Exploratory Data Analysis

Kylie Ariel Bemis

1/23/2018

Introduction to Exploratory Data Analysis

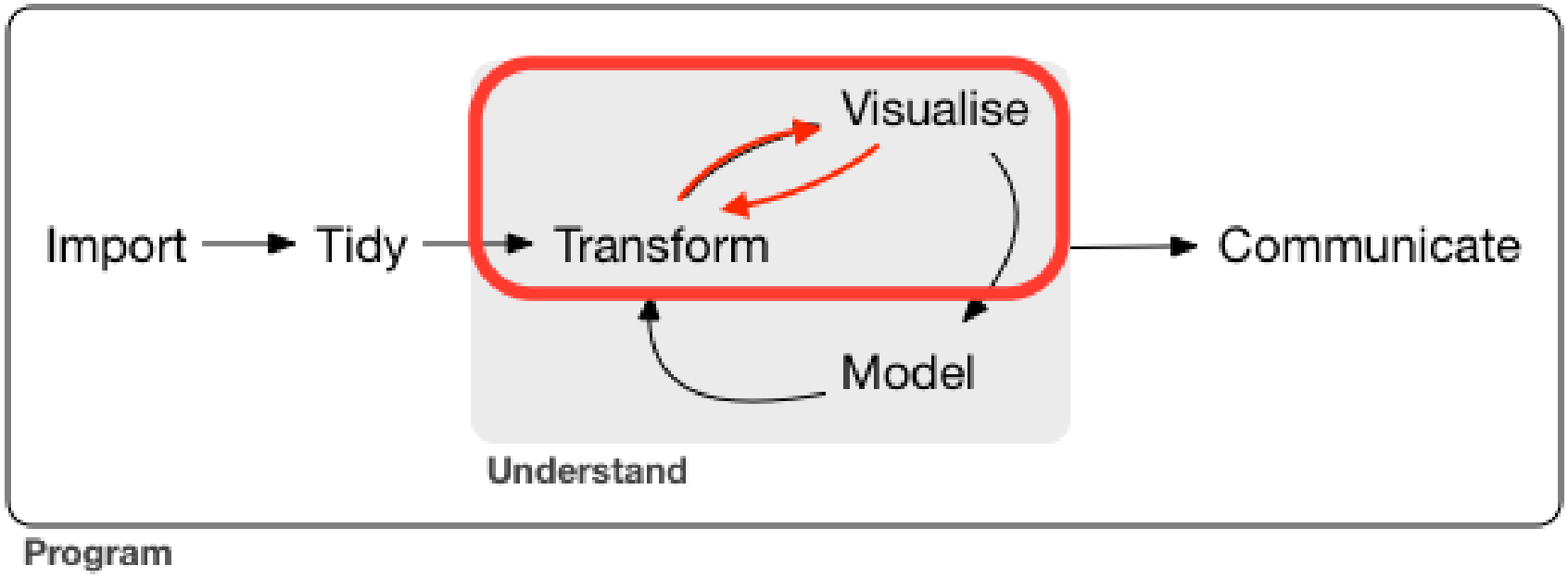


Figure 1: Wickham and Grolemund, *R for Data Science*

Introduction to Exploratory Data Analysis (cont'd)

Exploratory Data Analysis is an iterative, cyclical process of informally investigating the variables and relationships between them in a dataset.

- Generate questions about your data.
- Search for answers by transforming and visualising your data.
- Use this to refine your questions and/or generate new questions.

Modeling may be a part of this process, but can also be performed later as part of a more formal approach to answer rigorous scientific questions.

Questions

- What type of variation occurs within my variables?
- What type of covariation occurs between my variables?

Definitions

- A variable is a quantity, quality, or property that is measured
- A value is the state of a variable when you measure it
- An observation is a set of measurements made under similar condition

Example dataset: Diamonds

```
library(tidyverse)
```

```
diamonds
```

```
## # A tibble: 53,940 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  4.20  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
##  7  0.24 Very Good     I    VVS1  62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good     H     SI1  61.9    55   337  4.07  4.11  2.53
##  9  0.22      Fair     E     VS2  65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good     H     VS1  59.4    61   338  4.00  4.05  2.39
## # ... with 53,930 more rows
```

A note on R functions and code style

Consider the following ggplot2 code:

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

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

Both do the same thing.

R functions and formal arguments

```
foo(a = 1, b = 2, c = 3)
```

- All arguments in R functions may be named
- Unnamed arguments will be matched based on their order
- Named arguments may be used in any order
  - Be careful of ordering when mixing named and unnamed arguments!
- You may mix named arguments with unnamed arguments
- Arguments with defaults may be omitted
- Any arguments coming after `...` must be named

When to name arguments?

This is personal opinion. These are only guidelines!

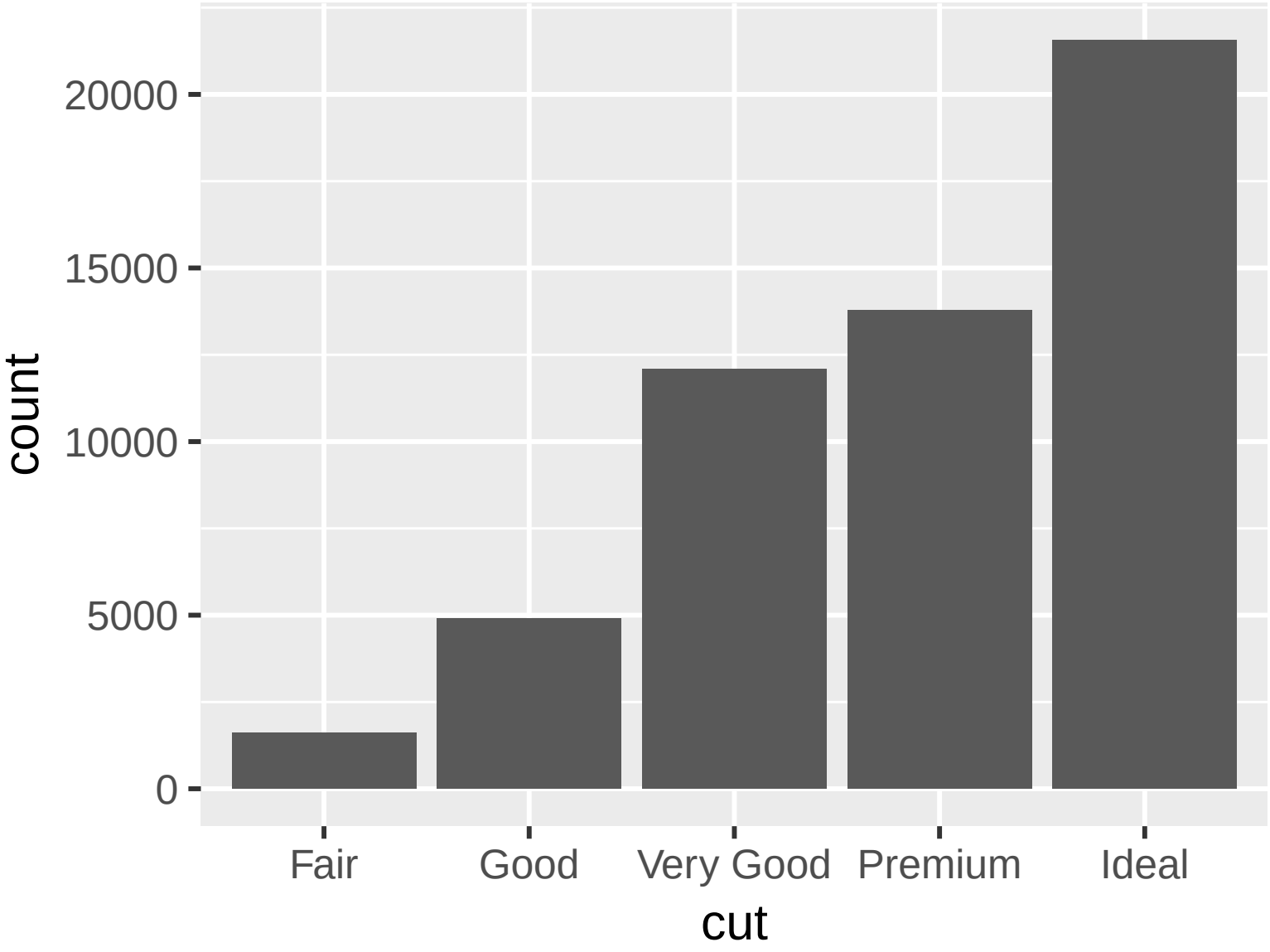
- The first argument to a function typically need not be named
- Arguments after the first argument should be named
- Arguments that "go together" should both be named or neither should be named
  - E.g. either `aes(x = a, y = b)` or `aes(a, b)`

Variation

- Variation is the tendency of values of variables to change from measurement to measurement
  - Natural variation
  - Measurement error
  - Change between subjects
  - Change over time
- Understand by visualizing the distribution of individual variables

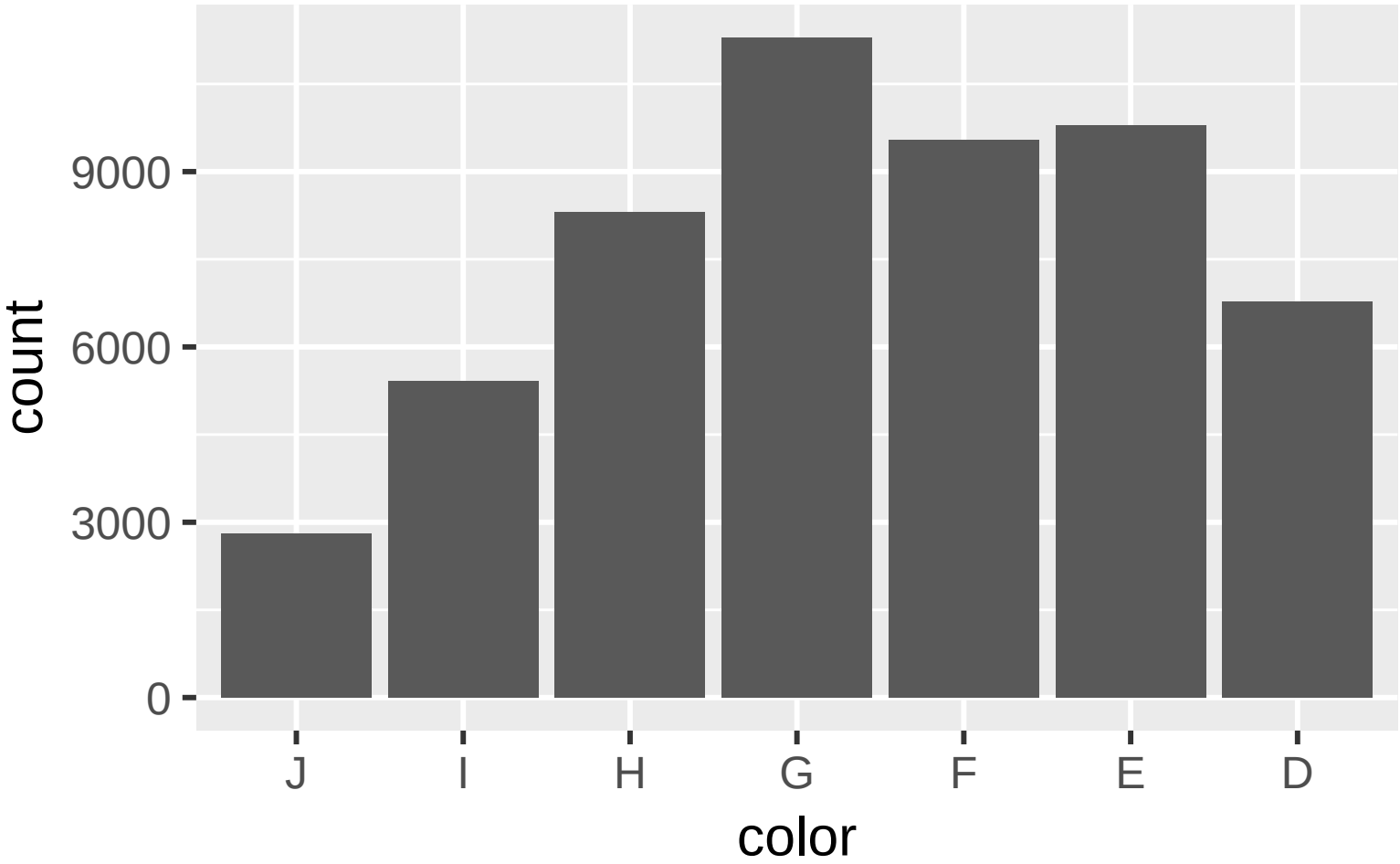
Visualizing distributions: categorical

```
ggplot(diamonds) + geom_bar(aes(x = cut))
```



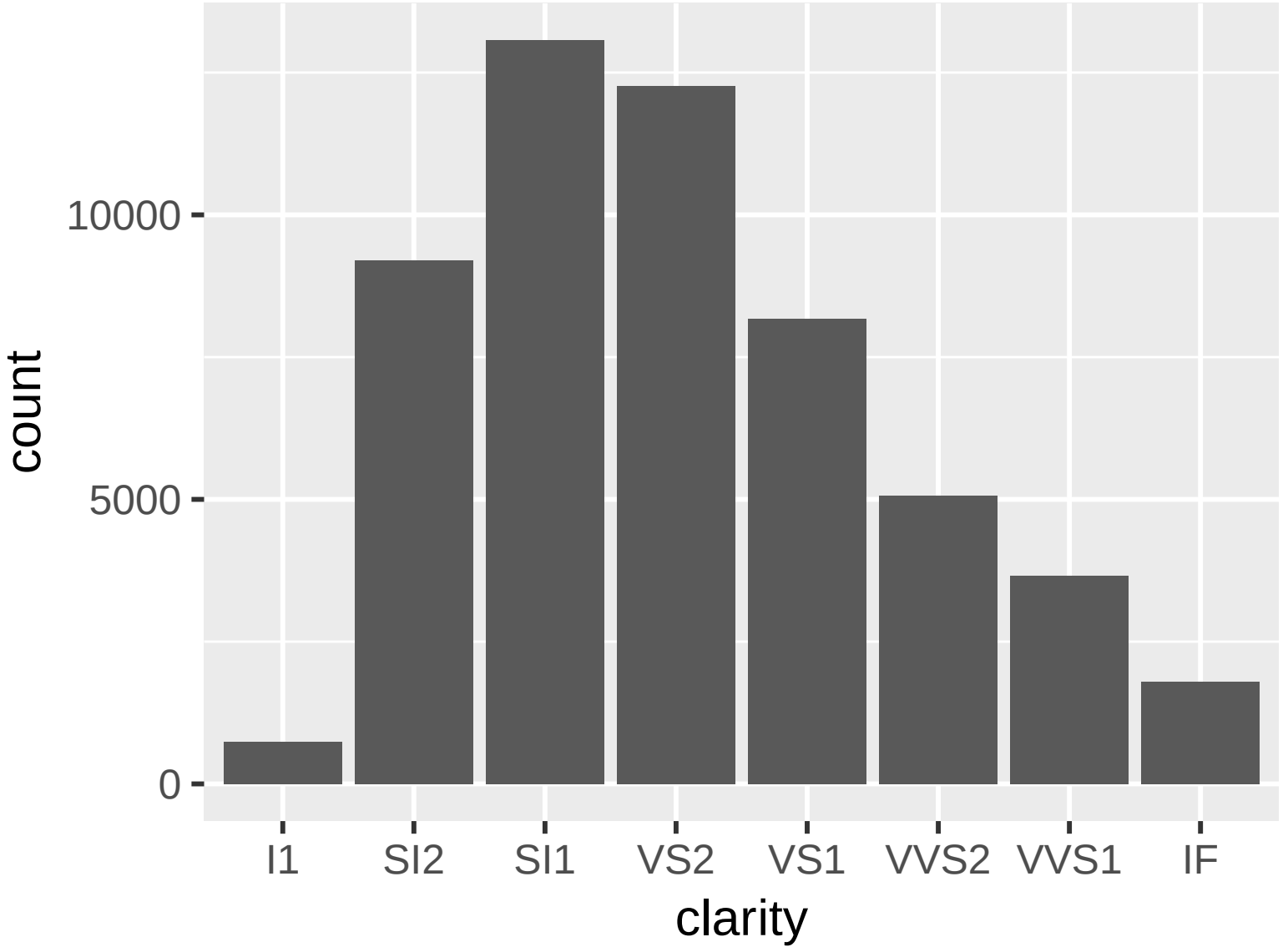
Visualizing distributions: categorical

```
diamonds <- mutate(diamonds,
                   color=ordered(color,
                                 levels=rev(levels(color))))
ggplot(diamonds) + geom_bar(aes(x = color))
```



Visualizing distributions: categorical

```
ggplot(diamonds) + geom_bar(aes(x = clarity))
```



