Chapter 3 Notes

Table of contents

# Chapter 3: Wrangling Census Data

## 3.1 The Tidyverse

* Developed by community, esp. Wickham from RStudio/Posit
* Includes many packages - author emphasizes the following:
  + **readr** for import/export
  + **dplyr** for data wrangling
  + **tidyr** for reshaping data
  + **purrr** for functional programming
  + **ggplot2** data visualization usin the Grammar of Graphics
  + **stringr** string manipulation
  + **forcats** working with factors
* core data structure is the ***tibble*** (a tidy table) which is an enhanced variant of data frames.
  + **tidycensus** returns tibbles by default

## 3.2 Exploring census data with tidyverse tools

First, load the required packages

```{r setup}  
library(tidycensus)  
library(tidyverse)  
```

── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
✔ dplyr 1.1.2 ✔ readr 2.1.4  
✔ forcats 1.0.0 ✔ stringr 1.5.0  
✔ ggplot2 3.4.2 ✔ tibble 3.2.1  
✔ lubridate 1.9.2 ✔ tidyr 1.3.0  
✔ purrr 1.0.2   
── 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

### Sorting and filtering data

Suppose median age data from 2016-2020 ACS

```{r}  
median\_age <- get\_acs(  
 geography = "county",  
 variables = "B01002\_001",  
 year = 2020,  
 cache\_table = TRUE  
)  
```

Getting data from the 2016-2020 5-year ACS

```{r}  
median\_age  
```

# A tibble: 3,221 × 5  
 GEOID NAME variable estimate moe  
 <chr> <chr> <chr> <dbl> <dbl>  
 1 01001 Autauga County, Alabama B01002\_001 38.6 0.6  
 2 01003 Baldwin County, Alabama B01002\_001 43.2 0.4  
 3 01005 Barbour County, Alabama B01002\_001 40.1 0.6  
 4 01007 Bibb County, Alabama B01002\_001 39.9 1.2  
 5 01009 Blount County, Alabama B01002\_001 41 0.5  
 6 01011 Bullock County, Alabama B01002\_001 39.7 1.9  
 7 01013 Butler County, Alabama B01002\_001 41.2 0.6  
 8 01015 Calhoun County, Alabama B01002\_001 39.5 0.4  
 9 01017 Chambers County, Alabama B01002\_001 41.9 0.7  
10 01019 Cherokee County, Alabama B01002\_001 46.8 0.5  
# ℹ 3,211 more rows

By default, printing a **tibble** shows the first 10 rows

**arrange()** will let us sort data (to determine youngest and oldest):

```{r}  
arrange(median\_age, estimate)  
arrange(median\_age, desc(estimate))  
```

# A tibble: 3,221 × 5  
 GEOID NAME variable estimate moe  
 <chr> <chr> <chr> <dbl> <dbl>  
 1 35011 De Baca County, New Mexico B01002\_001 22.2 6.9  
 2 51678 Lexington city, Virginia B01002\_001 22.2 0.8  
 3 16065 Madison County, Idaho B01002\_001 23.5 0.2  
 4 46121 Todd County, South Dakota B01002\_001 23.6 0.6  
 5 51750 Radford city, Virginia B01002\_001 23.7 0.6  
 6 13053 Chattahoochee County, Georgia B01002\_001 24 0.7  
 7 02158 Kusilvak Census Area, Alaska B01002\_001 24.1 0.2  
 8 49049 Utah County, Utah B01002\_001 25 0.1  
 9 46027 Clay County, South Dakota B01002\_001 25.2 0.5  
10 53075 Whitman County, Washington B01002\_001 25.2 0.2  
# ℹ 3,211 more rows  
# A tibble: 3,221 × 5  
 GEOID NAME variable estimate moe  
 <chr> <chr> <chr> <dbl> <dbl>  
 1 12119 Sumter County, Florida B01002\_001 68 0.3  
 2 48301 Loving County, Texas B01002\_001 62.2 37.8  
 3 48243 Jeff Davis County, Texas B01002\_001 61.3 36.8  
 4 08027 Custer County, Colorado B01002\_001 60.1 3.4  
 5 12015 Charlotte County, Florida B01002\_001 59.5 0.2  
 6 51091 Highland County, Virginia B01002\_001 59.5 4.8  
 7 35003 Catron County, New Mexico B01002\_001 59.4 2.6  
 8 51133 Northumberland County, Virginia B01002\_001 59.3 0.7  
 9 26131 Ontonagon County, Michigan B01002\_001 59.1 0.5  
10 48443 Terrell County, Texas B01002\_001 59.1 9.4  
# ℹ 3,211 more rows

**dplyr’s** filter() function can query within datasets (think SQL WHERE clause)

```{r}  
filter(median\_age, estimate >= 50)  
```

