PANDASEABORN PROJECT

Questions:

1)Select a dataset or a csv file in python project

2)Apply 20 functions on pandas in question form

3)20 graphs for visualization: unvaried, by varied, multivaried graphs

Title of the project

Covid-19 Data Analysis using Python

In this project, we are going to analyse Covid-19 data using Python.

Here the dataset used for analysis and visualization is a corona data, a csv file. Libraries are imported for NumPy, panda, matplotlib and seaborn.

It works on information related to the confirmed cases, active cases, recovered cases, serious/critical cases and death cases. In particular, we analyse data of top 10 countries' cases and plot information using unvaried, by varied, multivaried graphs.

PANDASEABORN PROJECT

1) Import the required python libraries.

In [1]:
```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

2) Reading the datasets. It contains datasets of world meter data combined data. The dataset is present in .csv extension files. The file reding is done with pd.read_csv("file name") to convert that file into data frame.

In [2]:
```
#to read a csv file
df=pd.read_csv('worldometer.csv')
df
```

Out[2]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	C
0	USA	North America	3.311981e+08	5032179	NaN	162804.0	NaN	2576668.0	NaN	2292707.0	18296.0	
1	Brazil	South America	2.127107e+08	2917562	NaN	98644.0	NaN	2047660.0	NaN	771258.0	8318.0	
2	India	Asia	1.381345e+09	2025409	NaN	41638.0	NaN	1377384.0	NaN	606387.0	8944.0	
3	Russia	Europe	1.459409e+08	871894	NaN	14606.0	NaN	676357.0	NaN	180931.0	2300.0	
4	South Africa	Africa	5.938157e+07	538184	NaN	9604.0	NaN	387316.0	NaN	141264.0	539.0	
...	...	...	...	...	...	...	...	...	...	...	...	
204	Montserrat	North America	4.992000e+03	13	NaN	1.0	NaN	10.0	NaN	2.0	NaN	
205	Caribbean Netherlands	North America	2.624700e+04	13	NaN	NaN	NaN	7.0	NaN	6.0	NaN	
206	Falkland Islands	South America	3.489000e+03	13	NaN	NaN	NaN	13.0	NaN	0.0	NaN	
207	Vatican City	Europe	8.010000e+02	12	NaN	NaN	NaN	12.0	NaN	0.0	NaN	

PANDASEABORN PROJECT

3) Pandas head() method is used to return top n (5 by default) rows of a data frame or series.

```
In [3]: #head() will give first five values
        df.head()
```

Out[3]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	Cas
0	USA	North America	3.311981e+08	5032179	NaN	162804.0	NaN	2576668.0	NaN	2292707.0	18296.0	1
1	Brazil	South America	2.127107e+08	2917562	NaN	98644.0	NaN	2047660.0	NaN	771258.0	8318.0	1
2	India	Asia	1.381345e+09	2025409	NaN	41638.0	NaN	1377384.0	NaN	606387.0	8944.0	
3	Russia	Europe	1.459409e+08	871894	NaN	14606.0	NaN	676357.0	NaN	180931.0	2300.0	
4	South Africa	Africa	5.938157e+07	538184	NaN	9604.0	NaN	387316.0	NaN	141264.0	539.0	

4) Pandas tail() method is used to return bottom n (5 by default) rows of a data frame or series.

```
In [4]: #tail() will give last five values
        df.tail()
```

Out[4]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	Cas
204	Montserrat	North America	4992.0	13	NaN	1.0	NaN	10.0	NaN	2.0	NaN	
205	Caribbean Netherlands	North America	26247.0	13	NaN	NaN	NaN	7.0	NaN	6.0	NaN	
206	Falkland Islands	South America	3489.0	13	NaN	NaN	NaN	13.0	NaN	0.0	NaN	
207	Vatican City	Europe	801.0	12	NaN	NaN	NaN	12.0	NaN	0.0	NaN	1
208	Western Sahara	Africa	598682.0	10	NaN	1.0	NaN	8.0	NaN	1.0	NaN	

5) **Pandas dataframe.info() function is used to get a concise summary of the dataframe. It comes really handy when doing exploratory analysis of the data.**

```
In [5]: #.info() will give information about whole data frame
        df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209 entries, 0 to 208
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Country/Region    209 non-null    object 
 1   Continent         208 non-null    object 
 2   Population        208 non-null    float64
 3   TotalCases        209 non-null    int64  
 4   NewCases          4 non-null      float64
 5   TotalDeaths       188 non-null    float64
 6   NewDeaths         3 non-null      float64
 7   TotalRecovered    205 non-null    float64
 8   NewRecovered      3 non-null      float64
 9   ActiveCases       205 non-null    float64
 10  Serious,Critical  122 non-null    float64
 11  Tot Cases/1M pop  208 non-null    float64
 12  Deaths/1M pop     187 non-null    float64
 13  TotalTests        191 non-null    float64
 14  Tests/1M pop      191 non-null    float64
 15  WHO Region        184 non-null    object 
dtypes: float64(12), int64(1), object(3)
memory usage: 26.2+ KB
```

PANDASEABORN PROJECT

6) **Pandas describe() is used to view some basic statistical details like percentile, mean, std, etc. of a data frame or a series of numeric values.**

In [6]:
```
#shows statistical anylysis of datafame
df.describe()
```

Out[6]:

	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	Tot Cases/1M pop	Deatl
count	2.080000e+02	2.090000e+02	4.000000	188.000000	3.000000	2.050000e+02	3.000000	2.050000e+02	122.000000	208.000000	187.0(
mean	3.041549e+07	9.171850e+04	1980.500000	3792.590426	300.000000	5.887898e+04	1706.000000	2.766433e+04	534.393443	3196.024038	98.6
std	1.047661e+08	4.325867e+05	3129.611424	15487.184877	451.199512	2.566984e+05	2154.779803	1.746327e+05	2047.518613	5191.986457	174.9
min	8.010000e+02	1.000000e+01	20.000000	1.000000	1.000000	7.000000e+00	42.000000	0.000000e+00	1.000000	3.000000	0.0
25%	9.663140e+05	7.120000e+02	27.500000	22.000000	40.500000	3.340000e+02	489.000000	8.600000e+01	3.250000	282.000000	6.0(
50%	7.041972e+06	4.491000e+03	656.000000	113.000000	80.000000	2.178000e+03	936.000000	8.990000e+02	27.500000	1015.000000	29.0(
75%	2.575614e+07	3.689600e+04	2609.000000	786.000000	449.500000	2.055300e+04	2538.000000	7.124000e+03	160.250000	3841.750000	98.0(
max	1.381345e+09	5.032179e+06	6590.000000	162804.000000	819.000000	2.576668e+06	4140.000000	2.292707e+06	18296.000000	39922.000000	1238.0(

PANDASEABORN PROJECT

7) Pandas isnull() and notnull() methods are used to check and manage NULL values in a data frame.

In [7]:
```
#isnull will shows null values in booan form in dataframe
#False =not a null value, True= A null value
df.isnull()
```

Out[7]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	Cas
0	False	False	False	False	True	False	True	False	True	False	False	
1	False	False	False	False	True	False	True	False	True	False	False	
2	False	False	False	False	True	False	True	False	True	False	False	
3	False	False	False	False	True	False	True	False	True	False	False	
4	False	False	False	False	True	False	True	False	True	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	
204	False	False	False	False	True	False	True	False	True	False	True	
205	False	False	False	False	True	True	True	False	True	False	True	
206	False	False	False	False	True	True	True	False	True	False	True	
207	False	False	False	False	True	True	True	False	True	False	True	
208	False	False	False	False	True	False	True	False	True	False	True	

209 rows × 16 columns

8) **Sum() calculates the sum of elements for each row and column. Since sum() calculate as True=1 and False=0 , you can count the number of missing values in each row and column by calling sum() from the result of isnull() . You can count missing values in each column by default, and in each row with axis=1.**

```
In [8]: #gives null values in a perticular column with count
        df.isnull().sum()
```

```
Out[8]: Country/Region        0
        Continent             1
        Population            1
        TotalCases            0
        NewCases            205
        TotalDeaths          21
        NewDeaths           206
        TotalRecovered        4
        NewRecovered        206
        ActiveCases           4
        Serious,Critical     87
        Tot Cases/1M pop      1
        Deaths/1M pop        22
        TotalTests           18
        Tests/1M pop         18
        WHO Region           25
        dtype: int64
```

PANDASEABORN PROJECT

9) Pandas DataFrame.columns attribute return the column labels of the given Dataframe.

In [9]:
```
#df.columns will gives us all the columns
#present in the dataset/csv file in list format
df.columns
```

Out[9]:
```
Index(['Country/Region', 'Continent', 'Population', 'TotalCases', 'NewCases',
       'TotalDeaths', 'NewDeaths', 'TotalRecovered', 'NewRecovered',
       'ActiveCases', 'Serious,Critical', 'Tot Cases/1M pop', 'Deaths/1M pop',
       'TotalTests', 'Tests/1M pop', 'WHO Region'],
      dtype='object')
```

In [10]:
```
df
```

Out[10]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	C
0	USA	North America	3.311981e+08	5032179	NaN	162804.0	NaN	2576668.0	NaN	2292707.0	18296.0	
1	Brazil	South America	2.127107e+08	2917562	NaN	98644.0	NaN	2047660.0	NaN	771258.0	8318.0	
2	India	Asia	1.381345e+09	2025409	NaN	41638.0	NaN	1377384.0	NaN	606387.0	8944.0	
3	Russia	Europe	1.459409e+08	871894	NaN	14606.0	NaN	676357.0	NaN	180931.0	2300.0	
4	South Africa	Africa	5.938157e+07	538184	NaN	9604.0	NaN	387316.0	NaN	141264.0	539.0	
...	...	...	...	...	...	...	...	...	...	...	...	
204	Montserrat	North America	4.992000e+03	13	NaN	1.0	NaN	10.0	NaN	2.0	NaN	
205	Caribbean Netherlands	North America	2.624700e+04	13	NaN	NaN	NaN	7.0	NaN	6.0	NaN	
206	Falkland Islands	South America	3.489000e+03	13	NaN	NaN	NaN	13.0	NaN	0.0	NaN	

PANDASEABORN PROJECT

Q.1 **How many countries data is being represented in this dataframe?**

nunique() function to find the number of unique values over, the column specified in a Dataframe

here nunique() function is used to find no of unique contries from the corona dataset wordometer.csv

```
In [11]: countries=df['Country/Region'].nunique()
         countries
```

```
Out[11]: 209
```

Q.2 **What is the total number of cases in each country worldwide?**

loc will extraxt all rows with column names 'Country/Region','TotalCases' with loc, we need to give column name slicing will help to extrac data from dataframe. Since loc is used we need to give the column name.

```
In [12]: ctr_tcases=df.loc[::,['Country/Region','TotalCases']]
         ctr_tcases
```

Out[12]:

	Country/Region	TotalCases
0	USA	5032179
1	Brazil	2917562
2	India	2025409
3	Russia	871894
4	South Africa	538184
...	...	...
204	Montserrat	13
205	Caribbean Netherlands	13
206	Falkland Islands	13

Q.3 **What are top 10 countries with the most number of Total cases?**

To find out total no of cases from top 10 countries, we need to extract data from TotalCases column
Pandas sort_values() function sorts a data frame in ascending or descending order of passed Column.
It's different than the sorted Python function since it cannot sort a data frame and particular column cannot be selected.

```
In [13]: ctr_10=df.sort_values(by=['TotalCases'],ascending=False).head(10)
         ctr_10
```

Out[13]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	Cas
0	USA	North America	3.311981e+08	5032179	NaN	162804.0	NaN	2576668.0	NaN	2292707.0	18296.0	1
1	Brazil	South America	2.127107e+08	2917562	NaN	98644.0	NaN	2047660.0	NaN	771258.0	8318.0	1
2	India	Asia	1.381345e+09	2025409	NaN	41638.0	NaN	1377384.0	NaN	606387.0	8944.0	
3	Russia	Europe	1.459409e+08	871894	NaN	14606.0	NaN	676357.0	NaN	180931.0	2300.0	
4	South Africa	Africa	5.938157e+07	538184	NaN	9604.0	NaN	387316.0	NaN	141264.0	539.0	
5	Mexico	North America	1.290662e+08	462690	6590.0	50517.0	819.0	308848.0	4140.0	103325.0	3987.0	
6	Peru	South America	3.301632e+07	455409	NaN	20424.0	NaN	310337.0	NaN	124648.0	1426.0	1
7	Chile	South America	1.913251e+07	366671	NaN	9889.0	NaN	340168.0	NaN	16614.0	1358.0	1
8	Colombia	South America	5.093626e+07	357710	NaN	11939.0	NaN	192355.0	NaN	153416.0	1493.0	

Q.4 List the countries with all cases of Novel Coronavirus (COVID-19) died

To find out countries with total cases=total deaths we need to use sort function to arrange total cases in secending order and check if total cases greater than or equal to 0.

Q.4 To list countries with all cases of Novel Coronavirus (COVID-19) died

In [14]:
```
data=df.groupby('Country/Region')['TotalCases','TotalDeaths','TotalRecovered','ActiveCases'].sum().reset_index()
res=data[data['TotalCases']==data['TotalDeaths']]
res= res[['Country/Region', 'TotalCases', 'TotalDeaths']]
res= res.sort_values('TotalCases', ascending=False)
res= res[res['TotalCases']>0]
print(res)
```

```
Empty DataFrame
Columns: [Country/Region, TotalCases, TotalDeaths]
Index: []
```

Q.5 List down the Serious/Critical cases in the world with country name.

Here we have to find out all serious/critical cases in all countries over the world with country name, so we need to select country/Region column and serious/critical column.

In [15]:
```
df[['Country/Region','Serious,Critical']]
```

Out[15]:

	Country/Region	Serious,Critical
0	USA	18296.0
1	Brazil	8318.0
2	India	8944.0
3	Russia	2300.0
4	South Africa	539.0
...	...	...
204	Montserrat	NaN
205	Caribbean Netherlands	NaN
206	Falkland Islands	NaN

Q.6 Display the country with highest no of cases.

Pandas nlargest() method is used to get n largest values from a data frame or a series. The syntax for nlargest() is

DataFrame.nlargest(n, columns, keep='first')

n: int, Number of values to select
columns: Column to check for values or user can select column while calling too. [For example: data["age"].nsmallest(3) OR data.nsmallest(3, "age")]
keep: object to set which value to select if duplicates exit. Options are 'first' or 'last'

Here in the dataset we have taken USA has highest no of cases.

```
In [16]: df.nlargest((1),'TotalCases')
```

Out[16]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	Case
0	USA	North America	331198130.0	5032179	NaN	162804.0	NaN	2576668.0	NaN	2292707.0	18296.0	15

Q.7 Display the country with lowest no of cases.

Pandas **nsmallest()** method is used to get n least values from a data frame or a series.

Syntax: DataFrame.nsmallest(n, columns, keep='first')
Parameters:
n: int, Number of values to select
columns: Column to check for least values or user can select column while calling too. [For example:

data[“age”].nsmallest(3) OR data.nsmallest(3, “age”)]
keep: object to set which value to select if duplicates exit. Options are ‘first’ or ‘last’.

Here in the dataset we have taken Africa has loewest no of cases.

Q.7 Display the country with lowest no of cases

```
In [17]: df.nsmallest((1),'TotalCases')
```

Out[17]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	Cas
208	Western Sahara	Africa	598682.0	10	NaN	1.0	NaN	8.0	NaN	1.0	NaN	

Q.8 Use of sort by to arrange data in ascending and descending order.

Here to find out country with highest no of cases, we can use sort function arrange the data in descending order with ascendint set as False and use iloc[0] index to extract highest no of cases in the present in the dataset.

Here in the dataset we have taken USA has highest no of cases.

PANDASEABORN PROJECT

```
In [18]: df.sort_values(by=['TotalCases'],ascending=False)
```

Out[18]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	C
0	USA	North America	3.311981e+08	5032179	NaN	162804.0	NaN	2576668.0	NaN	2292707.0	18296.0	
1	Brazil	South America	2.127107e+08	2917562	NaN	98644.0	NaN	2047660.0	NaN	771258.0	8318.0	
2	India	Asia	1.381345e+09	2025409	NaN	41638.0	NaN	1377384.0	NaN	606387.0	8944.0	
3	Russia	Europe	1.459409e+08	871894	NaN	14606.0	NaN	676357.0	NaN	180931.0	2300.0	
4	South Africa	Africa	5.938157e+07	538184	NaN	9604.0	NaN	387316.0	NaN	141264.0	539.0	
...	...	...	...	...	...	...	...	...	...	...	...	
204	Montserrat	North America	4.992000e+03	13	NaN	1.0	NaN	10.0	NaN	2.0	NaN	
205	Caribbean Netherlands	North America	2.624700e+04	13	NaN	NaN	NaN	7.0	NaN	6.0	NaN	
206	Falkland Islands	South America	3.489000e+03	13	NaN	NaN	NaN	13.0	NaN	0.0	NaN	
207	Vatican City	Europe	8.010000e+02	12	NaN	NaN	NaN	12.0	NaN	0.0	NaN	
208	Western Sahara	Africa	5.986820e+05	10	NaN	1.0	NaN	8.0	NaN	1.0	NaN	

209 rows × 16 columns

```
In [19]: df.iloc[0]
```

```
Out[19]: Country/Region                USA
         Continent           North America
         Population          331198130.0
         TotalCases              5032179
         NewCases                    NaN
         TotalDeaths            162804.0
         NewDeaths                   NaN
         TotalRecovered        2576668.0
         NewRecovered                NaN
         ActiveCases           2292707.0
         Serious,Critical        18296.0
         Tot Cases/1M pop        15194.0
         Deaths/1M pop             492.0
         TotalTests          63139605.0
         Tests/1M pop           190640.0
         WHO Region              Americas
         Name: 0, dtype: object
```

PANDASEABORN PROJECT

Here to find out country with lowest no of cases, we can use sort function arrange the data in ascending order with ascendint set as True and use iloc[208] index to extract lowest no of cases in the present in the dataset.

Here in the dataset we have taken Africa has loewest no of cases.

```
In [20]: df.sort_values(by=['TotalCases'],ascending=True)
```

Out[20]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	C
208	Western Sahara	Africa	5.986820e+05	10	NaN	1.0	NaN	8.0	NaN	1.0	NaN	
207	Vatican City	Europe	8.010000e+02	12	NaN	NaN	NaN	12.0	NaN	0.0	NaN	
205	Caribbean Netherlands	North America	2.624700e+04	13	NaN	NaN	NaN	7.0	NaN	6.0	NaN	
206	Falkland Islands	South America	3.489000e+03	13	NaN	NaN	NaN	13.0	NaN	0.0	NaN	
204	Montserrat	North America	4.992000e+03	13	NaN	1.0	NaN	10.0	NaN	2.0	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	
4	South Africa	Africa	5.938157e+07	538184	NaN	9604.0	NaN	387316.0	NaN	141264.0	539.0	
3	Russia	Europe	1.459409e+08	871894	NaN	14606.0	NaN	676357.0	NaN	180931.0	2300.0	
2	India	Asia	1.381345e+09	2025409	NaN	41638.0	NaN	1377384.0	NaN	606387.0	8944.0	
1	Brazil	South America	2.127107e+08	2917562	NaN	98644.0	NaN	2047660.0	NaN	771258.0	8318.0	
0	USA	North America	3.311981e+08	5032179	NaN	162804.0	NaN	2576668.0	NaN	2292707.0	18296.0	

209 rows × 16 columns

```
In [21]: df.iloc[208]
```

```
Out[21]: Country/Region      Western Sahara
         Continent                  Africa
         Population               598682.0
         TotalCases                     10
         NewCases                      NaN
         TotalDeaths                   1.0
         NewDeaths                     NaN
         TotalRecovered                8.0
         NewRecovered                  NaN
         ActiveCases                   1.0
         Serious,Critical              NaN
         Tot Cases/1M pop             17.0
         Deaths/1M pop                 2.0
         TotalTests                    NaN
         Tests/1M pop                  NaN
         WHO Region                 Africa
         Name: 208, dtype: object
```

Q.9 Check correlation between all columns. This method is used to find out pairwise correlations between all the columns of the DataFrame.

Pandas df.corr() function to find the correlation among the columns. We are only having four numeric columns in the Dataframe. The output Dataframe can be interpreted as for any cell, row variable correlation with the column variable is the value of the cell. As mentioned earlier, the correlation of a variable with itself is 1. For that reason, all the diagonal values are 1.00

Here the correlation is checked with each and every column.

