Visualizations II

Customizing plots in ggplot2

Joey Stanley
Doctoral Candidate in Linguistics, University of Georgia
joeystanley.com
orcid.org/0000-0002-9185-0048

Presented at the UGA Willson Center DigiLab
Friday, February 23, 2018

This is the sixth installment of the R workshop series in Spring 2018, and the second of three workshops that introduces how to make visualizations using the ggplot2 package. This document will cover additional topics in ggplot2 that let you customize plot in various ways: (1) adding and changing titles and axis labels; (2) custom colors; (3) renaming and reordering things; (4) legends; (5) faceting; (6) themes; and a custom section on saving plots.

Download this PDF from my website at
joeystanley.com/r2018

1 Introduction

In the last workshop, we looked at the basics of data visualization and data types and introduced the library `ggplot2`. Specifically, we looked at scatter plots and how we can plot shapes, color, size, and text. We looked at plotting one variable, whether it be categorical with a barplot or continuous with a histogram. Finally, we looked at boxplots and violin plots, and started to show how to overlay multiple plots in one image.

Today's workshop will focus less on the different kinds of plots and instead will show how you can modify things to suit your needs. Specifically, we'll take a closer look at modifying colors, reordering and renaming categorical variables, adding titles, modifying axes, legends, faceting, themes, and saving plots. It sounds like a lot, but it shouldn't be too bad. After today, you'll go from a the basic, default plots to something you might want to include in a presentaton or paper.

1.1 Today's dataset

Last week we looked at McDonald's menu items. Today, we'll look at another dataset made available through *Kaggle.com* that contain nutritional information of 80 different kinds of cereal. I've removed some of the columns to simplify the dataset and made it available on my website, so you can read it in straight from there.

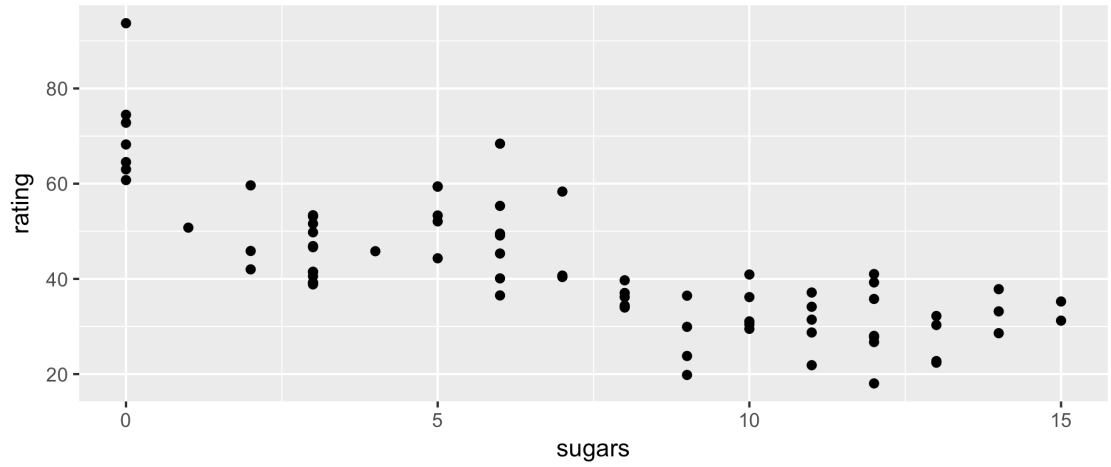
```
cereal <- read.csv("http://www.joeystanley.com/data/cereal.csv")
summary(cereal)
```

```
##                          name    mfr    type       shelf
##  100% Bran                  : 1   G:22   C:74   Min.   :1.000
##  100% Natural Bran          : 1   K:23   H: 1   1st Qu.:1.500
##  All-Bran                   : 1   N: 6          Median :2.000
##  All-Bran with Extra Fiber: 1   P: 9          Mean   :2.227
##  Almond Delight             : 1   Q: 7          3rd Qu.:3.000
##  Apple Cinnamon Cheerios    : 1   R: 8          Max.   :3.000
##  (Other)                    :69
##      weight          cups            rating         calories
##  Min.   :0.50   Min.   :0.2500   Min.   :18.04   Min.   : 50.0
##  1st Qu.:1.00   1st Qu.:0.6700   1st Qu.:32.69   1st Qu.:100.0
##  Median :1.00   Median :0.7500   Median :40.11   Median :110.0
##  Mean   :1.03   Mean   :0.8207   Mean   :42.39   Mean   :107.1
##  3rd Qu.:1.00   3rd Qu.:1.0000   3rd Qu.:50.28   3rd Qu.:110.0
##  Max.   :1.50   Max.   :1.5000   Max.   :93.70   Max.   :160.0
##
##      sugars         protein           fat          sodium
##  Min.   : 0.00   Min.   :1.000   Min.   :0.0   Min.   :  0.0
##  1st Qu.: 3.00   1st Qu.:2.000   1st Qu.:0.0   1st Qu.:135.0
```

```
##  Median : 7.00   Median :2.000   Median :1.0   Median :180.0
##  Mean   : 7.08   Mean   :2.493   Mean   :1.0   Mean   :163.9
##  3rd Qu.:11.00   3rd Qu.:3.000   3rd Qu.:1.5   3rd Qu.:215.0
##  Max.   :15.00   Max.   :6.000   Max.   :5.0   Max.   :320.0
##
##      fiber
##  Min.   : 0.000
##  1st Qu.: 1.000
##  Median : 2.000
##  Mean   : 2.173
##  3rd Qu.: 3.000
##  Max.   :14.000
##
```

As you can see we have a relatively simple dataset with a few categorical variables and mostly continuous variables. We'll use this for our plots today. To get us started, let's see if the amount of sugar correlates with the cereal's rating.

```
library(ggplot2)
ggplot(cereal, aes(sugars, rating)) +
    geom_point()
```



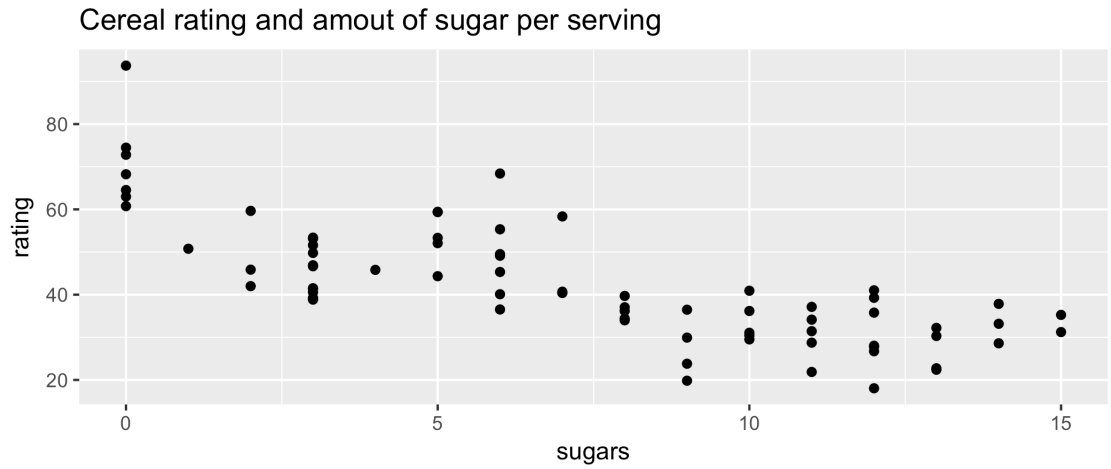
Surpringly, the general trend is that the more sugar a cereal has, the lower the rating is. Another way of looking at it is that cereals with zero sugar have the highest rating, between 1 and 7 grams per serving have a medium rating, and 8 or more has a low rating. Correlation is not causation, but the trend is interesting to see.

2 Titles and axes

The first thing we might want to do with a plot is to add a title or change the axis labels. We saw the code for these in our Shiny workshop a couple weeks ago and luckily, this is pretty straightforward.

For a title, all you need to do is add a layer to your plot using the `ggtitle` function, and then put the title you want to see in quotes.

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point() +
    ggtitle("Cereal rating and amout of sugar per serving")
```



We can also modify the axes of our plot by adding the `xlab` or `ylab` layers and, in quotes, put what you want the axes to be. Here are the kinds of changes I've had to do in the past that you might need to do as well:

1. When the columns of your data frame are something abbreviated, you can put the full name in a professional-looking plot. So, you can change `"mpg"` to `"miles per gallon"`).

2. Sometimes your column names are super long. An actual example I've seen is `"education_level_by_number_of_years"`, which could easily be shorted to `"eduacation (years)"`.

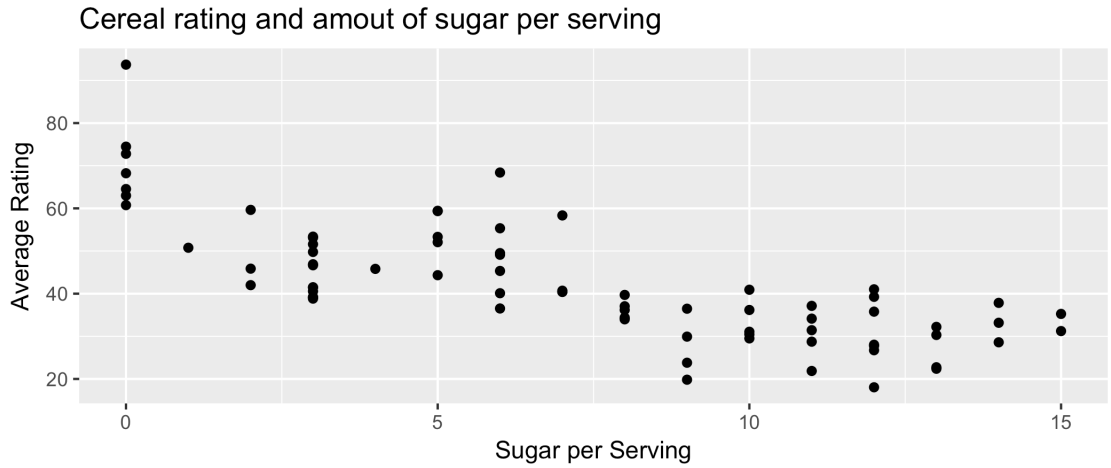
3. Often, all you'll need to do is change from lowercase to uppercase or vice versa (`"sugars"` to `"Sugars"`).

4. You may want to add a unit of measurement, so that `"height"` can be changed to `"height (in feet)"`

5. R doesn't like spaces in column names, but you'll probably want them in your plots, so you can change `"serving_size"` to `"serving size"`.

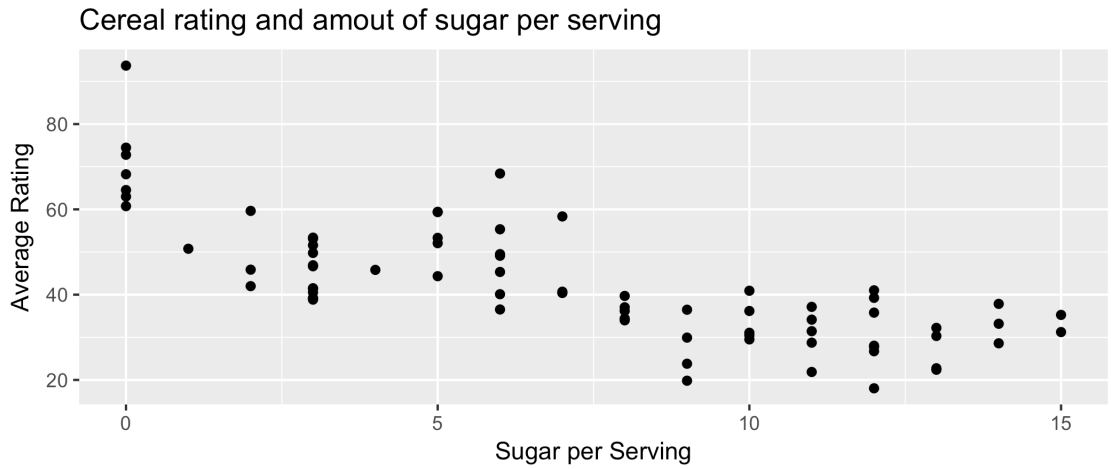
With ggplot2, you can make these changes so that they're reflected in your plot only without having to rename parts of your data frame.

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point() +
    ggtitle("Cereal rating and amout of sugar per serving") +
    xlab("Sugar per Serving") +
    ylab("Average Rating")
```



Now, what's interesting is that the above code is actually the shortcut functions. Instead, we can do the same thing by putting this same information inside of a function called `labs`, with `x`, `y`, and `title` being arguments to the function.

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point() +
    labs(x = "Sugar per Serving",
         y = "Average Rating",
         title = "Cereal rating and amout of sugar per serving")
```



The benefit of using this method is that we can also add subtitles and captions the same way.

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point() +
    labs(x = "Sugar per Serving",
         y = "Average Rating",
         title = "Cereal rating and amout of sugar per serving",
         subtitle = "In case the title wasn't long enough…",
         caption = "Figure 1: More sugar means a lower rating.")
```

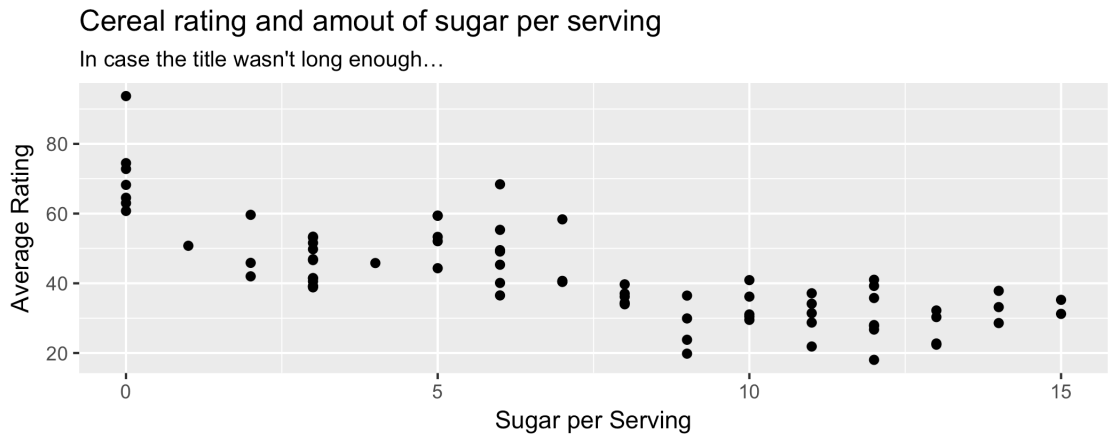


Figure 1: More sugar means a lower rating.

For now, we're stuck with the title and subtitle being left-aliged at the top and the caption being at the bottom right. Next week we'll learn how to modify these. For simplicity, we'll leave off the labels, but you can always go back and add it later on.

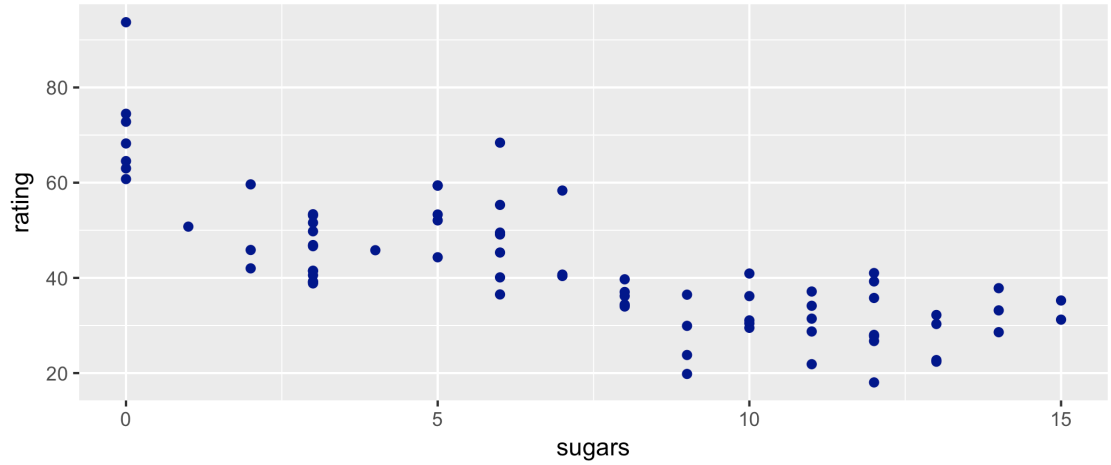
So now, let's see if we can start to see how we can modify other aspects of plots in ggplot2.

