Mapping the Census using R

Carina Xue Luo, Geospatial Data Analyst
carina@uwindsor.ca
February 26, 2020
Collaboratory, Leddy Library, University of Windsor
Academic Data Centre:
Drop-In Hours: 12:30-4:30PM
By appointment: libdata@uwindsor.ca

This tutorial will explore how to incorporate and visualize census data in R. Specifically, you will learn how to obtain and clean census data, join census tables with census boundary files, and create a thematic map to identify patterns about the population.

Contents

Getting Started 2
Download and install R and RStudio 2
A brief introduction to the Canadian Census 2
How census is related to geography? 2
Which census geography do I need? 4
Downloading Census Attribute Data and Census Boundary Files 5
Loading Data into R 7
Joining the Two Files 9
Making a Thematic Map 10
References and Resources 16

Getting Started

Download and install R and RStudio

Download and install R

- Windows User: https://cran.r-project.org/bin/windows/base/
- Mac User: https://cran.r-project.org/bin/macosx/

Install RStudio Desktop: https://www.rstudio.com/products/rstudio/download/

A brief introduction to the Canadian Census

The Canadian Census is among one of the most important datasets providing information about Canadian population and housing units by their demographic, social and economic characteristics. It is a mandatory survey collected by Statistics Canada. From 1851-1951, the census was taken every ten years. From 1956 to the present, the census is taken every 5 years. For the censuses administered between 1971 and 2006, two forms were used to collect information from residents, the "short form" and the "long form". Most households (80%) received a short-form questionnaire asking a minimum number of questions (e.g., age, sex, marital status, household). A sample of households (20%) received a long-form questionnaire that included additional questions about language, ethnicity, immigration, employment, income etc[1]. In 2011, the long form census was replaced by a voluntary National Household Survey (NHS). The content of the NHS varied slightly from the previous long form censuses and therefore comparisons needs to be made with caution. The long-form census was reinstated by Statistics Canada in 2016. As such, a sample of approximately 25% of Canadian households received a long-form questionnaire and the other households received a short-form questionnaire.

How census is related to geography?

Census data are disseminated by a variety of geographic areas which are designed specifically for disseminating data. The geographic areas used for disseminating census data range in size from Canada, provinces and territories, all the way down to dissemination blocks. Standard geographic areas used for data dissemination and their relationships to one another are depicted in Figure 1. Statistics Canada releases boundary files for these standard geographic areas along with census variables. Each unit of these geographic areas has a unique identifier which makes it possible to locate them on the ground and relate census statistics to them (Figure 2).

[1] 1971 was an exception when the short form distributed to two-thirds of Canadian households and the long form was distributed to the remaining third.

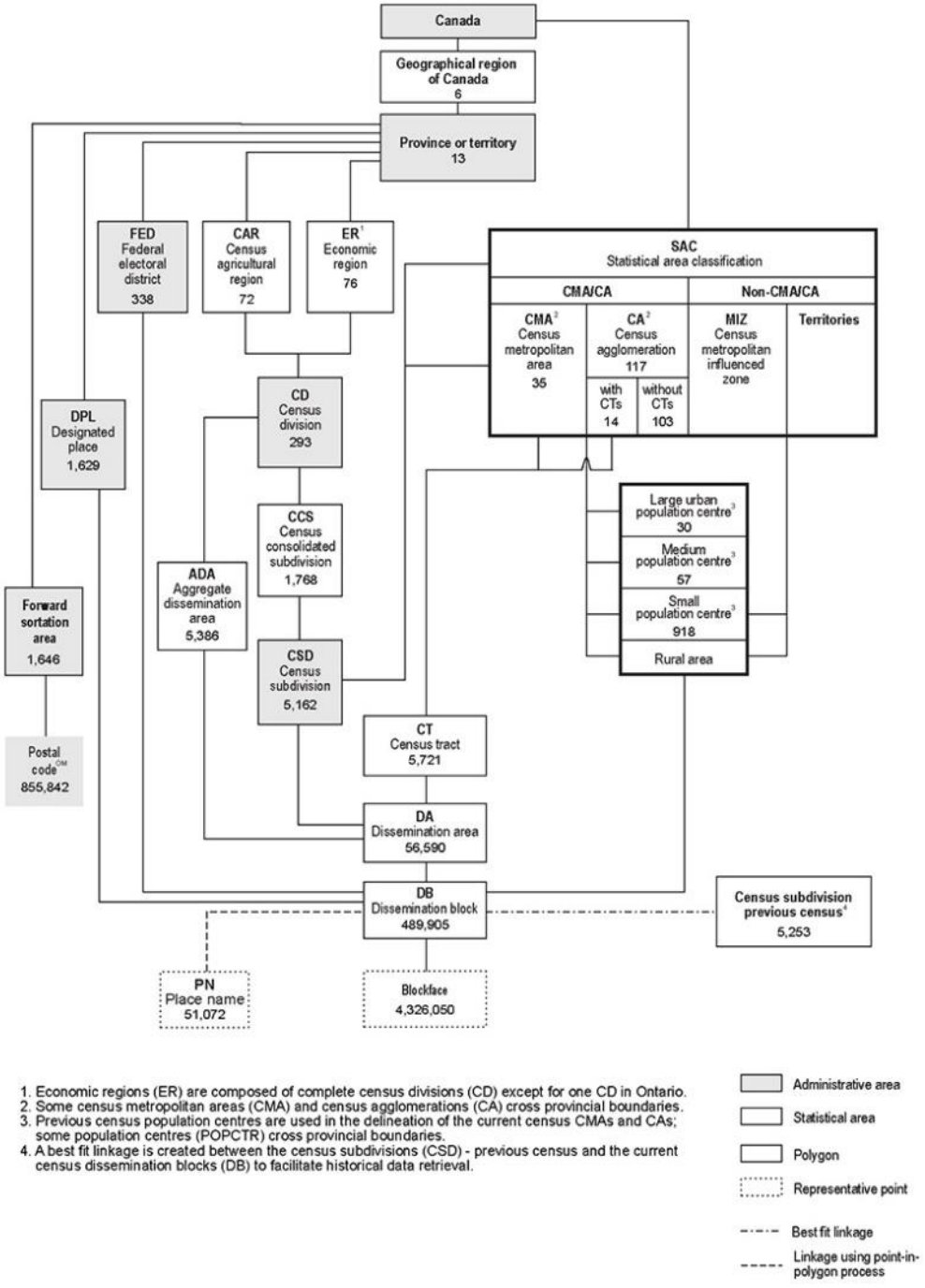


Figure 1-Standard geographic areas used for data dissemination and their relationships to one another. Source: https://www12.statcan.gc.ca/census-recensement/2016/ref/dict/figures/f1_1-eng.cfm

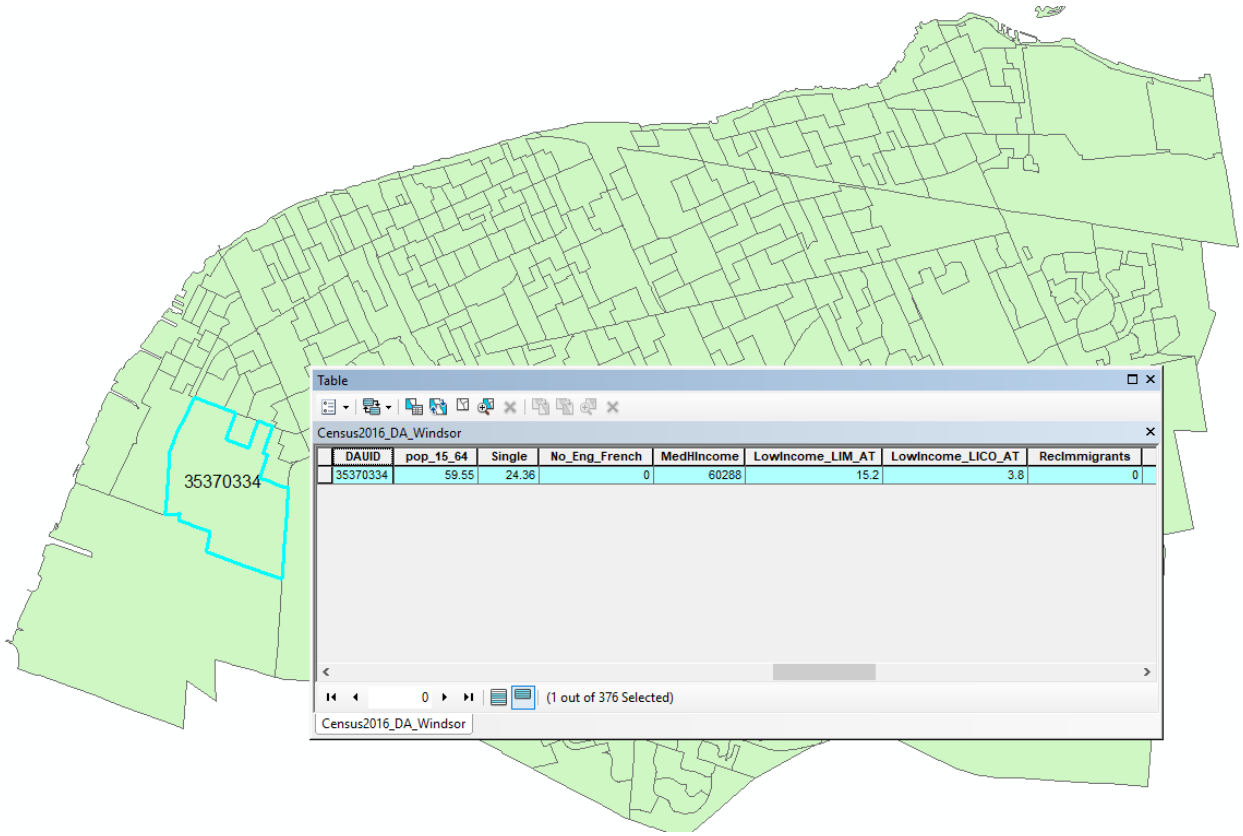


Figure 2-DA boundaries along with its census attribute table for City of Windsor, ON.

Which census geography do I need?

That depends entirely on the type of your research and what you want to look at. You may be looking for Canada as a whole, a province, a municipality (census subdivision) or a smaller geographic level. Or, you may want to create your own custom area. For example, you may want to know the number of people who worked outside Canada in the city of Windsor or you may want to compare income differences among neighbourhoods within the city. Or you may want to get a profile of the population that lives within a 10-minute drive of the university. Comparing neighbourhoods in a city or town usually involves looking at census tracts (CTs) or dissemination areas (DAs). Each CT usually has a population between 2,500 and 8,000 persons. They are located in census metropolitan areas (might not cover rural areas). Dissemination areas (DAs) fit within CTs and has an average population of 400 to 700 persons. DAs are the smallest standard geographic area for which all census data are released and cover all the territory of Canada. Figure 3 shows all the CTs and DAs in the Essex County.

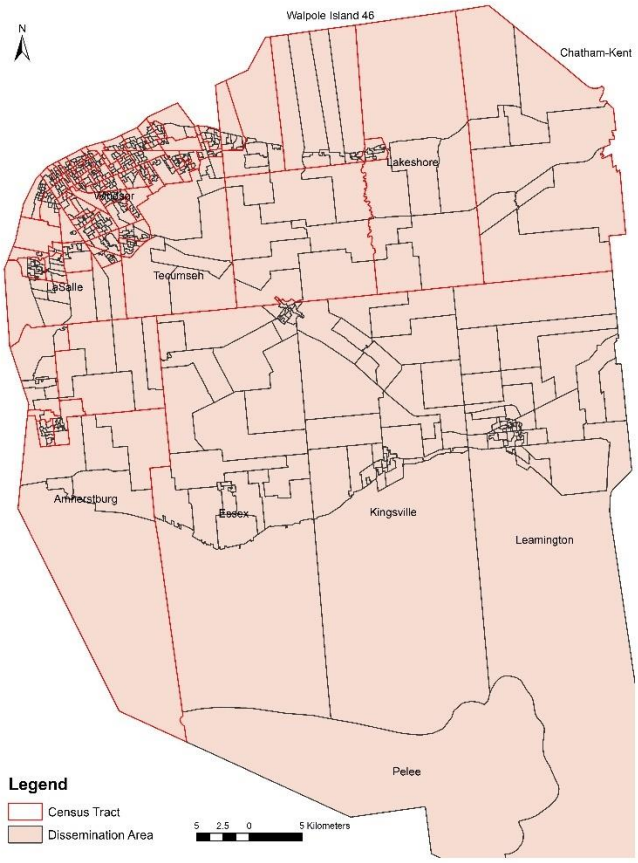


Figure 3- Geographic boundaries for CTs and DAs in Essex County.

In this tutorial, you will learn how to create a thematic map based on the DA-level census data using R - *an open source programming language and software environment for statistical computing and graphics*. You will need two types of files to complete the map: One file is a csv (comma-separated values) file which contains census variables at the DA level. Another file is a shapefiles containing DA's boundaries. You will start by downloading census attribute data (i.e., the 2016 population and dwelling count) and census boundary files from the Statistics Canada website. Next, you'll load these two files into R and join them together. Then, you'll make a map that shows Windsor's DAs` in different colors according to their population density.

Downloading Census Attribute Data and Census Boundary Files

1. Make a new folder on your hard drive (C drive or whatever drive you are working on) for this workshop (RGIS1). Folder path example: This PC>Local Disk (C:)> RGIS1
2. Go to https://www12.statcan.gc.ca/census-recensement/2016/dp-pd/hlt-fst/pd-pl/comprehensive.cfm and download the 2016 population and dwelling count at the DA levels (Figure 4). Save the data to the folder you just created on your hard drive.

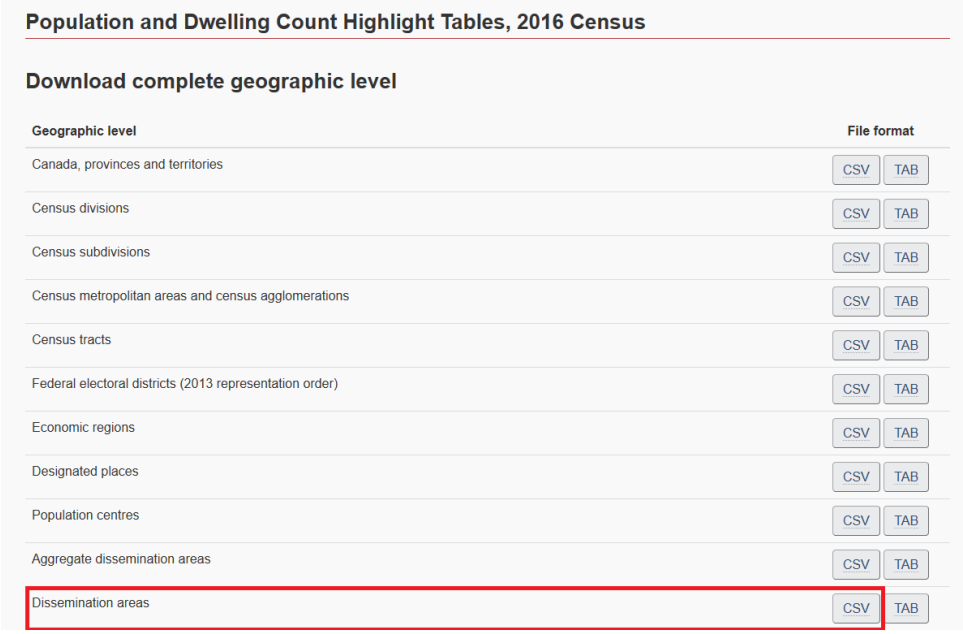


Figure 4-Screenshot of downloading population and dwelling count for DAs from 2016 Census

3. Go to http://www12.statcan.gc.ca/census-recensement/2011/geo/bound-limit/bound-limit-2016-eng.cfm and download the DA cartographic boundary shapefiles (Figure 5). It will be a .zip file (compressed files). Save the zipped file to your folder you just created, and right click on it and choose extract all. Please note that the boundary shapefiles contain four individual files including the .shp, .shx, .dbf, and .prj. A reference guide for using census boundary files can be accessed here: https://www150.statcan.gc.ca/n1/pub/92-160-g/92-160-g2016002-eng.pdf

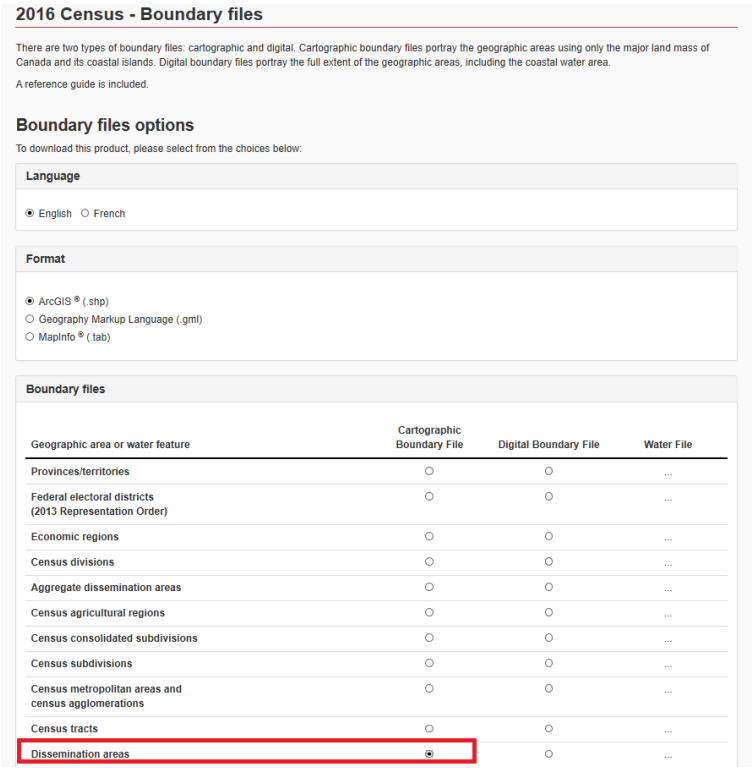


Figure 5- Screenshot of downloading DA cartographic boundary files for 2016 Census

Loading Data into R

First, set up the working directory. The working directory is where R finds all of its files for reading and writing on your computer.

```
# set up the working directory for R to read and write data on your computer
setwd("C:/RGIS1")
# check the working directory is correct
getwd()
```

Second, load the shapefiles and assign it to a new spatial data frame. We need the **rgdal** package installed to read the shapefiles. To use a package, you must first install it and then load it.

```
# Install the rgdal package
install.packages ("rgdal",dependencies = TRUE) #make sure the another package(s) it probably depends on to be also installed
# load the package "rgdal"
library(rgdal)
```

```
# load the shapefile data and assign it to a new spatial object called "DA"
DA <- readOGR(dsn = "lda_000b16a_e", layer = "lda_000b16a_e")
```

Note: *If reading a shapefile, the data source name (dsn= argument) is the folder (directory) where the shapefile is, and the layer is the name of the shapefile (without the .shp extension).*
To get information about a specific function, you can use ?function name or help('function name').

To take a peek at the data, using the **summary ()** function:

```
# get the quick summary of the data
summary (DA)
```

Note: *Spatial objects like the SpatialPolygonsDataFrame are made up of a number of different slots, the key slots being @data (non-spatial attribute data) and @polygons (or @lines for line data). The data slot can be thought of as an attribute table and the geometry slot is the polygons that make up the physical boundaries. Specific slots are accessed using the @ symbol.*

Let's now analyse the "DA" object with some basic commands:

```
# display the attribute table of the data as a spreadsheet
View(DA@data)
```

This will open up a separate window that displays a spreadsheet like this:

Filter

	DAUID	PRUID	PRNAME	CDUID	CDNAME	CDTYPE	CCSUID
0	10010244	10	Newfoundland and Labrador / Terre-Neuve-et-Labrador	1001	Division No. 1	CDR	1001519
1	10010245	10	Newfoundland and Labrador / Terre-Neuve-et-Labrador	1001	Division No. 1	CDR	1001519
2	10010246	10	Newfoundland and Labrador / Terre-Neuve-et-Labrador	1001	Division No. 1	CDR	1001519
3	10010247	10	Newfoundland and Labrador / Terre-Neuve-et-Labrador	1001	Division No. 1	CDR	1001519
4	10010248	10	Newfoundland and Labrador / Terre-Neuve-et-Labrador	1001	Division No. 1	CDR	1001519
5	10010249	10	Newfoundland and Labrador / Terre-Neuve-et-Labrador	1001	Division No. 1	CDR	1001519
6	10010250	10	Newfoundland and Labrador / Terre-Neuve-et-Labrador	1001	Division No. 1	CDR	1001519
7	10010251	10	Newfoundland and Labrador / Terre-Neuve-et-Labrador	1001	Division No. 1	CDR	1001519
8	10010252	10	Newfoundland and Labrador / Terre-Neuve-et-Labrador	1001	Division No. 1	CDR	1001519
9	10010253	10	Newfoundland and Labrador / Terre-Neuve-et-Labrador	1001	Division No. 1	CDR	1001519
10	10010254	10	Newfoundland and Labrador / Terre-Neuve-et-Labrador	1001	Division No. 1	CDR	1001519
11	10010255	10	Newfoundland and Labrador / Terre-Neuve-et-Labrador	1001	Division No. 1	CDR	1001519

Notice that each row in this data represents one DA, and each column represents the attributes associated with that DA.

```
# check the classes (data type) of all the variables (attributes) in a spatial data frame
sapply(DA@data, class)
```

Note: *The function* ***sapply()*** *is used to apply a Function over a list or vector.*

The result will look like this:

```
    DAUID     PRUID    PRNAME     CDUID    CDNAME    CDTYPE    CCSUID  CCSNAME    CSDUID  CSDNAME  CSDTYPE     ERUID    ERNAME  SACCODE
 "factor"  "factor"  "factor"  "factor"  "factor"  "factor"  "factor"  "factor"  "factor"  "factor"  "factor"  "factor"  "factor"  "factor"
  SACTYPE    CMAUID  CMAPUID  CMANAME  CMATYPE     CTUID    CTNAME    ADAUID
 "factor"  "factor"  "factor"  "factor"  "factor"  "factor"  "factor"  "factor"
```

Notice that all the variables are factors. This is because when you assign a shapefiles to a (spatial) data frame, data frame automatically converts character vectors (string variables) into factors. This might create problems when you join census data (which has the population and dwelling count) to the spatial data frame using "DAUID" as a key – i.e., you will get a warning message saying that "joining factors with different levels, coercing to character vector". It is recommended to convert factor "DAUID" back to its original character format using the following command:

```
# covert factor "DAUID" to the character format:
DA$DAUID <- as.character (DA$DAUID)
```

Note: *More information about factor in R can be found:*
https://www.stat.berkeley.edu/~s133/factors.html

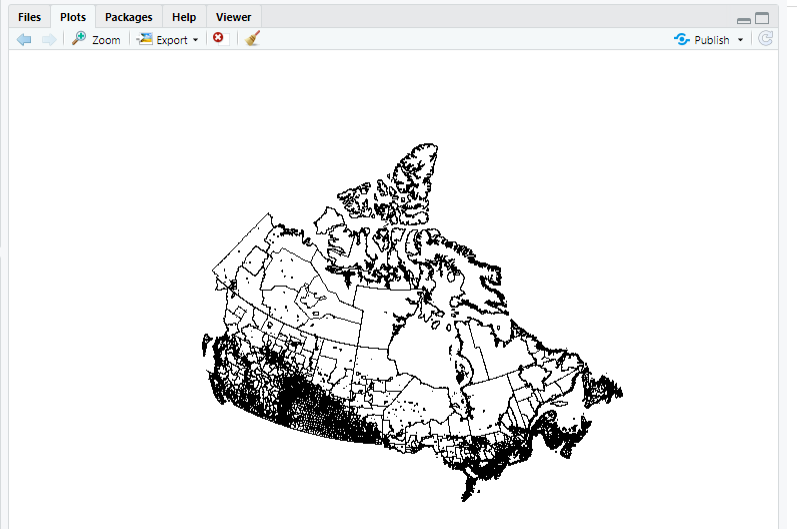
Alternatively, you can reimport shapefiles and pass the argument stringsAsFactors = FALSE:

```
DA <- readOGR(dsn = "lda_000b16a_e", layer = "lda_000b16a_e", stringsAsFactors = FALSE)
```

Now, let's make a simple map by using the **plot ()** function:

```
plot(DA)
```

The below is what appears on the R plotting window:



You can see that this boundary file contains all the DAs in Canada, but the map we are going to make is focused on the City of Windsor. So next you are going to create a subset of this data that only contains Windsor. The most basic way of subsetting a data frame in R is by using **square brackets** such that in

```
example [x,y]
```

Note*: example is the data frame we want to subset, 'x' consists of the rows we want returned, and 'y' consists of the columns we want returned.*

Now, suppose you only need the DAs (rows) that within the city of Windsor (whose CSDUID is "***3537039***") and the variable (the column named "DAUID") that has the unique identifier for each DA. Here is the basic way to retrieve that data in R:

```
DA_Windsor <- DA[DA$CSDUID=="3537039", "DAUID"]
```

Note: *You can find the standard geographical areas codes (unique identifier) for different levels of census geographic areas from this webpage: https://www12.statcan.gc.ca/census-recensement/2011/geo/map-carte/ref/sgc-cgt/index-eng.cfm*

Again, get a summary of the data and make a simple map:

```
summary(DA_Windsor)
plot(DA_Windsor)
```

Now, you have successfully loaded and extracted the geographic boundaries for DAs that are within Windsor. Next, you'll load the .csv file which has DA-level population and dwelling count and assign it to a new data frame.

```
# load the csv file and assign it to a new data frame
Census2016 <- read.csv("T1901EN.csv", header = TRUE, stringsAsFactors = FALSE)
```

Note*: The argument "header = TRUE" holds if the file contains the names of the variables as its first line. The argument "StringsAsFactors = FALSE" prevents character vectors from being converted to factors.*

Again, take an overview of the data:

```
# get a summary of the data
str(Census2016)
# take a look at the data as a spreadsheet
View(Census2016)
```

Take a look at the columns. Some of them are incomplete (i.e., column named "X") and some are not necessary for our further analysis (i.e., columns in French). And the column names have punctuation marks and spaces that might become problematic when you want to perform some R functions later on. So, you'll need to do some data cleaning first. For this mapping purpose, we are interested in the following demographic variables: population, total private dwellings, and population density along with the geographic code (with which can be linked to the above spatial data frame to make a map). Therefore, you'll create a subset of the "Census2016" data that only contains the desired columns.

```
Sub_census <- Census2016[,c(1,7,9,12)]
str(Sub_census)
```

Then, to fix the column names containing punctuation and spaces, just rewrite the column names:

```
names(Sub_census)<-c("DAUID","POP","Dwelling","PopDen")
names(Sub_census)
```

Joining the Two Files

Now you have loaded and cleaned the DA-level census attribute data and assigned it to a data frame ("Sub_census"), the next step is to join it to the spatial data frame ("DA_Windsor") to make a map. Remember that the spatial data frame now only has one attribute variable ("DAUID"). What you are going to do next is to add more variables from the census data ("Sub_census") to the data table of the spatial data frame ("DA_Windsor@data").

```
View(DA_Windsor@data)
View(Sub_census)
```

In order to join the two data together, they need to have a common variable that share the same information. In this example, both files have the unique geographic identifier for each DA ("DAUID"). First, make sure the data type of the two columns are the same:

```
class(DA_Windsor$DAUID)
class(Sub_census$DAUID)
```

Second, perform a quick check to see if values in the "DAUID" of "Windsor_DA" are also contained in the "DAUID" of the "Sub_census".

```
DA_Windsor$DAUID %in% Sub_census$DAUID
```

***Note**: %in% is used for value matching and "returns a vector of the positions of (first) matches of its first argument in its second" (See help('%in%')).*

```
> DA_Windsor$DAUID %in% Sub_census$DAUID
  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [26] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [51] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [76] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[101] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[126] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[151] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[176] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[201] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[226] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[251] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[276] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[301] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[326] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[351] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[376] TRUE
```

The result indicates that all the values in the "DAUID" columns from the two data match. Now you are ready the two data together.

First, you need to install and load the "dplyr" package.

```
# Install and load the package "dplyr"
install.packages("dplyr",dependencies = TRUE)
library(dplyr)
```

Then, use the left_join () function:

```
# join variables from the census data to the data slot of the spatial data frame
DA_Windsor@data <- left_join(DA_Windsor@data, Sub_census, by = "DAUID")
str(DA_Windsor@data)
```

***Note**: left_join is chosen here because we want the length of the spatial data frame ("DA_Windsor") to remain unchanged, with variables from the census data ("Sub_census") appended in new columns. Keep spatial data table (DA_Windsor@data) in first position.*

Take a look at the new DA_Windsor@data object. You should see new variables added, meaning the join was successful.

```
> str(DA_Windsor@data)
'data.frame':   376 obs. of  4 variables:
 $ DAUID   : chr  "35370078" "35370102" "35370103" "35370168" ...
 $ POP     : int  503 428 546 374 597 428 156 596 1478 326 ...
 $ Dwelling: int  248 220 212 238 518 178 78 201 401 156 ...
 $ PopDen  : num  4256 3859 5778 5305 5209 ...
```

There is one more thing we should do before making a map – checking the coordinate reference systems (CRS) of the data and make sure it is appropriate for the specific area you are interested in. CRS is a reference system used to represent the locations of geographic features within a common

framework. There are many different types of CRS (see the notes below), and each one of necessity distorts the representation of the earth in some way. Therefore, it is important to select a projection having the properties that are suited to the mapping situation.

Note: *There are mainly two types of CRS: Geographic coordinate systems (GCS) and Projected coordinate systems (PCS). GCS uses a three-dimensional spherical surface to define locations on the earth and records a location by a coordinate pair (latitude and longitude) in degrees and a reference datum. Since GCS locates latitude and longitude location using angles, the spacing of each line of latitude moving north and south is not uniform. Thus, GCS are not ideal for measuring distance. PCS is defined on a flat, two-dimensional surface. This involves a mathematical transformation (so called "map projection") that projects data from its three-dimensional curved surface onto a two-dimensional flat surface, which results in distortion in the shape, area, distance, or direction of the data. Different projections cause different types of distortions. Some projections are designed to minimize the distortion of one or two of the data's characteristics. Some projections can be used for a map of the whole world (at small scales), while others are appropriate for small areas only (at large scales). A PCS is always based on a GCS that is based on a sphere or spheroid. In addition to the GCS, a PCS includes a map projection, a set of projection parameters that customize the map projection for a particular location, and a linear unit of measure. It is important to understand the CRS that your data uses - particularly if you are working with different data stored in different CRS. In R, you need to transform data with different CRS to a common CRS so that they can align with one another. This is similar to making sure that their units are the same when measuring distance or area.*
More information about the coordinate reference system can be found here:
http://resources.esri.com/help/9.3/arcgisengine/dotnet/89b720a5-7339-44b0-8b58-0f5bf2843393.htm
https://mdl.library.utoronto.ca/technology/tutorials/selecting-right-projection

In this example, the CRS for the geographic boundaries downloaded from the Statistics Canada website is Lambert conformal conic projection, which is widely used for general maps of Canada at small scales. However, for mapping the City of Windsor, it is recommended to use the NAD 83 UTM Zone 17N – a CRS that maintains shape for small areas in North America between 84°W and 78°W. So, let's first retrieve the CRS information from the data using the **st_crs()** function from the "sf" package.

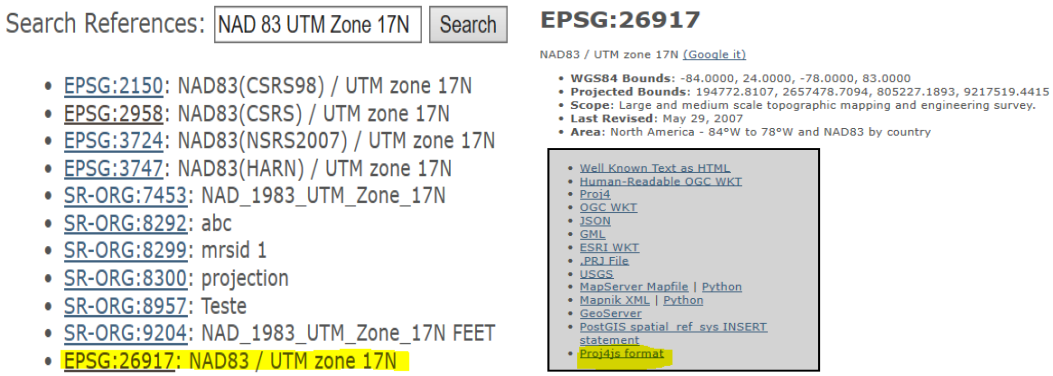
```
install.packages("sf",dependencies = TRUE)
library("sf")
st_crs(DA_Windsor)
```

