AE 18: Visualizing increased polarization of baby names

Suggested answers

APPLICATION EXERCISE ANSWERS

MODIFIED
April 9, 2025

```
library(tidyverse)
library(gganimate)
library(ggbeeswarm)
library(scales)

theme_set(theme_minimal())

# colors for the Democratic and Republican parties
dem <- "#00AEF3"
rep <- "#E81B23"
```

In this application exercise we will use animation and the {gganimate} package to visualize the increasing polarization of baby names in the United States.

Import data

Our data comes from the Social Security Administration which publishes detailed annual data on all babies born in the United States. We have prepared it by matching state-level births to the results of the 2024 U.S. presidential election so we can distinguish "red states" (those won by the Republican candidate Donald Trump) from "blue states" (those won by the Democratic candidate Kamala Harris). The data is stored in `data/partisan-names.csv`.

```
partisan_names <- read_csv(file = "data/partisan-names.csv") |>
  # convert year to a factor column for visualizations
  mutate(year = factor(year))
partisan_names
```

```
# A tibble: 200 × 7
   outcome sex   year  name     Trump Harris part_diff
   <chr>   <chr> <fct> <chr>    <dbl>  <dbl>     <dbl>
 1 Trump   M     1983  Kendrick 0.942 0.0580     0.884
 2 Trump   M     1983  Trey     0.932 0.0682     0.864
 3 Trump   M     1983  Rodrick  0.927 0.0732     0.854
 4 Trump   F     1983  Ashlea   0.917 0.0833     0.833
 5 Trump   F     1983  Tosha    0.890 0.110      0.780
 6 Trump   F     1983  Misti    0.886 0.114      0.773
 7 Trump   F     1983  Latoria  0.883 0.117      0.767
 8 Trump   M     1983  Jackie   0.876 0.124      0.753
 9 Trump   M     1983  Demarcus 0.870 0.130      0.740
10 Trump   F     1983  Angelia  0.869 0.131      0.737
# ℹ 190 more rows
```

We have aggregated the data to show the percentage of babies with a given name born in states that voted for Trump and Harris. The data contains the following columns:

- `outcome` - winner of the state-level vote in the 2024 U.S. presidential election
- `sex` - sex assigned at birth to the baby
- `year` - year of birth
- `name` - name of the baby
- `Trump` - percentage of the babies with the given name born in states that voted for Donald Trump in 2024
- `Harris` - percentage of the babies with the given name born in states that voted for Kamala Harris in 2024
- `part_diff` - difference between the percentage of babies with the given name born in Trump states and Harris states. Positive values indicate the name is more common in Trump states, whereas negative values indicate the name is more common in Harris states.

We have filtered the data to focus on the years 1983, 1993, 2003, 2013, and 2023 (the most recent year of births for which data is available), and also limited the data to the 20 most-highly partisan names for red states and blue states (i.e. each year contains 40 rows - the top 20 most "Republican" names and the top 20 most "Democratic" names).