3 Colors

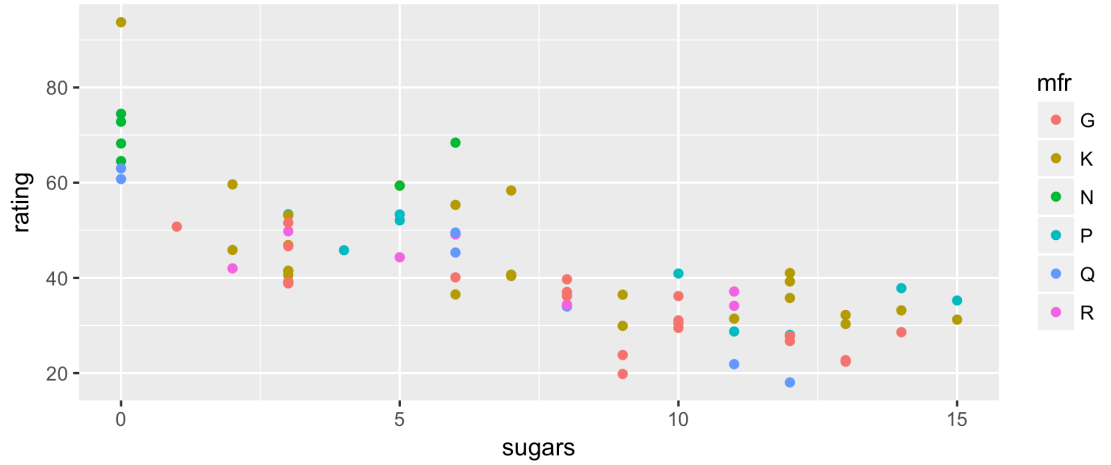
3.1 On categorical data

We'll start with colors, since this seems to be the most visible aspect of a plot. Last week, we saw how to change all the points by adding the `color` argument in `geom_point` itself, or we can add it within the `aes()` argument to have one color per manufacturer.

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point(color = "darkblue")
```



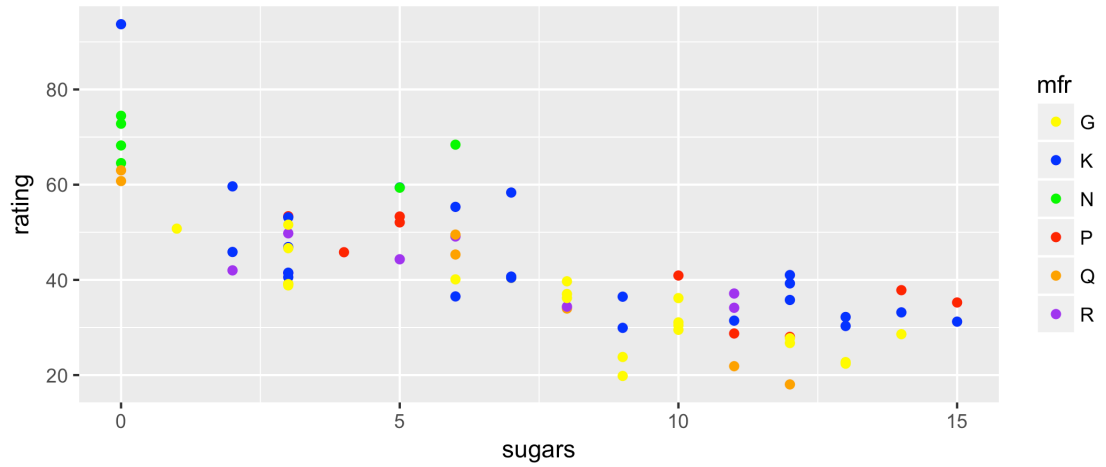

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point(aes(color = mfr))
```



Later in this workshop, we'll see how to modify the legend to make it clearer, but for now let's overlook the fact that it's not clear what the abbreviations stand for. We'll get to that in the next section.

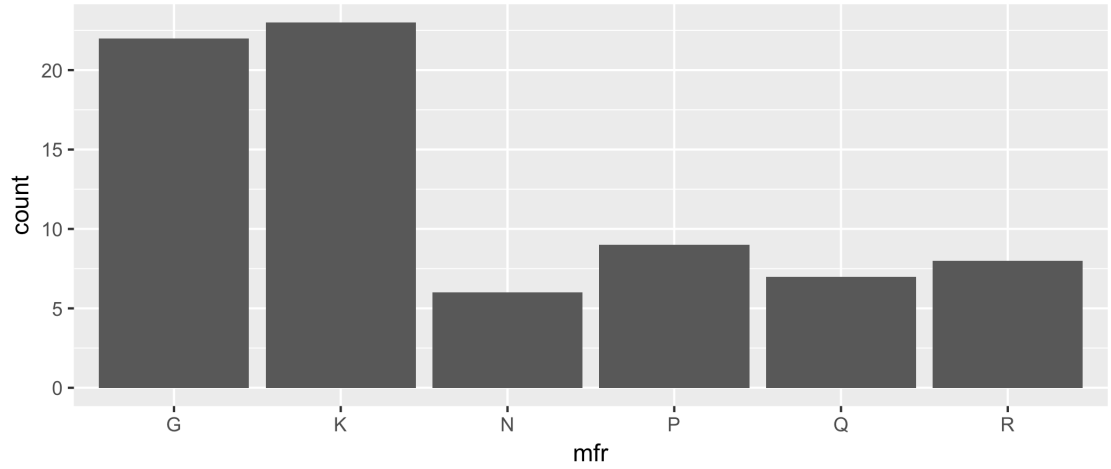
So these default colors are good for some purposes. They're evenly spaced around the color wheel so that they're maximally distinct. But for lots of reasons, we may want to change them. For example, we might not be satisified with the default color scheme and want to supply our own. We can do so with the function `scale_color_manual` and as an argument, provide a list of colors. Note that the order of the colors in your legend will be determined by the order you list them in `scale_color_manual`, so the first one is the top and the last one listed is the bottom. Here's an example of a not-so-good color scheme:

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point(aes(color = mfr)) +
    scale_color_manual(values = c("yellow", "blue", "green", "red", "orange",
"purple"))
```



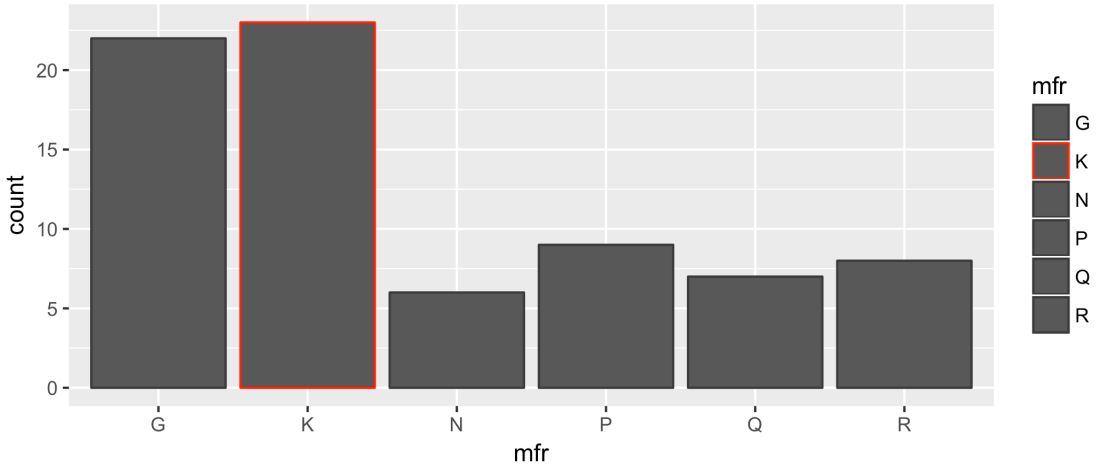
Being able to modify colors is useful not only to change all the colors, but it's also good for highlighting a single category. Let's switch to a barplot that lists the number of cereal items per manufacturer:

```
ggplot(cereal, aes(mfr)) +
    geom_bar()
```



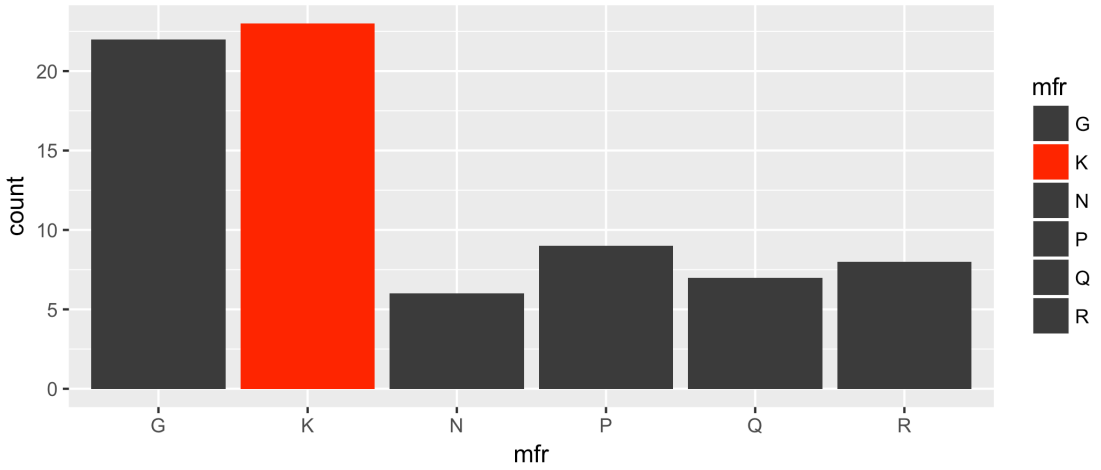
If you're familiar with cereal brands, you might deduce "K" stads for "Kellog's". We can highlight this by making them red and all the others a dark gray. The way this is done is by simply repeating the color names in the list in `scale_color_manual`:

```
ggplot(cereal, aes(mfr)) +
    geom_bar(aes(color = mfr)) +
    scale_color_manual(values = c("grey25", "red", "grey25", "grey25", "grey2
5", "grey25"))
```



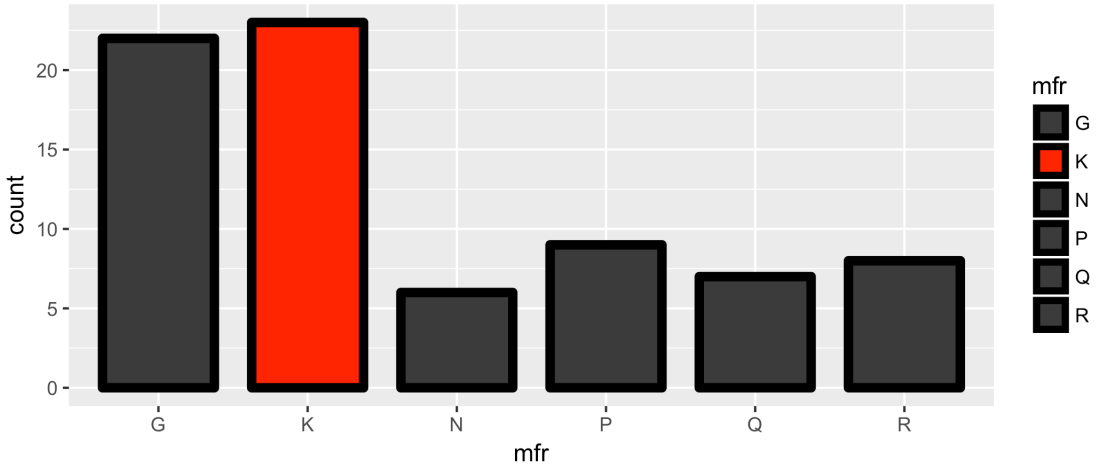
Oops! What happened here? Keep in mind that last week we learned that to color points in a scatterplot, you need to use `color` but for barplots you need to use `fill`. Consequently, we need to use the related function, `scale_fill_manual` to accomplish this task:

```
ggplot(cereal, aes(mfr)) +
    geom_bar(aes(fill = mfr)) +
    scale_fill_manual(values = c("grey25", "red", "grey25", "grey25", "grey25
", "grey25"))
```



Of course, if you don't like the way this looks, you can always modify the `color` argument still (as well as the size of the outline and width of the bars) and everything will behave as you expected.

```
ggplot(cereal, aes(mfr)) +
    geom_bar(aes(fill = mfr), color = "black", size = 2, width = 0.75) +
    scale_fill_manual(values = c("grey25", "red", "grey25", "grey25", "grey25
", "grey25"))
```



3.1.1 Now you try!

3.1.1.1 The challenge

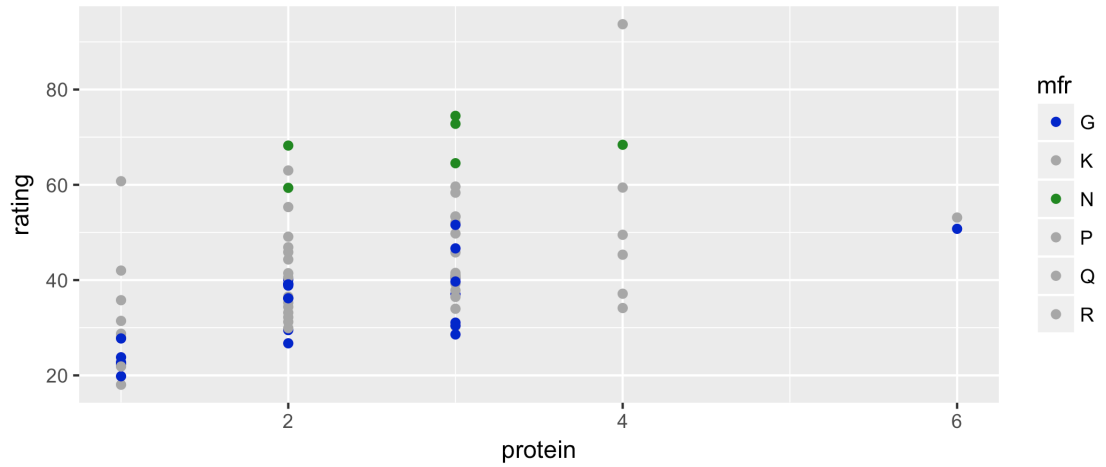
1. Try putting in different variables for the scatterplot and see what kinds of things you discover about cereal. When you find something interesting, perhaps about a single company, highlight just the dots for that manufacturer. Feel free to modify other aspects of the plot as you see fit.

2. Just as you can manually change the colors using `scale_color_manual`, you can also change the size of the points using `scale_size_manual`. In fact, you can change any of the aspects we've seen so far using `scale_*_manual` where the `*` is the part you're trying to change. To use these, you have to use those properties in the `aes()` function because you're overriding the defaults. In other words, if you want to highlight a particular manufacturer's size dot, you have to have `size = mfr` in `geom_point`. Try making a barplot that uses some of these additional overriding functions.

3.1.1.2 The solution

Here's something I found interesting. There's a slight trend such that the more protein a cereal has, the higher its rating is. But it seems like Nestle cereals are at the top of the pack while General Mills are lower.

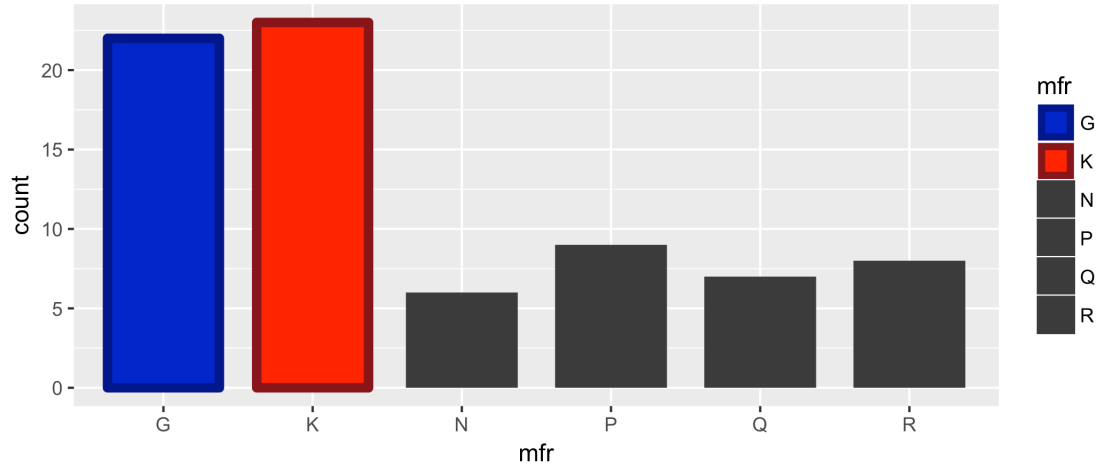
```
ggplot(cereal, aes(protein, rating)) +
    geom_point(aes(color = mfr)) +
    scale_color_manual(values = c("blue3", "darkgrey", "forestgreen",
                                  "darkgrey", "darkgrey", "darkgrey"))
```



Of course, this may just be that Nestle cereals have higher ratings than General Mills across the board. You may have to do some more plots to see for sure.

For the barplot, I've went ahead and modifed the fill, outlines color, and size of the bar corresponding to Kellog's and I added General Mills to for fun. I wasn't able to modify the width manually, so I kept that the same. Also, I set outline width to zero to remove it entirely on the other four

