```python
In [1]:
import os, re, glob
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import folium
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from IPython.display import IFrame
from IPython.display import Image
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

```python
In [2]:
root = '.'
recent_date = '11-24-2021'
previous_date = '10-24-2021'

duplicate_columns = {"Lat":"Latitude",
                     "Long_":"Longitude",
                     "Incidence_Rate":"Incident_Rate",
                     "Case-Fatality_Ratio": "Case_Fatality_Ratio",
                     "Province/State": "Province_State",
                     "Country/Region": "Country_Region",
                     "Last Update": "Last_Update"}

recent_df = pd.read_csv(os.path.join(root, (recent_date + ".csv")))
previous_df = pd.read_csv(os.path.join(root, (previous_date + ".csv")))

for key, value in duplicate_columns.items():
    if key in recent_df.columns:
        recent_df = recent_df.rename(columns={key: value})
    if key in previous_df.columns:
        previous_df = previous_df.rename(columns={key: value})
```

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

Out[3]:

	FIPS	Admin2	Province_State	Country_Region	Last_Update	Latitude	Longitude	Confirmed	Deat
0	NaN	NaN	NaN	Afghanistan	2021-11-25 04:22:33	33.93911	67.709953	157032	73
1	NaN	NaN	NaN	Albania	2021-11-25 04:22:33	41.15330	20.168300	197776	30
2	NaN	NaN	NaN	Algeria	2021-11-25 04:22:33	28.03390	1.659600	209463	60
3	NaN	NaN	NaN	Andorra	2021-11-25 04:22:33	42.50630	1.521800	16426	1
4	NaN	NaN	NaN	Angola	2021-11-25 04:22:33	-11.20270	17.873900	65080	17

```python
In [4]:
previous_df.head()
```

Out[4]:

	FIPS	Admin2	Province_State	Country_Region	Last_Update	Latitude	Longitude	Confirmed	Deat
0	NaN	NaN	NaN	Afghanistan	2021-10-25 04:22:31	33.93911	67.709953	155944	72
1	NaN	NaN	NaN	Albania	2021-10-25 04:22:31	41.15330	20.168300	181696	28
2	NaN	NaN	NaN	Algeria	2021-10-25 04:22:31	28.03390	1.659600	205822	58
3	NaN	NaN	NaN	Andorra	2021-10-25 04:22:31	42.50630	1.521800	15404	1
4	NaN	NaN	NaN	Angola	2021-10-25 04:22:31	-11.20270	17.873900	63861	16

```python
In [5]:
current_df = pd.DataFrame(columns=['Province_State','Country_Region','Confirmed','Deat
current_df['Province_State'] = recent_df['Province_State']
current_df['Country_Region'] = recent_df['Country_Region']
current_df['Confirmed'] = recent_df['Confirmed'] - previous_df['Confirmed']
current_df['Deaths'] = recent_df['Deaths'] - previous_df['Deaths']
```

```python
In [6]:
current_df.shape
```

```
Out[6]: (4006, 4)
```

```python
In [7]:
current_df.head()
```

Out[7]:

	Province_State	Country_Region	Confirmed	Deaths
0	NaN	Afghanistan	1088	51
1	NaN	Albania	16080	189
2	NaN	Algeria	3641	145
3	NaN	Andorra	1022	1
4	NaN	Angola	1219	34

```python
In [8]:
name_number = 'JonathanBenAvraham-2124450.csv'
current_df.to_csv(name_number, index=False)
```

```python
In [9]:
data = pd.read_csv(name_number)
```

```python
In [10]:
data.head()
```

Out[10]:

	Province_State	Country_Region	Confirmed	Deaths
0	NaN	Afghanistan	1088	51
1	NaN	Albania	16080	189
2	NaN	Algeria	3641	145
3	NaN	Andorra	1022	1
4	NaN	Angola	1219	34

```python
In [11]:
print(data.shape)
```

```
(4006, 4)
```

```python
In [12]:
print(data.count())
```

```
Province_State    3832
Country_Region    4006
Confirmed         4006
Deaths            4006
dtype: int64
```

Q1. Print how many null values exist in the dataset.

