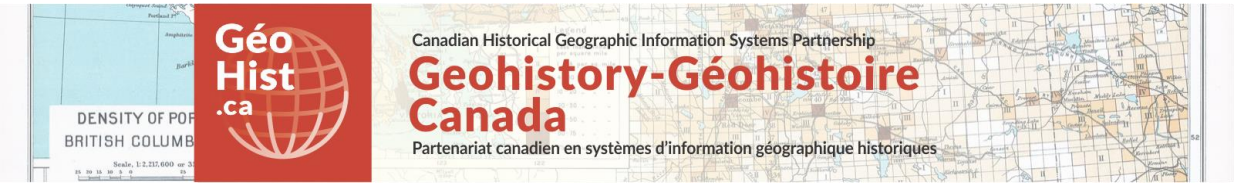


Development document for:
Historical Atlas of Canada Population by Census Divisions 1851-1961 web-map
Using: Mapbox Studio and Mapbox GL JS
Link to webmaps: The **maps are** viewable at: http://mercator.geog.utoronto.ca/Georia/mapbox-hacolp/

The Home page for the site contains links to 3 web maps: Population Growth, Population Density, and Population Distribution.

GOAL: Historical Atlas of Canada (HAC) Time series maps of Population by Census Division (CD), improved from HAC Online Learning project chapter: Summary of Population Growth, 1851-1961.
http://www.historicalatlas.ca/website/hacolp/national_perspectives/population/UNIT_25/index.htm

REQUIREMENTS: The original website has three interactive maps of population by Census Division: Population Density (choropleth), Population Growth (graduated circles) and Population Distribution (dot density) for 11 census years, 1851-1961, using old ArcIMS technology. These maps should be reproduced but updated to current webmap standards, improving performance and visualization, and using a Time-slider to click through the time period, replacing the Checkbox interface originally used. The project is also an appropriate one to use to explore the web-mapping software's capacity for legend design flexibility, and map projections other than the standard Web Mercator.

Credits: Historical Atlas of Canada Online Learning Project, Canadian Historical GIS Partnership Development project, GIS and Cartography Office Dept of Geography University of Toronto
Programming assistance: Jonathan Critchley http://jonathancritchley.ca/

Note: The Mapbox online documentation is extensive and comprehensive, and there are many users' examples online as well. Therefore many details will only be referenced to that documentation rather than explained in full here.

Approach: Decision made to use latest tools from Mapbox https://www.mapbox.com/ as of date of production (mid-2017).

Functionality:
Mapping functionality: Mapbox Studio is the online data-upload and basic webmap design interface.
Mapping and Interface functionality: Mapbox GL JS is "a JavaScript library that uses WebGL to render interactive maps from vector tiles and Mapbox styles. It is part of the Mapbox GL ecosystem..."
Projection functionality: Projections other than Web Mercator not currently supported for display in Mapbox. https://www.mapbox.com/help/how-mapbox-data-works/#mapbox-map-projections

Map Data: All data uploaded from shapefiles to Mapbox Vector tilesets
Overlay Mapping data: 1851-1961 Census Divisions polygons merged - Shapefile uploaded to tileset
1851-1961 Census Divisions boundaries Merged - Shapefile uploaded to tileset
1851-1961 Provinces/Territories Polygons Merged - Shapefile uploaded to tileset
Background Mapping data: Modern Cities, Modern Provincial Boundaries - Shapefile uploaded to tileset
Base map: Mapbox "Light" base map, customized in Mapbox Studio

Note re: Historical Atlas of Canada Data
These data are available publicly through the Historical Atlas of Canada Online Learning project, by email request. http://www.historicalatlas.ca
Census Divisions are also available through the Scholars Geoportal http://geo1.scholarsportal.info/

PRODUCTION NOTES

OVERLAY MAPPING DATA

QGIS preparation of map layers
Individual shapefiles for each year 1851 through 1961 were provided by the HACOLP and used as the basis for the project. QGIS was used to create the merged data layers necessary to create a coded layer with a YEAR field suitable for use in Mapbox GL for time-based display. The “Merge shapefiles to one” function under the menu Vector|Data management tool was used. Each merged layer was then re-projected from its original projection to Web Mercator (EPSG:3857). The merged shape files created were:

Census divisions 1851 to 1961 - polygon areas of CDs changing over time.
Census divisions boundaries 1851 to 1961 - linear boundaries of CDs changing over time.
Census divisions dot density 1851 to 1961 - dot density representations of CD populations over time (see explanation below)
Ecumene 1851 to 1961 - polygons representing settled area of CDs changing over time
Province boundaries 1851 to 1961 - linear boundaries of provinces and territories changing over time.
Province centroids 1851 to 1961 - used to label province names as they changed area over time.
Modern Cities - this layer is used to label modern cities as of 2017.
Modern Political Boundaries - this layer represents the boundaries of modern Provinces as of 2017.

