Graphing categorical data

Put your name here

Put the date here

Introduction

In this module, we will use the `ggplot2` package for creating nicely formatted charts and graphs for categorical data.

Instructions

Presumably, you have already created a new project and downloaded this file into it. From the `Run` menu above, select `Run All` to run all existing code chunks.

When prompted to complete an exercise or demonstrate skills, you will see the following lines in the document:

ANSWER

These lines demarcate the region of the R Markdown document in which you are to show your work.

Sometimes you will be asked to add your own R code. That will appear in this document as a code chunk with a request for you to add your own code, like so:

```
# Add code here
```

Be sure to remove the line `# Add code here` when you have added your own code. You should run each new code chunk you create by clicking on the dark green arrow in the upper-right corner of the code chunk.

Sometimes you will be asked to type up your thoughts. That will appear in the document with the words, "Please write up your answer here." Be sure to remove the line "Please write up your answer here" when you have written up your answer. In these areas of the assignment, please use contextually meaningful full sentences/paragraphs (unless otherwise indicated) and proper spelling, grammar, punctuation, etc. This is not R code, but rather a free response section where you talk about your analysis and conclusions. You may need to use inline R code in these sections.

When you are finished with the assignment, knit to PDF and proofread the PDF file **carefully**. Do not download the PDF file from the PDF viewer; rather, you should export the PDF file to your computer by selecting the check box next to the PDF file in the Files pane, clicking the `More` menu, and then clicking `Export`. Submit your assignment according to your professor's instructions.

Load Packages

We load the `mosaic` package as well as the `MASS` package for working with the birth weight data. (Note that the `ggplot2` package we will use for graphing is automatically loaded alongside the `mosaic` package.)

```
library(MASS)
library(mosaic)
```

Working with factor variables

R uses the term "factor variable" to refer to a categorical variable. Your data set may already come with its variables coded correctly as factor variables, but often they are not. For example, our birth weight data `birthwt` has several categorical variables, but they are all coded numerically. (More specifically, they are all coded as integers.) Observe:

```
str(birthwt)
```

```
## 'data.frame':    189 obs. of  10 variables:
##  $ low  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ age  : int  19 33 20 21 18 21 22 17 29 26 ...
##  $ lwt  : int  182 155 105 108 107 124 118 103 123 113 ...
##  $ race : int  2 3 1 1 1 3 1 3 1 1 ...
##  $ smoke: int  0 0 1 1 1 0 0 0 1 1 ...
##  $ ptl  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ ht   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ ui   : int  1 0 0 1 1 0 0 0 0 0 ...
##  $ ftv  : int  0 3 1 2 0 0 1 1 1 0 ...
##  $ bwt  : int  2523 2551 2557 2594 2600 2622 2637 2637 2663 2665 ...
```

We are interested in the `race` variable. At the Console, type

?birthwt

and scroll down to read information about how the `race` variable is coded. You'll see that "1 = white, 2 = black, 3 = other".

The code below is somewhat involved and technical. After the code chunk, I'll explain what each piece does.

```
race <- factor(birthwt$race,
               levels = c(1, 2, 3),
               labels = c("White", "Black", "Other"))
race_df <- data.frame(race)
```

First of all, because `birthwt` is a dataset defined in the `MASS` package, we don't want to modify it. Therefore, if we want to change something, we have to assign a new name to the result of any operation. That is why we have `race <-` at the beginning of the code line. The symbol `<-` is taking the result of the command on the right (in this case, the `factor` command) and giving it a new name.

The `factor` command converts `birthwt$race` into a factor variable. The `levels` of the variable are the pre-existing numerical values. The `labels` are the names we actually want to appear in our output.

The letter `c` in `c(1, 2, 3)` and `c("White", "Black", "Other")` is necessary whenever we want to combine more than one thing into a single expression. (In technical terms, the "c" stands for "combine" or "concatenate" and creates a "vector".)

The last line takes the single vector `race` and turns it into a data frame that we call `race_df`. Many of the commands we will use require that we analyze variables that are sitting inside of data frames. Let's see how this worked. First, go to the Console and type

View(race_df)

(Be sure to remember that `View` has a capital "V". We're doing this in the Console because the `View` command has no effect inside an R Markdown document. There's no way for R Markdown to show the spreadsheet view of the data that's available in RStudio.)

You can see that `race_df` is a data frame with one variable (one column). That variable is called `race`.

Another way to look at variables in a data frame is to use the `str` command.

```
str(race_df)
```

```
## 'data.frame':    189 obs. of  1 variable:
##  $ race: Factor w/ 3 levels "White","Black",..: 2 3 1 1 1 3 1 3 1 1 ...
```

Finally, we can use `head` to print some of our data in the output.

```
head(race_df)
```

```
##    race
## 1 Black
## 2 Other
## 3 White
## 4 White
## 5 White
## 6 Other
```

You can see from the output that this created a data frame called `race_df` containing a single factor variable called `race` sitting inside it. The values of race are no longer numbers 1, 2, or 3. Instead, those numbers have been replaced by human-readable words describing the three racial categories.

If a variable is already coded as a factor variable in its data frame, it is important **not** to use the `factor` command on it. This will mess up the data and make all future attempts at analysis break. It is, therefore, extremely important that you check the original data frame first using `str`. **If the variable of interest is already coded as a factor variable, you cannot use the `factor` command.**

ggplot

The `ggplot` command is an all-purpose graphing utility. It uses a graphing philosophy derived from a book called *The Grammar of Graphics* by Leland Wilkinson. The basic idea is that each variable you want to plot should correspond to some element or "aesthetic" component of the graph. The obvious places for data to go are along the y-axis or x-axis, but other aesthetics are important too; graphs often use color, shape, or size to illustrate different aspects of data. Once these aesthetics have been defined, we will add "layers" to the graph. These are objects like dots, boxes, lines, or bars that dictate the type of graph we want to see.

