Covid-19 Exploratory Data Analysis

Covid-19 is a disease caused by SARS-CoV-2. The World Health Organization declared the disease a pandemic on March 11th, 2020. Since then there have been numerous outlets that have compiled data and presented information pertaining to its spread. Information from Worldometer, a reference website reputed by the American Library Association, will be used as the primary dataset. Worldometer gathers and compiles information from various sources, including government communication channels as well as local media. It provides live updates on the total cases, new cases, total deaths, new deaths,recoveries and critical cases by country, territory or conveyance pertaining to Covid-19.

Objective:

This exploratory analysis will provide a broad level overview of the Covid-19 pandemic. It will contain information from the most recent update on the Worldometer website. Other datasets will be used in order to supplement the information from the Worldometer dataset in order to aid in visualization and geomappiThis analysis will show the steps for data wrangling using Pandas in order to make data ready for visualization, correlation analysis, and geomapping.

Steps in Analysis:

1. Import required libraries
2. Gather data
    - Webscraping with BeautifulSoup and Selenium
    - Reading in CSV files
    - Saving information into Pandas dataframes
3. Clean data
    - Manage null and missing values
    - Drop unwanted rows and columns
    - Rename columns and observations
    - Format datatypes
    - Add columns with calculations
    - Concatenate dataframes
4. Visualization
    - Seaborn | MatPlotLib | Plotly
    - Pearson Correlation Heatmaps
    - Bar Plots
    - Pair Plots
    - Pie Plots

- Bubble Plots

5. Global GDP Exploration

- Scatter Plots
- Pearson Correlation

6. GeoMapping

- Plotly Library
- Choropleth
- ScatterGeo

Data Sources:

- Worldometer Covid-19 Live Information (Countries and US States)
- GDP per Capita (Countries)
- Geospatial Data (Countries and US States)
- Land Size (Countries)

Import Libraries

In [2]:

```
# Data Gathering
import requests                                                 # HTTP requests
import requests, io                                             # Process I/O data
import urllib.request                                           # Open URL
from bs4 import BeautifulSoup                                   # Webscraping
import pandas as pd                                             # Pandas Dataframes
import re                                                       # Regular Expressions (
from numpy import inf                                           # Numpy Infinite Values
import json                                                     # Parse JSON from strin
import js2xml                                                   # Parse Javascript into
import time                                                     # Time access and conve
from selenium import webdriver                                  # Automated web browing
from selenium.webdriver.firefox.options import Options          # Headless browsing
from selenium.webdriver import firefox
from webdriverdownloader import GeckoDriverDownloader
from webdriver_manager.firefox import GeckoDriverManager


#-------------------------------------------------------------------------------

# Visualization
import seaborn as sns                                           # Visualization based o
import matplotlib as mpl                                        # Visualization for Pyt
from matplotlib import pyplot as plt                            # MATLAB style plotting
%matplotlib inline
from scipy import stats                                         # Statistical functions
from scipy.stats import norm                                    # Normal Continuous ran
import plotly.express as px                                     # Interactive Visualiza
from plotly.subplots import make_subplots                       # Subplots for Plotly
import plotly.graph_objects as go                               # Plotly Traces
```

```
#-------------------------------------------------------------------------------

# Statistical Analysis
import numpy as np                                          # Numerical data array
import statsmodels.formula.api as smf                       # Statistical testing
import statsmodels.api as sm                                # Used for Correlation

#-------------------------------------------------------------------------------

# GeoData
import os, sys                                              # Use operating system
import geopandas as gpd                                     # Geospatial data proce
from numpy import int64                                     # Process int64
from geopandas import GeoDataFrame                          # Process Geodataframes
import geopy                                                # Python Geocoding tool
from geopy.geocoders import Nominatim                       # OpenStreetMap API

#-------------------------------------------------------------------------------

# Interactive Plots to HTML5
# import plotly.offline as py                                 # Use plotly standalo
# from plotly.offline import plot                             # Use plotly in brows
# py.init_notebook_mode()                                     # Initialize notebook

#-------------------------------------------------------------------------------

# Exceptions
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)  # Ignore future depreca

#-------------------------------------------------------------------------------

# Confirm Libraries Import
print('Libraries Imported Successfully.')
```

```
Libraries Imported Successfully.
```

Data Gathering

WORLD DATA

Beautiful Soup library will be used to scrape the table from the Worldometer website. This table will then be stored into a Pandas dataframe called `global_data`. It will contain information about each country's Covid-19 cases. This dataframe will then be concatenated with a GDP dataframe called `country_gdp`, and a land size dataframe called `country_area`, later used to calculate population density

In [3]:

```
# Importing table with BeatifulSoup
url = 'https://www.worldometers.info/coronavirus/'  # Assign url
requests.get(url)
```

```
page = requests.get(url).text                          # Store request into page

# Use lxml parser to store nested html structure into Beautiful Soup object
soup = BeautifulSoup(page, 'lxml')

# Find table using html tags
table_data = soup.find('table', id = 'main_table_countries_today')

headers = []
for i in table_data.find_all('th'):          # This loop will iterate through table and
    title = i.text                           # tags <th> then save table header text int
    headers.append(title)                    # to be appended into headers array

# Create columns from headers strings
raw_data = pd.DataFrame(columns = headers)

for j in table_data.find_all('tr')[1:]:      # This loop will iterate through table rows
        row_data = j.find_all('td')          # Each <td> element will be stored inside r
        row = [tr.text for tr in row_data]   # Each element will then be stored in row v
        length = len(raw_data)
        raw_data.loc[length] = row

# Get updated date and time string
a= soup.findAll('div',attrs={"style":"font-size:13px; color:#999; margin-top:5px; text-

last_update = (a[0].string)
print(last_update)

# Save imported data table to 'global_data' dataframe
global_data = raw_data

# Strip preceding empty spaces in columns and replace spaces with underscore
global_data.columns= global_data.columns.to_series().apply(lambda x: x.strip())
global_data.columns = (global_data.columns.str.strip().str.upper().str.replace(' ', '_'
global_data.head()
```

```
Last updated: June 01, 2021, 01:38 GMT
```

Out[3]:

#	COUNTRY,OTHER	TOTALCASES	NEWCASES	TOTALDEATHS	NEWDEATHS	TOTALRECOVERED	NEV
0	\nNorth America\n	39,802,124	+932	893,942	+61	32,614,975	
1	\nAsia\n	51,277,108	+482	686,320	+4	47,308,382	
2	\nSouth America\n	28,808,932		780,416		25,979,567	
3	\nEurope\n	46,612,663		1,071,822		43,532,268	
4	\nAfrica\n	4,885,566		130,997		4,400,757	

5 rows × 22 columns

GDP Per Capita

This dataset will be used to see if GDP per capita has any influence in the amount of Covid-19 cases per country, the death rate, survival rate, etc. Each country's GDP per capita will be contained in the

dataframe, which will then be added to the `global_data` dataframe.

In [4]:
```
# Read in csv and save to 'country_gdp' dataframe
country_gdp = pd.read_csv("Resources\csvGDP.csv")
country_gdp.head()
```

Out[4]:

	rank	country	imfGDP	unGDP	gdpPerCapita	pop
0	1	United States	2.219812e+13	18624475000000	66678.0263	332915.073
1	2	China	1.546810e+13	11218281029298	10710.3777	1444216.107
2	3	Japan	5.495420e+12	4936211827875	43596.8659	126050.804
3	4	Germany	4.157120e+12	3477796274497	49548.2308	83900.473
4	5	India	3.257720e+12	2259642382872	2337.9495	1393409.038

Land Area (MI^2)

Land area dataset will be used to calculate the population density. Population density will be used to see if there is correlation with total cases, death rate, survival rate, etc. Each country's land size will be contained in the dataframe, which will then be added to the `global_data` dataframe

In [5]:
```
# Import table with BeatifulSoup
url3 = 'https://www.worldometers.info/geography/largest-countries-in-the-world/'
requests.get(url3)
page3 = requests.get(url3).text

soup3 = BeautifulSoup(page3, 'lxml')

table_data3 = soup3.find('table', id = 'example2')

headers = []
for i in table_data3.find_all('th'):
    title = i.text
    headers.append(title)

raw_data3 = pd.DataFrame(columns = headers)

for j in table_data3.find_all('tr')[1:]:
        row_data = j.find_all('td')
        row = [tr.text for tr in row_data]
        length = len(raw_data3)
        raw_data3.loc[length] = row

# Save imported data table to 'country_area' dataframe
country_area = raw_data3

# Strip preceding empty spaces in columns and replace spaces with underscore
country_area.columns= country_area.columns.to_series().apply(lambda x: x.strip())
country_area.columns = (country_area.columns.str.strip().str.upper().str.replace(' ', '
country_area.head()
```

Out[5]:

	#	COUNTRY	TOT._AREA_(KM^2)	TOT._AREA_(MI^2)	LAND_AREA_(KM^2)	LAND_AREA_(MI^2)	%_OF_WOR

	#	COUNTRY	TOT._AREA_(KM²)	TOT._AREA_(MI²)	LAND_AREA_(KM²)	LAND_AREA_(MI²)	%_OF_WOR
0	1	Russia	17,098,242	6,601,665	16,376,870	6,323,142	
1	2	Canada	9,984,670	3,855,101	9,093,510	3,511,022	
2	3	China	9,706,961	3,747,877	9,388,211	3,624,807	
3	4	United States	9,372,610	3,618,783	9,147,420	3,531,837	
4	5	Brazil	8,515,767	3,287,955	8,358,140	3,227,095	

UNITED STATES DATA

Using the same scraping method previously outlined, this data will primarily be used in visualization and geomapping. It will be placed into a dataframe called `us_data` and concatenated with `state_gdp`, which, will include information about each state's GDP per capita.

In [6]:
```
# Import table with BeatifulSoup
url2 = 'https://www.worldometers.info/coronavirus/country/us/'
requests.get(url2)
page2 = requests.get(url2).text

soup2 = BeautifulSoup(page2, 'lxml')

table_data2 = soup2.find('table', id = 'usa_table_countries_today')

headers = []
for i in table_data2.find_all('th')[0:13]:   # Specify only columns 0 to 12 be appended
    title = i.text
    headers.append(title)

raw_data2 = pd.DataFrame(columns = headers)

for j in table_data2.find_all('tr')[1:]:
        row_data = j.find_all('td')[:13]    # Specify only columns 0 to 12
        row = [tr.text for tr in row_data]
        length = len(raw_data2)
        raw_data2.loc[length] = row

# Save imported data table to 'us_data' dataframe
us_data = raw_data2

# Strip preceding empty spaces in columns and replace spaces with underscore
us_data.columns= us_data.columns.to_series().apply(lambda x: x.strip())
us_data.columns = (us_data.columns.str.strip().str.upper().str.replace(' ', '_'))
us_data.head()
```

Out[6]:

	#	USASTATE	TOTALCASES	NEWCASES	TOTALDEATHS	NEWDEATHS	TOTALRECOVERED	ACTIVECA
0		USA Total	34,113,146		609,767		27,863,840	5,639,
1	1	\nCalifornia	3,789,731	\n	\n63,247		\n2,054,430	\n1,672,

	#	USASTATE	TOTALCASES	NEWCASES	TOTALDEATHS	NEWDEATHS	TOTALRECOVERED	ACTIVECA
2	2	\nTexas	2,954,513	\n	\n51,728		\n2,835,681	\n67,
3	3	\nFlorida	2,320,818	\n	\n36,774		\n1,943,113	\n340,
4	4	\nNew York	2,154,361	\n	\n53,594		\n1,718,623	\n382,

Data Cleaning

Data cleaning is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data. It is necessary for data to be consistent and organized for it to be represented accurately later on during visualization and modeling.

Covered in this section:

- Dealing with null values
- Formatting text and datatypes
- Adding, removing, and renaming columns
- Renaming string variables contained in rows
- Editing CSV data

WORLD DATA

This dataset has inconsistencies in the form of newline characters, special characters, unnecessary columns and rows, and mislabeled country names. Null values, datatypes, and column names must be formatted before merging `global_data` with `country_gdp` and `country_area` . Several values are missing from the `country_gdp` dataset; after merging with the `global_data` dataframe, a dataframe containing `COUNTRY` and `GDP_PER_CAPITA` variables will be exported to csv, filled in manually, and added back into the dataframe.

In [7]:
```
# Format data
global_data=global_data.replace('\n','', regex=True)            # Remove newline charac
global_data=global_data.replace(',','', regex=True)             # Remove commas
global_data.columns=global_data.columns.str.replace('\n','')    # Remove newline charac
global_data = global_data.applymap(lambda x: x.strip('+'))      # 'applymap' applies la

# Drop top buttom unwanted rows
global_data= global_data.drop(global_data.index[[0,1,2,3,4,5,6,7]]).reset_index(drop=Tr

# Drop tail unwanted rows
global_data.drop(global_data.tail(8).index,inplace=True)
global_data.head()
```

Out[7]:

	#	COUNTRY,OTHER	TOTALCASES	NEWCASES	TOTALDEATHS	NEWDEATHS	TOTALRECOVERED	NEV
0	1	USA	34113146		609767		27863840	
1	2	India	28173655		331909		25939504	
2	3	Brazil	16547674		462966		14964631	
3	4	France	5667324		109528		5333597	
4	5	Turkey	5249404		47527		5114624	

5 rows × 22 columns

In [8]:
```
# Rename 'global_data' columns
global_data.rename(columns={global_data.columns[10]: 'TOTCASES_PER_1M'}, inplace= True)
global_data = global_data.rename(columns={'SERIOUS,CRITICAL': 'SERIOUS_CRITICAL',
                                          'COUNTRY,OTHER': 'COUNTRY',
                                          'DEATHS/1M_POP': 'DEATH_PER_1M',
                                          'TESTS/1M_POP': 'TESTS_PER_1M'})

# Drop 'global_data' columns that will not be used in analysis
global_data = global_data.drop(['#','1_CASEEVERY_X_PPL', '1_DEATHEVERY_X_PPL',
       '1_TESTEVERY_X_PPL', 'NEWCASES', 'NEWDEATHS', 'NEWRECOVERED', 'NEW_CASES/1M_POP'

global_data.columns    # Lists column names
```

Out[8]:
```
Index(['COUNTRY', 'TOTALCASES', 'TOTALDEATHS', 'TOTALRECOVERED', 'ACTIVECASES',
       'SERIOUS_CRITICAL', 'TOTCASES_PER_1M', 'DEATH_PER_1M', 'TOTALTESTS',
       'TESTS_PER_1M', 'POPULATION', 'CONTINENT'],
      dtype='object')
```

In [9]:
```
# Check datatypes of 'global_data' dataframe
global_data.dtypes
```

Out[9]:
```
COUNTRY             object
TOTALCASES          object
TOTALDEATHS         object
TOTALRECOVERED      object
ACTIVECASES         object
SERIOUS_CRITICAL    object
TOTCASES_PER_1M     object
DEATH_PER_1M        object
TOTALTESTS          object
TESTS_PER_1M        object
POPULATION          object
CONTINENT           object
dtype: object
```

In [10]:
```
# Convert global_data dataframe objects to strings
global_data = global_data.astype('string')
global_data.dtypes
```

Out[10]:
```
COUNTRY             string
TOTALCASES          string
```

```
TOTALDEATHS          string
TOTALRECOVERED       string
ACTIVECASES          string
SERIOUS_CRITICAL     string
TOTCASES_PER_1M      string
DEATH_PER_1M         string
TOTALTESTS           string
TESTS_PER_1M         string
POPULATION           string
CONTINENT            string
dtype: object
```

In [11]:
```
# Rename row information to keep consistent with 'country_gdp' dataframe
global_data.COUNTRY= global_data.COUNTRY.replace("USA","United States")
global_data.COUNTRY= global_data.COUNTRY.replace("UK","United Kingdom")
```

In [12]:
```
global_data.head()
```

Out[12]:

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCA
0	United States	34113146	609767	27863840	5639539	6129	
1	India	28173655	331909	25939504	1902242	8944	
2	Brazil	16547674	462966	14964631	1120077	8318	
3	France	5667324	109528	5333597	224199	2945	
4	Turkey	5249404	47527	5114624	87253	1339	

In [13]:
```
# Choose columns that will be converted to numeric
cols = ['TOTALCASES', 'TOTALDEATHS','TOTALRECOVERED', 'ACTIVECASES', 'SERIOUS_CRITICAL'
       'TOTCASES_PER_1M', 'DEATH_PER_1M', 'TOTALTESTS', 'TESTS_PER_1M',
       'POPULATION']

# Change global_data datatypes to numeric variables
global_data[cols] = global_data[cols].apply(pd.to_numeric, errors='coerce', axis=1)

# Convert NAN and INF values to 0
global_data.fillna(0, inplace=True)
global_data.replace(np.nan, 0, inplace=True)
global_data.replace(np.inf, 0, inplace=True)

# Convert columns 1 to 10 to int
for col in global_data.columns[1:11]:
    global_data[col]=global_data[col].apply(int)

global_data.dtypes
```

Out[13]:
```
COUNTRY              string
TOTALCASES            int64
TOTALDEATHS           int64
TOTALRECOVERED        int64
ACTIVECASES           int64
SERIOUS_CRITICAL      int64
```

```
TOTCASES_PER_1M      int64
DEATH_PER_1M         int64
TOTALTESTS           int64
TESTS_PER_1M         int64
POPULATION           int64
CONTINENT           string
dtype: object
```

In [14]:
```
# China's is out of order, so we have to sort TOTALCASES
global_data = global_data.sort_values(['TOTALCASES'], ascending= False)
global_data.head()
```

Out[14]:

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCA
0	United States	34113146	609767	27863840	5639539	6129	
1	India	28173655	331909	25939504	1902242	8944	
2	Brazil	16547674	462966	14964631	1120077	8318	
3	France	5667324	109528	5333597	224199	2945	
4	Turkey	5249404	47527	5114624	87253	1339	

Country GDP Per Capita

In [15]:
```
# Rename 'country_gdp' columns to be consistent with 'global_data' columns for later co
country_gdp = country_gdp.rename(columns = {'country': 'COUNTRY',
                                            'gdpPerCapita': 'GDP_PER_CAPITA'})

# Drop 'country_gdp' unnecessary columns
country_gdp = country_gdp.drop(['rank','imfGDP', 'unGDP', 'pop'], axis=1)

country_gdp.columns     # Lists column names
```

Out[15]:
```
Index(['COUNTRY', 'GDP_PER_CAPITA'], dtype='object')
```

In [16]:
```
# Check country_gdp datatype
country_gdp.dtypes
```

Out[16]:
```
COUNTRY           object
GDP_PER_CAPITA    float64
dtype: object
```

In [17]:
```
# Convert 'COUNTRY' variable from 'object' to 'string'
country_gdp['COUNTRY'] = country_gdp['COUNTRY'].astype('string')
country_gdp.dtypes
```

Out[17]:
```
COUNTRY           string
GDP_PER_CAPITA    float64
dtype: object
```

Country Land Area

In [18]:
```
# Remove newline characters
country_area=country_area.replace('\n','', regex=True)
country_area=country_area.replace(',','', regex=True)
country_area=country_area.replace('square miles','', regex=True)

# Drop and Rename Columns
country_area = country_area.drop(['#', 'TOT._AREA_(KM²)', 'TOT._AREA_(MI²)', '%_OF_WORL
country_area = country_area.rename(columns={'LAND_AREA_(MI²)': 'LAND_AREA'})

# Convert country_area dataframe objects to strings and numeric
country_area = country_area.astype('string')
country_area['LAND_AREA'] = country_area['LAND_AREA'].apply(pd.to_numeric)
country_area.head()
```

Out[18]:

	COUNTRY	LAND_AREA
0	Russia	6323142
1	Canada	3511022
2	China	3624807
3	United States	3531837
4	Brazil	3227095

In [19]:
```
# Rename rows to keep consistent with 'country_gdp' dataframe to successfully merge dat
country_area.COUNTRY= country_area.COUNTRY.replace("United Arab Emirates","UAE")
country_area.COUNTRY= country_area.COUNTRY.replace("State of Palestine","Palestine")
country_area.COUNTRY= country_area.COUNTRY.replace("Republic of North Macedonia","North
country_area.COUNTRY= country_area.COUNTRY.replace("South Korea","S. Korea")
country_area.COUNTRY= country_area.COUNTRY.replace("Côte d'Ivoire","Ivory Coast")
country_area.COUNTRY= country_area.COUNTRY.replace("DR Congo","DRC")
country_area.COUNTRY= country_area.COUNTRY.replace("China Hong Kong SAR","Hong Kong")
country_area.COUNTRY= country_area.COUNTRY.replace("Central African Republic","CAR")
country_area.COUNTRY= country_area.COUNTRY.replace("Turks and Caicos Islands","Turks an
country_area.COUNTRY= country_area.COUNTRY.replace("Saint Vincent and the Grenadines","
country_area.COUNTRY= country_area.COUNTRY.replace("Saint Barthélemy","St. Barth")
country_area.COUNTRY= country_area.COUNTRY.replace("Brunei Darussalam","Brunei")
country_area.COUNTRY= country_area.COUNTRY.replace("Wallis and Futuna Islands","Wallis
country_area.COUNTRY= country_area.COUNTRY.replace("China Macao SAR","Macao")
country_area.COUNTRY= country_area.COUNTRY.replace("Holy See","Vatican City")
country_area.COUNTRY= country_area.COUNTRY.replace("Saint Pierre and Miquelon","Saint P
```

Concatenation

The `global_data` dataframe is ready to be concatenated with the `country_gdp` dataframe. A left merge will be used in order to add the `'GDP_PER_CAPITA'` variable to the existing `global_data` dataframe. Merging on `'COUNTRY'` will allow each dataframe to use the `'COUNTRY'` column as its index to match each row from `global_data` to `country_gdp` based on the country's name. This explains why each country's name in both datasets had to be exact. Any

country name that is not identical in both dataframes will not merge, returning an NAN value in the `'GDP_PER_CAPITA'` column. `country_area` will be merged the same way.

