week6

Hakan Mehmetcik

Your goal during EDA is to develop an understanding of your data. The easiest way to do this is to use questions as tools to guide your investigation. When you ask a question, the question focuses your attention on a specific part of your dataset and helps you decide which graphs, models, or transformations to make.

Visual Tools for Data Analysis

In the realm of data analysis, the choice of visualization tools plays a pivotal role in effectively communicating insights derived from data. This table presents a curated collection of visual tools categorized by data type, each accompanied by a succinct description and its corresponding ggplot formula for implementation in R.

Whether dealing with categorical, numeric, time-series, geographic, or textual/nominal data, selecting the appropriate visualization technique can enhance understanding and facilitate interpretation. From bar plots illustrating frequency distributions to scatter plots revealing relationships between variables, these visualizations serve as indispensable aids in exploratory data analysis and presentation.

Explore this table to discern the most suitable visualization approach for your data type and analytical objectives, empowering you to convey compelling narratives and glean actionable insights from your datasets.

Data Type	Visual Tool	Description	ggplot Formula
Categorical	Bar Plot	Displays the frequency or count of each category.	`ggplot(data, aes(x = categorical_variable)) + geom_bar()`
Categorical	Pie Chart	Shows the proportions or percentages of each category.	`ggplot(data, aes(x = factor(1), fill = categorical_variable)) + geom_bar()`

Data Type	Visual Tool	Description	ggplot Formula
Categorical	Stacked Bar Plot	Compares the distribution of subcategories within categories.	`ggplot(data, aes(x = categorical_variable, fill = subcategory_variable)) + geom_bar()`
Numeric	Histogram	Represents the distribution of numeric values in intervals.	`ggplot(data, aes(x = numeric_variable)) + geom_histogram()`
Numeric	Box Plot (Box-and-Whisker)	Displays the summary statistics and identifies outliers.	`ggplot(data, aes(x = factor(1), y = numeric_variable)) + geom_boxplot()`
Numeric	Density Plot (Kernel Density)	Shows the probability density function of the data.	`ggplot(data, aes(x = numeric_variable)) + geom_density()`
Numeric	Scatter Plot	Depicts the relationship between two numeric variables.	`ggplot(data, aes(x = numeric_variable1, y = numeric_variable2)) + geom_point()`
Time Series	Time Series Plot	Visualizes data collected over time, such as stock prices.	`ggplot(data, aes(x = date_variable, y = numeric_variable)) + geom_line()`
Geographic	Scatter Plot on Maps	Maps data points with geographic coordinates.	`ggplot(data, aes(x = longitude_variable, y = latitude_variable)) + geom_point()`
Textual/Nominal	Word Frequency Plot	Shows the frequency of words in textual data.	`ggplot(data, aes(x = factor(1), fill = word_variable)) + geom_bar()`

Types of Analysis to Make:

EDA is fundamentally a creative process. And like most creative processes, the key to asking *quality* questions is to generate a large *quantity* of questions. It is difficult to ask revealing questions at the start of your analysis because you do not know what insights can be gleaned from your dataset. On the other hand, each new question that you ask will expose you to a

new aspect of your data and increase your chance of making a discovery. You can quickly drill down into the most interesting parts of your data—and develop a set of thought-provoking questions—if you follow up each question with a new question based on what you find.

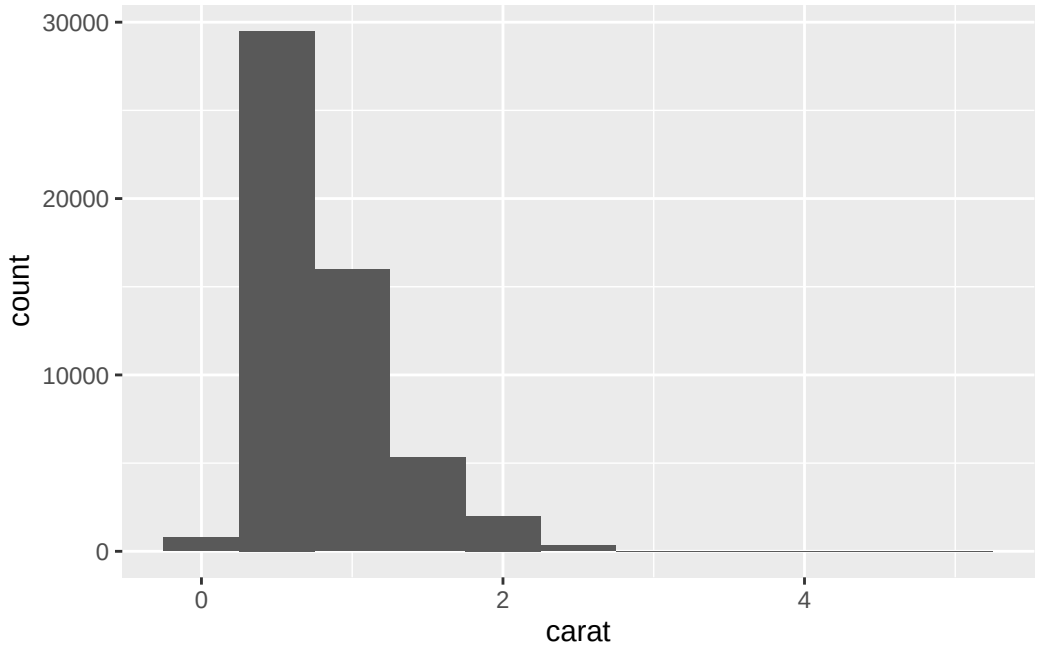
There is no rule about which questions you should ask to guide your research. However, two types of questions will always be useful for making discoveries within your data. You can loosely word these questions as:

1. What type of variation occurs within my variables?
2. What type of covariation occurs between my variables?

Variation

We'll start our exploration by visualizing the distribution of weights (`carat`) of ~54,000 diamonds from the `diamonds` dataset. Since `carat` is a numerical variable, we can use a histogram:

```
ggplot(data=diamonds)+
  (mapping=aes(x=carat))+
  (geom=geom_histogram(binwidth = 0.5)) # we used binwith 0.5 for conveninece
```



Now that you can visualize variation, what should you look for in your plots? And what type of follow-up questions should you ask? We've put together a list below of the most useful types of information that you will find in your graphs, along with some follow-up questions for each type of information. The key to asking good follow-up questions will be to rely on your curiosity (What do you want to learn more about?) as well as your skepticism (How could this be misleading?).

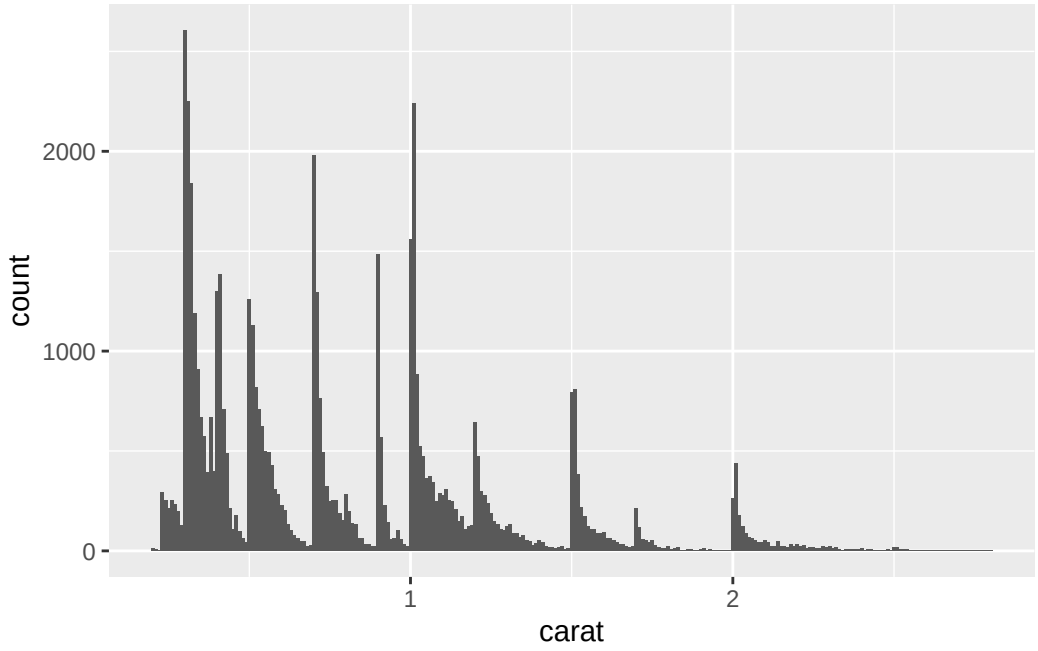
Typical Values

In both bar charts and histograms, tall bars show the common values of a variable, and shorter bars show less-common values. Places that do not have bars reveal values that were not seen in your data. To turn this information into useful questions, look for anything unexpected:

- Which values are the most common? Why?
- Which values are rare? Why? Does that match your expectations?
- Can you see any unusual patterns? What might explain them?

