LSC 563 Lecture 7: Basic Customization

Doug Joubert

2021-11-02

Table of Contents

[Introduction 1](#_Toc96011179)

[Data Used in this Lesson 1](#_Toc96011180)

[Packages Used in this Lesson 2](#_Toc96011181)

[Grouped Data and the “Group” Aesthetic 2](#_Toc96011182)

[Facet to Make Small Multiples 5](#_Toc96011183)

[Facet Wrap 5](#_Toc96011184)

[Facet Grid 7](#_Toc96011185)

[Exercise 1 9](#_Toc96011186)

[Exercise 1a: Answer 10](#_Toc96011187)

[Exercise 2 10](#_Toc96011188)

[Exercise 2: Answer 10](#_Toc96011189)

[Geoms Can Transform Data 11](#_Toc96011190)

[Avoid Transformations When Necessary 18](#_Toc96011191)

[Refining Plots 22](#_Toc96011192)

[Figure Titles 22](#_Toc96011193)

[Axis and Legend Titles 23](#_Toc96011194)

[Refining Plot Examples 25](#_Toc96011195)

[Using Color to Your Advantage 30](#_Toc96011196)

[RColorBrewer 30](#_Toc96011197)

[Using Themes 37](#_Toc96011198)

[References 39](#_Toc96011199)

# Introduction

## Data Used in this Lesson

* gapminder data as part of [gapminder package](https://www.rdocumentation.org/packages/gapminder/versions/0.3.0)

## Packages Used in this Lesson

* gapminder
* tidyverse
* socviz
* ggrepel
* RColorBrewer

# Grouped Data and the “Group” Aesthetic

First we need to load our libraries.

One of our goals is to learn how to make new kinds of graph. This means learning some new geoms, the functions that make particular kinds of plots. But we will also get a better sense of what ggplot is doing when it draws plots, and learn more about how to write code that prepares our data to be plotted (Healy, 2018b).

In this lesson we will discuss some useful features of ggplot and also talk about ways that these features can cause problems. They have to do with how to tell ggplot more about the internal structure of your data (*grouping*), how to break up your data into pieces for a plot (*faceting*), and how to get ggplot to perform some calculations on or summarize your data before producing the plot (*transforming*) (Healy, 2018b).

As I have mentioned both in class and in the labs, code almost never works properly the first time you write it. This is the main reason why it is important to type out the lab exercises and follow along manually. This gives you a much better sense of how the syntax of the language works.

As we noted in the last lecture The ggplot library is an implementation of the “grammar” of graphics, an idea developed by Wilkinson. The grammar is a set of rules for producing graphics from data, taking pieces of data and mapping them to geometric objects (like points and lines) that have aesthetic attributes (like position, color, and size), together with further rules for transforming the data if needed (Healy, 2018b).

Like other rules of syntax, the grammar limits the structure of what you can say, but it does not automatically make what you say sensible or meaningful (Healy, 2018b). However, sometimes your code will not produce a plot at all because of some syntax error in R. For example, you might forget a + sign between geom functions or lose a parenthesis somewhere so that your function statement becomes unbalanced (Healy, 2018b). In those cases R will complain (perhaps in an opaque way) that something has gone wrong (Healy, 2018b).

Let’s begin again with our Gapminder dataset. Imagine we wanted to plot the trajectory of life expectancy over time for each country in the data. We look in the R Documentation and determine we can do this with the line geom geom\_line().

Let us take a peek at our data, only looking at the first 5 rows of data:

head(gapminder, 5)

## # A tibble: 5 x 6  
## country continent year lifeExp pop gdpPercap  
## <fct> <fct> <int> <dbl> <int> <dbl>  
## 1 Afghanistan Asia 1952 28.8 8425333 779.  
## 2 Afghanistan Asia 1957 30.3 9240934 821.  
## 3 Afghanistan Asia 1962 32.0 10267083 853.  
## 4 Afghanistan Asia 1967 34.0 11537966 836.  
## 5 Afghanistan Asia 1972 36.1 13079460 740.

We need to map year to x and gdpPercap to y. How would we map this out in ggplot?

Let us build our graph, by first creating our graph object:

p <- ggplot(data = gapminder, mapping = aes(x = year, y = gdpPercap))

Now let us graph the data:

p +  
 geom\_line()

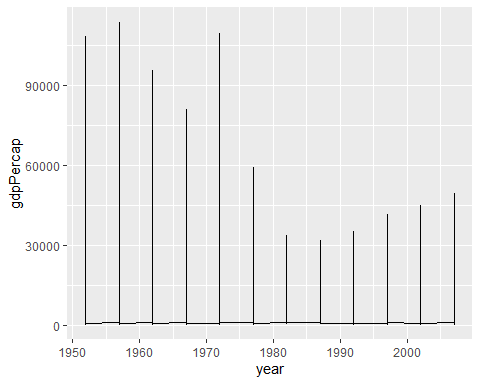


Figure 6.1: GDP per capita, by year

Wow, what happened? Well, ggplot tried to figure this out on its own. While ggplot will make a pretty good guess as to the structure of the data, it does not know that the yearly observations in the data are grouped by country. We have to tell it. Because we have not, geom\_line() tries to join up all the lines for each particular year in the order they appear in the dataset (Healy, 2018b).

When ggplot successfully makes a plot but the result looks insane, the reason is almost always that something has gone wrong in the mapping between the data and aesthetics for the geom being used (Healy, 2018b). This is so common there’s even a Twitter account devoted to the “Accidental aRt” that results.

So, let us try this again, using the group aesthetic to tell ggplot explicitly about this country-level structure.

p +  
 geom\_line(aes(group = country))

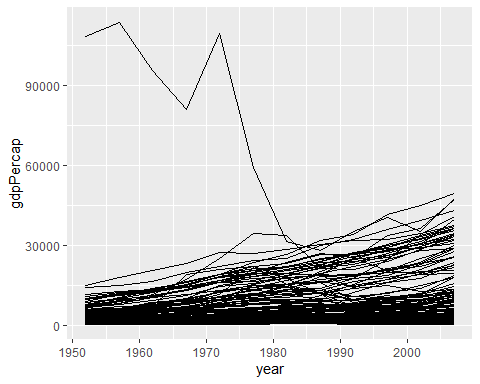


Figure 6.2: GDP per capita, by year, grouped by country.

The plot in figure 6.2 is still fairly rough, but it is showing the data properly, with each line representing the trajectory of a country over time. The gigantic outlier is Kuwait, in case you are interested (Healy, 2018b).

The group aesthetic is usually only needed when the grouping information you need to tell ggplot about is not built into the variables being mapped (Healy, 2018b).

# Facet to Make Small Multiples

The overall trend in Figure 6.2 is more or less clear; however, it looks a little messy. One option is to *facet* the data by some third variable, making a “small multiple” plot. This is a powerful technique that allows a lot of information to be presented compactly and in a consistently comparable way (Healy, 2018b). We will do this facet\_wrap function, that is part of ggplot.

Facets are not a geom but a way of organizing a series of geoms. In this case we have the *continent* variable available to us.

## Facet Wrap

We will use facet\_wrap() to split our plot by continent.

The facet\_wrap() function can take a series of arguments, which is specified using R’s “formula” syntax, which uses facets = vars(continent) within the facet\_wrap statement.

Just like [aes()](https://ggplot2.tidyverse.org/reference/aes.html), vars() is a [quoting function](https://rlang.r-lib.org/reference/nse-defuse.html) that takes inputs to be evaluated in the context of a dataset. These inputs can be:

* variable names
* complex expressions

Facets are usually a one-sided formula. Most of the time you will just want a single variable on the right side of the formula. For our first example, we will just use a single term in our formula, which is the variable we want the data broken up by: facet\_wrap(facets = vars(continent)).

p +  
 geom\_line(aes(group = country)) +  
 facet\_wrap(facets = vars(continent))

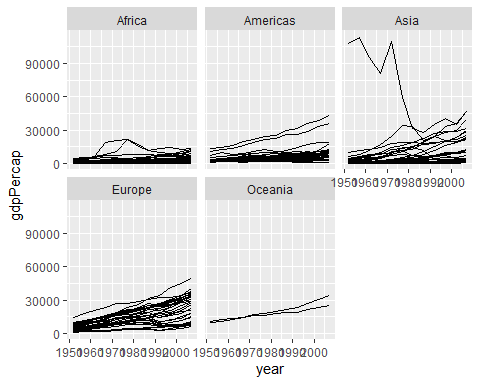


Figure 6.3: GDP per capita, by year, grouped by country and faceted by continent.

In Figure 6.3, each facet is labeled at the top. The overall layout of Figure 6.3 minimizes the duplication of axis labels and other scales. Remember, too, that we can still include other geoms as before, and they will be layered within each facet. We can also use the ncol argument to facet\_wrap() to control the number of columns used to lay out the facets (Healy, 2018b).

p +  
 geom\_line(aes(group = country)) +  
 scale\_y\_log10(labels=scales::dollar ) +  
 facet\_wrap(facets = vars(continent), ncol = 5) +  
 labs (x = "Year", y = "GDP per capita", title= "GDP per capita on Five Continents")

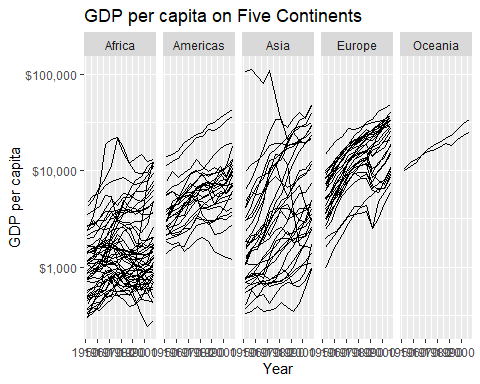


Figure 6.4: GDP per capita, by year, grouped by country, faceted by continent, and customized.

