Working with the tidycensus package

Learning objectives

- Locate Census variables and tables
- Use the tidycensus package to interact with the Census API
- Process Census data into a GIS-friendly format

Getting started

The US Census has stores a mass of data which serves as the primary resource for nearly all public health research and secondary datasets. However, that doesn't mean its particulary easy to access and/or process. A quick search on data.census.gov might leave you a little disoriented. There are so many different tables and figuring out how you can download the data you want at the geographic resolution you require can be a laborious task. Even more so when you account for the fact that you may have to do quite a bit of reshaping of the data to get it in the particular format you require.

These problems become further compounded when you have to do them every year or maybe even several times in a single year as you may be discovering new data. That's where `tidycensus` comes in. `tidycensus` is an R package that interfaces with the US Census API (Application Programming Interface) to make it relatively painless to download and process Decennial Census and American Community Survey (ACS) data.

`tidycensus` also has a dependency called `tigris`. `tigris` is an R package designed to download Census boundaries. We will see later on how we can use `tigris`'s capabilities both inside `tidycensus` itself or from the package proper. We will also use the `tidyverse` for data manipulation, but you can feel free to translate any `tidyverse` functions to base R or any other data manipulation package.

If you have not already, you can install both `tidycensus` and the `tidyverse` with:

```
install.packages(c("tidycensus", "tidyverse"))
```

And then load each of the packages we will use today with:

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.4     ✔ readr     2.1.5
✔ forcats   1.0.0     ✔ stringr   1.5.1
✔ ggplot2   3.5.2     ✔ tibble    3.3.0
✔ lubridate 1.9.4     ✔ tidyr     1.3.1
✔ purrr     1.1.0
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
to become errors
To enable caching of data, set `options(tigris_use_cache = TRUE)`
in your R script or .Rprofile.
```

One last note: the Decennial Census and American Community Survey each have their own intricacies so the following sections will all be split up by dataset. Additionally, since the discontinuation of the 3-year ACS estimates in 2015, the ACS has published two datasets: 1-year and 5-year estimates. We will focus on the 5-year estimates.

Getting an API key

Before using the Census API, it is good practice to identify ourselves. We can do this by obtaining an API key. Go to https://api.census.gov/data/key_signup.html and fill out your information to obtain an API key. You will recieve an email with your API key.

Run the following functions with your API key filled in.

```
census_api_key("YOUR API KEY GOES HERE")

# reload your environment variables
readRenviron("~/.Renviron")
```

Discovery

Both the American Community Survey and Decennial Census Survey variables can be explored in R with the `load_variables()` function. But this isn't always the easiest approach.

American Community Survey

Looking through the ACS5 estimates is quite simple.

```
acs5_2023_vars <- load_variables(2023, "acs5")
acs5_2023_vars
```

```
# A tibble: 28,261 × 4
   name        label                                    concept        geography
   <chr>       <chr>                                    <chr>          <chr>
 1 B01001A_001 Estimate!!Total:                         Sex by Age (W… tract
 2 B01001A_002 Estimate!!Total:!!Male:                  Sex by Age (W… tract
 3 B01001A_003 Estimate!!Total:!!Male:!!Under 5 years   Sex by Age (W… tract
 4 B01001A_004 Estimate!!Total:!!Male:!!5 to 9 years    Sex by Age (W… tract
 5 B01001A_005 Estimate!!Total:!!Male:!!10 to 14 years  Sex by Age (W… tract
 6 B01001A_006 Estimate!!Total:!!Male:!!15 to 17 years  Sex by Age (W… tract
 7 B01001A_007 Estimate!!Total:!!Male:!!18 and 19 years Sex by Age (W… tract
 8 B01001A_008 Estimate!!Total:!!Male:!!20 to 24 years  Sex by Age (W… tract
 9 B01001A_009 Estimate!!Total:!!Male:!!25 to 29 years  Sex by Age (W… tract
10 B01001A_010 Estimate!!Total:!!Male:!!30 to 34 years  Sex by Age (W… tract
# ℹ 28,251 more rows
```

There are 28,261 variables! That's a lot to go through.

```
acs5_2023_tables <- acs5_2023_vars |>
  mutate(table = str_split_i(name, "_", 1)) |>
```

```
  nest(variables = c(name, label, geography))
acs5_2023_tables
```

```
# A tibble: 1,193 × 3
   concept                                                    table variables
   <chr>                                                      <chr> <list>
 1 Sex by Age (White Alone)                                   B010… <tibble>
 2 Sex by Age (Black or African American Alone)               B010… <tibble>
 3 Sex by Age (American Indian and Alaska Native Alone)       B010… <tibble>
 4 Sex by Age (Asian Alone)                                   B010… <tibble>
 5 Sex by Age (Native Hawaiian and Other Pacific Islander Alone) B010… <tibble>
 6 Sex by Age (Some Other Race Alone)                         B010… <tibble>
 7 Sex by Age (Two or More Races)                             B010… <tibble>
 8 Sex by Age (White Alone, Not Hispanic or Latino)           B010… <tibble>
 9 Sex by Age (Hispanic or Latino)                            B010… <tibble>
10 Sex by Age                                                 B010… <tibble>
# i 1,183 more rows
```

We are probably more interested in overall topics so we could also search by table. Thankfully, there are a relatively more manageable 1,193 tables to go through.

You may notice that there are repetitive concepts that are the same concept but for a certain subgroup. We can remove these to simplify the concepts into "larger concepts".

```
acs5_2023_largerconcepts <- acs5_2023_tables |>
  mutate(larger_concept = str_remove(concept, " \\(.*\\)")) |>
  nest(tables = c(table, concept, variables))
acs5_2023_largerconcepts
```

```
# A tibble: 725 × 2
   larger_concept                                                     tables
   <chr>                                                              <list>
 1 Sex by Age                                                         <tibble>
 2 Median Age by Sex                                                  <tibble>
 3 Total Population                                                   <tibble>
 4 Race                                                               <tibble>
 5 White Alone or in Combination With One or More Other Races         <tibble>
 6 Black or African American Alone or in Combination With One or More … <tibble>
 7 American Indian and Alaska Native Alone or in Combination With One … <tibble>
 8 Asian Alone or in Combination With One or More Other Races         <tibble>
 9 Native Hawaiian and Other Pacific Islander Alone or in Combination … <tibble>
10 Some Other Race Alone or in Combination With One or More Other Races <tibble>
# i 715 more rows
```

This leaves only 725 larger concepts.

We can now search through these topics programatically. For example, we can look for health insurance related variables by detecting the prefix `"insur"`.