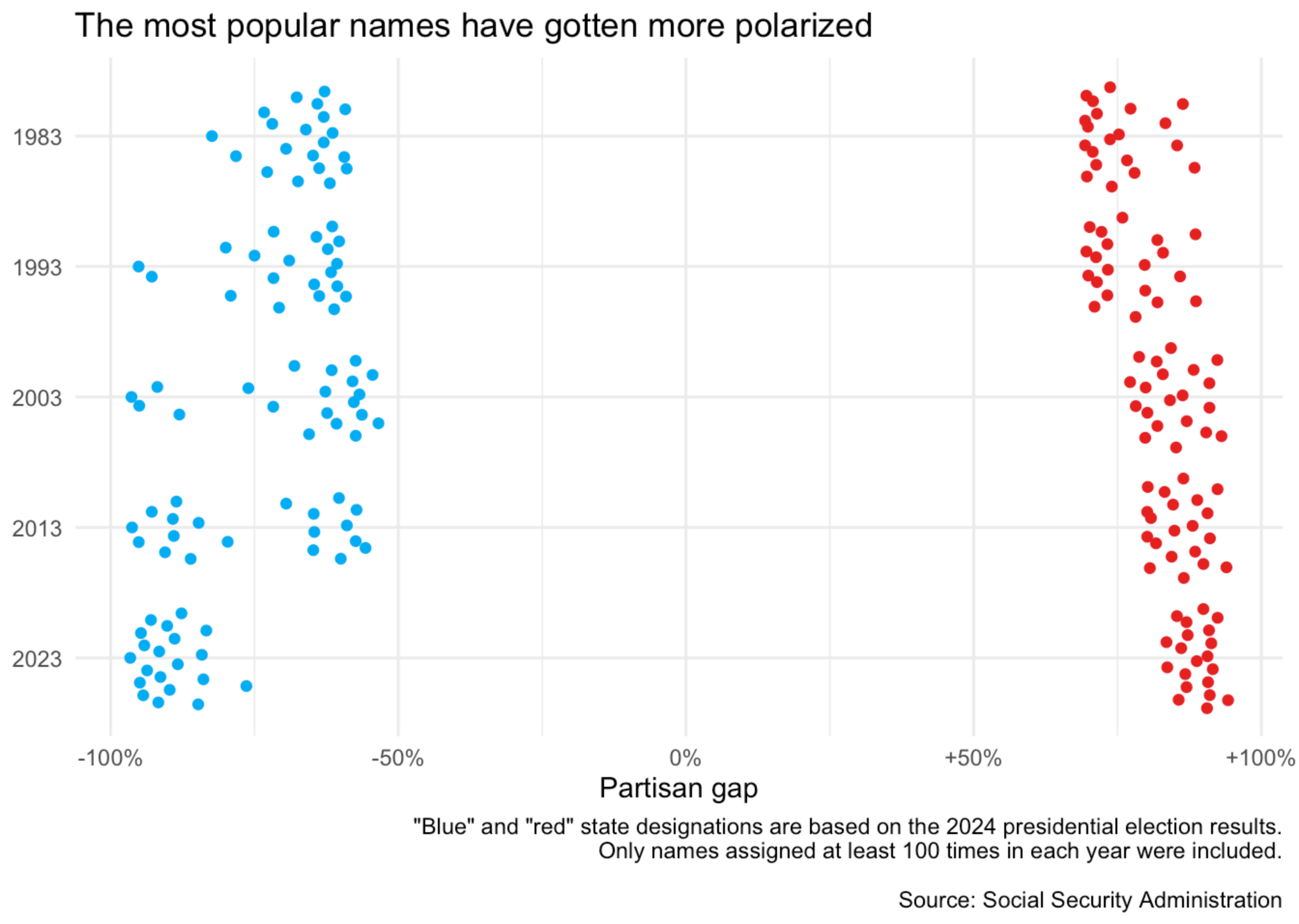
Create a static plot

Before we attempt to animate a chart, let's first create a single **static** visualization that communicates the change in polarization over time.

Your turn: Implement a jittered plot that shows the distribution of partisanship for names in red states and blue states for each year. Use the `geom_quasirandom()` function from the {ggbeeswarm} package to create the plot.[1]

```
ggplot(
  data = partisan_names,
  mapping = aes(
    x = part_diff,
    y = fct_rev(year),
    color = outcome
  )
```

```r
) +
  geom_quasirandom() +
  scale_x_continuous(labels = label_percent(style_positive = "plus")) +
  scale_color_manual(values = c(dem, rep), guide = "none") +
  labs(
    title = "The most popular names have gotten more polarized",
    x = "Partisan gap",
    y = NULL,
    caption = '"Blue" and "red" state designations are based on the 2024 presidential
          election results.\nOnly names assigned at least 100 times in each year were
          included.\n\nSource: Social Security Administration'
  )
```



