Graphing grouped numerical 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 grouped numerical 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 data on risk factors associated with low birth weight. (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 ...
```

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.

```
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 ...
```

```
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 x-axis or y-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 grouped numerical data.

Graphing grouped numerical data

Suppose you want to analyze one numerical variable and one categorical variable. Usually, the idea here is that the categorical variable divides up the data into groups and you are interested in understanding the numerical variable for each group separately. Another way to say this is that your numerical variable is response and your categorical variable is explanatory. (It is also possible for a categorical variable to be response and a numerical variable to be explanatory. This is common in so-called "classification" problems. We will not cover this possibility in this course, but it is covered in more advanced courses.)

For an example, let's consider the mother's weight by race. To be able to work with our newly labeled `race` variable, we'll need to make a data frame that contains it alongside any other variables we want to analyze (in this case, `lwt`). In the following code, note that `race` can be included in the data frame as is, but `birthwt$lwt` has to be renamed as `lwt`. This is because there is a variable called `race` defined in the environment. (We created it earlier by converting the variable `birthwt$race` to a factor variable.) But there is no variable called `lwt` in the environment. Of course, we don't need to *convert* `birthwt$lwt` because it's already coded numerically (in this case as `int`, meaning an integer, or whole number). But we do need to rename the variable so it enters the new data frame with the concise name `lwt`.

```
lwt_race <- data.frame(lwt = birthwt$lwt, race)
```

We check to make sure this worked.

```
str(lwt_race)
```

```
## 'data.frame':    189 obs. of  2 variables:
##  $ lwt : int  182 155 105 108 107 124 118 103 123 113 ...
##  $ race: Factor w/ 3 levels "White","Black",..: 2 3 1 1 1 3 1 3 1 1 ...
```

```
head(lwt_race)
```

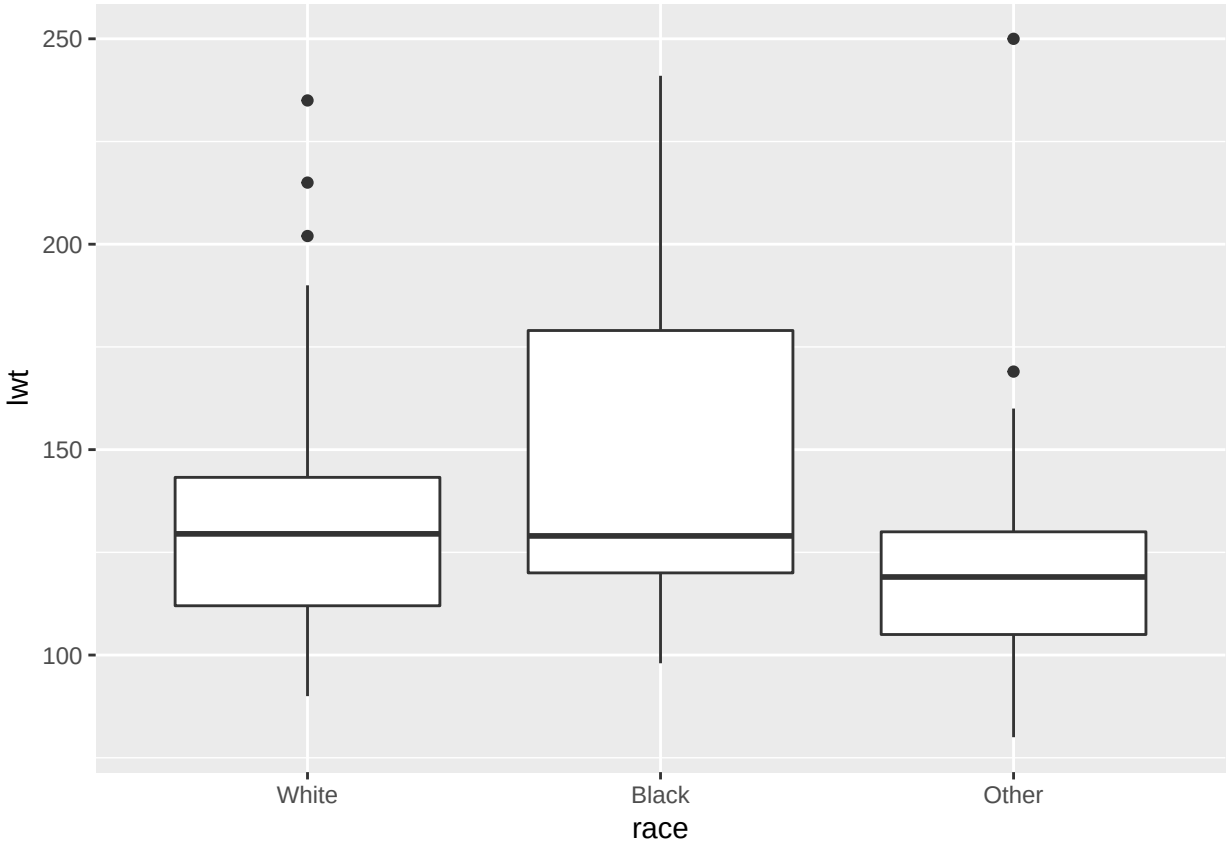
```
##   lwt  race
## 1 182 Black
## 2 155 Other
```

```
## 3 105 White
## 4 108 White
## 5 107 White
## 6 124 Other
```

