

Using Census Mapper

Simon J. Kiss

13/03/2020

Contents

1	Understanding The Hierarchical Nature of Statistics Canada Data	1
2	Geographic Files and Data Files	2
3	Getting a folder for your map assignment	2
4	Using Census Mapper	2
4.1	Installing the package and loading the library	2
4.2	Getting an API	3
4.3	Using your API in R	3
4.4	Getting the data out of Census Mapper.	6
4.5	Drawing the map	10
4.6	Data	11
4.7	How do we add in a third variable.	15

1 Understanding The Hierarchical Nature of Statistics Canada Data

As I mentioned in class the other day, Statistics Canada has a systematic - but complex - way of categorizing its data based on geographic areas. It is hierarchical in nature, so data is aggregated into a variety of levels.

One way is:

1. (PR) Province
- 2.(CD) Census District (or Division, I've seen both)

“Group of neighbouring municipalities joined together for the purposes of regional planning and managing common services (such as police or ambulance services). These groupings are established under laws in effect in certain provinces of Canada. Census division (CD) is the general term for provincially legislated areas (such as county, municipalité régionale de comté and regional district) or their equivalents.”
— Statistics Canada

3. Census Subdivision (CSD)
4. Census Tracts (CSD)

“Area that is small and relatively stable. Census tracts usually have a population between 2,500 and 8,000 persons. They are located in census metropolitan areas and in census agglomerations that have a core population of 50,000 or more.”
— Statistics Canada

5. Dissemination Areas (DA)

“A dissemination area (DA) is a small, relatively stable geographic unit composed of one or more adjacent dissemination blocks. It is the smallest standard geographic area for which all census data are disseminated. DAs cover all the territory of Canada.” — Statistics Canada

In addition, there are other categories that reflect a parallel hierarchy. Probably the most important of these are: Census Metropolitan Areas and Census Agglomerations.

“Area consisting of one or more neighbouring municipalities situated around a core. A census metropolitan area must have a total population of at least 100,000 of which 50,000 or more live in the core. A census agglomeration must have a core population of at least 10,000.” — Statistics Canada

2 Geographic Files and Data Files

As well, the other thing to keep in mind is that we need to get geography files, usually .shp file extension, that contain information about each boundary and an identifier. When you look at .shp files, you'll see that each subdivision in the file that you download will have also has variables that identify which geographic area (Census Tract, Census Subdivision, Census Division) that the boundary belongs to.

This allows for merging to add data to geography files.

Data files can come from anywhere, but the `cancensus` r package makes it very easy to get both the geography and the data file at once. However, you are welcome to take data file from anywhere. The key thing to remember is that the data you get must somehow have geography identifiers that match up with the geography file that you have downloaded.

3 Getting a folder for your map assignment

As with everything in this course, it is very useful to set up a folder with an RStudio project folder specifically for your map assignment. I have already actually created one on the course GitHub page. If you are able to pull changes from the DMJN328 course page in GitHub, by following the instructions here, then you should see a folder called “map assignment” in your course folder (see Figure 1).

Open the .Rproj file with RStudio.

If not, make your own folder in the course folder, call it whatever you want and make an RStudio Project file in that folder.

4 Using Census Mapper

Census Mapper is an open source project that makes it easier to access Statistics Canada's vast troves of census data and link it to geographical files for Canada to make it easier to visualize Canadian census data. It is meant to work seamlessly with R.

4.1 Installing the package and loading the library

As with everything, we need to install packages and load them.

Open an R Script (or you can open the pre-made `cancensus_tutorial.R` script that is in the `map_assignment` folder)

You can start with these commands to install census mapper and load the library.

Note, any code snippets that appear in this document are formatted so that you should be able to copy and paste them into an R script.

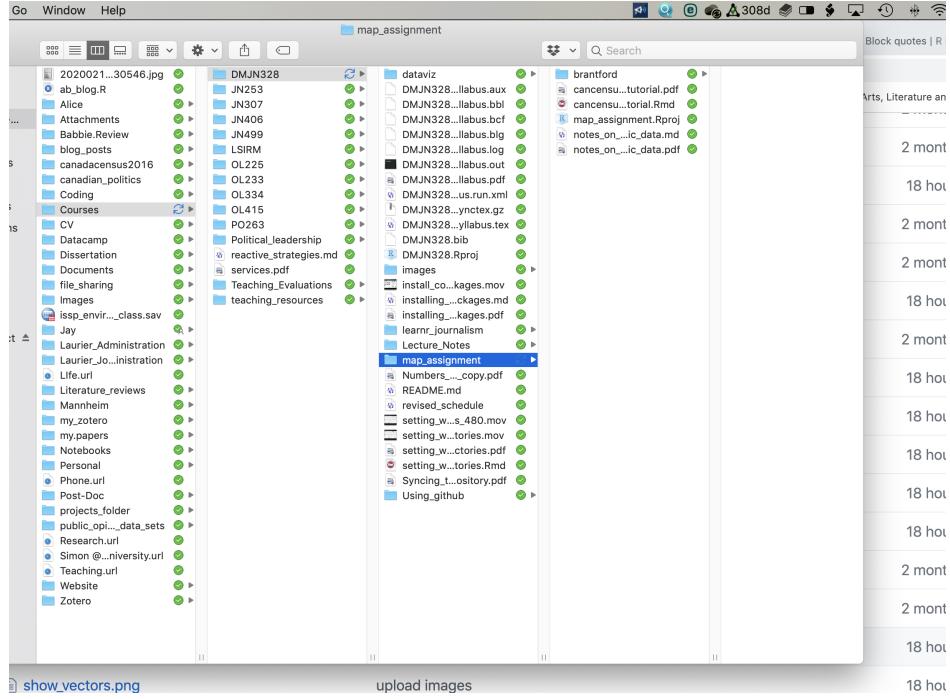


Figure 1: The map assignemnt folder.

```
install.packages('cancensus')
library(cancensus)
```

4.2 Getting an API

While it is free and open source, it is necessary to register for an API. Click on the top-right of the website to login and get an API (see Figure 2).