Define plot components for the animated chart

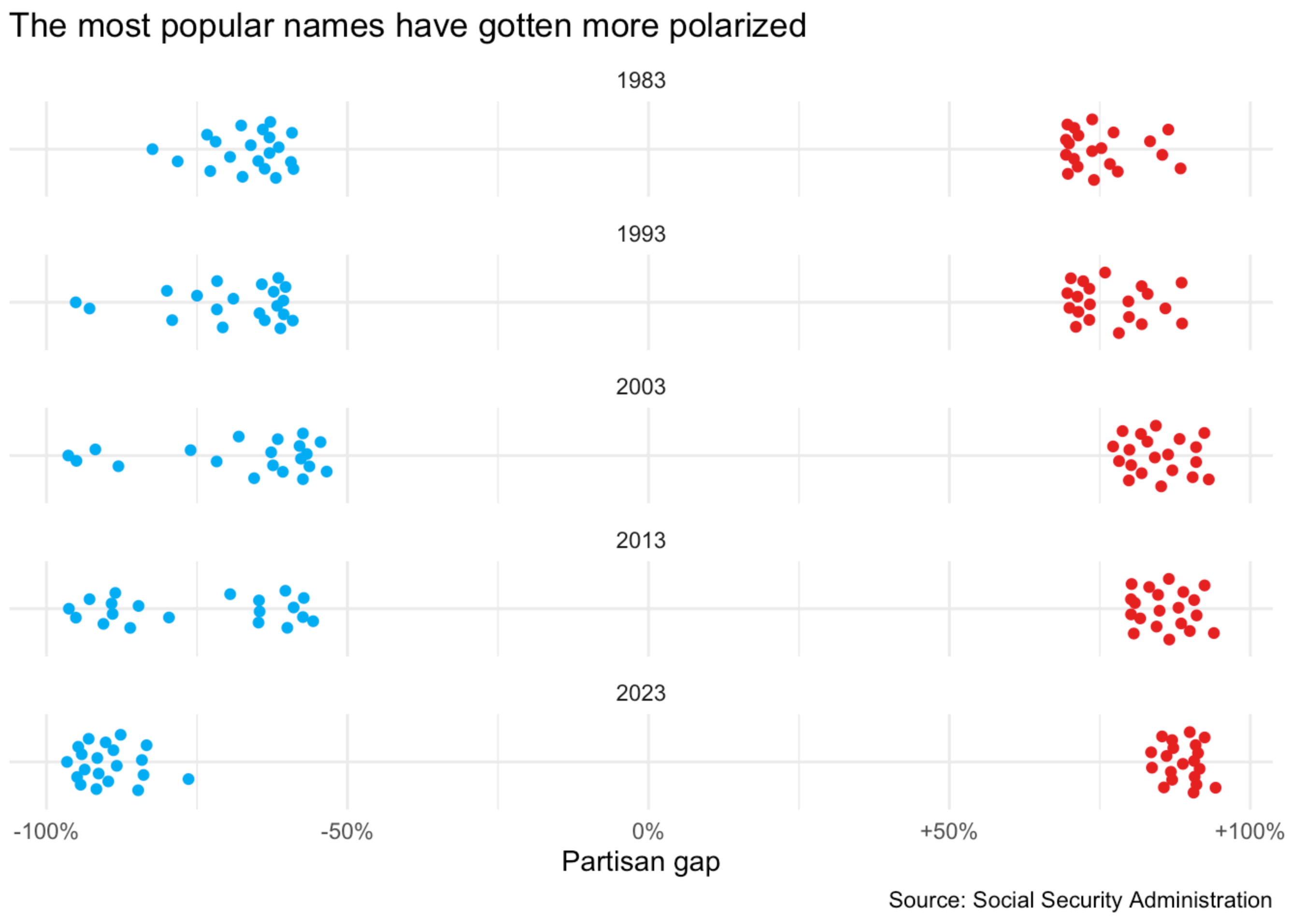
Before we attempt to animate the chart, let's first define the components of the chart that will remain static throughout the animation.

Your turn: Modify your plot above to have a single category on the y-axis. This is what will change throughout the animation.

Helpful hint

When we animate the chart, we will define each **frame** based on the `year` variable. When testing the static portions of the chart, you can facet the graph on the `year` variable to see how the chart will look for each year. When we animate it, each facet panel will essentially become the frames of the animation.

```r
ggplot(
  data = partisan_names,
  mapping = aes(
    x = part_diff,
    y = "1",                                                            1
    color = outcome
  )
) +
  geom_quasirandom() +
  scale_x_continuous(labels = label_percent(style_positive = "plus")) +
  scale_color_manual(values = c(dem, rep), guide = "none") +
  labs(
    title = "The most popular names have gotten more polarized",
    x = "Partisan gap",
    y = NULL,
    caption = "Source: Social Security Administration"
  ) +
  theme(                                                                2
    axis.text.y = element_blank()
  ) +
  facet_wrap(facets = vars(year), ncol = 1)                             3
```



