COVID-19 Contact Tracing using Machine Learning

WQD7006: Machine Learning for Data Science

Student Name: Norzarifah Kamarauzaman

Student ID: 17041741

Introduction

Case Investigation and **Contact Tracing** are the two-pronged strategy recommended by the Centers for Disease Control and Prevention (CDC) to alleviate and eventually halting COVID-19 transmission. Contact tracing is to be conducted following case investigation i.e. those with confirmed and probable diagnoses of COVID-19 need to recall everyone they had close contact with during the time when they may have been infectious for subsequent identification, monitoring, and support of the close contacts who have been exposed to and possibly infected with the virus.

In addition to a range of guidance documents, fact sheets and resources on how to conduct contact tracing, an integration with technology has made the process of contact tracing more efficient and give accurate results than if it was to be performed manually. Data collection via contact-tracing apps for example have been introduced in most countries to automate and simplify the tracking process. Subsequently, an application of machine learning algorithms seems crucial so that the vastly-collected data can be analysed to discover inherent patterns or groupings present in the data; for a data-driven decision-making.

Background ¶

An unsupervised machine learning method such as **clustering algoritms** are meant for pattern detection and descriptive modelling of which they ingest unlabelled data to detect patterns and summarise or group similar data points. For contact tracing purpose, a *Density-based Clustering* method such as **Density-Based Spatial Clustering of Applications with Noise (DBSCAN)** algorithm is the most appropriate in comparison to other clustering methods such as *Hierarchal-based Clustering* and *Partitioning-based Clustering*. In density-based clustering, clusters are formed based on the density of the region and for this reason, it fits the purpose. As diseases are transferred when an infected person comes in contact with others, more crowded areas will have more cases than less crowded ones.

Objective

To identify close contacts of an infected or a suspected COVID-19 person by leveraging on density-based clustering method.

Data Dimension

Although contact-tracing apps have been used extensively for data collection, the real-life data are hardly accessible due to data protection issues. For this project, a mock dataset has been generated here (https://www.mockaroo.com/). It contains **200 records** of **time and location** for **15 individuals** in a confined space with an **area of 1000m^2** within a specific timeframe (from **10:00 am until 10:00 pm**) in a particular day.

Information on each features are as follows:

- **personal_id:** a unique identifier to label each individual
- **timestamp:** record date and time of an individual present in the space
- **latitude:** record lateral position of an individual present in the space
- **longitude:** record vertical postion of an individual present in the space

Both latitude and longitude features represent the X and Y location of a person in terms of grid. In real-life, government or public health department often have access to GPS locations of the public captured via contact-tracing apps. In short, the dataset it created to simulate a situation in a commercial area e.g. supermarket, shopping mall etc. or a work area such as office, factory and warehouses.

All python code and dataset are available in GitHub (https://github.com/norzarifah/WQD7006-Assignment)

Methodology:

In [1]:
```
# import libraries: data manipulation
import numpy as np
import pandas as pd

# import libraries: data visualisation
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# import libraries: data pre-processing and modeling
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import DBSCAN
from sklearn import metrics

# clear warnings
import warnings
warnings.filterwarnings(action='ignore',category=FutureWarning)
```

In [2]:

```
# pre-define functions

# to view distribution of data points in a given space per individual
def scatterplot_pre_modeling(data):
    fig = plt.figure(figsize=(10,10))
    plt.rc('grid', linestyle=':', color='grey', linewidth=1)
    ax=sns.scatterplot(x='latitude', y='longitude', data=data, hue='personal_i
d', s=120, alpha=0.6)
    sorted_legend = sorted(data['personal_id'].unique())
    plt.legend(sorted_legend, bbox_to_anchor= [1, 1])
    plt.grid(True)

# to view lateral and vertical variation of each individuals in a given space
def boxplot_distribution(data):
    sorted_legend = sorted(data['personal_id'].unique())
    fig , ax = plt.subplots(nrows = 2, ncols = 1, figsize=(12,12))
    plt.subplots_adjust(hspace = 0.4)
    # boxplot for latitude
    sns.boxplot(ax=ax[0], x='personal_id', y='latitude', data=data, palette='S
pectral')
    ax[0].set_title("Variation in lateral position of each individual within t
he space", y=1.05)
    ax[0].set_xticklabels([i for i in sorted_legend])
    # boxplot for longitude
    sns.boxplot(ax=ax[1], x='personal_id', y='longitude', data=data, palette=
'Spectral')
    ax[1].set_title("Variation in vertical position of each individual within
 the space", y=1.05)
    ax[1].set_xticklabels([i for i in sorted_legend])

# to view distribution of clusters and noise points after modeling
def scatterplot_post_modeling(data, model):
    data = data.copy()
    labels = model.labels_
    fig = plt.figure(figsize=(10,10))
    ax=sns.scatterplot(data['latitude'], data['longitude'], hue = ['cluster-{}
'.format(x) for x in labels], s=120, alpha=0.6)
    plt.legend(bbox_to_anchor = [1, 1])

# to output a dataframe with cluster column
def cluster_output(data, model):
    data['cluster'] = model.labels_.tolist()
    ids = data[(data['cluster'] == -1)].index
    data.drop(ids, inplace = True)
    return data

# to view distribution of clusters without noise points after modeling
def scatterplot_clusters(data, model):
    cluster_output(data, model)
    fig = plt.figure(figsize=(10,10))
    ax=sns.scatterplot(data['latitude'], data['longitude'], hue = ['cluster-{}
'.format(x) for x in data['cluster']], s=120, alpha=0.6)
    plt.legend(bbox_to_anchor = [1, 1])
    output = pd.DataFrame(data)
```

```
# to list all individuals in a selected cluster
def get_allin_cluster(cluster_no):
    all_in_cluster = []
    if cluster_no != -1:
        all_in_cluster.append(df.loc[df['cluster'] == cluster_no, 'personal_i
d'].to_list())
        return all_in_cluster
    else:
        return 'Not a valid cluster'

# to identify all clusters that an individual is belonged to
def get_close_contact(personal_id):
    possible_cluster = []
    data = df[df['personal_id']==personal_id]
    possible_cluster.append(data['cluster'].sort_values().to_list())
    return possible_cluster
```

In [3]:
```
# load data: read csv file
path = 'contact_tracing_mock_data.csv'
df = pd.read_csv(path)
```

Step 2: Conduct Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a preliminary approach in data analysis. The purposes of EDA are to explore and understand relationships among variables and to identify problems such as presence of outliers and missing data. Both of Outlier Analysis and Missing Data Analysis require futher data manipulation if necessary.

