

# Covid-19 Spread in the US and Europe World

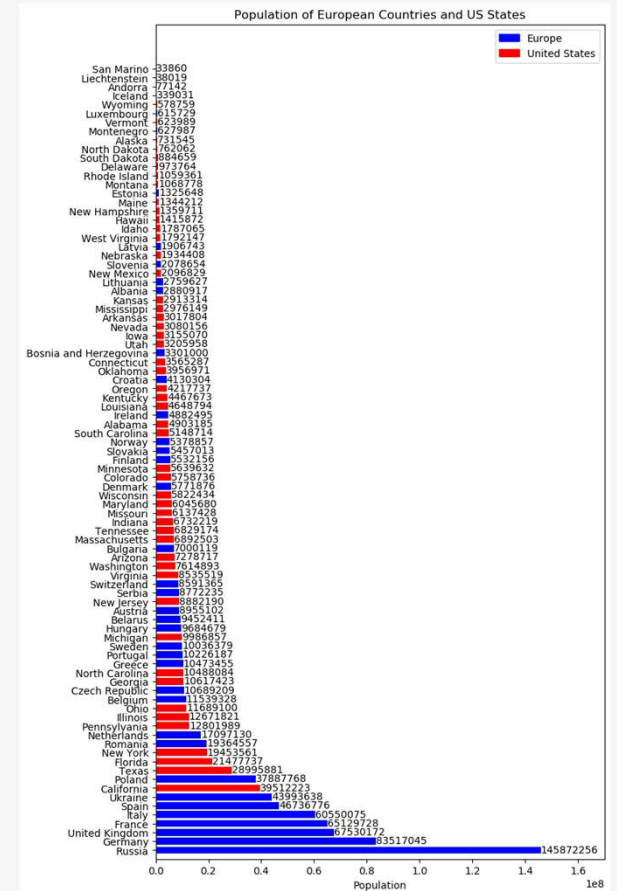
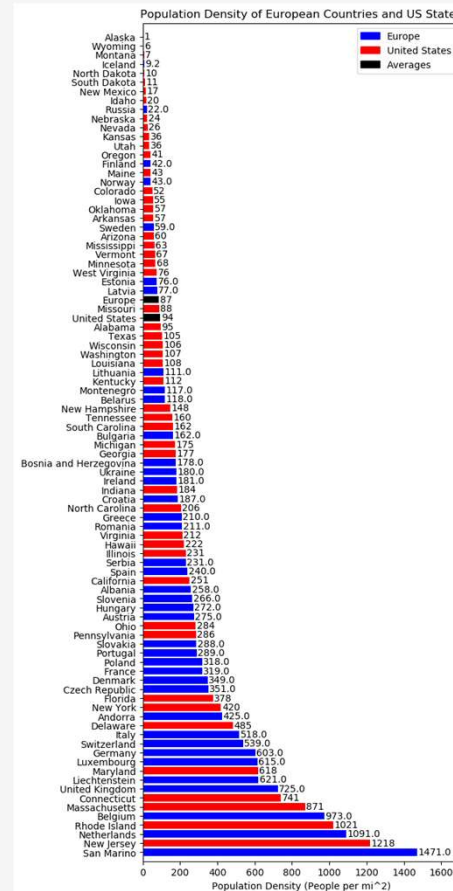
A Brief Study in Covid-19 Response by Gar Esborn

# Exploring Covid-19 Response: Populations

With a variety of comparisons being made between the US and European response to Covid-19 and a plethora of data, exploring the case curves can be telling about each government's response to the virus.

It can be helpful to explore state responses compared to individual European countries for several reasons. Firstly and most importantly, the US response is being predominantly carried out at the state and regional levels, often in populations approximate to those of European nations.

Next, even population adjusted statistics can be flawed as the virus's spread is dependent upon human interaction. This means population density and urbanization can heavily weigh on the spread. Per Capita statistics still fail to account that the population density of the US as a whole is lower than many of European Countries.