Implement basic animation

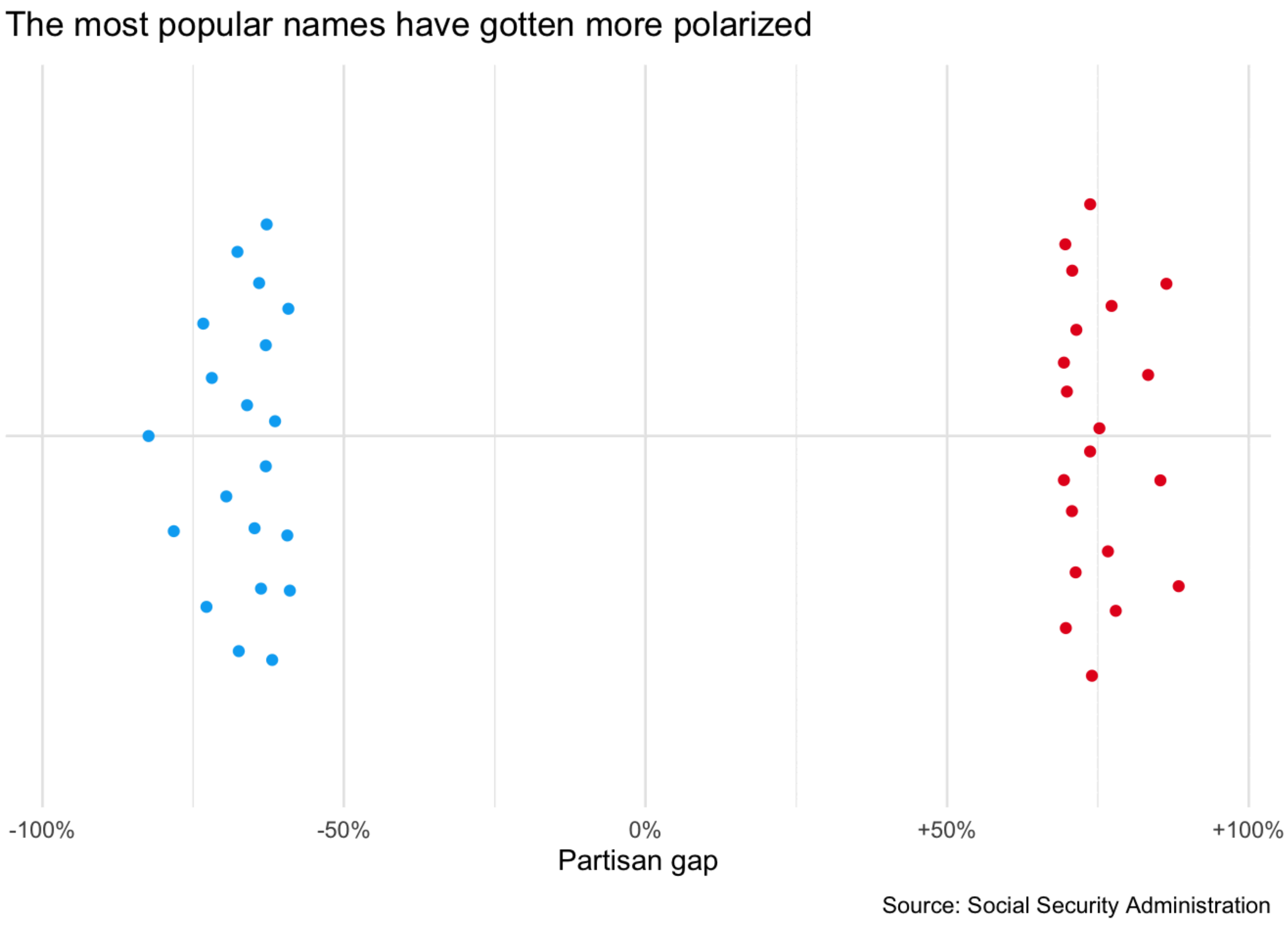
Now that we have the static components of the chart, we can animate it. The {gganimate} package provides a simple way to animate a plot by defining the `transition_*()` function.

Your turn: Implement the animation using the appropriate `transition_*()` function.

```
# store the basic plot to be reused for multiple animated charts
p_partisan <- ggplot(
  data = partisan_names,
  mapping = aes(
    x = part_diff,
    y = "1",
    color = outcome
  )
) +
  geom_quasirandom() +
  scale_x_continuous(labels = label_percent(style_positive = "plus")) +
  scale_color_manual(values = c(dem, rep), guide = "none") +
  labs(
```

```r
    title = "The most popular names have gotten more polarized",
    x = "Partisan gap",
    y = NULL,
    caption = "Source: Social Security Administration"
  ) +
  theme(
    axis.text.y = element_blank()
  )
```

```r
p_partisan +
  transition_states(states = year)
```

