# Grouping and Sorting with cuDF

In this notebook you will be introduced to grouping and sorting with cuDF, with performance comparisons to Pandas, before integrating what you learned in a short data analysis exercise.

## Objectives

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

- Perform GPU-accelerated group and sort operations with cuDF

## Imports

In [1]:
```
import cudf
import pandas as pd
```

## Read Data

We once again read the UK population data, returning to timed comparisons with Pandas.

In [2]:
```
%time gdf = cudf.read_csv('./data/pop_1-04.csv', dtype=['float32', 'str', 'str', 'floa
```

```
CPU times: user 2.08 s, sys: 600 ms, total: 2.68 s
Wall time: 2.68 s
```

In [3]:
```
%time df = pd.read_csv('./data/pop_1-04.csv')
```

```
CPU times: user 26 s, sys: 3.69 s, total: 29.7 s
Wall time: 29.7 s
```

In [4]:
```
gdf.dtypes
```

Out[4]:
```
age        float32
sex         object
county      object
lat        float32
long       float32
name        object
dtype: object
```

In [5]:
```
gdf.shape
```

Out[5]:
```
(58479894, 6)
```

In [6]:
```
gdf.head()
```

Out[6]:

| | age | sex | county | lat | long | name |
|---|---|---|---|---|---|---|
| **0** | 0.0 | m | Darlington | 54.533638 | -1.524400 | Francis |
| **1** | 0.0 | m | Darlington | 54.426254 | -1.465314 | Edward |
| **2** | 0.0 | m | Darlington | 54.555199 | -1.496417 | Teddy |
| **3** | 0.0 | m | Darlington | 54.547905 | -1.572341 | Angus |
| **4** | 0.0 | m | Darlington | 54.477638 | -1.605995 | Charlie |

# Grouping and Sorting

## Record Grouping

Record grouping with cuDF works the same way as in Pandas.

### cuDF

In [7]:

```
%%time
counties = gdf[['county', 'age']].groupby(['county'])
avg_ages = counties.mean()
print(avg_ages[:5])
```

```
                    age
county
Warrington     40.888416
Reading        35.868777
Derbyshire     42.913279
East Sussex    44.757385
Northumberland 44.626919
CPU times: user 64 ms, sys: 12 ms, total: 76 ms
Wall time: 75.2 ms
```

### Pandas

In [8]:

```
%%time
counties_pd = df[['county', 'age']].groupby(['county'])
avg_ages_pd = counties_pd.mean()
print(avg_ages_pd[:5])
```

```
                                age
county
Barking And Dagenham          33.056845
Barnet                        37.629770
Barnsley                      41.201061
Bath And North East Somerset  39.822837
Bedford                       39.715300
CPU times: user 3.9 s, sys: 994 ms, total: 4.89 s
Wall time: 4.87 s
```

# Sorting

Sorting is also very similar to Pandas, though cuDF does not support in-place sorting.

### cuDF

In [9]:
```
%time gdf_names = gdf['name'].sort_values()
print(gdf_names[:5]) # yes, "A" is an infrequent but correct given name in the UK, acc
print(gdf_names[-5:])
```

```
CPU times: user 1.65 s, sys: 7.59 ms, total: 1.66 s
Wall time: 1.66 s
26850     A
154537    A
165578    A
211428    A
236972    A
Name: name, dtype: object
58060377    Zyrah
58289490    Zyrah
58363665    Zyrah
58388727    Zyrah
58394184    Zyrah
Name: name, dtype: object
```

### Pandas

This operation takes a while with Pandas. Feel free to start the next exercise while you wait.

In [10]:
```
%time df_names = df['name'].sort_values()
print(df_names[:5])
print(df_names[-5:])
```

```
CPU times: user 1min 43s, sys: 1.62 s, total: 1min 45s
Wall time: 1min 45s
10811041    A
17931460    A
5060367     A
1842288     A
24866365    A
Name: name, dtype: object
47008072    Zyrah
47953653    Zyrah
31838209    Zyrah
53669567    Zyrah
54557840    Zyrah
Name: name, dtype: object
```

# Exercise: Youngest Names

For this exercise you will need to use both `groupby` and `sort_values` .

We would like to know which names are associated with the lowest average age and how many people have those names. Using the `mean` and `count` methods on the data grouped by name, identify the three names with the lowest mean age and their counts.

In [12]:
```
name_groups = gdf[['name', 'age']].groupby('name')

name_ages = name_groups['age'].mean()
name_counts = name_groups['age'].count()

ages_counts = cudf.DataFrame()
ages_counts['mean_age'] = name_ages
ages_counts['count'] = name_counts

ages_counts = ages_counts.sort_values('mean_age')
ages_counts.iloc[:3]
```

Out[12]:

| | mean_age | count |
|---|---|---|
| **name** | | |
| **Leart** | 34.911197 | 259 |
| **Luke-Junior** | 35.313725 | 255 |
| **Nameer** | 35.479675 | 246 |

## Solution

In [13]:
```
# %load solutions/youngest_names
name_groups = gdf[['name', 'age']].groupby('name')

name_ages = name_groups['age'].mean()
name_counts = name_groups['age'].count()

ages_counts = cudf.DataFrame()
ages_counts['mean_age'] = name_ages
ages_counts['count'] = name_counts

ages_counts = ages_counts.sort_values('mean_age')
ages_counts.iloc[:3]
```

Out[13]:

| | mean_age | count |
|---|---|---|
| **name** | | |
| **Leart** | 34.911197 | 259 |
| **Luke-Junior** | 35.313725 | 255 |
| **Nameer** | 35.479675 | 246 |

# Please Restart the Kernel

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

# Next

As part of our larger data science goal for this workshop, we will be working with data reflecting the entire road network of Great Britain. In the next notebook you will be exposed to additional cuDF techniques that you will use to transform columnar data into graph edge data that we will be using to construct a GPU-accelerated graph using the `cuGraph` library.