Visualizing distributions: categorical

Counts can also be calculated manually with `summarise()` or `count()`

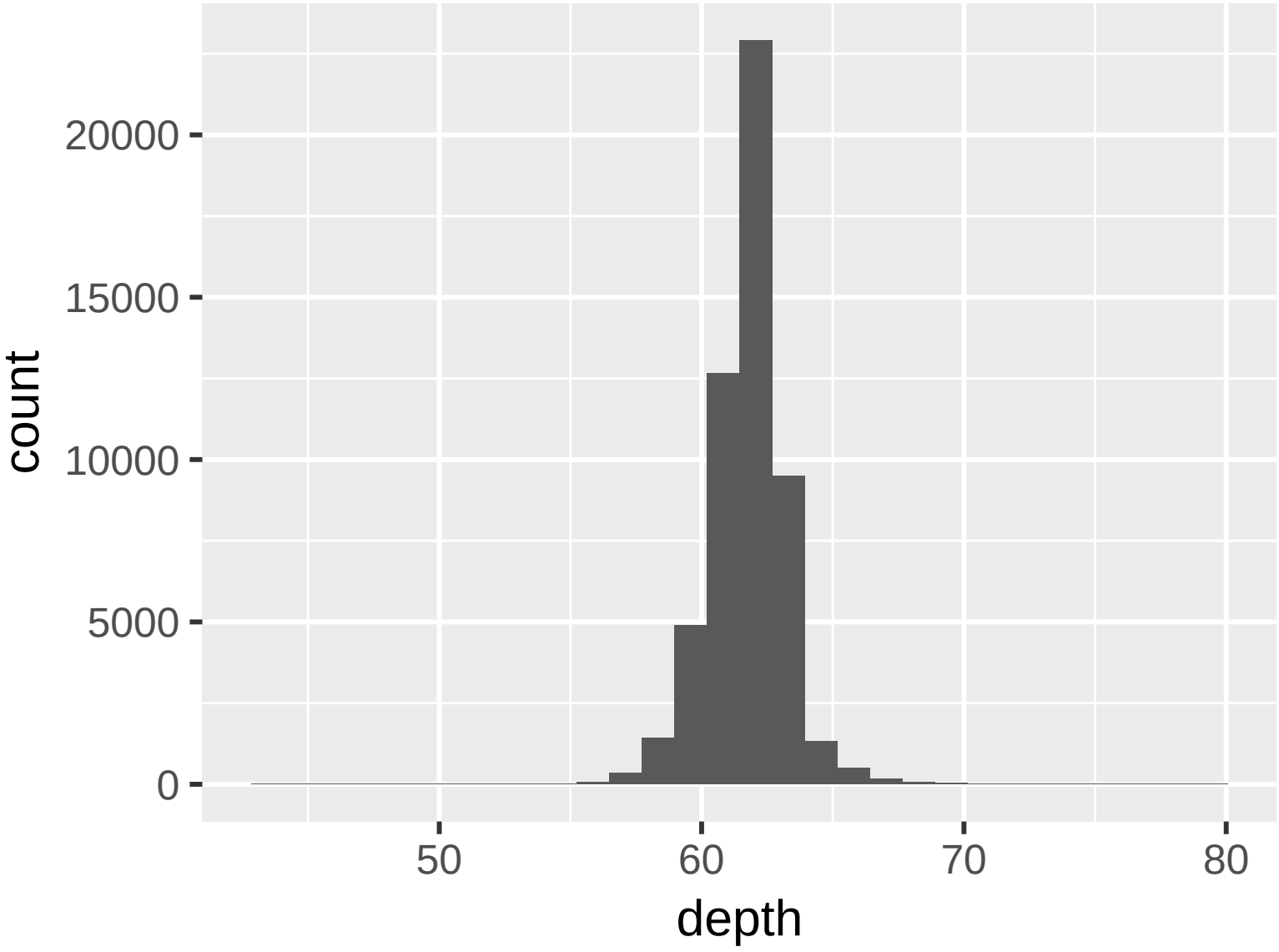
```
diamonds %>% group_by(cut) %>% summarise(n=n())
```

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

```
## # A tibble: 5 x 2
##         cut     n
##       <ord> <int>
## 1      Fair  1610
## 2      Good  4906
## 3 Very Good 12082
## 4   Premium 13791
## 5     Ideal 21551
```

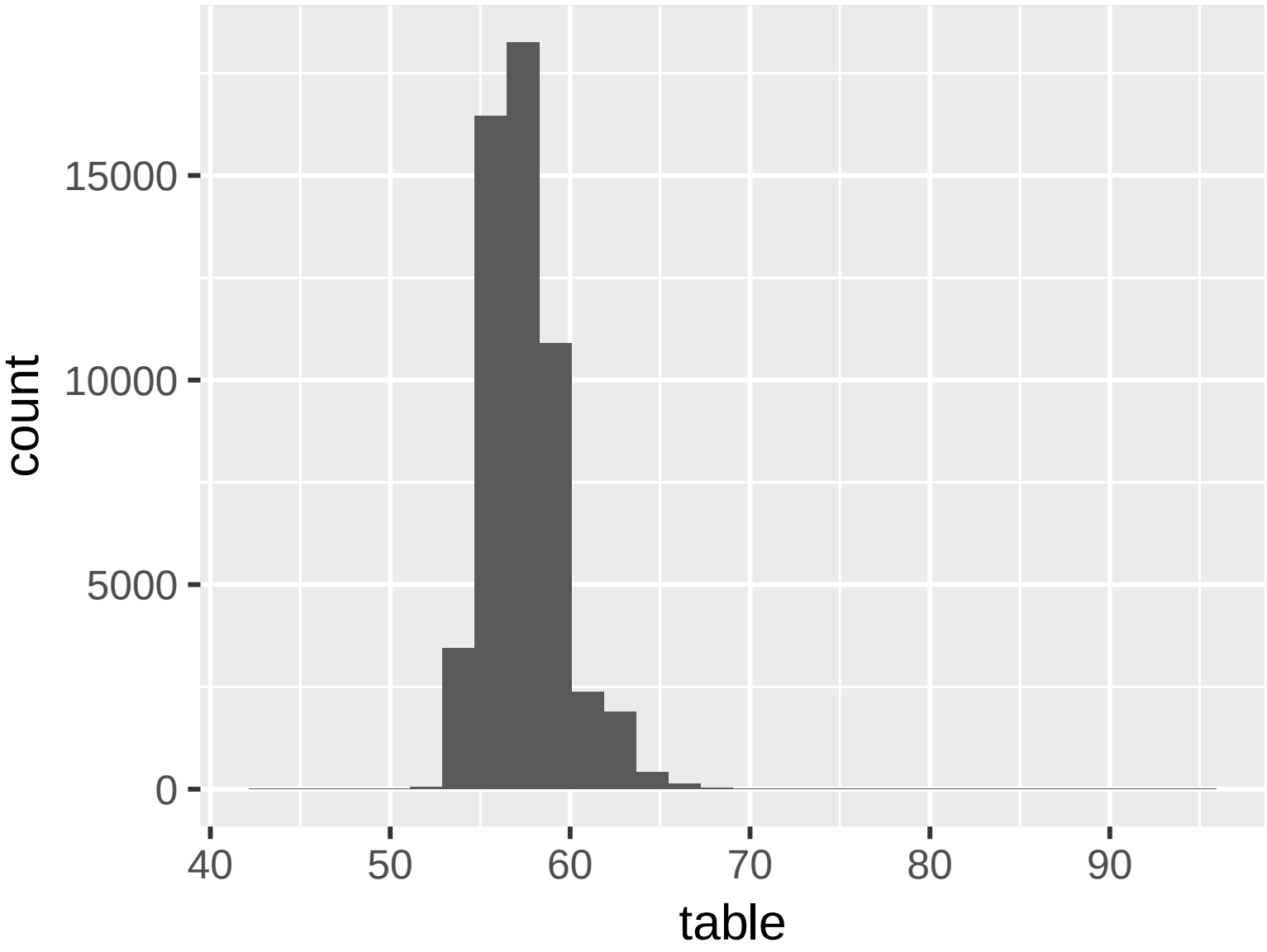
Visualizing distributions: continuous

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



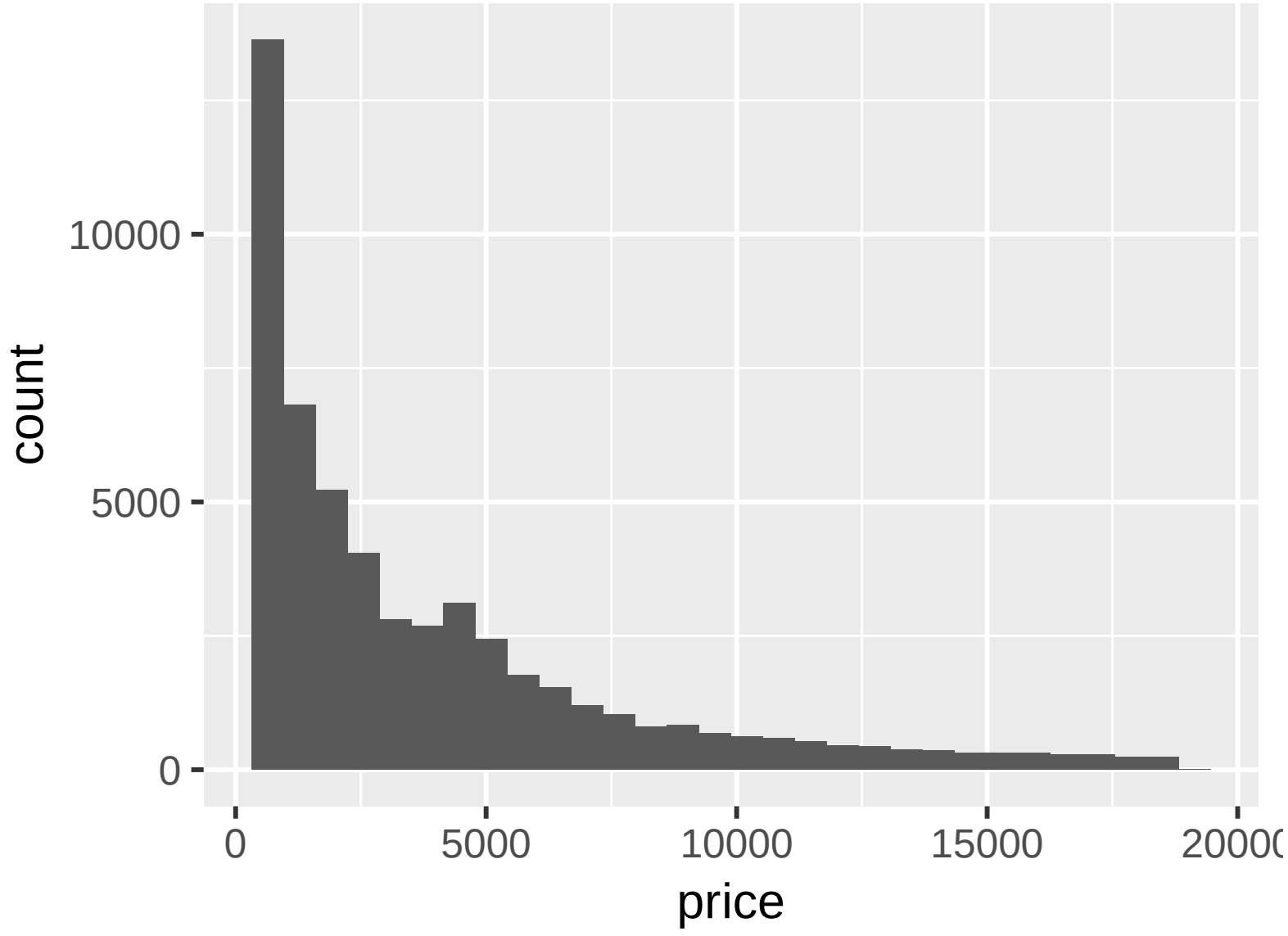
Visualizing distributions: continuous

```
ggplot(diamonds) + geom_histogram(aes(x = table))
```



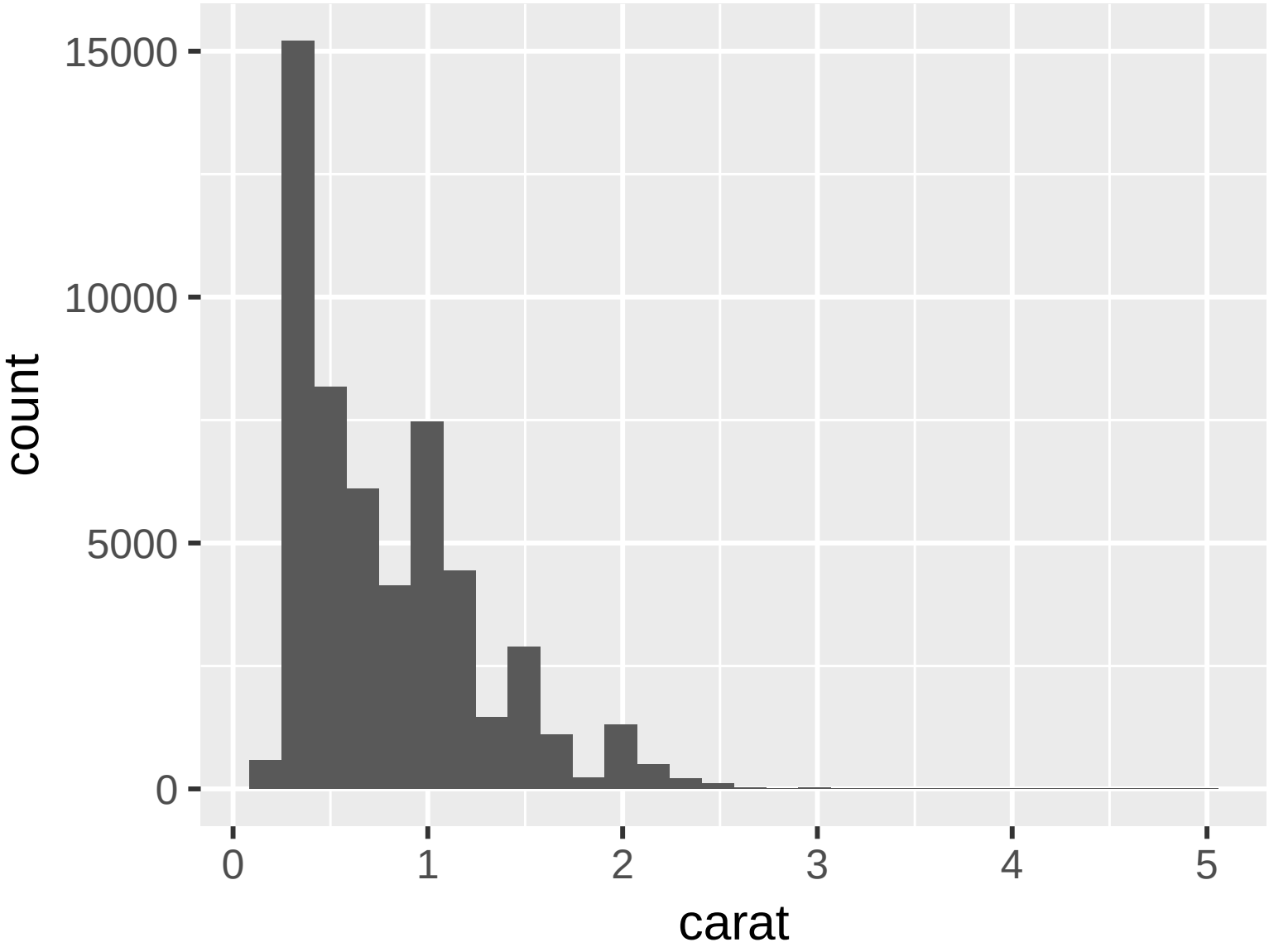
Visualizing distributions: continuous

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



Visualizing distributions: continuous

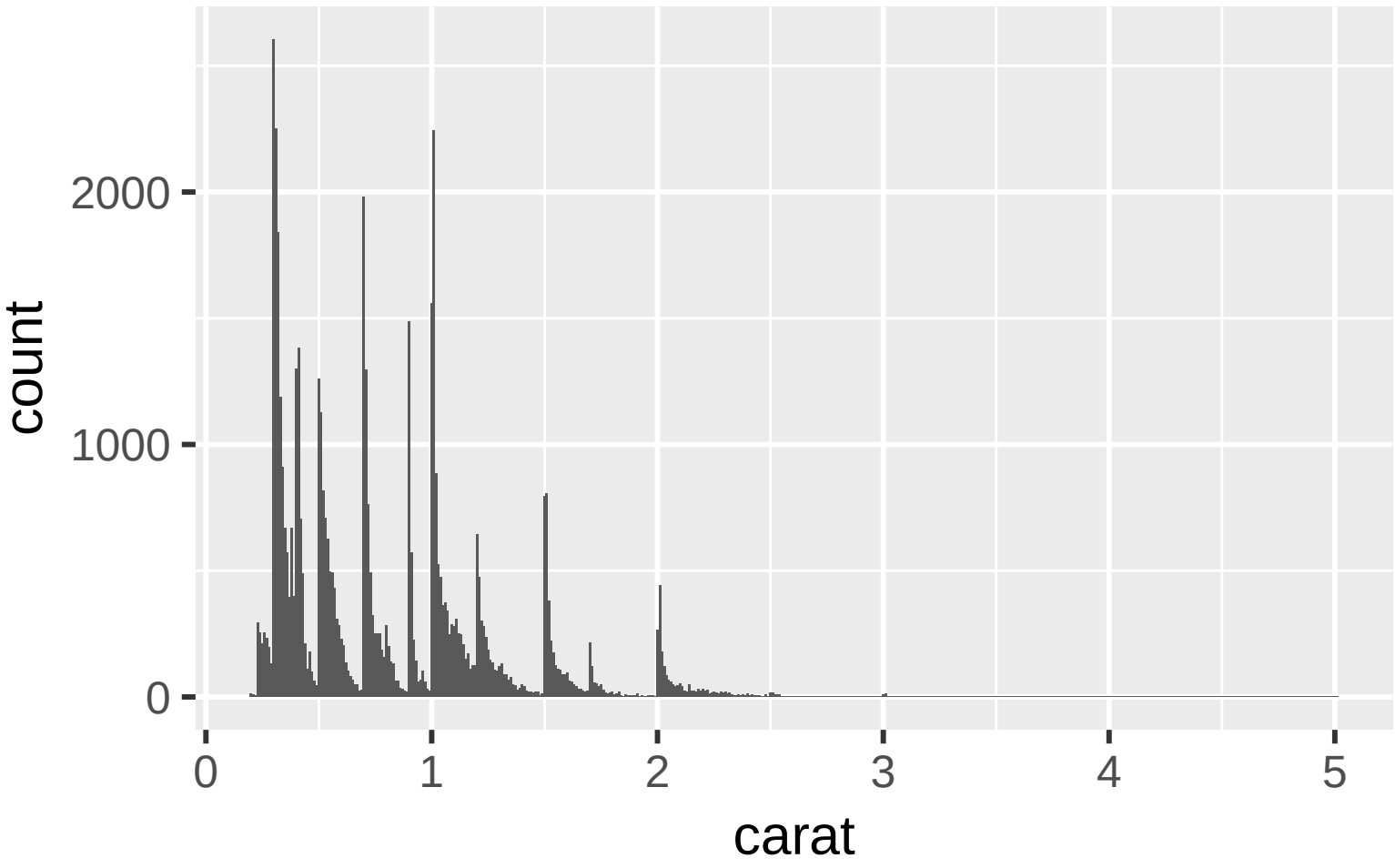
```
ggplot(diamonds) + geom_histogram(aes(x = carat))
```



Visualizing distributions: continuous

Use different bin widths to investigate further.