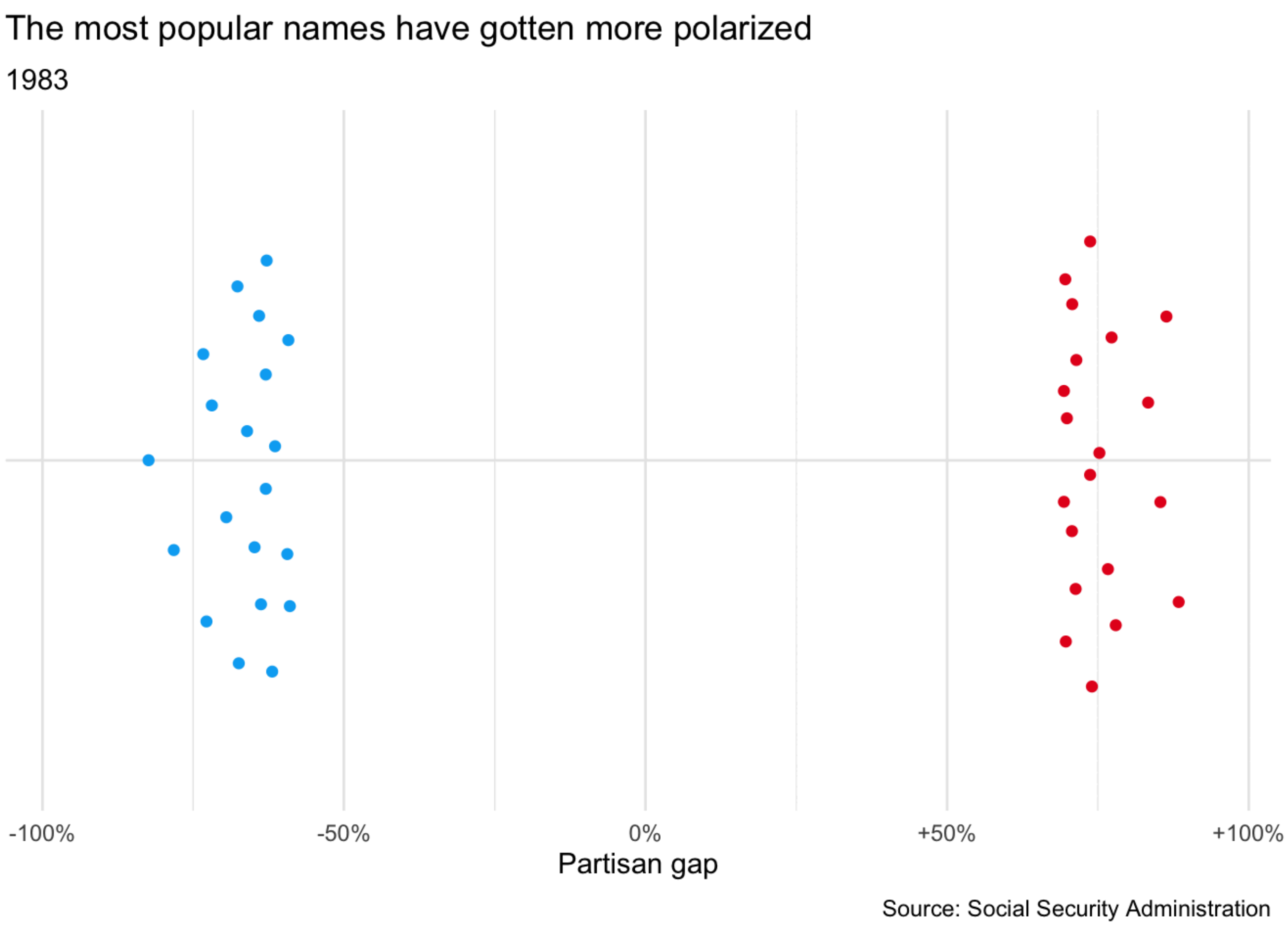


Your turn: We need to know what years the points are transitioning between. Add this using an appropriate label to the plot.

How do we know the current year?

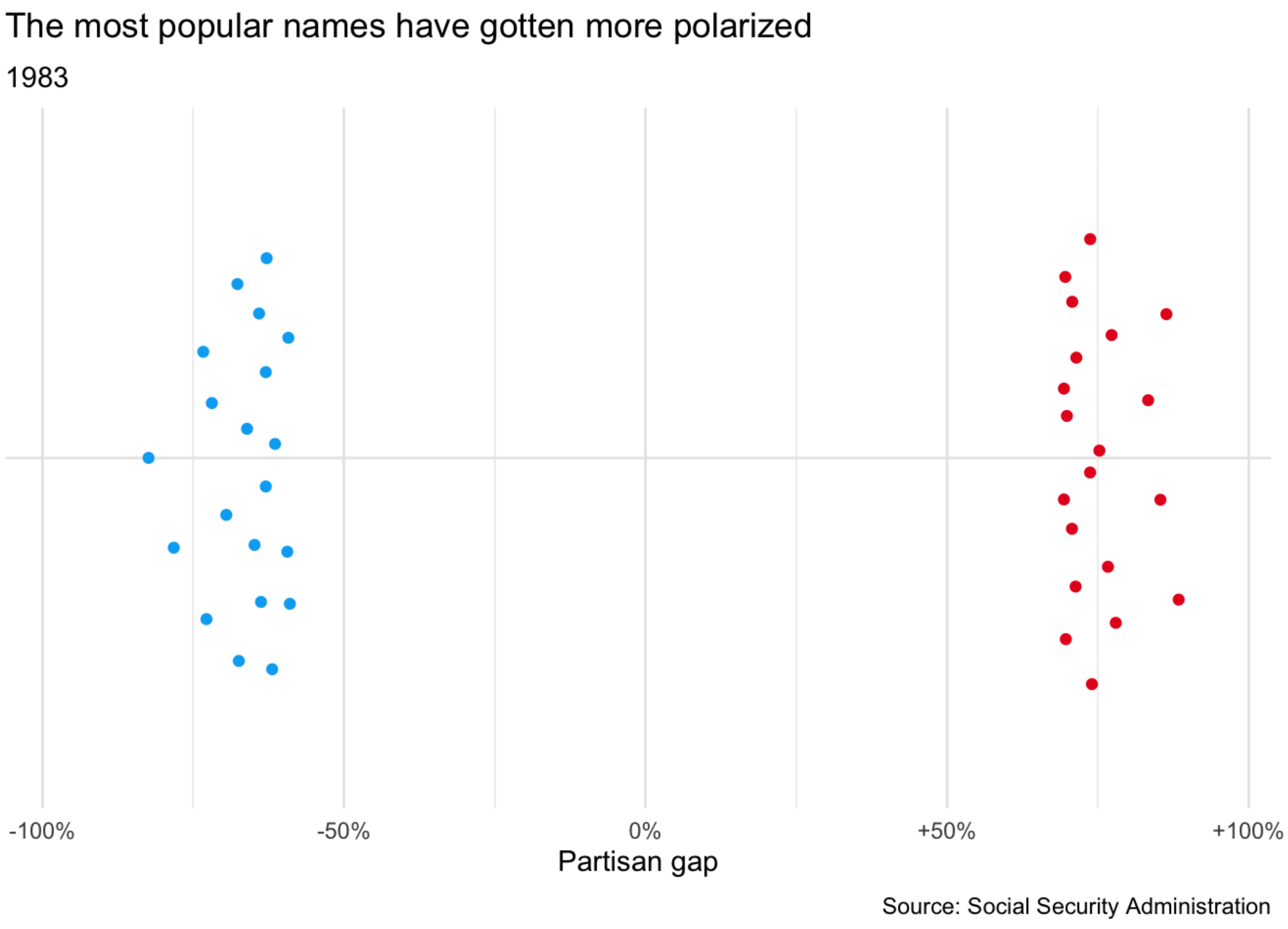
Look at the documentation for your `transition_*()` function. It should provide information on how to add labels to the animation based on the transitioning states.

```r
p_partisan +
  transition_states(states = year) +
  labs(subtitle = "{closest_state}")
```



