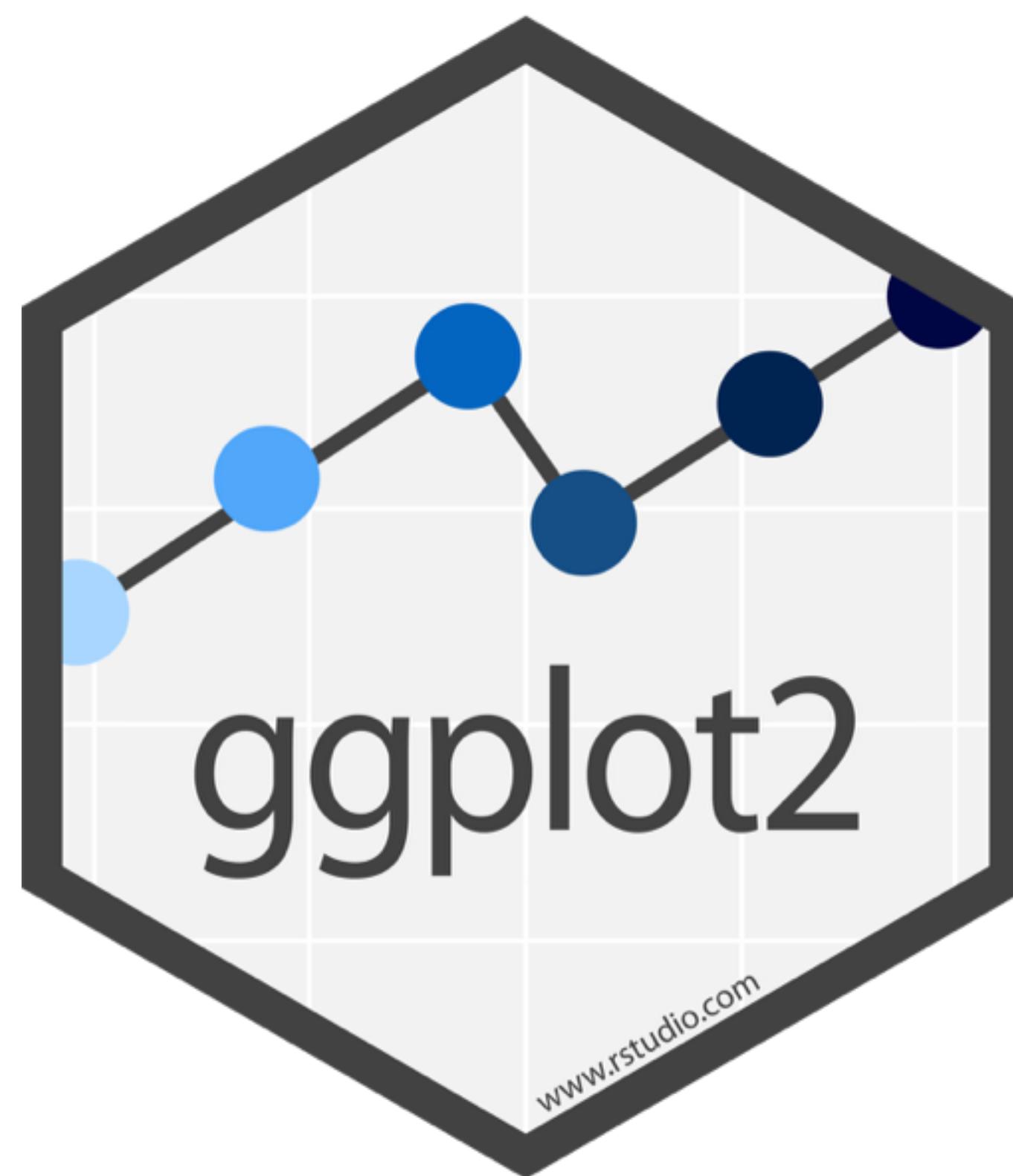
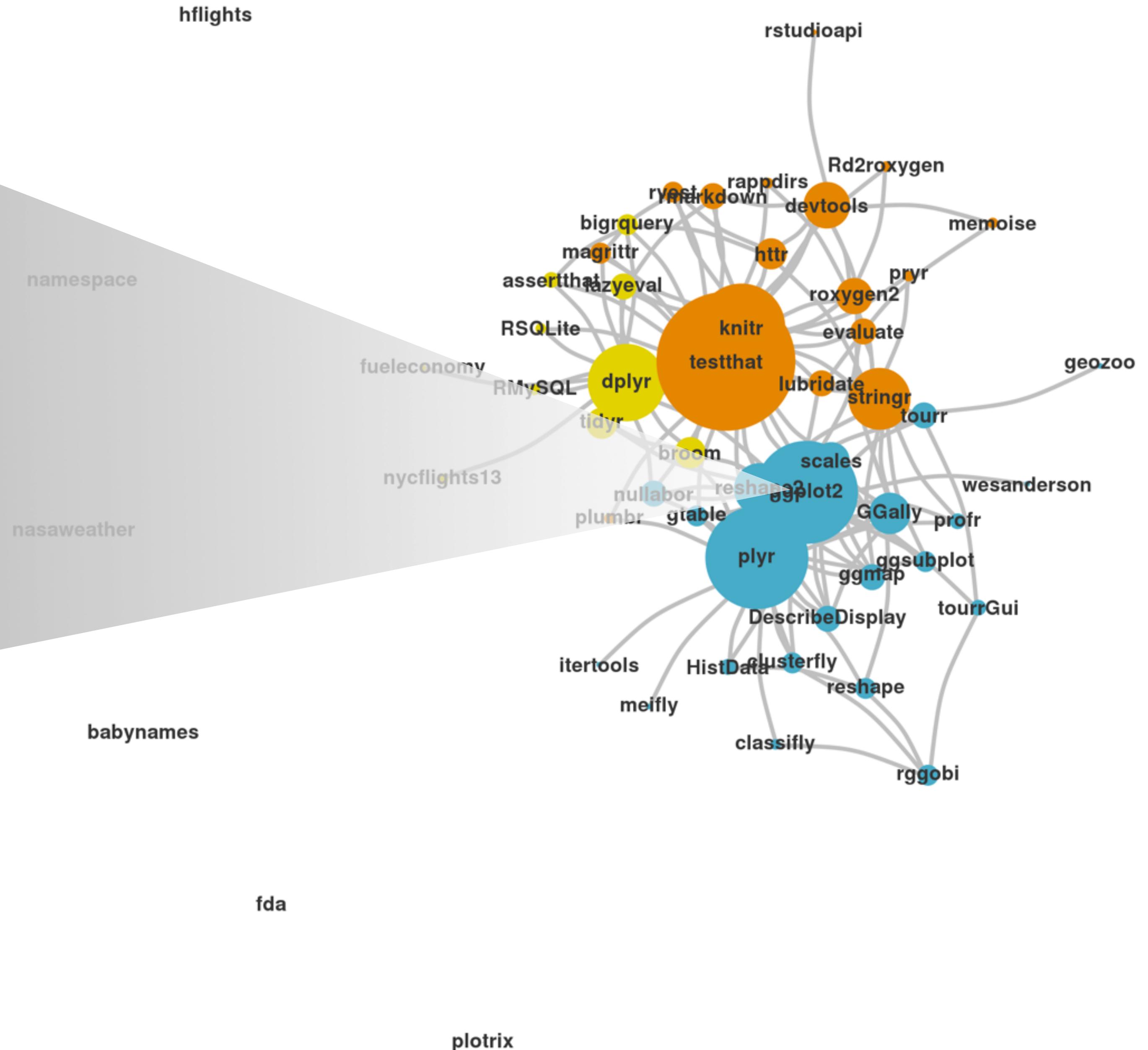


# Data Visualization with

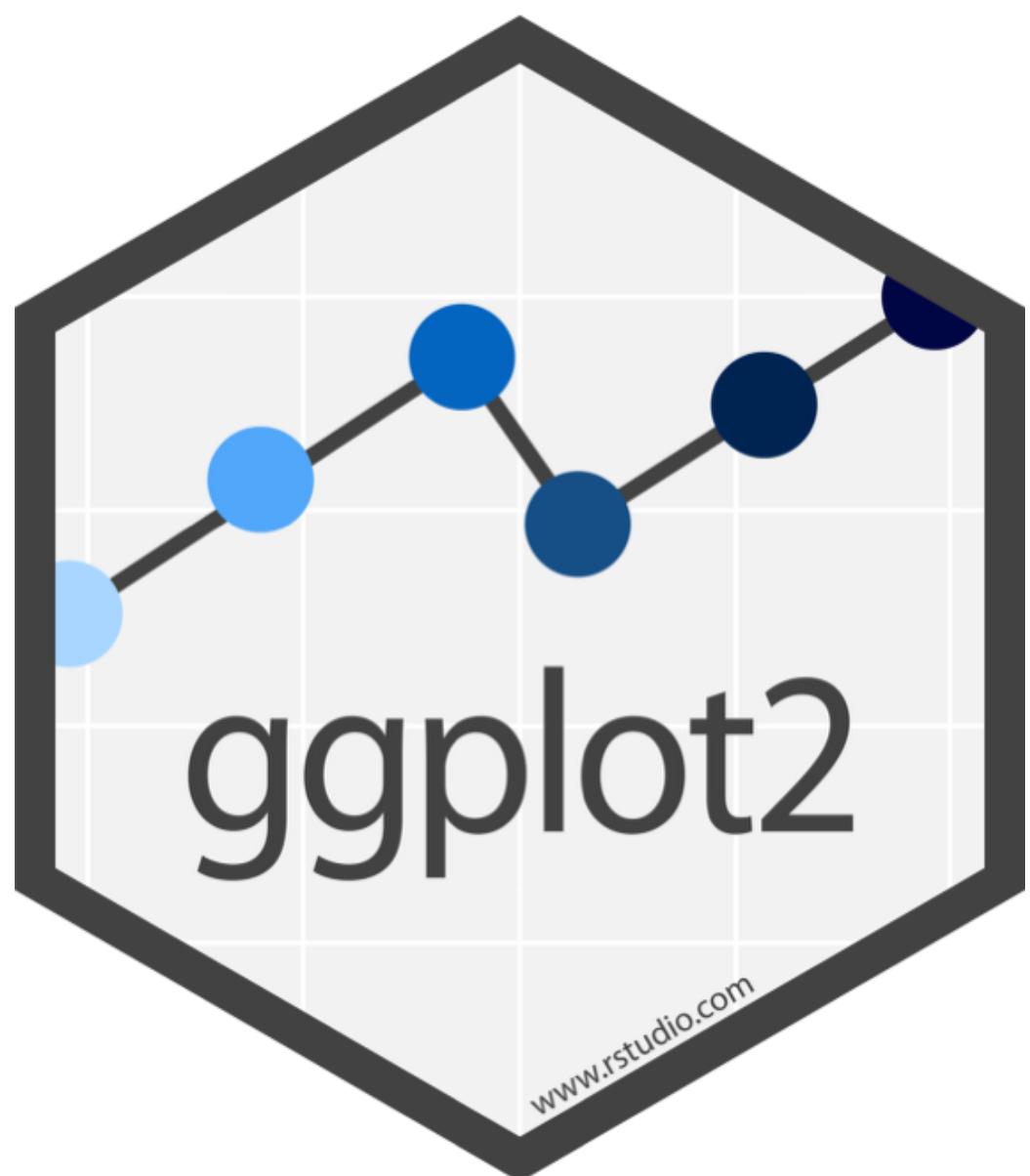


"The simple graph has brought more information to the data analyst's mind than any other device. "

- John Tukey

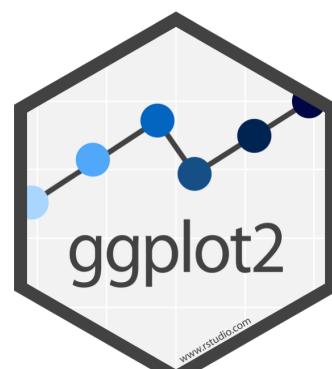


# ggplot2

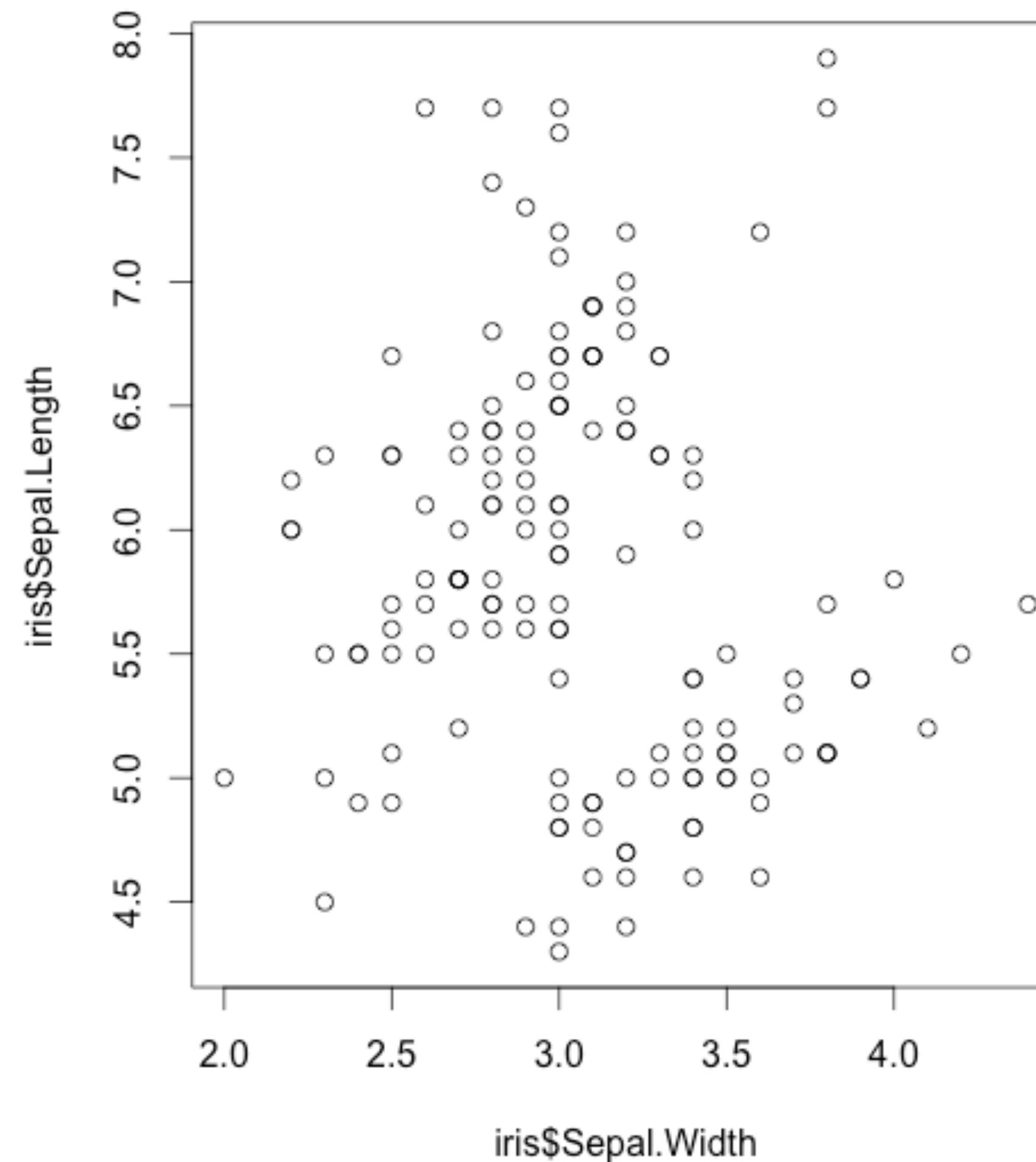


A package that visualizes data.

ggplot2 implements the *grammar of graphics*, a system for building visualizations that is built around **cases** and **variables**.

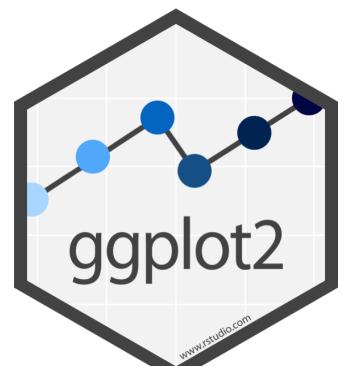


# plot

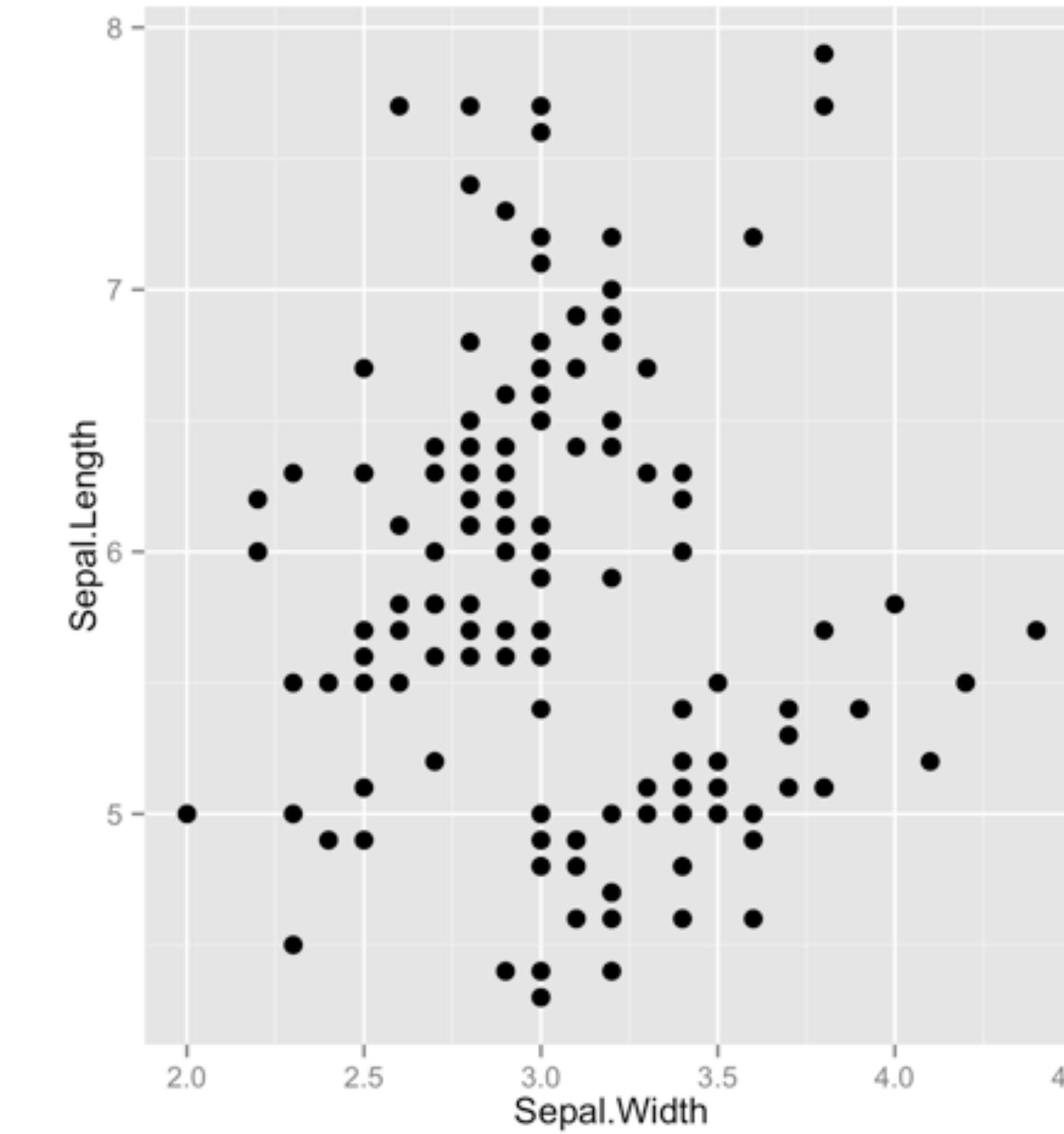
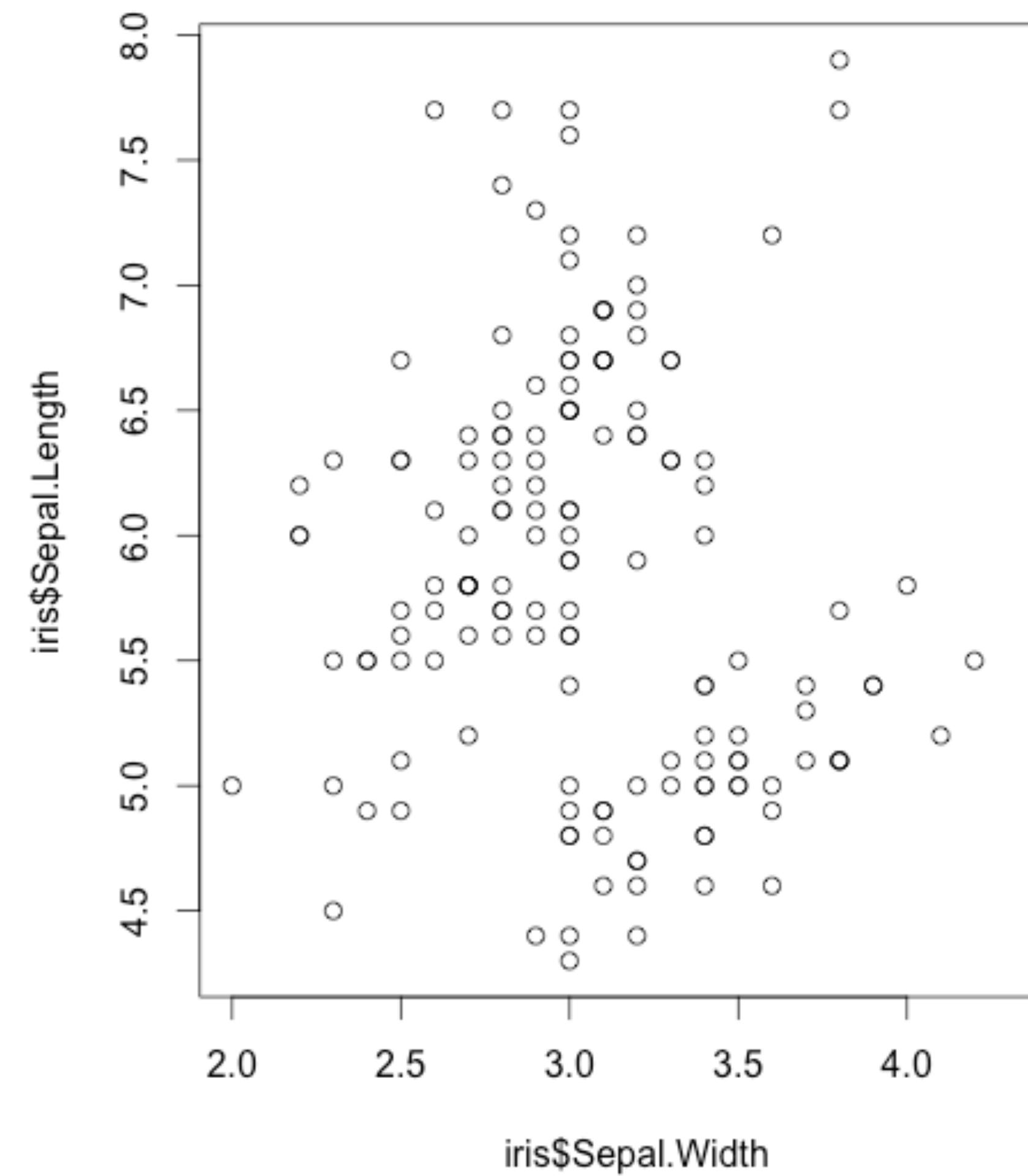


```
plot(iris$Sepal.Length,  
     iris$Sepal.Width)
```

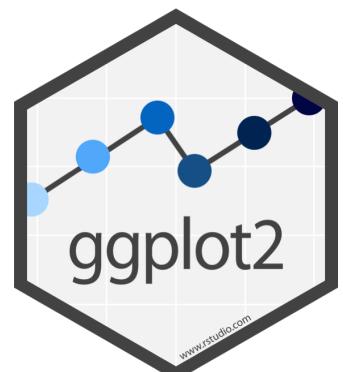
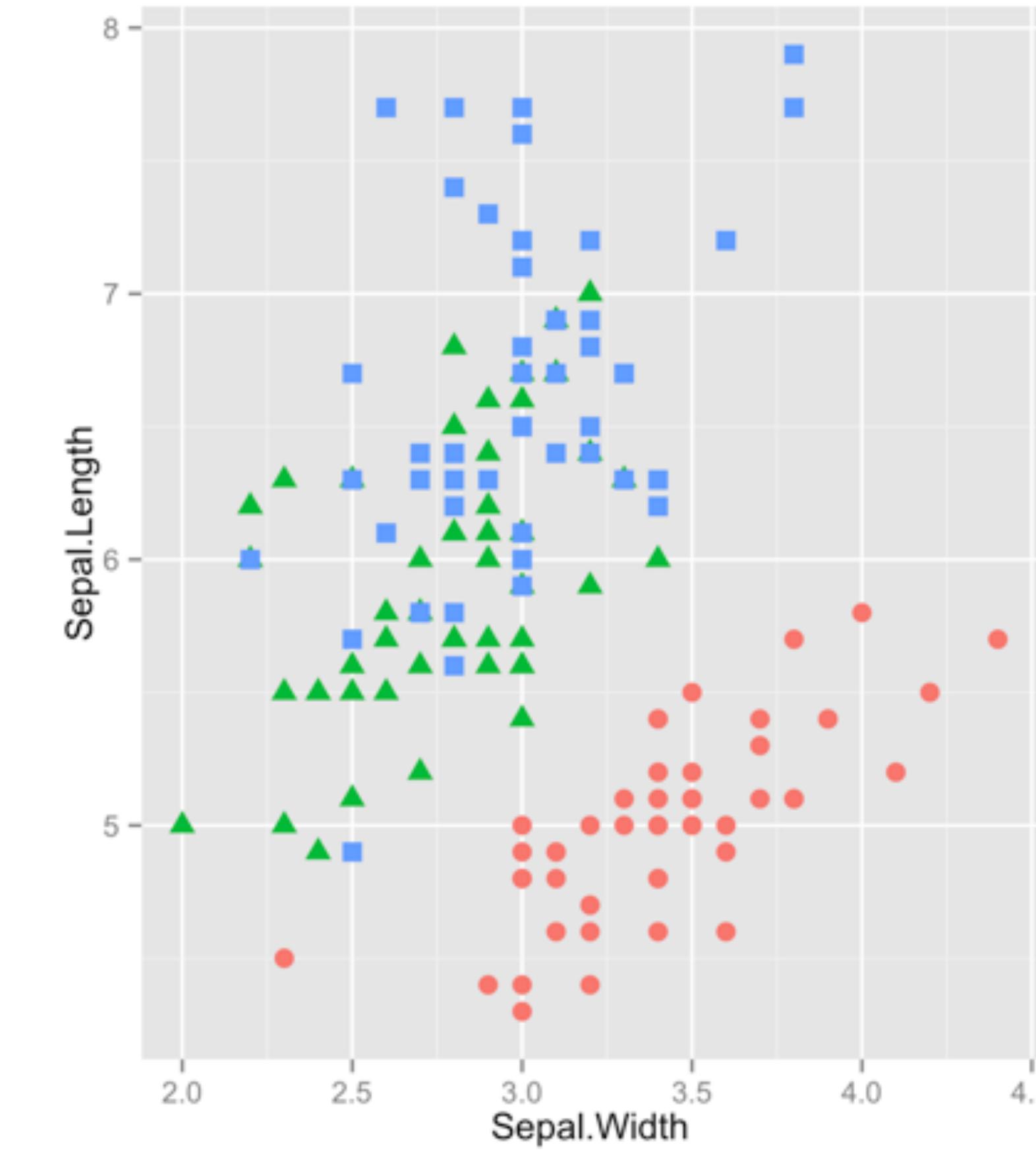
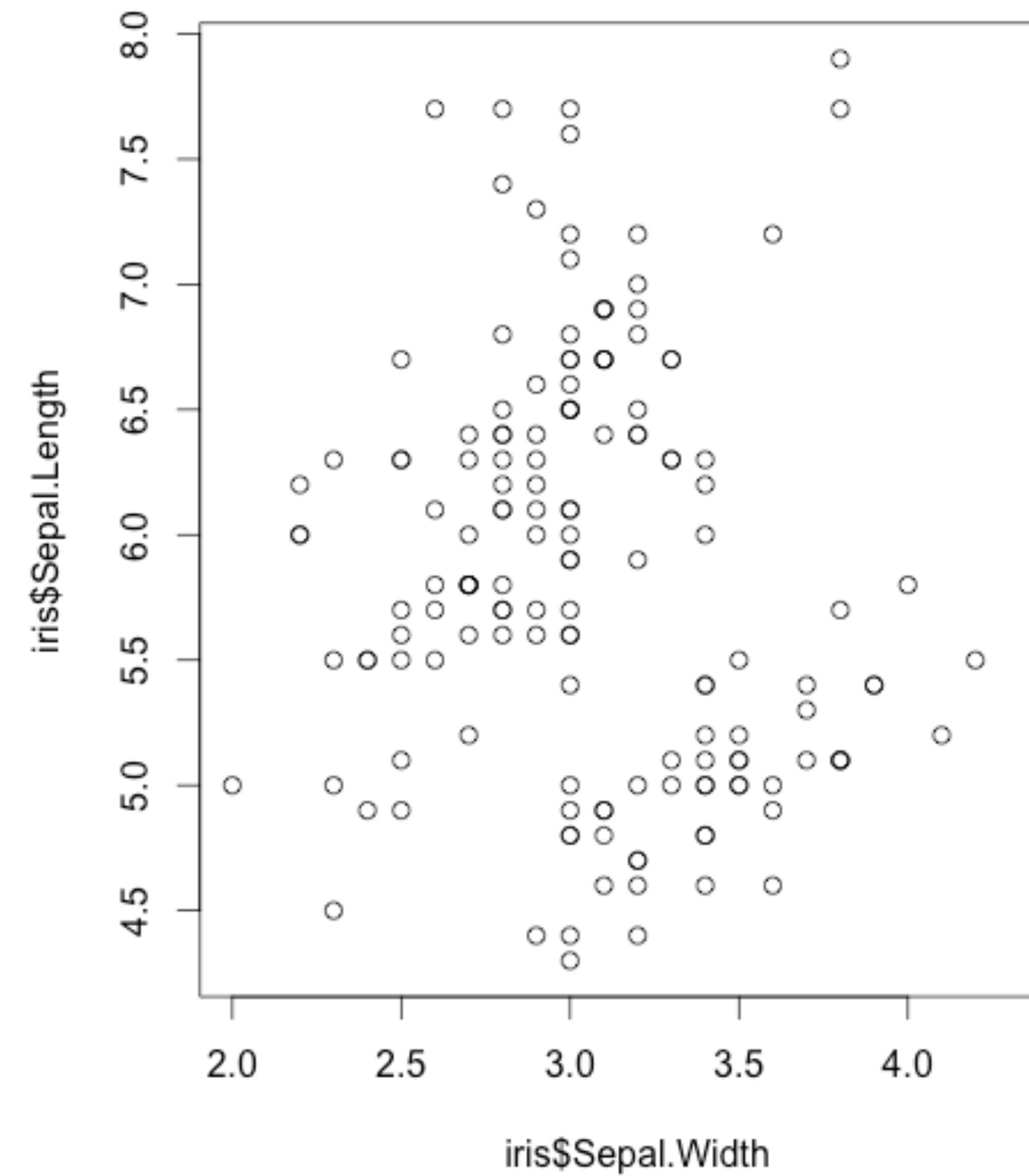
- R's basic plot method
- simple
- does different things in different contexts (usually in a helpful way)
- difficult to customize



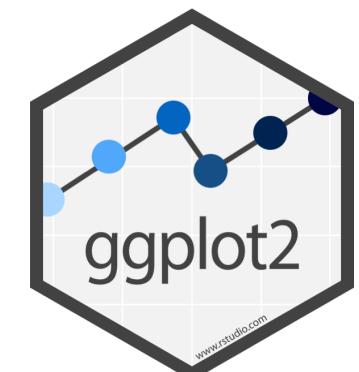
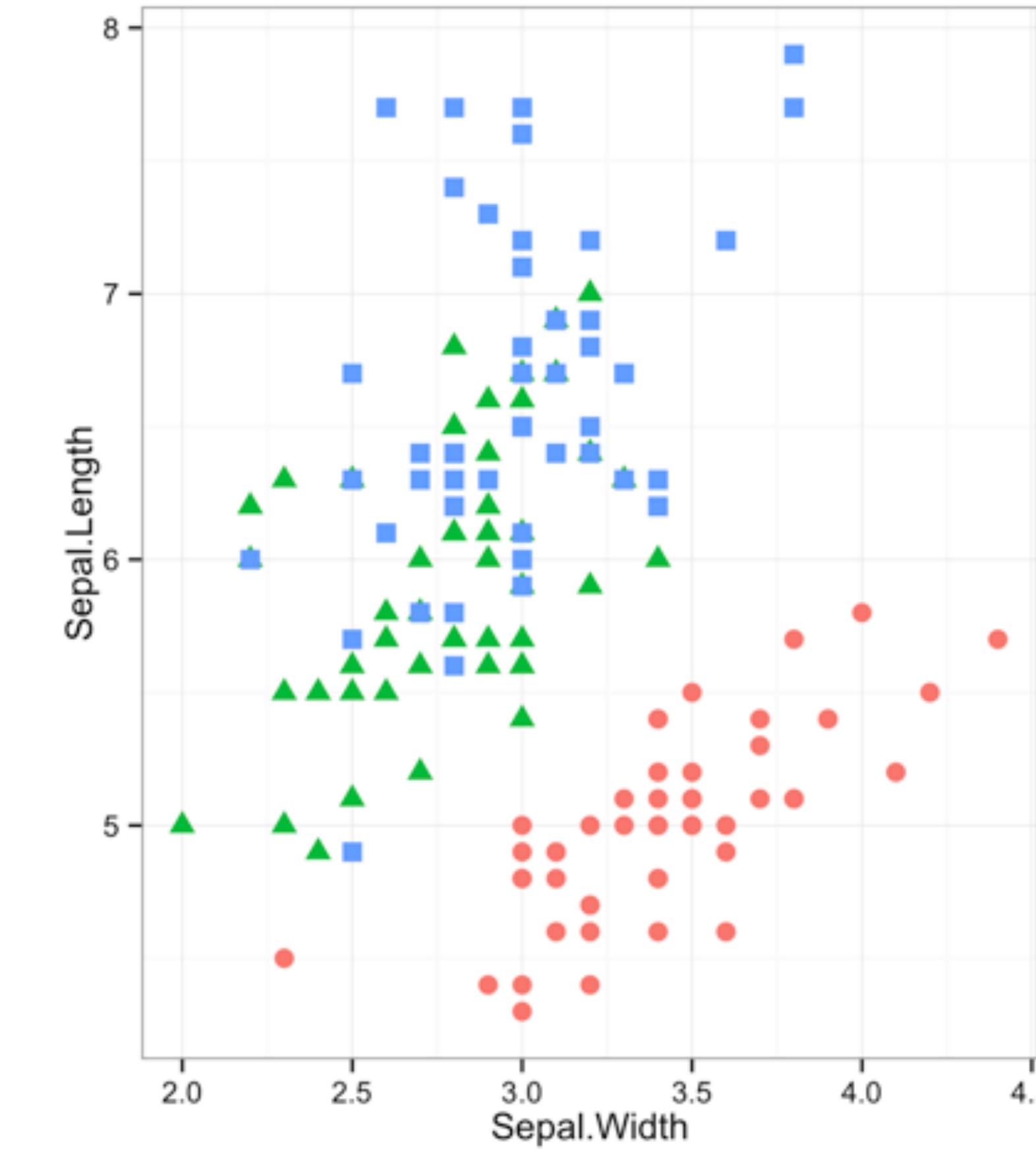
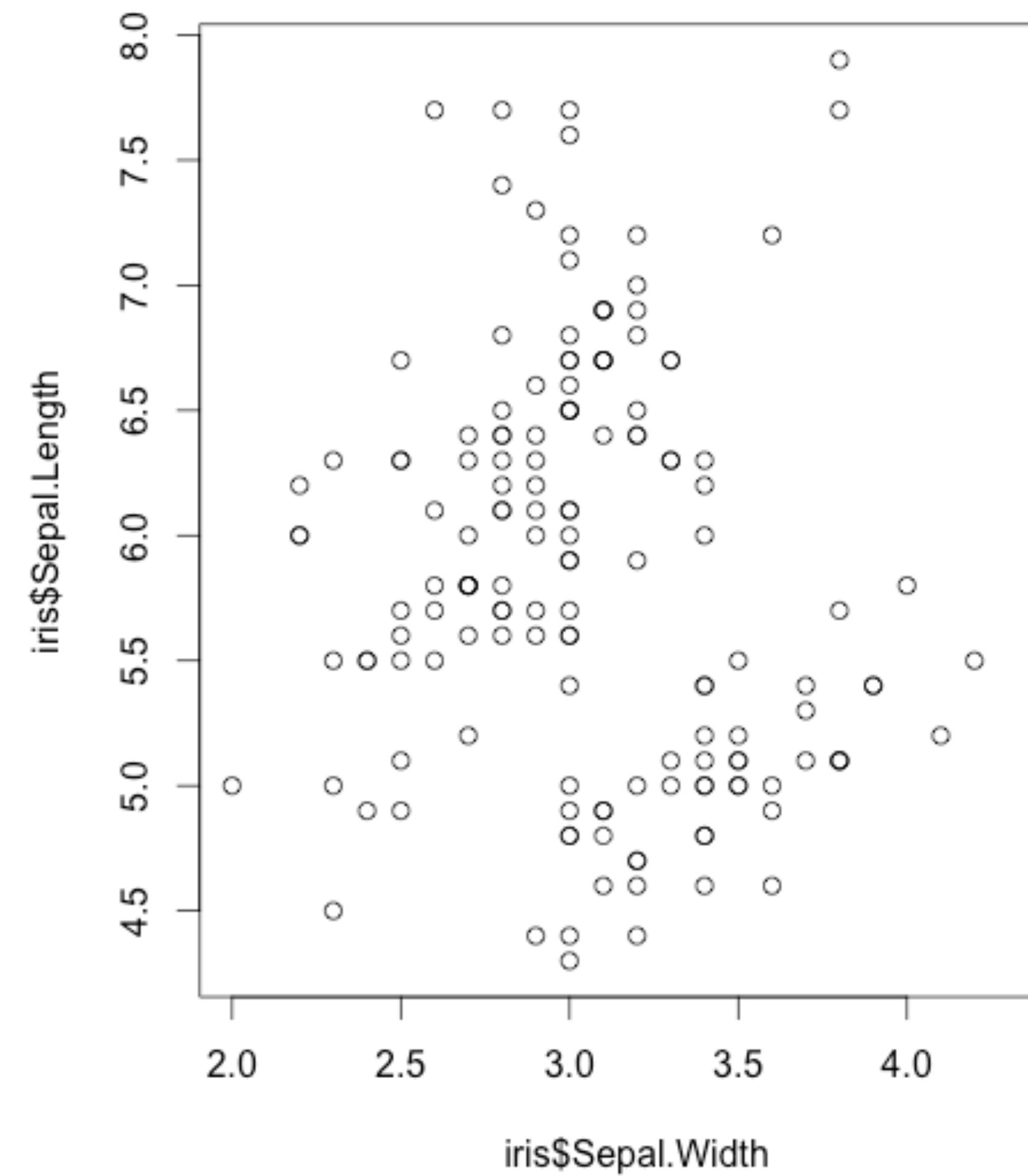
# ggplot2



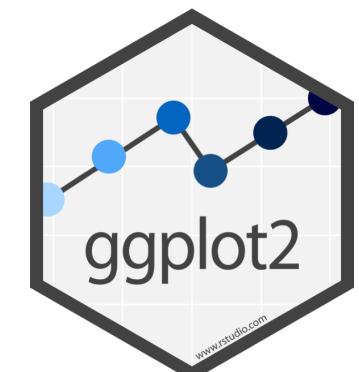
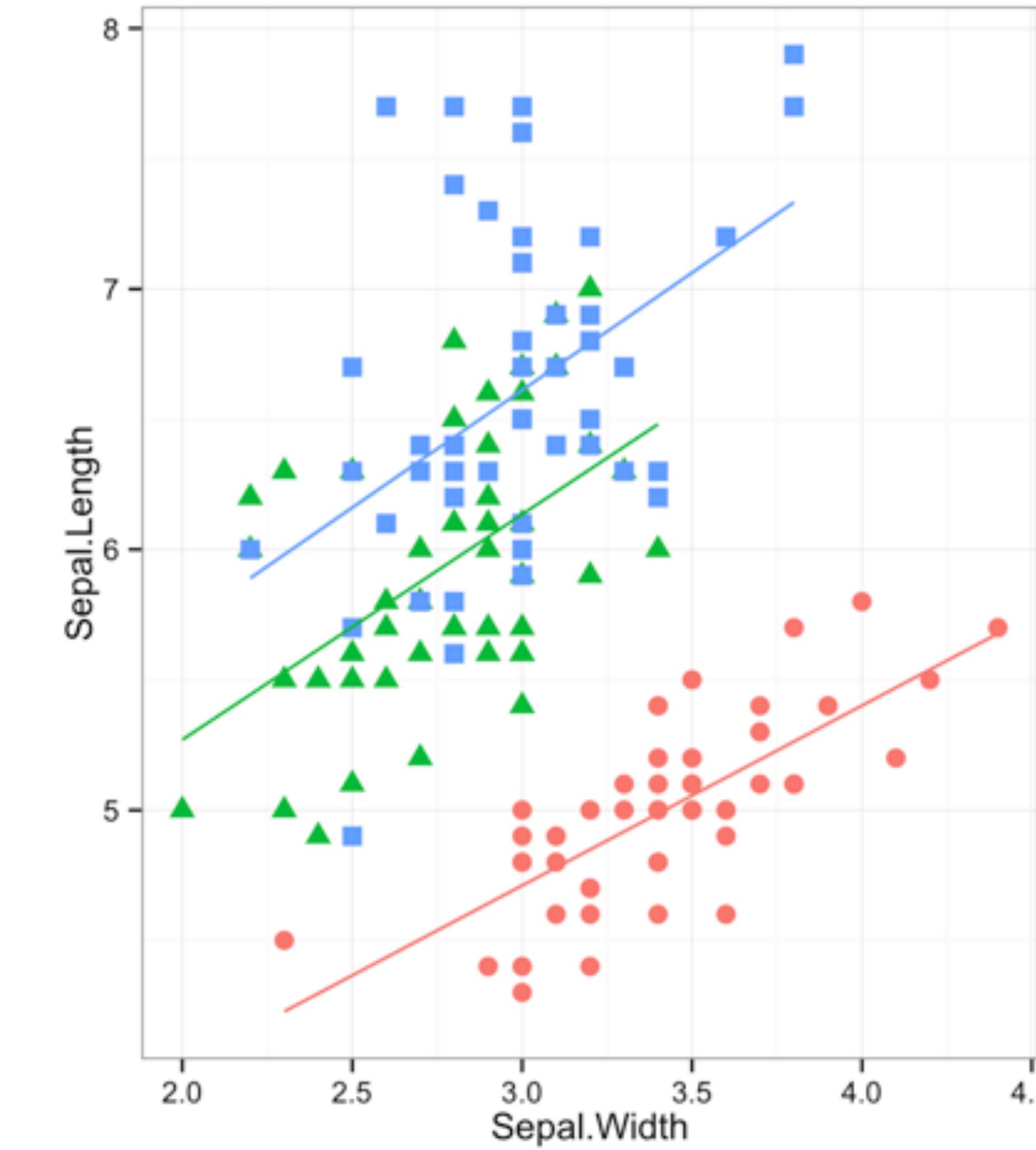
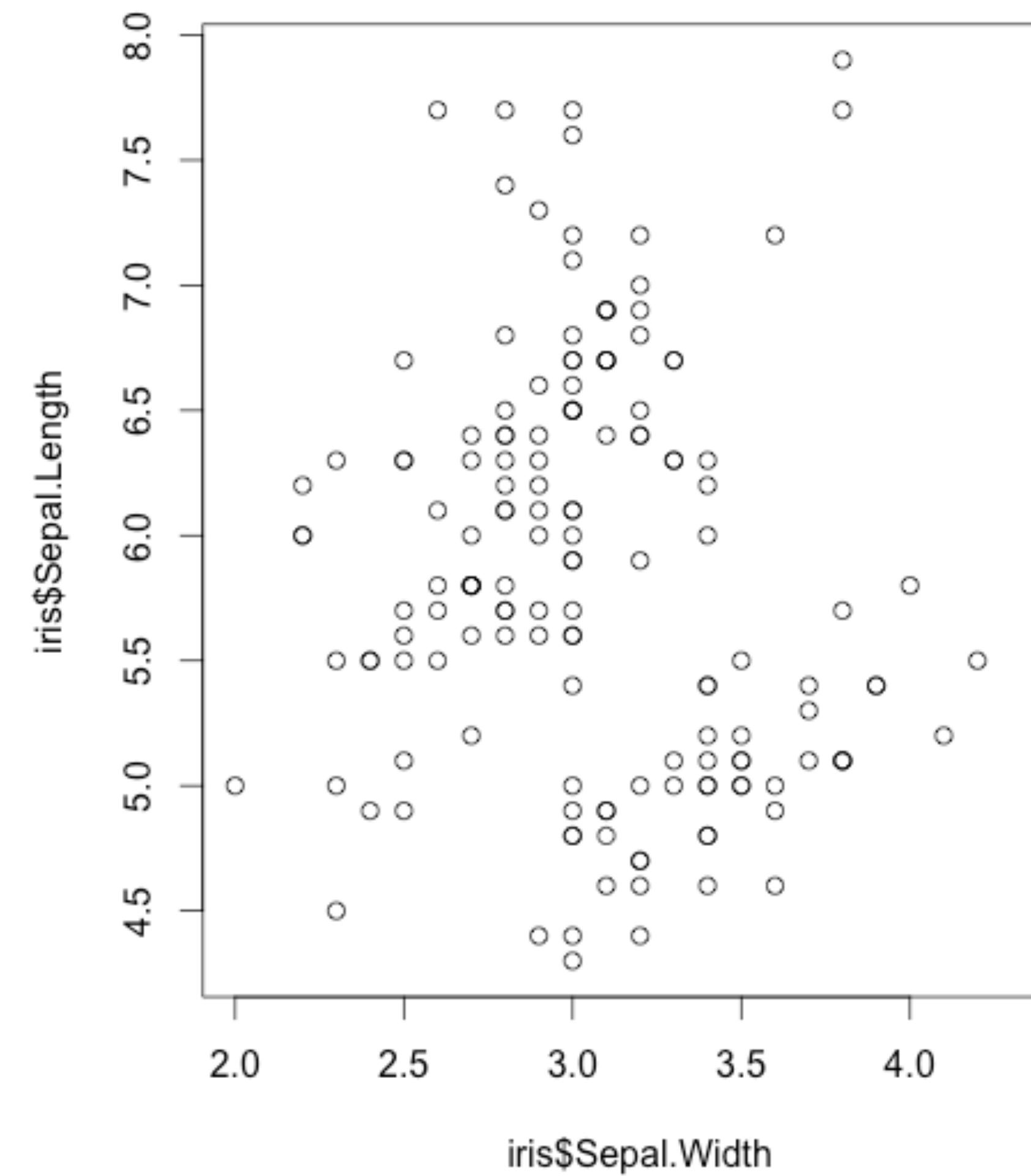
# ggplot2



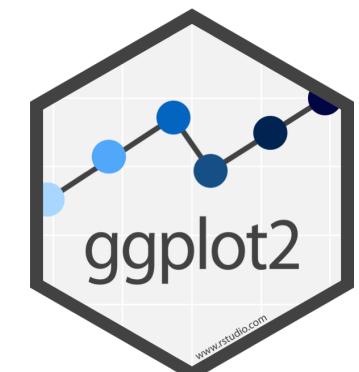
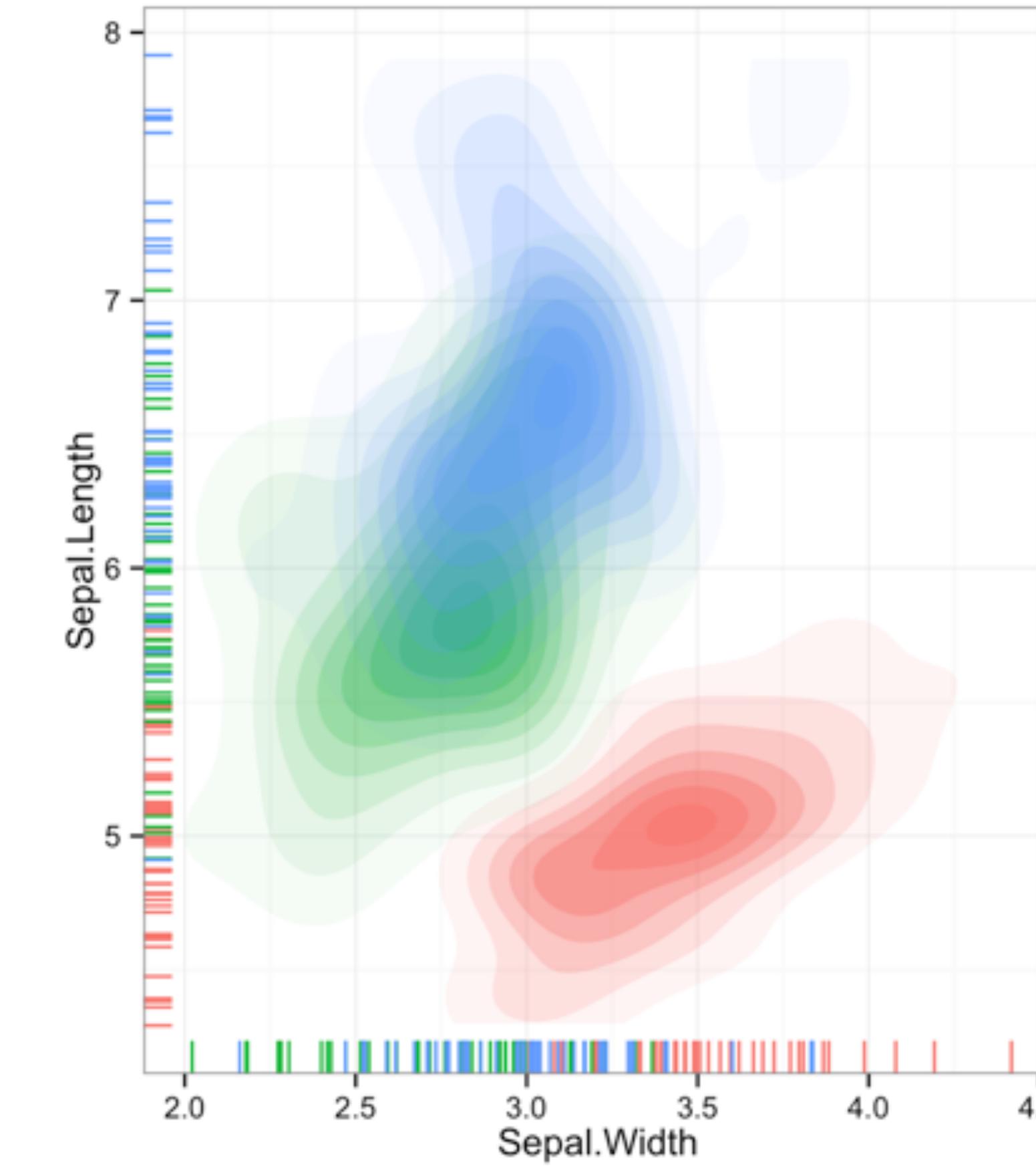
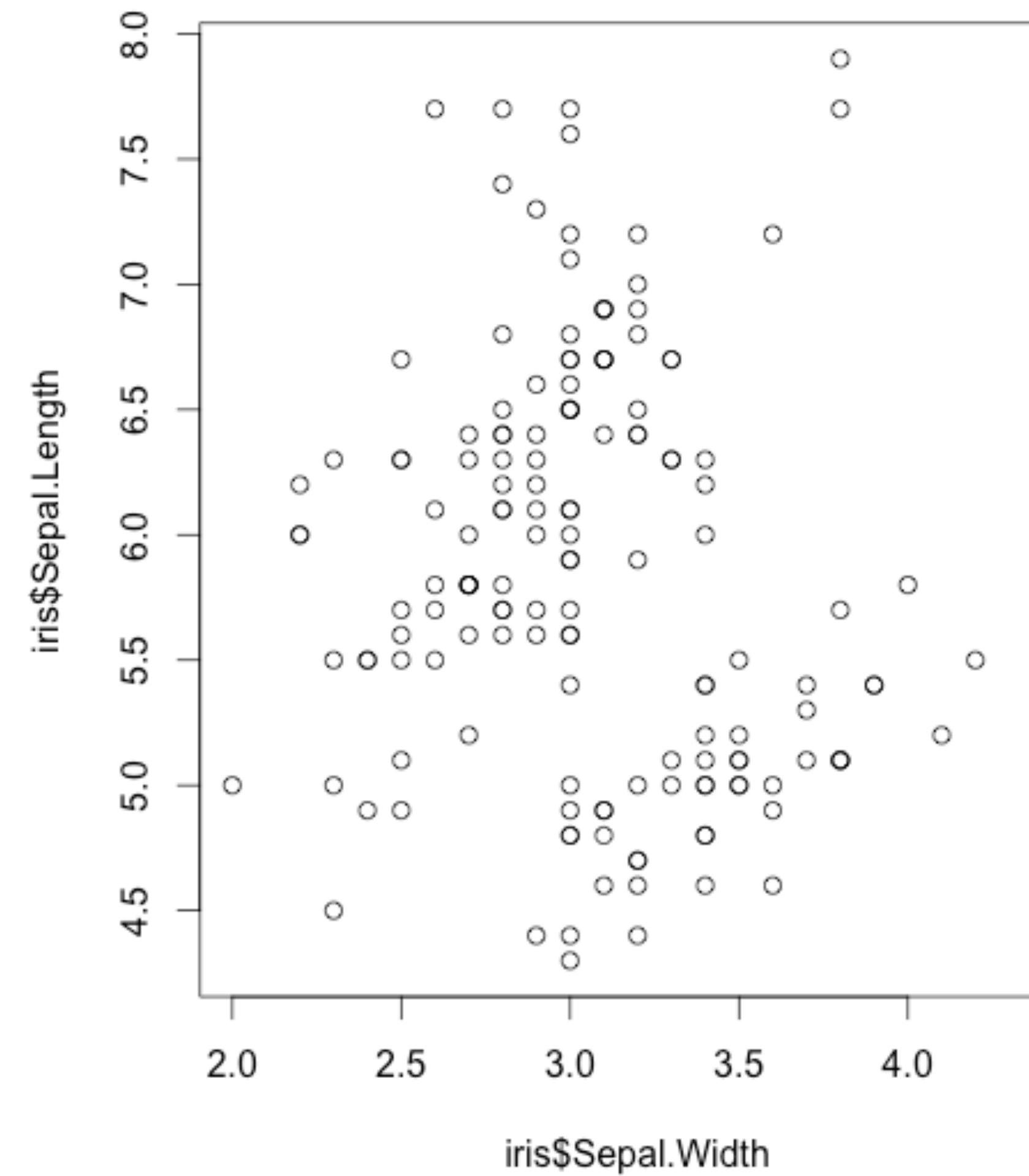
# ggplot2



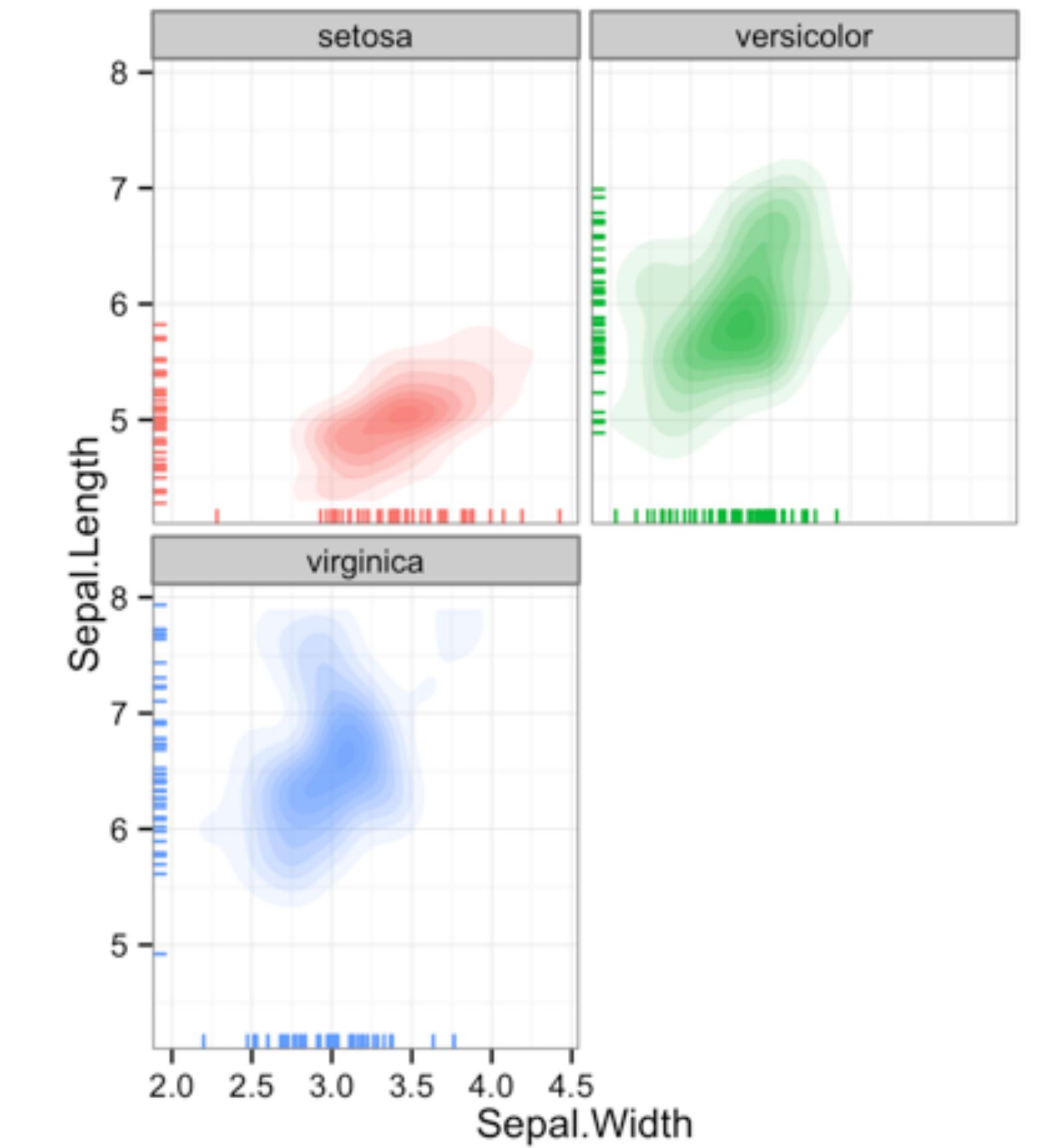
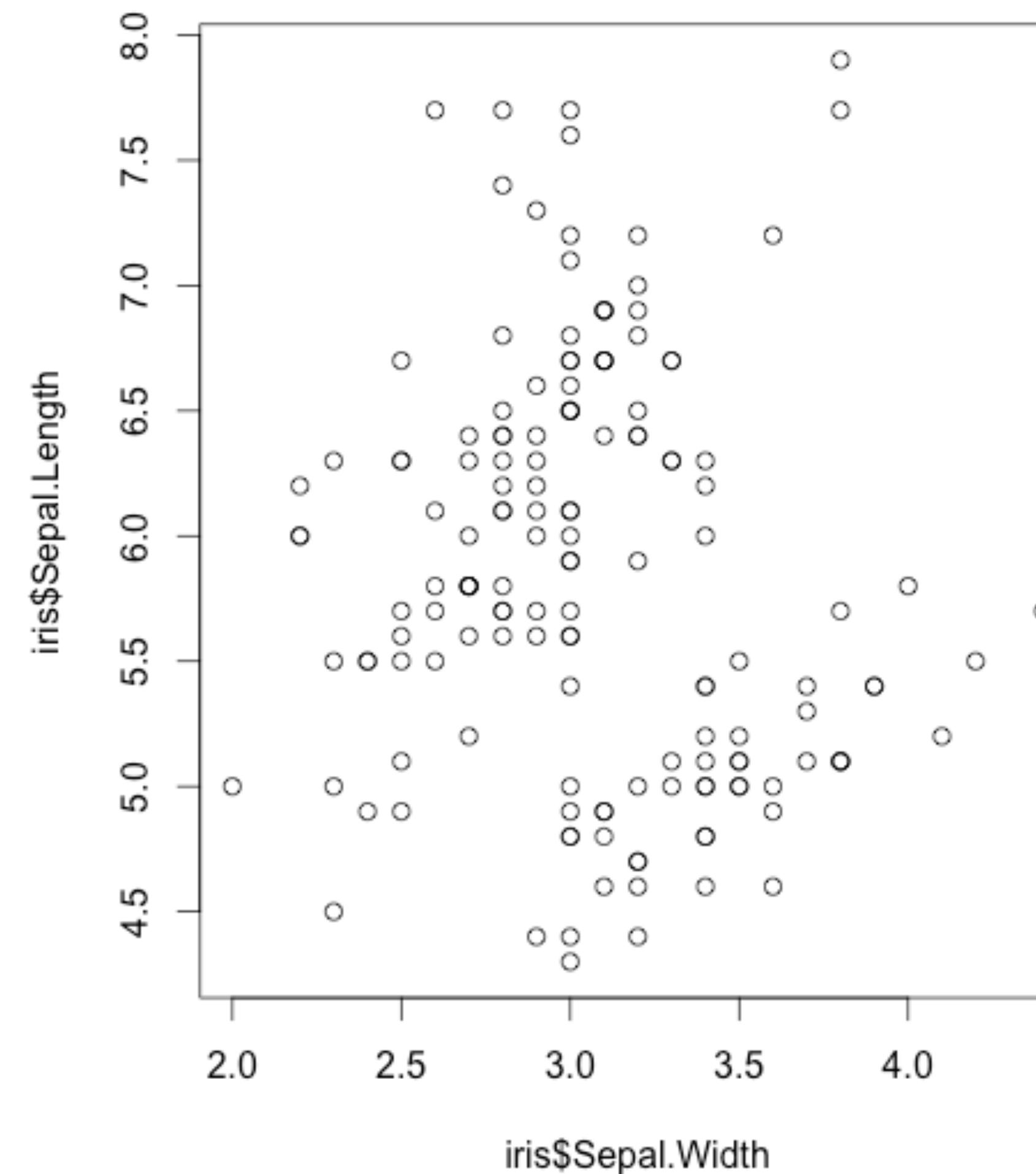
# ggplot2



# ggplot2

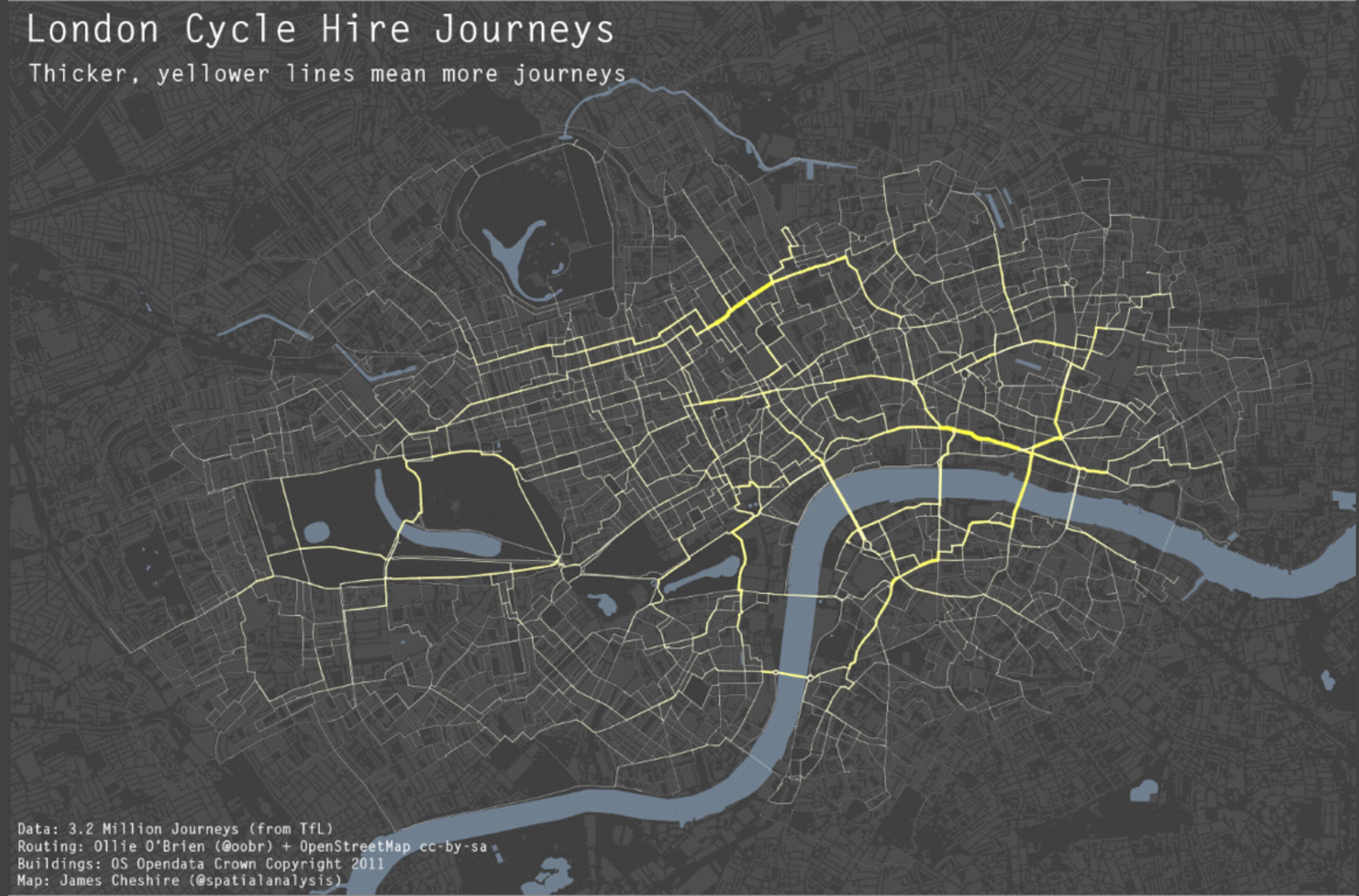


# ggplot2



# London Cycle Hire Journeys

Thicker, yellower lines mean more journeys



Data: 3.2 Million Journeys (from TfL)

Routing: Ollie O'Brien (@oobr) + OpenStreetMap cc-by-sa

Buildings: OS Opendata Crown Copyright 2011

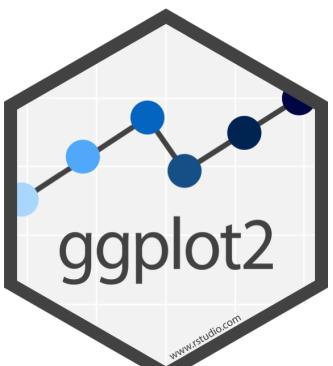
Map: James Cheshire (@spatialanalysis)

# mpg

Fuel economy data for 38 models of car.

```
View(mpg)
```

Capital "V"



# Your Turn

Confer with your group.

What relationship do you expect to see between engine size (displ) and mileage (hwy)?

No peeking ahead!