Let's take a look at the distribution of `carat` for smaller diamonds.

```
smaller <- diamonds |>
  filter(carat < 3)

ggplot(data=smaller, mapping = aes(x=carat)) +
  geom_histogram(binwidth=0.01)
```



This histogram suggests several interesting questions:

- Why are there more diamonds at whole carats and common fractions of carats?
- Why are there more diamonds slightly to the right of each peak than there are slightly to the left of each peak?

Visualizations can also reveal clusters, which suggest that subgroups exist in your data. To understand the subgroups, ask:

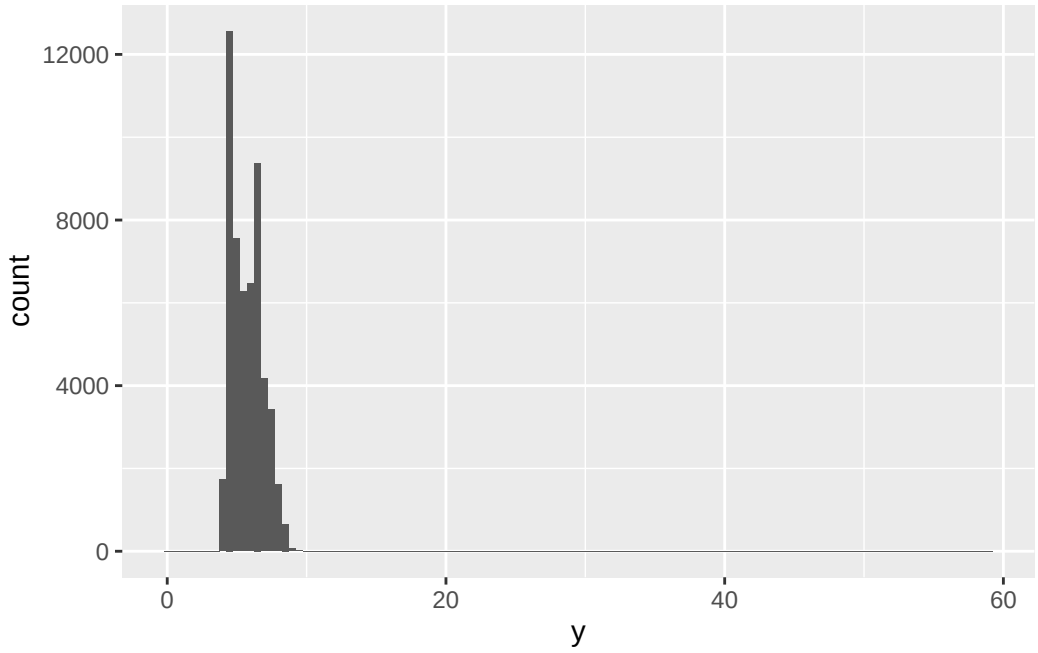
- How are the observations within each subgroup similar to each other?
- How are the observations in separate clusters different from each other?
- How can you explain or describe the clusters?
- Why might the appearance of clusters be misleading?

Some of these questions can be answered with the data while some will require domain expertise about the data. Many of them will prompt you to explore a relationship *between* variables, for example, to see if the values of one variable can explain the behavior of another variable. We'll get to that shortly.

Unusual Variables

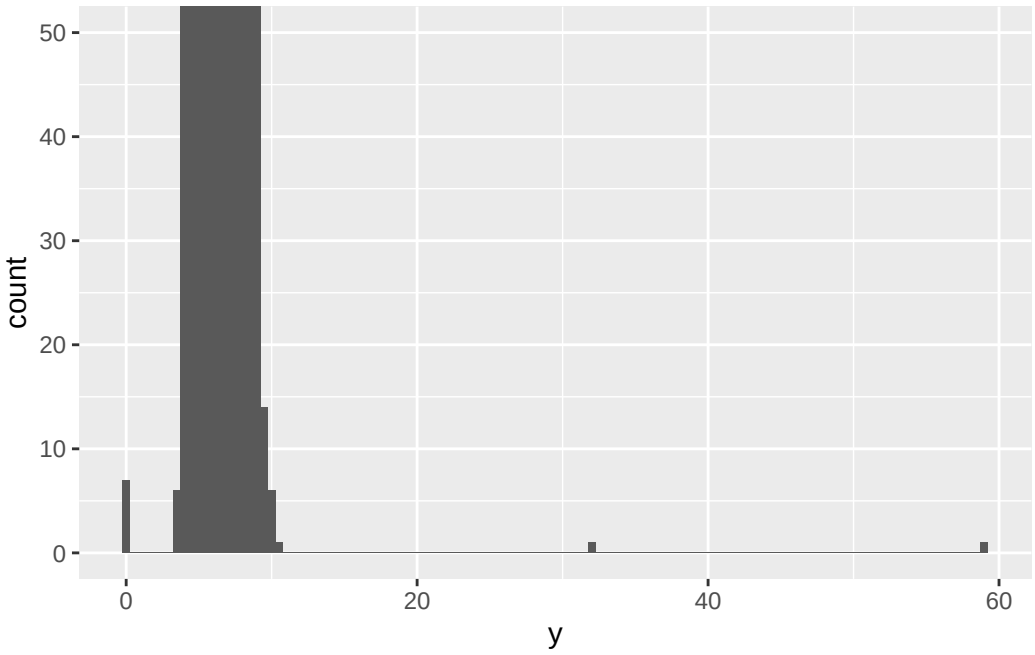
Outliers are observations that are unusual; data points that don't seem to fit the pattern. Sometimes outliers are data entry errors, sometimes they are simply values at the extremes that happened to be observed in this data collection, and other times they suggest important new discoveries. When you have a lot of data, outliers are sometimes difficult to see in a histogram. For example, take the distribution of the `y` variable from the diamonds dataset. The only evidence of outliers is the unusually wide limits on the x-axis.

```
ggplot(data=diamonds) +
  (mapping=aes(x = y)) +
  (geom=geom_histogram(binwidth = 0.5))
```



There are so many observations in the common bins that the rare bins are very short, making it very difficult to see them (although maybe if you stare intently at 0 you'll spot something). To make it easy to see the unusual values, we need to zoom to small values of the y-axis with `coord_cartesian()`:

```
ggplot(data=diamonds) +
  (mapping=aes(x = y)) +
  (geom=geom_histogram(binwidth = 0.5))+
  (geom= coord_cartesian(ylim=c(0,50)))
```



This allows us to see that there are three unusual values: 0, ~30, and ~60. We pluck them out with dplyr:

```
unusual <- diamonds |>
  filter(y < 3 | y > 20) |>
  select(price, x, y, z) |>
  arrange(y)
unusual
```

price	x	y	z
5139	0.00	0.0	0.00
6381	0.00	0.0	0.00
12800	0.00	0.0	0.00
15686	0.00	0.0	0.00
18034	0.00	0.0	0.00
2130	0.00	0.0	0.00
2130	0.00	0.0	0.00
2075	5.15	31.8	5.12
12210	8.09	58.9	8.06

The `y` variable measures one of the three dimensions of these diamonds, in mm. We know

that diamonds can't have a width of 0mm, so these values must be incorrect. By doing EDA, we have discovered missing data that was coded as 0, which we never would have found by simply searching for `NA`s. Going forward we might choose to re-code these values as `NA`s in order to prevent misleading calculations. We might also suspect that measurements of 32mm and 59mm are implausible: those diamonds are over an inch long, but don't cost hundreds of thousands of dollars!

