Notes on Ch 10: EDA

N_Lim

2025-07-03

- EDA is an iterative ccycle where you:

1. Generate questions about your data.
2. Search for answers by visualizing, transforming, and modelling your data.
3. Use what you learned to refine your questions and/or generate new questions.

- EDA is not a formal process.
- It has no strict set of rules.

Prerequisites

```
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   3.5.2     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- 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
```

Goal during EDA: - Develop an understanding of data. - Use questions as tools to guide your investigation.

Questions that will always be useful for discovering or uncovering something in your data: 1. What type of variation occurs within my variables? 2. What type of covariation occurs between my variables?

```
glimpse(diamonds)
```

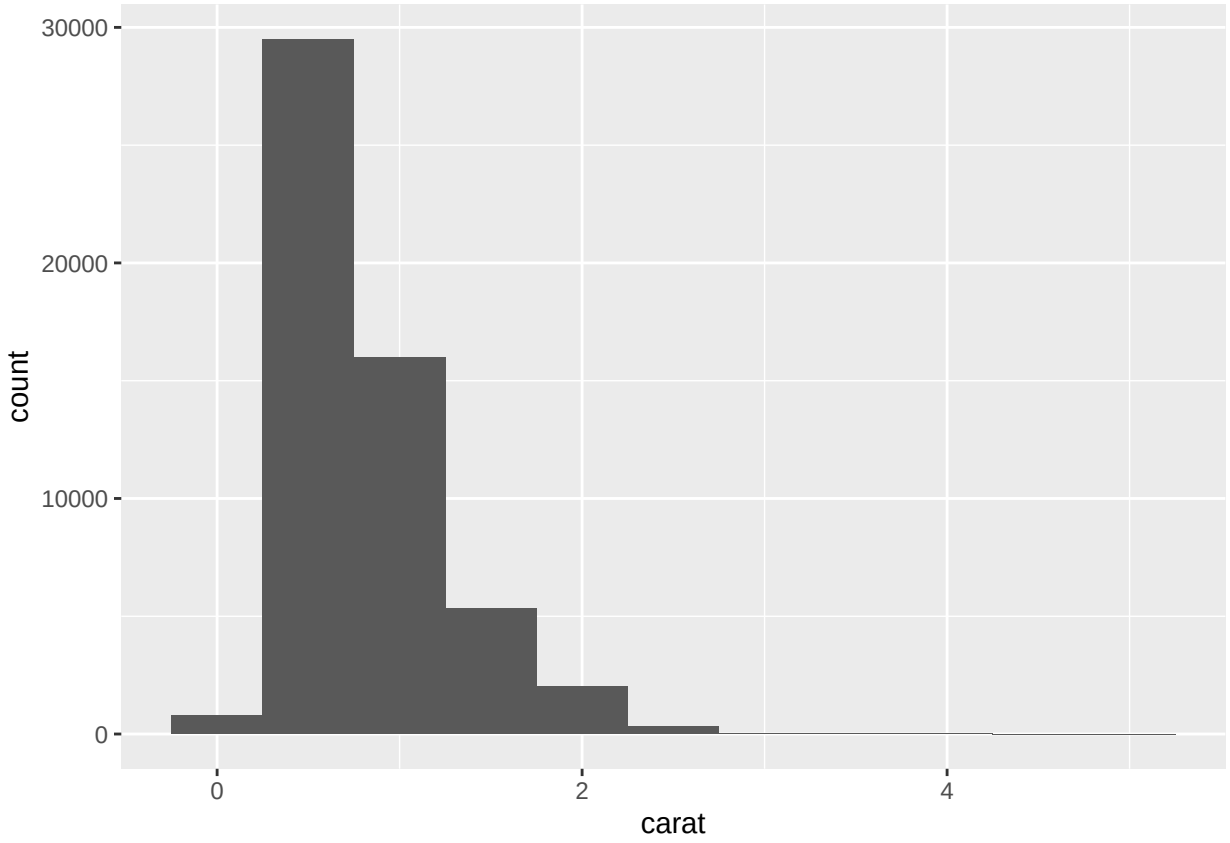
```
## Rows: 53,940
## Columns: 10
## $ carat   <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22, 0.23, 0.~
## $ cut     <ord> Ideal, Premium, Good, Premium, Good, Very Good, Very Good, Ver~
## $ color   <ord> E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E, I, J, J, J, I,~
## $ clarity <ord> SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1, SI1, VS1, ~
## $ depth   <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1, 59.4, 64~
## $ table   <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54, 62, 58~
## $ price   <int> 326, 326, 327, 334, 335, 336, 336, 337, 337, 338, 339, 340, 34~
## $ x       <dbl> 3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07, 3.87, 4.00, 4.~
## $ y       <dbl> 3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11, 3.78, 4.05, 4.~
## $ z       <dbl> 2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53, 2.49, 2.39, 2.~
```

Variation

- Tendency of the values of a variable to change from measurement to measurement.

Example: Visualizing the distribution of weights (`carat`) of about 54 000 diamonds from the `diamonds` dataset:

```
ggplot(diamonds, aes(x = carat)) +
  geom_histogram(binwidth = 0.5)
```

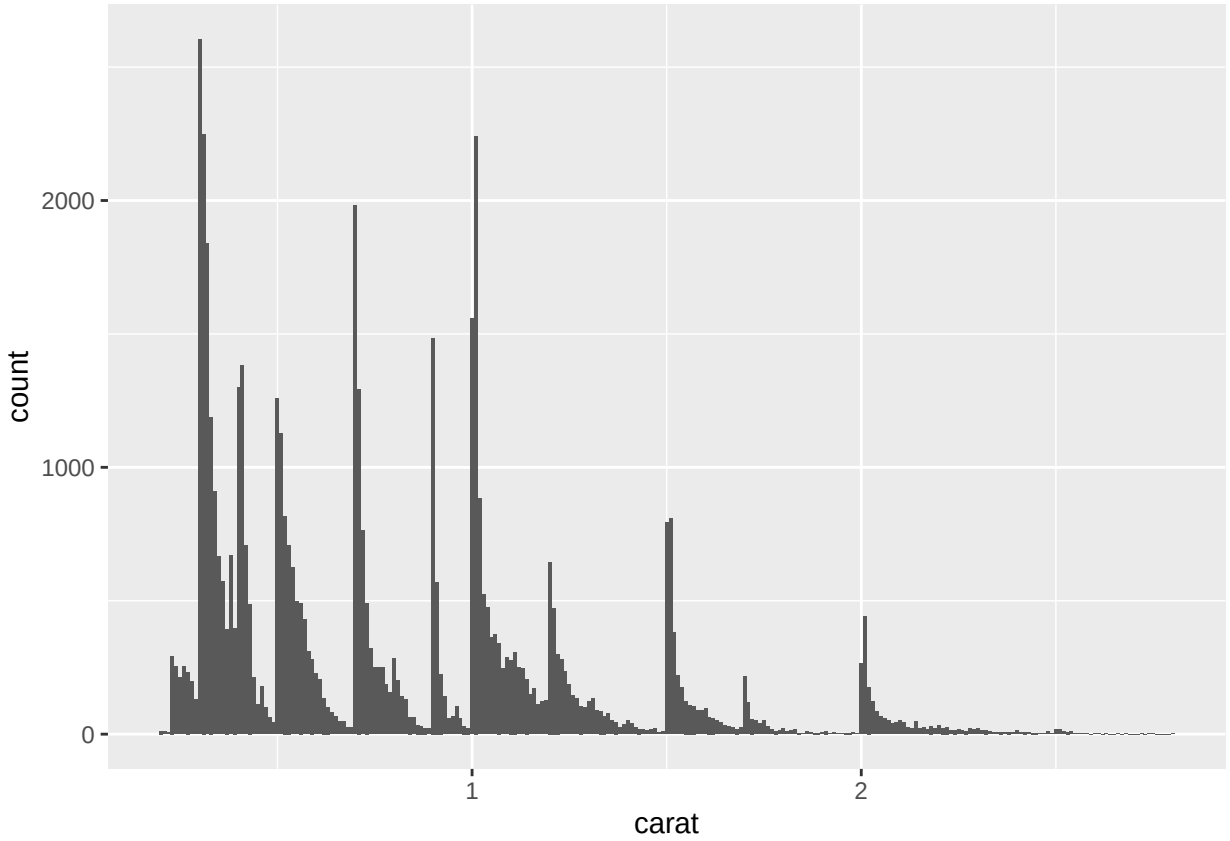


- The tall bars show the common values of a variable; the shorter bars show less-common values.

Ask: - Which values are the most common? Why? - Which values are rare? Why? Does that match your expectations? - Can you see any unusual patterns? If so, what might explain them?

Looking at the distribution of `carat` for smaller diamonds:

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

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



- It seems like there are more diamonds at whole carats and some fractions of carats. Why?
- Why are there more diamonds slightly to the right of each peak than there are to the left of each peak?

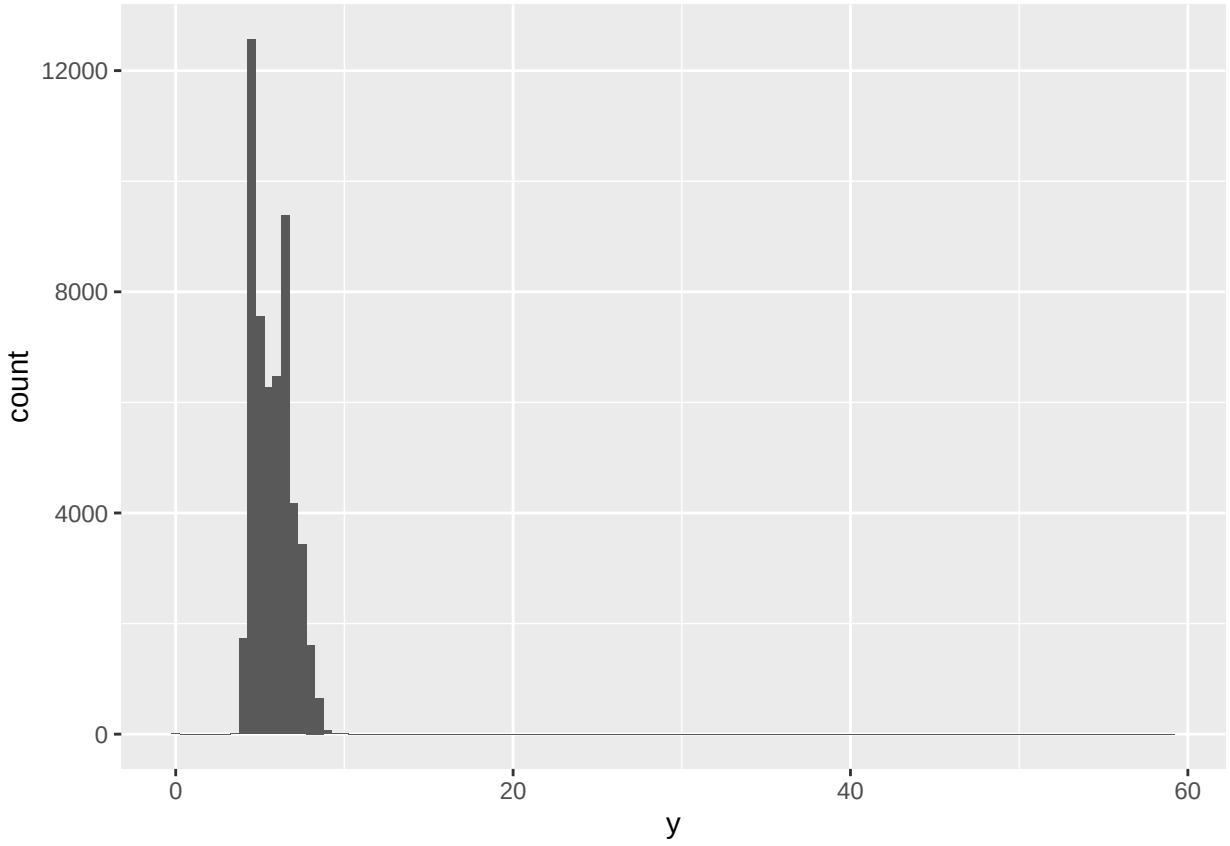
Visualizations can also reveal clusters and/or subgroups in the data. To understand the subgroups in `diamonds`, we can ask: - How are the observations within each subgroup similar to each other?
- How are the observations in separate clusters different from each other?
- How can you explain or describe the clusters?
- Why might the appearance of clusters be misleading?

Unusual values

Outliers: Observations that are unusual, data points that don't seem to fit the pattern.

Looking for outliers in `diamonds`:

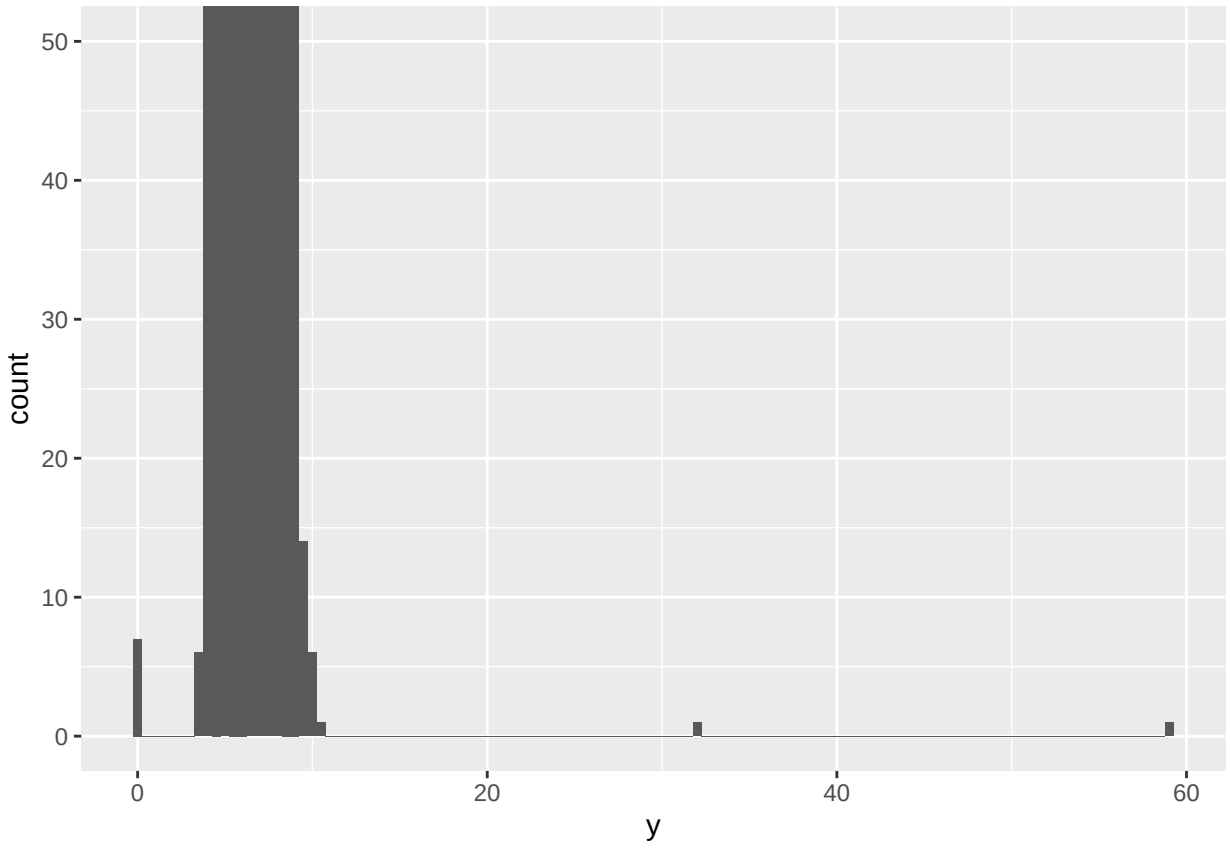
```
ggplot(diamonds, aes(x = y)) +
  geom_histogram(binwidth = 0.5)
```



It is hard to see if there are very short bars in this histogram. We need to zoom in closer:

Zooming in using `coor_cartesian()`:

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



Aha!

Note: `coord_catesian()` has an `xlim()` argument too. The regular `xlim()` and `ylim()` functions in `ggplot2` work slightly differently – they throw away the data outside the limits.

Extracting the unusual values:

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

unusual
```

```
## # A tibble: 9 x 4
##   price     x     y     z
##   <int> <dbl> <dbl> <dbl>
## 1  5139  0      0    0
## 2  6381  0      0    0
## 3 12800  0      0    0
## 4 15686  0      0    0
## 5 18034  0      0    0
## 6  2130  0      0    0
## 7  2130  0      0    0
## 8  2075  5.15  31.8  5.12
## 9 12210  8.09  58.9  8.06
```

- The zero-values seem to suggest that these are missing data that are coded as 0's instead of `NA`s. We might chose to re-code these values as `NA`.

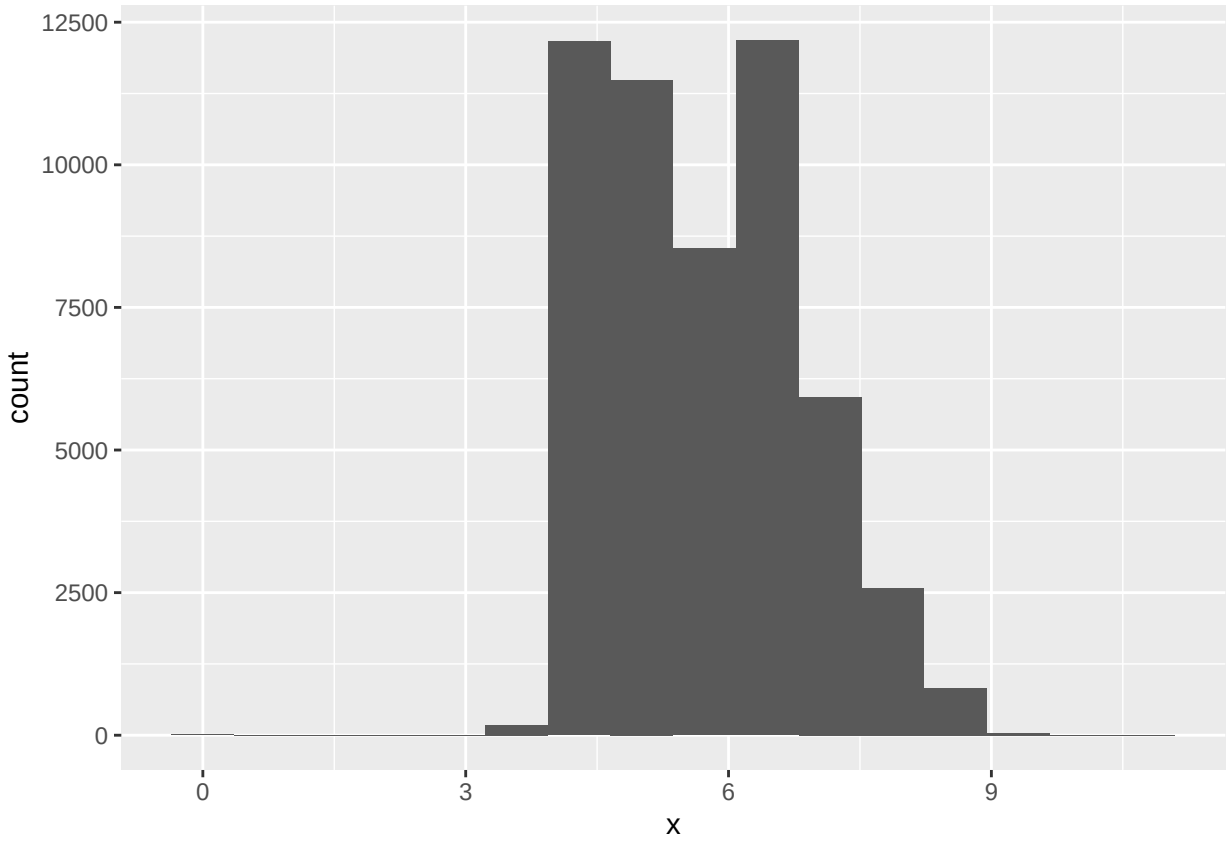
- The values higher than higher than 20 seem to be incorrect as well – the prices are too "cheap" for their size!

Exercises:

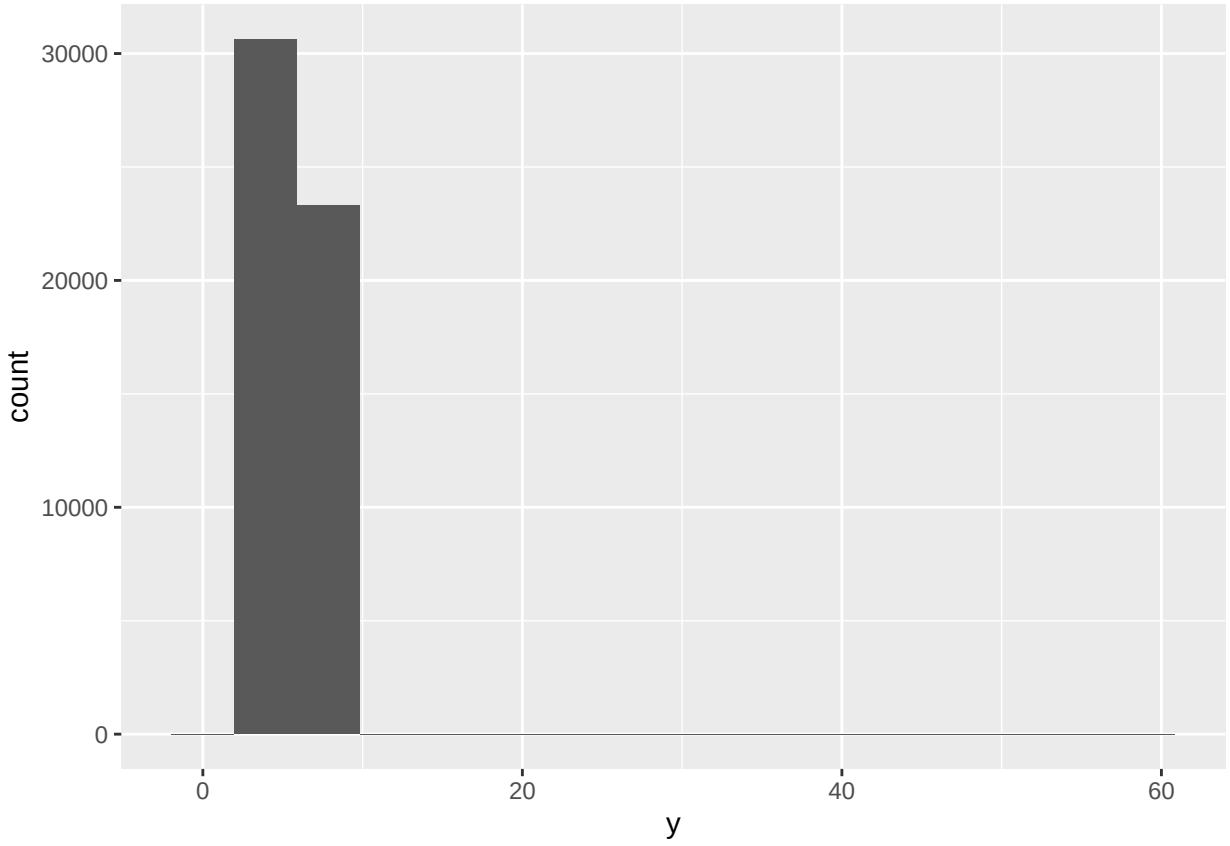
E1. Explore the distribution of each of the `x`, `y`, and `z` variables in `diamonds`. What do you learn? Thin about a diamond and how you might decide which dimension is the length, width, and depth.

Ans.

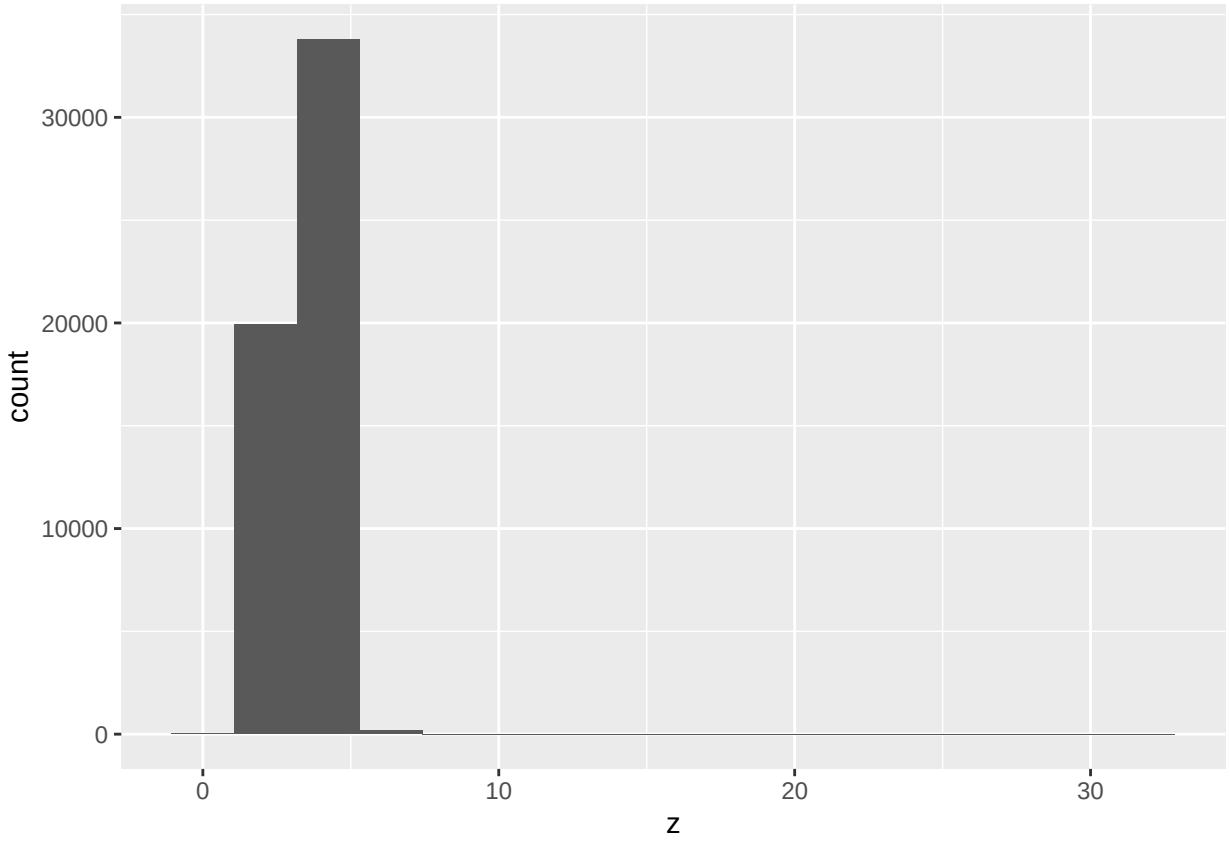
```
# using Sturge's rule to get optmal no. of bins
N = 53490
K = as.integer(1 + 3.322 * log10(N))

# distribution of x
ggplot(diamonds, aes(x = x)) +
  geom_histogram(bins = K)
```




```
# distribution of y
ggplot(diamonds, aes(x = y)) +
  geom_histogram(bins = K)
```



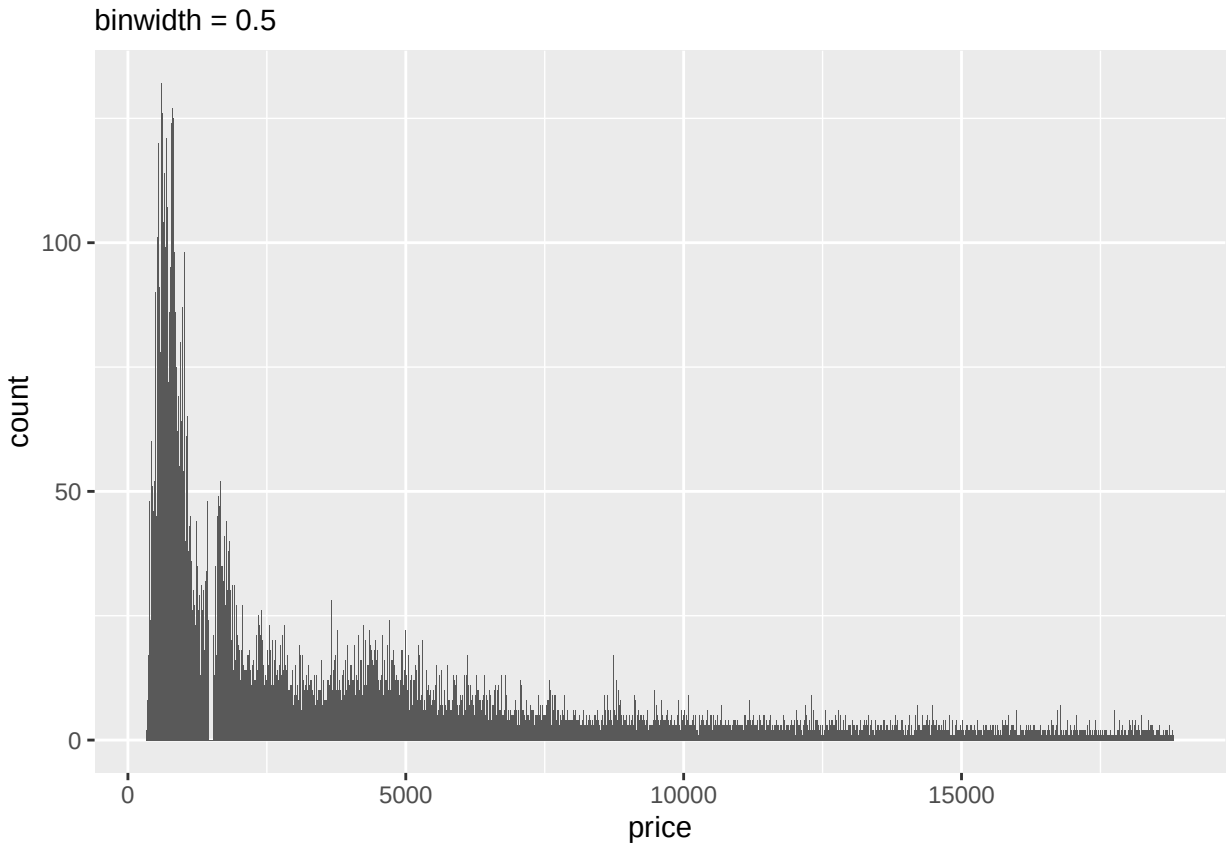

```
ggplot(diamonds, aes(x = z)) +
  geom_histogram(bins = K)
```



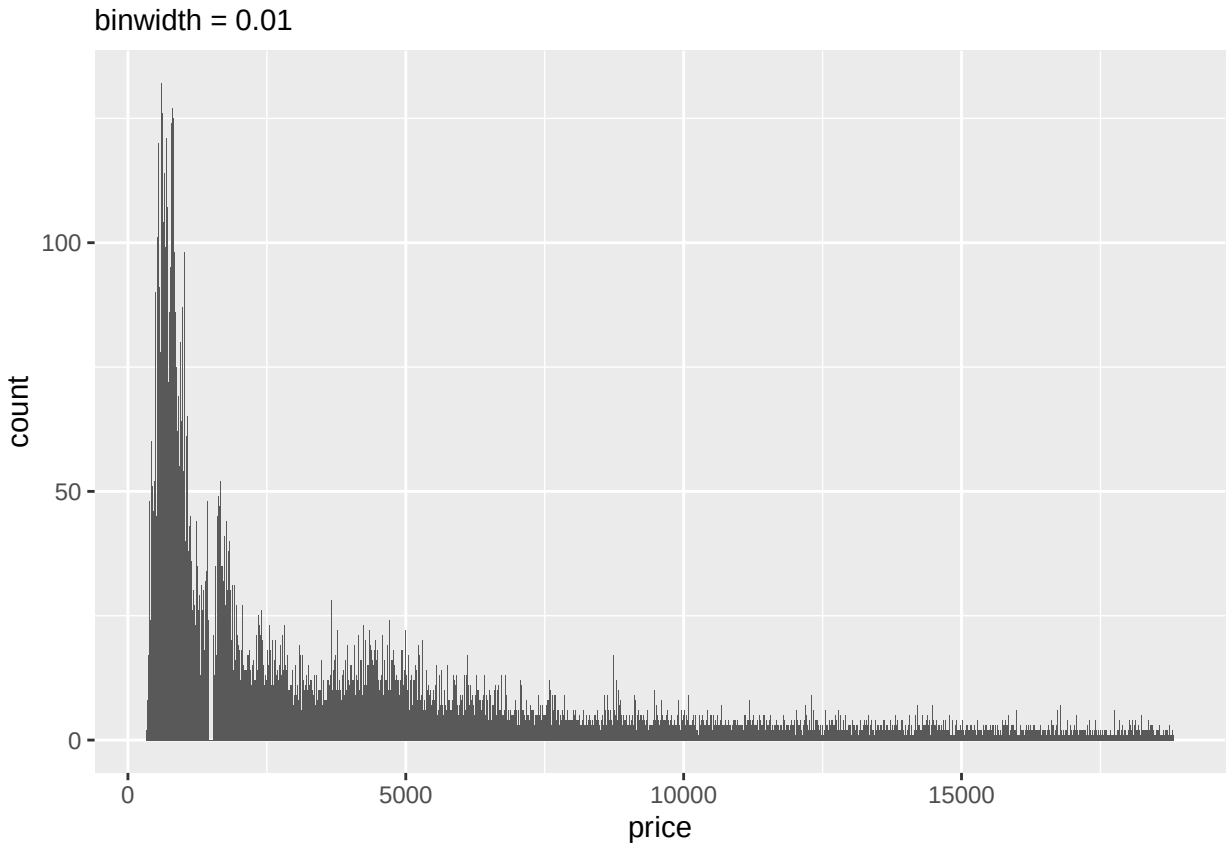
E2. Explore the distribution of `price`. Do you discover anything unusual or surprising? (hint: Carefully think about the `binwidth`) and make sure you try a wide range of values.

Ans.

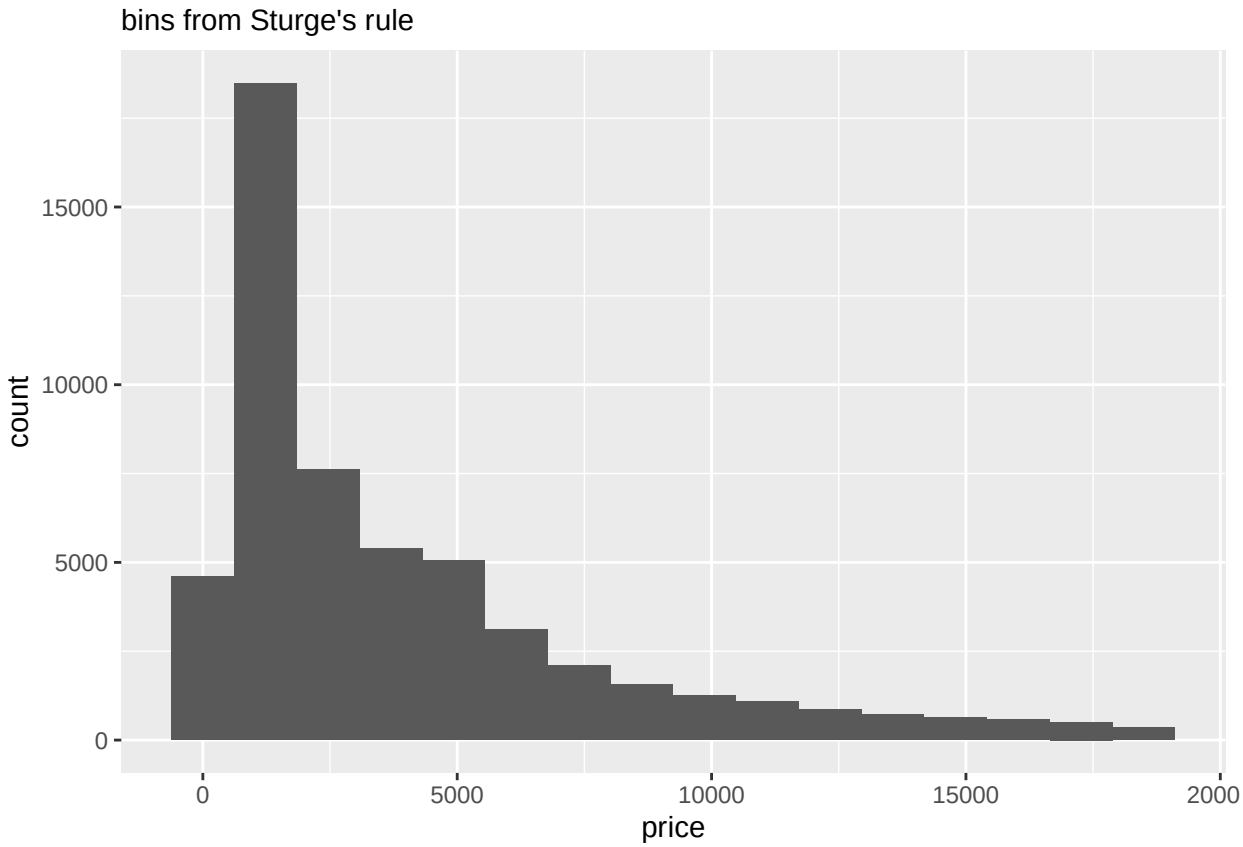
```
# binwidth = 0.5
ggplot(diamonds, aes(x = price)) +
  geom_histogram(binwidth = 0.5) +
  labs(subtitle = "binwidth = 0.5")
```




```
# binwidth = 0.01
ggplot(diamonds, aes(x = price)) +
  geom_histogram(binwidth = 0.5) +
  labs(subtitle = "binwidth = 0.01")
```




```
# optimal bins from sturge's rule
ggplot(diamonds, aes(x = price)) +
  geom_histogram(bins = K) +
  labs(subtitle = "bins from Sturge's rule")
```



Most diamonds priced below $2500. Using different values for `binwidth`, I struggled to see this distribution. Only when I used the theoretically optimal number of bins did I see the pattern.

#E3. How may diamonds are 0.99 carat? How many are 1 carat? What do you think is the cause of the difference?