The API should look something like this (see Figure 3).

4.3 Using your API in R

There are two ways to use your API: temporary and permanent.

4.3.1 Temporary

The temporary way is to insert the following command at the top of any script you write that interacts with census mapper. **Of course, you need to insert your own API key after the = sign.** The one below is a modified version of my own, so it won't work.

```
options(cancensus.api_key='CensusMapper_287500bb91a374ec69fdcf27')
```

After that, any commands you run in that script will interact with Census Mapper nicely.

The downside to the temporary approach is: 1) you need to do it every time you write a script to get data or files from Census Mapper and 2) if you share your code with anyone then you will expose your API. It's not a big deal, but it's something.

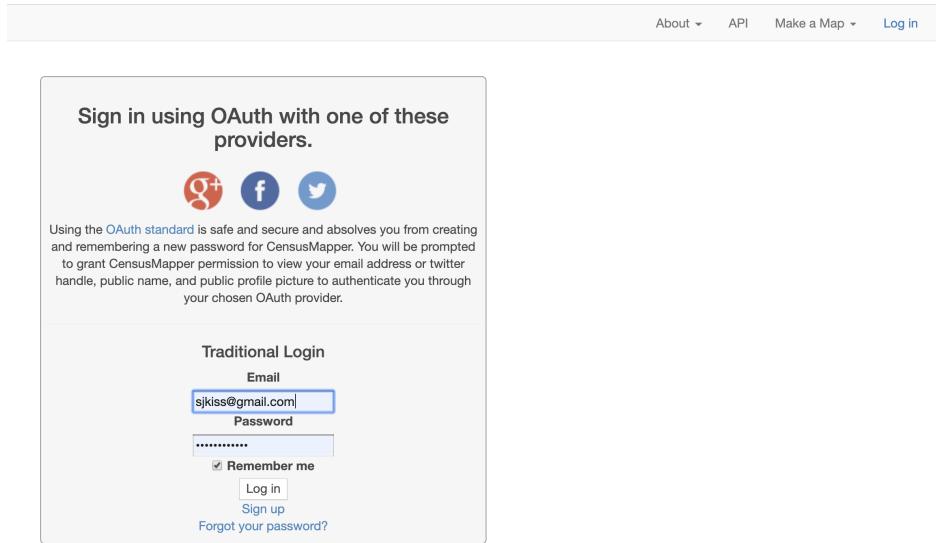


Figure 2: Getting an API

A screenshot of the CensusMapper API key page. The page title is 'Hi Simon Kiss'. Below it, a section titled 'Api Key' contains text explaining that the API key is needed to download CensusData using the 'Censusmapper data download functionality' or the 'cancensus R package'. It also states that the key ties into the CensusMapper data API. A placeholder text 'Your API key is' is followed by a box containing the API key: 'CensusMapper_287500bb91a374ec69fdcf270fb20ff7'. At the bottom, a message says 'You have 19290 points remaining this month and 9760 points remaining this day.' A large 'My Maps' button is at the bottom right.

Figure 3: What an API looks like.

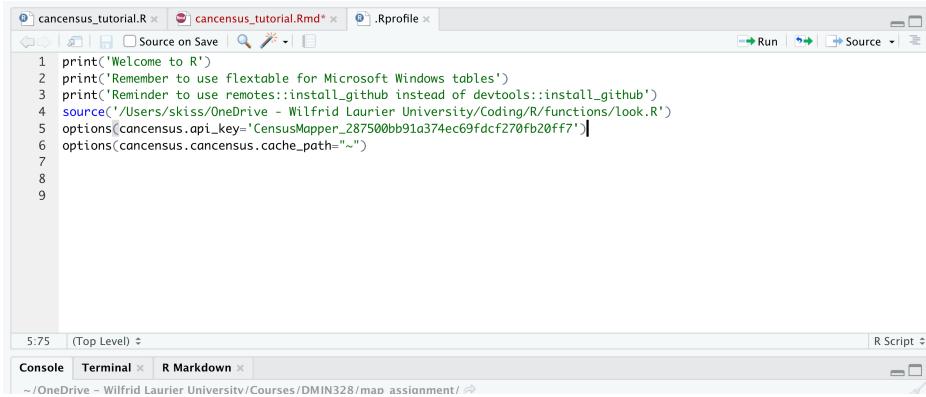
4.3.2 Permanent

The permanent approach sets your `census` API inside R permanently, or as long as you have this computer. The way you do it is to open an invisible file called `.Rprofile` on your computer and insert the options command in there.

The following command opens `.Rprofile` (or makes it if it does not already exist).

```
file.edit(file.path('~/Rprofile'))
```

Figure 4 is what mine looks like. Lines 1:3 are just notes for myself, line 4 is a little command that I use frequently and lines 5 and 6 are my `census` options.