Unlike ArcGIS, Mapbox does not support or require a Date formatted field or "time-awareness" in a layer, but uses straight coding and filtering of YEAR-specific layers to allow time-slider functionality.

Background Mapping data:

Modern Geography Reference layers: Modern Cities, Modern Provincial Boundaries - HACOLP shape files uploaded to Tilesets. Toggle Buttons created to turn layers on or off (see code reference below).
Colour coding: Yellow symbolization with label halos seen as most unobtrusive while providing modern geographic reference information. This follows the design set in the original HACOLP web maps.

Uploading of layers into Mapbox: Shapefiles, which comprise multiple component files, are compressed to ".zip" files and uploaded to Mapbox as "Tilesets", vector tiled feature layers.

BASE MAPS
Reference Base Map: Mapbox "Light" base map, was selected as the most useful option to provide a neutral base map as a geographic reference background for colour-based thematic mapping such as choropleth and graduated symbol representation. This was then customized in Mapbox Studio.

OVERVIEW OF WORKFLOW IN MAPBOX

Mapbox is a cloud-based open-source mapping platform for custom designed mapping. Mapbox is built on vector tiles for rendering maps, and they developed this format, "an advanced approach to mapping where data is delivered to the device and precisely rendered in real-time." (www.mapbox.com/maps) Esri and other industry leaders are now using vector tiles for their base mapping. Mapbox formerly allowed users to install a stand-alone version but now overwhelmingly works in the cloud. A Mapbox "account" is required to obtain a free "access token" to create original maps for inclusion in web applications. Pricing is free to start, up to 50,000 map views per month, then costs on a sliding scale.

Mapbox consists of a number of related products. Maps in Mapbox are built using Mapbox.js, the Mapbox Javascript API (https://www.mapbox.com/mapbox.js/api). **Mapbox Studio** is a data and map management platform and GUI interface for drag and drop, menu-based map composition, which can then be customized further by coding. Any customization not supported within Studio can be programmed within the impressive array of Developer Tools (SDKs and APIs) which are summarized online here: https://www.mapbox.com/developers/

For this project we used Mapbox Studio for our base map and for testing graphic design elements for mapping. Mapbox supplies its base map layers as Vector tile "styles" in Mapbox terminology, which can be cloned and customized by layer/feature selection and re-symbolization in Mapbox Studio. For information about their mapping environment, see: https://www.mapbox.com/maps/
We made a copy of their "Light" base map, designed to backstop colourful thematic overlay displays, and then used Mapbox Studio to turn off all of the data layers irrelevant in our historical context.

Thematic layers such as the choropleth, graduated symbols and dot density layers required for this project can be developed in Mapbox Studio, but it quickly became clear that the functionality we required would only be possible using one or more of their developer APIs. This is for two reasons:
1. Studio is designed for graphic representation of individual layers, rather than "data driven" representation of large or diverse datasets needing flexible symbolization.
2. Functionality such as the Time-Slider needs an interaction API like **Mapbox GL JS**, "...a JavaScript library that uses WebGL to render interactive maps from vector tiles and Mapbox styles..." See https://www.mapbox.com/mapbox-gl-js/api/

Mapbox makes it very easy to get started using their system, by providing lots of Documentation and Tutorial examples. Any users who might be intimidated by the prospect of "coding" are led through easy start-up projects step by step, and end up using Studio, and even Mapbox GL, before they know it. For example, the following examples create a choropleth map of the US by state in part 1, and then adds interaction to the display (mouseovers to display data) in part 2.
Make a choropleth map with Mapbox part 1: create a style with Mapbox Studio
https://www.mapbox.com/help/choropleth-studio-gl-pt-1/
Make a choropleth map with Mapbox part 2: publish your style with Mapbox GL JS
https://www.mapbox.com/help/choropleth-studio-gl-pt-2/
This type of example is good in establishing the workflow between Mapbox Studio and GL. It is kind of like ArcGISonline Web Map and Web App, except that the App part is all done by coding in GL rather than GUI (for now!)

However, this example is not a great model of how to actually do things - like how to make a choropleth map. Studio is quite basic, so in this example you make a separate layer for each of your classed interval

layers, and group them. It would be much preferable to have one layer and use the attribute value for Population Density to colour the map. There appears to be no way to do this in Mapbox Studio directly. To do so this it is necessary to use Mapbox GL, and a "Data Driven layer". There is an example of a choropleth using this, as well as changing level of geography by scale, here:
Update a choropleth map by zoom level
https://www.mapbox.com/mapbox-gl-js/example/updating-choropleth/
The critical part is the code below (truncated) which sets intervals and colours:

```
map.addLayer({
    'id': 'state-population',
        ...
    'paint': {
      'fill-color': {
        property: 'population',
        stops: [
          [0, '#F2F12D'],
          [500000, '#EED322'],
          [750000, '#E6B71E'],
        ...
```

The example that they use as the overtly data driven style is changes dot colour depending on ethnicity:
Style circles using data-driven styling
https://www.mapbox.com/mapbox-gl-js/example/data-driven-circle-colors/

So our Mapbox workflow involved finding examples, in the documentation or online by users or on github.com, and adapting them to the project requirements.