```r
acs5_2023_largerconcepts |>
  filter(str_detect(larger_concept, fixed("insur", ignore_case = TRUE)))
```

```
# A tibble: 27 × 2
   larger_concept                                                           tables
   <chr>                                                                    <list>
 1 Age by Disability Status by Health Insurance Coverage Status             <tibble>
 2 Homeowners Insurance Costs by Mortgage Status                            <tibble>
 3 Health Insurance Coverage Status by Sex by Age                           <tibble>
 4 Private Health Insurance Status by Sex by Age                            <tibble>
 5 Public Health Insurance Status by Sex by Age                             <tibble>
 6 Types of Health Insurance Coverage by Age                                <tibble>
 7 Health Insurance Coverage Status and Type by Employment Status           <tibble>
 8 Health Insurance Coverage Status and Type by Household Income in th… <tibble>
 9 Health Insurance Coverage Status and Type by Age by Educational Att… <tibble>
10 Health Insurance Coverage Status and Type by Citizenship Status          <tibble>
# ℹ 17 more rows
```

```r
# you can always unnest to look at the inidividual tables
acs5_2023_largerconcepts |>
  filter(str_detect(larger_concept, fixed("insur", ignore_case = TRUE))) |>
  unnest(tables)
```

```
# A tibble: 35 × 4
   larger_concept                                        table concept variables
   <chr>                                                 <chr> <chr>   <list>
 1 Age by Disability Status by Health Insurance Coverag… B181… Age by… <tibble>
 2 Homeowners Insurance Costs by Mortgage Status         B251… Homeow… <tibble>
 3 Health Insurance Coverage Status by Sex by Age        B270… Health… <tibble>
 4 Private Health Insurance Status by Sex by Age         B270… Privat… <tibble>
 5 Public Health Insurance Status by Sex by Age          B270… Public… <tibble>
 6 Types of Health Insurance Coverage by Age             B270… Types … <tibble>
 7 Health Insurance Coverage Status and Type by Employm… B270… Health… <tibble>
 8 Health Insurance Coverage Status and Type by Househo… B270… Health… <tibble>
 9 Health Insurance Coverage Status and Type by Age by … B270… Health… <tibble>
10 Health Insurance Coverage Status and Type by Citizen… B270… Health… <tibble>
# ℹ 25 more rows
```

```r
# or even look for individual variables
acs5_2023_largerconcepts |>
  filter(str_detect(
    larger_concept,
    fixed("Health Insurance Coverage Status by Sex by Age")
  )) |>
  unnest(tables) |>
  unnest(variables)
```

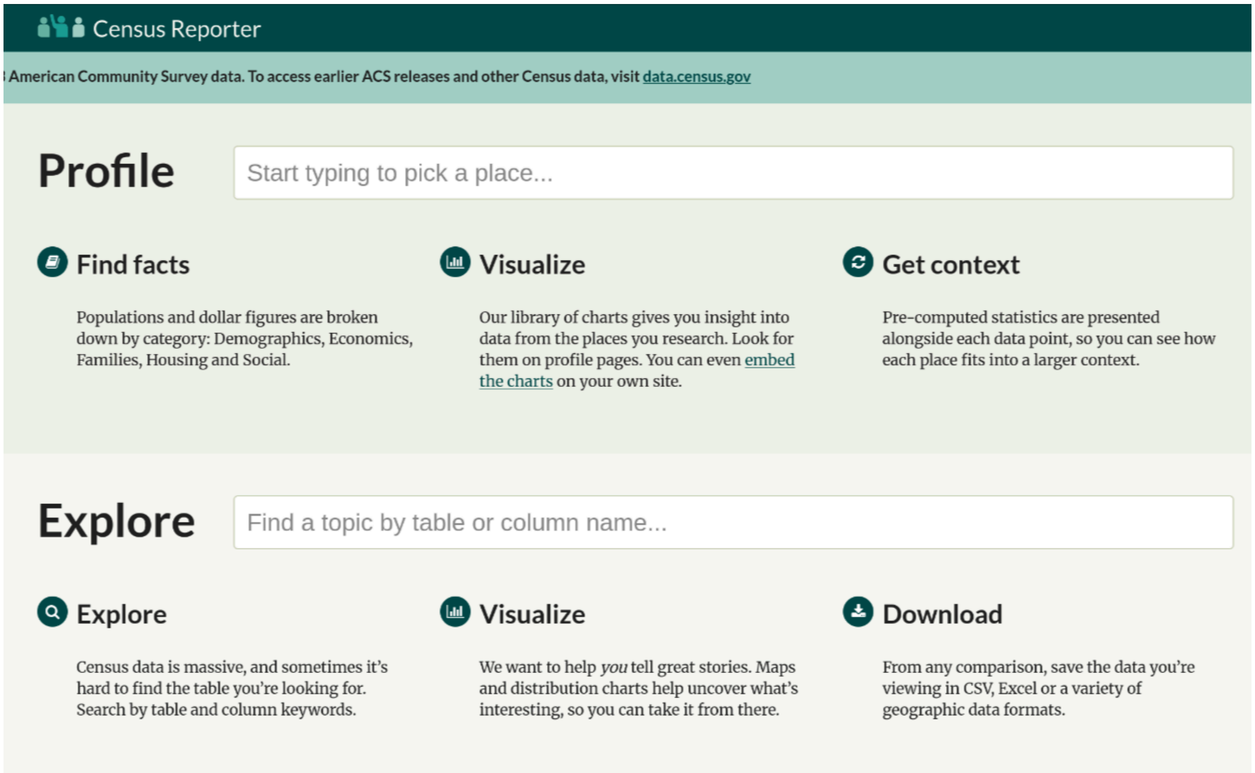
```
# A tibble: 57 × 6
   larger_concept                                table concept name  label geography
   <chr>                                         <chr> <chr>   <chr> <chr> <chr>
 1 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 2 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 3 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 4 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 5 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 6 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 7 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 8 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 9 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
10 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
# ℹ 47 more rows
```

Or we can look for variables within RStudio or Positron's data viewer.

