rpapoda Phase 2: Exploratory Data Analysis

```
library(knitr)
source(purl("rpapoda_codebook_phase_2.Rmd"))
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr
```

```
## Conflicts with tidy packages ----------------------------------------------
```

```
## filter(): dplyr, stats
## lag():    dplyr, stats
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   id = col_integer(),
##   goal = col_double(),
##   pledged = col_double(),
##   currency_trailing_code = col_logical(),
##   deadline = col_datetime(format = ""),
##   state_changed_at = col_datetime(format = ""),
##   created_at = col_datetime(format = ""),
##   launched_at = col_datetime(format = ""),
##   staff_pick = col_logical(),
##   is_starrable = col_logical(),
##   backers_count = col_integer(),
##   static_usd_rate = col_double(),
##   usd_pledged = col_double(),
##   spotlight = col_logical()
## )
```

```
## See spec(...) for full column specifications.
```

```
## Warning in strptime(xx, f <- "%Y-%m-%d %H:%M:%OS", tz = tz): unknown
## timezone 'zone/tz/2019b.1.0/zoneinfo/America/Denver'
```

Exploratory Data Analysis

This is one of the phases of data analysis that gets the most press (along with machine learning, AI, etc. that are talked about so much these days). Some of the more tangible evidence of your work will come from this phase in the form of the beautiful graphs that R is known for as well as useful tables.

During Exploratory Data Analysis, frequently referred to as EDA, we will explore various aspects of the variables in our dataset and the relationships among them, with the goals of gaining an understanding of our data, spotting any problems with the data, and generating ideas to test in the modeling phase. Much of our exploration is open ended and free-form; creative thinking is encouraged. If one graph raises a question, try to come up with a way to answer it, perhaps another graph.

Goals

There is a lot to cover in this phase, and therefore we will have several different levels of goals. I've organized them into "Goals", "Tasks", and "Steps". There are also several very important models that we will explore.

Primary Goals of this Phase of Our Analysis

- Gain an understanding of, familiarity with, and insight into our dataset
- Find (and deal with) any problems with our data
- Generate ideas to use and test in the modeling phase
- (Further) refine our question, where appropriate

Primary Tasks of this Phase

- Build plots or graphs
- Build useful tables
- Transform our data as necessary

Transforming our data may include formatting variables, creating new variables based on calculations from other variables, or otherwise making our data more useful.

Models or "Grammars" Learned in this Phase

- The Grammar of Graphics, developed by Leland Wilkinson
- The Grammar of Data Manipulation, developed by Hadley Wickham

Corresponding to each of those "grammars", we will become very familiar with two of the primary packages or tools of the `tidyverse`: `ggplot2` for graphics, and `dplyr` for data manipulation and transformation.

I mentioned earlier that this phase involves a lot of free-form thinking. While this is one of the most important aspects of this phase, and also one of the most fun, it is important to have some sort of roadmap to keep us focused and our analysis moving forward. For this purpose I really love the model layed out by Stephen Few in his book Signal: Understanding What Matters in a World of Noise (https://www.amazon.com/Signal-Understanding-Matters-World-Noise/dp/1938377052/ref=as_li_ss_tl?s=books&ie=UTF8&qid=1525375375&sr=1-1&keywords=signal+stephen+few&linkCode=ll1&tag=jaycreighton-20&linkId=7a1fa5a2bdec16aae2d0625b4a22ed3b). I've pared it down a little to suit our purposes, and it provides a simple set of types of exploration to work through that will very nicely allow us to accomplish all of our goals for this phase.

Steps of EDA, or Types of Graphs to Explore

> 1. Variation within categories
> 2. Variation within measures
> 3. Variation through time
> 4. Relationships among measures
> 5. Relationships among categories (Few, 2015)

To drill down a little further, we here's a preview of the following basic types of graphs that we will learn how to build:

Types of Graphs

- Scatterplots
- Bar Graphs
- Line Graphs
- Box Plots
- Dot Plots?
- Histograms
- Frequency Polygons

We will do much more than build simple graphs, however. We will make many adjustments, additions, and modifications to them in order to gain the insight that we need.

Additional Layers and Tools for Building Graphs

- Plot and axis titles
- Axis limits for "zooming"
- Smoothers to reveal trend
- Position adjustments
- Shape and color adjustments
- Facet wraps and grids

Just a quick note about our approach during this phase regarding the appearance of our graphics: during EDA we will focus on making the graphs as informative as possible, but we won't yet spend much time making them "pretty". During the Communication Phase, Phase 4, we will learn many tools to customize the appearance of our graphs to suit our presentation needs.

Along with relying on free-form thinking and creativity, EDA is very iterative phase. You might have a guess about what a graph will tell you, then it might show you something different when you create it. In this case, you might have to investigate a little further. As data analysts, iteration is a very frequent part of our job. After doing it for years, I found that I became much better at the process when I thought through it and formalized the steps. One final model that I will introduce before we get started is the model I use before and after each graph or table I make (or really for any other step in the analysis). I will demonstrate it on a few graphs, but I won't spell it out each time. Now that I know the steps, I follow them implicity rather than explicitly. I want you to learn the process, go through the steps a few times until you have a good feel for it, then adapt or modify it to your own needs if you see fit, then apply your own process intuitively and informally in your own analysis. We want as many of the tools as possible to become second nature, and iteration is one of the most important of those for good data analysis.

Iteration Cycle

1. Set a goal for the task ahead and define expectations and criteria for success
2. Attempt to complete the goal or perform the task

3. Compare your results to your expectations or your criteria for success
4. If necessary, use judgment and experience to decide whether you think the expectations should be altered, or the task should be repeated under different parameters
5. Make adjustments, and repeat as necessary

Clearly there is a lot to get to during this phase and sets of lessons. Don't worry about the volume of information as the models we will learn help organize and give meaning to it all, as well as making it simple and intuitive to use. I'll put the primary goals, tasks, and steps onto flashcards, as knowing them will help keep us focused and moving forward with our analysis.

An Introduction to ggplot2

With all of that out of the way, let's learn how to create graphics. The graphics package we will use is ggplot2, another of the many great packages in the tidyverse and developed by Hadley Wickham. It is based on the "Grammar of Graphics" (that's what the "gg" in ggplot2 stands for) developed by Leland Wilkinson. At its most basic level, the grammar describes "the deep features that underlie all statistical graphics" (Wickham, 2016). Dr. Wickham furthered Wilkinson's ideas to become the "Layered Grammar of Graphics", which, as the name implies, defines the role of each layer in a plot and the components that comprise it. This is done to facilitate our communication with R. The layered grammar is how we communicate with R and ggplot2 to build graphics.

In other words, every statistical graphic contains a certain set of features. The grammar outlines the categories of those features. Using the package to build graphics is simply a matter of telling the package what to do for each feature in a way that the software understands.

So let's start with those features common to every statistical graph, the components of the Grammar of Graphics.

The layered grammar defines a plot as the combination of:

1. A default dataset and set of mappings from variables to aesthetics
2. One or more layers, each composed of a geometric object, a statistical transformation, a position adjustment, and optionally, a dataset and aesthetic mappings
3. One scale for each aesthetic mapping
4. A coordinate system
5. The facetting specification (Wickham, 2010)

Note that the layer, as defined above, has several components. Layers create the objects we actually see on the graph.

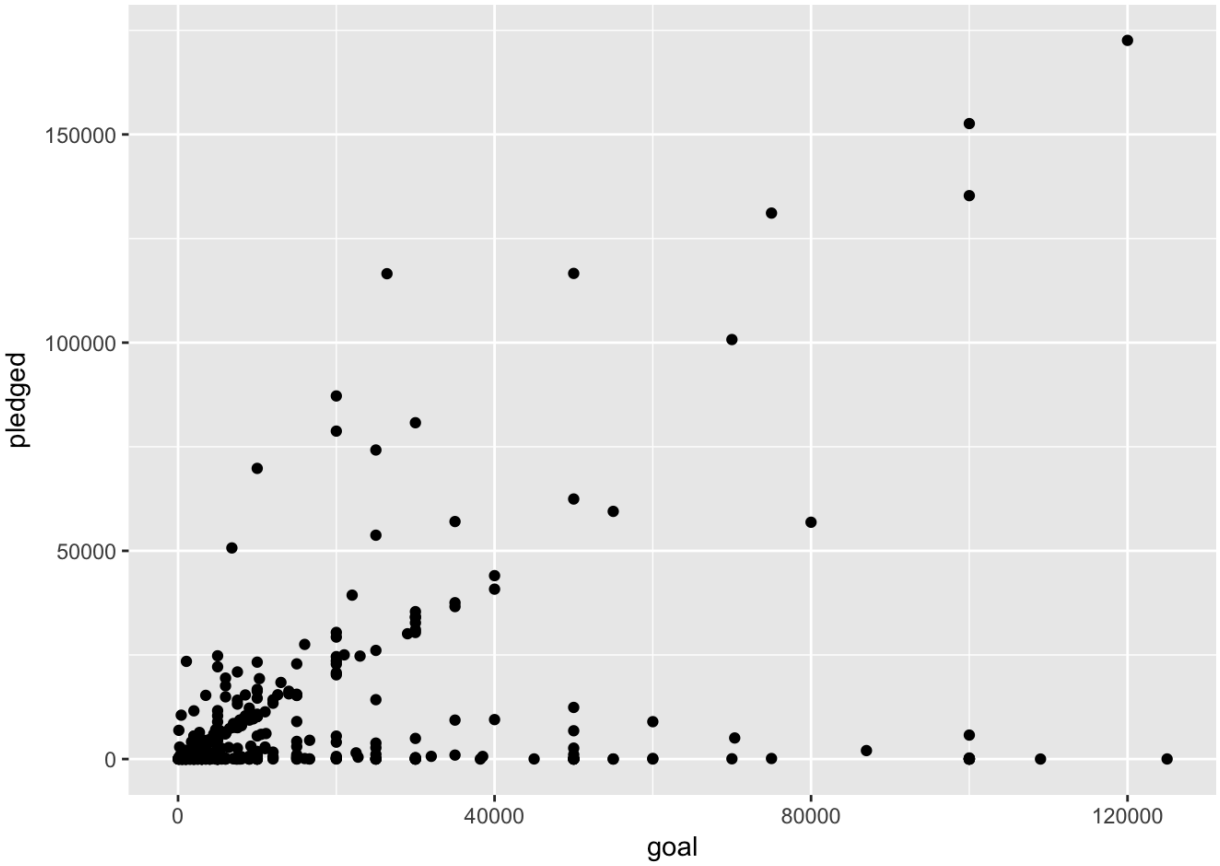
The Components of a Layer

- Data and aesthetic mapping
- A statistical transformation (stat)
- A geometric object (geom)
- A position adjustment (Wickham, 2010)

Every plot we make in ggplot will have each of these components. Fortunately, however, we are able to rely on defaults for some of the information, so we don't have to input all off those every time.

Let's look at a simple plot identify those components.

```
ggplot(ks, aes(x = goal, y = pledged)) +
    geom_point()
```



Let's first look at the code that creates this plot. This plot demonstrates just about the bare minimum required to make a ggplot graph. Qe must include a call to the `ggplot()` function, a call to the `aes()` function, usually as an argument to `ggplot()`, and a geom function. Also note the `+` sign following the `ggplot()` function; this is required between each function we add.

So let's go back through this and compare to the components of the Grammar of Graphics, including the components of layers, and see how things shape up.

A refresher:

The layered grammar defines a plot as the combination of:

1. A default dataset and set of mappings from variables to aesthetics
2. One or more layers, each composed of a geometric object, a statistical transformation, a position adjustment, and optionally, a dataset and aesthetic mappings
3. One scale for each aesthetic mapping
4. A coordinate system
5. The facetting specification (Wickham, 2010)

The Components of a Layer

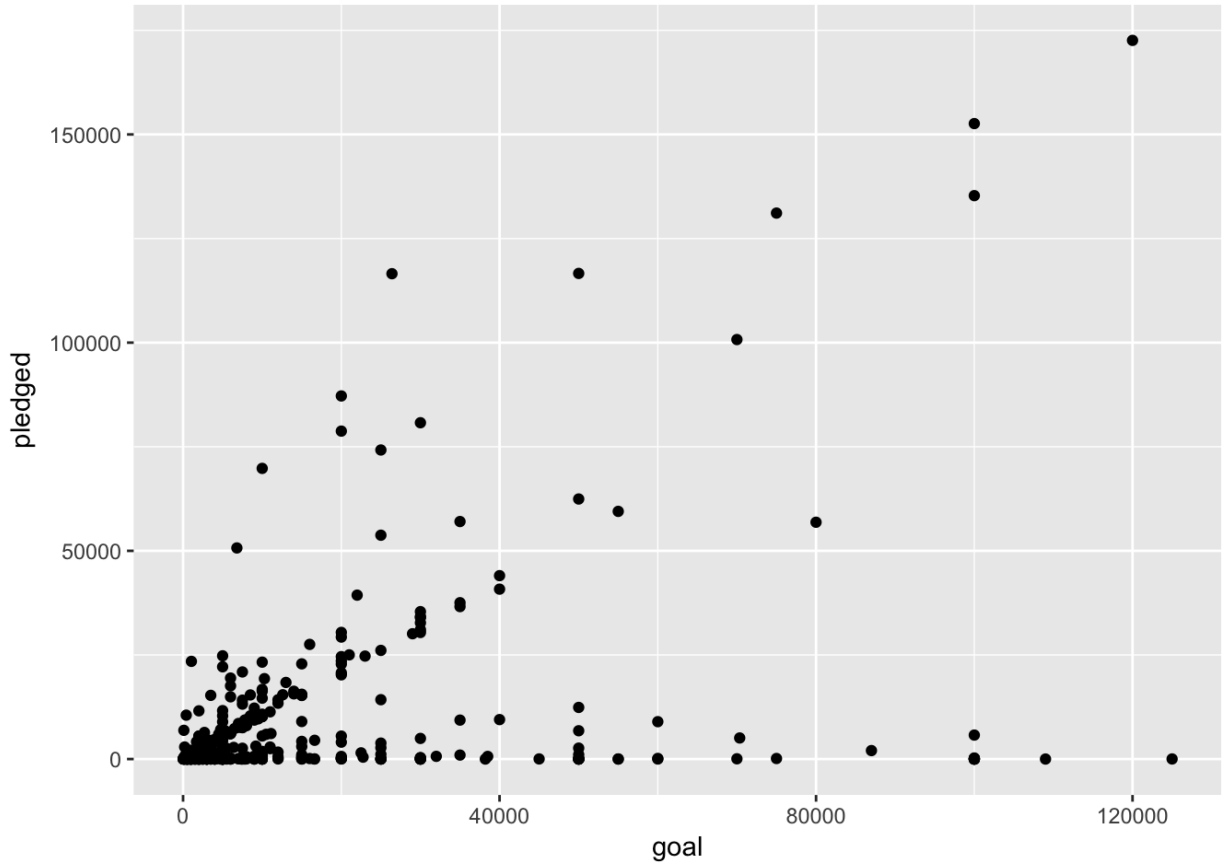
> - Data and aesthetic mapping
> - A statistical transformation (stat)
> - A geometric object (geom)
> - A position adjustment (Wickham, 2010)

Looking at that bit of code, we can identify a few of those components. The first argument to the `ggplot()` function is “ks”. That is our dataset, and that satisfies the first part of the first component of the grammar. Right after the `ks` argument, we have a function called `aes()`. This is short for “aesthetics”, which refer to the way that data is “mapped” to the plot. We’ll go into how aesthetics work in just a bit, but notice that within the `aes()` function we have specified the variables we want on our x and y axes. So the whole first part of the grammar has been taken care of: data and mapping of variables.

The first component is pretty straightforward; it’s all right there in the first line of our code. The second is a little less straightforward, so let’s walk through it. A “layer” is essentially what you actually see on the graph. In our example above, the points are the most tangible part of the layer, and we call the points the “geometric object”, or “geom’ for short. As you might have guessed, we tell ggplot to make those points with the `geom_point()` function. Notice that right now it has no arguments.

But there’s a lot more on the lists, and we’ve run out of code! Where are the statistical transformations, position adjustments, scales, coordinate systems, and facetting specifications? Fortunately, ggplot is designed with simplicity in mind, and we are able to rely on defaults for much of the information. Let’s take a peak at what the code for the same plot would look like without relying on defaults.

```
ggplot(ks, aes(x = goal, y = pledged)) +
    geom_point(data = ks, aes(x = goal, y = pledged), stat = "identity", position = "identity") +
    scale_x_continuous() +
    scale_y_continuous() +
    coord_cartesian() +
    facet_null()
```



Notice that I produced the exact same plot with a lot more code. Obviously this is not how we want to make our plots if we can avoid it, as it's not particularly efficient, but there are a few things you can learn from seeing it all spelled out this way.

Again, all the extra code I added into the second graph simply spelled out what was already going to happen by default. In other words, when we type the shorter, first set of code, the second set of code is what is actually executed behind the scenes. While we will never have to type that exact set of code, everything we do to add to or change the above plot will be overriding one of those commands.

Let's quickly walk through the second set of code just so we can see how we are accomplishing everything that is required by the Grammar of Graphics, and so that we have an idea of what everything means.

Let's work through the additions in the order they appear in the code. The first additions are the first two arguments in the `geom_point()`: `data = ks` and `aes(x = goal, y = pledged)`. The thing to notice here is that they duplicate the first two arguments we passed to the `ggplot()` function. When we pass data and aesthetic mapping arguments to the `ggplot()` function, those become the default for the whole plot. Later on we will add additional geoms, lines or bars or smooths, for example, and they will all use the data and mappings that we specify in the `ggplot()` function. That is, of course, unless we override the default by specifying something different in the function for any one of those geoms. If we want geoms that use different data or variables, we just have to include those arguments in the function for any geoms that don't use the default.

Moving on to the next arguments in `geom_point()`, we have `stat = "identity"` and `position = "identity"`. These arguments represent the default statistical transformation and position adjustment for the object `geom_point()`. In this particular case, the argument "identity" essentially means "do nothing". In other words, no statistical transformation was performed, and we didn't adjust the position. Instead, the points were just displayed according to their "identity". We will learn how to override these defaults to perform statistical transformations and position adjustments later. Going back to comparing the second set of code to our list of components above, we have now satisfied everything required for a layer:

The Components of a Layer

- Data and aesthetic mapping: `data = ks, aes(x = goal, y = pledged)`
- Statistical transformation: `stat = "identity"`

- Geometric object: `geom_point()`
- Position adjustment: `position = "identity"`
  (Wickham, 2010)

Moving forward in the code, we get to the scale functions. `scale_x_continuous()` and `scale_y_continuous()` tell ggplot to use a continuous scale on both axes, which is the default. Other options for scales include log and square root transformations, among others.

Next comes the coordinate function. The default coordinate system used is the Cartesian system which determines location by position relative to the x and y axes and is represented by the function `coord_cartesian()`. We can do many things with the coordinate system functions, including zooming in and out, flipping the axes, fixing the aspect ratio, or changing to non-linear coordinate systems such as Polar coordinates. We will work with the coordinate systems later.

The last function in the code for our plot is `facet_null()`. Facetting is a system for subsetting the data and creating multiple plots on a single page based on those subsetting. There is tremendous power in facetting and we will explore it later on. Much like `stat = "identity"` tells ggplot to not perform any statistical tranformations (the default), `facet_null()` also tells ggplot to perform the default, which is no facetting.

Now that we have run through all of the code, let's look at the components of the Grammar of Graphics and see if we have satisfied all of the requirements.

The layered grammar defines a plot as the combination of:

1. A default dataset and set of mappings from variables to aesthetics: `ggplot(ks, aes(x = goal, y = pledged))`
2. One or more layers, each composed of a geometric object, a statistical transformation, a position adjustment, and optionally, a dataset and aesthetic mappings:
   `geom_point(data = ks, aes(x = goal, y = pledged), stat = "identity", position = "identity")`
3. One scale for each aesthetic mapping: `scale_x_continuous()` and `scale_y_continuous()`
4. A coordinate system: `coord_cartesian()`
5. The facetting specification: `facet_null()`
   (Wickham, 2010)

At the beginning of the Phase 2: EDA, I laid out a list of types of graphics we would create.

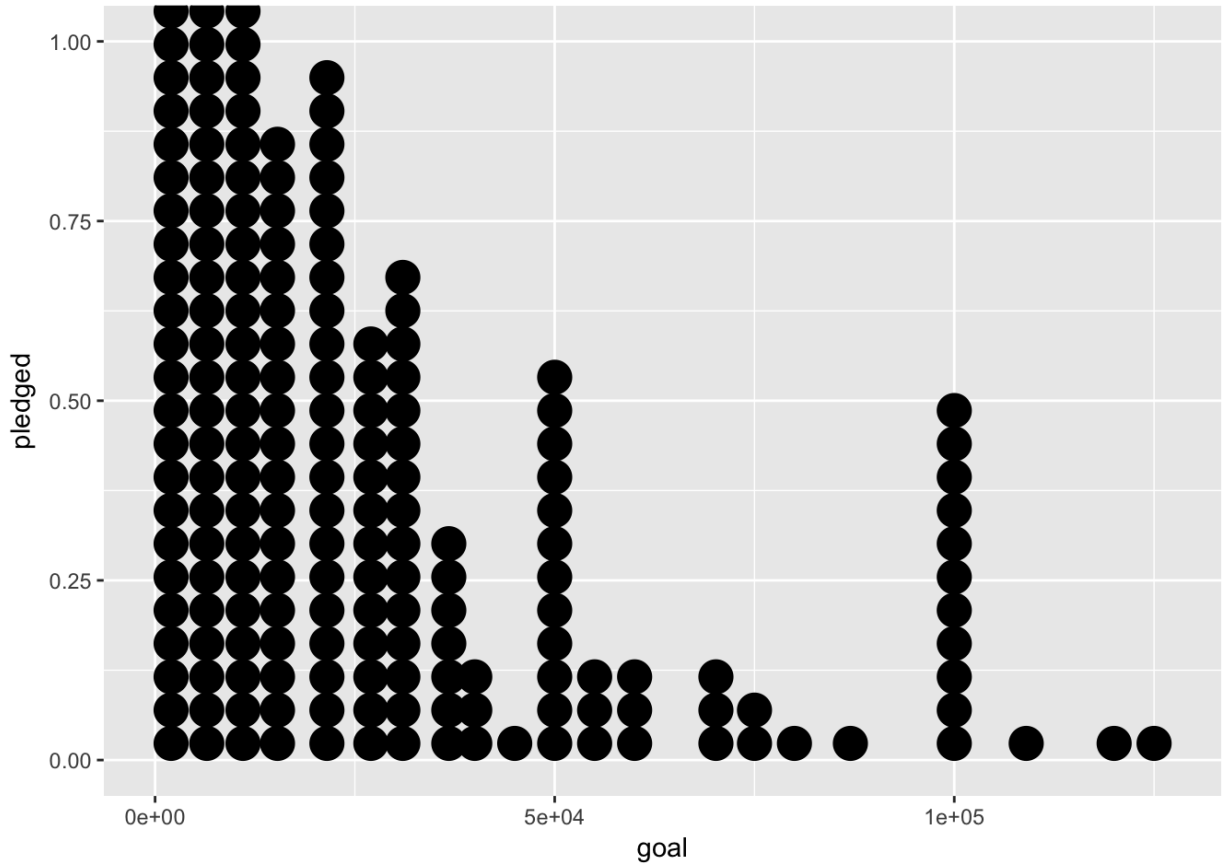
Types of Graphs

- Scatterplots
- Bar Graphs
- Line Graphs
- Box Plots
- Dot Plots?
- Histograms
- Frequency Polygons

In our example above, we created a scatterplot. That was determined by the type of geometric object, or "geom", we used: `geom_point()`. Each of the other types of graphs also have corresponding geoms. As long as the specified data works for that type of geom, switching from one type of plot to another can be as simple as changing the geom. Although it might be difficult to read with this particular set of data, we could create a dotplot graph just by changing `geom_point()` to `geom_dotplot()`.

```
ggplot(ks, aes(x = goal, y = pledged)) +
    geom_dotplot() # Note that this is the only I thing I changed
```

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



For the most part, the geom we choose is the primary part of the code responsible for determining the type of chart we are creating.

Here are the geoms associated with the types of charts listed above. We will create each of these before too long.

Types of Graphs

- Scatterplots: `geom_point()`
- Bar Graphs: `geom_bar()`
- Line Graphs: `geom_line()`
- Box Plots: `geom_boxplot()`
- Dot Plots: `geom_dotplot()`
- Histograms: `geom_histogram()`
- Frequency Polygons: `geom_freqpoly()`

Further customization of the chart comes primarily in two forms: adding additional geoms or overriding the defaults. Looking back at the list of additional layers and tools presented during the section on goals of this phase, we can see that all of them are either additional objects or overriding the defaults. Note that some geoms have different defaults associated with them than the ones we saw above with `geom_point()`. Can you guess which category each of the following might fall into, or how we might accomplish some of them? As promised at the beginning of the phase, we will learn how to do each of these and become very comfortable with each of them.

Additional Layers and Tools for Building Graphs

- Plot and axis titles
- Axis limits for “zooming”
- Smoothers to reveal trend

- Position adjustments
- Shape and color adjustments
- Facet wraps and grids

Some practice with ggplot2

Before we move any further along, I want to create a few graphs to help us get comfortable with ggplot. Because we don't yet have all of the tools to dive into the specific steps required for our analysis, we will use some data that is included with the ggplot2 package. The dataset we will look at is called `diamonds`. It comes preloaded in the ggplot2 package (which loads when we load the tidyverse package at the beginning of each session). Get a feel for the dataset by viewing the head of the data. Note that you can search the help for information on these preloaded datasets to get more in depth information.