# A tibble: 218 × 5  
 GEOID NAME variable estimate moe  
 <chr> <chr> <chr> <dbl> <dbl>  
 1 02105 Hoonah-Angoon Census Area, Alaska B01002\_001 52.1 2.9  
 2 04007 Gila County, Arizona B01002\_001 50.4 0.2  
 3 04012 La Paz County, Arizona B01002\_001 57.4 0.6  
 4 04015 Mohave County, Arizona B01002\_001 52.3 0.2  
 5 04025 Yavapai County, Arizona B01002\_001 54.1 0.2  
 6 05005 Baxter County, Arkansas B01002\_001 52.3 0.5  
 7 05089 Marion County, Arkansas B01002\_001 52.1 0.8  
 8 05097 Montgomery County, Arkansas B01002\_001 50.6 0.8  
 9 05137 Stone County, Arkansas B01002\_001 50 0.5  
10 06009 Calaveras County, California B01002\_001 52.8 0.6  
# ℹ 208 more rows

* arrange() and filter() operate on rows
* separate() from **tidyr** operates on columns

```{r}  
separate(  
 median\_age,  
 NAME,  
 into = c("county", "state"),  
 sep = ", "  
 )  
```

# A tibble: 3,221 × 6  
 GEOID county state variable estimate moe  
 <chr> <chr> <chr> <chr> <dbl> <dbl>  
 1 01001 Autauga County Alabama B01002\_001 38.6 0.6  
 2 01003 Baldwin County Alabama B01002\_001 43.2 0.4  
 3 01005 Barbour County Alabama B01002\_001 40.1 0.6  
 4 01007 Bibb County Alabama B01002\_001 39.9 1.2  
 5 01009 Blount County Alabama B01002\_001 41 0.5  
 6 01011 Bullock County Alabama B01002\_001 39.7 1.9  
 7 01013 Butler County Alabama B01002\_001 41.2 0.6  
 8 01015 Calhoun County Alabama B01002\_001 39.5 0.4  
 9 01017 Chambers County Alabama B01002\_001 41.9 0.7  
10 01019 Cherokee County Alabama B01002\_001 46.8 0.5  
# ℹ 3,211 more rows

**NOTE:** “Many tidyverse functions use *non-standard evaluation* to refer to column names”

* This means you don’t necessarily need to use quotes to references column names
  + This can make it difficult when you’re writing your own functions, however.

### Using summary variables and calculating new columns

* Data in Census and ACS tables are comprised of variables that that constitute sub-categories
  + i.e. numbers of households in different household income bands
* This data returns estimated counts
  + difficult to compare across geographies without normalizing
* So we **normalize** via dividing by an overall population that the sub-group is derived from.
  + Normalizing variables are frequently in ACS tables themselves.
* in ACS table B19001 (Household Income), variable B19001\_001 represents total number of households in enumeration unit
* get\_acs() and get\_decennial() both accept arguments for the parameter summary\_var which creates columns for a summary variable useful for normalization.
* ***Example: normalizing race and Hispanic origin against base population, Arizona***

```{r}  
race\_vars <- c(  
 White = "B03002\_003",  
 Black = "B03002\_004",  
 Native = "B03002\_005",  
 Asian = "B03002\_006",  
 HIPI = "B03002\_007",  
 Hispanic = "B03002\_012"  
)  
  
az\_race <- get\_acs(  
 geography = "county",  
 state = "AZ",  
 variables = race\_vars,  
 summary\_var = "B03002\_001", # total pop  
 year = 2020,  
 cache\_table = TRUE  
)  
```

Getting data from the 2016-2020 5-year ACS

```{r}  
az\_race  
```

# A tibble: 90 × 7  
 GEOID NAME variable estimate moe summary\_est summary\_moe  
 <chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl>  
 1 04001 Apache County, Arizona White 12993 56 71714 NA  
 2 04001 Apache County, Arizona Black 544 56 71714 NA  
 3 04001 Apache County, Arizona Native 51979 327 71714 NA  
 4 04001 Apache County, Arizona Asian 262 76 71714 NA  
 5 04001 Apache County, Arizona HIPI 49 14 71714 NA  
 6 04001 Apache County, Arizona Hispanic 4751 NA 71714 NA  
 7 04003 Cochise County, Arizona White 69095 350 126442 NA  
 8 04003 Cochise County, Arizona Black 4512 378 126442 NA  
 9 04003 Cochise County, Arizona Native 1058 176 126442 NA  
10 04003 Cochise County, Arizona Asian 2371 241 126442 NA  
# ℹ 80 more rows

* We may then use **dplyr’s** mutate() function to calculate a percent column
  + in the below example, we use select() to retain only the columns we want

***Example: Normalized demo data for Arizona***

```{r}  
az\_race\_percent <- az\_race %>%   
 mutate(percent = 100 \* (estimate / summary\_est)) %>%   
 select(NAME, variable, percent)  
  
az\_race\_percent  
```

# A tibble: 90 × 3  
 NAME variable percent  
 <chr> <chr> <dbl>  
 1 Apache County, Arizona White 18.1   
 2 Apache County, Arizona Black 0.759   
 3 Apache County, Arizona Native 72.5   
 4 Apache County, Arizona Asian 0.365   
 5 Apache County, Arizona HIPI 0.0683  
 6 Apache County, Arizona Hispanic 6.62   
 7 Cochise County, Arizona White 54.6   
 8 Cochise County, Arizona Black 3.57   
 9 Cochise County, Arizona Native 0.837   
10 Cochise County, Arizona Asian 1.88   
# ℹ 80 more rows