```
ggplot(cereal, aes(mfr)) +
    geom_bar(aes(fill = mfr, color = mfr, size = mfr), width = 0.75) +
    scale_fill_manual(values = c("blue3", "red", "grey25",
                                 "grey25", "grey25", "grey25")) +
    scale_color_manual(values = c("darkblue", "firebrick4", "black",
                                  "black", "black", "black")) +
    scale_size_manual(values = c(2, 2, 0, 0, 0, 0))
```

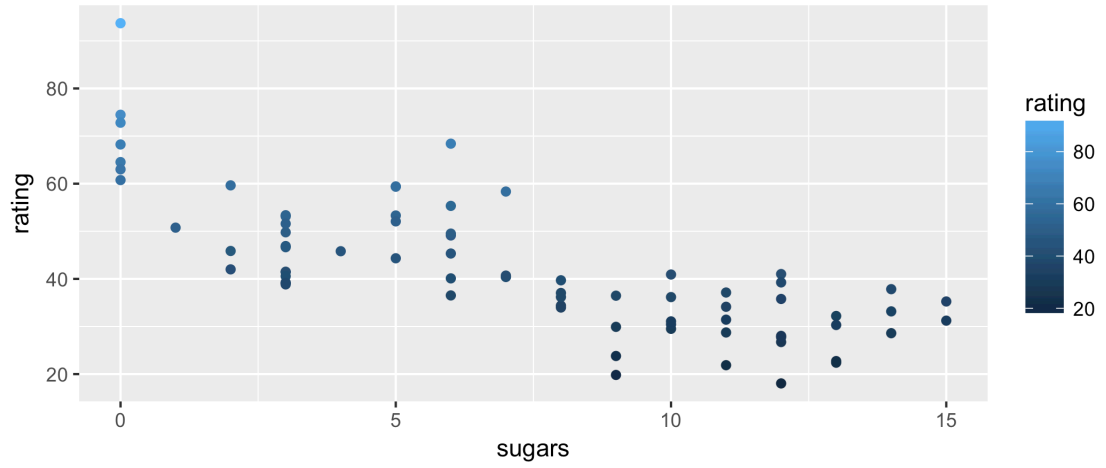


Using these `scale_*_manual` functions, it becomes easy to change many aspects of your plot. You can do this to change color schemes, or highlight particular points.

3.2 On continuous data

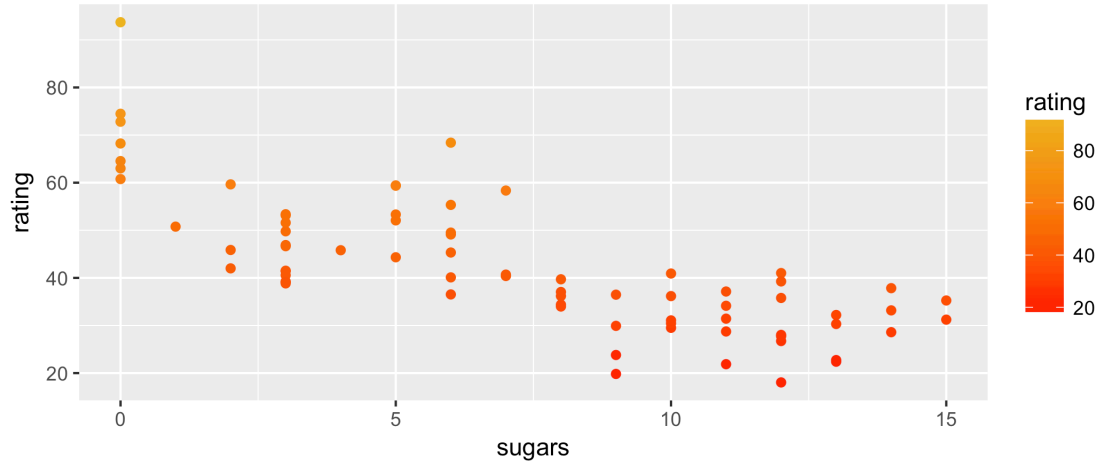
What we've seen so far is how to change colors for categorical variables. Sometimes we have continuous data, but we don't like the default black to blue:

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point(aes(color = rating))
```



The way we change this is similar to how we changed the categorical variables, but we use `scale_color_gradient` instead. Here, we can specify what the color for the lowest and highest values should be. R will automatically create a nice color gradience for you.

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point(aes(color = rating)) +
    scale_color_gradient(low = "red", high = "goldenrod2")
```



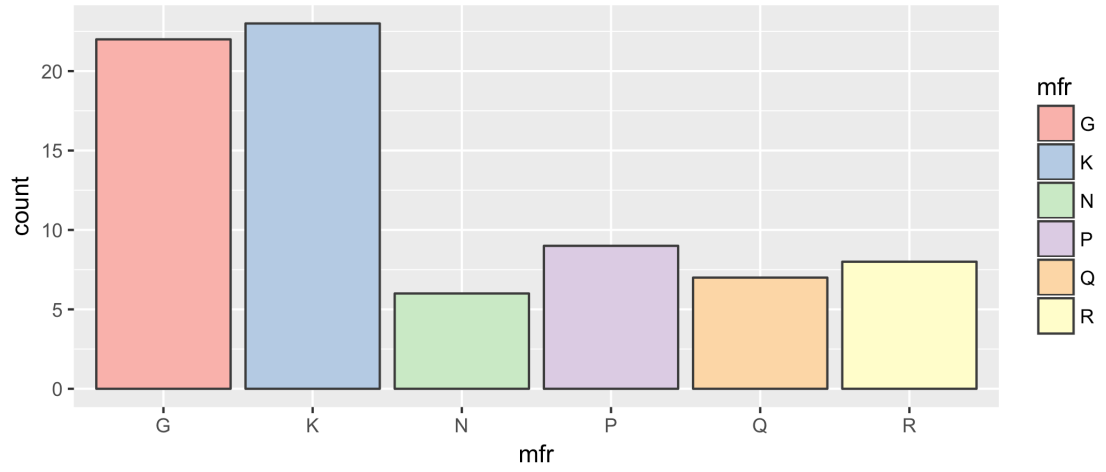
So it's easy to change color schemes if you don't like the default black to blue. I sometimes have a hard time choosing good colors (like try `"red"` to `"blue"`).

3.3 Color Brewer

When I create my own color schemes, the colors I try are usually too harsh. Fortunately, some smart people, particularly those who make maps, have developed color schemes that are easy on the eyes. Some of them are good for printing black-and-white and are color-blind friendly as well. You can see these schemes by going to *colorbrewer2.org*.

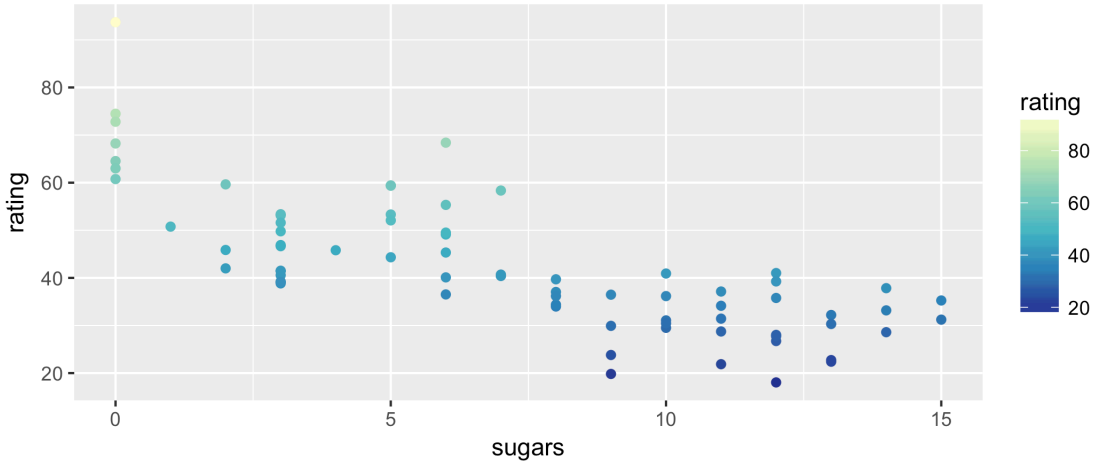
Luckily for us, ggplot2 has a built in function to work with these themes, `scale_color_brewer` and `scale_fill_brewer`. If you're working with categorical variables, you'll need to specify `type = "qual"` (for *qualitative*). You can use the default palette (`"Accent"`), or you try some of the other ones (`"Dark2"`, `"Pastel1"`, `"Pastel2"`, `"Set1"`, `"Set2"`, or `"Set3"`). These colors tend to be a bit lighter, so they'll pop out better when we learn how to make the background white instead of grey later in this workshop.

```
ggplot(cereal, aes(mfr)) +
    geom_bar(aes(fill = mfr), color = "grey25") +
    scale_fill_brewer(type = "qual", palette = "Pastel1")
```



For continuous variables, the process is similar, but you need to use `scale_color_distiller` or `scale_fill_distiller` instead. You'll need to specify `type = "seq"` (for *sequential*) and use one of the many types they have available. The mutli-hue options are `"BuGn"`, `"BuPu"`, `"GnBu"`, `"OrRd"`, `"PuBu"`, `"PuBuGn"`, `"PuRd"`, `"RdPu"`, `"YlGn"`, `"YlGnBu"`, `"YlOrBr"`, and `"YlOrRd"` and the single-hue options are `"Blues"`, `"Greens"`, `"Greys"`, `"Oranges"`, `"Purples"`, and `"Reds"`.

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point(aes(color = rating)) +
    scale_color_distiller(type = "seq", palette = "YlGnBu")
```



The downside to using the Scale Brewer colors is that it's a bit harder to modify them manually once you use them. It's possible, but it'll probably involve going to their website and copy and pasting the hexidecimal color codes and passing them into `scale_color_manual`.

3.3.1 Your turn!

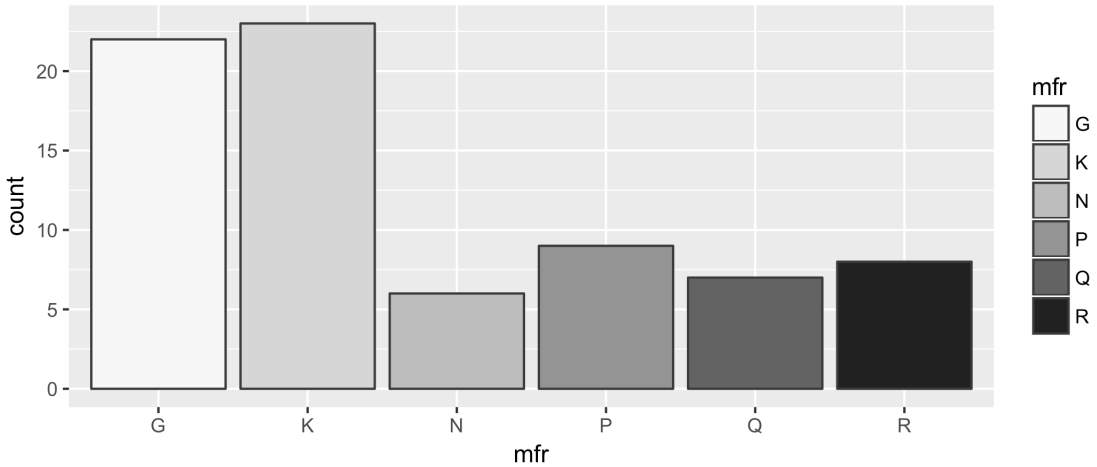
3.3.1.1 The challenge

1. Try using a categorical color scheme in `scale_color_distiller` and a continuous theme in `scale_color_brewer` and see what happens.

3.3.1.2 The solution

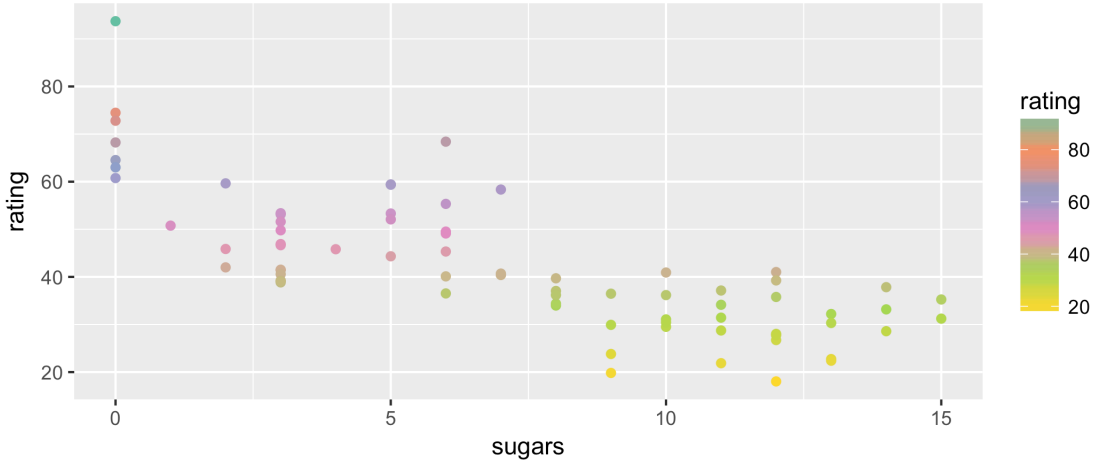
Fortunately, the folks at Color Brewer have made it so the continuous color schemes look great, even on categorical data. With six categorical variables, it's spread a little thin within the color scheme, but they're still distinct.

```
ggplot(cereal, aes(mfr)) +
    geom_bar(aes(fill = mfr), color = "grey25") +
    scale_fill_brewer(type = "qual", palette = "Greys")
```



Going the other way, ggplot2 is smart and will take the discrete colors of a categorical theme and fill in the gaps to create a continous color scheme.

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point(aes(color = rating)) +
    scale_color_distiller(type = "seq", palette = "Set2")
```



This is what they do on weather maps by the way. As it goes from cold to hot, the colors go from like a white to blue to green to red. Here it's a bit unneccesary, but it might be useful for you and your data.

3.4 Final remarks on color

So that's how you can modify colors in your plots. Colors are one of the most important parts of your plot, other than the data itself. A good use of color schemes can really make a plot look great so it's worth the time to learn to use them well.

4 Renaming and reordering

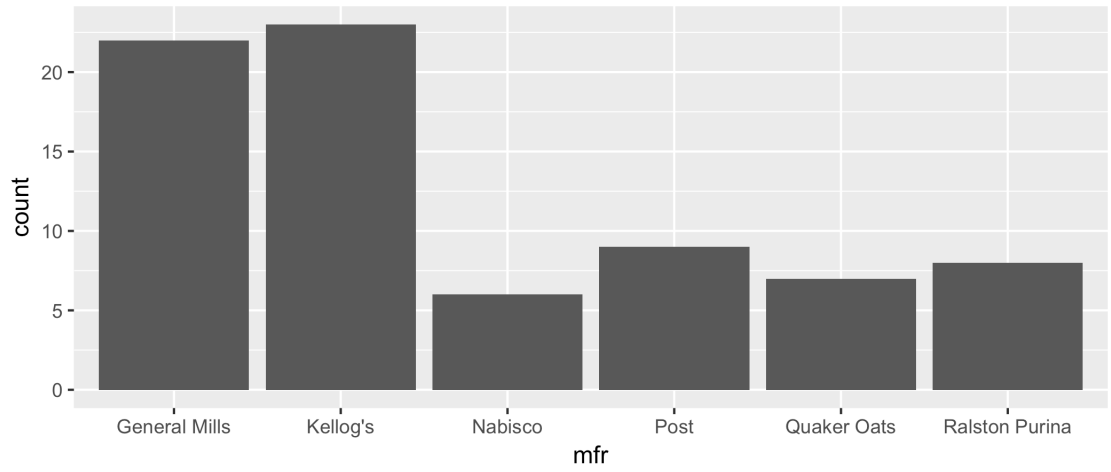
Reordering things is a pretty simple process, as we'll see in this section, but renaming them is a bit tricker. The way to do this is actually not in ggplot2 but to modify the dataframe itself and then feed this new data frame into `ggplot`. There are two ways to do this: the hard way using base R functions or the easy way using `tidyverse` functions. For reference, here's how you do it in base R.

