K-Means Clustering, Covid-19 - 2022 Data Set - Part B

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

Hypothesis

The severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) is responsible for the ongoing global outbreak of a coronavirus disease (herein referred to as COVID-19).

A novel coronavirus (CoV) began to circulate among humans in Wuhan, China, around December 2019. Initially, the impact of the virus on humans was poorly understood. Since then, this virus, named “severe acute respiratory syndrome coronavirus 2” (SARS-CoV-2), has emerged as the source of a global pandemic, with nearly 115 million confirmed cases reported worldwide and over 2.56 million fatalities as of early March 2021. The pervasiveness and detrimental impact of SARS-CoV-2 across the globe has established it among the most notorious pandemics that have ever been recorded in human history.

Before the introduction of the vacination, 2020 and 2021 was said to be the worst hit period for the virus and the spread slowed down towards the tail end of 2021 and few occurences in 2022 and 2023.

The effect of the virus caused a some counties to impose a travel restriction and forced countries into lockdown to contain the spread. The UK particularly placed some countries on the redlist these countries were considered as the worst hit countries by the UK government.

In this work we will be working with 2021 and 2022 dataset in analysing the spread of the covid virus across the world. This will involve analysing selected data based on the effect of the spread by Locations, Death rate, No of confirmed cases and Recovered cases. After which we will group the countries state into clusters by the worst hit to the lowest and allocate color legends (darkred, green, yellow and orange) for ease of discussion in the report session of this work. We will also be visualizing these state on the map and compare the outcome of our analysis with onlines articles and news letters. The color legends will also help us in reading the map for the worst hit, medium to lowest affected state.

The datset durations for our comparative analysis is;

- Current Data: 01-01-2022 and 08-01-2022
- Previous Data: 01-01-2021 and 08-01-2021

The raw data will be extracted from https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_daily_reports website and we will perform data analysis, data transformation, data clustering using confirmed cases and deaths degeocoding and visualizing the affected areas on the world map based on geo-location using K-means clustering.

This notebook only shows the analysis for one duration (Current 2022 Data), Previous Data will be analysed on a different notebook and conclusions/reports of both outcome will be recorded in the report session of this notebook.

Now let's dive into exploring the dataset.

Importing all the necessary libraries using import

In [11]:
```
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 [12]:
```
root = r'C:\Users\LenovoX260\Desktop\Assignment WK.3 Current - Covid\Covid'
recent_date = "01-01-2022"
previous_date = "08-01-2022"

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})
```

Let's visualise the two dataframes

In [13]:
```
recent_df.head()
```

Out[13]:

	FIPS	Admin2	Province_State	Country_Region	Last_Update	Latitude	Longitude	Confirmed	Dea
0	NaN	NaN	NaN	Afghanistan	2022-01-02 04:20:52	33.93911	67.709953	158107	7
1	NaN	NaN	NaN	Albania	2022-01-02 04:20:52	41.15330	20.168300	210224	3
2	NaN	NaN	NaN	Algeria	2022-01-02 04:20:52	28.03390	1.659600	218818	6
3	NaN	NaN	NaN	Andorra	2022-01-02 04:20:52	42.50630	1.521800	23740	
4	NaN	NaN	NaN	Angola	2022-01-02 04:20:52	-11.20270	17.873900	82398	1

In [14]:
```
previous_df.head()
```

Out[14]:

	FIPS	Admin2	Province_State	Country_Region	Last_Update	Latitude	Longitude	Confirmed	Dea
0	NaN	NaN	NaN	Afghanistan	2022-08-02 04:20:53	33.93911	67.709953	185930	7
1	NaN	NaN	NaN	Albania	2022-08-02 04:20:53	41.15330	20.168300	312375	3
2	NaN	NaN	NaN	Algeria	2022-08-02 04:20:53	28.03390	1.659600	267546	6
3	NaN	NaN	NaN	Andorra	2022-08-02 04:20:53	42.50630	1.521800	45508	
4	NaN	NaN	NaN	Angola	2022-08-02 04:20:53	-11.20270	17.873900	102301	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 [15]:
```
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 [16]:
```
current_df.shape
```