In [20]:
```
# Merge datasets
global_data = global_data.merge(country_gdp, on='COUNTRY', how='left')
global_data.head()
```

Out[20]:

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCA
0	United States	34113146	609767	27863840	5639539	6129	
1	India	28173655	331909	25939504	1902242	8944	
2	Brazil	16547674	462966	14964631	1120077	8318	
3	France	5667324	109528	5333597	224199	2945	
4	Turkey	5249404	47527	5114624	87253	1339	

Calculations will be made in order to add a `'DEATH_RATE'`, `'SURVIVAL_RATE'`, and `'PERCENT_TESTS_POSITIVE'` column to the `global_data` dataframe

In [21]:
```
# Death Rate = Total Deaths / Total Cases
global_data['DEATH_RATE'] = global_data.TOTALDEATHS / global_data.TOTALCASES

# Survival Rate = Total Recovered / Total Cases
global_data['SURVIVAL_RATE'] = global_data.TOTALRECOVERED / global_data.TOTALCASES

# Percentage of Tests Positive = Total Cases / Total Tests
global_data['PERCENT_TESTS_POSITIVE'] = global_data.TOTALCASES / global_data.TOTALTESTS
global_data = global_data.replace([np.inf, -np.inf], 0)     # If numerator is '0', infi
global_data.head()
```

Out[21]:

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCA
0	United States	34113146	609767	27863840	5639539	6129	
1	India	28173655	331909	25939504	1902242	8944	
2	Brazil	16547674	462966	14964631	1120077	8318	
3	France	5667324	109528	5333597	224199	2945	
4	Turkey	5249404	47527	5114624	87253	1339	

Null values

The `country_gdp` dataframe was missing GDP per capita values for several countries before it was added to `global_data`. These need to be added manually. First, a new dataframe will be created,

`gdp_entry` , which will only contain the `'COUNTRY'` and `'GDP_PER_CAPITA'` columns from `global_data` using a `copy()` function. This dataframe will then be exported to a csv file called `gdp_entry.csv` . Once the GDP values are manually added to the csv, it will be saved as `gdp_entry2.csv` and read into a dataframe called `gdp_df_update` . The `GDP_PER_CAPITA` column must first be dropped from `global_data` before merging the `gdp_df_update` dataframe. The rows `Diamond Princess` and `MS Zaandam` will be dropped because they are ships, but first `'COUNTRY'` must be set as the index in order to drop the rows by axis 0. After that, the index is reset.

In [22]:
```
# Check for NAN values
global_data[global_data.isna().any(axis=1)]
```

Out[22]:

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TO
19	Czechia	1661272	30108	1620098	11066	90	
39	UAE	570836	1680	550525	18631	0	
85	S. Korea	140799	1963	131463	7373	158	
118	DRC	31651	782	27665	3204	0	
120	Cabo Verde	30439	264	28710	1465	23	
124	Réunion	24901	189	22796	1916	35	
127	French Guiana	24036	116	9995	13925	32	
130	Mayotte	20176	171	2964	17041	6	
135	Guadeloupe	16530	221	2250	14059	29	
146	Curaçao	12272	122	12121	29	9	
147	Martinique	11960	95	98	11767	9	
150	Congo	11658	153	8208	3297	0	
161	CAR	7091	98	6859	134	2	
162	Timor-Leste	6994	16	4352	2626	0	
171	Gibraltar	4295	94	4193	8	0	
174	Channel Islands	4066	86	3956	24	0	
182	Turks and Caicos	2417	17	2381	19	2	
186	St. Vincent Grenadines	2035	12	1831	192	2	
187	Saint Martin	1915	12	1399	504	7	
190	Caribbean Netherlands	1613	17	1579	17	0	
191	Isle of Man	1592	29	1561	2	1	
194	St. Barth	1005	1	462	542	0	

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TO
195	Faeroe Islands	718	1	671	46	0	
196	Diamond Princess	712	13	699	0	0	
199	Wallis and Futuna	445	7	438	0	0	
202	Brunei	242	3	228	11	0	
208	Falkland Islands	63	0	63	0	0	
209	Macao	51	0	49	2	0	
211	Vatican City	27	0	27	0	0	
212	Saint Pierre Miquelon	25	0	25	0	0	
215	Western Sahara	10	1	8	1	0	
216	MS Zaandam	9	2	7	0	0	
220	Saint Helena	2	0	2	0	0	

In [23]:
```
# Create dataframe to have NAN values filled in manually
gdp_entry = global_data[['COUNTRY', 'GDP_PER_CAPITA']].copy()
gdp_entry.head()
```

Out[23]:

	COUNTRY	GDP_PER_CAPITA
0	United States	66678.0263
1	India	2337.9495
2	Brazil	9638.1461
3	France	43958.7034
4	Turkey	9519.3901

In [24]:
```
#Export dataframe to csv in order to manually add GDP information
gdp_entry.to_csv('Resources\gdp_entry.csv', index = False)
```

In [25]:
```
# Add updated csv to dataframe
gdp_df_update= pd.read_csv('Resources\gdp_entry2.csv')

# Drop current country_gdp column
global_data= global_data.drop('GDP_PER_CAPITA', axis=1)

# Merge updated gdp dataframe back into 'global_data'
```

```
global_data = global_data.merge(gdp_df_update, on='COUNTRY', how='left')

# Drop the rows that are ships and not countries
global_data = global_data.set_index('COUNTRY')
global_data = global_data.drop(['Diamond Princess','MS Zaandam'], axis = 0).reset_index
global_data.head()
```

Out[25]:

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCA
0	United States	34113146	609767	27863840	5639539	6129	
1	India	28173655	331909	25939504	1902242	8944	
2	Brazil	16547674	462966	14964631	1120077	8318	
3	France	5667324	109528	5333597	224199	2945	
4	Turkey	5249404	47527	5114624	87253	1339	

In [26]:

```
# Merge 'country_area' with 'global_data'
# Strip will be used on the 'COUNTRY' column in order remove any missed spaces before o
global_data.COUNTRY = global_data.COUNTRY.str.strip()
country_area.COUNTRY = country_area.COUNTRY.str.strip()
global_data = global_data.merge(country_area, on='COUNTRY', how='left')
```

In [27]:

```
# Percentage of Tests Positive
global_data['POPULATION_DENSITY'] = global_data.POPULATION / global_data.LAND_AREA
```

In [28]:

```
# Check dataframe for null values
global_data[global_data.isna().any(axis=1)]
```

Out[28]:

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCAS

In [29]:

```
global_data.head()
```

Out[29]:

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCA
0	United States	34113146	609767	27863840	5639539	6129	
1	India	28173655	331909	25939504	1902242	8944	
2	Brazil	16547674	462966	14964631	1120077	8318	
3	France	5667324	109528	5333597	224199	2945	
4	Turkey	5249404	47527	5114624	87253	1339	

UNITED STATES DATA

This dataset has inconsistencies in the form of newline characters, special characters, and unnecessary columns and rows. Datatypes and column names will be formatted before merging `us_data` with `state_gdp`. Woldometer does not have `ACTIVECASES` and `TOTALRECOVERED` data for Hawaii and South Carolina available in the tabe; this information will be retrieved from each state's URL.

In [30]:
```
#remove newline characters and special characters in dataframe
us_data=us_data.replace('\n','', regex=True)
us_data=us_data.replace(',','', regex=True)
us_data.columns=us_data.columns.str.replace('\n','')
us_data = us_data.applymap(lambda x: x.strip('+'))
us_data.head()
```

Out[30]:

	#	USASTATE	TOTALCASES	NEWCASES	TOTALDEATHS	NEWDEATHS	TOTALRECOVERED	ACTIVECAS
0		USA Total	34113146		609767		27863840	56395
1	1	California	3789731		63247		2054430	16720
2	2	Texas	2954513		51728		2835681	671
3	3	Florida	2320818		36774		1943113	3409
4	4	New York	2154361		53594		1718623	3821

In [31]:
```
#Drop top unwanted rows and reset index
us_data= us_data.drop(us_data.index[[0]]).reset_index(drop=True)

#drop tail unwanted rows
us_data.drop(us_data.tail(13).index,inplace=True)
us_data.head()
```

Out[31]:

	#	USASTATE	TOTALCASES	NEWCASES	TOTALDEATHS	NEWDEATHS	TOTALRECOVERED	ACTIVECAS
0	1	California	3789731		63247		2054430	16720
1	2	Texas	2954513		51728		2835681	671
2	3	Florida	2320818		36774		1943113	3409
3	4	New York	2154361		53594		1718623	3821
4	5	Illinois	1382186		25223		1301109	558

In [32]:
```
# Rename columns
us_data.rename(columns = {us_data.columns[8]: 'TOTCASES_PER_1M'}, inplace = True)

us_data = us_data.rename(columns = {'USASTATE':'STATE',
                                    'DEATHS/1M_POP': 'DEATH_PER_1M',
```

```
                                    'TOT CASES/1M_POP' : 'TOTCASES_PER_1M',
                                    'TESTS/1M_POP': 'TESTS_PER_1M'})

# Drop unwanted columns
us_data = us_data.drop(['#', 'NEWCASES', 'NEWDEATHS'], axis = 1)
us_data.columns
```

Out[32]:
```
Index(['STATE', 'TOTALCASES', 'TOTALDEATHS', 'TOTALRECOVERED', 'ACTIVECASES',
       'TOTCASES_PER_1M', 'DEATH_PER_1M', 'TOTALTESTS', 'TESTS_PER_1M',
       'POPULATION'],
      dtype='object')
```

In [33]:
```
# Convert global_data dataframe objects to strings
us_data = us_data.astype('string')
us_data.dtypes
```

Out[33]:
```
STATE              string
TOTALCASES         string
TOTALDEATHS        string
TOTALRECOVERED     string
ACTIVECASES        string
TOTCASES_PER_1M    string
DEATH_PER_1M       string
TOTALTESTS         string
TESTS_PER_1M       string
POPULATION         string
dtype: object
```

In [34]:
```
# Change us_data datatypes to numeric
cols = ['TOTALCASES', 'TOTALDEATHS','TOTALRECOVERED', 'ACTIVECASES',
       'TOTCASES_PER_1M', 'DEATH_PER_1M', 'TOTALTESTS', 'TESTS_PER_1M',
       'POPULATION']

us_data[cols] = us_data[cols].apply(pd.to_numeric, errors='coerce', axis=1)

# Convert columns to integer
for col in us_data.columns[1:]:
    us_data[col]=pd.Series(us_data[col], dtype=int)

us_data.dtypes
```

Out[34]:
```
STATE               string
TOTALCASES           int32
TOTALDEATHS          int32
TOTALRECOVERED     float64
ACTIVECASES        float64
TOTCASES_PER_1M      int32
DEATH_PER_1M         int32
TOTALTESTS           int32
TESTS_PER_1M         int32
POPULATION           int32
dtype: object
```

In [35]:
```
# ADD COLUMNS

# Death Rate per Country
us_data['DEATH_RATE'] = us_data.TOTALDEATHS / us_data.TOTALCASES

# Survival Rate
```

```
us_data['SURVIVAL_RATE'] = us_data.TOTALRECOVERED / us_data.TOTALCASES

# Percentage of Tests Positive
us_data['PERCENT_TESTS_POSITIVE'] = us_data.TOTALCASES / us_data.TOTALTESTS

#us_data = us_data.replace([np.inf, -np.inf], 0)
us_data.head()
```

Out[35]:

	STATE	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	TOTCASES_PER_1M	DEATH_
0	California	3789731	63247	2054430.0	1672054.0	95913	
1	Texas	2954513	51728	2835681.0	67104.0	101894	
2	Florida	2320818	36774	1943113.0	340931.0	108057	
3	New York	2154361	53594	1718623.0	382144.0	110744	
4	Illinois	1382186	25223	1301109.0	55854.0	109076	

Null Values

In order to get the most accurate visualization, null values must be taken care of. Some visualization libraries will not process cells that have values, so those must be coverted to NaN. In this dataset, NaN values can still be found an added to the dataframe using webscraping.

In [36]:
```
us_data[us_data.isna().any(axis =1)]
```

Out[36]:

	STATE	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	TOTCASES_PER_1M	DEATI
16	Wisconsin	674420	7856	NaN	NaN	115831	
20	Alabama	543405	11146	NaN	NaN	110827	
22	Louisiana	470685	10576	NaN	NaN	101249	
34	Nebraska	223368	2249	NaN	NaN	115471	
45	Maine	67780	825	NaN	NaN	50424	
49	Hawaii	36276	500	NaN	NaN	25621	

South Carolina, District of Columbia, and Hawaii are missing values in three columns. They will be added using BeautifulSoup from their individual state pages.

Get Hawaii Recovered Cases

The number of recovered cases can be found directly on the state's page in the `span` tag. It will be parsed using an lxml interpreter and stored in a soup variable. The variable `a` will store the elements of the `class: "maincounter-number"` tag, and then `find` will be used on `a` to find

the contents of the `span` tag. This process will be replicated for South Carolina. District of Columbia's GDP Per Capita will be found manually and added to the dataframe

In [37]:
```
url4 = 'https://www.worldometers.info/coronavirus/usa/hawaii/'
requests.get(url4)
page4 = requests.get(url4).text

soup4 = BeautifulSoup(page4, 'lxml')
# Get updated date and time string
a= soup4.find('div',attrs={"class": "maincounter-number", "style": "color:#8ACA2B "})

hawaii_recovered = a.find('span').text
hawaii_recovered = hawaii_recovered.replace(",", "")
hawaii_recovered = int(hawaii_recovered)
print(hawaii_recovered)
```

```
13182
```

Get Hawaii Active Cases

The information for Hawaii's active cases is stored in an interactive table on the state's website. `Selenium` will be used in order to browse the page using a Firefox webdriver. The elements of the JavaScript `Highchart.chart tag` will be stored in the variable `temp` . The data array will store all elements of `series` , which contains the number of active cases over time. The most recently updated value for Hawaii's active cases will be used. This process will be replicated for South Carolina.

In [38]:
```
website = "https://www.worldometers.info/coronavirus/usa/hawaii/"

option = Options()
option.headless = True
driver = webdriver.Firefox(options= option)
driver.get(website)
time.sleep(5)

temp = driver.execute_script('return window.Highcharts.charts[3]'
                             '.series[0].options.data')
data = [item for item in temp]
hawaii_active = (data[-1])
print(hawaii_active)
```

```
22594
```

Get South Carolina Recovered Cases

In [39]:
```
url5 = 'https://www.worldometers.info/coronavirus/usa/south-carolina/'
requests.get(url5)
page5 = requests.get(url5).text
soup5 = BeautifulSoup(page5, 'lxml')
# Get updated date and time string
a= soup5.find('div',attrs={"class": "maincounter-number", "style": "color:#8ACA2B "})

scarolina_recovered = a.find('span').text
```

```
scarolina_recovered = scarolina_recovered.replace(",", "")
scarolina_recovered = int(scarolina_recovered)
print(scarolina_recovered)
```

```
544306
```

Get South Carolina Active Cases

In [40]:

```
website = "https://www.worldometers.info/coronavirus/usa/south-carolina/"

driver = webdriver.Firefox(executable_path=GeckoDriverManager().install())
option = Options()
option.headless= True
driver = webdriver.Firefox(options= option)
driver.get(website)
time.sleep(5)

temp = driver.execute_script('return window.Highcharts.charts[3]'
                             '.series[0].options.data')

data = [item for item in temp]
scarolina_active = (data[-1])
scarolina_active = int(scarolina_active)
print(scarolina_active)
```

```
====== WebDriver manager ======
Current firefox version is 85.0.2
Get LATEST driver version for 85.0.2
Driver [C:\Users\omkar\.wdm\drivers\geckodriver\win64\v0.29.1\geckodriver.exe] found in
cache
39220
```

In [41]:

```
us_data=us_data.set_index('STATE')
```

REPLACE VALUES FOR HAWAII AND SOUTH CAROLINA

In [42]:

```
us_data.loc['Hawaii','TOTALRECOVERED'] = hawaii_recovered
us_data.loc['Hawaii','ACTIVECASES'] = hawaii_active
us_data.loc['Hawaii','SURVIVAL_RATE'] = (hawaii_recovered / (us_data.loc['Hawaii', 'TOT

us_data.loc['South Carolina','TOTALRECOVERED'] = scarolina_recovered
us_data.loc['South Carolina','ACTIVECASES'] = scarolina_active
us_data.loc['South Carolina','SURVIVAL_RATE'] = (scarolina_recovered / (us_data.loc['So
```

In [43]:

```
# DataFrame cleaned of all null values
us_data.isnull().any()
```

Out[43]:

```
TOTALCASES              True
TOTALDEATHS             True
TOTALRECOVERED          True
ACTIVECASES             True
TOTCASES_PER_1M         True
DEATH_PER_1M            True
TOTALTESTS              True
```

```
TESTS_PER_1M              True
POPULATION                True
DEATH_RATE                True
SURVIVAL_RATE             True
PERCENT_TESTS_POSITIVE    True
dtype: bool
```

In [44]:
```
# Reset index
us_data = us_data.reset_index()
us_data.head()
```

Out[44]:

	STATE	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	TOTCASES_PER_1M	DEATH_
0	California	3789731.0	63247.0	2054430.0	1672054.0	95913.0	
1	Texas	2954513.0	51728.0	2835681.0	67104.0	101894.0	
2	Florida	2320818.0	36774.0	1943113.0	340931.0	108057.0	
3	New York	2154361.0	53594.0	1718623.0	382144.0	110744.0	
4	Illinois	1382186.0	25223.0	1301109.0	55854.0	109076.0	

Visualization

</br>

Covered in this section:

- Bar chart subplots using Seaborn and MatPlotLib
- Interactive Pie chart subplots using Plotly Express
- Pearson Correlation Heatmap using Seaborn

Top Countries

In [45]:
```
# Create dataframe of the top ten countries with the most confirmed cases
top_cases = global_data[:10]
top_cases.head()
```

Out[45]:

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCA
0	United States	34113146	609767	27863840	5639539	6129	
1	India	28173655	331909	25939504	1902242	8944	
2	Brazil	16547674	462966	14964631	1120077	8318	
3	France	5667324	109528	5333597	224199	2945	
4	Turkey	5249404	47527	5114624	87253	1339	

Bar Subplot

A barplot shows the relationship between a numeric and a categoric variable. Each entity of the categoric variable is represented as a bar. The size of the bar represents its numeric value. A barplot is used to aggregate the categorical data according to some methods and by default it's the mean. It can also be understood as a visualization of the group by action. To use this plot we choose a categorical column for the x-axis and a numerical column for the y-axis, and we see that it creates a plot taking a mean per categorical column.

