basic-visualisaton

After loading the tidyverse package, we will have access to the `diamonds` dataframe. We can get more info about the dataset by using the `?diamonds` command.

```
data(diamonds)
diamonds
```

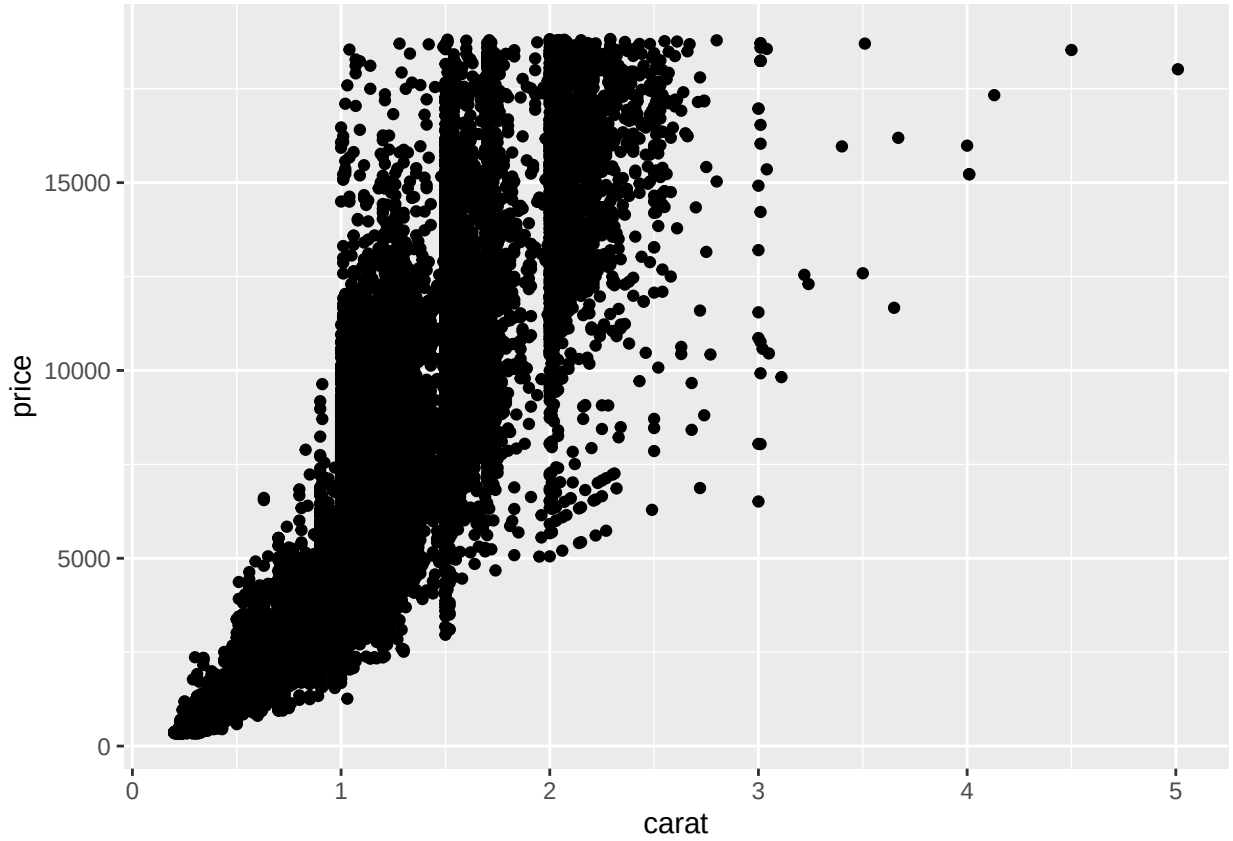
```
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  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     4.05  2.39
## # ... with 53,930 more rows
```

```
summary(diamonds)
```

```
##      carat               cut        color        clarity          depth      
##  Min.   :0.2000   Fair      : 1610   D: 6775   SI1    :13065   Min.   :43.00  
##  1st Qu.:0.4000   Good      : 4906   E: 9797   VS2    :12258   1st Qu.:61.00  
##  Median :0.7000   Very Good:12082   F: 9542   SI2    : 9194   Median :61.80  
##  Mean   :0.7979   Premium   :13791   G:11292   VS1    : 8171   Mean   :61.75  
##  3rd Qu.:1.0400   Ideal     :21551   H: 8304   VVS2   : 5066   3rd Qu.:62.50  
##  Max.   :5.0100                     I: 5422   VVS1   : 3655   Max.   :79.00  
##                                     J: 2808   (Other): 2531                  
##      table           price             x                y         
##  Min.   :43.00   Min.   :  326   Min.   : 0.000   Min.   : 0.000  
##  1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710   1st Qu.: 4.720  
##  Median :57.00   Median : 2401   Median : 5.700   Median : 5.710  
##  Mean   :57.46   Mean   : 3933   Mean   : 5.731   Mean   : 5.735  
##  3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540   3rd Qu.: 6.540  
##  Max.   :95.00   Max.   :18823   Max.   :10.740   Max.   :58.900  
##                                                                   
##        z         
##  Min.   : 0.000  
##  1st Qu.: 2.910  
##  Median : 3.530  
##  Mean   : 3.539  
##  3rd Qu.: 4.040  
##  Max.   :31.800  
##
```

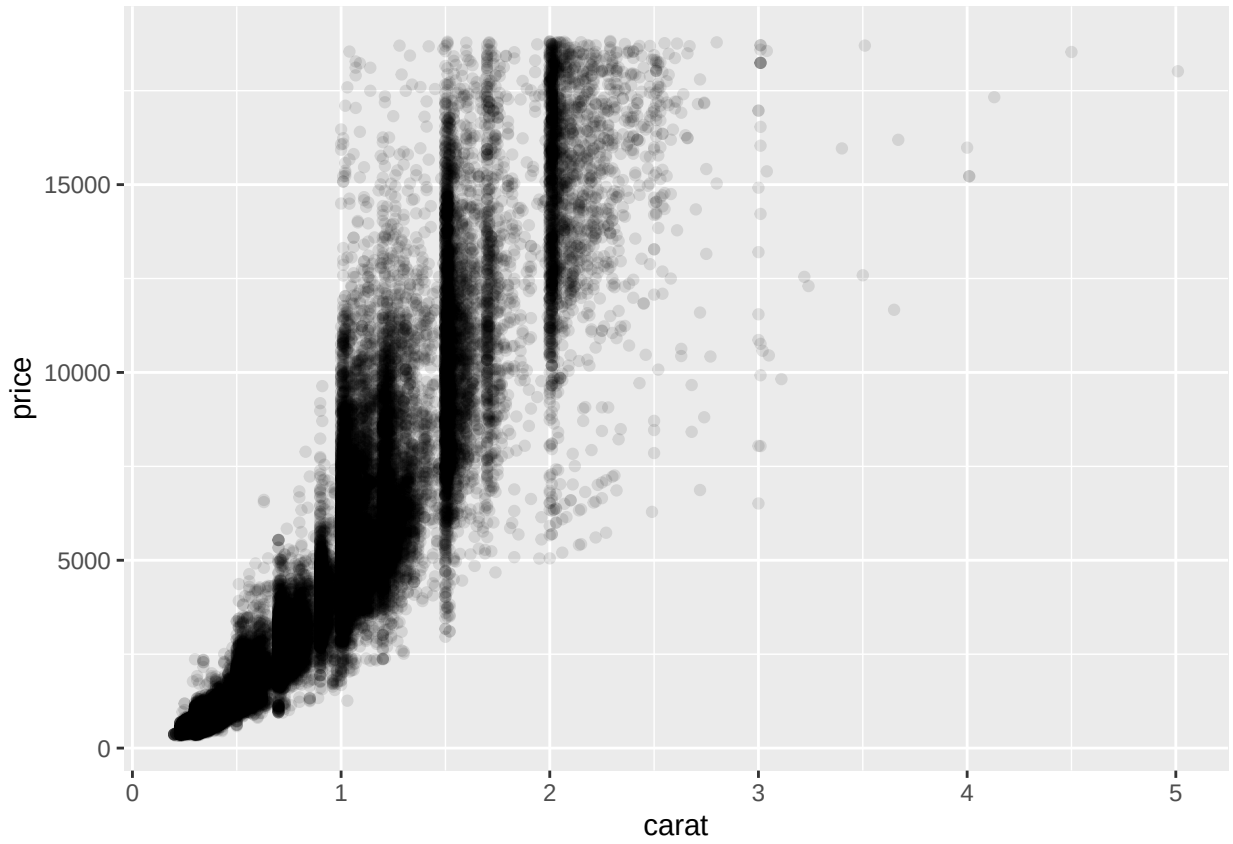
We see a colomn named `carat`, standing for the weight of the diamond, and a column `price`. We can also see, that we have 53k rows, which is a lot. Lets have a first look at the data.

