ggplot2 Workshop

Francie McQuarrie

4/7/2021

Preliminaries

If you want to follow along with the code in the provided R notebook, set your working directory in your console with

```
setwd("/path/to/folder/that/contains/these/materials")
```

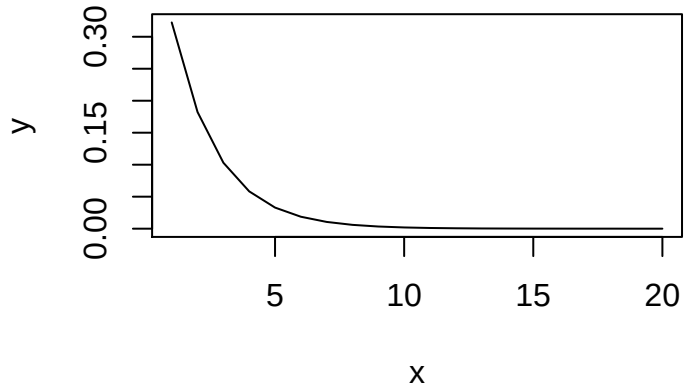
Why ggplot2?

Today's workshop will explore `ggplot2`, a plotting system in `R`.

Why do we need to learn a separate plotting library when `R` has built in plotting functions?

The base `R` plotting functions may be sufficient, albeit ugly, for quick homework plots or confirming some small assumption about data:

```
# Example base R plot
x <- 1:20
y <- 0.57*exp(-0.57*x)
plot(x, y, type = 'l')
```



However, if

- your plots need to be prettier/more professional/more colorful/more elegant than the above
- your plots need to be heavily customizable and versatile based on data
- your data has a complicated structure

then base `R` is no longer sufficient and `ggplot2` becomes necessary.

Introduction

`ggplot2` has a grammatical structure, just like any language [Ram]. Specifically, it implements the **grammar of graphics**, a "coherent system for describing and building graphs" [R for Data Science]. The grammar of `ggplot2` defines the components that makes up a plot, just like how English grammar defines components

that constitute a sentence. After learning the basic structure, you can quickly apply it for many places and contexts. Just like after learning a programming language for the first time, it's easier to learn other languages. The basic concepts like functions, loops, and conditionals are already understood, and you only need to adapt to different syntax.

Note: The term "Grammar of Graphics" was originally coined by Lee Wilkinson, and the theory applied towards `ggplot2` by Hadley Wickham can be found here.

Basic Structure

To begin, install the package and load it:

```
# Uncomment next line if running on own computer
# install.packages("ggplot2")  # only do once per machine
library("ggplot2")
```

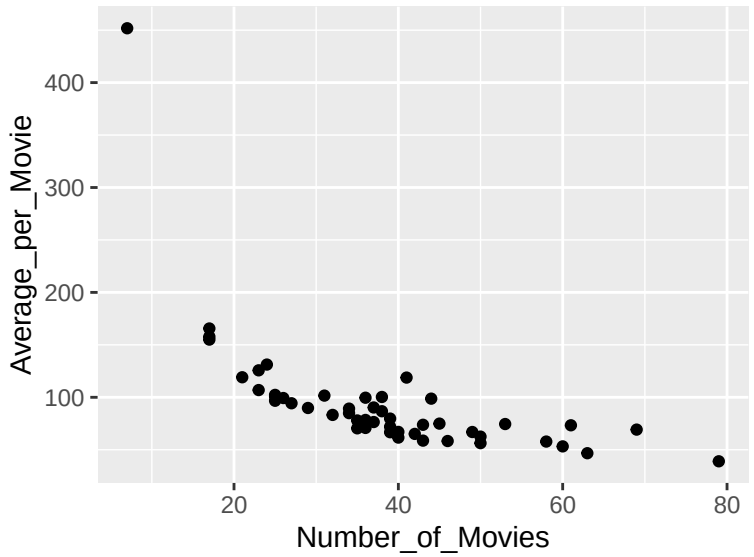
We begin with the function `ggplot()` (note the no "2" - why? See Freeman below). This creates the base empty graph tied to the dataset. Then, add one or more "layers" to the plot depending on your desired output. This sounds opaque, so let's practice on an example dataset. Instead of the usual `mpg` cars or `iris` dataset, we'll be investigating other datasets like top movies, actors and smoking mothers.

The following table displays information about the top 50 grossing actors. It's already sorted - Harrison Ford is the top grossing actor.

```
actors <- read.csv("actors.csv",
col.names = c('Actor', 'Total_Gross', 'Number_of_Movies', 'Average_per_Movie', 'Top_Movie', 'Gross'))
```

Let's investigate the relationship between the number of movies these actors have starred in versus the average gross domestic box office receipt (in millions of dollars) for all of the actor's movies:

```
ggplot(data = actors) +
  geom_point(mapping = aes(x = Number_of_Movies, y = Average_per_Movie))
```



The above call added a layer of points to the base graph to create a scatterplot. The function call used a `mapping` argument in the second layer. This specifies how variables in your dataset are mapped to visual properties in the graph. In this case, the `Number_of_Movies` variable is mapped to the x-axis and the `Avergage_per_Movie` is mapped to the y-axis. We'll explore other things that can be mapped besides the axes later.

You don't have to always specify the `mapping =` in front of `aes`, but you always have to specify the word `aes` with some argument.

Let's define a start template on how to structure plotting code:

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

To implement this template in your own code, replace the bracket sections with the correct input (dataset, geom function, or collection of mappings). **Important**: The `+` must always come at the end of the line, not the start.

Exercise: The plot above has two extreme outliers. In the cell below, display a two-row table whose rows correspond to the two actors with highest and lowest number of movies, respectively. If you don't recognize one of the names, google for a surprise!

