In [1]:
```python
import pandas as pd
import  numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

1. pd.read_csv

pd.read_csv is a function in the panda's library in Python that is used to read a CSV (Comma Separated Values) file and convert it into a pandas DataFrame.

In [2]:
```python
df = pd.read_csv('C:/Users/ADMIN/Desktop/Python/Datasets/Covid Dataset/covid_worldwide.csv')
```

2. df.head()

This function will give you the first 5 rows of the dataframe. This will help ypu to get the simple idea about the \ dataset

In [3]:
```python
df.head()
```

Out[3]:

	Serial Number	Country	Total Cases	Total Deaths	Total Recovered	Active Cases	Total Test	Population
0	1	USA	104,196,861	1,132,935	101,322,779	1,741,147	1,159,832,679	334,805,269
1	2	India	44,682,784	530,740	44,150,289	1,755	915,265,788	1,406,631,776
2	3	France	39,524,311	164,233	39,264,546	95,532	271,490,188	65,584,518
3	4	Germany	37,779,833	165,711	37,398,100	216,022	122,332,384	83,883,596
4	5	Brazil	36,824,580	697,074	35,919,372	208,134	63,776,166	215,353,593

3.df.isnull().sum()

This function will give you the info about which columns have the missing value in totall

In [4]:
```python
df.isnull().sum()
```

Out[4]:
```
Serial Number       0
Country             0
Total Cases         0
Total Deaths        6
Total Recovered    21
Active Cases       19
Total Test         18
Population          3
dtype: int64
```

4.df.dropna

The dropna() method removes the rows that contains NULL values.

The dropna() method returns a new DataFrame object unless the inplace parameter is set to True, in that case the dropna() method does the removing in the original DataFrame instead.

In [5]:
```python
df.dropna()
```

Out[5]:

	Serial Number	Country	Total Cases	Total Deaths	Total Recovered	Active Cases	Total Test	Population
0	1	USA	104,196,861	1,132,935	101,322,779	1,741,147	1,159,832,679	334,805,269
1	2	India	44,682,784	530,740	44,150,289	1,755	915,265,788	1,406,631,776
2	3	France	39,524,311	164,233	39,264,546	95,532	271,490,188	65,584,518
3	4	Germany	37,779,833	165,711	37,398,100	216,022	122,332,384	83,883,596
4	5	Brazil	36,824,580	697,074	35,919,372	208,134	63,776,166	215,353,593
...	...	...	...	...	...	...	...	...
217	218	Anguilla	3,904	12	3,879	13	51,382	15,230
218	219	Macao	3,488	120	3,357	11	7,850	667,490
219	220	Saint Pierre Miquelon	3,452	2	2,449	1,001	25,400	5,759
220	221	Wallis and Futuna	3,427	7	438	2,982	20,508	10,982
224	225	Montserrat	1,403	8	1,376	19	17,762	4,965

195 rows × 8 columns

5.df.info()

This function gives you a basic idea about the dataset what are the type of the data, how many null value do we have \ in the data,how many no of features do we have , whats the memory usage.

In [6]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 231 entries, 0 to 230
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Serial Number    231 non-null    int64
 1   Country          231 non-null    object
 2   Total Cases      231 non-null    object
 3   Total Deaths     225 non-null    object
 4   Total Recovered  210 non-null    object
 5   Active Cases     212 non-null    object
 6   Total Test       213 non-null    object
 7   Population       228 non-null    object
dtypes: int64(1), object(7)
memory usage: 14.6+ KB
```

In [ ]:

6. df.astype

This function is very imp to change the datatype of the column. The reason for changing the data type is to do some numerical calculation

In [7]:
```python
df['Total Cases'] = df['Total Cases'].str.replace(',' , '' , regex=True).astype("float")
```

In [8]:
```python
df['Total Deaths'] = df['Total Deaths'].str.replace(',' , '' , regex=True).astype("float")
```

In [9]:
```python
df['Total Recovered'] = df['Total Recovered'].str.replace(',' , '' , regex=True).astype("float")
```

In [10]:
```python
df['Active Cases'] = df['Active Cases'].str.replace(',' , '' , regex=True).astype("float")
```

In [11]:
```python
df['Total Test'] = df['Total Test'].str.replace(',' , '' , regex=True).astype("float")
```

