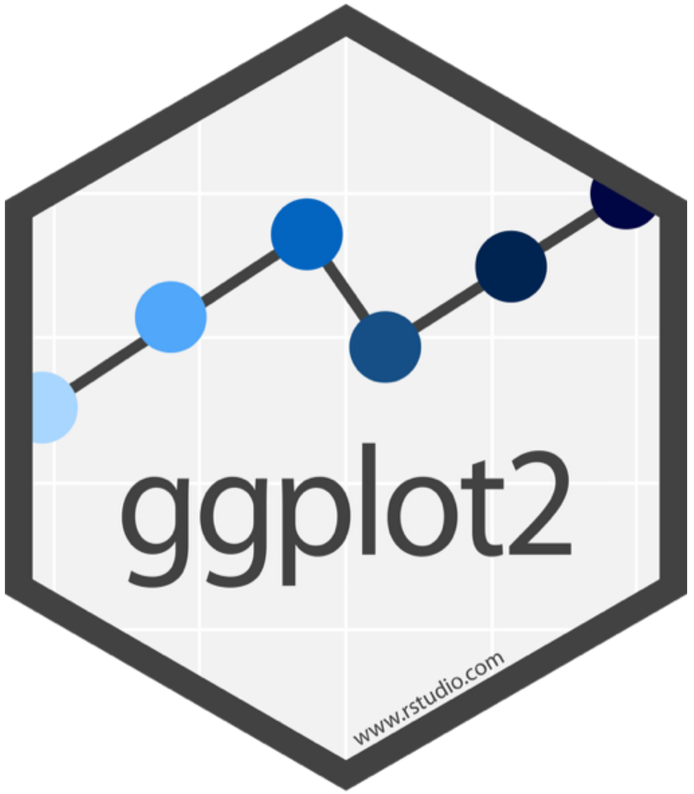
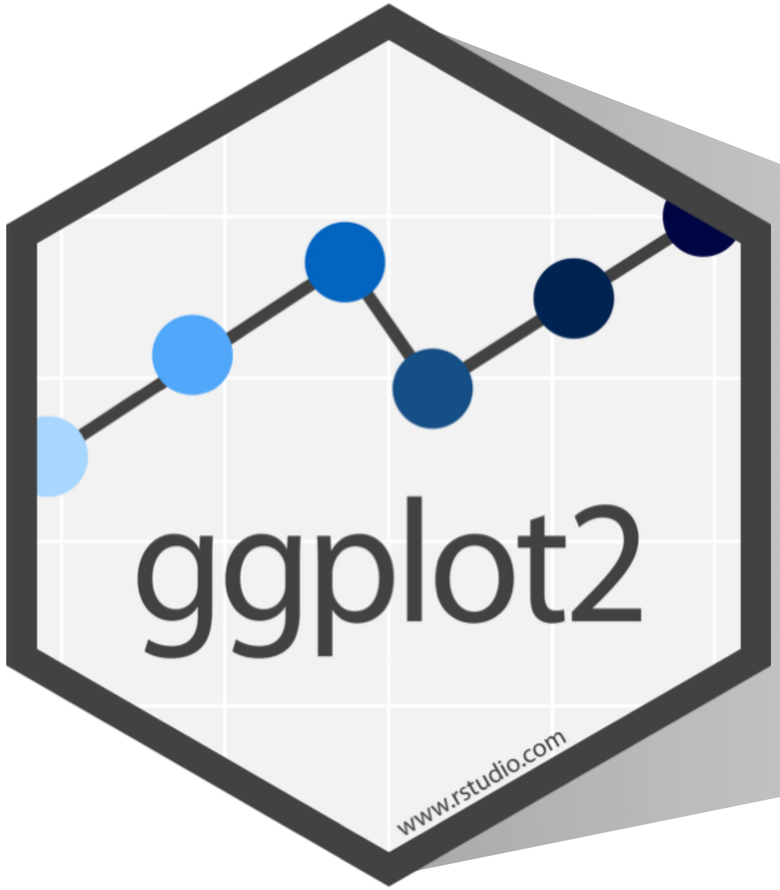
Data Visualization with



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

- John Tukey



hflights

namespace

fueleconomy

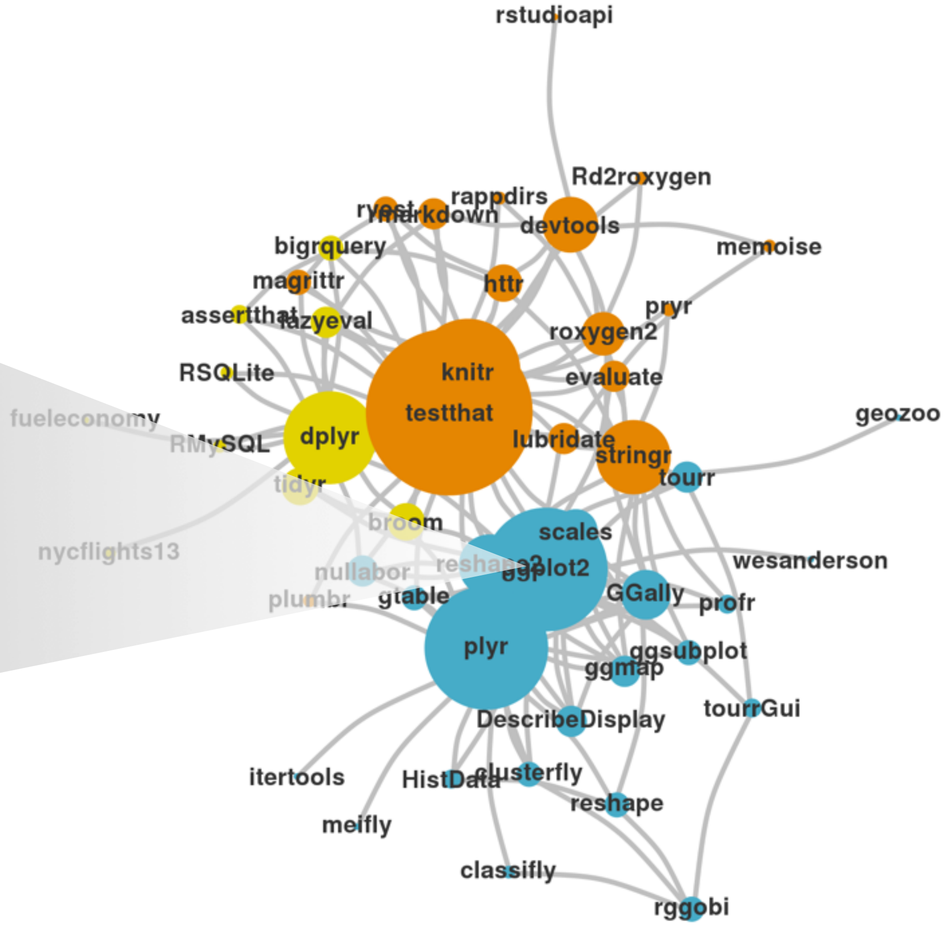
nycflights13

nasaweather

babynames

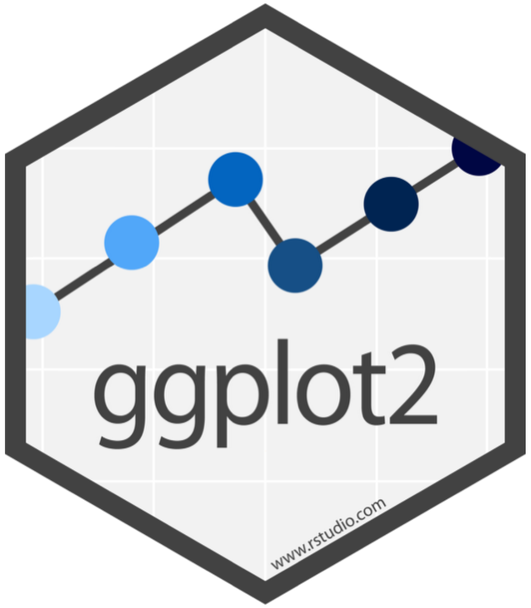
fda

plotrix



Display by Adolfo Álvarez

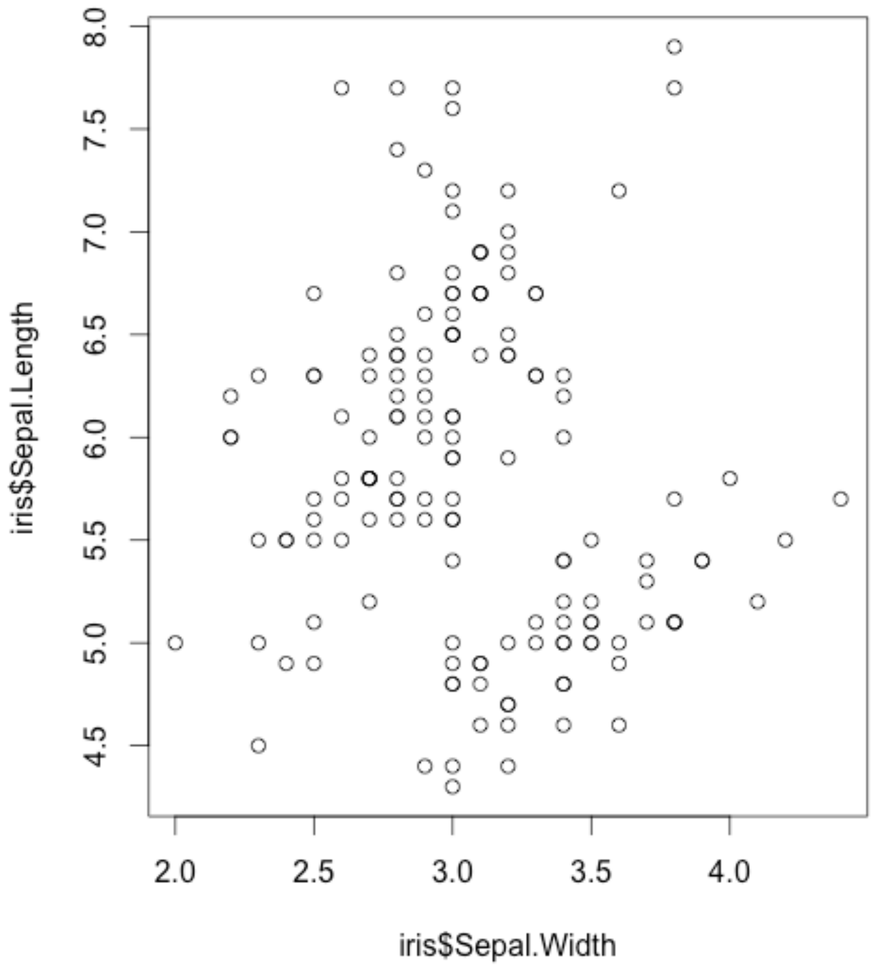
ggplot2



A package that visualizes data.

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

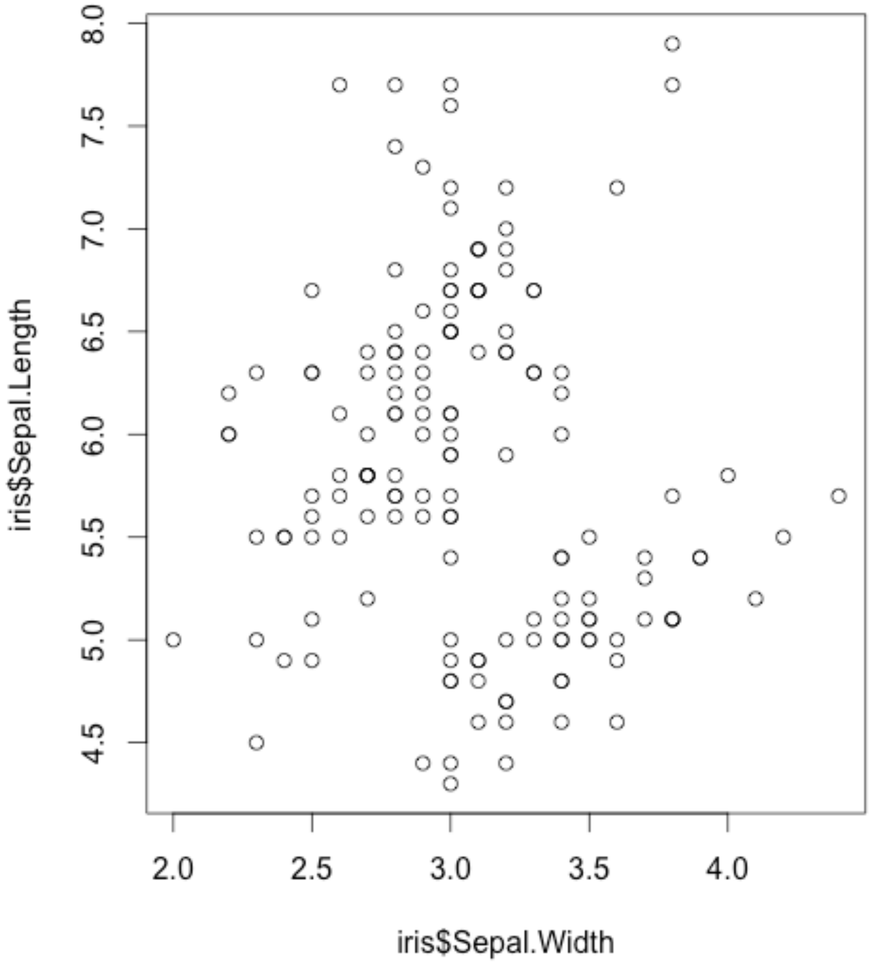
plot

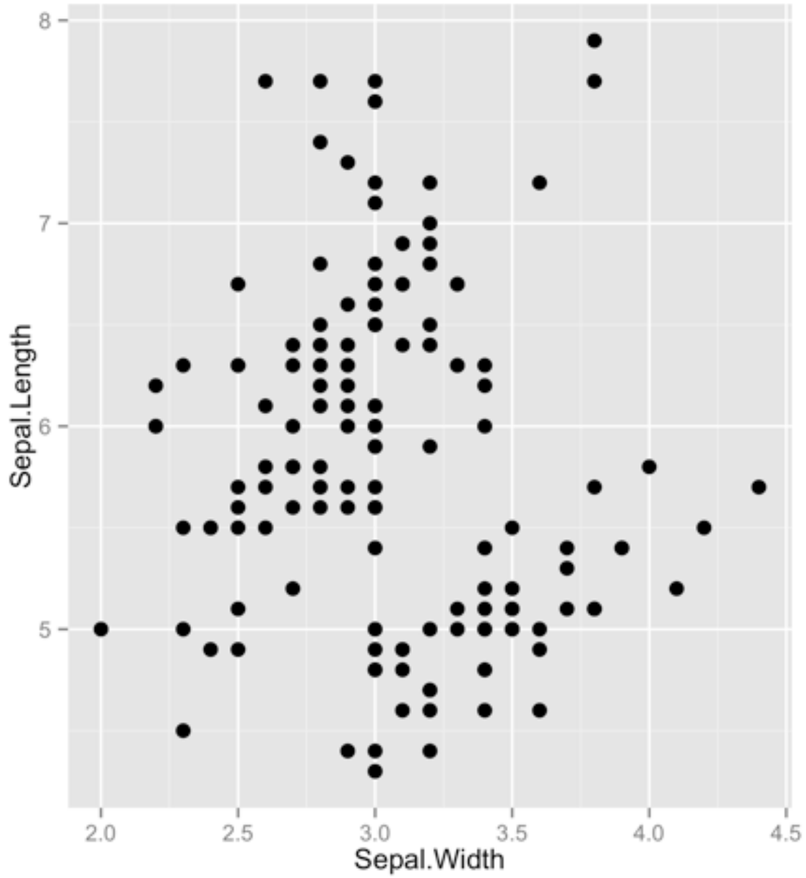



```
plot(iris$Sepal.Width,
     iris$Sepal.Length)
```

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

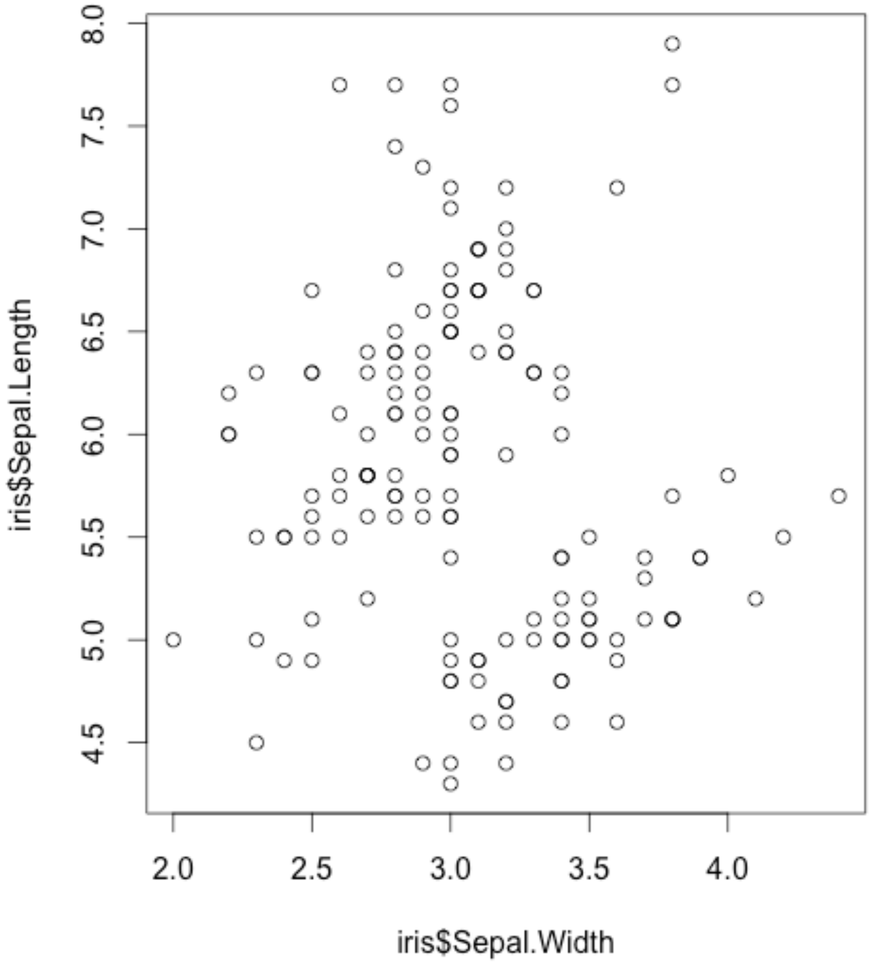
ggplot2

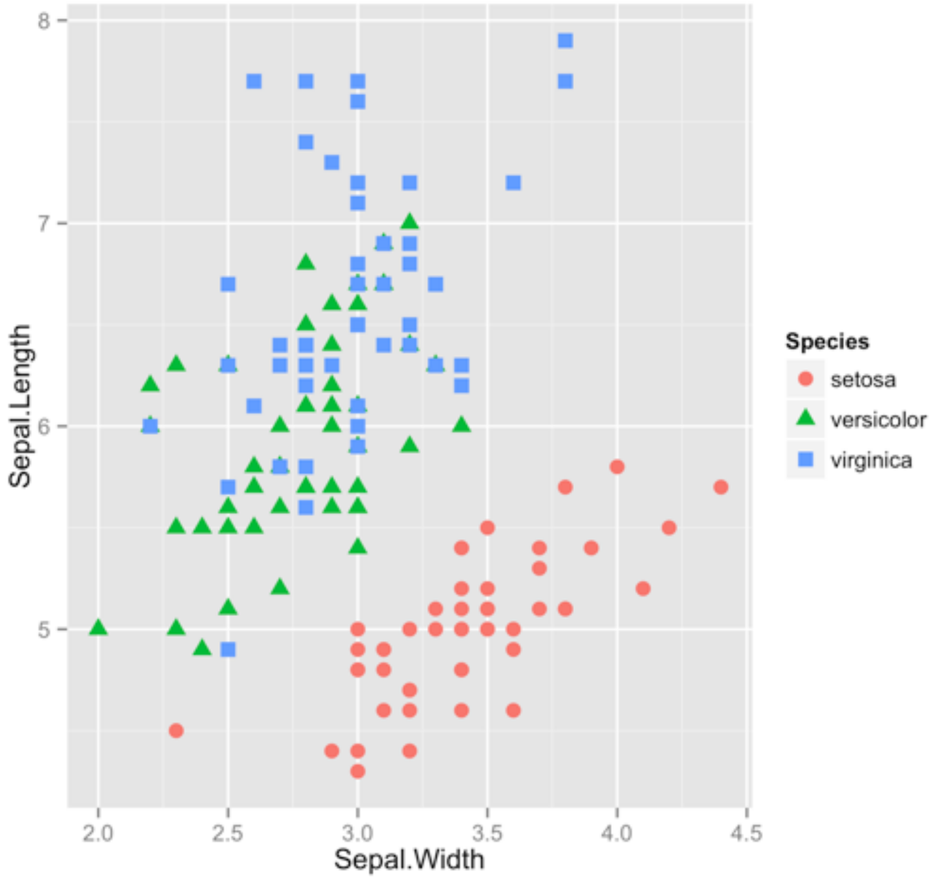

iris$Sepal.Length
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
2.0
2.5
3.0
3.5
4.0
iris$Sepal.Width


Sepal.Length
8
7
6
5
2.0
2.5
3.0
3.5
4.0
4.5
Sepal.Width


ggplot2

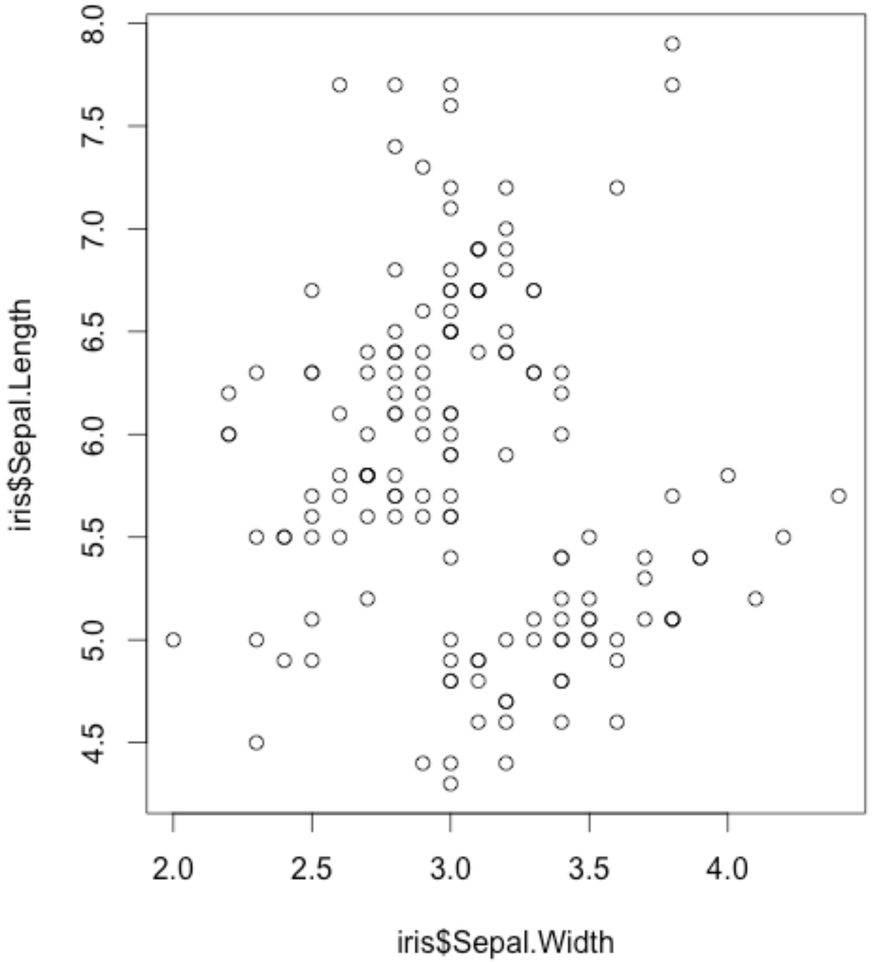
ggplot2

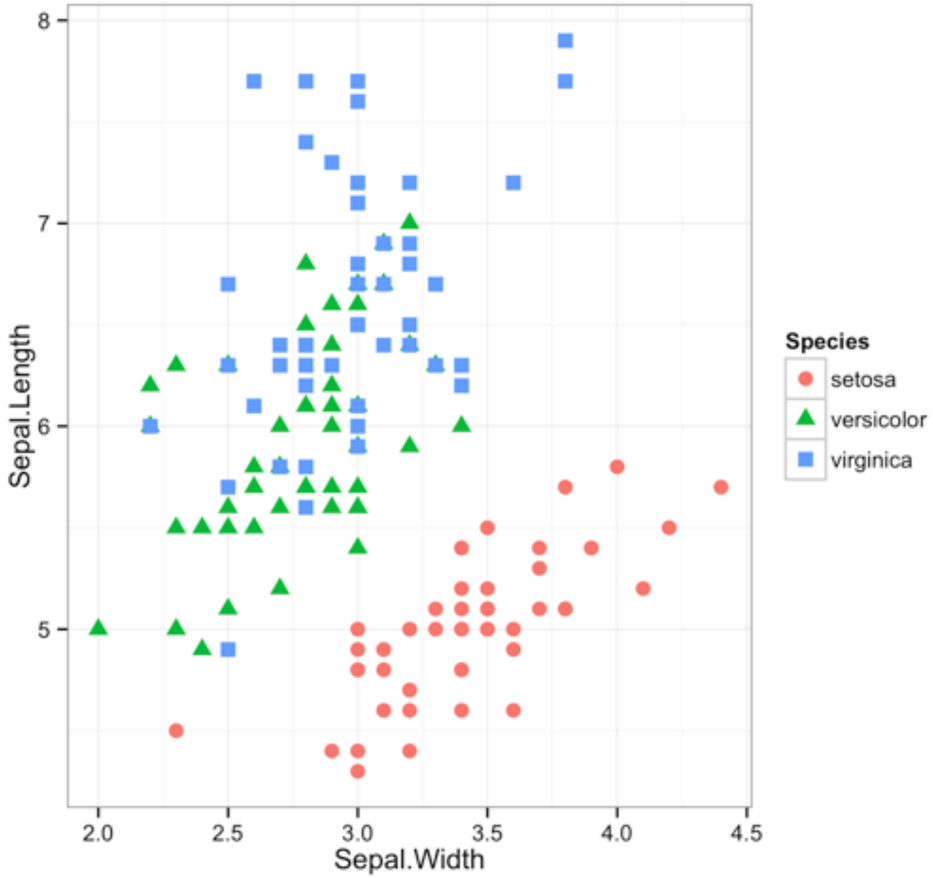

iris$Sepal.Length
iris$Sepal.Width
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
2.0
2.5
3.0
3.5
4.0


Sepal.Length
Sepal.Width
8
7
6
5
2.0
2.5
3.0
3.5
4.0
4.5
Species
setosa
versicolor
virginica