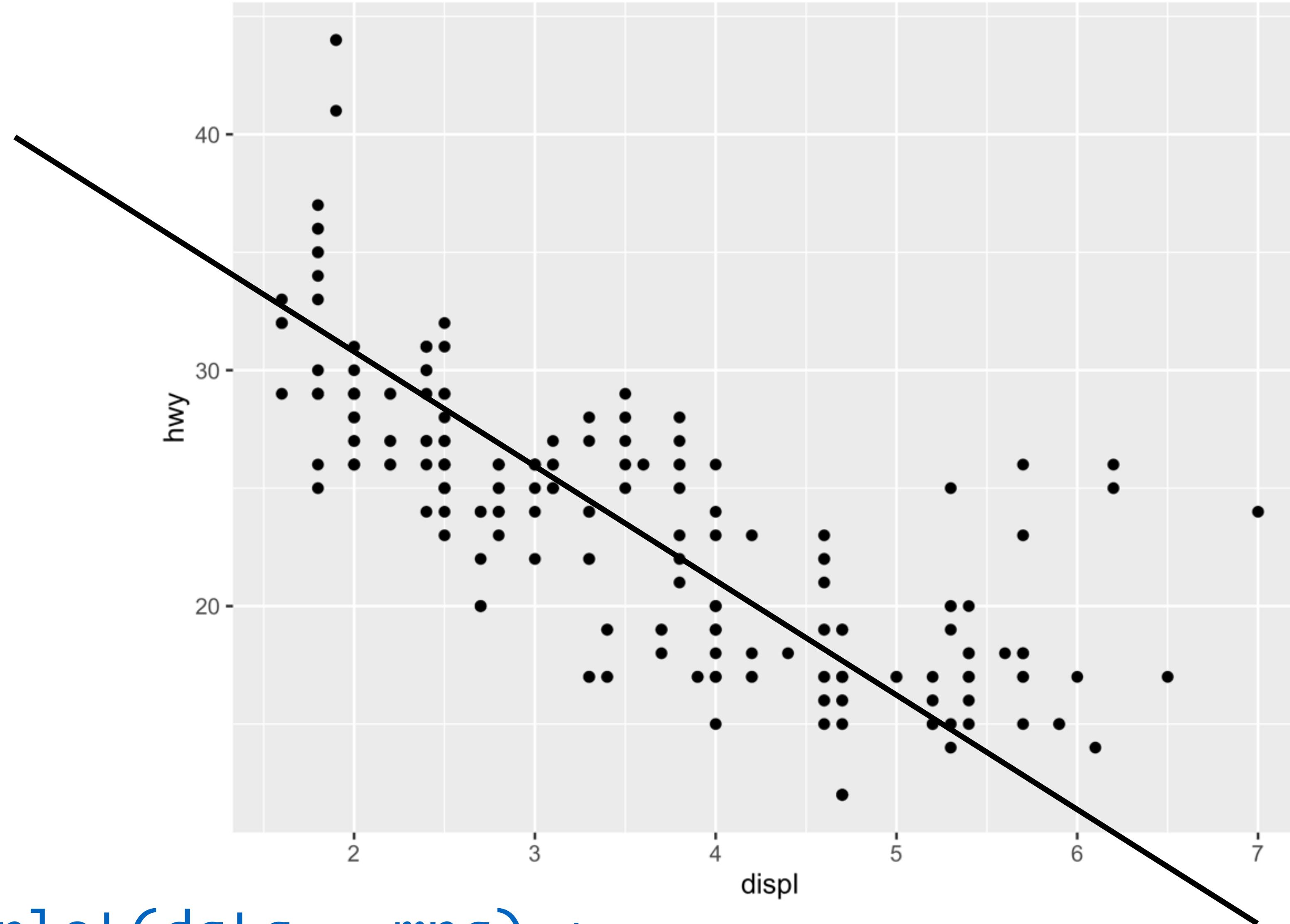
# Your Turn

Run this code at the command line to make a graph.

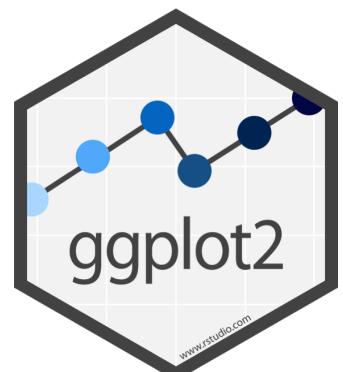
Pay strict attention to spelling, capitalization, and parentheses!

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```





```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```

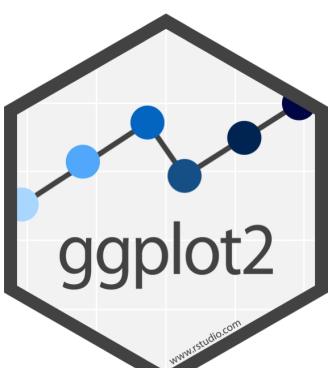


# To plot

1. "Initialize" a plot with `ggplot()`
2. Add layers with `geom_` functions

Pro tip: Always put the `+` at the end of a line,  
Never at the beginning

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```



# To plot

1. "Initialize" a plot with `ggplot()`
2. Add layers with `geom_` functions

data

+ before new line

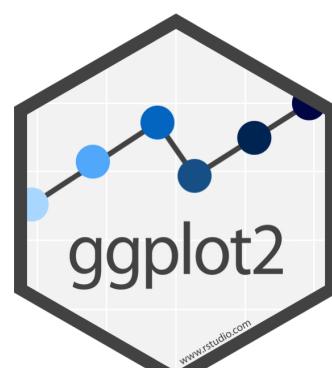
```
ggplot(mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```

type of layer

aes()

x variable

y variable

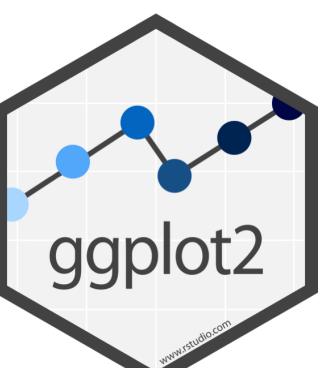


# A ggplot2 template

Make any plot by filling in the parameters of this template

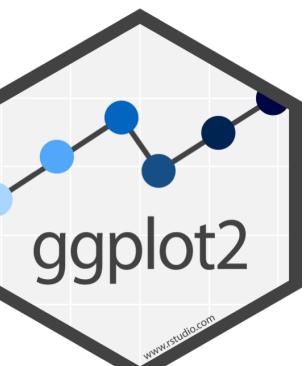
```
ggplot(data = <DATA>) +  
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```



```
ggplot(data = <DATA>) +  
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

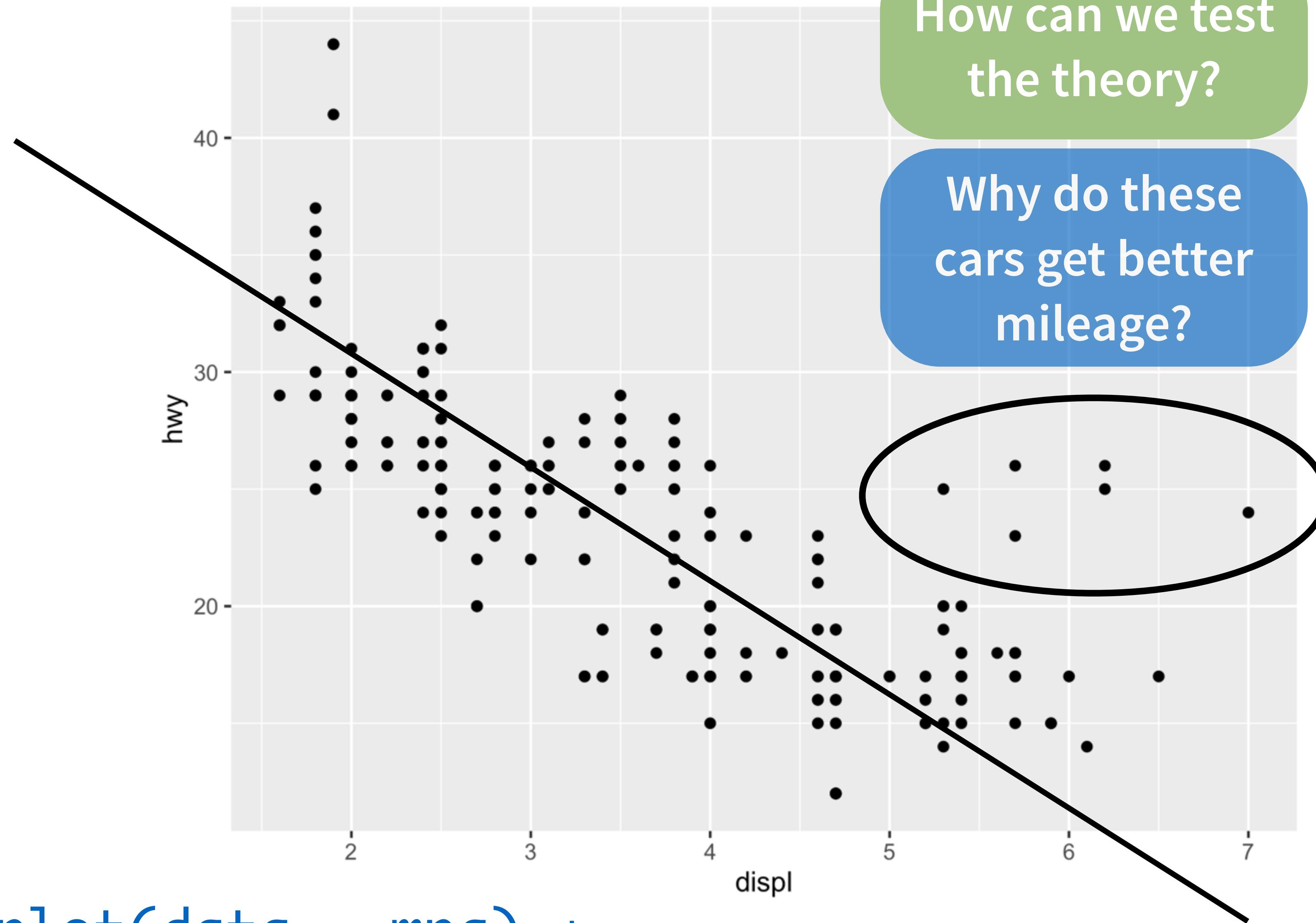
The variables to  
visualize



# Visualizing variables (aesthetic mappings)

"The greatest value of a picture is  
when it forces us to notice what we  
never expected to see."

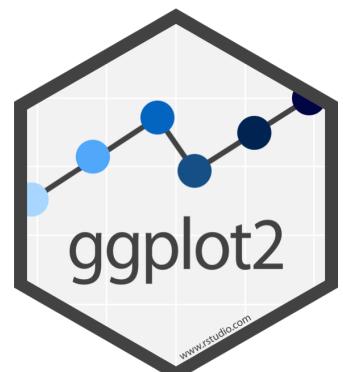
- John Tukey



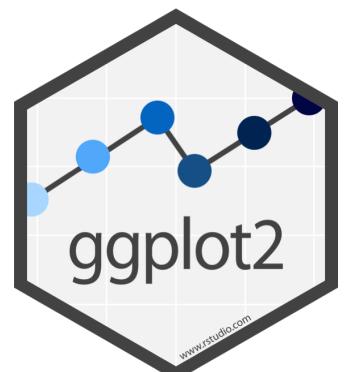
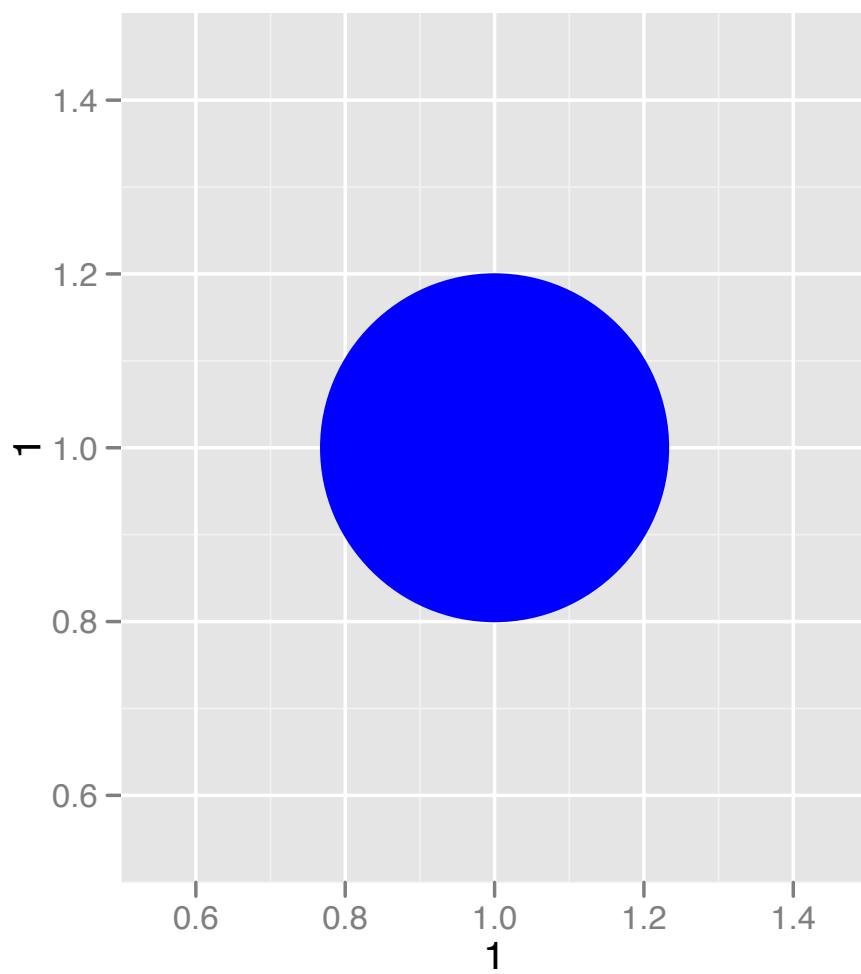
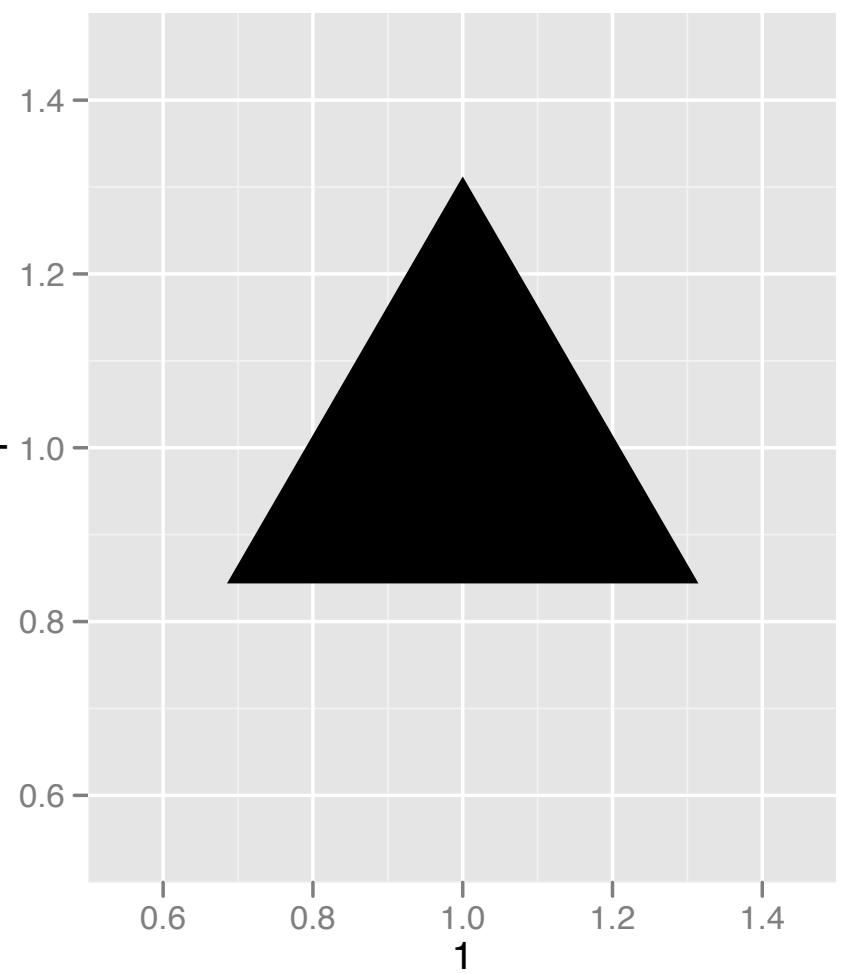
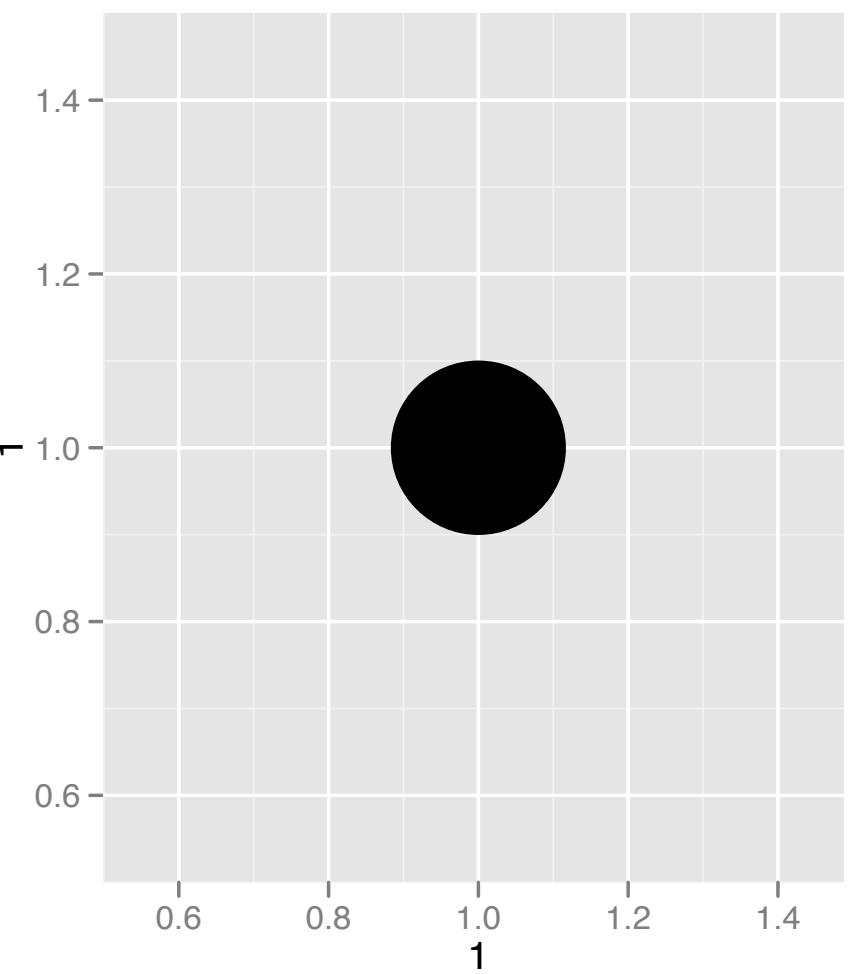
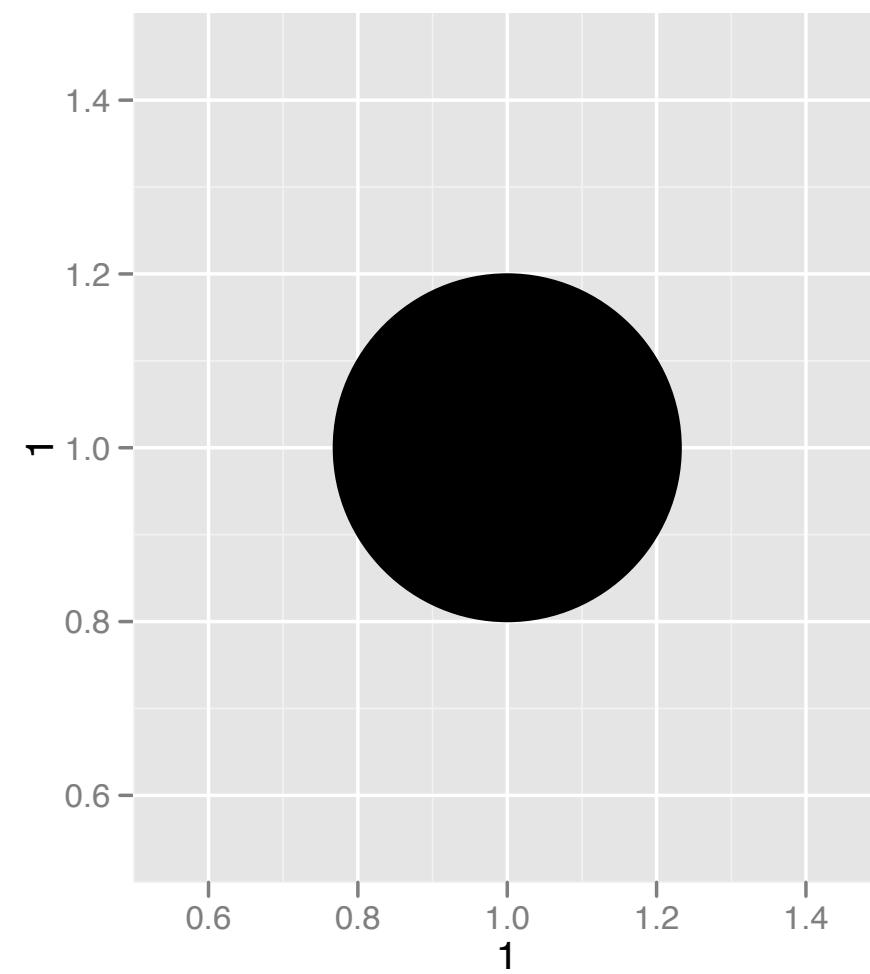
How can we test  
the theory?

Why do these  
cars get better  
mileage?

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```



# Aesthetics

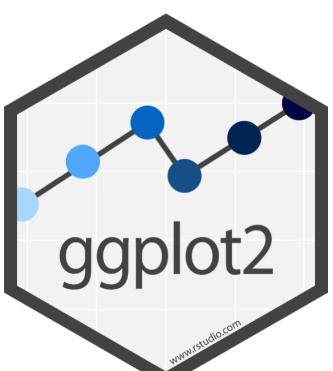


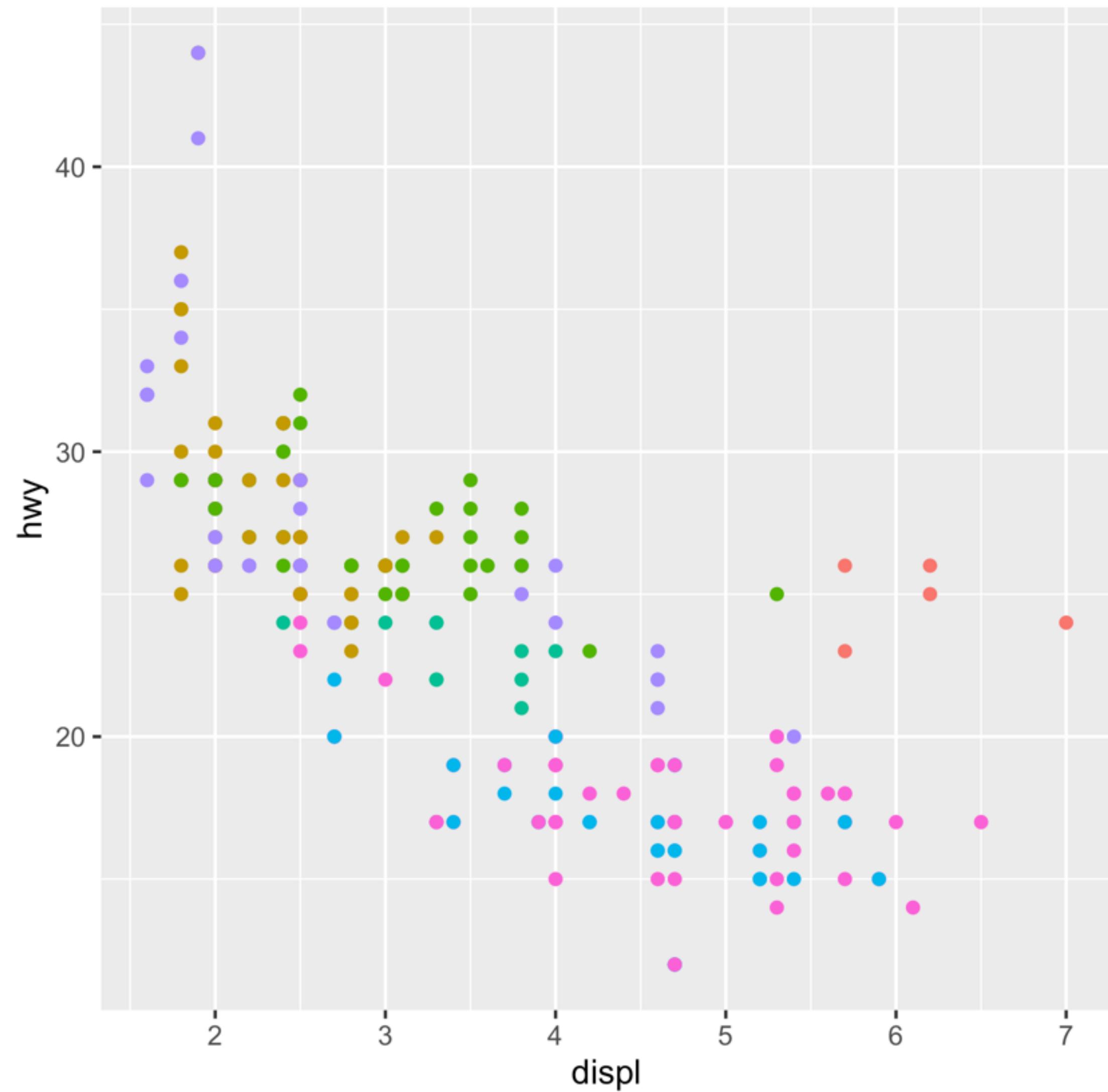
# Aesthetics

aesthetic  
property

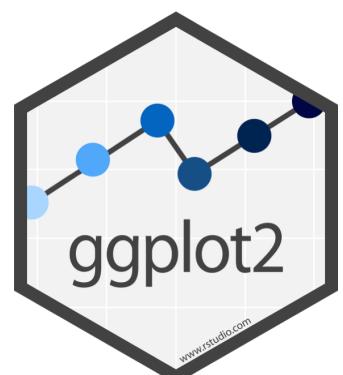
Variable to  
map it to

```
ggplot(mpg) + geom_point(aes(x = displ, y = hwy, color = class))  
ggplot(mpg) + geom_point(aes(x = displ, y = hwy, size = class))  
ggplot(mpg) + geom_point(aes(x = displ, y = hwy, shape = class))  
ggplot(mpg) + geom_point(aes(x = displ, y = hwy, alpha = class))
```





```
ggplot(mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



# Your Turn

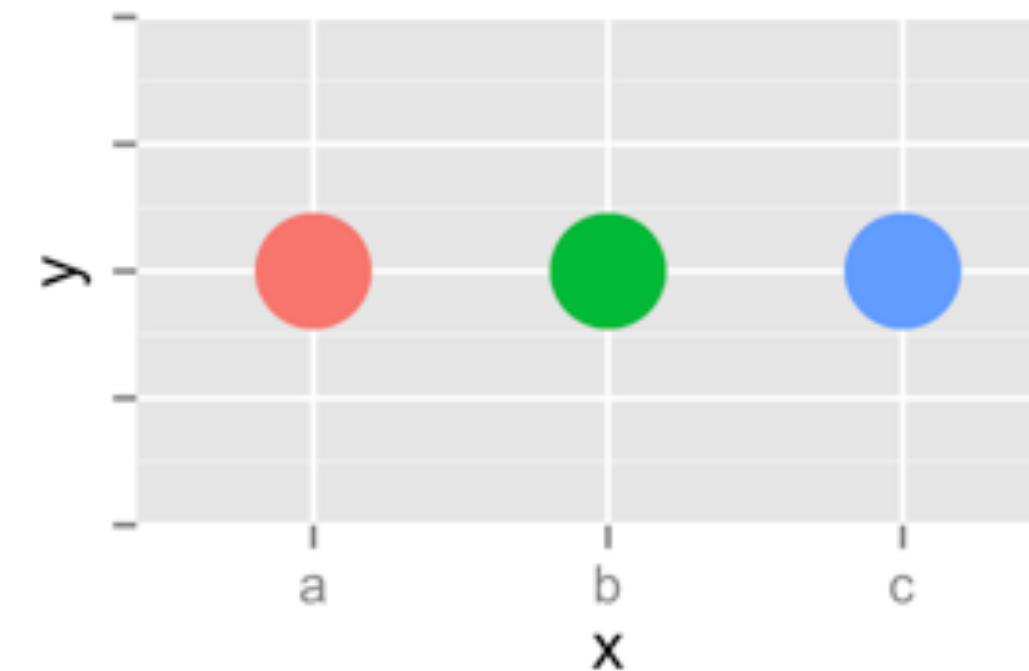
Add color, size, and shape aesthetics to your graph.  
Experiment.

Do different things happen when you map aesthetics to discrete and continuous variables?

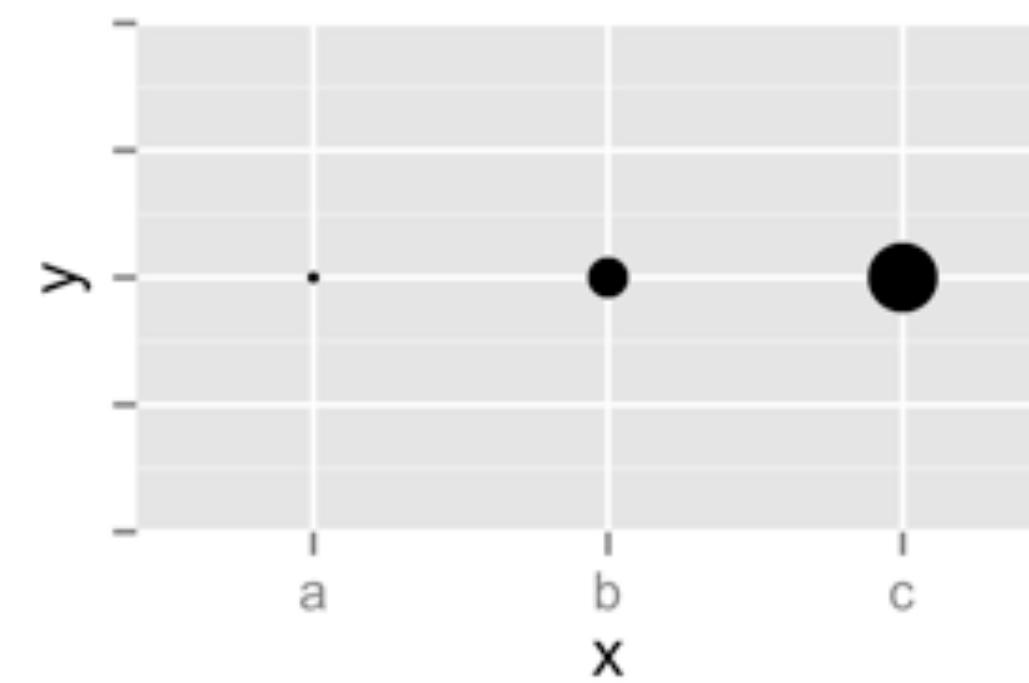
What happens when you use more than one aesthetic?



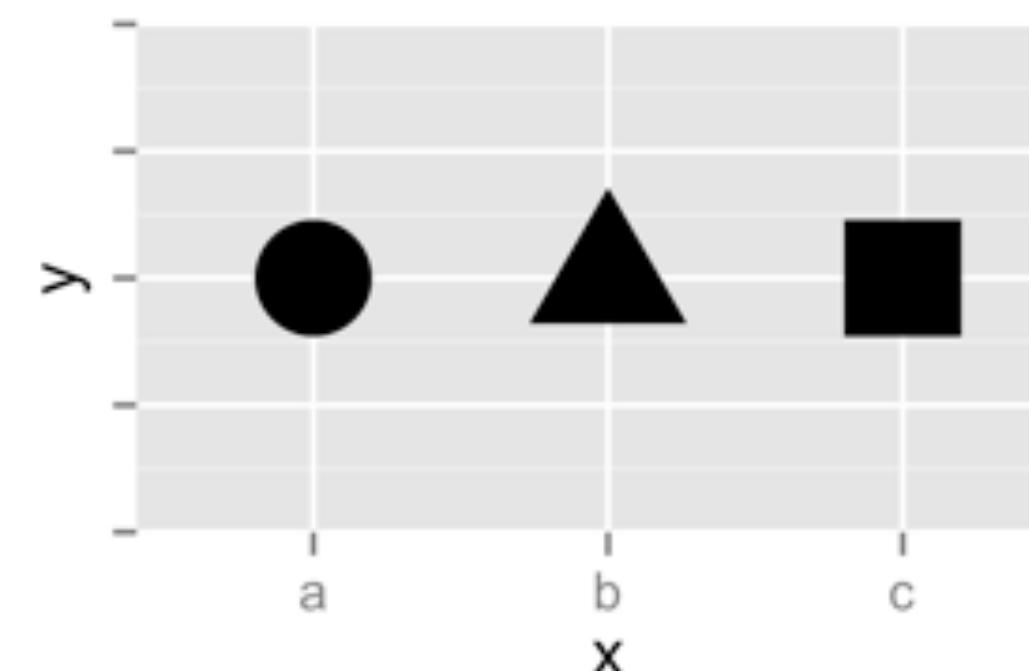
Color



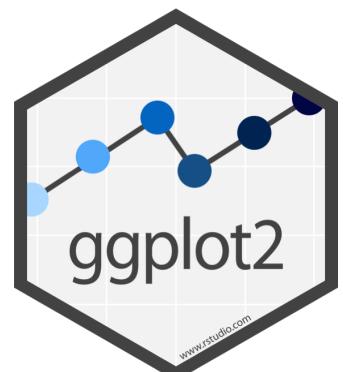
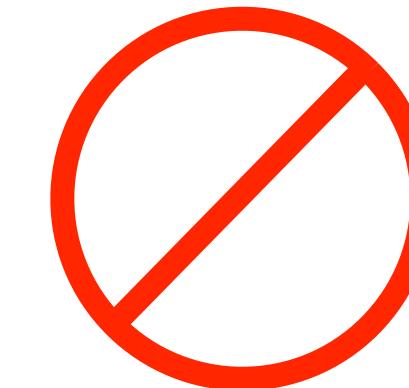
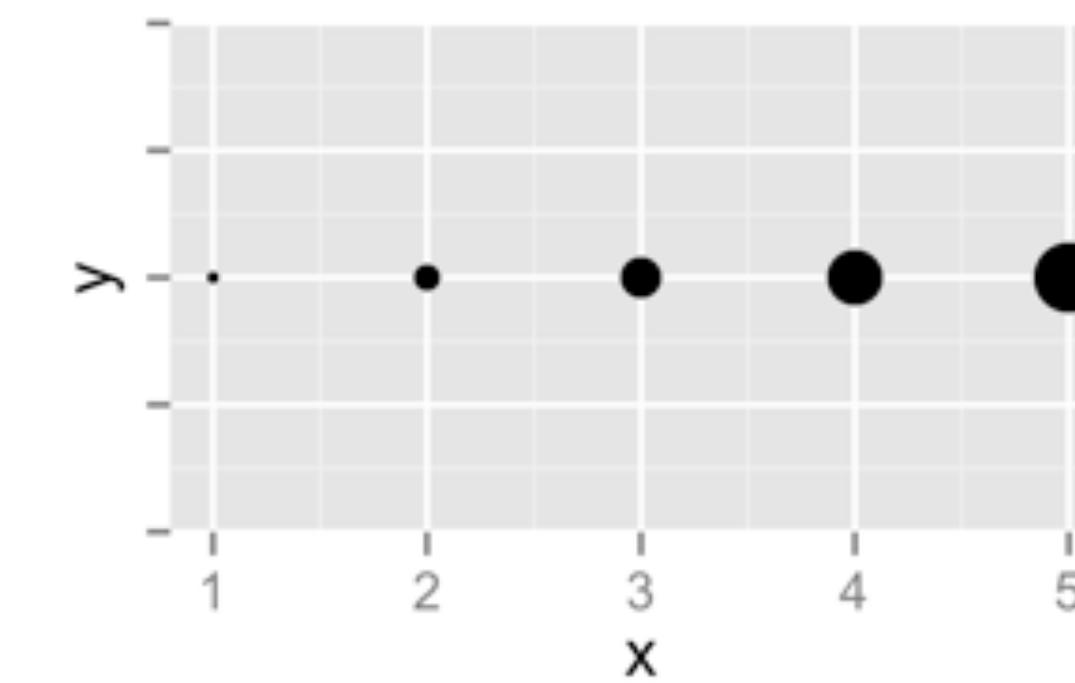
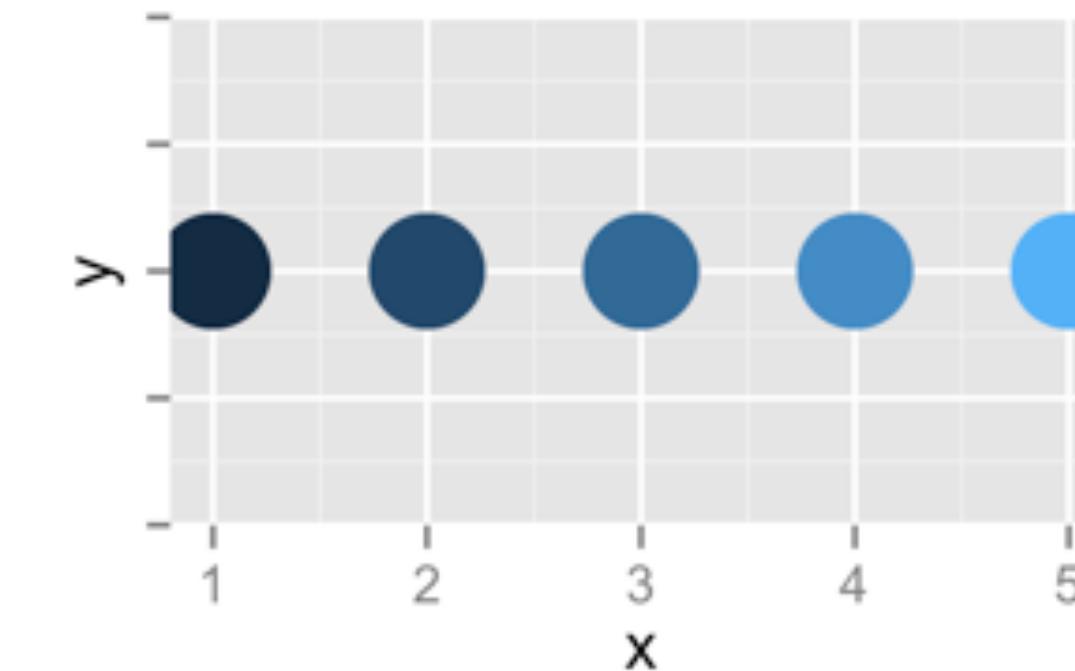
Size



Shape



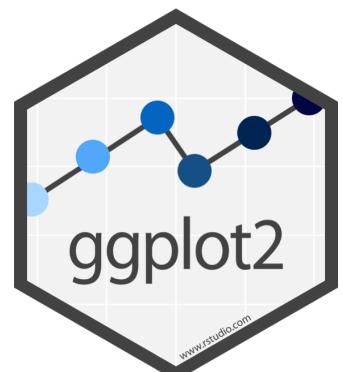
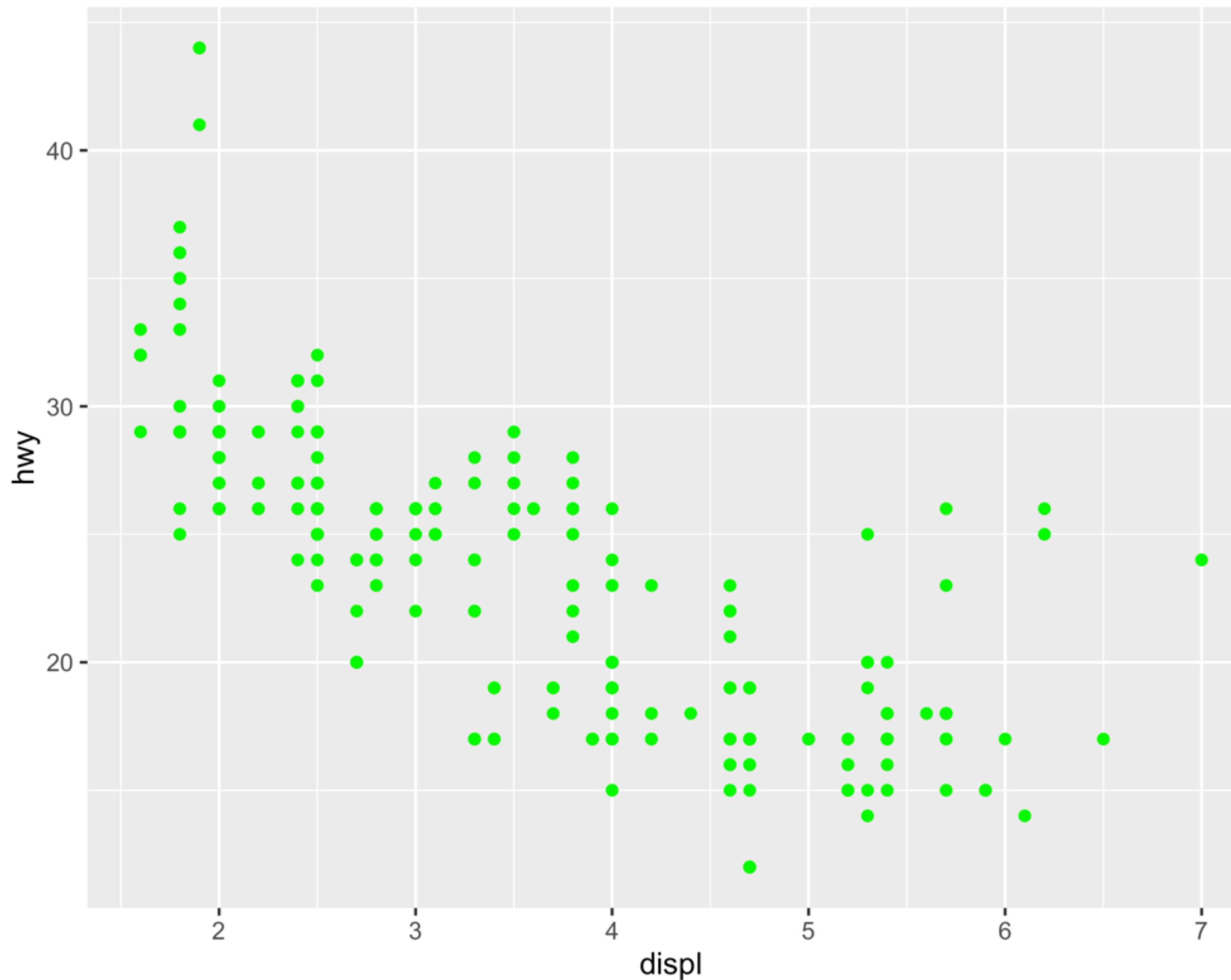
Continuous



# set vs. map

A faint watermark of the R logo is visible in the bottom right corner, consisting of a circular emblem with the letters "R" inside.

# How would you make this plot?



# Set vs. map

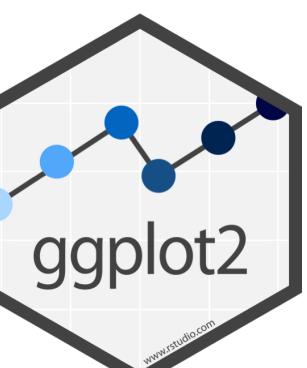
Inside of aes(): ggplot2 treats input as value in the data space and maps it to a value in the visual space.

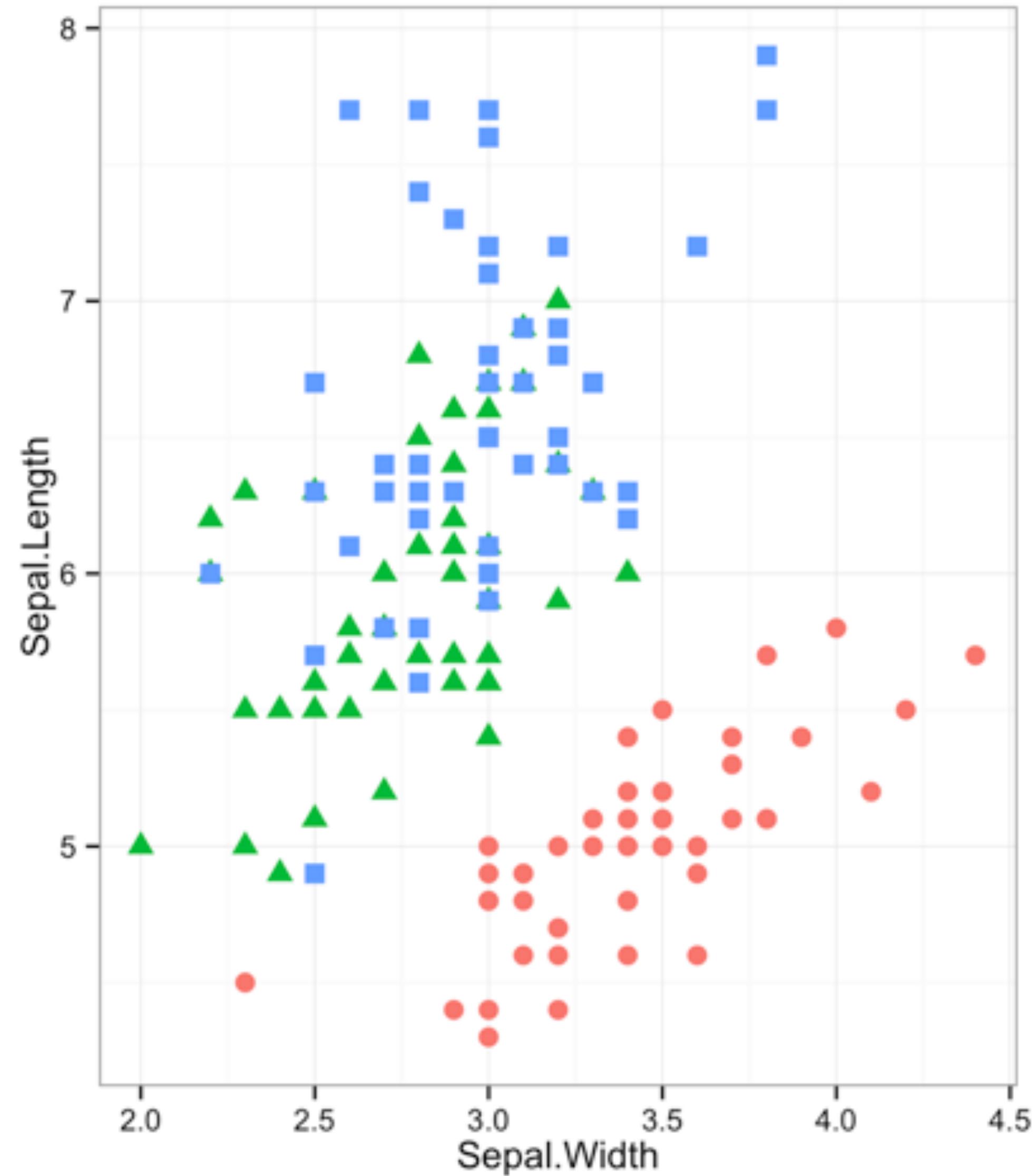
Inside aes()  
maps

```
ggplot(mpg) + geom_point(aes(x = displ, y = hwy, color = "green"))  
ggplot(mpg) + geom_point(aes(x = displ, y = hwy), color = "green")
```

Outside of aes(): ggplot2 treats input as value in the visual space and sets the property to it.

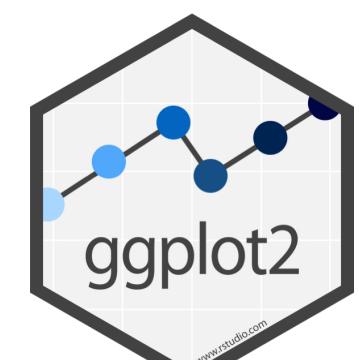
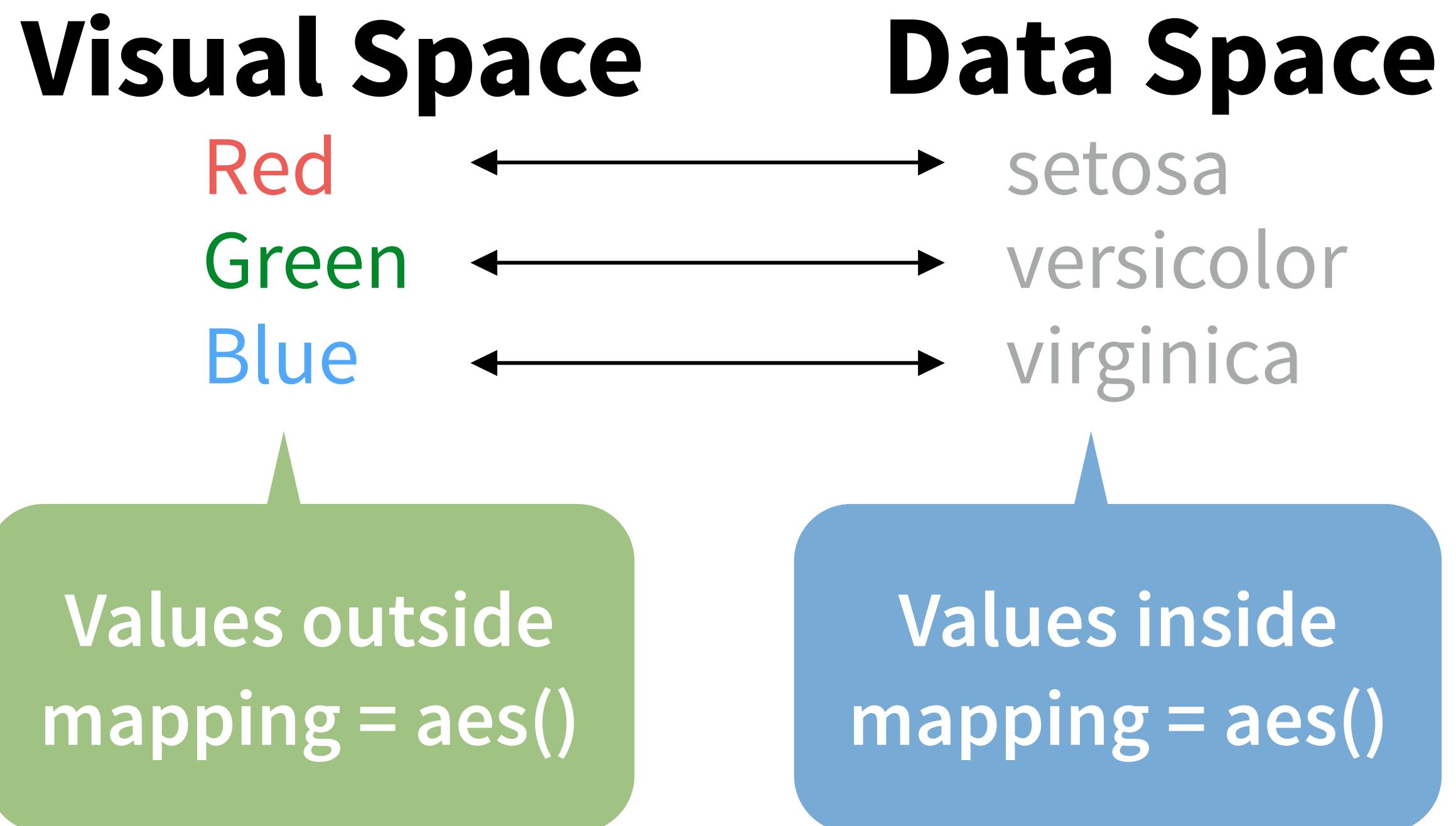
Outside aes()  
sets





Species

- setosa
- ▲ versicolor
- virginica



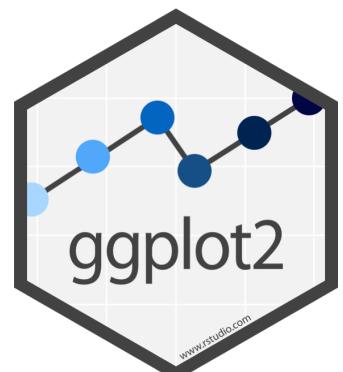
# Facets

A faint watermark of the R logo is visible in the bottom right corner of the slide.

R

# Facets

Subplots that display subsets of the data.



# Your turn

What do each of these do?

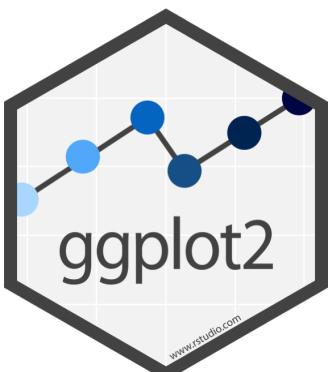
(run the code, interpret, convince your group)

```
q <- ggplot(mpg) + geom_point(aes(x = displ, y = hwy))  
q + facet_grid(. ~ cyl)  
q + facet_grid(drv ~ .)  
q + facet_grid(drv ~ cyl)  
q + facet_wrap(~ class)
```



# summary

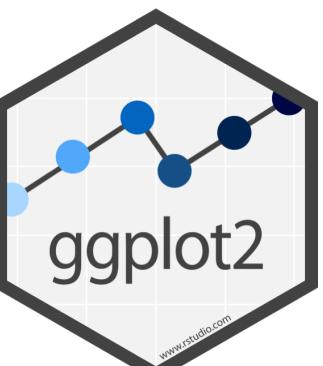
`facet_grid()` - 2D grid, rows ~ cols, . for no split  
`facet_wrap()` - 1D ribbon wrapped into 2D



# A ggplot2 template

Make any plot by filling in the parameters of this template

```
ggplot(data = <DATA>) +  
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>)) +  
<FACET_FUNCTION>
```

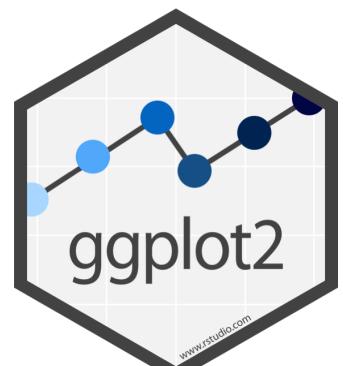
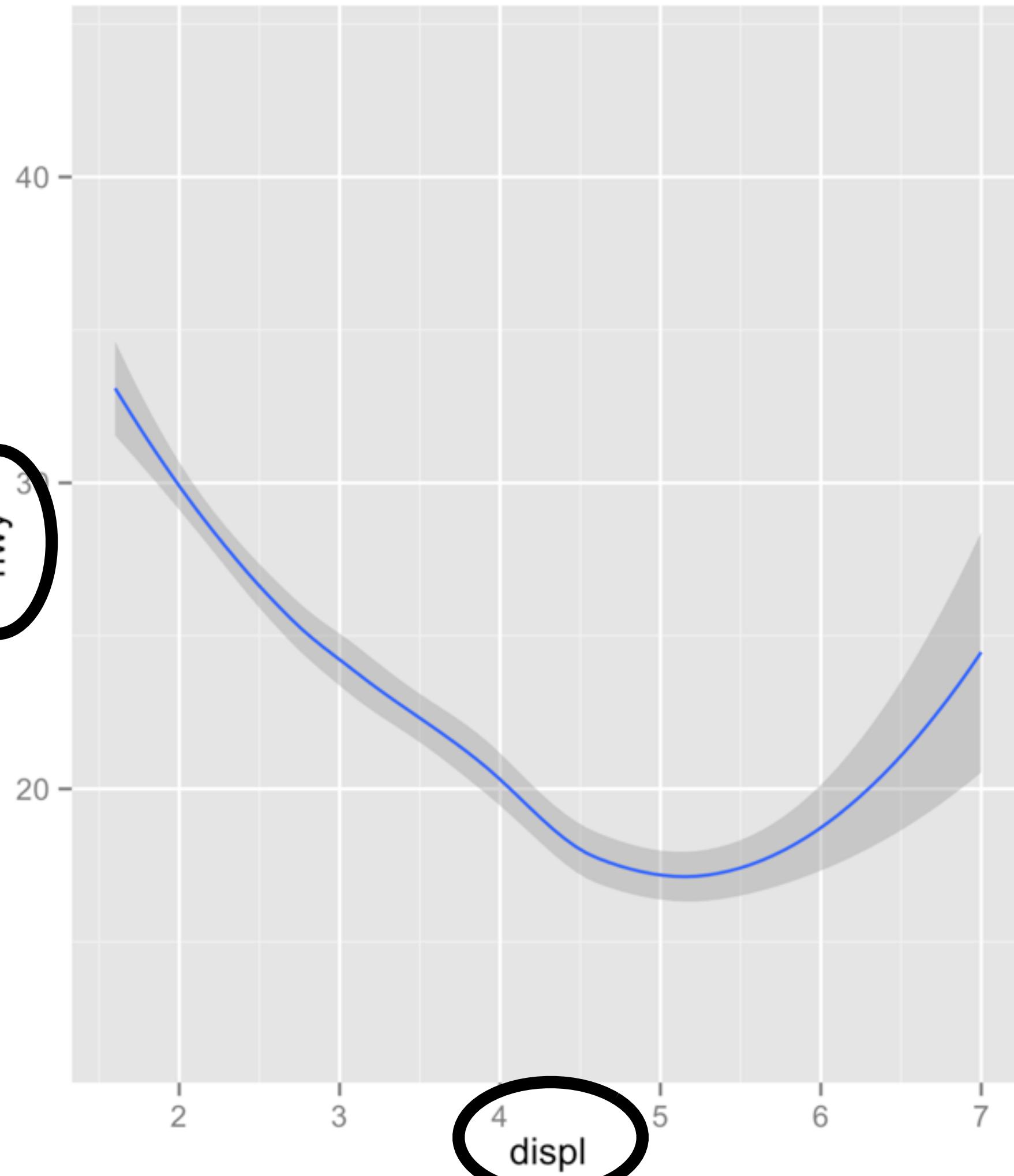
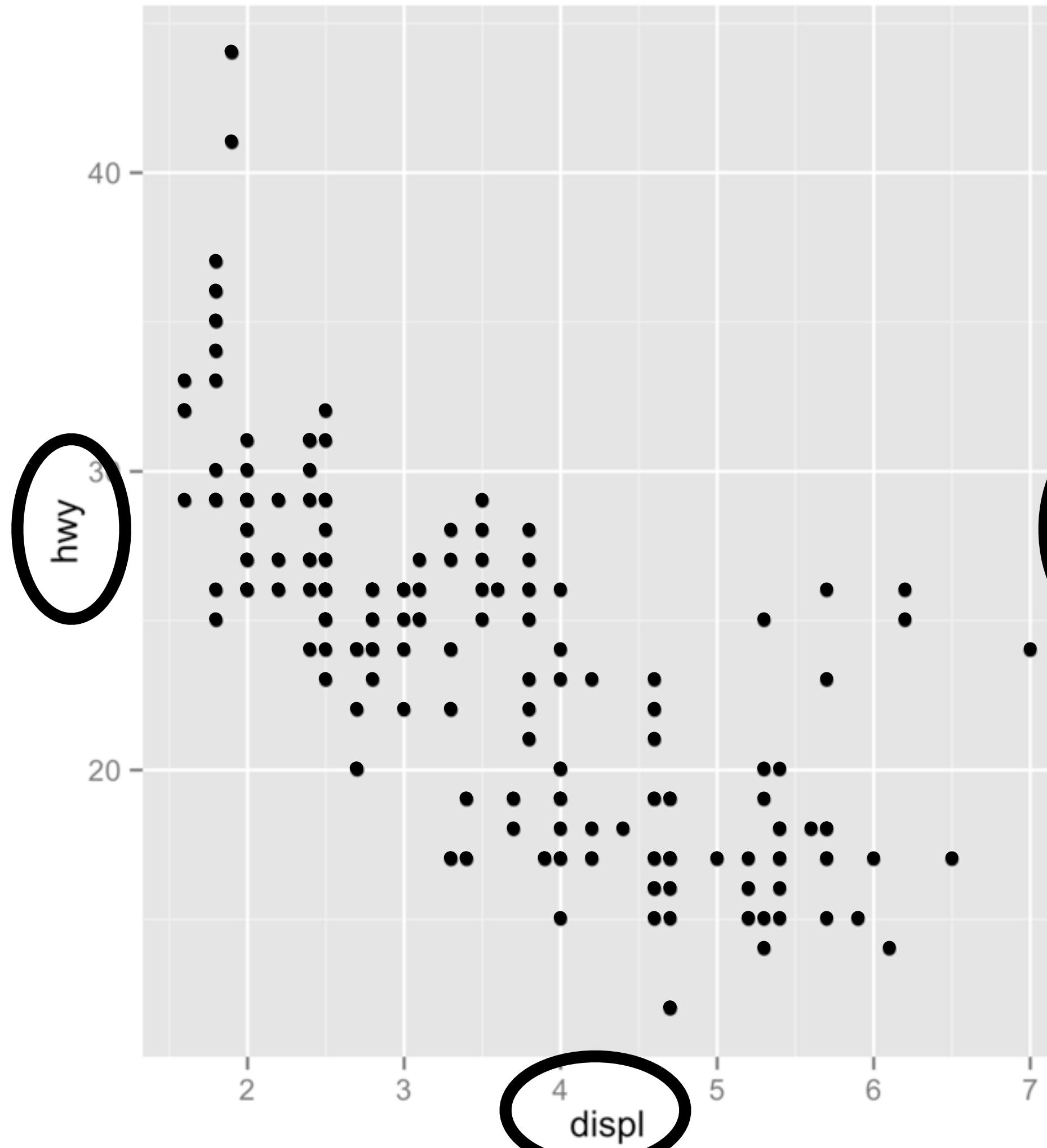


# Visualizing cases (geoms)



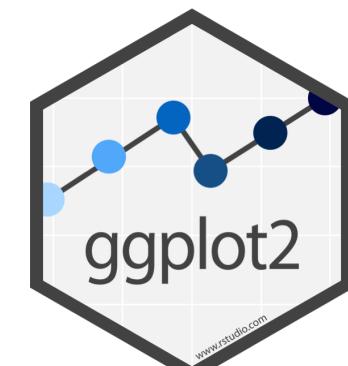
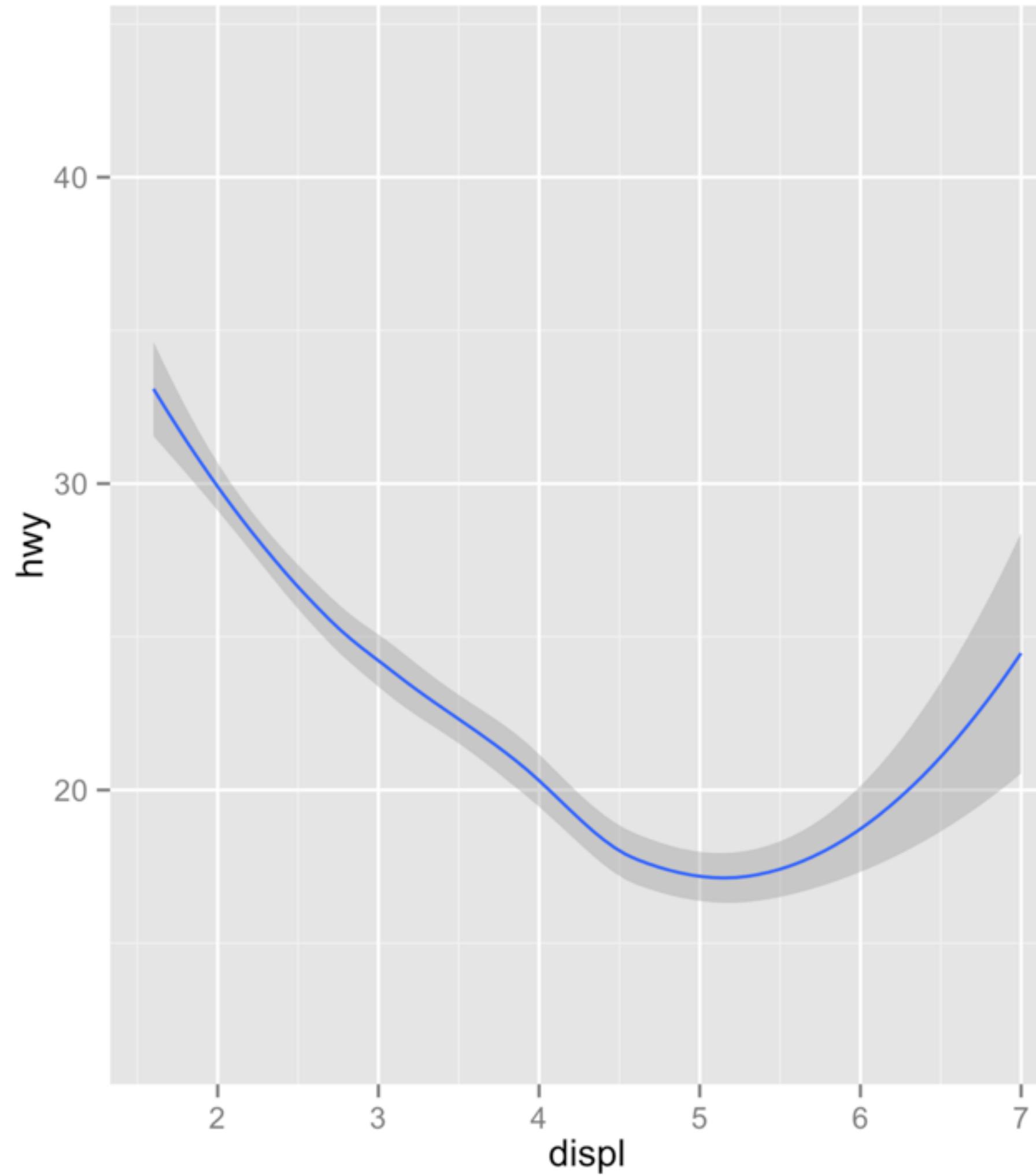
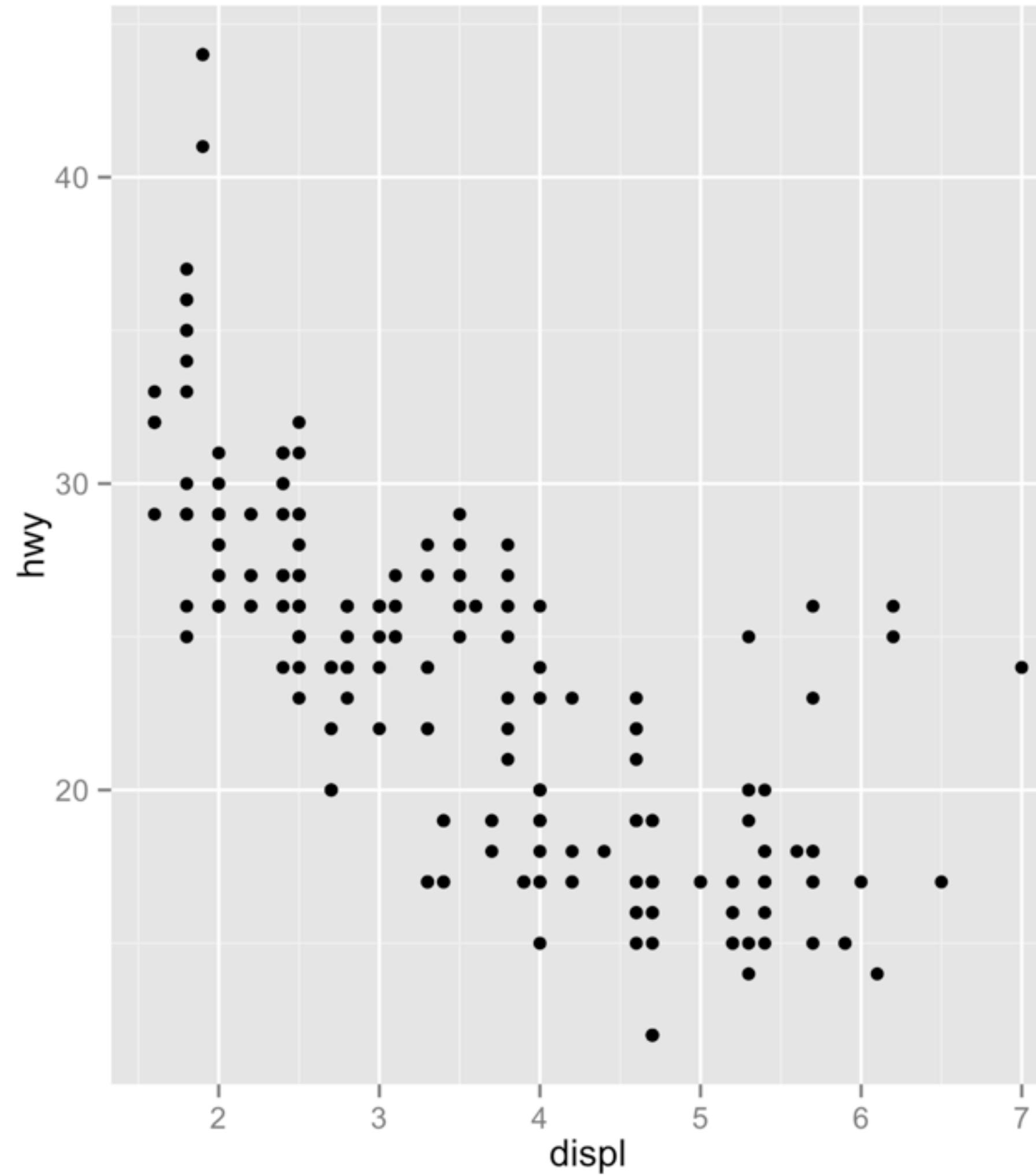
How are these plots similar?

Same: x var , y var , data



How are these plots different?

Different: geometric object (geom),  
e.g. the visual object used to represent the data

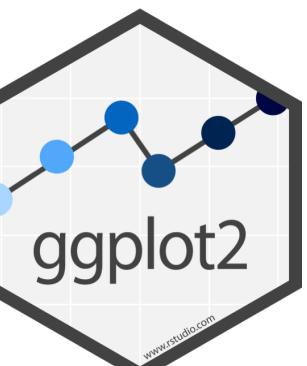


# geom

The visual objects that represents the cases. geom functions add layers to the graph.