Out[16]:
```
(4016, 4)
```

In [17]:
```
current_df["Confirmed"] = current_df["Confirmed"].apply(np.abs)
current_df["Deaths"] = current_df["Deaths"].apply(np.abs)
```

In [18]:
```
current_df.head()
```

Out[18]:

	Province_State	Country_Region	Confirmed	Deaths
0	NaN	Afghanistan	27823	395
1	NaN	Albania	102151	331
2	NaN	Algeria	48728	592
3	NaN	Andorra	21768	13
4	NaN	Angola	19903	140

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

In [24]:
```
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 [25]:
```
current_df["Confirmed"] = current_df["Confirmed"].apply(np.abs)
current_df["Deaths"] = current_df["Deaths"].apply(np.abs)
```

In [26]:
```
data = pd.read_csv(name_number)
```

In [27]:
```
data.head()
```

Out[27]:

	Province_State	Country_Region	Confirmed	Deaths
0	NaN	Afghanistan	27823	395
1	NaN	Albania	102151	331
2	NaN	Algeria	48728	592
3	NaN	Andorra	21768	13
4	NaN	Angola	19903	140

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

In [28]:
```
print(data.shape)
```

```
(4016, 4)
```

There are total 4016 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 [29]:
```
print(data.count())
```

```
Province_State    3837
Country_Region    4016
Confirmed         4016
Deaths            4016
dtype: int64
```

Printing how many null values exist in the dataset.

In [30]:
```
data.isnull().sum()
```

Out[30]:
```
Province_State    179
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 [31]:
```
data.loc[data['Province_State'].isnull(),'Province_State'] = data['Country_Region']
```

In [32]:
```
data.head()
```

Out[32]:

	Province_State	Country_Region	Confirmed	Deaths
0	Afghanistan	Afghanistan	27823	395
1	Albania	Albania	102151	331
2	Algeria	Algeria	48728	592
3	Andorra	Andorra	21768	13
4	Angola	Angola	19903	140

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

In [33]:
```
states = data['Province_State'].unique()
print("Number of unique States - ", len(states))
```

```
Number of unique States -  774
```

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

In [34]:
```
countries = data["Country_Region"].unique()
print("Number of unique countries - ", len(countries))
```

```
Number of unique countries -  201
```

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 [35]:
```
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 [36]:
```
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
```

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 [37]:
```
#data['Province_State'].value_counts()
```

In [38]:
```
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%|████████████████████████████████████████████████████████████████████████████████
| 774/774 [06:30<00:00,  1.98it/s]
Locations are not found for -  ['Repatriated Travellers', 'W.P. Kuala Lumpur', 'Sakha
(Yakutiya) Republic', 'Summer Olympics 2020']
```

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

In [39]:

```
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 [40]:

```
data.head()
```

Out[40]:

	Province_State	Country_Region	Confirmed	Deaths	Latitude	Longitude
0	Afghanistan	Afghanistan	27823	395	33.768006	66.238514
1	Albania	Albania	102151	331	41.000028	19.999962
2	Algeria	Algeria	48728	592	28.000027	2.999983
3	Andorra	Andorra	21768	13	42.540717	1.573203
4	Angola	Angola	19903	140	-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 any differences in values

In [41]:

```
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), ('W.P.
Kuala Lumpur', nan, None), ('Sakha (Yakutiya) Republic', nan, None), ('Summer Olympic
s 2020', nan, None)]


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

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

In [42]:
```
#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 [43]:
```
data.shape
```

Out[43]:
```
(4012, 6)
```

We have 4012 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 [44]:
```
clustering_data = data[["Confirmed", "Deaths"]]
```

Let's visualise our clustering dataframe

In [45]:
```
clustering_data.head()
```

Out[45]:

	Confirmed	Deaths
0	27823	395
1	102151	331
2	48728	592
3	21768	13
4	19903	140

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. For that, we use the StandardScaler.