In an introductory course, we won't get too fancy with these graphs. But be aware that there's a whole field of data visualization that studies clear and interesting ways to understand data graphically.

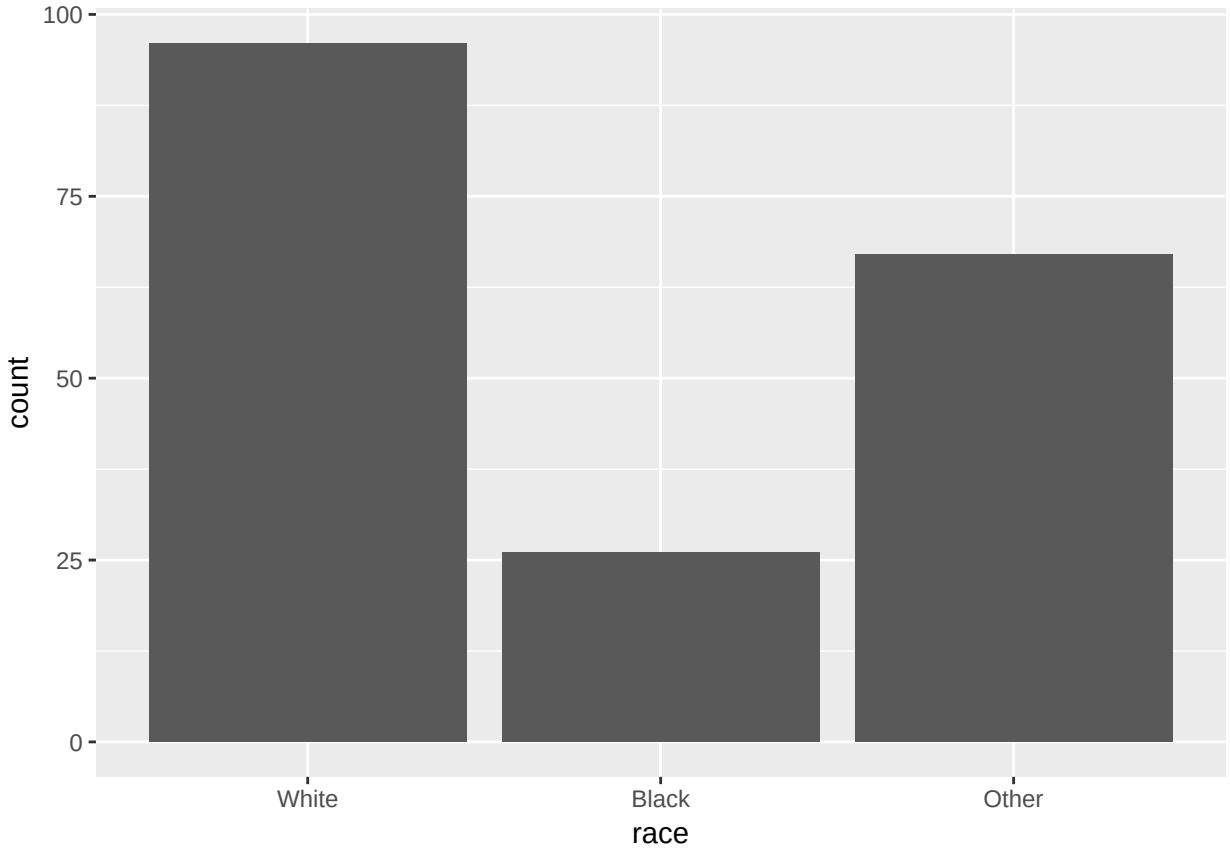
It will be easier to explain the `ggplot` syntax in the context of specific graph types, so let's proceed to the next section and start looking at ways to graph categorical data.

Graphing one categorical variable

When asked, "What type of graph should I use when graphing a single categorical variable?" the simple answer is "None." If you do need to summarize a categorical variable, a frequency table usually suffices. (See the module `Tables.Rmd`.)

If you really, really want a graph, the standard type is a bar chart. Here is the `ggplot` command to do that:

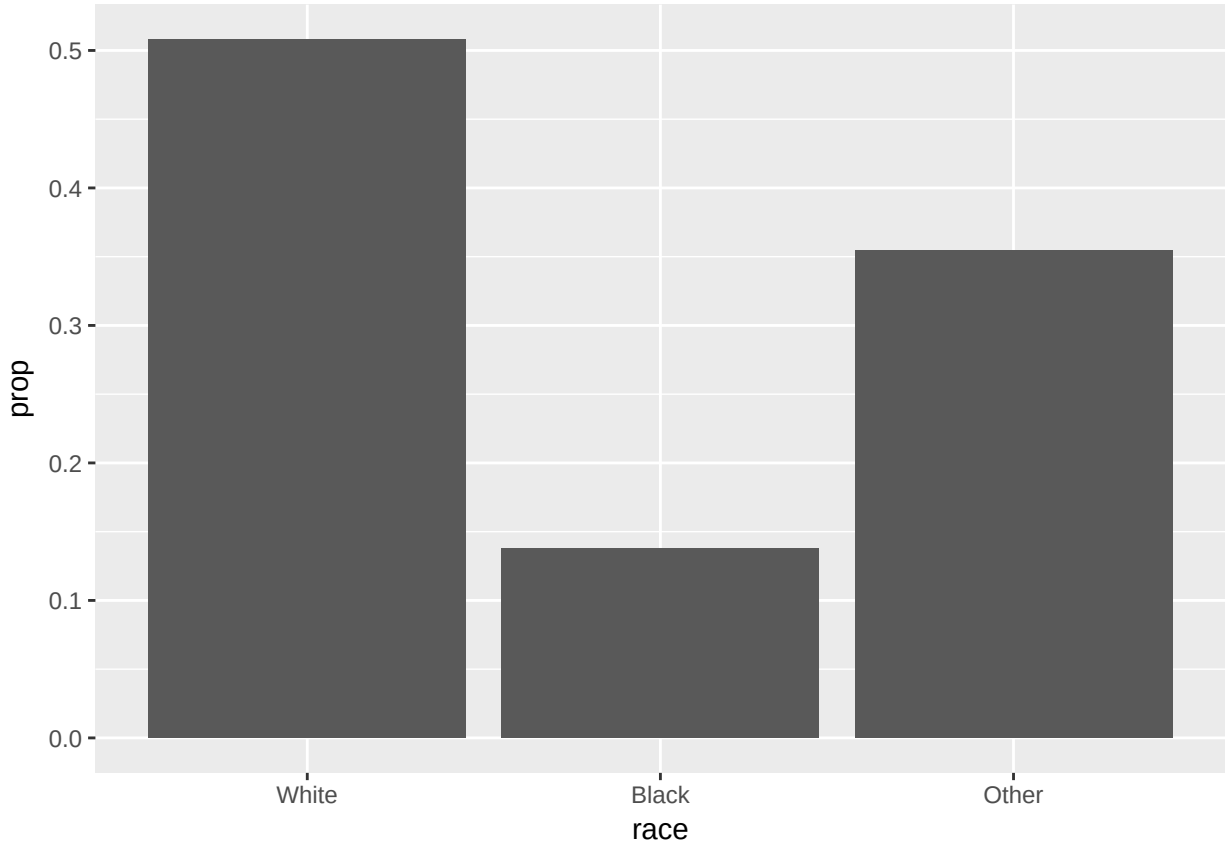
```
ggplot(race_df, aes(x = race)) +
    geom_bar()
```