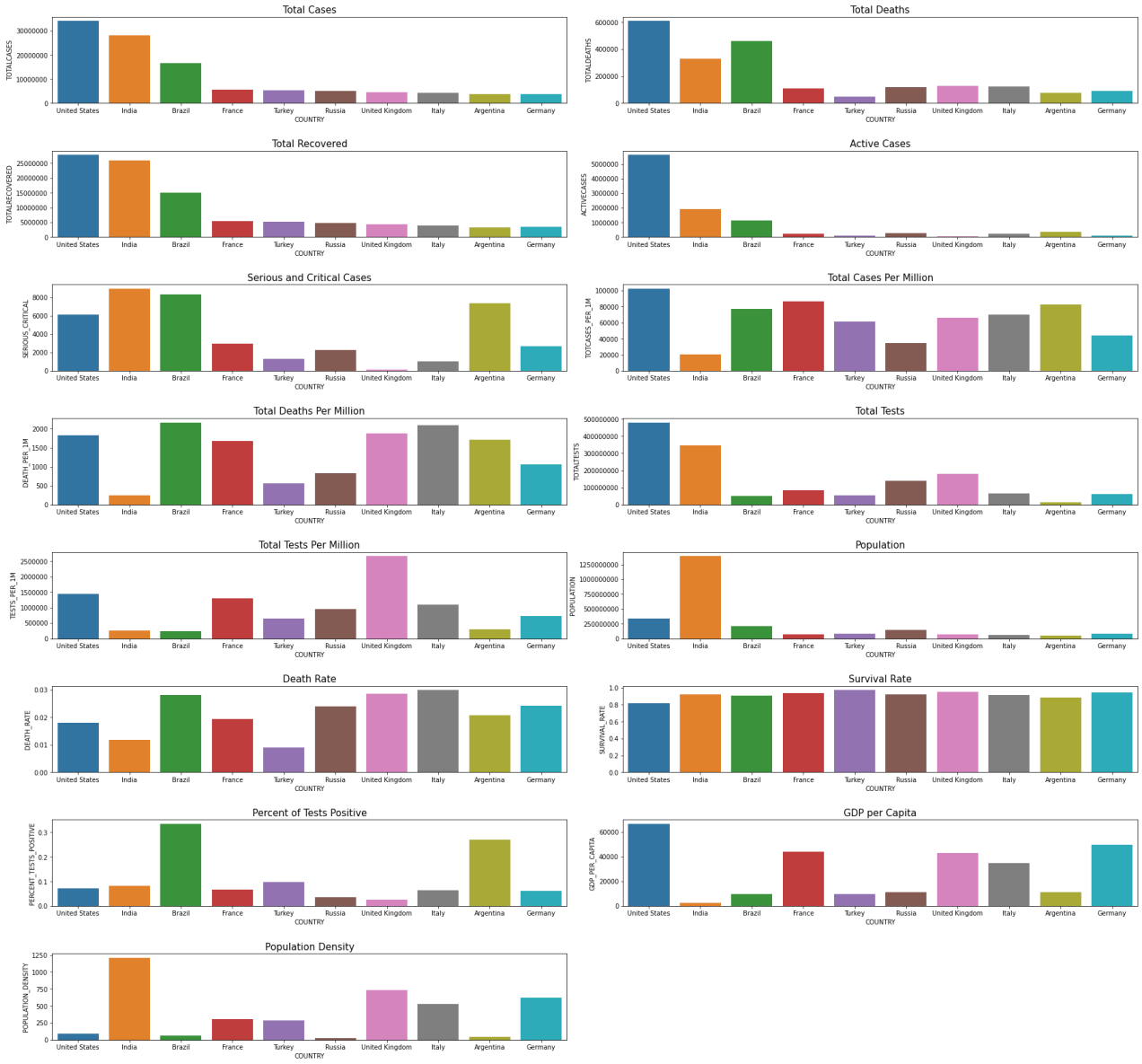
A barplot will be used to plot the top cases. Because there are fewer X variables, the variation between the bar bars will be easier to visualize

In [46]:

```
# Use Seaborn and Matplotlib to create bar subplot for all variable in 'topcases' dataf

fig, ax = plt.subplots(8, 2, figsize= (25,25))

ax1 = sns.barplot(top_cases["COUNTRY"], top_cases["TOTALCASES"], data= top_cases, ax =
ax1.set_title("Total Cases", fontsize=15)

ax2 = sns.barplot(top_cases["COUNTRY"], top_cases["TOTALDEATHS"], data= top_cases, ax =
ax2.set_title("Total Deaths", fontsize=15)

ax3 = sns.barplot(top_cases["COUNTRY"], top_cases["TOTALRECOVERED"], data= top_cases, a
ax3.set_title("Total Recovered", fontsize=15)

ax4 = sns.barplot(top_cases["COUNTRY"], top_cases["ACTIVECASES"], data= top_cases, ax =
ax4.set_title("Active Cases", fontsize=15)

ax5 = sns.barplot(top_cases["COUNTRY"], top_cases["SERIOUS_CRITICAL"], data= top_cases,
ax5.set_title("Serious and Critical Cases", fontsize=15)

ax6 = sns.barplot(top_cases["COUNTRY"], top_cases["TOTCASES_PER_1M"], data= top_cases,
ax6.set_title("Total Cases Per Million", fontsize=15)

ax7 = sns.barplot(top_cases["COUNTRY"], top_cases["DEATH_PER_1M"], data= top_cases, ax
ax7.set_title("Total Deaths Per Million", fontsize=15)

ax8 = sns.barplot(top_cases["COUNTRY"], top_cases["TOTALTESTS"], data= top_cases, ax =
ax8.set_title("Total Tests", fontsize=15)

ax9 = sns.barplot(top_cases["COUNTRY"], top_cases["TESTS_PER_1M"], data= top_cases, ax
ax9.set_title("Total Tests Per Million", fontsize=15)

ax10 = sns.barplot(top_cases["COUNTRY"], top_cases["POPULATION"], data= top_cases, ax =
ax10.set_title("Population", fontsize=15)

ax11 = sns.barplot(top_cases["COUNTRY"], top_cases["DEATH_RATE"], data= top_cases, ax =
ax11.set_title("Death Rate", fontsize=15)

ax12 = sns.barplot(top_cases["COUNTRY"], top_cases["SURVIVAL_RATE"], data= top_cases, a
ax12.set_title("Survival Rate", fontsize=15)

ax13 = sns.barplot(top_cases["COUNTRY"], top_cases["PERCENT_TESTS_POSITIVE"], data= top
```

```
ax13.set_title("Percent of Tests Positive", fontsize=15)

ax14 = sns.barplot(top_cases["COUNTRY"], top_cases["GDP_PER_CAPITA"], data= top_cases,
ax14.set_title("GDP per Capita", fontsize=15)

ax15 = sns.barplot(top_cases["COUNTRY"], top_cases["POPULATION_DENSITY"], data= top_cas
ax15.set_title("Population Density", fontsize=15)

ax16 = sns.barplot(top_cases["COUNTRY"], top_cases["POPULATION_DENSITY"], data= top_cas
ax16.set_title("Population Density", fontsize=15)

# Convert y axis label from 'e' notation to plain
for ax in ax.flat:
    ax.ticklabel_format(axis='y', style = 'plain')

# Delete empty subplot
fig.delaxes(ax = ax16)


fig.suptitle('TOP TEN COUNTRIES BAR PLOTS' + ', ' + last_update, fontsize = 30)     # B
fig.tight_layout(pad=0.6, w_pad=0.5, h_pad=3)                                       # P
fig.subplots_adjust(top=.92)                                                        # S
```

TOP TEN COUNTRIES BAR PLOTS, Last updated: June 01, 2021, 01:38 GMT



Interactive Pie Subplot

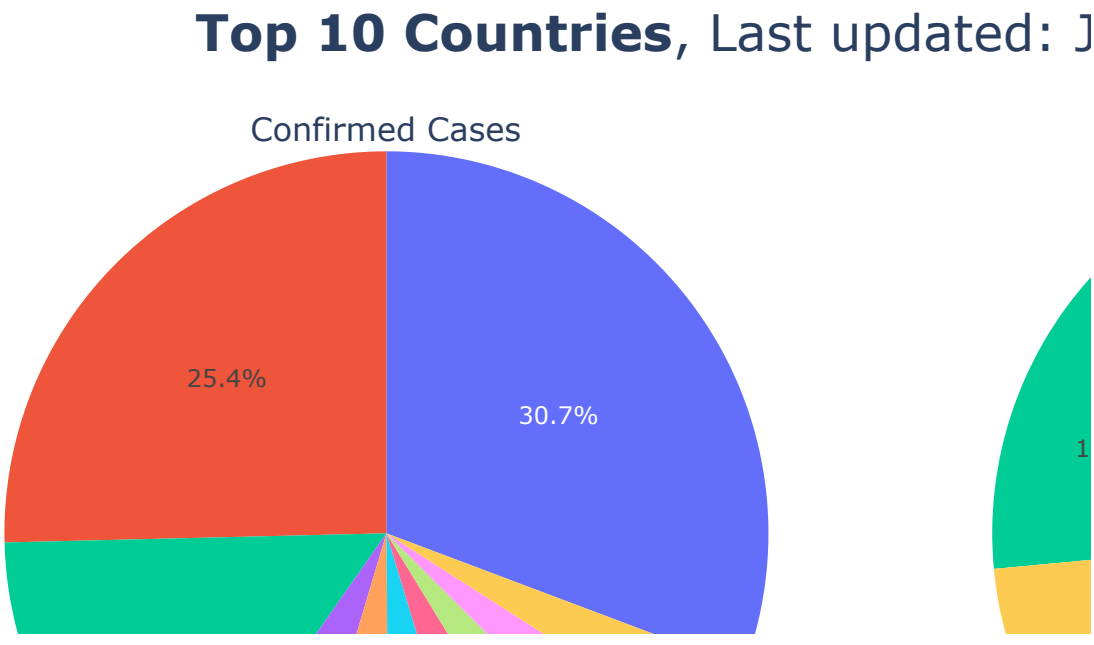
The interactive pie charts will be made using `Plotly Express` . A pie chart is a circular analytical chart, which is divided into region to symbolize numerical percentage. In px.pie, data anticipated by the sectors of the pie to set the values. All sectors are classified bynames. Pie chart is used to show the percentage with the next corresponding slice of pie. Pie chart easily understandable due to its different portions and color codings. Labels can be found by hovering over the chart.

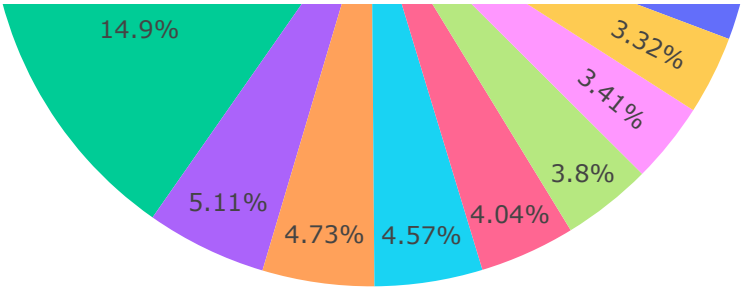
In [47]:

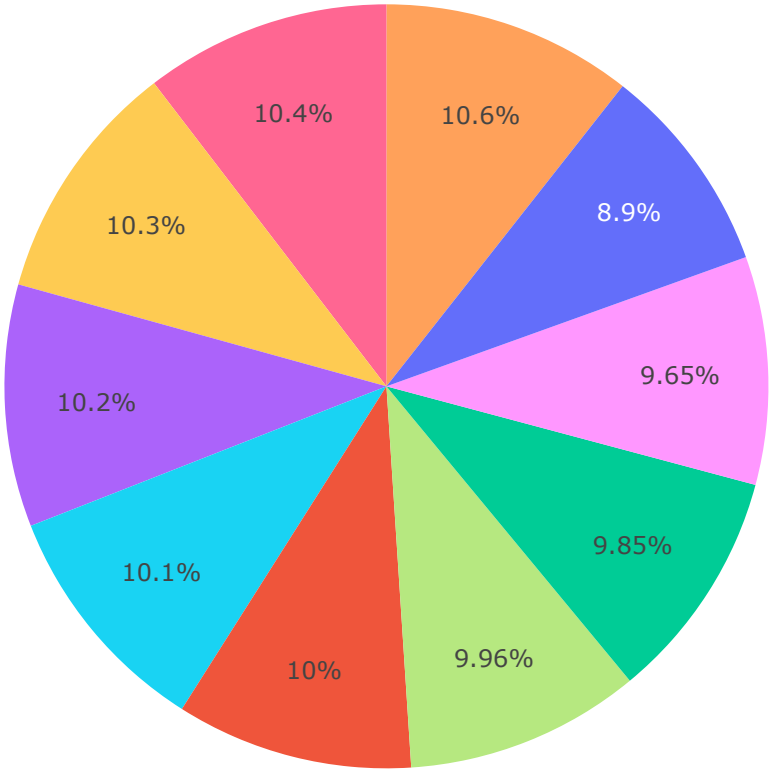
```
fig = make_subplots(rows=7,
                    cols=2,
                    specs=[[{"type": "pie"}, {"type": "pie"}], [{"type": "pie"}, {"type
                    subplot_titles=('Confirmed Cases', 'Death Rate', 'Survival Rate', '
                    vertical_spacing=0.05,)

# Add pie charts
fig.add_trace(go.Pie(                    # Add pie plot using Plotly Graph Object
    values=top_cases.TOTALCASES,         # y variable
```

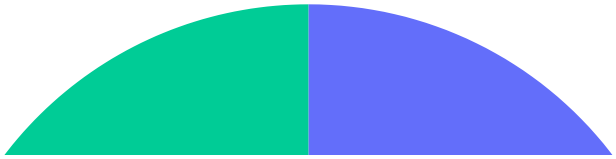
```
     labels=top_cases.COUNTRY,          # x variable
     domain=dict(x=[0, 0.5]),           # Position on Domain
     name="T_Cases"),                   # Tag
     row=1, col=1)                      # Position on Plot

fig.add_trace(go.Pie(
     values=top_cases.DEATH_RATE,
     labels=top_cases.COUNTRY,
     domain=dict(x=[0.5, 1.0]),
     name="Death Rate"),
     row=1, col=2)

fig.add_trace(go.Pie(
     values=top_cases.SURVIVAL_RATE,
     labels=top_cases.COUNTRY,
     domain=dict(x=[0, 0.5]),
     name="Survival"),
     row=2, col=1)

fig.add_trace(go.Pie(
     values=top_cases.PERCENT_TESTS_POSITIVE,
     labels=top_cases.COUNTRY,
     domain=dict(x=[0.5, 1.0]),
     name="Tests Pos"),
     row=2, col=2)

fig.add_trace(go.Pie(
     values=top_cases.TOTALDEATHS,
     labels=top_cases.COUNTRY,
     domain=dict(x=[0.5, 1.0]),
     name="T_Deaths"),
     row=3, col=1)

fig.add_trace(go.Pie(
     values=top_cases.TOTALRECOVERED,
     labels=top_cases.COUNTRY,
     domain=dict(x=[0, 0.5]),
     name="T_Recovered"),
     row=3, col=2)

fig.add_trace(go.Pie(
     values=top_cases.ACTIVECASES,
     labels=top_cases.COUNTRY,
     domain=dict(x=[0.5, 1.0]),
     name="Active"),
     row=4, col=1)

fig.add_trace(go.Pie(
     values=top_cases.SERIOUS_CRITICAL,
     labels=top_cases.COUNTRY,
     domain=dict(x=[0.5, 1.0]),
     name="Serious"),
     row=4, col=2)

fig.add_trace(go.Pie(
     values=top_cases.DEATH_PER_1M,
     labels=top_cases.COUNTRY,
     domain=dict(x=[0.5, 1.0]),
     name="DeathsPM"),
     row=5, col=1)
```

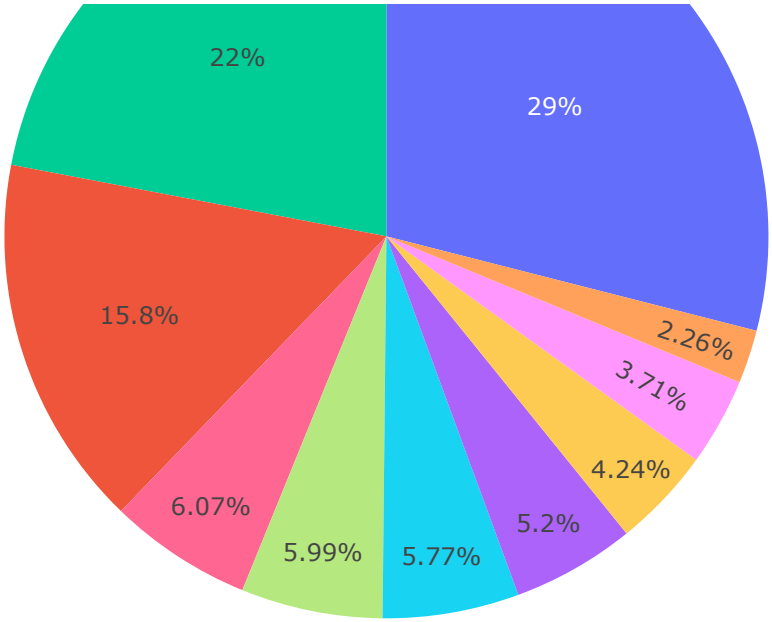
```
fig.add_trace(go.Pie(
      values=top_cases.TOTALTESTS,
      labels=top_cases.COUNTRY,
      domain=dict(x=[0, 0.5]),
      name="T_Tests"),
      row=5, col=2)

fig.add_trace(go.Pie(
      values=top_cases.TESTS_PER_1M,
      labels=top_cases.COUNTRY,
      domain=dict(x=[0.5, 1.0]),
      name="TestsPM"),
      row=6, col=1)

fig.add_trace(go.Pie(
      values=top_cases.POPULATION,
      labels=top_cases.COUNTRY,
      domain=dict(x=[0, 0.5]),
      name="Population"),
      row=6, col=2)

fig.add_trace(go.Pie(
      values=top_cases.POPULATION_DENSITY,
      labels=top_cases.COUNTRY,
      domain=dict(x=[0, 0.5]),
      name="Population Density"),
      row=7, col=1)

# Update layout
fig['layout'].update(
      height=4000,
      width=1150,
      title='<b>Top 10 Countries</b>' + ', ' + last_update,
      title_x= .5,
      title_font_size = 25,
      hovermode = "x unified")

fig.show()
```