In [46]:

```
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.

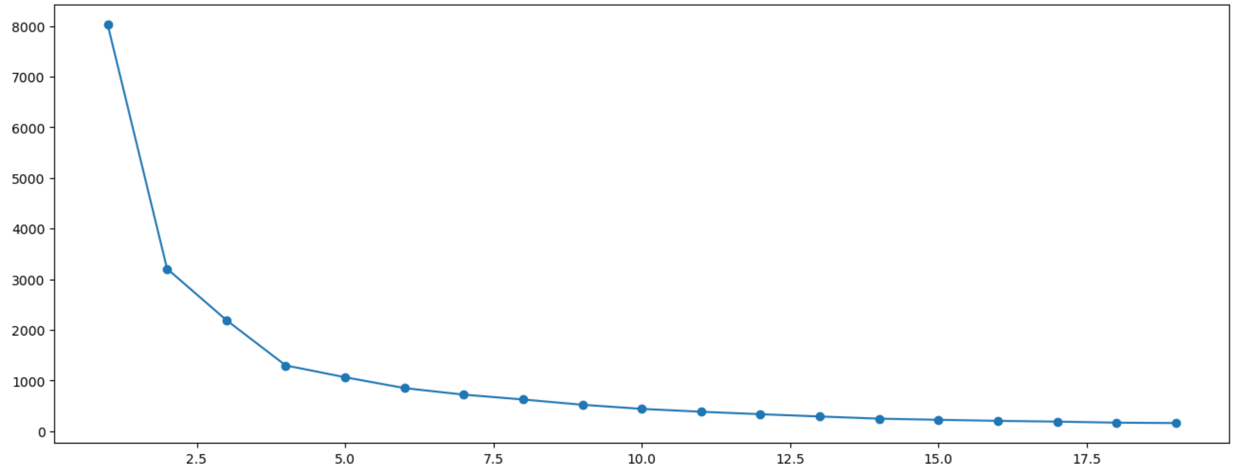
In [47]:

```
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 [48]:
```
# 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
```

Let's see how our clusters look now

In [49]:
```
clustering_data.head(10)
```

Out[49]:

	Confirmed	Deaths	cluster
0	27823	395	1
1	102151	331	1
2	48728	592	1
3	21768	13	1
4	19903	140	1
5	4272	119	1
6	5665655	117037	2
7	9215327	121394	2
8	420446	8618	1
9	36169	573	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 [51]:

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

Out[51]:

cluster	Confirmed	Deaths
1	54478.1	698.7
2	6130544.1	111566.2
3	21975941.2	110864.5
4	1704265.1	22445.8

As you can see our cluster 2 has the 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 [52]:

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

Out[52]:

	Province_State	cluster
6	Argentina	2
7	Armenia	2
65	Sao Paulo	2
66	Sergipe	2
216	France	2
266	Ladakh	2
269	Maharashtra	2
270	Manipur	2
286	Indonesia	2
288	Iraq	2
499	Madre de Dios	2
510	Poland	2
512	Qatar	2
614	South Africa	2
616	Andalusia	2
672	Turkey	2
3986	England	2
4006	Winter Olympics 2022	2

Similarly, to get an idea about the members in cluster 3, let's view them using the following code.

In [53]:

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

Out[53]:

	Province_State	cluster
217	Gabon	3
368	Kuwait	3
674	Alabama	3
3989	Isle of Man	3

Cluster 3 have the largest number of confirmed cases and second largest number of deaths

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.

Discuss what do you think about other clusters?

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

In [54]:
```
#Cluster 1
data['cluster'] = cluster
clusters = data[['Province_State', 'cluster']]
clusters.loc[clusters['cluster'] == 1]
```

Out[54]:

	Province_State	cluster
0	Afghanistan	1
1	Albania	1
2	Algeria	1
3	Andorra	1
4	Angola	1
...	...	...
4011	Unknown	1
4012	Nauru	1
4013	Niue	1
4014	Tuvalu	1
4015	Pitcairn Islands	1

3870 rows × 2 columns

Cluster 1 has the lowest number of confirmed Covid cases and lowest number of deaths

In [55]:
```
#Cluster 4
data['cluster'] = cluster
clusters = data[['Province_State', 'cluster']]
clusters.loc[clusters['cluster'] == 4]
```

Out[55]:

	Province_State	cluster
10	Northern Territory	4
15	Western Australia	4
17	Azerbaijan	4
20	Bangladesh	4
21	Barbados	4
...	...	...
2574	New York	4
3442	Texas	4
3993	Scotland	4
3998	Uzbekistan	4
4001	Vietnam	4