A screenshot of the RStudio interface showing the `.Rprofile` file. The code editor window contains the following R script:

```
1 print("Welcome to R!")
2 print("Remember to use flextable for Microsoft Windows tables")
3 print("Reminder to use remotes::install_github instead of devtools::install_github")
4 source('/Users/skiss/OneDrive - Wilfrid Laurier University/Coding/R/functions/look.R')
5 options(census.api_key='CensusMapper_287500bb91a374ec69fdcf270fb20ff7')
6 options(census.census.cache_path="~")
```

The RStudio interface includes tabs for `Console`, `Terminal`, and `R Markdown`. The status bar at the bottom shows the path `~/OneDrive - Wilfrid Laurier University/Courses/DMIN328/map assignment/`.

Figure 4: Simon's `.Rprofile` file, as an example

So, in your `.Rprofile` file, just plunk the `options()` command with your api key, just like this and save it the file.

```
options(census.api_key='CensusMapper_287500bb91a374ec69fdcf270fb2')
```

4.3.3 CensusMapper Cache

As you can imagine, downloading geographic files and Statistics Canada data can involve quite a lot of data. This can: a) take a long time and b) overload CensusMapper and Statistics Canada. To minimize the burden on everyone, it is recommended (but not necessary) to also set a *cache* for census mapper data.

A cache is just like a food cache; a place to store something you might need in the future.

So, in this case, you set a `census` cache as directory on your computer. When you download data from `census` once, then it will be stored there; if you need to run your script again, the `census` package will *first* look in the cache to see if it is there, allowing you to not go through the hassle of downloading the information from the internet again.

This step is *optional* but, for obvious reasons, recommended and will probably make your life easier.

You set the cache directory the exact same way as setting the API.

Note that you get to choose where you want to put your cache. I find it pretty unobtrusive just to put it in your root user directory, but it's up to you.

To set it *temporarily*, enter this command at the top of your script. Note the `~` is Mac-speak for the user's home directory. This *should* work on a PC, but it might not. You *might* need to play with this a little bit, if you have a PC.

```
options(census.census.cache_path="~")
```

To set it *permanently*, enter the same command in your `.rprofile` folder, following the steps above, save and restart.

If you have gone the permanent route and modified your `.Rprofile` file, you need to restart RStudio for the changes to take effect

4.4 Getting the data out of Census Mapper.

Now we can go over to to get our geography and data files.

Remember that to make our map, we need both some geography files and some data to visualize.

4.4.1 Geography

To select the geographic area you want to work with, play with the cursor to zoom to a region you are interested in.

If you click on “Region Selection” you can zoom quickly to different levels of geography, e.g. Province, census division, census sub-division, census metropolitan area, whatever. I find this convenient.

When you have a geographic boundary area that you are interested in playing with, select it and it should turn *blue*. If you click the same area again, it should deselect the region. Figure 5 is what it looks like if you have selected Ontario.

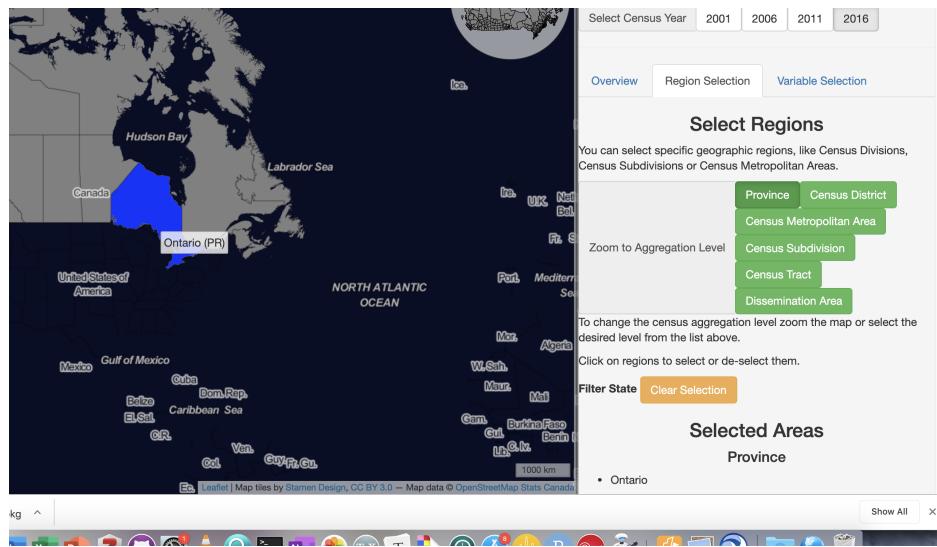


Figure 5: Selecting an area.

Please note: when you select a geographic region, you will still be able to get the boundary files and data from *sub-units*. So, in the image, I have selected Ontario, but at the next step, I will be able to get data and boundaries on sub-units (e.g. census sub-divisions for all of Ontario, or even all the census tracts). Here, I have just excluded other provinces.

So this is a choice you get to make. One thing to keep in mind, the larger the geographic unit that you select, the more data you will be downloading.

I am going to select Kitchener, so I'm going to zoom in on the Census Metropolitan Area level, drag the map to about the KW area and select Kitchener (see Figure 6).

Note that I had to select “Clear Selection” first to deselect Ontario, and then select “Kitchener”.

If I jump over to “Overview” I’m going to get a screen that looks like Figure (7 (below)). There’s some useful information here.