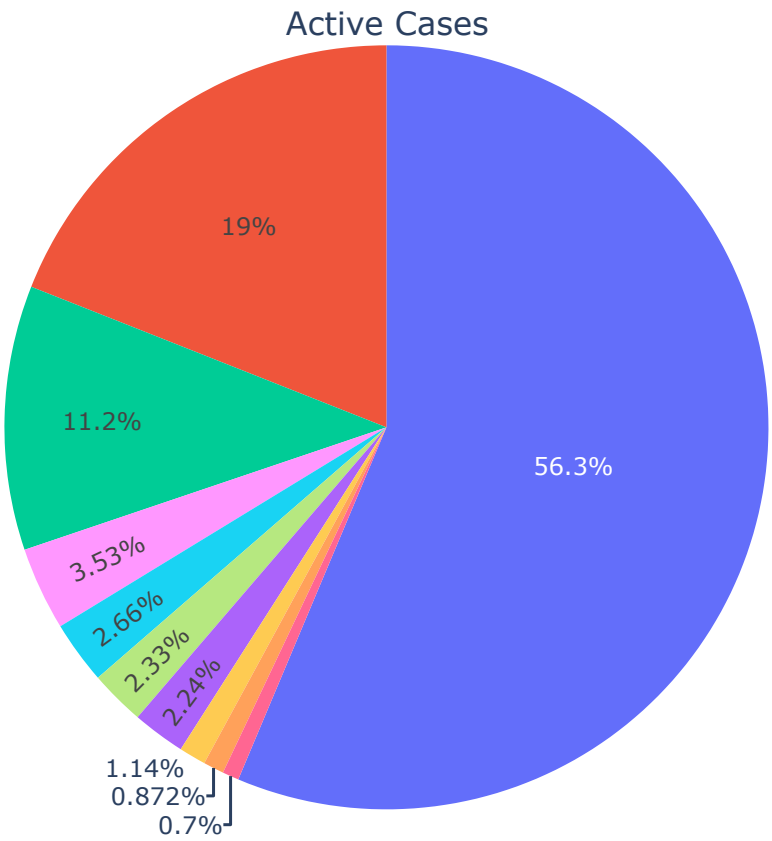


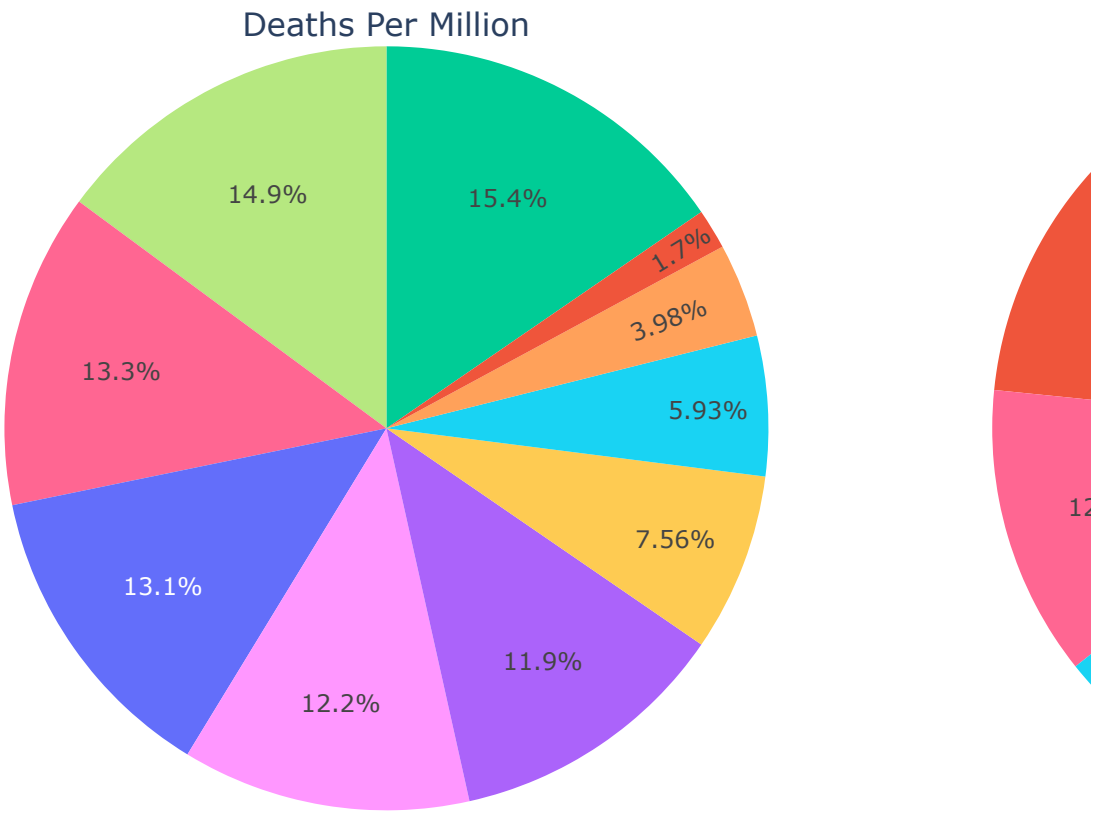

14.9%
5.11%
4.73%
4.57%
4.04%
3.8%
3.41%
3.32%

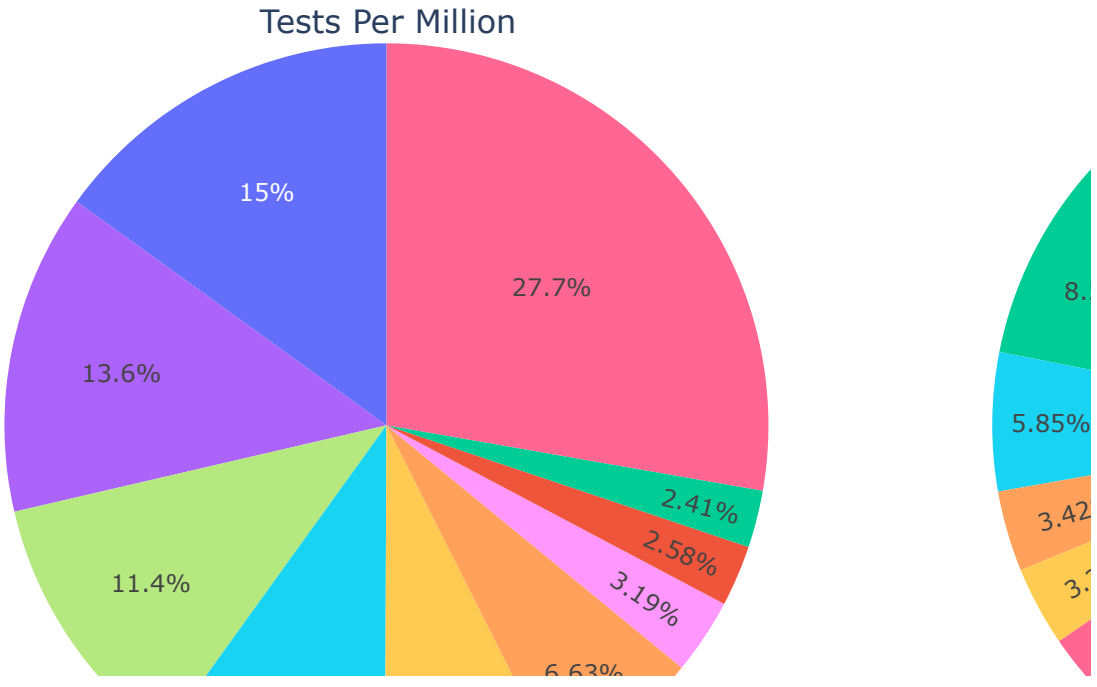

Survival Rate
10.4%
10.6%
8.9%
9.65%
9.85%
9.96%
10%
10.1%
10.2%
10.3%

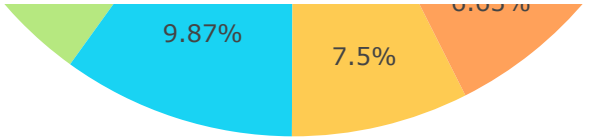

Total Deaths

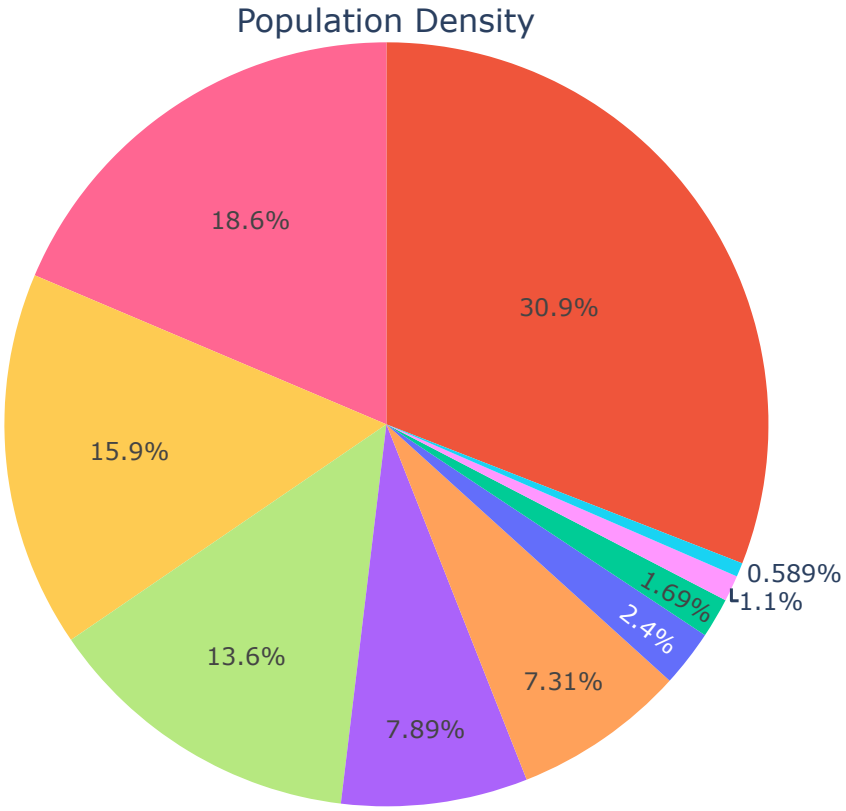

22%
29%
15.8%
2.26%
3.71%
4.24%
6.07%
5.2%
5.99%
5.77%


Active Cases
19%
11.2%
56.3%
3.53%
2.66%
2.33%
2.24%
1.14%
0.872%
0.7%


Deaths Per Million
15.4%
1.7%
3.98%
5.93%
7.56%
11.9%
12.2%
13.1%
13.3%
14.9%


Tests Per Million
27.7%
2.41%
2.58%
3.19%
15%
13.6%
11.4%
5.85%





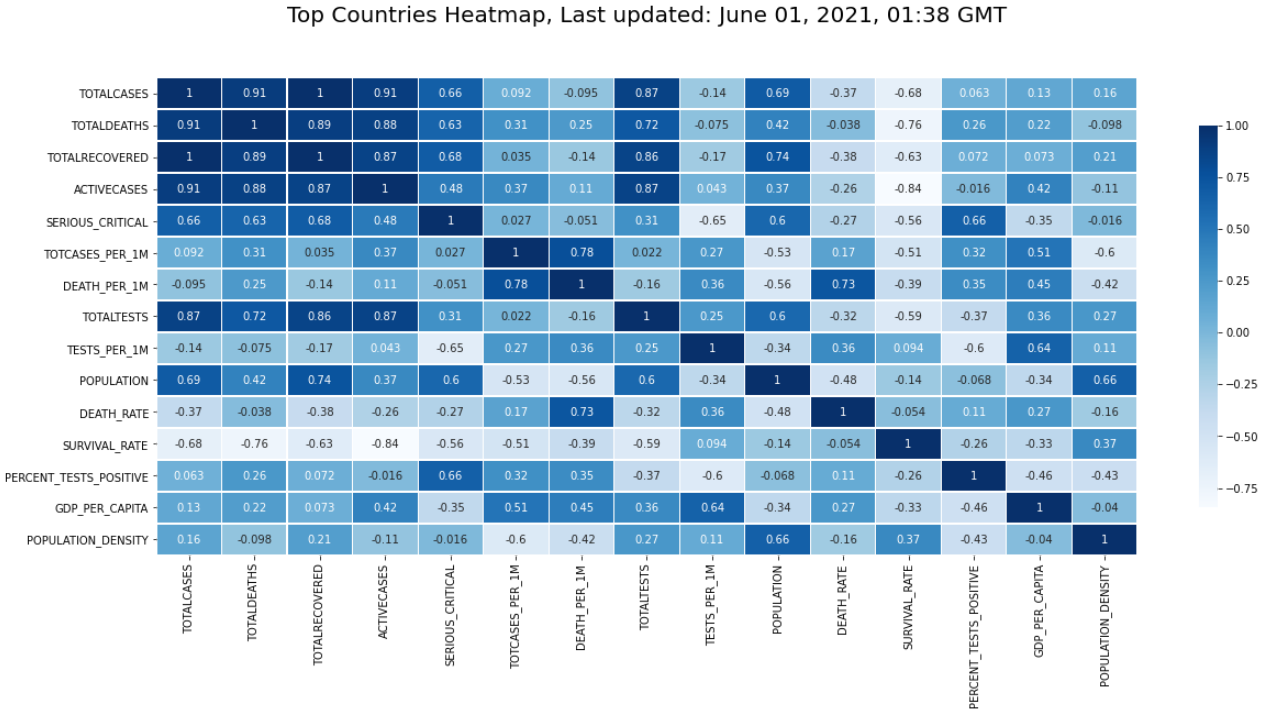
Pearson Correlation Heatmap

In [48]:

```
# Correlation Between Total Tests and Total Cases
plt.figure(figsize=(20,8))

# Numerical variables to be used in Pearson Correlation Heatmap
pc = top_cases[['TOTALCASES', 'TOTALDEATHS','TOTALRECOVERED', 'ACTIVECASES', 'SERIOUS_C
                'TOTCASES_PER_1M', 'DEATH_PER_1M', 'TOTALTESTS', 'TESTS_PER_1M',
                'POPULATION', 'DEATH_RATE', 'SURVIVAL_RATE', 'PERCENT_TESTS_POSITIVE', 'G

# Plot correlation using Seaborn
sns.heatmap(pc,
```

```
            cmap="Blues",
            linewidth=0.3,
            cbar_kws={"shrink": .8},
            annot= True)

# Set title
plt.title(  "Top Countries Heatmap" + ', ' + last_update, size= 20, pad = 50)
```

Out[48]: Text(0.5, 1.0, 'Top Countries Heatmap, Last updated: June 01, 2021, 01:38 GMT')



World Data

In [49]:
```
global_data.head()
```

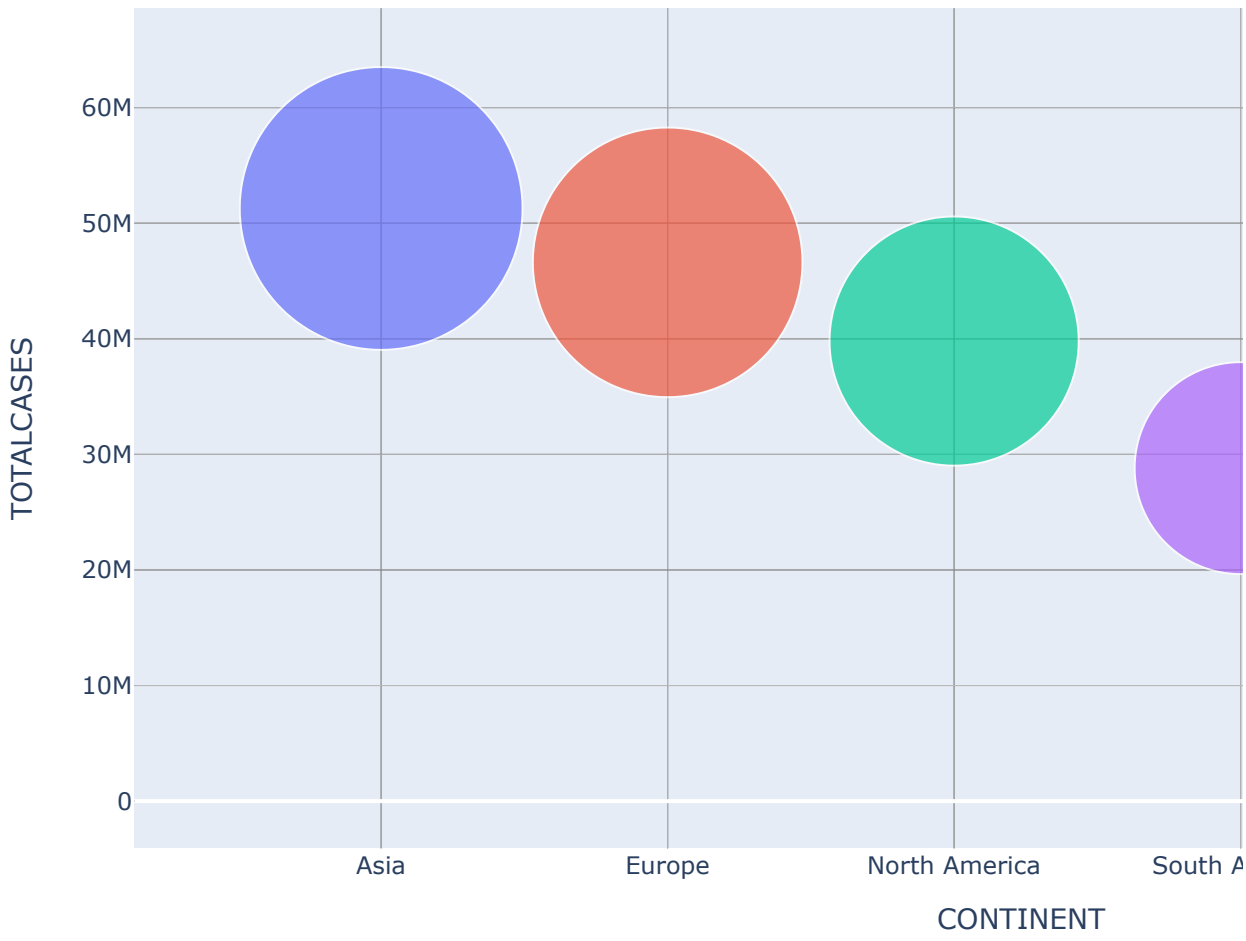
Out[49]:

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCA
0	United States	34113146	609767	27863840	5639539	6129	
1	India	28173655	331909	25939504	1902242	8944	
2	Brazil	16547674	462966	14964631	1120077	8318	
3	France	5667324	109528	5333597	224199	2945	
4	Turkey	5249404	47527	5114624	87253	1339	

Interactive Bubble Plot

The bubble chart in Plotly can be created using the `scatter()` method of plotly.express. A bubble chart is a data visualization which helps to displays multiple circles (bubbles) in a two-dimensional plot as same in scatter plot. A third dimension of the data is shown through the size of markers. A bubble chart is primarily used to depict and show relationships between numeric variables.

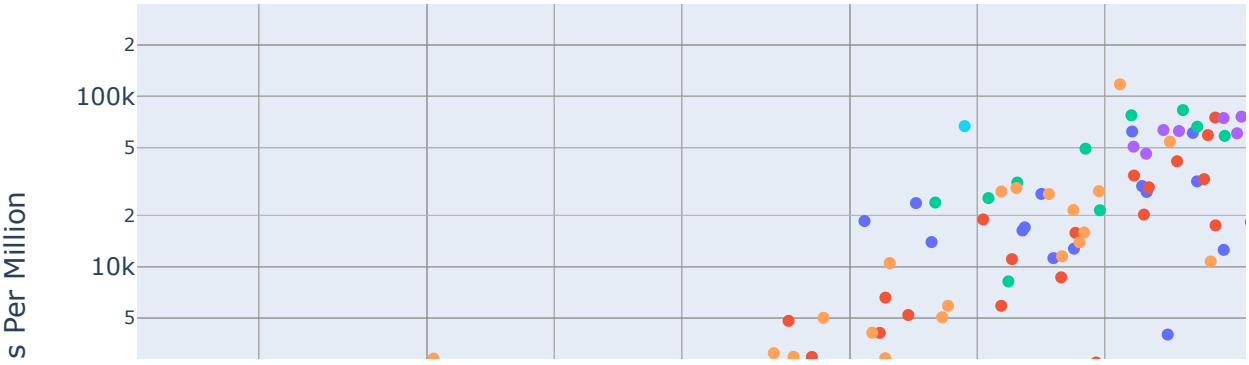
In [50]:
```
# Grouped Dataframe with Sorted Values
global_confirmed = pd.DataFrame(global_data.groupby('CONTINENT')['TOTALCASES'].sum().so

fig1 = px.scatter(global_confirmed,
                  x = global_confirmed.index,
                  y = 'TOTALCASES',
                  size = 'TOTALCASES',
                  size_max = 100,
                  color = global_confirmed.index,
                  title = '<b>Total Confirmed Cases by Continent</b>' + ', ' + last_updat
                  width= 1150,
                  height= 600)

fig1['layout'].update(title_x= .5)
fig1.show()
```

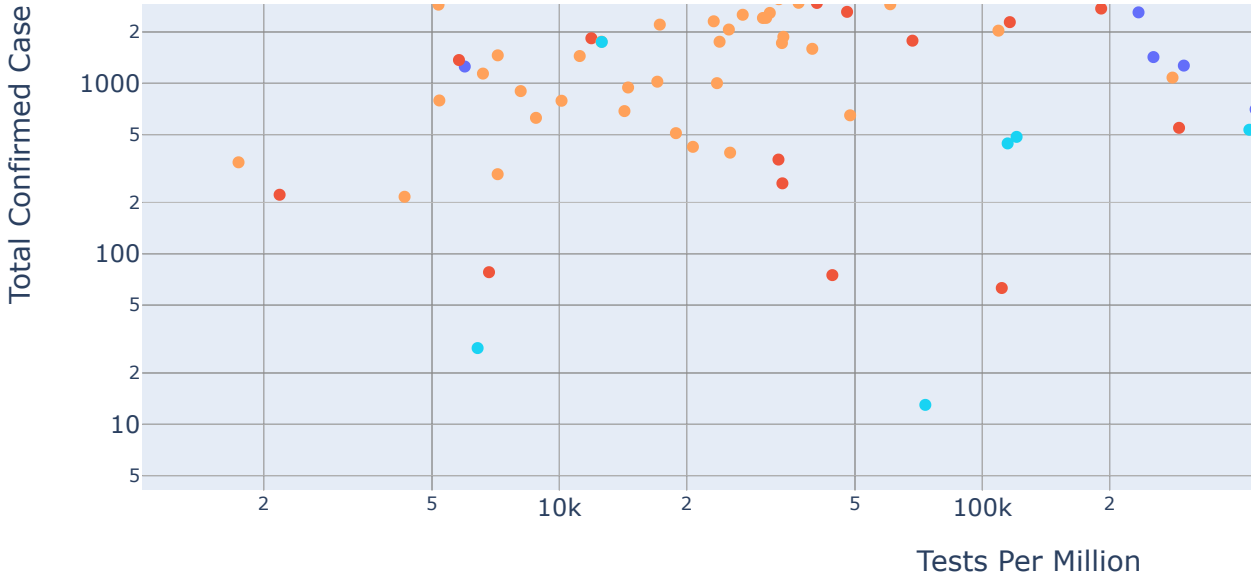


Interactive Scatter Plot

These interactive scatter plot was created using Plotly Express. A scatter plot is a diagram where each value is represented by the dot graph. Scatter plot needs arrays of the same length, one for the value of x-axis and the other for the y-axis. Each data is represented as a dot point, whose location is given by x and y columns. It can be created using the `scatter()` method of `plotly.express`

In [51]:
```
fig = px.scatter(global_data,
                 x="TESTS_PER_1M",
                 y="TOTCASES_PER_1M",
                 color="CONTINENT",
                 trendline= False,
                 log_x = True,
                 log_y= True,
                 title = '<b>Total confirmed COVID-19 cases per million vs Total tests p
                 labels=dict(TESTS_PER_1M="Tests Per Million", TOTCASES_PER_1M="Total Co
                 custom_data=["COUNTRY"])

fig.update_layout(  width=1150,
                    height=600,
                    title_x= .5,
                    showlegend= True,
                    hovermode="x unified")

yaxis={'tickformat':'e', 'rangemode': 'tozero',     # Format Y ticks
           'ticks': 'outside'}
xaxis={'tickformat':'e', 'rangemode': 'tozero',     # Format X ticks
           'ticks': 'outside'}

fig.update_traces(                                  # Update fig scatter plot
        hovertemplate="<br>".join([                 # Assign labels to hover box from v
        "TESTS PER MILLION: %{x}",                  # Customize name of X data
        "TOTAL CASES PER MILLION: %{y}",            # Customuze name of Y data
        "COUNTRY: %{customdata[0]}"                 # Assign to position 0 of custom_da
    ])
)
fig.show()
```

