

Lab 5 Solutions

Stats 32: Introduction to R for Undergraduates

Harrison Li

04/16/2024

Note: The content of this lab is partially borrowed from Kenneth Tay's course materials in the Autumn 2019 iteration of this course.

Today we'll be working with the `diamonds` dataset from the `ggplot2` package. We want to understand the distribution of diamond prices.

Let's load the `ggplot2` package and the `diamonds` dataset. It's part of the tidyverse. Look at the documentation to understand what the dataset is about.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
## v dplyr      1.1.4      v readr      2.1.5
## v forcats   1.0.0      v stringr   1.5.1
## v ggplot2   3.4.4      v tibble    3.2.1
## v lubridate 1.9.3      v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
data(diamonds)
?diamonds
```

As usual, we can use `str()` or `head()` to get a birds' eye view of the dataset:

```
str(diamonds)
```

```
## tibble [53,940 x 10] (S3: tbl_df/tbl/data.frame)
## $ carat : num [1:53940] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
## $ cut   : Ord.factor w/ 5 levels "Fair"<"Good"<...: 5 4 2 4 2 3 3 3 1 3 ...
## $ color : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<...: 2 2 2 6 7 7 6 5 2 5 ...
## $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<...: 2 3 5 4 2 6 7 3 4 5 ...
## $ depth : num [1:53940] 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
## $ table : num [1:53940] 55 61 65 58 58 57 57 55 61 61 ...
## $ price : int [1:53940] 326 326 327 334 335 336 336 337 337 338 ...
## $ x     : num [1:53940] 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
## $ y     : num [1:53940] 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
## $ z     : num [1:53940] 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

```
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut          color clarity depth table price     x     y     z
```

```
##   <dbl> <ord>      <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal      E     SI2    61.5  55   326  3.95  3.98  2.43
## 2  0.21 Premium   E     SI1    59.8  61   326  3.89  3.84  2.31
## 3  0.23 Good      E     VS1    56.9  65   327  4.05  4.07  2.31
## 4  0.29 Premium   I     VS2    62.4  58   334  4.2   4.23  2.63
## 5  0.31 Good      J     SI2    63.3  58   335  4.34  4.35  2.75
## 6  0.24 Very Good J     VVS2   62.8  57   336  3.94  3.96  2.48
```

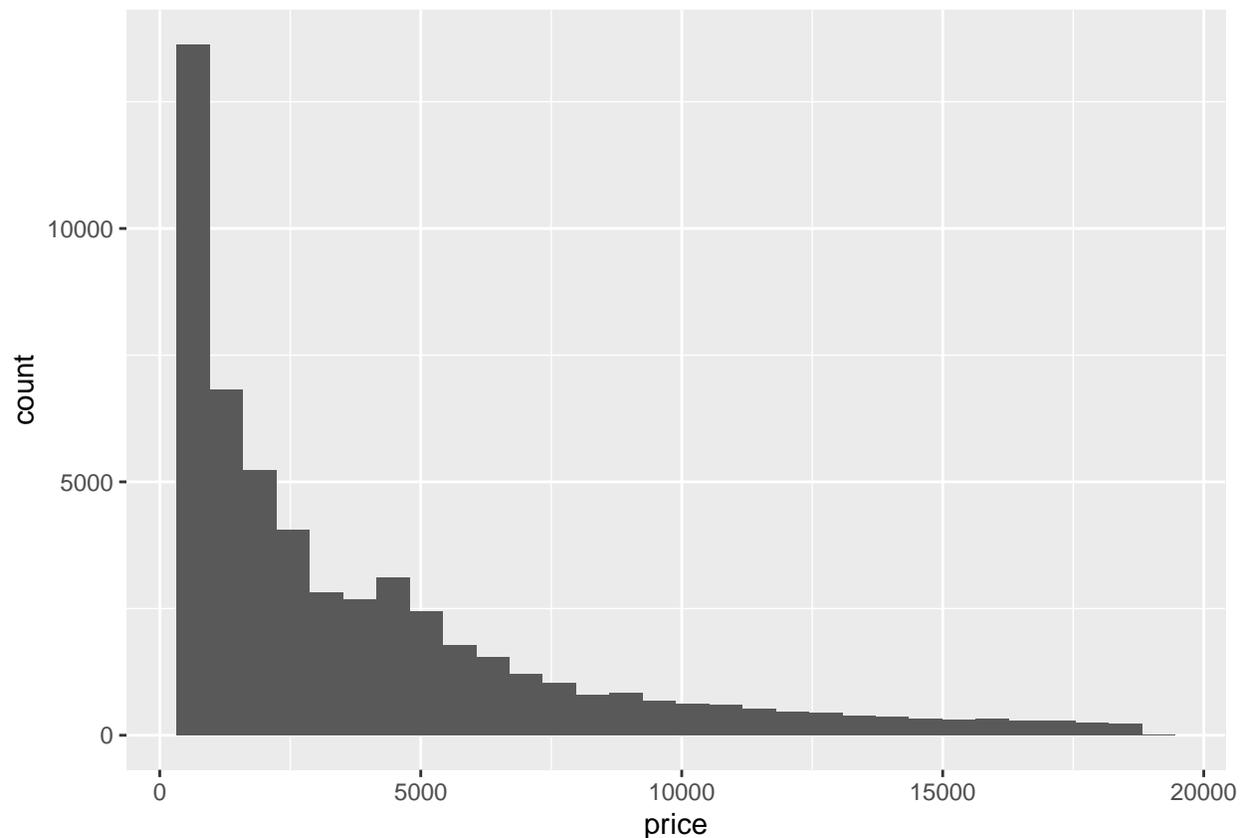
Histograms

A histogram is useful for visualizing the distribution of a single quantitative variable.

Let's start with a simple histogram of price:

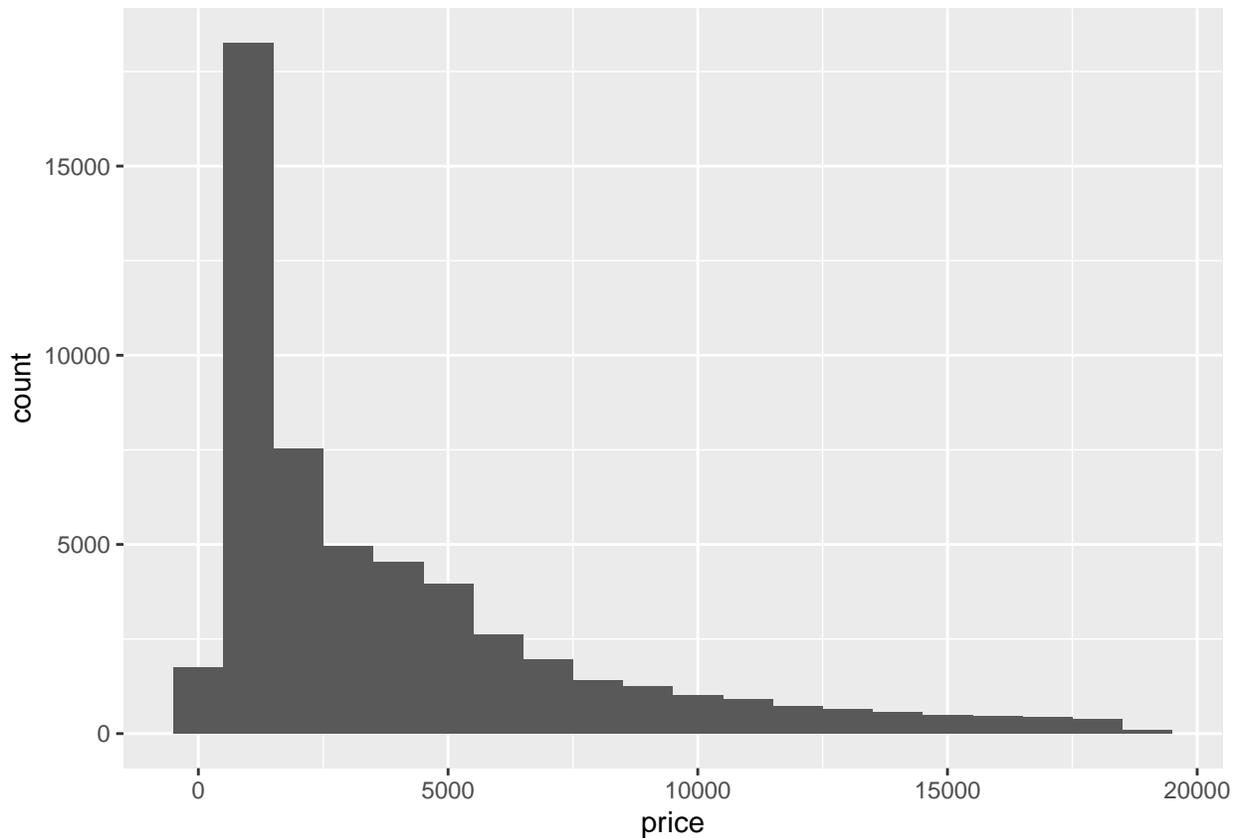
```
diamonds %>%
  ggplot(aes(x=price)) +
  geom_histogram()
```