Let is save the previous plot to and object and then use ggsaveto export the file.

plot\_gdp <- p +  
 geom\_line(aes(group = country)) +  
 scale\_y\_log10(labels=scales::dollar ) +  
 facet\_wrap(~continent, ncol = 5) +  
 labs (x = "Year", y = "GDP per capita", title= "GDP per capita on Five Continents")

As you can see in Figure 6.4, the plot is kind of squished. So we will need to modify the default width and height, when we save the file.

ggsave("../figures/plot\_gdp.pdf", plot = plot\_gdp, width = 11, height = 8.5)

## Facet Grid

The *facet\_wrap* geometry extracts plots into an arbitrary number of dimensions to allow them to cleanly fit on one page.

In contrast, the *facet\_grid* geometry allows you to explicitly specify how you want your plots to be arranged via formula notation. You control how you want your plots arranged using the following notation options:

p + facet\_grid(rows = vars(drv))

p + facet\_grid(cols = vars(cyl))

p + facet\_grid(vars(drv), vars(cyl))

Let us pull up the help documentation for facet\_grid.

?facet\_grid

## starting httpd help server ... done

As we can see facet\_grid() forms a matrix of panels defined by row and column faceting variables. It is most useful when you have two discrete variables, and all combinations of the variables exist in the data. If you have only one variable with many levels, try [facet\_wrap()](https://ggplot2.tidyverse.org/reference/facet_wrap.html).

To see the difference, let’s introduce [gss\_sm](https://www.rdocumentation.org/packages/socviz/versions/1.2/topics/gss_sm), a new dataset that we will use in the next few sections. The GSS is a long-running survey of American adults that asks about a range of topics of interest to social scientists. See this [site](http://gss.norc.org/Get-Documentation) for full documentation of the [variables](https://www.rdocumentation.org/packages/socviz/versions/1.2/topics/gss_sm).

The GSS data in gss\_sm contains many measures of this sort. You can take a peek at it, as usual, by typing its name at the console. You could also try glimpse(gss\_sm), which will give a compact summary of all the variables in the data.

In the social sciences, the data is often categorical. Sometimes the categories are unordered, as with ethnicity or sex. But they may also be ordered, as when we measure highest level of education attained on a scale ranging from elementary school to postgraduate degree.

We will make a smoothed scatterplot of the relationship between the age of the respondent and the number of children they have [Figure 6.1]. In gss\_sm the childs variable is a numeric count of the respondent’s children.

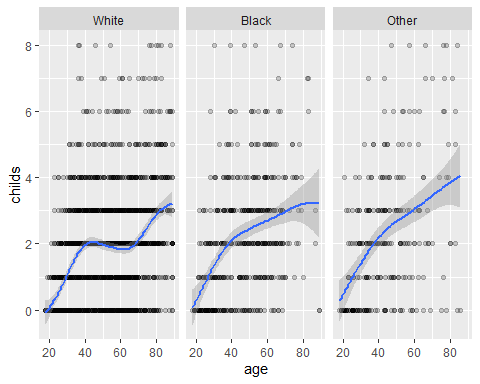
We will then facet this relationship by sex and race of the respondent. We use R’s formula notation in the facet\_grid function to facet sex and race. This time, because we are cross classifying our results, the formula is two-sided: face\_grid(sex ~ race).

p <- ggplot(data = gss\_sm, mapping = aes(x = age, y = childs))  
p +  
 geom\_point(alpha = 0.2) +  
 geom\_smooth() +  
 facet\_grid(rows = vars(sex)) +  
 facet\_grid(cols = vars(race))

## `geom\_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'

## Warning: Removed 18 rows containing non-finite values (stat\_smooth).

## Warning: Removed 18 rows containing missing values (geom\_point).



Multi-panel layouts of this kind are especially effective when used to summarize continuous variation (as in a scatterplot) across two or more categorical variables, with the categories (and hence the panels) ordered in some sensible way.

# Exercise 1

Create a new object called interviews\_plotting and load the interviews\_plotting.csv data into this object.

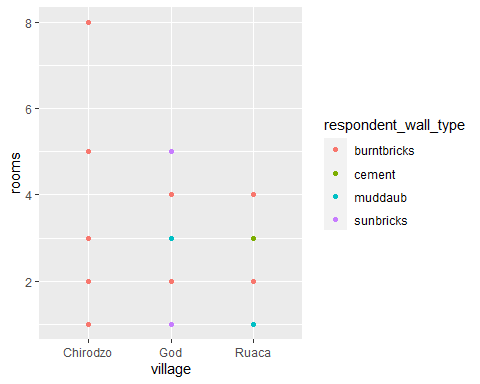
interviews\_plotting <- read\_csv("../raw\_data/interviews\_plotting.csv")

## Rows: 131 Columns: 45  
## -- Column specification --------------------------------------------------------  
## Delimiter: ","  
## chr (5): village, respondent\_wall\_type, memb\_assoc, affect\_conflicts, inst...  
## dbl (8): key\_ID, no\_membrs, years\_liv, rooms, liv\_count, no\_meals, number\_...  
## lgl (31): bicycle, television, solar\_panel, table, cow\_cart, radio, cow\_plo...  
## dttm (1): interview\_date  
##   
## i Use `spec()` to retrieve the full column specification for this data.  
## i Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

Use what you just learned to create a scatter plot of rooms by village with the respondent\_wall\_type showing in different colours. Does this seem like a good way to display the relationship between these variables? Looking at the data and what is displayed on the graph…what is going on?

## Exercise 1a: Answer

interviews\_plotting %>%   
 ggplot(mapping = aes(x = village, y = rooms)) +  
 geom\_point(mapping = aes(col = respondent\_wall\_type))

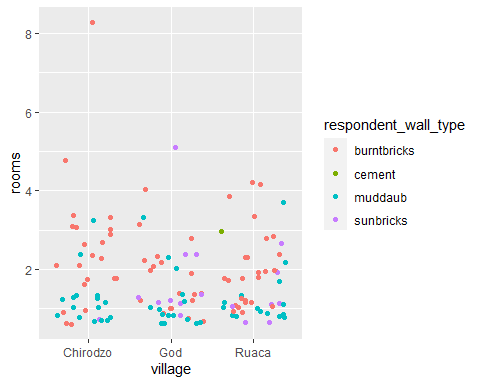


# Exercise 2

Try to fix your first graph, using another geom that Healy and Wilke discussed.

## Exercise 2: Answer

interviews\_plotting %>%   
 ggplot(mapping = aes(x = village, y = rooms)) +  
 geom\_jitter(mapping = aes(col = respondent\_wall\_type))



Is this a better way to display the data.

# Geoms Can Transform Data

STUDENTS CAN EXPLORE ON THEIR OWN, PICK UP IN USING COLORS TO YOUR ADVANTAGE SECTION

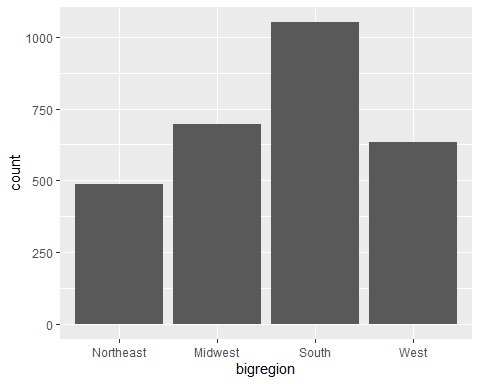
We have already seen several examples where geom\_smooth() was included as a way to add a trend line to the figure. Sometimes we plotted a [LOESS line](http://r-statistics.co/Loess-Regression-With-R.html), and sometimes a straight line from an [OLS regression](https://www.r-bloggers.com/2017/07/ordinary-least-squares-ols-linear-regression-in-r/). We did not have to have any strong idea of the differences between these methods.

Thus, some geoms plot our data directly on the figure, as is the case with geom\_point(), which takes variables designated as x and y and plots the points on a grid. However, other geoms clearly do more work on the data before it gets plotted. Every geom\_ function has an associated stat\_ function that it uses by default. The reverse is also the case: every stat\_ function has an associated geom\_function that it will plot by default if you ask it to.

This is not particularly important to know by itself, but sometimes want to calculate a different statistic. Also, sometimes the calculations being done by the stat\_ functions that work together with the geom\_functions might not be immediately obvious.

For example, consider Figure 6.2, what was produced by geom\_bar().

p <- ggplot(data = gss\_sm, mapping = aes(x = bigregion))  
p + geom\_bar()



How many mapping did we create in the code above? We specified just one mapping, aes (x = bigregion). The bar chart produced gives us a count of the number of (individual) observations in the data set by region of the United States. This seems sensible. However, there is also a y-axis variable here, count, that is not in the data. It has been calculated for us. Behind the scenes, geom\_bar called the default stat\_ function associated with it, stat\_count(). This function computes two new variables, count and prop (short for proportion). The count statistic is the one geom\_bar() uses by default.

What if we want a chart of relative frequencies rather than counts? In this case we will need to get the prop statistic instead. When ggplot calculates the count or the proportion, it returns temporary variables that we can use as mappings in our plots. The relevant statistic is called ..prop.. rather than prop. This syntax is to make sure these temporary variables won’t be confused with others we are working with. This is because we might already have a variable called count or prop in our dataset.

So our calls to it from the aes() function will generically look like this: <mapping> = <..statistic..>. In this case, we want y to use the calculated proportion, so we say aes(y =..prop..). Figure 6.3 displays the results.

p <- ggplot(data = gss\_sm, mapping = aes(x = bigregion))  
p +  
 geom\_bar(mapping = aes(y = ..prop..))