```r
View(acs5_2023_largerconcepts)

# or combine the two
acs5_2023_largerconcepts |>
  filter(str_detect(
    larger_concept,
    fixed("Health Insurance Coverage Status by Sex by Age")
  )) |>
  unnest(tables) |>
  unnest(variables) |>
  View()
```

But this is not the only way to search for ACS variables. I recommend checking out the immensely helpful Census Reporter website, which not only has a significantly simpler interface to search for ACS tables, but also a number of Topics articles which highlight tables which fall within broader topics of interest.



Once you find a table of interest, you can even identify the identifiers of specific variables by simply hovering over them within the variable name.

Decennial Census

Looking through the Decennial Census variables is a bit more complicated in comparsion. The first complication is that there are several summary files that make up the Decennial Census. We can use the `summary_files()` function to see just how many summary files compose the 2020 Census.

```
summary_files(2020)
```

```
 [1] "pl"    "dhc"   "dp"    "pes"   "dpas"  "ddhca" "dpmp"  "dpgu"  "dpvi"
[10] "ddhcb" "sdhc"  "dhcvi" "dhcgu" "dhcvi" "dhcas" "cd118"
```

16 different files! Not to worry though, the Census website provides documentation summarizing the 2020 Decennial Census Products.

We can also look through all the files together. First, let's get all the summary files.

```
dec_2020_sumfiles <- summary_files(2020)
```

Then let's filter down to only those `tidycensus` can retrieve info for AND only those that are for the 50 states.

```r
possible_sumfiles <- eval(formals(tidycensus::load_variables)$dataset)

dec_2020_sumfiles <- dec_2020_sumfiles |>
  keep(\(dataset) dataset %in% possible_sumfiles) |>
  # remove decennial census sumfiles for guam, virgin islands, american samoa,
northern marina islands
  str_subset("(gu|vi|as|mp)$", negate = TRUE)
```

```r
dec_2020_vars <- dec_2020_sumfiles |>
  set_names() |>
  map(\(dataset) load_variables(2020, dataset)) |>
  list_rbind(names_to = "sumfile")
dec_2020_vars
```

```
# A tibble: 19,744 × 4
   sumfile name    label                                                concept
   <chr>   <chr>   <chr>                                                <chr>
 1 pl      H1_001N " !!Total:"                                          OCCUPA…
 2 pl      H1_002N " !!Total:!!Occupied"                                OCCUPA…
 3 pl      H1_003N " !!Total:!!Vacant"                                  OCCUPA…
 4 pl      P1_001N " !!Total:"                                          RACE
 5 pl      P1_002N " !!Total:!!Population of one race:"                 RACE
 6 pl      P1_003N " !!Total:!!Population of one race:!!White alone"    RACE
 7 pl      P1_004N " !!Total:!!Population of one race:!!Black or Africa… RACE
 8 pl      P1_005N " !!Total:!!Population of one race:!!American Indian… RACE
 9 pl      P1_006N " !!Total:!!Population of one race:!!Asian alone"    RACE
10 pl      P1_007N " !!Total:!!Population of one race:!!Native Hawaiian… RACE
# ℹ 19,734 more rows
```

Notice that there are 19,744 variables across all of the summary files. We can use the same approach as before to look at the individual tables.

```r
dec_2020_tables <- dec_2020_vars |>
  mutate(table = str_split_i(name, "_", 1), .after = sumfile) |>
  nest(variables = c(name, label))
dec_2020_tables
```

```
# A tibble: 577 × 4
   sumfile table concept                                              variables
   <chr>   <chr> <chr>                                                <list>
 1 pl      H1    OCCUPANCY STATUS                                     <tibble>
 2 pl      P1    RACE                                                 <tibble>
 3 pl      P2    HISPANIC OR LATINO, AND NOT HISPANIC OR LATINO BY RA… <tibble>
 4 pl      P3    RACE FOR THE POPULATION 18 YEARS AND OVER            <tibble>
 5 pl      P4    HISPANIC OR LATINO, AND NOT HISPANIC OR LATINO BY RA… <tibble>
 6 pl      P5    GROUP QUARTERS POPULATION BY MAJOR GROUP QUARTERS TY… <tibble>
 7 dhc     H10   TENURE BY RACE OF HOUSEHOLDER                        <tibble>
```

```
 8 dhc     H11   TENURE BY HISPANIC OR LATINO ORIGIN OF HOUSEHOLDER    <tibble>
 9 dhc     H12A  TENURE BY HOUSEHOLD SIZE (WHITE ALONE HOUSEHOLDER)    <tibble>
10 dhc     H12B  TENURE BY HOUSEHOLD SIZE (BLACK OR AFRICAN AMERICAN … <tibble>
# ℹ 567 more rows
```

There's actually only 577 tables. Let's look at how many broader concepts there are using the same approach as before.