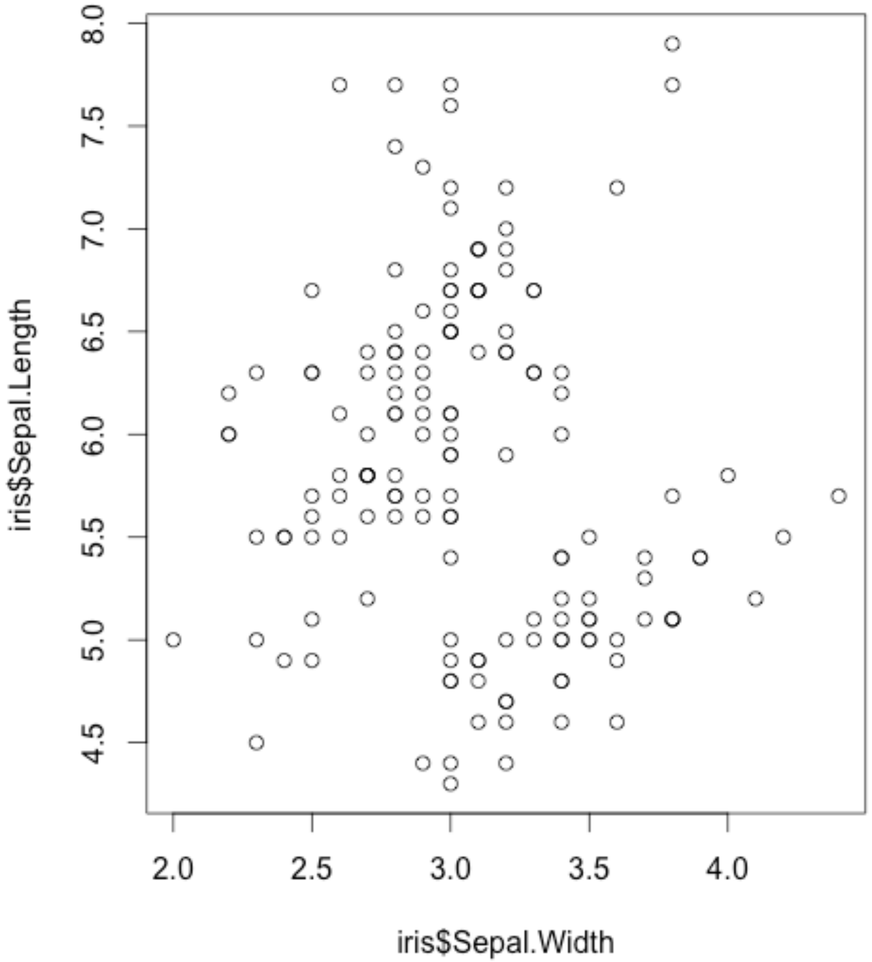

ggplot2

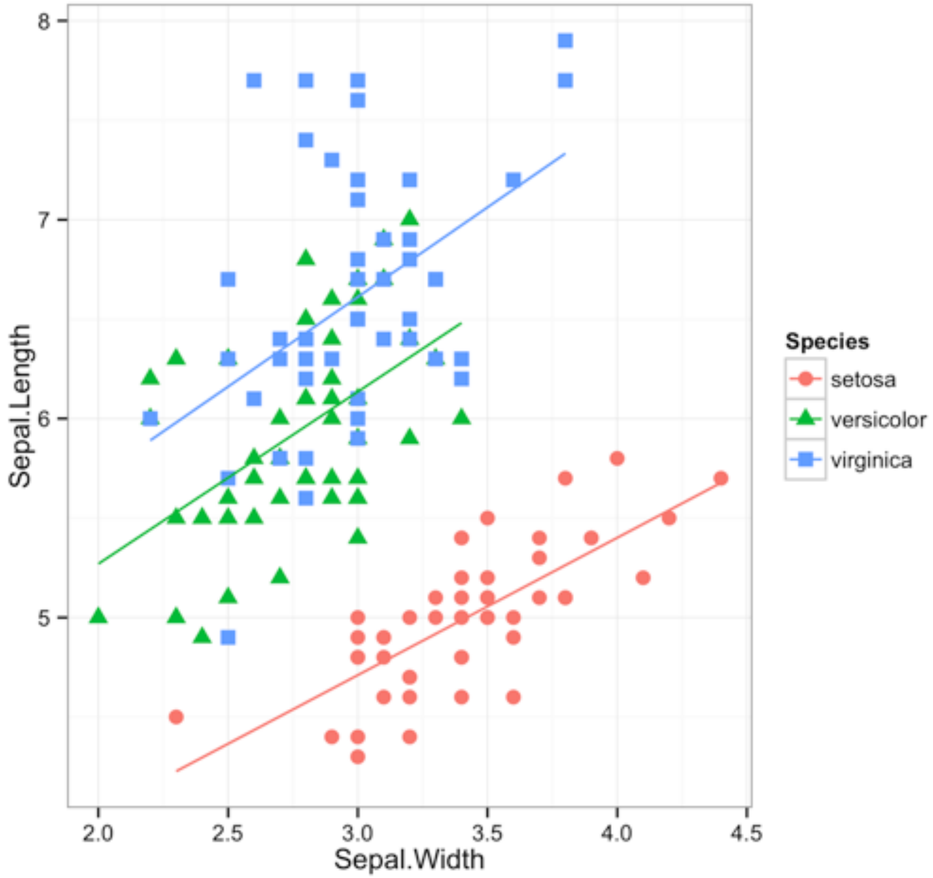
ggplot2


iris$Sepal.Length
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
2.0
2.5
3.0
3.5
4.0
iris$Sepal.Width


Sepal.Length
8
7
6
5
2.0
2.5
3.0
3.5
4.0
4.5
Sepal.Width
Species
setosa
versicolor
virginica

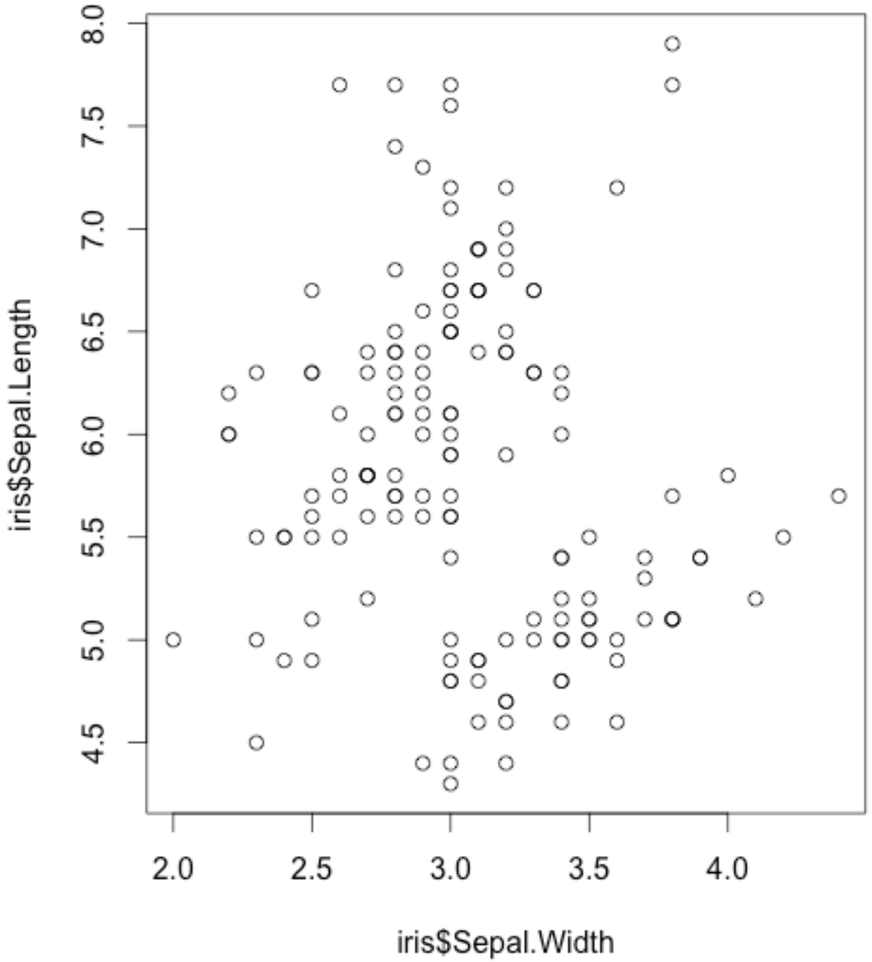
ggplot2

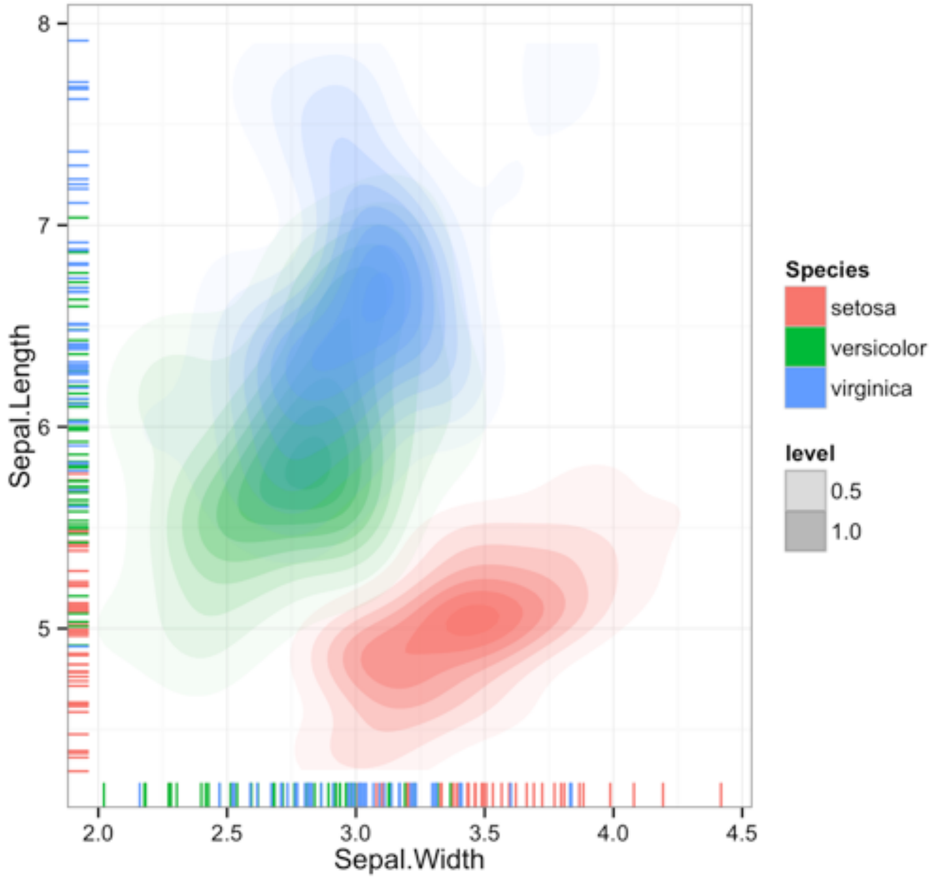

iris$Sepal.Length
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
2.0
2.5
3.0
3.5
4.0
iris$Sepal.Width


Sepal.Length
8
7
6
5
2.0
2.5
3.0
3.5
4.0
4.5
Sepal.Width
Species
setosa
versicolor
virginica


ggplot2

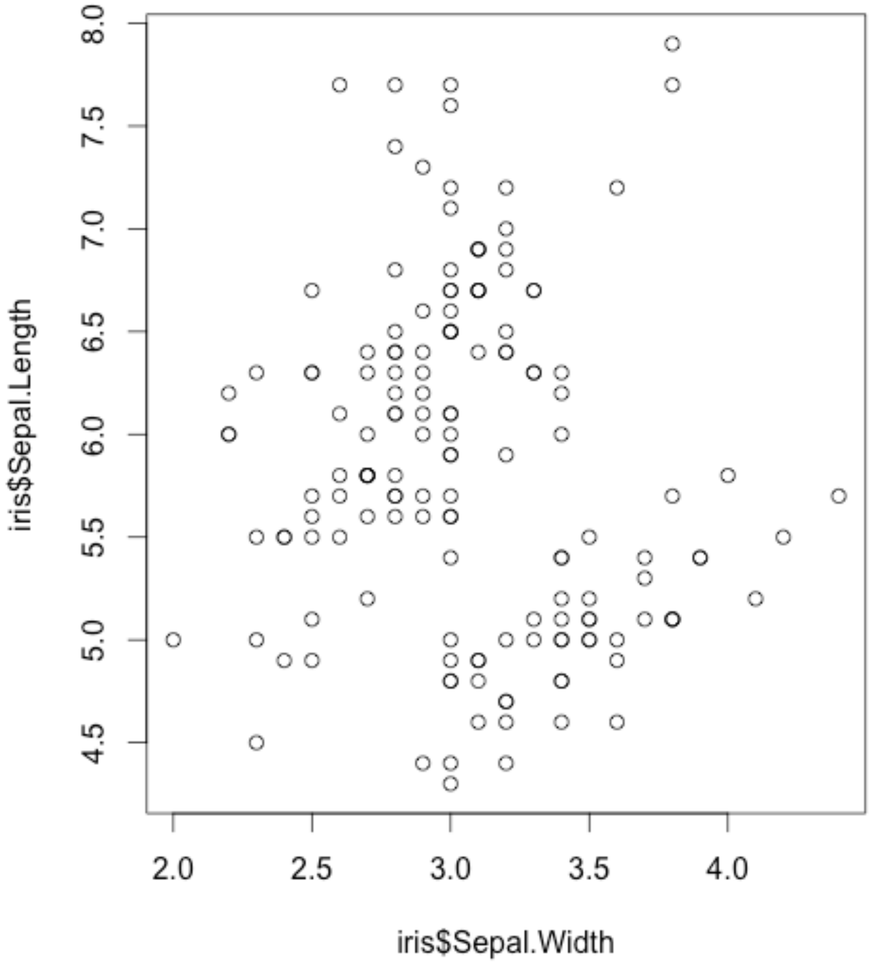
ggplot2

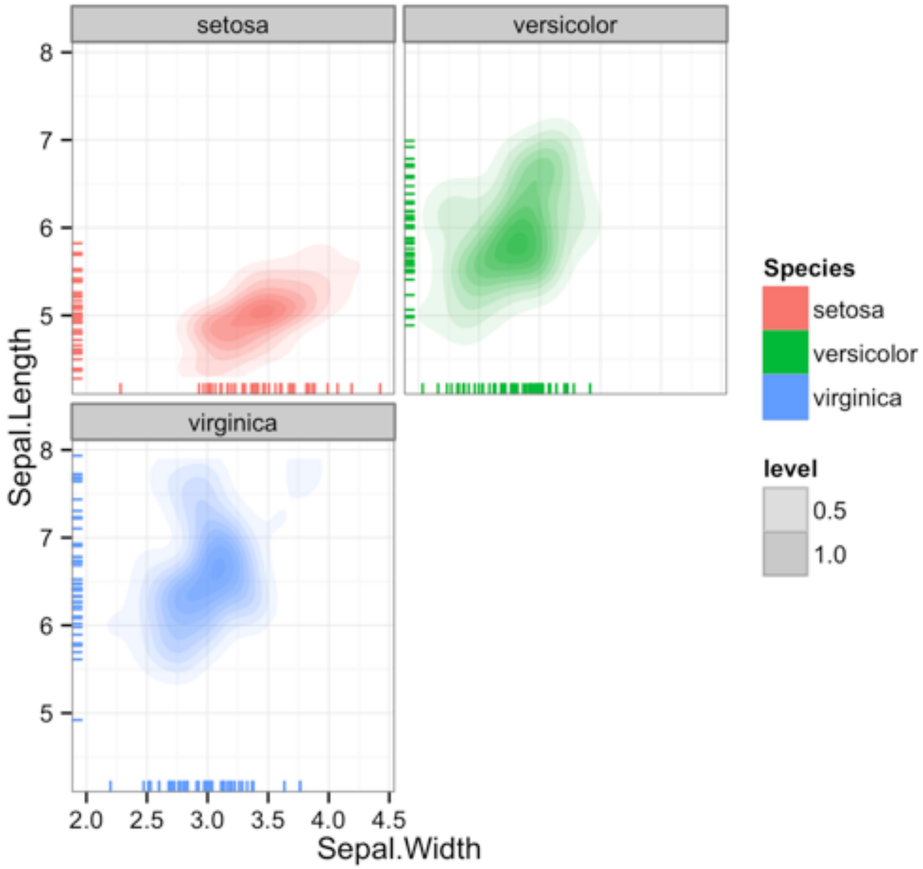

iris$Sepal.Length
iris$Sepal.Width
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
2.0
2.5
3.0
3.5
4.0


Sepal.Length
Sepal.Width
5
6
7
8
2.0
2.5
3.0
3.5
4.0
4.5
Species
setosa
versicolor
virginica
level
0.5
1.0


ggplot2

ggplot2


iris$Sepal.Length
iris$Sepal.Width
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
2.0
2.5
3.0
3.5
4.0


setosa
versicolor
virginica
Sepal.Length
Sepal.Width
Species
setosa
versicolor
virginica
level
0.5
1.0

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)?

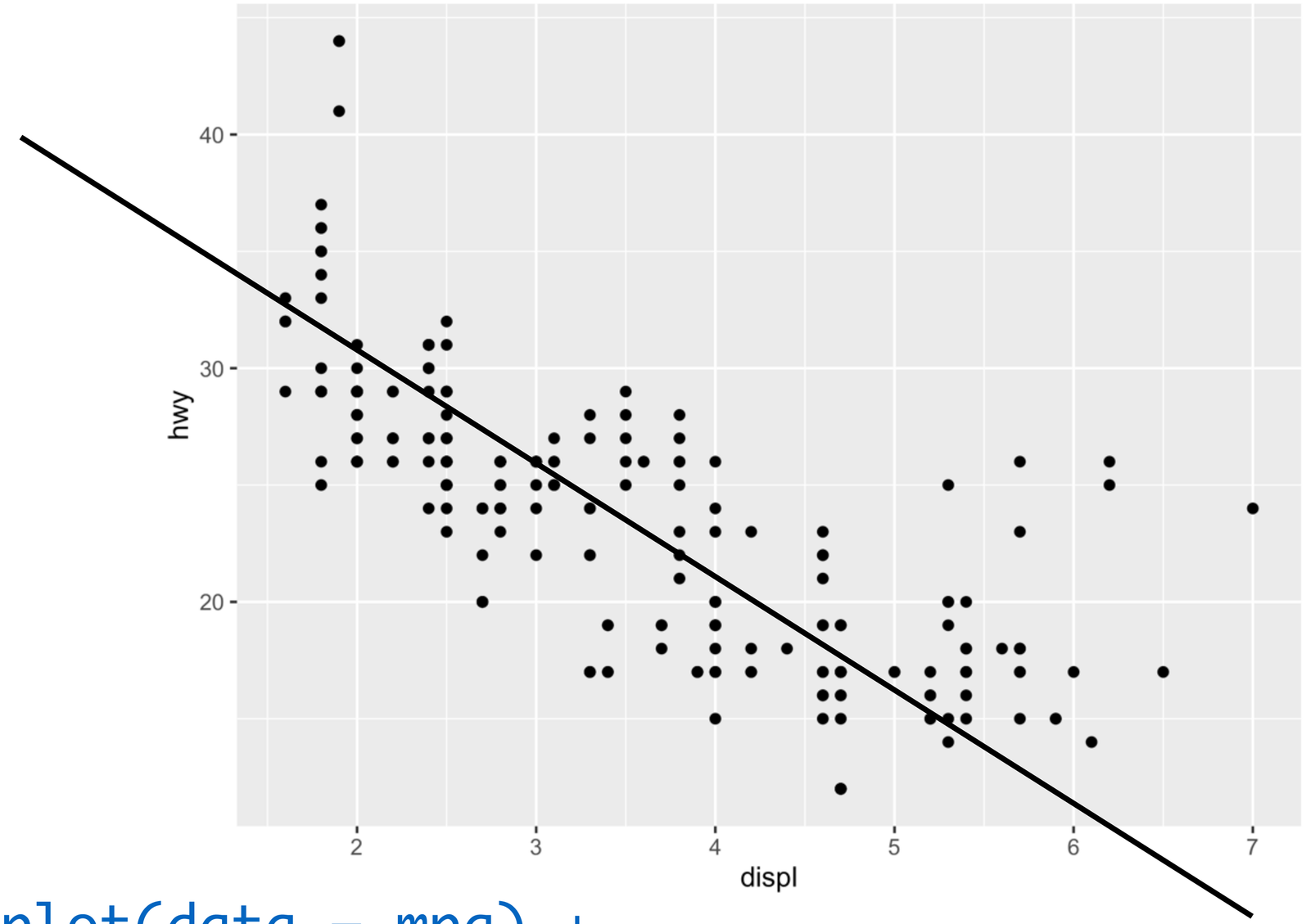
No peeking ahead!

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

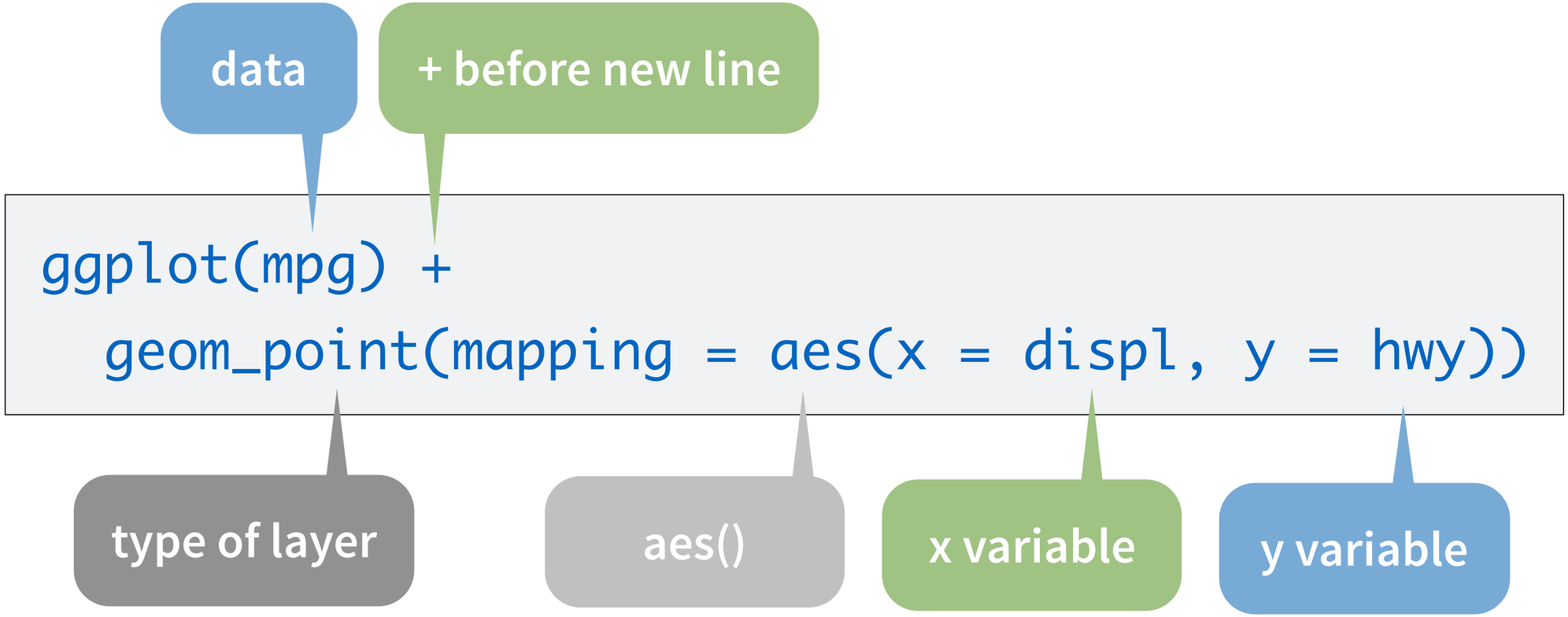
1. "Initialize" a plot with ggplot()
2. Add layers with geom_ functions

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



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))
```

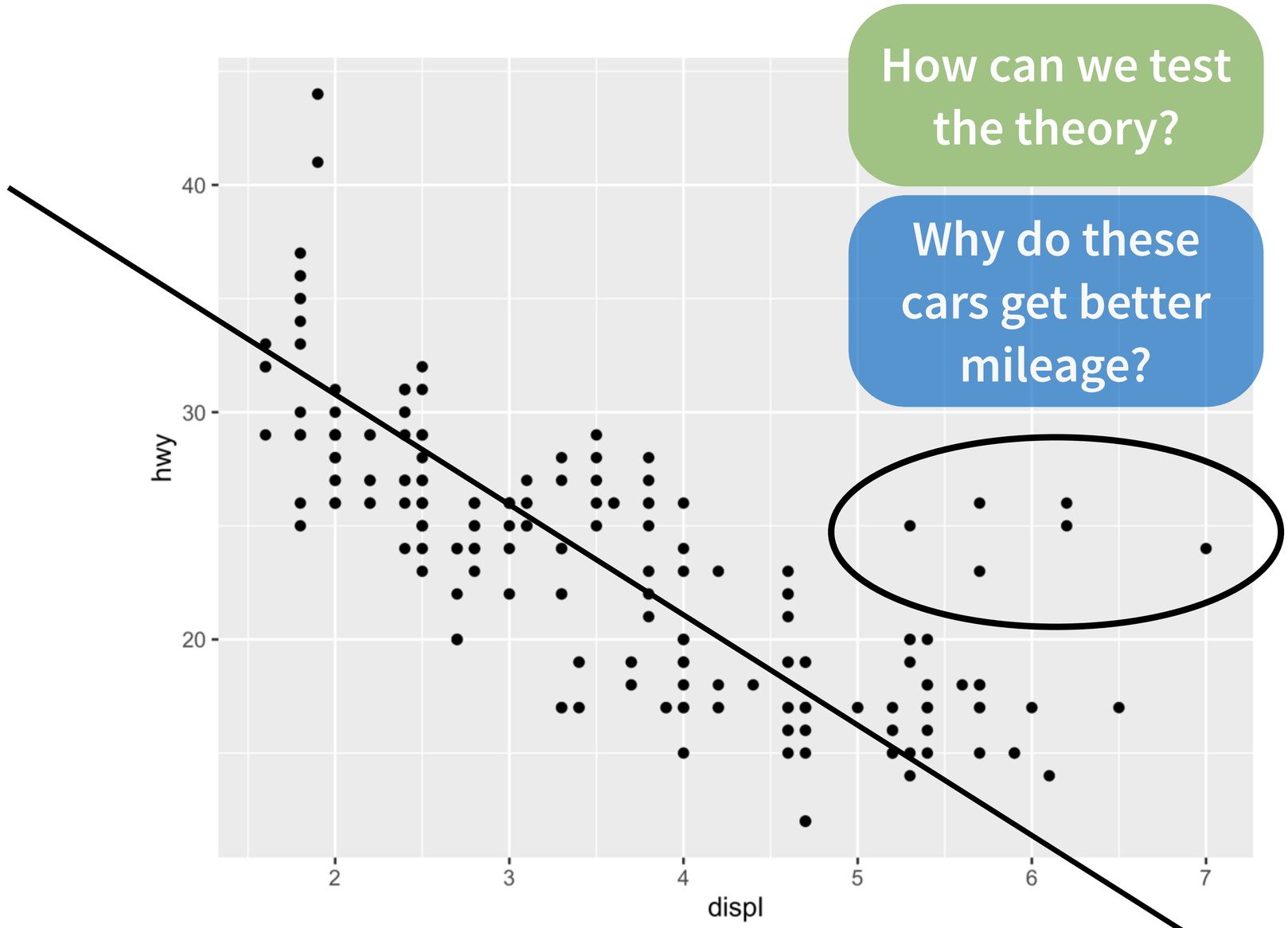
```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

The variables to visualize

Visualizing variables (aesthetic mappings)

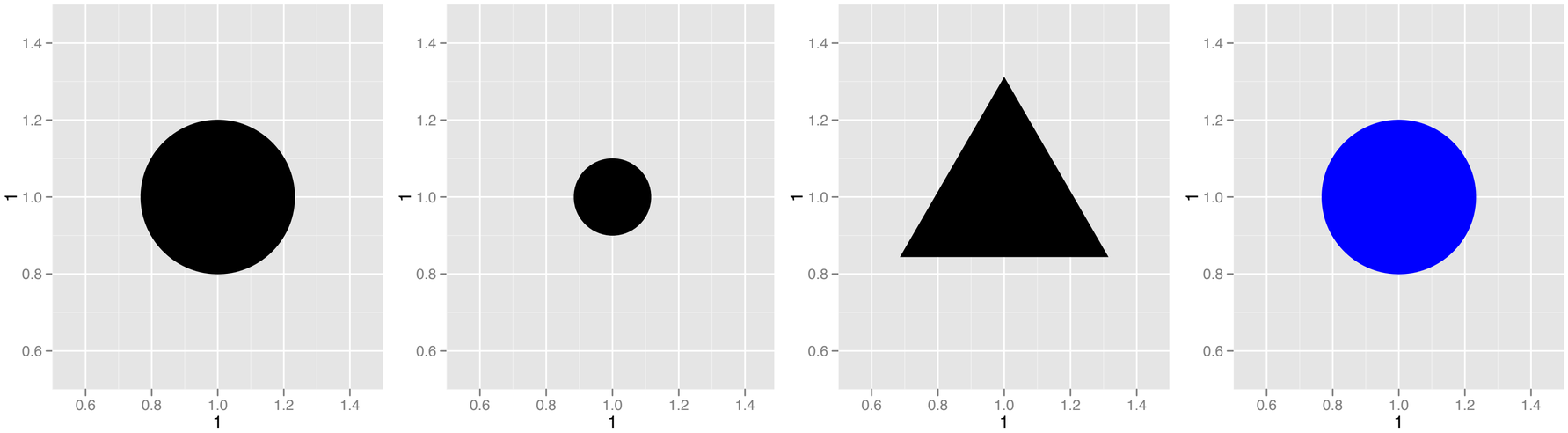
"The greatest value of a picture is when it forces us to notice what we never expected to see."