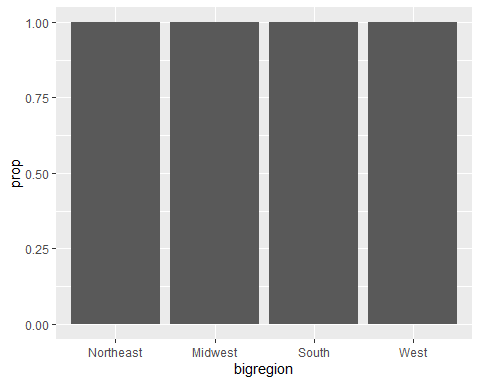


Figure 6.3:

The resulting plot in Figure 6.3 is still not right. In this case, we need to tell ggplot to ignore the x-categories when calculating denominator of the proportion and use the total number observations instead.

To do so we specify group = 1 inside the aes() call. The value of 1 is just a kind of “dummy group” that tells ggplot to use the whole dataset when establishing the denominator for its prop calculations. The result is shown in Figure 6.4

p <- ggplot(data = gss\_sm, mapping = aes(x = bigregion))  
p +   
 geom\_bar(mapping = aes(y = ..prop.., group = 1))

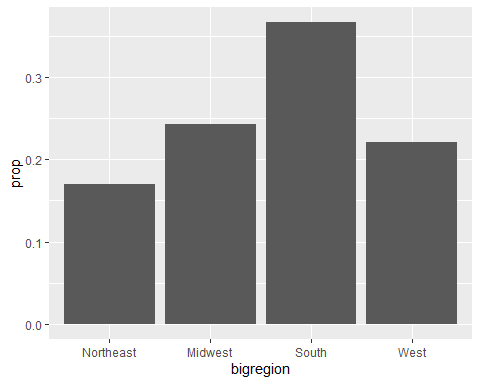


Figure 6.4:

Let’s look at another question from the survey. The gss\_sm data contains a religion variable derived from a question asking “What is your religious preference? Is it Protestant, Catholic, Jewish, some other religion, or no religion?”

To graph this, we want a bar chart with *religion* on the x axis (as a categorical variable), and with the bars in the chart also colored by religion. We also want to fill the bars with color, so we can map the religion variable to fill, in addition to mapping it to x. Remember, fill is for painting the insides of shapes. The result in displayed in Figure 6.5.

p <- ggplot(data = gss\_sm, mapping = aes(x = religion, fill = religion))  
p +  
geom\_bar()

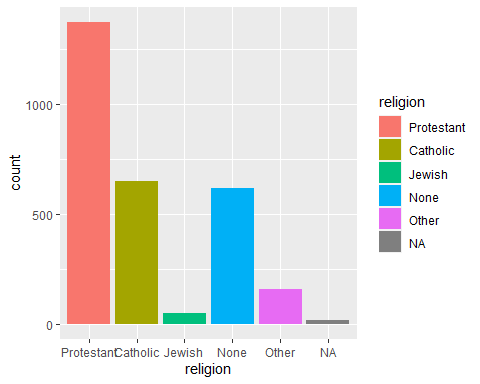


Figure 6.5: Religion data using fil, which maps to the inside of the shapes.

We have now mapped two *aesthetics* to the same variable. Both x and fill are mapped to religion. There is nothing wrong with this. However, these are still two separate mappings, and so they get two separate scales [Figure 6.5].

We can correct this by using the guides() function. In its simplest use, the guides() function controls whether guiding information about any particular mapping appears or not. If we set guides(fill = FALSE), the legend is removed. Setting the guide for some mapping to FALSE has an effect only if there is a legend to turn off to begin with.

p <- ggplot(data = gss\_sm, mapping = aes(x = religion, fill = religion))  
p +  
geom\_bar() +  
 guides(fill = FALSE)

## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =  
## "none")` instead.

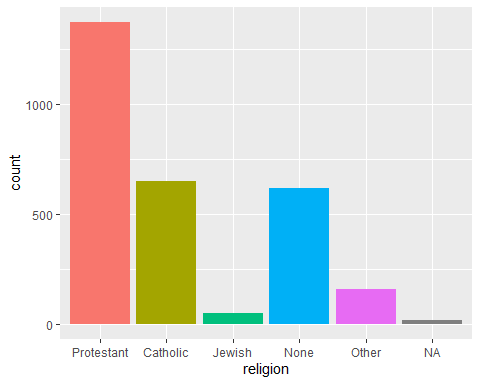


Figure 6.6: Religion data using fil, which maps to the inside of the shapes, without a legend using guides(fill = FALSE).

An interesting thing happened when we produced this graph. This error message appeared in the console:

“Warning: guides(<scale> = FALSE) is deprecated. Please use guides(<scale> = "none") instead.” This means that when we see/read guides(fill = FALSE) in the Healy text, we need to replace this with guides(fill = "none"). So, let us try this again and see if it works.

p <- ggplot(data = gss\_sm, mapping = aes(x = religion, fill = religion))  
p +  
geom\_bar() +  
 guides(fill = "none")

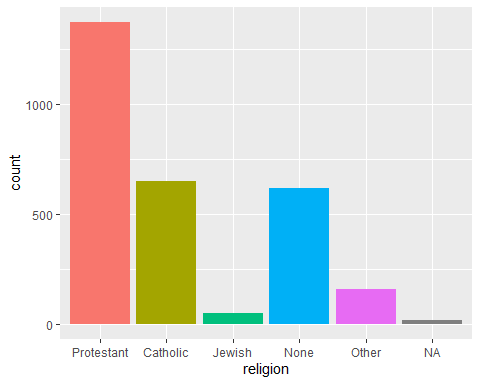


Figure 6.7: Religion data using fil, which maps to the inside of the shapes, without a legend using guides(fill = "none").

As a refresher, what would happy if we map religion to color? As we have already seen, only the border lines of the bars will be assigned colors, and the insides will remain gray. Figure 6.8 displays the results of that choice.

p <- ggplot(data = gss\_sm, mapping = aes(x = religion, color = religion))  
p +  
geom\_bar() +  
 guides(fill = "none")

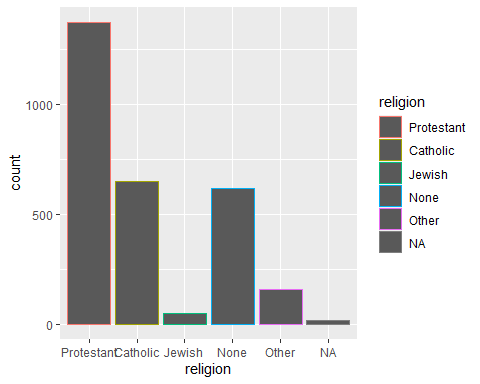


Figure 6.8: Religion data using color, which only maps to the borders of the shapes.

# Avoid Transformations When Necessary

As we have seen, ggplot normally makes its charts starting from a full dataset. When we call geom\_bar() it does its calculations on the fly using stat\_count() behind the scenes to produce the counts or proportions it displays.

But often our data is already a summary table. This can happen when we have computed a table of marginal frequencies or percentages from the original data. Let us look at the titanic dataset as an example:

titanic

## fate sex n percent  
## 1 perished male 1364 62.0  
## 2 perished female 126 5.7  
## 3 survived male 367 16.7  
## 4 survived female 344 15.6

Because we are working directly with percentage values in a summary table, we no longer have any need for ggplot to count up values for us or perform any other calculations. That is, we do not need the services of any stat\_ functions that geom\_bar() would normally call.

We can tell geom\_bar() not to do any work on the variable before plotting it. To do this we say stat = 'identity' in the geom\_bar() call [Figure 6.9].

p <- ggplot(data = titanic, mapping = aes(x = fate,   
 y = percent, fill = sex))  
  
p +  
 geom\_bar(position = "dodge", stat = "identity") +  
 theme(legend.position = "top")

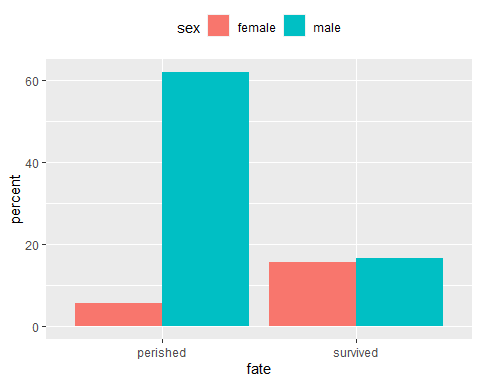
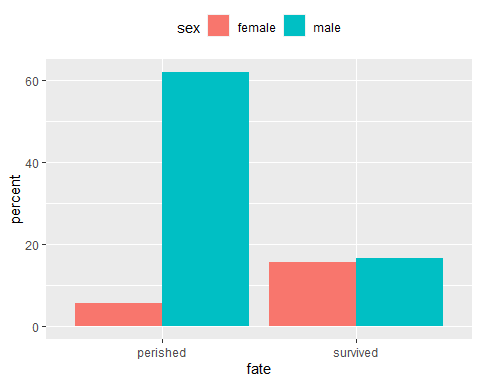


Figure 6.9: Visualizing Titanic data.

You will notice we used theme(legend.position = "top") to move the legend to the top of the graph.

gplot also provides geom\_col(), which has exactly the same effect but assumes that stat = "identity.” We will use this form in the future when we don’t need any calculations done on the plot.

p <- ggplot(data = titanic, mapping = aes(x = fate,   
 y = percent, fill = sex))  
  
p +  
 geom\_col(position = "dodge") +  
 theme(legend.position = "top")



The position argument in geom\_bar() and geom\_col() can also take the value of ” identity.” Just as stat = "identity" means “don’t do any summary calculations;’ position = "identity" means”just plot the values as given’ This allows us to do things like plotting a flow of positive and negative values in a bar chart. This sort of graph is an alternative to a line plot, and is often seen in public policy settings where changes relative to some threshold level or baseline are of interest.