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



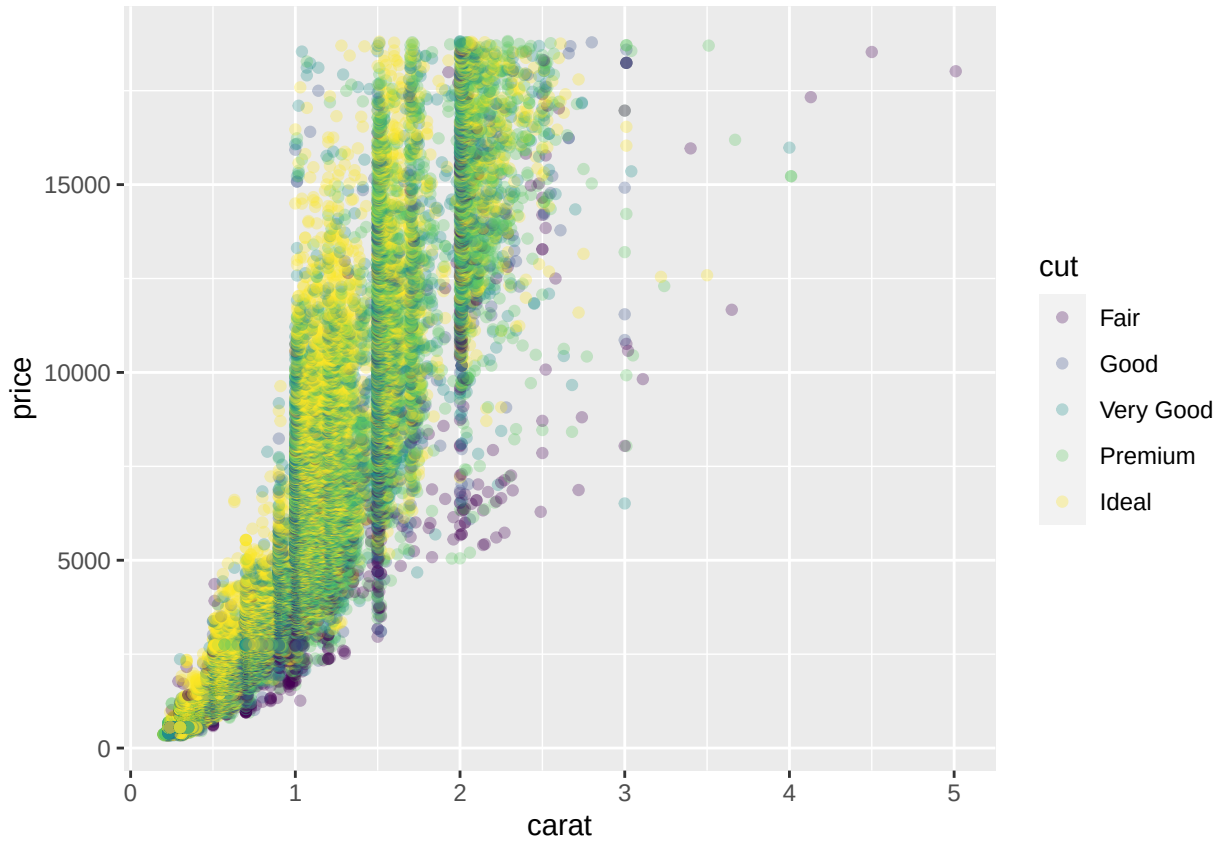
That's a lot of points. Because there are 53k points, probably a lot will overlap. Let's make the points a bit transparent.

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



That's better. At least we get some idea about the distribution. Let's try to zoom in a bit more. We see that another column is called `cut` which stands for the quality of the cut. This will probably have some impact. Lets use colors to see if that makes sense.

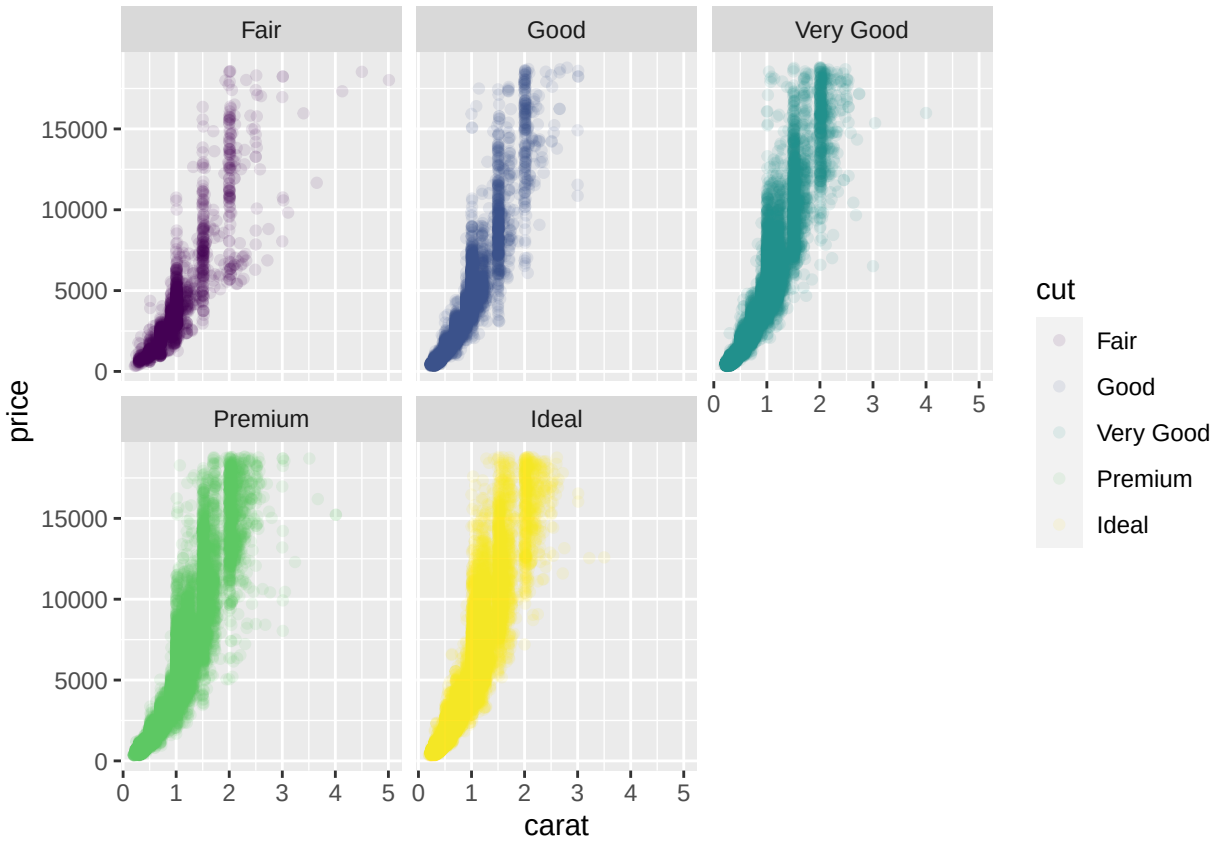
```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point(alpha = 0.3, aes(color = cut))
```



Ok, that sort of seems to work. We can see the purple teint at the bottom, and the yellow more at the top. This corresponds with `Fair` and `Ideal`. That seems to make sense: ideal cut diamonds are more expensive.

But the plot is still to crowded. We could use a facet wrap to split the colors:

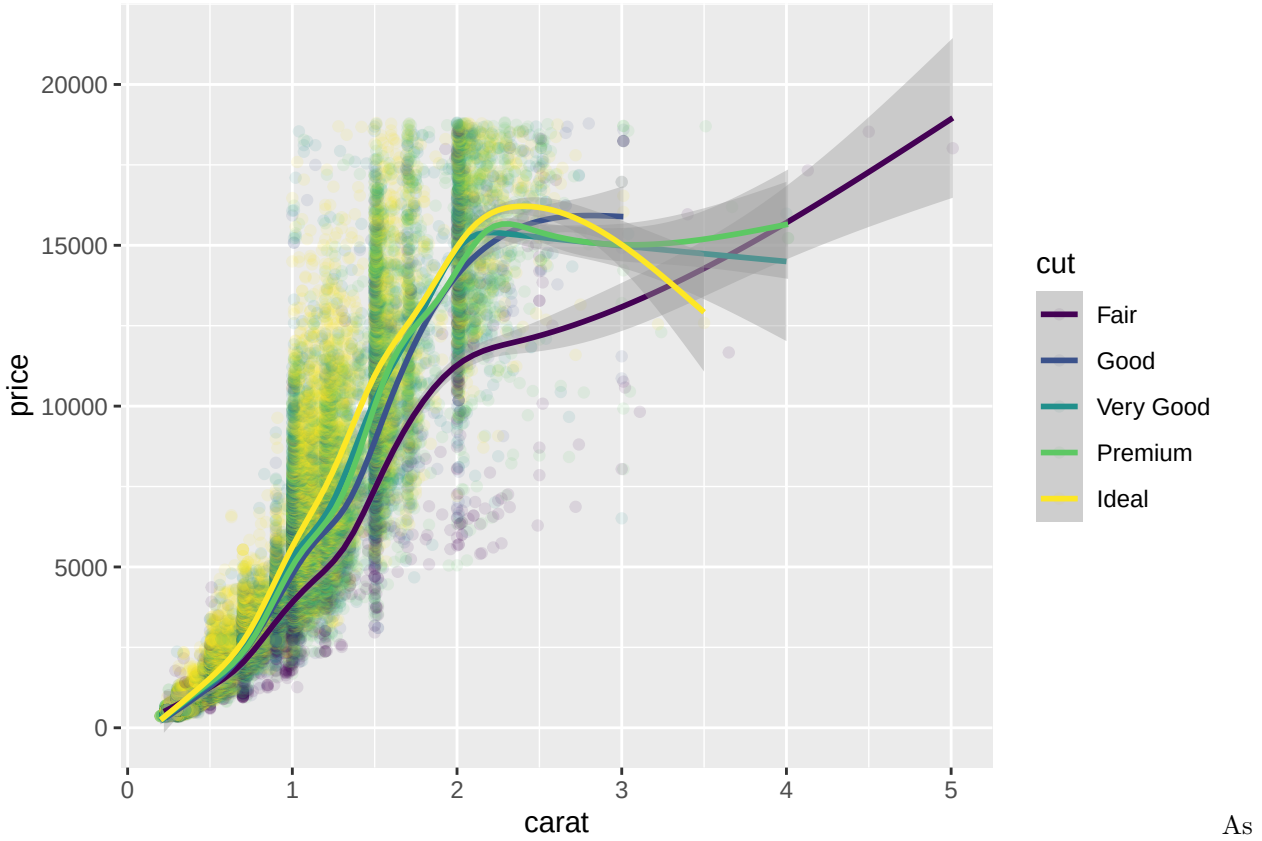
```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point(alpha = 0.1, aes(color = cut)) +
  facet_wrap(~cut)
```



The different cuts are separated. But now we have a harder time to compare the plots. What we want is to reduce the points to a simpler metric. We can use `geom_smooth` for that:

```
ggplot(diamonds, aes(x = carat, y = price, color = cut)) +
  geom_point(alpha = 0.1) +
  geom_smooth()
```

