K-Means

In this notebook you will use GPU-accelerated K-means to find the best locations for a fixed number of humanitarian supply airdrop depots.

Objectives

By the time you complete this notebook you will be able to:

- Use GPU-accelerated K-means
- Use cuXfilter to visualize K-means clusters

Imports

For the first time we import `cuml`, the RAPIDS GPU-accelerated library containing many common machine learning algorithms. We will be visualizing the results of your work in this notebook, so we also import `cuxfilter`.

In [ ]:
```
import cudf
import cuml

import cuxfilter as cxf
```

Load Data

For this notebook we load again the cleaned UK population data--in this case, we are not specifically looking at counties, so we omit that column and just keep the grid coordinate columns.

In [ ]:
```
gdf = cudf.read_csv('./data/pop_2-03.csv', usecols=['easting', 'northing'
print(gdf.dtypes)
gdf.shape
```

In [ ]:
```
gdf.head()
```

K-Means Clustering

The unsupervised K-means clustering algorithm will look for a fixed number *k* of centroids in the data and clusters each point with its closest centroid. K-means can be effective when the number of clusters *k* is known or has a good estimate (such as from a model of the underlying mechanics of a problem).

Assume that in addition to knowing the distribution of the population, which we do, we would like to estimate the best locations to build a fixed number of humanitarian supply depots from which we can perform airdrops and reach the population most efficiently. We can use K-means, setting *k* to the number of supply depots available and fitting on the locations of the population, to identify candidate locations.

GPU-accelerated K-means is just as easy as its CPU-only scikit-learn counterpart. In this series of exercises, you will use it to optimize the locations for 5 supply depots.

Exercise: Make a `KMeans` Instance for 5 Clusters

`cuml.KMeans()` will initialize a K-means instance. Use it now to initialize a K-means instance called `km`, passing the named argument `n_clusters` set equal to our desired number `5`:

In [ ]:

```

```

Solution

In [ ]:

```
%load solutions/make_k-means_instance
```

Exercise: Fit to Population

Use the `km.fit` method to fit `km` to the population's locations by passing it the population data. After fitting, add the cluster labels back to the `gdf` in a new column named `cluster`. Finally, you can use `km.cluster_centers_` to see where the algorithm created the 5 centroids.

In [ ]:

```

```

Solution

In [ ]:

```
%load solutions/km_fit
```

Visualize the Clusters

To help us understand where clusters are located, we make a visualization that separates them, using the same three steps as before.

Associate a Data Source with cuXfilter

In [ ]:

```
cxf_data = cxf.DataFrame.from_dataframe(gdf)
```

Define Charts and Widgets

In this case, we have an existing integer column to use with multi-select: `cluster`. We use the same technique to scale the scatterplot, then add a widget to let us select which cluster to look at.

In [ ]:
```
chart_width = 600
scatter_chart = cxf.charts.datashader.scatter(x='easting', y='northing',
                                               width=chart_width,
                                               height=int((gdf['northing']
                                                          (gdf['easting'].
                                                           chart_width))

cluster_widget = cxf.charts.panel_widgets.multi_select('cluster')
```

Create and Show the Dashboard

In [ ]:
```
dash = cxf_data.dashboard([scatter_chart, cluster_widget], theme=cxf.them
```

In [ ]:
```
scatter_chart.view()
```

In [ ]:
```
%%js
var host = window.location.host;
element.innerText = "'"+host+"'";
```

Set `my_url` in the next cell to the value just printed, making sure to include the quotes and ignoring the button (due to this contained cloud environment) as before:

In [ ]:
```
my_url = # TODO: Set this value to the print out of the cell above, inclu
dash.show(my_url, port=8789)
```

... and you can run the next cell to generate a link to the dashboard:

In [ ]:
```
%%js
var host = window.location.host;
var url = 'http://'+host+'/lab/proxy/8789/';
element.innerHTML = '<a style="color:blue;" target="_blank" href='+url+'>
```

In [ ]:
```
dash.stop()
```

Please Restart the Kernel

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

Next

In [the next notebook](), you will use GPU-accelerated DBSCAN to identify geographically dense clusters of infected people.