Indeed, we see in the output that there are two variables, one numerical (`lwt`), and one categorical (the factor variable `race`).

Graphically, there are two good options here. The first is a side-by-side boxplot.

```
ggplot(lwt_race, aes(y =lwt, x = race)) +
    geom_boxplot()
```

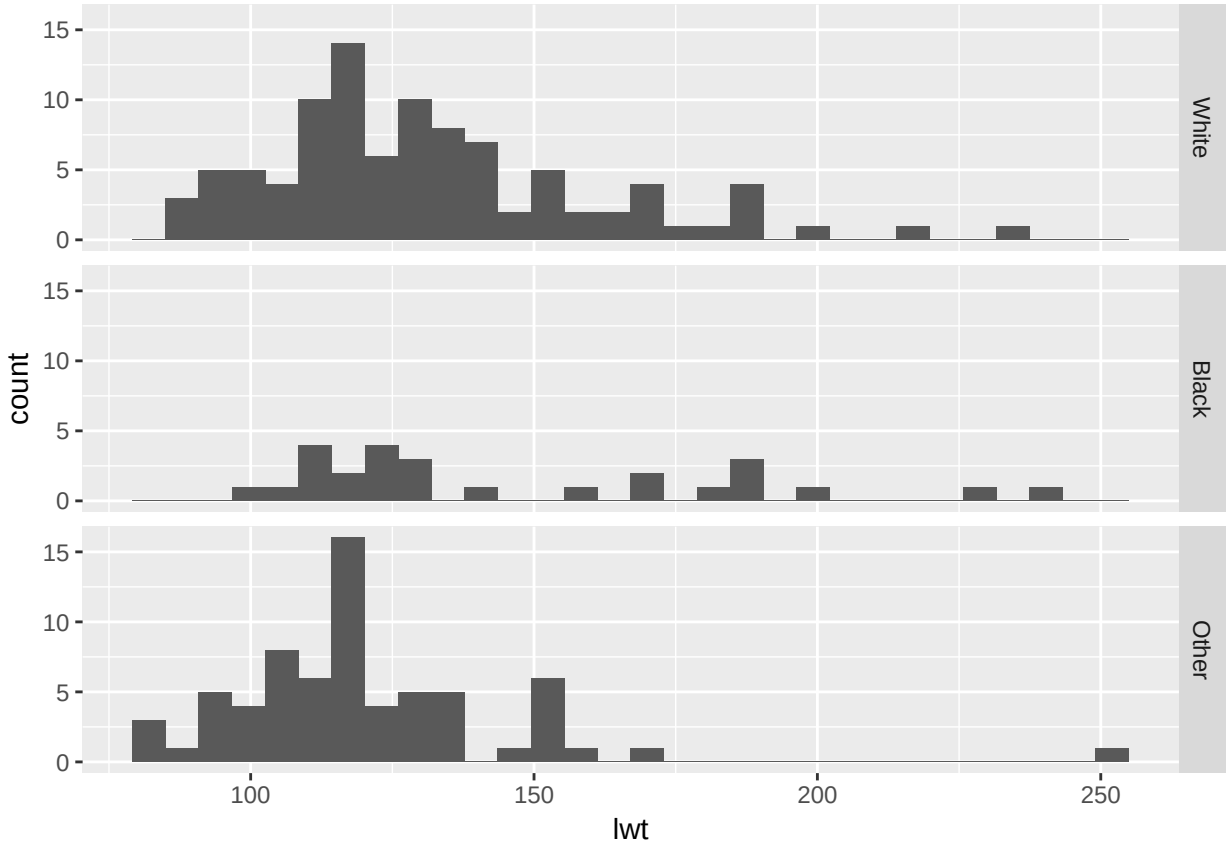


Notice the placement of the variables. The y-axis is `lwt`, the numerical variable. The x-axis variable is `race`; the groups are placed along the x-axis. This is consistent with other graph types that place the response variable on the y-axis and the explanatory variable on the x-axis.