It's already available, so we'll just put it into a data frame of the same name. Notice that when we do, it shows up in the environment pane on the top right.

```
diamonds <- diamonds
```

```
dim(diamonds)
```

```
## [1] 53940    10
```

```
head(diamonds)
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48

6 rows

```
?diamonds
```

Looks like we have 10 variables with nearly 54000 observations. In the head table we can look through the variables and get a feel for what we have available to us.

I won't walk through all of the steps of inspecting data that we did in Phase 1, but I strongly encourage you to hit pause and take a minute to do so.

As we talked about above, each of the components of the Grammar of Graphics is required, but the defaults can take care of much of that for us. Therefore we can strip the requirements down just a bit.

The bare minimum required for any plot:

1. Data
2. Aesthetic Mappings (from variables to properties on our graph)

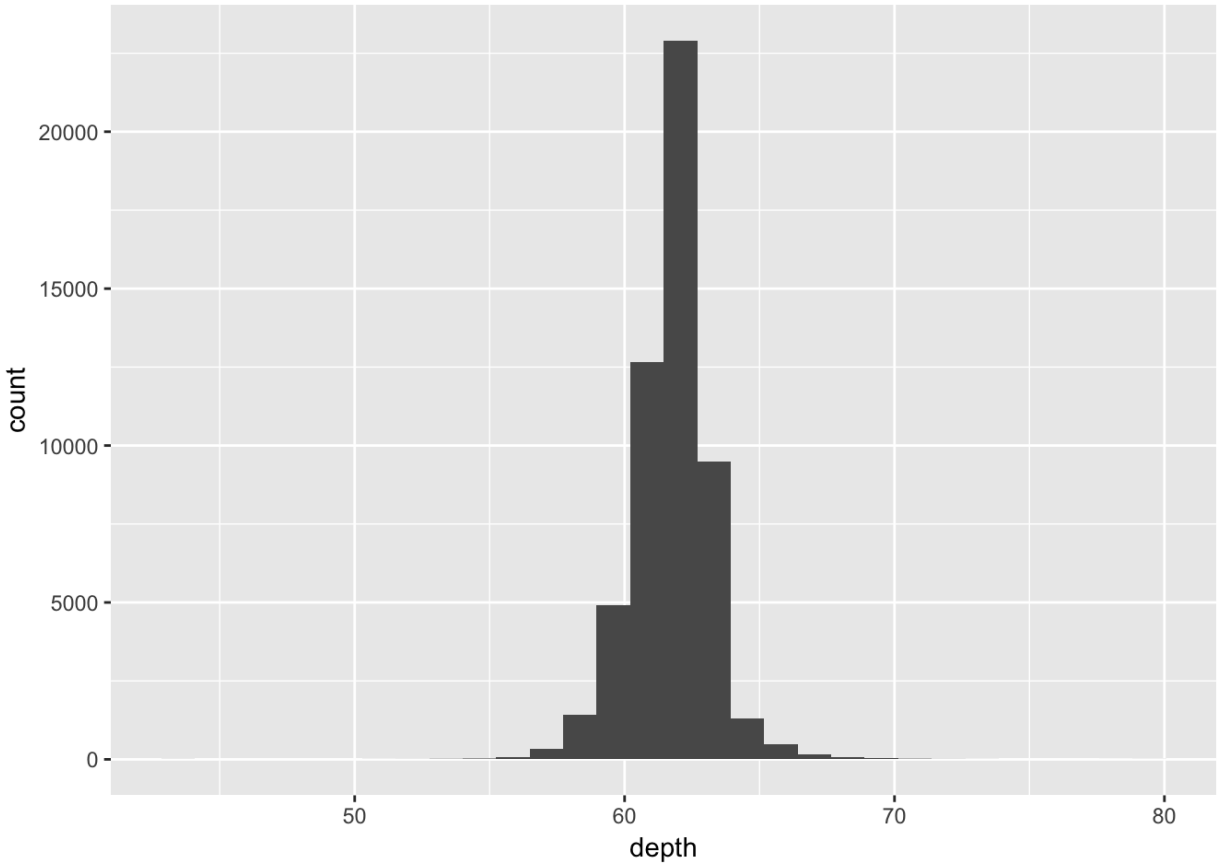
3. A Geom

More specifically, we must include a call to the `ggplot()` function, a call to the `aes()` function, usually as an argument to `ggplot()`, and a geom function. Don't forget to separate all functions with a `+` sign.

Let's make a few graphs. I'm going to run through a handful of graphs to illustrate a few things about how ggplot works and to begin to get you comfortable making them. I'm going to start with a very simple graph using just the bare minimum above, then I'm going to add a little complexity to it. Don't worry about memorizing every specific or understanding how everything works, just try to get a feel for the possibilities and the overall process.

```
ggplot(diamonds, aes(x = depth)) +
    geom_histogram()
```

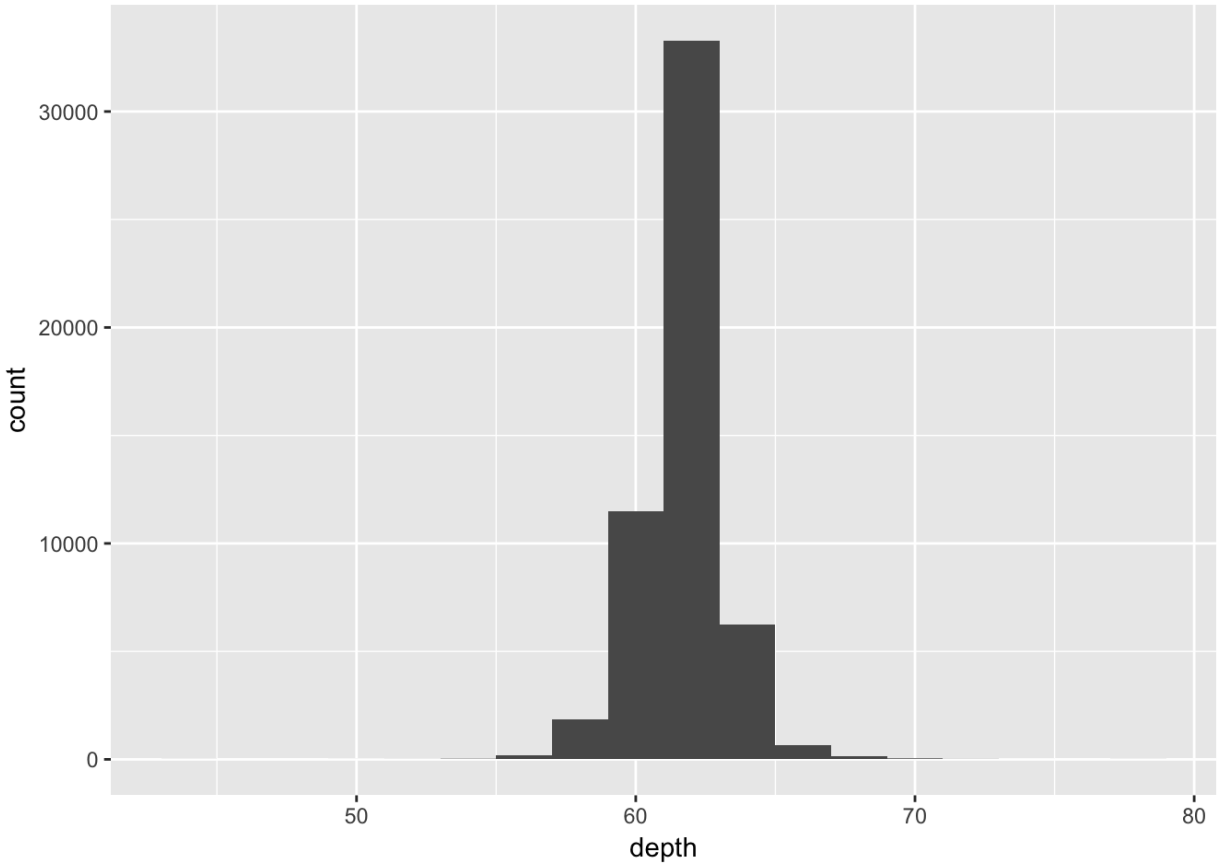
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Just the bare minimum: data, mapping, and a geom
```

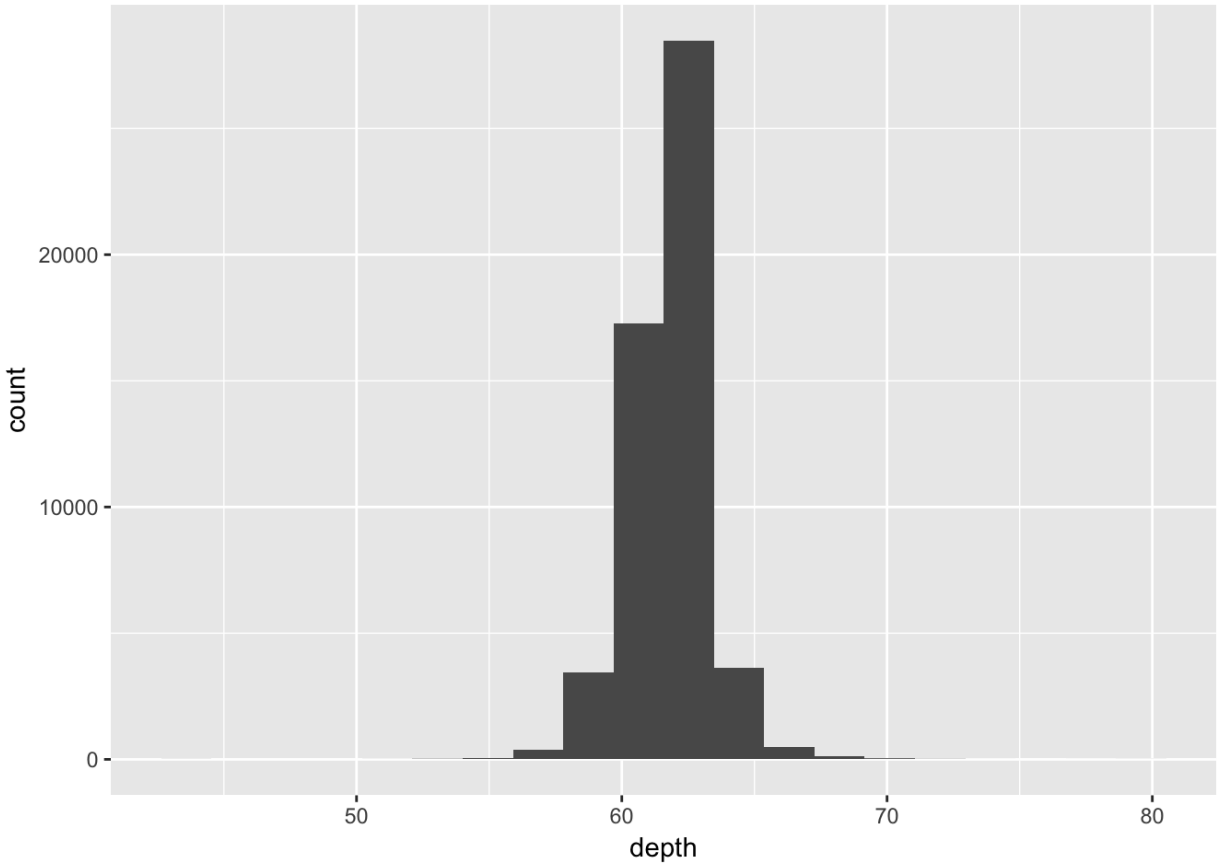
You'll see that error about the number of bins used anytime you use the default number of bins. The default is 30, but we can override that.

```
ggplot(diamonds, aes(depth)) +
    geom_histogram(binwidth = 2)
```



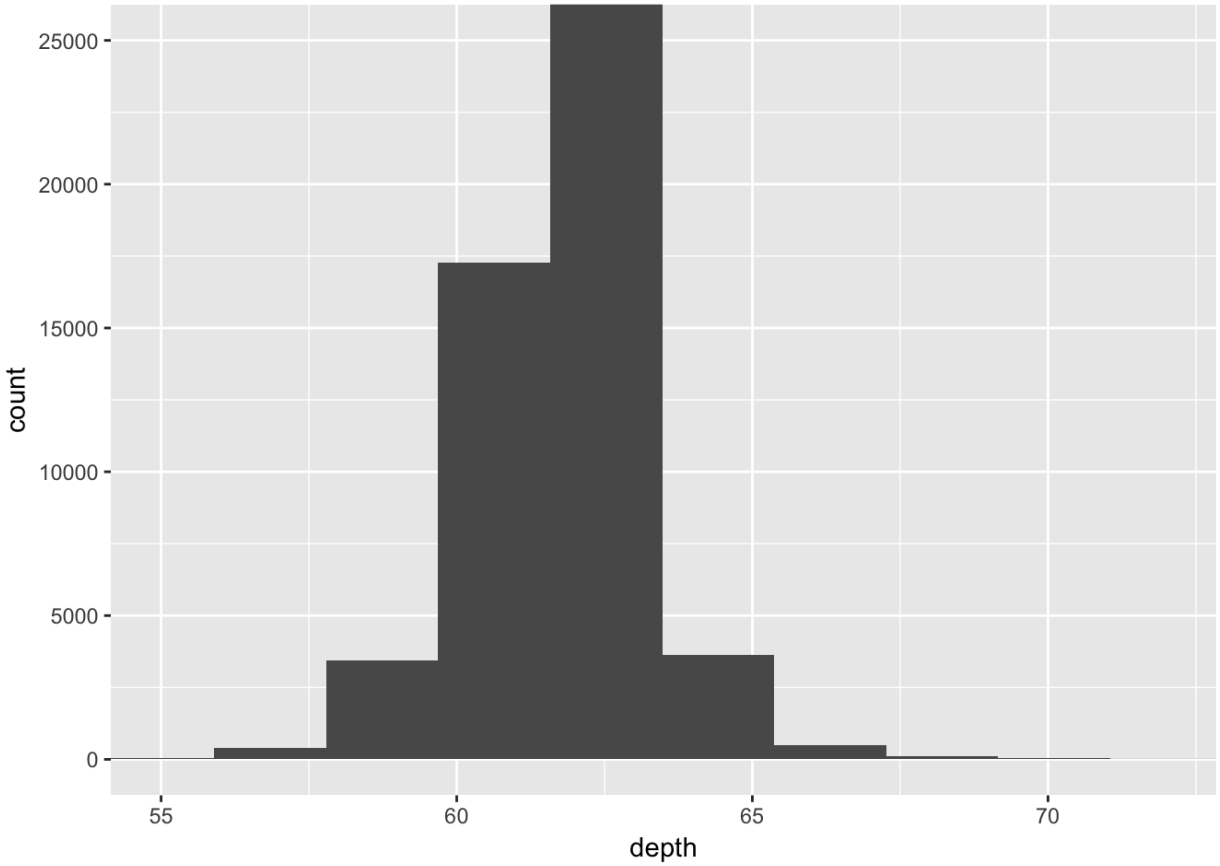
Sometimes it is more appropriate to change the number of bins than the binwidth.

```
ggplot(diamonds, aes(depth)) +
    geom_histogram(bins = 20)
```



Let's see an example of overriding a default like we talked about earlier. We'll zoom by adding/changing an argument in the coordinate system.

```
ggplot(diamonds, aes(depth)) +
    geom_histogram(bins = 20) +
    coord_cartesian(xlim = c(55, 72), ylim = c(0, 25000))
```

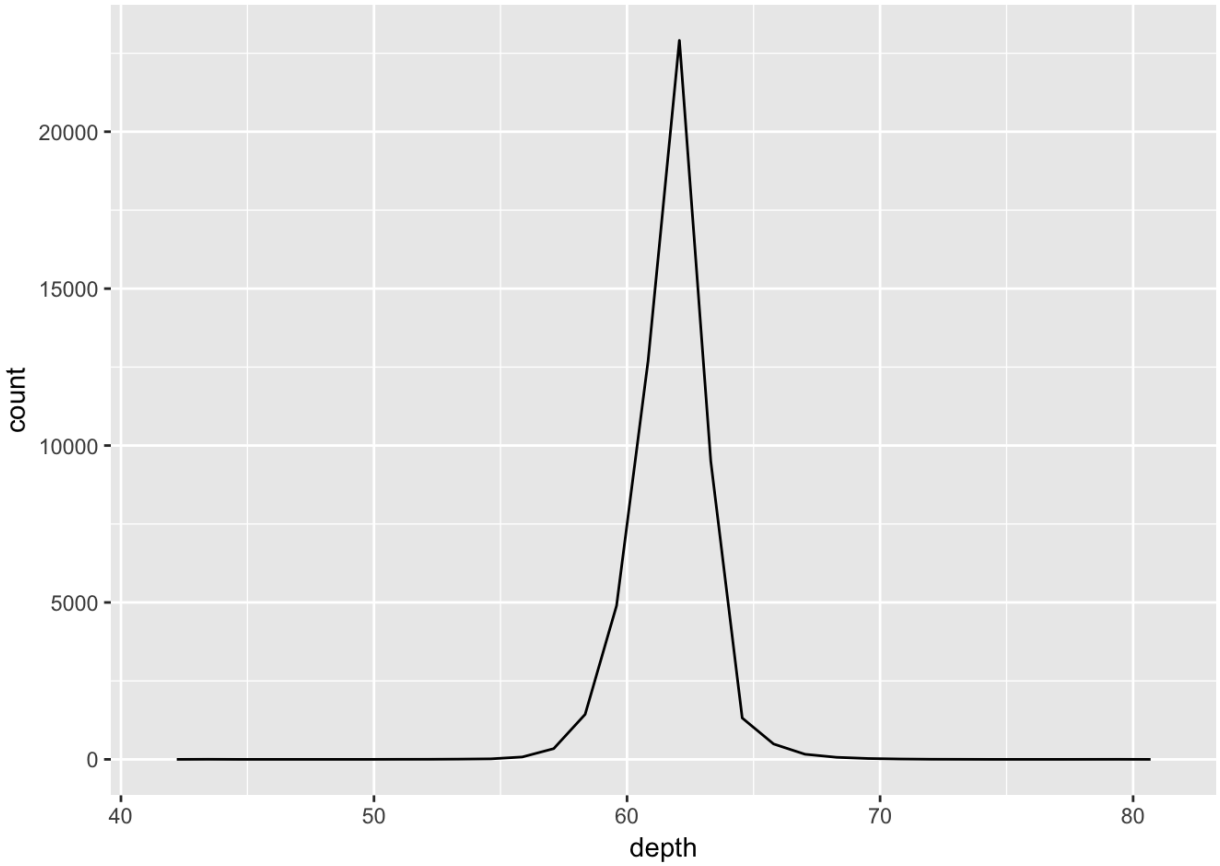


Notice that in the first histogram, I put `aes(x = depth)`, but in the next few I just put `aes(depth)`. The first two arguments expected by the `aes()` function are x and y, in that order (Run `args(aes)` or `?args` to check that). Remember that if we put arguments in the order the function expects them, we do not need to label them. If things are simple, feel free to omit the label, but if it helps keep things organized, feel free to label them, especially as things get more complicated.

We could change the look of that by changing to the frequency polygon geom.

```
ggplot(diamonds, aes(depth)) +
    geom_freqpoly()
```

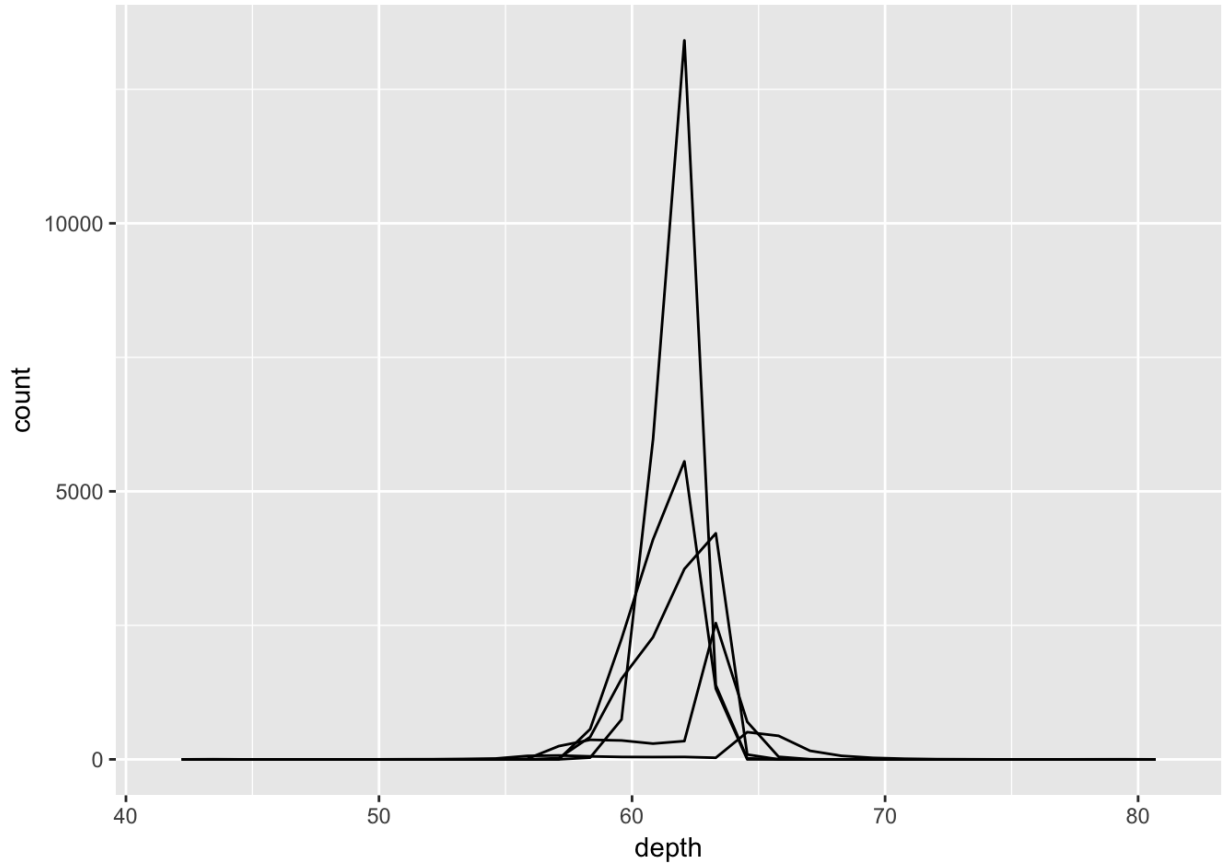
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Now we could really make use of the frequency polygon by breaking that up by cut.

```
ggplot(diamonds, aes(depth, by = cut)) +
    geom_freqpoly()
```

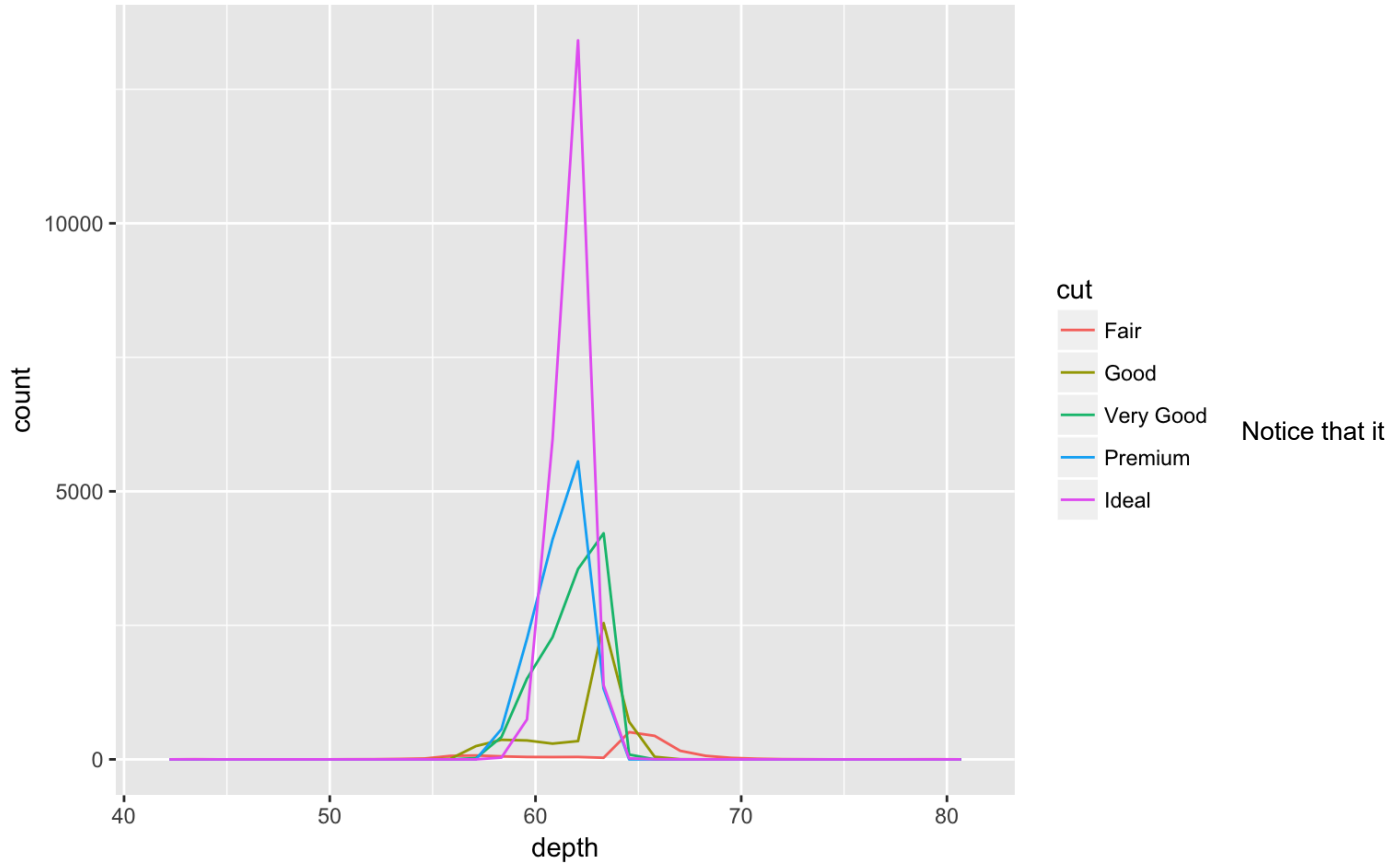
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



To make it more readable, and actually useful, let's make each line (which represents a cut) a different color.

```
ggplot(diamonds, aes(depth, color = cut)) +
    geom_freqpoly()
```

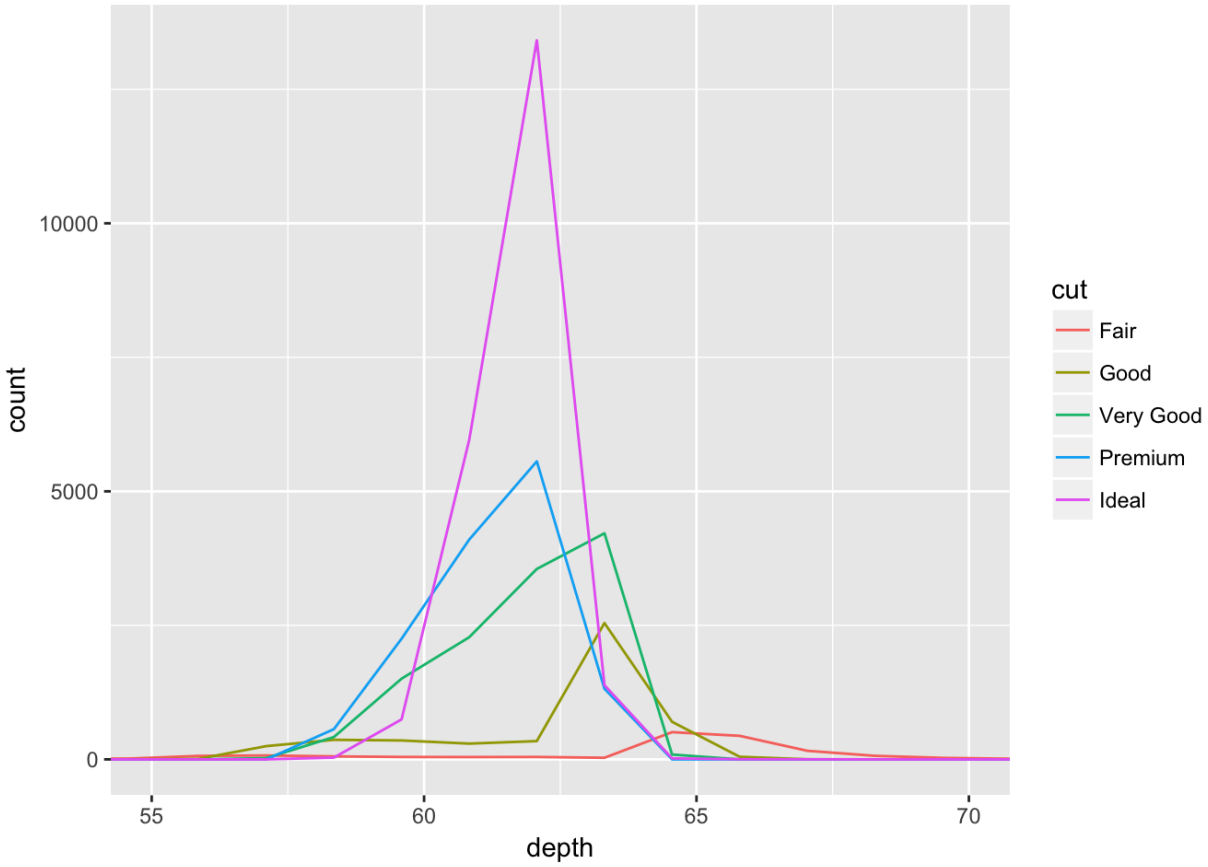
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Notice that it adds a nice legend for us. Let's zoom in a little to make it easier to see the detail.