Your turn: Adjust the animation to make it smoother. Consider adjusting appropriate parameters in the `transition_*()` function as well as the easing used for interpolation via `ease_aes()`.

```r
p_partisan +
  transition_states(
    states = factor(year),
    # devote twice as many frames to the pause at the states
    transition_length = 1,
    state_length = 2
  ) +
  labs(subtitle = "{closest_state}") +
  # use a cubic-in-out easing function
  ease_aes("cubic-in-out")
```

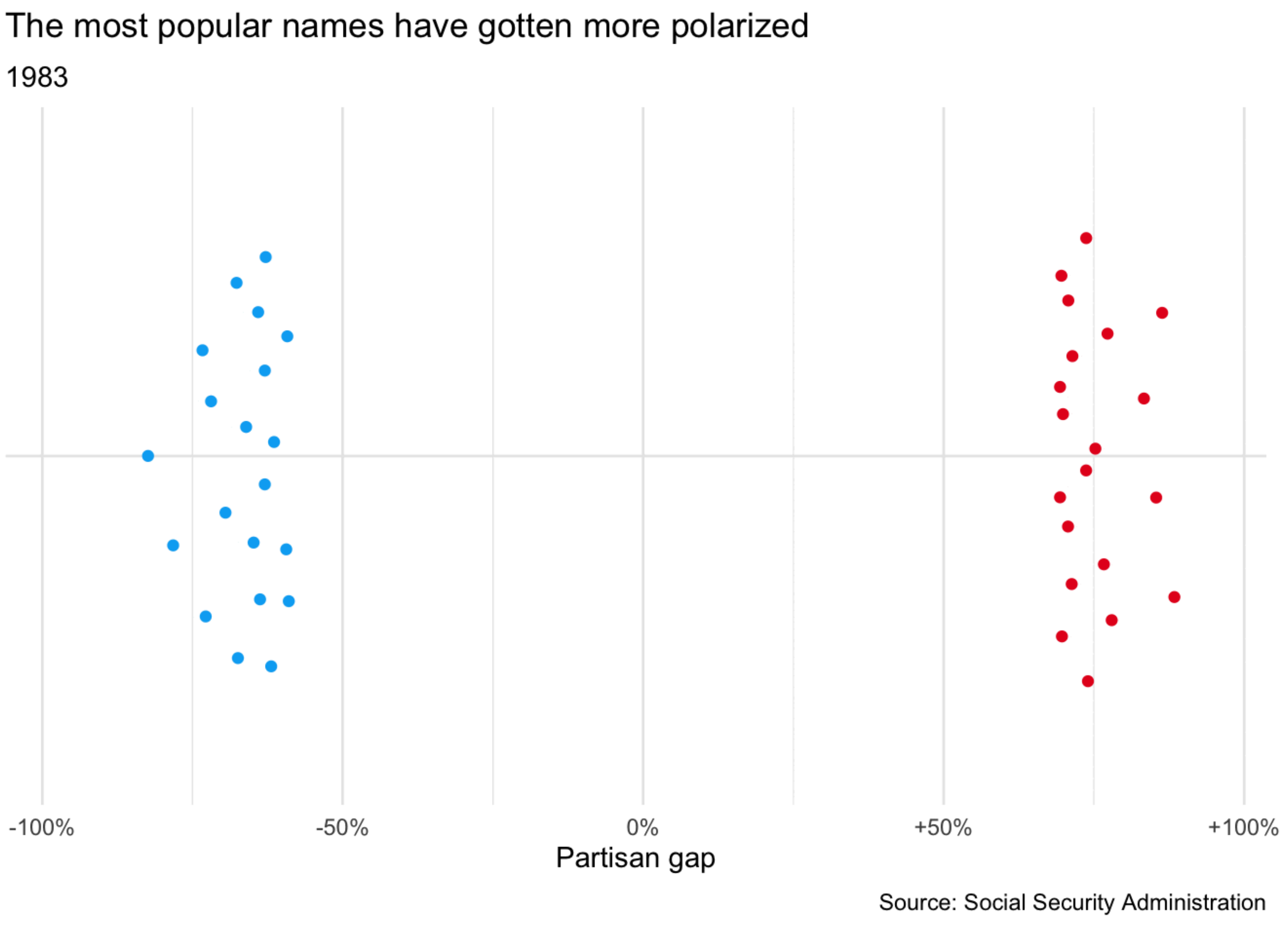


Add shadows

To make the animation easier to interpret, it's helpful to add reference marks during the transition