It's good practice to repeat your analysis with and without the outliers. If they have minimal effect on the results, and you can't figure out why they're there, it's reasonable to omit them, and move on. However, if they have a substantial effect on your results, you shouldn't drop them without justification. You'll need to figure out what caused them (e.g., a data entry error) and disclose that you removed them in your write-up.

If you've encountered unusual values in your dataset, and simply want to move on to the rest of your analysis, you have two options.

1. Drop the entire row with the strange values:

   ```
   diamonds2 <- diamonds |>
     filter(between(y, 3, 20))
   ```

   We don't recommend this option because one invalid value doesn't imply that all the other values for that observation are also invalid. Additionally, if you have low quality data, by the time that you've applied this approach to every variable you might find that you don't have any data left!

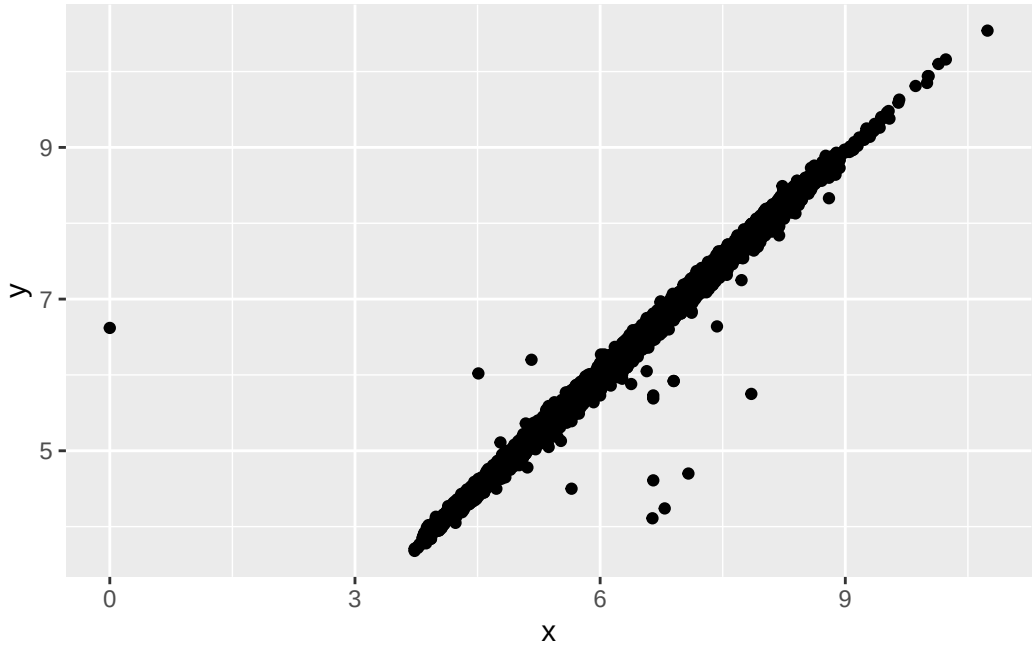
2. Instead, we recommend replacing the unusual values with missing values. The easiest way to do this is to use `mutate()` to replace the variable with a modified copy. You can use the `if_else()` function to replace unusual values with `NA`:

   ```
   diamonds2 <- diamonds |>
     mutate(y = if_else(y < 3 | y > 20, NA, y))
   ```

It's not obvious where you should plot missing values, so ggplot2 doesn't include them in the plot, but it does warn that they've been removed:

```
ggplot(diamonds2, aes(x = x, y = y)) +
  geom_point()
```

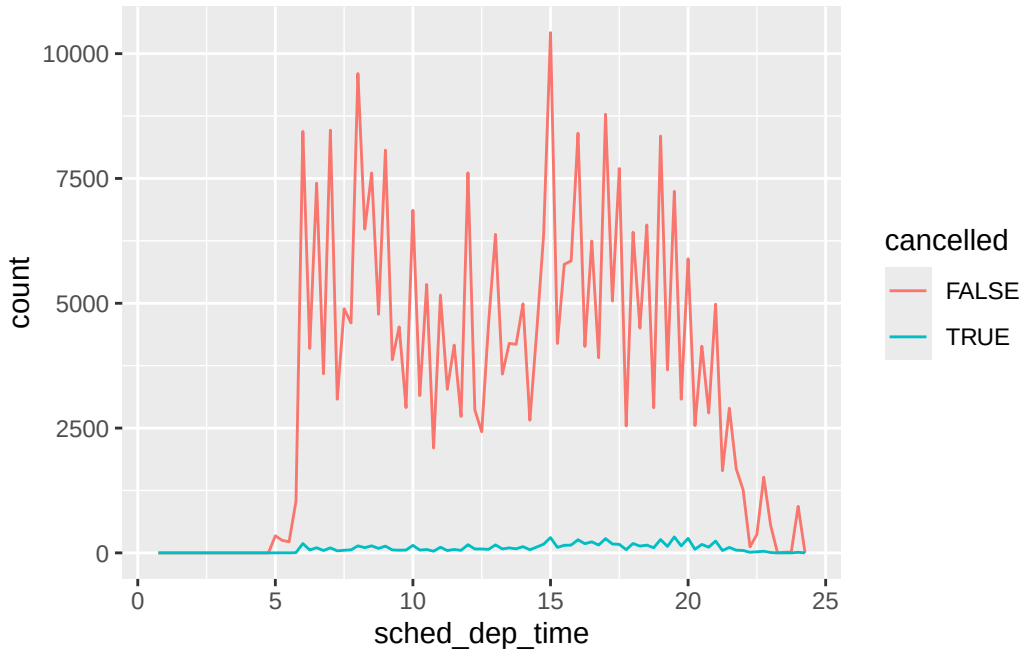
```
Warning: Removed 9 rows containing missing values or values outside the scale range
(`geom_point()`).
```




```
# To suppress that warning, set na.rm = TRUE in geom_point()
```

Other times you want to understand what makes observations with missing values different to observations with recorded values. For example, in `nycflights13::flights`[1], missing values in the `dep_time` variable indicate that the flight was cancelled. So you might want to compare the scheduled departure times for cancelled and non-cancelled times. You can do this by making a new variable, using `is.na()` to check if `dep_time` is missing.

```
nycflights13::flights |>
  mutate(
    cancelled = is.na(dep_time),
    sched_hour = sched_dep_time %/% 100,
    sched_min = sched_dep_time %% 100,
    sched_dep_time = sched_hour + (sched_min / 60)
  ) |>
  ggplot(aes(x = sched_dep_time)) +
  geom_freqpoly(aes(color = cancelled), binwidth = 1/4)
```



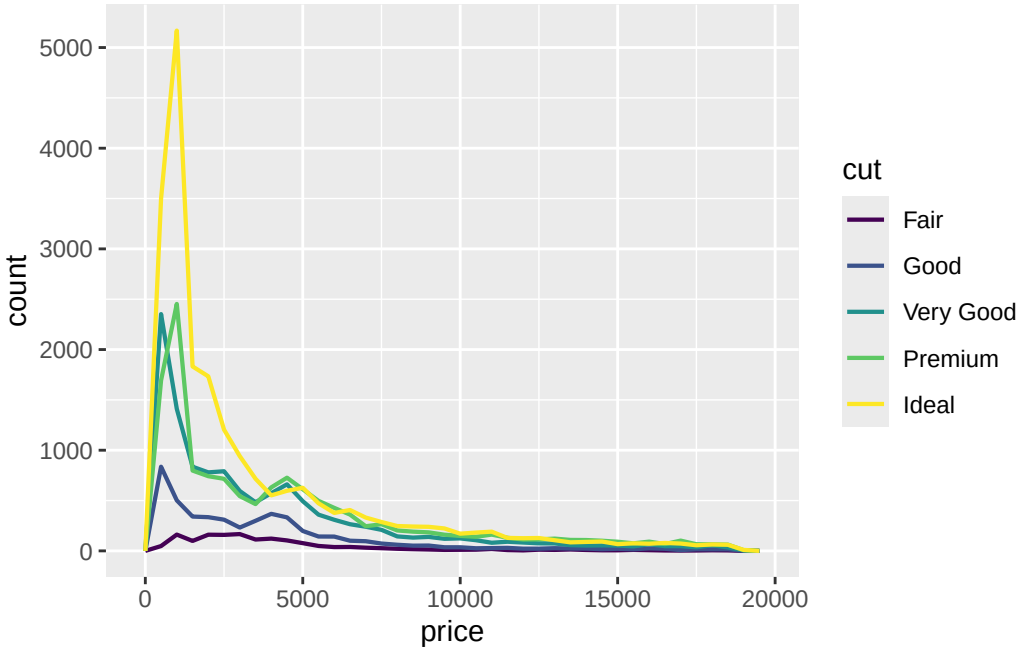
Covariation

If variation describes the behavior *within* a variable, covariation describes the behavior *between* variables. **Covariation** is the tendency for the values of two or more variables to vary together in a related way. The best way to spot covariation is to visualize the relationship between two or more variables.

