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 [1]:
```
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 [2]:
```
gdf = cudf.read_csv('./data/pop_2-03.csv', usecols=['easting', 'northing'])
print(gdf.dtypes)
gdf.shape
```

Out[2]:
```
northing    float64
easting     float64
dtype: object
(58479894, 2)
```

In [3]:
```
gdf.head()
```

Out[3]:

	northing	easting
0	515491.5313	430772.1875
1	503572.4688	434685.8750
2	517903.6563	432565.5313
3	517059.9063	427660.6250
4	509228.6875	425527.7813

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 [4]:
```
km= cuml.KMeans(n_clusters= 5)
```

Solution

In [5]:
```
# %load solutions/make_k-means_instance
km = cuml.KMeans(n_clusters=5)
```

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 [6]:
```
km.fit(gdf)
gdf['cluster'] = km.labels_
km.cluster_centers_
```

Out[6]:

	0	1
0	150988.668019	315543.991697
1	294886.581587	439240.631466
2	402934.342187	395973.013652
3	540894.692061	415736.503333
4	180242.130118	530767.081392

Solution

In [7]:
```
# %load solutions/km_fit
km.fit(gdf)
gdf['cluster'] = km.labels_
km.cluster_centers_
```

Out[7]:

	0	1	2
0	150988.668019	315543.991697	0.0
1	294886.581587	439240.631466	1.0
2	402934.342187	395973.013652	2.0
3	540894.692061	415736.503333	3.0
4	180242.130118	530767.081392	4.0

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 [8]:
```
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 [9]:
```
chart_width = 600
scatter_chart = cxf.charts.datashader.scatter(x='easting', y='northing',
                                              width=chart_width,
```

```
                                                height=int((gdf['northing'].max() - gdf[
                                                           (gdf['easting'].max() - gdf['
                                                            chart_width))

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

Create and Show the Dashboard

In [10]:
```
dash = cxf_data.dashboard(charts=[scatter_chart],sidebar=[cluster_widget], theme=cxf.t
```

In [11]:
```
scatter_chart.view()
```

Out[11]:

In [12]:
```
%%js
var host = window.location.host;
element.innerText = "'http://"+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 [13]:
```
my_url = 'http://dli-604a4aa51b37-32d463.aws.labs.courses.nvidia.com'
dash.show(my_url, port=8789)
```

```
Dashboard running at port 8789
```

Out[13]:

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

In [14]:
```
%%js
var host = window.location.host;
var url = 'http://'+host+'/lab/proxy/8789/';
element.innerHTML = '<a style="color:blue;" target="_blank" href='+url+'>Open Dashboar
```

In [15]:
```
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.