

Lecture 4: Generating 2D graphics with R

October 12, 2016

Contents

- Intro to ggplot2 package
- Comparison with base-graphics
- Geometric objects and aesthetics
- Statistical transformations
- Scales
- Faceting

Installation

```
install.packages("ggplot2")
install.packages("ggrepel")
install.packages("gridExtra")
library(ggplot2)
library(ggrepel)
library(gridExtra)
```

Intro to ggplot2 package

What is ggplot2?

“ggplot2 is a plotting system for R, based on the grammar of graphics. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics”[1].

[1] <http://ggplot2.org/>

Advantages of ggplot2

- Graphics are defined at a **high level of abstraction**.
- Plots are broken down into **layers**.
- The package is **flexible** and offers extensive **customization options**.
- The **documentation** is well-written.
- ggplot2 has a large user base => **it's easy find to help**.

Weaknesses of ggplot2

You should not or cannot use ggplot2 for:

- 3D graphics
 - use `rgl` or `ggplot2 + plotly` instead,
- graph/network plots with nodes and edges
 - use `igraph` instead
- interactive graphics:
 - use `ggvis`, `plotly` instead

What is a grammar of graphics?

- It is a concept **coined by Leland Wilkinson in 2005**.
- An **abstraction** which facilitates reasoning and communicating graphics.
- **ggplot2 is a layered grammar of graphics** which allow users to *independently specify the building blocks of a plot and combine them to create just a any kind of graphical display*

Building blocks of a ggplot2 graphs

- data
- aesthetic mapping
- geometric objects
- statistical transformations
- scales
- coordinate system
- positioning adjustments
- faceting

`ggplot()` function

- `ggplot()` is used to **initialize the basic graph structure**.
- It **cannot produce a plot alone by itself**.
- Instead, we need to **add extra components** to generate a graph.
- You need to **specify different parts** of a plot, and *add them all together* using an `+` operator.

The structure of ggplot object

```
ggplot(data = <default data set>,
aes(x = <default x axis variable>,
y = <default y axis variable>,
... <other default aesthetic mappings>),
... <other plot defaults>) +  
  
geom_<geom type>(aes(size = <size variable for this geom>,
... <other aesthetic mappings>),
data = <data for this point geom>,
stat = <statistic string or function>,
position = <position string or function>,
color = <"fixed color specification">,
... <other arguments, possibly passed to the _stat_ function>) +  
  
scale_<aesthetic>_<type>(name = <"scale label">,
breaks = <where to put tick marks>,
labels = <labels for tick marks>,
... <other options for the scale>) +  
  
theme(plot.background = element_rect(fill = "gray"),
... <other theme elements>)
```

Comparison with base-graphics

ggplot2 compared to base graphics is

- more verbose for **simple / out of the box** graphics
- less verbose for **complex / custom** graphics
- uses a different system for **adding plot elements** instead of calling new functions.
- more details on why to use ggplot2 over base plot can be found in this [blog](#).

Example 1: History of unemployment

ggplot2 has a built-in **economics** dataset, which includes time series data on unemployment from 1967 to 2015.

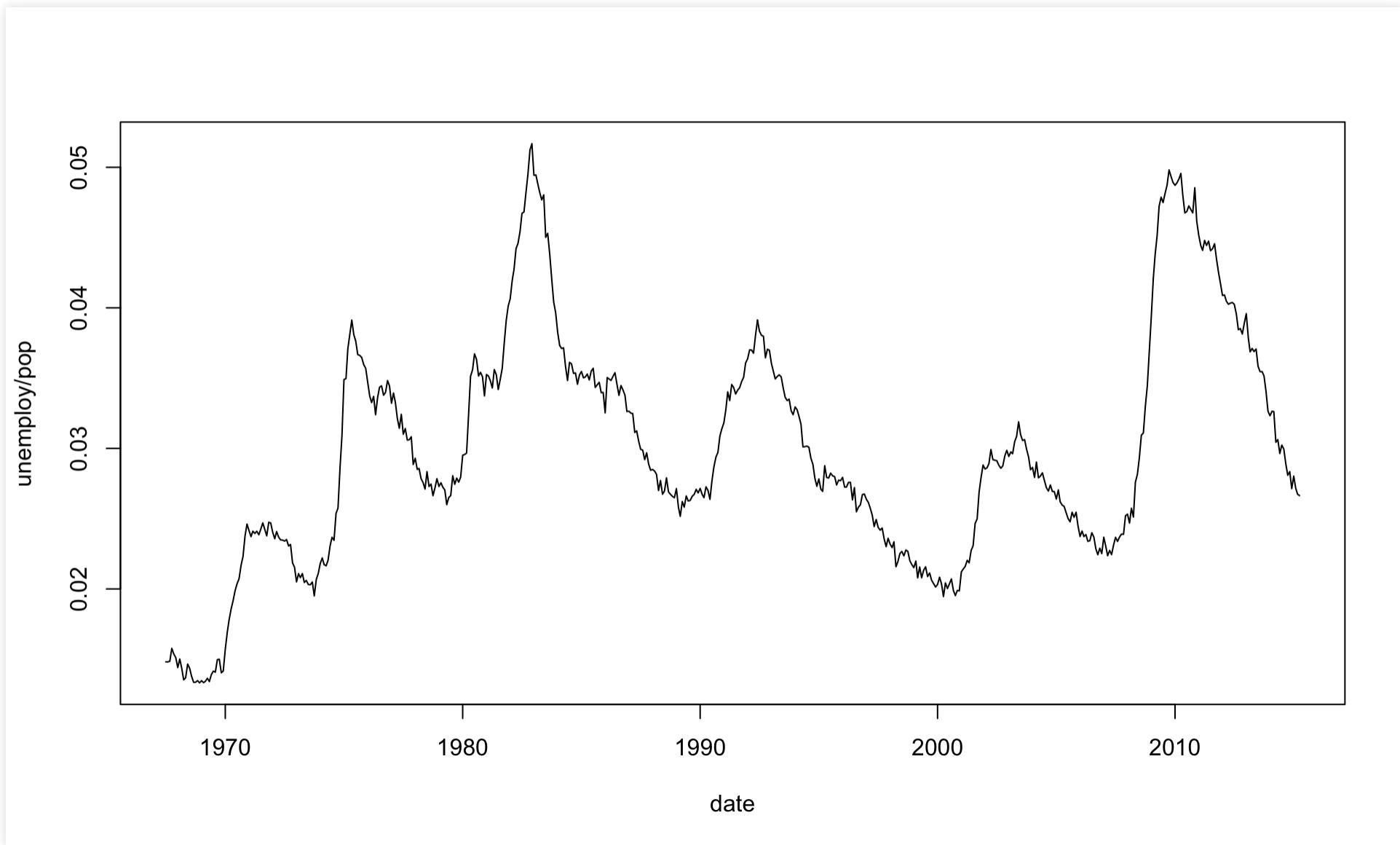
```
economics
```

```
## # A tibble: 574 x 6
##   date      pce    pop psavert uempmed unemploy
##   <date>    <dbl>  <int>    <dbl>    <dbl>    <int>
## 1 1967-07-01 507.4 198712 12.5     4.5    2944
## 2 1967-08-01 510.5 198911 12.5     4.7    2945
## 3 1967-09-01 516.3 199113 11.7     4.6    2958
## 4 1967-10-01 512.9 199311 12.5     4.9    3143
## 5 1967-11-01 518.1 199498 12.5     4.7    3066
## 6 1967-12-01 525.8 199657 12.1     4.8    3018
## 7 1968-01-01 531.5 199808 11.7     5.1    2878
## 8 1968-02-01 534.2 199920 12.2     4.5    3001
## 9 1968-03-01 544.9 200056 11.6     4.1    2877
## 10 1968-04-01 544.6 200208 12.2     4.6    2709
## # ... with 564 more rows
```

```
economics <- mutate(economics, unemp_rate = unemploy/pop)
```

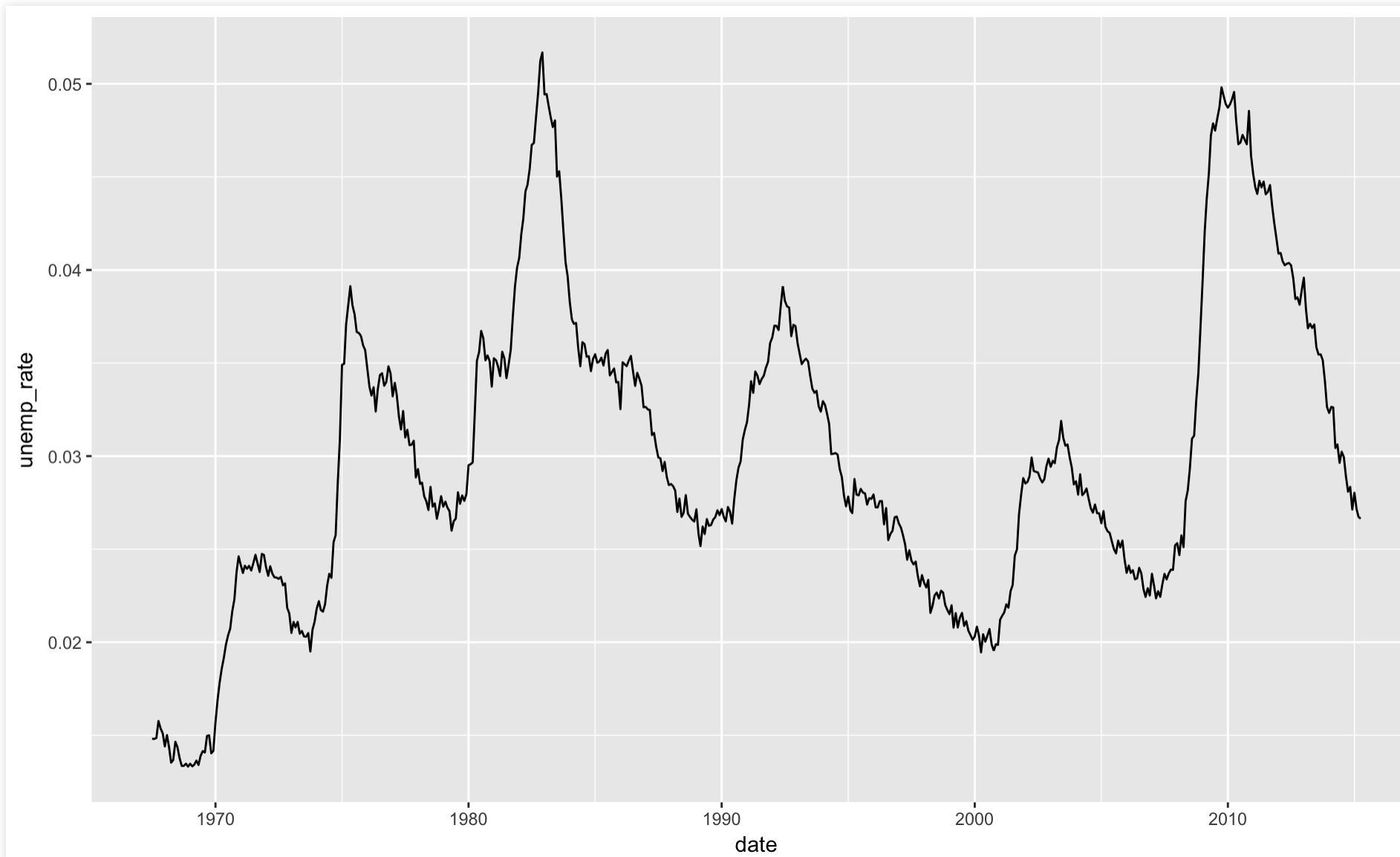
R base graphics

```
plot(unemploy/pop ~ date, data = economics, type = "l")
```



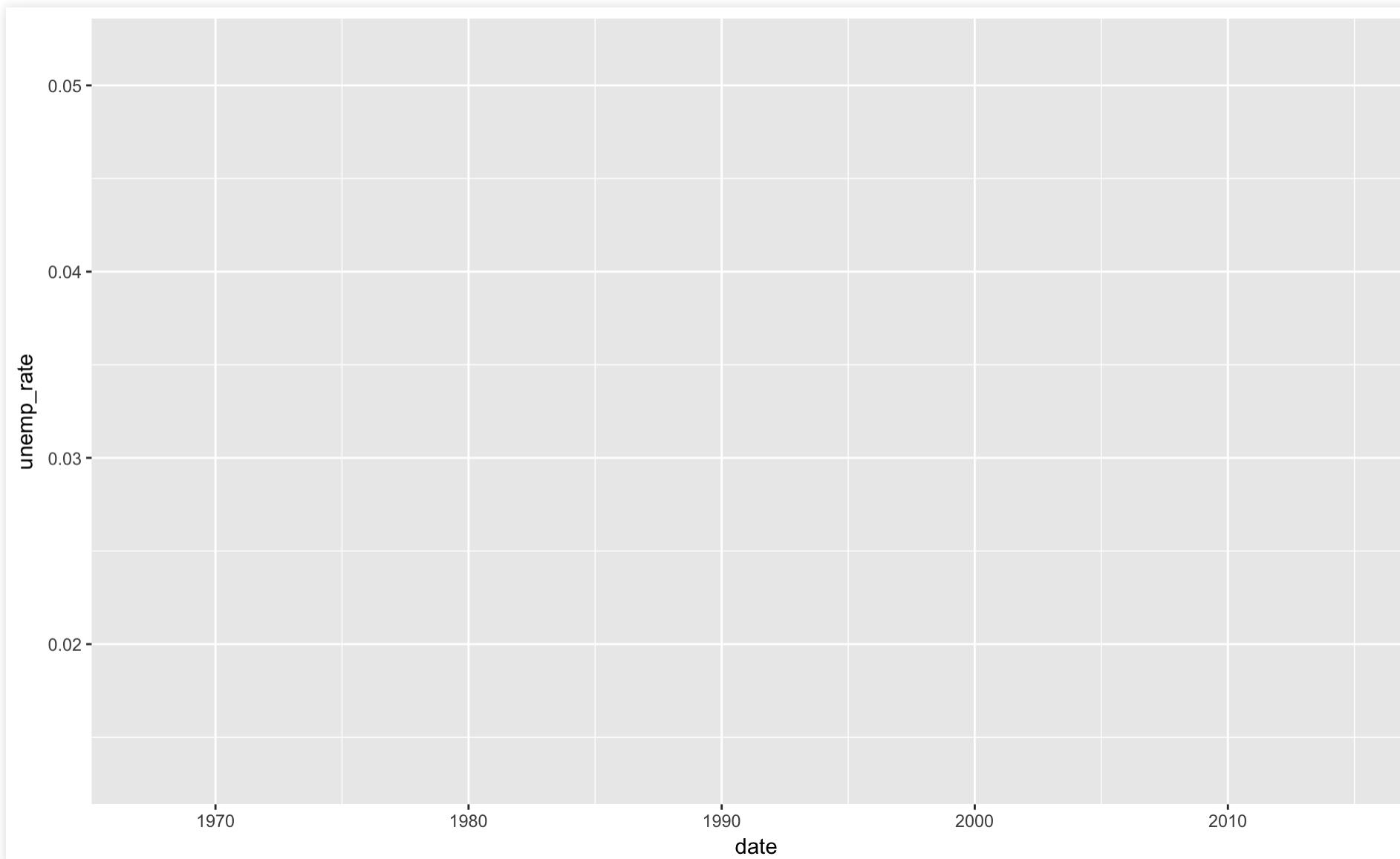
ggplot2 package

```
library(ggplot2)  
ggplot(data = economics, aes(x = date, y = unemp_rate)) + geom_line()
```



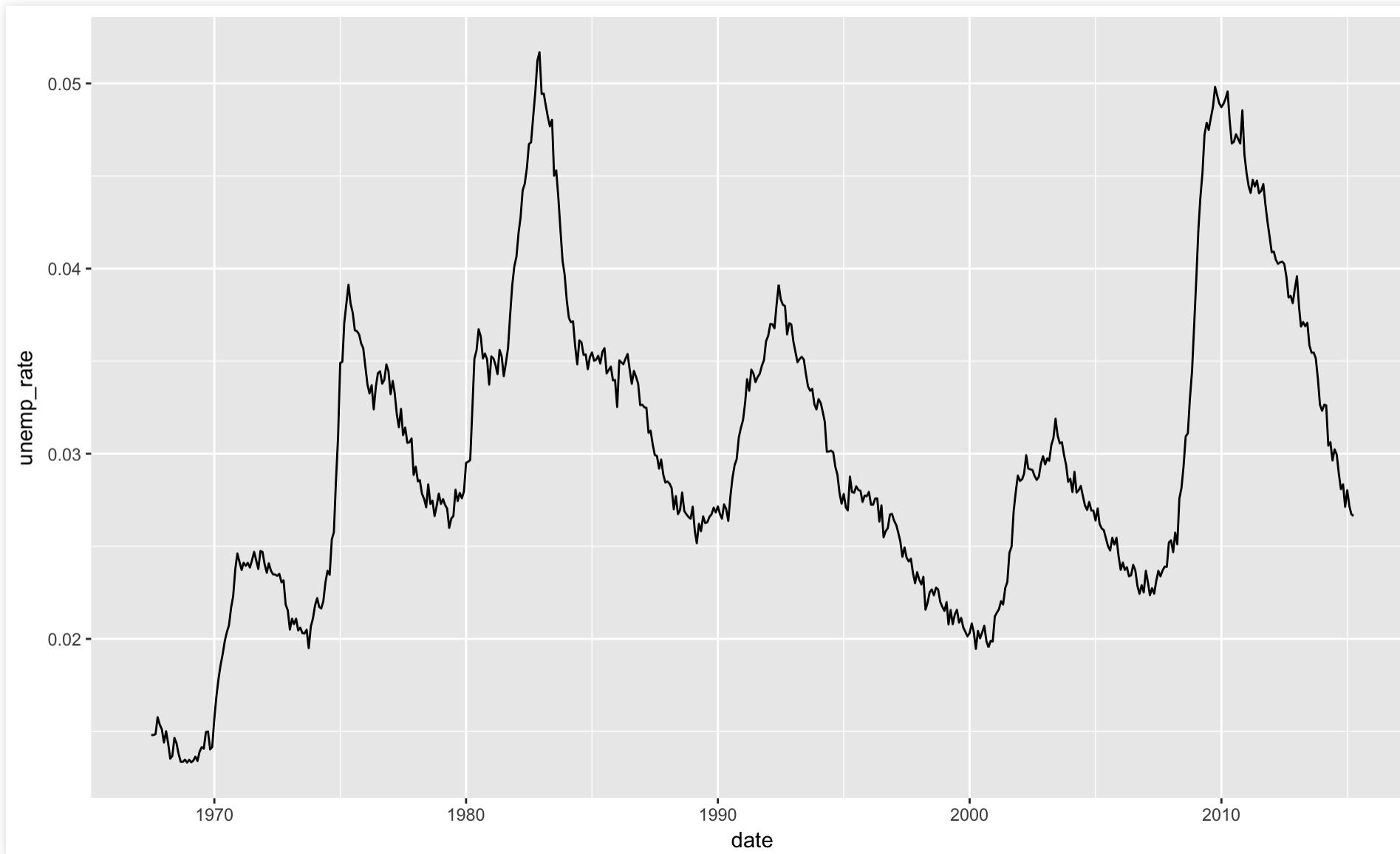
ggplot() by itself does not plot the data

```
ggplot(data = economics, aes(x = date, y = unemp_rate))
```



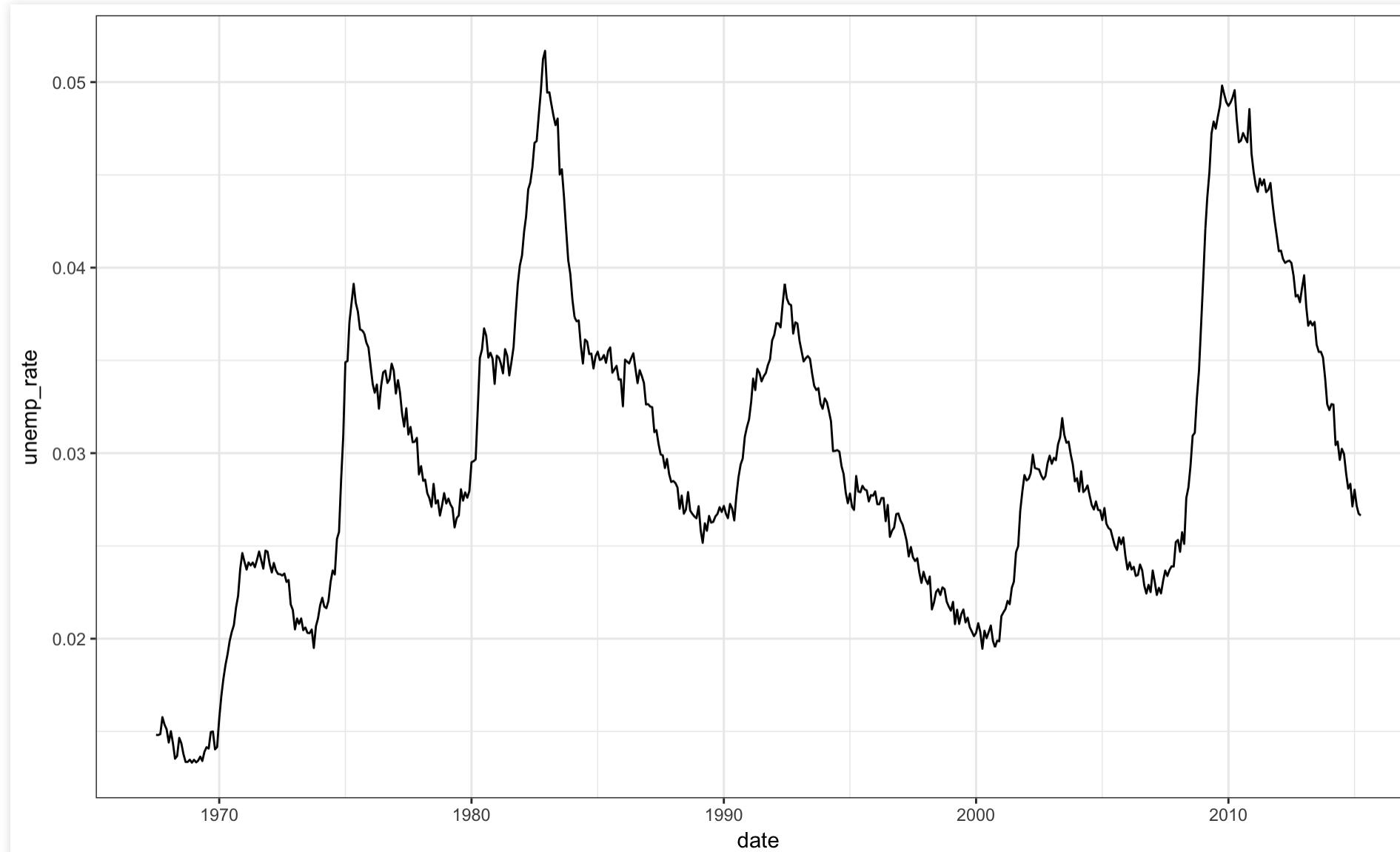
You need to add a line-layer

```
ggplot(data = economics, aes(x = date, y = unemp_rate)) + geom_line()
```



Change the background color to white

```
ggplot(data = economics, aes(x = date, y = unemp_rate)) +  
  geom_line() + theme_bw()
```



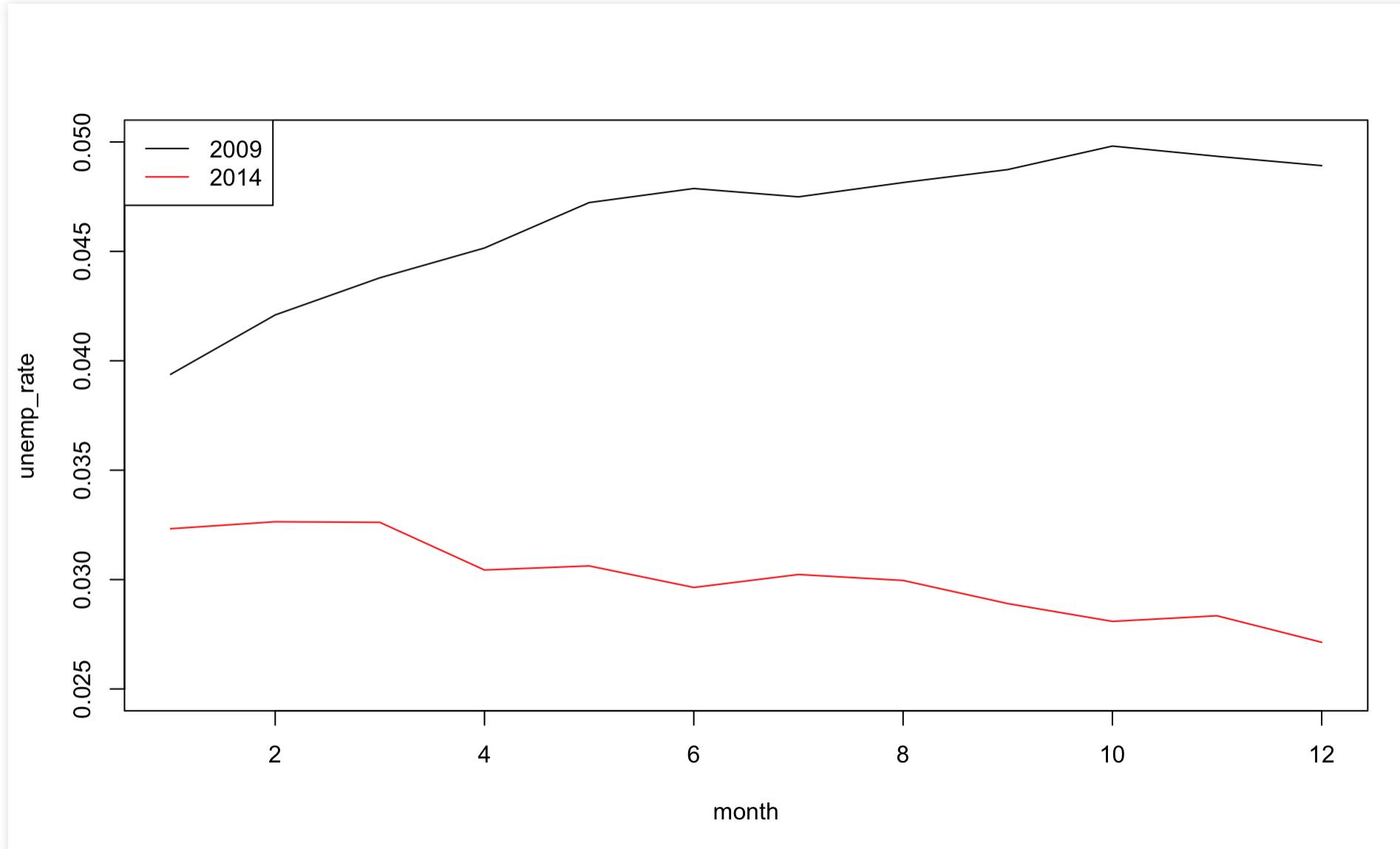
What about comparing 2009 to 2014?

```
# Add new variables for plotting
economics <- economics %>%
  mutate(month = as.numeric(format(date, format="%m")),
        year = as.numeric(format(date, format="%Y"))),
  economics %>% select(date, month, year, unemp_rate)
```

```
## # A tibble: 574 x 4
##   date     month year unemp_rate
##   <date>    <dbl> <dbl>      <dbl>
## 1 1967-07-01    7  1967  0.01481541
## 2 1967-08-01    8  1967  0.01480562
## 3 1967-09-01    9  1967  0.01485589
## 4 1967-10-01   10  1967  0.01576933
## 5 1967-11-01   11  1967  0.01536858
## 6 1967-12-01   12  1967  0.01511592
## 7 1968-01-01    1  1968  0.01440383
## 8 1968-02-01    2  1968  0.01501100
## 9 1968-03-01    3  1968  0.01438097
## 10 1968-04-01   4  1968  0.01353093
## # ... with 564 more rows
```