In [4]:
```
# view random samples of data
df.sample(n=7, random_state=123)
df.head()
```

Out[4]:

	personal_id	timestamp	latitude	longitude
0	ID003	14:21:14	95	52
1	ID007	12:59:47	90	65
2	ID004	14:35:45	77	36
3	ID014	13:49:18	73	27
4	ID001	19:32:00	53	61

In [5]:
```
# explore data type, data dimension and missing data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   personal_id  200 non-null    object
 1   timestamp    200 non-null    object
 2   latitude     200 non-null    int64 
 3   longitude    200 non-null    int64 
dtypes: int64(2), object(2)
memory usage: 6.4+ KB
```

Preliminary findings

- There are 200 entries, with no missing rows or records.
- There are four features of which two are of categorical type and the other two are numerical.
- Conversion of *timestamp* feature from categorical object to datetime format. Check the changes.

In [6]:
```
# convert datetime object to datetime type
df['timestamp'] = pd.to_datetime(df['timestamp'])
```

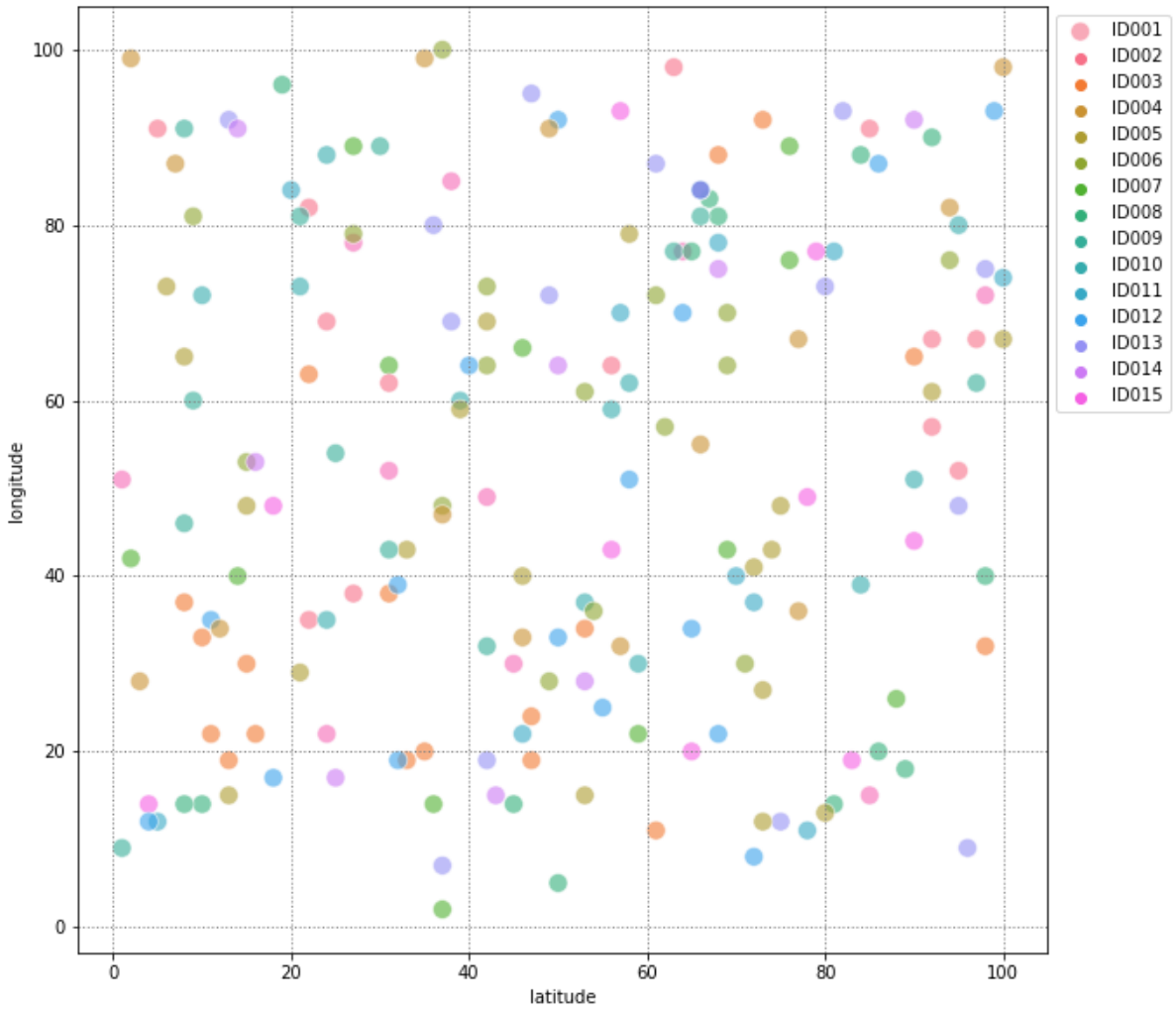
In [7]:
```
# check data dimension: changed timestamp Dtype
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype         
---  ------       --------------  -----         
 0   personal_id  200 non-null    object        
 1   timestamp    200 non-null    datetime64[ns]
 2   latitude     200 non-null    int64         
 3   longitude    200 non-null    int64         
dtypes: datetime64[ns](1), int64(2), object(1)
memory usage: 6.4+ KB
```

Data visualisation

To further understand the distribution of each points, it is best to visualise the data on a scatter plot.