## 3.3 Group-wise Census data analysis

* Author advocates for split-apply-combine data analysis, drawn from Wickham 2011
  + Identify groups in dataset for comparisons; split into one piece per group
  + apply function to each group (summarizing function, such as mean or maximum, etc)
  + combine back into dataset for comparison between groups
* The tidyverse implements split-apply-combine through group\_by() from **dplyr**

### ***Making Group-wise comparisons***

* Use az\_race\_percent created above
  + two columns for group definitions:
    - NAME representing county
    - variable representing racial or ethnic group
  + Identify largest racial or ethnic group in each county
    - Create new dataset largest\_group by:
      1. taking az\_race\_percent, THEN
      2. group\_by() NAME, THEN
      3. filter() by percent == max(percent))

```{r}  
largest\_group <- az\_race\_percent %>%   
 group\_by(NAME) %>%   
 filter(percent == max(percent))  
  
largest\_group  
```

# A tibble: 15 × 3  
# Groups: NAME [15]  
 NAME variable percent  
 <chr> <chr> <dbl>  
 1 Apache County, Arizona Native 72.5  
 2 Cochise County, Arizona White 54.6  
 3 Coconino County, Arizona White 53.8  
 4 Gila County, Arizona White 61.9  
 5 Graham County, Arizona White 50.9  
 6 Greenlee County, Arizona Hispanic 47.3  
 7 La Paz County, Arizona White 57.2  
 8 Maricopa County, Arizona White 54.6  
 9 Mohave County, Arizona White 76.7  
10 Navajo County, Arizona Native 42.7  
11 Pima County, Arizona White 51.1  
12 Pinal County, Arizona White 56.2  
13 Santa Cruz County, Arizona Hispanic 83.3  
14 Yavapai County, Arizona White 80.1  
15 Yuma County, Arizona Hispanic 64.1

* group\_by() frequently paired with summarize()

***Example: Median percentage for each racial/ethnic group across AZ counties***

```{r}  
az\_race\_percent %>%   
 group\_by(variable) %>%   
 summarize(median\_pct = median(percent))  
```

# A tibble: 6 × 2  
 variable median\_pct  
 <chr> <dbl>  
1 Asian 0.992  
2 Black 1.29   
3 HIPI 0.107  
4 Hispanic 30.5   
5 Native 3.63   
6 White 53.8

### Tabulating new groups

Analysts may calculate new custom groups to address questions.

Example: ACS table B19001 has households with hh income bucketed in different increments (bottom bucket is < $10K; top bucket is >=200K )

```{r}  
mn\_hh\_income <- get\_acs(  
 geography = "county",  
 table = "B19001",  
 state = "MN",  
 year = 2016,  
 cache\_table = TRUE  
)  
```

Getting data from the 2012-2016 5-year ACS

Loading ACS5 variables for 2016 from table B19001 and caching the dataset for faster future access.

```{r}  
mn\_hh\_income  
```

# A tibble: 1,479 × 5  
 GEOID NAME variable estimate moe  
 <chr> <chr> <chr> <dbl> <dbl>  
 1 27001 Aitkin County, Minnesota B19001\_001 7640 262  
 2 27001 Aitkin County, Minnesota B19001\_002 562 77  
 3 27001 Aitkin County, Minnesota B19001\_003 544 72  
 4 27001 Aitkin County, Minnesota B19001\_004 472 69  
 5 27001 Aitkin County, Minnesota B19001\_005 508 68  
 6 27001 Aitkin County, Minnesota B19001\_006 522 92  
 7 27001 Aitkin County, Minnesota B19001\_007 447 61  
 8 27001 Aitkin County, Minnesota B19001\_008 390 49  
 9 27001 Aitkin County, Minnesota B19001\_009 426 64  
10 27001 Aitkin County, Minnesota B19001\_010 415 65  
# ℹ 1,469 more rows

We want to make three income buckets: lt 35K, 35-75K, and 75K+

* first we transform the data set; remove B19001\_001 variable, or total count of households per county
* use case\_when() to identify groups of variables for binning
* case\_when() is used inside mutate() to create incgroup variable
  + The first condition is evaluated, and assigns the value below35k to all rows where variable < "B19001\_008"
    - That is, B19001\_002 (income less than $10k) to B19001\_007 (income between 30K and 34,999).
  + The second condition is evaluated, and assigns the value bw35kand75k to all remaining rows where variables < "B19001\_013"
    - That is, B19001\_008 (income between 35K and 39,999) to B19001\_012 (income between 70K and 74,999)
  + The final condition takes “all other values” (represented by TRUE) and assigns the value above75k

```{r}  
mn\_hh\_income\_recode <- mn\_hh\_income %>%   
 filter(variable != "B19001\_001") %>% # return all values EXCEPT "B19001\_001"  
 mutate(incgroup = case\_when(  
 variable < "B19001\_008" ~ "below35k", # find variable < "B19001\_008", assign it "below35k"  
 variable < "B19001\_013" ~ "bw35kand75k",  
 TRUE ~ "above75k"  
 ))  
  
mn\_hh\_income\_recode  
```