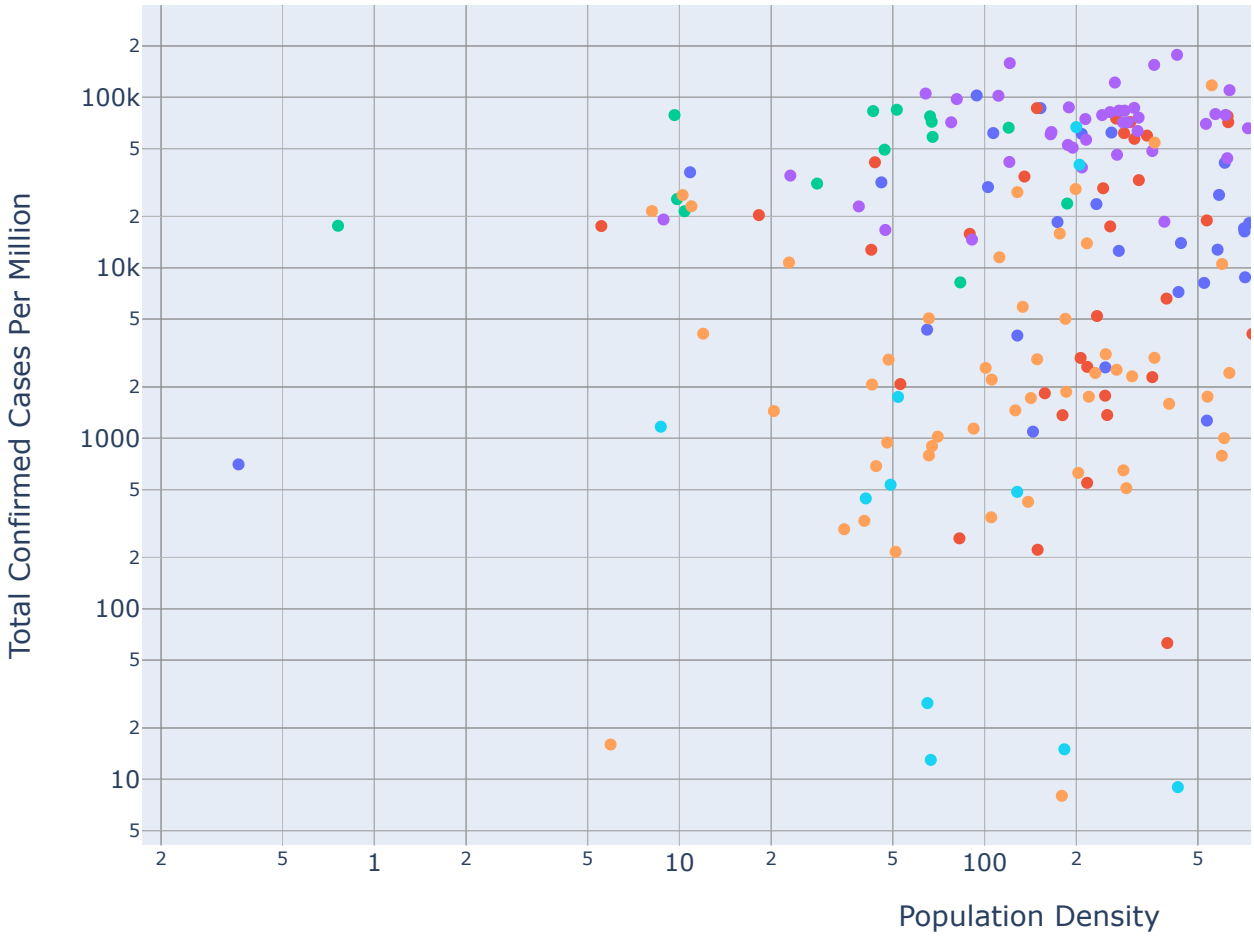




In [52]:

```
fig = px.scatter(global_data,
                 x="POPULATION_DENSITY",
                 y="TOTCASES_PER_1M",
                 color="CONTINENT",
                 trendline= False,
                 log_x = True,
                 log_y= True,
                 title = '<b>Total confirmed COVID-19 cases per million vs Population De
                 labels=dict(POPULATION_DENSITY="Population Density", TOTCASES_PER_1M="T
                 custom_data=["COUNTRY"])

fig.update_layout(  width=1150,
                    height=600,
                    title_x= .5,
                    showlegend= True,
                    hovermode="x unified")

yaxis={'tickformat':'e', 'rangemode': 'tozero',
           'ticks': 'outside'}
xaxis={'tickformat':'e', 'rangemode': 'tozero',
           'ticks': 'outside'}

fig.update_traces(
        hovertemplate="<br>".join([
        "POPULATION DENSITY: %{x}",
        "TOTAL CASES PER MILLION: %{y}",
        "COUNTRY: %{customdata[0]}"
    ])
)

fig.show()
```

Total confirmed COVID-19 cases per million vs Population



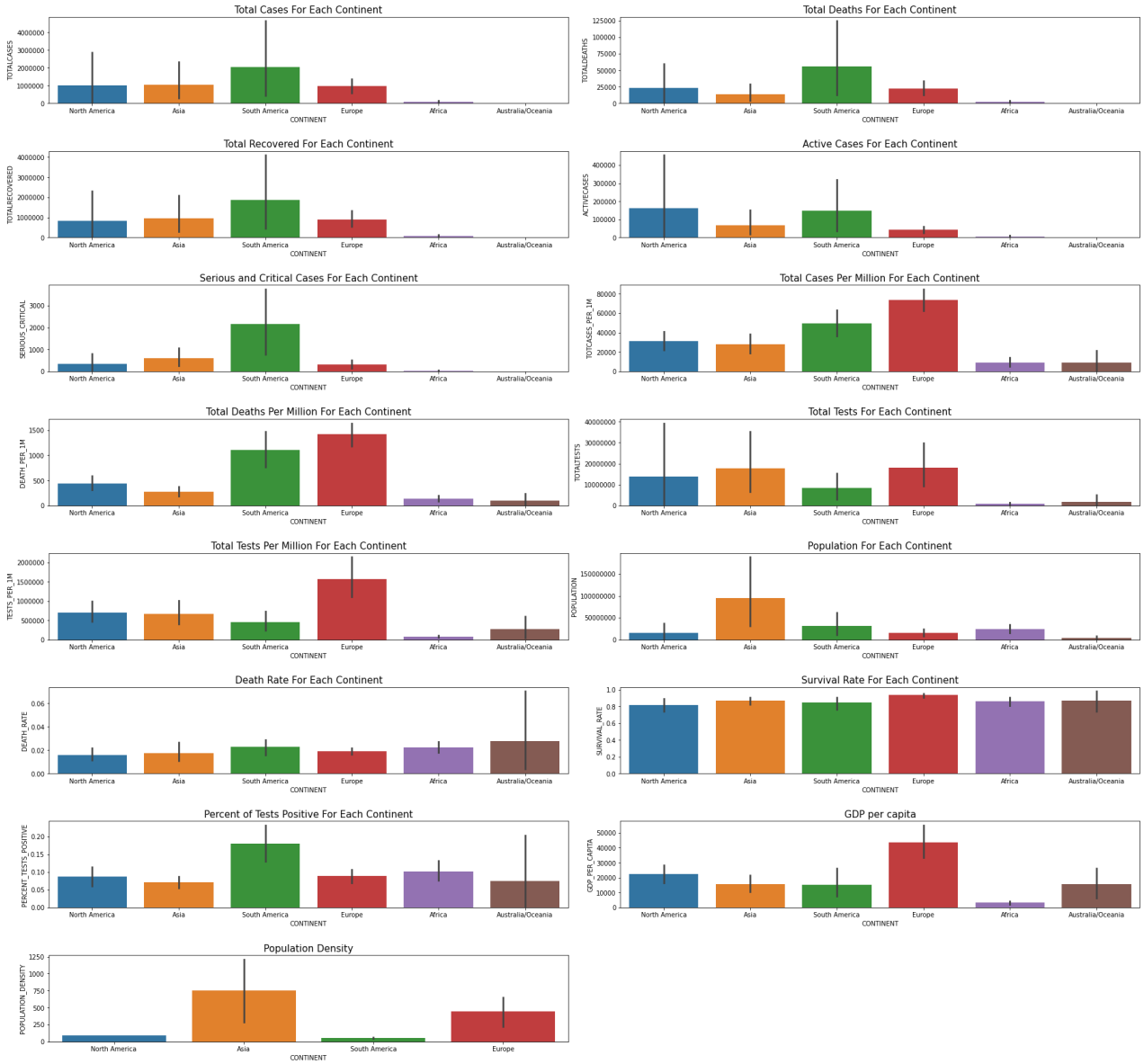
Bar Subplot

In [53]:

```
fig2, axes = plt.subplots(8, 2, figsize= (25,25) )

axes1 = sns.barplot(global_data["CONTINENT"], global_data["TOTALCASES"], data= global_d
axes1.set_title("Total Cases For Each Continent", fontsize=15)

axes2 = sns.barplot(global_data["CONTINENT"], global_data["TOTALDEATHS"], data= global_
axes2.set_title("Total Deaths For Each Continent", fontsize=15)

axes3 = sns.barplot(global_data["CONTINENT"], global_data["TOTALRECOVERED"], data= glob
axes3.set_title("Total Recovered For Each Continent", fontsize=15)

axes4 = sns.barplot(global_data["CONTINENT"], global_data["ACTIVECASES"], data= global_
axes4.set_title("Active Cases For Each Continent", fontsize=15)

axes5 = sns.barplot(global_data["CONTINENT"], global_data["SERIOUS_CRITICAL"], data= gl
axes5.set_title("Serious and Critical Cases For Each Continent", fontsize=15)

axes6 = sns.barplot(global_data["CONTINENT"], global_data["TOTCASES_PER_1M"], data= glo
axes6.set_title("Total Cases Per Million For Each Continent", fontsize=15)

axes7 = sns.barplot(global_data["CONTINENT"], global_data["DEATH_PER_1M"], data= global
axes7.set_title("Total Deaths Per Million For Each Continent", fontsize=15)
```

```
axes8 = sns.barplot(global_data["CONTINENT"], global_data["TOTALTESTS"], data= global_d
axes8.set_title("Total Tests For Each Continent", fontsize=15)

axes9 = sns.barplot(global_data["CONTINENT"], global_data["TESTS_PER_1M"], data= global
axes9.set_title("Total Tests Per Million For Each Continent", fontsize=15)

axes10 = sns.barplot(global_data["CONTINENT"], global_data["POPULATION"], data= global_
axes10.set_title("Population For Each Continent", fontsize=15)

axes11 = sns.barplot(global_data["CONTINENT"], global_data["DEATH_RATE"], data= global_
axes11.set_title("Death Rate For Each Continent", fontsize=15)

axes12 = sns.barplot(global_data["CONTINENT"], global_data["SURVIVAL_RATE"], data= glob
axes12.set_title("Survival Rate For Each Continent", fontsize=15)

axes13 = sns.barplot(global_data["CONTINENT"], global_data["PERCENT_TESTS_POSITIVE"], d
axes13.set_title("Percent of Tests Positive For Each Continent", fontsize=15)

axes14 = sns.barplot(global_data["CONTINENT"], global_data["GDP_PER_CAPITA"], data= glo
axes14.set_title("GDP per capita", fontsize=15)

axes15 = sns.barplot(top_cases["CONTINENT"], top_cases["POPULATION_DENSITY"], data= top
axes15.set_title("Population Density", fontsize=15)

axes16 = sns.barplot(top_cases["CONTINENT"], top_cases["POPULATION_DENSITY"], data= top
axes16.set_title("Population Density", fontsize=15)

fig2.delaxes(ax = axes16)

for axes in axes.flat:
    axes.ticklabel_format(axis='y', style = 'plain')
    axes.set_xticklabels(axes.get_xticklabels())

fig2.suptitle('CONTINENTS BAR PLOTS' + ', ' + last_update, fontsize = 30)
fig2.tight_layout(pad=0.6, w_pad=0.5, h_pad=3)
fig2.subplots_adjust(top=.92)
```

CONTINENTS BAR PLOTS, Last updated: June 01, 2021, 01:38 GMT



In [54]:
```
us_data.columns
```

Out[54]:
```
Index(['STATE', 'TOTALCASES', 'TOTALDEATHS', 'TOTALRECOVERED', 'ACTIVECASES',
       'TOTCASES_PER_1M', 'DEATH_PER_1M', 'TOTALTESTS', 'TESTS_PER_1M',
       'POPULATION', 'DEATH_RATE', 'SURVIVAL_RATE', 'PERCENT_TESTS_POSITIVE'],
      dtype='object')
```

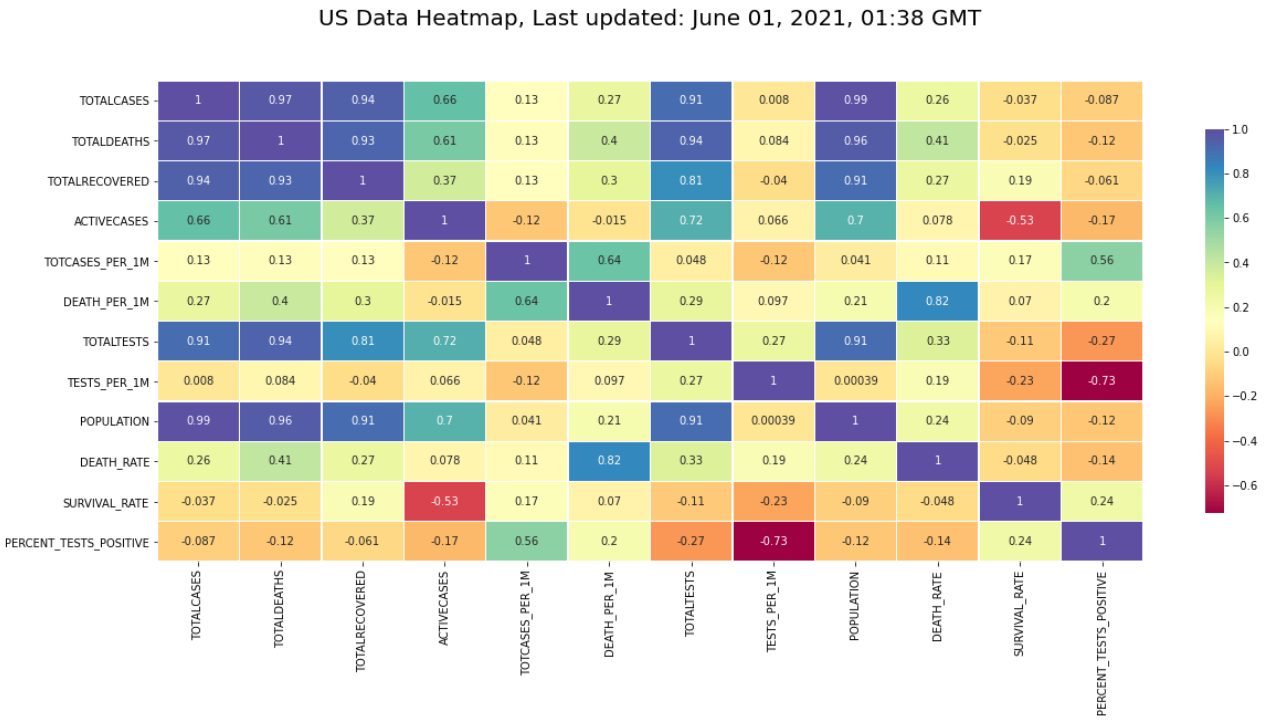
US STATES

Pearson Correlation Heatmap

In [55]:
```
#Correlation Between Total Tests and Total Cases
plt.figure(figsize=(20,8))

pc = us_data[['TOTALCASES', 'TOTALDEATHS','TOTALRECOVERED', 'ACTIVECASES',
       'TOTCASES_PER_1M', 'DEATH_PER_1M', 'TOTALTESTS', 'TESTS_PER_1M',
       'POPULATION', 'DEATH_RATE', 'SURVIVAL_RATE', 'PERCENT_TESTS_POSITIVE']].corr()
```

```
sns.heatmap(  pc,
              cmap="Spectral",              # Color scheme
              linewidth=0.3,                # Spacing between squares
              cbar_kws={"shrink": .8},      # Bar width
              annot= True)                  # Include correlation value

plt.title("US Data Heatmap" + ', ' + last_update, size= 20, pad = 50)
```

Out[55]: Text(0.5, 1.0, 'US Data Heatmap, Last updated: June 01, 2021, 01:38 GMT')

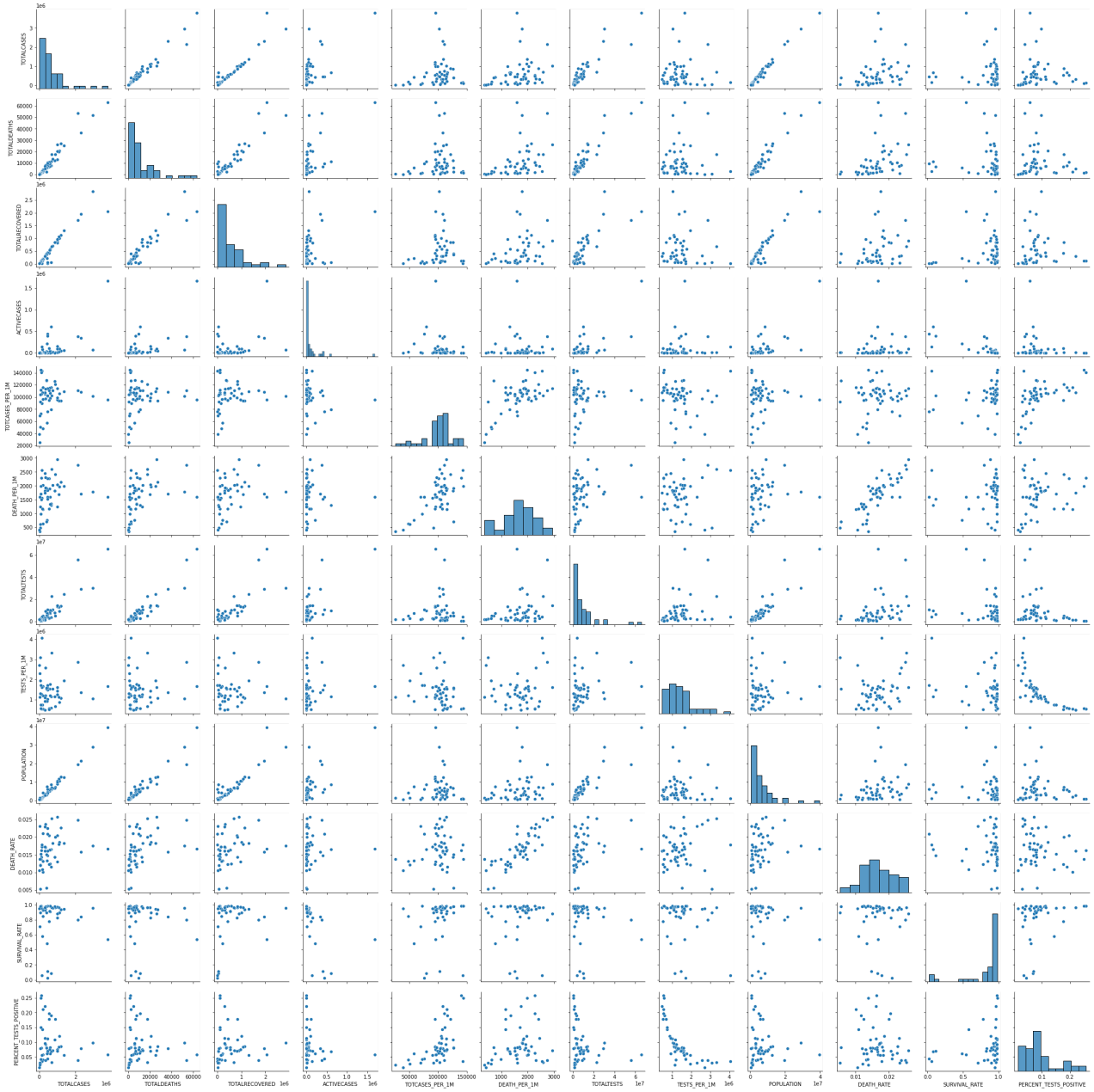


Pair Plot

Pairplot is a module of `Seaborn` library which provides a high-level interface for drawing attractive and informative statistical graphics. A pairplot provides a veiw of bivariate relationships in a dataset. The pairplot function creates a grid of Axes such that each variable in the data will by shared in the y-axis across a single row and in the x-axis across a single column.

In [56]:
```
sns.pairplot(us_data)
```

Out[56]: <seaborn.axisgrid.PairGrid at 0x1dd6db76e20>



CORRELATION ANALYSIS

Pearson Correlation Feature Selection

Pearson Correlation measures the statistical relationship between two continuous variables. It is known as the best method of measuring the association between variables of interest because it is based on the method of covariance. It gives information about the magnitude of the association, or correlation, as well as the direction of the relationship.

</br>

The Degree of Correlation denotes the following:

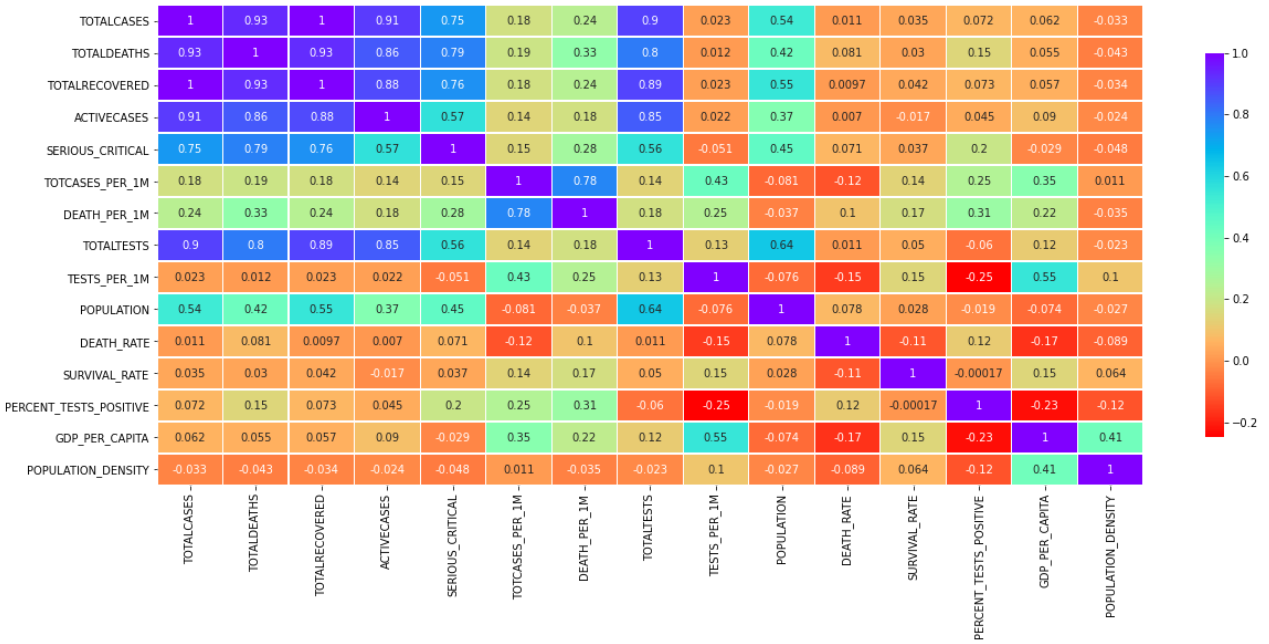
- **Perfect:** If the value is near ± 1, then it said to be a perfect correlation: as one variable increases, the other variable tends to also increase (if positive) or decrease (if negative).
- **High degree:** If the coefficient value lies between ± 0.50 and ± 1, then it is said to be a strong correlation.
- **Moderate degree:** If the value lies between ± 0.30 and ± 0.49, then it is said to be a medium correlation.
- **Low degree:** When the value lies below ± .29, then it is said to be a small correlation.
- **No correlation:** When the value is zero.

