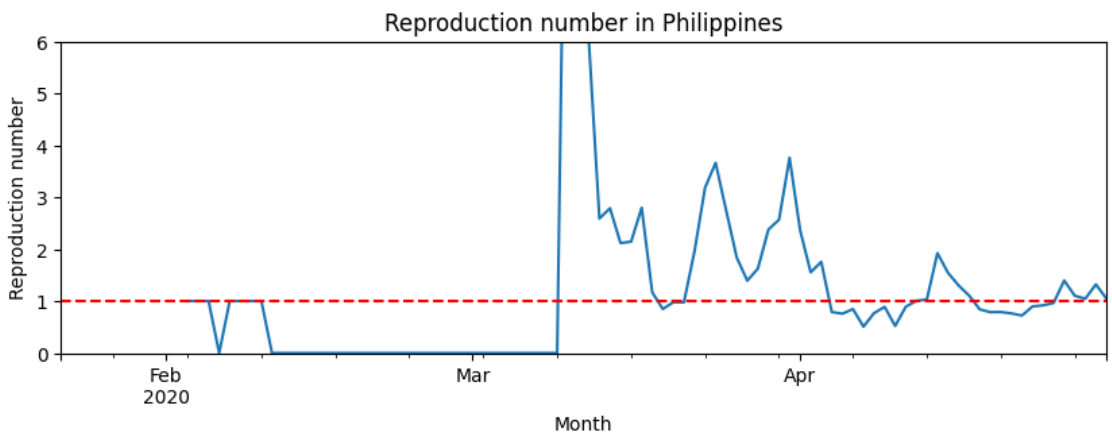
[100]:
```
             infected recovered   deaths
NaT                 0         0        0
2020-01-22          1         0        0
2020-01-23          1         0        0
2020-01-24          2         0        0
2020-01-25          2         0        0
…                 …        …       …
2023-03-05  103646975         0  1122134
2023-03-06  103655539         0  1122181
2023-03-07  103690910         0  1122516
2023-03-08  103755771         0  1123246
2023-03-09  103802702         0  1123836

[1144 rows x 3 columns]
```

[101]:
```python
analyze_countries = ["Philippines", "India", "Russia", "Italy", "Korea, South",␣
 ↪"Japan"]
print("Countries to analyze:", analyze_countries)
for country in analyze_countries:
```

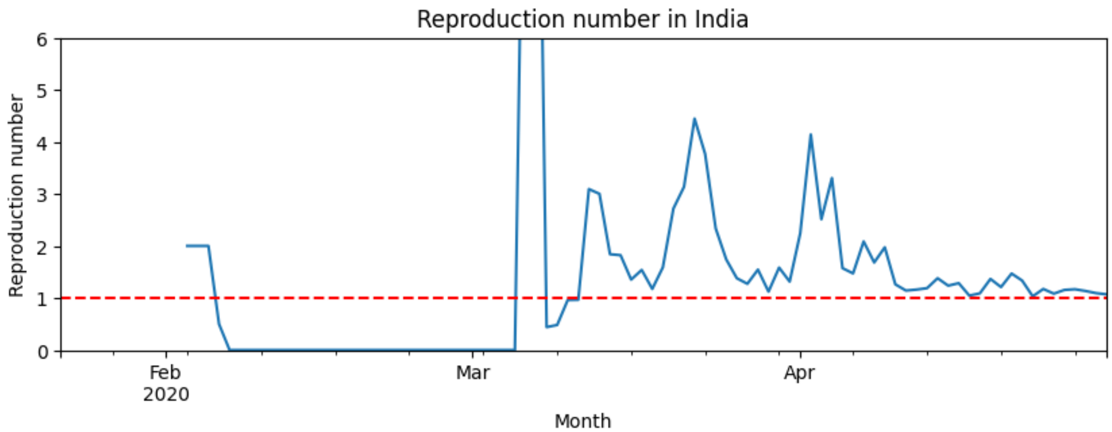
```
    df = mkframe(country)
    df['ninfected'] = df['infected'].diff()
    df['Rt'] = df['ninfected'].rolling(8).apply(lambda x: x[4:].sum()/x[:4].
↪sum())
    ax = df[df.index<"2020-05-01"]['Rt'].replace(np.inf,np.nan).
↪fillna(method='pad').plot(figsize=(10,3))
    ax.set_ylim([0,6])
    ax.axhline(1,linestyle='--',color='red')
    plt.title("Reproduction number in "+country)
    plt.xlabel("Month")
    plt.ylabel("Reproduction number")

    plt.show()
```

```
Countries to analyze: ['Philippines', 'India', 'Russia', 'Italy', 'Korea,
South', 'Japan']
```

```
/tmp/ipykernel_14391/202162943.py:7: FutureWarning: Series.fillna with 'method'
is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill()
instead.
  ax = df[df.index<"2020-05-
01"]['Rt'].replace(np.inf,np.nan).fillna(method='pad').plot(figsize=(10,3))
```