```
ggplot(data = <DATA>) +  
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

The geom

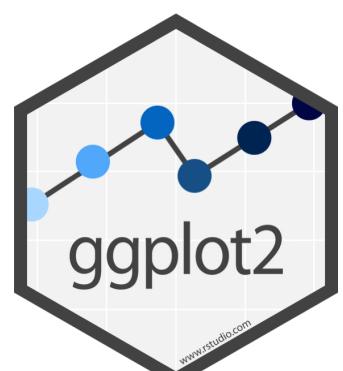


# geom\_ functions

Each has a mapping argument to set variables.

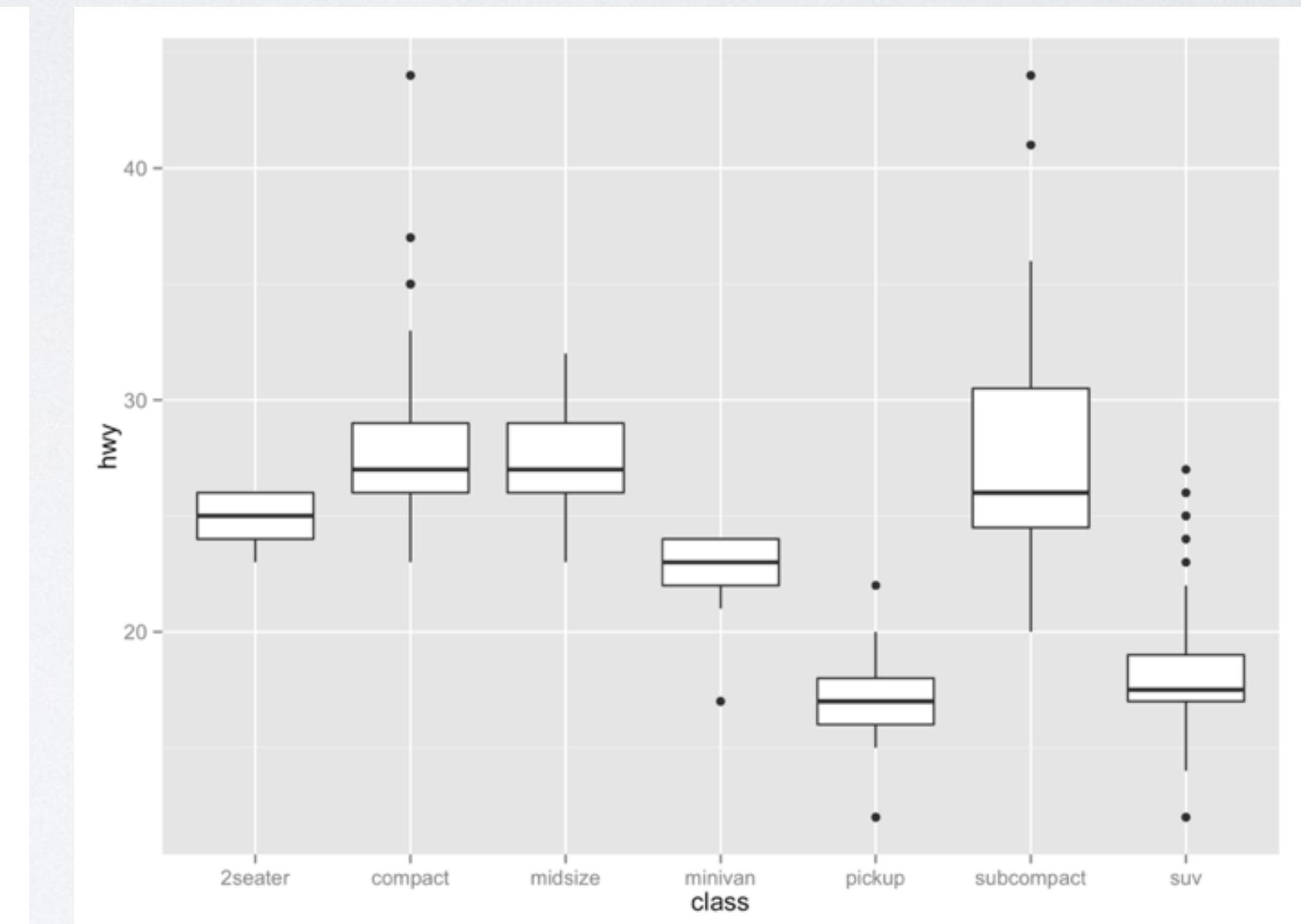
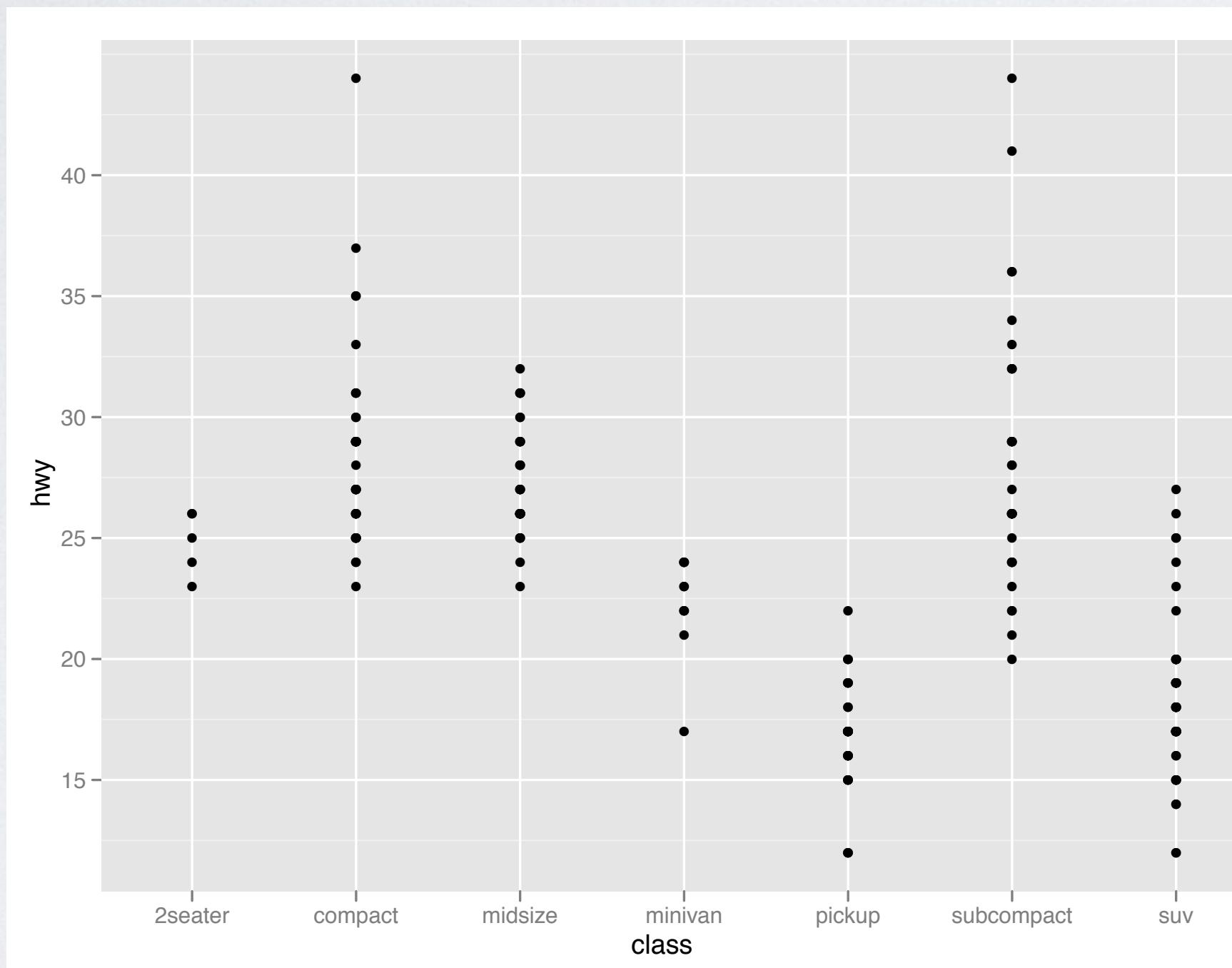


Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.	
<b>One Variable</b> <ul style="list-style-type: none"> <li><b>Continuous</b> <ul style="list-style-type: none"> <li>a + <code>geom_area(stat = "bin")</code> x, y, alpha, color, fill, linetype, size b + <code>geom_area(aes(y = ..density..), stat = "bin")</code></li> <li>a + <code>geom_density(kernel = "gaussian")</code> x, y, alpha, color, fill, linetype, size, weight b + <code>geom_density(aes(y = ..count..))</code></li> <li>a + <code>geom_dotplot()</code> x, y, alpha, color, fill</li> <li>a + <code>geom_freqpoly()</code> x, y, alpha, color, linetype, size b + <code>geom_freqpoly(aes(y = ..density..))</code></li> <li>a + <code>geom_histogram(binwidth = 5)</code> x, y, alpha, color, fill, linetype, size, weight b + <code>geom_histogram(aes(y = ..density..))</code></li> </ul> </li> <li><b>Discrete</b> <ul style="list-style-type: none"> <li>b &lt;- <code>ggplot(mpg, aes(f1))</code></li> <li>b + <code>geom_bar()</code> x, alpha, color, fill, linetype, size, weight</li> </ul> </li> </ul>	<b>Two Variables</b> <ul style="list-style-type: none"> <li><b>Continuous X, Continuous Y</b> <ul style="list-style-type: none"> <li>f + <code>geom_blank()</code> (Useful for expanding limits)</li> <li>f + <code>geom_jitter()</code> x, y, alpha, color, fill, shape, size</li> <li>f + <code>geom_point()</code> x, y, alpha, color, fill, shape, size</li> <li>f + <code>geom_quantile()</code> x, y, alpha, color, linetype, size, weight</li> <li>f + <code>geom_rug(sides = "bl")</code> alpha, color, linetype, size</li> <li>f + <code>geom_smooth(method = lm)</code> x, y, alpha, color, fill, linetype, size, weight</li> <li>C f + <code>geom_text(aes(label = cty))</code> x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust</li> </ul> </li> <li><b>Continuous Function</b> <ul style="list-style-type: none"> <li>j &lt;- <code>ggplot(economics, aes(date, unemploy))</code></li> <li>j + <code>geom_area()</code> x, y, alpha, color, fill, linetype, size</li> <li>j + <code>geom_line()</code> x, y, alpha, color, linetype, size</li> <li>j + <code>geom_step(direction = "hv")</code> x, y, alpha, color, linetype, size</li> </ul> </li> <li><b>Visualizing error</b> <ul style="list-style-type: none"> <li>df &lt;- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)</li> <li>k &lt;- <code>ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))</code></li> <li>k + <code>geom_crossbar(fatten = 2)</code> x, y, ymax, ymin, alpha, color, fill, linetype, size</li> <li>k + <code>geom_errorbar()</code> lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight</li> <li>k + <code>geom_linerange()</code> x, ymin, ymax, alpha, color, linetype, size</li> <li>k + <code>geom_pointrange()</code> x, y, ymin, ymax, alpha, color, fill, linetype, shape, size</li> </ul> </li> <li><b>Maps</b> <ul style="list-style-type: none"> <li>data &lt;- data.frame(murder = USArrests\$Murder, state = tolower(rownames(USArrests)))</li> <li>map &lt;- <code>map_data("state")</code></li> <li>l &lt;- <code>ggplot(data, aes(fill = murder))</code></li> <li>l + <code>geom_map(aes(map_id = state), map = map) + expand_limits(x = map\$long, y = map\$lat)</code> map_id, alpha, color, fill, linetype, size</li> </ul> </li> </ul>
<b>Graphical Primitives</b> <ul style="list-style-type: none"> <li>g &lt;- <code>ggplot(mpg, aes(class, hwy))</code></li> <li>g + <code>geom_bar(stat = "identity")</code> x, y, alpha, color, fill, linetype, size, weight</li> <li>c + <code>geom_polygon(aes(group = group))</code> x, y, alpha, color, fill, linetype, size</li> <li>d &lt;- <code>ggplot(economics, aes(date, unemploy))</code></li> <li>d + <code>geom_path(lineend = "butt", linejoin = "round", linemetre = 1)</code> x, y, alpha, color, linetype, size</li> <li>d + <code>geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900))</code> x, ymax, ymin, alpha, color, fill, linetype, size</li> <li>e &lt;- <code>ggplot(seals, aes(x = long, y = lat))</code></li> <li>e + <code>geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat))</code> x, xend, y, yend, alpha, color, linetype, size</li> <li>e + <code>geom_rect(aes(xmin = long, ymin = lat, xmax = long + delta_long, ymax = lat + delta_lat))</code> xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size</li> </ul>	<b>Three Variables</b> <ul style="list-style-type: none"> <li>seals\$z &lt;- with(seals, sqrt(delta_long^2 + delta_lat^2))</li> <li>m &lt;- <code>ggplot(seals, aes(long, lat))</code></li> <li>m + <code>geom_raster(aes(fill = z), hijst = 0.5, vjust = 0.5, interpolate = FALSE)</code> x, y, alpha, fill (fast)</li> <li>m + <code>geom_contour(aes(z = z))</code> x, y, z, alpha, colour, linetype, size, weight</li> <li>m + <code>geom_tile(aes(fill = z))</code> x, y, alpha, color, fill, linetype, size (slow)</li> </ul>



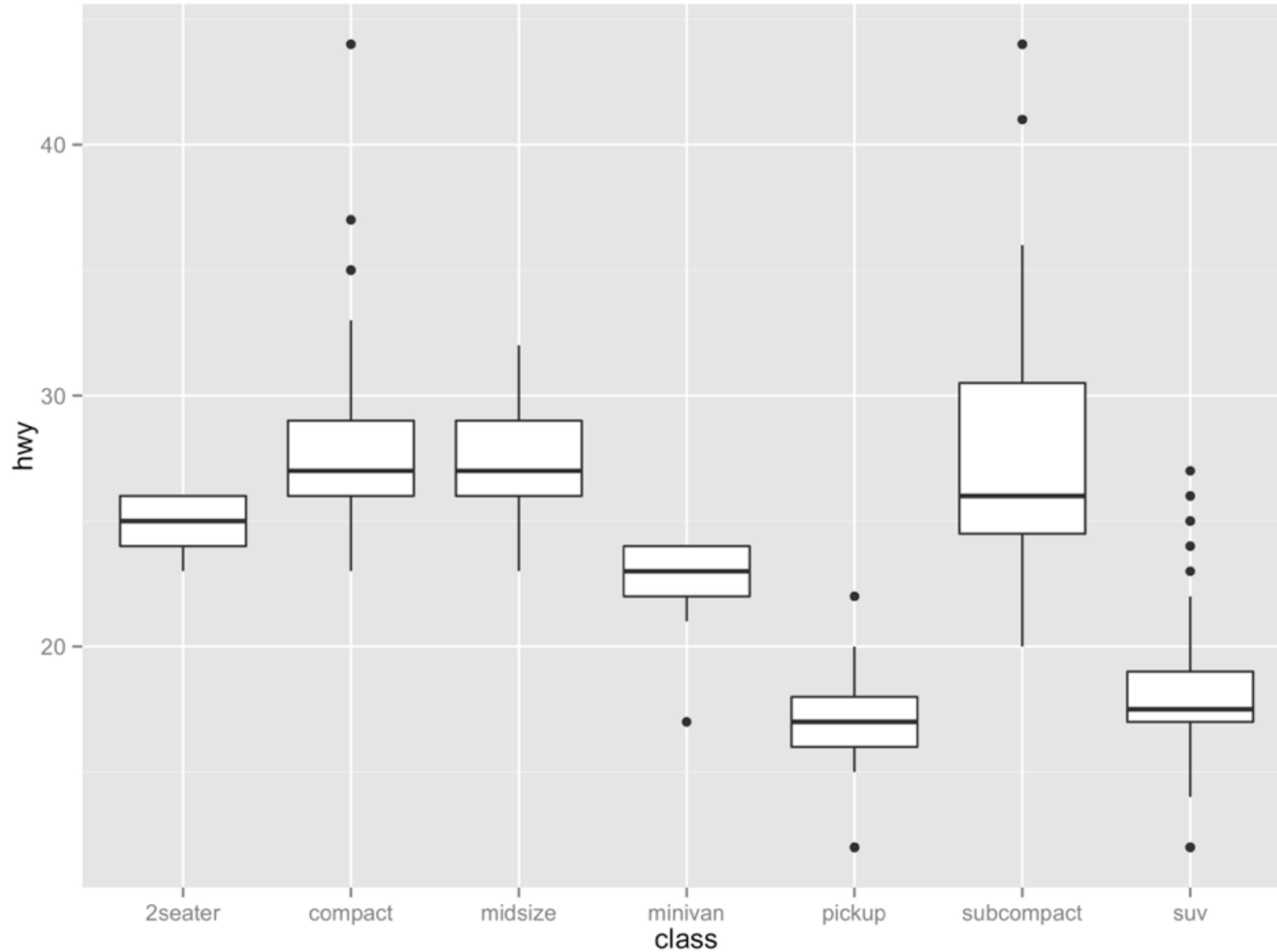
# Your Turn

In your group, decide how to replace this scatterplot with one that draws boxplots? Try out your best guess.

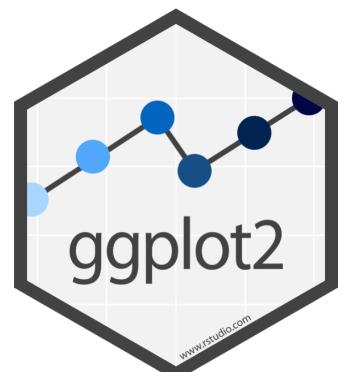


```
ggplot(mpg) + geom_point(aes(class, hwy))
```





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



# Multiple layers

R

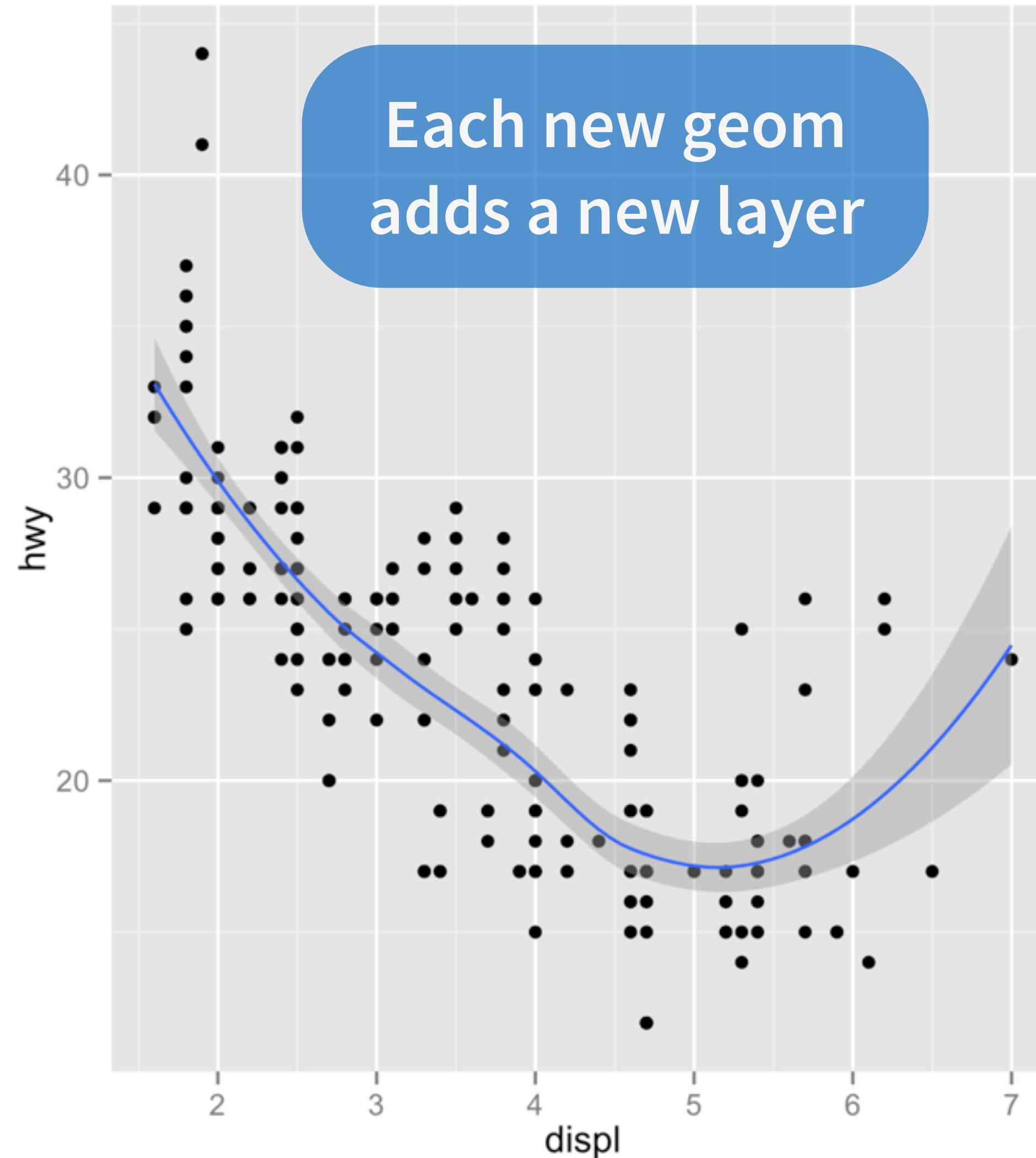
# Your Turn

In your group, predict what this code will do.

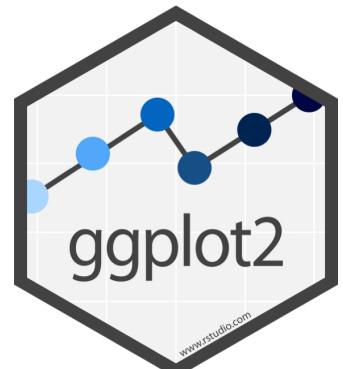
Then run it.

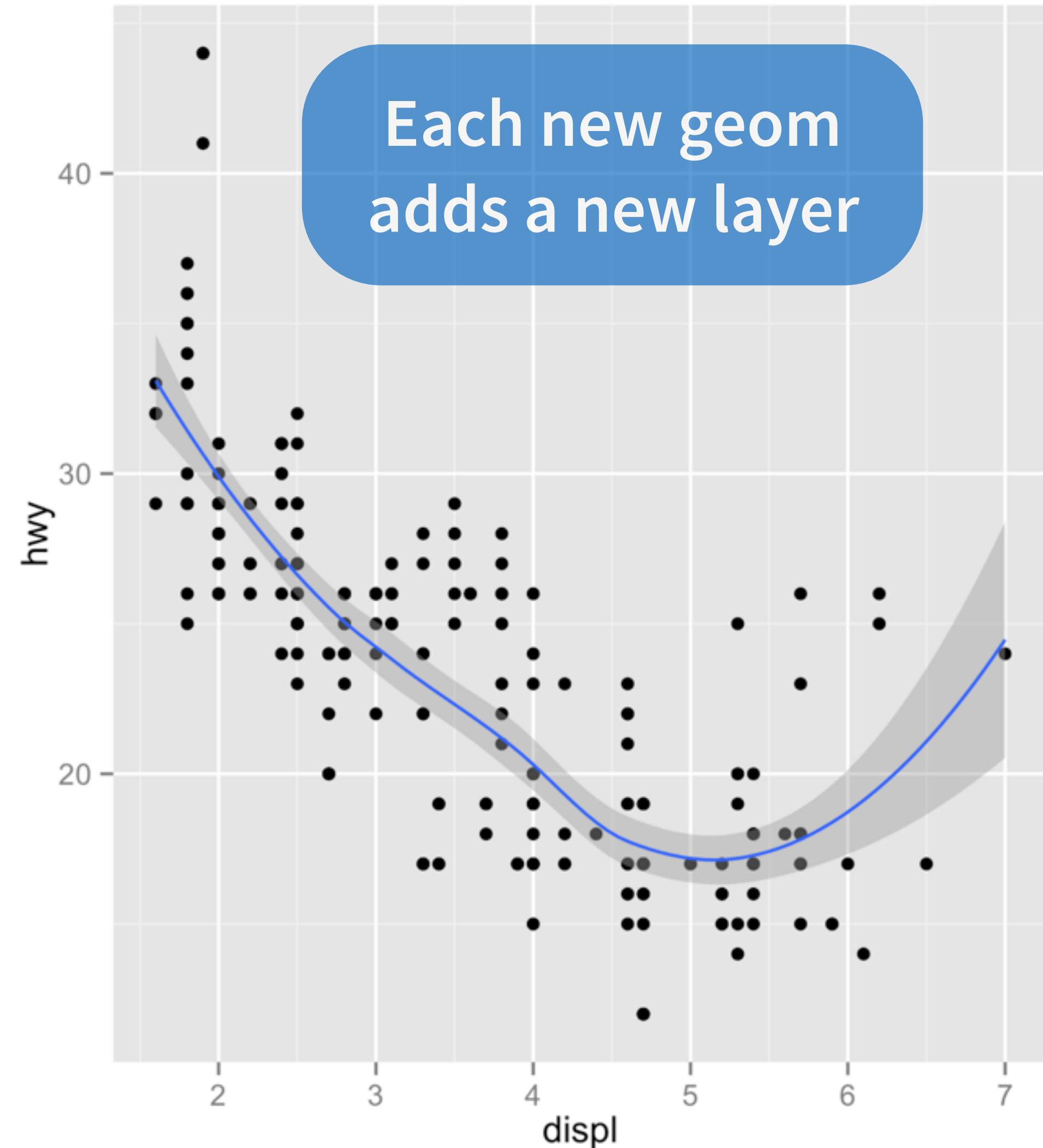
```
ggplot(mpg) +  
  geom_point(aes(displ, hwy)) +  
  geom_smooth(aes(displ, hwy))
```



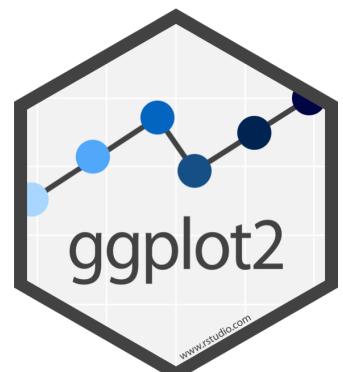


```
ggplot(mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  geom_smooth(mapping = aes(x = displ, y = hwy))
```



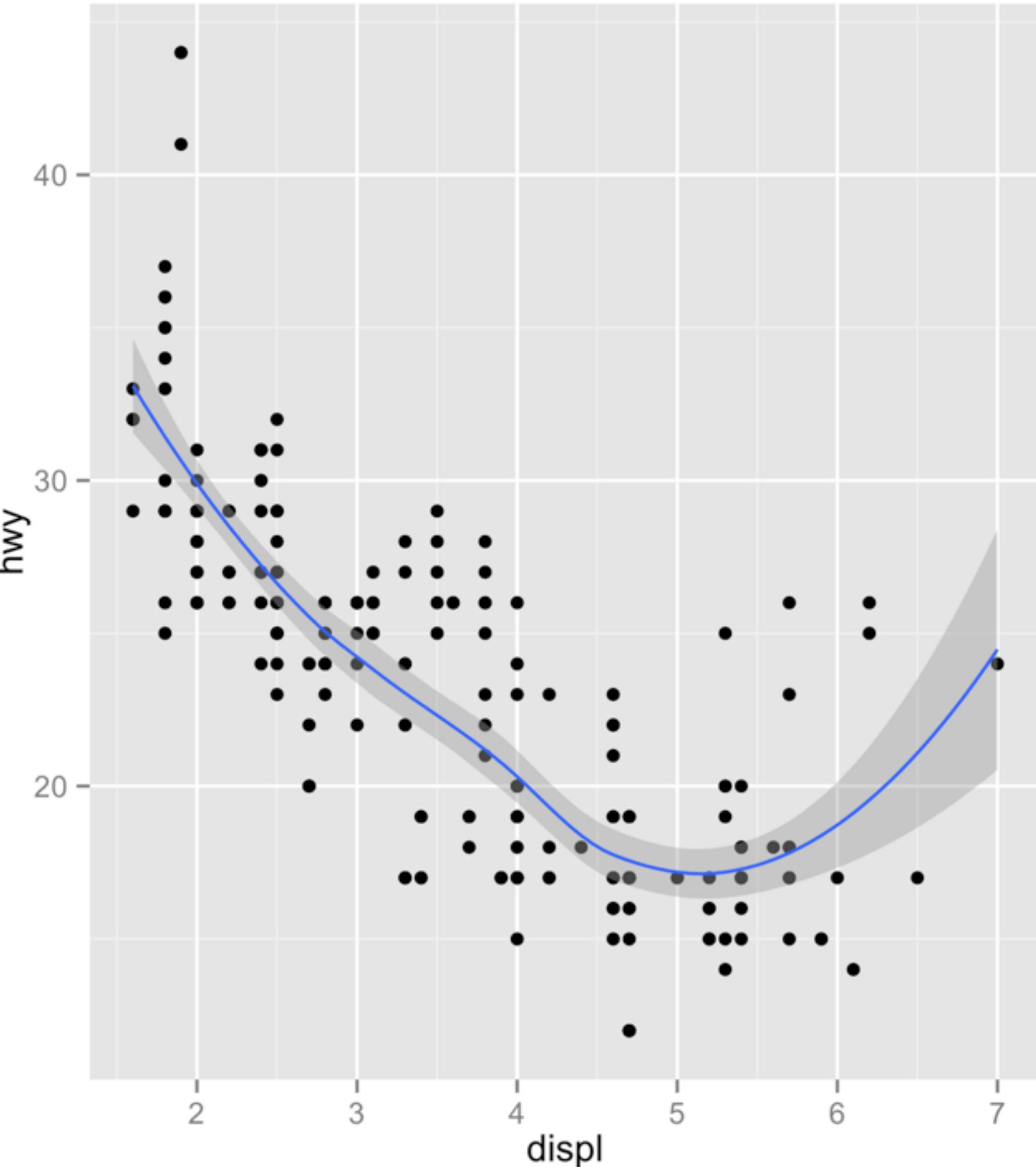


```
ggplot(mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  geom_smooth(mapping = aes(x = displ, y = hwy))
```



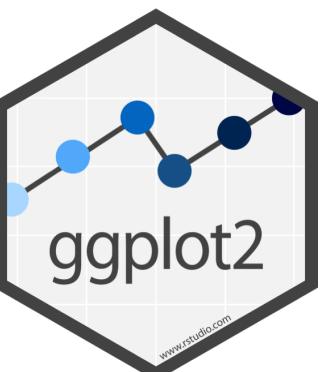
# global vs. local

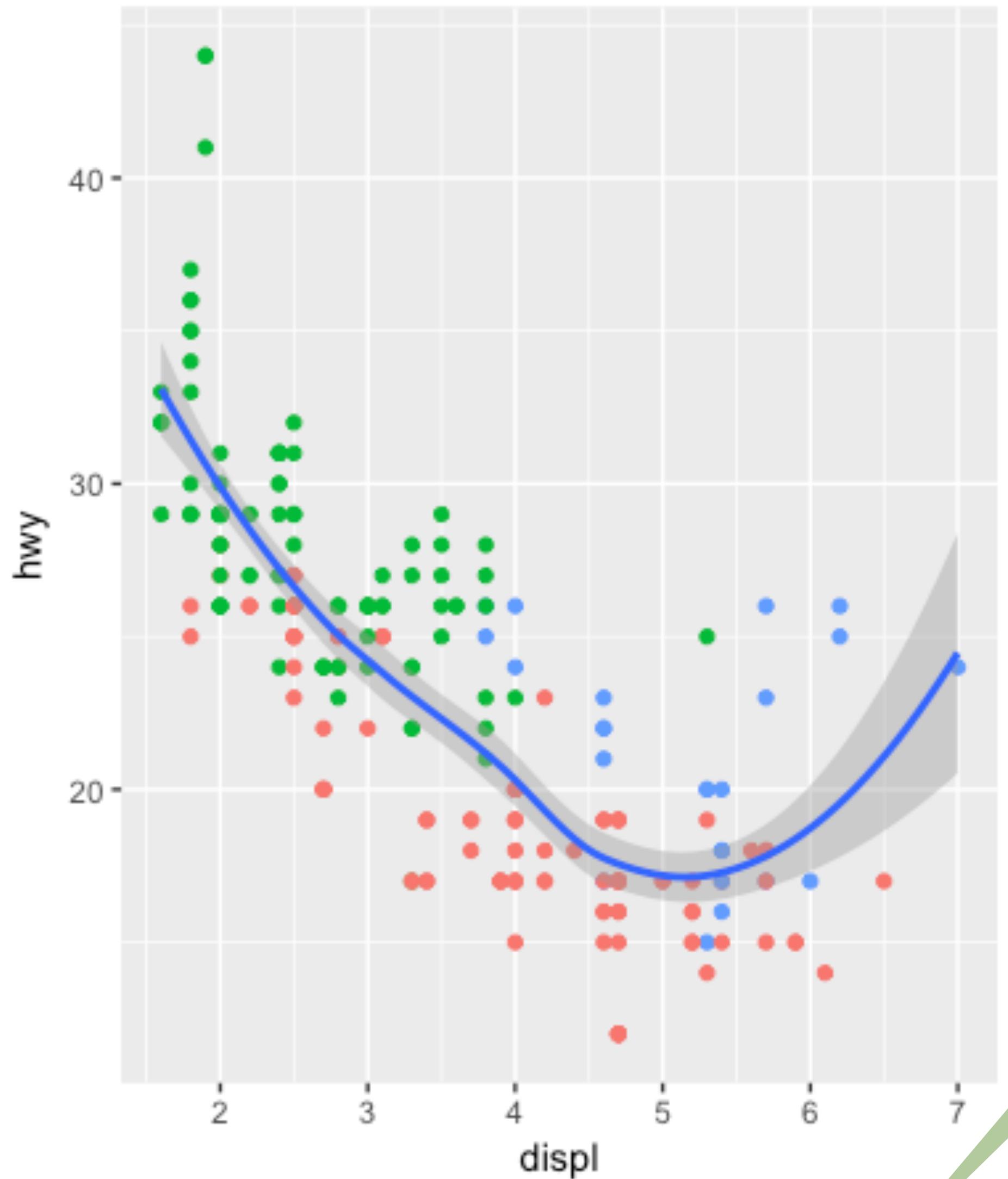
R



```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_point() +  
  geom_smooth()
```

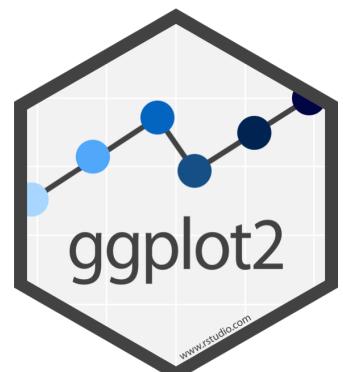
Mappings (and data)  
that appear in ggplot()  
will apply globally to  
every layer





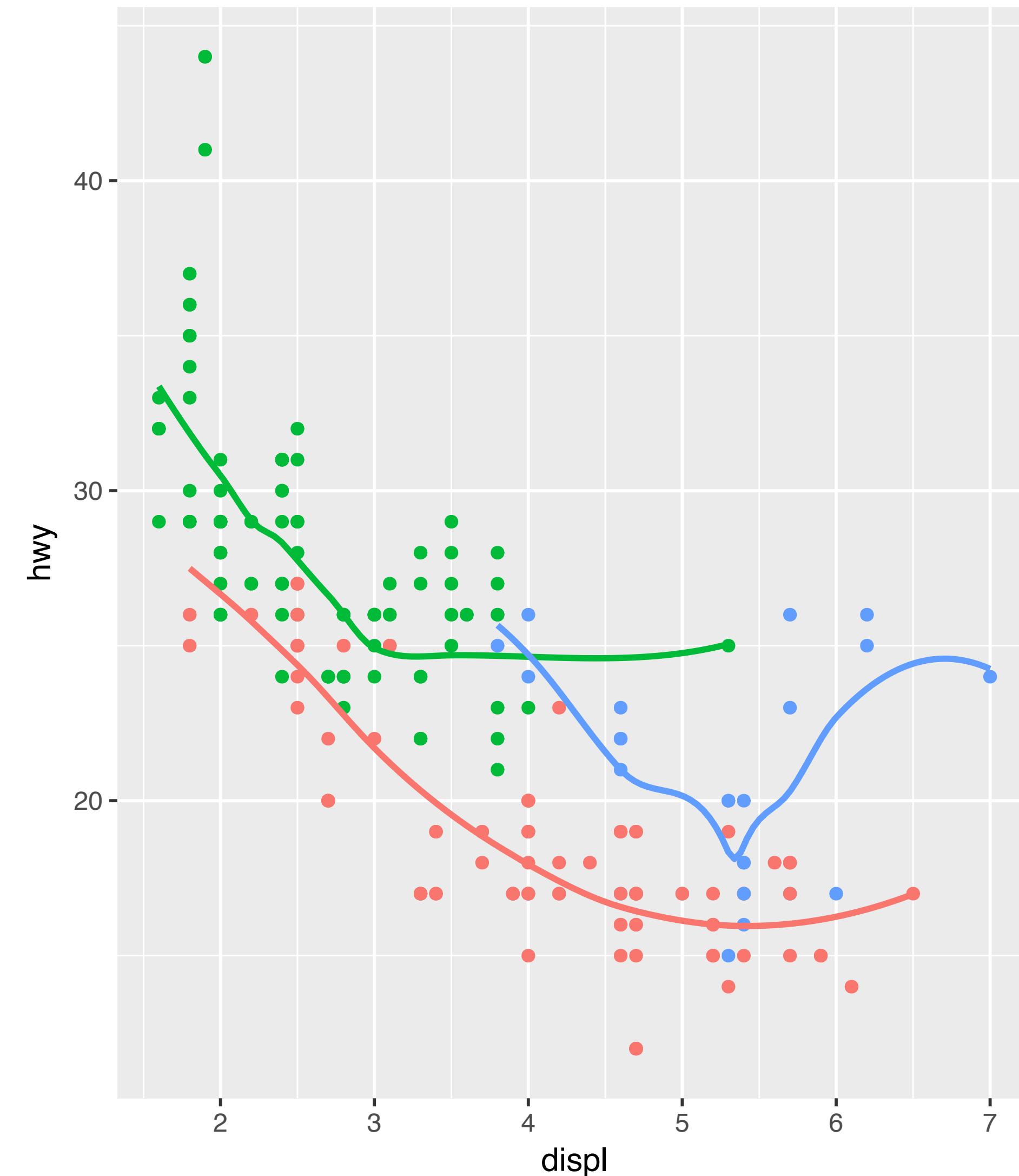
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_point(mapping = aes(color = drv)) +  
  geom_smooth()
```

Mappings (and data) that appear in a `geom_` function will add to or override the global mappings for that layer only



# Grouping cases

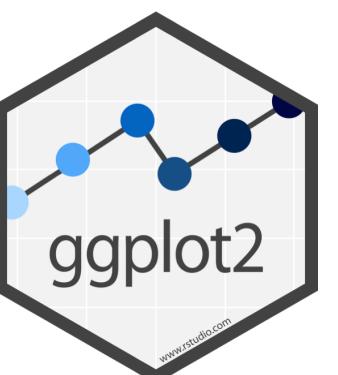
R

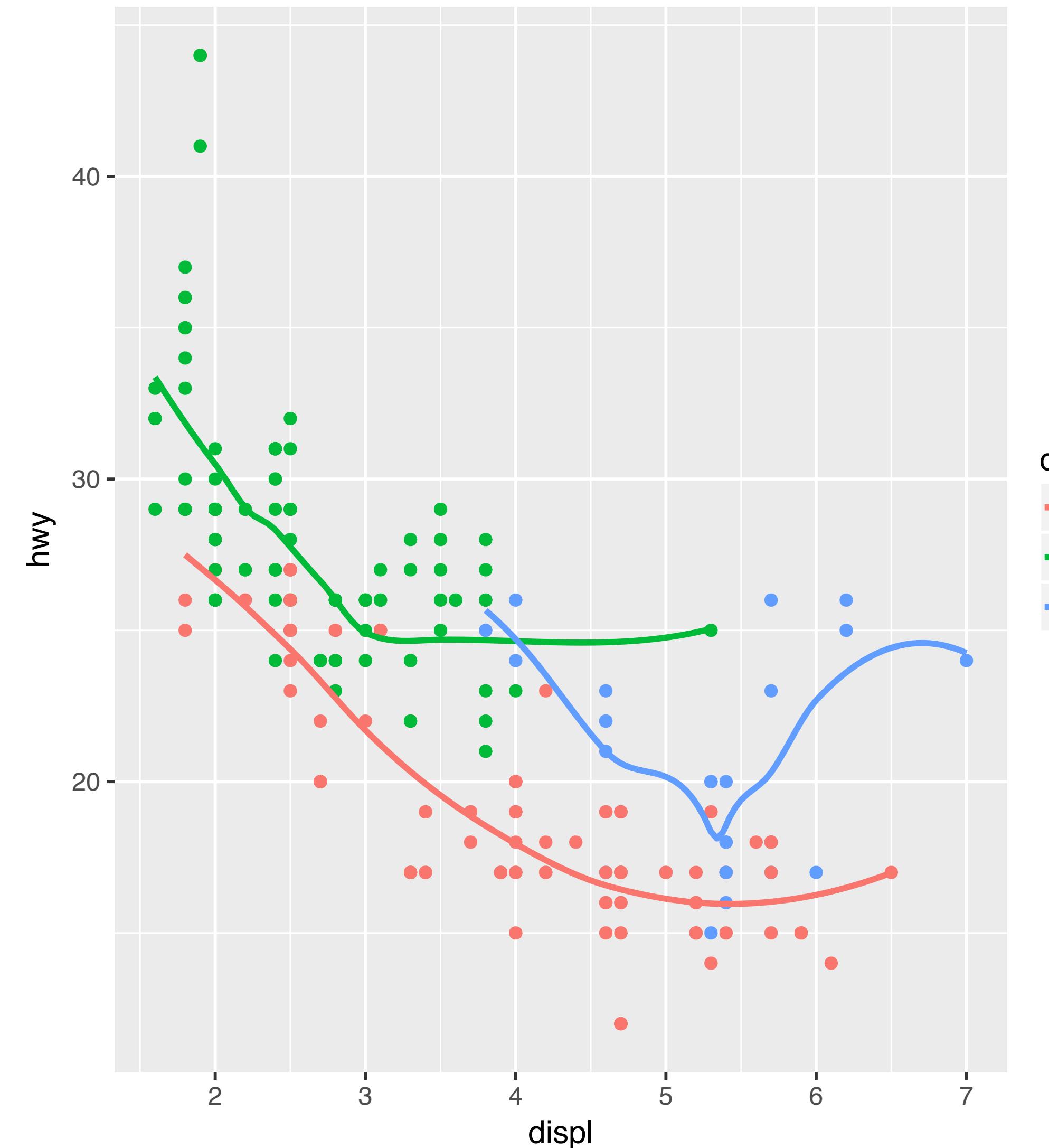


```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_point(mapping = aes(color = drv)) +  
  geom_smooth(mapping = aes(color = drv), se = FALSE)
```

Automatically draws a single line for each group implied by color.

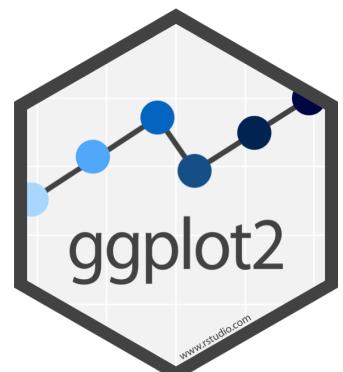
This occurs for other "monolithic" geoms as well.

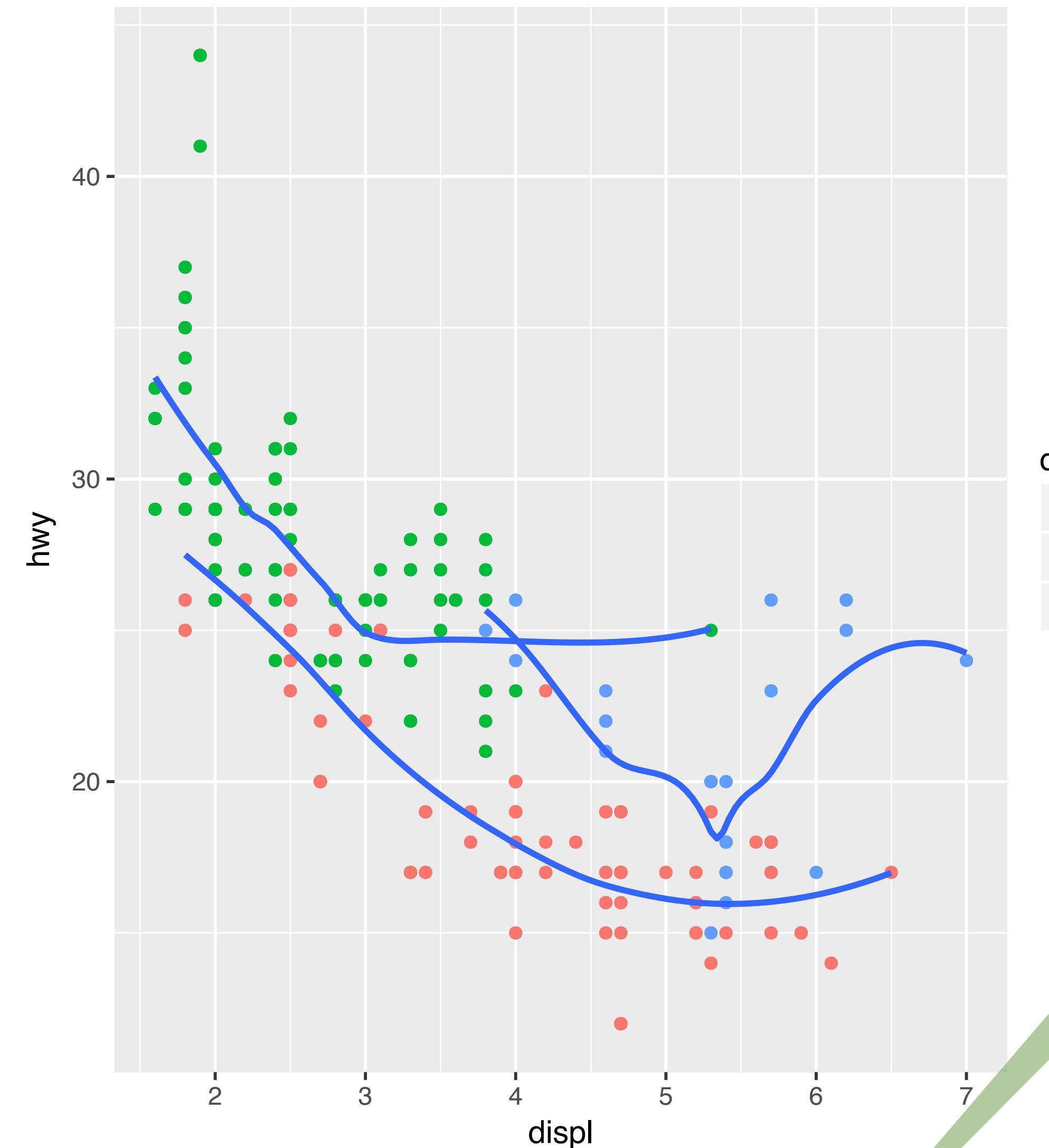




Standard  
error bars

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_point(mapping = aes(color = drv)) +  
  geom_smooth(mapping = aes(color = drv), se = FALSE)
```



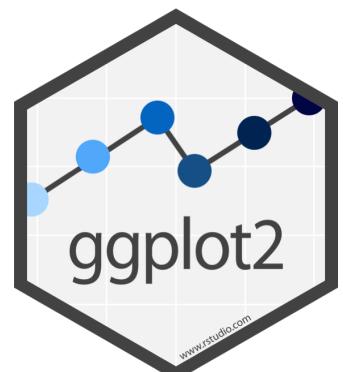


drv

- 4
- f
- r

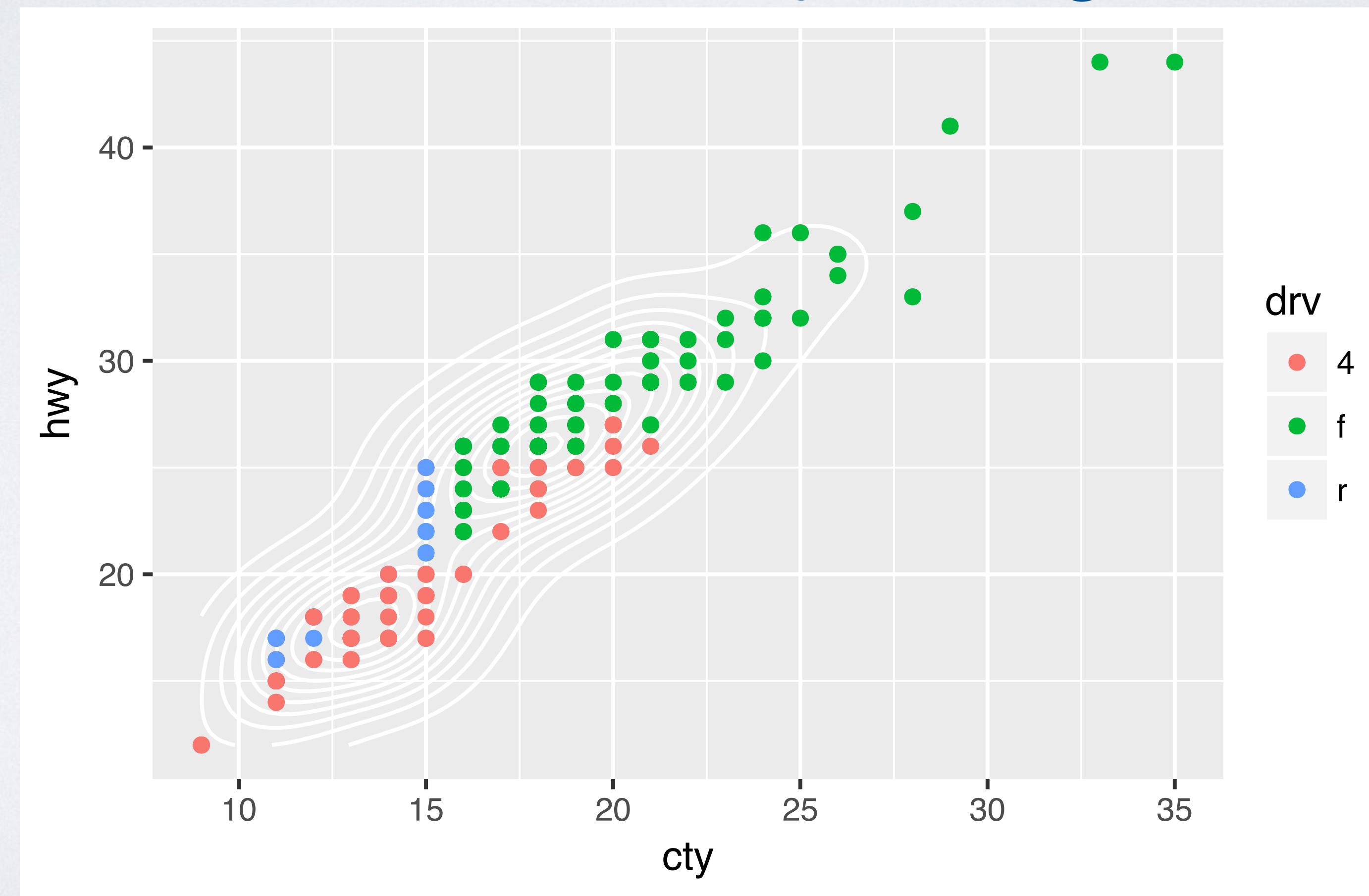
Groups without  
adding a visible  
aesthetic (or legend)

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_point(mapping = aes(color = drv)) +  
  geom_smooth(mapping = aes(group = drv), se = FALSE)
```

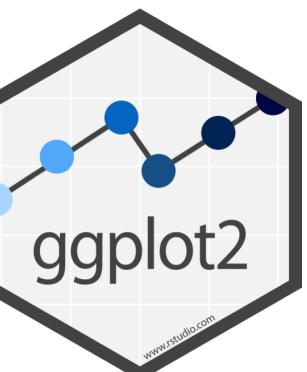


# Your Turn

Create this plot: a white density 2d overlaid with colored points.  
Consult the cheat sheet, and then your neighbor.



```
ggplot(data = mpg,  
       mapping = aes(x = cty, y = hwy, color = drv)) +  
       geom_density_2d(color = "white") +  
       geom_point()
```



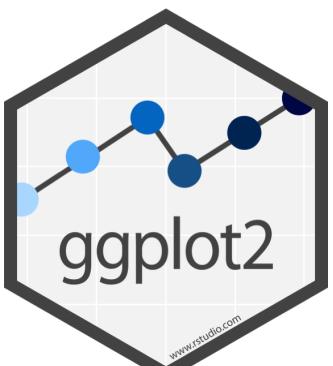
# Visualizing summaries (stats)

# iamonds

Price and description of 54,000 diamonds

```
View(diamonds)
```

Capital "V"



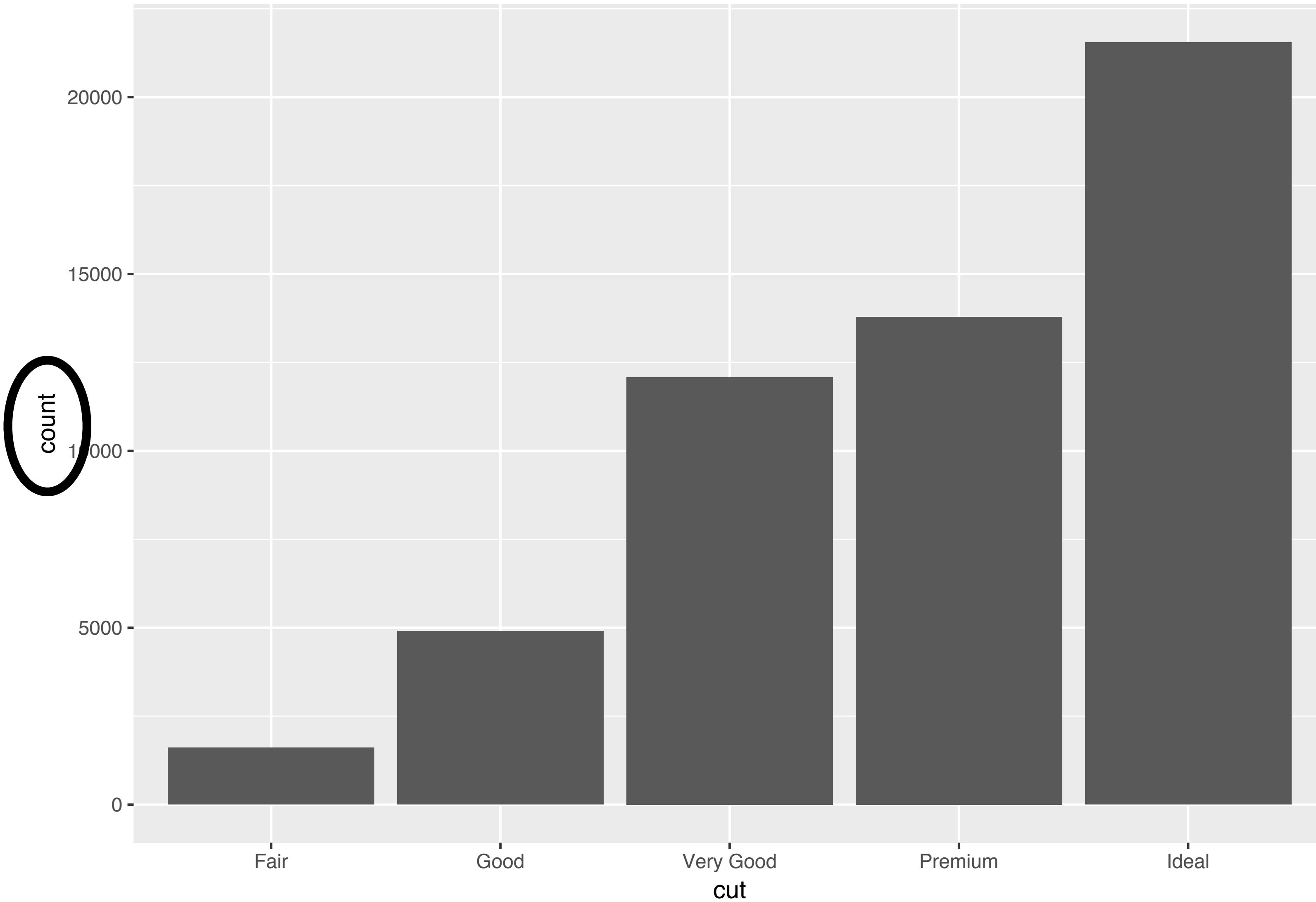
# Your Turn

If I run this code, what will ggplot2 put on the y axis?

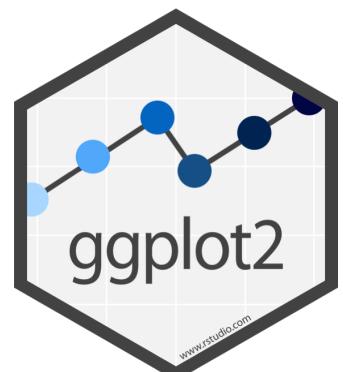
Try it and see.

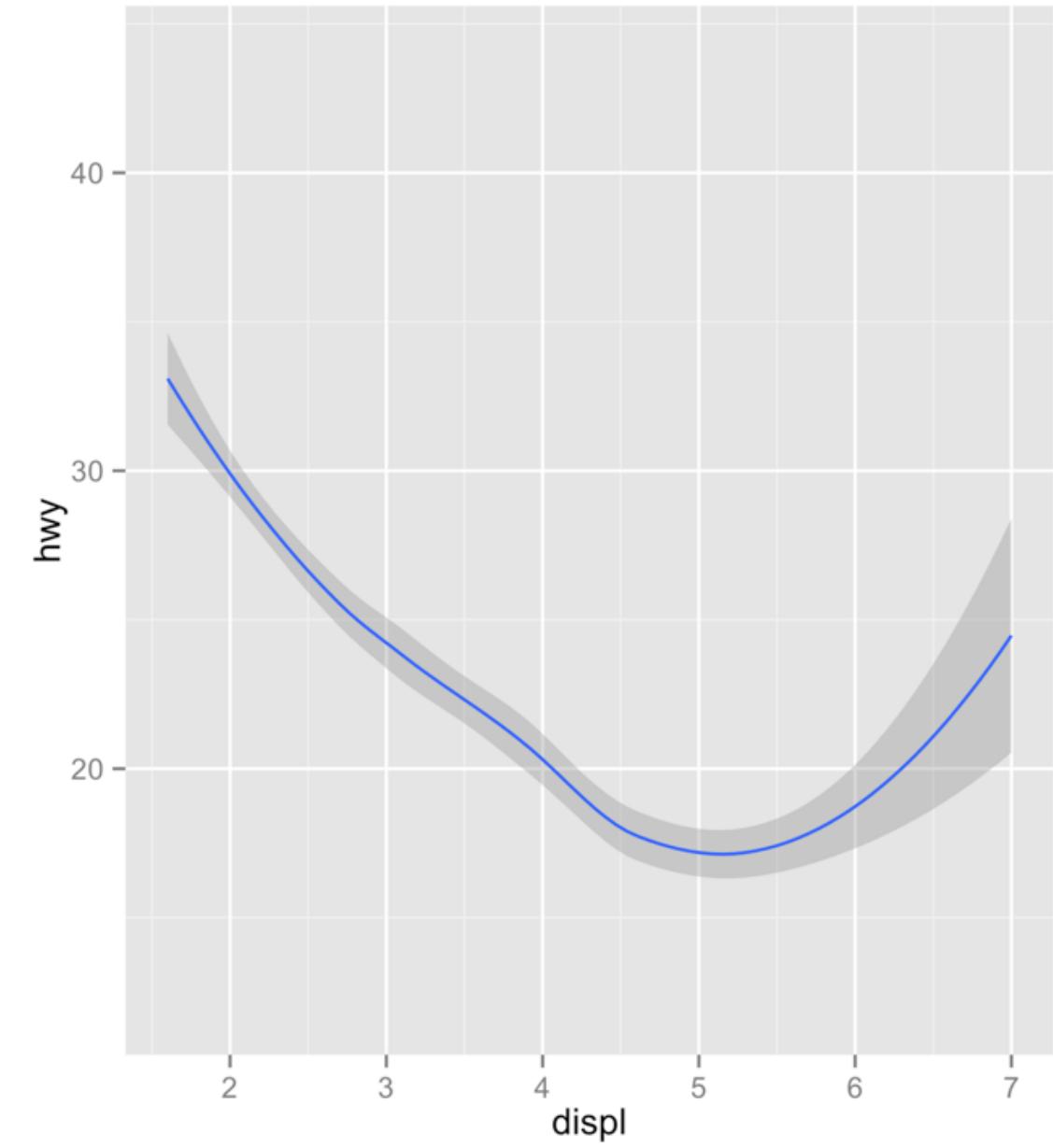
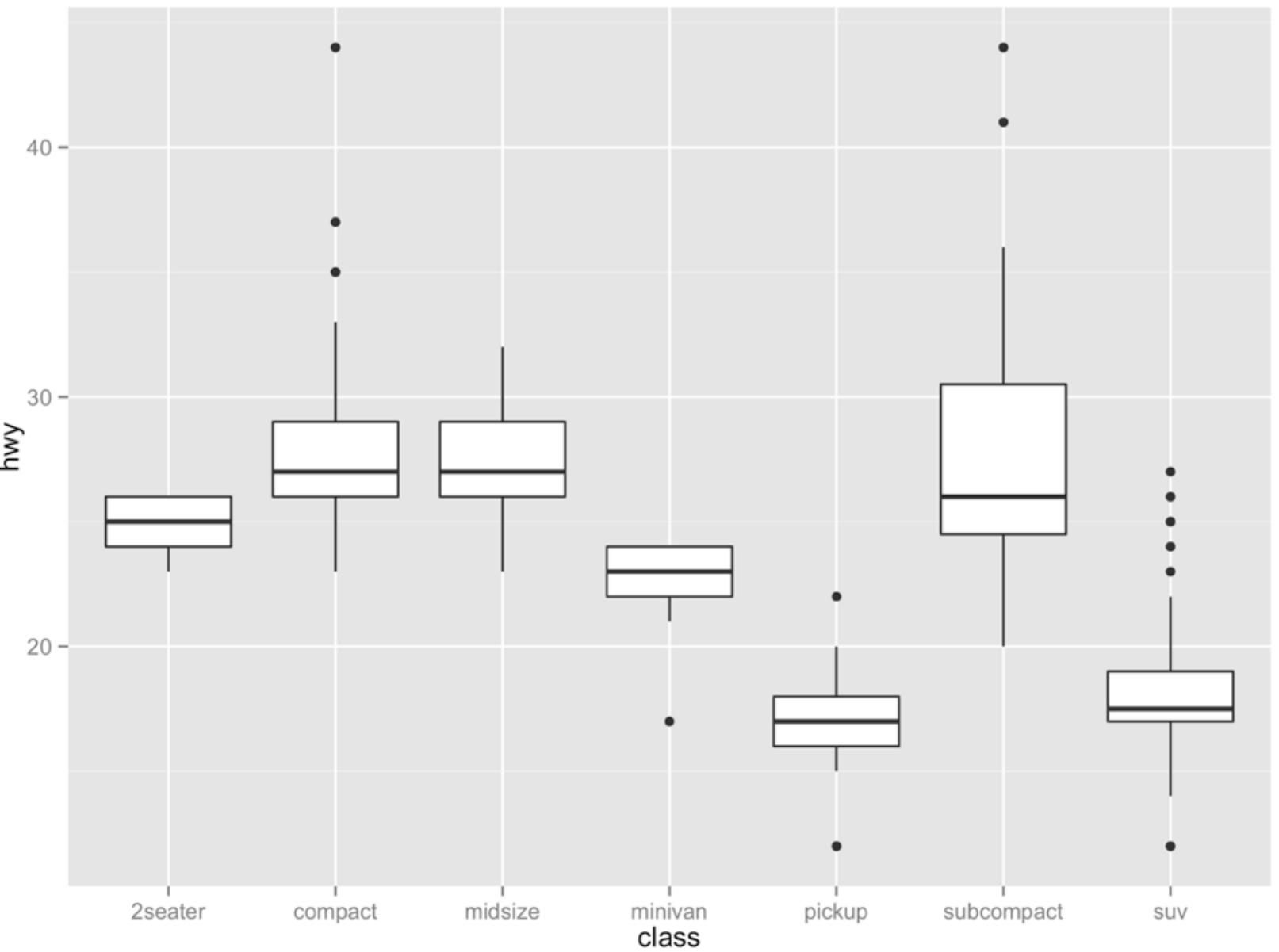
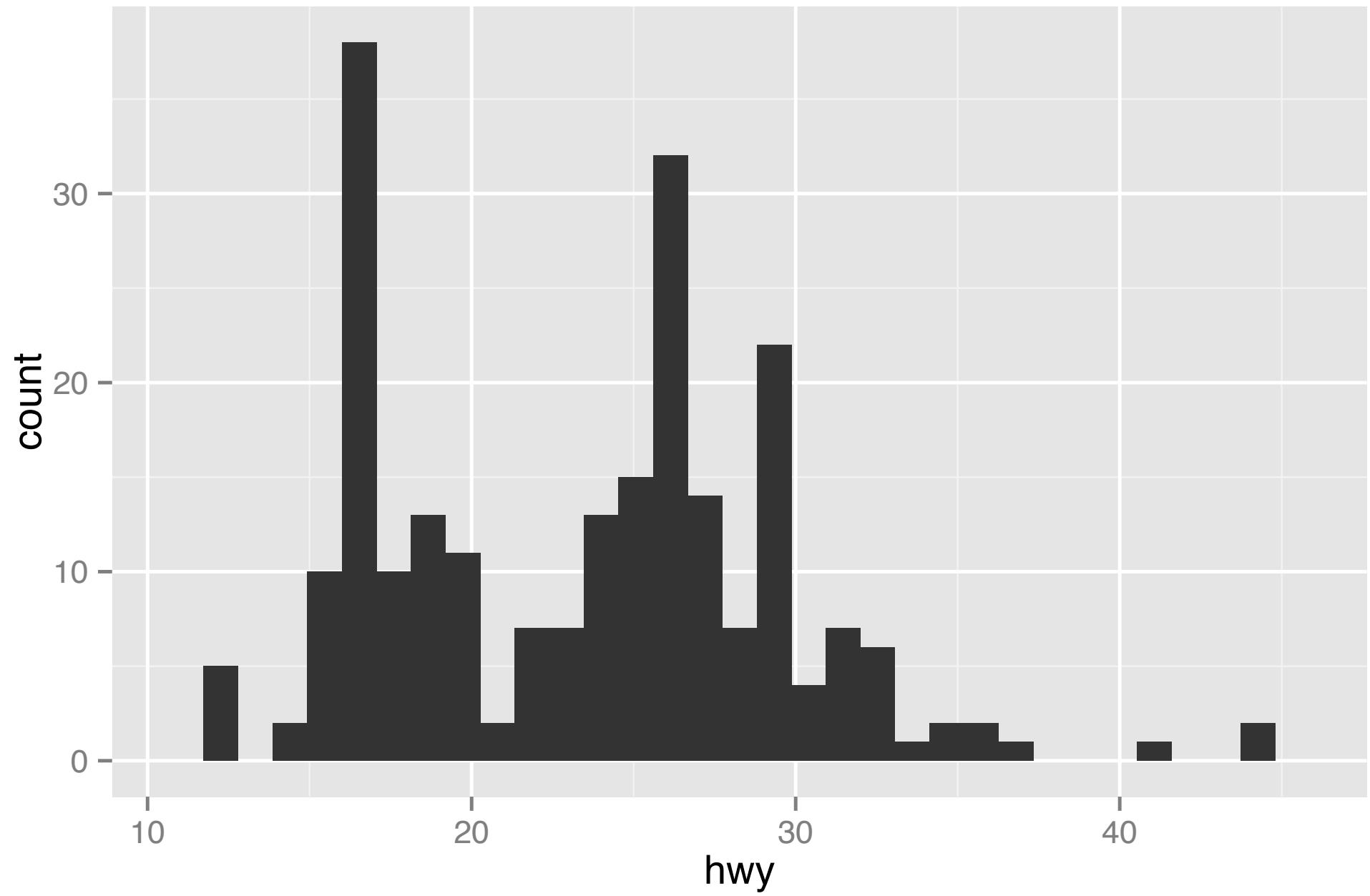
```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(x = cut))
```





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





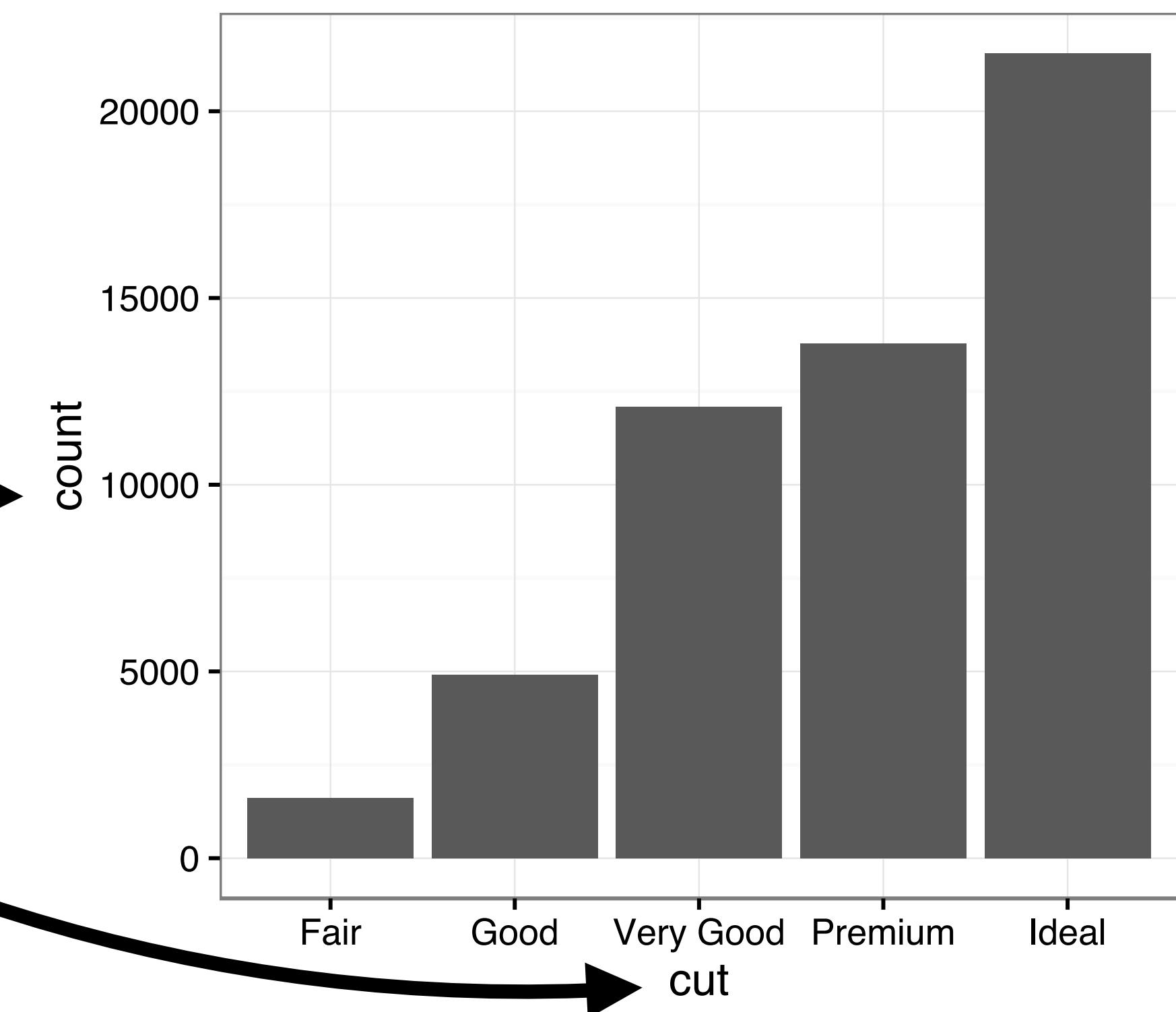
# stats

Many geoms display not raw data, but values derived from the data.  
The process that calculates the values is a **stat**.

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Fair	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Fair	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...

→ stat\_count()

cut	count	prop
Fair	1610	1
Good	4906	1
Very Good	12082	1
Premium	13791	1
Ideal	21551	1



# stat\_ functions

Each derives a new data set of values to plot from the original data.



**c + stat\_bin(binwidth = 1, origin = 10)** **1D distributions**

x, y | ..count.., ..ncount.., ..density.., ..ndensity..

**c + stat\_count(width = 1) x, y, | ..count.., ..prop..**

**c + stat\_density(adjust = 1, kernel = "gaussian")**

x, y, | ..count.., ..density.., ..scaled..

**e + stat\_bin\_2d(bins = 30, drop = T)** **2D distributions**

x, y, fill | ..count.., ..density..

**e + stat\_bin\_hex(bins=30) x, y, fill | ..count.., ..density..**

**e + stat\_density\_2d(contour = TRUE, n = 100)**

x, y, color, size | ..level..

**e + stat\_ellipse(level = 0.95, segments = 51, type = "t")**

**l + stat\_contour(aes(z = z)) x, y, z, order | ..level..**

**l + stat\_summary\_hex(aes(z = z), bins = 30, fun = max)**

x, y, z, fill | ..value..

**l + stat\_summary\_2d(aes(z = z), bins = 30, fun = mean)**

x, y, z, fill | ..value..

**3 Variables**

**f + stat\_boxplot(coef = 1.5)**

**Comparisons**

x, y | ..lower.., ..middle.., ..upper.., ..width.., ..ymin.., ..ymax..

**f + stat\_ydensity(kernel = "gaussian", scale = "area")**

x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

**e + stat\_ecdf(n = 40) x, y | ..x.., ..y..**

**Functions**

**e + stat\_quantile(quantiles = c(0.1, 0.9),**

formula = y ~ log(x), method = "rq") x, y | ..quantile..

**e + stat\_smooth(method = "lm", formula = y ~ x,**

se=T, level=0.95) x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

**ggplot() + stat\_function(aes(x = -3:3), n = 99,**

fun = dnorm, args = list(sd=0.5)) x | ..x.., ..y..

**e + stat\_identity(na.rm = TRUE)**

**ggplot() + stat\_qq(aes(sample=1:100), dist = qt,**

dparam=list(df=5)) sample, x, y | ..sample.., ..theoretical..

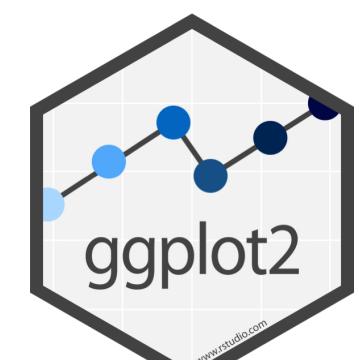
**e + stat\_sum() x, y, size | ..n.., ..prop..**

**e + stat\_summary(fun.data = "mean\_cl\_boot")**

**h + stat\_summary\_bin(fun.y = "mean", geom = "bar")**

**e + stat\_unique()**

**General Purpose**

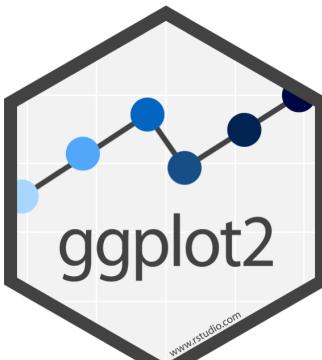


# A ggplot2 template

Make any plot by filling in the parameters of this template