A categorical and a numerical variable

For example, let's explore how the price of a diamond varies with its quality (measured by `cut`) using `geom_freqpoly()`:

```
ggplot(diamonds, aes(x = price)) +
  geom_freqpoly(aes(color = cut), binwidth = 500, linewidth = 0.75)
```

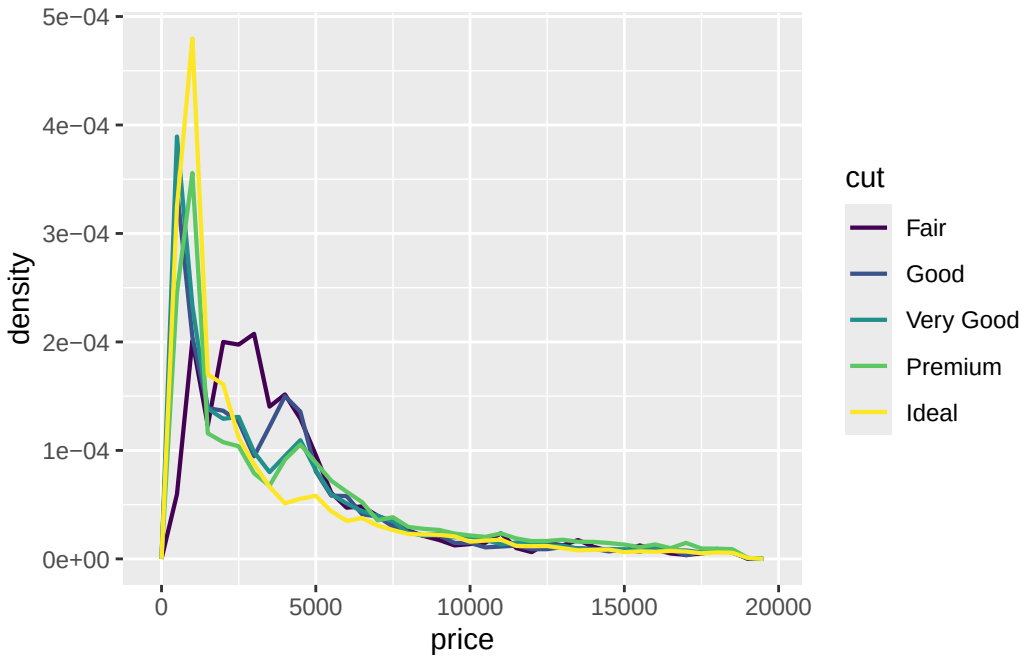


Note that ggplot2 uses an ordered color scale for `cut` because it's defined as an ordered factor variable in the data.

The default appearance of `geom_freqpoly()` is not that useful here because the height, determined by the overall count, differs so much across `cut`s, making it hard to see the differences in the shapes of their distributions.

To make the comparison easier we need to swap what is displayed on the y-axis. Instead of displaying count, we'll display the **density**, which is the count standardized so that the area under each frequency polygon is one.

```
ggplot(diamonds, aes(x = price, y = after_stat(density))) +
  geom_freqpoly(aes(color = cut), binwidth = 500, linewidth = 0.75)
```

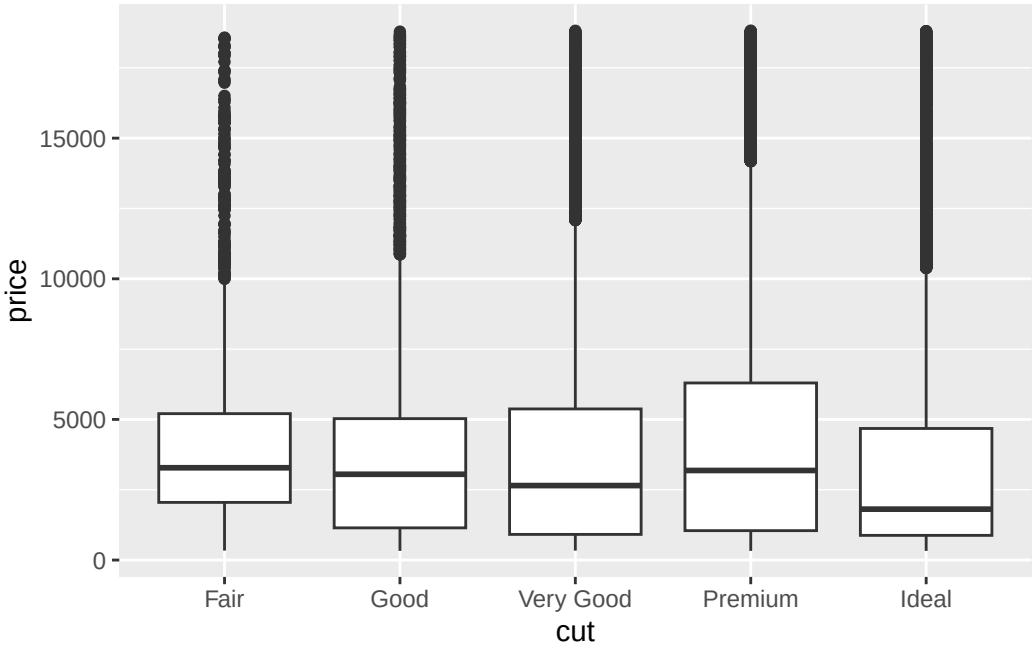


Note that we're mapping the density to `y`, but since `density` is not a variable in the `diamonds` dataset, we need to first calculate it. We use the `after_stat()` function to do so.

There's something rather surprising about this plot - it appears that fair diamonds (the lowest quality) have the highest average price! But maybe that's because frequency polygons are a little hard to interpret - there's a lot going on in this plot.

A visually simpler plot for exploring this relationship is using side-by-side boxplots.

```
ggplot(diamonds, aes(x = cut, y = price)) +
  geom_boxplot()
```

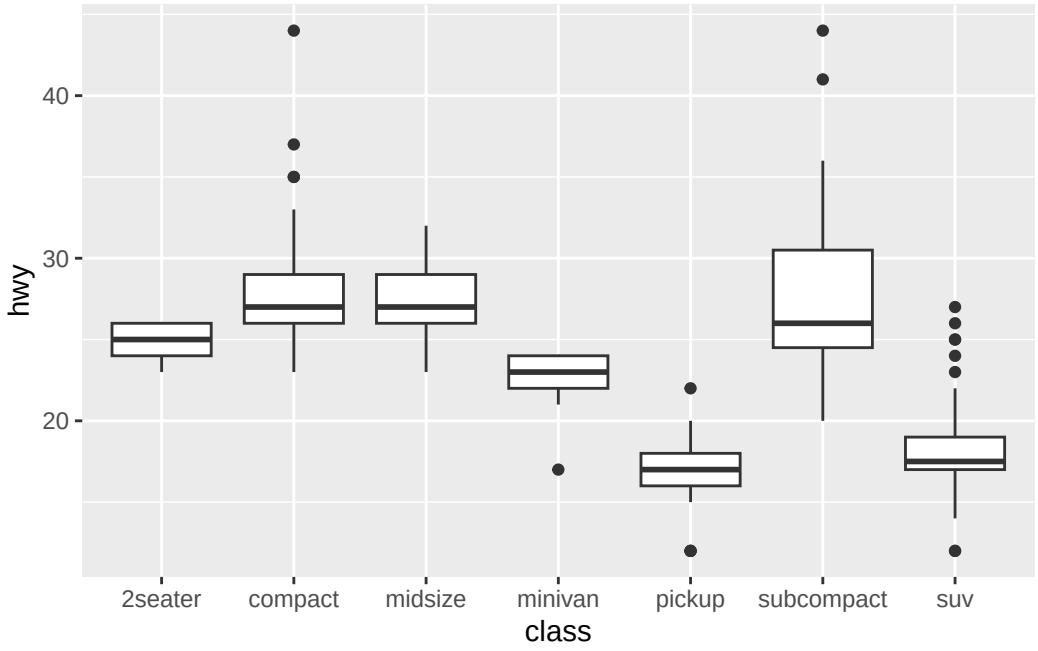


We see much less information about the distribution, but the boxplots are much more compact so we can more easily compare them (and fit more on one plot). It supports the counter-intuitive finding that better quality diamonds are typically cheaper! In the exercises, you'll be challenged to figure out why.

