Week 1: Find Clusters of Infected People

URGENT WARNING
We have been receiving reports from health facilities that a new, fast-spreading virus has been discovered in the population. To prepare our response, we need to understand the geospatial distribution of those who have been infected. Find out whether there are identifiable clusters of infected individuals and where they are.

Your goal for this notebook will be to estimate the location of dense geographic clusters of infected people using incoming data from week 1 of the simulated epidemic.

Imports

In [35]:
```
import cudf
import cuml

import cupy as cp
```

Load Data

Begin by loading the data you've received about week 1 of the outbreak into a cuDF data frame. The data is located at `'./data/week1.csv'`. For this notebook you will only need the `'lat'`, `'long'`, and `'infected'` columns. Either drop the columns after loading, or use the `cudf.read_csv` named argument `usecols` to provide a list of only the columns you need.

In [36]:
```
# Load the 'lat', 'long', and 'infected' columns from the CSV file into a cuDF Data
df_week1 = cudf.read_csv('./data/week1.csv', usecols=['lat', 'long', 'infected'])

# Display the DataFrame
df_week1.head()
```

Out[36]:

	lat	long	infected
0	54.522510	-1.571896	False
1	54.554030	-1.524968	False
2	54.552486	-1.435203	False
3	54.537189	-1.566215	False
4	54.528212	-1.588462	False

Make Data Frame of the Infected

Make a new cuDF data frame `infected_df` that contains only the infected members of the population.

In [37]:
```
# Filter the DataFrame to include only rows where 'infected' is greater than 0
infected_df = df_week1[df_week1['infected'] > 0]

# Display the filtered DataFrame
infected_df.head()
```

Out[37]:

	lat	long	infected
28928759	54.472766	-1.654932	True
28930512	54.529717	-1.667143	True
28930904	54.512986	-1.589866	True
28932226	54.522322	-1.380694	True
28933748	54.541660	-1.613490	True

Make Grid Coordinates for Infected Locations

Provided for you in the next cell (which you can expand by clicking on the "..." and contract again after executing by clicking on the blue left border of the cell) is the lat/long to OSGB36 grid coordinates converter you used earlier in the workshop. Use this converter to create grid coordinate values stored in `northing` and `easting` columns of the `infected_df` you created in the last step.

In [38]:
```
# https://www.ordnancesurvey.co.uk/docs/support/guide-coordinate-systems-great-brit

def latlong2osgbgrid_cupy(lat, long, input_degrees=True):
    '''
    Converts latitude and longitude (ellipsoidal) coordinates into northing and eas

    Inputs:
    lat: latitude coordinate (N)
    long: longitude coordinate (E)
    input_degrees: if True (default), interprets the coordinates as degrees; otherw

    Output:
    (northing, easting)
    '''

    if input_degrees:
        lat = lat * cp.pi/180
        long = long * cp.pi/180

    a = 6377563.396
    b = 6356256.909
    e2 = (a**2 - b**2) / a**2
```

```
    N0 = -100000 # northing of true origin
    E0 = 400000 # easting of true origin
    F0 = .9996012717 # scale factor on central meridian
    phi0 = 49 * cp.pi / 180 # latitude of true origin
    lambda0 = -2 * cp.pi / 180 # longitude of true origin and central meridian

    sinlat = cp.sin(lat)
    coslat = cp.cos(lat)
    tanlat = cp.tan(lat)

    latdiff = lat-phi0
    longdiff = long-lambda0

    n = (a-b) / (a+b)
    nu = a * F0 * (1 - e2 * sinlat ** 2) ** -.5
    rho = a * F0 * (1 - e2) * (1 - e2 * sinlat ** 2) ** -1.5
    eta2 = nu / rho - 1
    M = b * F0 * ((1 + n + 5/4 * (n**2 + n**3)) * latdiff -
                  (3*(n+n**2) + 21/8 * n**3) * cp.sin(latdiff) * cp.cos(lat+phi0) +
                  15/8 * (n**2 + n**3) * cp.sin(2*(latdiff)) * cp.cos(2*(lat+phi0)) -
                  35/24 * n**3 * cp.sin(3*(latdiff)) * cp.cos(3*(lat+phi0)))
    I = M + N0
    II = nu/2 * sinlat * coslat
    III = nu/24 * sinlat * coslat ** 3 * (5 - tanlat ** 2 + 9 * eta2)
    IIIA = nu/720 * sinlat * coslat ** 5 * (61-58 * tanlat**2 + tanlat**4)
    IV = nu * coslat
    V = nu / 6 * coslat**3 * (nu/rho - cp.tan(lat)**2)
    VI = nu / 120 * coslat ** 5 * (5 - 18 * tanlat**2 + tanlat**4 + 14 * eta2 - 58

    northing = I + II * longdiff**2 + III * longdiff**4 + IIIA * longdiff**6
    easting = E0 + IV * longdiff + V * longdiff**3 + VI * longdiff**5

    return(northing, easting)
```