```
ggplot(data = <DATA>) +  
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),  
                    stat = <STAT>) +  
  <FACET_FUNCTION>
```

ggplot2 supplies useful defaults for  
everything except DATA, GEOM\_FUNCTION,  
and MAPPINGS

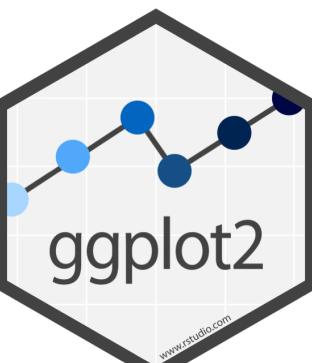


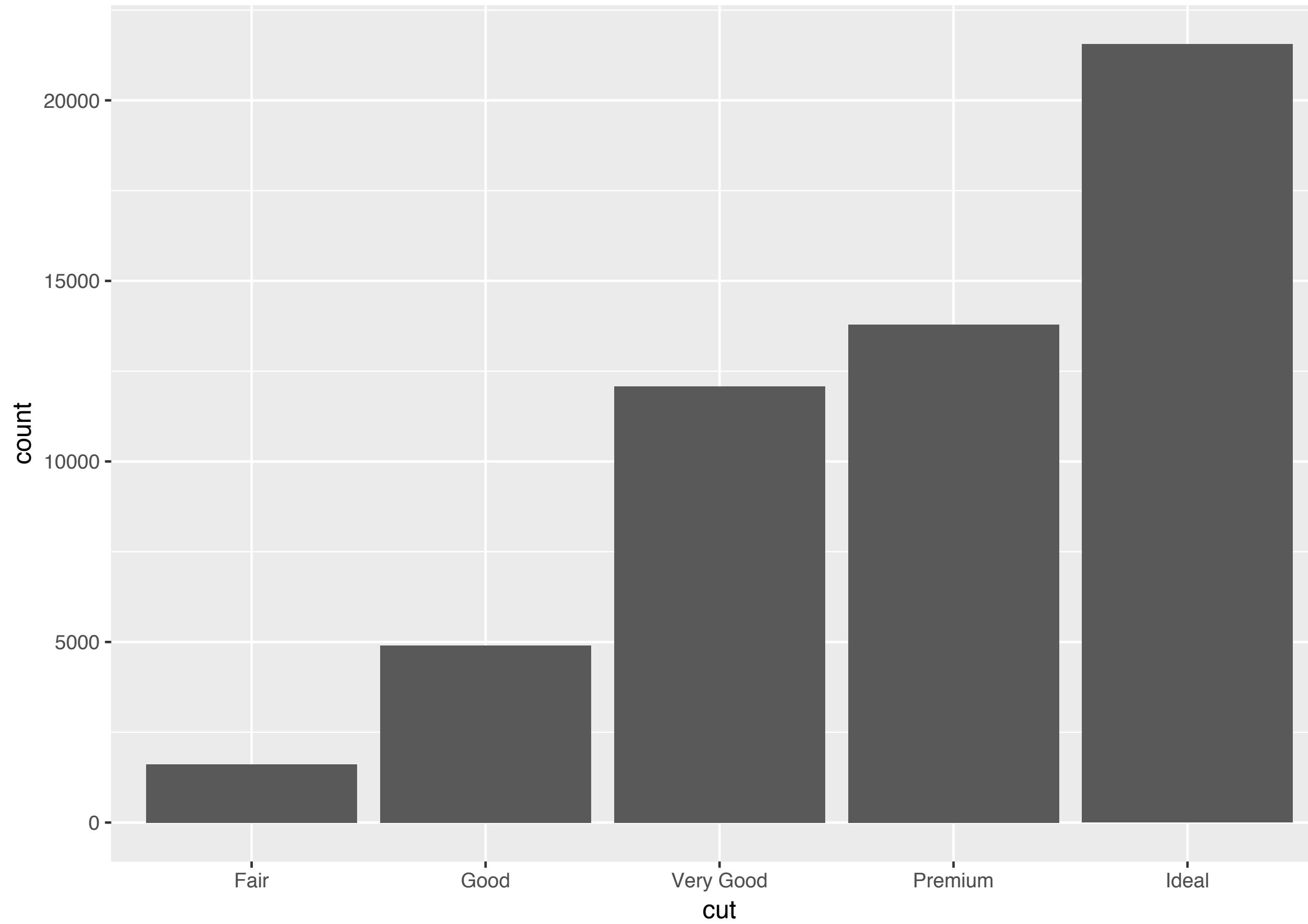
# A ggplot2 template

Make any plot by filling in the parameters of this template

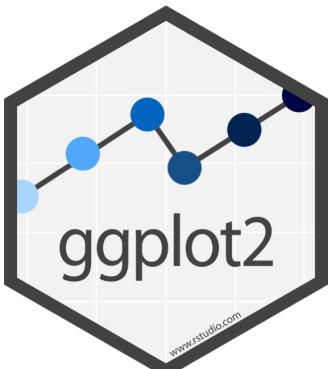
```
ggplot(data = <DATA>) +  
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),  
                    stat = <STAT>) +  
  <FACET_FUNCTION>
```

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



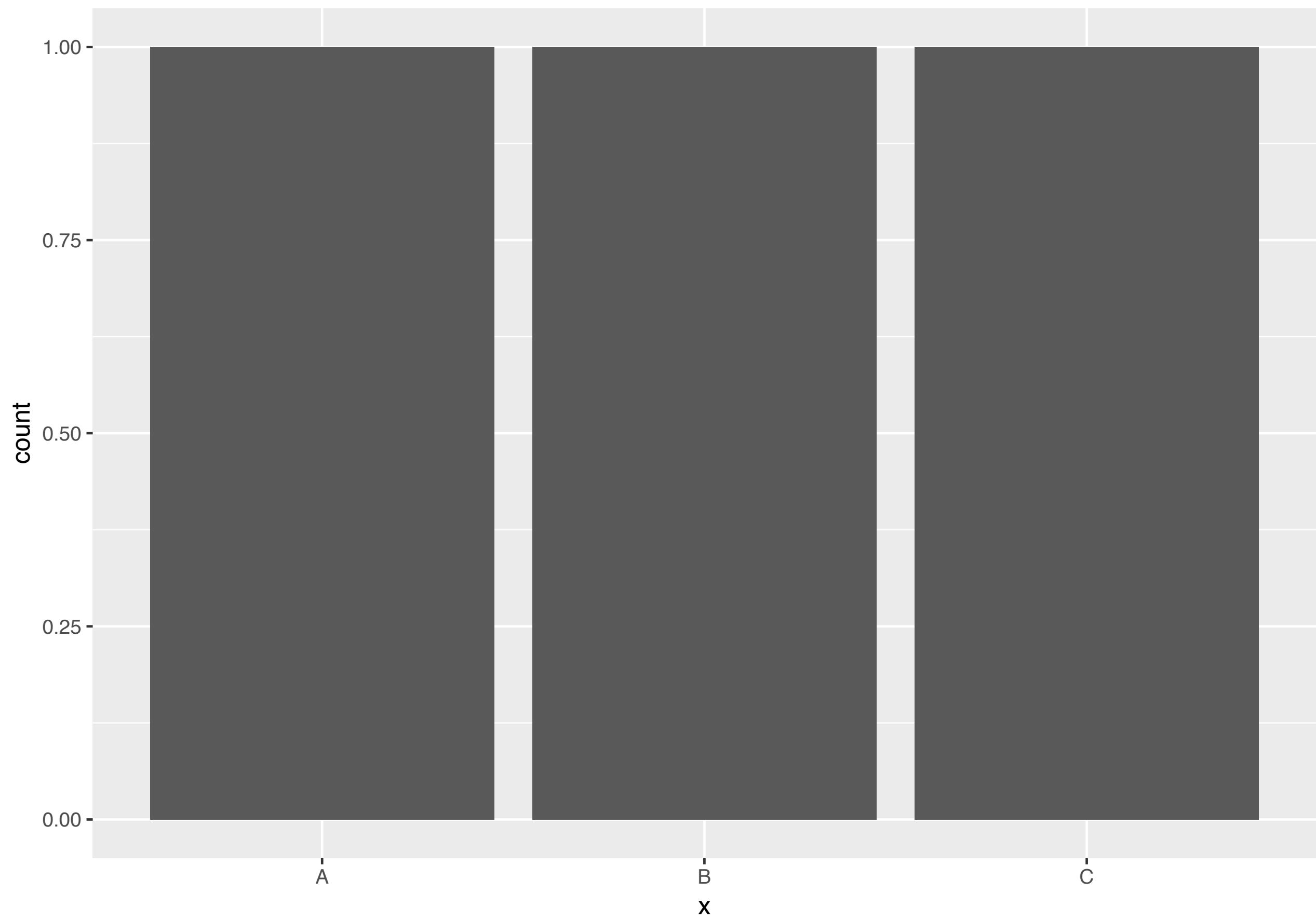


```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(x = cut),  
           stat = "count")
```

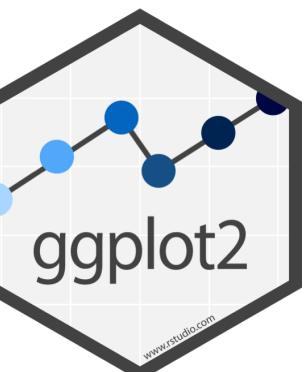


df

x	y
A	1
B	2
C	3

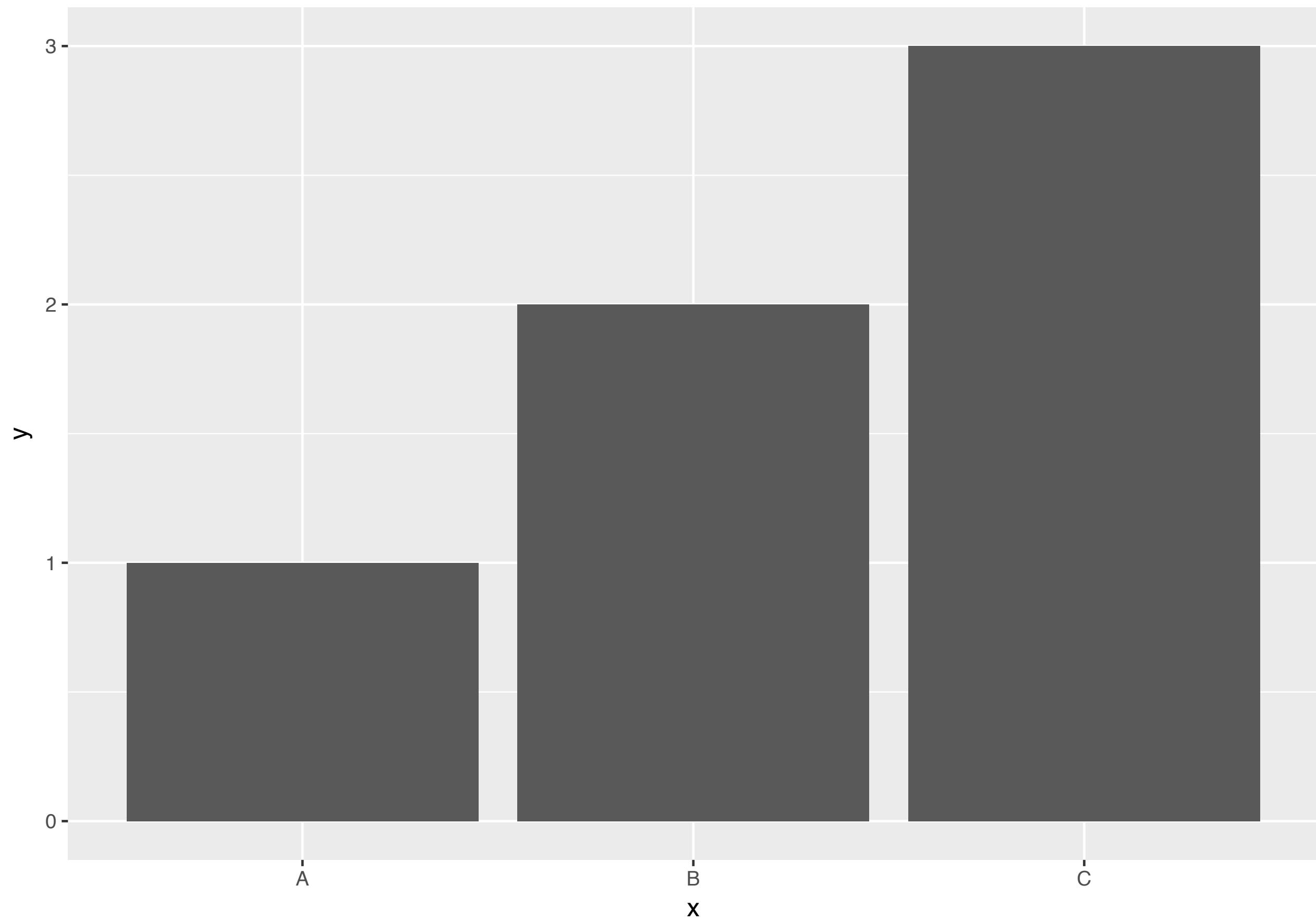


```
df <- data.frame(x = c("A", "B", "C"), y = 1:3)
ggplot(data = df) +
  geom_bar(mapping = aes(x = x), stat = "count")
```

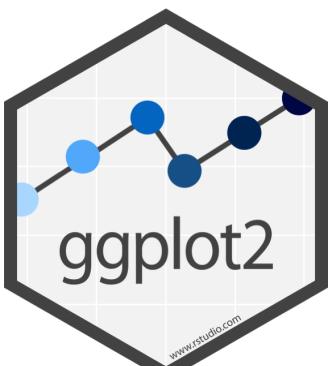


df

	x	y
A		1
B		2
C		3



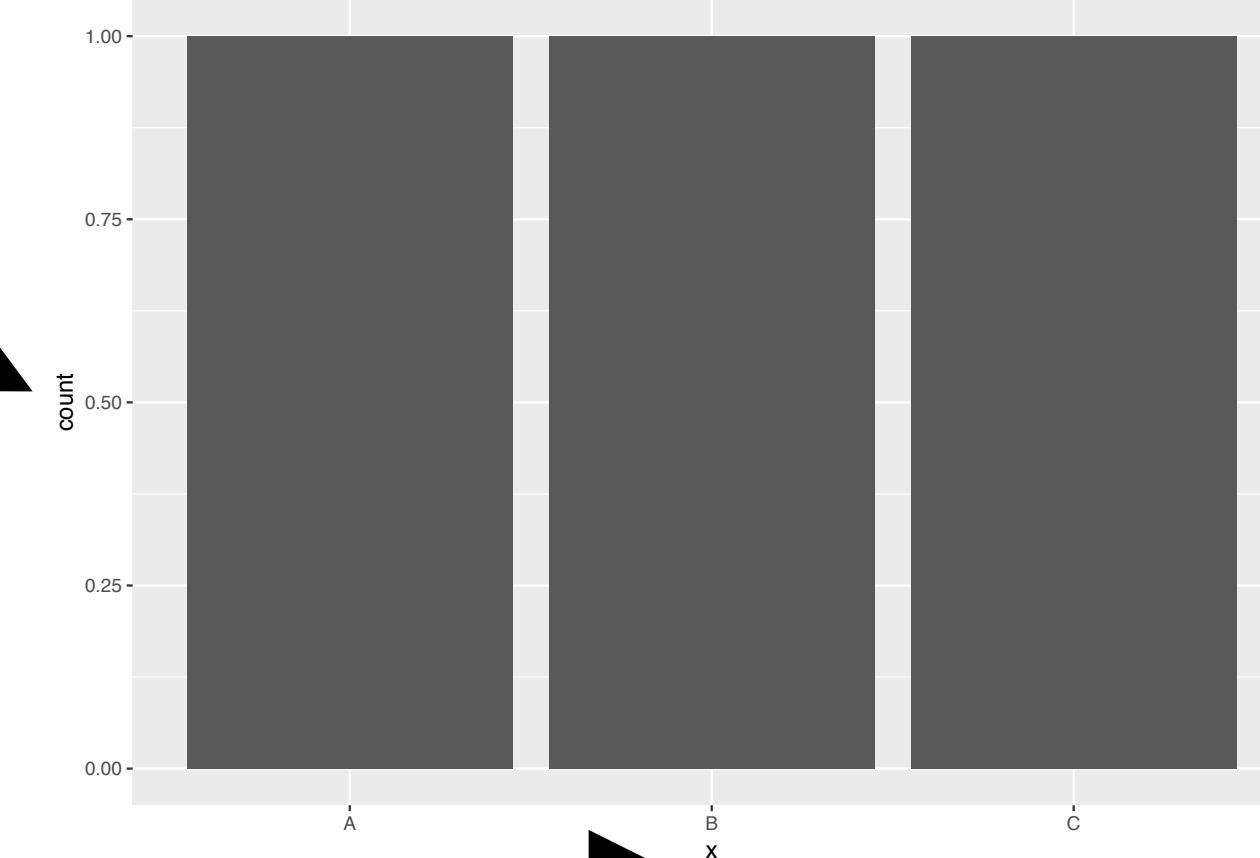
```
df <- data.frame(x = c("A", "B", "C"), y = 1:3)
ggplot(data = df) +
  geom_bar(mapping = aes(x = x, y = y), stat = "identity")
```



x	y
A	1
B	2
C	3

stat\_count()

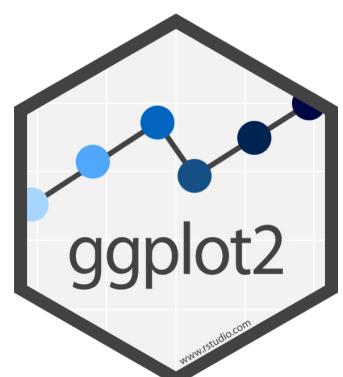
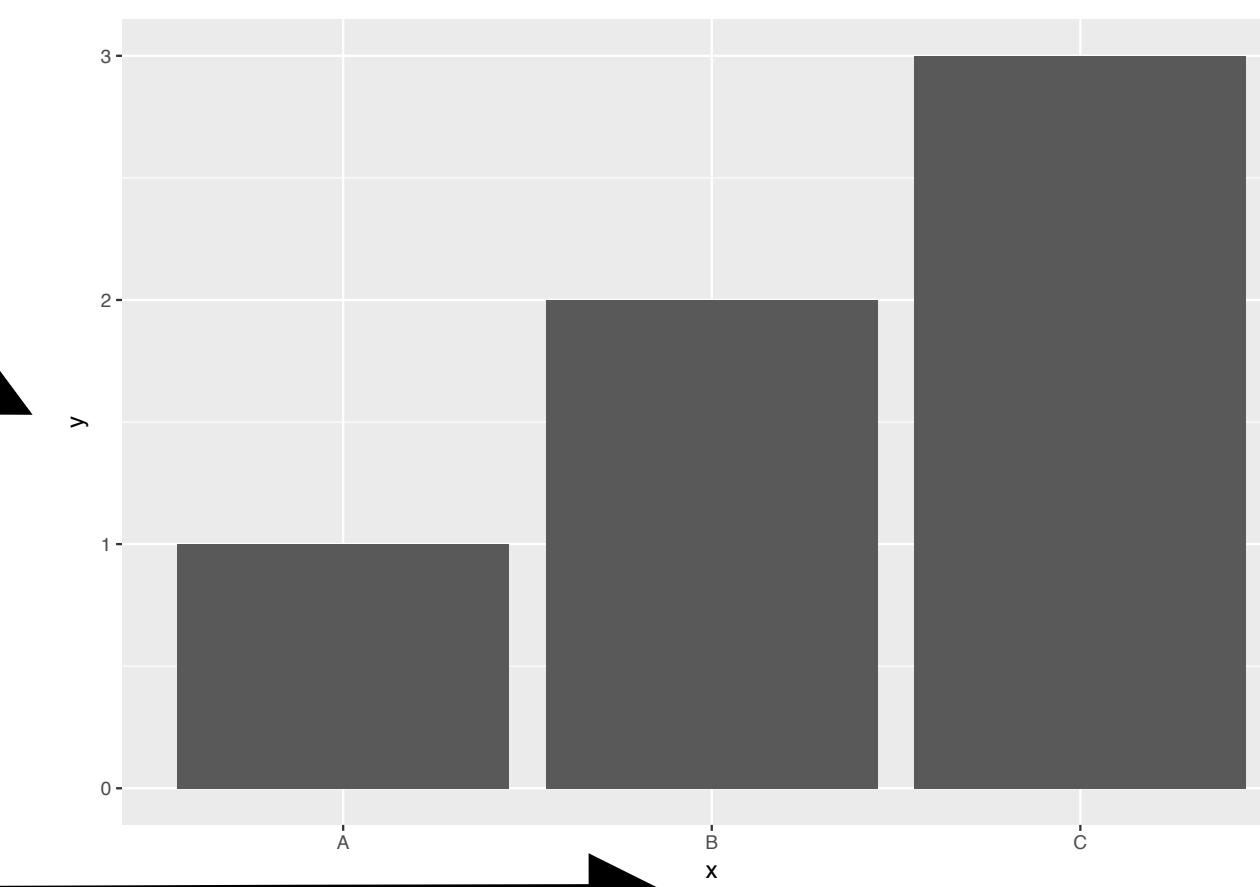
x	count
A	1
B	1
C	1



x	y
A	1
B	2
C	3

stat\_identity()

x	y
A	1
B	2
C	3



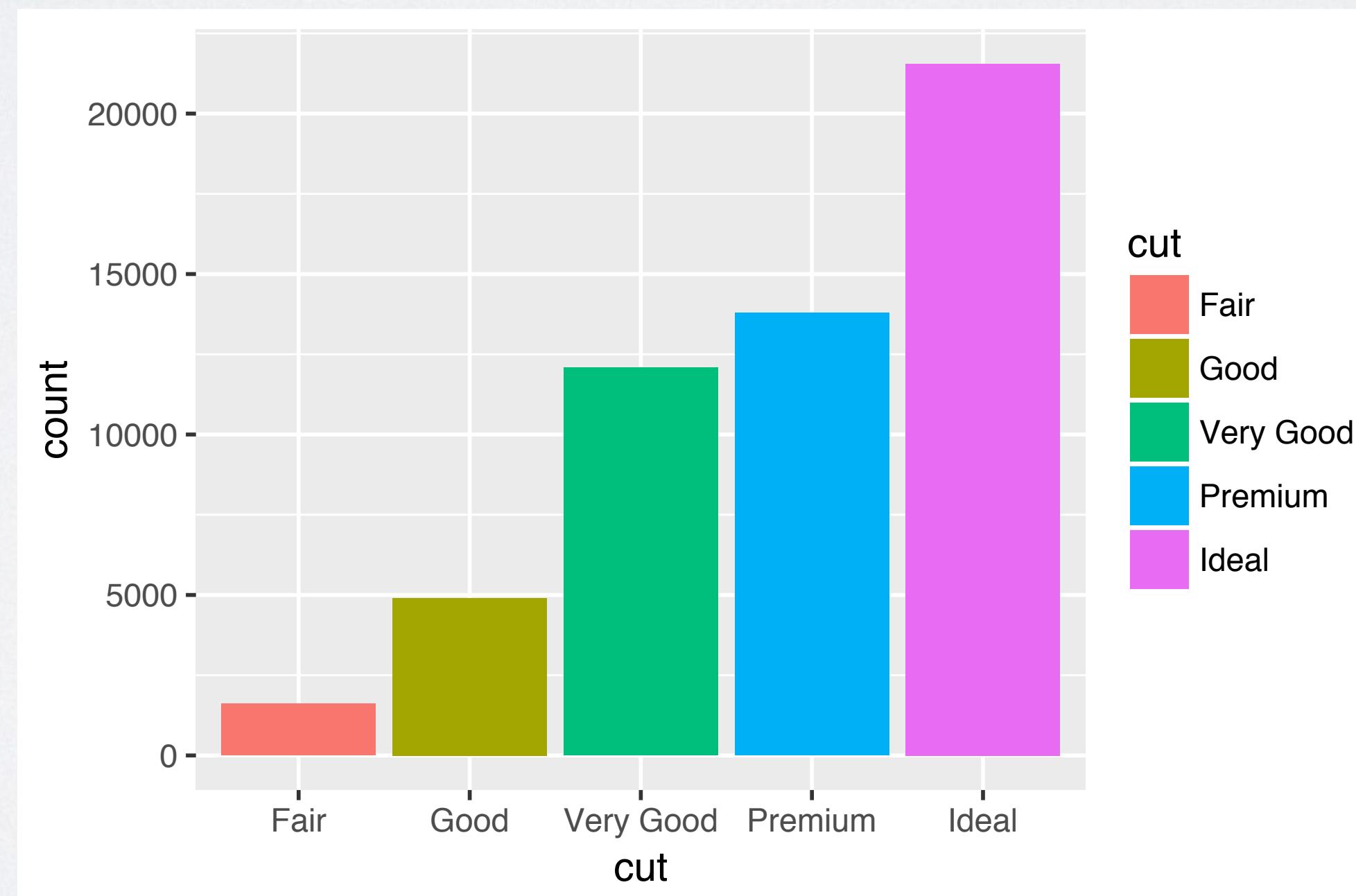
# Position adjustments



# Challenge

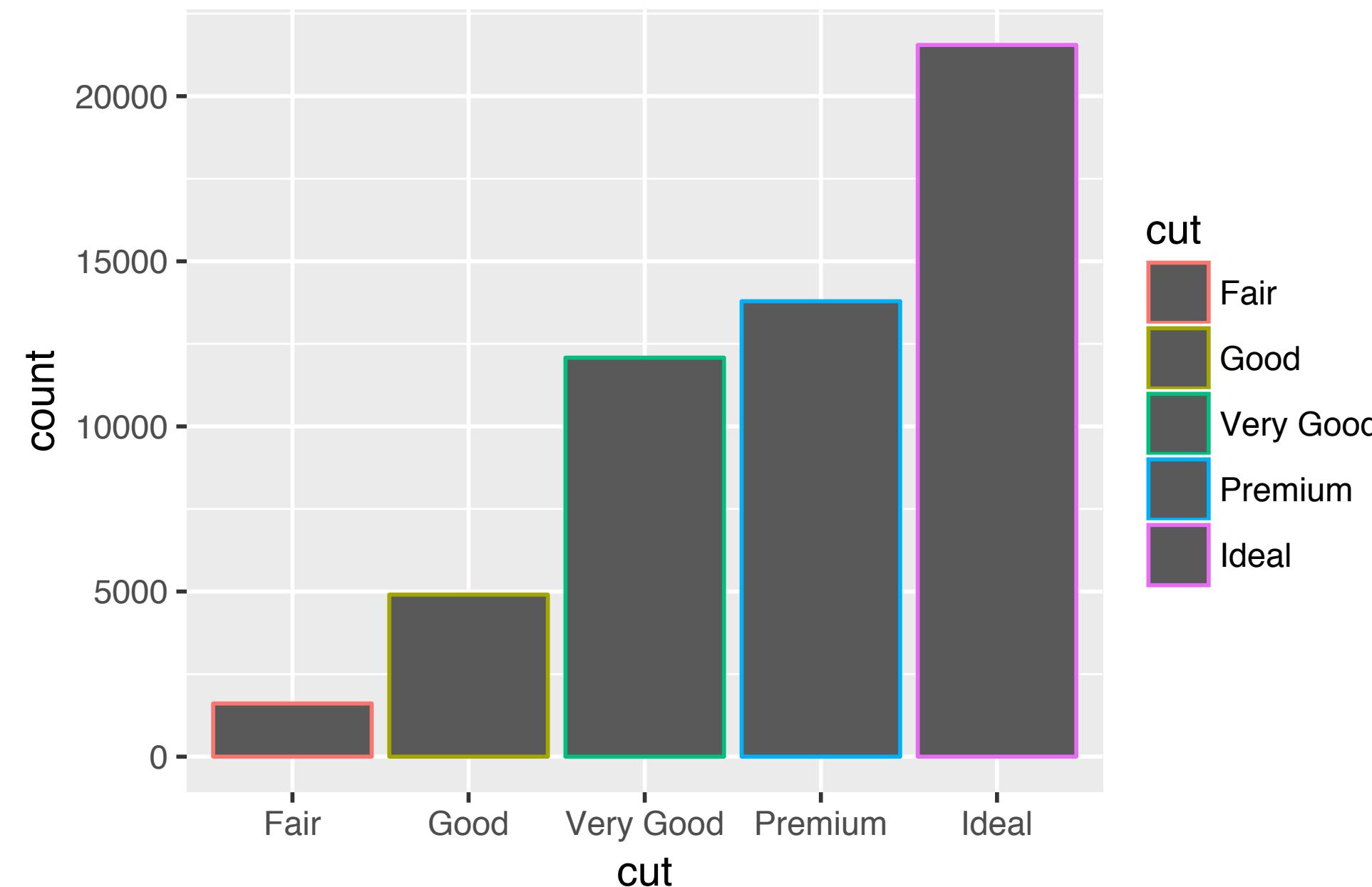
Can you alter this code to make this plot?

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

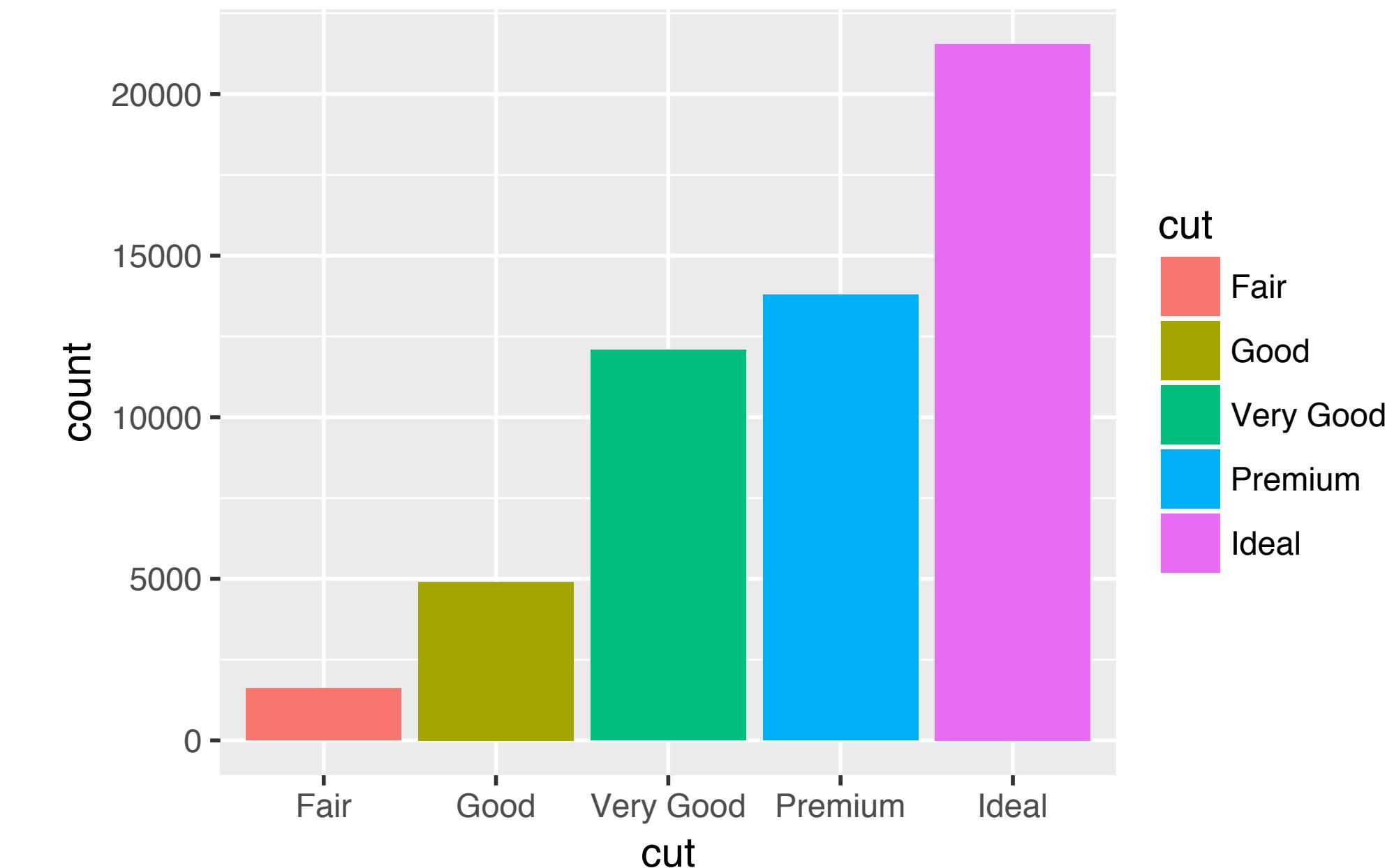


# fill

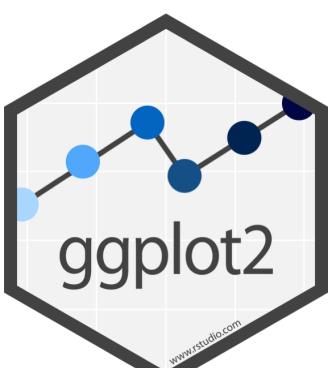
geoms that span an area have both a **color** and a **fill** aesthetic.



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



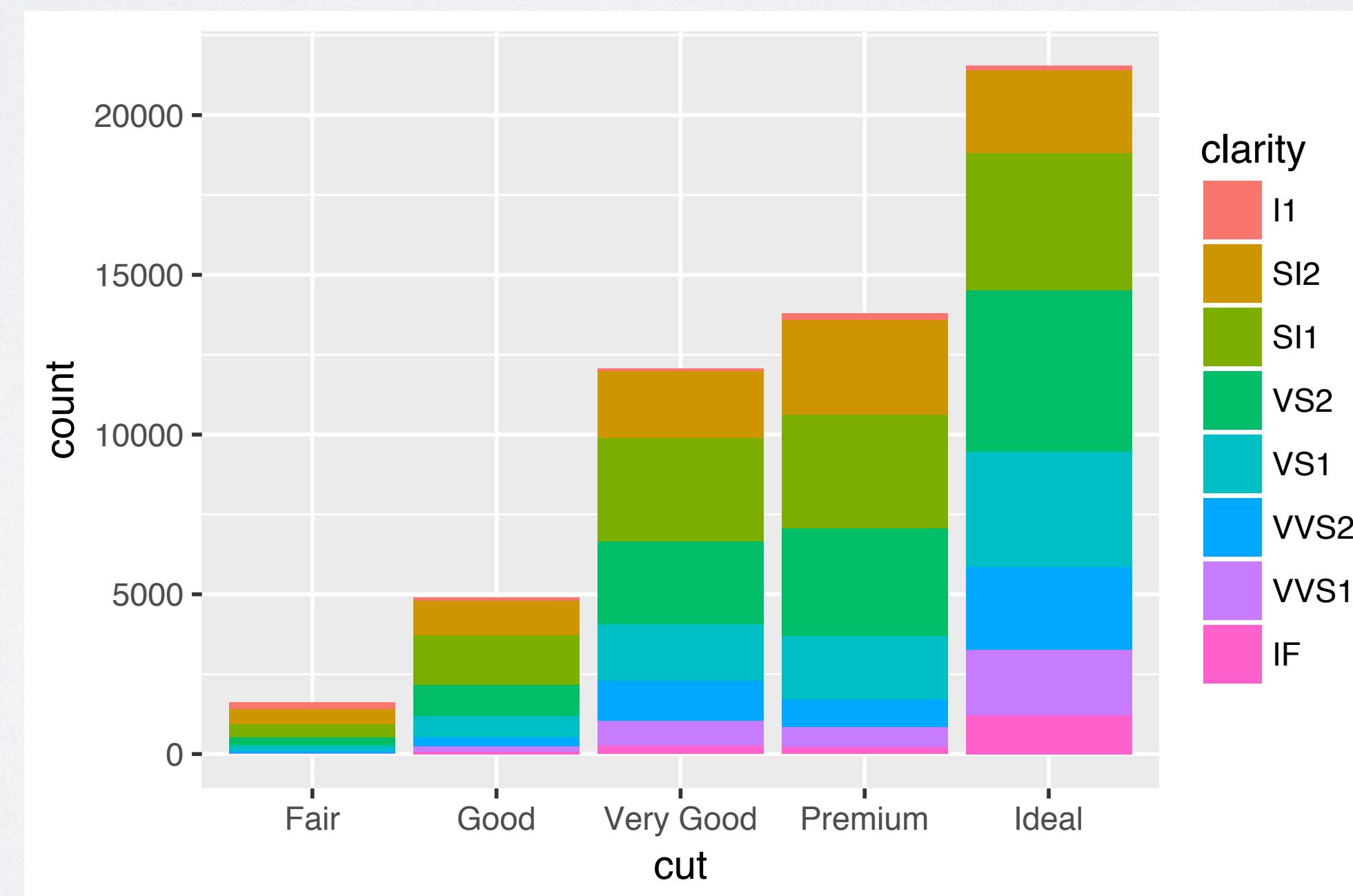
```
ggplot(diamonds, mapping = aes(x = cut)) +  
  geom_bar(mapping = aes(fill = cut))
```



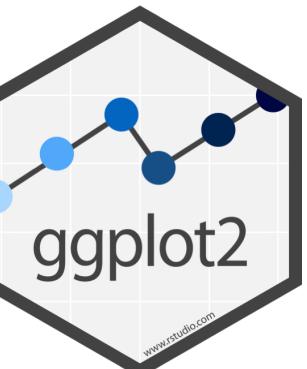
# Challenge

Can you alter this code to make this plot?

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



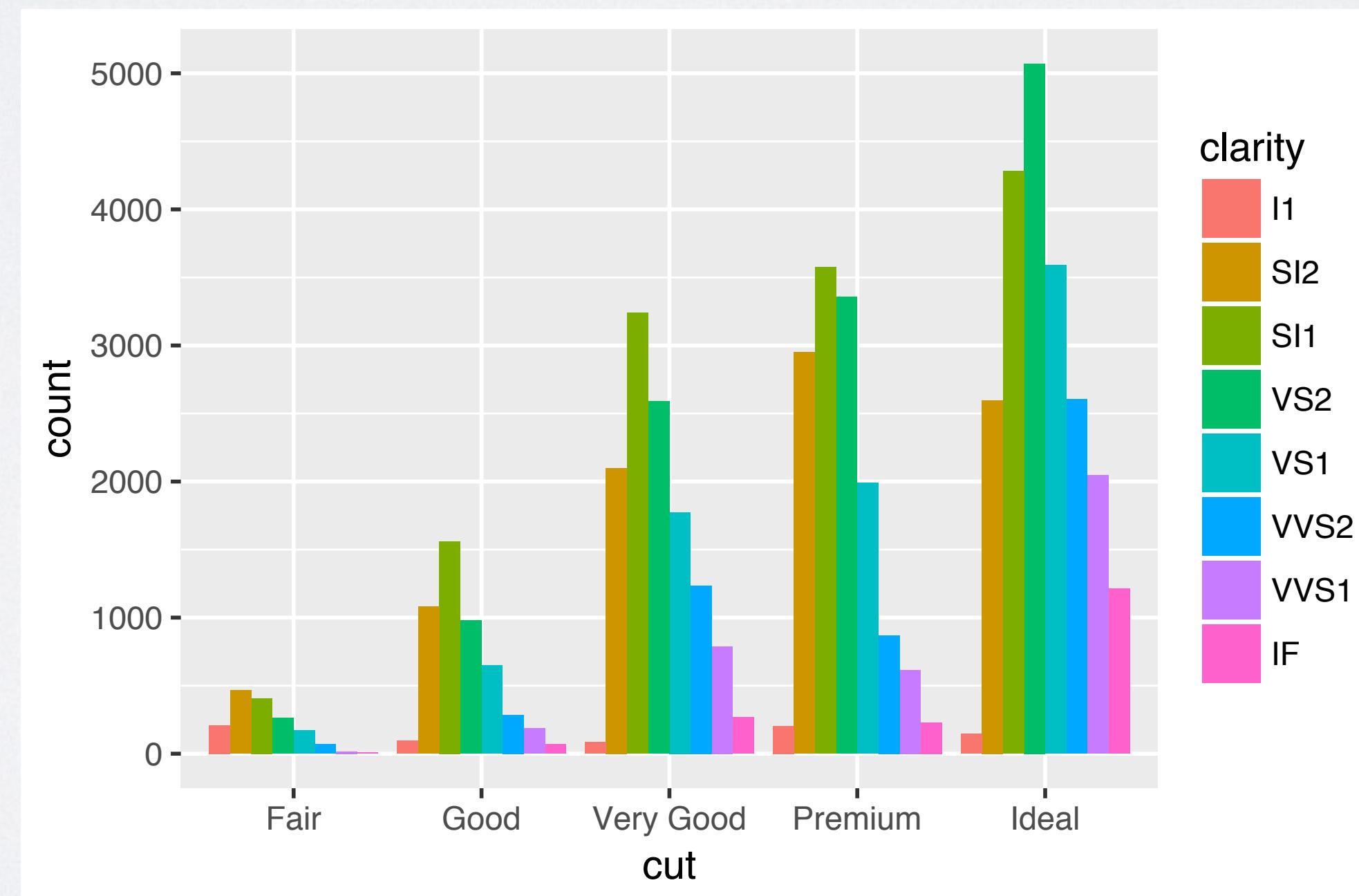
```
ggplot(diamonds, mapping = aes(x = cut)) +  
  geom_bar(mapping = aes(fill = clarity))
```



# Challenge

Can you alter this code to make this plot?

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

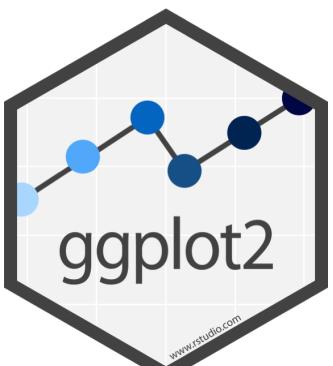


# Position Adjustment

How your plot arranges geoms that overlap with each other.

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

Set the adjustment  
method with the  
position argument



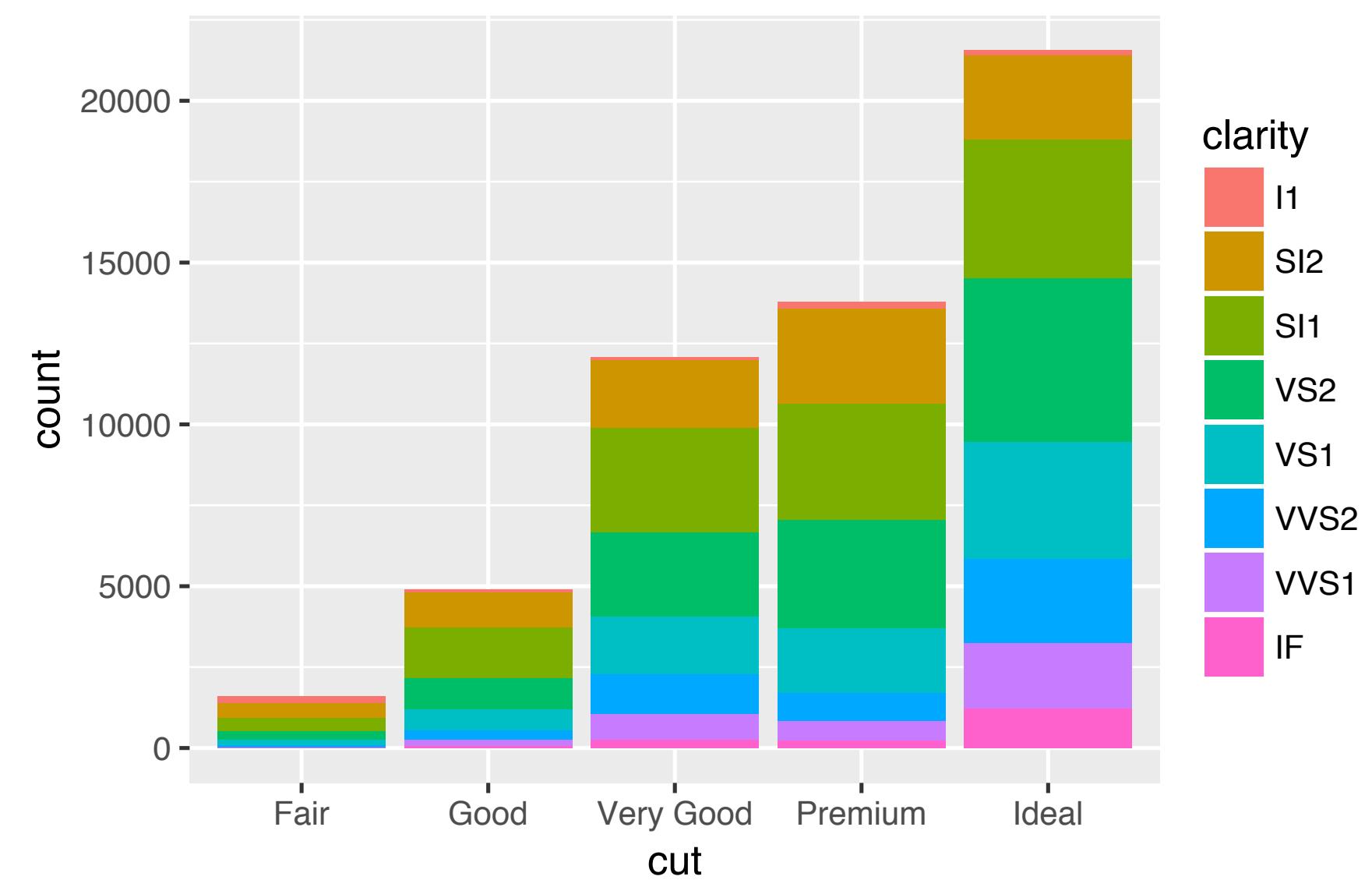
# Your turn

What do each of these adjustments do?  
(run the code, interpret, convince your group)

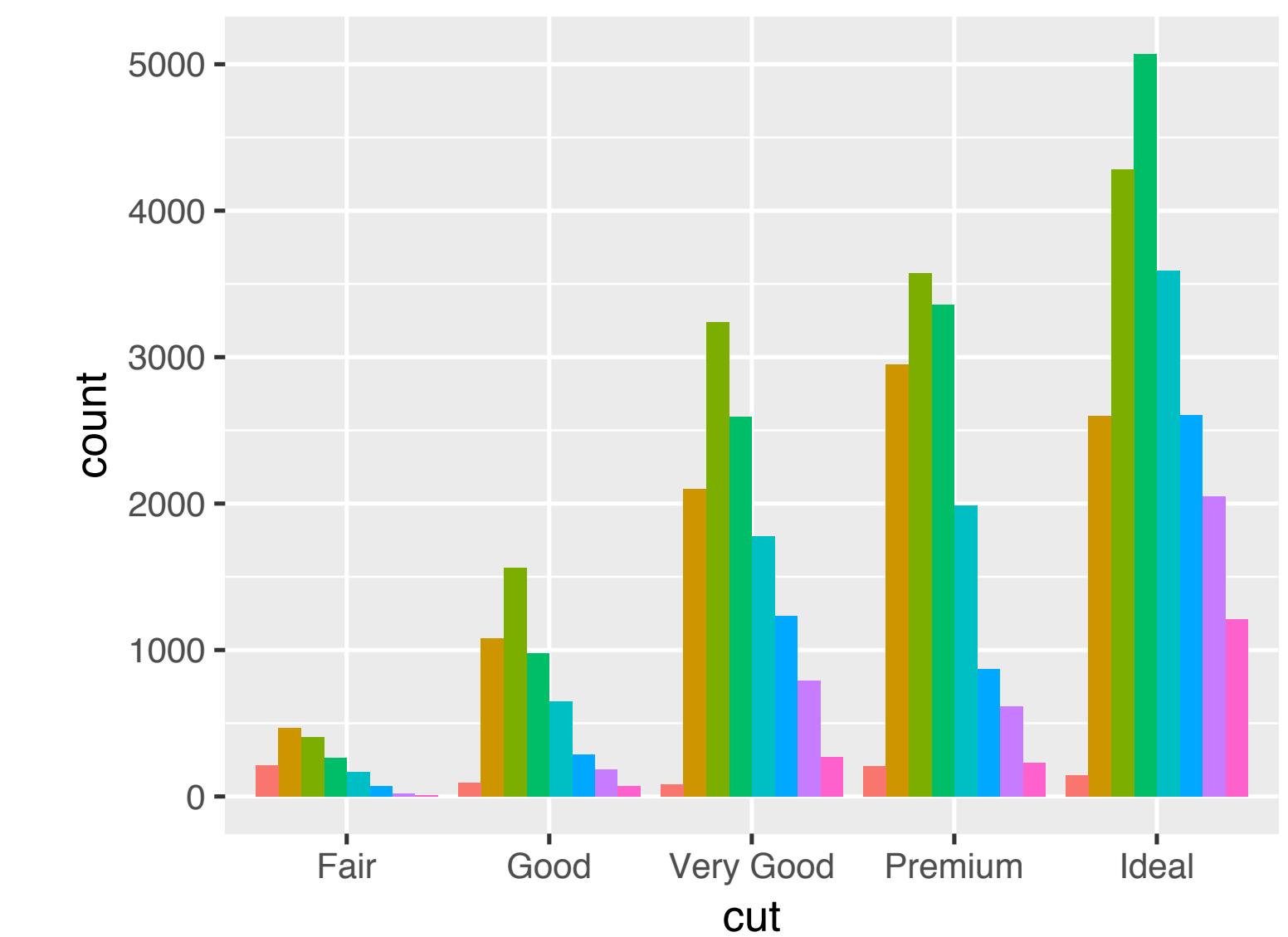
```
p <- ggplot(diamonds, aes(x = cut, fill = clarity))  
p + geom_bar(position = "stack")  
p + geom_bar(position = "dodge")  
p + geom_bar(position = "identity")  
p + geom_bar(position = "fill")
```



# stack

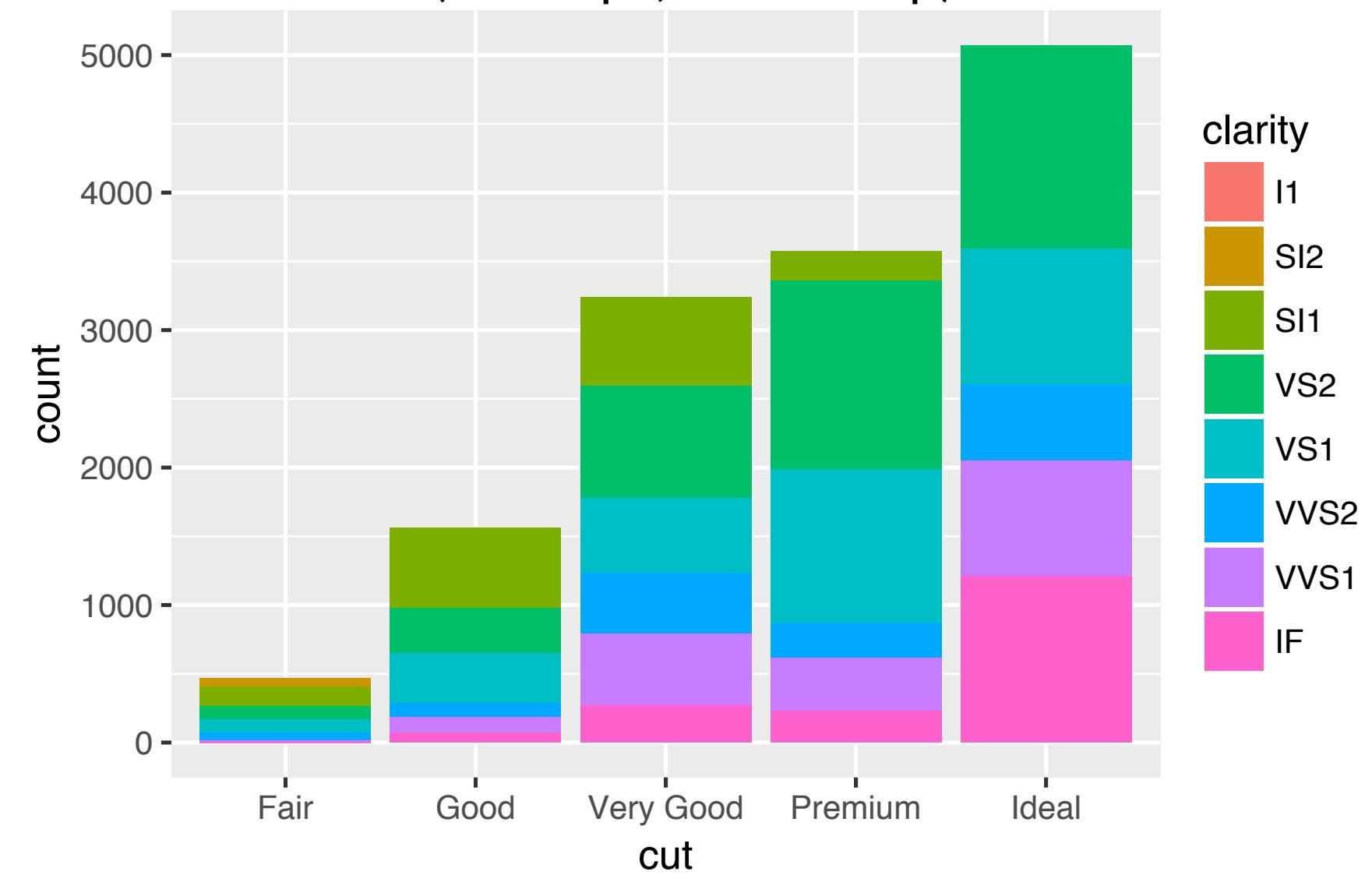


# dodge



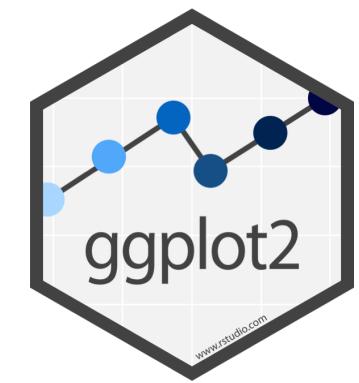
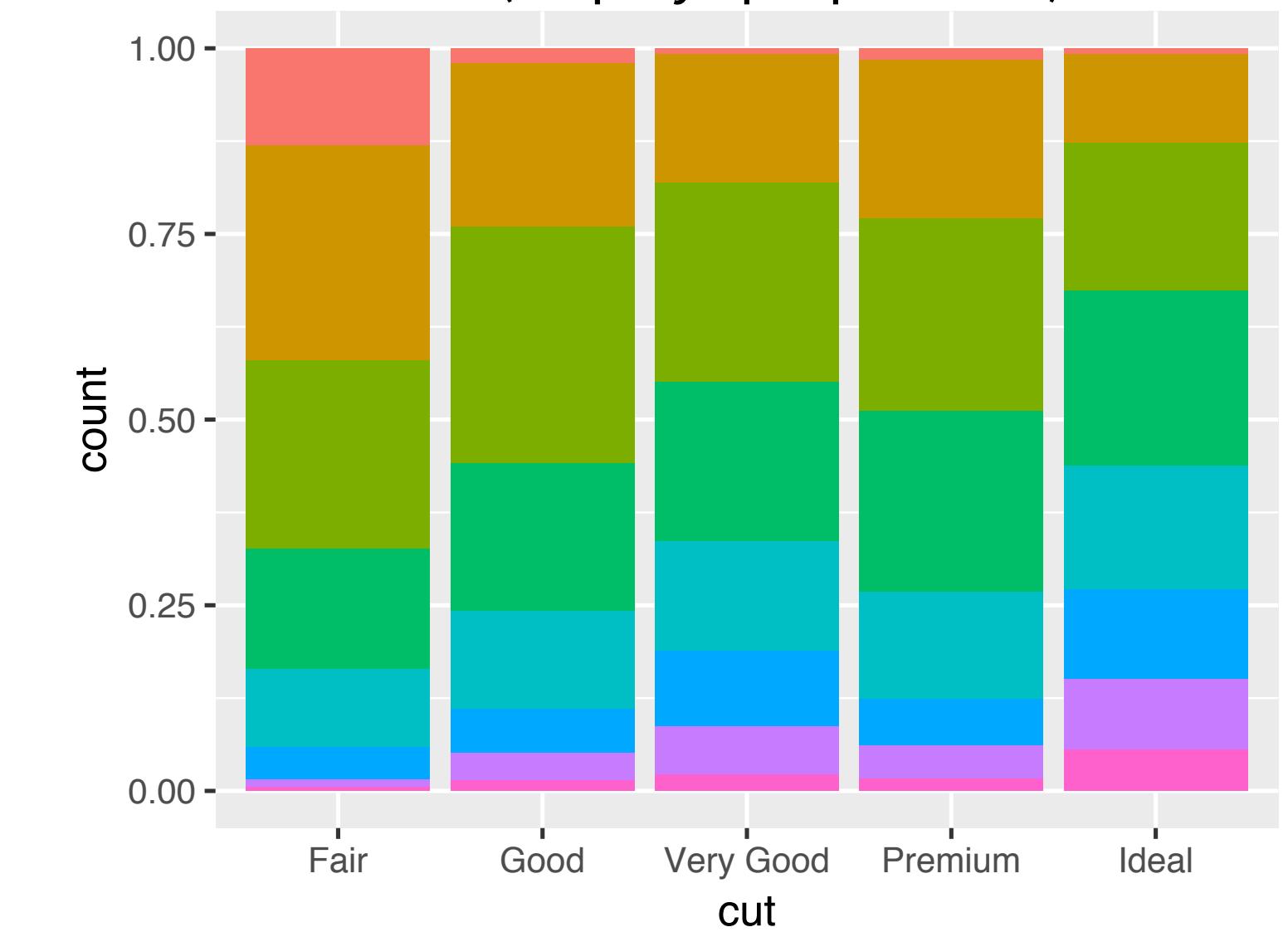
# identity

(overlaps, last on top)



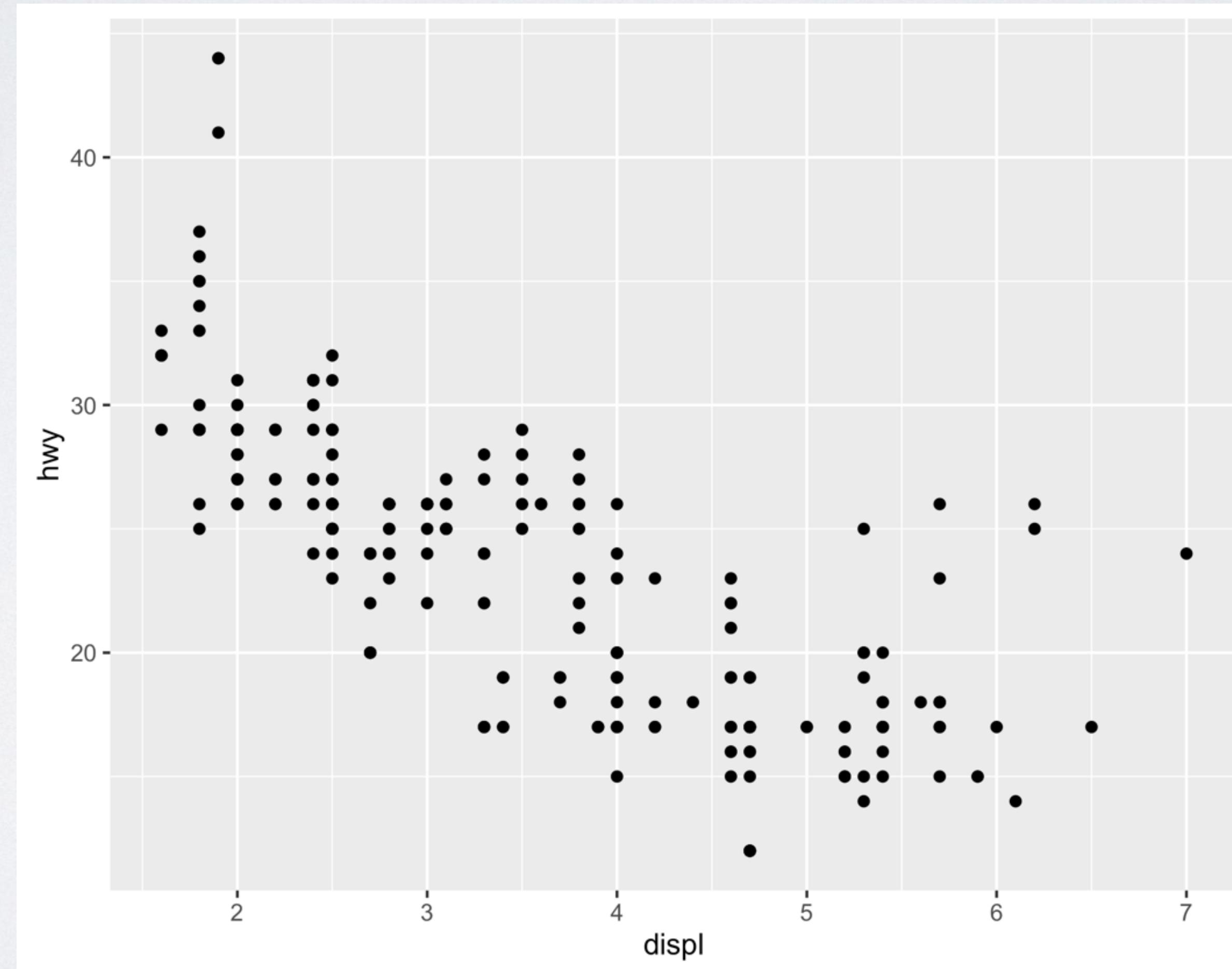
# fill

(displays proportions)



# Quiz

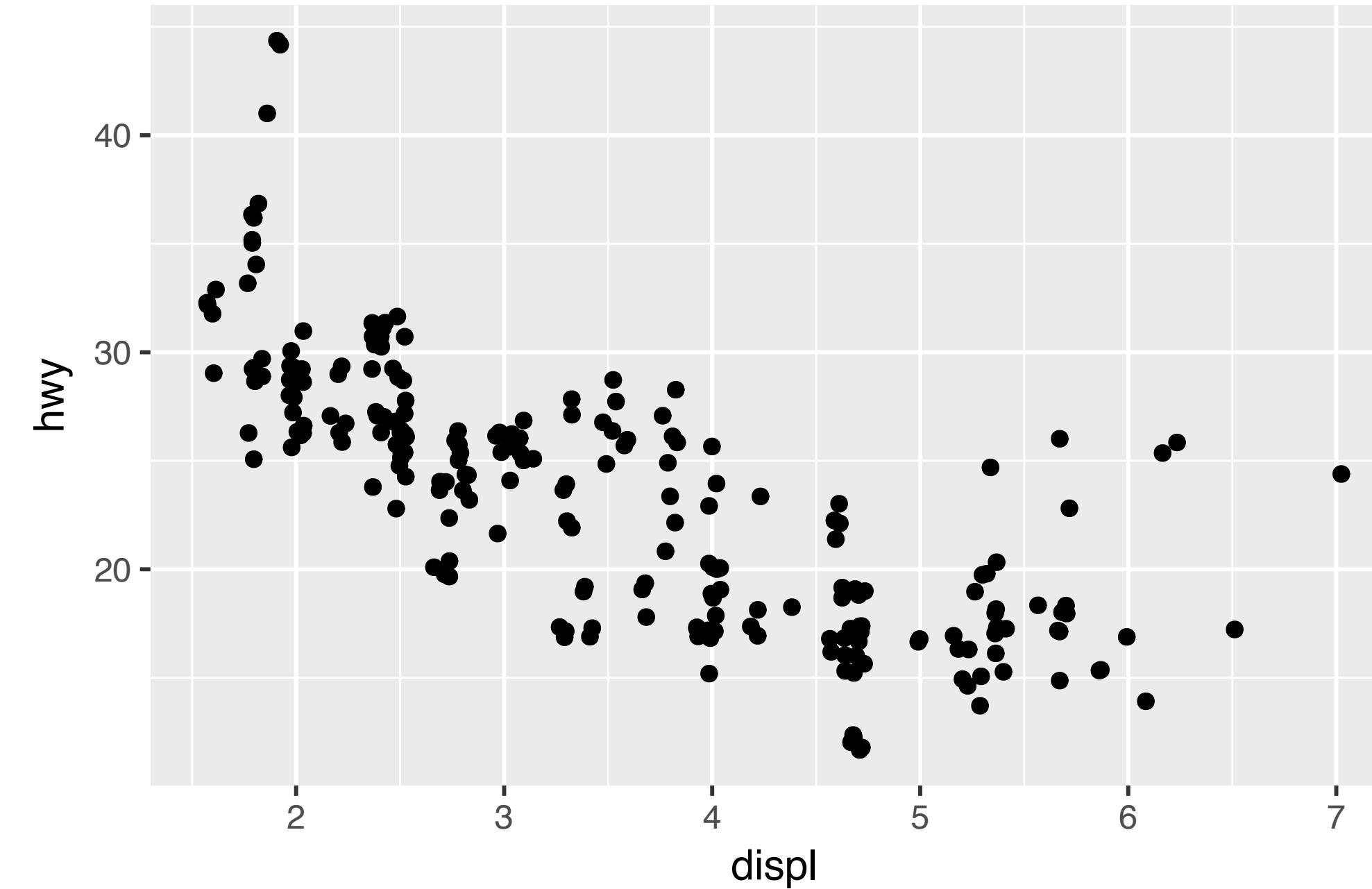
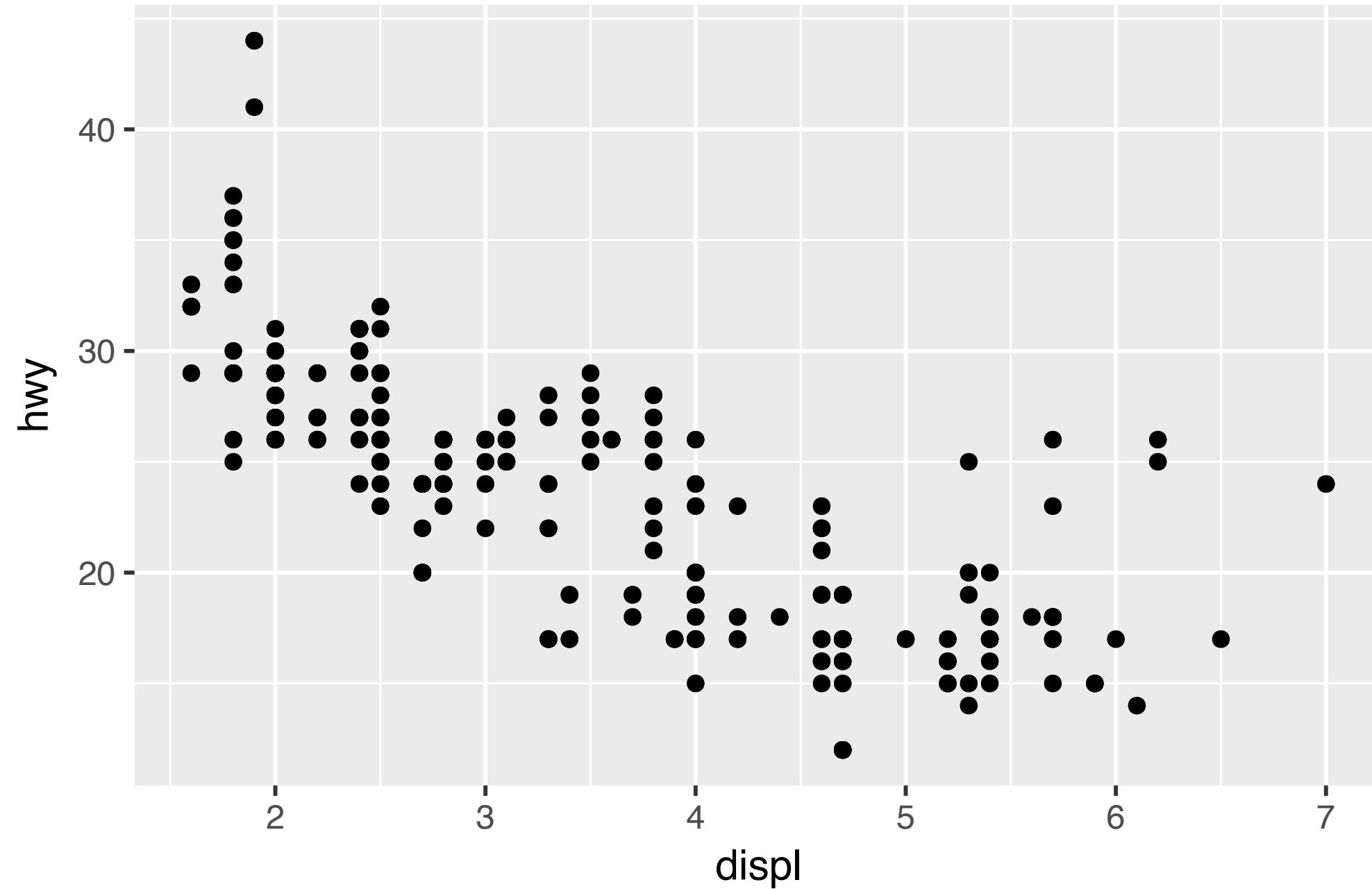
Why does this plot display 126 points? There are 234 in the data.