Let us look at an example using the *oecd\_sum* table in *socviz,* which contains information on average life expectancy at birth within the United States and across other OECD countries. Let us first take a peek at the data

oecd\_sum

## # A tibble: 57 x 5  
## # Groups: year [57]  
## year other usa diff hi\_lo  
## <int> <dbl> <dbl> <dbl> <chr>  
## 1 1960 68.6 69.9 1.30 Below  
## 2 1961 69.2 70.4 1.20 Below  
## 3 1962 68.9 70.2 1.30 Below  
## 4 1963 69.1 70 0.900 Below  
## 5 1964 69.5 70.3 0.800 Below  
## 6 1965 69.6 70.3 0.700 Below  
## 7 1966 69.9 70.3 0.400 Below  
## 8 1967 70.1 70.7 0.600 Below  
## 9 1968 70.1 70.4 0.300 Below  
## 10 1969 70.1 70.6 0.5 Below  
## # ... with 47 more rows

OK, what do each of these columns mean? The *other* column is the average life expectancy in a given year for OECD countries, excluding the United States. The *usa* column is the U.S. life expectancy, *diff* is the difference between the two values, and *hi\_lo* indicates whether the U.S. value for that year was above or below the OECD average.

Figure 6.10 plots the difference over time and use the hi\_lo variable to color the columns in the chart.

p <- ggplot(data = oecd\_sum, mapping = aes(x = year, y = diff, fill = hi\_lo))  
p +  
 geom\_col() +  
 guides(fill = "none") +  
 labs(x = NULL, y = "Difference in Years",  
 title = "The US Life Expectancy Gap",  
 subtitle = "Difference between US and OECD average life expectancies, 1960-2015",  
 caption = "Data: OECD. After a chart by Christopher Ingraham,  
 Washington Post, December 27th 2017.")

## Warning: Removed 1 rows containing missing values (position\_stack).

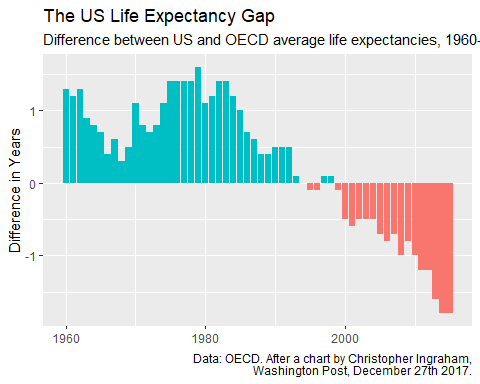


Figure 6.10: Difference over time by hi\_lo.

# Refining Plots

So far we used ggplot’s default output when making our plots. In general, when making figures during exploratory data analysis, the default settings in ggplot should work. It’s only when we have some specific plot in mind that the question of polishing the results comes up.

Refining a plot can mean several things (Healy, 2018a):

1. We might want customize based on personal tastes.
2. We might want to format it in a way that will meet the expectations of a journal, or a conference presentation

## Figure Titles

Every figure needs a title. The job of the title is to accurately convey to the reader what the figure is about (Wilke, 2019). However, the figure title may not necessarily appear where you were expecting to see it. For example look at FIgure 6.12, where the title appears as a caption at the bottom of the graph.

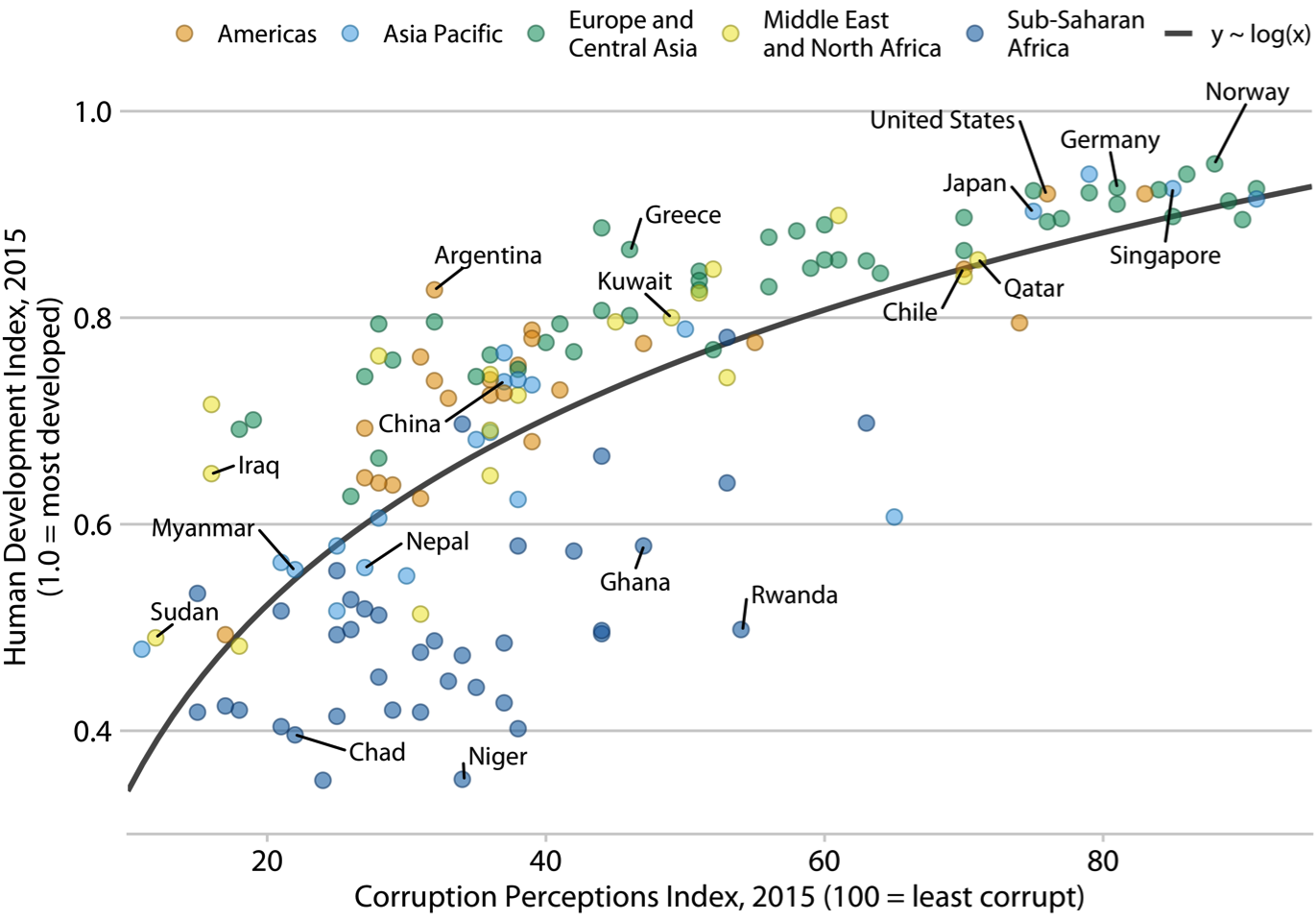


Figure 6.12: Corruption and human development: The most developed countries experience the least corruption. This figure was inspired by a posting in The Economist online ([2011](https://clauswilke.com/dataviz/figure-titles-captions.html#ref-Economist-corruption)). Data sources: Transparency International & UN Human Development Report

Alternatively, we incorporate the figure title - as well as other elements of the caption, such as the data source statement-into the main display (Figure 6.13) (Wilke, 2019).