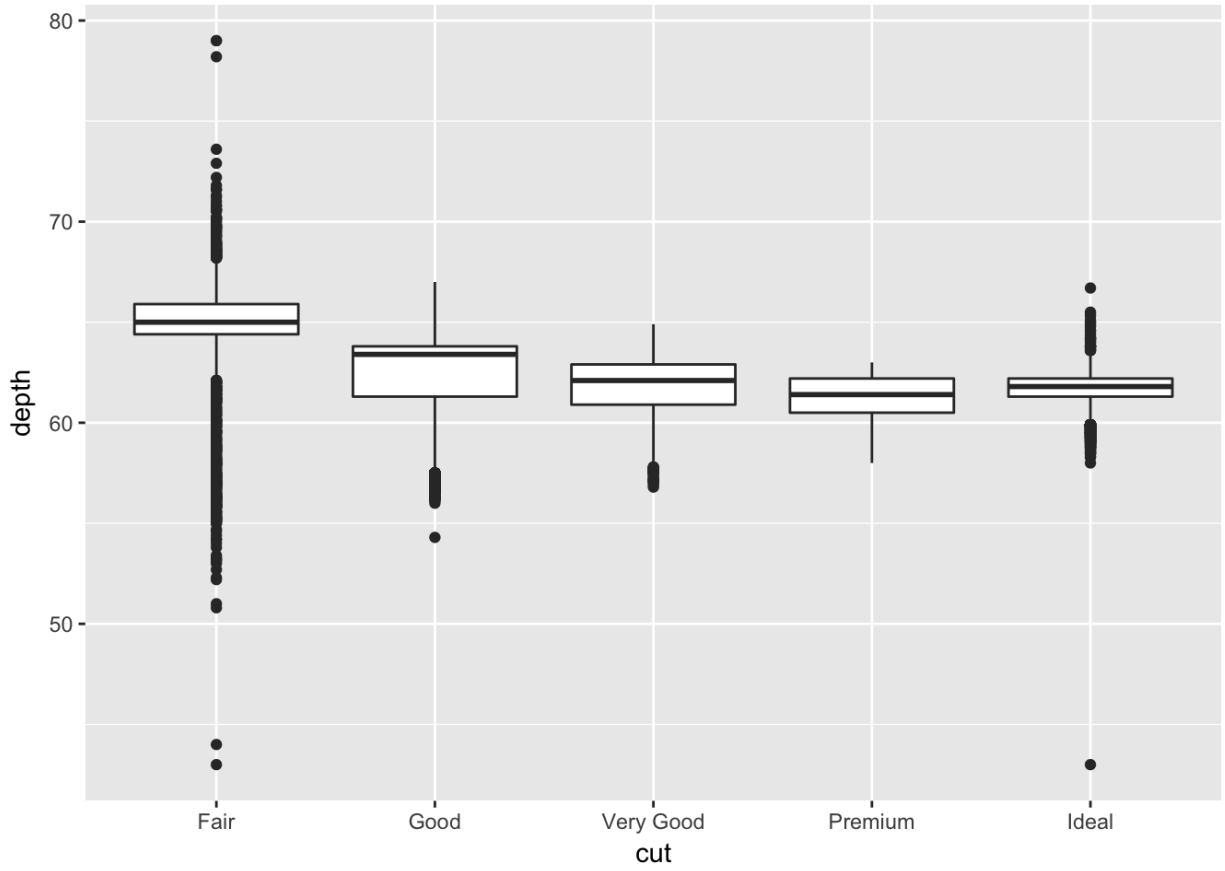
```
ggplot(diamonds, aes(depth, color = cut)) +
    geom_freqpoly() +
    coord_cartesian(xlim = c(55, 70))
```

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



While we're here let's look at one more quick way to look at distributions, the boxplot. Notice the order of the variables.

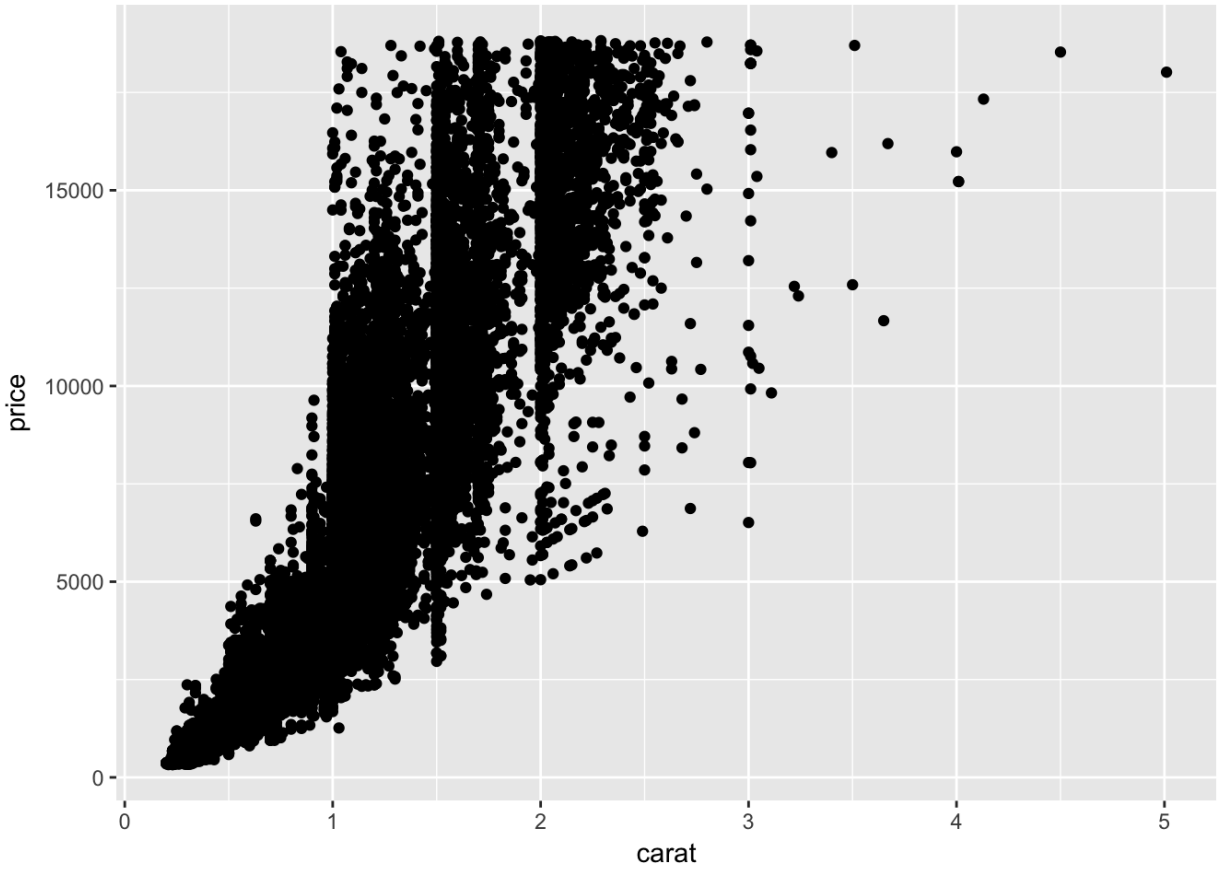
```
ggplot(diamonds, aes(cut, depth)) +
    geom_boxplot()
```



So far we've explored distributions by looking at histograms, frequency polygons, and boxplots. We've modified the bins and zoomed by modifying the coordinate system. We've broken things down by the cut variable.

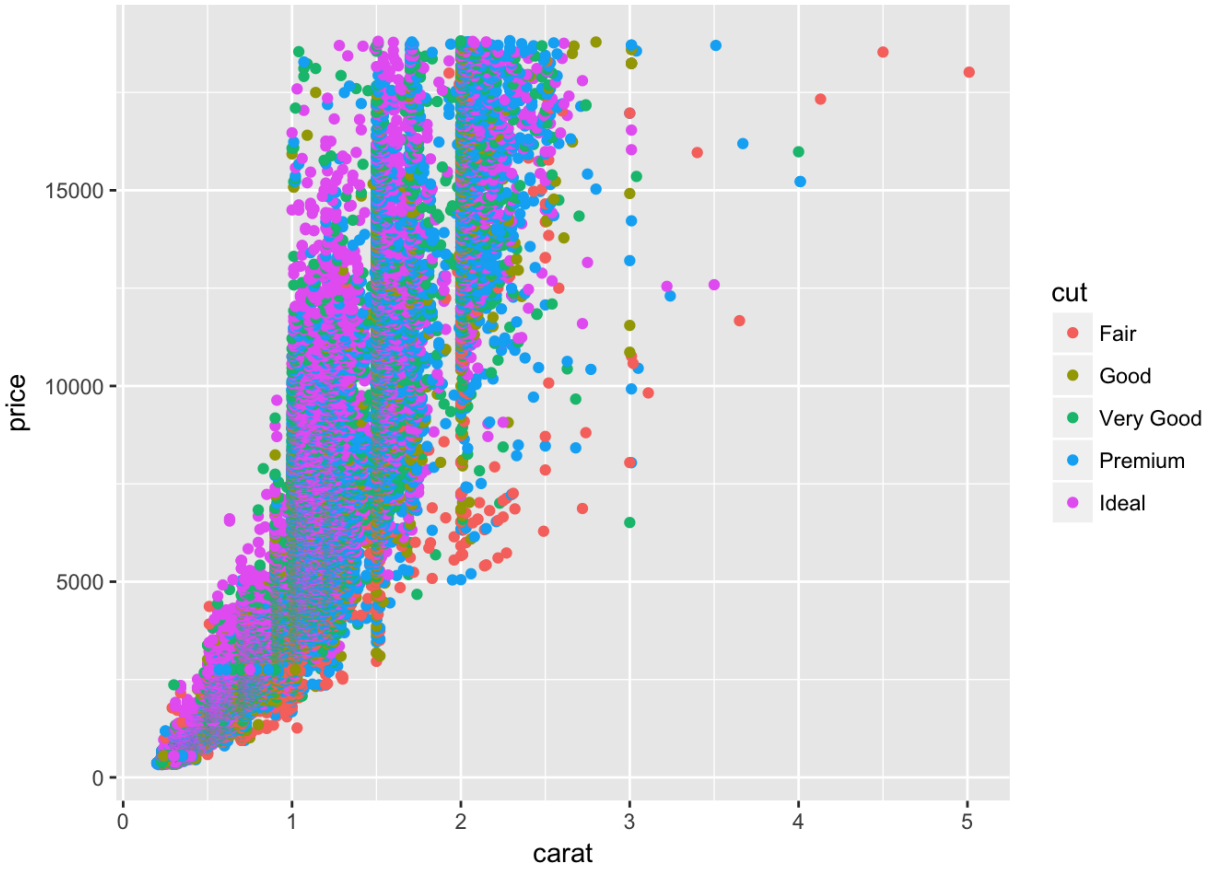
Let's switch things up just a little and look at the relationship between two variables. A scatterplot is a great way to begin there. It seems logical that there might be a relationship between carat and price, so let's start there.

```
ggplot(diamonds, aes(x = carat, y = price)) +
    geom_point()
```



I wonder how cut plays into the relationship. Let's see if we can find a way to display it as well. This is called mapping by a variable. Notice that the new argument is inside the `aes()` function. Everything that happens inside the `aes()` function is mapping variables to aesthetics.

```
ggplot(diamonds, aes(carat, price, color = cut)) +
    geom_point()
```



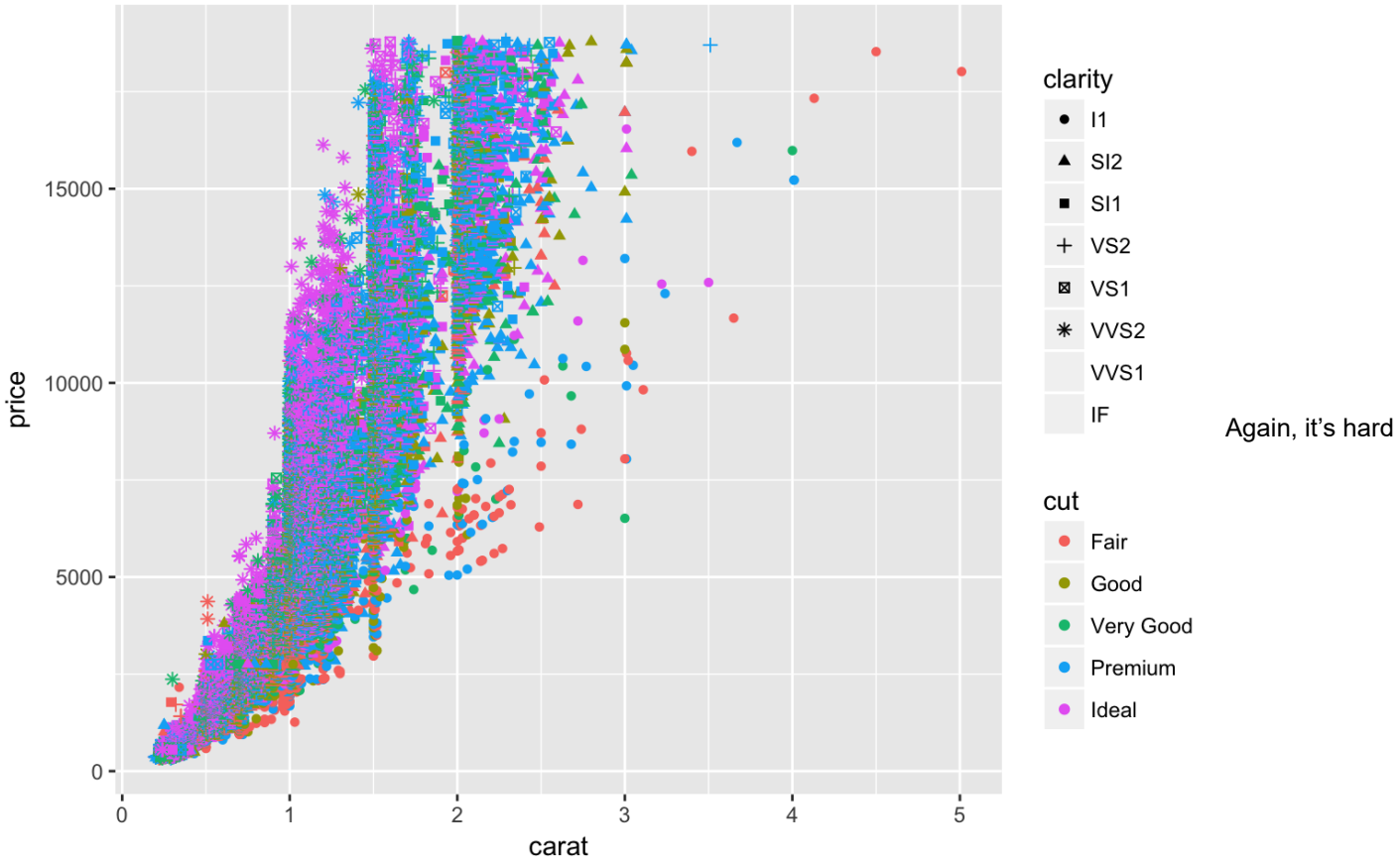
Hard to tell much in the middle, but the concentration red or orange on the bottom right indicates that “Fair” cut diamonds go for a lower price. Same thing for “Ideal” cuts on the high end.

I think it will be hard to see enough detail to get any information from it, but let’s try adding another variable. I wonder how clarity affects the relationship?

```
ggplot(diamonds, aes(x = carat, y = price, color = cut, shape = clarity)) +
    geom_point()
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values
## because more than 6 becomes difficult to discriminate; you have 8.
## Consider specifying shapes manually if you must have them.
```

```
## Warning: Removed 5445 rows containing missing values (geom_point).
```

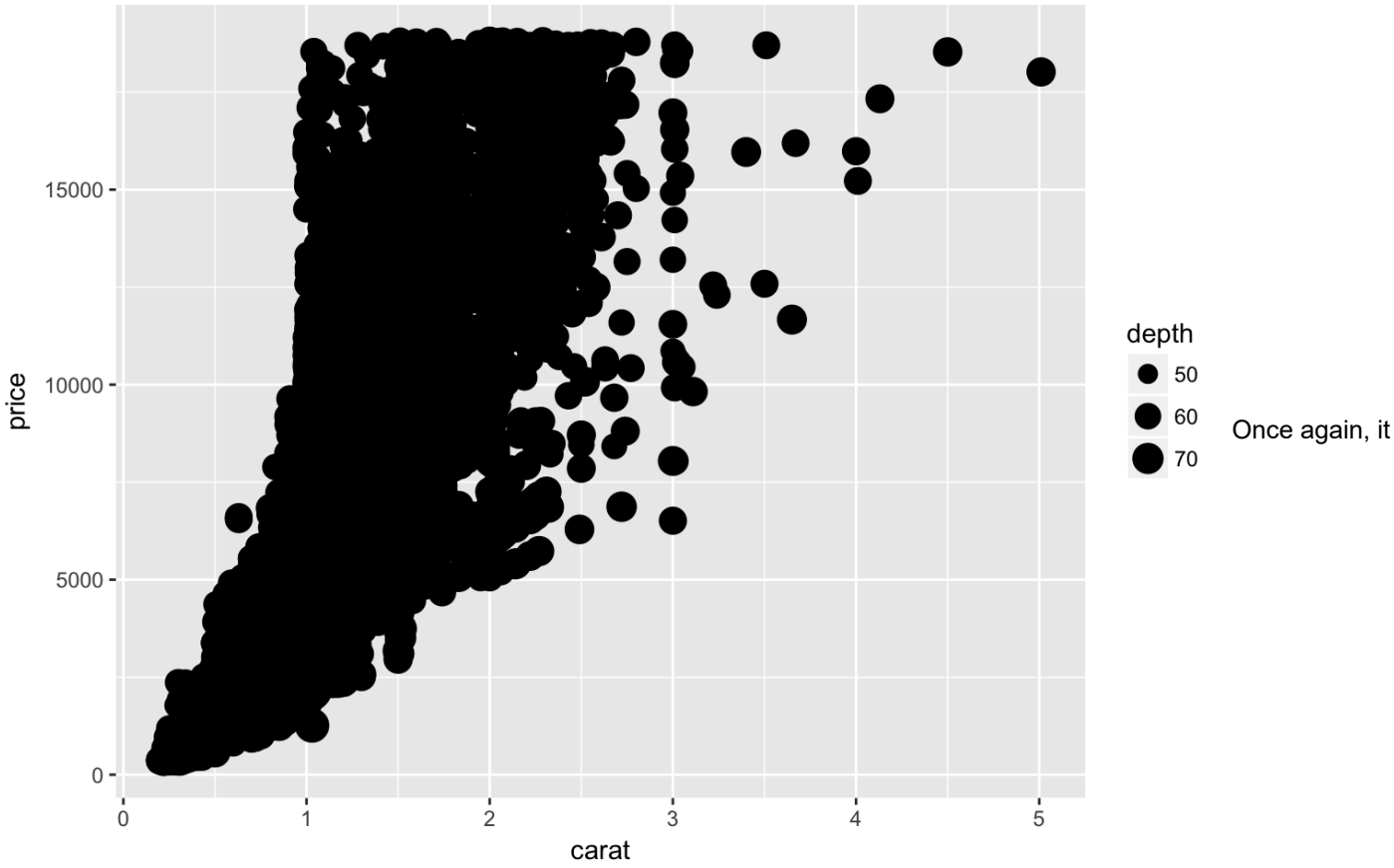


Again, it's hard to tell much from this particular chart at this point, but you can see how each point now has a shape that relates to its clarity rating. The warning message makes this point as well. But you can see here the option we have to map the shape of the points by a discrete or factor variable.

But what about continuous variables? We can map them as well. Let's take a look at depth's effect on the relationship.

One way is to map the size of each point to the continuous variable.

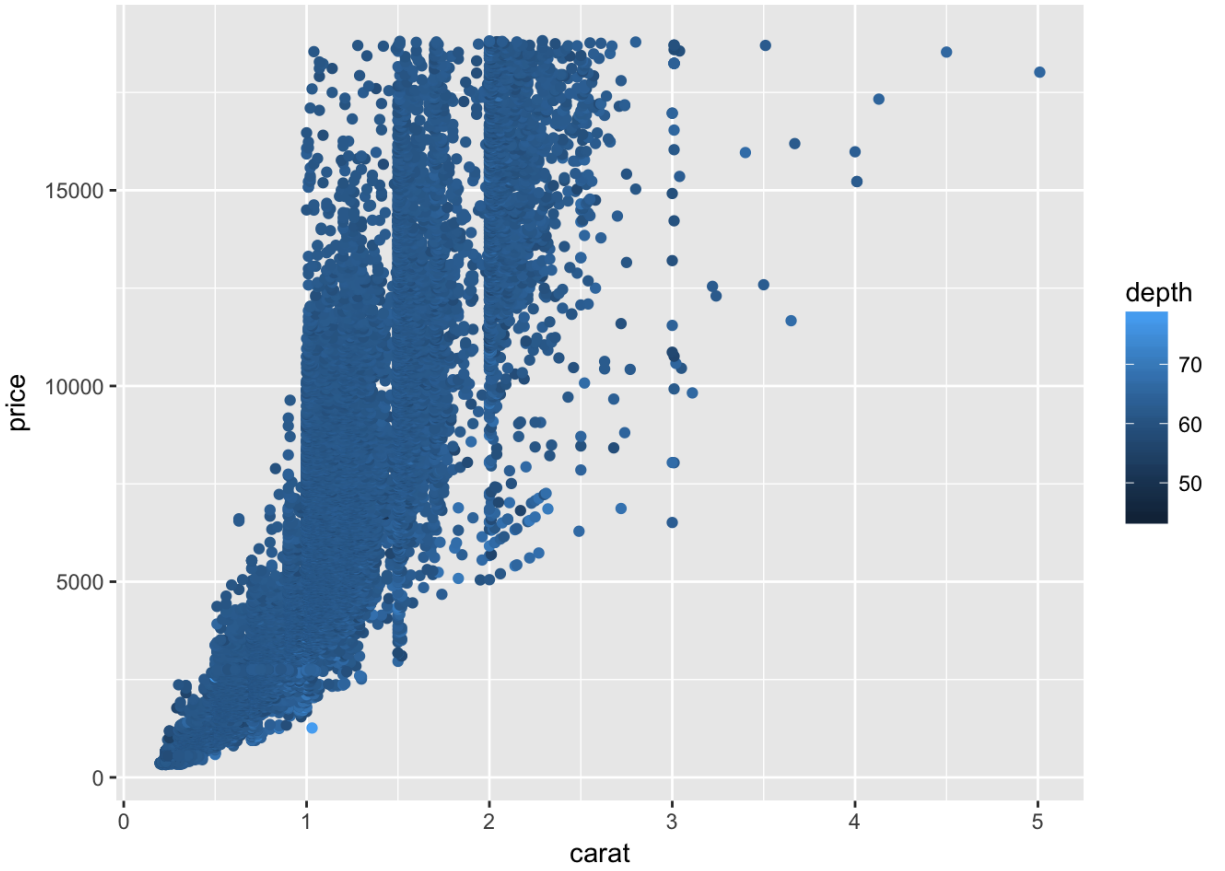
```
ggplot(diamonds, aes(carat, price, size = depth)) +
    geom_point()
```



Once again, it is very difficult to see due to the fact that we are looking at over 50,000 points, but the higher the depth, the larger the point.

Another way to map depth is one we used for discrete or factor variables: color.

```
ggplot(diamonds, aes(carat, price, color = depth)) +
    geom_point()
```

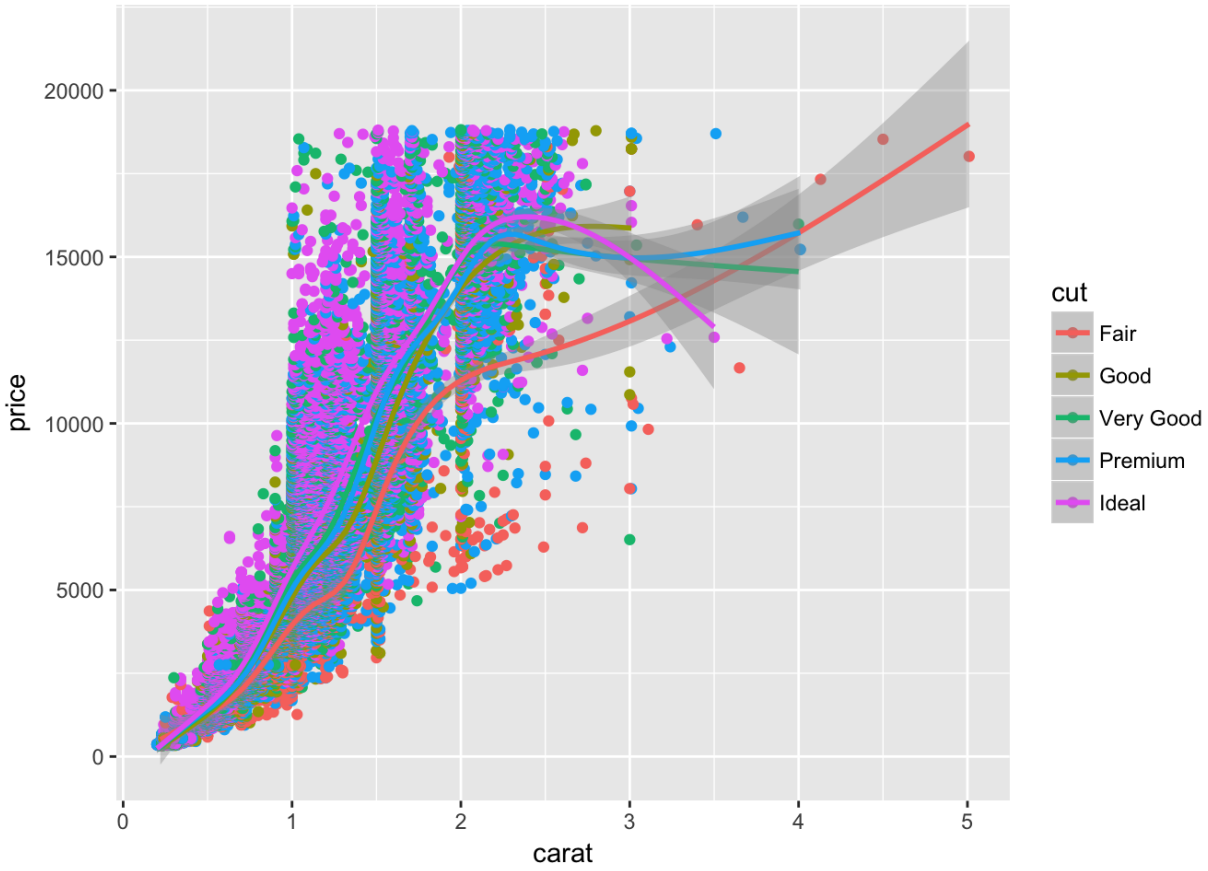


Notice that it uses a scale from blue to black to indicate the depth score.