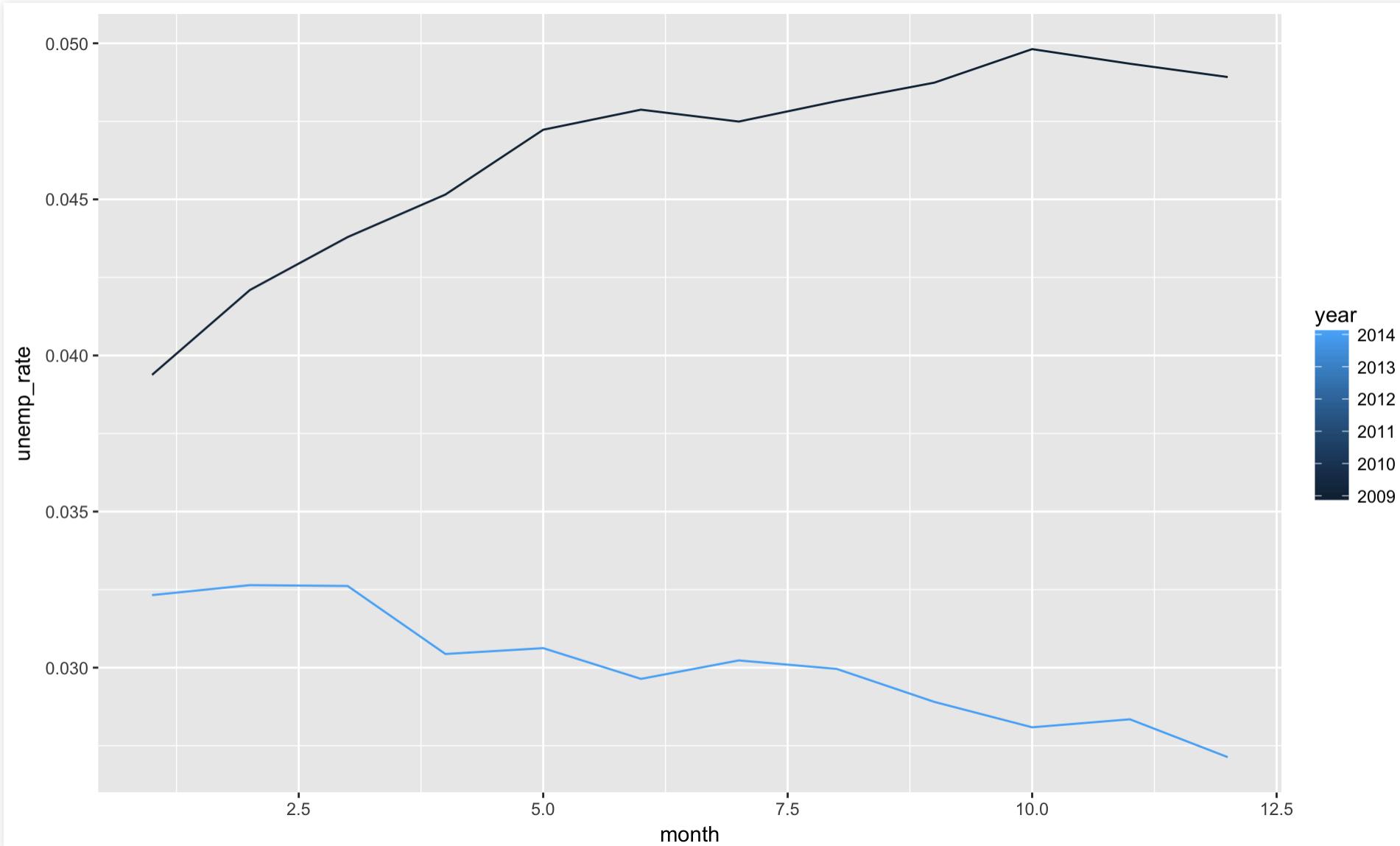
Using base graphics

```
data09 <- subset(economics, year == 2009)
data14 <- subset(economics, year == 2014)
plot(unemp_rate ~ month, data = data09, ylim = c(0.025, 0.05), type = "l")
lines(unemp_rate ~ month, data = data14, col = "red")
legend("topleft", c("2009", "2014"), col = c("black", "red"), lty = c(1,1))
```

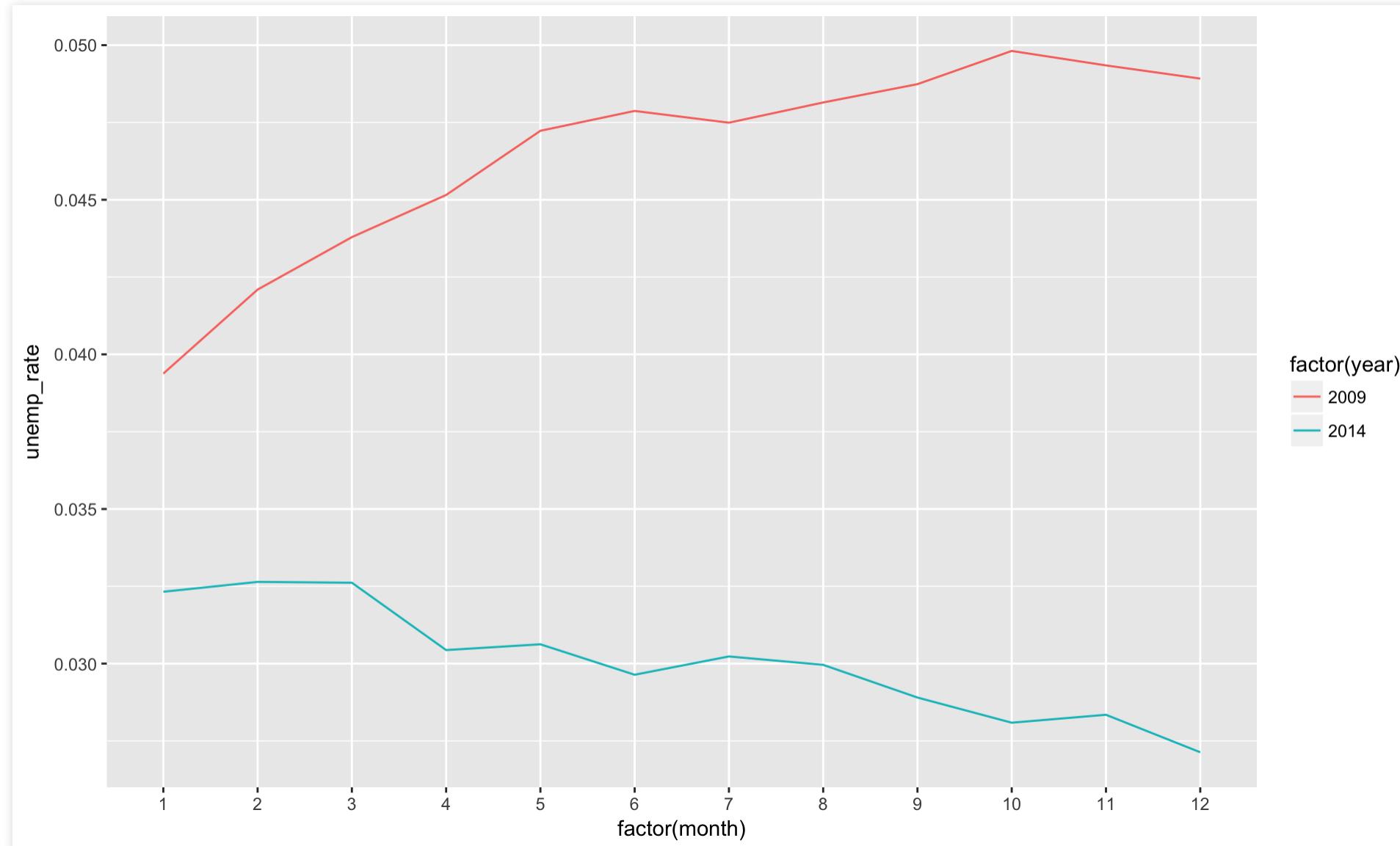


Using ggplot2

```
data2009_2014 <- subset(economics, year %in% c(2014, 2009))  
ggplot(data = data2009_2014, aes(x = month, y = unemp_rate)) +  
  geom_line(aes(group = year, color = year)) # No need to specify a legend
```

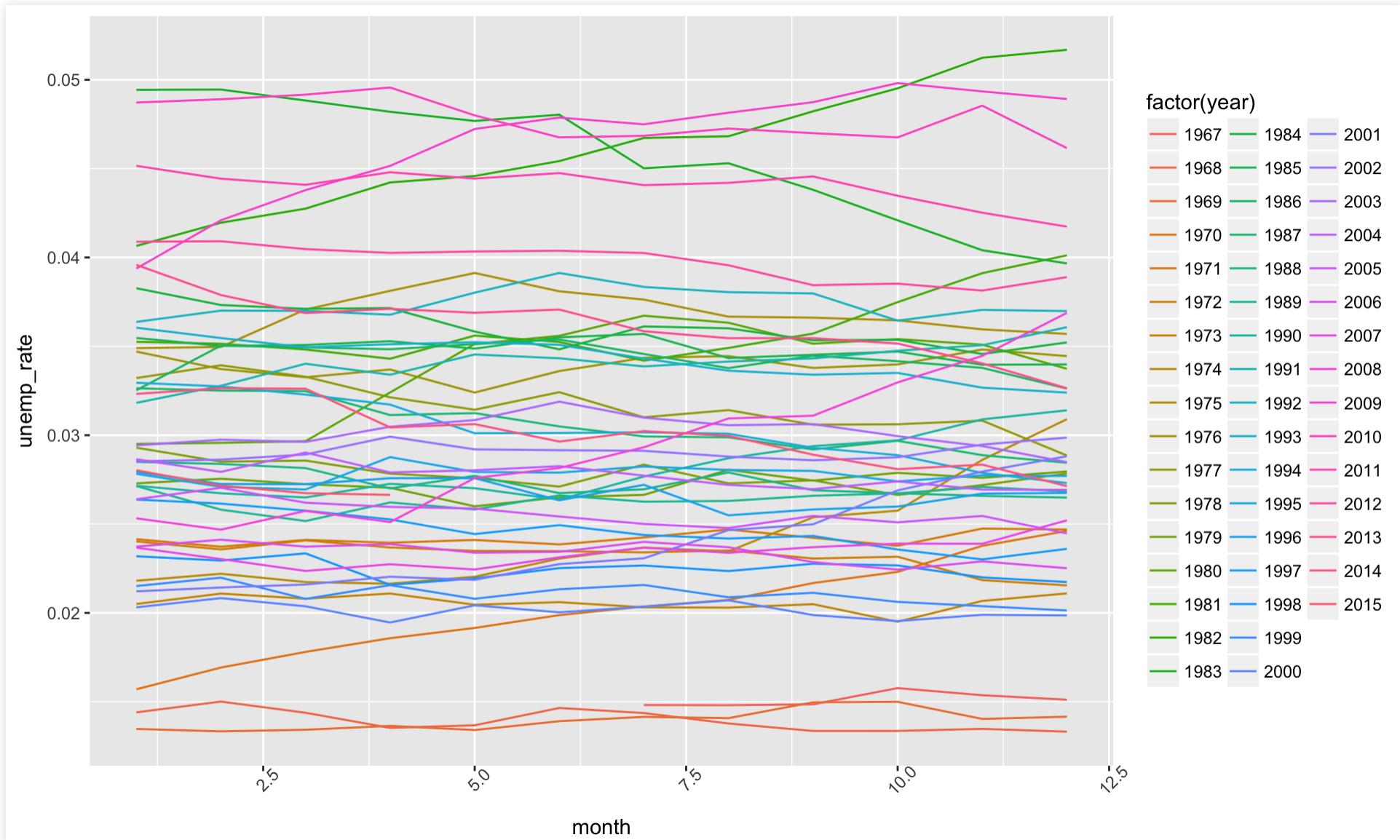


```
ggplot(data = data2009_2014, aes(x = factor(month), y = unemp_rate)) +  
  geom_line(aes(group = year, color = factor(year)))
```



Plotting all the years together is easy

```
ggplot(data = economics, aes(x = month, y = unemp_rate)) +  
  geom_line(aes(group = year, color = factor(year))) +  
  theme(axis.text.x = element_text(angle = 45))
```



Example 2: diamonds dataset

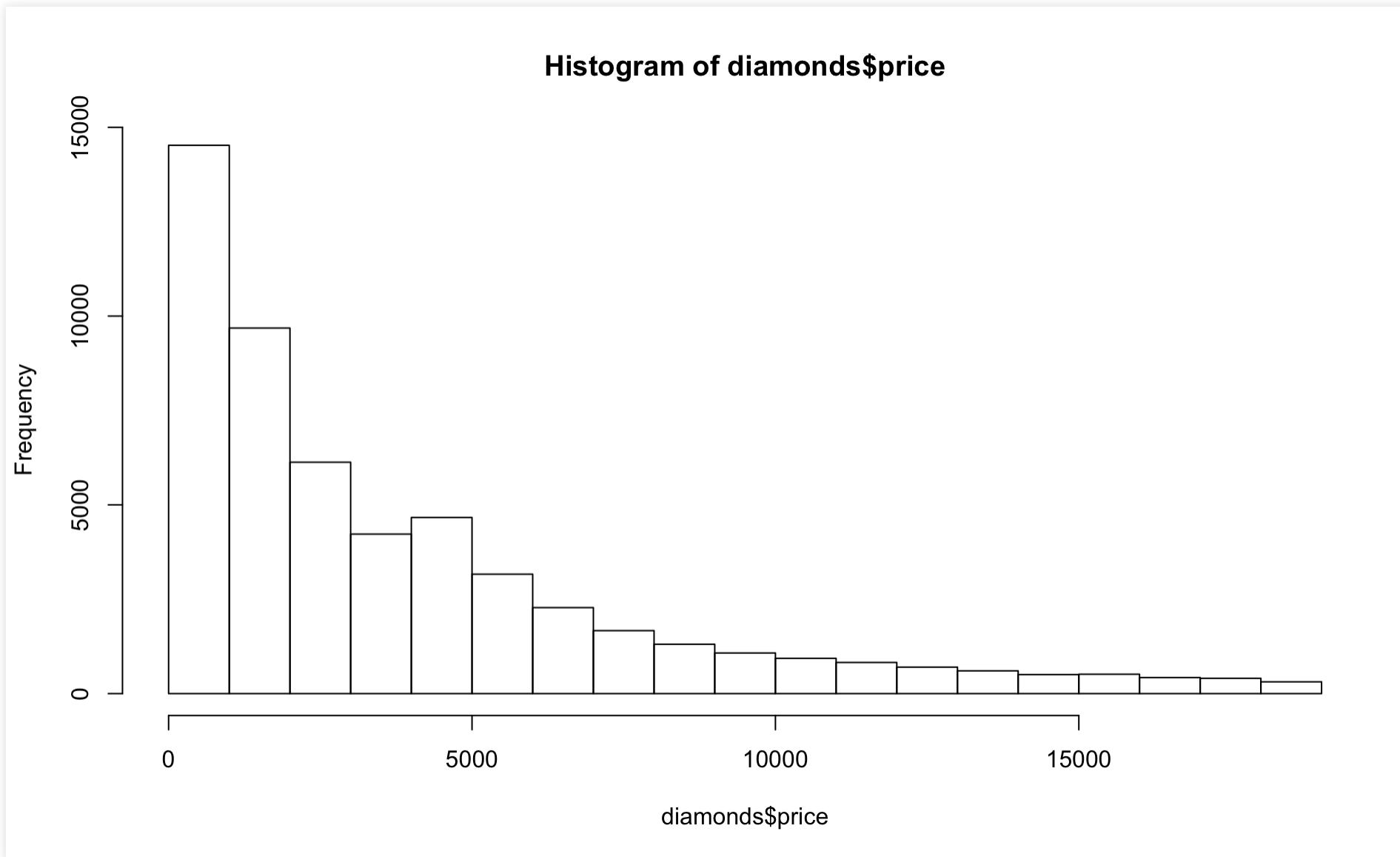
The dataset contains the prices and other attributes of almost 54,000 diamonds.
You can call `?diamonds` to learn more about the available attributes.

```
diamonds
```

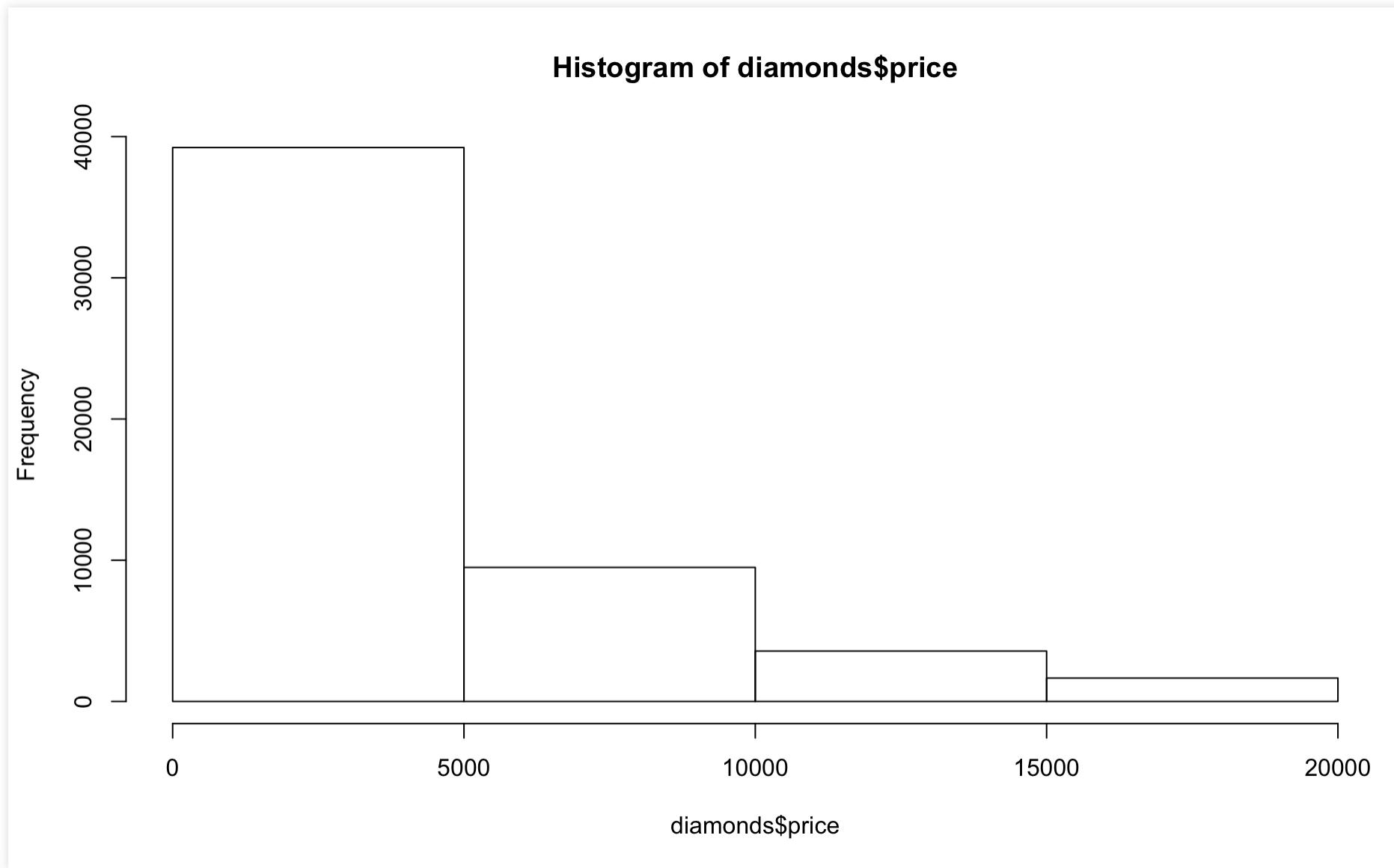
```
## # A tibble: 53,940 x 10
##   carat cut color clarity depth table price x y z
##   <dbl> <ord> <ord> <ord> <dbl> <dbl> <int> <dbl> <dbl>
## 1 0.23 Ideal E SI2 61.5 55 326 3.95 3.98 2.43
## 2 0.21 Premium E SI1 59.8 61 326 3.89 3.84 2.31
## 3 0.23 Good E VS1 56.9 65 327 4.05 4.07 2.31
## 4 0.29 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
```

Distribution of the diamonds prices with base graphics

```
hist(diamonds$price)
```



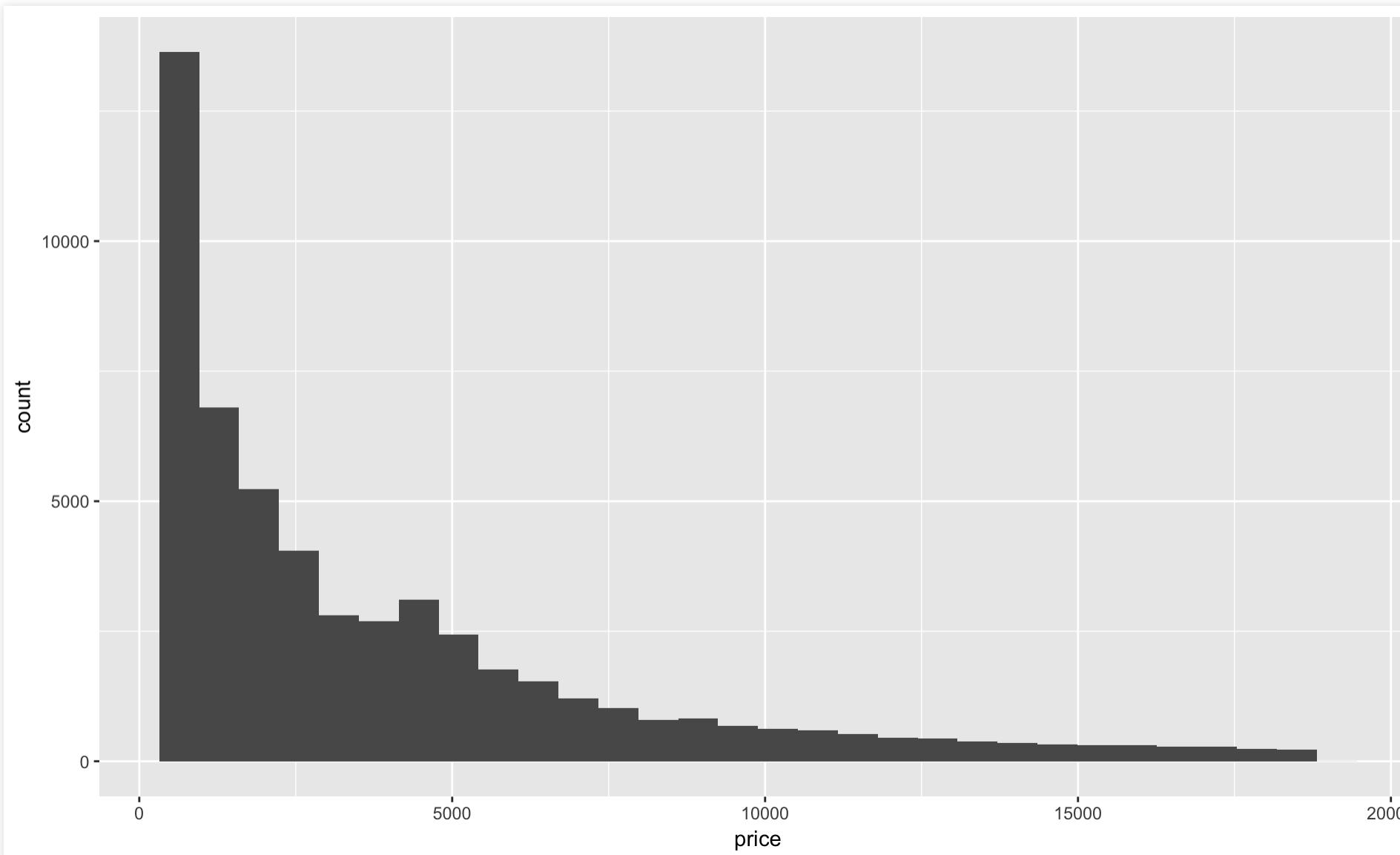
```
# breaks can be a vector, a function or a single number  
hist(diamonds$price, breaks = 5)
```



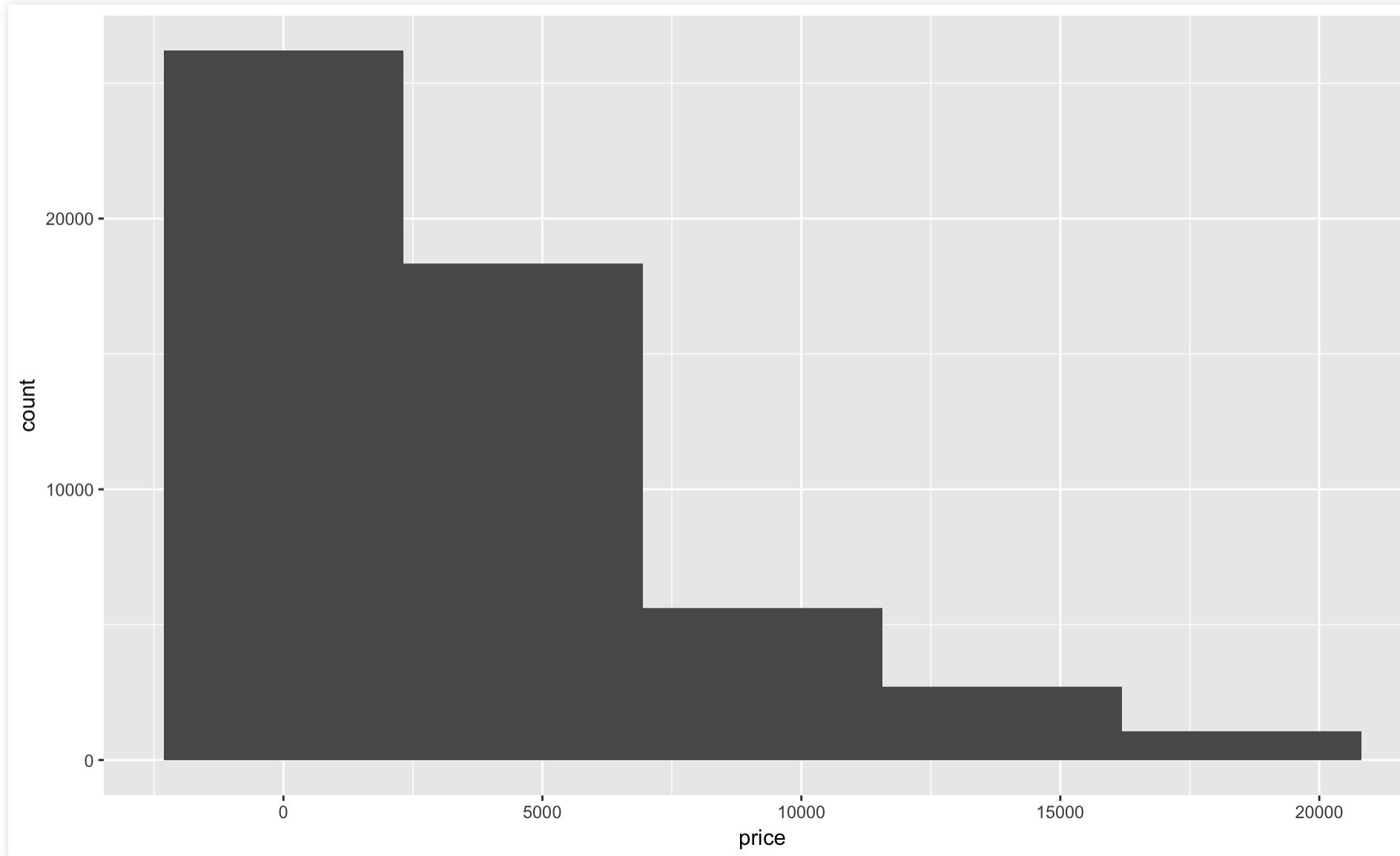
Histograms with ggplot2

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

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



```
# you can set bins= or binwidths =  
ggplot(diamonds, aes(x = price)) + geom_histogram(bins = 5)
```



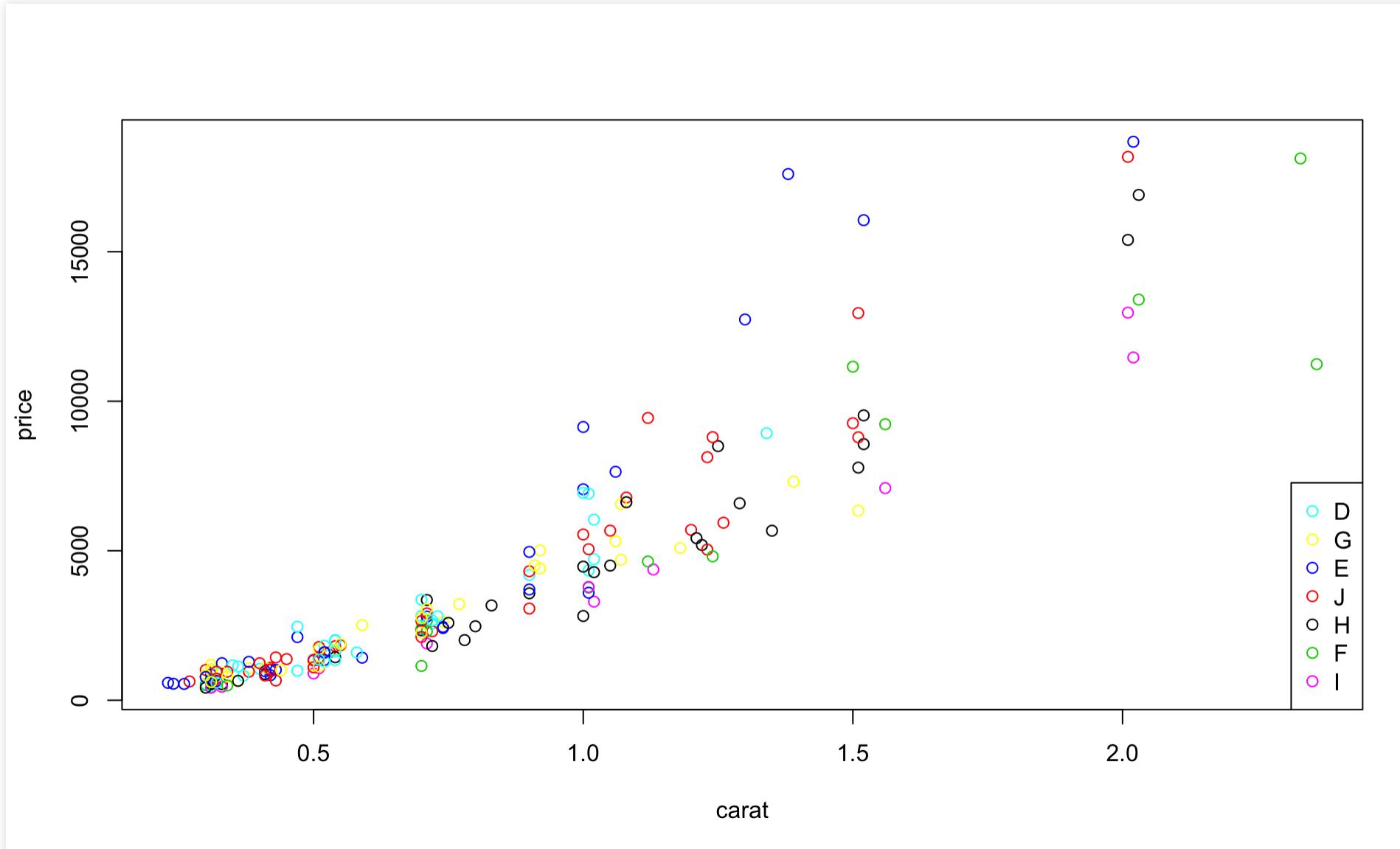
Subset the data

- We select a random subset of the data to use for plotting the relationship between the weights (carat = 200 mg) and prices (\$) of diamonds

```
set.seed(12345) # Make the sample reproducible  
dsmall <- diamonds[sample(nrow(diamonds), 200), ]
```