```
dec_2020_largerconcepts <- dec_2020_tables |>
  mutate(larger_concept = str_remove(concept, " \\(.*\\)")) |>
  nest(tables = c(table, concept, variables))
dec_2020_largerconcepts
```

```
# A tibble: 141 × 3
   sumfile larger_concept                                                tables
   <chr>   <chr>                                                         <list>
 1 pl      OCCUPANCY STATUS                                              <tibble>
 2 pl      RACE                                                          <tibble>
 3 pl      HISPANIC OR LATINO, AND NOT HISPANIC OR LATINO BY RACE        <tibble>
 4 pl      RACE FOR THE POPULATION 18 YEARS AND OVER                     <tibble>
 5 pl      HISPANIC OR LATINO, AND NOT HISPANIC OR LATINO BY RACE FOR … <tibble>
 6 pl      GROUP QUARTERS POPULATION BY MAJOR GROUP QUARTERS TYPE        <tibble>
 7 dhc     TENURE BY RACE OF HOUSEHOLDER                                 <tibble>
 8 dhc     TENURE BY HISPANIC OR LATINO ORIGIN OF HOUSEHOLDER            <tibble>
 9 dhc     TENURE BY HOUSEHOLD SIZE                                      <tibble>
10 dhc     TENURE BY AGE OF HOUSEHOLDER                                  <tibble>
# ℹ 131 more rows
```

There are only 141 larger concepts in our selected decennial census summary files. Now its a much more manageable task to find our variables/tables of interest.

Retrieval

American Community Survey

Retrieving ACS data is requires the `get_acs()` function. I recommend using a named vector to specify your variables as it will make it significantly easier to understand what data you are pulling.

```
il23_ac5_median_hh_inc <- get_acs(
  geography = "tract",
  variables = c(median_hh_inc = "B19013_001"),
  year = 2023,
  state = "IL"
)
```

```
Getting data from the 2019-2023 5-year ACS
```

```
Warning: • You have not set a Census API key. Users without a key are limited to 500
queries per day and may experience performance limitations.
ℹ For best results, get a Census API key at
http://api.census.gov/data/key_signup.html and then supply the key to the
`census_api_key()` function to use it throughout your tidycensus session.
This warning is displayed once per session.
```

```
il23_ac5_median_hh_inc
```

```
# A tibble: 3,265 × 5
   GEOID       NAME                                       variable estimate   moe
   <chr>       <chr>                                      <chr>       <dbl> <dbl>
 1 17001000100 Census Tract 1; Adams County; Illinois    median_…    69049 11441
 2 17001000201 Census Tract 2.01; Adams County; Illinois median_…    51979 11944
 3 17001000202 Census Tract 2.02; Adams County; Illinois median_…    71000 10147
 4 17001000400 Census Tract 4; Adams County; Illinois    median_…    39469  6780
 5 17001000500 Census Tract 5; Adams County; Illinois    median_…    45966 15681
 6 17001000600 Census Tract 6; Adams County; Illinois    median_…    72115 12787
 7 17001000700 Census Tract 7; Adams County; Illinois    median_…    19213  8360
 8 17001000800 Census Tract 8; Adams County; Illinois    median_…    31837  9862
 9 17001000900 Census Tract 9; Adams County; Illinois    median_…    46023 10698
10 17001001001 Census Tract 10.01; Adams County; Illino… median_…    62375 10246
# ℹ 3,255 more rows
```

Notice that since the ACS yields us survey estimates, a margin of error is also provided.

> ℹ Note
>
> The ACS provides margin of errors at 90% confidence level. For more information about the US Census methodology pertaining to margin of errors check Chapter 7 of the ACS Handbook.

We can also specify several variables.

```
il23_ac5_median_fam_inc <- get_acs(
  geography = "tract",
  variables = c(
    median_fam_inc_2 = "B19119_002",
    median_fam_inc_3 = "B19119_003",
    median_fam_inc_4 = "B19119_004",
    median_fam_inc_5 = "B19119_005",
    median_fam_inc_6 = "B19119_006",
    median_fam_inc_7p = "B19119_007"
  ),
  year = 2023,
  state = "IL"
)
```

```
Getting data from the 2019-2023 5-year ACS
```

```
il23_ac5_median_fam_inc
```

```
# A tibble: 19,590 × 5
   GEOID       NAME                                       variable estimate   moe
   <chr>       <chr>                                      <chr>       <dbl> <dbl>
 1 17001000100 Census Tract 1; Adams County; Illinois    median_…     93000 28497
 2 17001000100 Census Tract 1; Adams County; Illinois    median_…        NA    NA
 3 17001000100 Census Tract 1; Adams County; Illinois    median_…    165625 76390
 4 17001000100 Census Tract 1; Adams County; Illinois    median_…    112650 50234
 5 17001000100 Census Tract 1; Adams County; Illinois    median_…        NA    NA
 6 17001000100 Census Tract 1; Adams County; Illinois    median_…        NA    NA
 7 17001000201 Census Tract 2.01; Adams County; Illinois median_…     47500 12969
 8 17001000201 Census Tract 2.01; Adams County; Illinois median_…     66250 29786
 9 17001000201 Census Tract 2.01; Adams County; Illinois median_…    172533 60731
10 17001000201 Census Tract 2.01; Adams County; Illinois median_…    104342 83360
# ℹ 19,580 more rows
```

Or even whole tables.

```
il23_ac5_agg_hh_inc_by_age <- get_acs(
  geography = "tract",
  table = c(agg_hh_inc_by_age = "B19050"),
  year = 2023,
  state = "IL"
)
```

```
Getting data from the 2019-2023 5-year ACS
```

```
Loading ACS5 variables for 2023 from table B19050. To cache this dataset for faster
access to ACS tables in the future, run this function with `cache_table = TRUE`. You
only need to do this once per ACS dataset.
```

```
il23_ac5_agg_hh_inc_by_age
```

```
# A tibble: 16,325 × 5
   GEOID       NAME                                     variable estimate     moe
   <chr>       <chr>                                    <chr>       <dbl>   <dbl>
 1 17001000100 Census Tract 1; Adams County; Illinois   B19050_…       NA NA
 2 17001000100 Census Tract 1; Adams County; Illinois   B19050_…       NA NA
 3 17001000100 Census Tract 1; Adams County; Illinois   B19050_…       NA NA
 4 17001000100 Census Tract 1; Adams County; Illinois   B19050_…       NA NA
 5 17001000100 Census Tract 1; Adams County; Illinois   B19050_…       NA NA
 6 17001000201 Census Tract 2.01; Adams County; Illin… B19050_… 63052700   1.21e7
```

```
 7 17001000201 Census Tract 2.01; Adams County; Illin… B19050_…   972400  1.20e6
 8 17001000201 Census Tract 2.01; Adams County; Illin… B19050_… 23642900  1.10e7
 9 17001000201 Census Tract 2.01; Adams County; Illin… B19050_… 25601900  6.98e6
10 17001000201 Census Tract 2.01; Adams County; Illin… B19050_… 12835600  3.85e6
# ℹ 16,315 more rows
```

But notice that when you have several variables, you will recieve a table with several rows per geographic area. We can change that by changing the `output` from `"tidy"` to `"wide"`.

