Plotting Canadian Geographic Data in R with the mapcan Package

Andrew McCormack

2019-01-04

Contents

1 Overview of workshop

In this workshop, you will learn how to create informative plots in R using Canadian geographic data. Maps have become a popular method for visualizing data. They are visually pleasing and easy to understand. Yet, standard mapping techniques have a tendency to obscure rather than inform our understanding of how geographic data are distributed. This workshop will introduce you to two, increasingly popular alternative to standard maps: tile/hexagonal grid maps and population cartograms.

The bulk of the workshop will focus on `mapcan`, an R package that provides tools for plotting Canadian choropleth (standard) maps as well as tile/hexagonal grid maps and population cartograms.

While no prior knowledge of geographic data visualization is required, some experience with R and the `ggplot2` package will help.

1.1 Why `mapcan`?

Visualizing spatial data in R often involves working with large, specialized shape files that require a fair amount of conversion and manipulation before they are ready for use in `ggplot`. This can be a tedious process. `mapcan` has done most of this work already, providing flexible, geographic data in data frame format, making it easy to use with `ggplot`.[1]

[1] If you are interested in a more involved approach to spatial data visualization—for instance, if you have your own shapefile and you want to use it in R—the following resources are useful: Datacamp's "Working with Geospatial Data in R", Jesse Sadler's Introduction to GIS with R, and this guide from `ggplot2` on how to convert shapefiles so they work with `ggplot`.

2 Getting started

This workshop relies on functions from the R packages `mapcan`, `ggplot2`, and `dplyr`.

2.1 Installing and loading mapcan

`mapcan` is currently hosted on GitHub, and you can install it from there using the `install_github()` function from the `devtools` package.[2].

[2] `mapcan` will be up on CRAN soon (which will enable the use of `install.packages()`). For now, the `mapcan` package is hosted on GitHub, so we use `install_github()`

```
# If you don't have the devtools installed, first install:
# install.packages("devtools")
library(devtools)
# Install mapcan
install_github("mccormackandrew/mapcan", build_vignettes = TRUE, force = TRUE)

# Load mapcan
library(mapcan)
```

I suggest you pass the `build_vignettes = TRUE` argument to `install_github()`. The vignettes provide detailed guides on how `mapcan`'s functions operate.

2.2 *Installing and loading other required packages*

`ggplot2` and `dplyr` can be installed and loaded either individually or through the `tidyverse` package.

```
# If you don't have the tidyverse installed, first install:
# install.packages("tidyverse")
library(tidyverse)
```

3 *`ggplot` refresher*

3.1 *The `ggplot2` package*

`ggplot2` is a powerful package for data visualization that allows you to create many types of plots with a great deal of flexibility. It is especially useful for making maps in R. `ggplot2` is based on the *Grammar of Graphics*—quantitative plots are composed of elements that convey precise and clear messages much like the grammatical elements of sentences. To create quantitative plots, we work with a number layered elements. The strength of `ggplot2` is that each of these elements can be added iteratively (i.e. we can add one element at a time to create highly customized plots). Let's start from scratch with the first and most important element: data.

3.2 *Geographic data*

To plot maps in ggplot, we need 3 basic elements: geographic coordinates, a grouping variable, and some variable we wish to represent geographically. To illustrate, we will create some fictional data:

```
df <- data.frame(x = c(2, 3, 1, 4, 5, 6),
           y = c(1, 2, 3, 5, 8, 9),
           group = factor(c("t1", "t1", "t1",
                         "t2", "t2", "t2")),
           variable = c("blue", "blue", "blue",
                       "red", "red", "red"))
df
```

```
##   x y group variable
## 1 2 1    t1     blue
## 2 3 2    t1     blue
## 3 1 3    t1     blue
```

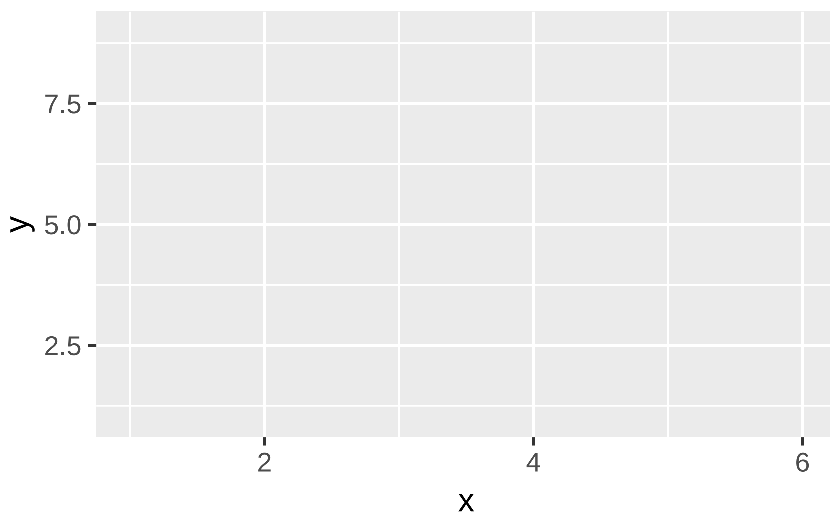
```
## 4 4 5    t2      red
## 5 5 8    t2      red
## 6 6 9    t2      red
```

The `x` and `y` variables are our geographic coordinates, which in this case are just two triangular "islands". Our grouping variable, `group`, will be used to tell `ggplot` that these triangles are two discrete objects. Our variable, `variable`, is simply the variable we want to visualize on our map.

3.3 *Aesthetic mapping*

Aesthetics refer to the variables we want to present. Aesthetic mappings in `ggplot2`, which go inside the `aes()` argument, define the variables that will be represented on our vertical (y) and horizontal (x) axes as well as how the data will be grouped. Let's initialize a `ggplot` object with the data we just created with an aesthetic mapping:

```
ggplot(df, aes(x, y))
```

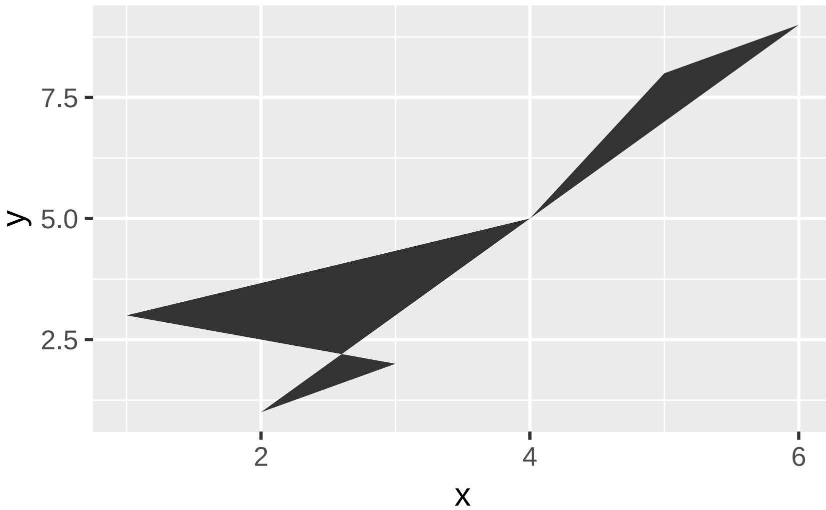


There's not much going on here. We need to tell `ggplot` the type of visual elements we want to plot—which in this case are geographic coordinates.