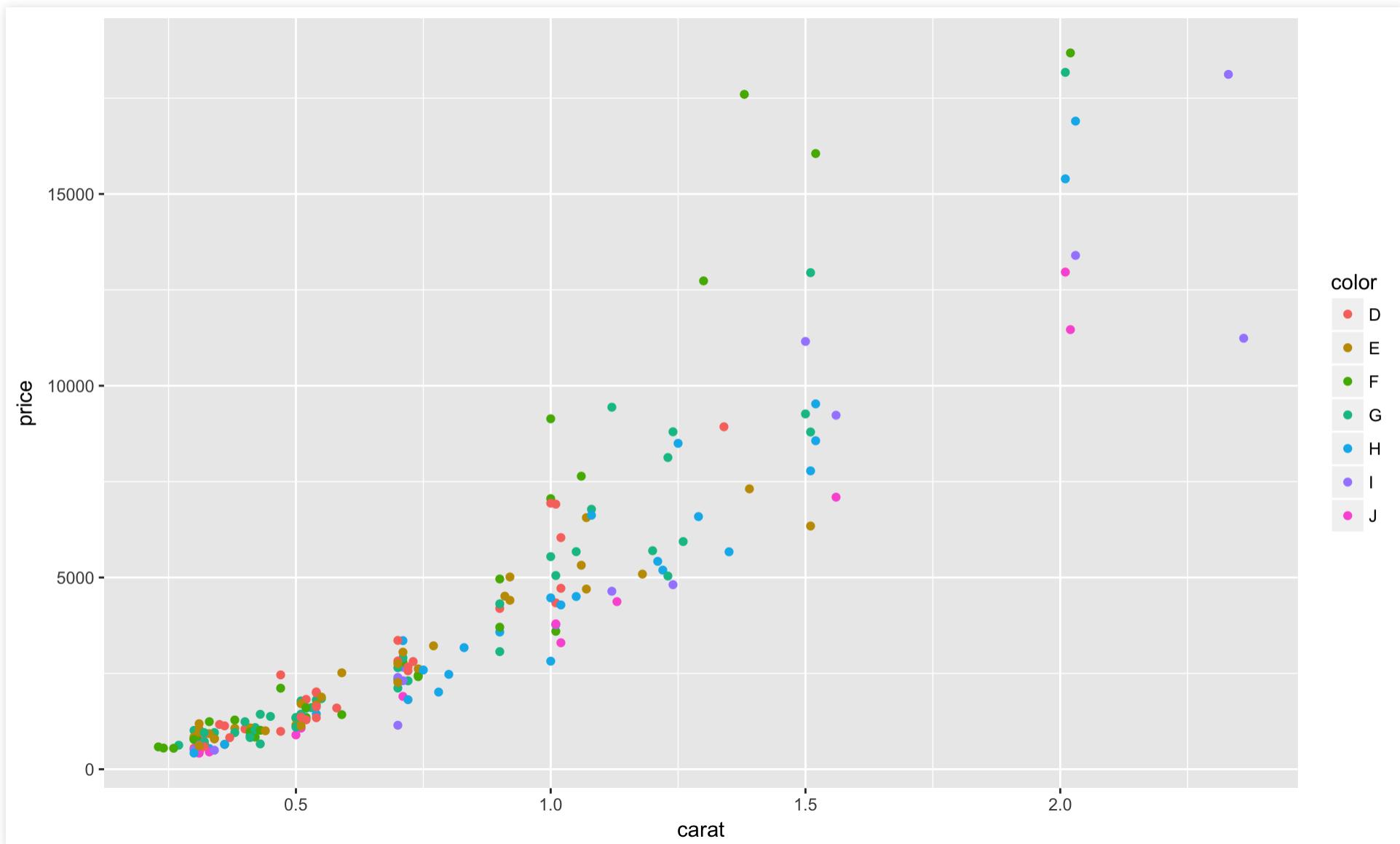
Scatter plot with base graphics

```
colorMap <- data.frame(row.names = unique(dsmall$color),  
color = rainbow(length(unique(dsmall$color))))  
plot(price ~ carat, data = dsmall, col = colorMap[dsmall$color, "color"] )  
legend(x = 'bottomright', legend = rownames(colorMap), col = colorMap$color, pch = 1)
```



Scatter plot with ggplot2

```
ggplot(data = dsmall, aes(x = carat, y = price, color = color)) +  
  geom_point()
```



Geometric objects and aesthetics

Geometric objects - actual items on the plot

- points (`geom_point()`, used for scatter plots)
- lines (`geom_line()`, used for time series, trend lines, etc.)
- boxplot (`geom_boxplot()` used for, well, boxplots!)

There is no upper limit to how many geom objects you can use. You can add a geom object to a plot using an `+` operator.

To get a list of available geometric objects use the following:

```
help.search("geom_", package = "ggplot2")
```

Aesthetic mapping

- In ggplot an **aesthetic mapping**, defined with `aes()`, describes how variables are mapped to visual properties or aesthetics.
- For example, we might map weight to x weight, height to y price, and cut to color for the diamonds dataset.
- The details of the mapping are described by the scales.

Examples of aesthetics are:

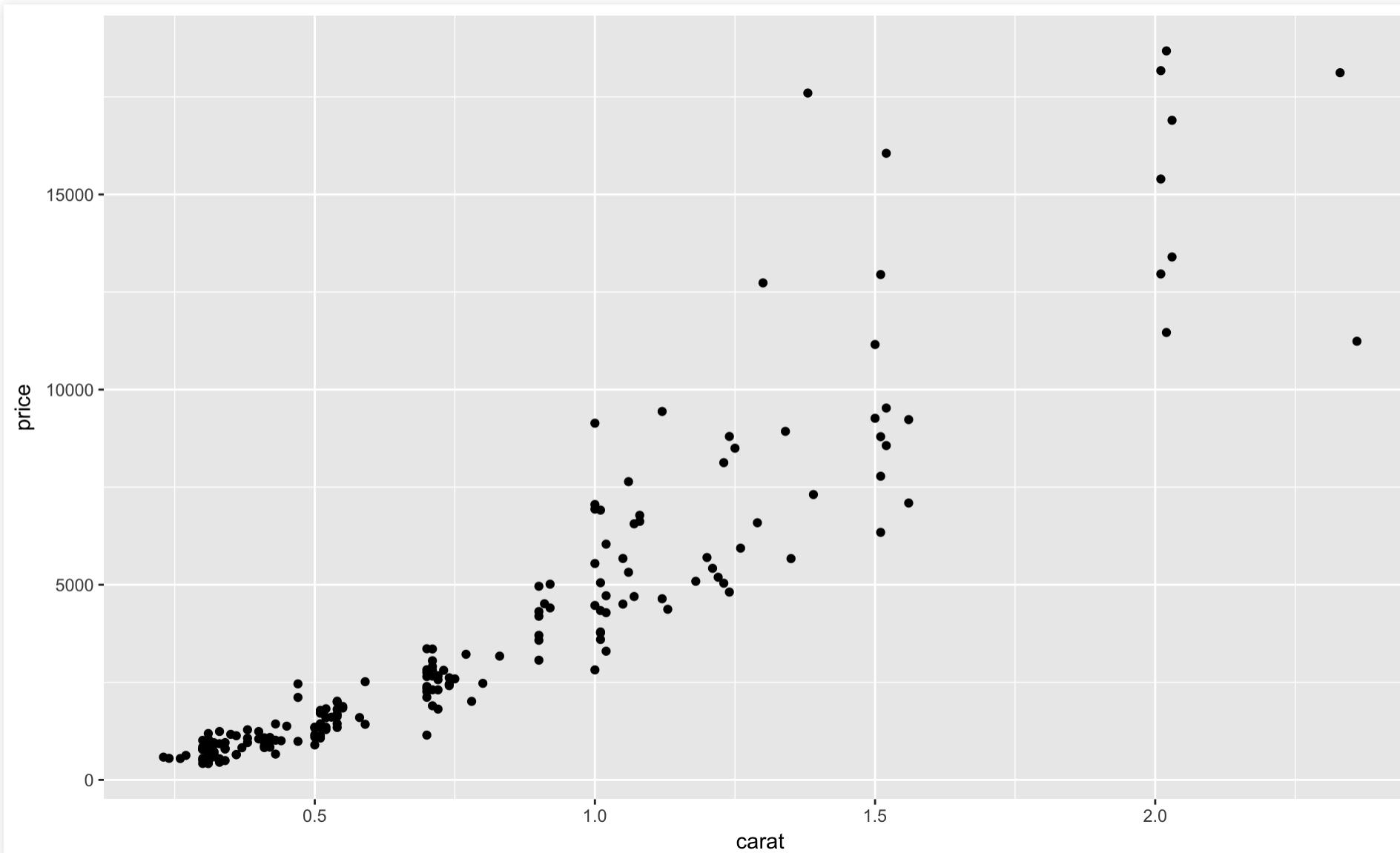
- position (i.e., on the x and y axes)
- shape
- linetype
- size
- color (“outside” color)
- fill (“inside” color)

Each type of geom objects accepts only a subset of aesthetics.

Refer to the geom help pages to see what mappings each geom accepts.

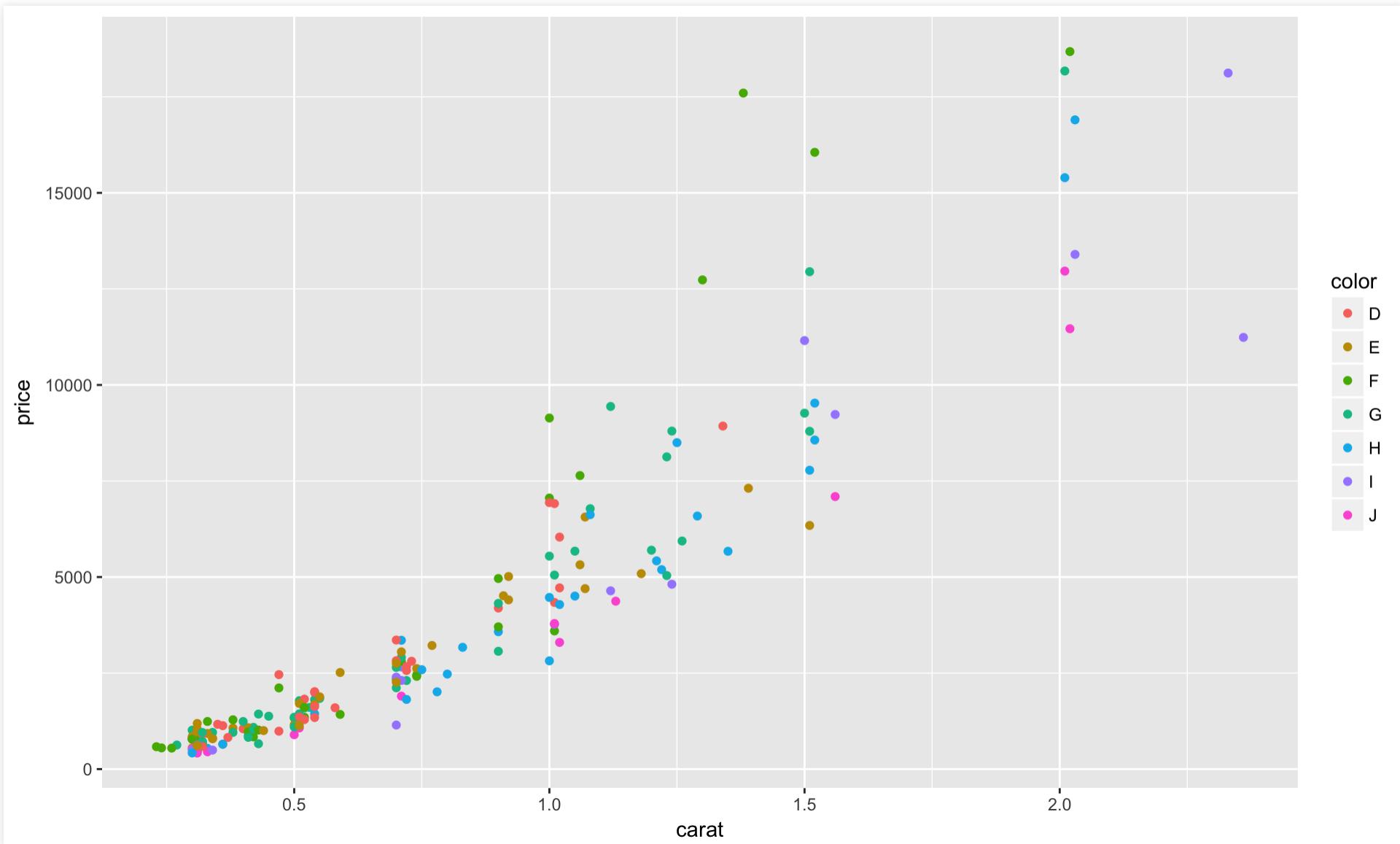
Scatter plots with geom_points

```
p1 <- ggplot(dsmall, aes(x = carat, y = price))  
p1 + geom_point()
```



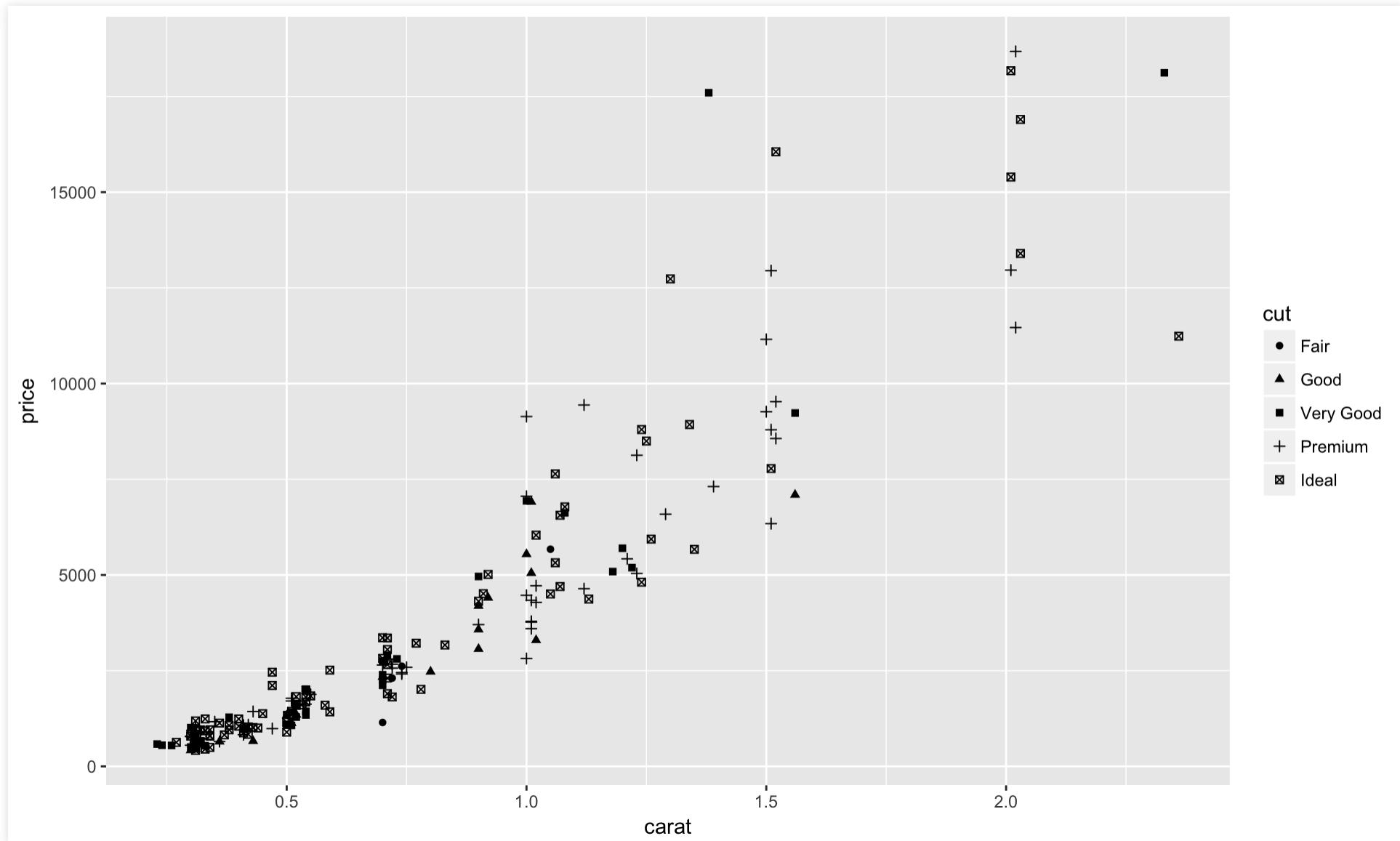
Color points

```
# color by diamonds color  
p1 + geom_point(aes(color = color))
```



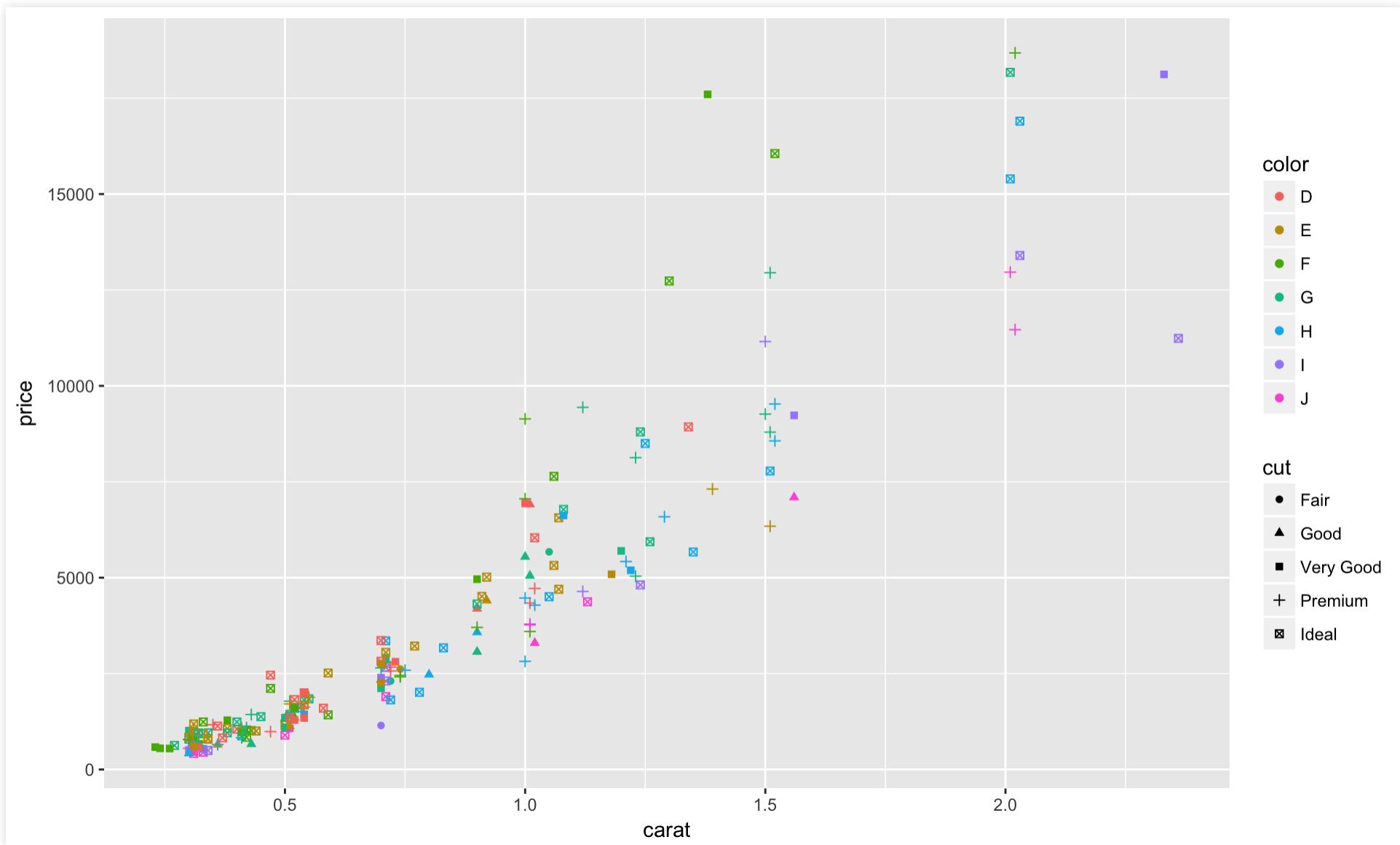
Set the shape of the points

```
# set shape by diamond cut  
p1 + geom_point(aes(shape = cut))
```



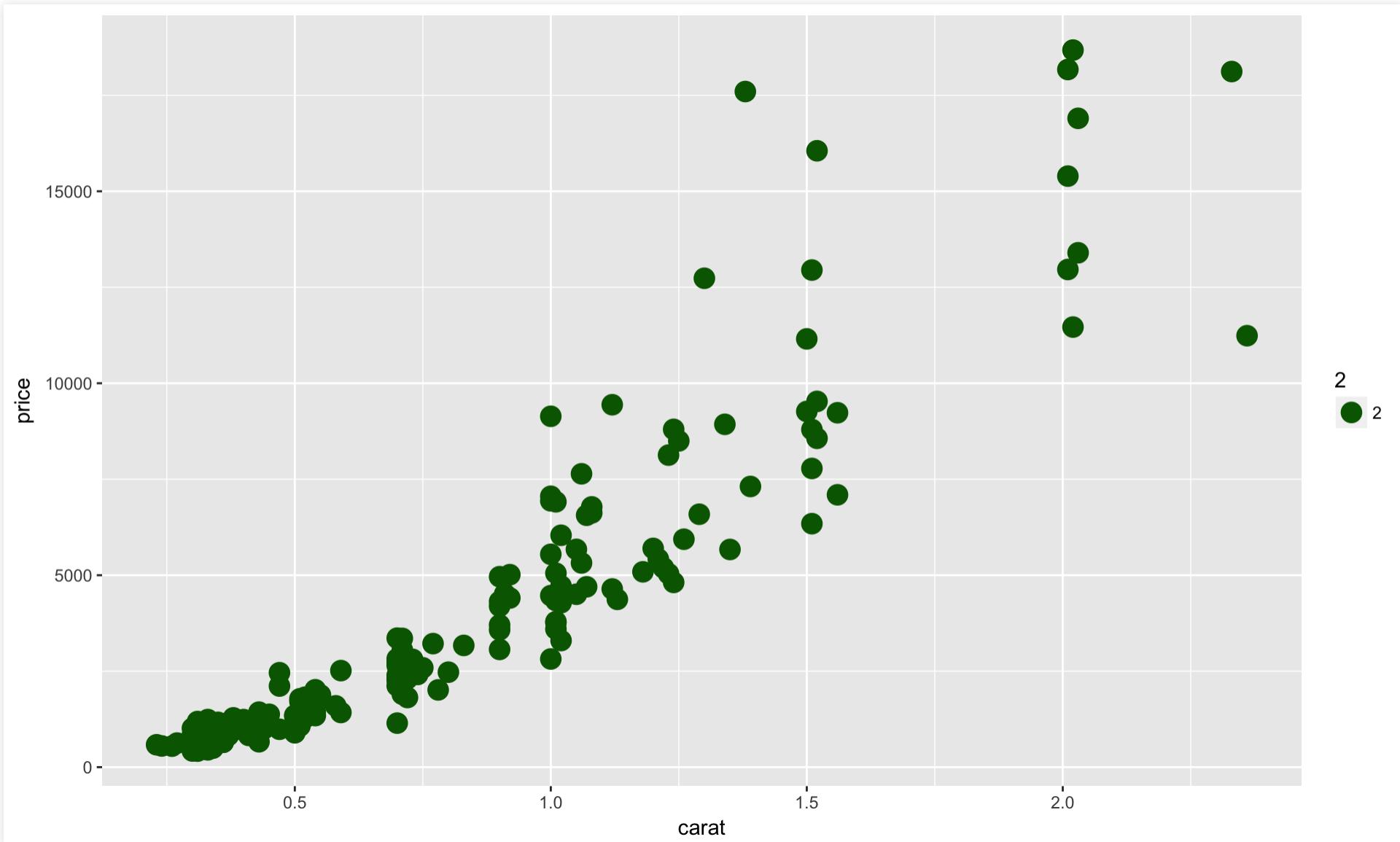
Set color and shape

```
p1 + geom_point(aes(shape = cut, color = color))
```

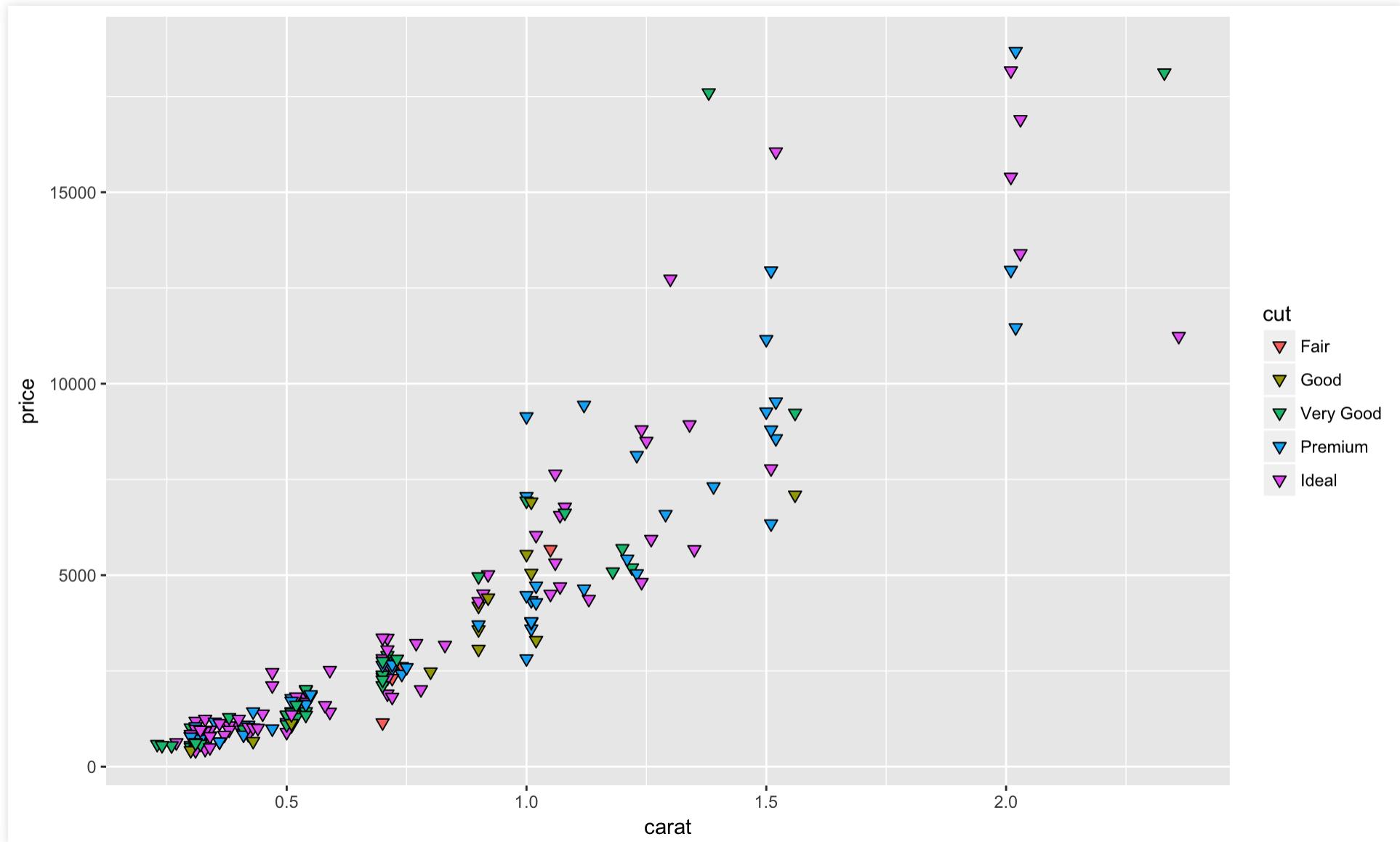


Variable vs fixed aesthetics

```
ggplot(data = dsmall, aes(x = carat, y = price)) +  
  geom_point(aes(size = 2), color = "darkgreen")
```