```
il23_ac5_agg_hh_inc_by_age <- get_acs(
  geography = "tract",
  table = c(agg_hh_inc_by_age = "B19050"),
  year = 2023,
  state = "IL",
  output = "wide"
)
```

```
Getting data from the 2019-2023 5-year ACS
```

```
Loading ACS5 variables for 2023 from table B19050. To cache this dataset for faster
access to ACS tables in the future, run this function with `cache_table = TRUE`. You
only need to do this once per ACS dataset.
```

```
il23_ac5_agg_hh_inc_by_age
```

```
# A tibble: 3,265 × 12
   GEOID       NAME   B19050_001E B19050_001M B19050_002E B19050_002M B19050_003E
   <chr>       <chr>        <dbl>       <dbl>       <dbl>       <dbl>       <dbl>
 1 17001000100 Cens…           NA          NA          NA          NA          NA
 2 17001000201 Cens…     63052700    12113027      972400     1198371    23642900
 3 17001000202 Cens…           NA          NA          NA          NA          NA
 4 17001000400 Cens…     68008100    13399808     1512600     1653786    23495900
 5 17001000500 Cens…     48978600     7824240     3420900     2763193    14019800
 6 17001000600 Cens…    207358900    35308120     4064600     3355019    64160400
 7 17001000700 Cens…     23361800     8081700      427300      539268    11263900
 8 17001000800 Cens…           NA          NA          NA          NA          NA
 9 17001000900 Cens…     80723500    26557560     5639700     4078839    24594300
10 17001001001 Cens…     94818200    14017916     3078400     2783122    30059000
# ℹ 3,255 more rows
# ℹ 5 more variables: B19050_003M <dbl>, B19050_004E <dbl>, B19050_004M <dbl>,
#   B19050_005E <dbl>, B19050_005M <dbl>
```

This is especially helpful if you would like to also attach the geospatial features to the dataset.

```
il23_ac5_agg_hh_inc_by_age_geom <- get_acs(
  geography = "tract",
```

```r
  table = c(agg_hh_inc_by_age = "B19050"),
  year = 2023,
  state = "IL",
  output = "wide",
  geometry = TRUE
)
il23_ac5_agg_hh_inc_by_age_geom
```

```
Simple feature collection with 3265 features and 12 fields (with 2 geometries empty)
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -91.51308 ymin: 36.9703 xmax: -87.4952 ymax: 42.50848
Geodetic CRS:  NAD83
First 10 features:
         GEOID                                          NAME B19050_001E
1  17143003000       Census Tract 30; Peoria County; Illinois   192132400
2  17143000300        Census Tract 3; Peoria County; Illinois          NA
3  17143001800       Census Tract 18; Peoria County; Illinois    99562100
4  17143002000       Census Tract 20; Peoria County; Illinois    60112500
5  17073030600       Census Tract 306; Henry County; Illinois    97636900
6  17073030900       Census Tract 309; Henry County; Illinois   116859800
7  17019001205 Census Tract 12.05; Champaign County; Illinois   504562100
8  17019000402  Census Tract 4.02; Champaign County; Illinois          NA
9  17019001100    Census Tract 11; Champaign County; Illinois   213503200
10 17115001400        Census Tract 14; Macon County; Illinois    95219100
   B19050_001M B19050_002E B19050_002M B19050_003E B19050_003M B19050_004E
1     24493856     1874200     1518071    83117600    18641108    70914400
2           NA          NA          NA          NA          NA          NA
3     44021194     7091700     2909145    21638600    13640550    60947800
4     12730919     1851900     1392719    16476100     6144981    32953900
5     36452740          NA          NA    20824500     7211725    47678300
6     19030424     3179000     2339218    48356800    20584014    43256600
7     86195276    19294700    17262873   153071900    44260562   264196800
8           NA          NA          NA          NA          NA          NA
9     46552975     3433300     2716808    64137100    24337792    87298800
10    26427792     3392100     3480502    49569600    24275845    20799800
   B19050_004M B19050_005E B19050_005M                       geometry
1     18728520    36226200     7544802 MULTIPOLYGON (((-89.65023 4...
2           NA          NA          NA MULTIPOLYGON (((-89.63284 4...
3     41768622     9884000     4364317 MULTIPOLYGON (((-89.61315 4...
4     13087480     8830600     4728553 MULTIPOLYGON (((-89.62541 4...
5     35197103    29134100    13211261 MULTIPOLYGON (((-89.97326 4...
6      9479061    22067400     7118214 MULTIPOLYGON (((-89.9504 41...
7     69184203    67998800    27290909 MULTIPOLYGON (((-88.35067 4...
8           NA          NA          NA MULTIPOLYGON (((-88.24273 4...
9     42325972    58633900    19041135 MULTIPOLYGON (((-88.27677 4...
10     8712500    21457500     5281076 MULTIPOLYGON (((-88.97343 3...
```

Decennial Census

Retrieving Decennial Census data requires the `get_decennial()` function. It's interface is very similar to `get_acs()`,

```
il23_dec_n_renters <- get_decennial(
  geography = "tract",
  variables = c(n_renters = "H5_002N"),
  year = 2020,
  sumfile = "dhc",
  state = "IL"
)
```

```
Getting data from the 2020 decennial Census
```

```
Using the Demographic and Housing Characteristics File
```

```
Note: 2020 decennial Census data use differential privacy, a technique that
introduces errors into data to preserve respondent confidentiality.
ℹ Small counts should be interpreted with caution.
ℹ See https://www.census.gov/library/fact-sheets/2021/protecting-the-confidentiality-
of-the-2020-census-redistricting-data.html for additional guidance.
This message is displayed once per session.
```