`cut` is an ordered factor: fair is worse than good, which is worse than very good and so on. Many categorical variables don't have such an intrinsic order, so you might want to reorder them to make a more informative display. One way to do that is with `fct_reorder()`.

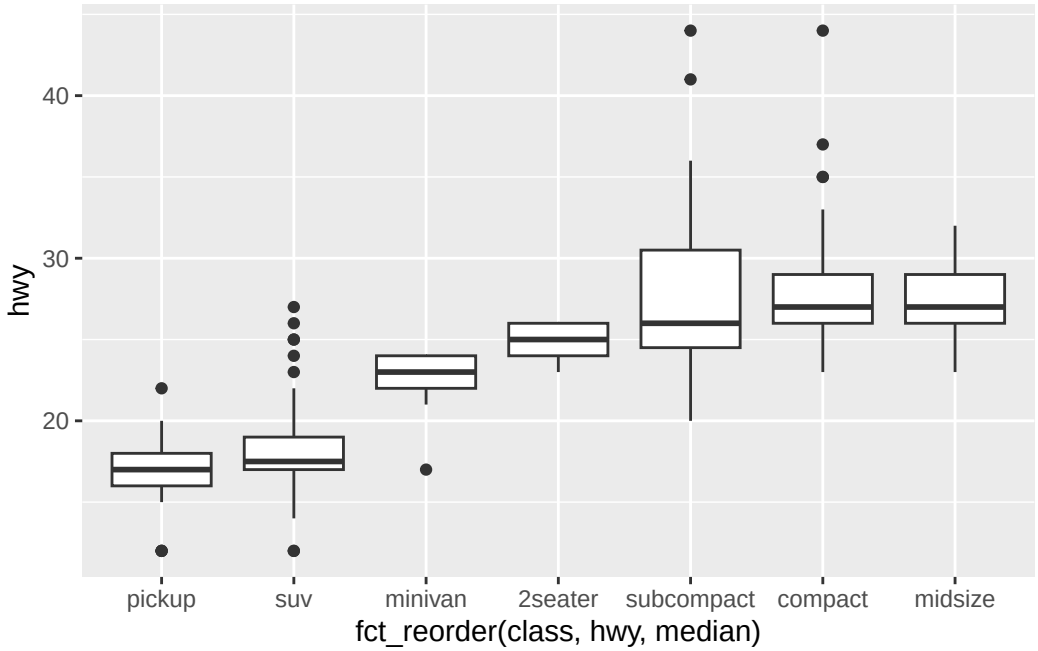
For example, take the `class` variable in the `mpg` dataset. You might be interested to know how highway mileage varies across classes:

```
ggplot(mpg, aes(x = class, y = hwy)) +
  geom_boxplot()
```



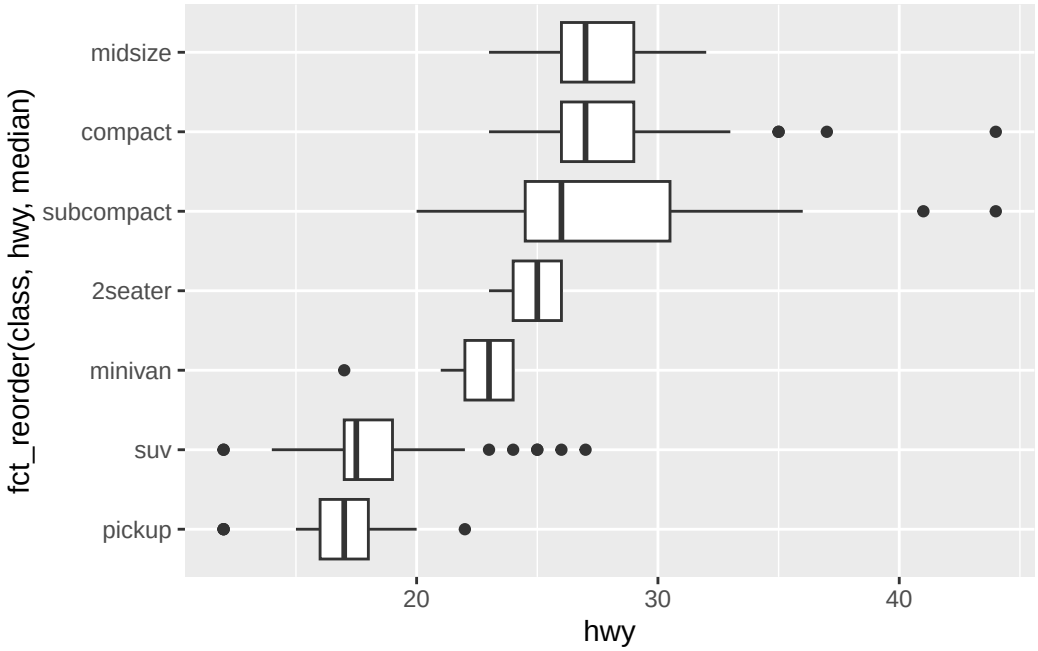
To make the trend easier to see, we can reorder `class` based on the median value of `hwy`:

```
ggplot(mpg, aes(x = fct_reorder(class, hwy, median), y = hwy)) +
  geom_boxplot()
```



If you have long variable names, `geom_boxplot()` will work better if you flip it 90°. You can do that by exchanging the x and y aesthetic mappings.

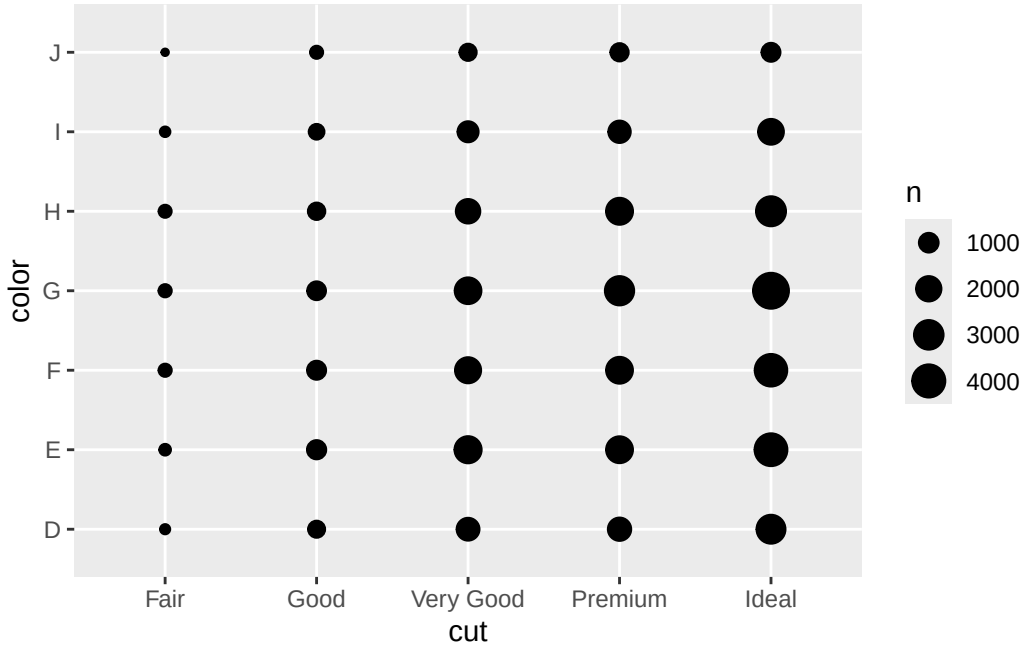
```
ggplot(mpg, aes(x = hwy, y = fct_reorder(class, hwy, median))) +
  geom_boxplot()
```



Two categorical variables

To visualize the covariation between categorical variables, you'll need to count the number of observations for each combination of levels of these categorical variables. One way to do that is to rely on the built-in `geom_count()`:

```
ggplot(diamonds, aes(x = cut, y = color)) +
  geom_count()
```



The size of each circle in the plot displays how many observations occurred at each combination of values. Covariation will appear as a strong correlation between specific x values and specific y values.