Let's take a quick look at layers and information that we can add. It looks to me like there is a positive correlation between carat and price. We can easily add a smooth to illustrate that trend. I thought the plot where we mapped cut to color was the most informative so far, so we'll use that one.

```
ggplot(diamonds, aes(carat, price, color = cut)) +
    geom_point() +
    geom_smooth()
```

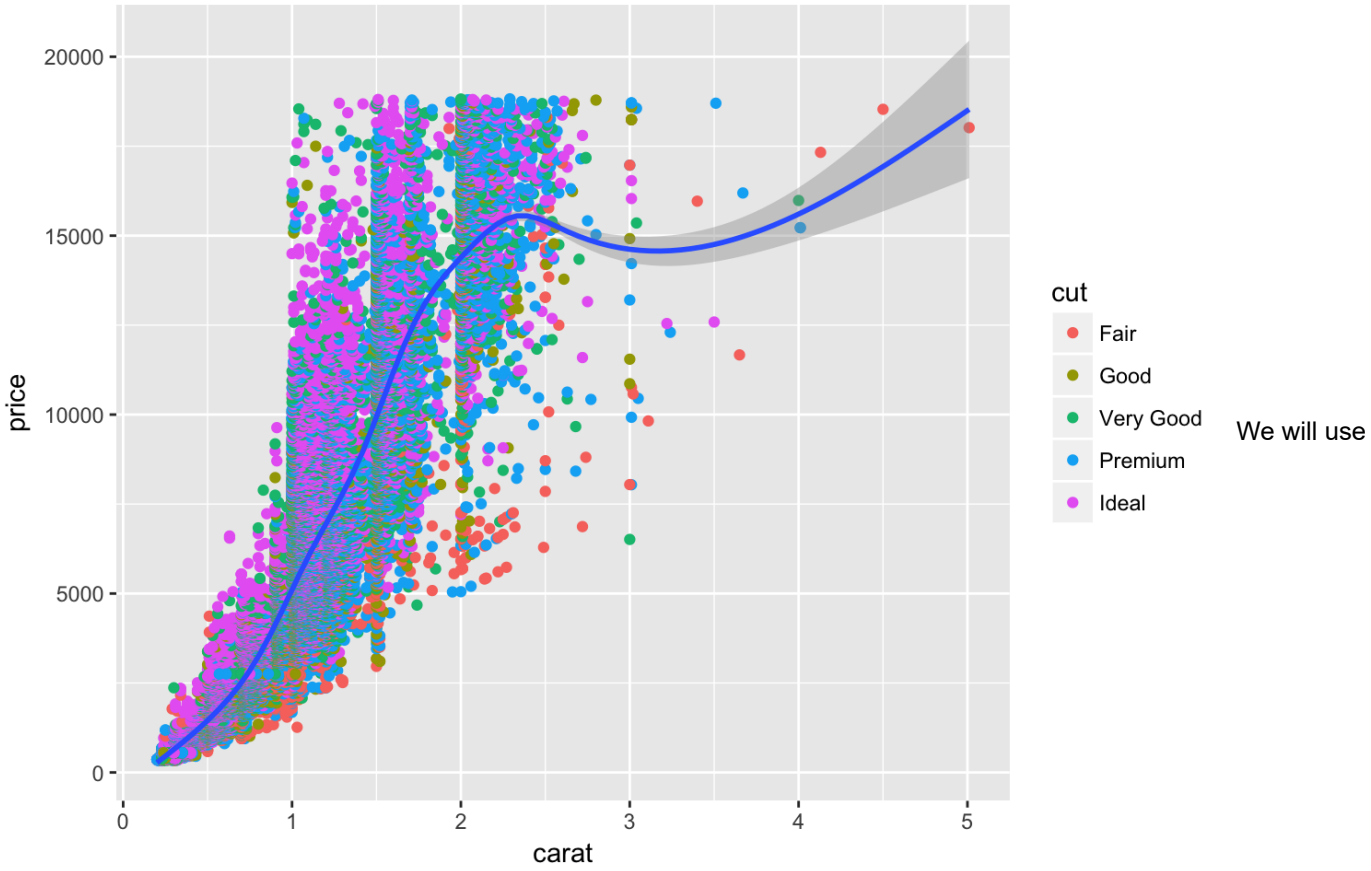
```
## `geom_smooth()` using method = 'gam'
```



Notice that it made a different smooth for each cut. If you remember from earlier, any data or aesthetics we specify in the `ggplot()` function call become the default for the whole plot. We can override that by doing the same thing within the function call for the geom. Because we don't want color to be the default for the smooth, we will specify it in the point geom.

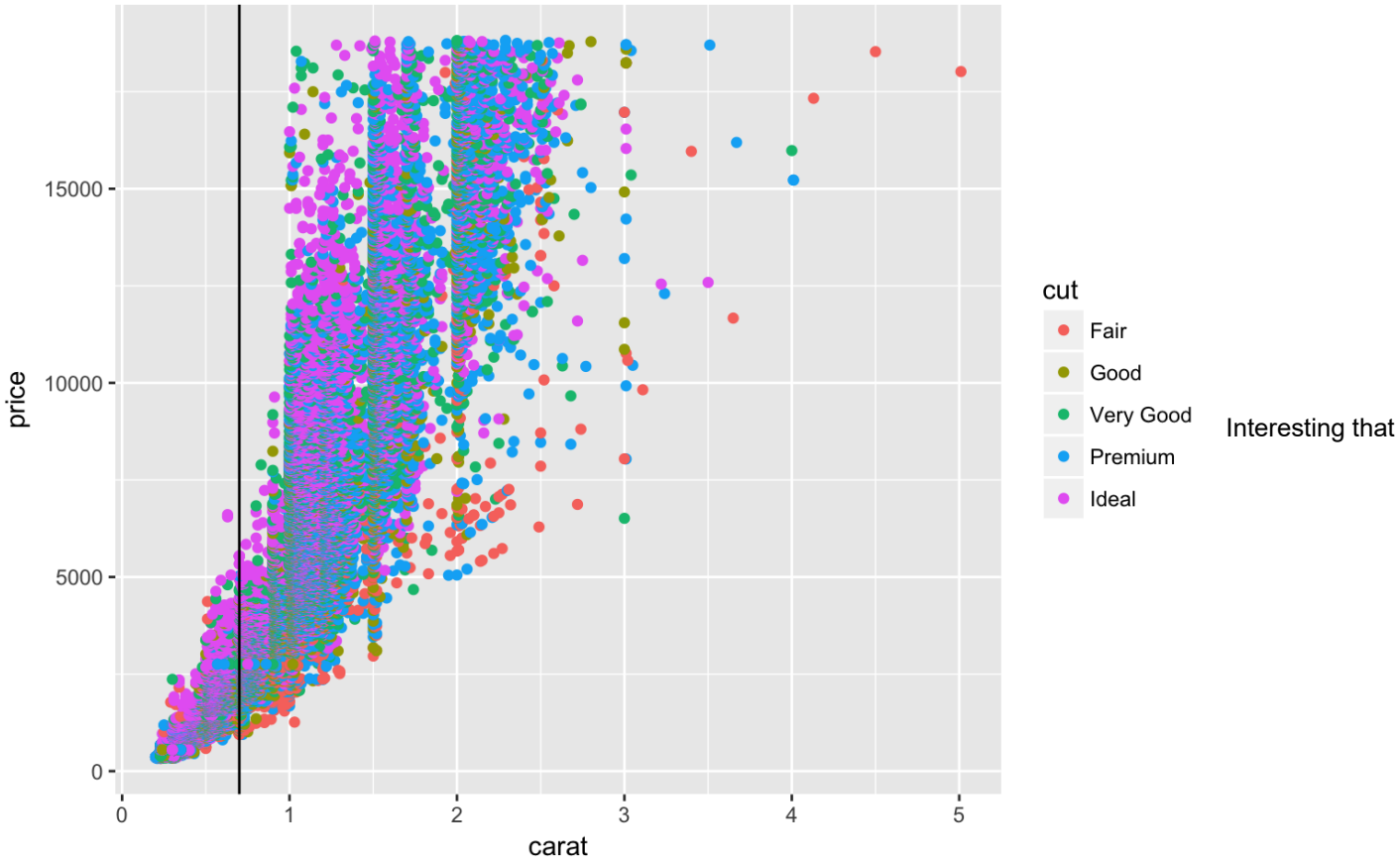
```
ggplot(diamonds, aes(carat, price)) +
    geom_point(aes(color = cut)) +
    geom_smooth()
```

```
## `geom_smooth()` using method = 'gam'
```



many other layers and additions to graphs during the rest of the lesson, but here's one more that I find particularly useful: a line that represents the median of one of the variables.

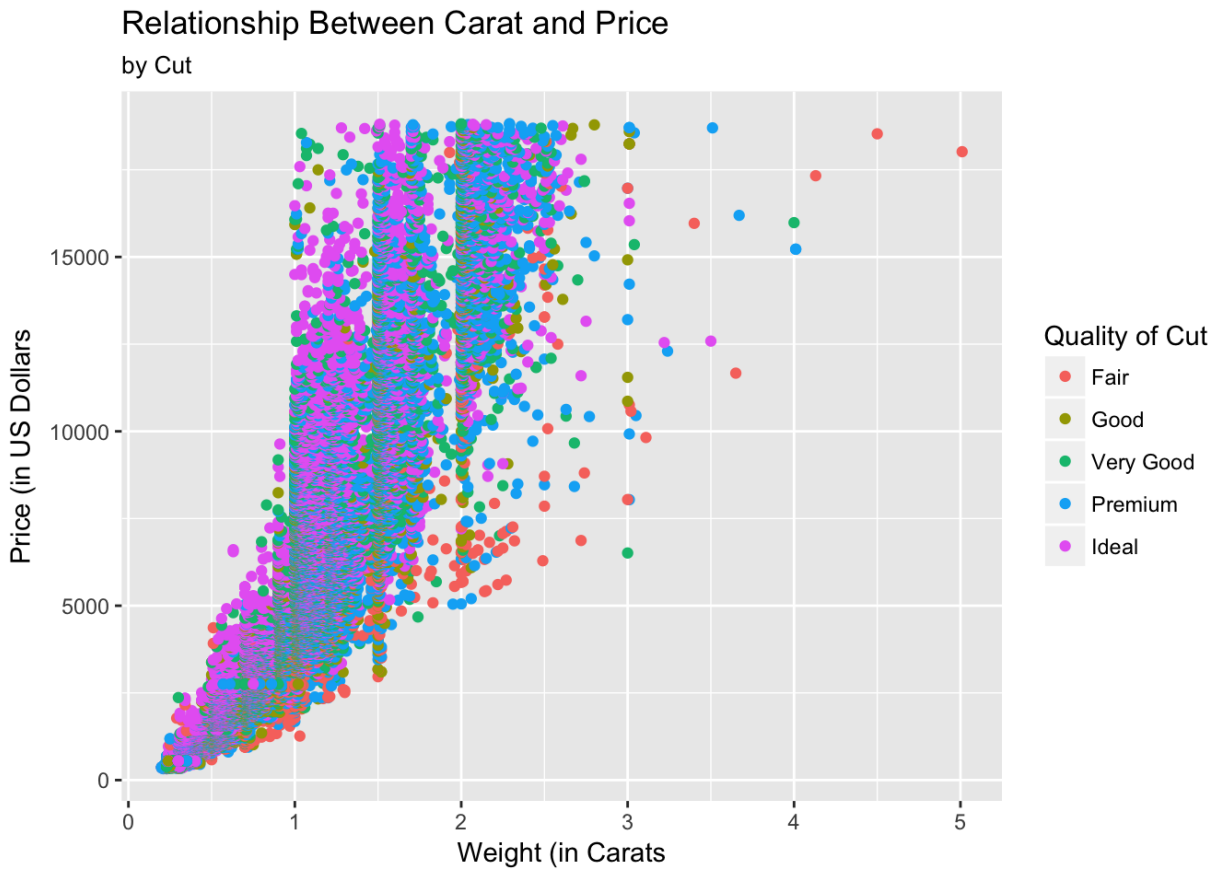
```
ggplot(diamonds, aes(carat, price, color = cut)) +
    geom_point() +
    geom_vline(aes(xintercept = median(carat)))
```



it is so far to the left. The data must be highly concentrated on the low end of carat, which is difficult to tell from this graph. If we were doing an in-depth of the diamonds dataset, we could create another graph or two to explore the variable, perhaps beginning with a histogram of carat.

One last topic I want to cover on our quick tour of ggplot: chart titles and axis labels. I mentioned earlier that we are more concerned at this stage about making our plots informative rather than pretty (we'll take care of pretty later), but I find that a well labeled chart can be more informative. Here's a simple example of each:

```
ggplot(diamonds, aes(carat, price, color = cut)) +
    geom_point() +
    ggtitle("Relationship Between Carat and Price", subtitle = "by Cut") + # label = ""
    xlab("Weight (in Carats") +
    ylab("Price (in US Dollars") +
    labs(color = "Quality of Cut") # The legend title
```



We covered a lot during our quick tour of ggplot2. Throughout the rest of this phase, we will look at many more geoms and ways to customize the chart and transform the data so that it conveys the information that we need. We will go into much more depth on each of these features, and I assure you that you will be very comfortable with the primary tools of the ggplot package. Right now, you may not be able to create every chart that you can conceive, but I encourage you to explore a little bit and see what you can come up with.

In addition to the diamonds dataset, ggplot and the base R package called “datasets” have many built in datasets that you can experiment with. Some of the more popular ones include “mpg”, “iris”, and “economics”. You can find more by running the command `data()` in the console (no arguments needed). Remember that you can get more information about any of these preloaded datasets by running `?name_of_dataset` in the console.

Spend some time and play around. Experiment a little. Not every graph will turn out exactly how you intended it, but see if you can tweak your code to make it do what you want. Regardless of how skilled you are, there is always trial and error in your R programming, especially during EDA.

Now that we know a little about `ggplot2` and the “Grammar of Graphics”, in the next lesson we will learn about `dplyr` and the “Grammar of Data Manipulation”. We will use it for most changes we make to our dataset, from permanent additions like calculating a new variable, to quickly looking only at the largest values for a single variable.

dplyr, and the Grammar of Data Transformation

In addition to making graphs, we will spend a lot of time during EDA working with the data itself. It goes without saying that we will be exploring the data throughout this phase, but we will do that in quite a few ways. We might break the data into parts and look at summaries of each, or we might zoom in on certain subsets of the data, or we might create new variables that will help with our analysis.

In order to do such manipulations and transformations, we will use yet another of Hadley Wickham’s excellent tools: dplyr. dplyr provides us with a set of very simple, intuitive tools to handle most of the transformation tasks we will run into. There are six primary functions in the package. When used in conjunction with each other, the options for transforming your data are endless. These six functions make up the core of the “Grammar of Data Transformation”.

The Grammar of Data Tranformation

- filter() allows us to look at a subset of our data by filtering out what we don't need
- arrange() sorts data in ascending or descending order of whatever criteria we choose
- select() displays only the columns or variables that we ask for
- mutate() creates new columns
- summarise() gives us whatever summary statistics we ask of it
- group_by() makes each of the above functions more useful by breaking the data into groups of our choosing (Wickham & Grolemund, 2017)

Each of these are both very powerful and simple to use. They all use similar syntax and are designed to mesh well when used together. Note that the "grammar" is full of verbs. R is a "functional" language and relies more on verbs than nouns.

Each of these functions have a similar structure. The first argument is always the dataframe the function will be applied to, and the rest of the arguments are the conditions by which we apply it.

```
args(filter)
```

```
## function (.data, ...)
## NULL
```

```
args(arrange)
```

```
## function (.data, ...)
## NULL
```

```
args(select)
```

```
## function (.data, ...)
## NULL
```

```
args(mutate)
```

```
## function (.data, ...)
## NULL
```

```
args(summarise)
```

```
## function (.data, ...)
## NULL
```

```
args(group_by)
```

```
## function (.data, ..., add = FALSE)
## NULL
```

Let's do a few quick examples. We can go back to the diamonds dataset.

```
diamonds <- diamonds
colnames(diamonds)
```

```
##  [1] "carat"   "cut"     "color"   "clarity" "depth"   "table"   "price"
##  [8] "x"       "y"       "z"
```

```
# Pull up only the Ideal cut diamonds
filter(diamonds, cut == "Ideal")
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
0.23	Ideal	J	VS1	62.8	56.0	340	3.93	3.90	2.46
0.31	Ideal	J	SI2	62.2	54.0	344	4.35	4.37	2.71
0.30	Ideal	I	SI2	62.0	54.0	348	4.31	4.34	2.68
0.33	Ideal	I	SI2	61.8	55.0	403	4.49	4.51	2.78
0.33	Ideal	I	SI2	61.2	56.0	403	4.49	4.50	2.75
0.33	Ideal	J	SI1	61.1	56.0	403	4.49	4.55	2.76
0.23	Ideal	G	VS1	61.9	54.0	404	3.93	3.95	2.44
0.32	Ideal	I	SI1	60.9	55.0	404	4.45	4.48	2.72
0.30	Ideal	I	SI2	61.0	59.0	405	4.30	4.33	2.63

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
arrange(diamonds, price)
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
arrange(diamonds, desc(price))
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
2.29	Premium	I	VS2	60.8	60.0	18823	8.50	8.47	5.16
2.00	Very Good	G	SI1	63.5	56.0	18818	7.90	7.97	5.04
1.51	Ideal	G	IF	61.7	55.0	18806	7.37	7.41	4.56
2.07	Ideal	G	SI2	62.5	55.0	18804	8.20	8.13	5.11
2.00	Very Good	H	SI1	62.8	57.0	18803	7.95	8.00	5.01
2.29	Premium	I	SI1	61.8	59.0	18797	8.52	8.45	5.24
2.04	Premium	H	SI1	58.1	60.0	18795	8.37	8.28	4.84
2.00	Premium	I	VS1	60.8	59.0	18795	8.13	8.02	4.91
1.71	Premium	F	VS2	62.3	59.0	18791	7.57	7.53	4.70
2.15	Ideal	G	SI2	62.6	54.0	18791	8.29	8.35	5.21

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
select(diamonds, carat, cut, price)
```

carat <dbl>	cut <ord>	price <int>
0.23	Ideal	326
0.21	Premium	326
0.23	Good	327
0.29	Premium	334
0.31	Good	335
0.24	Very Good	336
0.24	Very Good	336
0.26	Very Good	337
0.22	Fair	337
0.23	Very Good	338

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
mutate(diamonds, price_per_carat = price / carat)
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39

1-10 of 10,000 rows | 1-10 of 11 columns Previous **1** 2 3 4 5 6 … 1000 Next

Seems simple enough, right? We'll go into more detail into these soon. But the real power of these functions comes from linking them together, but to do so in an efficient way, we need to learn one very important tool. Let's quickly look at a way to link them together with only the tools we have learned so far. And then we'll learn a much better way to do things and never used this one again!

```
diamonds2 <- mutate(diamonds, price_per_carat = price / carat)
diamonds3 <- arrange(diamonds2, desc(price_per_carat))
select(diamonds3, carat, cut, price, price_per_carat)
```

carat <dbl>	cut <ord>	price <int>	price_per_carat <dbl>
1.04	Very Good	18542	17828.846
1.07	Premium	18279	17083.178
1.03	Ideal	17590	17077.670
1.07	Very Good	18114	16928.972
1.02	Very Good	17100	16764.706
1.07	Very Good	17909	16737.383
1.09	Very Good	18231	16725.688
1.00	Ideal	16469	16469.000
1.01	Premium	16234	16073.267
1.00	Very Good	16073	16073.000

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
# Now forget you ever saw that!
rm(diamonds2, diamonds3)
```

See the extra, intermediate variables? We can do away with those and make things much simpler and easier to follow and write. We do so with the pipe `%>%`.

The pipe takes what is on its left and passes to the right. It can be read as “then” when reading your code. What comes from the left becomes and completely replaces the first argument of the function on the right. Since output of all the dplyr verbs is a data frame, and the first argument of each is also a data frame, the pipe works especially well with dplyr. We are effectively passing a data frame to one of the functions, transforming it, then (potentially) passing it to another for further transformation.

Here are a few examples of piping, which also illustrate the power of chaining the dplyr verbs. Remember to read the pipe as “then”, and notice how you are able to always read left to right and down the page, as we normally read.

```
diamonds %>%
    group_by(cut) %>%
    summarise(mean(price))
```

cut <ord>	mean(price) <dbl>
Fair	4358.758
Good	3928.864
Very Good	3981.760
Premium	4584.258
Ideal	3457.542

5 rows

```
diamonds %>%
    filter(carat >= 1) %>%
    group_by(color) %>%
    summarise(mean_depth = mean(depth)) %>%
    arrange(desc(mean_depth))
```

color <ord>	mean_depth <dbl>
J	61.85793
H	61.82380
F	61.78533
G	61.77574
I	61.77546
E	61.71910
D	61.68368

7 rows

You are limited only by your imagination in the way you chain the dplyr functions. A couple quick notes about the pipe: the keyboard shortcut for it is ctrl/cmd + shift + m, and you must load the tidyverse package for the pipe to work, or else the magrittr package if for some reason you weren’t loading the tidyverse.

If this is your first introduction to the pipe, the syntax may seem a little unusual, but you’ll get used to it very quickly. Once you’ve used it for a while it will feel much more natural than doing without. For that reason, I’m going to use the pipe whenever we use the dplyr verbs, even if we only need a single verb, or when other good opportunities to use it present themselves. Use it whenever you practice and it will be second nature in no time.

A Look At Each of The Primary dplyr Verbs

I'll run through each of the verbs, giving any considerations we must keep in mind when we use it and also give an example or two.

filter()

The filter() function allows us to filter out rows. We use the normal set of comparison and logical operators to tell R which rows to filter out. The comparison operators include the following:

Comparison Operators

- `>` (greater than)
- `<` (less than)
- `>=` (greater than or equal to)
- `<=` (less than or equal to)
- `!=` (not equal)
- `==` (equal)

Logical Operators

- `&` (and)
- `|` (or)
- `!` (not)

So let's look at a few examples.

```
diamonds %>%
    filter(cut == "Ideal")
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
0.23	Ideal	J	VS1	62.8	56.0	340	3.93	3.90	2.46
0.31	Ideal	J	SI2	62.2	54.0	344	4.35	4.37	2.71
0.30	Ideal	I	SI2	62.0	54.0	348	4.31	4.34	2.68
0.33	Ideal	I	SI2	61.8	55.0	403	4.49	4.51	2.78
0.33	Ideal	I	SI2	61.2	56.0	403	4.49	4.50	2.75
0.33	Ideal	J	SI1	61.1	56.0	403	4.49	4.55	2.76
0.23	Ideal	G	VS1	61.9	54.0	404	3.93	3.95	2.44
0.32	Ideal	I	SI1	60.9	55.0	404	4.45	4.48	2.72
0.30	Ideal	I	SI2	61.0	59.0	405	4.30	4.33	2.63

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
diamonds %>%
    filter(cut == "Ideal", color == "E")
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
0.26	Ideal	E	VVS2	62.9	58.0	554	4.02	4.06	2.54
0.70	Ideal	E	SI1	62.5	57.0	2757	5.70	5.72	3.57
0.59	Ideal	E	VVS2	62.0	55.0	2761	5.38	5.43	3.35
0.74	Ideal	E	SI2	62.2	56.0	2761	5.80	5.84	3.62
0.70	Ideal	E	VS2	60.7	58.0	2762	5.73	5.76	3.49
0.74	Ideal	E	SI1	62.3	54.0	2762	5.80	5.83	3.62
0.70	Ideal	E	SI1	60.9	57.0	2768	5.73	5.76	3.50
0.60	Ideal	E	VS1	61.7	55.0	2774	5.41	5.44	3.35
0.70	Ideal	E	SI1	62.7	55.0	2774	5.68	5.74	3.58

1-10 of 3,903 rows Previous **1** 2 3 4 5 6 … 391 Next

```
# Same as above
diamonds %>%
    filter(cut == "Ideal" & color == "E")
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
0.26	Ideal	E	VVS2	62.9	58.0	554	4.02	4.06	2.54
0.70	Ideal	E	SI1	62.5	57.0	2757	5.70	5.72	3.57
0.59	Ideal	E	VVS2	62.0	55.0	2761	5.38	5.43	3.35
0.74	Ideal	E	SI2	62.2	56.0	2761	5.80	5.84	3.62
0.70	Ideal	E	VS2	60.7	58.0	2762	5.73	5.76	3.49
0.74	Ideal	E	SI1	62.3	54.0	2762	5.80	5.83	3.62
0.70	Ideal	E	SI1	60.9	57.0	2768	5.73	5.76	3.50
0.60	Ideal	E	VS1	61.7	55.0	2774	5.41	5.44	3.35
0.70	Ideal	E	SI1	62.7	55.0	2774	5.68	5.74	3.58

1-10 of 3,903 rows Previous **1** 2 3 4 5 6 … 391 Next

```
diamonds %>%
    filter(price >= 15000)
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
1.60	Ideal	G	VS2	61.9	56.0	15000	7.53	7.47	4.64

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
1.54	Premium	E	VS2	62.3	58.0	15002	7.31	7.39	4.58
1.19	Ideal	F	VVS1	61.5	55.0	15005	6.82	6.84	4.20
2.10	Premium	I	SI1	61.5	57.0	15007	8.25	8.21	5.06
1.69	Ideal	D	SI1	60.8	57.0	15011	7.69	7.71	4.68
1.50	Very Good	G	VVS2	62.9	56.0	15013	7.22	7.32	4.57
1.73	Very Good	G	VS1	62.8	57.0	15014	7.57	7.72	4.80
2.02	Premium	G	SI2	63.0	59.0	15014	8.05	7.95	5.03
2.05	Very Good	F	SI2	61.9	56.0	15017	8.13	8.18	5.05
1.50	Very Good	F	VS1	61.6	58.0	15022	7.35	7.43	4.55

1-10 of 1,656 rows Previous **1** 2 3 4 5 6 … 166 Next

```
diamonds %>%
    filter(carat <= 1)
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
diamonds %>%
    filter(carat > 2, price < 6000)
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
2.06	Premium	J	I1	61.2	58	5203	8.10	8.07	4.95
2.14	Fair	J	I1	69.4	57	5405	7.74	7.70	5.36
2.15	Fair	J	I1	65.5	57	5430	8.01	7.95	5.23

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
2.22	Fair	J	I1	66.7	56	5607	8.04	8.02	5.36
2.01	Fair	I	I1	67.4	58	5696	7.71	7.64	5.17
2.01	Fair	I	I1	55.9	64	5696	8.48	8.39	4.71
2.27	Fair	J	I1	67.6	55	5733	8.05	8.00	5.43

7 rows

```
diamonds %>%
    filter(carat > 3 | cut == "Ideal")
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
0.23	Ideal	J	VS1	62.8	56.0	340	3.93	3.90	2.46
0.31	Ideal	J	SI2	62.2	54.0	344	4.35	4.37	2.71
0.30	Ideal	I	SI2	62.0	54.0	348	4.31	4.34	2.68
0.33	Ideal	I	SI2	61.8	55.0	403	4.49	4.51	2.78
0.33	Ideal	I	SI2	61.2	56.0	403	4.49	4.50	2.75
0.33	Ideal	J	SI1	61.1	56.0	403	4.49	4.55	2.76
0.23	Ideal	G	VS1	61.9	54.0	404	3.93	3.95	2.44
0.32	Ideal	I	SI1	60.9	55.0	404	4.45	4.48	2.72
0.30	Ideal	I	SI2	61.0	59.0	405	4.30	4.33	2.63

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
diamonds %>%
    filter(!(clarity == "I1" | color == "J"))
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.22	Premium	F	SI1	60.4	61.0	342	3.88	3.84	2.33
0.20	Premium	E	SI2	60.2	62.0	345	3.79	3.75	2.27

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

arrange()

The arrange() function allows us to reorder the data based on the ascending or descending values of one or more variables. Ascending is the default order.