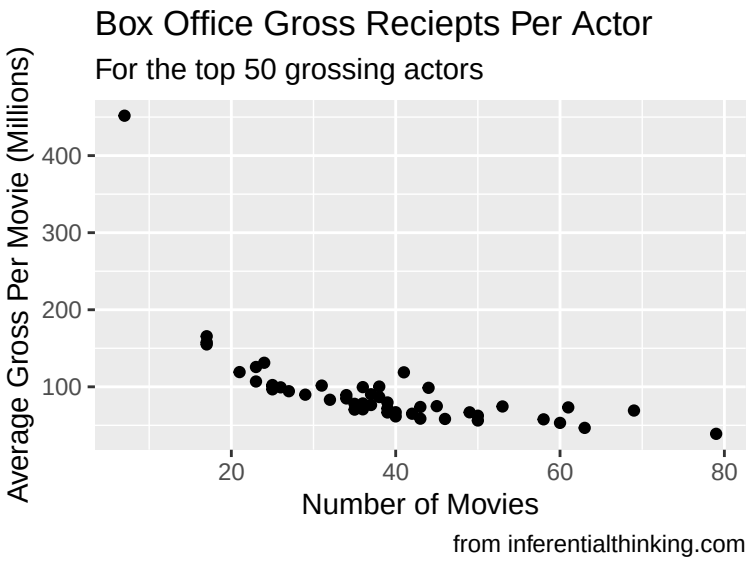
```
# Inserttable exploration here
```

Aesthetics

There are many ways to customize the appearance of plots by tweaking small bits of the call above. Specifically, we want to explore changing the **aesthetics** of the graph, which are visual properties of the objects in the plot. The following examples will be demonstrated on the scatterplot, but they can be specified for any type of plot.

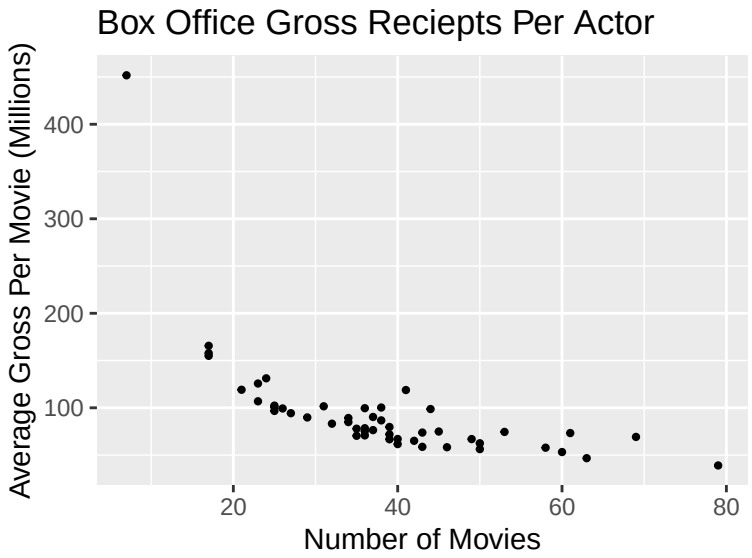
There's one glaring issue with the plot above that any statistician or data scientist should cringe at: the badly formatted axes labels and missing figure title. Always label your plots! Correct that using built-in `ggplot2` functions:

```
ggplot(data = actors) +
  geom_point(mapping = aes(x = Number_of_Movies, y = Average_per_Movie)) +
  labs(x = 'Number of Movies', y = 'Average Gross Per Movie (Millions)',
       title = 'Box Office Gross Reciepts Per Actor',
       subtitle = 'For the top 50 grossing actors',
       caption = 'from inferentialthinking.com')
```



What if we want to change the size of the points? Do so with `size`:

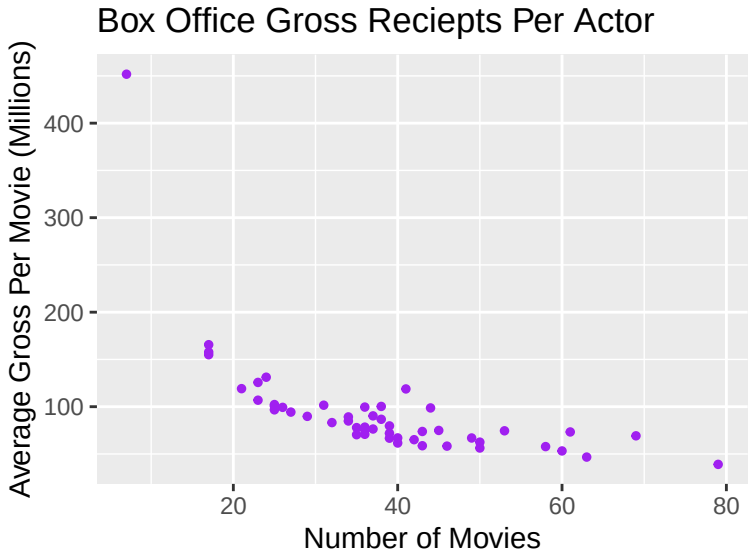
```
# Size is specified in mm
ggplot(data = actors) +
  geom_point(mapping = aes(x = Number_of_Movies, y = Average_per_Movie),
             size = 0.8) +
  labs(x = 'Number of Movies', y = 'Average Gross Per Movie (Millions)',
       title = 'Box Office Gross Reciepts Per Actor')
```



If we are *manually* changing a visual part of the graph, we need to specify it *outside* the `aes()` function (more on this later!).