Another approach for exploring the relationship between these variables is computing the counts with dplyr:

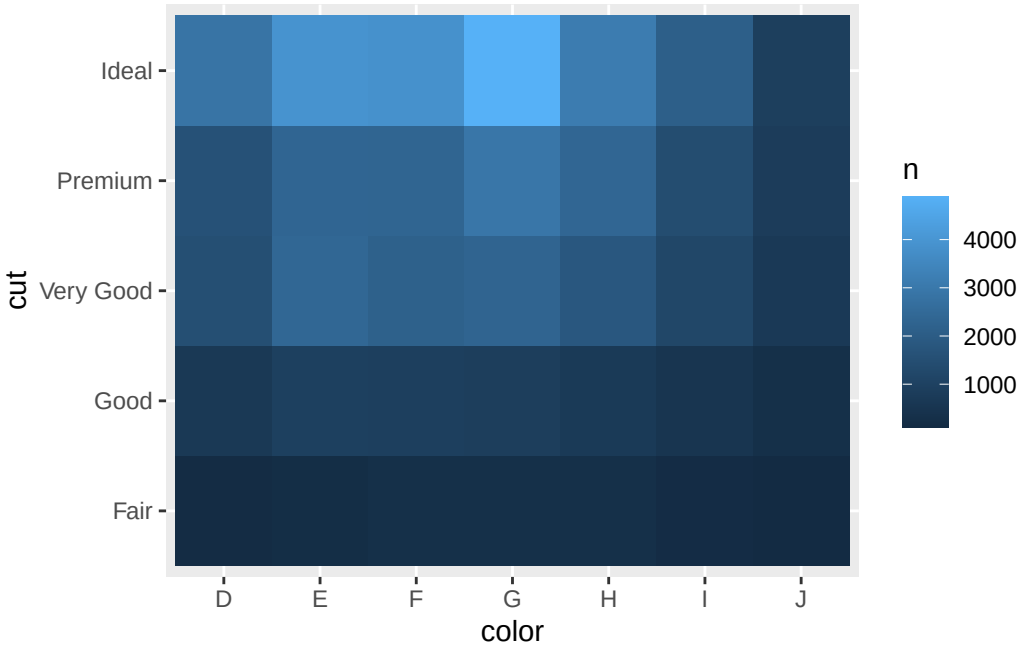
```
diamonds |>
  count(color, cut)
```

color	cut	n
D	Fair	163
D	Good	662
D	Very Good	1513
D	Premium	1603
D	Ideal	2834
E	Fair	224
E	Good	933
E	Very Good	2400
E	Premium	2337
E	Ideal	3903
F	Fair	312

color	cut	n
F	Good	909
F	Very Good	2164
F	Premium	2331
F	Ideal	3826
G	Fair	314
G	Good	871
G	Very Good	2299
G	Premium	2924
G	Ideal	4884
H	Fair	303
H	Good	702
H	Very Good	1824
H	Premium	2360
H	Ideal	3115
I	Fair	175
I	Good	522
I	Very Good	1204
I	Premium	1428
I	Ideal	2093
J	Fair	119
J	Good	307
J	Very Good	678
J	Premium	808
J	Ideal	896

Then visualize with `geom_tile()` and the fill aesthetic:

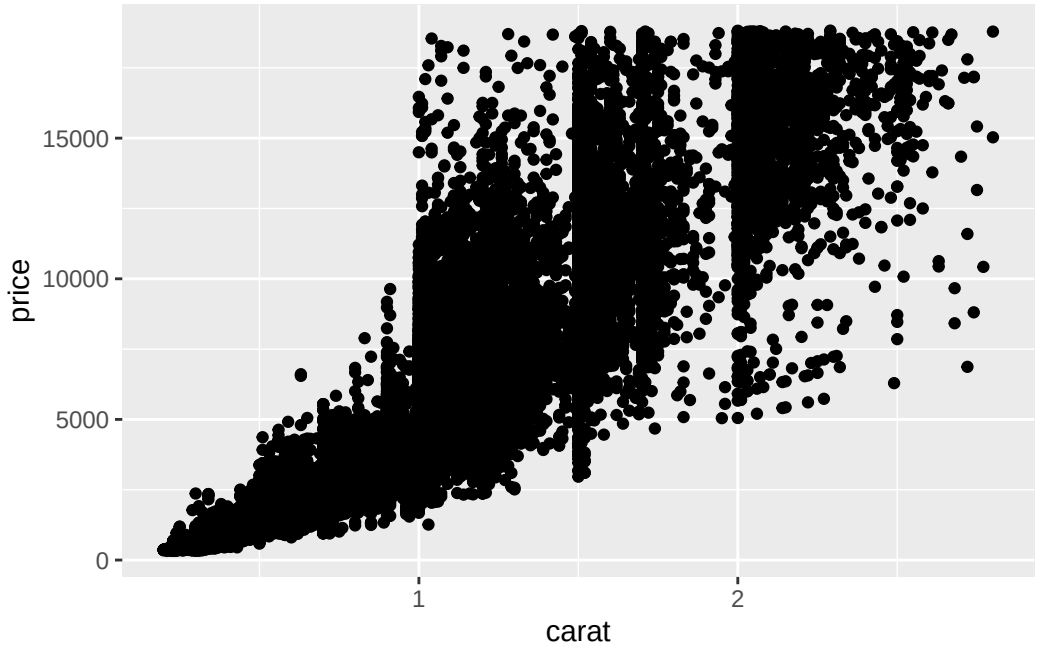
```
diamonds |>
  count(color, cut) |>
  ggplot(aes(x = color, y = cut)) +
  geom_tile(aes(fill = n))
```



Two numerical variables

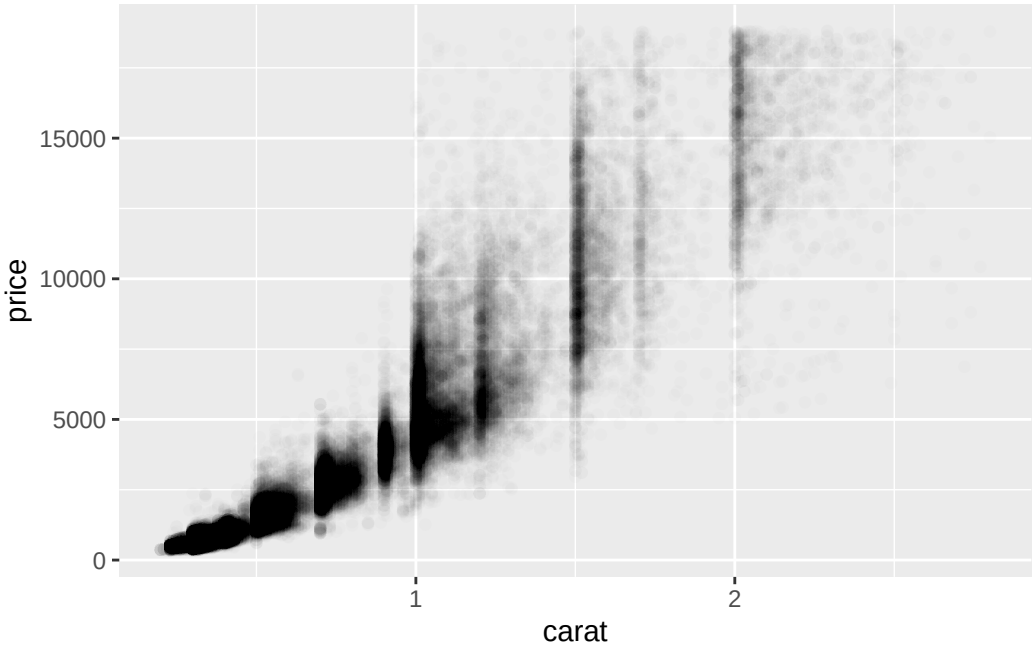
You've already seen one great way to visualize the covariation between two numerical variables: draw a scatterplot with `geom_point()`. You can see covariation as a pattern in the points. For example, you can see a positive relationship between the carat size and price of a diamond: diamonds with more carats have a higher price. The relationship is exponential.

