Covid-19 Spread in the US and Europe World

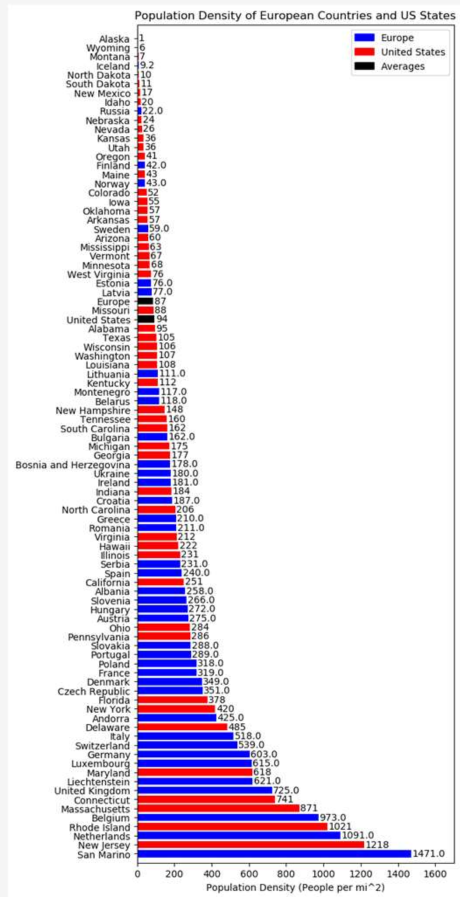
A Brief Study in Covid-19 Response

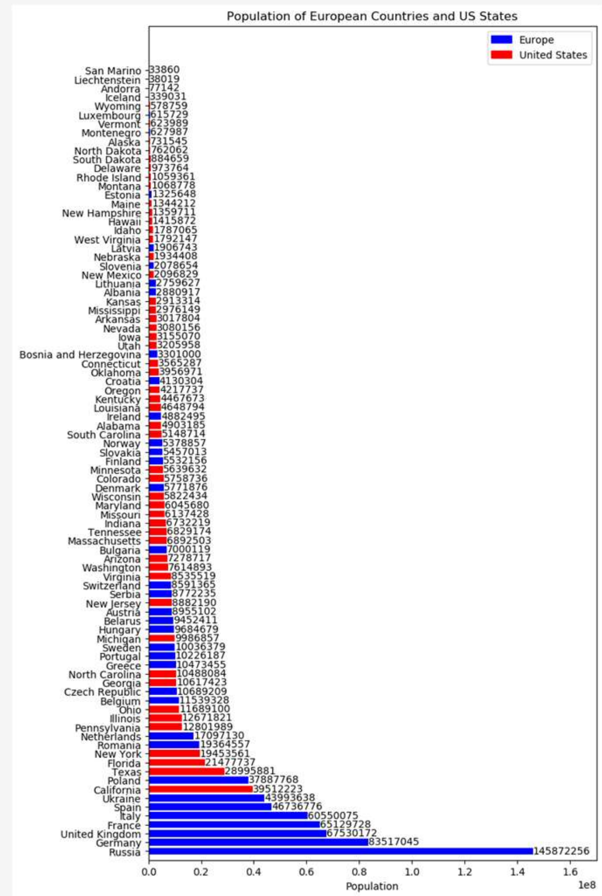
Exploring Covid-19 Response: Populations

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.