3.4 *Geometries*

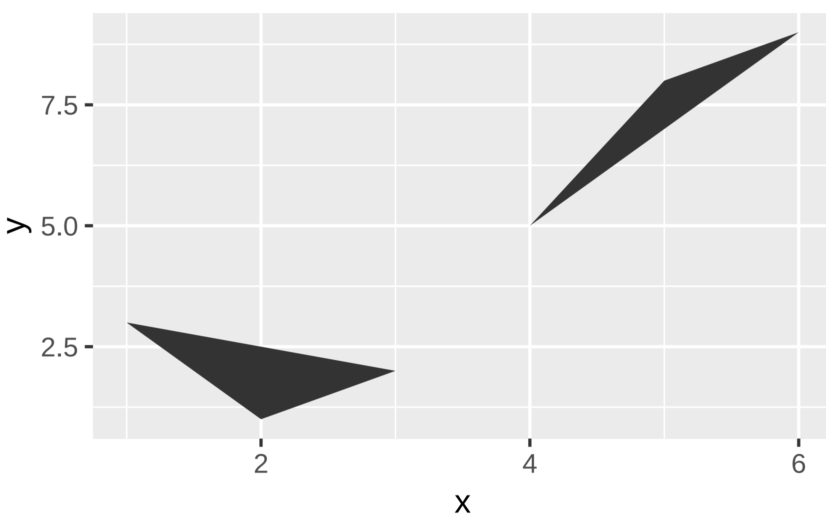
Visual elements in `ggplot2` are called `geoms` (as in geometric objects). The appearance and location of these geoms are controlled by the aesthetic properties. There are many different `geoms` to choose from in `ggplot`, but for creating maps, `geom_polygon()` will be the most useful:

```
ggplot(df, aes(x, y)) +
  geom_polygon()
```



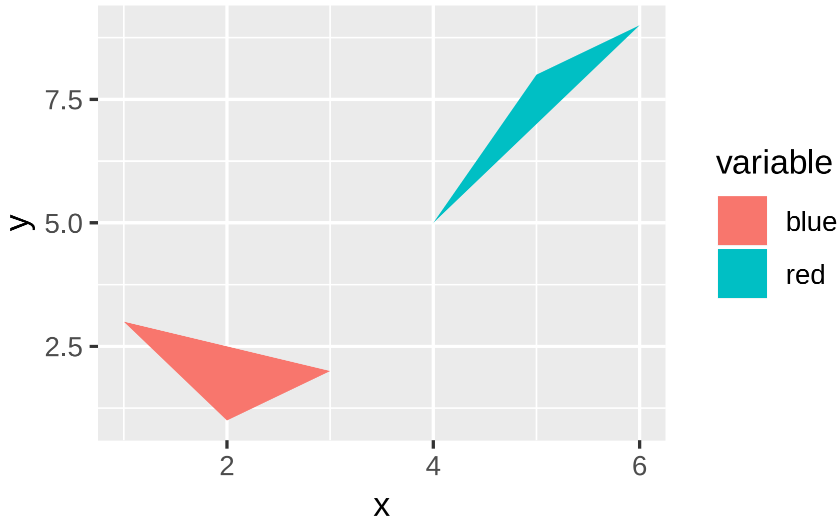
You will notice that this plot does not contain the two triangular islands that I promised above. The reason for this is that if we feed `geom_polygon()` only x and y coordinates, it just connects the points with no regard for how the points are grouped. This is where the grouping variable comes in. The grouping variable will tell `geom_polygon()` that each triangle is its own distinct region:

```
ggplot(df, aes(x, y, group = group)) +
  geom_polygon()
```



This looks better, but it doesn't tell us much about the triangles. From the `variable` column in our data, `df`, we know that triangle one (`t1`) is categorized as `red` and triangle two (`t2`) is categorized as `blue`. We can illustrate this in the plot with a `fill` aesthetic:

```
ggplot(df, aes(x, y, group = group, fill = variable)) +
  geom_polygon()
```



This basic example provides the foundation for plotting geographic data in `ggplot`. We will follow similar procedures throughout the workshop, though the polygons we plot will be Canadian geographic units, not triangles. Moreover, we will work with actual latitudinal and longitudinal values rather than fictional `x` and `y` coordinates.

4 *Choropleth maps*

We begin with standard choropleth[3] maps. `mapcan` users can take their pick from a number of Canadian geographic boundaries: census division, federal riding, Quebec provincial riding[4], and provincial.

The `mapcan()` function returns a data frame with geographic data that can be used in `ggplot2`. To make a standard choropleth map of Canada, we need two ingredients: geographic coordinates and geographic data.

[3] "A thematic map where areas are shaded according to the value of a variable, such as population characteristics or political voting behaviour. The data displayed are aggregated for defined areal units such as wards or counties, and colour-coded using a progressive scale to denominate high and low values." (Castree, Kitchin, and Rogers 2013)

[4] Future iterations of the package will incorporate a broader range of provincial riding boundaries

4.1 *Obtaining geographic coordinates with `mapcan()`*

We will use the `mapcan()` function to access geographic coordinates. At the most basic level, `mapcan()` requires two arguments: `boundaries` and `type`.

- Set `boundaries = province` if your geographic data is at the provincial level, `boundaries = census` for data at the census division level, or `boundaries = ridings` for geographic data at the federal riding level

- To produce geographic data for a standard choropleth map, set `type = standard`.[5]

Let's start with a map of Canadian provinces and territories:

```
mapcan(boundaries = provinces,
       type = standard) %>%
  head(n = 3L)
```

```
##      long     lat order  hole piece
## 1 8308393 2582704     1 FALSE     1
## 2 8307867 2580301     2 FALSE     1
## 3 8310787 2580545     3 FALSE     1
##   pr_sgc_code group
## 1          10  10.1
## 2          10  10.1
## 3          10  10.1
##                  pr_english
## 1 Newfoundland and Labrador
## 2 Newfoundland and Labrador
## 3 Newfoundland and Labrador
##                 pr_french pr_alpha
## 1 Terre-Neuve-et-Labrador       NL
## 2 Terre-Neuve-et-Labrador       NL
## 3 Terre-Neuve-et-Labrador       NL
```

As we can see, `mapcan()` returns a data frame with all of the components we need to create a provincial-level choropleth.

[5] Though we focus on choropleth maps in this section, `mapcan()` can also be used to produce population cartogram data (for maps that alter the geography based on the population size at the province or census division level) with `type = cartogram` and tile grid map data at the federal riding level with `type = bins`. Note that while `type = bins` will provide *data* (i.e. coordinates) for use in tile grid maps, `mapcan::riding_binplot()` is a superior option for creating actual tile/hexagonal grid *maps* in `ggplot`. This will be demonstrated later.

Notice the use of the pipe operator `%>%` in this code. In this instance, the pipe passes the dataframe generated by `mapcan()` through to the `head()` function. The head function is used here to print out only the first 3 observations from the dataframe, giving us a general idea of what the data looks like without printing the whole dataframe. The pipe operator makes code more readable, reduces the number of nested functions, and makes it easy to add steps anywhere in the sequence of operations. It will be used regularly throughout the workshop. Read more here.

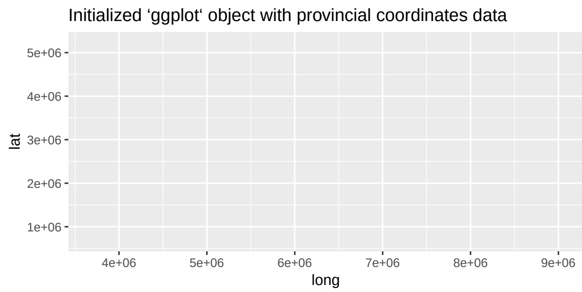
4.2 *Plotting geographic coordinates with* `ggplot`

To create a choropleth with data from `mapcan()`, the following aesthetic mappings are required: `x = long` (longitude), `y = lat` (latitude), and `group = group` (this tells `geom_polygon()` how to group observations—in this case, provinces).