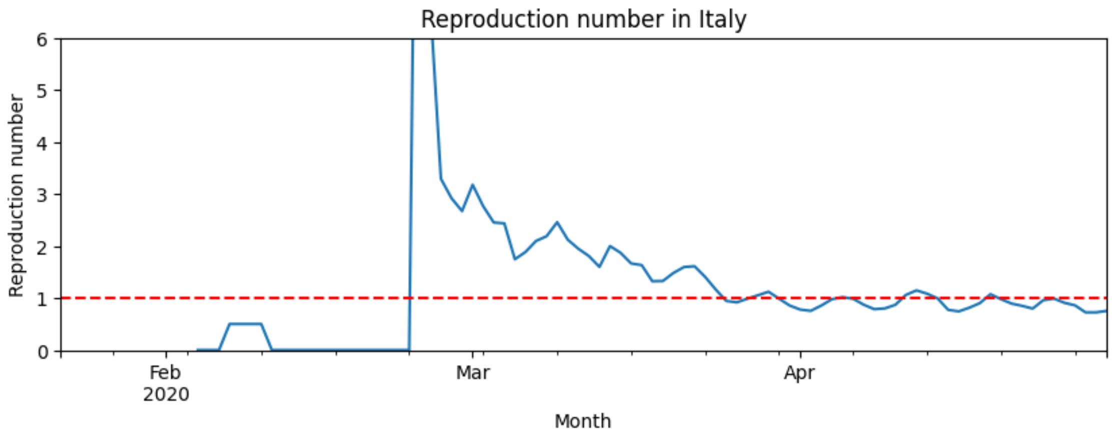



```
/tmp/ipykernel_14391/202162943.py:7: FutureWarning: Series.fillna with 'method'
is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill()
instead.
  ax = df[df.index<"2020-05-
01"]['Rt'].replace(np.inf,np.nan).fillna(method='pad').plot(figsize=(10,3))
```

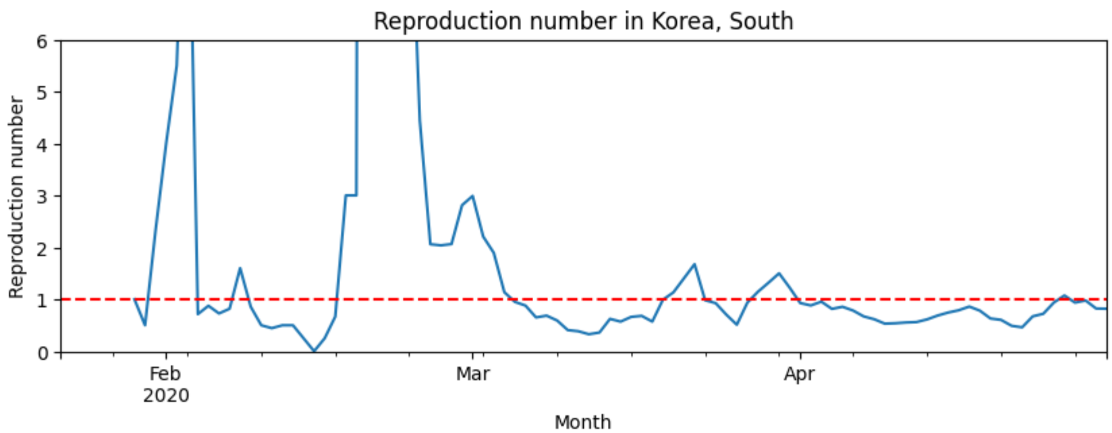



```
/tmp/ipykernel_14391/202162943.py:7: FutureWarning: Series.fillna with 'method'
is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill()
instead.
  ax = df[df.index<"2020-05-
01"]['Rt'].replace(np.inf,np.nan).fillna(method='pad').plot(figsize=(10,3))
```