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



Scatterplots become less useful as the size of your dataset grows, because points begin to overplot, and pile up into areas of uniform black, making it hard to judge differences in the density of the data across the 2-dimensional space as well as making it hard to spot the trend. You've already seen one way to fix the problem: using the `alpha` aesthetic to add transparency.

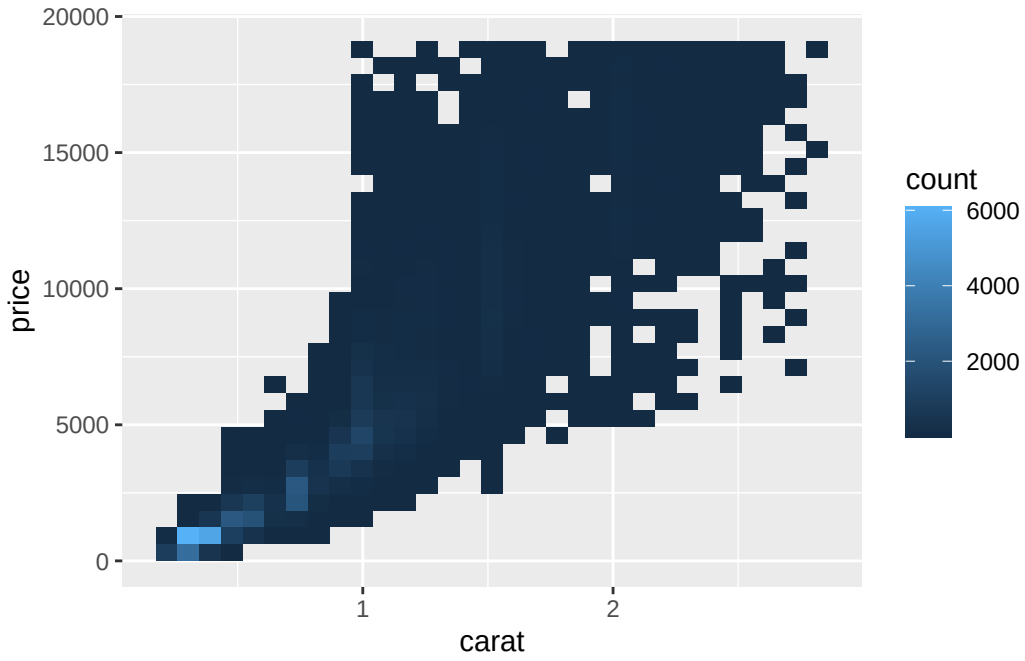
```
ggplot(smaller, aes(x = carat, y = price)) +
  geom_point(alpha = 1 / 100)
```



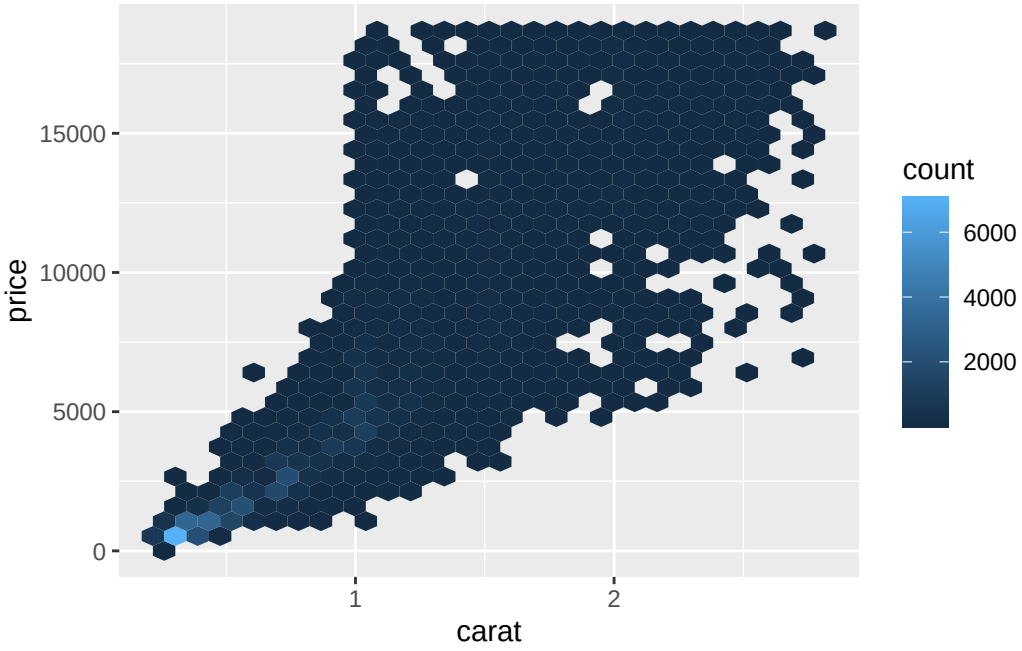
But using transparency can be challenging for very large datasets. Another solution is to use bin. Previously you used `geom_histogram()` and `geom_freqpoly()` to bin in one dimension. Now you'll learn how to use `geom_bin2d()` and `geom_hex()` to bin in two dimensions.

`geom_bin2d()` and `geom_hex()` divide the coordinate plane into 2d bins and then use a fill color to display how many points fall into each bin. `geom_bin2d()` creates rectangular bins. `geom_hex()` creates hexagonal bins. You will need to install the hexbin package to use `geom_hex()`.

```
ggplot(smaller, aes(x = carat, y = price)) +
  geom_bin2d()
```

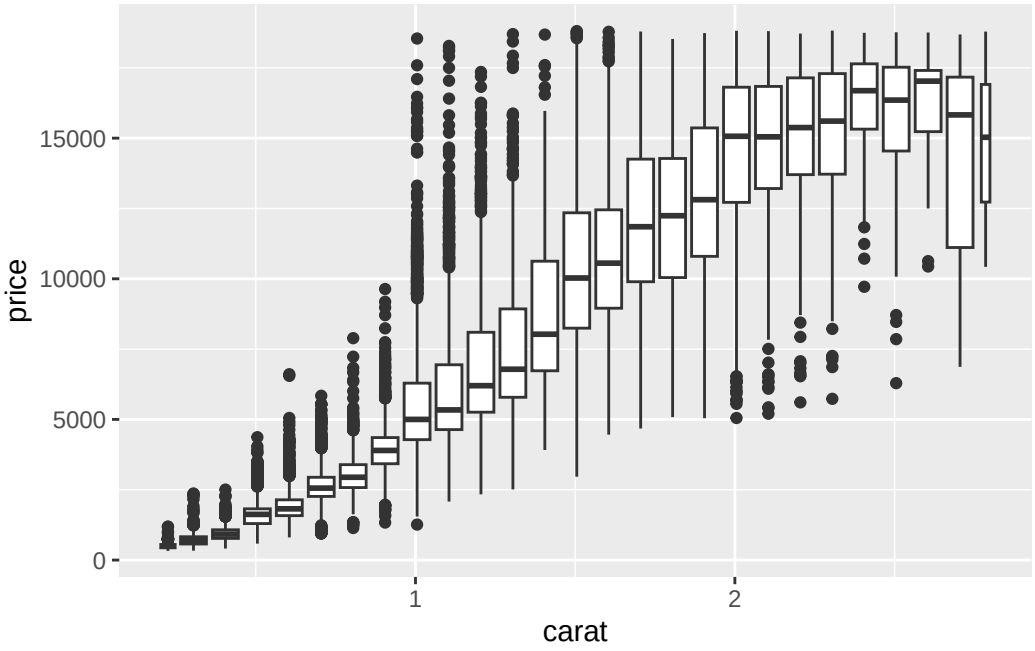



```
# install.packages("hexbin")
library(hexbin)
ggplot(smaller, aes(x = carat, y = price)) +
  geom_hex()
```



Another option is to bin one continuous variable so it acts like a categorical variable. Then you can use one of the techniques for visualizing the combination of a categorical and a continuous variable that you learned about. For example, you could bin `carat` and then for each group, display a boxplot:

```
ggplot(smaller, aes(x = carat, y = price)) +
  geom_boxplot(aes(group = cut_width(carat, 0.1)))
```



Patterns and models

If a systematic relationship exists between two variables it will appear as a pattern in the data. If you spot a pattern, ask yourself:

- Could this pattern be due to coincidence (i.e. random chance)?
- How can you describe the relationship implied by the pattern?
- How strong is the relationship implied by the pattern?
- What other variables might affect the relationship?
- Does the relationship change if you look at individual subgroups of the data?