```
# First make a copy
cereal_renamed <- cereal

# Copy and paste this complicated code one-by-one
levels(cereal_renamed$mfr)[levels(cereal_renamed$mfr)=="K"] <- "Kellog's"
```

```
levels(cereal_renamed$mfr)[levels(cereal_renamed$mfr)=="G"] <- "General Mills
"
levels(cereal_renamed$mfr)[levels(cereal_renamed$mfr)=="P"] <- "Post"
levels(cereal_renamed$mfr)[levels(cereal_renamed$mfr)=="Q"] <- "Quaker Oats"
levels(cereal_renamed$mfr)[levels(cereal_renamed$mfr)=="R"] <- "Ralston Purin
a" # bought by General Mills
levels(cereal_renamed$mfr)[levels(cereal_renamed$mfr)=="N"] <- "Nabisco"
# bought by Post

# Plot it
ggplot(cereal_renamed, aes(mfr)) +
    geom_bar()
```



So it works! the problem is there's a lot of code there to do what should be a pretty simple task. You'll see this a lot more later, but `tidyverse` functions provide a nice way to make these kinds of things easier. If you haven't installed it already, you'll need to load the `forcats` library, which you should have already if you've downloaded `tidyverse`.

```
install.packages("forcats") # If you haven't already
```

With that loaded, we can use `fct_recode` to change them all. Note that in the base R code, the old name is on the left and the new name is on the right. Here it's the opposite.

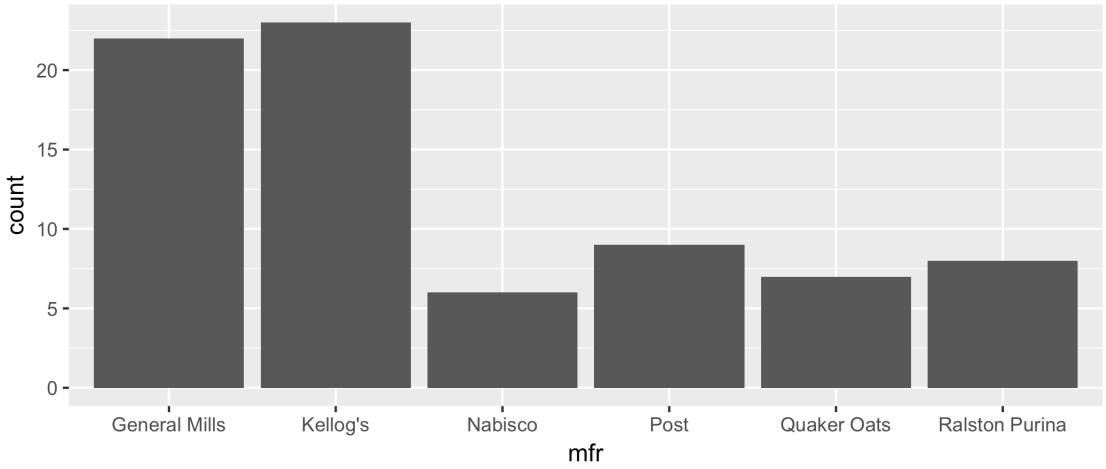
```
# Load the package
library(forcats)

# Make a copy again
cereal_renamed <- cereal

# Modify them like this.
cereal_renamed$mfr <- fct_recode(cereal_renamed$mfr,
                                 "Kellog's" = "K",
                                 "General Mills" = "G",
                                 "Post" = "P",
                                 "Quaker Oats" = "Q",
```

```
                                "Ralston Purina" = "R",
                                "Nabisco" = "N")

# Plot it
ggplot(cereal_renamed, aes(mfr)) +
    geom_bar()
```



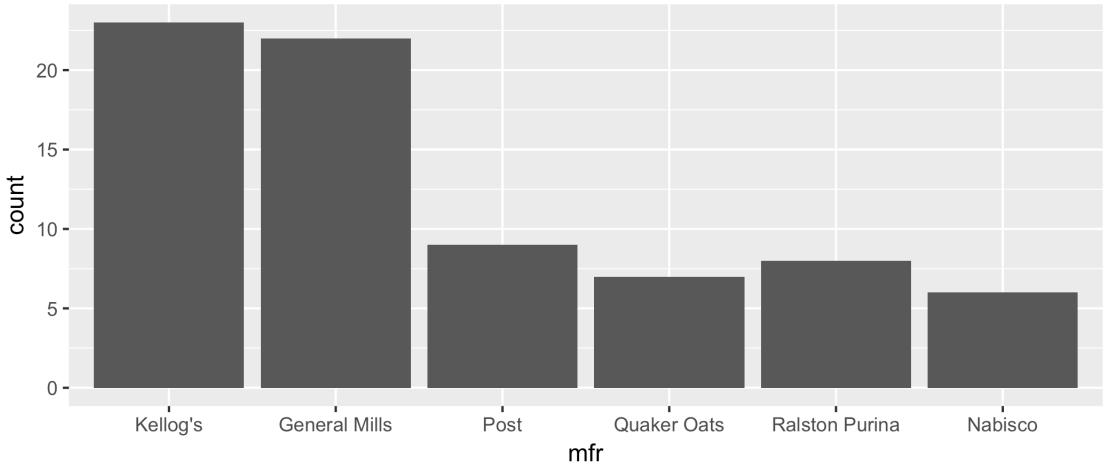
The `fct_recode` function does this a lot better if you ask me. Either way, the point is that now we have different names in the plot, which ultimately makes it much easier to read an interpret.

What you may want to do at this point is change the order. By default, R will order your categorical variables alphabetically. This is a good default, but you might want to change that in your own plot. Let's start with our barplot we had before.

One way to reorder these is to put them in order of frequency. There is a way to do this automatically, but that'll have to wait until we get to the Tidyverse workshops because it's a bit more involved than `fct_recode`. Fortunately, we can just do it by hand, which doesn't take too much work.

Like renaming, we actually take care of this by modifying the dataframe itself and then sending it off to `ggplot`. We modify the `mfr` column so that it's a factor with levels that you specify. The order of this list is the order they'll appear in your plot. Do keep in mind that you'll have to use the new names if you've changed them.

```
cereal_ordered <- cereal_renamed
cereal_ordered$mfr <- factor(cereal_ordered$mfr,
                             levels = c("Kellog's", "General Mills", "Post",
                                        "Quaker Oats", "Ralston Purina", "Nab
isco"))
ggplot(cereal_ordered, aes(mfr)) +
    geom_bar()
```



That's really all there is to it. If you don't like the order of things, it's pretty easy to change it once you've got the code.

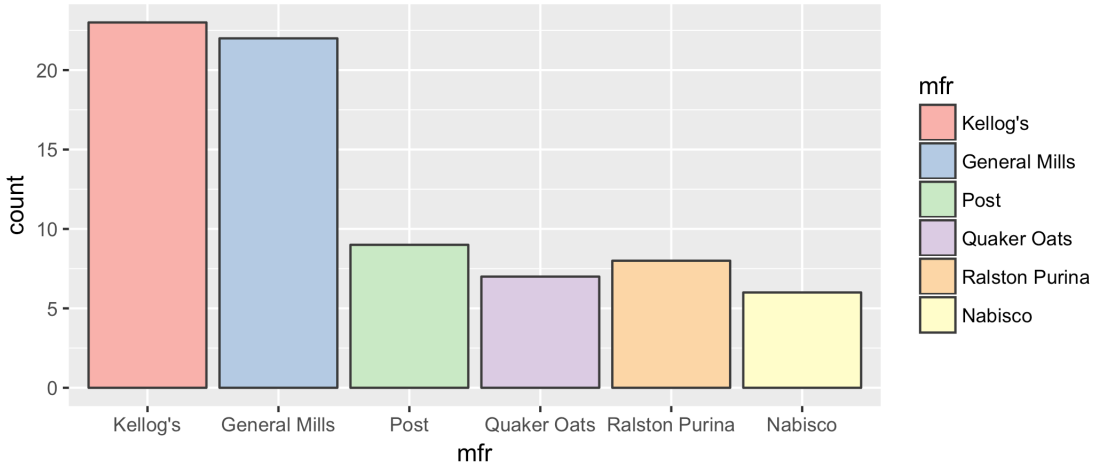
5 Legends

A critical part of datavisualization is the legend. Sometimes the legend is completely uncessary but other times it's the key to understanding the plot. We'll look at some examples of each and see how we can improve the visualization.

5.1 Removing the legend

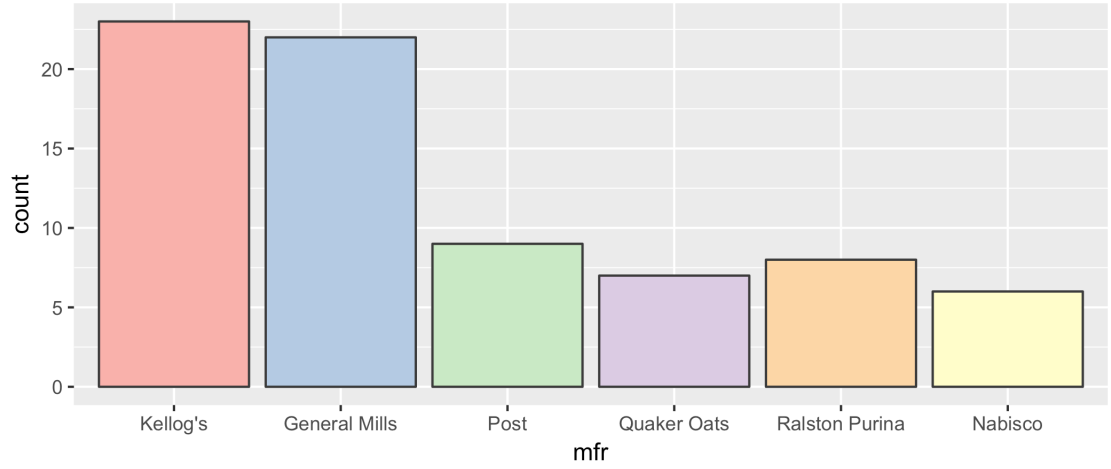
Let's take a bar plot of the manufacturers and color it using Color Brewer.

```
ggplot(cereal_ordered, aes(mfr)) +
    geom_bar(aes(fill = mfr), color = "grey25") +
    scale_fill_brewer(type = "qual", palette = "Pastel1")
```



This is nice example where the legend is not necessary. All the information from the legend itself is already contained in the plot. Fortunately, it's easy to remove the legend using the `theme` function. This function is actually a monster with dozens and dozens of arguments because it's the key to modifying pretty much everything on the plot, but we'll get to that next week. For now, if we set `legend.position` argument to `"none"` it'll remove it.

```
ggplot(cereal_ordered, aes(mfr)) +
    geom_bar(aes(fill = mfr), color = "grey25") +
    scale_fill_brewer(type = "qual", palette = "Pastel1") +
    theme(legend.position = "none")
```

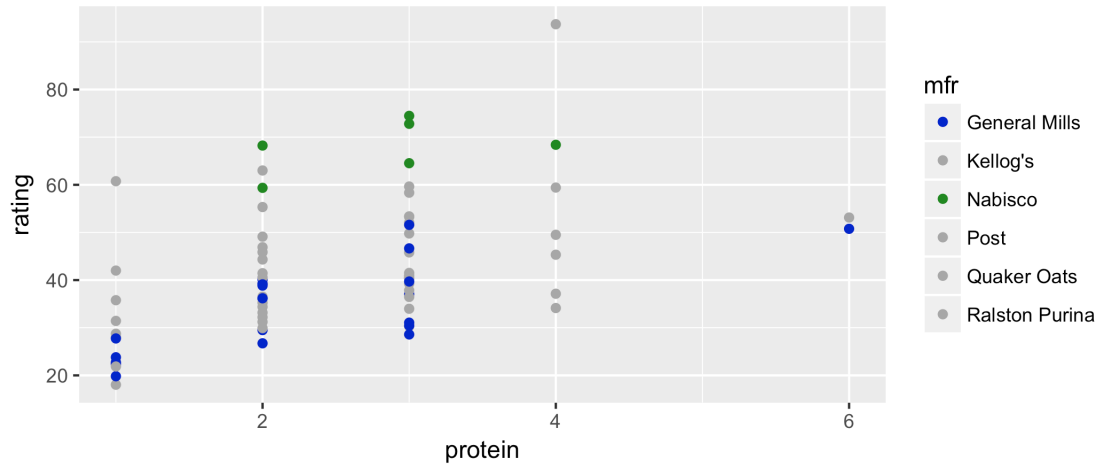


This not only removes the legend, but it actually stretches out the plot a little bit to fill the whole space. Side note, instead of using `"none"`, you can also change the legend's location by using `"top"`, `"bottom"`, or `"left"`. But for this plot, it's probably best to just remove it entirely. In fact, we can probably remove the colors since they don't actually serve a purpose, but that's up to you.

5.2 Modifying the order

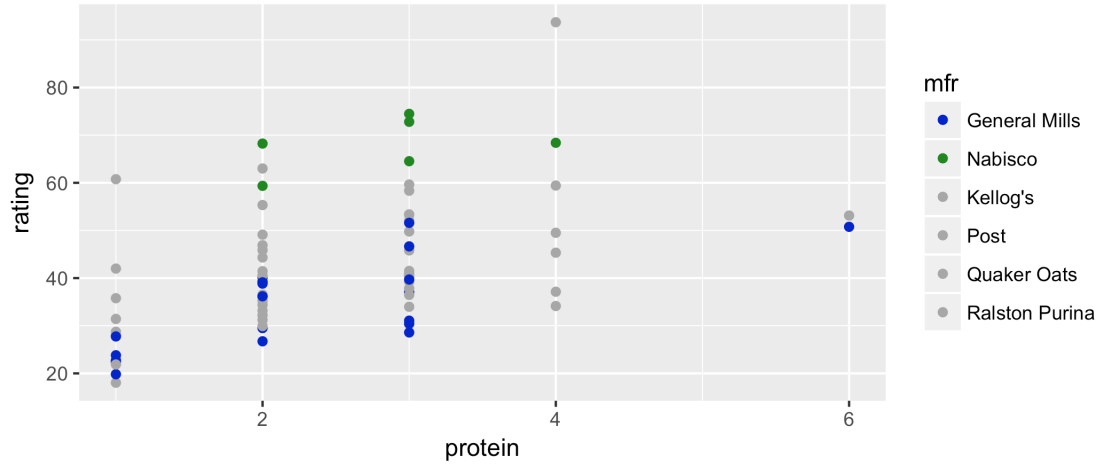
Previously, we saw how to modify the order that the manufacurers appeared in the barplot. The legend automatically updates. We can actually modify the legend independently of the bars. Let's take a barplot we did earlier that emphasized Kellog's. This time we'll use the `cereal_renamed` object so we can get the actual names.