```python
In [13]:
for column in data.columns:
    print(column + ' : ' + str(data[column].isna().sum()))
```

```
Province_State : 174
Country_Region : 0
Confirmed : 0
Deaths : 0
```

```python
In [14]:
data.loc[data['Province_State'].isnull(),'Province_State'] = data['Country_Region']
```

```python
In [15]:
data.head()
```

Out[15]:

	Province_State	Country_Region	Confirmed	Deaths
0	Afghanistan	Afghanistan	1088	51
1	Albania	Albania	16080	189
2	Algeria	Algeria	3641	145
3	Andorra	Andorra	1022	1
4	Angola	Angola	1219	34

```python
In [16]:
states = data['Province_State'].unique()
print("Number of unique States - ", len(states))
```

```
Number of unique States -  765
```

Q2. Print how many unique countries exist in the dataset using a similar approach.

```python
In [17]:
print("Number of unique Countries - ", len(states))
```

```
Number of unique Countries -  765
```

```python
In [18]:
import datetime, time, requests
from time import sleep
from geopy.geocoders import Nominatim


def get_lat_lon(place):
    geolocator = Nominatim(user_agent=name_number)
    location = geolocator.geocode(place)
    lat_lon = location.latitude, location.longitude

    output = [float(i) for i in lat_lon]
    return output
```

```python
In [19]:
from tqdm import tqdm
geo_lat = []
geo_lon = []
not_found = []
for state in tqdm(states):
    time.sleep(0.2)
    lat_lon = [None, None]
    try:
        lat_lon = get_lat_lon(state)
    except:
        not_found.append(state)

    geo_lat.append(lat_lon[0])
    geo_lon.append(lat_lon[1])
if len(not_found) > 0:
    print("Locations are not found for - ", not_found)
else:
    print("Found all the locations")
```

```
100%|██████████| 765/765 [21:39<00:00,  1.70s/it]
Locations are not found for -  ['Antwerp', 'Liege', 'Limburg', 'Unknown', 'Walloon Bra
bant', 'Bhutan', 'Bolivia', 'Alberta', 'Grand Princess', 'New Brunswick', 'Northwest T
erritories', 'Nunavut', 'Ontario', 'Repatriated Travellers', 'Saskatchewan', 'Yukon',
'Central African Republic', 'Araucania', 'Atacama', 'Biobio', 'Los Lagos', 'Magallane
s', 'Metropolitana', 'Valparaiso', 'Fujian', 'Gansu', 'Hong Kong', 'Jilin', 'Ningxia',
'Qinghai', 'Shanxi', 'Tianjin', 'Tibet', 'Xinjiang', 'Zhejiang', 'Atlantico', 'Boliva
r', 'Caqueta', 'Casanare', 'Guainia', 'Huila', 'Narino', 'Putumayo', 'Equatorial Guine
a', 'Eswatini', 'Finland', 'French Guiana', 'French Polynesia', 'Martinique', 'Mayott
e', 'Saint Barthelemy', 'Saint Pierre and Miquelon', 'Wallis and Futuna', 'France', 'B
aden-Wurttemberg', 'Bayern', 'Brandenburg', 'Hamburg', 'Haiti', 'Andaman and Nicobar I
slands', 'Andhra Pradesh', 'Assam', 'Chhattisgarh', 'Dadra and Nagar Haveli and Daman
and Diu', 'Haryana', 'Jharkhand', 'Lakshadweep', 'Madhya Pradesh', 'Maharashtra', 'Cal
abria', 'Lazio', 'Marche', 'Hiroshima', 'Hyogo', 'Iwate', 'Kanagawa', 'Kyoto', 'Mie',
'Nagano', 'Nagasaki', 'Niigata', 'Okinawa', 'Lesotho', 'MS Zaandam', 'Kedah', 'Kelanta
n', 'Melaka', 'Sabah', 'Sarawak', 'Selangor', 'W.P. Putrajaya', 'Michoacan', 'Morelo
s', 'Nayarit', 'Sonora', 'Tabasco', 'Tamaulipas', 'Yucatan', 'Zacatecas', 'Morocco',
'Namibia', 'Aruba', 'Bonaire, Sint Eustatius and Saba', 'Drenthe', 'Flevoland', 'Cajam
arca', 'Cusco', 'Huanuco', 'Junin', 'Lima', 'Loreto', 'Piura', 'Tumbes', 'Ucayali', 'Q
atar', 'Adygea Republic', 'Altai Krai', 'Amur Oblast', 'Belgorod Oblast', 'Bryansk Obl
ast', 'Chelyabinsk Oblast', 'Chukotka Autonomous Okrug', 'Mordovia Republic', 'Sakha
(Yakutiya) Republic', 'Samara Oblast', 'Saratov Oblast', 'Andalusia', 'Sri Lanka', 'Su
mmer Olympics 2020', 'Suriname', 'Gotland', 'Halland', 'Uppsala', 'Varmland', 'Vastra
Gotaland', 'Switzerland', 'Tajikistan', 'Tanzania', 'Togo', 'Oklahoma', 'Oregon', 'Ten
nessee', 'Texas', 'Virgin Islands', 'Virginia', 'West Virginia', 'Wisconsin', 'Wyomin
g', 'Crimea Republic*', 'Dnipropetrovsk Oblast', 'Ivano-Frankivsk Oblast', 'Khmelnytsk
yi Oblast', 'Kiev', 'Kiev Oblast', 'Luhansk Oblast', 'Mykolaiv Oblast', 'Sumy Oblast',
'Ternopil Oblast', 'Zhytomyr Oblast', 'United Arab Emirates', 'Bermuda', 'Channel Isla
nds', 'England', 'Falkland Islands (Malvinas)', 'Northern Ireland', 'Wales', 'Vietna
m', 'West Bank and Gaza', 'Zambia']
```