- John Tukey




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

Aesthetics

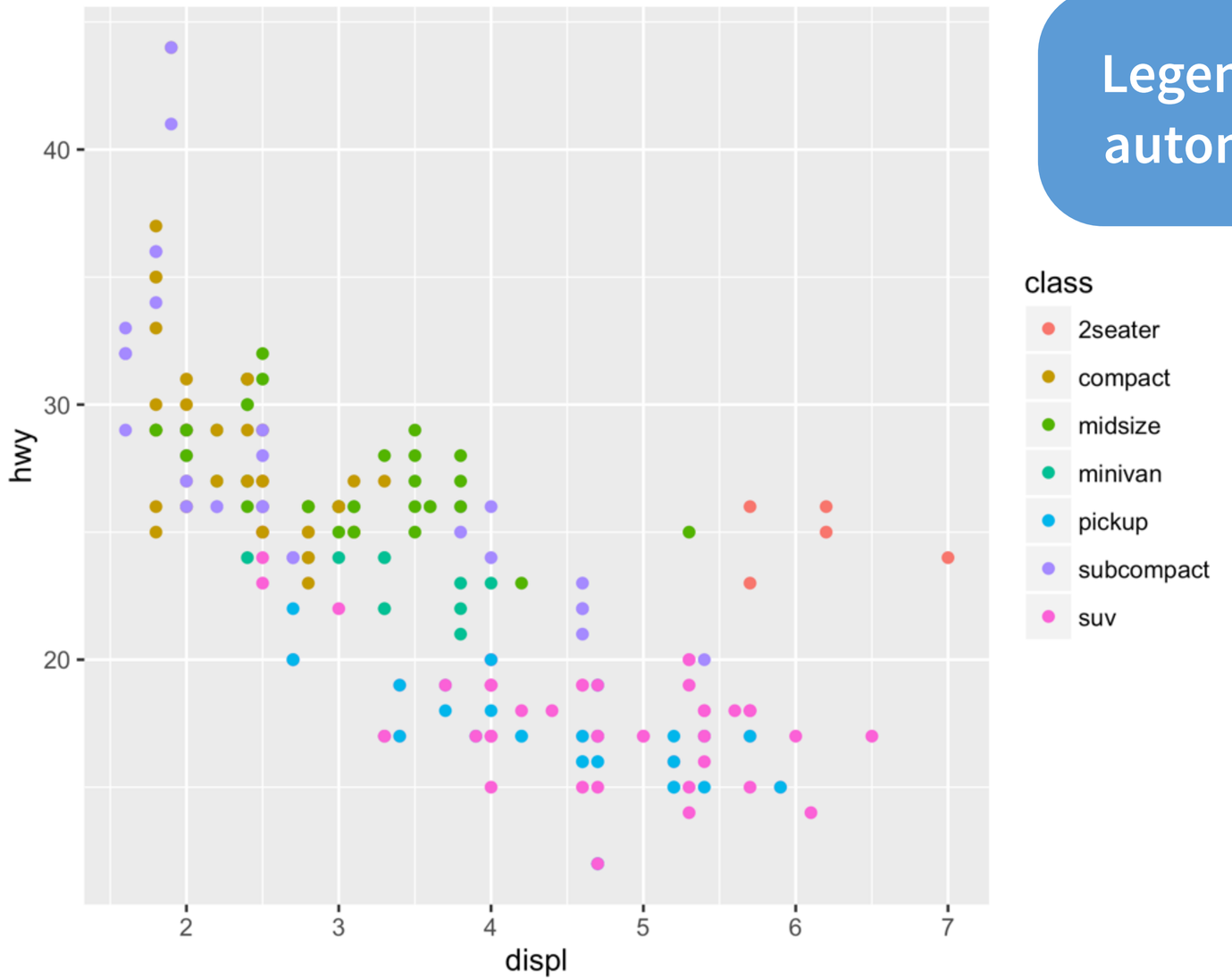


Aesthetics

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))
```



Legend added automatically

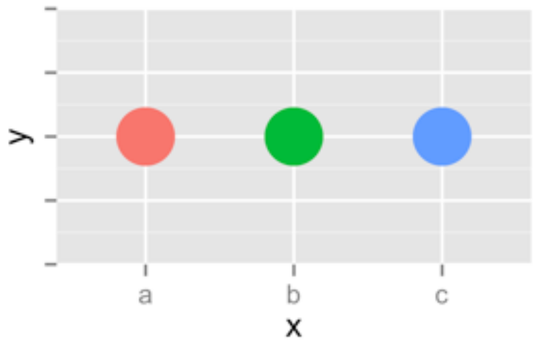
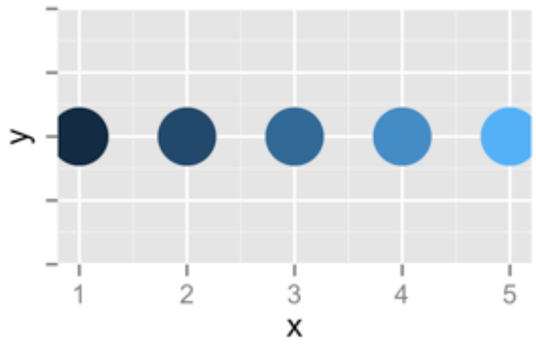
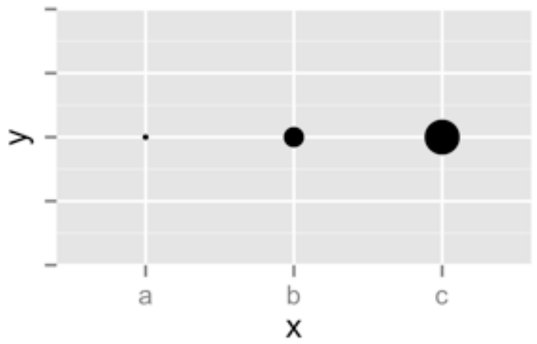
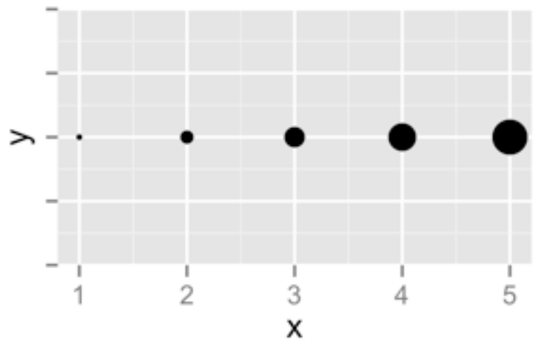
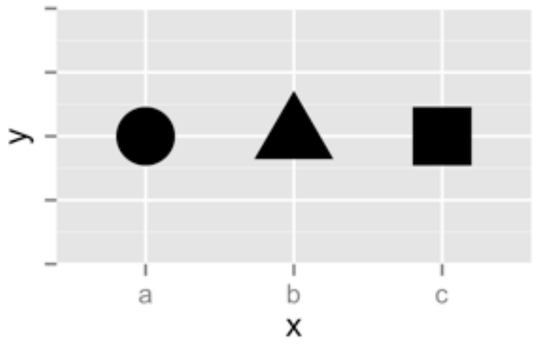
```
ggplot(mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```

Your Turn

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

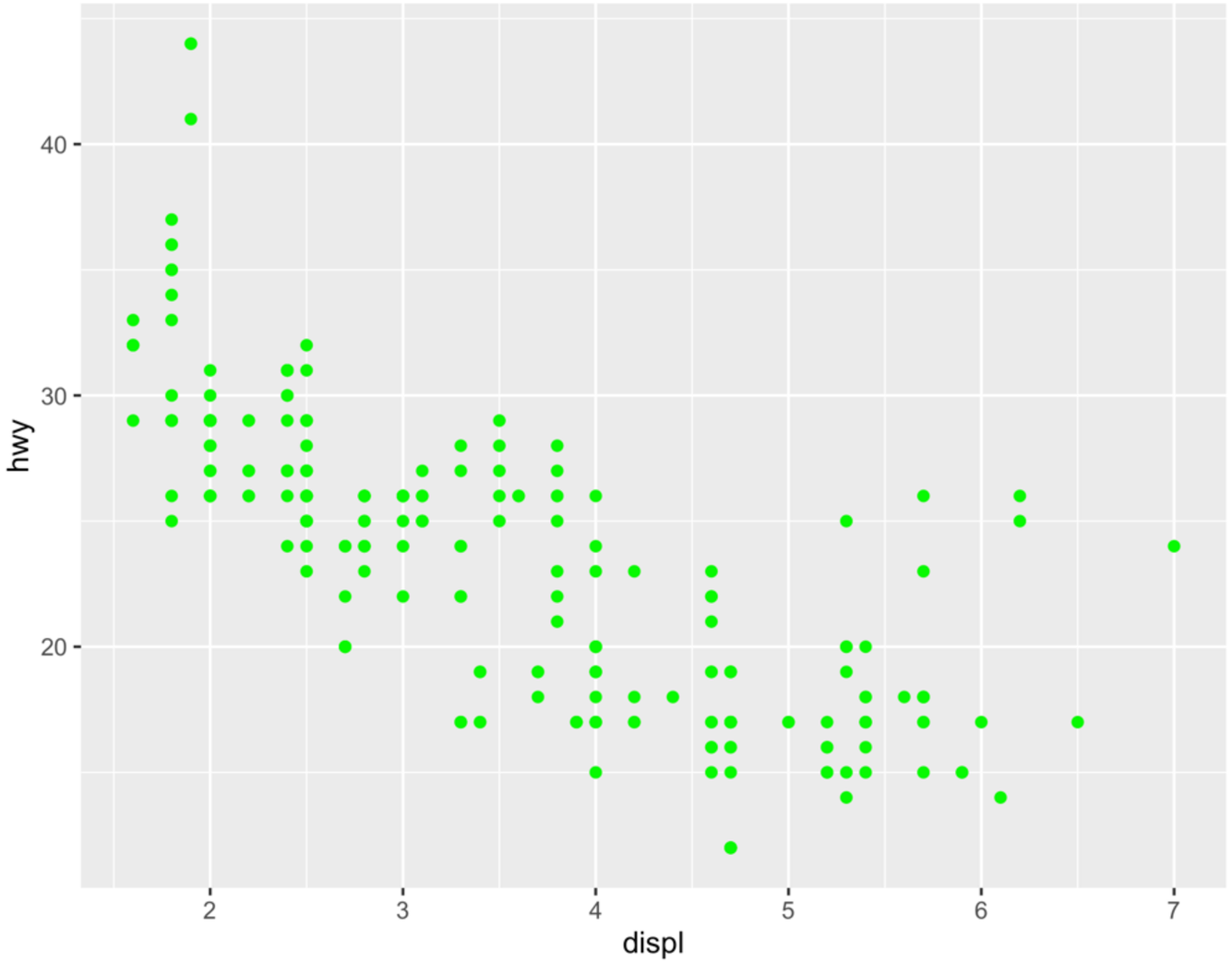
Do different things happen when you map aesthetics to discrete and continuous variables?

What happens when you use more than one aesthetic?

	Discrete	Continuous
Color		
Size		
Shape		

set vs. map

How would you make this plot?



Set vs. map

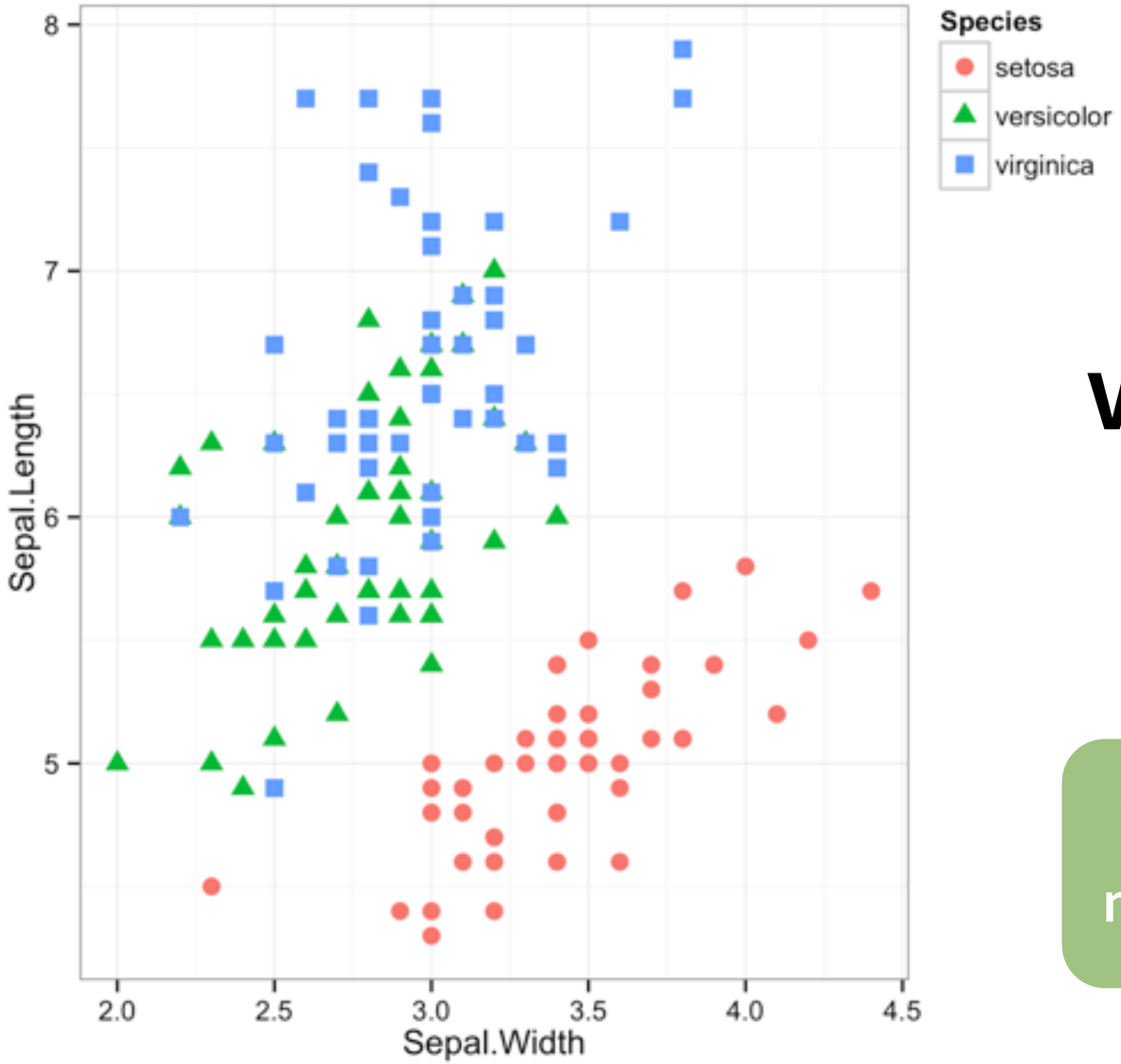
Inside of aes(): ggplot2 treats input as value in the data space and maps it to a value in the visual space.

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



Visual Space ⟷ Data Space

Red ⟷ setosa

Green ⟷ versicolor

Blue ⟷ virginica

Values outside mapping = aes()

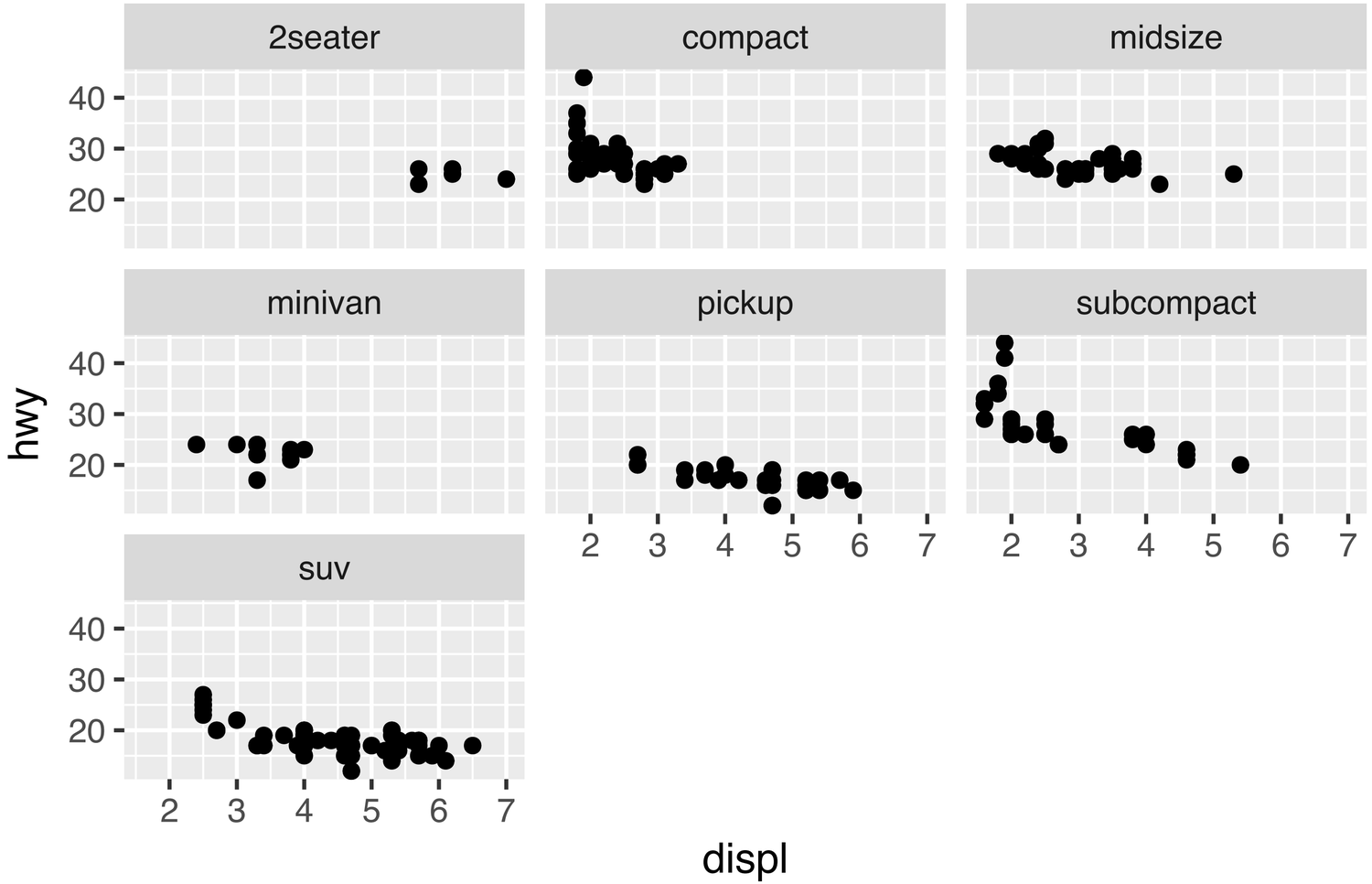
Values inside mapping = aes()

ggplot2

Facets

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

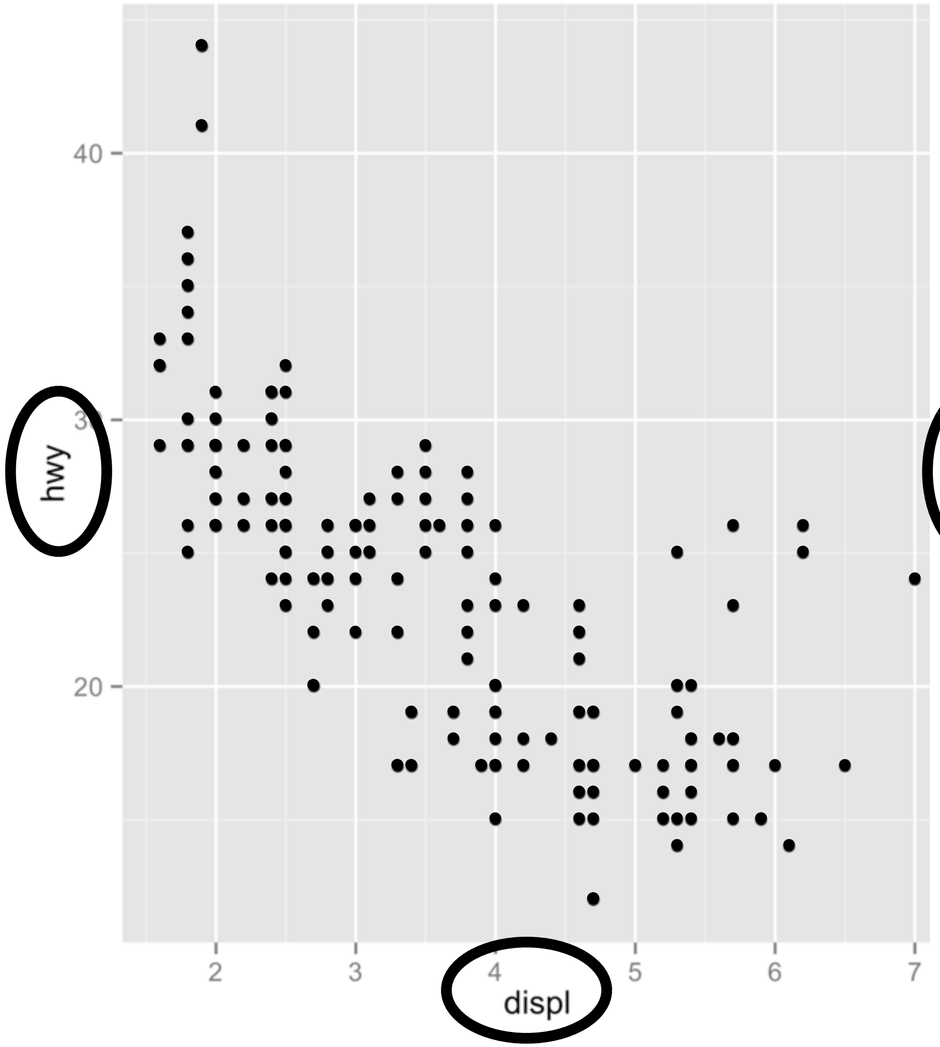
```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>)) +
  <FACET_FUNCTION>
```

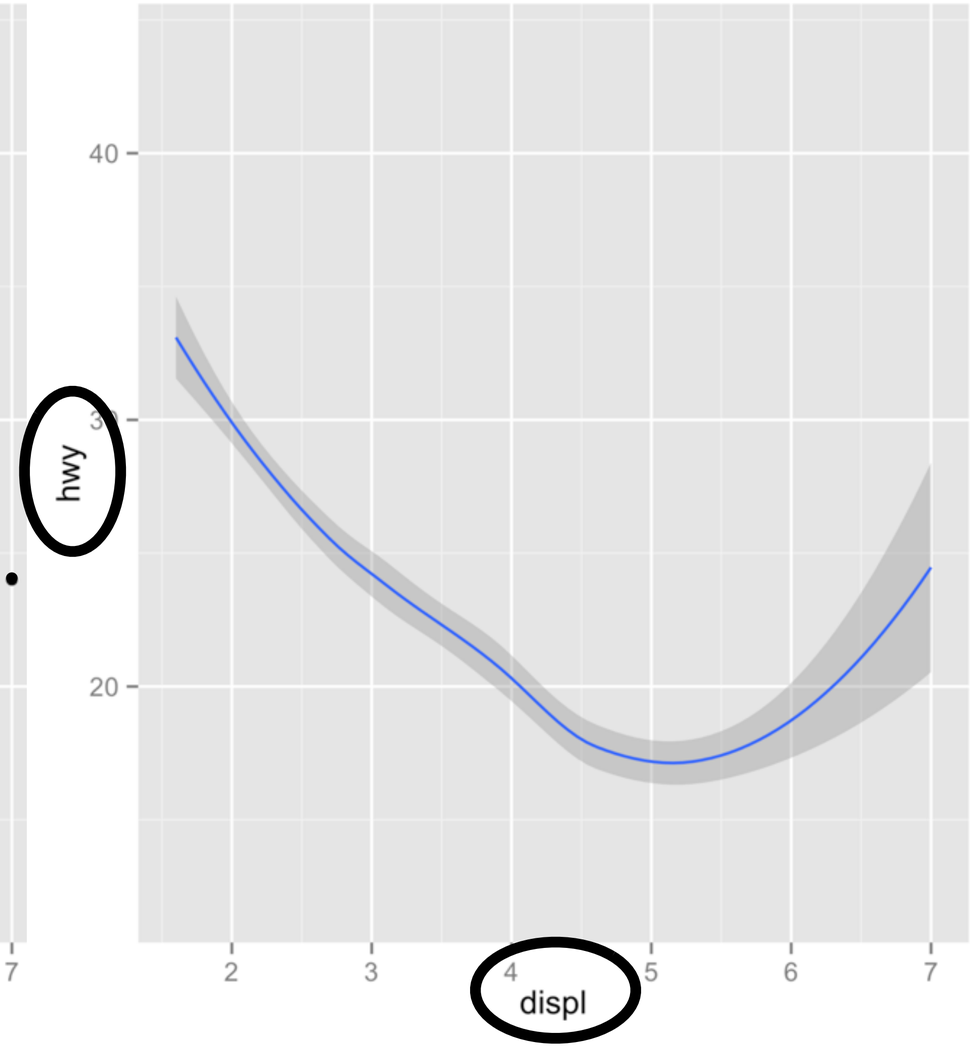
Visualizing cases

(geoms)

How are these plots similar?