```
# number of diamonds that are 0.99 carat
diamonds |>
  filter(carat == 0.99) |>
  select(carat) |>
  summarize(n = n()) |>
  pull()
```

```
## [1] 23
```

```
diamonds |>
  filter(carat == 1) |>
  select(carat) |>
  summarize(n = n()) |>
  pull()
```

```
## [1] 1558
```

```
# number of diamonds that are 1 carat
diamonds |>
  mutate(
    is_099_carat = ifelse(carat == 0.99, T, F),
    is_1_carat = ifelse(carat == 1, T, F)
  ) |>
```

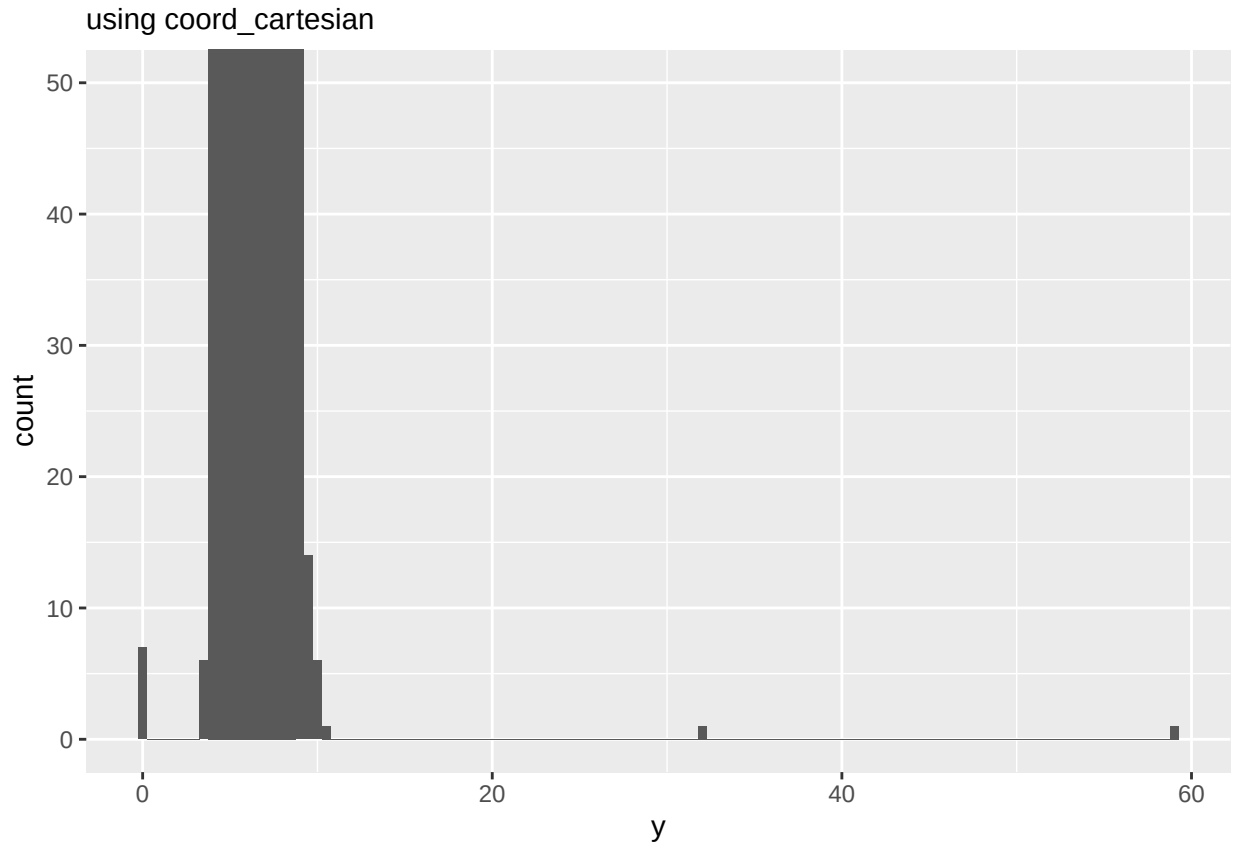
```
  select(is_099_carat, is_1_carat) |>
  summarize(
    sum_099_carat = sum(is_099_carat),
    sum_1_carat = sum(is_1_carat)
  )
```

```
## # A tibble: 1 x 2
##   sum_099_carat sum_1_carat
##           <int>       <int>
## 1            23        1558
```

I don't know why this is so. It could be due to mistakes during cutting i.e. the artisan cut a bit too much?

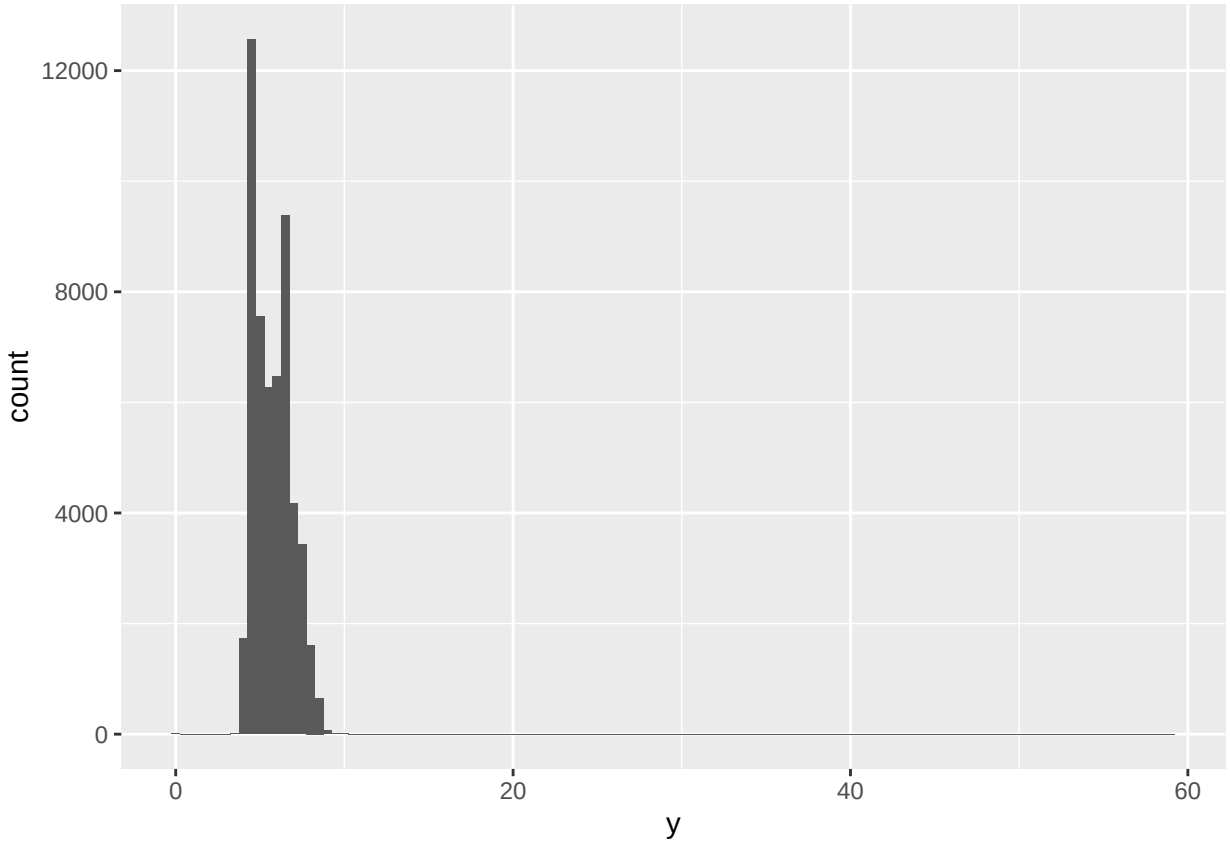
E4. Compare and contrast `coord_cartesian()` vs. `xlim()` or `ylim()` when zooming in on a histogram. What happens of you leave binwidth unset? What happens if you try and zoom so only half a bar shows?

```
ggplot(diamonds, aes(x = y)) +
  geom_histogram(binwidth = 0.5) +
  coord_cartesian(ylim = c(0, 50)) +
  labs(subtitle = "using coord_cartesian")
```




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

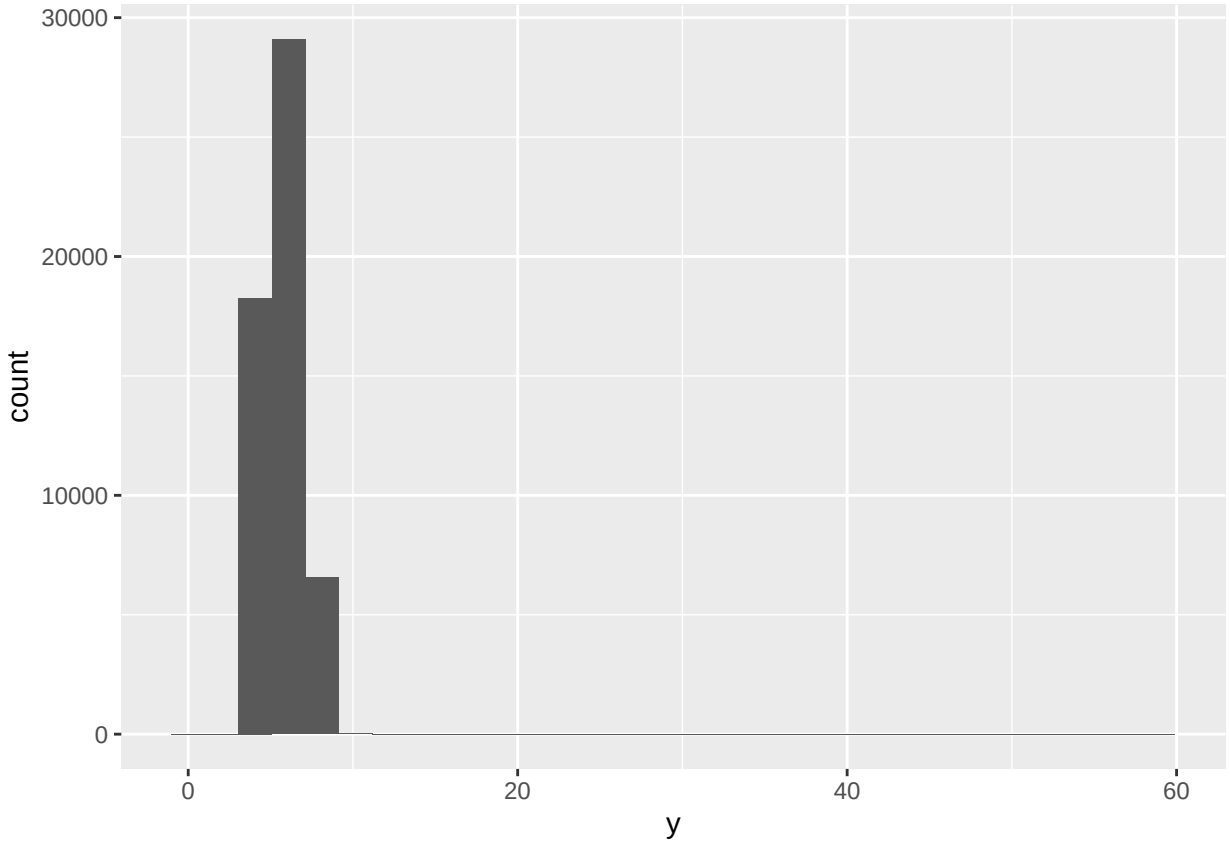
```
## Warning in geom_histogram(binwidth = 0.5, ylim = c(0, 50)): Ignoring unknown
## parameters: `ylim`
```




```
ggplot(diamonds, aes(x = y)) +
  geom_histogram(ylim = c(0, 50))
```

```
## Warning in geom_histogram(ylim = c(0, 50)): Ignoring unknown parameters: `ylim`
```

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



- For zooming in on the graph, `coord_cartesian()` works, using the `ylim()` argument does not work since it removes the data outside of the limits.

Unusual values

Option 1: dropping the row with strange values:

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

Dropping the unusual values is not recommended since you are throwing away data – if there are many unusual values, you won't have much data to work with.

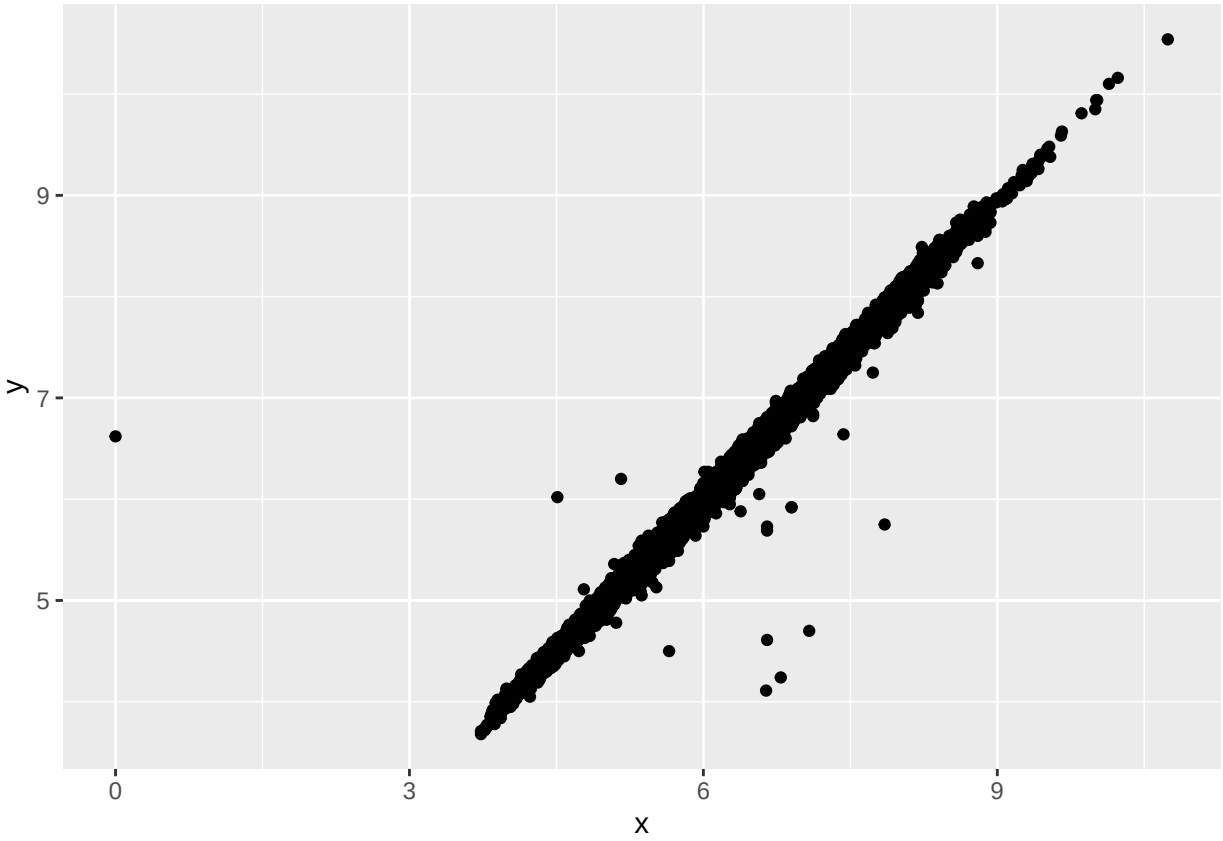
Option 2: Replacing the unusual values with missing values:

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

When plotting, ggplot does not show the missing values:

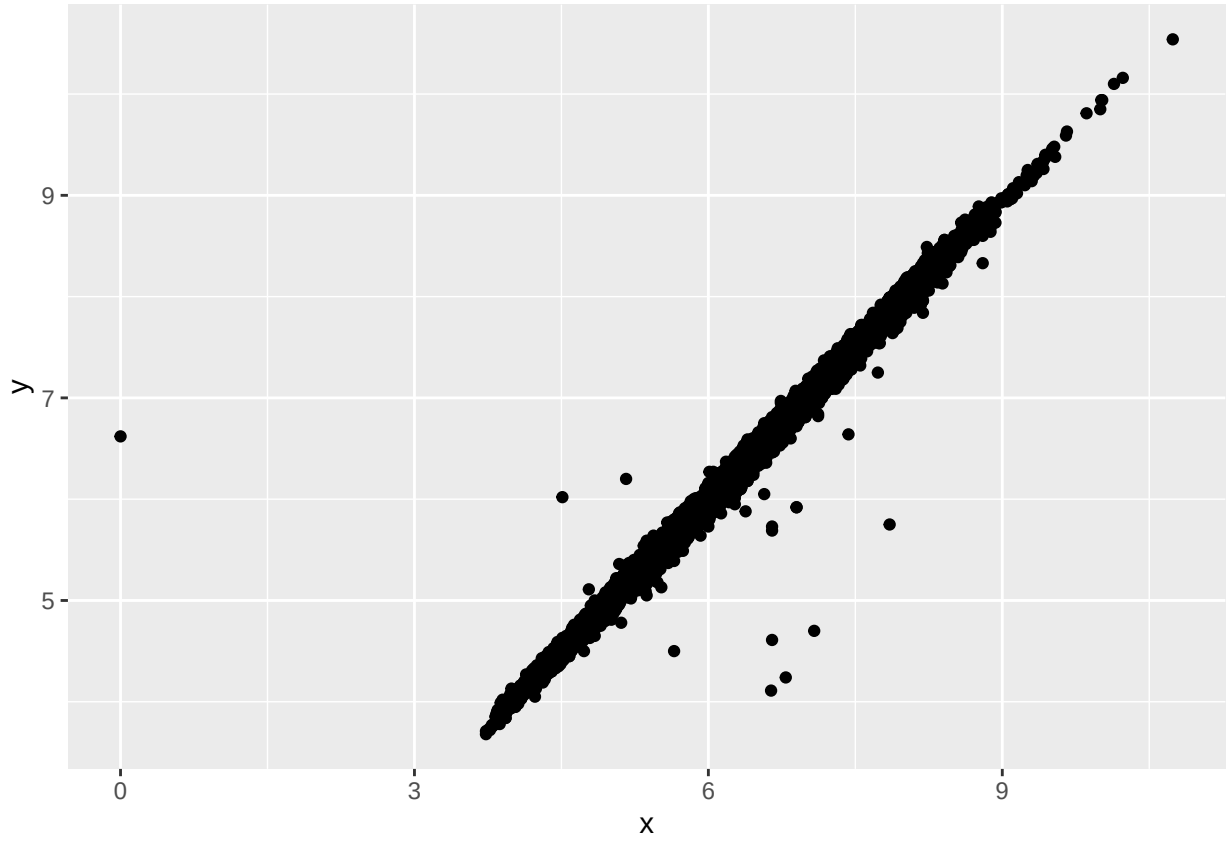
```
ggplot(diamonds2, aes(x = x, y = y)) +
  geom_point()
```

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



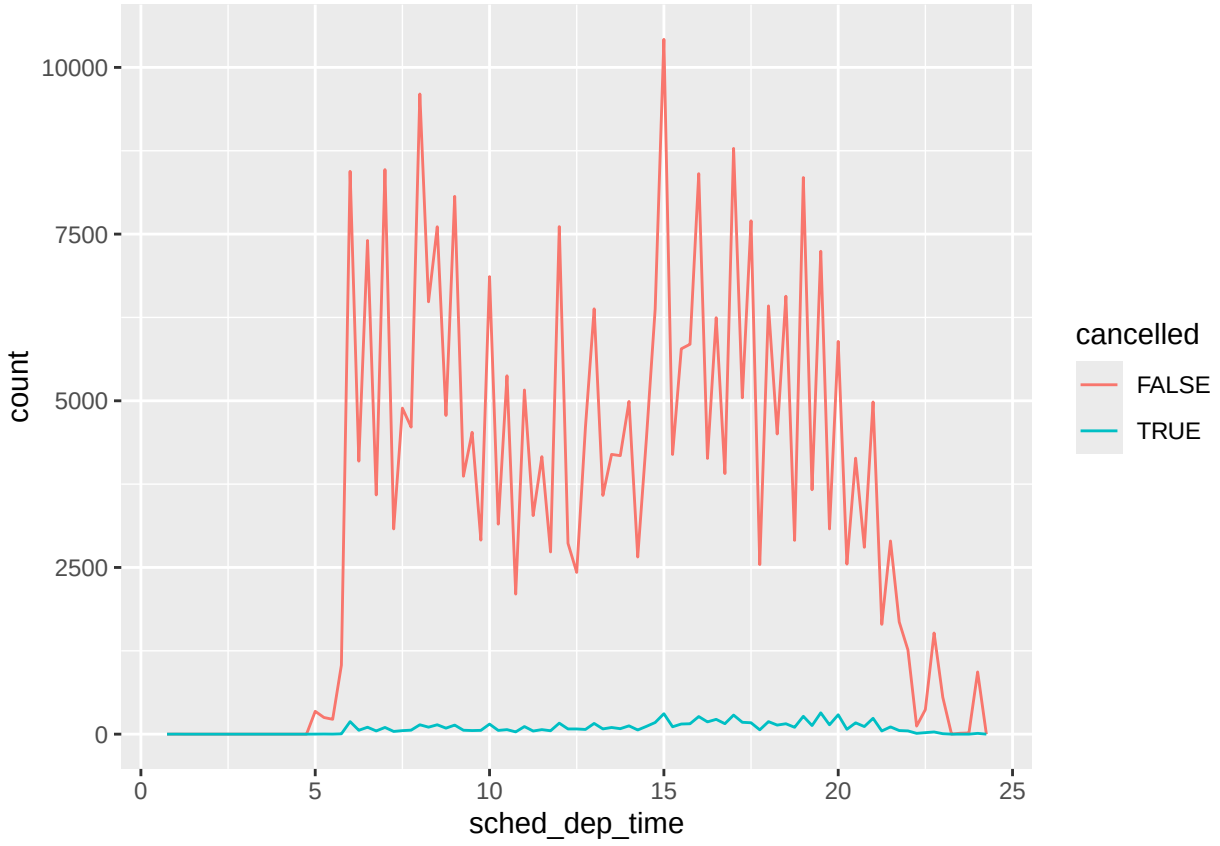
Supressing the waring with `na.rm =TRUE` argument:

```
ggplot(diamonds2, aes(x = x, y = y)) +
  geom_point(na.rm = TRUE)
```



It is also a good idea to plot the missing values to understand the difference between them and the values which are not 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)
```



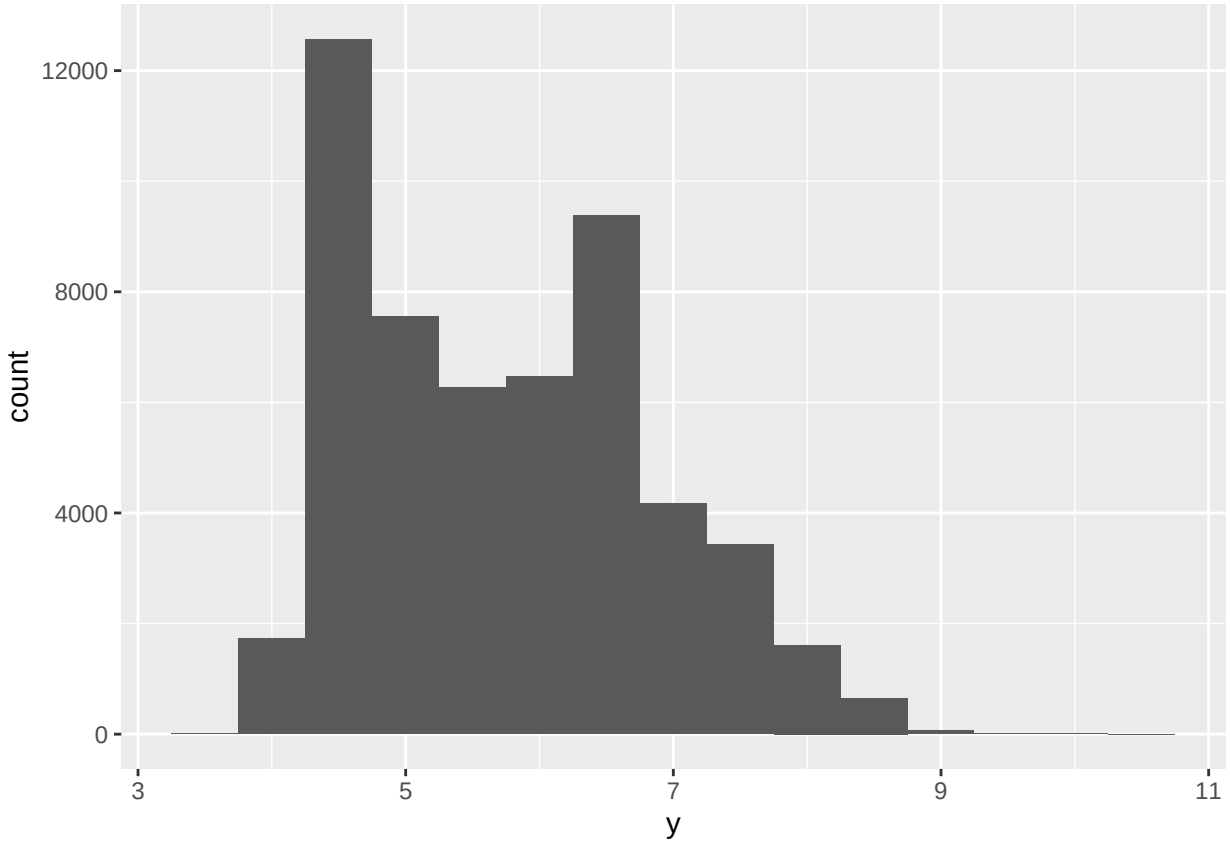
Exercises

E1. What happens to missing values in a histogram? What happens to missing values in a bar chart? Why is there a difference in how missing values are handled in histograms and bar charts?

Ans.

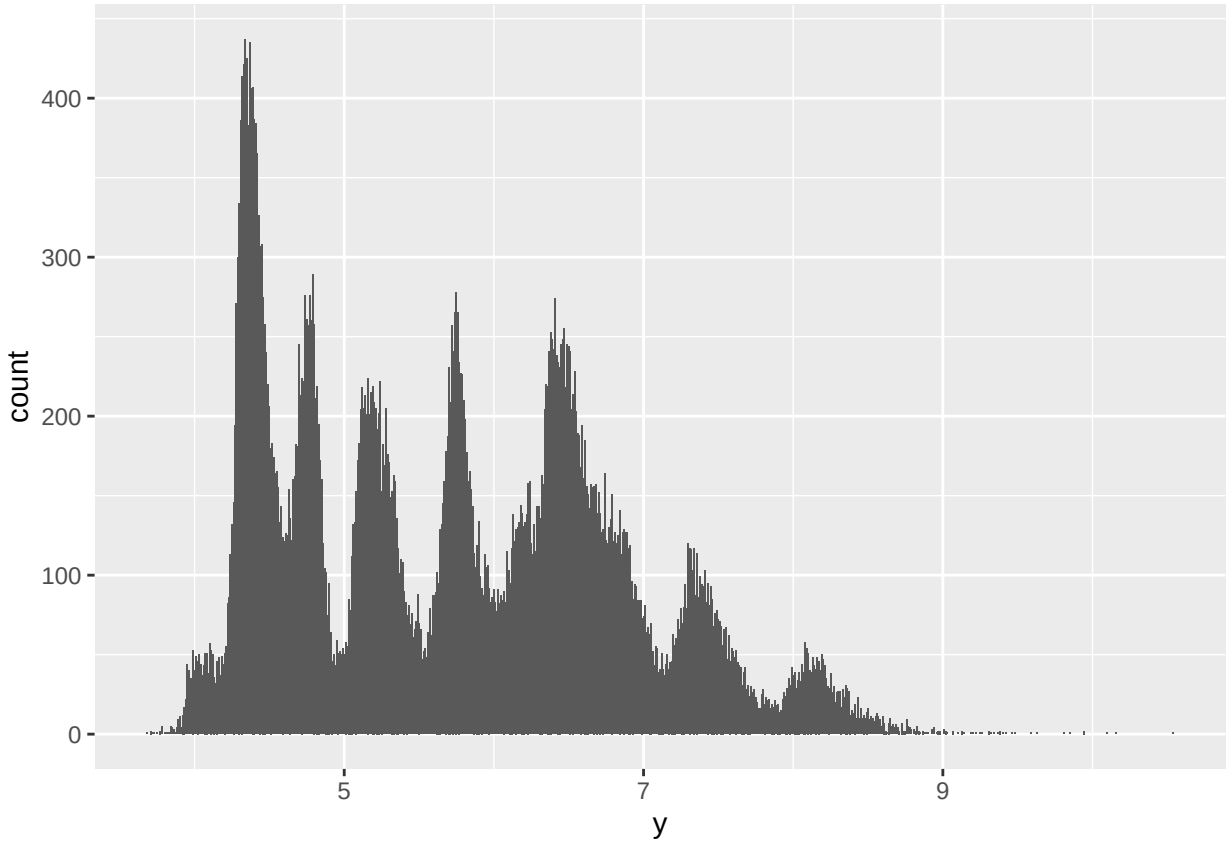
```
ggplot(diamonds2, aes(x = y)) +
  geom_histogram(binwidth = 0.5)
```

```
## Warning: Removed 9 rows containing non-finite outside the scale range
## (`stat_bin()`).
```




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

```
## Warning: Removed 9 rows containing non-finite outside the scale range
## (`stat_count()`).
```

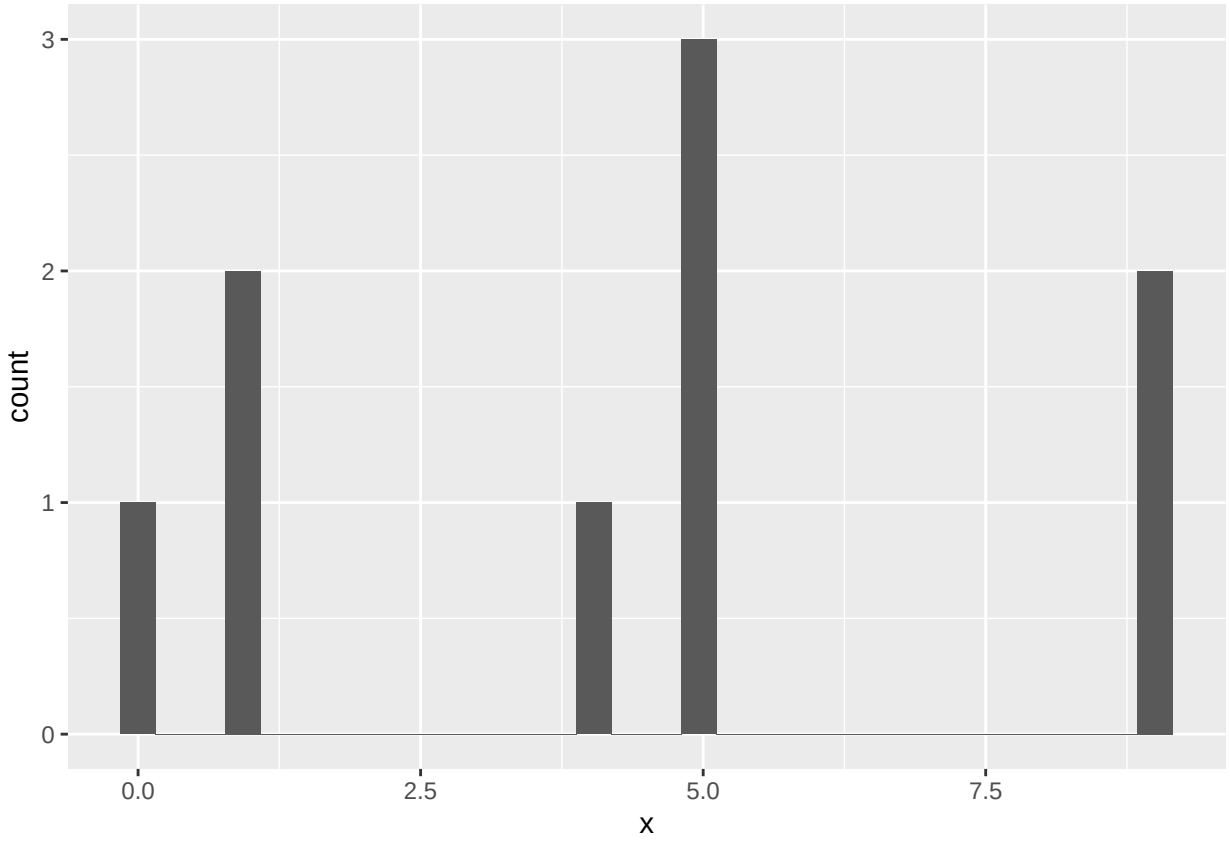


`geom_histogram()` and `geom_bar()` removed the missing values.

```
x <-  c(9, 9, 4, 5, 1, 0, 5, 5, 1, NA)
df <- data.frame(x)

ggplot(df, aes(x = x)) +
  geom_histogram()
```

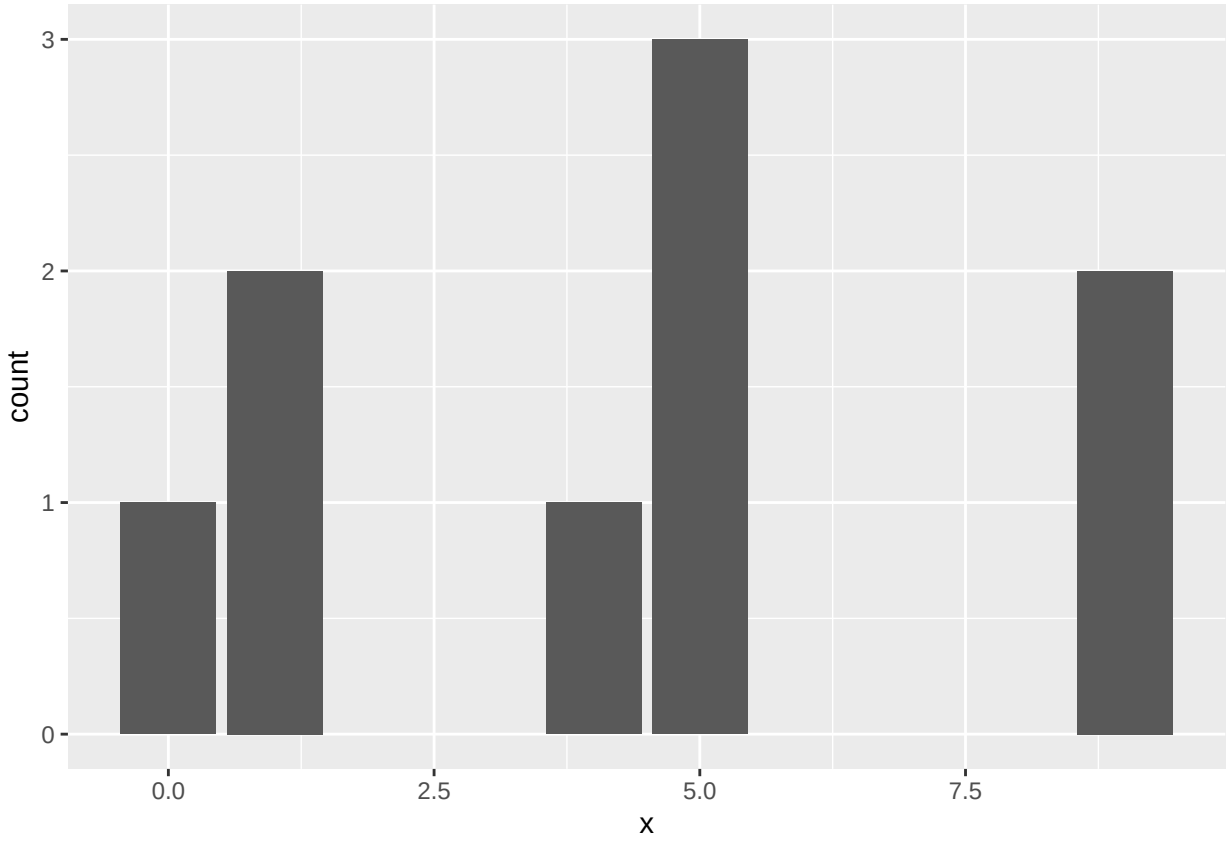
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_bin()`).
```




```
ggplot(df, aes(x = x)) +
  geom_bar()
```

```
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_count()`).
```



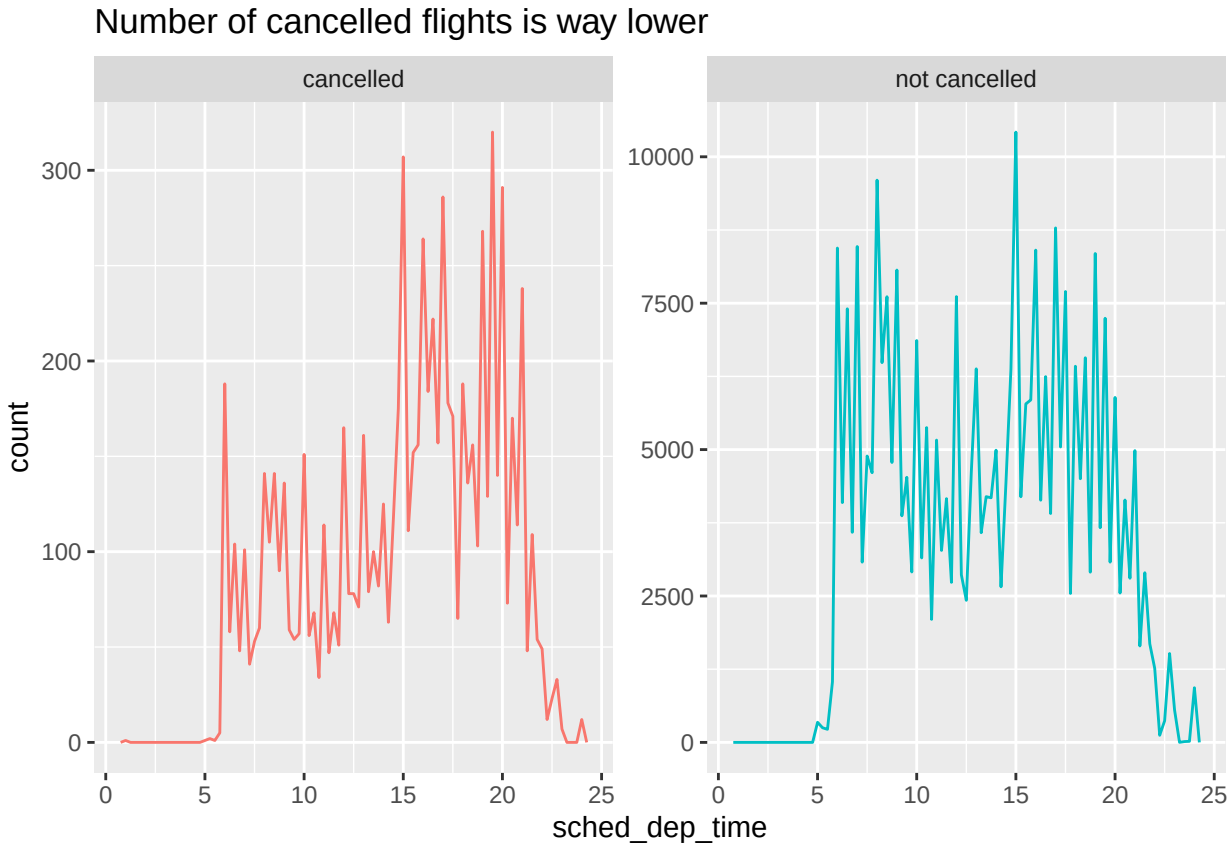
I think that in previous versions of R, missing values are treated differently by `geom_histogram()` and `geom_bar()`

E2. What does `na.rm = TRUE` do in `mean()` and `sum()`?

Ans. The argument removes `NA`s first before performing the mean and sum operations, respectively.

E3. Recreate the frequency plot of `scheduled_dep_time` colored by whether the flight was cancelled or not. Also facet by the cancelled variable. Experiment with different values of the scales variable in the faceting function to mitigate the effect of more non-cancelled flights than cancelled flights.