# A tibble: 1,392 × 6  
 GEOID NAME variable estimate moe incgroup   
 <chr> <chr> <chr> <dbl> <dbl> <chr>   
 1 27001 Aitkin County, Minnesota B19001\_002 562 77 below35k   
 2 27001 Aitkin County, Minnesota B19001\_003 544 72 below35k   
 3 27001 Aitkin County, Minnesota B19001\_004 472 69 below35k   
 4 27001 Aitkin County, Minnesota B19001\_005 508 68 below35k   
 5 27001 Aitkin County, Minnesota B19001\_006 522 92 below35k   
 6 27001 Aitkin County, Minnesota B19001\_007 447 61 below35k   
 7 27001 Aitkin County, Minnesota B19001\_008 390 49 bw35kand75k  
 8 27001 Aitkin County, Minnesota B19001\_009 426 64 bw35kand75k  
 9 27001 Aitkin County, Minnesota B19001\_010 415 65 bw35kand75k  
10 27001 Aitkin County, Minnesota B19001\_011 706 81 bw35kand75k  
# ℹ 1,382 more rows

* We may now do group-wise comparisons with the new groups

```{r}  
mn\_group\_sums <- mn\_hh\_income\_recode %>%   
 group\_by(GEOID, incgroup) %>%   
 summarize(estimate = sum(estimate))  
```

`summarise()` has grouped output by 'GEOID'. You can override using the  
`.groups` argument.

```{r}  
mn\_group\_sums  
```

# A tibble: 261 × 3  
# Groups: GEOID [87]  
 GEOID incgroup estimate  
 <chr> <chr> <dbl>  
 1 27001 above75k 1706  
 2 27001 below35k 3055  
 3 27001 bw35kand75k 2879  
 4 27003 above75k 61403  
 5 27003 below35k 24546  
 6 27003 bw35kand75k 39311  
 7 27005 above75k 4390  
 8 27005 below35k 4528  
 9 27005 bw35kand75k 4577  
10 27007 above75k 4491  
# ℹ 251 more rows

## 3.4 Comparing ACS estimates over time

Census data through the API only goes back to 2000. NHGIS goes back to 1790; covered later in book

### Time Series Analysis cautions

* Geography changes over time.

***Example: Oglala Lakota County, South Dakota***

* We get the 2020 data for Oglala Lakota County

```{r}  
oglala\_lakota\_age <- get\_acs(  
 geography = "county",  
 state = "SD",  
 county = "Oglala Lakota",  
 table = "B01001",  
 year = 2020,  
 cache\_table = TRUE  
)  
```

Getting data from the 2016-2020 5-year ACS

Loading ACS5 variables for 2020 from table B01001 and caching the dataset for faster future access.

```{r}  
oglala\_lakota\_age  
```

# A tibble: 49 × 5  
 GEOID NAME variable estimate moe  
 <chr> <chr> <chr> <dbl> <dbl>  
 1 46102 Oglala Lakota County, South Dakota B01001\_001 14277 NA  
 2 46102 Oglala Lakota County, South Dakota B01001\_002 6930 132  
 3 46102 Oglala Lakota County, South Dakota B01001\_003 761 66  
 4 46102 Oglala Lakota County, South Dakota B01001\_004 794 128  
 5 46102 Oglala Lakota County, South Dakota B01001\_005 707 123  
 6 46102 Oglala Lakota County, South Dakota B01001\_006 394 20  
 7 46102 Oglala Lakota County, South Dakota B01001\_007 227 15  
 8 46102 Oglala Lakota County, South Dakota B01001\_008 85 53  
 9 46102 Oglala Lakota County, South Dakota B01001\_009 165 70  
10 46102 Oglala Lakota County, South Dakota B01001\_010 356 101  
# ℹ 39 more rows

* We want to look at change over time, so we try to pull in 2010

```{r}  
oglala\_lakota\_age\_10 <- get\_acs(  
 geography = "county",  
 state = "SD",  
 county = "Oglala Lakota",  
 table = "B01001",  
 year = 2010,  
 cache\_table = TRUE  
)  
```

Getting data from the 2006-2010 5-year ACS

Loading ACS5 variables for 2010 from table B01001 and caching the dataset for faster future access.

No encoding supplied: defaulting to UTF-8.

Error in `map()`:  
ℹ In index: 1.  
ℹ With name: 1.  
Caused by error:  
! Your API call has errors. The API message returned is .

```{r}  
oglala\_lakota\_age\_10  
```

Error in eval(expr, envir, enclos): object 'oglala\_lakota\_age\_10' not found

* An error is returned, because Oglala Lakota County had a different name in 2010 - Shannon County.

```{r}  
oglala\_lakota\_age\_10 <- get\_acs(  
 geography = "county",  
 state = "SD",  
 county = "Shannon",  
 table = "B01001",  
 year = 2010,  
 cache\_table = TRUE  
)  
```

Getting data from the 2006-2010 5-year ACS

Loading ACS5 variables for 2010 from table B01001 and caching the dataset for faster future access.

```{r}  
oglala\_lakota\_age\_10  
```