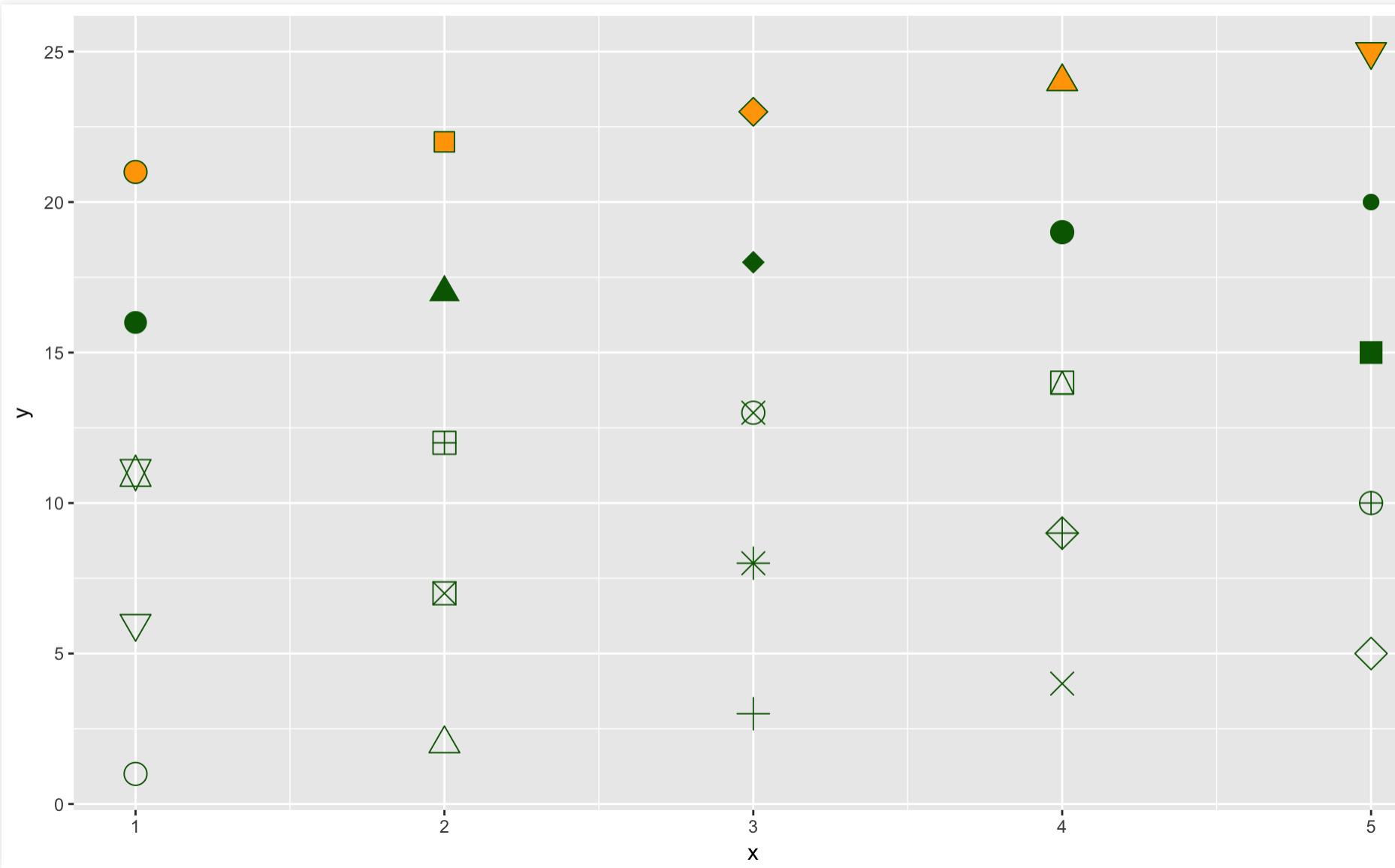


```
ggplot(data = dsmall, aes(x = carat, y = price)) +  
  geom_point(aes(fill = cut), size = 2, color = "black", shape = 25)
```



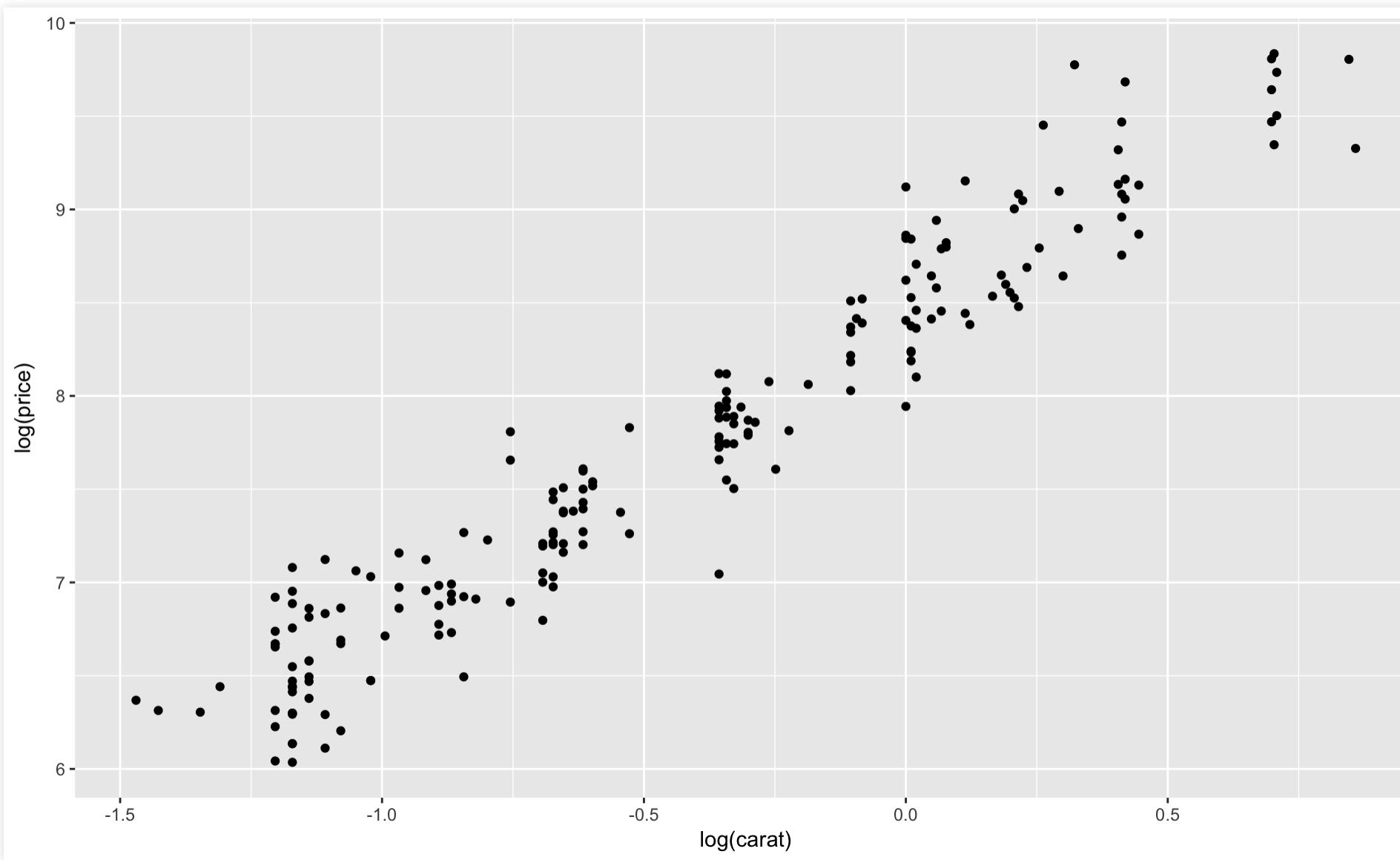
All 25 shape configurations

```
ggplot(data.frame(x = 1:5 , y = 1:25 , z = 1:25) , aes(x = x, y = y)) +  
  geom_point(aes(shape = z), size = 5, colour = "darkgreen", fill = "orange") +  
  scale_shape_identity()
```



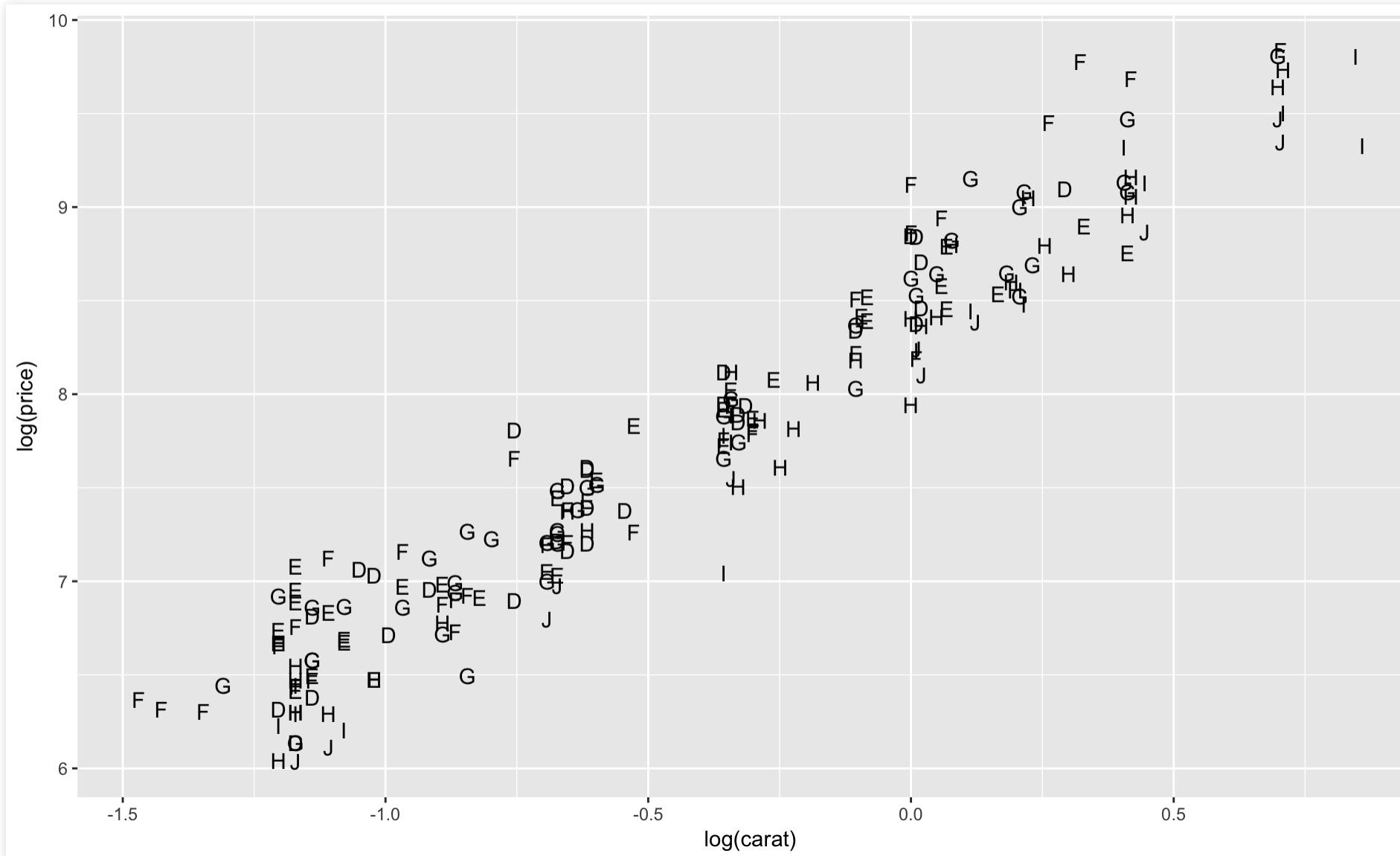
Data transformations

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



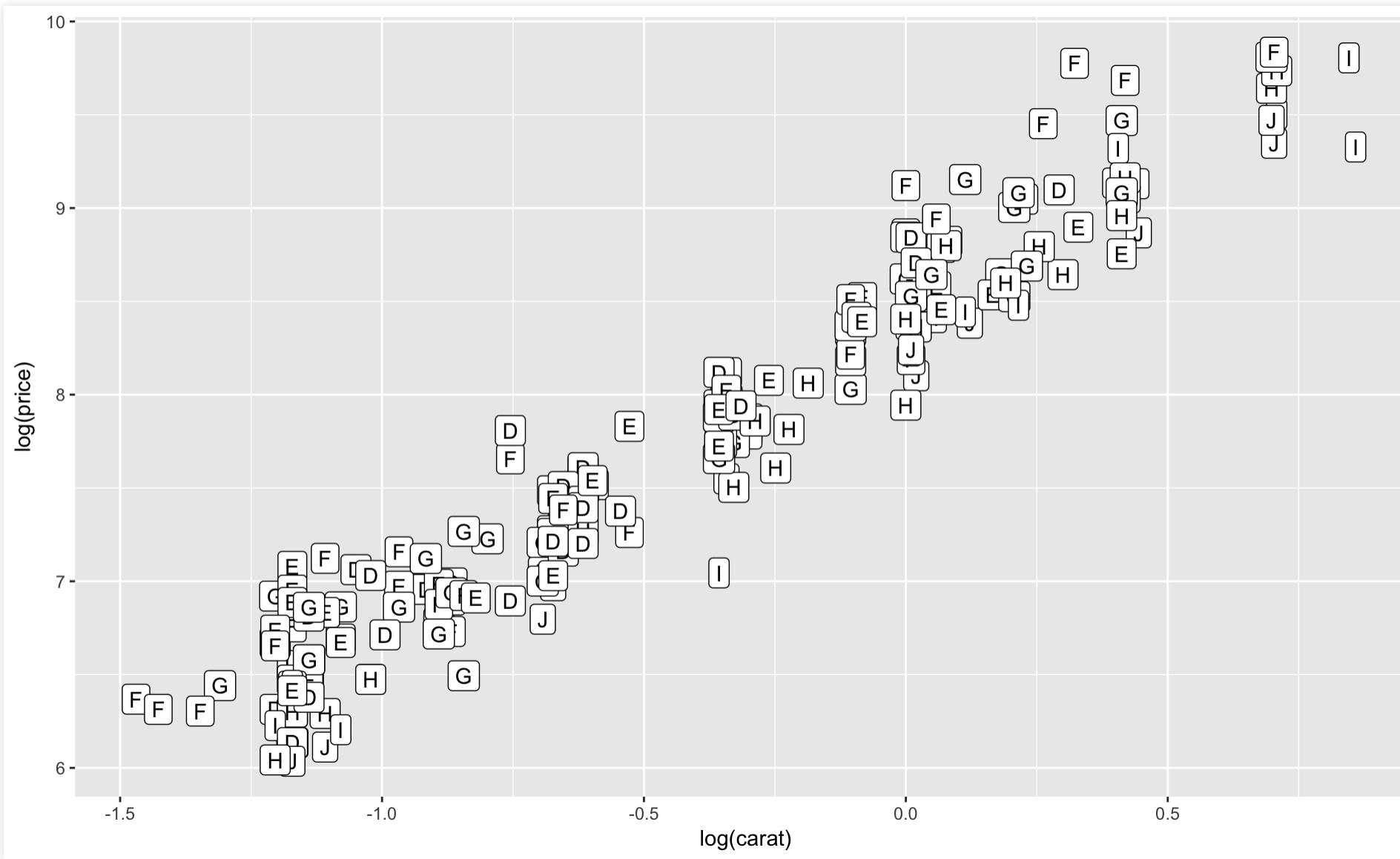
Text labels

```
p2 <- ggplot(dsmall, aes(x = log(carat), y = log(price)))  
p2 + geom_text(aes(label = color))
```



Text with rectangle plates

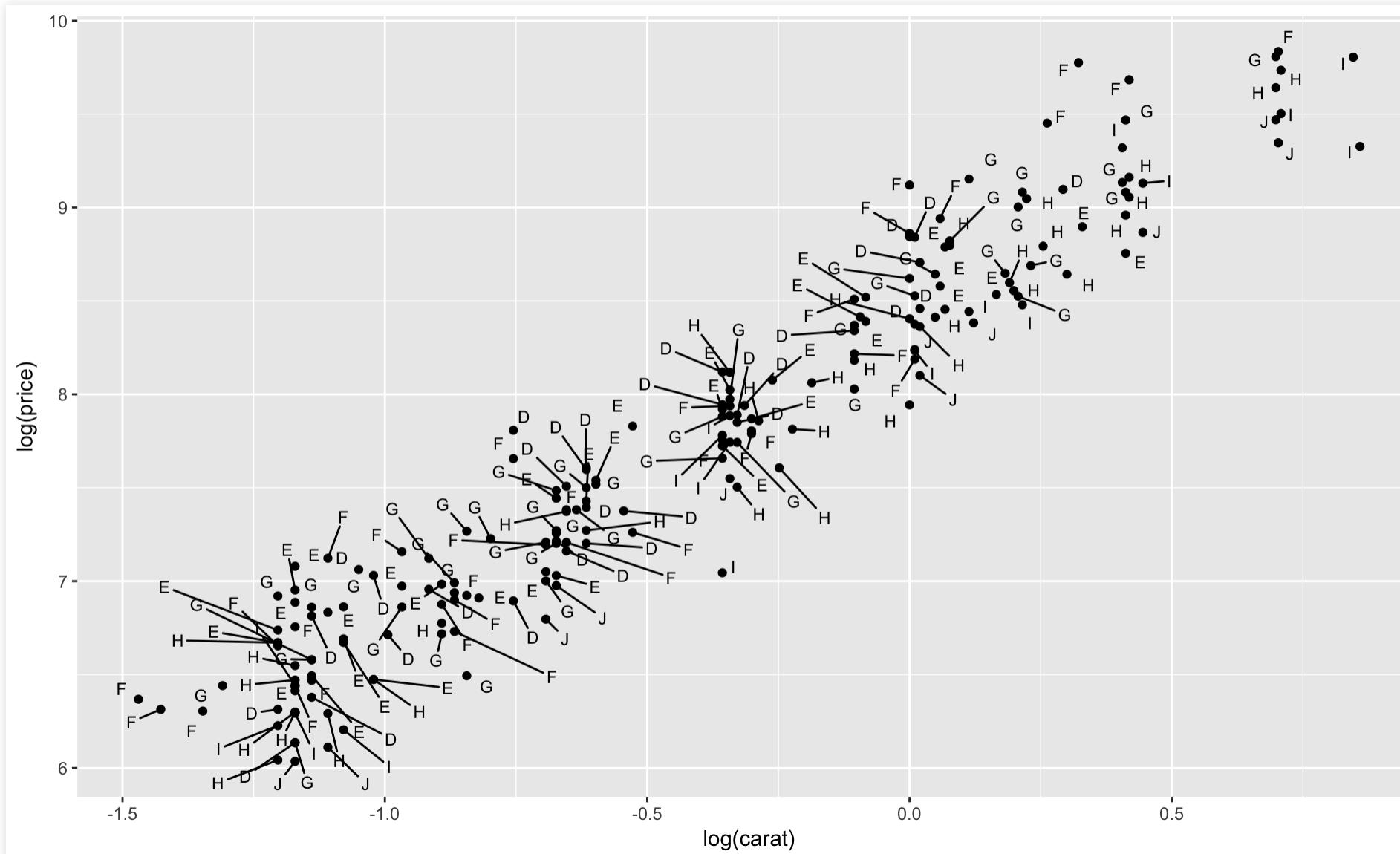
```
p2 + geom_label(aes(label = color))
```



ggrepel package for annotation

ggrepel helps annotating overlapping labels.

```
library(ggrepel)  
p2 + geom_point() + geom_text_repel(aes(label=color), size = 3)
```



Exercise 1

- Go to “Lec4_Exercises.Rmd” on the class website.
- Complete Exercise 1.

Statistical Transformations

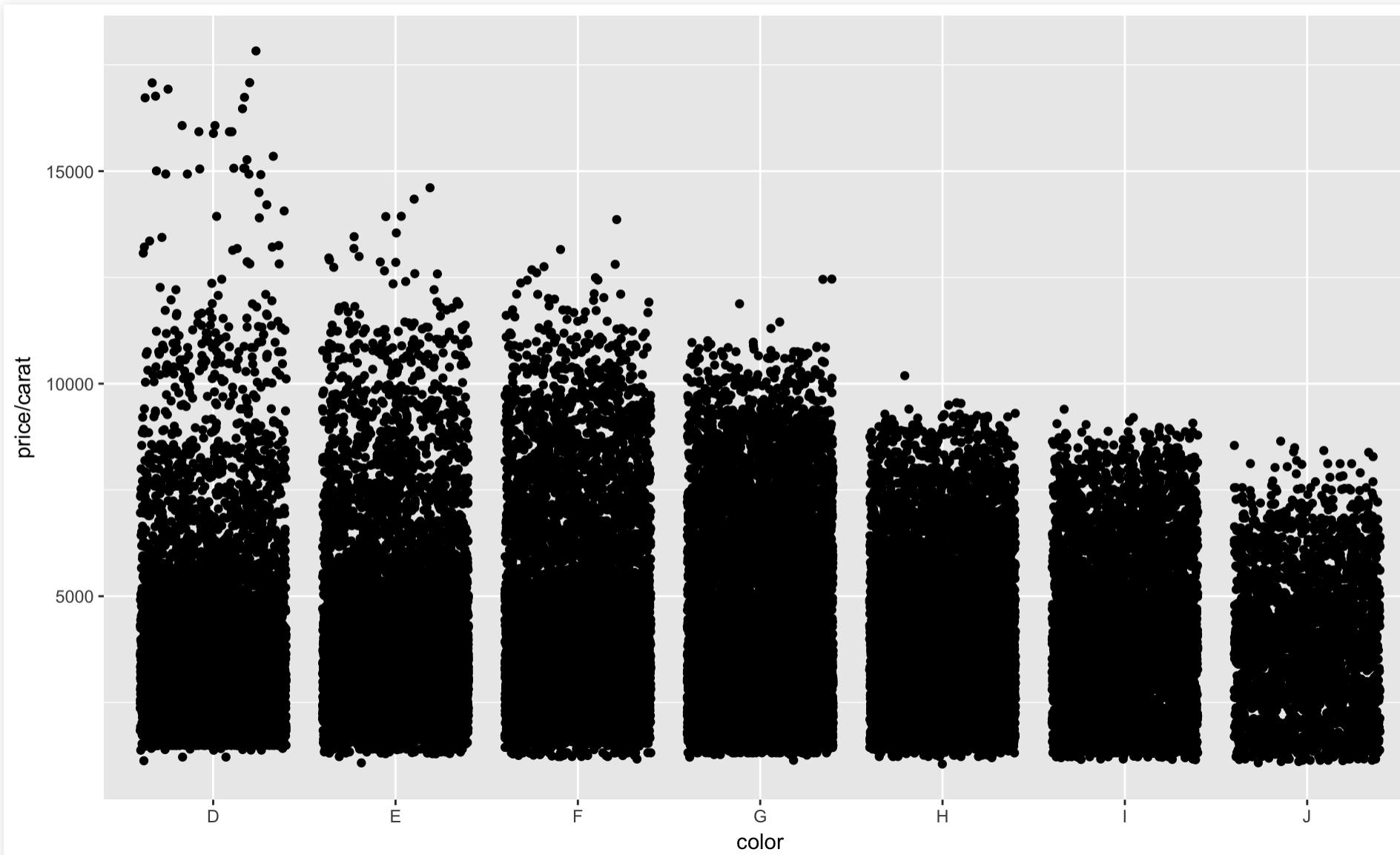
Types of statistical transformations

Plots often require some statistical data transformation or computation before they can be plotted:

- **boxplots:** calculate the median, lower and upper quartiles,
- **histograms:** group the values into bins,
- **smoothers:** prediction lines / predicted y-values,
- **bar charts:** number of class occurrences.

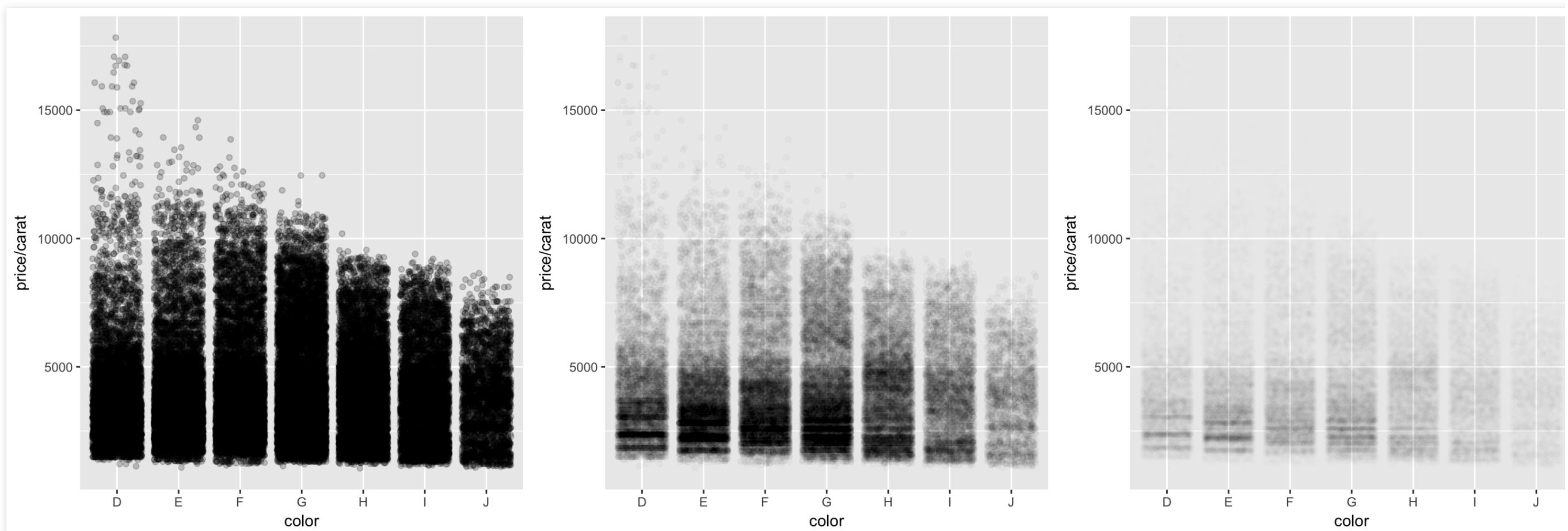
Jittered points

```
ggplot(data = diamonds, aes(x = color, y = price/carat)) +  
  geom_jitter()
```



Alpha parameter for transparency

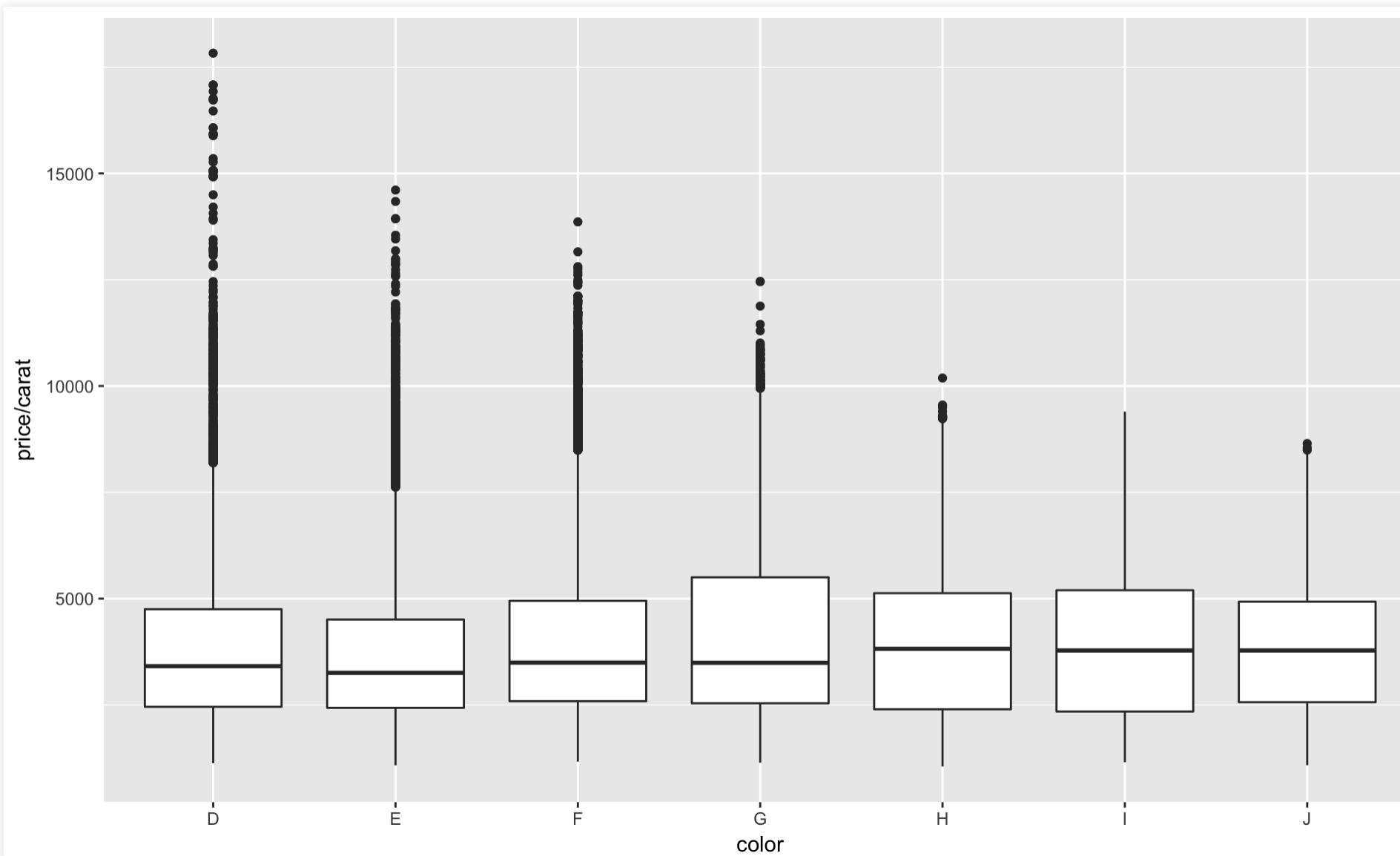
```
j1 <- ggplot(data = diamonds, aes(x = color, y = price/carat)) + geom_jitter(alpha = 1/5)
j2 <- ggplot(data = diamonds, aes(x = color, y = price/carat)) + geom_jitter(alpha = 1/50)
j3 <- ggplot(data = diamonds, aes(x = color, y = price/carat)) + geom_jitter(alpha = 1/200)
# We use grid.arrange from gridExtra to display multiple plots
grid.arrange(j1, j2, j3, ncol = 3)
```



Box plot transformation

Plotting a summary (less data) can be more insightful.

