Mapping American Community Survey with R

Lee Hachadoorian

August 14, 2017

Introduction

The American Community Survey (ACS) annually collects and disseminates a wealth of demographic and economic data, including race, sex, age, occupation, household income, commuting behavior, disability status, etc. The US Census Bureau exposes ACS data via API and the R packages `tidycensus` automates the downloading and joining of Census geographies and demographic data. This workshop will introduce the users to the data available in ACS, the use of R and RStudio, the `tidycensus` package for data downloading, and the `tmap` pakcage for thematic mapping of selected ACS variables. In addition, the workshop will introduce some basic functionalities of the tidyverse (including `dplyr` and `ggplot2` packages for data wrangling and visualization).

Prerequisites

- Familiarity with a programming language.
- Familiarity with basic principles of thematic mapping.
- A laptop with the following software installed:
  - R 3.3.0+
  - RStudio
  - The R packages `tidycensus`, `tmap`, `tidyverse`, `acs`, and dependencies
  - An activated Census API key

This workshop presumes no previous knowledge of R, RStudio, or American Community Survey or other Census data products. The workshop will probably be somewhat difficult to follow for someone who does not have a basic, scripting-level understanding of working with a programming language, or for someone who has never made a thematic map.

Instructions on installing the software and obtaining a Census API key have been provided separately.

Overture

Let's see where we're going.

The `tidycensus` package has two main functions: `get_acs` for getting American Community Survey data, and `get_decennial` for getting decennial census data. This workshop will only look at `get_acs`, but `get_decennial` has similar syntax.

First, lets load one variable, `B19003_001` or Median Household Income:

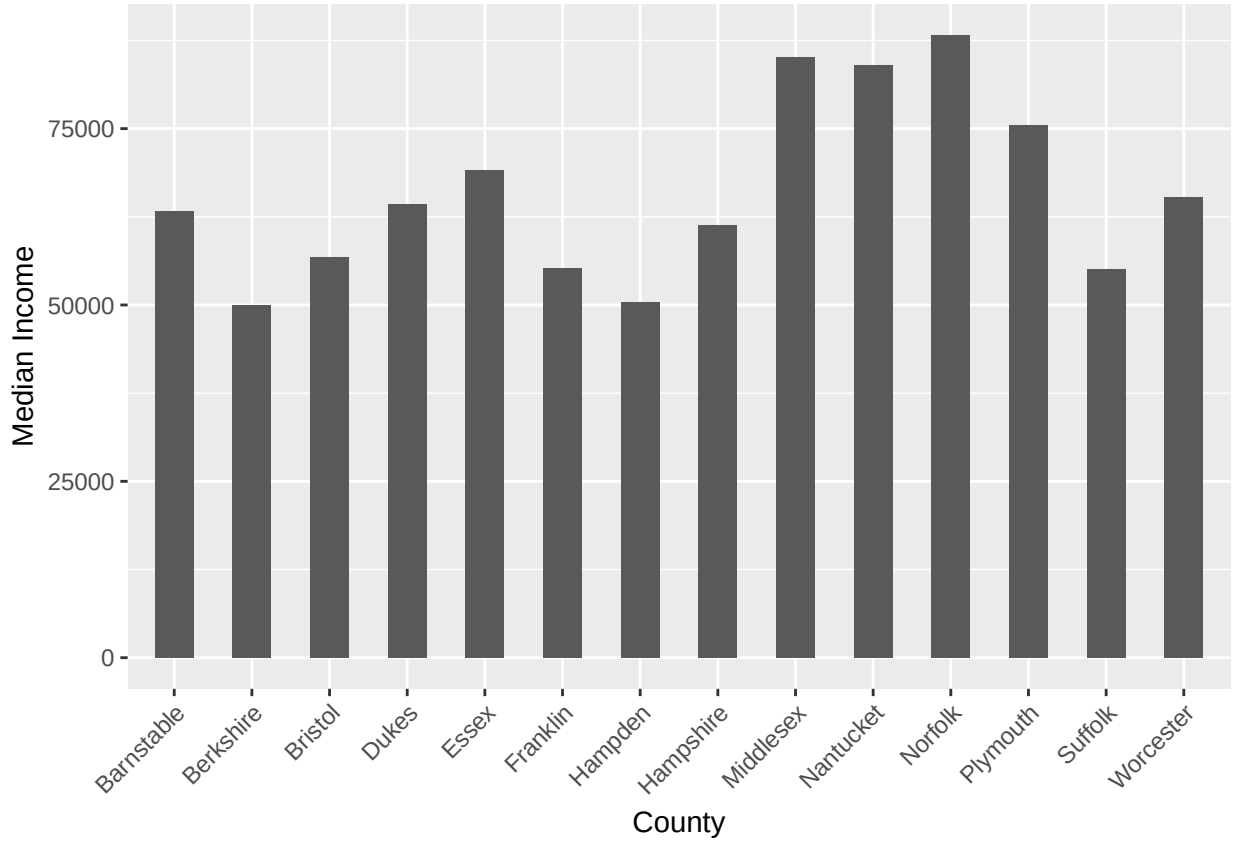
```
# Load necessary packages, set global options
library(tidycensus)
library(tidyverse)
library(tmap)
library(stringr)
options(tigris_use_cache = TRUE)

# Load the data
```

```
mass_counties = get_acs(
  geography = "county",
  variables = "B19013_001",
  endyear = 2015,
  output = "wide",
  state = "MA",
  geometry = TRUE
)
mass_counties$NAME = str_replace(
  mass_counties$NAME, " County, Massachusetts", ""
)
```