```
nycflights13::flights |>
  mutate(
    cancelled = is.na(dep_time),
    cancelled = ifelse(cancelled == T, "cancelled", "not cancelled"),
    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, show.legend = FALSE) +
  facet_wrap(~ cancelled, scales = "free_y") +
  labs(
    title = "Number of cancelled flights is way lower"
  )
```

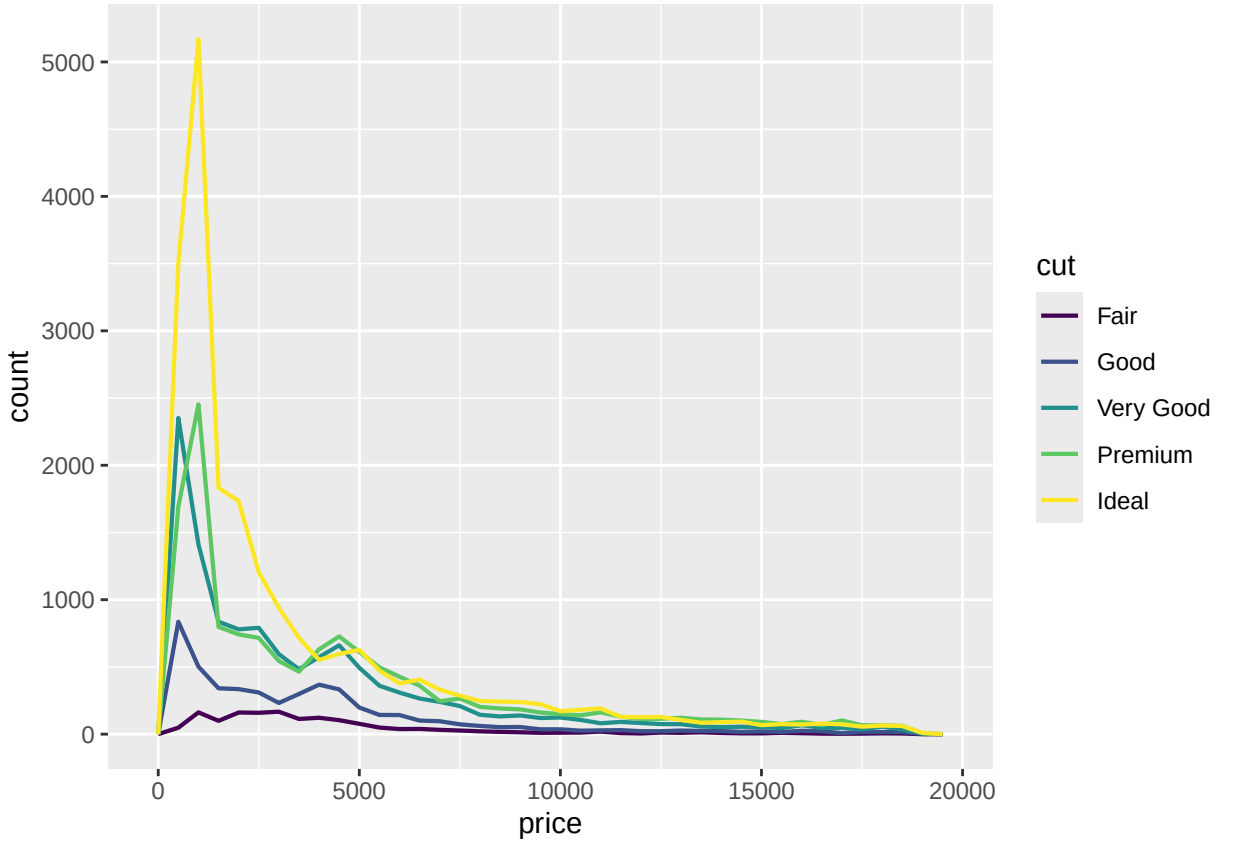


Covariation

- **Covariation** is the tendency for the values of two variables to vary together in a related way.
- The best way to spot this is through visualization.

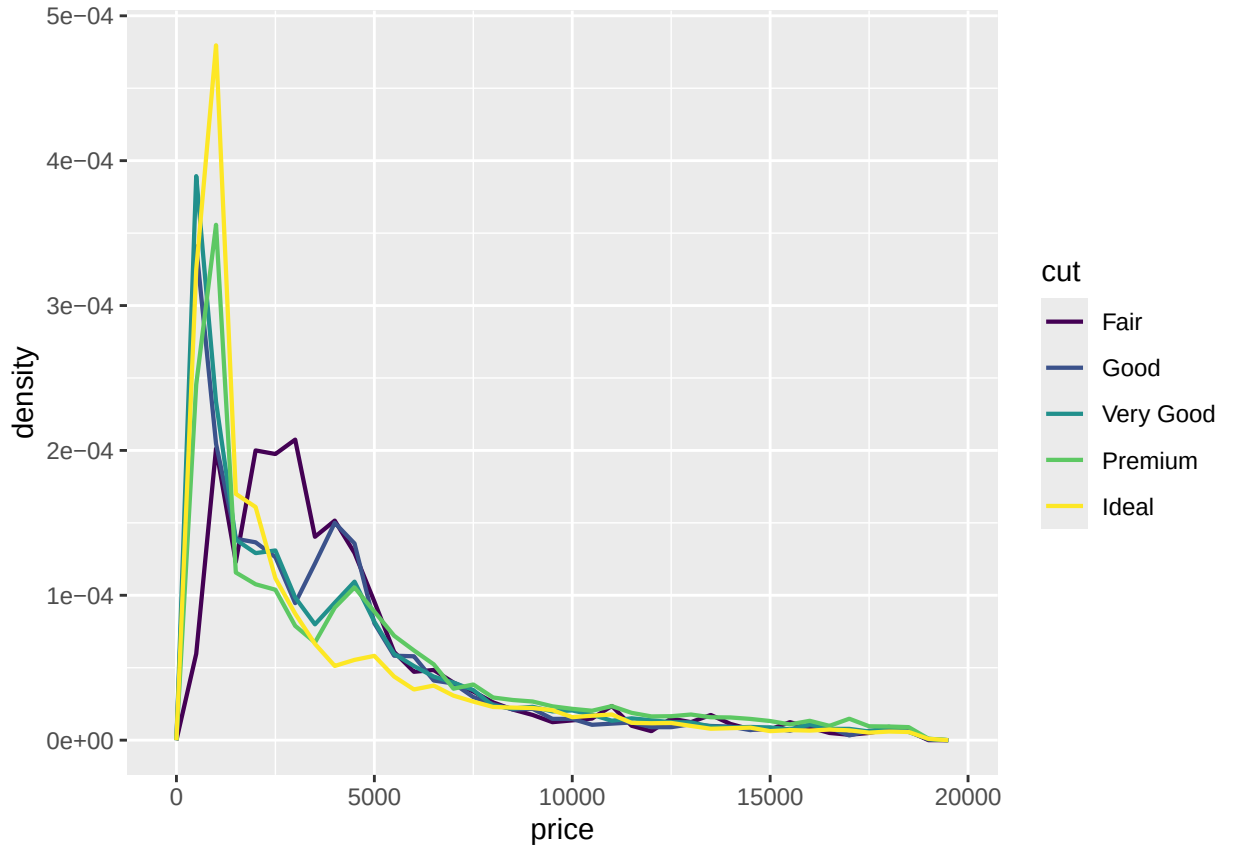
Example:

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



- The default appearance of `geom_freqpoly()` is not that useful here because it shows th overall count. - To make the comparison easier, we can display the *density* instead of count.

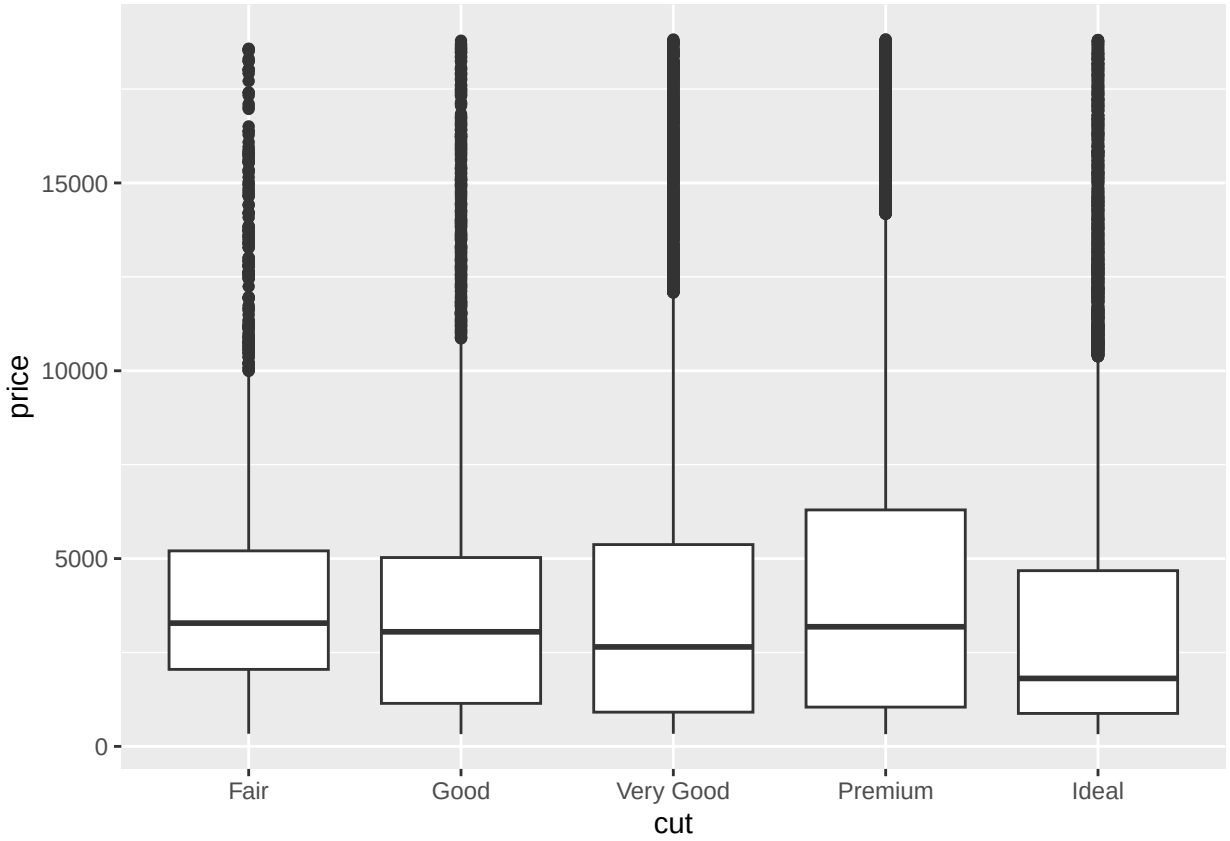
```
ggplot(diamonds, aes(
  x = price, y = after_stat(density)
)) +
  geom_freqpoly(
    aes(color = cut),
    binwidth = 500,
    linewidth = 0.75
  )
```



Note: density is mapped to y, but since `density` is not a variable in the dataset, it must be calculated first – the `after_stat()` function was used for that.

In the plot, it looks like the diamond with the lowest quality (fair) has the highest prices. Is this correct or is this due to the complexity of frequency plots? Perhaps a simpler plot is necessary.

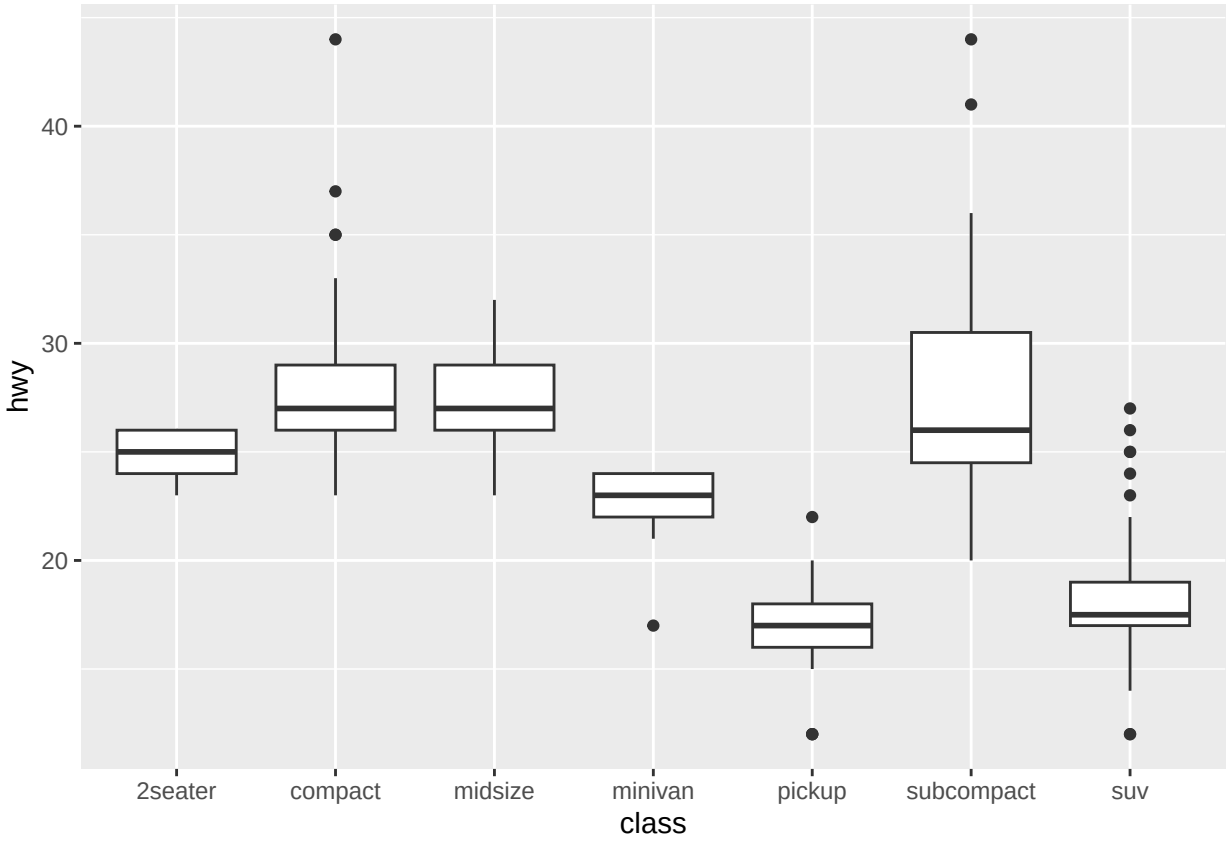
```
ggplot(diamonds, aes(x = cut, y = price)) +
  geom_boxplot()
```



Note: The varible `cut` is an ordered factor in the data, i.e. Fair -> Good -> Very Good -> Premium... If the variable is not an ordered factor, the order can be assigned using the function `fct_reorder()`.

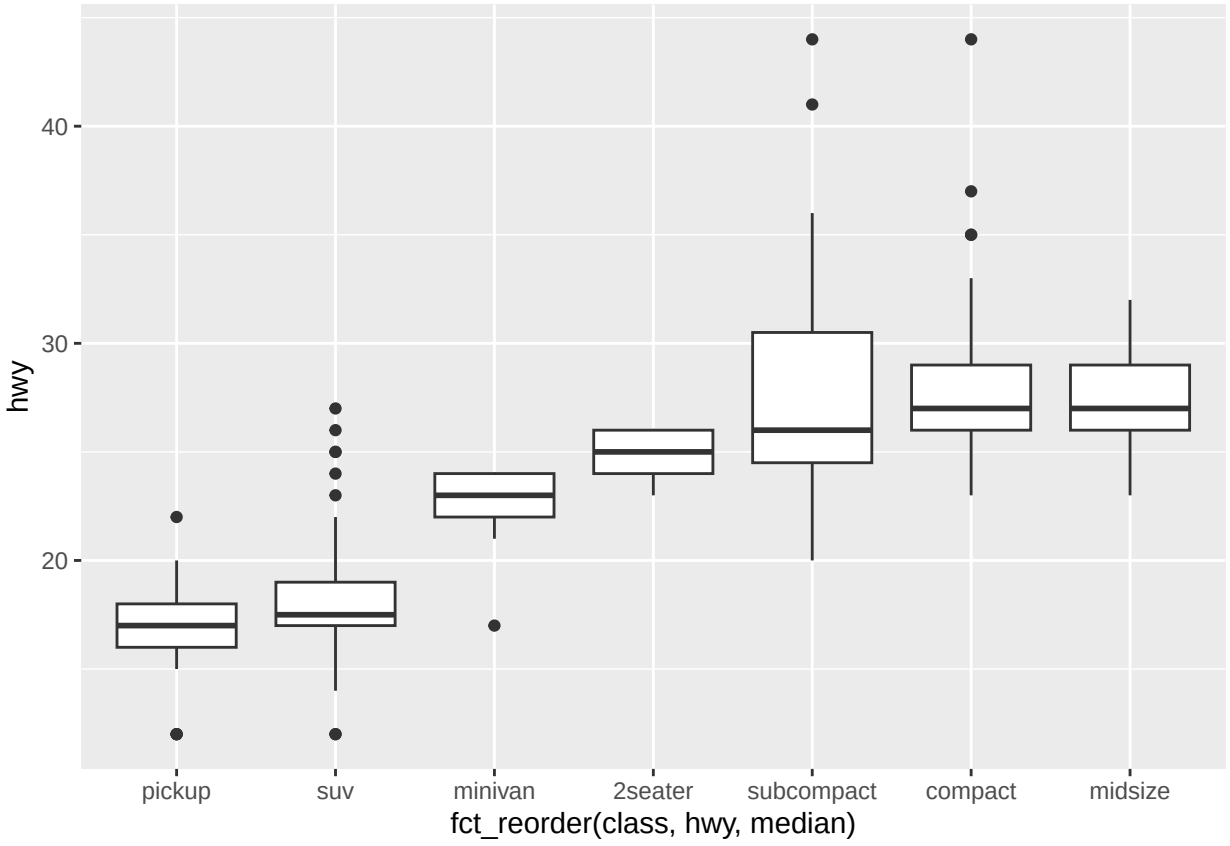
Another Example:

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



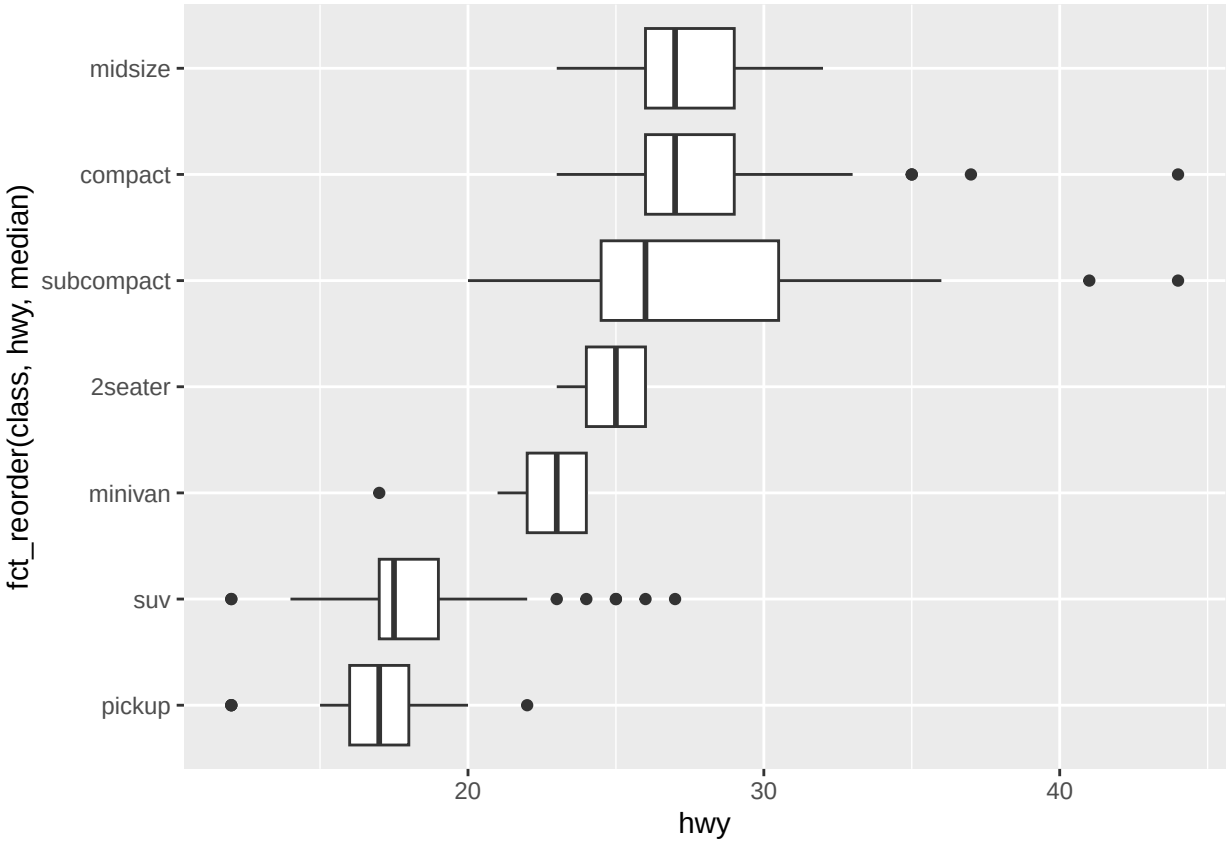
The variable `class` is not ordered, in this case. We can set an order to `class` like this:

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



Rotated version:

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



#xercises

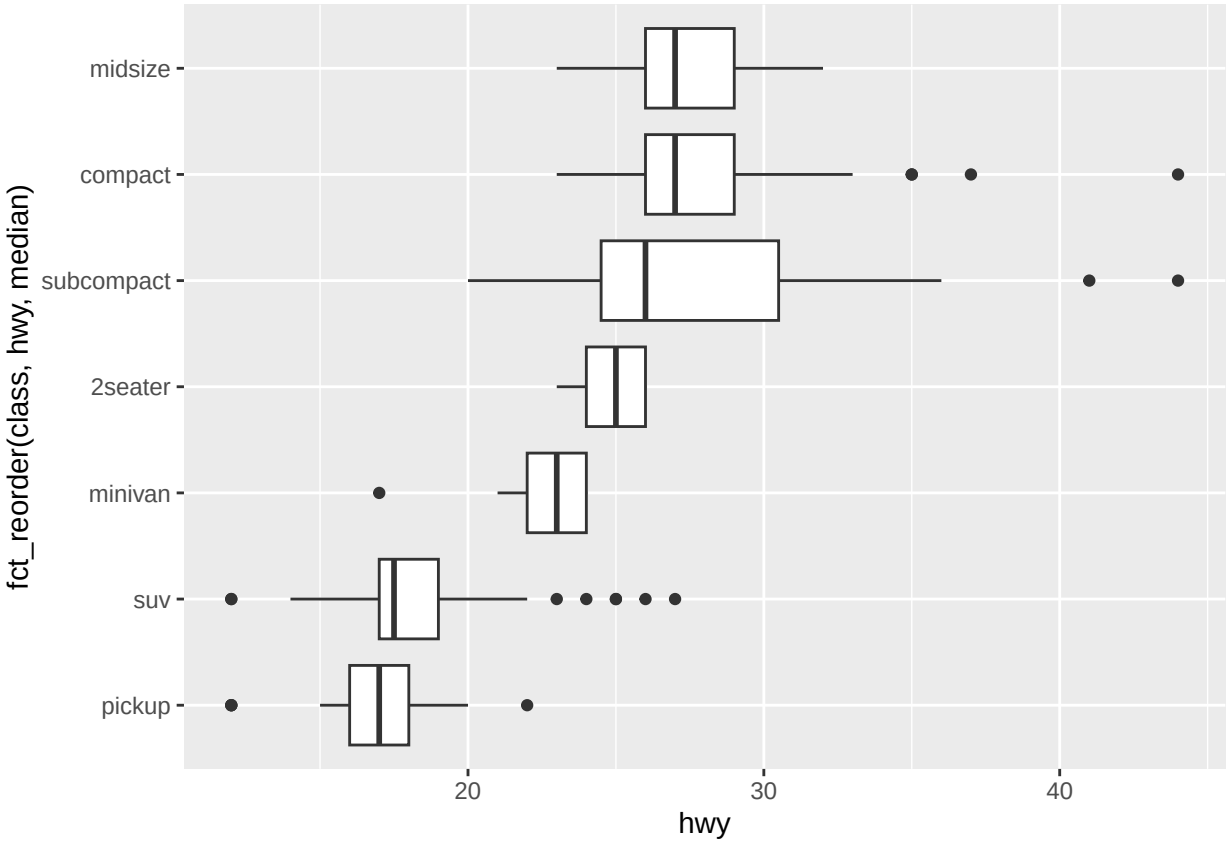
E1. Use what you've learned to improve the visualization of the departure times of cancelled vs. non-cancelled flights.

E2. Based on EDA, what variable in the `diamonds` dataset appears to be most important for predicting the price of a diamond? How is that variable correlated with cut? Why does the combination of those two relationships lead to lower quality diamonds being more expensive?

E3. Instead of exchanging the x and y variables, add `coord_flip()` as a new layer to the vertical boxplot to create a horizontal one. How does this compare to exchanging the variables?

Ans.

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



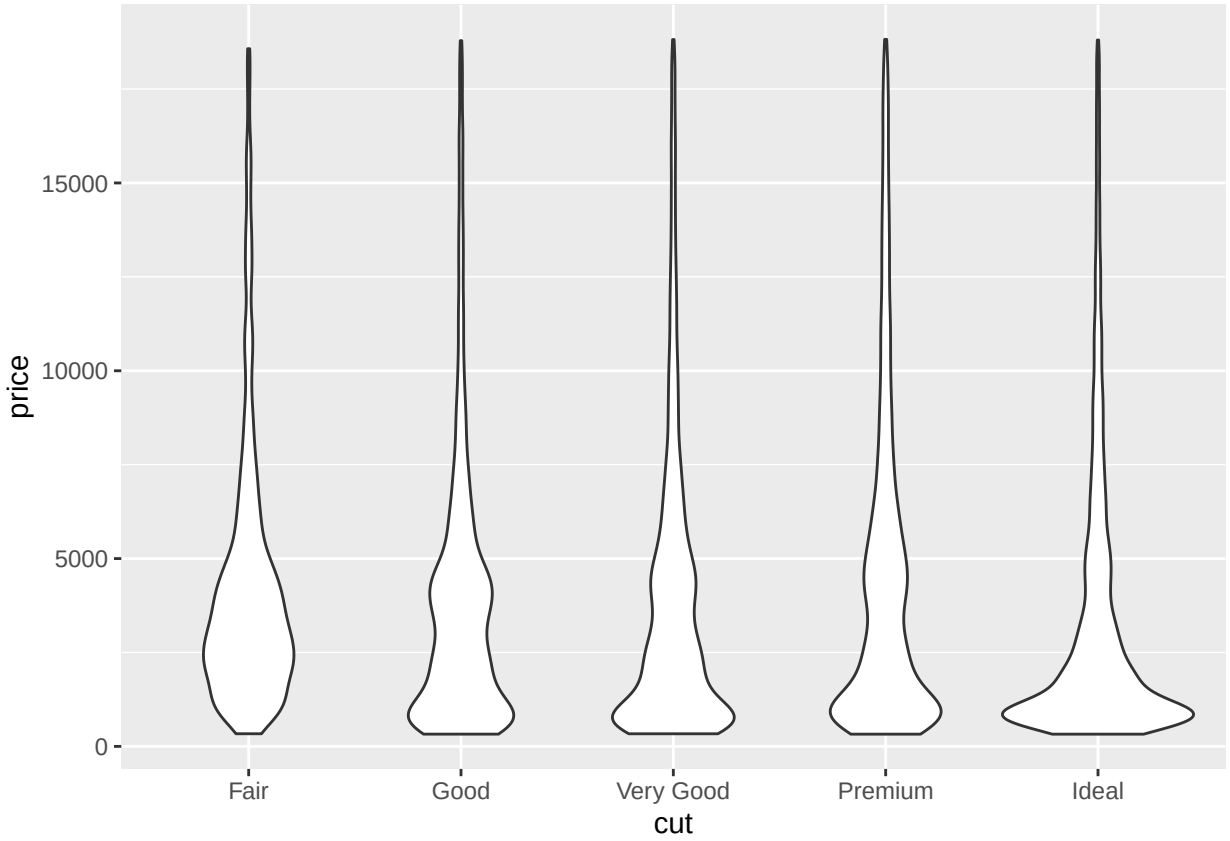
The result is exactly similar.

E4. One problem with boxplots is that they were developed in an era of much smaller datasets and tend to display a prohibitively large number of "outlying values". One approach to remedy this problem is the letter value plot. Install the `lvplot` package, and try using `geom_lv()` to display the distribution of price vs. cut. What do you learn? How do you interpret the plots?

E5. Create a visualization of diamond prices vs. a categorical variable from the diamonds dataset using geom_violin(), then a faceted geom_histogram(), then a colored `geom_freqpoly()`, and then a colored geom_density(). Compare and contrast the four plots. What are the pros and cons of each method of visualizing the distribution of a numerical variable based on the levels of a categorical variable?

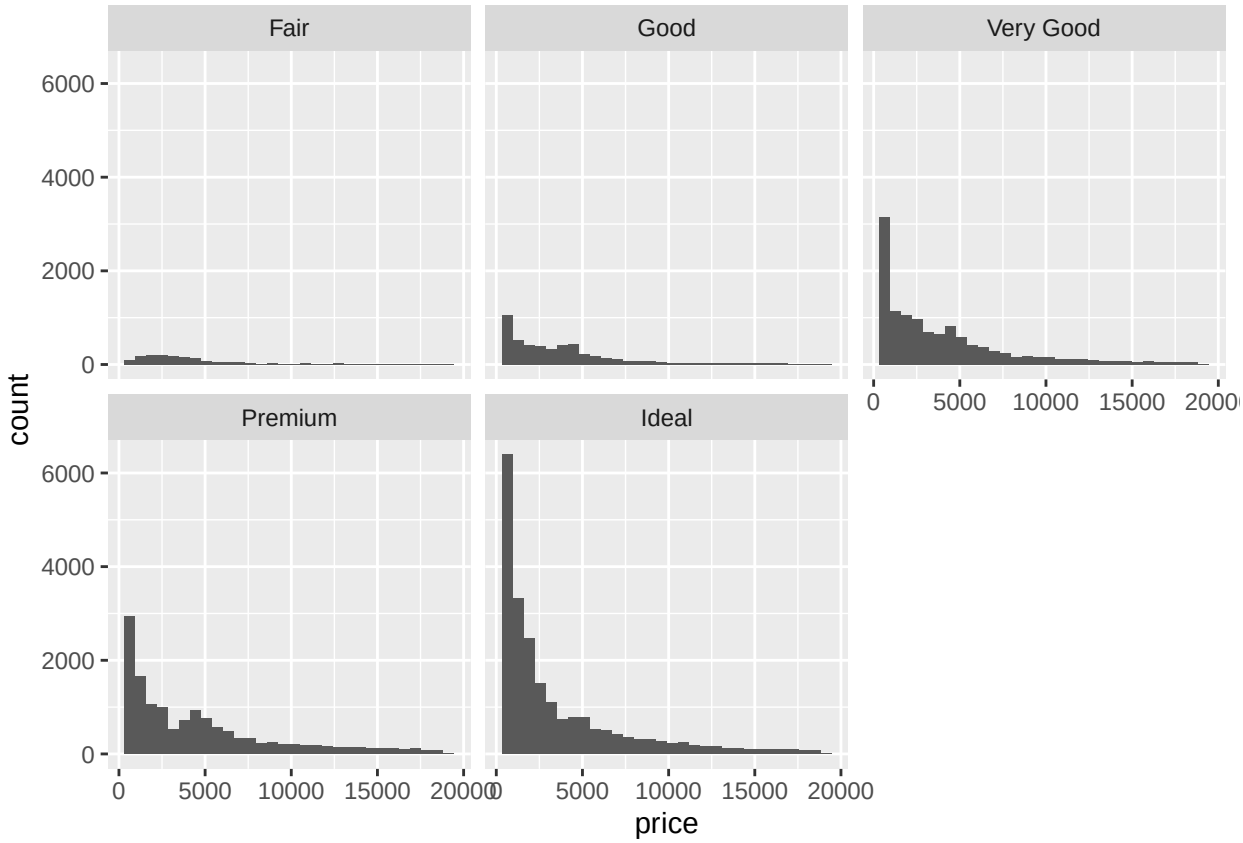
Ans.