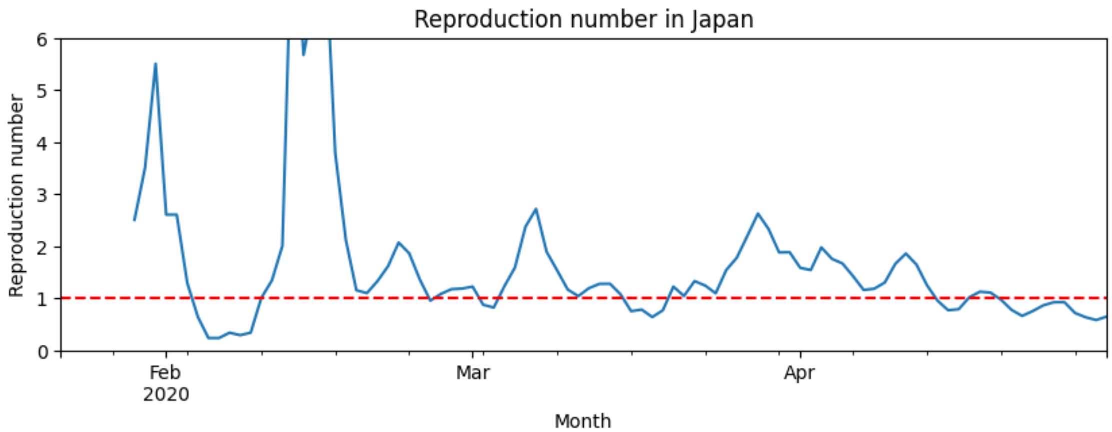

```
/tmp/ipykernel_14391/202162943.py:7: FutureWarning: Series.fillna with 'method'
is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill()
instead.
  ax = df[df.index<"2020-05-
01"]['Rt'].replace(np.inf,np.nan).fillna(method='pad').plot(figsize=(10,3))
```



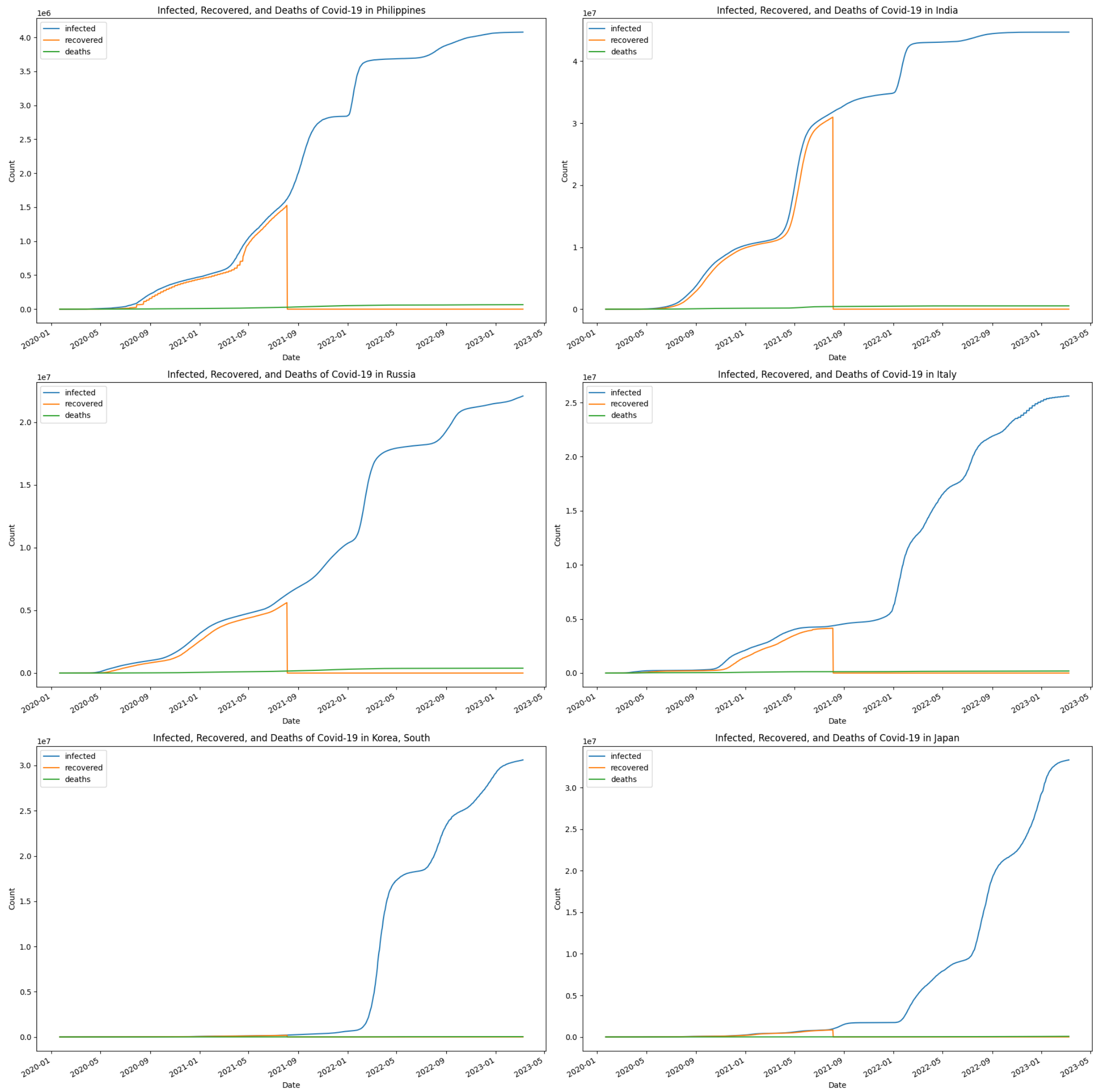

```
/tmp/ipykernel_14391/202162943.py:7: FutureWarning: Series.fillna with 'method'
is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill()
instead.
  ax = df[df.index<"2020-05-
01"]['Rt'].replace(np.inf,np.nan).fillna(method='pad').plot(figsize=(10,3))
```




```
/tmp/ipykernel_14391/202162943.py:7: FutureWarning: Series.fillna with 'method'
is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill()
instead.
  ax = df[df.index<"2020-05-
01"]['Rt'].replace(np.inf,np.nan).fillna(method='pad').plot(figsize=(10,3))
```