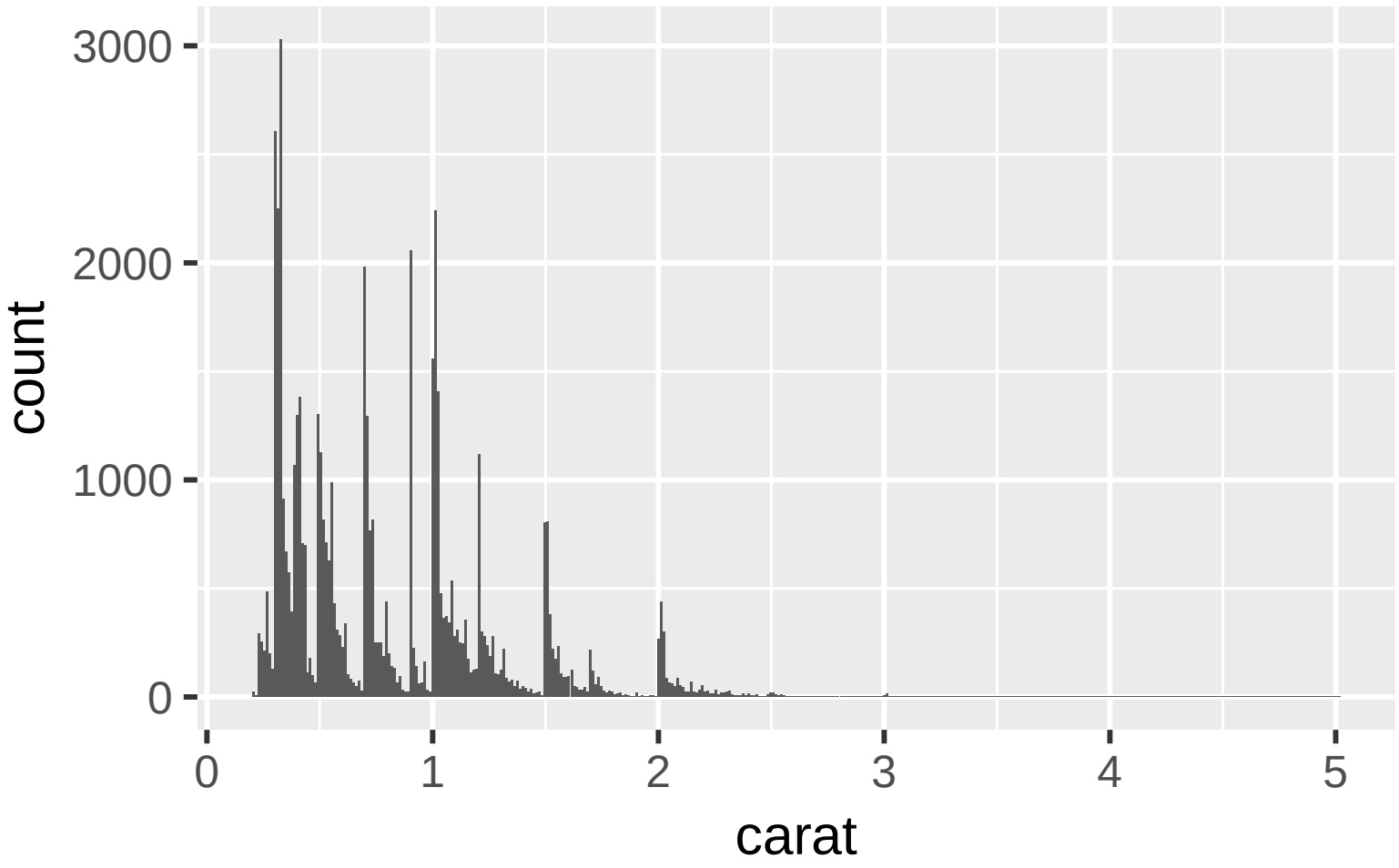
```
ggplot(diamonds) + geom_histogram(aes(x = carat),
                                  binwidth = 0.01)
```



Visualizing distributions: continuous

Alternatively, set the number of bins.

```
ggplot(diamonds) + geom_histogram(aes(x = carat),
                                  bins = 400)
```



Visualizing distributions: continuous

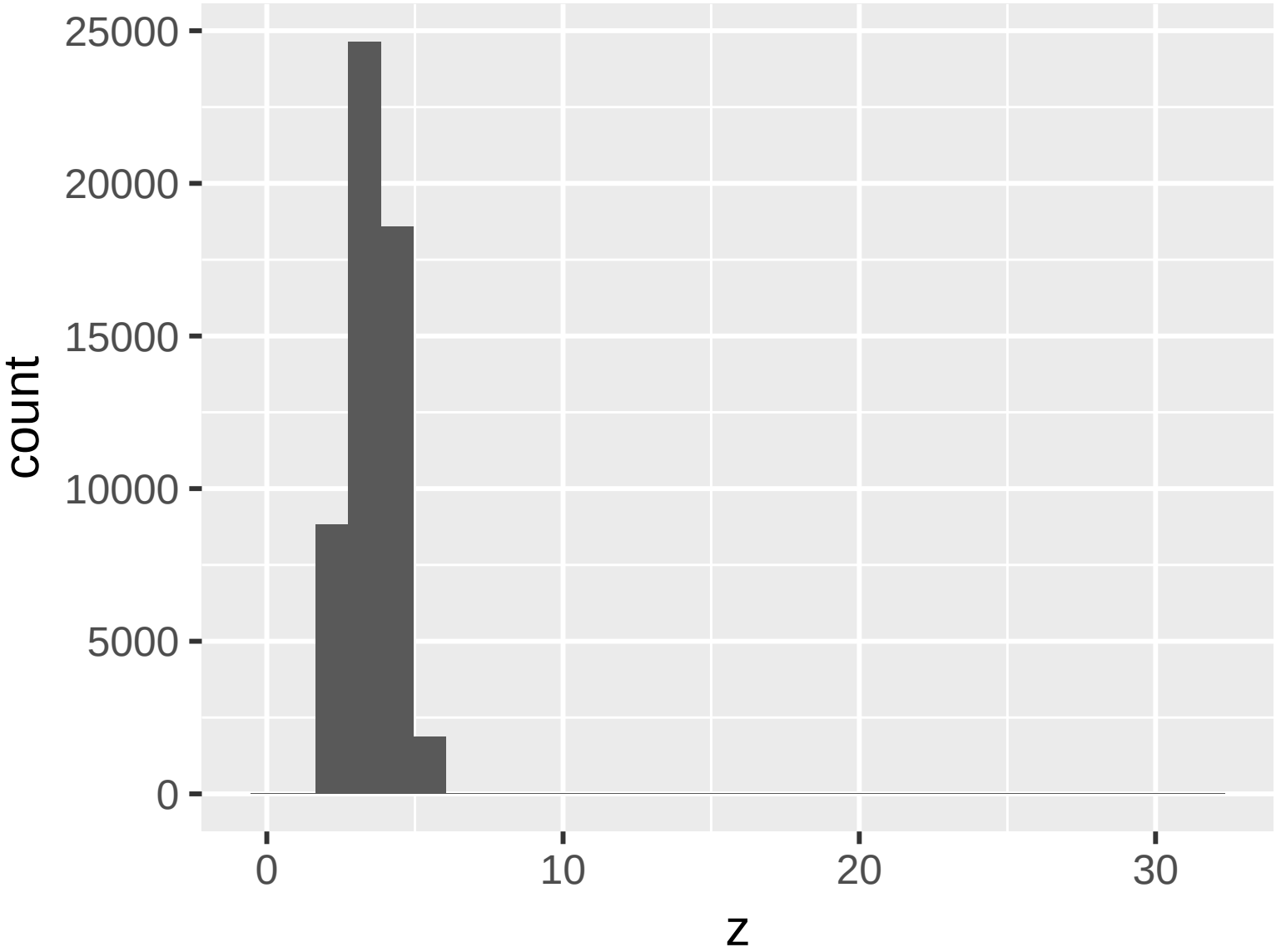
Bins can also be calculated manually with `cut_interval()` or `cut_width()`

```
diamonds %>% count(cut_interval(carat, n=10))
```

```
## # A tibble: 10 x 2
##    `cut_interval(carat, n = 10)`     n
##                           <fctr> <int>
##  1                  [0.2,0.681] 25155
##  2                 (0.681,1.16] 18626
##  3                  (1.16,1.64]  7129
##  4                  (1.64,2.12]  2349
##  5                   (2.12,2.6]   614
##  6                   (2.6,3.09]    53
##  7                  (3.09,3.57]     6
##  8                  (3.57,4.05]     5
##  9                  (4.05,4.53]     2
## 10                  (4.53,5.01]     1
```

Outliers

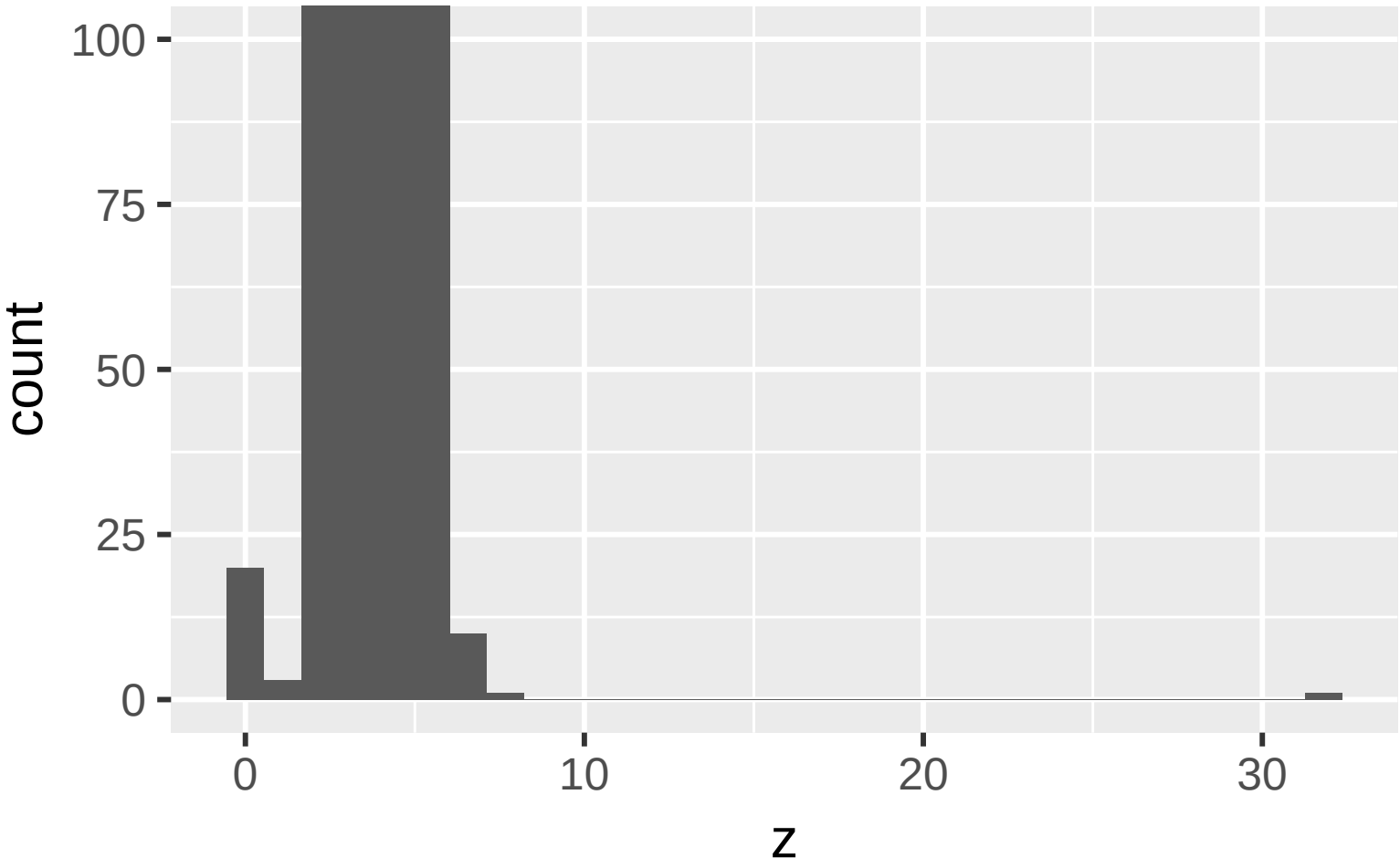
```
ggplot(diamonds) + geom_histogram(aes(x = z))
```



Outliers

Zoom in on y-axis to see the outliers (x = 0 and x > 30).

```
ggplot(diamonds) + geom_histogram(aes(x = z)) +
  coord_cartesian(ylim=c(0, 100))
```



Outliers

```
diamonds %>%
  select(carat:price, z) %>%
  arrange(desc(z))
```

```
## # A tibble: 53,940 x 8
##    carat       cut color clarity depth table price     z
##    <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl>
##  1  0.51 Very Good     E     VS1  61.8  54.7  1970 31.80
##  2  2.00   Premium     H     SI2  58.9  57.0 12210  8.06
##  3  5.01      Fair     J      I1  65.5  59.0 18018  6.98
##  4  4.50      Fair     J      I1  65.8  58.0 18531  6.72
##  5  4.13      Fair     H      I1  64.8  61.0 17329  6.43
##  6  3.65      Fair     H      I1  67.1  53.0 11668  6.38
##  7  4.00 Very Good     I      I1  63.3  58.0 15984  6.31
##  8  3.40      Fair     D      I1  66.8  52.0 15964  6.27
##  9  4.01   Premium     J      I1  62.5  62.0 15223  6.24
## 10  4.01   Premium     I      I1  61.0  61.0 15223  6.17
## # ... with 53,930 more rows
```

Remove outliers?

```
filter(diamonds, z > 0, z < 30)
```

```
## # A tibble: 53,919 x 10
##    carat       cut color clarity depth table price     x
##    <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl
##  1  0.23     Ideal     E     SI2  61.5    55   326  3.95  3.9
##  2  0.21   Premium     E     SI1  59.8    61   326  3.89  3.8
##  3  0.23      Good     E     VS1  56.9    65   327  4.05  4.0
##  4  0.29   Premium     I     VS2  62.4    58   334  4.20  4.2
##  5  0.31      Good     J     SI2  63.3    58   335  4.34  4.3
##  6  0.24 Very Good     J    VVS2  62.8    57   336  3.94  3.9
##  7  0.24 Very Good     I    VVS1  62.3    57   336  3.95  3.9
##  8  0.26 Very Good     H     SI1  61.9    55   337  4.07  4.1
##  9  0.22      Fair     E     VS2  65.1    61   337  3.87  3.7
## 10  0.23 Very Good     H     VS1  59.4    61   338  4.00  4.0
## # ... with 53,909 more rows
```

Missing values?

```
mutate(diamonds, z=ifelse(z > 0 & z < 30, z, NA))
```

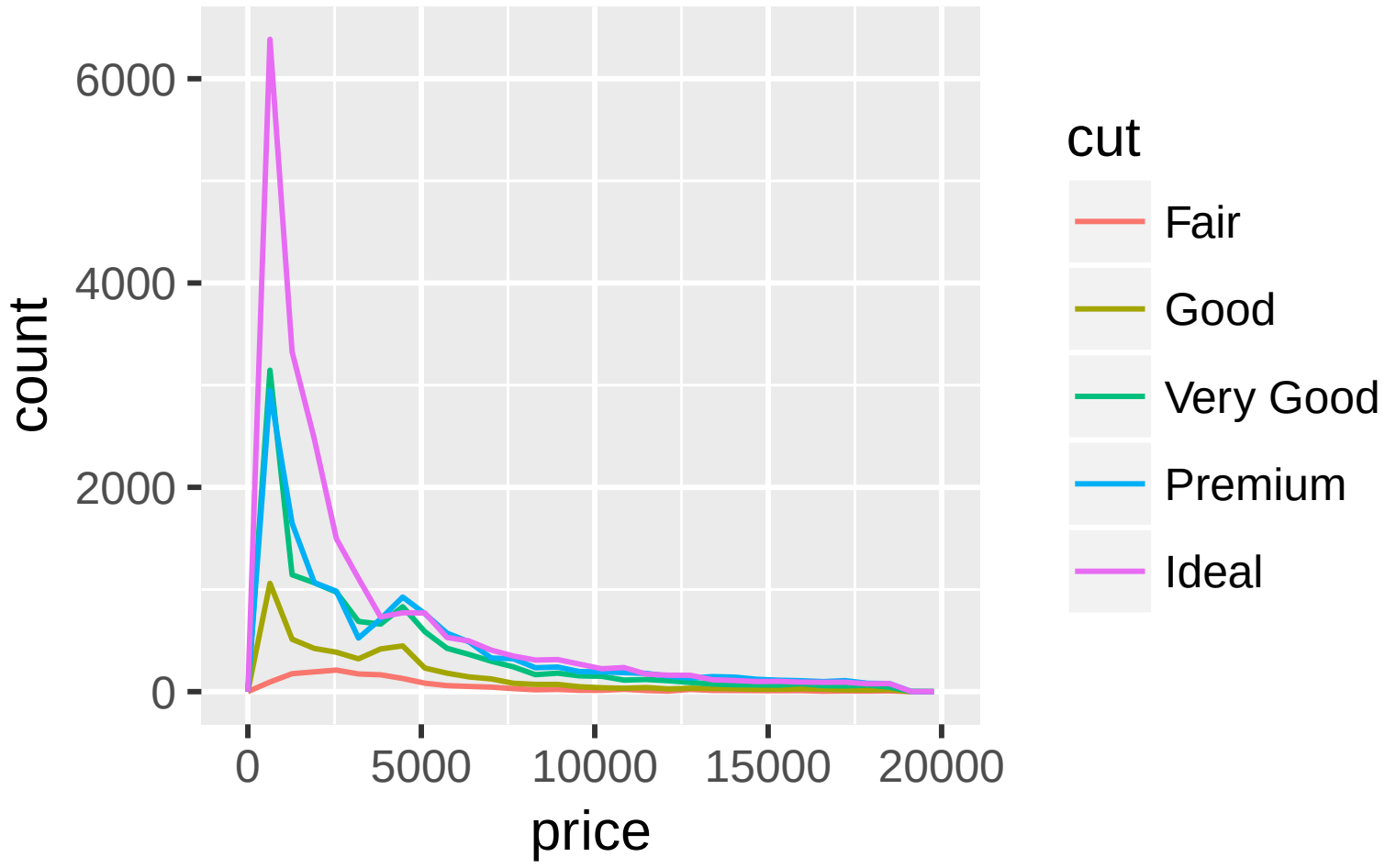
```
## # A tibble: 53,940 x 10
##    carat       cut color clarity depth table price     x
##    <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl
##  1  0.23     Ideal     E     SI2  61.5    55   326  3.95  3.9
##  2  0.21   Premium     E     SI1  59.8    61   326  3.89  3.8
##  3  0.23      Good     E     VS1  56.9    65   327  4.05  4.0
##  4  0.29   Premium     I     VS2  62.4    58   334  4.20  4.2
##  5  0.31      Good     J     SI2  63.3    58   335  4.34  4.3
##  6  0.24 Very Good     J    VVS2  62.8    57   336  3.94  3.9
##  7  0.24 Very Good     I    VVS1  62.3    57   336  3.95  3.9
##  8  0.26 Very Good     H     SI1  61.9    55   337  4.07  4.1
##  9  0.22      Fair     E     VS2  65.1    61   337  3.87  3.7
## 10  0.23 Very Good     H     VS1  59.4    61   338  4.00  4.0
## # ... with 53,930 more rows
```

Covariation

- Covariation is the tendency of values of two or more variables to vary together in a related way
  - Dependency
  - Confounding
- Understand by visualizing the relationship between two or more variables

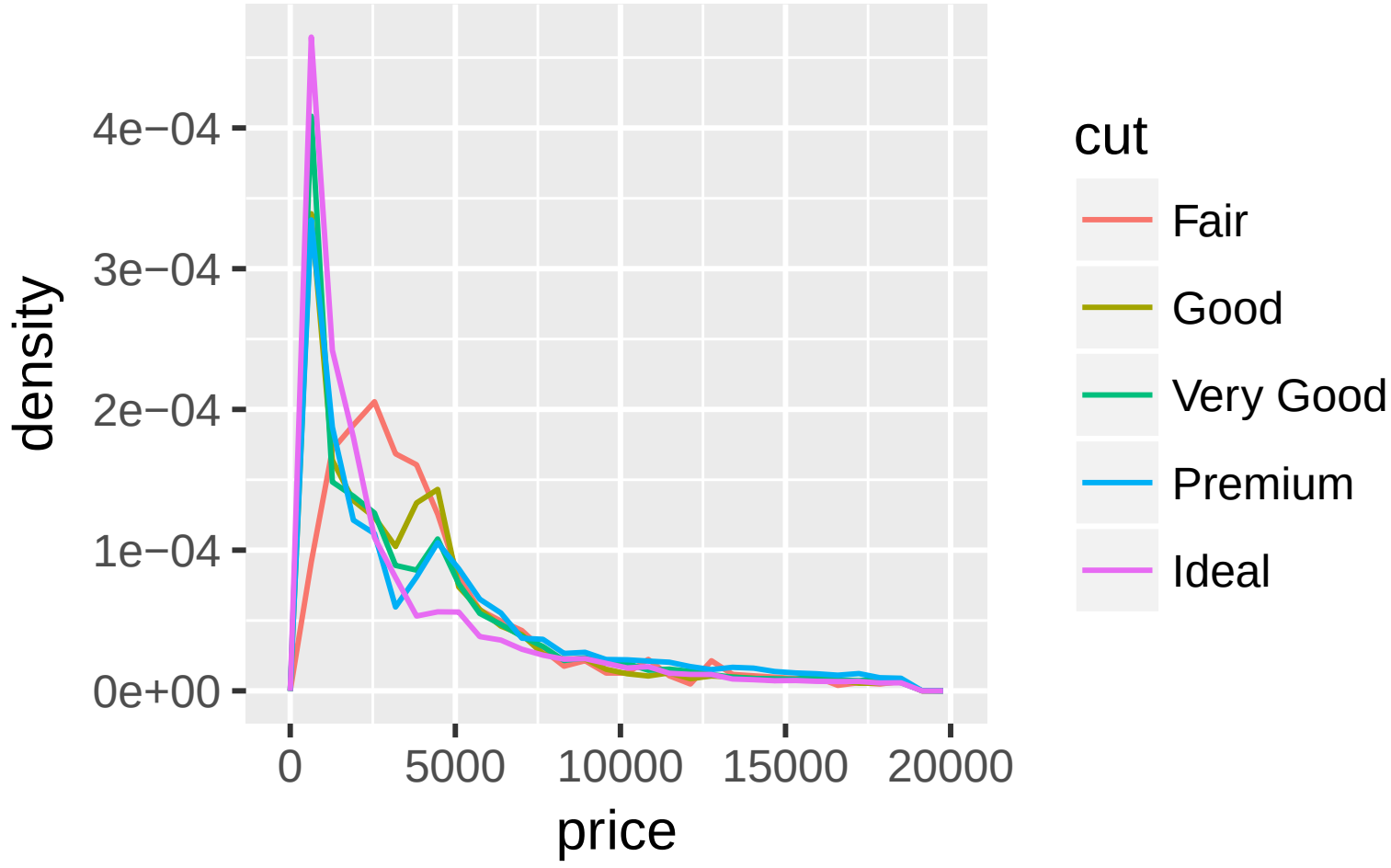
Between a categorical and a continuous variable

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



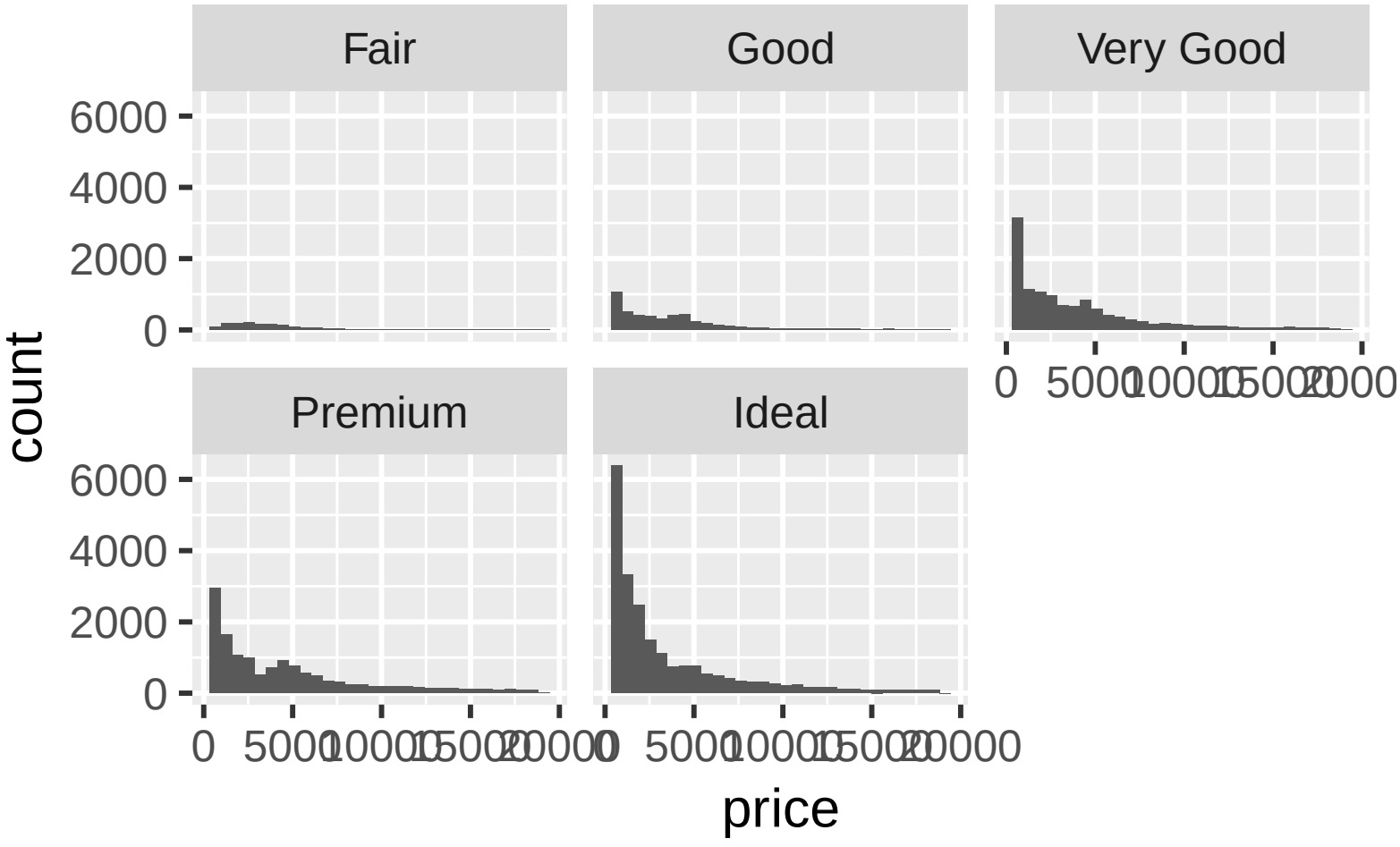
Between a categorical and a continuous variable

```
ggplot(diamonds) + geom_freqpoly(aes(x=price,
                                     y=..density..,
                                     color=cut))
```



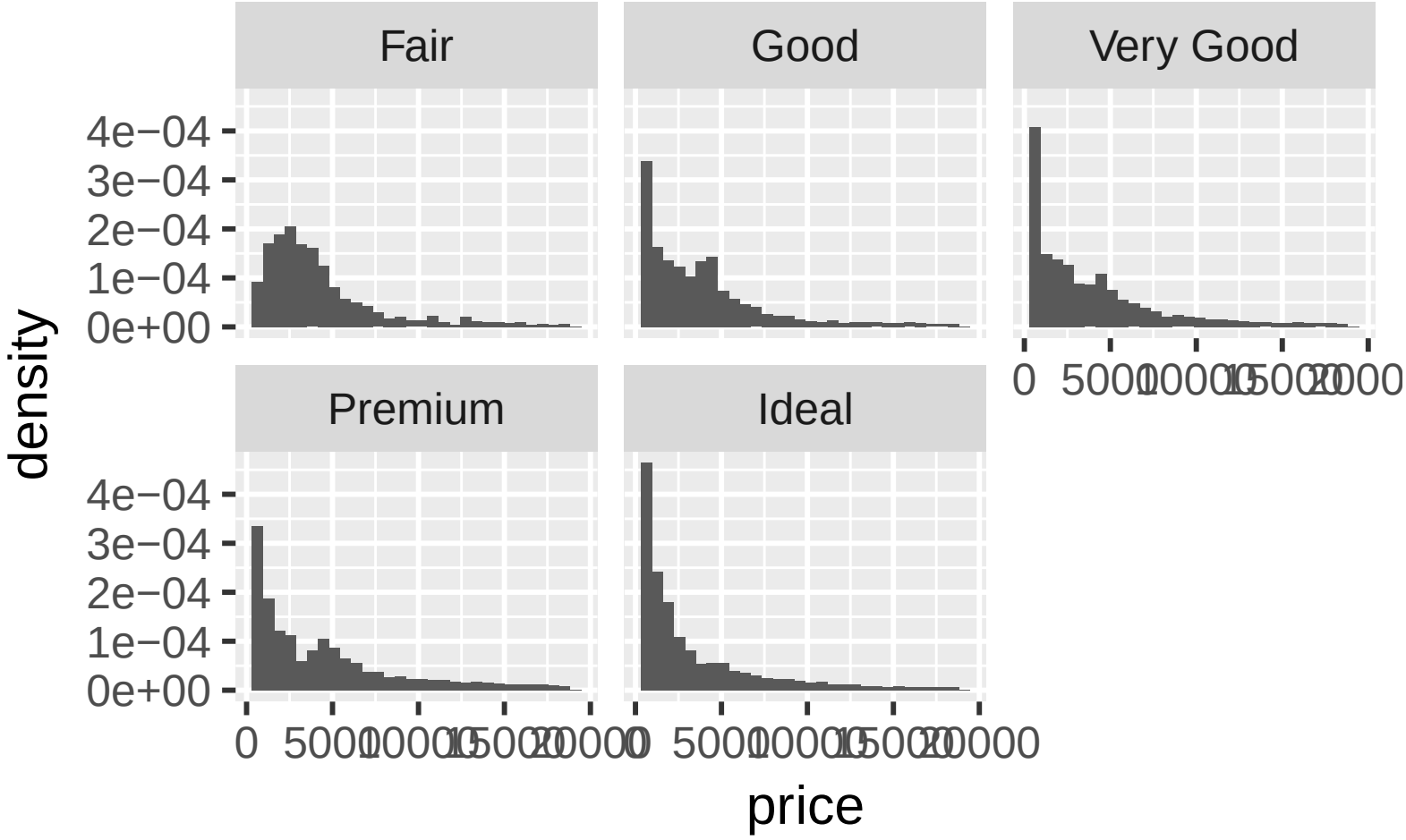
Between a categorical and a continuous variable

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



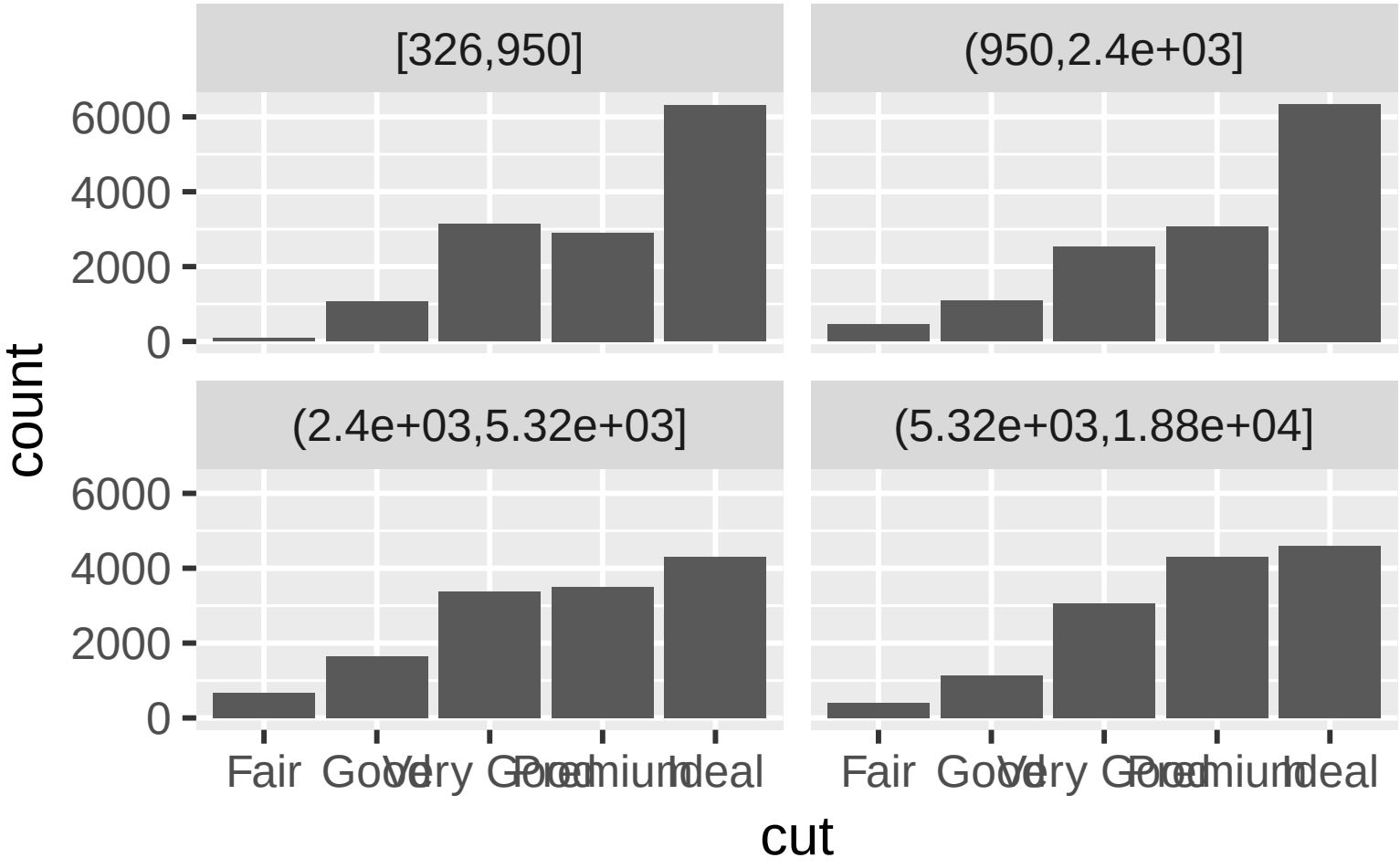
Between a categorical and a continuous variable

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



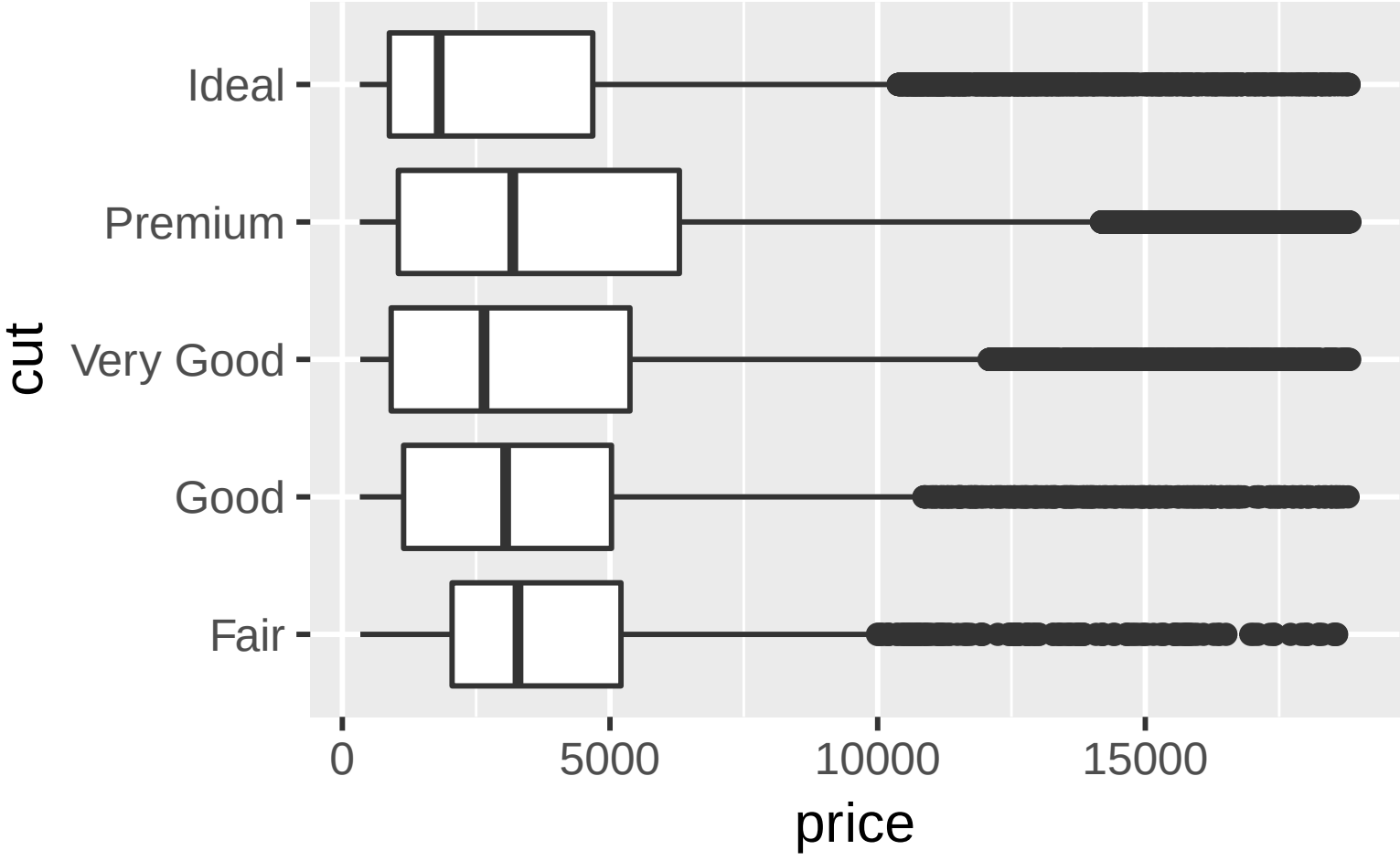
Between a categorical and a continuous variable

```
ggplot(diamonds) + geom_bar(aes(x=cut)) +
  facet_wrap(~ cut_number(price, n=4))
```



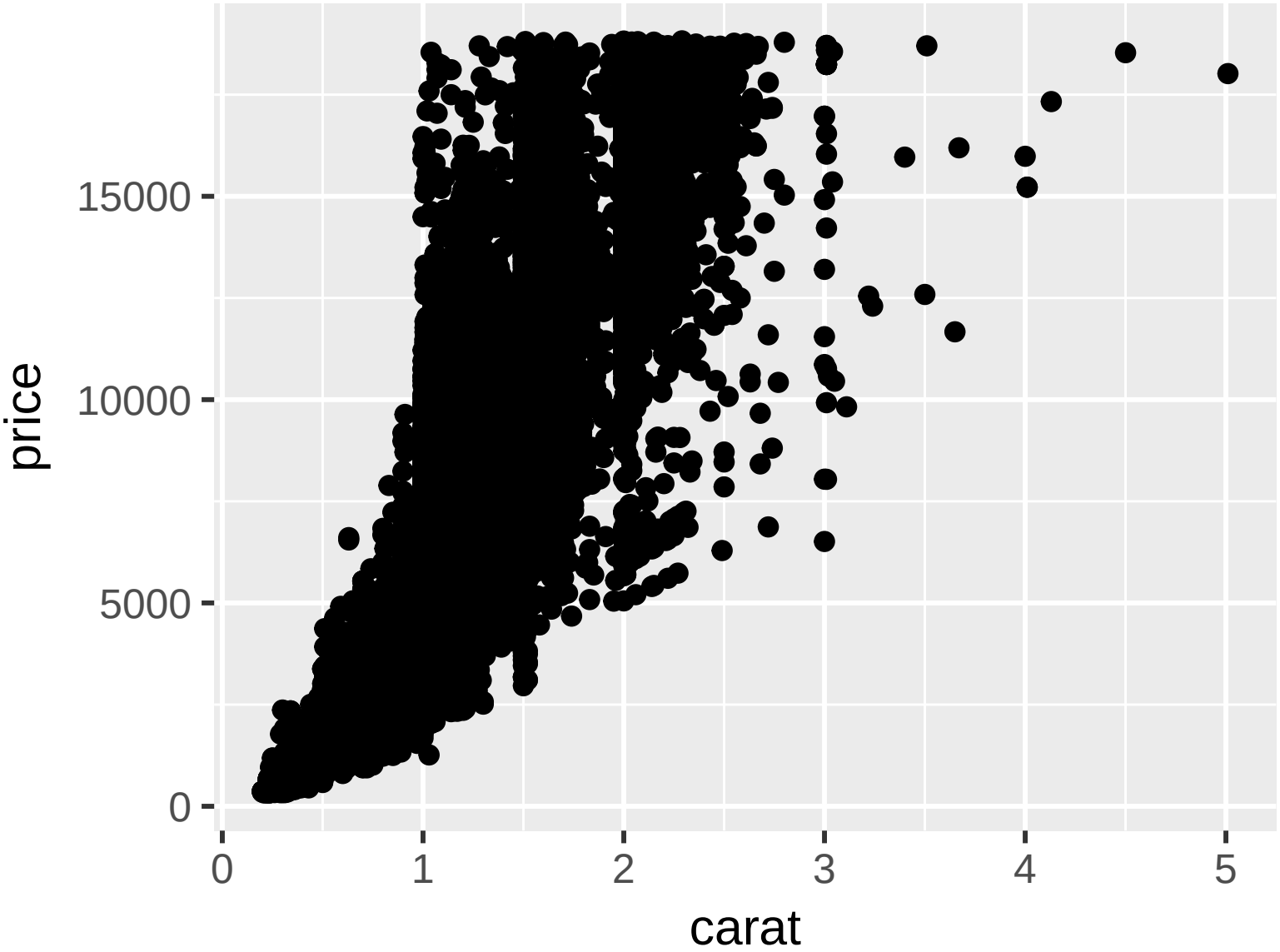
Between a categorical and a continuous variable

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



Between two continuous variables

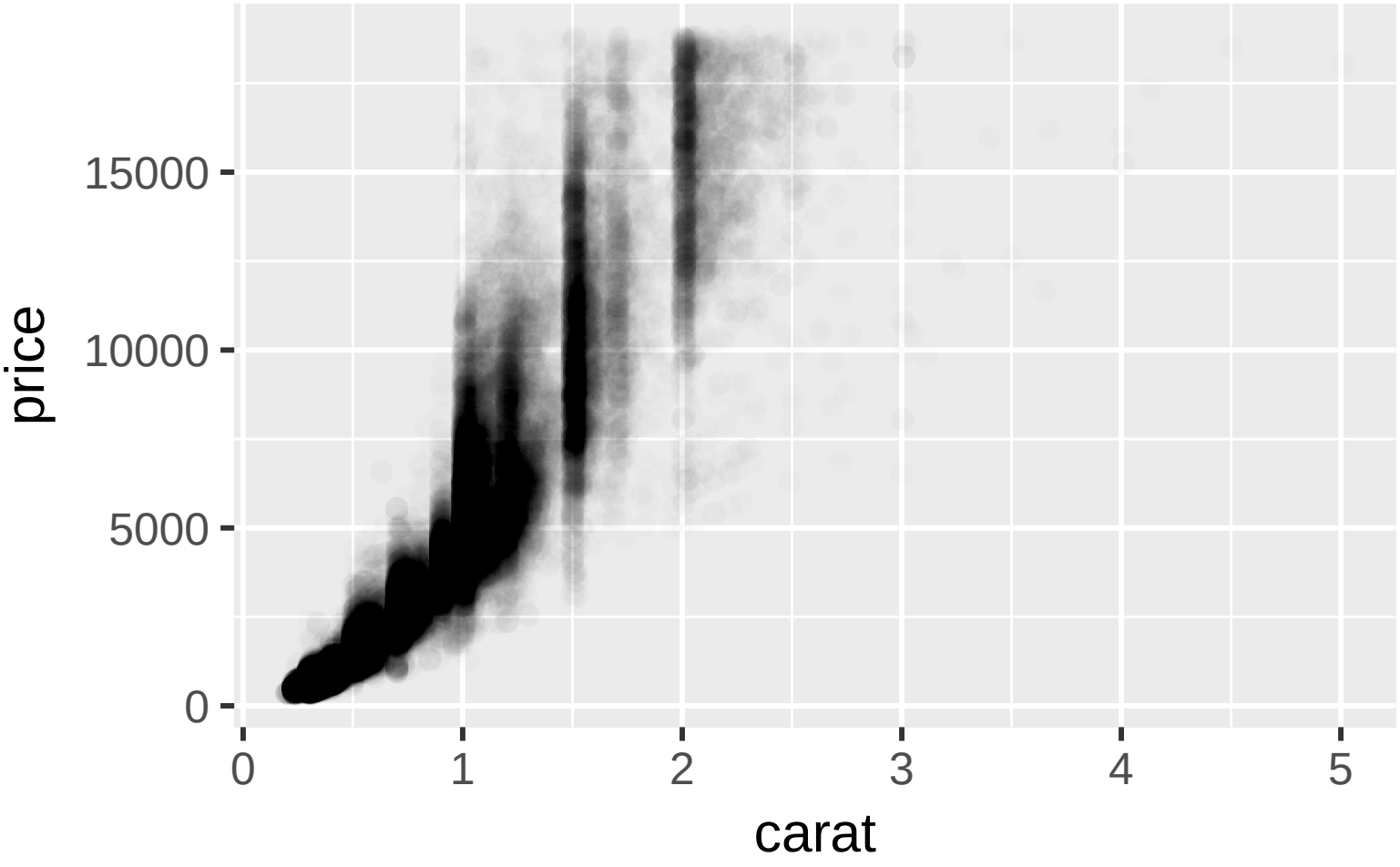
```
ggplot(diamonds) + geom_point(aes(x=carat, y=price))
```



Between two continuous variables

Use transparency to fix overplotting.

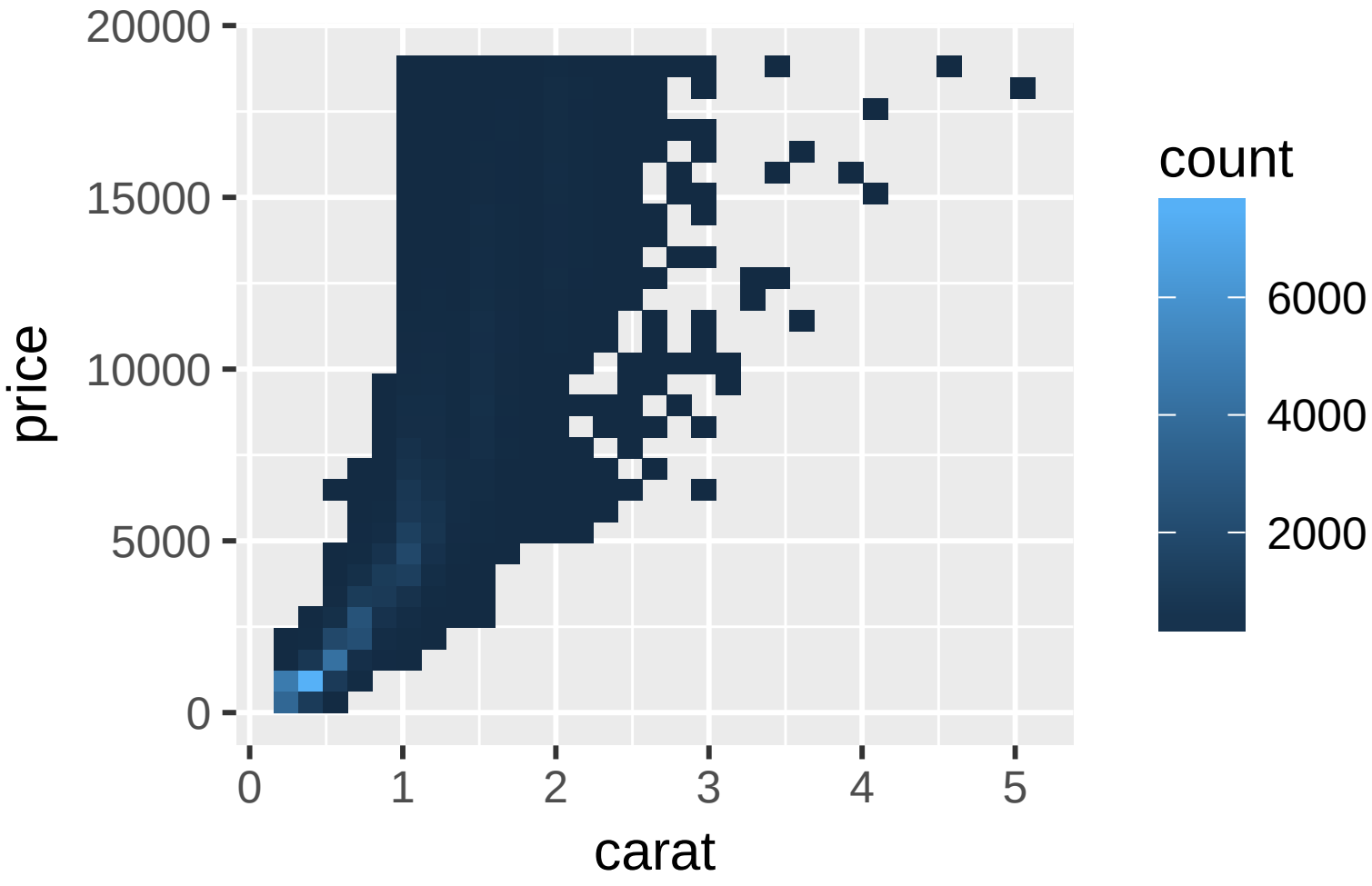
```
ggplot(diamonds) + geom_point(aes(x=carat, y=price), alpha=1/100
```



Between two continuous variables

Use 2D binning to fix overplotting.

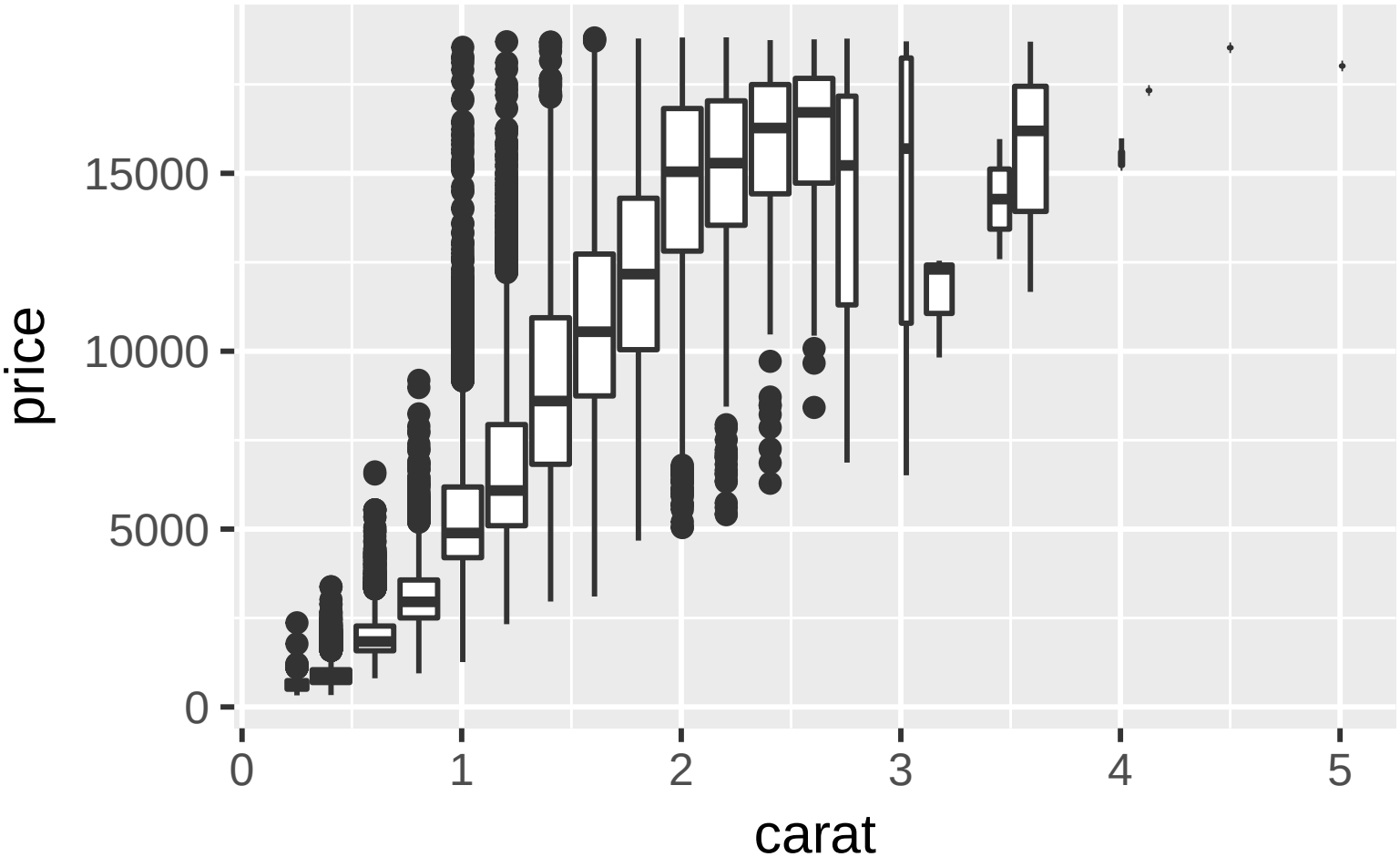
```
ggplot(diamonds) + geom_bin2d(aes(x=carat, y=price))
```



Between two continuous variables

Use fixed-width binned boxplots to fix overplotting.

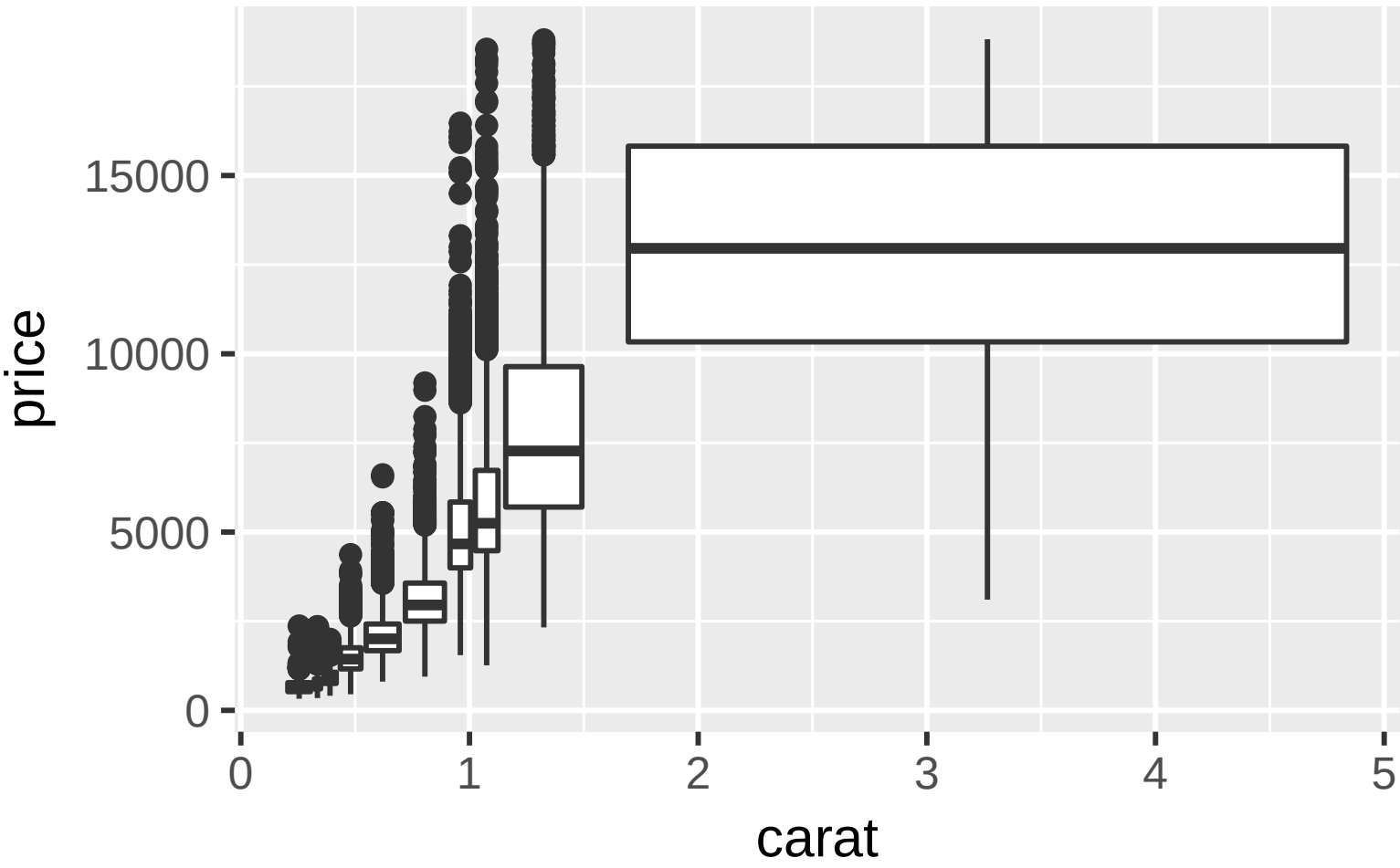
```
ggplot(diamonds, aes(x=carat, y=price)) +
  geom_boxplot(aes(group=cut_width(carat, 0.2)))
```



Between two continuous variables

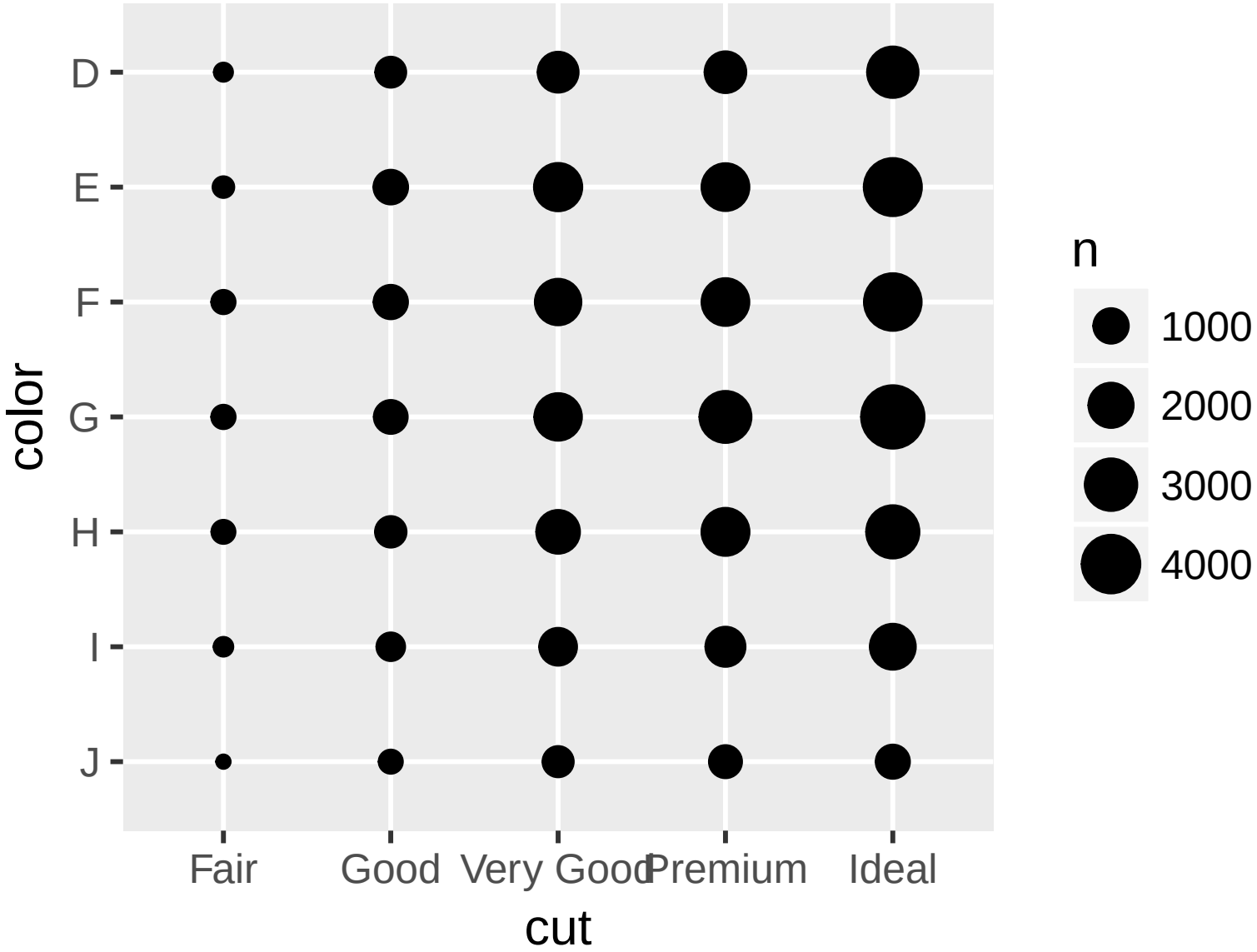
Use variable-width binned boxplots to fix overplotting.

```
ggplot(diamonds, aes(x=carat, y=price)) +
  geom_boxplot(aes(group=cut_number(carat, 10)))
```



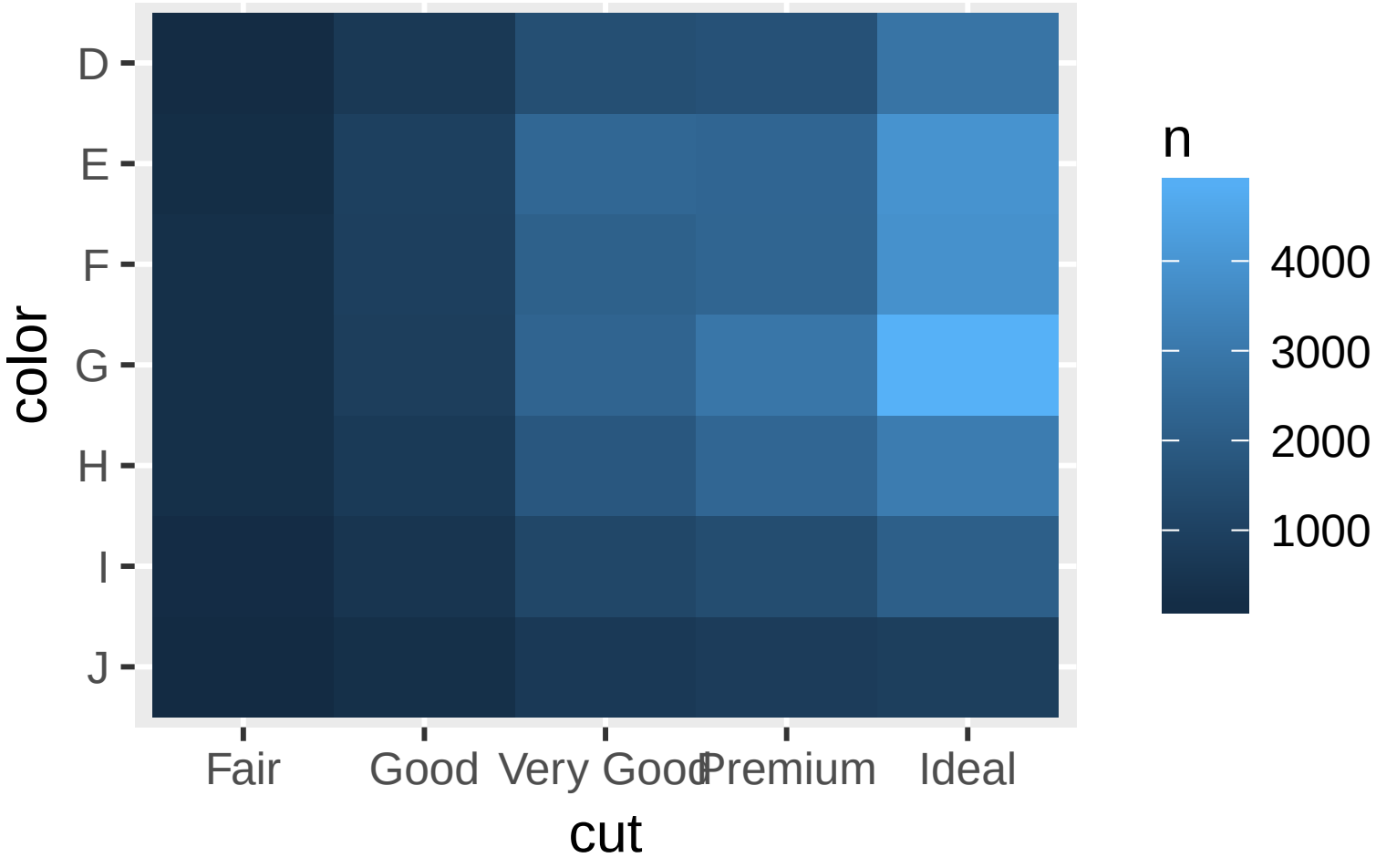
Between two categorical variables

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



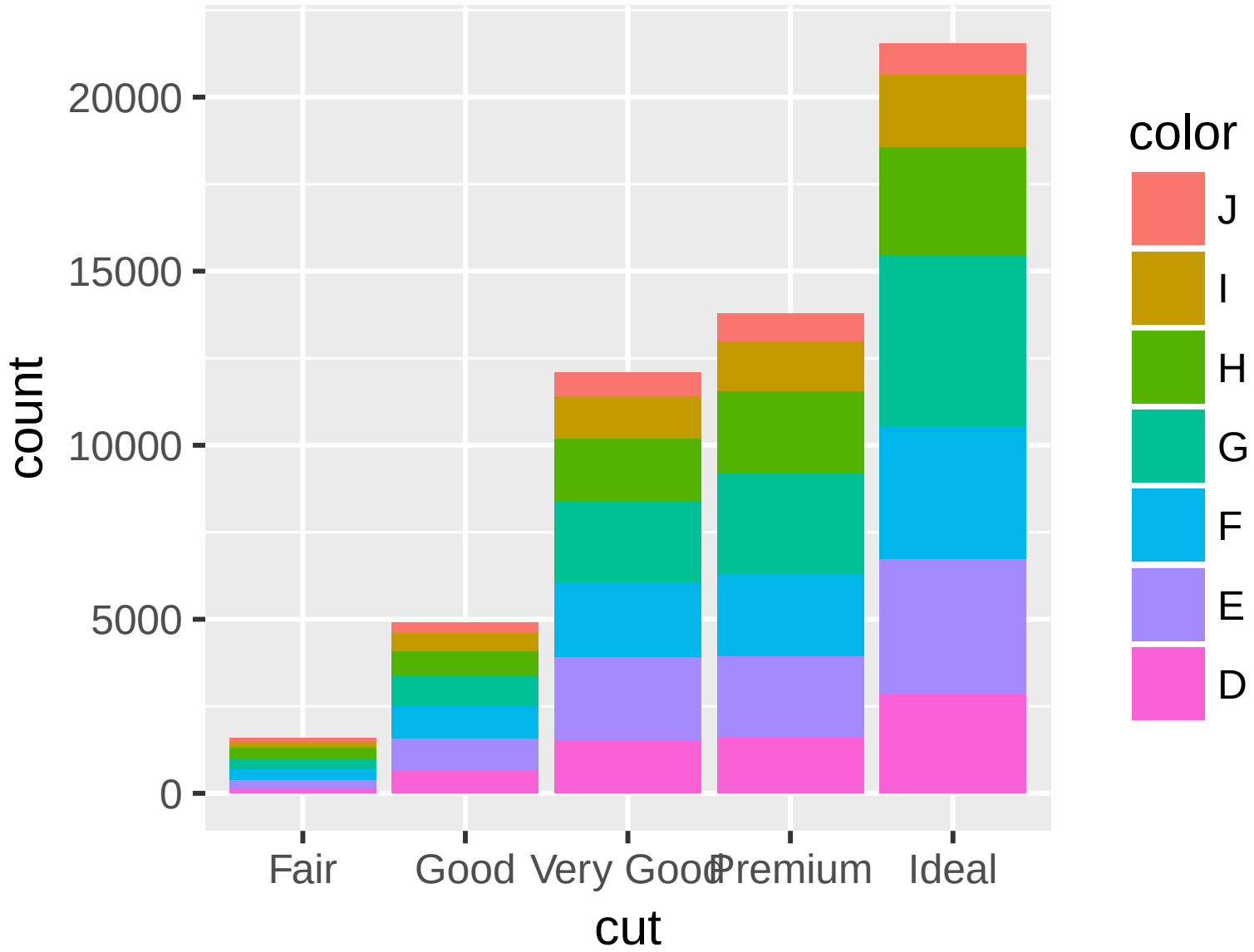
Between two categorical variables

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



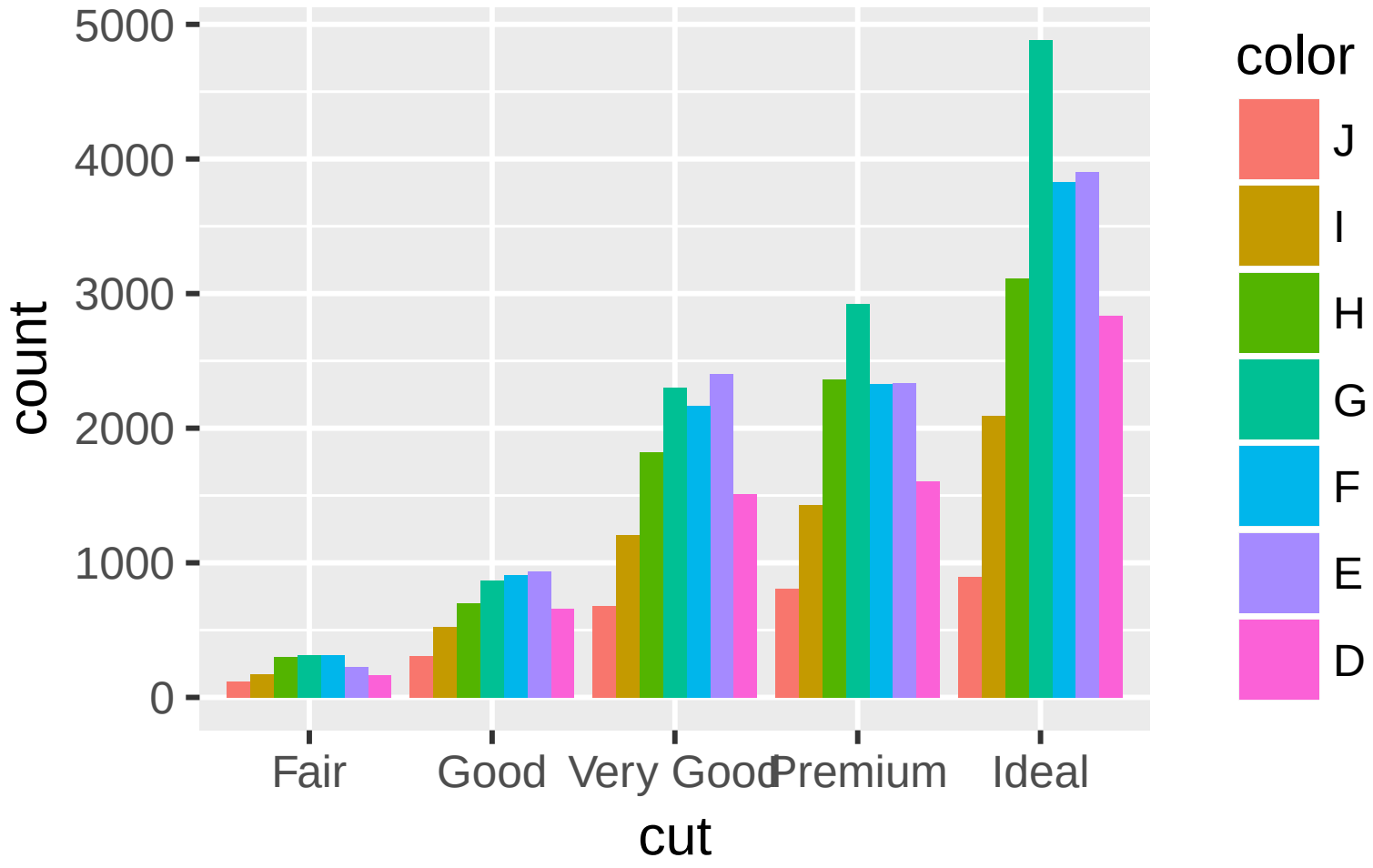
Between two categorical variables

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



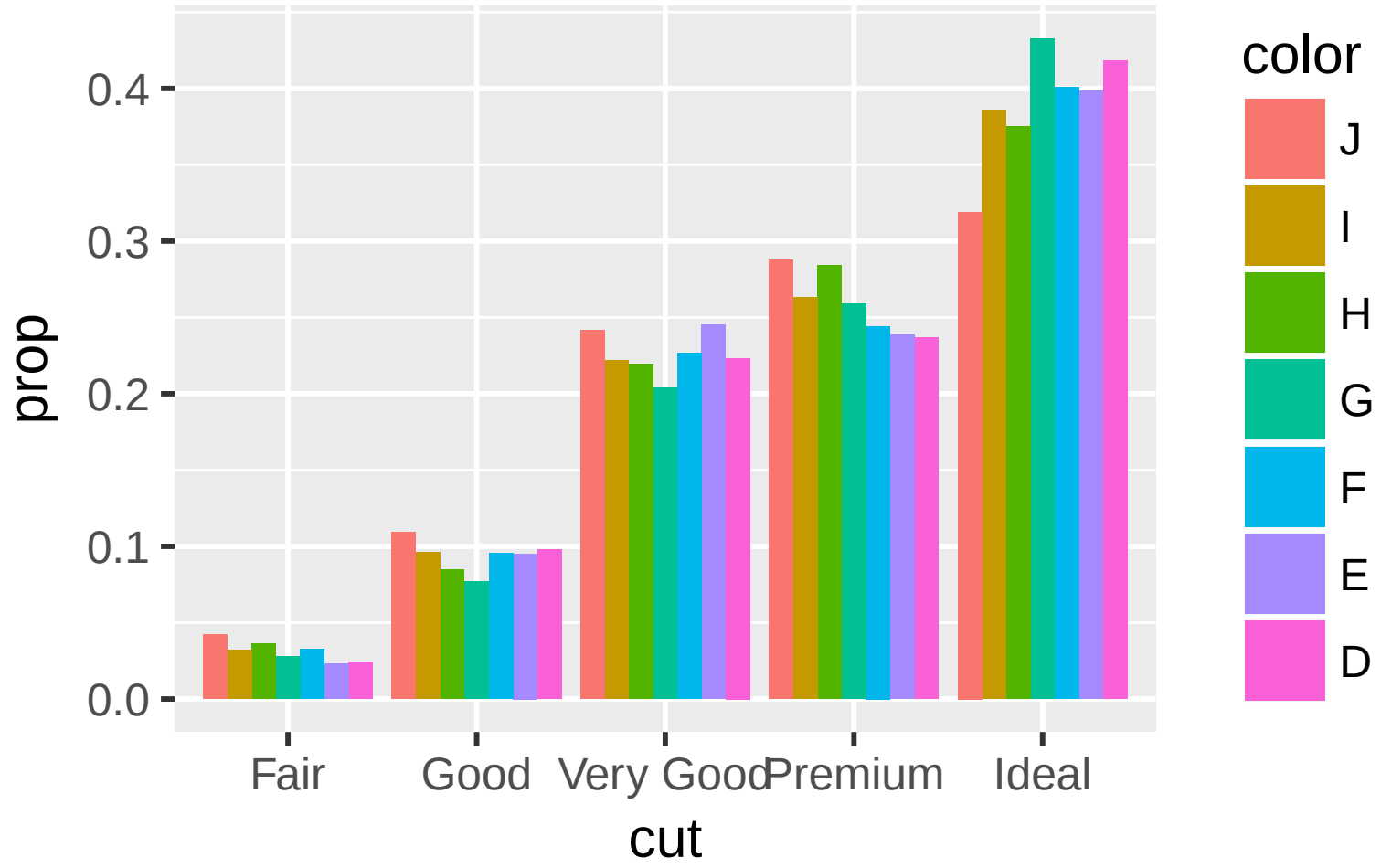
Between two categorical variables

```
ggplot(diamonds) + geom_bar(aes(x=cut, fill=color),
                            position="dodge")
```



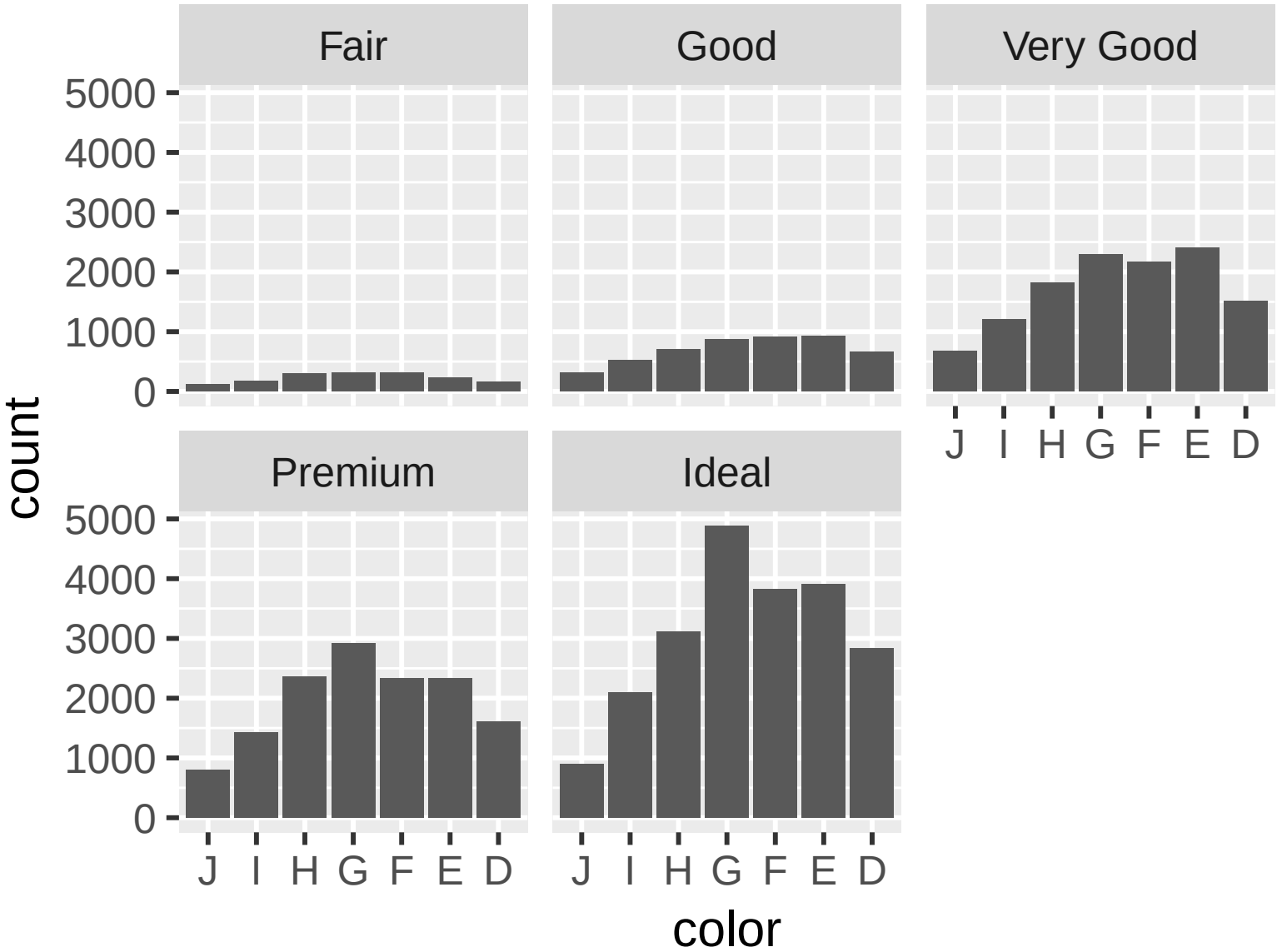
Between two categorical variables

```
ggplot(diamonds) + geom_bar(aes(x=cut, y=..prop..,
                                fill=color, group=color),
                            position="dodge")
```



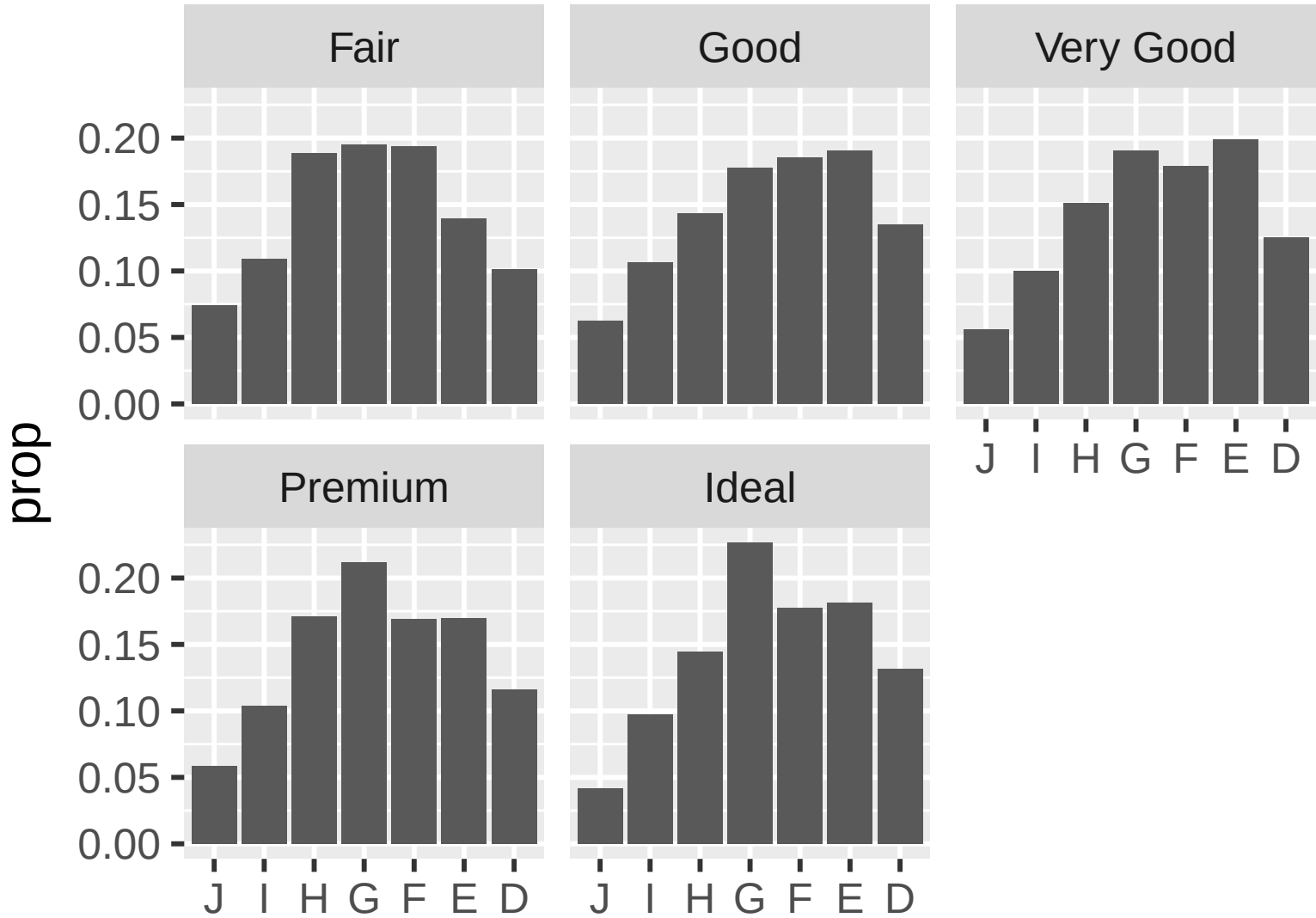
Between two categorical variables

```
ggplot(diamonds) + geom_bar(aes(x=color)) +
  facet_wrap(~ cut)
```



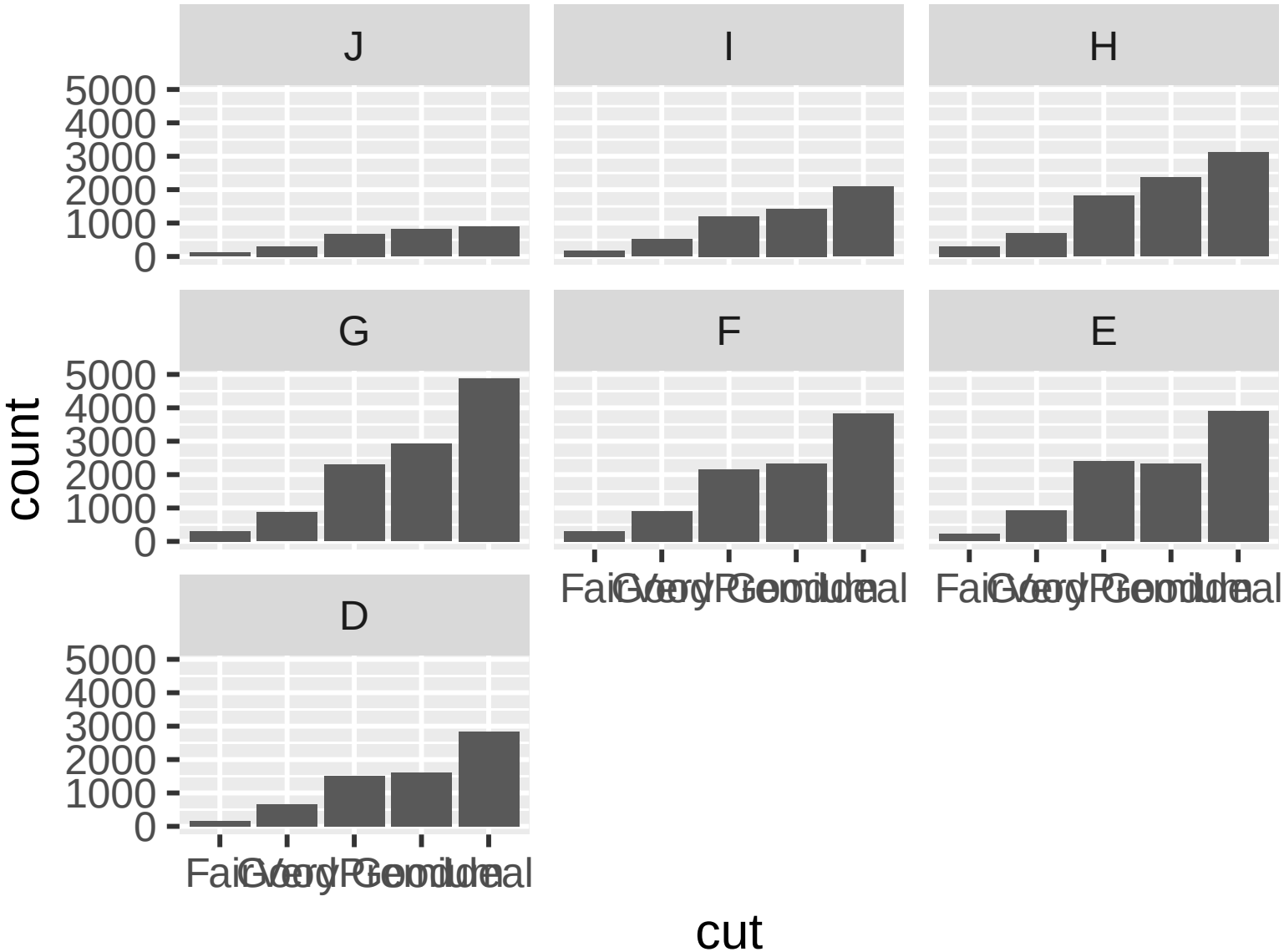
Between two categorical variables

```
ggplot(diamonds) + geom_bar(aes(x=color, y=..prop..,
                                group=cut)) +
  facet_wrap(~ cut)
```



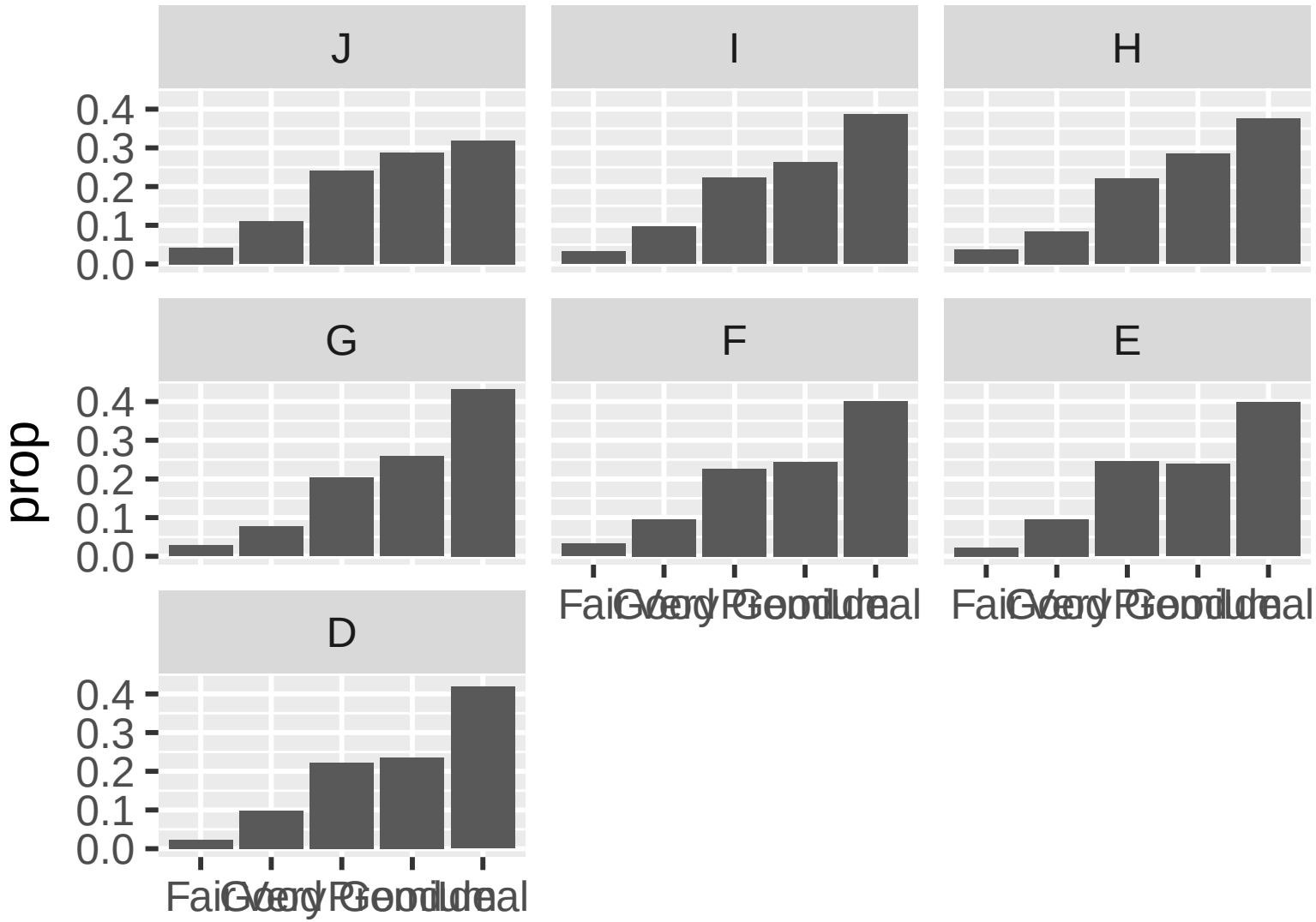
Between two categorical variables

```
ggplot(diamonds) + geom_bar(aes(x=cut)) +
  facet_wrap(~ color)
```



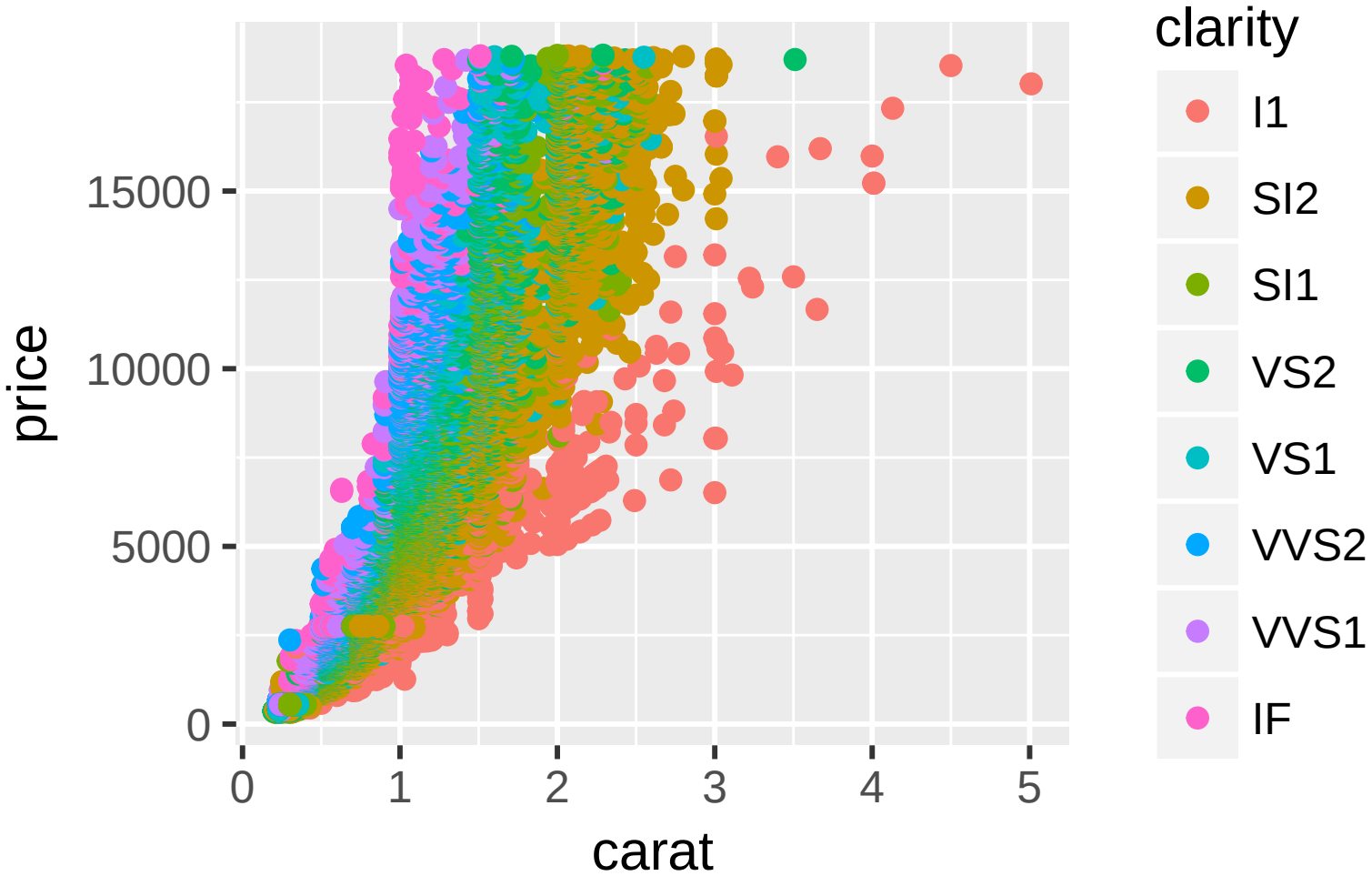
Between two categorical variables

```
ggplot(diamonds) + geom_bar(aes(x=cut, y=..prop..,
                                group=color)) +
  facet_wrap(~ color)
```



Between three or more variables

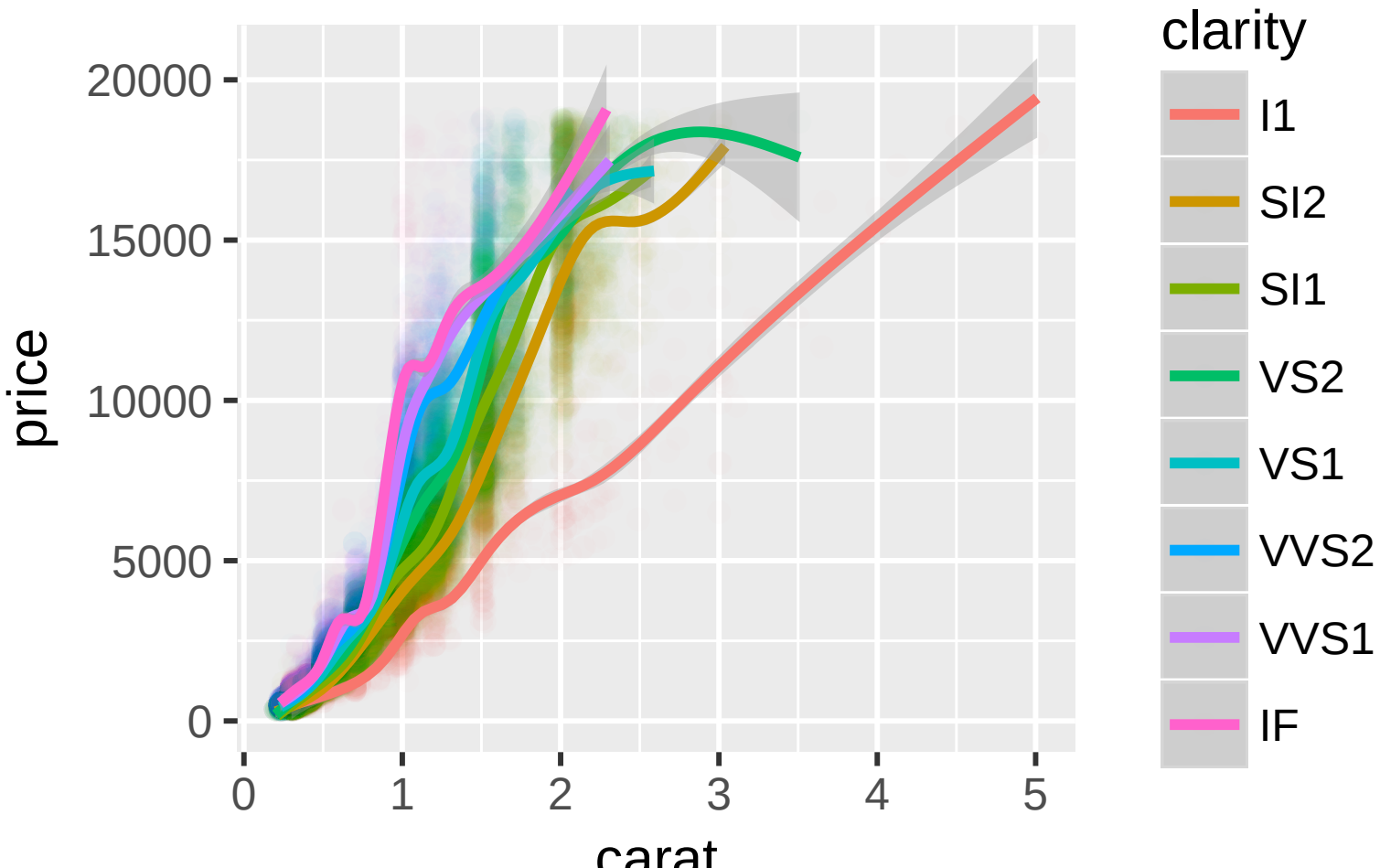
```
ggplot(diamonds, aes(x=carat, y=price, color=clarity)) +
  geom_point()
```



Between three or more variables

```
ggplot(diamonds, aes(x=carat, y=price, color=clarity)) +
  geom_point(alpha=1/100) +
  geom_smooth()
```

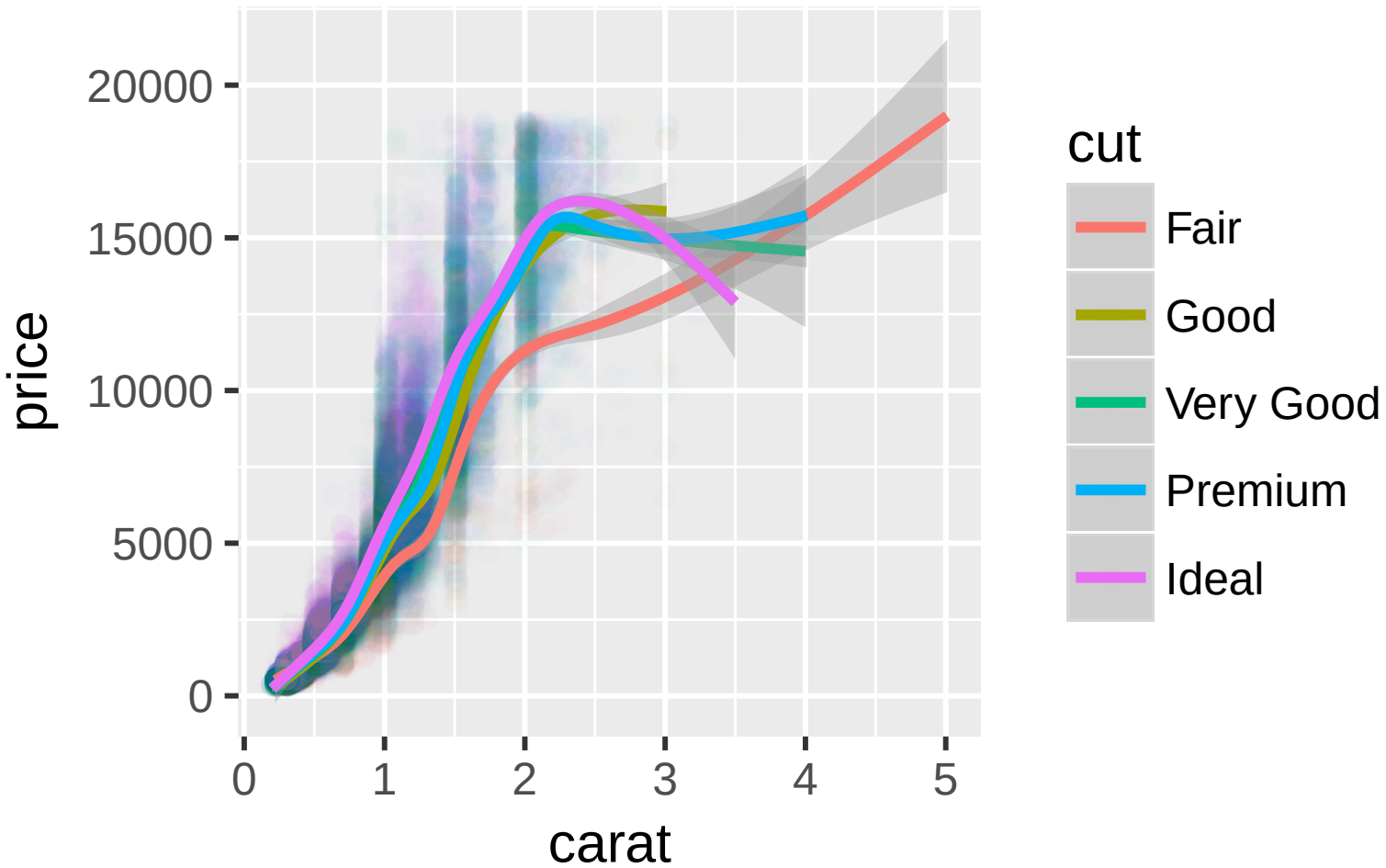
```
## `geom_smooth()` using method = 'gam'
```



Between three or more variables

```
ggplot(diamonds, aes(x=carat, y=price, color=cut)) +
  geom_point(alpha=1/100) +
  geom_smooth()
```

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



More “Fair” cut diamonds are larger

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