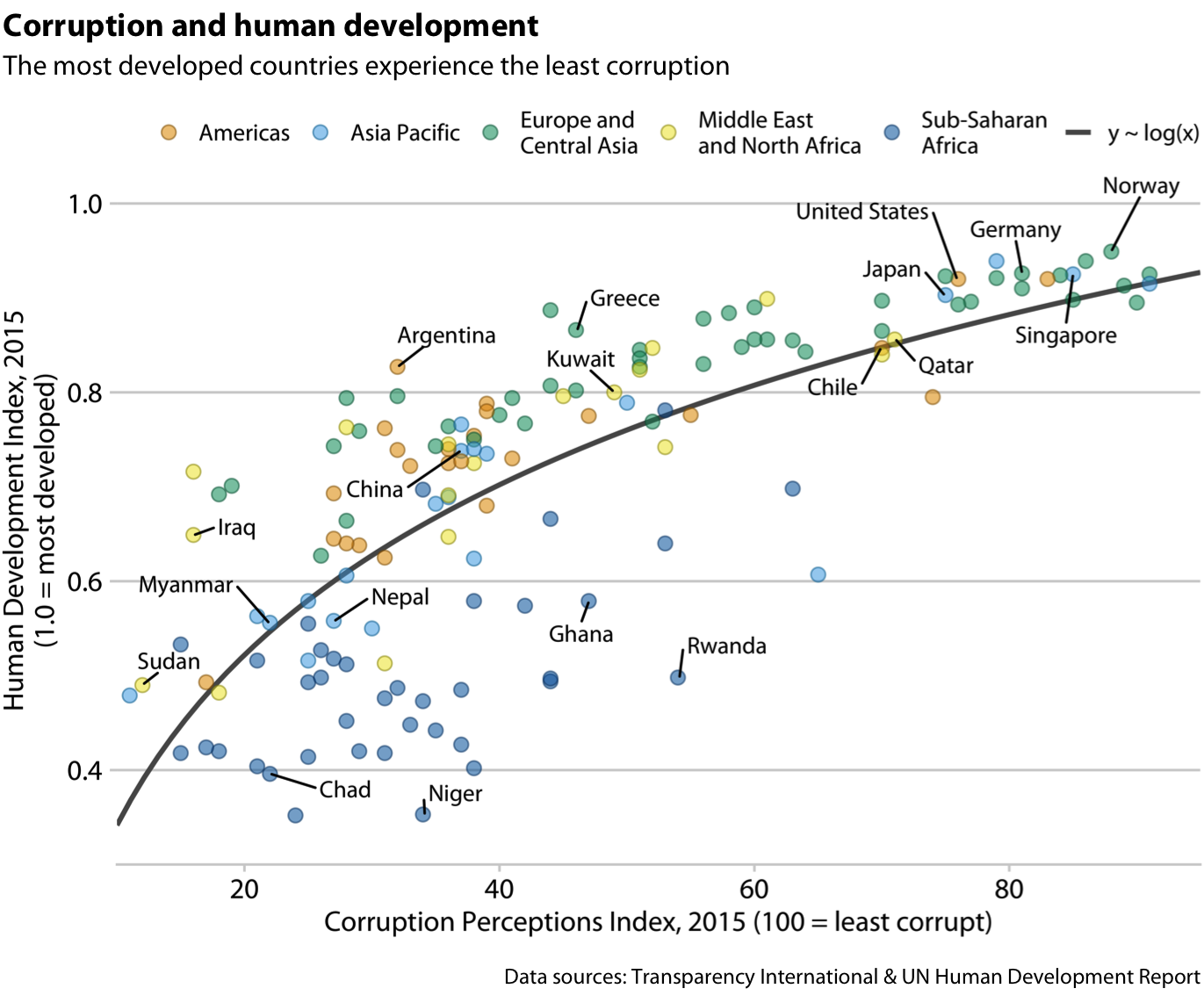


Figure 6.13: Infographic version of Figure 6.12.

You might be wondering why I choose to use the latter style throughout this book. I do so because the two styles have different application areas, and figures with integrated titles are not appropriate for conventional book layouts (Wilke, 2019). The underlying principle is that a figure can have only one title. Either the title is integrated into the actual figure display or it is provided as the first element of the caption underneath the figure (Wilke, 2019).

Tip: If your document layout uses caption blocks underneath each figure, then place the figure titles as the first element of each caption block, not on top of the figures (Wilke, 2019).

## Axis and Legend Titles

Axis titles are often referred to as axis labels (Wilke, 2019). Axis and legend titles and labels explain what the displayed data values are and how they map to plot aesthetics (**1917?**).

Figure 6.14, is an example of a plot where all axes and legends are appropriately labeled and titled. This graph was created from the the blue jay dataset, which is discussed Chapter 12.

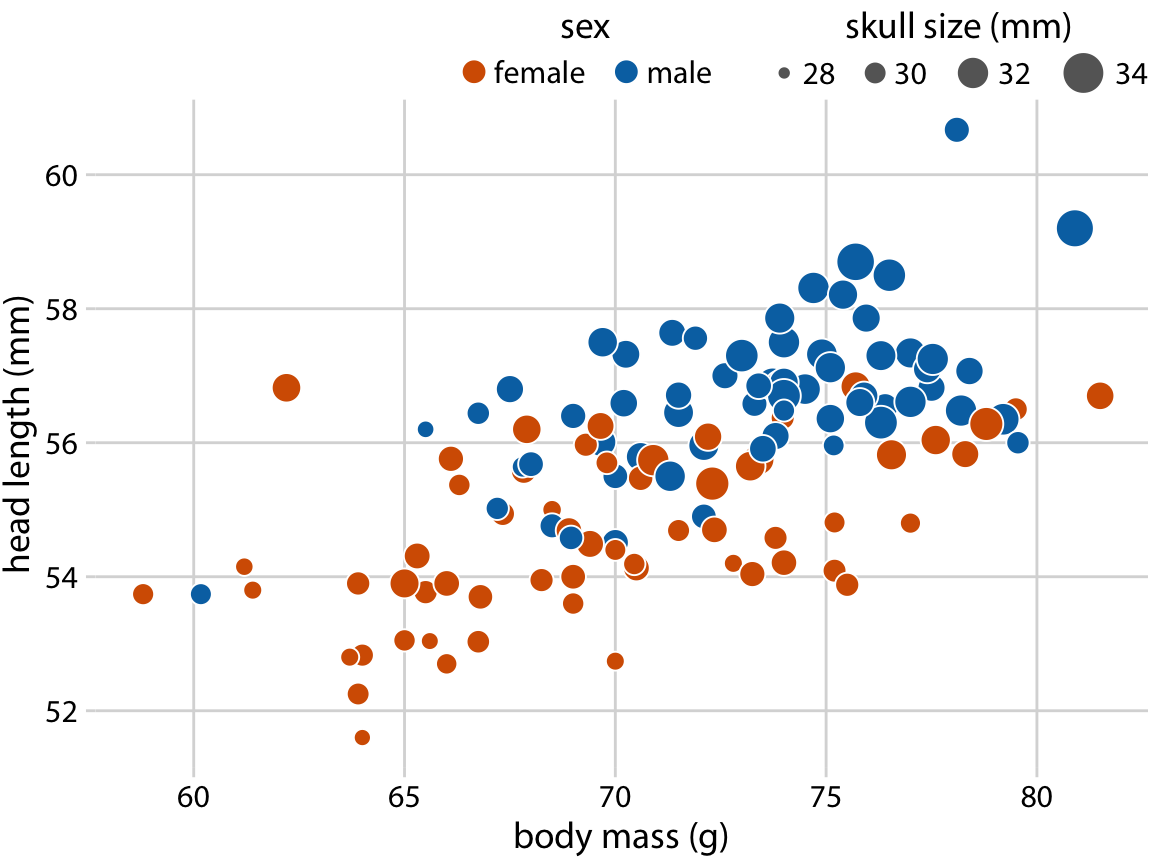


Figure 6.14: Head length versus body mass for 123 blue jays. Data source: Keith Tarvin, Oberlin College.

In Figure 6.14 the birds’ sex is indicated by color, and the birds’ skull size by symbol size (Inc., 2020). Head length measurements include the length of the bill while skull size measurements do not (Wilke, 2019).

In Figure 6.14, the axis titles indicate that the x axis shows body mass in grams and they axis shows head length in millimeters (Wilke, 2019). Similarly, the legend titles show that point coloring indicates the birds’ sex and point size indicates the birds’ skull size in millimeters (Wilke, 2019).

For this particular example (Figure 6.14), WIlke emphasize that for all numerical variables (body mass, head length, and skull size) the relevant titles not only state the **variables show**n but also the **units in which the variables** are measured. This is good practice and should be done whenever possible. Categorical variables (such as sex) do not require units.

However, there are cases when axis or legend titles can be omitted, namely when the labels themselves are fully explanatory (Wilke, 2019). For example, a legend showing two differently colored dots labeled “female” and “male” already indicates that color encodes sex. We have already seen this in Chapter 6. I am including the figure below (Figure 6.15).

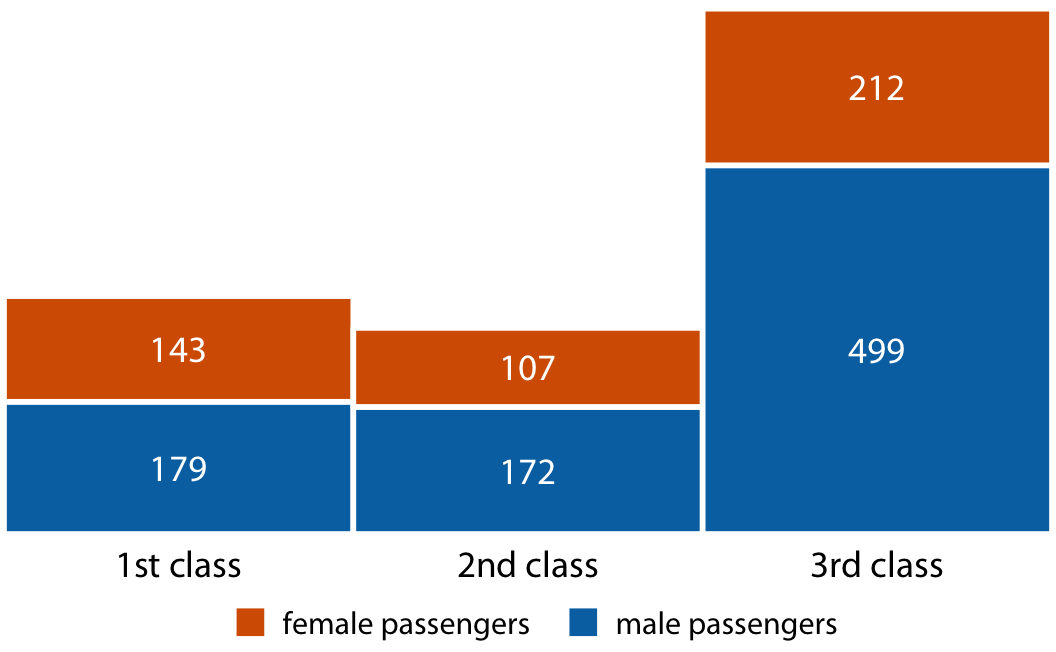


Figure 6.15: Numbers of female and male passengers on the Titanic traveling in 1st, 2nd, and 3rd class.

