Code

STAT 209: Lab 4

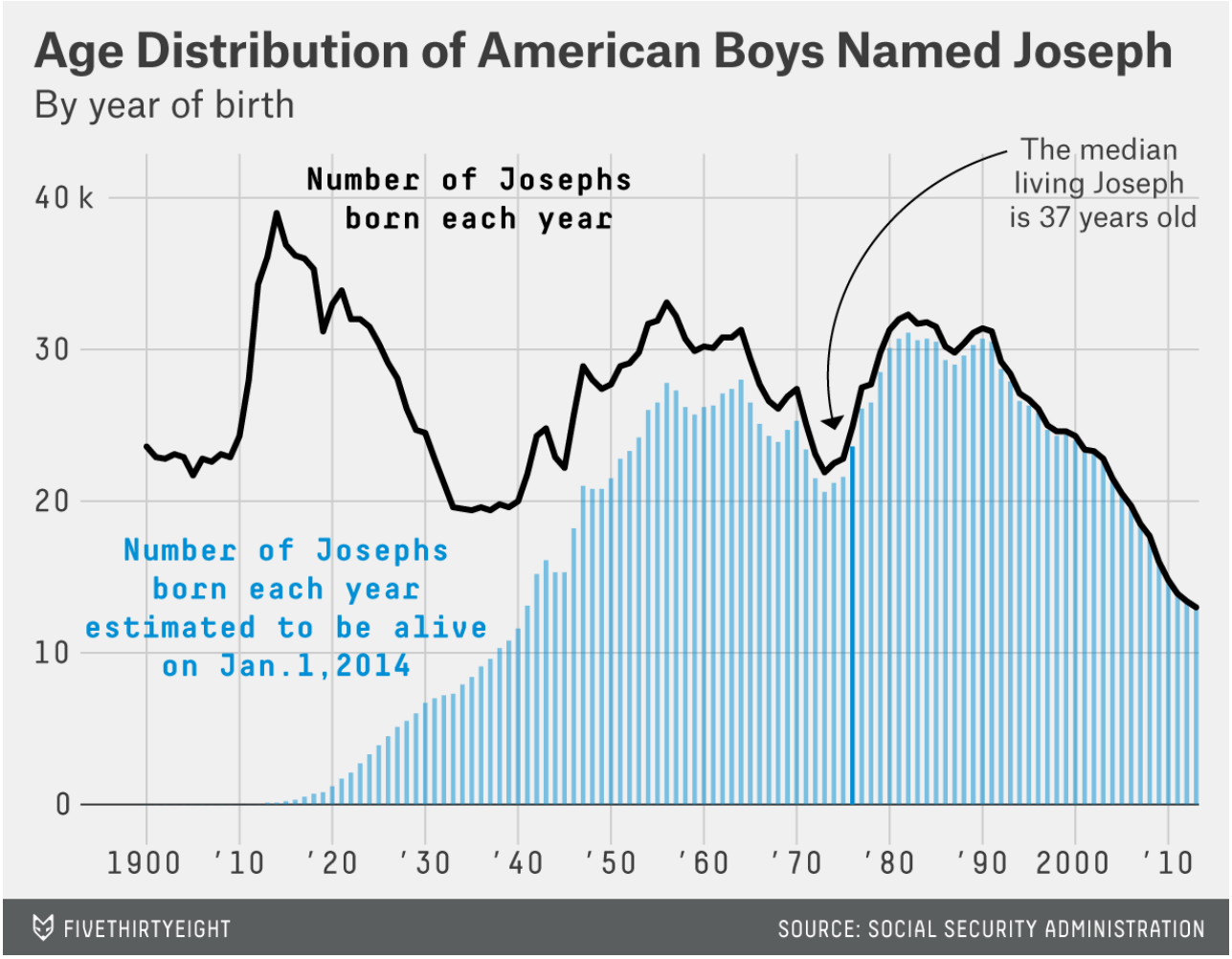
Reverse Engineering a Professional Writeup

Goal

Take a published data visualization and recreate it step-by-step in R with `ggplot2` . This lab is based on section 3.3 of the textbook *Modern Data Science with R*.

The Source

The visualization we will recreate comes from this (https://fivethirtyeight.com/features/how-to-tell-someones-age-when-all-you-know-is-her-name/) FiveThirtyEight post about the historical popularity of various baby names. We will focus on the second plot, depicting how many “Josephs” were born in each year, and how many of those are estimated to be alive today. For convenience, I’ve embedded the plot in the lab below.



Exercise 1 Before we write any code, examine the plot and write a sentence or two summarizing what it tells us. Then identify the data dimensions (variables) involved, what visual cue(s) each variable is mapped to, what coordinate systems and scales are used.

Data

The Social Security Administration (SSA) provides historical data on how many babies were born in each year with various names. The relevant data is provided in the `lifetables` data frame from the `babynames` package. Let's load it now.

Code

Code

To make life easier, the `mdsr` package provides a function called `make_baby_names_dist()` that does some useful preprocessing of the data so that we don't have to deal with any data-wrangling yet. It takes no arguments and returns a modified data frame.

Code

Code

```
## Observations: 1,639,722
## Variables: 9
## $ year            <dbl> 1900, 1900, 1900, 1900, 1900, 1900, 1900, 1900, …
## $ sex             <chr> "F", "F", "F", "F", "F", "F", "F", "F", "F", "F"…
## $ name            <chr> "Mary", "Helen", "Anna", "Margaret", "Ruth", "El…
## $ n               <int> 16706, 6343, 6114, 5304, 4765, 4096, 3920, 3896,…
## $ prop            <dbl> 0.05257559, 0.01996211, 0.01924142, 0.01669226, …
## $ alive_prob      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
## $ count_thousands <dbl> 16.706, 6.343, 6.114, 5.304, 4.765, 4.096, 3.920…
## $ age_today       <dbl> 114, 114, 114, 114, 114, 114, 114, 114, 114, 114…
## $ est_alive_today <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
```

Extracting the data we need

The plot only involves male babies named “Joseph”, so let’s create a filtered dataset with just those births.

