K-Means Clustering, Covid-19 - 2021 Data - Part A

Importing all the necessary libraries using import

This work is divided into two part - Part A starting from 1st of January 2021 to 1st of August 2021 and Part B starting from 1st of January 2022 to 1st of August 2022 as we are considering a period of two years (2021-2022). This is the Part A part of this work.

In this notebook, we will be using clustering to group the places according to number of covid deaths and cases. With this we will be able to identify low-risk and high-risk places based on the spread of COVID-19 during a period of time.

To start, we are reading one file for Part A work

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
```

In [2]:
```
root = r'C:\Users\LenovoX260\Desktop\Assignment WK. 3 Covid - Part 2 - Previous\Covid'
recent_date = "01-01-2021"
previous_date = "08-01-2021"

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})
```

Lets visualise the two dataframes

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

Out[3]:

	FIPS	Admin2	Province_State	Country_Region	Last_Update	Latitude	Longitude	Confirmed	Dea
0	NaN	NaN	NaN	Afghanistan	2021-01-02 05:22:33	33.93911	67.709953	52513	2
1	NaN	NaN	NaN	Albania	2021-01-02 05:22:33	41.15330	20.168300	58316	1
2	NaN	NaN	NaN	Algeria	2021-01-02 05:22:33	28.03390	1.659600	99897	2
3	NaN	NaN	NaN	Andorra	2021-01-02 05:22:33	42.50630	1.521800	8117	
4	NaN	NaN	NaN	Angola	2021-01-02 05:22:33	-11.20270	17.873900	17568	

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

Out[4]:

	FIPS	Admin2	Province_State	Country_Region	Last_Update	Latitude	Longitude	Confirmed	Dea
0	NaN	NaN	NaN	Afghanistan	2021-08-02 04:21:36	33.93911	67.709953	147501	6
1	NaN	NaN	NaN	Albania	2021-08-02 04:21:36	41.15330	20.168300	133121	2
2	NaN	NaN	NaN	Algeria	2021-08-02 04:21:36	28.03390	1.659600	172564	4
3	NaN	NaN	NaN	Andorra	2021-08-02 04:21:36	42.50630	1.521800	14678	
4	NaN	NaN	NaN	Angola	2021-08-02 04:21:36	-11.20270	17.873900	42815	1

Now, let's create a separate dataset for the preferred period. We are interested in number of Confirmed cases and number of Deaths happened during the chosen period. Since we need the specific values only for this period, we subtract the number of confirmed cases and number of deaths from recent_df and previous_df. For other fields: Province_State and Country_Region, we keep the same values

```
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']
```

Let's check the dimensions of the created dataframe.

In [6]:
```
current_df.shape
```
Out[6]:
```
(4011, 4)
```
In [7]:
```
current_df["Confirmed"] = current_df["Confirmed"].apply(np.abs)
current_df["Deaths"] = current_df["Deaths"].apply(np.abs)
```
In [8]:
```
current_df.head()
```
Out[8]:

	Province_State	Country_Region	Confirmed	Deaths
0	NaN	Afghanistan	94988.0	4536.0
1	NaN	Albania	74805.0	1276.0
2	NaN	Algeria	72667.0	1529.0
3	NaN	Andorra	6561.0	44.0
4	NaN	Angola	25247.0	611.0

Now lets save the Pandas dataframe in a csv using the following code.

In [9]:
```
current_number = 'AdaobiEjiasi-2306317.csv'
current_df.to_csv(current_number, index=False)
```

Now read the saved csv to a Pandas dataframe 'data' using the following code.

In [10]:
```
data = pd.read_csv(current_number)
```
In [11]:
```
data.head()
```
Out[11]:

	Province_State	Country_Region	Confirmed	Deaths
0	NaN	Afghanistan	94988.0	4536.0
1	NaN	Albania	74805.0	1276.0
2	NaN	Algeria	72667.0	1529.0
3	NaN	Andorra	6561.0	44.0
4	NaN	Angola	25247.0	611.0

Let's check the number of rows available in the data.

In [12]:
```
print(data.shape)
```
```
(4011, 4)
```

There are total 4011 rows and 4 columns available in the data, which we can get from data.shape. Just to see how many values exist in the data columns, we can use code below which will give the count of values if exist, rest of the column values are null. Null values can be checked using isnull() function

In [13]:
```
print(data.count())
```