## Refining Plot Examples

Let’s begin by looking at a new dataset, asasec. This contains data on membership in the American Sociological Association. Let us display the first 5 rows of data

head(asasec,5)

## Section Sname Beginning Revenues  
## 1 Aging and the Life Course (018) Aging 12752 12104  
## 2 Alcohol, Drugs and Tobacco (030) Alcohol/Drugs 11933 1144  
## 3 Altruism and Social Solidarity (047) Altruism 1139 1862  
## 4 Animals and Society (042) Animals 473 820  
## 5 Asia/Asian America (024) Asia 9056 2116  
## Expenses Ending Journal Year Members  
## 1 12007 12849 No 2005 598  
## 2 400 12677 No 2005 301  
## 3 1875 1126 No 2005 NA  
## 4 1116 177 No 2005 209  
## 5 1710 9462 No 2005 365

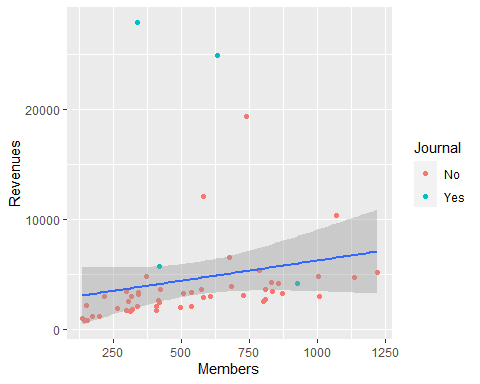
This dataset has membership data for each section over a ten-year period, but the data on section reserves and income (the Beginning and Revenues variables) is for 2015 only. Let’s look at the relationship between section membership and section revenues for a single year, 2014. Let us also add a smooth line to the graph. This plot is going to get complicated really quickly. So, we are going to do this in stages, all saving to “p(x).”

p1 <- ggplot(data = subset(asasec, Year == 2014),  
 mapping = aes(x = Members, y = Revenues, label = Sname)) + geom\_point(mapping = aes(color = Journal)) +  
 geom\_smooth(method = "lm")

Let us take a peek at the graph that we just created:

p1

## `geom\_smooth()` using formula 'y ~ x'



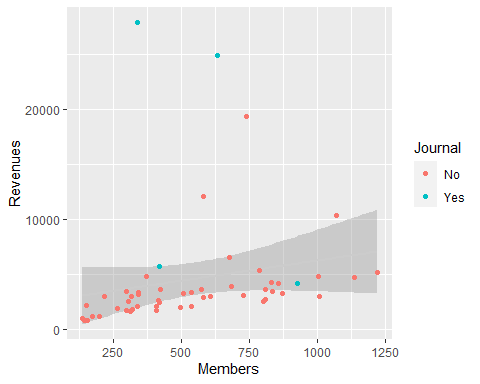
As you can see from the figure above, This is our basic scatterplot-and-smoother graph. To refine it, let’s begin by identifying some outliers, switch from [loess](https://ggplot2.tidyverse.org/reference/geom_smooth.html) to OLS, and introduce a third variable.

p2 <- p1 + geom\_smooth(method = "lm", se = FALSE, color = "gray80") +  
 geom\_point(mapping = aes(color = Journal))

Let us take a peek at the graph that we just created:

p2

## `geom\_smooth()` using formula 'y ~ x'  
## `geom\_smooth()` using formula 'y ~ x'



For the next section, we are going to use [ggrepel](https://ggrepel.slowkow.com/articles/examples.html), which provides geoms for ggplot2 to repel overlapping text labels. We turned on this library before we started the lab. In particular, we will add some text labels using geom\_text\_repel.

The figure below illustrates the differences between geom\_text() and geom\_text\_repel():

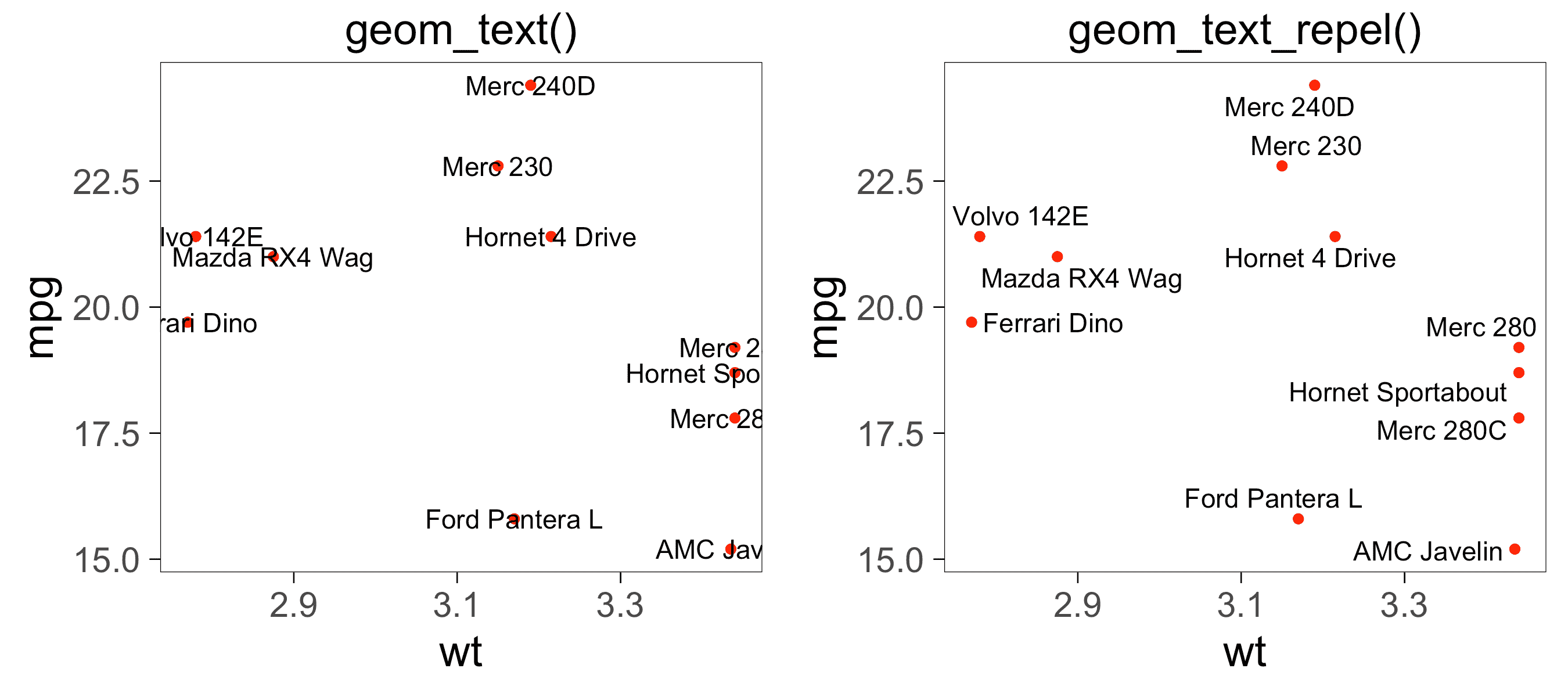


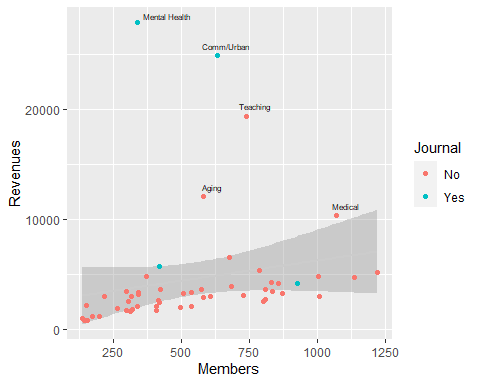
Figure 6.11: Differences between geom\_text and geom\_text\_repel.

p3 <- p2 + geom\_text\_repel(data=subset(asasec, Year == 2014 & Revenues > 7000), size = 2)

Let us take a peek at the graph that we just created:

p3

## `geom\_smooth()` using formula 'y ~ x'  
## `geom\_smooth()` using formula 'y ~ x'



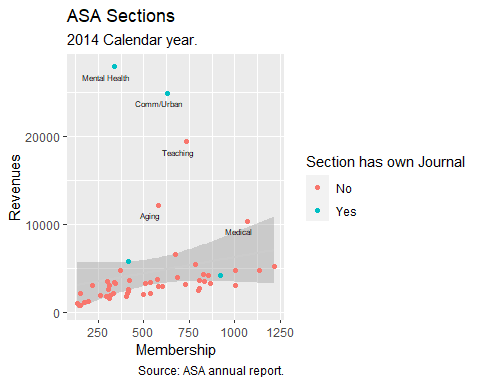
In the next step we are going to add labels to the y-axis, add a legend, title, subtitle, caption.

p4 <- p3 + labs(x="Membership",  
 y="Revenues",  
 color = "Section has own Journal",  
 title = "ASA Sections",  
 subtitle = "2014 Calendar year.",  
 caption = "Source: ASA annual report.")

Let us take a peek at the graph that we just created:

p4

## `geom\_smooth()` using formula 'y ~ x'  
## `geom\_smooth()` using formula 'y ~ x'



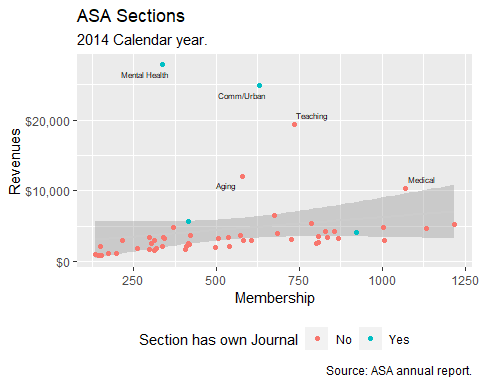
For the last portion of the graph we will explore the use of scale\_x\_continuous() and scale\_y\_continuous() are the default scales for [continuous x and y aesthetics](https://ggplot2.tidyverse.org/reference/scale_continuous.html). There are three variants that set the trans argument for commonly used transformations: scale\_\*\_log10(), scale\_\*\_sqrt() and scale\_\*\_reverse(). We are also using [labels](https://scales.r-lib.org/reference/label_dollar.html#examples)to have the y-axis label show as currency. We also moved the legend to make better use of the space in the plot

p5 <- p4 + scale\_y\_continuous(labels = scales::dollar) +  
 theme(legend.position = "bottom")

Now, let us look at the final graph:

p5

## `geom\_smooth()` using formula 'y ~ x'  
## `geom\_smooth()` using formula 'y ~ x'



It is OK if these formatting options see a little overwhelming. You can also refer to the R-documentation for help.