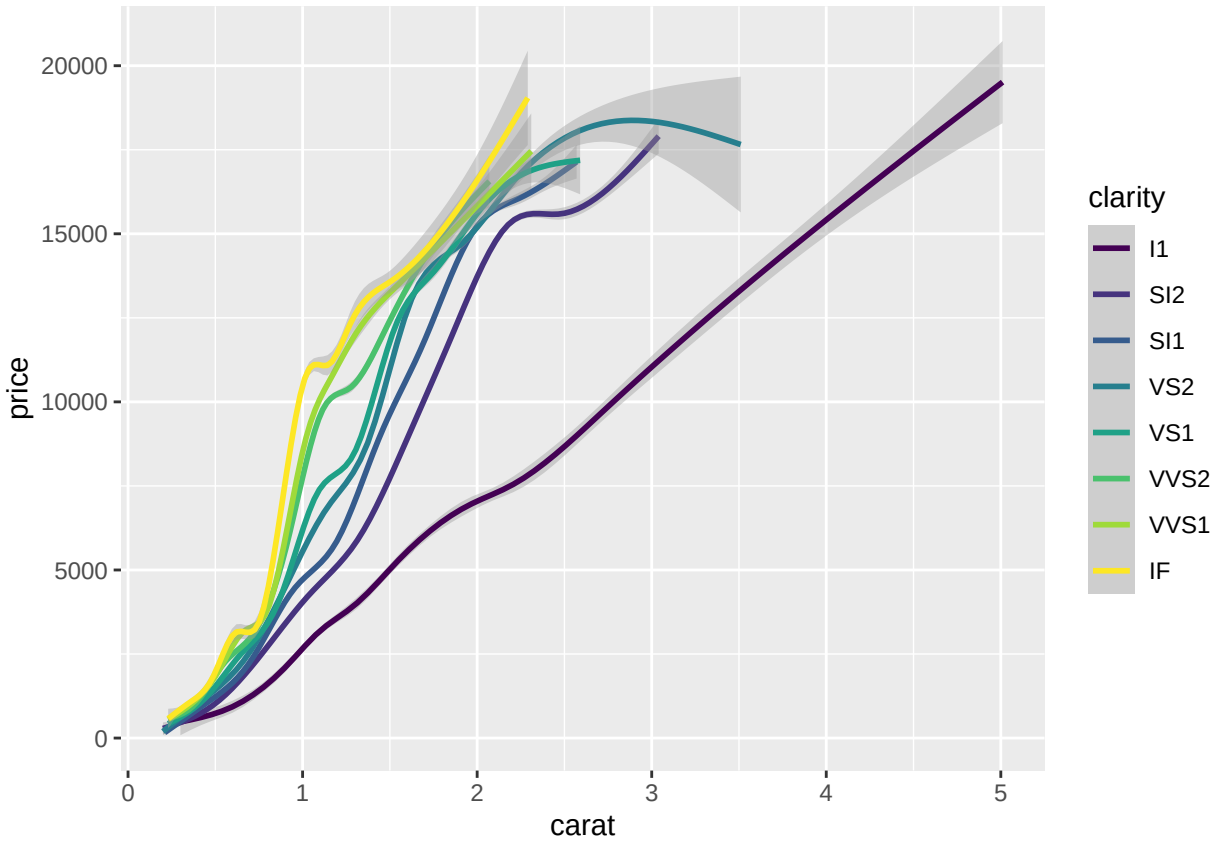
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



As you can see, we can just stack plots onto eachother with the `+`. Ok, maybe things are still a bit overcrowded, so let's just stick to the `geom_smooth`

```
diamonds %>%
  ggplot(aes(carat, price, color = clarity)) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



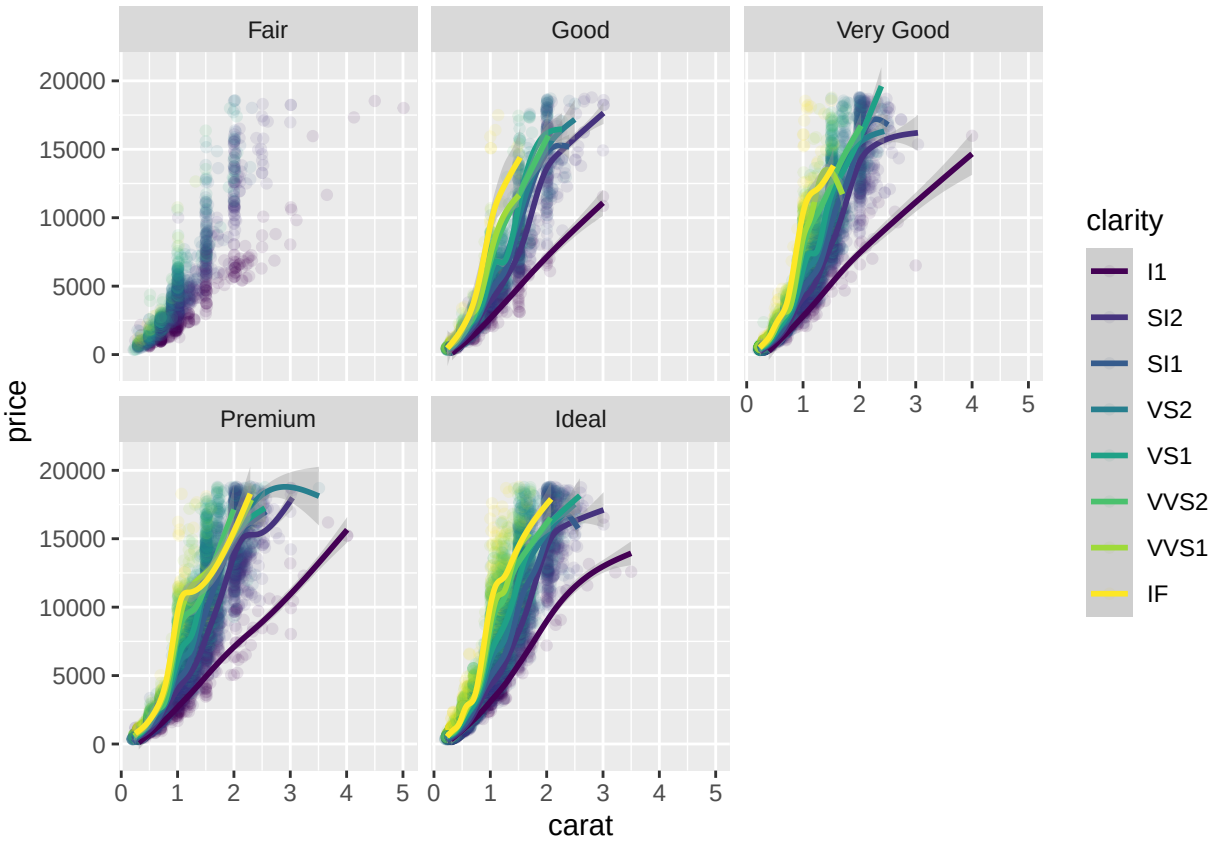
This is nice. We have a clear view of what we expected. And we can even see errorbars (the grey shaded area) around the lines where there is less data to be certain.

Now let's try to find more properties that influence the price. We have `clarity`. That might influence the price, too. Let's use the plot we had, and add a facet wrap to split on clarity.

```
diamonds %>%
  ggplot(aes(carat, price, color = clarity)) +
  geom_point(alpha = 0.1) +
  geom_smooth() +
  facet_wrap(~cut)
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

```
## Warning: Computation failed in `stat_smooth()`:
## x has insufficient unique values to support 10 knots: reduce k.
```

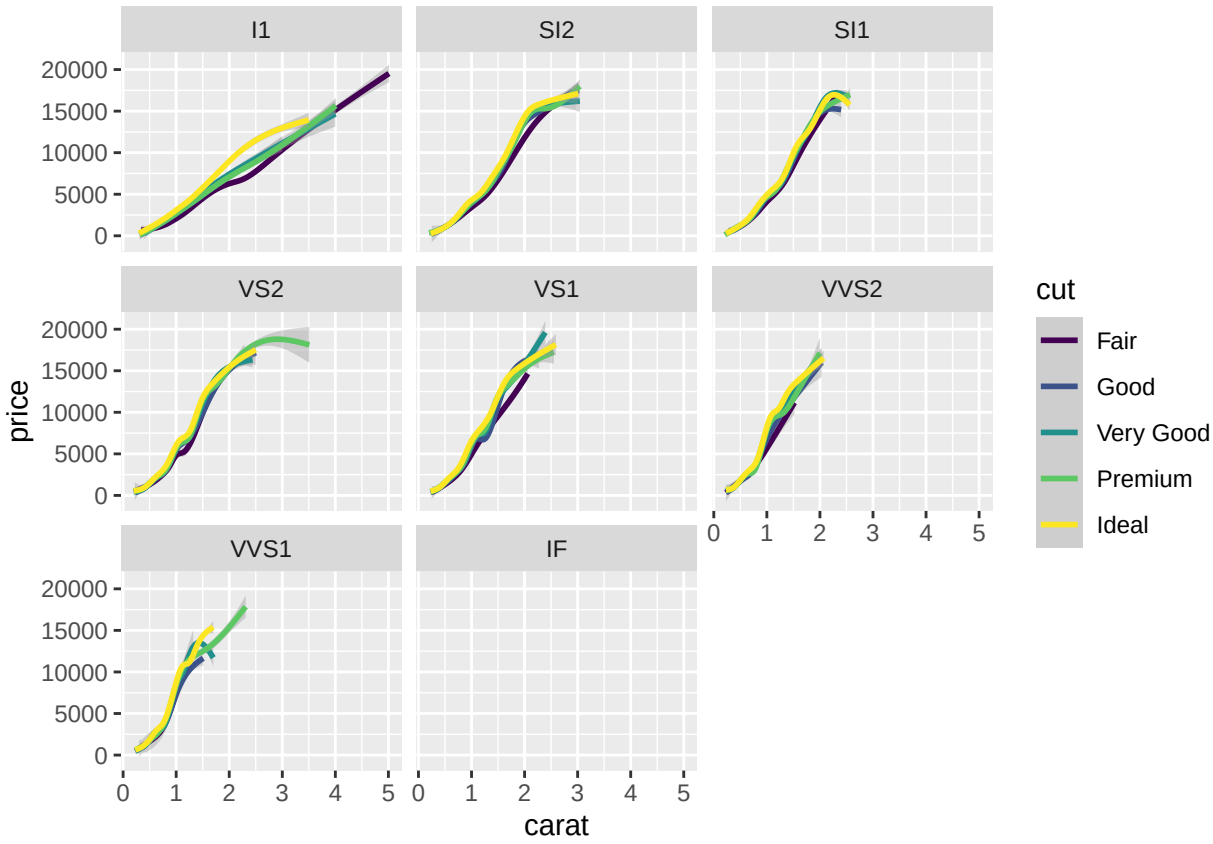


We get an error for smooth. We seem to miss one of the clarity categories for smooth in the `Fair` cut, so we get a warning there, and not line. But like this, it is kind of hard to conclude if the same clarity of a `Good` cut has an impact on the price, when compared to an `Ideal` cut. So lets switch the facet wrap.

```
diamonds %>%
  ggplot(aes(carat, price, color = cut)) +
  geom_smooth() +
  facet_wrap(~clarity)
```

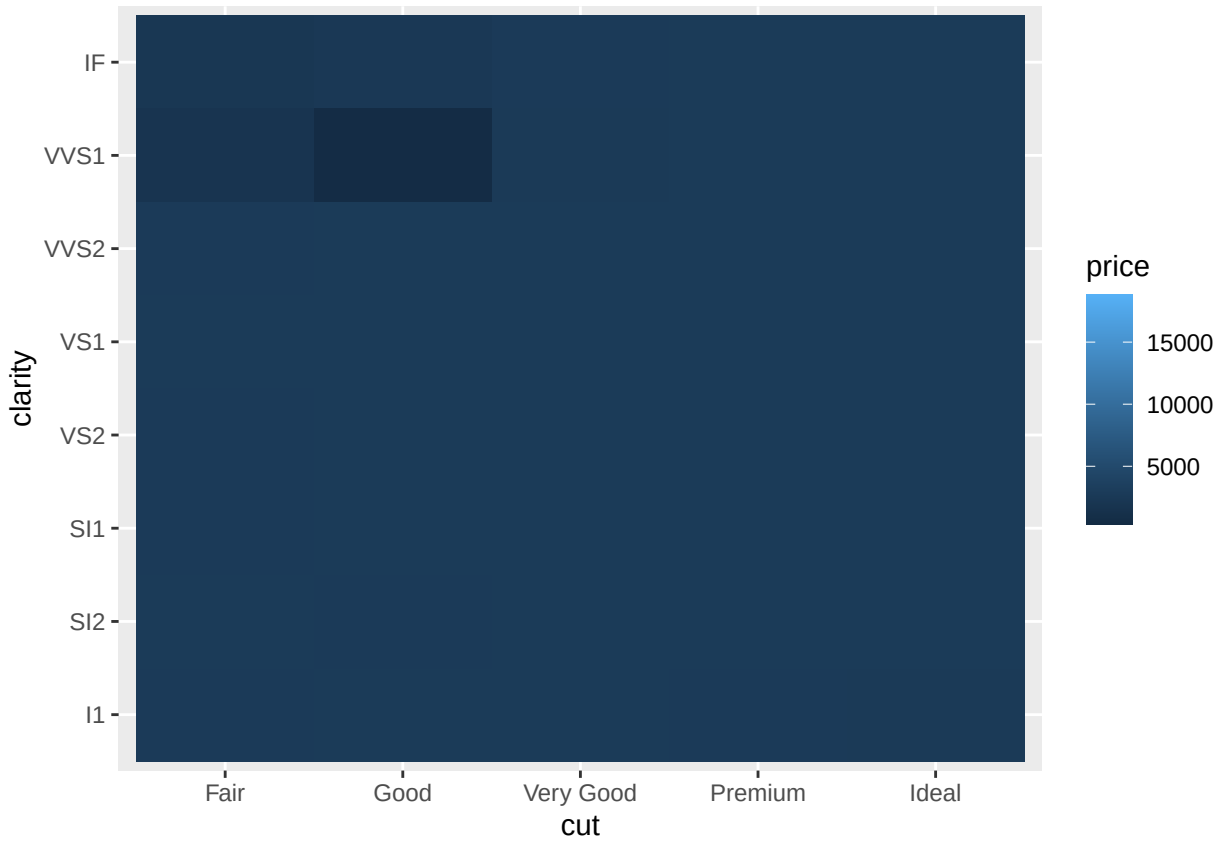
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

```
## Warning: Computation failed in `stat_smooth()`:
## x has insufficient unique values to support 10 knots: reduce k.
```



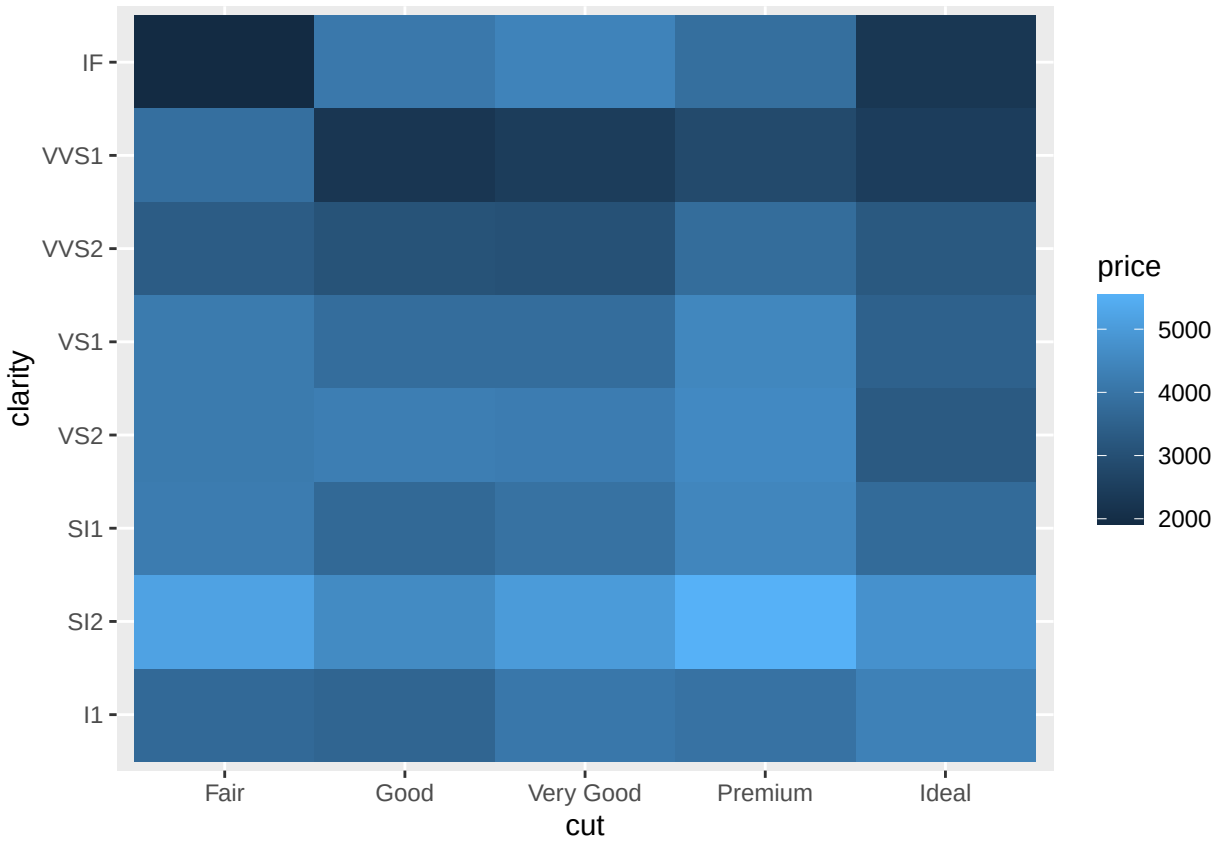
We can see that the impact of the cut is much smaller then the impact of clarity. Now let's try a different type of visualisation, the heatmap.

```
diamonds %>%
  ggplot(aes(cut, clarity, fill = price)) +
  geom_raster()
```