</br>

Using Seaborn Library, a heatmap will be constructed to provide a high-level view of the Pearson Correlation coefficients

In [57]:
```
#Pearson correlation heatmap
plt.figure(figsize=(20,8))

pc = global_data[['TOTALCASES', 'TOTALDEATHS','TOTALRECOVERED', 'ACTIVECASES', 'SERIOUS

sns.heatmap(pc,
            cmap="rainbow_r",
            linewidth=0.3,
            cbar_kws={"shrink": .8},
            annot= True)

plt.title("World Data Pearson Correlation Heatmap",
            size= 20,
            pad = 50)
```

Out[57]:
```
Text(0.5, 1.0, 'World Data Pearson Correlation Heatmap')
```



The first question that was posed in the analysis aimed to determine whether the spread of Covid-19 is influenced by population density. A higher population density could imply less social distancing. The heatmap shows that the dependent variable, `POPULATION_DENSITY` has a very low correlation with all other variables except for `GDP_PER_CAPITA`. Due to the low correlation, we can infer that population density is not a factor influencing Covid-19 cases.

The second question postulates that `GDP_PER_CAPITA` has an influence on `DEATH_RATE`, with a higher GDP indicating that a country has a better response in dealing with Covid-19 cases. `GDP_PER_CAPITA` has a medium correlation with `TOTCASES_PER_1M`, `DEATH_PER_MILLION`, and `TESTS_PER_MILLION`. A Statsmodels summary evaluation will be performed to investigate this correlation further.

Linear Regression in Statsmodels

Linear regression is a statistical method for modelling relationship between a dependent variable with a given set of independent variables. Simple linear regression is an approach for predicting a response using a single feature. It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

In [58]:
```
global_data.describe()
```

Out[58]:

	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCASES_PE
count	2.200000e+02	220.000000	2.200000e+02	2.200000e+02	220.000000	220.0
mean	7.793409e+05	16203.404545	6.995552e+05	6.358236e+04	417.068182	34032.8
std	3.267779e+06	60942.046237	2.845664e+06	4.087259e+05	1286.190758	38361.4

	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCASES_PE
min	1.000000e+00	0.000000	1.000000e+00	0.000000e+00	0.000000	8.0
25%	6.437750e+03	90.000000	4.133750e+03	1.307500e+02	0.000000	2059.2
50%	4.792450e+04	800.000000	4.188200e+04	3.114500e+03	10.000000	17963.5
75%	3.443408e+05	6159.250000	3.014488e+05	1.894975e+04	136.750000	61776.2
max	3.411315e+07	609767.000000	2.786384e+07	5.639539e+06	8944.000000	177402.0

The `DataFrame.describe()` method is used to view some basic statistical details like count, mean, std, min, max, and percentile (25th, 50th, and 75th percentiles) of a data frame or a series of numeric values.

Statsmodels.summary()

This linear regression analysis will be performed using the Ordinary Least Squares function from the `statsmodels` library. It compares the difference between individual points in the data set and the predicted best fit line to measure the amount of error produced. Least squares is a standard approach in regression analysis to approximate the solution by minimising the sum of the squares of the residuals.The `smf.ols()` function requires two inputs, the formula for producing the best fit line, and the dataset.

Summary report will display the following:

- **R-Squared**: Percent of variance explained by the model.
- **Adj. R-Squared**: R-Squared where additional independent variables are penalized
- **F-statistic**: Significance of fit
- **Prob (F-statistic)**: Probability of seeing F-statistic from a sample
- **Log-likelihood**: Log of the likelihood function
- **AIC**: Akaike Information Criterion, penalizes model when more independent variables are added.
- **BIC**: Bayesian Information Criterion, similar to AIC but with higher penalties
- **coef**: Estimated coefficient value
- **std err**: Standard error of the coefficient estimate
- **t**: Measure of statistical significance for coefficient
- **P>|t|**: Probability value that the coefficient is equal to 0
- **[0.025 0.975]**: Lower and upper halves of 95% confidence interval

- **Omnibus**: Omnibus D'Angostino's test, statistical test for skewness and kurtosis
- **Prob(Omnibus)**: Omnibus statistic as a probability
- **Skew**: Measure of data mean symmetry
- **Kurtosis**: Measure of shape of the distribution
- **Durbin-Watson**: Test for autocorrelation
- **Jarque-Bera (JB)**: Test for skewness & kurtosis
- **Prob (JB)**: Jarque-Bera statistic as a probability
- **Cond. No.**: Test for multicollinearity

Total Cases per Million vs GDP

Total Cases Per Million Without Constant

In [59]:
```
X = global_data[['GDP_PER_CAPITA']]              # Independent vars
y = global_data['TOTCASES_PER_1M']               # Dependant variable

model = sm.OLS(y,X, missing = 'drop').fit()      # Line of best fit
predictions = model.predict(X)                   # Prediction

model.summary()
```

Out[59]:

OLS Regression Results

Dep. Variable:	TOTCASES_PER_1M	**R-squared (uncentered):**	0.366
Model:	OLS	**Adj. R-squared (uncentered):**	0.363
Method:	Least Squares	**F-statistic:**	126.5
Date:	Mon, 31 May 2021	**Prob (F-statistic):**	1.85e-23
Time:	21:59:21	**Log-Likelihood:**	-2647.7
No. Observations:	220	**AIC:**	5297.
Df Residuals:	219	**BIC:**	5301.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
GDP_PER_CAPITA	0.9380	0.083	11.246	0.000	0.774	1.102

Omnibus:	18.097	**Durbin-Watson:**	1.349
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	25.482
Skew:	0.536	**Prob(JB):**	2.93e-06

Kurtosis:	4.278	**Cond. No.**	1.00

Notes:

[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.

[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [ ]:

Total Cases Per Million With Constant

In [60]:

```
X = global_data['GDP_PER_CAPITA']
y = global_data['TOTCASES_PER_1M']
X = sm.add_constant(X) # intercept (beta_0) added to model

model = sm.OLS(y,X, missing='drop').fit()
predictions = model.predict(X)

model.summary()
```

Out[60]:

OLS Regression Results

Dep. Variable:	TOTCASES_PER_1M	**R-squared:**	0.124
Model:	OLS	**Adj. R-squared:**	0.120
Method:	Least Squares	**F-statistic:**	30.88
Date:	Mon, 31 May 2021	**Prob (F-statistic):**	7.95e-08
Time:	21:59:21	**Log-Likelihood:**	-2619.1
No. Observations:	220	**AIC:**	5242.
Df Residuals:	218	**BIC:**	5249.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	2.416e+04	3007.449	8.032	0.000	1.82e+04	3.01e+04
GDP_PER_CAPITA	0.5059	0.091	5.557	0.000	0.326	0.685

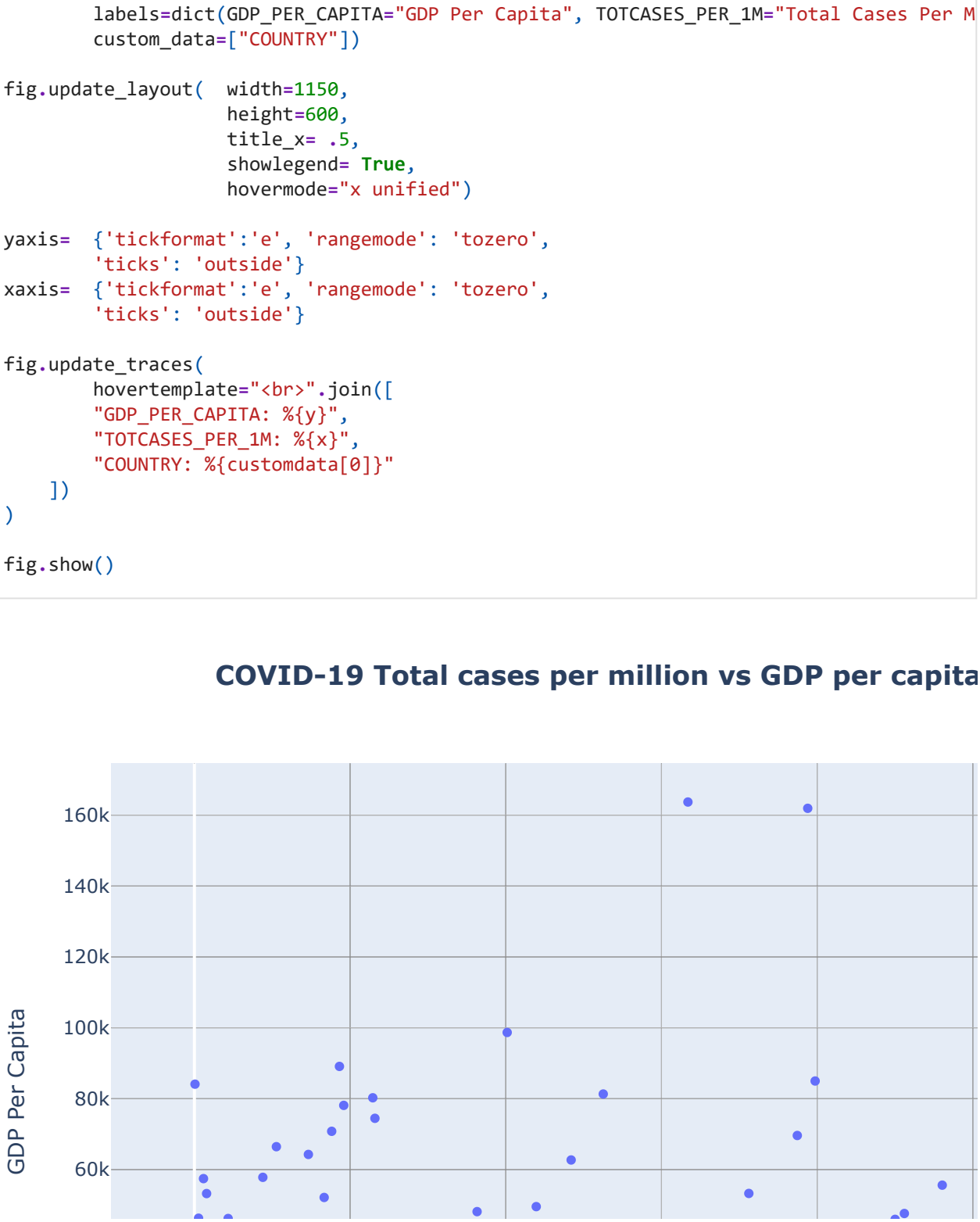
Omnibus:	42.521	**Durbin-Watson:**	1.633
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	61.947
Skew:	1.149	**Prob(JB):**	3.53e-14
Kurtosis:	4.214	**Cond. No.**	4.10e+04

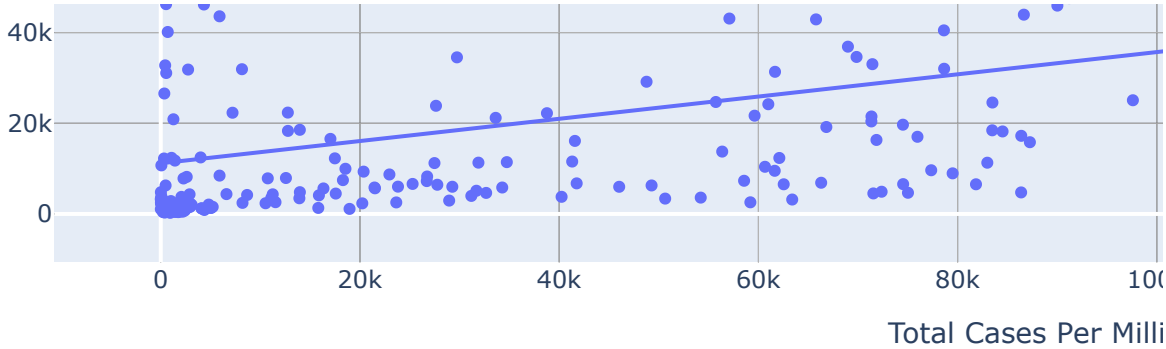
Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.1e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [61]:

```
fig =   px.scatter(global_data,
        y="GDP_PER_CAPITA",
        x="TOTCASES_PER_1M",
        trendline= "ols",
        log_x = False,
        log_y= False,
        title = '<b>COVID-19 Total cases per million vs GDP per capita</b>' + ', ' + la
        labels=dict(GDP_PER_CAPITA="GDP Per Capita", TOTCASES_PER_1M="Total Cases Per M
        custom_data=["COUNTRY"])

fig.update_layout(  width=1150,
                    height=600,
                    title_x= .5,
                    showlegend= True,
                    hovermode="x unified")

yaxis=  {'tickformat':'e', 'rangemode': 'tozero',
        'ticks': 'outside'}
xaxis=  {'tickformat':'e', 'rangemode': 'tozero',
        'ticks': 'outside'}

fig.update_traces(
        hovertemplate="<br>".join([
        "GDP_PER_CAPITA: %{y}",
        "TOTCASES_PER_1M: %{x}",
        "COUNTRY: %{customdata[0]}"
    ])
)

fig.show()
```





Deaths Per Million vs GDP

Deaths Per Million Without Constant

In [62]:
```
X = global_data[['GDP_PER_CAPITA']]
y = global_data['DEATH_PER_1M']

model = sm.OLS(y,X, missing='drop').fit()
predictions = model.predict(X)

model.summary()
```

Out[62]:

OLS Regression Results

Dep. Variable:	DEATH_PER_1M	**R-squared (uncentered):**	0.254
Model:	OLS	**Adj. R-squared (uncentered):**	0.251
Method:	Least Squares	**F-statistic:**	74.66
Date:	Mon, 31 May 2021	**Prob (F-statistic):**	1.18e-15
Time:	21:59:22	**Log-Likelihood:**	-1779.3
No. Observations:	220	**AIC:**	3561.
Df Residuals:	219	**BIC:**	3564.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
GDP_PER_CAPITA	0.0139	0.002	8.641	0.000	0.011	0.017

Omnibus:	41.336	**Durbin-Watson:**	1.233
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	62.692
Skew:	1.072	**Prob(JB):**	2.44e-14
Kurtosis:	4.497	**Cond. No.**	1.00

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Deaths Per Million With Constant

In [63]:
```
X = global_data[['GDP_PER_CAPITA']]
y = global_data['DEATH_PER_1M']
X = sm.add_constant(X)

model = sm.OLS(y,X, missing='drop').fit()
predictions = model.predict(X)

model.summary()
```

Out[63]:

OLS Regression Results

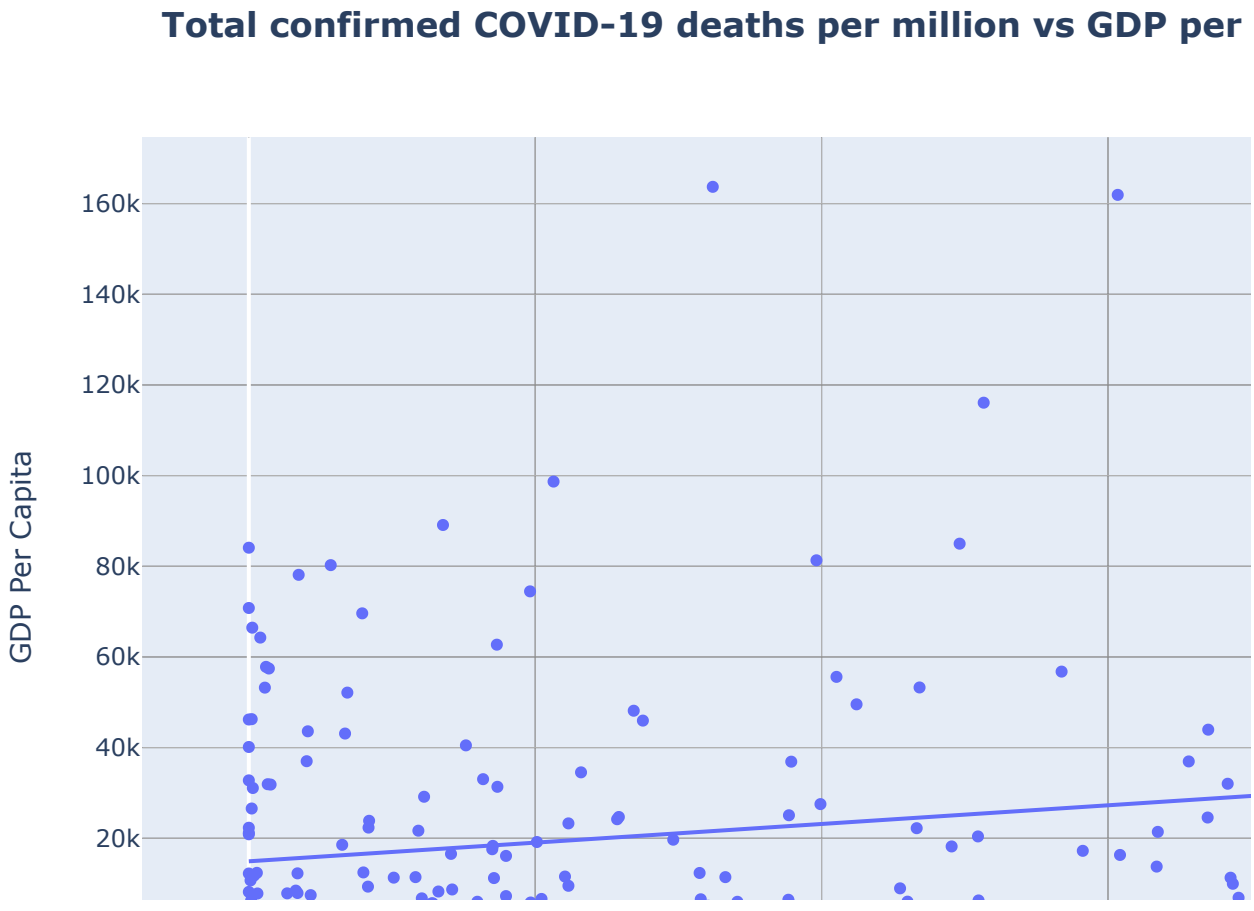
Dep. Variable:	DEATH_PER_1M	**R-squared:**	0.050
Model:	OLS	**Adj. R-squared:**	0.045
Method:	Least Squares	**F-statistic:**	11.38
Date:	Mon, 31 May 2021	**Prob (F-statistic):**	0.000879
Time:	21:59:22	**Log-Likelihood:**	-1754.2
No. Observations:	220	**AIC:**	3512.
Df Residuals:	218	**BIC:**	3519.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	441.5011	58.980	7.486	0.000	325.258	557.745
GDP_PER_CAPITA	0.0060	0.002	3.373	0.001	0.003	0.010

Omnibus:	62.038	**Durbin-Watson:**	1.484
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	109.408
Skew:	1.507	**Prob(JB):**	1.75e-24
Kurtosis:	4.689	**Cond. No.**	4.10e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.1e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [64]:
```
fig =   px.scatter(global_data,
        y="GDP_PER_CAPITA",
        x="DEATH_PER_1M",
        trendline= "ols",
        log_x = False,
        log_y= False,
        title = '<b>Total confirmed COVID-19 deaths per million vs GDP per capita</b>'
        labels=dict(GDP_PER_CAPITA="GDP Per Capita", DEATH_PER_1M="Deaths Per Million")
        custom_data=["COUNTRY", "CONTINENT"])

fig.update_layout(  width=1150,
                    height=600,
                    title_x= .5,
                    showlegend= True,
                    hovermode="x unified")

yaxis={'tickformat':'e', 'rangemode': 'tozero',
           'ticks': 'outside'}
xaxis={'tickformat':'e', 'rangemode': 'tozero',
           'ticks': 'outside'}

fig.update_traces(
        hovertemplate="<br>".join([
        "GDP_PER_CAPITA: %{y}",
        "DEATH_PER_1M: %{x}",
        "COUNTRY: %{customdata[0]}"
    ])
)

fig.show()
```



Tests Per Million vs GDP Per Capita

Tests Per Million Without Constant

In [65]:
```
X = global_data[['GDP_PER_CAPITA']] #independent vars
y = global_data['TESTS_PER_1M'] #dependant variable

model = sm.OLS(y,X, missing='drop').fit()
predictions = model.predict(X)

model.summary()
```

Out[65]:

OLS Regression Results

Dep. Variable:	TESTS_PER_1M	**R-squared (uncentered):**	0.458
Model:	OLS	**Adj. R-squared (uncentered):**	0.456
Method:	Least Squares	**F-statistic:**	185.3
Date:	Mon, 31 May 2021	**Prob (F-statistic):**	5.48e-31
Time:	21:59:22	**Log-Likelihood:**	-3357.5
No. Observations:	220	**AIC:**	6717.
Df Residuals:	219	**BIC:**	6720.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
GDP_PER_CAPITA	28.5989	2.101	13.614	0.000	24.459	32.739

Omnibus:	179.406	**Durbin-Watson:**	1.925
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3811.760
Skew:	2.936	**Prob(JB):**	0.00
Kurtosis:	22.528	**Cond. No.**	1.00

Notes:

[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.

