Data Visualization for Black Lives: Interactive Workshop

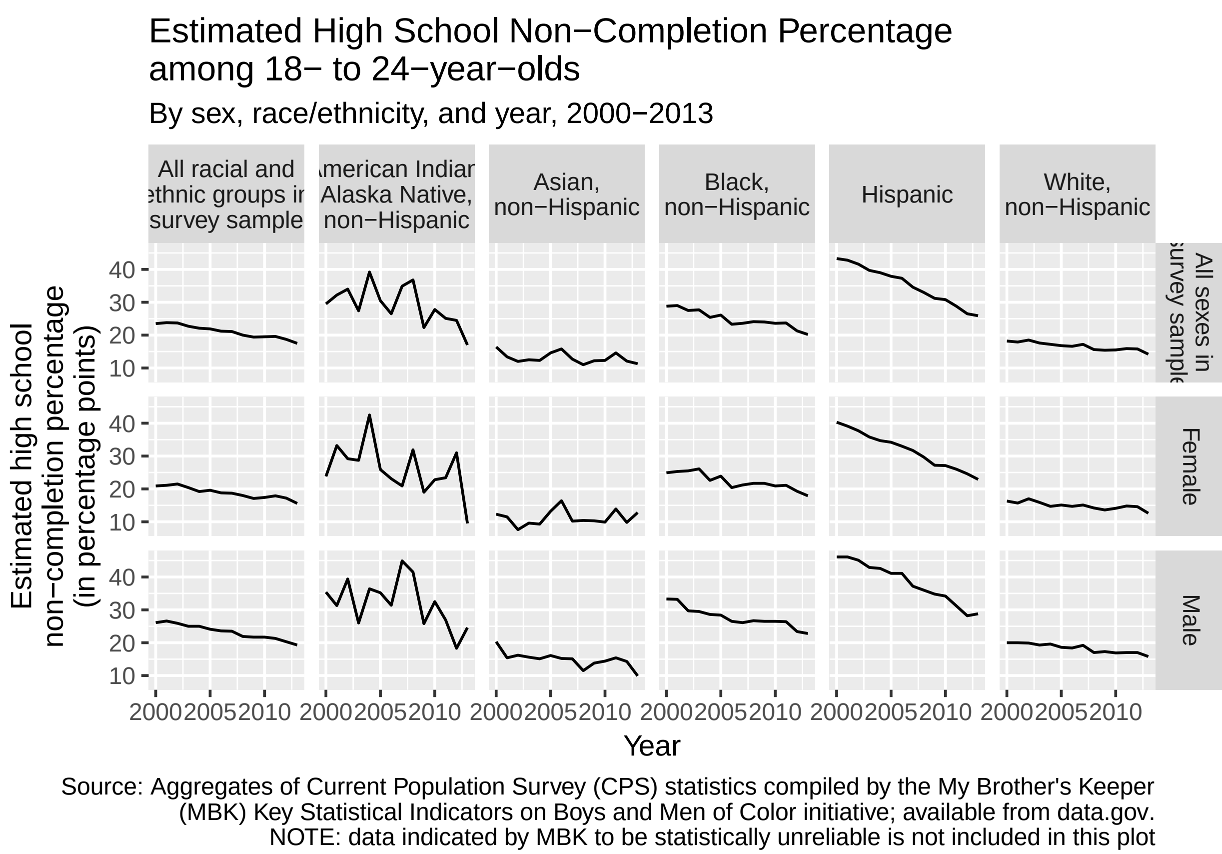
Cameron Clarke and Brecia Young

January 12, 2019

0. Preface

Welcome to the interactive part of the Data Visualization for Black Lives workshop! In this hands-on session we'll work our way up to making plots like this one:

```
ggplot2::ggplot(data = full_data_df) +
  ggplot2::geom_line(mapping = aes(x = year, y = percentage_estimate)) +
  ggplot2::facet_grid(sex_plot_label ~ race_ethnicity_plot_label) +
  ggplot2::xlab("Year") +
  ggplot2::ylab("Estimated high school\nnon-completion percentage\n(in percentage points)") +
  ggplot2::labs(title = "Estimated High School Non-Completion Percentage\namong 18- to 24-year-olds",
                subtitle = "By sex, race/ethnicity, and year, 2000-2013",
                caption = "Source: Aggregates of Current Population Survey (CPS) statistics compiled by
```



First, to orient ourselves, we'll go through some parts of the R programming lanugage that will be foundational for the visualizations we'll do. Then we'll briefly get to know the particular dataset we'll want to visualize: an open and publicly available dataset from the My Brother's Keeper Key (MBK) initiative. We'll finish off with a tour of some of the main parts of the `ggplot2` package, an increasingly popular package and framework for doing data visualizations in R. We'll work our way up to building the visualization shown above, highlighting some of the flexibility that the `ggplot2` allows in buiding plots layer by layer.

We also include some of the data cleaning work needed to get the data in plot-ready form in the Appendix at the end. We won't go through that in depth, but will explain along the way why certain data cleaning steps are needed and how they relate to the kinds of visualizations we might want to do.

1. A Brief Introduction to R Programming

1.1 Assignment, packages, functions

Below, we're going to make use of the R language. Here are some specific parts we'll use

- `<-`: this is the primary *assignment operator* in R. We use this to assign a *value* to a *name*. For example, `x <- 3` assigns the value 3 to `x`. Then, whenever we call `x`, we'll get 3 in its place.

```
# Example - Assigning a variable, y, the value of 2
y <- 2

# Exercise - Try assiging the value of 3 to variable y
```

- The syntax `<PACKAGE>::<FUNCTION>`, e.g., `dplyr::mutate()`, allows us to call a specific function from an R package that it may be a part of. Not all functions are part of a package; for example, functions you define directly from your RStudio interactive prompt will not immediately be part of a package. But, a main way that R tends to grow as a language is through innovations in packages, and so it is very common to encounter them when programming with R. (For more on packages, see Hadley Wickham's book "R Packages".)
  - A note here: it is not strictly necessary to specify a package name every time you want to use a function from that package. If you first call the `library()` function—e.g., `library(tidyverse)`—all functions from the specified package's *namespace* will loaded be *attached* on the search path that R uses when trying to parse user input. (For this reason, there may be performance reasons to not always use the `<PACKAGE>::<FUNCTION>` syntax; we do so in this workshop, though, for reasons of clarity.)
- In R it is possible to define your own functions. A function is basically an instruction, or a set of instructions, that is given a name. In this sense, functions are a kind of *value*; and not coincidentally, we use the assignment operator to give a function its name.
- `%>%`: this is the *pipe operator*. The core idea behind the pipe operator is to allow us to write code that is easier to read. It lets us chain together instructions (especially functions) in our code by letting us "pipe" the output of one function into another function as that second function's input.

1.2 Examples

Let's look at some examples of these things. First, let's define a function called `myfunc` which takes in two arguments concatenates them with a separator (here, a comma and a space) in between, and returns the resulting character string:

```r
# `myfunc`:
#   - input:  arg_1, arg_2 (must both be able to be cast as
#             character strings)
#   - output: the concatenation of arg_1 and arg_2 as characters,
#             with a comma and a space as a separator
myfunc <- function(arg_1, arg_2) {

  output <- paste0(arg_1, ", ", arg_2)

  return(output)

}
```

Combining some of the above information:

```r
myfunc(arg_1 = "a", arg_2 = "b")
```

```
## [1] "a, b"
```

```r
myfunc("a", "b")
```

```
## [1] "a, b"
```

```r
myfunc("b", "a")
```

```
## [1] "b, a"
```

```r
myfunc("b", a)
```

```
## Error in paste0(arg_1, ", ", arg_2): object 'a' not found
```

```r
myvar <- myfunc("a", "b")
myvar
```

```
## [1] "a, b"
```

```r
myvar_2 <- "a" %>% myfunc("b")
myvar_2
```

```
## [1] "a, b"
```

```r
myvar_3 <- "b" %>% myfunc("a")
myvar_3
```

```
## [1] "b, a"
```

```
myvar_4 <- "b" %>% myfunc("a", .)
myvar_4
```

```
## [1] "a, b"
```

```
myvar_5 <- "b" %>% myfunc(arg_1 = "a", arg_2 = .)
myvar_5
```

```
## [1] "a, b"
```

We'll make use of all of these R langauge constructs througout the workshop.

2. Loading in the data

First, we'll load in the data. This data comes from from the My Brother's Keeper Key (MBK) Statistical Indicators on Boys and Men of Color initiative. According to the U.S. Department of Education, the MBK data itself comes from the Current Population Survey (CPS). What we see in the file is likely a pre-aggregated version of the CPS data. (For more information on the CPS itself, including its questionnaire and methodology, see here: https://www.census.gov/programs-surveys/cps.html)

The description given of the data is as follows: > Percentage of 18- to 24-year-olds who have not completed high school by sex and race/ethnicity, 2000-2013

The original source for the data is here, but we've provided some pre-cleaned versions of that data. (We'll explain what "cleaning" the data means and looks like in this context, in the next section.) Go ahead and load those in using RStudio as follows (specifying the `readr::read_csv()` function as the one to use when reading it in:

- Go to File > Import Dataset > From Text (readr)
- In the prompt that pops up, use the "Browse" button to navigate to each of the files we've provided, one at a time
  - For the file "mbk_data_preprocessed_full.csv", use the "Import Options" section to specify the variable name `full_data_df` using the "Name:" option, and click "Import"
  - For the file "mbk_data_preprocessed_missing.csv", use the "Import Options" section to specify the variable name `missing_data_df` using the "Name:" option, and click "Import"

Note the *column specifications* indicated by the `readr::read_csv()` function: these column specifications tell you what types the resulting columns will be. (Often, it's a good idea to specify these types explicitly, but we don't do so in this case; see here for more information about what this would look like.)

3. Getting to know the data

3.1 A first look: `full_data_df`

Let's look at the data to see what we're working with:

```
full_data_df
```

```
## # A tibble: 252 x 12
##    sex   race_ethnicity sex_plot_label race_ethnicity_~  year
##    <chr> <chr>          <chr>          <chr>            <int>
##  1 All ~ All racial an~ "All sexes in~ "All racial and~  2000
##  2 All ~ All racial an~ "All sexes in~ "All racial and~  2001
##  3 All ~ All racial an~ "All sexes in~ "All racial and~  2002
##  4 All ~ All racial an~ "All sexes in~ "All racial and~  2003
##  5 All ~ All racial an~ "All sexes in~ "All racial and~  2004
##  6 All ~ All racial an~ "All sexes in~ "All racial and~  2005
##  7 All ~ All racial an~ "All sexes in~ "All racial and~  2006
##  8 All ~ All racial an~ "All sexes in~ "All racial and~  2007
##  9 All ~ All racial an~ "All sexes in~ "All racial and~  2008
## 10 All ~ All racial an~ "All sexes in~ "All racial and~  2009
## # ... with 242 more rows, and 7 more variables: percentage_estimate <dbl>,
## #   percentage_stderr_estimate <dbl>, note_on_percentage <chr>,
## #   count_estimate <dbl>, count_stderr_estimate <dbl>,
## #   note_on_count <chr>, has_relevant_NAs <lgl>
```

Tabbing through the above table, we can see if there's anything we notice (column names, number of rows, number of columns, the range of values in each column, missing values, etc.).

Some things we can note right away:

- The `sex` and `race_ethnicity` columns each follow a classificatory scheme that is inadequate in important ways. These schemes have their basis in the way that demographic information is asked about (or collected otherwise) on the CPS.
- The `sex`, `race_ethnicity`, and `year` columns together define the relevant *subgroups* of our data.
- Each choice of values for these three variables that define the relevant subgroups is "related" to a specific value of `percentage_estimate` and `percentage_stderr_estimate`.

3.2 Missing data in `missing_data_df`

There is some pre-processing that we've done to separate out subgroups of the data for which we have all of the relevant data for plotting, and subgroups for which we are missing some of that data. Taking a look at `missing_data_df` we see…

```
missing_data_df
```

```
## # A tibble: 84 x 12
##    sex   race_ethnicity  year percentage_esti~ percentage_stde~
##    <chr> <chr>          <int>            <dbl>            <dbl>
##  1 All ~ Pacific Islan~  2000             NA              NA
##  2 All ~ Pacific Islan~  2001             NA              NA
##  3 All ~ Pacific Islan~  2002             NA              NA
##  4 All ~ Pacific Islan~  2003             10.8             4.62
##  5 All ~ Pacific Islan~  2004             14.4             5.06
##  6 All ~ Pacific Islan~  2005             17.7             4.8
##  7 All ~ Pacific Islan~  2006             17.2             5.45
##  8 All ~ Pacific Islan~  2007             10.5             3.74
##  9 All ~ Pacific Islan~  2008             17.1             5.68
## 10 All ~ Pacific Islan~  2009             15.6             5.76
## # ... with 74 more rows, and 7 more variables: note_on_percentage <chr>,
## #   count_estimate <dbl>, count_stderr_estimate <dbl>,
```

```
## #   note_on_count <chr>, has_relevant_NAs <lgl>, sex_plot_label <chr>,
## #   race_ethnicity_plot_label <chr>
```

…that data specifically related to the `race_ethnicity` subgroups “Pacific Islander, non-Hispanic” and “Two or more races, non-Hispanic” have many missing values in the `percentage_estimate` column.

For a truly complete and representative analysis, we’d look to fill this missing data through supplemental data sources. We have not done that work here, but want to point to sources like the datasets found at the Asian American, Native Hawaiian, and Pacific Islander Dataset Repository. We recognize that this non-representativeness of the dataset we use is a limitation on the analysis we present here.

3. Visualizing the data

There are a couple main things we’ll aim to work through in this section:

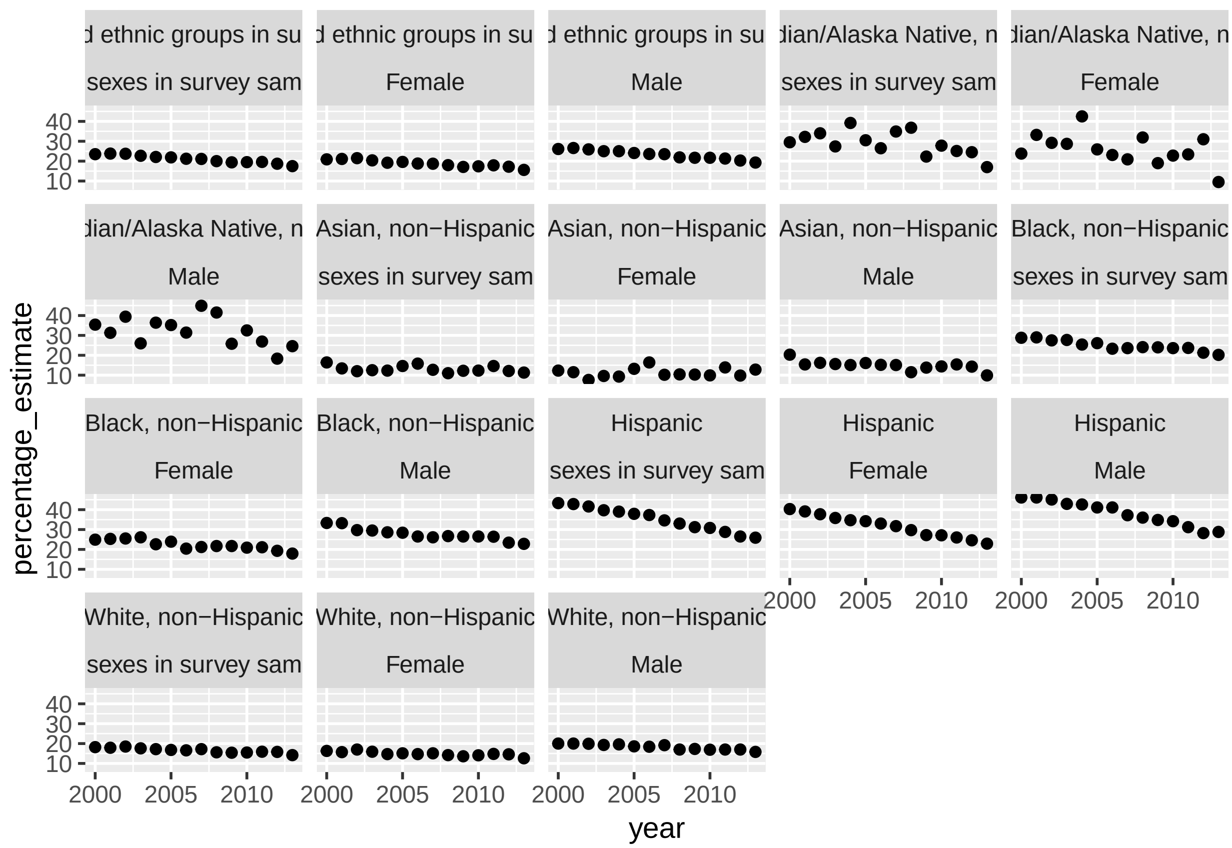
- A first pass at a `ggplot2` object
- Aesthetic mappings
- Faceting
- Geometric objects
- Visualizing uncertainty
- Statistical transformations
- Position adjustments
- Coordinate systems
- Titles and labels

As a note, before we begin: the presentation of this material is largely inspired by this section of the book *R for Data Science* by Garrett Grolemund and Hadley Wickham. We *highly* recommend working through that book if the tools you see here are ones you’d like to know more about; that book is a canonical resource for these tools.