In [12]:
```python
df['Population'] = df['Population'].str.replace(',' , '' , regex=True).astype("float")
```

In [13]:
```python
df.info()
## as we can see we have change the data type from object to float now we will be able to perform some numerical problems
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 231 entries, 0 to 230
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Serial Number    231 non-null    int64
 1   Country          231 non-null    object
 2   Total Cases      231 non-null    float64
 3   Total Deaths     225 non-null    float64
 4   Total Recovered  210 non-null    float64
 5   Active Cases     212 non-null    float64
 6   Total Test       213 non-null    float64
 7   Population       228 non-null    float64
dtypes: float64(6), int64(1), object(1)
memory usage: 14.6+ KB
```

7.df.describe()

The df.describe() method in pandas is used to generate summary statistics of various features of a DataFrame. It returns a new DataFrame that contains the count, mean, standard deviation, minimum, 25th percentile, median, 75th percentile, and maximum of each numerical column in the original DataFrame

In [14]:
```python
df.describe()
```

Out[14]:

	Serial Number	Total Cases	Total Deaths	Total Recovered	Active Cases	Total Test	Population
count	231.000000	2.310000e+02	2.250000e+02	2.100000e+02	2.120000e+02	2.130000e+02	2.280000e+02
mean	116.000000	2.923460e+06	3.005778e+04	2.993905e+06	9.099885e+04	3.249316e+07	2.849326e+07
std	66.828138	9.479286e+06	1.053806e+05	9.520209e+06	7.663888e+05	1.177370e+08	1.022803e+08
min	1.000000	5.000000e+00	1.000000e+00	2.000000e+00	0.000000e+00	7.850000e+03	7.990000e+02
25%	58.500000	2.400100e+04	2.230000e+02	2.197250e+04	5.250000e+01	3.478150e+05	4.450815e+05
50%	116.000000	2.065920e+05	2.179000e+03	2.351455e+05	1.115000e+03	2.216193e+06	5.676382e+06
75%	173.500000	1.296146e+06	1.445200e+04	1.465768e+06	1.022875e+04	1.298426e+07	2.170759e+07
max	231.000000	1.041969e+08	1.132935e+06	1.013228e+08	1.095262e+07	1.159833e+09	1.406632e+09

You can also include or exclude certain columns and also include non-numerical columns by passing appropriate arguments to the method.

In [ ]:

In [15]:
```python
df.describe(include = 'all')
```

Out[15]:

	Serial Number	Country	Total Cases	Total Deaths	Total Recovered	Active Cases	Total Test	Population
count	231.000000	231	2.310000e+02	2.250000e+02	2.100000e+02	2.120000e+02	2.130000e+02	2.280000e+02
unique	NaN	231	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	USA	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	1	NaN	NaN	NaN	NaN	NaN	NaN
mean	116.000000	NaN	2.923460e+06	3.005778e+04	2.993905e+06	9.099885e+04	3.249316e+07	2.849326e+07
std	66.828138	NaN	9.479286e+06	1.053806e+05	9.520209e+06	7.663888e+05	1.177370e+08	1.022803e+08
min	1.000000	NaN	5.000000e+00	1.000000e+00	2.000000e+00	0.000000e+00	7.850000e+03	7.990000e+02
25%	58.500000	NaN	2.400100e+04	2.230000e+02	2.197250e+04	5.250000e+01	3.478150e+05	4.450815e+05
50%	116.000000	NaN	2.065920e+05	2.179000e+03	2.351455e+05	1.115000e+03	2.216193e+06	5.676382e+06
75%	173.500000	NaN	1.296146e+06	1.445200e+04	1.465768e+06	1.022875e+04	1.298426e+07	2.170759e+07
max	231.000000	NaN	1.041969e+08	1.132935e+06	1.013228e+08	1.095262e+07	1.159833e+09	1.406632e+09

In [16]:
```python
df.describe(exclude = 'number')   # to exclude the numerical column
```

Out[16]:

	Country
count	231
unique	231
top	USA
freq	1

8.df.drop()

This function is usefull in dropping a particular column

In [17]:
```python
# serial nuber column is of no use so its better to drop the column
df.drop("Serial Number" , axis = 1 , inplace = True)
# here we have to sepcify the axis as 1 cause the function needs to know wether we are deleting a cloumn or a row
# here inplace means the changes should be reflected in the original dataset
```