[102]:
```
fig, axs = plt.subplots(3, 2, figsize = (20,20))
for country in analyze_countries:
    df = mkframe(country)
    row = analyze_countries.index(country) // 2
    col = analyze_countries.index(country) % 2
    ax = axs[row, col]
    df.plot(ax=ax)
    ax.set_title(f"Infected, Recovered, and Deaths of Covid-19 in {country}")
    ax.set_xlabel("Date")
    ax.set_ylabel("Count")

plt.tight_layout()
plt.show()
```



2.1 Relationship between number of Infected to the number of Recovered and Death

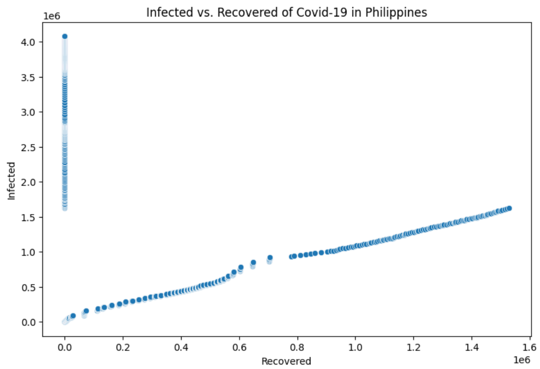
From the graph, we can observe that the deaths follows an almost horizontal line. This is due to the scale of the y-axis being affected by the number of the infected. It shows that there is a huge difference between the number of deaths and the infected, leaving the plot of the number of deaths basically useless in determining the relationship between the two. Nevertheless, we can see that, at least, for the countries South Korea, Japan, and Russia,the number of those infected spiked when the number of those who recovered showed a decline.