```
p <- ggplot(diamonds, aes(
  x = cut, y = price
))

p + geom_violin()
```

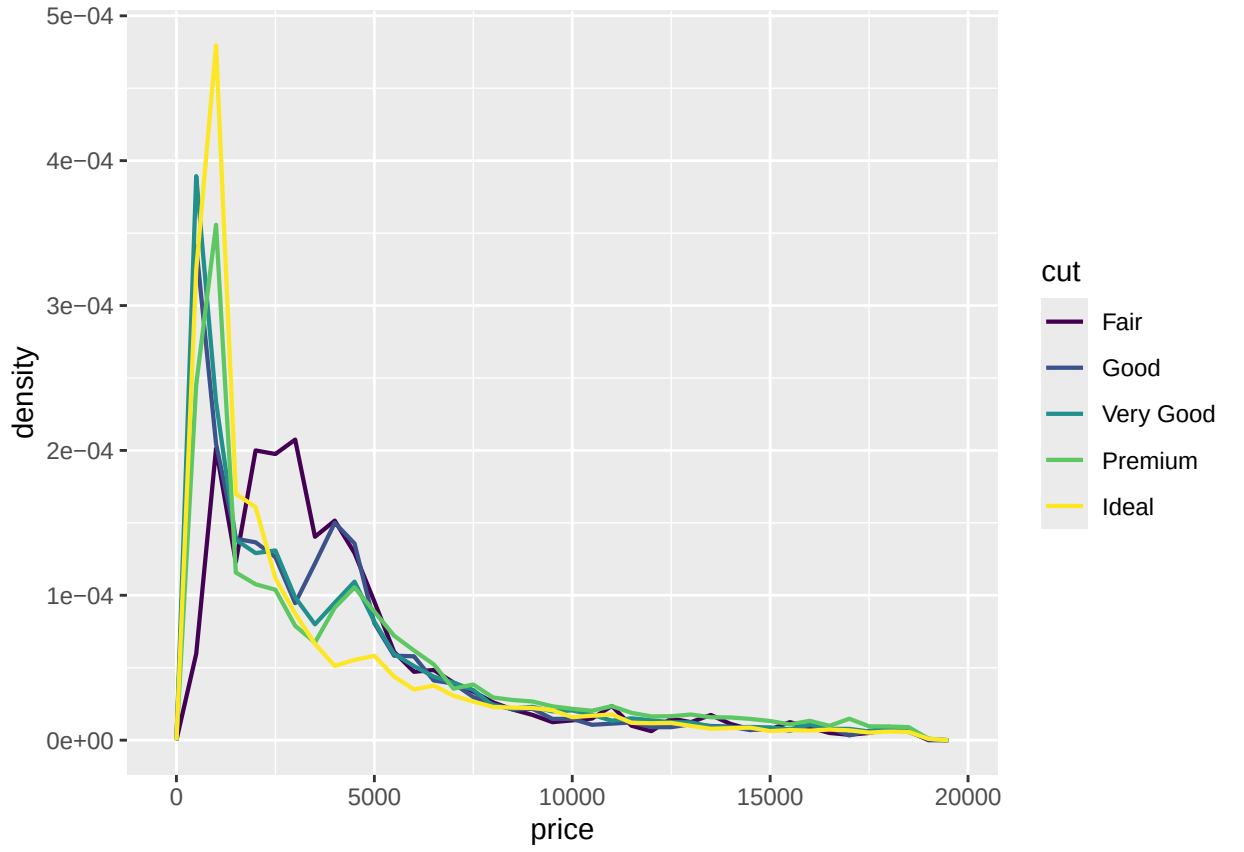



```
ggplot(diamonds, aes(x = price)) +
  geom_histogram() +
  facet_wrap(~ cut)
```

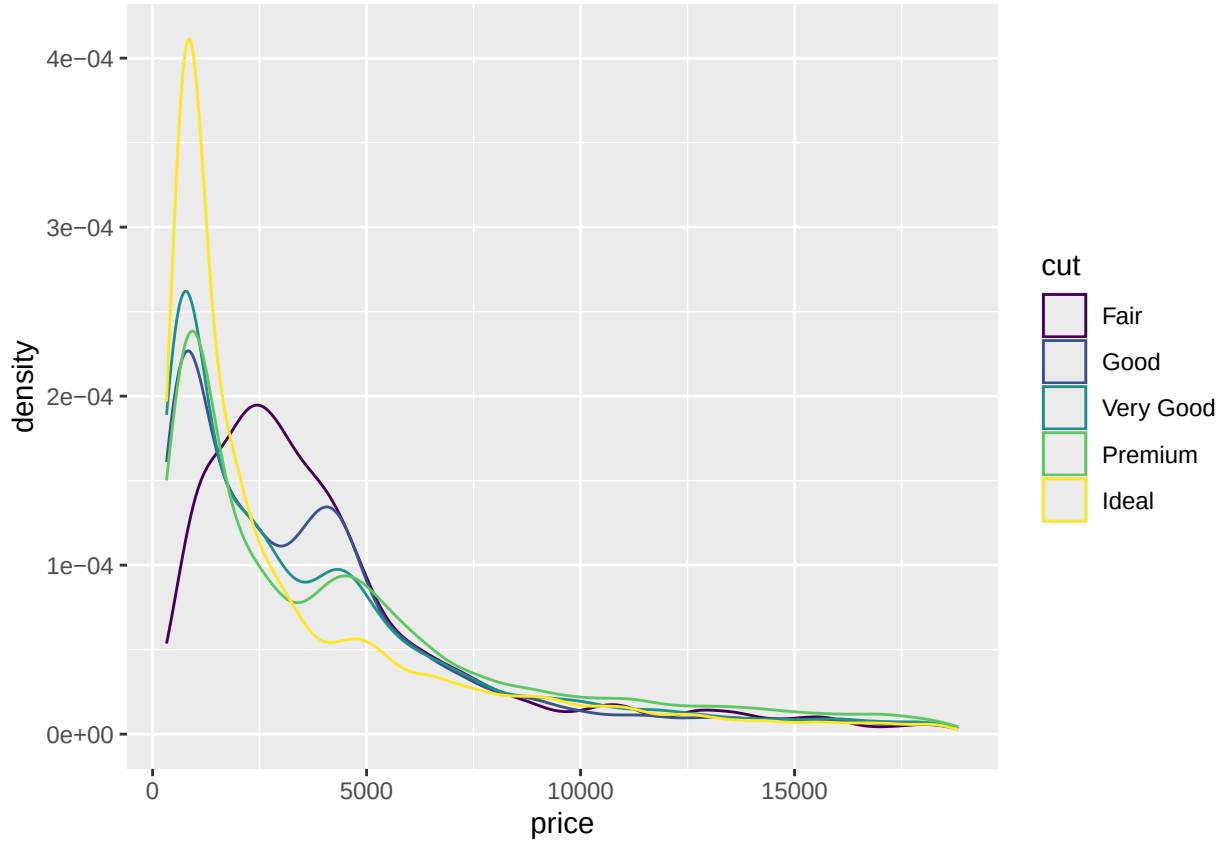
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




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