```
il23_dec_n_renters <- get_decennial(
  geography = "tract",
  variables = c(n_renters = "H5_002N"),
  year = 2020,
  sumfile = "dhc",
  state = "IL"
)
```

```
Getting data from the 2020 decennial Census
```

```
Using the Demographic and Housing Characteristics File
```

Generation of derived estimates

Both the ACS and Decennial census provide a ton of information. However, often times you may want to create a derived estimate. Examples could be estimates for larger age groups, converting count estimates into rates, or computing odds ratios. Whatever the purpose, `tidycensus` can help!

Let's look at the first example: creating estimates for larger groups.

I want to find the health insurance coverage rate for individuals 18 and below within Illinois at the census tract level.

When I looked for health insurance coverage, I find that health insurance coverage status is provided stratified by sex and age.

```
acs5_2023_largerconcepts |>
  filter(str_detect(
    larger_concept,
    fixed("Health Insurance Coverage Status by Sex by Age")
  )) |>
  unnest(tables) |>
  unnest(variables)
```

```
# A tibble: 57 × 6
   larger_concept                            table concept name  label geography
   <chr>                                     <chr> <chr>   <chr> <chr> <chr>
 1 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 2 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 3 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 4 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 5 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 6 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 7 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 8 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
 9 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
10 Health Insurance Coverage Status by Sex … B270… Health… B270… Esti… tract
# ℹ 47 more rows
```

This means I'm going to have to add up all the variables for males and females 18 and below.

We can get the data first by specifying our query using `get_acs()`.

```
il23_ac5_healthins <- get_acs(
  geography = "tract",
  table = "B27001",
  year = 2023,
  state = "IL"
)
```

```
Getting data from the 2019-2023 5-year ACS
```

```
Loading ACS5 variables for 2023 from table B27001. To cache this dataset for faster
access to ACS tables in the future, run this function with `cache_table = TRUE`. You
only need to do this once per ACS dataset.
```

Using our variable information we retrieved earlier, we can add labels to all the variables.

```
il23_ac5_healthins <- il23_ac5_healthins |>
  left_join(acs5_2023_vars, join_by(variable == name)) |>
```

```r
  select(GEOID, label, estimate, moe) |>
  mutate(
    gender = str_remove(str_split_i(label, "!!", 3), ":"),
    age_group = str_remove(str_split_i(label, "!!", 4), ":"),
    coverage_status = str_remove(str_split_i(label, "!!", 5), ":")
  )
il23_ac5_healthins
```

```
# A tibble: 186,105 × 7
   GEOID       label                estimate   moe gender age_group coverage_status
   <chr>       <chr>                   <dbl> <dbl> <chr>  <chr>     <chr>
 1 17001000100 Estimate!!Total:         4459   493 <NA>   <NA>      <NA>
 2 17001000100 Estimate!!Total:…        1839   291 Male   <NA>      <NA>
 3 17001000100 Estimate!!Total:…          50    65 Male   Under 6 … <NA>
 4 17001000100 Estimate!!Total:…          50    65 Male   Under 6 … With health in…
 5 17001000100 Estimate!!Total:…           0    12 Male   Under 6 … No health insu…
 6 17001000100 Estimate!!Total:…         230    90 Male   6 to 18 … <NA>
 7 17001000100 Estimate!!Total:…         230    90 Male   6 to 18 … With health in…
 8 17001000100 Estimate!!Total:…           0    12 Male   6 to 18 … No health insu…
 9 17001000100 Estimate!!Total:…          49    68 Male   19 to 25… <NA>
10 17001000100 Estimate!!Total:…          49    68 Male   19 to 25… With health in…
# ℹ 186,095 more rows
```

And then, we can 1. Filter out the variables with no gender, age_group or coverage_status. 2. Filter the variables for those that represent 18 and under. 3. Calculate the total number of 18 and under individuals with coverage and in total. 4. Calculate the number of individuals covered divided by the number of individuals in total.

```r
il23_ac5_healthins18below <- il23_ac5_healthins |>
  filter(!if_any(c(gender, age_group, coverage_status), is.na)) |>
  filter(age_group %in% c("Under 6 years", "6 to 18 years")) |>
  group_by(GEOID) |>
  summarize(
    n_covered = sum(estimate[
      coverage_status == "With health insurance coverage"
    ]),
    n_total = sum(estimate)
  ) |>
  mutate(
    pct_covered = n_covered / n_total
  )
il23_ac5_healthins18below
```

```
# A tibble: 3,265 × 4
   GEOID       n_covered n_total pct_covered
   <chr>           <dbl>   <dbl>       <dbl>
 1 17001000100       796     796           1
 2 17001000201       450     450           1
```

```
 3 17001000202       587     587       1    
 4 17001000400       858     858       1    
 5 17001000500       509     516       0.986
 6 17001000600       911     948       0.961
 7 17001000700       105     110       0.955
 8 17001000800       529     543       0.974
 9 17001000900       538     538       1    
10 17001001001       811     827       0.981
# ℹ 3,255 more rows
```

One thing you may noticed is that in this process we dropped our margin or error estimates. This is not ideal because we lose any idea of reliability of our estimates. Thankfully, `tidycensus` can help with that. `tidycensus` has a host of `moe_*()` functions that can help you estimate margin of error for derived estimates. Let's use two of these, `moe_sum()` and `moe_prop()`.