```
p <- ggplot(mpg, aes(displ, hwy))
```

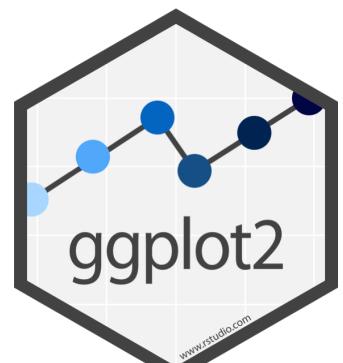
## jitter

(moves each point by a small, random amount)



```
p + geom_point()
```

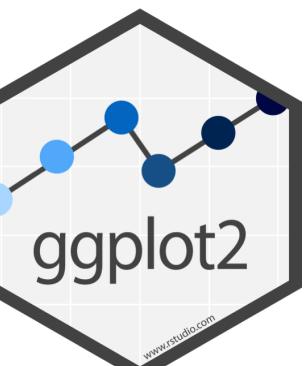
```
p + geom_point(position = "jitter")  
P + geom_jitter()
```



# A ggplot2 template

Make any plot by filling in the parameters of this template

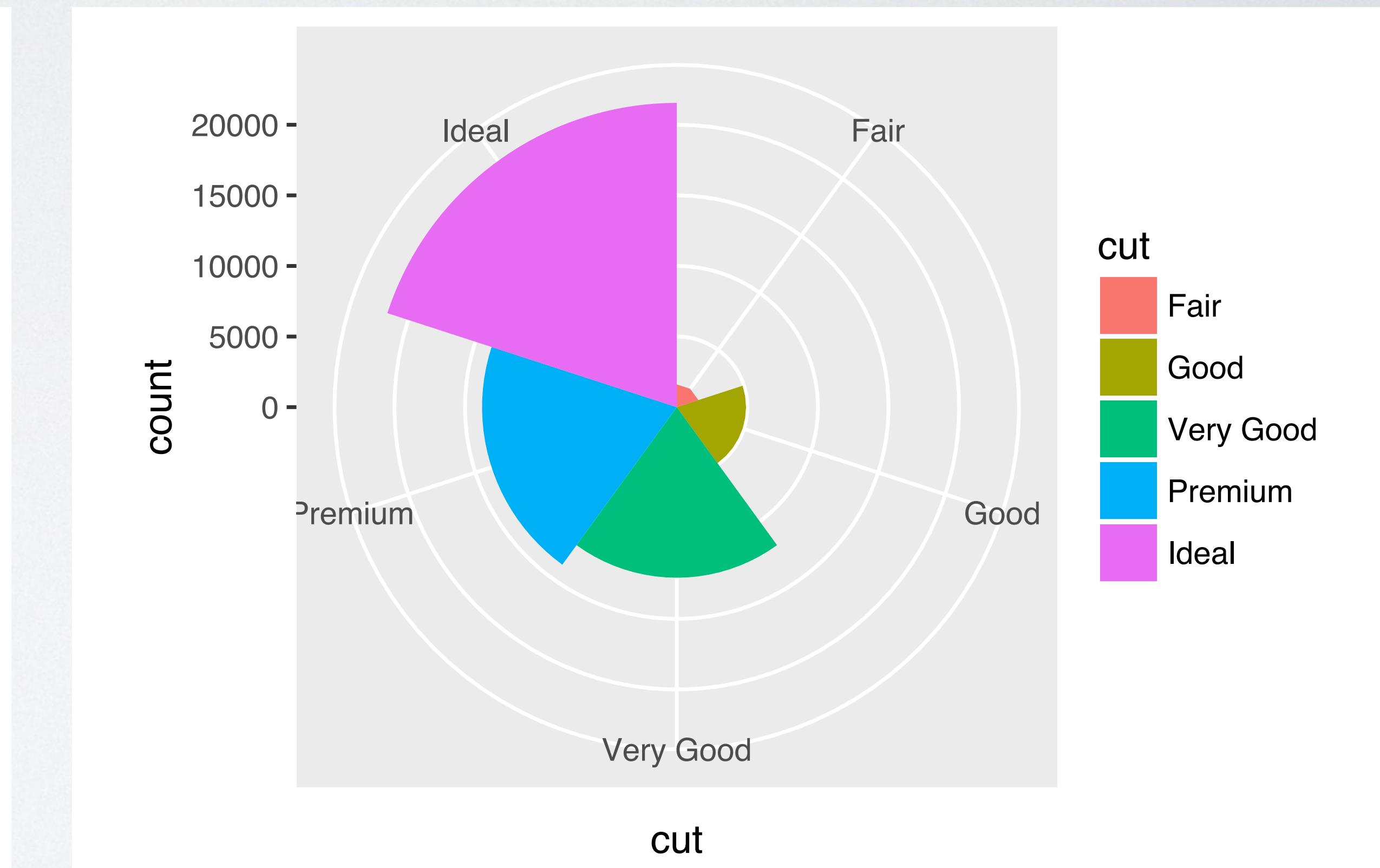
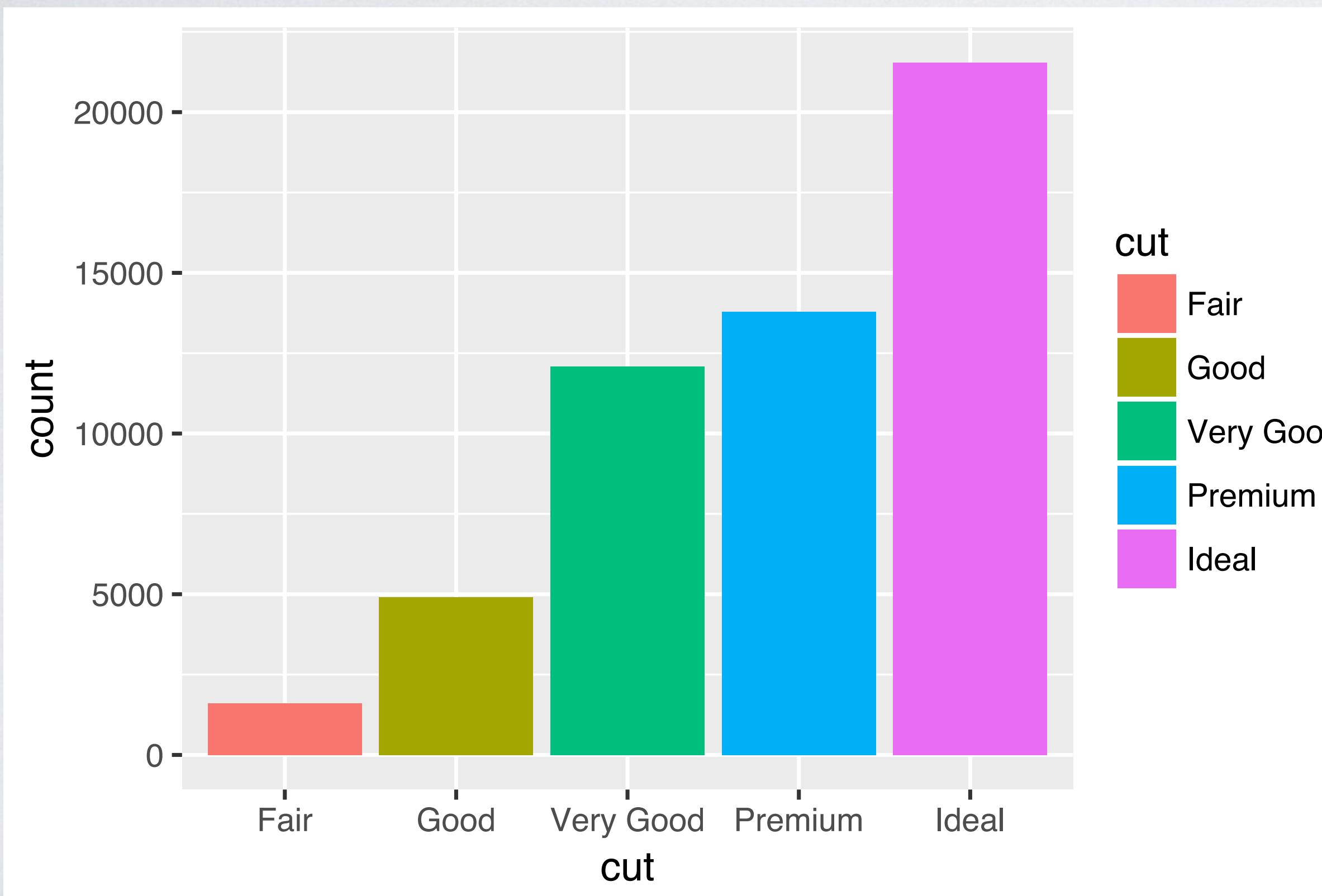
```
ggplot(data = <DATA>) +  
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),  
                    stat = <STAT>,  
                    position = <POSITION>) +  
  <FACET_FUNCTION>
```



# Coordinate systems

# Quiz

How is a bar chart (left) similar to a coxcomb plot (right)?

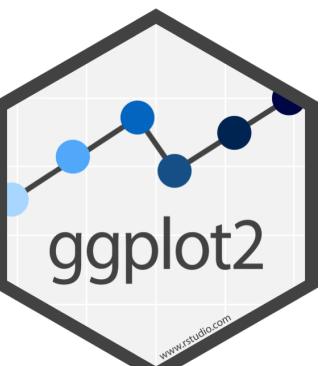


# Answer

A coxcomb plot is just a bar chart in polar coordinates

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

change the coordinate  
system by adding a  
coord\_ function



# coord\_ functions

Adds a coordinate system to a graph. ggplot2 adds coord\_cartesian() by default.



`r <- d + geom_bar()`

`r + coord_cartesian(xlim = c(0, 5))`

xlim, ylim

The default cartesian coordinate system

`r + coord_fixed(ratio = 1/2)`

ratio, xlim, ylim

Cartesian coordinates with fixed aspect ratio between x and y units

`r + coord_flip()`

xlim, ylim

Flipped Cartesian coordinates

`r + coord_polar(theta = "x", direction=1 )`

theta, start, direction

Polar coordinates

`r + coord_trans(ytrans = "sqrt")`

xtrans, ytrans, limx, limy

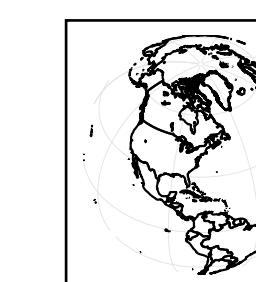
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

`π + coord_quickmap()`

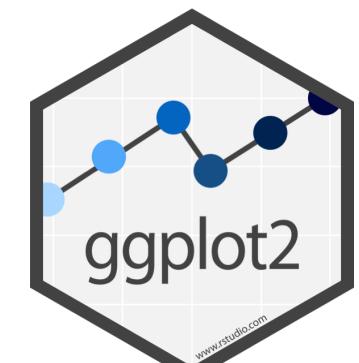
`π + coord_map(projection = "ortho",`

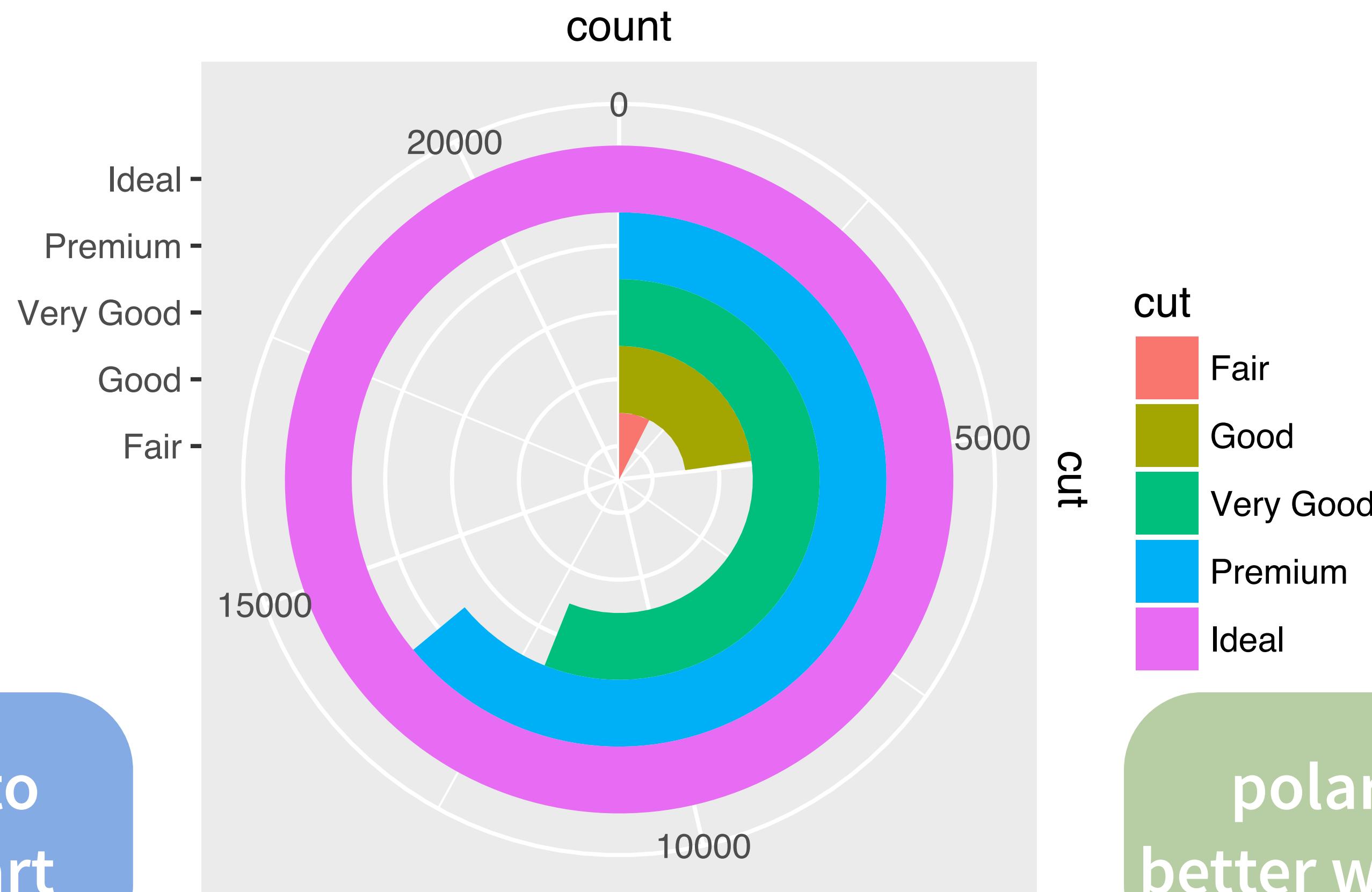
projection=c(41, -74, 0))

projection, orientation, xlim, ylim



Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

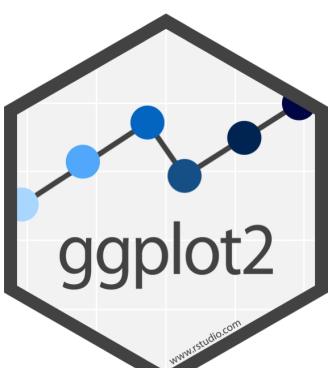




Add theta = "y" to make a radar chart

```
ggplot(diamonds) +  
  geom_bar(mapping = aes(x = cut, fill = cut), width = 1) +  
  coord_polar(theta = y)
```

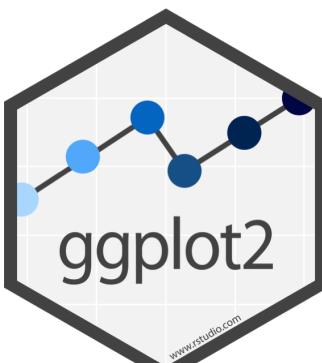
polar charts look better when bars touch



# A ggplot2 template

Make any plot by filling in the parameters of this template

```
ggplot(data = <DATA>) +  
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),  
    stat = <STAT>,  
    position = <POSITION>) +  
  <COORD_FUNCTION> +  
  <FACET_FUNCTION>
```



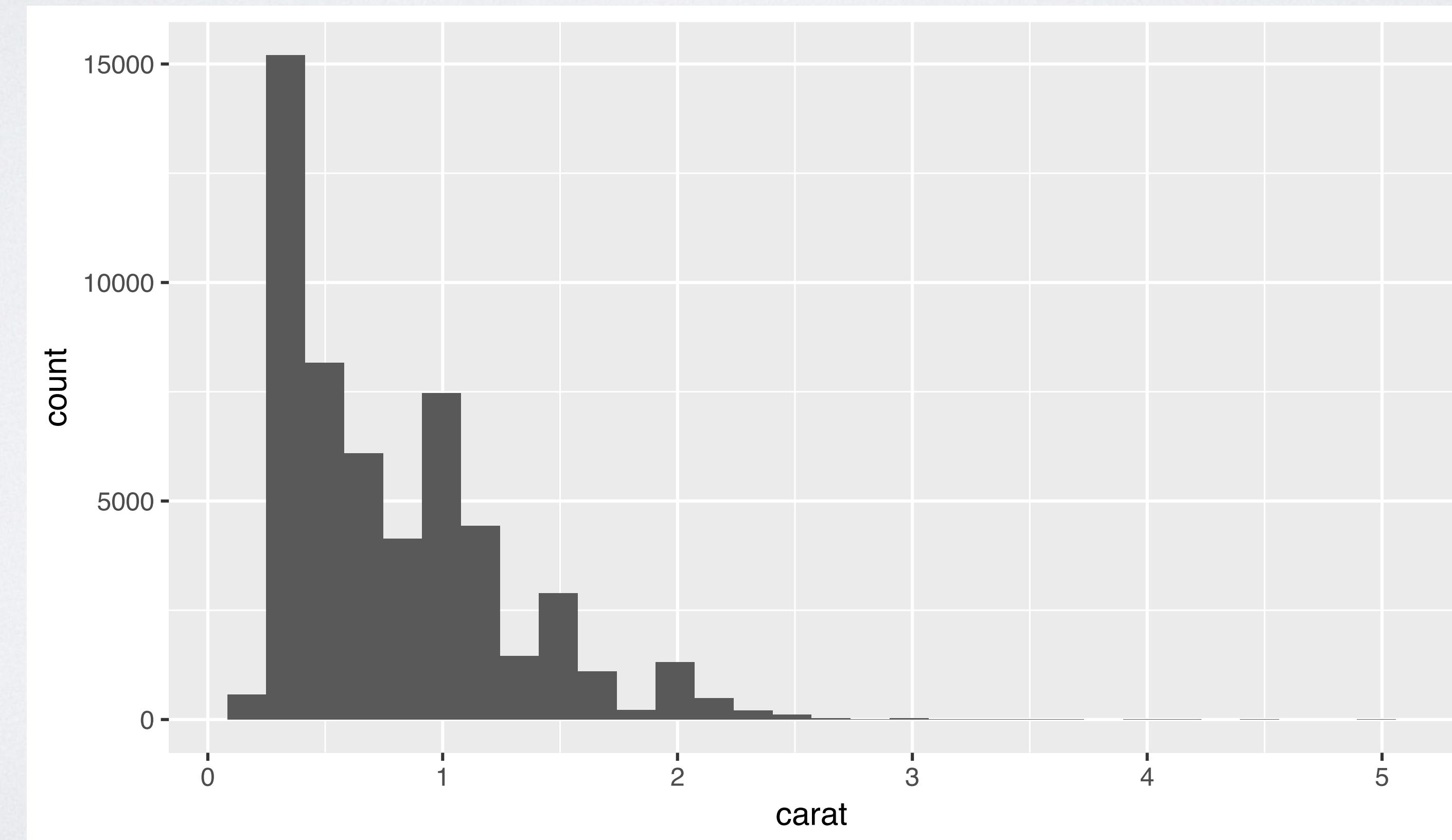
# Parameters

R

# Quiz

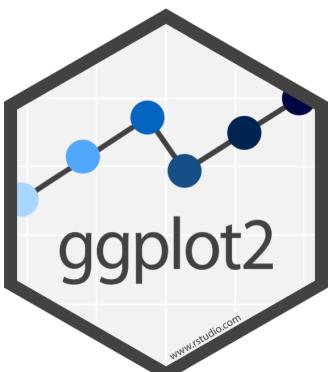
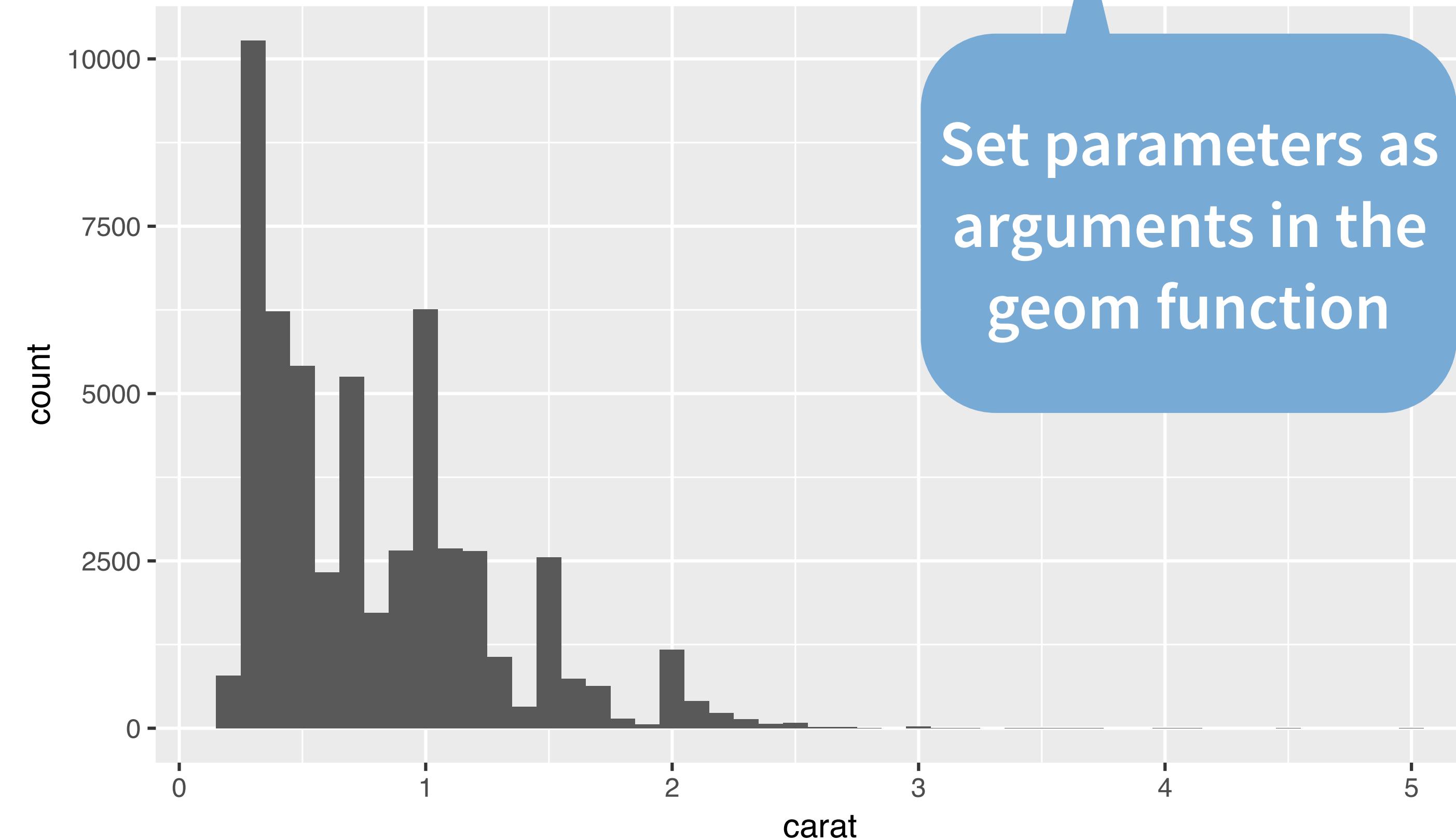
What do the y values in this histogram represent?

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



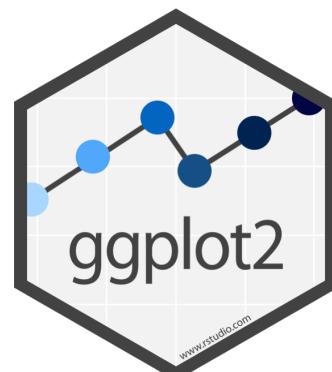
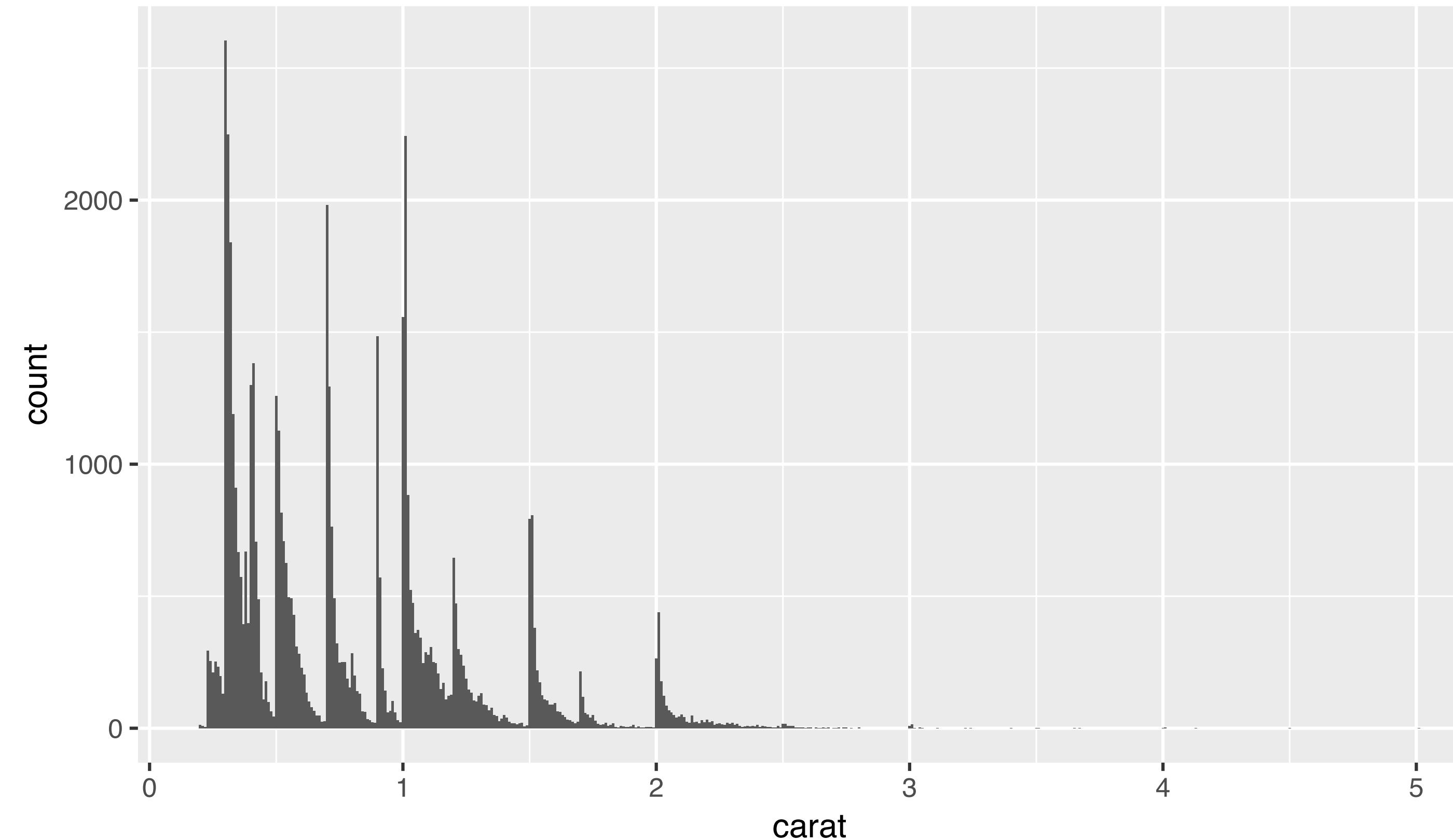
# Parameters

```
ggplot(diamonds) +  
  geom_histogram(aes(carat), binwidth = 0.1)
```



# Parameters

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

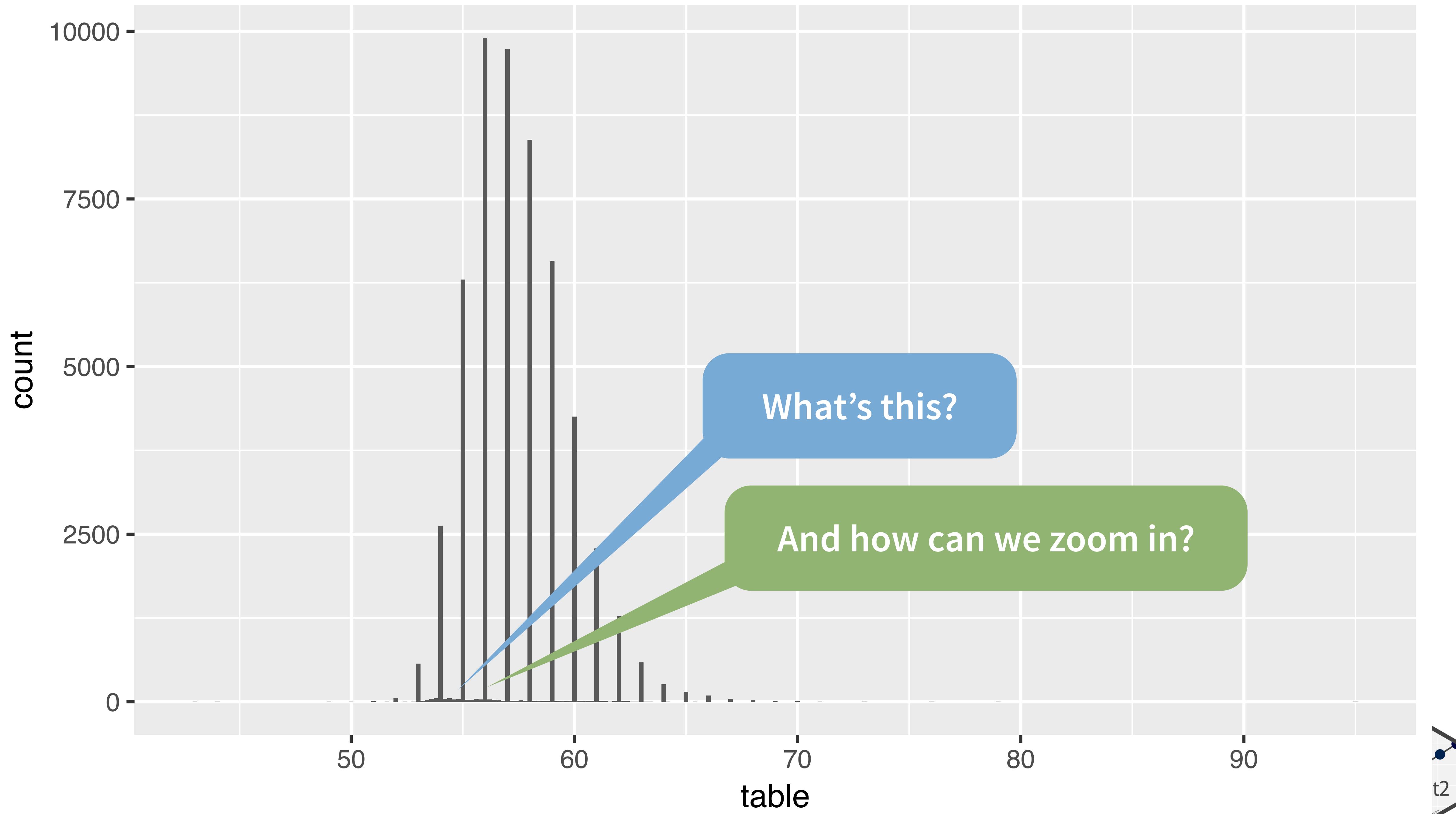


# Quiz

Try setting the binwidth to 0.2 or less.

Look closely Do you notice anything weird?

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



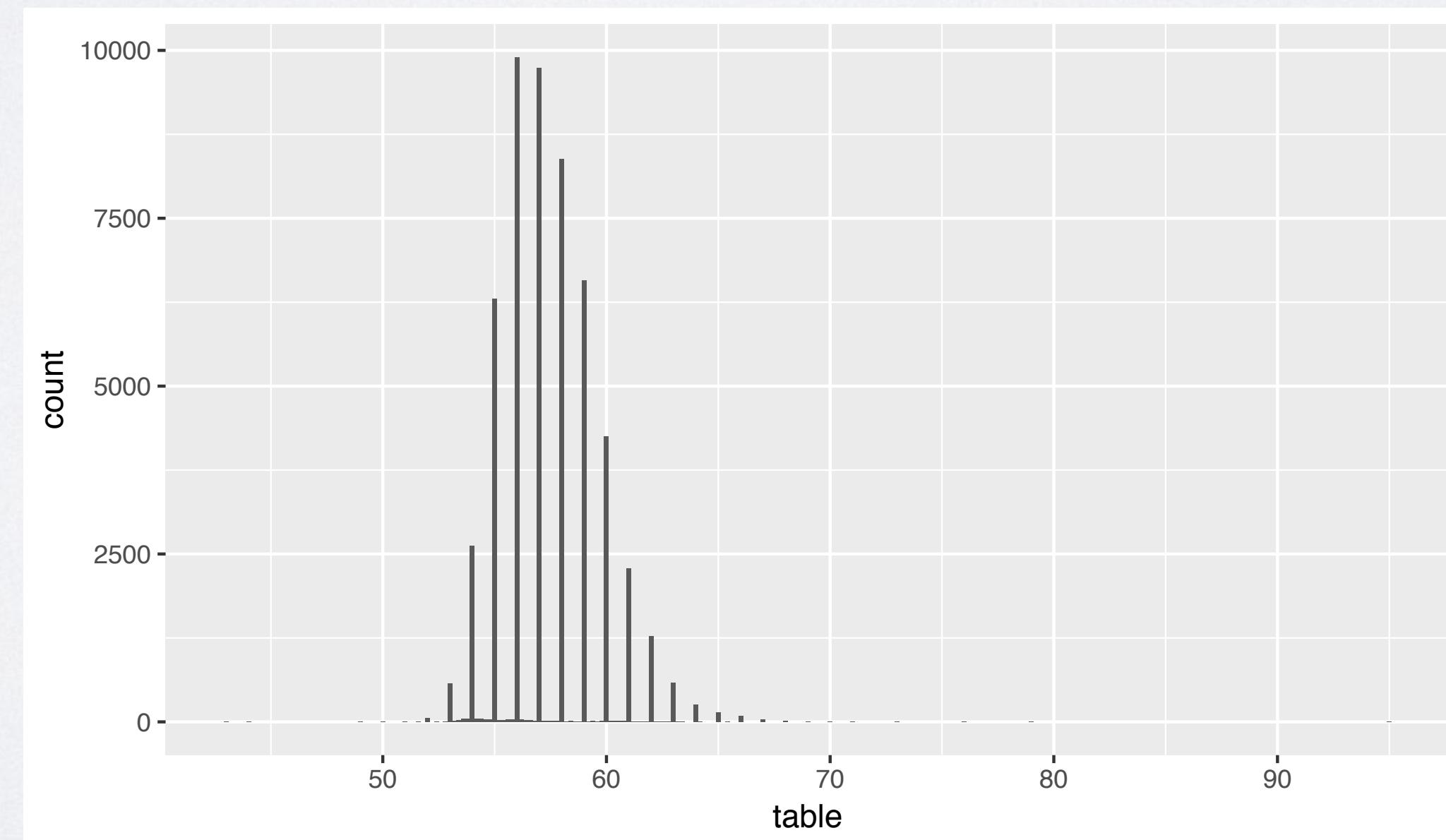
# Zooming

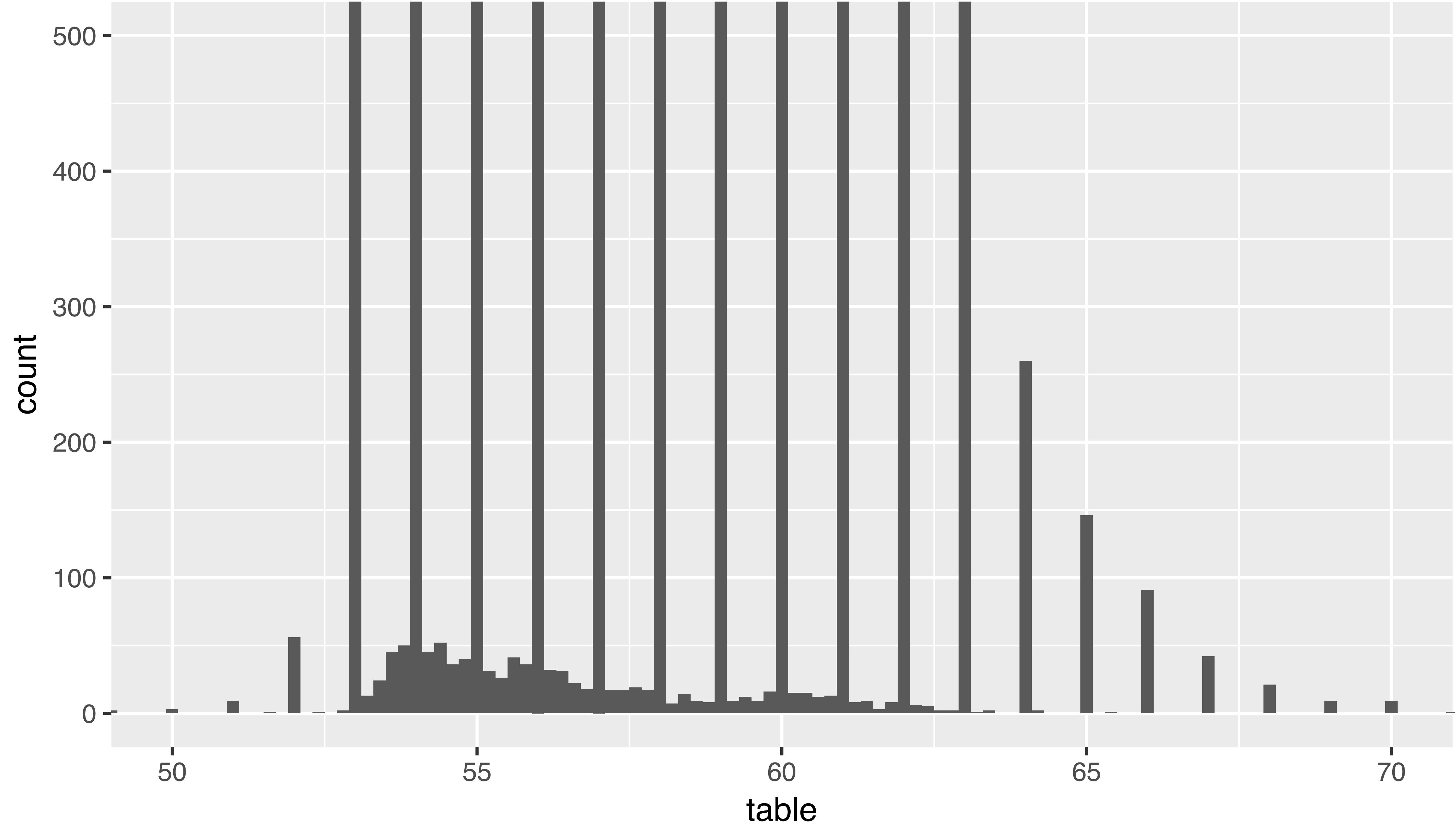


# Try it

Zoom with `xlim` and `ylim` in `coord_cartesian()`, e.g.

```
ggplot(diamonds) +  
  geom_histogram(aes(table), binwidth = 0.2) +  
  coord_cartesian(xlim = c(50, 70), ylim = c(0, 500))
```



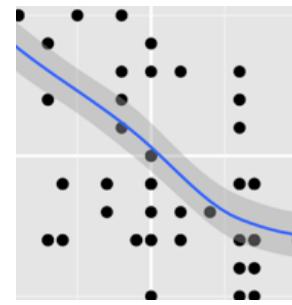


# Recap

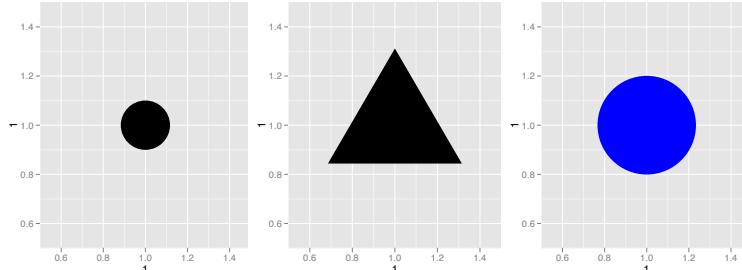
R

# ggplot2 recap

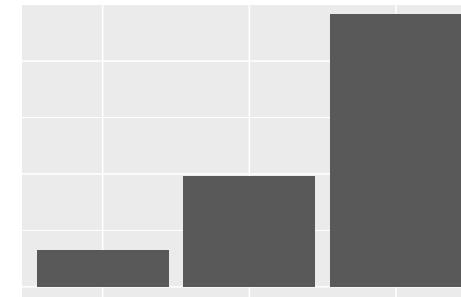
- Use a **geom\_** function to visualize **cases**.



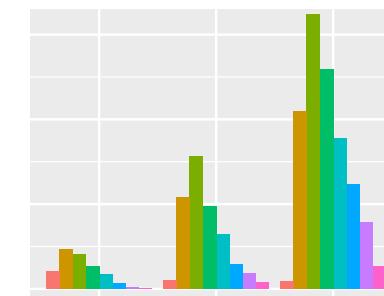
Add new layers with new **geom\_** functions



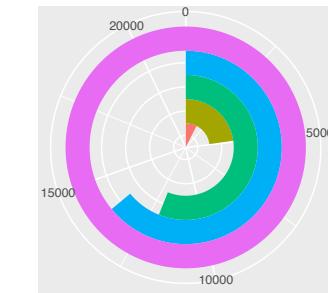
Map **variables** to aesthetics with **mapping = aes()**



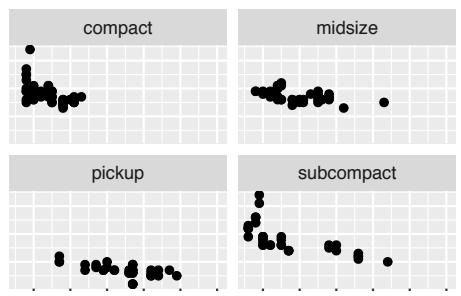
Plot different summaries with **stat =**



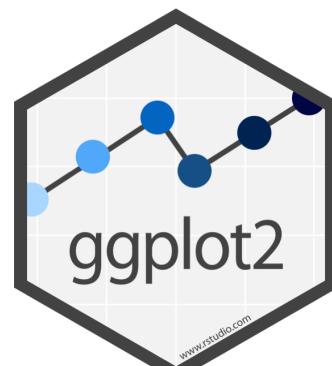
Handle collisions with **position =**



Change coordinate systems with a **coord\_** function



Facet the plot with a **facet\_** function



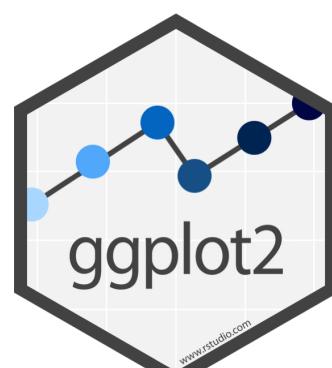
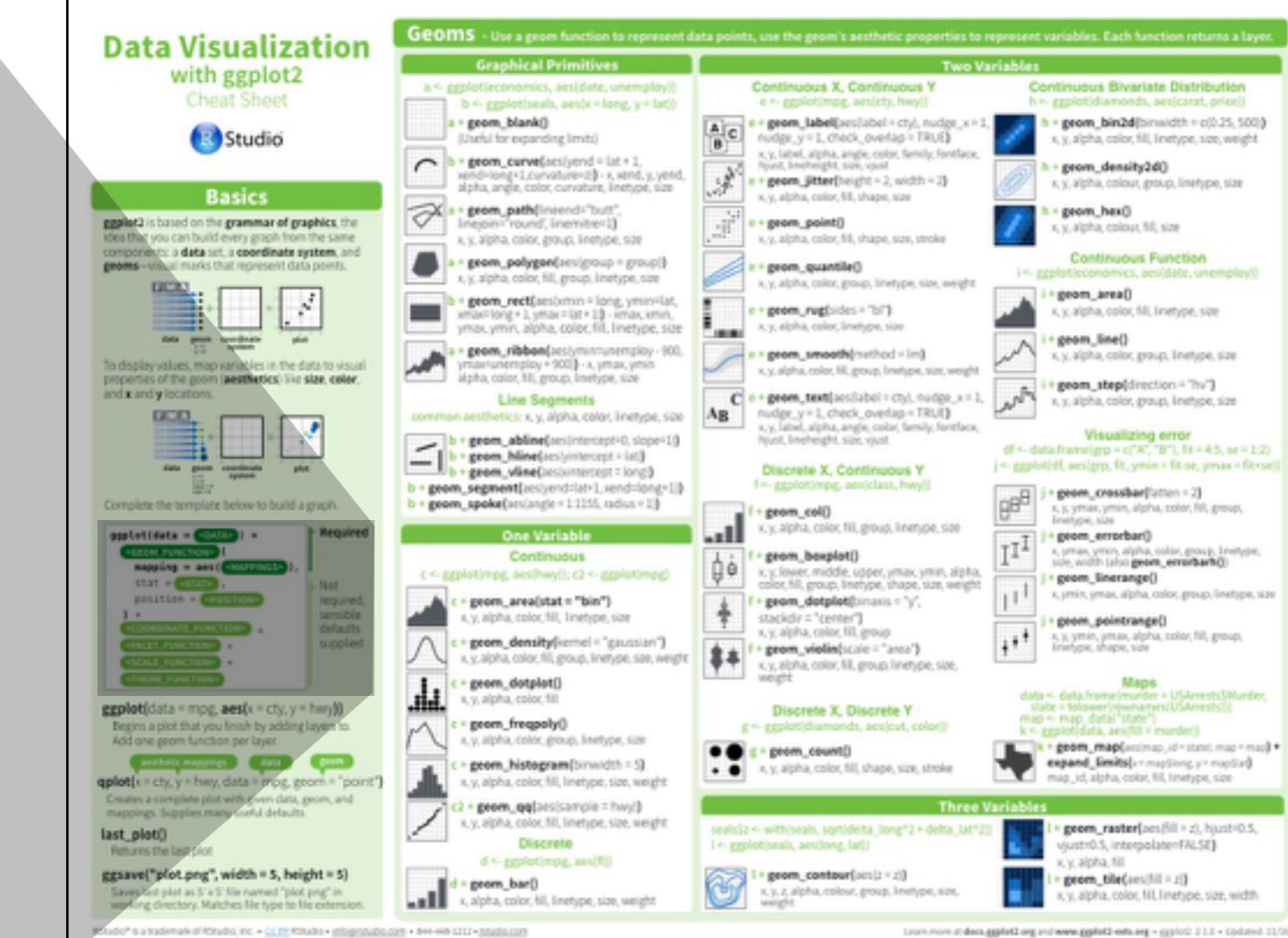
# A ggplot2 template

Make any plot by filling in the parameters of this template

```
ggplot(data = <DATA>) +  
  <GEOM_FUNCTION>(  
    mapping = aes(<MAPPINGS>),  
    stat = <STAT>,  
    position = <POSITION>  
  ) +  
  <COORDINATE_FUNCTION> +  
  <FACET_FUNCTION> +  
  <SCALE_FUNCTION> +  
  <THEME_FUNCTION>
```

Required

Not required,  
sensible  
defaults  
supplied



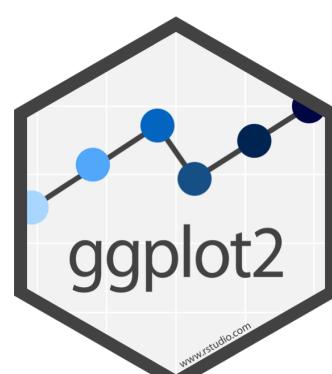
# A ggplot2 template

```
ggplot(data = <DATA>) +  
  <GEOM_FUNCTION>(  
    mapping = aes(<MAPPINGS>),  
    stat = <STAT>,  
    position = <POSITION>  
  ) +  
  <COORDINATE_FUNCTION> +  
  <FACET_FUNCTION> +  
  <SCALE_FUNCTION> +  
  <THEME_FUNCTION>
```

Required

Not required,  
sensible  
defaults  
supplied