The result will look like this:

```
> st_crs(DA_Windsor)
Coordinate Reference System:
  No EPSG code
  proj4string: "+proj=lcc +lat_1=49 +lat_2=77 +lat_0=63.390675 +lon_0=-91.86666666666666 +x_0=6200000 +y_0=3000000 +ellps=GRS80 +towgs84=
0,0,0,0,0,0,0 +units=m +no_defs"
```

In R, there are two ways of defining a coordinate system: via the EPSG numeric code (which stands for European Petroleum Survey Group) or via the PROJ.4 notation. The EPSG code may not always be available for a particular coordinate system, but if a spatial object has a defined coordinate system, it will always have a PROJ.4 notation. You can obtain the PROJ.4 notation for a particular CRS on this website: http://spatialreference.org/ref/

For example, here is the search result on http://spatialreference.org/ref/ for the CRS "NAD 83 UTM Zone 17N". Note that after clicking the EPSG:26917 link, you must then select the Proj4 syntax from a list of available syntaxes to view the projection parameters.



Now, you can transform the CRS of data to the desired NAD 83 UTM Zone 17N by using the **spTransform()** function from the rgdal package.

```
# Transform the CRS to NAD_1983_UTM_Zone_17N
newcrs <- CRS("+proj=utm +zone=17 +ellps=GRS80 +datum=NAD83 +units=m +no_defs ")
DA_windsor <- spTransform(DA_windsor, newcrs)
```

Congratulations! Now you are ready to make a thematic map for the Windsor's neighbourhoods (DAs) by their census attributes.

Making a Thematic Map

A thematic map (also called a choropleth map) is a map in which areas are distinctly colored or shaded in proportion to a statistical value that represents an aggregate summary of a characteristic within each area (e.g., population density, average income). It is good at utilizing data to represent spatial variation of the desired measurement across a region.
Next, you are going to create a thematic map showing Windsor's DAs in different colors according to their population density in 2016 (see the example map below).



The important steps involve in creating a thematic map in R include: 1) choose the variable you want to map; 2) decide the number of classes that the variable will be categorized into; 3) decide the classification method for grouping the variable into separate classes (e.g., equal interval, quantile) ; 4) assign colors to classes. In order to do so, first you need to install and load the two packages: "RColorBrewer" and "classInt".

```
# Install and load mutiple packages
install.packages(c("RColorBrewer","classInt"),dependencies = TRUE)
lapply(c("RColorBrewer","classInt"), require, character.only = TRUE)
```

Second, create two temporary variables to store the variable you want to map and the number of classes (colors):

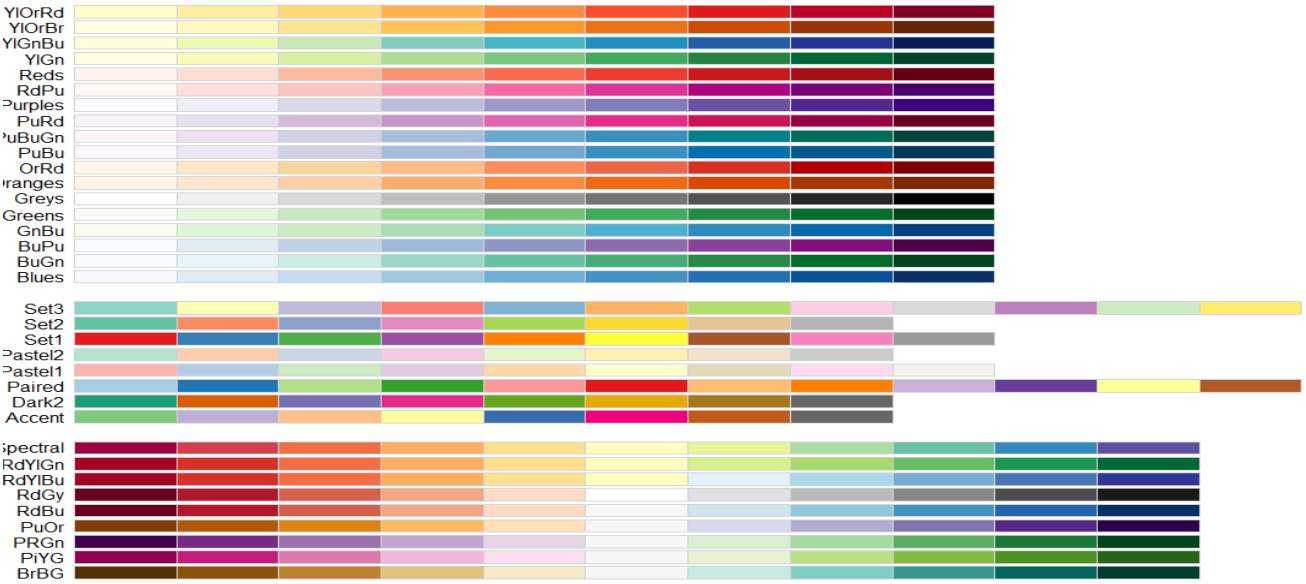
```
# the variable you want to map
plotvar <- DA_Windsor$PopDen
# the number of classes that the variable will be categorized into
nclr <- 5
```