```
ggplot(cereal_renamed, aes(protein, rating)) +
    geom_point(aes(color = mfr)) +
    scale_color_manual(values = c("blue3", "darkgrey", "forestgreen",
                                  "darkgrey", "darkgrey", "darkgrey"))
```



Here's a good example of when we might want to reorder the legend. Of course we can do it using the technique in the Renaming and reordering section above. (We also need to modify the order of the colors in `scale_color_manual` since now the forest green one is second.)

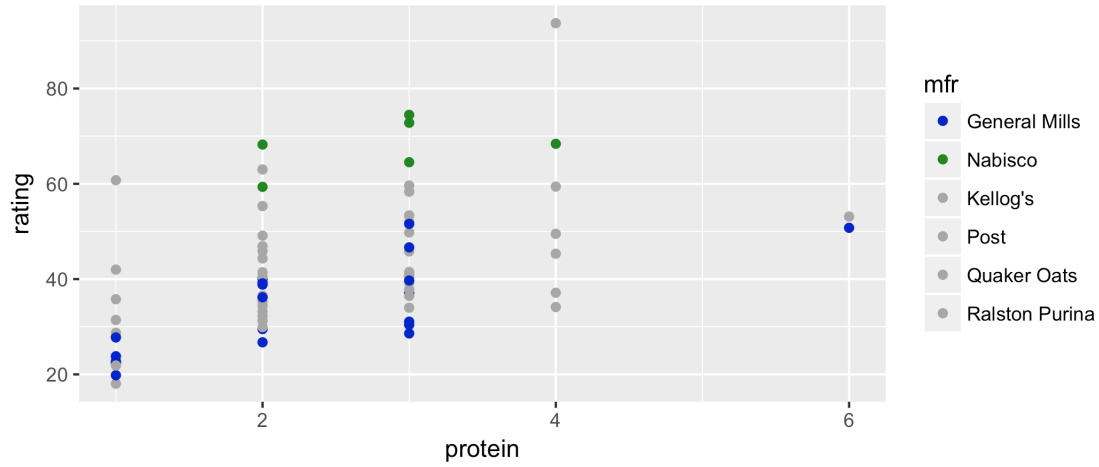
```
cereal_ordered2 <- cereal_ordered
cereal_ordered2$mfr <- factor(cereal_ordered2$mfr,
                              levels = c("General Mills", "Nabisco", "Kellog's
", "Post",
                                         "Quaker Oats", "Ralston Purina"))
ggplot(cereal_ordered2, aes(protein, rating)) +
    geom_point(aes(color = mfr)) +
    scale_color_manual(values = c("blue3", "forestgreen", "darkgrey",
                                  "darkgrey", "darkgrey", "darkgrey"))
```



The problem with this is we either have to 1) modify the original dataset (`cereal`), which is sometimes something you don't want to do, or 2) keep track of several very similar objects (`cereal_renamed`, `cereal_ordered` and now `cereal_reordered2`). Instad, we can add another argumnet to `scale_color_manual`, the `breaks` argument and simply list the order we want to see. The tricky part about this is that the order of colors in the `values` list has to match the

original order. The reason for this is that the `breaks` makes a superficial change, but the order stays the same under the hood.

```
ggplot(cereal_renamed, aes(protein, rating)) +
    geom_point(aes(color = mfr)) +
    scale_color_manual(breaks = c("General Mills", "Nabisco", "Kellog's", "Po
st",
                                  "Quaker Oats", "Ralston Purina"),
                       values = c("blue3", "darkgrey", "forestgreen",
                                  "darkgrey", "darkgrey", "darkgrey"))
```



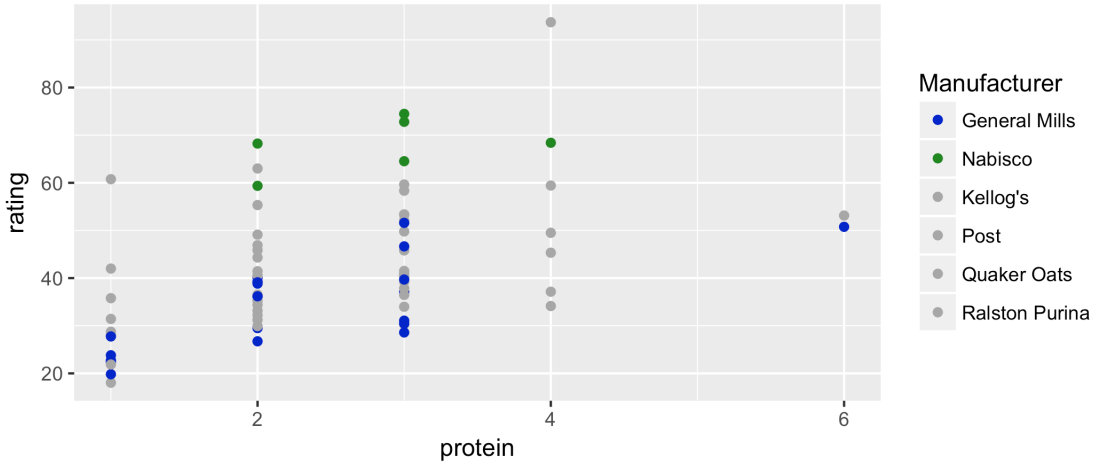
This is admittedly a bit annoying, but you only have to worry about it once and then it'll always be the same after that in your plots.

5.3 Modifying the legend text

One super annoying part of about the legend as it is now is that we see the abbreviation "mfr" as the title. That's the name of the column in our spreadsheet, and it's handy to have it short because, well, less typing. But in a professional visualization, you'll probably want to spell it out.

You can modify the original data, but it might just be easier to make a superficial change. This can happen in the `scale_color_manual` function again, with the argument `name`.

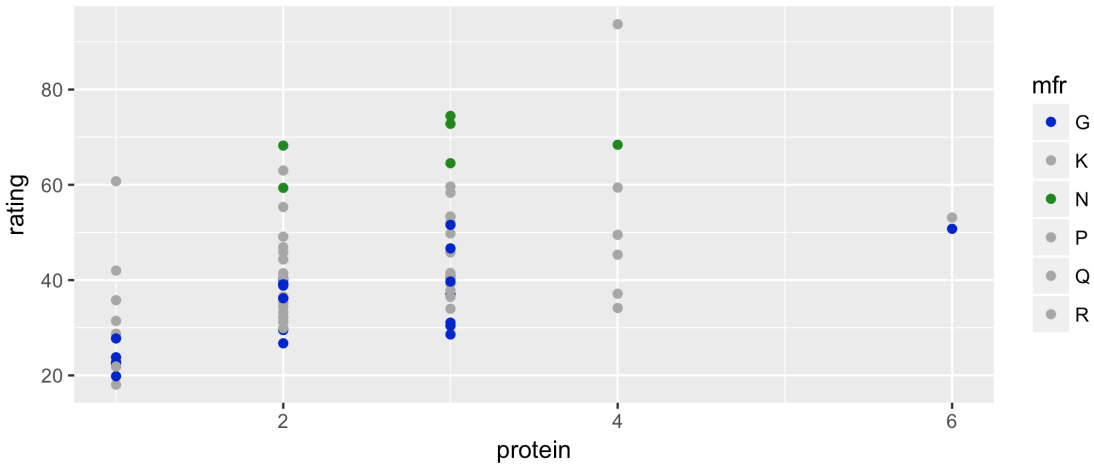
```
ggplot(cereal_renamed, aes(protein, rating)) +
    geom_point(aes(color = mfr)) +
    scale_color_manual(name = "Manufacturer",
                       breaks = c("General Mills", "Nabisco", "Kellog's", "Po
st",
                                  "Quaker Oats", "Ralston Purina"),
                       values = c("blue3", "darkgrey", "forestgreen",
                                  "darkgrey", "darkgrey", "darkgrey"))
```



So now we have three arguments in `scale_color_manual`: `name`, `breaks`, and `values`. The order that they appear makes no difference. But I like to put the `name` at the top since it appears at the top in the final plot.

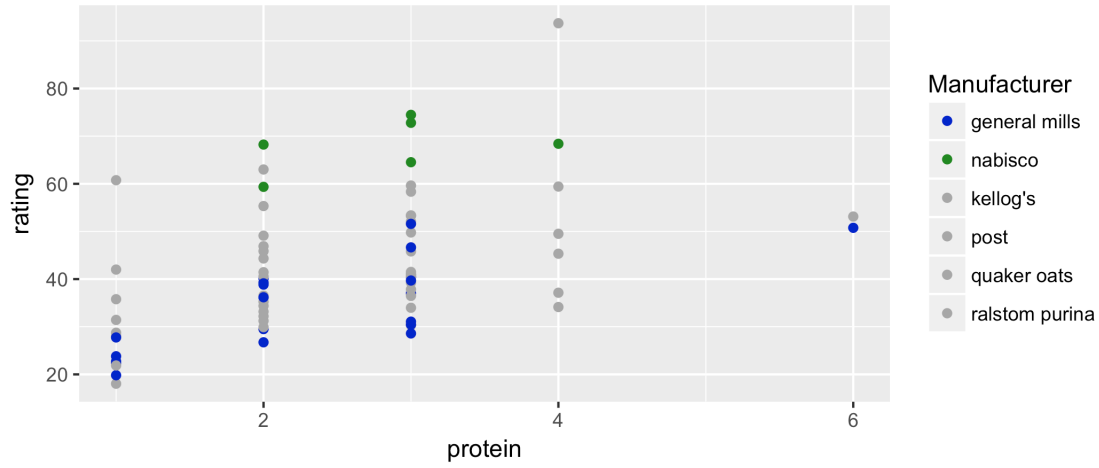
We can also modify the legend labels. We saw how to change the underlying data above in the Renaming and reordering section. But just as you sometimes want to make an order change just for the purpose of a single plot, you can rename things superficially while keeping the underlying data intact. To illustrate this, I'll use the original `cereal` dataset, which has the one-letter abbreviations for the manufacturers, and I'll change them in the plot (and I'll make them lowercase to show that they're different). To accomplish this, I add the `labels` argument to the other three.

```
# Plot with no changes to the legend (other than color)
ggplot(cereal, aes(protein, rating)) +
    geom_point(aes(color = mfr)) +
    scale_color_manual(values = c("blue3", "darkgrey", "forestgreen",
                                  "darkgrey", "darkgrey", "darkgrey"))
```




```
# Plot with all changes in title, order, labels, and colors
ggplot(cereal, aes(protein, rating)) +
```

```
    geom_point(aes(color = mfr)) +
    scale_color_manual(name = "Manufacturer",
                       breaks = c("G", "N", "K", "P", "Q", "R"),
                       labels = c("general mills", "nabisco", "kellog's",
                                  "post", "quaker oats", "ralstom purina"),
                       values = c("blue3", "darkgrey", "forestgreen",
                                  "darkgrey", "darkgrey", "darkgrey"))
```



Thus, with a little bit of typing, you can modify whatever you want with the legend.

5.3.1 Your turn!

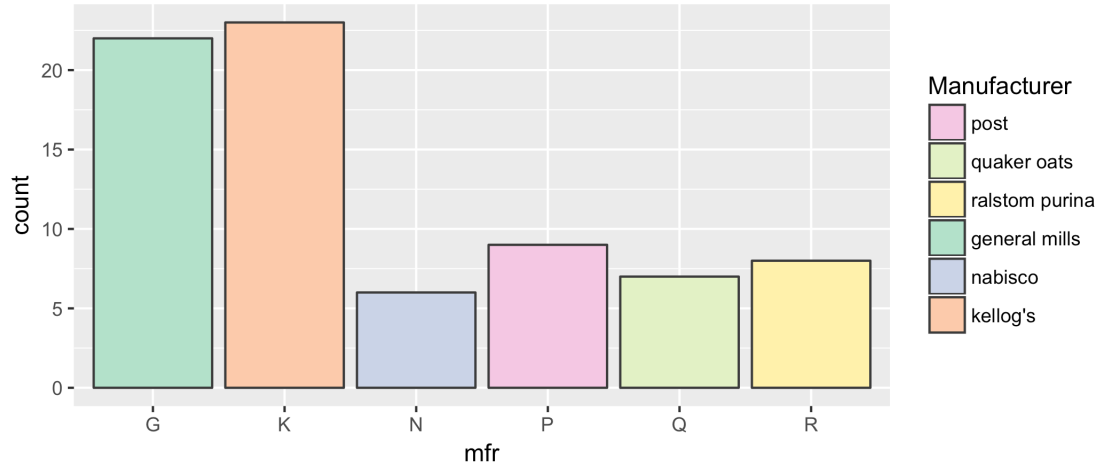
5.3.1.1 The challenge

1. Everything about manually changing things in the legend in `scale_color_manual` also applies to `scale_fill_manual` and `scale_fill_brewer`. Try making barplot with the original `cereal` dataset but manually modify the name, order, and labels (and colors unless you use Color Brewer). The result should be such that the legend is in your custom order but bars are still in alphabetical order (with one-letter abbreviations underneath). Not a useful plot, but a useful exercise.

5.3.1.2 The solution

I decided to use `scale_fill_brewer`, so I didn't need to manually set the colors.

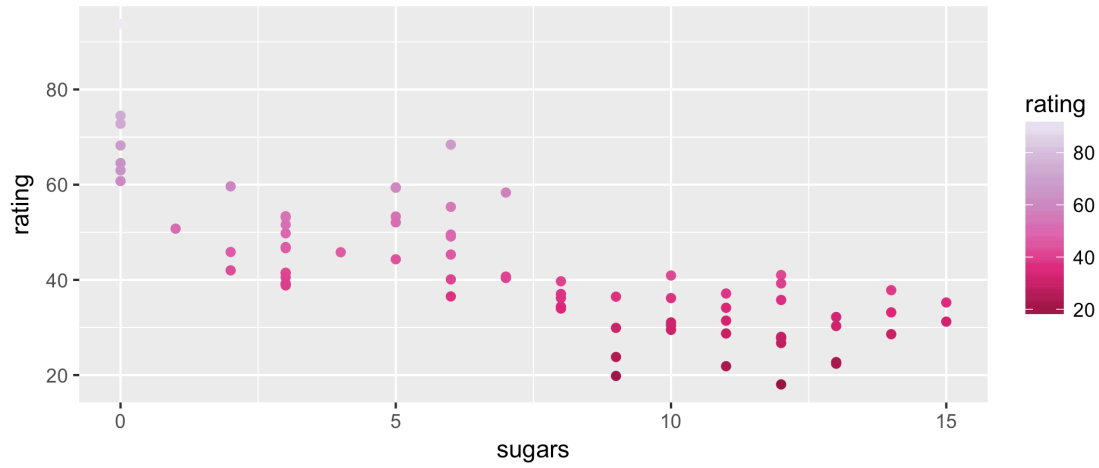
```
ggplot(cereal, aes(mfr)) +
    geom_bar(aes(fill = mfr), color = "grey25") +
    scale_fill_brewer(type = "qual", palette = "Pastel2",
                      name = "Manufacturer",
                       breaks = c("P", "Q", "R", "G", "N", "K"),
                       labels = c("post", "quaker oats", "ralstom purina",
                                  "general mills", "nabisco", "kellog's"))
```



5.4 Bonus: Special changes for Color Brewer

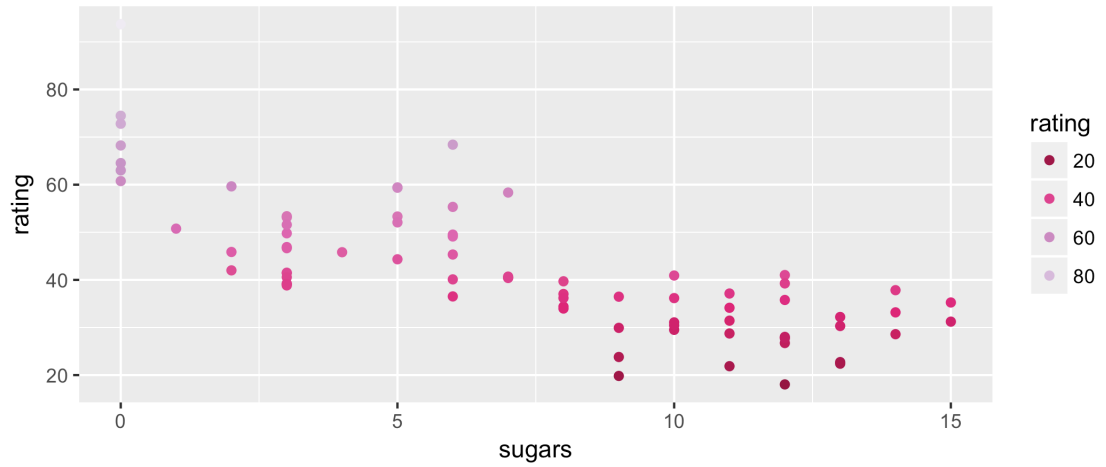
If you're using Color Brewer for your colors, you have a couple additional changes you can make to your legend. If you're plotting a continuous variable, you normally get a continuous scale in your legend:

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point(aes(color = rating)) +
    scale_color_distiller(type = "seq", palette = "PuRd")
```



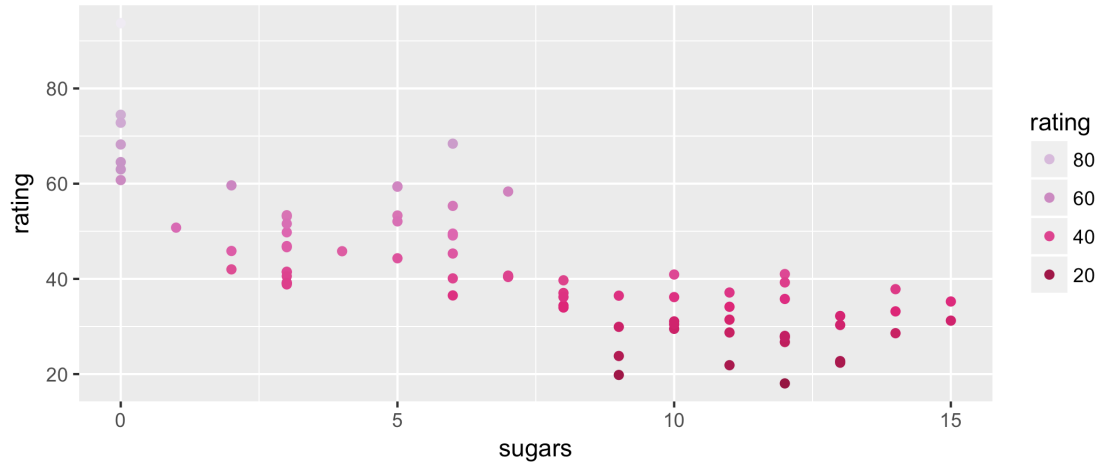
You can actually add the `guide = "legend"` argument, and it'll turn it into a categorical-looking legend. To be clear, the dots on the graph still use a continuous color scheme, but the legend is at least a little bit cleaner.