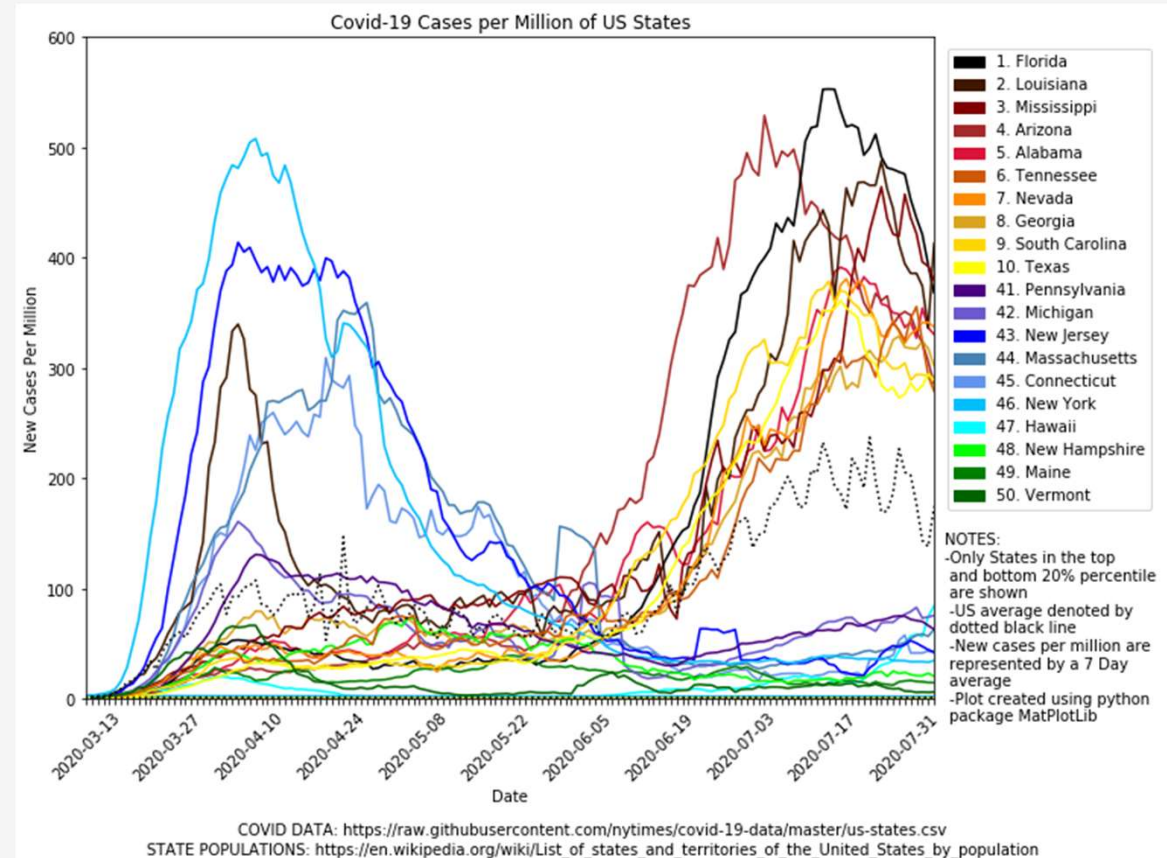


# Exploring Covid-19 Response: United States

The graph on the right demonstrates the daily change in cases per million people of American states. For simplicity, only the top and bottom 10 states in terms of current cases are shown (as of Aug 1, 2020).

Coincidentally, the graph shows every state that peaked over 300 cases per million people with the exceptions of RI and ID. The only state to have dual peaks over 300 cases per million is Louisiana.

The graph also demonstrates the successes of some regional efforts put into controlling the spread. Of the states in the Northeast, only Rhode Island currently sits above the bottom 10 in current cases (#36). This could be partially due to it having the 2<sup>nd</sup> highest population density in the US, and the highest in New England.