A Timeline slider example also exists for Mapbox GL**: Create a timeline animation**
https://www.mapbox.com/mapbox-gl-js/example/timeline-animation/
Another Mapbox GL example we used was **Show-change-over-time**
https://www.mapbox.com/help/show-changes-over-time/
This uses both a time slider as well as radio buttons to filter data.
These are data driven graduated circles rather than a choropleth map - but the principle is the same as ours, and should be adaptable. However, the range slider in this Mapbox example is a basic HTML5 "input" object of type "range", which is not very sophisticated or adaptable to graphic customization.

We can use another example webmap (not from Mapbox, but a great visualization) to guide us to a better method for including a slider, which we should be able to adapt. The webmap is the **Morphocode Urban Layers** map (historical buildings in Manhattan, NYC.)
http://io.morphocode.com/urban-layers/
Morphocode is the company that made it; they used Mapbox GL, and some custom coding using JQuery and D3. Fortunately, they also included some info about how the map and controls were made:
http://morphocode.com/making-urban-layers/
From this we see that "The Slider is based on jQuery Draggable, allow you to isolate only buildings built during the selected time frame." This is a two handled range slider, but the JQuery Draggable is a set of sliders with options, which include one with just "snap to increments" or "steps," which is what we need. http://jqueryui.com/slider/#steps

DOCUMENTATION OF CODING FOR HACOLP

The code for Historical Atlas of Canada Population by Census Divisions 1851-1961 Mapbox GL web-map front page and maps is available on github at:
https://github.com/geohist-ca/hacolp-cds
Examples of specific functionality are also available there in the documents:
HACOLP_template_LayerFilter.html and HACOLP_template_Legend.html

What follows is some detailed documentation of code development written by Jonathan Critchley who provided programming assistance for the project, including snippets of code, links to examples, web sites or posts that inspired the adaptation of code. These require some understanding of javascript and are best understood if read in conjunction with the mapping HTML pages. The main libraries used are:

- **MAPBOX GL** https://www.mapbox.com/mapbox-gl-js/api/
- **JQUERY** https://jquery.com/
- **BOOTSTRAP** http://getbootstrap.com/

FEATURES AND SYMBOLIZATION

POPULATION DENSITY – CHOROPLETH MAP

Using the polygon merged Census division layer from 1851 to 1961.
To create a choropleth map using data driven stying in Mapbox GL used the following layer parameters, setting the filter to display the first year 1851 on page load:

```
'type': 'fill',
    'filter': ['==', 'YEAR_', 1851],
    'paint': {
      'fill-color': {
```

Set fill color based on attribute for population density 'CDPOPDENS' using the property item - https://www.mapbox.com/help/gl-dds-ref/#property-functions.

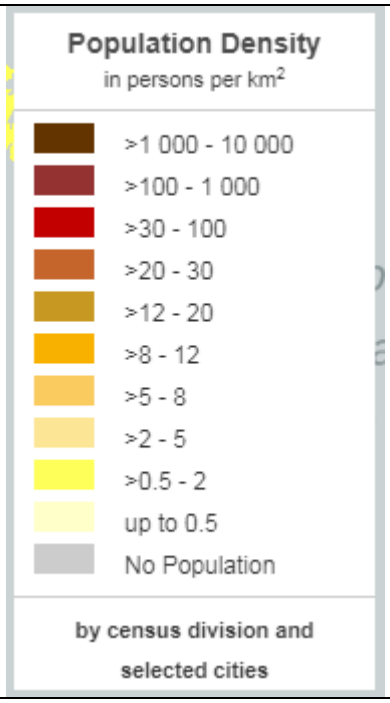
```
        property: 'CDPOPDENS',
```

Type item - https://www.mapbox.com/help/gl-dds-ref/#property-functions. Type is set to interval: Return the output value of the stop just less than the function input:

```
        type: 'interval',
```

Stops represent the lower range of a category break, specify color associated with the stop range. RGB values for colours were extracted from original HACOLP choropleth map. See image below.

```
stops: [
    [0.0, 'rgb(204,204,204)'],
    [0.0001, 'rgb(255,255,204)'],
     [0.5, 'rgb(255,255,102)'],
    [2.0, 'rgb(255,229,153)'],
    [4.0, 'rgb(255,204,102)'],
    [8.0, 'rgb(255,178,0)'],
    [12.0, 'rgb(204,153,48)'],
    [20.0, 'rgb(204,102,51)'],
    [30.0, 'rgb(204,0,0)'],
    [100.0, 'rgb(153,51,51)'],
    [1000.0, 'rgb(102,51,0)'],
  ]
 },
```



ISSUES with Outline symbols in Mapbox GL

A limitation of the Mapbox GL 'fill' layer type is that the fill-outline-width is set to 1 and cannot be changed. To specify a line weight it is necessary to generate a line boundary and import as "line" layer type.

TIME SLIDER FOR YEAR USED ON ALL MAPS

Created a jQuery slider tied to Mapbox GL layer to filter based on Census Division Layer, Census Division Boundaries layer, Province Boundaries layer, and Province centroid/labels layer filtered on the YEAR_ attribute. http://api.jqueryui.com/slider/. See image below.

The slider parameters are set as follows:

Initial placement of the handle on the slider matching the filter specified in the layer properties:

value:1851,

Set the minimum slider value:

min: 1851,

Set the maximum slider value:

max: 1961,

Provide the number of steps/intervals required between the max and min values:

step: 10,

The slider responds by using two jQuery slider events:

Change - Change event captures when the slider handle HAS MOVED location. Change event only updates/captured after the slider handle is moved.

Slide - Slide event captures when the slider handle IS BEING MOVED location. Slide event updates/captured while the slider handle is being moved.

The jQuery slider is customized using CSS to update appearance, using the styles:

.ui-slider
.ui-slider-handle

We created a counter that updates based on the position of the slider/value selected to help the user identify exactly what is selected.

#yearCount

Labels are created below the jQuery slider to also help users identify what year/value they have selected.

label_scale

We created a variable containing an array of all year labels.

var items =[]

To change the text in the labels they must be updated in the array. The number of values in the array must match the number of intervals including start and end.

var items =['1851','1861','1871','1881','1891','1901','1911','1921','1931','1941','1951','1961'];

In this example there are 12 labels.

The start label must match the start slider parameter.
The last/end label must match the end slider parameter.

POPULATION GROWTH – GRADUATED SYMBOLS MAP

Generated a point file of polygon centroids from the merged Census Division layer to be used for locations of graduate symbols. To create graduated symbols using data driven styling in Mapbox GL used the following layer parameters:

```
'type': 'circle',
    'filter': ['==', 'YEAR_', 1851],
    'paint': {
    'circle-radius': {
```

Set circle size based on attribute for population ‘CDPOP’ using the property item - https://www.mapbox.com/help/gl-dds-ref/#property-functions.

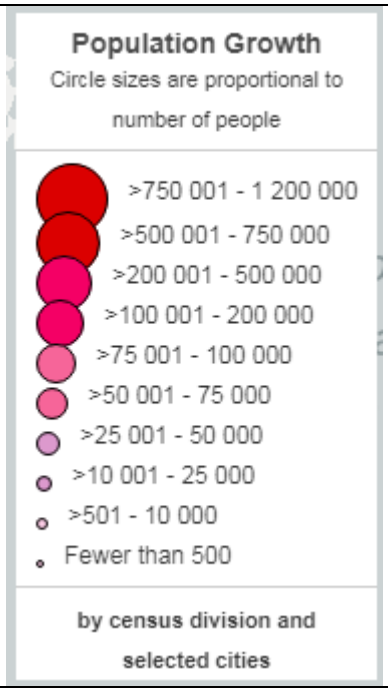
```
        property: 'CDPOP',
```

Type item - https://www.mapbox.com/help/gl-dds-ref/#property-functions. Type is set to interval: Return the output value of the stop just less than the function input.

```
type: 'interval',
```

Stops represent the lower range of a category break; specify circle size associated with the stop range. Sizes for circles were extracted from the original HACOLP Graduated circles map. See image below.

```
stops: [
        [0, 0],
        [0.1, 2],
        [500.0, 3],
        [10000.0, 4],
        [25000.0, 6],
        [50000.0, 8],
        [75000.0, 10],
        [100000.0, 12],
        [200000.0, 14],
        [500000.0, 16],
        [750000.0, 18],
       ]
 },
```



Circle colour was set based on attribute for population 'CDPOP' using the property item - https://www.mapbox.com/help/gl-dds-ref/#property-functions.

```
'circle-color': {
        property: 'CDPOP',
```

Type item - https://www.mapbox.com/help/gl-dds-ref/#property-functions. Type is set to interval: Return the output value of the stop just less than the function input

```
type: 'interval',
```

Stops represent the lower range of a category break, specify color associated with the stop range. RGB values for colours were extracted from original HACOLP graduated circles map.

```
stops: [
        [0.1, 'rgb(255,204,229)'],
        [500.0, 'rgb(255,204,229)'],
        [10000.0, 'rgb(225,153,204)'],
        [25000.0, 'rgb(225,153,204)'],
```

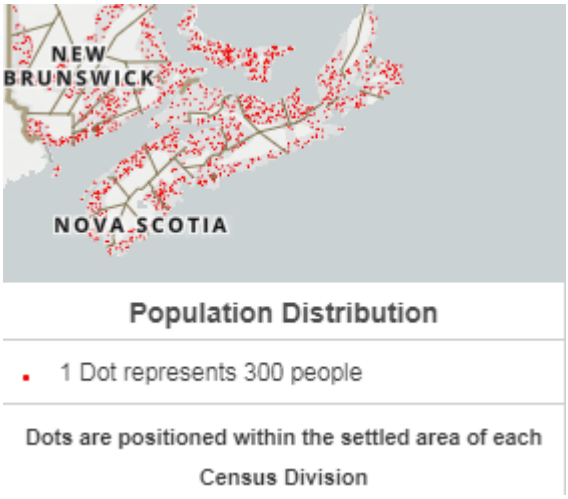
```
        [50000.0, 'rgb(255,102,153)'],
        [75000.0, 'rgb(255,102,153)'],
        [100000.0, 'rgb(255,0,102)'],
        [200000.0, 'rgb(255,0,102)'],
        [500000.0, 'rgb(229,0,0)'],
        [750000.0, 'rgb(229,0,0)'],
    ]
},
```

POPULATION DISTRIBUTION - DOT DENSITY MAP

Mapbox does not support a dot density mapping representation in Studio, or directly in any of the Mapbox GL examples we found. Therefor it was decided to use QGIS to create dot features using the QGIS Random Points tool, which creates points in a polygon area based on value from an input field.

This process was run on each CD polygon layer individually by year before merging them, as the output shapefile only contains feature IDs, no other attributes. This meant that a YEAR_ attribute needed to be added for each new point layer generated using Random Points tool before merging all year layers together.

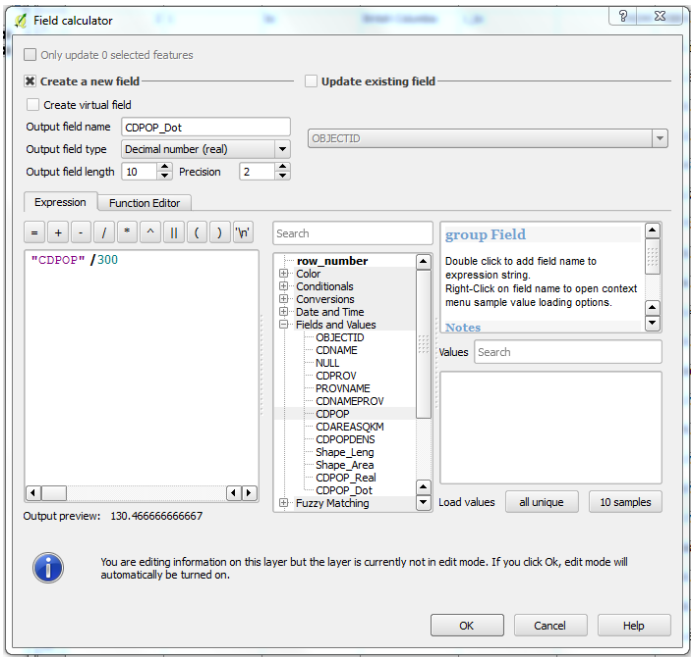
Creating one dot from the total Population in Each CD would result in too large a file (CDPOP = 1000 will result in 1000 points). **To match the original maps, 1 point = 300 people.** See resultant legend image below:



In order to accomplish this, a new attribute was created CDPOP_Dot using the QGIS Field Calculator.

Create a new field:

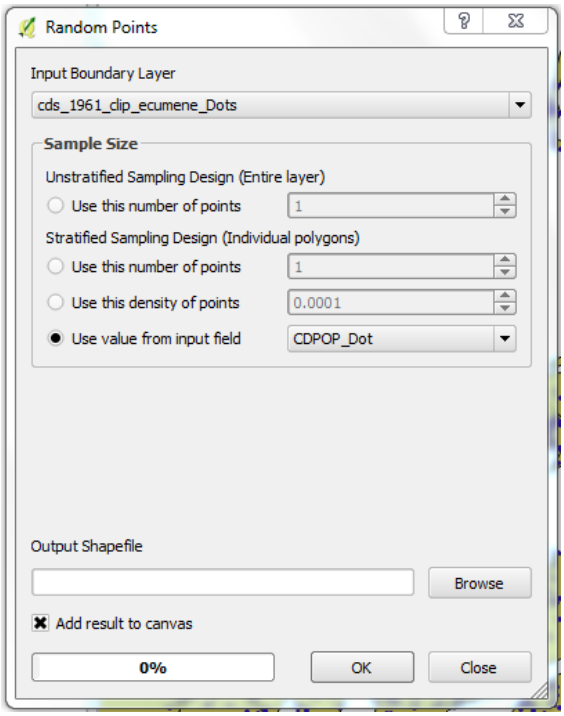
- Output field name: CDPOP_Dot
- Output field type: Decimal Number (real)
- Output field length: 10
- Precision: 2
- Expression: “CDPOP”/300



The Random Points tool requires the attribute field to be a Real number (decimal) for it to display the attribute to be used.

Vector > Research Tools > Random Points

- Input Boundary: cds_1961_clip_ecumene
- Use value from input field: CDPOP_Dot



The Output Shapefile only contains ID attributes. Create a new YEAR_ attribute, in each shapefile before merging.

- **Output field name: YEAR_**
- Output field type: Whole Number (integer)
- Output field length: 10
- Expression: '1961'

THE MAPBOX CODE FOR DISPLAYING POPULATION MAP LAYERS

The Mapbox code for displaying the population distribution map is straightforward, and similar to those from the previous maps. Below is an example extracted from the **distributionmap.html**

```
// Defines dot distribution map layer, sets a Layer id and options
// IMPORTANT - This layer Zoom extent is z3 ~ z9 due to its complexity - mapbox limits zooms when
//data is too complex
   map.addLayer({
      'id': 'cd-pop-dots',
      'source': {
                  'type': 'vector',
                  'url': 'mapbox://byronmoldofsky.1hf7xwc6'
                  },
      'source-layer': 'cds_1851_1961_clip_ecumene_Do-3awy39',
      'type': 'circle',
//sets filter to only display 1851 data in this layer on page load
      'filter': ['==', 'YEAR_', 1851],
      'paint': {
       'circle-radius': 1,
 // Set circle color based on the value specified in the breaks array
 // there is only one array so specify the specific values from the array
      'circle-color': breaks[0][1],
       // Set circle opacity and stroke
      'circle-opacity': 1,
      }
     },
```

DATA DRIVEN STYLING AND AUTOMATED LEGEND CREATION

One of the requirements of this pilot project was to explore the web-mapping software's capacity for legend design flexibility. We therefore designed a template to use for generating legends semi-automatedly from the data, based on the data-driven style parameters the user sets for styling. Below is the example of its **implementation for the Population Density Choropleth map** (see **densitymap.html**)

We created global variables to be used to apply style to layer and legend, rather than hard coding the "stops" as indicated in the legend code pictured above. This was inspired by code provided in the following blog post: http://thinkingspatial.com/mapbox-gl-js-3d-buildings/

One array is used to style the layer census division and provide color to legend keys
Stop domain values must appear in ascending order to be used in the Mapbox GL layer:

```
var breaks = [
                    [0.0, 'rgb(204,204,204)'],
                    [0.0001, 'rgb(255,255,204)'],
                    [0.5, 'rgb(255,255,102)'],
                    [2.0, 'rgb(255,229,153)'],
                    [4.0, 'rgb(255,204,102)'],
                    [8.0, 'rgb(255,178,0)'],
                    [12.0, 'rgb(204,153,48)'],
                    [20.0, 'rgb(204,102,51)'],
                    [30.0, 'rgb(204,0,0)'],
                    [100.0, 'rgb(153,51,51)'],
                    [1000.0, 'rgb(102,51,0)'],
];
```

Another array contains the text used to populate the legend labels

```
var legendLabels = [
                    ['No Population'],
                    ['up to 0.5'],
                    ['up to 2'],
                    ['up to 5'],
                    ['up to 8'],
                    ['up to 12'],
                    ['up to 20'],
                    ['up to 30'],
                    ['up to 100'],
                    ['up to 1000'],
                    ['up to 10000'],
];
```

When setting the options for the 'cd-popdensity' layer, the breaks variable containing the arrays is declared for the paint > fill-color stops parameter.

```
    'paint': {
```

```
        'fill-color': {
Set polygon fill color based on attribute for population density CDPOPDENS
          property: 'CDPOPDENS',
                type : 'interval',

Loading array declared in the breaks variable
          stops: breaks,
                },
         'fill-opacity': 1
         }
```

The **slice() method** returns the selected elements in an array, as a new array object. The original array will not be changed. https://www.w3schools.com/jsref/jsref_slice_array.asp

The reverse() method reverses the order of the elements in an array.
https://www.w3schools.com/jsref/jsref_reverse.asp

This is used to reverse the ordering of the legend colors and label from how they are provided

The legends Labels arrays could have been provided in reverse order and not required slice().reverse(). but for this example were provided reversed for consistency with the layers arrays order.

```
breaks Rev = breaks.slice().reverse();
legendLabelsRev = legendLabels.slice().reverse();
```

The legend div is placed between text. div class='legend' id='cd-legend' is empty but is filled with the legend key and text using javascript.

```
<!-- Bottom right Legend -->
<div class='legend-container'>
<div class='legend' id='legend' >
  <h3>Population Density</h3> <p>in persons per km<sup>2</sup></p>
  <hr/>
<div class='legend' id='cd-legend' >
</div>
        <hr/>
        <!-- <b> tag bolding text -->
  <p><b>by census division and selected cities</b></p>
</div>
</div>
```

The Legend with id = 'cd-legend' is declared a variable to be used in in the script. Load the legend from the HTML document

```
        var legend = document.getElementById('cd-legend');
```

Using Javascript for each method. The forEach() method calls a provided function once for each element in the breakRev variable array, in order. https://www.w3schools.com/jsref/jsref_forEach.asp

Creating elements to fill the legend div with id = 'cd-legend'. Each stop gets a 'row'
```
var item = document.createElement('div');
```

Add a 'key' to the row. A key will be the color key
```
var key = document.createElement('span');
```

Add a value variable to the 'row' in the legend
```
var value = document.createElement('span');
```

The key will take on the shape and style properties defined here, in the HTML
```
key.className = 'legend-key';
```

The background color is retrieved from the breaks array
```
key.style.backgroundColor = layer[1];
```

Give the value variable a placeholder id that we can access and update with custom labels
```
value.id = 'legend-value-' + i;
```

Add the key (color key) to the legend row
```
item.appendChild(key);
```

Add the value to the legend row
```
item.appendChild(value);
```

Add row to the legend
```
legend.appendChild(item);
```

Using Javascript for each method. The forEach() method calls a provided function once for each element in the legendLabels variable array, in order. https://www.w3schools.com/jsref/jsref_forEach.asp

```
legendLabelsRev.forEach(function(layer, i){
```

As we iterate through the arrays get the correct row, and add the appropriate text for the legend labels

```
document.getElementById('legend-value-' + i).textContent = layer[0];
});
```

Data Driven legend amendments for Population Growth Graduated Symbol Map

For the Population Growth map a third array is included because we are not only using different colors based on breaks but also setting circle radius. (See **growthmap.html**)

Arrays used to style the layer circle size
```
var radius = [
```

Stop domain values must appear in ascending order
```
Population threshold [0] Size [1]
```

```
            [0.1, 2],
            [500.0, 3],
            [10000.0, 4],
            [25000.0, 6],
            [50000.0, 8],
            [75000.0, 10],
            [100000.0, 12],
            [200000.0, 14],
            [500000.0, 16],
            [750000.0, 18],
];
```

After creating the legend items and declaring the key color we also use the radius array to calculate a diameter used to specify a height and width for they legend key.

Declare legend label circle size, Mapbox values are declared as circle radius, we must declare height and width css property as diameter (2 * radius) so that the key is proportional.

We also use the getElementById() javascript method to set additional styling to create the circles:
https://www.w3schools.com/jsref/met_document_getelementbyid.asp

```
radiusRev.forEach(function(layer, i){
```

We set the height based on the radius value from the array, multiplying by 2 (layer[1]*2)
https://www.w3schools.com/JSREF/prop_style_height.asp

```
document.getElementById('legend-key-' + i).style.height = layer[1]*2 + 'px';
```

We set the width based on the radius value from the array, multiplying by 2 (layer[1]*2)
https://www.w3schools.com/JSREF/prop_style_height.asp

```
document.getElementById('legend-key-' + i).style.width = layer[1]*2 + 'px';
```

Other settings are as follows, also referenced on https://www.w3schools.com/jsref

```
document.getElementById('legend-key-' + i).style.display = 'inline-table';
document.getElementById('legend-key-' + i).style.borderRadius = '50px';
document.getElementById('legend-key-' + i).style.border = '0.5px solid #000000';
document.getElementById('legend-key-' + i).style.marginRight  = '10px';
document.getElementById('legend-key-' + i).style.marginLeft  = '10px';
```

Reusing the radius value from the array, using it as a negative bottom margin so that the cirlcles in the legend key overlap

```
document.getElementById('legend-key-' + i).style.marginBottom = "-" + layer[1] + "px";
});
```

MAPBOX GL and JQUERY UI COMMON TO ALL MAPS

Map Controls and options

Layer maxZoom is set to 9, preventing the map view from zooming beyond that point.

Disable map rotation using right click + drag

```
map.dragRotate.disable();
```

Disable map rotation using touch rotation gesture for mobile

```
map.touchZoomRotate.disableRotation();
```

Added Zoom Controls

```
map.addControl(new mapboxgl.NavigationControl());
```

Popup attribute display

Click to open popup, to display attributes/properties from the layer uploaded to Mapbox:

Click on a feature is captured in a click event only on the layers specified.

```
map.on('click', function(e) {
var features = map.queryRenderedFeatures(e.point, { layers: ['cd-popdensity'] });
```

When clicking on the layer a popup is created where the user clicked on the map. The contents of the popup are set by specifying the order of layer attributes/properties:

feature.properties['PROVNAME'] = the province name attribute/property tied to the feature that is clicked.