Though we don't have any geographic data just yet, we can still plot these coordinates to make a map in `ggplot`. Let's initialize a `ggplot` object with our provincial coordinates data:

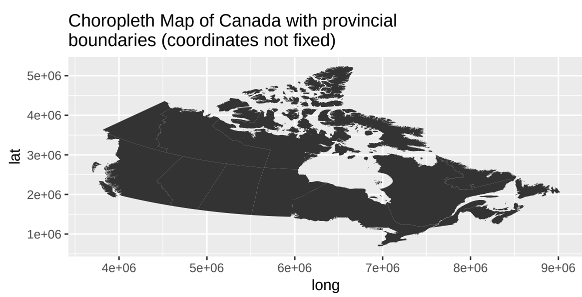
```
# Obtain coordinate data
pr_coordinates <- mapcan(boundaries = province,
       type = standard)

# Initialize ggplot object
pr_map <- ggplot(data = pr_coordinates,
                 mapping = aes(x = long,
```

```
                                   y = lat,
                                   group = group)) +
  # Add a title
  ggtitle("Initialized ‘ggplot‘ object with provincial coordinates data")

# Print ggplot
pr_map
```



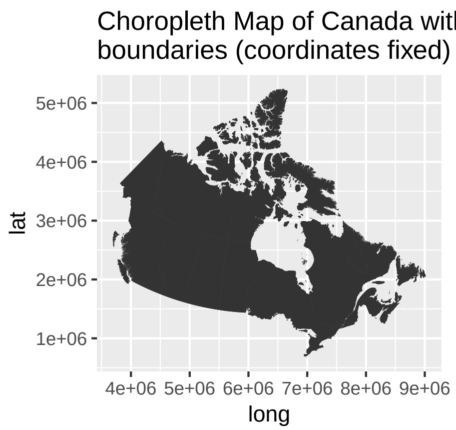
This plot is not very exciting. We need to add a `geom` to visualize the map. To draw maps using `ggplot`, we use `geom_polygon()`, which, as its name suggests, draws polygons based on the `x` and `y` coordinates that we provide.

```
pr_map <- pr_map +
  geom_polygon() +
  ggtitle("Choropleth Map of Canada with provincial \nboundaries (coordinates not fixed)")
pr_map
```



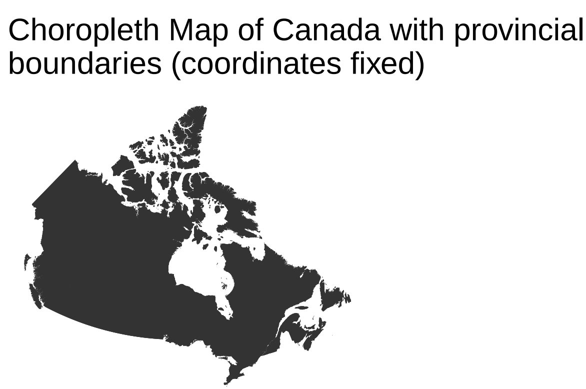
Something is off here—Canada is more squished than we remember. To get an accurate map of Canada, we need to specify `coord_fixed()`. This fixes the relationship between the axes such that one unit on the x-axis (longitude) is the same length as one unit on the y-axis (latitude):

```
pr_map <- pr_map +
  geom_polygon() +
  # Fix the coordinates
  coord_fixed() +
  ggtitle("Choropleth Map of Canada with provincial \nboundaries (coordinates fixed)")

pr_map
```



You will also notice that the axis text has no substantive significance. You can remove it, along with the axis ticks and background grid using `theme_mapcan` function, a `ggplot` theme that is part of the `mapcan` package:

```
pr_map <- pr_map +
  geom_polygon() +
  # Fix the coordinates
  coord_fixed() +
  theme_mapcan()

pr_map
```



Though beautiful, this map is not very informative (unless you are unfamiliar with the shape of Canada). Let's add some province-level data.

4.3 *Incorporate provincial-level statistics*

It is relatively straightforward to merge your own province-level statistics into the geographic data that `mapcan()` provides. To illustrate, we will work with the `province_pop_annual` data frame that is included in the `mapcan` package. This dataset provides annual provincial/territorial population estimates dating back to 1971. Let's use the most recent population data, from 2017:

```
pop_2017 <- mapcan::province_pop_annual %>%
  filter(year == 2017)

head(pop_2017)
```

```
##   year                  province population
## 1 2017                    Canada   36708083
## 2 2017 Newfoundland and Labrador     528817
## 3 2017      Prince Edward Island     152021
## 4 2017               Nova Scotia     953869
## 5 2017             New Brunswick     759655
## 6 2017                    Quebec    8394034
```

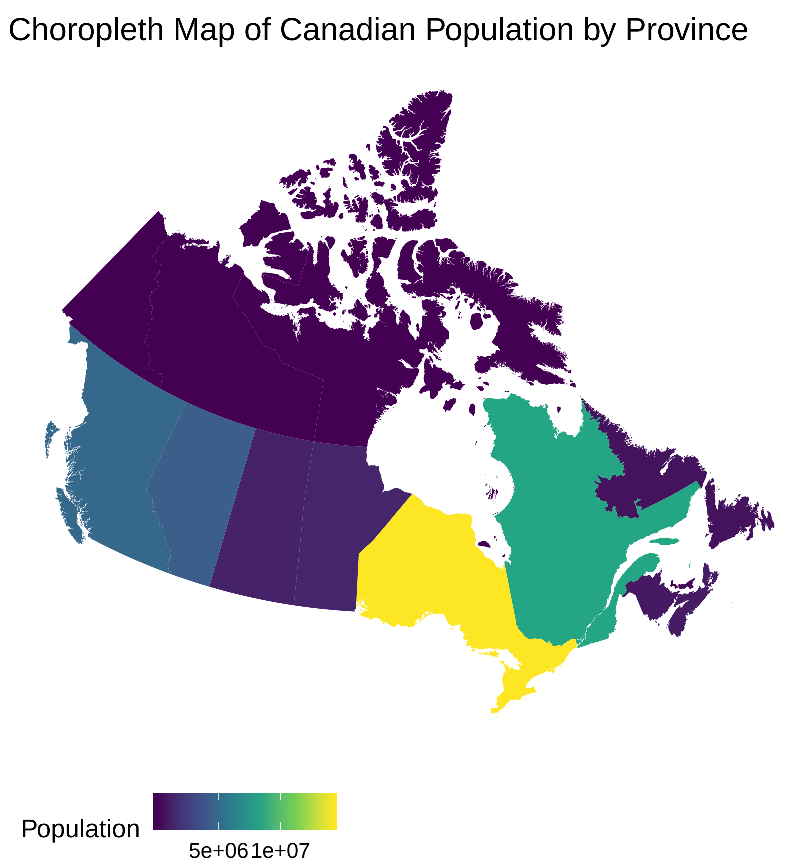
The next step is to attach these numbers to the coordinates in the pr_coordinates object we created above. To do this, we will use the inner_join() function from the dplyr package to merge in the pop_2017 data[6]

[6] Note that the provincial coordinate created by mapcan() provides a number of variables on which to join our own geographic data: full province names (in English or French), 2-letter provincial identifiers, and 2-number Province Standard Geographical Classification (SGC) codes.

```
pr_coordinates <- inner_join(pr_coordinates,
           pop_2017,
           by = c("pr_english" = "province"))
```

To colour the provinces according to their population size, set the population variable as a fill aesthetic. Because population is a continuous variable (and because I don't like the default colours), I will use the scale_fill_viridis_c() colour scale to colour the map.