The other possible graph is a stacked histogram. This uses a feature called "faceting" that creates a different plot for each group. The syntax is a little unusual.

```
ggplot(lwt_race, aes(x = lwt)) +
    geom_histogram() +
    facet_grid(race ~ .)
```

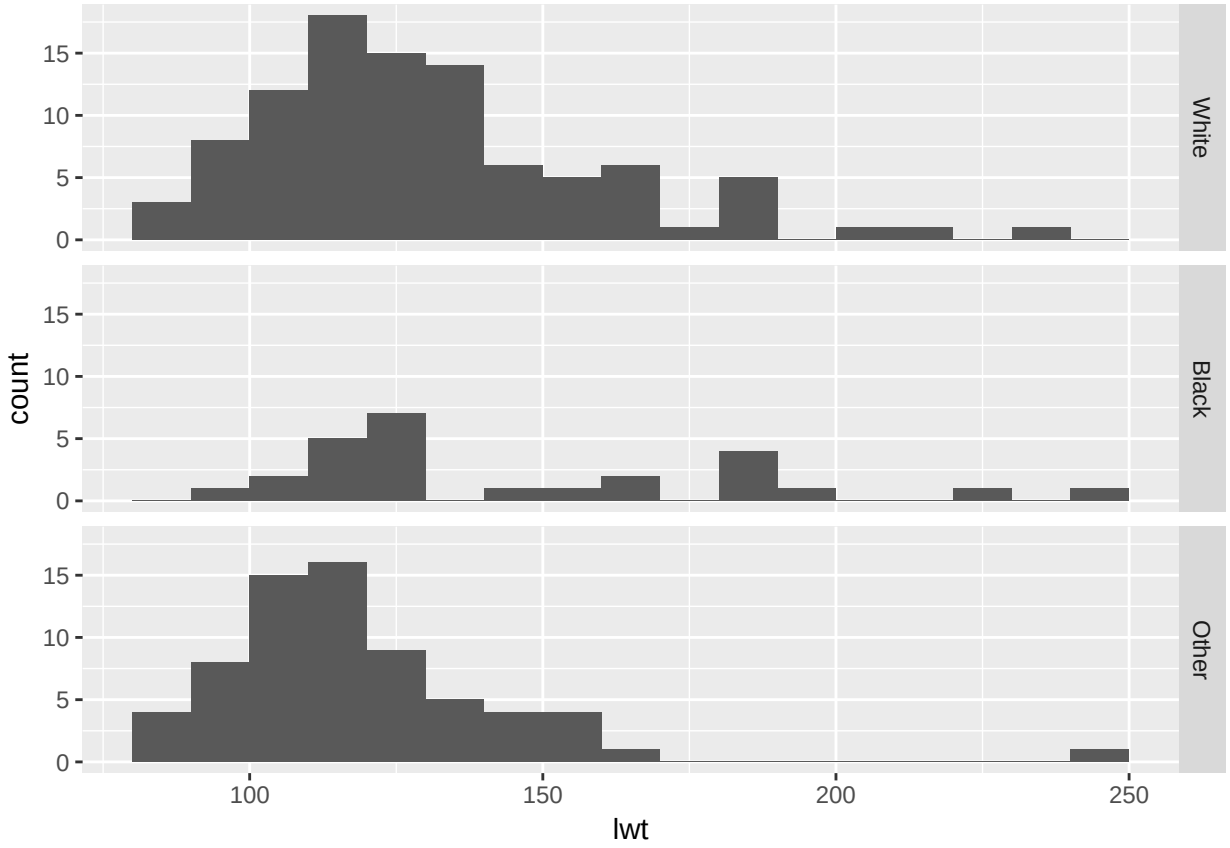
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The argument `race ~ .` in the `facet_grid` function means, "Put each race on a different row." We'll explore this notation a little later.