```
Province_State    3833
Country_Region    4011
Confirmed         4011
Deaths            4011
dtype: int64
```

Printing how many null values exist in the dataset.

In [14]:
```
data.isnull().sum()
```

Out[14]:
```
Province_State    178
Country_Region      0
Confirmed           0
Deaths              0
dtype: int64
```

We are looking at clustering high-risk regions during the selected period of time by Province/State. But as you can see there are some rows where the Province/State is null. For those rows we are using the value in Country/Region as Province/State, using following code

In [15]:
```
data.loc[data['Province_State'].isnull(),'Province_State'] = data['Country_Region']
```

After performing this operation, let's visualise our dataframe

In [16]:
```
data.head()
```

Out[16]:

	Province_State	Country_Region	Confirmed	Deaths
0	Afghanistan	Afghanistan	94988.0	4536.0
1	Albania	Albania	74805.0	1276.0
2	Algeria	Algeria	72667.0	1529.0
3	Andorra	Andorra	6561.0	44.0
4	Angola	Angola	25247.0	611.0

Since our analysis is based on States, let's calculate how many unique values are available for the Province_State.

In [17]:
```
states = data['Province_State'].unique()
print("Number of unique States - ", len(states))
```

```
Number of unique States -  771
```

Printing how many unique countries exist in the dataset using a similar approach

In [18]:
```
countries = data["Country_Region"].unique()
print("Number of unique countries - ", len(countries))
```

```
Number of unique countries -  200
```

In the function below, we will get the latitude and longitude using geopy library. We will add them back to the dataset. You can read more about the geopy library at https://geopy.readthedocs.io/en/stable/ (https://geopy.readthedocs.io/en/stable/). Another option to get latitude and longitude is to use the latitude and longitude columns in the first dataframe. However, to get familiar with the gropy library, we will get the latitude and longitude using geopy.

In [19]:
```
pip install geopy
```

```
Requirement already satisfied: geopy in c:\users\lenovox260\anaconda3\lib\site-packag
es (2.3.0)
Requirement already satisfied: geographiclib<3,>=1.52 in c:\users\lenovox260\anaconda
3\lib\site-packages (from geopy) (2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [20]:
```
import datetime, time, requests
from time import sleep
from geopy.geocoders import Nominatim

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

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

In the following code, we iterate through the states in our dataset and retrieve the longitude and latitude from the get_lat_lon method. This code cell will take some time to run. Therefore, we are using a progress bar to monitor the progress. More information about putting a progress bar can be found in https://tqdm.github.io/ (https://tqdm.github.io/). There are some states where the geopy library fails to retrieve longitude and latitude. We print them at the end of the code.

In [21]:
```
from tqdm import tqdm

geo_lat = []
geo_lon = []

not_found = []
found = []
for state in tqdm(states):
    time.sleep(0.2)
    lat_lon = [None, None]
    try:
        lat_lon = get_lat_lon(state)
        found.append(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")

#if len(found) > 0:
# print("Locations are found for - ", found)
```

```
100%|██████████████████████████████████████████████████████████████████████
| 771/771 [06:26<00:00,  1.99it/s]
Locations are not found for -  ['Repatriated Travellers', 'Sakha (Yakutiya) Republi
c', 'Summer Olympics 2020', 'W.P. Kuala Lumpur']
```

We are appending the latitude and longitude information to the dataframe.

In [22]:
```
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']
    index_list = states_list.index(state)
    lats.append(geo_lat[index_list])
    lons.append(geo_lon[index_list])


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

Let's look at our dataset now

In [23]:
```
data.head()
```

Out[23]:

	Province_State	Country_Region	Confirmed	Deaths	Latitude	Longitude
0	Afghanistan	Afghanistan	94988.0	4536.0	33.768006	66.238514
1	Albania	Albania	74805.0	1276.0	41.000028	19.999962
2	Algeria	Algeria	72667.0	1529.0	28.000027	2.999983
3	Andorra	Andorra	6561.0	44.0	42.540717	1.573203
4	Angola	Angola	25247.0	611.0	-11.877577	17.569124