3.1 A first pass at a `ggplot2` object

To begin with, let’s take a look at the following visualization:

```
ggplot2::ggplot(data = full_data_df) +
  ggplot2::geom_point(aes(x = year, y = percentage_estimate)) +
  ggplot2::facet_wrap(race_ethnicity ~ sex)
```



There are lots of things to note about this initial visualization, which we'll discuss more below. What are some that stand out to you?

3.2 Aesthetic mappings

Technically speaking, the only call we need to make to get a ggplot object is the `ggplot2::ggplot() function`. This function initializes a ggplot object. Let's see what we get by calling that function:

```
ggplot2::ggplot()
```

Without any further instruction, the `ggplot2::ggplot()` function gives us a skeleton of a ggplot object.

`ggplot2` works by building up plots in *layers*. To stack together these layers, we use the `+` operator. (This use of the `+` operator is specific to `ggplot2`.)

The first kind of layers we'll encounter are *aesthetic mappings*. An aesthetic mapping is happening whevever you see the `aes()` function. Theese mappings are how we tell `ggplot2` what information in our dataset to actually display, and where on the plot to display it.

In code, the basic setup (cited from here) of this aesthetic mapping is:

```
ggplot2::ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

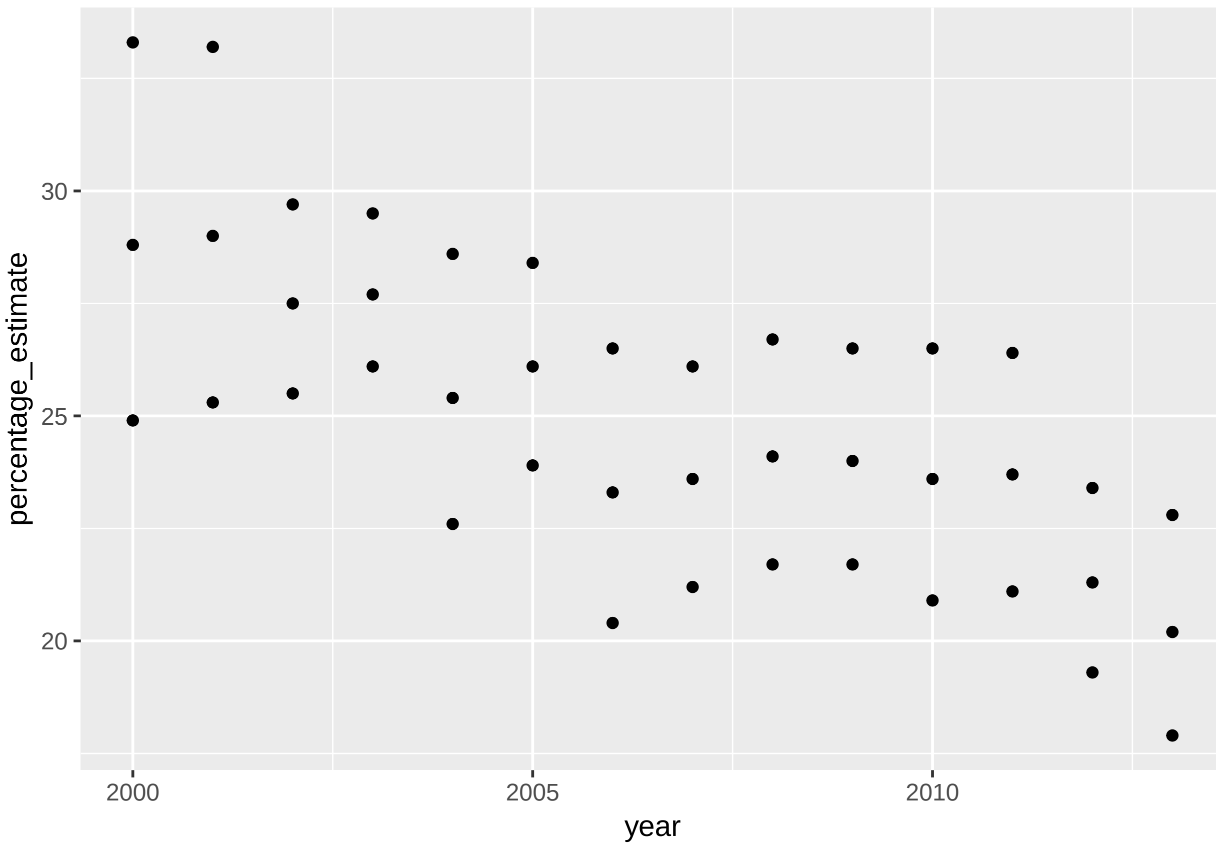
Breaking this down a little bit:

- The <DATA> value is the dataset that has the information we want to visulalize.
  - In the above example, this was `full_data_df`.
- The <GEOM_FUNCTION> value is a function that tells `ggplot2` exactly what kind of visualization we want (points/dots? bars? lines?).
  - In the above example we used `ggplot2::geom_point()`, which tells our plot to render the information as points.
- The <MAPPINGS> value tells `ggplot2` which columns of that dataset to use for visualization, and how to *map* those columns to the plot itself. + Above, we included the instruction `aes(x = year, y = Percentage)`, saying that the `year` column of `full_data_df` was to be mapped to the `x` axis of the plot, and that the `percentage` of `full_data_df` was to be mapped to the `y` axis of the plot.

(In the first plot, we also included an instruction to *facet* our plot into sub-plots, instead of showing everything on one big plot; more on that later.)

As an example, let's plot the examples in our dataset classified as having a `race_ethnicity` value of "Black, non-Hispanic" over time (e.g., `year`), ignoring for now the breakouts that exist for this group by `sex` in our dataset.

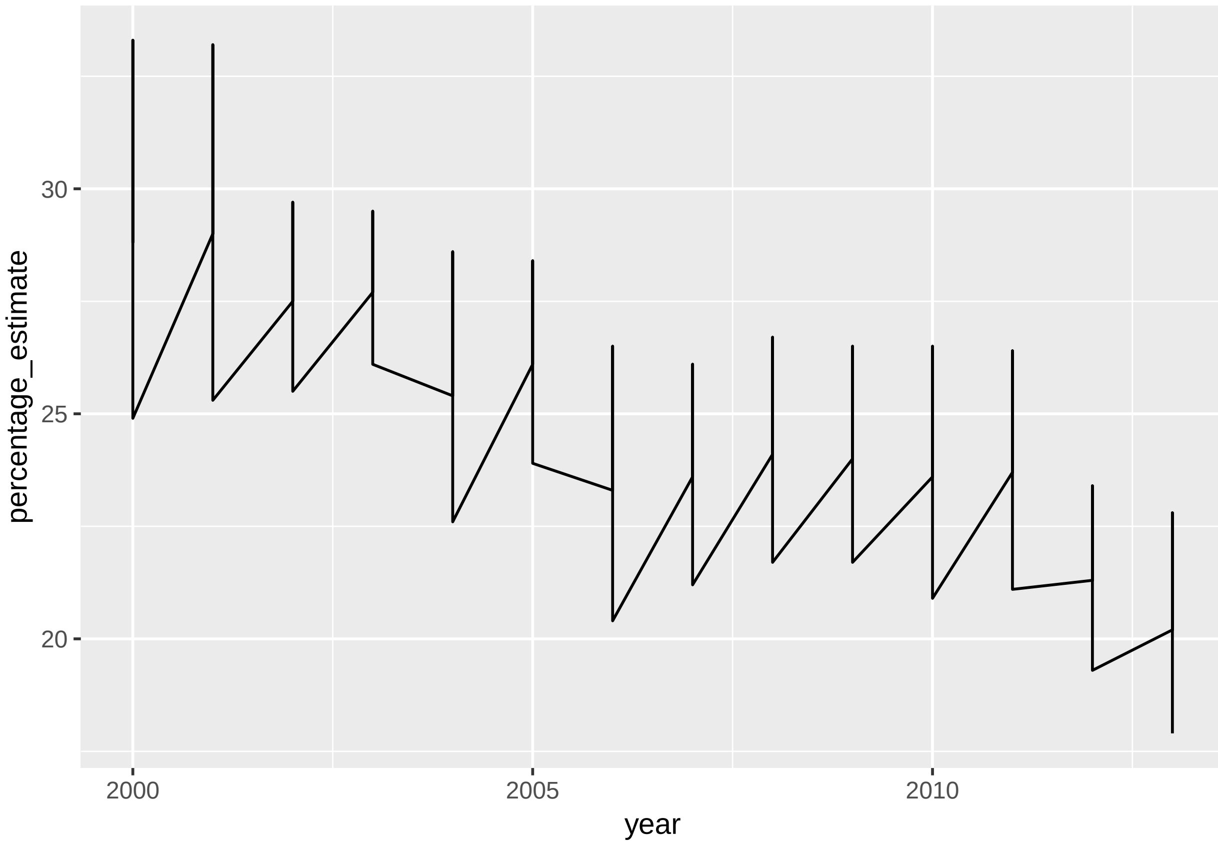
```
non_hisp_black_df <- full_data_df %>%
  filter(race_ethnicity == "Black, non-Hispanic")

ggplot2::ggplot(data = non_hisp_black_df) +
  ggplot2::geom_point(mapping = aes(x = year, y = percentage_estimate))
```



This plot is a little hard to interpret on its own; we can infer an overall downward trend over time in the estimated percentage of people classified as "Black, non-Hispanic" aged 18-24 in the US who do not graduate high school, but otherwise it's hard to discern much.

Visually, part of this confusion might be due to the fact that we're using points as opposed to another kind of `geom_` function… Let's try making a line plot using the `ggplot2::geom_line` function:

```
ggplot2::ggplot(data = non_hisp_black_df) +
  ggplot2::geom_line(mapping = aes(x = year, y = percentage_estimate))
```

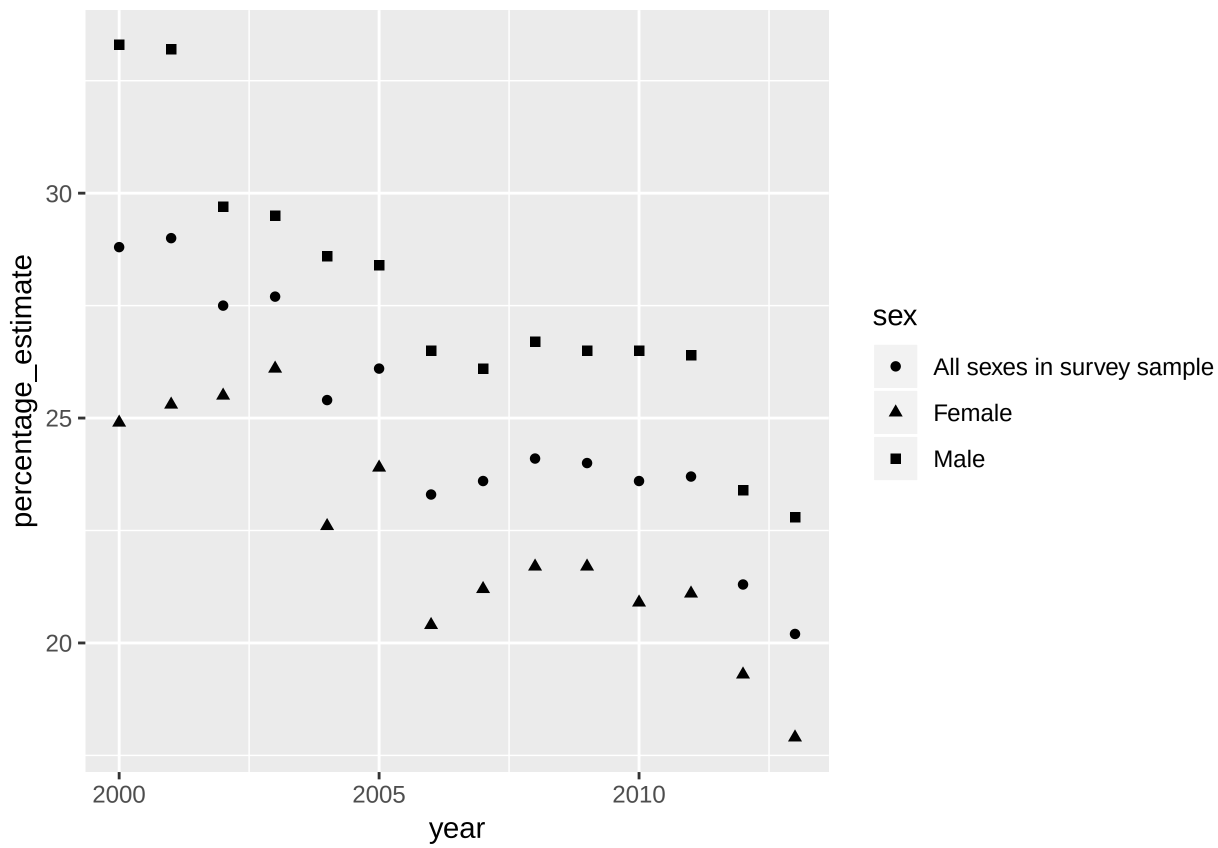


This arguably looks even more strange! What could be going on…

…as we know, our dataset has a subgrouping of its records beyond the `race_ethnicity` column: there is also the `sex` column. With this in mind, we realize that what's being plotted above is data from all three groups of the `sex` column at once, but that `ggplot2` can't detect that there's a further relevant subgrouping on its own.

There are a couple ways to proceed here. One way is to think to try to add some distiguishing features for points associated with each of the three unique values ("Female", "Male", "All sexes in survey sample") of our `sex` column, specific to each of those values. We might think to try the *shape* of the points:

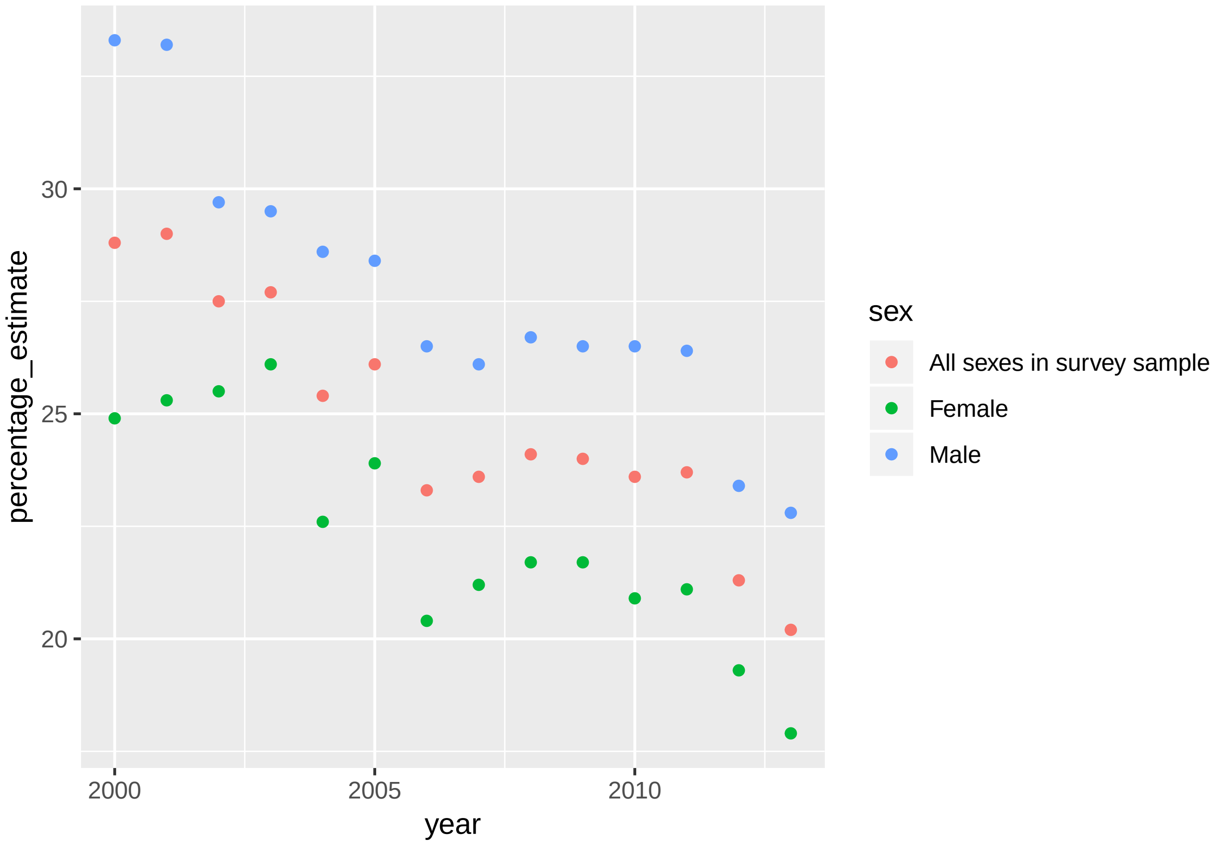
```
ggplot2::ggplot(data = non_hisp_black_df) +
  ggplot2::geom_point(mapping = aes(x = year, y = percentage_estimate, shape = sex))
```



By now passing the `shape = sex` instruction to our `aes()` call, we get a different shaped point for each level of the `Sex` variable, and also a *legend* on the right side of the plot.

Maybe these points are still a little hard to distinguish, though. We can also group the points by *color*, by passing `shape = sex` instead of `color = sex`:

```
ggplot2::ggplot(data = non_hisp_black_df) +
  ggplot2::geom_point(mapping = aes(x = year, y = percentage_estimate, color = sex))
```

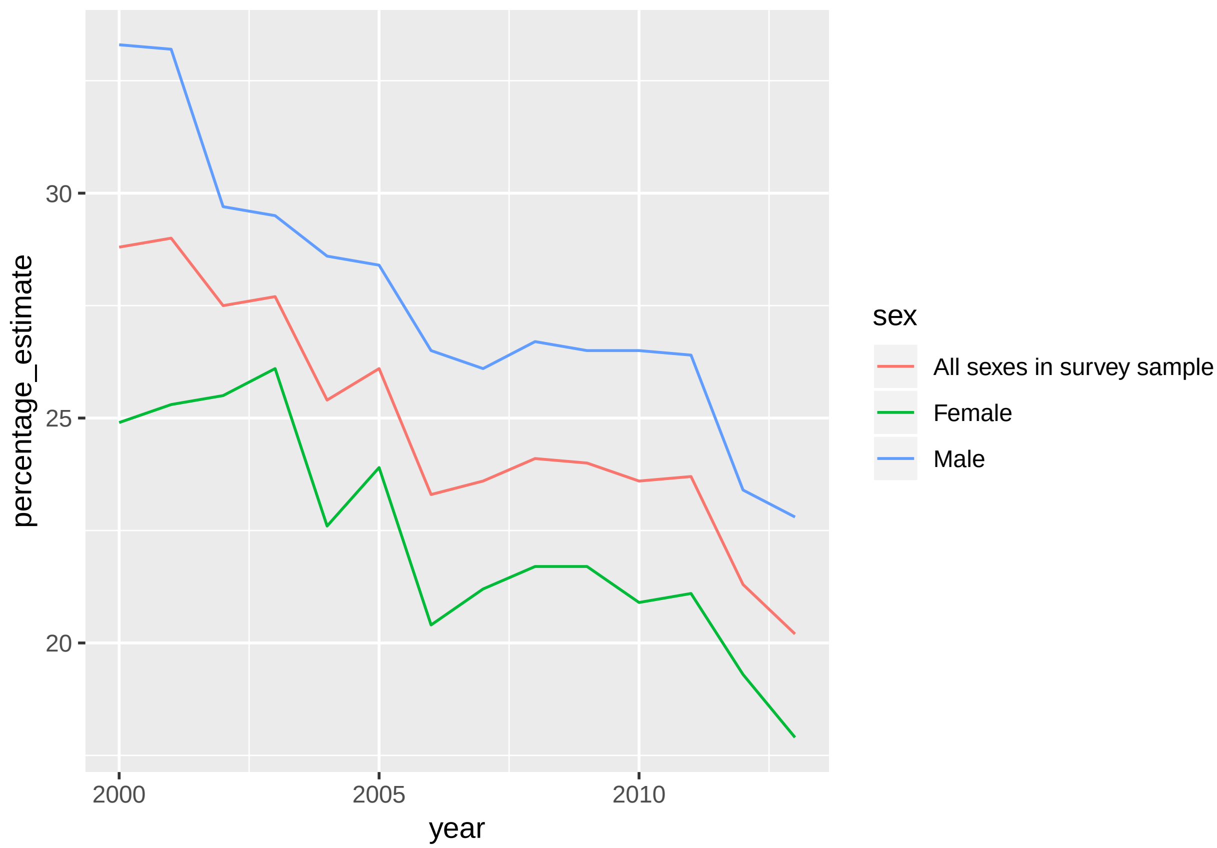


This might be a little clearer. We might think things would be even clearer, though, if we could use lines instead of points.

Luckily, passing in `color = sex` not only explicitly colors by level of the `sex` column, but also *groups* that data under the hood so that rows in our data that share a value of `sex` are grouped together.

To see what we mean, let's go back to using `ggplot::geom_line()`:

```
ggplot2::ggplot(data = non_hisp_black_df) +
  ggplot2::geom_line(mapping = aes(x = year, y = percentage_estimate, color = sex))
```



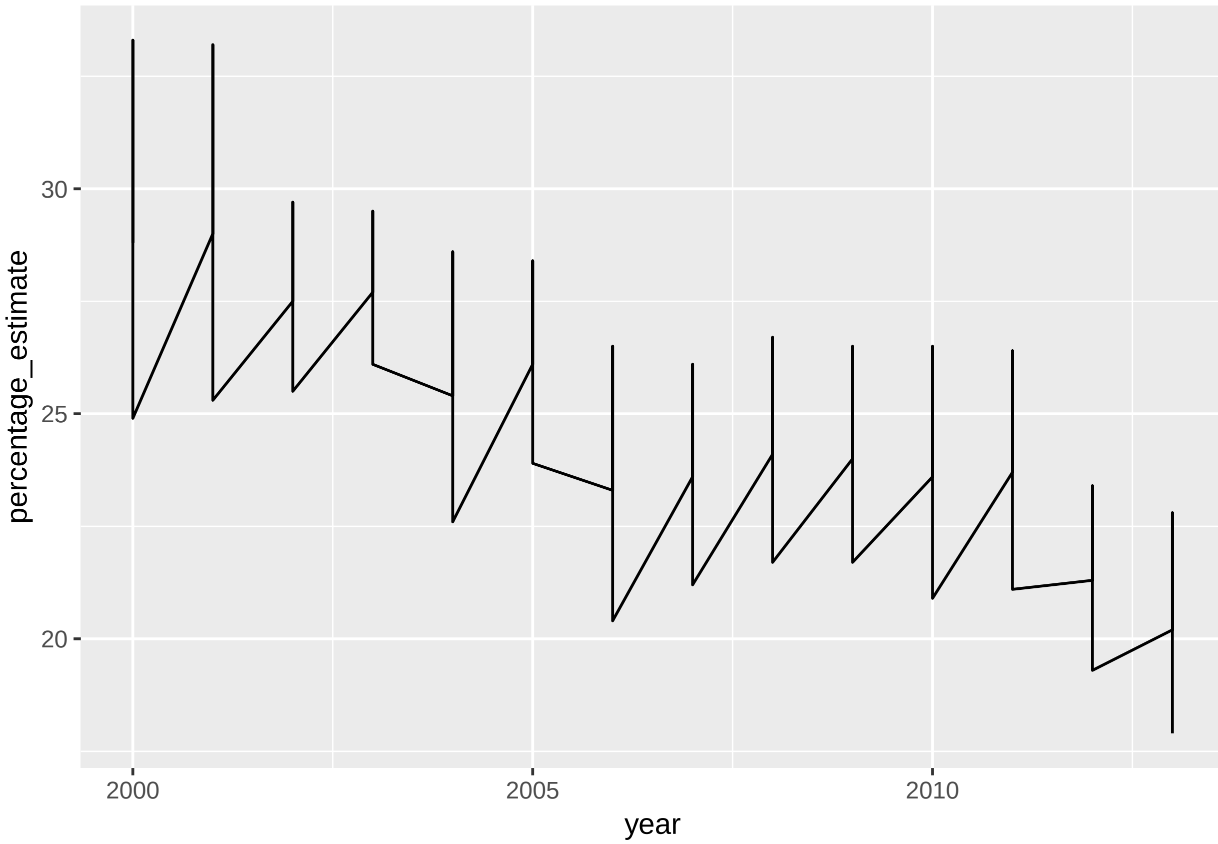
This seems like a more suitable way to show our data in a more disaggregated way. Let's consider another way.

3.3 Faceting

Faceting will allow us to show the same information as above, except will give an automatic way to show things on *separate* plots as opposed to the same plot.

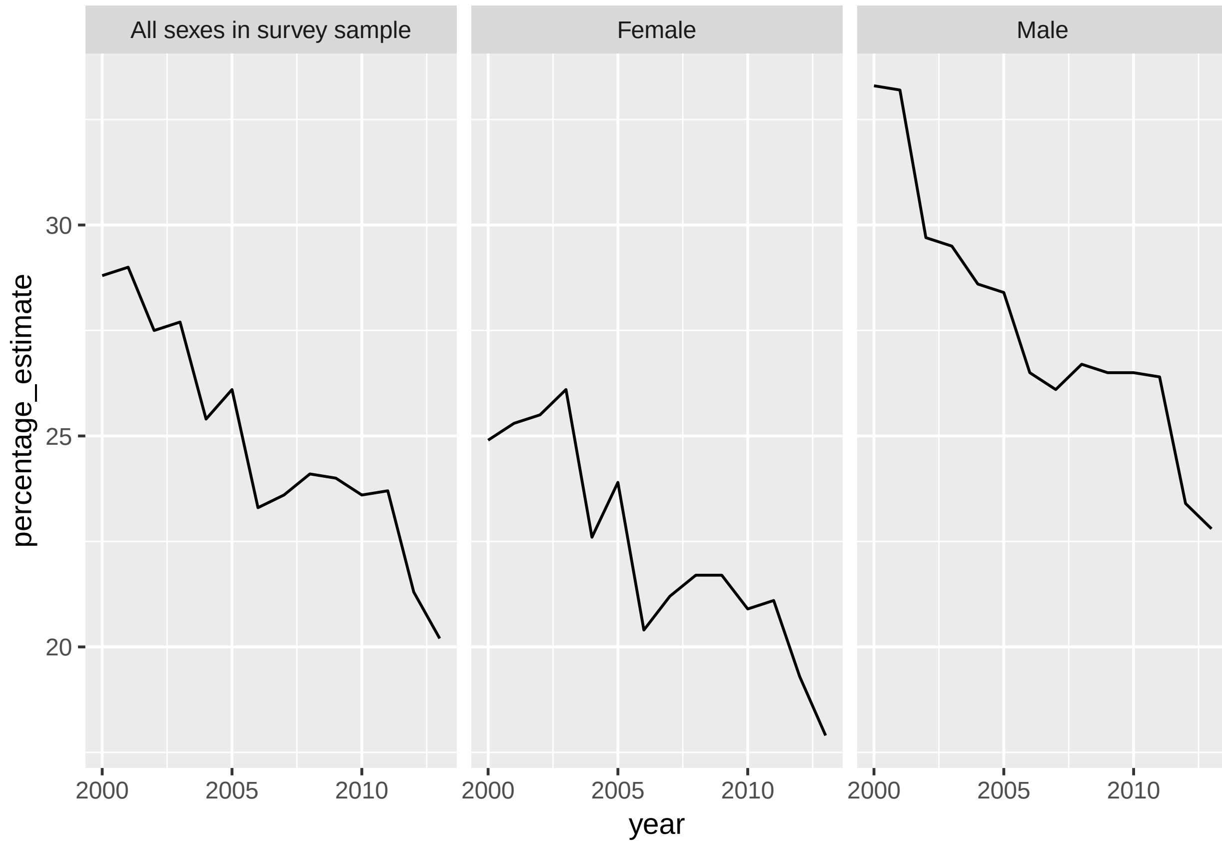
Above, we started with:

```
ggplot2::ggplot(data = non_hisp_black_df) +
  ggplot2::geom_line(mapping = aes(x = year, y = percentage_estimate))
```



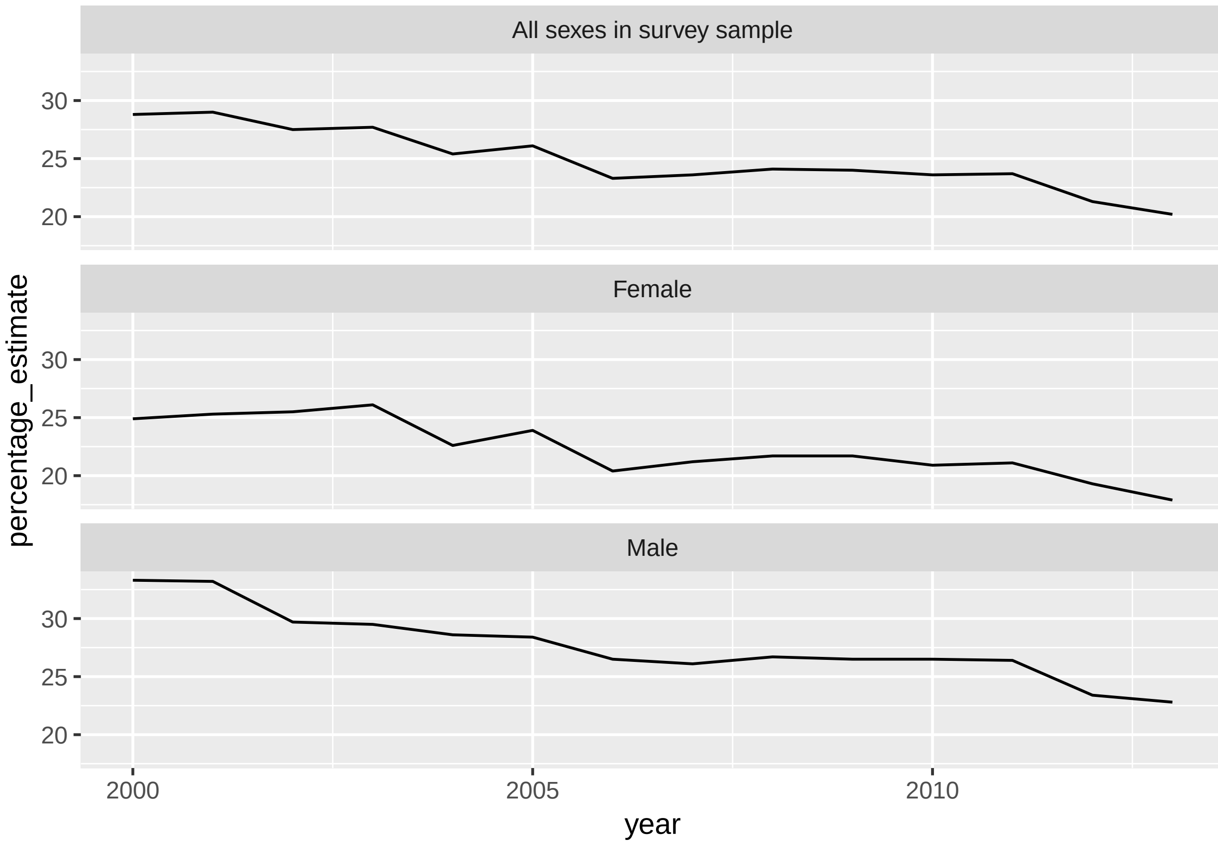
Let's see what happens if we add a call to the `ggplot2::facet_wrap()` function:

```
ggplot2::ggplot(data = non_hisp_black_df) +
  ggplot2::geom_line(mapping = aes(x = year, y = percentage_estimate)) +
  ggplot2::facet_wrap(~ sex)
```



This call to `facet_wrap()` took as its argument a *formula* data structure, which is created with the `~` character. By default, the plots are listed as separate *columns* in the plot. Instead of this, we could specify that we want there to be only *one* column:

```
ggplot2::ggplot(data = non_hisp_black_df) +
  ggplot2::geom_line(mapping = aes(x = year, y = percentage_estimate)) +
  ggplot2::facet_wrap(~ sex, ncol = 1)
```

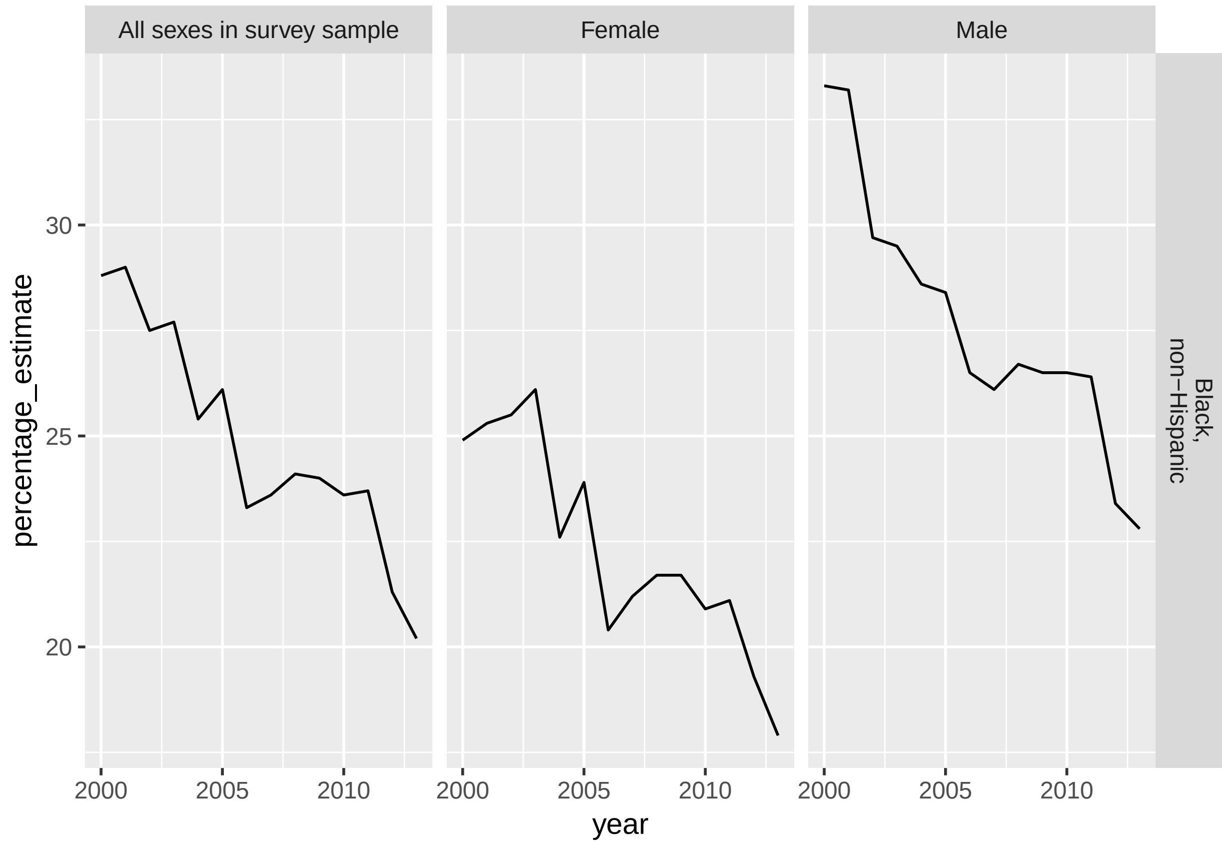


Depending on the task, this might be a more appropriate way to visualize the data.

We might also want to be more explicit about which race/ethnicity group is being plotted on the body of the plot itself. A closely related function to `ggplot2::facet_wrap()` that does this, but with slightly nicer default display options, is `ggplot2::facet_grid()`. (`facet_wrap()` and `facet_grid()` are similar, but they do differ in how they handle subgroups/intersections with *no* data in them. For more on this, see a helpful Stack Overflow post.

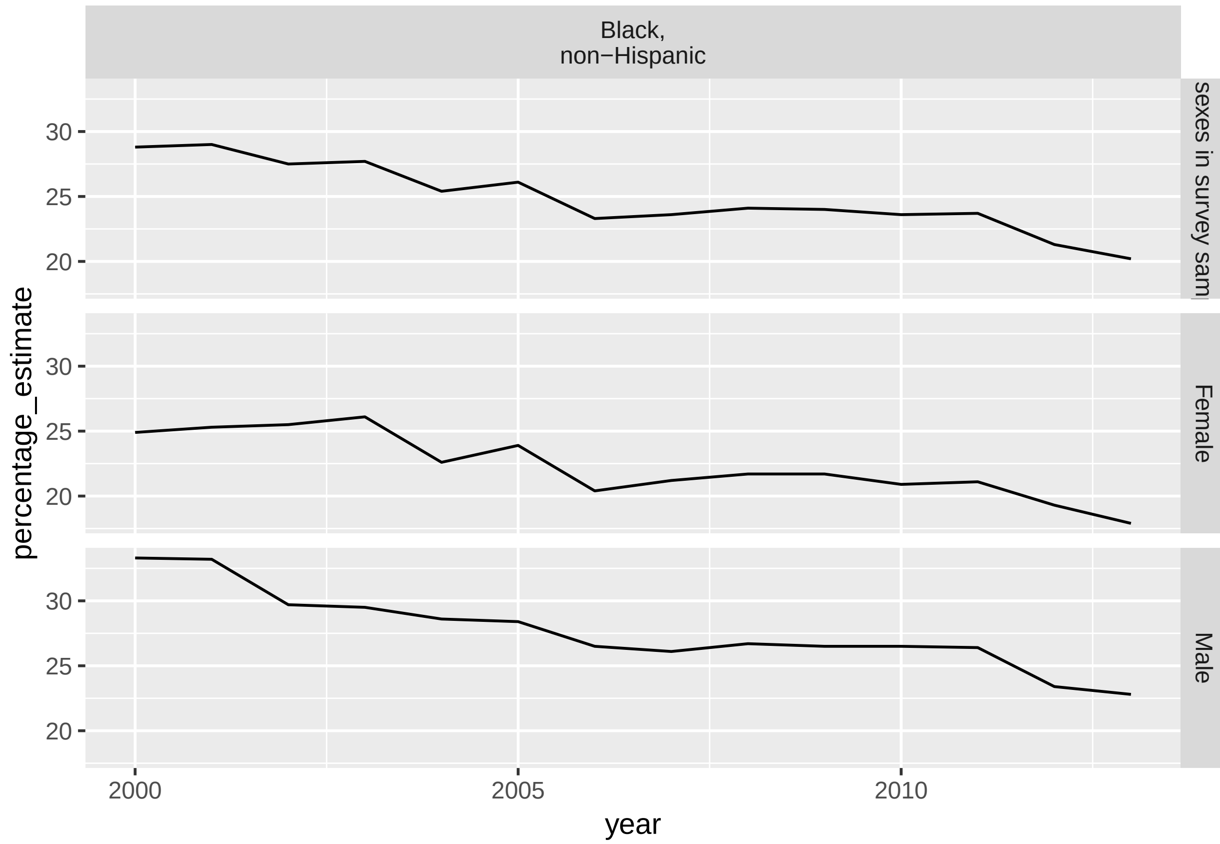
We use a *two-sided formula* as the argument of `ggplot2::facet_grid()`:

```
ggplot2::ggplot(data = non_hisp_black_df) +
  ggplot2::geom_line(mapping = aes(x = year, y = percentage_estimate)) +
  ggplot2::facet_grid(race_ethnicity_plot_label ~ sex)
```



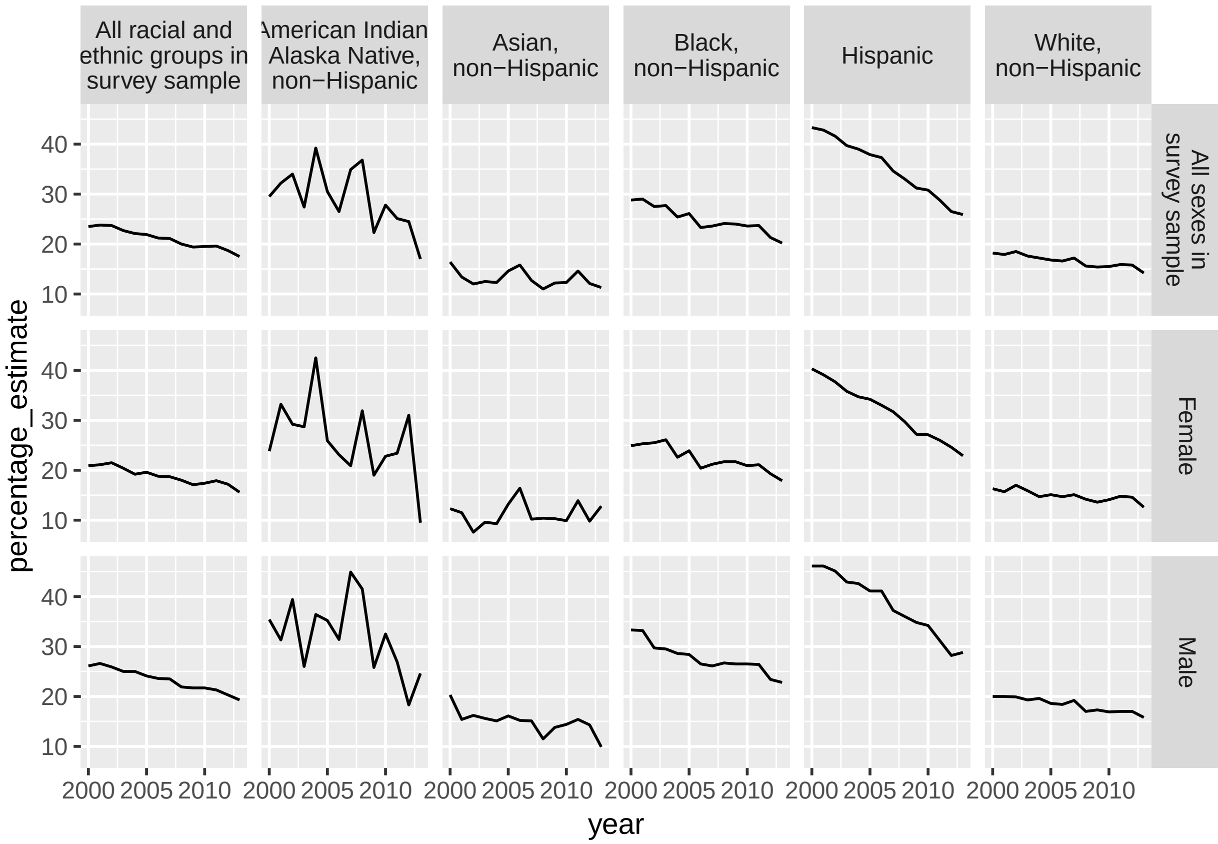
What this call to `ggplot2::facet_grid()` with a two-sided formula does in context is to make it more explicit that we are breaking down our data into `sex`/`race_ethnicity` subgroups. We can also flip the direction, putting the `race_ethnicity` category on top and the `sex` categories on the side:

```
ggplot2::ggplot(data = non_hisp_black_df) +
  ggplot2::geom_line(mapping = aes(x = year, y = percentage_estimate)) +
  ggplot2::facet_grid(sex ~ race_ethnicity_plot_label)
```



We can even do this on the larger `full_data_df` dataset:

```
ggplot2::ggplot(data = full_data_df) +
  ggplot2::geom_line(mapping = aes(x = year, y = percentage_estimate)) +
  ggplot2::facet_grid(sex_plot_label ~ race_ethnicity_plot_label)
```

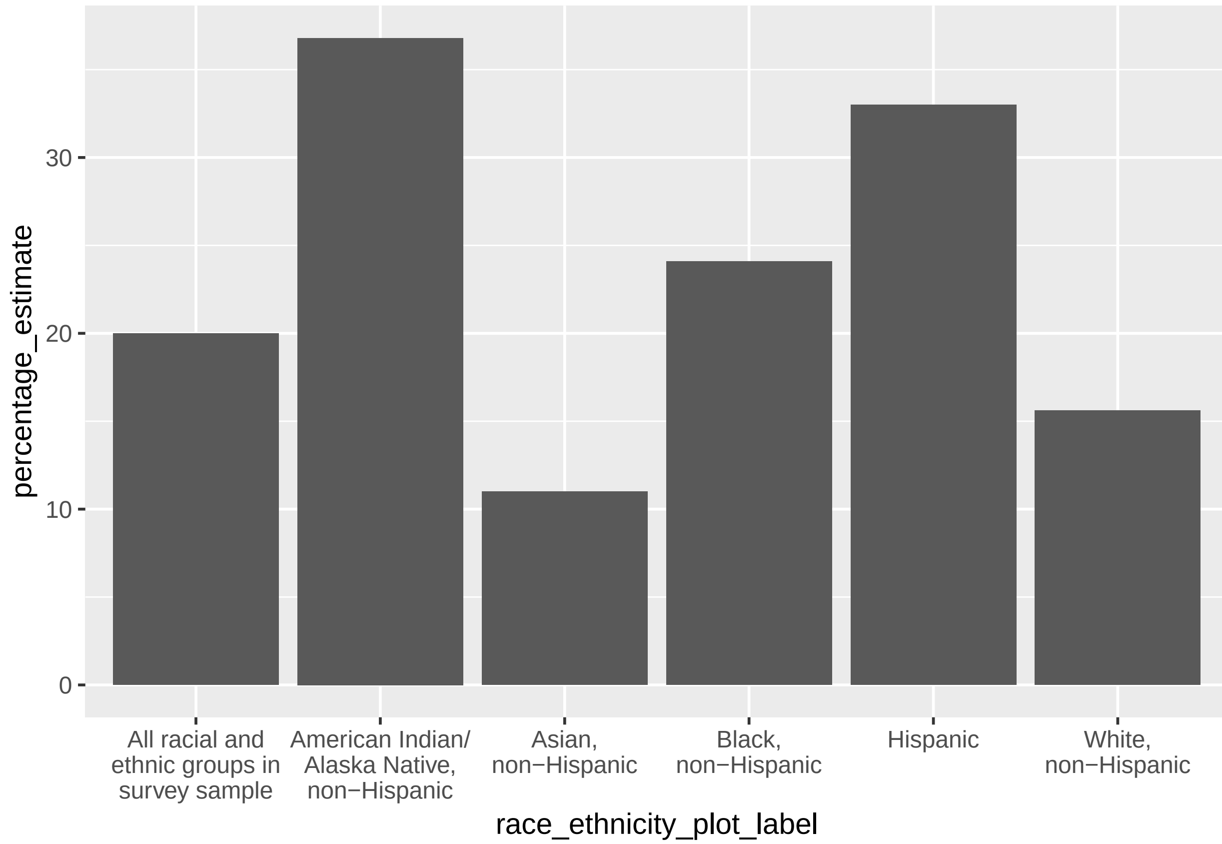


Note that we'll still want to play around with how the labels end up being rendered on our plot; this is something we'll get to later.

3.4 Geometric objects

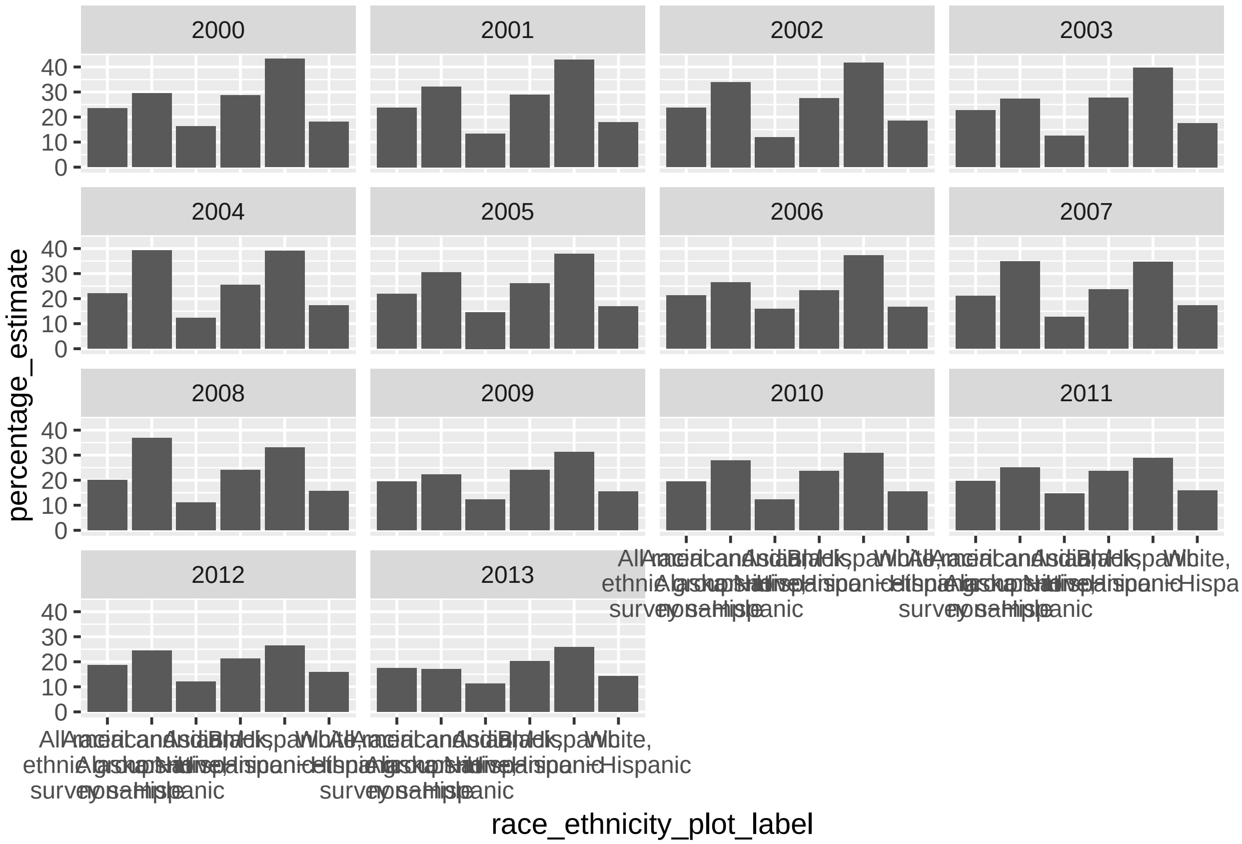
We saw two kinds of *geometric objects* above: `ggplot2::geom_line()` and `ggplot2::geom_point()`. Another kind of geometric object we might want to use is `ggplot2::geom_bar()`, to (for example) make a comparison of a given `sex` group's trends in high school drop out rate by `race_ethnicity`. Let's look at this for the year 2008:

```
df_for_bars <- full_data_df %>%
  dplyr::filter(year == 2008, sex == 'All sexes in survey sample')

ggplot2::ggplot(data = df_for_bars) +
  ggplot2::geom_bar(mapping = aes(x = race_ethnicity_plot_label, y = percentage_estimate), stat = 'iden
```



We can even combine this with a faceting instruction:

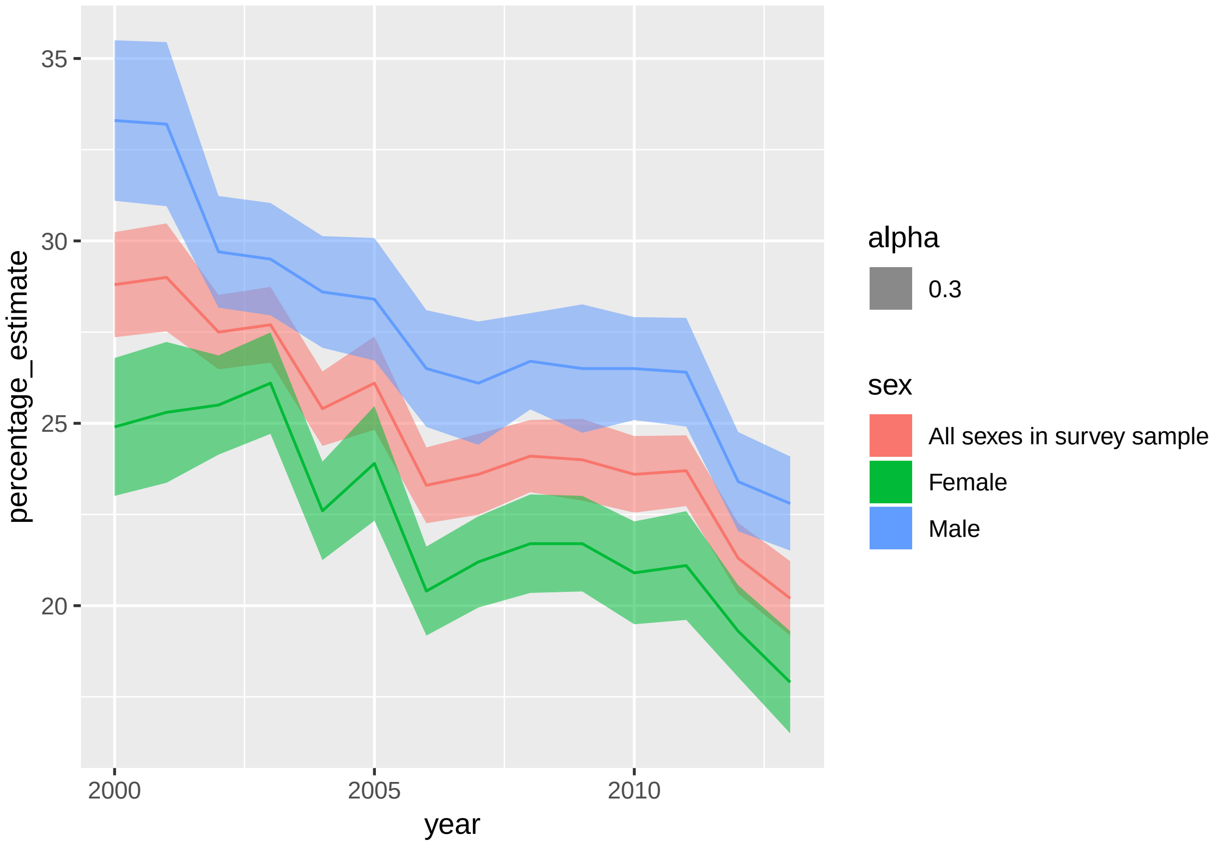
```
all_sexes_df <- full_data_df %>%
  dplyr::filter(sex == 'All sexes in survey sample')

ggplot2::ggplot(data = all_sexes_df) +
  ggplot2::geom_bar(mapping = aes(x = race_ethnicity_plot_label, y = percentage_estimate), stat = 'iden
  ggplot2::facet_wrap(~ year)
```



3.4.1 Visualizing uncertainty

Another geometric mapping that's especially helpful for plotting *uncertainty* is `ggplot2::geom_ribbon()`, which fills in an interval (a "ribbon") within some given bounds. Again using `non_hisp_black_df`, for example, we might see:

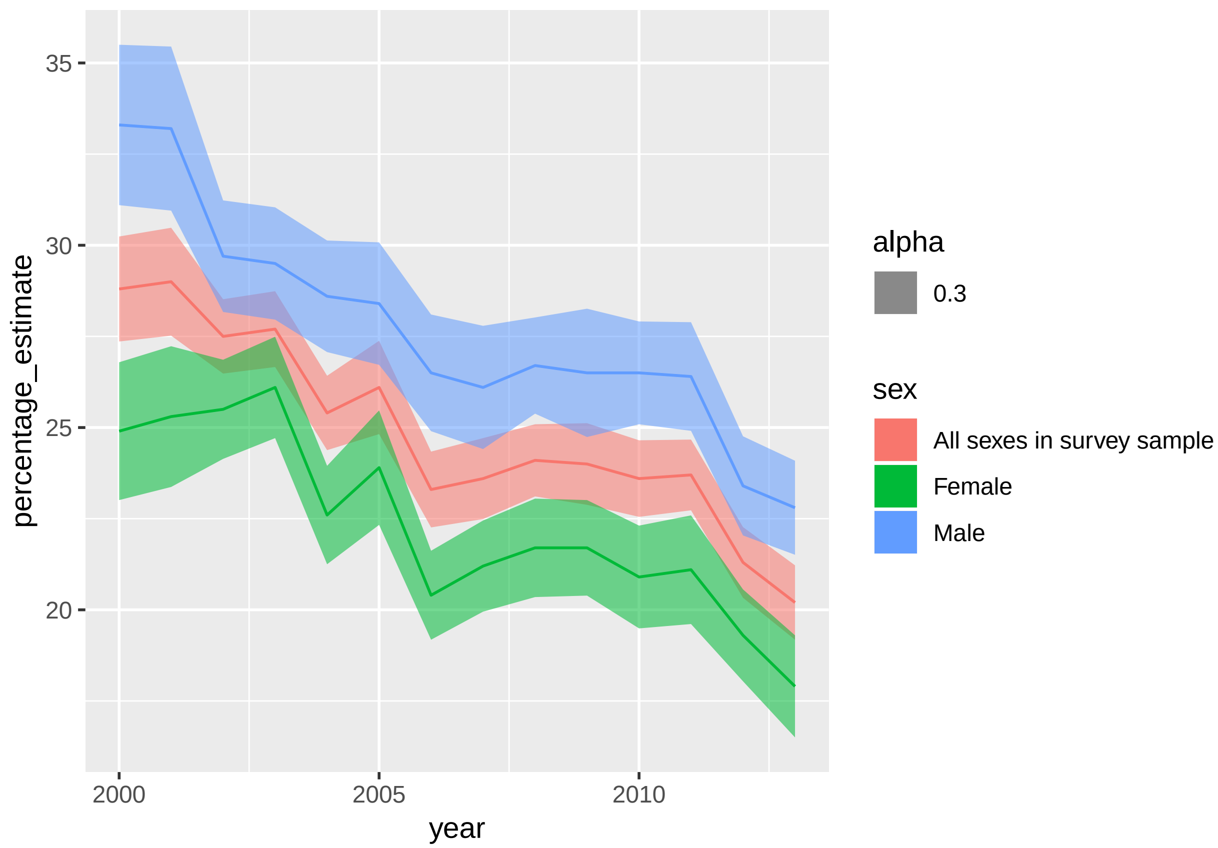
```
ggplot2::ggplot(data = non_hisp_black_df) +
  ggplot2::geom_line(mapping = aes(x = year, y = percentage_estimate, color = sex)) +
  ggplot2::geom_ribbon(mapping = aes(x = year,
                                     ymin = percentage_estimate - percentage_stderr_estimate,
                                     ymax = percentage_estimate + percentage_stderr_estimate,
                                     fill = sex,
                                     alpha = 0.3))
```



Here, we encounter two new possible arguments for the `aes()` function: `fill` (which controls what color `geom_ribbon()` uses to "fill" the area it defines) and `alpha` (which defines the transparency of the fill color).

Since we're now in the domain of using *more than one geometric object on the same plot*, we might think that we want these geometric objects to share (some of) the same aesthetic mappings. To do this, we can move the shared aesthetic mappings up to the original call of the `ggplot2::ggplot()` function as follows:

```
ggplot2::ggplot(
  data = non_hisp_black_df,
  mapping = aes(x = year,
                y = percentage_estimate,
                ymin = percentage_estimate - percentage_stderr_estimate,
                ymax = percentage_estimate + percentage_stderr_estimate)
  ) +
  ggplot2::geom_line(aes(color = sex)) +
  ggplot2::geom_ribbon(aes(fill = sex, alpha = 0.3))
```

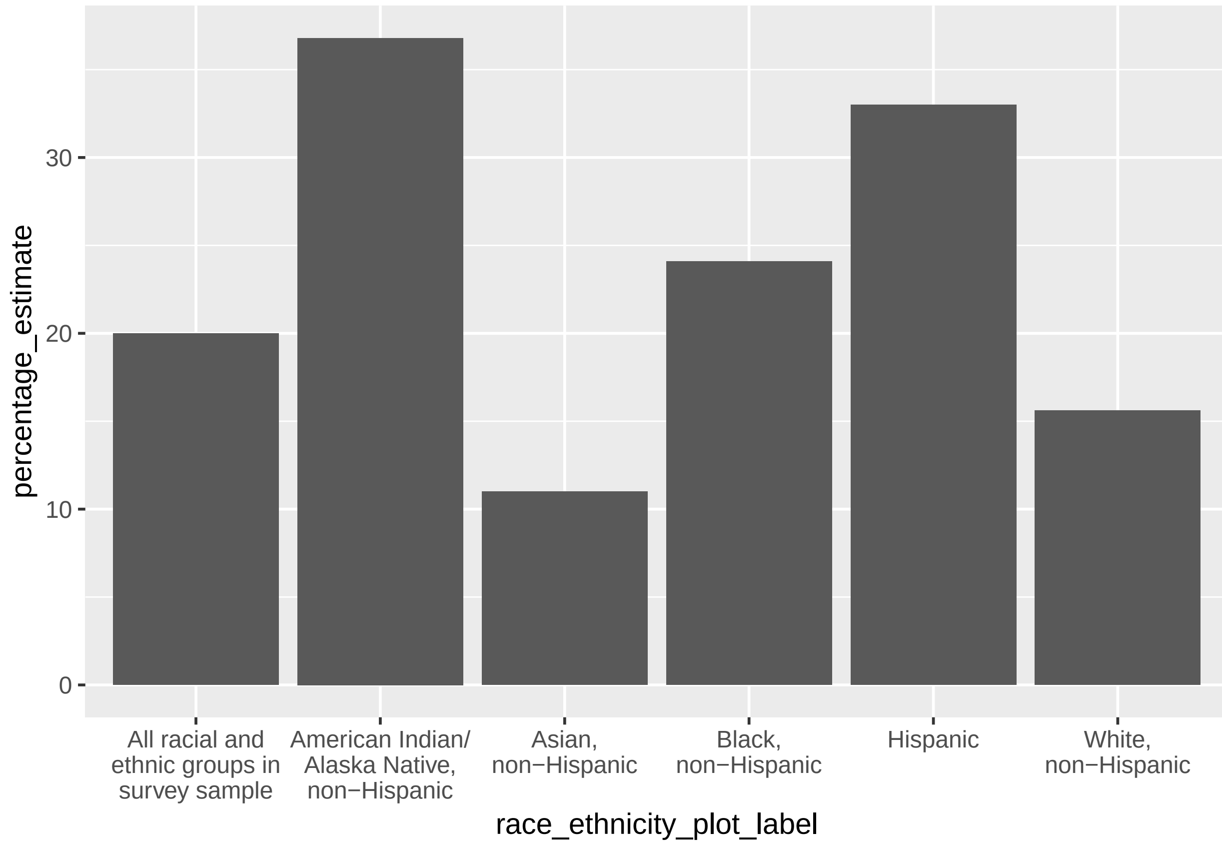


Below we'll briefly cover a few more slightly advanced topics:

3.5 Statistical transformations

Notice that in our original call to make bar plots…