Let's walk through this syntax step by step. The first argument of the `ggplot` command is the name of the data frame, in this case, `race_df`. Next we define the aesthetics using `aes` and parentheses. Inside the parentheses, we assign any variables we want to plot to aesthetics of the graph. For this analysis, we are only interested in the variable `race` and for a bar chart, the categorical variable typically goes on the x-axis. That's why it says `x = race` inside the `aes` argument. Next, `ggplot` needs to know what kind of graph we want. Graph types are called "geoms" in the `ggplot` world, and `geom_bar()` tells `ggplot` to add a bar chart layer. (Adding a layer is accomplished by literally typing a plus sign.)

This can be modified somewhat to give proportions (relative frequencies) on the y-axis instead of counts. Unfortunately, the `ggplot` syntax is not very transparent here. My recommendation is to copy and paste the code below if you need to make a relative frequency bar chart in the future, making the necessary changes to the data frame and variable, of course.

```
ggplot(race_df, aes(x = race, y = ..prop.., group = 1)) +
    geom_bar()
```



These bar charts are the graphical analogues of a frequency table and a relative frequency table, respectively. (See the module `Tables.Rmd`.)

Exercise

In a sentence or two at most, describe the distribution of race in this data set.

ANSWER

Please write up your answer here.

What about pie charts? Just. Don't.

Seriously. Pie charts suck.[1]

Graphing two categorical variables

A somewhat effective way to display two categorical variables is with a side-by-side bar chart.

First things first, though. We need to create one more factor variable. We'll use the `smoke` variable about whether the mothers smoked during pregnancy.

[1] https://medium.com/the-mission/to-pie-charts-3b1f57bcb34a

```
smoke <- factor(birthwt$smoke,
                levels = c(1, 0),
                labels = c("Yes", "No"))
smoke_race <- data.frame(smoke, race)
```

The `smoke` variable is created in exactly the same way as the `race` variable was earlier. Now, though, because we want to analyze both `smoke` and `race` together, we create a new data frame called `smoke_race` with both variables.

When we work with two variables, typically we think of one variable as response and the other as explanatory. For reasons that will become more clear later, we will adopt the convention of always listing the response variable first, followed by the explanatory variable. So we use `smoke_race` instead of `race_smoke`.

Let's make sure the above commands did what we intended. Type

View(smoke_race)

in the Console.

Then, we'll use `str` and `head` to print output here in this file.

```
str(smoke_race)
```

```
## 'data.frame':    189 obs. of  2 variables:
##  $ smoke: Factor w/ 2 levels "Yes","No": 2 2 1 1 1 2 2 2 1 1 ...
##  $ race : Factor w/ 3 levels "White","Black",..: 2 3 1 1 1 3 1 3 1 1 ...
```

```
head(smoke_race)
```

```
##   smoke  race
## 1    No Black
## 2    No Other
## 3   Yes White
## 4   Yes White
## 5   Yes White
## 6    No Other
```

Examine the output from the above code to make sure we have a data frame with the two factor variables we want.

You may be wondering why the command looked like this:

`smoke <- factor(birthwt$smoke, levels = c(1, 0), labels = c("Yes", "No"))`

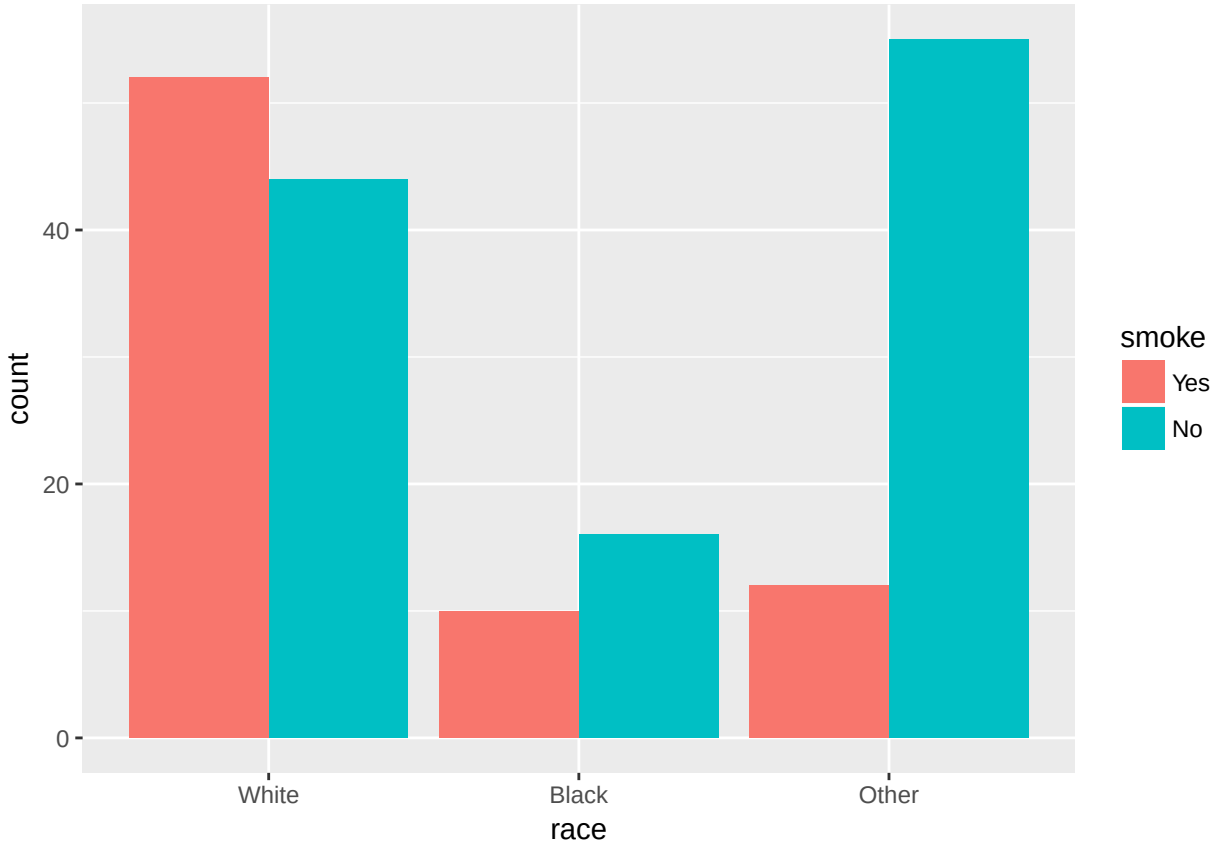
instead of like this:

`smoke <- factor(birthwt$smoke, levels = c(0, 1), labels = c("No", "Yes")).`

For most response variables, we often designate one category as a category of interest in our study. This category is often called the "Success" condition. For example, the question we're asking about this data is, "Within each racial category, what percentage of mothers smoked?" Therefore, the "Yes" condition in the `smoke` variable is considered the "Success" category (even though there's obviously nothing "successful" about smoking while pregnant!). When we list that success category first in the factor command, R will treat it a little differently than other categories. It's not so important to us here, but it will be very important in the future. This typically only matters for the response variable. (The explanatory variable is `race` and our research question does not single out one racial category as being of more interest to us than any other.)

Here the code for a side-by-side bar chart:

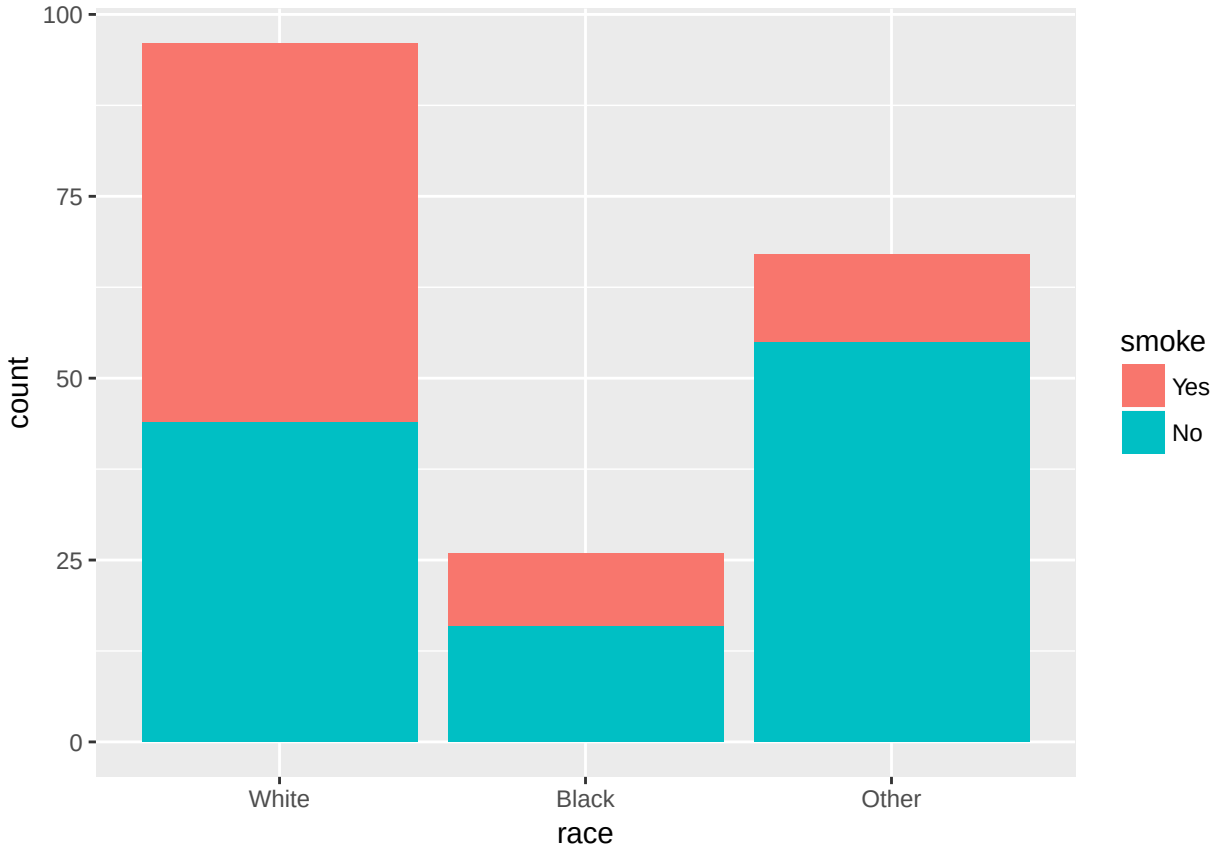
```
ggplot(smoke_race, aes(fill = smoke, x = race)) +
    geom_bar(position = "dodge")
```