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



Let's say you wanted to see the histogram binned every \$1,000. This can be done using the `binwidth` argument (see help menu for `geom_histogram()`):

```
diamonds %>%
  ggplot(aes(x=price)) +
  geom_histogram(binwidth=1000)
```



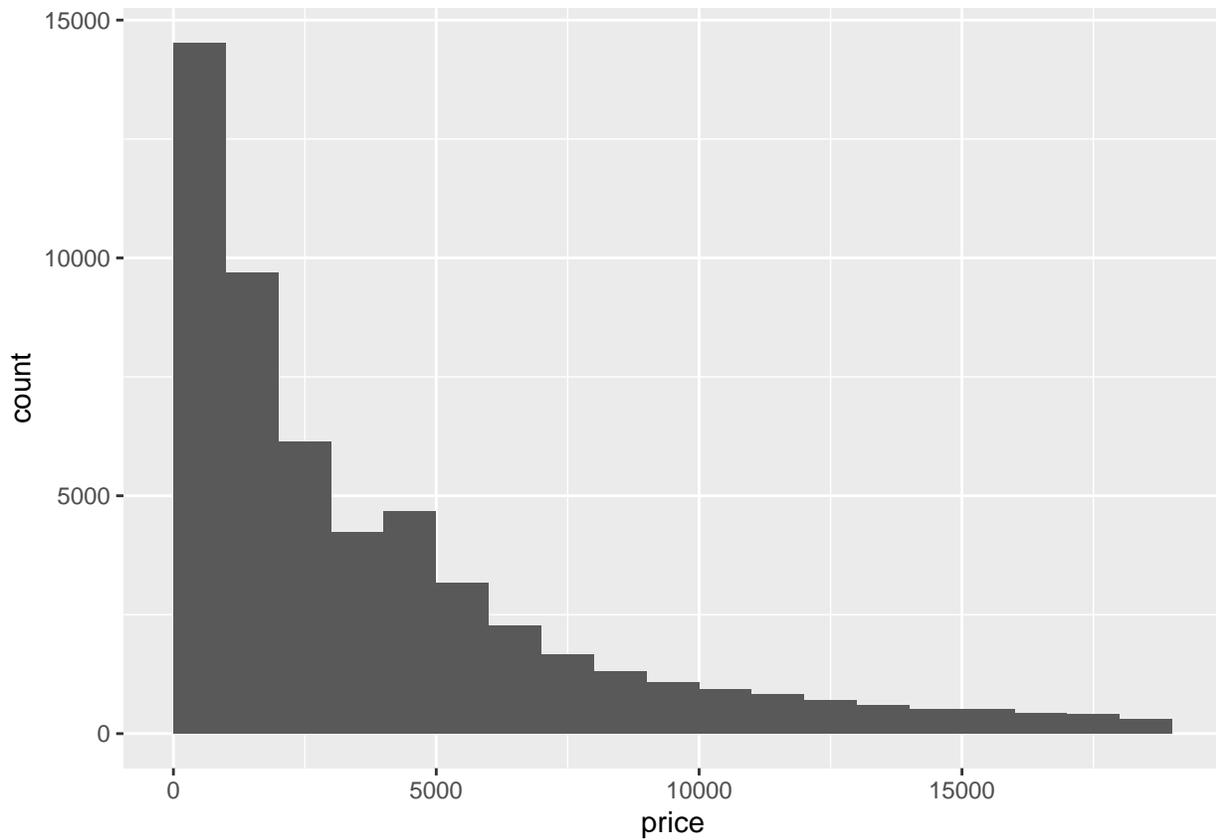
Hmm... the bins are not aligned with 0. This is because `geom_histogram()` starts binning based on the lowest observed value, which is above 0:

```
min(diamonds$price)
```

```
## [1] 326
```

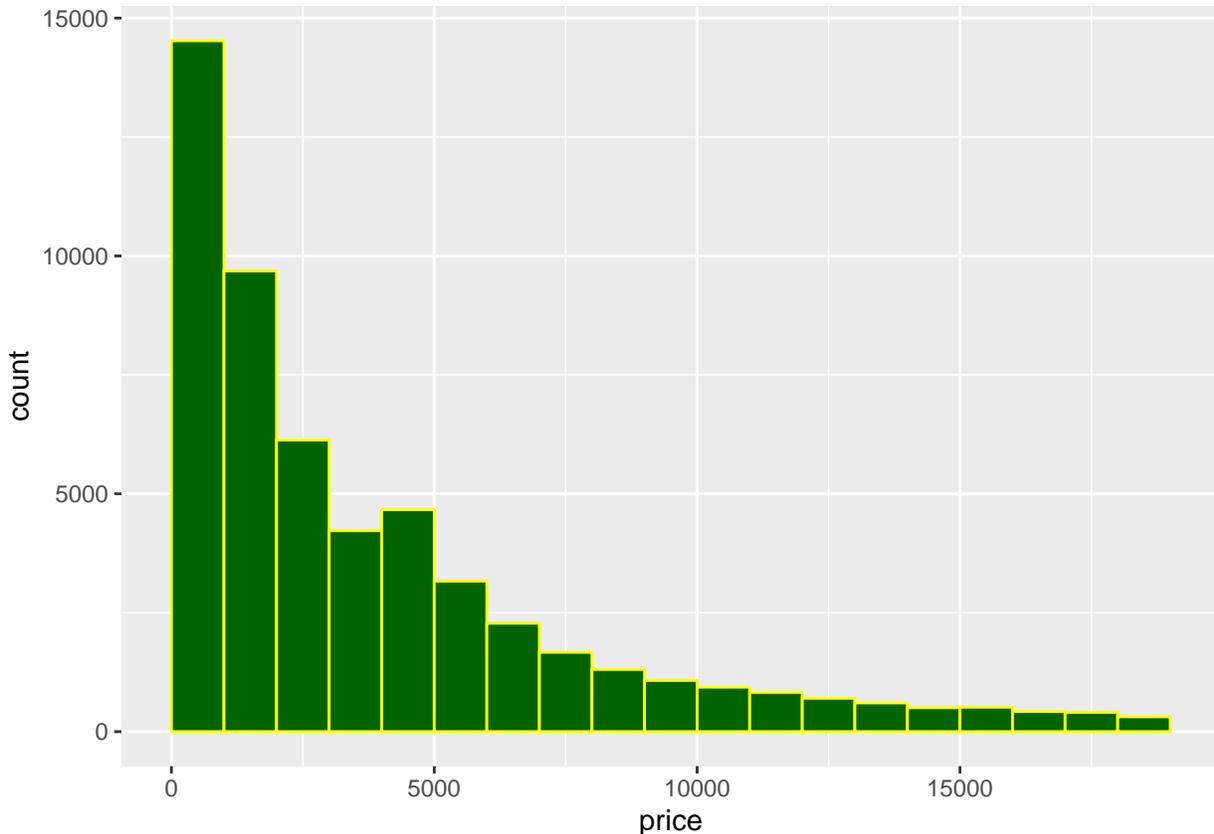
It's probably more natural to have the bins align with the even \$1,000's. This can be done by specifying the `boundary` argument:

```
diamonds %>%  
  ggplot(aes(x=price)) +  
  geom_histogram(boundary=0, binwidth=1000)
```



Lastly we can change the bar outline color and the fill. If these are static values, remember NOT to specify these inside `aes()`:

```
diamonds %>%  
  ggplot(aes(x=price)) +  
  geom_histogram(boundary=-1000000, binwidth=1000, colour="yellow", fill="darkgreen")
```



Based on the histogram, it is clear that the distribution of diamond prices is quite *right-skewed*. That is, the histogram has a long right tail. A right-skewed distribution has a small number of observations far above the typical values. Right-skewed data tend to have an average higher than the median. Recall the median of a set of numbers is the value v for which (approximately) 50% of observations are below v , and the other 50% are above v . Thus, for right-skewed data, less than half the observations will be above the mean. We will investigate this further in the homework.