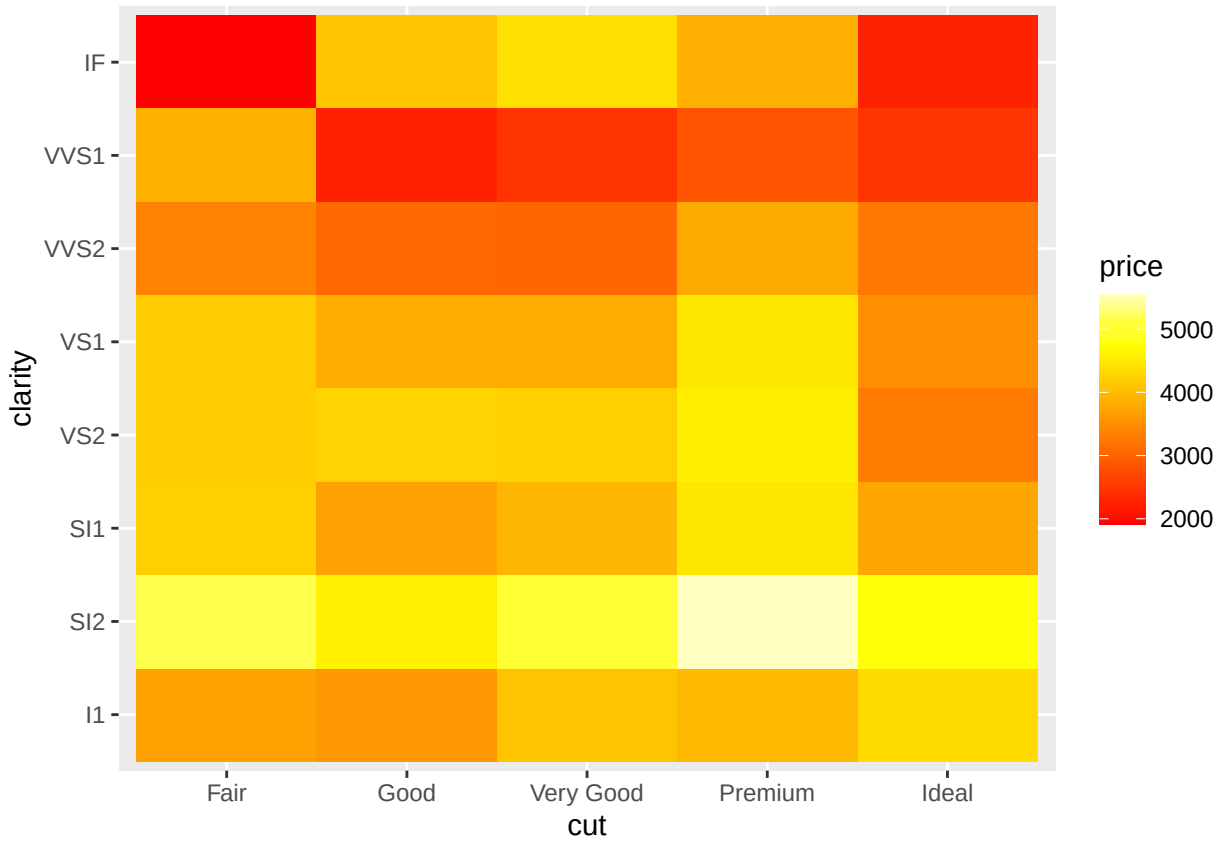
Oh, this doesnt seem to work. What is going wront? Well, we specified the fill, but actually, there is no single value for the fill. We want some type of grouped value for the fill, like the mean. So we first have to summarise the data. What we want, is a mean for every cut-clarity group.

```
p <- diamonds %>%
  group_by(cut, clarity) %>% # we make groups for every cut x clarity combo
  summarise(price = mean(price)) %>% # and add a mean price. Summarise follows the groups.
  ggplot(aes(cut, clarity, fill = price)) +
  geom_raster()
p
```



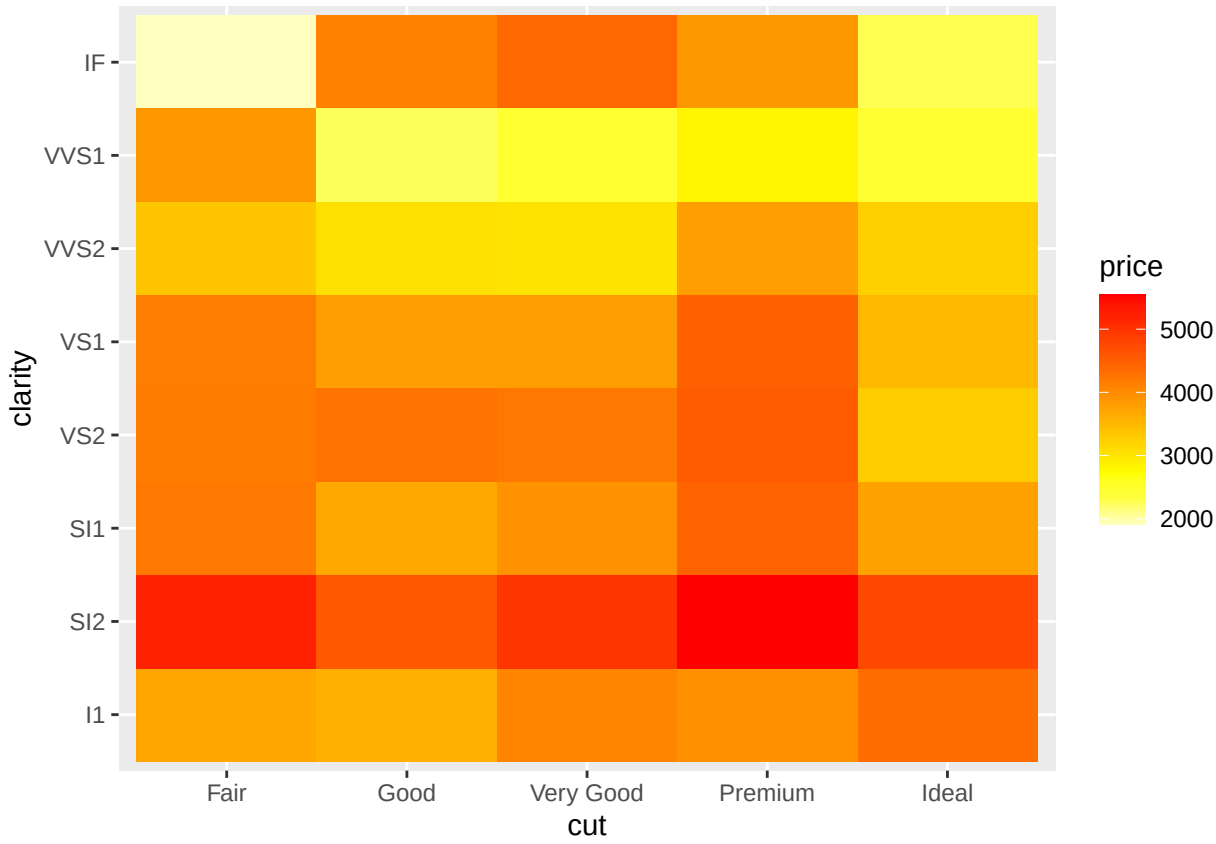
This is a bit more like what we want. Maybe the colors aren't really satisfactory. So let's define our own set of gradient colors instead of the default.

```
p + scale_fill_gradientn(colours = heat.colors(10)) # change colors
```



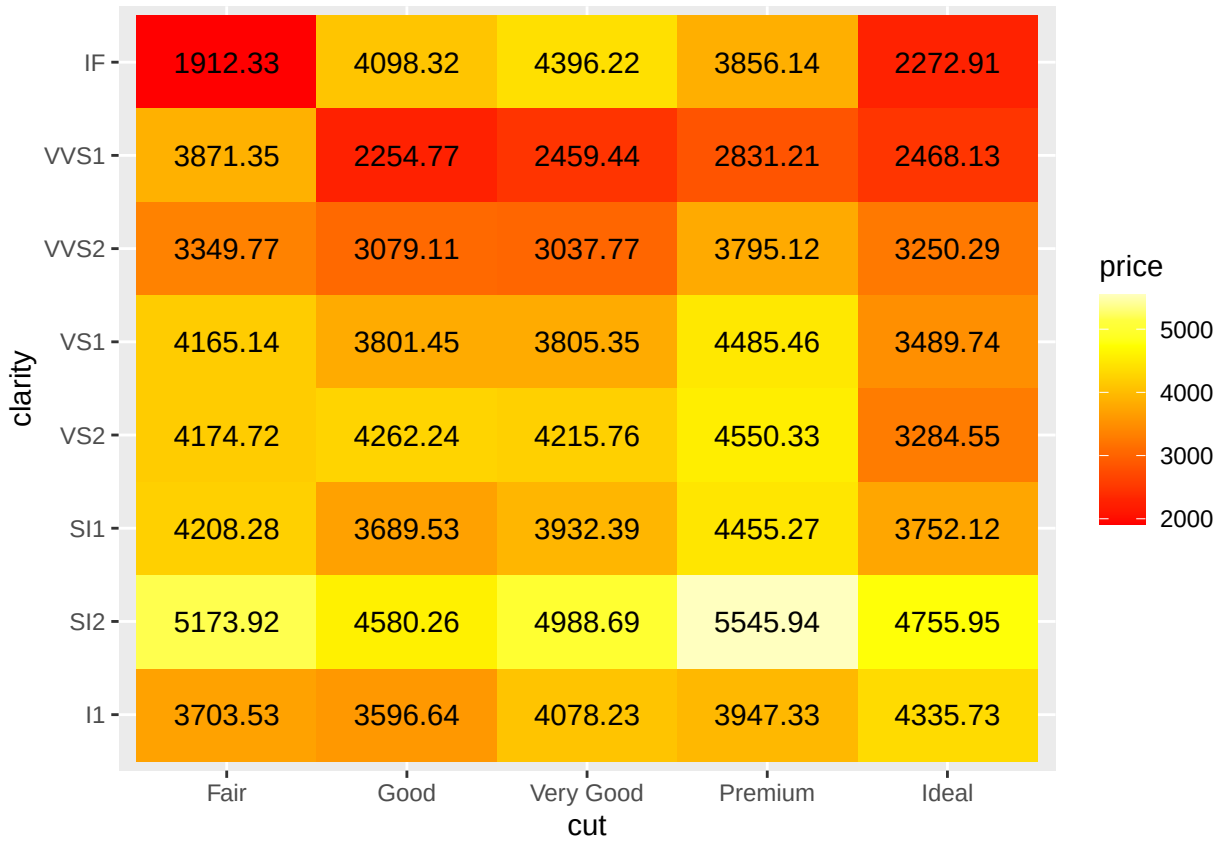
Ok, this looks more like a real heatmap. Maybe reverse the colors, so that red is warmer.

```
p + scale_fill_gradientn(colours = rev(heat.colors(10))) # change colors
```



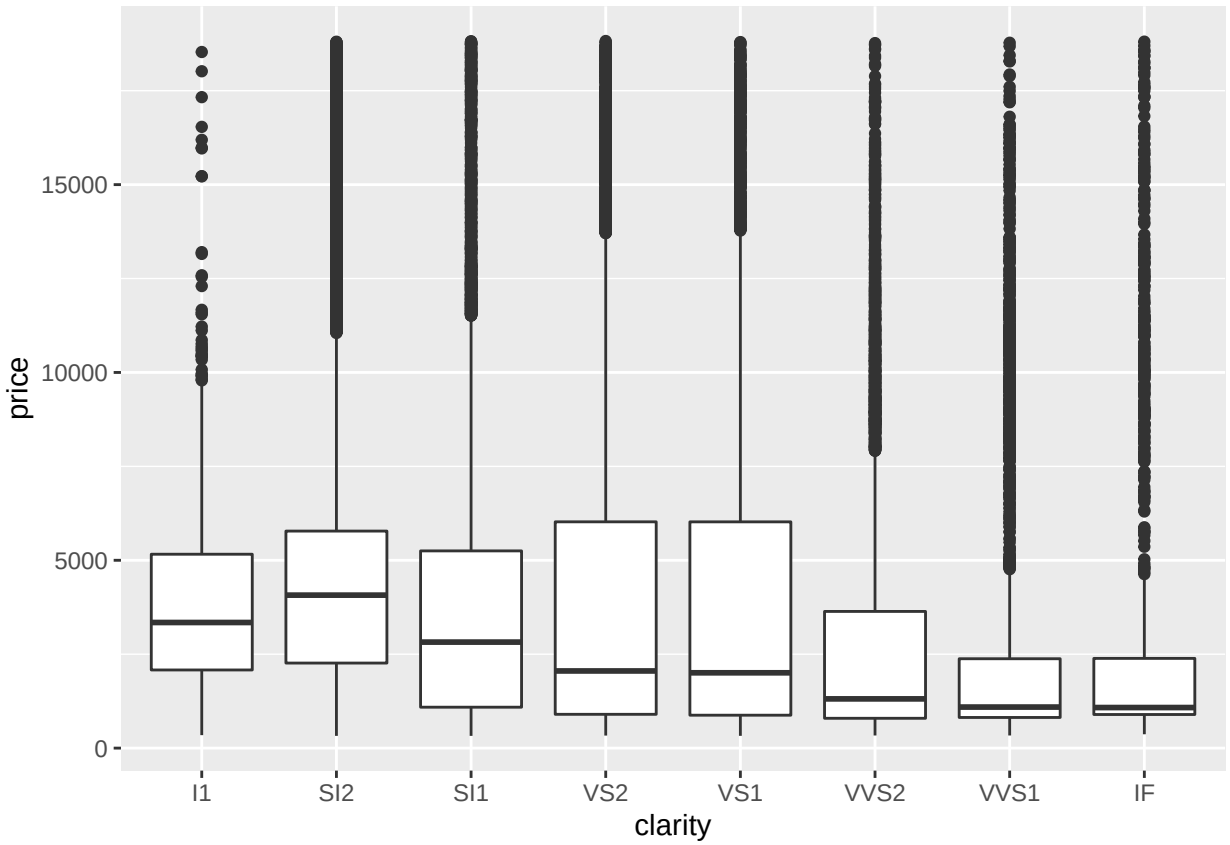
We might wonder what those mean values are. Well, let's add some text to this plot.

```
p + scale_fill_gradientn(colours = heat.colors(10)) + # change colors
  geom_text(aes(label = round(price, 2)))  # add text
```



This gives us a nice idea. But the mean is very sensitive to outliers. We could wonder what the distribution actually is, for every category. So, we want to look at the distribution of the prices in every clarity group. Let's put the clarity on the x-axis, and the prices on the y-axis, with a boxplot.

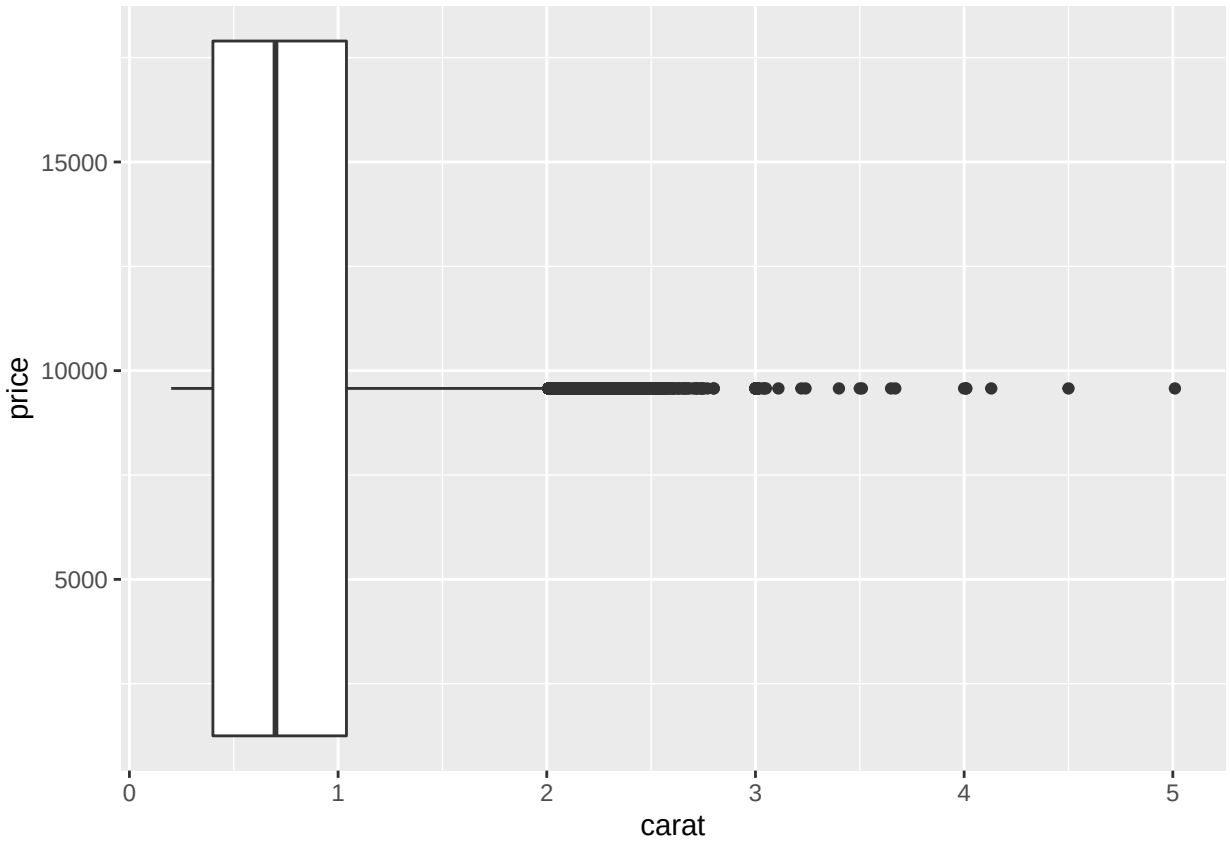
```
diamonds %>%
  ggplot(aes(clarity, price))+
  geom_boxplot()
```



Well, that's something you might not have expected. What's going on? The carat has the biggest impact on price, right? So we might want to put the caret on the x-axis. But we run into problems if we do that just like that:

```
diamonds %>%
  ggplot(aes(carat, price)) +
  geom_boxplot()
```

```
## Warning: Continuous y aesthetic -- did you forget aes(group=...)?
```



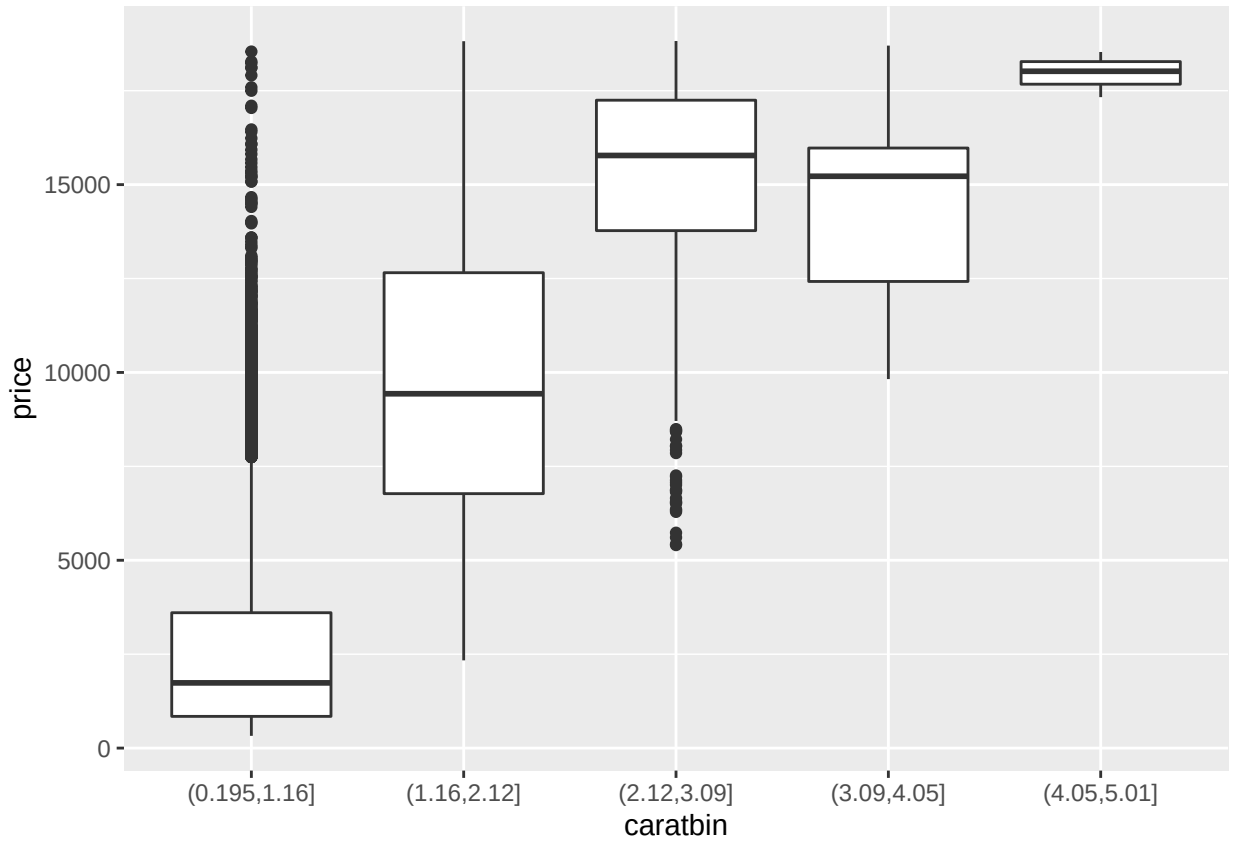
What's going on? `carat` is a continuous variable. If we want to make boxplot, we need groups on which we want to calculate a boxplot. So let's make our own bin's on the x-axis to fix this.

```
diamonds %>%
  mutate(caratbin = cut(carat, breaks = 5)) %>%
  distinct(caratbin)
```

```
## # A tibble: 5 x 1
##   caratbin
##   <fct>
## 1 (0.195,1.16]
## 2 (1.16,2.12]
## 3 (2.12,3.09]
## 4 (3.09,4.05]
## 5 (4.05,5.01]
```

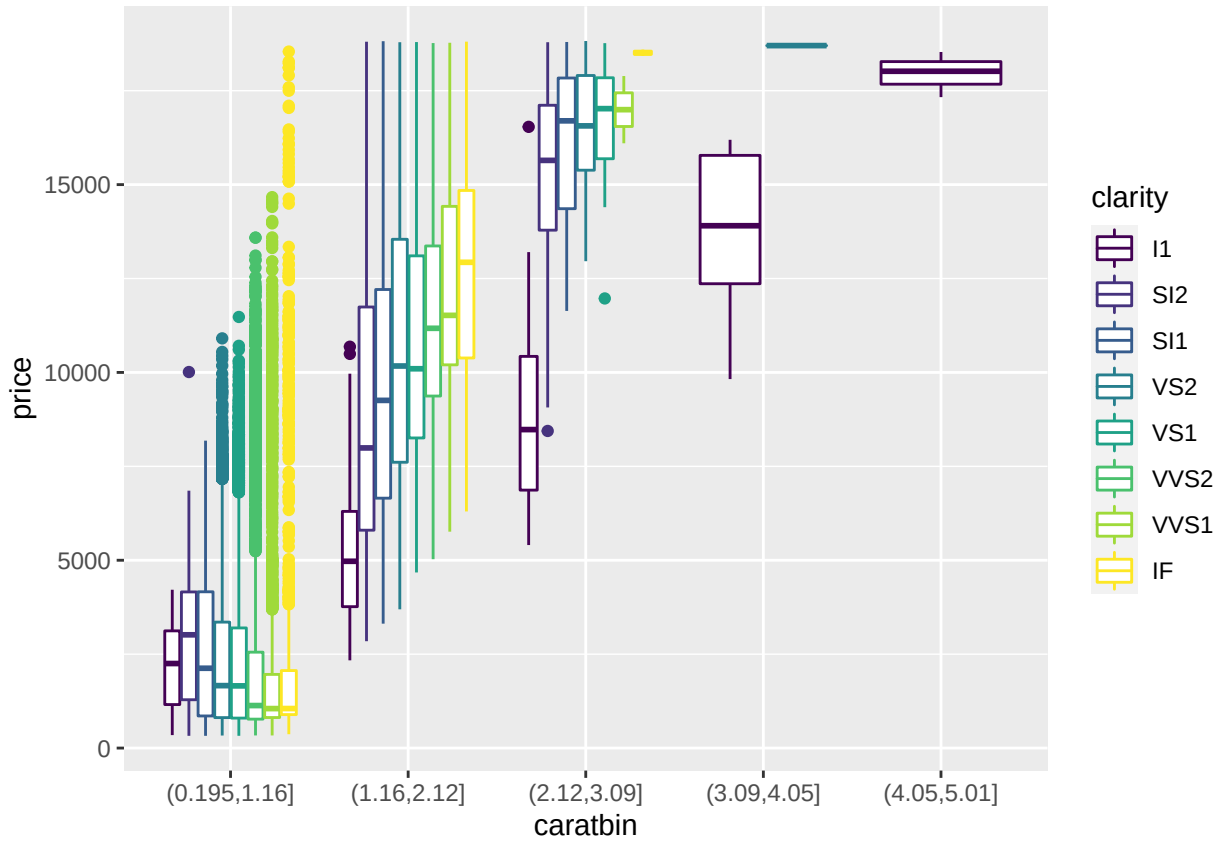
So, with the `cut` function, R cuts all the carat variables into five distinct groups. We can change the `breaks` argument to make more or less groups, but let's stick with five.

```
diamonds %>%
  mutate(caratbin = cut(carat, 5)) %>%
  ggplot(aes(caratbin, price)) +
  geom_boxplot()
```



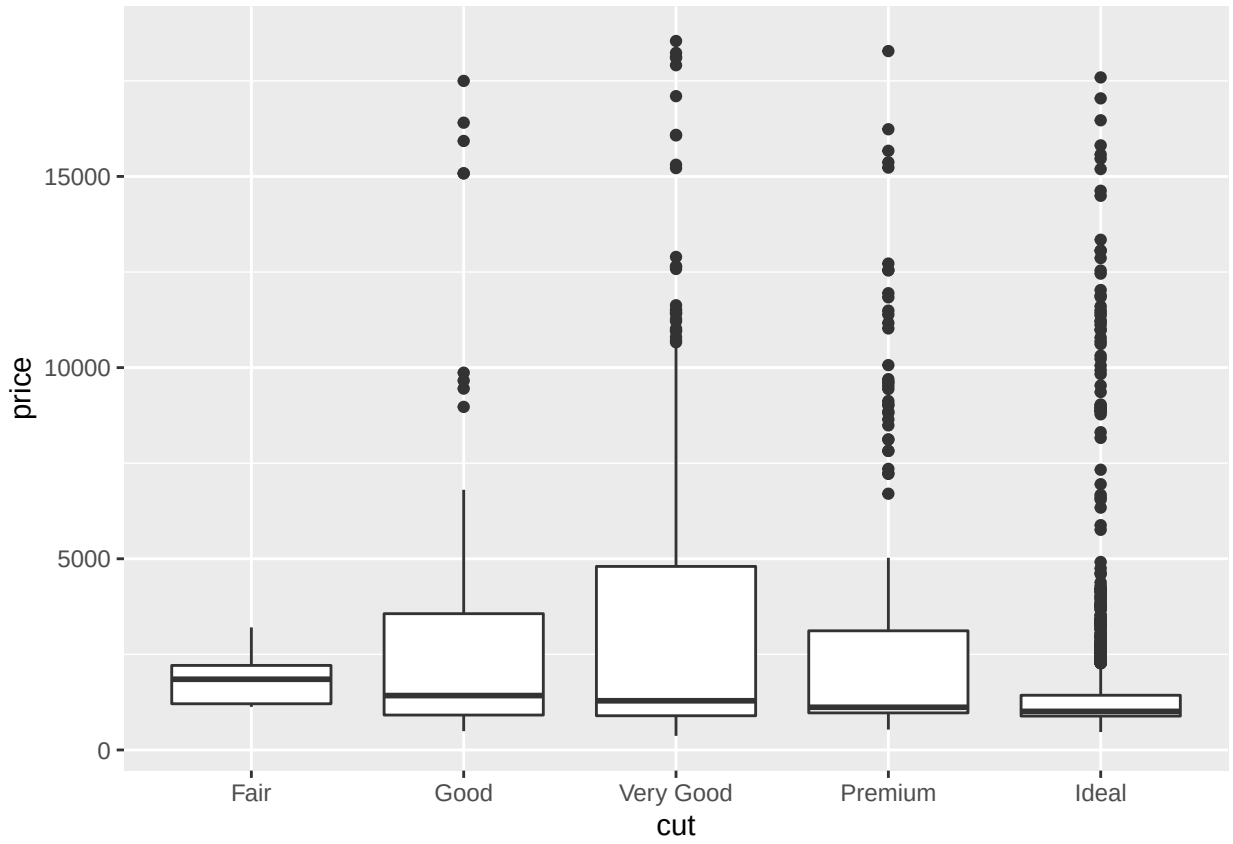
This looks better! We can see how the price grows in every group. We can also see the outliers, and how the groups get smaller. Maybe we can split this up again? Let's say, we could use clarity to split up the caratbins into even smaller groups?

```
diamonds %>%
  mutate(caratbin = cut(carat, 5)) %>%
  ggplot(aes(caratbin, price, color = clarity)) +
  geom_boxplot()
```



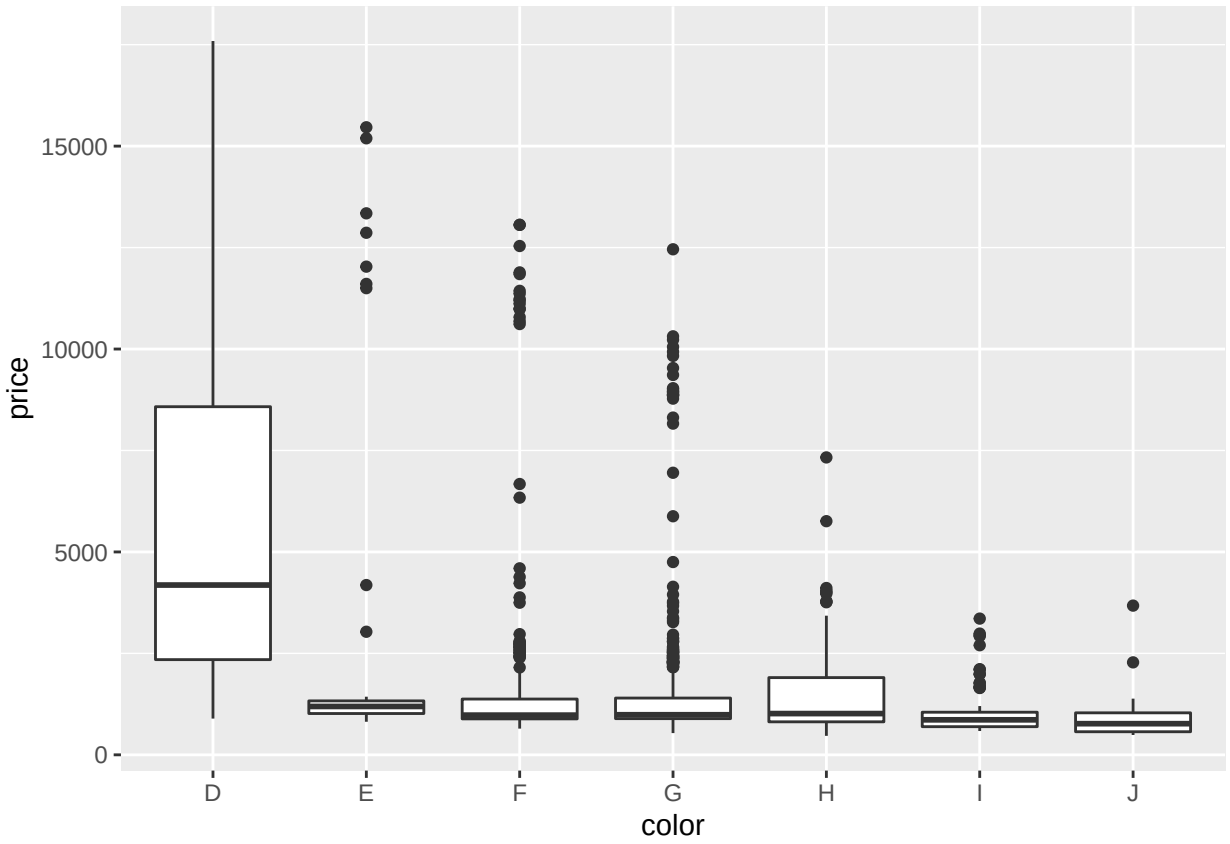
This gives a really nice overview. We can still see how the groups differ in size. We can also see very clearly how the clarity for the small diamonds has an impact on price, but isn't distributed normally. For the rest of the groups, there is more or less a normal distribution.

```
diamonds %>%
  filter(carat < 1.16 & clarity == "IF") %>%
  ggplot(aes(cut, price))+
  geom_boxplot()
```



So the anomaly is mainly in the group with the `Ideal` cut. Let's add that too and try to figure out if we can pin-point some sort of cause. What have we left, maybe color?

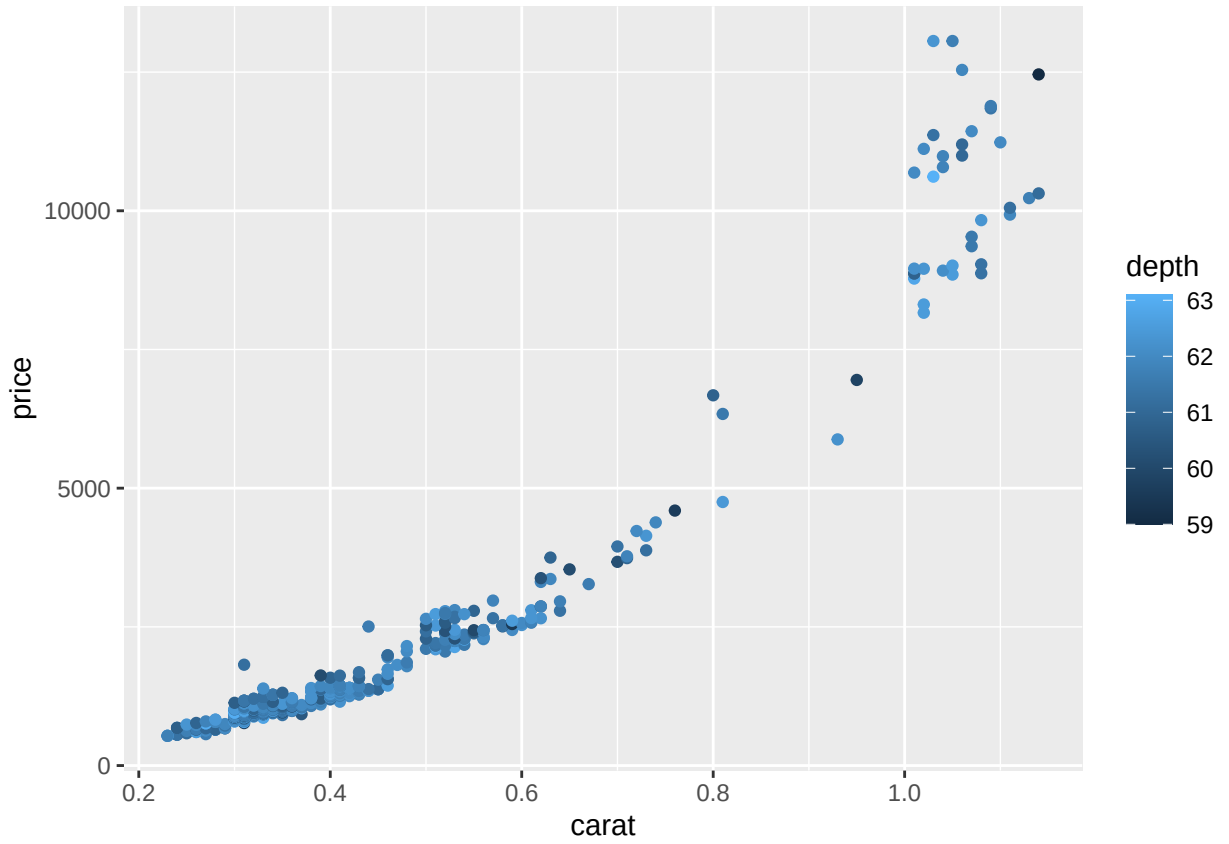
```
diamonds %>%
  filter(carat < 1.16 & clarity == "IF" & cut == "Ideal") %>%
  ggplot(aes(color, price))+
  geom_boxplot()
```



Hmm, so the biggest part of the outliers is in the F and G groups. What's going on there?

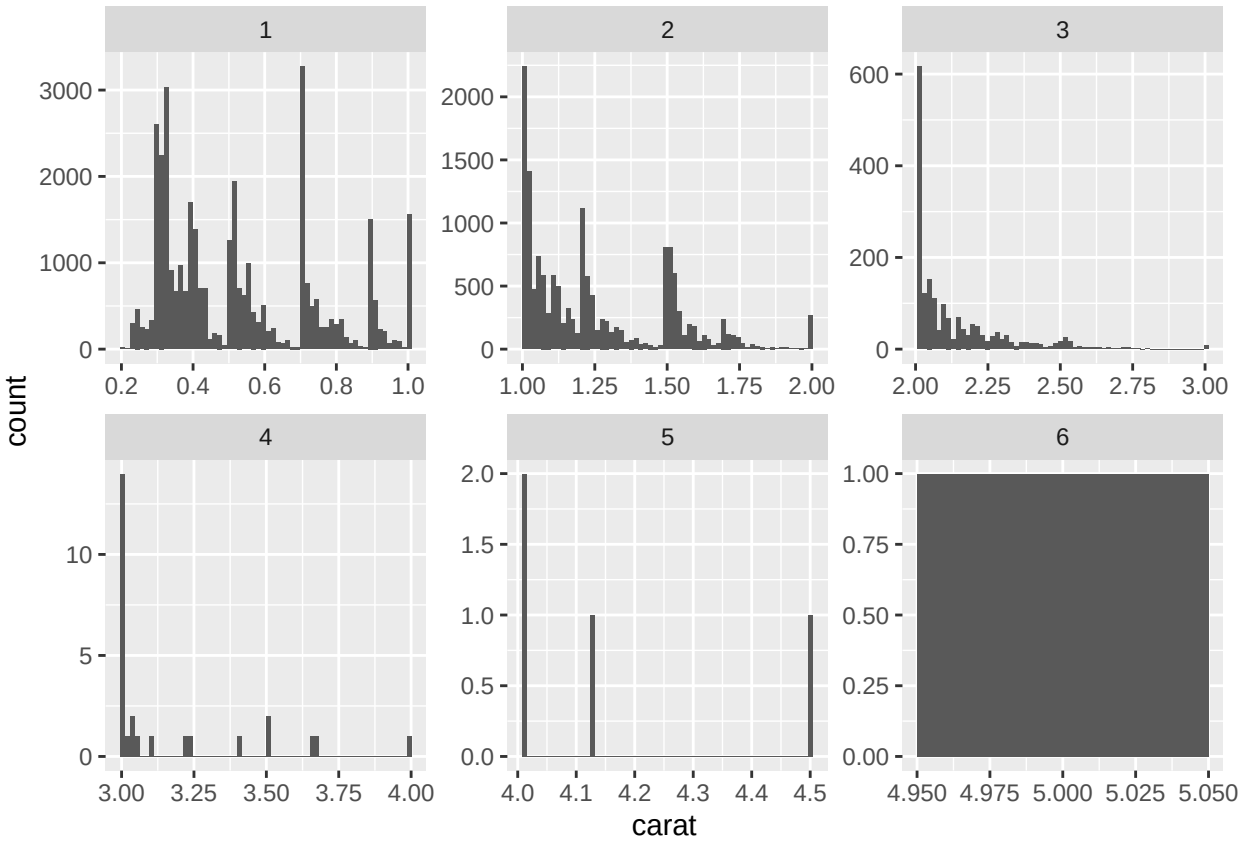
```
subset <- diamonds %>%
  # note the necessary extra parentheses around the two color groups
  filter(carat < 1.16 & clarity == "IF" & cut == "Ideal" & (color == "F"| color == "G")) %>%
  # lets remove all the columns that don't distinguish anymore
  select(-cut, -clarity, -color)


subset %>%
  ggplot(aes(carat, price, color = depth)) +
  geom_point()
```



Ah, there we have it. The group between 0.8-1.0 is missing. This gives the boxplot the unbalanced look. If this was a normal distribution, we would have expected diamonds between 0.8 and 1.0. This is a pattern we actually could have noticed already in the first plots. For some reason, carats tend to converge be distributed more around the start of a carat, and tend to be very rare close to the end of a carat. My hypothesis is, that this is caused by a human decision. We can see how the carat converges to certain numbers by checking the histogram.

```
diamonds %>%
  mutate(caratceiling = ceiling(carat)) %>%
  ggplot(aes(carat))+
  geom_histogram(bins = 60) +
  facet_wrap(~caratceiling, scales = "free")
```


1
2
3
4
5
6
count
carat
0
1000
2000
3000
500
1500
200
400
600
5
10
0.0
0.5
1.0
1.5
2.0
0.25
0.50
0.75
1.00
0.2
0.4
0.6
0.8
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00
4.1
4.2
4.3
4.4
4.5
4.950
4.975
5.000
5.025
5.050