```
In [22]: df.corr()
```

Out[22]:

	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	Tot Cases/1M pop	Deaths/1M pop
Population	1.000000	0.546158	0.889610	0.431072	0.910836	0.590239	0.854194	0.459124	0.595379	-0.009337	0.025686
TotalCases	0.546158	1.000000	0.999459	0.938622	0.998062	0.985764	0.998258	0.969423	0.967270	0.252627	0.280476
NewCases	0.889610	0.999459	1.000000	0.991894	0.995554	0.991416	0.999636	0.935500	0.985526	0.321203	0.842468
TotalDeaths	0.431072	0.938622	0.991894	1.000000	0.999507	0.935410	0.988362	0.927625	0.906627	0.237206	0.425186
NewDeaths	0.910836	0.998062	0.995554	0.999507	1.000000	0.998935	0.992651	0.888717	0.997112	0.046499	0.751519
TotalRecovered	0.590239	0.985764	0.991416	0.935410	0.998935	1.000000	0.986012	0.914566	0.949628	0.268483	0.296443
NewRecovered	0.854194	0.998258	0.999636	0.988362	0.992651	0.986012	1.000000	0.937663	0.980595	0.167036	0.825828
ActiveCases	0.459124	0.969423	0.935500	0.927625	0.888717	0.914566	0.937663	1.000000	0.945943	0.212899	0.255001
Serious,Critical	0.595379	0.967270	0.985526	0.906627	0.997112	0.949628	0.980595	0.945943	1.000000	0.209668	0.271842
Tot Cases/1M pop	-0.009337	0.252627	0.321203	0.237206	0.046499	0.268483	0.167036	0.212899	0.209668	1.000000	0.502134
Deaths/1M pop	0.025686	0.280476	0.842468	0.425186	0.751519	0.296443	0.825828	0.255001	0.271842	0.502134	1.000000
TotalTests	0.497937	0.891001	0.214572	0.850304	0.039460	0.870113	-0.081745	0.911488	0.859526	0.194120	0.264218
Tests/1M pop	-0.075129	0.029141	-0.669339	0.053870	-0.803470	0.022027	-0.869608	0.028963	0.020649	0.302728	0.156827

PANDASEABORN PROJECT

Q.10 To check total no of deaths>10000 in the countries worldwide.

Here df.where() is used to check the condition given as deaths>10000

This function checks the DataFrame for a given condition and replaces values at all the locations with NaN where the condition is False.

Here in the dataset four countries are having death count>than 10000 are USA,Brazil,India,Russia.

```
In [23]: condition = df["TotalDeaths"] >10000
         df.where(condition)
```

Out[23]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	C
0	USA	North America	3.311981e+08	5032179.0	NaN	162804.0	NaN	2576668.0	NaN	2292707.0	18296.0	
1	Brazil	South America	2.127107e+08	2917562.0	NaN	98644.0	NaN	2047660.0	NaN	771258.0	8318.0	
2	India	Asia	1.381345e+09	2025409.0	NaN	41638.0	NaN	1377384.0	NaN	606387.0	8944.0	
3	Russia	Europe	1.459409e+08	871894.0	NaN	14606.0	NaN	676357.0	NaN	180931.0	2300.0	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	
204	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
205	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
206	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
207	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
208	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

209 rows × 16 columns

Q.11 Use of value counts value_counts()

Return a Series containing counts of unique rows in the DataFrame.

Here we are getting the unique no of countries in each continent is calculated with help of df[‘columnname’].value_counts()

```
In [24]: df['Continent'].value_counts()

Out[24]: Africa              57
         Asia                48
         Europe              48
         North America       35
         South America       14
         Australia/Oceania    6
         Name: Continent, dtype: int64
```

Q.12 Drop Duplicate values in dataframe

drop_duplicates() returns a Pandas DataFrame with duplicate rows removed.

Even among duplicates, there is an option to keep the first occurrence (record) of the duplicate or the last. You can also specify the inplace and ignore_index attribute.

PANDASEABORN PROJECT

```
In [25]: df.drop_duplicates()
```

Out[25]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	C
0	USA	North America	3.311981e+08	5032179	NaN	162804.0	NaN	2576668.0	NaN	2292707.0	18296.0	
1	Brazil	South America	2.127107e+08	2917562	NaN	98644.0	NaN	2047660.0	NaN	771258.0	8318.0	
2	India	Asia	1.381345e+09	2025409	NaN	41638.0	NaN	1377384.0	NaN	606387.0	8944.0	
3	Russia	Europe	1.459409e+08	871894	NaN	14606.0	NaN	676357.0	NaN	180931.0	2300.0	
4	South Africa	Africa	5.938157e+07	538184	NaN	9604.0	NaN	387316.0	NaN	141264.0	539.0	
...	...	...	...	...	...	...	...	...	...	...	...	
204	Montserrat	North America	4.992000e+03	13	NaN	1.0	NaN	10.0	NaN	2.0	NaN	
205	Caribbean Netherlands	North America	2.624700e+04	13	NaN	NaN	NaN	7.0	NaN	6.0	NaN	
206	Falkland Islands	South America	3.489000e+03	13	NaN	NaN	NaN	13.0	NaN	0.0	NaN	
207	Vatican City	Europe	8.010000e+02	12	NaN	NaN	NaN	12.0	NaN	0.0	NaN	
208	Western Sahara	Africa	5.986820e+05	10	NaN	1.0	NaN	8.0	NaN	1.0	NaN	

209 rows × 16 columns

Q.14 Select 50 samples from dataframe

df.sample (n = no of samples to be displayed)

here randomly 50 no of samples are selected since n=50

```
In [27]: df.sample(n = 50)
```

Out[27]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critica
35	Belgium	Europe	1.159474e+07	71158	NaN	9859.0	NaN	17661.0	NaN	43638.0	61.(
203	Greenland	North America	5.678000e+04	14	NaN	NaN	NaN	14.0	NaN	0.0	NaN
31	Israel	Asia	9.197590e+06	79559	NaN	576.0	NaN	53427.0	NaN	25556.0	358.(
64	Austria	Europe	9.011577e+06	21696	NaN	719.0	NaN	19596.0	NaN	1381.0	25.(
85	Gabon	Africa	2.230563e+06	7787	NaN	51.0	NaN	5609.0	NaN	2127.0	11.(
193	Macao	Asia	6.501930e+05	46	NaN	NaN	NaN	46.0	NaN	0.0	NaN
154	Vietnam	Asia	9.742547e+07	747	NaN	10.0	NaN	392.0	NaN	345.0	NaN
96	Croatia	Europe	4.102577e+06	5404	NaN	155.0	NaN	4688.0	NaN	561.0	7.(
78	Sudan	Africa	4.394354e+07	11780	NaN	763.0	NaN	6194.0	NaN	4823.0	NaN
113	Eswatini	Africa	1.161348e+06	2968	NaN	55.0	NaN	1476.0	NaN	1437.0	5.(

Q.15 Get 5 countries with ActiveCases between 50 and 70

Here between operator is used to find out the active cases between 50 and 70 excluding both the values. Only 5 countries requied so sample(5) is used.

```
In [28]: df[df["ActiveCases"].between(50, 70, inclusive="neither")].sample(5)
```

Out[28]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	Cas
148	Andorra	Europe	77278.0	944	NaN	52.0	NaN	828.0	NaN	64.0	1.0	12
175	Trinidad and Tobago	North America	1399950.0	210	NaN	8.0	NaN	135.0	NaN	67.0	NaN	
169	Eritrea	Africa	3551175.0	282	NaN	NaN	NaN	225.0	NaN	57.0	NaN	
151	Sao Tome and Principe	Africa	219544.0	878	NaN	15.0	NaN	797.0	NaN	66.0	NaN	3
189	Belize	North America	398312.0	86	NaN	2.0	NaN	31.0	NaN	53.0	2.0	

Q.16 Get the country with its id having maximum new cases.

When you call max or min on a column, pandas returns the value that is largest/smallest. However, sometimes you want the position of the min/max, which is not possible with these functions. Instead, you should use idxmax/idxmin:

Return index of first occurrence of maximum over requested axis.NA/null values are excluded.

In the dataset we are taking first 10 countries with df.head(10). So from first 10 countries we are selecting a country with having highest newcases . Here with help of idmax, index of country having highest no of new cases is obtain in result.

So to check the country name here we have used iloc[5], we get the country as Mexico.

PANDASEABORN PROJECT

```
In [29]: df.head(10)
```

Out[29]:

	Country/Region	Continent	Population	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious,Critical	Cas
0	USA	North America	3.311981e+08	5032179	NaN	162804.0	NaN	2576668.0	NaN	2292707.0	18296.0	1
1	Brazil	South America	2.127107e+08	2917562	NaN	98644.0	NaN	2047660.0	NaN	771258.0	8318.0	1
2	India	Asia	1.381345e+09	2025409	NaN	41638.0	NaN	1377384.0	NaN	606387.0	8944.0	
3	Russia	Europe	1.459409e+08	871894	NaN	14606.0	NaN	676357.0	NaN	180931.0	2300.0	
4	South Africa	Africa	5.938157e+07	538184	NaN	9604.0	NaN	387316.0	NaN	141264.0	539.0	
5	Mexico	North America	1.290662e+08	462690	6590.0	50517.0	819.0	308848.0	4140.0	103325.0	3987.0	
6	Peru	South America	3.301632e+07	455409	NaN	20424.0	NaN	310337.0	NaN	124648.0	1426.0	1
7	Chile	South America	1.913251e+07	366671	NaN	9889.0	NaN	340168.0	NaN	16614.0	1358.0	1
8	Colombia	South America	5.093626e+07	357710	NaN	11939.0	NaN	192355.0	NaN	153416.0	1493.0	
9	Spain	Europe	4.675665e+07	354530	NaN	28500.0	NaN	NaN	NaN	NaN	617.0	

```
In [33]: df.NewCases.idxmax()
```

Out[33]: 5

PANDASEABORN PROJECT

```
In [34]: df.iloc[5]
```

```
Out[34]: Country/Region            Mexico
         Continent           North America
         Population          129066160.0
         TotalCases               462690
         NewCases                 6590.0
         TotalDeaths              50517.0
         NewDeaths                  819.0
         TotalRecovered           308848.0
         NewRecovered               4140.0
         ActiveCases              103325.0
         Serious,Critical           3987.0
         Tot Cases/1M pop           3585.0
         Deaths/1M pop               391.0
         TotalTests              1056915.0
         Tests/1M pop               8189.0
         WHO Region                Americas
         Name: 5, dtype: object
```

PANDASEABORN PROJECT

Seaborn is an amazing visualization library for statistical graphics plotting in Python. It provides beautiful default styles and color palettes to make statistical plots more attractive. It is built on the top of matplotlib library and also closely integrated into the data structures from pandas.

The number of variables of interest featured by the data classifies it as **univariate, bivariate, or multivariate**. For eg., If the data features only one variable of interest then it is a uni-variate data. Further, based on the characteristics of data, it can be classified as **categorical/discrete** and **continuous** data.

Visualization
Univariate plots: univariate data visualization (data is visualized in one-dimension)

1.Continious

a) Histogram

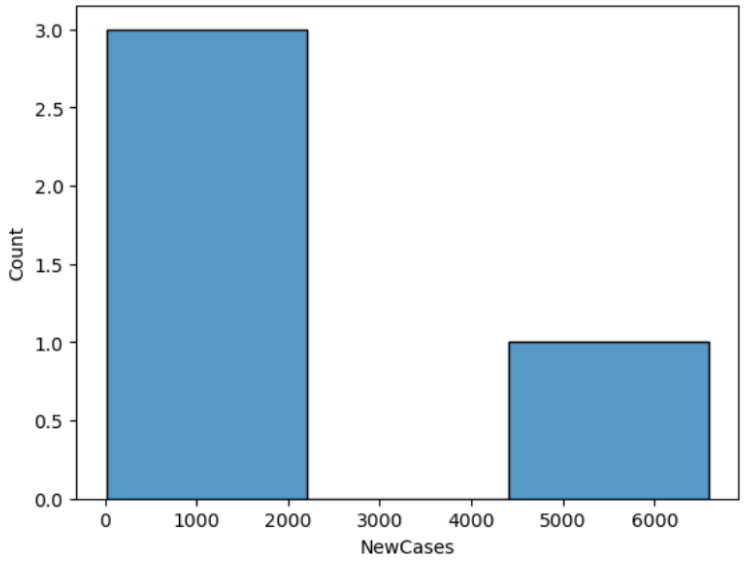
Histograms are visualization tools that represent the distribution of a set of continuous data. In a histogram, the data is divided into a set of intervals or bins (usually on the x-axis) and the count of data points that fall into each bin corresponding to the height of the bar above that bin. These bins may or may not be equal in width but are adjacent (with no gaps).

A density plot (also known as kernel density plot) is another visualization tool for evaluating data distributions. It can be considered as a smoothed histogram. The peaks of a density plot help display where values are concentrated over the interval. There are a variety of smoothing techniques. Kernel Density Estimation (KDE) is one of the techniques used to smooth a histogram.

1)Histogram

```
In [22]: sns.histplot(data=df,x='NewCases')
```

```
Out[22]: <Axes: xlabel='NewCases', ylabel='Count'>
```



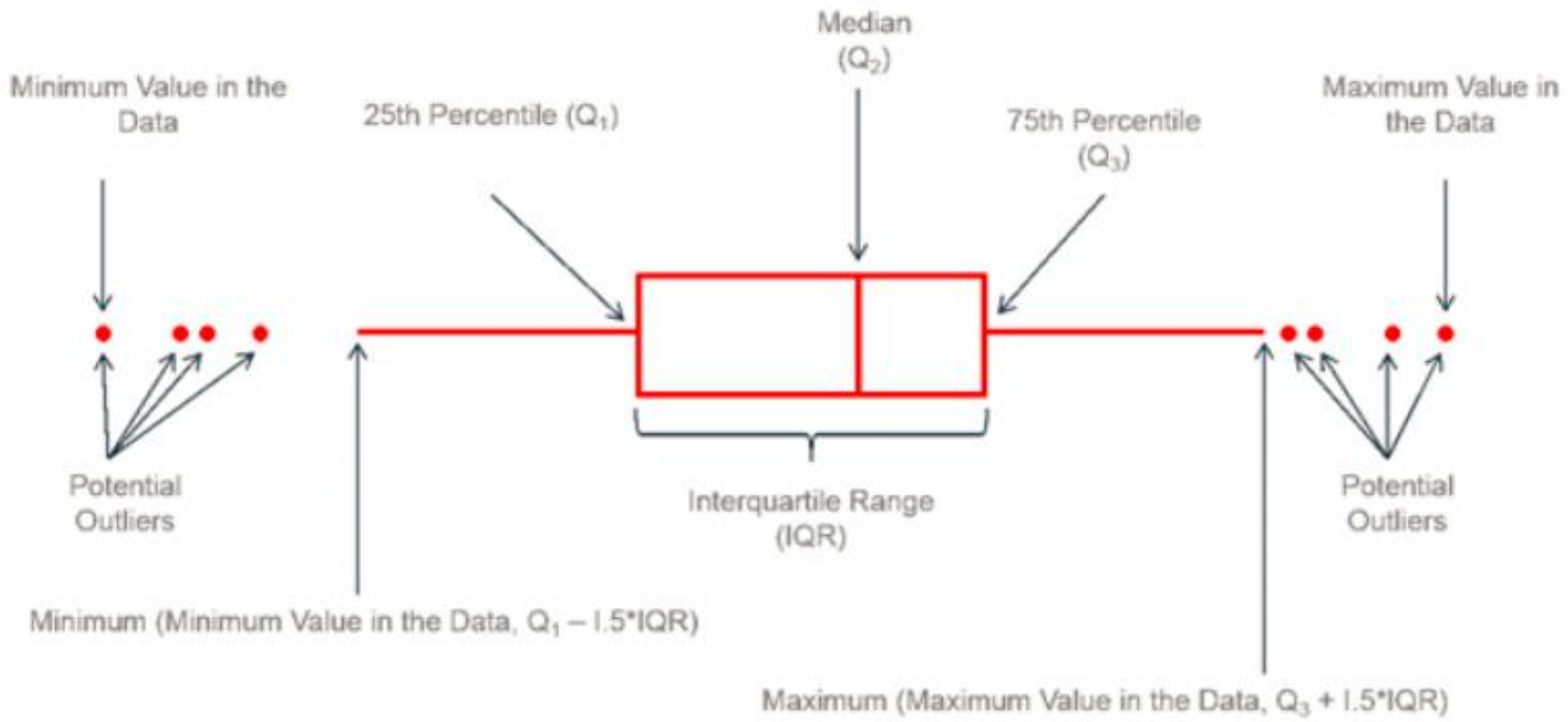
b) Boxplot

Box Plot is the visual representation of the depicting groups of numerical data through their quartiles. Boxplot is also used for detect the outlier in data set. It captures the summary of the data efficiently with a simple box and whiskers and allows us to compare easily across groups. Boxplot summarizes a sample data using 25th, 50th and 75th percentiles. These percentiles are also known as the lower quartile, median and upper quartile.

PANDASEABORN PROJECT

A box plot consist of 5 things.

- Minimum,First Quartile or 25%, Median (Second Quartile) or 50%, Third Quartile or 75%, Maximum

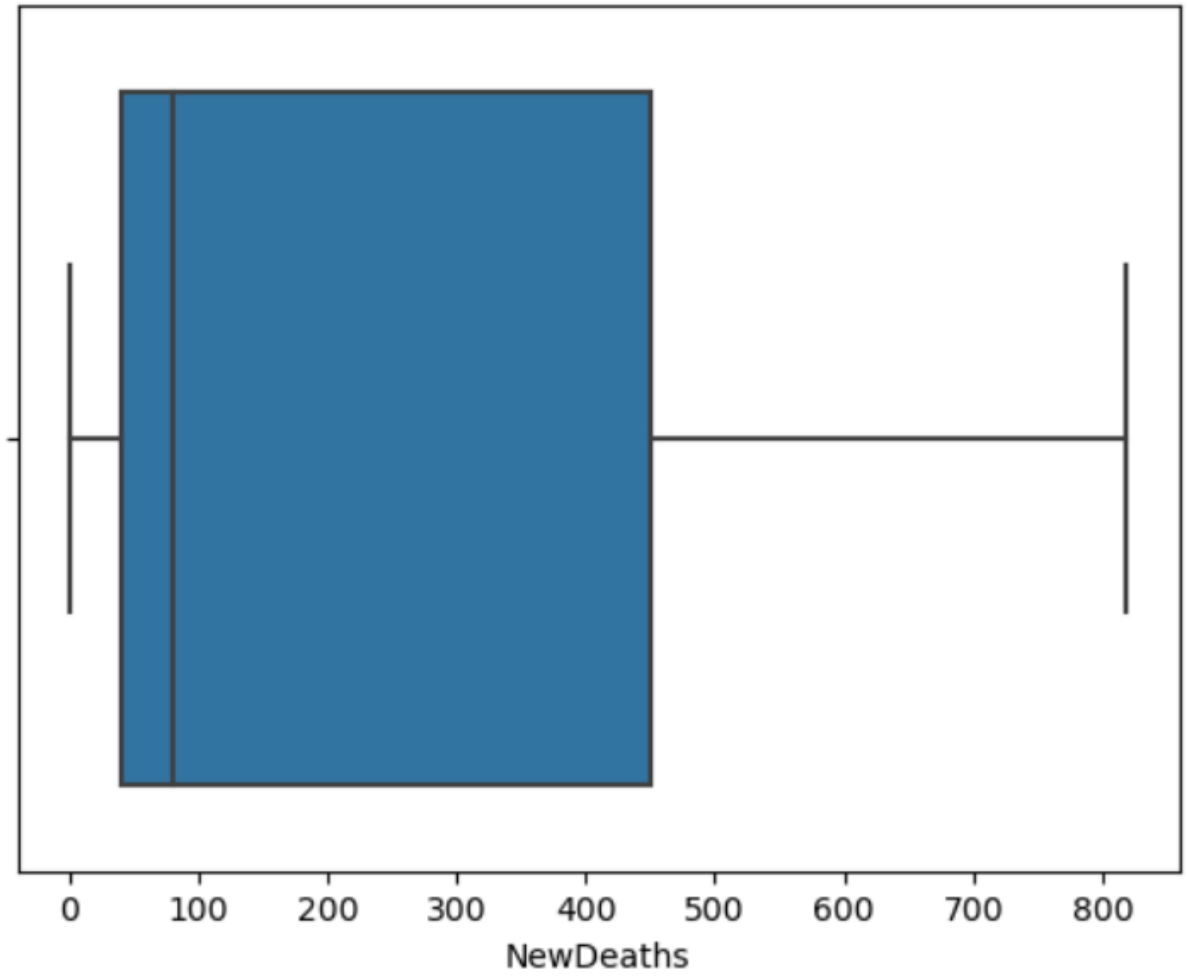


PANDASEABORN PROJECT

2)Box plot

```
In [23]: sns.boxplot(data=df,x='NewDeaths')
```

```
Out[23]: <Axes: xlabel='NewDeaths'>
```

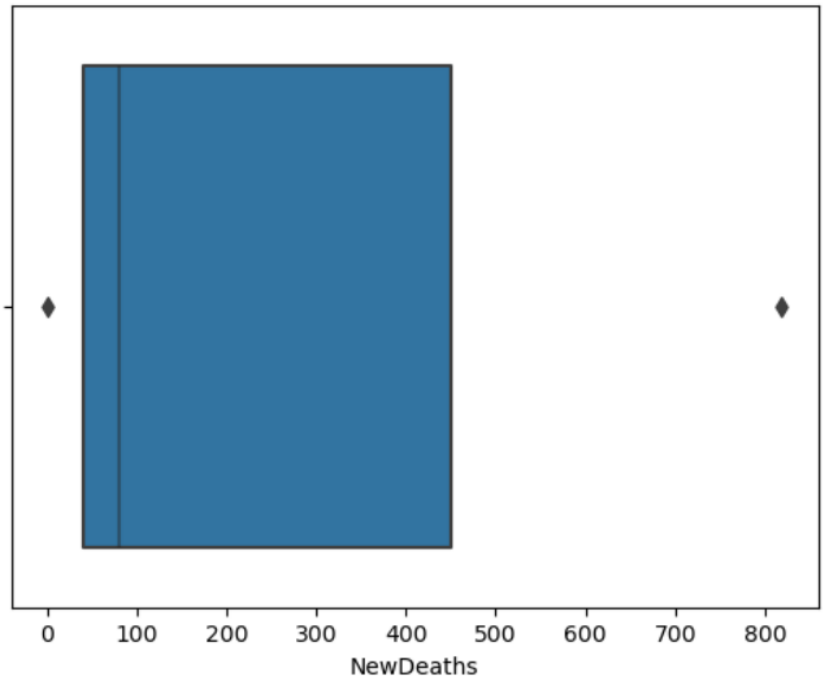


c) Boxenplot

An enhanced box plot for larger datasets. This style of plot was originally named a “letter value” plot because it shows a large number of quantiles that are defined as “letter values”. It is similar to a box plot in plotting a nonparametric representation of a distribution in which all features correspond to actual observations. By plotting more quantiles, it provides more information about the shape of the distribution, particularly in the tails.

3)Boxenplot