What if we wanted to change the color of the points? Do so with `color`:

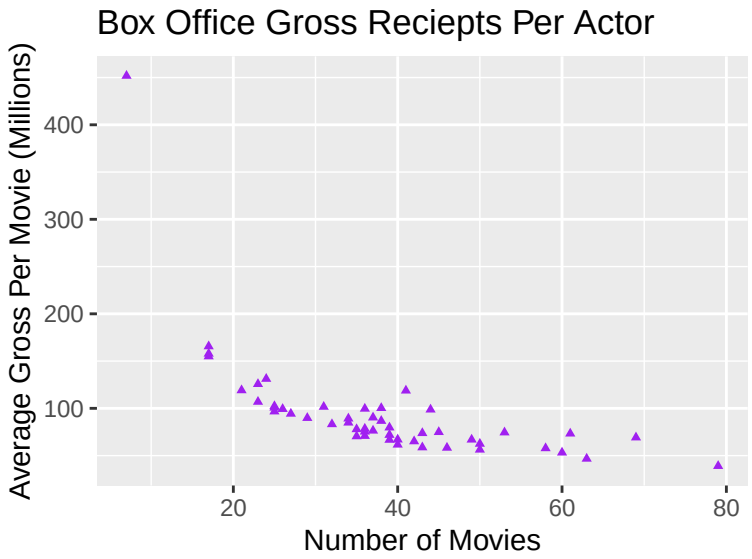
```
ggplot(data = actors) +
  geom_point(mapping = aes(x = Number_of_Movies, y = Average_per_Movie),
             size = 1, color = 'purple') +
  labs(x = 'Number of Movies', y = 'Average Gross Per Movie (Millions)',
       title = 'Box Office Gross Reciepts Per Actor')
```



What if we wanted to change the shape of the points? Scatterplots don't always have to display circular

points! Do so with `shape`:

```
ggplot(data = actors) +
  geom_point(mapping = aes(x = Number_of_Movies, y = Average_per_Movie),
             size = 1, color = 'purple', shape = 17) +
  labs(x = 'Number of Movies', y = 'Average Gross Per Movie (Millions)',
       title = 'Box Office Gross Reciepts Per Actor')
```



The possible shapes of points are specified by numbers. See here for a complete chart of numbers and shapes.

Exercise: Replicate the above graph such that the markers are size 1.3mm, red, and square shaped.

```
# insert code here
```

Everything we did above seems possible in base R graphics (albeit with somewhat clunkier code). But what if you wanted to change the aesthetics of the points based on another *variable* in your dataset? That's where the power of `ggplot2` comes into play.

Let's explore this ability with the `baby` dataset, which details a baby's birth weight and gestational period, as well as information about the mother. We are interested in the relationship betweeen gestational days and birth weight .

Exercise: Make a scatterplot with Gestational Days on the x-axis and Birth Weight on the y-axis. Specify the points to be 1mm circles and label the x and y axes.

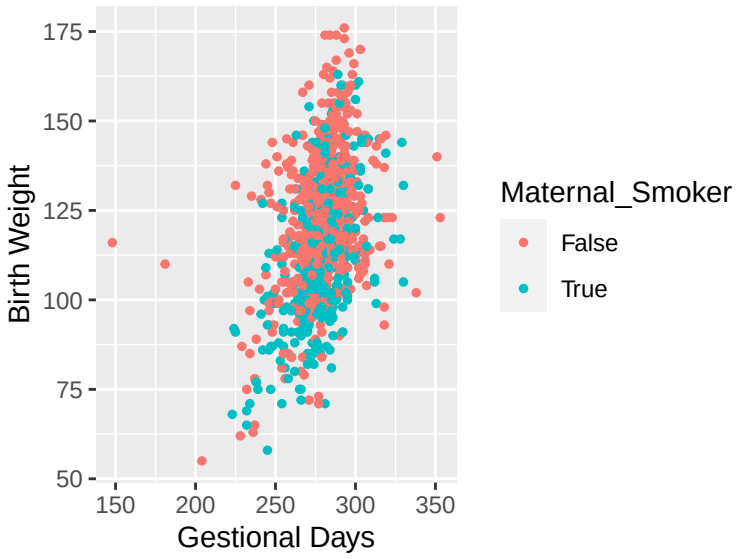
```
baby <- read.csv("baby.csv",
col.names = c("Birth_Weight", 'Gestational_Days', 'Maternal_Age', 'Maternal_Height', 'Maternal_Pregnancy
```

```
# insert code here
```

We hypothesize that a baby's gestational days and birth weight might be affected by whether the mother was a smoker. How can we incorporate that information into the above plot?

Color the points *by* the `Maternal_Smoker` variable in the table:

```
ggplot(data = baby) +
  geom_point(aes(x = Gestational_Days, y = Birth_Weight, color = Maternal_Smoker), size = 1) +
  labs(x = 'Gestional Days', y = 'Birth Weight')
```



What happened in the call above? We mapped the aesthetic `color` to the Maternal_ Smoker *variable* in our dataset. Since we're specifying the aesthetic as a variable of our data, instead of adjusting it manually, specify *inside* the `aes` function. `ggplot2` will automatically assign a unique level of the aesthetic (here a unique color) to each unique value of the variable - it even automatically adds a legend to distinguish the different values! Note that earlier, `ggplot2` created a "legend" for the x and y aesthetics by creating an axis line with tick marks and a label.

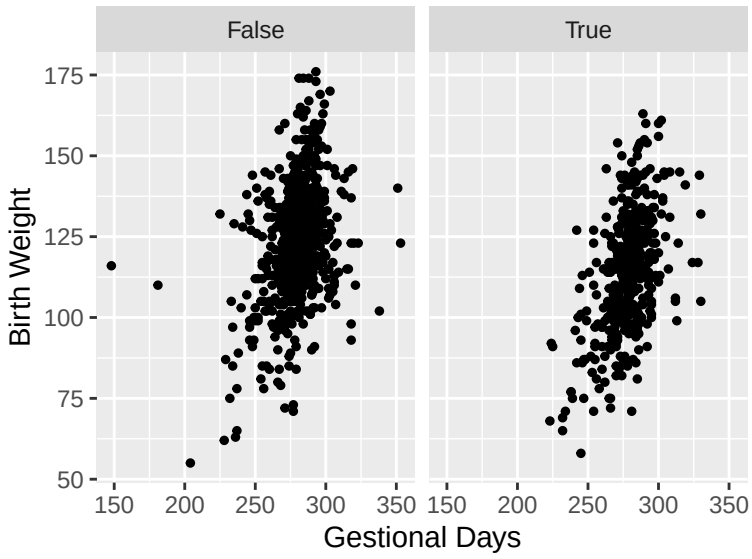
Exercise: Replicate the plot above but use the *shape* of a point to denote whether the mother was a smoker or not. What's wrong with this graph?

```
# Insert optional code here
```