In [8]:
```
# visualise the distribution of data points per personal_id
scatterplot_pre_modeling(df)
```



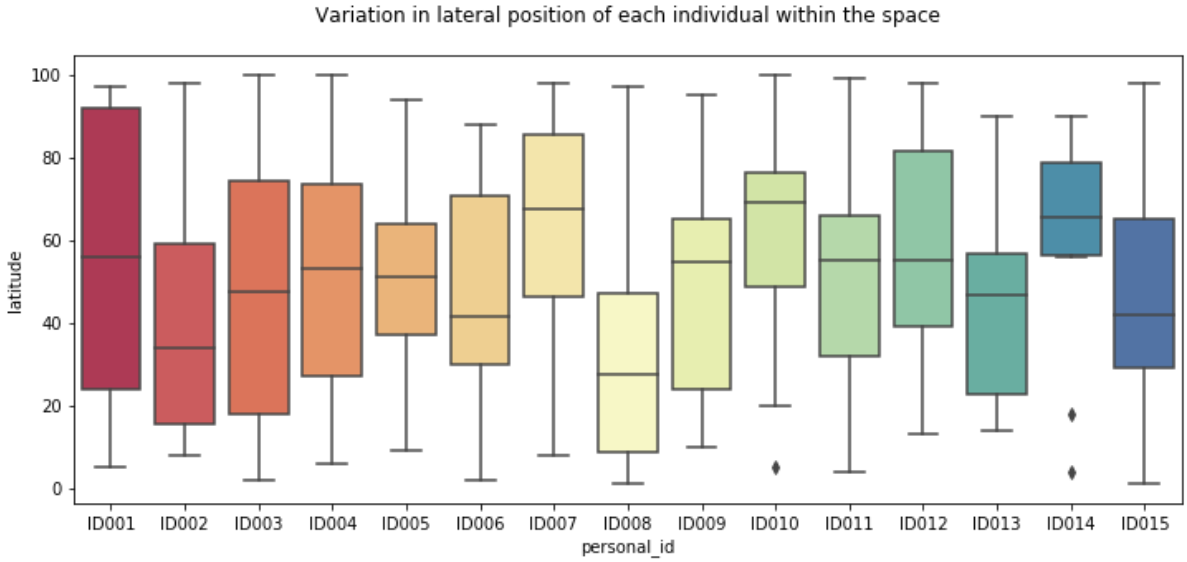
In [9]:
```
# explore numerical variables: statistical summary
df.groupby('personal_id').describe()
```

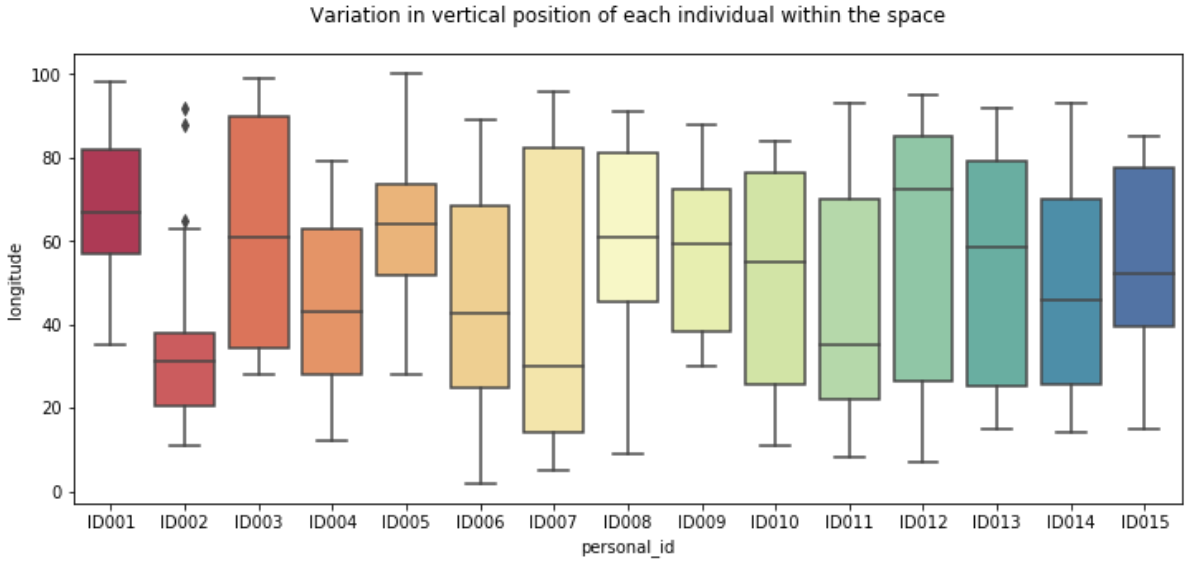
Out[9]:

	latitude								longitude		
	count	mean	std	min	25%	50%	75%	max	count	mean	std
personal_id											
ID001	16.0	49.437500	22.039264	9.0	37.00	51.0	63.75	94.0	16.0	62.000000	19.
ID002	12.0	46.750000	26.938060	2.0	30.00	41.5	70.75	88.0	12.0	47.750000	29.
ID003	13.0	54.692308	34.235422	5.0	24.00	56.0	92.00	97.0	13.0	67.153846	19.
ID004	14.0	47.285714	33.181154	2.0	17.75	47.5	74.25	100.0	14.0	63.428571	28.
ID005	10.0	59.700000	28.871555	5.0	48.75	69.0	76.50	100.0	10.0	50.500000	29.
ID006	17.0	51.176471	25.807546	4.0	32.00	55.0	66.00	99.0	17.0	46.176471	29.
ID007	18.0	40.611111	28.297036	8.0	15.25	34.0	59.00	98.0	18.0	37.111111	23.
ID008	12.0	33.416667	28.974780	1.0	8.75	27.5	47.25	97.0	12.0	60.416667	25.
ID009	14.0	60.642857	27.006003	13.0	39.00	55.0	81.50	98.0	14.0	59.357143	33.
ID010	8.0	44.875000	26.319127	14.0	22.75	46.5	56.75	90.0	8.0	54.375000	31.
ID011	10.0	59.600000	28.084001	4.0	56.25	65.5	78.75	90.0	10.0	49.100000	27.
ID012	14.0	61.571429	30.778900	8.0	46.25	67.5	85.50	98.0	14.0	46.714286	36.
ID013	11.0	47.363636	28.468483	1.0	29.00	42.0	65.00	98.0	11.0	55.909091	25.
ID014	19.0	51.210526	29.417642	6.0	27.00	53.0	73.50	100.0	19.0	44.578947	21.
ID015	12.0	51.083333	28.398010	10.0	24.00	54.5	65.25	95.0	12.0	57.166667	19.