This is somewhat different from other `ggplot` examples you've seen before, so let's take a moment to go through the syntax. The first argument is the data frame `smoke_race`; no mystery there. The second aesthetic `x = race` also makes a lot of sense. As `race` is our explanatory variable—we're using race to group the women, and then within each racial group, we're interested in how many women smoked during pregnancy—`race` goes on the x-axis. However, `smoke` does not go on the y-axis! (This is a very common mistake for novices.) The y-axis of a bar chart is always a count or a percentage, so no variable should ever go on the y-axis of a bar chart. In that case, how does `smoke` enter the picture? Through the use of color! The aesthetic `fill = smoke` says to use the `smoke` variable to shade or "fill" the bars with different colors. You'll also notice that `ggplot` makes a legend automatically with the colors so you can see which color corresponds to which value (in this case, "Yes" or "No").

Another unusual feature is the argument `position = "dodge"` in the `geom_bar` layer. Let's see what happens if we remove it.

```
ggplot(smoke_race, aes(fill = smoke, x = race)) +
    geom_bar()
```

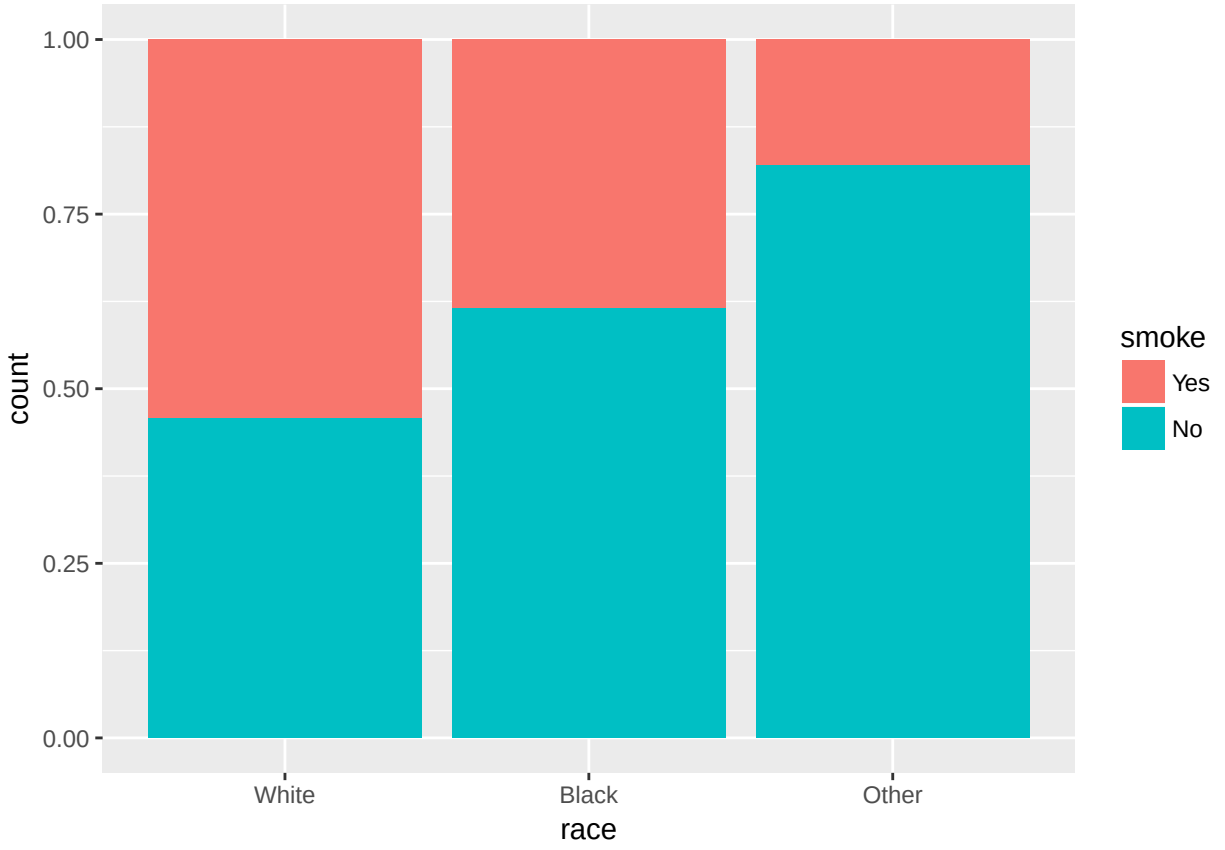


We get a stacked bar chart! This is another popular way of displaying two categorical variables, but I don't tend to prefer it. Notice how difficult it is to compare the number of smokers across races; since there is no common baseline for the red segments of each bar, it is harder to determine which ones are bigger or smaller. (In this case, it's obvious for the "White" group, but not so obvious between "Black" and "Other".)

So let's agree to use side-by-side bar charts. There is still one aspect of the side-by-side bar chart that is misleading, though. For example, there are 10 black women who smoked during pregnancy and 12 women from the "Other" category who smoked during pregnancy. So are women from the "Other" category more likely to smoke during pregnancy? No! The 10 black women are only out of 26 total black women, whereas the 12 other women are out of a total of 67. That's really 38.5% of black women compared to 17.9% other women.