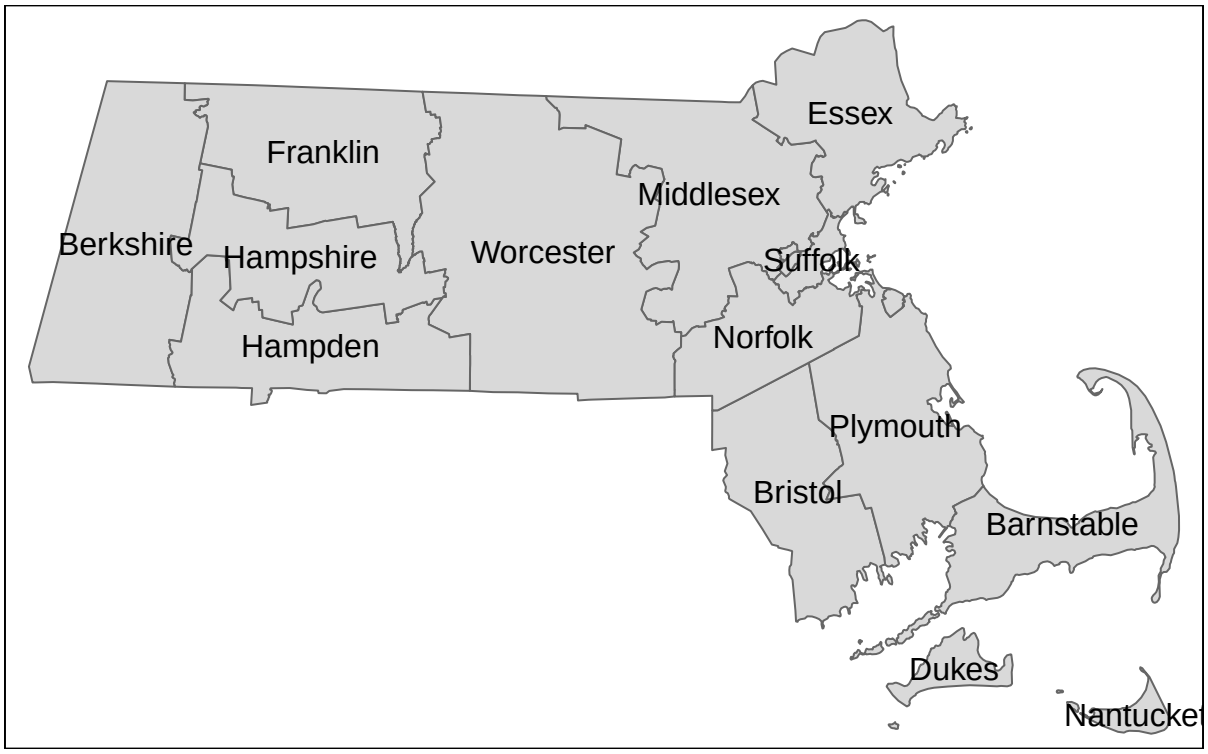
Now let's make a very basic graph using `ggplot`, a tidyverse package for data visualization:

```
ggplot(mass_counties, aes(x = NAME, y = B19013_001E)) +
  geom_col(width = 0.5) +
  xlab("County") + ylab("Median Income") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



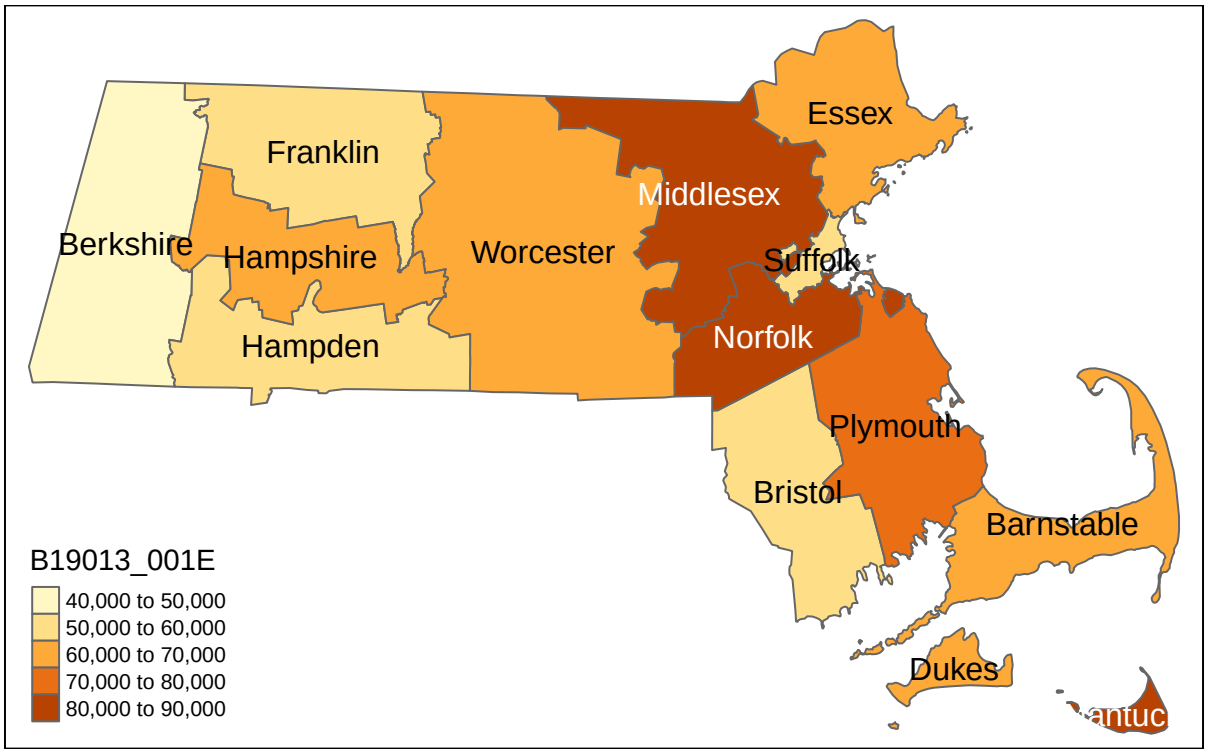
Let's see some basic maps that can be made using the tmap package. In each case, we are allowing tmap to apply some default styling. First, we use the `tm_polygons` function to just show the geometries, and `tm_text` to add labels:

```
tm_shape(mass_counties) +
  tm_polygons() +
  tm_text("NAME")
```



Next, pass a variable name to the first (color) parameter of `tm_polygons`. Tmap realizes that if you want to color an area based on a variable value, you must want a choropleth map:

```
tm_shape(mass_counties) +
  tm_polygons("B19013_001E") +
  tm_text("NAME")
```

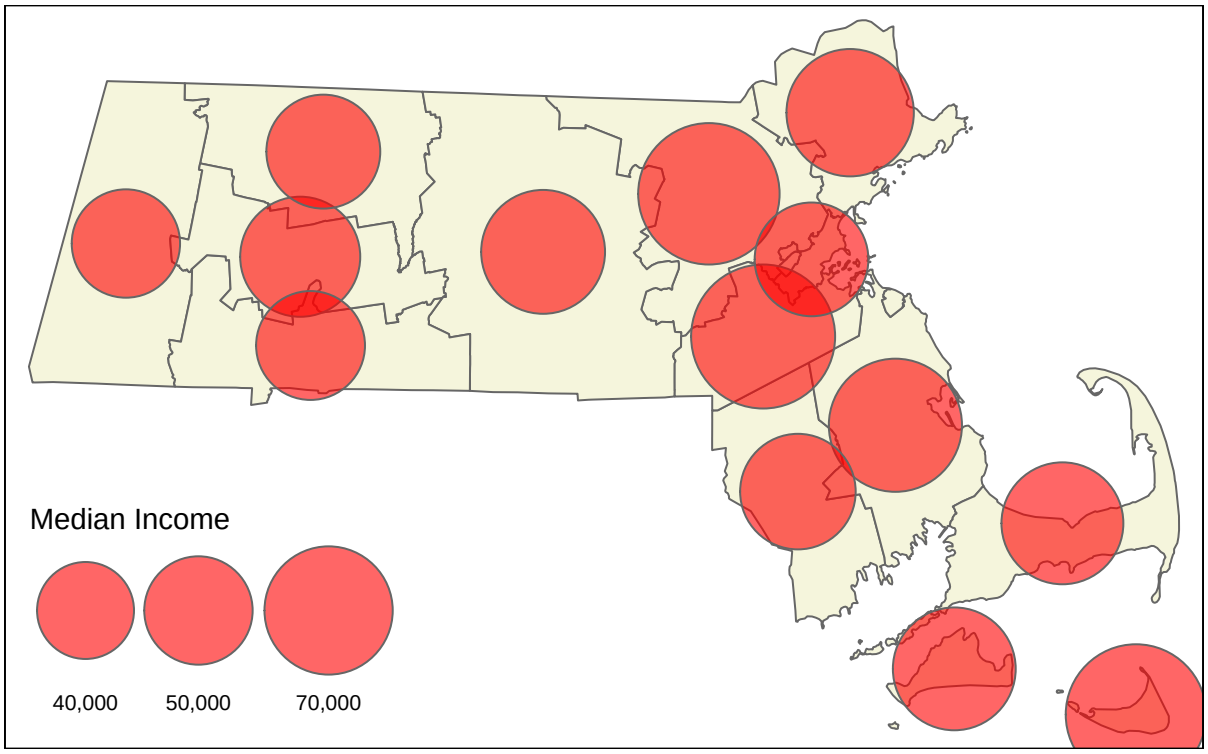


Tmap will automatically make label text light on a dark background and dark on a light background. As you can see, this may not always be desirable. Label color can be controlled with the `col` parameter, or the background color can be adjusted.

Finally, pass the same variable to the first (size) parameter of `tm_bubbles` in order to get a bubble map (proportional symbol map for GIS purists). Note that we are now passing a specific color to `tm_polygons` to control the background color:

```
tm_shape(mass_counties) +
  tm_polygons("beige") +
  tm_bubbles("B19013_001E", scale = 5, alpha = 0.6,
             col = "red", title.size = "Median Income")
```

```
## Warning: The legend is too narrow to place all symbol sizes.
```



The function `get_acs` builds an API call, sends it to USCB servers, and directly ingests the data. If you set the parameter `geometry = TRUE`, as we have done, it will also download a matching set of shapefiles. These geography files are *not* exposed via API, so a zip archive will be downloaded to your working directory, then loaded into R as a spatial object.

Getting ACS Data

Finding variables

In the opening example, you were given the ACS variable for median household income `B19013_001`. The variable name is not intuitive. Where did it come from? Every variable is a combination of a subject table and a "line number", which we more naturally refer to as a variable. The official list can be downloaded from the Census. The list for 2015 is ftp://ftp2.census.gov/programs-surveys/acs/summary_file/2015/documentation/user_tools/ACS_5yr_Seq_Table_Number_Lookup.txt. Download it and open it with a spreadsheet viewer.

Search for `B19013`. This subject table is a little unusual, in that the table has only a single variable (or "CELL"). This is because it is a summary statistic, representing a single value which applies over the relevant universe, which in this case is households. Most subject tables instead contain counts, and are broken down into hierarchically nested count variables. Consider table `B01001`, "Sex by Age", which is near the top of the list. This table has the structure:

- Total
  - Male
    - Under 5 years
    - 5 to 9 years

    * [. . .]
    * 85 years and over
- Female
    * Under 5 years
    * 5 to 9 years
    * [. . .]
    * 85 years and over

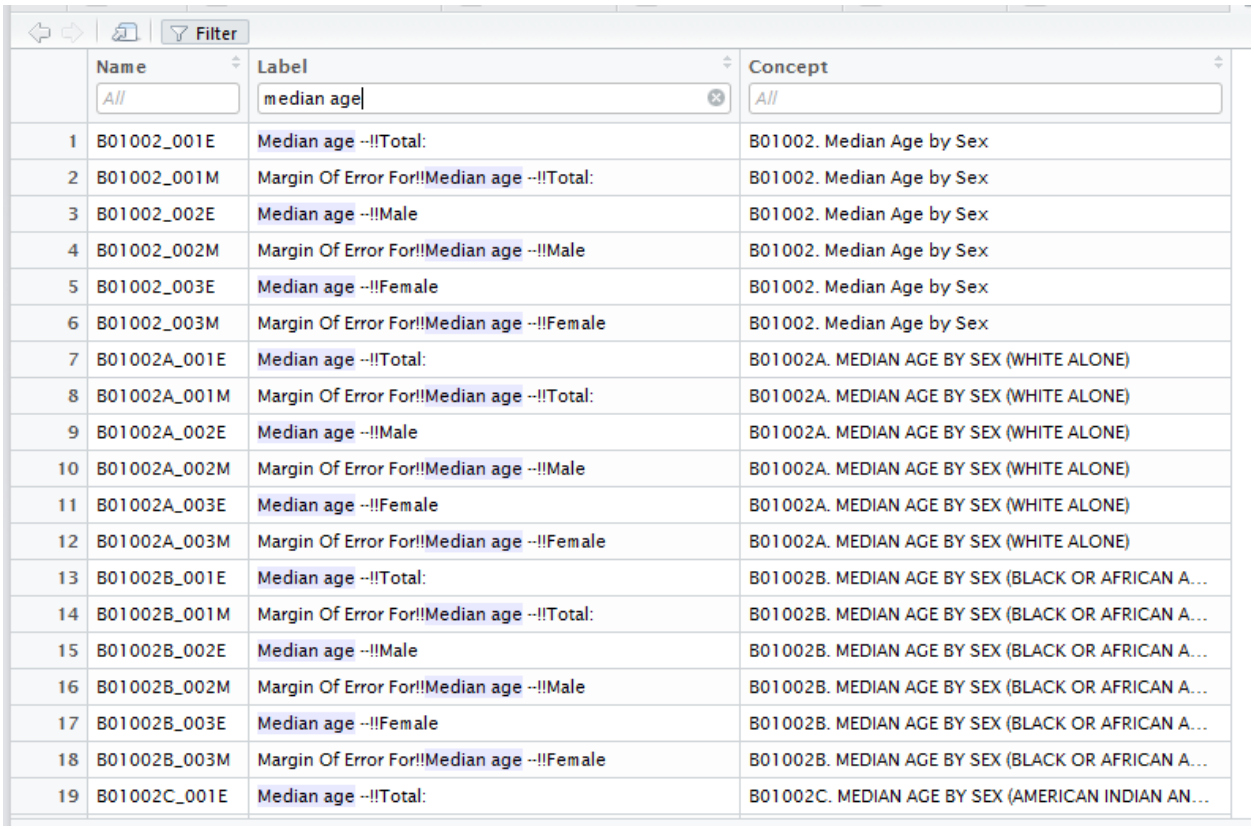
In this table "Total" is the total count (and since the universe is total population, this means all persons). The next indentation level has two variables, "Male" and "Female". These two variables add up to "Total". The next indentation level is the various age buckets. All of the variables under the variable "Male" will add up to the total number of males. If you want to know the total number of persons under 5, you would have to add `B01001_003` (males under 5) and `B01001_027` (females under 5).

Most ACS subject tables are structured like this, and have the total count of the relevant universe (persons, households, families, etc.) in the first variable.

`tidycensus` also provides a built-in way to access this data using the

```
vars2015 = load_variables(2015, "acs5", cache = TRUE)
```

The data frame can be viewed in a spreadsheet-like grid in RStudio by clicking its name in the Environment pane. This pane has a filter box. Experiment with searching for particular variables of interest.



Filter

	Name	Label	Concept
	All	median age	All
1	B01002_001E	Median age --!!Total:	B01002. Median Age by Sex
2	B01002_001M	Margin Of Error For!!Median age --!!Total:	B01002. Median Age by Sex
3	B01002_002E	Median age --!!Male	B01002. Median Age by Sex
4	B01002_002M	Margin Of Error For!!Median age --!!Male	B01002. Median Age by Sex
5	B01002_003E	Median age --!!Female	B01002. Median Age by Sex
6	B01002_003M	Margin Of Error For!!Median age --!!Female	B01002. Median Age by Sex
7	B01002A_001E	Median age --!!Total:	B01002A. MEDIAN AGE BY SEX (WHITE ALONE)
8	B01002A_001M	Margin Of Error For!!Median age --!!Total:	B01002A. MEDIAN AGE BY SEX (WHITE ALONE)
9	B01002A_002E	Median age --!!Male	B01002A. MEDIAN AGE BY SEX (WHITE ALONE)
10	B01002A_002M	Margin Of Error For!!Median age --!!Male	B01002A. MEDIAN AGE BY SEX (WHITE ALONE)
11	B01002A_003E	Median age --!!Female	B01002A. MEDIAN AGE BY SEX (WHITE ALONE)
12	B01002A_003M	Margin Of Error For!!Median age --!!Female	B01002A. MEDIAN AGE BY SEX (WHITE ALONE)
13	B01002B_001E	Median age --!!Total:	B01002B. MEDIAN AGE BY SEX (BLACK OR AFRICAN A...
14	B01002B_001M	Margin Of Error For!!Median age --!!Total:	B01002B. MEDIAN AGE BY SEX (BLACK OR AFRICAN A...
15	B01002B_002E	Median age --!!Male	B01002B. MEDIAN AGE BY SEX (BLACK OR AFRICAN A...
16	B01002B_002M	Margin Of Error For!!Median age --!!Male	B01002B. MEDIAN AGE BY SEX (BLACK OR AFRICAN A...
17	B01002B_003E	Median age --!!Female	B01002B. MEDIAN AGE BY SEX (BLACK OR AFRICAN A...
18	B01002B_003M	Margin Of Error For!!Median age --!!Female	B01002B. MEDIAN AGE BY SEX (BLACK OR AFRICAN A...
19	B01002C_001E	Median age --!!Total:	B01002C. MEDIAN AGE BY SEX (AMERICAN INDIAN AN...

ACS is so vast (over 20,000 variables) that it can be difficult to browse if you don't know what you are looking for. An excellent website for exploring the various subjects is https://censusreporter.org/. For example, to find subject tables related to commuting, click the Commute link on the home page. From there you can drill down on subject tables. Look at Table B08301: Means of Transportation to Work. Indentation of the variable names also makes clear which variables are folded into higher-level variables. Unfortunately, it does not display the variable number, so you will have to count or refer back to the sequence spreadsheet or

`load_variables` data frame.

Where to the subject table names come from? Subject table names are formatted as `[B|C]ssnnn[A-I][PR]`. Let's unpack this, from the inside out.

`ssnnn` refers to a two-digit general topic area, such as "Commute" or "Income", followed a three-digit index. A partial list of topics is

```
01 = Age and Sex
02 = Race
03 = Hispanic or Latino Origin
05 = Foreign Born; Citizenship; Year of Entry; Nativity
07 = Residence 1 Year Ago; Migration
15 = Educational Attainment
17 = Poverty
18 = Disability
19 = Income (Households and Families)
23 = Employment Status; Work Experience; Labor Force
```

This root is followed by nothing, or by a letter from `A` to `I`. This letter, if present, indicates a race/ethnicity iteration. The table is present multiple times, but the universe in each table is restricted to a particular racial or ethnic group:

- `A` = White alone
- `B` = Black or African American Alone
- `C` = American Indian and Alaska Native Alone
- `D` = Asian Alone
- `E` = Native Hawaiian and Other Pacific Islander Alone
- `F` = Some Other Race Alone
- `G` = Two or More Races
- `H` = White Alone, Not Hispanic or Latino
- `I` = Hispanic or Latino

In order to avoid disclosure of private information, the number of person in each group must meet a minimum threshold. In the race/ethnicity iteration tables, this may be done by **collapsing** some of the variable buckets into larger buckets. This is what the `B` or `C` at the front of a table name means. `B` is a "base" table, while `C` is a collapsed table. For example, table `B15002` shows educational attainment for all persons. Tables beginning with `C15002` show educational attainment for race/ethnic groups:

`B15002`	`C15002[A-I]`
No schooling	Less than high school diploma
Nursery to 4th grade	High school grad, GED, or alt.
5th and 6th grade	Some college or associate's
7th and 8th grade	Bachelor's degree or higher
9th grade	
etc...	

Finally, a table name ending in `PR` includes data for Puerto Rico only. Puerto Rico **is** represented in most ACS tables, but sometimes it is broken out into a separate table with slightly different variables. In this case, the table with same base name does **not** include any data for Puerto Rico.

Loading Bucketed Data

Most ACS data represent **counts** presented in various "buckets". Let's load some commute data. The data can be loaded in a "tidy" or "wide" data frame. Most GIS professionals will be more familiar with the wide

format, where each row represents a unique location—a point location or a geographic unit—and each column represents an attribute associated with that location.

The tidy format is a form of entity-attribute-value (EAV) data representation. Each row represents *one* attribute (or variable) and associated value of a geography (location). Because of this, the geography will be repeated multiple times in the table. It is not very useful for traditional desktop GIS, but can be useful for some R functions.

Let's load the commuting data from table B08301:

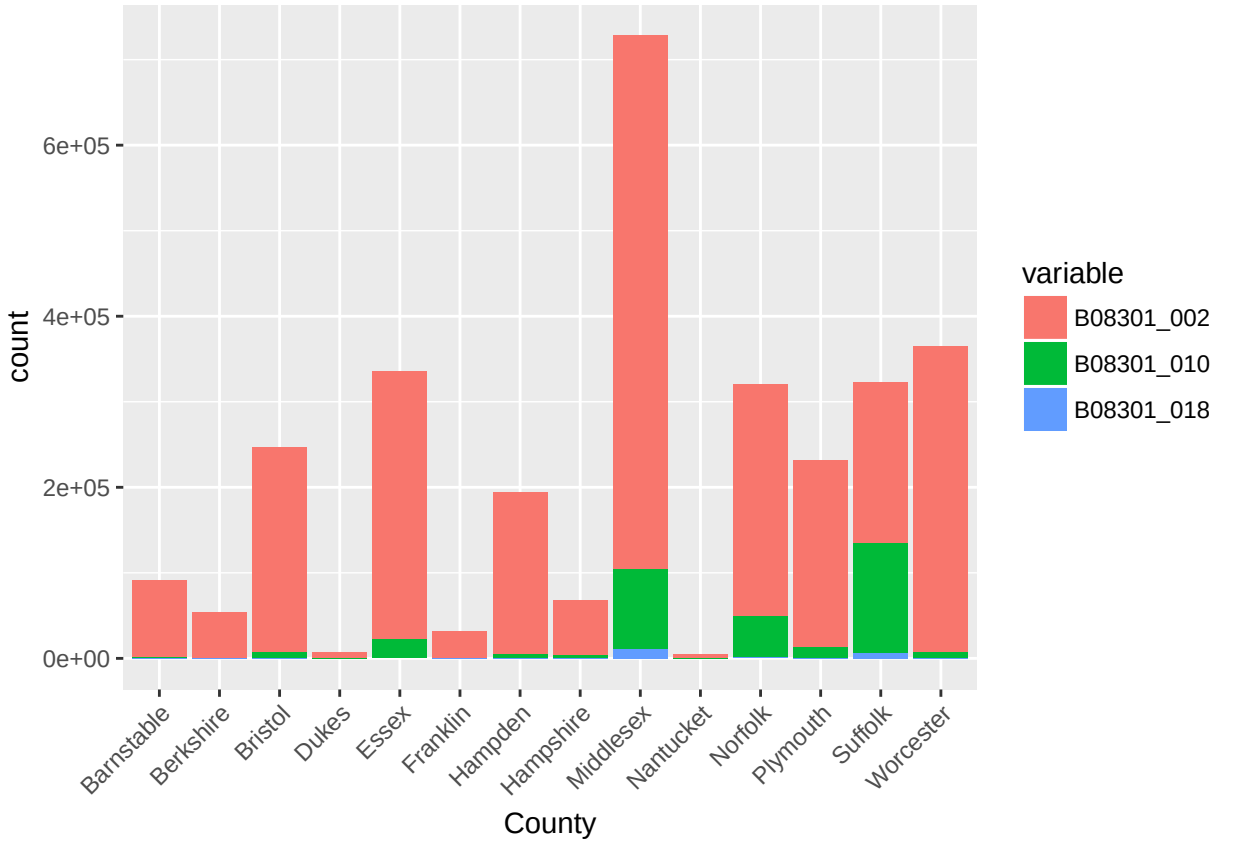
```
commute_vars = paste("B08301", str_pad(1:21, 3, "left", "0"), sep = "_")
mass_commute = get_acs(
  geography = "county",
  variables = commute_vars,
  endyear = 2015,
  output = "tidy",
  state = "MA",
  geometry = TRUE
)
mass_commute$NAME = str_replace(
  mass_commute$NAME, " County, Massachusetts", ""
)
```

Load the data in the viewer to see its structure.

Let's load this data into a bar plot by county. In the following code, we introduce `%>%`, the "pipe" operator. This is a tidyverse operator that lets you pipe, or send, the result of one statement to a further statement. Earlier, we saw that `ggplot` expects a data object as its first parameter. A function that is piped into automatically puts the result of the previous operation into the first parameter. Thus, in the code below, we *omit* a data object in the call to `ggplot`—it is piped in. If we try to read this in English it would be something like "Take the mass_commute data frame, filter it to return certain variables, then build a plot from the filtered data frame":

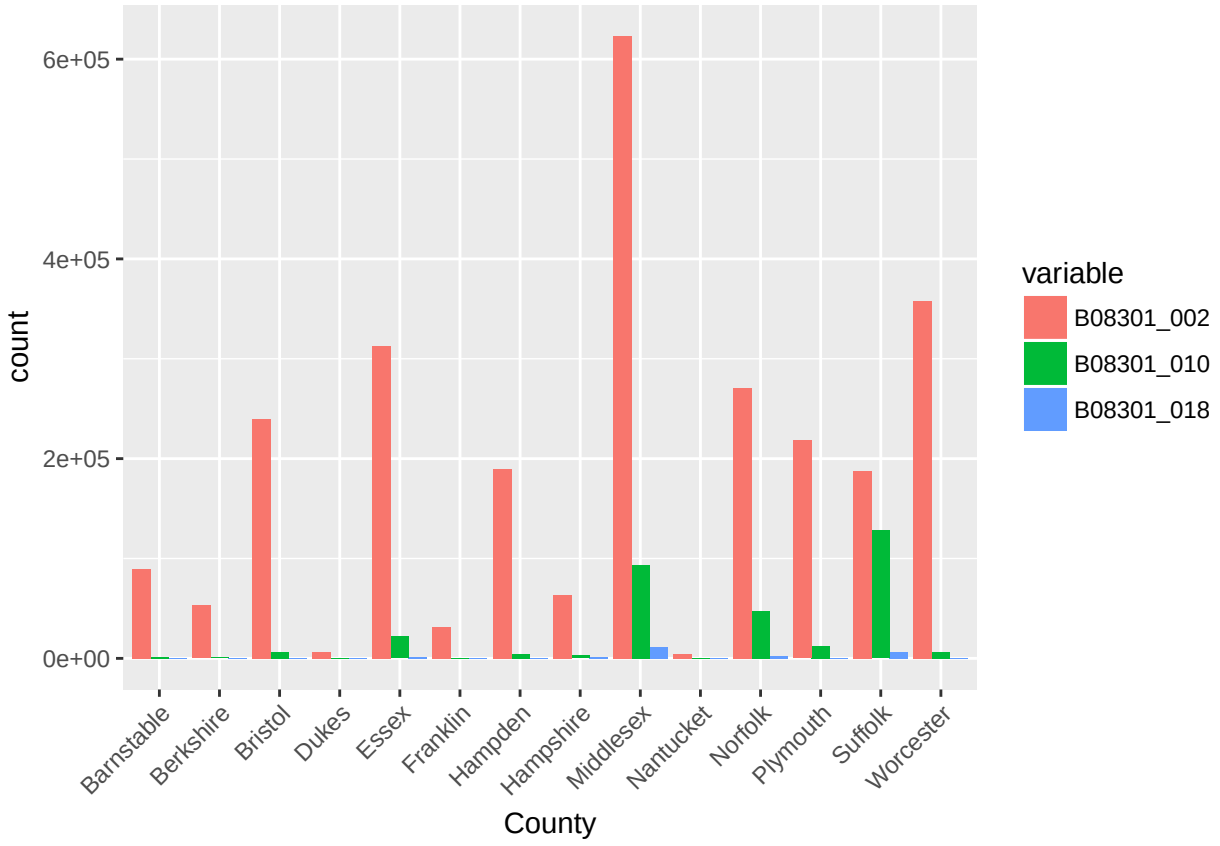
```
commute_filter = c("B08301_002", "B08301_010", "B08301_018")

