COVID19 EXPLORATORY DATA ANALYSIS WITH PYTHON

This notebook conatins an Exploratory Data Analysis of the pandemic Covid19 based on the datas
University(https://github.com/CSSEGISandData/COVID-19). I have made no statistical or predictiv
sesitivity of the situation and some problems that predictions can create.

The main reasons for using the JHU data are:

JHU is already a trusted and respected institution, They cite many sources, which are themselves
provided directly in the github repository (.csv in a github repository).

Exploratory data analysis and visualization using Python

Imports and data

Let's import the necessary packages from the SciPy stack and get the data.

```
# Import packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Set style & figures inline
sns.set()
%matplotlib inline
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarnin
  import pandas.util.testing as tm
```

```
# Data urls
base_url = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19
confirmed_cases_data_url = base_url + 'time_series_covid19_confirmed_global.csv'
death_cases_data_url = base_url + 'time_series_covid19_deaths_global.csv'
recovery_cases_data_url = base_url+ 'time_series_covid19_recovered_global.csv'

# Import datasets as pandas dataframes
raw_confirmed_df = pd.read_csv(confirmed_cases_data_url)
raw_deaths_df = pd.read_csv(death_cases_data_url)
raw_recovered_df = pd.read_csv(recovery_cases_data_url)
```

Analysing the Confirmed cases of COVID-19

```
raw_confirmed_df.head()
```

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/2
0	NaN	Afghanistan	33.0000	65.0000	0	0	0	
1	NaN	Albania	41.1533	20.1683	0	0	0	
2	NaN	Algeria	28.0339	1.6596	0	0	0	
3	NaN	Andorra	42.5063	1.5218	0	0	0	
4	NaN	Angola	-11.2027	17.8739	0	0	0	

5 rows × 85 columns

Using .info() and .describe()

```
raw_confirmed_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 264 entries, 0 to 263
Data columns (total 85 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Province/State  82 non-null     object 
 1   Country/Region  264 non-null    object 
 2   Lat             264 non-null    float64
 3   Long            264 non-null    float64
 4   1/22/20         264 non-null    int64  
 5   1/23/20         264 non-null    int64  
 6   1/24/20         264 non-null    int64  
 7   1/25/20         264 non-null    int64  
 8   1/26/20         264 non-null    int64  
 9   1/27/20         264 non-null    int64  
 10  1/28/20         264 non-null    int64  
 11  1/29/20         264 non-null    int64  
 12  1/30/20         264 non-null    int64  
 13  1/31/20         264 non-null    int64  
 14  2/1/20          264 non-null    int64  
 15  2/2/20          264 non-null    int64  
 16  2/3/20          264 non-null    int64  
 17  2/4/20          264 non-null    int64  
 18  2/5/20          264 non-null    int64  
 19  2/6/20          264 non-null    int64  
 20  2/7/20          264 non-null    int64  
 21  2/8/20          264 non-null    int64  
 22  2/9/20          264 non-null    int64  
 23  2/10/20         264 non-null    int64  
 24  2/11/20         264 non-null    int64  
 25  2/12/20         264 non-null    int64  
 26  2/13/20         264 non-null    int64  
 27  2/14/20         264 non-null    int64  
 28  2/15/20         264 non-null    int64  
 29  2/16/20         264 non-null    int64  
 30  2/17/20         264 non-null    int64  
 31  2/18/20         264 non-null    int64  
 32  2/19/20         264 non-null    int64  
 33  2/20/20         264 non-null    int64  
 34  2/21/20         264 non-null    int64  
 35  2/22/20         264 non-null    int64  
 36  2/23/20         264 non-null    int64  
 37  2/24/20         264 non-null    int64  
 38  2/25/20         264 non-null    int64  
 39  2/26/20         264 non-null    int64  
 40  2/27/20         264 non-null    int64  
 41  2/28/20         264 non-null    int64  
 42  2/29/20         264 non-null    int64  
 43  3/1/20          264 non-null    int64  
 44  3/2/20          264 non-null    int64  
 45  3/3/20          264 non-null    int64  
 46  3/4/20          264 non-null    int64  
 47  3/5/20          264 non-null    int64  
 48  3/6/20          264 non-null    int64  
 49  3/7/20          264 non-null    int64  
 50  3/8/20          264 non-null    int64  
 51  3/9/20          264 non-null    int64  
 52  3/10/20         264 non-null    int64  
 53  3/11/20         264 non-null    int64  
 54  3/12/20         264 non-null    int64  
 55  3/13/20         264 non-null    int64  
```

```
 56  3/14/20         264 non-null    int64
 57  3/15/20         264 non-null    int64
 58  3/16/20         264 non-null    int64
 59  3/17/20         264 non-null    int64
 60  3/18/20         264 non-null    int64
 61  3/19/20         264 non-null    int64
 62  3/20/20         264 non-null    int64
 63  3/21/20         264 non-null    int64
 64  3/22/20         264 non-null    int64
 65  3/23/20         264 non-null    int64
 66  3/24/20         264 non-null    int64
 67  3/25/20         264 non-null    int64
 68  3/26/20         264 non-null    int64
 69  3/27/20         264 non-null    int64
 70  3/28/20         264 non-null    int64
 71  3/29/20         264 non-null    int64
 72  3/30/20         264 non-null    int64
 73  3/31/20         264 non-null    int64
 74  4/1/20          264 non-null    int64
 75  4/2/20          264 non-null    int64
 76  4/3/20          264 non-null    int64
 77  4/4/20          264 non-null    int64
 78  4/5/20          264 non-null    int64
 79  4/6/20          264 non-null    int64
 80  4/7/20          264 non-null    int64
 81  4/8/20          264 non-null    int64
 82  4/9/20          264 non-null    int64
 83  4/10/20         264 non-null    int64
 84  4/11/20         264 non-null    int64
dtypes: float64(2), int64(81), object(2)
memory usage: 175.4+ KB
```

```
raw_confirmed_df.describe()
```

	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26,
count	264.000000	264.000000	264.000000	264.000000	264.000000	264.000000	264.000(
mean	21.317326	22.168315	2.102273	2.477273	3.564394	5.431818	8.022
std	24.734994	70.669996	27.382118	27.480921	34.210982	47.612615	66.537
min	-51.796300	-135.000000	0.000000	0.000000	0.000000	0.000000	0.000(
25%	6.969250	-20.026050	0.000000	0.000000	0.000000	0.000000	0.000(
50%	23.488100	20.535638	0.000000	0.000000	0.000000	0.000000	0.000(
75%	41.166075	78.750000	0.000000	0.000000	0.000000	0.000000	0.000(
max	71.706900	178.065000	444.000000	444.000000	549.000000	761.000000	1058.000(

8 rows × 83 columns

Number of confirmed cases by country

```
raw_confirmed_df.tail()
```

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20
259	Saint Pierre and Miquelon	France	46.885200	-56.315900	0	0	0
260	NaN	South Sudan	6.877000	31.307000	0	0	0
261	NaN	Western Sahara	24.215500	-12.885800	0	0	0
262	NaN	Sao Tome and Principe	0.186360	6.613081	0	0	0
263	NaN	Yemen	15.552727	48.516388	0	0	0

5 rows × 85 columns

```
raw_confirmed_df.head()
```

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/2
0	NaN	Afghanistan	33.0000	65.0000	0	0	0	
1	NaN	Albania	41.1533	20.1683	0	0	0	
2	NaN	Algeria	28.0339	1.6596	0	0	0	
3	NaN	Andorra	42.5063	1.5218	0	0	0	
4	NaN	Angola	-11.2027	17.8739	0	0	0	

5 rows × 85 columns

From the head and tail observations, its visible that each entry contains the data belonging to the I

We will take all the rows (*regions/provinces*) that correspond to that country and add up the numb speak, we want to **group by** the country column and sum up all the values for the other columns.

```
# Group by region (we'll also drop 'Lat', 'Long' as it doesn't make sense to sum them here
confirmed_df = raw_confirmed_df.groupby(['Country/Region']).sum().drop(["Lat", "Long"], ax
confirmed_df.head()
```

	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	1/28/20	1/29/2(
Country/Region								
Afghanistan	0	0	0	0	0	0	0	(
Albania	0	0	0	0	0	0	0	(
Algeria	0	0	0	0	0	0	0	(
Andorra	0	0	0	0	0	0	0	(
Angola	0	0	0	0	0	0	0	(

5 rows × 81 columns

So each row of our new dataframe `confirmed_df` is a time series of the number of confirmed case the index of our dataframe.

```
confirmed_df.index
```

```
Index(['Afghanistan', 'Albania', 'Algeria', 'Andorra', 'Angola',
       'Antigua and Barbuda', 'Argentina', 'Armenia', 'Australia', 'Austria',
       ...
       'United Kingdom', 'Uruguay', 'Uzbekistan', 'Venezuela', 'Vietnam',
       'West Bank and Gaza', 'Western Sahara', 'Yemen', 'Zambia', 'Zimbabwe'],
      dtype='object', name='Country/Region', length=185)
```

It's indexed by `Country/Region`. That's all good **but** if we index by date **instead**, it will allow us to p immediately.

To make the index the set of dates, notice that the column names are the dates. To turn column na make the columns the rows and vice versa. This corresponds to taking the transpose of the datafr

```
confirmed_df = confirmed_df.transpose()
confirmed_df.head()
```

Now, let's have a look at our index to see whether it actually consists of DateTimes or not

```
confirmed_df.index
```

```
Index(['1/22/20', '1/23/20', '1/24/20', '1/25/20', '1/26/20', '1/27/20',
       '1/28/20', '1/29/20', '1/30/20', '1/31/20', '2/1/20', '2/2/20',
       '2/3/20', '2/4/20', '2/5/20', '2/6/20', '2/7/20', '2/8/20', '2/9/20',
       '2/10/20', '2/11/20', '2/12/20', '2/13/20', '2/14/20', '2/15/20',
       '2/16/20', '2/17/20', '2/18/20', '2/19/20', '2/20/20', '2/21/20',
       '2/22/20', '2/23/20', '2/24/20', '2/25/20', '2/26/20', '2/27/20',
       '2/28/20', '2/29/20', '3/1/20', '3/2/20', '3/3/20', '3/4/20', '3/5/20',
       '3/6/20', '3/7/20', '3/8/20', '3/9/20', '3/10/20', '3/11/20', '3/12/20',
       '3/13/20', '3/14/20', '3/15/20', '3/16/20', '3/17/20', '3/18/20',
       '3/19/20', '3/20/20', '3/21/20', '3/22/20', '3/23/20', '3/24/20',
       '3/25/20', '3/26/20', '3/27/20', '3/28/20', '3/29/20', '3/30/20',
       '3/31/20', '4/1/20', '4/2/20', '4/3/20', '4/4/20', '4/5/20', '4/6/20',
       '4/7/20', '4/8/20', '4/9/20', '4/10/20', '4/11/20'],
      dtype='object')
```

Note that `dtype='object'` which means that these are strings, not DateTimes. We will use `pandas`

```
# Set index as DateTimeIndex
datetime_index = pd.DatetimeIndex(confirmed_df.index)
confirmed_df.set_index(datetime_index, inplace=True)
# Check out index
confirmed_df.index
```

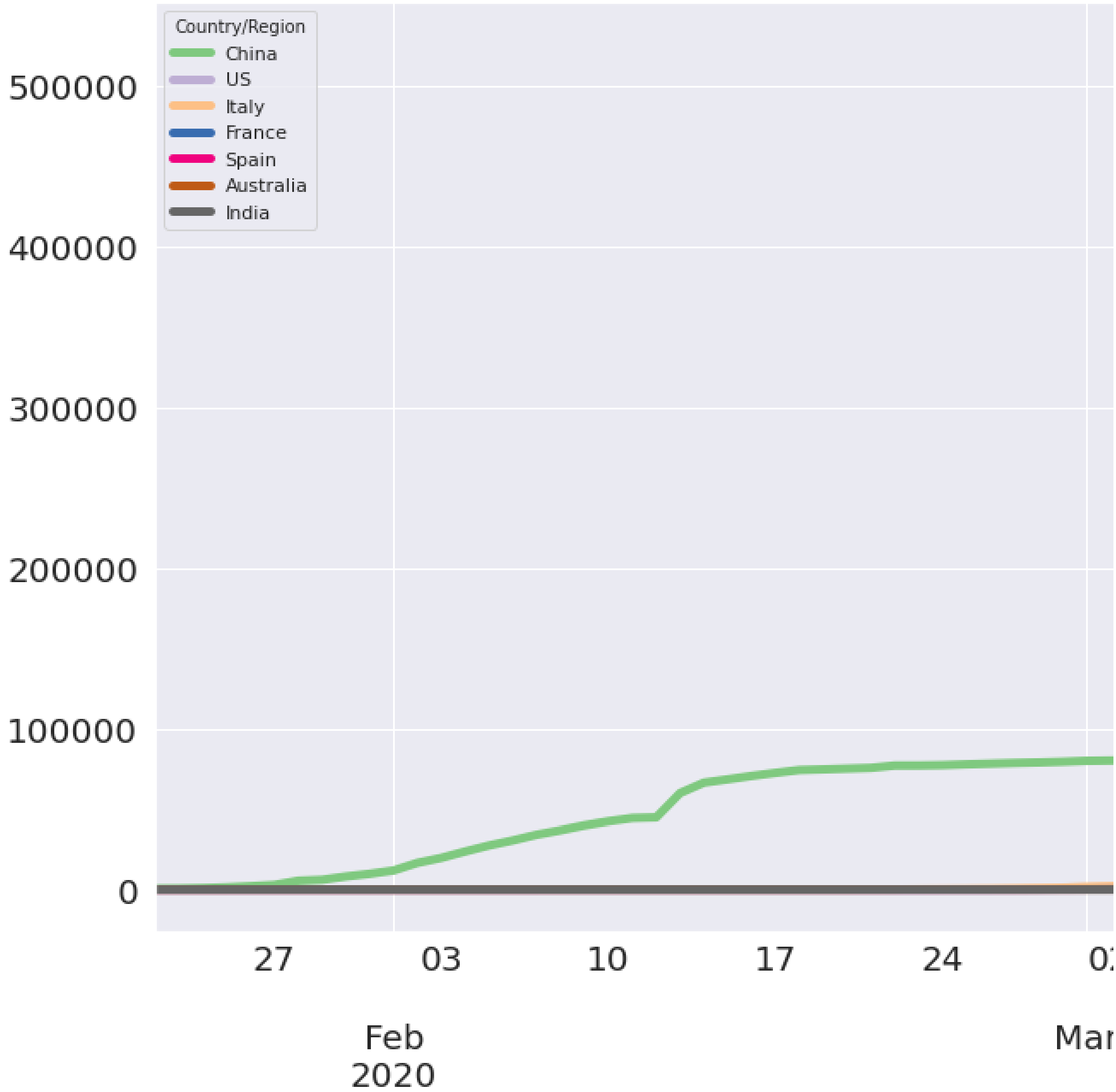
```
DatetimeIndex(['2020-01-22', '2020-01-23', '2020-01-24', '2020-01-25',
               '2020-01-26', '2020-01-27', '2020-01-28', '2020-01-29',
               '2020-01-30', '2020-01-31', '2020-02-01', '2020-02-02',
               '2020-02-03', '2020-02-04', '2020-02-05', '2020-02-06',
               '2020-02-07', '2020-02-08', '2020-02-09', '2020-02-10',
               '2020-02-11', '2020-02-12', '2020-02-13', '2020-02-14',
               '2020-02-15', '2020-02-16', '2020-02-17', '2020-02-18',
               '2020-02-19', '2020-02-20', '2020-02-21', '2020-02-22',
               '2020-02-23', '2020-02-24', '2020-02-25', '2020-02-26',
               '2020-02-27', '2020-02-28', '2020-02-29', '2020-03-01',
               '2020-03-02', '2020-03-03', '2020-03-04', '2020-03-05',
               '2020-03-06', '2020-03-07', '2020-03-08', '2020-03-09',
               '2020-03-10', '2020-03-11', '2020-03-12', '2020-03-13',
               '2020-03-14', '2020-03-15', '2020-03-16', '2020-03-17',
               '2020-03-18', '2020-03-19', '2020-03-20', '2020-03-21',
               '2020-03-22', '2020-03-23', '2020-03-24', '2020-03-25',
               '2020-03-26', '2020-03-27', '2020-03-28', '2020-03-29',
               '2020-03-30', '2020-03-31', '2020-04-01', '2020-04-02',
               '2020-04-03', '2020-04-04', '2020-04-05', '2020-04-06',
               '2020-04-07', '2020-04-08', '2020-04-09', '2020-04-10',
               '2020-04-11'],
              dtype='datetime64[ns]', freq=None)
```