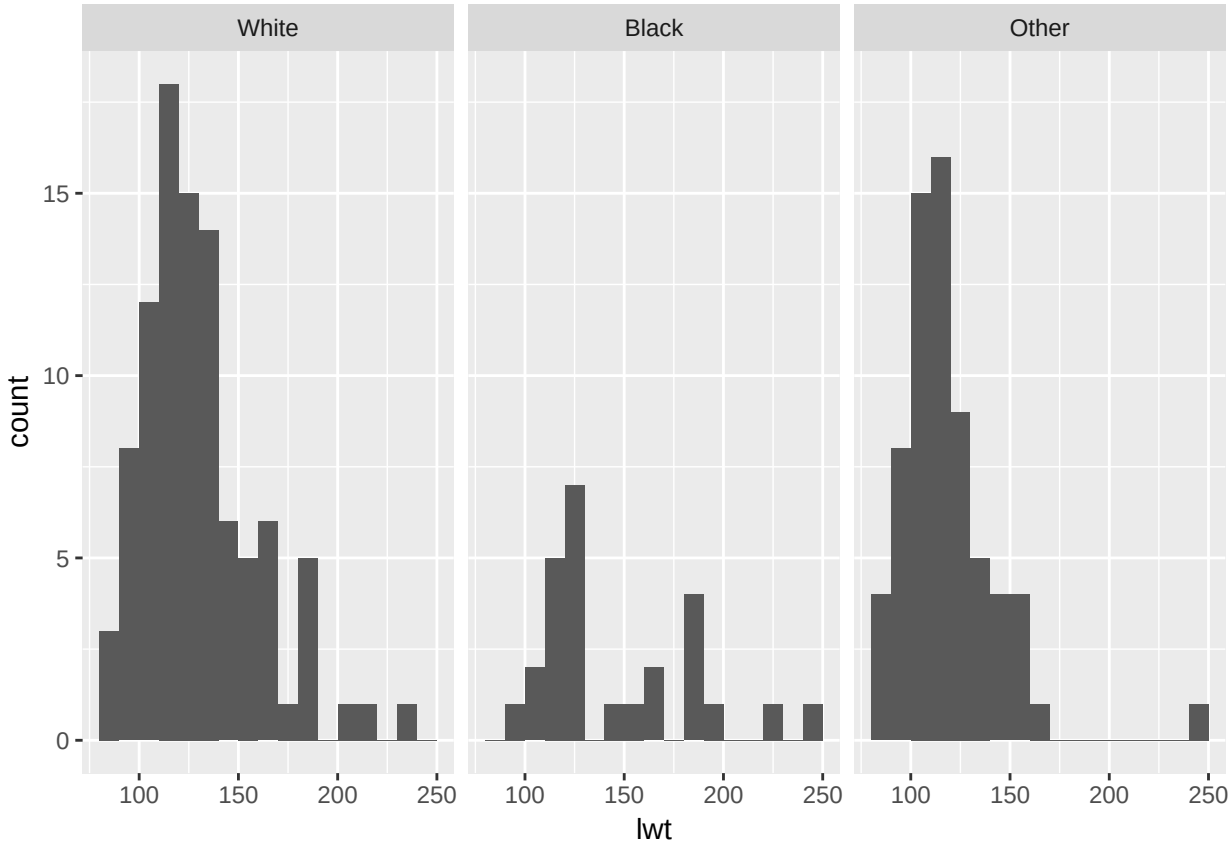
As always, the default bins suck, so let's change them.

```
ggplot(lwt_race, aes(x = lwt)) +
    geom_histogram(binwidth = 10, boundary = 100) +
    facet_grid(race ~ .)
```



