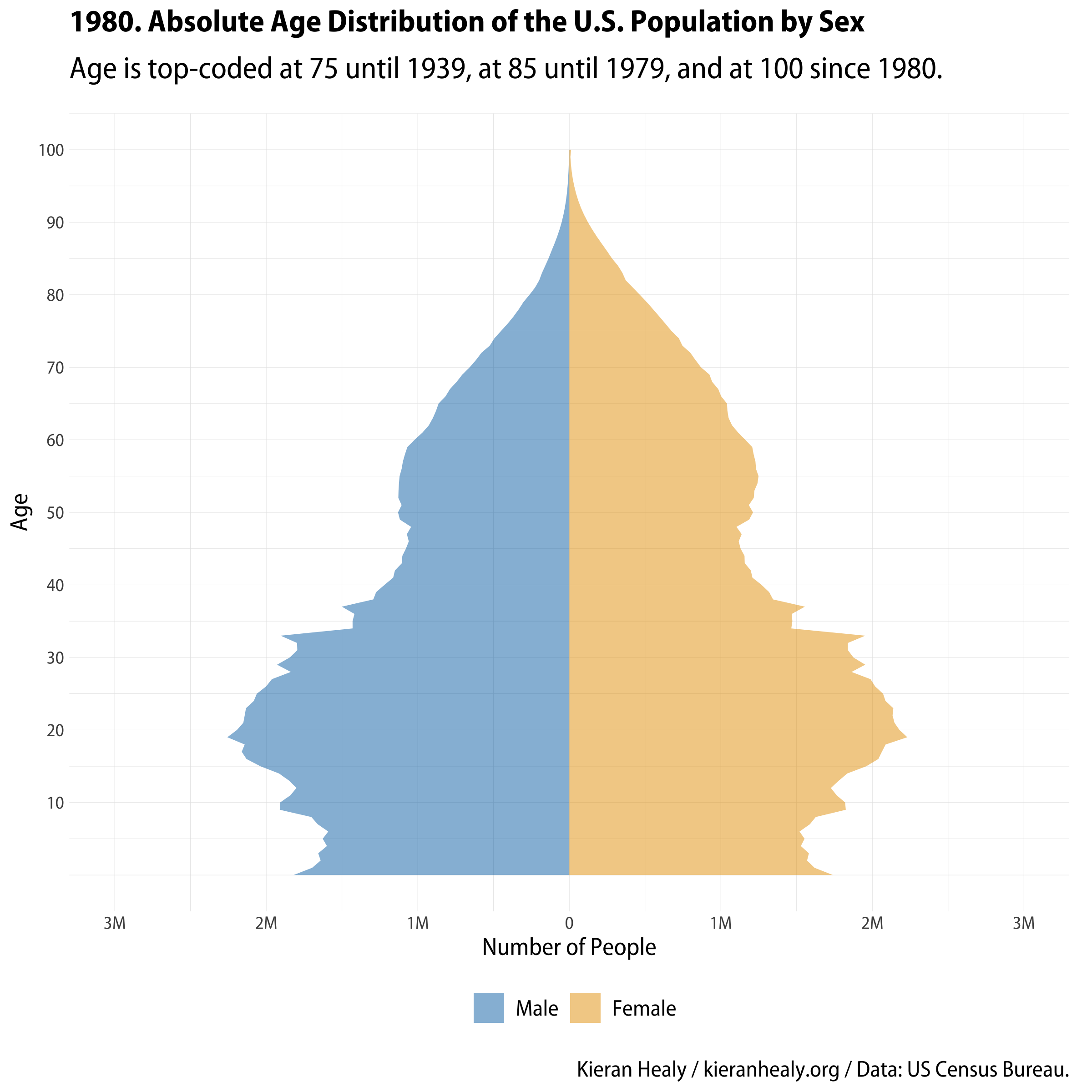
With the 2020 U.S. Census in motion already, I’ve been looking at various pieces of data from the [Census Bureau](http://census.gov). I decided I wanted to draw some population pyramids for the U.S. over as long a time series as I could. What’s needed for that are tables for, say, as many years as possible that show the number of males and females alive at every year of age from zero to the highest age you’re willing to track. This sort of data *is* available on the Census website. But it tuned out to be somewhat tedious to assemble into a single usable series. (Perhaps it’s available in an easy-to-digest form elsewhere, but I couldn’t find it.) I initially worked with a couple of the excellent R packages that talk to the Census API (tidycensus and censusapi), hoping they’d give me what I needed. But in the end I wrangled an annual year-of-age series from 1900 to 2019 by grabbing the data from the Census and cleaning it myself. As always, 95% of data analysis is in fact data acquisition and data cleaning.