Code

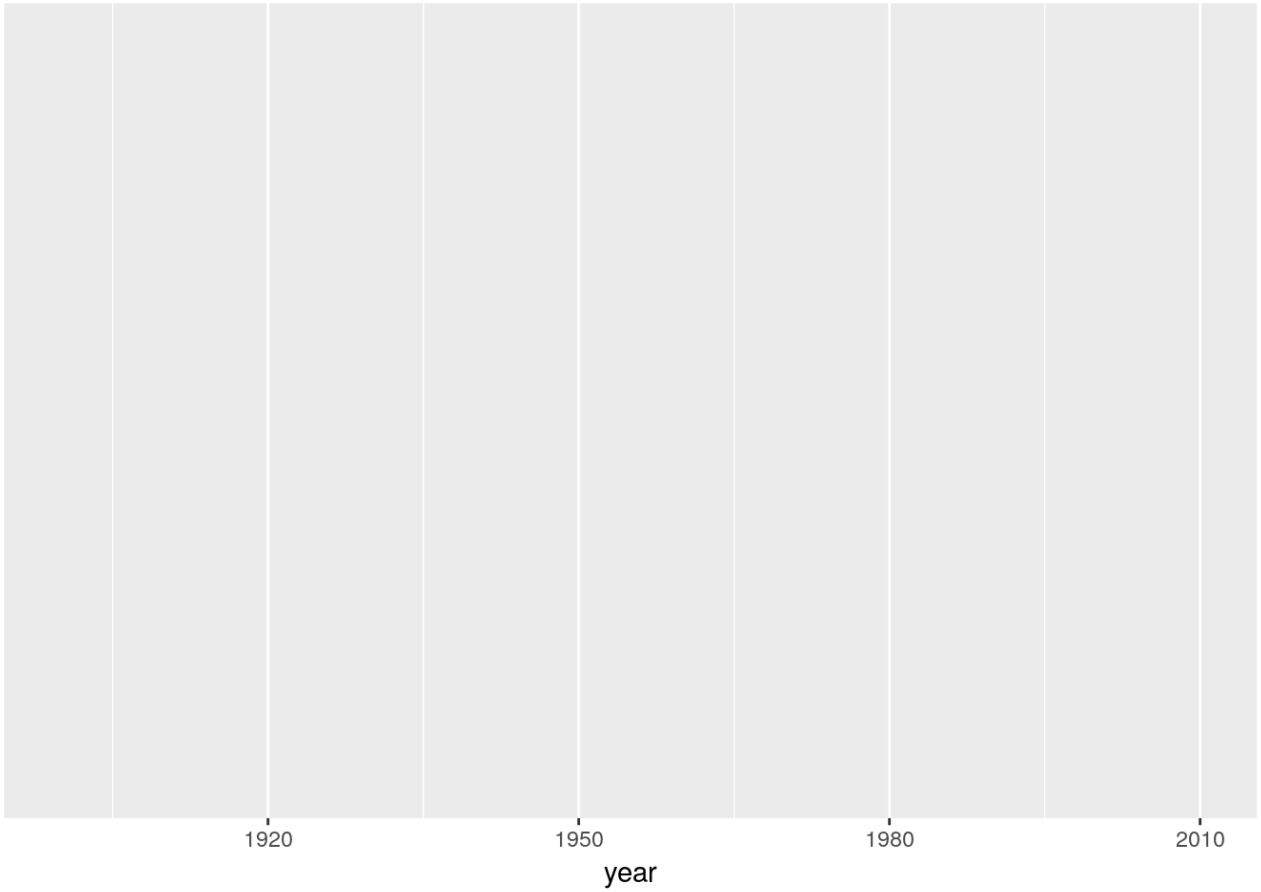
```
## Observations: 111
## Variables: 9
## $ year            <dbl> 1900, 1901, 1902, 1903, 1904, 1905, 1906, 1907, …
## $ sex             <chr> "M", "M", "M", "M", "M", "M", "M", "M", "M", "M"…
## $ name            <chr> "Joseph", "Joseph", "Joseph", "Joseph", "Joseph"…
## $ n               <int> 3714, 2766, 3098, 3121, 3291, 3302, 3527, 3844, …
## $ prop            <dbl> 0.02290712, 0.02392837, 0.02333710, 0.02413281, …
## $ alive_prob      <dbl> 0.000000, 0.000025, 0.000050, 0.000075, 0.000100…
## $ count_thousands <dbl> 3.714, 2.766, 3.098, 3.121, 3.291, 3.302, 3.527,…
## $ age_today       <dbl> 114, 113, 112, 111, 110, 109, 108, 107, 106, 105…
## $ est_alive_today <dbl> 0.000000, 0.069150, 0.154900, 0.234075, 0.329100…
```

Setting up the base plot

There are two `geoms` in this plot: a line depicting the total number of Josephs born in each year, and a set of bars depicting the number estimated to be alive today. In both cases, the x-axis depicts the year; however, there are two different variables mapped to the y-axis. So we don’t want to set up a *global* mapping to the y-axis; instead we will define the y mapping at the level of each geom. At the global level, we’ll just define the x mapping.

Code

Code



Adding the `geom`s

The bar graph component of the plot is based on the estimated number of male “Joseph”s alive today from each birth year. This is approximately the number of male Josephs born that year (`count_thousands`) times the probability that someone born that year is still alive now (`alive_prob`). The product of these two variables has already been computed for us as `est_alive_today` .

Code

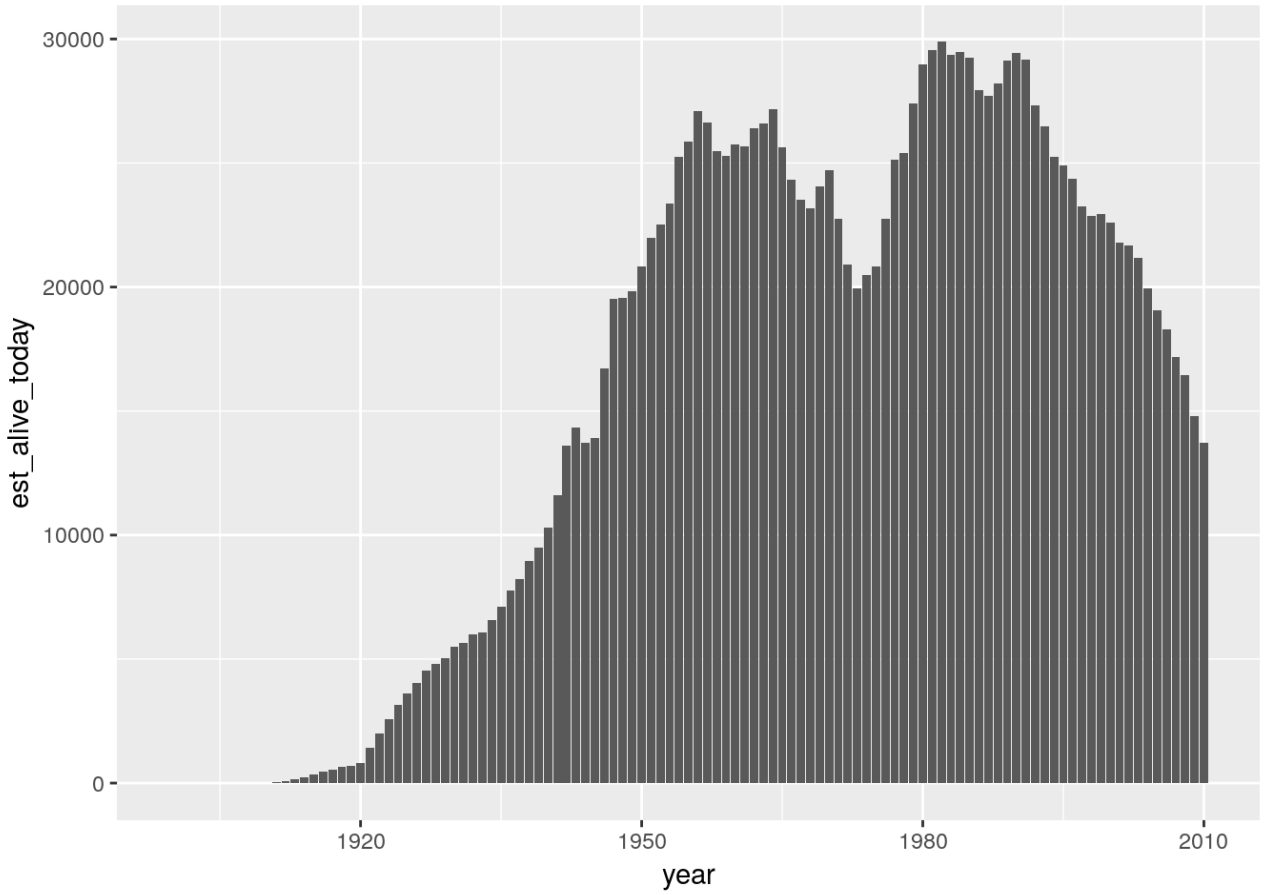
Oops, we get an error message.

The reason this gives an error is that the default for a bar graph is to count how often each level of the “x” variable occurs, and plot that count on the “y” axis. This is what we would want to do if each case in our data was an individual Joseph. In this type of plot, we can’t also map a variable to the y dimension; hence the error.

In our data, each case is a year, and we have the count precomputed in the `est_alive_today` variable.