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point(aes(color = rating)) +
    scale_color_distiller(type = "seq", palette = "PuRd", guide = "legend")
```



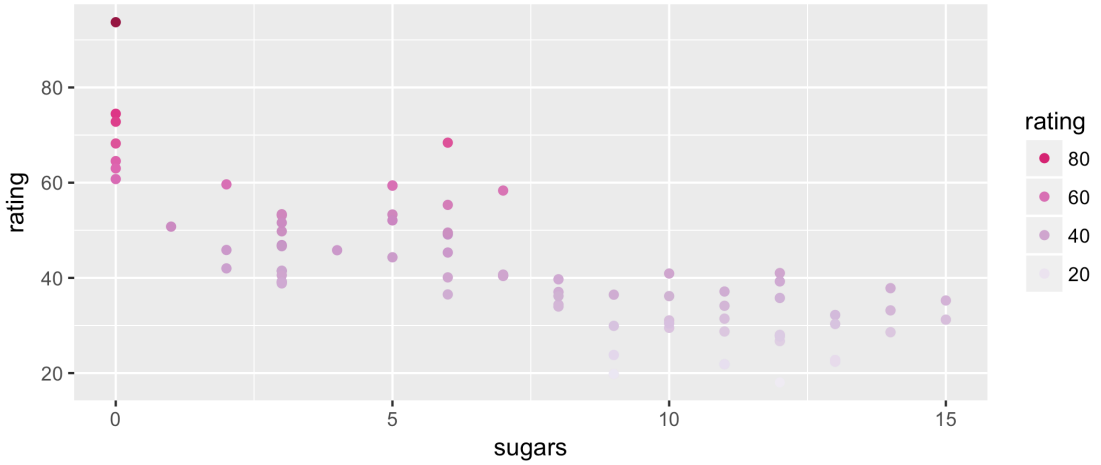
For some reason though, this has the effect of reversing the order so that high numbers are at the bottom. We can flip the way they appear in the legend by taking out `guide = "legend"` and using `guide = guide_legend(reverse = TRUE)` instead. Slightly cumbersome, but it gets the job done.

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point(aes(color = rating)) +
    scale_color_distiller(type = "seq", palette = "PuRd",
                          guide = guide_legend(reverse=TRUE))
```



Of course now if we want to change it so that high numbers get the darker color, we have to add `direction = 1` to reverse the order.

```
ggplot(cereal, aes(sugars, rating)) +
    geom_point(aes(color = rating)) +
    scale_color_distiller(type = "seq", palette = "PuRd",
                          guide = guide_legend(reverse=TRUE), direction = 1)
```



By the way, if `direction = 1` doesn't make any changes in other plots, try `direction = -1` instead. I can't figure out which one to use.

Anyway, because Color Brewer does soem cool things, it takes a little more work to get things done, but the result is a pretty good looking plot.

5.5 Credit to R Cookbook

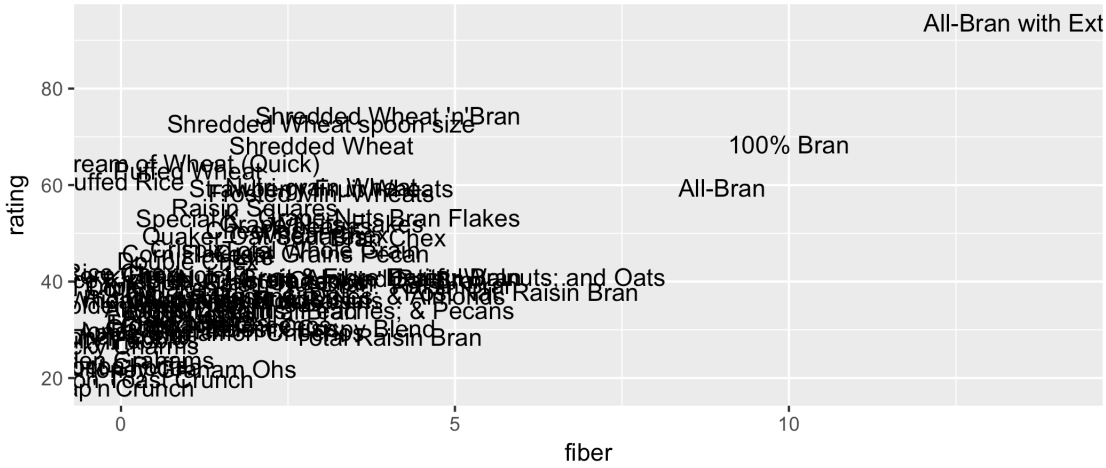
Much of the material from this section was borrowed from Winston Chang's *http://www.cookbook-r.com*, specifically the page on *legends in ggplot2*. I refer to this page all the time in my own research, and there's so much more that is covered there that I couldn't get to here. Be sure to check it out.

6 Faceting

Let's say we make a plot and it gets a little cumbersome because there are lots of points being shown. This is particularly true when you have text being displayed instead of points. Sometimes it's nice to split the plot up into meaningful groups and look at each group individually.

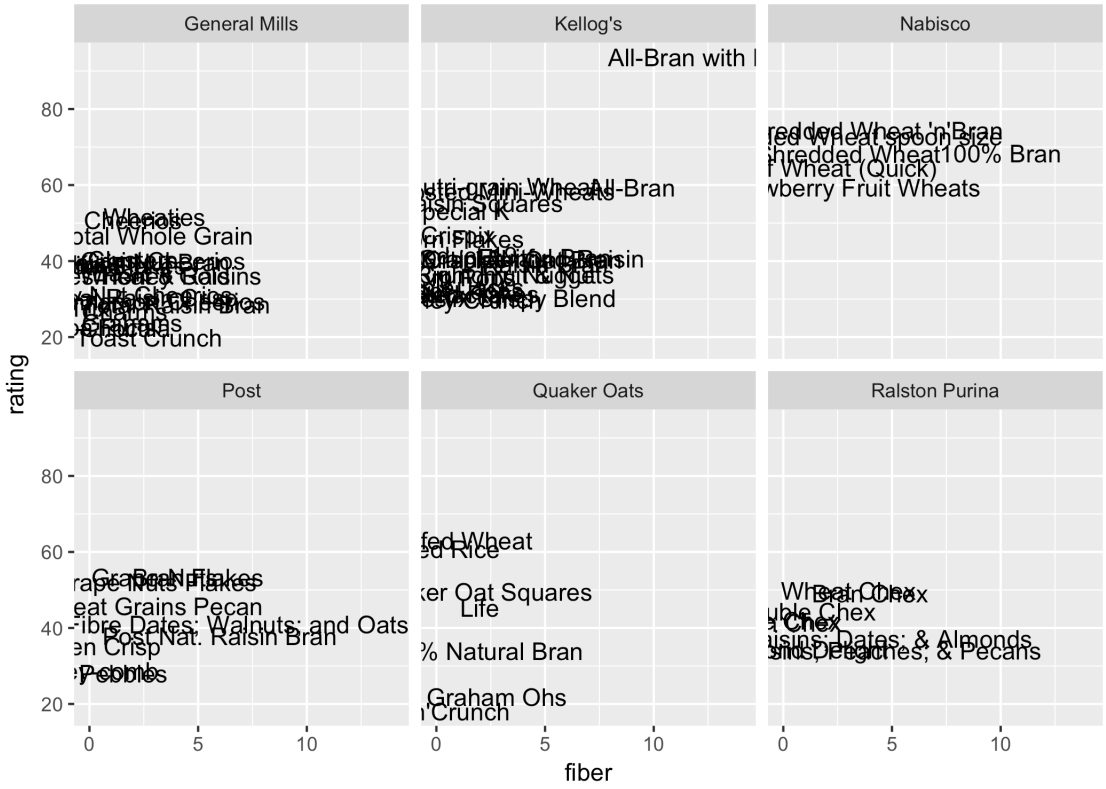
For example, let's say we want to see how much fiber a cereal has and compare it to how it's rating is. But we want to be able to see the name of the cereal itself, so we use `geom_text` instead of `geom_point`.

```
ggplot(cereal_renamed, aes(fiber, rating)) +
    geom_text(aes(label = name))
```



Ugh. That's a little messy. Maybe it might help if we could look at each manufacturer individually. The long way to do this is to create six different datasets by subsetting the original, and then doing six different plots, and finding a way to plot them all in one in a grid-like layout. But that's way too much work. Fortunately, ggplot2 has a way to do this with just one simple line of code: `facet_wrap`. Here, we can put the name of the column we want to split it up by—but it has to be after a tilde "~" (that squiggle thing left of the number "1" key). The reasons behind this are beyond the scope of this workshop, but just know you have to put it there. The result is exactly what we need.

```
ggplot(cereal_renamed, aes(fiber, rating)) +
    geom_text(aes(label = name)) +
    facet_wrap(~mfr)
```

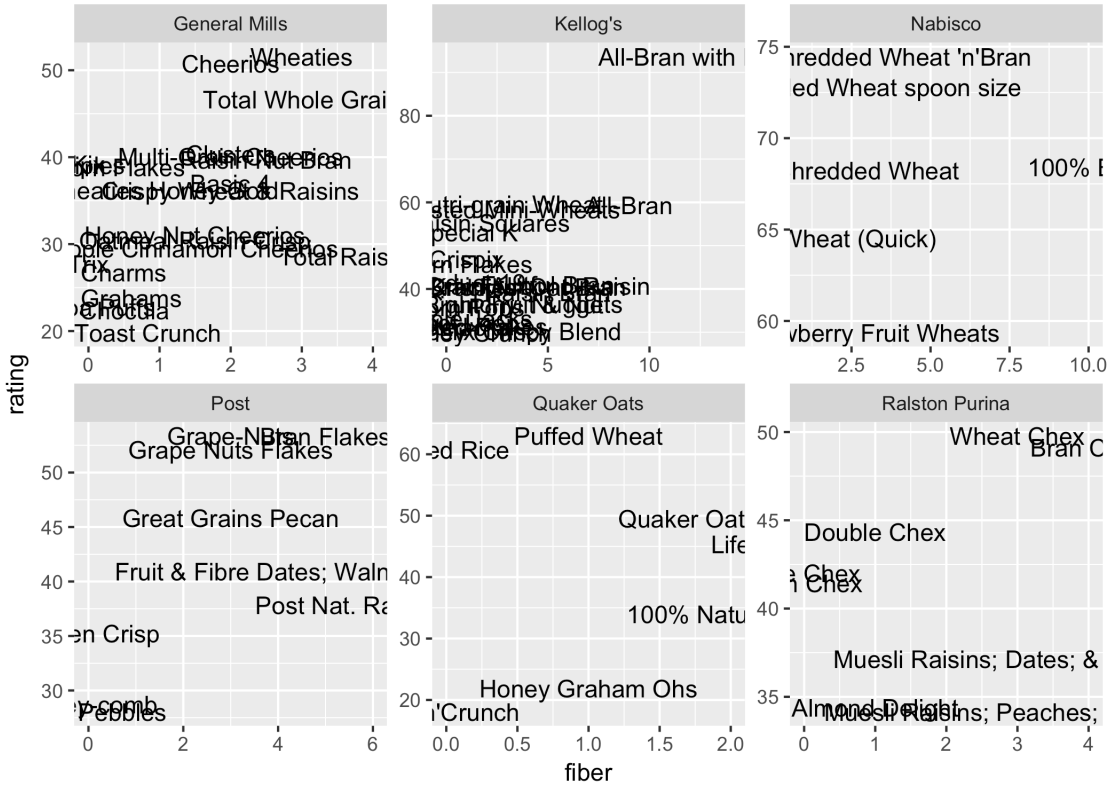


Here we can see that each manufacturer has its own plot, and the order is determined by the underlying order in the dataset. Here's it's alphabetical, but you can change that the same way that was shown above.

If you look closely, you'll see that the *x*- and *y*-axes are the same for each plot. In other words, it's clear to see that Nabisco cereals have a generally higher rating than most of the other ones and that Kellog's All Bran with Extra Fiber is true to its name and easy has the most fiber (and the highest rating). This is good because it makes comparisons across manufacturers easy.

But if we don't care about differences *between* manufacturers, and just want to look at differences *within* them, we can have each plot zoom in to just the data that's being shown. We do this by adding the `scales = "free"` argument to `facet_wrap`.

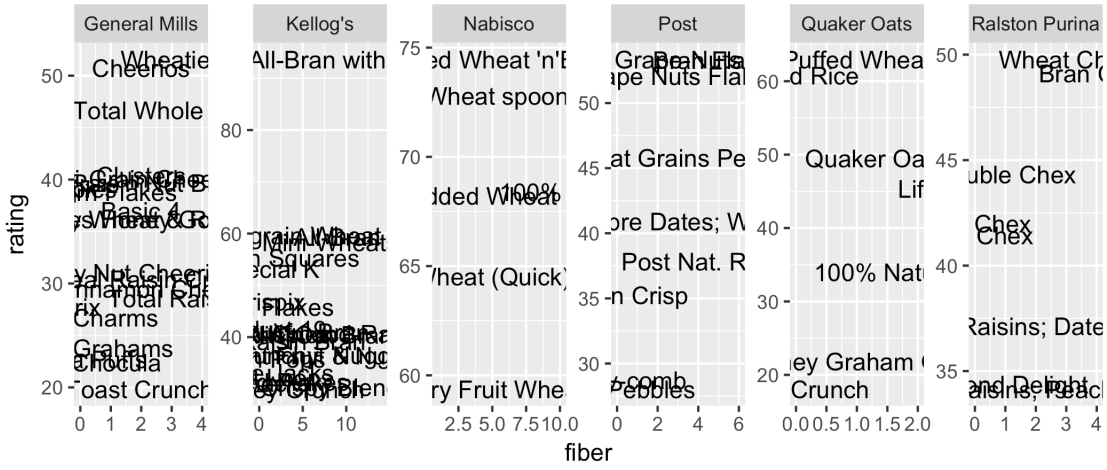
```
ggplot(cereal_renamed, aes(fiber, rating)) +
    geom_text(aes(label = name)) +
    facet_wrap(~mfr, scales = "free")
```