In [18]:
```python
df
# as we can see now there is no serial number column
```

Out[18]:

	Country	Total Cases	Total Deaths	Total Recovered	Active Cases	Total Test	Population
0	USA	104196861.0	1132935.0	101322779.0	1741147.0	1.159833e+09	3.348053e+08
1	India	44682784.0	530740.0	44150289.0	1755.0	9.152658e+08	1.406632e+09
2	France	39524311.0	164233.0	39264546.0	95532.0	2.714902e+08	6.558452e+07
3	Germany	37779833.0	165711.0	37398100.0	216022.0	1.223324e+08	8.388360e+07
4	Brazil	36824580.0	697074.0	35919372.0	208134.0	6.377617e+07	2.153536e+08
...	...	...	...	...	...	...	...
226	Diamond Princess	712.0	13.0	699.0	0.0	NaN	NaN
227	Vatican City	29.0	NaN	29.0	0.0	NaN	7.990000e+02
228	Western Sahara	10.0	1.0	9.0	0.0	NaN	6.261610e+05
229	MS Zaandam	9.0	2.0	7.0	0.0	NaN	NaN
230	Tokelau	5.0	NaN	NaN	5.0	NaN	1.378000e+03

231 rows × 7 columns

9.df.isnull()

This function will help us to know which columns have NAN values

In [19]:
```python
df.isnull()
# this function will give a dataframe with true and false value, if the value is flase then there is no null value
# id the value is true then there is a null value
```

Out[19]:

	Country	Total Cases	Total Deaths	Total Recovered	Active Cases	Total Test	Population
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
226	False	False	False	False	False	True	True
227	False	False	True	False	False	True	False
228	False	False	False	False	False	True	False
229	False	False	False	False	False	True	True
230	False	False	True	True	False	True	False

231 rows × 7 columns

10.The fillna()

method in Pandas is used to fill in missing values in a DataFrame with a specified value or method. By default, it replaces missing values with NaN, but you can specify a different value to use instead as shown below:

1.value: Specifies the value to use to fill in the missing values. Can be a scalar value or a dict of values for different columns. \ 2.method: Specifies the method to use for filling in missing values. Can be 'ffill' (forward-fill) or 'bfill' (backward-fill) or 'interpolate'(interpolate values) or 'pad' or 'backfill' \ 3.axis: Specifies the axis along which to fill in missing values. It can be 0 (rows) or 1 (columns). \ 4.inplace: Whether to fill in the missing values in place (modifying the original DataFrame) or to return a new DataFrame with the missing values filled in. \ 5.limit: Specifies the maximum number of consecutive missing values to fill. \ 6.downcast: Specifies a dictionary of values to use to downcast the data types of columns. \

In [20]:
```python
df['Total Deaths'].fillna(0 , inplace = True)
# here we have filled the na value with 0
```

In [21]:
```python
df['Total Deaths'].isnull().sum()
# now as we can see we have 0 null value left same we can do for other column
```

Out[21]:
```
0
```

In [22]:
```python
df.dropna(inplace=True)
```

In [23]:
```python
df.isnull().sum()
# now by using the drop na function we have removed all the na values
```

Out[23]:
```
Country            0
Total Cases        0
Total Deaths       0
Total Recovered    0
Active Cases       0
Total Test         0
Population         0
dtype: int64
```

Now we have covered all the ways of dealing with missing value

10.df.assign()

pandas' .assign() function is used to add new columns to a DataFrame, based on the computation of existing columns. It allows you to add new columns to a DataFrame without modifying the original dataframe. The function returns a new DataFrame with the added columns.

It's important to note that the original DataFrame df remains unchanged and the new DataFrame df_new is returned with the new columns added.