Checking whether the latitude and longitude values we retrieved from geopy are same as the latitude and longitude given in the dataset. Identifying and reporting differences in values

In [24]:
```
diff_lat = []
diff_lon = []
for i, r in data.iterrows():
    state = r['Province_State']
    index_list = states_list.index(state)
    lat = r['Latitude']
    lon = r['Longitude']
    if lat != geo_lat[index_list]:
        diff_lat.append((state, lat, geo_lat[index_list]))
    if lon != geo_lon [index_list]:
        diff_lon.append((state, lon, geo_lon[index_list]))
```

```
if len(diff_lat) > 0 or len(diff_lon) > 0:
    print('Differences found in Latitude values:', diff_lat)
    print('\n')
    print('Differences found in Longitude values:', diff_lon)
else:
    print('No Difference found in Latitude or Longitude values')
```

```
Differences found in Latitude values: [('Repatriated Travellers', nan, None), ('Sakha
(Yakutiya) Republic', nan, None), ('Summer Olympics 2020', nan, None), ('W.P. Kuala L
umpur', nan, None)]


Differences found in Longitude values: [('Repatriated Travellers', nan, None), ('Sakh
a (Yakutiya) Republic', nan, None), ('Summer Olympics 2020', nan, None), ('W.P. Kuala
Lumpur', nan, None)]
```

Let's select only the rows that have Latitude and Longitude information.

In [25]:
```
#Selected rows without NaN
data = data[data['Latitude'].notna()]
```

After removing the rows where Latitude and Longitude are null, let's check the size of the data

In [26]:
```
data.shape
```

Out[26]:
```
(4007, 6)
```

We have 4011 rows and 6 columns for the clustering process. Since our clustering process will be based on Confirmed and Deaths, we will create a new dataframe having only these two columns.

In [27]:
```
clustering_data = data[["Confirmed", "Deaths"]]
```

Let's visualise our clustering dataframe

In [28]:
```
clustering_data.head()
```

Out[28]:

	Confirmed	Deaths
0	94988.0	4536.0
1	74805.0	1276.0
2	72667.0	1529.0
3	6561.0	44.0
4	25247.0	611.0

For the clustering, we have two features, number of cases and number of deaths. These two features are on different scales as number of cases are usually higher than number of deaths. Since the k-means algorithm is dependent on Euclidean distance, having two features on different scales can

be problematic to the k-means algorithm. Therefore, we first perform a normalisation step on the clustering dataset.

```
In [29]: scaler = StandardScaler()
         X_scaled = scaler.fit(clustering_data).transform(clustering_data.astype(np.float))
```