To fix this problem, a better option here would be to use relative frequencies (i.e., percentages within each group) instead of counts on the y-axis. (This is analogous to using percentages in a contingency table; see the module `Tables.Rmd`.) Unfortunately, it is rather difficult to do this with `ggplot`. A compromise is available: by using `position = fill`, you can create a stacked bar chart that scales every group to 100%. Making comparisons across groups can still be hard, as explained above for any kind of stacked bar chart, but it works okay if there are only two categories in the response variable (as is the case with `smoke` here).

```
ggplot(smoke_race, aes(fill = smoke, x = race)) +
    geom_bar(position = "fill")
```



This graph does correctly show that a larger proportion of black women smoke during pregnancy when compared to women from the "Other" category. (And white women smoked the most of all; the majority of white women smoked during pregnancy in this data set!)

Your turn

Choose two categorical variables of interest from the `birthwt` data set. Identify one as response and one as explanatory. (Choose at least one variable other than `race` or `smoke`.) Turn them into factor variables with meaningful labels. (If you use `race` or `smoke` again, **do not** use factor again!) Create a new data frame containing both variables. Then create a side-by-side bar chart. Comment on the association (or independence) of the two variables.

ANSWER

```
# Add code here to convert one or more variables to factor variables
# and make a data frame.
```

```
# Add code here to make a side-by-side bar chart.
```

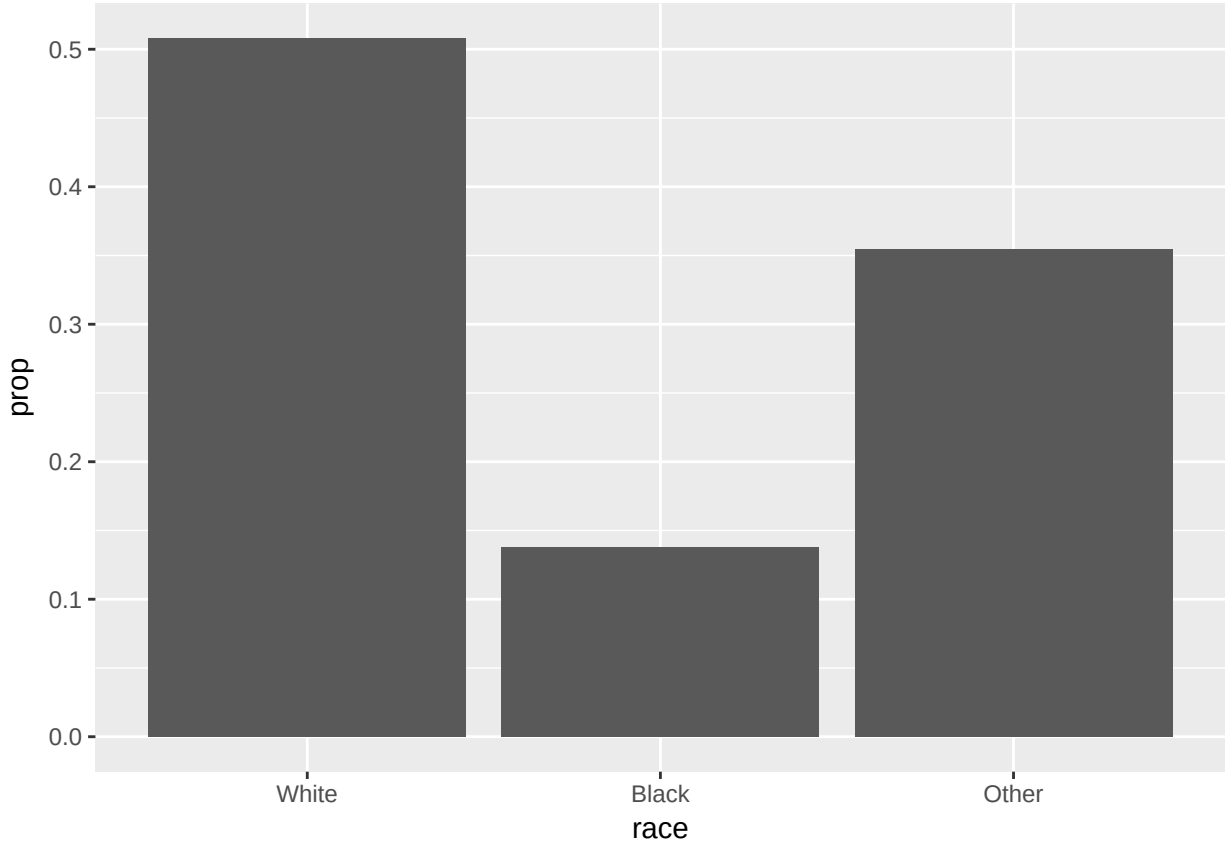
Please write up your answer here.

Publication-ready graphics

The great thing about `ggplot2` graphics is that they are already quite pretty. To take them from exploratory data analysis to the next level, there are a few things we can do to tidy them up.

