Working With Data

2023-03-22

Introduction

First, create a folder structure somewhere logical on your computer. Name it something like DSSG Working With Data. Inside this folder, have a folder called "Raw data", and a folder called "Exports". Download the data files from Canvas into "Raw data" and there they shall stay, no touching them! Any altered form of your raw data will go into "Exports".

Some basics: in R, to assign a value to a variable, you use something that looks like an arrow: `<-`.

```r
x <- 6
#This is a comment, a way to write text even in code blocks. Put your cursor in a line of code and pres
y <- 4
#Since you've assigned values to variables, you can do operations on them:
x+y
```

```
## [1] 10
```

When you're programming, you must document and describe everything you do. You might think you'll remember your thinking next time you open your code, but you'd be amazed at how quickly you have no idea what you did. If you're using an R script, you'll use comments to document. If you make a markdown file like this one, you can put your commentary around the code blocks as well as through comments.

#First function

What if you don't want to assign a single value to a variable, but a series? The `c()` function is very commonly used in R and is an easy way to combine values together.

```r
#You can combine numbers together using c() like this:
x <- c(1,2,3)
#Or a sequence of numbers
x <- c(1:5)
#You can combine different data types and even do simple operations in c()
x <- c("cat", 1:10, x+y)
x #displays in console
```

```
##  [1] "cat" "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "5"
## [13] "6"   "7"   "8"   "9"
```

```r
#Overwrite
x <- c(x, "another cat")
```

What if you forget what I'm telling you now about how `c()` works? Never fear, you can either click on the help tab to the right and type in `c()`, or you can type `?c()` in the console. This documentation is terse and rather hard to understand, but also provides some examples, which can be helpful. Otherwise, google away! So many people have made tutorials or asked questions about R in StackOverflow and other places.

These objects we've made are rather silly and we don't need them anymore. Let's clear them away. Click on the little broom icon in the Environment tab on the upper right.

##Installing Packages

Now let's install the packages we need for our census project today. R has a base set of functions you can call on and use – and devoted users or groups of users have an idea of some other functionality they'd like and build a package with a library of functions, usually around a certain theme or use case. These libraries are called packages. Let's install the very best one: TIDYVERSE! https://www.tidyverse.org/

In the lower right box where c() documentation currently is, click on the packages tab. Search for "tidyverse". Install it, and all its dependencies. The code for it is:

```r
#install.packages("tidyverse")
#Once it's installed, you have to load it. You can check it off in the list of packages, or use the fol
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.1     v purrr   1.0.1
## v tibble  3.2.1     v dplyr   1.1.1
## v tidyr   1.3.0     v stringr 1.5.0
## v readr   2.1.2     v forcats 1.0.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
#a few library() calls are always at the top of any scripts I write. For our task, we just need one lib
#Change working directory -- click on the "files" tab in the bottom right box. Navigate to the folder wh
#Code for this is:
setwd("~/Documents/R/DSSG Working With Data/2023 spring")
```