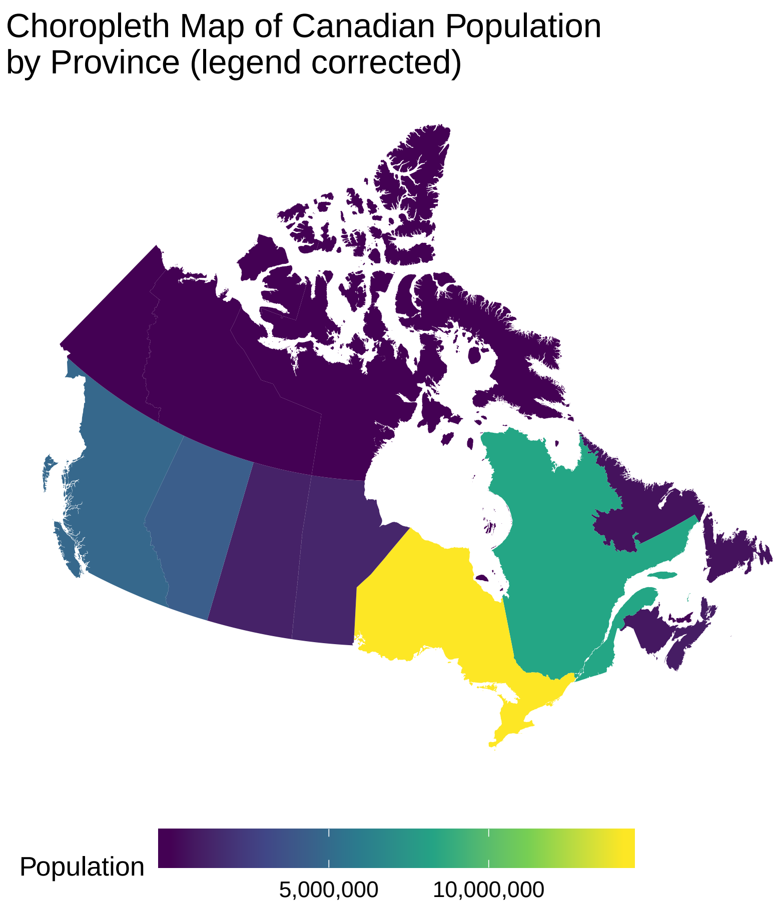
```
pr_coordinates %>%
  ggplot(aes(x = long,
             y = lat,
             group = group,
             fill = population)) +
  geom_polygon() +
  coord_fixed() +
  theme_mapcan() +
  scale_fill_viridis_c(name = "Population") +
  ggtitle("Choropleth Map of Canadian Population by Province")
```



This confirms our suspicion that Ontario and Quebec are the most populous provinces, yet the legend looks like it still needs some work. Using the `scales` package, we can add some commas to these large numbers to make them more readable. We can also extend the width of the legend using the `legend.key.width` argument in `ggplot`'s `theme()` function. Lastly, we can also add a title to the plot with `ggplot`'s `ggtitle()` function:

```
# Run install.packages("scales") if you do not have this package installed
library(scales)

pr_coordinates %>%
  ggplot(aes(x = long,
             y = lat,
             group = group,
             fill = population)) +
  geom_polygon() +
  coord_fixed() +
  theme_mapcan() +
  # Add labels = comma from the scales package
  scale_fill_viridis_c(name = "Population",
```

```
                          labels = comma) +
  # Increase the width of the legend
  theme(legend.key.width = unit(1.5, "cm")) +
  # Add a title
  ggtitle("Choropleth Map of Canadian Population \nby Province (legend corrected)")
```



4.4 *Creating a choropleth map with federal riding boundaries*

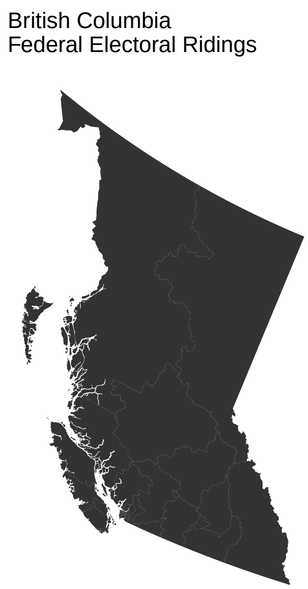
It is likely that your geographic data are at a lower level of aggregation than the provincial level. Students of Canadian politics may find federal ridings more relevant. For the sake of illustration, let's visualize the federal ridings of British Columbia.

```
bc_ridings <- mapcan(boundaries = ridings,
       type = standard,
       # Subset data to only include British Columbia
       province = BC)

head(bc_ridings, n = 3L)

##           long      lat order  hole piece
```

```
## 20030 -1917601 428903.3 20030 FALSE     1
## 20031 -1917311 426212.4 20031 FALSE     1
## 20032 -1919423 421595.4 20032 FALSE     1
##       riding_code   group
## 20030       59001 59001.1
## 20031       59001 59001.1
## 20032       59001 59001.1
##       riding_name_english riding_name_french
## 20030          Abbotsford         Abbotsford
## 20031          Abbotsford         Abbotsford
## 20032          Abbotsford         Abbotsford
##       pr_sgc_code       pr_english
## 20030          59 British Columbia
## 20031          59 British Columbia
## 20032          59 British Columbia
##                 pr_french pr_alpha
## 20030 Colombie-Britannique       BC
## 20031 Colombie-Britannique       BC
## 20032 Colombie-Britannique       BC
##       centroid_long centroid_lat
## 20030      -1929376     423992.3
## 20031            NA           NA
## 20032            NA           NA
```

Notice that `mapcan()` also has a `province` argument, which subsets the coordinate data to include only one province. Having obtained the coordinate data we need, we can draw a map:

```
ggplot(bc_ridings, aes(x = long, y = lat, group = group)) +
  geom_polygon() +
  coord_fixed() +
  theme_mapcan() +
  ggtitle("British Columbia \nFederal Electoral Ridings")
```



Like with our province-level statistics above, we can also merge our own riding-level statistics into the riding-level geographic data that `mapcan()` has produced. We will work with the `federal_election_results` data frame that is included in the `mapcan` package. This dataset provides federal election results dating back to 1996 with variables for riding codes/names, provinces, population, voter turnout and the winning party for each riding. We will use the results of 2015 federal election to colour the ridings in British Columbia.[7]

[7] At the moment, `mapcan()` only provides coordinate data for the electoral boundaries (2013 Representation Order) of the 2015 federal election. Future iterations of the package will incorporate coordinate data from earlier representation orders.

```
bc_results <- mapcan::federal_election_results %>%
  # Restrict data to include just 2015 election results from BC
  filter(election_year == 2015 & pr_alpha == "BC")
head(bc_results, n = 3L)
```

```
##      riding_name_english
## 1             Abbotsford
## 2 Burnaby North--Seymour
## 3          Burnaby South
##       riding_name_french riding_code
## 1             Abbotsford       59001
## 2 Burnaby North--Seymour       59002
## 3          Burnaby South       59003
##   population voter_turnout        candidate
## 1      96819          69.7         Fast, Ed
## 2     100632          70.3     Beech, Terry
## 3     105037          60.8 Stewart, Kennedy
##   election_year        party pr_alpha
## 1          2015 Conservative       BC
## 2          2015      Liberal       BC
```

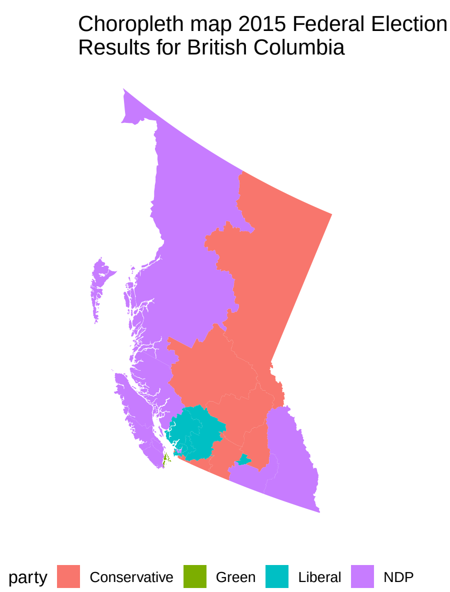
```
## 3          2015          NDP       BC
##                pr_french pr_sgc_code
## 1 Colombie-Britannique          59
## 2 Colombie-Britannique          59
## 3 Colombie-Britannique          59
##           pr_english
## 1 British Columbia
## 2 British Columbia
## 3 British Columbia
```