In [24]:
```python
df_new = df.assign(total_deaths_plus5 = df['Total Deaths']+5)
```

In [25]:
```python
df_new
# as you can see we have created a new column name total_deaths_plus_5
```

Out[25]:

	Country	Total Cases	Total Deaths	Total Recovered	Active Cases	Total Test	Population	total_deaths_plus5
0	USA	104196861.0	1132935.0	101322779.0	1741147.0	1.159833e+09	3.348053e+08	1132940.0
1	India	44682784.0	530740.0	44150289.0	1755.0	9.152658e+08	1.406632e+09	530745.0
2	France	39524311.0	164233.0	39264546.0	95532.0	2.714902e+08	6.558452e+07	164238.0
3	Germany	37779833.0	165711.0	37398100.0	216022.0	1.223324e+08	8.388360e+07	165716.0
4	Brazil	36824580.0	697074.0	35919372.0	208134.0	6.377617e+07	2.153536e+08	697079.0
...	...	...	...	...	...	...	...	...
218	Macao	3488.0	120.0	3357.0	11.0	7.850000e+03	6.674900e+05	125.0
219	Saint Pierre Miquelon	3452.0	2.0	2449.0	1001.0	2.540000e+04	5.759000e+03	7.0
220	Wallis and Futuna	3427.0	7.0	438.0	2982.0	2.050800e+04	1.098200e+04	12.0
223	Falkland Islands	1930.0	0.0	1930.0	0.0	8.632000e+03	3.539000e+03	5.0
224	Montserrat	1403.0	8.0	1376.0	19.0	1.776200e+04	4.965000e+03	13.0

196 rows × 8 columns

11.df.columns()

This method you can use to assign new column names to the dataframe, there are other ways as well to assign new names to the data frame

In the above method you have to provide a list of column names

In [26]:
```python
df.columns = ['Country' , 'Total_cases' , 'Total_deaths' , 'Total_recovered' , 'Active_cases' , 'Total_test' , 'Population']
```

In [27]:
```python
df
```

Out[27]:

	Country	Total_cases	Total_deaths	Total_recovered	Active_cases	Total_test	Population
0	USA	104196861.0	1132935.0	101322779.0	1741147.0	1.159833e+09	3.348053e+08
1	India	44682784.0	530740.0	44150289.0	1755.0	9.152658e+08	1.406632e+09
2	France	39524311.0	164233.0	39264546.0	95532.0	2.714902e+08	6.558452e+07
3	Germany	37779833.0	165711.0	37398100.0	216022.0	1.223324e+08	8.388360e+07
4	Brazil	36824580.0	697074.0	35919372.0	208134.0	6.377617e+07	2.153536e+08
...	...	...	...	...	...	...	...
218	Macao	3488.0	120.0	3357.0	11.0	7.850000e+03	6.674900e+05
219	Saint Pierre Miquelon	3452.0	2.0	2449.0	1001.0	2.540000e+04	5.759000e+03
220	Wallis and Futuna	3427.0	7.0	438.0	2982.0	2.050800e+04	1.098200e+04
223	Falkland Islands	1930.0	0.0	1930.0	0.0	8.632000e+03	3.539000e+03
224	Montserrat	1403.0	8.0	1376.0	19.0	1.776200e+04	4.965000e+03

196 rows × 7 columns

What if you want to assign a specific name to particular column

12.df.query()

Pandas' .query() function allows you to filter a DataFrame based on a Boolean expression. It allows you to select rows from a DataFrame using a query string similar to SQL.\

The function returns a new DataFrame containing only the rows that satisfy the Boolean expression

In [28]:
```python
df_query = df.query('Total_deaths > 1000')
df_query
```

Out[28]:

	Country	Total_cases	Total_deaths	Total_recovered	Active_cases	Total_test	Population
0	USA	104196861.0	1132935.0	101322779.0	1741147.0	1.159833e+09	3.348053e+08
1	India	44682784.0	530740.0	44150289.0	1755.0	9.152658e+08	1.406632e+09
2	France	39524311.0	164233.0	39264546.0	95532.0	2.714902e+08	6.558452e+07
3	Germany	37779833.0	165711.0	37398100.0	216022.0	1.223324e+08	8.388360e+07
4	Brazil	36824580.0	697074.0	35919372.0	208134.0	6.377617e+07	2.153536e+08
...	...	...	...	...	...	...	...
144	Sudan	63742.0	5001.0	58284.0	457.0	5.629410e+05	4.599202e+07
148	Syria	57453.0	3164.0	54289.0	0.0	1.462690e+05	1.936481e+07
158	Mauritius	41652.0	1043.0	39943.0	666.0	3.586750e+05	1.274727e+06
170	Somalia	27318.0	1361.0	13182.0	12775.0	4.004660e+05	1.684180e+07
196	Yemen	11945.0	2159.0	9124.0	662.0	3.295920e+05	3.115487e+07

125 rows × 7 columns

13.df.sort_values()