##Loading the Data

Let's read in the data using read_csv, and insodoing, learn about tab completion.

```r
read_csv(file = "raw data/nhgis0003_ts_nominal_county.csv")
```

```
## Rows: 15781 Columns: 31
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (8): GISJOIN, STATE, STATEFP, STATENH, COUNTY, COUNTYFP, COUNTYNH, NAME
## dbl (23): YEAR, A57AA, A57AB, A57AC, A57AD, B57AA, B57AB, B57AC, B57AD, B57A...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 15,781 x 31
##    GISJOIN   YEAR STATE   STATEFP STATENH COUNTY   COUNTYFP COUNTYNH NAME  A57AA
##    <chr>    <dbl> <chr>   <chr>   <chr>   <chr>    <chr>    <chr>    <chr> <dbl>
##  1 G0100010  1970 Alabama 01      010     Autauga~ 001      0010     Auta~ 13116
##  2 G0100030  1970 Alabama 01      010     Baldwin~ 003      0030     Bald~ 15815
##  3 G0100050  1970 Alabama 01      010     Barbour~ 005      0050     Barb~  9102
##  4 G0100070  1970 Alabama 01      010     Bibb Co~ 007      0070     Bibb      0
##  5 G0100090  1970 Alabama 01      010     Blount ~ 009      0090     Blou~  4390
##  6 G0100110  1970 Alabama 01      010     Bullock~ 011      0110     Bull~  4324
##  7 G0100130  1970 Alabama 01      010     Butler ~ 013      0130     Butl~  8033
##  8 G0100150  1970 Alabama 01      010     Calhoun~ 015      0150     Calh~ 66130
##  9 G0100170  1970 Alabama 01      010     Chamber~ 017      0170     Cham~ 15892
## 10 G0100190  1970 Alabama 01      010     Cheroke~ 019      0190     Cher~     0
## # i 15,771 more rows
## # i 21 more variables: A57AB <dbl>, A57AC <dbl>, A57AD <dbl>, B57AA <dbl>,
## #   B57AB <dbl>, B57AC <dbl>, B57AD <dbl>, B57AE <dbl>, B57AF <dbl>,
## #   B57AG <dbl>, B57AH <dbl>, B57AI <dbl>, B57AJ <dbl>, B57AK <dbl>,
## #   B57AL <dbl>, B57AM <dbl>, B57AN <dbl>, B57AO <dbl>, B57AP <dbl>,
## #   B57AQ <dbl>, B57AR <dbl>
```

```
#Don't forget to assign it to a name, though:
census_data_raw <- read_csv(file = "Raw data/nhgis0003_ts_nominal_county.csv")
```

```
## Rows: 15781 Columns: 31
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (8): GISJOIN, STATE, STATEFP, STATENH, COUNTY, COUNTYFP, COUNTYNH, NAME
## dbl (23): YEAR, A57AA, A57AB, A57AC, A57AD, B57AA, B57AB, B57AC, B57AD, B57A...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Please note that `read_csv()` is a function that's part of the tidyverse and is more efficient than the `read.csv()` function in base R.

R is telling us how it's interpreting each of the columns, and if it's wrong we can do `read_csv()` again and tell it specifically how we want it to interpret any of these columns. One other cool thing RStudio lets you do is click on a file and check it out. I do this all the time. Equivalently, you can run the code `View(census_data)`.

We now have a dataframe called census_data, but as you can tell, there are *a lot* of columns and a lot of really pointless names. The codebook will translate from the column codes to human-readable column names. Let's import our codebook and take a look.

##Activity: load the codebook yourself

You write the code! Our codebook is in Raw data and called codebook.csv . Import codebook, assign it the name codebook in R, and then view it. Don't forget to comment out what you're doing.

```
codebook <- read_csv("raw data/codebook.csv")
```

```
## Rows: 31 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): Code, Field
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
#Take a look at it
View(codebook)
```

Changing the column names en masse

Great! Now let's do something clever. Note the dimensions of our two dataframes in our Global Environment. 31 is repeated twice. Basically, we want the column names of `census_data` to be what we see in the `Field` column of `codebook`. And we can do that. To see the names of a dataframe, you can use the `names()` function.

```
#Checking the names of census_data
names(census_data_raw) #note tab-complete can save time and typos here!
```

```
##  [1] "GISJOIN"  "YEAR"     "STATE"    "STATEFP"  "STATENH"  "COUNTY"
##  [7] "COUNTYFP" "COUNTYNH" "NAME"     "A57AA"    "A57AB"    "A57AC"
## [13] "A57AD"    "B57AA"    "B57AB"    "B57AC"    "B57AD"    "B57AE"
## [19] "B57AF"    "B57AG"    "B57AH"    "B57AI"    "B57AJ"    "B57AK"
## [25] "B57AL"    "B57AM"    "B57AN"    "B57AO"    "B57AP"    "B57AQ"
## [31] "B57AR"
```

```
#Now I'll teach you how to grab a column of a dataframe in R. You use $, and again, tab-complete can be
codebook$Field
```

```
##  [1] "GIS Join Match Code"
##  [2] "Row Source Year"
##  [3] "NHGIS Integrated State Name"
##  [4] "FIPS State Code"
##  [5] "NHGIS Integrated State Code"
##  [6] "NHGIS Integrated County Name"
##  [7] "FIPS County Code"
##  [8] "NHGIS Integrated County Code"
##  [9] "Year-Specific Area Name"
## [10] "Urban"
## [11] "Urban--Inside urbanized areas"
## [12] "Urban--Outside urbanized areas (in urban clusters)"
## [13] "Rural"
## [14] "Under 5 years"
## [15] "5 to 9 years"
## [16] "10 to 14 years"
## [17] "15 to 17 years"
## [18] "18 and 19 years"
## [19] "20 years"
## [20] "21 years"
## [21] "22 to 24 years"
## [22] "25 to 29 years"
## [23] "30 to 34 years"
## [24] "35 to 44 years"
## [25] "45 to 54 years"
## [26] "55 to 59 years"
## [27] "60 and 61 years"
## [28] "62 to 64 years"
## [29] "65 to 74 years"
## [30] "75 to 84 years"
## [31] "85 years and over"
```

```
#We want to replace the current census data names with the values in codebook$field. We can do that by
#to the other, like so:
names(census_data_raw) <- codebook$Field
```

Now look at the census_data again! Especially those absurd age breakdowns. Don't worry, we'll be fixing all of this in the next script, when we start talking about tidying the data by combining columns. But first, even R, let's keep our raw data raw and make a new dataframe to manipulate in our next steps.

##Selecting, reordering, and combining columns

We're going to dig deeper in the Tidyverse in this part and really get into reshaping our data in ways that'll make it useful to be visualized, in map or other forms.

First, let's only keep the columns we want from this sheet. The `select()` function is really amazing and flexible at letting us do this efficiently.

Also, for this section, we're going to be using the pipe operator, `%>%`. You can make one with command-shift-m. What is it? It's a useful little tool that allows you to build and run code line by line. It basically says, take the thing on my left and feed it into the next function. In words, you can think of it as "then". So:

```r
census_data_raw %>% select(`Row Source Year`, `NHGIS Integrated State Name`)
```

```
## # A tibble: 15,781 x 2
##    `Row Source Year` `NHGIS Integrated State Name`
##                <dbl> <chr>
##  1              1970 Alabama
##  2              1970 Alabama
##  3              1970 Alabama
##  4              1970 Alabama
##  5              1970 Alabama
##  6              1970 Alabama
##  7              1970 Alabama
##  8              1970 Alabama
##  9              1970 Alabama
## 10              1970 Alabama
## # i 15,771 more rows
```

```r
#You can read this code as "Take census data then select Row Source Year and NGIS Integrated State Name
select(census_data_raw, `Row Source Year`, `NHGIS Integrated State Name`)
```

```
## # A tibble: 15,781 x 2
##    `Row Source Year` `NHGIS Integrated State Name`
##                <dbl> <chr>
##  1              1970 Alabama
##  2              1970 Alabama
##  3              1970 Alabama
##  4              1970 Alabama
##  5              1970 Alabama
##  6              1970 Alabama
##  7              1970 Alabama
##  8              1970 Alabama
##  9              1970 Alabama
## 10              1970 Alabama
## # i 15,771 more rows
```

You can do really powerful things with `select()`. Let's run `?select()` to look at some examples. `select()!` is incredibly flexible; you can even rename columns while selecting them, or unselect columns by saying -colname. `data %>% select(col_new_name=col_old_name, colb:colz, -colq)`...

Which columns do we want to keep? We're not going to use FIPS in this part of the workshop. We'll just keep the state and county names. Taking only the columns we need; using 'Year-Specific Area Name' rather than the County Name field because in the latter county is repeated over and over again. Renaming some fields while selecting.

```r
census_data_cleaned <- census_data_raw %>% select(state=`NHGIS Integrated State Name`, county=`Year-Spe
                                                  year=`Row Source Year`, Urban:`85 years and over`)
```

Now we're down to 25 columns, which is a start. Let's combine some ages. I'd like the age ranges to be, for the most part, 10 years. We're going to create new columns to do that. The function for creating new columns is called `mutate()`. Again, it's always a good idea to check documentation for examples before using a new function. Let's run `?mutate()`

Let's start with one new column to see how this all works. Let's combine 5-14 years and create a new object to do so.

```r
census_data_cleaned <- census_data_cleaned %>% mutate(`5 to 14 years`= `5 to 9 years` + `10 to 14 years
                                                          #Optionally one can remove the columns used to ma
```

```
                                                  #this point using the following:
                                                  #`5 to 9 years` = NULL,
                                                  #`10 to 14 years`=NULL
                                                  )
```

We now have a new column at the end. We'll get rid of the columns we used to create the new column later with `select()`. Let's create the rest of our combined columns all at once, since `mutate()` allows you to create multiple columns at once. I'll code this for you because it's tedious to type, but read through to make sure you understand.

```
census_data_cleaned <- census_data_cleaned %>% mutate(`15 to 24 years`= `15 to 17 years` + `20 years`+
                                                  `21 years`+`22 to 24 years`,
                                                  `25 to 34 years` = `25 to 29 years` + `30 to 34 ye
                                                  `55 to 64 years` = `55 to 59 years` + `60 and 61 
#Let's get rid of the columns we used to make our new columns with select().
census_data_cleaned <- census_data_cleaned %>% select(-(`5 to 9 years`:`30 to 34 years`),
                                                  -(`55 to 59 years`:`62 to 64 years`))
```

##Changing Data from Wide to Long

Combining columns is not all I want to do with my data. I'm going to be putting it in a program where the preferred format is the long format, as in, I want a column called "Age". There's an easy function to do this and the reverse operation called `pivot_longer()` and `pivot_wider()`. Let's take a look at what the function takes with `?pivot_longer()`.

```
census_data_cleaned <- census_data_cleaned %>% pivot_longer(cols = `Under 5 years`:`55 to 64 years`,
                                                        names_to = "Age", values_to = "No. of peopl
```

It's a bit risky to overwrite my dataframe as I did in the steps above. If I'd made a mistake (which I did occasionally), I'd have to go back to chunk 9, where I select columns from `census_data_raw`. I find this to be an okay compromise rather than have a bunch of intermediate dataframes in my environment, but every coder has their own style and best practices, and you might prefer a different approach.

#Export csv of cleaner data

I find it to be a good practice to export a csv once you get your data into a reasonably cleaned format. We're going to work more with this data in a moment, but for now, let's write a csv with the complete, updated dataset.

```
write_csv(census_data_cleaned, "exports/census_data_cleaned.csv")
```

##Filter data

My first research question involves exploring aging trends by county in Massachusetts. To answer this question, I obviously don't need information on the other states in my dataframe. I'm going to use the `filter()` function to keep just the values I want. Keep in mind that you can do much fancier things with `filter()`, though; check out the documentation for examples.

```
#Check examples first
?filter()
```

```
## Help on topic 'filter' was found in the following packages:
##
##   Package               Library
##   stats                 /Library/Frameworks/R.framework/Versions/4.2/Resources/library
##   dplyr                 /Library/Frameworks/R.framework/Versions/4.2/Resources/library
##
##
## Using the first match ...
```

```r
census_data_MA <- census_data_cleaned %>% filter(state=="Massachusetts")
#Optionally: write to a file
write_csv(census_data_MA, "exports/census_data_MA.csv")
```

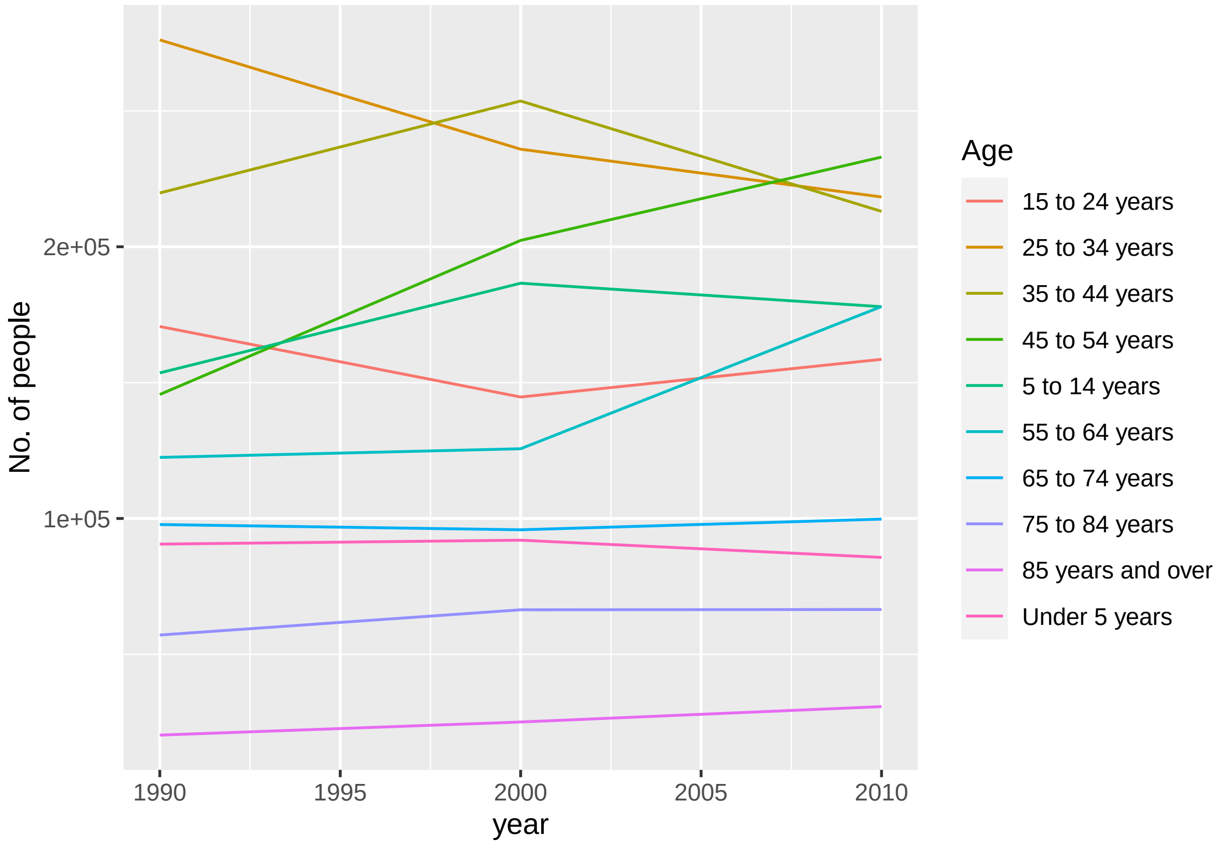
#Filter activity

Harvard University is in Middlesex County. Let's take a closer look at aging trends for just Middlesex County. Your activity is to filter for Middlesex County and call your resulting dataframe `census_data_MA_county`.

```r
census_data_MA_county <- census_data_MA %>% filter(county=="Middlesex County")
```

I'm not going to teach you data visualization today, but run the following chunk of code to get a preview of what this would look like.

```r
census_data_MA_county %>% ggplot(aes(x=year, y=`No. of people`, group=Age)) +
  geom_line(aes(color=Age))
```



##Grouping and Summarizing

Next research question: on a state level, has the population been aging over time? Our data is currently formatted such that a row is a snapshot of the number of people in an age group by county and by year. To look at how MA's population is changing over time, we want to have our data broken down by number of people in an age group by year – not broken down by county as well.

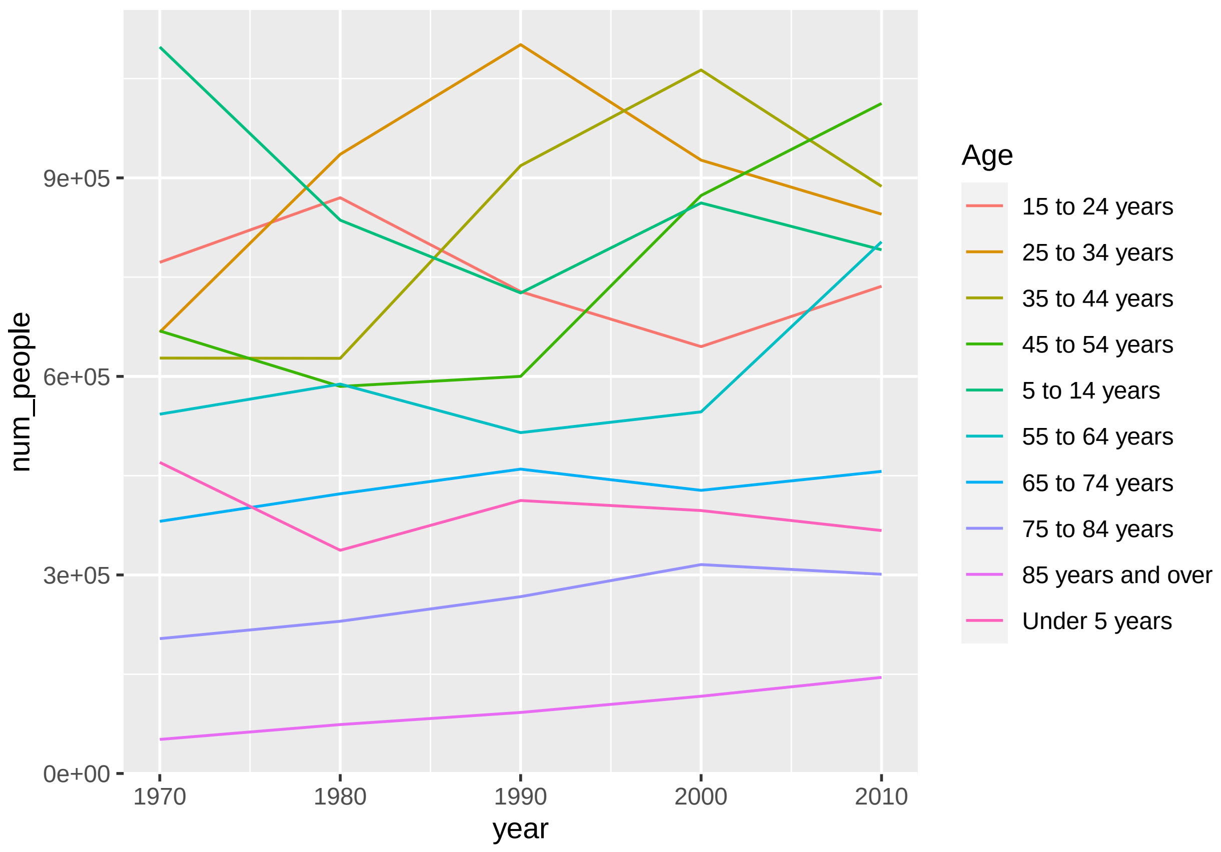
So, for example, in the year 1990, we want to add together all the people in the `Under 5 years` age group from all of the counties. How can we do this? In R, we use a combination of `group_by()` and `summarise()`.

`group_by()` was once described to me as dplyr's black magic. But basically, you're making transient pivot tables and doing operations within these transient pivot tables.

```r
census_data_MA_age <- census_data_MA %>% group_by(year, Age) %>%
  summarise(num_people=sum(`No. of people`)) %>%
  ungroup()
```

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

```r
ggplot(census_data_MA_age, aes(x = year, y = num_people, group = Age)) +
  geom_line(aes(color=Age))
```



#Advanced: Grouping within a column

What if I wanted to zone in on the story of older versus younger people? I'd like to compare under-35s to middle ages to 65-plus. It's possible to do a conditional mutate in R that would create a new column with these breakdowns.

Basically, I want to create a new column that will say `65 plus` whenever it finds rows in the `Age` column marked `65 to 74 years`, `75 to 84 years`, or `85 years and over`. I used this how-to guide to figure out how this could work, and the starwars `case_when()` example (reminder to always check `?case_when()` before trying a new function).

```r
broader_age_breakdown <- census_data_MA_age %>% mutate(
    broader_age= case_when(Age=="65 to 74 years" | Age=="75 to 84 years" | Age=="85 years and over" ~ "
         Age=="35 to 44 years" | Age=="45 to 54 years" | Age=="55 to 64 years" ~ "35 to 64",
         .default = "Under 35"))
```

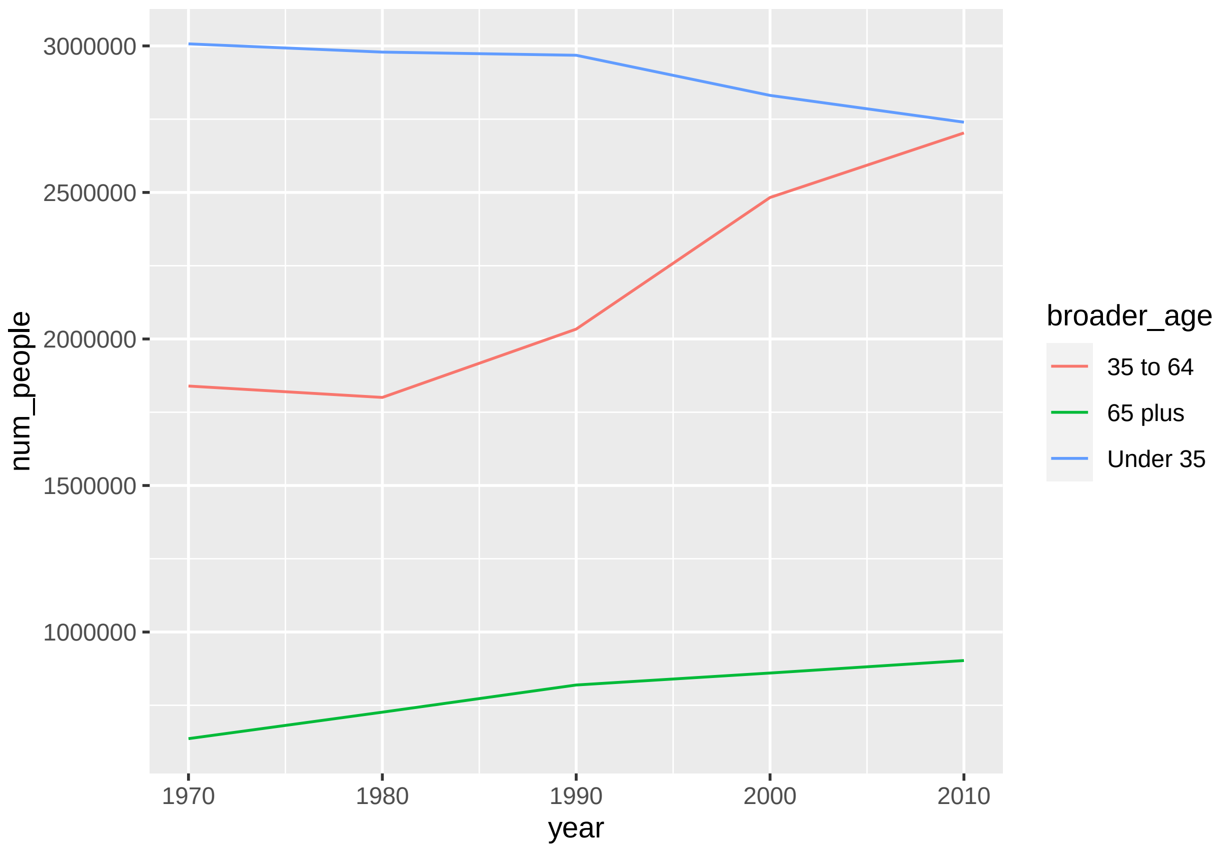
#Advanced activity: do your own `group_by()` and `summarise()`

Use the `broader_age_breakdown` to create a new dataframe with the columns year, broader_age, and num_people.

```
broader_age_breakdown <- broader_age_breakdown %>% group_by(year, broader_age) %>% summarise(num_people
```

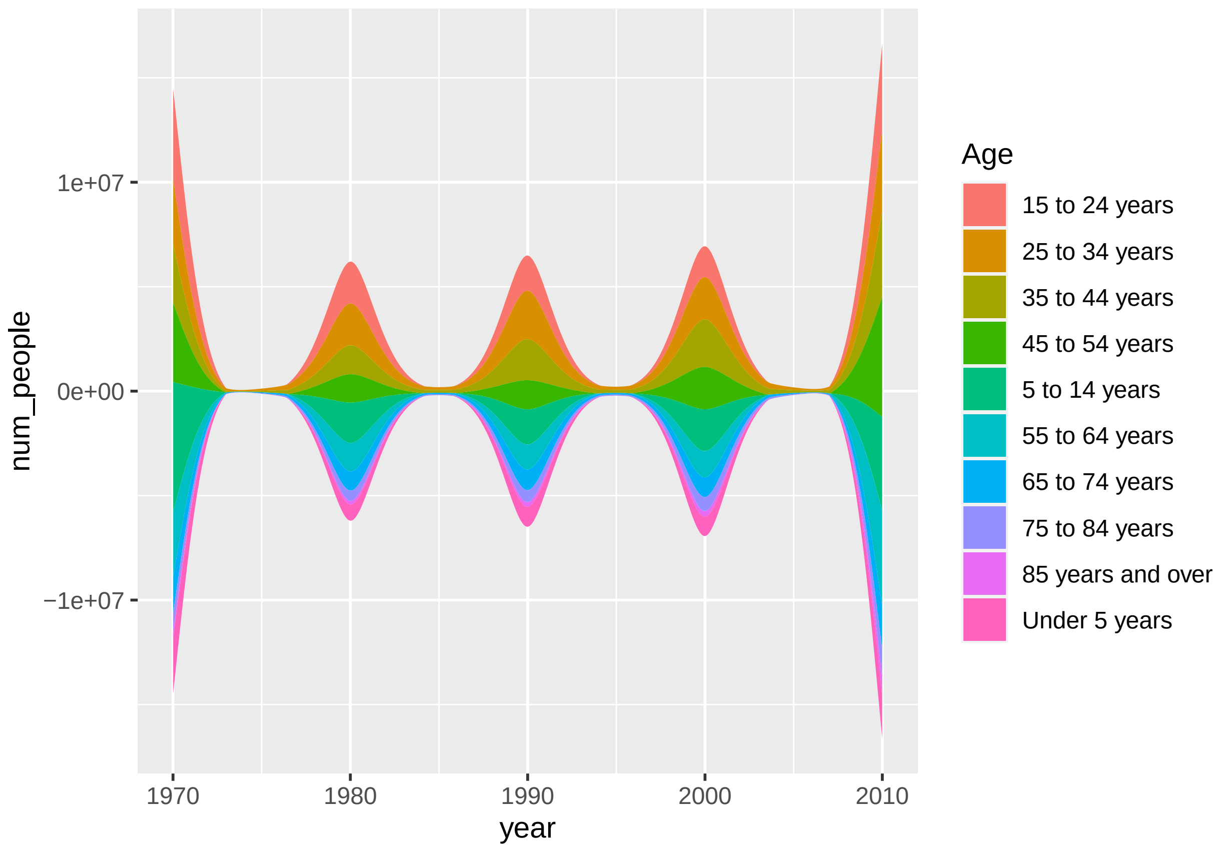
```
## `summarise()` has grouped output by 'year'. You can override using the
## `.groups` argument.
```

```
ggplot(broader_age_breakdown, aes(x = year, y = num_people, group = broader_age)) +
  geom_line(aes(color=broader_age))
```



Probably going to delete code below but it was cool.

```
#install.packages("remotes")
#remotes::install_github("davidsjoberg/ggstream")
library(ggstream)
ggplot(census_data_MA_age, aes(x = year, y = num_people, fill = Age)) +
  geom_stream()
```



1e+07
0e+00
−1e+07
num_people
1970
1980
1990
2000
2010
year
Age
15 to 24 years
25 to 34 years
35 to 44 years
45 to 54 years
5 to 14 years
55 to 64 years
65 to 74 years
75 to 84 years
85 years and over
Under 5 years