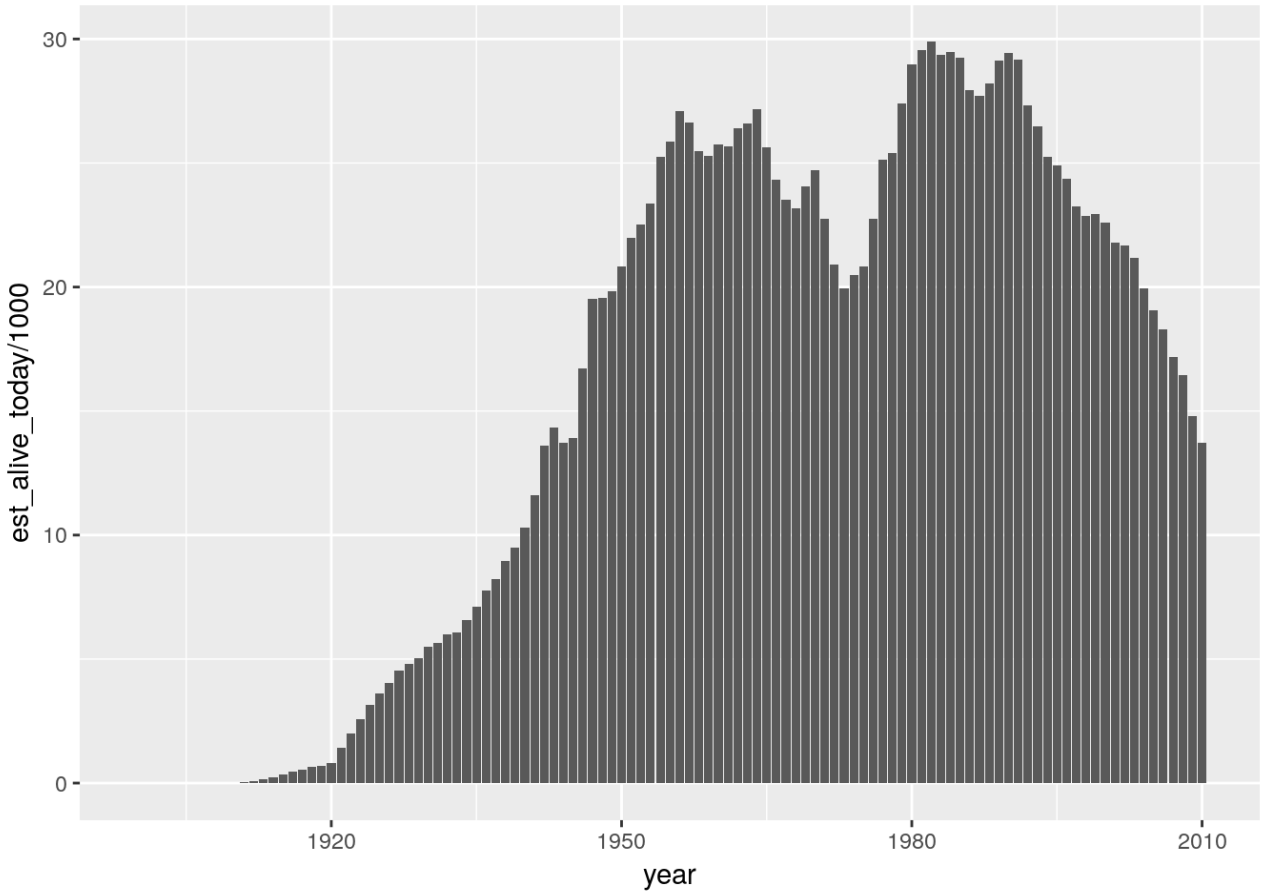
To tell `geom_bar` to use an existing variable instead of counting cases, we can specify `stat = "identity"` .

Code



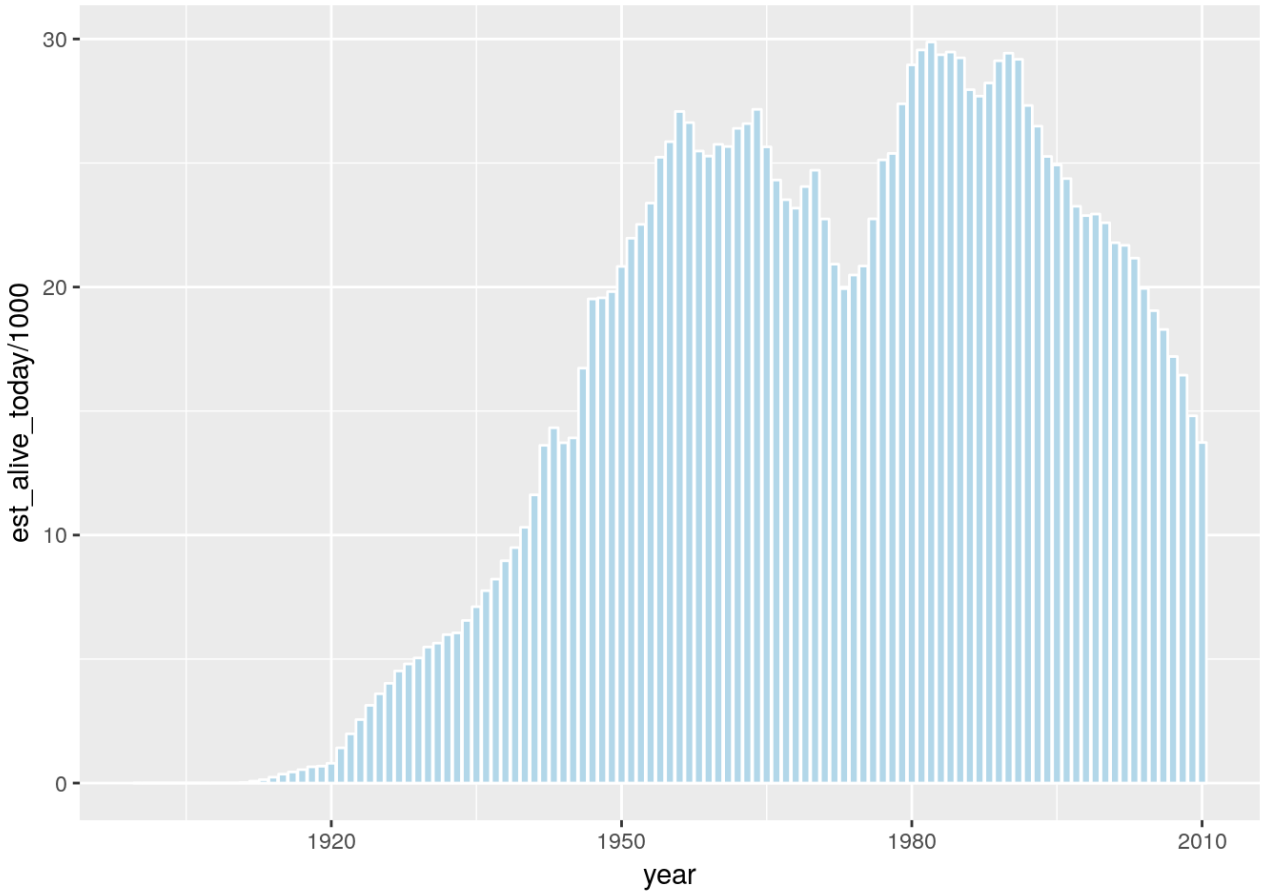
Hmm… This shows the data we want, but it's a little cluttered to have such big numbers on the y axis. The original plot shows thousands on the y axis; let's do the same.

Code



That looks cleaner. Let's try to match the colors of the original bars. The light blue fill is roughly color `#b2d7e9` , and for the white outlines, we'll just set `color="white"` .