```
il23_ac5_healthins18below <- il23_ac5_healthins |>
  filter(!if_any(c(gender, age_group, coverage_status), is.na)) |>
  filter(age_group %in% c("Under 6 years", "6 to 18 years")) |>
  group_by(GEOID) |>
  summarize(
    n_covered = sum(estimate[
      coverage_status == "With health insurance coverage"
    ]),
    moe_n_covered = moe_sum(
      moe[coverage_status == "With health insurance coverage"],
      estimate = estimate[coverage_status == "With health insurance coverage"]
    ),
    n_total = sum(estimate),
    moe_n_total = moe_sum(moe, estimate = estimate)
  ) |>
  mutate(
    pct_covered = n_covered / n_total,
    moe_pct_covered = moe_prop(n_covered, n_total, moe_n_covered, moe_n_total)
  )
il23_ac5_healthins18below
```

```
# A tibble: 3,265 × 7
  GEOID n_covered moe_n_covered n_total moe_n_total pct_covered moe_pct_covered
  <chr>     <dbl>         <dbl>   <dbl>       <dbl>       <dbl>           <dbl>
1 1700…       796         167.      796       168.        1              0.297
2 1700…       450         154.      450       155.        1              0.485
3 1700…       587         158.      587       158.        1              0.381
4 1700…       858         250.      858       251.        1              0.413
5 1700…       509         144.      516       145.        0.986          0.0333
6 1700…       911         202.      948       210.        0.961          0.302
7 1700…       105          64.4     110        66.6       0.955          0.0935
8 1700…       529         215.      543       216.        0.974          0.0761
9 1700…       538         159.      538       159.        1              0.417
```

```
10 1700…        811          243.       827        244.         0.981            0.0471
# ℹ 3,255 more rows
```

Retrieval of geographic boundaries

We retrieved and processed our data, but now lets transfer that data to ArcGIS Pro to visualize it.

Our table above has our tract level data, but no geometry. We can use the `tigris` package to download the tract geometry and add it to our table!

```
il23tracts <- tracts(state = "IL", cb = TRUE, year = 2023) |>
  select(GEOID)

il23_ac5_healthins18below_geom <- il23_ac5_healthins18below |>
  left_join(x = il23tracts)
```

```
Joining with `by = join_by(GEOID)`
```

```
il23_ac5_healthins18below_geom
```

```
Simple feature collection with 3263 features and 7 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -91.51308 ymin: 36.9703 xmax: -87.4952 ymax: 42.50848
Geodetic CRS:  NAD83
First 10 features:
         GEOID n_covered moe_n_covered n_total moe_n_total pct_covered
1  17143003000      1378     251.03984    1378   251.32648   1.0000000
2  17143000300       561     195.42262     579   197.92675   0.9689119
3  17143001800       518     188.40913     576   194.36821   0.8993056
4  17143002000       776     149.08052     788   151.02980   0.9847716
5  17073030600       486     163.66429     500   166.82326   0.9720000
6  17073030900       635     203.85779     635   204.21068   1.0000000
7  17019001205      2253     528.51395    2332   534.50070   0.9661235
8  17019000402       313      91.13177     313    91.13177   1.0000000
9  17019001100       809     212.58175     832   213.99299   0.9723558
10 17115001400       483     187.45399     483   187.83770   1.0000000
   moe_pct_covered                       geometry
1       0.25778424 MULTIPOLYGON (((-89.65023 4...
2       0.06491956 MULTIPOLYGON (((-89.63284 4...
3       0.12207485 MULTIPOLYGON (((-89.61315 4...
4       0.01296847 MULTIPOLYGON (((-89.62541 4...
5       0.04439195 MULTIPOLYGON (((-89.97326 4...
6       0.45440644 MULTIPOLYGON (((-89.9504 41...
7       0.04825769 MULTIPOLYGON (((-88.35067 4...
8       0.41175651 MULTIPOLYGON (((-88.24273 4...
```

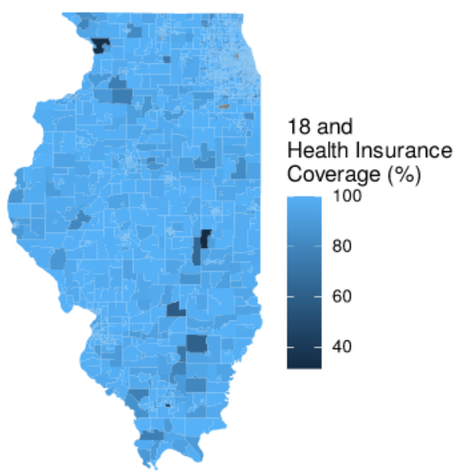
```
9       0.05231928 MULTIPOLYGON (((-88.27677 4...
10      0.54942327 MULTIPOLYGON (((-88.97343 3...
```

Visualization of estimates in R

Making quick visualizations of geospatial data is easy with `ggplot2`, which is included in the tidyverse. However, it lacks the capability to add many of the elements you may be use to out of the box (e.g., scale bars, north arrows, etc.). Additionally, it is not as quite easy to set a classification method as in ArcGIS Pro. There are other packages that can help with these concerns like `tmap` or `ggspatial`, but going all of the R cartography tools could be a whole months long course on its own.

Here we use `geom_sf()` to specify that we want to create a plot based on our spatial data where the fill is equal to the `pct_covered` variable. We also specify the label with `labs()` and set the theme to void to get rid of the background that `ggplot2` typically includes.

```
ggplot(il23_ac5_healthins18below_geom) +
  geom_sf(aes(fill = pct_covered * 100), color = NA) +
  labs(fill = "18 and \nHealth Insurance\nCoverage (%)") +
  theme_void()
```



Visualization estimates in ArcGIS Pro

Since we have the geography, can write a shapefile, geojson, geodatabase, geopackage, or any other major spatial format.

```
library(sf)

st_write(il23_ac5_healthins18below_geom, "il23_ac5_healthins18below.shp")
# or
st_write(il23_ac5_healthins18below_geom, "il23_ac5_healthins18below.geojson")
# or
st_write(
```

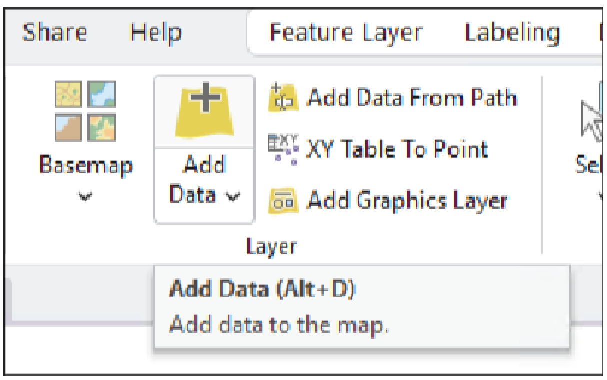
```
  il23_ac5_healthins18below_geom,
  "il23_ac5_healthins18below.gdb",
  "il23_ac5_healthins18below"
)
# or
st_write(
  il23_ac5_healthins18below_geom,
  "il23_ac5_healthins18below.gpkg",
  "il23_ac5_healthins18below"
)
```

Geopackage is a good choice for working within R and ArcGIS Pro because it is open source and supported by both ArcGIS Pro and R. So let's run:

```
st_write(
  il23_ac5_healthins18below_geom,
  "il23_ac5_healthins18below.gpkg",
  "il23_ac5_healthins18below"
)
```

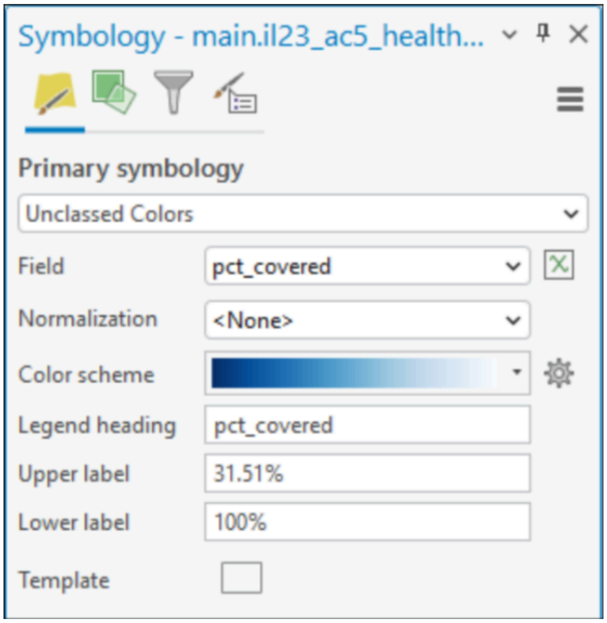
Within ArcGIS Pro, create a new Map Project.

Click on **Add Data** and navigate to where you stored the *il23_ac5_healthins18below.gpkg* file.

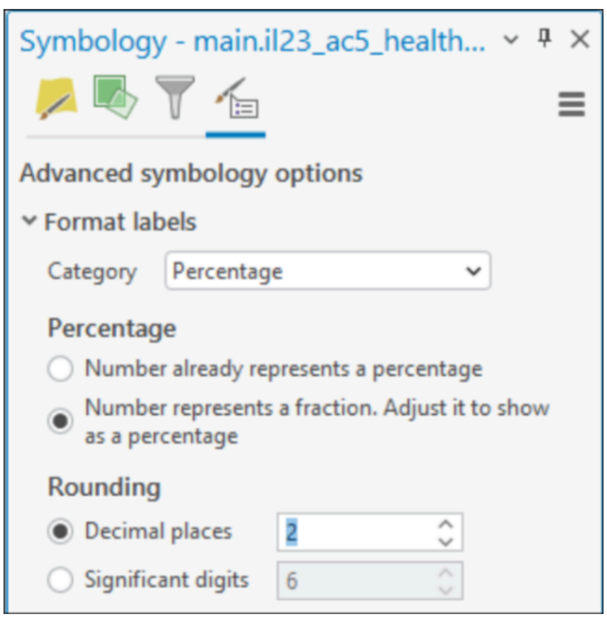


Right click on the *main.il23_ac5_healthins18below* layer within your **Contents** pane and open up **Symbology**.

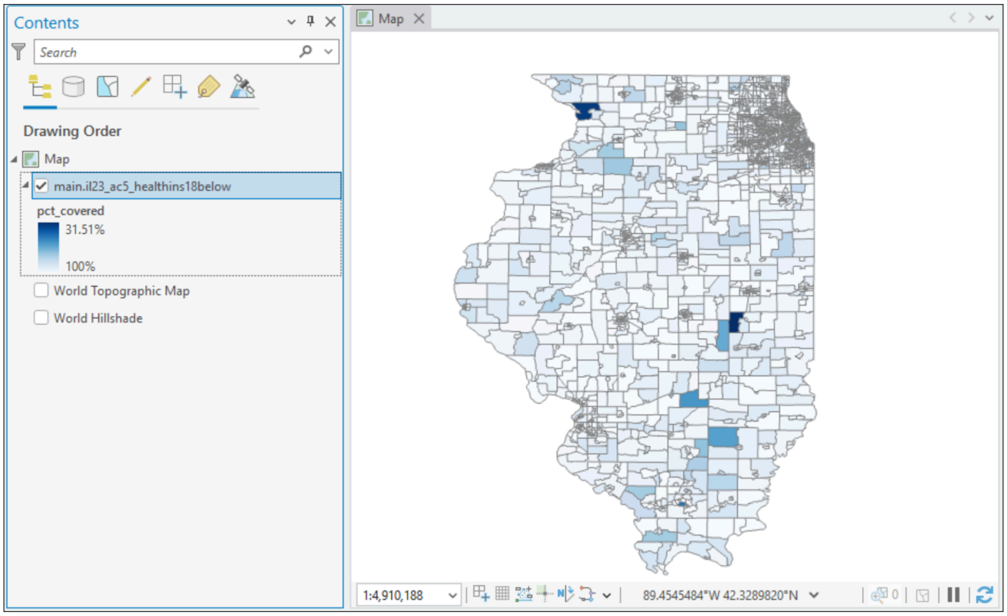
Set the **Primary Symbology** to *Unclassed colors*, **Field** to *pct_covered* and use your preferred color scheme.



Under **Advanced Symbology Options** → **Format labels** you may also want to set **Percentage** as *Number represents a fraction* and **Rounding decimal places** to *2*.



Now you have a map like:



There is so much more you can do to make this map your own and we encourage you to do so! But now you know how to find, retrieve, and visualize Census data!