mass_commute %>%
  filter(variable %in% commute_filter) %>%
  ggplot(aes(x = NAME, y = estimate, fill = variable)) +
    geom_col() +
    xlab("County") + ylab("count") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



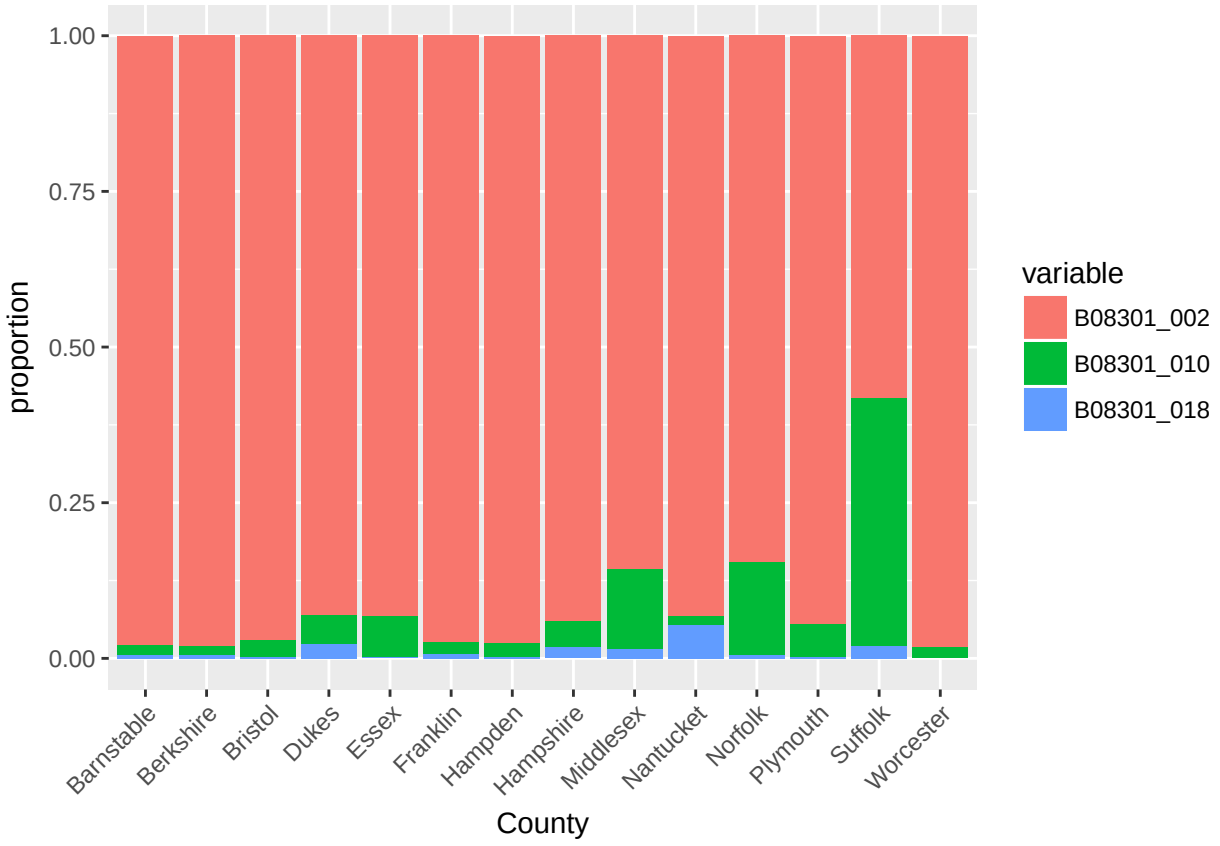
By default, `geom_col` (and `geom_bar`) produce stacked bar plots. To display the bars side-by-side, use the paramter `position = "dodge"`:

```
mass_commute %>%
  filter(variable %in% commute_filter) %>%
  ggplot(aes(x = NAME, y = estimate, fill = variable)) +
  geom_col(position = "dodge") +
  xlab("County") + ylab("count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Alternatively, the bar height can be standardized using the parameter `position = "fill"`, so that now you can see the *proportion* that each variable contributes to the whole:

```
mass_commute %>%
  filter(variable %in% commute_filter) %>%
  ggplot(aes(x = NAME, y = estimate, fill = variable)) +
  geom_col(position = "fill") +
  xlab("County") + ylab("proportion") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Loading Data for Mapping

In order to load data for mapping, we want it in the wide format. Tidy data can be mapped, but it is slower because each geometry is overplotted once for each variable. (This overplotting can also cause linework and text to be bold or fuzzy.)

We will overwrite our previous `mass_counties` object, and get both commute variables and the median household income we used previously.

```
# Get county geometries again. Include MHI.
mass_counties = get_acs(
  geography = "county",
  variables = c("B19013_001", commute_vars),
  endyear = 2015,
  output = "wide",
  state = "MA",
  geometry = TRUE
)
mass_counties$NAME = str_replace(
  mass_counties$NAME, " County, Massachusetts", ""
)
```

Note that the download went much faster because `get_acs` is reusing the geography data are already downloaded (look in your working directory).

Look at the structure of `mass_counties` in the data viewer. This is the wide format, with each variable in a separate column. Each variable actually appears in two columns with `E` and `M` appended to indicate the

estimate (remember, ACS is a survey, not a count) and a **margin of error** (90% confidence interval by default, although `tidycensus` lets you calculate other MOEs).

Since these data represent *counts*, they are not appropriate for choropleth mapping. After all, larger areas will tend to have more people. We can use dplyr to add proportion variables. In the process, we can also create variable names that are easier to work with.