On the other hand, a *left-skewed* distribution is characterized by a histogram with a long left tail. These distributions have a small number of observations far below the typical values. The abalone lengths from lecture are moderately left skewed.

1. Suppose I have some quantitative variable that is right-skewed. If I multiply all the values by -1 , will it remain right-skewed, become left-skewed, or neither? Justify your answer.

Answer: Left-skewed, since multiplying by -1 flips the ordering of the data, so that after multiplying by -1 we will have a small number of observations far above the typical value.

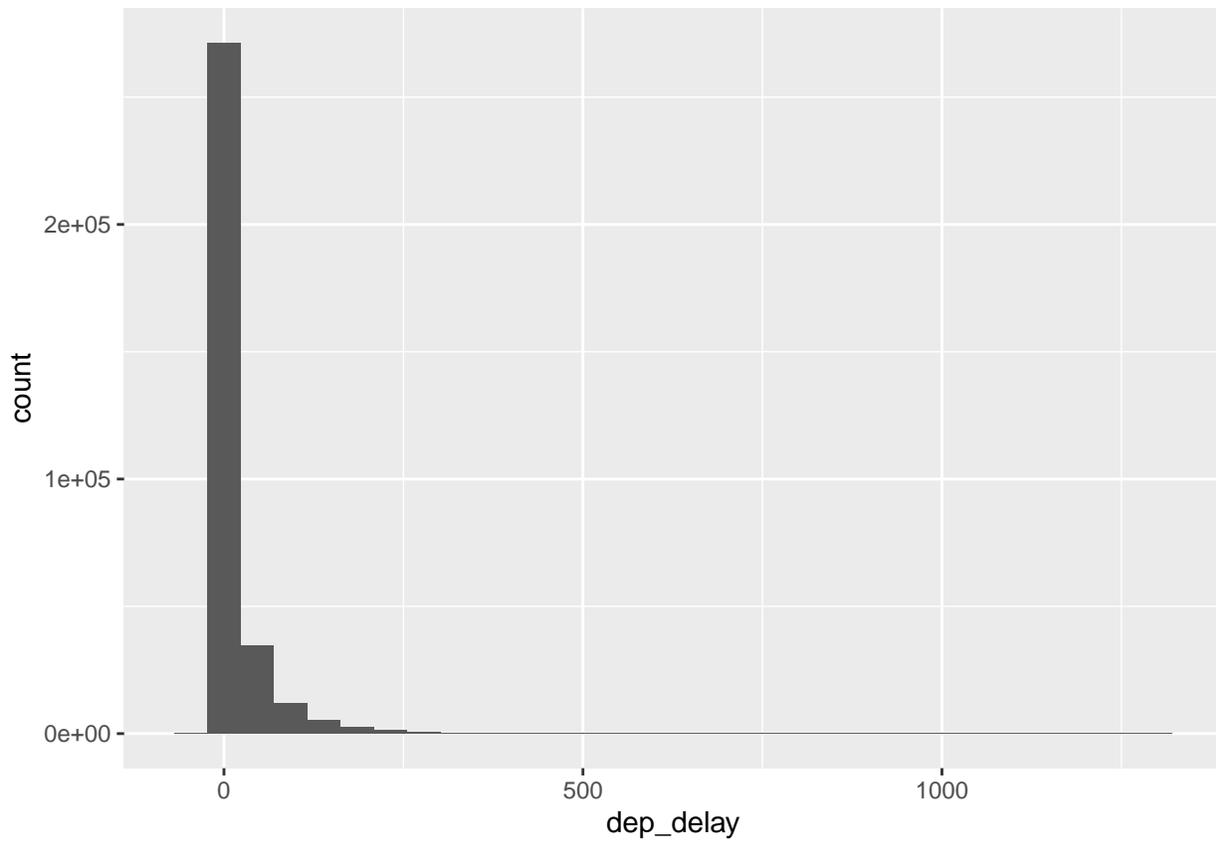
2. Load the `flights` tibble from the `nycflights13` package, and generate a histogram of departure delays for all flights departing from JFK airport.

Answer:

```
library(nycflights13)
flights %>%
  ggplot(aes(x=dep_delay)) +
  geom_histogram()
```

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

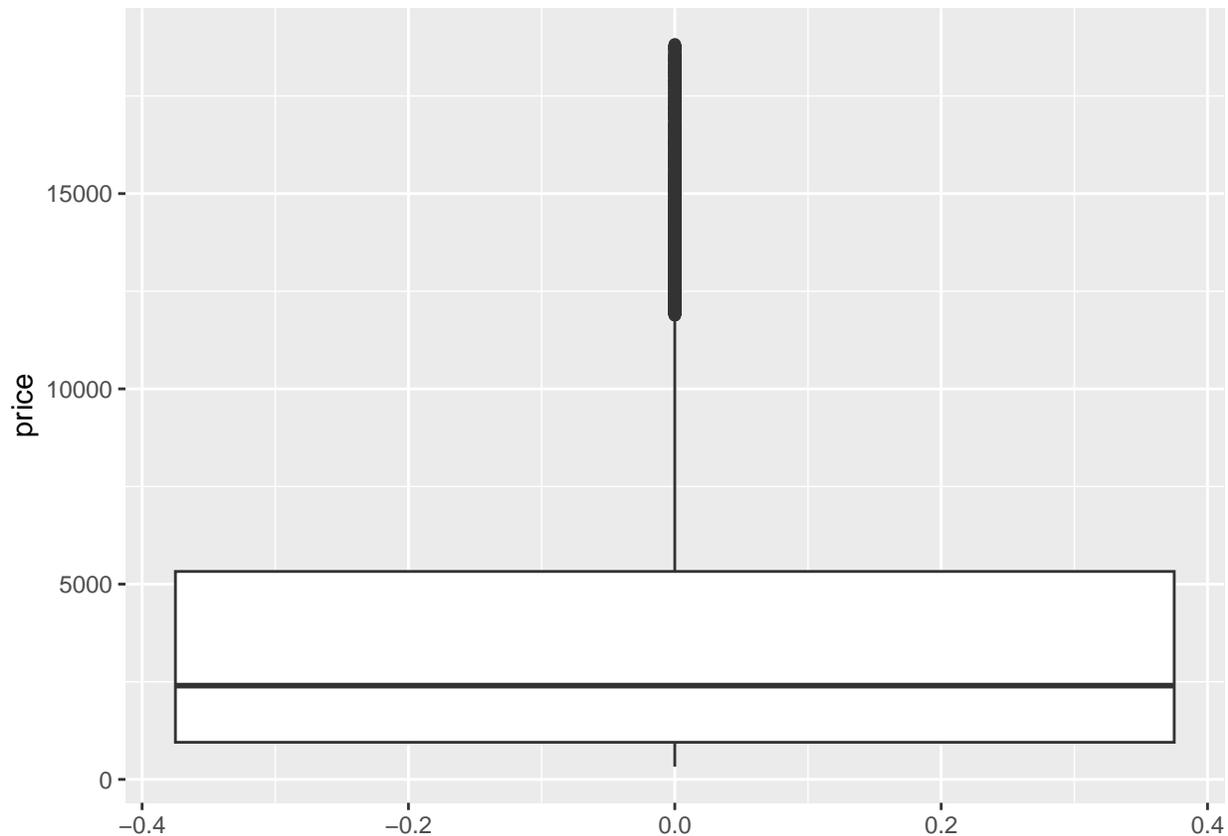
```
## Warning: Removed 8255 rows containing non-finite values (`stat_bin()`).
```



Boxplots

An alternative visualization for the distribution of a single quantitative variable is a boxplot:

```
diamonds %>%  
  ggplot(aes(y=price)) +  
  geom_boxplot()
```



The dark line around 2,500 represents the median of the data. The lower and upper edges of the box correspond to the *first and third quartiles*, respectively, a.k.a. the *25th and 75th percentiles*.

3. What is the exact median price in the `diamonds` tibble?

Answer:

```
median(diamonds$price)
```

```
## [1] 2401
```

By definition, around 25% of diamonds (in the tibble) had a price below the first quartile. Around 25% of diamonds had a price *higher* than the third quartile. About 50% of diamonds had a price below (or above) the median. Note these percentages are not exact due to different ways on how to handle rounding.

The **interquartile range (IQR)** is defined as 3rd quartile minus 1st quartile.

The vertical lines above and below the box (“whiskers”) extend out to the furthest observations within $1.5 \times \text{IQR}$ above and below the median (the 1.5 multiplier can be changed using the `coef` argument).