Pandas' .sort_values() function allows you to sort a DataFrame by one or multiple columns. It sorts the DataFrame based on the values of one or more columns, in ascending or descending order. The function returns a new DataFrame sorted by the specified column(s)

In [29]:
```python
df_sorted_by_total_deaths = df.sort_values(by='Total_deaths')
```

In [30]:
```python
df_sorted_by_total_deaths
# now this function has given us a new dataframe but its in ascending order (smallest to largest)
```

Out[30]:

	Country	Total_cases	Total_deaths	Total_recovered	Active_cases	Total_test	Population
223	Falkland Islands	1930.0	0.0	1930.0	0.0	8.632000e+03	3.539000e+03
216	Nauru	4621.0	1.0	4609.0	11.0	2.050900e+04	1.090300e+04
209	Cook Islands	7011.0	2.0	6993.0	16.0	1.969000e+04	1.757100e+04
219	Saint Pierre Miquelon	3452.0	2.0	2449.0	1001.0	2.540000e+04	5.759000e+03
220	Wallis and Futuna	3427.0	7.0	438.0	2982.0	2.050800e+04	1.098200e+04
...	...	...	...	...	...	...	...
18	Mexico	7368252.0	332198.0	6606633.0	429421.0	1.935620e+07	1.315628e+08
9	Russia	21958696.0	395108.0	21356008.0	207580.0	2.734000e+08	1.458059e+08
1	India	44682784.0	530740.0	44150289.0	1755.0	9.152658e+08	1.406632e+09
4	Brazil	36824580.0	697074.0	35919372.0	208134.0	6.377617e+07	2.153536e+08
0	USA	104196861.0	1132935.0	101322779.0	1741147.0	1.159833e+09	3.348053e+08

196 rows × 7 columns

In [31]:
```python
df_sorted_by_total_cases = df.sort_values(by='Total_cases' , ascending=False)
df_sorted_by_total_cases
# now here we can see it has given us the country which has the highest active cases
```

Out[31]:

	Country	Total_cases	Total_deaths	Total_recovered	Active_cases	Total_test	Population
0	USA	104196861.0	1132935.0	101322779.0	1741147.0	1.159833e+09	3.348053e+08
1	India	44682784.0	530740.0	44150289.0	1755.0	9.152658e+08	1.406632e+09
2	France	39524311.0	164233.0	39264546.0	95532.0	2.714902e+08	6.558452e+07
3	Germany	37779833.0	165711.0	37398100.0	216022.0	1.223324e+08	8.388360e+07
4	Brazil	36824580.0	697074.0	35919372.0	208134.0	6.377617e+07	2.153536e+08
...	...	...	...	...	...	...	...
218	Macao	3488.0	120.0	3357.0	11.0	7.850000e+03	6.674900e+05
219	Saint Pierre Miquelon	3452.0	2.0	2449.0	1001.0	2.540000e+04	5.759000e+03
220	Wallis and Futuna	3427.0	7.0	438.0	2982.0	2.050800e+04	1.098200e+04
223	Falkland Islands	1930.0	0.0	1930.0	0.0	8.632000e+03	3.539000e+03
224	Montserrat	1403.0	8.0	1376.0	19.0	1.776200e+04	4.965000e+03

196 rows × 7 columns

You can also sort by multiple columns

In [32]:
```python
df_sorted_by_total_cases_and_totalrecovered = df.sort_values(['Total_cases' , 'Total_recovered'])
```

In [33]:
```python
df_sorted_by_total_cases_and_totalrecovered
```

Out[33]:

	Country	Total_cases	Total_deaths	Total_recovered	Active_cases	Total_test	Population
224	Montserrat	1403.0	8.0	1376.0	19.0	1.776200e+04	4.965000e+03
223	Falkland Islands	1930.0	0.0	1930.0	0.0	8.632000e+03	3.539000e+03
220	Wallis and Futuna	3427.0	7.0	438.0	2982.0	2.050800e+04	1.098200e+04
219	Saint Pierre Miquelon	3452.0	2.0	2449.0	1001.0	2.540000e+04	5.759000e+03
218	Macao	3488.0	120.0	3357.0	11.0	7.850000e+03	6.674900e+05
...	...	...	...	...	...	...	...
4	Brazil	36824580.0	697074.0	35919372.0	208134.0	6.377617e+07	2.153536e+08
3	Germany	37779833.0	165711.0	37398100.0	216022.0	1.223324e+08	8.388360e+07
2	France	39524311.0	164233.0	39264546.0	95532.0	2.714902e+08	6.558452e+07
1	India	44682784.0	530740.0	44150289.0	1755.0	9.152658e+08	1.406632e+09
0	USA	104196861.0	1132935.0	101322779.0	1741147.0	1.159833e+09	3.348053e+08