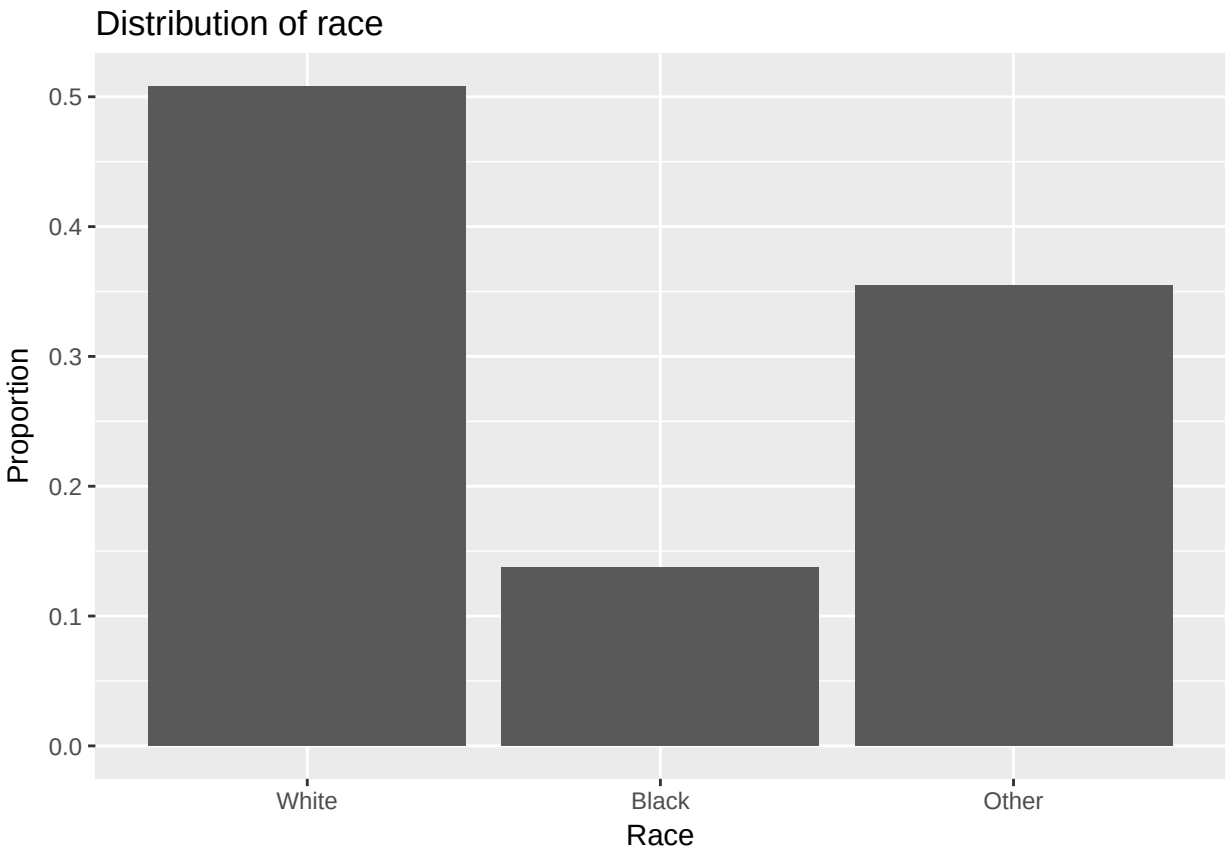
Let's go back to the first relative frequency bar chart from this module.

```
ggplot(race_df, aes(x = race, y = ..prop.., group = 1)) +
    geom_bar()
```



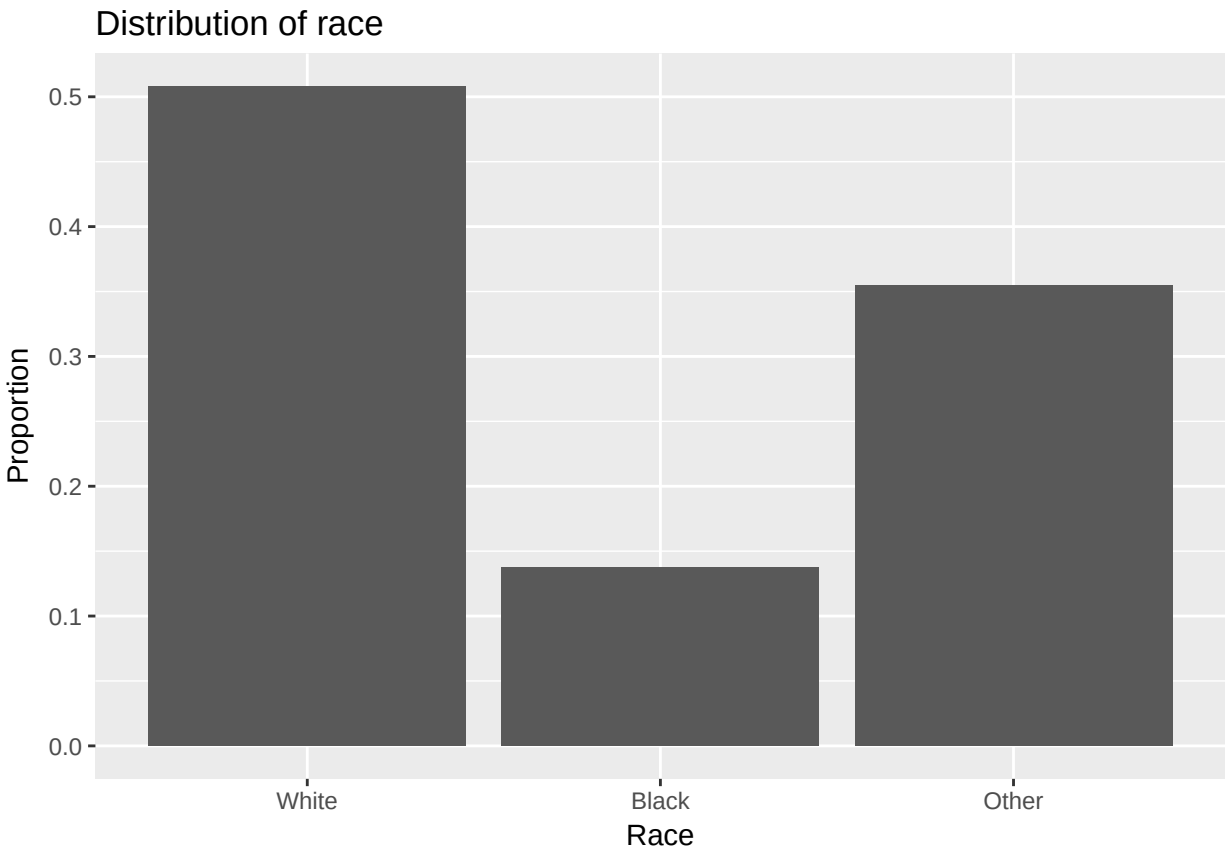
The variable name `race` is already informative, but the y-axis is labeled with "prop". Also note that this graph could use a title. We can do all this with `labs` (for labels). Observe:

```
ggplot(race_df, aes(x = race, y = ..prop.., group = 1)) +
    geom_bar() +
    labs(title = "Distribution of race",
         y = "Proportion",
         x = "Race")
```