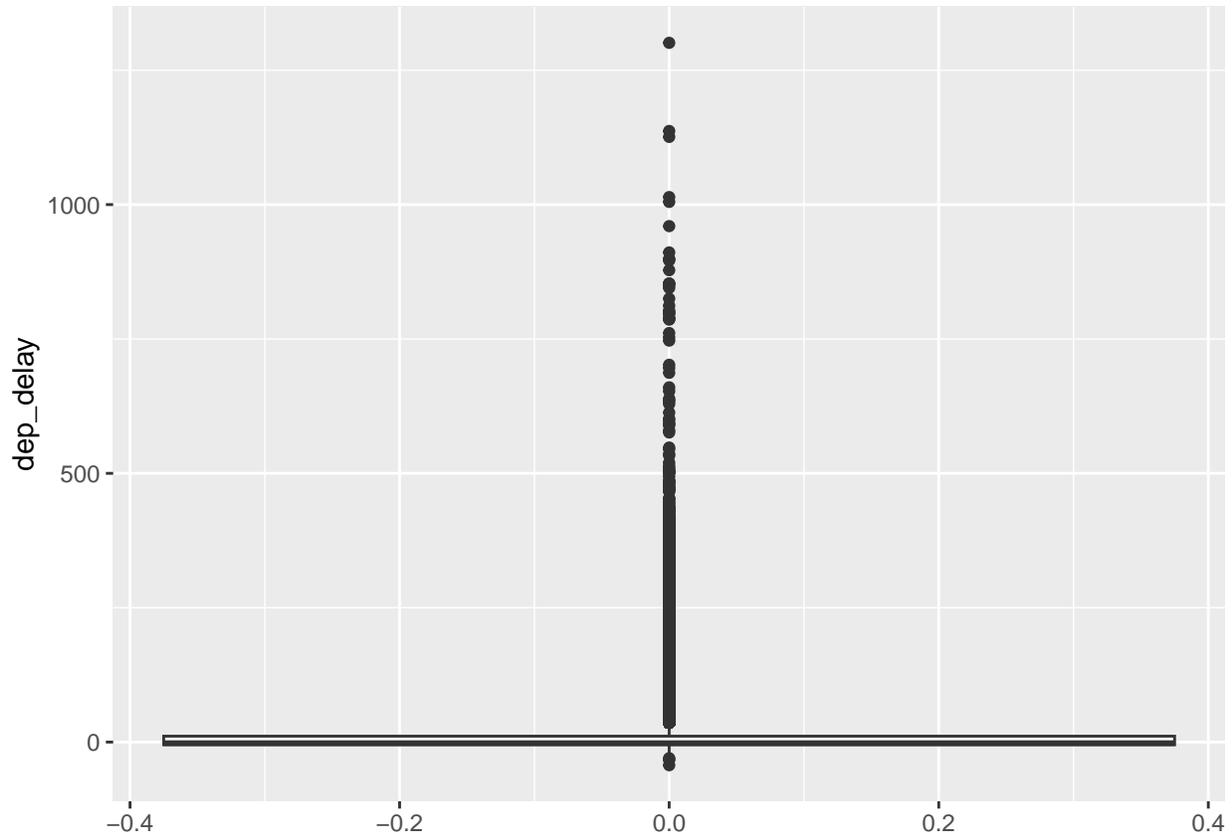
Finally, the dots above the top of the upper whisker (there are a lot, so they kind of look like a thicker vertical line) are all observations more than 1.5 times the IQR above the median. Such observations far from the center are typically called **outliers** (note that there is no single standard definition of an outlier; whether a data point is an outlier is a mostly qualitative judgment, and depends on context). There are no dots below the bottom whisker, since the minimum price is less than 1.5 IQR’s below the median. Many outliers above the median (but fewer below the median) are a good indicator of right skew.

Compared to histograms, boxplots show less detailed information, but tend to be more concise and take up less space. With a boxplot, you also don’t have to worry about picking good bins, like you do with a histogram.

4. Re-do problem 2 with a boxplot instead of a histogram.

```
flights %>%  
  ggplot(aes(y=dep_delay)) +  
  geom_boxplot()
```

```
## Warning: Removed 8255 rows containing non-finite values (``stat_boxplot``).
```



Bar plots

A bar plot is the best way to visualize the distribution of a single *categorical* variable.

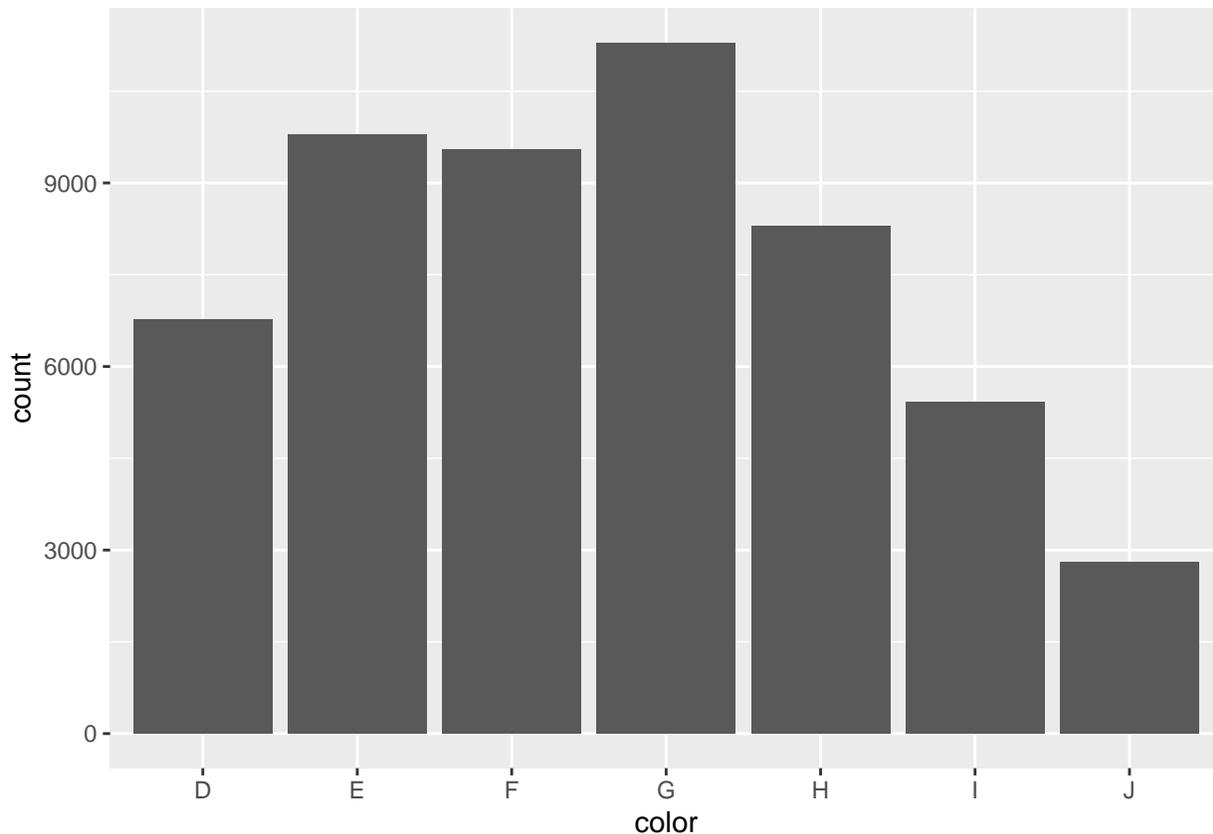
Diamonds have colors. In this dataset they range from D (best) to J (worst), and are coded as factor variables (see Lab 2):

```
levels(diamonds$color)
```

```
## [1] "D" "E" "F" "G" "H" "I" "J"
```

Let's look at the color frequency distribution of the diamonds in the tibble. Since we want counts, we can use `geom_bar()` directly:

```
diamonds %>%  
  ggplot(aes(x=color)) +  
  geom_bar()
```



There are a good number of diamonds of all colors; a plurality are of color G.