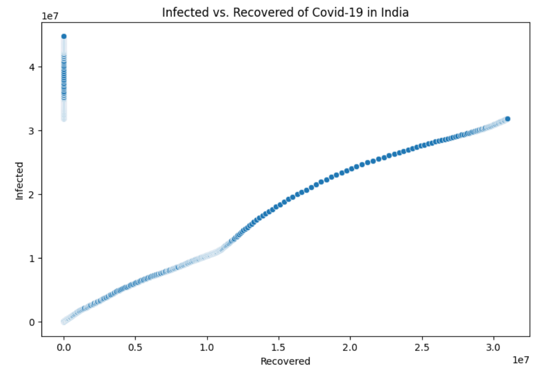
[106]:
```
import seaborn as sns
fig, axs = plt.subplots(3, 2, figsize = (20,20))
```

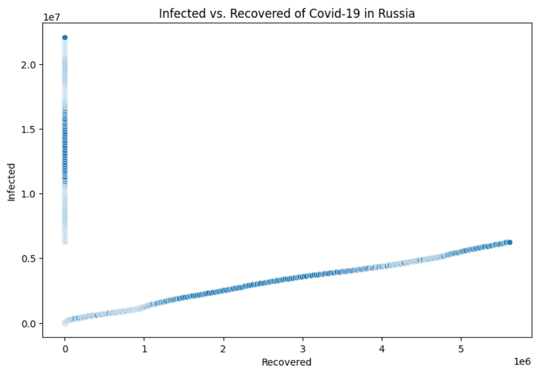
```
print("Graph of infected vs. recovered data from the 6 countries and their␣
↪respective correction")
for country in analyze_countries:
    df = mkframe(country)
    ax = axs[analyze_countries.index(country) // 2, analyze_countries.
↪index(country) % 2]
    sns.scatterplot(data=df, x="recovered", y="infected", ax = ax)

    ax.set_title(f"Infected vs. Recovered of Covid-19 in {country}")
    ax.set_xlabel("Recovered")
    ax.set_ylabel("Infected")
    print(f"Correlation for {country}: {df['infected'].corr(df['recovered'])}")

plt.plot()
```

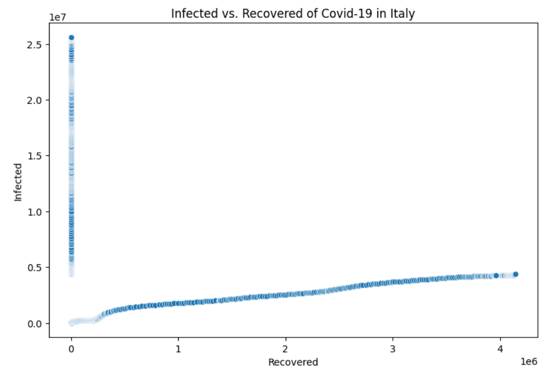
```
Graph of infected vs. recovered data from the 6 countries and their respective
correction
Correlation for Philippines: -0.34364097669551025
Correlation for India: -0.15991442913802678
Correlation for Russia: -0.3800006340779427
Correlation for Italy: -0.31171778849439463
Correlation for Korea, South: -0.37340772595720256
Correlation for Japan: -0.3071647440301201
```

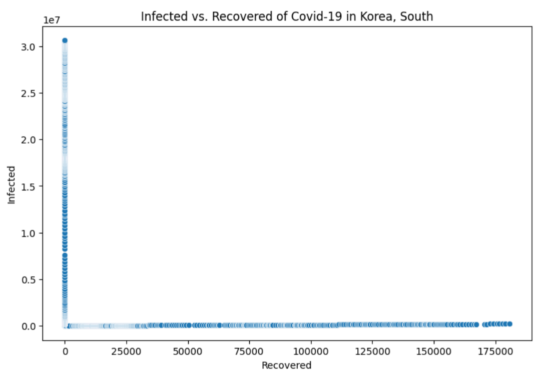
[106]: []

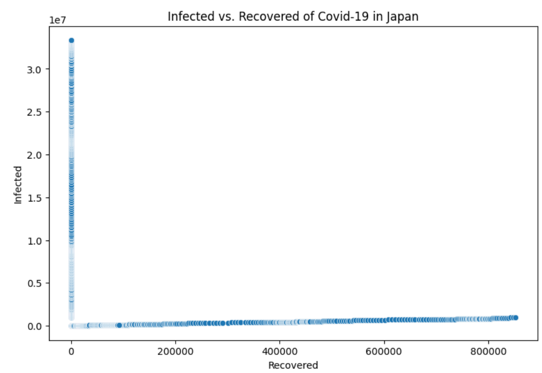












We can see that although some countries show a positive trend, the correlation coefficient shows a negative value for all the countries, which is counterintuitive. The reason for this is because of the vertical line when the number of recovery is approximately, where the number of infected drastically rise. This is also backed up by the plot of each country addressing the number of infected, recovered, and death.

```
[ ]:
```