```
In [24]: sns.boxenplot(data=df,x='NewDeaths')
Out[24]: <Axes: xlabel='NewDeaths'>
```



d) Rugplot

The Seaborn.rugplot() method is used to plot marginal distributions and allows to draw ticks along the x and y axes. By clearly displaying the locations of individual observations, this function is meant to supplement existing displays.So the rugplot() method essentially add rugs or ticks along an already existing plot.

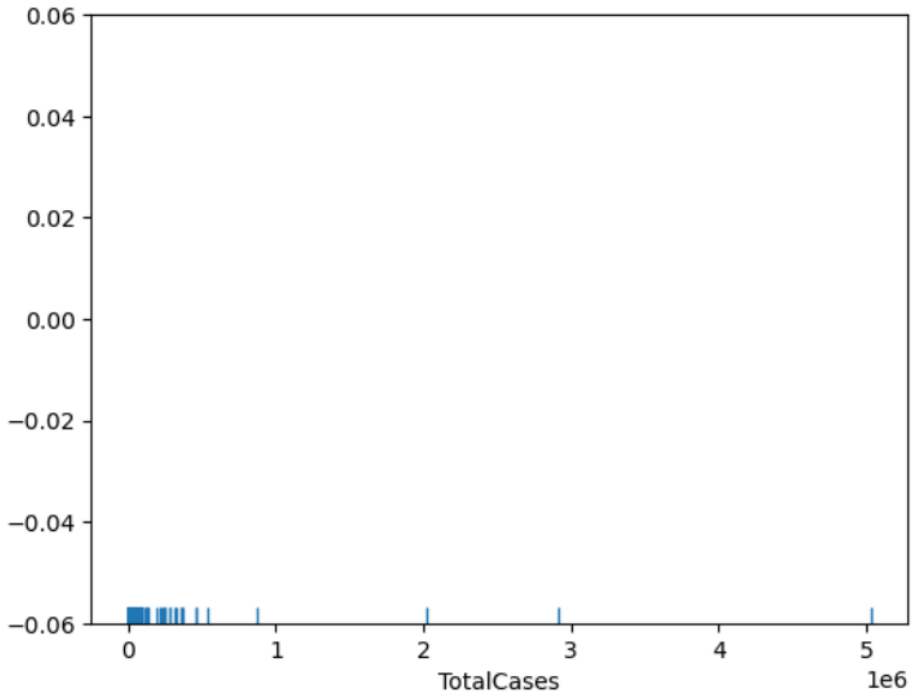
A rug plot is a graph that plots data for single quantitative values that is displayed as marks along the axes. These plots are usually used for two-dimensional scatter plots by marking the rug plot of x values along the x-axis and rug plot of y values along the y-axis.A rug plot is a very simple, but also an ideal legitimate, way of representing a distribution. It consists of vertical lines at each data point. Here, the height is arbitrary. The density of the distribution can be known by how dense the tick-marks are.

The connection between the rug plot and histogram is very direct: a histogram just creates bins along with the range of the data and then draws a bar with height equal to the number of ticks in each bin. In a rug plot, all of the data points are plotted on a single axis, one tick mark or line for each one.

Compared to a marginal histogram, the rug plot suffers somewhat in terms of readability of the distribution, but it is more compact in its representation of the data. A rug is a very short, long display of point symbols, one for each distinct value. Often a vertical pipe symbol | is used to minimize overlap. Rug plot may not be considered as a primary plot choice, but it can be a good supporter plot in certain circumstances.

4)Rugplot

```
In [25]: sns.rugplot(data=df,x='TotalCases')
Out[25]: <Axes: xlabel='TotalCases'>
```

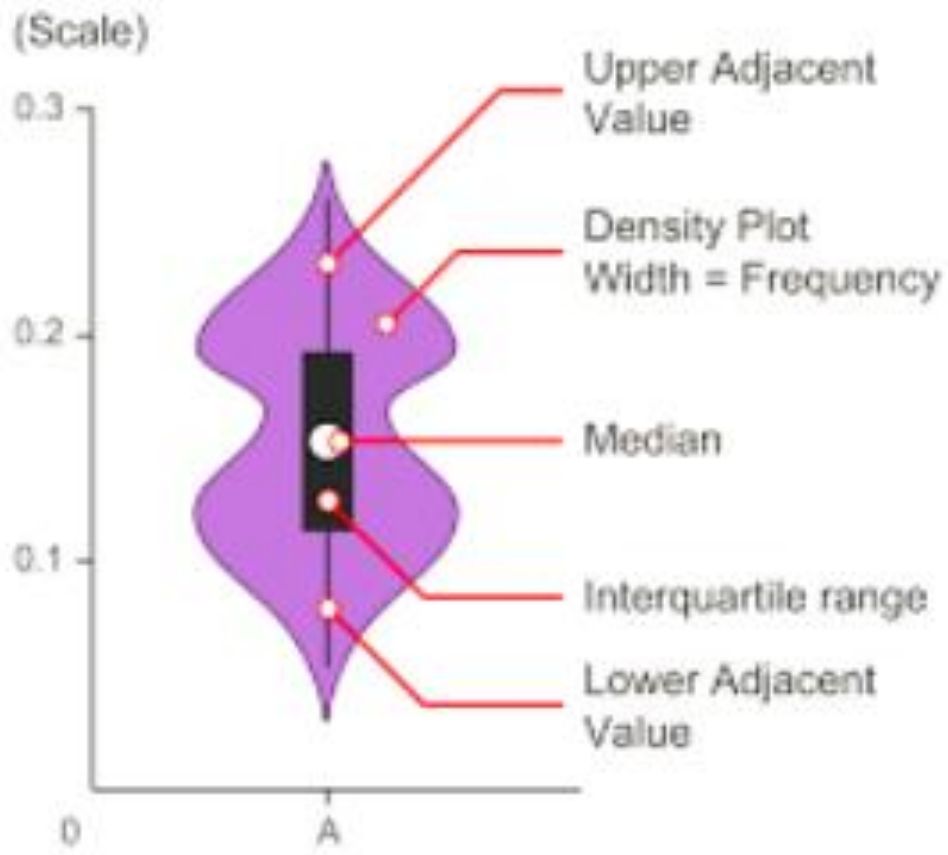


e) Violinplot

A violin plot plays a similar activity that is pursued through whisker or box plot do. As it shows several quantitative data across one or more categorical variables. It can be an effective and attractive way to show multiple data at several units. A “wide-form” Data Frame helps to maintain each numeric column which can be plotted on the graph. It is possible to use NumPy or Python objects, but pandas objects are preferable because the associated names will be used to annotate the axes.

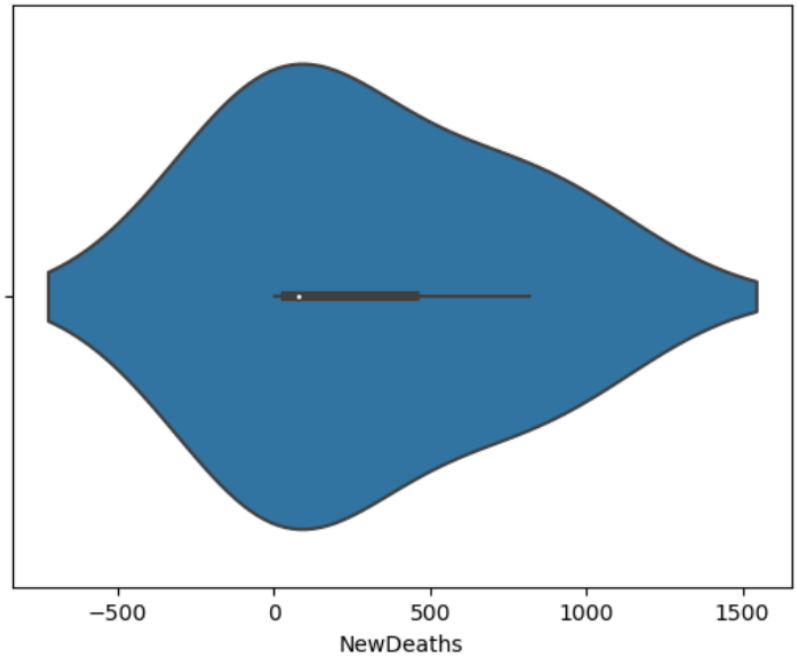
PANDASEABORN PROJECT

The Violin plot is very much similar to a box plot, with the addition of a rotated kernel density plot on each side. It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared.



5)Violin plot

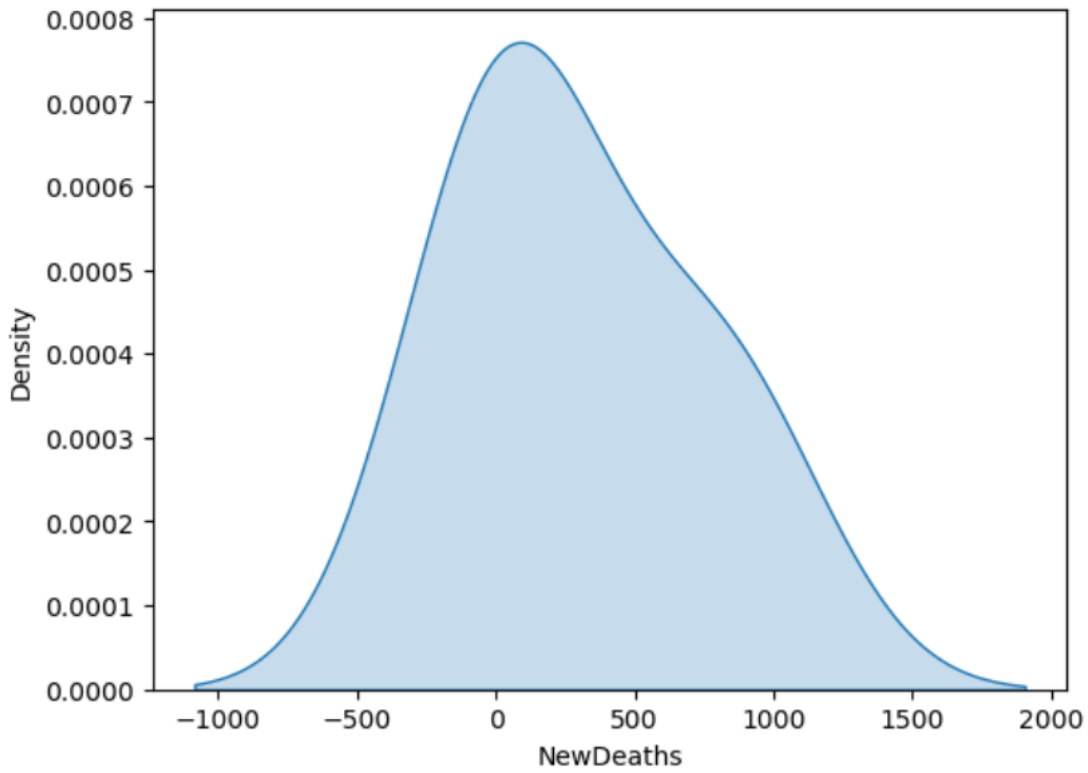
```
In [26]: sns.violinplot(data=df,x='NewDeaths')
Out[26]: <Axes: xlabel='NewDeaths'>
```



f) KDEplot

Kernel Density Estimate (KDE) Plot and Kdeplot allows us to estimate the probability density function of the continuous or non-parametric from our data set curve in one or more dimensions it means we can create plot a single graph for multiple samples which helps in more efficient data visualization.

```
In [27]: sns.kdeplot(data=df,x='NewDeaths',shade=True)
Out[27]: <Axes: xlabel='NewDeaths', ylabel='Density'>
```

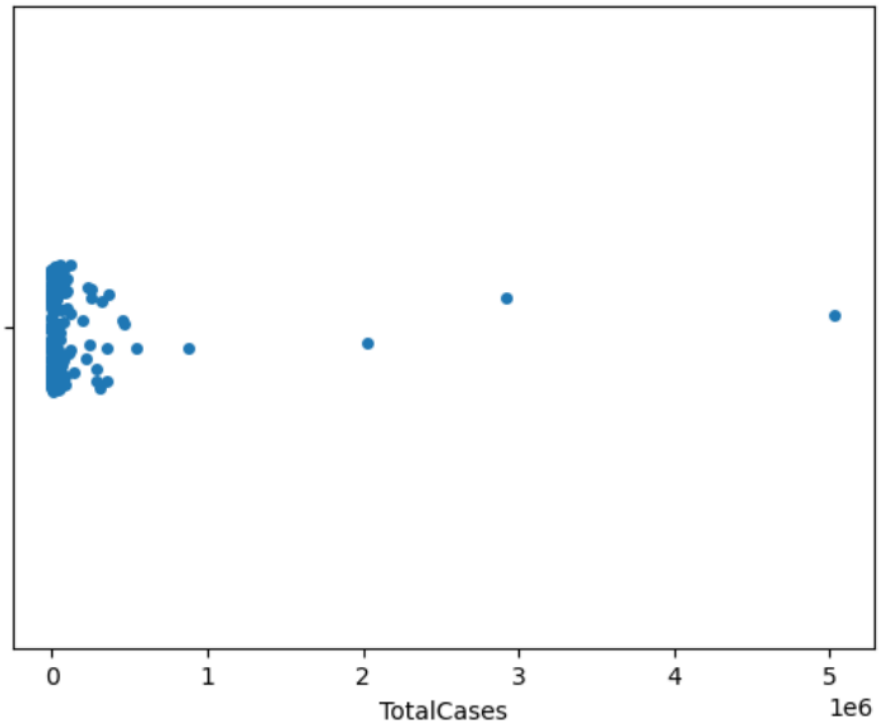


g) Stripplot

A strip plot is drawn on its own. It is a good complement to a boxplot or violinplot in cases where all observations are shown along with some representation of the underlying distribution. It is used to draw a scatter plot based on the category.

7)Strip plot

```
In [28]: sns.stripplot(data=df,x='TotalCases')
Out[28]: <Axes: xlabel='TotalCases'>
```



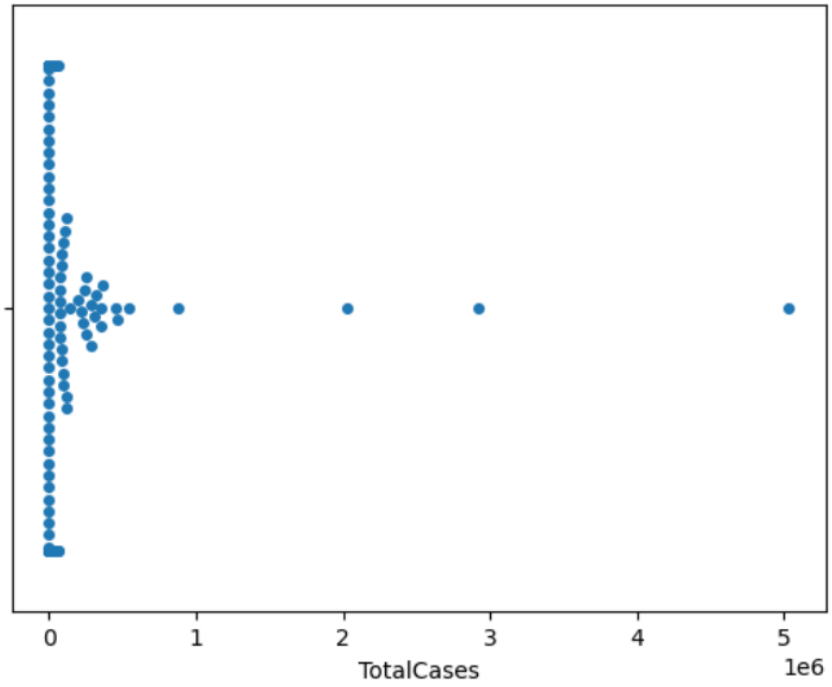
h)Swarmp plot

Seaborn swarmplot is probably similar to stripplot, only the points are adjusted so it won't get overlap to each other as it helps to represent the better representation of the distribution of values. A swarm plot can be drawn on its own, but it is also

a good complement to a box, preferable because the associated names will be used to annotate the axes. This type of plot sometimes known as “beeswarm”.

8)swarmplot

```
In [29]: sns.swarmplot(data=df,x='TotalCases')
Out[29]: <Axes: xlabel='TotalCases'>
```



PANDASEABORN PROJECT

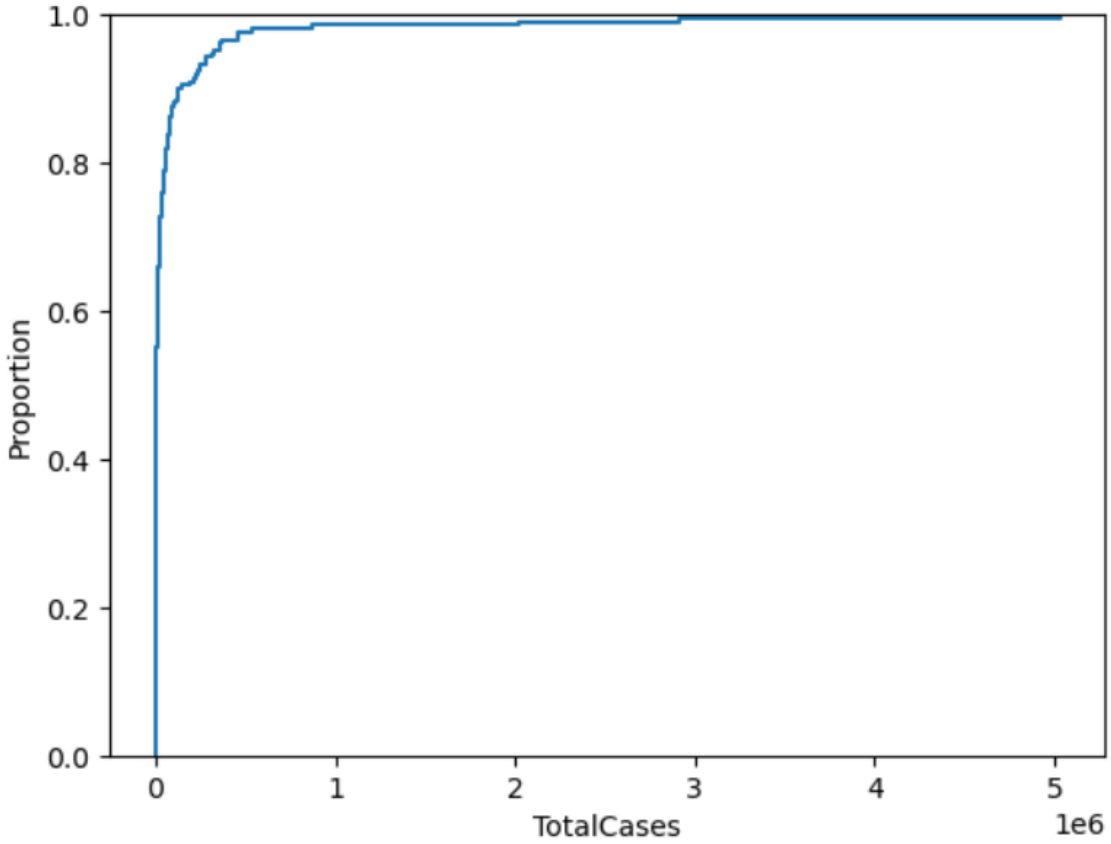
i) Ecdfplot

- ECDF stands for Empirical Commutative Distribution. It is more likely to use instead of the histogram for visualizing the data because the ECDF plot visualizes each and every data point of the dataset directly, which makes it easy for the user to interact with the plot.
- This plot contains more information because it has no bin size setting, which means it doesn't have any smoothing parameters.
- Since its curves are monotonically increasing, so it is well suited for comparing multiple distributions at the same time.
- In an ECDF plot, the x-axis corresponds to the range of values for the variable whereas the y-axis corresponds to the proportion of data points that are less than or equal to the corresponding value of the x-axis.
- We can make the ECDF plot directly by using ecdfplot() function, or we can also make the plot by using displot() function with the new Seaborn version.

9)ecdfplot