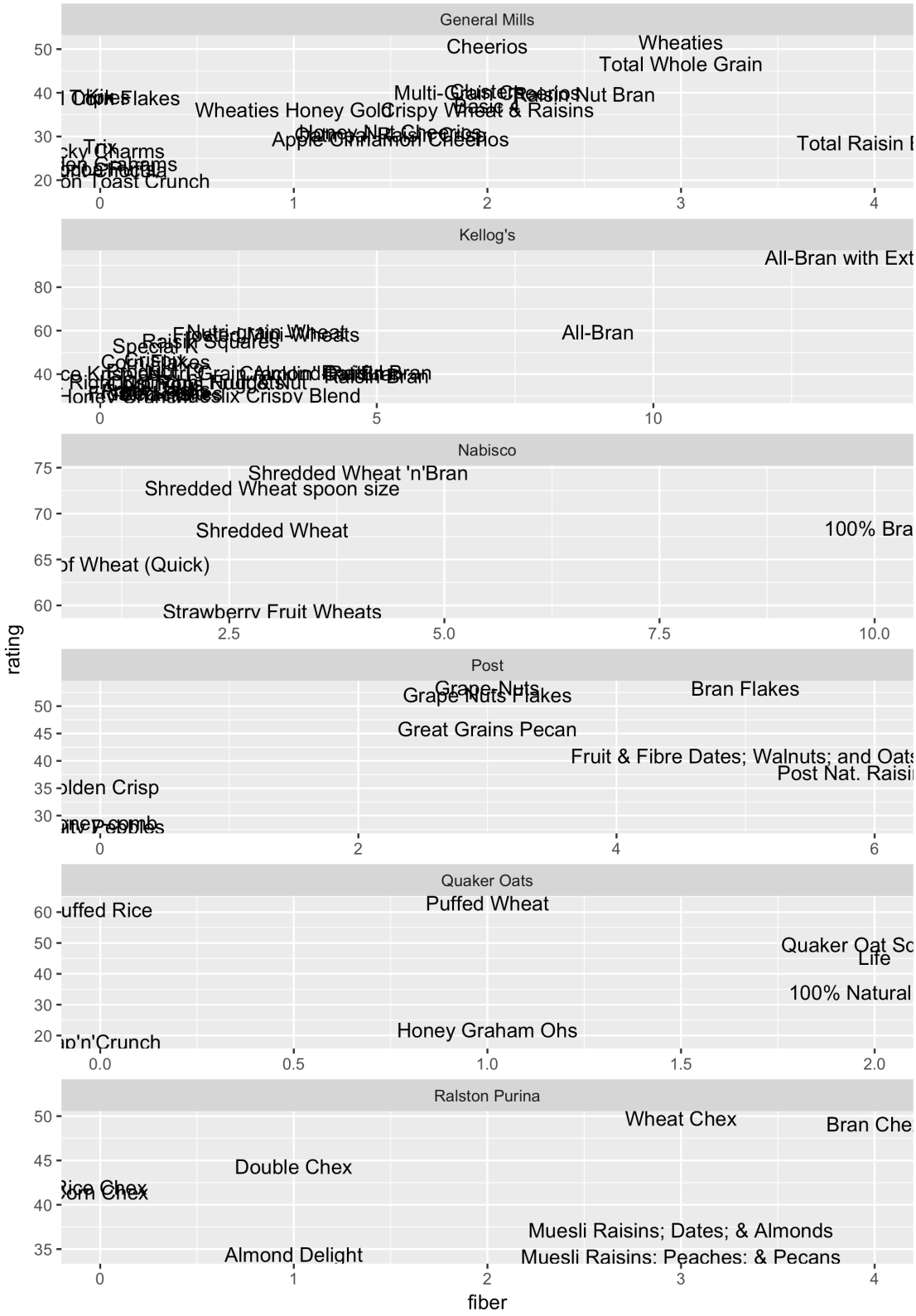
This plot shows the exact same data, but it highlights different things. For example, we can see clearly see the differences between Quaker Oats cereals, which was harder to see when they were all squished together before.

We can also change the number of rows and columns. By default, it'll do roughly a square layout with approximately equal numbers of rows and columns. If you want them all to be side-by-side (perhaps you're making a poster), you can set `nrow = 1` or if you want them all vertical, you can set `ncol = 1`.

```
ggplot(cereal_renamed, aes(fiber, rating)) +
    geom_text(aes(label = name)) +
    facet_wrap(~mfr, scales = "free", nrow = 1)
```




```
ggplot(cereal_renamed, aes(fiber, rating)) +
    geom_text(aes(label = name)) +
    facet_wrap(~mfr, scales = "free", ncol = 1)
```



More commonly, you'll want to specify something in between here, but it's a good trick to be aware of.

6.1.1 Your turn!

6.1.1.1 The challenge

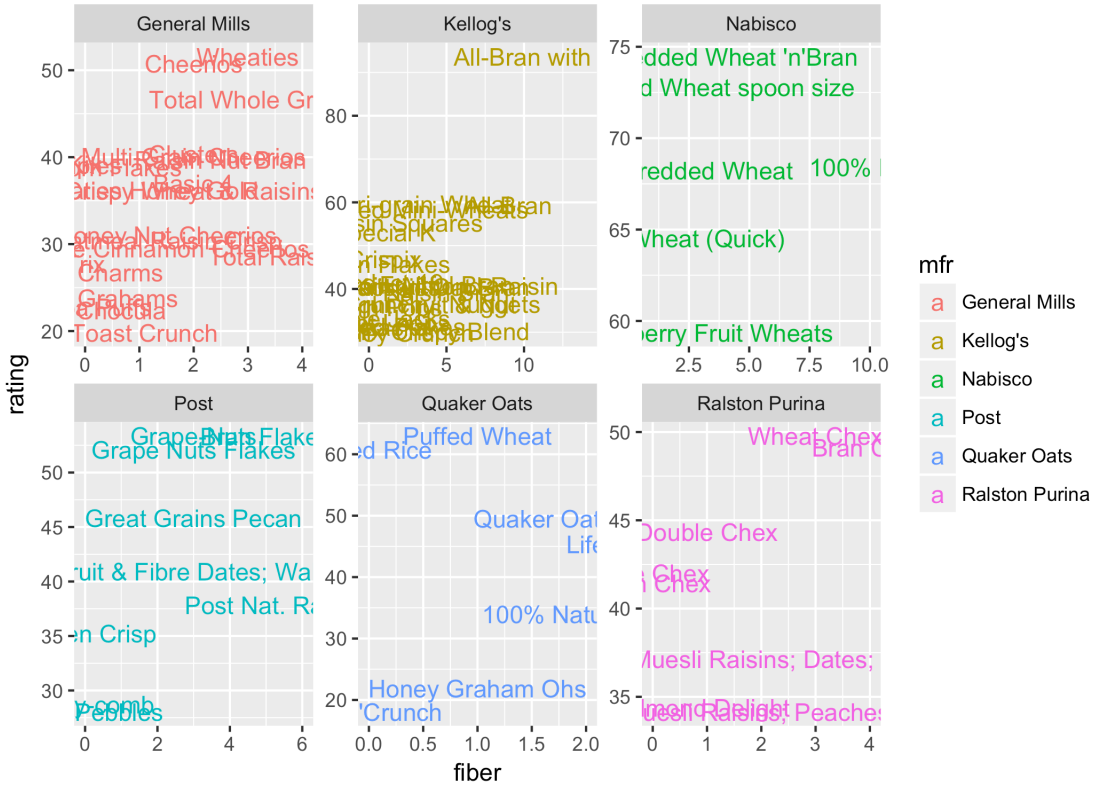
1. What happens if you use `facet_wrap` on some variable that's aready used in your plot? For example, what if you modify the above plot to have the manufacturers each with their own color?

2. What happens if you do a bar plot with fiber and manufacturer, but facet it by manufacturer? Is this a useful thing?

6.1.1.2 The solution

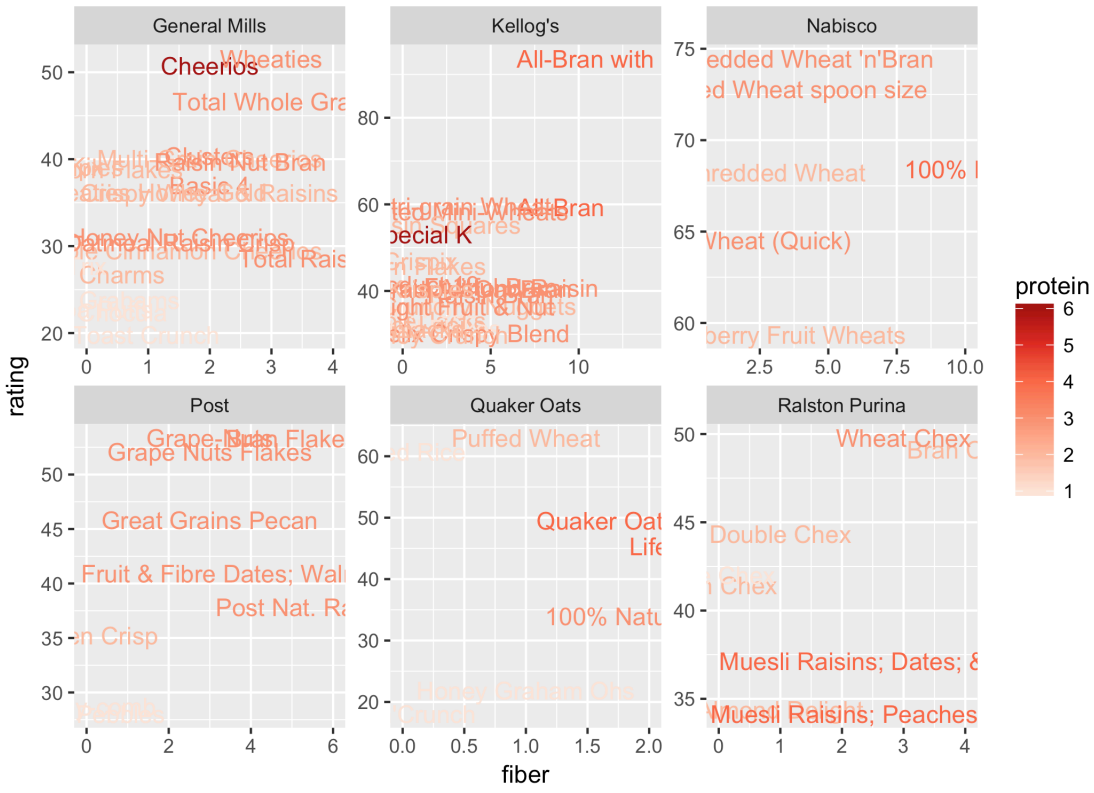
If you use the same variable for color and faceting, it does what you expect. The colors just don't add much to the plot.

```
ggplot(cereal_renamed, aes(fiber, rating, color = mfr)) +
    geom_text(aes(label = name)) +
    facet_wrap(~mfr, scales = "free")
```



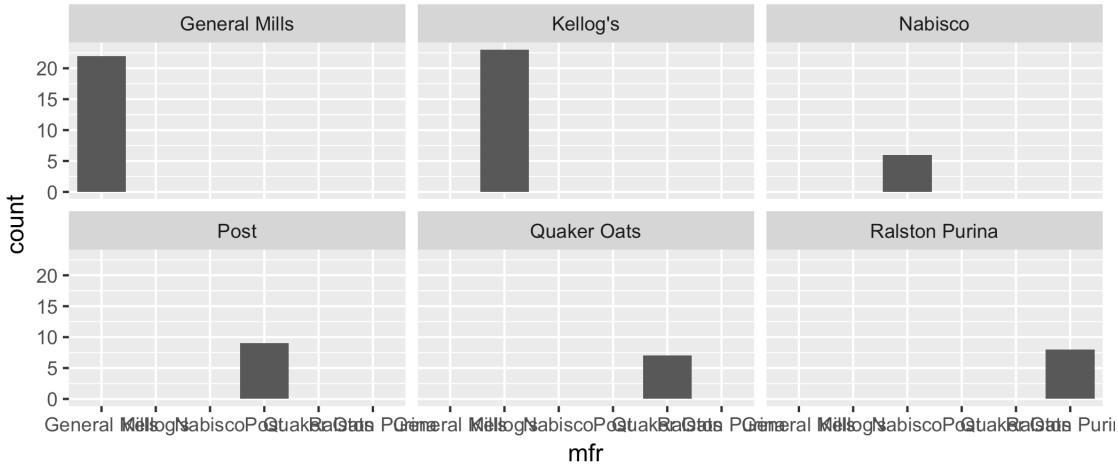
Instead it might be more useful to add another variable for color.

```
ggplot(cereal_renamed, aes(fiber, rating, color = protein)) +
    geom_text(aes(label = name)) +
    scale_color_distiller(type = "seq", palette = "Reds", direction = 1) +
    facet_wrap(~mfr, scales = "free")
```



If you use a facet wrap on the same variable as the columns, it doesn't really help much and it turns into a pretty lame graph.

```
ggplot(cereal_renamed, aes(mfr)) +
    geom_bar() +
    facet_wrap(~mfr)
```

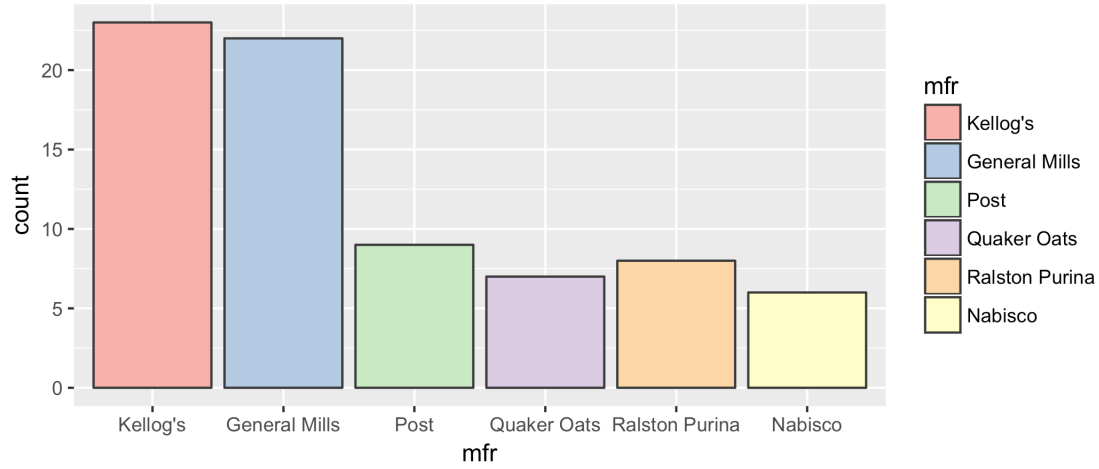


7 Themes

The last topic today is how to finally change the overall theme of your plot. By now, you might want a background that isn't grey. Fortunately, ggplot2 comes with several themes preinstalled, so it's easy to switch between them.

First, what I'll do is actually save a plot as an R object. I'll choose the barplot that we made earlier with the Color Brewer colors. When we save it as an object (by prefacing the `ggplot` call with `p <-`), it doesn't plot right away. Instead, you have to type the name of the object (we're calling it `p`, but it whatever you want).

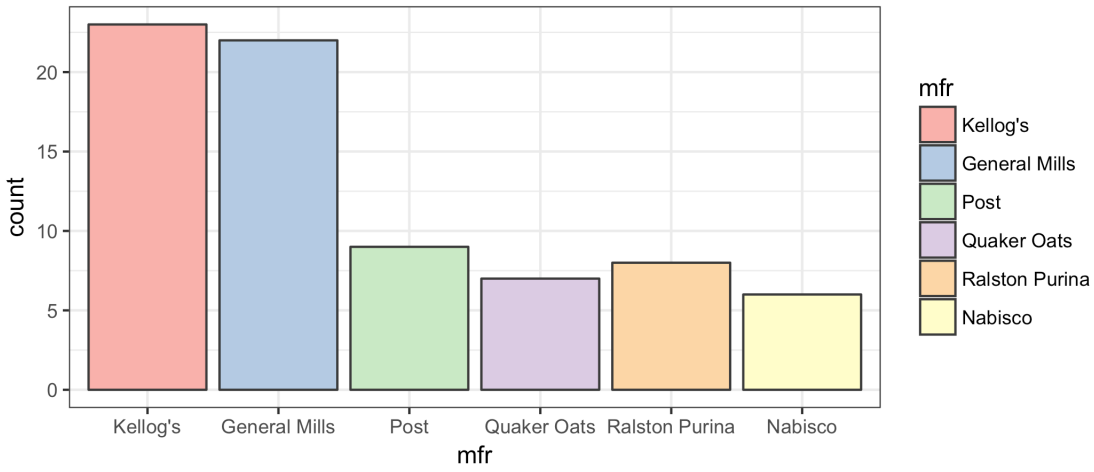
```
p <- ggplot(cereal_ordered, aes(mfr)) +
    geom_bar(aes(fill = mfr), color = "grey25") +
    scale_fill_brewer(type = "qual", palette = "Pastel1")
p
```



The reason for this is that I don't have to sit there and copy and paste all the code in every time. I can just add layers to this `p` and it'll update the whole thing.