[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Tests Per Million With Constant

In [66]:
```
X = global_data[['GDP_PER_CAPITA']] #independent vars
y = global_data['TESTS_PER_1M'] #dependant variable
X = sm.add_constant(X) # intercept (beta_0) added to model

model = sm.OLS(y,X, missing='drop').fit()
predictions = model.predict(X)

model.summary()
```

Out[66]:

OLS Regression Results

Dep. Variable:	TESTS_PER_1M	**R-squared:**	0.305
Model:	OLS	**Adj. R-squared:**	0.302
Method:	Least Squares	**F-statistic:**	95.71
Date:	Mon, 31 May 2021	**Prob (F-statistic):**	5.69e-19
Time:	21:59:22	**Log-Likelihood:**	-3355.1
No. Observations:	220	**AIC:**	6714.
Df Residuals:	218	**BIC:**	6721.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	1.867e+05	8.53e+04	2.189	0.030	1.86e+04	3.55e+05
GDP_PER_CAPITA	25.2595	2.582	9.783	0.000	20.171	30.348

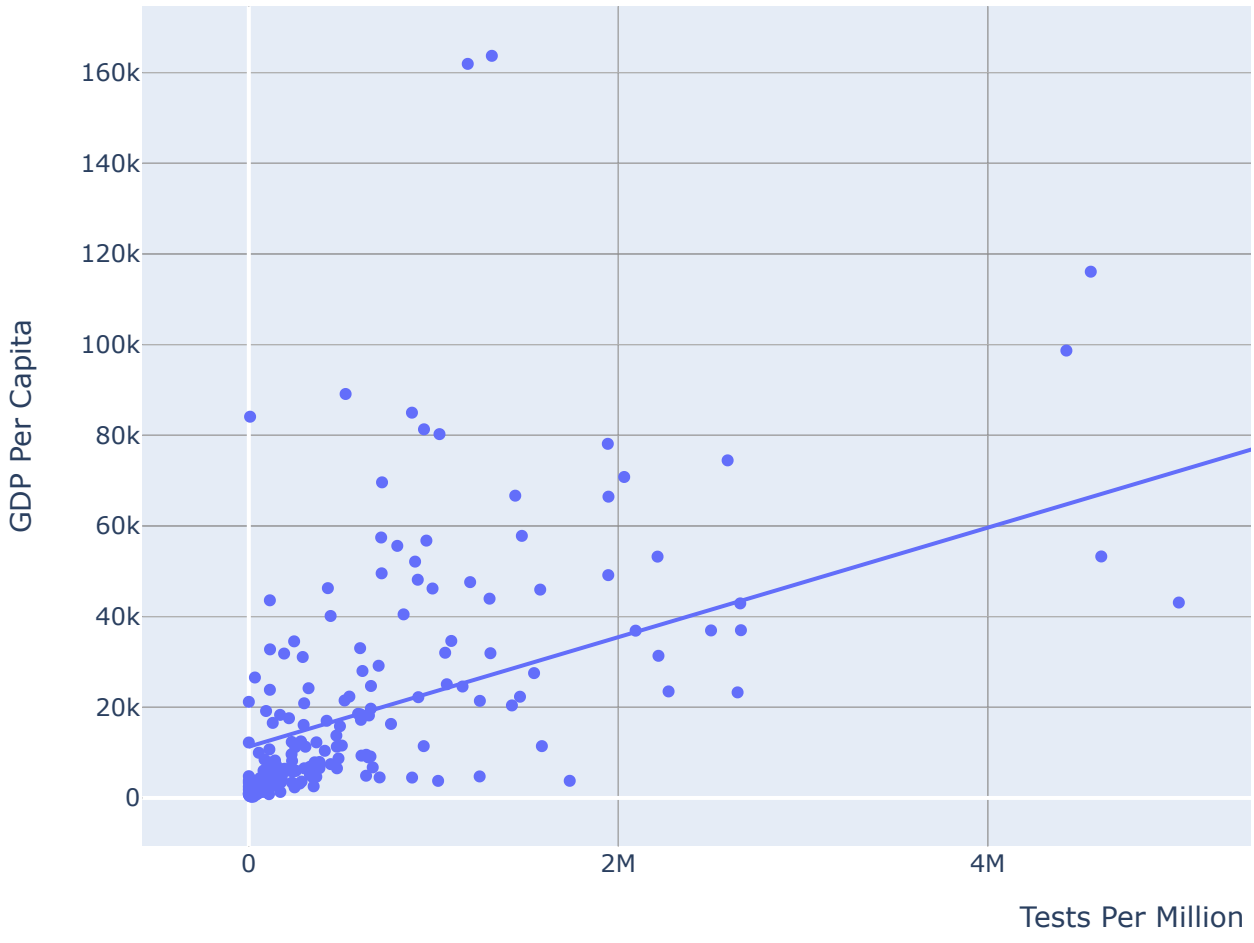
Omnibus:	199.156	**Durbin-Watson:**	1.961
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	4512.469
Skew:	3.431	**Prob(JB):**	0.00
Kurtosis:	24.099	**Cond. No.**	4.10e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.1e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [67]:
```
fig = px.scatter(global_data,
                 y="GDP_PER_CAPITA",
                 x="TESTS_PER_1M",
                 trendline= "ols",
                 log_x = False,
                 log_y= False,
                 title = '<b>Total COVID-19 Tests administed per million vs GDP per capi
```

```
                labels=dict(GDP_PER_CAPITA="GDP Per Capita", TESTS_PER_1M="Tests Per Mi
                custom_data=["COUNTRY", "CONTINENT"])

fig.update_layout(  width=1150,
                    height=600,
                    title_x= .5,
                    showlegend= True,
                    hovermode="x unified")

yaxis={'tickformat':'e', 'rangemode': 'tozero',
           'ticks': 'outside'}
xaxis={'tickformat':'e', 'rangemode': 'tozero',
           'ticks': 'outside'}

fig.update_traces(
        hovertemplate="<br>".join([
        "GDP_PER_CAPITA: %{y}",
        "TESTS_PER_1M: %{x}",
        "COUNTRY: %{customdata[0]}"
    ])
)
fig.show()
```

Total COVID-19 Tests administed per million vs GDP per



GEOMAPPING

Spatial data, geospatial data, GIS data or geodata, are names for numeric data that identifies the geographical location of a physical object such as a building, a street, a town, a city, a country, etc. according to a geographic coordinate system. From the spatial data, you can find out not only the location but also the length, size, area or shape of any object. An example of a kind of spatial data that you can get are: coordinates of an object such as latitude, longitude, and elevation. Geographic Information Systems (GIS) or other specialized software applications can be used to access, visualize, manipulate and analyze geospatial data.

Gather Geospatial Data

The geospatial data used will consist of shapefiles. A shapefile is a simple, nontopological format for storing the geometric location and attribute information of geographic features. Geographic features in a shapefile can be represented by points, lines, or polygons. Latitude and longitute coordinates will be added to the dataframe using `Nominatim` to gather data from OpenStreetMap

Geodata will be gathered from two different shapefiles. Each one contains country data that is missing in the other. The `df.update` function will be used to modify in place using non-NA values from a dataframe created from `global_data` merged with the dataframe containing the shapefiles.

World Geospatial Data

In [68]:
```
global_data.head()
```

Out[68]:

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCA
0	United States	34113146	609767	27863840	5639539	6129	
1	India	28173655	331909	25939504	1902242	8944	
2	Brazil	16547674	462966	14964631	1120077	8318	
3	France	5667324	109528	5333597	224199	2945	
4	Turkey	5249404	47527	5114624	87253	1339	

Get Lat and Long Coordinates

In [69]:
```
# Use Nominatim to get Longitude and Latitude
locator = Nominatim(user_agent="myGeocoder")

loc1_df=[]                                                    # Create empty
for i in global_data.COUNTRY:                                 # Iterate throu
        location = locator.geocode([i])                       # Find location
        loc1_df.append([location.latitude, location.longitude])  # Append locati
```

In [70]:
```
global_loc = pd.DataFrame(loc1_df, columns = ['LAT', 'LONG'])  # Create datafr
global_loc.head()
```

Out[70]:

	LAT	LONG
0	39.783730	-100.445882
1	22.351115	78.667743
2	-10.333333	-53.200000
3	46.603354	1.888334
4	38.959759	34.924965

In [71]:
```
global_data = global_data.join(global_loc)      # Join 'global_loc' dataframe with 'glo
global_data.head()
```

Out[71]:

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCA
0	United States	34113146	609767	27863840	5639539	6129	
1	India	28173655	331909	25939504	1902242	8944	
2	Brazil	16547674	462966	14964631	1120077	8318	
3	France	5667324	109528	5333597	224199	2945	
4	Turkey	5249404	47527	5114624	87253	1339	

Create Shapefile DataFrames

In [72]:
```
# Copy relative path of .shp file
shapefile = 'Resources\\UIA_World_Countries_Boundaries-shp\World_Countries__Generalized_

# Read shapefile using Geopandas
gdf = gpd.read_file(shapefile)

# Create dataframe to be merged later
gdf1=gdf[['COUNTRY','geometry']].copy()
gdf1.head()
```

Out[72]:

	COUNTRY	geometry
0	American Samoa	POLYGON ((-170.74390 -14.37555, -170.74942 -14...

	COUNTRY	geometry
1	United States Minor Outlying Islands	MULTIPOLYGON (((-160.02114 -0.39805, -160.0281...
2	Cook Islands	MULTIPOLYGON (((-159.74698 -21.25667, -159.793...
3	French Polynesia	MULTIPOLYGON (((-149.17920 -17.87084, -149.258...
4	Niue	POLYGON ((-169.89389 -19.14556, -169.93088 -19...

In [73]:
```
# Copy relative path of .shp file
shapefile = 'Resources\Longitude_Graticules_and_World_Countries_Boundaries-shp\99bfd9e7

# Read shapefile using Geopandas
gdf2 = gpd.read_file(shapefile)
gdf2= gdf2.rename(columns={'CNTRY_NAME': 'COUNTRY'})

# Create dataframe to be merged later
gdf3 = gdf2[['COUNTRY','geometry']].copy()
gdf3.head()
```

Out[73]:

	COUNTRY	geometry
0	Aruba	POLYGON ((-69.88223 12.41111, -69.94695 12.436...
1	Antigua and Barbuda	MULTIPOLYGON (((-61.73889 17.54055, -61.75195 ...
2	Afghanistan	POLYGON ((61.27656 35.60725, 61.29638 35.62853...
3	Algeria	POLYGON ((-5.15213 30.18047, -5.13917 30.19236...
4	Azerbaijan	MULTIPOLYGON (((45.02583 41.03055, 45.00999 41...

In [74]:
```
# The Country names in the 'gdf1' dataframe are inconsistent with those in 'global_data
gdf1.COUNTRY= gdf1.COUNTRY.replace("Russian Federation","Russia")
gdf1.COUNTRY= gdf1.COUNTRY.replace("Palestinian Territory","Palestine")
gdf1.COUNTRY= gdf1.COUNTRY.replace("Czech Republic","Czechia")
gdf1.COUNTRY= gdf1.COUNTRY.replace("United Arab Emirates","UAE")
gdf1.COUNTRY= gdf1.COUNTRY.replace("South Korea","S. Korea")
gdf1.COUNTRY= gdf1.COUNTRY.replace("Côte d'Ivoire","Ivory Coast")
gdf1.COUNTRY= gdf1.COUNTRY.replace("Congo DRC","DRC")
gdf1.COUNTRY= gdf1.COUNTRY.replace("Central African Republic","CAR")
gdf1.COUNTRY= gdf1.COUNTRY.replace("Turks and Caicos Islands","Turks and Caicos")
gdf1.COUNTRY= gdf1.COUNTRY.replace("Saint Vincent and the Grenadines","St. Vincent Gren
gdf1.COUNTRY= gdf1.COUNTRY.replace("Saint Barthelemy","St. Barth")
gdf1.COUNTRY= gdf1.COUNTRY.replace("Brunei Darussalam","Brunei")
gdf1.COUNTRY= gdf1.COUNTRY.replace("Saint Pierre and Miquelon","Saint Pierre Miquelon")
gdf1.COUNTRY= gdf1.COUNTRY.replace("Curacao","Curaçao")
gdf1.COUNTRY= gdf1.COUNTRY.replace("Faroe Islands","Faeroe Islands")
```

Merge DataFrames

In [75]:
```
# Set Country to String
gdf1['COUNTRY'] =pd.Series(gdf1['COUNTRY'], dtype= "string")
gdf2['COUNTRY'] =pd.Series(gdf2['COUNTRY'], dtype= "string")

# Strip Country columns
```

```
gdf1.COUNTRY = gdf1.COUNTRY.str.strip()
gdf3.COUNTRY = gdf3.COUNTRY.str.strip()
global_data.COUNTRY =global_data.COUNTRY.str.strip()

# Create new
geo_global_data = global_data.merge(gdf1, on='COUNTRY', how='left')
geo_global_data['COUNTRY'] =pd.Series(geo_global_data['COUNTRY'], dtype= "string")
geo_global_data2 = global_data.merge(gdf3, on='COUNTRY', how= 'left')


geo_global_data.update(geo_global_data2)

# Check datatypes
geo_global_data.dtypes
```

Out[75]:
```
COUNTRY                  object
TOTALCASES                int64
TOTALDEATHS               int64
TOTALRECOVERED            int64
ACTIVECASES               int64
SERIOUS_CRITICAL          int64
TOTCASES_PER_1M           int64
DEATH_PER_1M              int64
TOTALTESTS                int64
TESTS_PER_1M              int64
POPULATION                int64
CONTINENT                object
DEATH_RATE              float64
SURVIVAL_RATE           float64
PERCENT_TESTS_POSITIVE  float64
GDP_PER_CAPITA          float64
LAND_AREA                 int64
POPULATION_DENSITY      float64
LAT                     float64
LONG                    float64
geometry                 object
dtype: object
```

Convert to GeoDataFrame

In [76]:
```
# Convert 'geo_global_data' to GeoDataFrame
geo_global_data = GeoDataFrame(geo_global_data)
type(geo_global_data)
```

Out[76]:
```
geopandas.geodataframe.GeoDataFrame
```

Null values

In [77]:
```
geo_global_data[geo_global_data.isna().any(axis=1)]
```

Out[77]:

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TO
148	Hong Kong	11842	210	11572	60	1	
174	Channel Islands	4066	86	3956	24	0	
190	Caribbean Netherlands	1613	17	1579	17	0	

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TO
208	Macao	51	0	49	2	0	

4 rows × 21 columns

Hong Kong Shapefile

In [78]:
```
# Copy relative path of .shp file
shapefile = 'Resources\HK-shp\gadm36_HKG_0.shp'                          # Relative path t

# Read shapefile using Geopandas
hk_shp = gpd.read_file(shapefile)                                  # Create Hong Kong datafram
hk_shp= hk_shp.rename(columns={'NAME_0': 'COUNTRY'})               # Rename column
hk_shp.drop(['GID_0'], axis = 1, inplace= True)                    # Drop column
hk_shp
```

Out[78]:

	COUNTRY	geometry
0	Hong Kong	MULTIPOLYGON (((113.92319 22.15681, 113.92319 ...

Carribean Netherlands Shapefile

In [79]:
```
# Copy relative path of .shp file
shapefile = 'Resources\CarNetherlands-shp\BES_adm0.shp'

# Read shapefile using Geopandas
CN_shp = gpd.read_file(shapefile)
CN_shp= CN_shp.rename(columns={'NAME_0': 'COUNTRY'})
CN_shp.drop(['adm0code'], axis = 1, inplace= True)
CN_shp
```

Out[79]:

	COUNTRY	geometry
0	Caribbean Netherlands	MULTIPOLYGON (((-68.30847 12.16875, -68.30847 ...

Channel Island Shapefile

In [80]:
```
# Copy relative path of .shp file
shapefile = 'Resources\Channel-shp\cinms_py.shp'

# Read shapefile using Geopandas
ch_shp = gpd.read_file(shapefile)
ch_shp= ch_shp.rename(columns={'AREA_NAME': 'COUNTRY'})                                  # R
ch_shp.drop(['SANCTUARY', 'POLY_ID', 'DATUM'], axis = 1, inplace= True)                  # D
ch_shp.drop([0], axis = 0, inplace = True)                                               # D
ch_shp.COUNTRY= ch_shp.COUNTRY.replace("Northern Section","Channel Islands")             # R
ch_shp
```

Out[80]:

	COUNTRY	geometry
1	Channel Islands	POLYGON ((-120.41801 34.20707, -120.38830 34.2...

Macao Shapefile

In [81]:
```
# Macao Shapefile
# Copy relative path of .shp file
shapefile = 'Resources\Macao-shp\MAC_adm0.shp'

# Read shapefile using Geopandas
macao_shp = gpd.read_file(shapefile)
macao_shp= macao_shp.rename(columns={'NAME_ENGLI': 'COUNTRY'})
macao_shp.drop(macao_shp.iloc[:,:2], axis = 1, inplace= True)
macao_shp.drop(macao_shp.iloc[:,1:-1], axis = 1, inplace= True)
macao_shp.COUNTRY= macao_shp.COUNTRY.replace("Northern Section","Channel Islands")
macao_shp
```

Out[81]:

	COUNTRY	geometry
0	Macao	MULTIPOLYGON (((113.58178 22.21544, 113.58263 ...