# Using Color to Your Advantage

A choice of color palette should be based on its ability to express the data you are plotting.

An unordered categorical variable like “country” or “sex’ requires distinct colors that won’t be easily confused with one another. An ordered categorical variable like”level of education’ requires a graded color scheme of some kind running from less to more or earlier to later.

There are other considerations. For example, if your variable is ordered, is your scale centered on a neutral midpoint with departures to extremes in each direction, as in a Likert scale?

Take care to choose a palette that reflects the structure of your data. For example, do not map sequential scales to categorical palettes, or use a diverging palette for a variable with no well-defined midpoint.

## RColorBrewer

We choose color palettes for mappings through one of the scale\_ functions for color or fill. While it is possible to very finely control the look of your color schemes by varying the hue, chroma, and luminance of each color you use via scale\_color\_hue(), or scale\_fill\_hue(), in general this is not recommended. Instead you should use the RColorBrewer package to make a wide range of named color palettes available to you, and choose from those.

The [RColorBrewer](file:///C:\Users\doujo\Documents\lsc-563-joubert\lecture_rmd\ggplot2.tidyverse.org\reference\scale_brewer.html) package provides a wide range of named color palettes to choose from. The nice thing about RColorBrewer is that it will show you all of the palettes in a graphics window. To make this work type the follow code in the Console:

display.brewer.all()

Let us run this, and then save the plot as a PDF. Plots>Export>Save as PDF. Use the settings listed in Figure 6.12. You are going to need this image for the lab.

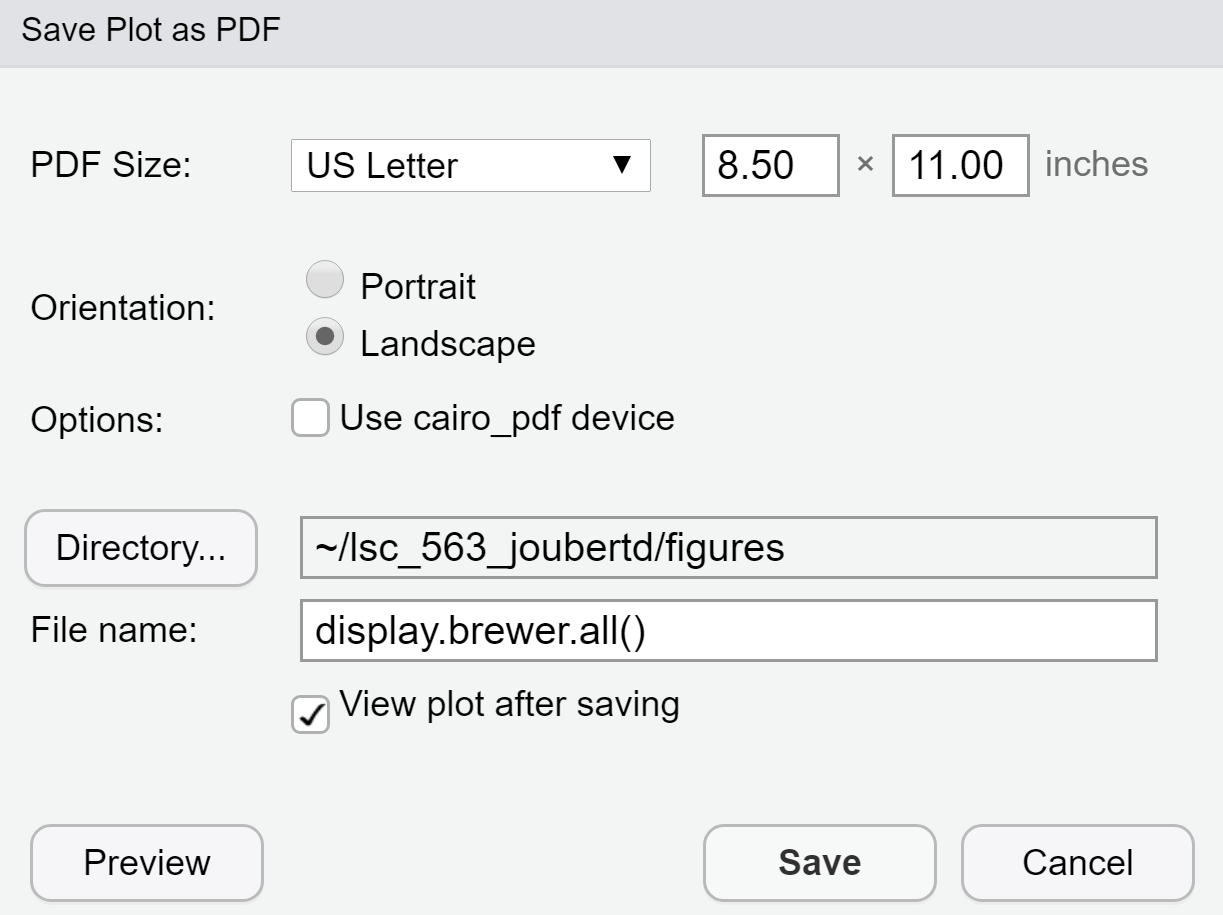


Figure 6.12: Settings for exporting an image as a PDF.

There are 3 types of Color Brewer palettes:

1. Sequential palettes are suited to ordered data that progress from low to high. Lightness steps dominate the look of these schemes, with light colors for low data values to dark colors for high data values.

2. Qualitative palettes do not imply magnitude differences between legend classes. Qualitative schemes are best suited to representing nominal or categorical data

3. Diverging palettes put equal emphasis on mid-range critical values and extremes at both ends of the data range. The critical class or break in the middle of the legend is emphasized with light colors and low and high extremes are emphasized with dark colors that have contrasting hues.

Figures Figure 6.12 show the available options for sequential variables.

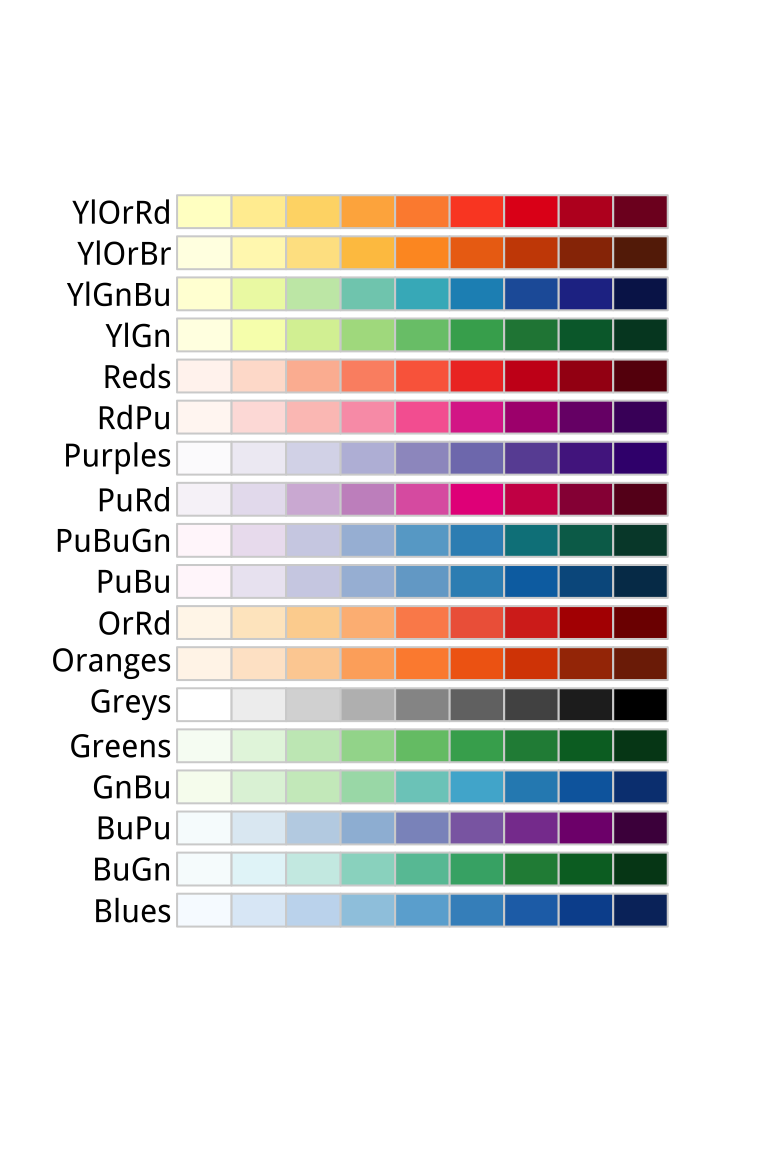


Figure 6.12 RColorBrewer’s sequential palettes.

Figure 6.13 is displaying RColorBrewer’s diverging palettes.