Patterns in your data provide clues about relationships, i.e., they reveal covariation. If you think of variation as a phenomenon that creates uncertainty, covariation is a phenomenon that reduces it. If two variables covary, you can use the values of one variable to make better predictions about the values of the second. If the covariation is due to a causal relationship (a special case), then you can use the value of one variable to control the value of the second.

Models are a tool for extracting patterns out of data. For example, consider the diamonds data. It's hard to understand the relationship between cut and price, because cut and carat, and carat and price are tightly related. It's possible to use a model to remove the very strong relationship between price and carat so we can explore the subtleties that remain. The

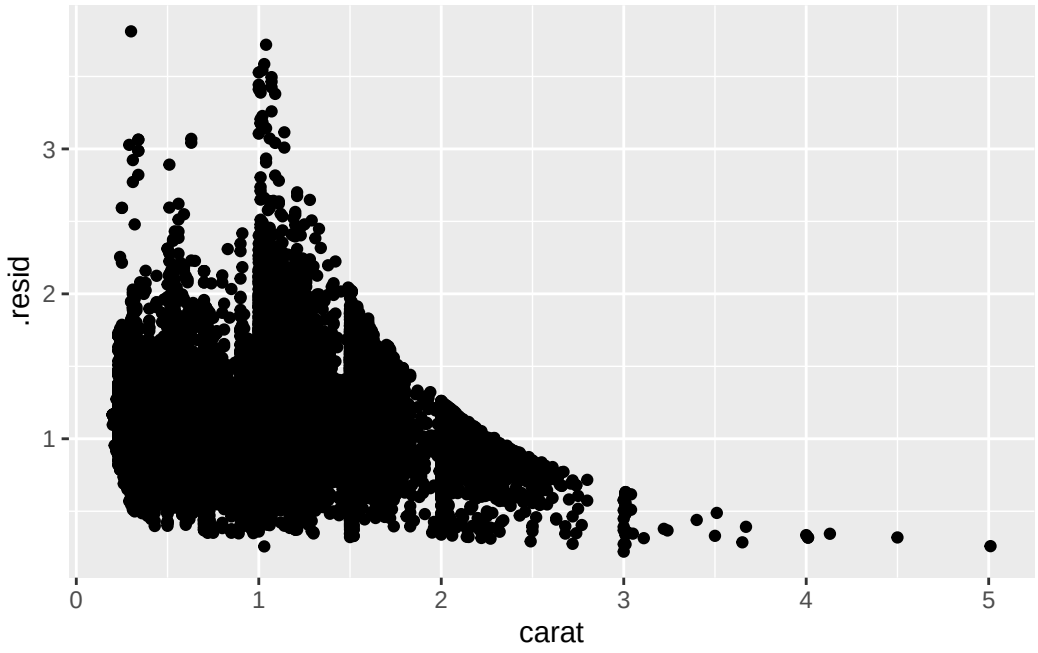
following code fits a model that predicts `price` from `carat` and then computes the residuals (the difference between the predicted value and the actual value). The residuals give us a view of the price of the diamond, once the effect of carat has been removed. Note that instead of using the raw values of `price` and `carat`, we log transform them first, and fit a model to the log-transformed values. Then, we exponentiate the residuals to put them back in the scale of raw prices.

```
library(tidymodels)
```

```
-- Attaching packages --------------------------------------- tidymodels 1.2.0 --

v broom        1.0.5      v rsample      1.2.1
v dials        1.2.1      v tune         1.2.0
v infer        1.0.7      v workflows    1.1.4
v modeldata    1.3.0      v workflowsets 1.1.0
v parsnip      1.2.1      v yardstick    1.3.1
v recipes      1.0.10

-- Conflicts ------------------------------------------ tidymodels_conflicts() --
x scales::discard() masks purrr::discard()
x dplyr::filter()   masks stats::filter()
x recipes::fixed()  masks stringr::fixed()
x dplyr::lag()      masks stats::lag()
x yardstick::spec() masks readr::spec()
x recipes::step()   masks stats::step()
* Dig deeper into tidy modeling with R at https://www.tmwr.org
```

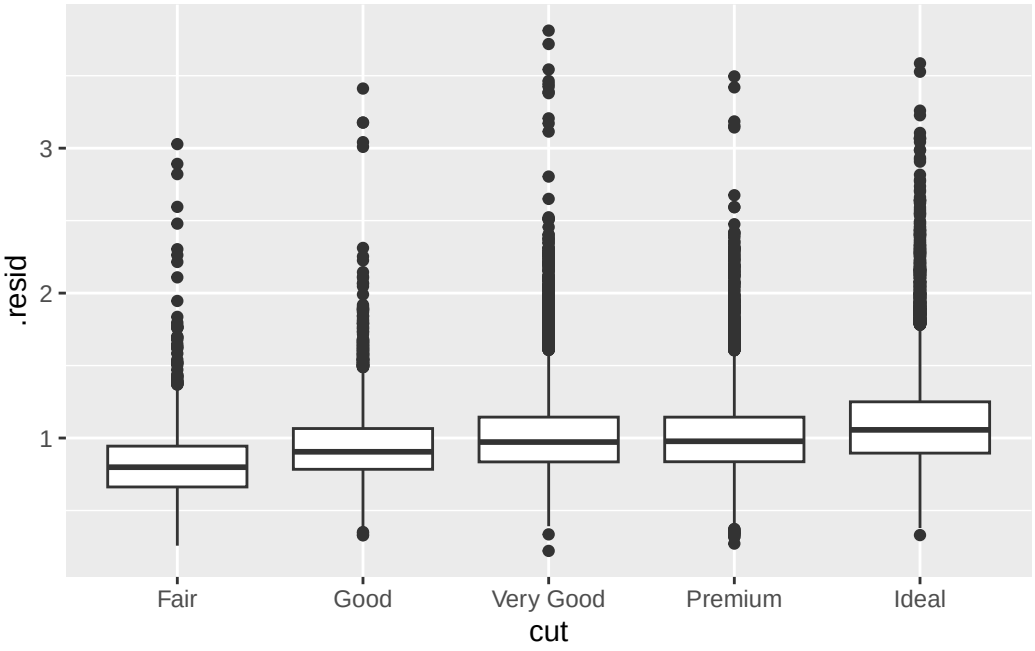
```
diamonds <- diamonds |>
  mutate(
    log_price = log(price),
    log_carat = log(carat)
  )

diamonds_fit <- linear_reg() |>
  fit(log_price ~ log_carat, data = diamonds)

diamonds_aug <- augment(diamonds_fit, new_data = diamonds) |>
  mutate(.resid = exp(.resid))

ggplot(diamonds_aug, aes(x = carat, y = .resid)) +
```

```
  geom_point()
```



Once you've removed the strong relationship between carat and price, you can see what you expect in the relationship between cut and price: relative to their size, better quality diamonds are more expensive.

```
ggplot(diamonds_aug, aes(x = cut, y = .resid)) +
  geom_boxplot()
```


.resid
3
2
1
Fair
Good
Very Good
Premium
Ideal
cut