As a sanity check, let's use `summarise()` to view the raw numbers of diamonds of each color:

```
diamonds %>%
  group_by(color) %>%
  summarise(freq=n())
```

```
## # A tibble: 7 x 2
##   color freq
##   <ord> <int>
## 1 D     6775
## 2 E     9797
## 3 F     9542
## 4 G    11292
## 5 H     8304
## 6 I     5422
## 7 J     2808
```

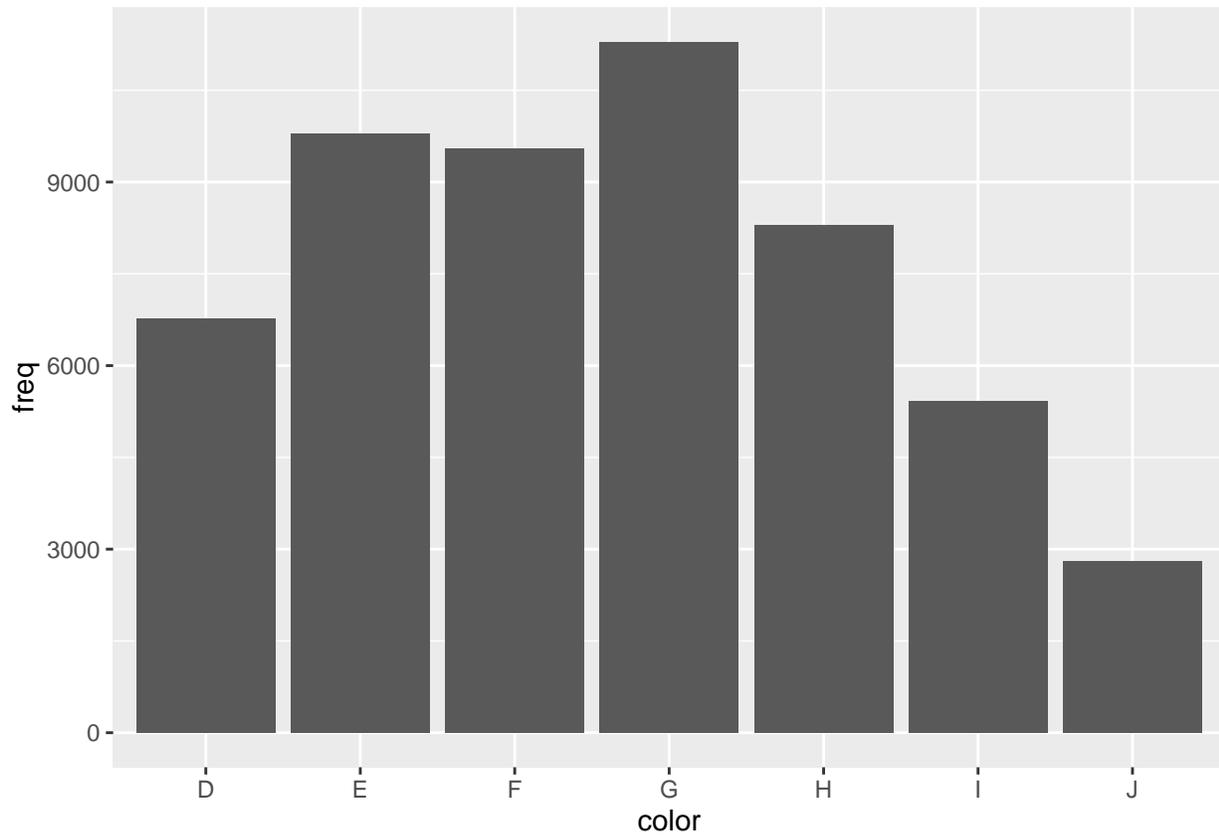
They seem to match our bar plot!

5. Reproduce the bar plot above using `geom_col()`.

Answer: To use `geom_col()` we need to use a tibble containing the counts of diamonds of each color. That's done for us in the chunk above, so we just assign it to a new variable `diamond_summary` that we pipe into a call to `ggplot()` with `geom_col()`:

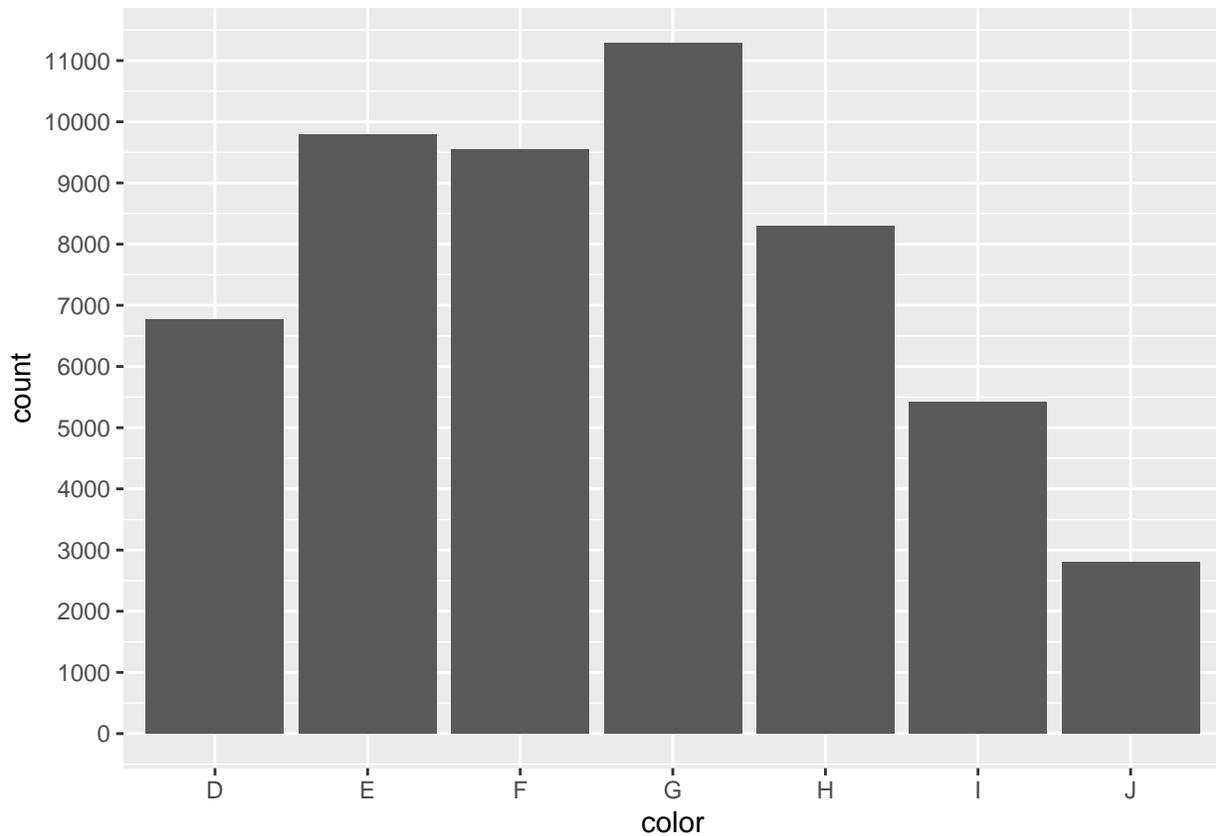
```
diamond_summary <- diamonds %>%
  group_by(color) %>%
  summarise(freq=n())
```

```
diamond_summary %>%  
  ggplot(aes(x=color, y=freq)) +  
  geom_col()
```