```
ggplot(data = <DATA>) +  
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),  
    stat = <STAT>,  
    position = <POSITION>) +  
  <COORD_FUNCTION> +  
  <FACET_FUNCTION>
```



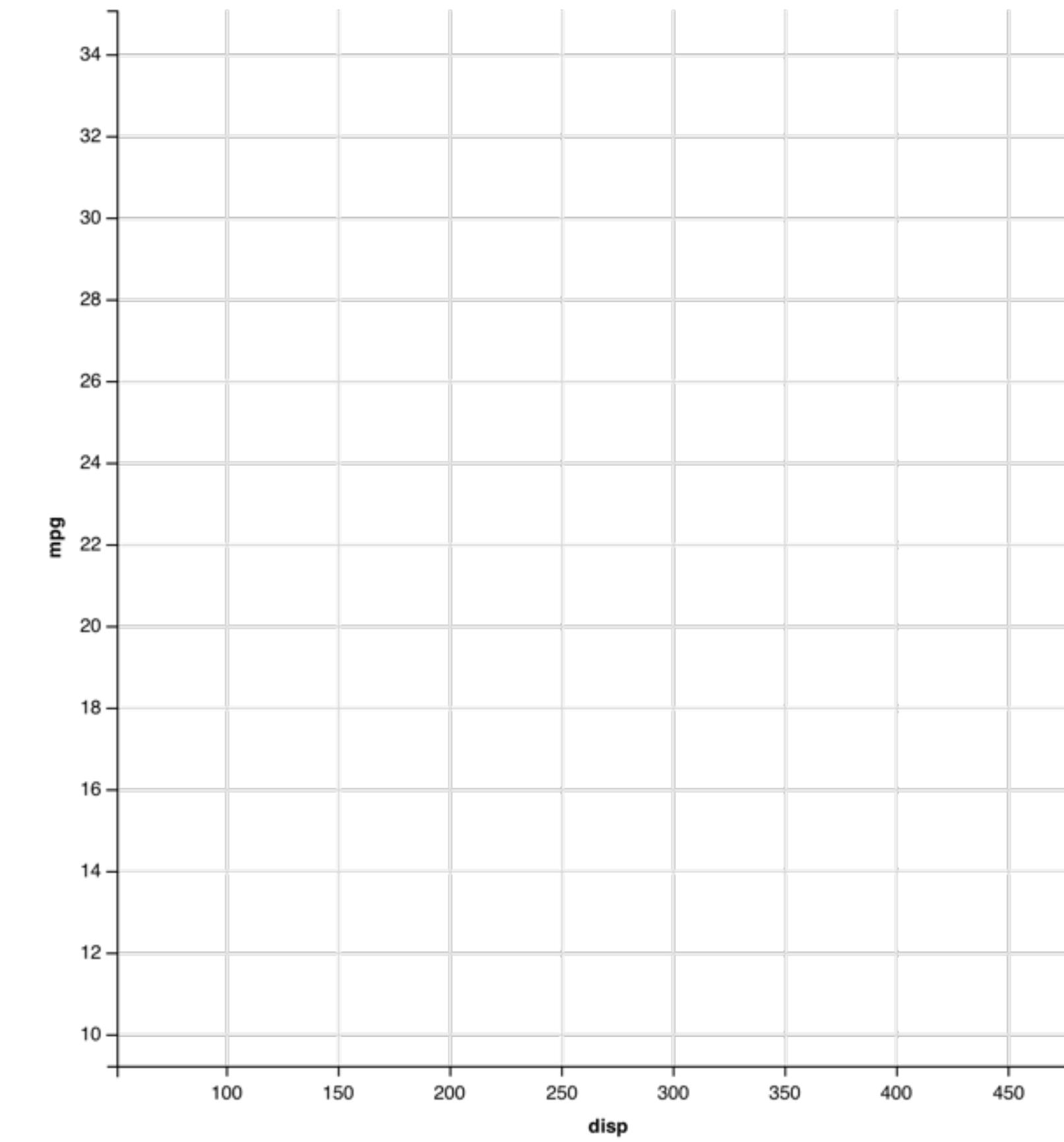
# Grammar of Graphics



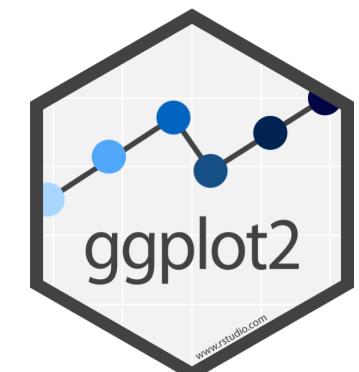
mpg	cyl	disp	hp
21.0	6	160.0	2
21.0	6	160.0	2
22.8	4	108.0	1
21.4	6	258.0	2
18.7	8	360.0	3
18.1	6	225.0	2
14.3	8	360.0	5
24.4	4	146.7	1
22.8	4	140.8	1
19.2	6	167.6	2
17.8	6	167.6	2
16.4	8	275.8	3
17.3	8	275.8	3
15.2	8	275.8	3
10.4	8	472.0	4
10.4	8	460.0	4
14.7	8	440.0	4
32.4	4	78.7	1
30.4	4	75.7	1
33.9	4	71.1	1

data

geom

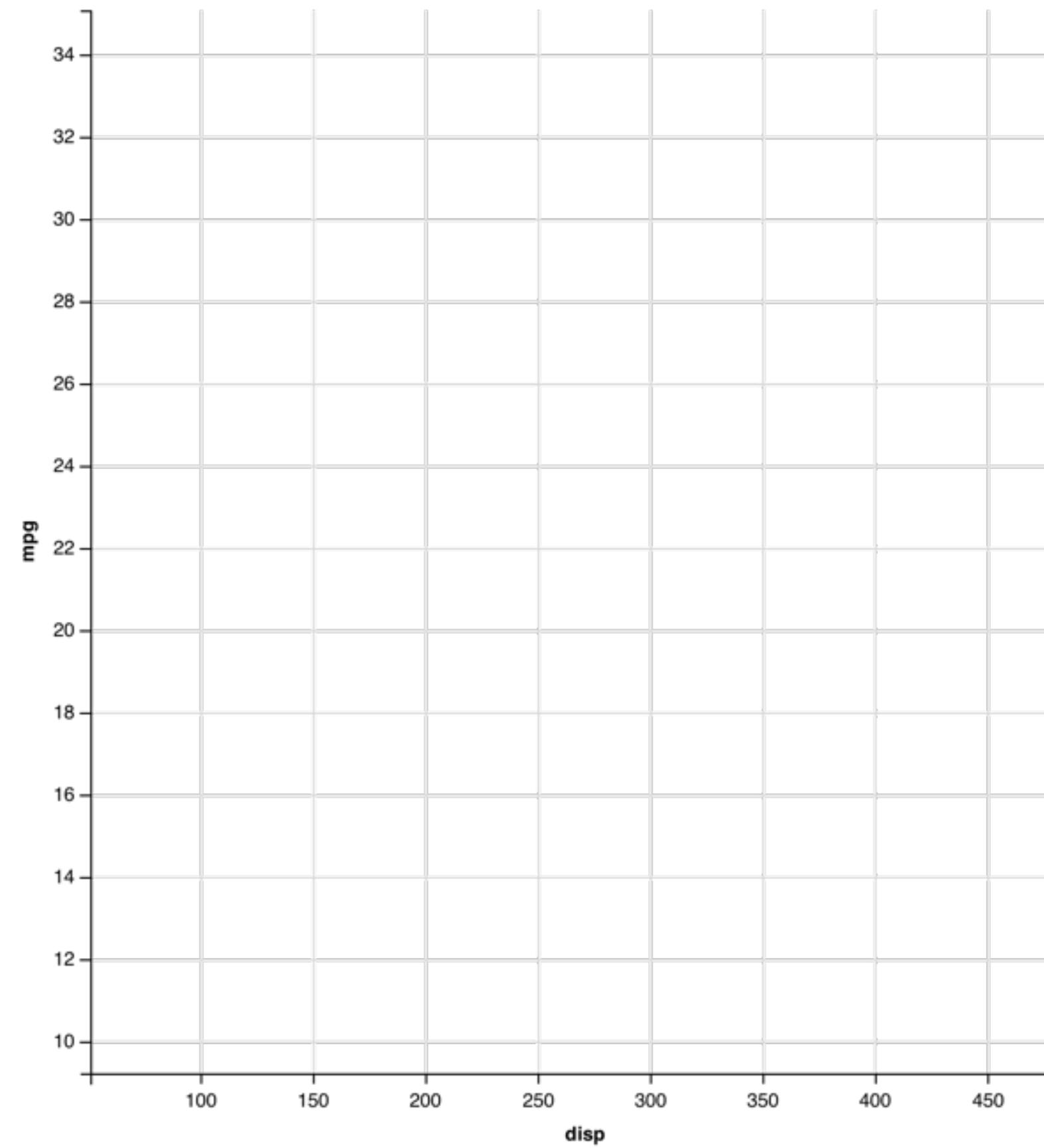


coordinate  
system



# properties

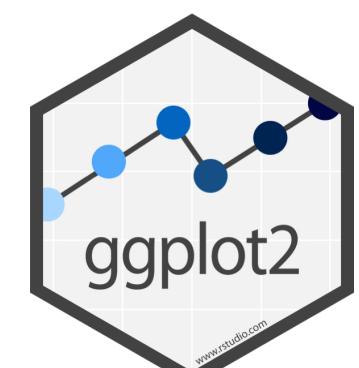
mpg	cyl	disp	hp	fill
21.0	6	160.0	2	●
21.0	6	160.0	2	●
22.8	4	108.0	1	●
21.4	6	258.0	2	●
18.7	8	360.0	3	●
18.1	6	225.0	2	●
14.3	8	360.0	5	●
24.4	4	146.7	1	●
22.8	4	140.8	1	●
19.2	6	167.6	2	●
17.8	6	167.6	2	●
16.4	8	275.8	3	●
17.3	8	275.8	3	●
15.2	8	275.8	3	●
10.4	8	472.0	4	●
10.4	8	460.0	4	●
14.7	8	440.0	4	●
32.4	4	78.7	1	●
30.4	4	75.7	1	●
33.9	4	71.1	1	●



data

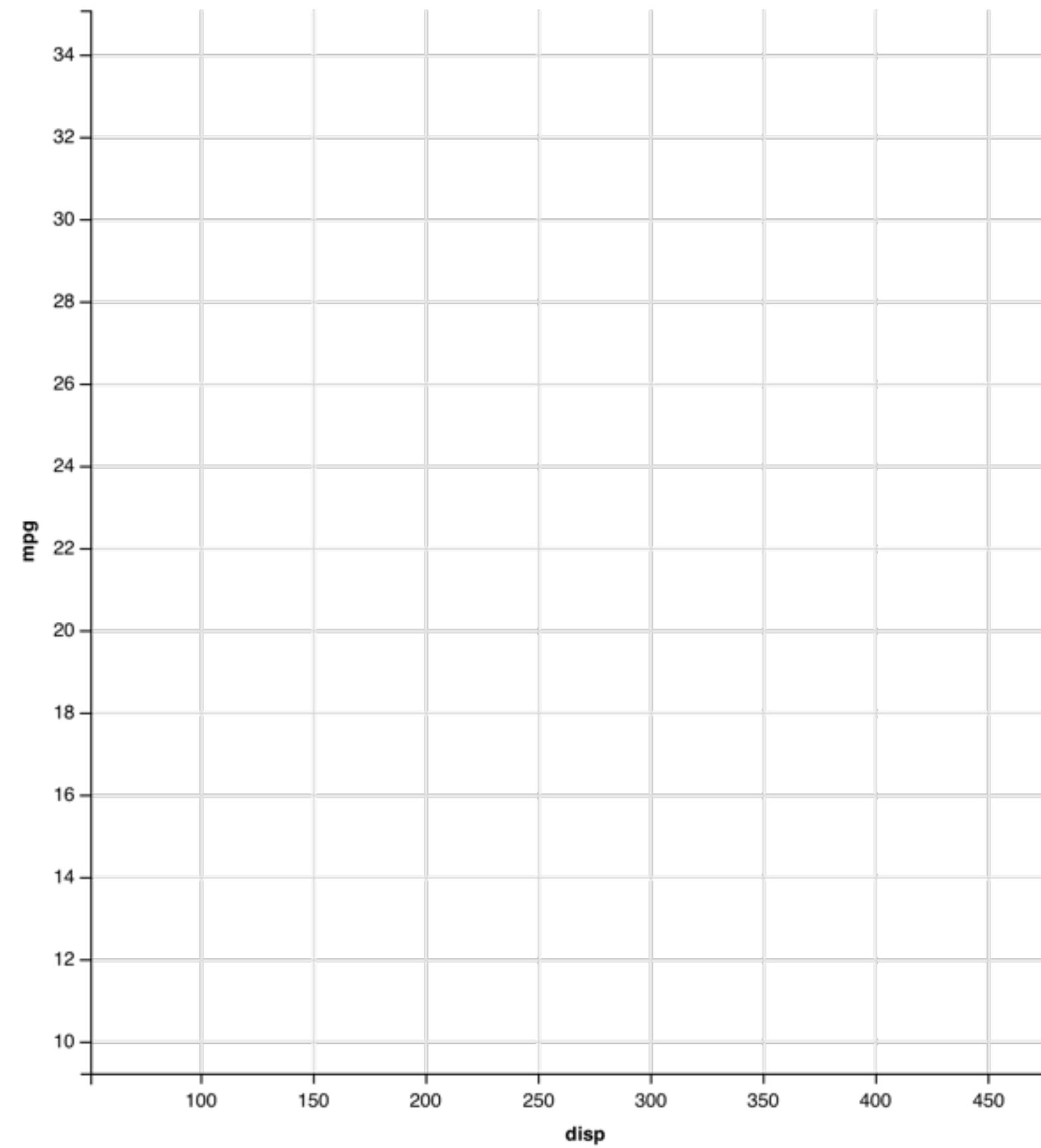
geom

coordinate  
system



# properties

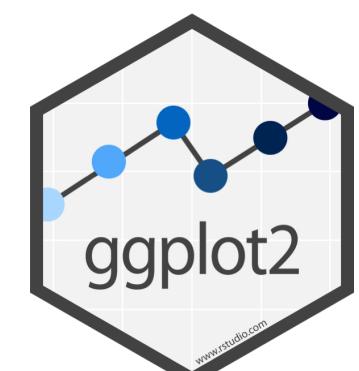
mpg	shape		fill
	cyl	hp	
21.0	6 +	160.0	2
21.0	6 +	160.0	2
22.8	4 ●	108.0	1
21.4	6 +	258.0	2
18.7	8 ♦	360.0	3
18.1	6 +	225.0	2
14.3	8 ♦	360.0	5
24.4	4 ●	146.7	1
22.8	4 ●	140.8	1
19.2	6 +	167.6	2
17.8	6 +	167.6	2
16.4	8 ♦	275.8	3
17.3	8 ♦	275.8	3
15.2	8 ♦	275.8	3
10.4	8 ♦	472.0	4
10.4	8 ♦	460.0	4
14.7	8 ♦	440.0	4
32.4	4 ●	78.7	1
30.4	4 ●	75.7	1
33.9	4 ●	71.1	1



data

geom

coordinate  
system



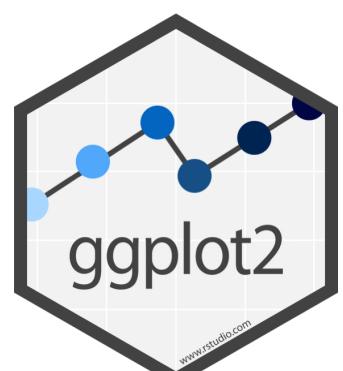
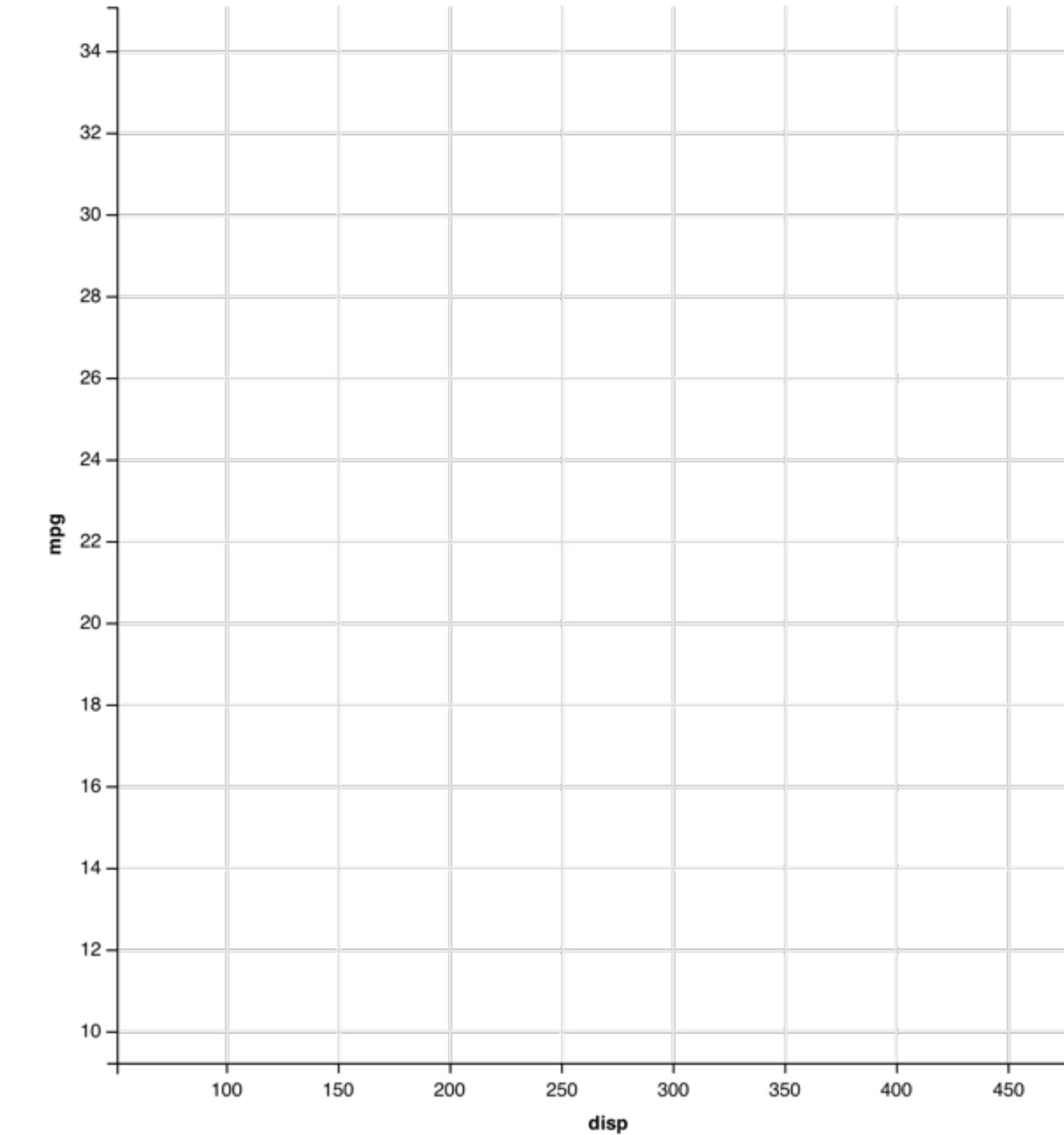
# properties

mpg	cyl	disp	hp
21.0	6	160.0	2
21.0	6	160.0	2
22.8	4	108.0	1
21.4	6	258.0	2
18.7	8	360.0	3
18.1	6	225.0	2
14.3	8	360.0	5
24.4	4	146.7	1
22.8	4	140.8	1
19.2	6	167.6	2
17.8	6	167.6	2
16.4	8	275.8	3
17.3	8	275.8	3
15.2	8	275.8	3
10.4	8	472.0	4
10.4	8	460.0	4
14.7	8	440.0	4
32.4	4	78.7	1
30.4	4	75.7	1
33.9	4	71.1	1

data

geom

coordinate  
system



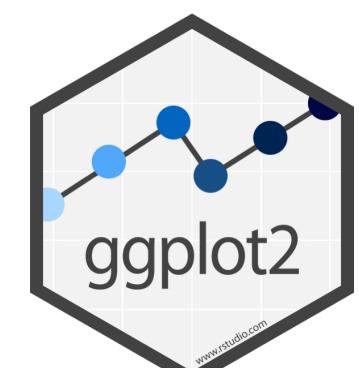
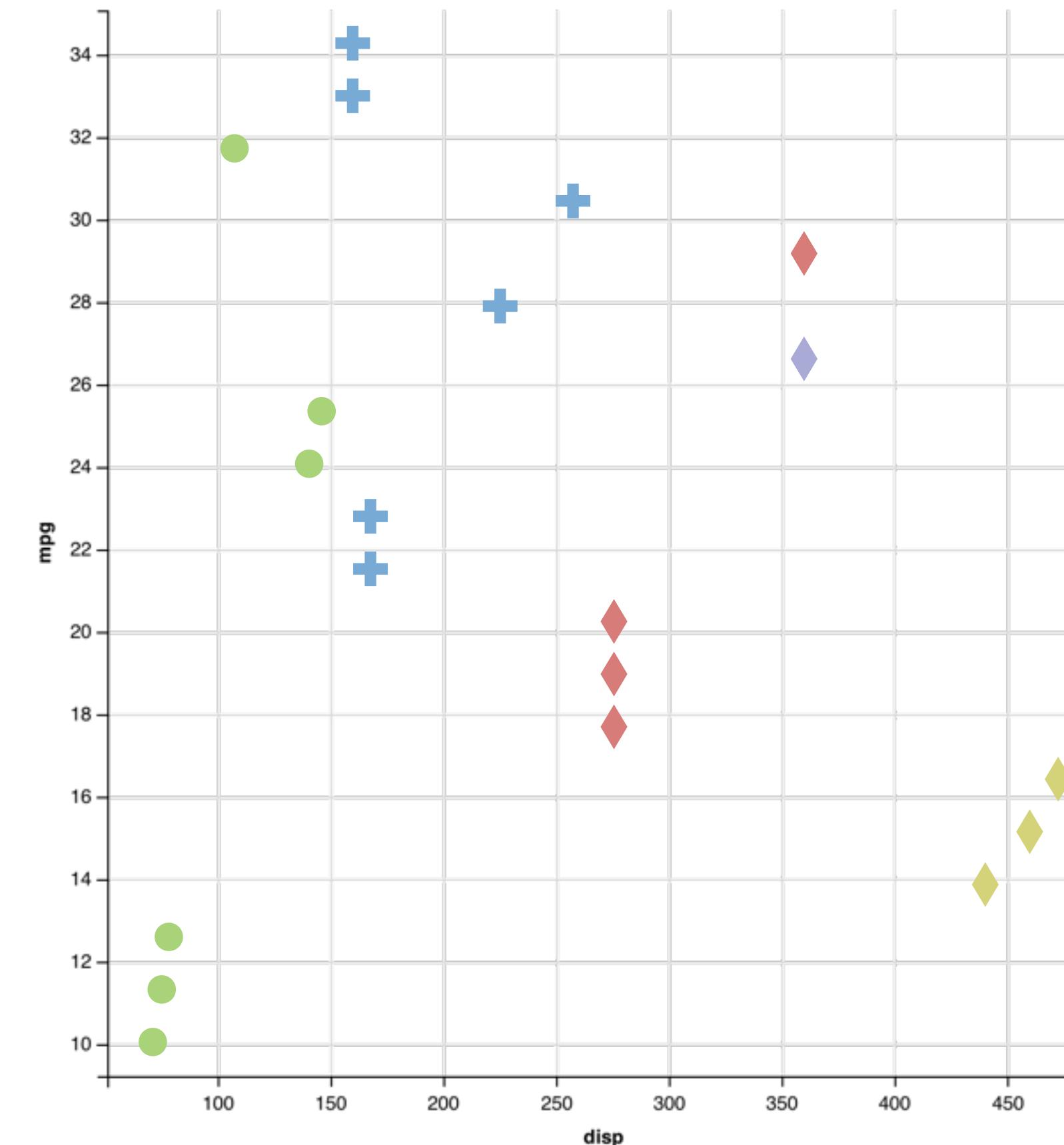
# properties

	y	shape	x	fill
	mpg	cyl	disp	hp
21.0	6	160.0	2	
21.0	6	160.0	2	
22.8	4	108.0	1	
21.4	6	258.0	2	
18.7	8	360.0	3	
18.1	6	225.0	2	
14.3	8	360.0	5	
24.4	4	146.7	1	
22.8	4	140.8	1	
19.2	6	167.6	2	
17.8	6	167.6	2	
16.4	8	275.8	3	
17.3	8	275.8	3	
15.2	8	275.8	3	
10.4	8	472.0	4	
10.4	8	460.0	4	
14.7	8	440.0	4	
32.4	4	78.7	1	
30.4	4	75.7	1	
33.9	4	71.1	1	

data

geom

coordinate  
system

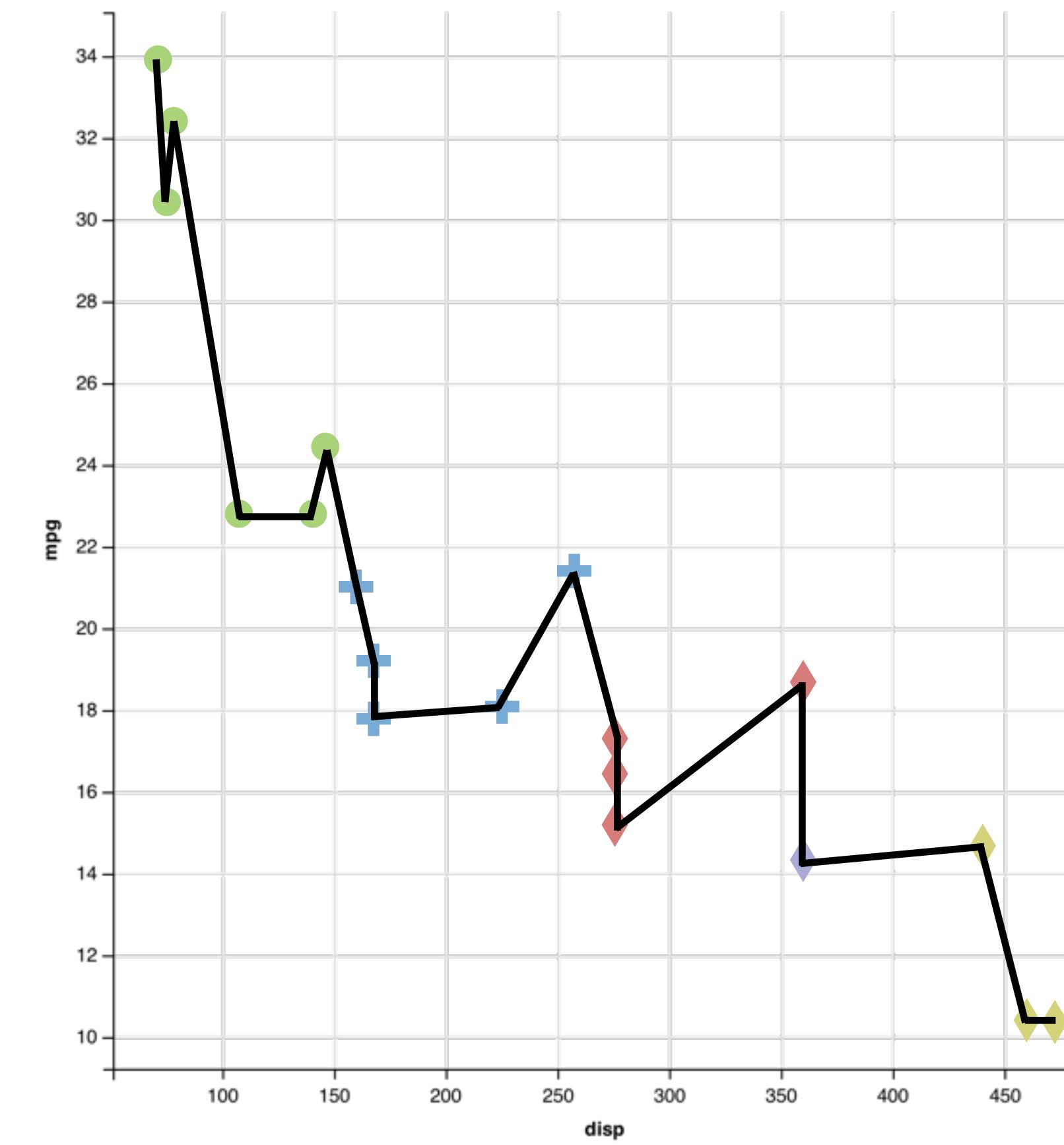


# properties

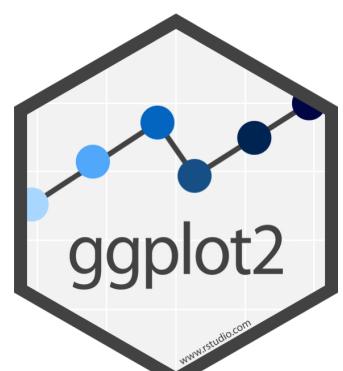
	y	shape	x	fill
	mpg	cyl	disp	hp
21.0	6	160.0	2	
21.0	6	160.0	2	
22.8	4	108.0	1	
21.4	6	258.0	2	
18.7	8	360.0	3	
18.1	6	225.0	2	
14.3	8	360.0	5	
24.4	4	146.7	1	
22.8	4	140.8	1	
19.2	6	167.6	2	
17.8	6	167.6	2	
16.4	8	275.8	3	
17.3	8	275.8	3	
15.2	8	275.8	3	
10.4	8	472.0	4	
10.4	8	460.0	4	
14.7	8	440.0	4	
32.4	4	78.7	1	
30.4	4	75.7	1	
33.9	4	71.1	1	

data

geom  
points  
lines



coordinate  
system



# properties

Y  
↑  
mpg cyl disp hp

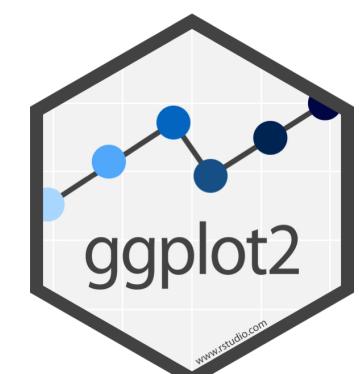
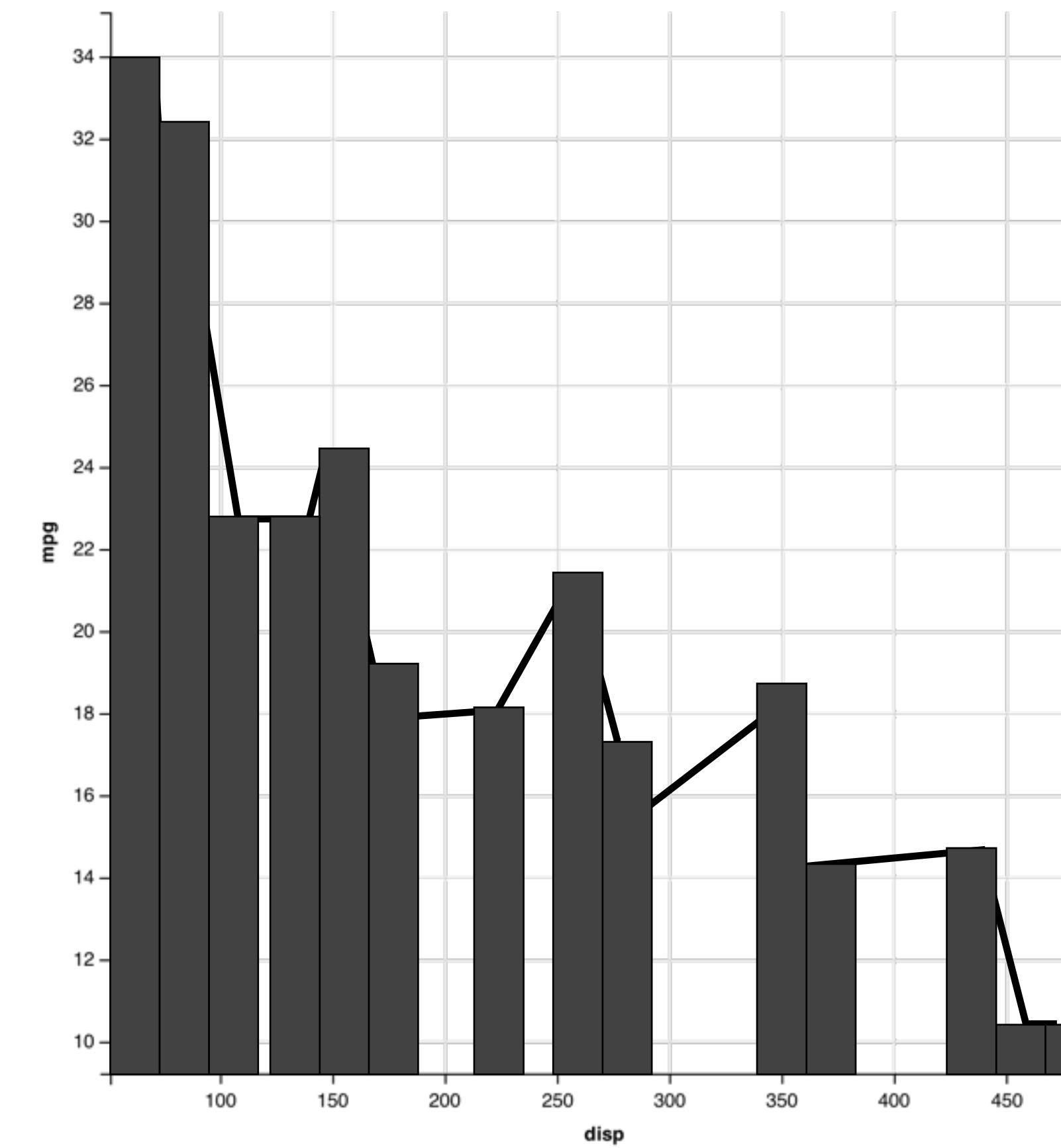
X  
↓  
mpg cyl disp hp

mpg	cyl	disp	hp
21.0	6	160.0	2
21.0	6	160.0	2
22.8	4	108.0	1
21.4	6	258.0	2
18.7	8	360.0	3
18.1	6	225.0	2
14.3	8	360.0	5
24.4	4	146.7	1
22.8	4	140.8	1
19.2	6	167.6	2
17.8	6	167.6	2
16.4	8	275.8	3
17.3	8	275.8	3
15.2	8	275.8	3
10.4	8	472.0	4
10.4	8	460.0	4
14.7	8	440.0	4
32.4	4	78.7	1
30.4	4	75.7	1
33.9	4	71.1	1

data

geom  
points  
lines  
bars

coordinate  
system



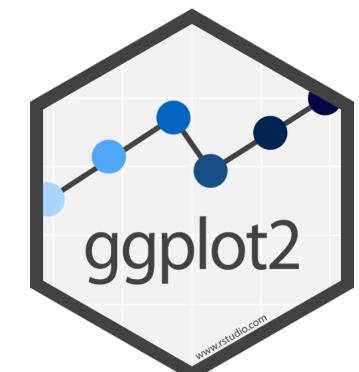
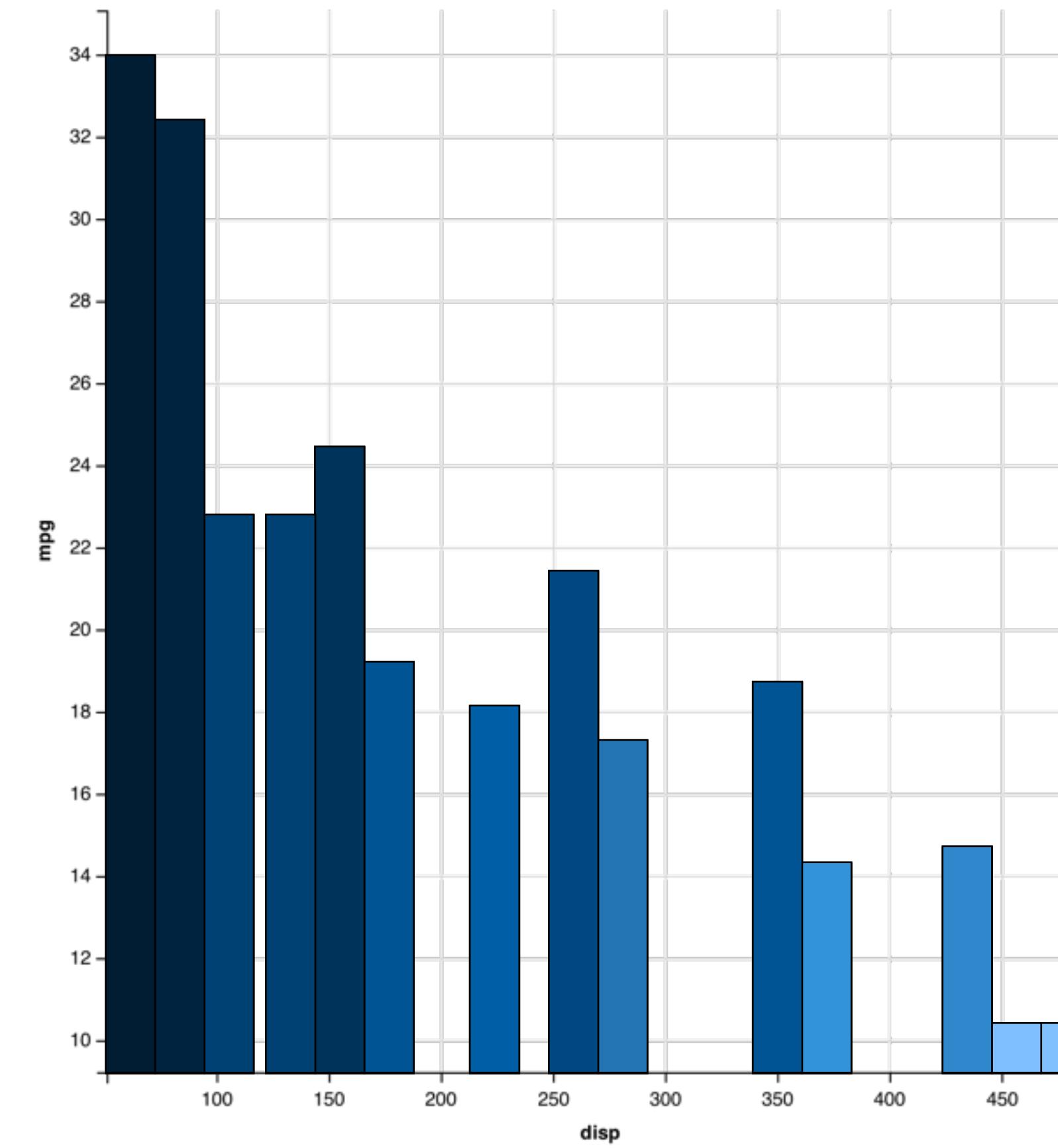
# properties

mpg	cyl	disp	hp
21.0	6	160.0	2
21.0	6	160.0	2
22.8	4	108.0	1
21.4	6	258.0	2
18.7	8	360.0	3
18.1	6	225.0	2
14.3	8	360.0	5
24.4	4	146.7	1
22.8	4	140.8	1
19.2	6	167.6	2
17.8	6	167.6	2
16.4	8	275.8	3
17.3	8	275.8	3
15.2	8	275.8	3
10.4	8	472.0	4
10.4	8	460.0	4
14.7	8	440.0	4
32.4	4	78.7	1
30.4	4	75.7	1
33.9	4	71.1	1

data

geom  
points  
lines  
bars

coordinate  
system



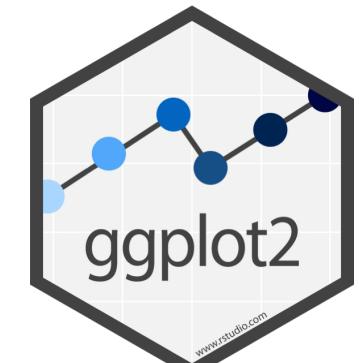
# To make a graph

mpg	cyl	disp	hp	geom
21.0	6	160.0	2	•
21.0	6	160.0	2	•
22.8	4	108.0	1	•
21.4	6	258.0	2	•
18.7	8	360.0	3	•
18.1	6	225.0	2	•
14.3	8	360.0	5	•
24.4	4	146.7	1	•
22.8	4	140.8	1	•
19.2	6	167.6	2	•
17.8	6	167.6	2	•
16.4	8	275.8	3	•
17.3	8	275.8	3	•
15.2	8	275.8	3	•
10.4	8	472.0	4	•
10.4	8	460.0	4	•
14.7	8	440.0	4	•
32.4	4	78.7	1	•
30.4	4	75.7	1	•
33.9	4	71.1	1	•

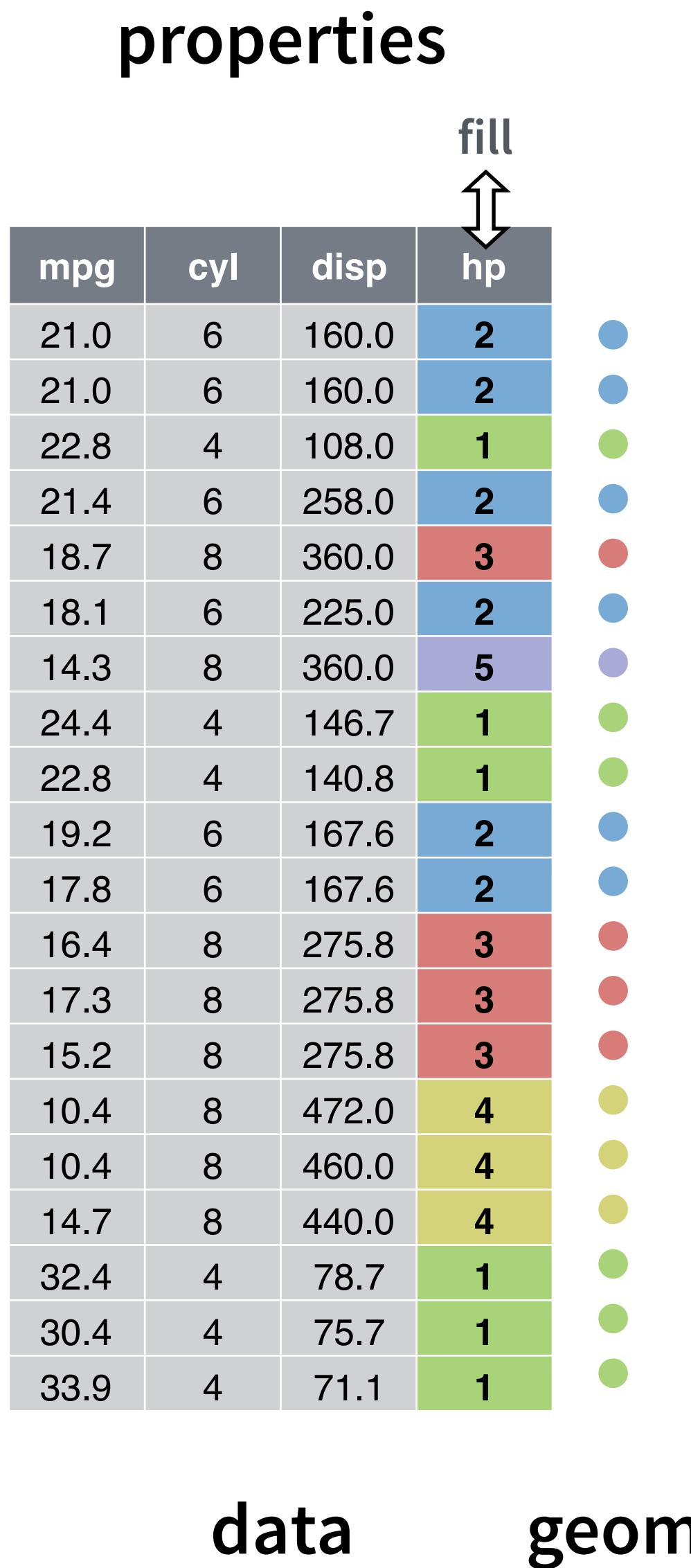
data

geom

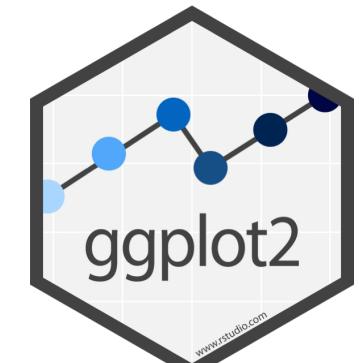
1. Display **cases** as visual objects  
(geoms)



# To make a graph

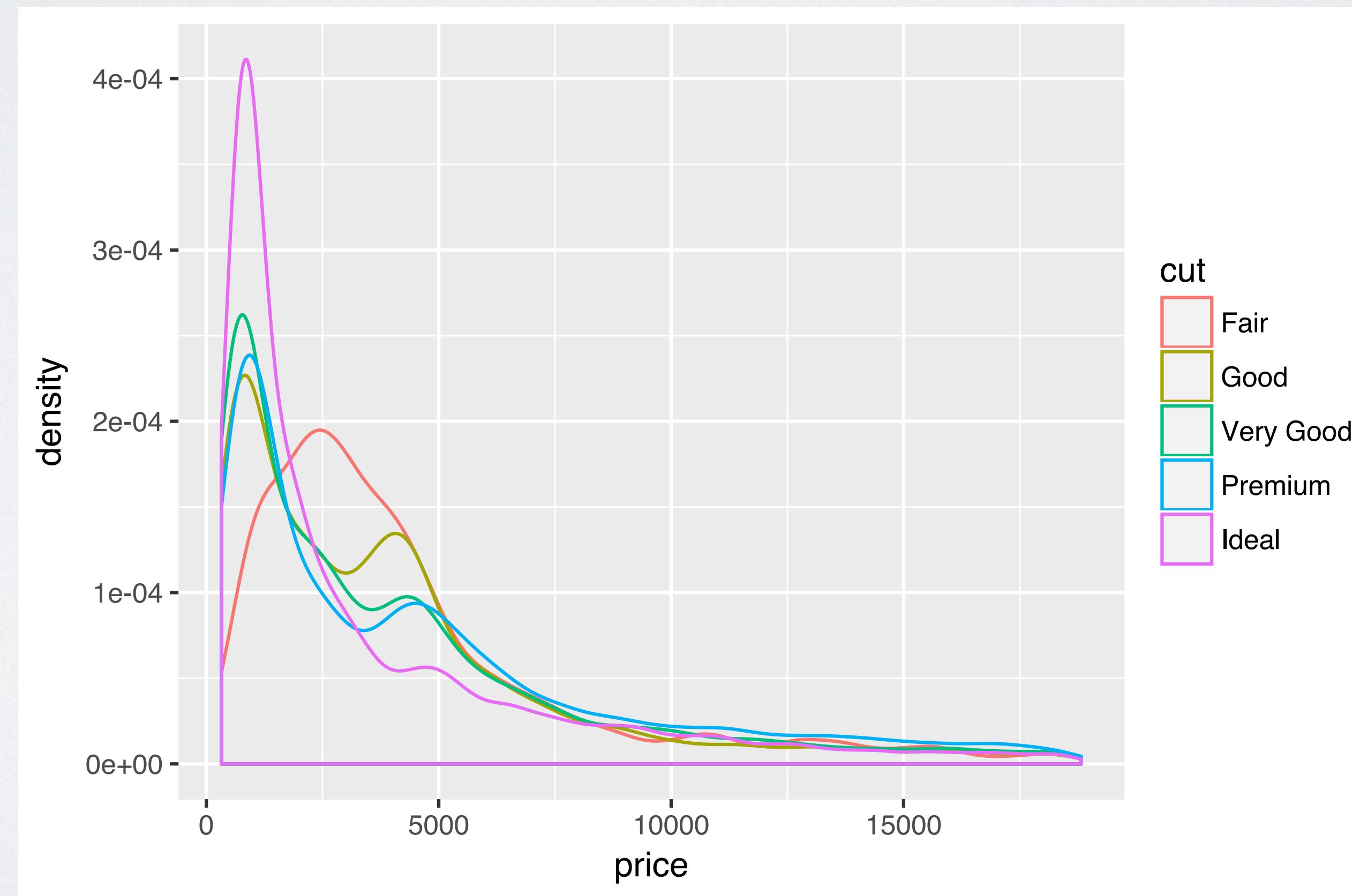


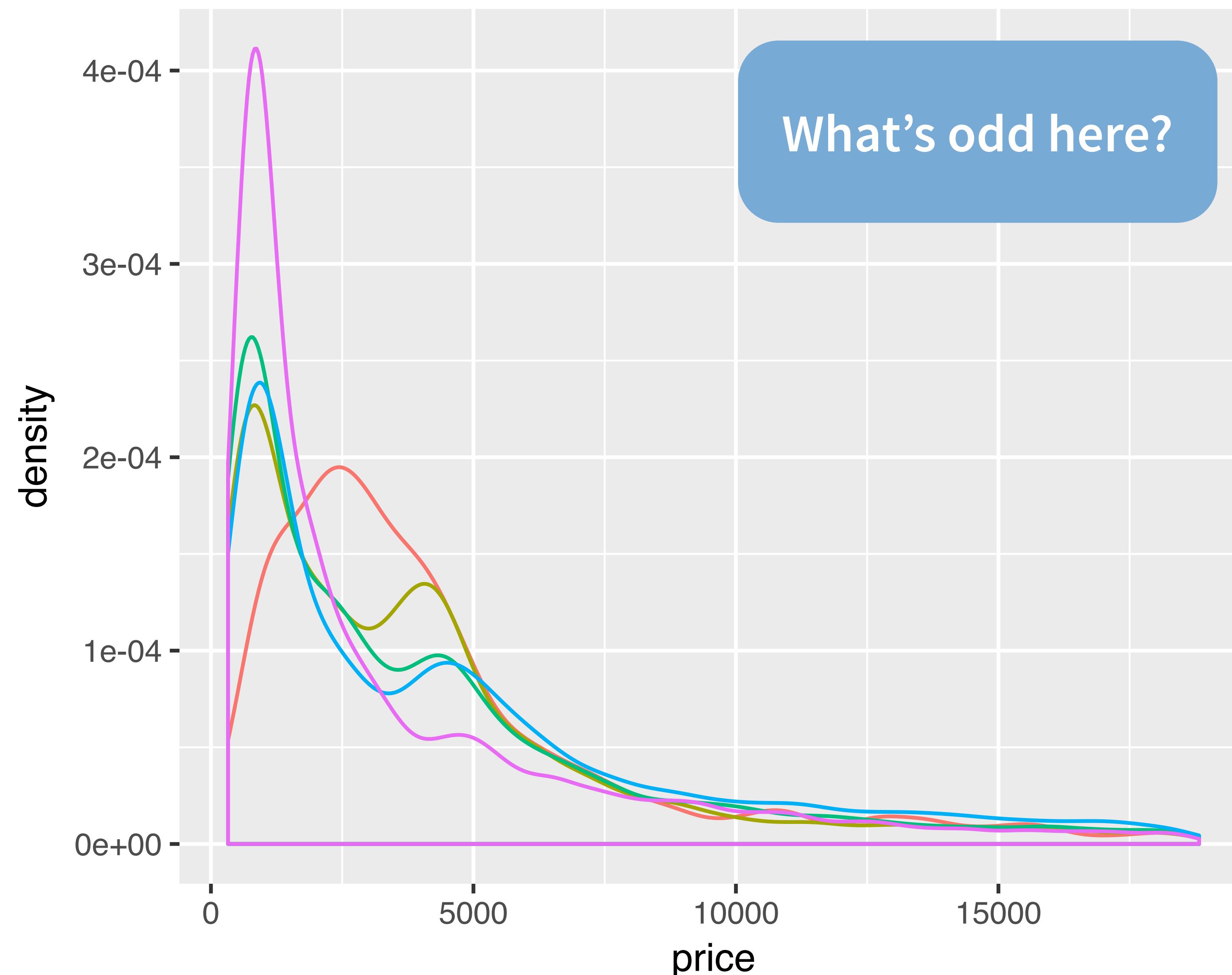
1. Display **cases** as visual objects  
(geoms)
2. Display **variables** as visual properties of the geoms  
(aesthetics)
3. Tweak details.



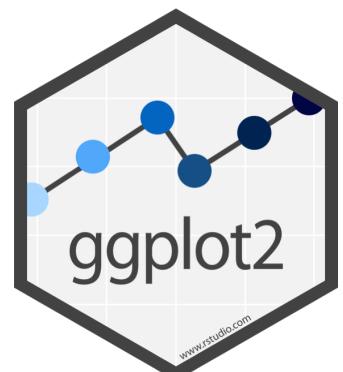
# Your turn

Make this plot of the density of diamond prices grouped and colored by cut.



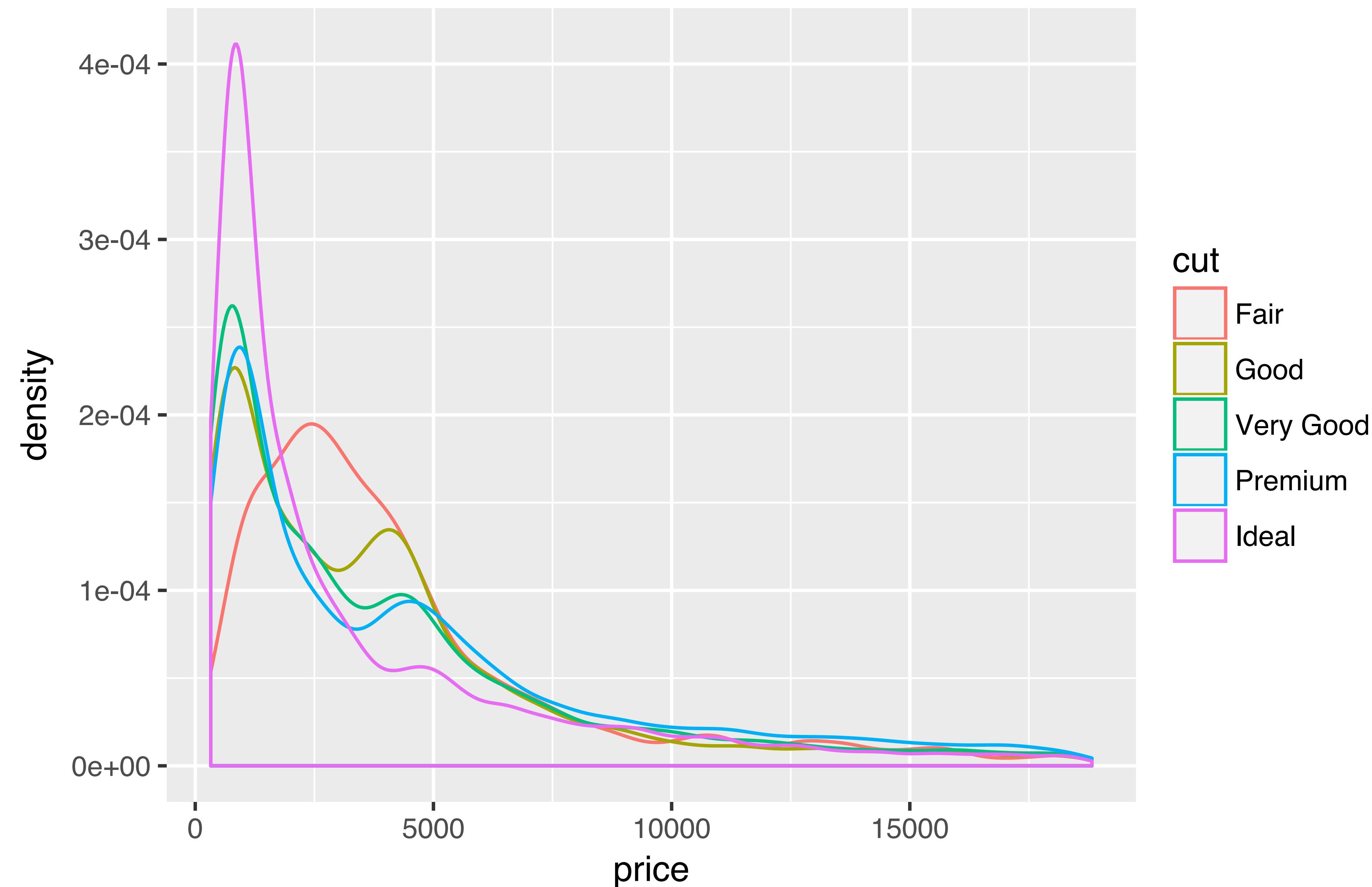


```
ggplot(diamonds, aes(x = price, color = cut)) +  
  geom_density()
```

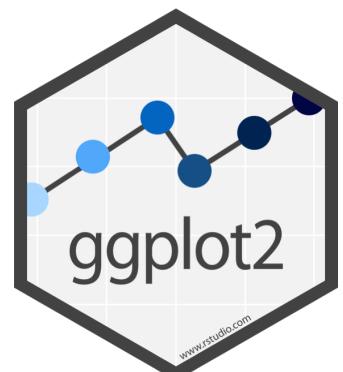


# Communication

(labels, themes, and scales)



```
p1 <- ggplot(diamonds, aes(x = price, color = cut)) +  
  geom_density()
```



# labels



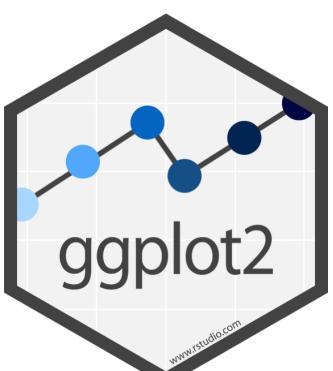
# labs()

Add titles, subtitles, and more

```
plot + labs(...)
```

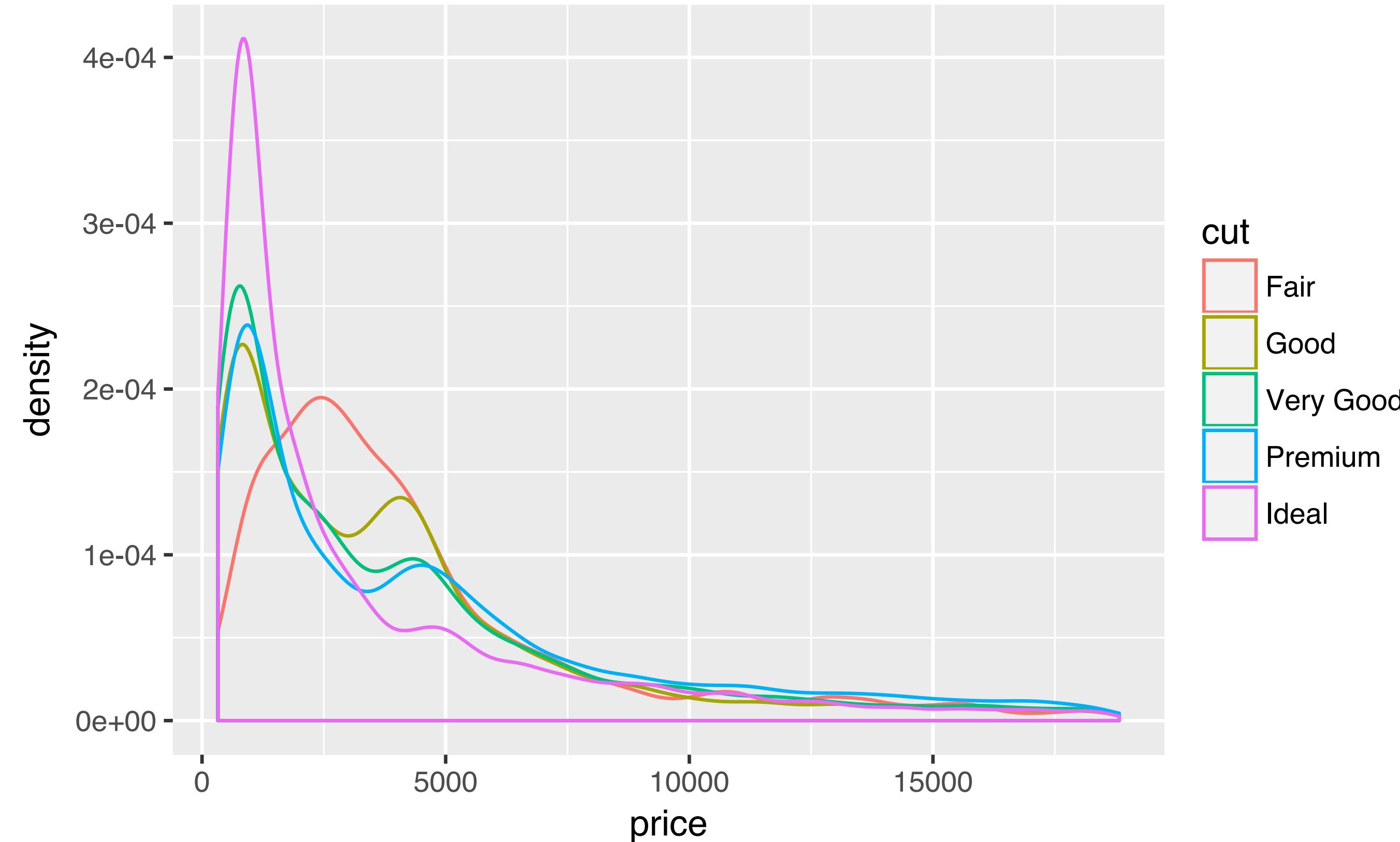
**plot to add  
labels to**

**Character strings  
to add by type**



# titles

Poorly cut diamonds appear to fetch unexpectedly high prices

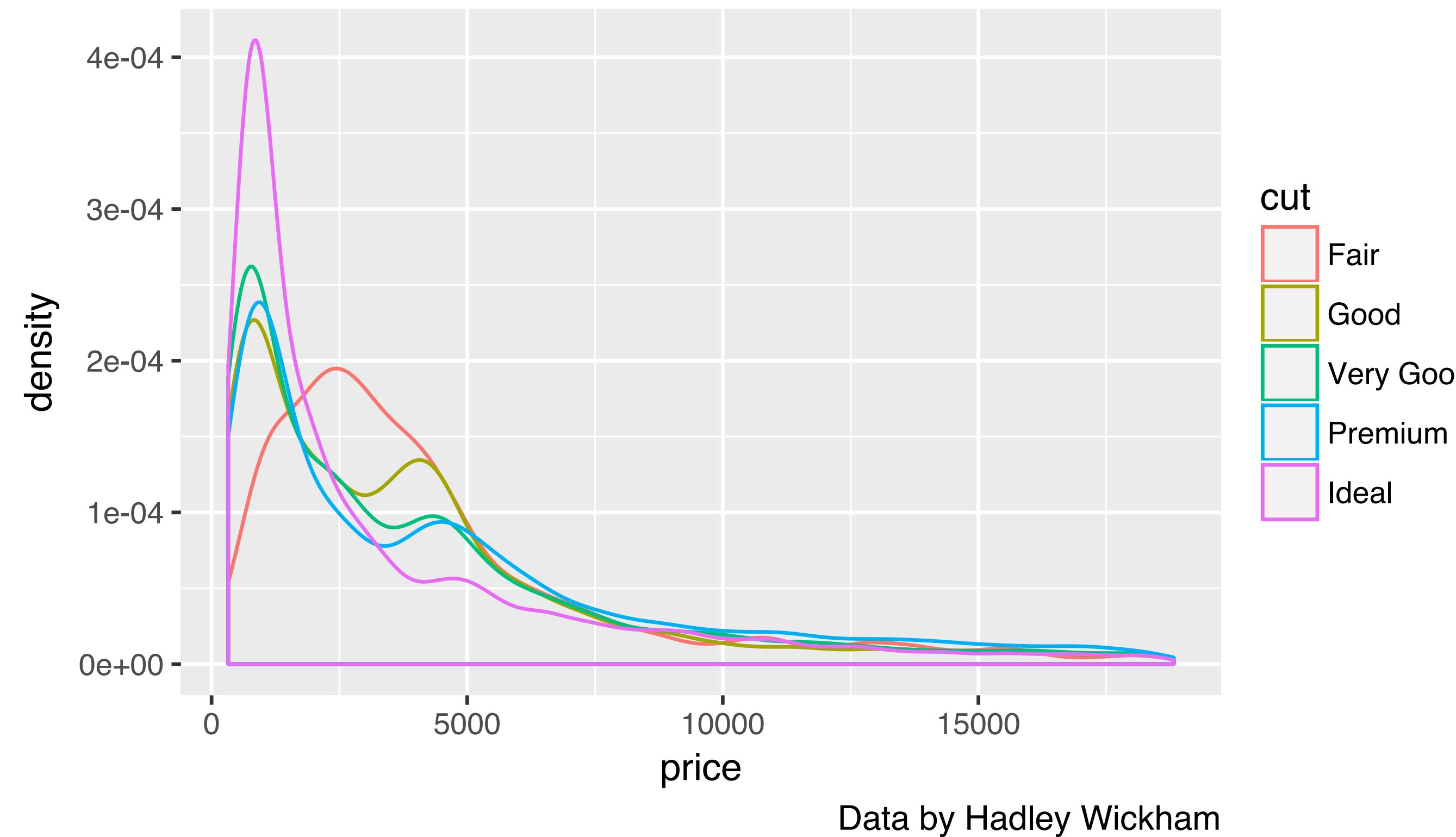


```
p1 + labs(title = "Poorly cut diamonds appear to fetch  
unexpectedly high prices")
```

# subtitles and captions

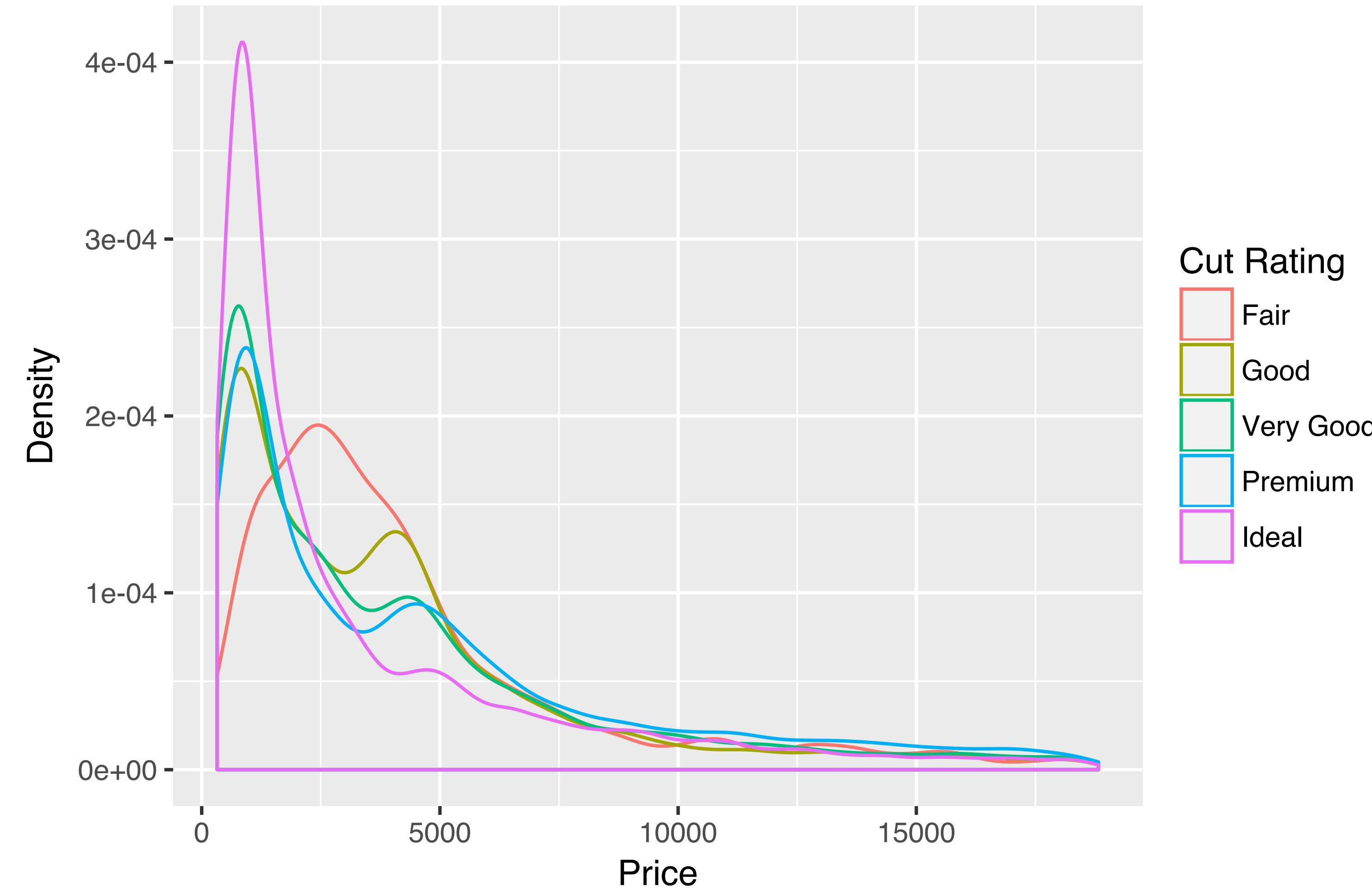
Poorly cut diamonds appear to fetch unexpectedly high prices

The lowest cut rating is associated with the highest mode price



```
p1 + labs(title = "Poorly cut diamonds appear to fetch unexpectedly high prices",  
          subtitle = "The lowest cut rating is associated with the highest mode price",  
          caption = "Data by Hadley Wickham")
```

# axis labels and legend titles



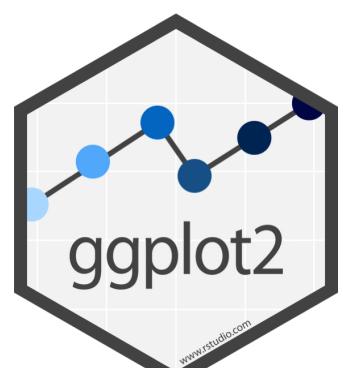
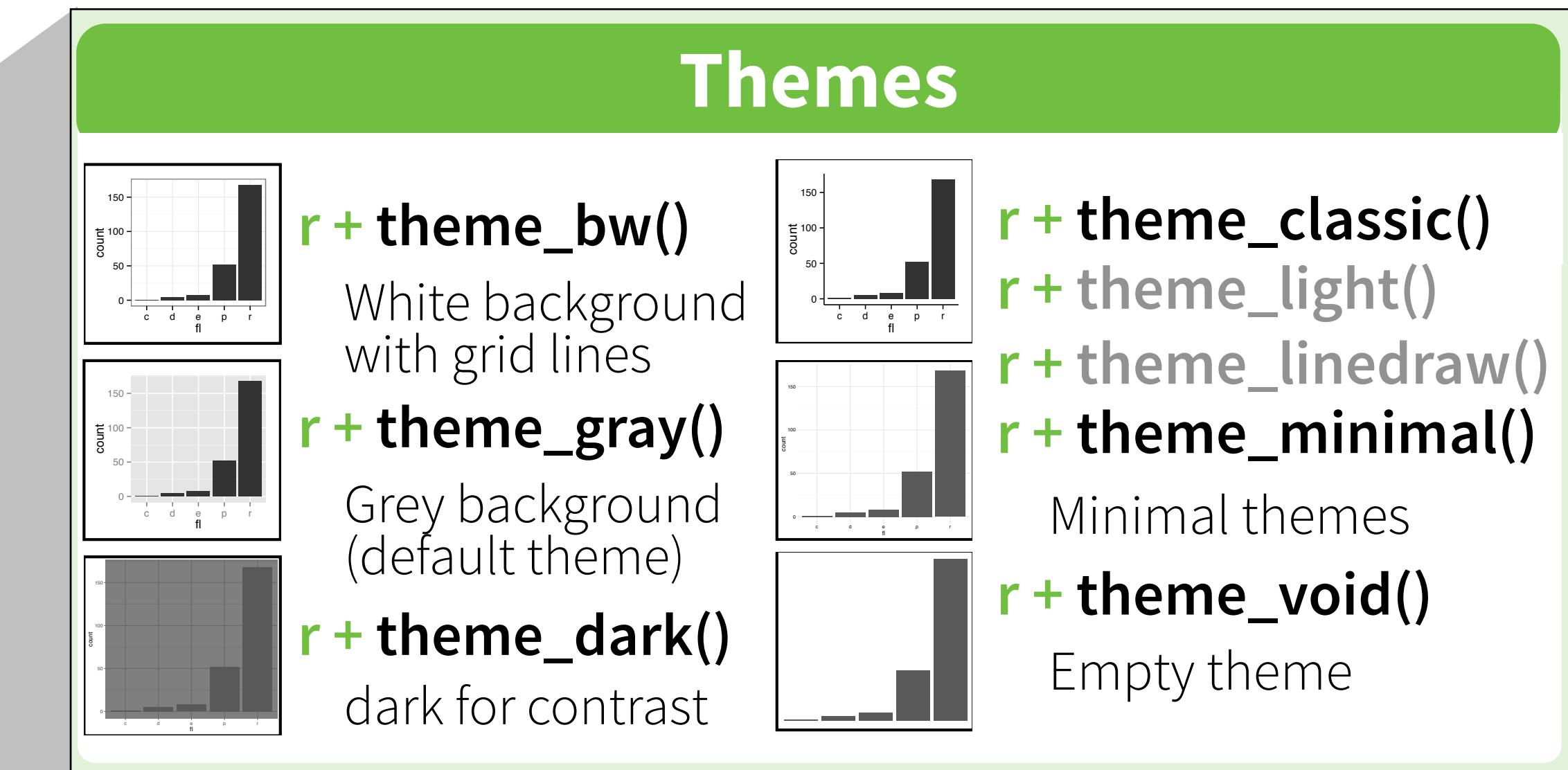
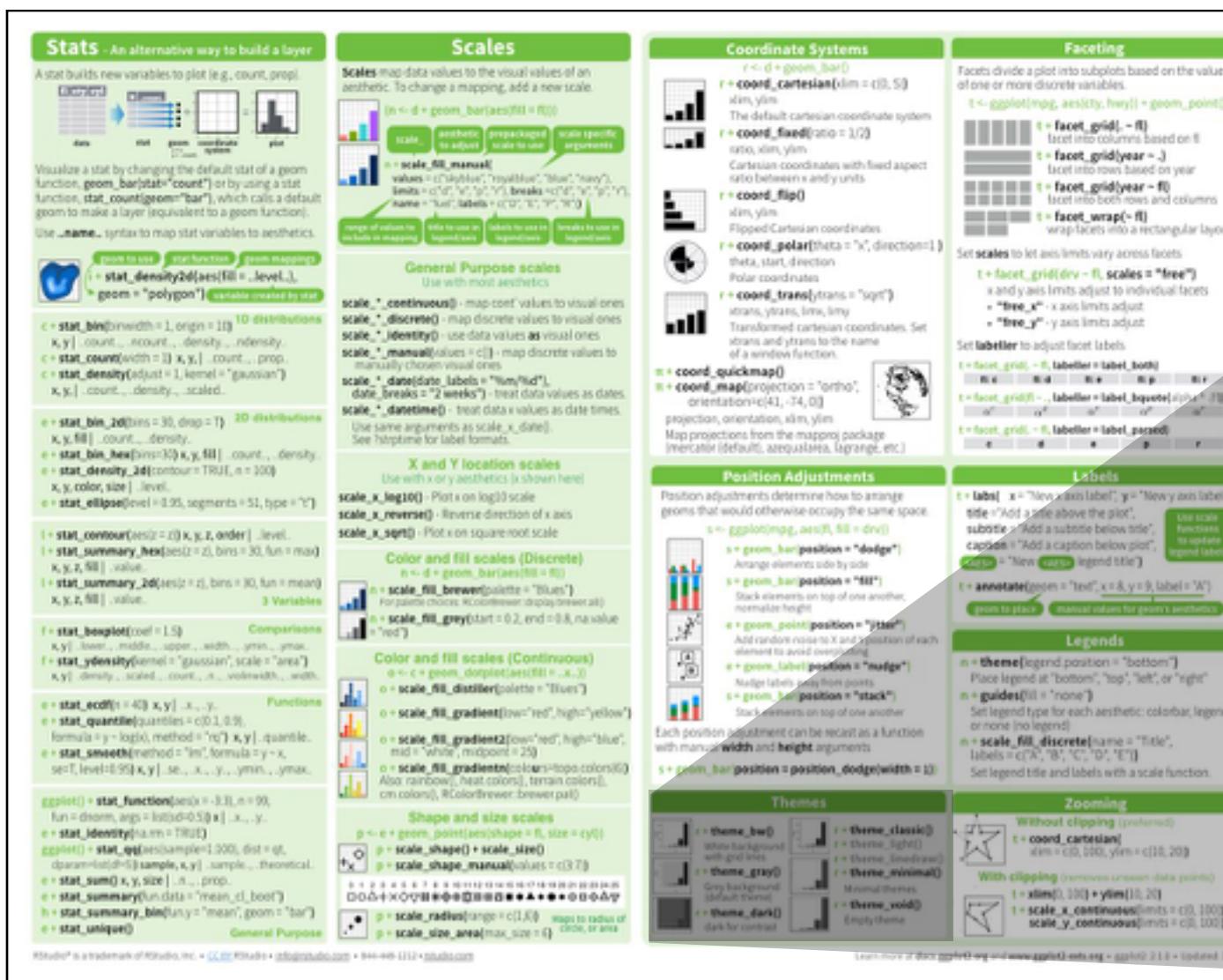
```
p1 + labs(x = "Price", y = "Density", color = "Cut Rating")
```

# themes



# theme functions

Each supplies a different appearance for non-data elements



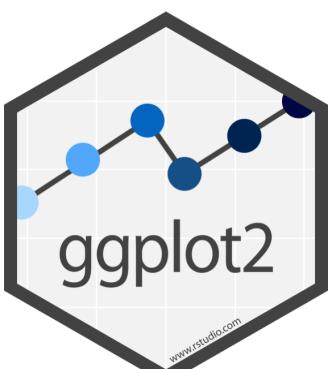
# theme\_\*( )

Change appearance of non-data elements

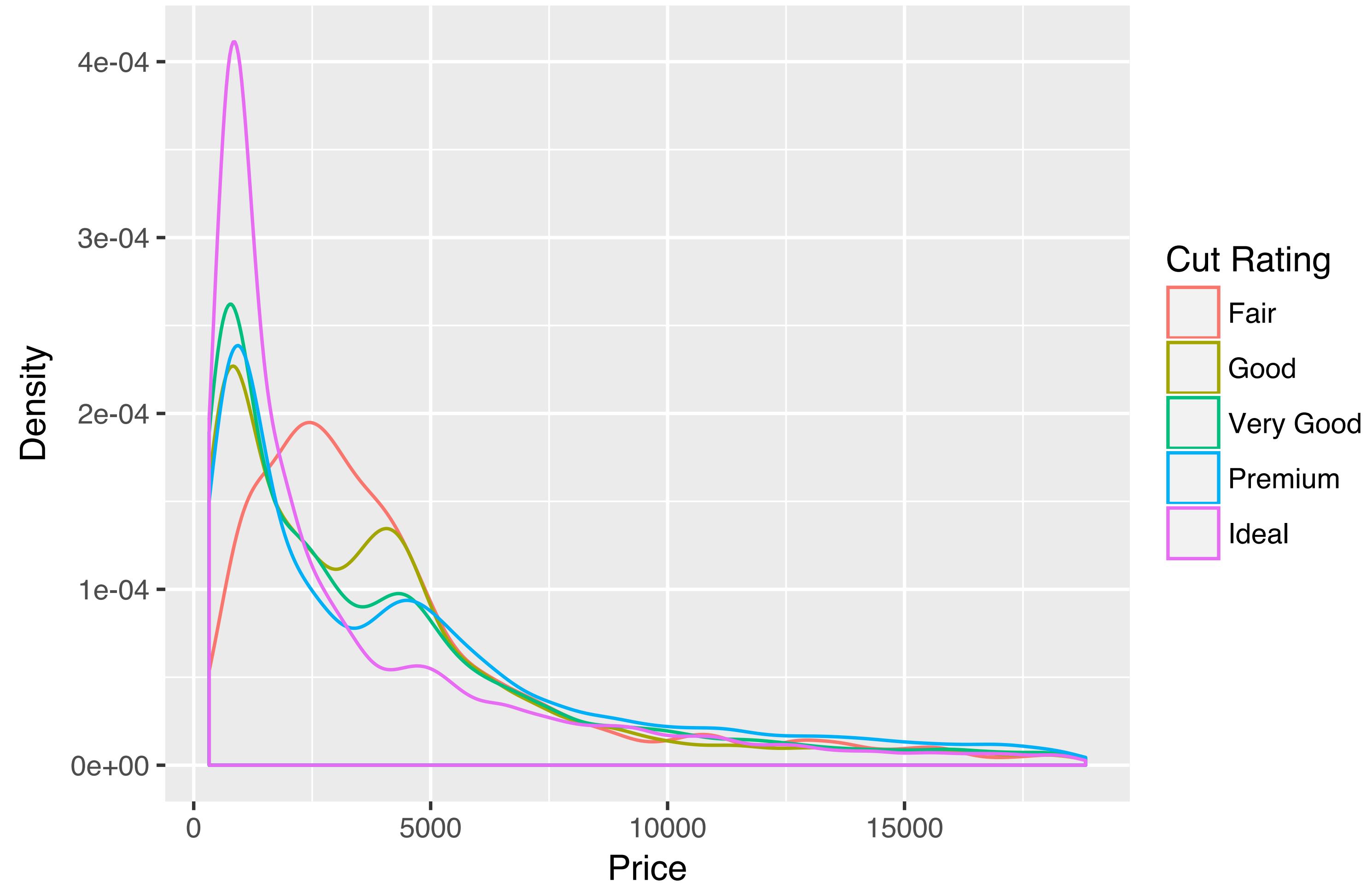
```
plot + theme_bw()
```

plot to add  
theme to

function with  
prefix theme\_

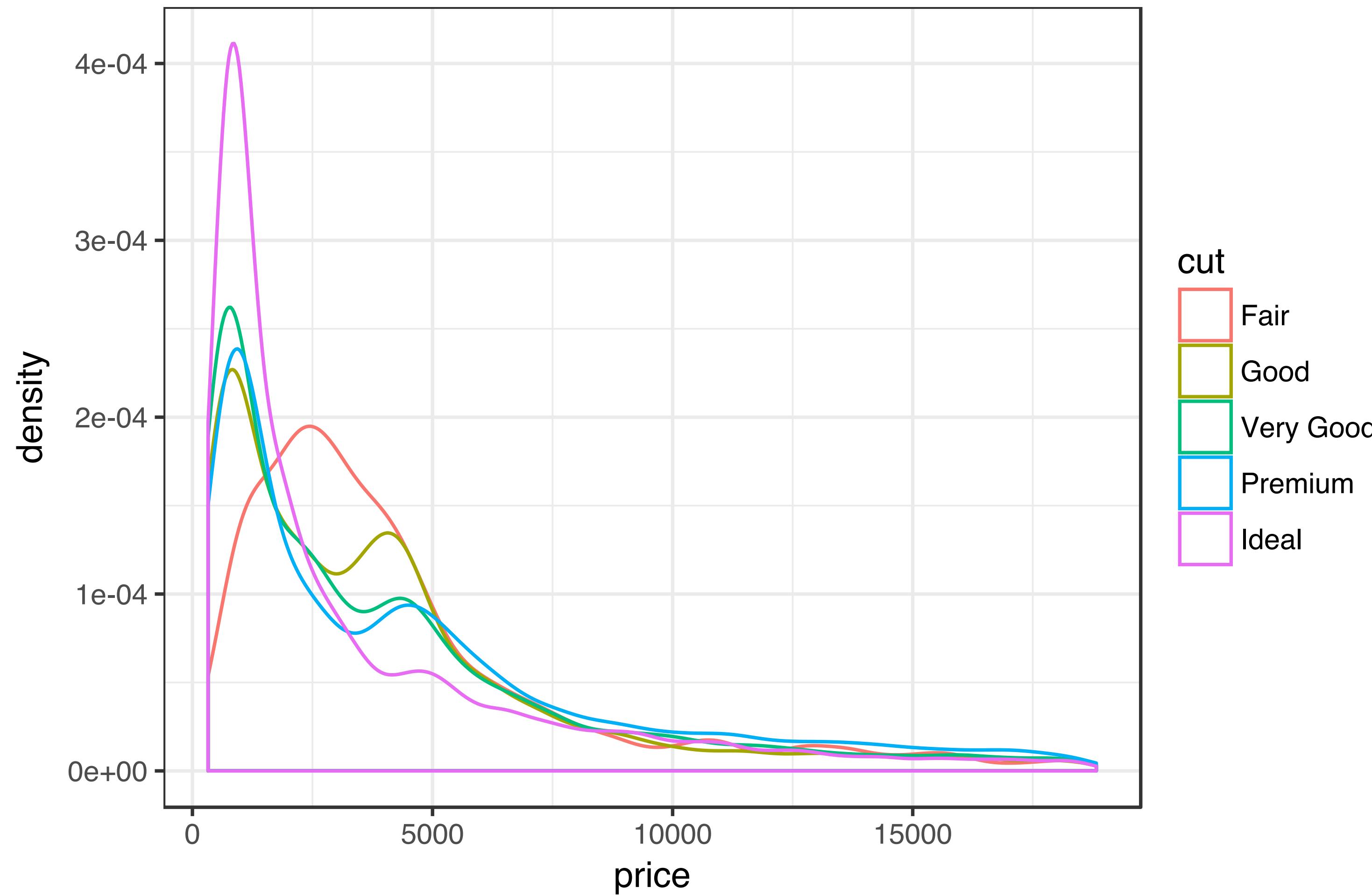


# theme\_grey()



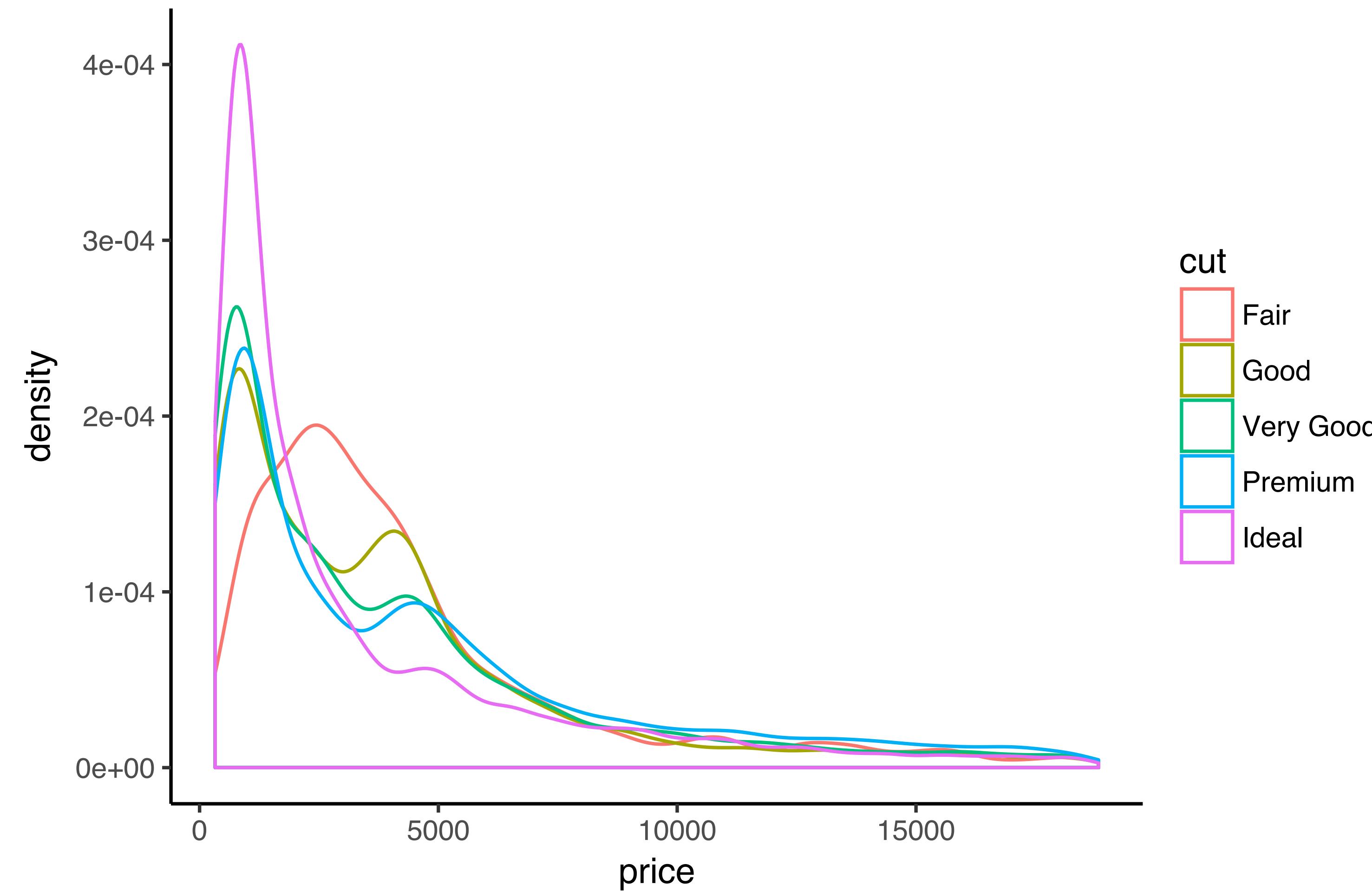
p1 + theme\_grey()

# theme\_bw()



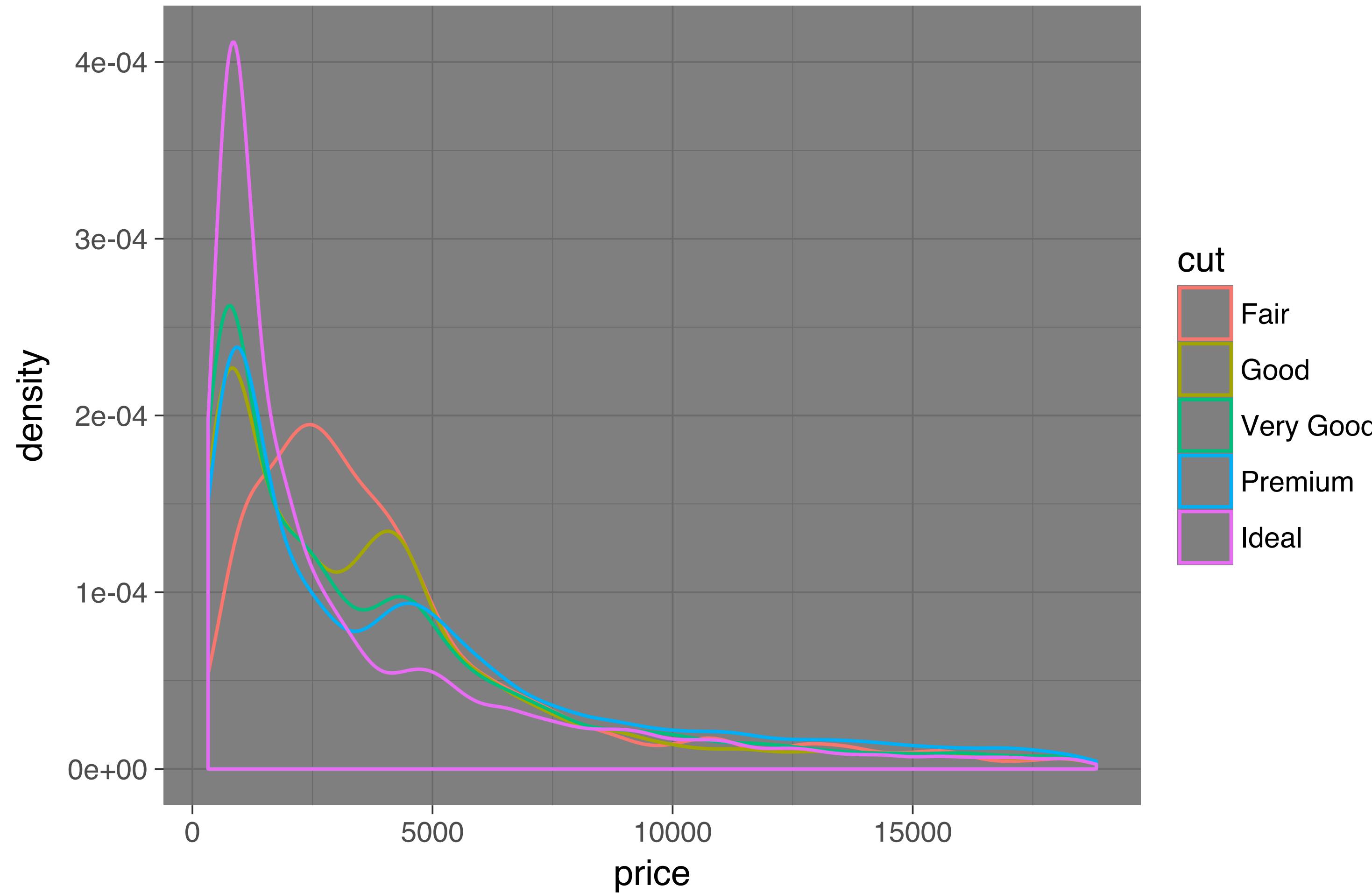
p1 + theme\_bw()

# theme\_classic()



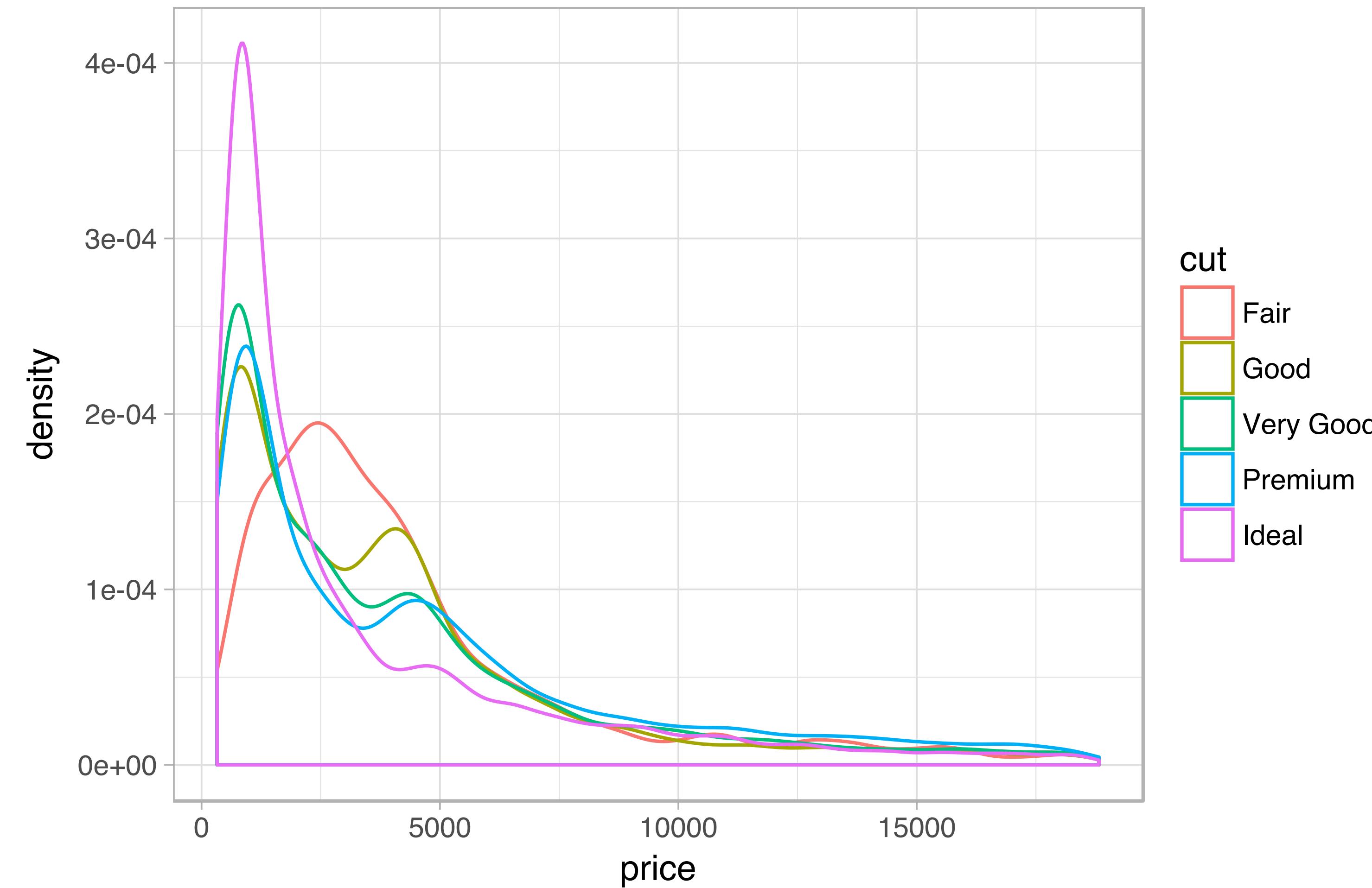
p1 + theme\_classic()

# theme\_dark()



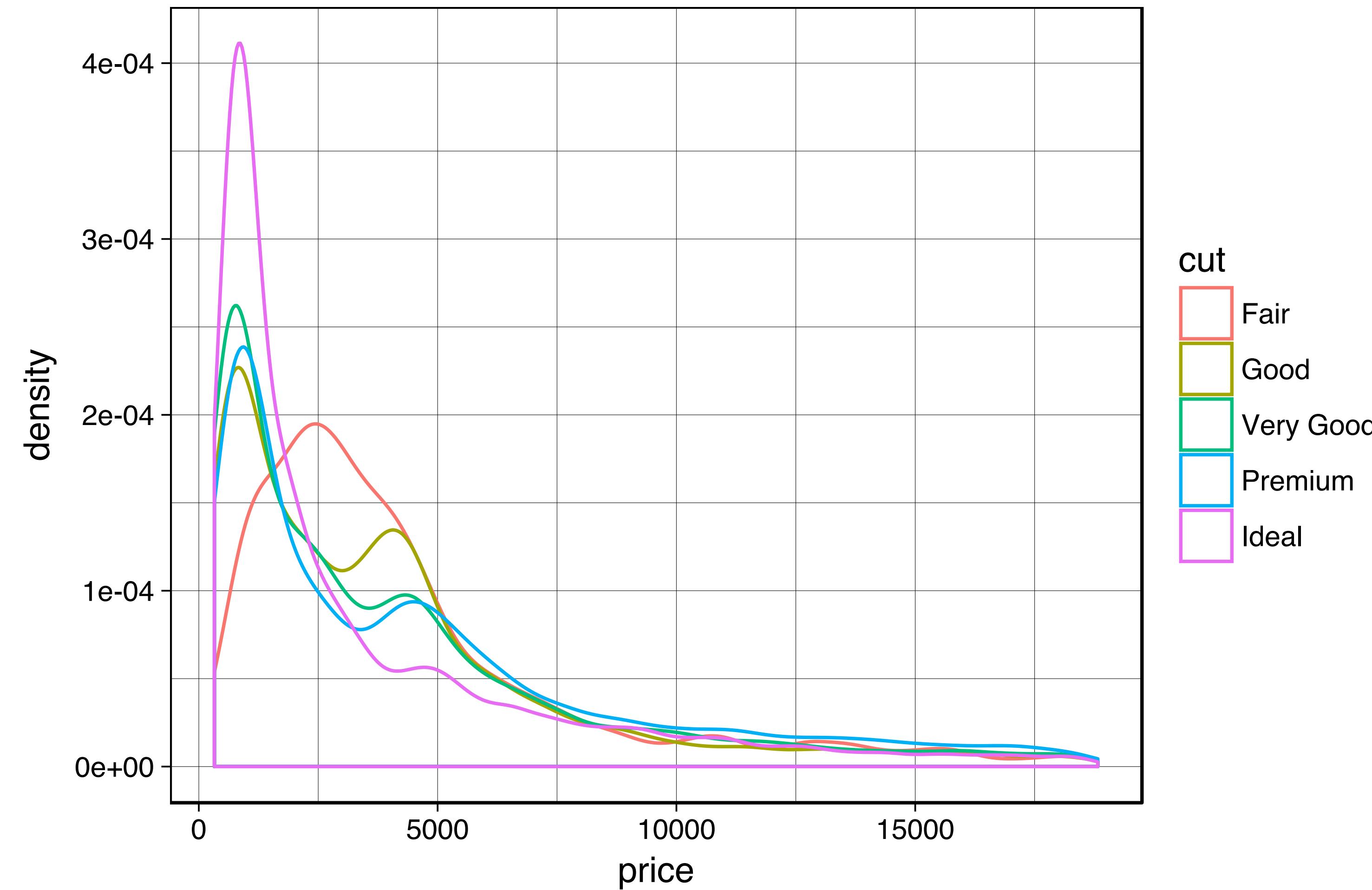
p1 + theme\_dark()

# theme\_light()



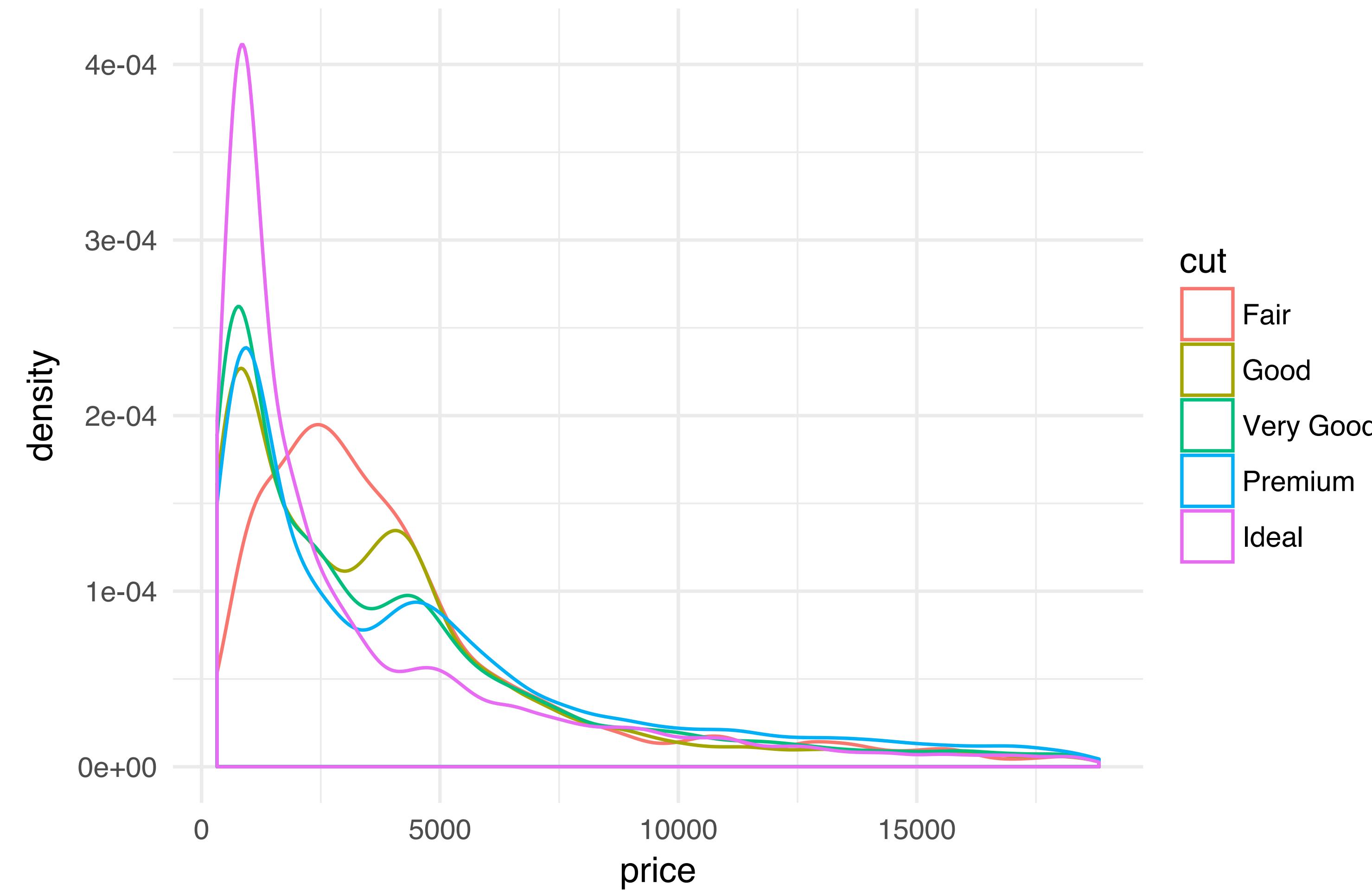
p1 + theme\_light()

# theme\_linedraw()



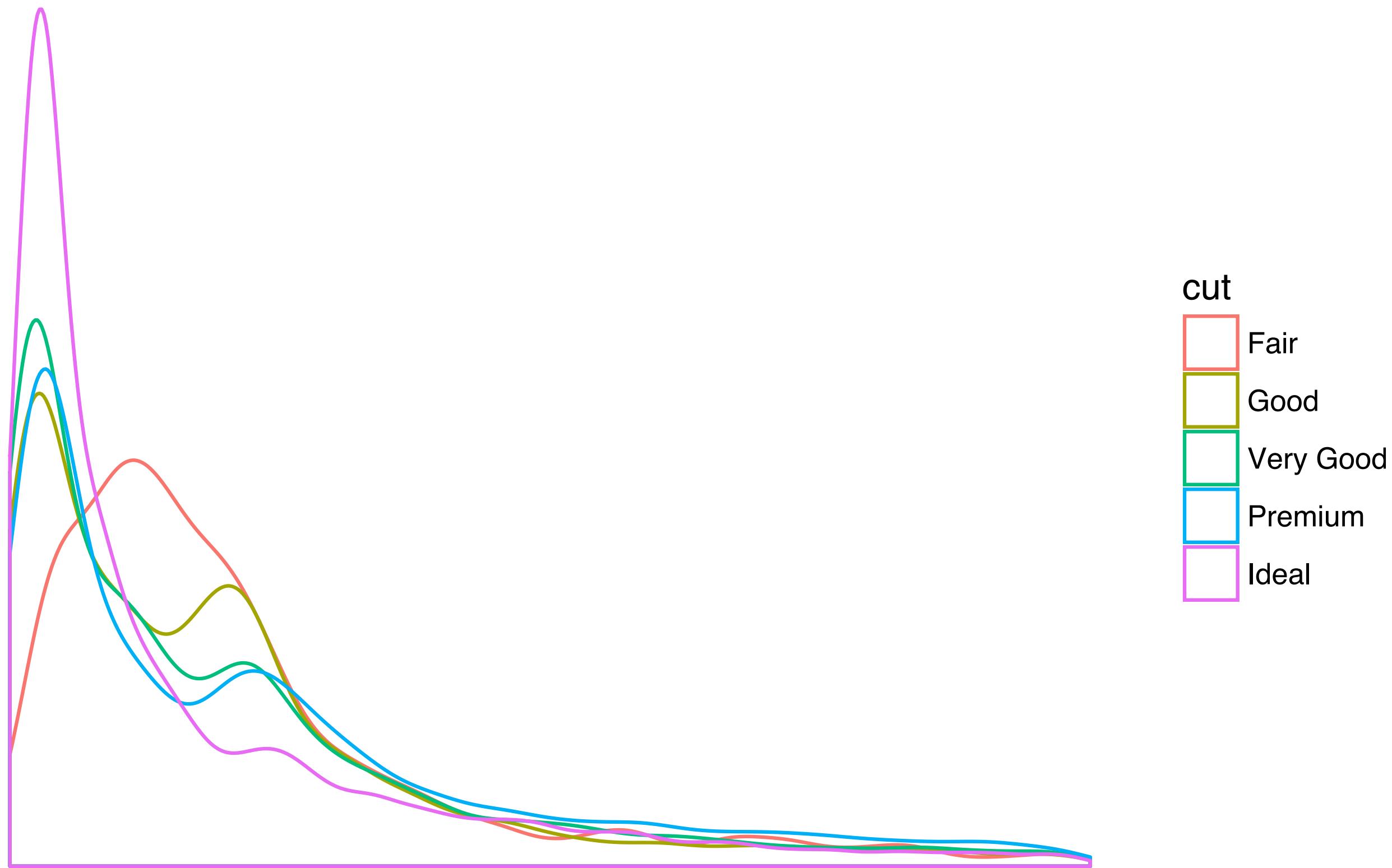
p1 + theme\_linedraw()

# theme\_minimal()



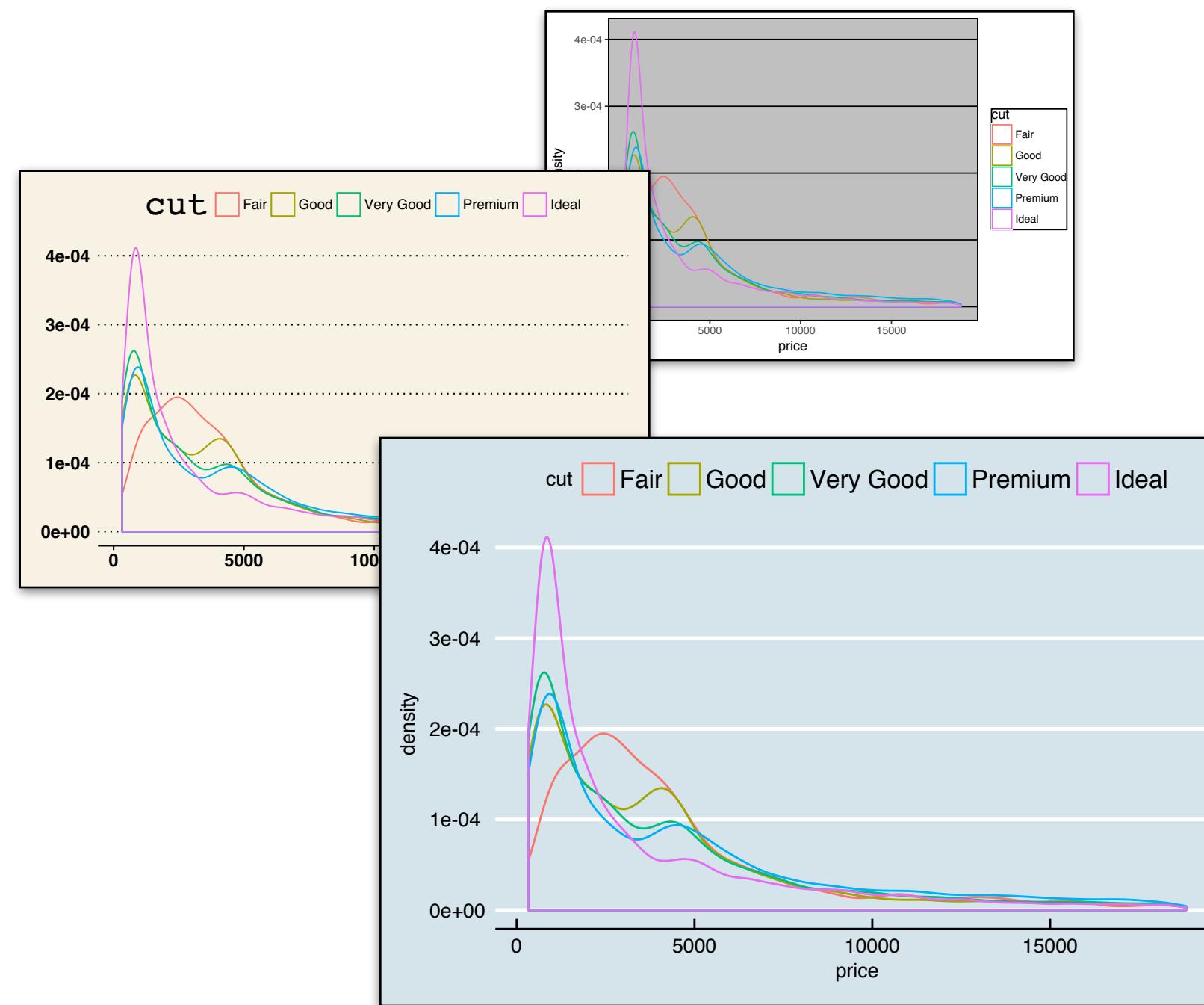
p1 + theme\_minimal()

# theme\_void()



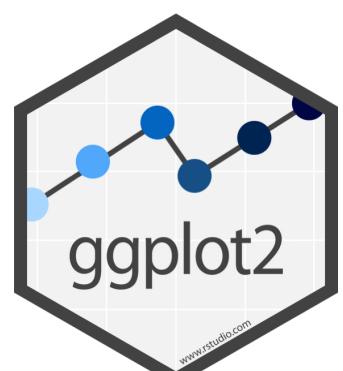
p1 + theme\_void()

# ggthemes



A package of additional ggplot2 themes

```
# install.packages("ggthemes")
library(ggthemes)
```

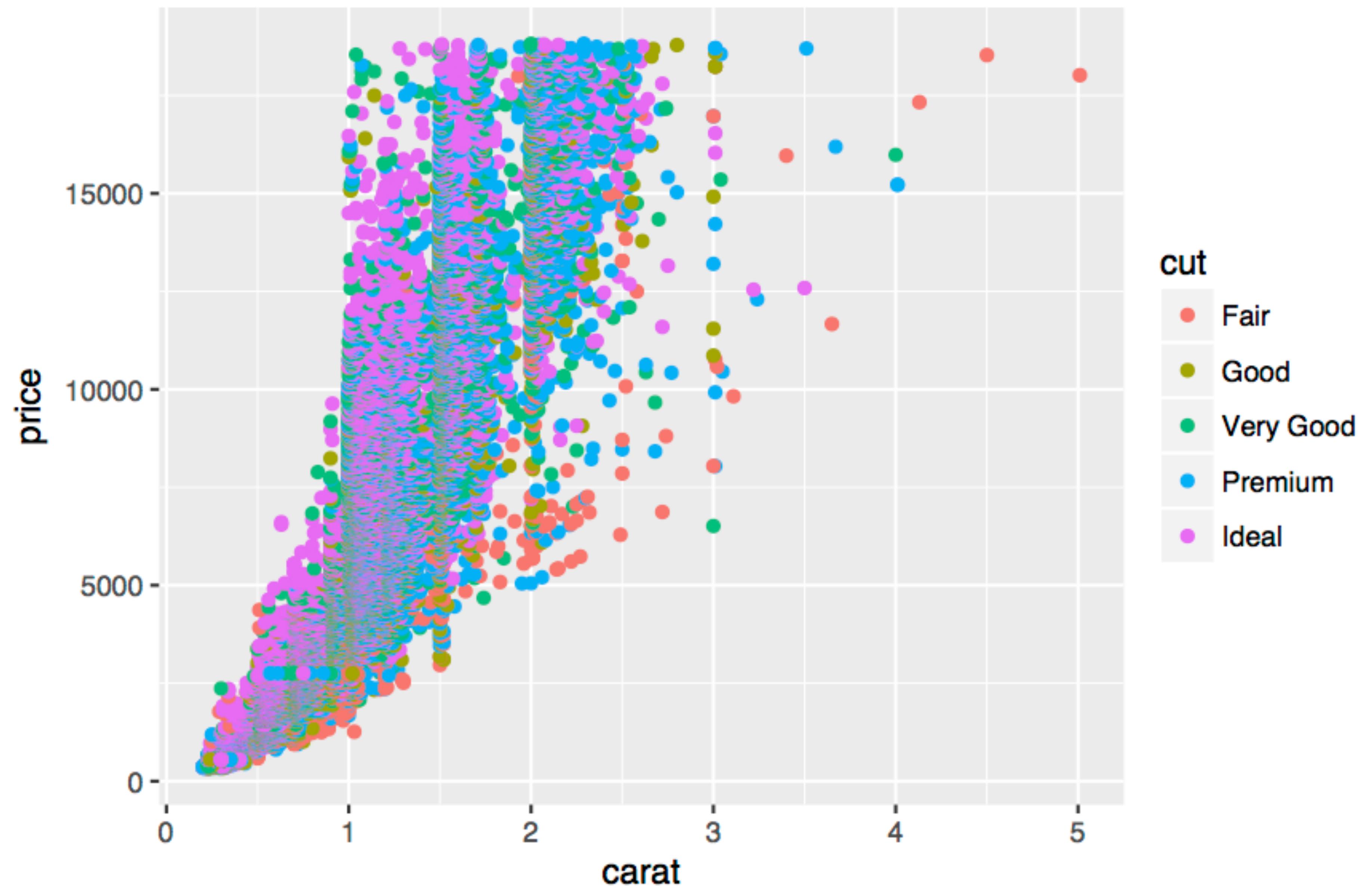


# scales

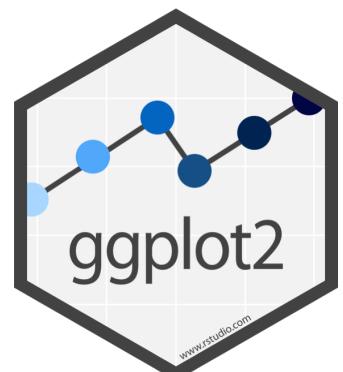


# Quiz

Why do you think poorly cut diamonds are associated with higher prices?



```
s <- ggplot(diamonds, aes(carat, price)) +  
  geom_point(aes(color = cut))
```



# scale functions

# Scales

**Scales** map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

(`n <- d + geom_bar(aes(fill = fl))`)

scale\_

aesthetic to adjust

prepackaged scale to use

scale specific arguments

`n + scale_fill_manual(`

`values = c("skyblue", "royalblue", "blue", "navy"),  
limits = c("d", "e", "p", "r"), breaks = c("d", "e", "p", "r")  
name = "fuel", labels = c("D", "E", "P", "R"))`

range of values to include in mapping

title to use in legend/axis

labels to use in legend/axis

breaks to use in legend/axis

## General Purpose scales

Use with most aesthetics

`scale_*_continuous()` - map cont' values to visual ones

`scale_*_discrete()` - map discrete values to visual ones

`scale_*_identity()` - use data values as visual ones

`scale_*_manual(values = c())` - map discrete values to manually chosen visual ones

`scale_*_date(date_labels = "%m/%d"),  
date_breaks = "2 weeks")` - treat data values as dates.

`scale_*_datetime()` - treat data x values as date times.

Use same arguments as `scale_x_date()`.

See `?strptime` for label formats.

## X and Y location scales

Use with x or y aesthetics (x shown here)

`scale_x_log10()` - Plot x on log10 scale

`scale_x_reverse()` - Reverse direction of x axis

`scale_x_sqrt()` - Plot x on square root scale

## Color and fill scales (Discrete)

`n <- d + geom_bar(aes(fill = fl))`

`n + scale_fill_brewer(palette = "Blues")`

For palette choices: `RColorBrewer::display.brewer.all()`

`n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")`

## Color and fill scales (Continuous)

`o <- c + geom_dotplot(aes(fill = ..x..))`

`o + scale_fill_distiller(palette = "Blues")`

`o + scale_fill_gradient(low = "red", high = "yellow")`

`o + scale_fill_gradient2(low = "red", high = "blue", mid = "white", midpoint = 25)`

`o + scale_fill_gradientn(colours = topo.colors(6))`

Also: `rainbow()`, `heat.colors()`, `terrain.colors()`, `cm.colors()`, `RColorBrewer::brewer.pal()`

## Shape and size scales

`p <- e + geom_point(aes(shape = fl, size = cyl))`

`p + scale_shape() + scale_size()`

`p + scale_shape_manual(values = c(3:7))`

`p + scale_radius(range = c(1,6))` Maps to radius of circle, or area

`p + scale_size_area(max_size = 6)` Maps to radius of circle, or area

**Stats** - An alternative way to build a layer  
A stat builds new variables to plot (e.g., count, prop).

Visualize a stat by changing the default stat of a geom function, `geom_bar(stat = "count")` or by using a stat function, `stat_count(geom = "bar")`, which calls a default geom to make a layer (equivalent to a geom function). Use `.name_` syntax to map stat variables to aesthetics.

**Scales**

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

(`d <- d + geom_bar(aes(stat = "count"))`)

**Coordinate Systems**

`r <- d + geom_bar()`  
`r + coord_cartesian(xlim = c(0, 5))`  
xmin, xmax

The default cartesian coordinate system

`r + coord_fixed(ratio = 1/2)`  
ratio, xlim, ylim

Cartesian coordinates with fixed aspect ratio between x and y units

`r + coord_flip()`  
xmin, xmax

Flipped-Cartesian coordinates

`r + coord_polar(theta = "x", direction = 1)`  
theta, start, direction

Polar coordinates

`r + coord_trans(xtrans = "sqrt")`  
xtrans, ytrans, xlim, ylim

Transformed cartesian coordinates. Set xtrans and ytrans to the name of a transfun function.

**Faceting**

Facets divide a plot into subplots based on the values of one or more discrete variables.

`t <- ggplot(mpg, aes(cyl, hwy))`  
`t + facet_grid(~ cyl, ~ fl)`

Facet into columns based on fl

`t + facet_grid(~ year)`

Facet into rows based on year

`t + facet_grid(~ fl)`

Facet into both rows and columns

`t + facet_wrap(~ fl)`

Wrap facets into a rectangular layout

**scales** to let axes limits vary across facets

`t + facet_grid(~ fl, scales = "free")`

x and y axis limits adjust to individual facets

- `"free_x"` - x axis limits adjust
- `"free_y"` - y axis limits adjust

**Set labeler** to adjust facet labels

`t + facet_grid(~ fl, labeler = label_both)`

`t + facet_grid(~ fl, labeler = label_bquote(alpha ~ alpha))`

`t + facet_grid(~ fl, labeler = label_parsed(alpha ~ alpha))`

**Labels**

`t + labs(x = "New x axis label", y = "New y axis label", title = "Add a title above the plot", subtitle = "Add a subtitle below title", caption = "Add a caption below plot", tag = "New new legend title")`

**Annotations**

`t + annotate(guide = "text", x = 8, y = 9, label = "A")`

`geom_text` - manual values for generic aesthetics

**Position Adjustments**

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

`s + ggplot(mpg, aes(cyl, hwy + 10))`

`s + geom_bar(position = "dodge")`  
Arrange elements side by side

`s + geom_bar(position = "fill")`  
Stack elements on top of one another, maximal height

`s + geom_point(position = "jitter")`  
Add random noise to X and Y position of each element to avoid overlapping

`s + geom_label(position = "nudge")`  
Nudge labels away from points

`s + geom_bar(position = "stack")`  
Stack elements on top of one another

Each position adjustment can be recast as a function with manual `width` and `height` arguments

`s + geom_bar(position = position_dodge(width = 10))`

**Legends**

`s + theme(legend.position = "bottom")`  
Place legend at "bottom", "top", "left", or "right"

`s + guides(fill = "none")`  
Set legend type for each aesthetic: colorbar, legend, or none (no legend)

`s + scale_fill_discrete(name = "title", labels = c("A", "B", "C", "D", "E"))`  
Set legend title and labels with a scale function.

**Themes**

`s + theme_bw()`  
white background with grid lines

`s + theme_shape()`  
code, size

`s + scale_size_manual(values = c(3, 7))`

`s + theme_rect()`  
maps to radius of circle, or area

`s + theme_radios()`  
maps to radius of circle, or area

`s + theme_size_area(max_size = 4)`

**Zooming**

**Without clipping** (preferred)

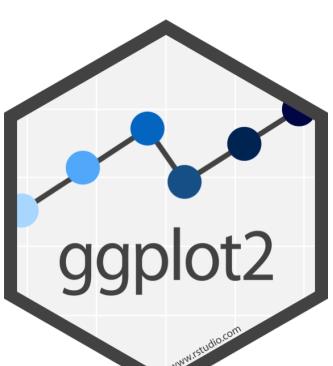
`t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))`

**With clipping** (removes unseen data points)

`t + xlim(0, 100) + ylim(10, 20)`

`s + scale_x_continuous(limits = c(0, 100)) + scale_y_continuous(limits = c(0, 100))`

Scales control the appearance of data elements.

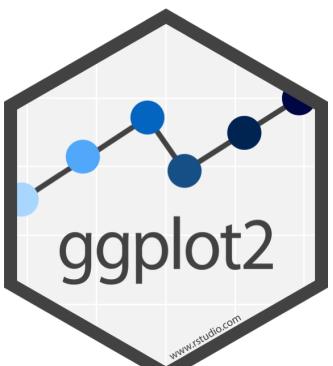


## **Aesthetic mapping**

What variable to map to color, shape, etc.

## **Scale functions**

How to map the variable to color, shape, etc.  
(which values are paired with which levels)



# naming conventions

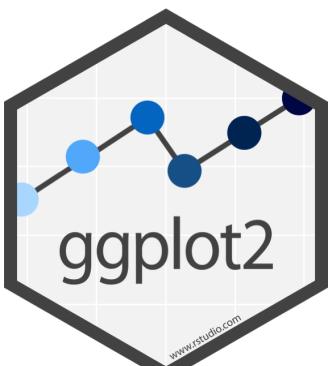
You can add a scale function for each aesthetic.

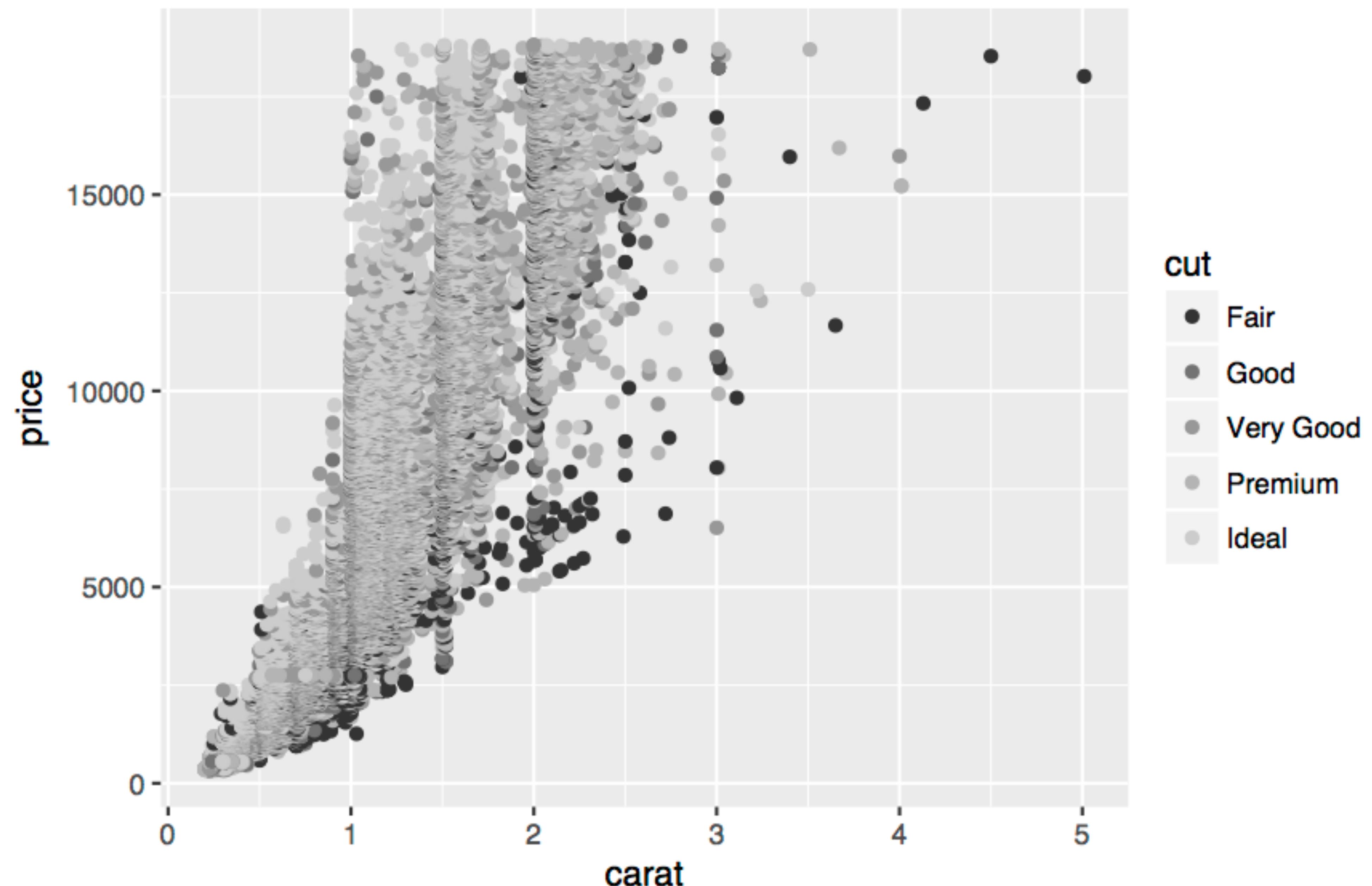
```
s + scale_color_grey()
```

scale\_

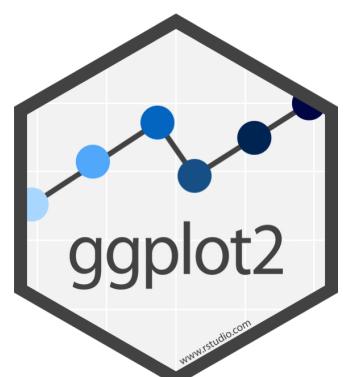
aesthetic name

specific scale name  
(see cheatsheet or  
[docs.ggplot2.org](http://docs.ggplot2.org))





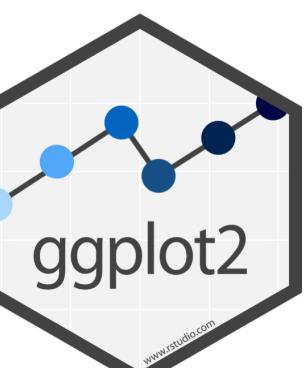
`s + scale_color_grey()`

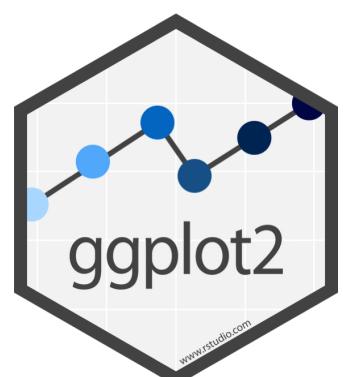
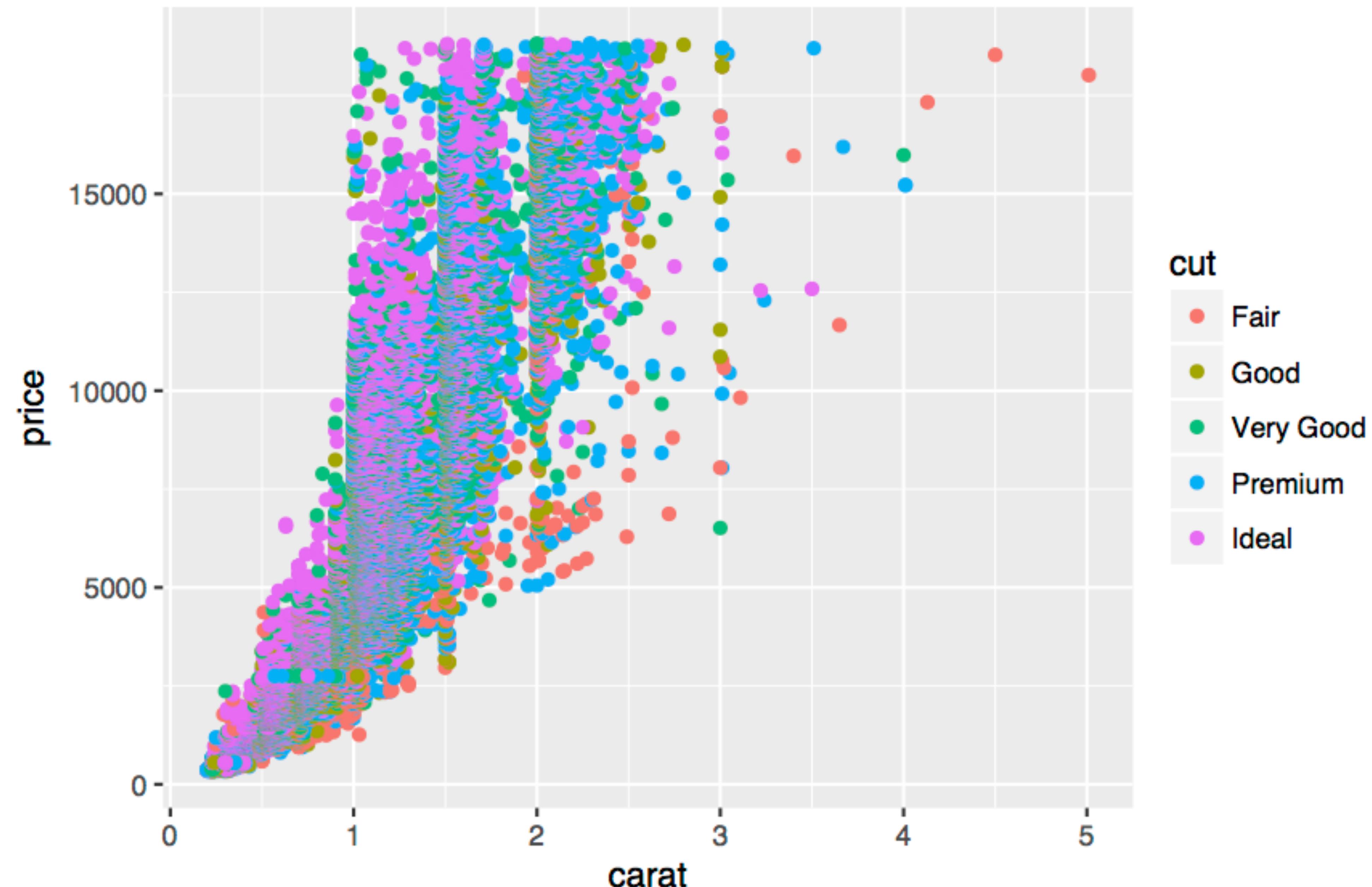


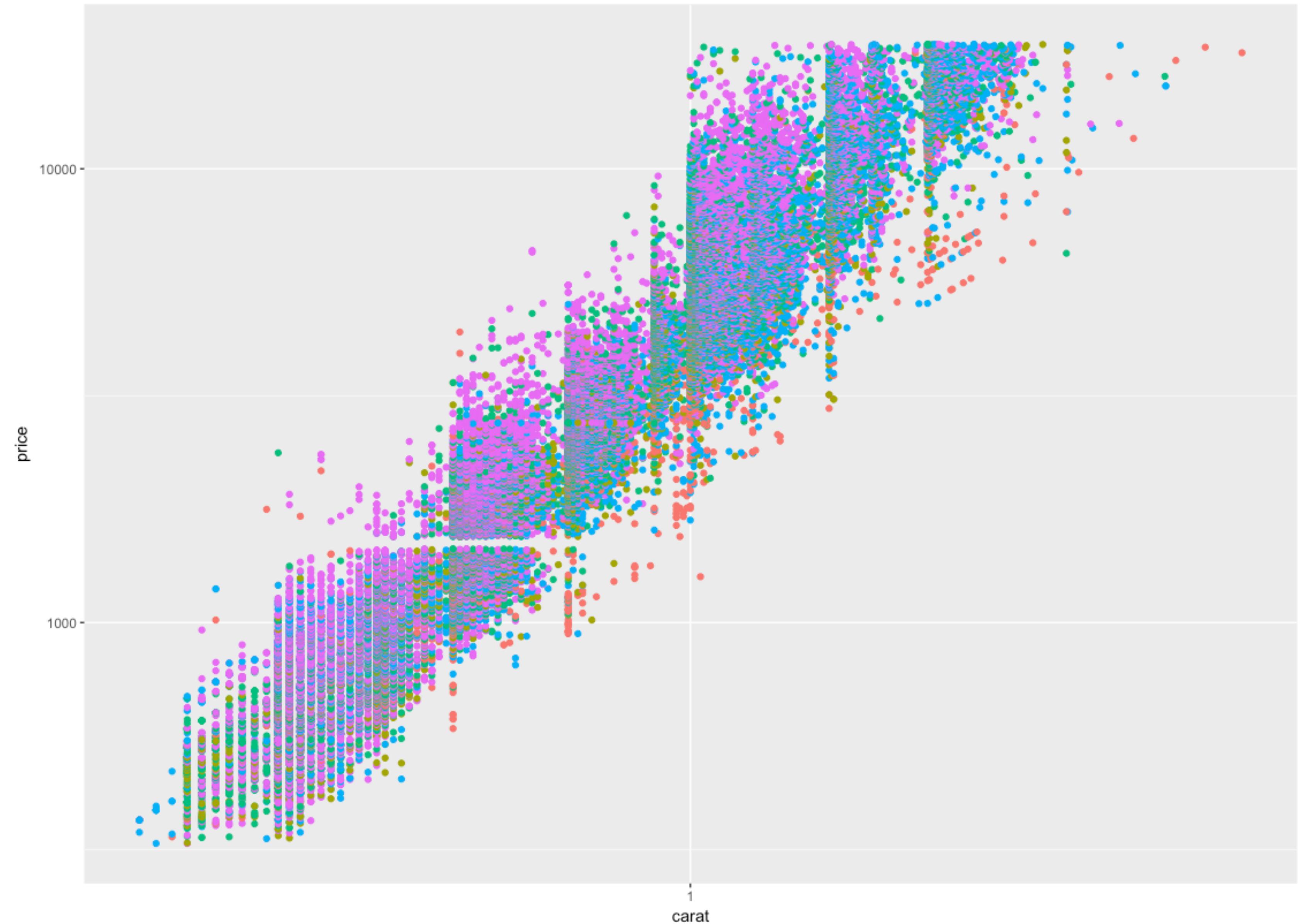
# common scales

The two most common uses for scales are to

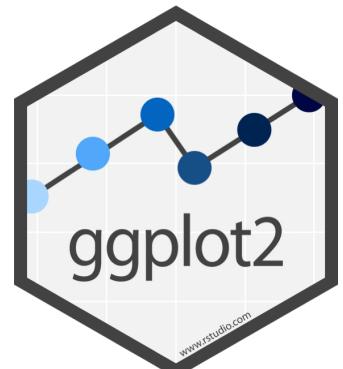
1. **change the axis scales**
2. **change color themes**

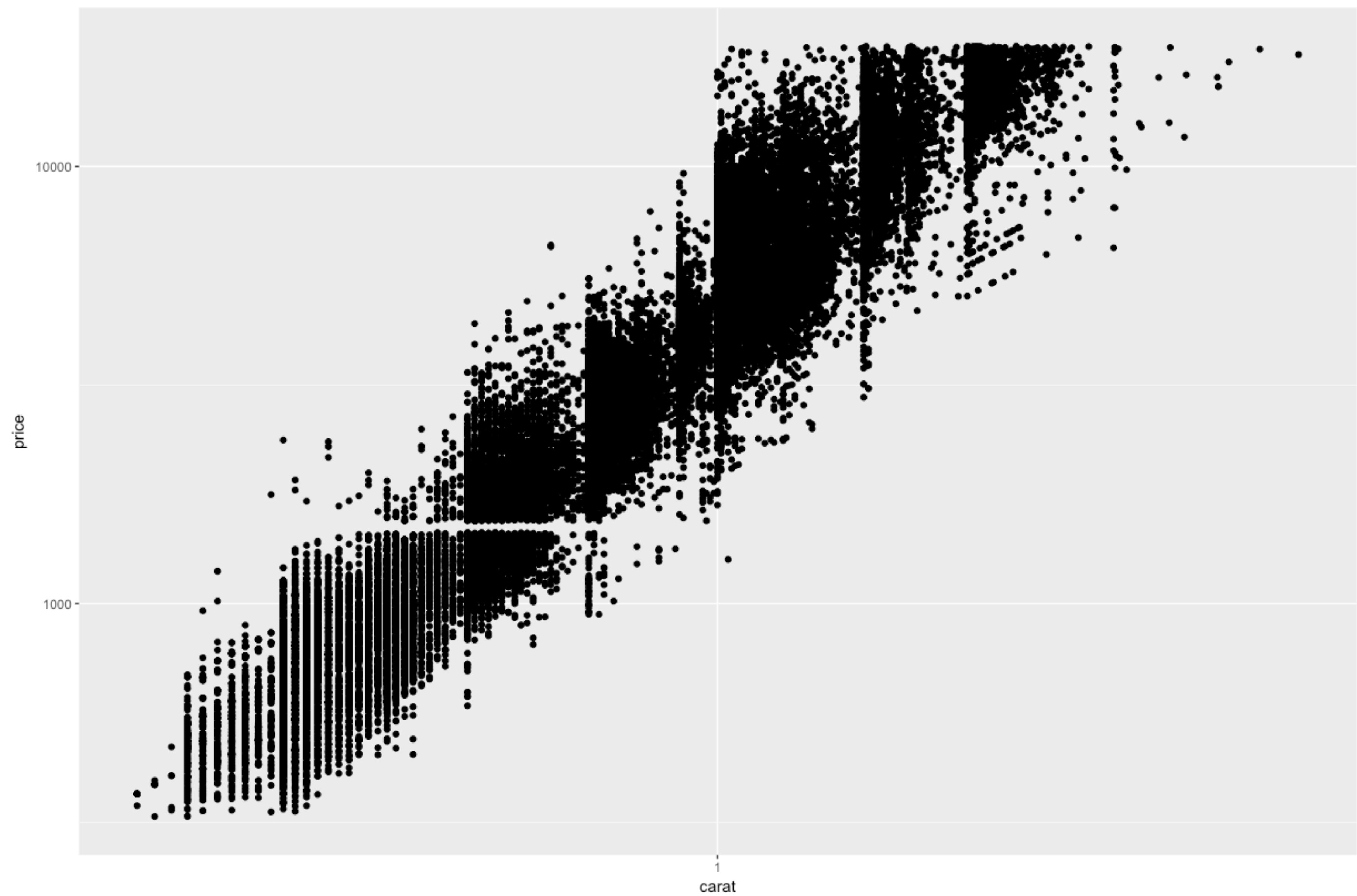




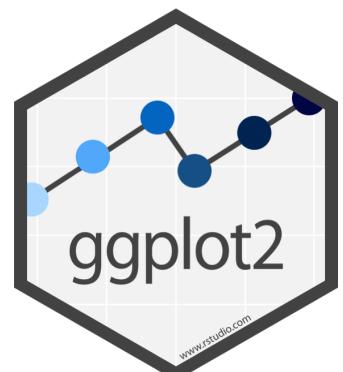


`s + scale_x_log10() + scale_y_log10()`

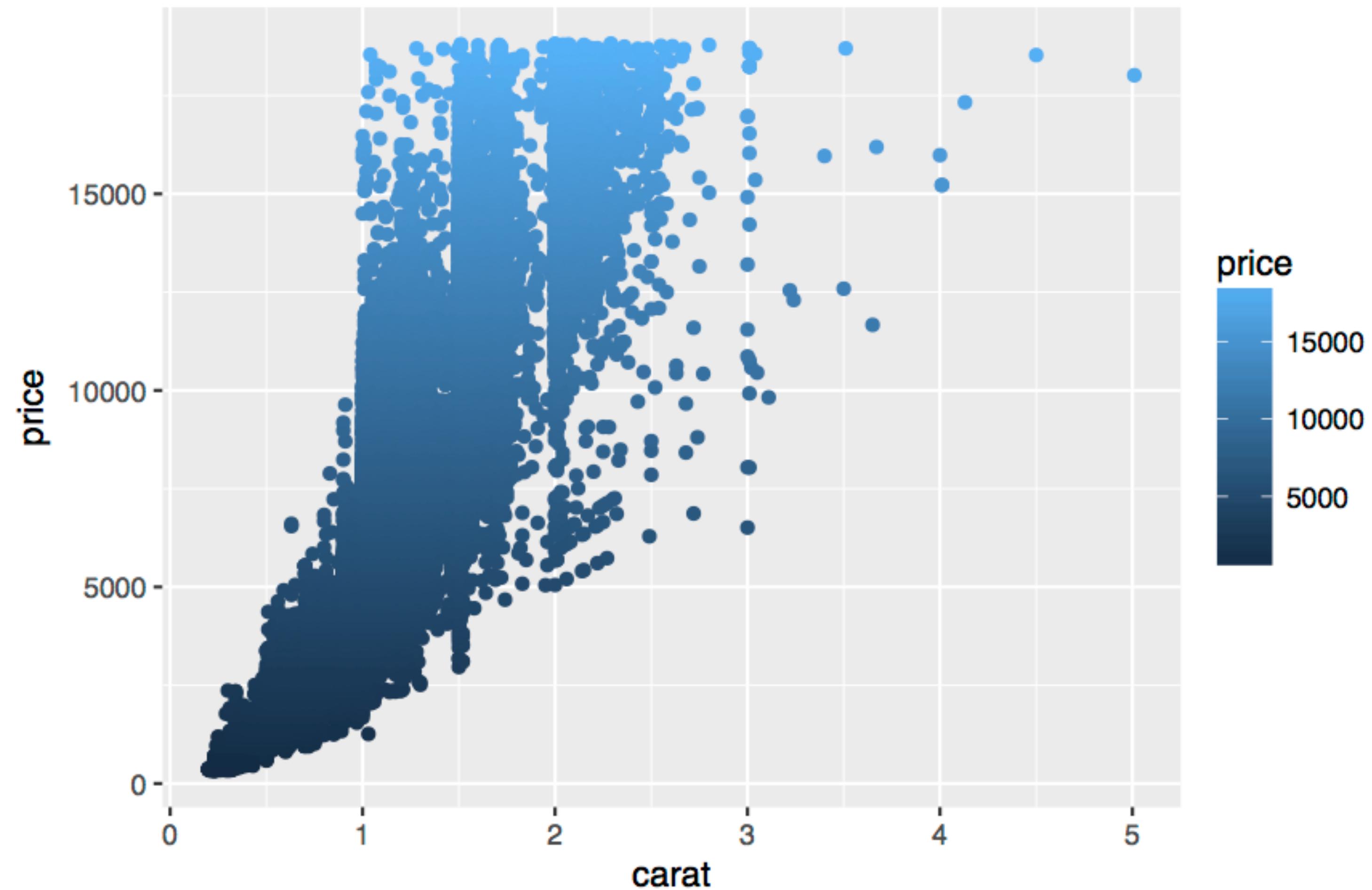




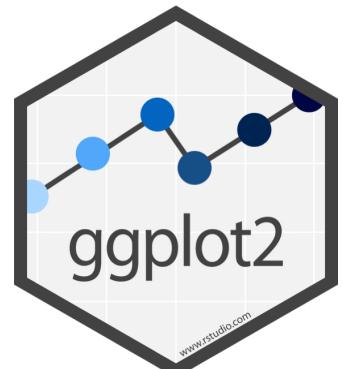
`s + scale_x_log10() + scale_y_log10()`



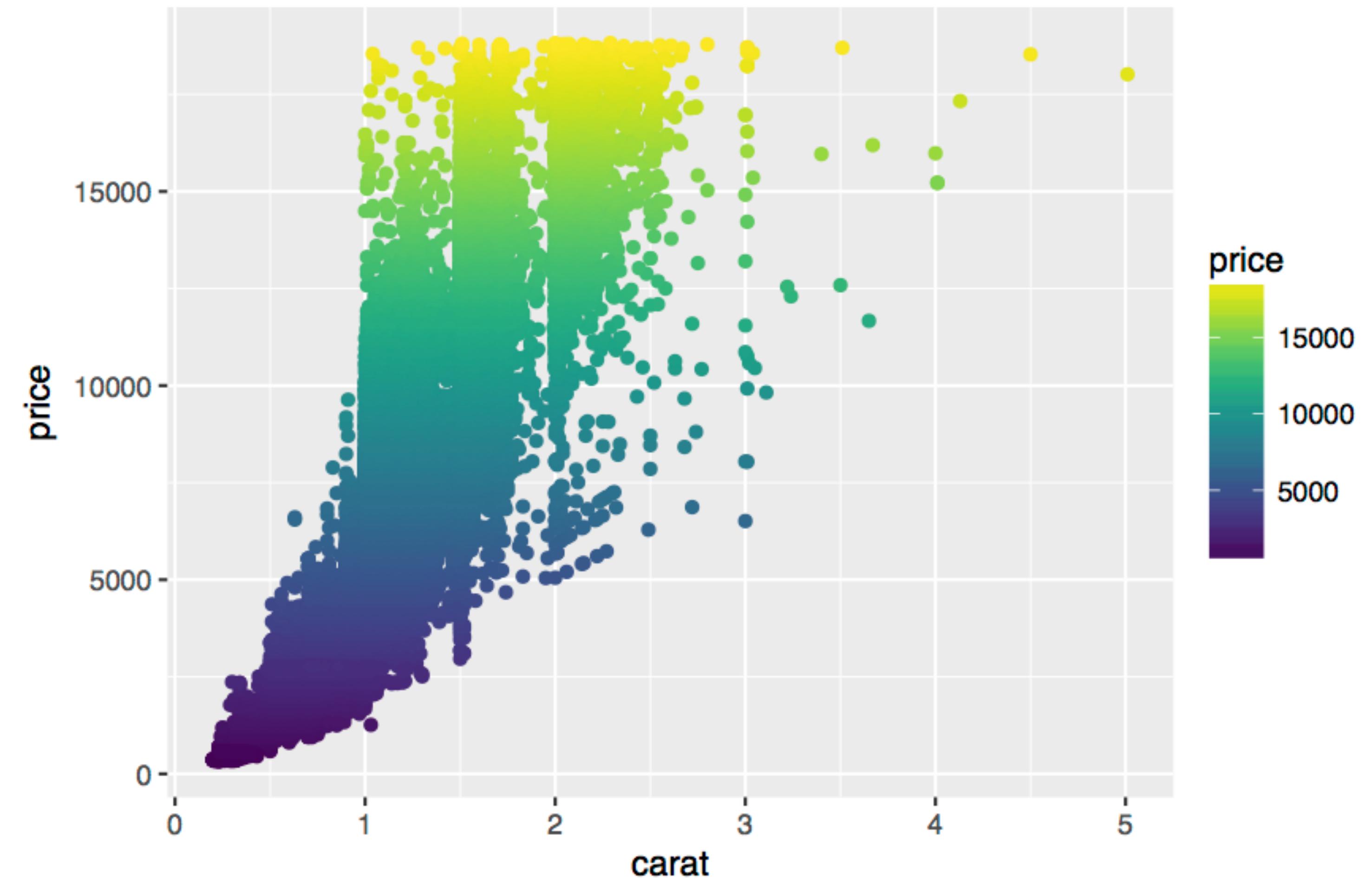
# Continuous colors



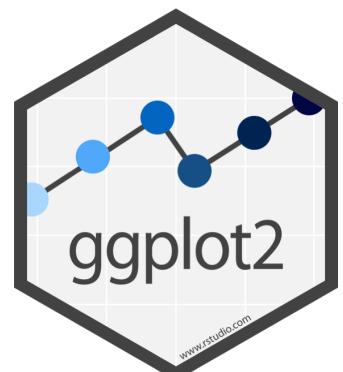
```
s1 <- ggplot(diamonds, aes(carat, price)) +  
  geom_point(aes(color = price))
```



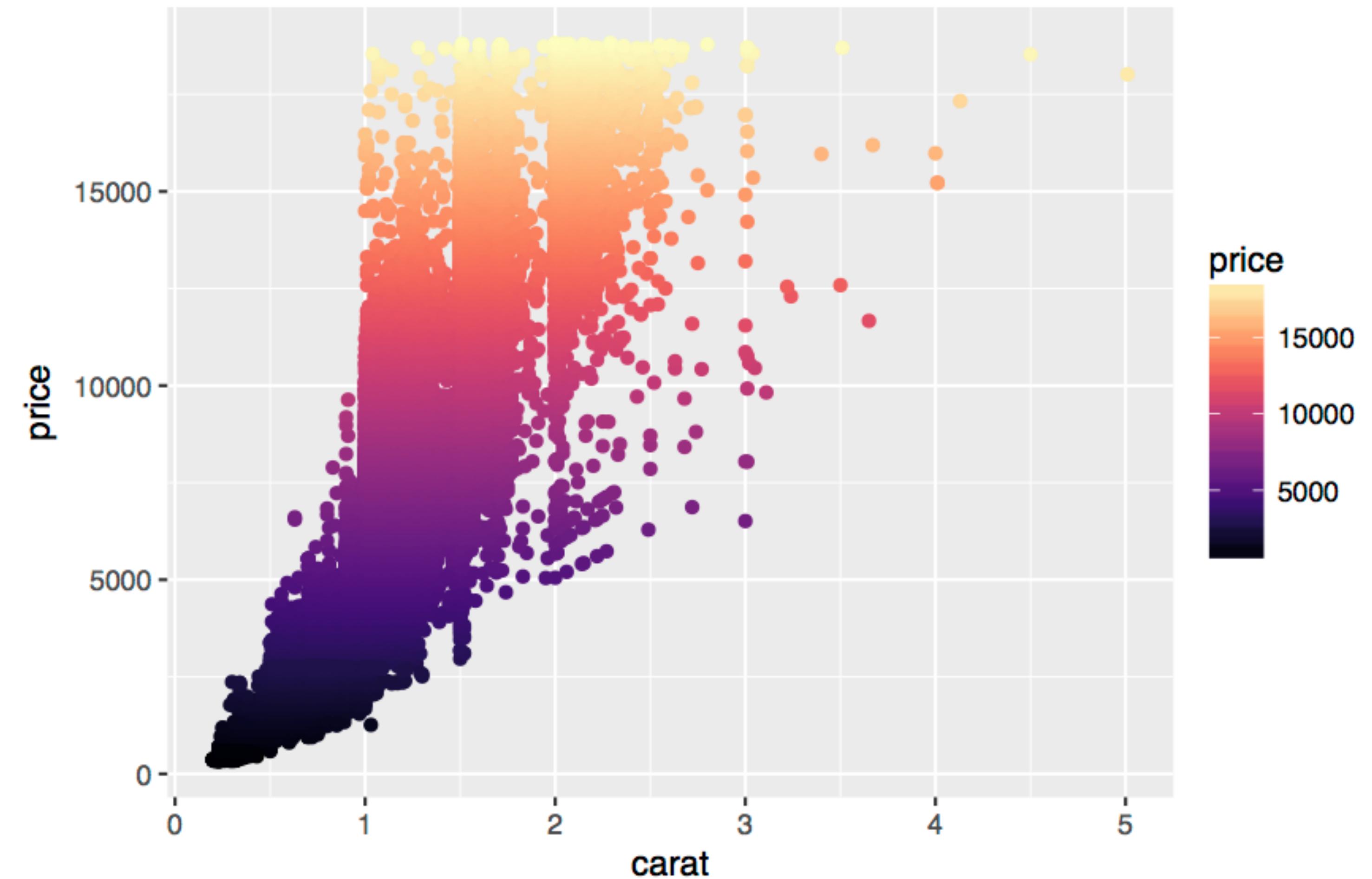
# Viridis



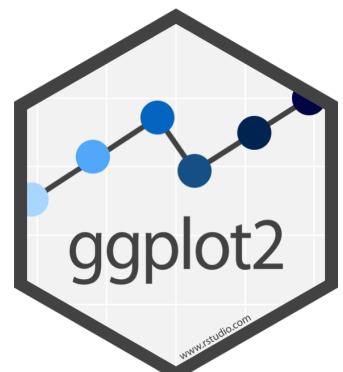
```
library(viridis)  
s1 + scale_color_viridis()
```



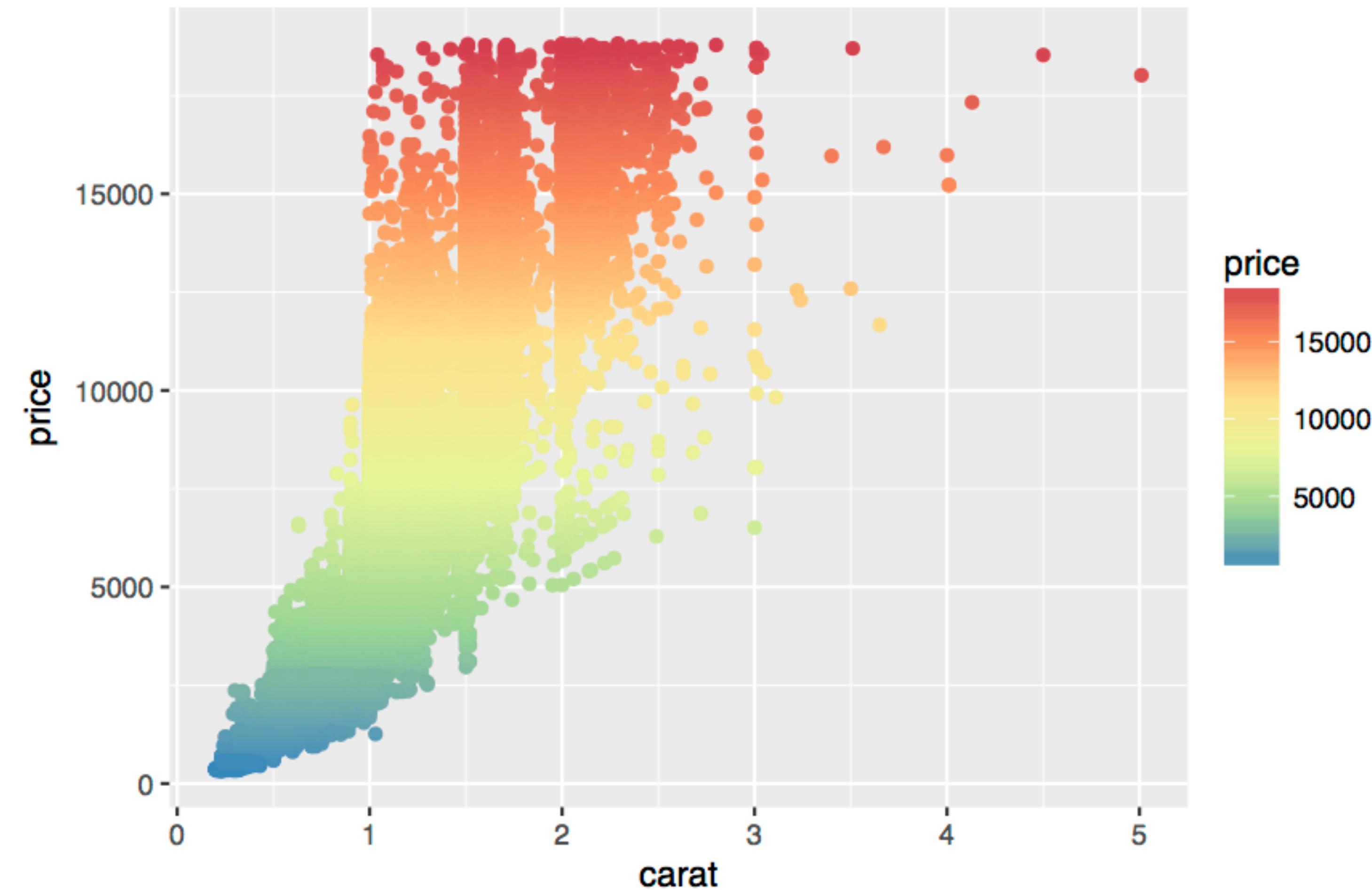
# Viridis



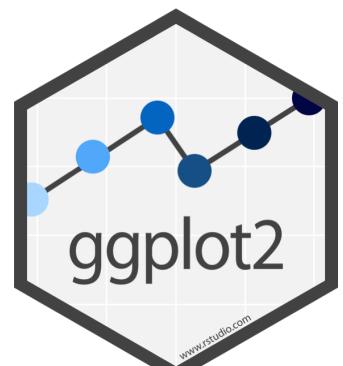
```
library(viridis)
s1 + scale_color_viridis(option = "A")
```



# distiller