Consider the following subtle change in notation:

```
ggplot(lwt_race, aes(x = lwt)) +
    geom_histogram(binwidth = 10, boundary = 100) +
    facet_grid(. ~ race)
```



Exercise

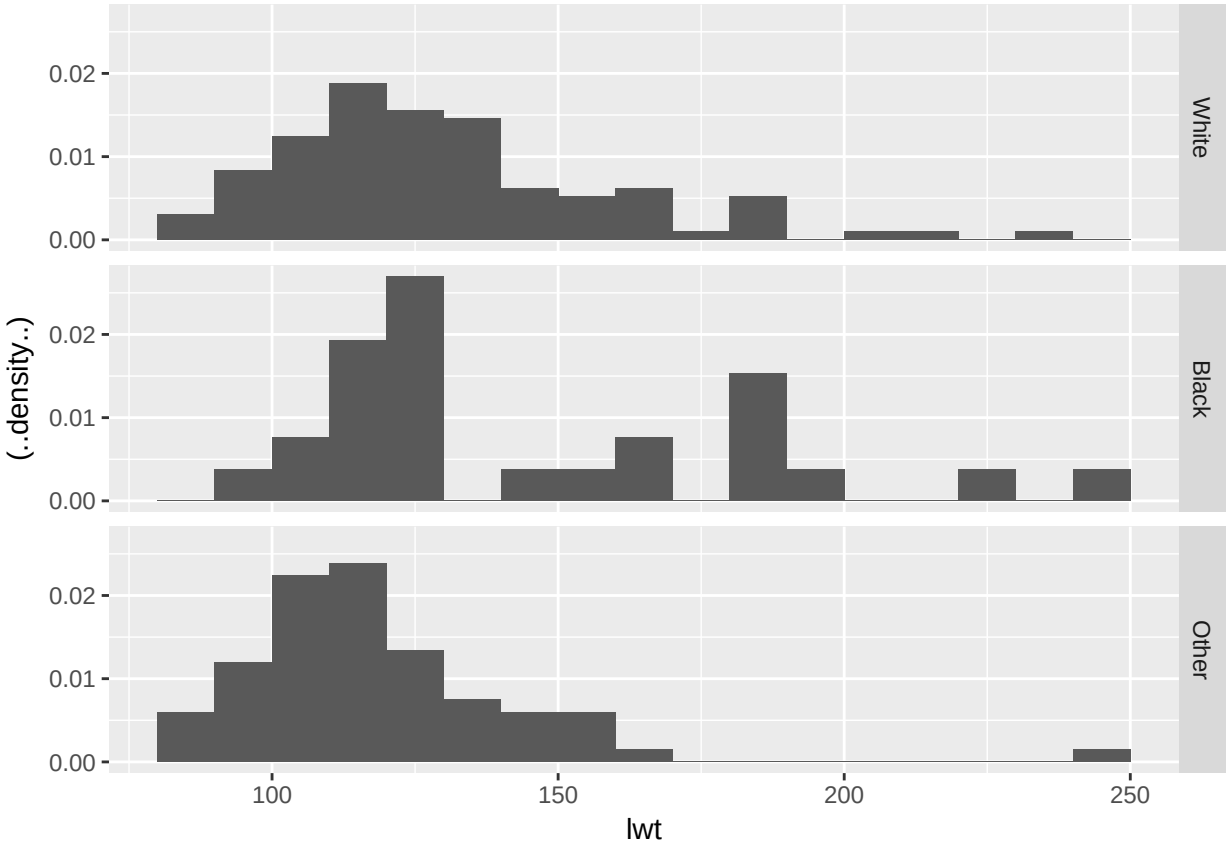
Explain why that last graph (which might be called a side-by-side histogram) is less effective than the earlier stacked histogram. (Hint: what stays lined up when the histograms are stacked vertically rather than horizontally?) Also, can you figure out what's going on with the weird syntax of `race ~ .` vs `. ~ race`?

ANSWER

Please write up your answer here.