Code

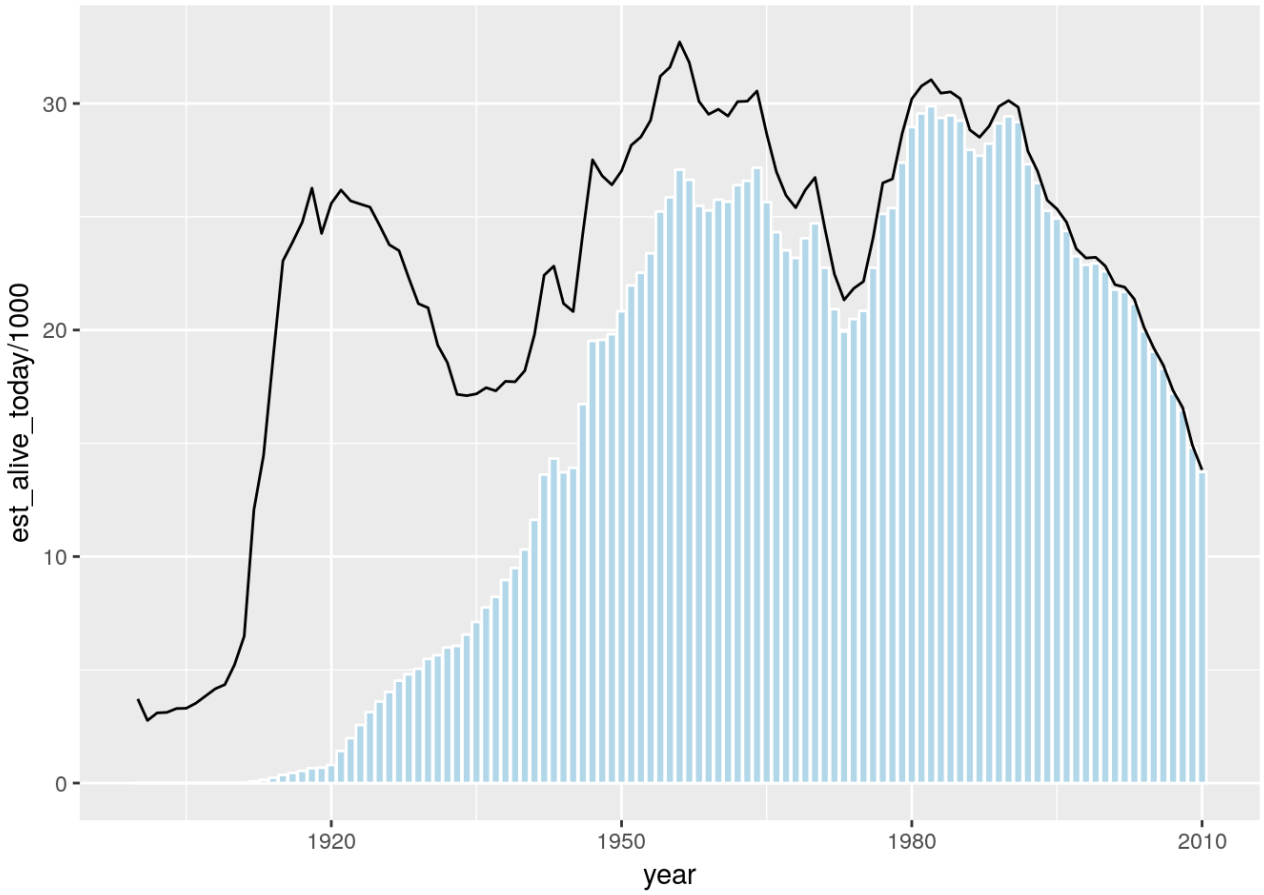


Ok, we should be done tinkering with the bar component, so let's save our updated plot.

Code

Now let's add the line. This should depict the total number of births in thousands, which is calculated already in the `count_thousands` variable.

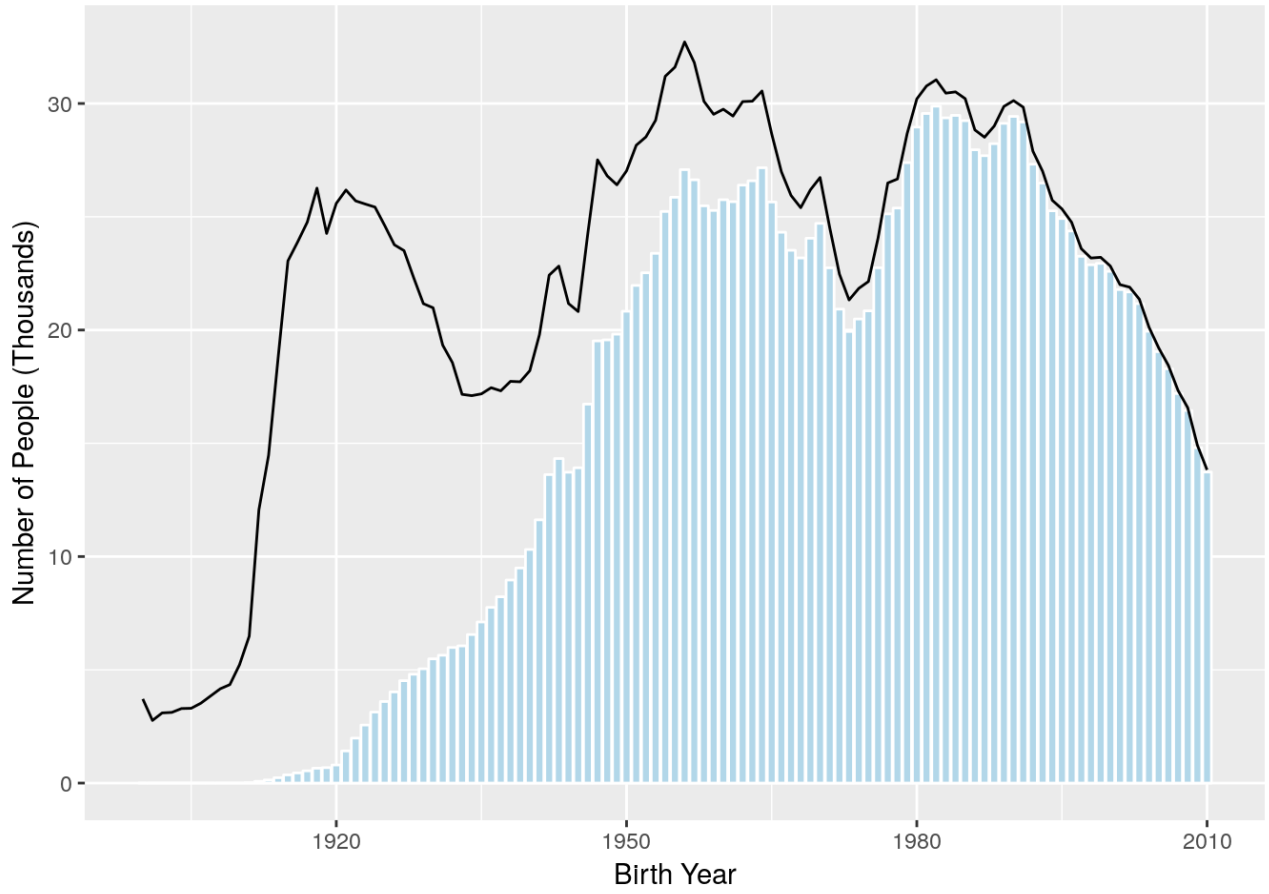
Code



Adding axis labels

We can provide more informative axis labels with the `xlab()` and `ylab()` functions. These are quick wrapper functions that let us easily change just the labels (for more fine-grained control over the axes, including things like the tick marks, we could use `scale_x_continuous()` or `scale_y_continuous()`).

Code



Since we created the plot over the course of several assignment operations, it can be useful to summarize what the `joseph_plot` object holds. By typing `summary(joseph_plot)` we can review all of the geometries and aesthetic mappings that we have defined so far (this will also display a big list of nuts-and-bolts properties attached to the “faceting” element, which you can ignore)

Code

```
## data: year, sex, name, n, prop, alive_prob, count_thousands,
##   age_today, est_alive_today [111x9]
## mapping:  x = ~year
## faceting: <ggproto object: Class FacetNull, Facet, gg>
##     compute_layout: function
##     draw_back: function
##     draw_front: function
##     draw_labels: function
##     draw_panels: function
##     finish_data: function
##     init_scales: function
##     map_data: function
##     params: list
##     setup_data: function
##     setup_params: function
##     shrink: TRUE
##     train_scales: function
##     vars: function
##     super:  <ggproto object: Class FacetNull, Facet, gg>
## ----------------------------------
## mapping: y = ~est_alive_today/1000
## geom_bar: width = NULL, na.rm = FALSE
## stat_identity: na.rm = FALSE
## position_stack
##
## mapping: y = ~count_thousands
## geom_line: na.rm = FALSE
## stat_identity: na.rm = FALSE
## position_identity
```

Highlighting a particular value in the bargraph

In the source graphic, the median age of living male Josephs is depicted both in text and with a darker colored bar. We'll address the annotation elements in a bit; for now let's try to (a) have R calculate the median age from the data, and (b) change the color of the bar.

Calculating the median age

If we had raw data on individual Josephs, we could simply calculate the median of an age variable. Since we instead have (estimated) counts at each birth year, we need to take a weighted median, finding the birth year for which half of the total count is on either side. Visually, we are finding the value that divides the "blue ink" in half horizontally.