Here are a few examples:

```
diamonds %>%
    arrange(price)
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
diamonds %>%
    arrange(desc(price))
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
2.29	Premium	I	VS2	60.8	60.0	18823	8.50	8.47	5.16
2.00	Very Good	G	SI1	63.5	56.0	18818	7.90	7.97	5.04
1.51	Ideal	G	IF	61.7	55.0	18806	7.37	7.41	4.56
2.07	Ideal	G	SI2	62.5	55.0	18804	8.20	8.13	5.11
2.00	Very Good	H	SI1	62.8	57.0	18803	7.95	8.00	5.01

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
2.29	Premium	I	SI1	61.8	59.0	18797	8.52	8.45	5.24
2.04	Premium	H	SI1	58.1	60.0	18795	8.37	8.28	4.84
2.00	Premium	I	VS1	60.8	59.0	18795	8.13	8.02	4.91
1.71	Premium	F	VS2	62.3	59.0	18791	7.57	7.53	4.70
2.15	Ideal	G	SI2	62.6	54.0	18791	8.29	8.35	5.21

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
diamonds %>%
    arrange(desc(cut), color, desc(clarity))
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.51	Ideal	D	IF	62.0	56.0	3446	5.14	5.18	3.20
0.51	Ideal	D	IF	62.1	55.0	3446	5.12	5.13	3.19
0.53	Ideal	D	IF	61.5	54.0	3517	5.27	5.21	3.22
0.53	Ideal	D	IF	62.2	55.0	3812	5.17	5.19	3.22
0.63	Ideal	D	IF	61.2	53.0	3832	5.55	5.60	3.41
0.59	Ideal	D	IF	60.7	58.0	4161	5.45	5.49	3.32
0.59	Ideal	D	IF	60.9	57.0	4208	5.40	5.43	3.30
0.56	Ideal	D	IF	62.4	56.0	4216	5.24	5.28	3.28
0.56	Ideal	D	IF	61.9	57.0	4293	5.28	5.31	3.28
0.56	Ideal	D	IF	60.8	58.0	4632	5.35	5.31	3.24

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
# color is in ascending order because R does not know that the levels are in reverse alphabetical order.
```

select()

The select() function allows us to reorder rows or display only a selection of columns. They will be displayed in the order they are referenced.

```
diamonds %>%
    select(price, carat, cut)
```

price <int>	carat <dbl>	cut <ord>
326	0.23	Ideal
326	0.21	Premium

price <int>	carat <dbl>	cut <ord>
327	0.23	Good
334	0.29	Premium
335	0.31	Good
336	0.24	Very Good
336	0.24	Very Good
337	0.26	Very Good
337	0.22	Fair
338	0.23	Very Good

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
diamonds %>%
    select(price, carat, cut, everything())
```

price <int>	carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	x <dbl>	y <dbl>	z <dbl>
326	0.23	Ideal	E	SI2	61.5	55.0	3.95	3.98	2.43
326	0.21	Premium	E	SI1	59.8	61.0	3.89	3.84	2.31
327	0.23	Good	E	VS1	56.9	65.0	4.05	4.07	2.31
334	0.29	Premium	I	VS2	62.4	58.0	4.20	4.23	2.63
335	0.31	Good	J	SI2	63.3	58.0	4.34	4.35	2.75
336	0.24	Very Good	J	VVS2	62.8	57.0	3.94	3.96	2.48
336	0.24	Very Good	I	VVS1	62.3	57.0	3.95	3.98	2.47
337	0.26	Very Good	H	SI1	61.9	55.0	4.07	4.11	2.53
337	0.22	Fair	E	VS2	65.1	61.0	3.87	3.78	2.49
338	0.23	Very Good	H	VS1	59.4	61.0	4.00	4.05	2.39

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
diamonds %>%
    select(-price)
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	x <dbl>	y <dbl>	z <dbl>
0.23	Ideal	E	SI2	61.5	55.0	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61.0	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65.0	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58.0	4.20	4.23	2.63

carat <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	x <dbl>	y <dbl>	z <dbl>
0.31	Good	J	SI2	63.3	58.0	4.34	4.35	2.75
0.24	Very Good	J	VVS2	62.8	57.0	3.94	3.96	2.48
0.24	Very Good	I	VVS1	62.3	57.0	3.95	3.98	2.47
0.26	Very Good	H	SI1	61.9	55.0	4.07	4.11	2.53
0.22	Fair	E	VS2	65.1	61.0	3.87	3.78	2.49
0.23	Very Good	H	VS1	59.4	61.0	4.00	4.05	2.39

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
diamonds %>%
    select(carat:clarity)
```

carat <dbl>	cut <ord>	color <ord>	clarity <ord>
0.23	Ideal	E	SI2
0.21	Premium	E	SI1
0.23	Good	E	VS1
0.29	Premium	I	VS2
0.31	Good	J	SI2
0.24	Very Good	J	VVS2
0.24	Very Good	I	VVS1
0.26	Very Good	H	SI1
0.22	Fair	E	VS2
0.23	Very Good	H	VS1

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
diamonds %>%
    select(price, carat:clarity)
```

price <int>	carat <dbl>	cut <ord>	color <ord>	clarity <ord>
326	0.23	Ideal	E	SI2
326	0.21	Premium	E	SI1
327	0.23	Good	E	VS1
334	0.29	Premium	I	VS2
335	0.31	Good	J	SI2
336	0.24	Very Good	J	VVS2

price <int>	carat <dbl>	cut <ord>	color <ord>	clarity <ord>
336	0.24	Very Good	I	VVS1
337	0.26	Very Good	H	SI1
337	0.22	Fair	E	VS2
338	0.23	Very Good	H	VS1

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

Note the use of the `everything()` function, which tells dplyr to include whatever columns are left.

While we're working with `select()`, let's put it to use on our project. If you remember from Phase 1 that while we were inspecting our dataset, we came across a few empty columns, or columns whose variables were all "NA". Let's re-run the check for NA's we ran in Phase 1:

```
colSums(is.na(ks))
```

```
##                     id                  photo                   name
##                      0                      0                      0
##                  blurb                   goal                pledged
##                      0                      0                      0
##                  state                   slug                country
##                      0                      0                      0
##               currency currency_trailing_code               deadline
##                      0                      0                      0
##       state_changed_at             created_at            launched_at
##                      0                      0                      0
##             staff_pick           is_starrable         backers_count
##                      0                      0                      0
##        static_usd_rate            usd_pledged                creator
##                      0                      0                      0
##               location               category                profile
##                      2                      0                      0
##              spotlight                   urls             source_url
##                      0                      0                      0
##                friends             is_starred             is_backing
##                    500                    500                    500
##            permissions
##                    500
```

Note the last four variables, "friends", "is_starred", "is_backing", and "permissions". Each are showing that all of their values equal "NA", so we can remove them. Even though we can undo it by reverting to our original dataset, I like to confirm things before I start deleting data.

```
ks %>%
    select(friends, is_starred, is_backing, permissions)
```

friends <chr>	is_starred <chr>	is_backing <chr>	permissions <chr>
NA	*NA*	*NA*	*NA*
NA	*NA*	*NA*	*NA*

friends <chr>	is_starred <chr>	is_backing <chr>	permissions <chr>
NA	*NA*	*NA*	*NA*
NA	*NA*	*NA*	*NA*
NA	*NA*	*NA*	*NA*
NA	*NA*	*NA*	*NA*
NA	*NA*	*NA*	*NA*
NA	*NA*	*NA*	*NA*
NA	*NA*	*NA*	*NA*
NA	*NA*	*NA*	*NA*

1-10 of 500 rows Previous **1** 2 3 4 5 6 … 50 Next

Clicking through there, it looks like everything is, in fact, empty or equal to “NA”. We can delete those columns.

Before we run the following code, look in the environment pane at the number of variables for our dataset, “ks”. It currently shows 31.

```
(ks <- ks %>%
    select(-c(friends, is_starred, is_backing, permissions)))
```

id <int>
832570985
1054135507
1567381435
1329110416
1403665672
568986755
468138435
305131391
1098019088
700642359

1-10 of 500 rows | 1-1 of 27 columns Previous **1** 2 3 4 5 6 … 50 Next

Now look at the number of variables; it has now changed to 27. Success! We can also click through the output to confirm that the empty columns are not there.

Don’t forget that since we just modified our dataset, we need to update our codebook. Remember that since we are using a separate codebook, we need to make all of our changes to ks in the codebook. We’ll do that now.

rename()

The rename() function is a variant of the select() function that allows us to easily rename a column/variable.

```
diamonds %>%
    rename(weight = carat)
```

weight <dbl>	cut <ord>	color <ord>	clarity <ord>	depth <dbl>	table <dbl>	price <int>	x <dbl>	y <dbl>	z <dbl>
0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

mutate()

The mutate function allows us to add columns that we calculate from the existing data. Let's first trim a few columns off so that we can see the new columns without scrolling. Note that we could use the pipe to eliminate the need for a new object, but since we'll reuse it, it makes sense to make the copy.

```
(diamonds_trimmed <- diamonds %>%
    select(price, carat, depth, table))
```

price <int>	carat <dbl>	depth <dbl>	table <dbl>
326	0.23	61.5	55.0
326	0.21	59.8	61.0
327	0.23	56.9	65.0
334	0.29	62.4	58.0
335	0.31	63.3	58.0
336	0.24	62.8	57.0
336	0.24	62.3	57.0
337	0.26	61.9	55.0

price <int>	carat <dbl>	depth <dbl>	table <dbl>
337	0.22	65.1	61.0
338	0.23	59.4	61.0

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
# Remember that the parentheses allow us to both assign to a new object and print the results
```

And now for a few examples of what we can do with mutate():

```
diamonds_trimmed %>%
    mutate(price / carat)
```

price <int>	carat <dbl>	depth <dbl>	table <dbl>	price/carat <dbl>
326	0.23	61.5	55.0	1417.391
326	0.21	59.8	61.0	1552.381
327	0.23	56.9	65.0	1421.739
334	0.29	62.4	58.0	1151.724
335	0.31	63.3	58.0	1080.645
336	0.24	62.8	57.0	1400.000
336	0.24	62.3	57.0	1400.000
337	0.26	61.9	55.0	1296.154
337	0.22	65.1	61.0	1531.818
338	0.23	59.4	61.0	1469.565

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
diamonds_trimmed %>%
    mutate(price_per_table = price / table)
```

price <int>	carat <dbl>	depth <dbl>	table <dbl>	price_per_table <dbl>
326	0.23	61.5	55.0	5.927273
326	0.21	59.8	61.0	5.344262
327	0.23	56.9	65.0	5.030769
334	0.29	62.4	58.0	5.758621
335	0.31	63.3	58.0	5.775862
336	0.24	62.8	57.0	5.894737
336	0.24	62.3	57.0	5.894737

price <int>	carat <dbl>	depth <dbl>	table <dbl>	price_per_table <dbl>
337	0.26	61.9	55.0	6.127273
337	0.22	65.1	61.0	5.524590
338	0.23	59.4	61.0	5.540984

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

```
diamonds_trimmed %>%
    mutate(log(price))
```

price <int>	carat <dbl>	depth <dbl>	table <dbl>	log(price) <dbl>
326	0.23	61.5	55.0	5.786897
326	0.21	59.8	61.0	5.786897
327	0.23	56.9	65.0	5.789960
334	0.29	62.4	58.0	5.811141
335	0.31	63.3	58.0	5.814131
336	0.24	62.8	57.0	5.817111
336	0.24	62.3	57.0	5.817111
337	0.26	61.9	55.0	5.820083
337	0.22	65.1	61.0	5.820083
338	0.23	59.4	61.0	5.823046

1-10 of 10,000 rows Previous **1** 2 3 4 5 6 … 1000 Next

summarise()

The summarise() function allows us to calculate sumarry statistics on our data. At its most basic level, it looks like this:

```
diamonds %>%
    summarise(mean(price))
```

mean(price) <dbl>
3932.8

1 row

```
diamonds %>%
    summarise(Mean = mean(price), Median = median(price))
```

Mean <dbl>	Median <dbl>
3932.8	2401

1 row

```
diamonds %>%
    summarise(mean(carat), mean(price), mean(depth))
```

mean(carat) <dbl>	mean(price) <dbl>	mean(depth) <dbl>
0.7979397	3932.8	61.7494

1 row

The real power of summarise() comes when we combine it with the last dplyr verb, group_by(). group_by() allows us to break the data into groups and then summarise or perform other tasks. Here are a few examples:

```
diamonds %>%
    group_by(cut) %>%
    summarise(mean(price))
```

cut <ord>	mean(price) <dbl>
Fair	4358.758
Good	3928.864
Very Good	3981.760
Premium	4584.258
Ideal	3457.542

5 rows

```
diamonds %>%
    group_by(cut) %>%
    summarise(mean(carat))
```

cut <ord>	mean(carat) <dbl>
Fair	1.0461366
Good	0.8491847
Very Good	0.8063814
Premium	0.8919549
Ideal	0.7028370

5 rows

```
diamonds %>%
    group_by(cut) %>%
    summarise(mean(carat), n())
```

cut <ord>	mean(carat) <dbl>	n() <int>
Fair	1.0461366	1610
Good	0.8491847	4906
Very Good	0.8063814	12082
Premium	0.8919549	13791
Ideal	0.7028370	21551

5 rows

```
diamonds %>%
    group_by(cut) %>%
    count(cut)
```

cut <ord>	n <int>
Fair	1610
Good	4906
Very Good	12082
Premium	13791
Ideal	21551

5 rows

You've now seen how each of the verbs work by themselves and in conjunction with others. Remember that we can go on to link them together anyway that makes sense for our analysis. We can calculate some statistics, then arrange them highest to lowest. Or we can filter everything but Ideal cut diamonds and go from there. Or we can create a new variable and use its value for our analysis. The options are endless, and more often limited by our own creativity than anything else. As we move forward through the rest of the EDA phase, we will get lots of experience chaining dplyr verbs together and frequently building plots with the data. Before long, you will have plenty of practice using these verbs and will be very comfortable doing so.

Diving into EDA

We now have the necessary tools under our belt, ggplot2 and dplyr. As we move forward, these will be the primary tools we use to explore our data. Let's take a quick look back at the goals of this phase.

Primary Goals of Exploratory Data Analysis (EDA)

- Gain an understanding of, familiarity with, and insight into our dataset
- Find (and deal with) any problems with our data
- Generate ideas to use and test in the modeling phase
- (Further) refine our question, where appropriate

In order to guide us along the path towards accomplishing those goals, we will use the pared down version of the framework laid out by Stephen Few. Here it is again:

Steps of EDA, or Types of Graphs to Explore

> 1. Variation within categories
> 2. Variation within measures
> 3. Variation through time
> 4. Relationships among measures
> 5. Relationships among categories (Few, 2015)

As we go through each of the above steps, we will use graphs and other relevant statistics to accomplish the first goal of EDA from above (Gain an understanding of, familiarity with, and insight into our dataset). Along the way we will likely find some problems with our dataset that need to be addressed, and we will take care of those (the second goal from above). Any insight we gain from the process will give us a feel for the potential candidates to be included and tested as a part of the model we will attempt to create in the next phase, and we will continue to refine our question as we go, if its appropriate to do so. Let's dive in.

Note that my most common workflow is to build a graph for a quick impression, then follow that by confirming the actual numbers with a table of some sort. I don't always feel the need to make the table or confirm the numbers, so sometimes I skip the step. Many times I assume that I will want that confirmation and go ahead and create the graph and the table at the same time. All of these are acceptable options, and I encourage you to do whatever feels comfortable.

Variation within categories

Variation within categories refers to separating the data into categories, and then analysing it by some measure. Some of the most common types of analysis we will be doing at this point is ranking each category and considering their contribution to the whole.

For example, we might break the data down by country, and then rank the countries according to which had the most projects, or by which country had the most successful projects.

The first step for me in this process is to come up with a couple of lists of variables to guide me through this stage of my analysis. Since I am breaking the data into categories, the first list will involve variables that could the categories to explore. The second list will be variables that might contain interesting information about those categories.

To refresh our memory on our Kickstarter data and the variables we have to work with, we can run some functions to inspect the data again like we did in Phase 1. I'll just do one here to start with, but feel free to reacquaint yourself with the dataset however you see fit.

```
ks %>% summary
```

```
##        id                photo                name
##  Min.   :3.703e+06   Length:500         Length:500
##  1st Qu.:5.918e+08   Class :character   Class :character
##  Median :1.162e+09   Mode  :character   Mode  :character
##  Mean   :1.124e+09
##  3rd Qu.:1.649e+09
##  Max.   :2.147e+09
##     blurb                goal           pledged
##  Length:500         Min.   :    10   Min.   :     0.00
##  Class :character   1st Qu.:  2000   1st Qu.:    54.75
##  Mode  :character   Median :  5000   Median :  1076.00
##                     Mean   : 12673   Mean   :  7434.08
##                     3rd Qu.: 12000   3rd Qu.:  5503.58
##                     Max.   :125000   Max.   :172586.00
##     state              slug             country
##  Length:500         Length:500         Length:500
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##    currency         currency_trailing_code    deadline
##  Length:500         Mode :logical          Min.   :2010-03-07 05:00:00
##  Class :character   FALSE:60               1st Qu.:2013-09-16 12:26:19
##  Mode  :character   TRUE :440              Median :2015-03-28 19:49:20
##                     NA's :0                Mean   :2014-12-07 00:39:29
##                                            3rd Qu.:2016-04-16 15:34:53
##                                            Max.   :2017-09-19 22:21:48
##  state_changed_at               created_at
##  Min.   :2010-03-07 05:00:10   Min.   :2010-01-13 17:25:44
##  1st Qu.:2013-09-16 12:26:56   1st Qu.:2013-06-26 11:01:40
##  Median :2015-03-28 19:49:21   Median :2015-01-05 03:29:48
##  Mean   :2014-12-04 11:11:37   Mean   :2014-09-21 12:36:46
##  3rd Qu.:2016-04-15 06:05:55   3rd Qu.:2016-02-07 05:27:22
##  Max.   :2017-08-15 18:01:03   Max.   :2017-08-10 17:16:07
##   launched_at                  staff_pick      is_starrable
##  Min.   :2010-01-13 22:35:38   Mode :logical   Mode :logical
##  1st Qu.:2013-08-11 02:22:38   FALSE:440       FALSE:488
##  Median :2015-02-23 10:40:26   TRUE :60        TRUE :12
##  Mean   :2014-11-03 00:13:57   NA's :0         NA's :0
##  3rd Qu.:2016-03-15 20:33:17
##  Max.   :2017-08-15 18:01:02
##  backers_count     static_usd_rate    usd_pledged       creator
##  Min.   :   0.00   Min.   :0.05474   Min.   :     0   Length:500
##  1st Qu.:   2.00   1st Qu.:1.00000   1st Qu.:    60   Class :character
##  Median :  21.00   Median :1.00000   Median :  1100   Mode  :character
##  Mean   :  87.84   Mean   :1.00932   Mean   :  6632
##  3rd Qu.:  72.25   3rd Qu.:1.00000   3rd Qu.:  5145
##  Max.   :3399.00   Max.   :1.69769   Max.   :152604
##    location           category           profile          spotlight
##  Length:500         Length:500         Length:500         Mode :logical
##  Class :character   Class :character   Class :character   FALSE:264
##  Mode  :character   Mode  :character   Mode  :character   TRUE :236
##                                                           NA's :0
##
##
```

```
##      urls            source_url
##  Length:500         Length:500
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

Notice the use of the pipe?

So when we are talking about variation within categories, we are talking about breaking the data up into different categories, and seeing how the values of other variables differ for each of the new groups. Looking through the summary of each variable above, we can pick out a few that might be useful for breaking our data into categories. “state” looks like it could hold information about the success of each project and would make a great group of categories. It also looks like the following variables could be useful: country, currency, location, category, spotlight, and is_starrable. What each of these variables have in common is that they are (or can be) factor variables. Factor variables are categorical data. The range of possible values is fixed; those values are known as “levels”. They can be ordered or not. Let’s take a look at the levels of a few of the factor variables mentioned above.

```
levels(ks$state)
```

```
## NULL
```

The reason we got null here is that R has the state variable is encoded as a character variable. You can scroll up to where we ran the summary function to see that (where it says `Class: character`). We simply need to tell R to interpret it as a factor. We’ll use the `factor()` function from the forcats package.

```
library(forcats)
# I want this to take place in the codebook

levels(factor(ks$state))
```

```
## [1] "canceled"   "failed"     "live"       "successful" "suspended"
```

The code above simply shows us what those levels would be if “state” were a factor variable. To make the change permanent, we simply need to reassign the state variable:

```
ks$state <- factor(ks$state)
```