Now we have a DateTimeIndex and Countries for columns, we can use the dataframe plotting met
number of cases by country.

Plotting confirmed cases by country

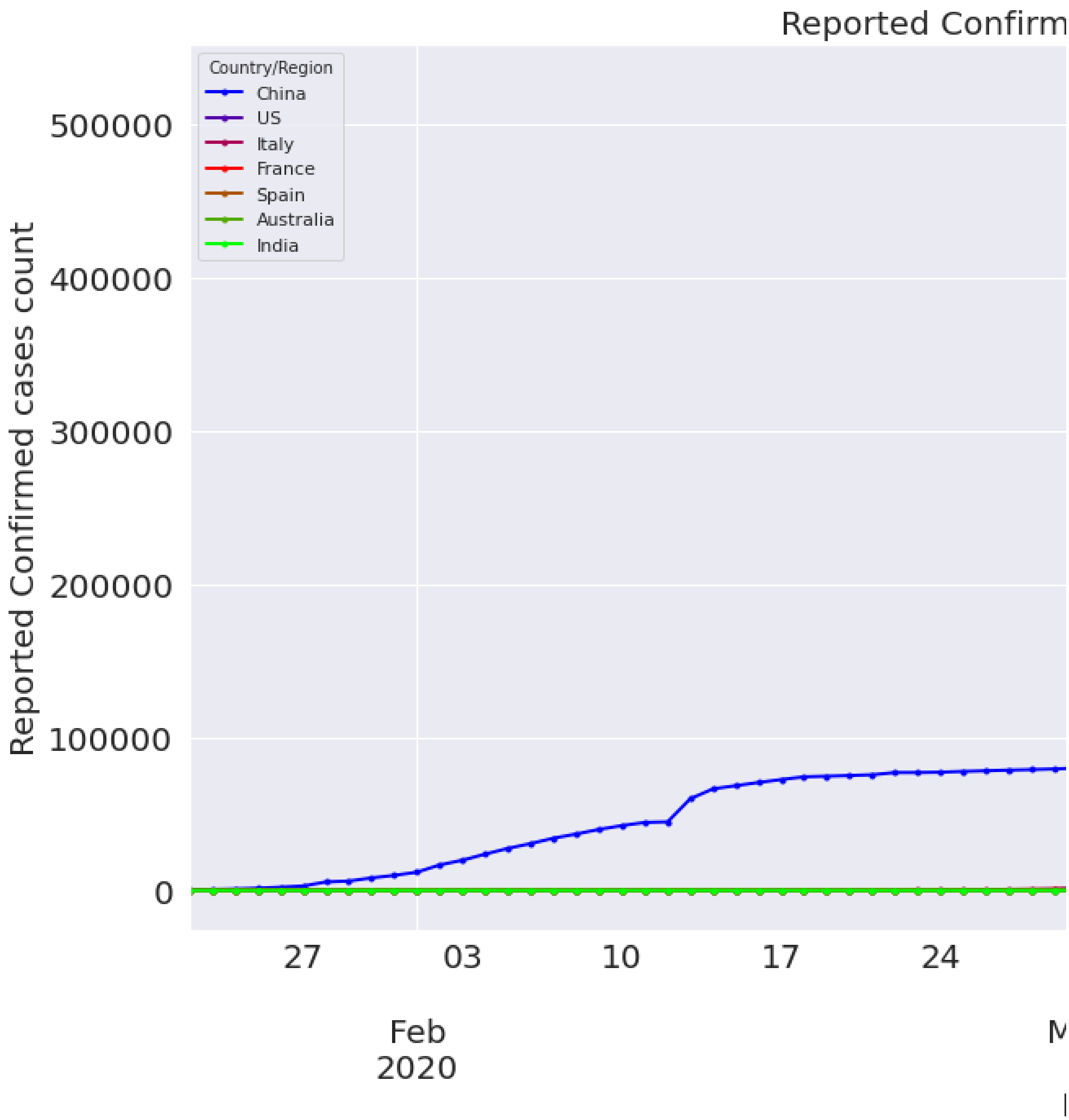
```
# Plotting time series of several countries (as plotting for all the countries will make t
countries = ['China', 'US', 'Italy', 'France', 'Spain', 'Australia', 'India']
confirmed_df[countries].plot(figsize=(20,10), linewidth=5, colormap='Accent', fontsize=20)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3369fe75c0>
```



Now, Let's label our axes and give the figure a title. We'll also thin the line and add points for the da

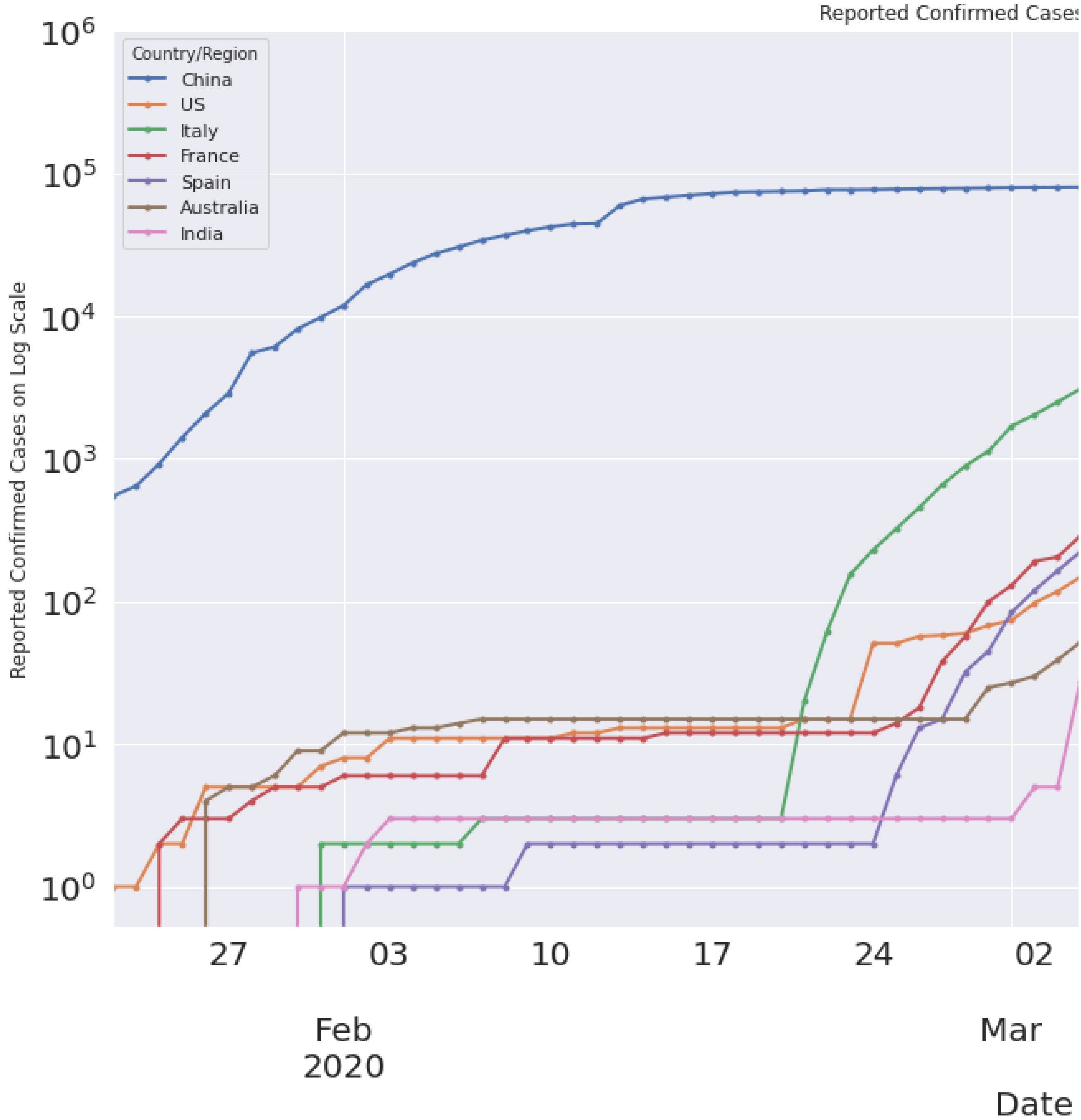
```
# Plot time series of several countries of interest
confirmed_df[countries].plot(figsize=(20,10), linewidth=2, marker='.', colormap='brg', fon
plt.xlabel('Date', fontsize=20);
plt.ylabel('Reported Confirmed cases count', fontsize=20);
plt.title('Reported Confirmed Cases Time Series', fontsize=20);
```



Now, since the US data seems to be going really high. lets take the y-axis on logarithmic scale:

```
# Plot time series of several countries of interest
confirmed_df[countries].plot(figsize=(20,10), linewidth=2, marker='.', fontsize=20, logy =
plt.xlabel('Date', fontsize = 20)
plt.ylabel('Reported Confirmed Cases on Log Scale')
plt.title('Reported Confirmed Cases Time Series Plot')
```

```
Text(0.5, 1.0, 'Reported Confirmed Cases Time Series Plot')
```



Till now, we have explored the confirmed cases data and :

- looked at the dataset containing the number of reported confirmed cases for each region,
- wrangled the data to look at the number of reported confirmed cases by country,
- plotted the number of reported confirmed cases by country (both log and semi-log),
- Used log plots for the data.

Number of reported deaths

As we did above for `raw_data_confirmed`, let's check out the head and the info of the `raw_data_de`

```
raw_deaths_df.head()
```

```
raw_deaths_df.head()
```

	Province/State	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/2
0	NaN	Afghanistan	33.0000	65.0000	0	0	0	
1	NaN	Albania	41.1533	20.1683	0	0	0	
2	NaN	Algeria	28.0339	1.6596	0	0	0	
3	NaN	Andorra	42.5063	1.5218	0	0	0	
4	NaN	Angola	-11.2027	17.8739	0	0	0	

5 rows × 85 columns

```
raw_deaths_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 264 entries, 0 to 263
Data columns (total 85 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Province/State  82 non-null     object 
 1   Country/Region  264 non-null    object 
 2   Lat             264 non-null    float64
 3   Long            264 non-null    float64
 4   1/22/20         264 non-null    int64  
 5   1/23/20         264 non-null    int64  
 6   1/24/20         264 non-null    int64  
 7   1/25/20         264 non-null    int64  
 8   1/26/20         264 non-null    int64  
 9   1/27/20         264 non-null    int64  
 10  1/28/20         264 non-null    int64  
 11  1/29/20         264 non-null    int64  
 12  1/30/20         264 non-null    int64  
 13  1/31/20         264 non-null    int64  
 14  2/1/20          264 non-null    int64  
 15  2/2/20          264 non-null    int64  
 16  2/3/20          264 non-null    int64  
 17  2/4/20          264 non-null    int64  
 18  2/5/20          264 non-null    int64  
 19  2/6/20          264 non-null    int64  
 20  2/7/20          264 non-null    int64  
 21  2/8/20          264 non-null    int64  
 22  2/9/20          264 non-null    int64  
 23  2/10/20         264 non-null    int64  
 24  2/11/20         264 non-null    int64  
 25  2/12/20         264 non-null    int64  
 26  2/13/20         264 non-null    int64  
 27  2/14/20         264 non-null    int64  
 28  2/15/20         264 non-null    int64  
 29  2/16/20         264 non-null    int64  
 30  2/17/20         264 non-null    int64  
 31  2/18/20         264 non-null    int64  
 32  2/19/20         264 non-null    int64  
 33  2/20/20         264 non-null    int64  
 34  2/21/20         264 non-null    int64  
 35  2/22/20         264 non-null    int64  
 36  2/23/20         264 non-null    int64  
 37  2/24/20         264 non-null    int64  
 38  2/25/20         264 non-null    int64  
 39  2/26/20         264 non-null    int64  
 40  2/27/20         264 non-null    int64  
 41  2/28/20         264 non-null    int64  
 42  2/29/20         264 non-null    int64  
 43  3/1/20          264 non-null    int64  
 44  3/2/20          264 non-null    int64  
 45  3/3/20          264 non-null    int64  
 46  3/4/20          264 non-null    int64  
 47  3/5/20          264 non-null    int64  
 48  3/6/20          264 non-null    int64  
 49  3/7/20          264 non-null    int64  
 50  3/8/20          264 non-null    int64  
 51  3/9/20          264 non-null    int64  
 52  3/10/20         264 non-null    int64  
 53  3/11/20         264 non-null    int64  
 54  3/12/20         264 non-null    int64  
 55  3/13/20         264 non-null    int64  
```

```
 56  3/14/20         264 non-null    int64
 57  3/15/20         264 non-null    int64
 58  3/16/20         264 non-null    int64
 59  3/17/20         264 non-null    int64
 60  3/18/20         264 non-null    int64
 61  3/19/20         264 non-null    int64
 62  3/20/20         264 non-null    int64
 63  3/21/20         264 non-null    int64
 64  3/22/20         264 non-null    int64
 65  3/23/20         264 non-null    int64
 66  3/24/20         264 non-null    int64
 67  3/25/20         264 non-null    int64
 68  3/26/20         264 non-null    int64
 69  3/27/20         264 non-null    int64
 70  3/28/20         264 non-null    int64
 71  3/29/20         264 non-null    int64
 72  3/30/20         264 non-null    int64
 73  3/31/20         264 non-null    int64
 74  4/1/20          264 non-null    int64
 75  4/2/20          264 non-null    int64
 76  4/3/20          264 non-null    int64
 77  4/4/20          264 non-null    int64
 78  4/5/20          264 non-null    int64
 79  4/6/20          264 non-null    int64
 80  4/7/20          264 non-null    int64
 81  4/8/20          264 non-null    int64
 82  4/9/20          264 non-null    int64
 83  4/10/20         264 non-null    int64
 84  4/11/20         264 non-null    int64
dtypes: float64(2), int64(81), object(2)
memory usage: 175.4+ KB
```

The structure of this data is similar to the `raw_confirmed_df`, so we can apply the same steps use

Number of reported deaths by country

```
#group-by countries
deaths_df = raw_deaths_df.groupby(['Country/Region']).sum().drop(['Lat','Long'], axis=1)
deaths_df.head()
```

```
# Transpose
deaths_df = deaths_df.transpose()

deaths_df.head()
```

Country/Region	Afghanistan	Albania	Algeria	Andorra	Angola	Antigua and Barbuda	Argentina	A
1/22/20	0	0	0	0	0	0	0	
1/23/20	0	0	0	0	0	0	0	
1/24/20	0	0	0	0	0	0	0	
1/25/20	0	0	0	0	0	0	0	
1/26/20	0	0	0	0	0	0	0	

5 rows × 185 columns

```
# Set index as DateTimeIndex
datetime_index = pd.DatetimeIndex(deaths_df.index)
deaths_df.set_index(datetime_index, inplace=True)

# Check out head
deaths_df.head()
```

```
deaths_df.index
```

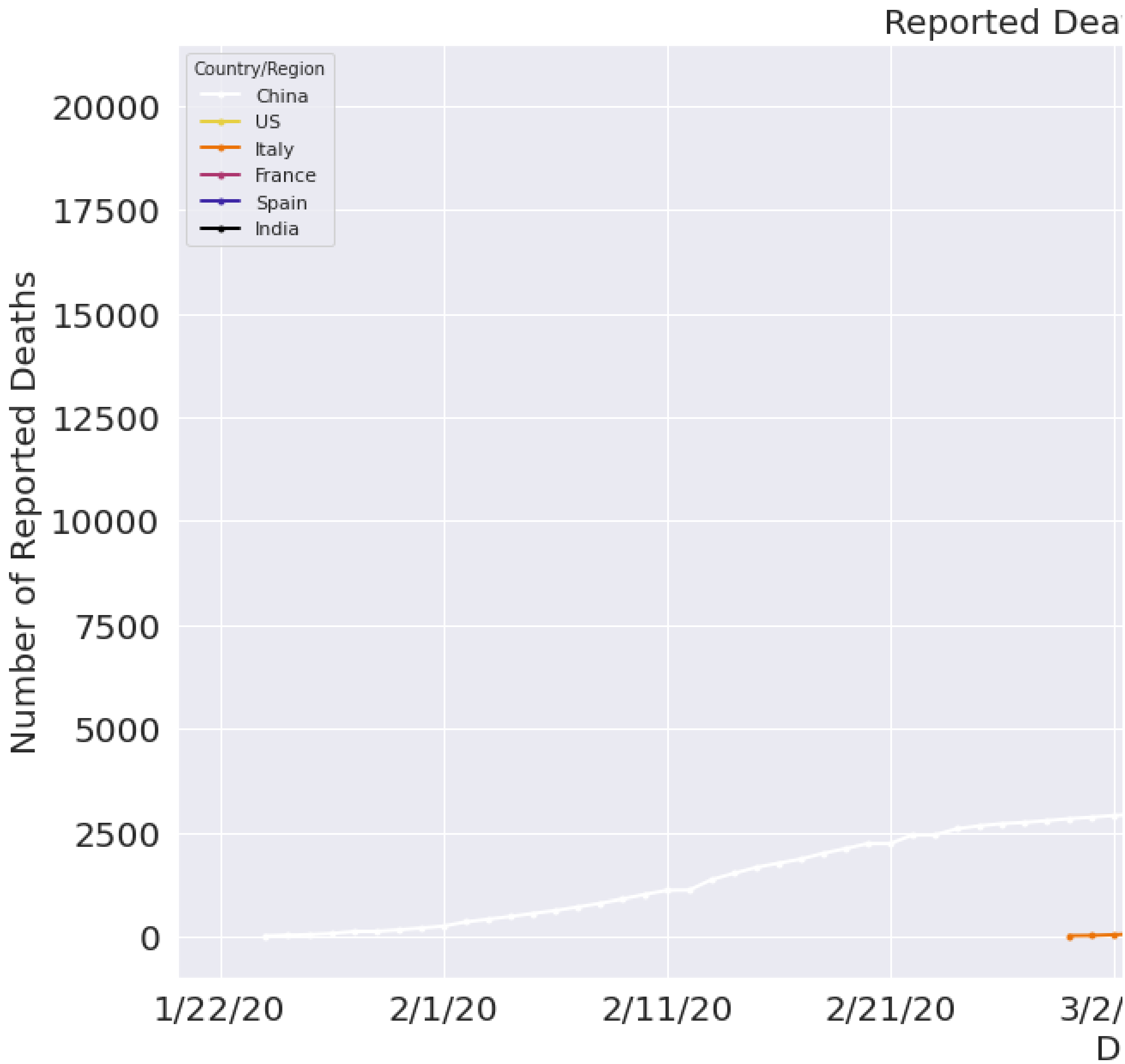
```
Index(['1/22/20', '1/23/20', '1/24/20', '1/25/20', '1/26/20', '1/27/20',
       '1/28/20', '1/29/20', '1/30/20', '1/31/20', '2/1/20', '2/2/20',
       '2/3/20', '2/4/20', '2/5/20', '2/6/20', '2/7/20', '2/8/20', '2/9/20',
       '2/10/20', '2/11/20', '2/12/20', '2/13/20', '2/14/20', '2/15/20',
       '2/16/20', '2/17/20', '2/18/20', '2/19/20', '2/20/20', '2/21/20',
       '2/22/20', '2/23/20', '2/24/20', '2/25/20', '2/26/20', '2/27/20',
       '2/28/20', '2/29/20', '3/1/20', '3/2/20', '3/3/20', '3/4/20', '3/5/20',
       '3/6/20', '3/7/20', '3/8/20', '3/9/20', '3/10/20', '3/11/20', '3/12/20',
       '3/13/20', '3/14/20', '3/15/20', '3/16/20', '3/17/20', '3/18/20',
       '3/19/20', '3/20/20', '3/21/20', '3/22/20', '3/23/20', '3/24/20',
       '3/25/20', '3/26/20', '3/27/20', '3/28/20', '3/29/20', '3/30/20',
       '3/31/20', '4/1/20', '4/2/20', '4/3/20', '4/4/20', '4/5/20', '4/6/20',
       '4/7/20', '4/8/20', '4/9/20', '4/10/20', '4/11/20'],
      dtype='object')
```

Plotting number of reported deaths by country

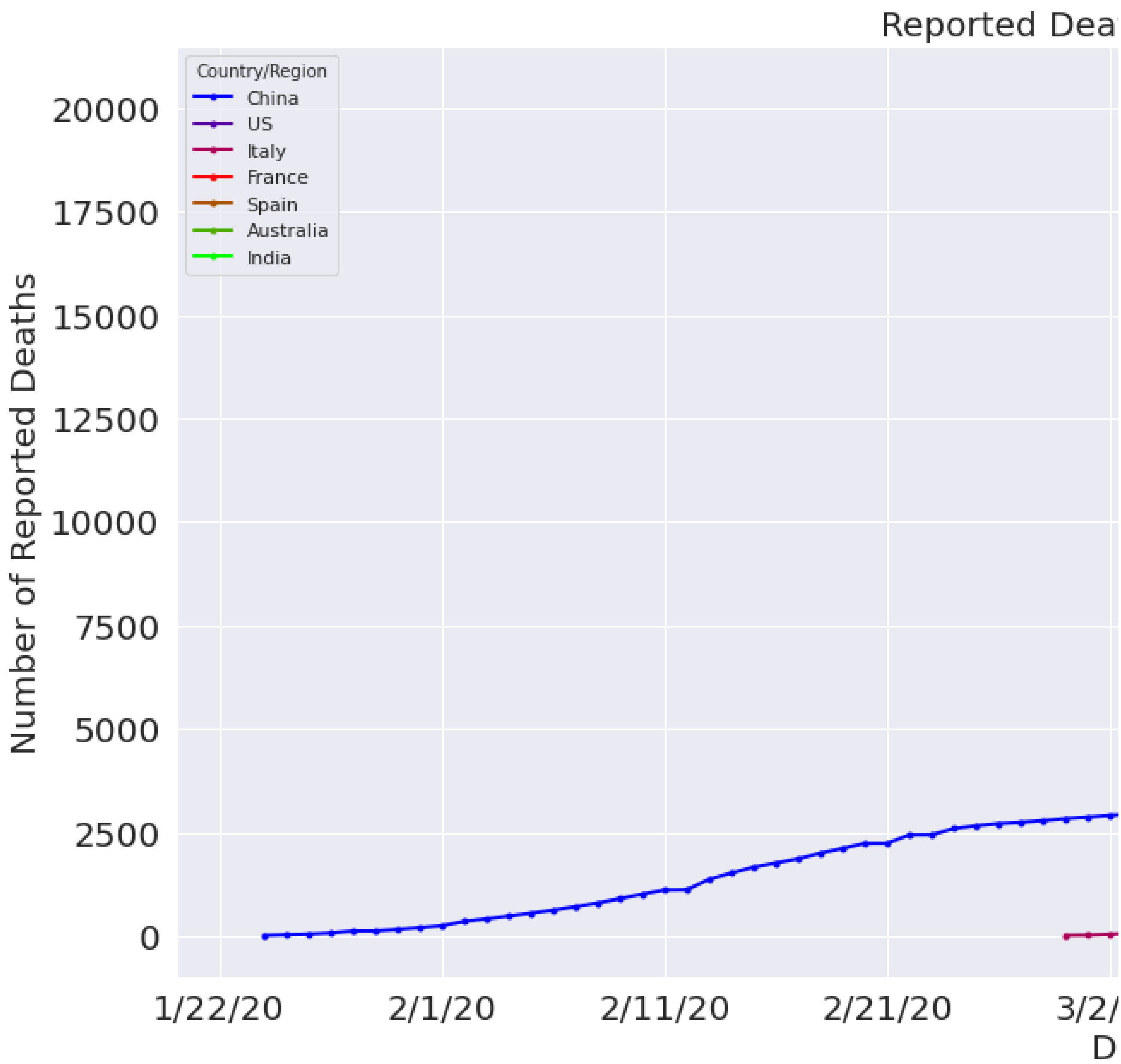
Visualizing the number of reported deaths:

```
# Plot time series of several countries of interest
deaths_df[countries].plot(figsize=(20,10), linewidth=2, marker='.', colormap='CMRmap_r', f
plt.xlabel('Date', fontsize=20);
```

```
plt.xlabel('Date', fontsize=20);
plt.ylabel('Number of Reported Deaths', fontsize=20);
plt.title('Reported Deaths Time Series', fontsize=20);
```

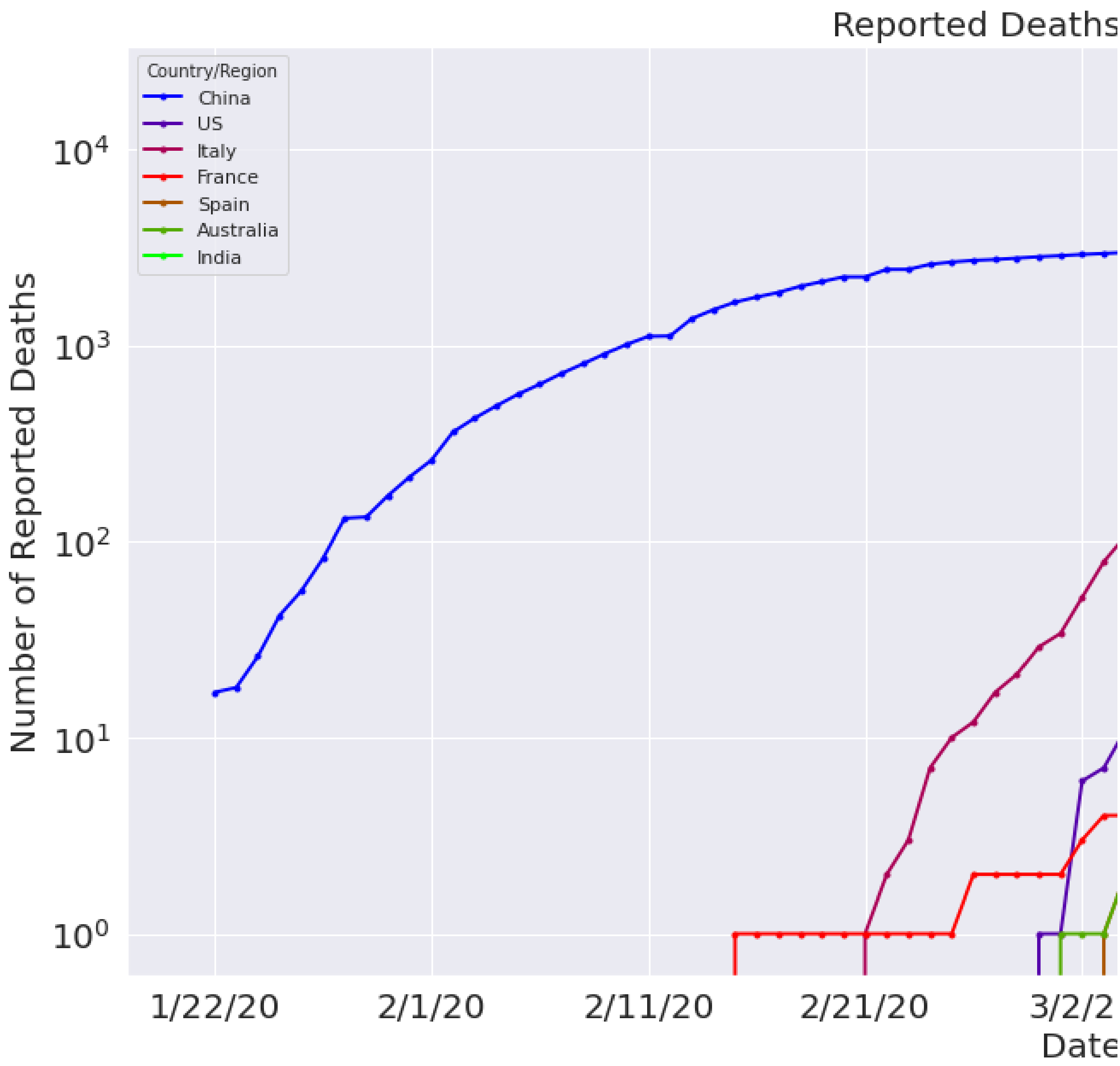



```
# Plot time series of several countries of interest
deaths_df[countries].plot(figsize=(20,10), linewidth=2, marker='.', colormap='brg', fontsi
plt.xlabel('Date', fontsize=20);
plt.ylabel('Number of Reported Deaths', fontsize=20);
plt.title('Reported Deaths Time Series', fontsize=20);
```



Now on a semi-log plot:

```
# Plot time series of countries on log scale
deaths_df[countries].plot(figsize=(20,10), linewidth=2, marker='.', colormap='brg', fontsi
plt.xlabel('Date', fontsize=20);
plt.ylabel('Number of Reported Deaths', fontsize=20);
plt.title('Reported Deaths Time Series', fontsize=20);
```



Aligning growth curves to start with day of number of known deaths ≥ 25

To compare what's happening in different countries, we can align each country's growth curves to known deaths ≥ 25, such as reported in the first figure [here](). To achieve this, first off, let's set set al associated data points don't get plotted at all when we visualize the data:

```
# Loop over columns & set values < 25 to None
for col in deaths_df.columns:
    deaths_df.loc[(deaths_df[col] < 25),col] = None

# Check out tail
deaths_df.tail()
```

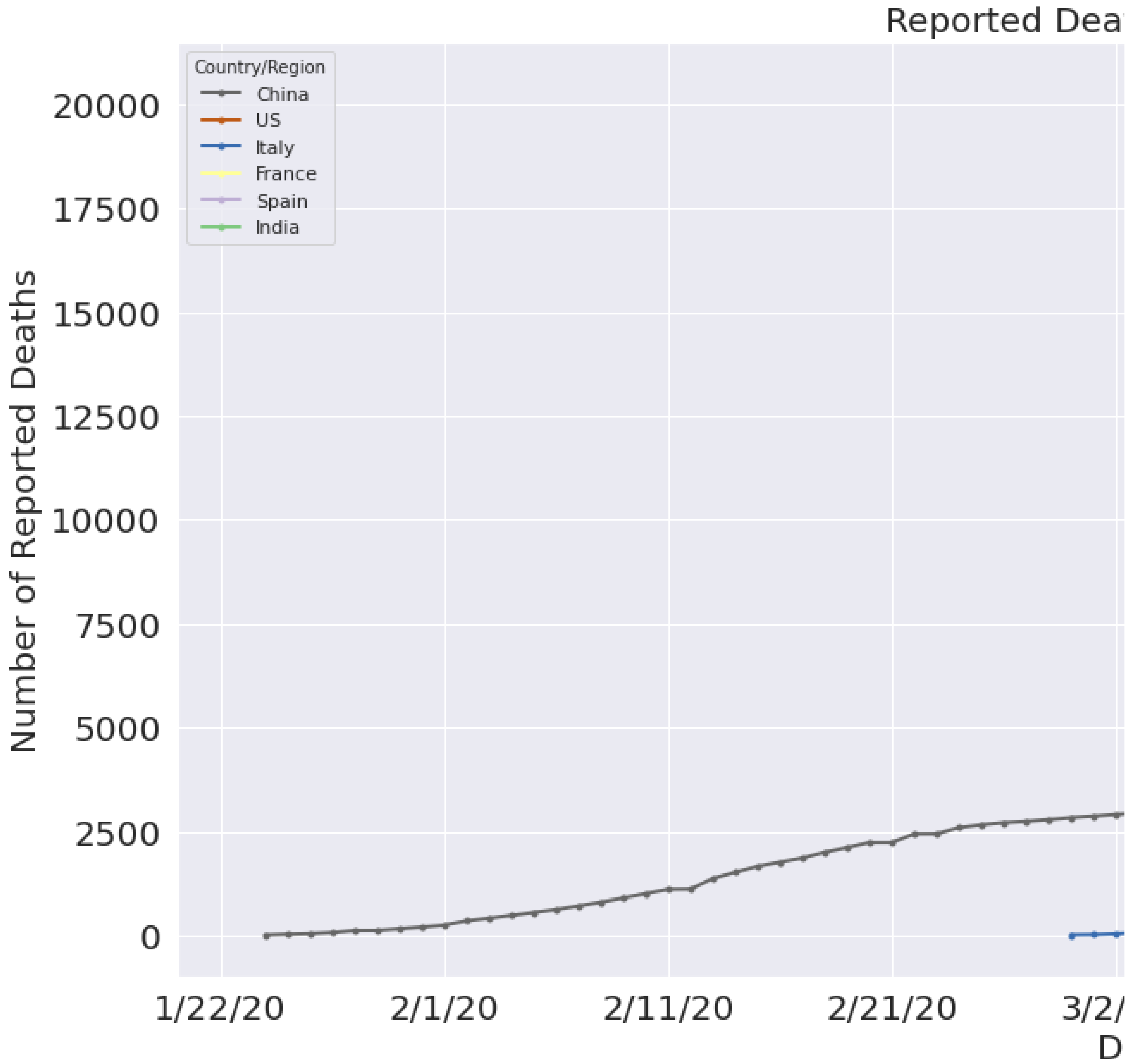
Country/Region	Afghanistan	Albania	Algeria	Andorra	Angola	Antigua and Barbuda	Argentina	A
4/7/20	NaN	NaN	193.0	NaN	NaN	NaN	56.0	
4/8/20	NaN	NaN	205.0	NaN	NaN	NaN	63.0	
4/9/20	NaN	NaN	235.0	25.0	NaN	NaN	72.0	
4/10/20	NaN	NaN	256.0	26.0	NaN	NaN	82.0	
4/11/20	NaN	NaN	275.0	26.0	NaN	NaN	83.0	

5 rows × 185 columns

Now let's plot as above to make sure we see what we think we should see:

```
# Plot time series of several countries of interest
countries = ['China', 'US', 'Italy', 'France', 'Spain', 'India']
deaths_df[countries].plot(figsize=(20,10), linewidth=2, marker='.', colormap='Accent_r', f
plt.xlabel('Date', fontsize=20)
plt.ylabel('Number of Reported Deaths', fontsize=20)
plt.title('Reported Deaths Time Series', fontsize=20)
```

```
Text(0.5, 1.0, 'Reported Deaths Time Series')
```



The countries that have seen less than 25 total deaths will have columns of all NaNs now so let's c
we have left:

```
# Drop columns that are all NaNs (i.e. countries that haven't yet reached 25 deaths)
deaths_df.dropna(axis=1, how='all', inplace=True)
deaths_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 81 entries, 1/22/20 to 4/11/20
Data columns (total 64 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Algeria                 17 non-null     float64
 1   Andorra                 3 non-null      float64
 2   Argentina               12 non-null     float64
 3   Australia               9 non-null      float64
 4   Austria                 19 non-null     float64
 5   Bangladesh              2 non-null      float64
 6   Belgium                 23 non-null     float64
 7   Bosnia and Herzegovina  6 non-null      float64
 8   Brazil                  21 non-null     float64
 9   Bulgaria                2 non-null      float64
 10  Burkina Faso            1 non-null      float64
 11  Canada                  20 non-null     float64
 12  Chile                   8 non-null      float64
 13  China                   79 non-null     float64
 14  Colombia                9 non-null      float64
 15  Czechia                 12 non-null     float64
 16  Denmark                 19 non-null     float64
 17  Dominican Republic      15 non-null     float64
 18  Ecuador                 19 non-null     float64
 19  Egypt                   16 non-null     float64
 20  Finland                 8 non-null      float64
 21  France                  33 non-null     float64
 22  Germany                 25 non-null     float64
 23  Greece                  17 non-null     float64
 24  Hungary                 9 non-null      float64
 25  India                   14 non-null     float64
 26  Indonesia               24 non-null     float64
 27  Iran                    45 non-null     float64
 28  Iraq                    19 non-null     float64
 29  Ireland                 15 non-null     float64
 30  Israel                  11 non-null     float64
 31  Italy                   43 non-null     float64
 32  Japan                   27 non-null     float64
 33  Korea, South            41 non-null     float64
 34  Luxembourg              11 non-null     float64
 35  Malaysia                16 non-null     float64
 36  Mexico                  12 non-null     float64
 37  Moldova                 4 non-null      float64
 38  Morocco                 15 non-null     float64
 39  Netherlands             26 non-null     float64
 40  North Macedonia         5 non-null      float64
 41  Norway                  14 non-null     float64
 42  Pakistan                12 non-null     float64
 43  Panama                  12 non-null     float64
 44  Peru                    12 non-null     float64
 45  Philippines             21 non-null     float64
 46  Poland                  13 non-null     float64
 47  Portugal                19 non-null     float64
 48  Romania                 16 non-null     float64
 49  Russia                  10 non-null     float64
 50  San Marino              13 non-null     float64
 51  Saudi Arabia            9 non-null      float64
 52  Serbia                  11 non-null     float64
 53  Slovenia                7 non-null      float64
 54  South Africa            1 non-null      float64
 55  Spain                   34 non-null     float64
```

```
 56  Sweden                 20 non-null     float64
 57  Switzerland            26 non-null     float64
 58  Thailand               6 non-null      float64
 59  Tunisia                3 non-null      float64
 60  Turkey                 21 non-null     float64
 61  US                     33 non-null     float64
 62  Ukraine                9 non-null      float64
 63  United Kingdom         27 non-null     float64
dtypes: float64(64)
memory usage: 43.6+ KB
```

As we're going to align the countries from the day they first had at least 25 deaths, we won't need t
date at all. So we can

- Reset the Index, which will give us an ordinal index (which turns the date into a regular colum
- Drop the date column (which will be called 'index) after the reset.

```
# sort index, drop date column
deaths_df_drop = deaths_df.reset_index().drop(['index'], axis=1)
deaths_df_drop.head()
```

Country/Region	Algeria	Andorra	Argentina	Australia	Austria	Bangladesh	Belgium
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN

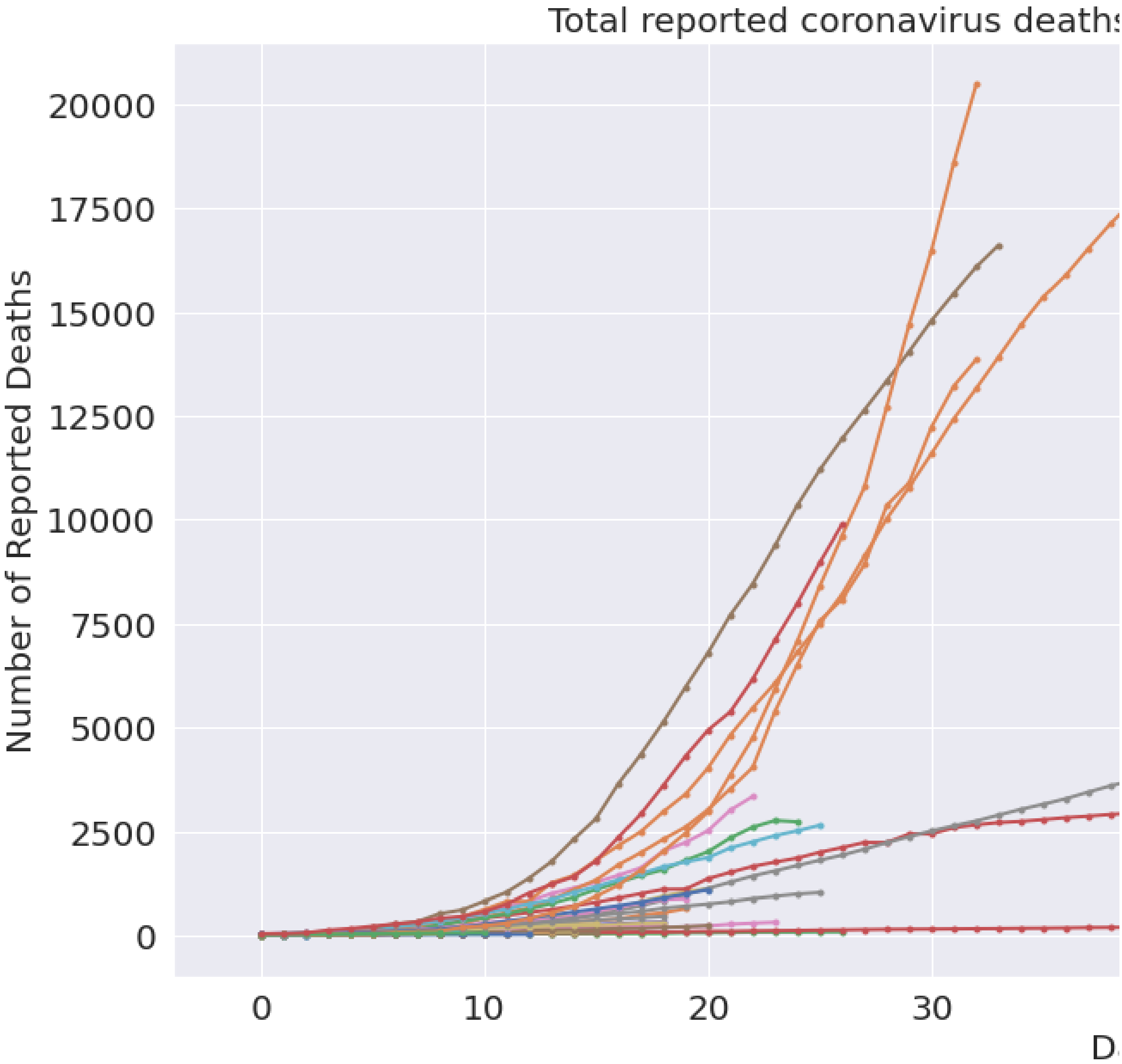
Now it's time to shift each column so that the first entry is the first NaN value that it contains! To d
each column. How much do we shift each column, though? The magnitude of the shift is given by
column, which we can retrieve using the `first_valid_index()` method on the column **but** we wan
convention and perhaps intuition). SO let's do it.

```
# shift
for col in deaths_df_drop.columns:
    deaths_df_drop[col] = deaths_df_drop[col].shift(-deaths_df_drop[col].first_valid_index
# check out head
deaths_df_drop.head()
```

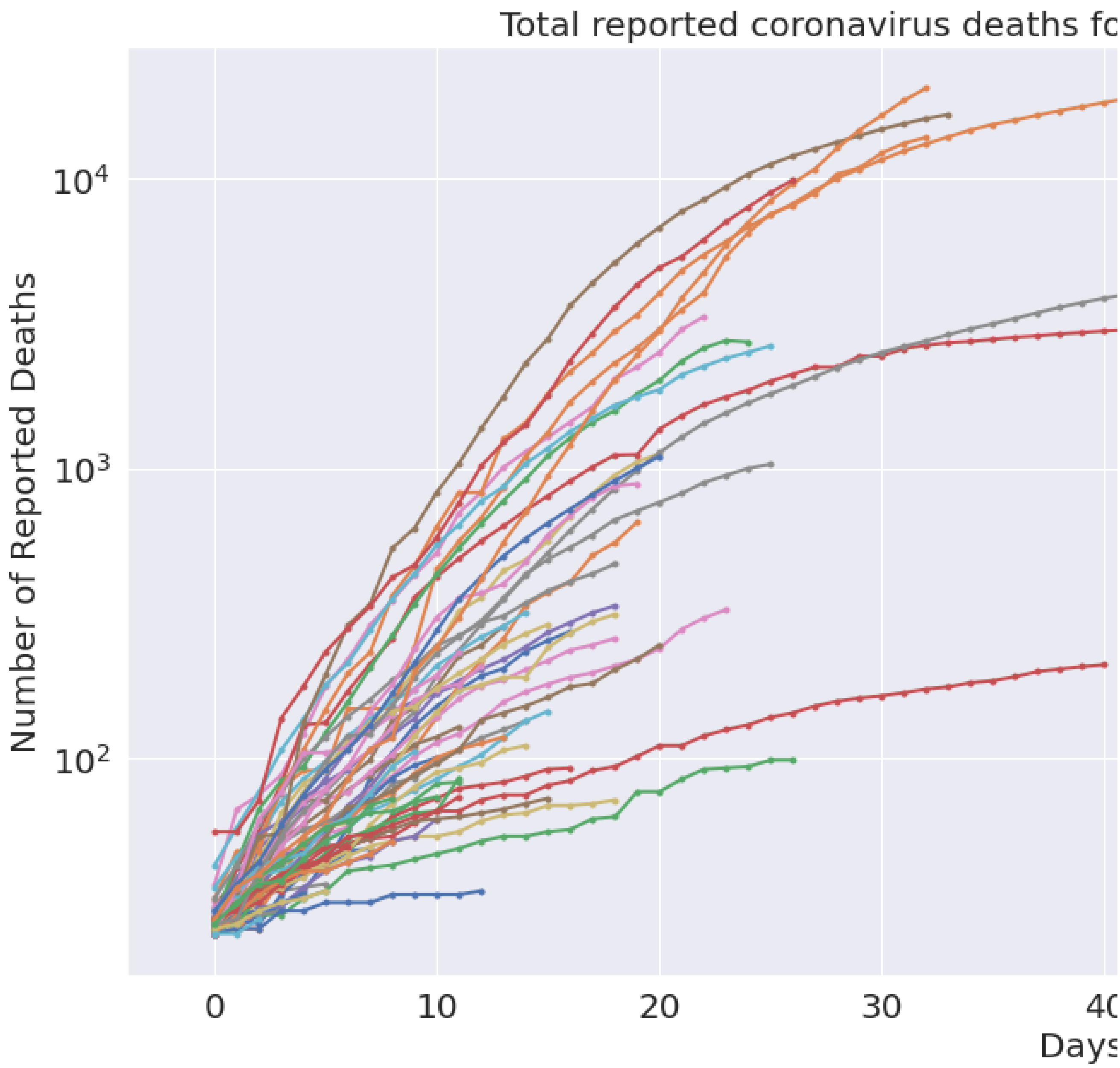
Country/Region	Algeria	Andorra	Argentina	Australia	Austria	Bangladesh	Belgium
0	25.0	25.0	27.0	28.0	28.0	27.0	37.0
1	26.0	26.0	28.0	30.0	30.0	30.0	67.0
2	29.0	26.0	36.0	35.0	49.0	NaN	75.0
3	31.0	NaN	39.0	40.0	58.0	NaN	88.0
4	35.0	NaN	43.0	45.0	68.0	NaN	122.0

Now we get to plot our time series, first with linear axes, then semi-log:

```
# Plot time series
ax = deaths_df_drop.plot(figsize=(20,10), linewidth=2, marker=".", fontsize=20)
ax.legend(ncol=3, loc='upper right')
plt.xlabel('Days', fontsize=20);
plt.ylabel('Number of Reported Deaths', fontsize=20);
plt.title('Total reported coronavirus deaths for places with at least 25 deaths', fontsize
```

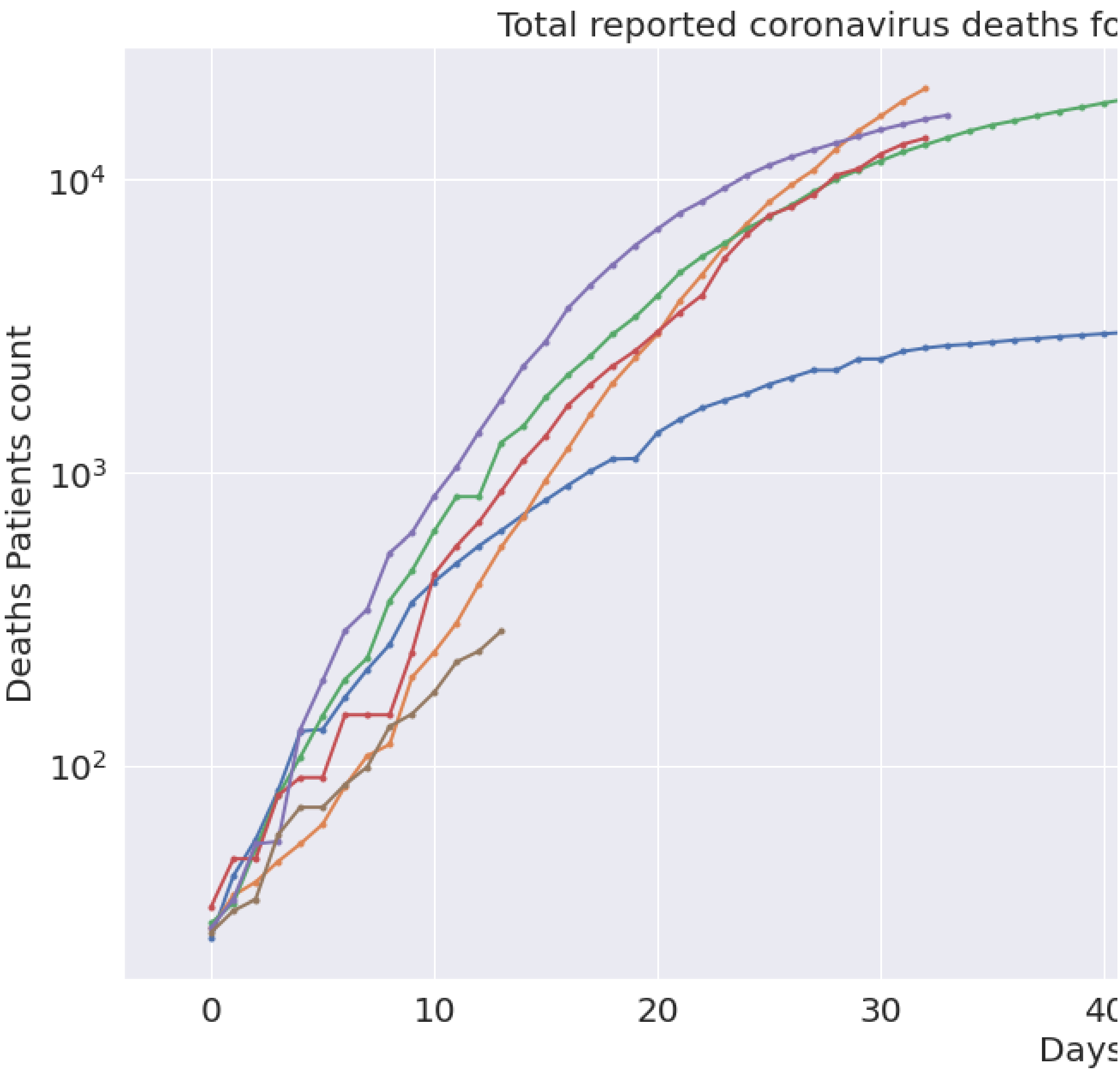



```
# Plot time series
ax = deaths_df_drop.plot(figsize=(20,10), linewidth=2, marker=".", fontsize=20, logy=True)
ax.legend(ncol=3, loc='upper right')
plt.xlabel('Days', fontsize=20);
plt.ylabel('Number of Reported Deaths', fontsize=20);
plt.title('Total reported coronavirus deaths for places with at least 25 deaths', fontsize
```



Note: although the plot is what we wanted, the above plots are challenging to retrieve any meaninç growth curves so that it's very crowded **and** too many colours look the same so it's difficult to tell v plot less curves now and further down in the notebook I'll use the python package Altair to introdu

```
# Plot semi log time series
ax = deaths_df_drop[countries].plot(figsize=(20,10), linewidth=2, marker='.', fontsize=20,
ax.legend(ncol=3, loc='upper right')
plt.xlabel('Days', fontsize=20);
plt.ylabel('Deaths Patients count', fontsize=20);
plt.title('Total reported coronavirus deaths for places with at least 25 deaths', fontsize
```



Till Now, We

- looked at the dataset containing the number of reported deaths for each region,
- wrangled the data to look at the number of reported deaths by country,
- plotted the number of reported deaths by country on linear and log scale.
- aligned growth curves to start with day of number of known deaths ≥ 25.

Plotting number of recovered people

The third dataset in the Hopkins repository is the number of recovered. We want to do similar data *could* copy and paste our code again *but*, if you're writing the same code three times, it's likely time

```
# Function for grouping countries by region
def group_by_country(raw_data):

    # Group by
```

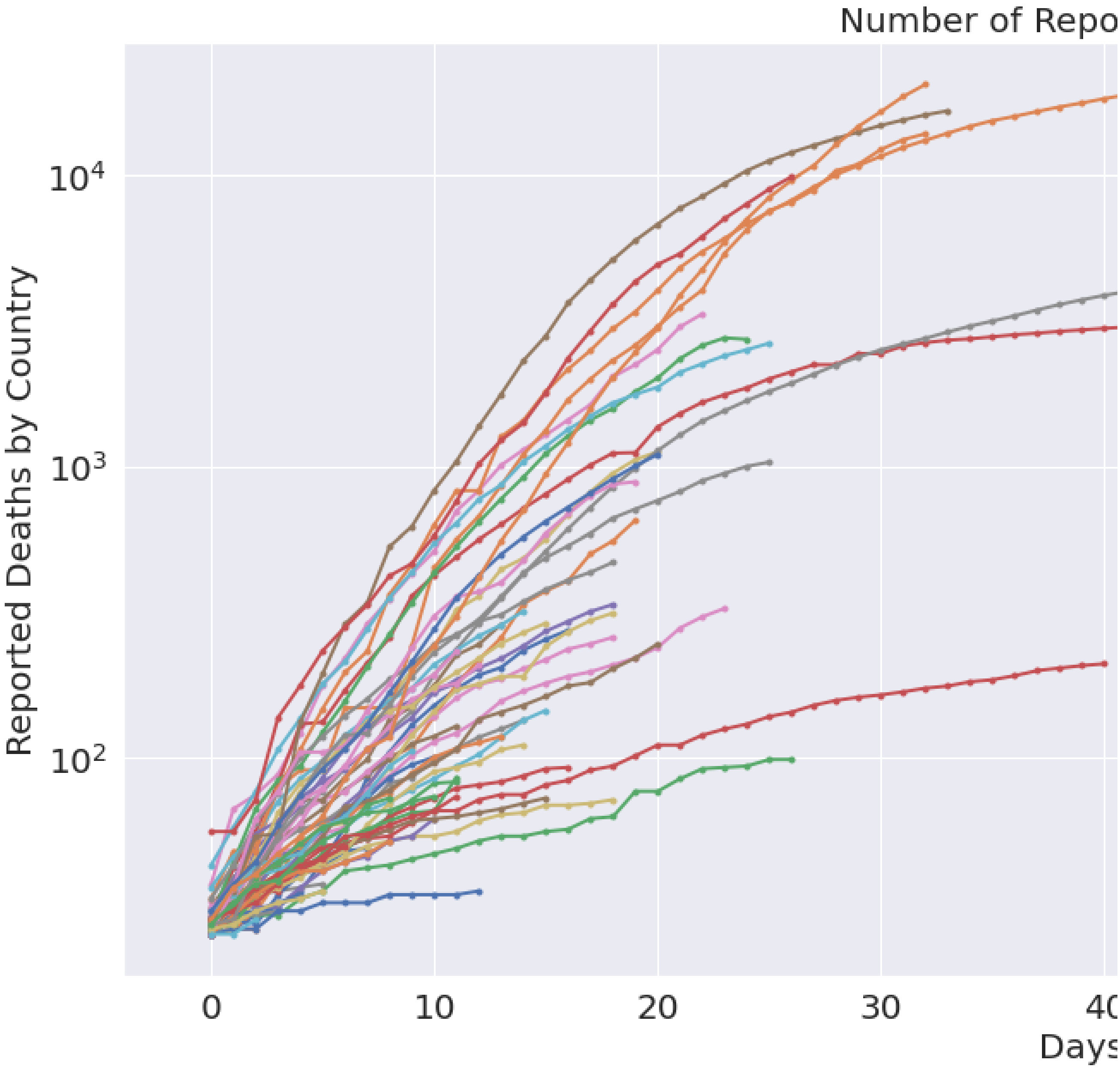
```
    # Group by
    data = raw_data.groupby(['Country/Region']).sum().drop(['Lat', 'Long'], axis=1)
    # Transpose
    data = data.transpose()
    # Set index as DateTimeIndex
    datetime_index = pd.DatetimeIndex(data.index)
    data.set_index(datetime_index, inplace=True)
    return data
```

```
# Function to align growth curves
def align_curves(data, min_val):

    # Loop over columns & set values < min_val to None
    for col in data.columns:
        data.loc[(data[col] < min_val), col] = None
    # Drop columns with all NaNs
    data.dropna(axis=1, how="all", inplace=True)
    # Reset index, drop date
    data = data.reset_index().drop(['index'], axis=1)
    # Shift each column to begin with first valid index
    for col in data.columns:
        data[col] = data[col].shift(-data[col].first_valid_index())
    return data
```

```
# Function to plot time series
def plot_time_series(df, plot_title, x_label, y_label, logy=False):

    ax = df.plot(figsize=(20,10), linewidth=2, marker='.', fontsize=20, logy=logy)
    ax.legend(ncol=3, loc='lower right')
    plt.xlabel(x_label, fontsize=20);
    plt.ylabel(y_label, fontsize=20);
    plt.title(plot_title, fontsize=20);
```

Trying these functions at work on the 'number of deaths' data:

```
deaths_country_drop = (group_by_country(raw_deaths_df))
deaths_country_drop = align_curves(deaths_country_drop, min_val=25)
plot_time_series(deaths_country_drop, 'Number of Reported Deaths', 'Days', 'Reported Death
```



Now let's check use our functions to group, wrangle, and plot the recovered patients data:

```
# group by country and check out tail
recovered_df = group_by_country(raw_recovered_df)
recovered_df.tail()
```

Country/Region	Afghanistan	Albania	Algeria	Andorra	Angola	Antigua and Barbuda	Argentina	A
2020-04-07	18	131	113	39	2	0	338	
2020-04-08	29	154	237	52	2	0	358	
2020-04-09	32	165	347	58	2	0	365	
2020-04-10	32	182	405	71	2	0	375	
2020-04-11	32	197	460	71	4	0	440	

5 rows × 185 columns

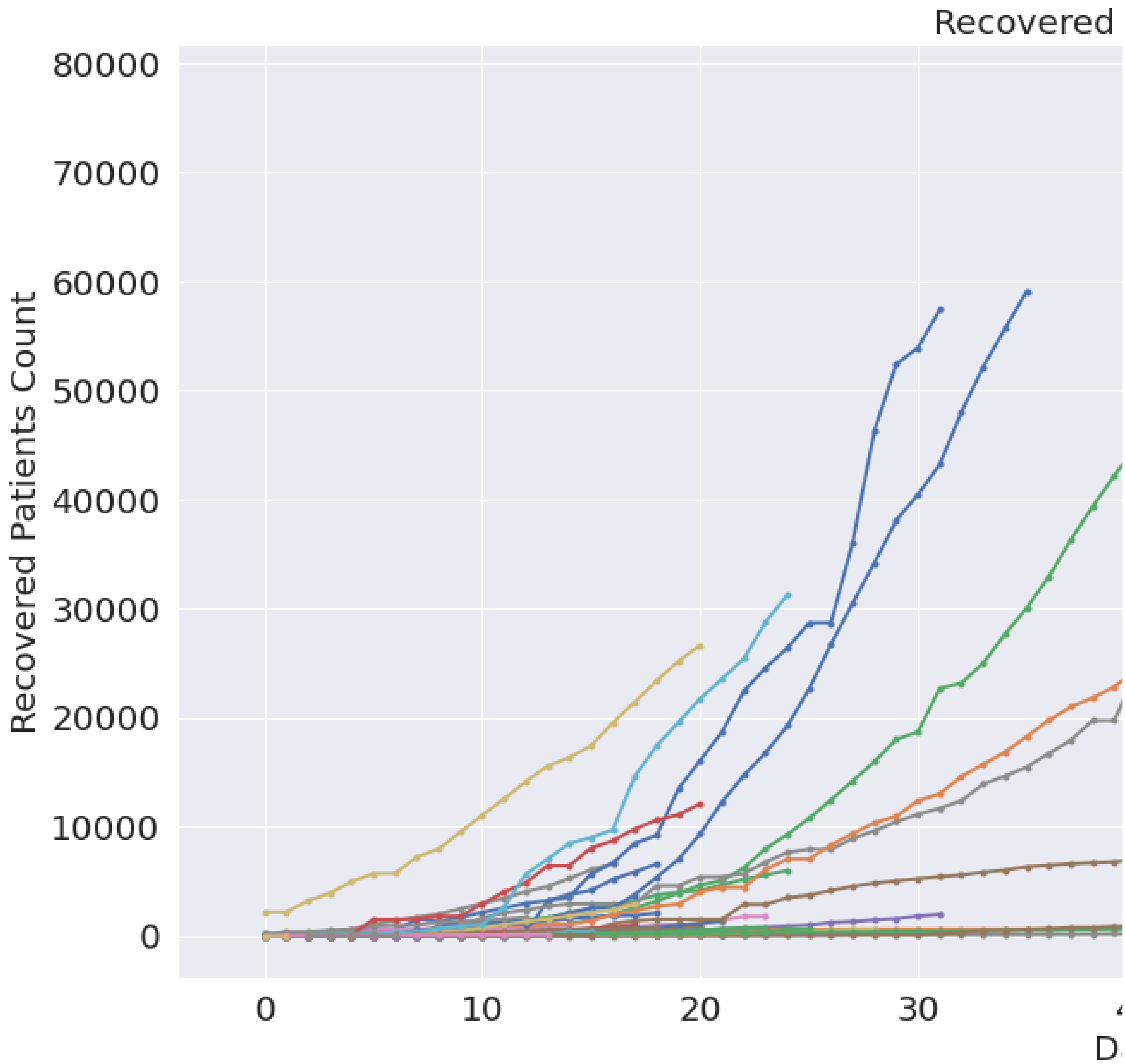
```
# align curves and check out head
recovered_df_drop = align_curves(recovered_df, min_val=25)
recovered_df_drop.head()
```

Country/Region	Afghanistan	Albania	Algeria	Andorra	Argentina	Armenia	Australia
0	29.0	31.0	32.0	26.0	52.0	28.0	26.0
1	32.0	31.0	32.0	31.0	52.0	30.0	26.0
2	32.0	33.0	32.0	39.0	63.0	30.0	26.0
3	32.0	44.0	65.0	52.0	72.0	30.0	88.0
4	NaN	52.0	65.0	58.0	72.0	30.0	88.0

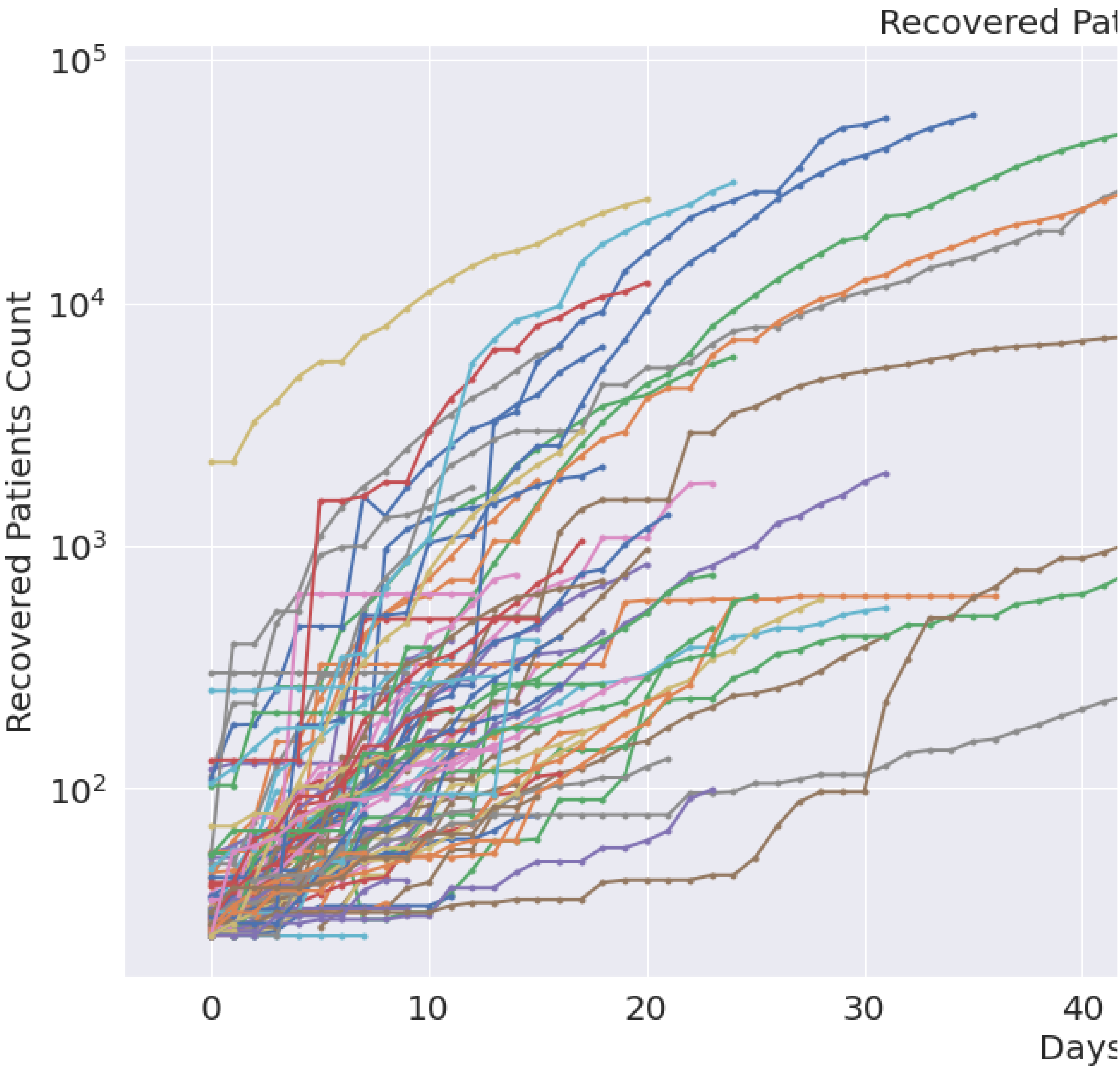
5 rows × 109 columns

Plot time series:

```
plot_time_series(recovered_df_drop, 'Recovered Patients Plot', 'Days', 'Recovered Patients
```

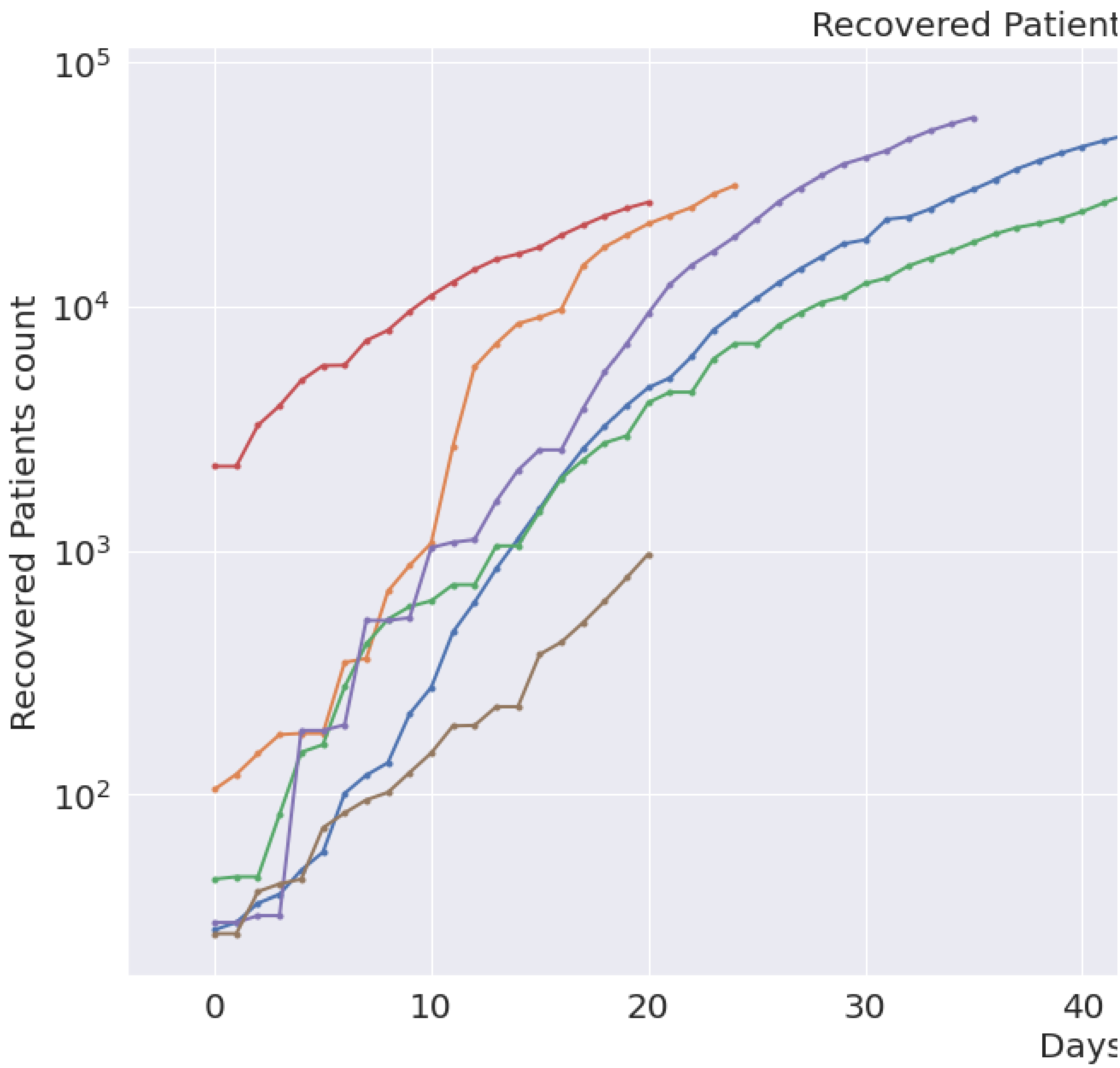



```
plot_time_series(recovered_df_drop, 'Recovered Patients Plot', 'Days', 'Recovered Patients
```



Since this plot gets messed up because of the number of countries, I will agair

```
plot_time_series(recovered_df_drop[countries], 'Recovered Patients Time Series', 'Days', '
```



Now, I looked at the dataset containing the number of reported recoveries for each region, written the data along with using these functions.

Interactive plots with altair

Now for some interactive data visualizations, I will be using Altair which can produce visualization of confirmed number of deaths by country for places with at least 25 deaths, similar to the one ab is also really good.

Before going to Altair, I will reshape our `deaths_df` dataset. Notice that it's currently in **wide data f** row for each "day" (where day 1 is the first day with over 25 confirmed deaths).

```
# Look at head
deaths_df_drop.head()
```

Country/Region	Algeria	Andorra	Argentina	Australia	Austria	Bangladesh	Belgium
0	25.0	25.0	27.0	28.0	28.0	27.0	37.0
1	26.0	26.0	28.0	30.0	30.0	30.0	67.0
2	29.0	26.0	36.0	35.0	49.0	NaN	75.0
3	31.0	NaN	39.0	40.0	58.0	NaN	88.0
4	35.0	NaN	43.0	45.0	68.0	NaN	122.0

For Altair, we'll want to convert the data into **long data format**. What this will do essentially have a will be 'Day', 'Country', and number of 'Deaths'. We do this using the dataframe method `.melt()` as

```
# create long data for deaths
deaths_long = deaths_df_drop.reset_index().melt(id_vars='index', value_name='Deaths').rena
deaths_long.head()
```

	Day	Country/Region	Deaths
0	0	Algeria	25.0
1	1	Algeria	26.0
2	2	Algeria	29.0
3	3	Algeria	31.0
4	4	Algeria	35.0

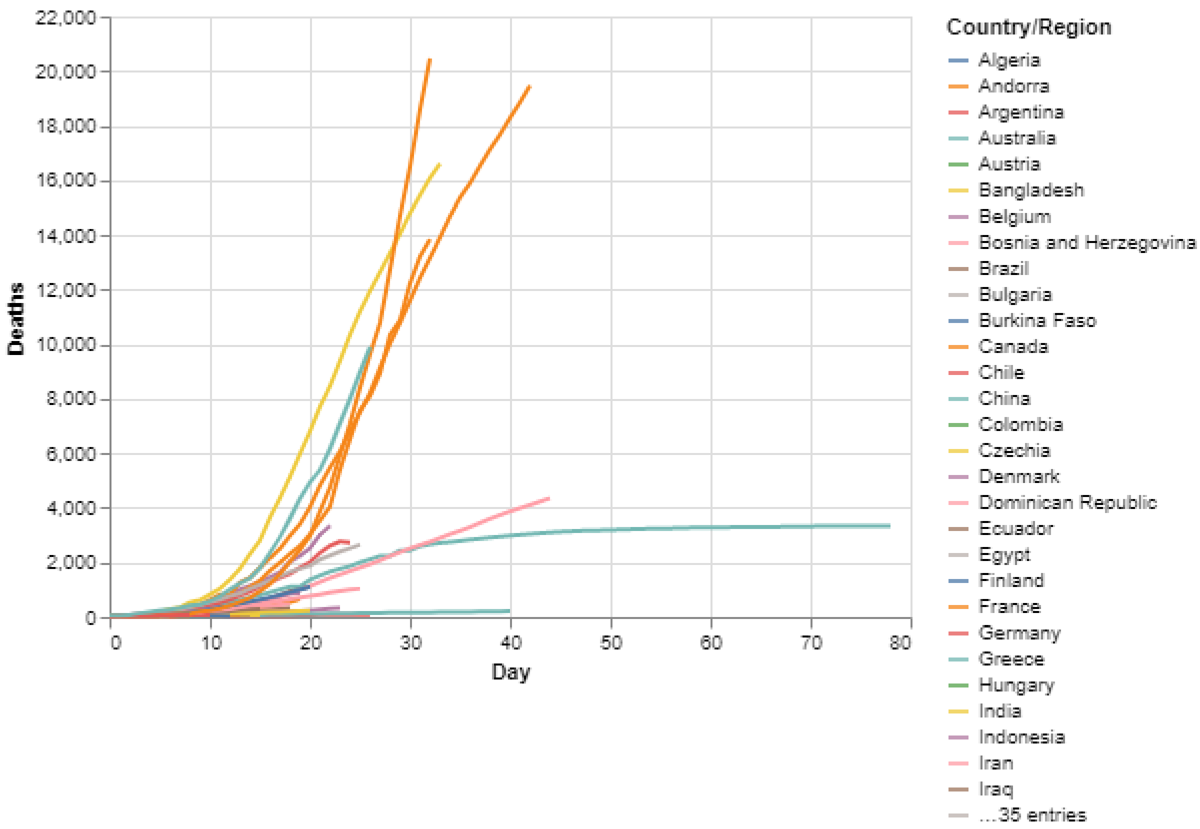
```
deaths_long.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5184 entries, 0 to 5183
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Day             5184 non-null   int64
 1   Country/Region  5184 non-null   object
 2   Deaths          1081 non-null   float64
dtypes: float64(1), int64(1), object(1)
memory usage: 121.6+ KB
```

We'll see the power of having long data when using Altair. Now having transformed our data, let's i

```
import altair as alt

alt.data_transformers.disable_max_rows()
#This particular line of code is to be used when we have more than 5000 rows in our datase
#This limit has just been set to prevent our notebook from growing excessive in size.
```

```
# altair plotting
alt.Chart(deaths_long).mark_line().encode(
    x='Day',
    y='Deaths',
    color='Country/Region')
```



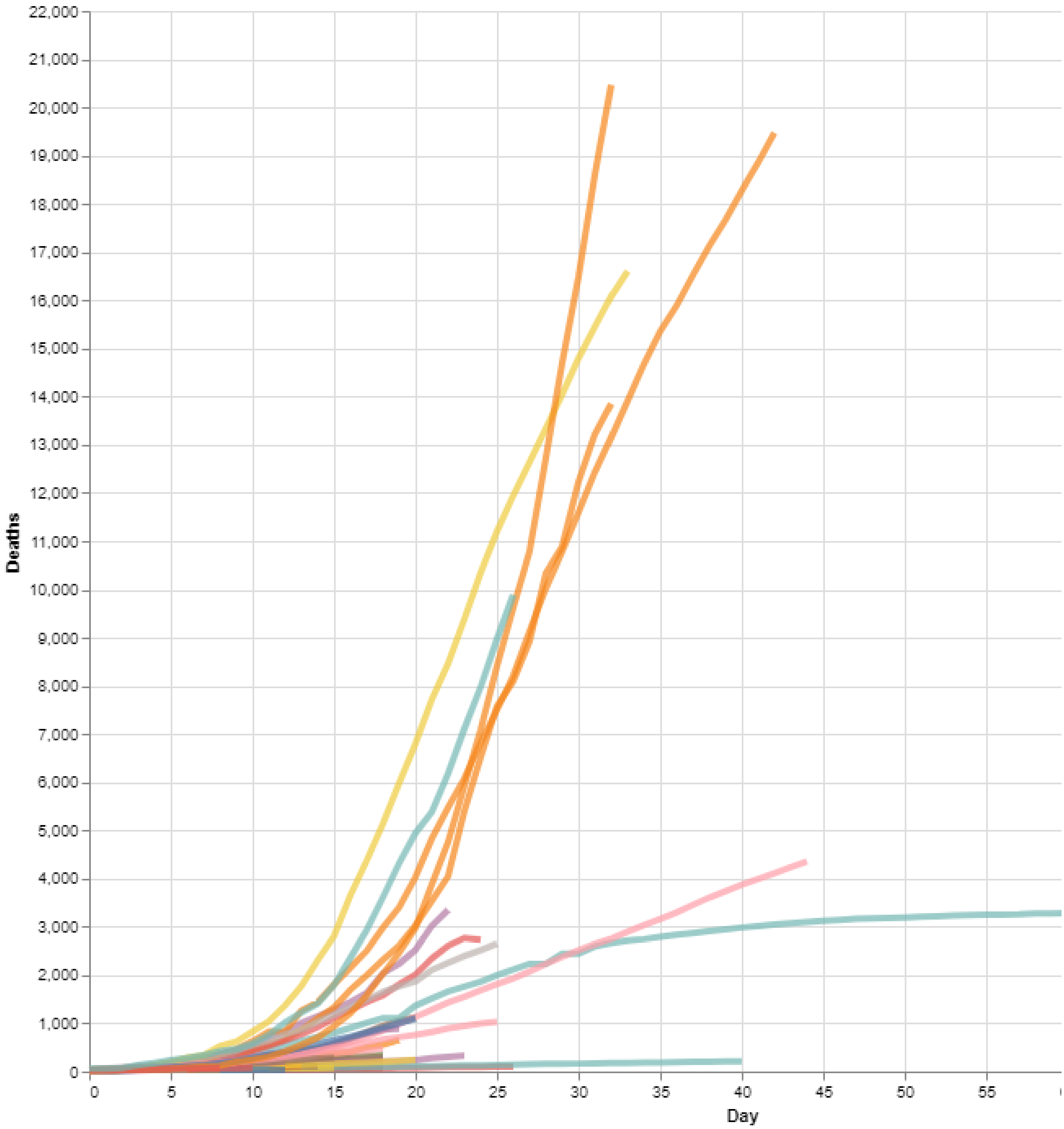
So, we have successfully made the plot.

The [Altair documentation states](),

> The key idea is that you are declaring links between *data columns* and *visual encoding channe*
> etc. The rest of the plot details are handled automatically. Building on this declarative plotting
> to sophisticated plots and visualizations can be created using a relatively concise grammar.

I can now customize the code to thicken the line width, to alter the opacity, and to make the chart l

```
# altair plot
alt.Chart(deaths_long).mark_line(strokeWidth=4, opacity=0.7).encode(
    x='Day',
    y='Deaths',
    color='Country/Region'
).properties(
    width=800, height=650
)
```



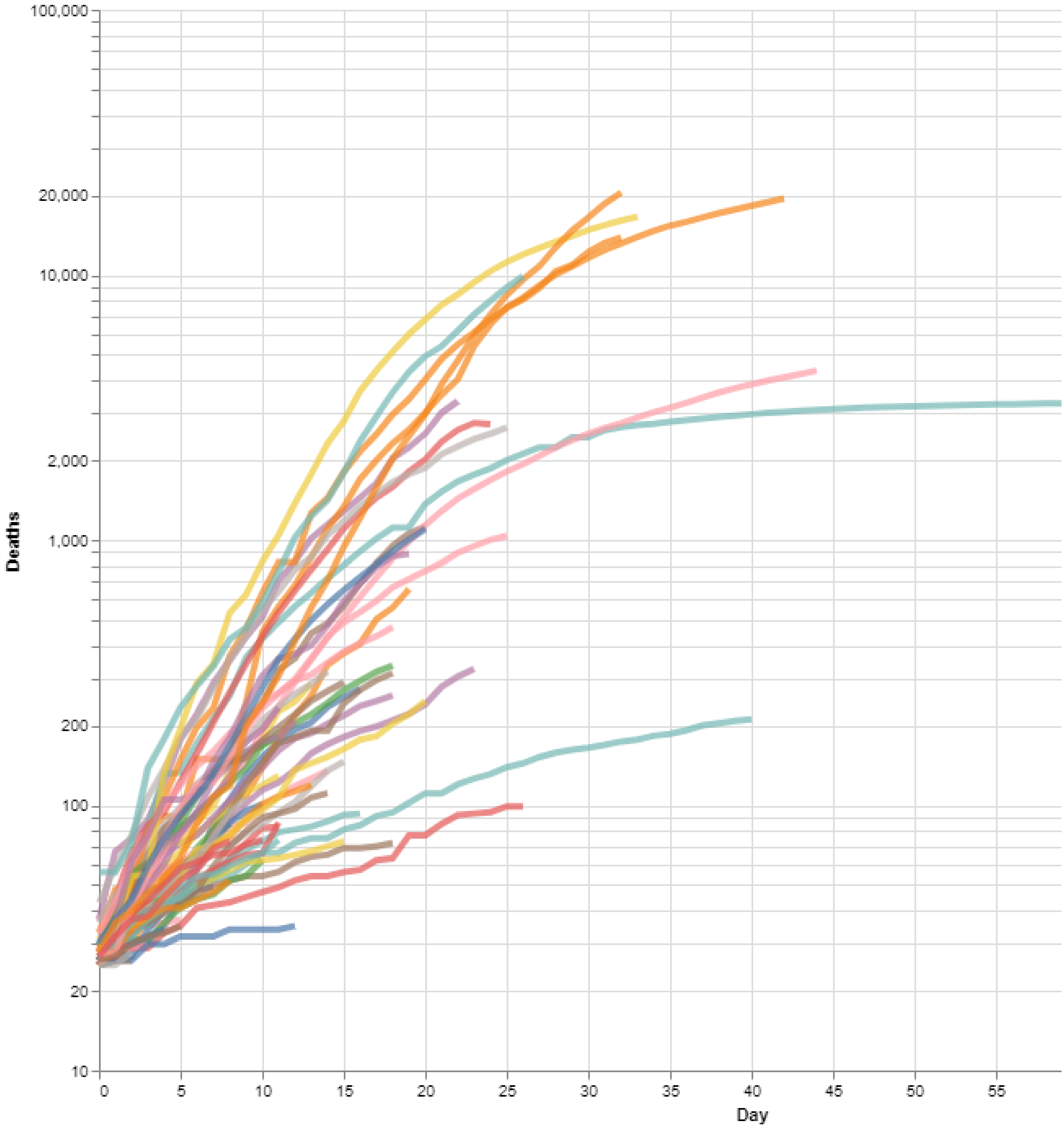
We can also add a log y-axis. To do this, The long-form, we express the types using the long-form
the Altair documentation

> useful when doing more fine-tuned adjustments to the encoding, such as binning, axis and sc

We'll also now add a hover tooltip so that, when we hover our cursor over any point on any of the li
the number of 'Deaths'.

```
# altair plot
alt.Chart(deaths_long).mark_line(strokeWidth=4, opacity=0.7).encode(
    x=alt.X('Day'),
    y=alt.Y('Deaths', scale=alt.Scale(type='log')),
    color='Country/Region',
    tooltip=['Country/Region', 'Day','Deaths']
```

```
).properties(
    width=800,
    height=650
)
```

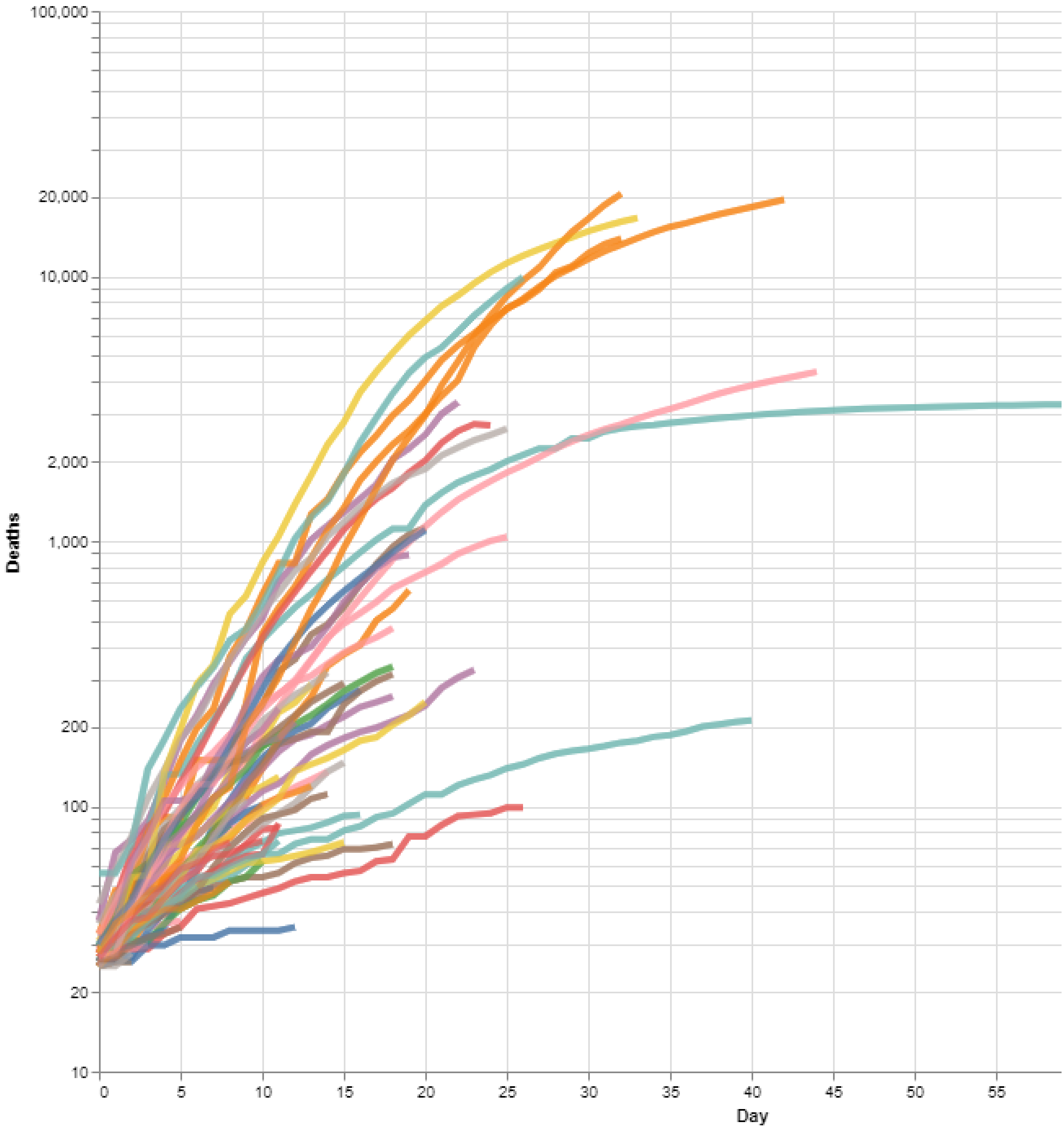


It's great that we could add that useful hover tooltip with one line of code `tooltip=['Country/Regi` such information rich interaction to the chart. One useful aspect of the NYTimes chart was that, w made it stand out against the other. We're going to do something similar here: in the resulting char grey.

Note: When first attempting to build this chart, I discovered [here]() that "multiple conditional values i Lite spec," which is what Altair uses. For this reason, we build the chart, then an overlay, and then c

```
# Selection tool
selection = alt.selection_single(fields=['Country/Region'])
```

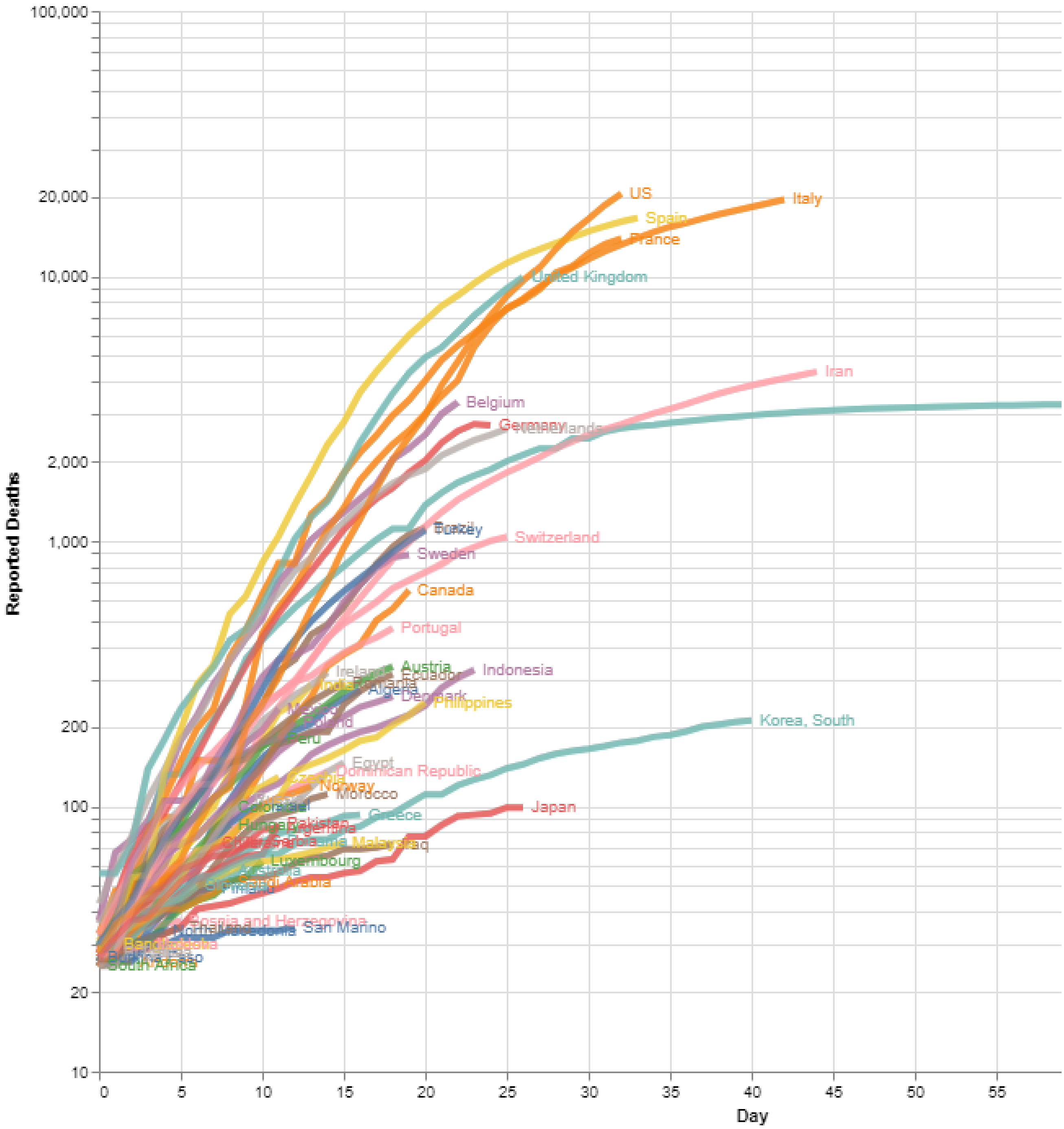
```
# Color change when clicked
color = alt.condition(selection,
                    alt.Color('Country/Region:N'),
                    alt.value('lightgray'))


# Base altair plot
base = alt.Chart(deaths_long).mark_line(strokeWidth=4, opacity=0.7).encode(
    x=alt.X('Day'),
    y=alt.Y('Deaths', scale=alt.Scale(type='log')),
    color='Country/Region',
    tooltip=['Country/Region', 'Day','Deaths']
).properties(
    width=800,
    height=650
)

# Chart
chart = base.encode(
  color=alt.condition(selection, 'Country/Region:N', alt.value('lightgray'))
).add_selection(
  selection
)

# Overlay
overlay = base.encode(
    color='Country/Region',
  opacity=alt.value(0.5),
  tooltip=['Country/Region:N', 'Name:N']
).transform_filter(
  selection
)

# Sum em up!
chart + overlay
```



It's not super easy to line up the legend with the curves on the chart so let's put the labels on the cl

```
# drop NaNs
deaths_long = deaths_long.dropna()

# Selection tool
selection = alt.selection_single(fields=['Country/Region'])
# Color change when clicked
color = alt.condition(selection,
                    alt.Color('Country/Region:N'),
                    alt.value('lightgray'))


# Base altair plot
base = alt.Chart(deaths_long).mark_line(strokeWidth=4, opacity=0.7).encode(
```

```
    x=alt.X('Day'),
    y=alt.Y('Deaths', scale=alt.Scale(type='log')),
    color=alt.Color('Country/Region', legend=None),
).properties(
    width=800,
    height=650
)

# Chart
chart = base.encode(
  color=alt.condition(selection, 'Country/Region:N', alt.value('lightgray'))
).add_selection(
  selection
)

# Overlay
overlay = base.encode(
  color='Country/Region',
  opacity=alt.value(0.5),
  tooltip=['Country/Region:N', 'Name:N']
).transform_filter(
  selection
)

# Text labels
text = base.mark_text(
    align='left',
    dx=5,
    size=10
).encode(
    x=alt.X('Day', aggregate='max',  axis=alt.Axis(title='Day')),
    y=alt.Y('Deaths', aggregate={'argmax': 'Day'}, axis=alt.Axis(title='Reported Deaths'))
    text='Country/Region',
).transform_filter(
    selection
)

# Sum em up!
chart + overlay + text
```



Summary: So, now we have

- melted the data into long format,
- used Altair to make interactive plots of increasing richness,

Thank You! You can check out [my github]() for more interesting EDAs and projects.