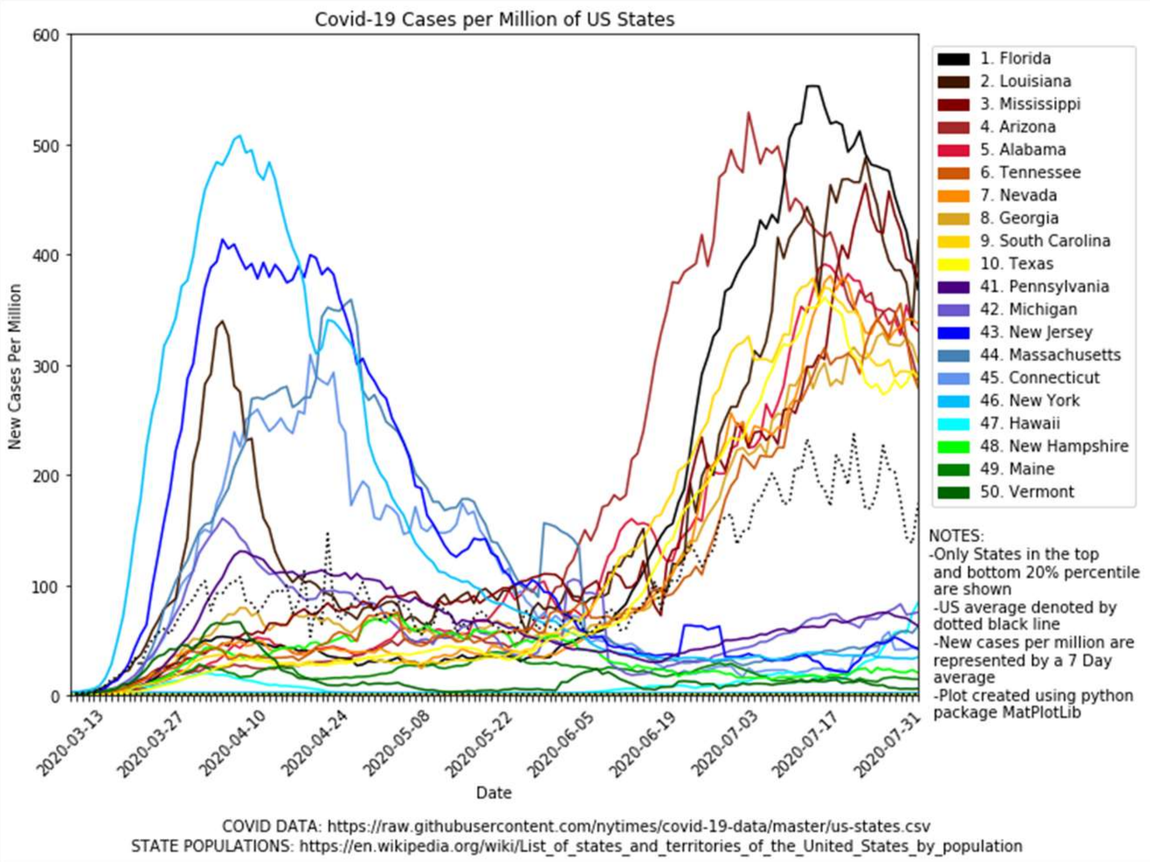


Exploring Covid-19 Response: United States

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 2nd highest population density in the US, and the highest in New England.