```
In [30]: sns.ecdfplot(data=df,x='TotalCases')
```

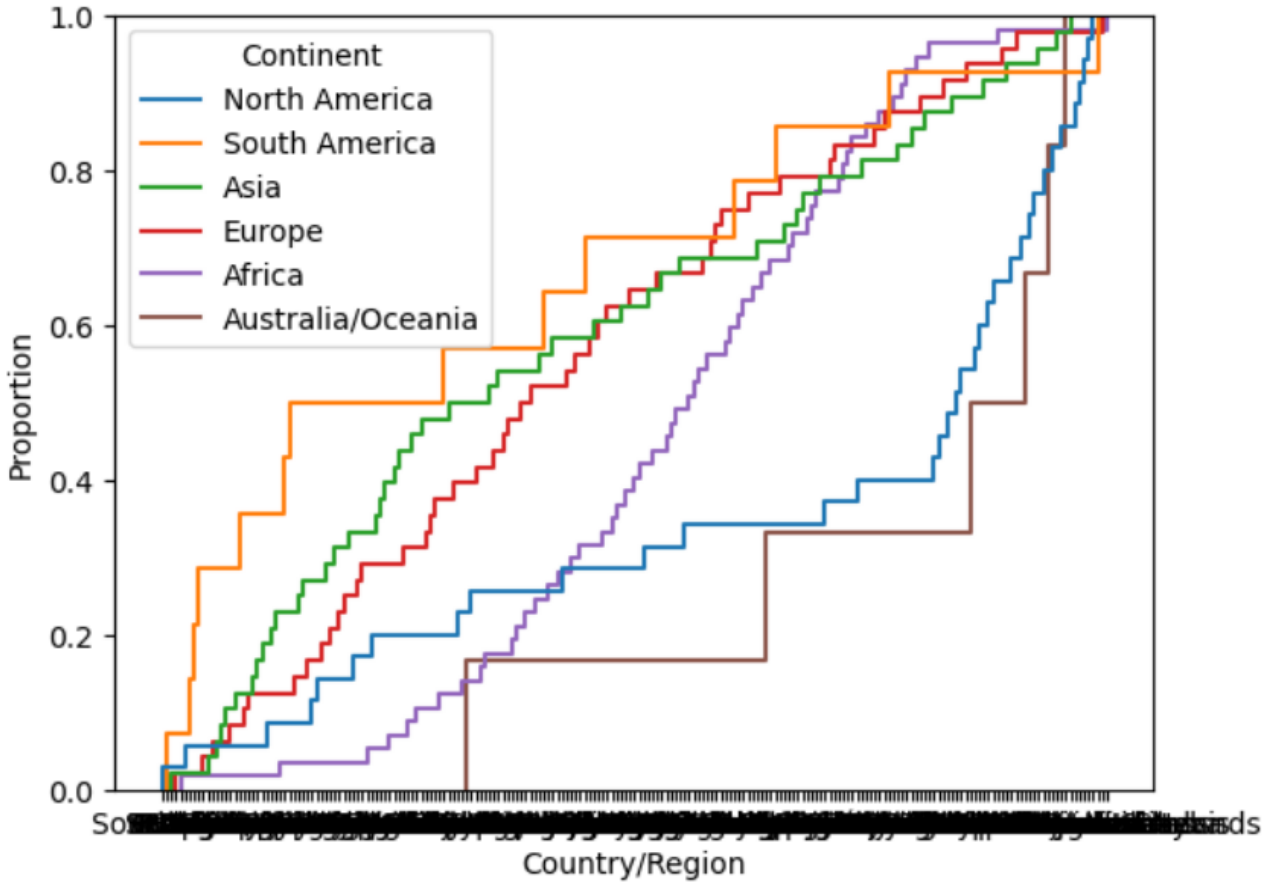
```
Out[30]: <Axes: xlabel='TotalCases', ylabel='Proportion'>
```



10. Using the grouping variable HUE

```
In [31]: sns.ecdfplot(data=df,x='Country/Region',hue='Continent')
```

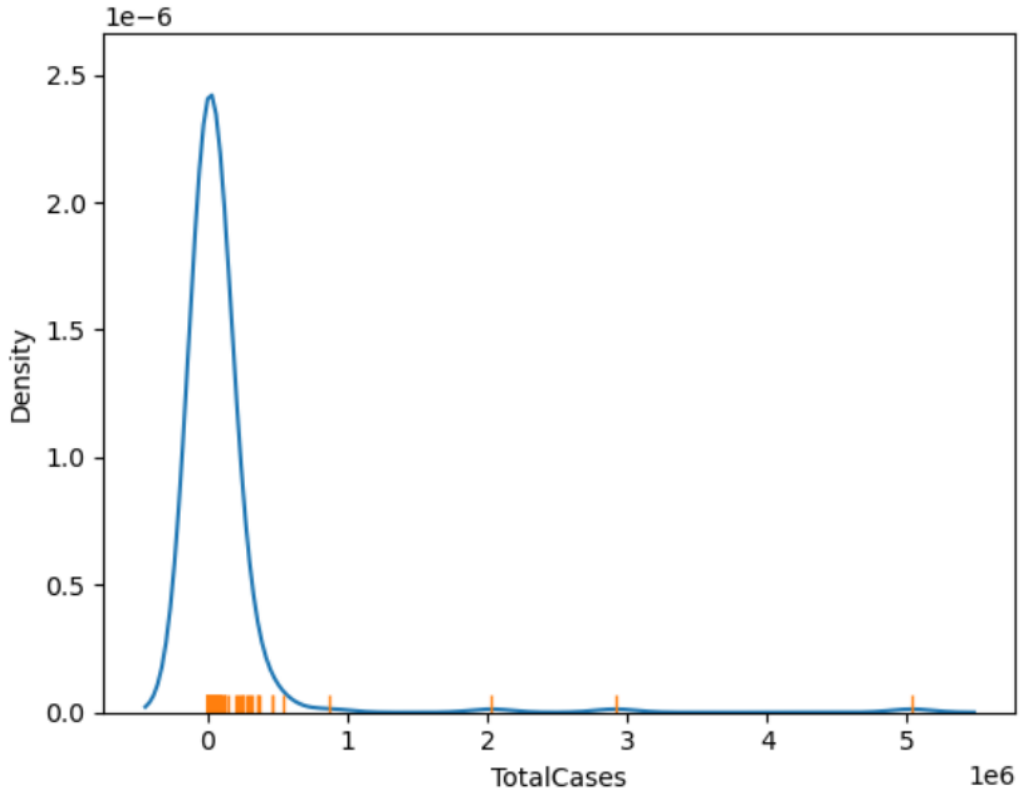
```
Out[31]: <Axes: xlabel='Country/Region', ylabel='Proportion'>
```



Combining two univariate plots

```
In [32]: sns.kdeplot(data=df,x='TotalCases')
         sns.rugplot(data=df,x='TotalCases')
```

```
Out[32]: <Axes: xlabel='TotalCases', ylabel='Density'>
```



2.Categorical

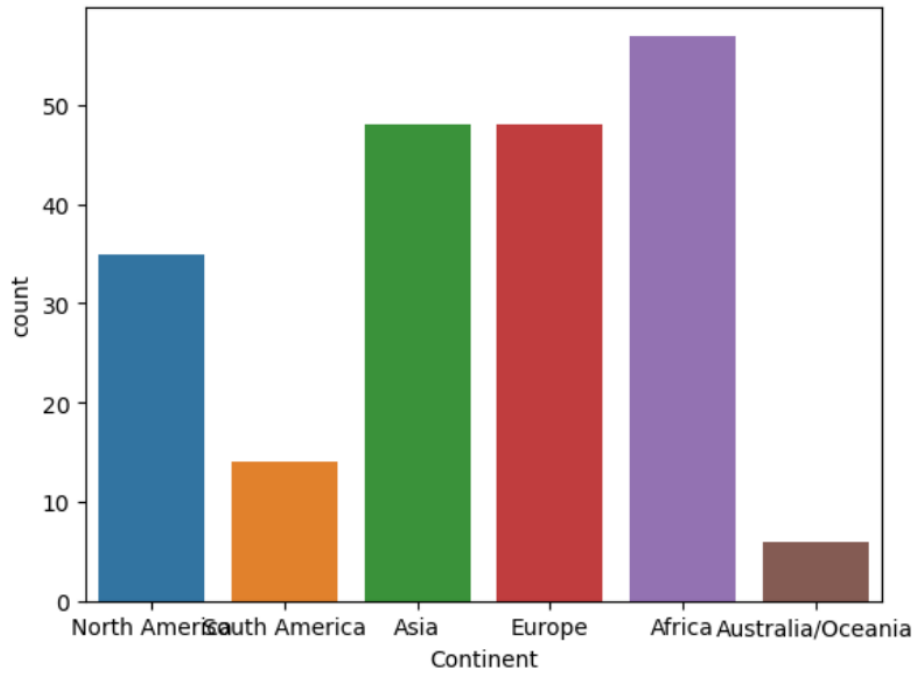
a)countplot

seaborn.countplot() method is used to Show the counts of observations in each categorical bin using bars.

12. Count plot

```
In [33]: sns.countplot(data=df,x='Continent')
```

```
Out[33]: <Axes: xlabel='Continent', ylabel='count'>
```



Plot Focusing on 2 variables (Bivariate Analysis)

1. Continuous

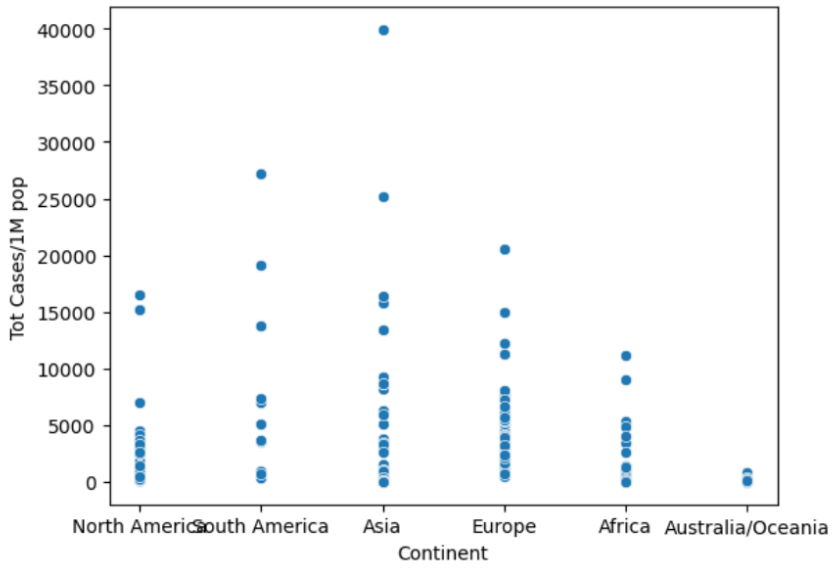
a) Scatter Plot

Scatterplot can be used with several semantic groupings which can help to understand well in a graph. They can plot two-dimensional graphics that can be enhanced by mapping up to three additional variables while using the semantics of hue, size, and style parameters. All the parameter control visual semantic which are used to identify the different subsets. Using redundant semantics can be helpful for making graphics more accessible.

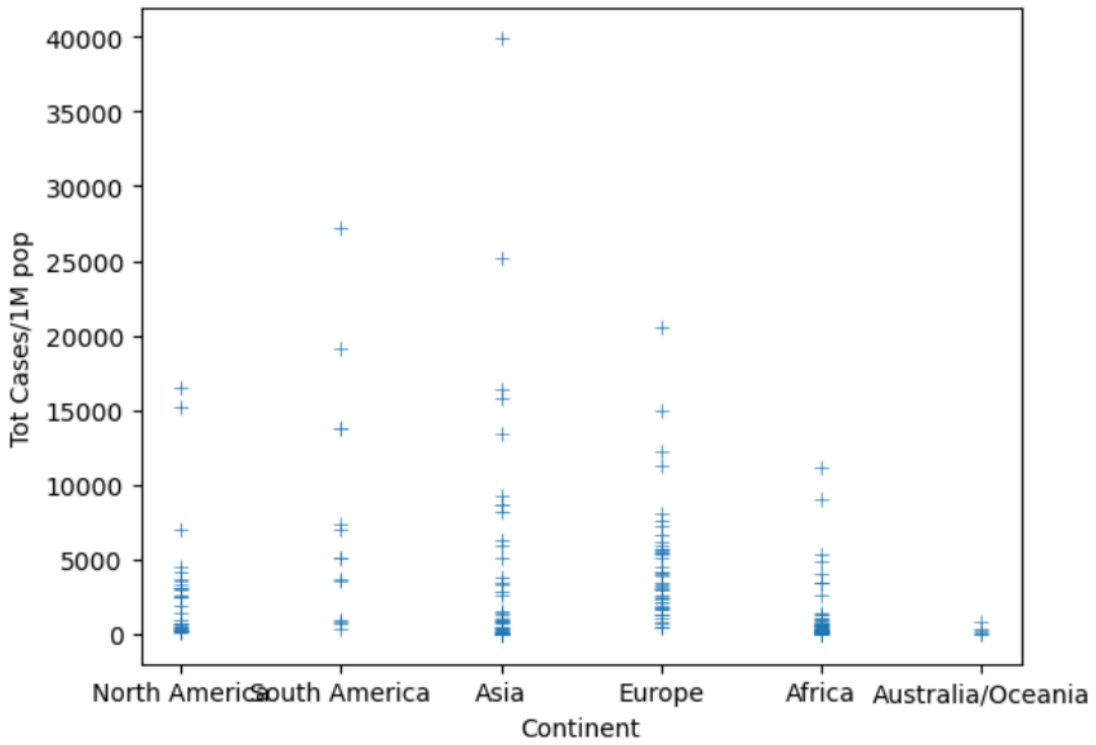
16. Scatter Plot

```
In [38]: sns.scatterplot(data=df,x='Continent',y='Tot Cases/1M pop')
```

```
Out[38]: <Axes: xlabel='Continent', ylabel='Tot Cases/1M pop'>
```



```
In [39]: sns.scatterplot(data=df,x='Continent',y='Tot Cases/1M pop',marker='+')
Out[39]: <Axes: xlabel='Continent', ylabel='Tot Cases/1M pop'>
```



Adding the marker attributes

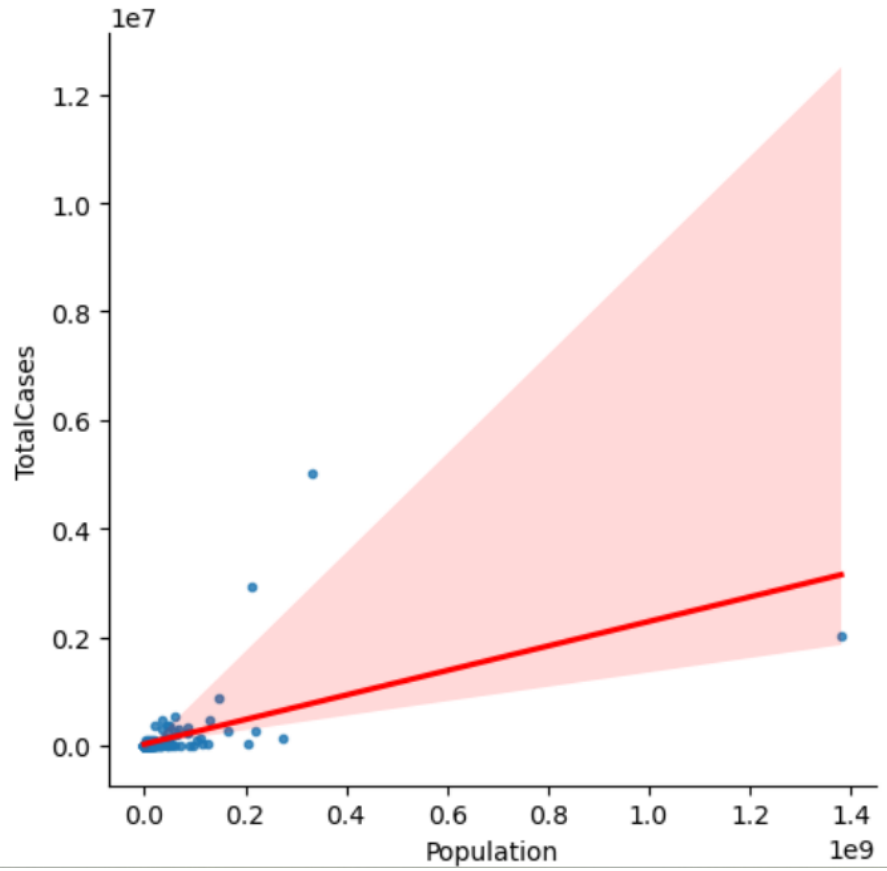
The circle is used to represent the data point and the default marker here is a blue circle. In the above output, we are seeing the default output for the marker, but we can customize this blue circle with **marker** attributes.

b) Lm Plot

seaborn.lmplot() method is used to draw a scatter plot onto a FacetGrid.

```
In [45]: sns.lmplot(data=df, x="Population", y="TotalCases", markers=".",line_kws={'color':'red'})
```

```
Out[45]: <seaborn.axisgrid.FacetGrid at 0x2a70df529b0>
```

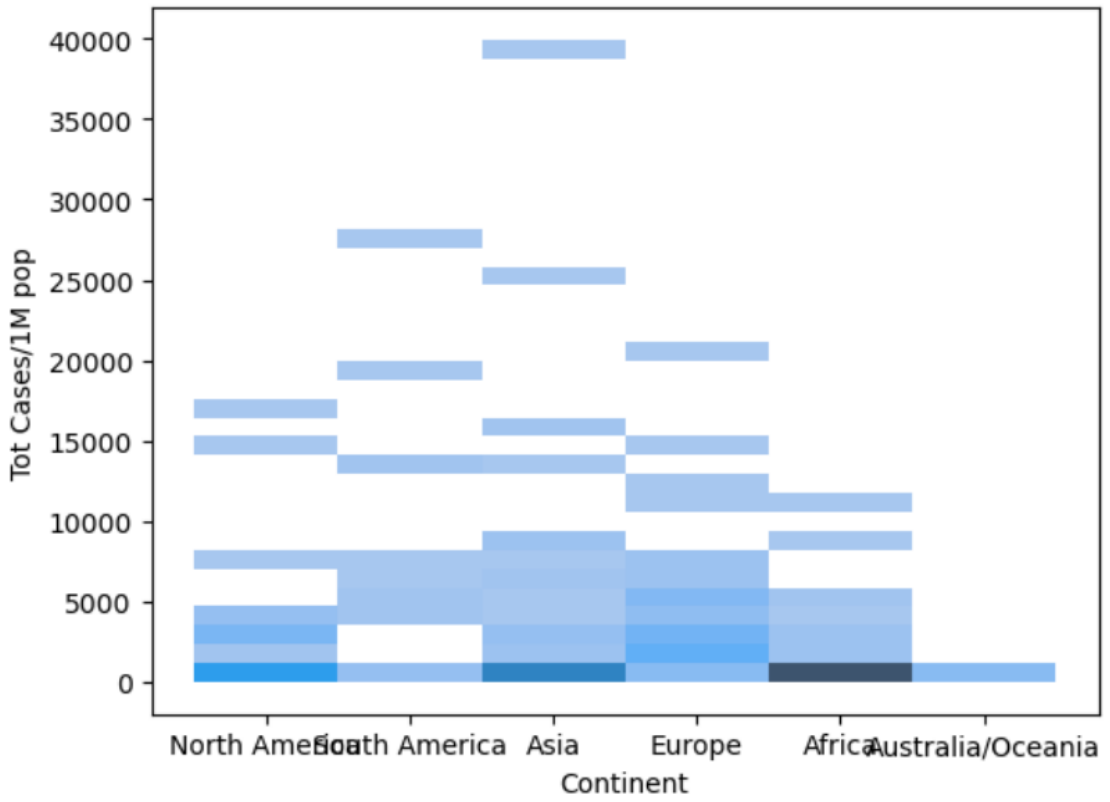


c)Hist Plot

18)Histplot

```
In [42]: sns.histplot(data=df,x='Continent',y='Tot Cases/1M pop')
```

```
Out[42]: <Axes: xlabel='Continent', ylabel='Tot Cases/1M pop'>
```

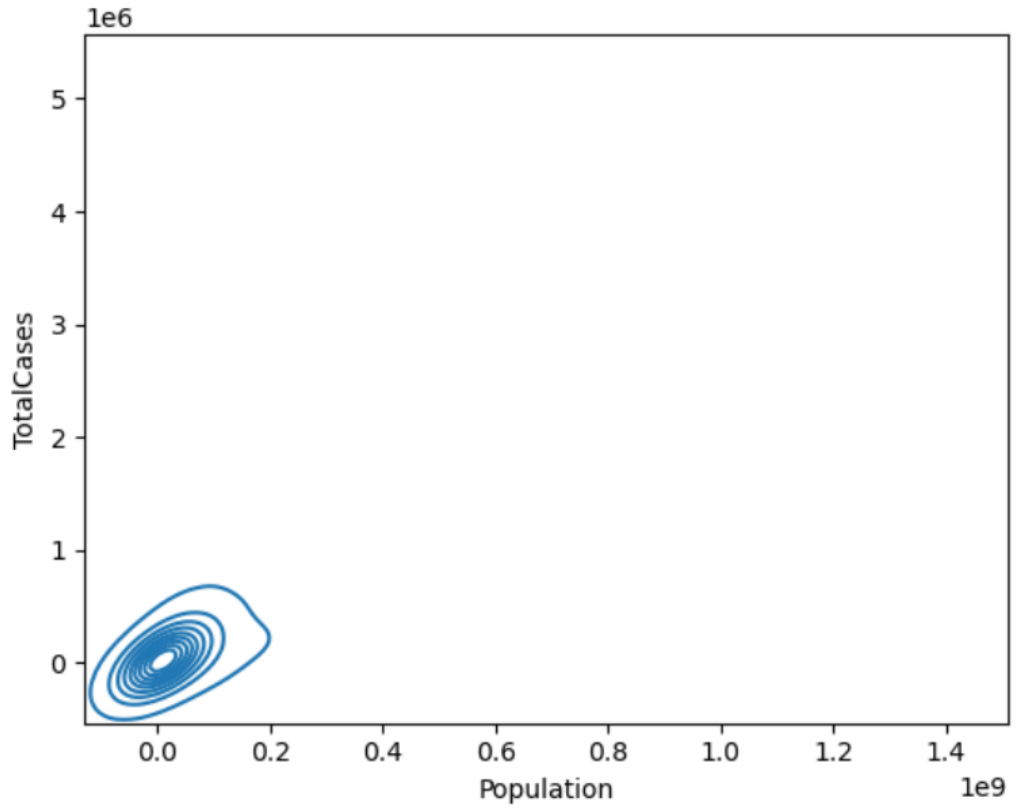


d)KDE plot

17)KDE plot