Next, a quick look at the statistical summary of both of the numerical features i.e. *latitude* and *longitude* denotes that each **ID014** and **ID010** has the most and the least number of data points respectively. The box plots illustrates the variation in lateral and vertical position of each individuals within the given space. Although there are some presence of outliers i.e. **ID010**, **ID014** and **ID002**; those data points are still admissible as long as they fall within the specified latitude and longitude range.

In [10]:
```
# explore numerical variables: via visualisation
boxplot_distribution(df)
```





Step 3: Applying DBSCAN Algorithm

As mentioned earlier, **DBSCAN** algorithm establishes high-density regions as clusters which are separated by regions of low density. The algorithm requires two important parameters to determine the clusters:

<ul style="font-family: calibri; font-size:12pt; text-align: justify; margin-right:15px"">
- **eps:** the maximum distance between a pair of points.
- **min_samples:** the minimum number of samples in a neighborhood for a point to be considered as a core point.

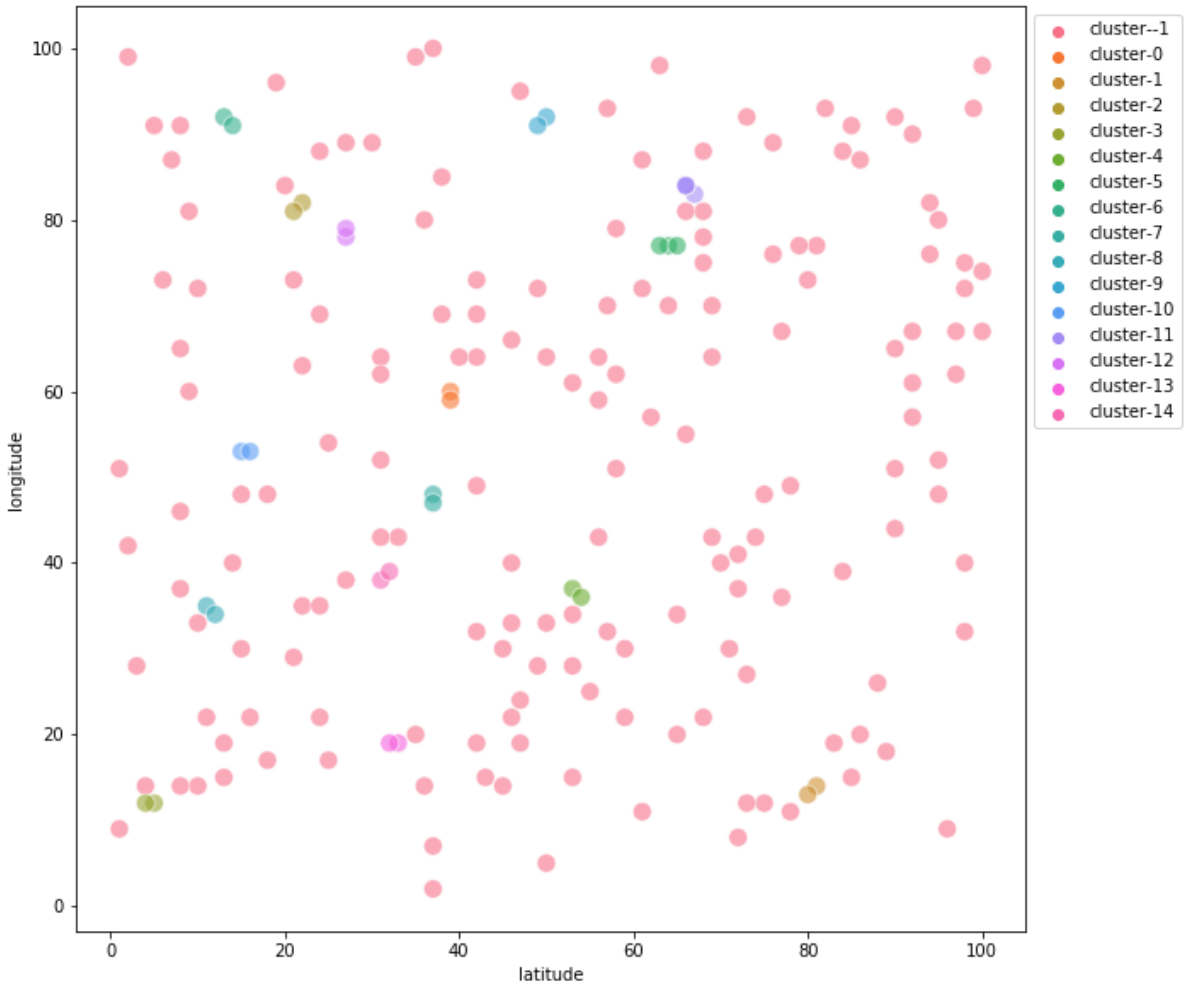
Theoretically, the algorithm begins by randomly and uniformly selecting a point from a set of data points. It checks if the selected point is a core point i.e. if the point contains the minimum number of points defined in min_samples (including itself) in its epsilon-neighborhood. Two points are considered as neighbors if and only if they are separated by a distance less than or equal to epsilon. Next, the algorithm finds the connected components of all the core points, ignoring non-core points. Each non-core points are assigned to the nearest cluster if the cluster is its epsilon-neighbor or otherwise, it is assigned as noise. The algorithm stops when it explores all the points one by one and classifies them as either **core point**, **border point** or **noise point**.

In the CDC guideline, a close contact for COVID-19 is defined as anyone who was within 6 feet of an infected person for a total of 15 minutes. Adhering to this rule, let's build a clustering model with **epsilon** is set to **1.8288** (an equivalent of the 6-ft criteria in meter unit) and **min_samples** is set to **2**, to denote that any infected person can possibly infect any individual he comes in contact with.