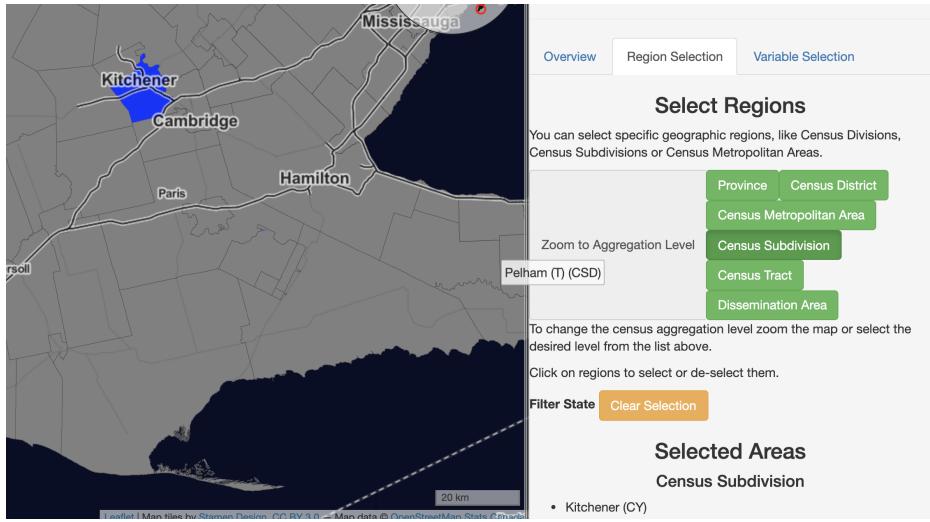


Figure 6: Selecting Kitchener

Notice under “Selected Variables” it says `vectors = []`. That is because we have not yet selected any data to download.

And under “Selected Regions”, it says that we have selected Kitchener (CY) and that `regions = {"CSD": ["3530013"]}`. The `regions`` stuff is code for Census Subdivision (CSD) 3530013.

The more you play with Statistics Canada’s data, the more you will learn how it works, but basically, the CSD code is made of a province code (Ontario = 35), a two-digit Census Division code (Waterloo Region = 30) and a 3-digit Census Sub-division code (City of Kitchener = 013).

So, Statistics Canada has a very systematic way of coding Canada’s geography. Kitchener is always through all of StatsCan’s data coded as 35300013.

Under Geographic Aggregation Level, there isn’t really anything selected. Here, we can select what level of data we want to visualize and what boundaries we want to draw. Because we have only selected the City of Kitchener (which is a Census Sub-Division) there are only three options that we can download:

1. CSD (Census Subdivision) which will download the outer boundaries of the CSD Kitchener
2. CT (Census Tracts) which are about maybe ward-level boundaries inside the City of Ktichener
3. DA (Dissemination Areas) which are almost block-level boundaries inside Kitchener.

Obviously option 3 will give us the most fine-grained view of the City of Kitchener, but you might not get as much data.

I raised this in class the ohter day, but in looking at it, I see that there actually is quite a lot of data. So, let’s do it.

Let’s select DA (see Figure ??).

Then go back to overview. If you look at the bottom, there is some R code there and it should look just like this.

```
census_data <- get_census(dataset='CA16', regions=list(CSD="3530013"), vectors=c(), labels="detailed",
```

I am going to modify it slightly and comment it so we know what is going on.

```
#Instead of saving it as census_data, I am saving it as kitchener, for ease of memory.
#dataset=CA16 says get data or boundary files from the 2016 Census
kitchener<- get_census(dataset='CA16',
```

The screenshot shows a web-based application interface for data selection. At the top, there is a navigation bar with tabs: "Toronto (C) (CSD)" (which is active), "Region Selection", and "Variable Selection". Below the navigation bar, the word "Overview" is centered in a large, bold font. A descriptive text follows: "Before being able to download data you will need to register for an account and log in. API requests are limited in volume, please contact us if you need a larger quota." Under the "Overview" section, there is a heading "Selected Variables" followed by a code snippet: `vectors=[]`. Below this, there is a heading "Selected Regions" followed by "Census Subdivision" and a list: • Kitchener (CY). Another code snippet follows: `regions={"CSD": ["3530013"]}`. At the bottom, there is a section titled "Geographic Aggregation Level" with a button labeled "Select Level for Download". To the right of this button are three blue buttons: "Selected Regions", "CSD", and "CT". Below these is a blue button labeled "DA". A message at the bottom states: "You need to select a geographic level first...". At the very bottom of the interface are two buttons: "Show All" and a close button represented by an "X".

Figure 7: Looking at the overview

- Kitchener (CY)

```
regions={"CSD": ["3530013"]}
```

Geographic Aggregation Level

Select Level for Download

Selected Regions

CSD

CT

DA

```
install.packages("cancensus")
library(cancensus)
options(cancensus.api_key='CensusMapper_287500bb
91a374ec69fdcf270fb20ff7')
options(cancensus.cache_path = <your long term c
ache path>)
```

```
census_data <- get_census(dataset='CA16', region
s=list(CSD="3530013"), vectors=c(), labels="deta
iled", geo_format=NA, level='DA')
```

```
# set geo_format to "sf" or "sp" to also download geographic data
```

Figure 8: Selecting Dissemination Areas

```
regions=list(CSD="3530013"), vectors=c(), labels="detailed",
geo_format="sf", level='DA')
```

This should leave us with an object called `kitchener` in our environment.