We can do the same for several other variables that should be treated as factors. Let’s see what we find out from a few of the other variables. Checking the levels should tell us whether or not they may be good candidates to be treated as factors.

```
levels(factor(ks$country))
```

```
##  [1] "AU" "BE" "CA" "DE" "DK" "ES" "GB" "HK" "IT" "MX" "NO" "NZ" "SE" "US"
```

```
levels(factor(ks$category))
```

```
##   [1] "3D Printing"       "Academic"          "Accessories"
##   [4] "Action"            "Animation"         "Anthologies"
##   [7] "Apparel"           "Apps"              "Architecture"
##  [10] "Art Books"         "Audio"             "Blues"
##  [13] "Camera Equipment"  "Candles"           "Ceramics"
##  [16] "Children's Books"  "Childrenswear"     "Classical Music"
##  [19] "Comedy"            "Comic Books"       "Conceptual Art"
##  [22] "Cookbooks"         "Country & Folk"    "Couture"
##  [25] "Digital Art"       "DIY"               "DIY Electronics"
##  [28] "Documentary"       "Drama"             "Drinks"
##  [31] "Electronic Music"  "Events"            "Experimental"
##  [34] "Fabrication Tools" "Faith"             "Family"
##  [37] "Fantasy"           "Farmer's Markets"  "Farms"
##  [40] "Festivals"         "Fiction"           "Fine Art"
##  [43] "Food Trucks"       "Footwear"          "Gadgets"
##  [46] "Gaming Hardware"   "Glass"             "Graphic Design"
##  [49] "Graphic Novels"    "Hardware"          "Hip-Hop"
##  [52] "Horror"            "Illustration"      "Indie Rock"
##  [55] "Jazz"              "Jewelry"           "Literary Journals"
##  [58] "Live Games"        "Makerspaces"       "Metal"
##  [61] "Mixed Media"       "Mobile Games"      "Music Videos"
##  [64] "Musical"           "Narrative Film"    "Nature"
##  [67] "Nonfiction"        "Painting"          "People"
##  [70] "Performance Art"   "Performances"      "Periodicals"
##  [73] "Photo"             "Photobooks"        "Places"
##  [76] "Playing Cards"     "Plays"             "Poetry"
##  [79] "Pop"               "Print"             "Printing"
##  [82] "Product Design"    "Public Art"        "Puzzles"
##  [85] "Quilts"            "R&B"               "Radio & Podcasts"
##  [88] "Ready-to-wear"     "Restaurants"       "Robots"
##  [91] "Rock"              "Romance"           "Science Fiction"
##  [94] "Sculpture"         "Shorts"            "Small Batch"
##  [97] "Software"          "Spaces"            "Tabletop Games"
## [100] "Television"        "Textiles"          "Thrillers"
## [103] "Translations"      "Video"             "Video Art"
## [106] "Video Games"       "Wearables"         "Web"
## [109] "Webcomics"         "Webseries"         "Woodworking"
## [112] "World Music"       "Young Adult"       "Zines"
```

```
levels(factor(ks$is_starrable))
```

```
## [1] "FALSE" "TRUE"
```

```
levels(factor(ks$spotlight))
```

```
## [1] "FALSE" "TRUE"
```

```
levels(factor(ks$currency))
```

```
##  [1] "AUD" "CAD" "DKK" "EUR" "GBP" "HKD" "MXN" "NOK" "NZD" "SEK" "USD"
```

```
levels(factor(ks$location))
```

```
##   [1] "Abingdon, UK"                  "Acton, MA"
##   [3] "Ada, OK"                       "Albuquerque, NM"
##   [5] "Alexandria, VA"                "Amarillo, TX"
##   [7] "Amsterdam, Netherlands"        "Anaheim, CA"
##   [9] "Antioch, CA"                   "Apoteri, Guyana"
##  [11] "Arhus, Denmark"                "Arlington, VA"
##  [13] "Asbury Park, NJ"               "Atascadero, CA"
##  [15] "Athens, OH"                    "Atlanta, GA"
##  [17] "Auckland, NZ"                  "Aurora, IL"
##  [19] "Austin, TX"                    "Bakersfield, CA"
##  [21] "Baldwin, GA"                   "Baltimore, MD"
##  [23] "Banner Elk, NC"                "Barcelona, Spain"
##  [25] "Bayside, Queens, NY"           "Beaufort, NC"
##  [27] "Bellingham, WA"                "Bergen, Norway"
##  [29] "Berlin, Germany"               "Birmingham, UK"
##  [31] "Blacksburg, VA"                "Bloenduos, Iceland"
##  [33] "Bloomington, IL"               "Bloomington, IN"
##  [35] "Bonn, Germany"                 "Boston, MA"
##  [37] "Bowling Green, KY"             "Brisbane, AU"
##  [39] "Bristol, UK"                   "Bronx, NY"
##  [41] "Brooklyn, NY"                  "Brownsville, TX"
##  [43] "Buenos Aires, Argentina"       "Burbank, CA"
##  [45] "Cadillac, MI"                  "Cambridge, MA"
##  [47] "Canberra, AU"                  "Canterbury, UK"
##  [49] "Catonsville, MD"               "Cazenovia, NY"
##  [51] "Charleston, SC"                "Charlotte, NC"
##  [53] "Chicago, IL"                   "Cincinnati, OH"
##  [55] "Clayton, OH"                   "Cleveland, OH"
##  [57] "Cologne, Germany"              "Colorado Springs, CO"
##  [59] "Columbus, GA"                  "Columbus, OH"
##  [61] "Copenhagen, Denmark"           "Costa Mesa, CA"
##  [63] "Dallas, TX"                    "Davis, CA"
##  [65] "De Kalb, IL"                   "Decatur, GA"
##  [67] "Denver, CO"                    "Doncaster, UK"
##  [69] "Dover, NH"                     "Downtown Toronto, Canada"
##  [71] "Dumfries, VA"                  "Durango, CO"
##  [73] "Durham, NC"                    "East Lansing, MI"
##  [75] "Elk Grove, CA"                 "Elmwood Park, IL"
##  [77] "Erfurt, Germany"               "Espa\xed\xb1a, Spain"
##  [79] "Evanston, IL"                  "Federal Way, WA"
##  [81] "Fenton, MI"                    "Flagstaff, AZ"
##  [83] "Fort Lauderdale, FL"           "Fort Worth, TX"
##  [85] "Foyil, OK"                     "Frankfurt, Germany"
##  [87] "Fredericksburg, VA"            "Freising, Germany"
##  [89] "Funkstown, MD"                 "Georgetown, TX"
##  [91] "Gerlach, NV"                   "Gig Harbor, WA"
##  [93] "Gladstone, OR"                 "Glencoe, KY"
##  [95] "Glenolden, PA"                 "Gothenburg, Sweden"
##  [97] "Grafton, VT"                   "Granada, Spain"
##  [99] "Greater London, UK"            "Greater Manchester, UK"
## [101] "Guildford, UK"                 "Gulfport, MS"
## [103] "Halifax, Canada"               "Hamburg, Germany"
## [105] "Harrisburg, PA"                "Haugesund, Norway"
## [107] "Heber City, UT"                "Helena, MT"
## [109] "Hereford, UK"                  "Hermosa Beach, CA"
## [111] "High Laver, UK"                "Hilton Head Island, SC"
```

```
## [113] "Hoi An, Viet Nam"                 "Hong Kong, Hong Kong"
## [115] "Houston, TX"                      "Hove, UK"
## [117] "Hsinchu City, Taiwan"             "Huntingdon, PA"
## [119] "Huntington Beach, CA"             "Idaho Falls, ID"
## [121] "Isleworth, UK"                    "Jacksonville, FL"
## [123] "Kalamazoo, MI"                    "Kansas City, MO"
## [125] "Kelowna, Canada"                  "Kennewick, WA"
## [127] "Ketchum, ID"                      "Key Biscayne, FL"
## [129] "Kiev, Ukraine"                    "King of Prussia, PA"
## [131] "Knoxville, TN"                    "Kortrijk, Belgium"
## [133] "Kosice, Slovakia"                 "La Salle, IL"
## [135] "Lakeville, MN"                    "Lakewood, CO"
## [137] "Lancashire, UK"                   "Lansing, MI"
## [139] "Las Vegas, NV"                    "Lawton, OK"
## [141] "Lehighton, PA"                    "Leicester, UK"
## [143] "Leicestershire, UK"               "Lexington, KY"
## [145] "Little Rock, AR"                  "Lombard, IL"
## [147] "London, Canada"                   "London, UK"
## [149] "Los Angeles, CA"                  "Louisville, KY"
## [151] "Lynchburg, VA"                    "Madrid, Spain"
## [153] "Malaga, Spain"                    "Malm̦, Sweden"
## [155] "Manassas Park, VA"                "Manchester, UK"
## [157] "Mandan, ND"                       "Manhattan, NY"
## [159] "Manila, Philippines"              "Martinsburg, WV"
## [161] "Melbourne, AU"                    "Memphis, TN"
## [163] "Mesa, AZ"                         "Mestre, Italy"
## [165] "Mexico, Mexico"                   "Miami, FL"
## [167] "Middleboro, MA"                   "Minneapolis, MN"
## [169] "Monroe, NC"                       "Monterey, CA"
## [171] "Monterrey, Mexico"                "Montreal, Canada"
## [173] "Mundelein, IL"                    "Nanaimo, Canada"
## [175] "Naples, FL"                       "Nashville, TN"
## [177] "New Orleans, LA"                  "New York, NY"
## [179] "Newport, UK"                      "Norcross, GA"
## [181] "North Hollywood, Los Angeles, CA" "North Ipswich, AU"
## [183] "North Yorkshire, UK"              "Nyack, NY"
## [185] "Oakland, CA"                      "Oklahoma City, OK"
## [187] "Old Town Stony Plain, Canada"     "Omaha, NE"
## [189] "Ontario, CA"                      "Orimattila, Finland"
## [191] "Orlando, FL"                      "Oshkosh, WI"
## [193] "Palma de Mallorca, Spain"         "Palo Alto, CA"
## [195] "Pasadena, CA"                     "Peekskill, NY"
## [197] "Pensacola, FL"                    "Philadelphia, PA"
## [199] "Phoenix, AZ"                      "Pittsburgh, PA"
## [201] "Placerville, CA"                  "Plantation, FL"
## [203] "Plymouth, MI"                     "Portland, ME"
## [205] "Portland, OR"                     "Portsmouth, UK"
## [207] "Provo, UT"                        "Queens, NY"
## [209] "Redmond, WA"                      "Richmond, KY"
## [211] "Richmond, VA"                     "Riverside, CA"
## [213] "Roscoe, IL"                       "Royal Oak, MI"
## [215] "Sacramento, CA"                   "Saddle River, NJ"
## [217] "Sag Harbor, NY"                   "Salem, NH"
## [219] "Salt Lake City, UT"               "San Antonio, TX"
## [221] "San Diego, CA"                    "San Francisco, CA"
## [223] "San Marcos, TX"                   "Sandwich, MA"
## [225] "Sandy, UT"                        "Santa Ana, CA"
```

```
## [227] "Santa Clara, CA"                "Santa Cruz, CA"
## [229] "Santa Fe, NM"                   "Santa Monica, CA"
## [231] "Scarborough, AU"                "Schaumburg, IL"
## [233] "Scranton, PA"                   "Scunthorpe, UK"
## [235] "Seattle, WA"                    "Selma, CA"
## [237] "Seoul, South Korea"             "Shanghai, China"
## [239] "Shawnee, KS"                    "Snowflake, AZ"
## [241] "Somerset, KY"                   "South Houston, TX"
## [243] "Spirit Lake, IA"                "Spokane, WA"
## [245] "Spring Hill, TN"                "Springfield, MO"
## [247] "St. Augustine, FL"              "St. Louis, MO"
## [249] "St. Paul, MN"                   "St.-Bruno-de-Montarville, Canada"
## [251] "Staten Island, NY"              "Stockholm, Sweden"
## [253] "Sturgis, SD"                    "Summerside, Canada"
## [255] "Sussex, NJ"                     "Syracuse, NY"
## [257] "Tampa, FL"                      "Timmins, Canada"
## [259] "Titusville, FL"                 "Toronto, Canada"
## [261] "Trento, Italy"                  "Trondheim, Norway"
## [263] "Tucson, AZ"                     "Twin Falls, ID"
## [265] "Tyler, TX"                      "Ukiah, CA"
## [267] "Upland, IN"                     "Vestnes, Norway"
## [269] "Vigo, Spain"                    "Vilnius, Lithuania"
## [271] "Waco, TX"                       "Warner Robins, GA"
## [273] "Washington, DC"                 "Wayland, NY"
## [275] "Wesley Chapel, FL"              "West Monroe, LA"
## [277] "Wheaton, IL"                    "Whistler, Canada"
## [279] "White River Junction, VT"       "Wichita, KS"
## [281] "Wiesbaden, Germany"             "Willimantic, CT"
## [283] "Wilmington, DE"                 "Wilmington, NC"
## [285] "Windsor Locks, CT"              "Winona, MN"
## [287] "Woodbury, MN"                   "Zacatecas, Mexico"
```

```
levels(factor(ks$creator))
```

```
##   [1] "Aaron and Jan Geibel"
##   [2] "Abigail Scollay"
##   [3] "abode"
##   [4] "Acad Version"
##   [5] "Adam Geiger"
##   [6] "Adam Leech"
##   [7] "Adam Marie"
##   [8] "Adam Metropolis"
##   [9] "Adisa Zvekic"
##  [10] "Adrian Allen"
##  [11] "Aevi Watches"
##  [12] "Ahmad Merheb"
##  [13] "Airpaq"
##  [14] "Airship Isabella"
##  [15] "AJ Sikes"
##  [16] "Alan Wood"
##  [17] "Alan Yeung"
##  [18] "Alexandra"
##  [19] "Alexandra Blue"
##  [20] "Alexandra Ritchie"
##  [21] "AlexHubbell"
##  [22] "Ali"
##  [23] "Alika Davis"
##  [24] "Alisa McCance"
##  [25] "Allen Roe"
##  [26] "Allison"
##  [27] "AmbeRed"
##  [28] "amirhossein momen"
##  [29] "Amita Nathwani"
##  [30] "Andre Johnson"
##  [31] "Andrew"
##  [32] "Andrew Blossom"
##  [33] "Andrew DeChristopher"
##  [34] "Andy Levy"
##  [35] "Angry Inch Brewing"
##  [36] "Ann Marie Coviello"
##  [37] "Anthony Djuren"
##  [38] "Anthony Piper"
##  [39] "Antonio Casasanta"
##  [40] "Apartment 5E Theater Company"
##  [41] "Ara Gureghian"
##  [42] "Ari Rice (deleted)"
##  [43] "Ashley Allen"
##  [44] "Ashley Carr"
##  [45] "Ballistic Studios"
##  [46] "Ben B"
##  [47] "Benjamin I Bryan"
##  [48] "Bernd Ott & Emily Besa"
##  [49] "Bill Elgin (deleted)"
##  [50] "Billy W. Mitchell"
##  [51] "Biscotte Yarns"
##  [52] "Blake Louis Hocker"
##  [53] "Bob Humphrey"
##  [54] "Bobby Choy"
##  [55] "Brad Christmann"
##  [56] "Brandy Lawhorn"
```

```
##  [57] "Brian Garber"
##  [58] "Brian Hawkins"
##  [59] "Brian K. Palmer"
##  [60] "Brock DeBoer"
##  [61] "Brooke Smith (deleted)"
##  [62] "Bucket Siler"
##  [63] "Caleb Gave Mathis"
##  [64] "Caleb Stephens"
##  [65] "Canadian Institute for Czech Music"
##  [66] "Carl Rossi"
##  [67] "Carlin Adelson"
##  [68] "Carly Plasha"
##  [69] "Casey Hayes"
##  [70] "Cassandra Turner"
##  [71] "Cassie McDaniel"
##  [72] "Catherine Weiss-Celley"
##  [73] "Charles Johnson Jr."
##  [74] "Chelsea Hrynick Browne"
##  [75] "Chris Andersen"
##  [76] "Chris Calzia"
##  [77] "Chris Coyne"
##  [78] "Chris Matthewman"
##  [79] "Christian Bartram"
##  [80] "Christian Rosier"
##  [81] "Christopher Campbell"
##  [82] "Christopher Ciesiel"
##  [83] "Christopher Head (deleted)"
##  [84] "Christopher Herrera"
##  [85] "christopher nicholas"
##  [86] "Cineridge Entertainment, LLC."
##  [87] "CJP"
##  [88] "Clarence Oates"
##  [89] "Classy Cake Creations"
##  [90] "Claudia Stocker"
##  [91] "Codie Cosgrove"
##  [92] "Colin Blakely"
##  [93] "Colin Momeyer"
##  [94] "Communist Daughter"
##  [95] "Corey Landen"
##  [96] "corey underwood"
##  [97] "Cornelius Sullivan"
##  [98] "Cyrus Farivar"
##  [99] "Dan Phelps CD release"
## [100] "Daniel Eggington (deleted)"
## [101] "Daniel Jensen"
## [102] "Daniel Sanchez"
## [103] "Daniel Tidwell"
## [104] "Dante"
## [105] "Danza-RevistaMX"
## [106] "Darts Connect"
## [107] "DAVE WEISBERG"
## [108] "David Bui"
## [109] "David Cornelson"
## [110] "David Guinn"
## [111] "David J. Morris"
## [112] "David Toledo"
## [113] "David Wanczyk"
```

```
## [114] "David White"
## [115] "David Zawacki"
## [116] "Dawn Deason (deleted)"
## [117] "Deborah Walther"
## [118] "Denis and Terri Zafiros"
## [119] "Desiree Turner"
## [120] "Devyn DeLoera"
## [121] "Dia Proimos"
## [122] "Doctor Octoroc"
## [123] "Dorothy Gambrell"
## [124] "Down In Light"
## [125] "Drawing From Heaven"
## [126] "Dreaming City Books (Jim Kirkland Pub.)"
## [127] "Dueling Wizards, LLC"
## [128] "Dustin"
## [129] "Dustin White"
## [130] "Dylan Guffey"
## [131] "easyshower"
## [132] "Ed Galloway Totem Pole Park"
## [133] "Ed Goldberg"
## [134] "Edwin Premberg (deleted)"
## [135] "Eileen"
## [136] "Elizabeth Raybee"
## [137] "Elly Blue"
## [138] "EQUIPT for PLAY"
## [139] "Eric Anderson"
## [140] "Eric Holstein"
## [141] "Eric Jeong"
## [142] "Erik Carl"
## [143] "Erik Kim Malmberg"
## [144] "Evading Azrael"
## [145] "Evanston Escola de Samba"
## [146] "Evelyn Aira"
## [147] "Evelyne Dubois"
## [148] "Evil Girlfriend Media"
## [149] "EXPLOSHIELD Limited"
## [150] "Filippo Sterrantino"
## [151] "Fledge"
## [152] "Flloyd"
## [153] "FOG DOG"
## [154] "Folding Firebox"
## [155] "France Garrido"
## [156] "Freak Show"
## [157] "FrigidFox"
## [158] "Full of Win Games"
## [159] "Gabriel Lubell"
## [160] "Galen Ihlenfeldt"
## [161] "Gary Dressler"
## [162] "Gast\xcc_n Arballo"
## [163] "Gee"
## [164] "Genealogical Society of Pennsylvania"
## [165] "Gizbee LLC"
## [166] "Gozer Games"
## [167] "Greg Adkins"
## [168] "Greg Stolze"
## [169] "Hamish John Appleby"
## [170] "Hannah Harvigsson"
```

```
## [171] "Happy Hour Hero3+ Productions (deleted)"
## [172] "Harrison Mead"
## [173] "Harry Herzberg"
## [174] "Heather Craig"
## [175] "Herbie J Pilato"
## [176] "Holly Hunt"
## [177] "Hotep TheArtist"
## [178] "Ian Pudney"
## [179] "Ian Reagan"
## [180] "In-Label Records"
## [181] "IndieCarry"
## [182] "Intermezzo"
## [183] "Ireca Sims"
## [184] "Iryna Kucheryava, James Warwick"
## [185] "Isabel Draves"
## [186] "isaiah lucero"
## [187] "Jack C. Newell"
## [188] "Jacob"
## [189] "Jacob Friedman"
## [190] "Jacob Porter"
## [191] "Jaime Armas"
## [192] "Jaime Wright"
## [193] "Jake Green"
## [194] "James A. Owen"
## [195] "James and Anna (deleted)"
## [196] "James Black (deleted)"
## [197] "James K. Holder II"
## [198] "James Kelley"
## [199] "James Smith"
## [200] "James Tradgett"
## [201] "jami lyn"
## [202] "Jamie"
## [203] "Jamie Bianchini"
## [204] "Jamie Martin"
## [205] "Jamie Plante"
## [206] "Jason Boone"
## [207] "Jason Peach"
## [208] "Jay B"
## [209] "jayblack"
## [210] "Jen Reeves"
## [211] "Jennica Schwartzman"
## [212] "Jennifer Silvey"
## [213] "Jenny Jarnagin"
## [214] "jerome"
## [215] "Jesse Banda"
## [216] "Jesse Manfra"
## [217] "Jesse Robison"
## [218] "Jim Ettwein"
## [219] "Joe Trojnor-Barron"
## [220] "John Alexander Miller"
## [221] "John Berendzen"
## [222] "John C. Henneberg"
## [223] "John Cullen"
## [224] "John Elefante"
## [225] "John Rap"
## [226] "John Santagada"
## [227] "Jon Antcliff"
```

```
## [228] "Jonathan"
## [229] "Jonathon High"
## [230] "Jonne Ziengs"
## [231] "Jordan Clark"
## [232] "Josh Bramos"
## [233] "Josh Gray"
## [234] "Joshua Adams"
## [235] "Joshua Emdon"
## [236] "Joshua R. Pinkas"
## [237] "Jozef Karpiel (deleted)"
## [238] "Juli Chavez"
## [239] "Julia M. Doughty / Doug Wood"
## [240] "Julie Renee McCarty"
## [241] "Justice Pirkey"
## [242] "Justin Terveen"
## [243] "J\xcc_rgen Scholz"
## [244] "Kairu Photography"
## [245] "Kait Rhoads"
## [246] "Kara McMaster"
## [247] "Karen Hansen"
## [248] "Karin Pihl"
## [249] "Karina Rocha"
## [250] "Karl Raschke"
## [251] "Kate Bell"
## [252] "Kate Wengier (and kids)"
## [253] "katemilford"
## [254] "Kathryn M Highfield"
## [255] "Kathy Fox - Fox Foods"
## [256] "Keith Newton & Steve Gorman"
## [257] "Kelly Matthews"
## [258] "Kelly Schatz"
## [259] "Ken Avery"
## [260] "Ken Bishop"
## [261] "Kenneth  Green"
## [262] "Kenneth Helm"
## [263] "kermit eby lll"
## [264] "Kevin Fishburne"
## [265] "KEVIN HANLEY"
## [266] "Kevin Krysiak"
## [267] "Kevin Maloney"
## [268] "Kevin Shoemaker & Skylar Bennett"
## [269] "Kevis Antonio"
## [270] "Kharis Featuring Kendre Streeter"
## [271] "King Non"
## [272] "Kip Jalal BRITTON"
## [273] "Kirsten Berg"
## [274] "KitchEco by J\xcc\xfcrgen & Jacob"
## [275] "Kurt Vincent"
## [276] "Landfill Dzine"
## [277] "Landon Purser (deleted)"
## [278] "Laura Larson"
## [279] "Laura Preble"
## [280] "Leah @ RogueJewels"
## [281] "Lee Guerringue"
## [282] "Leonard Patton"
## [283] "Lesley Jones"
## [284] "Lew Lefton"
```

```
## [285] "Lichie"
## [286] "Lisa Maxwell"
## [287] "Logan Crannell"
## [288] "Lori fraize"
## [289] "Louis Williams"
## [290] "Luis Martmen"
## [291] "Lynn Hershman Leeson"
## [292] "Lynne M. Thomas"
## [293] "Mad Traffic"
## [294] "Major Skinner"
## [295] "MAKI - Games"
## [296] "Mamahuhu"
## [297] "Marcus Bittle"
## [298] "Marina"
## [299] "Mario Sosa"
## [300] "Marissa Quinn"
## [301] "Mark Miko and Istvan Vecsernyes"
## [302] "Mark Shirley"
## [303] "Mark Titus"
## [304] "MarQ P"
## [305] "Marshall Moose Moore"
## [306] "Martin Garan\x80\x8dovsk\xcc_"
## [307] "Marty Allen"
## [308] "Mary Gregory"
## [309] "Mary Kulikowski"
## [310] "Mary Trunk"
## [311] "Mat Coleman"
## [312] "Matt"
## [313] "matt leidecker"
## [314] "Matt Santoli"
## [315] "Max Frost"
## [316] "Melica Bloom"
## [317] "Mercedes Parker (deleted)"
## [318] "Mew Mew & Fluffy LTD"
## [319] "Micaton Ergonomics, S.L."
## [320] "Michael Hanna and Jeff Johnson"
## [321] "Michael James Farmer"
## [322] "Michael Muldoon"
## [323] "Michael Newberry"
## [324] "Michael Okincha"
## [325] "Michael Papathanasakis"
## [326] "Michael Patrick Flanigan Jr."
## [327] "Michael Reilly"
## [328] "Michelle \\\"Gearhead\\\" Haunold"
## [329] "Michelle Tran"
## [330] "Mick"
## [331] "Migo"
## [332] "Mike and Julie"
## [333] "Mina Yoo"
## [334] "Minnesota Dance Collaborative"
## [335] "Mr H"
## [336] "Mustapha"
## [337] "N. L. Kerr"
## [338] "NaDA Publishing"
## [339] "Nadia Karim"
## [340] "Naseem Nossiff"
## [341] "Nate"
```

```
## [342] "National Icon"
## [343] "Natures Talk Show LLC Voice Of Nature"
## [344] "Navasota String Band"
## [345] "Neil Meister"
## [346] "NetToons, Inc."
## [347] "Nic Carter"
## [348] "Nick Chiodras"
## [349] "NICOLAS LINARES"
## [350] "Nolan Brundige / NMB Creations"
## [351] "Normal Games Co"
## [352] "NorthsideComedy.com"
## [353] "Oleg Dergachov"
## [354] "Olga Almansky"
## [355] "OPUS High Technology Corp"
## [356] "Orestes Manousos"
## [357] "Oxford American"
## [358] "Parlor Hawk"
## [359] "ParteePartee (deleted)"
## [360] "Patricia Anaya"
## [361] "Patricia Noworol Dance Theater"
## [362] "Patrick C. Simpson-Jones"
## [363] "Patrick Healy"
## [364] "PeaceTones"
## [365] "Pelorus Press"
## [366] "Pensacola Little Theatre"
## [367] "Pete Kolo"
## [368] "Peter Allen"
## [369] "Peter Bond"
## [370] "Peter Sand"
## [371] "Petter Bendiksen"
## [372] "Petunia Tech"
## [373] "Philip Rice"
## [374] "Pitu Sanchez"
## [375] "PLAY AGAIN"
## [376] "PRETTYTHESERIES"
## [377] "Priscilla Aroean"
## [378] "QingYing E&T LLC"
## [379] "Rachelle Robinson"
## [380] "Raistlin"
## [381] "Rame Pizzeria"
## [382] "Randy Rodriguez"
## [383] "Red Scotch Software"
## [384] "Restoration Bid Inc."
## [385] "Rhonda Slone"
## [386] "Richard Tucci"
## [387] "Rishi Sethi"
## [388] "RJ4L"
## [389] "RNDM Design"
## [390] "Robert D. Jansen"
## [391] "Robert James"
## [392] "Robert P. Singleton"
## [393] "Roberto de Farias"
## [394] "Robin Bond"
## [395] "Rocco Panetta"
## [396] "Rodrigo M. Malmsten"
## [397] "Ron Edwards (deleted)"
## [398] "Roschman Dance and Wallis Knot"
```

```
## [399] "Row 1 Productions"
## [400] "Ruben Tello"
## [401] "Rudi"
## [402] "Rush Hicks"
## [403] "Ryan Ovadia"
## [404] "Ryan Pietrzak"
## [405] "Sabrina Cotugno"
## [406] "Sam Hayes"
## [407] "Sandra Golden"
## [408] "Scott Drotar"
## [409] "Scott Lost"
## [410] "Scott Thomson"
## [411] "Sean & Emma"
## [412] "Sean Taylor"
## [413] "Selective Perspective Collective"
## [414] "Sentinel Games"
## [415] "Shaina Tantuico"
## [416] "Shamek V Farrah"
## [417] "Shannon Byrne"
## [418] "Shasta Palmer"
## [419] "Shawn French"
## [420] "Sian Wheatcroft"
## [421] "Simon Arvidsson"
## [422] "Simon Harrison"
## [423] "Simon Horrocks"
## [424] "Simon Von Bargen"
## [425] "Simone's Market Stall"
## [426] "Sithari D"
## [427] "Smieszek"
## [428] "Spencer Miskoviak"
## [429] "Stacy Arnold-Strider"
## [430] "Stephanie"
## [431] "Stephanie Law"
## [432] "Stephen Greenberg"
## [433] "Steven (SVen)"
## [434] "Steven Battey (deleted)"
## [435] "Strange Biology"
## [436] "Styrman & Crew"
## [437] "Sukey Molloy"
## [438] "Supreme Clans"
## [439] "Suzanne Brockmann & small or LARGE"
## [440] "Suzy Liebermann"
## [441] "Sven  Moss"
## [442] "Swayzee"
## [443] "S\xcc\xfcren F. Fantini"
## [444] "Tam Quoc Tran"
## [445] "Tanya"
## [446] "Taro"
## [447] "Taylor"
## [448] "Tea Silvestre Godfrey"
## [449] "Team Kaiju"
## [450] "Team Playout"
## [451] "The Art of Cool Project"
## [452] "The Beekeepers"
## [453] "The Hanser-McClellan Guitar Duo"
## [454] "The Queen Of England Stole my Parents"
## [455] "The SportPod"
```

```
## [456] "Theda Fresques"
## [457] "TheJobJob.com (deleted)"
## [458] "Theo Grimshaw"
## [459] "Theodore Sipes"
## [460] "Thom Turner"
## [461] "thomas mcglone"
## [462] "Thomas Walbert"
## [463] "Thor Platter"
## [464] "Thoren Rogers"
## [465] "Tim Rodriguez"
## [466] "Timothy Blakely"
## [467] "Tony MacGregor"
## [468] "Traces"
## [469] "Travis Greene"
## [470] "Tristan Wiener"
## [471] "Trusty Sidekick Theater Company"
## [472] "Tyler McNamer"
## [473] "Uber and Lyft Driver"
## [474] "Undefined Worship"
## [475] "UNlogical"
## [476] "USA Great Buys, LLC"
## [477] "Vernon Thompson"
## [478] "Veronica Rochelle Raggs"
## [479] "victor franco"
## [480] "Victoria Ann Van Arnam"
## [481] "Victoria Cosplay"
## [482] "Video Daughters"
## [483] "Viktoria Korman"
## [484] "Vincent Amaya"
## [485] "Vision Global"
## [486] "Vyonna Maldonado (deleted)"
## [487] "Wendy Martinez"
## [488] "Werner John"
## [489] "Wes Modes"
## [490] "Whiskey Mother Sucker Productions"
## [491] "Xavier Vargas"
## [492] "Yankee & The Foreigners"
## [493] "Yossra El Said"
## [494] "Zachary Brian Roth"
## [495] "Zhuhai CTC Electronic Co., LTD"
## [496] "Zoe Nicholson"
## [497] "Zona Jennifer"
```

Hey look its my hometown!

```
ks %>% filter(location == "Amarillo, TX")
```

id <int>
2065190192

1 row | 1-1 of 27 columns

Unfortunately, it looks like the Cat Box Pin Shop was cancelled. I guess we'll never get to find out what that even means!

Looking back at the previous chunk, we printed every level for each of the variables listed as data type “character”. In order to be a true categorical value, or category, there should be substantially fewer categories than there are total observations. Looking back through the output at the number of levels for each factor, all but “location” and “creator” have far fewer levels than total observations, which for our dataset is 500.

```
n_distinct(levels(as.factor(ks$location)))
```

```
## [1] 288
```

```
n_distinct(levels(as.factor(ks$creator)))
```

```
## [1] 497
```

Those numbers tell me that a few (very few) of our projects have the same creator, and there are quite a few that share a similar location. We could choose to leave these two as character data (especially creator), but there is some benefit to encoding them as factors despite not being true categories. We can always undo it later if we need to.

```
ks$country <- factor(ks$country)
ks$category <- factor(ks$category)
ks$is_starrable <- factor(ks$is_starrable)
ks$spotlight <- factor(ks$spotlight)
ks$currency <- factor(ks$currency)
ks$location <- factor(ks$location)
ks$creator <- factor(ks$creator)
```

We can verify each of our new factor variables:

```
is.factor(ks$state)
```

```
## [1] TRUE
```

```
is.factor(ks$category)
```

```
## [1] TRUE
```

```
is.factor(ks$is_starrable)
```

```
## [1] TRUE
```

```
is.factor(ks$spotlight)
```

```
## [1] TRUE
```

```
is.factor(ks$currency)
```

```
## [1] TRUE
```

```
is.factor(ks$location)
```

```
## [1] TRUE
```

```
is.factor(ks$creator)
```

```
## [1] TRUE
```

```
# Or

class(ks$state)
```

```
## [1] "factor"
```

```
class(ks$category)
```

```
## [1] "factor"
```

```
class(ks$is_starrable)
```

```
## [1] "factor"
```

```
class(ks$spotlight)
```

```
## [1] "factor"
```

```
class(ks$currency)
```

```
## [1] "factor"
```

```
class(ks$location)
```

```
## [1] "factor"
```

```
class(ks$creator)
```

```
## [1] "factor"
```

We just changed our working dataset again, so we need to update the codebook again. Let's head over there now. Don't forget about "state", which we encoded in a separate, earlier chunk.