Same: x var , y var , data



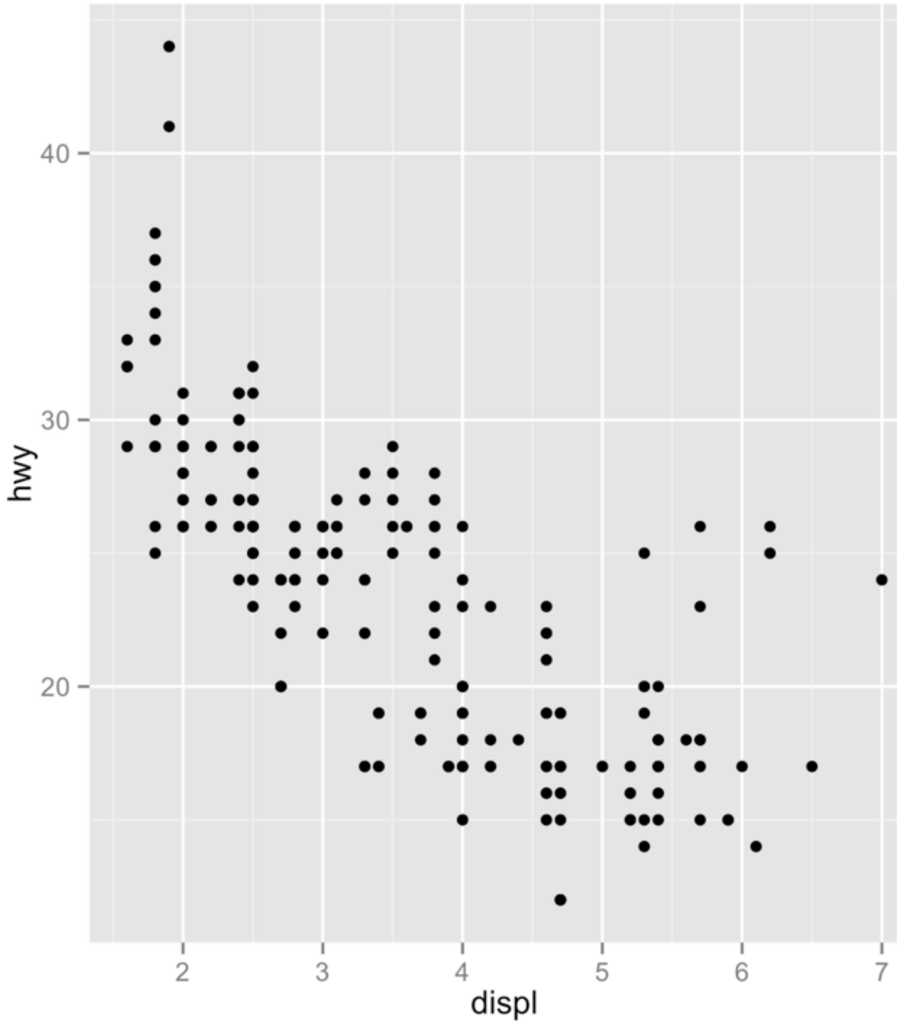


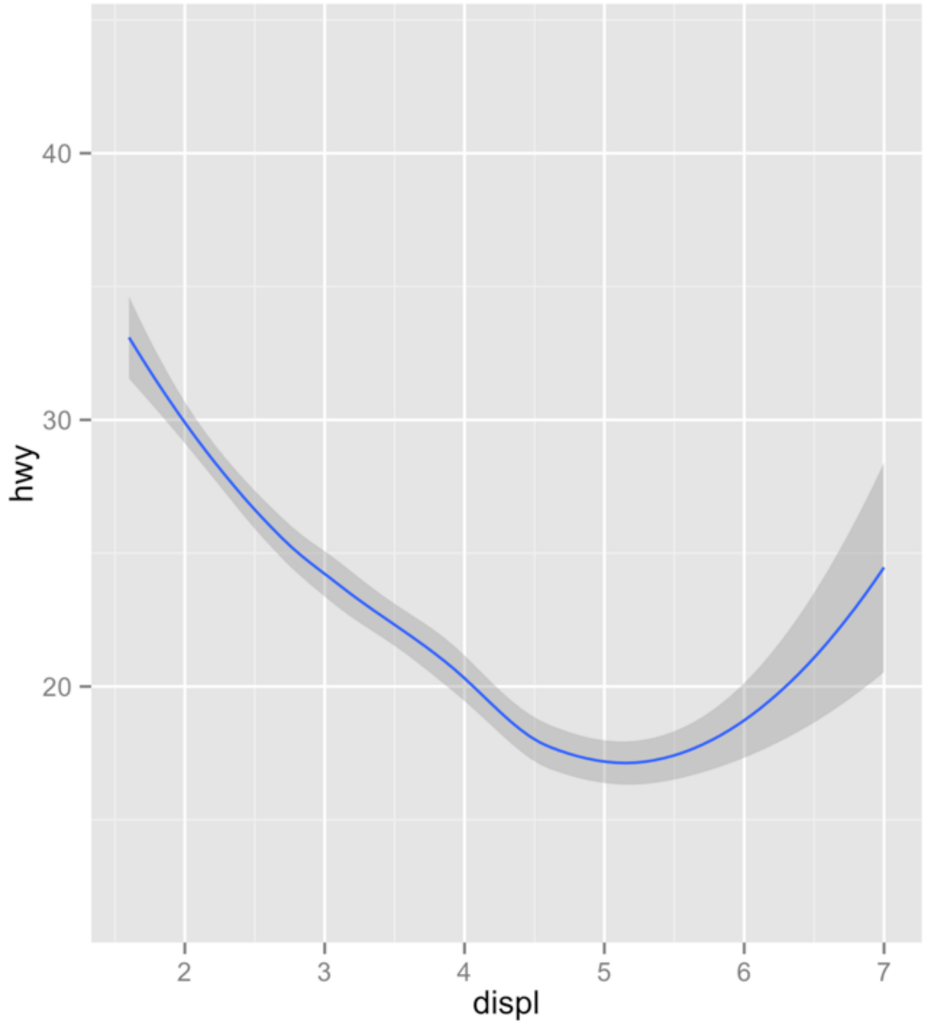
ggplot2

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.

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

- a + **geom_area(stat = "bin")** — x, y, alpha, color, fill, linetype, size; b + geom_area(aes(y = ..density..), stat = "bin")
- a + **geom_density(**kernel = "gaussian"**)** — x, y, alpha, color, fill, linetype, size, weight; b + geom_density(aes(y = ..county..))
- a + **geom_dotplot()** — x, y, alpha, color, fill
- a + **geom_freqpoly()** — x, y, alpha, color, linetype, size; b + geom_freqpoly(aes(y = ..density..))
- a + **geom_histogram(**binwidth = 5**)** — x, y, alpha, color, fill, linetype, size, weight; b + geom_histogram(aes(y = ..density..))

Discrete

b <- ggplot(mpg, aes(fl))

- b + **geom_bar()** — x, alpha, color, fill, linetype, size, weight

Graphical Primitives

map <- map_data("state")
c <- ggplot(map, aes(long, lat))

- c + **geom_polygon(**aes(group = group)**)** — x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

- d + **geom_path(**lineend="butt", linejoin="round', linemitre=1**)** — x, y, alpha, color, linetype, size
- d + **geom_ribbon(**aes(ymin=unemploy - 900, ymax=unemploy + 900**)**) — x, ymax, ymin, alpha, color, fill, linetype, size

e <- ggplot(seals, aes(x = long, y = lat))

- e + **geom_segment(**aes(xend = long + delta_long, yend = lat + delta_lat**)**) — x, xend, y, yend, alpha, color, linetype, size
- e + **geom_rect(**aes(xmin = long, ymin = lat, xmax= long + delta_long, ymax = lat + delta_lat**)**) — xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

- f + **geom_blank()** — (Useful for expanding limits)
- f + **geom_jitter()** — x, y, alpha, color, fill, shape, size
- f + **geom_point()** — x, y, alpha, color, fill, shape, size
- f + **geom_quantile()** — x, y, alpha, color, linetype, size, weight
- f + **geom_rug(**sides = "bl"**)** — alpha, color, linetype, size
- f + **geom_smooth(**method = lm**)** — x, y, alpha, color, fill, linetype, size, weight
- f + **geom_text(**aes(label = cty)**)** — x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

- g + **geom_bar(stat = "identity")** — x, y, alpha, color, fill, linetype, size, weight
- g + **geom_boxplot()** — lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight
- g + **geom_dotplot(**binaxis = "y", stackdir = "center"**)** — x, y, alpha, color, fill
- g + **geom_violin(**scale = "area"**)** — x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

- h + **geom_jitter()** — x, y, alpha, color, fill, shape, size

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

- i + **geom_bin2d(**binwidth = c(5, 0.5)**)** — xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight
- i + **geom_density2d()** — x, y, alpha, colour, linetype, size
- i + **geom_hex()** — x, y, alpha, colour, fill size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))

- j + **geom_area()** — x, y, alpha, color, fill, linetype, size
- j + **geom_line()** — x, y, alpha, color, linetype, size
- j + **geom_step(**direction = "hv"**)** — x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

- k + **geom_crossbar(**fatten = 2**)** — x, y, ymax, ymin, alpha, color, fill, linetype, size
- k + **geom_errorbar()** — x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)
- k + **geom_linerange()** — x, ymin, ymax, alpha, color, linetype, size
- k + **geom_pointrange()** — x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

- l + **geom_map(**aes(map_id = state), map = map**)** + **expand_limits(**x = map$long, y = map$lat**)** — map_id, alpha, color, fill, linetype, size

Three Variables

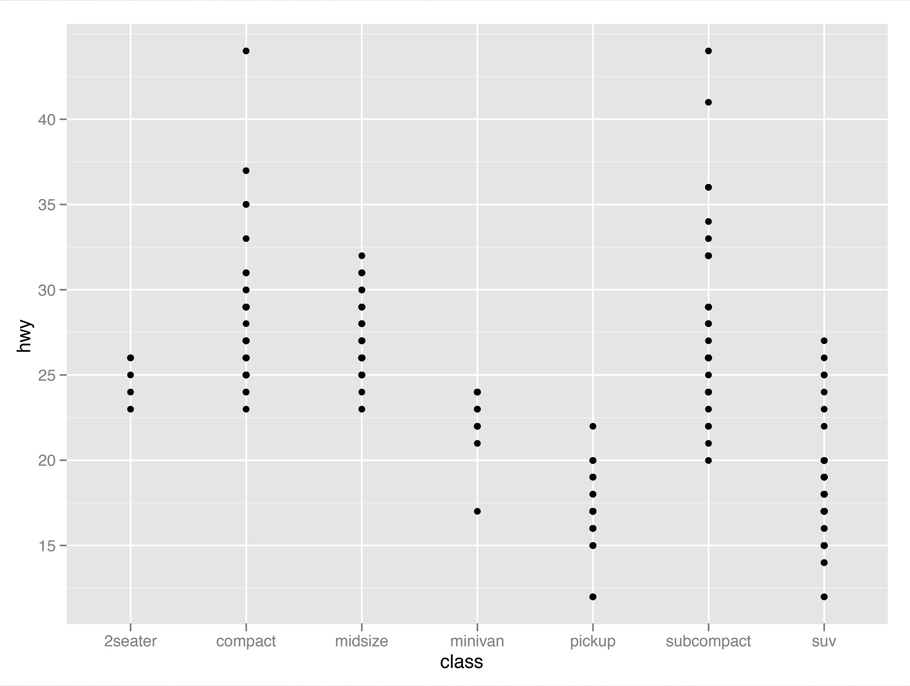
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

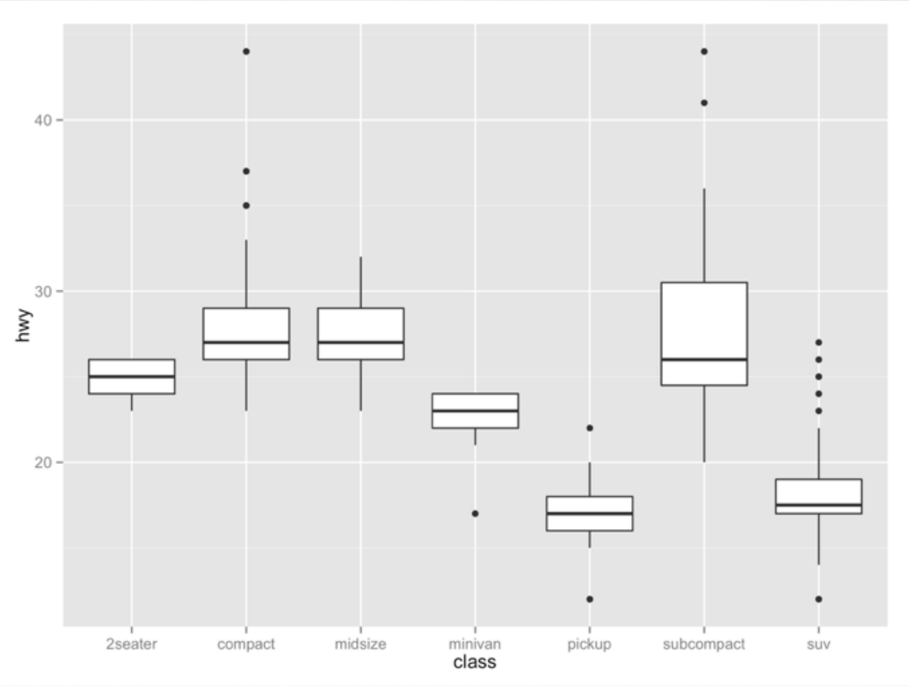
- m + **geom_contour(**aes(z = z)**)** — x, y, z, alpha, colour, linetype, size, weight
- m + **geom_raster(**aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)** — x, y, alpha, fill (fast)
- m + **geom_tile(**aes(fill = z)**)** — x, y, alpha, color, fill, linetype, size (slow)

ggplot2

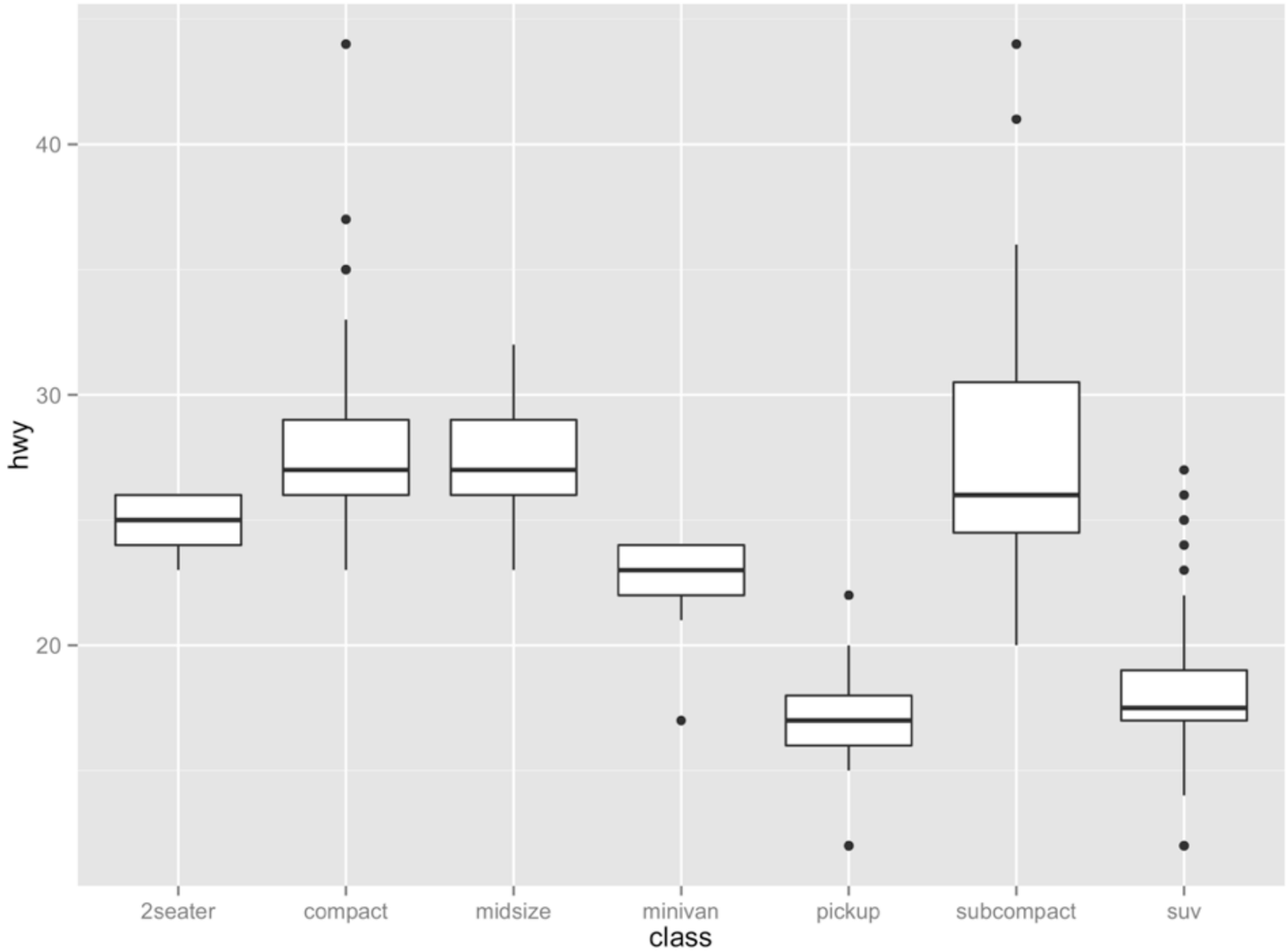
Your Turn

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))
```

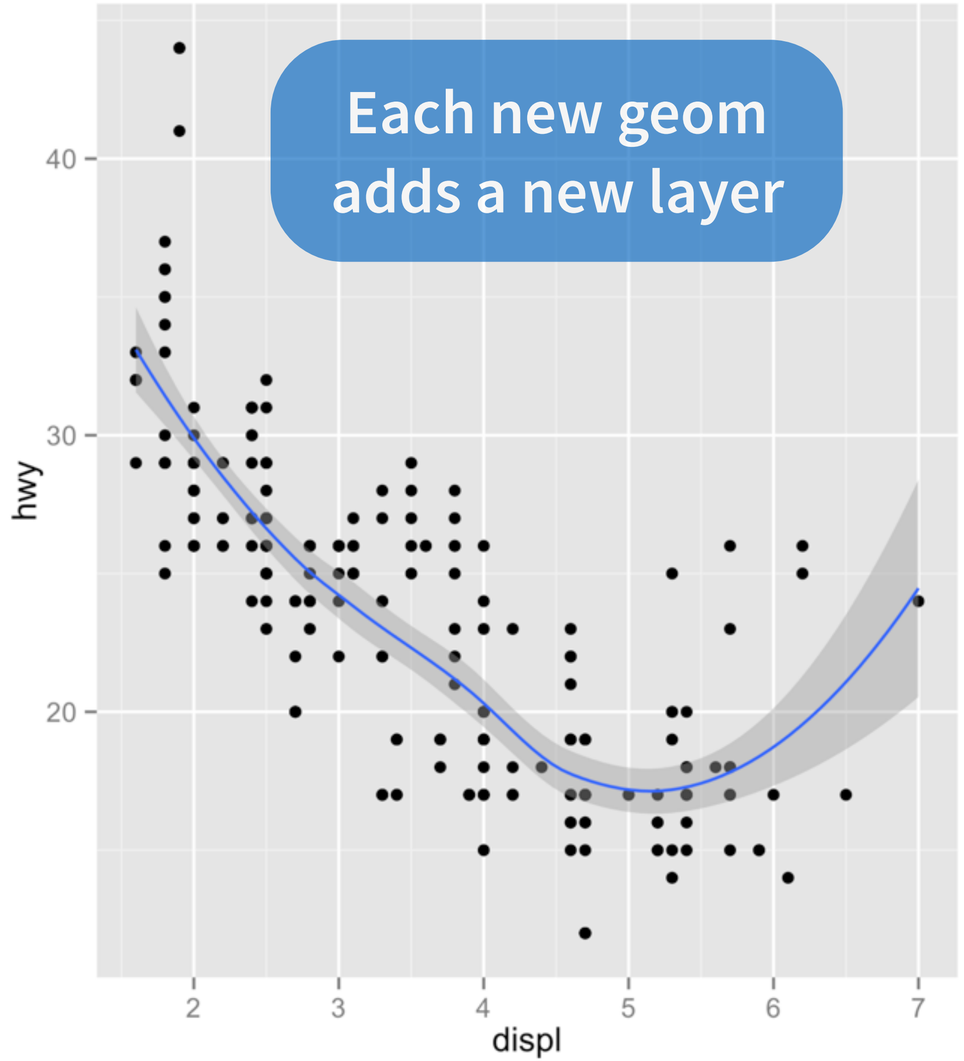
Multiple layers

Your Turn

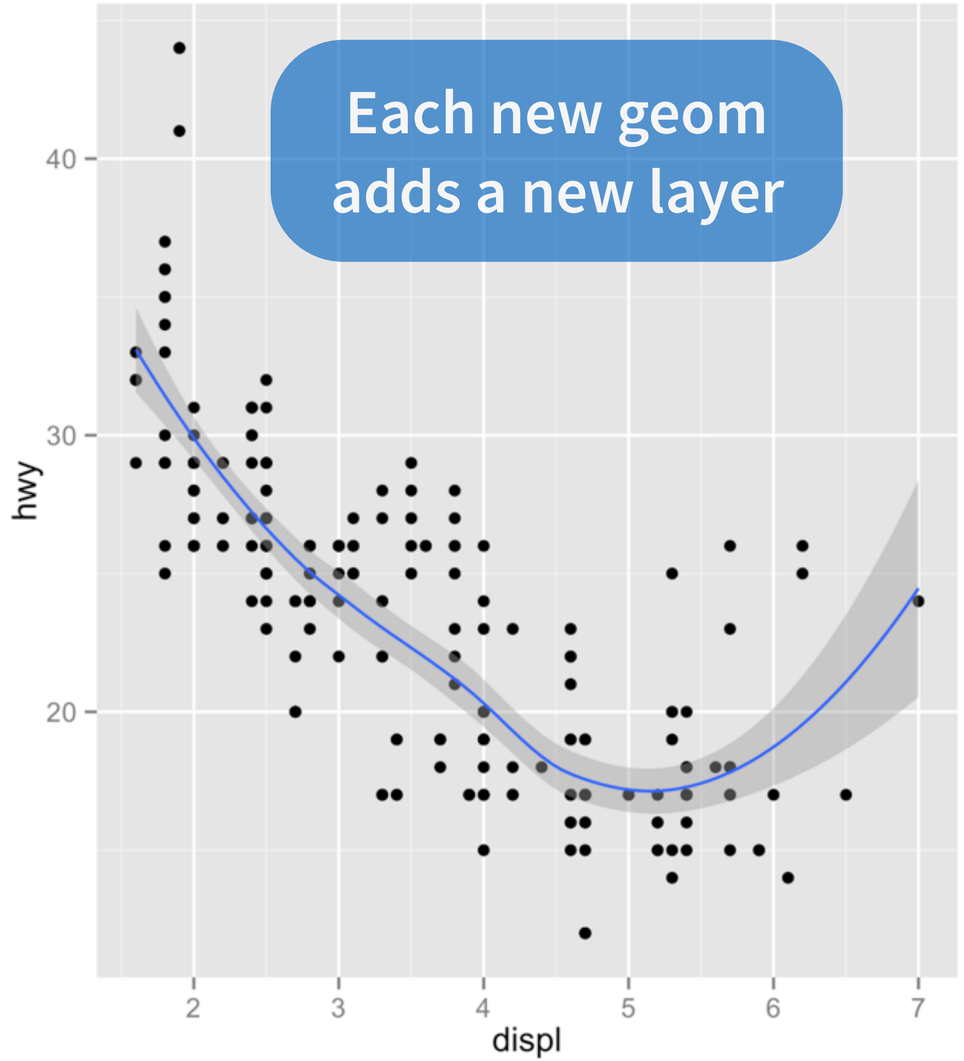
In your group, predict what this code will do.

Then run it.

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

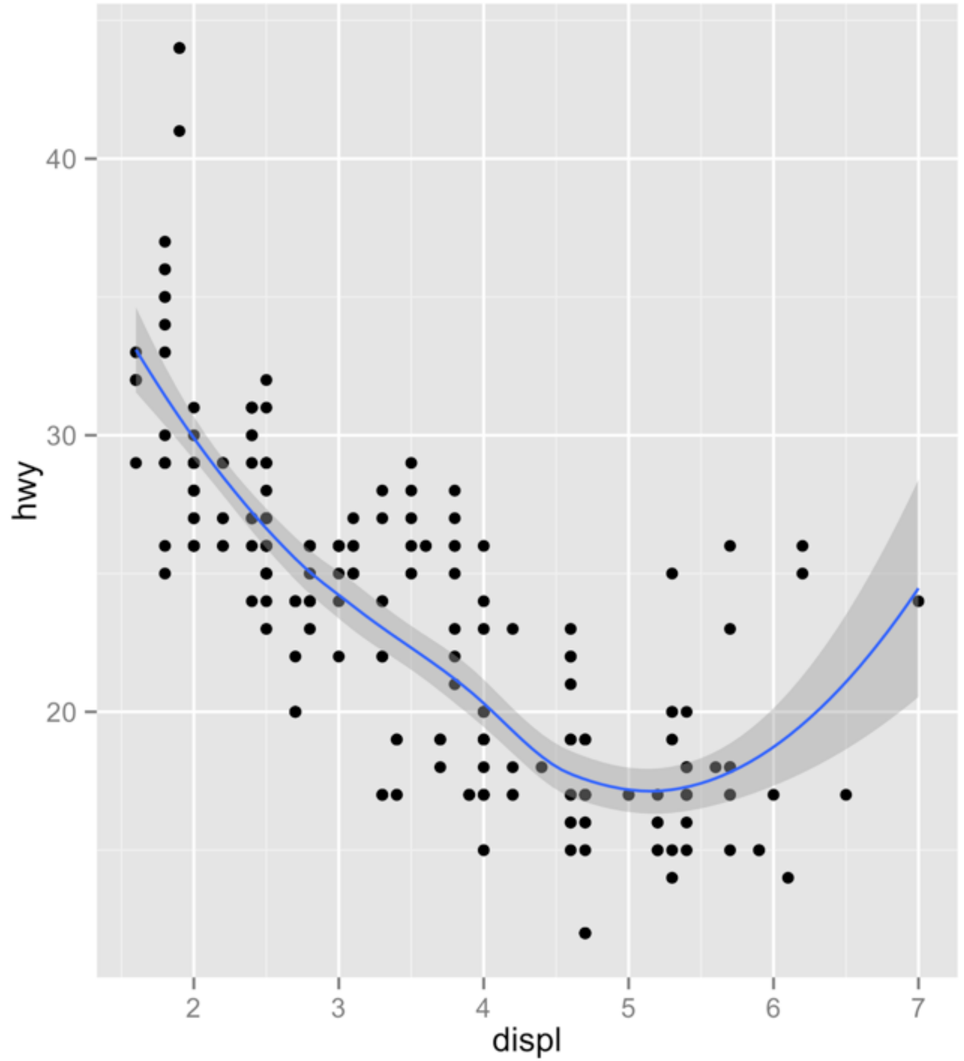