Try this command. Note: loading the `tidyverse` library is necessary to run commands like `glimpse()`, `gather()` and `ggplot()`.

```
library(tidyverse)
glimpse(kitchener)
```

```
## Observations: 314
## Variables: 11
## $ `Shape Area` <dbl> 0.78200, 0.33188, 0.56447, 0.18680, 0.16206, 0.15935, ...
## $ Type <fct> DA, DA...
## $ Dwellings <int> 230, 191, 254, 210, 199, 223, 306, 283, 185, 360, 226, ...
## $ Households <int> 223, 168, 244, 210, 194, 217, 284, 268, 185, 357, 215, ...
## $ GeoUID <chr> "35300097", "35300098", "35300099", "35300100", "3530...
## $ Population <int> 588, 402, 552, 491, 425, 513, 584, 532, 459, 530, 374, ...
## $ CD_UID <chr> "3530", "3530", "3530", "3530", "3530", "3530"...
## $ CSD_UID <chr> "3530013", "3530013", "3530013", "3530013", "3530013", ...
## $ CT_UID <chr> "5410024.00", "5410024.00", "5410023.00", "5410023.00"...
## $ CMA_UID <chr> "35541", "35541", "35541", "35541", "35541", "35541", ...
## $ geometry <MULTIPOLYGON [°]> MULTIPOLYGON (((-80.49342 4..., MULTIPOLY...)
```

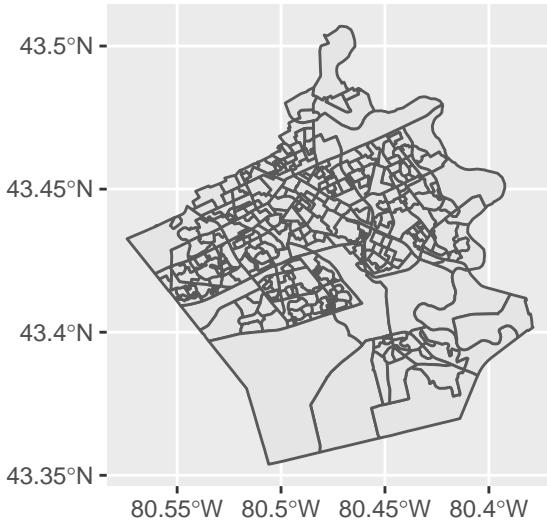
So there you have it: the dissemination area boundary files for the city of Kitchener. There are 314 observations (rows) of 11 variables. We can go through them one-by-one because it is quite informative.

1. **Shape Area** - I think this is the size of each dissemination area.
2. **Type** - is a factor (fct), and it just says that each row is a dissemination area (DA)
3. **Dwellings** - integer (int) variable that says how many dwellings are in each DA
4. **Households** - integer (int) variable that says how many households are in each. I don't quite understand the difference; one would have to read some documentation at Statistics Canada.
5. **GEOUID** - character (so not a number). From the looks of this, this looks like it is the unique identifier for each dissemination Area (more here)
6. **Population** - Integer (int) variable showing the population of each DA
7. **CD_UID** - variable showing the census district identifier 35 for Ontario and 30 for the Census District of Region of Waterloo
8. **CSD_UID** - variable showing the Census sub-division identifier: 35 for Ontario and 30 for the Census District of Region of Waterloo, and 013 for the CSD of Kitchener
9. **CT_UID** - variable showing the Census Tract identifier for the CT that each DA is in. 10. **CMA_UID** - variable showing the ID for the CMA (Census Metropolitan Area) that each DA is in. Kitchener-Waterloo-Cambridge is the CMA 35541 (35 for Ontario and 541) for all of K-W-C.
10. **geometry** - there it; those are the geometry coordinates for each of the DAs.

4.5 Drawing the map

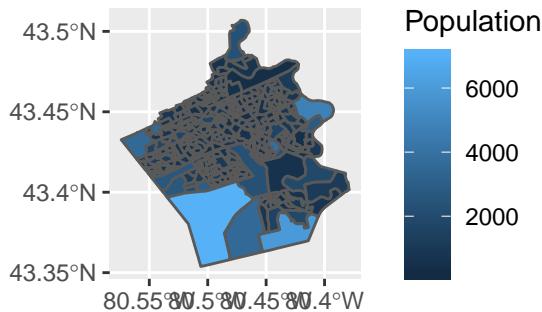
Now it should be dead easy to draw the map, because it's *exactly the way it works in the lesson*.

```
library(ggplot2)
ggplot(kitchener, aes(geometry=geometry))+geom_sf()
```



When you download the boundary files for a region, it looks like you get basic population data. So with just this command, we could fill in each DA with the population #'s exactly the way that we did with the boston data in the exercises.

```
ggplot(kitchener, aes(geometry=geometry, fill=Population))+geom_sf()
```



I'm going to stop there, because we've practiced different ways of gussying up the map in the exercises, and it's pretty easy.

The next section talks about how to add different variables, other than population via `cancensus`

4.6 Data

go back to the web browser and click on “Variable Selection” Honestly, this product is so incredible it kind of makes your head spin (see 9). Bear with me.

```
include_graphics("images/select_kitchener_variables.png")
```

Note that there are two categories at the top: **100% Data** and **25% Data**. The difference between the two is that the first is the results from *every single household and person that filled out the census* and the second is the results of sample drawn from the full census (25% of the results). So, it is a *huge* sample and extremely accurate, but it is still just a *sample*.

Some variables are only released with sample data, basically out of privacy concerns (I think).

You can search by keyword, or browse through by topics.