# Exploring Covid-19 Response: Europe

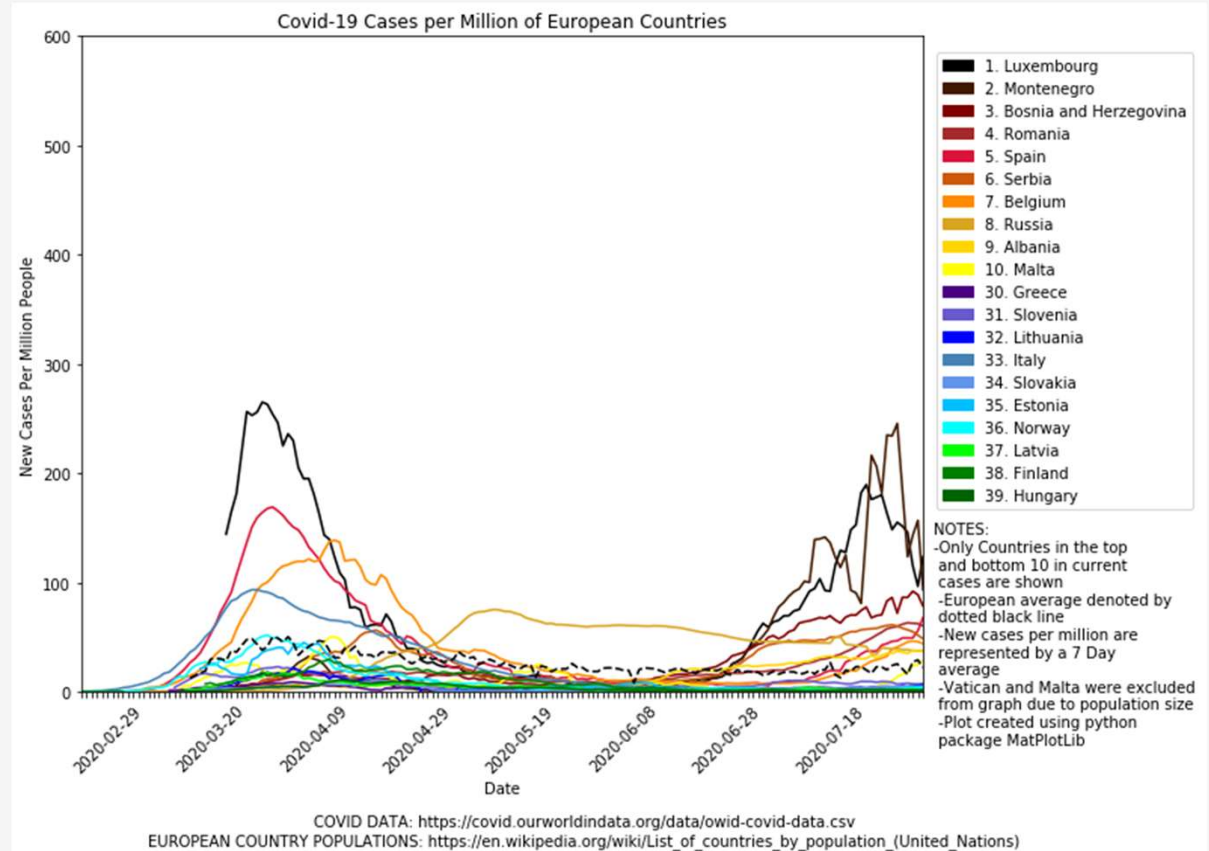
The graph on the right shows the daily change in Covid-19 cases of European countries on the same scale as the US graph (See next page for side by side). Note that the Vatican, San Marino, and Andorra were removed the plot due to their minute populations (<100,000).

The current European average of new cases per million people (~25) is less than that of all but 2 US states, Vermont and Maine.