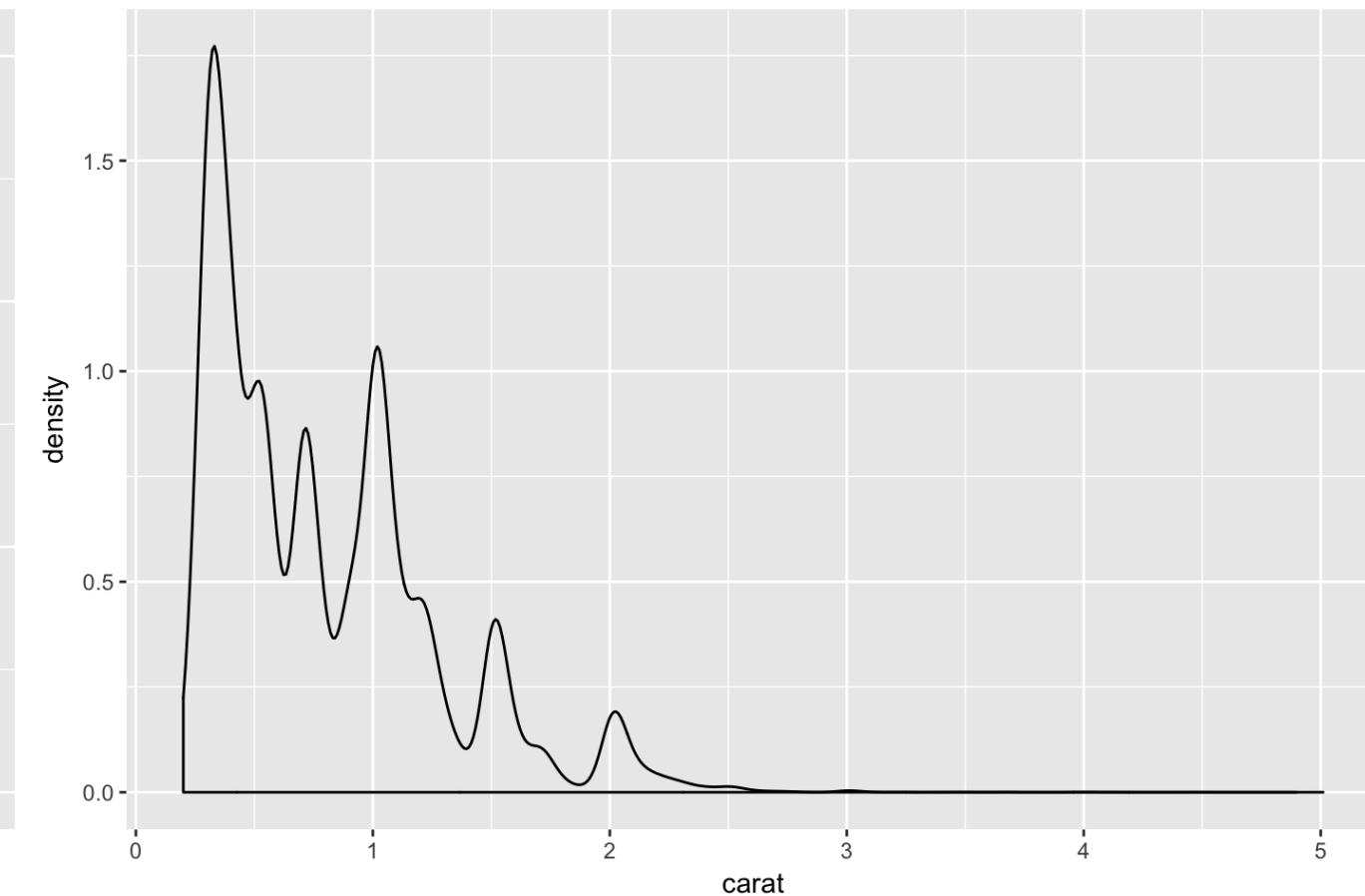
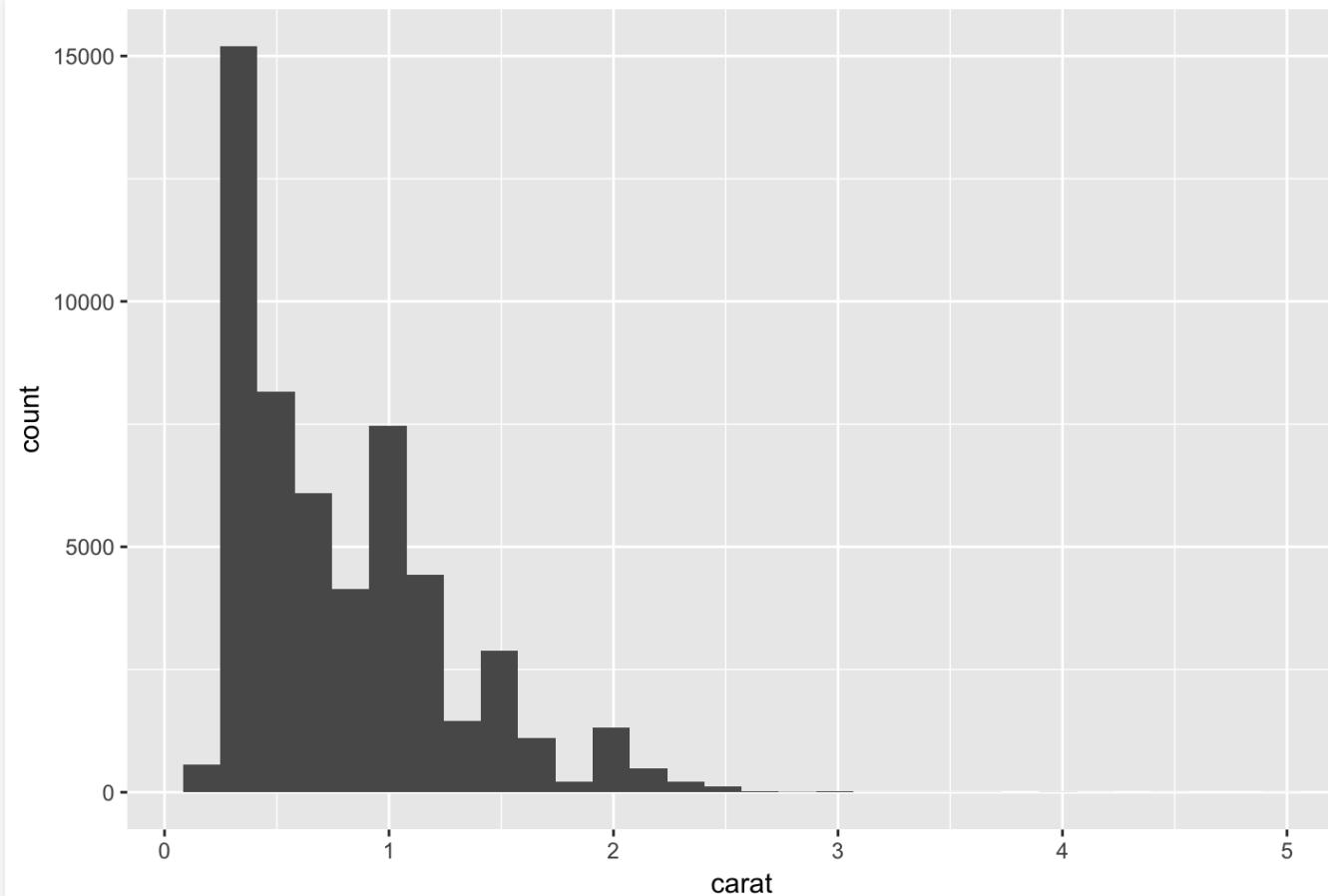
```
ggplot(data = diamonds, aes(x = color, y = price/carat)) +  
  geom_boxplot()
```



Histogram and density plots

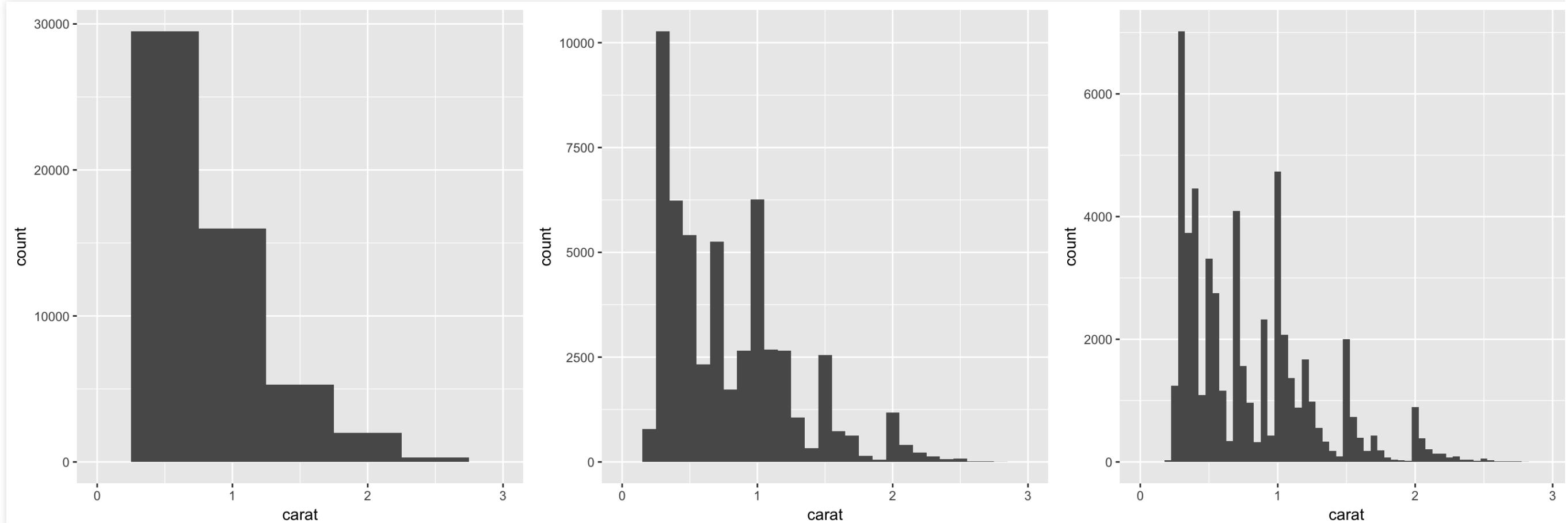
```
# Distribution of the carats (weights) of the diamonds.  
h <- ggplot(data = diamonds, aes(x = carat)) + geom_histogram()  
d <- ggplot(data = diamonds, aes(x = carat)) + geom_density()  
grid.arrange(h, d, ncol = 2)
```

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

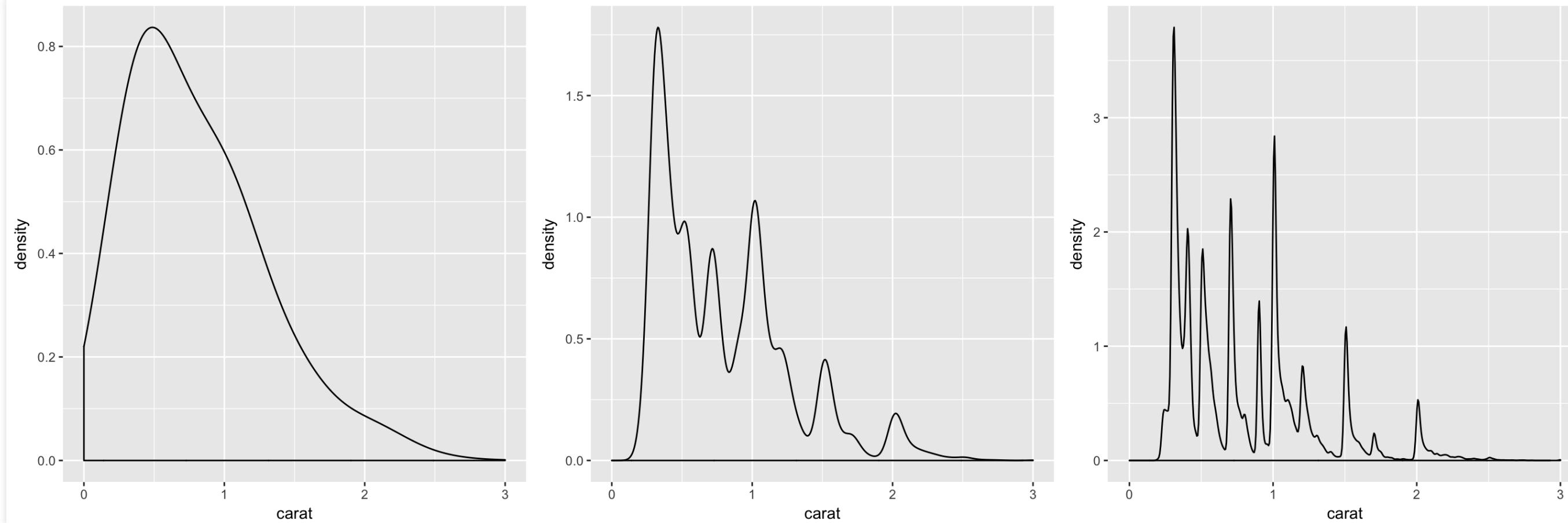


- For the histogram, the smoothness is controlled with **bins** and **binwidth** arguments. (Break points can also be specified explicitly, using the **breaks** argument.)
- For the density plot, the **bw** (the smoothing bandwidth) and **adjust** argument controls the degree of smoothness (high values of adjust produce smoother plots).

```
p <- ggplot(data = diamonds, aes(x = carat)) + xlim(0, 3)  
h1 <- p + geom_histogram(binwidth = 0.5)  
h2 <- p + geom_histogram(binwidth = 0.1)  
h3 <- p + geom_histogram(binwidth = 0.05)  
grid.arrange(h1, h2, h3, ncol = 3)
```

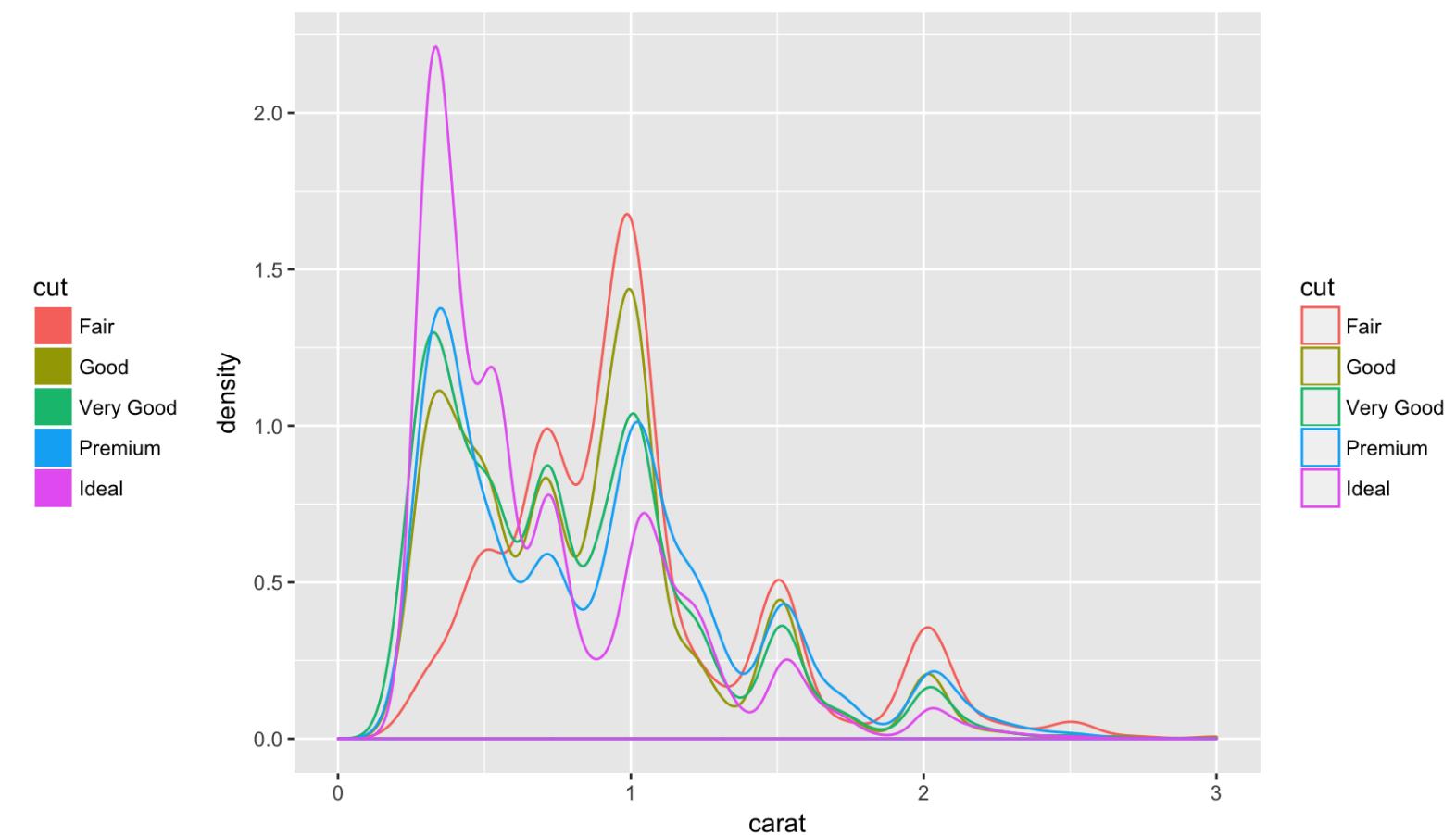
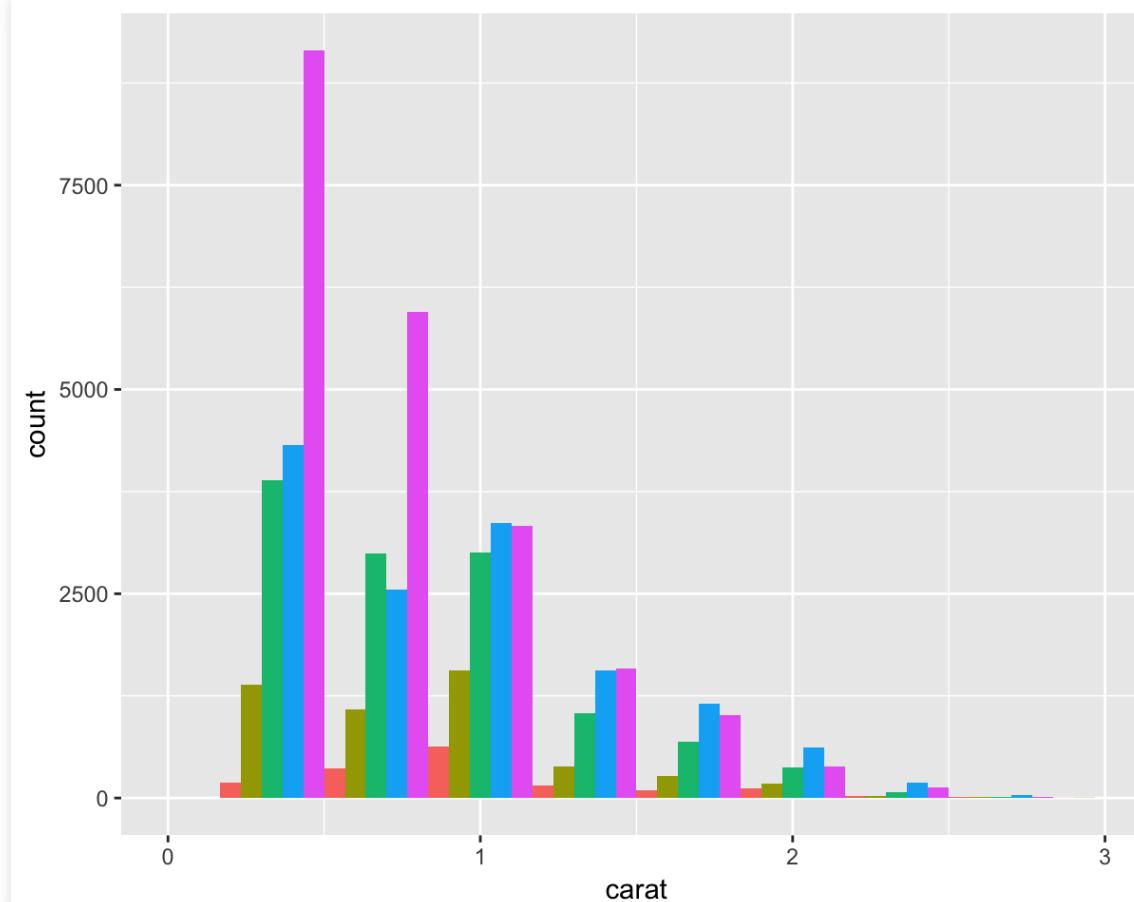


```
d1 <- p + geom_density(adjust = 5)
d2 <- p + geom_density(adjust = 1)
d3 <- p + geom_density(adjust = 1/5)
grid.arrange(d1, d2, d3, ncol = 3)
```



Histograms for separate groups

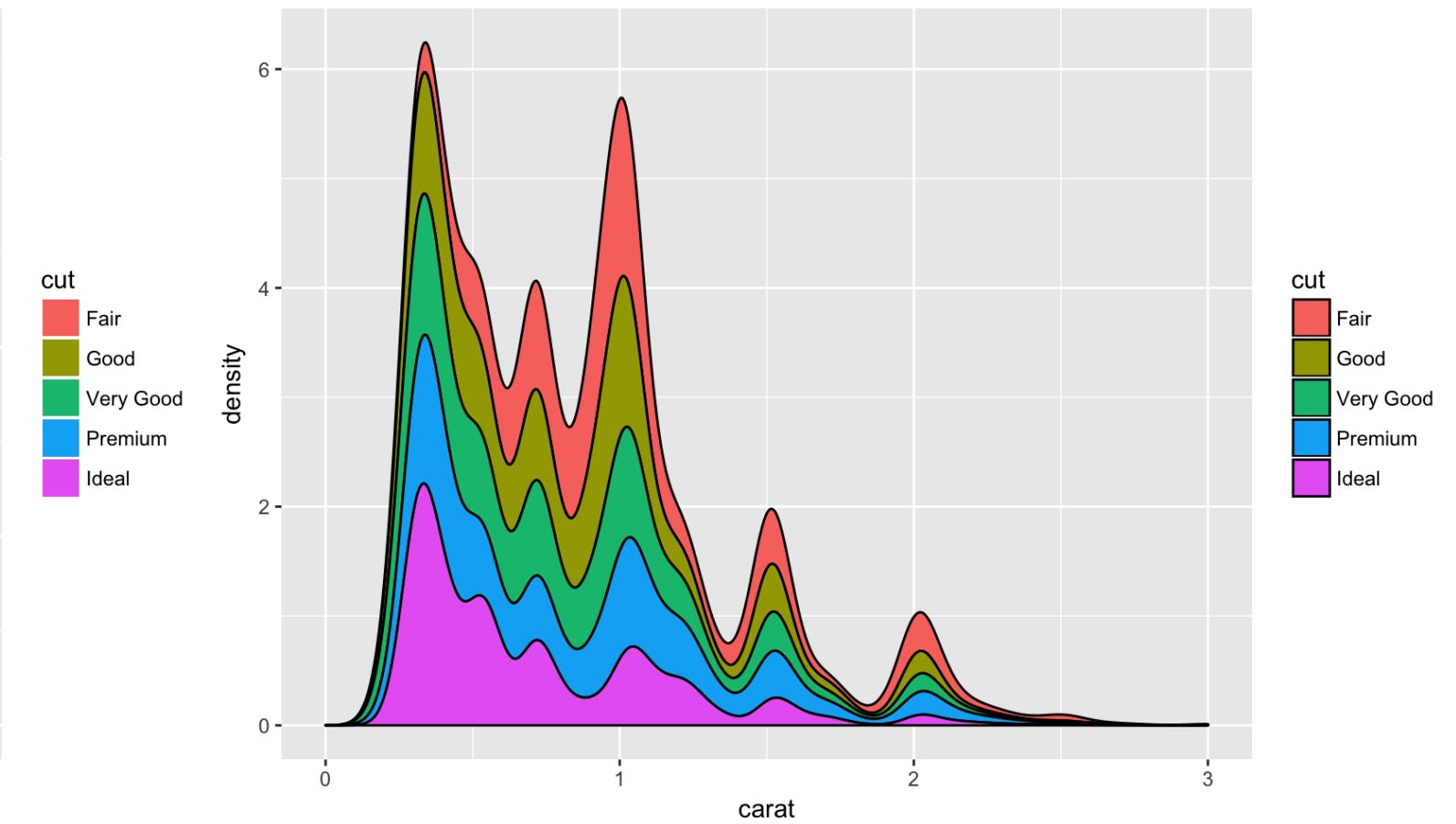
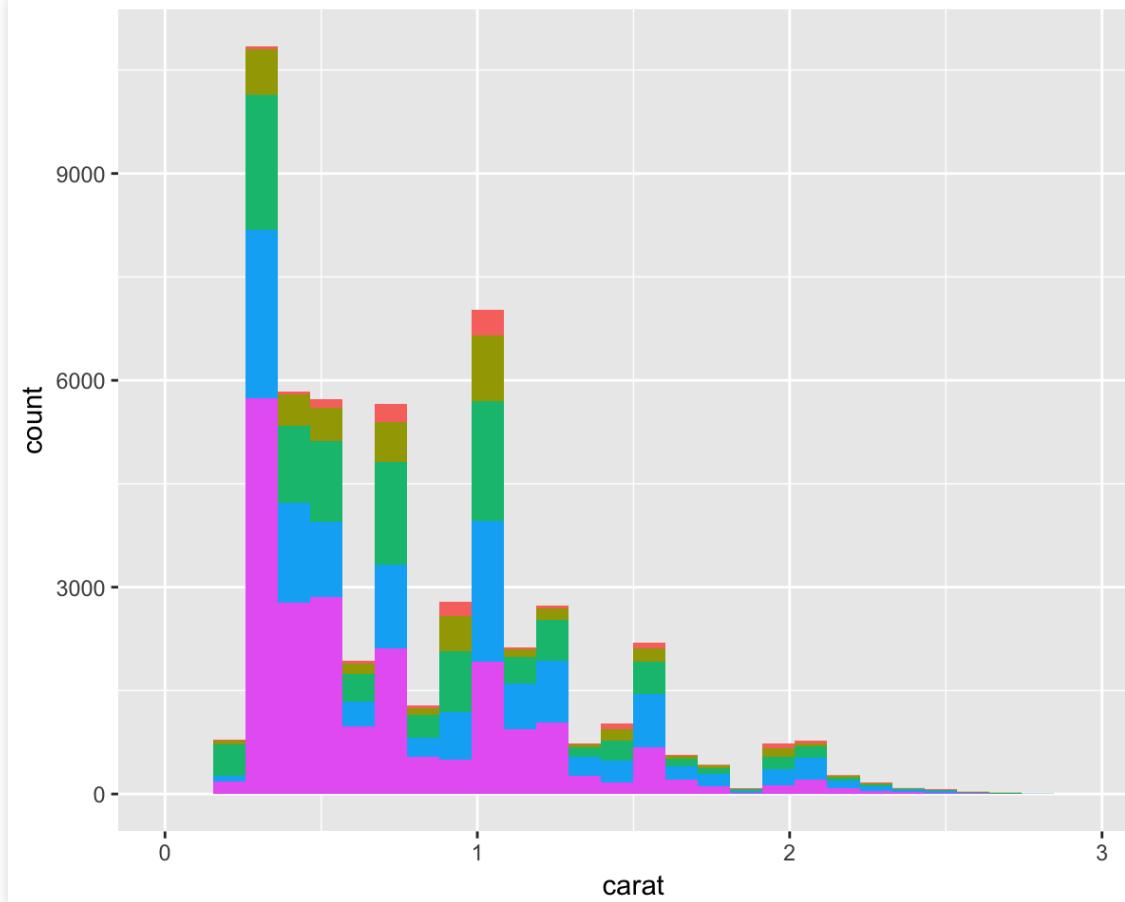
```
# Here we show grouping by diamonds cut.  
h <- p + geom_histogram(aes(fill = cut), position = "dodge", bins = 10)  
d <- p + geom_density(aes(color = cut))  
grid.arrange(h, d, ncol = 2)
```



Instead of marginal distributions, we can plot distribution of components **stack** on top of each other to see the contribution from each of group.

```
h <- p + geom_histogram(aes(fill = cut), position = "stack")
d <- p + geom_density(aes(fill = cut), position = "stack")
grid.arrange(h, d, ncol = 2)
```

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



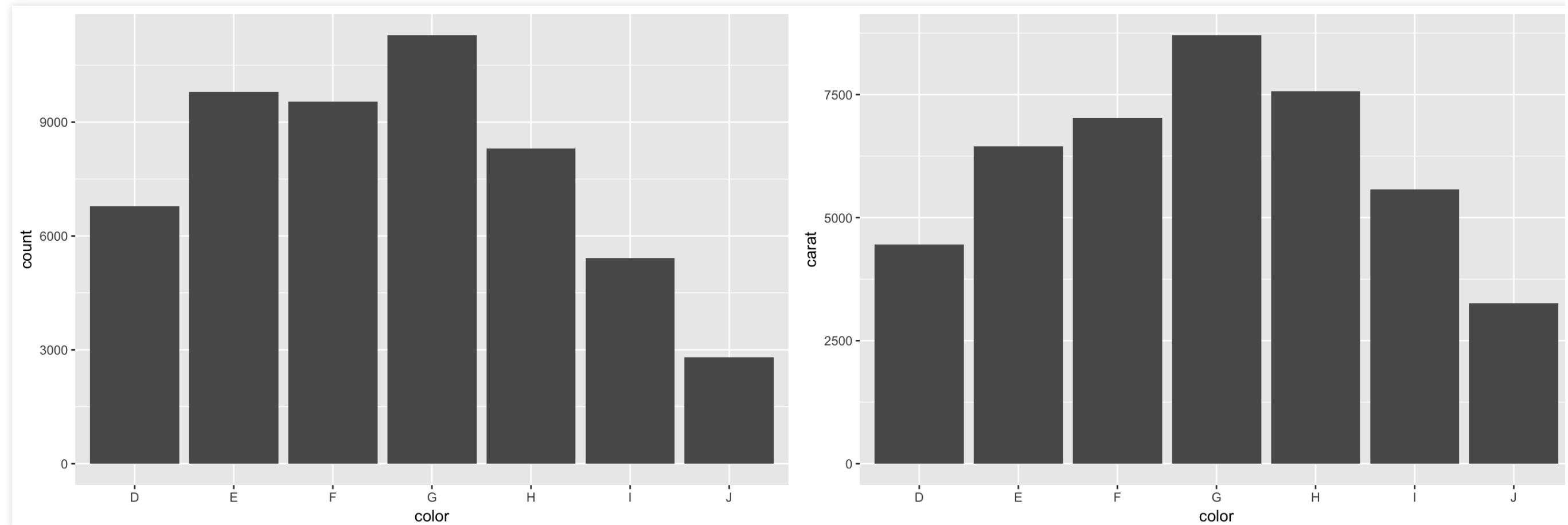
Bar charts

- A discrete analogue of a histogram is the bar chart, `geom_bar()`.
- Instead of partitioning the values into bins like histograms, the bar geom **counts the number of instances of each discrete class**. The counts are then plotted as columns for each distinct class.
- If you'd like include **unequal weights** for different observations, you can use the `weight` aesthetic.

```

b1 <- ggplot(diamonds, aes(x = color)) + geom_bar()
b2 <- ggplot(diamonds, aes(x = color)) + geom_bar(aes(weight = carat)) + ylab("carat")
grid.arrange(b1, b2, ncol = 2)

```



The left plot shows the number of diamonds in each color group, and the right plot shows the count weighted by carat, which is equivalent to showing the total weight of diamonds in each color group.

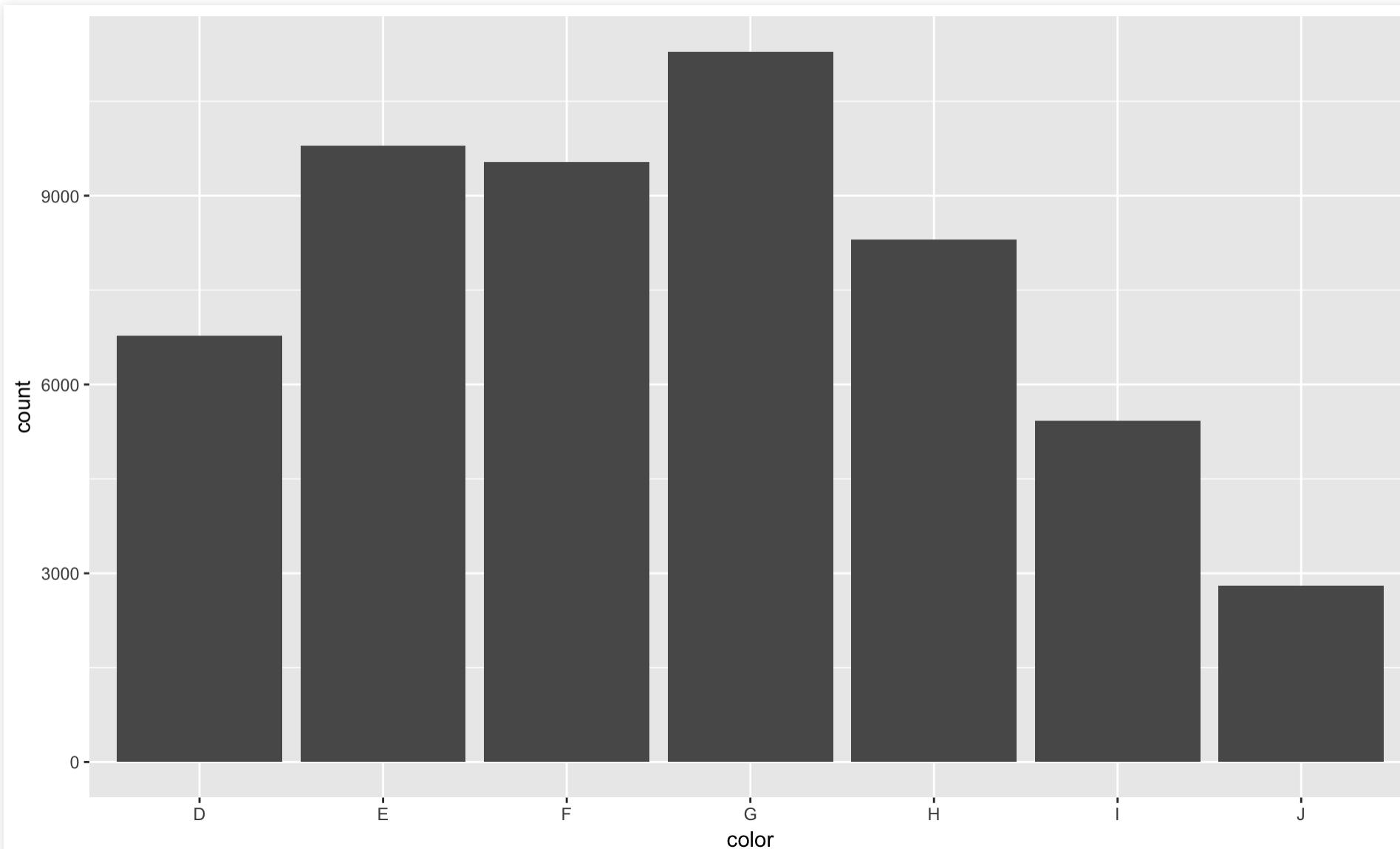
- As you see, in ggplot2 (unlike base graphics) it is **not necessary tabulate the values**, i.e. compute the counts of each category beforehand. The computation is done automatically for you.
- However, if you have already summarized data, you can still use `geom_bar` but you need to specify an identity transformation, `stat = "identity"` rather than the default `stat = "count"`.

```
diamond.counts <- table(diamonds[ "color" ])  
df <- data.frame(diamond.counts)  
colnames(df) <- c( "color", "count" )  
df
```

```
## color count  
## 1 D 6775  
## 2 E 9797  
## 3 F 9542  
## 4 G 11292  
## 5 H 8304  
## 6 I 5422  
## 7 J 2808
```

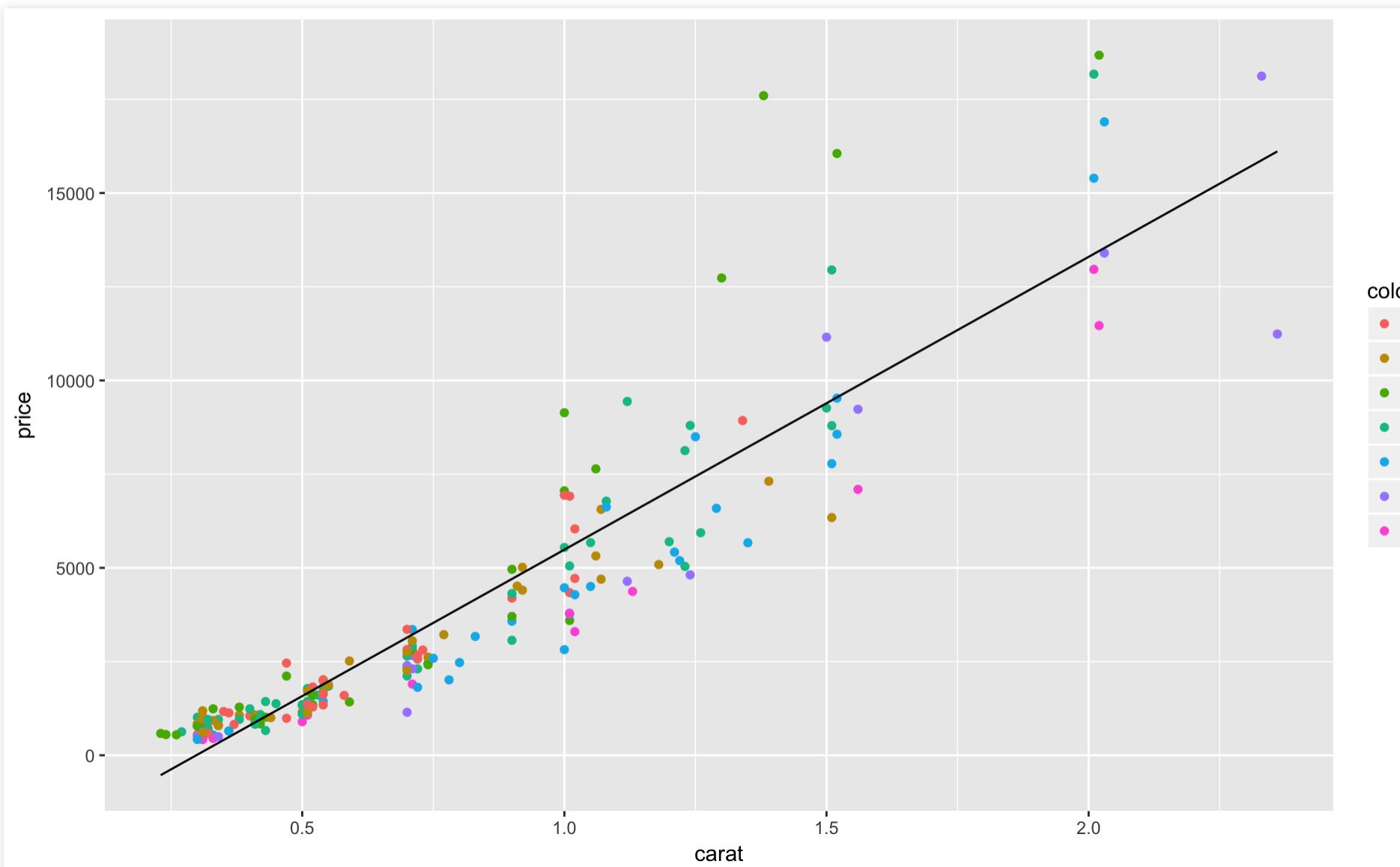
```
# The default option generates an error:  
ggplot(df, aes(x=color, y=count)) + geom_bar()  
# Error: stat_count() must not be used with a y aesthetic.
```

```
# You need to do the following:  
ggplot(df, aes(x=color, y=count)) + geom_bar(stat="identity")
```



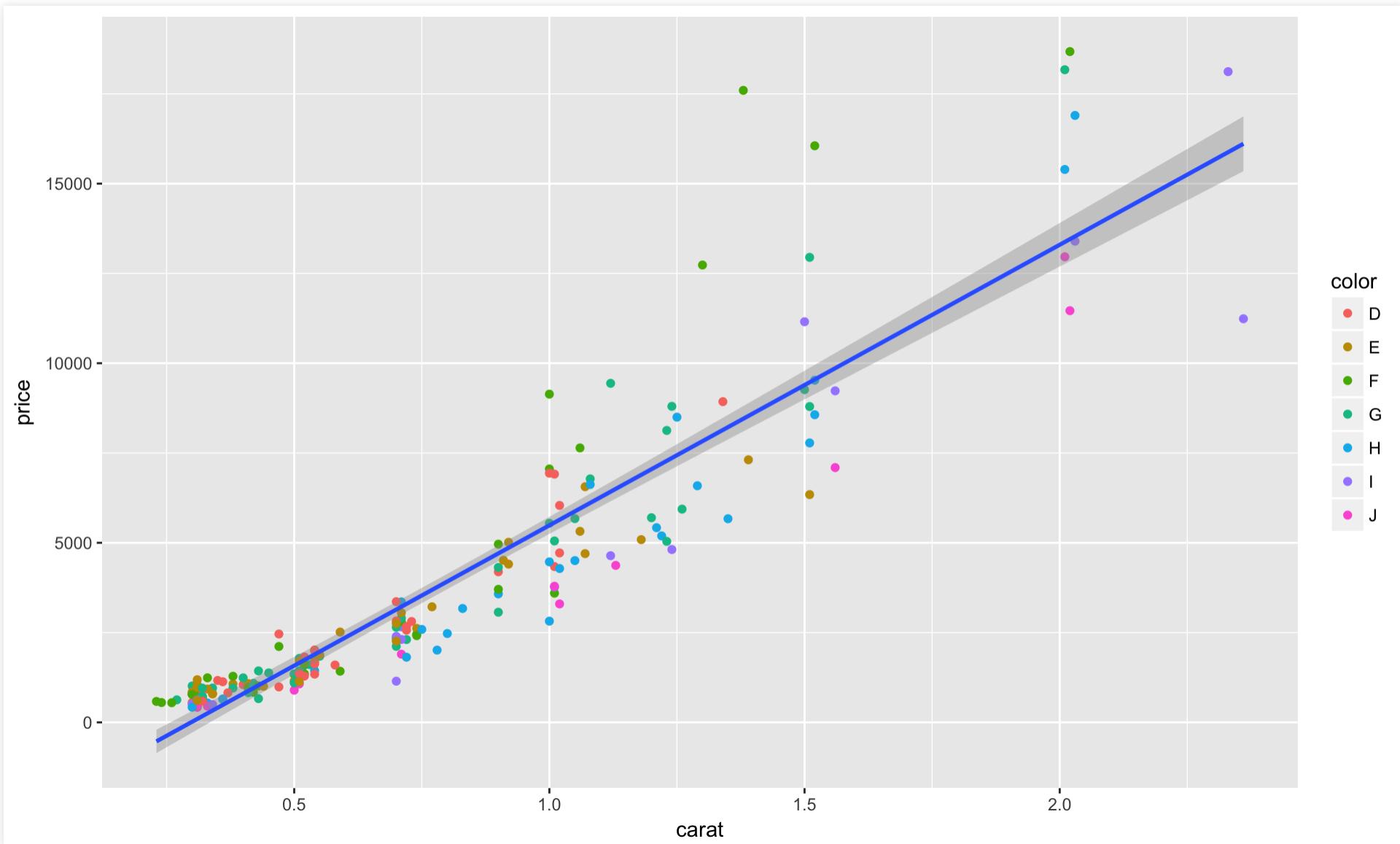
Regression lines

```
# You can fit a linear model with lm(y ~ x, data) and use it for prediction.  
dsmall$pred.price <- predict(lm(price ~ carat, data = dsmall))  
# And add the regression line in a standard way  
p1 <- ggplot(dsmall, aes(x = carat, y = price))  
p1 + geom_point(aes(color = color)) + geom_line(aes(y = pred.price))
```



Regression lines with ggplot2

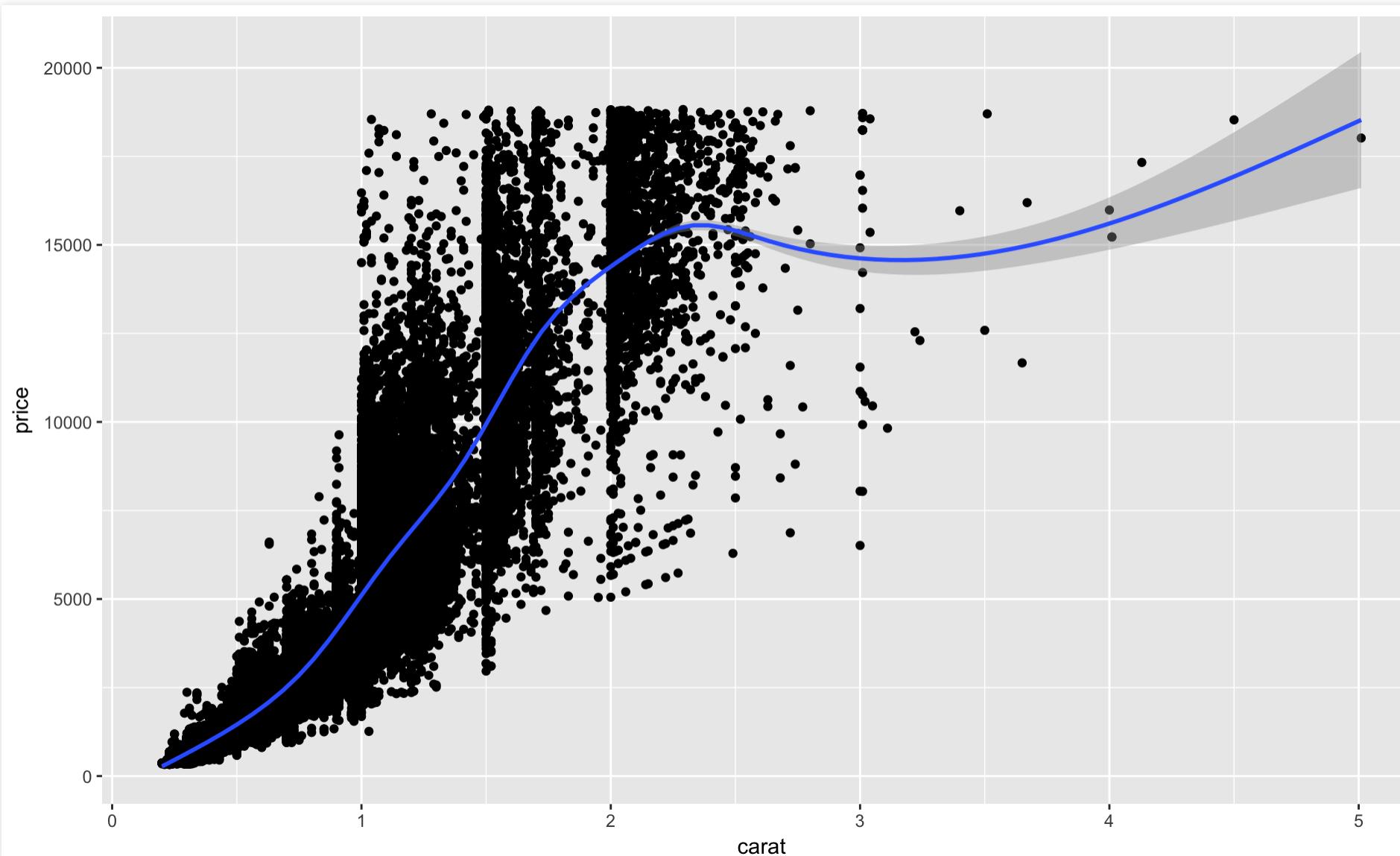
```
# Or you can simply use geom_smooth()!  
p1 + geom_point(aes(color = color)) + geom_smooth(method = "lm")
```



Smoothers are trend lines

```
# Smoothers help discern patterns in the data  
ggplot(data = diamonds, aes(x = carat, y = price)) +  
  geom_point() + geom_smooth()
```

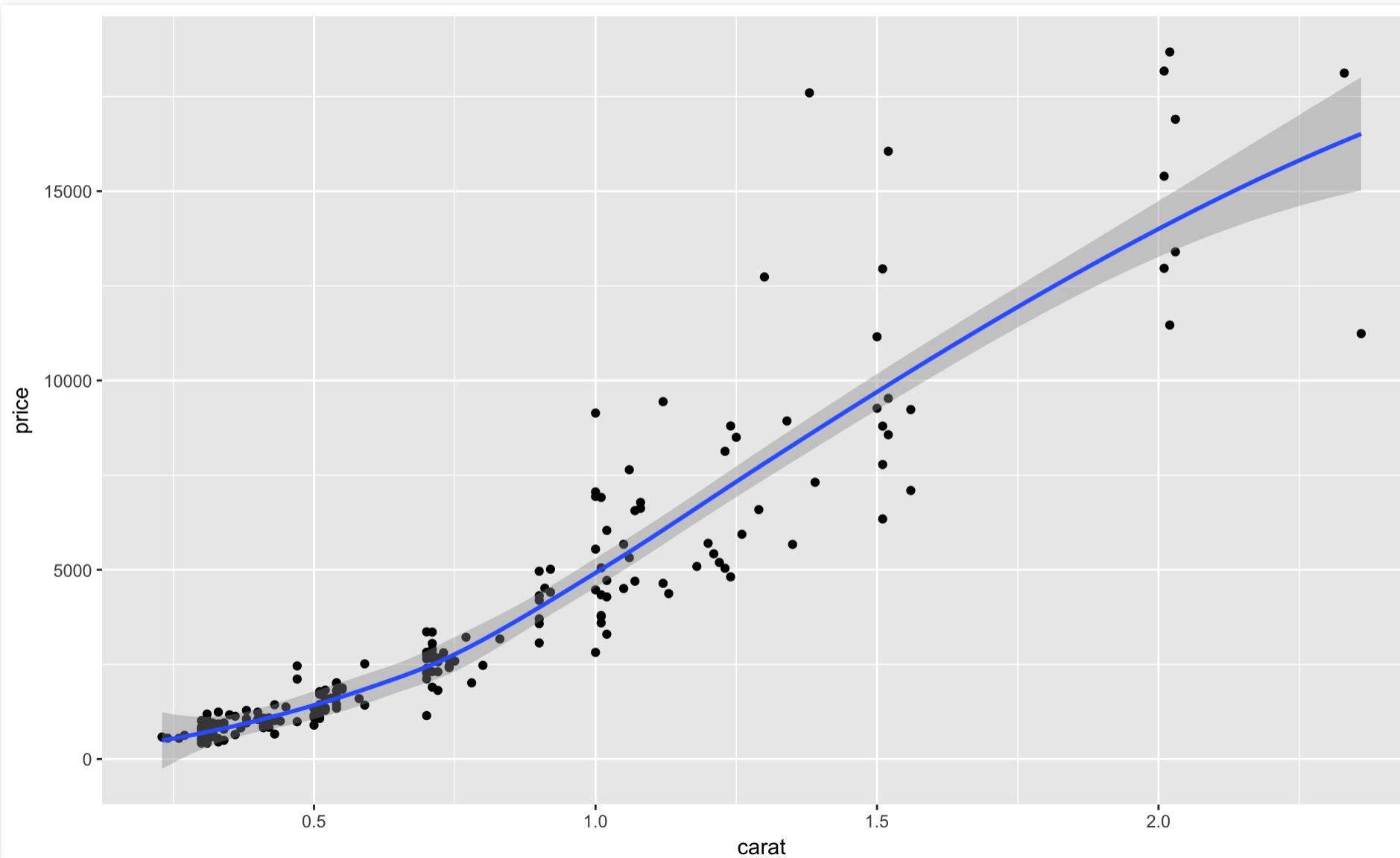
```
## `geom_smooth()` using method = 'gam'
```



For the small subset of the diamonds dataset we have:

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

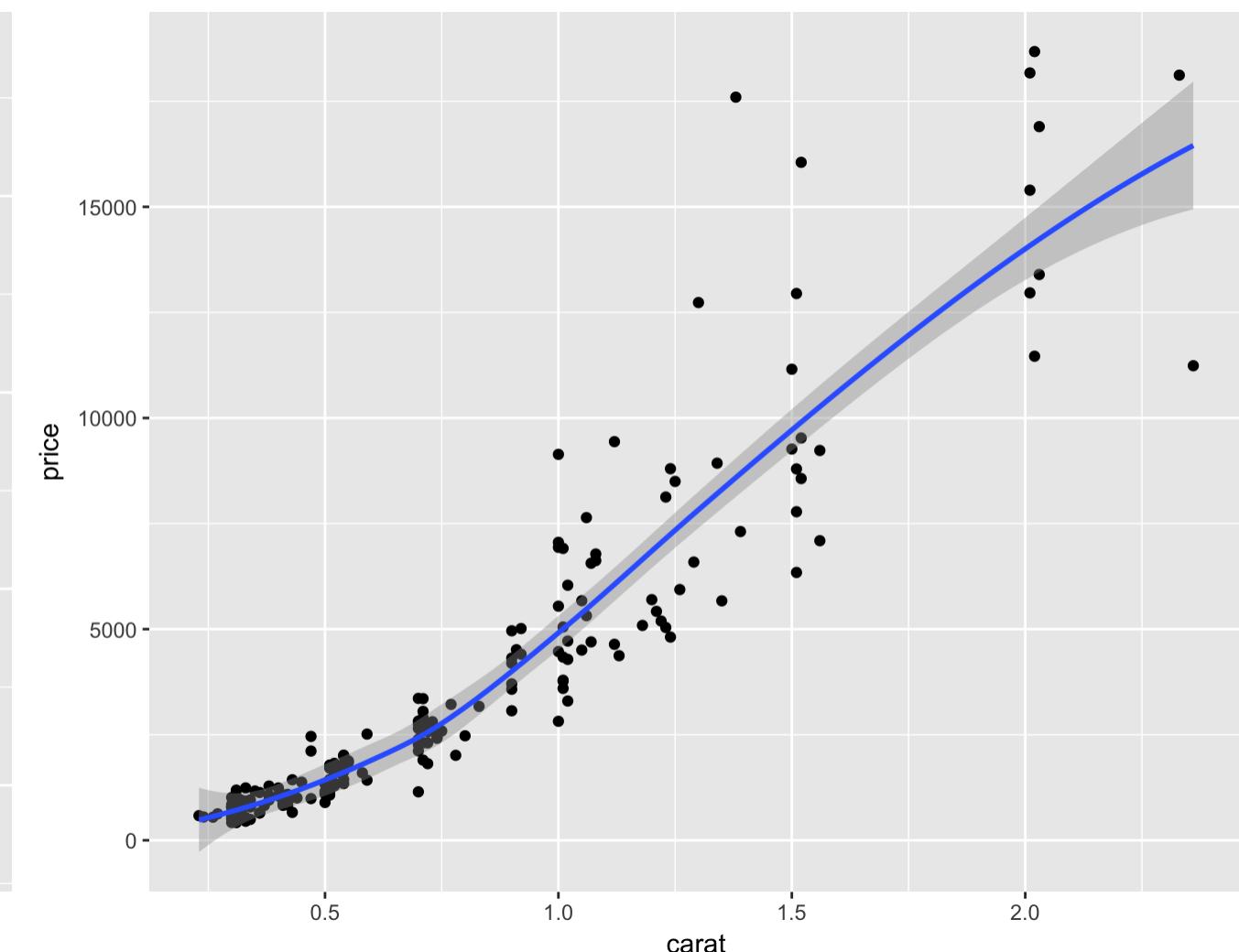
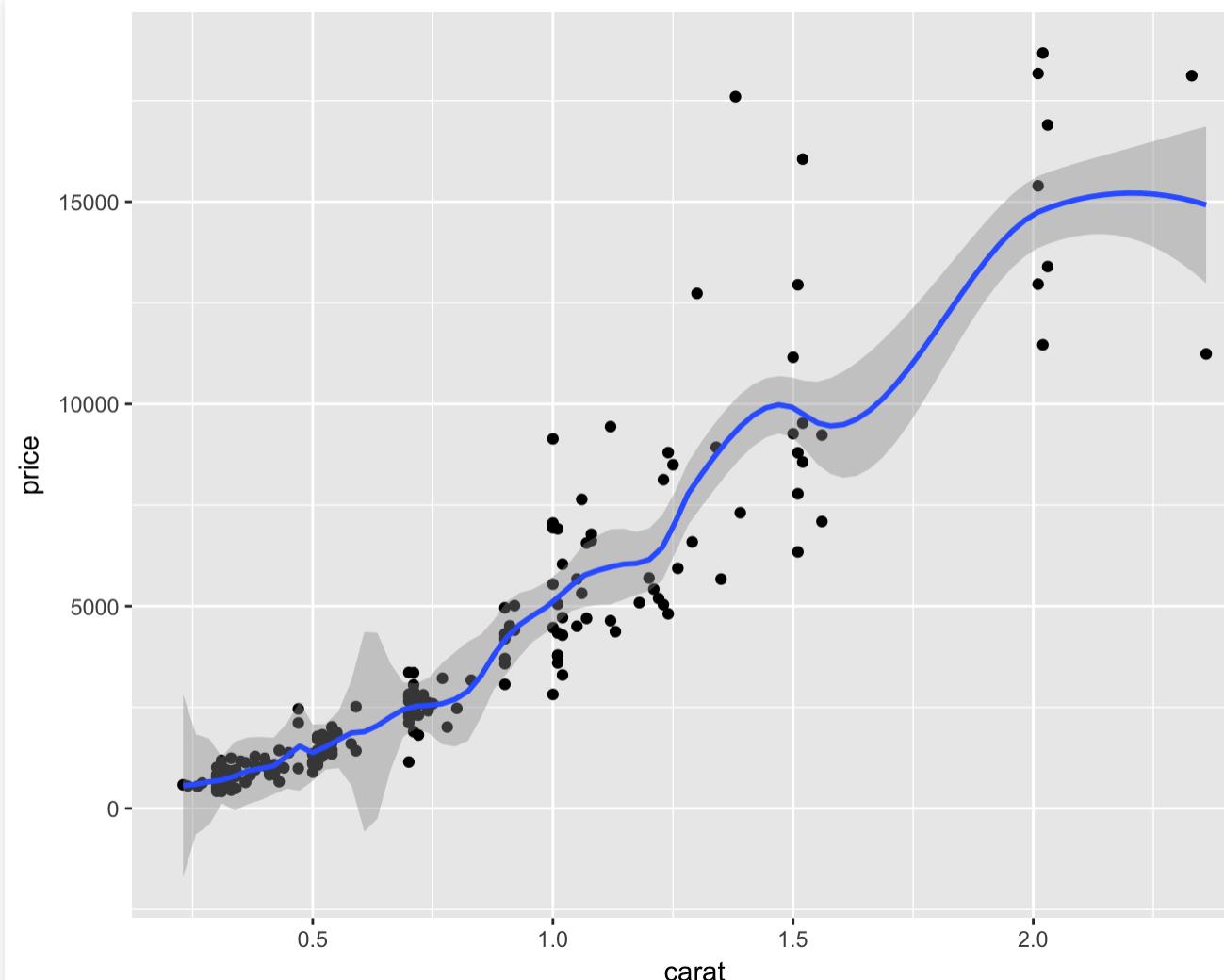
```
## `geom_smooth()` using method = 'loess'
```



'span' determines the wigglyness of the curve (smaller span => more wiggly)

```
grid.arrange(p1 + geom_point() + geom_smooth(span = 0.2),  
p1 + geom_point() + geom_smooth(span = 0.7), ncol = 2)
```

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



- By default, when applied to datasets with a few observations ($n < 1000$) `geom_smooth()` uses a `method = "loess"` for fitting a curve.
- This method uses a smooth local regression. More details about the algorithm used can be found in `?loess`.
- Loess does not work well for large datasets (it's $O(n^2)$ in memory), and so an alternative smoothing algorithm is used when n is greater than 1,000.