```
In [40]: sns.kdeplot(data=df,x='Population',y='TotalCases')
```

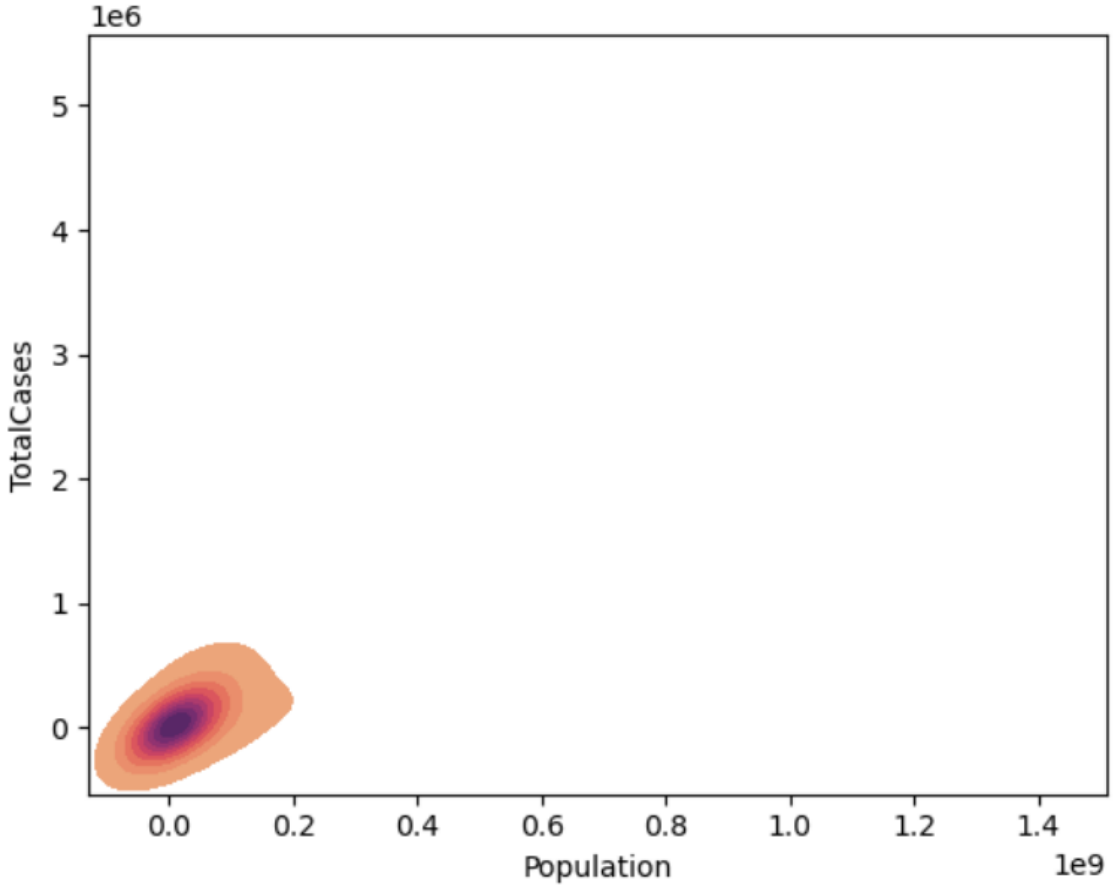
```
Out[40]: <Axes: xlabel='Population', ylabel='TotalCases'>
```



PANDASEABORN PROJECT

```
In [41]: sns.kdeplot(data=df,x='Population',y='TotalCases',fill=True,cmap='flare')
```

```
Out[41]: <Axes: xlabel='Population', ylabel='TotalCases'>
```

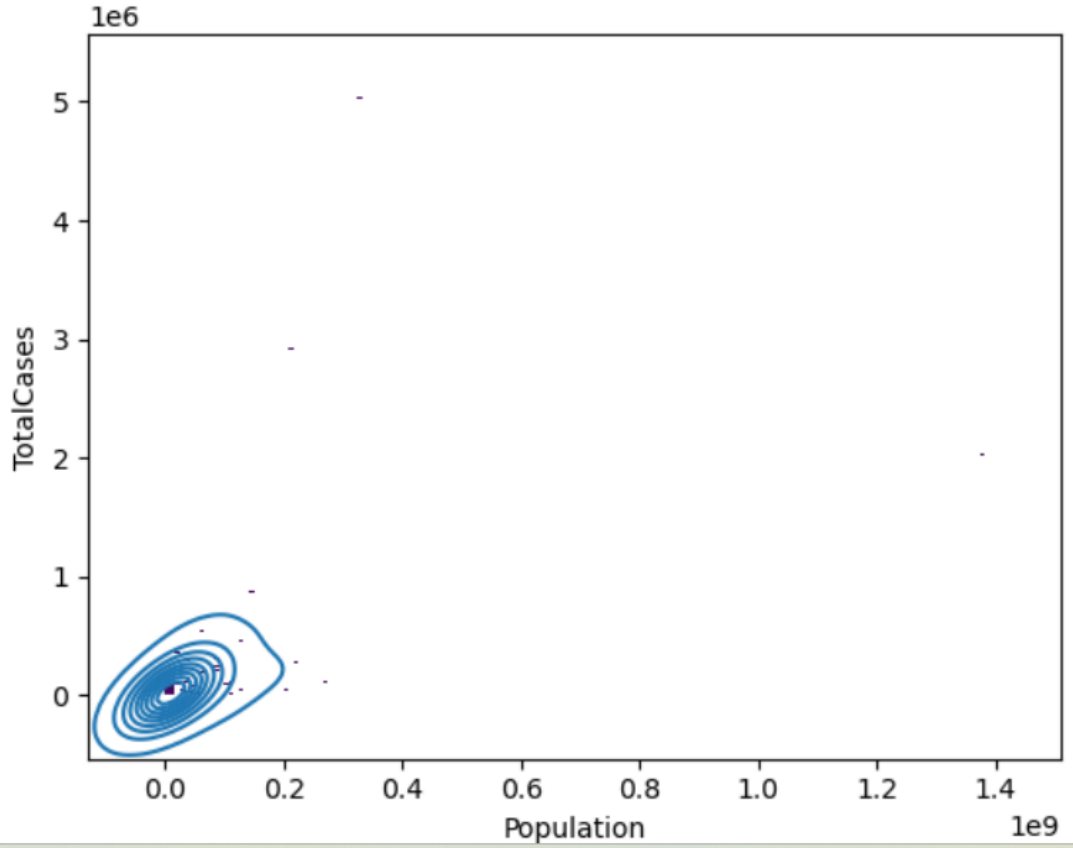


PANDASEABORN PROJECT

19. Combining two bivariate plots

In [43]:
```
sns.kdeplot(data=df,x='Population',y='TotalCases')
sns.histplot(data=df,x='Population',y='TotalCases',cmap='viridis')
```

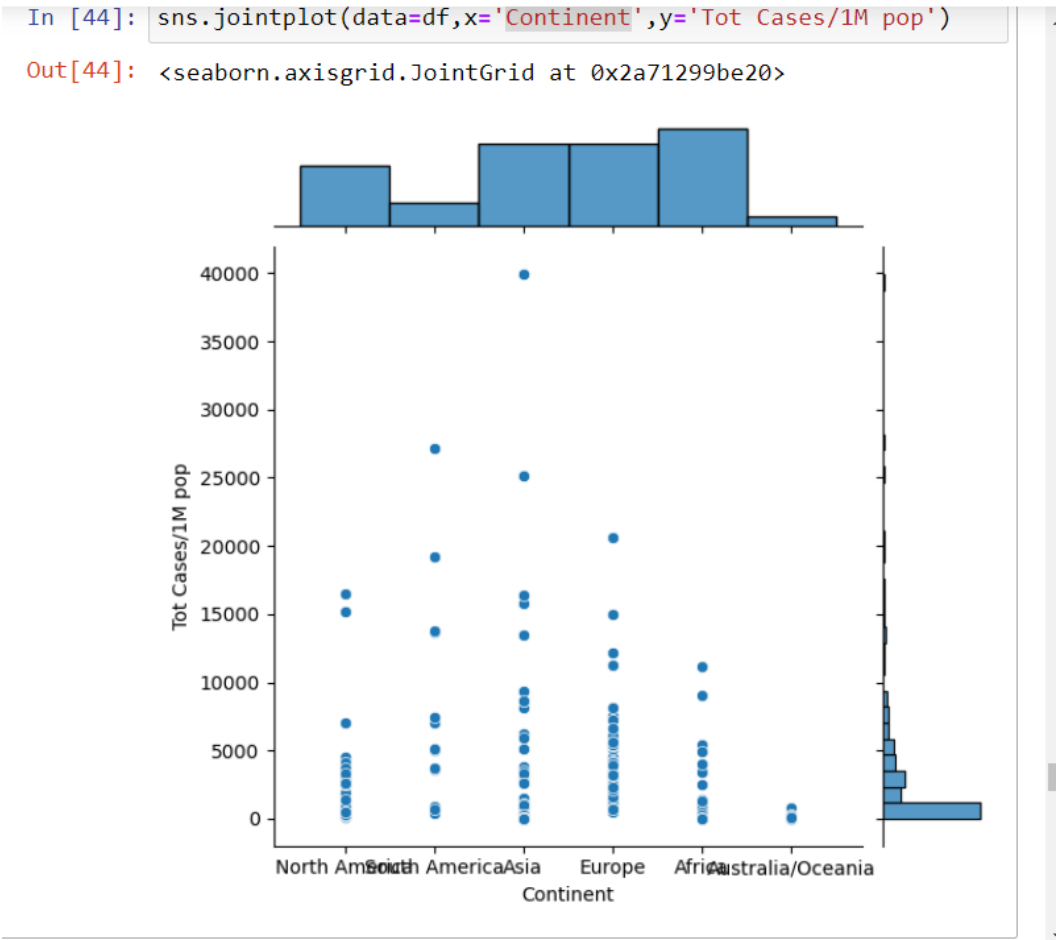
Out[43]:
```
<Axes: xlabel='Population', ylabel='TotalCases'>
```



e)Joint Plot

To draw a plot of two variables with bivariate and univariate graphs. This function provides a convenient interface to the 'JointGrid' class, with several canned plot kinds. This is intended to be a fairly lightweight wrapper; if you need more flexibility, you should use :class:'JointGrid' directly.

```
In [44]: sns.jointplot(data=df,x='Continent',y='Tot Cases/1M pop')
Out[44]: <seaborn.axisgrid.JointGrid at 0x2a71299be20>
```



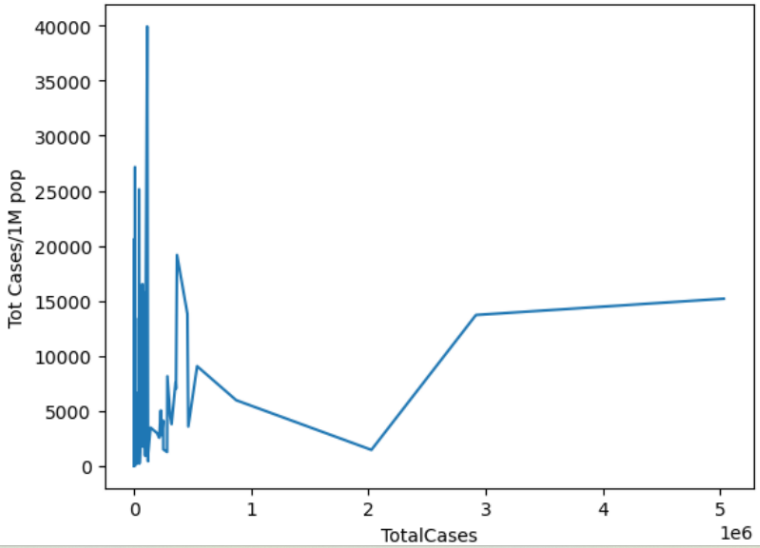
2. Categotical

a) Line Plot

The relationship between x and y can be shown for different subsets of the data using the hue, size, and style parameters. These parameters control what visual semantics are used to identify the different subsets. It is possible to show up to three dimensions independently by using all three semantic types, but this style of plot can be hard to interpret and is often ineffective. Using redundant semantics (i.e. both hue and style for the same variable) can be helpful for making graphics more accessible.

13. Line Plot

```
In [34]: sns.lineplot(data=df,x='TotalCases',y='Tot Cases/1M pop')
Out[34]: <Axes: xlabel='TotalCases', ylabel='Tot Cases/1M pop'>
```



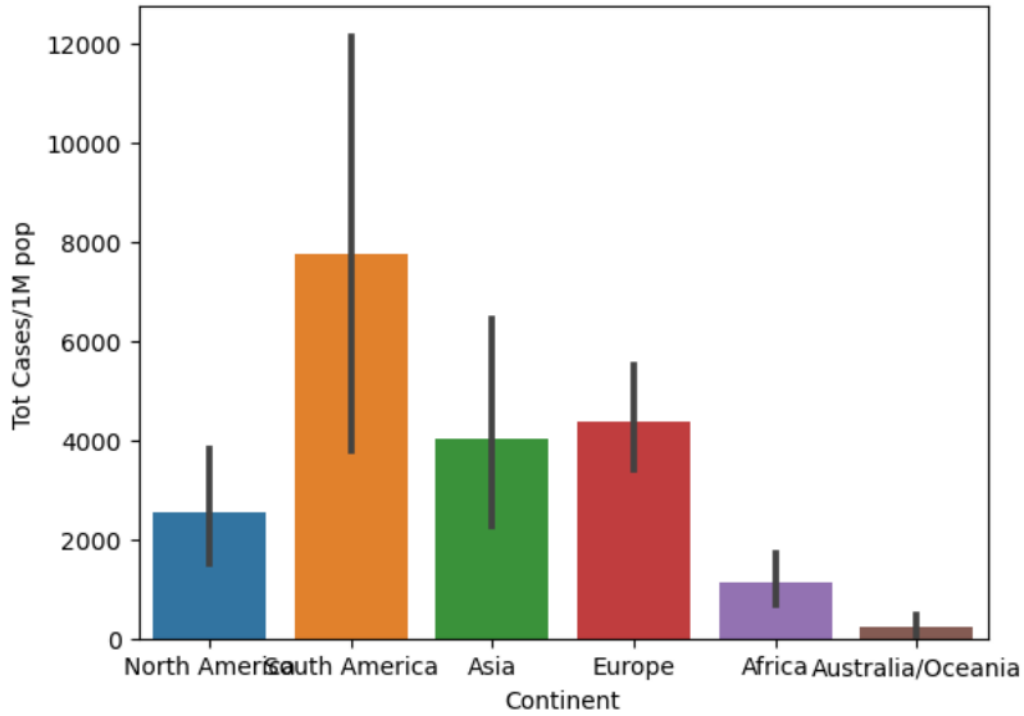
b) Bar plot

A barplot is basically 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.

15. Bar Plot

```
In [36]: sns.barplot(data=df,x='Continent',y='Tot Cases/1M pop')
```

```
Out[36]: <Axes: xlabel='Continent', ylabel='Tot Cases/1M pop'>
```



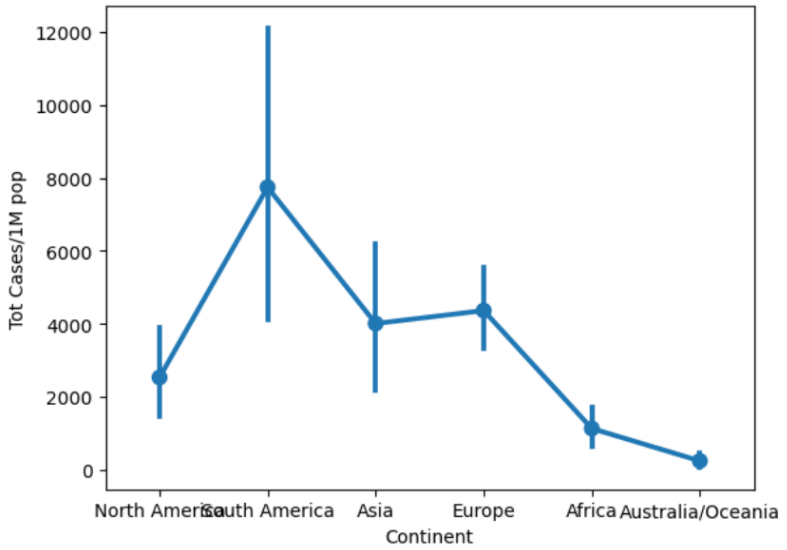
c) Point Plot

This method is used to show point estimates and confidence intervals using scatter plot glyphs. A point plot represents an estimate of central tendency for a numeric variable by the position of scatter plot points and provides some indication of the uncertainty around that estimate using error bars.

This function always treats one of the variables as categorical and draws data at ordinal positions (0, 1, … n) on the relevant axis, even when the data has a numeric or date type.

14. Point Plot

```
In [35]: sns.pointplot(data=df,x='Continent',y='Tot Cases/1M pop')
Out[35]: <Axes: xlabel='Continent', ylabel='Tot Cases/1M pop'>
```



PANDASEABORN PROJECT

Relationship plots

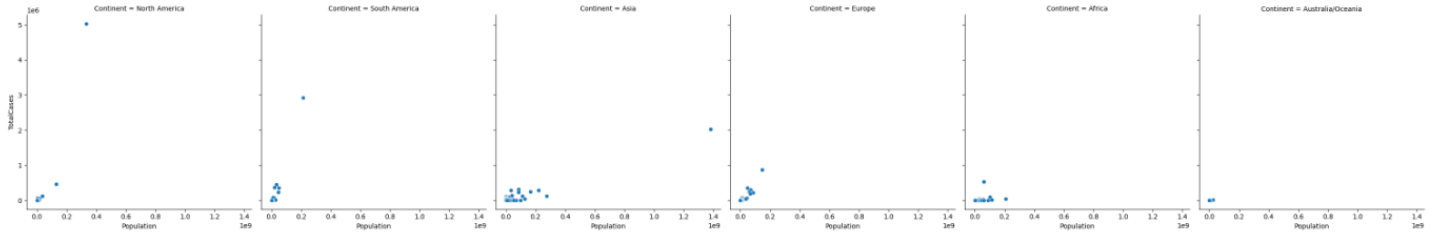
a) Relplot

Relational plots are used for visualizing the statistical relationship between the data points. Visualization is necessary because it allows the human to see trends and patterns in the data. The process of understanding how the variables in the dataset relate each other and their relationships are termed as Statistical analysis. Seaborn, unlike to matplotlib, also provides some default datasets. In this article, we will be using a default dataset named 'tips'. This dataset gives information about people who had food at some restaurant and whether they left tip for waiters or not, their gender and whether they do smoke or not, and more.

22. Relplot

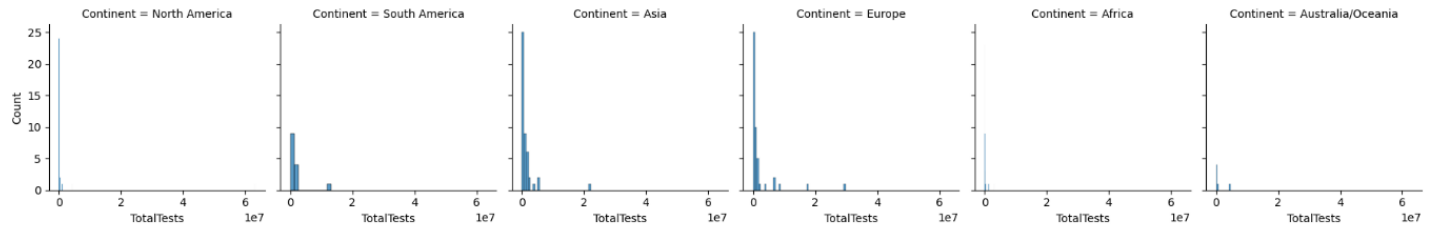
```
In [46]: sns.relplot(data=df, x="Population", y="TotalCases",col="Continent")
```

```
Out[46]: <seaborn.axisgrid.FacetGrid at 0x2a713283490>
```



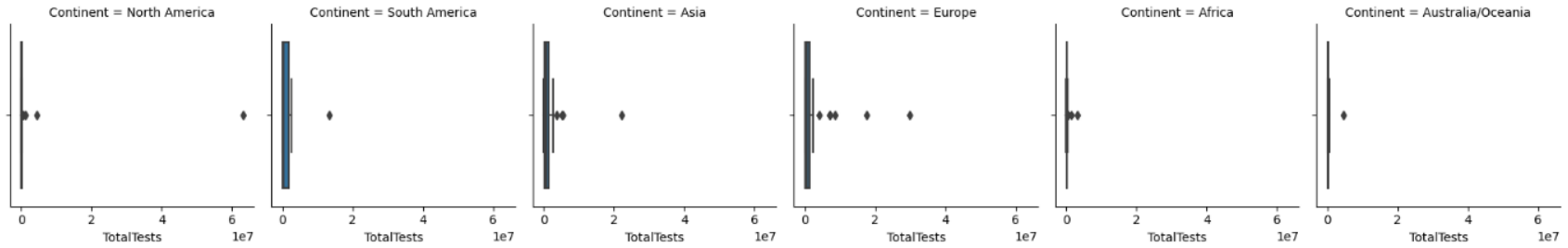

```
In [47]: g = sns.FacetGrid(data=df, col="Continent")
         g.map_dataframe(sns.histplot, x="TotalTests")
```