```python
In [71]:
states_list = states.tolist() #converting states to list to index list's items
lats = []
lons = []

for i, r in data.iterrows():
    state = r['Province_State']
    if state in states_list:
        index_list = states_list.index(state)
    else:
        pass
    lats.append(geo_lat[index_list])
    lons.append(geo_lon[index_list])

data['Latitude'] = lats
data['Longitude'] = lons
```

```python
In [21]:
data.head()
```

Out[21]:

	Province_State	Country_Region	Confirmed	Deaths	Latitude	Longitude
0	Afghanistan	Afghanistan	1088	51	33.768006	66.238514
1	Albania	Albania	16080	189	41.000028	19.999962
2	Algeria	Algeria	3641	145	28.000027	2.999983
3	Andorra	Andorra	1022	1	42.540717	1.573203
4	Angola	Angola	1219	34	-11.877577	17.569124

Q3. Check whether the latitude and longitude values we retrieved from geopy are same as the latitude and longitude given in the dataset. Identify and report differences in values.

```python
In [22]:
for i, r in data.iterrows():
    if r['Latitude'] == previous_df.iloc[i]['Latitude'] and r['Longitude'] == previous
        print('True')
    else:
        print('False')
```

```
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
True
False
False
False
False
False
False
False
False
True
False
False
True
False
False
False
True
False
True
False
True
False
False
False
False
False
True
True
True
True
False
False
True
True
True
False
True
False
True
False
False
False
False
False
True
False
False
False
False
False
True
False
False
False
False
False
False
False
False
False
False
False
True
True
False
False
True
False
False
True
False
True
True
False
True
False
True
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
```

```
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
False
```

In [23]:

```
data = data[data['Latitude'].notna()]
```

In [24]:

```
data.shape
```

Out[24]: (3076, 6)

In [25]:

```
clustering_data = data[["Confirmed", "Deaths"]]
```

In [26]:

```
clustering_data.head()
```

Out[26]:

	Confirmed	Deaths
0	1088	51
1	16080	189
2	3641	145
3	1022	1
4	1219	34

In [27]:

```
scaler = StandardScaler()
X_scaled = scaler.fit(clustering_data).transform(clustering_data.astype(np.float))
```

In [28]:

```
cluster_range = range( 1, 20 )
cluster_errors = []
for num_clusters in cluster_range:
    clusters = KMeans( num_clusters )
    clusters.fit(X_scaled)
    cluster_errors.append(clusters.inertia_)

clusters_df = pd.DataFrame( {"num_clusters":cluster_range,
                             "cluster_errors": cluster_errors } )
plt.figure(figsize=(16,6))
plt.plot(clusters_df.num_clusters, clusters_df.cluster_errors, marker = "o" );
```

In [29]:

```
#       Fitting K-Means to      the     dataset
kmeans  =       KMeans(n_clusters       = 4,    init    = 'k-means++',  random_state
y_kmeans        =       kmeans.fit_predict(X_scaled)
#beginning      of              the     cluster numbering       with    1       instea
y_kmeans1=y_kmeans+1
#       New     list    called  cluster
cluster = list(y_kmeans1)
#       Adding  cluster to      our     data    set
clustering_data['cluster']      =       cluster
```

In [30]:

```
clustering_data.head(10)
```

Out[30]:

	Confirmed	Deaths	cluster
0	1088	51	1
1	16080	189	1
2	3641	145	1
3	1022	1	1
4	1219	34	1
5	119	18	1
6	39509	632	1
7	40370	1366	1
8	376	3	1
9	6896	69	1

In [31]:

```
kmeans_mean_cluster     =       pd.DataFrame(round(clustering_data.groupby('cluster')
kmeans_mean_cluster
```

Out[31]:

	Confirmed	Deaths
cluster		
1	4783.3	92.8
2	8644095.0	75470.0
3	-7135073.0	-43756.0
4	-576299.7	-15667.4

In [32]:

```
data['cluster'] =       cluster
clusters        =       data[['Province_State', 'cluster']]
clusters.loc[clusters['cluster']        == 2]
```

Out[32]:

	Province_State	cluster
672	Turkey	2

In [33]:

```
data['cluster'] =       cluster
clusters        =       data[['Province_State', 'cluster']]
clusters.loc[clusters['cluster']        == 3]
```

Out[33]:

	Province_State	cluster
671	Tunisia	3

Q4. Discuss what do you think about other clusters?

answer to q4

In [87]:

```
data.head()
```

Out[87]:

	Province_State	Country_Region	Confirmed	Deaths	Latitude	Longitude	cluster	color
0	Afghanistan	Afghanistan	1088	51	33.768006	66.238514	1	green
1	Albania	Albania	16080	189	41.000028	19.999962	1	green
2	Algeria	Algeria	3641	145	28.000027	2.999983	1	green
3	Andorra	Andorra	1022	1	42.540717	1.573203	1	green
4	Angola	Angola	1219	34	-11.877577	17.569124	1	green

In [35]:

```
def get_color(cluster_id):
    if cluster_id == 2:
        return 'darkred'
    if cluster_id == 1:
        return 'green'
    if cluster_id == 3:
        return 'orange'
    if cluster_id == 4:
        return 'yellow'

data["color"] = data["cluster"].apply(lambda x: get_color(x))
```

In [36]:

```
data.head(10)
```

Out[36]:

	Province_State	Country_Region	Confirmed	Deaths	Latitude	Longitude	cluster	color
0	Afghanistan	Afghanistan	1088	51	33.768006	66.238514	1	green
1	Albania	Albania	16080	189	41.000028	19.999962	1	green
2	Algeria	Algeria	3641	145	28.000027	2.999983	1	green
3	Andorra	Andorra	1022	1	42.540717	1.573203	1	green
4	Angola	Angola	1219	34	-11.877577	17.569124	1	green
5	Antigua and Barbuda	Antigua and Barbuda	119	18	17.223472	-61.955461	1	green
6	Argentina	Argentina	39509	632	-34.996496	-64.967282	1	green
7	Armenia	Armenia	40370	1366	40.769627	44.673665	1	green
8	Australian Capital Territory	Australia	376	3	-35.488350	149.002694	1	green
9	New South Wales	Australia	6896	69	-31.875984	147.286949	1	green

In [103…

```
#create a       map
this_map        =       folium.Map(location     =[data["Latitude"].mean(),

def plot_dot(point):
                                '''input:       series  that    contains        a
                                this    function        creates a       CircleMarker
                                folium.CircleMarker(location=[point.Latitude,   point




#clustered_full.apply(,axis=1)  #use    this    to      iterate through every   row
data.apply(plot_dot,    axis    = 1)
#Set    the     zoom    to      the     maximum possible
this_map.fit_bounds(this_map.get_bounds())
#Save   the     map     to      an      HTML    file
this_map.save(os.path.join('covid_map.html'))
```

Q5. Write a report describing your understanding of the clustering results (based on the map). Discuss whether you agree or disagree with the results achieved? Provide a evidence to support the findings. E.g., a news website supporting or contrasting the clustering result.

In [38]:

```
for state in data['Province_State']:
    if state == 'Lesotho':
        index_of_state = data[data['Province_State']== state].index.values
        print(data.loc[index_of_state]['color'])
```

In [39]:

```
for country in data['Country_Region']:
    if country == 'United Kingdom':
        index_of_state = data[data['Country_Region']== country].index.values
        print(data.loc[index_of_state]['color'])
```

```
3981     green
3983     green
3984     green
3988     green
3989     green
3990     green
3992    yellow
3993     green
3994     green
Name: color, dtype: object
3981     green
3983     green
3984     green
3988     green
3989     green
3990     green
3992    yellow
3993     green
3994     green
Name: color, dtype: object
3981     green
3983     green
3984     green
3988     green
3989     green
3990     green
3992    yellow
3993     green
3994     green
Name: color, dtype: object
3981     green
3983     green
3984     green
3988     green
3989     green
3990     green
3992    yellow
3993     green
3994     green
Name: color, dtype: object
3981     green
3983     green
3984     green
3988     green
3989     green
3990     green
3992    yellow
3993     green
3994     green
Name: color, dtype: object
3981     green
3983     green
3984     green
3988     green
3989     green
3990     green
3992    yellow
3993     green
3994     green
Name: color, dtype: object
3981     green
3983     green
3984     green
3988     green
3989     green
3990     green
3992    yellow
3993     green
3994     green
Name: color, dtype: object
3981     green
3983     green
3984     green
3988     green
3989     green
3990     green
3992    yellow
3993     green
3994     green
Name: color, dtype: object
3981     green
3983     green
3984     green
3988     green
3989     green
3990     green
3992    yellow
3993     green
3994     green
Name: color, dtype: object
```

In [40]:

```
for city in data['Province_State']:
    if city == 'Scotland':
        index_of_scotland = data[data['Province_State'] == city].index.values
        print(data.loc[index_of_scotland]['color'] + ' ' +data.loc[index_of_scotland]
```

```
3993    green United Kingdom
dtype: object
```

In [41]:

```
for country in data['Country_Region']:
    if country == 'South Africa':
        index_of_southafrica = data[data['Country_Region']==country].index.values
        print(data.loc[index_of_southafrica]['color'] + ' ' + data.loc[index_of_southa
```

```
614     green South Africa
dtype: object
```

In [45]:

```
for country in data['Country_Region']:
    if country == 'South Africa':
        index = data[data['Country_Region'] == country].index.values
        print(data.loc[index]['color'] + ' ' + str(data.loc[index]['Deaths']))
        print('\n')
```

```
614     green 614    732\nName: Deaths, dtype: int64
Name: color, dtype: object
```

In [51]:

```
for country in data['Province_State']:
    if country == 'Namibia':
        index = data[data['Province_State'] == country].index.values
        print(data.loc[index]['color'])
```

In [63]:

```
for country in current_df['Country_Region']:
    if country == 'Namibia':
        index = current_df[current_df['Country_Region']== country].index.values
        print(current_df.loc[index]['Deaths'])
```

```
445    21
Name: Deaths, dtype: int64
```

In [58]:

```
data.head()
```

Out[58]:

	Province_State	Country_Region	Confirmed	Deaths	Latitude	Longitude	cluster	color
0	Afghanistan	Afghanistan	1088	51	33.768006	66.238514	1	green
1	Albania	Albania	16080	189	41.000028	19.999962	1	green
2	Algeria	Algeria	3641	145	28.000027	2.999983	1	green
3	Andorra	Andorra	1022	1	42.540717	1.573203	1	green
4	Angola	Angola	1219	34	-11.877577	17.569124	1	green

In [66]:

```
recent_df.head()
```

Out[66]:

	FIPS	Admin2	Province_State	Country_Region	Last_Update	Latitude	Longitude	Confirmed	Deat
0	NaN	NaN	NaN	Afghanistan	2021-11-25 04:22:33	33.93911	67.709953	157032	73
1	NaN	NaN	NaN	Albania	2021-11-25 04:22:33	41.15330	20.168300	197776	30
2	NaN	NaN	NaN	Algeria	2021-11-25 04:22:33	28.03390	1.659600	209463	60
3	NaN	NaN	NaN	Andorra	2021-11-25 04:22:33	42.50630	1.521800	16426	1
4	NaN	NaN	NaN	Angola	2021-11-25 04:22:33	-11.20270	17.873900	65080	17

In [134…