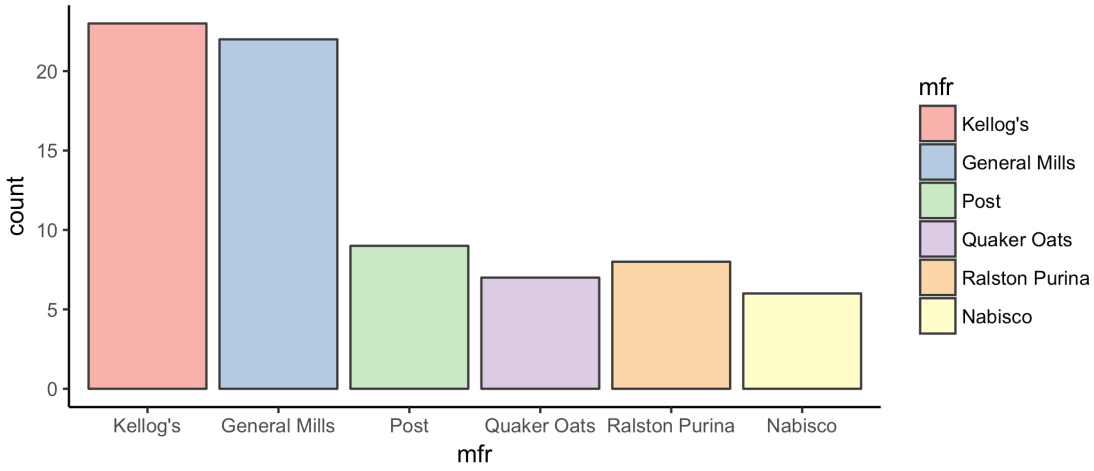
My go-to theme is called `theme_bw()` for just "black and white." The biggest differnce is that now the background is white instead of gray But it also adds a thin black border around the whole thing and has faint grey grid lines instead of white ones.

```
p + theme_bw()
```



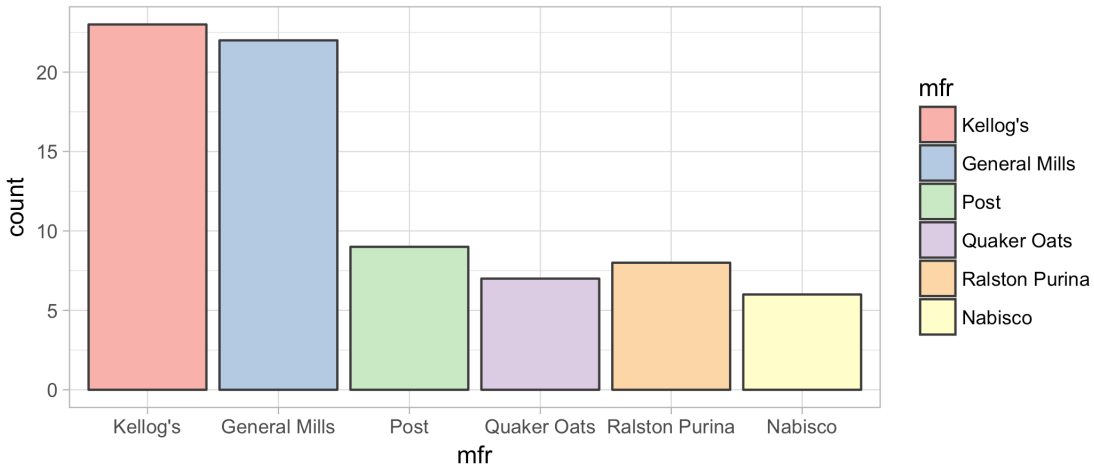
The `classic` theme has no grid and the top and right parts of the box are gone too, just leaving the x and y axes.

```
p + theme_classic()
```



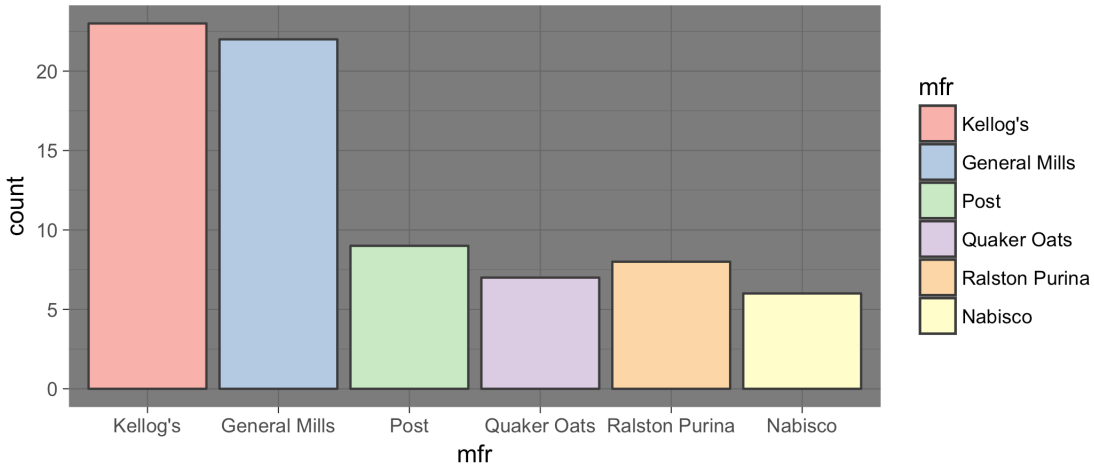
The `light` theme is very similar to `bw`. The biggest difference is the outside box is lighter.

```
p + theme_light()
```



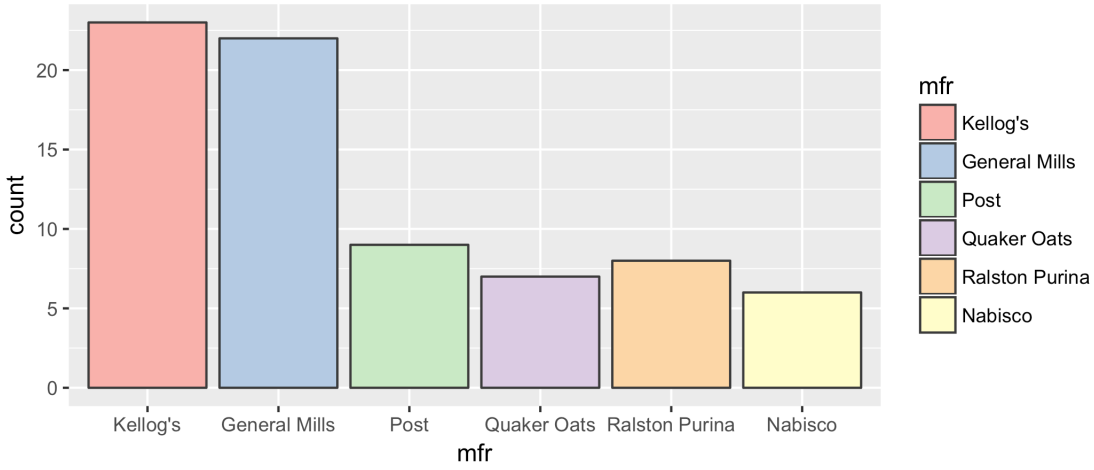
If you really like the gray, you can go for the `dark` theme, which has a darker background and a darker gray for the grid lines.

```
p + theme_dark()
```



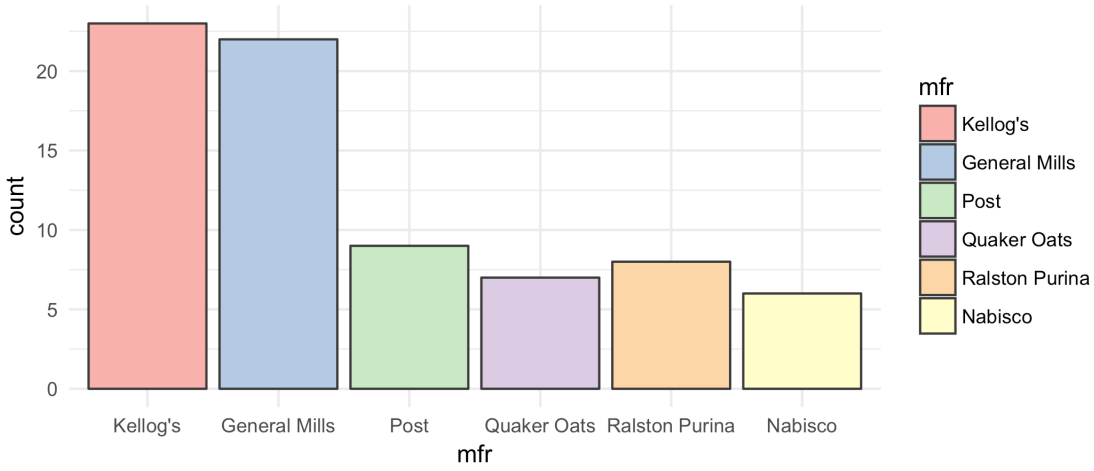
In fact, if you really like default background, you can set it specifically. This is the default for a reason because that specific shade of gray has been chosen to make colors stand out more.

```
p + theme_gray()
```



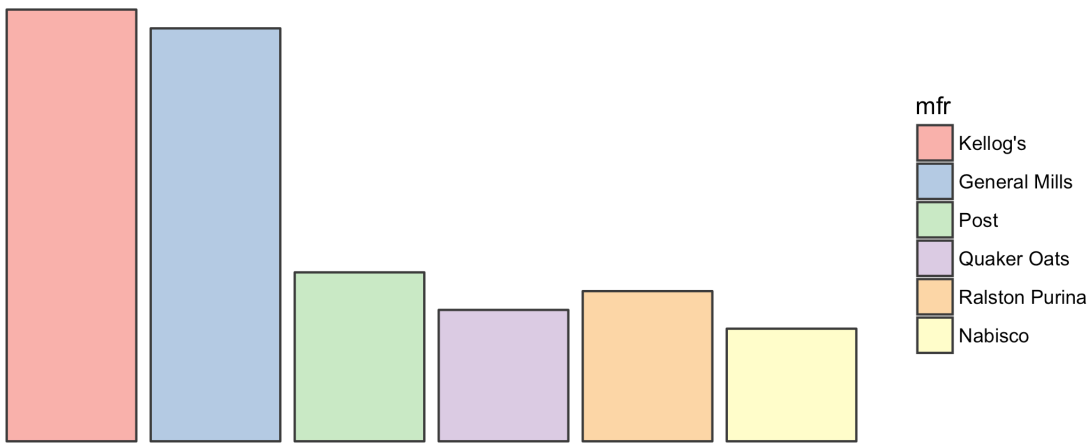
The `minimal` theme is even simpler than `light`. It doens't have the outside border but it does retain the inside grid.

```
p + theme_minimal()
```



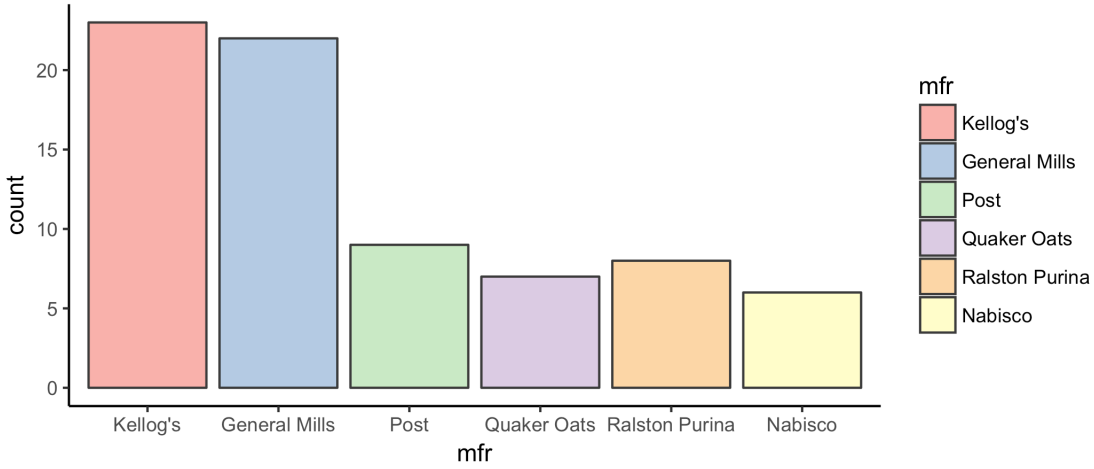
Finally, you can go completely blank with `void`. This may seem a little weird, but it does have useful purposes, like when creating maps.

```
p + theme_void()
```



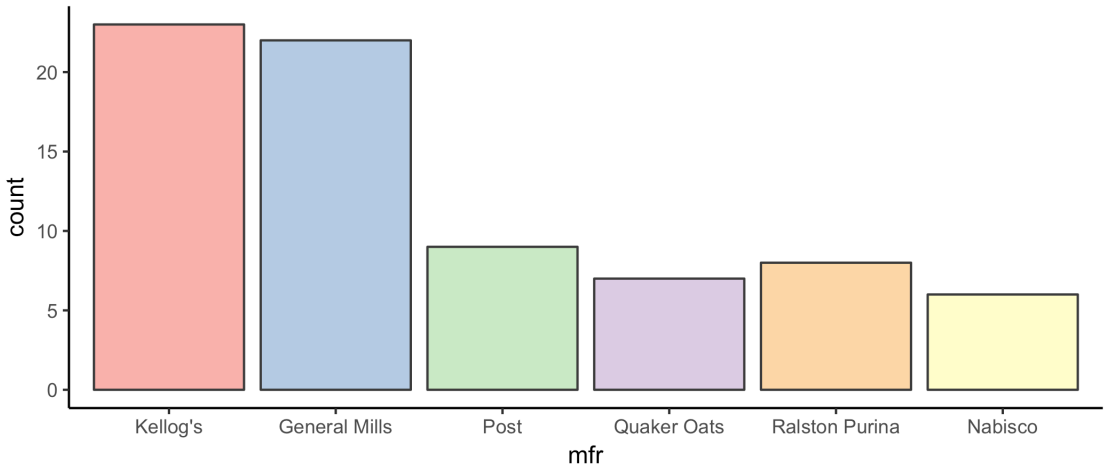
Something to keep in mind with themes is that they override any other `theme` layer you might have included. For example, if you want to remove the legend and then use the `classic` theme, it's still there:

```
p + theme(legend.position = "none") +
    theme_classic()
```



No this isn't a bug. The reason for this is because `theme_*` is actually a shortcut for a whole bunch of other `theme` elements. As a result, there are two `theme` functions, so the second one overrides the first one. Fortunately, we can fix this by just reversing the order of `theme` and `theme_classic`:

```
p + theme_classic() +
    theme(legend.position = "none")
```



If you don't like any of these themes, or you want to change just one aspect of them (the width of the lines, the color of the background, the gridlines, etc.), you can! That's what we'll be doing next week is diving deep into the `theme` function and seeing what kinds of things you can do to change your plot, and how to wrap all these changes up into a custom theme.

8 Bonus: Saving plots

Everything we've done so far is just a temporary image that will disappear when you close R. Crucially, it's not going to show up in your powerpoints or papers. There is a way to save plots by clicking things in RStudio, but the whole purpose of this workshop is to do things via code because it's a lot easier to reproduce it.

The way to save things is to use the function `ggsave` immediately after creating a plot. You can specify the path to where you want it saved by typing a full or relative path. Note that Windows users will need a double back slash (\\) while Mac users need a single forward slash (/). Also be sure to specify the name of the file itself and the filetype (`".png"`, `".jpg"`, etc.). You can specify the `width` and `height` in inches to control the size, which is super useful for making comparison charts. And you can specify the resolution using `dpi` ("dots per inch"). The default, which is the standard for many publications, is `300`.

```
p + theme_classic() +
    theme(legend.position = "none")

# For Macs
ggsave("/Users/joeystanley/Desktop/plots/barplot.png",
       dpi = 300, height = 7, width = 7)

# For Windows
ggsave("C:\\Users\\joeystanley\\Desktop\\plots\\barplot.png",
       dpi = 300, height = 7, width = 7)
```

9 Final Remarks

The last two workshops have shown us that ggplot2 is a workhorse and can do a lot things. Today we saw how to change the big things in your plots (titles, axes, colors, names, orders, legends, facets, themes) and how it's relatively straightforward to make these changes. With these tools under your belt, you're on your way to making some really professional-looking visualizations.