Note: *When you have more classes, you get more variation but sometimes making it harder to separate shading. It is recommended to limit the number of classes to, for example six or less.*

Third, create a custom color palette for your map using the RColorBrewer library:

```
# display a list of all the color palettes available
display.brewer.all()
```

A plot showing all the color palettes will show up in the plotting window:



Once you've picked a palette you like, create a palette object as follows, where you specify the number of classes you want to use (i.e., "nclr"), and the name of the color ramp. Note that different palettes have different limits on the maximum and minimum number of cuts allowed.

```
plotclr <- brewer.pal(nclr, name = "YlOrRd")
```

Then you will use **classIntervals()** to obtain break points for each class and assign the individual observations (DAs) to one of the classes. You need to specify the classification method through the "style=" argument. In this example, you'll choose quantiles in which each class contains an equal number of DAs.

```
q5 <- classIntervals(plotvar, nclr, style="quantile")
```

You can take a look at the break points for each class:

```
> q5
style: quantile
   [28.3,1908.4)  [1908.4,2492.5)  [2492.5,3235.5)  [3235.5,4553.2)
              75               75               76               74
[4553.2,84313.7]
              76
```

And then, you can use the **findColours()** function to assign colours to each DA which also returns two attributes which can be used in constructing a legend later.

To obtain the color codes for each class:

```
q5Colours <- findColours(q5,plotclr)
```

Take a look at the data structure of the output object:

```
> str(q5Colours)
 chr [1:376] "#F03B20" "#F03B20" "#BD0026" "#BD0026" "#BD0026" "#FECC5C" ...
 - attr(*, "palette")= chr [1:5] "#FFFFB2" "#FECC5C" "#FD8D3C" "#F03B20" ...
 - attr(*, "table")= 'table' int [1:5(1d)] 75 75 76 74 76
  ..- attr(*, "dimnames")=List of 1
  .. ..$ : chr [1:5] "[28.3,1908.4)" "[1908.4,2492.5)" "[2492.5,3235.5)" "[3235.5,455
3.2)" ...
```

It is a character vector of colours with two attributes: "palette", a character vector of colours corresponding to the specified breaks, and "table", a named frequency table. These two attributes will be used to create a legend in the map.

Finally, you can just pass these color codes to **plot()**:

```
# set the margin of your plot in inches by using mar = c (bottom, left, top, right)
par(mar = c(5, 1, 1, 1))
plot(DA_Windsor, col = q5Colours, border=NA) #remove the borders since looks prettier
# add a map title
title(main = "City of Windsor, 2016", sub = "By Census Dissemination Area")
```

Note: *You can adjust the size of the plot margins by specifying a margin parameter using the syntax par(mar = c(bottom, left, top, right)), where the arguments bottom, left … are the size (in inches) of the margins. The default value for mar is c(5.1, 4.1, 4.1, 2.1).*

Now you have made a map that shows the spatial variation in population density among Windsor's DAs. Next, you are going to add a legend, scale bar and north arrow to the map.

First, create a custom legend for your map by using the **legend()** function. Sometimes, you have missing values in your dataset which are represented as "NA" in R. We don't have any in this particular dataset, but if we did, we would have seen they were left as the background color of the map. Therefore, it's important to include "NA" value in the legend. Since the default background color of all plots in R is white, you'll add the color "white" to the colors of symbols (Use **colors ()** to see all 657 built in color names in R) and add "No data" to the legend text.

```
# create a custom legend for your map with dealing with "NA" values
legend(x = "bottomright", # legend position
       legend = c("No Data", names(attr(q5Colours, "table"))), # text indicates what the different symbols correspond to
       fill = c("white", attr(q5Colours,"palette")), #  colors of symbol
       bty = "n", #showing no box around the legend
       title="Population density per square kilometer",
       cex = 0.8) #adjust legend size
```

- x: the position of the legend. The possible values can be "topright", "topleft", "bottomright", "bottomleft".
- legend: a vector of text indicates what the different symbols correspond to;
- fill: colors of symbols (boxes);
- bty indicates the type of box to be drawn around the legend. The allowed values are "o" (the default) and "n" (no box).
- title: the title of legend;
- cex: the character expansion factor, whose default value is 1. To reduce the legend size, use a cex value of less than 1; to increase the legend size, use a cex value greater than 1.

Finally, you'll add a north arrow and a scale bar to the map by using the "prettymapr" library.

```
# add a north arrow to the map
install.packages("prettymapr",dependencies = TRUE)
library(prettymapr)
addnortharrow(pos = "topright", padin = c(0.15, 0.15), scale = 0.5,
              lwd = 0.5, border = "black", cols = c("white", "black"),
              text.col = "black")
# add a scale bar to the map
addscalebar(plotunit = "m", plotepsg = 26917, widthhint = 0.25,
            unitcategory = "metric", htin = 0.05, padin = c(0.15, 0.15),
            style = "bar", bar.cols = c("black", "white"), lwd = 1,
            linecol = "black", tick.cex = 0.7, labelpadin = 0.08, label.cex = 0.8,
            label.col = "black", pos = "bottomleft")
```

Note: *You can find more information about the prettymapr library here: https://cran.r-project.org/web/packages/prettymapr/prettymapr.pdf*

References and Resources

Statistics Canada. History of the Census of Canada. 2015.
Statistics Canada. Timeline 100 years of the Canadian Census. 2018.
Statistics Canada. Census of Population. 2017.