Concatenate Shapefile DataFrames

In [82]:
```
shp_df = pd.concat([macao_shp, ch_shp, CN_shp, hk_shp], axis=0)  # Vertically stack Dat
```

In [83]:
```
shp_df['COUNTRY'] =pd.Series(shp_df['COUNTRY'], dtype= "string")
geo_global_data['COUNTRY'] =pd.Series(geo_global_data['COUNTRY'], dtype= "string")
shp_df.COUNTRY = shp_df.COUNTRY.str.strip()
```

Update GeoDataFrame

In [84]:
```
geo_global_data3= global_data.merge(shp_df, on='COUNTRY', how= 'left')      # Create ne
geo_global_data.update(geo_global_data3)                                    # Update 'g
geo_global_data.head()
```

Out[84]:

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCA
0	United States	34113146	609767	27863840	5639539	6129	
1	India	28173655	331909	25939504	1902242	8944	
2	Brazil	16547674	462966	14964631	1120077	8318	
3	France	5667324	109528	5333597	224199	2945	

	COUNTRY	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	SERIOUS_CRITICAL	TOTCA
4	Turkey	5249404	47527	5114624	87253	1339	

5 rows × 21 columns

In [85]:
```
# DataFrame clean of all null values
geo_global_data.isnull().any()
```

Out[85]:
```
COUNTRY                 False
TOTALCASES              False
TOTALDEATHS             False
TOTALRECOVERED          False
ACTIVECASES             False
SERIOUS_CRITICAL        False
TOTCASES_PER_1M         False
DEATH_PER_1M            False
TOTALTESTS              False
TESTS_PER_1M            False
POPULATION              False
CONTINENT               False
DEATH_RATE              False
SURVIVAL_RATE           False
PERCENT_TESTS_POSITIVE  False
GDP_PER_CAPITA          False
LAND_AREA               False
POPULATION_DENSITY      False
LAT                     False
LONG                    False
geometry                False
dtype: bool
```

State Geospatial Data

In [86]:
```
us_data.head()
```

Out[86]:

	STATE	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	TOTCASES_PER_1M	DEATH_
0	California	3789731.0	63247.0	2054430.0	1672054.0	95913.0	
1	Texas	2954513.0	51728.0	2835681.0	67104.0	101894.0	
2	Florida	2320818.0	36774.0	1943113.0	340931.0	108057.0	
3	New York	2154361.0	53594.0	1718623.0	382144.0	110744.0	
4	Illinois	1382186.0	25223.0	1301109.0	55854.0	109076.0	

Lat and Long Coordinates

In [87]:
```
# Instantiate Nominatim
```

```
locator = Nominatim(user_agent="myGeocoder")

# Gather Longitude and Latitude coordinates
state_locs=[]                                                    # Empty
for i in us_data.STATE:                                          # Itera
        location = locator.geocode([i])                          # Gathe
        state_locs.append([location.latitude, location.longitude])  # Appen
```

In [88]:
```
state_locs
```

Out[88]:
```
[[36.7014631, -118.755997],
 [31.8160381, -99.5120986],
 [27.7567667, -81.4639835],
 [40.7127281, -74.0060152],
 [40.0796606, -89.4337288],
 [40.9699889, -77.7278831],
 [32.3293809, -83.1137366],
 [40.2253569, -82.6881395],
 [40.0757384, -74.4041622],
 [35.6729639, -79.0392919],
 [43.6211955, -84.6824346],
 [34.395342, -111.763275],
 [35.7730076, -86.2820081],
 [40.3270127, -86.1746933],
 [42.3788774, -72.032366],
 [37.1232245, -78.4927721],
 [44.4308975, -89.6884637],
 [45.9896587, -94.6113288],
 [38.7604815, -92.5617875],
 [33.6874388, -80.4363743],
 [33.2588817, -86.8295337],
 [38.7251776, -105.607716],
 [30.8703881, -92.007126],
 [39.5162234, -76.9382069],
 [37.5726028, -85.1551411],
 [34.9550817, -97.2684063],
 [38.8949924, -77.0365581],
 [39.4225192, -111.714358],
 [41.9216734, -93.3122705],
 [41.6500201, -72.7342163],
 [35.2048883, -92.4479108],
 [39.5158825, -116.8537227],
 [32.9715645, -89.7348497],
 [38.27312, -98.5821872],
 [41.7370229, -99.5873816],
 [34.5708167, -105.993007],
 [43.9792797, -120.737257],
 [43.6447642, -114.015407],
 [38.4758406, -80.8408415],
 [41.7962409, -71.5992372],
 [44.6471761, -100.348761],
 [47.3752671, -109.638757],
 [47.6201461, -100.540737],
 [38.6920451, -75.4013315],
 [43.4849133, -71.6553992],
 [45.709097, -68.8590201],
 [64.4459613, -149.680909],
 [43.1700264, -107.568534],
 [38.89379365, -76.98799757261312],
 [19.58726775, -155.42688965312746],
 [44.5990718, -72.5002608],
```

```
[19.58726775, -155.42688965312746],
[33.6874388, -80.4363743]]
```

In [89]:
```
# Create DataFrame with coordinates
state_loc = pd.DataFrame(state_locs, columns = ['LAT', 'LONG'])
state_loc.head()
```

Out[89]:

	LAT	LONG
0	36.701463	-118.755997
1	31.816038	-99.512099
2	27.756767	-81.463983
3	40.712728	-74.006015
4	40.079661	-89.433729

In [90]:
```
# Add coordinates to 'us_data' DataFrame
us_data = us_data.join(state_loc)
us_data.head()
```

Out[90]:

	STATE	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	TOTCASES_PER_1M	DEATH_
0	California	3789731.0	63247.0	2054430.0	1672054.0	95913.0	
1	Texas	2954513.0	51728.0	2835681.0	67104.0	101894.0	
2	Florida	2320818.0	36774.0	1943113.0	340931.0	108057.0	
3	New York	2154361.0	53594.0	1718623.0	382144.0	110744.0	
4	Illinois	1382186.0	25223.0	1301109.0	55854.0	109076.0	

Create Shapefile Dataframe

In [91]:
```
# Store relative path
shapefile = 'Resources\stateshapes\cb_2018_us_state_500k.shp'

# Read shapefile using Geopandas
gdus = gpd.read_file(shapefile)
gdus= gdus.rename(columns={'NAME': 'STATE'})
gdus.head()
```

Out[91]:

	STATEFP	STATENS	AFFGEOID	GEOID	STUSPS	STATE	LSAD	ALAND	AWATER	
0	28	01779790	0400000US28	28	MS	Mississippi	00	121533519481	3926919758	M
1	37	01027616	0400000US37	37	NC	North Carolina	00	125923656064	13466071395	M

	STATEFP	STATENS	AFFGEOID	GEOID	STUSPS	STATE	LSAD	ALAND	AWATER
2	40	01102857	0400000US40	40	OK	Oklahoma	00	177662925723	3374587997
3	51	01779803	0400000US51	51	VA	Virginia	00	102257717110	8528531774
4	54	01779805	0400000US54	54	WV	West Virginia	00	62266474513	489028543

Merge DataFrames

In [92]:
```
us_data.STATE = us_data.STATE.str.strip()
gdus.STATE = gdus.STATE.str.strip()
us_geodata=us_data.merge(gdus, on='STATE', how='left')
```

Cast to GeoDataFrame

In [93]:
```
us_geodata = GeoDataFrame(us_geodata)
type(us_geodata)
```

Out[93]:
```
geopandas.geodataframe.GeoDataFrame
```

Clean GeoDataFrame

In [94]:
```
us_geodata.columns
```

Out[94]:
```
Index(['STATE', 'TOTALCASES', 'TOTALDEATHS', 'TOTALRECOVERED', 'ACTIVECASES',
       'TOTCASES_PER_1M', 'DEATH_PER_1M', 'TOTALTESTS', 'TESTS_PER_1M',
       'POPULATION', 'DEATH_RATE', 'SURVIVAL_RATE', 'PERCENT_TESTS_POSITIVE',
       'LAT', 'LONG', 'STATEFP', 'STATENS', 'AFFGEOID', 'GEOID', 'STUSPS',
       'LSAD', 'ALAND', 'AWATER', 'geometry'],
      dtype='object')
```

In [95]:
```
us_geodata=us_geodata.drop(['STATEFP', 'STATENS',
       'AFFGEOID', 'GEOID', 'LSAD', 'ALAND', 'AWATER'], axis=1)

us_geodata=us_geodata.rename(columns= {'STUSPS':'CODE'})
```

In [96]:
```
us_geodata.head()
```

Out[96]:

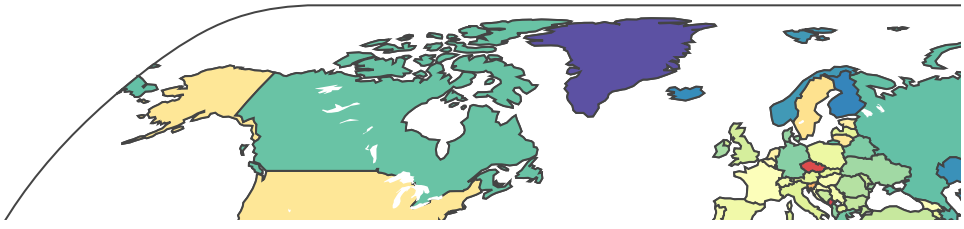
	STATE	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	TOTCASES_PER_1M	DEATH_

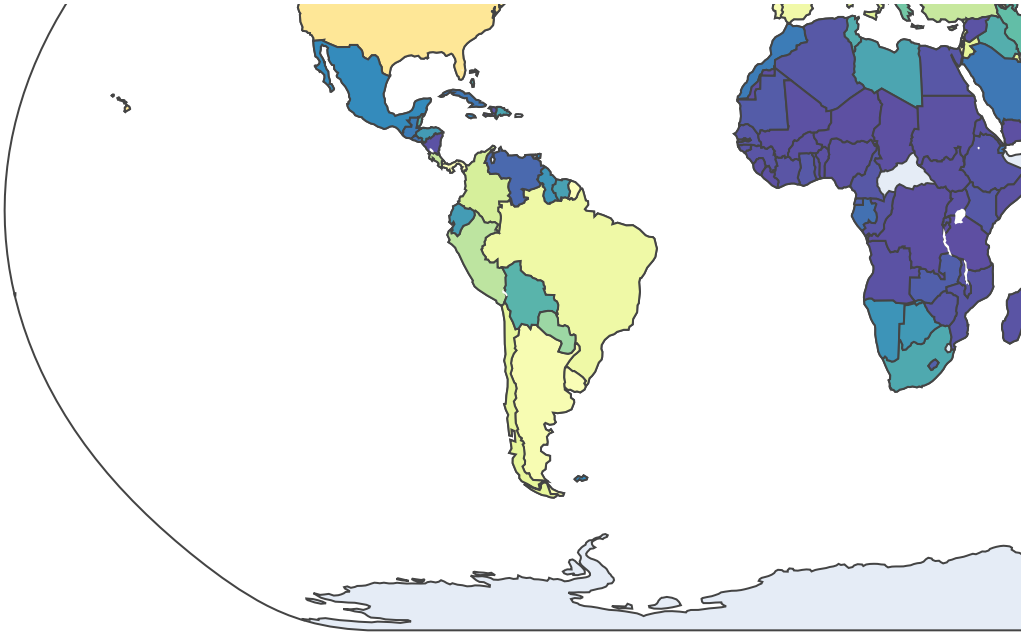
	STATE	TOTALCASES	TOTALDEATHS	TOTALRECOVERED	ACTIVECASES	TOTCASES_PER_1M	DEATH_
0	California	3789731.0	63247.0	2054430.0	1672054.0	95913.0	
1	Texas	2954513.0	51728.0	2835681.0	67104.0	101894.0	
2	Florida	2320818.0	36774.0	1943113.0	340931.0	108057.0	
3	New York	2154361.0	53594.0	1718623.0	382144.0	110744.0	
4	Illinois	1382186.0	25223.0	1301109.0	55854.0	109076.0	

GeoMap using Plotly

In [97]:
```
fig = px.choropleth(geo_global_data,                                    # Create Ch
                    geojson=geo_global_data.geometry,                   # Use geome
                    locations=geo_global_data.COUNTRY,                  # Country n
                    locationmode='country names',                       # Set locat
                    color="TOTCASES_PER_1M",                            # Numerical
                    projection = "natural earth",                       # Map layou
                    width=1150,
                    height=600,
                    color_continuous_scale="spectral_r",                # Color sch
                    title='Total Cases Per Million by Country')

#fig.update_geos(fitbounds="locations", visible=False)
fig.update_layout(title_x=.5)                                           # Center ti
fig.show()
```

Total Cases Per Million b

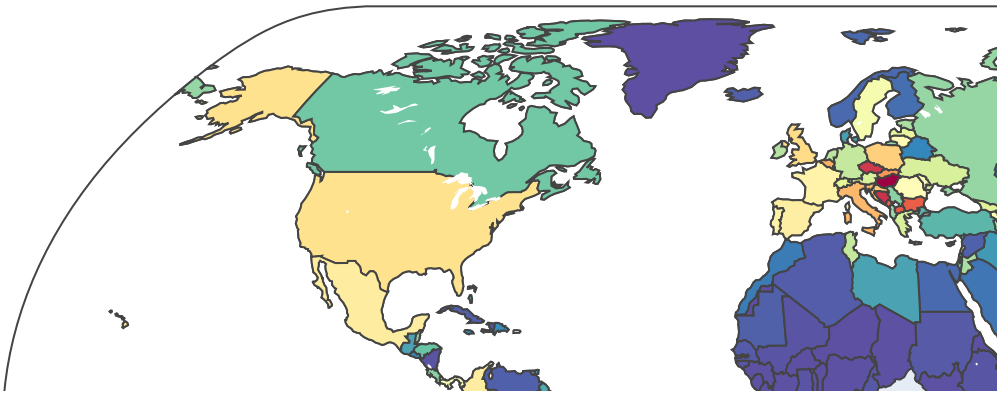




In [98]:

```
fig = px.choropleth(geo_global_data,
                    geojson=geo_global_data.geometry,
                    locations=geo_global_data.COUNTRY,
                    locationmode='country names',
                    projection = "natural earth",
                    color="DEATH_PER_1M",
                    width=1150,
                    height=600,
                    color_continuous_scale="spectral_r",
                    title='Total Deaths Per Million by Country')

#fig.update_geos(fitbounds="locations", visible=False)
fig.update_layout(title_x=.5)
fig.show()
```

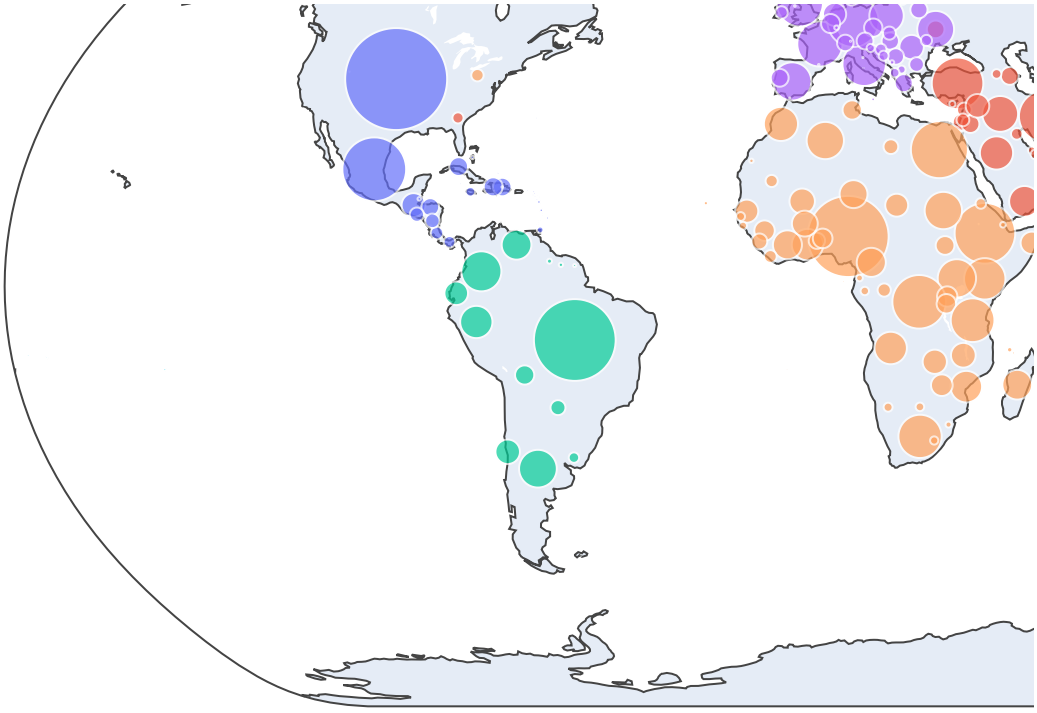
Total Deaths Per Million b



In [99]:

```
px.set_mapbox_access_token("pk.eyJ1Ijoib21rYXJzMSIsImEiOiJja2x3b2VxZGYwZWNtMnVrdnl1aTFh

fig = px.scatter_geo(geo_global_data,
                     lat=geo_global_data.LAT,                # Latitutinal values
                     lon=geo_global_data.LONG,               # Longitudinal values
                     hover_name="COUNTRY",                   # Hovering over point will
                     projection="natural earth",             # Map type
                     size = 'POPULATION',                    # Numerical value of popula
                     color = 'CONTINENT',                    # Hue
                     width=1150,
                     height=600,
                     size_max = 75,                          # Max size of bubbles
                     title = 'Population by Country',
                     custom_data=["COUNTRY","TOTALCASES","TESTS_PER_1M","SURVIVAL_RATE",


fig.update_traces(hovertemplate="<br>".join([
        "COUNTRY: %{customdata[0]}",
        "TOTAL CASES: %{customdata[1]}",
        "TESTS PER MILLION: %{customdata[2]}",
        "SURVIVAL RATE : %{customdata[3]}",
        "DEATH RATE: %{customdata[4]}",
        "ACTIVE CASES: %{customdata[5]}",
        "GDP PER CAPITA: %{customdata[6]}"
    ]))
fig.update_layout(title_x=.5)
fig.show()
```

Population by Cou

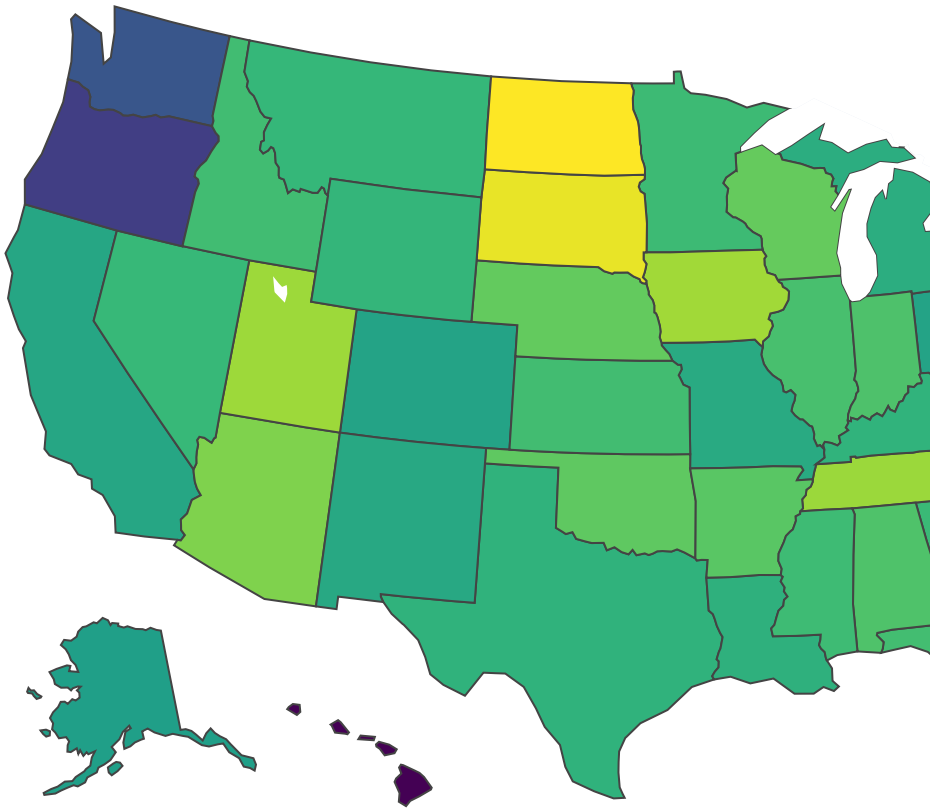


State Data

In [100…

```
fig = px.choropleth(us_geodata,
                    geojson=us_geodata.geometry,
                    locations=us_geodata.CODE,
                    color="TOTCASES_PER_1M",
                    color_continuous_scale="Viridis",
                    locationmode = "USA-states",
                    scope="usa",
                    labels={'TOTCASES_PER_1M':'Total Confirmed Cases Per Million'},
                    title= 'Total Confirmed Cases by State',
                    width=1150,
                    height=600,
                    hover_name="STATE",
                    custom_data=["STATE","TOTCASES_PER_1M","POPULATION"])


fig.update_traces(hovertemplate="<br>".join([
        "STATE: %{customdata[0]}",
        "TOTCASES_PER_1M: %{customdata[1]}",
        "POPULATION: %{customdata[2]}"
    ]))

fig.update_layout(title_x=.5)
fig.show()
```

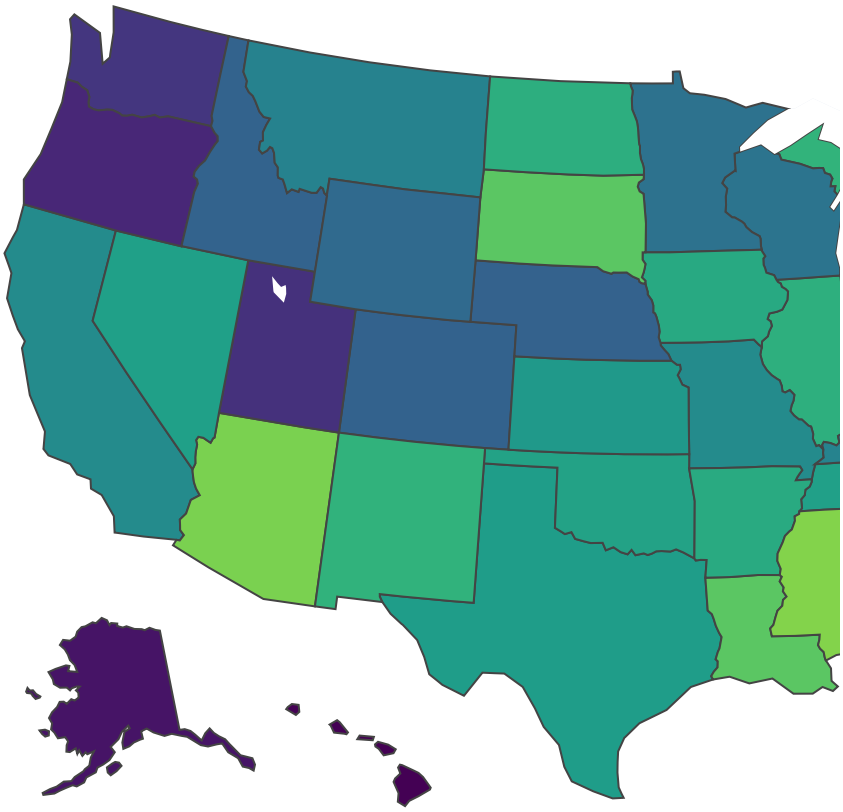
Total Confirmed Cases



In [101…

```
fig = px.choropleth(us_geodata,
                    geojson=us_geodata.geometry,
                    locations=us_geodata.CODE,
                    color="DEATH_PER_1M",
                    color_continuous_scale="Viridis",
                    locationmode = "USA-states",
                    scope="usa",
                    labels={'DEATH_PER_1M':'Deaths per million'},
                    title= 'Deaths Per Million by State',
                    width=1150,
                    height=600,
                    hover_name="STATE",
                    custom_data=["STATE","DEATH_PER_1M","POPULATION"])

fig.update_traces(hovertemplate="<br>".join([
        "STATE: %{customdata[0]}",
        "DEATHS PER MILLION: %{customdata[1]}",
        "POPULATION: %{customdata[2]}"
    ]))

fig.update_layout(title_x=.5)
fig.show()
```

Deaths Per Million by



In [ ]: