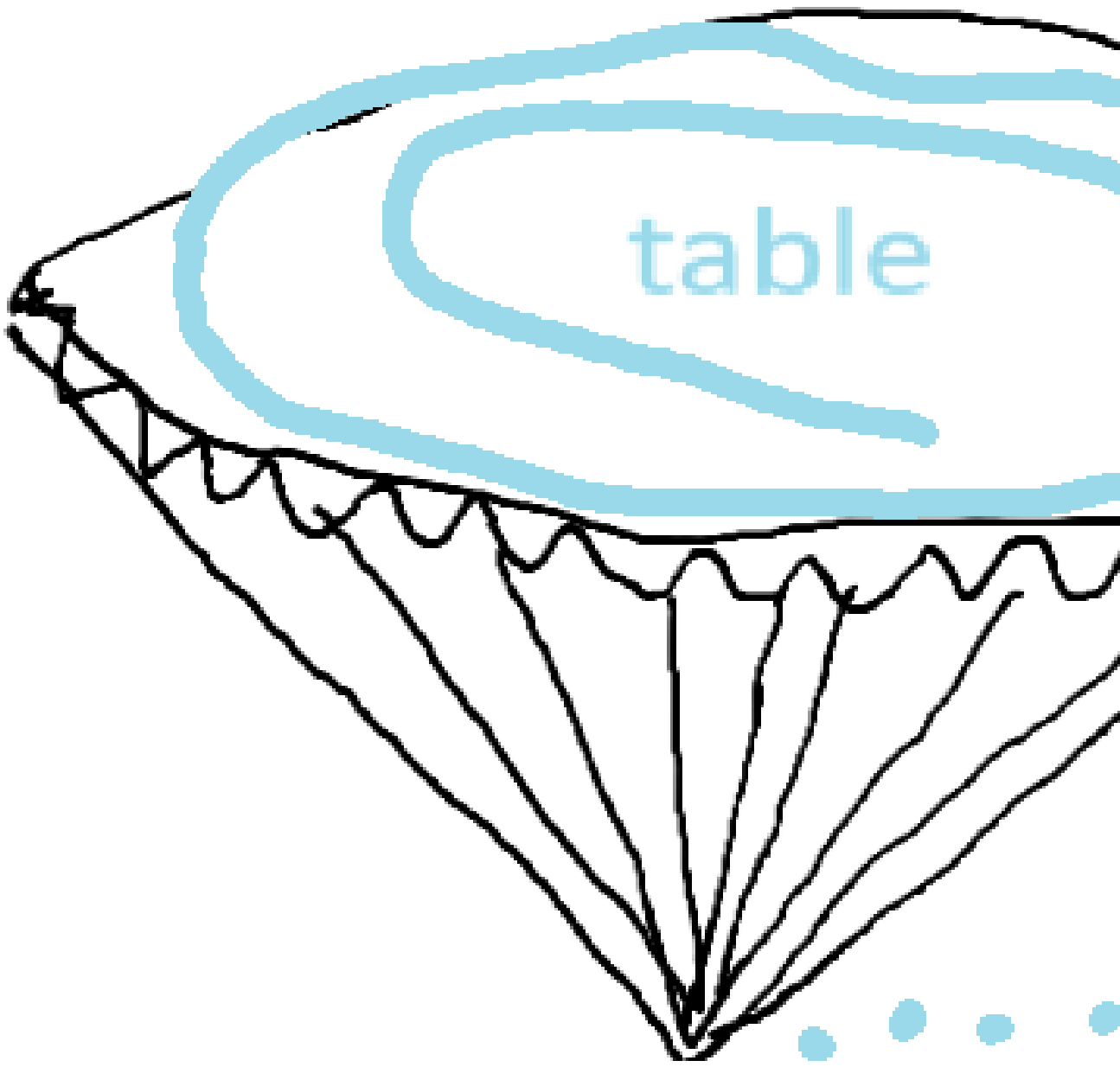
Aesthetic scales and geometries with statistical transformations

Table of contents

1 Different plots for different stories

- use ggplot2 cheatsheet (hardcopy or available in RStudio Help top left menu bar)
- some demonstrations on built-in dataset `diamonds`



table

```r
library(dplyr, warn.conflicts = FALSE, quietly = TRUE)
library(ggplot2, warn.conflicts = FALSE, quietly = TRUE)
colnames(diamonds)
```

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

`diamonds` is a built-in dataset of `ggplot2`. It describes a collection of almost 54,000 diamonds in ten variables. Some of them are numeric. The others are ordinal, which for the purposes of plotting is the same as categorical. The variables are:

- `carat` (weight unit) - numeric
- `cut` (quality of the cut) - ordinal, intelligible values like *Fair*, *Good*...
- `color` - ordinal, colors encoded with random capital letters
- `clarity` - ordinal, encoded with combinations of capital letters and digits
- `depth` - numeric (distance between the flat side and the point of the pointed one)
- `table` - numeric (size of the flat part)
- `price` - numeric
- `x`, `y`, `z` - numeric variables, sizes of some other dimensions

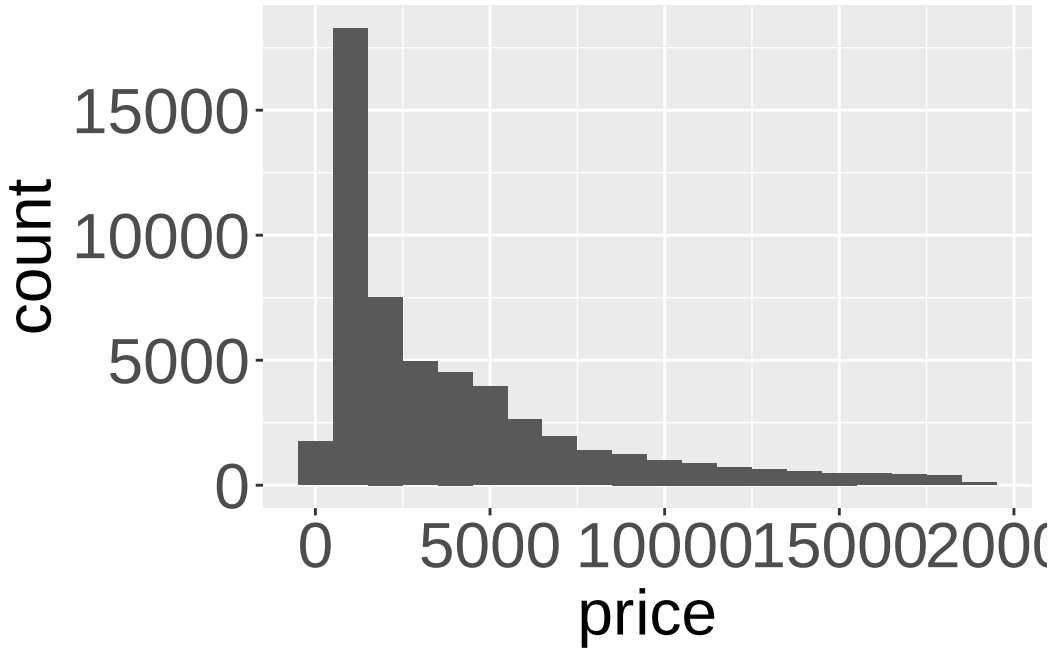
2 Structure of the `diamonds` data frame

```
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 ...
```

3 Histogram

- one quantitative variable
- frequency distribution of value intervals

```
ggplot(data = diamonds) +
  geom_histogram(mapping = aes(x = price), binwidth = 1000) +
  theme(axis.text = element_text(size = 24),
        axis.title = element_text(size = 24))
```

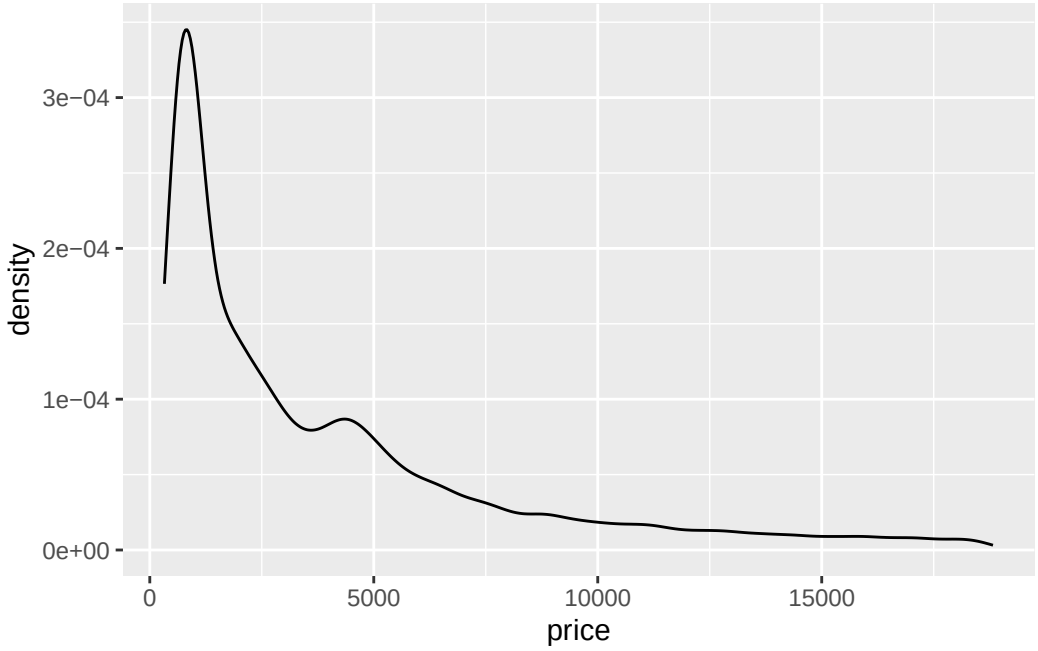


It takes your variable, sorts the values and divides them into equally wide intervals (30 by default). They are called **bins** and look like bars. The height of each bin is the count of observations that fit in the bin. Here the bin width is set to 1000 (USD). You can either determine the width of the bin like here, or how many bins you want (using an argument called `bins`).

4 Density

- like histogram but computes how likely you pay that exact sum in USD if you pick a random diamond.

```
ggplot(data = diamonds) + geom_density(mapping = aes(x = price))
```



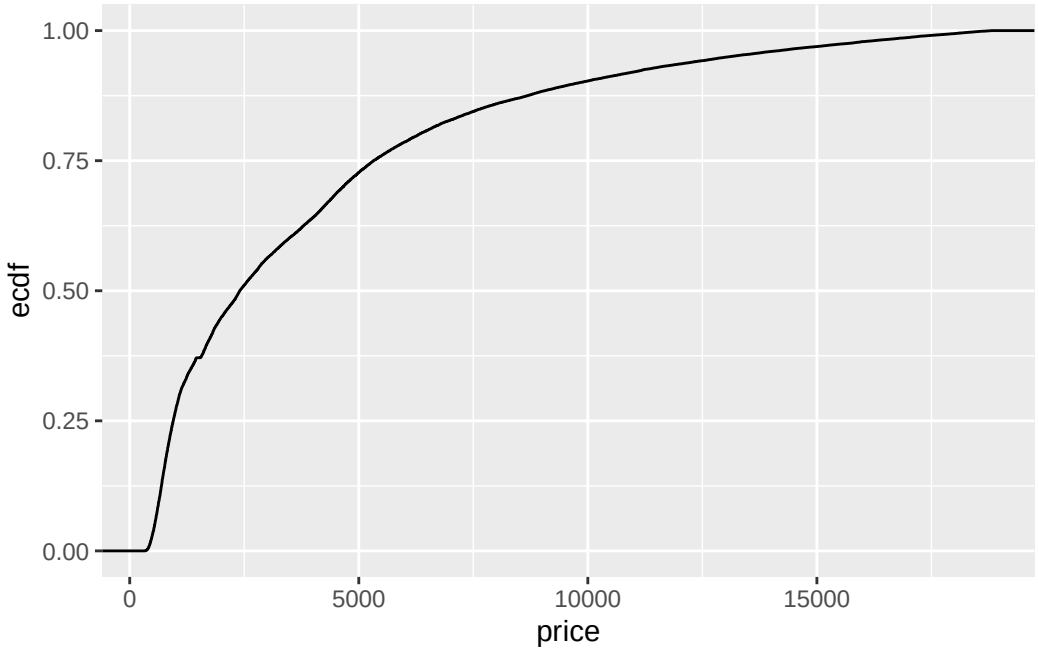
You see that you will pay most likely between a few hundred and 2,000 USD, but the probabilities are very low (around 0.035% the highest) because of the unit being one dollar. Would you hazard picking a random diamond if you did not see the entire curve? Almost certainly not.

You would probably want to know how likely you will pay at most a given price! That is what **cumulative density** is for.

5 (Empirical) Cumulative Density (Function)

- shows you how likely you pay that and lower price for a random diamond

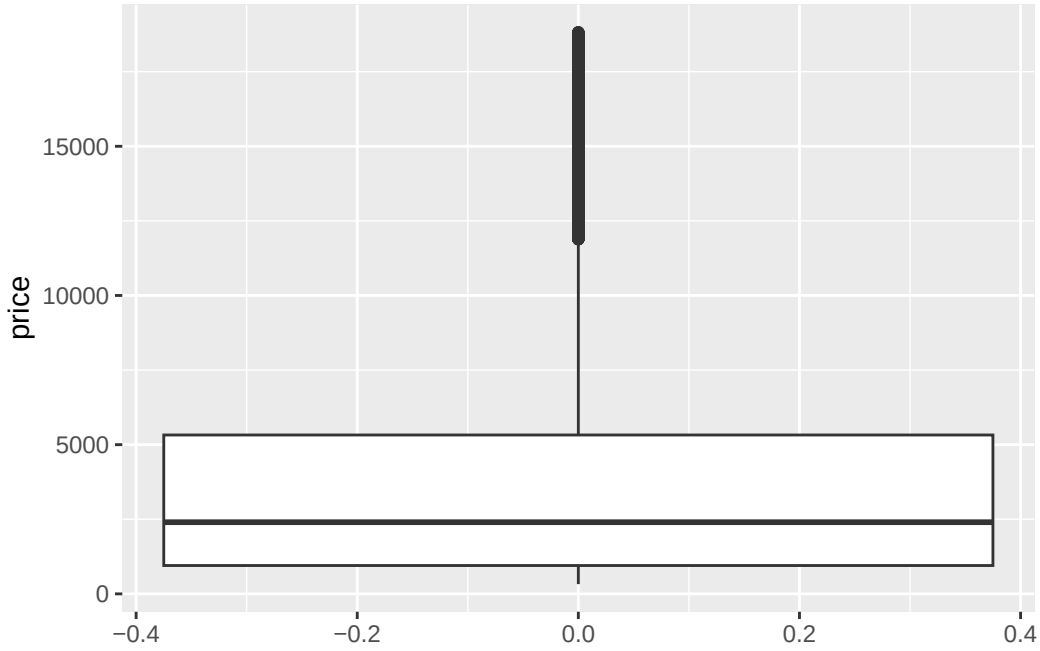
```
ggplot(data = diamonds) + geom_density(mapping = aes(x = price), stat =
↪  "ecdf")
```



This tells you that you have a 50% chance to get your random diamond for at most USD 2,500, and a 75% chance that you pay at most 5,000 USD. This is a much better estimation of whether a random purchase is going to ruin you! Note that `stat` argument and look it up in the cheat sheet. It is mostly calculations in different flavors of bins and probability density curves. All this fits well with a histogram.

6 Boxplot

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



This is a boxplot. The function sorts the observations and computes their percentiles. The box reaches from the first to the third quartile. That is, half of the observations seen in the data fit into the box. The vertical bar is the median value (50% of observations are lower than this or equal). The whiskers show the range of observations within 1.5 height of the box from each side. What is beyond counts as **outliers**, that is, extreme values. The height of the box, the location of the vertical bar, as well as the presence of outliers and length of the whiskers give you an idea of how the values are distributed. Boxplots are very nicely described for instance here: https://www.simplypsychology.org/boxplots.html.

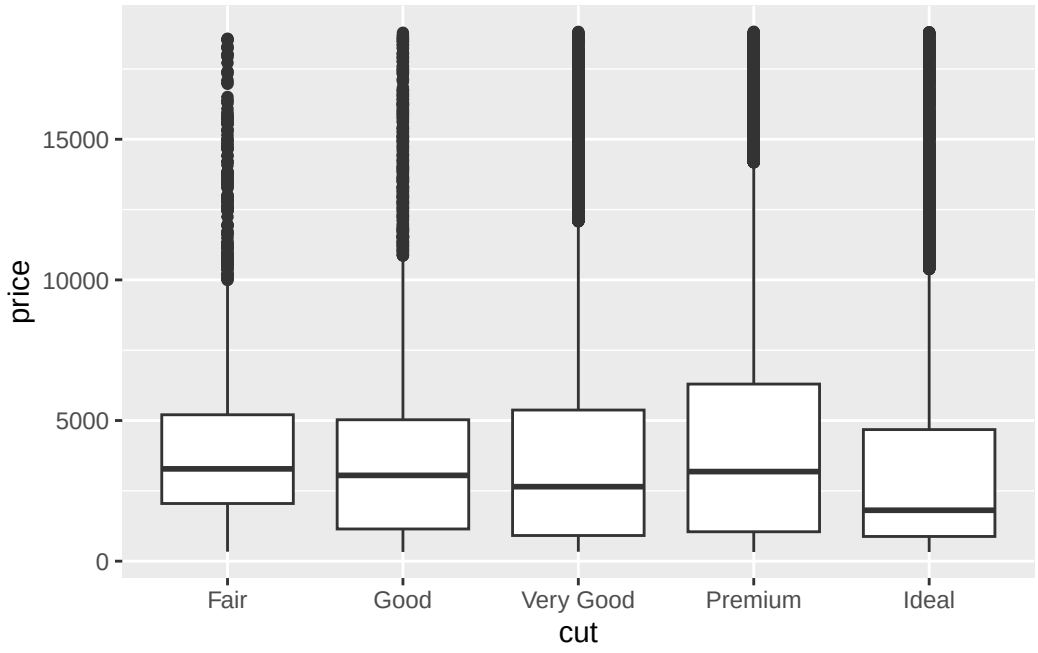
Note that the numbers on the X-axis do not have any meaning and ought to be removed for a presentation.

In this boxplot, we see that almost 75% of the diamonds are cheaper than 5,000 USD and prices above approx. 12,000 USD count as extreme (but you would still have a decent choice of the very premium pieces, judging by how crowded the outlier zone is).

7 Boxplots compared

- You can break them by a discrete variable (categorical or ordinal)
- for instance map `cut` on the x-axis …

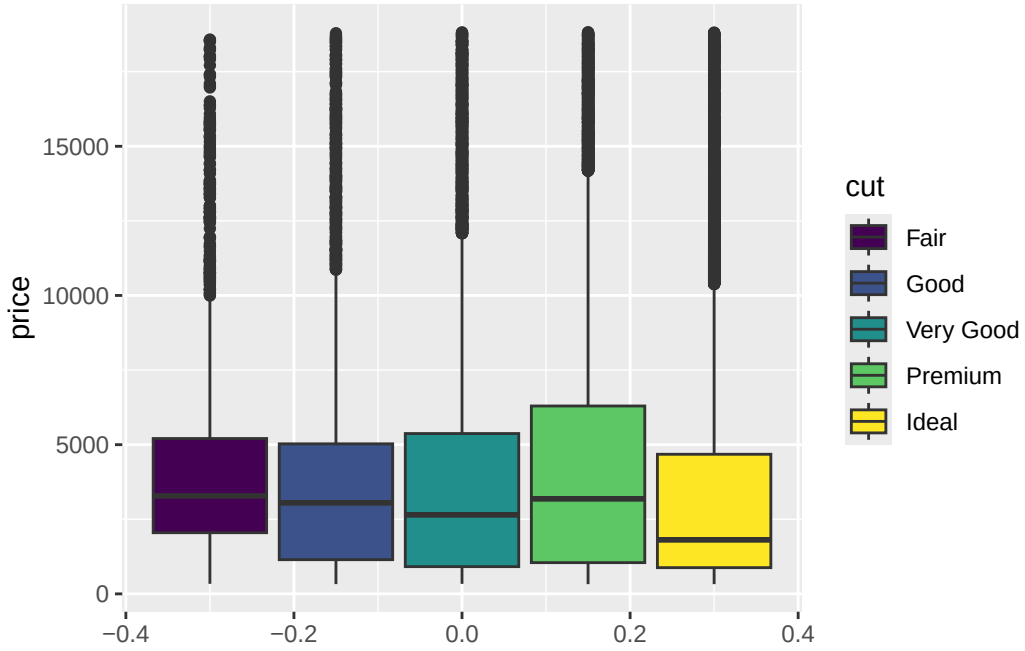
```
ggplot(data = diamonds) + geom_boxplot(mapping = aes(y = price,  x = cut))
```



8 Boxplots compared

- or map `cut` on `fill` or `color`

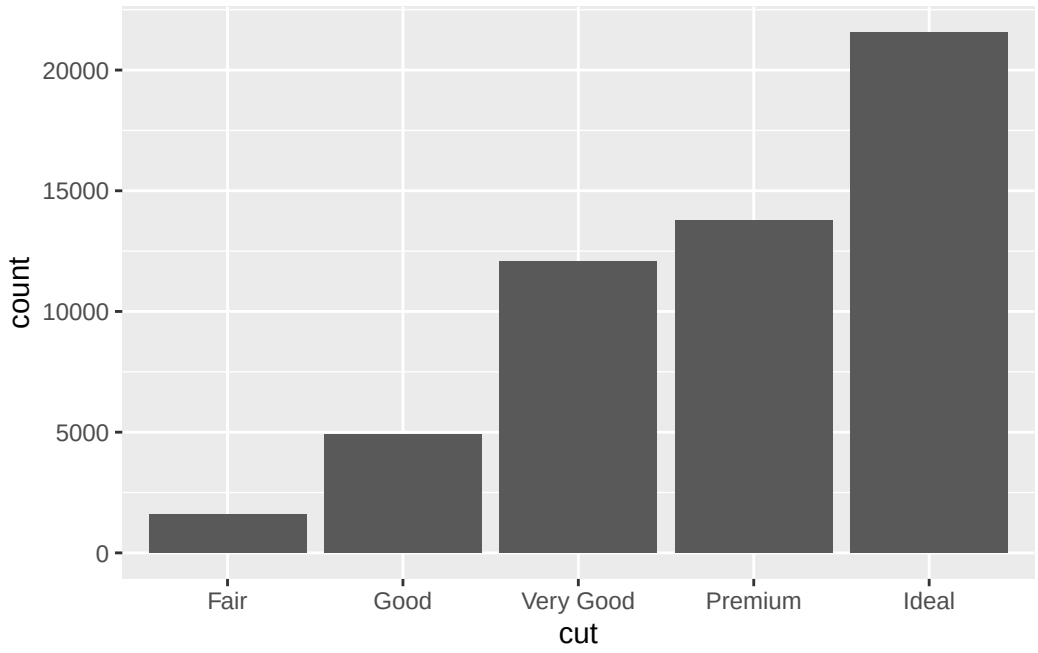
```
ggplot(data = diamonds) + geom_boxplot(mapping = aes( y = price, fill = cut))
↪
```



9 Bar plot with counts (default)

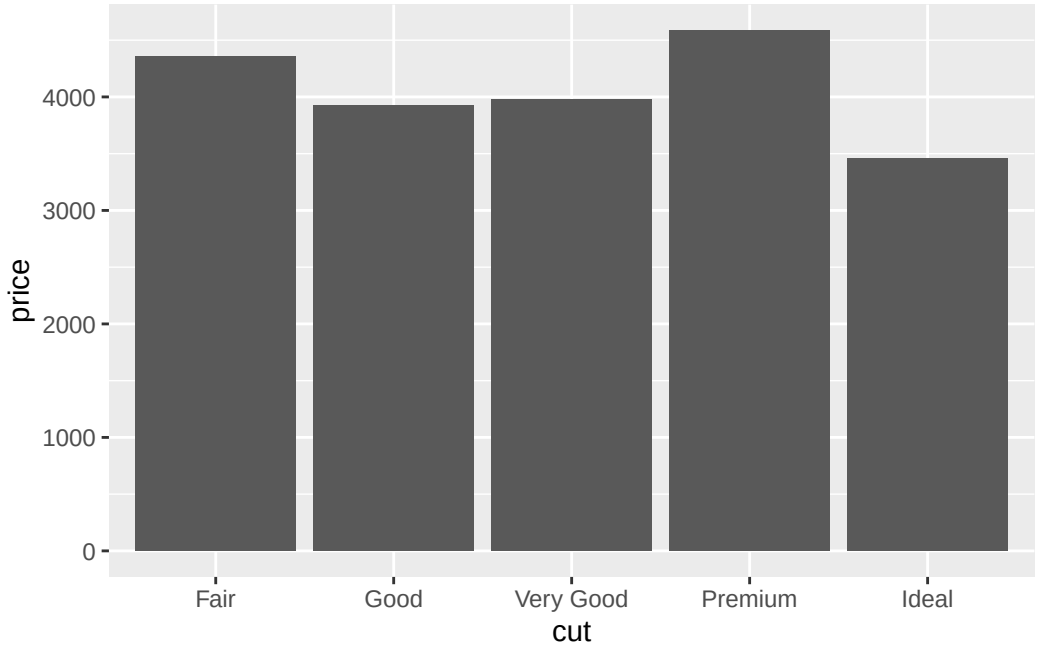
Each bar represents one variable category (value), height shows **count** (by default)

```
ggplot(data = diamonds) + geom_bar(mapping = aes(x = cut), stat = "count")
```



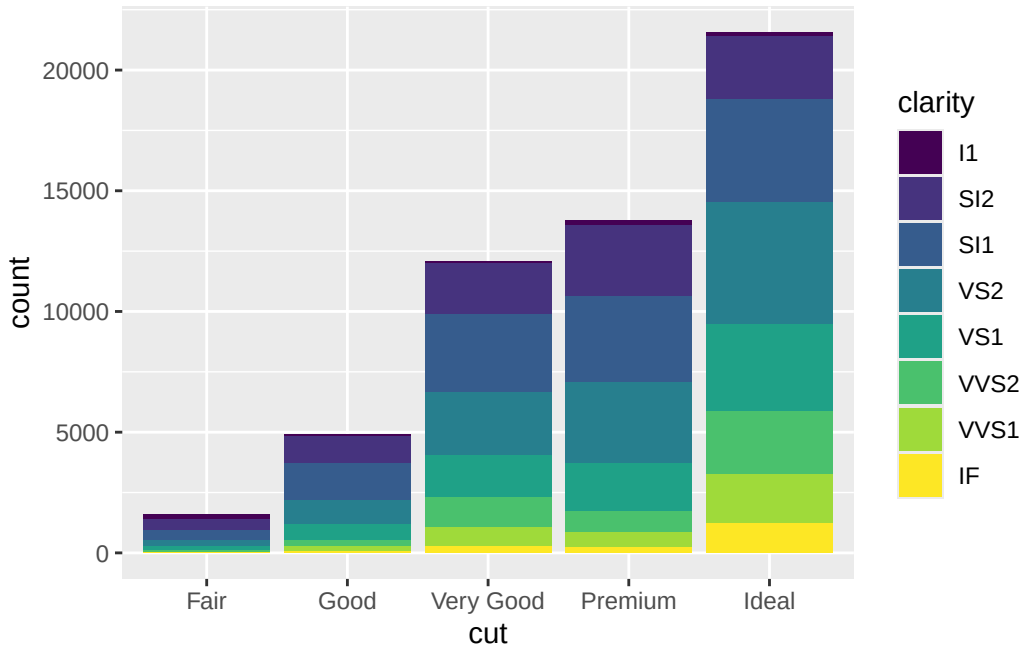
10 Override default stat on Y in a bar plot

```
ggplot(data = diamonds) + geom_bar(mapping = aes(x = cut, y = price),
           stat = "summary", fun = mean) # note where to (not) write quotes
```



11 Bar positions with another categorical variable: `stack`

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "stack")
```

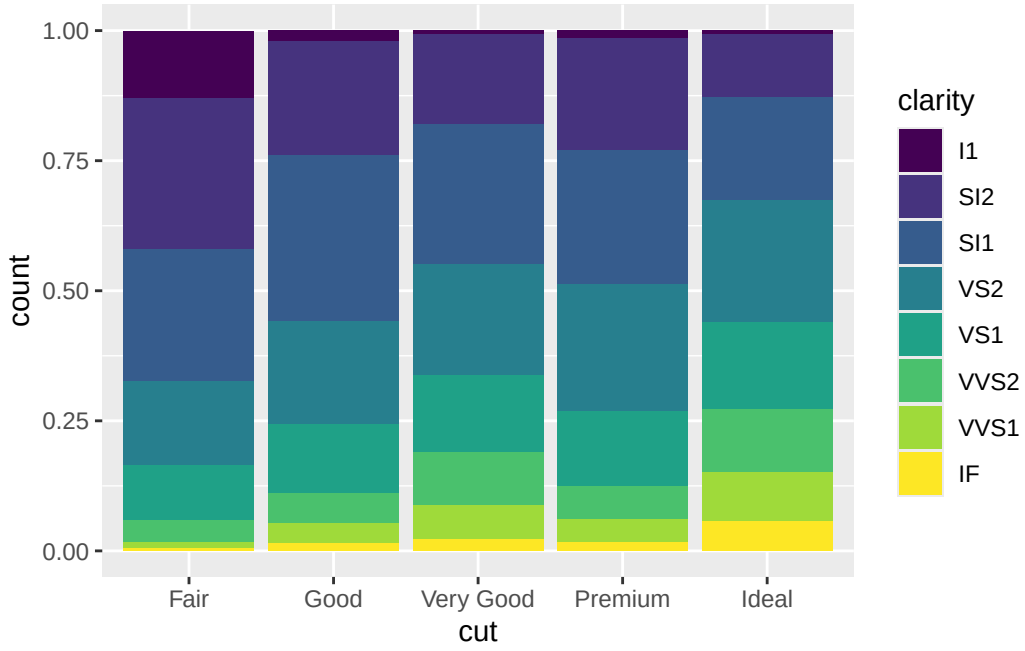



```
# default
```

`geom_bar` can break the categorical variable on X by another variable. The default position of bars is **stack**. Absolute counts of the values of the second categorical variable are stacked on top of each other. The total height of the bar is the sum of all counts.

12 Bar positions with another categorical variable: `fill`

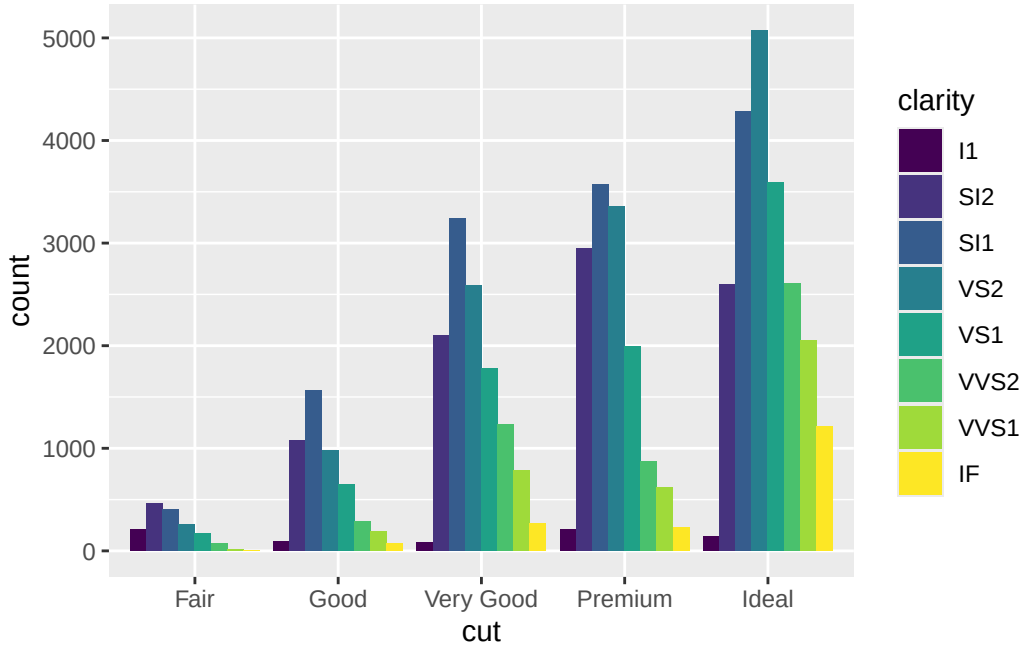
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "fill")
```



Another option is **fill**. Do not confuse with the color fill aesthetic scale. This shows stacked **proportions** of the second categorical variable. The total height of the bar is always 1 (i.e. 100%). The Y- axis still says "count", but it is actually proportion.

13 Bar positions with another categorical variable: `dodge`

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

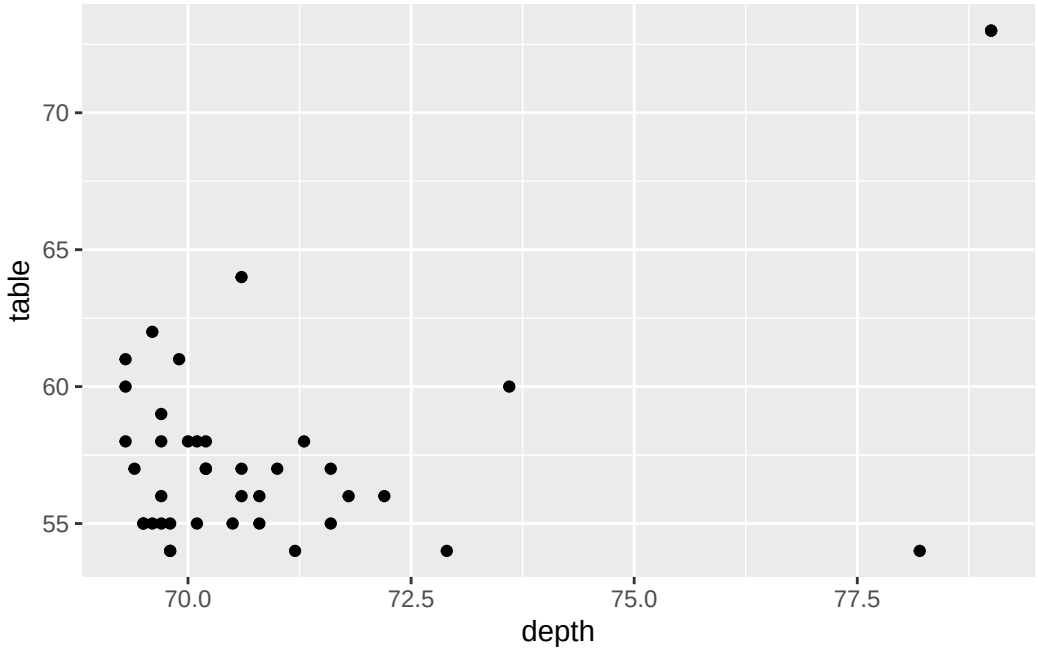


The last bar position option is `dodge`. It shows the absolute counts again, with each value of the second categorical variable in its own bar. The bars are grouped together by the first categorical variable represented by the X-axis.

14 Scatterplot

- best to explore the association between two continuous variables

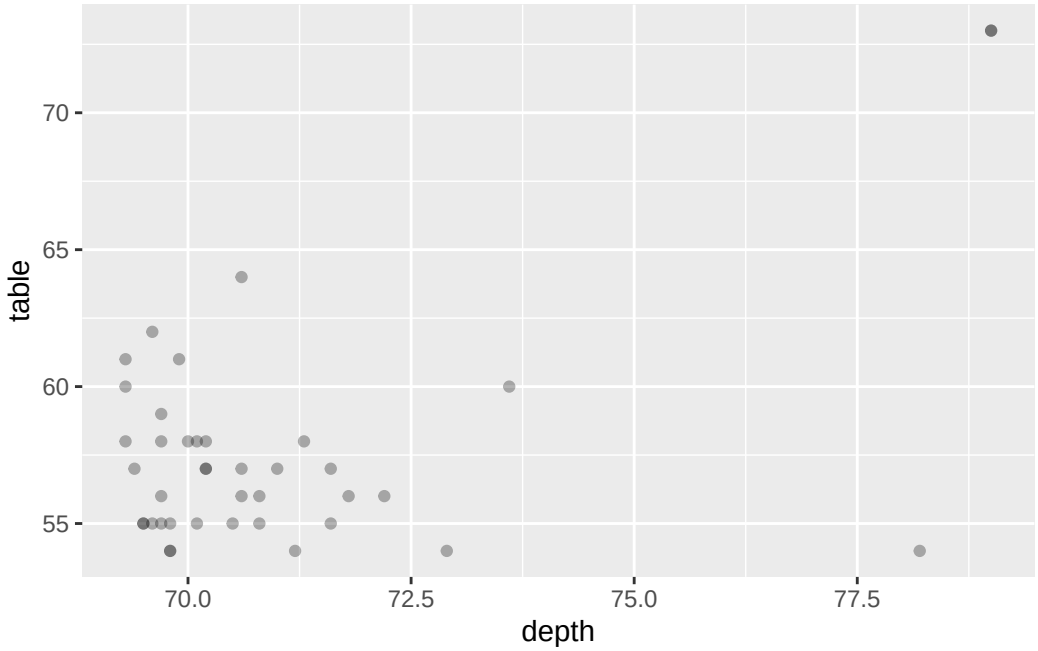
```
biggest_diamonds <- slice_max(diamonds, order_by = depth, n = 40)
ggplot(data = biggest_diamonds) +
  geom_point(mapping = aes(x = depth, y = table) )
```



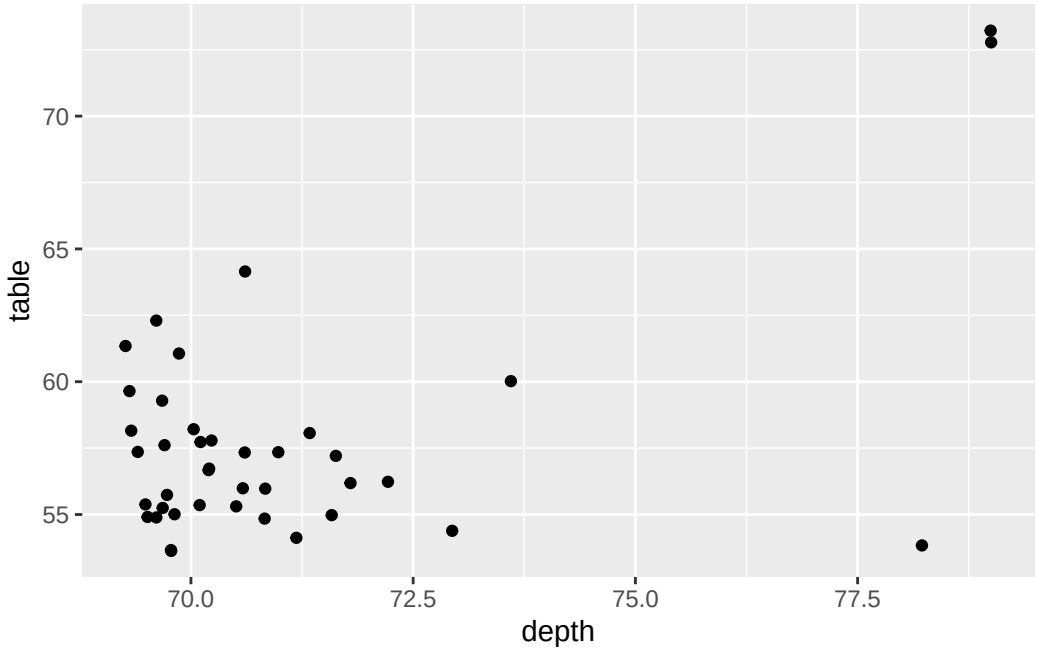
A selection of forty diamonds with the largest depth. We cannot see much association, can we? Clearly there are diamonds with high depth and small table, as well as the other way round, as well as some very variable balance between both. At the same time, we see 37 of 40 points because of overlap which makes the dataset looks smaller than it is and would easily obscure trends for you. Too many overlapping points cause **overplotting**. The next slides show strategies to deal with overplotting.

15 Alleviate overplotting with alpha and jitter

```
ggplot(biggest_diamonds) +
  geom_point(aes(x = depth,
                 y = table),
             alpha = 0.3)
```




```
set.seed(2525) # reproducible random numbers
ggplot(biggest_diamonds) +
  geom_point(aes(x = depth,
                 y = table), position = "jitter")
```

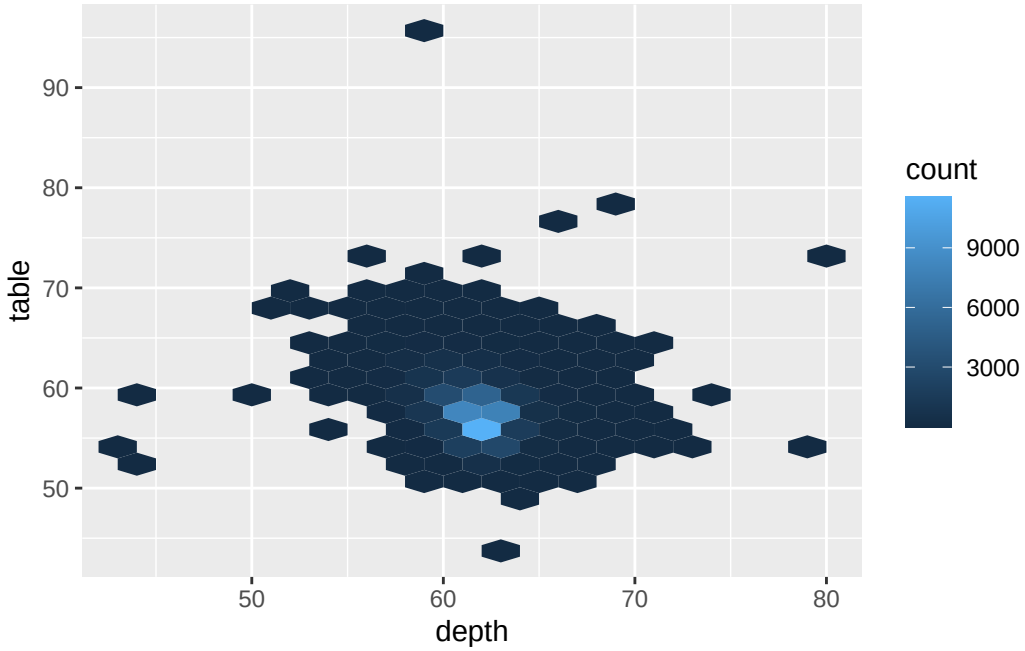


alpha is one of the aesthetic scales, just like color, the X and Y axes, or shape. It controls the transparency. It takes values between 0 and 1. Alpha at 0.1 means that each point has only a 10%-visibility. Jittering is a technique that adds small random noise to each value. Without jittering, some points lie on a line. Jittering will scatter them a bit.

`set.seed` is a function you use when you compute something with random numbers but want them to be reproducible; that is, you want the same random numbers every time you run this script. The function wants a random number. Here it makes sure that the jittered points will always be positioned like you see in the figure. You can set a seed, run a function with random numbers, and if you do not like the result, re-run it with a different seed. Iterate until you are happy with the result.

16 Overplotting reduction: Scatterplot with hexagonal bins

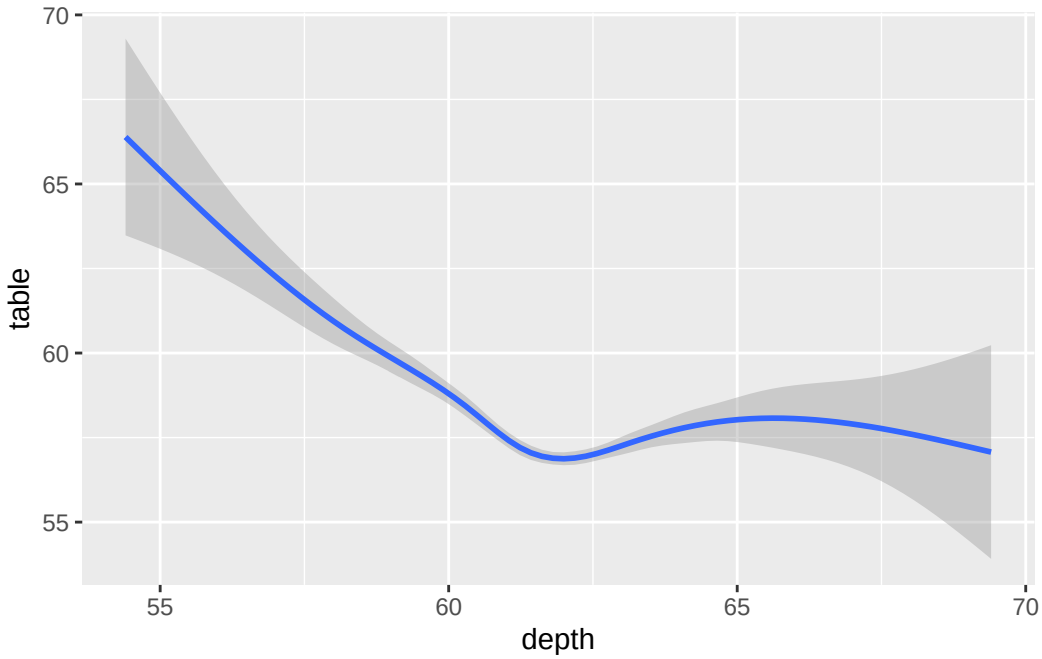
```
ggplot(data = diamonds) + geom_hex(aes(x = depth, y = table),  binwidth = 2)
```



This plot automatically introduces color for the `count` variable it computes. The lighter the color, the more frequent the value. The points are no longer individual observations but observations within an interval, just like with histograms. You can adjust the number of bins or their width.

17 Overplotting reduction: regression models

```
set.seed(559900)
diamonds_sample <- sample_n(tbl = diamonds, size = 1000)
ggplot(data = diamonds_sample) +
  geom_smooth(aes(x = depth, y = table), method = "gam", se = TRUE )
```



When you want to see trends rather than individual points, use `geom_smooth`. It implements various regression models. These models use algorithms to draw a line (straight or curvy) among the data points so that the line is as close to each point as possible (they minimize the sum of distances between each data point and the line). You can have the line embedded in a ribbon controled by `se=TRUE/FALSE`. The ribbon width means: "With this amount of data, the line could be shaped anywhere within this span." The narrower the line, the stronger the trend (and the more reliable your model if you want to use it to make predictions).

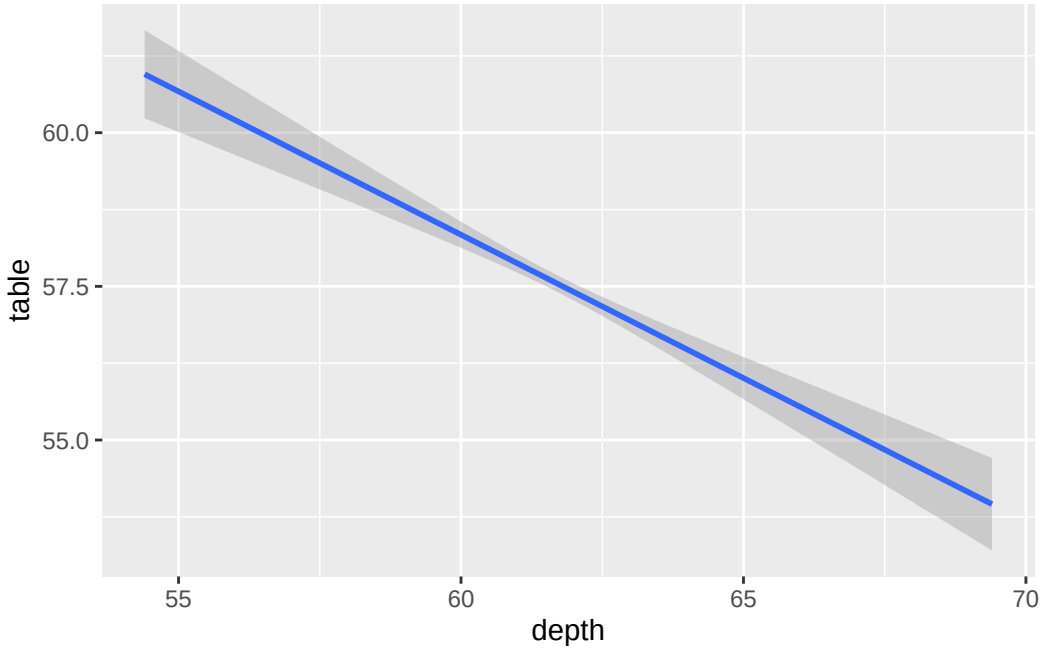
`gam` and `loess` make curves; the other algorithms make lines. `loess` can only be used with small data (small thousands of data points), otherwise it freezes your R session.

This dataset consists of 1,000 random diamonds. The complete `diamonds` dataset is so huge that the models will be very well backed and you will hardly be able to see the ribbon.

Models are built with formulas. The default formula in geom_smooth says `predicted values ~ observed values` . You do not need to write this. When you do not, ggplot2 will just throw a message that it is using this formula.

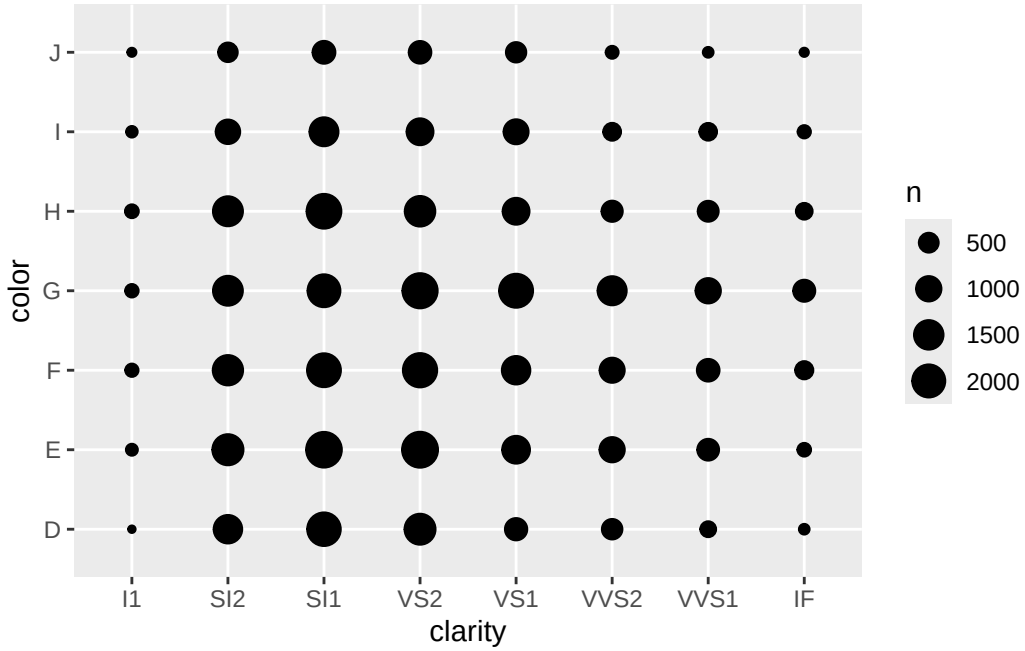
18 Linear regression model

```
ggplot(data = diamonds_sample) +
  geom_smooth(aes(x = depth, y = table), method = "lm", se = TRUE, formula =
↪  y~x )
```



19 Scatterplots with discrete variables

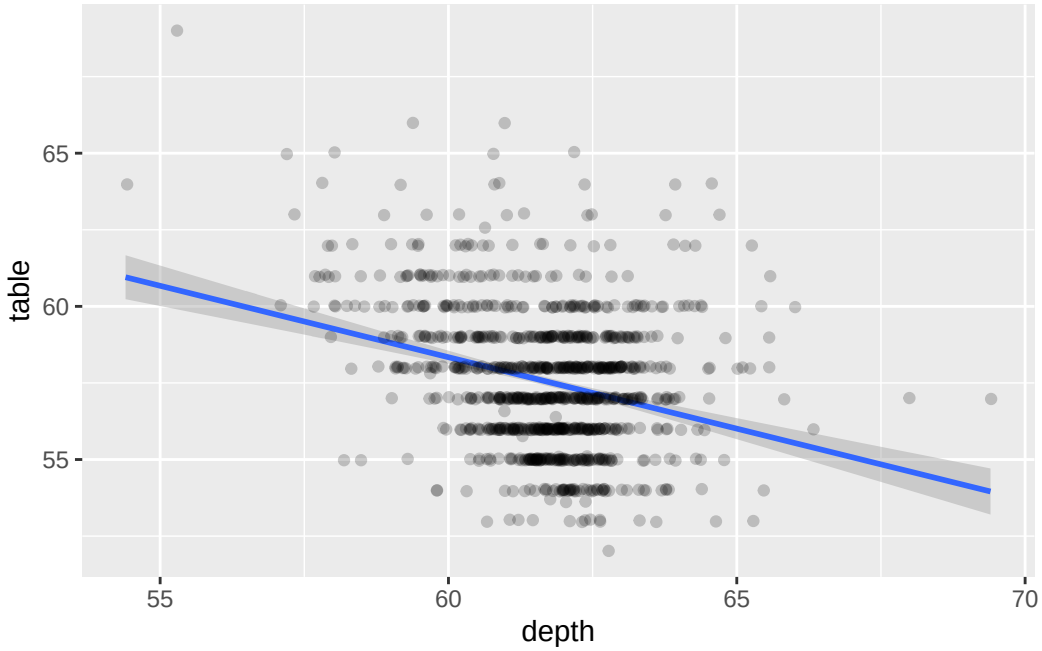
```
ggplot(data = diamonds, mapping = aes(x = clarity, y = color)) +
  geom_count( )
```



`geom_count` is a variant of `geom_point` that computes the count of each combination of X and Y values and maps it on point size.

20 Several geoms in one plot: smooth and scatterplot

```
ggplot(data = diamonds_sample, mapping = aes(x = depth, y = table)) +
  geom_smooth( method = "lm", se = TRUE, formula = y~x ) +
  geom_point(alpha = 0.2, position = "jitter")
```

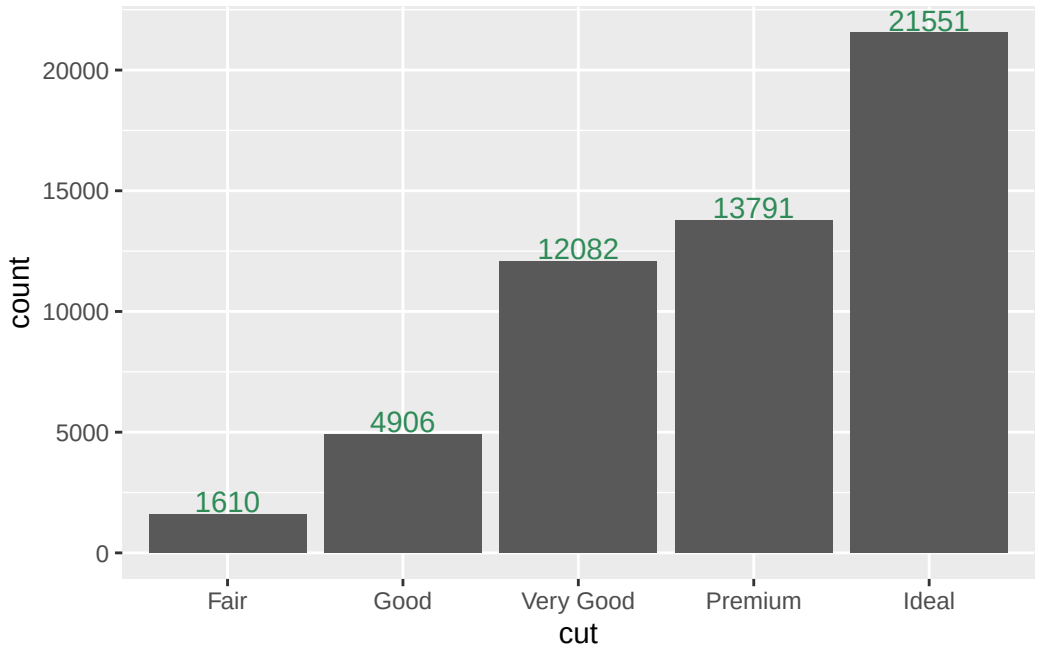


You may still want to have a visual idea of how the amount of near data points corresponds to the width of the ribbon or so. Both geoms look inside the `ggplot` function to find the data and mapping aesthetics. Each geom first searches them inside itself and then in `ggplot`. So if you have one aesthetic mapping inside ggplot and another in a geom, that geom will work according to what it has inside itself. If you want several geoms to share something, put it into `ggplot`, never just in one of the geoms, because geoms cannot look inside each other.

21 Combination with `geom_text`

- `geom_text` is typically used like a scatterplot with labels instead of points.
- or to label bars in a bar plot:

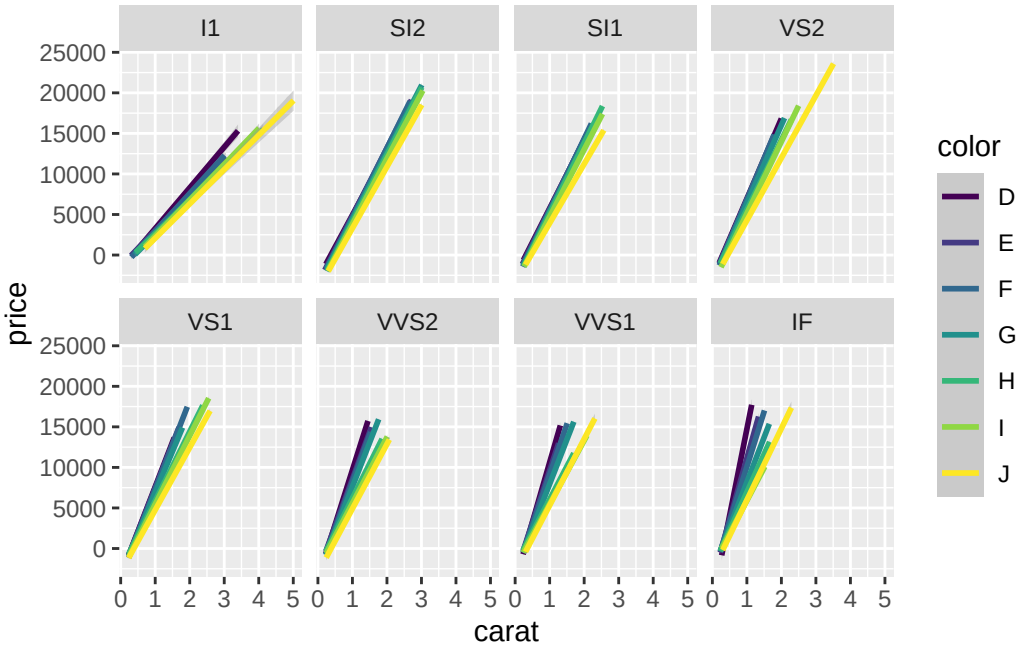
```
ggplot(diamonds, aes(x = cut)) +
  geom_bar() +
  geom_text(
    aes(label = after_stat(count)),
    stat = "count",
    vjust = -0.1,# sligthly above the bar
    color = "seagreen"
  )
```



22 Facets (wrap)

- subgraphs: yet another way to break by a categorical variable

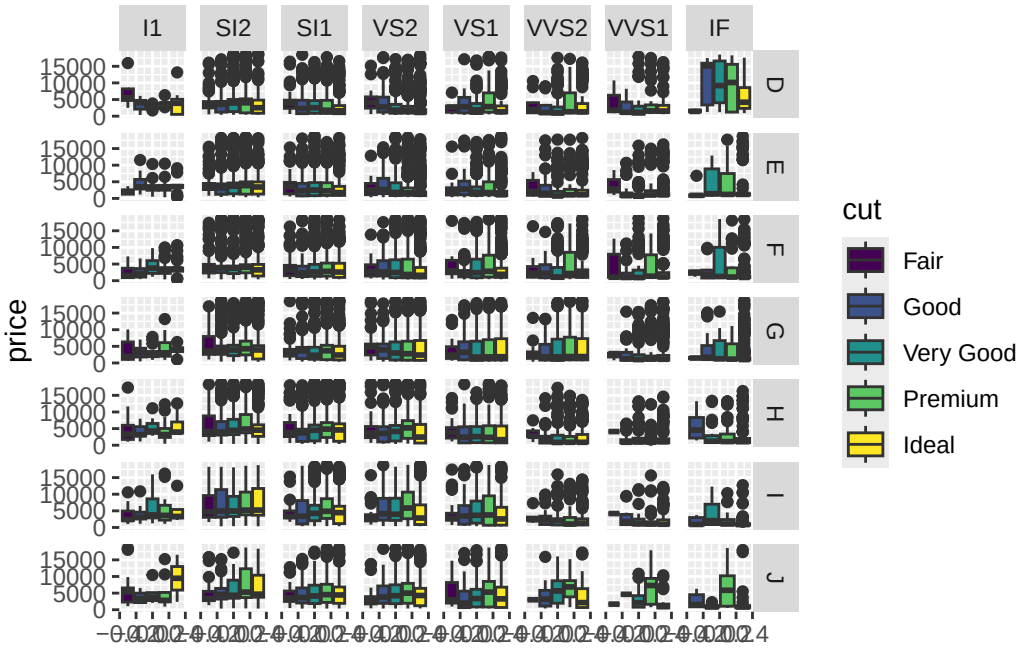
```
ggplot(data = diamonds) +
  geom_smooth(mapping = aes(x = carat, y = price, color = color),
              method = "lm", formula = y ~ x) +
  facet_wrap(~ clarity, ncol = 4)
```



Facets are a separate layer of graphics, although they act like another aesthetic scale. Two facet functions: `facet_wrap` and `facet_grid`. Use `facet_wrap` when you want to break the data by one such variable. The notation is `~ <that variable>`. You can also say how many rows or columns you want (here 4, it did 3 by default).

23 Facets (grid)

```
ggplot(data = diamonds) + geom_boxplot(aes(y = price, fill = cut)) +
  facet_grid(color ~ clarity)
```



When you want to break the data into subgraphs by two categorical variables, use `facet_grid`. The first-named variable will be described by rows, the second after `~` will be described by columns.