The other thing that kind of sucks is the fact that the y-axis is showing counts. That makes it hard to see the distribution of weight among black women, for example, as there are fewer of them in the data set. It would be nice to scale these using percentages.

```
ggplot(lwt_race, aes(x = lwt)) +
    geom_histogram(aes(y = (..density..)), binwidth = 10, boundary = 100) +
    facet_grid(race ~ .)
```



Due to some technical issues in `ggplot2`, these are not strictly proportions. (If you were to add up the heights of all the bars, they would not add up to 100%.) Nevertheless, the graph is still useful because it does scale the groups to put them on equal footing. In other words, it treats each group as if they all had the same sample size.

Your turn

Choose an interesting numerical variable and an interesting categorical variable from the `birthwt` data set. (Choose at least one variable you haven't used already.) Convert the categorical variable to a factor variable. Create a data frame with your chosen variables. Then make both a side-by-side boxplot and a stacked histogram. Discuss the resulting graphs. Comment on the association (or independence) of the two variables.

ANSWER

```
# Add code here to convert a categorical variable to a factor variable
# and make a data frame.
```

```
# Add code here to create a side-by-side boxplot.
```

```
# Add code here to create a stacked histogram.
```

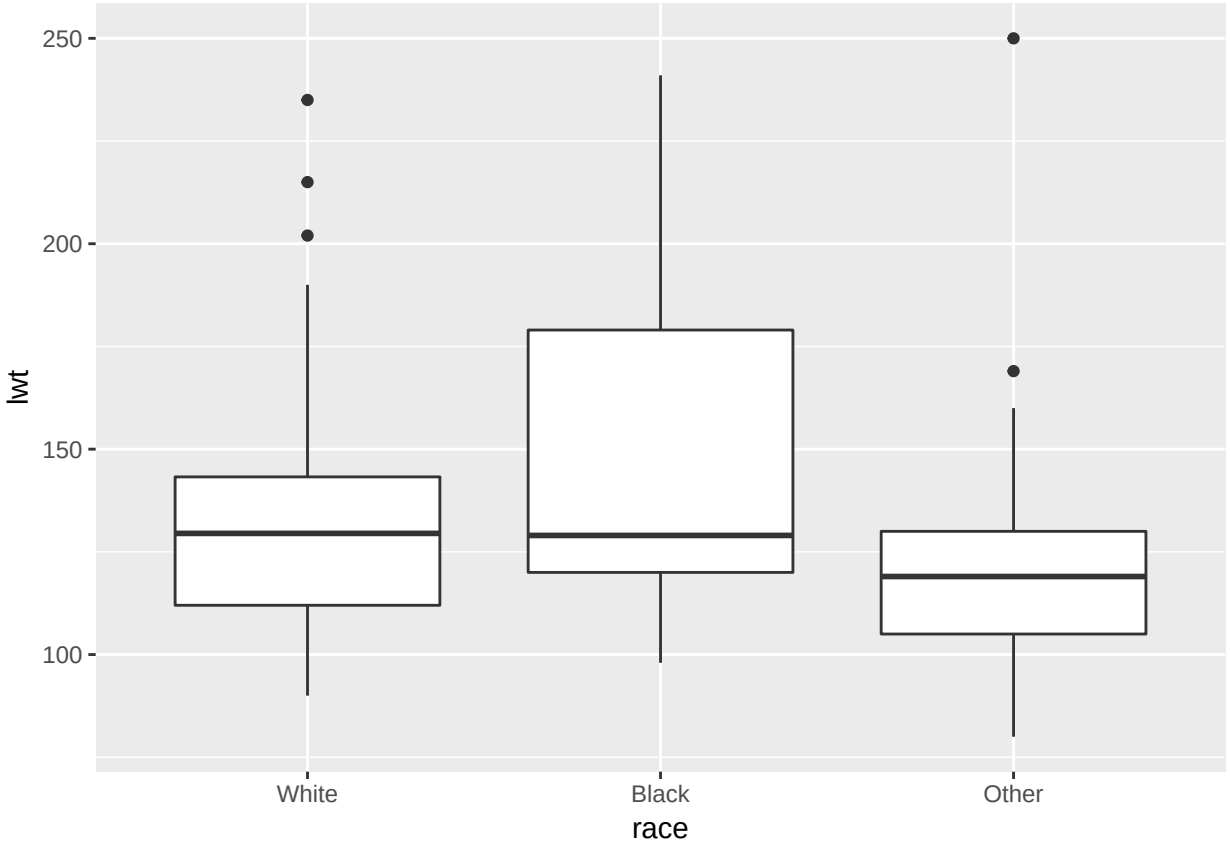
Please write up your answer here.