Facets

The plot above gives some sense of how a baby might be affected by a smoking mother, but the two classes are heavily overlapped. It's difficult to discern any insights between the two groups. What if we wanted to make two subplots, that display the same information but subdivided by `Maternal_Smoker`? We can use facets! If we want to facet our plot with a single variable, we use `facet_wrap`. The first argument should be a formula: `~` followed by a variable name. Remember "formula" doesn't necessarily mean "equation" in R - it's just a data structure. Importantly, the variable you want to facet on must be *discrete* (what would happen otherwise?).

```
ggplot(data = baby) +
  geom_point(aes(x = Gestational_Days, y = Birth_Weight), size = 1) +
  facet_wrap(~Maternal_Smoker) +
  labs(x = 'Gestional Days', y = 'Birth Weight')
```

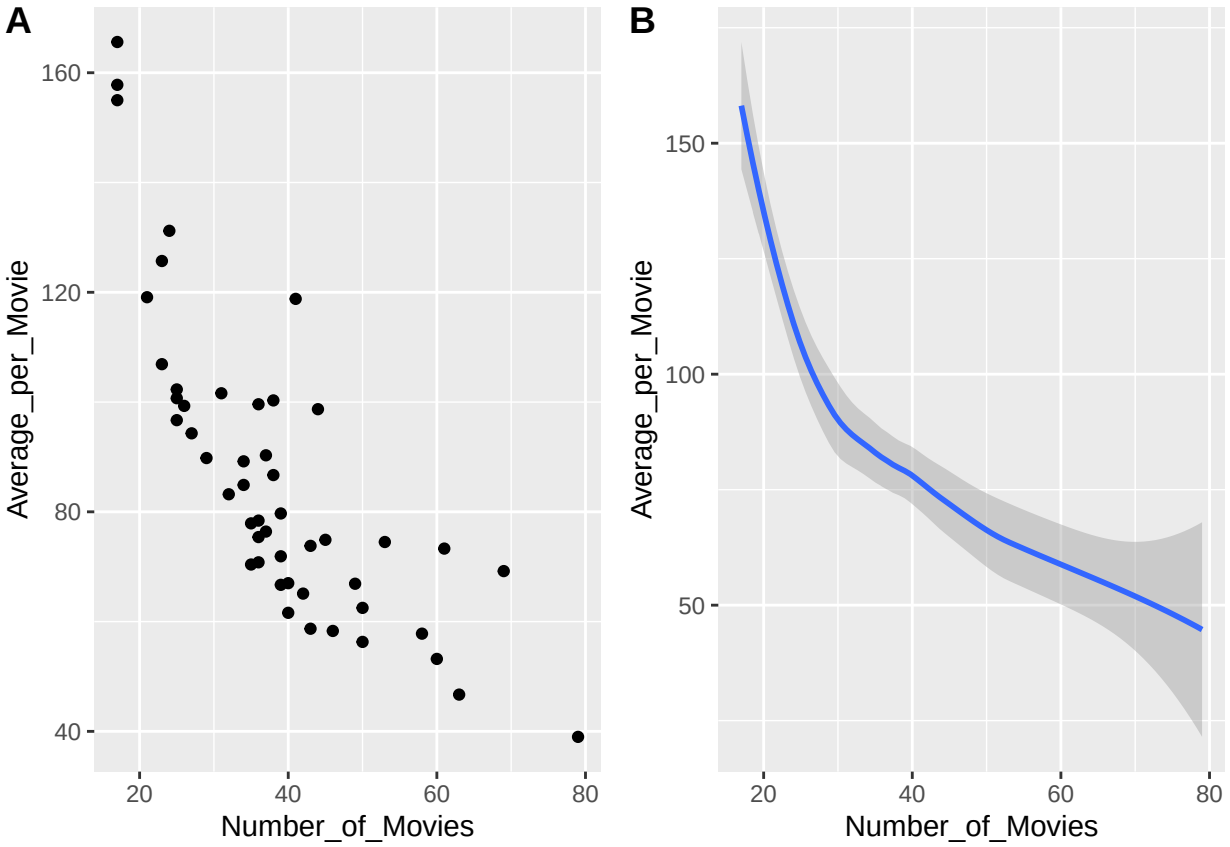


(Optional Statistics) Exercise: Earlier, we hypothesized that smoking during pregancy might be associated with a negative effect on a baby's in-utero development. But the plot above seems to indicate the non-smoking mothers have some lower weight and premature babies than smoking mothers. What could explain the dissonance between our intuition and this plot? Jot your ideas below:

Answer:

If you want to subdivide by two variables, use `facet_grid()`, and your formula argument should contain two variables - `a ~ b` - to denote the columns and rows of the subplots.

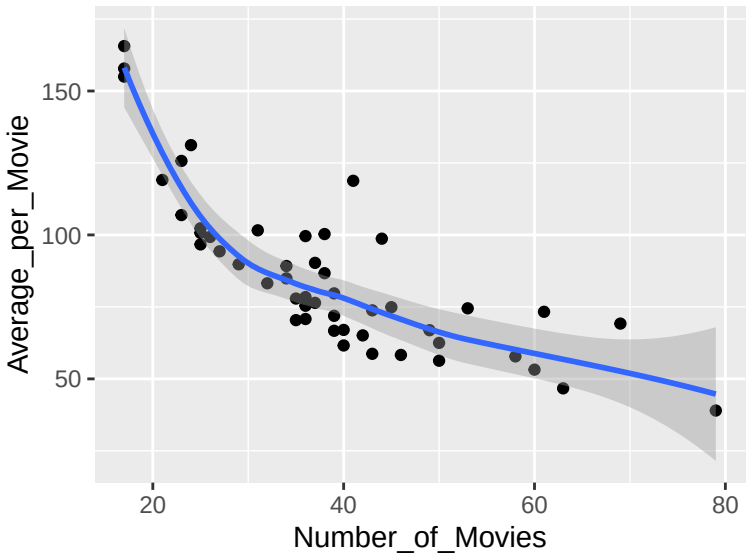
Smoothers



What's the difference between the two plots above? They have the same x variable, y variable, and they describe the same data. But each uses a different visual object to represent the data. In `ggplot2` syntax, we say they use different **geoms**. A geom is the geometrical object that a plot uses to represent the data. We describe plots by the type of geom they use and specify with `geom_<>()` (where the brackets are replaced by the object you want). Not every aesthetic works with every geom. Does it make sense to specify `shape = ...` in the right example above? What does it mean to alter the "shape" of a line?