In [11]:
```
# subset features for clustering
X = df[['latitude', 'longitude']]
```

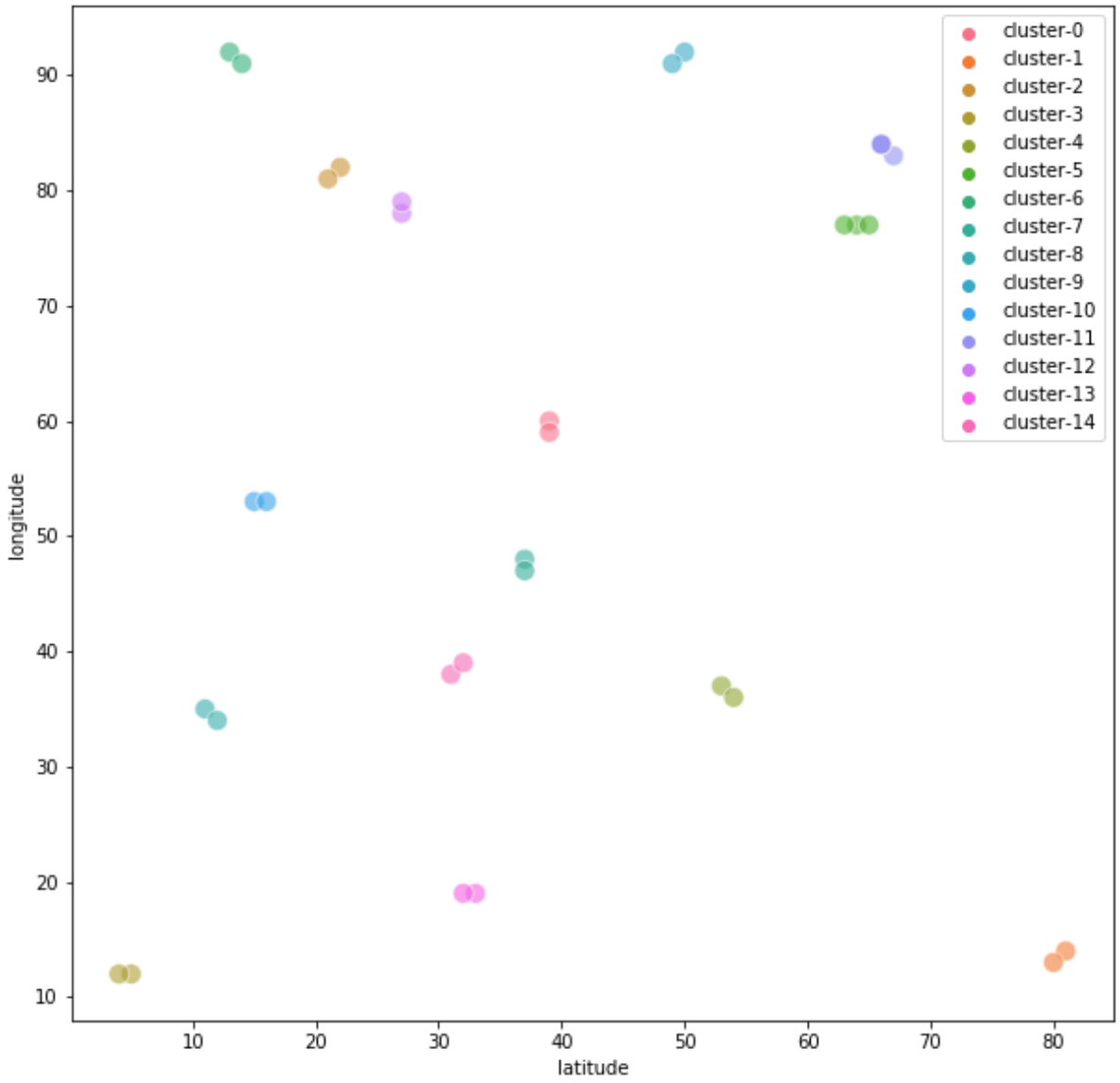
In [12]:
```
# build model
epsilon = 1.8288
model = DBSCAN(eps=epsilon, min_samples=2).fit(X)
```

In [13]:
```
# visualise the identified clusters (with noise)
scatterplot_post_modeling(df, model)
```



Any data points without a cluster or **noise point** is assigned as **cluster -1**. The presence of **15 clusters** can be clearly seen after removing noise points.

In [14]:
```
# visualise the identified clusters (without noise)
scatterplot_clusters(df, model)
```

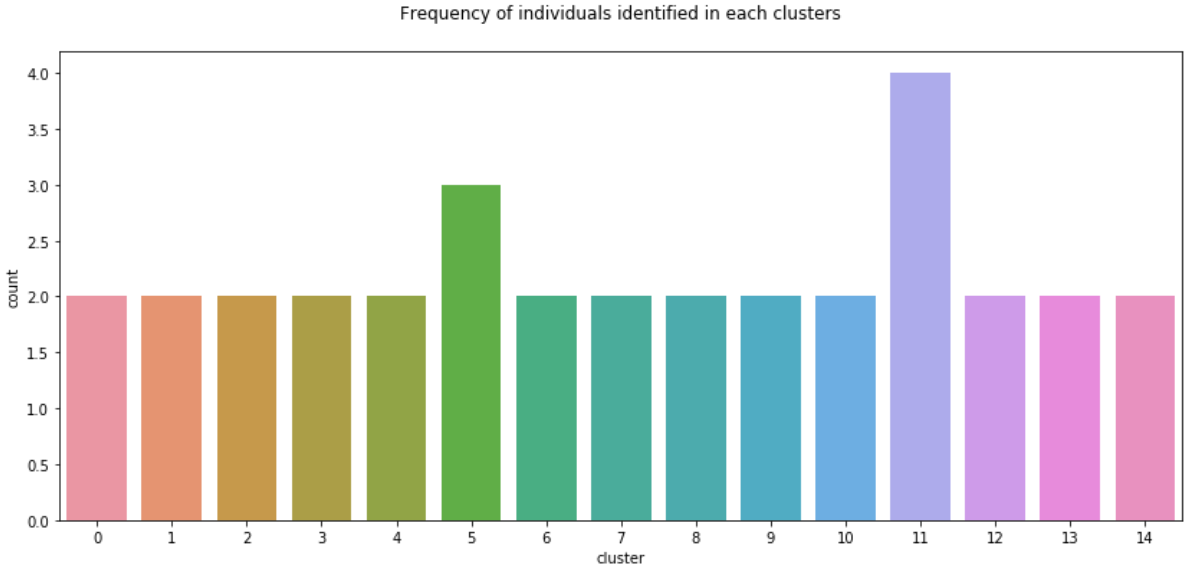


Model interpretation

The following barplot further shows the count of individuals in each clusters - so now we know **cluster 11 has the highest number of close contacts**.

