Working with the U.S. Census

Adrien Allorant

2024-02-20

Goals of this activity

1. Download Census data using their API
2. Learn more data wrangling functions

If you have not done so already, install the following packages by running the following code:

```r
knitr::opts_chunk$set(
  warning = FALSE,
  message = FALSE
)
install.packages("tidycensus")
```

Let's load the packages we'll be using in this activity.

```
## Warning: package 'dplyr' was built under R version 4.2.3
```

```
## Warning: package 'stringr' was built under R version 4.2.3
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

Using the Census API: `tidycensus`

In order to directly download data from the Census API, you need a key. You can sign up for a free key here, which you should have already done during class.

Type your key in quotes using the `census_api_key()` command

```r
census_api_key("YOUR API KEY GOES HERE", install = TRUE)
```

The parameter `install = TRUE` tells R to use this API key in the future so you won't have to use `census_api_key()` again unless you update or re-download R.

The function for downloading American Community Survey (ACS) Census data is `get_acs()`. The command for downloading decennial Census data is `get_decennial()`. Getting variables using the Census API requires knowing the variable ID - and there are thousands of variables (and thus thousands of IDs) across the different Census files. To rapidly search for variables, use the commands `load_variables()` and `View()`. Because we'll be using the ACS in this guide, let's check the variables in the 2017-2021 5-year ACS using the following commands.

```
v21 <- load_variables(2021, "acs5", cache = TRUE)
View(v21)
```

A window should have popped up showing you a record layout of the 2017-2021 ACS. To search for specific data, select "Filter" located at the top left of this window and use the search boxes that pop up. For example, type in "Hispanic" in the box under "Label". You should see near the top of the list the first set of variables we'll want to download - race/ethnicity.

Let's extract race/ethnicity data and total population for Oregon counties using the `get_acs()` command.

```
or <- get_acs(geography = "county",
              year = 2021,
              variables = c(total_pop = "B03002_001",
                            white_pop = "B03002_003", black_pop = "B03002_004",
                            asian_pop = "B03002_006", hisp_pop = "B03002_012"),
              state = "OR",
              survey = "acs5",
              output = "wide")
```

```
## Getting data from the 2017-2021 5-year ACS
```

In the above code, we specified the following arguments:

- geography: The level of geography we want the data in; in our case, the county. Other geographic options can be found here. For many geographies, `tidycensus` supports more granular requests that are subsetted by state or even by county, if supported by the API. If a geographic subset is in bold, it is required; if not, it is optional. For example, if you supply `county = "Multnomah"` to the above code, you will get data just for Multnomah county. Note that you need to supply the state argument in order to get a specific county.
- year: The end year of the data (because we want 2017-2021, we use 2021).
- variables: The variables we want to bring in as specified in a vector you create using the function `c()`. Note that we created variable names of our own (e.g. "white_pop") and we put the ACS IDs in quotes ("B03002_003"). Had we not done this, the variable names will come in as they are named in the ACS, which are not very descriptive.
- state: We can filter the counties to those in a specific state. Here it is "OR" for Oregon. If we don't specify this, we get all counties in the United States.
- survey: The specific Census survey were extracting data from. We want data from the 5-year American Community Survey, so we specify "acs5". The ACS comes in 1- and 5-year varieties.
- output: The argument tells R to return a wide dataset as opposed to a long dataset. Last week, we saw how to transform data from wide to long using the `pivot_longer()` function. We won't be using this function in this guide, but it's good to know that you can get data in either format.

Another useful option to set is `cache_table = TRUE`, so you don't have to re-download after you've downloaded successfully the first time. Type in `?get_acs()` to see the full list of options.

Boundaries changed in 2020, which means that the 2017-2021 data will not completely merge with ACS data between 2010 and 2019. So make sure you merge 2020 data only with 2020 data (but you can merge 2019 data with data between 2010-2019). This is especially important for tract data, with many new tracts created in 2020 and existing tracts experiencing dramatic changes in their boundaries between 2010 and 2020.

What does our data set look like? Take a `glimpse()`:

```
glimpse(or)
```

```
## Rows: 36
## Columns: 12
## $ GEOID       <chr> "41001", "41003", "41005", "41007", "41009", "41011", "4101~
## $ NAME        <chr> "Baker County, Oregon", "Benton County, Oregon", "Clackamas~
## $ total_popE <dbl> 16539, 94667, 418577, 40720, 52381, 64619, 24300, 23234, 19~
## $ total_popM <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ white_popE <dbl> 14819, 74930, 334309, 33748, 45253, 54351, 21213, 19715, 16~
## $ white_popM <dbl> 70, 244, 1013, 559, 414, 216, 77, 119, 441, 376, 74, 49, 12~
## $ black_popE <dbl> 187, 1024, 3324, 379, 253, 294, 41, 25, 870, 332, 0, 4, 57,~
## $ black_popM <dbl> 24, 225, 409, 64, 84, 82, 51, 40, 280, 111, 13, 10, 35, 66,~
## $ asian_popE <dbl> 58, 6905, 18777, 416, 401, 686, 66, 212, 2101, 944, 11, 40,~
## $ asian_popM <dbl> 49, 309, 791, 135, 114, 176, 43, 62, 331, 244, 11, 18, 18, ~
## $ hisp_popE  <dbl> 796, 7468, 38231, 3617, 3007, 4476, 1918, 1779, 16215, 6809~
## $ hisp_popM  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 45, NA, NA, NA, NA,~
```

The tibble contains counties with their estimates for race/ethnicity. These variables end with the letter "E". It also contains the margins of error for each estimate. These variables end with the letter "M". Although important to evaluate, we won't be using the margins of error much in this class.

Congrats! You've just learned how to grab Census data from the Census API using `tidycensus`!

Data Wrangling

The ultimate goal is to merge the files `or` with some geographic information stored in `or_shp`.

```
library(sf)
library(tigris)
or_sf <- counties(state = "OR", cb = TRUE)
```

```
##   |                                                                              |
```

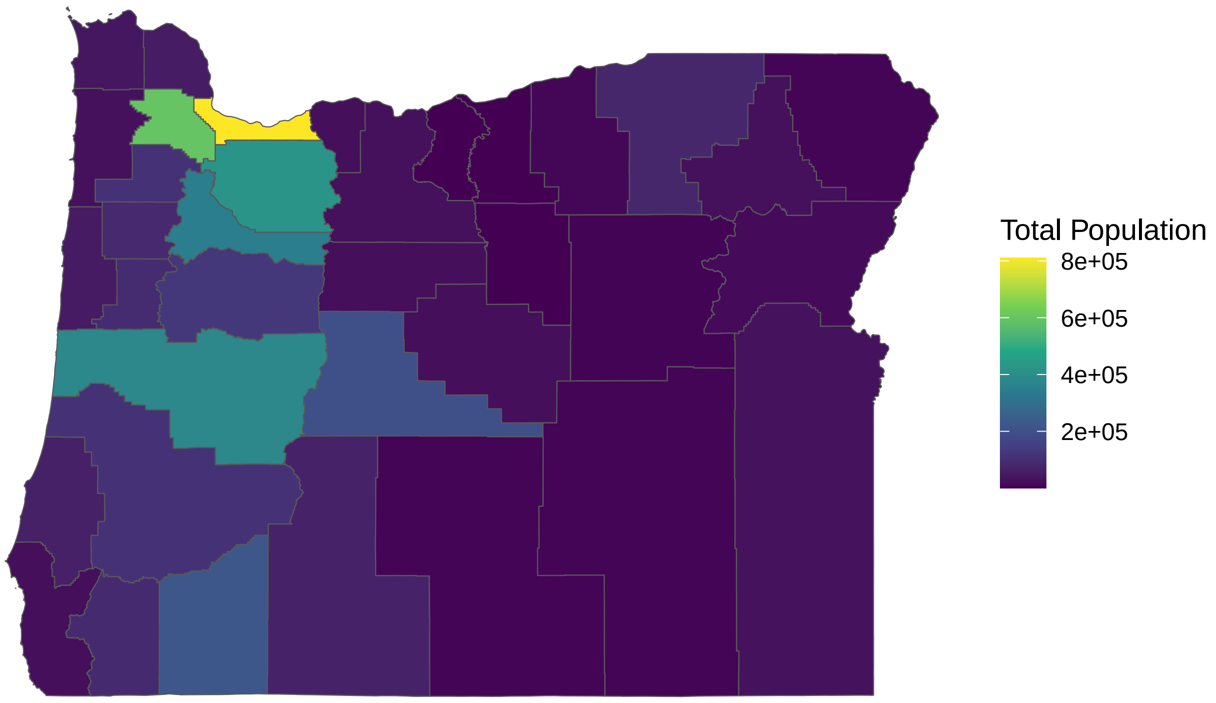
We will talk a lot more about spatial data in the next weeks. For now, let's focus on the data wrangling.

Joining tables

Our goal is to merge together the datasets `or` and `or_sf`. To merge the two datasets together, use the function left_join(), which matches pairs of observations whenever their keys or IDs are equal. We match on the variables GEOID save the merged data set into a new object called or_county.

```r
or_county <- left_join(or, or_sf, by = c("GEOID" = "GEOID"))

or_county %>%
  st_as_sf() %>%
  ggplot() +
  geom_sf(aes(fill = total_popE)) +
  scale_fill_viridis_c() +
  labs(fill = "Total Population") +
  theme_void()
```



Creating new variables

We can create new variables using the `mutate()` function. For example, you can create a variable for the percentage of the population that is white for each county in Oregon, and map it.

Census tracts

So far in this activity we've been working with county-level data. That is, the rows or units of observations in our data set represented counties. In most of the labs and activities moving forward, we will be working with neighborhood data, using census tracts to represent neighborhoods.

Let's bring in census tract racial/ethnic composition using `get_acs()`.

```r
ortracts <- get_acs(geography = "tract",
               year = 2021,
               variables = c(total_pop = "B03002_001",
                             white_pop = "B03002_003", black_pop = "B03002_004",
                             asian_pop = "B03002_006", hisp_pop = "B03002_012"),
               state = "OR",
               survey = "acs5",
               output = "wide")
```

```
## Getting data from the 2017-2021 5-year ACS
```

You'll find that the code is nearly identical to the code we used to bring in the county data, except we replace county with tract for `geography =`.

If you wanted tracts just in Multnomah county, then you specify `county = "Multnomah"`.

```
mutlnomah_tracts <- get_acs(geography = "tract",
              year = 2021,
              variables = c(total_pop = "B03002_001",
                            white_pop = "B03002_003", black_pop = "B03002_004",
                            asian_pop = "B03002_006", hisp_pop = "B03002_012"),
              state = "OR",
              county = "Multnomah",
              survey = "acs5",
              output = "wide")
```

```
## Getting data from the 2017-2021 5-year ACS
```

Look at the data to see if it contains only Multnomah county tracts.

Data Wrangling and Visualization

Let's look at housing values `B25077_001`. Choose your county of birth (or any county you like) and all its adjacent counties. Create a visualization of the distribution of the median home values by census tract, for each county, using the most recent ACS data available (probably 2022).

Next steps

We'll work with tract data more next week when we explore methods and techniques in spatial data analysis.

For more information on the `tidycensus` package, check out Kyle Walker's excellent book Analyzing US Census Data.