```
ggplot(diamonds, aes(x = price)) +
  geom_density(aes(color = cut))
```



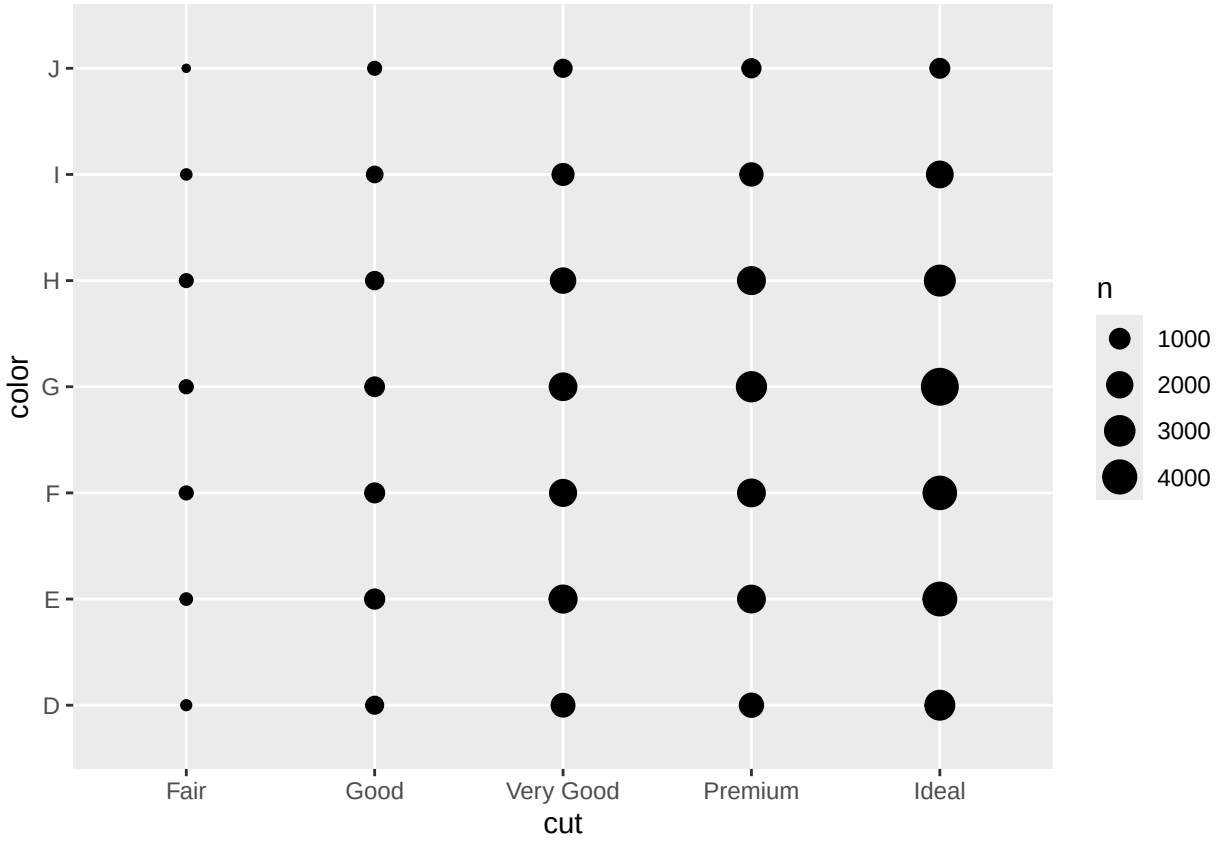
E6. If you have a small dataset, it's sometimes useful to use `geom_jitter()` to avoid overplotting to more easily see the relationship between a continuous and categorical variable. The `ggbeeswarm` package provides a number of methods similar to `geom_jitter()`. List them and briefly describe what each one does.

Ans. `geom_jitter()` a random noise to the data to avoid overplotting. In `ggbeeswarm`, commonly used in column-wise scatterplots arrange the data points in such a way that a bee-swarm-like pattern appears. This is also done to avoid overplotting.

Visualizing two categorical variables

- using `geom_count()`:

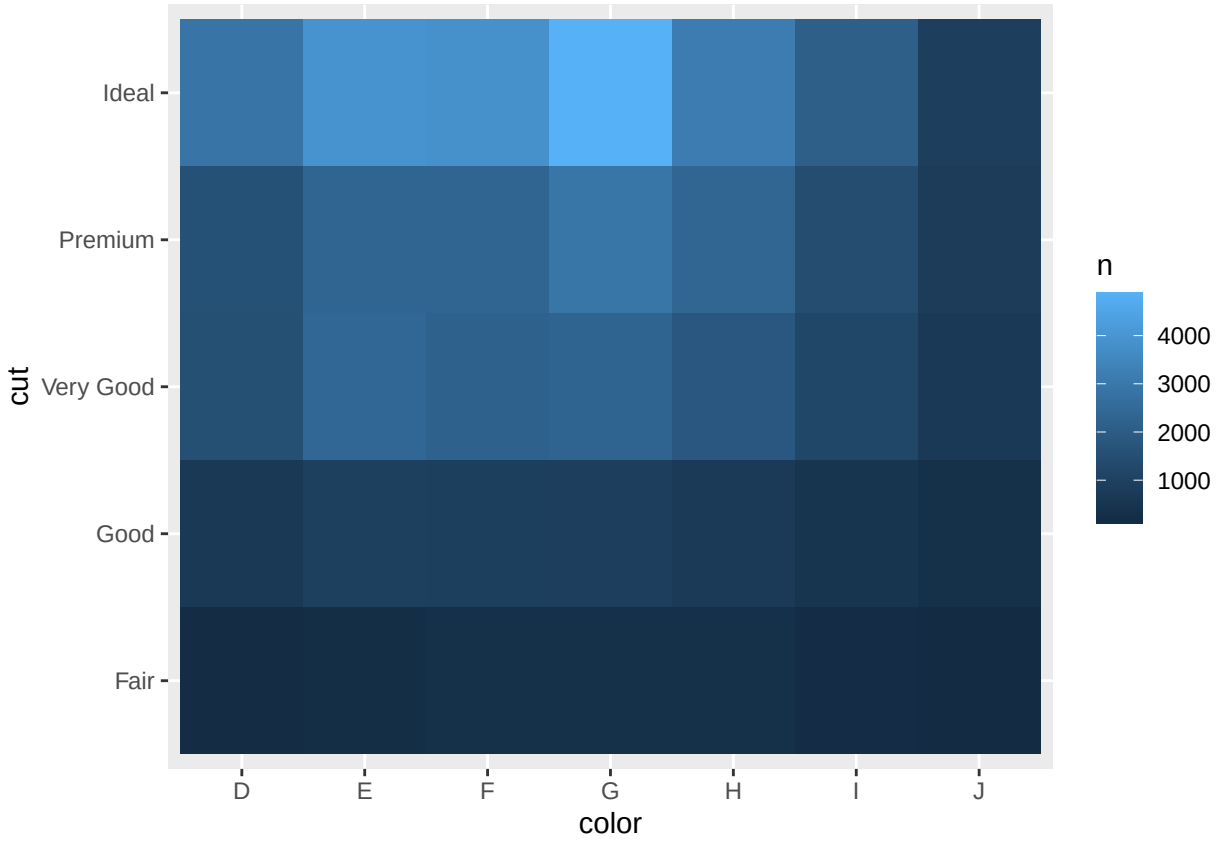
```
ggplot(diamonds, aes(x = cut, y = color)) +
  geom_count()
```



- the size of the circles depends on the number of observations that occurred for each combination of values (cut and color)

Counting first the using `geom_tile()`:

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



- This is a heatmap.

Exercises

E1. How could you rescale the count dataset above to more clearly show the distribution of cut within color, or color within cut?

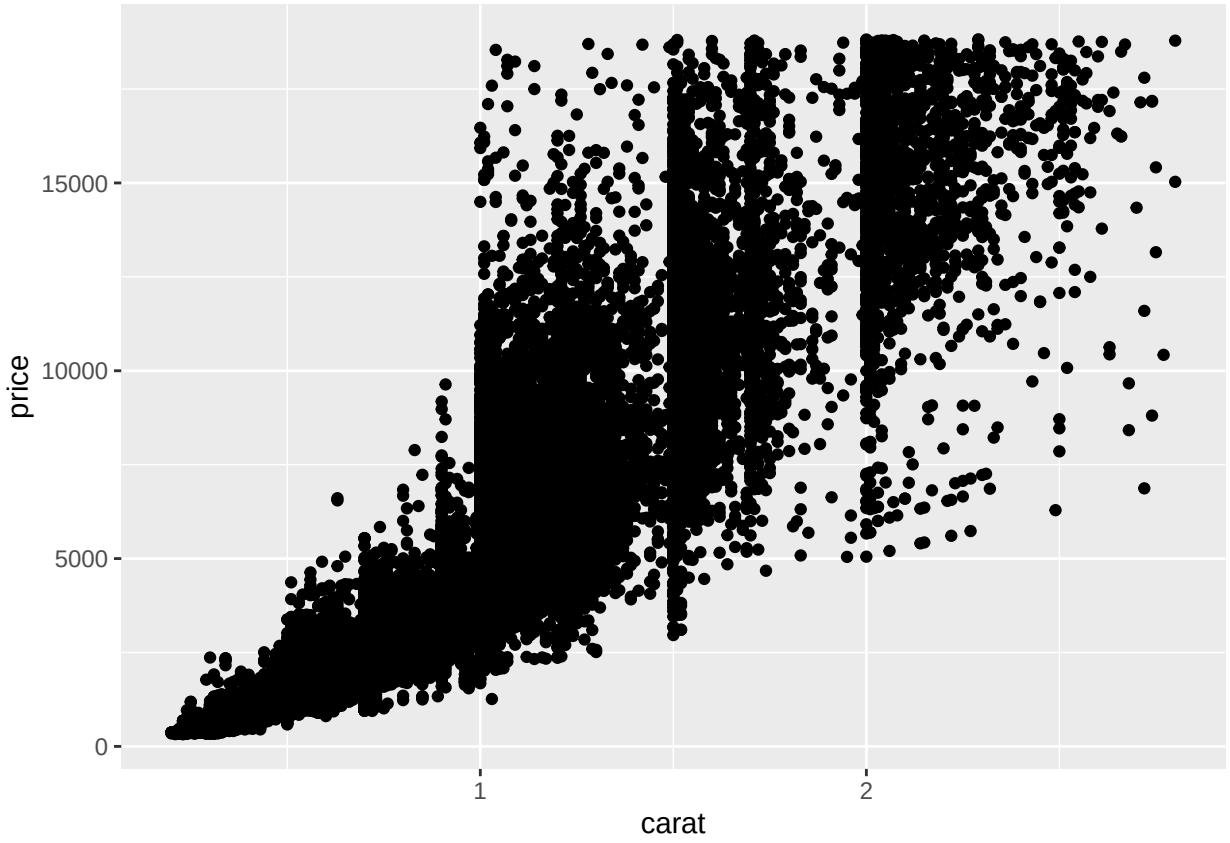
E2. What different data insights do you get with a segmented bar chart if color is mapped to the x aesthetic and cut is mapped to the fill aesthetic? Calculate the counts that fall into each of the segments.

E3. Use `geom_tile()` together with dplyr to explore how average flight departure delays vary by destination and month of year. What makes the plot difficult to read? How could you improve it?

Two numerical variables

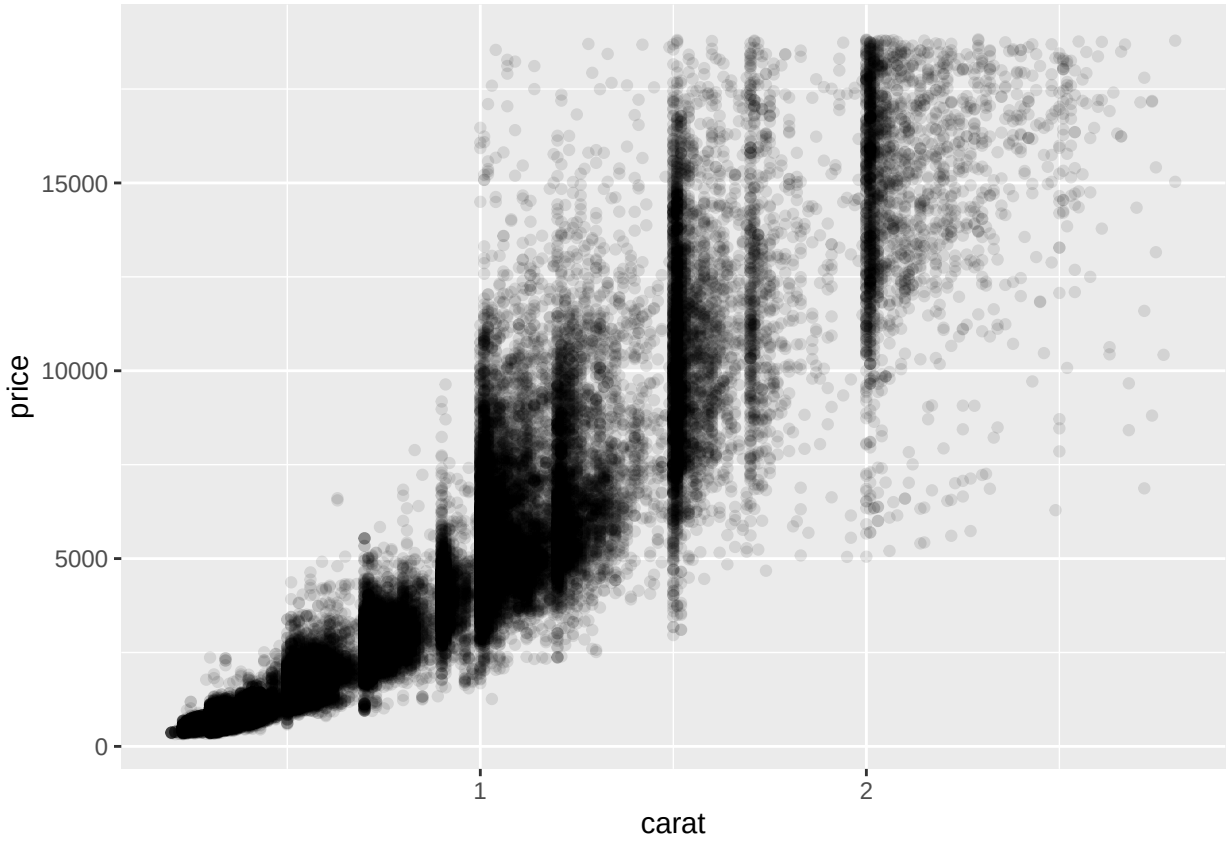
- with a scatterplot:

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



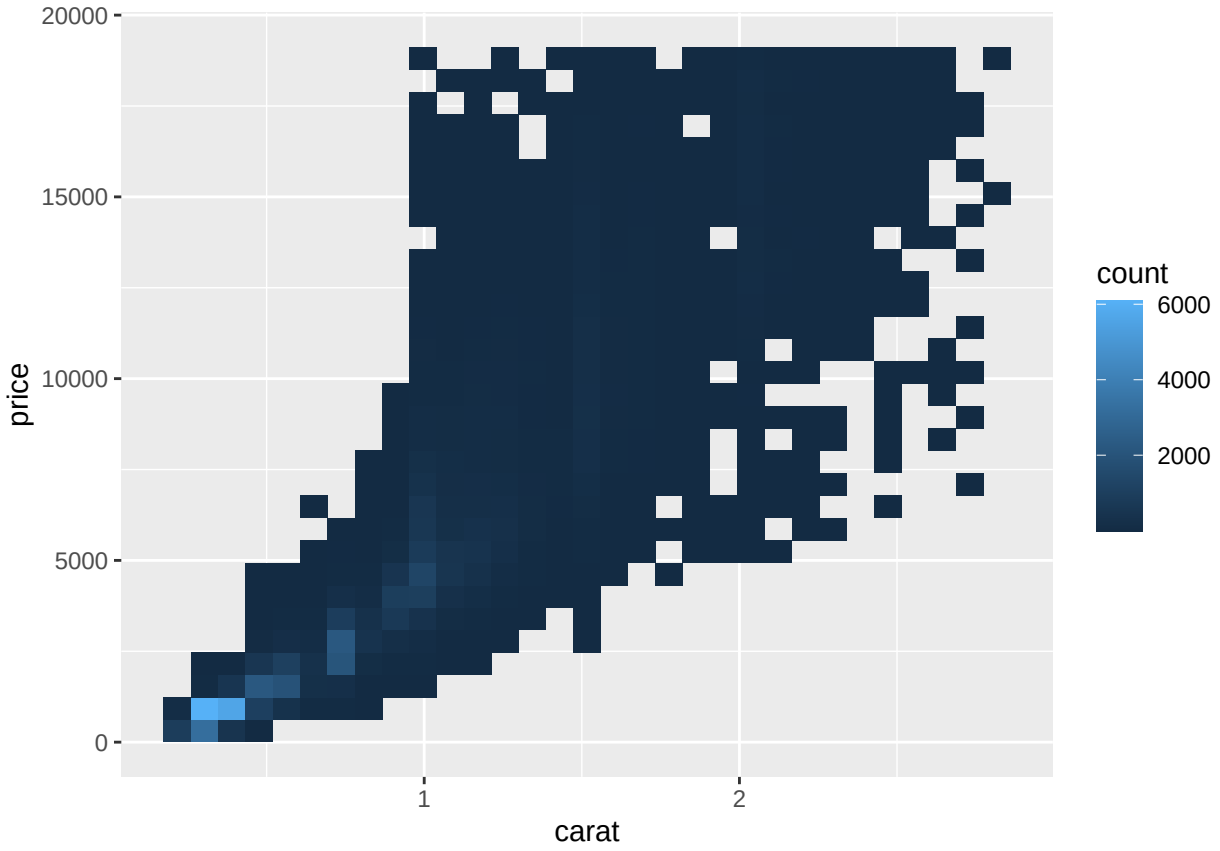
- scatterplot with transparency (alpha):