There are a few things to really note:

1. Each variable appears to be summarized by sex. That is, if you want, you can track the results of each variable in the sense for men, women or total (if you do not care to differentiate by sex.)

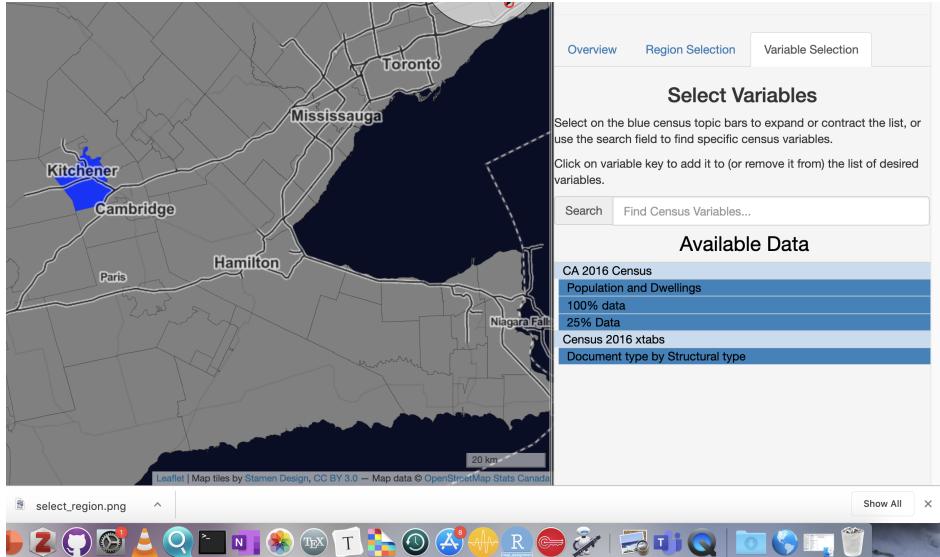


Figure 9: Selecting variables to map on Kitchener

2. Each variable has a code attached to it that starts with a v. e.g. The Total age variable is V_CA16_1. V stands for vector, CA16 is the 2016 Census and 1 is just the identifier of the individual vector.

A vector is a string of numbers that represents the variable in question.

Now, we're not going to get that vector, because it will only tell us the Total number of people who provided Age data to the census.

But if you click on it and you'll see that it expands into more categories, with more vectors. Now, there is a female 0 to 14 years vector, a male 0 to 14 years vector and a total 0 to 14 years vector.

So those three vectors will provide us with the number of females, males and total (both sexes) 0 to 14 years old in each dissemination area in Kitchener.

You can click on the 0 to 14 years vector as well and you see that you get finer-grained vectors. Now you have the number of 0 to 4 year olds, 5 to 9 year olds etc, for males and females and total (both sexes.)

Click on 0 to 4 one more time and you see you get right down to the one year category. Note: not all variables in the census will have vectors with this level of detail.

For now, let's just select the vector that has the total number of 0 to 4 year olds. This will tell us where the toddlers live in Kitchener. Clicking on a vector selects it, and it should appear in the text box above the list of vectors. Clicking on it again, deselects and clears it.

Go back to the Overview screen and you'll see R code again. It should look like this.

```
census_data <- get_census(dataset='CA16', regions=list(CSD="3530013"), vectors=c("v_CA16_4"), labels="de")
```

Notice that this is exactly the same code as above; the only difference is that the vectors argument actually has something in it; namely v_CA16_, which is the total age vector.

I'm going to modify the code a little bit. We'll call the new dataframe kitchener again and remember you have to specify which geographical file format you want.

```
kitchener2 <- get_census(dataset='CA16', regions=list(CSD="3530013"), vectors=c("v_CA16_4"), labels="de")
```

```
## Downloading: 4.5 kB      Downloading: 4.5 kB      Downloading: 4.5 kB      Downloading: 4.5 kB      Downl...
```

Now we glimpse the data again to take a look.

```

glimpse(kitchener2)

## # Observations: 314
## # Variables: 14
## $ `Shape Area` <dbl> 0.78200, 0.33188, 0.56447, 0.18680, 0.162...
## $ Type <fct> DA, DA, DA, DA, DA, DA, DA, DA, DA, D...
## $ Dwellings <int> 230, 191, 254, 210, 199, 223, 306, 283, 1...
## $ Households <int> 223, 168, 244, 210, 194, 217, 284, 268, 1...
## $ GeoUID <chr> "35300097", "35300098", "35300099", "3530...
## $ Population <int> 588, 402, 552, 491, 425, 513, 584, 532, 4...
## $ CD_UID <chr> "3530", "3530", "3530", "3530", "3530", ...
## $ CSD_UID <chr> "3530013", "3530013", "3530013", "3530013...
## $ CT_UID <chr> "5410024.00", "5410024.00", "5410023.00", ...
## $ CMA_UID <chr> "35541", "35541", "35541", "35541", "3554...
## $ geometry <MULTIPOLYGON [°]> MULTIPOLYGON (((-80.49342 4...
## $ `Region Name` <fct> Kitchener, Kitchener, Kitchener, Kitchene...
## $ `Area (sq km)` <dbl> 0.78200, 0.33188, 0.56447, 0.18680, 0.162...
## $ `v_CA16_4: 0 to 14 years` <dbl> 90, 60, 70, 70, 55, 90, 60, 65, 75, 25, 3...

```

Now, we have something that looks like the same dataset, but it has 14, rather than 11 variables. And the 14th variable `v_CA16_4: 0 to 14 years` is the one that we really want. It's a `dbl` variable, which is a fancy, overly complicated way of saying that it's a number that can have decimal numbers.

Now we could map out where the toddlers in Kitchener live. But before we do that, we might want to rename that ugly variable name.

We have practiced using the `rename()` command. There are two ways to use it, often without pipes one with. Both of these below are equivalent.

#Check the original names

```
names(kitchener2)
```

```

## [1] "Shape Area"           "Type"
## [3] "Dwellings"            "Households"
## [5] "GeoUID"                "Population"
## [7] "CD_UID"                "CSD_UID"
## [9] "CT_UID"                "CMA_UID"
## [11] "geometry"              "Region Name"
## [13] "Area (sq km)"          "v_CA16_4: 0 to 14 years"

```

#Option 1

```
kitchener3<-rename(kitchener2, "Toddlers"=`v_CA16_4: 0 to 14 years`)
```

#Option 2 with pipes

```
kitchener3<-kitchener2 %>%

```

```
  rename("Toddlers"=`v_CA16_4: 0 to 14 years`)
```

#Option 2a with pipes

This is my favourite way because it is so logical

The saving part comes at the end

```
kitchener2 %>%
```

```
  rename("Toddlers"=`v_CA16_4: 0 to 14 years`)->kitchener3
```

#Check the new names

```
names(kitchener3)
```

```

## [1] "Shape Area"   "Type"        "Dwellings"    "Households"   "GeoUID"

```

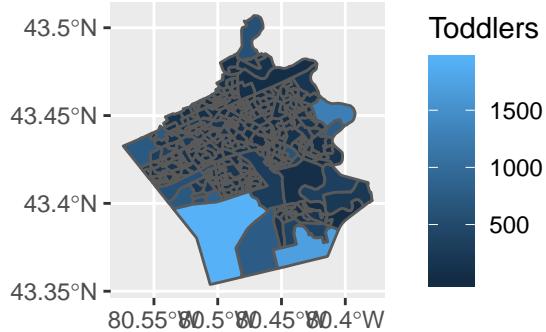
```

## [6] "Population"      "CD_UID"          "CSD_UID"          "CT_UID"          "CMA_UID"
## [11] "geometry"        "Region Name"     "Area (sq km)"    "Toddlers"

```

Then we can draw the map showing the number of toddlers in each dissemination area.

```
ggplot(kitchener3, aes(geometry=geometry, fill=Toddlers))+geom_sf()
```



Now, note that these are *raw* numbers. If you wanted to show the percent of the population to control for how many people live in each dissemination area, we would have to make a percentage variable. That is pretty easy and we have done it before in the dataset on whales earlier in the online exercises.

```
#check the names
names(kitchener3)
```

```

## [1] "Shape Area"      "Type"          "Dwellings"       "Households"     "GeoUID"
## [6] "Population"     "CD_UID"        "CSD_UID"        "CT_UID"        "CMA_UID"
## [11] "geometry"        "Region Name"   "Area (sq km)"  "Toddlers"

```

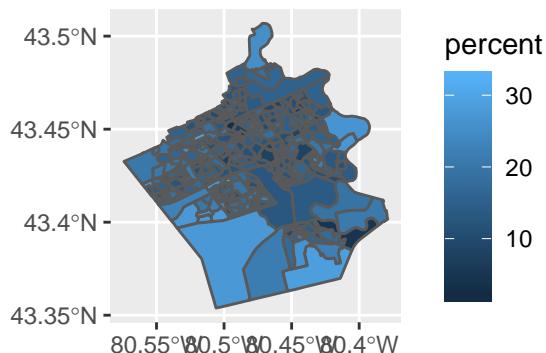
Because we already have the Population variable, we can just divide the number of Toddlers by the population and then we will get a percentage of the total population and multiply by 100

We use the `mutate()` command to make a new variable. The following two ways are *equivalent*

```
#Option 1 without pipes
kitchener4<-mutate(kitchener3, percent=(Toddlers/Population)*100)
#option 2 with pipes
kitchener4<-kitchener3 %>%
  mutate(percent=(Toddlers/Population*100))
#option 2a with pipes
kitchener3 %>%
  mutate(percent=(Toddlers/Population)*100)->kitchener4
```

Then we can redraw using the exact same code. Note that our `fill` variable is different now though, because we are not filling based on the number of Toddlers, but the percent variable that we created above.

```
ggplot(kitchener4, aes(geometry=geometry, fill=percent))+geom_sf()
```



4.7 How do we add in a third variable.

Let's just say that we want to break out the toddlers by sex. It's important to remember: Statistics Canada is not providing us with the individual level data from the census. They are aggregating it for us. What this means is that we cannot do a lot of crosstabulations by different variables.

For example, we could not show the distribution of toddlers of different ethnicities, for example unless Statistics Canada provides us with a table that has those two variables already combined.

The only variable that is almost always crosstabbed is sex. So we can always go back to the 0 to 4 years category and select the female and the male vectors.