```
ggplot2::ggplot(data = df_for_bars) +
  ggplot2::geom_bar(mapping = aes(x = race_ethnicity_plot_label, y = percentage_estimate), stat = 'iden
```



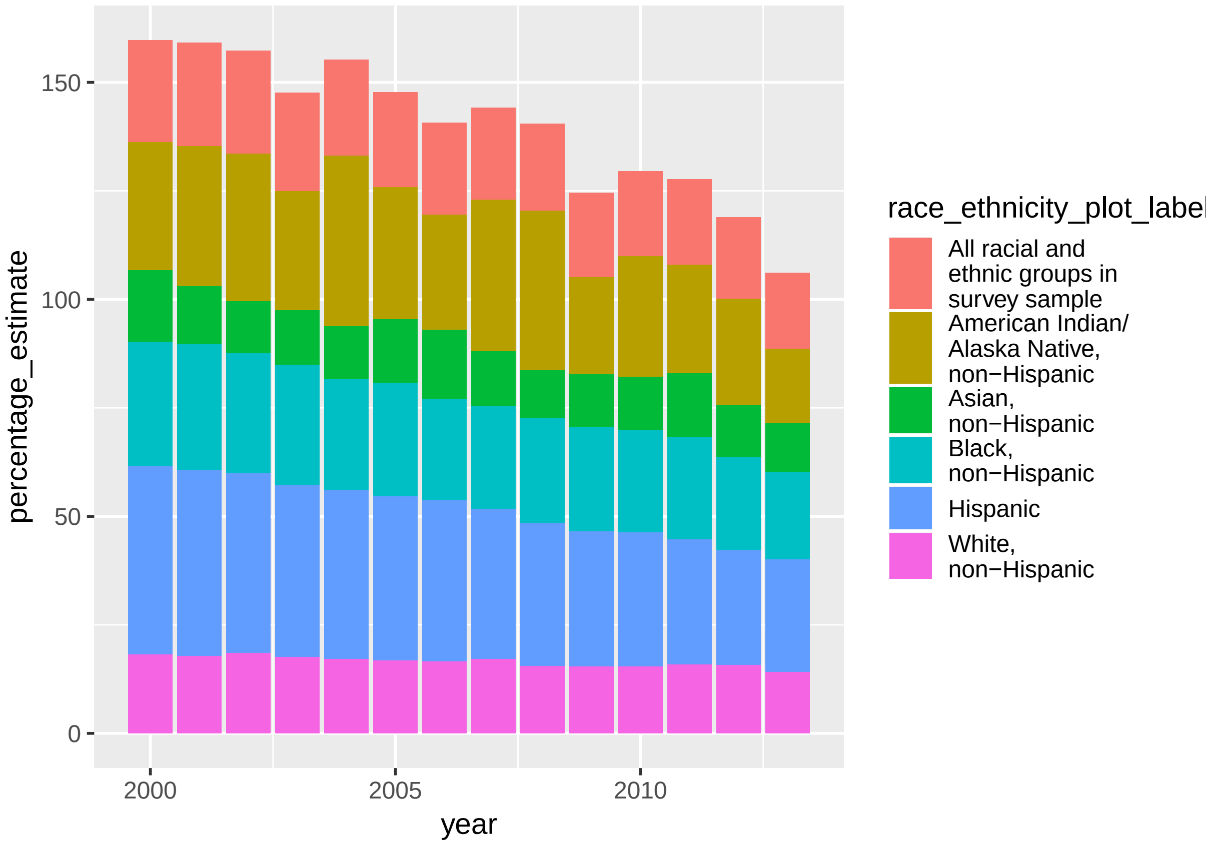
…we included the call `stat = 'identity'`. This `'identity'` value is an example of a *statistical transformation*.

Depending on the form your data is in, you may have to use different statistical transformations on your data in order to get the desired metrics you want. For example, if you had *individual-level* data, where each row represented for example one individual person, but you wanted to get metrics about *groups of people*, you would most likely need to use a different statistical transformation. See the hyperlink above for more information.

3.6 Position adjustments

Imagine that instead of faceting to get bar plots by year, you wanted all of the plots on the same axis. We might think to try:

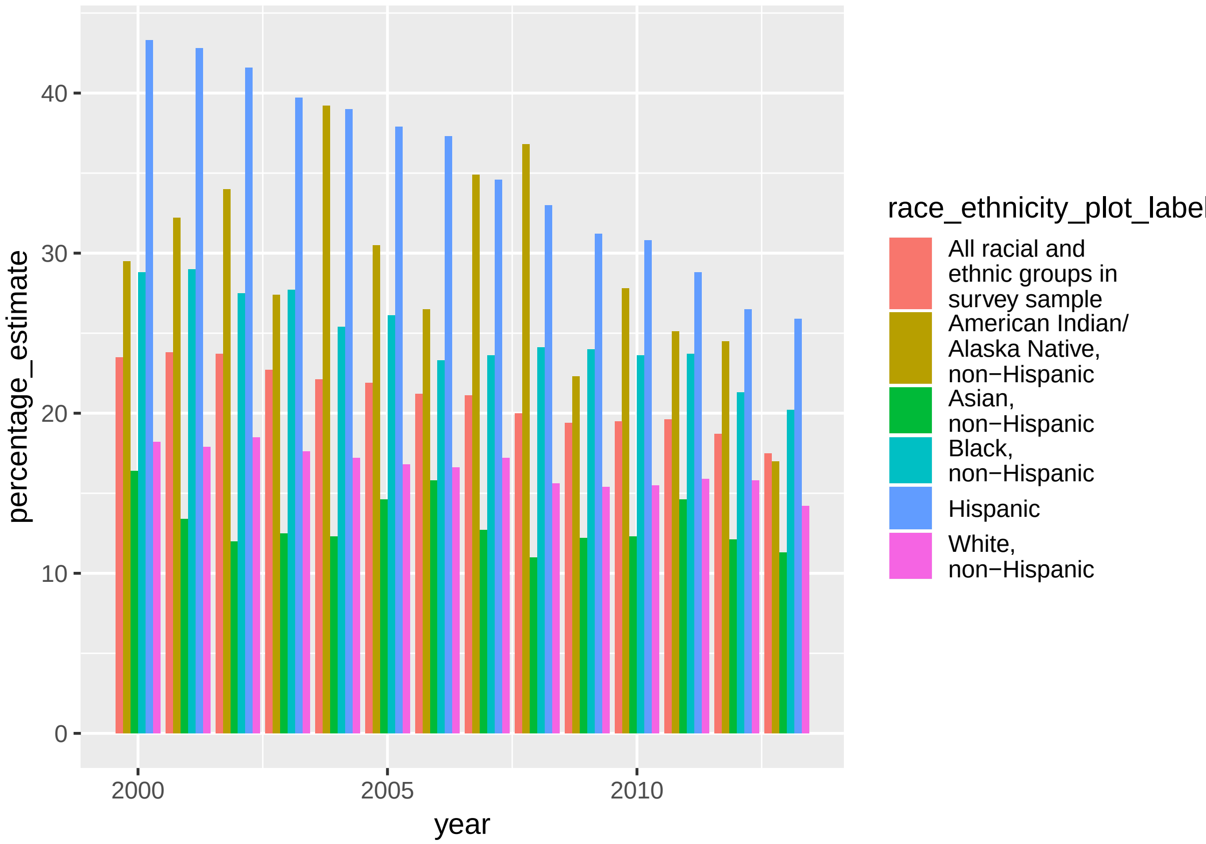
```
ggplot2::ggplot(data = all_sexes_df) +
  ggplot2::geom_bar(mapping = aes(x = year, y = percentage_estimate, fill = race_ethnicity_plot_label),
```



But that doesn't look right: the value of `Percentage` goes past 100! (Although this kind of "stacked bar" plot might be appropriate for other applications.)

Let's make use of the `position` paramter of `ggplot2::geom_bar()`:

```
ggplot2::ggplot(data = all_sexes_df) +
  ggplot2::geom_bar(mapping = aes(x = year, y = percentage_estimate, fill = race_ethnicity_plot_label),
                    stat = 'identity', position = "dodge")
```

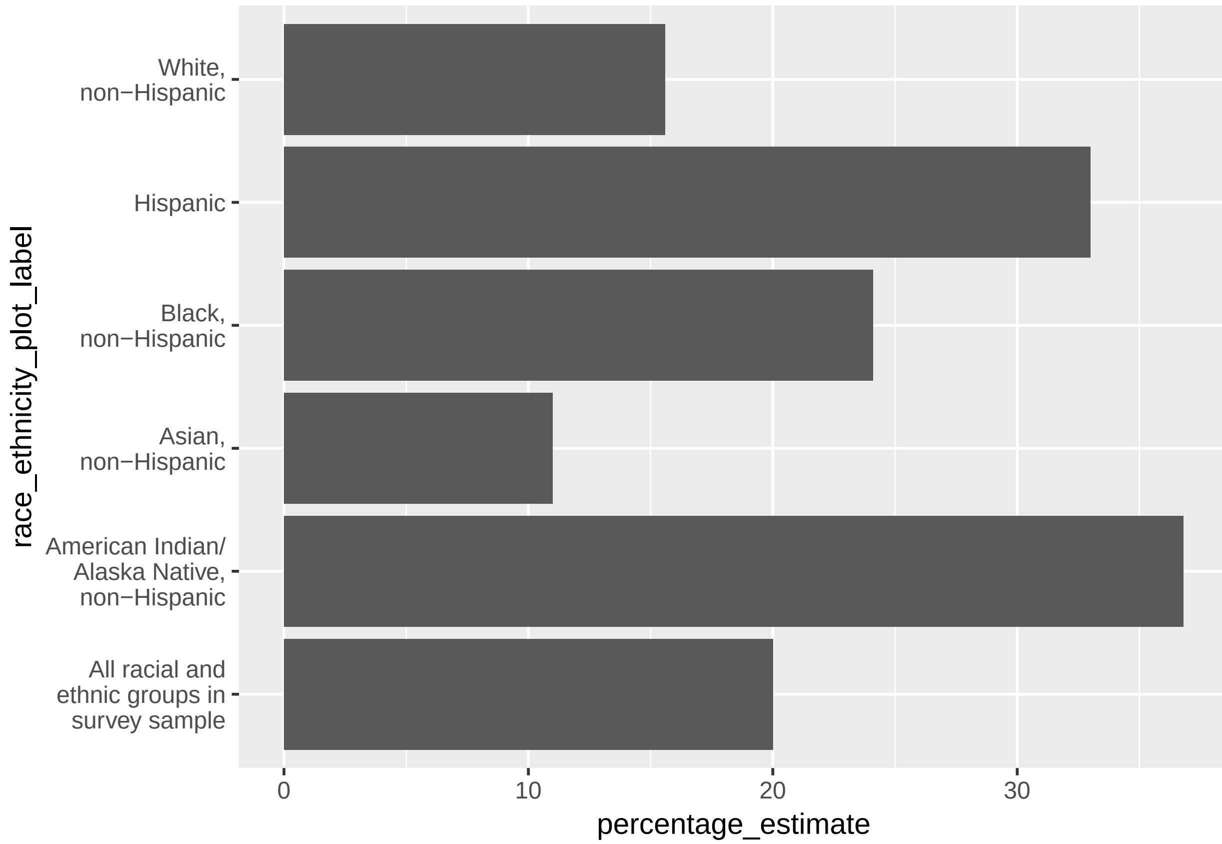


See more about position adjustment for plot layers here.

3.7 Coordinate systems

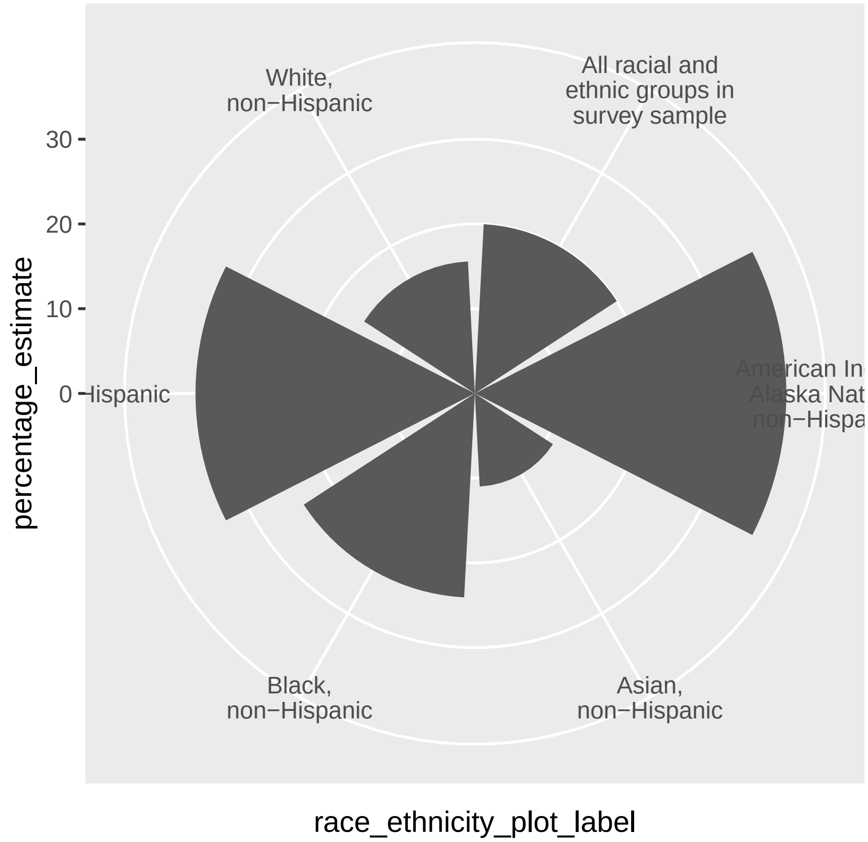
Instead of wanting vertical bars, we might want horizontal bars. We can do this with the `ggplot2::coord_flip()` function:

```
ggplot2::ggplot(data = df_for_bars) +
  ggplot2::geom_bar(mapping = aes(x = race_ethnicity_plot_label, y = percentage_estimate), stat = 'iden
  ggplot2::coord_flip()
```



We can also tell to make a polar area diagram using the `ggplot2::coord_polar()` function:

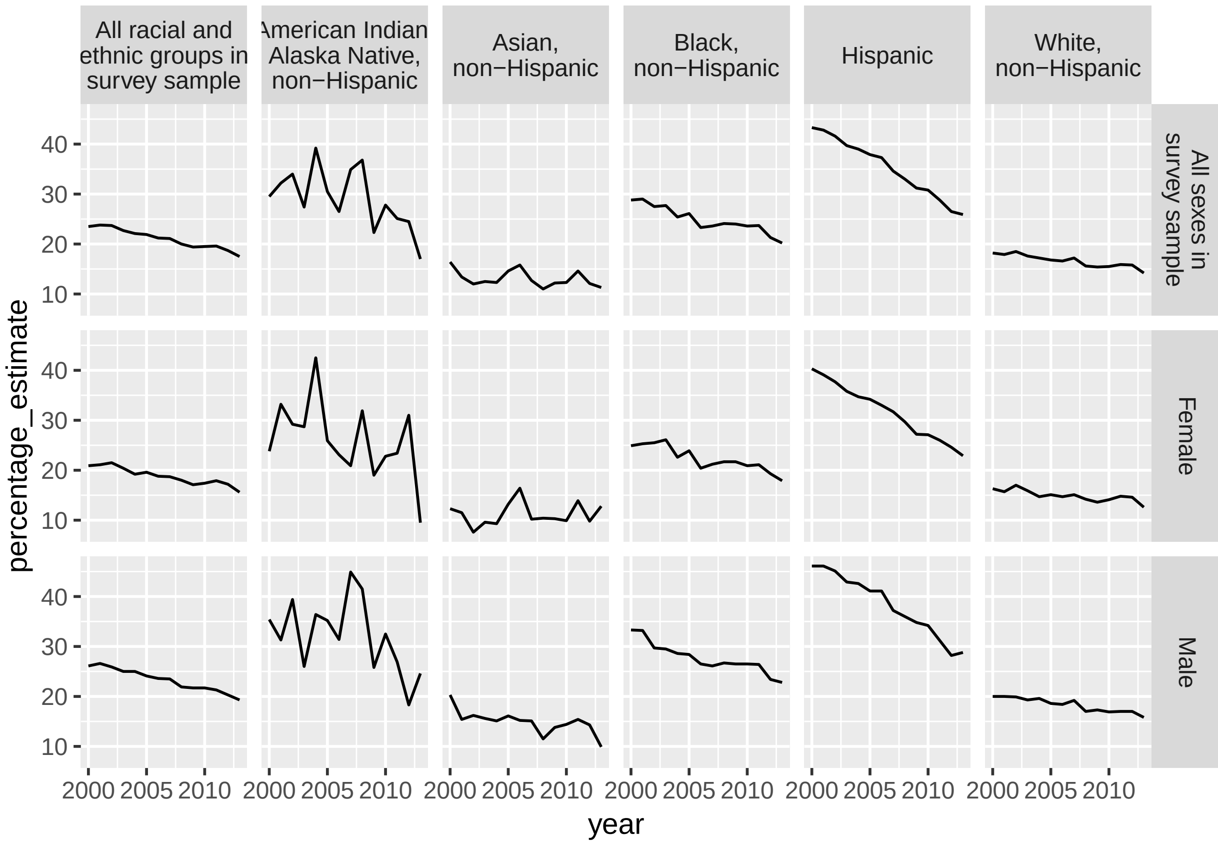
```
ggplot2::ggplot(data = df_for_bars) +
  ggplot2::geom_bar(mapping = aes(x = race_ethnicity_plot_label, y = percentage_estimate), stat = 'iden
  ggplot2::coord_polar()
```



3.8 Making a finalized version of a plot: titles and labels

Often when we're making plots, we'll want to take care to clean up the labels and add descriptive titles. Let's turn back to the example of one of our earlier plots:

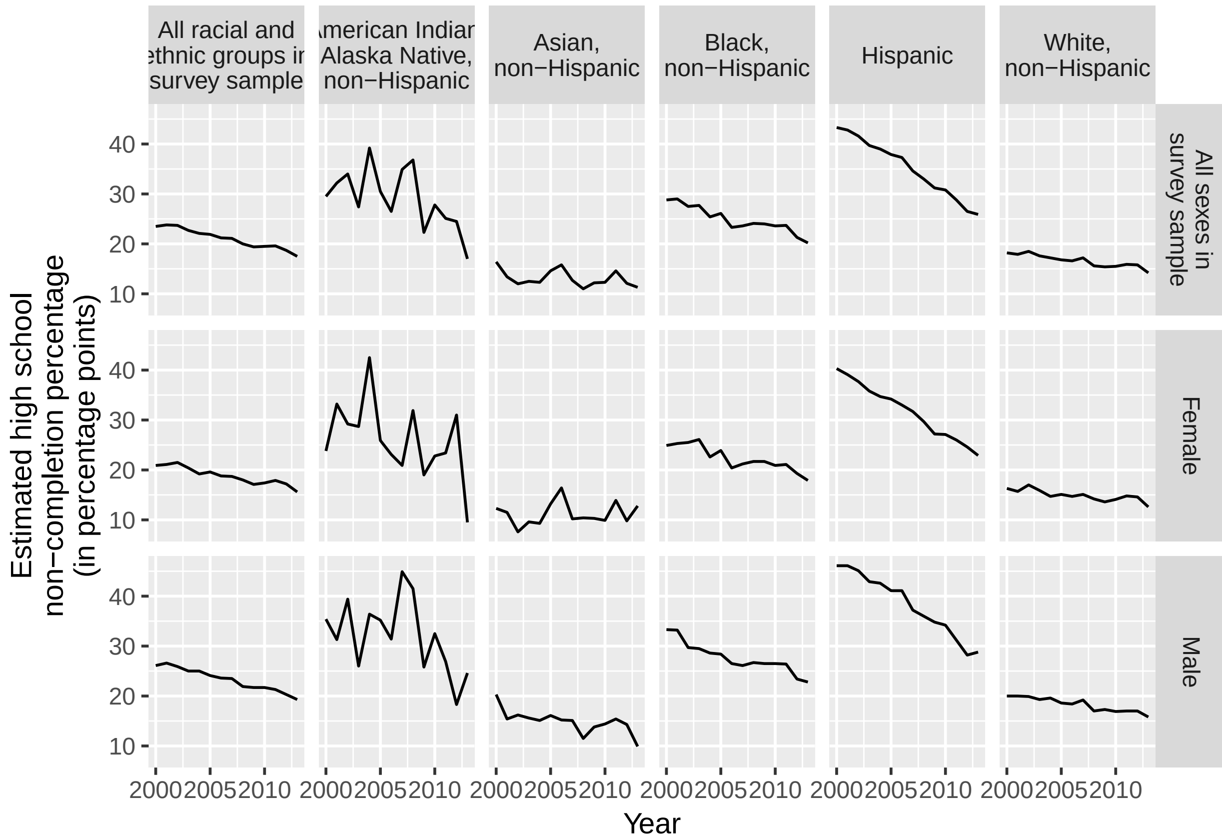
```
ggplot2::ggplot(data = full_data_df) +
  ggplot2::geom_line(mapping = aes(x = year, y = percentage_estimate)) +
  ggplot2::facet_grid(sex_plot_label ~ race_ethnicity_plot_label)
```



This plot looks fairly clean by default: for example, we can see all of the subgroup labels in the facet headings, and the axis scales seem to be legible.

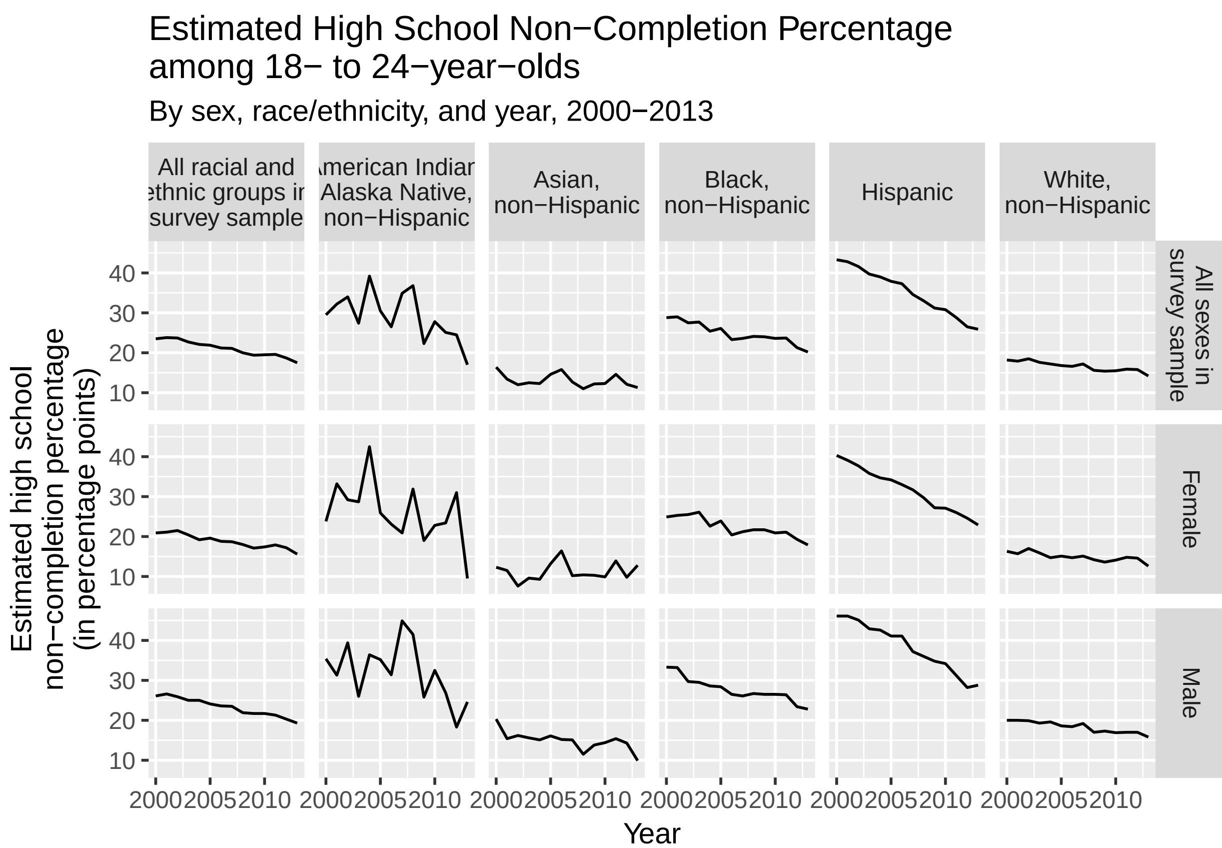
What if we wanted to change our axis labels to something more descriptive, or at least make them uppercase? For the given plot, since we have both an x axis and a y axis, we can use the `ggplot2::xlab()` and `ggplot2::ylab()` functions:

```
ggplot2::ggplot(data = full_data_df) +
  ggplot2::geom_line(mapping = aes(x = year, y = percentage_estimate)) +
  ggplot2::facet_grid(sex_plot_label ~ race_ethnicity_plot_label) +
  ggplot2::xlab("Year") +
  ggplot2::ylab("Estimated high school\nnon-completion percentage\n(in percentage points)")
```



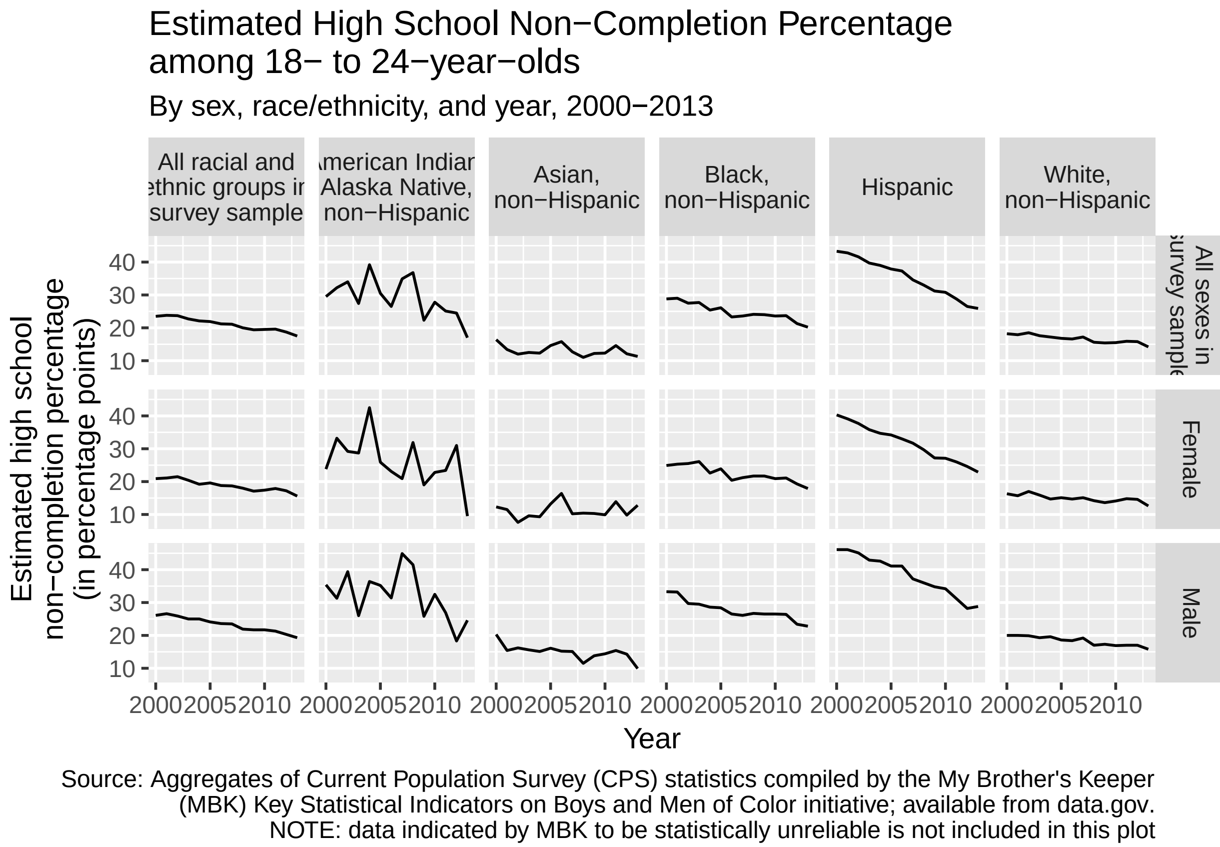
Adding a title is done in a similar way: we can use the `ggplot2::labs()` function:

```
ggplot2::ggplot(data = full_data_df) +
  ggplot2::geom_line(mapping = aes(x = year, y = percentage_estimate)) +
  ggplot2::facet_grid(sex_plot_label ~ race_ethnicity_plot_label) +
  ggplot2::xlab("Year") +
  ggplot2::ylab("Estimated high school\nnon-completion percentage\n(in percentage points)") +
  ggplot2::labs(title = "Estimated High School Non-Completion Percentage\namong 18- to 24-year-olds",
              subtitle = "By sex, race/ethnicity, and year, 2000-2013")
```



In our code, we can also display a caption on our image. Here, we might include notes about the provenance and limitations of our data:

```
ggplot2::ggplot(data = full_data_df) +
  ggplot2::geom_line(mapping = aes(x = year, y = percentage_estimate)) +
  ggplot2::facet_grid(sex_plot_label ~ race_ethnicity_plot_label) +
  ggplot2::xlab("Year") +
  ggplot2::ylab("Estimated high school\nnon-completion percentage\n(in percentage points)") +
  ggplot2::labs(title = "Estimated High School Non-Completion Percentage\namong 18- to 24-year-olds",
                subtitle = "By sex, race/ethnicity, and year, 2000-2013",
                caption = "Source: Aggregates of Current Population Survey (CPS) statistics compiled by
```



In total, we've now seen how to customize our visualization and get it closer to a form that's ready to be presented to a public audience.

Appendix

Appendix A: Data Cleaning

In this appendix, we'll talk about some of the work that went into cleaning the raw data found at the My Brother's Keeper data.gov site.

At first, this data looks like:

```
mbk_all_data_df
```

```
## # A tibble: 336 x 10
##    Characteristic Sex   `Race/ethnicity`  Year Percentage `Standard Error~
##    <chr>          <chr> <chr>            <int> <chr>                 <dbl>
##  1 Total          <NA>  <NA>              2000 23.5%                  0.52
##  2 Total          <NA>  <NA>              2001 23.8%                  0.52
##  3 Total          <NA>  <NA>              2002 23.7%                  0.36
##  4 Total          <NA>  <NA>              2003 22.7%                  0.36
##  5 Total          <NA>  <NA>              2004 22.1%                  0.35
##  6 Total          <NA>  <NA>              2005 21.9%                  0.42
##  7 Total          <NA>  <NA>              2006 21.2%                  0.36
##  8 Total          <NA>  <NA>              2007 21.1%                  0.37
```

```
##  9 Total          <NA>  <NA>              2008 20%                         0.34
## 10 Total          <NA>  <NA>              2009 19.4%                       0.34
## # ... with 326 more rows, and 4 more variables: `Note on
## #   Percentage` <chr>, `Count (in thousands)` <int>, `Standard Error on
## #   Count (in thousands)` <dbl>, `Note on Count` <chr>
```

A.1 Missing Data

One thing we can note is that are *missing* in our data set. This missingness might affect the kinds of visualizations we can later make, so we'll try to get a sense for it up front.

Firstly, let's define a function that lets us find all the columns that have missing values (making use of this very helpful Stack Overflow post):

```
cols_with_NAs <- function(df) {
  return(colnames(df)[colSums(is.na(df)) > 0])
}
```

Now, let's use this function to find the names of all columns that have missing values.

```
cols_with_NAs(mbk_all_data_df)
```

```
## [1] "Sex"
## [2] "Race/ethnicity"
## [3] "Percentage"
## [4] "Standard Error on Percentage"
## [5] "Note on Percentage"
## [6] "Count (in thousands)"
## [7] "Standard Error on Count (in thousands)"
## [8] "Note on Count"
```

Looks like most of the colums in our dataset have missing values. But, things might be missing in different places for different reasons, as we'll consider below.

A.2 Cleaning the data

Now, let's take some steps to clean the data before we get to plotting in. We'll think about what we might do for each of the columns in the dataset.

`Percentage`

We can note right away that `readr::read_csv` read in `Percentage` as a *character* column. We, however, want it as a *numeric* type. We can do this as follows:

```
mbk_all_data_df <- mbk_all_data_df %>%
  mutate(
    Percentage = stringr::str_sub(Percentage, start = 1, end = -2),
    Percentage = as.double(Percentage)
  )
```

Sex

For the `Sex` column, one of the things we want to check about it are its unique values. Here, we'll make use of the `dplyr::select()` function, and another function, `unique()` (a standard, or "base" R function), that will give us back the unique values in the column we're interested in:

```
mbk_all_data_df %>%
  dplyr::select(Sex) %>%
  unique()
```

```
## # A tibble: 3 x 1
##   Sex
##   <chr>
## 1 <NA>
## 2 Male
## 3 Female
```

So we see that the `Sex` column takes on three values in our dataset: `Female`, `Male`, and `NA`. What's going on with the `NA` cases? To try to figure out, we can use the `dplyr::filter()` function, along with the `is.na()` function (which tells us whether or not a value is null, row-by-row):

```
mbk_all_data_df %>%
  dplyr::filter(is.na(Sex))
```

```
## # A tibble: 112 x 10
##    Characteristic Sex   `Race/ethnicity`  Year Percentage `Standard Error~
##    <chr>          <chr> <chr>            <int>      <dbl>            <dbl>
##  1 Total          <NA>  <NA>              2000       23.5             0.52
##  2 Total          <NA>  <NA>              2001       23.8             0.52
##  3 Total          <NA>  <NA>              2002       23.7             0.36
##  4 Total          <NA>  <NA>              2003       22.7             0.36
##  5 Total          <NA>  <NA>              2004       22.1             0.35
##  6 Total          <NA>  <NA>              2005       21.9             0.42
##  7 Total          <NA>  <NA>              2006       21.2             0.36
##  8 Total          <NA>  <NA>              2007       21.1             0.37
##  9 Total          <NA>  <NA>              2008       20               0.34
## 10 Total          <NA>  <NA>              2009       19.4             0.34
## # ... with 102 more rows, and 4 more variables: `Note on
## #   Percentage` <chr>, `Count (in thousands)` <int>, `Standard Error on
## #   Count (in thousands)` <dbl>, `Note on Count` <chr>
```

Tabbing through, we can see that the only values for the `Characteristic` column that remain are those that have "Total" as the value and those that have "Total" along with one of the values found in the `Race/ethnicity` column. These values correspond to rows that are *irrespective of the value of the* ***`Sex`*** *column*—in other words, those that include statistics about both "Male" and "Female" groups combined, specific to each `Race/ethnicity` group. Because of this, one way we might think to recode the nulls in this group is to impute the value "All sexes in survey sample". We'll do so using the dplyr::mutate() function and the tidyr::replace_na() function:

```
mbk_all_data_df <- mbk_all_data_df %>%
  dplyr::mutate(
    Sex = tidyr::replace_na(Sex, "All sexes in survey sample")
  )
```

What are we left with now? We can see that the `Sex` column no longer has `NA` values:

```
mbk_all_data_df %>%
  dplyr::select(Sex) %>%
  unique()
```

```
## # A tibble: 3 x 1
##   Sex
##   <chr>
## 1 All sexes in survey sample
## 2 Male
## 3 Female
```

and that we get the same set of rows back when we filter to `Sex == "All sexes in survey sample"` as we did before when we filtered to `is.na(Sex)`:

```
mbk_all_data_df %>%
  dplyr::filter(Sex == "All sexes in survey sample")
```

```
## # A tibble: 112 x 10
##    Characteristic Sex   `Race/ethnicity`  Year Percentage `Standard Error~
##    <chr>          <chr> <chr>            <int>      <dbl>            <dbl>
##  1 Total          All ~ <NA>              2000       23.5             0.52
##  2 Total          All ~ <NA>              2001       23.8             0.52
##  3 Total          All ~ <NA>              2002       23.7             0.36
##  4 Total          All ~ <NA>              2003       22.7             0.36
##  5 Total          All ~ <NA>              2004       22.1             0.35
##  6 Total          All ~ <NA>              2005       21.9             0.42
##  7 Total          All ~ <NA>              2006       21.2             0.36
##  8 Total          All ~ <NA>              2007       21.1             0.37
##  9 Total          All ~ <NA>              2008       20               0.34
## 10 Total          All ~ <NA>              2009       19.4             0.34
## # ... with 102 more rows, and 4 more variables: `Note on
## #   Percentage` <chr>, `Count (in thousands)` <int>, `Standard Error on
## #   Count (in thousands)` <dbl>, `Note on Count` <chr>
```

`Race/ethnicity`

An analysis for missing values of the `Race/ethnicity` column would go similarly to the above analysis of missing values of the `Sex` column for this particular dataset. For brevity, we skip that analysis here and do a similar transformation of the data:

```
mbk_all_data_df <- mbk_all_data_df %>%
  dplyr::mutate(
    `Race/ethnicity` = tidyr::replace_na(`Race/ethnicity`, "All racial and ethnic groups in survey sampl
  )
```

`Characteristic`

With full data now present in both the `Sex` and `Race/ethnicity` columns, we can get rid of the `Characteristic` column, which contains the same information. We'll see later that for plotting, especially *faceting*, having this information separated into two columns will be helpful.

We'll get rid of the `Characteristic` column with the `dplyr::select()` function, as well as the `-` (minus) operator. What the command below basically says is "select all columns from `mbk_all_data_df` *except* (hence the minus sign) the `Characteristic` column, and assign the same name (`mbk_all_data_df`) back to the result":

```
mbk_all_data_df <- mbk_all_data_df %>%
  dplyr::select(-Characteristic)
```

Looking at `mbk_all_data_df`, we see that there's no longer a `Characteristic` column present:

```
mbk_all_data_df
```

```
## # A tibble: 336 x 9
##    Sex   `Race/ethnicity`  Year Percentage `Standard Error~
##    <chr> <chr>            <int>      <dbl>            <dbl>
##  1 All ~ All racial and ~  2000       23.5             0.52
##  2 All ~ All racial and ~  2001       23.8             0.52
##  3 All ~ All racial and ~  2002       23.7             0.36
##  4 All ~ All racial and ~  2003       22.7             0.36
##  5 All ~ All racial and ~  2004       22.1             0.35
##  6 All ~ All racial and ~  2005       21.9             0.42
##  7 All ~ All racial and ~  2006       21.2             0.36
##  8 All ~ All racial and ~  2007       21.1             0.37
##  9 All ~ All racial and ~  2008       20               0.34
## 10 All ~ All racial and ~  2009       19.4             0.34
## # ... with 326 more rows, and 4 more variables: `Note on
## #   Percentage` <chr>, `Count (in thousands)` <int>, `Standard Error on
## #   Count (in thousands)` <dbl>, `Note on Count` <chr>
```

Okay—let's pause for a second and see what are data look like now in terms of missingness:

```
cols_with_NAs(mbk_all_data_df)
```

```
## [1] "Percentage"
## [2] "Standard Error on Percentage"
## [3] "Note on Percentage"
## [4] "Count (in thousands)"
## [5] "Standard Error on Count (in thousands)"
## [6] "Note on Count"
```

So, we're closer. But let's investigate the further missingness.

`Percentage`

Let's see if we can deduce what's happening when the `Percentage` measurement is missing. This is a value that's going to be crucial for our visualizations, so we'll want to figure it out up front if possible. Let's again use the `dplyr::filter()` and `is.na()` functions we encountered above:

```
mbk_all_data_df %>%
  dplyr::filter(is.na(Percentage))
```

```
## # A tibble: 25 x 9
##    Sex   `Race/ethnicity`  Year Percentage `Standard Error~
##    <chr> <chr>            <int>      <dbl>            <dbl>
##  1 All ~ Pacific Islande~  2000         NA               NA
##  2 All ~ Pacific Islande~  2001         NA               NA
##  3 All ~ Pacific Islande~  2002         NA               NA
##  4 All ~ Two or more rac~  2000         NA               NA
##  5 All ~ Two or more rac~  2001         NA               NA
##  6 All ~ Two or more rac~  2002         NA               NA
##  7 Male  Pacific Islande~  2000         NA               NA
##  8 Male  Pacific Islande~  2001         NA               NA
##  9 Male  Pacific Islande~  2002         NA               NA
## 10 Male  Pacific Islande~  2003         NA               NA
## # ... with 15 more rows, and 4 more variables: `Note on Percentage` <chr>,
## #   `Count (in thousands)` <int>, `Standard Error on Count (in
## #   thousands)` <dbl>, `Note on Count` <chr>
```

Let's try *sorting* the rows of the dataset to see if we can see any kind of patterns that emerge in the missingness of `Percentage`. To do so, we'll use the `dplyr::arrange()` function. Below, we use `dplyr::arrange()` to sort (in ascending alphabetical or numeric order, depending on the data type of the column) first by `Race/ethnicity`, then by `Sex`, and finally by `Year`:

```
mbk_all_data_df %>%
  dplyr::filter(is.na(Percentage)) %>%
  dplyr::arrange(`Race/ethnicity`, Sex, Year)
```

```
## # A tibble: 25 x 9
##    Sex   `Race/ethnicity`  Year Percentage `Standard Error~
##    <chr> <chr>            <int>      <dbl>            <dbl>
##  1 All ~ Pacific Islande~  2000         NA               NA
##  2 All ~ Pacific Islande~  2001         NA               NA
##  3 All ~ Pacific Islande~  2002         NA               NA
##  4 Fema~ Pacific Islande~  2000         NA               NA
##  5 Fema~ Pacific Islande~  2001         NA               NA
##  6 Fema~ Pacific Islande~  2002         NA               NA
##  7 Fema~ Pacific Islande~  2003         NA               NA
##  8 Fema~ Pacific Islande~  2004         NA               NA
##  9 Fema~ Pacific Islande~  2005         NA               NA
## 10 Fema~ Pacific Islande~  2012         NA               NA
## # ... with 15 more rows, and 4 more variables: `Note on Percentage` <chr>,
## #   `Count (in thousands)` <int>, `Standard Error on Count (in
## #   thousands)` <dbl>, `Note on Count` <chr>
```

We can see that there *is* some regularity to the missingness here: things are missing only in rows that have a `Race/ethnicity` value of "Pacific Islander, non-Hispanic" or "Two or more races, non-Hispanic". We can also note that there are *some* cases where we have data for "Pacific Islander, non-Hispanic" and "Two or more races, non-Hispanic" rows; it's only for certain `Year` values that we'll have missing data for these groups.

This is an exmaple where missingness might have some reason other than just a data coding convention of the kind we saw for missingness about the `Sex` and `Race/ethnicity` columns. Here, we're actually given notes: in the `Note on Percentage` column: either data was not available, or reporting standards were not met. We're not going to do anything about this missingness immediately, but we'll pay attention to it again when we get to actually plotting things.

Let's pivot to looking more closely at the `Note on Percentage` column. Above, we considered it in the context of rows where the `Percentage` value is missing. But do we have any notes for rows where we have full `Percentage` data? Let's ask our dataset. Here, we'll use the very same functions as above, except we'll put a `!` (the logical negation operator in R) in front of our call to `is.na(Percentage)`, and add the condition `` !is.na(`Note on Percentage`) `` to our filter to get rows neither `Percentage` nor `Note on Percentage` is missing:

```
mbk_all_data_df %>%
  dplyr::filter(!is.na(Percentage), !is.na(`Note on Percentage`)) %>%
  dplyr::arrange(`Race/ethnicity`, Sex, Year) %>%
  dplyr::select(`Race/ethnicity`, Sex, Year, `Note on Percentage`)
```

```
## # A tibble: 23 x 4
##    `Race/ethnicity`        Sex         Year `Note on Percentage`
##    <chr>                   <chr>      <int> <chr>
##  1 American Indian/Alaska~ Female      2000 Interpret data with caution. ~
##  2 Pacific Islander, non-~ All sexes~  2003 Interpret data with caution. ~
##  3 Pacific Islander, non-~ All sexes~  2004 Interpret data with caution. ~
##  4 Pacific Islander, non-~ All sexes~  2006 Interpret data with caution. ~
##  5 Pacific Islander, non-~ All sexes~  2007 Interpret data with caution. ~
##  6 Pacific Islander, non-~ All sexes~  2008 Interpret data with caution. ~
##  7 Pacific Islander, non-~ All sexes~  2009 Interpret data with caution. ~
##  8 Pacific Islander, non-~ All sexes~  2012 Interpret data with caution. ~
##  9 Pacific Islander, non-~ All sexes~  2013 Interpret data with caution. ~
## 10 Pacific Islander, non-~ Female      2006 Interpret data with caution. ~
## # ... with 13 more rows
```

So it looks like this data source does not have *reliable* information about the "Pacific Islander, non-Hispanic" group in lots of years, even though there's a value reported. This will be something that we'll want to note (as a caption, subtitle, or something else) if we end up using these data for visualization.

For convenience, going forward we'll separate our data into two mutually exclusive datasets:

- `missing_data_df`: a data set where we have missing data for some cases
- `full_data_df`: a data set where we have full data for all cases

This will require a couple steps.

First, we'll step back and take a slightly different view on our data. Eventually, the visualizations we're going to talk about will (in the ideal) incorporate both the `Percentage` and `Standard Error on Percentage` columns. So, we want to distinguish between subgroups (defined, for the time being, by the intersection of `Sex` and `Race/ethnicity`) of our dataset that have both `Percentage` and `Standard Error on Percentage` information for all years in question, and those for which we this information is not all present.

To accomplish this, we'll use two new functions: `dplyr::group_by()` and `dplyr::summarize()`. `dplyr::group_by()` will set our data up so that we can do operations on entire groups of our data at a time, and `dplyr::summarize()` will allow us to compute *summaries* (think here of "summary statistics" like mean, median, variance, etc.) of each group of our data.

We do the following:

```
group_data_is_missing <- mbk_all_data_df %>%
  dplyr::group_by(Sex, `Race/ethnicity`) %>%
  dplyr::summarize(has_relevant_NAs = any(is.na(Percentage) | is.na(`Standard Error on Percentage`))) %>%
  dplyr::select(Sex, `Race/ethnicity`, has_relevant_NAs) %>%
  dplyr::ungroup()
```

What's the result? Let's look:

```r
group_data_is_missing
```

```
## # A tibble: 24 x 3
##    Sex                  `Race/ethnicity`                    has_relevant_NAs
##    <chr>                <chr>                               <lgl>
##  1 All sexes in surve~ All racial and ethnic groups in s~ FALSE
##  2 All sexes in surve~ American Indian/Alaska Native, no~ FALSE
##  3 All sexes in surve~ Asian, non-Hispanic                 FALSE
##  4 All sexes in surve~ Black, non-Hispanic                 FALSE
##  5 All sexes in surve~ Hispanic                            FALSE
##  6 All sexes in surve~ Pacific Islander, non-Hispanic      TRUE
##  7 All sexes in surve~ Two or more races, non-Hispanic     TRUE
##  8 All sexes in surve~ White, non-Hispanic                 FALSE
##  9 Female               All racial and ethnic groups in s~ FALSE
## 10 Female               American Indian/Alaska Native, no~ FALSE
## # ... with 14 more rows
```

Basically, what we know have is a filter condition for each `Sex/Race/ethnicity` group in our dataset. We can use this to construct the two datasets that we want. First, though, we'll need to *join* this new table back to `mbk_all_data_df`. We'll make use the `dplyr::inner_join()` to connect these two datasets. Below, we'll tell the `dplyr::inner_join()` function to look for cases where the values of both the `Sex` and the `Race/ethnicity` values match up in both tables, and associate the value of `has_relevant_NAs` accordingly:

```r
mbk_all_data_df <- dplyr::inner_join(mbk_all_data_df, group_data_is_missing,
                                     by = c("Sex", "Race/ethnicity"))
```

We now have all we need in place to do a straightforward filter of our dataset to get the kinds of `missing_data_df` and `full_data_df` tables we want.

```r
missing_data_df <- mbk_all_data_df %>%
  dplyr::filter(has_relevant_NAs == TRUE)

full_data_df <- mbk_all_data_df %>%
  dplyr::filter(has_relevant_NAs == FALSE)
```

Let's see what we're now left with:

```r
missing_data_df
```

```
## # A tibble: 84 x 10
##    Sex   `Race/ethnicity`  Year Percentage `Standard Error~
##    <chr> <chr>            <int>      <dbl>            <dbl>
##  1 All ~ Pacific Islande~  2000       NA               NA
##  2 All ~ Pacific Islande~  2001       NA               NA
##  3 All ~ Pacific Islande~  2002       NA               NA
##  4 All ~ Pacific Islande~  2003       10.8              4.62
##  5 All ~ Pacific Islande~  2004       14.4              5.06
##  6 All ~ Pacific Islande~  2005       17.7              4.8
##  7 All ~ Pacific Islande~  2006       17.2              5.45
##  8 All ~ Pacific Islande~  2007       10.5              3.74
```

```
##  9 All ~ Pacific Islande~  2008       17.1             5.68
## 10 All ~ Pacific Islande~  2009       15.6             5.76
## # ... with 74 more rows, and 5 more variables: `Note on Percentage` <chr>,
## #   `Count (in thousands)` <int>, `Standard Error on Count (in
## #   thousands)` <dbl>, `Note on Count` <chr>, has_relevant_NAs <lgl>
```

```
full_data_df
```

```
## # A tibble: 252 x 10
##    Sex   `Race/ethnicity`  Year Percentage `Standard Error~
##    <chr> <chr>            <int>      <dbl>            <dbl>
##  1 All ~ All racial and ~  2000       23.5             0.52
##  2 All ~ All racial and ~  2001       23.8             0.52
##  3 All ~ All racial and ~  2002       23.7             0.36
##  4 All ~ All racial and ~  2003       22.7             0.36
##  5 All ~ All racial and ~  2004       22.1             0.35
##  6 All ~ All racial and ~  2005       21.9             0.42
##  7 All ~ All racial and ~  2006       21.2             0.36
##  8 All ~ All racial and ~  2007       21.1             0.37
##  9 All ~ All racial and ~  2008       20               0.34
## 10 All ~ All racial and ~  2009       19.4             0.34
## # ... with 242 more rows, and 5 more variables: `Note on
## #   Percentage` <chr>, `Count (in thousands)` <int>, `Standard Error on
## #   Count (in thousands)` <dbl>, `Note on Count` <chr>,
## #   has_relevant_NAs <lgl>
```

This completes the data cleaning steps taken on the original data. With this work finished, visualization is a more straigtforward task.