```
ggplot(mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
```




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

global vs. local



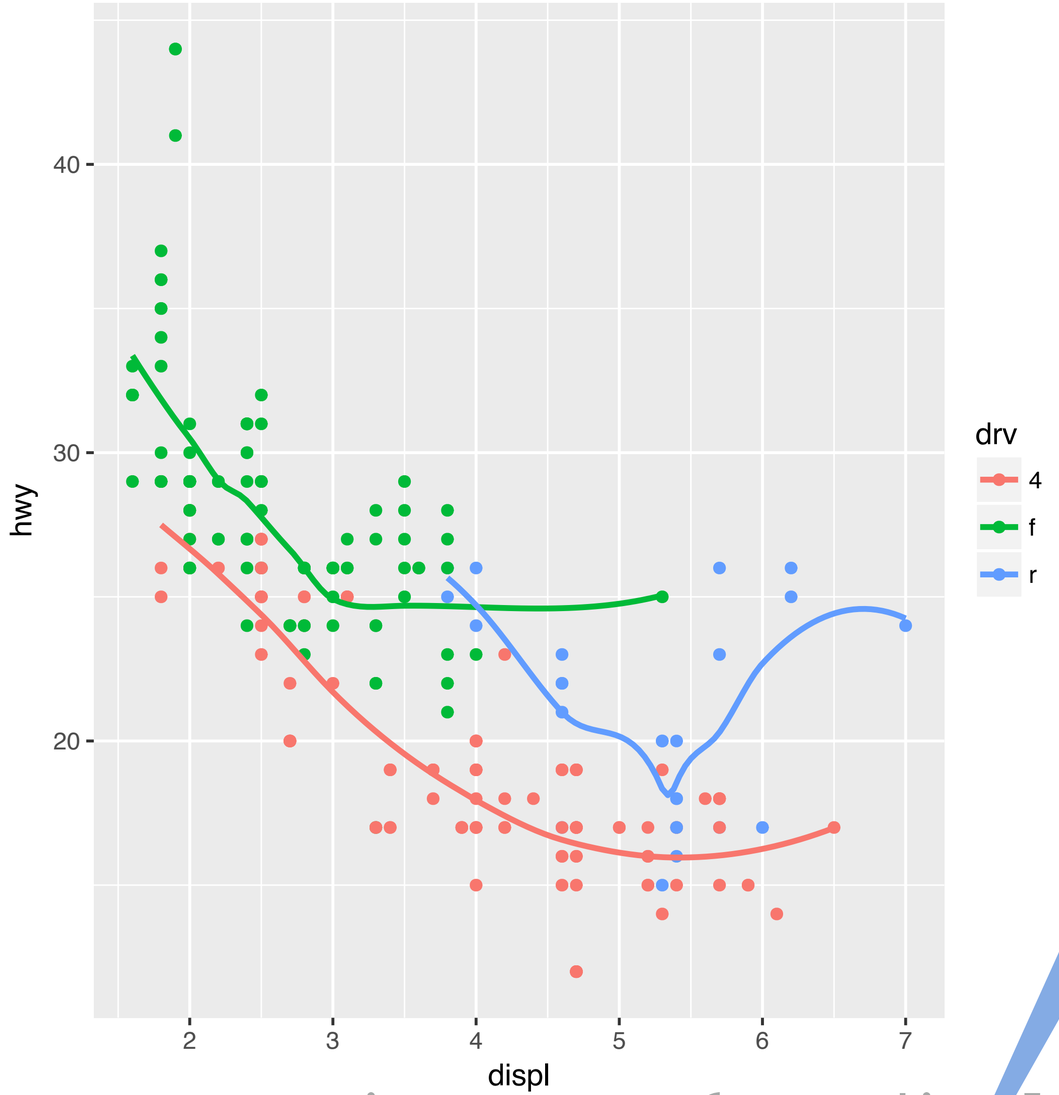
Mappings (and data) that appear in ggplot() will apply globally to every layer

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

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

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

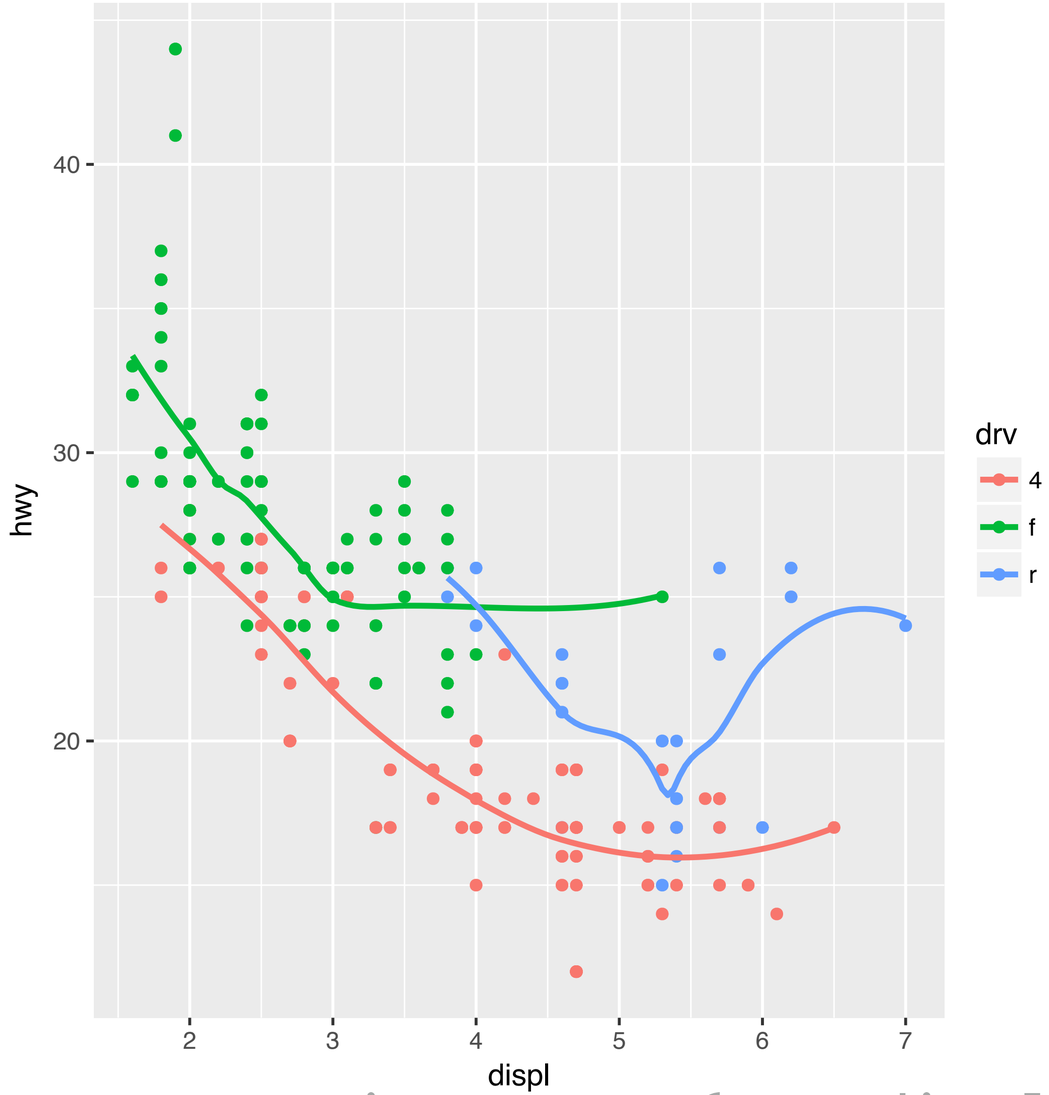
Grouping cases



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

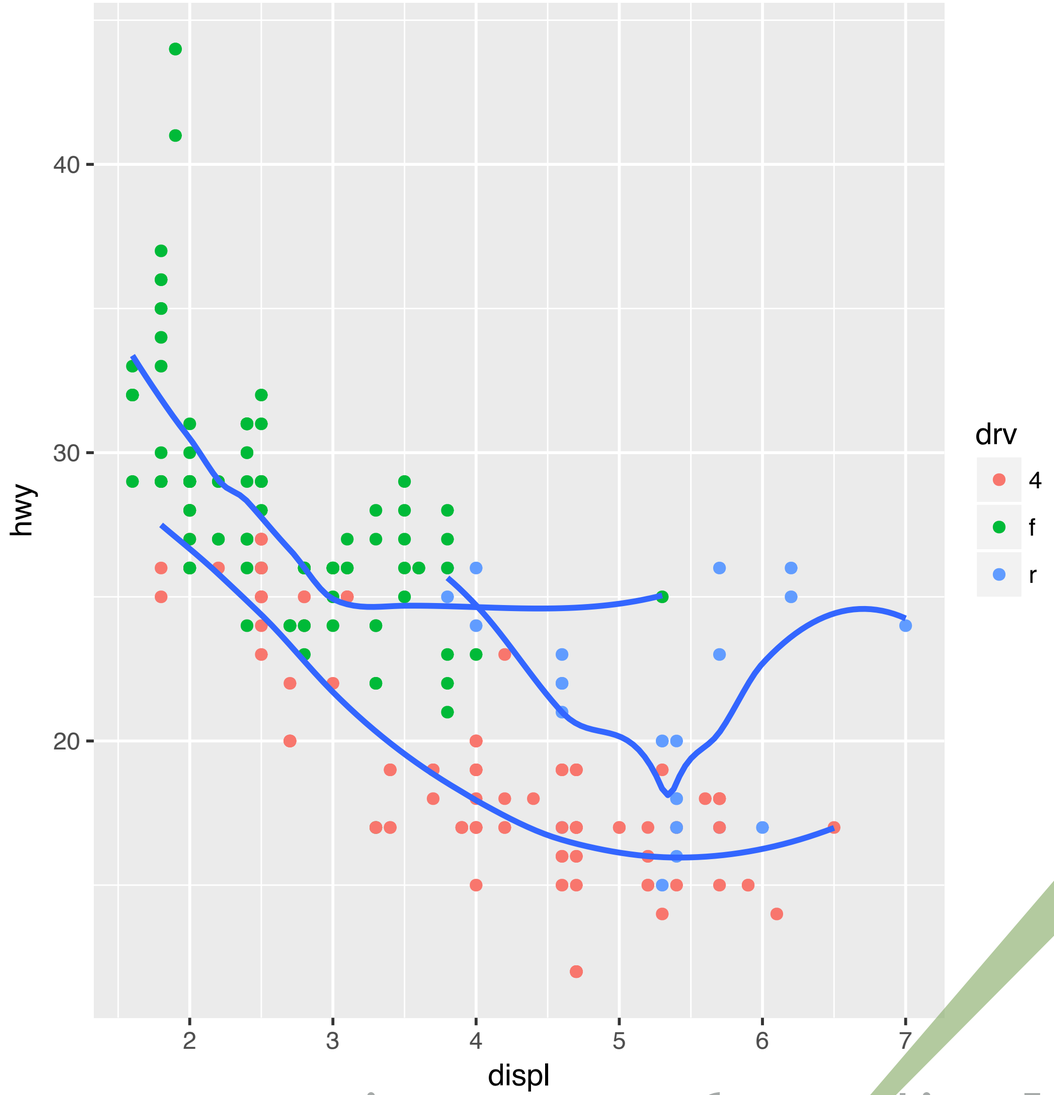
This occurs for other "monolithic" geoms as well.

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




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

Standard error bars

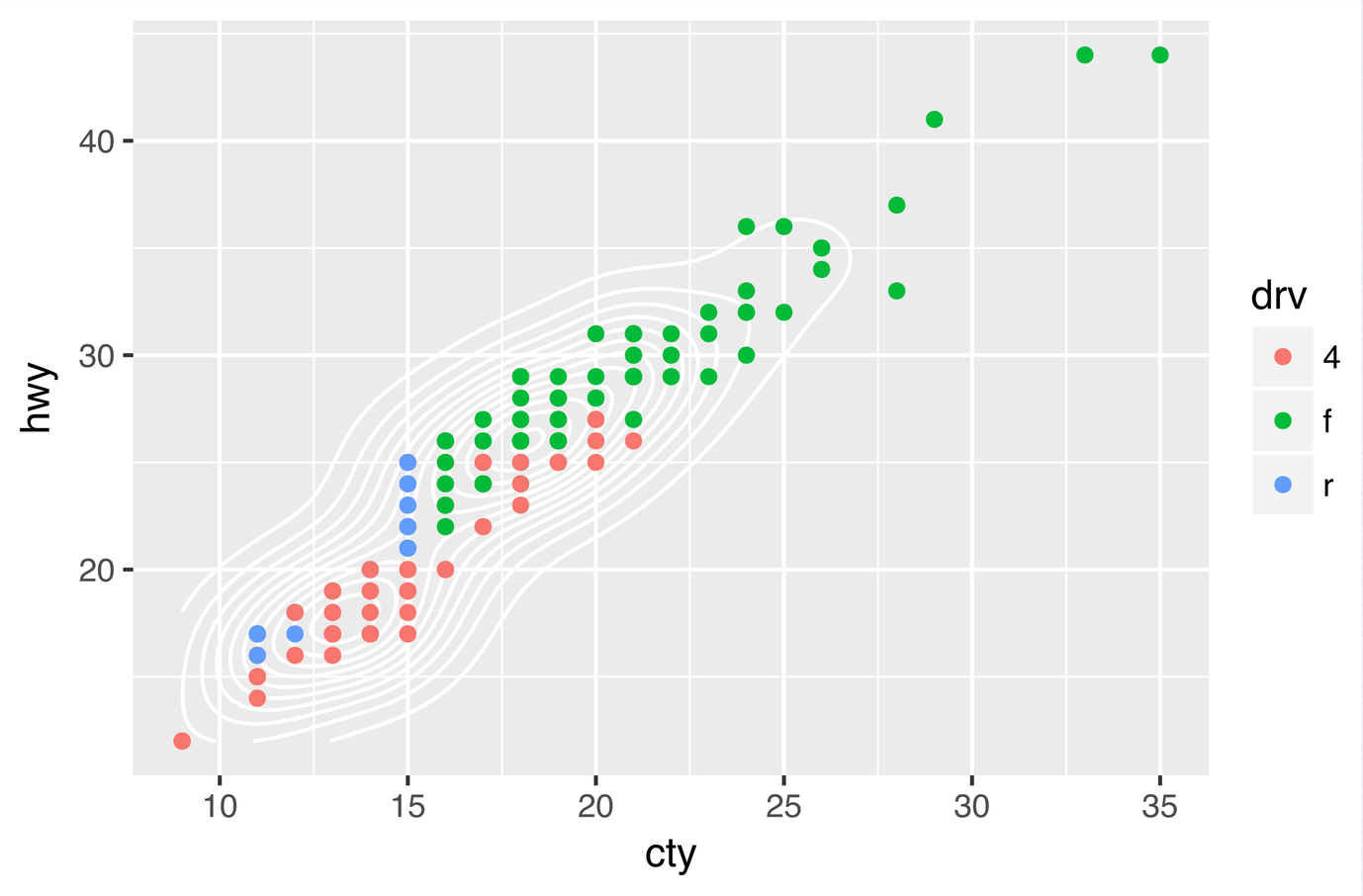


Groups without adding a visible aesthetic (or legend)

```
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)

diamonds

Price and description of 54,000 diamonds

```
View(diamonds)
```

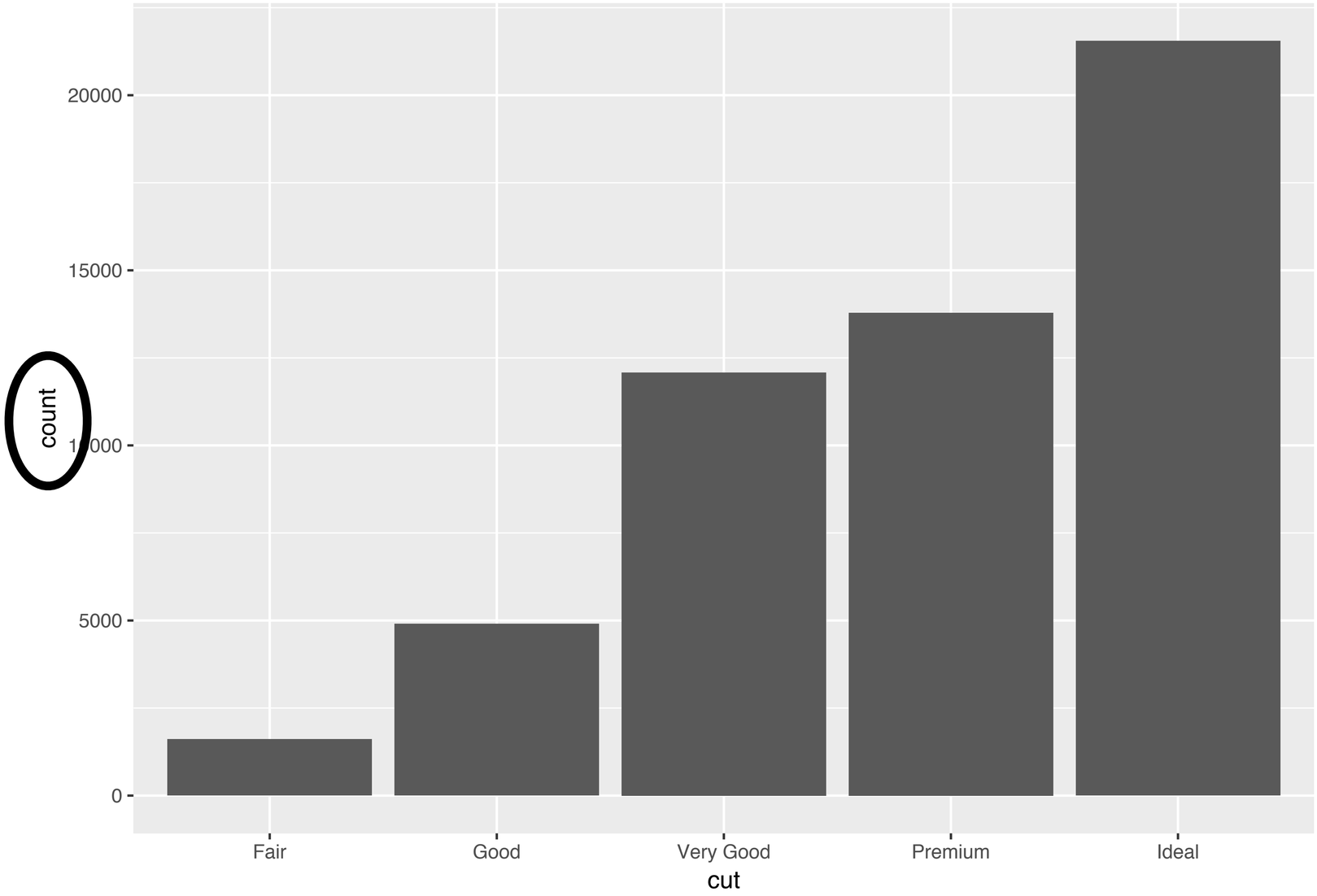
Capital "V"

Your Turn

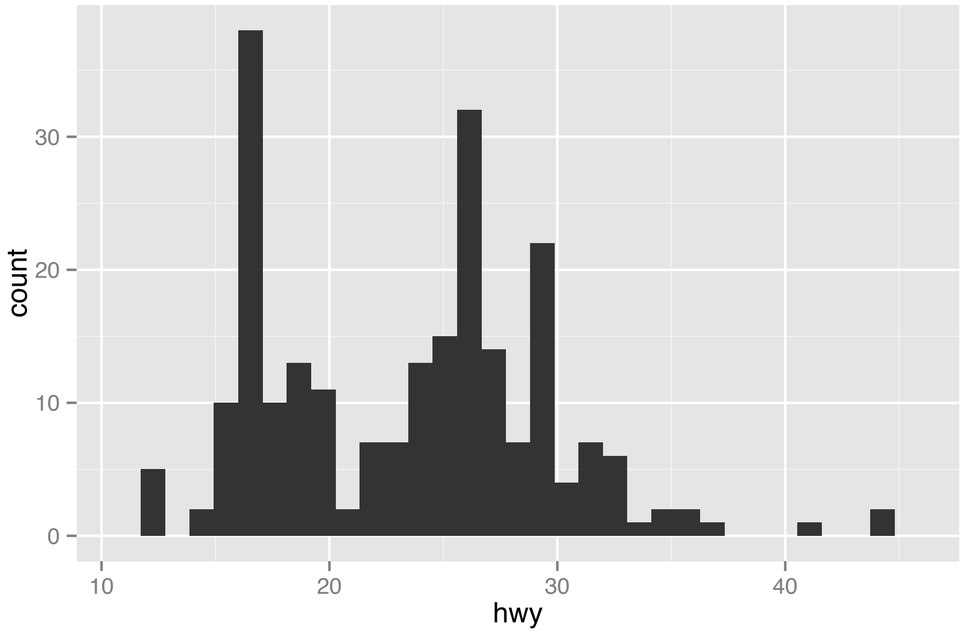
If I run this code, what will ggplot2 put on the y axis?

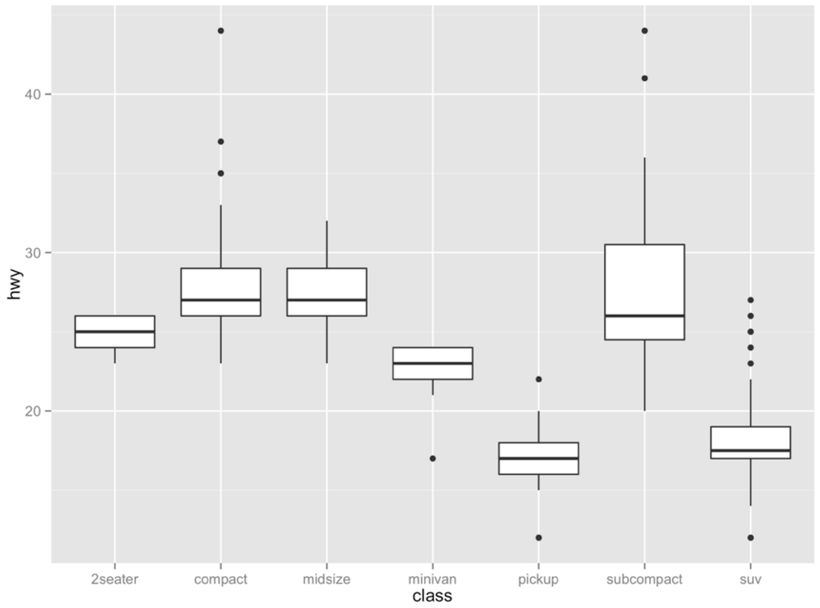
Try it and see.