Next, we merge the two data frames (i.e. the coordinate data and the election results data):

```
bc_ridings <- inner_join(bc_results, bc_ridings, by = "riding_code")
```

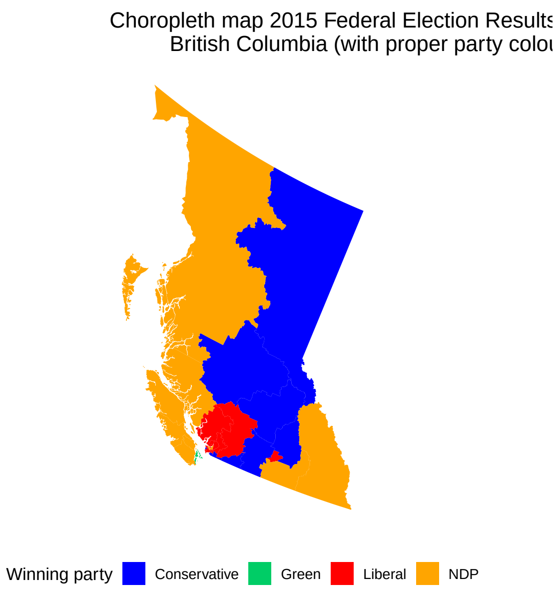
To colour the ridings according the winning party of the 2015 election, set the party variable as a fill aesthetic:

```
bc_riding_map <- bc_ridings %>%
  ggplot(aes(x = long, y = lat, group = group, fill = party)) +
  geom_polygon() +
  coord_fixed() +
  theme_mapcan() +
  ggtitle("Choropleth map 2015 Federal Election \nResults for British Columbia")

bc_riding_map
```

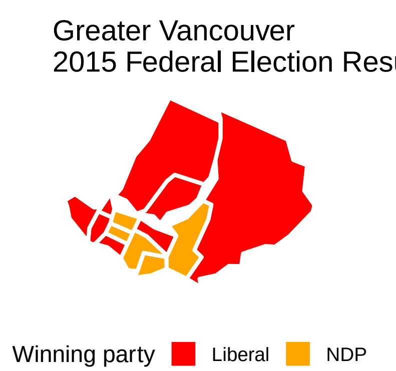


These default colours are not ideal. We can easily provide our own custom colours that correspond to the colours associated with the different parties. Because these are categorical values, we use ggplot's `scale_fill_manual()` (as opposed to `scale_fill_continuous()`, `scale_fill_viridis_c()`, and so forth):

```
bc_riding_map +
  scale_fill_manual(name = "Winning party",
                    values = c("blue",
                               "springgreen3",
                               "red",
                               "Orange")) +
  ggtitle("Choropleth map 2015 Federal Election Results for
          British Columbia (with proper party colours)")
```



This map concludes our foray into standard choropleths and serves as a useful segue into the next section. Although it plots exactly what we had intended—winning parties by riding in British Columbia—this map is not a very useful visual aid for understanding how Canadians voted. It overemphasizes larger geographical units by assigning them a stronger visual weight. Because it is not land area itself we are interested in, this leads to a distorted impression of the data. For instance, as the figure to the right demonstrates, there are several geographically small ridings in Greater Vancouver that are barely visible in the larger map.



5 *Tile/hexagonal grid maps*

Although Canada is a very large country, around 80% of its population is concentrated in urban areas close to the 49^{th} parallel. Because of this, the majority of federal electoral ridings are small and urban while a much smaller number of ridings in sparsely-populated regions account for the majority of Canada's landmass. This makes visualizing statistics at the riding level with standard cloropeth maps less than ideal. The `riding_binplot()` function of the `mapcan` package offers an alternative: the hexagonal/tile grid map. Inspired by the `statebins` package[8], federal riding hexagonal/tile grid maps are a way of visualizing riding-level data when it is not neccessary to accurately represent the ridings geographically.[9]

`riding_binplot()` can be used to create hexagonal and tile grid maps at federal and provincial riding levels (note that only Quebec provincial ridings are supported right now). This function works somewhat differently than `mapcan()` in that it returns a `ggplot` object rather than a dataframe with geographic coordinates.

8 (Rudis 2017)

9 Given the geographic distribution of Canadian ridings, it is challenging to maintain a recognizable shape of Canada while also making riding squares faithful to their actual geographic location. For this reason, `riding_binplot()` is most useful if you are interested in how provinces compare in the distribution of some riding-level variable, but not necessarily in the actual locations of the ridings themselves. If you have ideas on how to provide a more geographically faithful mapping of riding tiles, or some other mapping technique that can be included in future iterations of the package, `mapcan`'s creators are interested in hearing from you!

5.1 *Preparing data for use with `riding_binplot()`*

`riding_binplot()` takes a dataframe as its input. This dataframe needs to have the following two components: (1) the *riding-level variable* you wish to visualize and (2) a *numeric riding code* variable.

Though you can use a dataset of your own choosing, we will once again use `mapcan`'s built-in federal election data (the `federal_election_results` data frame):

```
head(mapcan::federal_election_results, n = 3L)
```

```
##              riding_name_english
## 1 Bonavista--Trinity--Conception
## 2            Burin--St. George's
## 3            Gander--Grand Falls
##               riding_name_french riding_code
## 1 Bonavista--Trinity--Conception       10001
## 2            Burin--St. George's       10002
## 3            Gander--Grand Falls       10003
##   population voter_turnout        candidate
## 1      94842          54.2 Mifflin, Fred J.
## 2      79263          54.7   Matthews, Bill
## 3      82408          44.1    Baker, George
##   election_year                   party
## 1          1997                 Liberal
## 2          1997 Progressive Conservative
## 3          1997                 Liberal
##   pr_alpha               pr_french
## 1       NL Terre-Neuve-et-Labrador
## 2       NL Terre-Neuve-et-Labrador
## 3       NL Terre-Neuve-et-Labrador
##   pr_sgc_code                pr_english
```

```
## 1          10 Newfoundland and Labrador
## 2          10 Newfoundland and Labrador
## 3          10 Newfoundland and Labrador
```

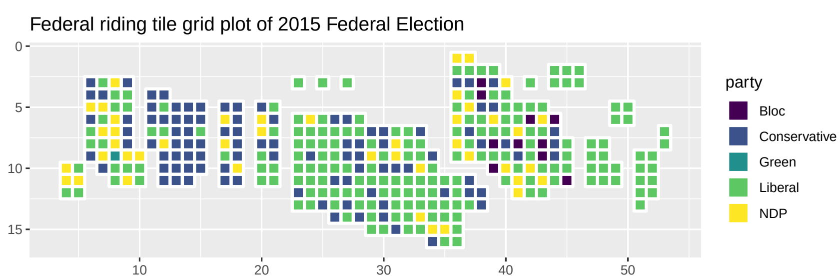
Let's restrict the `federal_election_results` data to include only observations (ridings) from the 2015 election. Because we only need (1) a riding characteristic variable and (2) a riding code variable, we select only these columns from the data.

```
fed_2015 <- mapcan::federal_election_results %>%
  filter(election_year == 2015) %>%
  dplyr::select(riding_code, party)
```