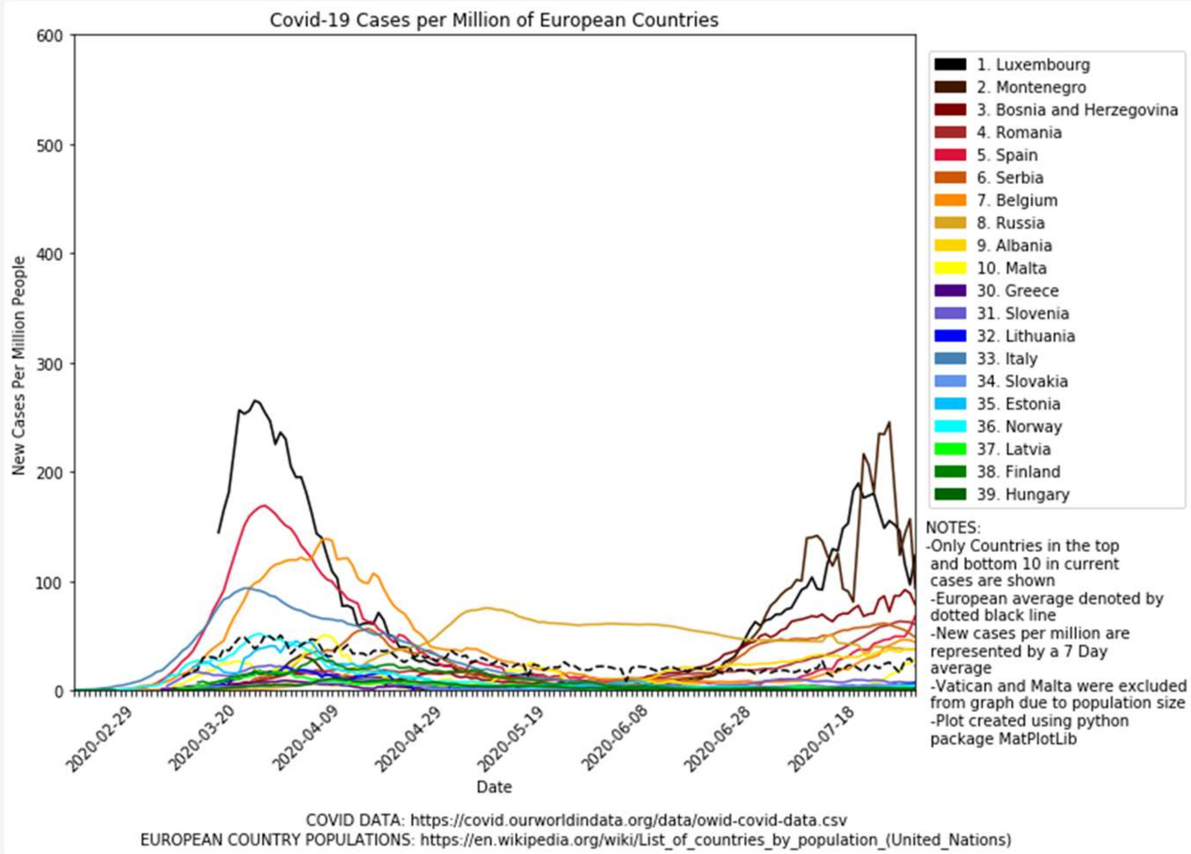
Exploring Covid-19 Response: Europe

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).

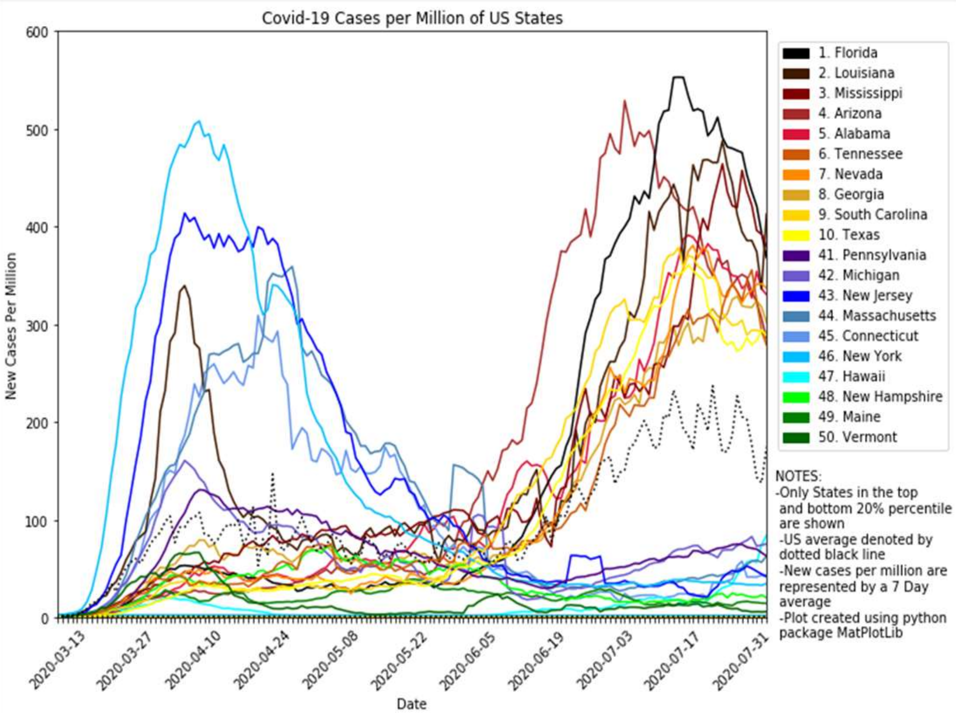
The current European average of new cases per million people (~25) is less than that of all but 2 US states, Vermont and Maine.