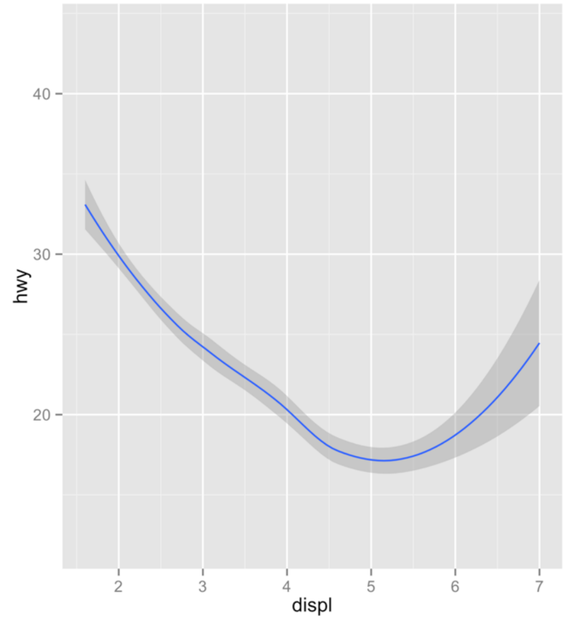
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut))
```




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


count
30
20
10
0
10
20
30
40
hwy


hwy
40
30
20
2seater
compact
midsize
minivan
pickup
subcompact
suv
class


hwy
40
30
20
2
3
4
5
6
7
displ


ggplot2

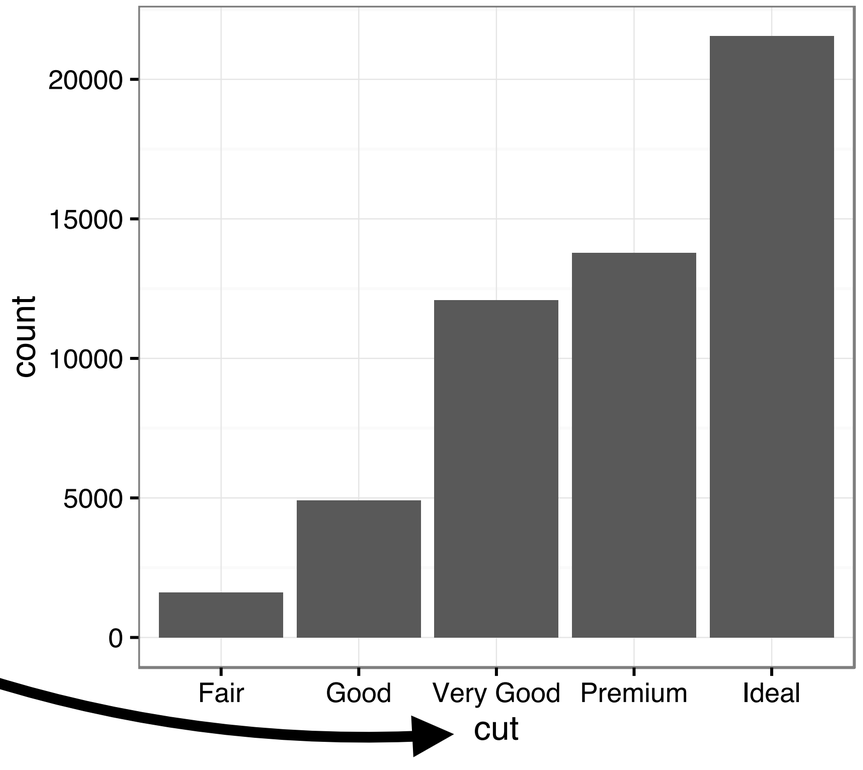
stats

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.

1D distributions

c + **stat_bin(**binwidth = 1, origin = 10**)**
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..

2D distributions

e + **stat_bin_2d(**bins = 30, drop = T**)**
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"**)**

3 Variables

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..

Comparisons

f + **stat_boxplot(**coef = 1.5**)**
x, y | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..
f + **stat_ydensity(**kernel = "gaussian", scale = "area"**)**
x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

Functions

e + **stat_ecdf(**n = 40**)** x, y | ..x.., ..y..
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..

General Purpose

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()**

ggplot2

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>
```

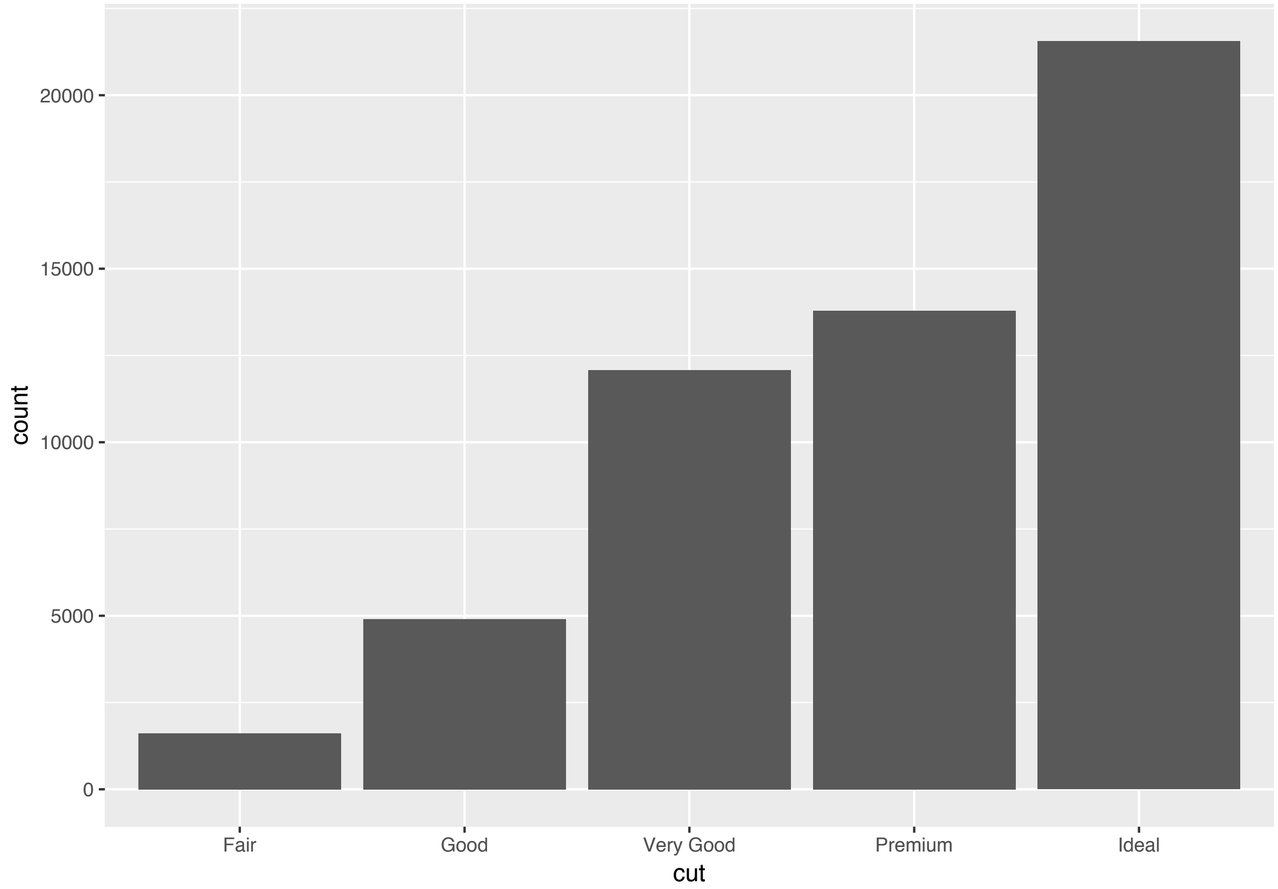
ggplot2 supplies useful defaults for everything except DATA, GEOM_FUNCTION, and MAPPINGS

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>
```

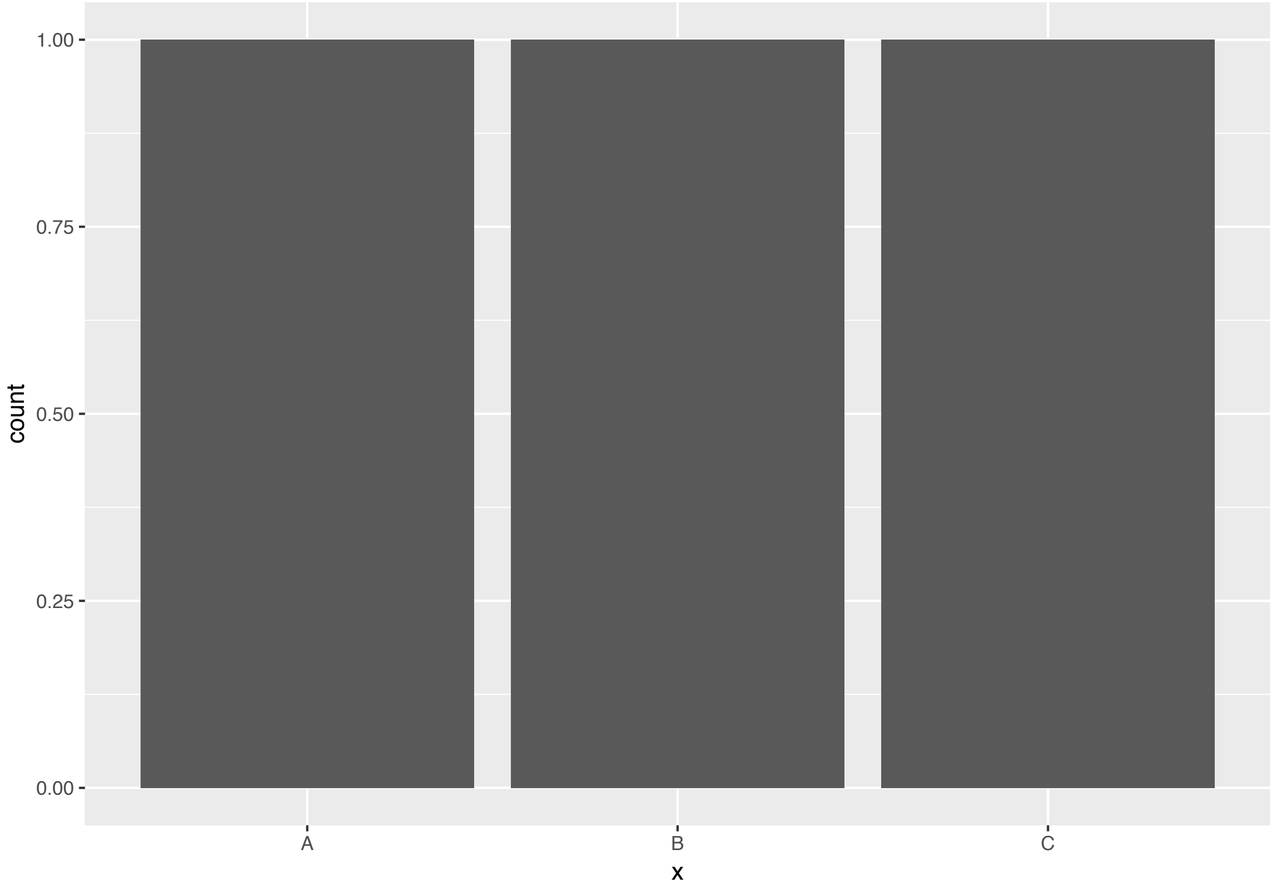
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut), stat = "count")
```




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

df

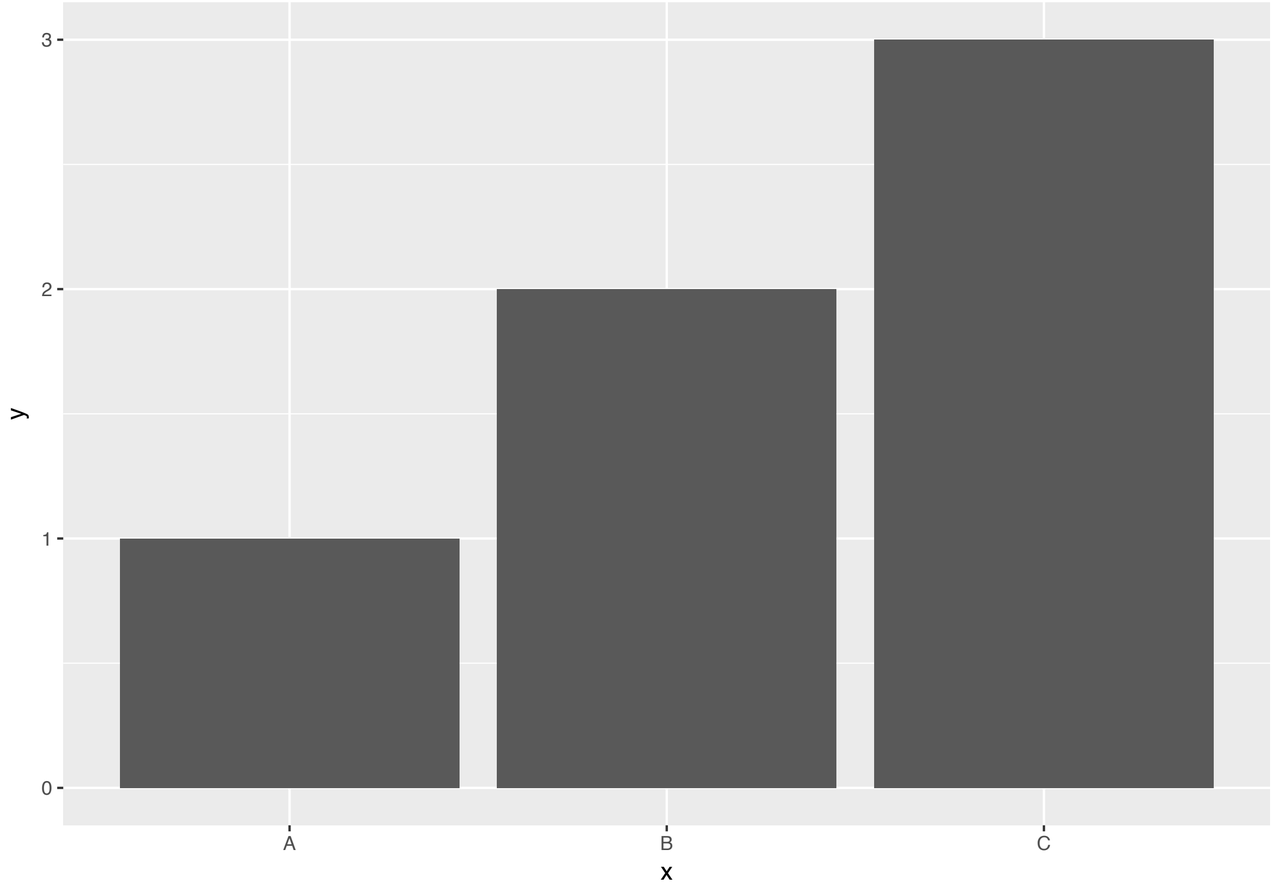
x	y
A	1
B	2
C	3




```
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

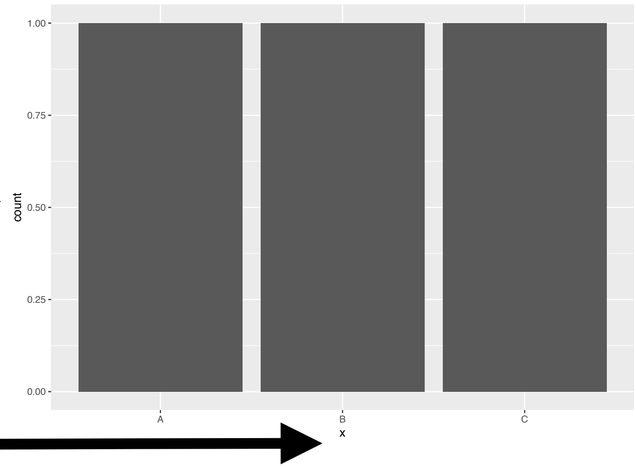



```
df <- data.frame(x = c("A", "B", "C"), y = 1:3)
ggplot(data = df) +
  geom_bar(mapping = aes(x = x, y = y), stat = "identity")
```

x	y
A	1
B	2
C	3

stat_count()

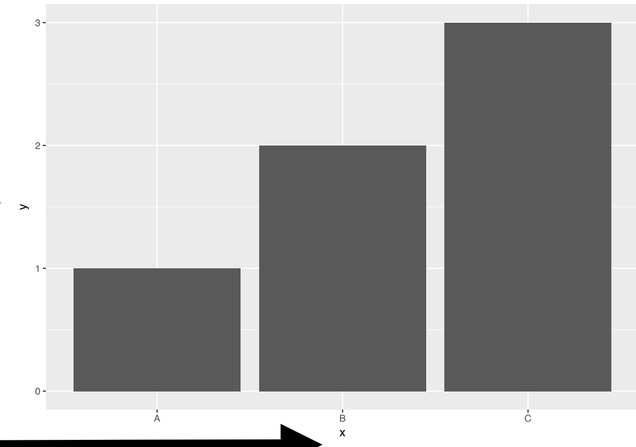
x	count
A	1
B	1
C	1



x	y
A	1
B	2
C	3

stat_identity()

x	y
A	1
B	2
C	3



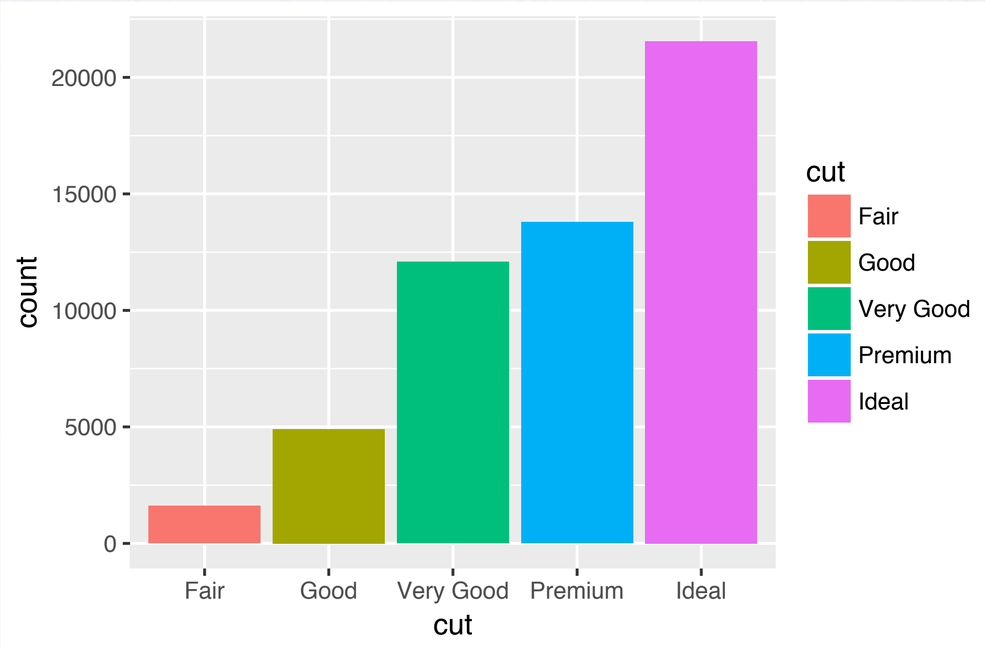
ggplot2

Position adjustments

Challenge

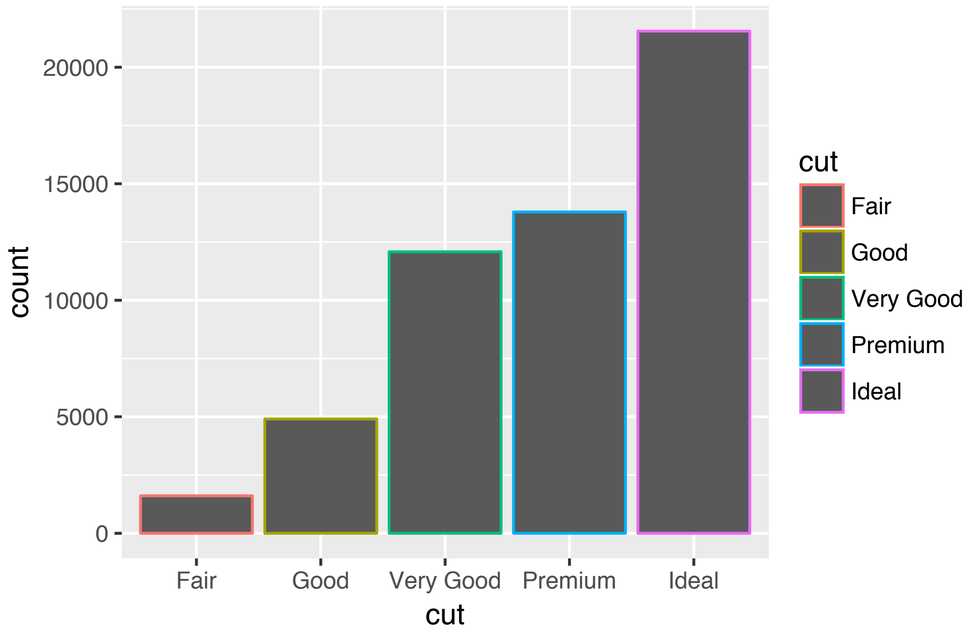
Can you alter this code to make this plot?

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

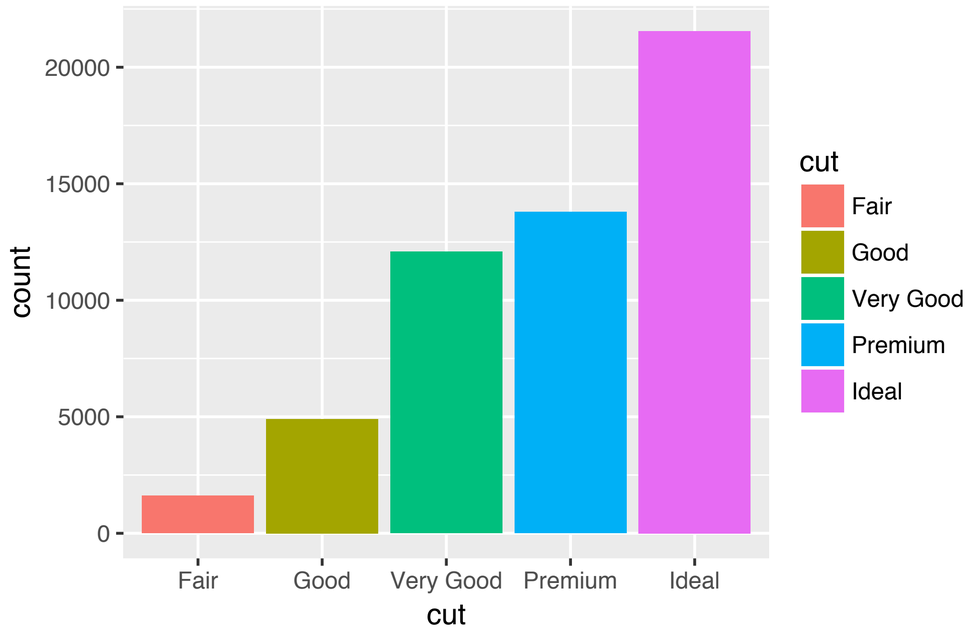


fill

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))
```

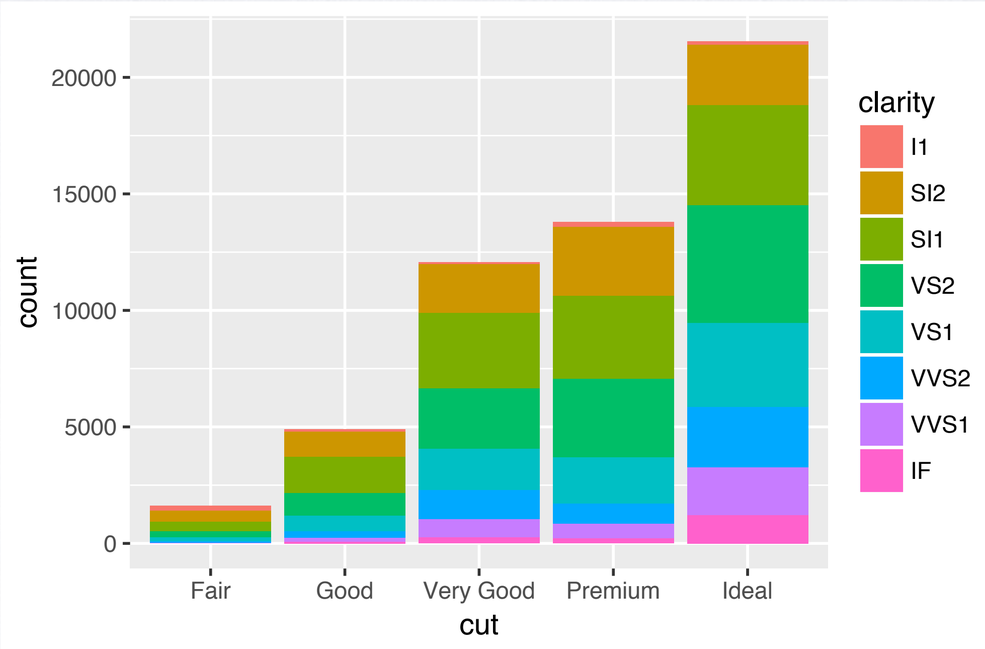



```
ggplot(diamonds, mapping = aes(x = cut)) +
  geom_bar(mapping = aes(fill = cut))
```

Challenge

Can you alter this code to make this plot?

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

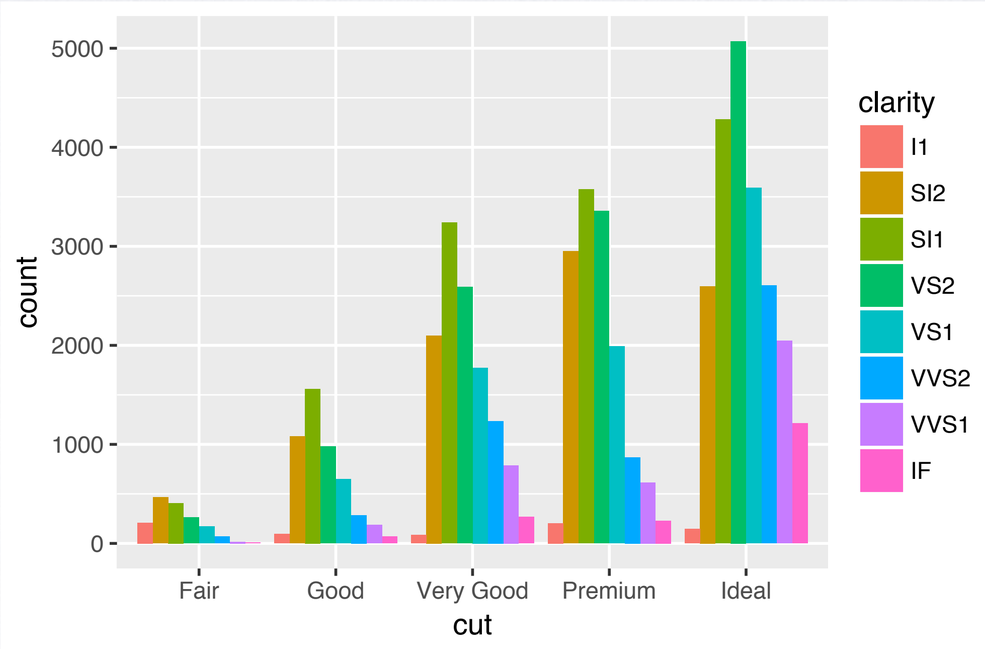


```
ggplot(diamonds, mapping = aes(x = cut)) +
  geom_bar(mapping = aes(fill = clarity))
```

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")
```

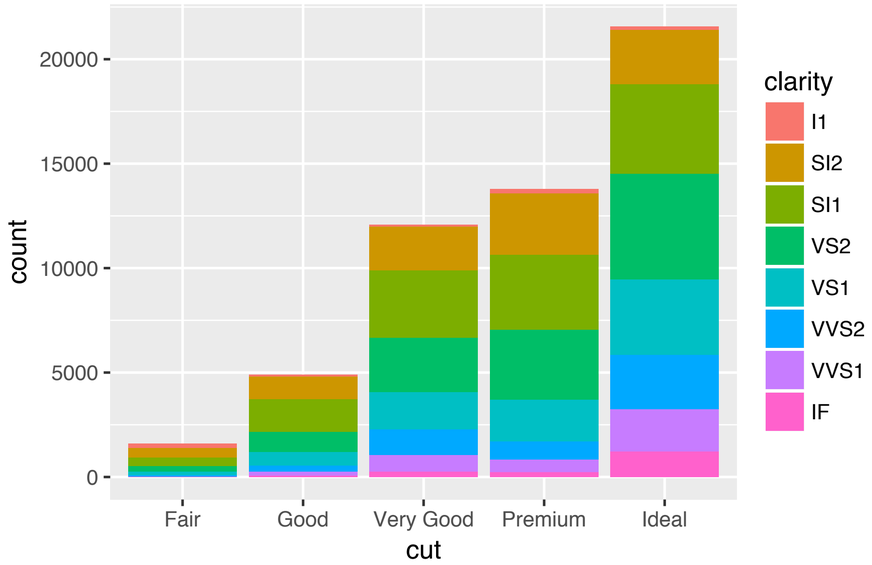
Set the adjustment method with the position argument

Your turn

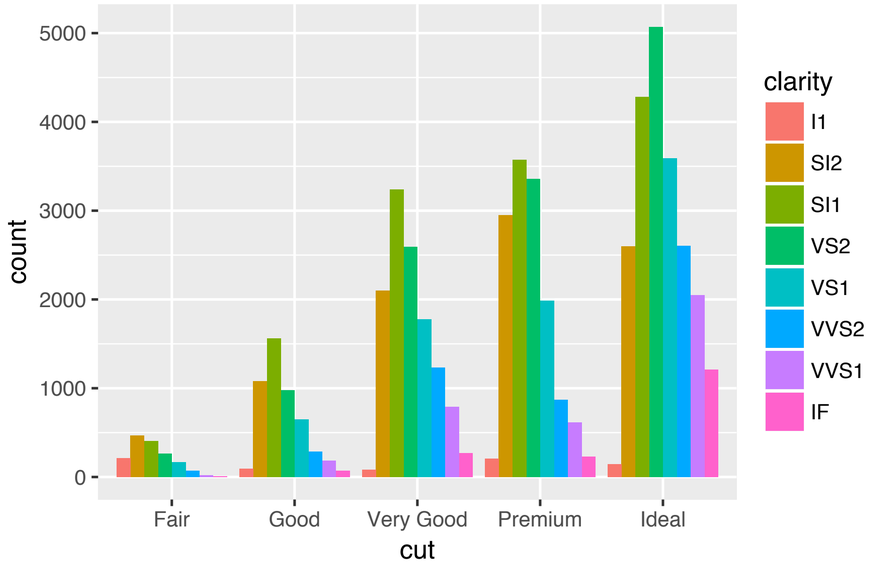
What do each of these adjustments do?
(run the code, interpret, convince your group)

