Week 2: Identify Nearest Health Facilities

UPDATE
Thank you for your analysis. Despite our warning efforts so far, the virus continues to spread rapidly. We want to get infected individuals treatment as quickly as possible, so we need your help to calculate which hospital or clinic is closest to each known infected individual in the population.

Your goal for this notebook will be to identify the nearest hospital or clinic for each infected person.

Imports

In [1]:
```
import cudf
import cuml
import cupy as cp
```

Load Population Data

Begin by loading the `lat`, `long` and `infected` columns from `'./data/week2.csv'` into a cuDF data frame called `gdf`.

In [2]:
```
# Load the specified columns from the CSV file into a cuDF DataFrame
gdf = cudf.read_csv('./data/week2.csv', usecols=['lat', 'long', 'infected'])
```

```
missing cuda symbols while dynamic loading
cuFile initialization failed
```

Load Hospital and Clinics Data

For this step, your goal is to create an `all_med` cuDF data frame that contains the latitudes and longitudes of all the hospitals (data found at `'./data/hospitals.csv'`) and clinics (data found at `'./data/clinics.csv'`).

In [3]:
```
# Load the latitudes and longitudes from both hospitals and clinics CSV files
hospitals_df = cudf.read_csv('./data/hospitals.csv', usecols=['Latitude', 'Longitud
clinics_df = cudf.read_csv('./data/clinics.csv', usecols=['Latitude', 'Longitude'])

# Concatenate the two DataFrames to create all_med
all_med = cudf.concat([hospitals_df, clinics_df], ignore_index=True)
all_med.head()
```

Out[3]:

	Latitude	Longitude
0	51.379997	-0.406042
1	51.315132	-0.556289
2	51.437195	-2.847193
3	53.459743	-2.245469
4	52.078121	-0.030604

Since we will be using the coordinates of those facilities, keep only those rows that are non-null in both `Latitude` and `Longitude`.

In [4]:
```
# Drop rows where either latitude or longitude is null
all_med = all_med.dropna(subset=['Latitude', 'Longitude'])
```

Make Grid Coordinates for Medical Facilities

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 grid coordinates converter you have used earlier in the workshop. Use this converter to create grid coordinate values stored in `northing` and `easting` columns of the `all_med` data frame you created in the last step.

In [5]:
```
# 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 [6]:
```
all_med['northing'], all_med['easting'] = latlong2osgbgrid_cupy(all_med['Latitude']

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

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

```
      northing        easting
0  165810.473974  510917.517174
1  158381.343420  500604.836652
2  171305.775859  341119.365090
3  395944.561405  383703.600293
4  244071.710013  534945.182860
```

Find Closest Hospital or Clinic for Infected

Fit `cuml.NearestNeighbors` with `all_med`'s `northing` and `easting` values, using the named argument `n_neighbors` set to `1`, and save the model as `knn`.

In [7]:
```
from cuml.neighbors import NearestNeighbors

# Fit the NearestNeighbors model with all_med's latitude and longitude
```

```
knn = NearestNeighbors(n_neighbors=1)
knn.fit(all_med[['northing', 'easting']])
```

Out[7]:
```
NearestNeighbors()
```

Save every infected member in `gdf` into a new dataframe called `infected_gdf`.

In [8]:
```
infected_gdf = gdf[gdf['infected'] > 0]
infected_gdf.head()
```

Out[8]:

	lat	long	infected
1346586	53.715826	-2.430079	1.0
1350932	53.664881	-2.425673	1.0
1352085	53.696765	-2.488940	1.0
1352799	53.696966	-2.488897	1.0
1357529	53.727804	-2.392959	1.0

Create `northing` and `easting` values for `infected_gdf`.

In [9]:
```
infected_gdf['northing'], infected_gdf['easting'] = latlong2osgbgrid_cupy(infected_

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

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

```
         infected       northing        easting
1346586       1.0  424489.783814  371619.678741
1350932       1.0  418820.687944  371876.492369
1352085       1.0  422394.398940  367721.000265
1352799       1.0  422416.821887  367723.973098
1357529       1.0  425808.109929  374076.557677
```

Use `knn.kneighbors` with `n_neighbors=1` on `infected_gdf`'s `northing` and `easting` values. Save the return values in `distances` and `indices`.

In [10]:
```
# Use the k-nearest neighbors model to find the nearest facility for each infected
distances, indices = knn.kneighbors(infected_gdf[['northing', 'easting']], n_neighb
indices[:5]
```

Out[10]:
```
1346586    18316
1350932    12816
1352085     7785
1352799     7785
1357529     4962
dtype: int64
```

Check Your Solution

`indices`, returned from your use of `knn.kneighbors` immediately above, should map person indices to their closest clinic/hospital indices:

In [11]:
```
indices.head()
```

Out[11]:
```
1346586    18316
1350932    12816
1352085     7785
1352799     7785
1357529     4962
dtype: int64
```

Here you can print an infected individual's coordinates from `infected_gdf`:

In [12]:
```
infected_gdf.iloc[0] # get the coords of an infected individual (in this case, indi
```

Out[12]:
```
infected         1.000000
northing    424489.783814
easting     371619.678741
Name: 1346586, dtype: float64
```

You should be able to used the mapped index for the nearest facility to see that indeed the nearest facility is at a nearby coordinate:

In [13]:
```
all_med.iloc[18316] # printing the entry for facility 1234 (replace with the index
```

Out[13]:
```
northing    426435.969251
easting     369952.220236
Name: 18324, dtype: float64
```

Please Restart the Kernel

...before moving to the next notebook.

In [14]:
```
import IPython
app = IPython.Application.instance()
app.kernel.do_shutdown(True)
```

Out[14]:
```
{'status': 'ok', 'restart': True}
```