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

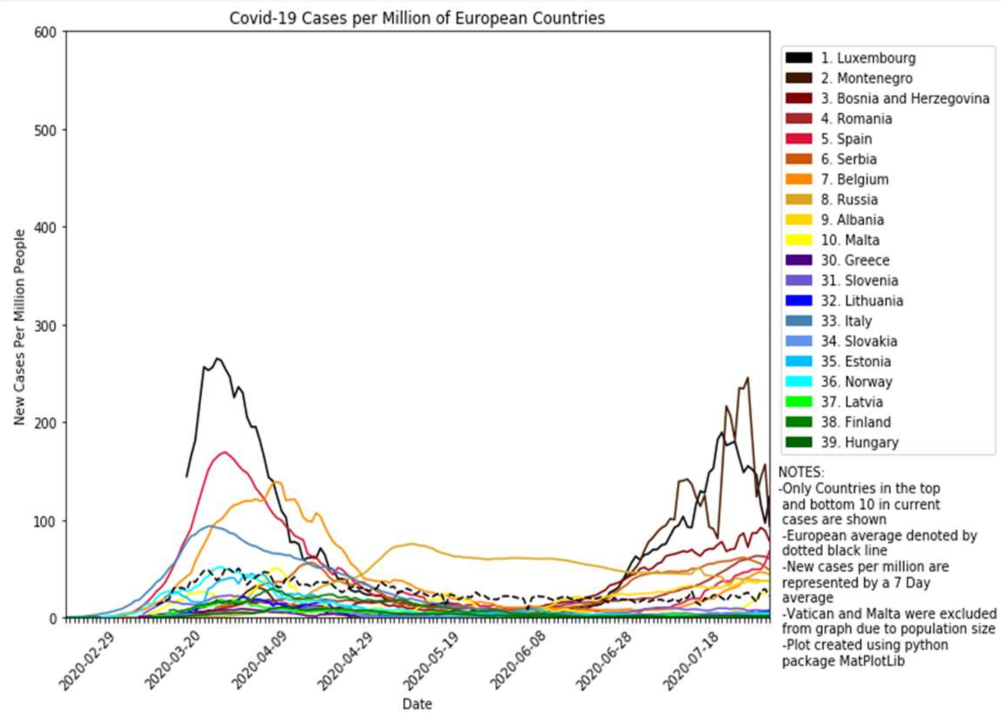
Of the 5 European countries to peak over 200 cases per million, Montenegro has the largest population with only 630,000 people. San Marino, Luxembourg, and Andorra have population densities in the 80th percentile of out dataset. The last country of the 5 highest European peaks, Iceland, is the 6th most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



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)

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.

```python

#  Iterate through state data
for i in st_df.index:
#-----------------------------------------------------------------------------
#                     Convert Total Cases to New Daily Cases
#-----------------------------------------------------------------------------
    try:
        #Append current elements state value to state list
        state.append(st_df['state'][i])
        #If adjacent data points are from the same state
        if st_df['state'][i] == st_df['state'][i-1]:
            #append the difference between current and previous date (new cases that day) to nc list
            nc.append(int(st_df['cases'][i]-st_df['cases'][i-1]))

        #When data is from a new state
        else:
            #Append 1st data point for that state as new cases on day 1
            nc.append(int(st_df['cases'][i]))

    #NOTE: try/except used to ignore 1st instance in list where [i-1] cannot be performed
    except:
        nc.append(int(st_df['cases'][i]))
#-----------------------------------------------------------------------------
#                   Find min and max dates for plot range
#-----------------------------------------------------------------------------
    try:
        # if beginning of data set for new state
        # and data is from date before current minimum date
        if st_df['state'][i] != st_df['state'][i-1] and st_df['date'][i] < dmin:
            # replace min date
            dmin = st_df['date'][i]
    except:
        # min date already set to first date in table
        pass
    try:
        # if last date entry in current state data set
        # and entry is from date later than current max
        # NOTE: all state data sets should have same max date (in case datasets are incomplete)
        if st_df['state'][i] != st_df['state'][i+1] and st_df['date'][i] > dmax:
            dmax = st_df['date'][i]
    except:
        if st_df['date'][i] > dmax:
            dmax = st_df['date'][i]
    #add unique state codes into codes dictionary
    if st_df['fips'][i] not in codes:
        codes[st_df['fips'][i]]=[st_df['state'][i]]

#  Create new cases DF
nc_df = pd.DataFrame({'state': state, 'date': st_df['date'],'New Cases': nc})