to show where the points are moving from and to. This can be done using the `shadow_*()` functions.

Your turn: Implement a `shadow_*()` function to show the transition more smoothly.

```
p_partisan +
  transition_states(
    states = factor(year),
    # devote twice as many frames to the pause at the states
    transition_length = 1,
    state_length = 2
  ) +
  labs(subtitle = "{closest_state}") +
  # use a cubic-in-out easing function
  ease_aes("cubic-in-out") +
  shadow_wake(wake_length = 0.05)
```



Rendering

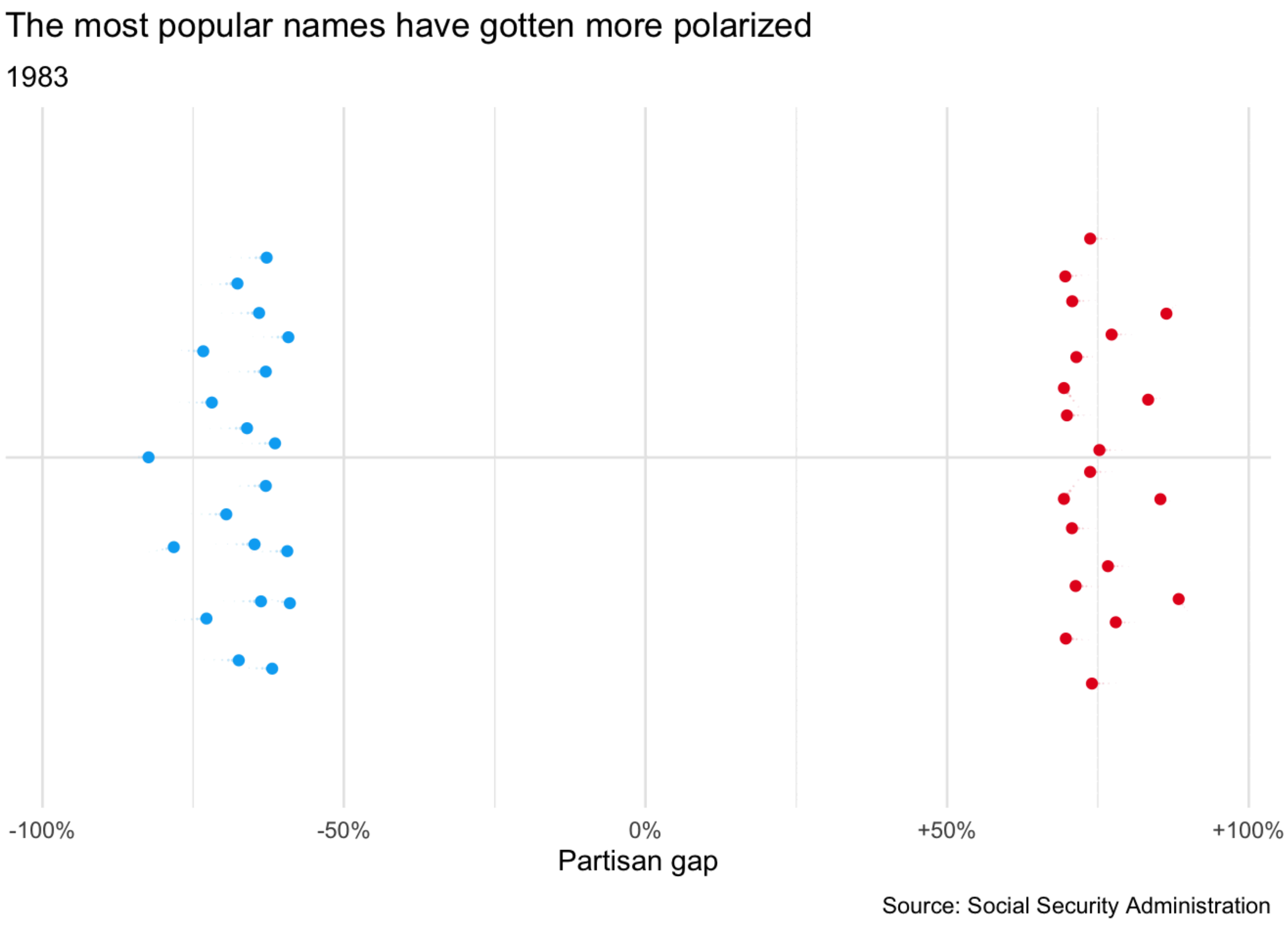
We can control the rendering process using the `animate()` function.