A convenient function to calculate this is the `wtd.quantile()` function, available in the `Hmisc` package. Since it's an older package which is not part of the `tidyverse`, its syntax is unfortunately not the same as what we're used to; in particular, it doesn't provide a `data=` argument. Instead, we can specify the context in which our variable names are embedded using the `with()` function from base R, as follows:

Code

```
##  50%
## 1975
```

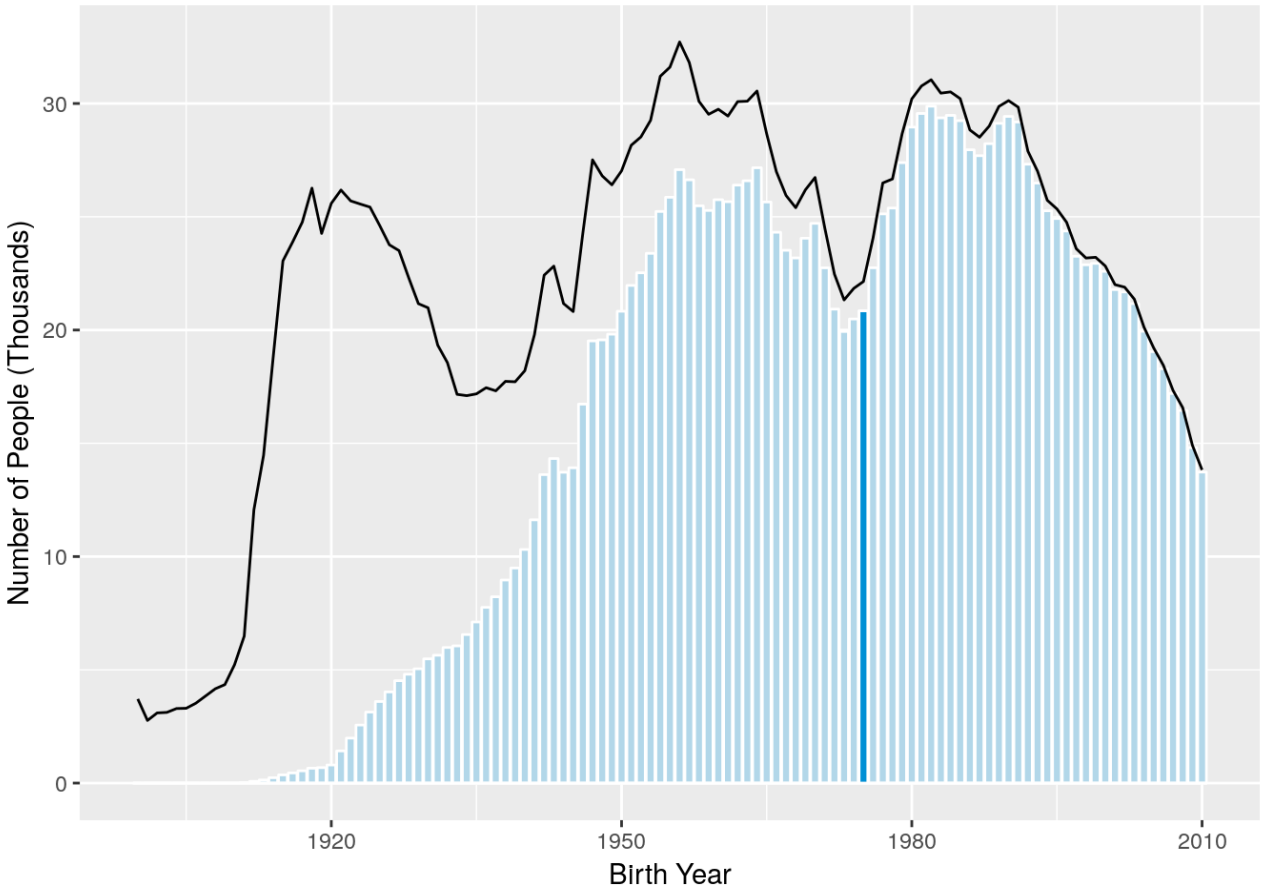
So the median male Joseph in this dataset was born in 1975.

Overplotting a special bar

To add the darker blue bar at 1975, we can add a second `geom_bar` to the plot. Since the x axis is mapped globally to the `year` variable, we need a trick to have ggplot draw just one bar. Namely, we will define a variable which is equal to `est_alive_today` if `year == median.birthyear`, and zero otherwise, and map this variable to the y axis. Technically this will draw a whole new set of bars, but all but the one we care about will have height zero.

We can define this variable using the `ifelse()` function.

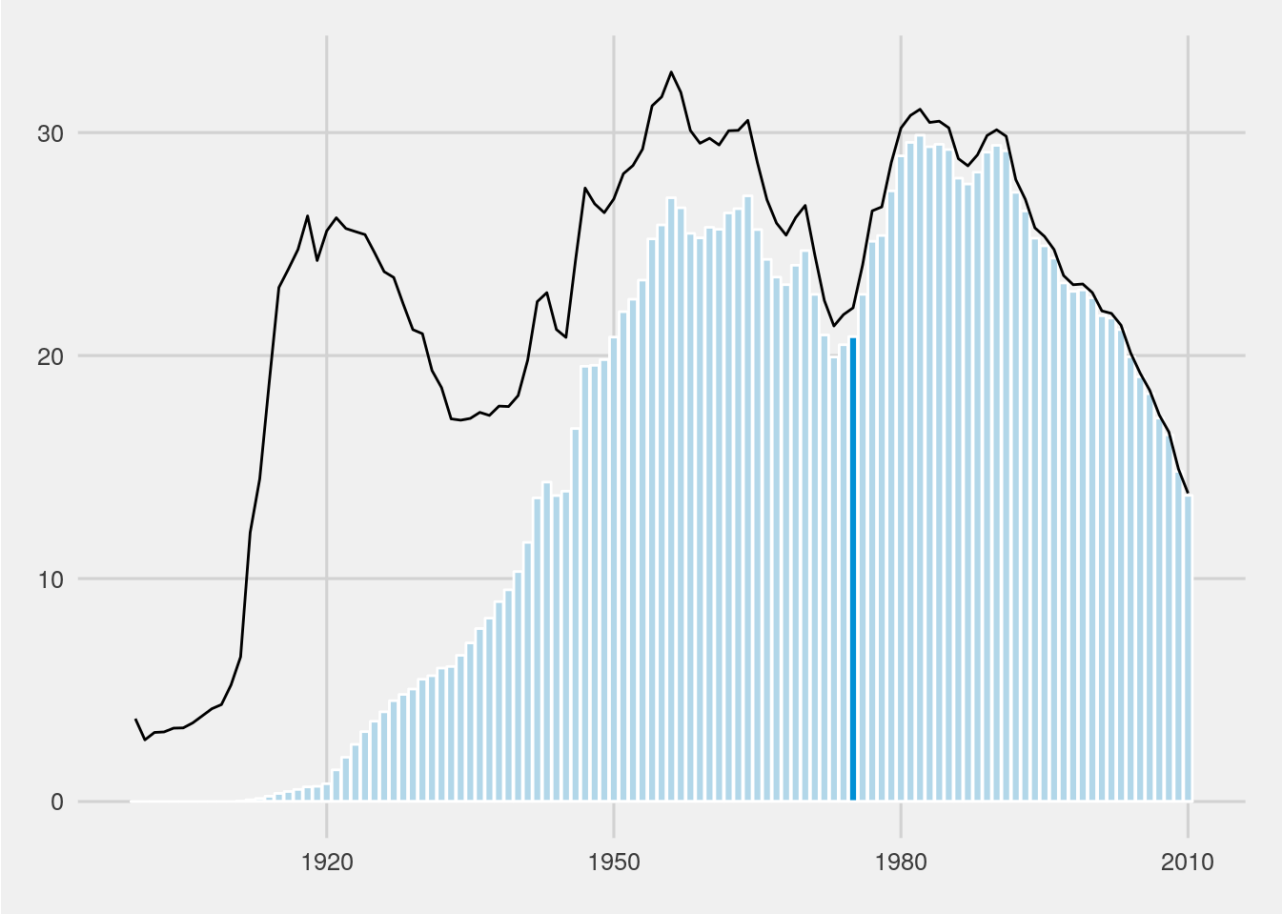
Code



Customizing the “look”

Since this plot came from FiveThirtyEight, let’s see how close we can get to the look of the original by applying the `fivethirtyeight` theme from `ggthemes`.

Code

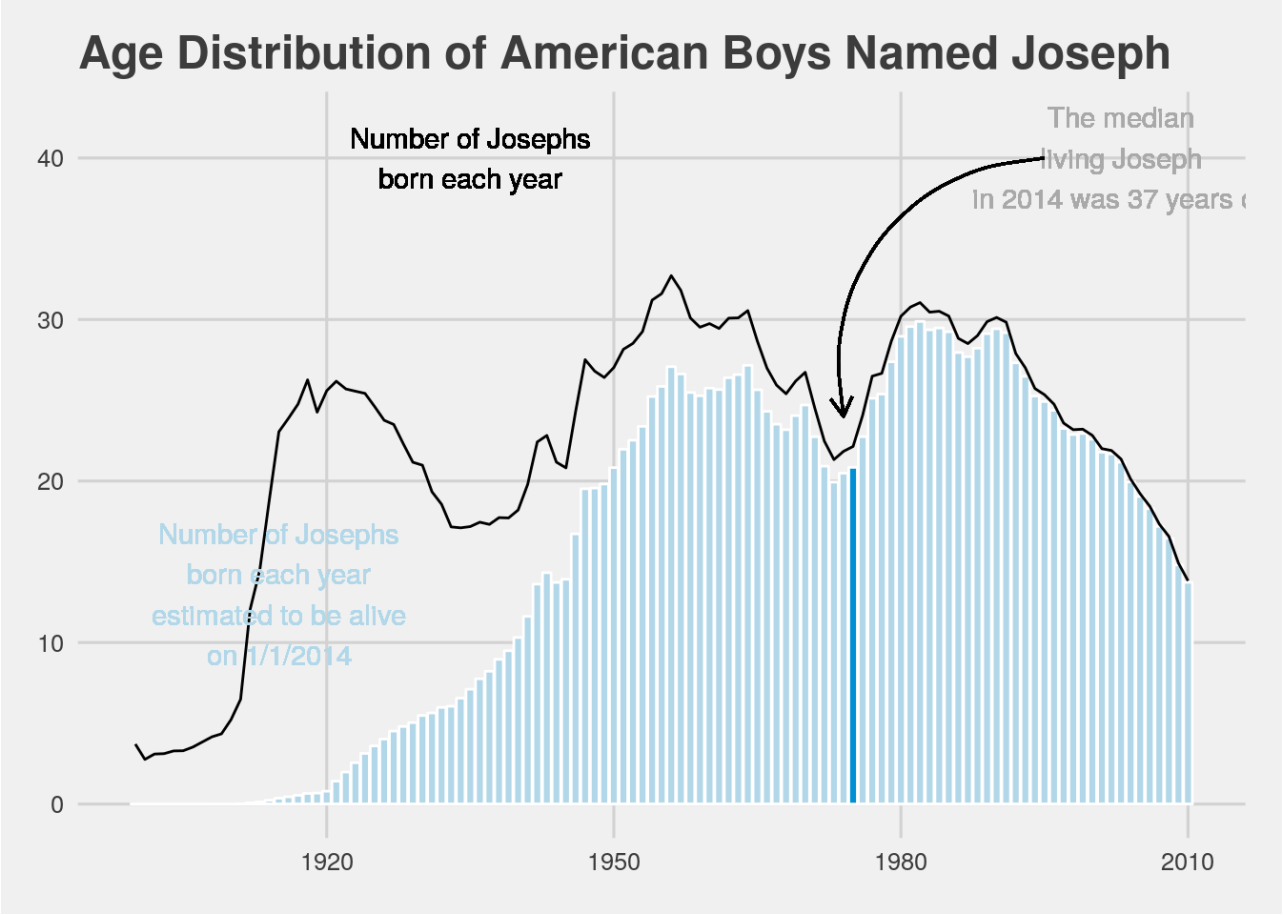


Evidently this theme removes the y axis label, which I'm not crazy about, though it is closer to the original. If we're going to do without a y axis label, we'd better be sure to supply that context elsewhere.

Adding context and annotation

We can add a title and some text boxes with `ggtitle()` and `geom_text()`. We can draw the annotation arrow with `geom_curve()` and just hardcode the locations of the endpoints.

Code

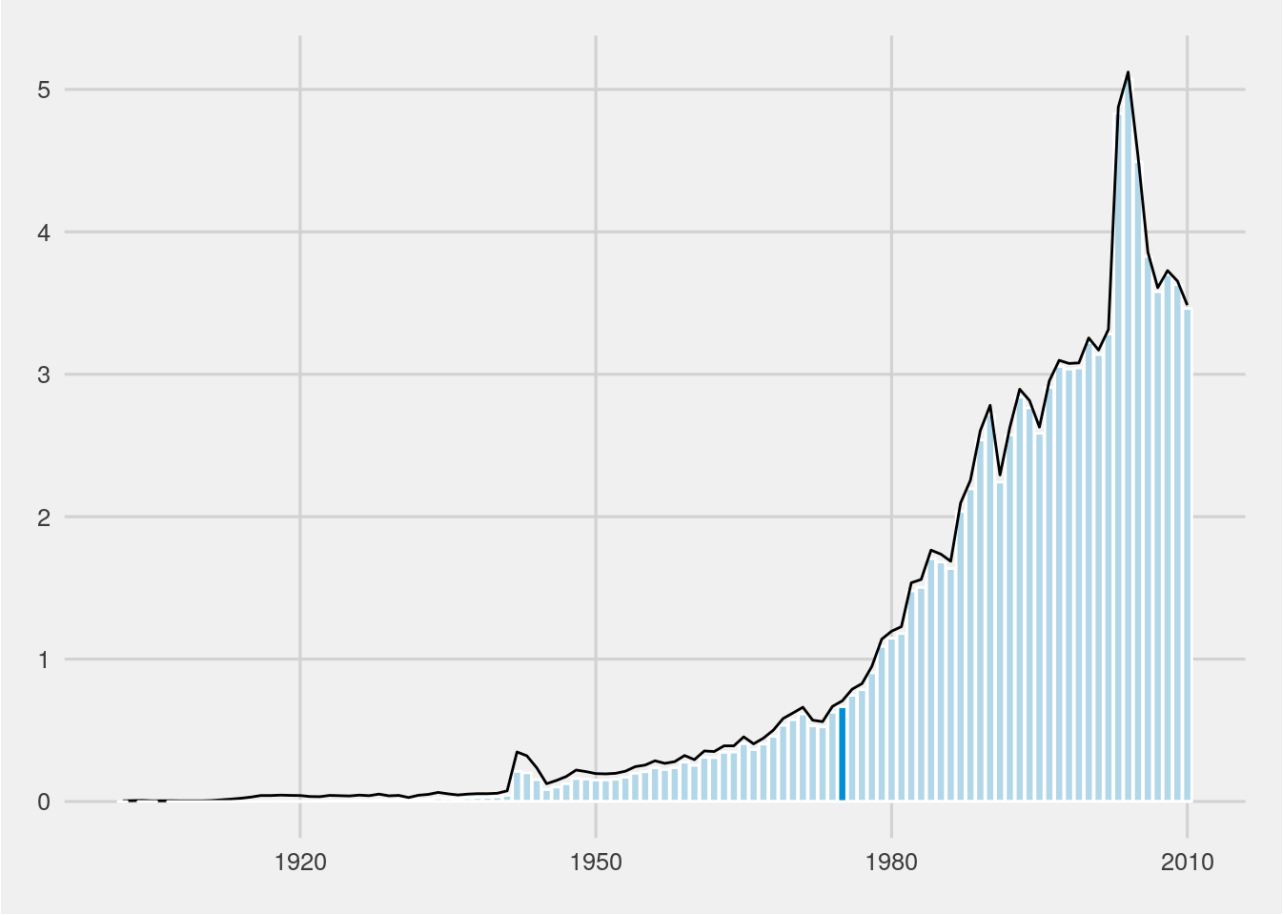


You can find out more about how to use functions like `geom_curve()` and `arrow()` using R's help interface. Remember that practicing statisticians and data scientists usually don't have all the code they use memorized; even pros have to look things up *constantly*.

Subbing in new data

A cool feature about `ggplot`s is that they enable us to define the structure of a plot once and "swap in" a new dataset to get the analogous plot. For example, if we are interested in creating an analogous plot for males with the name "Colin" (a fine name, in my opinion), we can use the `%+%` operator as follows:

Code



It seems like Colin is such a youthful name that nearly everyone who has ever been born with that name in the U.S. is likely still alive.
Note that I started with the plot that didn't have all of the extra context and annotations, since that was all specific to the name Joseph.