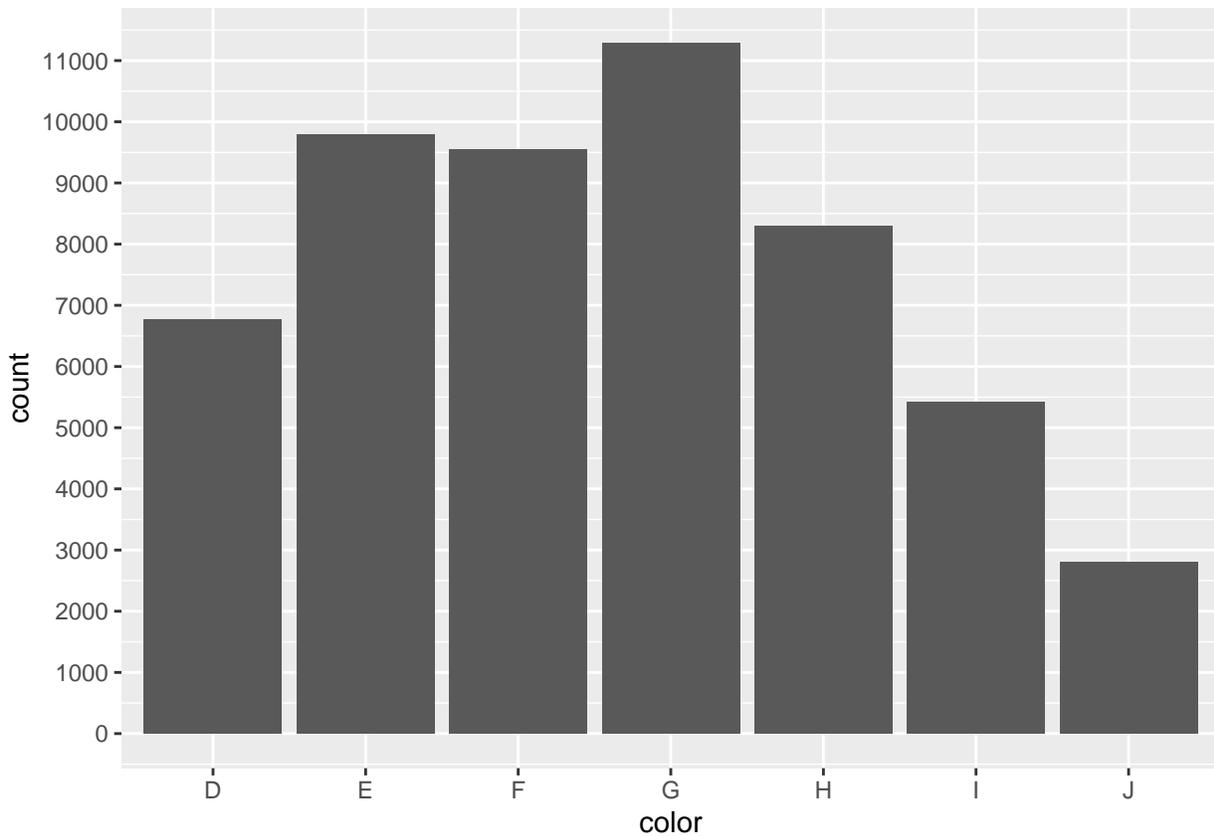
We might want to have more frequent y axis markings on our plot. We can achieve this using `scale_y_continuous()`:

```
diamonds %>%  
  ggplot(aes(x=color)) +  
  geom_bar() +  
  scale_y_continuous(n.breaks=11)
```



We choose 11 breaks so that the axis labels are every 1,000. Note the function has a preference to make the labels at nice, “round” numbers, so you don’t have to be super exact with your `n_breaks`:

```
diamonds %>%  
  ggplot(aes(x=color)) +  
  geom_bar() +  
  scale_y_continuous(n.breaks=13)
```

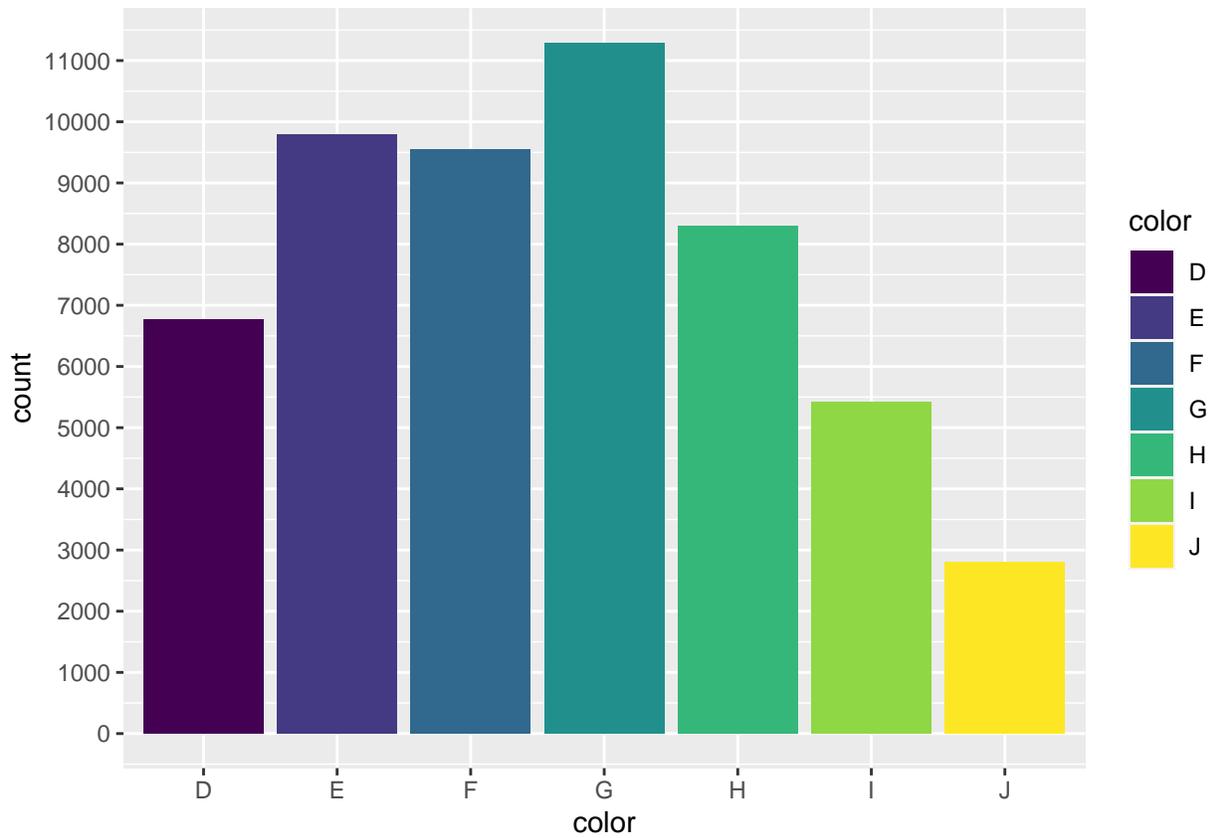


If you want even more control over the axis labels, you can specify the breaks exactly using the `breaks` argument inside `scale_y_continuous()`.

aesthetics

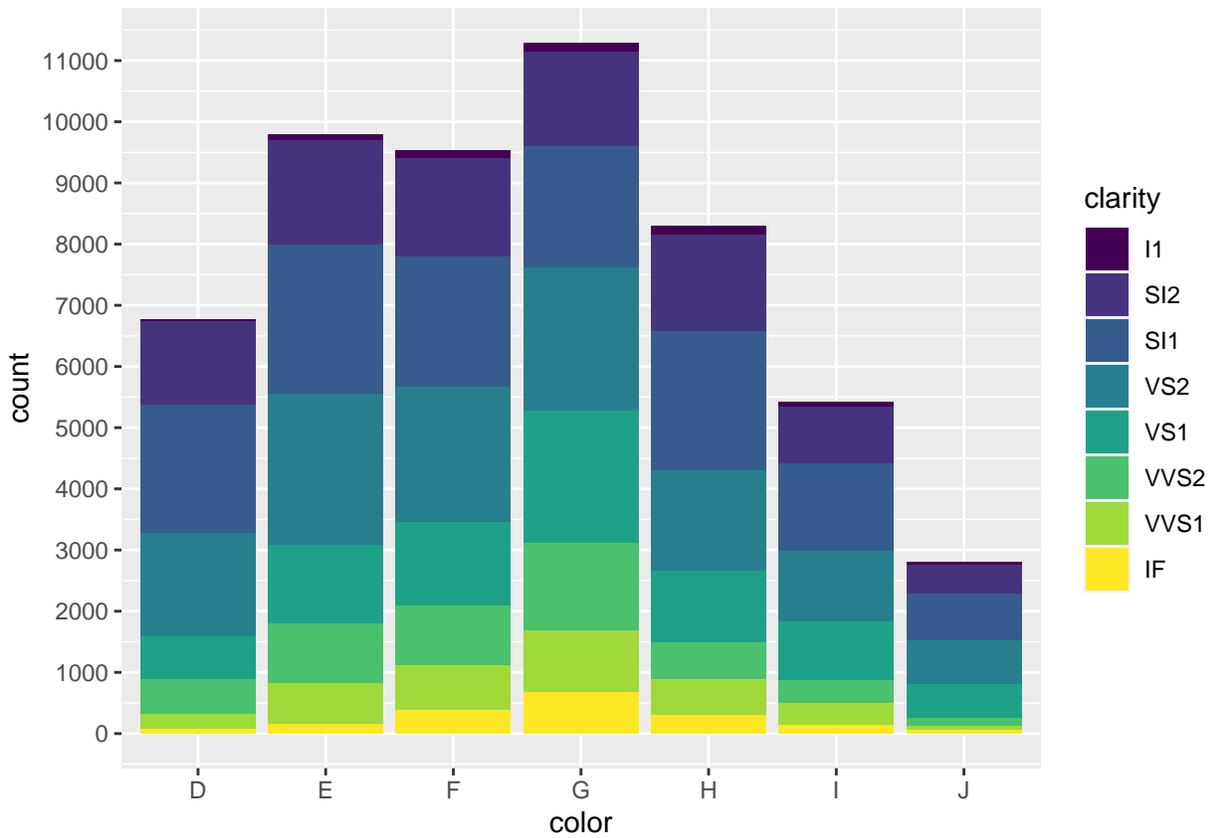
What if we wanted a different fill for each bar? Recall we use `aes()` to specify aesthetic attributes that should vary based on the data.

```
diamonds %>%  
  ggplot(aes(x=color)) +  
  geom_bar(aes(fill=color)) +  
  scale_y_continuous(n.breaks=13)
```