196 rows × 7 columns

14.df.sample()

Pandas' .sample() function allows you to randomly select rows from a DataFrame. It returns a new DataFrame containing the randomly selected rows. The function takes several parameters that allow you to control the sampling process, such as the number of rows to return, and whether or not to sample with replacement and seed for reproducibility.

In [34]:
```python
df_sample = df.sample(n=2 , replace = False , random_state=1)
df_sample

# here i wanted 2 column where replacement is false
```

Out[34]:

	Country	Total_cases	Total_deaths	Total_recovered	Active_cases	Total_test	Population
139	Eswatini	74133.0	1422.0	72676.0	35.0	1048704.0	1184817.0
12	Vietnam	11526508.0	43186.0	10612479.0	870843.0	85826548.0	98953541.0

In [35]:
```python
df_sample = df.sample(n=2 , replace = True , random_state=1)
df_sample

# here i wanted 2 column where replacement is True
```

Out[35]:

	Country	Total_cases	Total_deaths	Total_recovered	Active_cases	Total_test	Population
39	Denmark	3173247.0	8145.0	3163849.0	1253.0	129144754.0	5834950.0
153	Papua New Guinea	46750.0	670.0	43982.0	2098.0	249149.0	9292169.0

In [36]:
```python
# if i want a specific column
df_sample_2 = df.sample(n=4 , replace = False , random_state=1 , axis = 1)

df_sample_2
```

Out[36]:

	Population	Total_deaths	Total_cases	Country
0	3.348053e+08	1132935.0	104196861.0	USA
1	1.406632e+09	530740.0	44682784.0	India
2	6.558452e+07	164233.0	39524311.0	France
3	8.388360e+07	165711.0	37779833.0	Germany
4	2.153536e+08	697074.0	36824580.0	Brazil
...	...	...	...	...
218	6.674900e+05	120.0	3488.0	Macao
219	5.759000e+03	2.0	3452.0	Saint Pierre Miquelon
220	1.098200e+04	7.0	3427.0	Wallis and Futuna
223	3.539000e+03	0.0	1930.0	Falkland Islands
224	4.965000e+03	8.0	1403.0	Montserrat

196 rows × 4 columns

15.df.transpose()

df.transpose() is a method in the Pandas library used to transpose the rows and columns of a DataFrame? This means that the rows become columns and the columns become rows.

In [37]:
```python
df_transposed = df.transpose()
```

In [39]:
```python
df_transposed.head()
# Now here as you can see we have transposed rows into columns
```

Out[39]:

	0	1	2	3	4	5	6	7	8	9	...	211	212	213	216	217	218	219
Country	USA	India	France	Germany	Brazil	Japan	S. Korea	Italy	UK	Russia	...	Turks and Caicos	Sao Tome and Principe	Palau	Nauru	Anguilla	Macao	Saint Pierre Miquelon
Total_cases	104196861.0	44682784.0	39524311.0	37779833.0	36824580.0	32588442.0	30197066.0	25453789.0	24274361.0	21958696.0	...	6522.0	6280.0	5986.0	4621.0	3904.0	3488.0	3452.0
Total_deaths	1132935.0	530740.0	164233.0	165711.0	697074.0	68399.0	33486.0	186833.0	204171.0	395108.0	...	38.0	77.0	9.0	1.0	12.0	120.0	2.0
Total_recovered	101322779.0	44150289.0	39264546.0	37398100.0	35919372.0	21567425.0	29740877.0	25014986.0	24020088.0	21356008.0	...	6451.0	6202.0	5976.0	4609.0	3879.0	3357.0	2449.0
Active_cases	1741147.0	1755.0	95532.0	216022.0	208134.0	10952618.0	422703.0	251970.0	50102.0	207580.0	...	33.0	1.0	1.0	11.0	13.0	11.0	1001.0

5 rows × 196 columns

Thank you

In [ ]: