.**Introduction**

Now that we are to start working with Canadian Census data, let’s first briefly address the question why you may need to use it. After all, [CANSIM data](https://dataenthusiast.ca/?p=127) is often more up-to-date and covers a *much* broader range of topics than the national census data, which is gathered every five years in respect of a limited number of questions.

The main reason is that CANSIM data is far less granular geographically. Most of it is collected at the provincial or even higher regional level. You may be able to find CANSIM data on a limited number of questions for some of the country’s largest metropolitan areas, but if you need the data for a specific census division, city, town, or village, you’ll have to use the Census.

To illustrate the use of **cancensus** package, First, in this post we’ll retrieve these key labor force characteristics of the largest metropolitan areas in each of the [five geographic regions of Canada](https://dataenthusiast.ca/?p=766#annex-notes):

* Labor force participation rate, employment rate, and unemployment rate.
* Percent of workers by work situation: full time vs part time, by gender.
* Education levels of people aged 25 to 64, by gender.

The cities (metropolitan areas) that we are going to look at, are: Calgary, Halifax, Toronto, Vancouver, and Whitehorse. We’ll also get these data for Canada as a whole for comparison and to illustrate the retrieval of data at different geographic levels

Next, in the upcoming Part 6 of the “Working with Statistics Canada Data in R” series, we will visualize these data, including making a faceted plot and writing a function to automate repetitive plotting tasks.

Keep in mind that **cancensus** also allows you to retrieve *geospatial* data, that is, borders of census regions at various geographic levels, in **sp** and **sf** formats. Retrieving and visualizing Statistics Canada geospatial data will be covered later in these series.

So, let’s get started by loading the required packages:

library(cancensus)

library(tidyverse)

**Searching for Data**

**cancensus** retrieves census data with the get\_census function. get\_census can take a number of arguments, the most important of which are dataset, regions, and vectors, which have no defaults. Thus, in order to be able to retrieve census data, you’ll first need to figure out:

* your dataset,
* your region(s), and
* your data vector(s).

**Find Census Datasets**

Let’s see which census datasets are available through the [CensusMapper API](https://censusmapper.ca/api):

list\_census\_datasets()

Currently, datasets earlier than 2001 are not available, so if you need to work with the 20th century census data, you won’t be able to retrieve it with **cancensus**.

**Find Census Regions**

Next, let’s find the regions that we’ll be getting the data for. To search for census regions, use the search\_census\_regions function.

Let’s take a look at what region search returns for Toronto. Note that **cancensus** functions return their output as dataframes, so it is easy to subset. Here I limited the output to the most relevant columns to make sure it fits on screen. You can run the code without [c(1:5, 8)] to see all of it.

# all census levels

search\_census\_regions(searchterm = "Toronto",

dataset = "CA16")[c(1:5, 8)]

Returns:

A tibble: 3 x 6

# region name level pop municipal\_status PR\_UID

1 35535 Toronto CMA 5928040 B 35

2 3520 Toronto CD 2731571 CDR 35

3 3520005 Toronto CSD 2731571 C 35

You may have expected to get only one region: the city of Toronto, but instead you got three! So, what is the difference? Look at the column ‘level’ for the answer. Often, the same geographic region can be represented by several census levels, as is the case here. There are three levels for Toronto, which is simultaneously a census metropolitan area, a census division, and a census sub-division. Note also the ‘PR\_UID’ column that contains numeric codes for Canada’s provinces and territories, which can help you distinguish between different census regions that have the same or similar names. For an example, run the code above replacing “Toronto” with “Windsor”.

Remember that we were going to plot the data for census metropolitan areas? You can choose the geographic level with the level argument, which can take the following values: ‘C’ for Canada (national level), ‘PR’ for province, ‘CMA’ for census metropolitan area, ‘CD’ for census division, ‘CSD’ for census sub-division, or NA:

# specific census level

search\_census\_regions("Toronto", "CA16", level = "CMA")

Let’s now list census regions that may be relevant for our project:

# explore available census regions

names <- c("Canada", "Calgary", "Halifax",

"Toronto", "Vancouver", "Whitehorse")

map\_df(names, ~ search\_census\_regions(., dataset = "CA16"))

purrr::map\_df function applies search\_census\_regions iteratively to each element of the names vector and returns output as a single dataframe. Note also the ~ . syntax. Think of it as the tilde taking names and passing it as an argument to a place indicated by the dot in the search\_census\_regions function. It may be a good idea to read the whole **purrr** is a super-useful package, but not the easiest to learn, and this tutorial does a great job explaining the basics.

So as you can see, there are multiple entries for each search term, so we’ll need to choose the results for census metropolitan areas, and for census sub-division in case of Whitehorse, since Whitehorse is too small to be considered a census metropolitan area:

# select only the regions we need: CMAs (and CSD for Whitehorse)

regions <- list\_census\_regions(dataset = "CA16") %>%

filter(grepl("Calgary|Halifax|Toronto|Vancouver", name) &

grepl("CMA", level) |

grepl("Canada|Whitehorse$", name)) %>%

as\_census\_region\_list()

Pay attention to the use of logical operators to filter the output by several conditions at once; also note using $ regex meta-character to choose the entry *ending* with ‘Whitehorse’ from the ‘names’ column (to filter out ‘Whitehorse, Unorganized’.