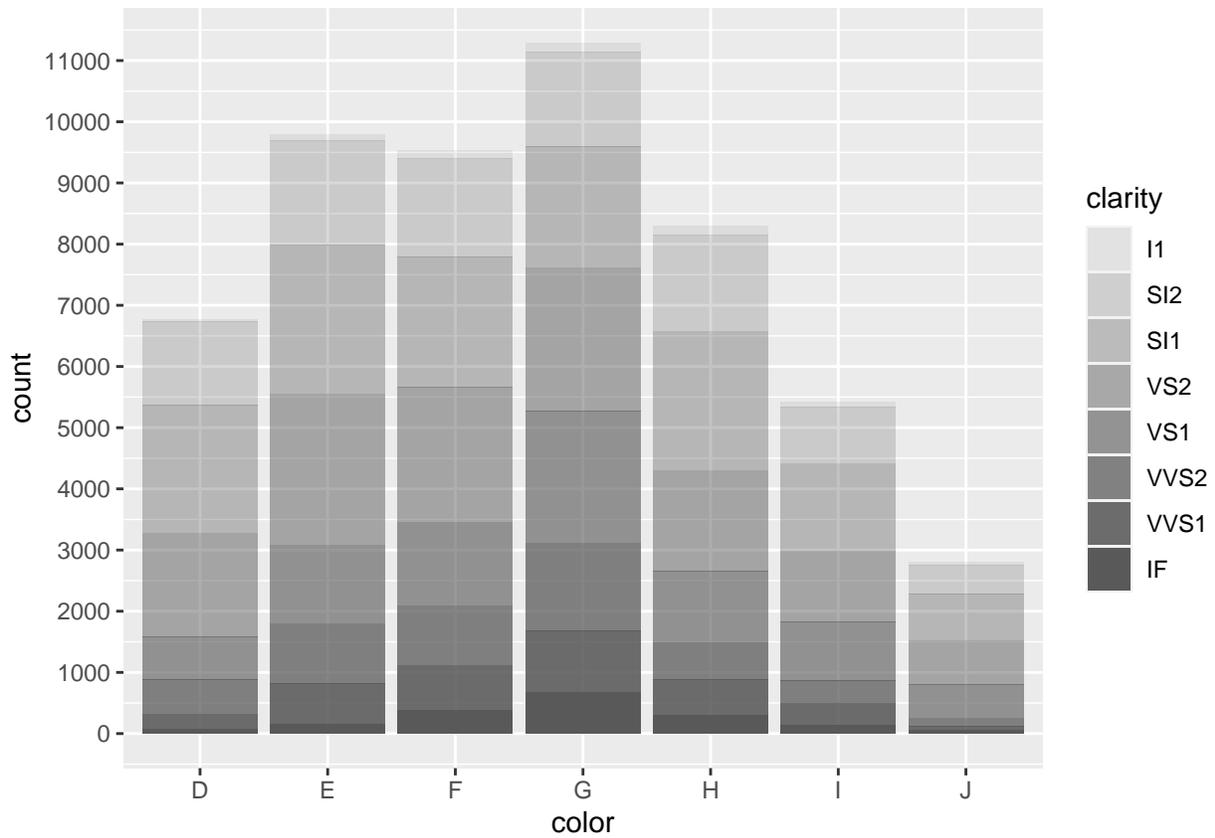
Very nice! As seen in lecture, the fill color could be based on a different variable than the one on the x axis. For instance, we can have a different fill color for diamonds based on the clarity:

```
diamonds %>%
  ggplot(aes(x=color)) +
  geom_bar(aes(fill=clarity)) +
  scale_y_continuous(n.breaks=13)
```



Fill and color are not the only aesthetic attributes you can set. Refer to the help menu for each geom to see what other aesthetic attributes you could change. For example, you can change the transparency of each bar in a bar plot using the `alpha` aesthetic:

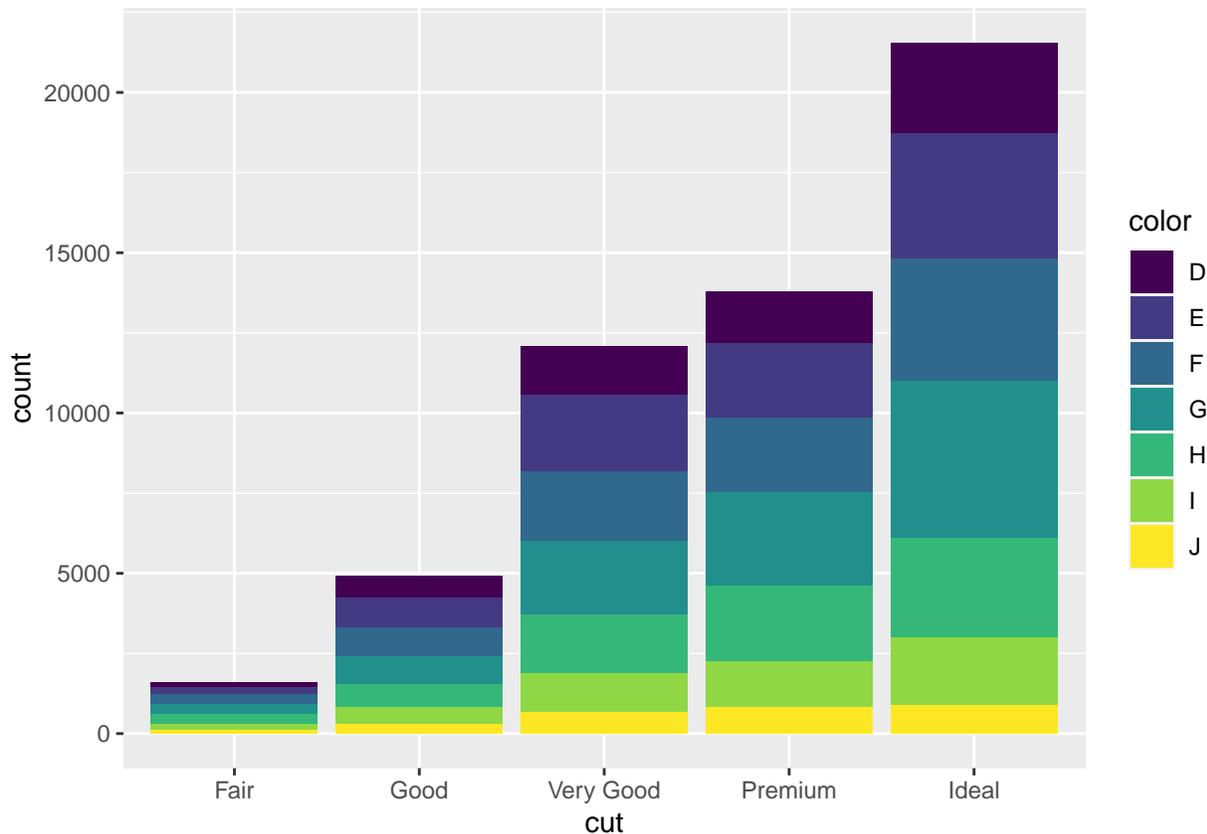
```
diamonds %>%
  ggplot(aes(x=color)) +
  geom_bar(aes(alpha=clarity)) +
  scale_y_continuous(n.breaks=13)
```



6. Visualize the distribution of diamond cuts, broken down by diamond color.

Answer:

```
diamonds %>%
  ggplot(aes(x=cut)) +
  geom_bar(aes(fill=color))
```



Color palettes

Colors are very important in visualizations, so it's worth spending a little time understanding how to customize them.

A color palette is an ordering of colors that R will use. If you don't specify the color palette, R will use a default sequence of colors that may not be what you want.

There are many, many different color palettes out there. We will only explore a few, to get the idea.

Two common sets of palettes that work well with `ggplot2` are `viridis` (requires `viridis` package) and `RColorBrewer`.

Viridis

Viridis palettes are specially designed to be both printer-friendly and color blindness-friendly.