# A tibble: 49 × 5  
 GEOID NAME variable estimate moe  
 <chr> <chr> <chr> <dbl> <dbl>  
 1 46113 Shannon County, South Dakota B01001\_001 13437 NA  
 2 46113 Shannon County, South Dakota B01001\_002 6553 47  
 3 46113 Shannon County, South Dakota B01001\_003 770 99  
 4 46113 Shannon County, South Dakota B01001\_004 565 151  
 5 46113 Shannon County, South Dakota B01001\_005 833 151  
 6 46113 Shannon County, South Dakota B01001\_006 541 47  
 7 46113 Shannon County, South Dakota B01001\_007 275 99  
 8 46113 Shannon County, South Dakota B01001\_008 164 89  
 9 46113 Shannon County, South Dakota B01001\_009 143 54  
10 46113 Shannon County, South Dakota B01001\_010 342 83  
# ℹ 39 more rows

* The GEOID is different, as well - when a geographic entity changes its name, Census Bureau assigns a new GEOID.
  + [The Census updates Table and Geography Changes annually](https://www.census.gov/programs-surveys/acs/technical-documentation/table-and-geography-changes.html)
* Variable IDs can change as well
  + We look for residents 25+ age, with 4-year degrees or higher from 2019 ( "DP02\_0068P", here)

```{r}  
co\_college19 <- get\_acs(  
 geography = "county",  
 variables = "DP02\_0068P",  
 state = "CO",  
 survey = "acs1",  
 year = 2019  
)  
```

Getting data from the 2019 1-year ACS

The 1-year ACS provides data for geographies with populations of 65,000 and greater.

Using the ACS Data Profile

```{r}  
co\_college19  
```

# A tibble: 12 × 5  
 GEOID NAME variable estimate moe  
 <chr> <chr> <chr> <dbl> <dbl>  
 1 08001 Adams County, Colorado DP02\_0068P 25.4 1.3  
 2 08005 Arapahoe County, Colorado DP02\_0068P 43.8 1.3  
 3 08013 Boulder County, Colorado DP02\_0068P 64.8 1.8  
 4 08014 Broomfield County, Colorado DP02\_0068P 56.9 3.3  
 5 08031 Denver County, Colorado DP02\_0068P 53.1 1.1  
 6 08035 Douglas County, Colorado DP02\_0068P 58.1 1.8  
 7 08041 El Paso County, Colorado DP02\_0068P 39 1.3  
 8 08059 Jefferson County, Colorado DP02\_0068P 47.6 1.2  
 9 08069 Larimer County, Colorado DP02\_0068P 49 1.8  
10 08077 Mesa County, Colorado DP02\_0068P 29.8 2.6  
11 08101 Pueblo County, Colorado DP02\_0068P 23.4 2   
12 08123 Weld County, Colorado DP02\_0068P 29.9 1.6

* We try the same query for 2018

```{r}  
co\_college18 <- get\_acs(  
 geography = "county",  
 variables = "DP02\_0068P",  
 state = "CO",  
 survey = "acs1",  
 year = 2018  
)  
```

Getting data from the 2018 1-year ACS

The 1-year ACS provides data for geographies with populations of 65,000 and greater.

Using the ACS Data Profile

```{r}  
co\_college18  
```

# A tibble: 12 × 5  
 GEOID NAME variable estimate moe  
 <chr> <chr> <chr> <dbl> <dbl>  
 1 08001 Adams County, Colorado DP02\_0068P 375798 NA  
 2 08005 Arapahoe County, Colorado DP02\_0068P 497198 NA  
 3 08013 Boulder County, Colorado DP02\_0068P 263938 NA  
 4 08014 Broomfield County, Colorado DP02\_0068P 53400 NA  
 5 08031 Denver County, Colorado DP02\_0068P 575870 NA  
 6 08035 Douglas County, Colorado DP02\_0068P 252922 NA  
 7 08041 El Paso County, Colorado DP02\_0068P 513528 NA  
 8 08059 Jefferson County, Colorado DP02\_0068P 465615 NA  
 9 08069 Larimer County, Colorado DP02\_0068P 281086 NA  
10 08077 Mesa County, Colorado DP02\_0068P 119498 NA  
11 08101 Pueblo County, Colorado DP02\_0068P 129899 NA  
12 08123 Weld County, Colorado DP02\_0068P 231613 NA

* The values are VERY different, and not percentages.
  + variable IDs in acs Data Profile are *unique each year* and so should not be used for time-series analysis.

### Preparing time-series ACS estimates

* safest option: use Comparison Profile Tables (1 year or 5 year)
  + allows for comparison of demo indicators over past 5 years for year given.
  + also has more variable harmonization - inflation-adjustments, etc.

Example for accessing ACS Comparison Profile:

```{r}  
ak\_income\_compare <- get\_acs(  
 geography = "county",  
 variables = c(  
 income15 = "CP03\_2015\_062",  
 income20 = "CP03\_2020\_062"  
 ),  
 state = "AK",  
 year = 2020  
)  
```

Getting data from the 2016-2020 5-year ACS

Using the ACS Comparison Profile

```{r}  
ak\_income\_compare  
```

# A tibble: 34 × 4  
 GEOID NAME variable estimate  
 <chr> <chr> <chr> <dbl>  
 1 02016 Aleutians West Census Area, Alaska income15 92500  
 2 02016 Aleutians West Census Area, Alaska income20 87443  
 3 02020 Anchorage Municipality, Alaska income15 85534  
 4 02020 Anchorage Municipality, Alaska income20 84813  
 5 02050 Bethel Census Area, Alaska income15 55692  
 6 02050 Bethel Census Area, Alaska income20 54400  
 7 02090 Fairbanks North Star Borough, Alaska income15 77590  
 8 02090 Fairbanks North Star Borough, Alaska income20 76464  
 9 02110 Juneau City and Borough, Alaska income15 93836  
10 02110 Juneau City and Borough, Alaska income20 88077  
# ℹ 24 more rows

* from 2016-2020 ACS used for 2020, comparison year is 2015, which uses 2011-2015.

#### iterating over ACS years with tidyverse tools

* using Detailed Tables is also a safer option - variable IDs remain consistent across years
  + pitfalls that still exist; variables may be removed or added from survey to survey.
  + Always check on data availability using load\_variables() for planned years

Revisit Colorado Bachelor Degrees:

* in Detailed Tables, “bachelor’s degree or higher” is partitioned by sex and by attainment tiers in ACS table 15002
* We only need variables for populations 25+ with a 4 year or graduate degrees, by sex)
  + We’ll pull only the variables we need

```{r}  
college\_vars <- c("B15002\_015",  
 "B15002\_016",  
 "B15002\_017",  
 "B15002\_018",  
 "B15002\_032",  
 "B15002\_033",  
 "B15002\_034",  
 "B15002\_035")  
```

* We’ll perform iteration using purrr’s map\_\*() functions
  + map() returns a list
  + map\_int() returns an integer vector
  + map\_chr() returns a character vector
  + map\_dfr() iterates over an input, passes it to a function/process defined by the user, and row-binds the result to a data frame.

```{r}  
years <- 2010:2019 # create year vector  
names(years) <- years # set vector names to vector values  
  
college\_by\_year <- map\_dfr(years,  
 ~{  
 get\_acs(  
 geography = "county",  
 variables = college\_vars,  
 state = "CO",  
 summary\_var = "B15002\_001",  
 survey = "acs1",  
 year = .x)},  
 .id = "year")  
```

Getting data from the 2010 1-year ACS

The 1-year ACS provides data for geographies with populations of 65,000 and greater.

Getting data from the 2011 1-year ACS

The 1-year ACS provides data for geographies with populations of 65,000 and greater.

Getting data from the 2012 1-year ACS

The 1-year ACS provides data for geographies with populations of 65,000 and greater.

Getting data from the 2013 1-year ACS

The 1-year ACS provides data for geographies with populations of 65,000 and greater.

Getting data from the 2014 1-year ACS

The 1-year ACS provides data for geographies with populations of 65,000 and greater.

Getting data from the 2015 1-year ACS

The 1-year ACS provides data for geographies with populations of 65,000 and greater.

Getting data from the 2016 1-year ACS

The 1-year ACS provides data for geographies with populations of 65,000 and greater.

Getting data from the 2017 1-year ACS

The 1-year ACS provides data for geographies with populations of 65,000 and greater.

Getting data from the 2018 1-year ACS

The 1-year ACS provides data for geographies with populations of 65,000 and greater.

Getting data from the 2019 1-year ACS

The 1-year ACS provides data for geographies with populations of 65,000 and greater.

* first argument, object to be iterated over
* second argument, formula (specified with ~ and enclosed in {} to be ran once for each element
  + .x is local variable; it is passed the element from the object in the first argument
  + once each run is performed, the result is combined in a single output dataframe
* third argument (optional), .id creates new column in data frame containing names from object in first argument

We can review the result like so:

```{r}  
college\_by\_year %>%   
 arrange(NAME, variable, year)  
```

# A tibble: 920 × 8  
 year GEOID NAME variable estimate moe summary\_est summary\_moe  
 <chr> <chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl>  
 1 2010 08001 Adams County, Co… B15002\_… 20501 1983 275849 790  
 2 2011 08001 Adams County, Co… B15002\_… 21233 2124 281231 865  
 3 2012 08001 Adams County, Co… B15002\_… 19238 2020 287924 693  
 4 2013 08001 Adams County, Co… B15002\_… 23818 2445 295122 673  
 5 2014 08001 Adams County, Co… B15002\_… 20255 1928 304394 541  
 6 2015 08001 Adams County, Co… B15002\_… 22962 2018 312281 705  
 7 2016 08001 Adams County, Co… B15002\_… 25744 2149 318077 525  
 8 2017 08001 Adams County, Co… B15002\_… 26159 2320 324185 562  
 9 2018 08001 Adams County, Co… B15002\_… 28113 2078 331247 955  
10 2019 08001 Adams County, Co… B15002\_… 27552 2070 336931 705  
# ℹ 910 more rows

* note that table is in long/tidy form; we can pivot it to a wide table more suitable for display or interpretation in conventional forms

```{r}  
college\_by\_year %>%   
 group\_by(NAME, year) %>%   
 summarize(estimate = sum(estimate)) %>%  
 pivot\_wider(id\_cols = NAME,  
 names\_from = year,  
 values\_from = estimate)  
```

`summarise()` has grouped output by 'NAME'. You can override using the  
`.groups` argument.

# A tibble: 12 × 11  
# Groups: NAME [12]  
 NAME `2010` `2011` `2012` `2013` `2014` `2015` `2016` `2017` `2018` `2019`  
 <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
 1 Adams … 56753 57703 59443 68155 67482 71192 73008 74159 85268 85579  
 2 Arapah… 140056 147777 155755 159530 169464 173750 178155 190774 190035 197697  
 3 Boulde… 109145 113622 113657 117131 117732 124859 127679 134092 135132 141110  
 4 Broomf… NA NA NA NA NA 25212 23561 26001 27729 28176  
 5 Denver… 168536 182644 195576 201961 207030 227925 234447 250415 266782 281723  
 6 Dougla… 101704 100495 106182 113513 114411 117353 128220 129888 133351 136812  
 7 El Pas… 134813 143655 142748 146990 153902 156557 170871 177264 180268 184045  
 8 Jeffer… 151508 148930 158435 159218 165533 173775 178031 189181 192668 202777  
 9 Larime… 88532 84543 90126 88787 89525 100339 103892 108482 110808 116752  
10 Mesa C… 24473 25531 22854 27447 25148 30276 25198 26653 31357 31974  
11 Pueblo… 20569 25227 21252 23168 25482 23886 23759 25141 25035 27254  
12 Weld C… 38864 39499 43007 46275 45114 46781 51217 53856 55447 62942

* Expanding on this, we can generate further wide tables of new variables, such as below where we compute a table with the (estimate) percentage of college educated population per county for each year.

```{r}  
percent\_college\_by\_year <- college\_by\_year %>%   
 group\_by(NAME, year) %>%   
 summarize(numerator = sum(estimate),  
 denominator = first(summary\_est)) %>%   
 mutate(pct\_college = 100 \* (numerator / denominator)) %>%   
 pivot\_wider(id\_cols = NAME,  
 names\_from = year,  
 values\_from = pct\_college)  
```

`summarise()` has grouped output by 'NAME'. You can override using the  
`.groups` argument.

```{r}  
percent\_college\_by\_year  
```

# A tibble: 12 × 11  
# Groups: NAME [12]  
 NAME `2010` `2011` `2012` `2013` `2014` `2015` `2016` `2017` `2018` `2019`  
 <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
 1 Adams … 20.6 20.5 20.6 23.1 22.2 22.8 23.0 22.9 25.7 25.4  
 2 Arapah… 37.0 38.2 39.3 39.4 40.9 41.0 41.5 43.7 42.7 43.8  
 3 Boulde… 57.5 59.1 57.9 58.5 58.0 60.6 60.6 63.2 62.5 64.8  
 4 Broomf… NA NA NA NA NA 56.1 51.9 55.1 56.3 56.9  
 5 Denver… 40.9 43.0 44.7 44.4 44.3 47.1 47.4 49.3 51.3 53.1  
 6 Dougla… 55.0 53.3 55.1 57.7 56.5 56.1 59.4 58.5 58.4 58.1  
 7 El Pas… 34.1 35.7 34.9 35.5 36.5 36.4 38.7 39.2 38.8 39.0  
 8 Jeffer… 40.8 39.5 41.4 41.0 42.0 43.2 43.5 45.6 45.8 47.6  
 9 Larime… 45.8 42.8 44.7 43.3 42.7 46.2 46.8 47.9 47.6 49.0  
10 Mesa C… 25.0 25.8 23.0 27.6 25.1 30.3 25.0 25.8 30.0 29.8  
11 Pueblo… 19.5 23.5 19.9 21.3 23.6 21.6 21.2 22.2 21.8 23.4  
12 Weld C… 25.1 24.7 26.3 27.4 25.7 25.9 27.2 27.5 27.4 29.9

## 3.5 Handling Margins of Error in ACS

* MOEs are vital to account for when using ACS data
  + ACS is not a full population census - rather, estimates are generated from a sample
  + By default, Margins of Error are returned for 90% Confidence Level (CL)
    - A 90% CL means “we are 90 percent sure that the true value falls within a range defined by the estimate plus or minus the margin of error.”
  + **tidycensus** will always return margin of error associated with the estimate
  + MOE CLs can be controlled via moe\_level argument in get\_acs()
    - MOE CL arguments are 90 (default) 95 and 99, computed by Census Bureau formulas

***Example: median household income by county, Rhode Island, 2020***

The first call returns the 90% CI; the second call returns the 99% CI

* Note that stricter (higher) CIs result in larger margins of error

```{r RI-moe90}  
get\_acs(geography = "county",  
 state = "Rhode Island",  
 variables = "B19013\_001",  
 year = 2020  
)  
```

Getting data from the 2016-2020 5-year ACS

```{r RI-moe90}  
get\_acs(geography = "county",  
 state = "Rhode Island",  
 variables = "B19013\_001",  
 year = 2020,  
 moe\_level = 99)  
```

Getting data from the 2016-2020 5-year ACS

# A tibble: 5 × 5  
 GEOID NAME variable estimate moe  
 <chr> <chr> <chr> <dbl> <dbl>  
1 44001 Bristol County, Rhode Island B19013\_001 85413 6122  
2 44003 Kent County, Rhode Island B19013\_001 75857 2022  
3 44005 Newport County, Rhode Island B19013\_001 84282 2629  
4 44007 Providence County, Rhode Island B19013\_001 62323 1270  
5 44009 Washington County, Rhode Island B19013\_001 86970 3651  
# A tibble: 5 × 5  
 GEOID NAME variable estimate moe  
 <chr> <chr> <chr> <dbl> <dbl>  
1 44001 Bristol County, Rhode Island B19013\_001 85413 9587.  
2 44003 Kent County, Rhode Island B19013\_001 75857 3166.  
3 44005 Newport County, Rhode Island B19013\_001 84282 4117.  
4 44007 Providence County, Rhode Island B19013\_001 62323 1989.  
5 44009 Washington County, Rhode Island B19013\_001 86970 5717.