To wrap things up, we've essentially accomplished two things in this section. We have come up with a list of variables to be used as our potential categories, and we have converted them to a format that is useful to us: factor variables.

That list includes the following variables: state category is_starrable spotlight currency location creator

Next, we need to come up with that second list of variables I mentioned earlier: the variables that might contain information of interest about the categories.

I'll quickly run another function to refresh our memory on the dataset, specifically the variable names. Earlier I used `summary()`, so this time I will use the much more succinct `colnames()`.

```
ks %>% colnames
```

```
##  [1] "id"                    "photo"
##  [3] "name"                  "blurb"
##  [5] "goal"                  "pledged"
##  [7] "state"                 "slug"
##  [9] "country"               "currency"
## [11] "currency_trailing_code" "deadline"
## [13] "state_changed_at"       "created_at"
## [15] "launched_at"            "staff_pick"
## [17] "is_starrable"           "backers_count"
## [19] "static_usd_rate"        "usd_pledged"
## [21] "creator"                "location"
## [23] "category"               "profile"
## [25] "spotlight"              "urls"
## [27] "source_url"
```

Just glancing through there, here are the variables that jump out to me that could be of interest once we separate our data into categories:

state pledged usd_pledged goal backers_count average pledged spotlight

Yes, state is also one of our categories, but imagine separating the data by country, then ranking the countries by percent of successful projects or something along those lines. Same goes for other categorical values on the list.

Note that there is nothing that keeps us from calculating new variables from other variables. For example, we might want to consider if the length the project is open has any impact on success. We don't have the length the project is open, so we could calculate the interval using "create_at" and "deadline".

Also note that this list (or any other similar list I make at any point) is definitely not exhaustive. We can add to it as we go, especially as we see it fit to calculate new variables. But there just might also be another variable you think might be appropriate to explore. If that's the case, by all means, explore! That's the goal of this phase after all! And if you find something especially interesting that I glossed over, let me know!

In order to be able to reference these type of lists quickly and easily, I usually put them under a heading that is labeled to stick out. I usually begin them with something in all caps such as "LIST:" or "REFERENCE:" just to make them more visible. It's easy to click the drop down menu below, scroll to find them quickly, and then just hit CMD+Down to jump back to the bottom of the document when done. I update these lists as I go with any kind of changes I feel necessary, and sometimes I check them off as I complete them to show my progress and keep myself organized.

Note that this sort of goes against my general workflow rule of working down the page in order to keep things reproducible. Therefore, we just have to be careful to not change anything substantive when we go back up to the lists, and make sure we go down to the very bottom before changing anything again.

Because of this, if the project is especially large or involved I might just create another document called something along the lines of "quick_reference.Rmd" that I can keep open in a tab next to the documents I'm working in. It's easy as can be to switch back and forth between tabs without losing my place.

One last thing about these lists is that they are there to help keep you organized and moving forward in an efficient manner. Nothing says they have to be in your final report or presentation document, so feel free to make notes or changes however helps you. You can always clean it up or remove it later if it makes sense. Just make sure it doesn't effect the flow or reproducibility of the project.

So here are my quick reference lists:

LIST: Variation Within Categories Variables

The Categories: Potential Independent Variables:

- state
- category
- is_starrable
- spotlight
- currency
- location
- creator

The Variation: Potential Dependent Variables:

- state
- pledged
- usd_pledged
- goal
- backers_count
- average pledged
- spotlight

Exploring the “state” Variable

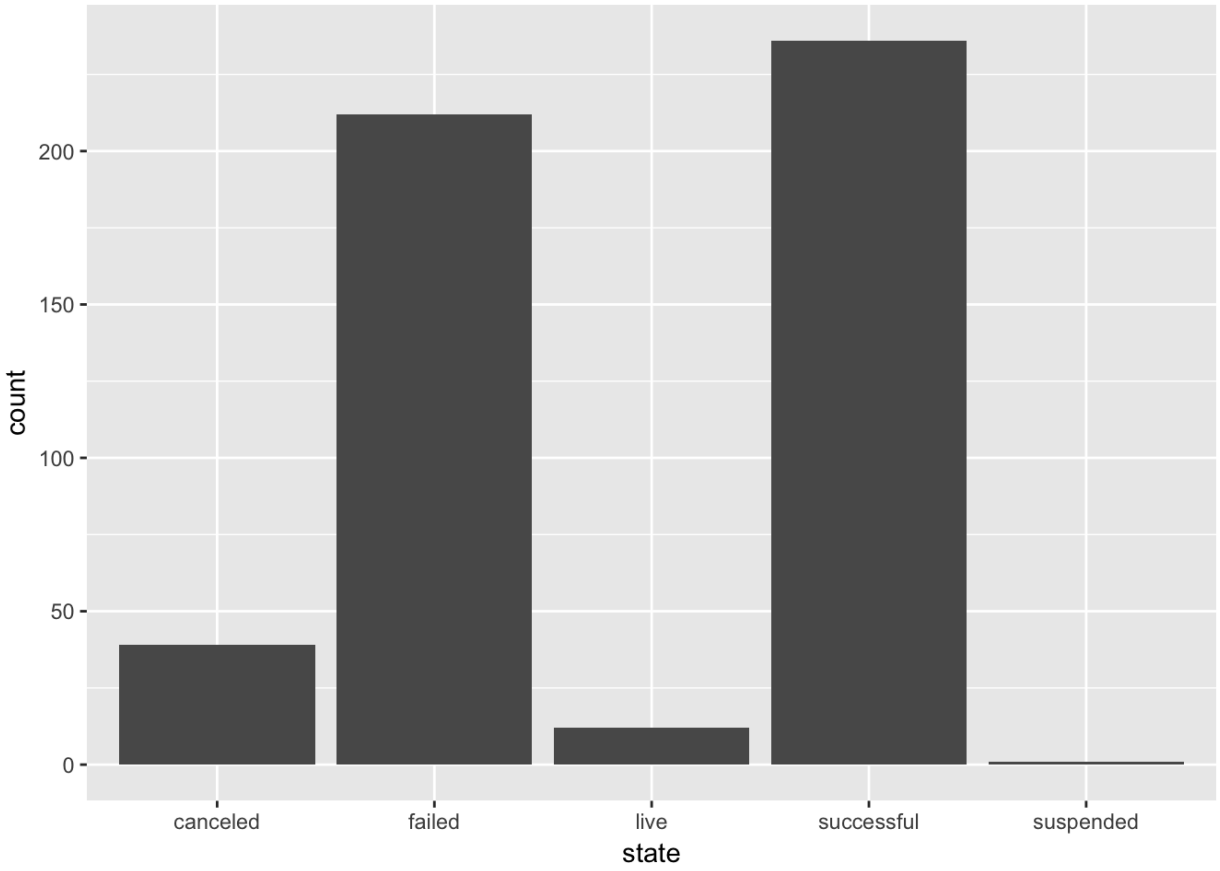
Because the state variable contains information about whether or not each project was successful, it seems like a logical place to start exploring.

```
levels(ks$state)
```

```
## [1] "canceled"   "failed"     "live"       "successful" "suspended"
```

Let’s take a look at how many of our projects fall into each of the categories.

```
ks %>%
    ggplot(aes(x = state)) +
    geom_bar()
```




```
ks %>%
    group_by(state) %>%
    summarise(n())
```

state <fctr>	n() <int>
canceled	39
failed	212
live	12
successful	236
suspended	1

5 rows

```
ks %>%
    group_by(state) %>%
    summarise(n())
```

state <fctr>	n() <int>
canceled	39
failed	212
live	12

state <fctr>	n() <int>
successful	236
suspended	1

5 rows

Looking at the chart it's easy to see that most of our projects fall into either the "failed" or "successful" categories. Thinking through the meaning of these, I think the failed and successful projects are probably of the most interest to us. The projects that are still live likely don't help us much, since they don't yet have an outcome. A little research as to why Kickstarter would suspend a project reveals that there are a number of possible reasons, but most can be summed up as a violation of Kickstarter's rules or terms. Let's pull up the suspended project to see if we can tell what happened here.

```
ks %>%
    filter(state == "suspended")
```

id <int>
896770658

1 row | 1-1 of 27 columns

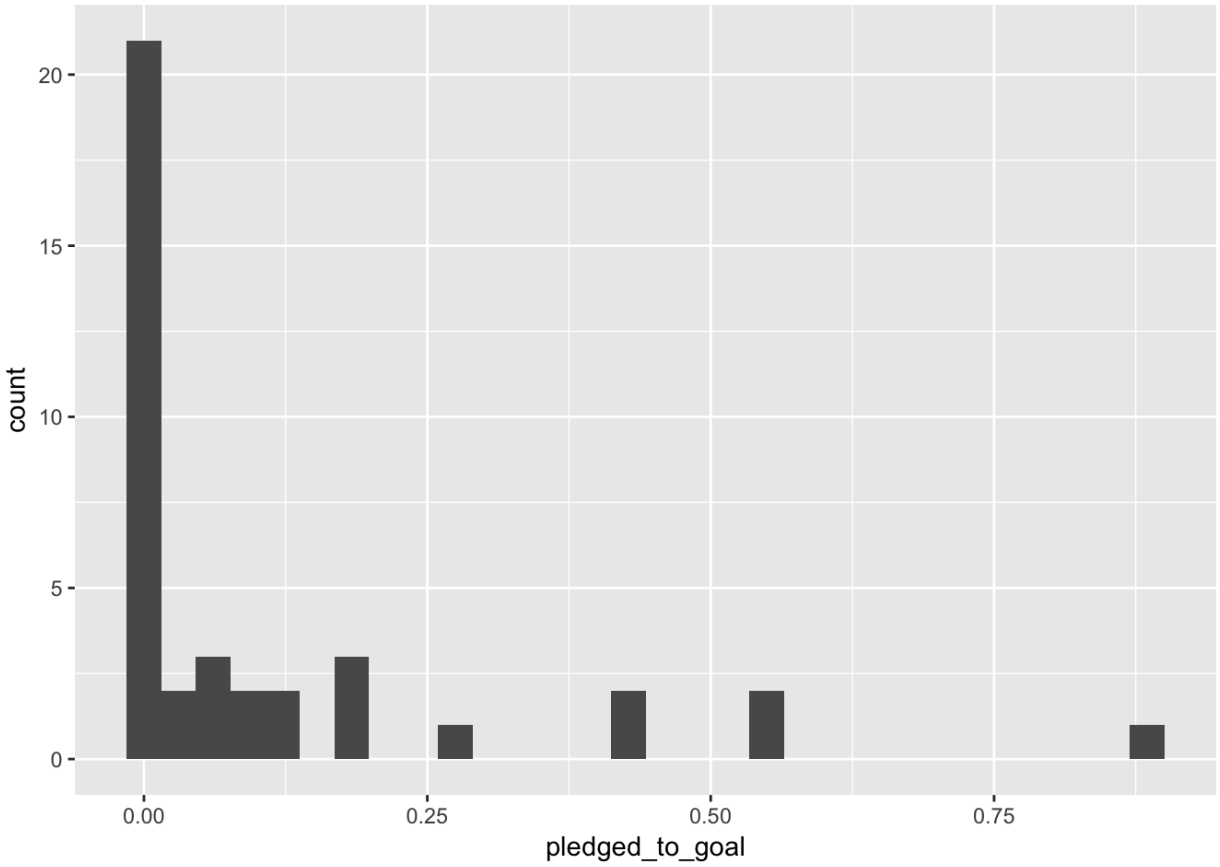
So we have a project called Boltivate. What I find most interesting is that the amount pledged was far above the goal at the time the project was suspended. I found Boltivate's profile on Kickstarter, which is still active at the time I'm putting this together. There is some interesting discussion in the comments section from the day they were suspended and shortly thereafter if you're interested.

My initial thought had been that suspended projects could potentially be lumped in with the failed projects. With Boltivate having received pledges of over four times its goal, calling it failed doesn't seem appropriate. I think we should ignore it for our analysis.

I also originally thought that cancelled projects could be included with failed projects. I'm basing this on the assumption that projects' creators cancel a project when it becomes apparent that it will inevitably fail. Let's test this assumtion and see if it holds any water. One way we can do that is to compare the amount pledged to the goal.

```
ks %>%
    filter(state == "canceled") %>%
    mutate(pledged_to_goal = pledged / goal) %>%
    ggplot(aes(pledged_to_goal)) +
    geom_histogram()
```

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



Looks like most of the cancelled projects had a pledged to goal ratio of at or very near 0, meaning 0 or very few pledges. Some are higher, but none appear to have met their goal (a ratio of greater than 1). Let's look a little closer at the more succesful of those projects.

```
ks %>%
    mutate(pledged_to_goal = pledged / goal) %>%
    filter(state == "canceled", pledged_to_goal >= .5)
```

id <int>
534784401
1196513490
1089870539

3 rows | 1-1 of 28 columns

A few things I notice from looking at the entries above: first, the range of when they were cancelled to their respective deadlines varies from a few hours before to a couple weeks. In fact, the most successful project (88% funded) was cancelled well before its deadline. This seems to invalidate my assumption that projects were cancelled when it became inevitable that they would fail.

My personal takeaway is that we should ignore cancelled projects as well since we don't have much (or possibly any) information about why they were cancelled and don't want to make generalizations that would be inaccurate in at least some instances.

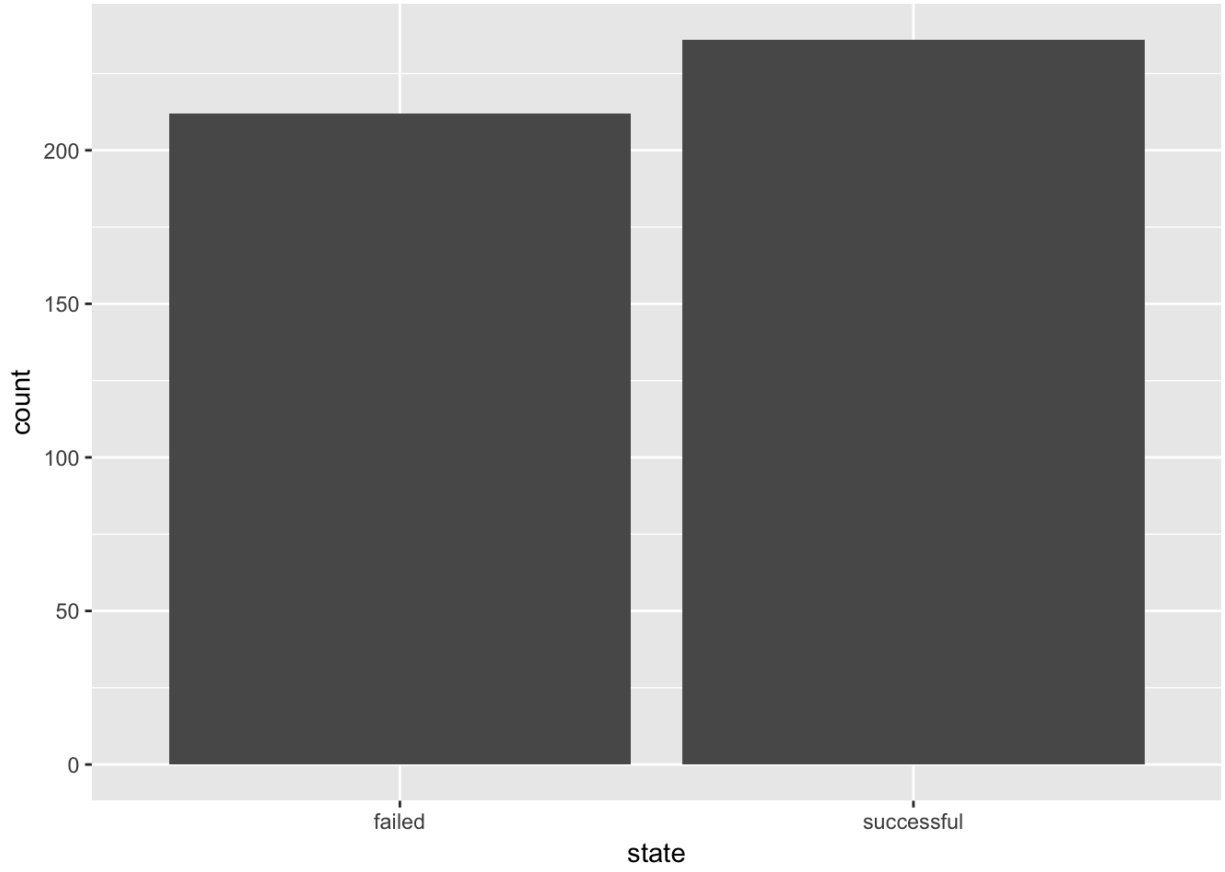
So let's focus in on the successful and failed projects. We'll use the filter function.

```
(ks <- ks %>%
    filter(state == "successful" | state == "failed"))
```

id <int>
832570985
1054135507
1567381435
1329110416
568986755
468138435
305131391
1098019088
700642359
1927640262

1-10 of 448 rows | 1-1 of 27 columns Previous **1** 2 3 4 5 6 … 45 Next

```
ks %>%
    ggplot(aes(x = state)) +
    geom_bar()
```



```
ks %>%
    group_by(state) %>%
    summarise(n())
```

state <fctr>	n() <int>
failed	212
successful	236

2 rows

So we now have a categorical variable with two possible options: “failed” or “successful”.

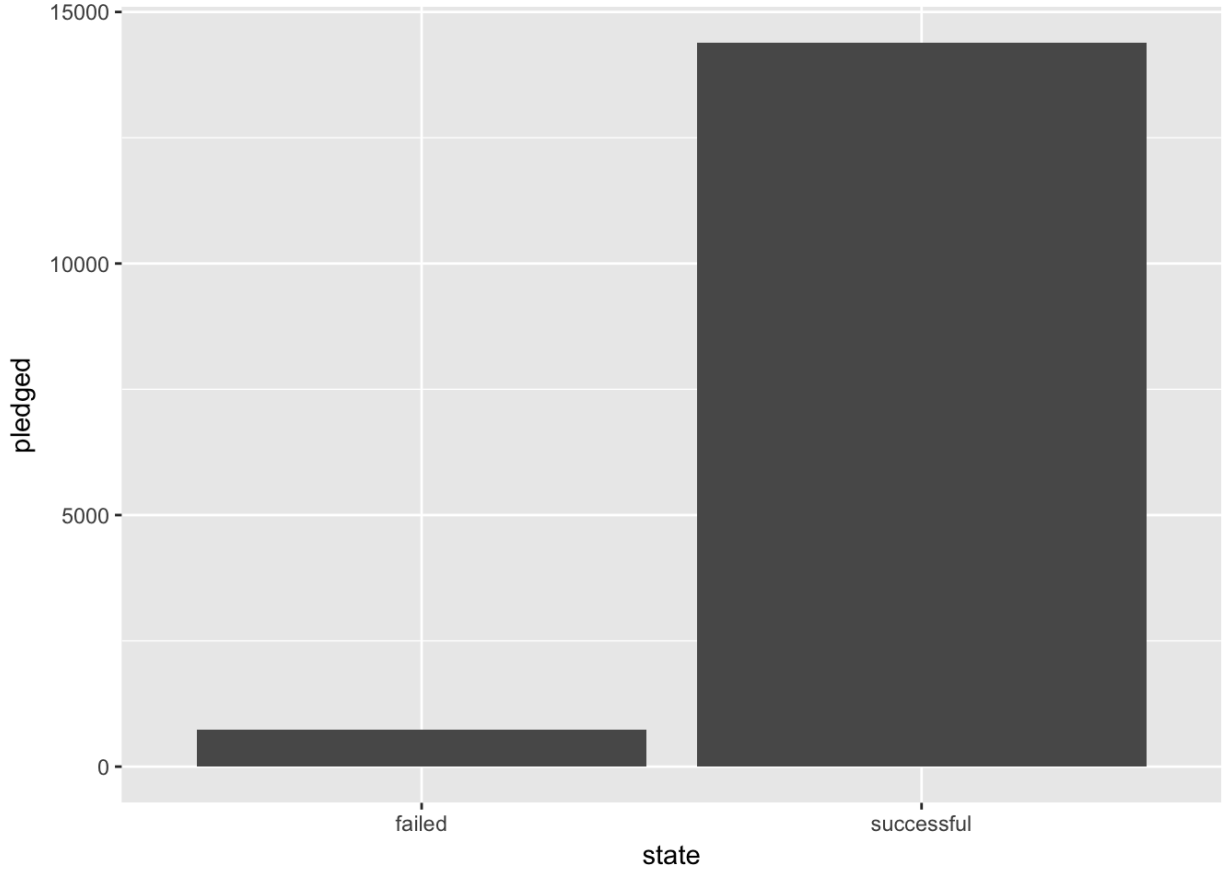
Based on the question we are trying to answer, we will be building a model in the next phase which attempts to predict or explain success of Kickstarter projects. While actually building the model is the subject of Phase 3, during this phase we are essentially preparing ourselves and our data to build that model. One of our underlying goals is to be thinking of ways to answer our question. Because the variable contains information about the success or failure of each project, it becomes an obvious candidate to be our dependent variable. Being a binary variable, meaning each case falls into one of two categories, we will have the opportunity to build a logistic regression model. Logistic regression models attempt to explain the relationship between a binary dependent variable and one or more independent variables.

When the time comes, however, I will also want to build another type of model: one using linear regression. Linear regression uses a continuous, rather than categorical, dependent variable. In this particular situation, this may prove important because it will have the potential to explain degrees of success rather than simply success or failure. For example, if the binary view is taken, projects that barely miss their goal are classified as just as much of a failure as projects that get no funding at all. Similarly, projects that barely meet their goal are viewed the same as projects that exceed their goal by ten fold. Each view may by useful in different situations so we will be sure you are comfortable using both.

With the goal in mind of also being able to create a linear regression model, we need to begin thinking about the dependent variable to use. “state” is out because it is categorical. “pledged” is the next obvious candidate, as its value certainly demonstrates success.

Let’s quickly make a few graphs to see how pledged relates to state.

```
ks %>%  ggplot(aes(state, pledged)) +
    geom_bar(stat = "summary_bin", fun.y = "mean")
```



So the successful projects had more money pledged to them than failed projects. No surprise there! But I can think of at least a couple of issues we may have by looking at this particular relationship.

First, the “pledged” variable is that by itself it does not contain enough information to help us. For example, pledges of $1000 would be a massive success for a project with a goal of $100, but for others that same $1000 would mean massive failure. Therefore we have to consider the amount pledged relative to each project’s goal.

The second problem is that we have a variable called pledged and another called usd_pledged. There’s also a couple of other variables relating to currency. What does this tell us? It’s pretty easy to deduce that some of our projects may be listed in currencies other than US dollars. As it turns out, “pledged” and “goal” are in home currency, while “usd_pledged” is standardized. Therefore, on their own, both “pledged” and “goal” cannot be compared to each other, because there are many different currencies involved. We could just use “usd_pledged” since it has been standardized, but we don’t have a “usd_goal”. We should be able to make one easily using `mutate()`.

But there is a much simpler way to deal with it all, including the problem mentioned above about “pledged” not giving us a very accurate indication of the success of the project. All we have to do to solve all of these problems is create a new variable that shows the ratio of “pledged” to “goal”. Since both should be in the same currency, the ratio of one to the other should tell us all we need to know about the success of the project.

Note: We could dig deep to explore and prove all of that, but I’m going to keep things from getting unnecessarily complicated and just show how to fix it. If you don’t buy my assertions above or want to explore a little more deeply, let’s discuss ways to prove or disprove them in the comments. There is more than one way to do so, so let’s see what you come up with! And of course I’m happy to answer questions about it as well.

To deal with the problems mentioned above, we will create a new variable that contains the ratio of the amount pledged to the goal for each project. We have actually already used a variable just like this when we were exploring the cancelled projects before we decided to remove them. We just didn’t make it permanent at the time.

To refresh your memory, here is how to create the temporary variable:

```
ks %>%
    mutate(pledged_to_goal = pledged / goal)
```

id <int>
832570985
1054135507
1567381435
1329110416
568986755
468138435
305131391
1098019088
700642359
1927640262

1-10 of 448 rows | 1-1 of 28 columns Previous **1** 2 3 4 5 6 … 45 Next

If you scroll to the right in the output you'll see that we have all of the columns we already had in the dataset, with the addition of pledged_to_goal at the very end. Therefore, all we have to do to actually add the new variable to our working dataset is to assign the output to the variable we have been using to hold our dataset. We are essentially rewriting that object, rather than actually adding to it.

```
ks <- ks %>%
    mutate(pledged_to_goal = pledged / goal)
```

And we can check it by running the 'head()' function. Note that had we put the whole thing above into parentheses, it would have both added the variable and shown the output.

```
head(ks)
```

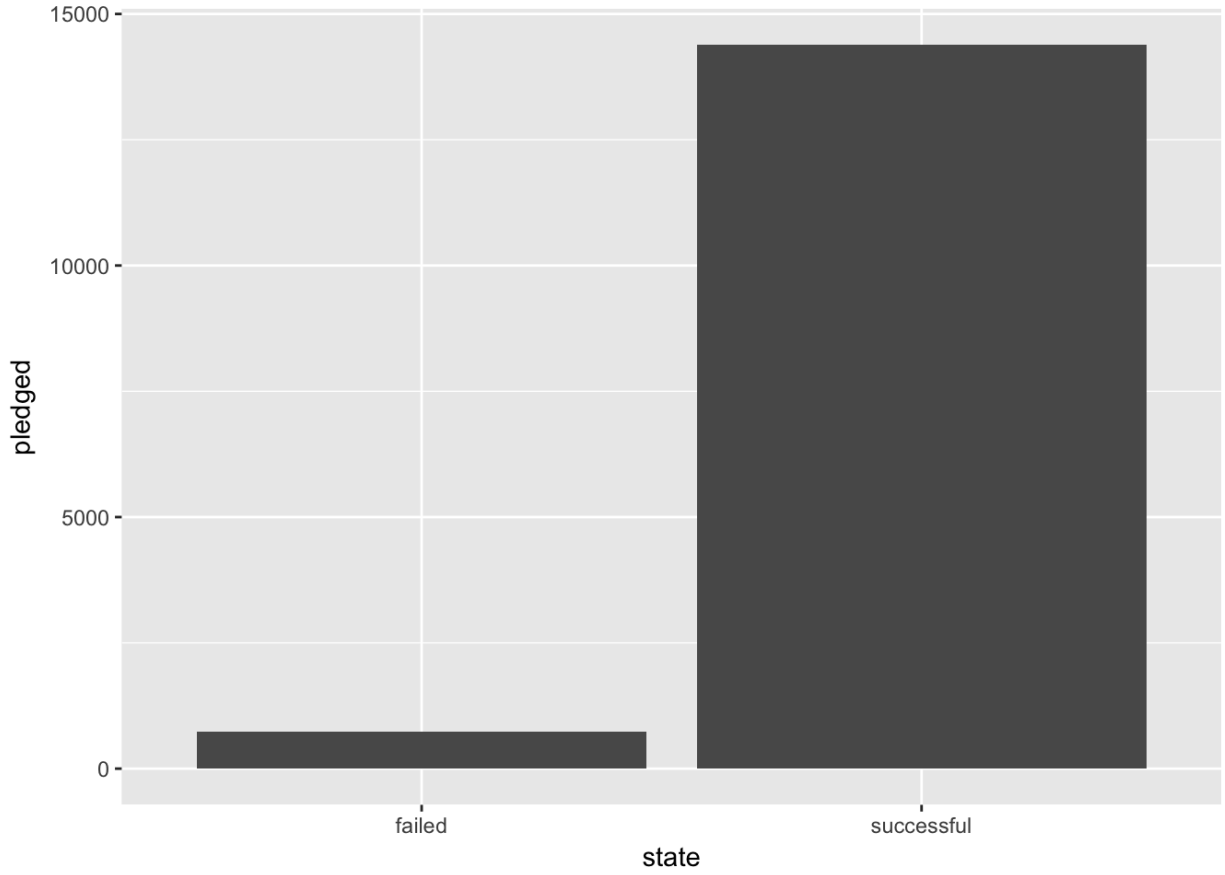
id <int>
832570985
1054135507
1567381435
1329110416
568986755
468138435

6 rows | 1-1 of 28 columns

Since we just changed our dataset by adding the new variable, let's go ahead and add it to the codebook.