A quick note about formatting in R code chunks. Notice that I put different parts of the last `ggplot` command on their own separate lines. The command would still work if I did this:

```
ggplot(race_df, aes(x = race, y = ..prop.., group = 1)) + geom_bar() + labs(title = "Distribution of ra
```



In the R Markdown document, the code chunk "wraps" to the next line so it's all visible. But now knit the document and look at the PDF version of the code chunk above.

Do you see how it runs off the right edge of the page?

Now imagine your PDF output is being graded and the grader can only see the first few lines of code before it runs off the edge of the paper. Do you think you will earn full points for code that the grader can't even see?

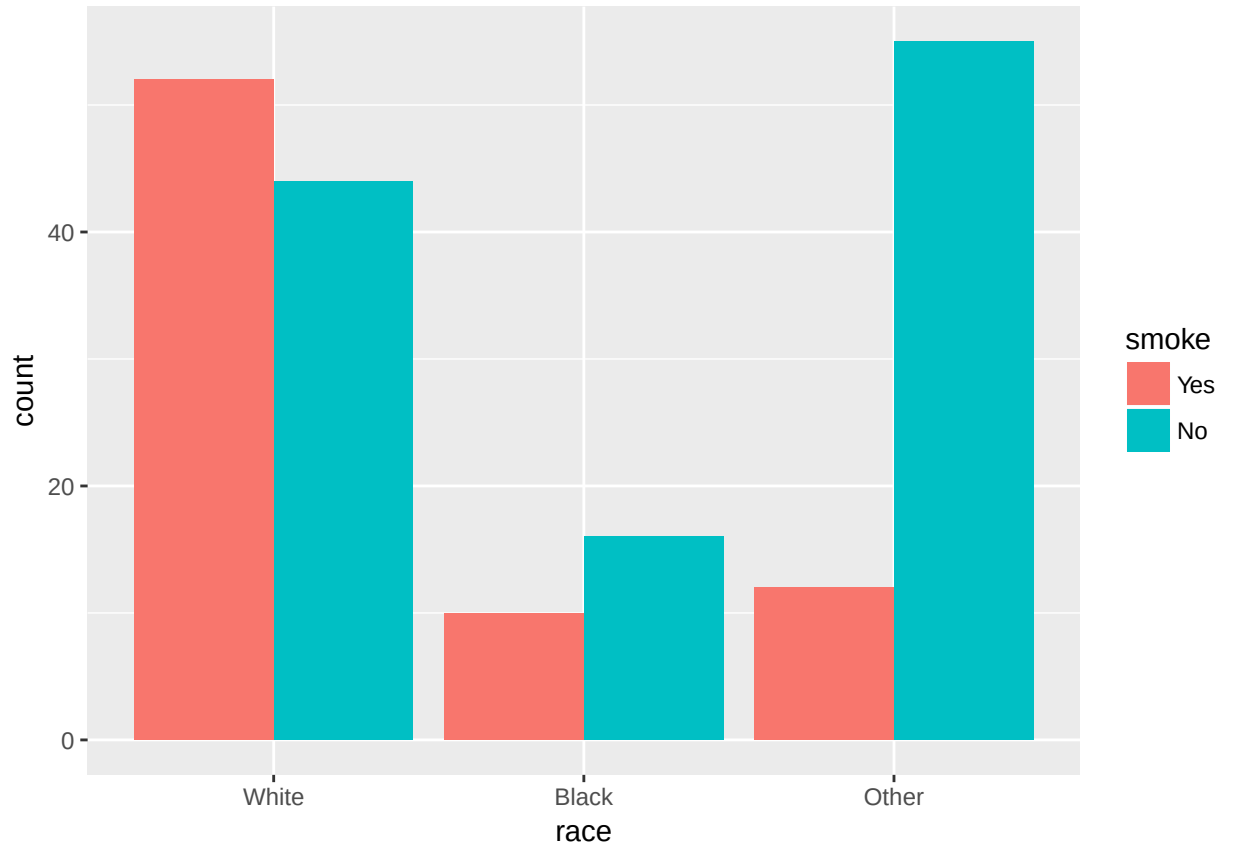
The moral of the story is this: use line breaks judiciously to format your code. If it wraps to the next line in RStudio, there's a good chance it will run off the page in the PDF.

Exercise

Modify the following side-by-side bar chart by adding a title and labels for both the fill variable and the x-axis variable.

ANSWER

```
# Modify the following side-by-side bar chart by adding a title and
# labels for both the x-axis and the fill variable.
ggplot(smoke_race, aes(fill = smoke, x = race)) +
    geom_bar(position = "dodge")
```



Every part of the graph can be customized, from the color scheme to the tick marks on the axes, to the major and minor grid lines that appear on the background. We won't go into all that, but you can look at the ggplot2 documentation online and search Google for examples if you want to dig in and figure out how to do some of that stuff. However, the default options are often (but not always) the best, so be careful that your messing around doesn't inadvertently make the graph less clear or less appealing.

Conclusion

For a single categorical variable, a chart is usually overkill; just use a frequency table. But if you really want a chart, the bar chart is the best option.

For two categorical variables, the best choice is usually a side-by-side bar chart. A stacked bar chart will also work, especially if using relative frequencies on the y-axis, but it can be hard to compare across groups when the response variable has three or more categories.