Exercise 2: The Economist Data

- Go back to “Lec4_Exercises.Rmd”
- Complete the exercise 2

Scales

Aesthetic mapping vs variable scaling

- `aes()` is responsible for assigning an aesthetic to a variable; it doesn't determine how mapping should be done.
- For example, `aes(shape = x)` or `aes(color = z)` do not specify what shapes or what colors should be used. To choose colors/shapes/sizes etc. you need to **modify the corresponding scale**.
- `ggplot2` includes scales for:
 - position
 - color and fill
 - size
 - shape
 - line type

Scales can be modified with functions of the form:

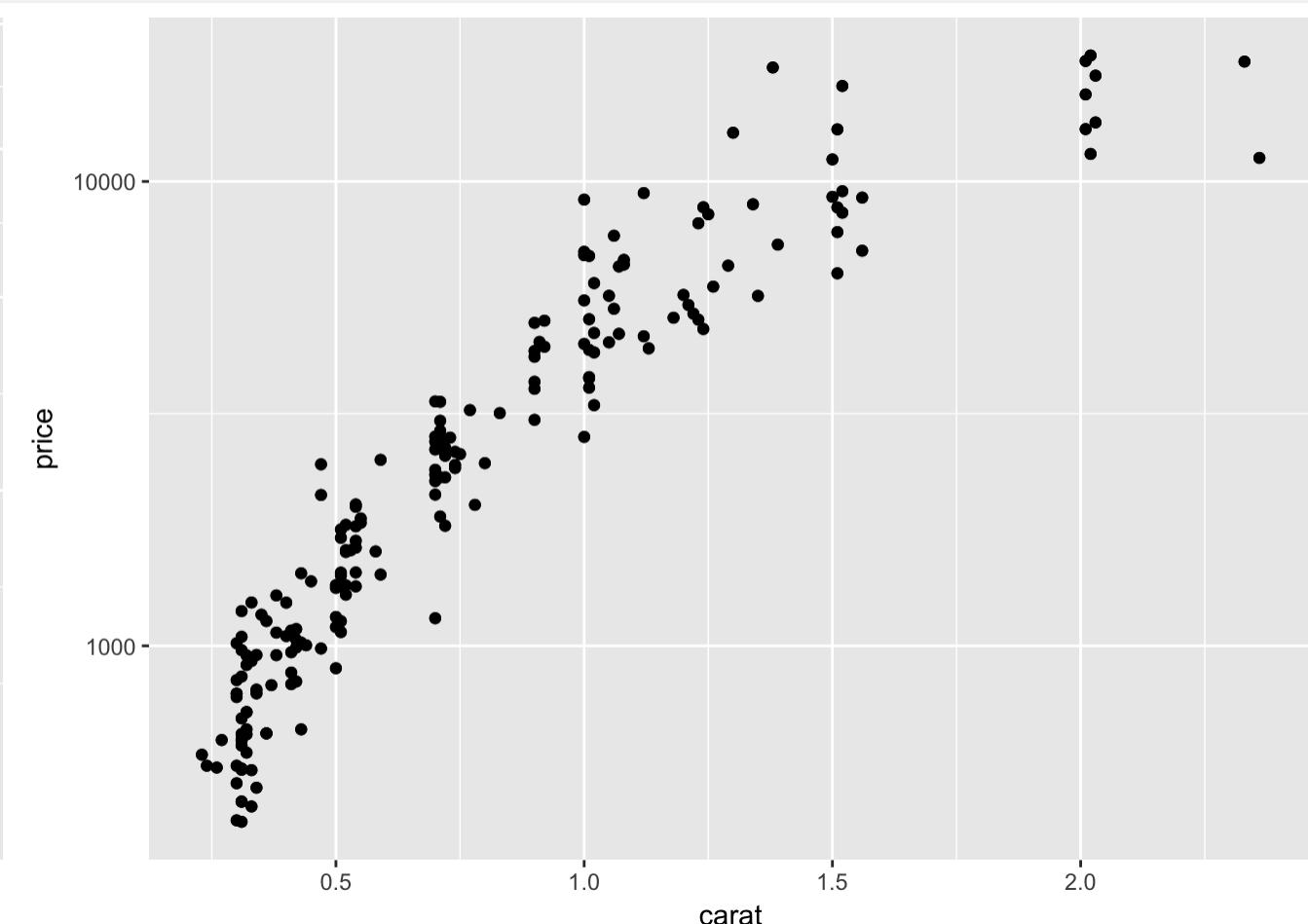
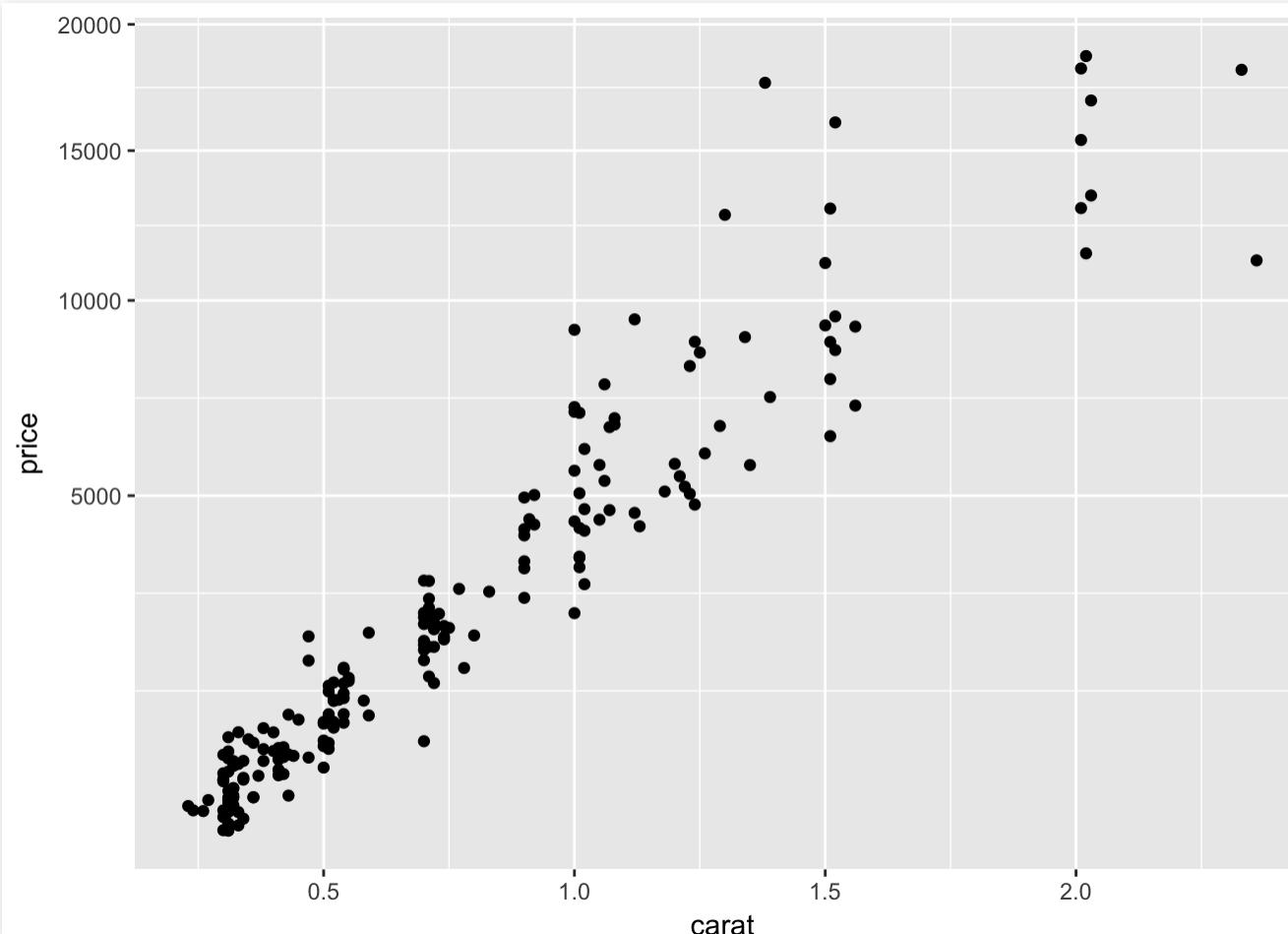
`scale_<aesthetic>_<type>()`, e.g. `scale_color_discrete()`.

Try typing `scale_<tab>()` to see a list of scale modification functions.

- **Common Scale Arguments:**
- **name**: the first argument gives the axis or legend title
- **limits**: the minimum and maximum of the scale
- **breaks**: the points along the scale where labels should appear
- **labels**: the labels that appear at each break

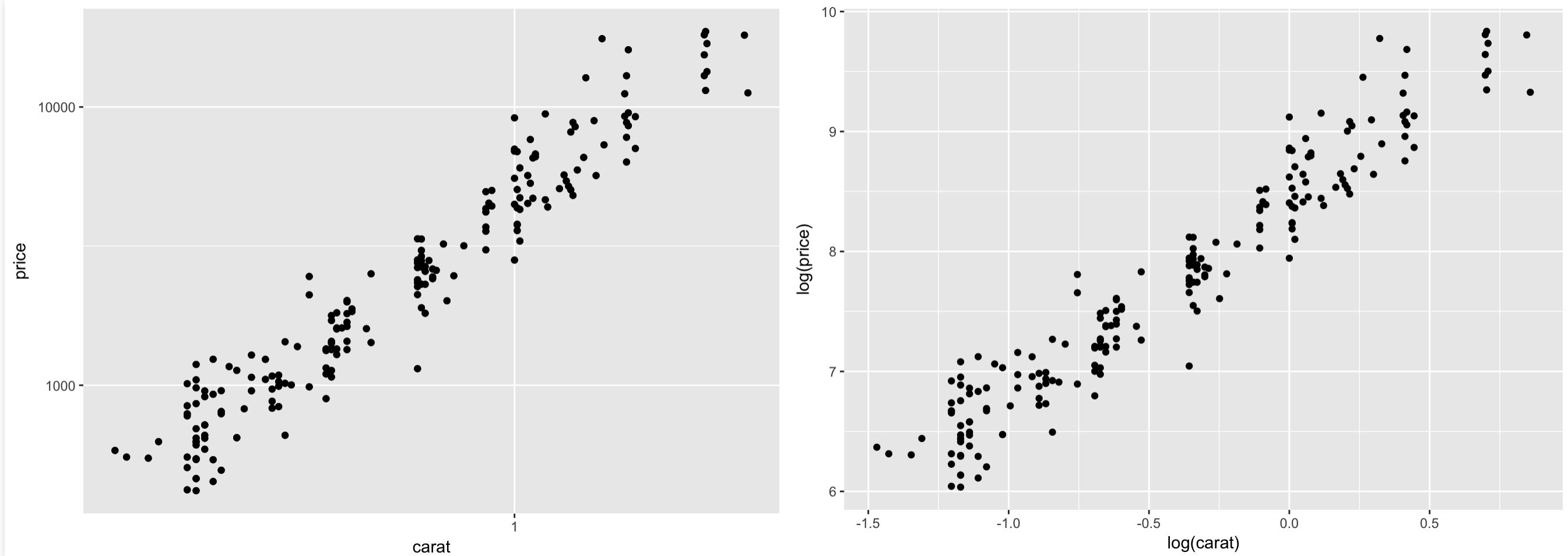
Scale for axes

```
# Square root y-axis transformation
p1 <- ggplot(dsmall, aes(x = carat, y = price)) + geom_point()
psqrt <- p1 + scale_y_sqrt()
# Log base 10 y-axis transformation
plog10 <- p1 + geom_point() + scale_y_log10()
grid.arrange(psqrt, plog10, ncol = 2)
```



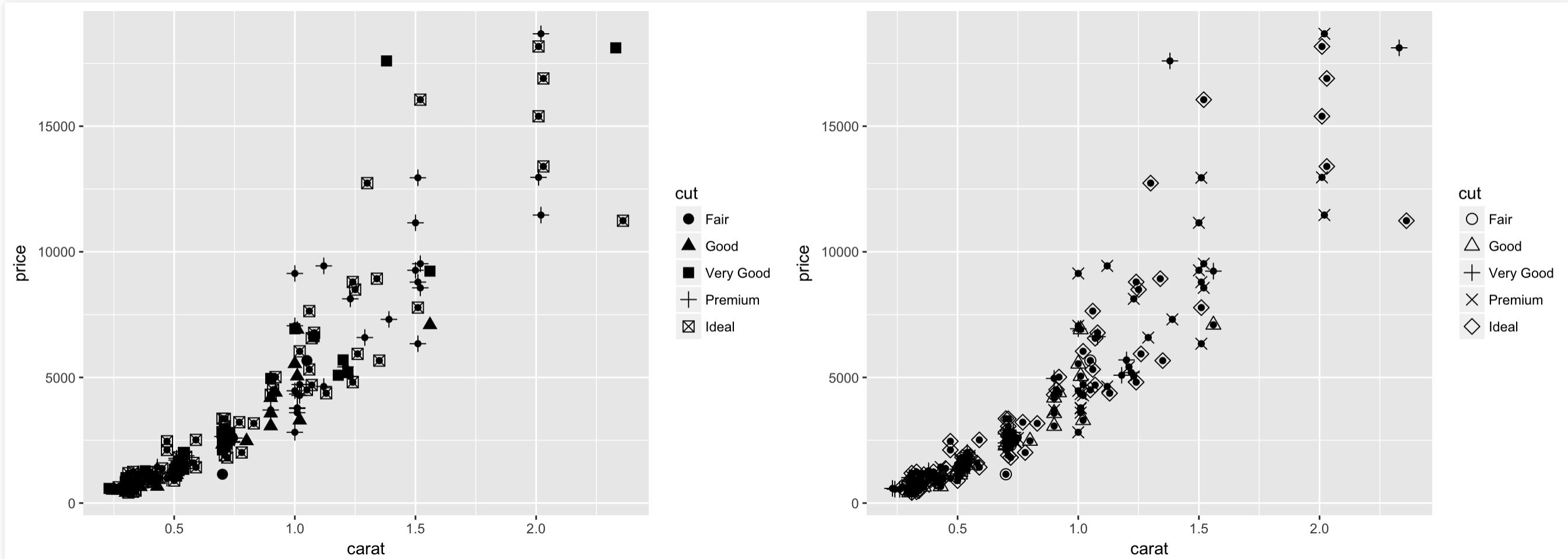
Log base 10 transformation of x and y axes. Note the differences.

```
ploglog1 <- p1 + geom_point() + scale_y_log10() + scale_x_log10()  
ploglog2 <- ggplot(dsmall, aes(x = log(carat), y = log(price))) + geom_point()  
grid.arrange(ploglog1, ploglog2, ncol = 2)
```



Scale for shapes

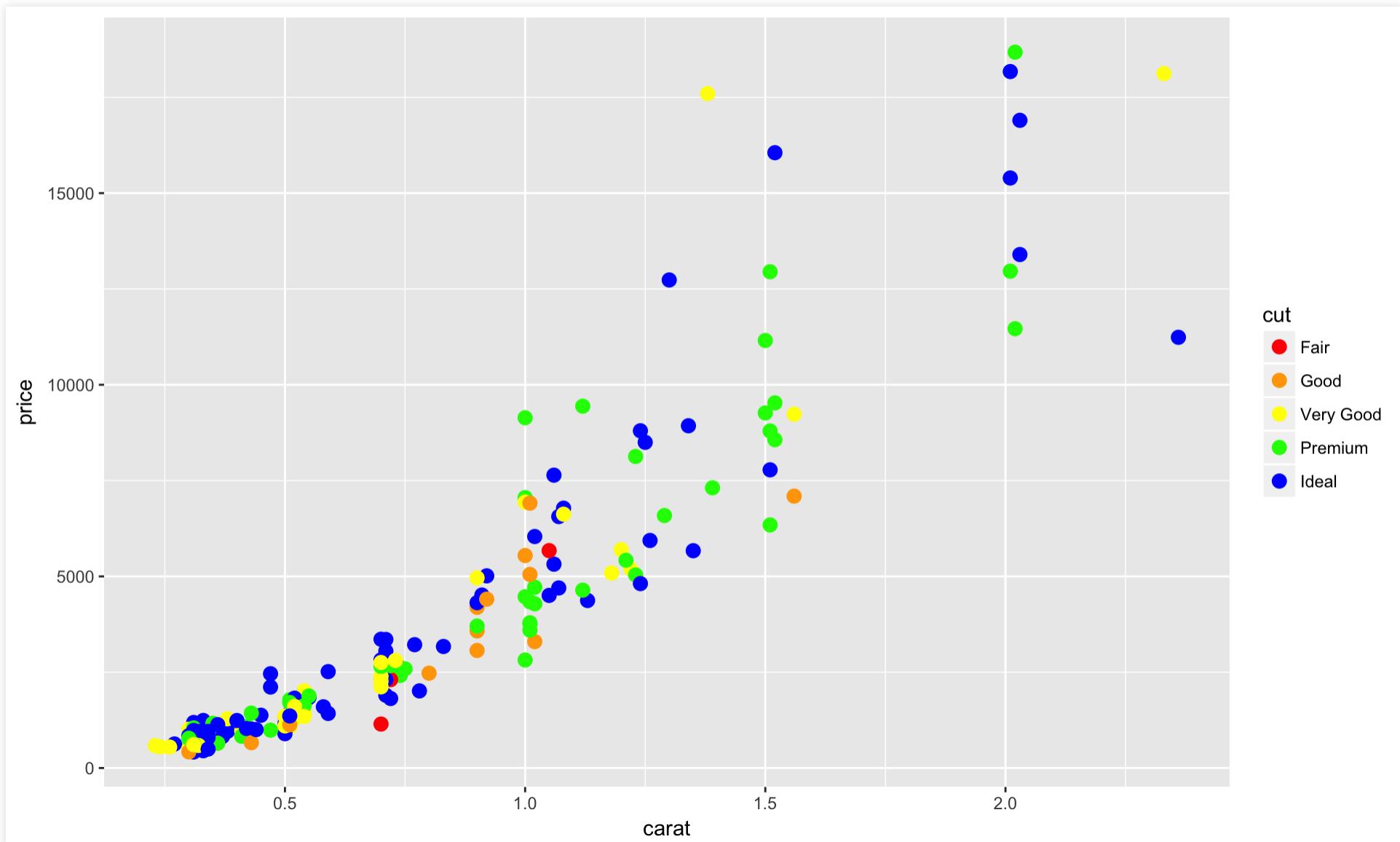
```
p11 <- p1 + geom_point(aes(shape = cut), size = 3)
p12 <- p1 + geom_point(aes(shape = cut), size = 3) + scale_shape_manual(values = c(1:5))
grid.arrange(p11, p12, ncol = 2)
```



Scale for colors

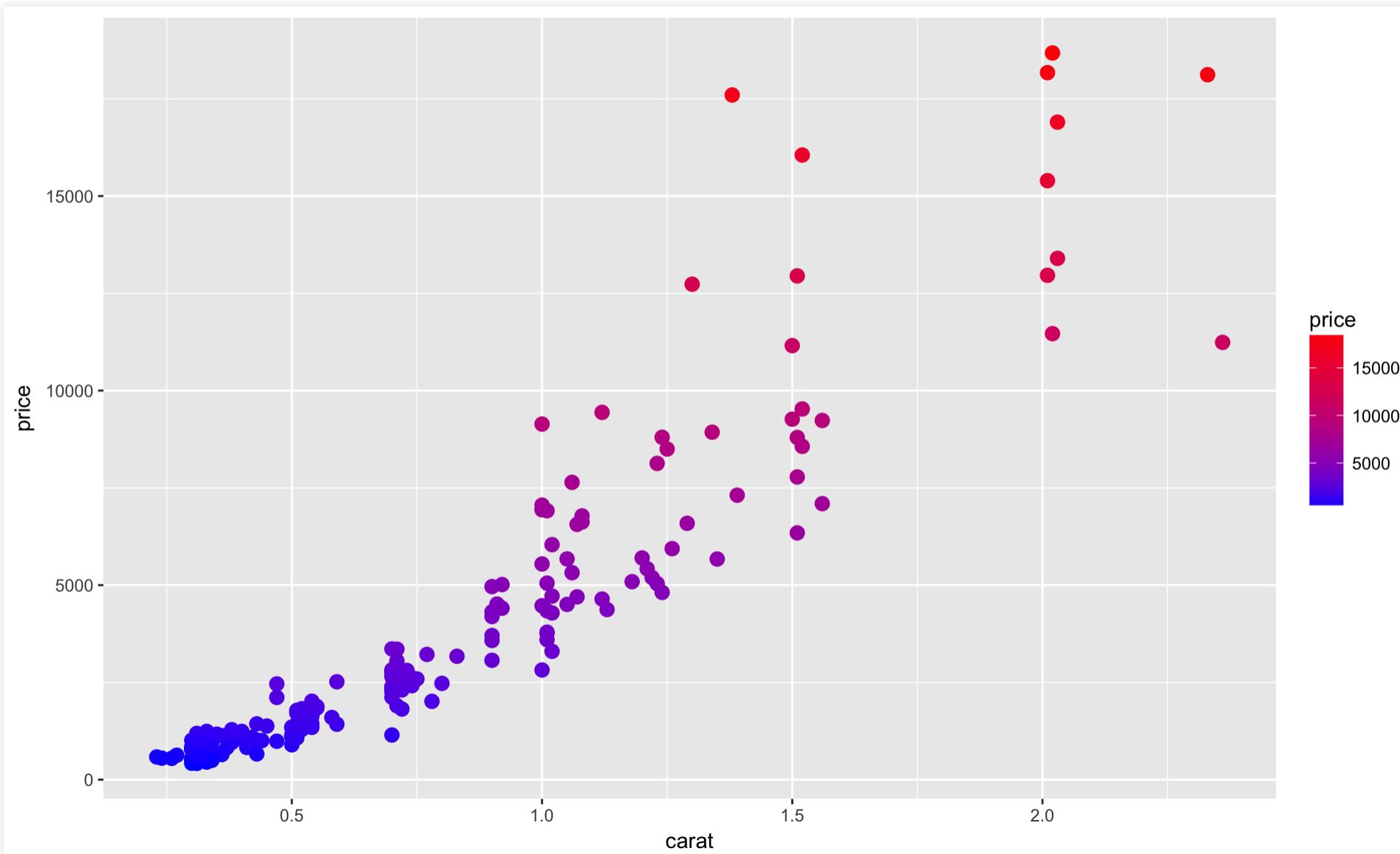
To choose specific colors for **discrete** variables we use `scale_color_manual`:

```
p1 + geom_point(aes(color = cut), size = 3) +  
  scale_color_manual(values = c("red", "orange", "yellow", "green", "blue"))
```



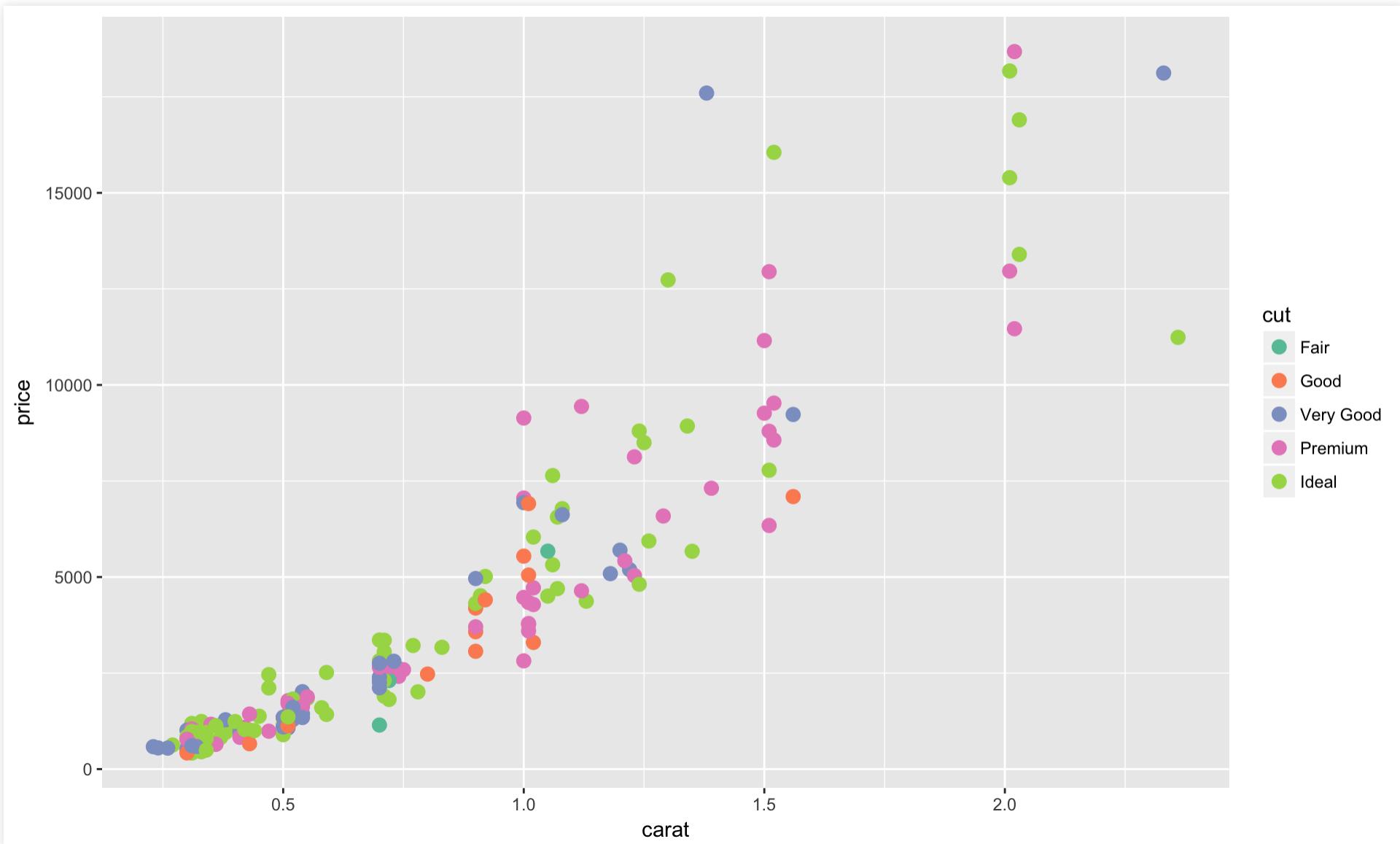
For **continuous** variables we use `scale_color_gradient`, and specify the ends of the color spectrum.

```
p1 + geom_point(aes(color = price), size = 3) +  
  scale_color_gradient(low = "blue", high = "red")
```



`scale_color_brewer` lets you choose nice color palettes for discrete variables

```
p1 + geom_point(aes(color = cut), size = 3) +  
  scale_color_brewer(palette = "Set2")
```

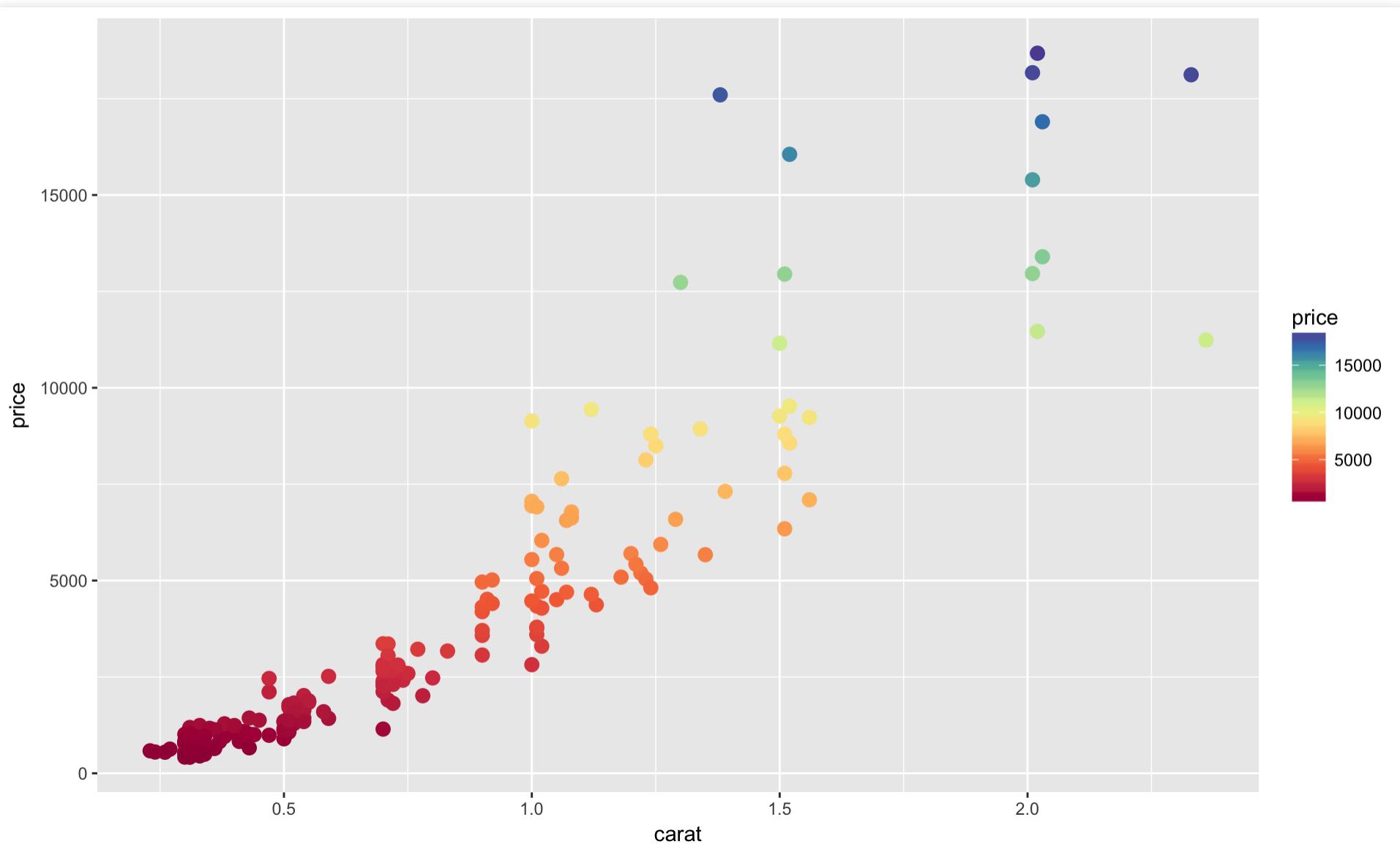


Unfortunately, `scale_color_brewer` doesn't work for continuous variables

```
# scale_color_brewer() does not work with continuous variables
# and will result in an error
p1 + geom_point(aes(shape = price), size = 3) +
  scale_color_brewer(palette = "Spectral")
# Error: A continuous variable can not be mapped to shape
```