```
for country in recent_df['Country_Region']:
    if country == 'Namibia':
        index = recent_df[recent_df['Country_Region']==country].index.values
        print(recent_df.loc[index]['Latitude'])
        print(recent_df.loc[index]['Longitude'])
        data.loc[3077] ={'Province_State': recent_df['Province_State'].iloc[index],
                         'Country_Region': recent_df['Country_Region'].iloc[index],
                         'Confirmed': recent_df['Confirmed'].iloc[index],
                         'Deaths': recent_df['Deaths'].iloc[index],
                         'Latitude': recent_df['Latitude'].iloc[index],
                         'Longitude': recent_df['Longitude'].iloc[index],
                         'cluster': 4,
                         'color': 'yellow'
                        }
```

```
445   -22.9576
Name: Latitude, dtype: float64
445    18.4904
Name: Longitude, dtype: float64
```

In [139…

```
print(recent_df['Country_Region'].iloc[index])
```

```
445    Namibia
Name: Country_Region, dtype: object
```

In [136…

```
data.shape
```

Out[136… (3076, 8)

In [155…

```
recent_df.shape
```

Out[155… (4006, 14)

In [156…

```
data.shape
```

Out[156… (3076, 8)

In [176…

```
geolocator = Nominatim(user_agent=name_number)
print(geolocator.geocode('Lesotho'))
location = geolocator.geocode('Lesotho')
lat_lon = location.latitude, location.longitude
print(lat_lon)
```