The R code looks like this. Note this is the exact code to download the Kitchener data as above. I've just changed the vector numbers and saved this in `kitchener5`

```
kitchener5 <- get_census(dataset='CA16', regions=list(CSD="3530013"), vectors=c("v_CA16_9","v_CA16_8"))
```

Note, how we're getting two new vectors.

```
glimpse(kitchener5)
```

```
## Observations: 314
## Variables: 15
## $ `Shape Area`          <dbl> 0.78200, 0.33188, 0.56447, 0.18680, 0.1620...
## $ Type                  <fct> DA, DA, DA, DA, DA, DA, DA, DA, DA...
## $ Dwellings              <int> 230, 191, 254, 210, 199, 223, 306, 283, 18...
## $ Households             <int> 223, 168, 244, 210, 194, 217, 284, 268, 18...
## $ GeoUID                 <chr> "35300097", "35300098", "35300099", "35300...
## $ Population              <int> 588, 402, 552, 491, 425, 513, 584, 532, 45...
## $ CD_UID                 <chr> "3530", "3530", "3530", "3530", "3530", "3...
## $ CSD_UID                <chr> "3530013", "3530013", "3530013", "3530013"...
## $ CT_UID                 <chr> "5410024.00", "5410024.00", "5410023.00", ...
## $ CMA_UID                <chr> "35541", "35541", "35541", "35541", "35541...
## $ geometry                <MULTIPOLYGON [°]> MULTIPOLYGON (((-80.49342 4....
## $ `Region Name`           <fct> Kitchener, Kitchener, Kitchener, Kitchener...
## $ `Area (sq km)`          <dbl> 0.78200, 0.33188, 0.56447, 0.18680, 0.1620...
## $ `v_CA16_9: 0 to 4 years` <dbl> 10, 10, 20, 10, 10, 15, 20, 15, 10, 5, ...
## $ `v_CA16_8: 0 to 4 years` <dbl> 15, 10, 10, 15, 15, 15, 10, 15, 10, 10, 10...
```

Those variable names are ugly so we should rename them again. If we go back to the website, we know that vector `v_CA16_9` is females and vector `v_CA16_8` is males.

And remember, we can use the `rename` command based on position as well as by name. This would work on the `rename()` commands above, as well, you just need to identify what the variable names are that you need to rename.

```
#check names
names(kitchener5)
```

```
## [1] "Shape Area"          "Type"                  "Dwellings"
## [4] "Households"           "GeoUID"                 "Population"
## [7] "CD_UID"                "CSD_UID"                "CT_UID"
## [10] "CMA_UID"               "geometry"               "Region Name"
## [13] "Area (sq km)"          "v_CA16_9: 0 to 4 years" "v_CA16_8: 0 to 4 years"
```

We are going to change the names of variables 14 and 15.

```
#option 1
kitchener6<-rename(kitchener5, Females=14,Males=15 )
```

```

#option 2 with pipes
kitchener6<-kitchener5 %>%
  rename(Females=14, Males=15)

#option 2a
kitchener5 %>%
  rename(Females=14, Males=15)-> kitchener6

#rechecknames
names(kitchener6)

```

```

## [1] "Shape Area"      "Type"           "Dwellings"       "Households"     "GeoUID"
## [6] "Population"      "CD_UID"         "CSD_UID"        "CT_UID"        "CMA_UID"
## [11] "geometry"        "Region Name"    "Area (sq km)"  "Females"       "Males"

```

Now the othe problem is that this is not tidy data. There are two different y-variables (`Females` and `Males`.)

Using the `gather()` command, we say what is the new `key` variable going to be called (i.e. `Sex`, the new value variable `n` and which variables get gathered).

```

#option 1
kitchener7<-gather(kitchener6, Sex, n, Females, Males)

#Optino 2
# you can do it with pipes.
kitchener7<-kitchener6 %>%
  gather(Sex, n, Females, Males)

#option2a
kitchener6 %>%
  gather(Sex, n, Females, Males)-> kitchener7

```

```
glimpse(kitchener7)
```

```

## Observations: 628
## Variables: 15
## $ `Shape Area` <dbl> 0.78200, 0.33188, 0.56447, 0.18680, 0.16206, 0.15935...
## $ Type <fct> DA, ...
## $ Dwellings <int> 230, 191, 254, 210, 199, 223, 306, 283, 185, 360, 22...
## $ Households <int> 223, 168, 244, 210, 194, 217, 284, 268, 185, 357, 21...
## $ GeoUID <chr> "35300097", "35300098", "35300099", "35300100", "353...
## $ Population <int> 588, 402, 552, 491, 425, 513, 584, 532, 459, 530, 37...
## $ CD_UID <chr> "3530", "3530", "3530", "3530", "3530", "353...
## $ CSD_UID <chr> "3530013", "3530013", "3530013", "3530013", "3530013...
## $ CT_UID <chr> "5410024.00", "5410024.00", "5410023.00", "5410023.0...
## $ CMA_UID <chr> "35541", "35541", "35541", "35541", "35541", "35541"...
## $ geometry <MULTIPOLYGON [°]> MULTIPOLYGON ((((-80.49342 4..., MULTIPOLY...
## $ `Region Name` <fct> Kitchener, Kitchener, Kitchener, Kitchener, Kitchene...
## $ `Area (sq km)` <dbl> 0.78200, 0.33188, 0.56447, 0.18680, 0.16206, 0.15935...
## $ Sex <chr> "Females", "Females", "Females", "Females", "Females...
## $ n <dbl> 10, 10, 20, 10, 10, 15, 20, 15, 10, 5, 5, 10, 15...

```

Now we could draw the map and facet on sex. Note now we have named the variable that has the actual number of toddlers to be n.

```
ggplot(kitchener7, aes(geometry=geometry, fill=n))+  
  geom_sf()+

```

```
facet_wrap(~Sex)
```