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.

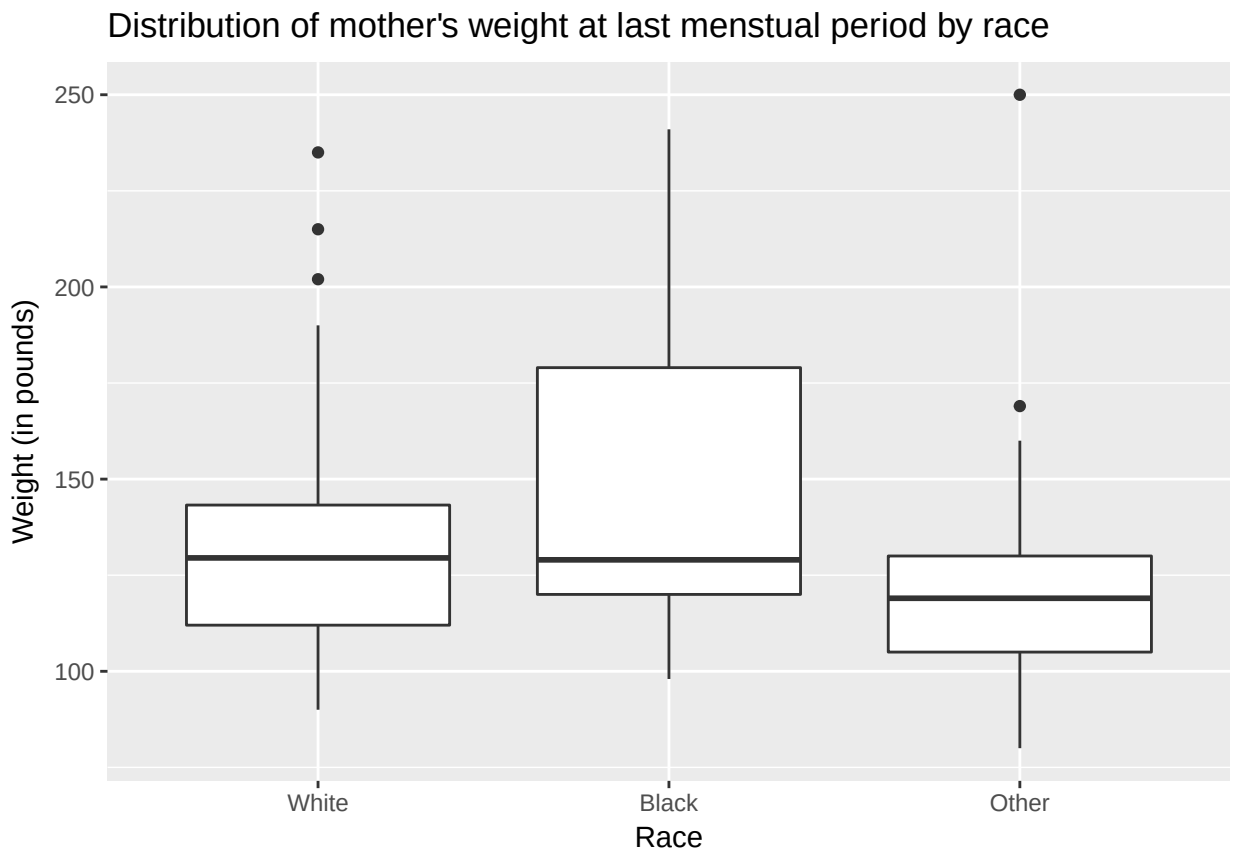
Let's go back to the side-by-side boxplot from earlier in the module.

```
ggplot(lwt_race, aes(y =lwt, x = race)) +
    geom_boxplot()
```



Note that the variable names of this data set are not terribly informative. In other words, if you were using this graph in a publication or presentation for an audience, they would have no idea what `lwt` was. Also note that this graph could use a title. We can do all this with `labs` (for labels). Observe:

```
ggplot(lwt_race, aes(y =lwt, x = race)) +
    geom_boxplot() +
    labs(title = "Distribution of mother's weight at last menstual period by race",
         y = "Weight (in pounds)",
         x = "Race")
```