You can use `scale_colour_viridis()` or `scale_fill_viridis()` with a call to `ggplot()` to impose a viridis palette. A colour is for points, lines, and shape outlines. A fill is for the interior of a shape.

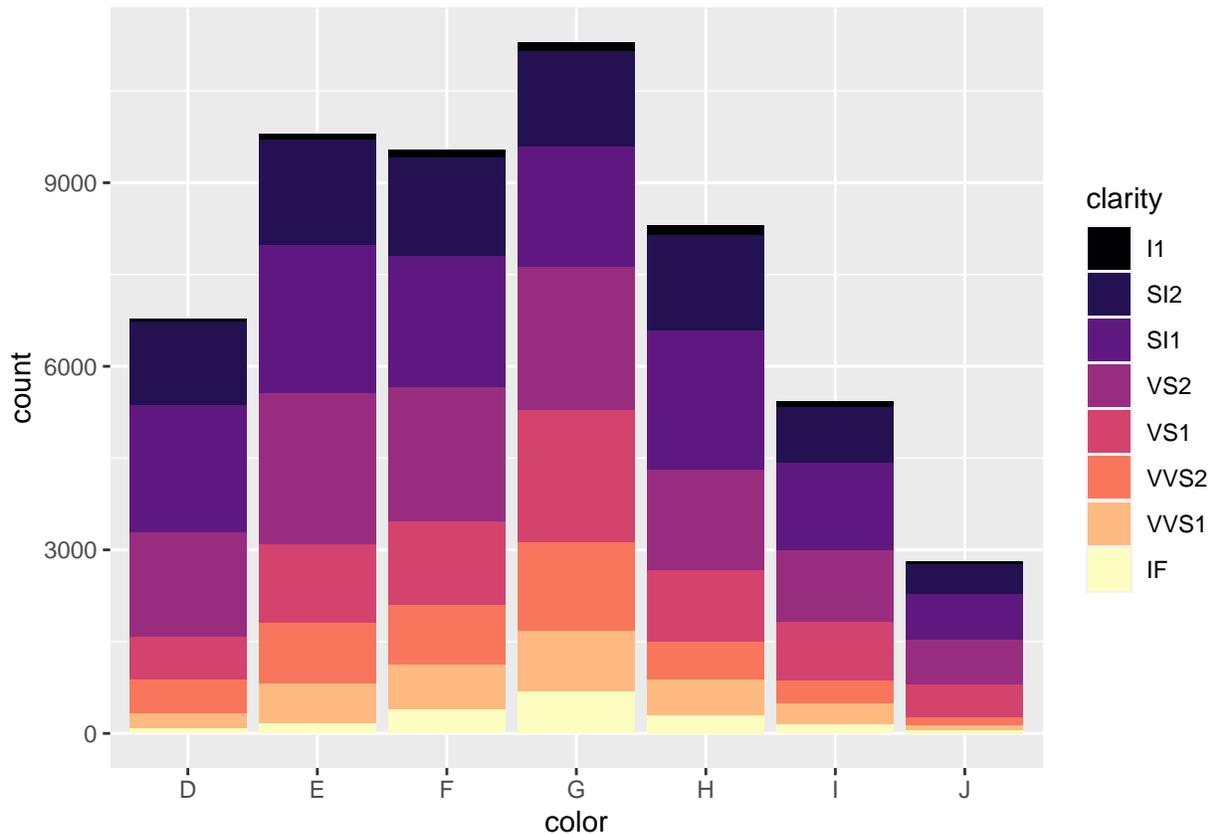
Note there are 8 different viridis palettes you can use, chosen by the "option" argument (see help menu). You also need to specify `discrete=TRUE` if your colour/fill is for a categorical variable. If you don't specify that, then the function will default to `discrete=FALSE`, and you will get a continuous color scale.

```
library(viridis)
```

```
## Loading required package: viridisLite
```

```
diamonds %>%
  ggplot(aes(x=color)) +
```

```
geom_bar(aes(fill=clarity)) +
scale_fill_viridis(discrete=TRUE, option="A")
```



Viridis color reference:

<https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html>

R Color Brewer

RColorBrewer is an older package with a lot more built-in palettes. These palettes for categorical variables, though can be made to work with quantitative data as well. You can use these palettes in the same way as with the viridis palettes, with `scale_colour_brewer()` or `scale_fill_brewer()`.

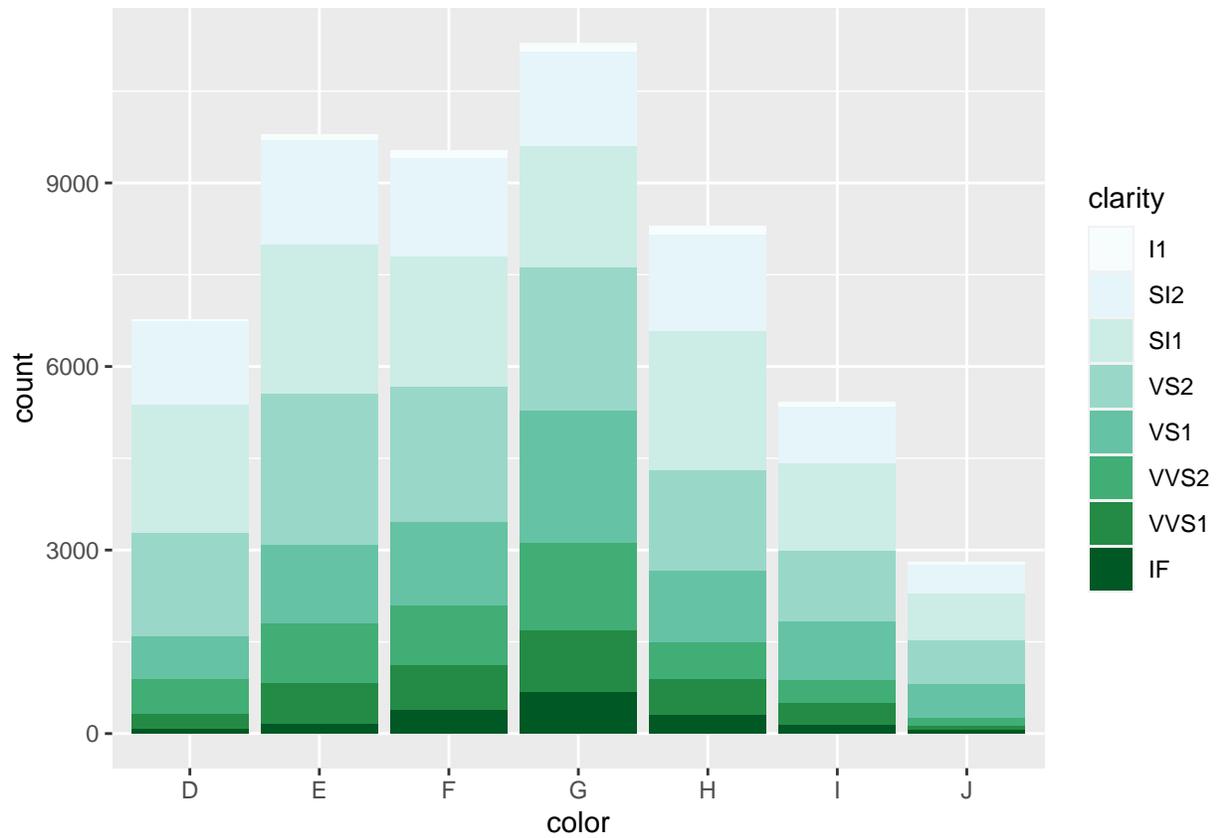
RColorBrewer has a few dozen palettes divided into 3 types: diverging, qualitative, and sequential. Choose the type that makes the most sense for your application.

A *diverging palette* has two contrasting colors at the ends, and gently grades to them in the middle. Example use case: The RdBu palette to show Democrats vs. Republicans.

Qualitative palette: A sequence of different colors that aren't related to each other. Example use case: Any non-ordered categorical variable.

Sequential palette: A sequence of different shades of the same/similar color, designed to vary in intensity. Example use case: An ordered categorical variable.

```
diamonds %>%
  ggplot(aes(x=color)) +
  geom_bar(aes(fill=clarity)) +
  scale_fill_brewer(type="seq", palette=2)
```



7. What type of palette did I use in this last plot? Why do you think I did so?

Answer: A sequential palette was used. This makes sense because `clarity` is an ordered categorical variable, going from the least clear IF to the clearest I1 (you can get this information from the help page for `diamonds`).

Useful RColorBrewer cheatsheet: <https://www.nceas.ucsb.edu/sites/default/files/2020-04/colorPaletteCheatSheet.pdf> (last page)