In [15]:
```
# count the number of individuals present in each idenfied clusters
fig = plt.figure(figsize=(14,6))
ax=sns.countplot(x='cluster', data=df)
ax.set_title("Frequency of individuals identified in each clusters", y=1.05)
```

Out[15]: Text(0.5, 1.05, 'Frequency of individuals identified in each clusters')



If one person in a cluster is infected with COVID-19, the other contacts may have contracted the disease as well. Once contacts are identified, public health departments need to notify them of a potential exposure, and work together to help prevent further spread of the disease. This typically involves a period of self-isolation.

Dive deeper into **cluster 5** and **cluster 11** i.e. the top two clusters with the highest frequency of close contacts, both have **ID012** and **ID013** as the common contacts. Further checking reveals that these two individuals belong to other clusters as well i.e. cluster 1 and 12. Just imagine the severity of COVID-19 transmission when either ID012 or ID012 person is infected with the virus! 😨

In [16]:
```
# call function: to list all individuals in a selected cluster
cluster_no=11
print("{} belong to cluster {}".format(get_allin_cluster(cluster_no), cluster_
no))

cluster_no=5
print("{} belong to cluster {}".format(get_allin_cluster(cluster_no), cluster_
no))
```

```
[['ID012', 'ID013', 'ID011', 'ID006']] belong to cluster 11
[['ID013', 'ID008', 'ID012']] belong to cluster 5
```

In [17]:
```
# call function: to identify which cluster(s) an individual belong to
personal_id='ID012'
print("{} belong to cluster {}".format(personal_id, get_close_contact(personal
_id)))

personal_id='ID013' # adjust personal_id
print("{} belong to cluster {}".format(personal_id, get_close_contact(personal
_id)))
```

```
ID012 belong to cluster [[1, 5, 11]]
ID013 belong to cluster [[5, 11, 12]]
```

Model performance

To understand how well the model has performed with the pre-defined parameters, its performance must be evaluated using a metric. In this situation, **silhouette coefficient** is regarded as the metric as it gives an idea of separation between clusters without the need to know the labeling of the dataset.

The score is bounded between **-1** for incorrect clustering and **+1** for highly dense clustering. Scores around **0** indicate overlapping clusters.

In [18]:
```
# model evaluation:

# count the number of clusters in labels (ignoring noise if present)
labels = model.labels_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

# print output
print('Estimated number of clusters: %d' % n_clusters_)
print("Silhouette coefficient: %0.3f" % metrics.silhouette_score(X, model.labe
ls_.tolist(), metric='euclidean'))
```

```
Estimated number of clusters: 15
Silhouette coefficient: -0.453
```

With a pre-defined epsilon value of 1.8288, the metric gives negative value, thus implying a slightly incorrect clustering. 😔. As the coefficient value is closer to zero (than -1), the model mostly detecting overlapping clusters as seen in the previous scatter plot.

Step 4: Model improvement

Feature scaling is one of the methods that can be used to improve model performance however as both latitude and longitude features have the same quantitative measurements and in similar magnitudes, this approach seems redundant.

Other option is to identify the **optimal value of epsilon** using K-Nearest Neighbors algorithm and Elbow method, in three steps:

1. Calculate the distance between each point and its nearest neighboring points.
2. Sort and plot the output onto K-distance graph @ elbow plot.
3. Take the point of maximum curvature as the optimal value of epsilon.

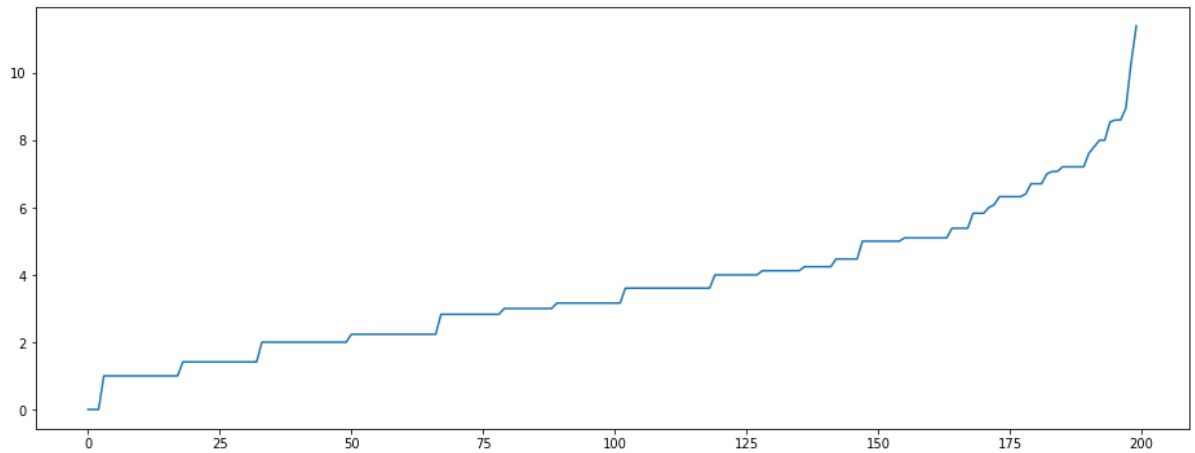
In [19]:
```
# resubset data
df_reload = pd.read_csv(path)
X = df_reload[['latitude', 'longitude']]

# step 1: calculate the distances to the nearest neighbors
knn = NearestNeighbors(n_neighbors=2)
neighbors = knn.fit(X)
distances, indices = neighbors.kneighbors(X)

# step 2: sort and plot the distances
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.figure(figsize=(16,6))
plt.rc('grid', linestyle=':', color='grey', linewidth=1)
plt.plot(distances)

# step 3: identify the optimal epsilon value from plot i.e. the elbow
```

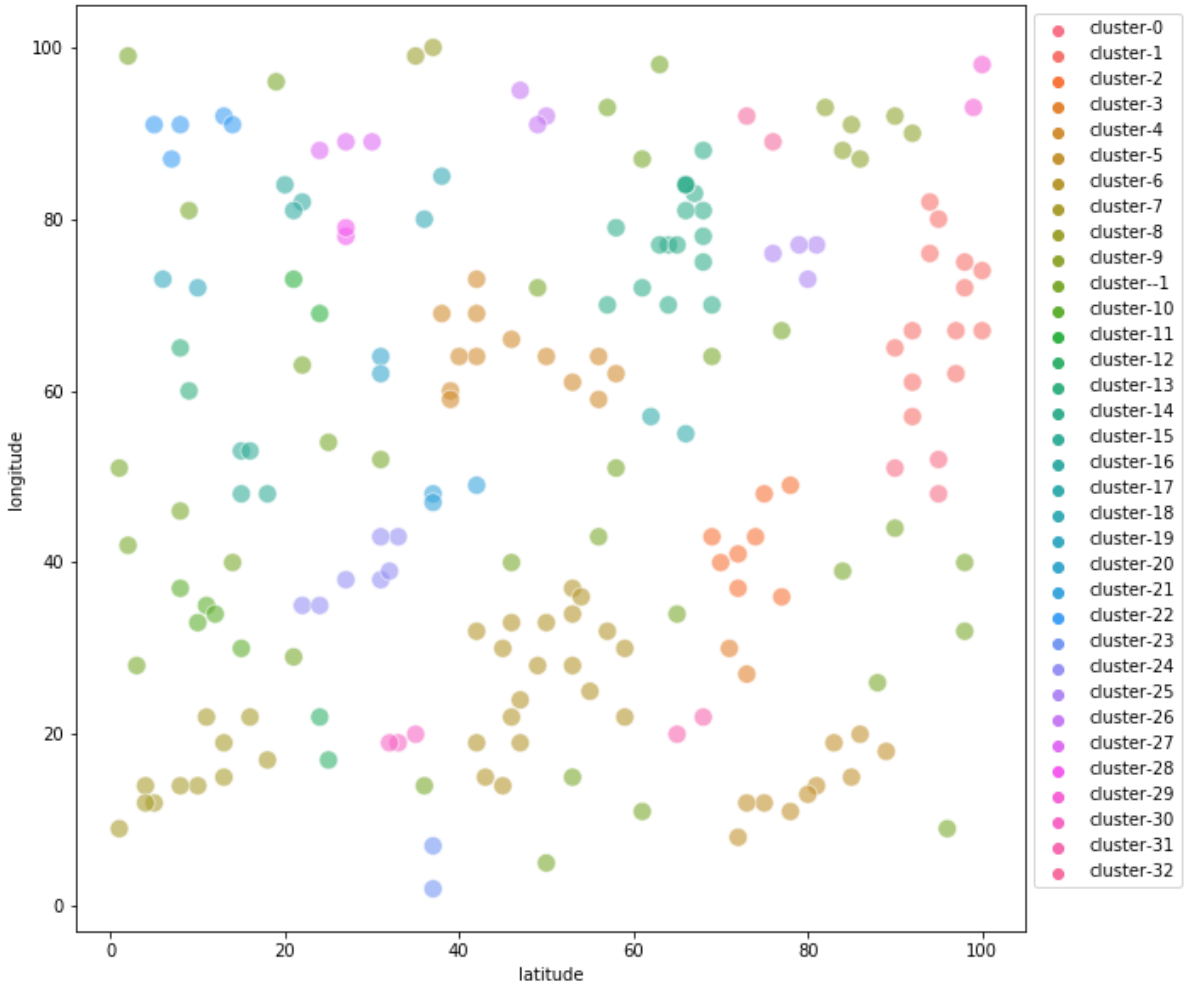
Out[19]:
```
[<matplotlib.lines.Line2D at 0x17d5f07fb08>]
```



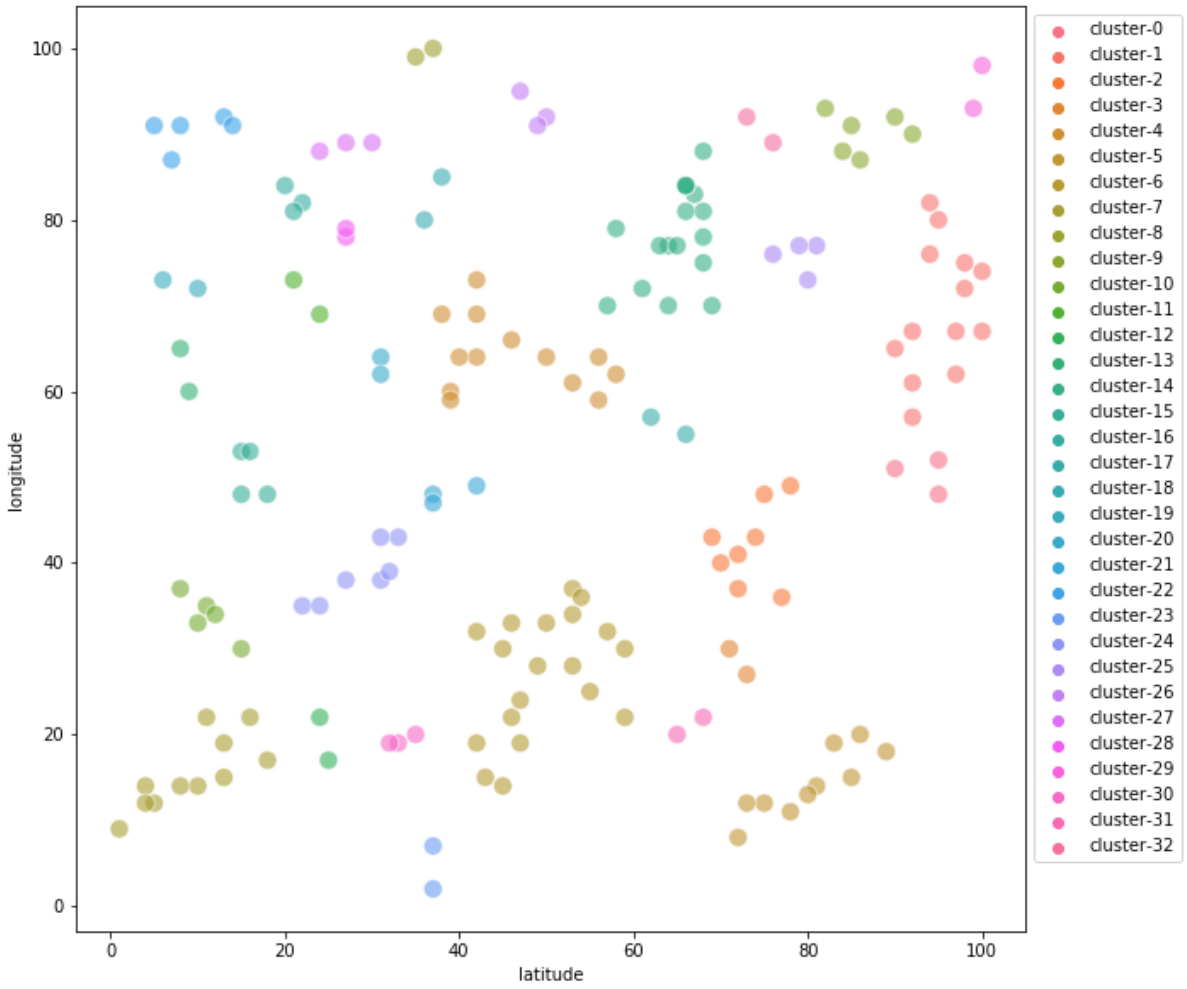
The value of y-axis at the point of maximum curvature @ the epsilon value is **5.5**. Let's update the model and evaluate model performance.

In [20]:
```
# update model
epsilon = 5.5
model_updated = DBSCAN(eps=epsilon, min_samples=2).fit(X)
```

In [21]:
```
# visualise the identified clusters (with noise)
scatterplot_post_modeling(df_reload, model_updated)
```

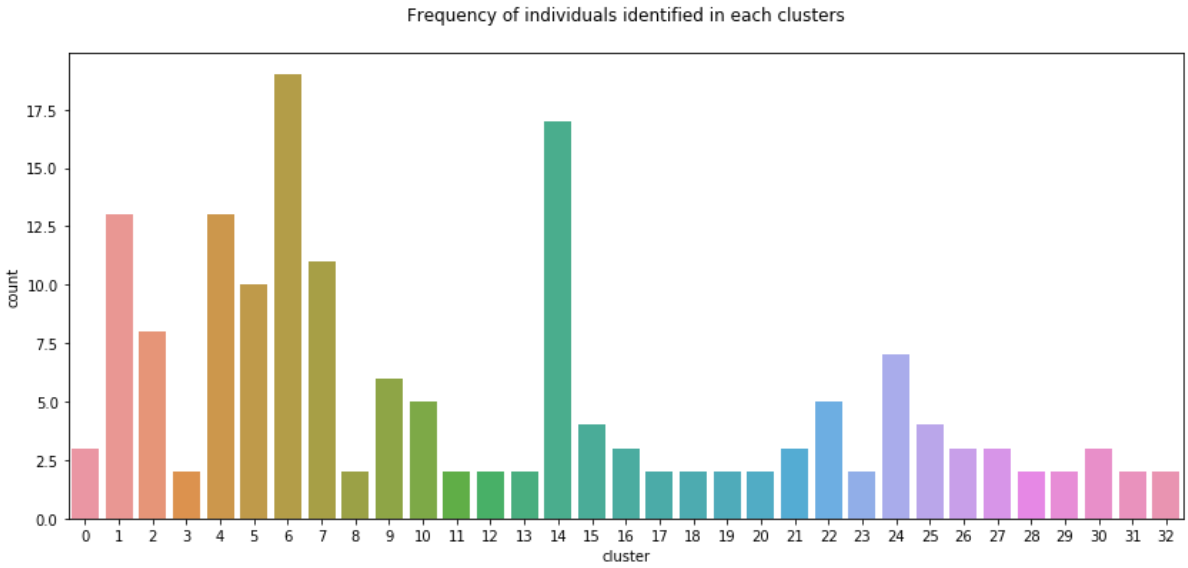


In [22]:
```
# visualise the identified clusters (without noise)
scatterplot_clusters(df_reload, model_updated)
```



In [23]:
```
# count the number of individuals present in each idenfied clusters
fig = plt.figure(figsize=(14,6))
ax=sns.countplot(x='cluster', data=df_reload)
ax.set_title("Frequency of individuals identified in each clusters", y=1.05)
```

Out[23]: Text(0.5, 1.05, 'Frequency of individuals identified in each clusters')



In [24]:
```
# model evaluation:

# count the number of clusters in labels (ignoring noise if present)
labels = model_updated.labels_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

# print output
print('Estimated number of clusters: %d' % n_clusters_)
print("Silhouette coefficient: %0.3f" % metrics.silhouette_score(X, model_upda
ted.labels_.tolist(), metric='euclidean'))
```

```
Estimated number of clusters: 33
Silhouette coefficient: 0.231
```

The updated model indentifies **33 clusters** and **cluster 6** gives the highest number of close contacts.

The **silhouette coefficient** has significantly improved from **-0.453** to **0.231** thus indicating correct clustering this time, with a slightly dense clustering.

Increasing the value of epsilon from 1.828 to 5.5 increases the sparsity of clusters and some data points that were previously identified as noise points are now considered as core points. With the updated epsilon value, the criteria of defining close contacts should be revised - such that anyone within a radial distance of 5.5 meter with an infected person is probable to catch the virus too!

Conclusion

Ideally, more data and information should be incorporated in the model such as current health conditions and previous medical histories of each individuals - for more accurate identification of close contacts of an infected or suspected COVID-19 persons. In real world, such analysis should be carried out more frequently or near real-time (perhaps at every hour as anyone is possible to catch the virus after 15 minutes of contact with an infected person) with a larger group of people.

The detection of close contacts via unsupervised machine learning can be a data-driven life saving approach in curbing COVID-19 transmission. It definitely helps in alerting government and public health department, to provide immediate treatment to the potentially infected contacts.