120 rows × 2 columns

Cluster 4 has the third largest of confirmed Covid cases and third largest number of deaths

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

In [56]:

```
data.head()
```

Out[56]:

	Province_State	Country_Region	Confirmed	Deaths	Latitude	Longitude	cluster
0	Afghanistan	Afghanistan	27823	395	33.768006	66.238514	1
1	Albania	Albania	102151	331	41.000028	19.999962	1
2	Algeria	Algeria	48728	592	28.000027	2.999983	1
3	Andorra	Andorra	21768	13	42.540717	1.573203	1
4	Angola	Angola	19903	140	-11.877577	17.569124	1

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

In [57]:
```
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))
```

Let's see how our dataframe looks now.

In [58]:
```
data.head(10)
```

Out[58]:

	Province_State	Country_Region	Confirmed	Deaths	Latitude	Longitude	cluster	color
0	Afghanistan	Afghanistan	27823	395	33.768006	66.238514	1	green
1	Albania	Albania	102151	331	41.000028	19.999962	1	green
2	Algeria	Algeria	48728	592	28.000027	2.999983	1	green
3	Andorra	Andorra	21768	13	42.540717	1.573203	1	green
4	Angola	Angola	19903	140	-11.877577	17.569124	1	green
5	Antigua and Barbuda	Antigua and Barbuda	4272	119	17.223472	-61.955461	1	green
6	Argentina	Argentina	5665655	117037	-34.996496	-64.967282	2	darkred
7	Armenia	Armenia	9215327	121394	40.769627	44.673665	2	darkred
8	Australian Capital Territory	Australia	420446	8618	-35.488350	149.002694	1	green
9	New South Wales	Australia	36169	573	-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. The map will be saved in the same folder as this notebook. It is named as "covid_map.html" and you can open it using a web browser such as chrome.

In [59]:
```
#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 longitude
 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 datafram
```

```
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

This reports provides a detailed analysis steps used in pre-processing of the raw data to ensure data is clean to be analysed. The aim is to group the data into 4 clusters based on confirmed cases and number of deaths from the worst hit states to the least hit. The first step we did was extraction of the data using the link stated in the hypothesis part of the work then we substracted the number of confirmed cases from the number of deaths from the recent and previous data. We then saved the data in a new csv file with my name and a random number. Next step was to check for missing values where i noticed Province_State had some missing values and we filled the missing values using the values in Country/Region. Then we proceeded to check for unique state for Province_state where i found 771 for 2021 part A data and 774 for 2022 part B data. Next step was to retrive the coordinate using geopy library and monitored the progress of this action using https://tqdm.github.io/ which iterated through the states in our dtataset and retrieved the coordinates from the get_lat_lon method then appended the coordinates into the dataset. Then i furthered checked if the coordinates retrieved from geopy corresponds with the coordinate in the dataset and there was no difference. Next step is to only select rows that have cordinates lat and long and viewed our data shape and proceeded to Clustering the data based on confirmed deaths and saved data in a new dataframe. The clustering will take two features, number of cases and number of deaths. These two features were on different scales as number of cases is usually greater than number of deaths and 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 performed a normalisation step on the clustering dataset using StandardScaler. Then we used the elbow method which involves plotting within cluster sum of squares against the number of clusters and then selected the number of clusters which produced the values 1,2,3,4 based on the clustering algorithm. Then we proceeded to allocate the 4 clusters color legends of dark red, green, orange and yellow and visualised the clusters on a map using the legends to different. The map shows the distribution of confirmed deaths according to states.

Finally, in comparing the result of the clustering using the map i agree with the outcome of the analysis and further reference the online newsletters used to draw this conclusion.

Reference

Web Reference

https://github.com/CSSEGISandData/COVID-19/blob/master/csse_covid_19_data/csse_covid_19_daily_reports/01-01-2021.csv
https://www.nature.com/articles/s12276-021-00604-z
https://www.gov.uk/government/publications/covid-19-reported-sars-cov-2-deaths-in-england/covid-19-confirmed-deaths-in-england-report?ref=pmp-magazine
https://www.theguardian.com/world/2021/dec/01/covid-world-map-which-countries-have-the-most-coronavirus-vaccinations-cases-and-deaths