```

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

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.

```

#  head is a variable to represent the 7 in 7 day average,
#  but was also used in debugging hence the need for its flexibility
head = 7

av7 = []
tot7 = []
drange = [dmin, dmax]

# Iterate through alphabetically organized states in st_pops dictionary
for i in st_pops:
    peak = 0
#--------------------------------------------------------------------------------
#                 Convert New Daily Cases to 7 Day Average
#--------------------------------------------------------------------------------
    #create temporary dataframe for current state
    temp_df = nc_df.loc[nc_df['state'] == i]
    temp_df = temp_df.reset_index(drop = True)
    # Iterate through data in temp df to create 7 day av
    for j in temp_df.index:
        # ignore data until we have 7 days to average
        if j-7 < 0:
            tot7.append(None)
            av7.append(None)
        else:
            # total last 7 days of cases
            rtot = sum(temp_df['New Cases'][j-7:j])
            # divide 7 day total by 7 days and by st population (in millions)
            av = sum(temp_df['New Cases'][j-7:j])/(7*st_pops[i])
            #append 7 day average per million
            av7.append(av)
            # append 7 day total
            tot7.append(rtot)
            #Note: if-statement used for debugging and to provide context to order of operations
            #if j == head:
                #print(i)
                #print('Pop:', st_pops[i])
                #print(temp_df['New Cases'][j-7:j])
                #print('Total:', rtot)
                #print('Tot type', type(rtot))
                #print("7 day av", av)

#Confirm that length of lists matches
print(len(nc))
print(len(av7))

# Create new DF with 7 day average
av_df = pd.DataFrame({'state': state, 'date': st_df['date'], '7 Day total': tot7,'7 Day Average': av7})

```

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.

```python
from bs4 import BeautifulSoup
import urllib.request, urllib.parse, urllib.error
import xml.etree.ElementTree as ET
import ssl
from state_pops import st_pops
import pandas as pd
import sqlite3 as sql

# Ignore SSL certificate errors
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

def open_link(url, file_type = 'html'):
    #URL Bypass
    if file_type == 'html':
        soup_arg = 'html.parser'
    #Open URL, class = 'bytes'
    uh = urllib.request.urlopen(url, context = ctx).read()
    #Parse using BS4, class = 'bs4.BeautifulSoup'
    data = BeautifulSoup(uh,soup_arg)
    return ET.fromstring(data.decode())


url = 'https://en.wikipedia.org/wiki/List_of_states_and_territories_of_the_United_States_by_population_density'

# open link and convert to XML tree
pop_den = open_link(url)

#list of US territories and DC
terr = ['District of Columbia', 'Puerto Rico', 'Guam', 'US Virgin Islands', 'American Samoa', 'Northern Mariana I

st_pop_dens = {}    # initialize a dictionary for population density values

tbody_tags = pop_den.findall('.//tbody')    # finding pop den table in XML tree

for tr in tbody_tags[0]:
    try:
        if tr[0][1].text not in terr:   # ignore territories of the US
            st_pop_dens[tr[0][1].text] = int(tr[3].text) # add pop densities to dictionary
    except:
        pass

# create pandas DataFrame of state populations
US_pop_data = pd.DataFrame(list(st_pops.items()),columns = ['State','Population'])

# convert from millions to total population
for i in US_pop_data.index:
    US_pop_data['Population'][i] = US_pop_data['Population'][i]*1000000

# create pandas DataFrame of state population densities
us_pop_den = pd.DataFrame(list(st_pop_dens.items()), columns = ['State', 'Pop Density (per mi^2)'])

# merge state pop dataframes
US_pop_data = pd.merge(US_pop_data, us_pop_den, on = 'State', how = 'left')
```