```
s1 + scale_color_distiller(palette = "Spectral")
```



# RColorBrewer

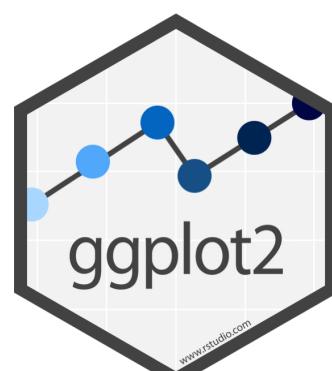


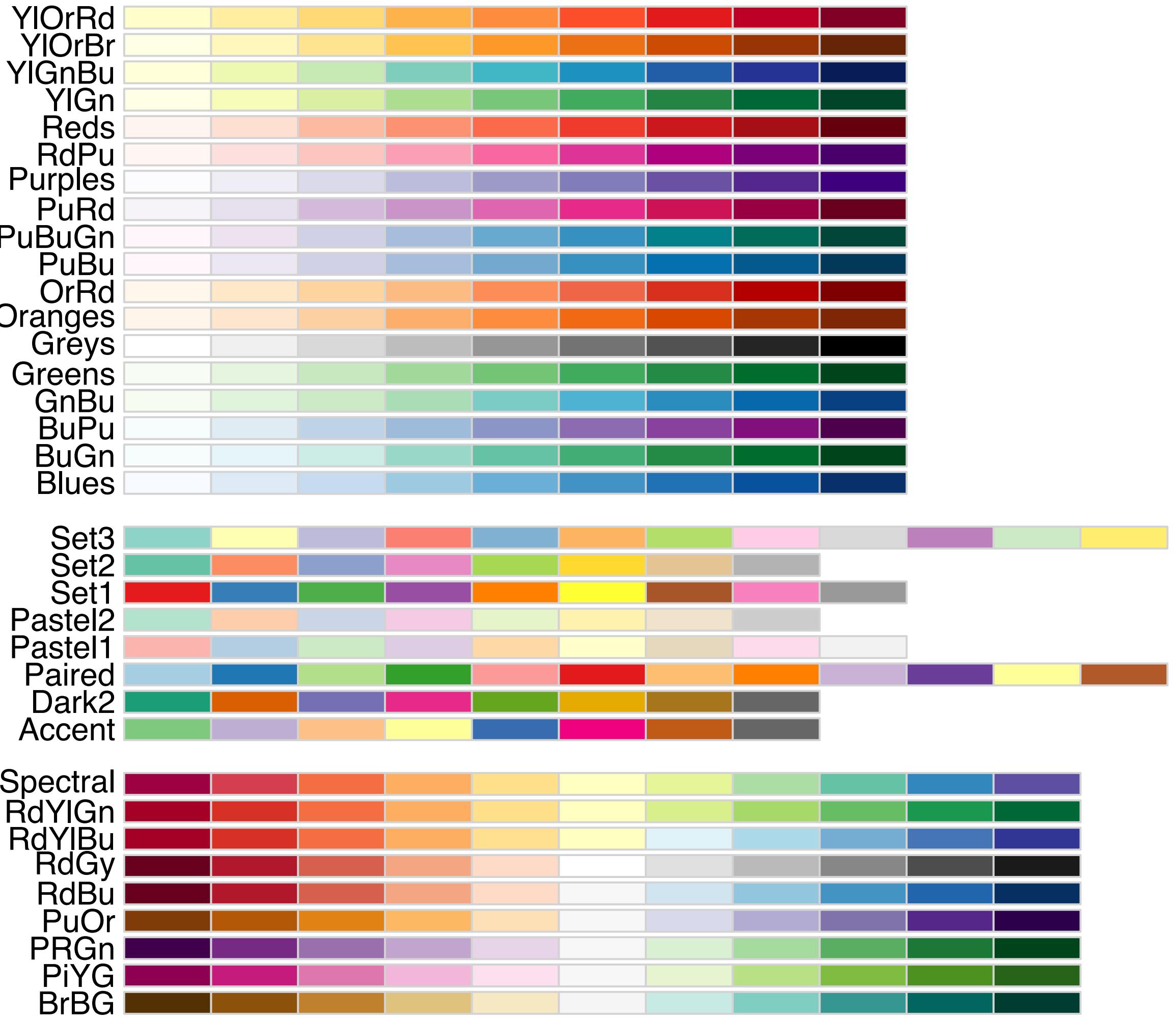
A package of color palettes

```
# install.packages("RColorBrewer")  
library(RColorBrewer)
```

scale\_color\_distiller()\* → continuous variables  
scale\_color\_brewer()\* → discrete variables

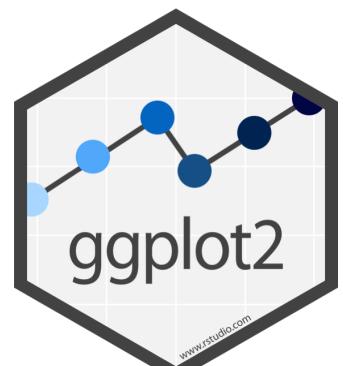
\* Also scale\_fill\_distiller() and scale\_fill\_brewer()



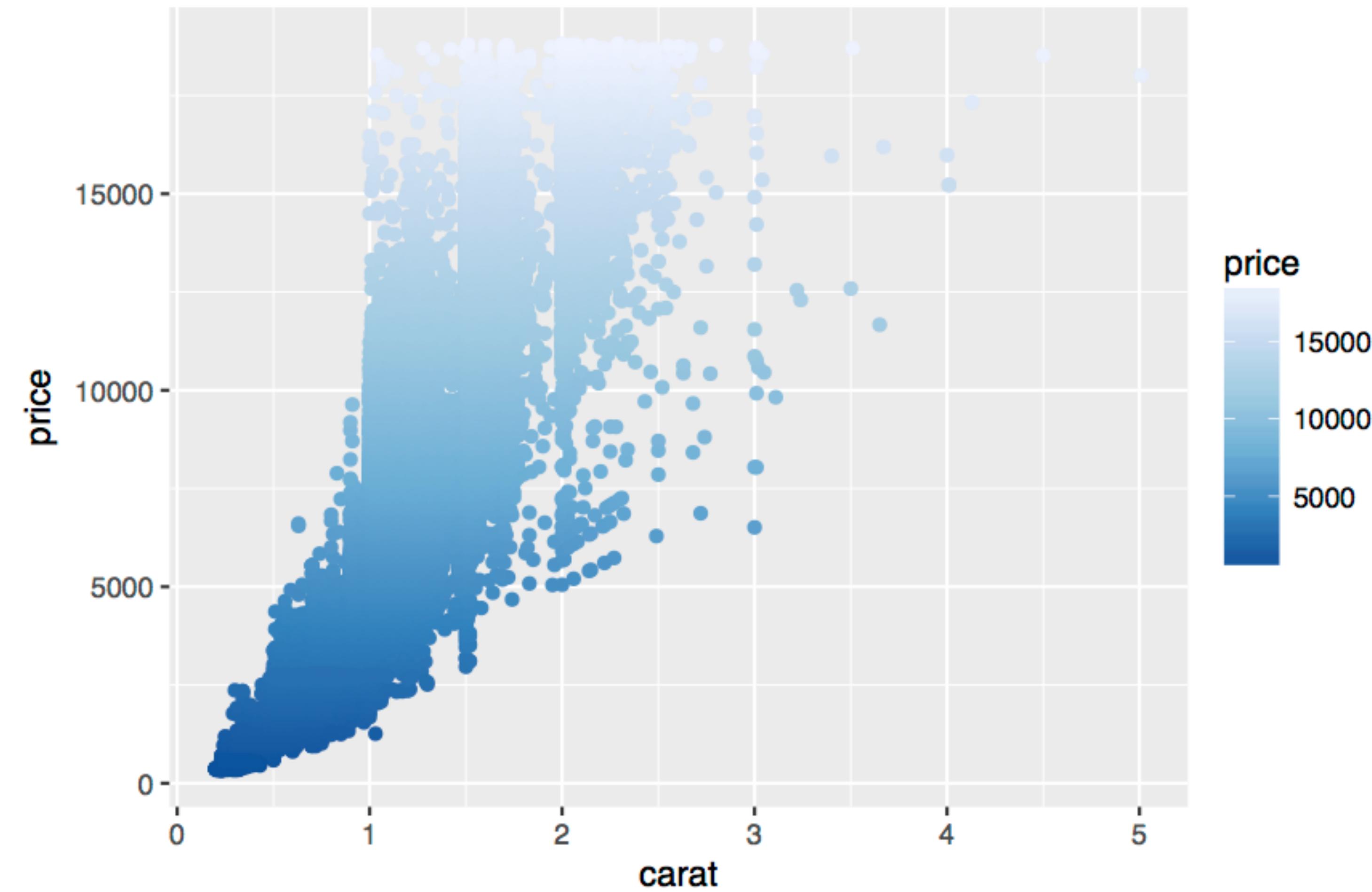


To see available palettes

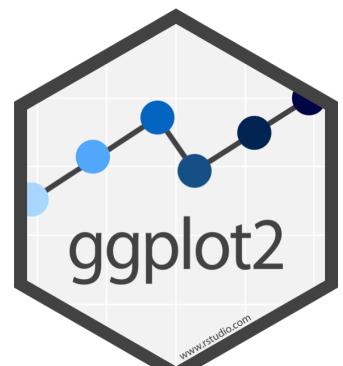
```
RColorBrewer::display.brewer.all()
```



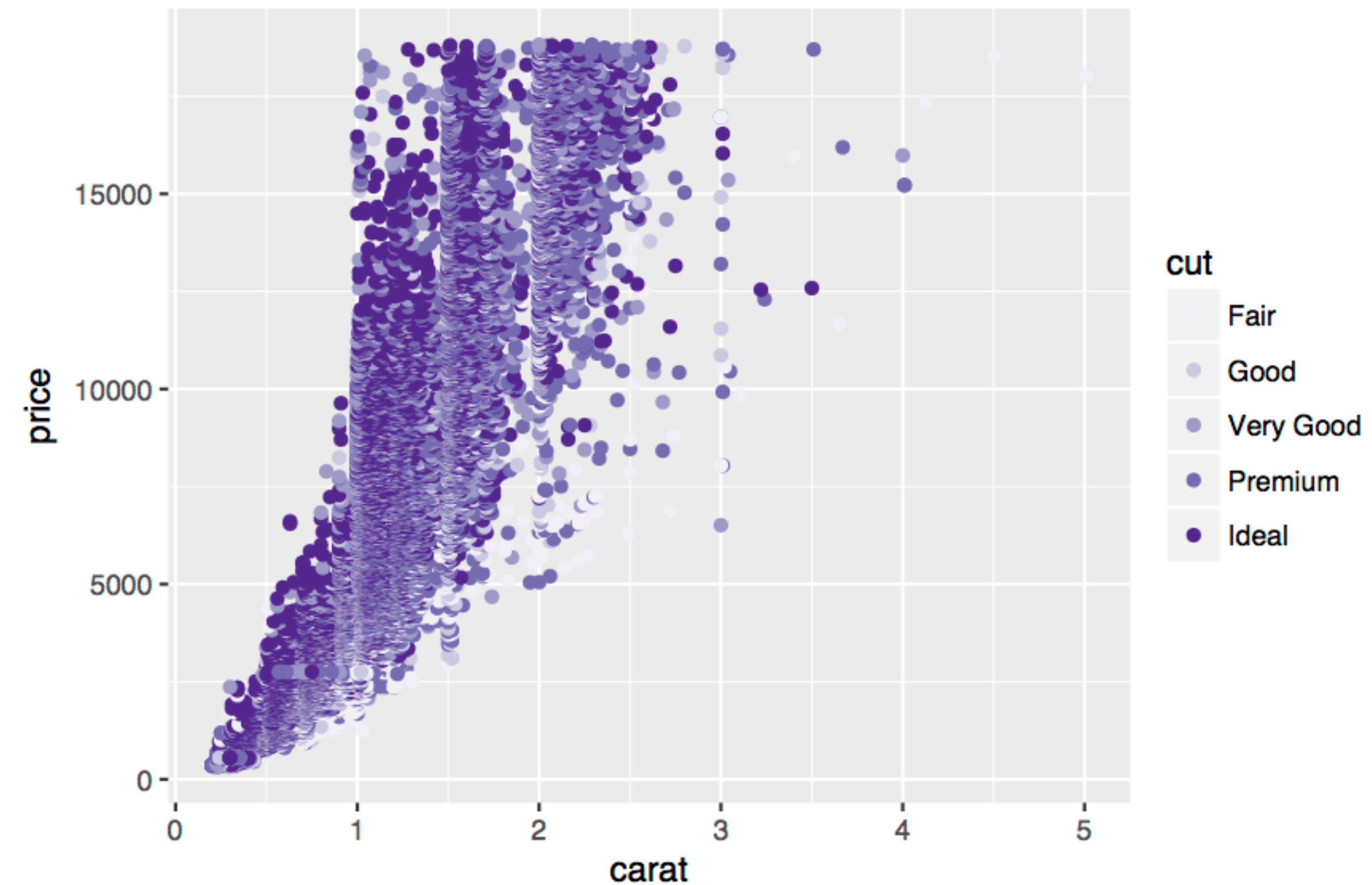
# distiller



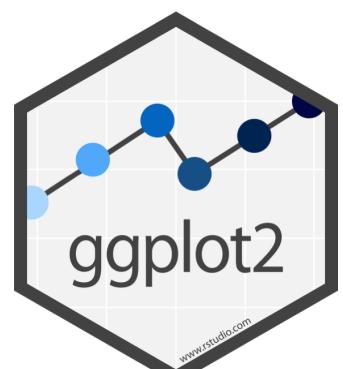
```
s1 + scale_color_distiller(palette = "Blues")
```



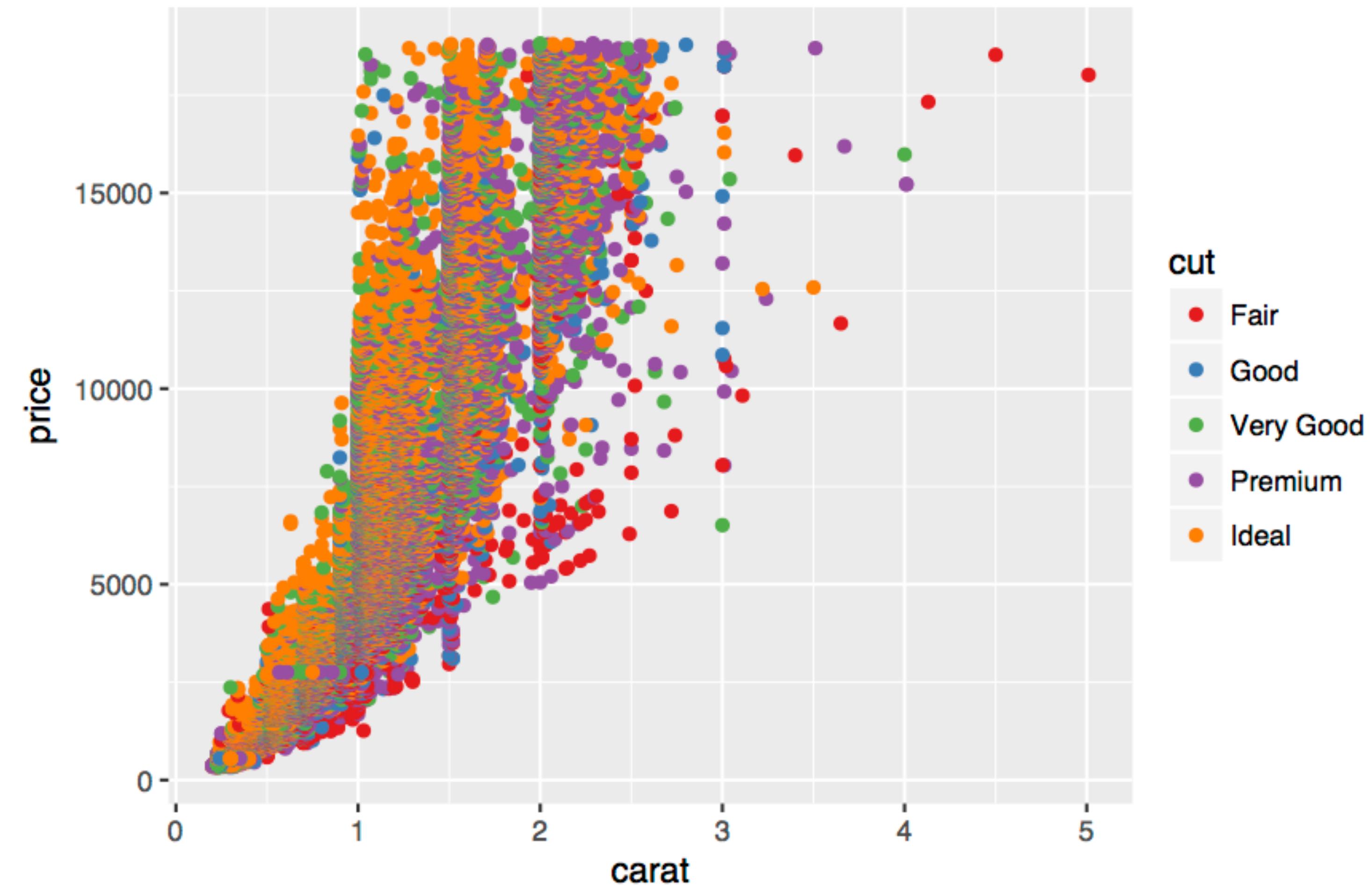
# brewer



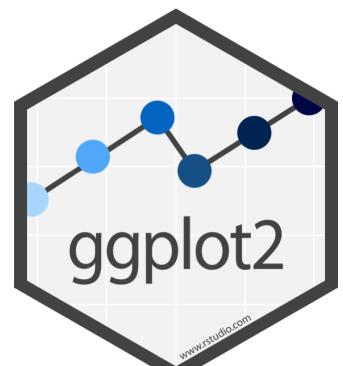
```
s1 + scale_color_brewer(palette = "Purples")
```



# brewer

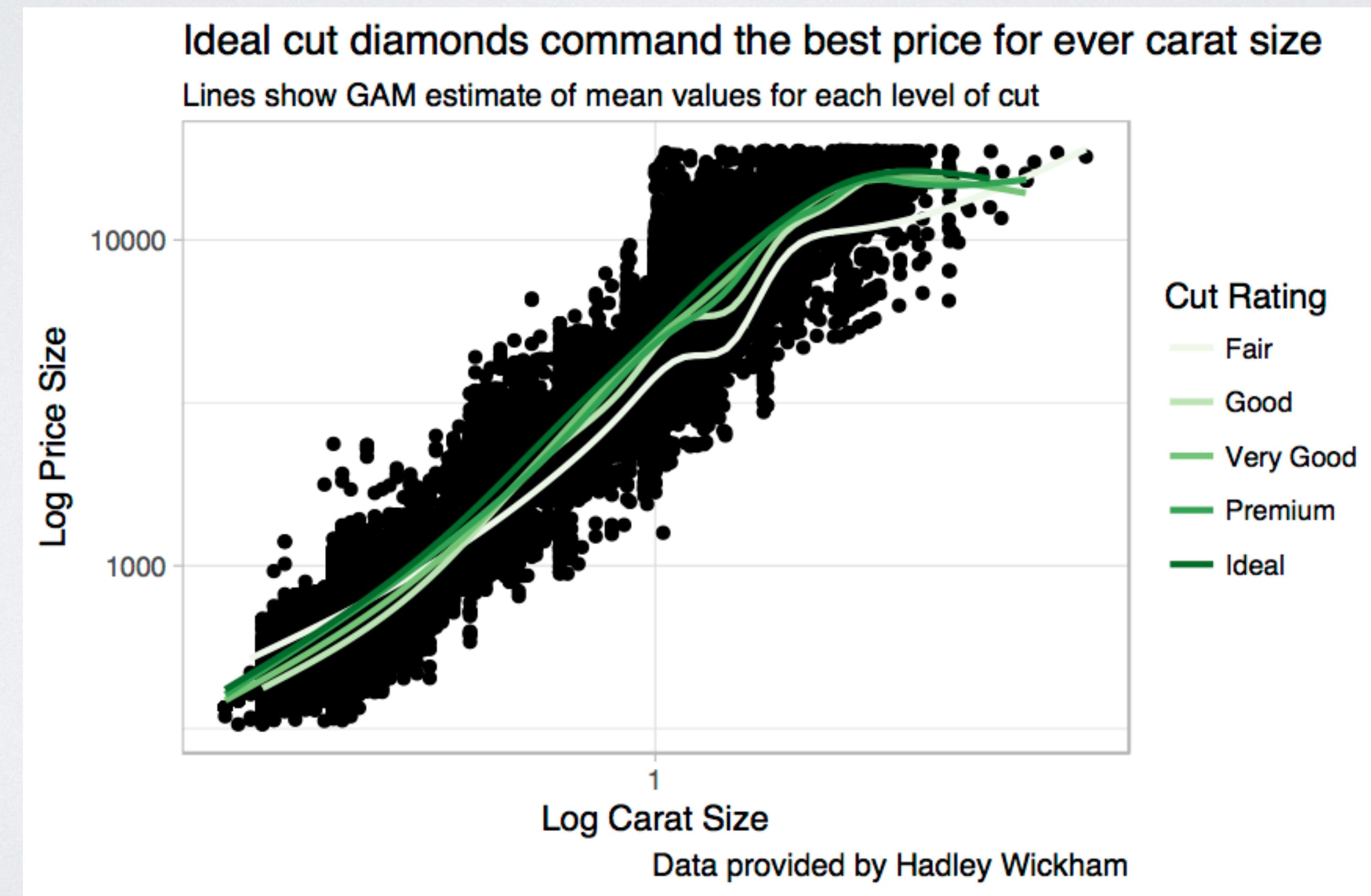


```
s1 + scale_color_brewer(palette = "Set1")
```



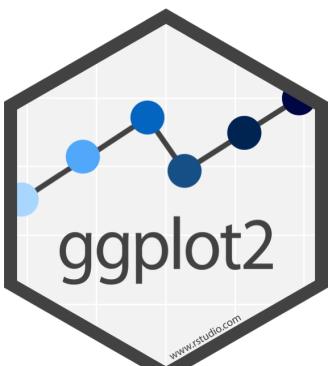
# Your turn

Experiment with labels, themes, and scales to make a more clear graph, perhaps this one.



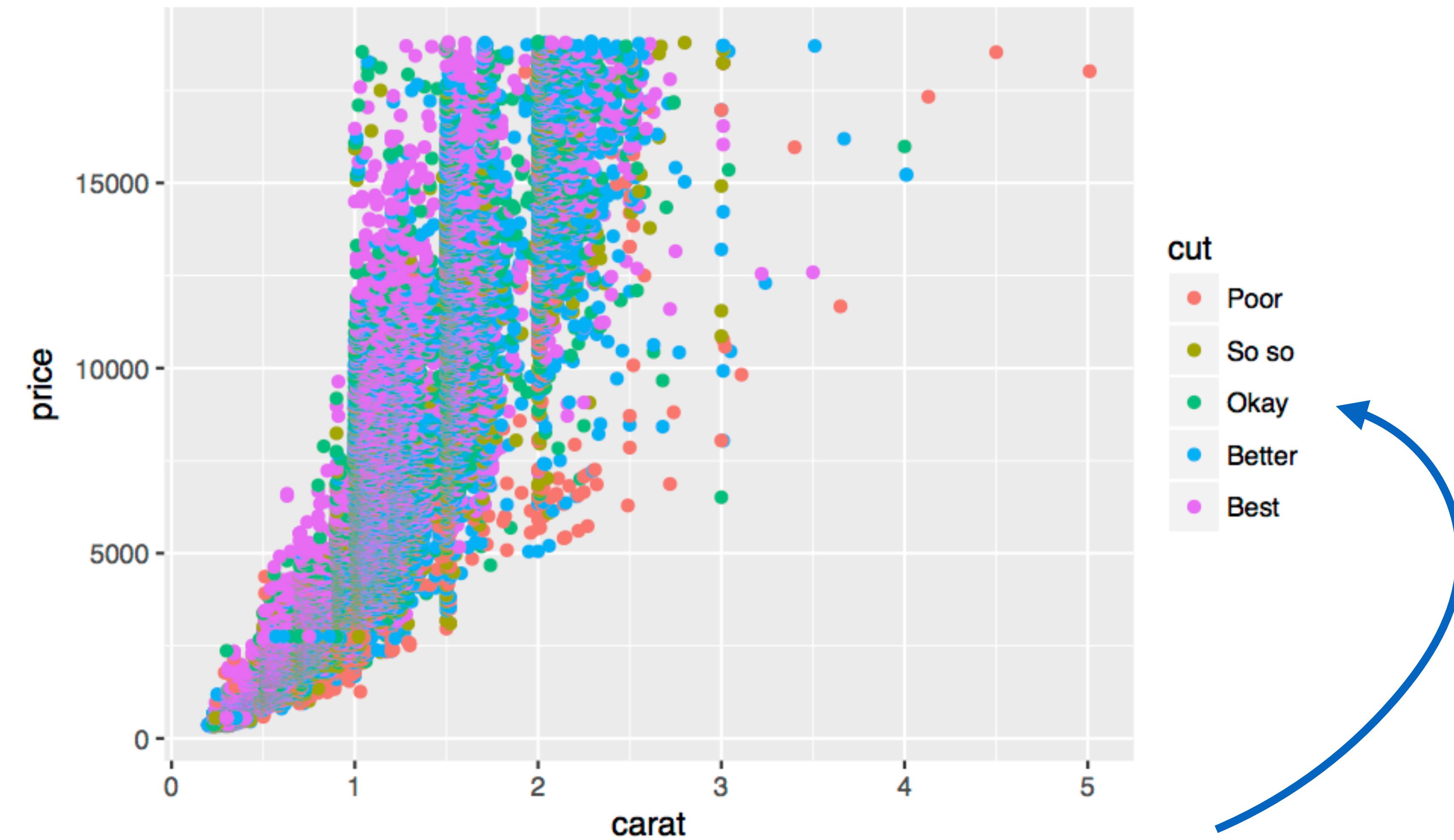
10 : 00

```
ggplot(diamonds, aes(x = carat, y = price)) +  
  geom_point() +  
  geom_smooth(aes(color = cut), se = FALSE) +  
  labs(title = "Ideal cut diamonds command the best price for ever carat size",  
       subtitle = "Lines show GAM estimate of mean values for each level of cut",  
       caption = "Data provided by Hadley Wickham",  
       x = "Log Carat Size",  
       y = "Log Price Size",  
       color = "Cut Rating") +  
  scale_x_log10() +  
  scale_y_log10() +  
  scale_color_brewer(palette = "Greens") +  
  theme_light()
```



# legend labels

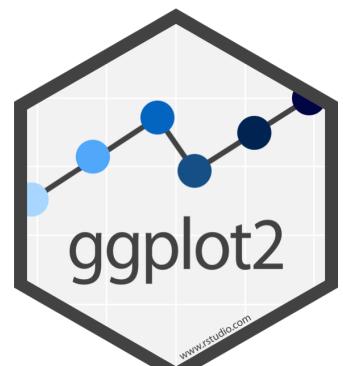
The labels argument of a scale controls the legend/axis values



```
s + scale_color_discrete(labels = c("Poor" , "So so", "Okay", "Better", "Best"))
```

# Default scales

aesthetic	variable	default
x	continuous	scale_x_continuous()
	discrete	scale_x_discrete()
y	continuous	scale_y_continuous()
	discrete	scale_y_discrete()
color	continuous	scale_color_continuous()
	discrete	scale_color_discrete()
fill	continuous	scale_fill_continuous()
	discrete	scale_fill_discrete()
size	continuous	scale_size()
shape	discrete	scale_shape()

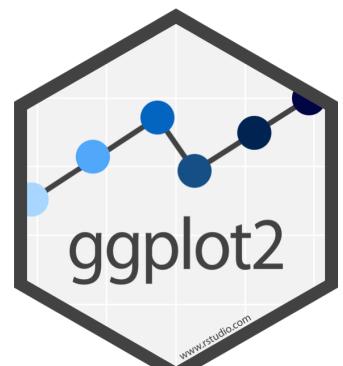
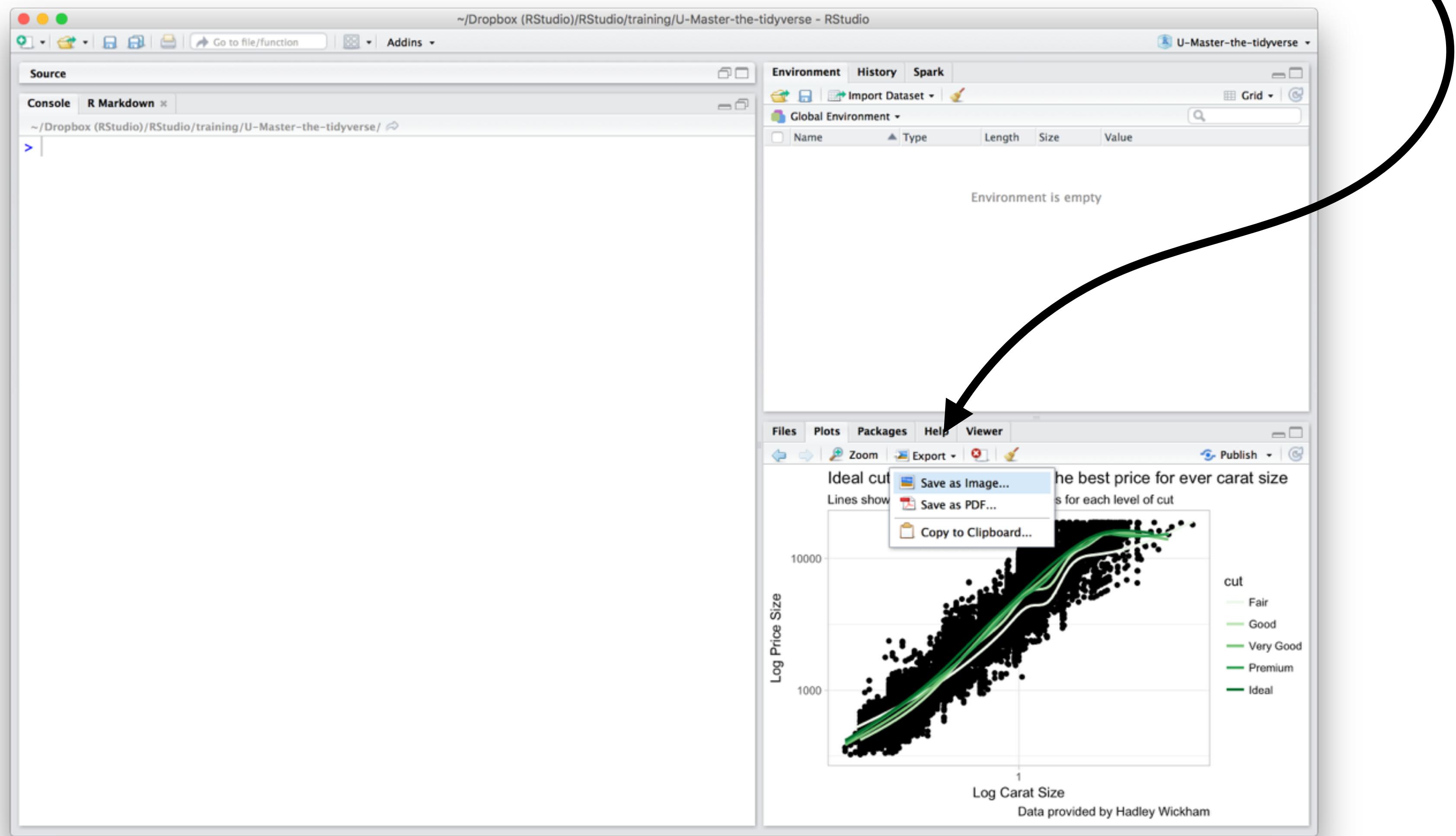


# Saving graphs



# Manually saving plots

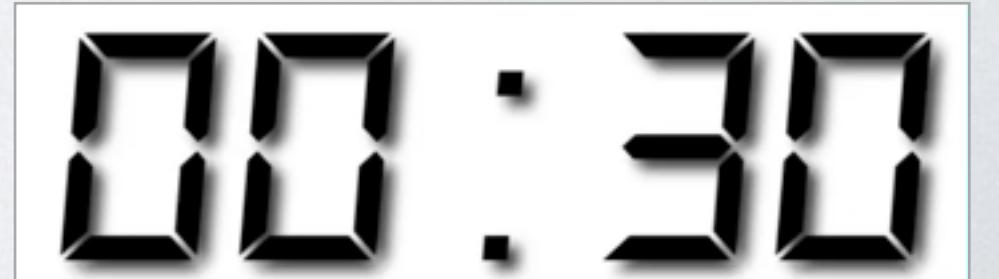
Save plots manually with the export menu



# Your turn

What does this command return?

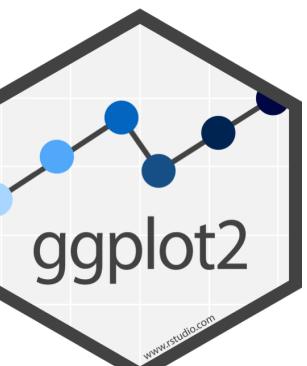
get\_wd()



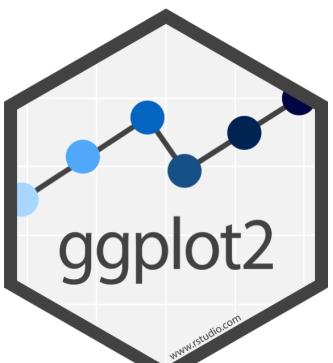
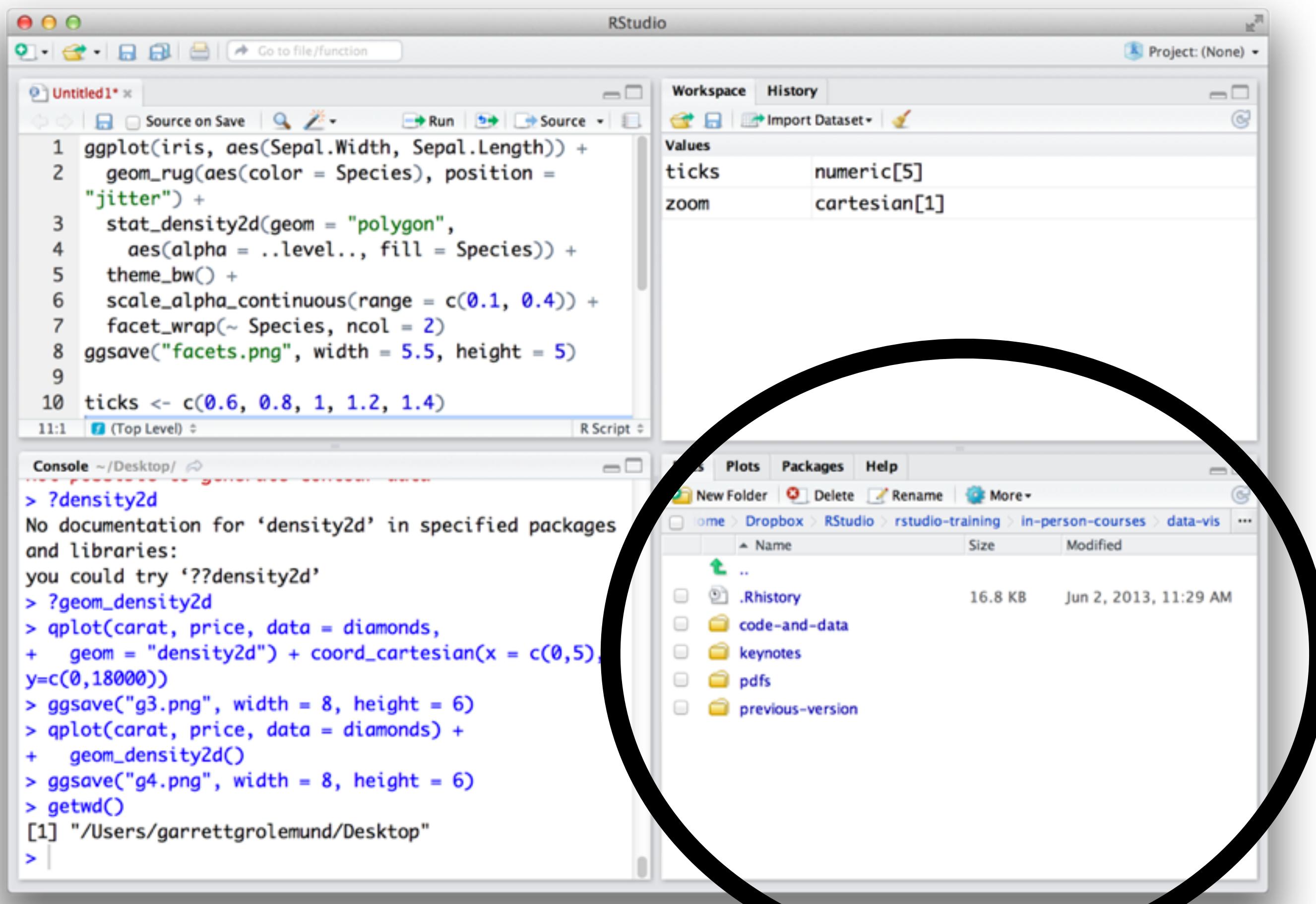
# Working directory

R associates itself with a folder (i.e. directory) on your computer.

- This folder is known as your "working directory"
- When you save files, R will save them here
- When you load files, R will look for them here

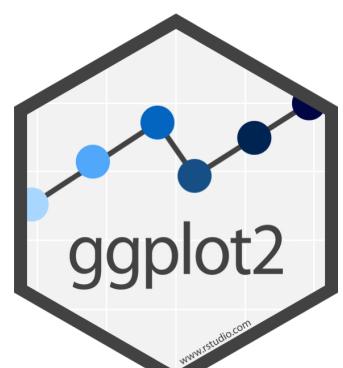
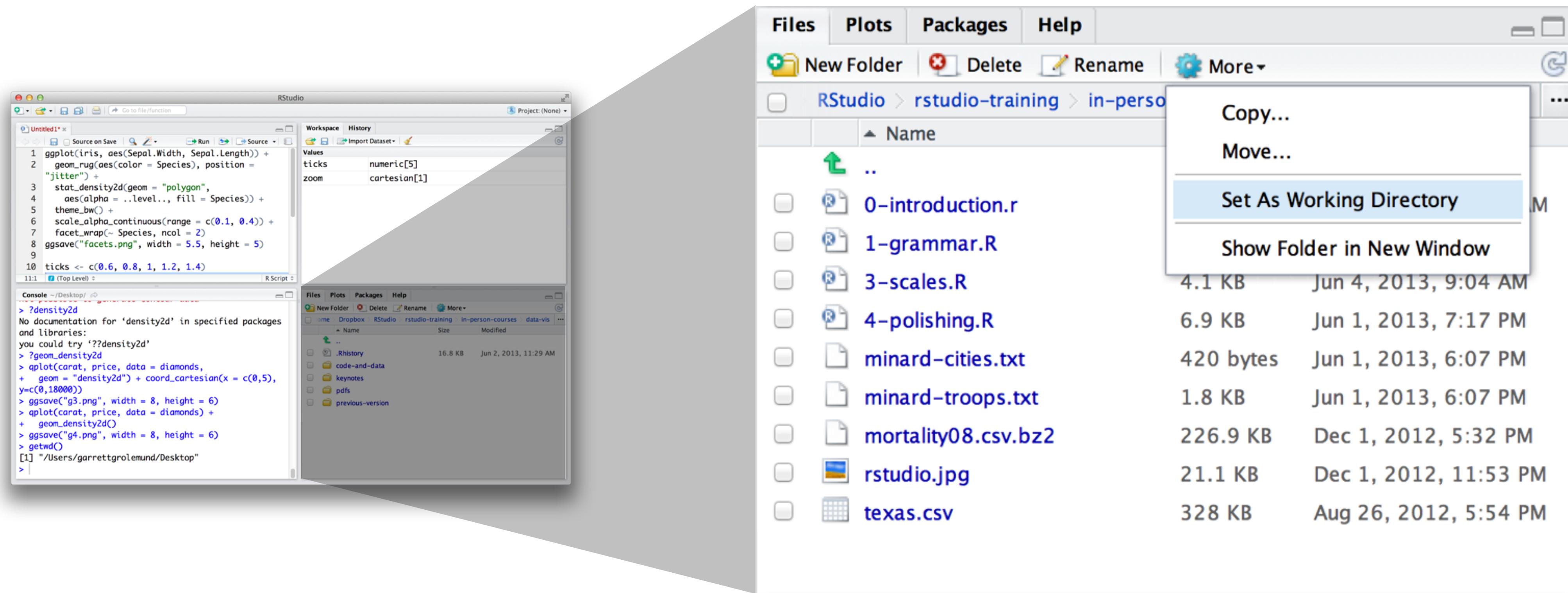


# The files pane of the IDE displays the contents of your working directory



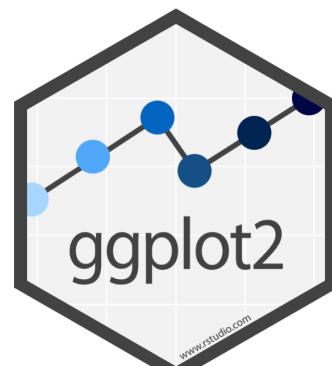
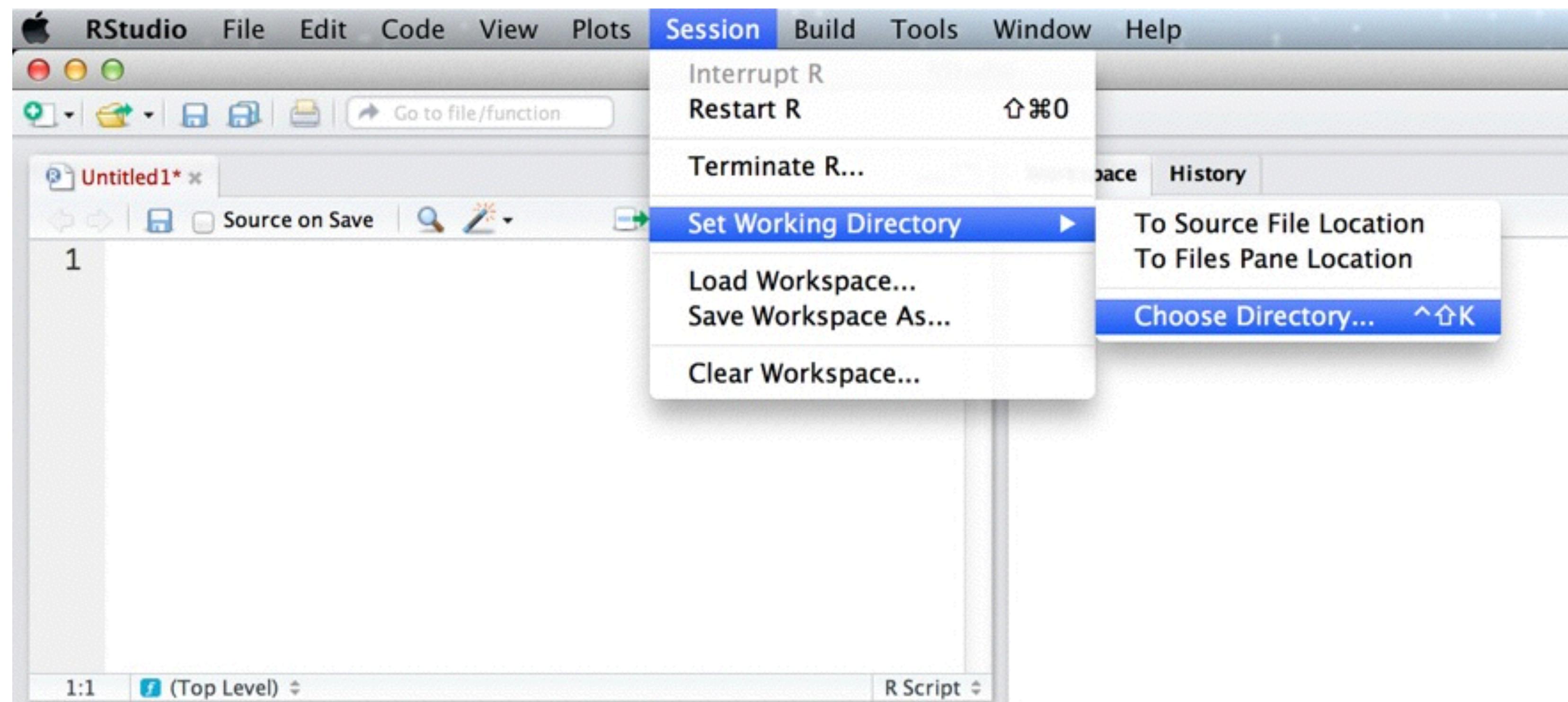
# Changing the Working directory

First option: Navigate in the files pane to a new directory.  
Click More>Set As Working Directory



# Changing the Working directory

**Second option:** In the toolbar, go to  
**Session>Set Working Directory>Choose Directory...**



# Saving plots

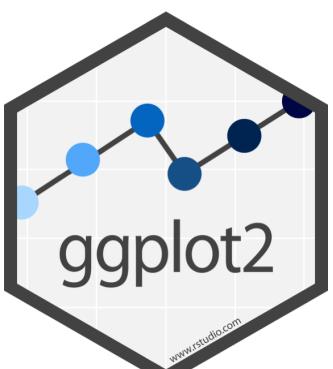
`ggsave()` saves the last plot.

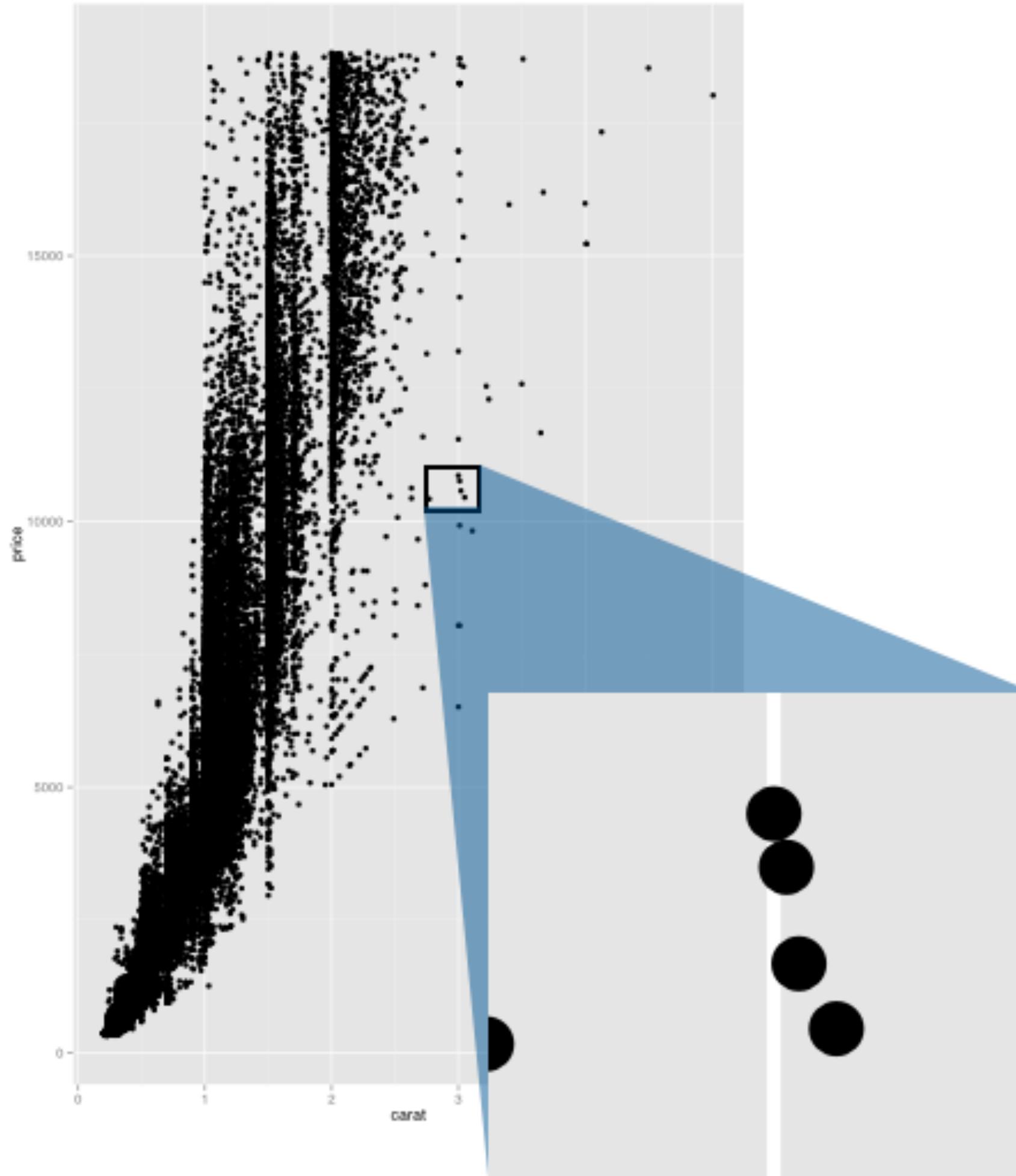
Uses size on screen:

```
ggsave("my-plot.pdf")  
ggsave("my-plot.png")
```

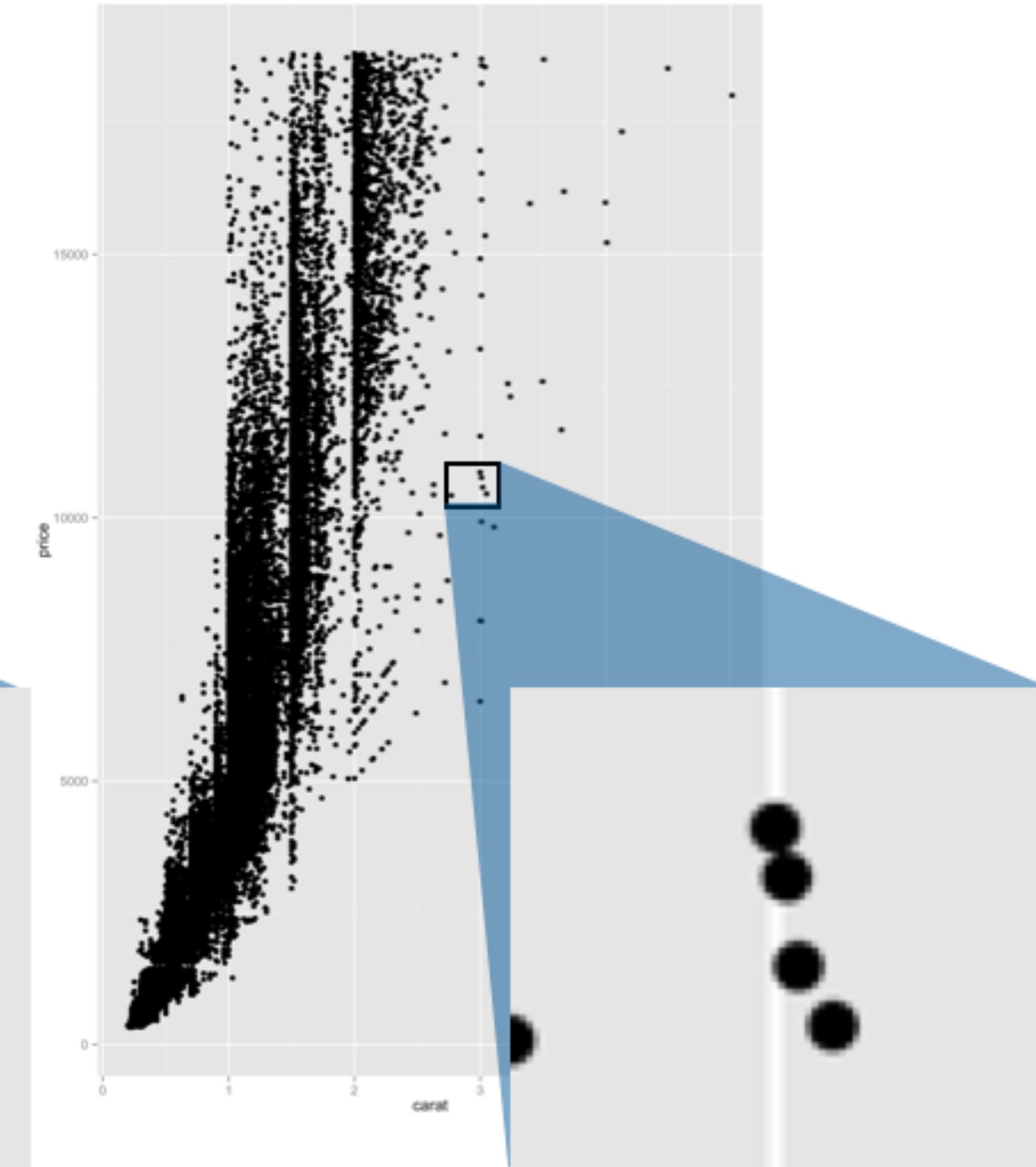
Specify size in inches

```
ggsave("my-plot.pdf", width = 6, height = 6)
```

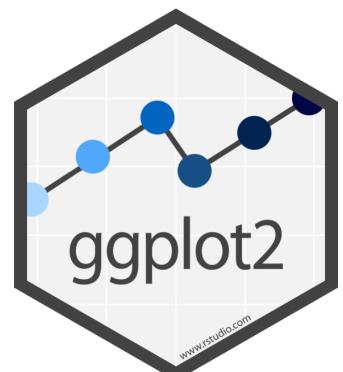




**PDF**  
vector based  
good for most plots

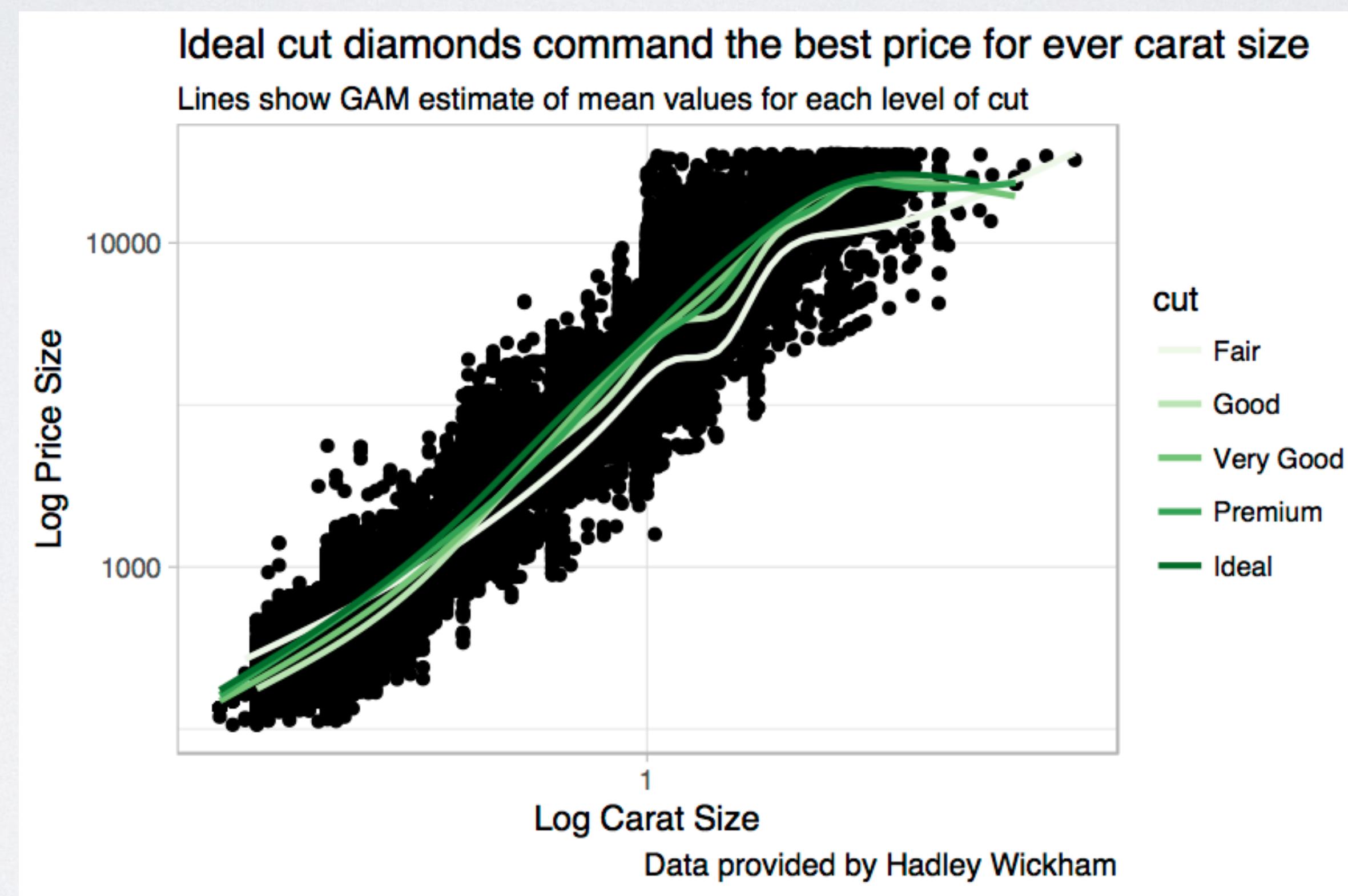


**PNG**  
raster based  
faster for many points



# Your turn

Save your last plot and then locate it in your files pane.



# Data Visualization with