```
Out[47]: <seaborn.axisgrid.FacetGrid at 0x2a713afcfa0>
```



In [48]:
```
g = sns.FacetGrid(data=df, col="Continent")
g.map_dataframe(sns.boxplot, x="TotalTests")
```

Out[48]: <seaborn.axisgrid.FacetGrid at 0x2a715df4880>

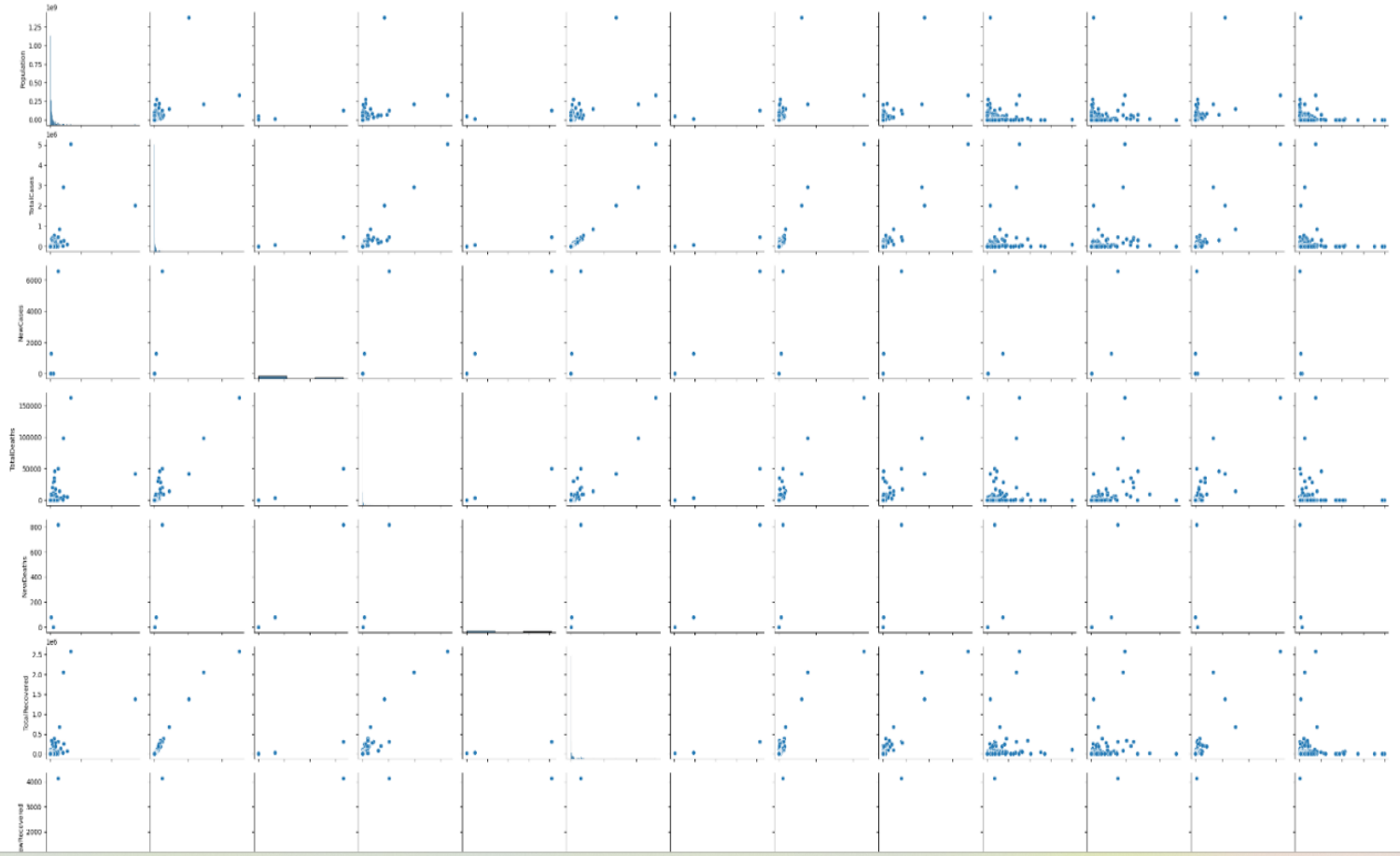


b) pair plot

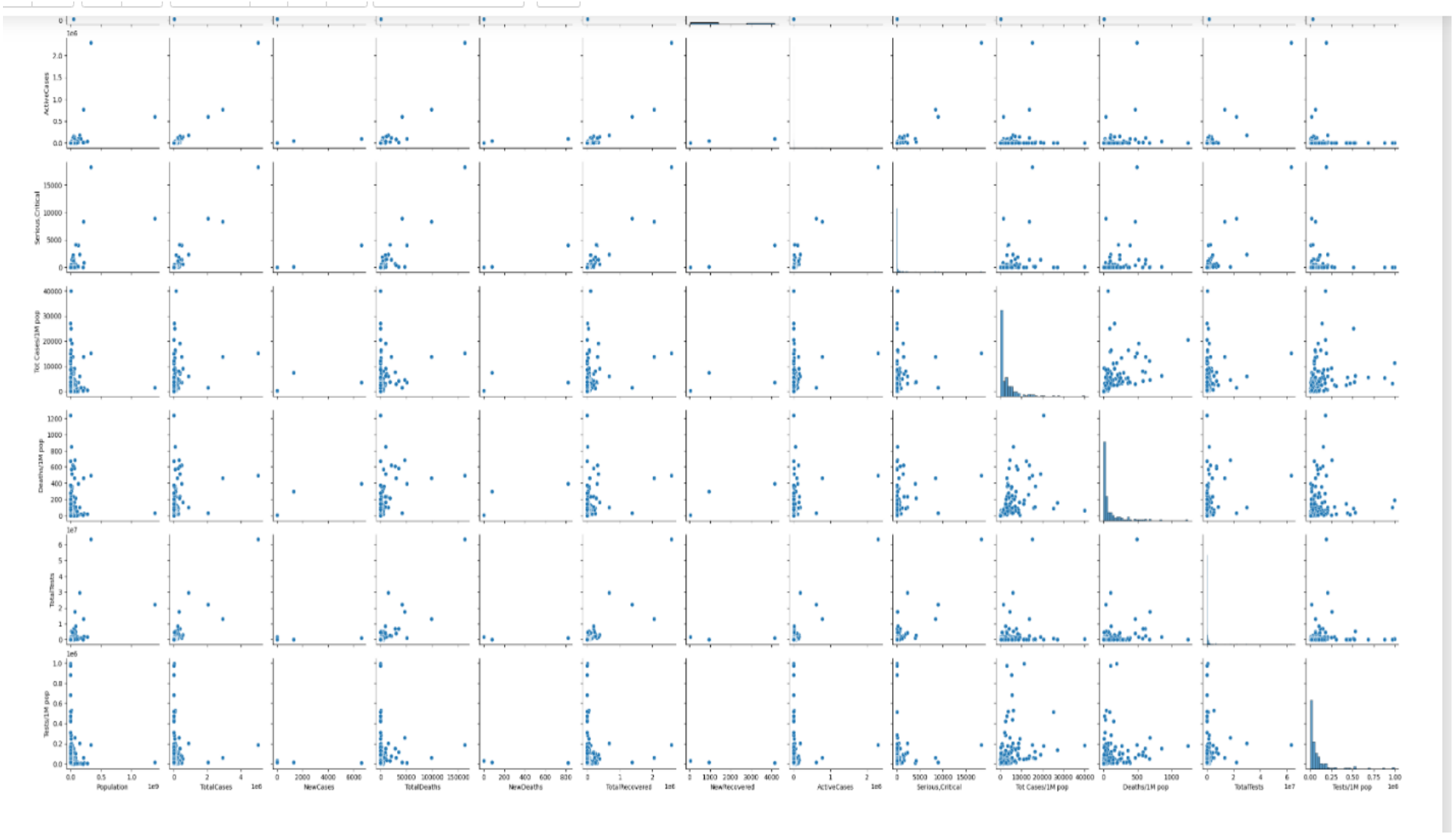
PANDASEABORN PROJECT

```
In [49]: sns.pairplot(df)
```

```
Out[49]: <seaborn.axisgrid.PairGrid at 0x2a7164ac460>
```



PANDASEABORN PROJECT

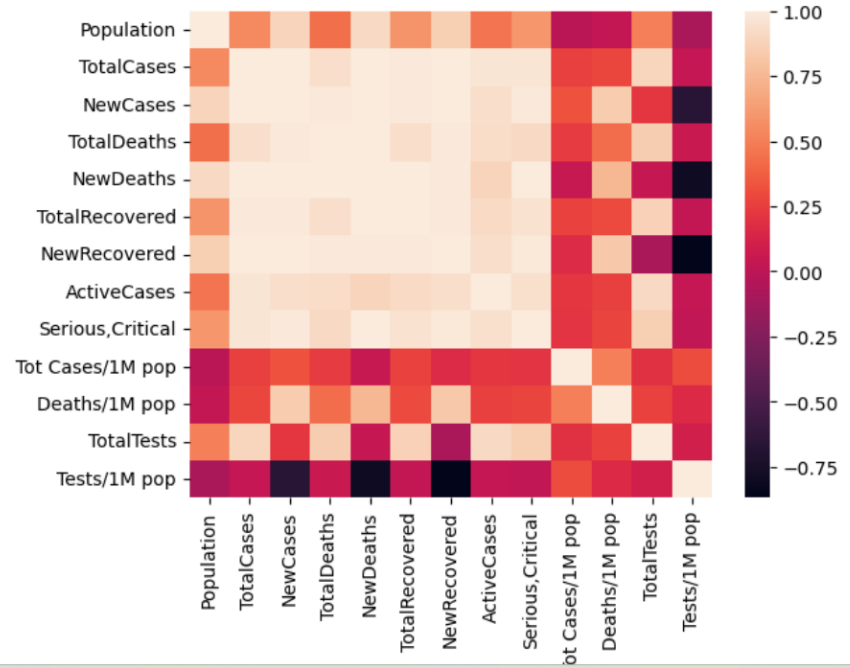


Correlations

Heatmap

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. In this, to represent more common values or higher activities brighter colors basically reddish colors are used and to represent less common or activity values, darker colors are preferred. Heatmap is also defined by the name of the shading matrix. Heatmaps in Seaborn can be plotted by using the seaborn.heatmap() function.

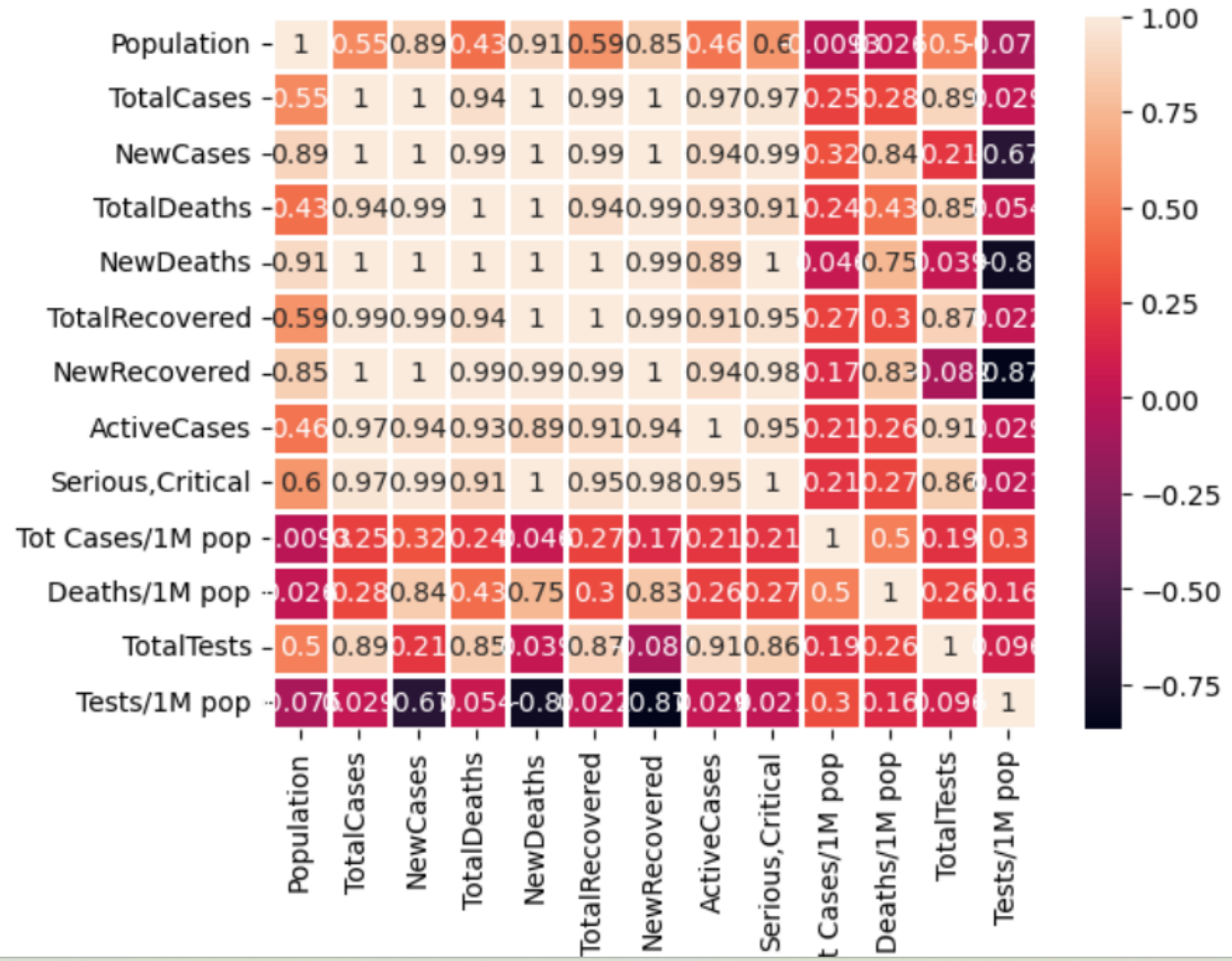
```
In [50]: sns.heatmap(df.corr())
Out[50]: <Axes: >
```



PANDASEABORN PROJECT

```
In [51]: sns.heatmap(df.corr(),annot=True,linewidths=1)
```

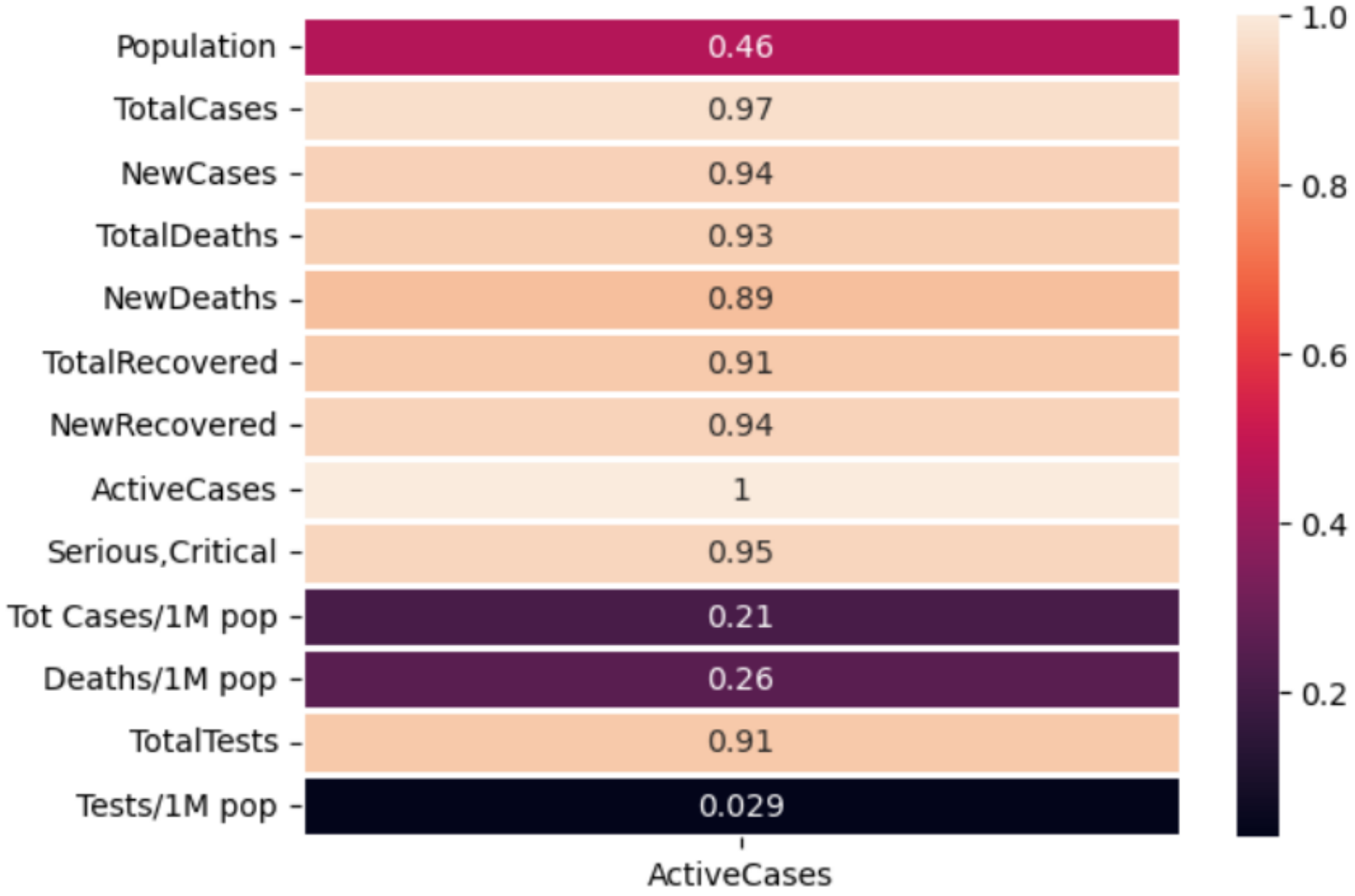
```
Out[51]: <Axes: >
```



PANDASEABORN PROJECT

```
In [56]: sns.heatmap(df.corr()[['ActiveCases']],annot=True,linewidths=1)
```

```
Out[56]: <Axes: >
```



PANDASEABORN PROJECT

Q.1) Analyze country-wise information, representation of data of top 20 countries.

We obtain information of the countries in terms of total cases, active-cases, recovered cases and death cases. We plot this information using treemap and pie charts.

The librarirs used are

import plotly

import plotly.express as px

from plotly.subplots import make_subplots

import plotly.graph_objects as go

The Plotly Python library is an interactive open-source library. This can be a very helpful tool for data visualization and understanding the data simply and easily. plotly graph objects are a high-level interface to plotly which are easy to use. It can plot various types of graphs and charts like scatter plots, line charts, bar charts, box plots, histograms, pie charts, etc.

Plotly has hover tool capabilities that allow us to detect any outliers or anomalies in a large number of data points.

It is visually attractive that can be accepted by a wide range of audiences.

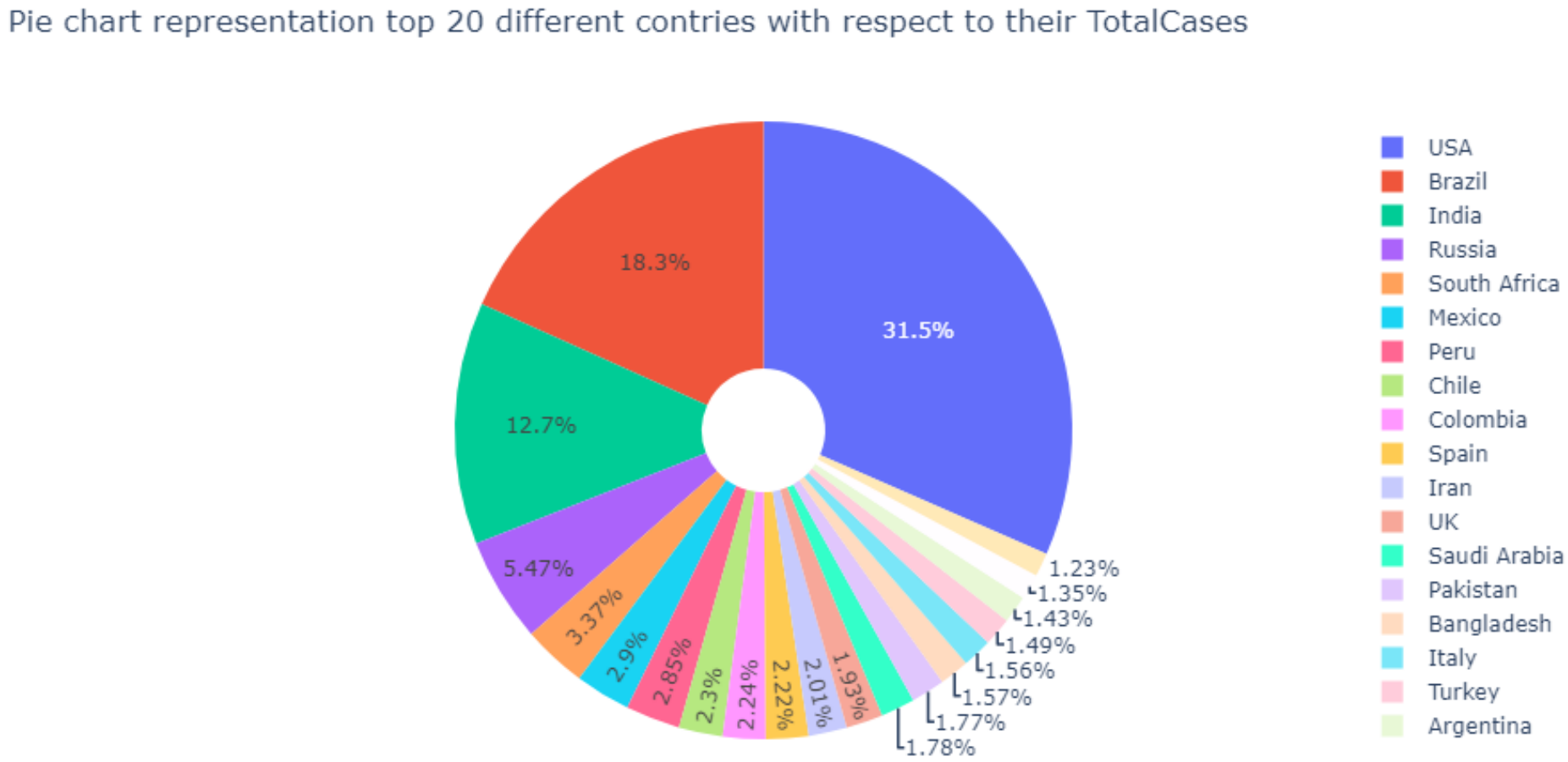
It allows us for the endless customization of our graphs that makes our plot more meaningful and understandable for others.

In [64]:
```
# representation of data of top 20 countries
cases = ['TotalCases','TotalDeaths','TotalRecovered', 'ActiveCases']
labels = df[0:20]['Country/Region'].values
top_20_total_cases = df.iloc[0:20]

#treemap and pie chart
for i in range(len(cases)):
    fig1 = px.treemap(top_20_total_cases, values = cases[i], path = ['Country/Region'],
                      title = 'Treemap representation different contries with respect to their {}'.format(cases[i]))
    fig1.show()
    fig2 = px.pie(top_20_total_cases, values = cases[i], names=labels, hole = 0.2,
                  title = "Pie chart representation top 20 different contries with respect to their {}".format(cases[i]))

    fig2.show()
```

PANDASEABORN PROJECT

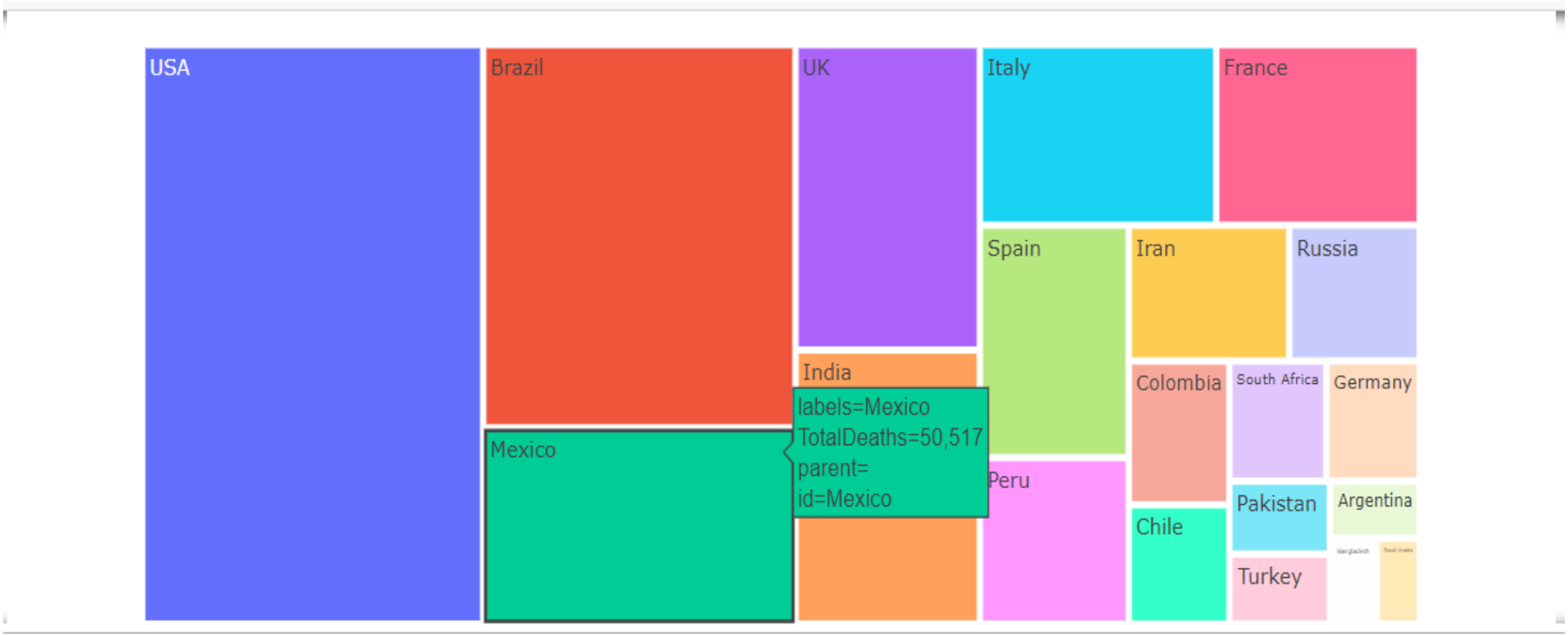
a)Treemap representation of different countries with replect to their TotalCases.

b) Pie chart representation top 20 different countries with respect to their TotalCases

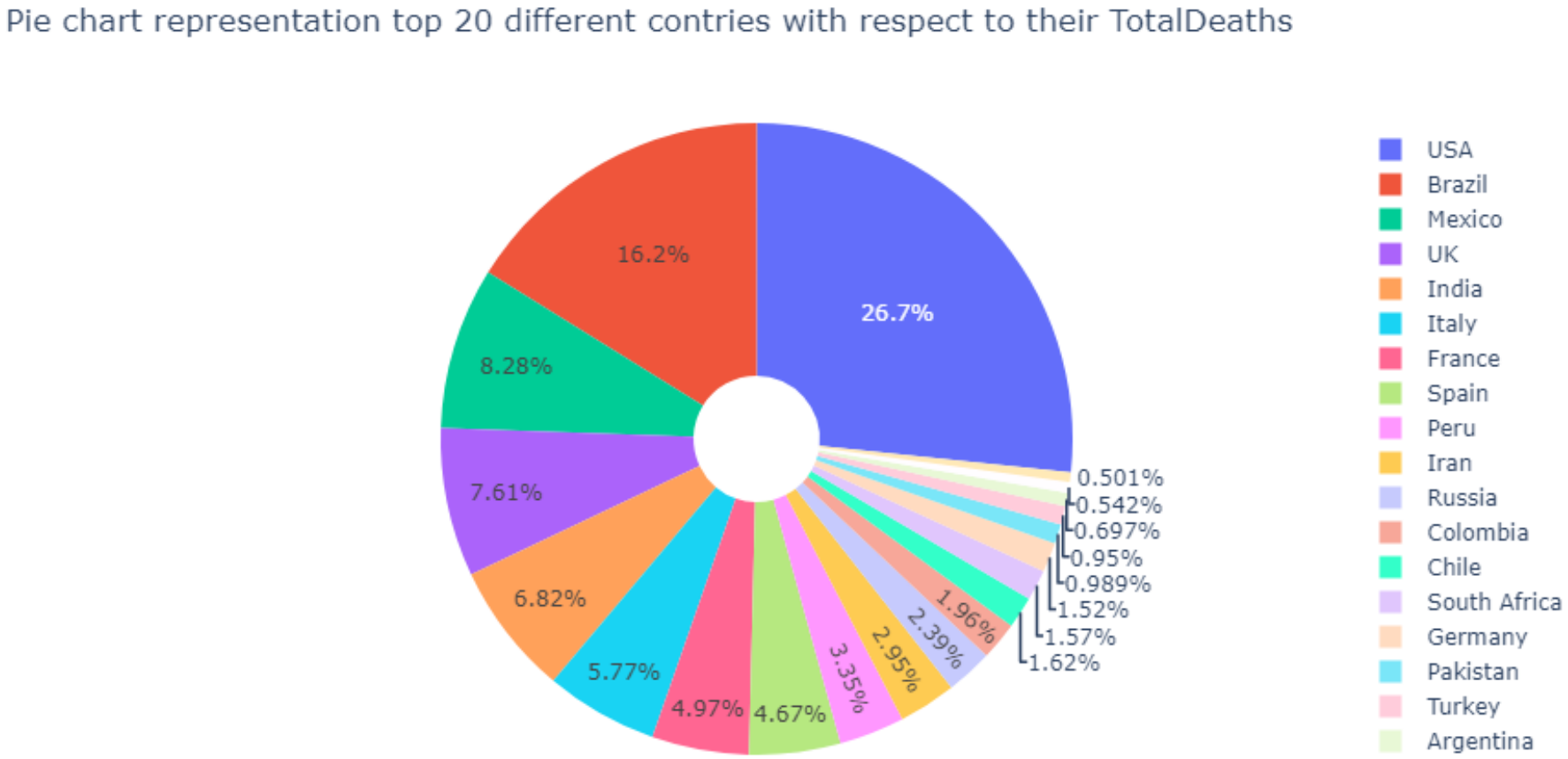


PANDASEABORN PROJECT

c) Treemap representation of different countries with replect to their TotalDeaths.

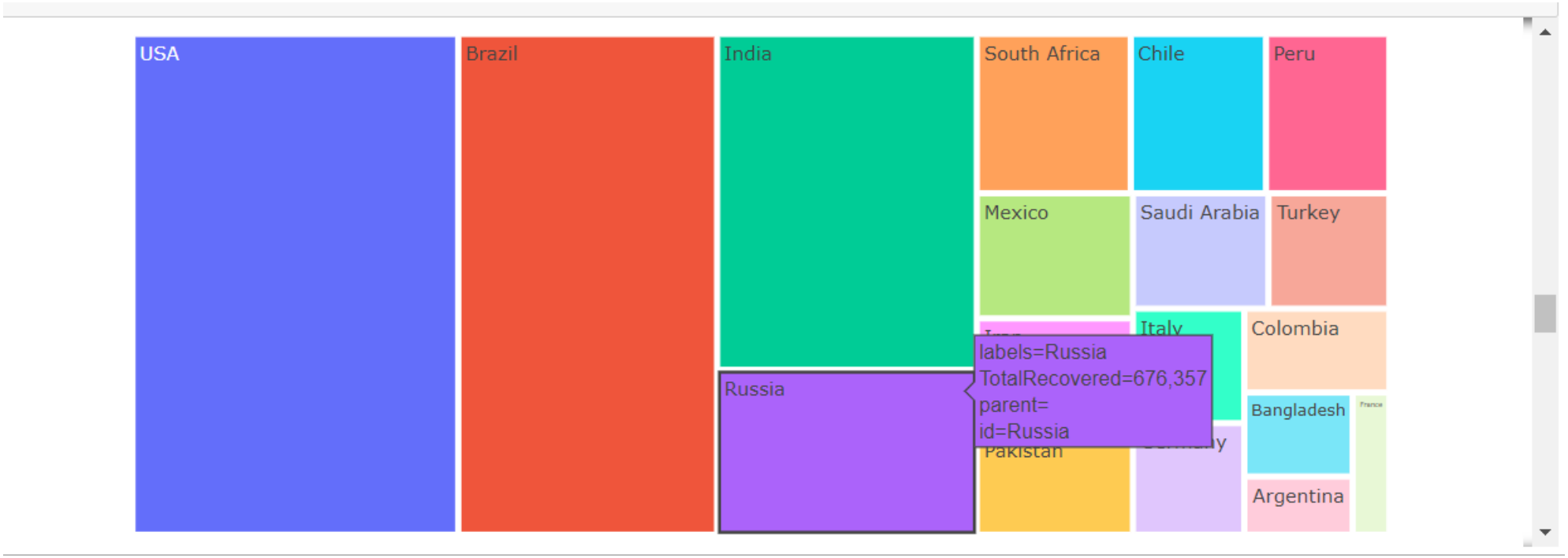


d)

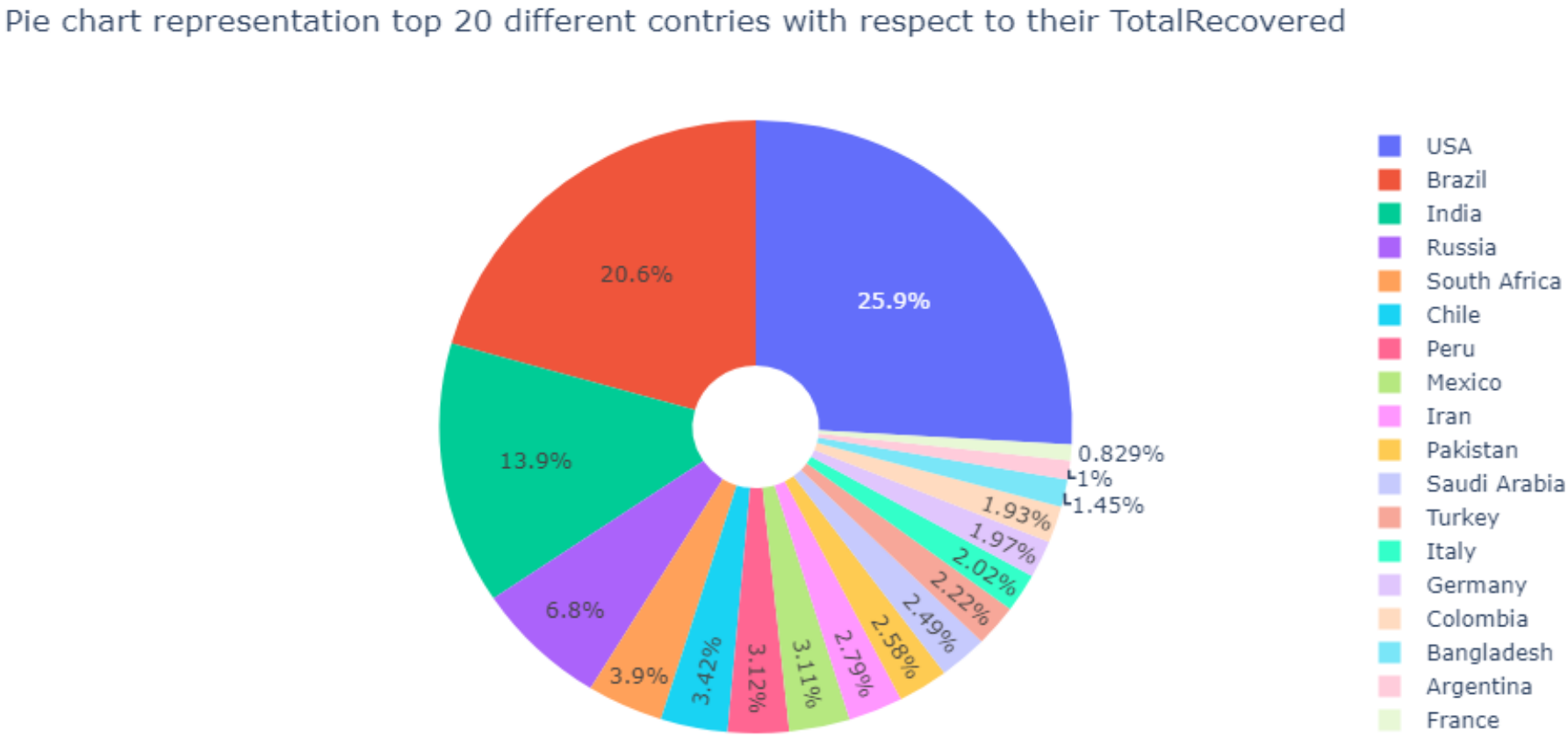

Pie chart representation top 20 different contries with respect to their TotalDeaths
26.7%
16.2%
8.28%
7.61%
6.82%
5.77%
4.97%
4.67%
3.35%
2.95%
2.39%
1.96%
1.62%
1.57%
1.52%
0.989%
0.95%
0.697%
0.542%
0.501%
USA
Brazil
Mexico
UK
India
Italy
France
Spain
Peru
Iran
Russia
Colombia
Chile
South Africa
Germany
Pakistan
Turkey
Argentina

PANDASEABORN PROJECT

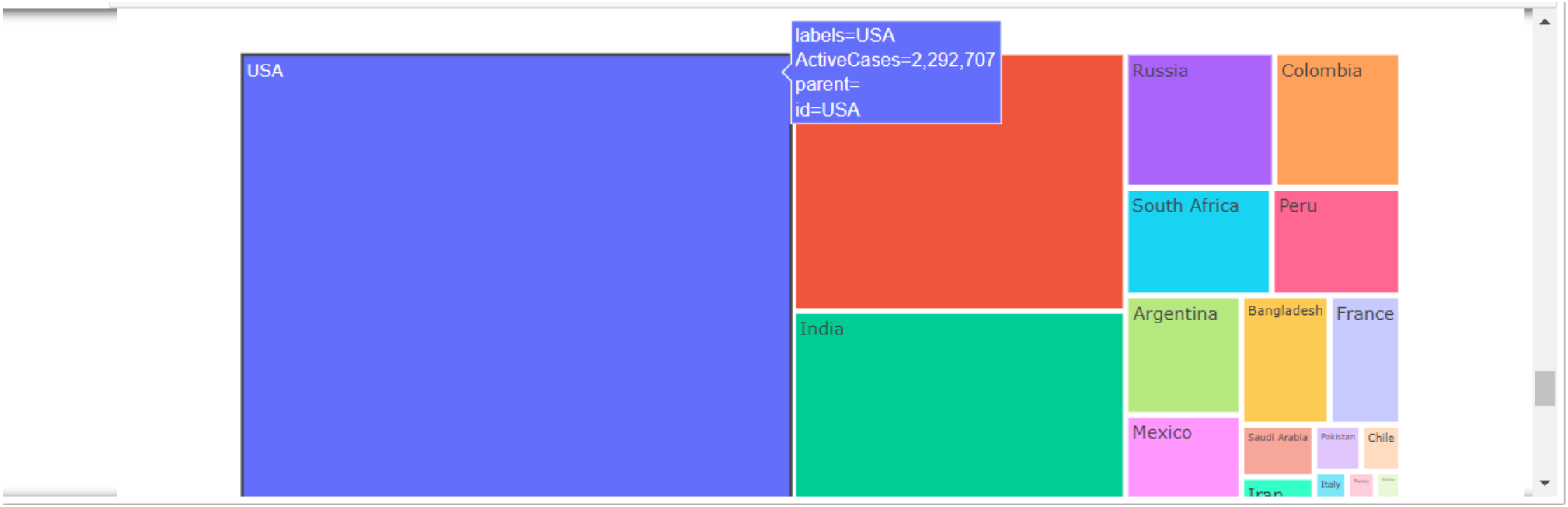
e) Treemap representation of different countries with replect to their TotalRecovered.



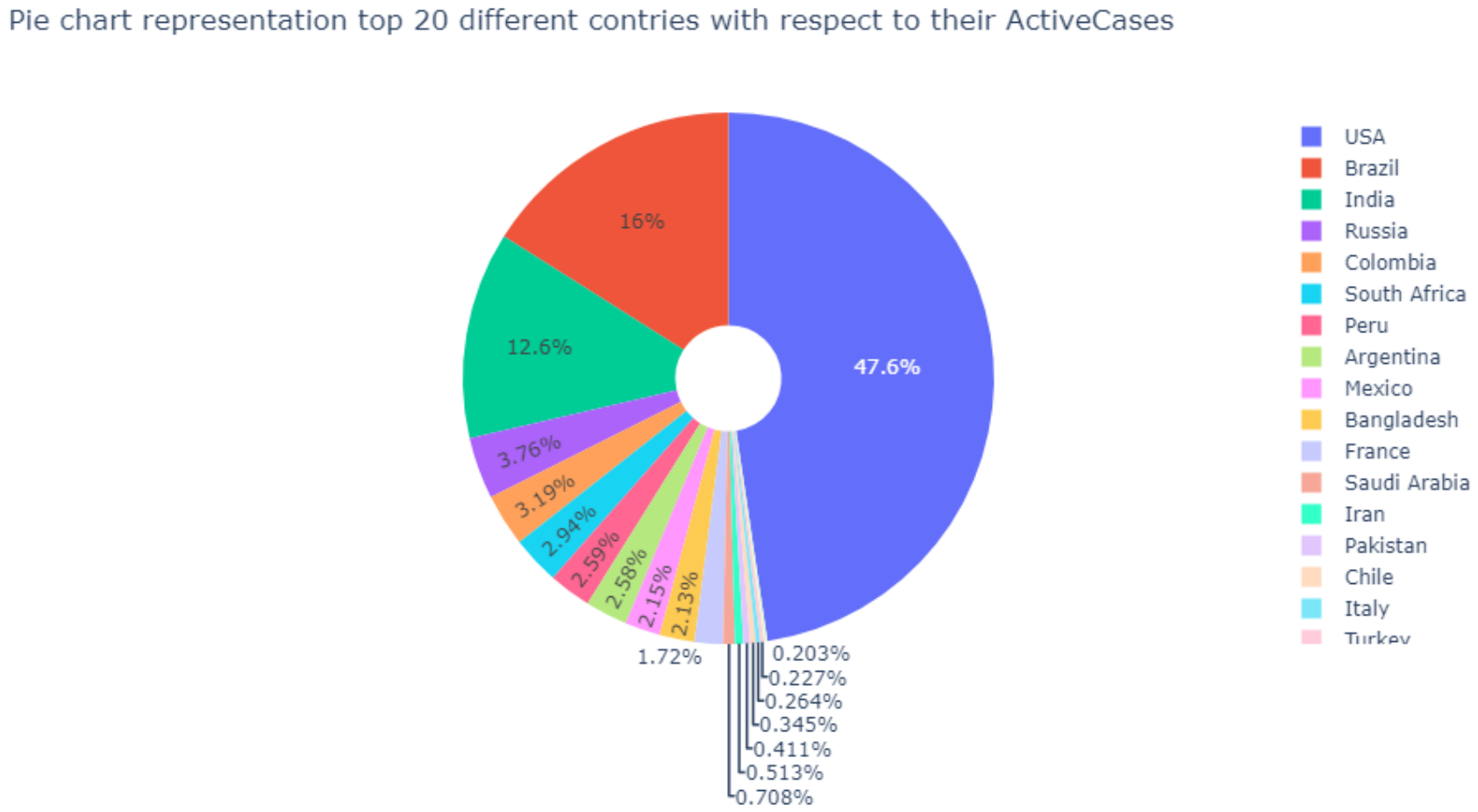
f)


Pie chart representation top 20 different contries with respect to their TotalRecovered
25.9%
20.6%
13.9%
6.8%
3.9%
3.42%
3.12%
3.11%
2.79%
2.58%
2.49%
2.22%
2.02%
1.97%
1.93%
1.45%
1%
0.829%
USA
Brazil
India
Russia
South Africa
Chile
Peru
Mexico
Iran
Pakistan
Saudi Arabia
Turkey
Italy
Germany
Colombia
Bangladesh
Argentina
France

g) Treemap representation of different countries with replect to their ActiveCases.



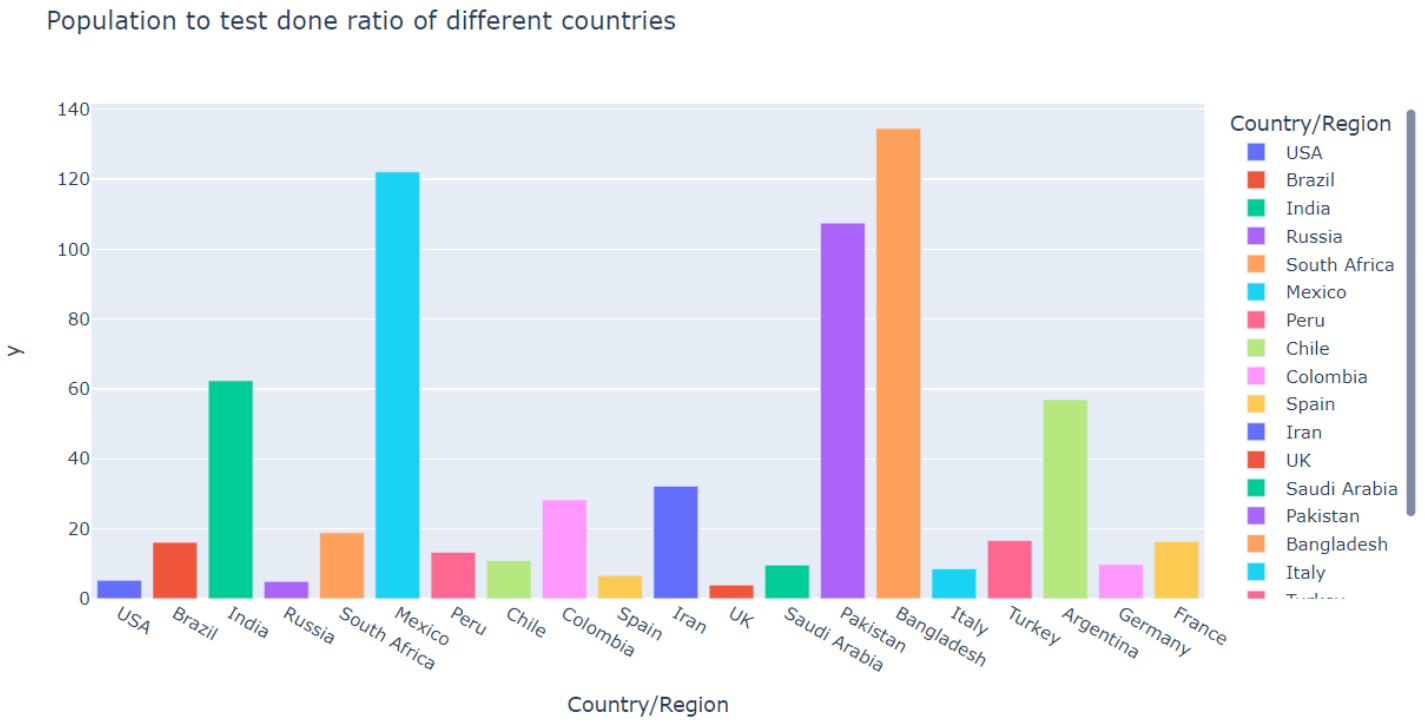
h)


Pie chart representation top 20 different contries with respect to their ActiveCases
16%
12.6%
47.6%
3.76%
3.19%
2.94%
2.59%
2.58%
2.15%
2.13%
1.72%
0.203%
0.227%
0.264%
0.345%
0.411%
0.513%
0.708%
USA
Brazil
India
Russia
Colombia
South Africa
Peru
Argentina
Mexico
Bangladesh
France
Saudi Arabia
Iran
Pakistan
Chile
Italy
Turkey

Q.2 Calculate the ratio between population and test done

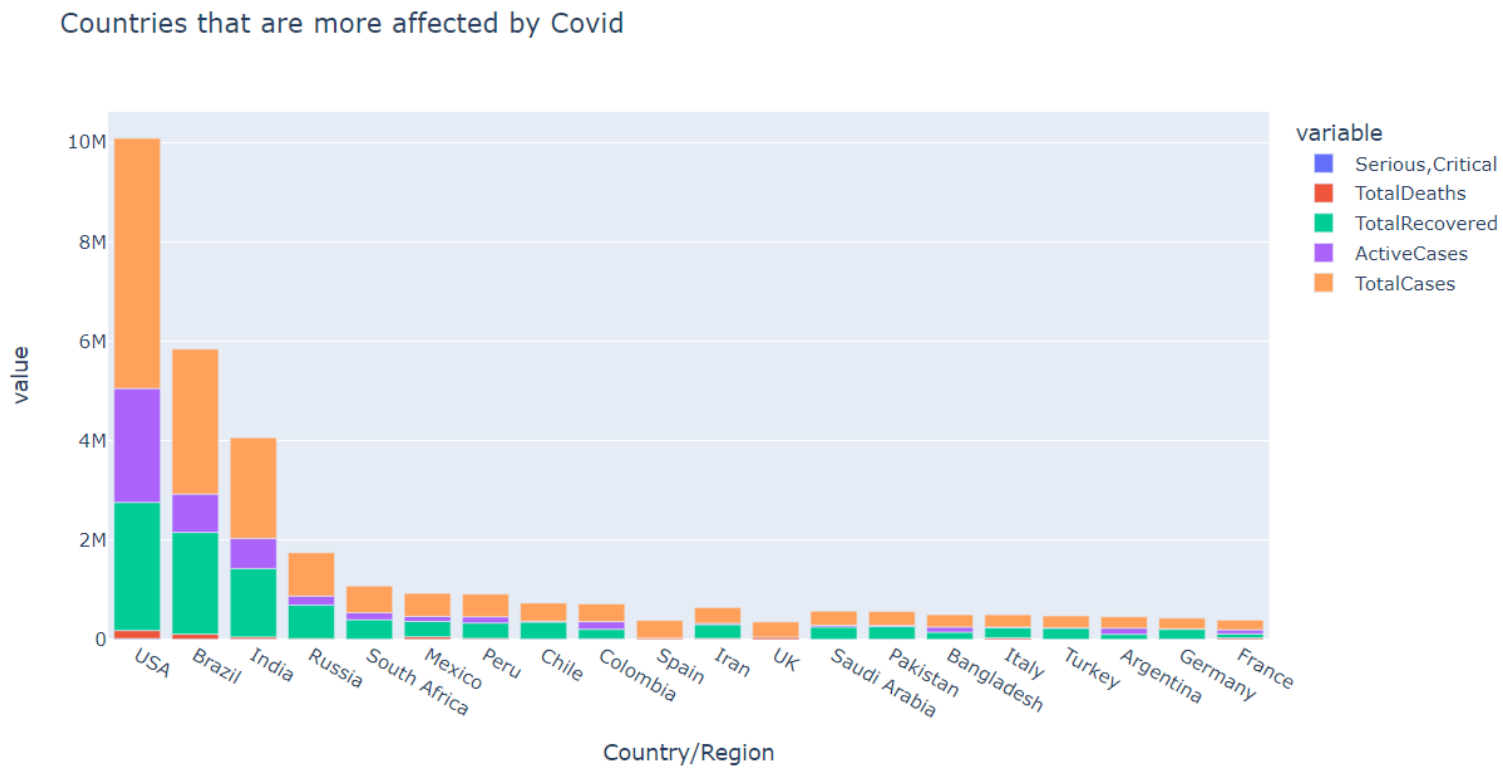
In [ ]:
```
df.head()
```

In [66]:
```
population_to_test_ratio = df['Population']/df['TotalTests'].iloc[0:20]