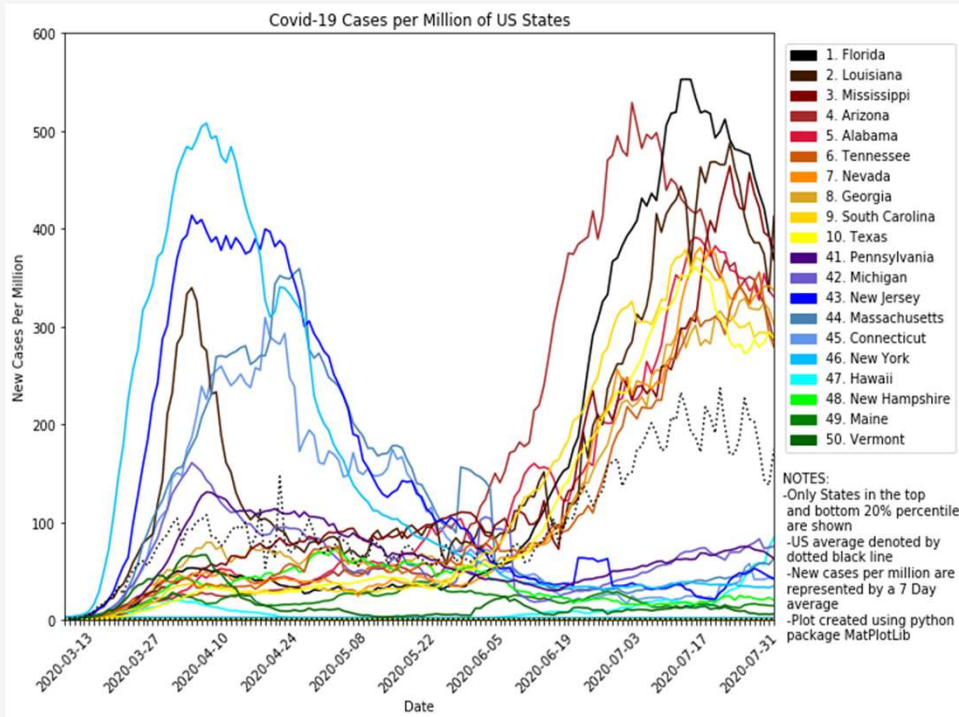
Currently, Luxembourg has the highest case rate in Europe; yet the 8<sup>th</sup> most population dense country in Europe has a lower rate than that of 34 American states.

Of the 5 European countries to peak over 200 cases per million, Montenegro has the largest population with only 630,000 people. San Marino, Luxembourg, and Andorra have population densities in the 80<sup>th</sup> percentile of our dataset. The last country of the 5 highest European peaks, Iceland, is the 6<sup>th</sup> most urban country in the world.

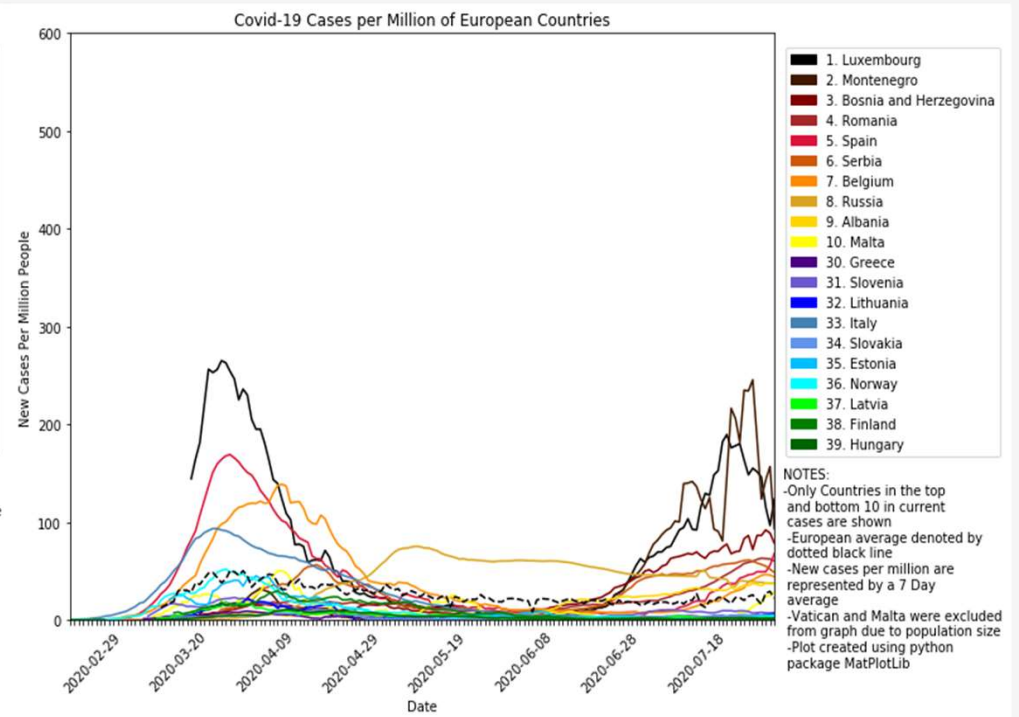
([https://en.wikipedia.org/wiki/Urbanization\\_by\\_country](https://en.wikipedia.org/wiki/Urbanization_by_country))



# Exploring Covid-19 Response: United States vs Europe



COVID DATA: <https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-states.csv>  
STATE POPULATIONS: [https://en.wikipedia.org/wiki/List\\_of\\_states\\_and\\_territories\\_of\\_the\\_United\\_States\\_by\\_population](https://en.wikipedia.org/wiki/List_of_states_and_territories_of_the_United_States_by_population)



COVID DATA: <https://covid.ourworldindata.org/data/owid-covid-data.csv>  
EUROPEAN COUNTRY POPULATIONS: [https://en.wikipedia.org/wiki/List\\_of\\_countries\\_by\\_population\\_\(United\\_Nations\)](https://en.wikipedia.org/wiki/List_of_countries_by_population_(United_Nations))



# Methodology: Converting to Daily Cases

Covid-19 data from:

<https://raw.githubusercontent.com/nytimes/covid-19-data/master/us-states.csv>

The csv file only provided total cases for each state. That data needed to be converted into Daily new cases.

First, the csv was converted into a Pandas DataFrame. Once this was done, the program iterated through that DataFrame using the code in the top section of the picture on the right.

The code would cycle through the data, subtracting the total from the day before from the total of the current day. Then, this value was added to a list.

It also requested the state the data was coming from, and if it was different from the date of the previous data point, it would move to the next state's count.

This was then placed into a new Pandas DataFrame.

```
74 # Iterate through state data
75 for i in st_df.index:
76 #-----
77 # Convert Total Cases to New Daily Cases
78 #-----
79 try:
80     #Append current elements state value to state list
81     state.append(st_df['state'][i])
82     #If adjacent data points are from the same state
83     if st_df['state'][i] == st_df['state'][i-1]:
84         #append the difference between current and previous date (new cases that day) to nc list
85         nc.append(int(st_df['cases'][i]-st_df['cases'][i-1]))
86     #When data is from a new state
87     else:
88         #Append 1st data point for that state as new cases on day 1
89         nc.append(int(st_df['cases'][i]))
90
91 #NOTE: try/except used to ignore 1st instance in list where [i-1] cannot be performed
92 except:
93     nc.append(int(st_df['cases'][i]))
94 #-----
95 # Find min and max dates for plot range
96 #-----
97 try:
98     # if beginning of data set for new state
99     # and data is from date before current minimum date
100     if st_df['state'][i] != st_df['state'][i-1] and st_df['date'][i] < dmin:
101         # replace min date
102         dmin = st_df['date'][i]
103 except:
104     # min date already set to first date in table
105     pass
106
107 try:
108     # if last date entry in current state data set
109     # and entry is from date later than current max
110     # NOTE: all state data sets should have same max date (in case datasets are incomplete)
111     if st_df['state'][i] != st_df['state'][i+1] and st_df['date'][i] > dmax:
112         dmax = st_df['date'][i]
113 except:
114     if st_df['date'][i] > dmax:
115         dmax = st_df['date'][i]
116
117 #add unique state codes into codes dictionary
118 if st_df['fips'][i] not in codes:
119     codes[st_df['fips'][i]] = st_df['state'][i]
120
121 # Create new cases DF
122 nc_df = pd.DataFrame({'state': state, 'date': st_df['date'], 'New Cases': nc})
123
```

# Methodology: Converting to a 7 Day Normalized Average

Seeing as daily case reporting is inconsistent, the data was transformed into a 7-day running average to smooth out the graphs.

This was done by creating a temporary dataframe for each state, then iterating through the new daily cases and averaging the last 7 days.

The daily cases were also normalized to each state's population using data that was scraped by parsing through the XML tree of this webpage:  
[https://en.wikipedia.org/wiki/List\\_of\\_states\\_and\\_territories\\_of\\_the\\_United\\_States\\_by\\_population](https://en.wikipedia.org/wiki/List_of_states_and_territories_of_the_United_States_by_population)

Once this iteration was complete, the new data was added to a Pandas DataFrame, and ultimately plotted into the graph on page 3. This process was duplicated for European data.

```
123
124 # head is a variable to represent the 7 in 7 day average,
125 # but was also used in debugging hence the need for its flexibility
126 head = 7
127
128 av7 = []
129 tot7 = []
130 drange = [dmin, dmax]
131
132 # Iterate through alphabetically organized states in st_pops dictionary
133 for i in st_pops:
134     peak = 0
135     #-----
136     # Convert New Daily Cases to 7 Day Average
137     #-----
138     #create temporary dataframe for current state
139     temp_df = nc_df.loc[nc_df['state'] == i]
140     temp_df = temp_df.reset_index(drop = True)
141     # Iterate through data in temp df to create 7 day av
142     for j in temp_df.index:
143         # ignore data until we have 7 days to average
144         if j-7 < 0:
145             tot7.append(None)
146             av7.append(None)
147         else:
148             # total last 7 days of cases
149             rtot = sum(temp_df['New Cases'][j-7:j])
150             # divide 7 day total by 7 days and by st population (in millions)
151             av = sum(temp_df['New Cases'][j-7:j])/(7*st_pops[i])
152             #append 7 day average per million
153             av7.append(av)
154             # append 7 day total
155             tot7.append(rtot)
156             #Note: if-statement used for debugging and to provide context to order of operations
157             if j == head:
158                 #print(i)
159                 #print('Pop:', st_pops[i])
160                 #print(temp_df['New Cases'][j-7:j])
161                 #print('Total:', rtot)
162                 #print('Tot type', type(rtot))
163                 #print("7 day av", av)
164
165 #Confirm that length of lists matches
166 print(len(nc))
167 print(len(av7))
168
169 # Create new DF with 7 day average
170 av_df = pd.DataFrame({'state': state, 'date': st_df['date'], '7 Day total': tot7, '7 Day Average': av7})
171
```

# Methodology: Webscraping Population Data

The population metrics used in this exercise required webscraping. The open\_link function is a method I recycled from some other webscraping projects. The data is filter through BeautifulSoup to correct any syntax errors. Then it is converted to an XML element tree.

Once this has been completed, the program locates the table of desired data using tbody tags. The program iterates though the table collecting the proper text entries and converting them to our desired format.

This data is compiled into a dictionary and ultimately converted into a pandas DataFrame. Later, this DataFrame will be merged with other desired metrics and converted into a SQL database table.

```
1  from bs4 import BeautifulSoup
2  import urllib.request, urllib.parse, urllib.error
3  import xml.etree.ElementTree as ET
4  import ssl
5  from state_pops import st_pops
6  import pandas as pd
7  import sqlite3 as sql
8
9  # Ignore SSL certificate errors
10 ctx = ssl.create_default_context()
11 ctx.check_hostname = False
12 ctx.verify_mode = ssl.CERT_NONE
13
14 def open_link(url, file_type = 'html'):
15     #URL Bypass
16     if file_type == 'html':
17         soup_arg = 'html.parser'
18         #Open URL, class = 'bytes'
19         uh = urllib.request.urlopen(url, context = ctx).read()
20         #Parse using BS4, class = 'bs4.BeautifulSoup'
21         data = BeautifulSoup(uh,soup_arg)
22         return ET.fromstring(data.decode())
23
24 url = 'https://en.wikipedia.org/wiki/List_of_states_and_territories_of_the_United_States_by_population_density'
25
26 # open link and convert to XML tree
27 pop_den = open_link(url)
28
29 #list of US territories and DC
30 terr = ['District of Columbia', 'Puerto Rico', 'Guam', 'US Virgin Islands', 'American Samoa', 'Northern Mariana I
31
32 st_pop_dens = {} # initialize a dictionary for population density values
33
34 tbody_tags = pop_den.findall('.//tbody') # finding pop den table in XML tree
35
36 for tr in tbody_tags[0]:
37     try:
38         if tr[0][1].text not in terr: # ignore territories of the US
39             st_pop_dens[tr[0][1].text] = int(tr[3].text) # add pop densities to dictionary
40     except:
41         pass
42
43 # create pandas DataFrame of state populations
44 US_pop_data = pd.DataFrame(list(st_pops.items()), columns = ['State', 'Population'])
45
46 # convert from millions to total population
47 for i in US_pop_data.index:
48     US_pop_data['Population'][i] = US_pop_data['Population'][i]*1000000
49
50 # create pandas DataFrame of state population densities
51 us_pop_den = pd.DataFrame(list(st_pop_dens.items()), columns = ['State', 'Pop Density (per mi^2)'])
52
53 # merge state pop dataframes
54 US_pop_data = pd.merge(US_pop_data, us_pop_den, on = 'State', how = 'left')
```