Your turn: Improve the animated chart through it's rendering. Some suggestions include:

- Increase the number of frames to make the animation smoother.
- Increase the length of the animation to give the reader more time to interpret each frame.
- Add a pause at the start and end of the animation to give the reader time to interpret the chart.

```
p_final <- p_partisan +
  transition_states(
    states = factor(year),
    # devote twice as many frames to the pause at the states
    transition_length = 1,
    state_length = 2
  ) +
  labs(subtitle = "{closest_state}") +
  # use a cubic-in-out easing function
  ease_aes("cubic-in-out") +
  shadow_wake(wake_length = 0.05)

# smoother transition
animate(p_final, duration = 20, fps = 20, start_pause = 30, end_pause = 30)
```



Your turn: Implement the same settings using a Quarto code chunk option. Adjust the aspect ratio of the plot to make it more compact vertically.

Implementing the `gganimate` code chunk option

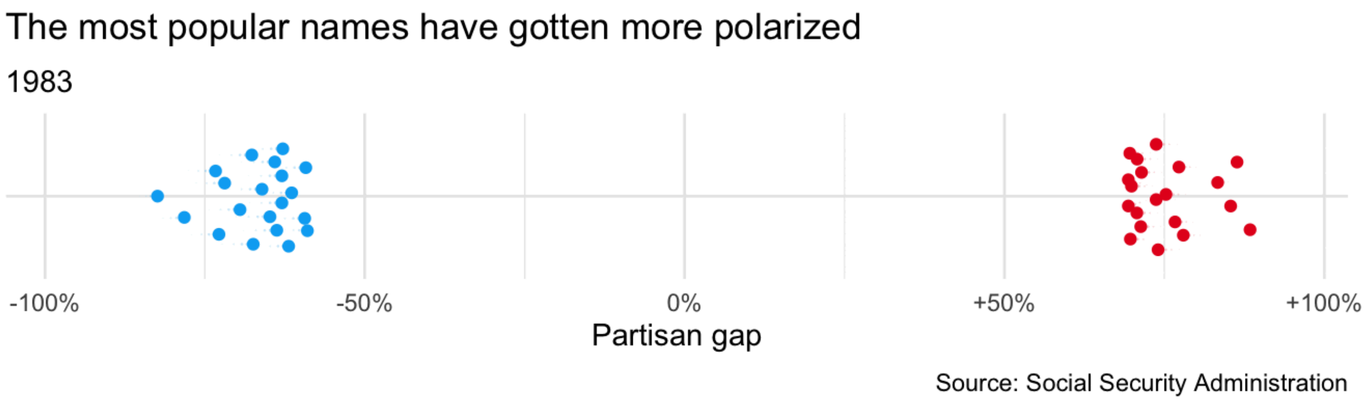
Quarto can pass arbitrary R expressions through code chunk options using the syntax `!expr`. For example, if we wanted to render the animation with 300 frames at 15 frames per second, we could use the following code chunk option:

```
#| gganimate: !expr list(nframes = 300, fps = 15)
```

~~~
```{r}
#| label: render-animation-quarto
#| dependson: render-animation
#| fig-asp: 0.3
~~~

```
#| gganimate: !expr list(nframes = 300, fps = 25, start_pause = 10, end_pause = 10)

p_final
```
```



Acknowledgments

- Application exercise inspired by Philip Bump's *How to Read This Chart* March 29th newsletter

Session information

Footnotes

1. Unlike `geom_jitter()`, this will ensure there are no overlapping points in the plot. You can also use `geom_beeswarm()` but this introduces unnecessary curvatures into the jittering. ↩

Made with and Quarto.
All content licensed under CC BY-NC 4.0.
```