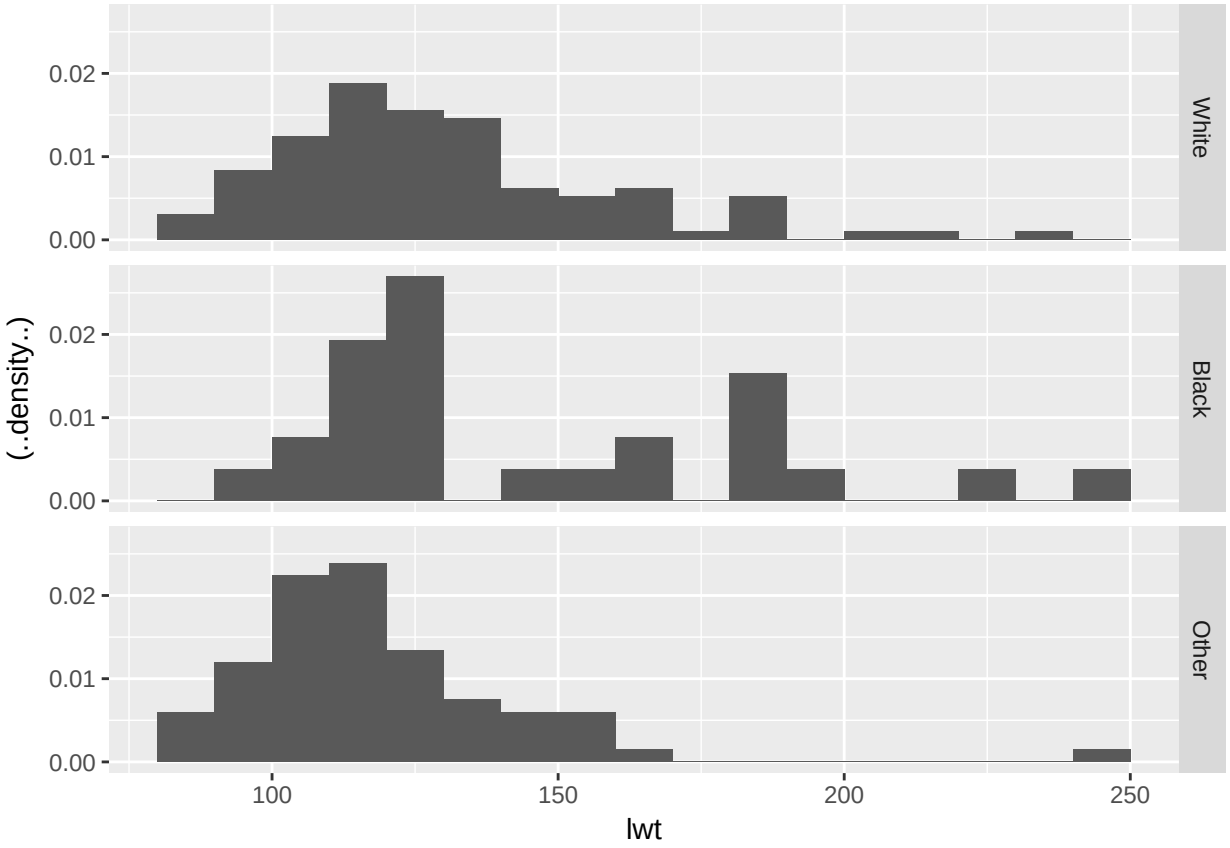
You can also see that we took the opportunity to mention the units of measurement (pounds) for our numerical variable in the y-axis label. This is good practice.

Exercise

Modify the following stacked histogram by adding a title and labels for both the y-axis and x-axis.

ANSWER

```
# Modify the following stacked histogram by adding a title and
# labels for both the y-axis and x-axis.
ggplot(lwt_race, aes(x = lwt)) +
    geom_histogram(aes(y = (..density..)), binwidth = 10, boundary = 100) +
    facet_grid(race ~ .)
```



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

When you want to analyze a numerical response variable using an explanatory categorical variable as a grouping variable, there are two good options: the side-by-side boxplot and the stacked histogram.