You can even represent two geom objects in the same graph:

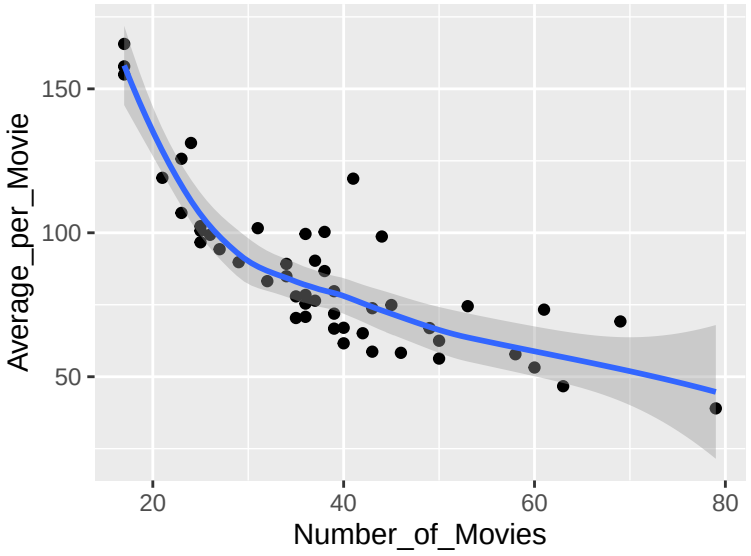
```
ggplot(data = without_outlier) +
  geom_point(aes(x = Number_of_Movies, y = Average_per_Movie)) +
  geom_smooth(aes(x = Number_of_Movies, y = Average_per_Movie))
```



Now we have a smoothed estimate of the x-y relationship overlaid with the points for more information. But notice we have redundancy in the code above - we specify the same mapping twice. What if we wanted to change the y variable, but forgot to change both occurances? We can avoid this repetition by passing the mappings to the outer `ggplot()` function. This will treat the mappings as a "global" setting that is applied to every succeeding geom object in the graph. This code is more succinct:

```
ggplot(data = without_outlier, aes(x = Number_of_Movies, y = Average_per_Movie)) +
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



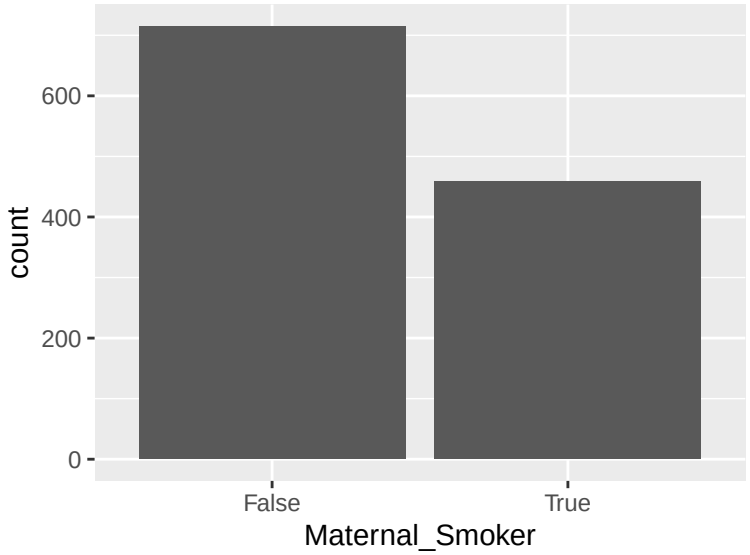
Any mapping within a geom object is a *local* mapping that will override the global mapping for that layer only. This allows you to display different aesthetics for different layers:

```
ggplot(data = without_outlier, aes(x = Number_of_Movies, y = Average_per_Movie)) +
  geom_point(mapping = aes(color = something)) +
  geom_smooth()
```

Bar Charts

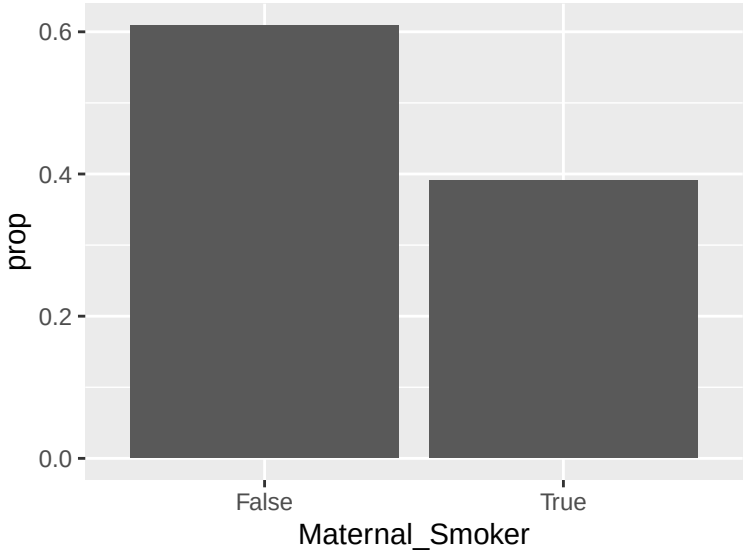
Let's return to the baby example, and investigate the distribution of smoking mothers in the dataset. Since `Maternal_Smoker` is a categorical variable, we want to use a bar chart/bar plot:

```
ggplot(data = baby) +
  geom_bar(mapping = aes(x = Maternal_Smoker))
```



Notice the y-axis displays the count of the two groups, but `count` wasn't an original variable in our dataset! Some functions in `ggplot2` will compute values for you to create the graphs - barcharts and histograms bin the data, smoothers fit a model and compute predictions, and boxplots summarize the distribution. The objects do this with an algorithm called a **stat**, or statistical transformation. Many of the geoms mentioned above have their own built in `stat` functions (that can be found in documentation). This is useful because it allows you to override the default settings if you want something more customizable. For example, I always prefer to plot bar charts in terms of proportions rather than raw counts .

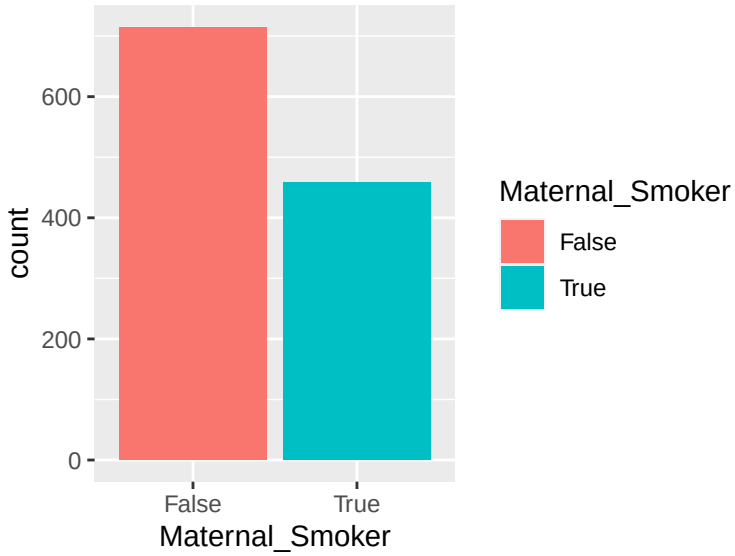
```
ggplot(data = baby) +
  geom_bar(mapping = aes(x = Maternal_Smoker, y = stat(prop), group = 1))
```



(Why `group = 1`? See here).

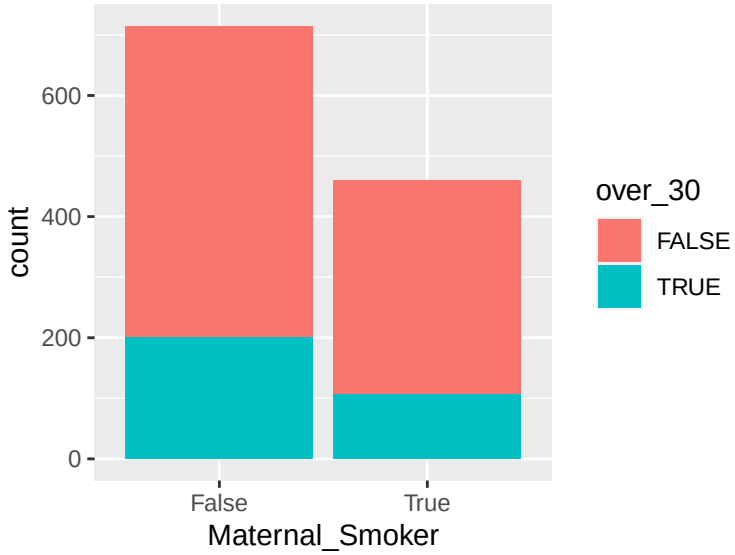
What if we wanted to brighten up the plot above? You can color it!

```
ggplot(data = baby) +
  geom_bar(mapping = aes(x = Maternal_Smoker, fill = Maternal_Smoker))
```



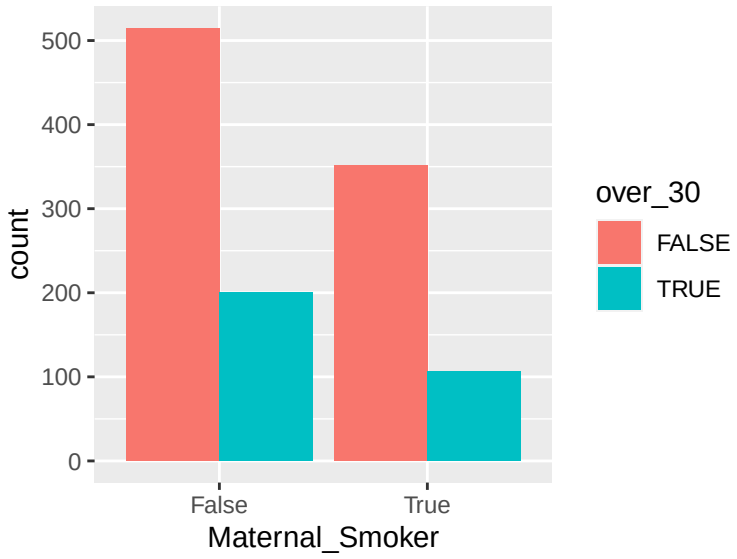
This can help visually distinguish the two bars. But currently, this displays the same variable twice (on x-axis and in legend), which feels redundant. What if we wanted to color by *another variable*?

```
# Create another categorical variable for this example
baby$over_30 <- baby[, 'Maternal_Age'] >30
ggplot(data = baby) +
  geom_bar(aes(x = Maternal_Smoker, fill = over_30))
```



Now we can see each category of Maternal Smoker subdivided by *another categorical* variable. When doing this, `ggplot2` automatically stacks the bars with every combination of the two variables. If you want something different, you can place the bars next to each other:

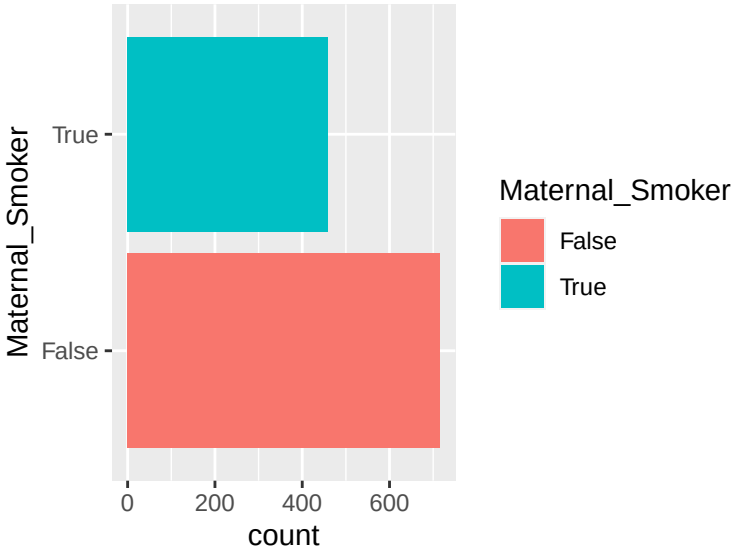
```
ggplot(data = baby) +
  geom_bar(aes(x = Maternal_Smoker, fill = over_30), position = 'dodge')
```



Position can also be specified to make the stacked bars the same height (for proportion comparision), or on top of each other but not stacked (not so great for bar charts but better for points). The `position` argument can also be used to **jitter** overlapped points in a scatterplot.

Finally, you can also swapped the orientation of your bar charts (which can make long labels on the x-axis easer to read):

```
ggplot(data = baby) +
  geom_bar(mapping = aes(x = Maternal_Smoker, fill = Maternal_Smoker)) +
  coord_flip()
```



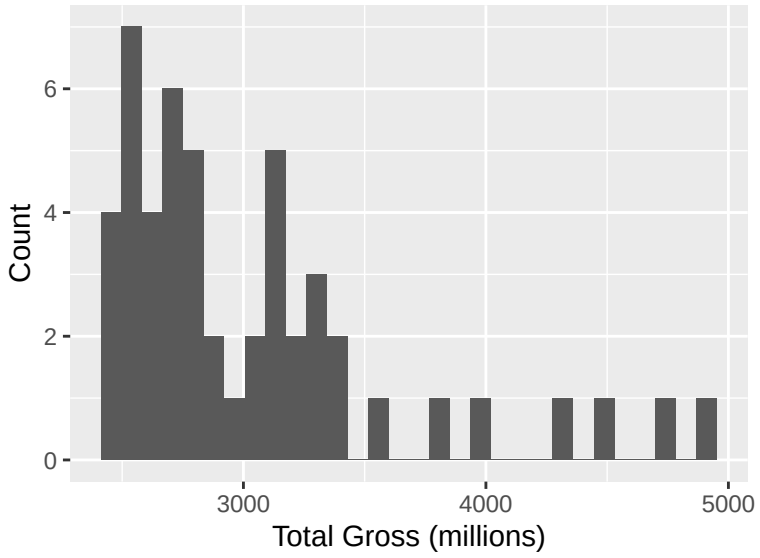
`coord_flip()` is an example of a function that adjusts the coordinate axes of a plot. This can be useful when adjusting aspect ratios for maps or plotting polar coordinates (see R for Data Science tutorial in References for examples).

Histograms

Visualizing the distribution of a single numerical variable is hugely important in statistics.

```
ggplot(data = actors, aes(x = Total_Gross)) +
  geom_histogram() +
  labs(x = 'Total Gross (millions)', y = 'Count')
```

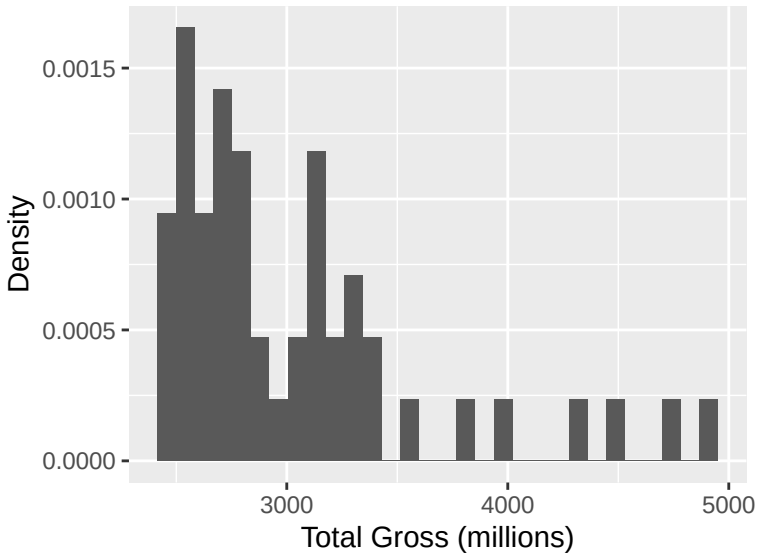
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



By default, `ggplot2` uses an automatic binwidth of 30. This value may not be suitable, so you can set the value yourself. Further, the default for histograms is to make the y-axis the count. But just like barcharts, count is unideal. You can make a histogram on the density scale by specifying density in `aes`:

```
ggplot(data = actors, aes(x = Total_Gross, y = ..density..)) +
  geom_histogram() +
  labs(x = 'Total Gross (millions)', y = 'Density')
```

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



(Optional) Exercise: Play around with possible binwidths in the plot above. What happens when bindwidth is very small? Very large?

Again, oftentimes we want to display information about two groups in the same plot. We want to investigate how the distribution of `Gestational_Days` changes for smoking and non-smoking mothers.

Exercise: To show two distributions, would it be better to `facet_wrap` on `Maternal_Smoker` or specify `Maternal_Smoker` as an `aes`? If the latter, which `aes`?

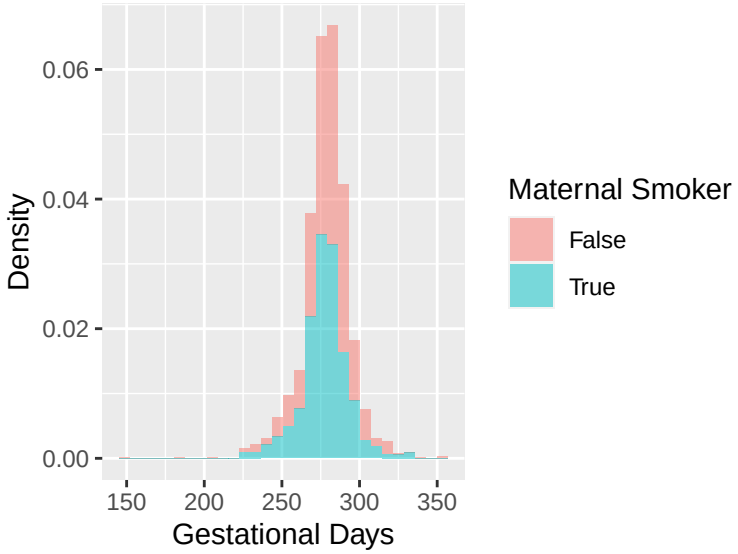
Answer:

```
# code for grouped plot here
```

For overlaid histograms, we may worry that the color of one distribution blocks some aspect of the other. We can control the opacity of the distributions with the **alpha** parameter:

```
ggplot(data = baby) +
  geom_histogram(aes(x = Gestational_Days, y = ..density..,  fill = Maternal_Smoker), alpha = 0.5) +
  labs(x = 'Gestational Days', y = 'Density', fill = 'Maternal Smoker')
```

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



Exercise: Play around with the `alpha` values above. What happens when `alpha` is small? Large?

Answer

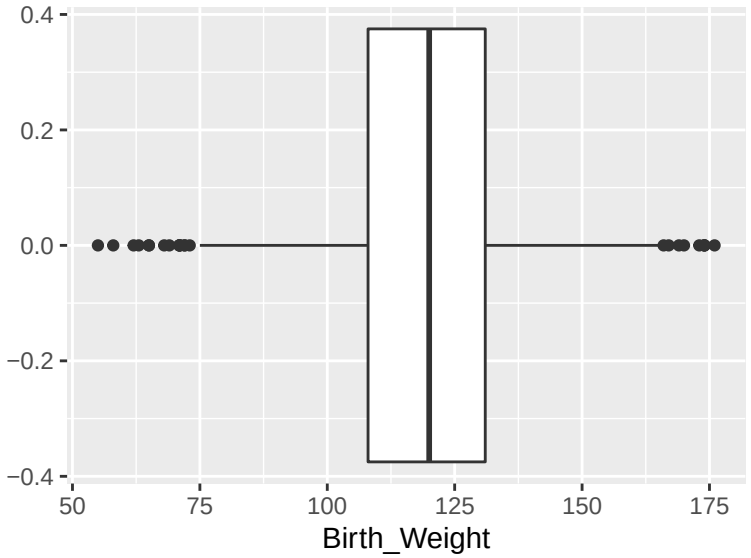
Exercise: What happens if you try to set the `alpha` aesthetic by the `Maternal_Smoker` variable?

Answer:

Boxplots

If we're more interested in a broader summary of the distribution of a single numerical variable rather than its shape, we can make a boxplot:

```
ggplot(data = baby) +
  geom_boxplot(mapping =aes(x = Birth_Weight))
```



Exercise: Create a figure of boxplots of birth weight subdivided by `Maternal_Smoker` in the cell below.

```
# insert code here
```

Template for Layered Plots

We've covered a lot of options above. Let's summarize the options by adding to our template ggplot call:

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```

Replace the brackets <> with your specifics. You don't always have to specify everything (e.g. `stat`) because of defaults within `ggplot2`. You'll get a better feel for this with more practice.

Saving Plots

Once you've created a plot, you'll want to save it as an image to use elsewhere. Instead of screenshotting the figure to your desktop, use `ggsave`.

By default, `ggsave` will save the last plot that you displayed. If you want to save the figure to your *current working directory*, use the command:

```
# Saves to current working directory
# Saves most recent plot run
ggsave(filename = "filename.png")
```

If you want to specify a file path where the figure should be saved (perhaps an `images` folder within your working directory), use the command:

```
# Saves most recent plot run
ggsave(filename = 'filename.png', path = "/path/to/figure/")
```

If you want to save a different plot, one that has been saved to a variable, do so with the command:

```
# Somewhere earlier there should be code
# myplot = ggplot(data = dat) + ....
ggsave(plot = myplot, filename = 'filename.png', path = "/path/to/figure/")
```

If you want to specify a size of the plot, to fit within your paper or homework assignment, you can do so with:

```
# default units are of current graphics device
ggsave(filename = 'filename.png', width = 4, height = 4)
# But can specify units as well!
ggsave(filename = 'filename.png', width = 20, height = 20, units = 'cm')
```

Advanced Topics + Next Steps

There are more advanced topics not covered here that allow for further customization of figures, such as:

- Adding custom themes (something other than grey background with white gridlines)
- Create functions to automate plotting (if you use the same format for many figures)
- Adjusting axes scales

For references for the above topics see [Ram] below.

In general, becoming familiar with these functions requires lots of practice and reading of documentation. Another tutorial (*with practice problems*) can be found here, a complete reference of functions can be found here, and a cheat sheet of syntax can be found here. Just like with learning a new programming language, becoming well-versed in googling "r ggplot how to (task)" will serve you well in finding the syntax for a desired task.

Finally, `ggplot2` is part of the larger `R tidyverse` functionality. Oftentimes data must be wrangled into a proper shape before you can visualize the desired attributes. The `dplyr` library has many functions for manipulating the structure of your data. A good tutorial for the larger tidyverse can be found here.

References

The following references were used to create this tutorial.

- Adhikari, Ani, DeNero John. "Chapter 7: Visualization". Computational and Inferential Thinking: The Foundations of Data Science. https://inferentialthinking.com/chapters/07/Visualization.html
- "Chapter 3: Data Visualization". R for Data Science. https://r4ds.had.co.nz/data-visualisation.html
- Freeman, Michael, Ross, Joel. "Chapter 13: The `ggplot2` Library". Technical Foundations of Informatics. https://info201.github.io/ggplot2.html
- Ram, Karthik. "Data Visualization with R & ggplot2". UC Berkeley STAT133 Lecture, September 2, 2013. https://github.com/ucb-stat133/stat133-slides/blob/master/ggplot-karthik.pdf