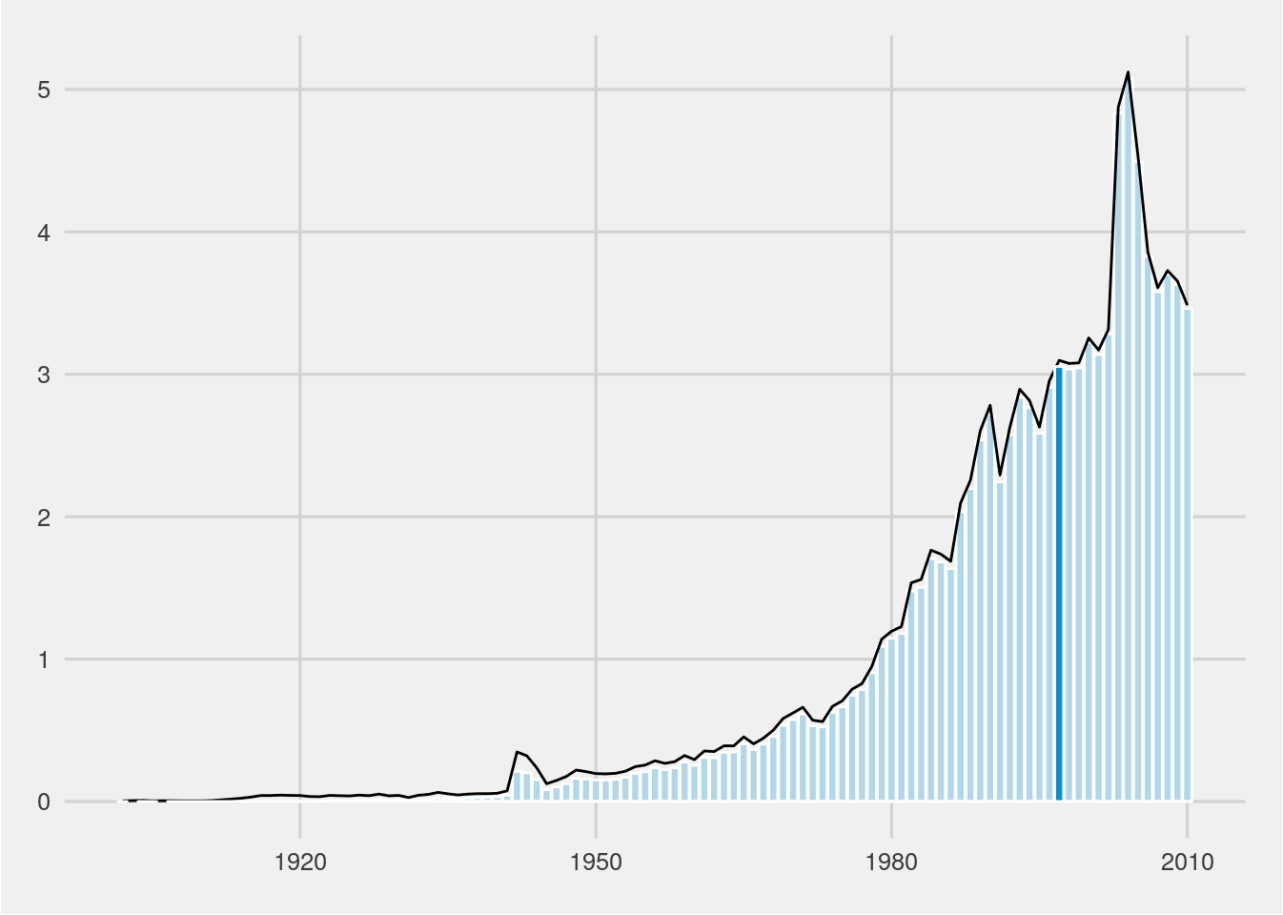
In fact, something is wrong here. Do you know what it is?

Spoiler

Code

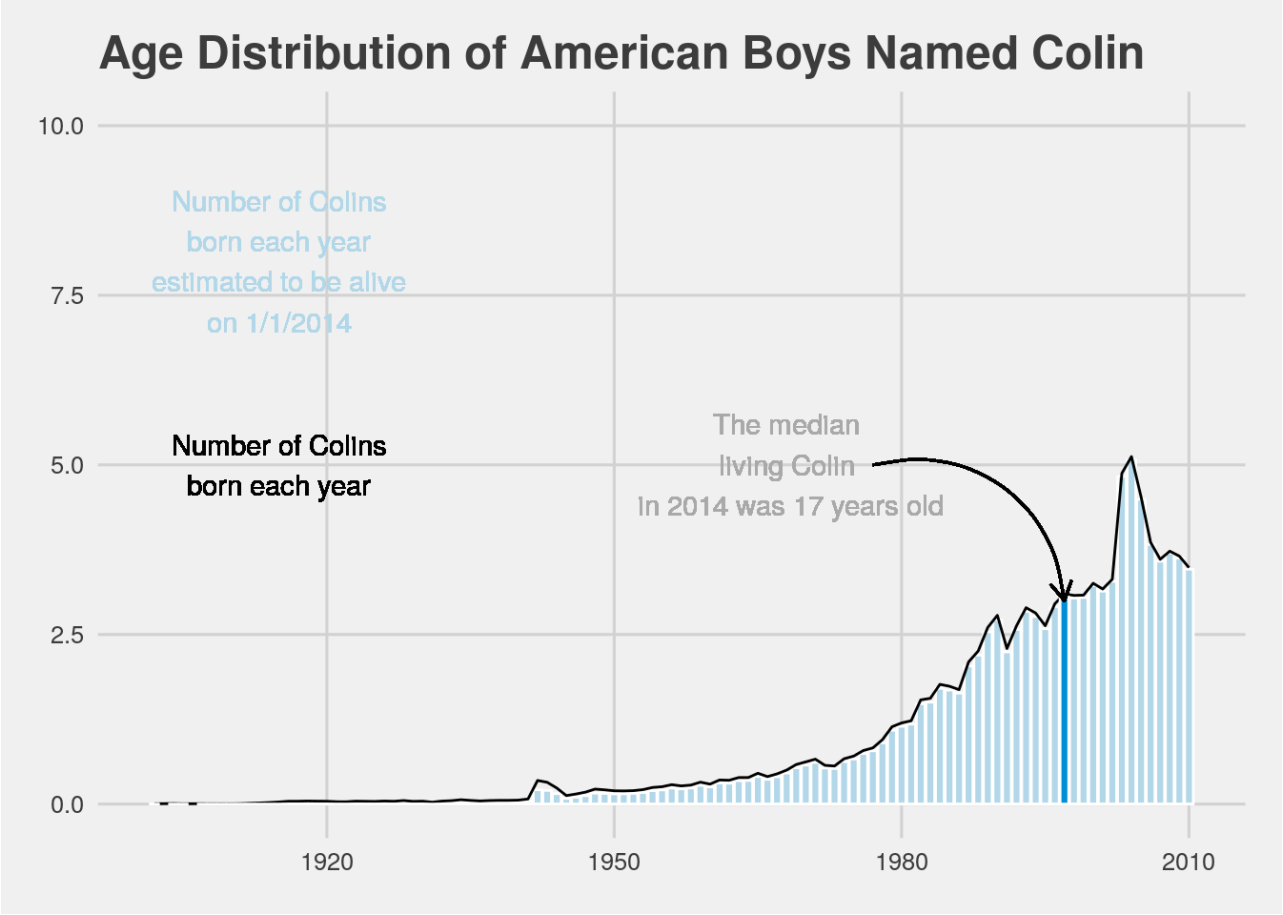
Fortunately, since we defined our highlighting condition using a variable name, all we need to do is update the value of this variable before we generate the plot.

Code



Let’s add some annotation. It takes some doing to get the locations of things right, especially since Colin is so much less common than Joseph, so the scale of the plot is quite different.