```
ggplot(smaller, aes(x = carat, y = price)) +
  geom_point(alpha = 0.1)
```



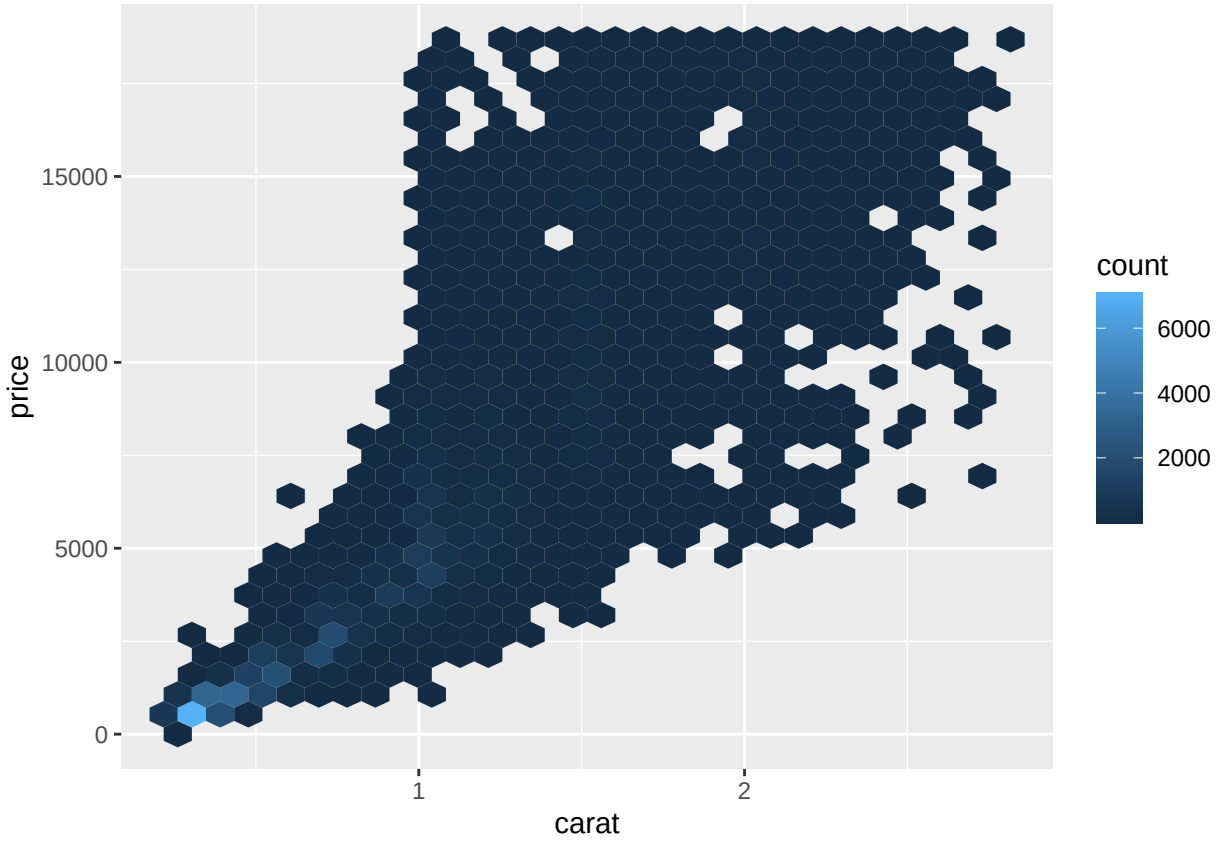
- using `geom_bin2d()`

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



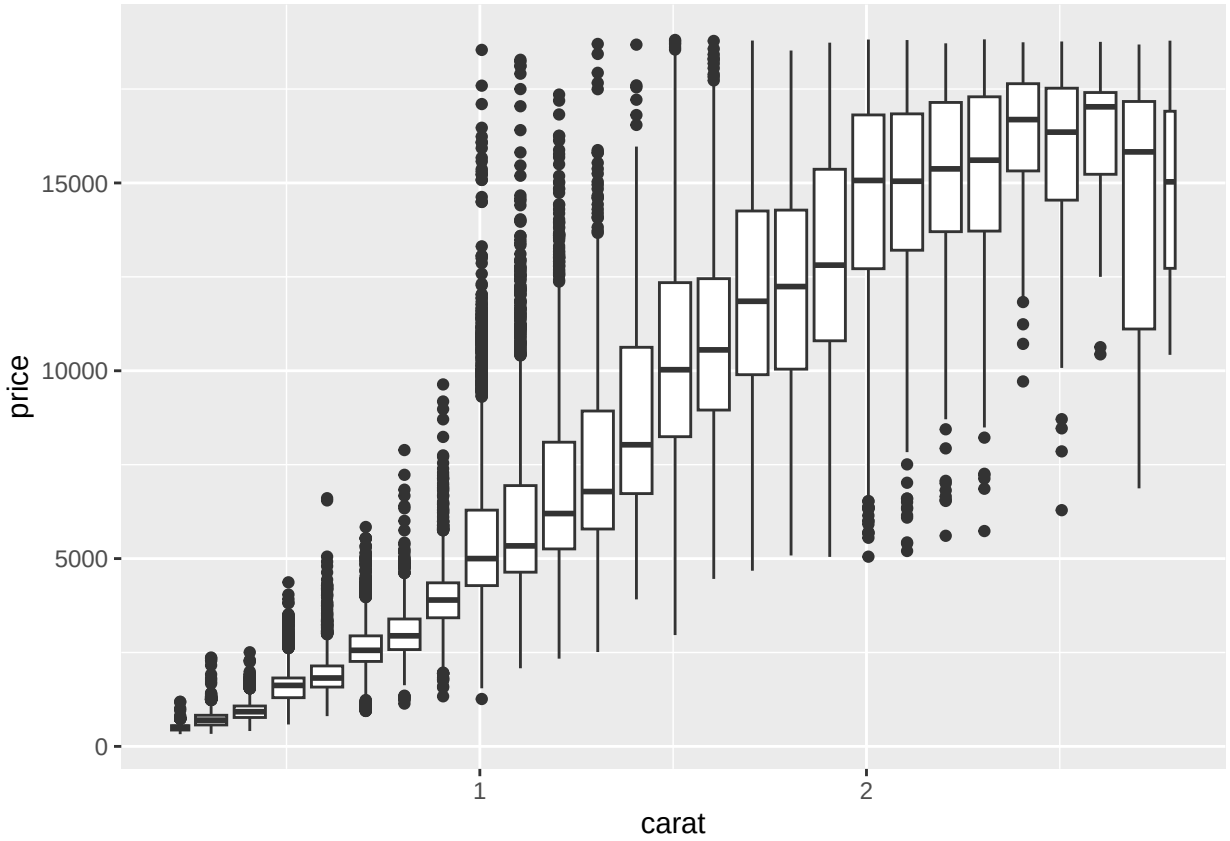
- using `geom_hex()`:

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



- binning one continuous variable so it acts like a categorical variable:

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



The function `cut_width(x, width)` divides `x` into bins of width `width`.

Exercises

E1. Instead of summarizing the conditional distribution with a boxplot, you could use a frequency polygon. What do you need to consider when using cut_width() vs. cut_number()? How does that impact a visualization of the 2d distribution of carat and price?

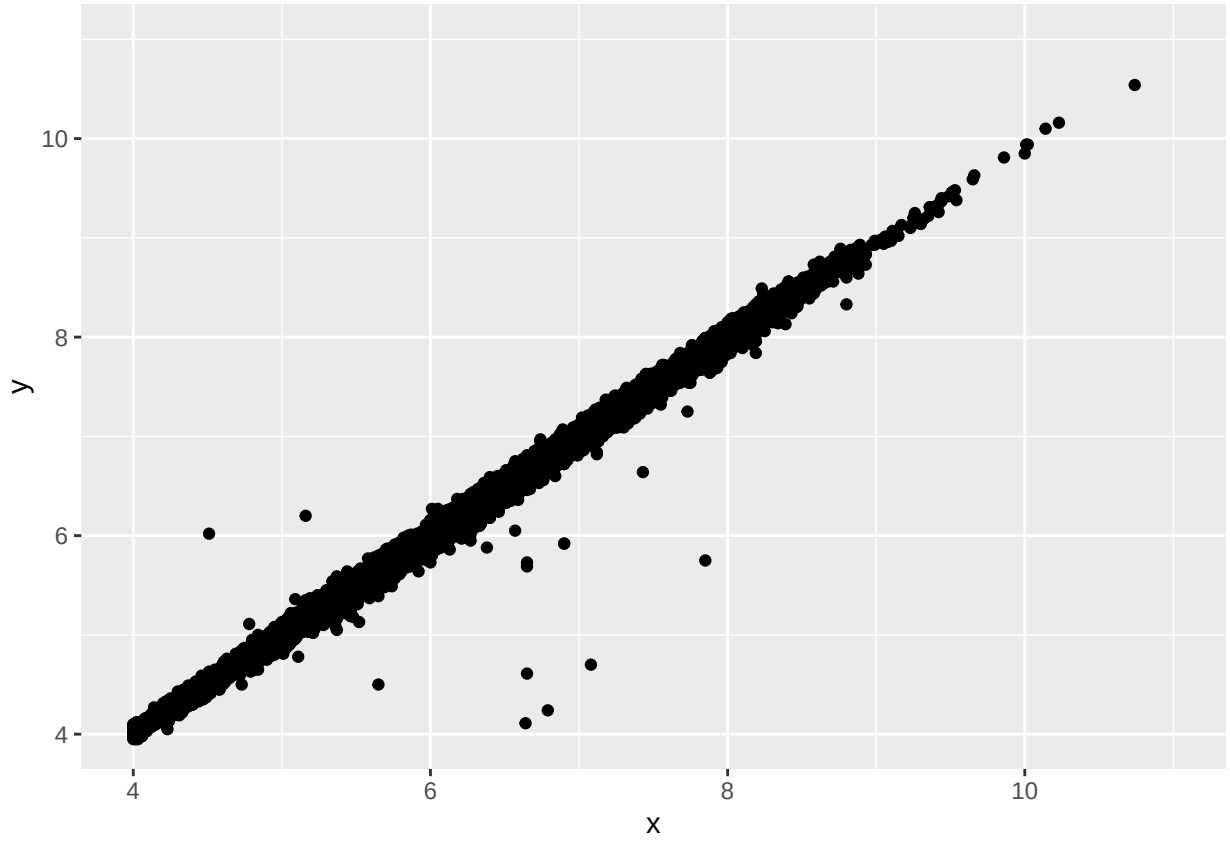
E2. Visualize the distribution of carat, partitioned by price.

E3. How does the price distribution of very large diamonds compare to small diamonds? Is it as you expect, or does it surprise you?

E4. Combine two of the techniques you've learned to visualize the combined distribution of cut, carat, and price.

E5. Two dimensional plots reveal outliers that are not visible in one dimensional plots. For example, some points in the following plot have an unusual combination of x and y values, which makes the points outliers even though their x and y values appear normal when examined separately. Why is a scatterplot a better display than a binned plot for this case?

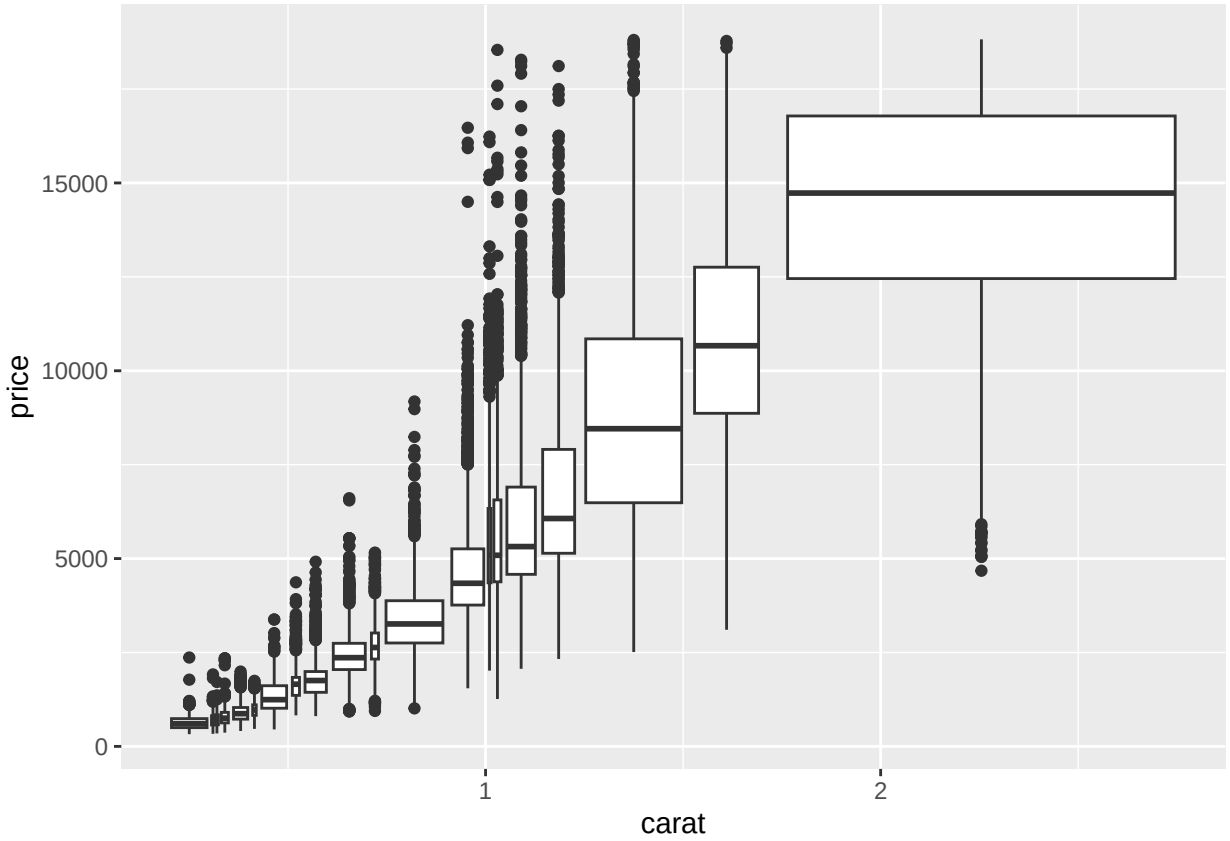
```
diamonds |>
  filter(x >= 4) |>
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  coord_cartesian(xlim = c(4, 11), ylim = c(4, 11))
```



Ans. With a scatterplot, it is easier to visually compare one dot against the rest of dots. Binning will take away some of this feature, because by doing so, we will be comparing one data point against the contents of the bin it happens to belong to. This is also useful, but less "powerful" when used to visually inspect the data for outliers.

E6. Instead of creating boxes of equal width with cut_width(), we could create boxes that contain roughly equal number of points with cut_number(). What are the advantages and disadvantages of this approach?

```
ggplot(smaller, aes(x = carat, y = price)) +
  geom_boxplot(aes(group = cut_number(carat, 20)))
```



Ans. `cut_number()`, cuts the vector into intervals containing equal number of points. Depending on the distribution of data, this can end up line the graph above, where the width of each boxplot is not similar.

Patterns and models

Ask: - Could the pattern be due to coincidence? - How can you describe the relationship implied by the pattern? - How strong is the relationship implied by the pattern? - Does the relationship change if you look at individual subgroups of the data?

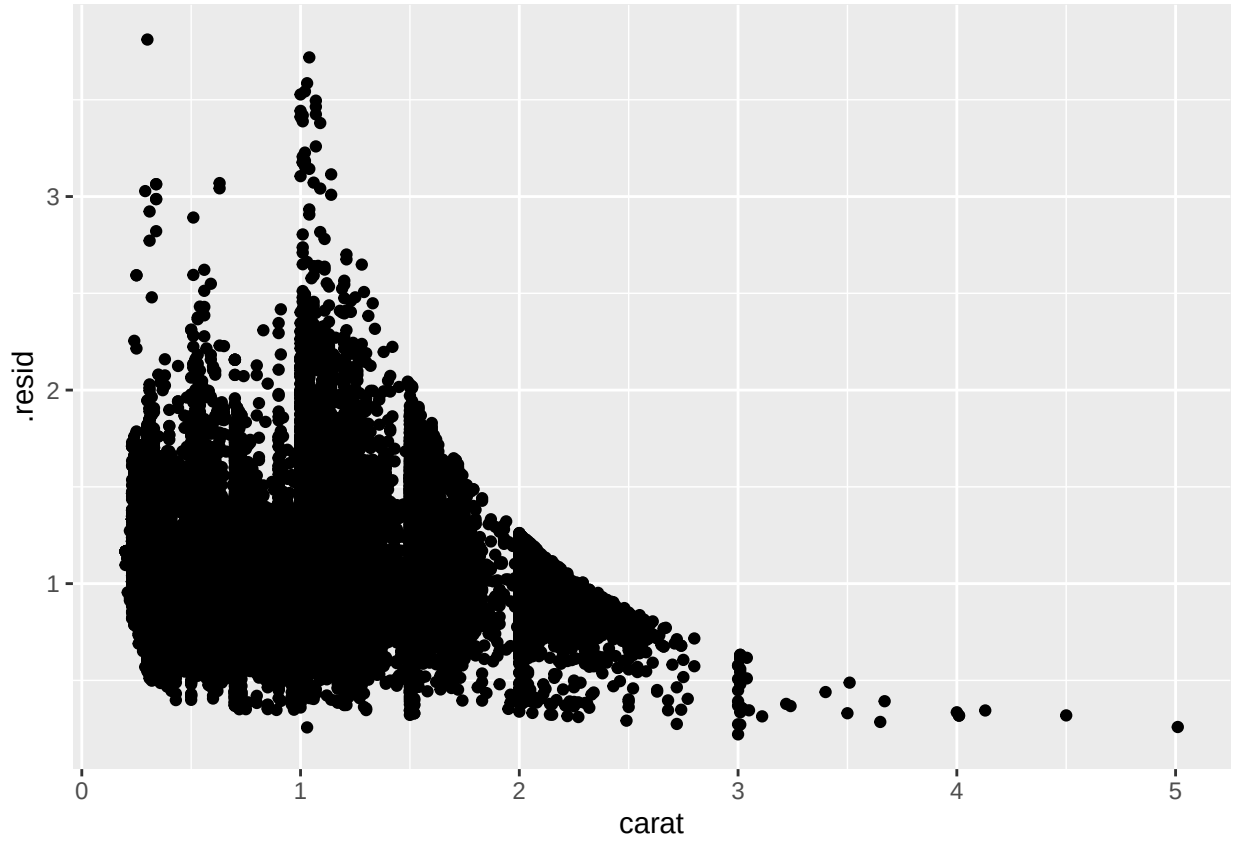
Modeling Example Fitting a model that predicts `price` from `carat` and then computes the residuals aka the difference between the predicted value and the actual value:

```
library(tidymodels)
```

```
## -- Attaching packages --------------------------------------- tidymodels 1.3.0 --
## v broom        1.0.8     v rsample      1.2.1
## v dials        1.4.0     v tune         1.3.0
## v infer        1.0.8     v workflows    1.2.0
## v modeldata    1.4.0     v workflowsets 1.1.1
## v parsnip      1.3.2     v yardstick    1.3.2
## v recipes      1.3.1
## -- 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()
```

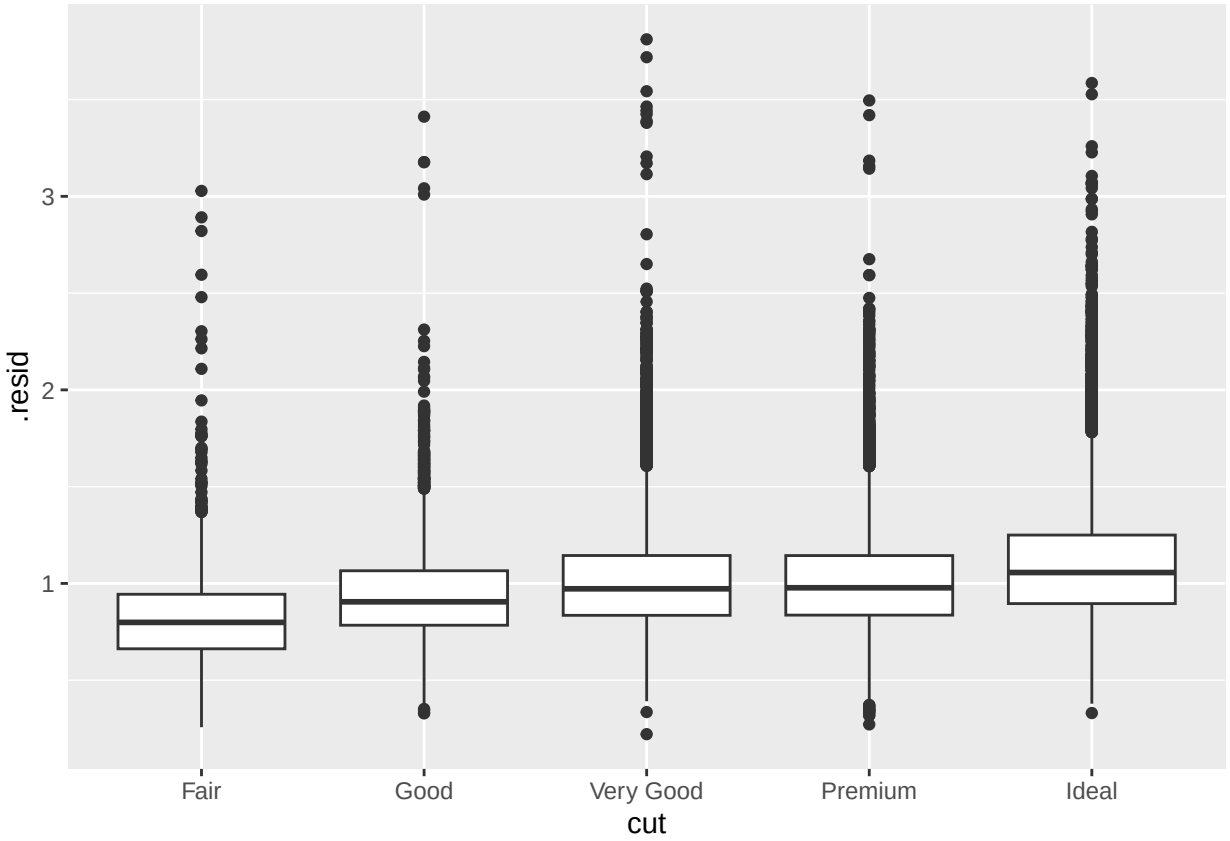
```
## x recipes::step()   masks stats::step()
```

```
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()
```



The relationship between cut and price relative to size can be visualized with this boxplot:

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


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