[](https://kieranhealy.org/files/misc/us_pyramid_1980.png)

First we get ourselves set up as usual.

r

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | library(tidyverse)  library(here)  library(janitor)  library(socviz)  library(gganimate)  library(transformr)  ## --------------------------------------------------------------------  ## Custom font and theme, omit if you don't have the myriad library  ## (https://github.com/kjhealy/myriad) and associated Adobe fonts.  ## --------------------------------------------------------------------  library(showtext)  showtext\_auto()  library(myriad)  import\_myriad\_semi()  theme\_set(theme\_myriad\_semi()) |

Now, the data. What we want are the decennial and intercensal estimates by year, sex, and year of age. These aren’t all in the same place. Moreover, they aren’t all in the same format. The estimates for 1900 to 1979 are available [at this link](https://www2.census.gov/programs-surveys/popest/tables/1900-1980/national/asrh/?C=N;O=D), but (as quickly became clear), the format of the CSV file changes slightly. Subsequent decades vary their format and expand the range of measures counted. Some of the formats are rather difficult to work with. For example, here’s part of the description of the 1980-89 files:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | The 1990 monthly postcensal national population estimate data files have  an identical layout. All records contain 222 characters. All data fields  are right-justified.  Location Type Data  1-2 Character Series  3-4 Numeric Month  5-8 Numeric Year  9-11 Numeric Age (years)  12 (blank) (blank)  13-22 Numeric Total population  23-32 Numeric Total male population  33-42 Numeric Total female population  43-52 Numeric White male population  53-62 Numeric White female population  63-72 Numeric Black male population  73-82 Numeric Black female population |

And then:

|  |  |
| --- | --- |
| 1  2  3  4  5 | Within each file, the records are first sorted by the reference date  (Month-Year) in chronological order. For each reference date, the first  record lists the population counts for all ages combined. The remaining  records list the population counts by single year of age in ascending  order. |

That means that the data file for any particular year during this period looks something like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 2I 780 98 14234 3485  2I 780 99 9652 2409  2I 780100 15099 3244  2I1080999 227924215 110746612  2I1080 0 3582352 1832733  2I1080 1 3360607 1718828  2I1080 2 3217219 1645162 |

Not so nice. The cleanest way to work with stuff like this would be to write a spec to read in the data by column position. In the end I wrote a series of short scripts using some old-fashioned Unix tools, especially sed, to do the slicing and dicing for me. They looked like this:

bash

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | ## Extract every row between the first  ## July estimate (^2I 7) to oct (2I10)  for filename in \*.TXT; do  gsed -i.bak -n '/^2I 7/,/2I10/p;/^\+/p' "$filename"  done  ## cut away the first 7 columns  for filename in \*.TXT; do  cut -c7- <"$filename" >"${filename%.TXT}.new"  done  ## trim the first and last lines  for filename in \*.new; do  gsed -i '1d;$d' "$filename"  done |

In the end I had some fairly clean delimited files that I could work with that needed only a little more cleaning in R. For each batch of files I’d do something like this: get a list of the files needed from the directory, read the contents into a tibble and harmonize the column names if needed. Here’s the segment for the 1980s files, for example:

r

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | target <- "1980\_1989"  path <- paste0("data/",target)  filenames <- dir(path = here(path),  pattern = "\*.new$",  full.names = TRUE)  pop\_1980\_1989 <- tibble(  year = get\_80syr(filenames),  path = filenames,  data = map(filenames, ~ read\_delim(., delim = " "))  ) %>%  mutate(data = map(data, ~  .x %>%  mutate\_if(is.character, as.numeric)))  pop\_1980\_1989  # A tibble: 10 x 3  year path data  <chr> <chr> <list>  1 1980 /Users/kjhealy/Documents/data/misc/census\_pop… <tibble [101 …  2 1981 /Users/kjhealy/Documents/data/misc/census\_pop… <tibble [101 …  3 1982 /Users/kjhealy/Documents/data/misc/census\_pop… <tibble [101 …  4 1983 /Users/kjhealy/Documents/data/misc/census\_pop… <tibble [101 …  5 1984 /Users/kjhealy/Documents/data/misc/census\_pop… <tibble [101 …  6 1985 /Users/kjhealy/Documents/data/misc/census\_pop… <tibble [101 …  7 1986 /Users/kjhealy/Documents/data/misc/census\_pop… <tibble [101 …  8 1987 /Users/kjhealy/Documents/data/misc/census\_pop… <tibble [101 …  9 1988 /Users/kjhealy/Documents/data/misc/census\_pop… <tibble [101 …  10 1989 /Users/kjhealy/Documents/data/misc/census\_pop… <tibble [101 … |