Code

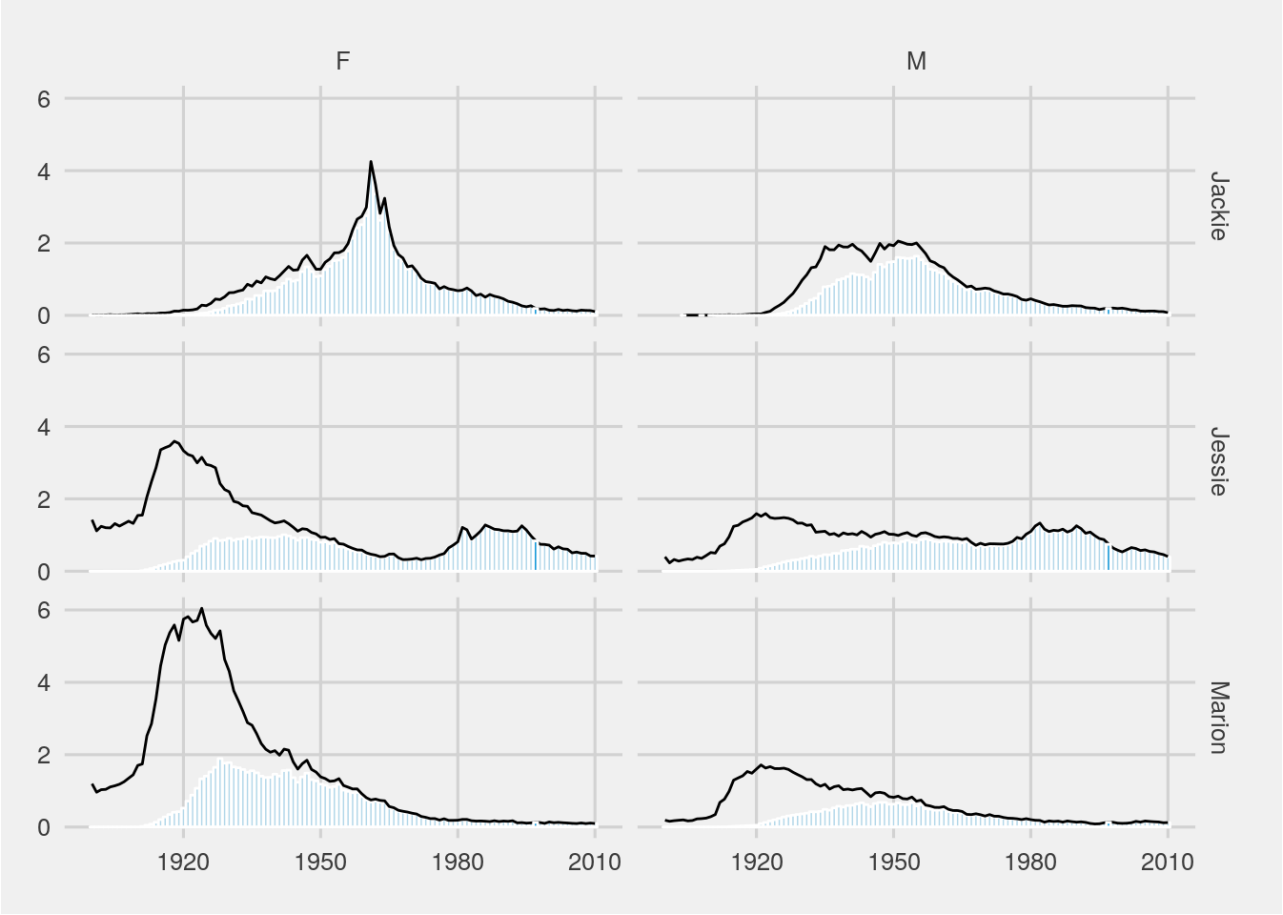


Exercise 2 Create a similar plot for your own name (if it's in the database – if not, pick another name you like) and post it to `#lab4` on Slack.

Adding some comparisons

If we are interested in comparing, say, the sex breakdown of names that are common for both male and female babies, we can easily add some facets to our plot. For example, the three most "unisex" names throughout this historical period have been "Jessie", "Marion", and "Jackie". But the gender connotation of names is not constant over time, so let's examine for these names how common each name was for each sex over time.

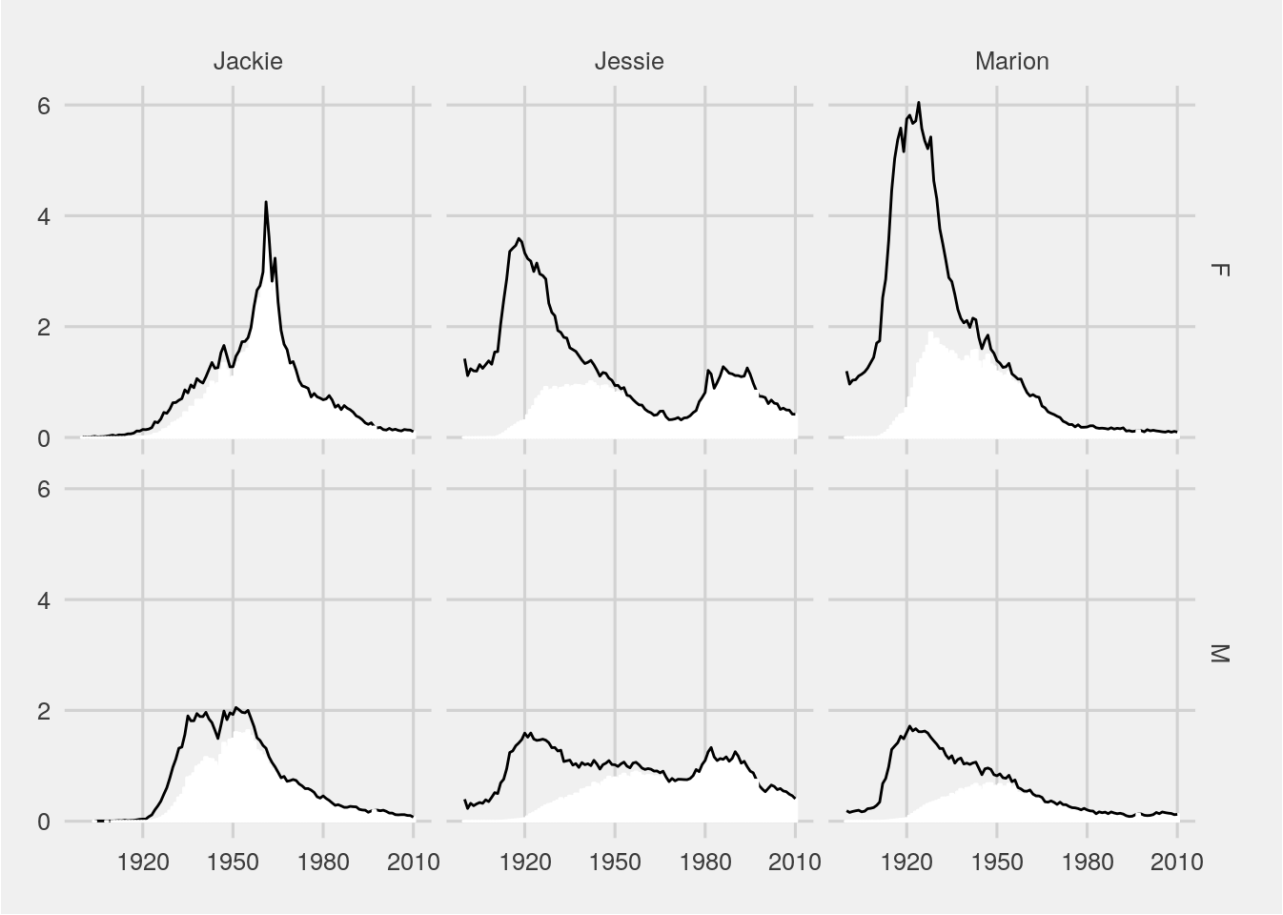
Code



(Note, I haven't bothered to fix the median highlight here; it's still showing the median for Colin)

This plot makes it easy to compare across names within a sex for a particular year, since year is lined up within a column, but we are more interested in comparing between sexes for a particular name. So let's reverse which variable defines the rows and which defines the columns.

Code



Exercise 3 What do you take away from this plot?

Exercise 4 Examine the other variables available in this dataset, and use faceting to explore a comparison of interest. Post your plots to the `#lab4` channel along with a short takeaway.

Exercise 5 What did you find interesting/challenging/confusing about this process? Post your response on `#lab4` .

This lab was adapted by Colin Dawson for STAT 209: Data Computing and Visualization, at Oberlin College, from a similar lab by Jordan Crouser and Ben Baumer used for SDS 192: Introduction to Data Science, at Smith College.