```
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

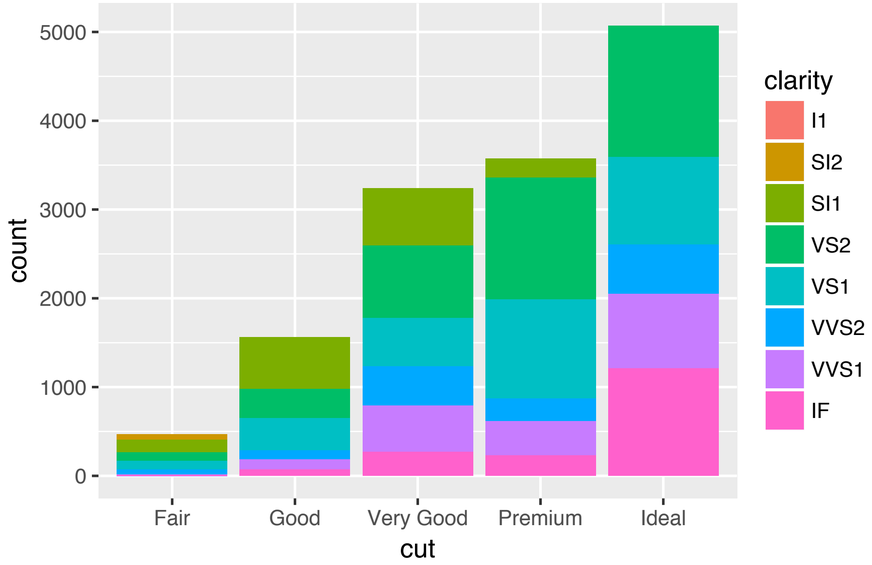


dodge



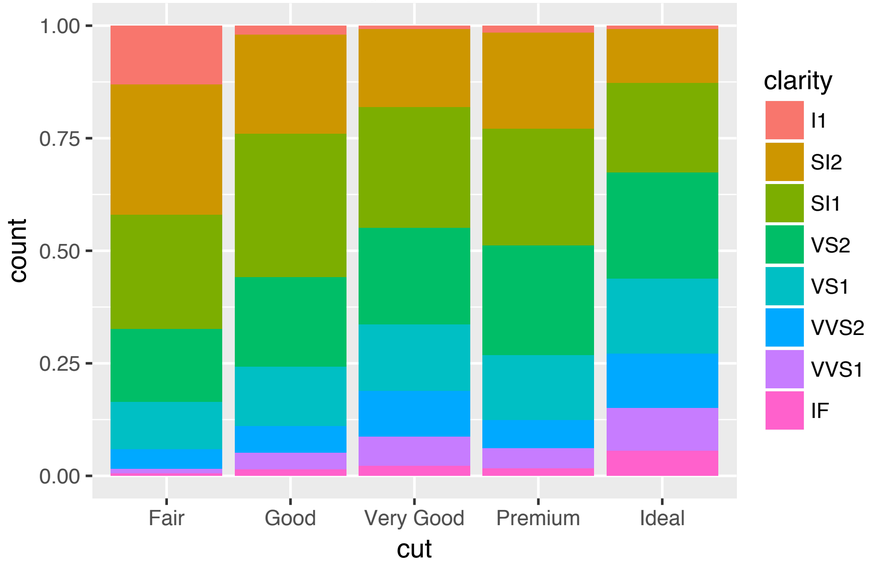
identity

(overlaps, last on top)



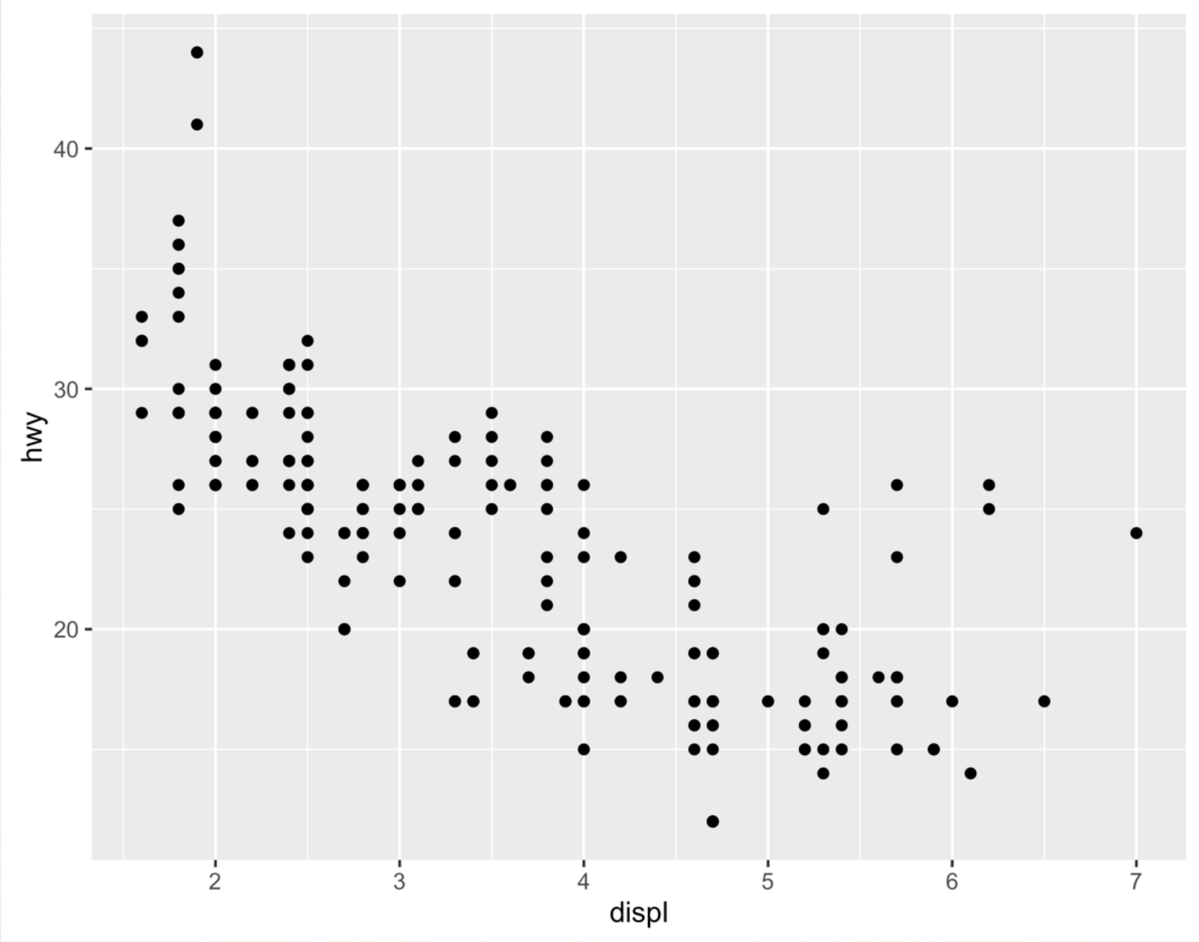
fill

(displays proportions)

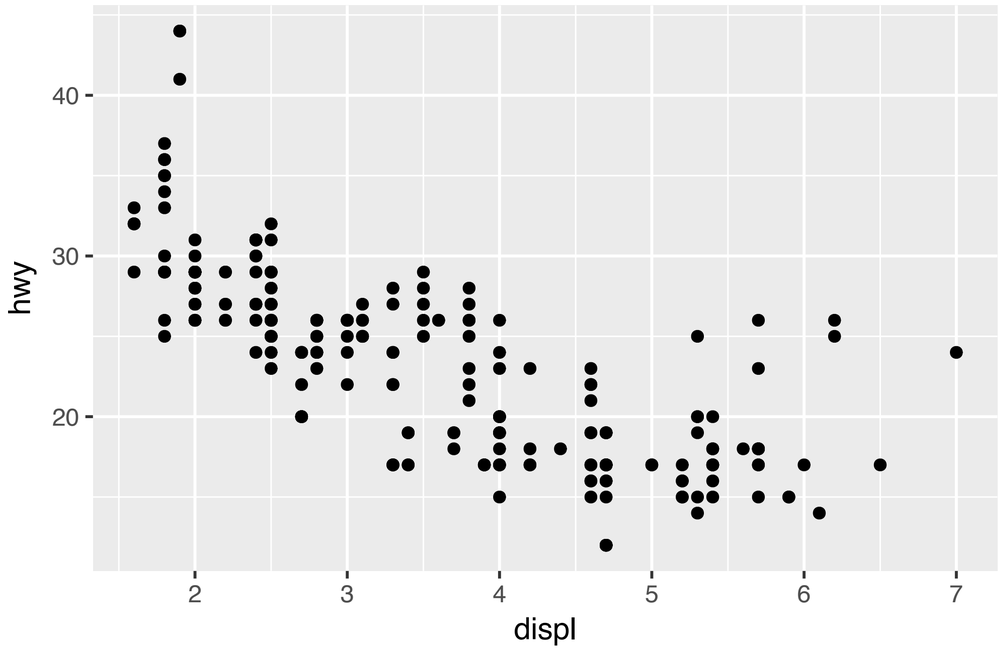


Quiz

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



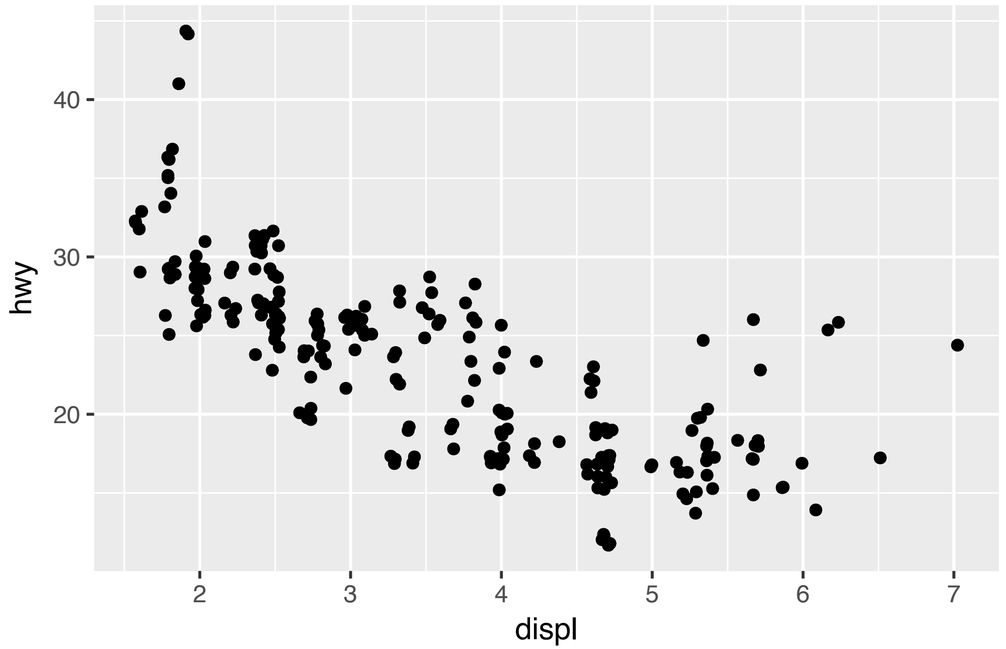
```
p <- ggplot(mpg, aes(displ, hwy))
```




```
p + geom_point()
```

jitter

(moves each point by a small, random amount)




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

A ggplot2 template

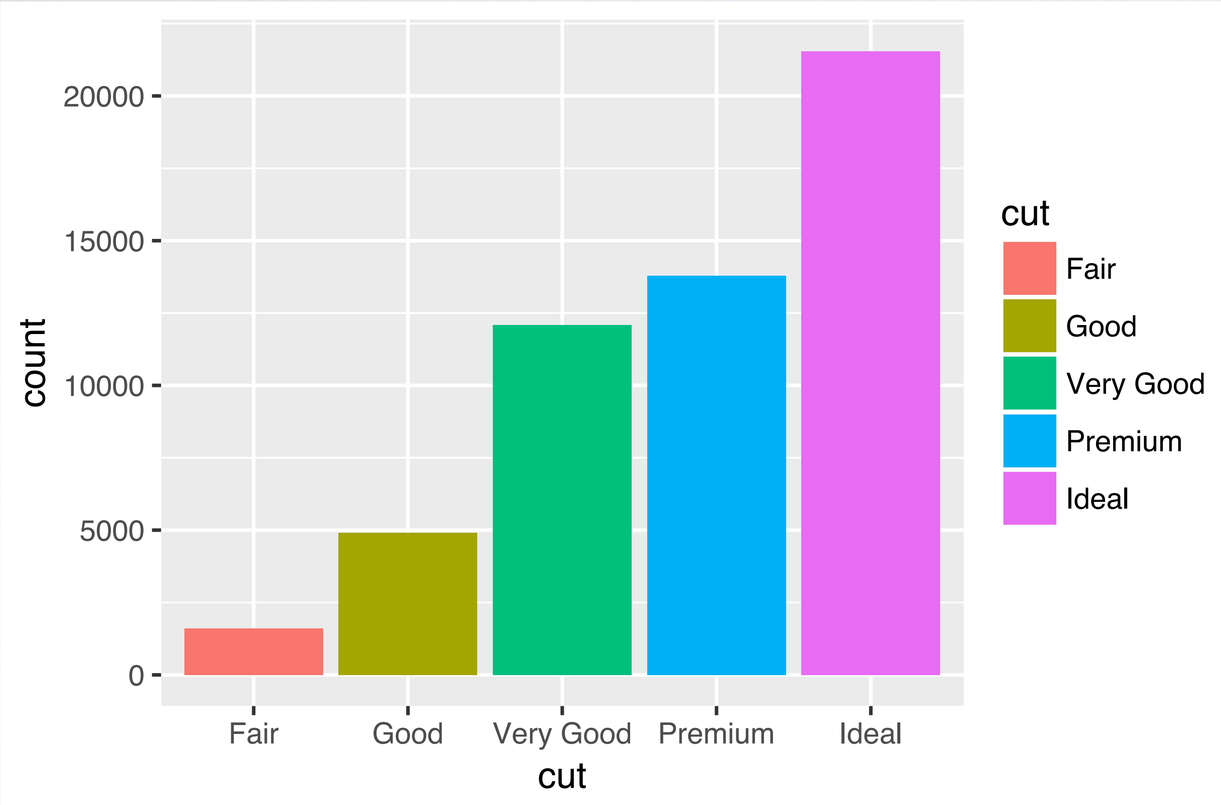
Make any plot by filling in the parameters of this template

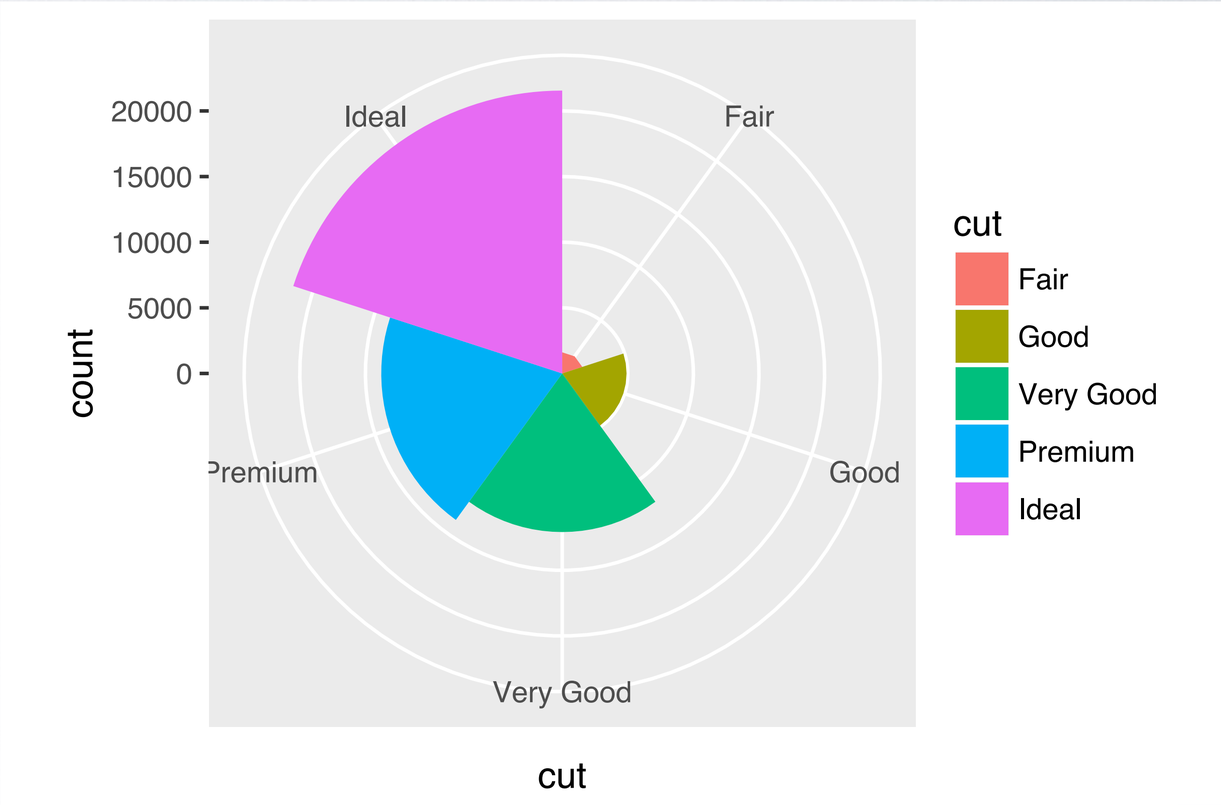
```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
                  stat = <STAT>,
                  position = <POSITION>) +
  <FACET_FUNCTION>
```

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

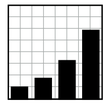
```
ggplot(diamonds) +
  geom_bar(mapping = aes(x = cut, fill = cut)) +
  coord_polar()
```

change the coordinate system by adding a coord_ function

coord_ functions

Adds a coordinate system to a graph. ggplot2 adds coord_cartesian() by default.

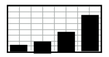
r <- d + geom_bar()



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()

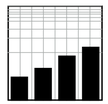
xlim, ylim

Flipped Cartesian coordinates

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

theta, start, direction

Polar coordinates



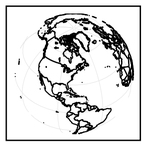
r + coord_trans(ytrans = "sqrt"**)**

xtrans, ytrans, limx, limy

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

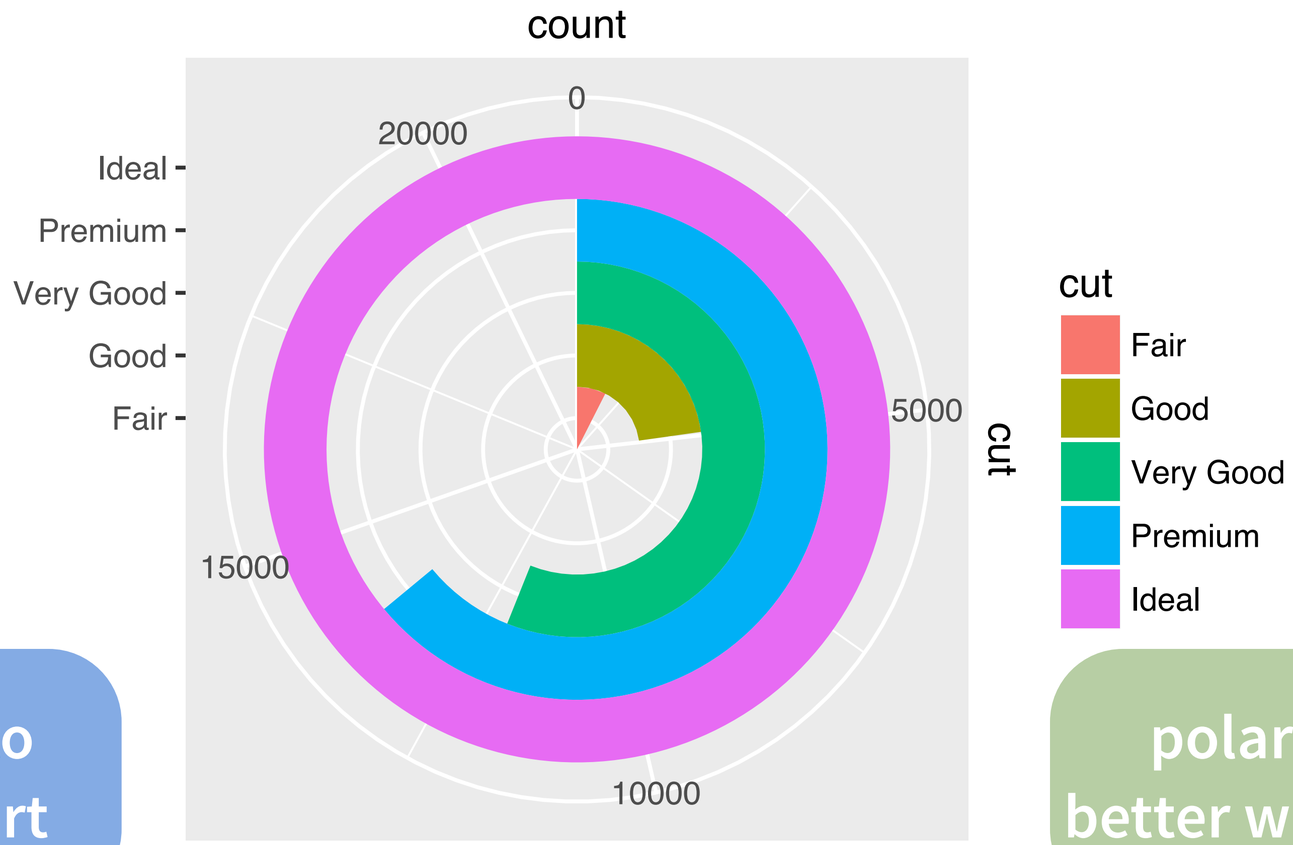
π + coord_quickmap()

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

polar charts look better when bars touch

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

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>
```

labels

labs()

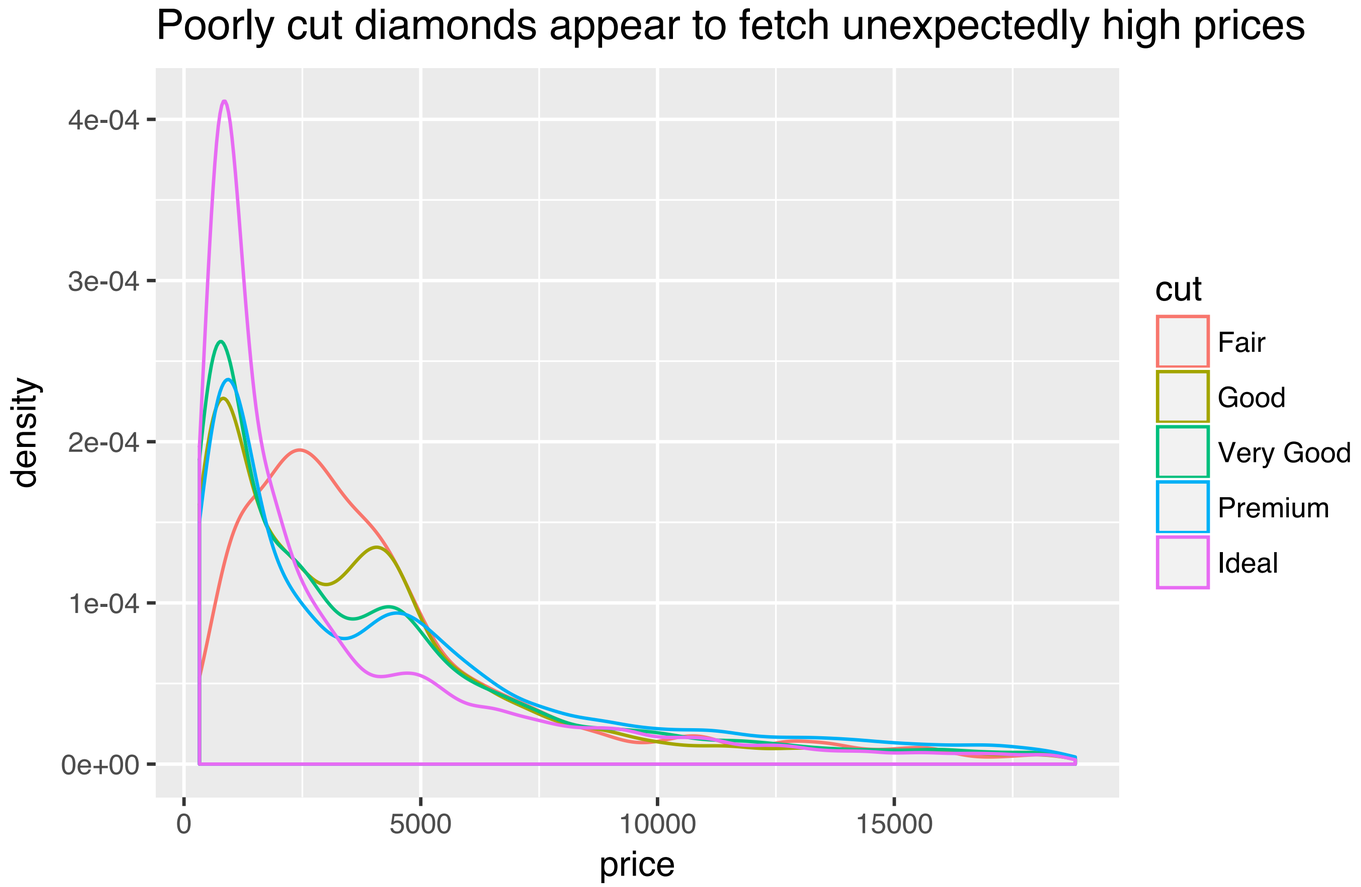
Add titles, subtitles, and more