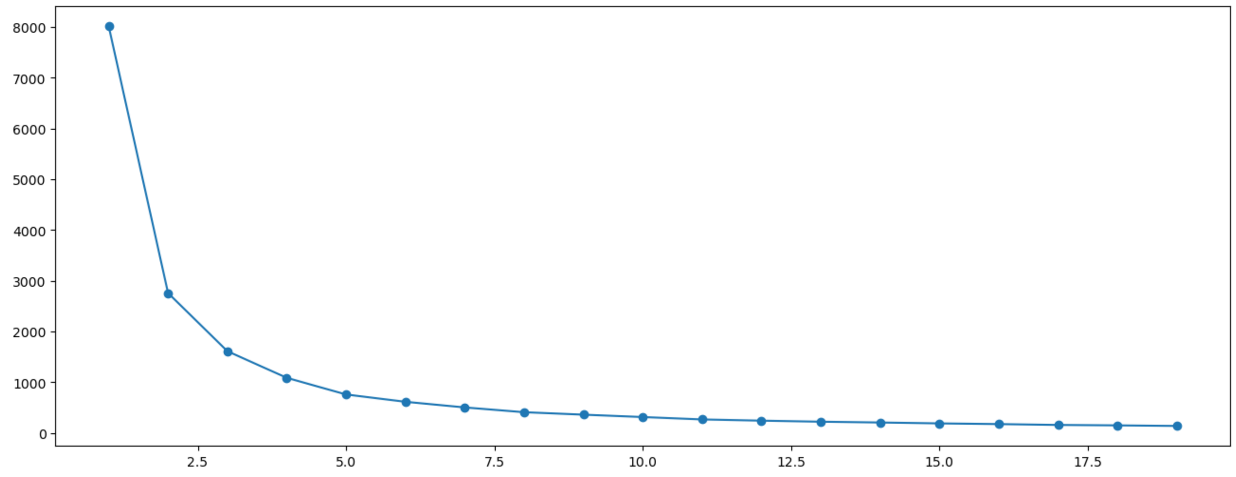
We need to input K (i.e., how many clusters we are expecting) to K-Means clustering. i.e., the number of clusters must be pre-determined for this method. To do so, let's start with creating scree plot. Scree plot is a plot between WCSS (Within cluster sum of squares) and number of clusters. Without sound domain knowledge or in the scenarios with unclear motives, the scree plots help us decide the number of clusters to specify

```
In [30]: 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 the plot above: X-axis represents the k and Y-axis represents cluster error. We are interested in finding Elbow point, which defines the optimal value of k.

In this graph there are two optimal values for k; k=4 and k=6. SSE is decreasing linearly after both of these points. Therefore, number of clusters in this dataset can be both 4 or 6. For ease of explainability of this notebook, we have considered number of clusters to be 4. But if you prefer, 6 clusters, you can use it in your analysis.

Let's input number of clusters as 4 to the K-Means algorithm. Since number of deaths and confirmed cases are in different scales, we are feeding the scaled dataframe (X_scaled) to the K-Means algorithm.

In [31]:

```
# Fitting K-Means to the dataset
kmeans = KMeans(n_clusters = 4, init = 'k-means++', random_state = 10)
y_kmeans = kmeans.fit_predict(X_scaled)
#beginning of the cluster numbering with 1 instead of 0
y_kmeans1=y_kmeans+1
# New list called cluster
cluster = list(y_kmeans1)
# Adding cluster to our data set
clustering_data['cluster'] = cluster
```

Lets see how our clustters look now

In [32]:

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

Out[32]:

	Confirmed	Deaths	cluster
0	94988.0	4536.0	1
1	74805.0	1276.0	1
2	72667.0	1529.0	1
3	6561.0	44.0	1
4	25247.0	611.0	1
5	1144.0	38.0	1
6	3306253.0	62453.0	2
7	70601.0	1791.0	1
8	6.0	0.0	1
9	4413.0	15.0	1

We need to understand the intuition behind the clusters. To do that let's inspect the clusters. With the following code, we calculate the mean value of k clusters.

In [33]:

```
kmeans_mean_cluster = pd.DataFrame(round(clustering_data.groupby('cluster').mean(),1)
kmeans_mean_cluster
```

Out[33]:

	Confirmed	Deaths
cluster		
1	21122.9	436.6
2	2627777.2	64455.5
3	721588.2	15592.1
4	5772401.2	106152.5

As you can see our cluster 2 has the second highest number of deaths and Cluster 1 has the lowest number of deaths and confirmed cases. To get an idea about the members in cluster 2, let's view them using the following code.

In [34]:
```
data['cluster'] = cluster
clusters = data[['Province_State', 'cluster']]
clusters.loc[clusters['cluster'] == 2]
```

Out[34]:

	Province_State	cluster
6	Argentina	2
65	Sao Paulo	2
214	France	2
264	Ladakh	2
265	Lakshadweep	2
267	Maharashtra	2
279	Tripura	2
286	Iraq	2
287	Ireland	2
480	Puno	2
493	Altai Republic	2
597	C. Valenciana	2
3956	England	2

In [35]:
```
data['cluster'] = cluster
clusters = data[['Province_State', 'cluster']]
clusters.loc[clusters['cluster'] == 3]
```

Out[35]:

	Province_State	cluster
20	Bangladesh	3
45	Bahia	3
46	Ceara	3
49	Goias	3
53	Minas Gerais	3
...	...	...
1276	Illinois	3
1285	Illinois	3
3420	Texas	3
3960	Montserrat	3
3973	Yemen	3

83 rows × 2 columns

Cluster 3 has the third largest number of deaths and confirmed cases

Most of these countries were in the red list countries in UK. Therefore, we can use the news sources to confirm that our clustering algorithm makes sense.

Discussing what do you think about other clusters?

We will display the data for cluster 1 and 4 and discuss the outcome

In [36]:
```
#Cluster 1
data['cluster'] = cluster
clusters = data[['Province_State', 'cluster']]
clusters.loc[clusters['cluster'] == 1]
```

Out[36]:

	Province_State	cluster
0	Afghanistan	1
1	Albania	1
2	Algeria	1
3	Andorra	1
4	Angola	1
...	...	...
4006	Unknown	1
4007	Nauru	1
4008	Niue	1
4009	Tuvalu	1
4010	Pitcairn Islands	1

3907 rows × 2 columns

Cluster 1 has the lowest number of confirmed cases and deaths

In [37]:
```
#Cluster 4
data['cluster'] = cluster
clusters = data[['Province_State', 'cluster']]
clusters.loc[clusters['cluster'] == 4]
```

Out[37]:

	Province_State	cluster
216	Gambia	4
269	Meghalaya	4
654	Alabama	4
3966	Wales	4

Cluster 4 has the largest number of confirmed cases and deaths

Now, we need to visualise the clustered data in a map. First let's see how our dataframe looks like.

In [38]:
```
data.head()
```

Out[38]:

	Province_State	Country_Region	Confirmed	Deaths	Latitude	Longitude	cluster
0	Afghanistan	Afghanistan	94988.0	4536.0	33.768006	66.238514	1
1	Albania	Albania	74805.0	1276.0	41.000028	19.999962	1
2	Algeria	Algeria	72667.0	1529.0	28.000027	2.999983	1
3	Andorra	Andorra	6561.0	44.0	42.540717	1.573203	1
4	Angola	Angola	25247.0	611.0	-11.877577	17.569124	1

Let's assign a color to each cluster using the following method

In [39]:
```
def get_color(cluster_id):
    if cluster_id == 4:
        return 'darkred'
    if cluster_id == 1:
        return 'green'
    if cluster_id == 3:
        return 'orange'
    if cluster_id == 2:
        return 'yellow'

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

Let's see how our dataframe looks now.

In [40]:
```
data.head(10)
```

Out[40]:

	Province_State	Country_Region	Confirmed	Deaths	Latitude	Longitude	cluster	color
0	Afghanistan	Afghanistan	94988.0	4536.0	33.768006	66.238514	1	green
1	Albania	Albania	74805.0	1276.0	41.000028	19.999962	1	green
2	Algeria	Algeria	72667.0	1529.0	28.000027	2.999983	1	green
3	Andorra	Andorra	6561.0	44.0	42.540717	1.573203	1	green
4	Angola	Angola	25247.0	611.0	-11.877577	17.569124	1	green
5	Antigua and Barbuda	Antigua and Barbuda	1144.0	38.0	17.223472	-61.955461	1	green
6	Argentina	Argentina	3306253.0	62453.0	-34.996496	-64.967282	2	yellow
7	Armenia	Armenia	70601.0	1791.0	40.769627	44.673665	1	green
8	Australian Capital Territory	Australia	6.0	0.0	-35.488350	149.002694	1	green
9	New South Wales	Australia	4413.0	15.0	-31.875984	147.286949	1	green

As you can see, every state now has a color. Now let's put them in a map using the following code.

In [41]:
```
#create a map
this_map = folium.Map(location =[data["Latitude"].mean(),
                                 data["Longitude"].mean()], zoom_start=5)

def plot_dot(point):
    '''input: series that contains a numeric named latitude and a numeric named longit
    this function creates a CircleMarker and adds it to your this_map'''
    folium.CircleMarker(location=[point.Latitude, point.Longitude],
                        radius=2,
                        color=point.color,
                        weight=1).add_to(this_map)



#clustered_full.apply(,axis=1) #use this to iterate through every row in your datafran
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'))
```

Report

Writing a report describing my understanding of the clustering results (based on the map), discussing whether i agree or disagree with the results achieved by providing an evidence to support the findings. E.g., a news website supporting or contrasting the clustering result.

- The report part of this work will be done in the Part B (Covid 2022 dataset).