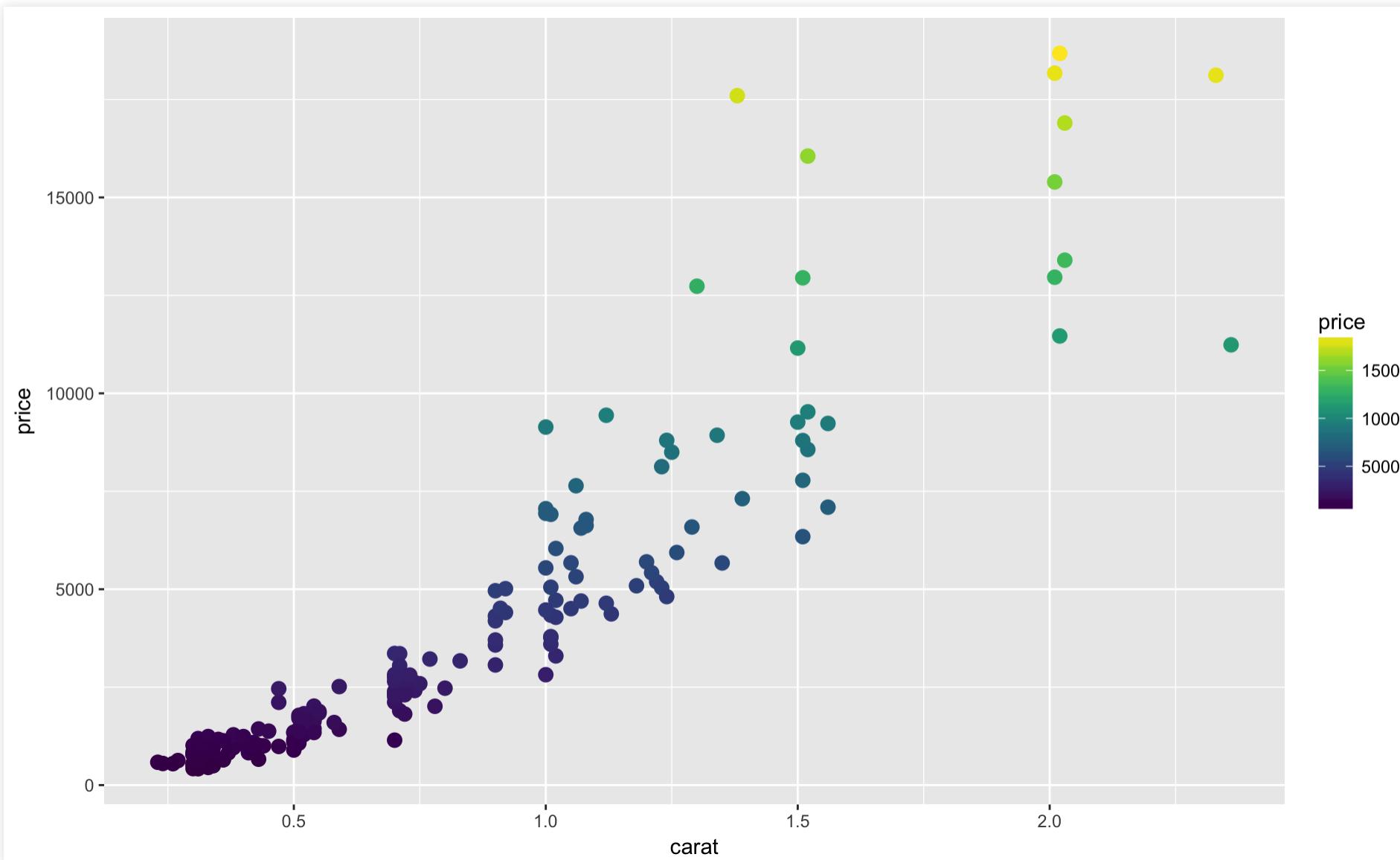
We can get around this issue using the `RColorBrewer` package and `scale_color_gradientn` function, which **interpolates colors** from the brew palettes.

```
# install.packages("RColorBrewer")
library(RColorBrewer)
p1 + geom_point(aes(color = price), size = 3) +
  scale_color_gradientn(colours = brewer.pal(name = "Spectral", n = 10))
```

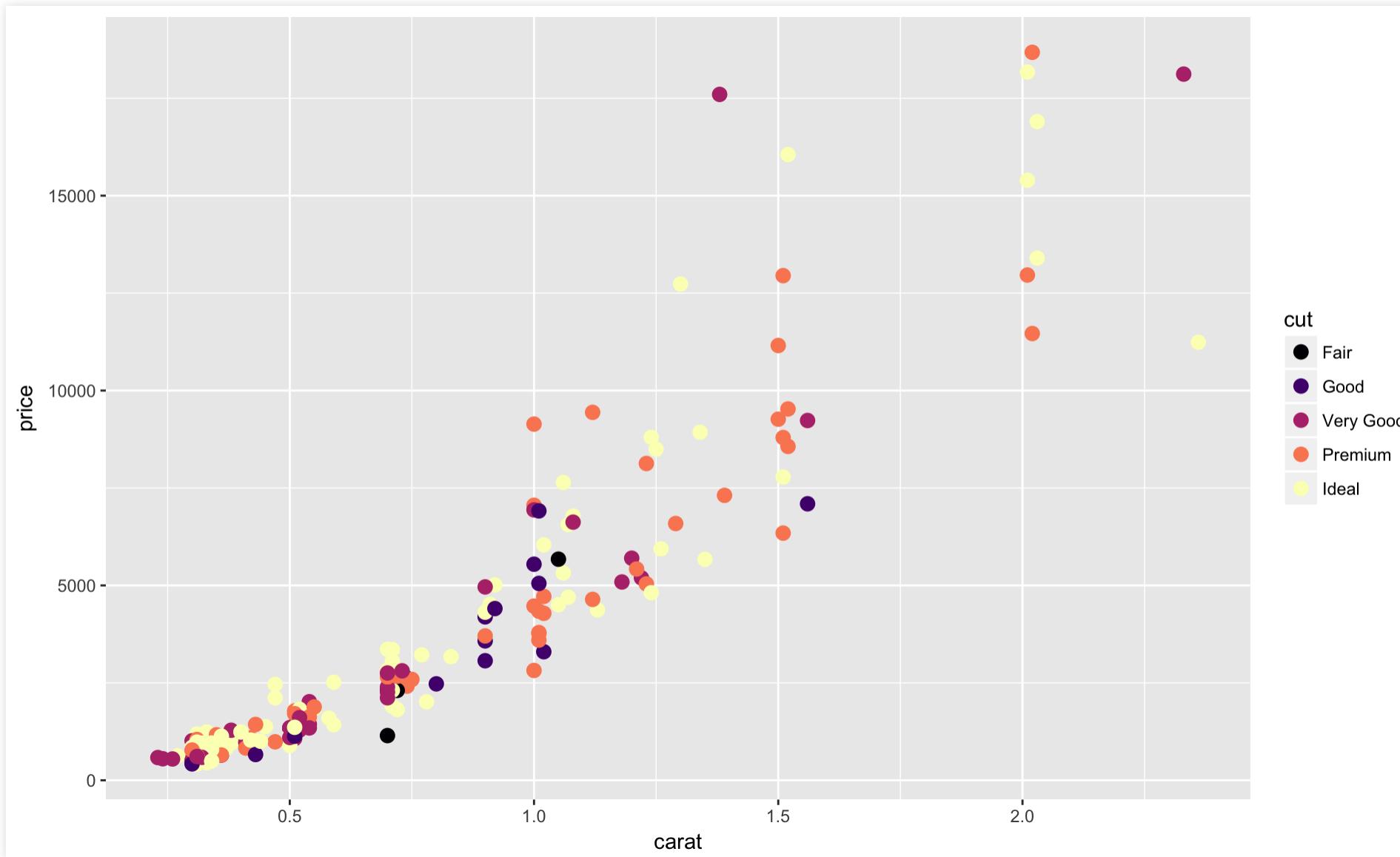


Another popular color scheme package, `viridis`, supports both discrete and continuous variables:

```
# install.packages("viridis")
library(viridis)
p1 + geom_point(aes(color = price), size = 3) + scale_color_viridis()
```



```
p1 + geom_point(aes(color = cut), size = 3) +  
  scale_color_viridis(discrete = TRUE, option = "magma")
```



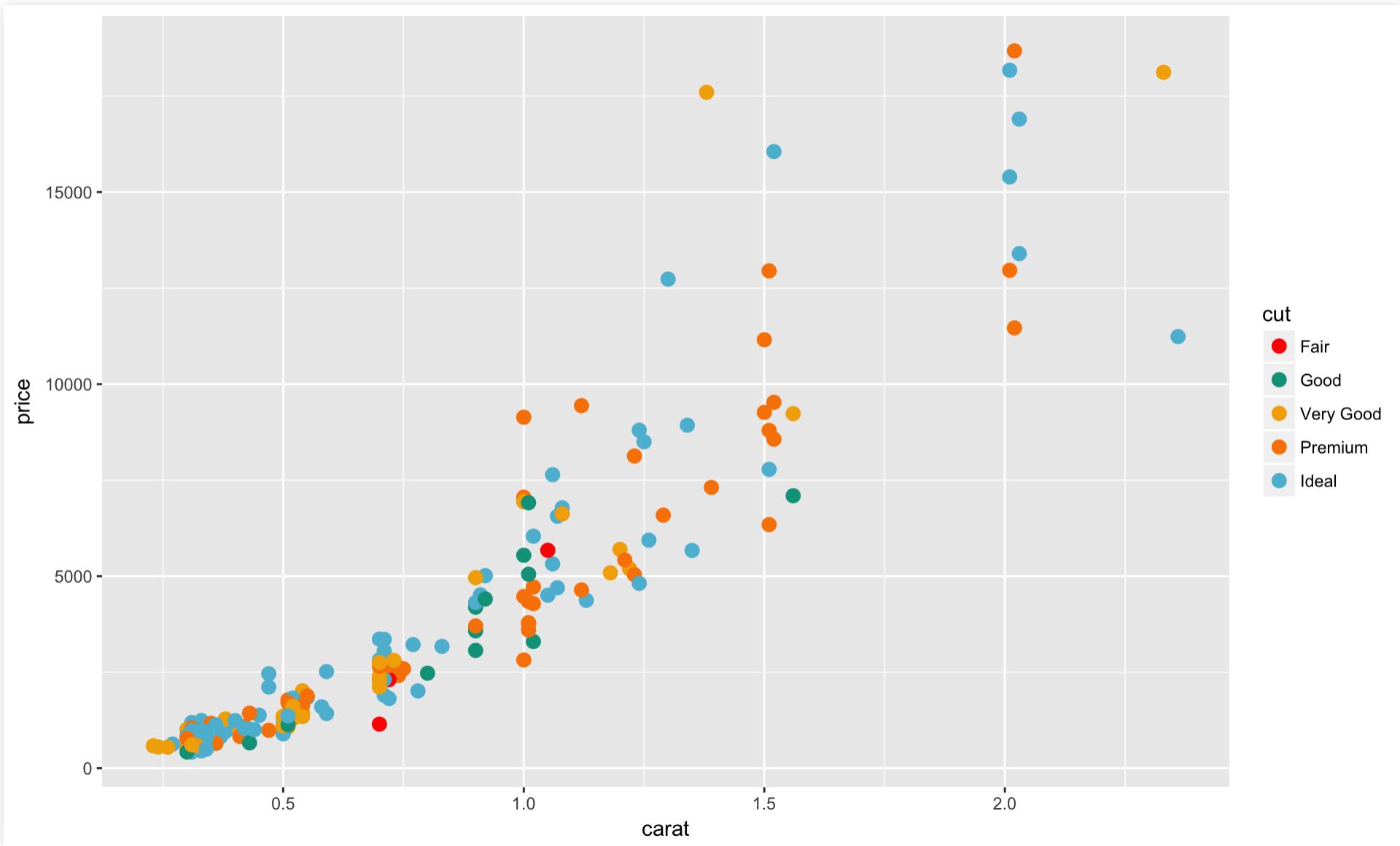
... there are also other unconventional schemes such as, one based on Wes Anderson movies :

```
#install.packages("wesanderson")
library(wesanderson)
names(wes_palettes)

## [1] "GrandBudapest" "Moonrise1" "Royal1" "Moonrise2"
## [5] "Cavalcanti"    "Royal2"    "GrandBudapest2" "Moonrise3"
## [9] "Chevalier"     "Zissou"    "FantasticFox"  "Darjeeling"
## [13] "Rushmore"      "BottleRocket" "Darjeeling2"
```

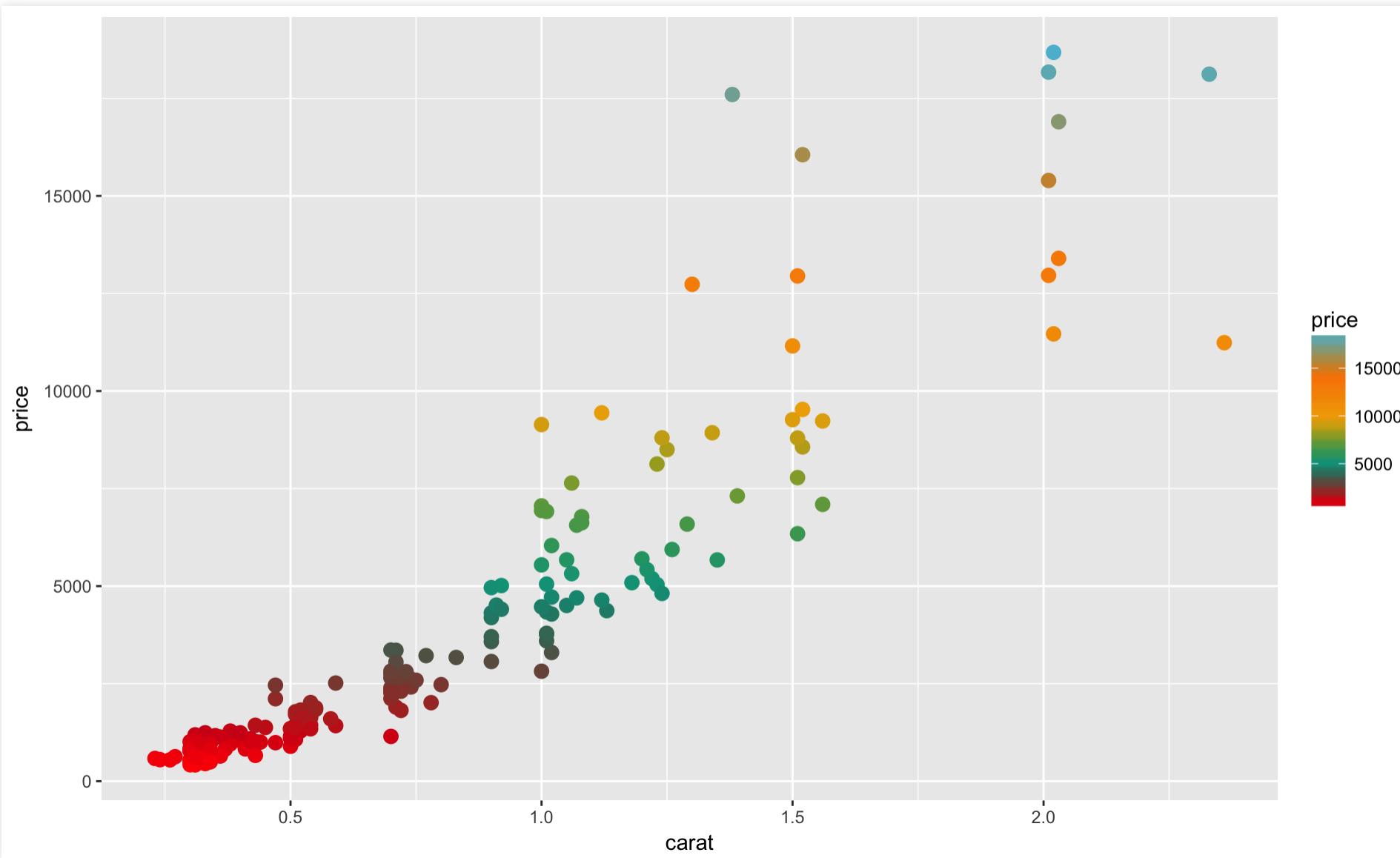
Wes Anderson color palette:

```
# For discrete variables  
p1 + geom_point(aes(color = cut), size = 3) +  
  scale_color_manual(values = wes_palette("Darjeeling", n = 5))
```



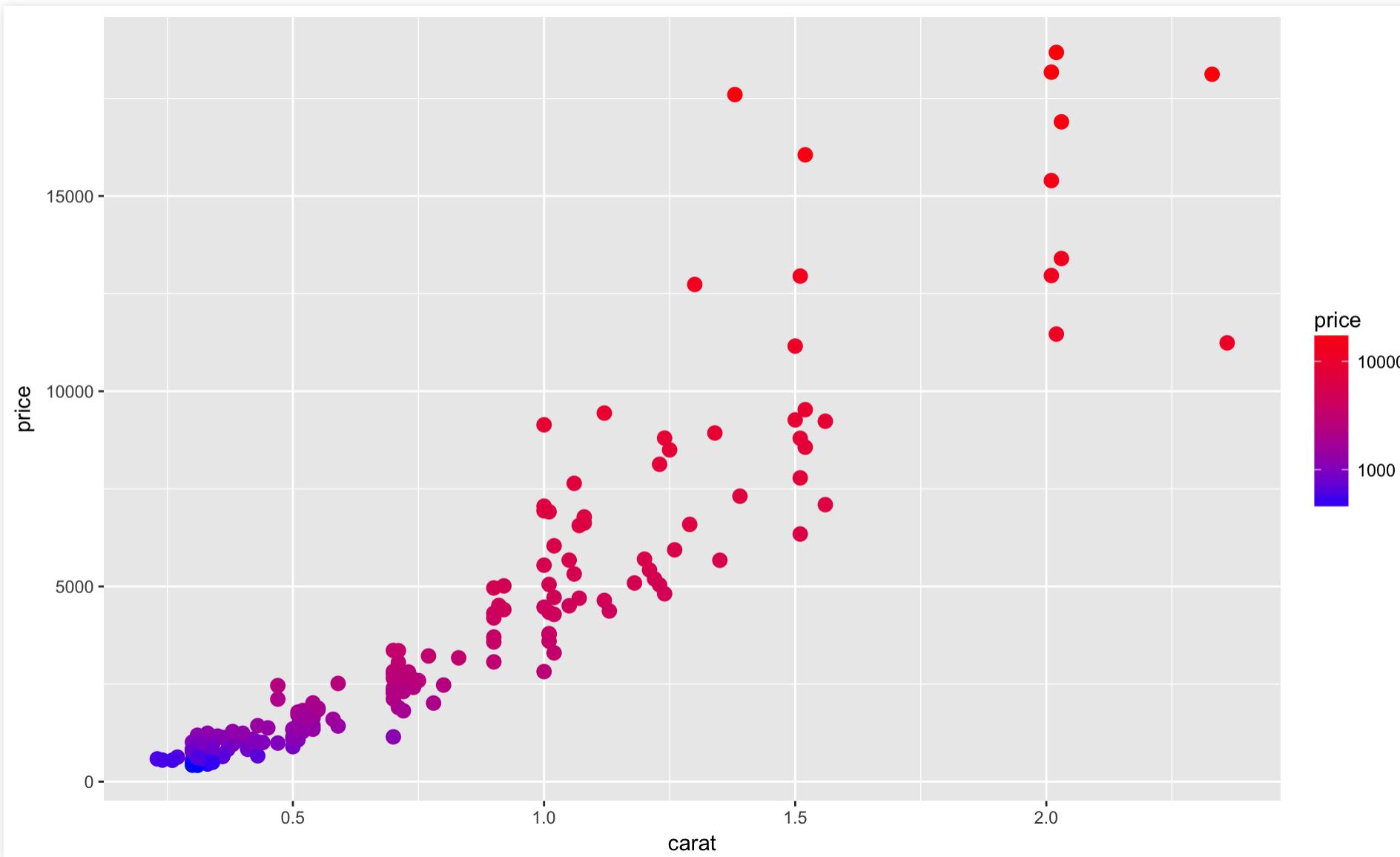
```
# For continuous variables:
```

```
p1 + geom_point(aes(color = price), size = 3) +  
  scale_color_gradientn(colours = wes_palette("Darjeeling", 100, type = "continuous"))
```



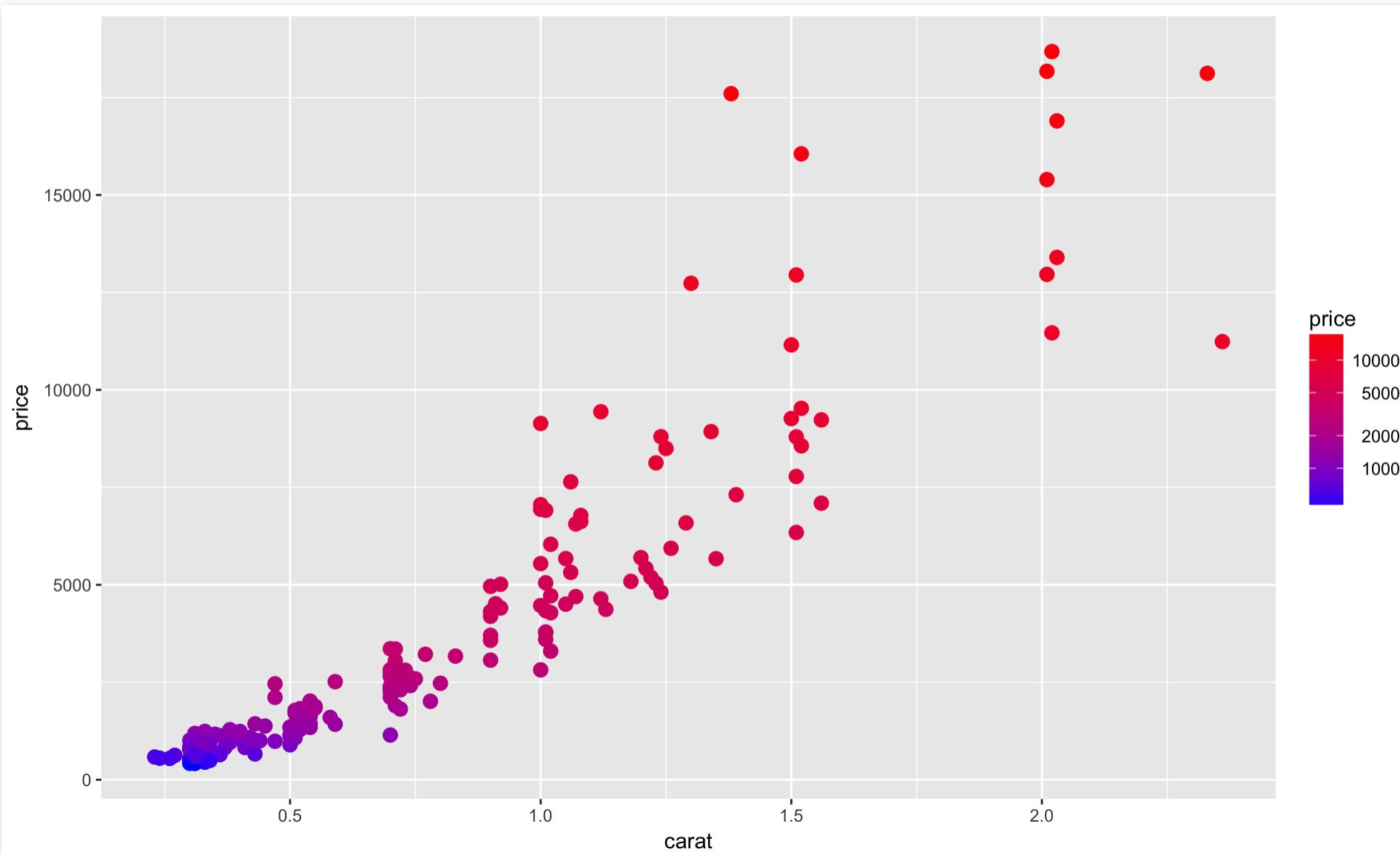
You can also scale the values of the variable corresponding to color.

```
p1 + geom_point(aes(color = price), size = 3) +  
  scale_color_gradient(low = "blue", high = "red", trans = "log10")
```



Or and add your own breaks

```
p1 + geom_point(aes(color = price), size = 3) +  
  scale_color_gradient(low = "blue", high = "red", trans = "log10",  
  breaks = c(1000, 2000, 5000, 10000),  
  labels = c(" 1000", " 2000", " 5000", "10000"))
```



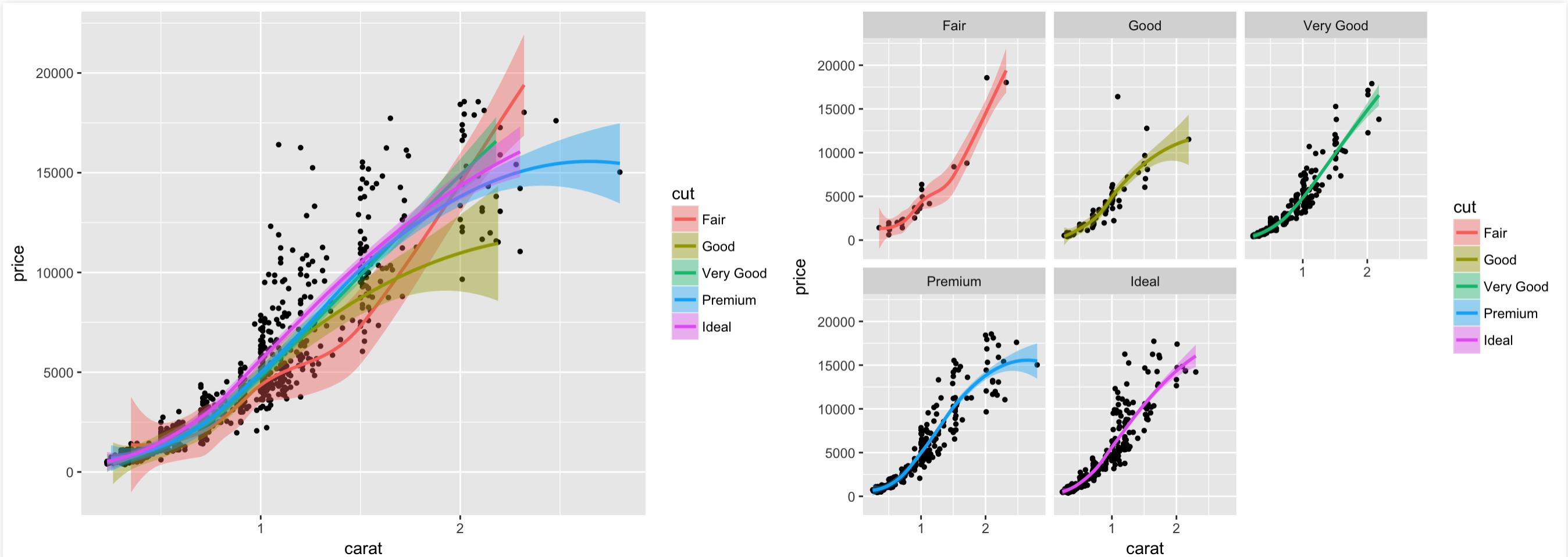
Exercise 3

- Go to back “Lec4_Exercises.Rmd”
- Complete Exercise 3

Faceting

Facetting allows you to split up your data by one or more variables and plot the subsets of data together.

```
dsmall <- diamonds[sample(nrow(diamonds), 1000), ]  
p0 <- ggplot(data = dsmall, aes(x = carat, y = price)) +geom_point(size = 1) +  
  geom_smooth(aes(colour = cut, fill = cut))  
p1 <- p0 + facet_wrap(~ cut)  
grid.arrange(p0, p1, ncol = 2)
```



```
ggplot(diamonds, aes(x = carat)) +  
  geom_density() +  
  facet_grid(color ~ cut)
```