5.2 *Creating a tile grid map of federal ridings with `riding_binplot()`*

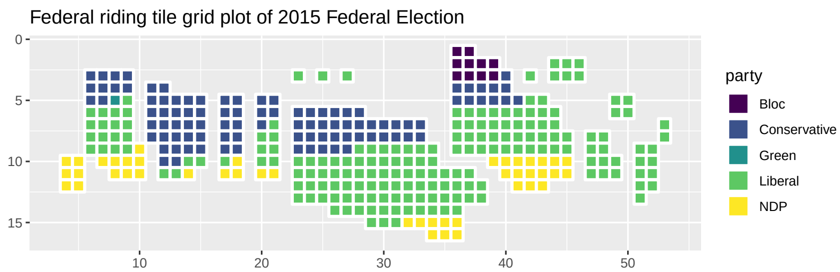
Using the `riding_binplot()` function, we specify the riding-level variable of interest (winning party, `party`) in the `value_col` argument. In the `riding_col` argument, we input the numeric riding code variable (`riding_code`). Lastly, `riding_binplot()` wants to know whether we are feeding it continuous or discrete (categorical) data. Because `party` is a categorical variable, we specify `continuous = FALSE`.

```
riding_binplot(riding_data = fed_2015,
               # Riding level characteristic
               value_col = party,
               # Riding code
               riding_col = riding_code,
               # Set continuous = FALSE if you have a categorical variable
               continuous = FALSE) +
  ggtitle("Federal riding tile grid plot of 2015 Federal Election")
```



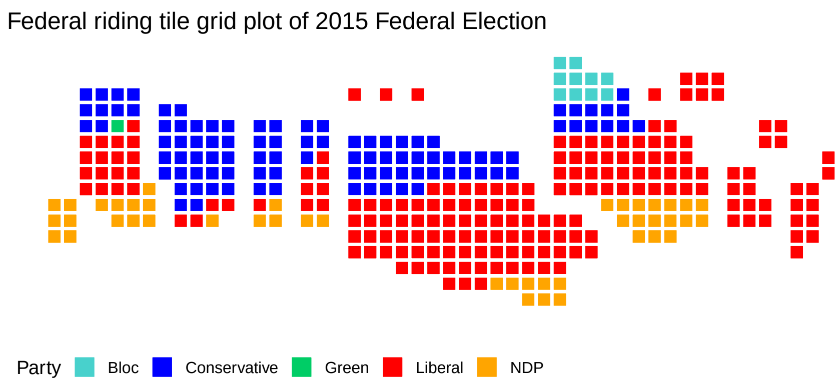
This is a good start, but we can do more. In this tile grid map, tiles are arranged by province yet, because seats are highly concentrated in urban areas, each tile only roughly corresponds to the geographic location of the riding it represents. The `arrange = TRUE` option provides a better representation of the disribution of variables within provinces when the exact location of the riding is not relevant.

```r
riding_binplot(riding_data = fed_2015,
               # Riding level characteristic
               value_col = party,
               # Riding code
               riding_col = riding_code,
               # Set continuous = FALSE if you have a categorical variable
               continuous = FALSE,
               arrange = TRUE) +
  ggtitle("Federal riding tile grid plot of 2015 Federal Election")
```

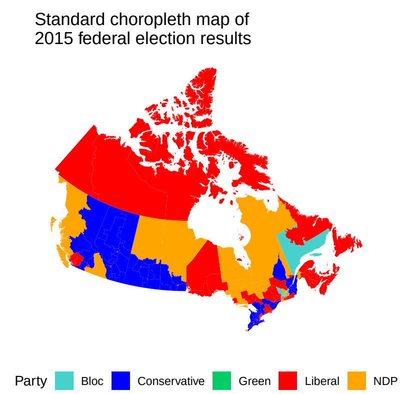


This gives us a clearer representation of the partisan makeup of each province. As with the choropleth maps we created above, the axis text has no substantive significance (it is just the coordinates of the tiles). You can remove it, along with the axis ticks and background grid using the `theme_mapcan()` function, a `ggplot` theme that is part of the `mapcan` package. We can also once again provide our own custom colours that correspond to the colours associated with the different parties using `scale_fill_manual()`. Because `riding_binplot()` outputs a `ggplot` object, implemented these changes is as simple as adding them onto the end the `riding_binplot()` function.

```r
riding_binplot(riding_data = fed_2015,
               # Riding level characteristic
               value_col = party,
               # Riding code
               riding_col = riding_code,
               # Set continuous = FALSE if you have a categorical variable
               continuous = FALSE,
               arrange = TRUE) +
  theme_mapcan() +
  scale_fill_manual(name = "Party",
                    values = c("mediumturquoise", "blue", "springgreen3", "red", "orange")) +
  ggtitle("Federal riding tile grid plot of 2015 Federal Election")
```



Compared to a standard cloropeth map (pictured on the right), this tile grid map gives us a better sense of how the 2015 federal election shaped up. Now that each federal riding is the same size, vast and sparsely populated areas not longer get all the attention.

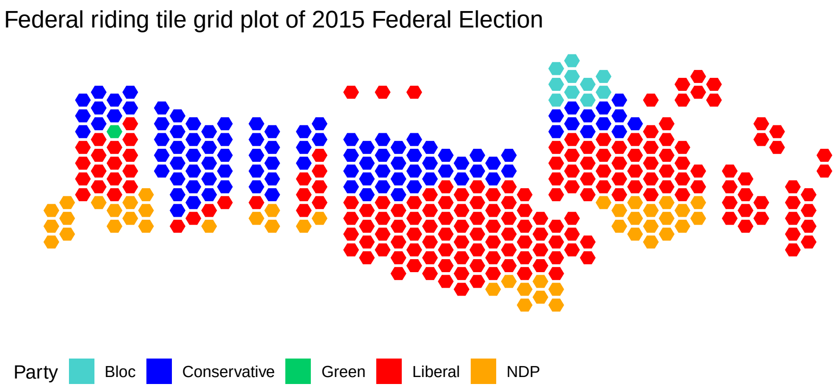


5.3 *Creating a hexagonal grid map with `riding_binplot()`*

Instead of square tiles, we can also create represent federal ridings as hexagons with `riding_binplot()`. Hexagonal grid maps have become increasingly popular to visualize spatial data, though whether you choose hexagons or tiles is a matter of taste.

To create a hexagonal grid map, we use the same code as above, except this time we pass the argument `shape = "hexagon"` to the `riding_binplot()` function. Note that `shape = "square"` is the default so, if squares are what you desire, you do not need to specify this.

```
riding_binplot(riding_data = fed_2015,
               # Riding level characteristic
               value_col = party,
               # Riding code
               riding_col = riding_code,
               # Set continuous = FALSE if you have a categorical variable
               continuous = FALSE,
               arrange = TRUE,
               shape = "hexagon") +
  theme_mapcan() +
  scale_fill_manual(name = "Party",
                    values = c("mediumturquoise", "blue", "springgreen3", "red", "orange")) +
  ggtitle("Federal riding tile grid plot of 2015 Federal Election")
```



5.4 *Creating a tile grid map of Quebec provincial ridings with `riding_binplot()`*

`riding_binplot()` can also be used to create a tile or hexagonal grid map for Quebec provincial ridings. In this case, unlike with the federal riding map, the positions of the tiles/hexagons are arranged to mimic the actual location of the provincial ridings.[10]

[10] The positions of the Quebec riding tiles in `riding_binplot()` were drawn from Bob Rudis' Quebec hex grid map and from a CBC inforgraphic of the Quebec provincial election.

To create our Quebec tile/hexagonal grid map, we first need a Quebec provincial riding-level variable matched to Quebec provincial riding codes. The `mapcan` package comes pre-loaded with Quebec provincial data (though, once again, `riding_binplot()` is designed for use with any data at the riding level). Let's take a look at this data:

```
mapcan::quebec_provincial_results %>%
  head(n = 3L)
```

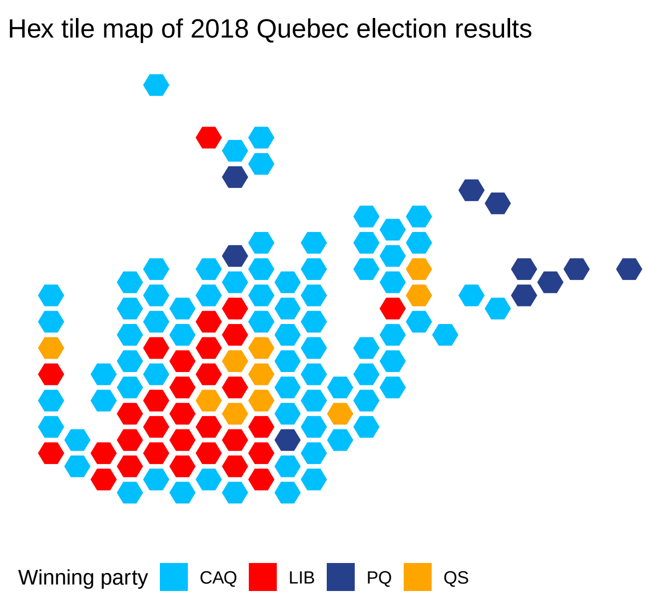
```
##   party vote_share            candidate
## 1   CAQ       42.2        Pierre Dufour
## 2   CAQ       34.1        Suzanne Blais
## 3   LIB       53.8  Christine St-Pierre
##   riding_code   riding_name
## 1         648   Abitibi-Est
## 2         642 Abitibi-Ouest
## 3         338        Acadie
##                  region
## 1 Abitibi-Temiscamingue
## 2 Abitibi-Temiscamingue
## 3              Montreal
```

As can be seen, this data frame contains variables for the winning party of the riding, the vote share of the winning candidate, the candidate's name, the riding code, the riding name, and the region.

We will create a hexagonal grid map to visualize the winning part in each region. When plotting Quebec provincial ridings in hex/tile grid form, we work with two additional arguments to the `riding_binplot()` function: `provincial` and `province`. We set `provincial = TRUE` to let `riding_binplot()` know we are plotting provincial ridings, and `province = QC` to specify the province. Though only Quebec is supported right now, future iterations of the `mapcan` package will provide this functionality for all of the provinces.

Next, the `party` variable (our riding-level variable of interest) goes in the `value_col` argument, and `riding_code` (numeric codes of the ridings) goes in the `riding_col` argument:

```
riding_binplot(quebec_provincial_results,
               value_col = party,
               riding_col = riding_code,
               continuous = FALSE,
               provincial = TRUE,
               province = QC,
               shape = "hexagon") +
  scale_fill_manual(name = "Winning party",
                    values = c("deepskyblue1", "red","royalblue4",  "orange")) +
  theme_mapcan() +
  ggtitle("Hex tile map of 2018 Quebec election results")
```



6 *Population cartograms*

So far, we have used `mapcan` to create choropleth and tile/hexagonal grid maps. Tile grid maps help us overcome the area size bias associated with choropleth maps, but they also involve a high degree of abstraction away from the original boundaries of the map. One last type of map that maintains the boundaries of the map, albeit in a distored form, is the population cartogram. Population cartogram substitute geographic area for population information and disort the map accordingly. Based on the geographic distribution of Canadians (most Canadians live near the US border and very few live in the north), these maps are highly distorted.

6.1 Obtaining population cartogram coordinates with `mapcan()`

Making population cartograms in `mapcan` is straightforward and does not differ much from making choropleth maps. We access geographic coordinate data (longitude, latitude, and group) with the `mapcan()` function. First, specify `boundaries = census` for census division boundaries. Next, instead of `type = standard`, which we specify when making a standard choropleth, we use `type = cartogram`.

```
census_carto <- mapcan(boundaries = census,
                       type = cartogram)
head(census_carto)
```

```
##       long      lat order  hole piece  group
## 1 232.9025 129.9149     1 FALSE     1 1001.1
## 2 232.7886 130.3680     2 FALSE     1 1001.1
## 3 232.7423 130.5379     3 FALSE     1 1001.1
## 4 232.8323 130.5547     4 FALSE     1 1001.1
## 5 233.0826 130.8538     5 FALSE     1 1001.1
## 6 233.2843 130.9002     6 FALSE     1 1001.1
##   census_code census_division_name
## 1        1001       Division No. 1
## 2        1001       Division No. 1
## 3        1001       Division No. 1
## 4        1001       Division No. 1
## 5        1001       Division No. 1
## 6        1001       Division No. 1
##   census_division_type pr_sgc_code pr_alpha
## 1      Census division          10       NL
## 2      Census division          10       NL
## 3      Census division          10       NL
## 4      Census division          10       NL
## 5      Census division          10       NL
## 6      Census division          10       NL
##                   pr_english
## 1 Newfoundland and Labrador
## 2 Newfoundland and Labrador
```

```
## 3 Newfoundland and Labrador
## 4 Newfoundland and Labrador
## 5 Newfoundland and Labrador
## 6 Newfoundland and Labrador
##                  pr_french
## 1 Terre-Neuve-et-Labrador
## 2 Terre-Neuve-et-Labrador
## 3 Terre-Neuve-et-Labrador
## 4 Terre-Neuve-et-Labrador
## 5 Terre-Neuve-et-Labrador
## 6 Terre-Neuve-et-Labrador
```

6.2 *Incorporating census division-level statistics*

`mapcan`'s population cartograms are created using census division and provincial boundaries, so your data should be at one of these two levels. Let's create a population cartogram of the percent of foreign-born population by census division. These data are available in the `census_pop2016` data frame, which is included in the mapcan package.

```
head(census_pop2016, n = 3L)
```

```
##   census_code census_division_type
## 1        1001      Census division
## 2        1002      Census division
## 3        1003      Census division
##   pr_sgc_code                pr_english
## 1          10 Newfoundland and Labrador
## 2          10 Newfoundland and Labrador
## 3          10 Newfoundland and Labrador
##   population_2016 population_density_2016
## 1          270348                    29.3
## 2           20372                     3.3
## 3           15560                     0.8
##   land_area_2016 population_2011
## 1        9220.61          262410
## 2        6099.08           21351
## 3       19912.67           16306
##                  pr_french pr_alpha
## 1 Terre-Neuve-et-Labrador       NL
## 2 Terre-Neuve-et-Labrador       NL
## 3 Terre-Neuve-et-Labrador       NL
##   born_outside_canada census_division_name
## 1                8780       Division No. 1
```

```
## 2                 165       Division No. 2
## 3                  80       Division No. 3
##   born_outside_canada_share
## 1               0.032476660
## 2               0.008099352
## 3               0.005141388
```

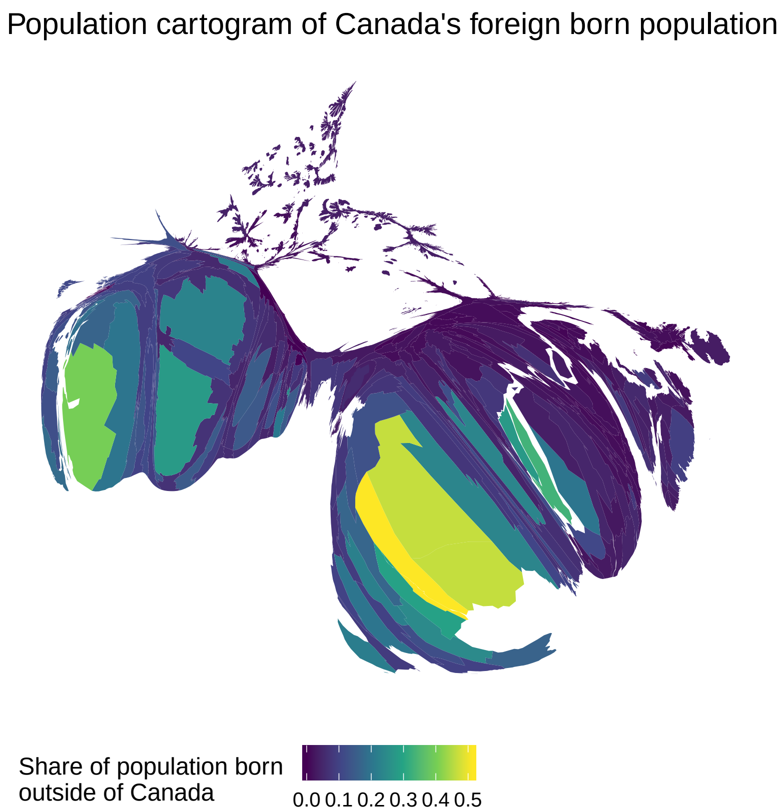
As you can see, this data frame contains a number of census division-level characteristics. For our purposes, we are interested in the `born_outside_canada_share` variable. Let's attach these coordinates to the census division population cartogram coordinates, `census_carto`, we created above:

```
census_carto <- inner_join(census_pop2016,
                           census_carto,
                           by = "census_code")
```

6.3 *Creating a population cartogram with census division boundaries*

To colour the census divisions according to share of population born outside of Canada, we specify `born_outside_canada` as a fill aesthetic. As we did with the choropleth maps above, we will also specify `coord_fixed()` to fix the relationship between longitude and latitude, `theme_mapcan()` to rid the map of unecessary axes, and `scale_fill_viridis_c()` for pretty colours.

```
carto_plot <- ggplot(census_carto, aes(x = long,
                         y = lat,
                         group = group,
                         fill = born_outside_canada_share)) +
  geom_polygon() +
  coord_fixed() +
  theme_mapcan() +
  scale_fill_viridis_c() +
  # Add labels = comma from the scales package
  scale_fill_viridis_c(name = "Share of population born \noutside of Canada") +
  ggtitle("Population cartogram of Canada's foreign born population")
carto_plot
```

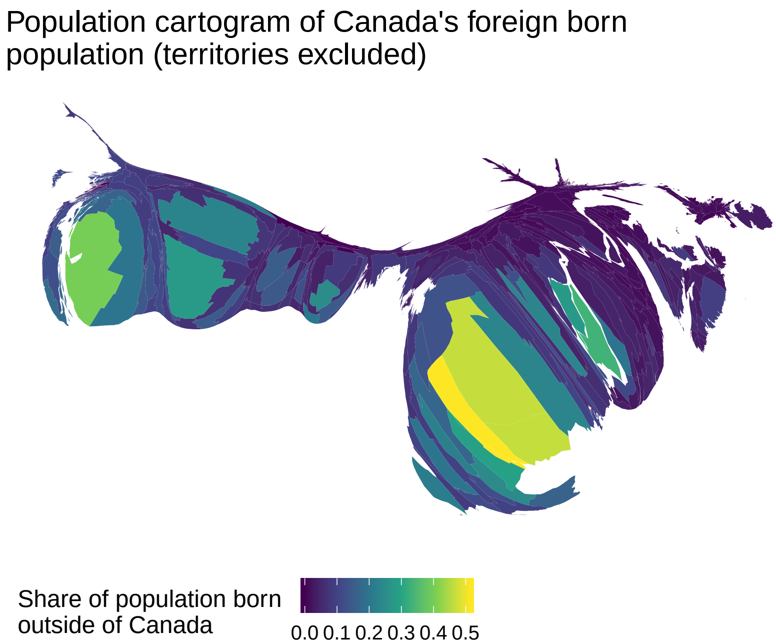


As we can see, because of their high population density, census divisions in Vancouver, Edmonton, Calgary, Toronto, and Montreal feature most prominently on this map. As it happens, these metropolitan areas are where much of Canada's foreign born population reside.

Because the three territories are very sparsely populated, we can exclude these from the map using `territories = FALSE` in the `mapcan()` function.

```
census_carto_noterr <- mapcan(boundaries = census,
                              type = cartogram,
                              territories = FALSE)

# Merge in census data
census_carto_noterr <- inner_join(census_pop2016,
                                  census_carto_noterr,
                                  by = "census_code")

carto_noterr_plot <- ggplot(census_carto_noterr, aes(x = long,
                            y = lat,
                            group = group,
                            fill = born_outside_canada_share)) +
  geom_polygon() +
```

```
  coord_fixed() +
  theme_mapcan() +
  scale_fill_viridis_c() +
  # Add labels = comma from the scales package
  scale_fill_viridis_c(name = "Share of population born \noutside of Canada") +
  ggtitle("Population cartogram of Canada's foreign born \npopulation (territories excluded)")
carto_noterr_plot
```



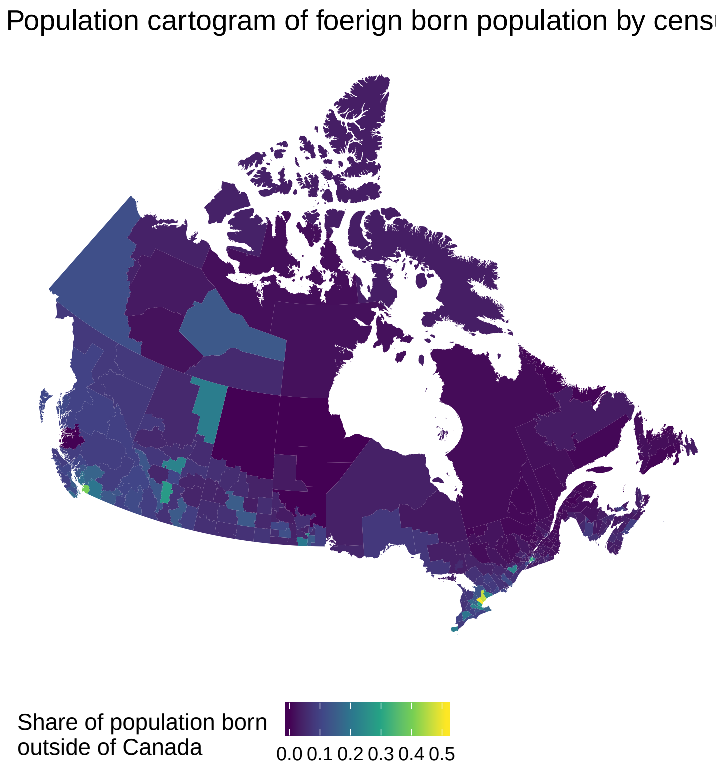
This results in somewhat less distortion, though a fair amount of distortion is still present due to the distribution of Canada's population.

6.4 *Comparing population cartograms to standard choropleth maps*

To illustrate the value of population cartograms for displaying geographic information more accurately, we can compare the cartogram above to a standard choropleth map with the same data and boundaries.

```
# Get population cartogram geograpic data
census_coordinates <- mapcan(boundaries = census,
       type = standard)

# Merge together
census_coordinates <- inner_join(census_coordinates,
                                 census_pop2016,
                                 by = c("census_division_code" = "census_code"))
```

```
ggplot(census_coordinates, aes(long, lat, group = group, fill = born_outside_canada_share)) +
  geom_polygon() +
  scale_fill_viridis_c(name = "Share of population born \noutside of Canada") +
  theme_mapcan() +
  coord_fixed() +
  ggtitle("Population cartogram of foerign born population by census division")
```



Comparing this standard choropleth map the the population cartogram above, it is clear that the standard choropleth map visually understates the share of the population that is foreign born in Canada.

References

Castree, Noel, Rob Kitchin, and Alisdair Rogers. 2013. "Choropleth Map." http://www.oxfordreference.com/view/10.1093/acref/9780199599868.001.0001/acref-9780199599868-e-207.

Rudis, Bob. 2017. *Statebins: Create 'U.S.' Uniform Square State Cartogram Heatmaps*. https://github.com/hrbrmstr/statebins.