```
Lesotho
(-29.6039267, 28.3350193)
```

Report

Hypothesis

Due to the new variant of Covid-19, B.1.1.529. I would expect countries in Southern Africa to be marked as unsafe to visit; denoted by belonging to the red cluster, or a red circle on the rendered map. Given that the green cluster is likely to contain countries that have low covid case and death rates, and the red and yellow clusters are likely to have countries with significantly higher confirmed case and death rates. I have used data collected during the dates 24-10-2021 and 24-11-2021, we should see that South Africa and its neighbouring countries have had an alarming rise in confirmed cases in the past month.

Dataset Used

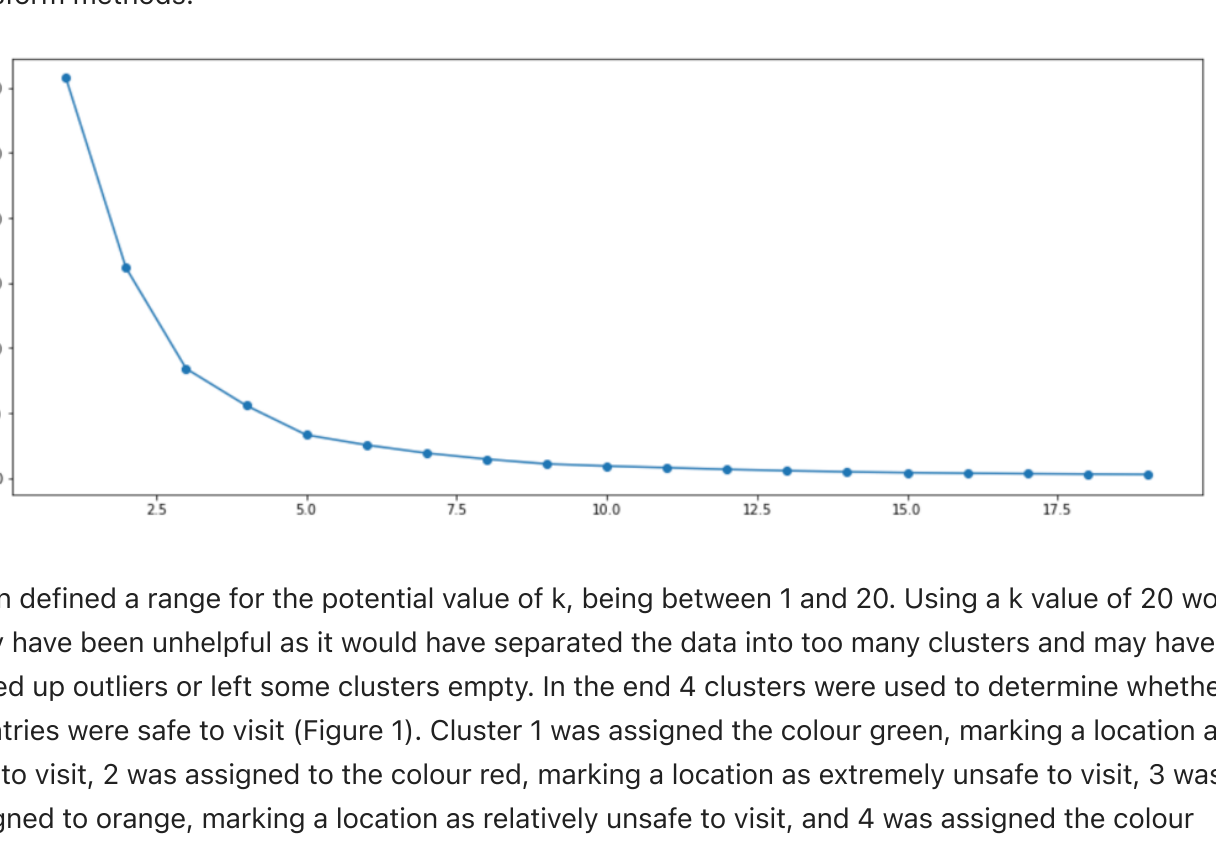
The dataset used was obtained from the following GitHub Repo: Dataset Link (COVID-19, n.d., accessed 1.9.22)

Testing The Hypothesis

To test the hypothesis, I ran through the worksheet as directed. Unfortunately, some of the neighbouring countries that I would have liked to test were not available in the final dataset. Countries such as Namibia, eSwatini and Lesotho did not make the final list. This was due to longitudes and latitudes not being found for them during the stage where geopy was being used to get the longitudes and latitudes of each country.

However, I was able to visualise results for South Africa, Zimbabwe and Botswana. Based on the results in my covid_map.html file, South Africa has been placed in the green cluster along with Botswana and Namibia has been placed in the yellow cluster.

During the pre-processing stage, we had to normalize the data in the Confirmed Cases and Deaths columns, because the values for Confirmed Cases were significantly higher than the values for the Deaths column. To normalize the values the StandardScaler class was used. StandardScaler is a preprocessing tool provided in the scikitlearn library. It allows us to normalise our data using the fit and transform methods.



I then defined a range for the potential value of k, being between 1 and 20. Using a k value of 20 would likely have been unhelpful as it would have separated the data into too many clusters and may have picked up outliers or left some clusters empty. In the end 4 clusters were used to determine whether countries were safe to visit (Figure 1). Cluster 1 was assigned the colour green, marking a location as safe to visit, 2 was assigned to the colour red, marking a location as extremely unsafe to visit, 3 was assigned to orange, marking a location as relatively unsafe to visit, and 4 was assigned the colour yellow, making a country as relatively safe to visit.

Each location was then plotted on a map using the Map class of the folium library. This map would mark a circle in the colour assigned to each location by its cluster. The map was then rendered and saved in a html document called covid_map.html.

Conclusion

As the clustering results were based on Confirmed Cases and Deaths for each location, I would normally agree with the results. However as new data has come to light in recent days about the emergence of a new Covid-19 variant, and its mutations being likely to improve the virus' ability to spread, I must disagree with the results and affirm that the countries listed, including Namibia, Lesotho and eSwatini which were not listed should not be within the green or yellow clusters but in the red cluster.

In conclusion, using only data provided by Confirmed Cases and Deaths for each location may not be the best solution for determining whether or not a country is safe to visit using the k-means clustering method. Adding other dimensions to the data like variants for each location, how easy each variant spreads and how lethal each variant is, might prove to produce better predictions.

Bibliography

COVID-19/csse_covid_19_data/csse_covid_19_daily_reports at master · CSSEGISandData/COVID-19 [WWW Document], n.d. . GitHub. URL https://github.com/CSSEGISandData/COVID-19 (accessed 1.9.22).


```
In [ ]:
```