In [39]:
```
infected_df['northing'], infected_df['easting'] = latlong2osgbgrid_cupy(infected_df

# Optionally, drop the original lat and long columns if they are no longer needed
infected_df.drop(columns=['lat', 'long'], inplace=True)

# Display the updated DataFrame
print(infected_df.head())
```

```
          infected       northing        easting
28928759      True  508670.060234  422359.759523
28930512      True  515002.666798  421538.547038
28930904      True  513167.535850  426549.874086
28932226      True  514305.280055  440081.234798
28933748      True  516349.132042  425003.005560
```

Find Clusters of Infected People

Use DBSCAN to find clusters of at least 25 infected people where no member is more than 2000m from at least one other cluster member. Create a new column in `infected_df`

which contains the cluster to which each infected person belongs.

In [40]:
```
from cuml.cluster import DBSCAN
import cupy as cp

# Combine 'easting' and 'northing' into a 2D array for DBSCAN
coords = cp.vstack((easting, northing)).T

# Set the parameters for DBSCAN: eps=2000 meters, min_samples=25
dbscan = DBSCAN(eps=2000, min_samples=25)

# Fit the DBSCAN model on the 'easting' and 'northing' coordinates
clusters = dbscan.fit_predict(coords)

# Add a new column 'cluster' to infected_df for the cluster labels
infected_df['cluster'] = clusters

# Display the updated DataFrame
infected_df.head()
```

Out[40]:

	infected	northing	easting	cluster
28928759	True	508670.060234	422359.759523	-1
28930512	True	515002.666798	421538.547038	-1
28930904	True	513167.535850	426549.874086	-1
28932226	True	514305.280055	440081.234798	-1
28933748	True	516349.132042	425003.005560	-1

Find the Centroid of Each Cluster

Use grouping to find the mean `northing` and `easting` values for each cluster identified above.

In [41]:
```
# Calculate the centroids of each cluster
centroids = infected_df.groupby('cluster').agg({'easting': 'mean', 'northing': 'mea

# Rename the columns for clarity
centroids.columns = ['cluster', 'centroid_easting', 'centroid_northing']

# Display the centroids DataFrame
centroids.value_counts
```

Out[41]:
```
<bound method DataFrame.value_counts of     cluster  centroid_easting  centroid_no
rthing
0        10     435937.780795      334208.230907
1         6     406985.282976      434970.334950
2        11     391901.512758      300567.933051
3         9     409583.740733      417322.517251
4         4     431158.142881      391630.079963
5        -1     401877.070477      378085.504251
6         5     426559.091880      386471.292123
7         1     332980.455514      436475.467158
8        13     394518.294994      289854.874937
9         8     414765.634582      415807.314112
10        7     410069.665645      412772.647531
11        2     389386.821165      347062.237166
12        3     379638.020073      359668.638420
13        0     371410.022807      397661.052147
14       12     401640.667572      291539.411185>
```

Find the number of people in each cluster by counting the number of appearances of each cluster's label in the column produced by DBSCAN.

In [42]:
```
# Count the number of appearances of each cluster label using groupby
cluster_counts = infected_df.groupby('cluster').size().reset_index(name='count')

# Display the cluster counts
print(cluster_counts)
```

```
    cluster  count
0        10     64
1         6     27
2        11     68
3         9     21
4         4     66
5        -1   8449
6         5     43
7         1     68
8        13     71
9         8     94
10        7     39
11        2    403
12        3     25
13        0   8638
14       12     72
```

Take the Assessment

After completing the work above, visit the *Launch Section* web page that you used to launch this Jupyter Lab. Scroll down below where you launched Jupyter Lab, and answer the question *Week 1 Assessment*. You can view your overall progress in the assessment by visiting the same *Launch Section* page and clicking on the link to the *Progress* page.

There will be additional questions for you to answer after completing the remaining notebooks. On the *Progress* page, if you have successfully answered all the assessment

questions, you can click on *Generate Certificate* to receive your certificate in the course.

Please Restart the Kernel

In [43]:
```
import IPython
app = IPython.Application.instance()
app.kernel.do_shutdown(True)
```

Out[43]:
```
{'status': 'ok', 'restart': True}
```