fig4 = px.bar(df.iloc[0:20], x = 'Country/Region', y = population_to_test_ratio[0:20],
              color = 'Country/Region', title = 'Population to test done ratio of different countries')
fig4.show()
```



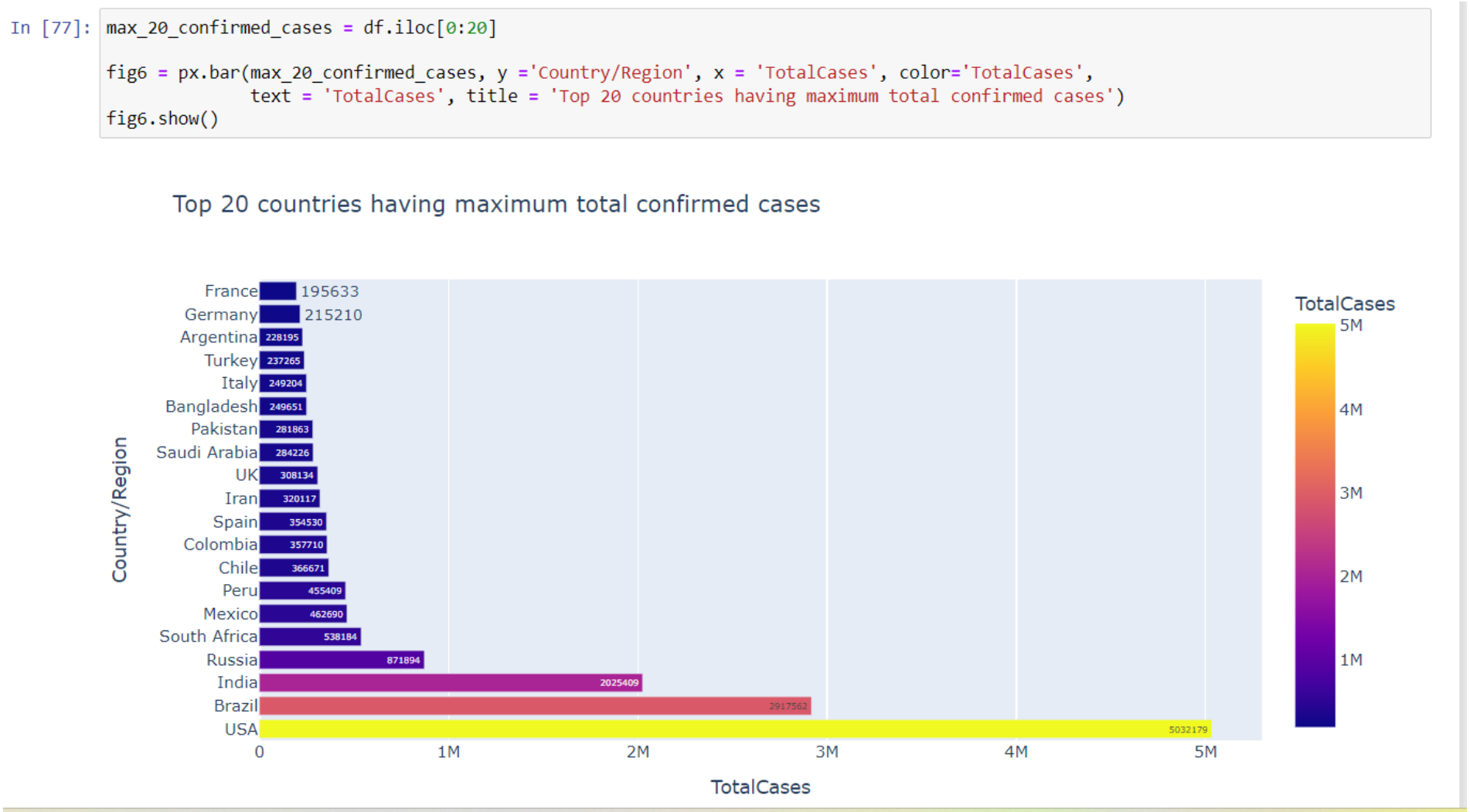
Q.3 check for the top 20 countries in terms of max total confirmed cases, max total active cases, max total recovered cases, max total deaths and serious critical condition cases

```
In [70]: cases = ['Serious,Critical','TotalDeaths', 'TotalRecovered','ActiveCases','TotalCases']

         max_20_total_cases = df.iloc[0:20]
         fig5 = px.bar(max_20_total_cases, x = 'Country/Region', y = cases, title='Countries that are more affected by Covid')
         fig5.show()
```



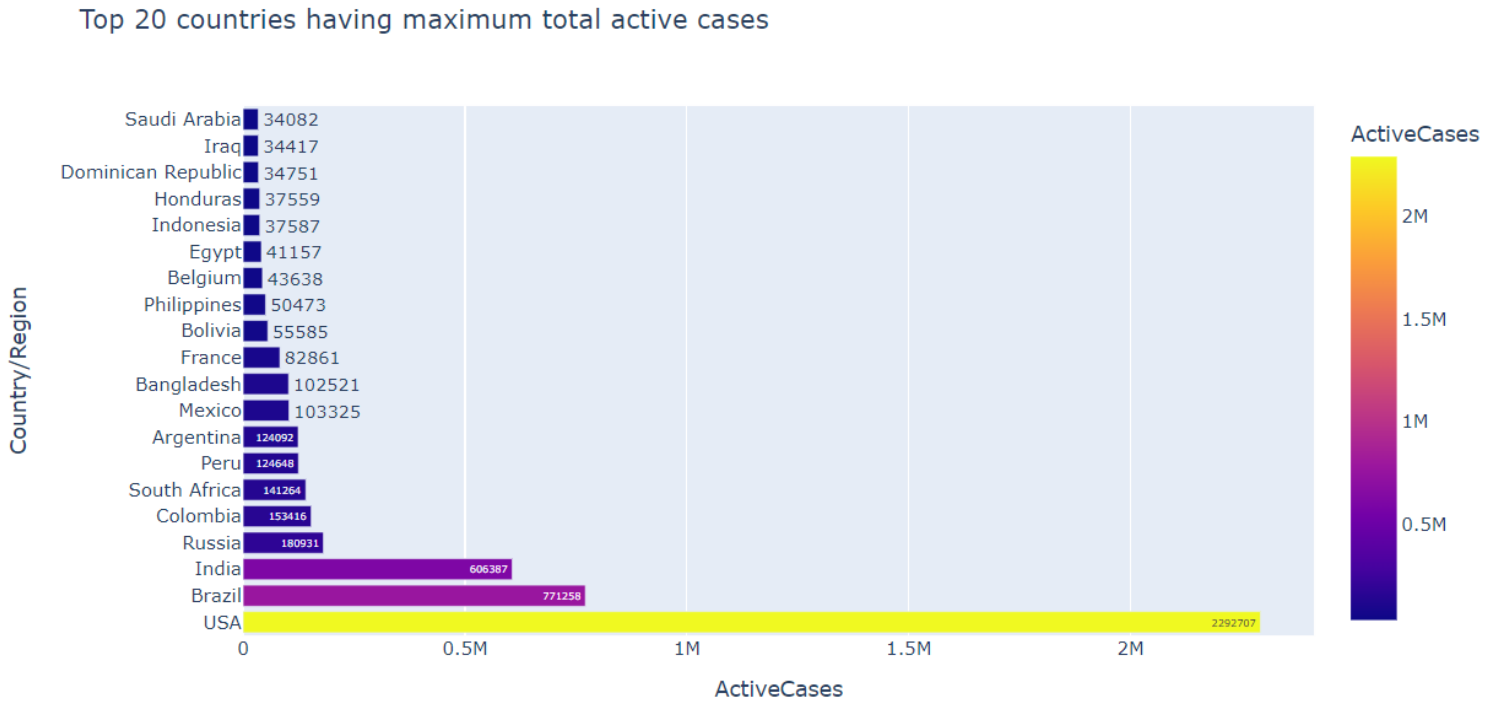
Q.4 Top 20countries having maximum total confirmed cases.

```
In [77]: max_20_confirmed_cases = df.iloc[0:20]

fig6 = px.bar(max_20_confirmed_cases, y ='Country/Region', x = 'TotalCases', color='TotalCases',
              text = 'TotalCases', title = 'Top 20 countries having maximum total confirmed cases')
fig6.show()
```



Q.5 Top 20countries having maximum total confirmed cases.

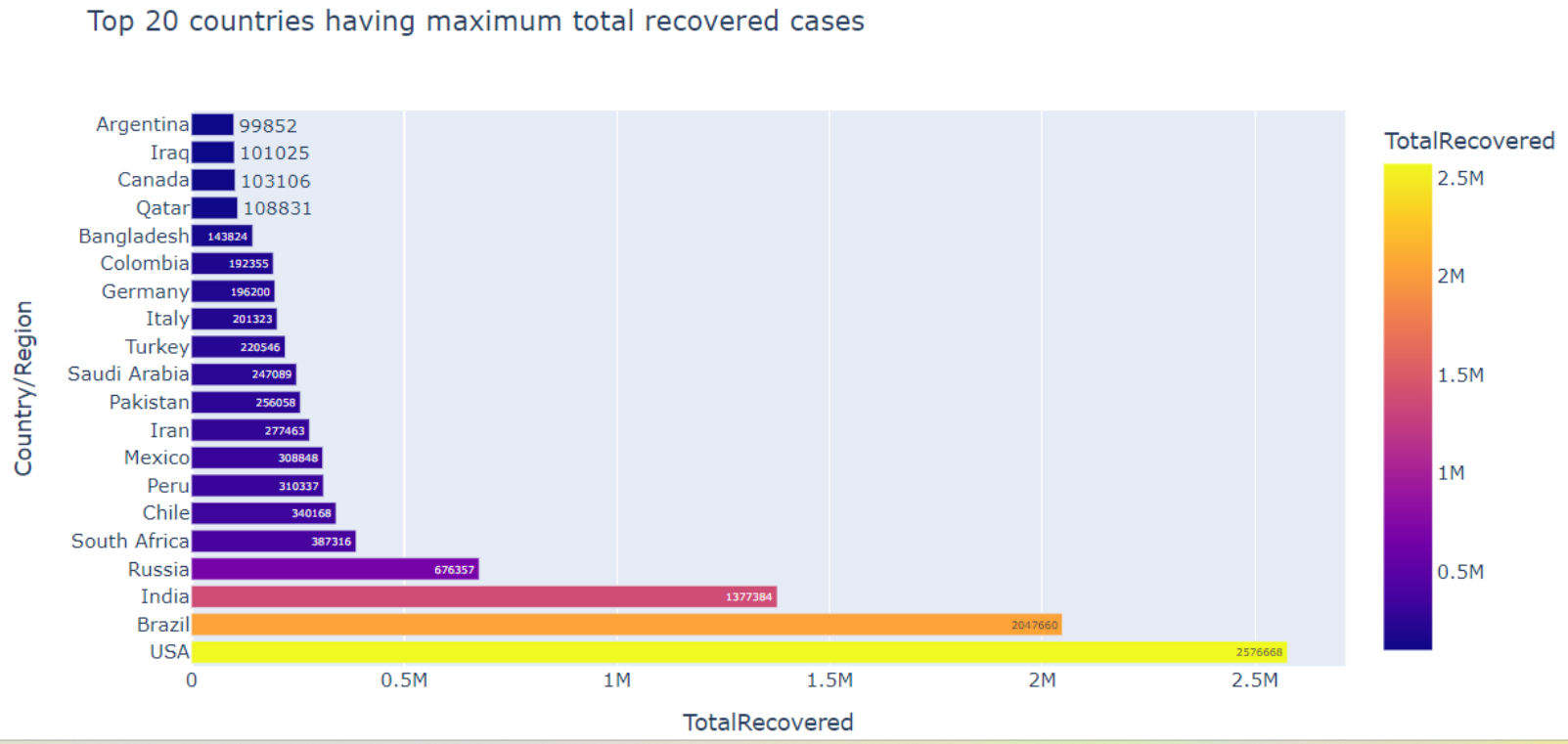
```
In [79]: max_20_active_cases = df.sort_values(by='ActiveCases', ascending=False).iloc[0:20]

         fig7 = px.bar(max_20_active_cases, y ='Country/Region', x = 'ActiveCases', color='ActiveCases',
                       text = 'ActiveCases', title = 'Top 20 countries having maximum total active cases')
         fig7.show()
```



Q.6 Top 20 countries having maximum total recovered cases

f)Top 20 countries having maximum total recovered cases

In [80]:
```
max_20_recovered_data = df.sort_values(by='TotalRecovered', ascending=False).iloc[0:20]

fig8 = px.bar(max_20_recovered_data, y ='Country/Region', x = 'TotalRecovered', color='TotalRecovered',
              text = 'TotalRecovered', title = 'Top 20 countries having maximum total recovered cases')
fig8.show()
```



Q.7 Top 20 countries having maximum total deaths.

In [81]:
```
max_20_total_deaths = df.sort_values(by='TotalDeaths', ascending=False).iloc[0:20]

fig9 = px.bar(max_20_total_deaths, y ='Country/Region', x = 'TotalDeaths', color='TotalDeaths',
              text = 'TotalDeaths', title = 'Top 20 countries having maximum total death cases')
fig9.show()
```