Let's continue exploring the state variable. An obvious choice to explore a little further might be the average amount pledged for each of the two categories left (failed and successful).

```
ks %>% ggplot(aes(state, pledged)) +
    geom_bar(stat = "summary_bin", fun.y = mean)
```



No surprise here that the successful projects received more funding. We can see that the successful projects received pledges a little under $15000, while those that failed received what appear to be under $1000. We can easily get those numbers.

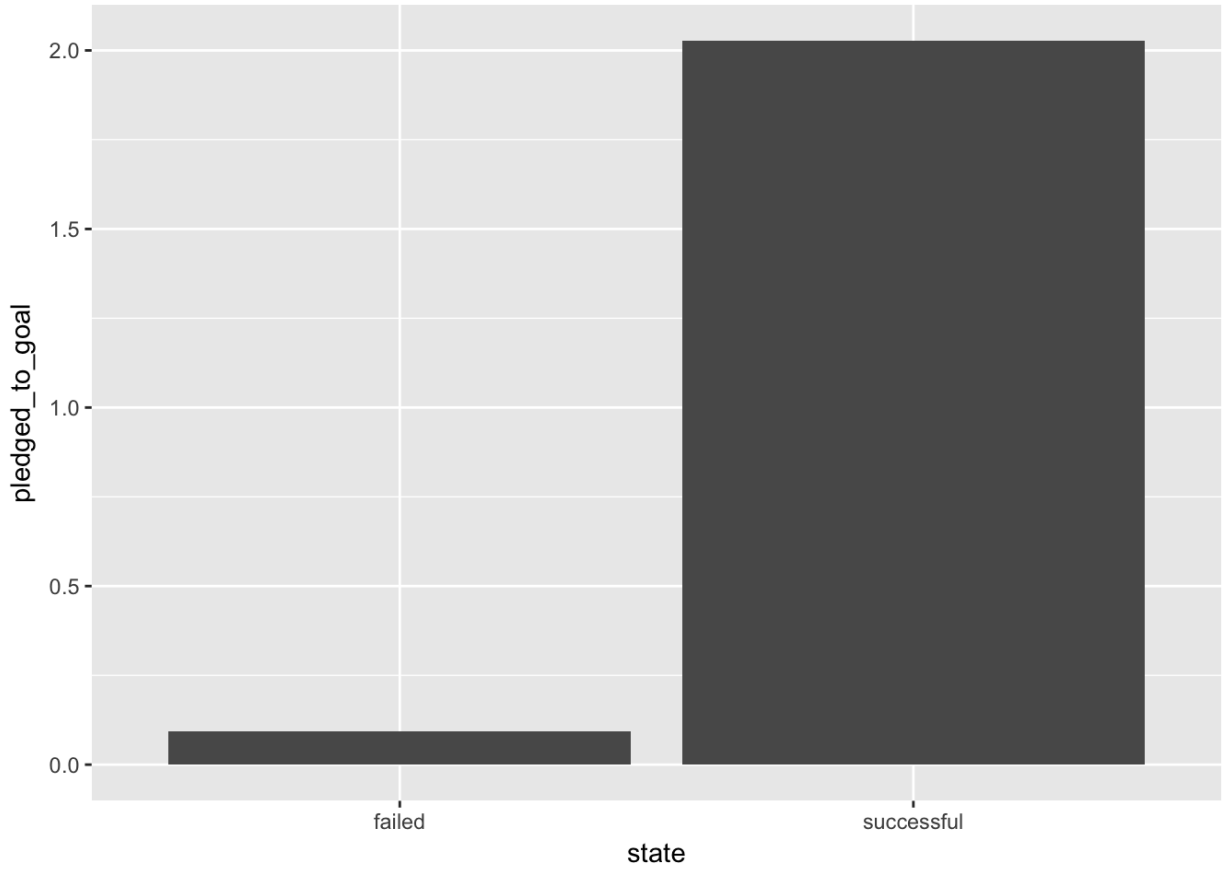
```
ks %>% group_by(state) %>%
    summarise(mean(pledged))
```

state <fctr>	**mean(pledged)** <dbl>
failed	734.6099
successful	14390.0189

2 rows

But since these numbers don't account for the difference between large and small projects, let's use our new, relative variable.

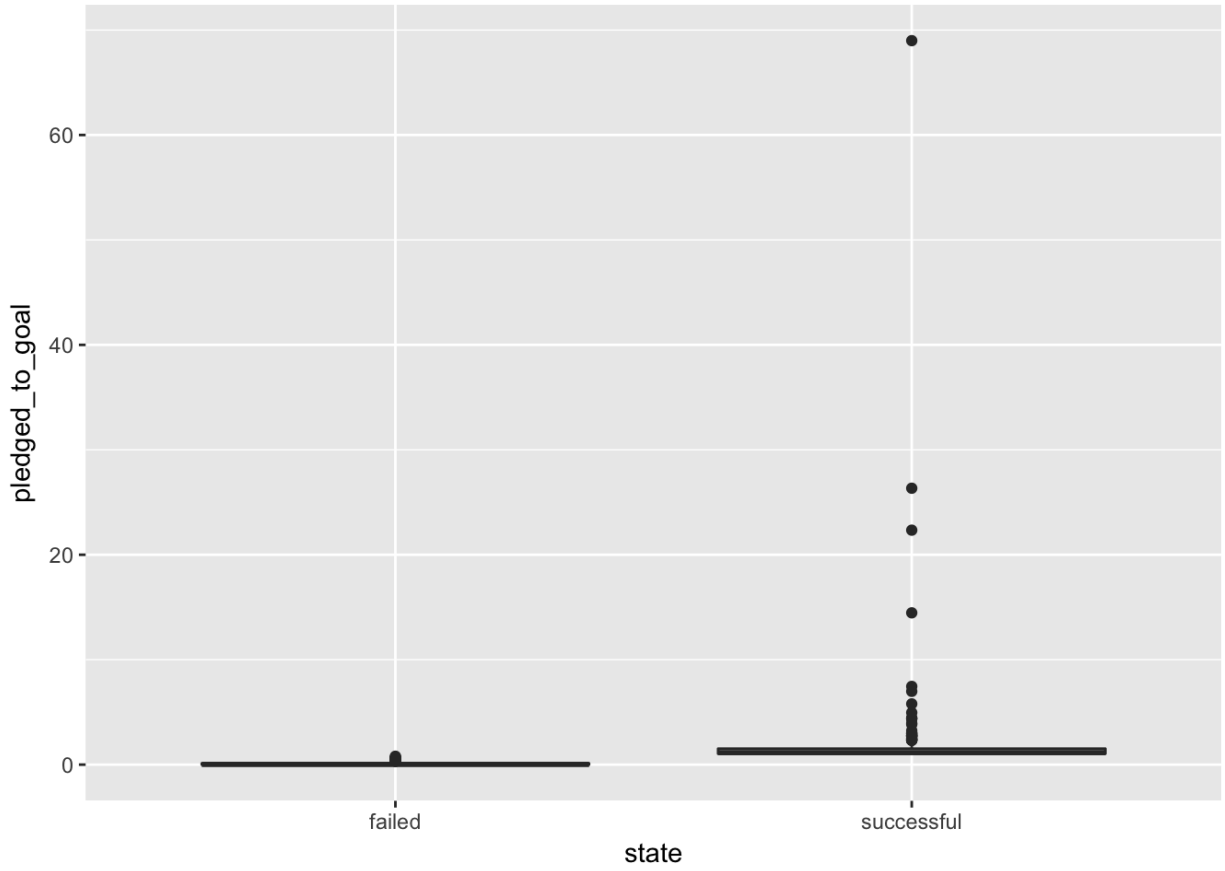
```
ks %>% ggplot(aes(state, pledged_to_goal)) +
    geom_bar(stat = "summary_bin", fun.y = mean)
```



Pretty similar looking graph, but notice the units on the y axis on the left We now have a range from pretty close to 0 for failed to overtwo times the amount pledged for successful projects.

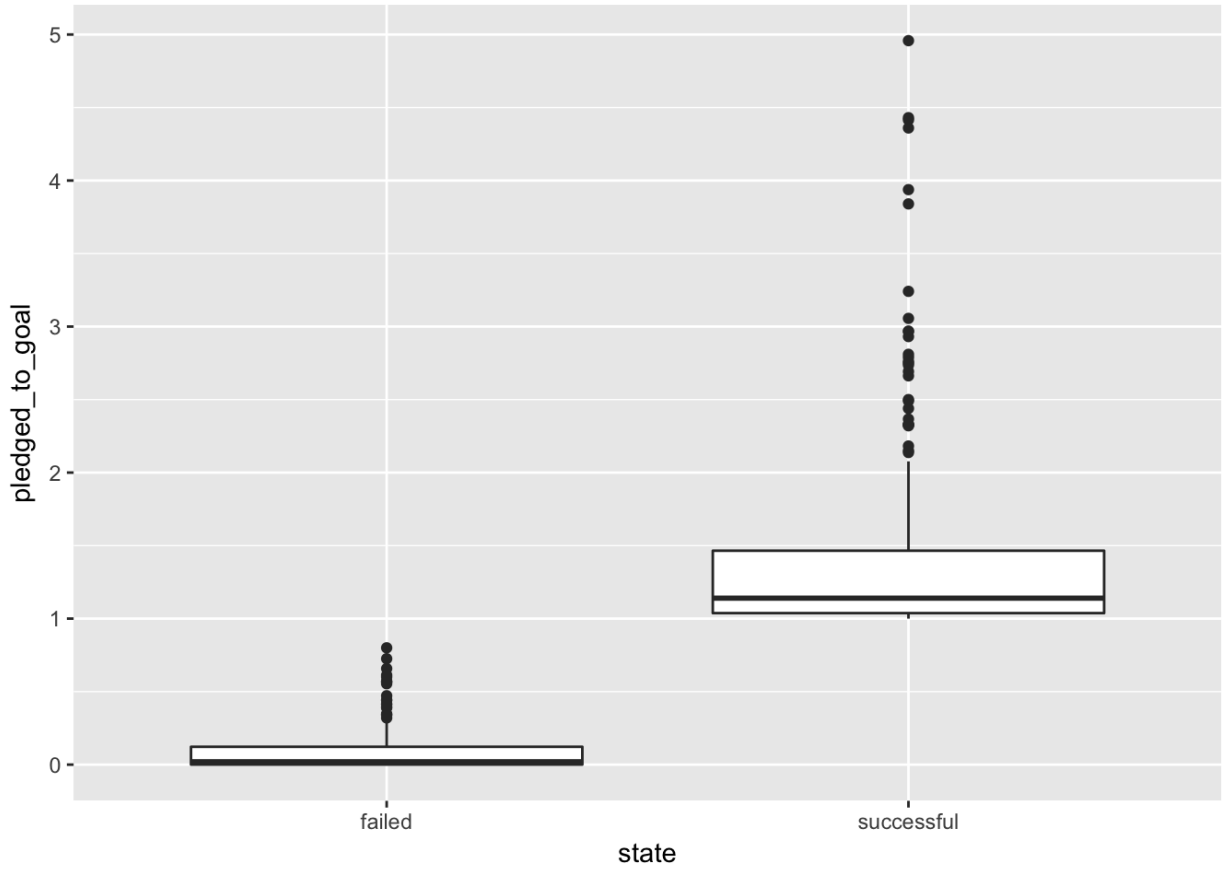
Logically, we would think that each of the failed projects should have a pledged_to_goal of less than 1, and each of the successful projects should have a pledged_to_goal of greater than or equal to one. A boxplot would be a great way to check that.

```
ks %>% ggplot(aes(state, pledged_to_goal)) +
    geom_boxplot()
```



Because of outliers, that tells us very little. Let's just do some quick filtering to make it a little more readable. From the looks of the outliers above, a max of about 5 should do the trick.

```
ks %>% filter(pledged_to_goal < 5) %>%
    ggplot(aes(state, pledged_to_goal)) +
    geom_boxplot()
```



Looks like it confirms what should have been true about pledged_to_goal for failed and successful projects: failed are all less than 1, while successful projects are all greater than 1.

You could say that in the full interest of accuracy that we could possibly have filtered something out that invalidates that assumption. We can check it manually with a filter just to be sure.

```
ks %>% filter(state == "failed", pledged_to_goal >= 1)
```

0 rows | 1-10 of 28 columns

```
ks %>% filter(state == "successful", pledged_to_goal < 1)
```

0 rows | 1-10 of 28 columns

No problems or surprises here!

Let’s recap a little of what we just did. We started off by separating the data into categories: failed or successful (the state variable), and plotting how much was pledged to each. We realized the logical deficiency in looking at “pledged” by itself, so we created a new variable consisting of a ratio of pledged to goal. We then checked that the values given made sense, and they do.

Let’s move on to the next variable on our dependent list.

state by goal

We have already dealt with goal a little while including creating our new variable, pledged_to_goal and while considering the currency issue. Since we know that goal is home currency, we can’t compare one to another with any validity. Let’s start off by converting the goal variable to US dollars.

```
ks <- ks %>% mutate(exchange_rate = usd_pledged / pledged, usd_goal = goal * exchange_rate)

ks %>% select(currency, exchange_rate, pledged, usd_pledged, goal, usd_goal)
```

currency <fctr>	exchange_rate <dbl>	pledged <dbl>	usd_pledged <dbl>	goal <dbl>	usd_goal <dbl>
GBP	1.64810624	2371.00	3.907660e+03	2300.00	3790.6444
USD	1.00000000	205.00	2.050000e+02	1700.00	1700.0000
USD	1.00000000	130.00	1.300000e+02	75000.00	75000.0000
USD	1.00000000	20917.00	2.091700e+04	7500.00	7500.0000
USD	1.00000000	1.00	1.000000e+00	109000.00	109000.0000
GBP	1.69726554	265.00	4.497754e+02	250.00	424.3164
USD	1.00000000	1095.00	1.095000e+03	750.00	750.0000
USD	1.00000000	1465.00	1.465000e+03	1000.00	1000.0000
USD	1.00000000	2.00	2.000000e+00	5000.00	5000.0000
GBP	1.54285988	10.00	1.542860e+01	1000.00	1542.8599

1-10 of 448 rows Previous **1** 2 3 4 5 6 … 45 Next

```
# remember that you can use intermediate variables in the same call to mutate
```

So a few things to notice here. First off, all the projects that began in USD have an exchange rate of 1, which is exactly what we would hope for. Next, many of the on USD projects are in pounds, GBP. The interesting thing to note is that the exchange rate is not the same for every GBP project, which I attribute to exchange rate fluctuations over time. Each project was begun at a different time, and therefore has a different exchange rate.

We can check the validity of our assumptions here in a couple ways. The ultimate test would to compute a new ratio by dividing usd_pledged by usd_goal, and then see if it produces the same result as the last ratio we calculated, pledged_to_goal. After calculating the same ratio as before but using the values converted to USD. Then I will subtract one from the other, hoping that the difference is 0. I expect there could be some slight differences due to rounding errors, but we will see.

```
colnames(ks)
```

```
##  [1] "id"                    "photo"
##  [3] "name"                  "blurb"
##  [5] "goal"                  "pledged"
##  [7] "state"                 "slug"
##  [9] "country"               "currency"
## [11] "currency_trailing_code" "deadline"
## [13] "state_changed_at"       "created_at"
## [15] "launched_at"            "staff_pick"
## [17] "is_starrable"           "backers_count"
## [19] "static_usd_rate"        "usd_pledged"
## [21] "creator"                "location"
## [23] "category"               "profile"
## [25] "spotlight"              "urls"
## [27] "source_url"             "pledged_to_goal"
## [29] "exchange_rate"          "usd_goal"
```

```
ks %>% mutate(usd_pledged_to_goal = usd_pledged / usd_goal, resid_test = pledged_to_goal - usd_pledged_to
_goal) %>%
    select(usd_pledged_to_goal, pledged_to_goal, resid_test)
```

usd_pledged_to_goal <dbl>	pledged_to_goal <dbl>	resid_test <dbl>
1.030870e+00	1.030870e+00	0.000000e+00
1.205882e-01	1.205882e-01	0.000000e+00
1.733333e-03	1.733333e-03	0.000000e+00
2.788933e+00	2.788933e+00	0.000000e+00
9.174312e-06	9.174312e-06	0.000000e+00
1.060000e+00	1.060000e+00	0.000000e+00
1.460000e+00	1.460000e+00	0.000000e+00
1.465000e+00	1.465000e+00	0.000000e+00
4.000000e-04	4.000000e-04	0.000000e+00
1.000000e-02	1.000000e-02	0.000000e+00

1-10 of 448 rows Previous **1** 2 3 4 5 6 … 45 Next

Scrolling through the output, we see several non-zero values. Let's take a closer look at those values.

```
ks %>% mutate(usd_pledged_to_goal = usd_pledged / usd_goal, resid_test = pledged_to_goal - usd_pledged_to
_goal) %>%
    select(usd_pledged_to_goal, pledged_to_goal, resid_test) %>%
    filter(resid_test != 0)
```

usd_pledged_to_goal <dbl>	pledged_to_goal <dbl>	resid_test <dbl>
1.201282051	1.201282051	-2.220446e-16
0.070125000	0.070125000	1.387779e-17

usd_pledged_to_goal <dbl>	pledged_to_goal <dbl>	resid_test <dbl>
0.053982006	0.053982006	6.938894e-18
0.248000000	0.248000000	-2.775558e-17
6.980200000	6.980200000	-8.881784e-16
1.035433333	1.035433333	2.220446e-16
0.000400000	0.000400000	5.421011e-20
1.005000000	1.005000000	-2.220446e-16
2.490666667	2.490666667	4.440892e-16
0.199866667	0.199866667	-2.775558e-17

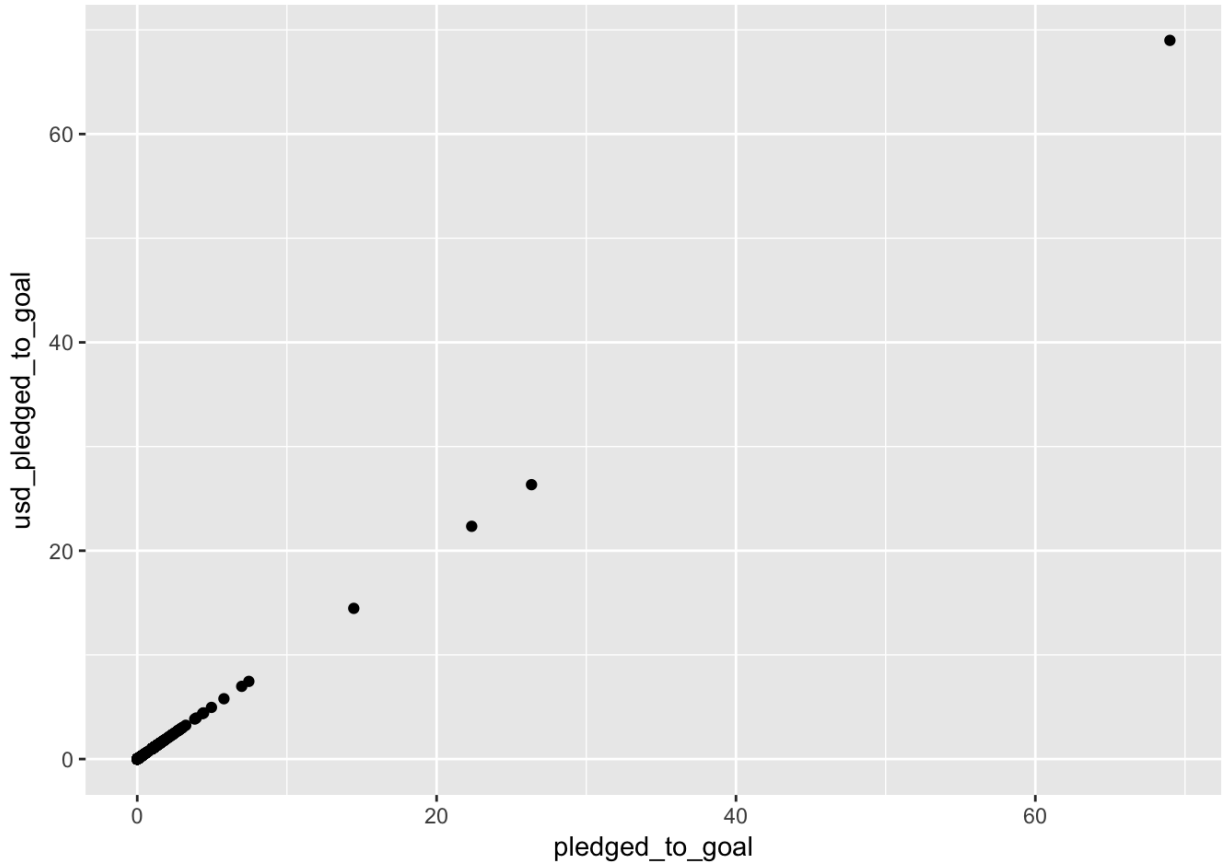
1-10 of 29 rows Previous **1** 2 3 Next

The non-zero values here are very small. The exponential notation used here each show the decimal moving left 16 or more places, meaning we are dealing with very small numbers. I think it is safe to say that these differences are due to rounding error and are not too big of a concern.

Let's do one more quick check, this time getting a visual representation of how well they correlate.

```
ks %>% mutate(usd_pledged_to_goal = usd_pledged / usd_goal) %>%
    ggplot(aes(pledged_to_goal, usd_pledged_to_goal)) +
    geom_point()
```

```
## Warning: Removed 39 rows containing missing values (geom_point).
```

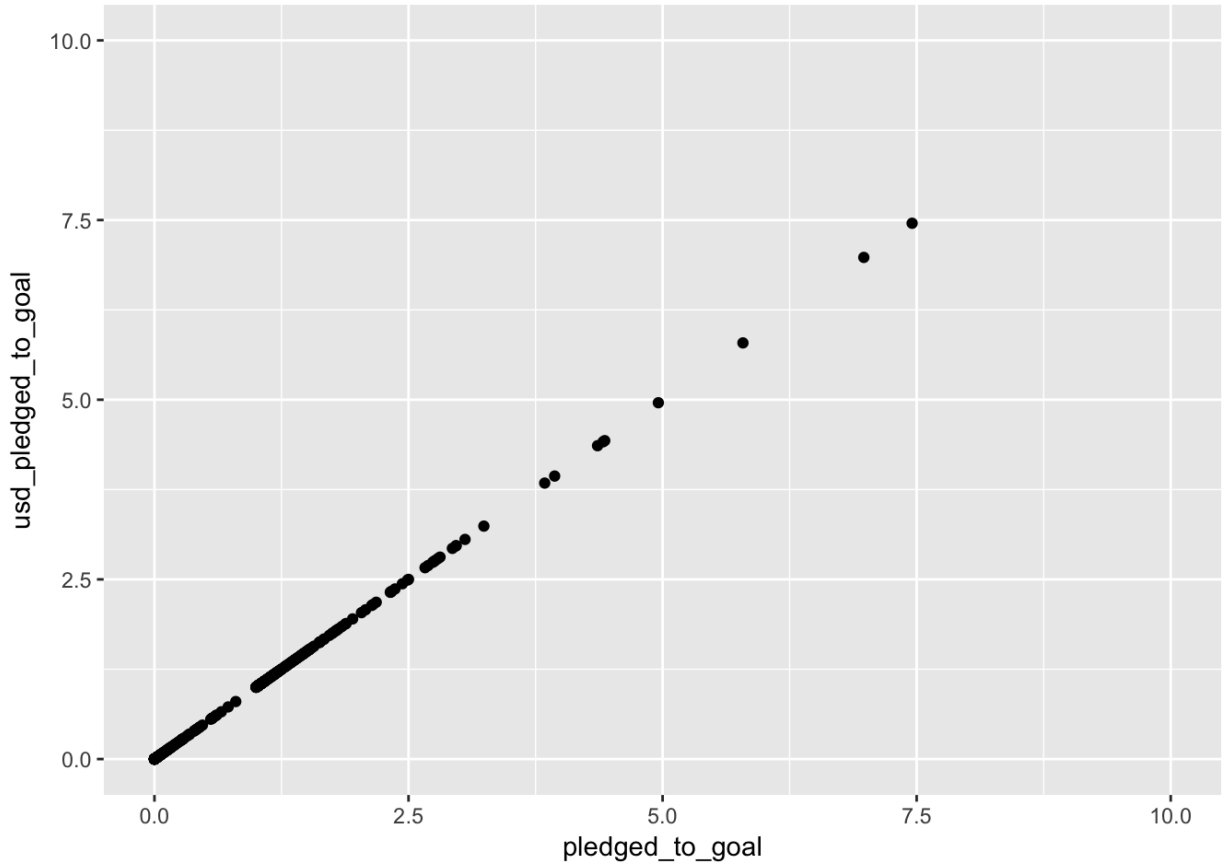


We could remove some outliers to get a better look, but I'm pretty much convinced that we're close enough.

But just for fun, let's do the same thing but remove any everything under 10.

```
ks %>% mutate(usd_pledged_to_goal = usd_pledged / usd_goal) %>%
    ggplot(aes(pledged_to_goal, usd_pledged_to_goal)) +
    geom_point() +
    xlim(NA, 10) +
    ylim(NA, 10)
```

```
## Warning: Removed 43 rows containing missing values (geom_point).
```



Looks pretty spot on to me!

References

Few, S. (2015). *Signal: Understanding what matters in a world of noise* (First Edition). Analytics Press.

Wickham, H. (2010). A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, *19*(1), 3–28.

Wickham, H. (2016). *Ggplot2: Elegant graphics for data analysis* (Second Edition). Springer.

Wickham, H., & Grolemund, G. (2017). *R for data science* (First Edition). O'Reilly Media.