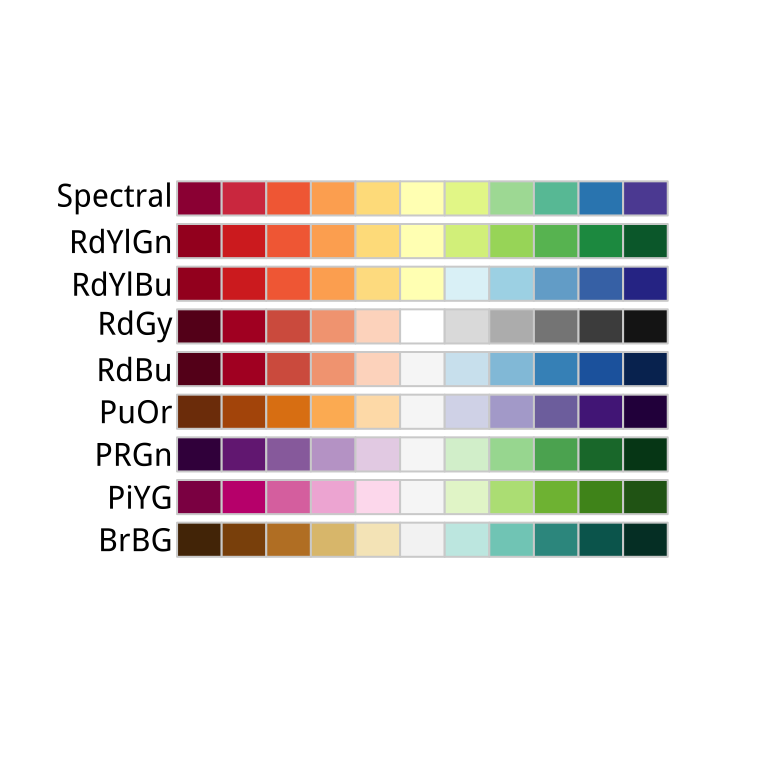


Figure 6.13: RColorBrewer’s diverging palettes.

Last, Figure 6.14 is displaying RColorBrewer’s qualitative palettes.



Figure 6.14: RColorBrewer’s qualitative palettes.

When used in conjunction with ggplot, you access these colors by specifying the scale\_color\_brewer() or scale\_fill\_brewer() functions, depending on the aesthetic you are mapping.

Let us look at some examples using the organ donor data.

organdata

## # A tibble: 238 x 21  
## country year donors pop pop\_dens gdp gdp\_lag health health\_lag  
## <chr> <date> <dbl> <int> <dbl> <int> <int> <dbl> <dbl>  
## 1 Australia NA NA 17065 0.220 16774 16591 1300 1224  
## 2 Australia 1991-01-01 12.1 17284 0.223 17171 16774 1379 1300  
## 3 Australia 1992-01-01 12.4 17495 0.226 17914 17171 1455 1379  
## 4 Australia 1993-01-01 12.5 17667 0.228 18883 17914 1540 1455  
## 5 Australia 1994-01-01 10.2 17855 0.231 19849 18883 1626 1540  
## 6 Australia 1995-01-01 10.2 18072 0.233 21079 19849 1737 1626  
## 7 Australia 1996-01-01 10.6 18311 0.237 21923 21079 1846 1737  
## 8 Australia 1997-01-01 10.3 18518 0.239 22961 21923 1948 1846  
## 9 Australia 1998-01-01 10.5 18711 0.242 24148 22961 2077 1948  
## 10 Australia 1999-01-01 8.67 18926 0.244 25445 24148 2231 2077  
## # ... with 228 more rows, and 12 more variables: pubhealth <dbl>, roads <dbl>,  
## # cerebvas <int>, assault <int>, external <int>, txp\_pop <dbl>, world <chr>,  
## # opt <chr>, consent\_law <chr>, consent\_practice <chr>, consistent <chr>,  
## # ccode <chr>

p <- organdata %>%   
 ggplot(mapping = aes(x = roads, y = donors, color = world))  
  
p +  
 geom\_point(size = 2)+  
 scale\_color\_brewer(palette = "Set2") +  
 theme(legend.position = "top")

## Warning: Removed 46 rows containing missing values (geom\_point).

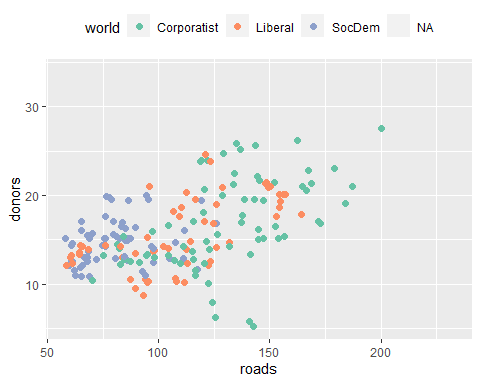


Figure 6.15: Organ data using one of the qualitative palettes (Set2).

We will explore some of the other palettes below.

You can also specify colors manually, via scale\_color\_manual() or scale\_fill\_manual(). These functions take a value argument that can be specified as vector of color names or color values that R knows about. R knows many color names (like red, and green, and cornflowerblue. Try demo('colors') for an overview. Alternatively, color values can be specified via their hexadecimal RGB value. This is a way of encoding color values in the RGB colorspace, where each channel can take a value from 0 to 255. A color hex value begins with a hash or pound character, #, followed by three pairs of hexadecimal or “hex” numbers.

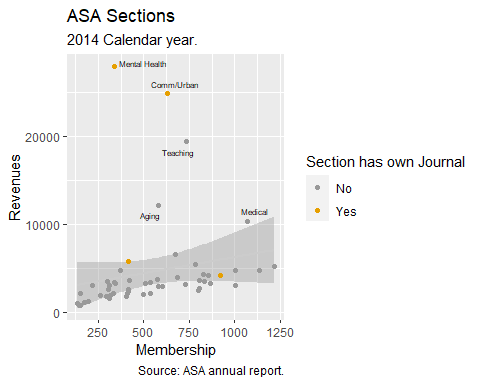
Let us look at an example using some CVD palettes created by (Chang, 2021).

cb\_palette <- c("#999999", "#E69F00", "#56B4E9", "#009E73",  
 "#F0E442", "#0072B2", "#D55E00", "#CC79A7")

Let us talk about what is going on in this statement. cb\_palette is a vector that contain characters. The quotes around “#999999,” “#E69F00,” etc. are essential here. Without the quotes R will assume there are objects called #E69F00, and #999999. As these objects don’t exist in R’s memory, there will be an error message if we do not include the quotes. Now let us use this in a graph, using the p4 graph that we previously created.

p4 + scale\_color\_manual(values = cb\_palette)

## `geom\_smooth()` using formula 'y ~ x'  
## `geom\_smooth()` using formula 'y ~ x'



There are specific packages that include safe palettes for color-blind viewers. Examples include the dichromat, colorblindr, and viridispackages.

The viridiscolor scales come with the current version of ggplot2 (3.0.0). To see more on the different viridis palettes, see ?scales::viridis\_pal. The cetcolor scales: <https://github.com/coatless/cetcolor>.

The Color Oracle program ([<http://colororacle.org>](http://colororacle.org/)) can simulate how things on your screen appear to someone with color vision deficiency, but keep in mind that the simulation isn’t perfect.

# Using Themes

The ggplot library comes with several built-in themes, including theme\_minimal() and theme\_classic(), with theme\_gray() or theme\_grey() as the default. If these are not to your taste, install the ggthemes library for many more options. You can, for example, make ggplot output look like it has been featured in the *Economist*, or the *Wall Street Journal.*

The theme() function allows you to exert very fine-grained control over the appearance of all kinds of text and graphical elements in a plot. For example, we can change the color, typeface, and font weight of text. Let us try two different themes on our p4 plot:

theme\_set(theme\_bw())  
p4 + theme(legend.position="top")

## `geom\_smooth()` using formula 'y ~ x'  
## `geom\_smooth()` using formula 'y ~ x'

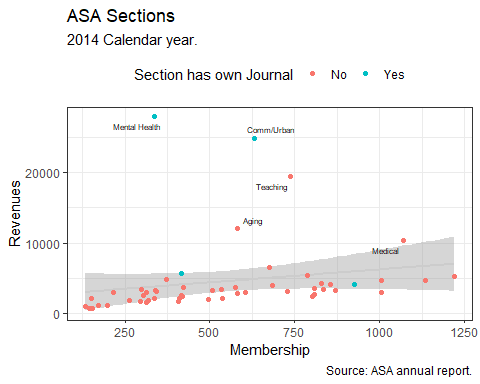


Figure 6.16: Membership data data using the black and white theme.

theme\_set(theme\_dark())  
p4 + theme(legend.position="top")

## `geom\_smooth()` using formula 'y ~ x'  
## `geom\_smooth()` using formula 'y ~ x'

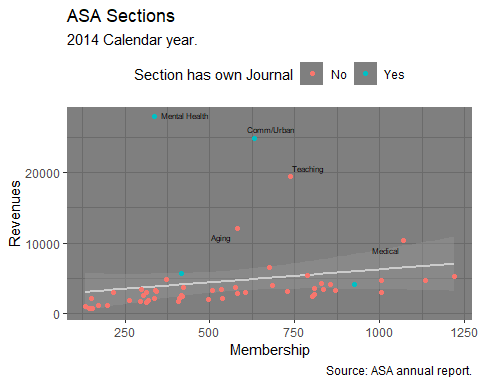


Figure 6.17: Membership data data using the dark theme.

# References

Chang, W. (2021). Using colors in plots [Book Section]. In *R graphics cookbook : Practical recipes for visualizing data* (2nd ed.). O’Reilly. <https://r-graphics.org/chapter-colors>

Healy, K. (2018a). Refine your plots [Book Section]. In *Data visualization: A practical introduction* (pp. 199–230). Princeton University Press.

Healy, K. (2018b). Show the right numbers [Book Section]. In *Data visualization: A practical introduction* (pp. 73–92). Princeton University Press.

Inc., S. I. (2020). *JMP®14* [Computer Program]. SAS Institute Inc.

Wilke, C. (2019). Titles, captions, and tables [Book Section]. In *Fundamentals of data visualization: A primer on making informative and compelling figures* (First edition., pp. 267–276). O’Reilly Media.