Finally, as\_census\_region\_list converts list\_census\_regions output to a data object of type list that can be passed to the get\_census function as its regions argument.

**Find Census Vectors**

Canadian census data is made up of individual variables, aka *census vectors*. Vector number(s) is another argument you need to specify in order to retrieve data with the get\_census function.

**cancensus** has two functions that allow you to search through census data variables: list\_census\_vectors and search\_census\_vectors.

list\_census\_vectors returns *all* available vectors for a given dataset as a single dataframe containing vectors and their descriptions:

# structure of list\_census\_vectors output

str(list\_census\_vectors(dataset = 'CA16'))

# count variables in 'CA16' dataset

nrow(list\_census\_vectors(dataset = 'CA16'))

As you can see, there are 6623 (as of the time of writing this) variables in the 2016 census dataset, so list\_census\_vectors won’t be the most convenient function to find a specific vector. Note however that there are situations, in which list\_census\_vectors would be appropriate.

Usually it is more convenient to use search\_census\_vectors to search for vectors. Just pass the text string of what you are looking for as the searchterm argument.

Let’s now find census data vectors for labor force involvement rates:

# get census data vectors for labor force involvement rates

lf\_vectors <-

search\_census\_vectors(searchterm = "employment rate",

dataset = "CA16") %>%

union(search\_census\_vectors("participation rate", "CA16")) %>%

filter(type == "Total") %>%

pull(vector)

Let’s take a look at what this code does. Since searchterm doesn’t have to be a precise match, “employment rate” search term retrieves unemployment rate vectors too. In the next line, union merges dataframes returned by search\_census\_vectors into a single dataframe. Note that in this case union could be substituted with bind\_rows. I recommend using union in order to avoid data duplication. Next, we choose only the “Total” numbers, since we are not going to plot labor force indicators by gender. Finally, the pull command extracts a single vector from the dataframe, just like the $ subsetting operator: we need ‘lf\_vectors’ to be a data object of type vector in order to pass it to the vectors argument of the get\_census function.

The second labor force indicator we are looking for, is the number of people who work full-time and part-time, broken down by gender. But before we proceed with getting the respective vectors, let me show you another way to figure out search terms to put inside the search\_census\_vectors function: use Statistics Canada online [Census Profile tool](https://www12.statcan.gc.ca/census-recensement/2016/dp-pd/prof/index.cfm). It can be used to quickly explore census data as well as to figure out variable names (search terms) and their hierarchical structure.

For example, let’s look at [census labor data for Calgary metropolitan area](https://www12.statcan.gc.ca/census-recensement/2016/dp-pd/prof/details/page.cfm?Lang=E&Geo1=CMACA&Code1=825&Geo2=PR&Code2=48&SearchText=Calgary&SearchType=Begins&SearchPR=01&B1=Labour&TABID=1&type=0). Scrolling down, you will quickly find the numbers and text labels for full-time and part-time workers:

Now we know the exact search terms, so we can get precisely the vectors we need, free from any extraneous data:

# get census data vectors for full and part time work

# get vectors and labels

work\_vectors\_labels <-

search\_census\_vectors("full year, full time", "CA16") %>%

union(search\_census\_vectors("part year and/or part time", "CA16")) %>%

filter(type != "Total") %>%

select(1:3) %>%

mutate(label = str\_remove(label, ".\*, |.\*and/or ")) %>%

mutate(type = fct\_drop(type)) %>%

setNames(c("vector", "gender", "type"))

# extract vectors

work\_vectors <- work\_vectors\_labels$vector

Note how this code differs from the code with which we extracted labor force involvement rates: since we need the data to be sub-divided both by the type of work *and* by gender (hence no “Total” values here), we create a dataframe that assigns respective labels to each vector number. This work\_vectors\_labels dataframe will supply categorical labels to be attached to the data retrieved with get\_census.

Also, note these three lines:

mutate(label = str\_remove(label, ".\*, |.\*and/or ")) %>%

mutate(type = fct\_drop(type)) %>%

setNames(c("vector", "gender", "type"))

The first mutate call removes all text up to and including ‘, ‘ and ‘and/or ‘ (spaces included) from the ‘label’ column. The second drops unused factor level “Total” – it is a good practice to make sure there are no unused factor levels if you are going to use **ggplot2** to plot your data. Finally, setNames renames variables for convenience.

Finally, let’s retrieve vectors for the education data for the age group from 25 to 64 years, by gender. Before we do this, I’d like to draw your attention to the fact that some of the census data is hierarchical, which means that some variables (census vectors) are included into parent and/or include child variables. It is very important to choose vectors at proper hierarchical levels so that you do not double-count or omit your data.