Eventually all the series are read in and can be bound together and the year, age, and population counts extracted.

r

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | # Now we're suckin' diesel  pop\_data <- bind\_rows(pop\_1900\_1959,  pop\_1960\_1979,  pop\_1980\_1989,  pop\_1990\_1999,  pop\_2000\_2009,  pop\_2010\_2019)  pop\_series <- unnest(pop\_data, cols = c(data)) %>%  select(-path) %>%  select(year, age, pop, male, female)  pop\_series  # A tibble: 10,520 x 5  year age pop male female  <chr> <dbl> <dbl> <dbl> <dbl>  1 1900 0 1811000 919000 892000  2 1900 1 1835000 928000 907000  3 1900 2 1846000 932000 914000  4 1900 3 1848000 932000 916000  5 1900 4 1841000 928000 913000  6 1900 5 1827000 921000 906000  7 1900 6 1806000 911000 895000  8 1900 7 1780000 899000 881000  9 1900 8 1750000 884000 866000  10 1900 9 1717000 868000 849000  # … with 10,510 more rows |

From there we pivot series to long format:

r

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | pop\_lon <- pop\_series %>% select(year, age, male, female) %>%  pivot\_longer(male:female, names\_to = "group", values\_to = "count") %>%  group\_by(year, group) %>%  mutate(total = sum(count),  pct = (count/total)\*100,  base = 0)  pop\_lon  # A tibble: 21,040 x 7  # Groups: year, group [240]  year age group count total pct base  <chr> <dbl> <chr> <dbl> <dbl> <dbl> <dbl>  1 1900 0 male 919000 38867000 2.36 0  2 1900 0 female 892000 37227000 2.40 0  3 1900 1 male 928000 38867000 2.39 0  4 1900 1 female 907000 37227000 2.44 0  5 1900 2 male 932000 38867000 2.40 0  6 1900 2 female 914000 37227000 2.46 0  7 1900 3 male 932000 38867000 2.40 0  8 1900 3 female 916000 37227000 2.46 0  9 1900 4 male 928000 38867000 2.39 0  10 1900 4 female 913000 37227000 2.45 0  # … with 21,030 more rows |

Here, within each year and for males and females, we calculate the percentage of the total population that is of any particular age. As I mentioned, one feature of the Census data is that over the years the top-code for age—the highest age the Census tables report—gradually increases. We can see what those limits are and when they change:

r

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | pop\_series %>%  group\_by(year) %>%  summarize(max\_age = max(age)) %>%  group\_by(max\_age) %>%  summarize(minyr = min(year),  maxyr = max(year))  # A tibble: 3 x 3  max\_age minyr maxyr  <dbl> <chr> <chr>  1 75 1900 1939  2 85 1940 1979  3 100 1980 2019 |

Now we can make some animations. First, rather than a population pyramid, we’ll use geom\_density() to produce kernel density estimates of the age distribution for every year, for both males and females. In cases like this, when we have a variable like year and a summary count for each age in that year (but not individual-level observations), the way to get the density is to put age on the x-axis and use the proportion (pct/100) to weight each year-of-age. (Weights need to sum to 1, hence the use of proportions rather than percents.) Here we’re using the after\_stat() function that’s new in the scales package and ggplot2 version 3.3.0. This way of expressing what we want to do replaces earlier syntaxes like the double-period ..density.. convention.

r

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | p\_dens <- pop\_lon %>%  ggplot(aes(x = age,  y = after\_stat(density),  weight = pct/100,  fill = group,  group = group)) +  geom\_density(color = "black", alpha = 0.5) +  scale\_fill\_manual(values = my.colors("bly"),  labels = c("Female", "Male")) +  scale\_x\_continuous(breaks = seq(0, 100, 10),  labels = as.character(seq(0, 100, 10))) +  guides(fill = guide\_legend(label.position = "bottom", keywidth = 2),  color = guide\_legend(label.position = "bottom", keywidth = 2)) +  labs(x = "Age", y = "Estimated Density",  color = NULL, fill = NULL,  title = "{frame\_time}. Relative Age Distribution of the U.S. Population by Sex",  subtitle = "Age is top-coded at 75 until 1939, at 85 until 1979, and at 100 since 1980.",  caption = "@kjhealy / http://kieranhealy.org.") +  theme(legend.position = "bottom",  plot.title = element\_text(size = rel(3), face = "bold"),  plot.subtitle = element\_text(size = rel(3)),  plot.caption = element\_text(size = rel(2)),  axis.text.y = element\_text(size = rel(3)),  axis.title.x = element\_text(size = rel(3)),  axis.title.y = element\_text(size = rel(3)),  axis.text.x = element\_text(size = rel(3)),  legend.text = element\_text(size = rel(3))) +  transition\_time(as.integer(year(year))) +  ease\_aes("linear") +  transition\_time(as.integer(year)) +  ease\_aes("cubic-in-out")    animate(p\_dens, fps = 25, duration = 30, width = 1024, height = 1024, renderer = ffmpeg\_renderer()) |

The theme() calls are all about making the default label text larger, using the handy rel() function to boost size in relative terms rather than worrying about units. We get the animation almost for free, thanks to Thomas Lin Pedersen’s gganimate package. Just the two functions, transition\_time() and ease\_aes() do all the work. Then we use animate() to actually render the animation. After saving the results as an mp4 file, here’s what we get.

The curves here are estimated kernel densities. A more conventional way to represent the demographic data we have is with a *population pyramid*, where we put ages on the x axis and population counts (or percentages) on the y axis, and then put males on the left and females on the right. To accomplish this in R we’ll use geom\_ribbon() and cheat a little bit by making the ages for males all be negative. Then we’ll set the base of the male and female ribbons to be zero. Here’s how that works. We’re going to show the absolute rather than the relative population distribution, so we can watch the size of the pyramid grow over time as well as see its shape change.

r

|  |  |
| --- | --- |
| 1  2  3  4 | pop\_pyr <- pop\_lon  ## Make all the Male ages negative  pop\_pyr$count[pop\_pyr$group == "male"] <- -pop\_pyr$count[pop\_pyr$group == "male"] |

The code for the plot is very similar to before:

r

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | mbreaks <- c("1M", "2M", "3M")  p <- ggplot(data = pop\_pyr,  mapping = aes(x = age, ymin = base,  ymax = count, fill = group))  p\_pyr\_count <- p + geom\_ribbon(alpha = 0.5) +  scale\_y\_continuous(labels = c(rev(mbreaks), "0", mbreaks),  breaks = seq(-3e6, 3e6, 1e6),  limits = c(-3e6, 3e6)) +  scale\_x\_continuous(breaks = seq(10, 100, 10)) +  scale\_fill\_manual(values = my.colors("bly"), labels = c("Female", "Male")) +  guides(fill = guide\_legend(reverse = TRUE)) +  labs(x = "Age", y = "Number of People",  title = "{frame\_time}. Absolute Age Distribution of the U.S. Population by Sex",  subtitle = "Age is top-coded at 75 until 1939, at 85 until 1979, and at 100 since 1980.",  caption = "Kieran Healy / kieranhealy.org / Data: US Census Bureau.",  fill = "") +  theme(legend.position = "bottom",  plot.title = element\_text(size = rel(3), face = "bold"),  plot.subtitle = element\_text(size = rel(3)),  plot.caption = element\_text(size = rel(2)),  axis.text.y = element\_text(size = rel(3)),  axis.text.x = element\_text(size = rel(3)),  axis.title.x = element\_text(size = rel(3)),  axis.title.y = element\_text(size = rel(3)),  legend.text = element\_text(size = rel(3))) +  coord\_flip() +  transition\_time(as.integer(year)) +  ease\_aes("cubic-in-out")      animate(p\_pyr\_count, fps = 25, duration = 60, width = 1024, height = 1024, renderer = ffmpeg\_renderer()) |

The main changes are in the labeling. geom\_ribbon needs a ymin and a ymax value. The former will always be zero. The latter will be the population count for that age. We make a little vector of population labels, mbreaks, for the x-axis, and join it up first in reverse, and then in regular order on either side of zero: labels = c(rev(mbreaks), "0", mbreaks). We also set the breaks between -3 million and 3 millon in steps of 1 millon: breaks = seq(-3e6, 3e6, 1e6). The cubic-in-out easing function makes for a better-looking step-by-step animation than the default linear, which bobbles around too much.

And here’s the result.

Just look at those Boomers go after 1946.

I’ve been a little sketchy about the details of the cleaning process above because what I want to do is package up the clean dataset shortly so that other people don’t have to experience the thrill of learning about the many virtues of [sed](https://www.gnu.org/software/sed/manual/sed.html).