### Calculating derived margins of error

* Smaller enumeration units or smaller populations tend to have larger margins of error
  + this may include MOEs that are larger than their respective estimates

***Example: age groups by sex for 65+ pop, Census Tracts, Salt Lake County, Utah***

Steps are as follows:

1. Generate a vector of variable IDs, named vars
   1. create two vectors of integers ranging 20-25 (first argument in c(), corresponds to Male variables), and from 44-49 (second argument in c(), corresponds to Female variables)
   2. c() merges these two vectors into a single integer vector
   3. Take the string prefix (for these variables, "B01001\_0") and iteratively append the integers to the string using paste0() - concatenation without spaces

```{r}  
vars <- paste0("B01001\_0", c(20:25, 44:49))  
vars  
```

[1] "B01001\_020" "B01001\_021" "B01001\_022" "B01001\_023" "B01001\_024"  
 [6] "B01001\_025" "B01001\_044" "B01001\_045" "B01001\_046" "B01001\_047"  
[11] "B01001\_048" "B01001\_049"

1. use get\_acs() to get our Salt Lake County census tracts with data

```{r}  
salt\_lake <- get\_acs(  
 geography = "tract",  
 variables = vars,  
 state = "Utah",  
 county = "Salt Lake",  
 year = 2020  
)  
```

Getting data from the 2016-2020 5-year ACS

1. Examine estimate and error in a specific Census tract

```{r}  
example\_tract <- salt\_lake %>%   
 filter(GEOID == "49035100100")  
  
example\_tract %>%   
 select(-NAME)  
```

# A tibble: 12 × 4  
 GEOID variable estimate moe  
 <chr> <chr> <dbl> <dbl>  
 1 49035100100 B01001\_020 11 13  
 2 49035100100 B01001\_021 25 18  
 3 49035100100 B01001\_022 7 10  
 4 49035100100 B01001\_023 4 7  
 5 49035100100 B01001\_024 0 12  
 6 49035100100 B01001\_025 17 20  
 7 49035100100 B01001\_044 0 12  
 8 49035100100 B01001\_045 4 7  
 9 49035100100 B01001\_046 21 17  
10 49035100100 B01001\_047 123 168  
11 49035100100 B01001\_048 17 21  
12 49035100100 B01001\_049 0 12

* Note how many variables have estimates < moe
  + Remember, the MOE value presented is a plus or minus value -
    - For the first value, Males between 65 and 66 years of age, the estimate is 11. the MOE is (plus or minus) 13.
    - This means that “we are 90 percent sure that the true population of Males between the age of 65 and 66 years of age in this Census Tract falls within a range between -2 and 24, with the estimate of 11”
    - This is problematic, because we can’t have -2 65-66 year old people. ESRI’s reading would have us cut off at 0; no negative numbers.

NOTE: Colorado State Demography Office published a set of guidelines titled [Margins of Error and their Size](https://drive.google.com/uc?export=download&id=0B2oqdPZKJqK7bC1hYUxPNVVmRnM) which establishes the following guidelines:

* Always consider the context and how the estimate will be used,
* Use with caution when the MOE is 20% to 50% of the estimate,
* If the MOE is larger than 50% of the estimate consider the range created by the MOE. In many cases a MOE larger than 50% of the estimate makes the estimate not usable or relevant. Other times a large MOE wont impact what the estimate is stating.
* All this to say, working with ACS data for small enumeration units is difficult and has pitfalls
* We may aggregate data to a suitable MOE
  + **tidycensus** has the following functions to use Census Bureau formulas for derived estimates:
    - moe\_sum(): moe for a derived sum
    - moe\_product(): moe for a derived product
    - moe\_ratio(): moe for a derived ratio
    - moe\_prop(): moe for a derived proportion

***Example: hypothetical derived proportion MOE***

* We have an ACS variable (per conversation with author on github, “if there were an estimate of 25 people in a Census tract aged over 25 with a bachelor’s degree (with a margin of error of 5 around the estimate)”
* We have the ACS total for that Census tract (“100 total people aged over 25 in that Census tract (with a margin of error of 3 around that estimate)”
* The derived proportion would be 25/100 = 0.25
* We can determine the MOE for the derived proportion of 0.25 via moe\_prop() as such:

```{r}  
moe\_prop(25, 100, 5, 3)  
```

[1] 0.0494343

* we have a derived estimate of 25% of the people in this hypothetical census tract having a Bachelor’s Degree, with a Margin of Error of 4.943%

### Calculating Group-wise margins of error