Education data is a good example of hierarchical data. So, let’s now retrieve and label the education data vectors:

# get census vectors for education levels data

# get vectors and labels

ed\_vectors\_labels <-

search\_census\_vectors("certificate", "CA16") %>%

union(search\_census\_vectors("degree", "CA16")) %>%

union(search\_census\_vectors("doctorate", "CA16")) %>%

filter(type != "Total") %>%

filter(grepl("25 to 64 years", details)) %>%

slice(-1,-2,-7,-8,-11:-14,-19,-20,-23:-28) %>%

select(1:3) %>%

mutate(label =

str\_remove\_all(label,

" cert.\*diploma| dipl.\*cate|, CEGEP| level|")) %>%

mutate(label =

str\_replace\_all(label,

c("No.\*" = "None",

"Secondary.\*" = "High school or equivalent",

"other non-university" = "equivalent",

"University above" = "Cert. or dipl. above",

"medicine.\*" = "health\*\*",

".\*doctorate$" = "Doctorate\*"))) %>%

mutate(type = fct\_drop(type)) %>%

setNames(c("vector", "gender", "level"))

# extract vectors

ed\_vectors <- ed\_vectors\_labels$vector

Note the slice function that allows to manually select specific rows from a dataframe: positive numbers choose rows to keep, negative numbers choose rows to drop. I used slice to drop the hierarchical levels from the data that are either too generalized or too granular. Note also that I had to edit text strings in the data. Finally, I added asterisks after “Doctorate” and “health”. These are not regex symbols, but actual asterisks that will be used to refer to footnotes in plot captions later on.

Now that we have figured out our dataset, regions, and data vectors (and labelled the vectors, too), we are finally ready to retrieve the data itself.

**Retrieve Census Data**

To retrieve census data, feed the dataset, regions, and data vectors into get\_census as its’ respective arguments. Note also that get\_census has use\_cache argument (set to TRUE by default), which tells get\_census to retrieve data from cache if available. If there is no cached data, the function will query CensusMapper API for the data and will save it in cache, while use\_cache = FALSE will force get\_census to query the API and update the cache.

# get census data for labor force involvement rates

# feed regions and vectors into get\_census()

labor <-

get\_census(dataset = "CA16",

regions = regions,

vectors = lf\_vectors) %>%

select(-c(1, 2, 4:7)) %>%

setNames(c("region", "employment rate",

"unemployment rate",

"participation rate")) %>%

mutate(region = str\_remove(region, " (.\*)")) %>%

pivot\_longer("employment rate":"participation rate",

names\_to = "indicator",

values\_to = "rate") %>%

mutate\_if(is.character, as\_factor)

The select call drops columns with irrelevant data, setNames renames columns to remove vector numbers from variable names, which will be then converted to values in the ‘indicator’ column; str\_remove inside the mutate call drops municipal status codes ‘(B)’ and ‘(CY)’ from region names; finally, mutate\_if converts characters to factors for subsequent plotting.

An important function here is tidyr::pivot\_longer. It converts the dataframe from wide to long format. It takes three columns: ‘employment rate’, ‘unemployment rate’, and ‘participation rate’, and converts their names into values of the ‘indicator’ variable, while their numeric values are passed to the ‘rate’ variable. The reason for the conversion is that we are going to plot the data for all three labor force indicators in the same graphic, which makes it necessary to store the indicators as a single factor variable.

Next, let’s retrieve census data about the percent of full time vs part time workers, by gender, and the data about the education levels of people aged 25 to 64, by gender:

# get census data for full time and part time work

work <-

get\_census(dataset = "CA16",

regions = regions,

vectors = work\_vectors) %>%

select(-c(1, 2, 4:7)) %>%

rename(region = "Region Name") %>%

pivot\_longer(2:5, names\_to = "vector",

values\_to = "count") %>%

mutate(region = str\_remove(region, " (.\*)")) %>%

mutate(vector = str\_remove(vector, ":.\*")) %>%

left\_join(work\_vectors\_labels, by = "vector") %>%

mutate(gender = str\_to\_lower(gender)) %>%

mutate\_if(is.character, as\_factor)

# get census data for education levels

education <-

get\_census(dataset = "CA16",

regions = regions,

vectors = ed\_vectors) %>%

select(-c(1, 2, 4:7)) %>%

rename(region = "Region Name") %>%

pivot\_longer(2:21, names\_to = "vector",

values\_to = "count") %>%

mutate(region = str\_remove(region, " (.\*)")) %>%

mutate(vector = str\_remove(vector, ":.\*")) %>%

left\_join(ed\_vectors\_labels, by = "vector") %>%

mutate\_if(is.character, as\_factor)

Note one important difference from the code I used to retrieve the labor force involvement data: here I added the dplyr::left\_join function that joins labels to the census data.

We now have the data and are ready to visualize it, which will be done in the next post.

**Annex: Notes and Definitions**

For those of you who are outside of Canada, Canada’s *geographic* regions and their largest metropolitan areas are:

* The Atlantic Provinces – Halifax
* Central Canada – Toronto
* The Prairie Provinces – Calgary
* The West Coast – Vancouver
* The Northern Territories – Whitehorse

These regions should not be confused with 10 provinces and 3 territories, which are Canada’s sub-national *administrative* divisions, much like states in the U.S. Each region consists of several provinces or territories, except the West Coast, which includes only one province – British Columbia.