There is code in the jQuery slider event Change that ensures any open popup is removed when the slider is used as the layer is updated and the popup is no longer displaying information related to the displayed layer features.

```
popup.remove();
$(".mapboxgl-popup").toggle();
```

It is specified that the mouse cursor changes to a pointer when it hovers over a layer that a user can interact with, in this case open a popup for a layer.

```
map.on('mousemove', function(e) {
    var features = map.queryRenderedFeatures(e.point, { layers: ['cd-popdensity'] });
    map.getCanvas().style.cursor = (features.length) ? 'pointer' : '';
});
```

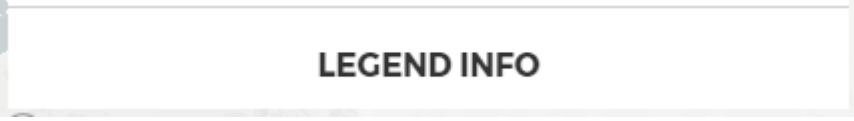
Modern Geography Reference Layers toggle

Created buttons to turn on and off modern layers. Based on Mapbox example:
https://www.mapbox.com/mapbox-gl-js/example/toggle-layers/

The values in the array must match the layerID of the layer you want to toggle that is set when declaring the layers above. Important to consider that the layerID text in the below array is how it will appear in the label, so set layerID as you want it displayed when declaring the layer.

Legend Information Container controlled by open and close button

Using jQuery to create buttons and control behaviour when buttons are clicked
Button open info div using jQuery toggle() method
http://api.jquery.com/toggle/



With no parameters, the .toggle() method simply toggles the visibility of elements

```
$(document).ready(function(){
```

Declaring the div to listen for click event

```
$("#buttonOpen").click(function(){
```

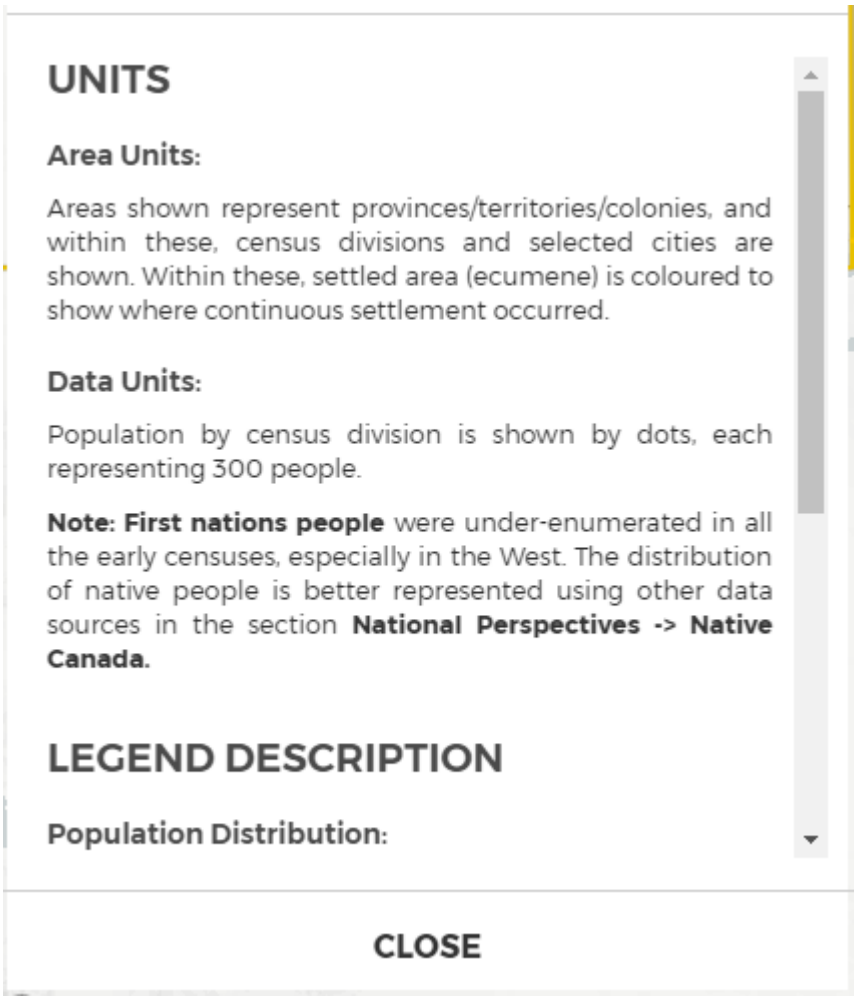
Declaring action when that div is clicked, hide #buttonOpen, display #buttonClode and #infoContainer

```
$("#buttonOpen").toggle();
$("#buttonClose").toggle();
$("#infoContainer").toggle();
});
```

Button close info div using jQuery. Reverse of what is declared above
when clicking #button close hide #buttonClose and #infoContainer, display #buttonOpen

```
$("#buttonClose").click(function(){
$("#buttonOpen").toggle();
$("#buttonClose").toggle();
 $("#infoContainer").toggle();
```

See image below for view of Legend Information Container open.


UNITS
Area Units:
Areas shown represent provinces/territories/colonies, and within these, census divisions and selected cities are shown. Within these, settled area (ecumene) is coloured to show where continuous settlement occurred.
Data Units:
Population by census division is shown by dots, each representing 300 people.
Note: First nations people were under-enumerated in all the early censuses, especially in the West. The distribution of native people is better represented using other data sources in the section National Perspectives -> Native Canada.
LEGEND DESCRIPTION
Population Distribution:
CLOSE