```
plot + labs(…)
```

plot to add labels to

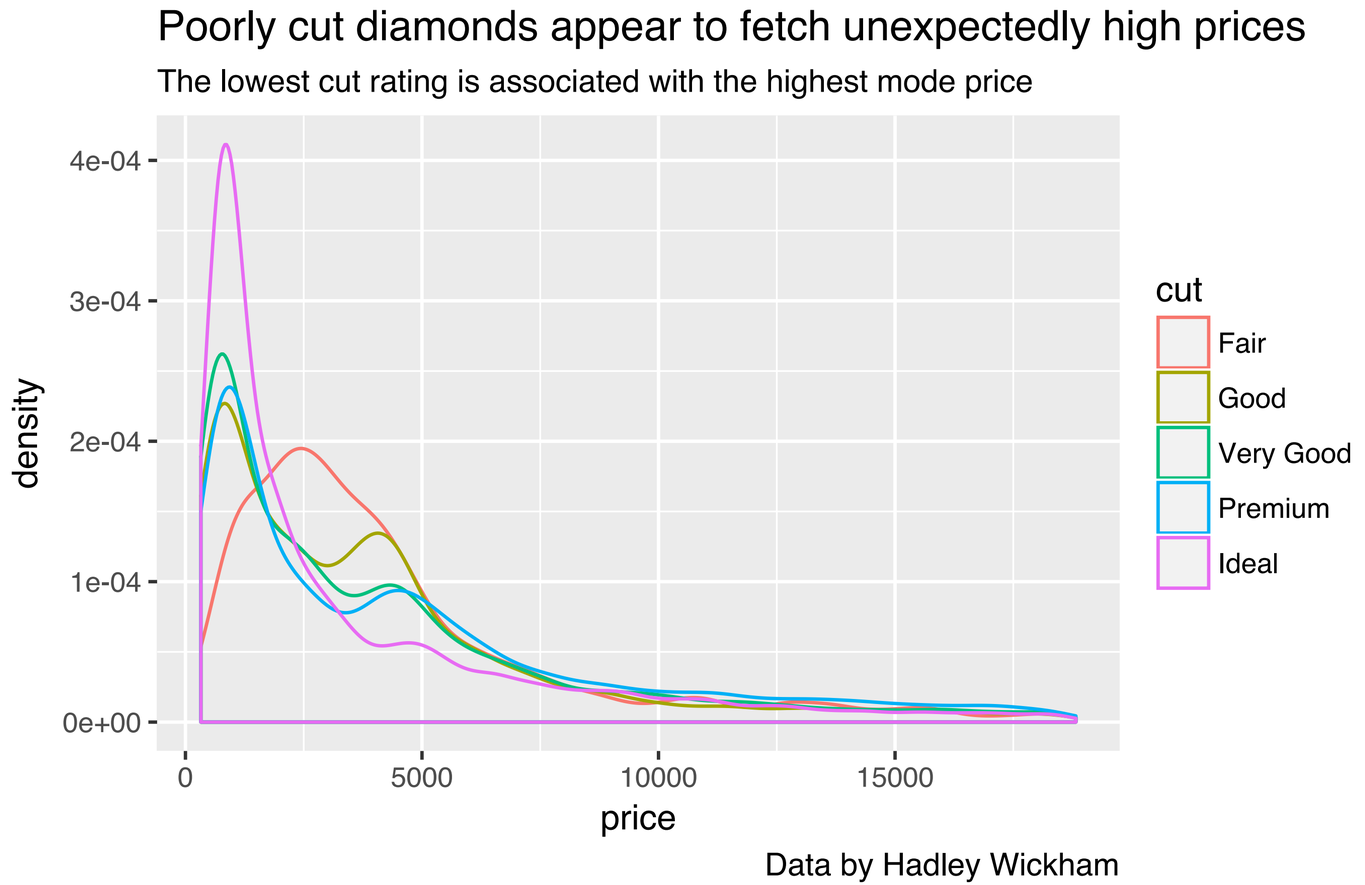
Character strings to add by type

titles



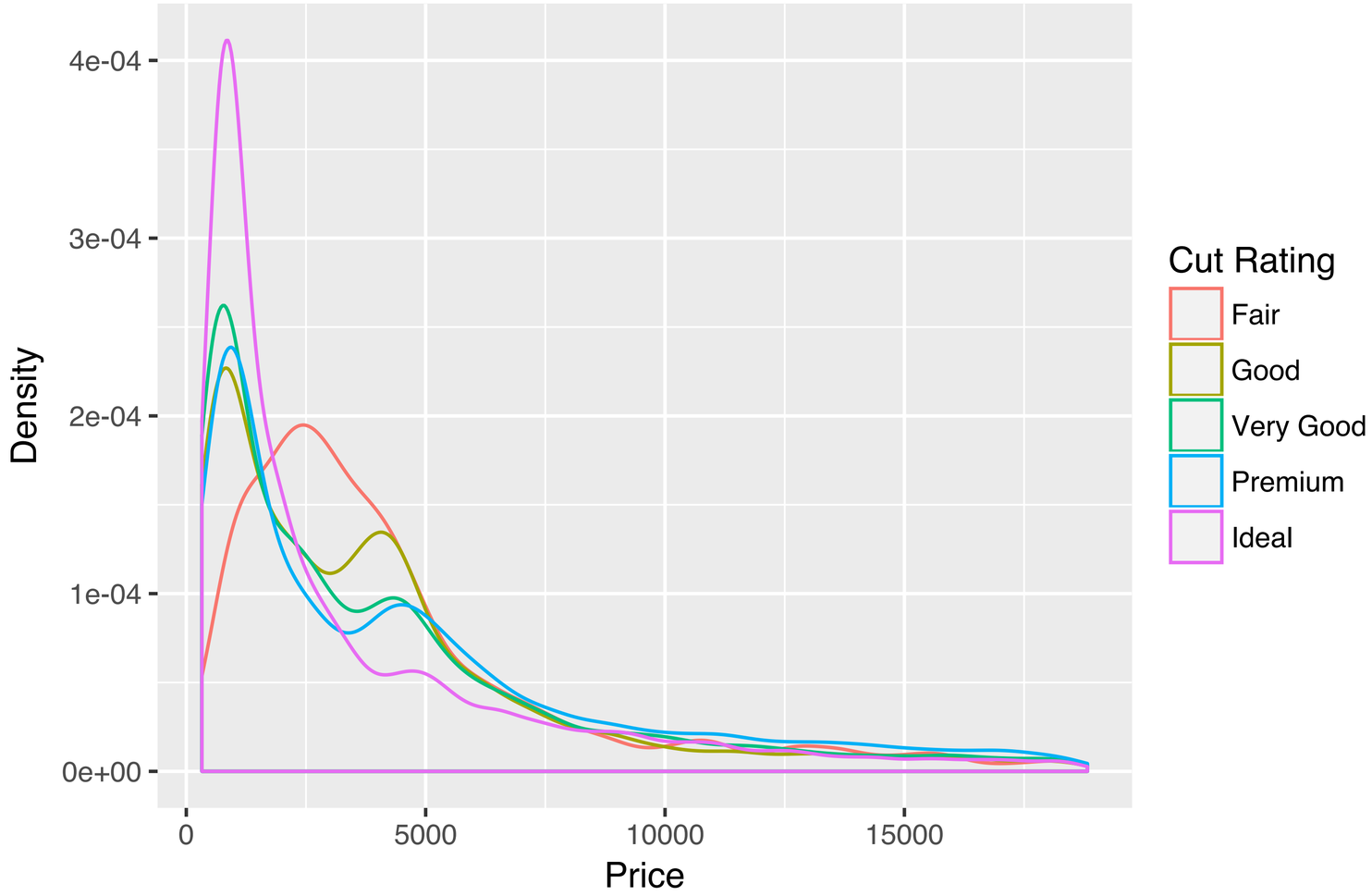

```
p1 + labs(title = "Poorly cut diamonds appear to fetch
unexpectedly high prices")
```

subtitles and captions




```
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



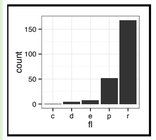

```
p1 + labs(x = "Price", y = "Density", color = "Cut Rating")
```

themes

theme functions

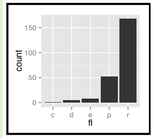
Each supplies a different appearance for non-data elements

Themes



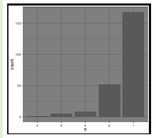
r + theme_bw()

White background with grid lines



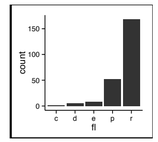
r + theme_gray()

Grey background (default theme)



r + theme_dark()

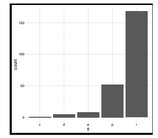
dark for contrast



r + theme_classic()

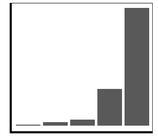
r + theme_light()

r + theme_linedraw()



r + theme_minimal()

Minimal themes



r + theme_void()

Empty theme

theme_*()

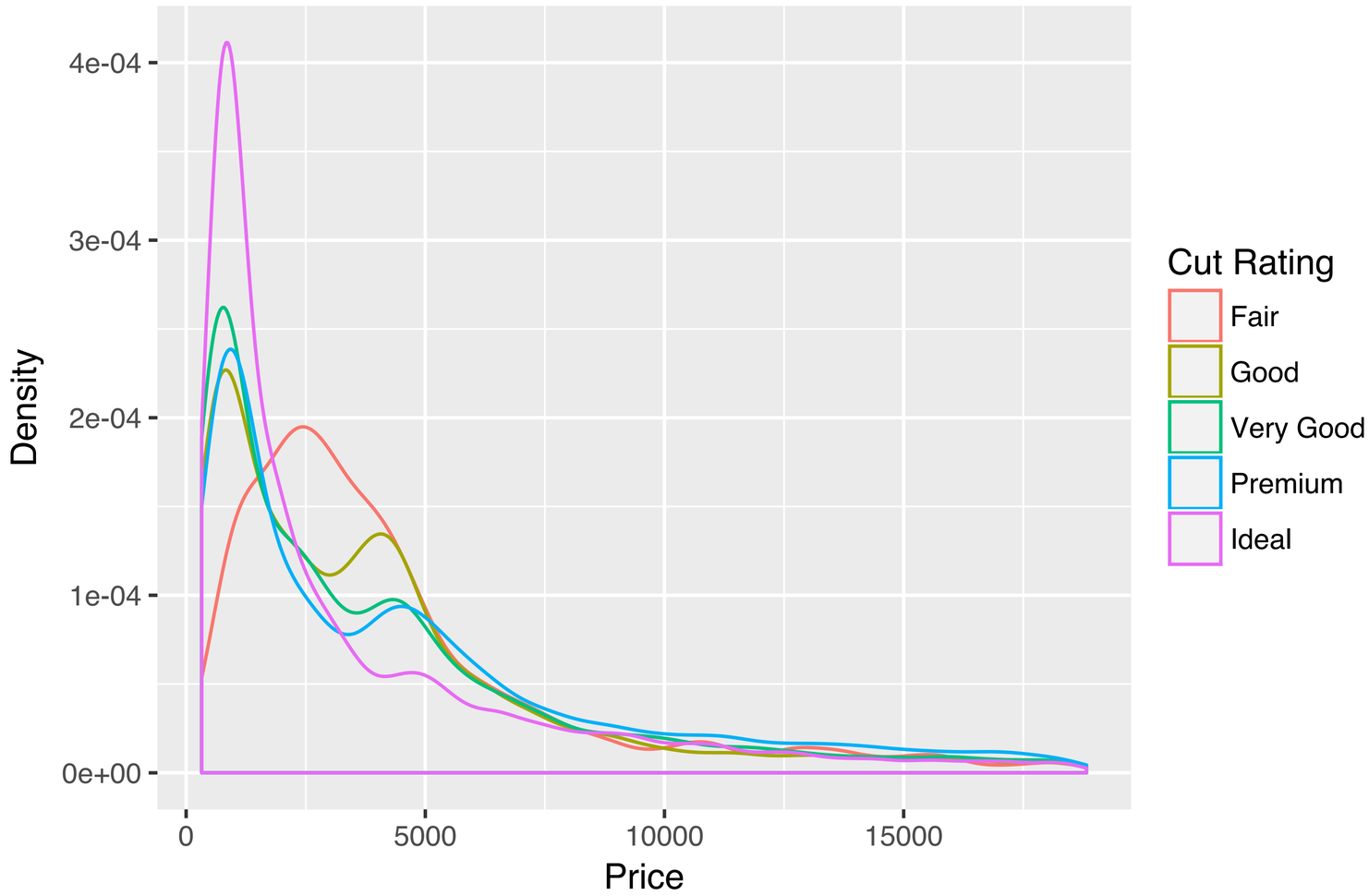
Change appearance of non-data elements

```
plot + theme_bw()
```

plot to add theme to

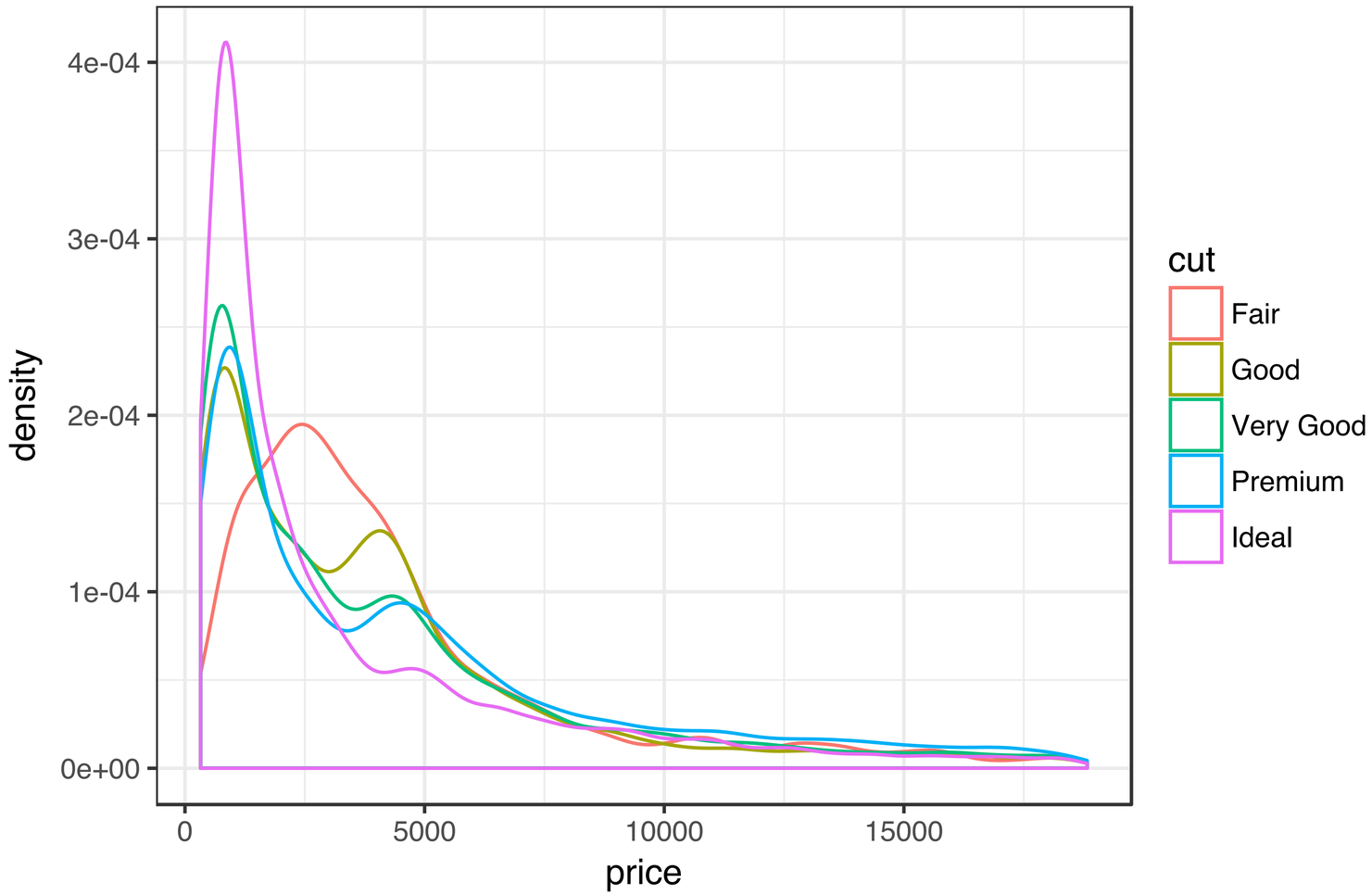
function with prefix theme_

theme_grey()



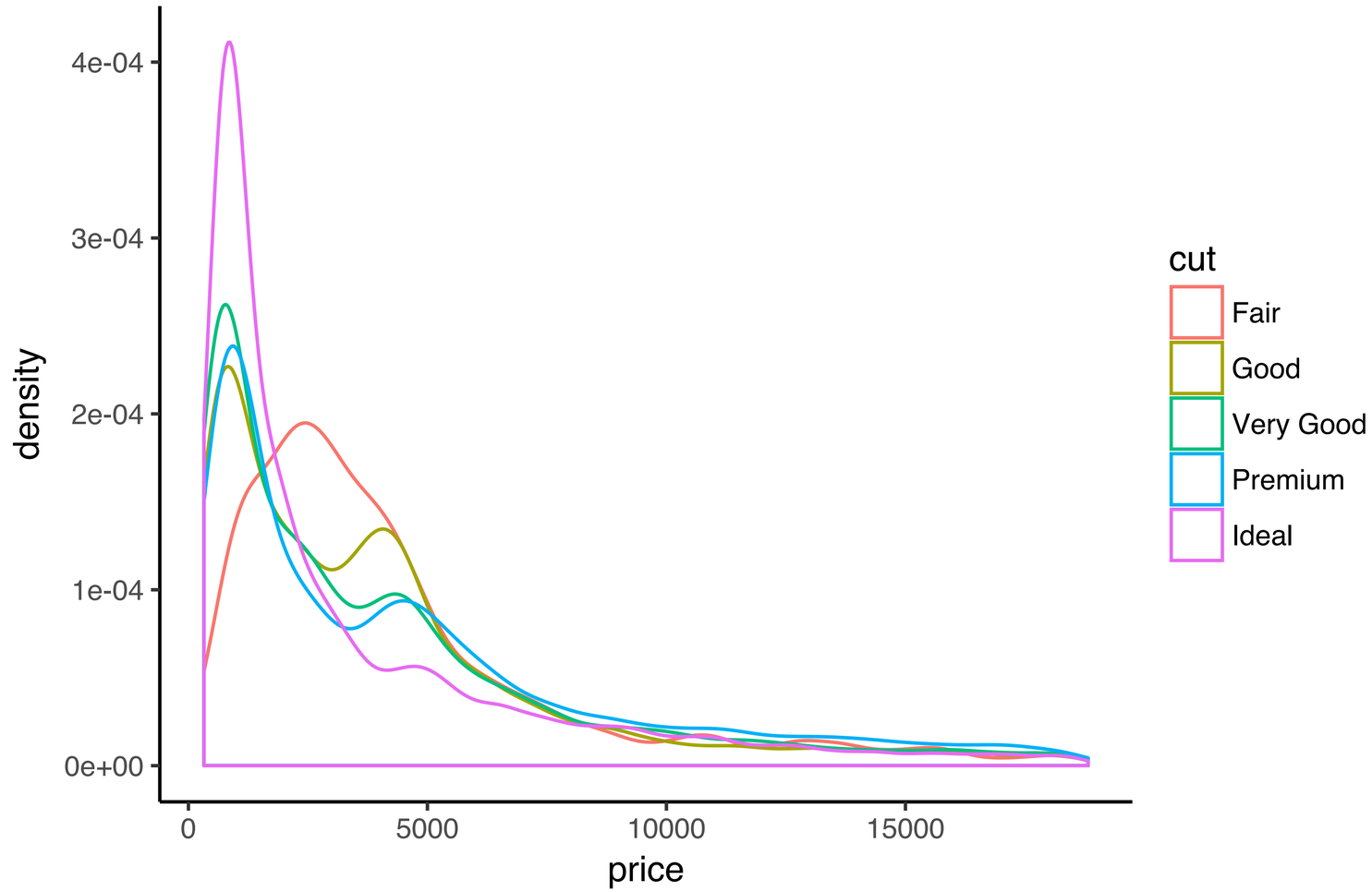

```
p1 + theme_grey()
```

theme_bw()



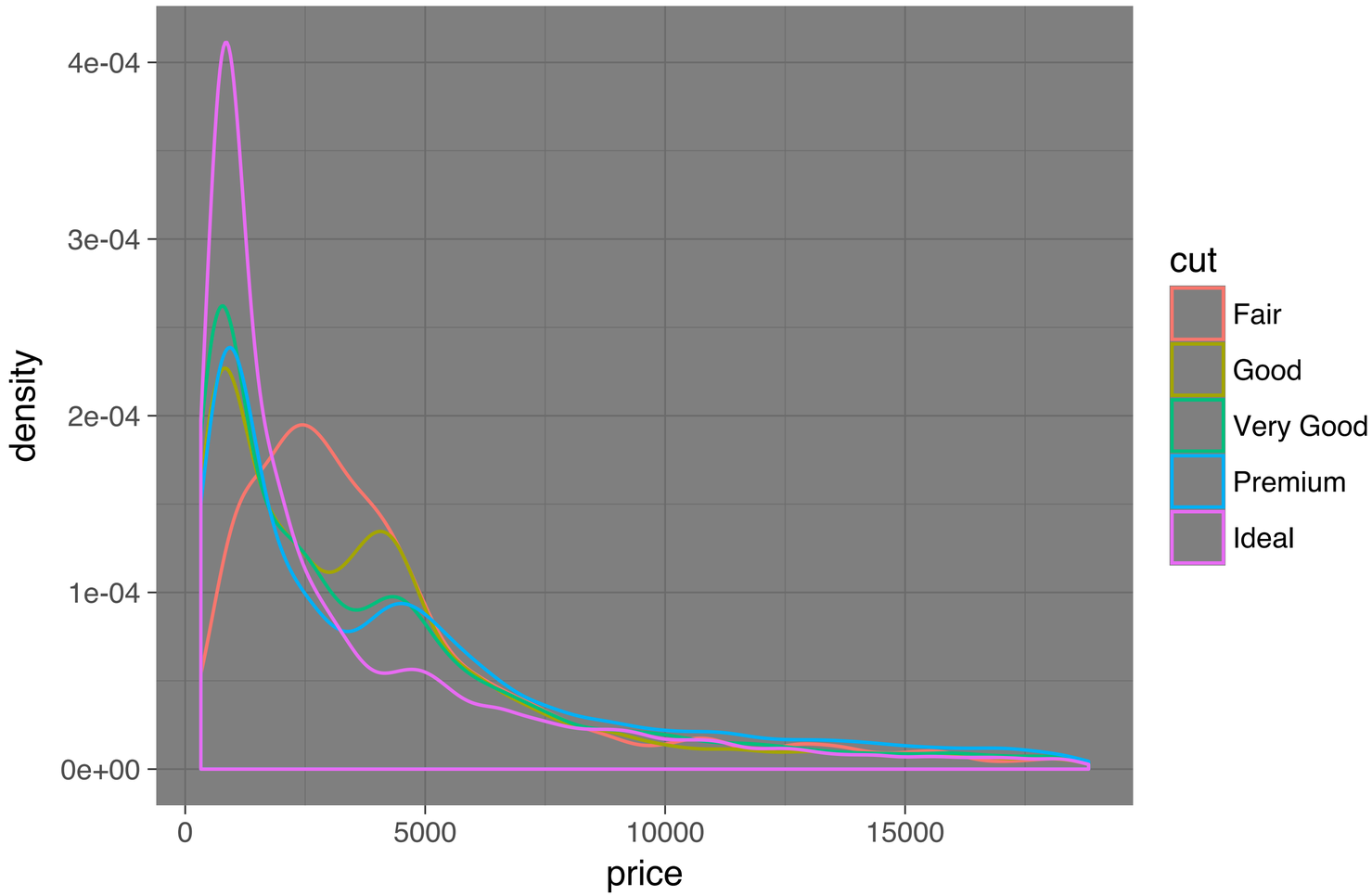

```
p1 + theme_bw()
```

theme_classic()



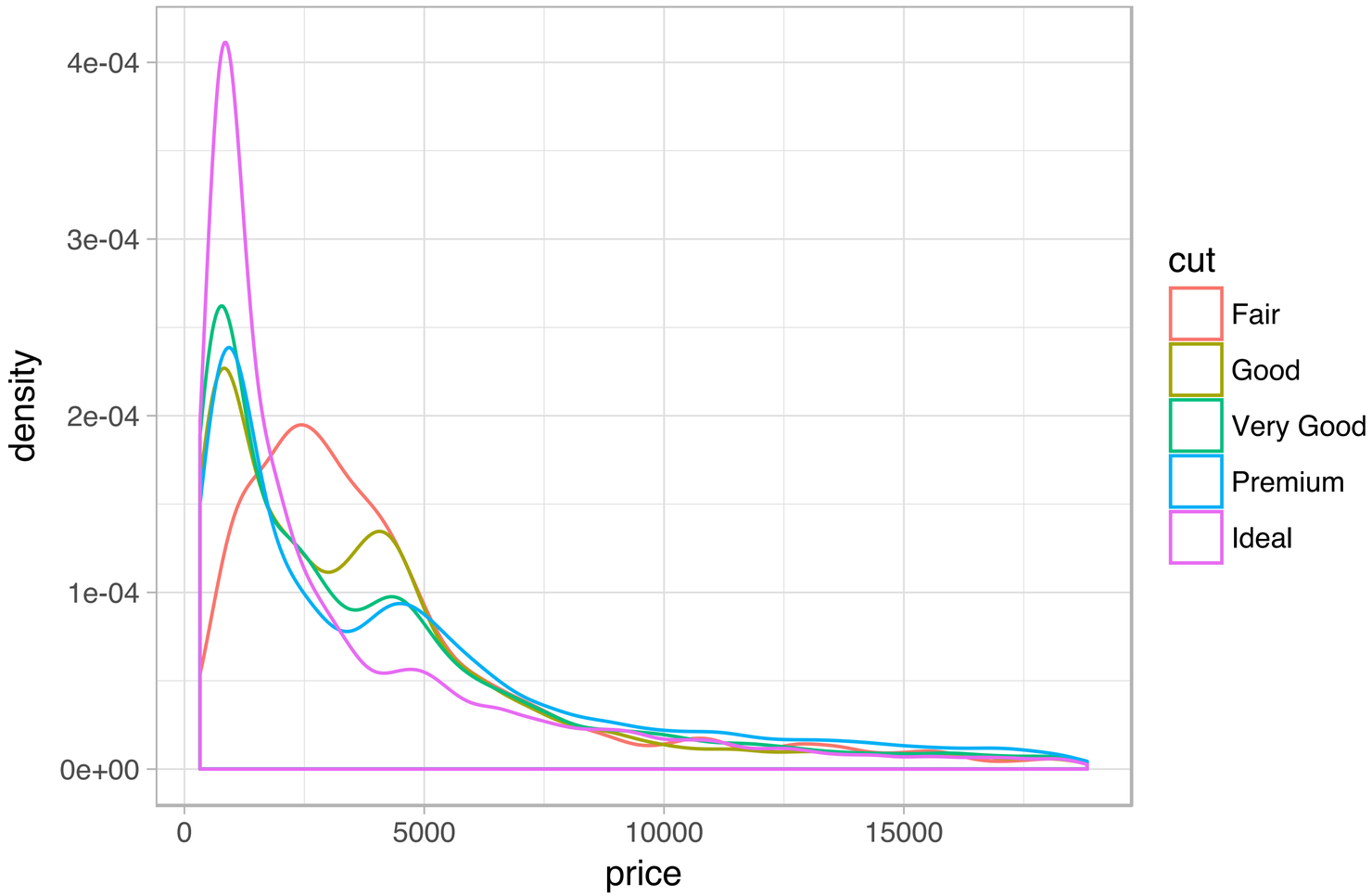

```
p1 + theme_classic()
```

theme_dark()



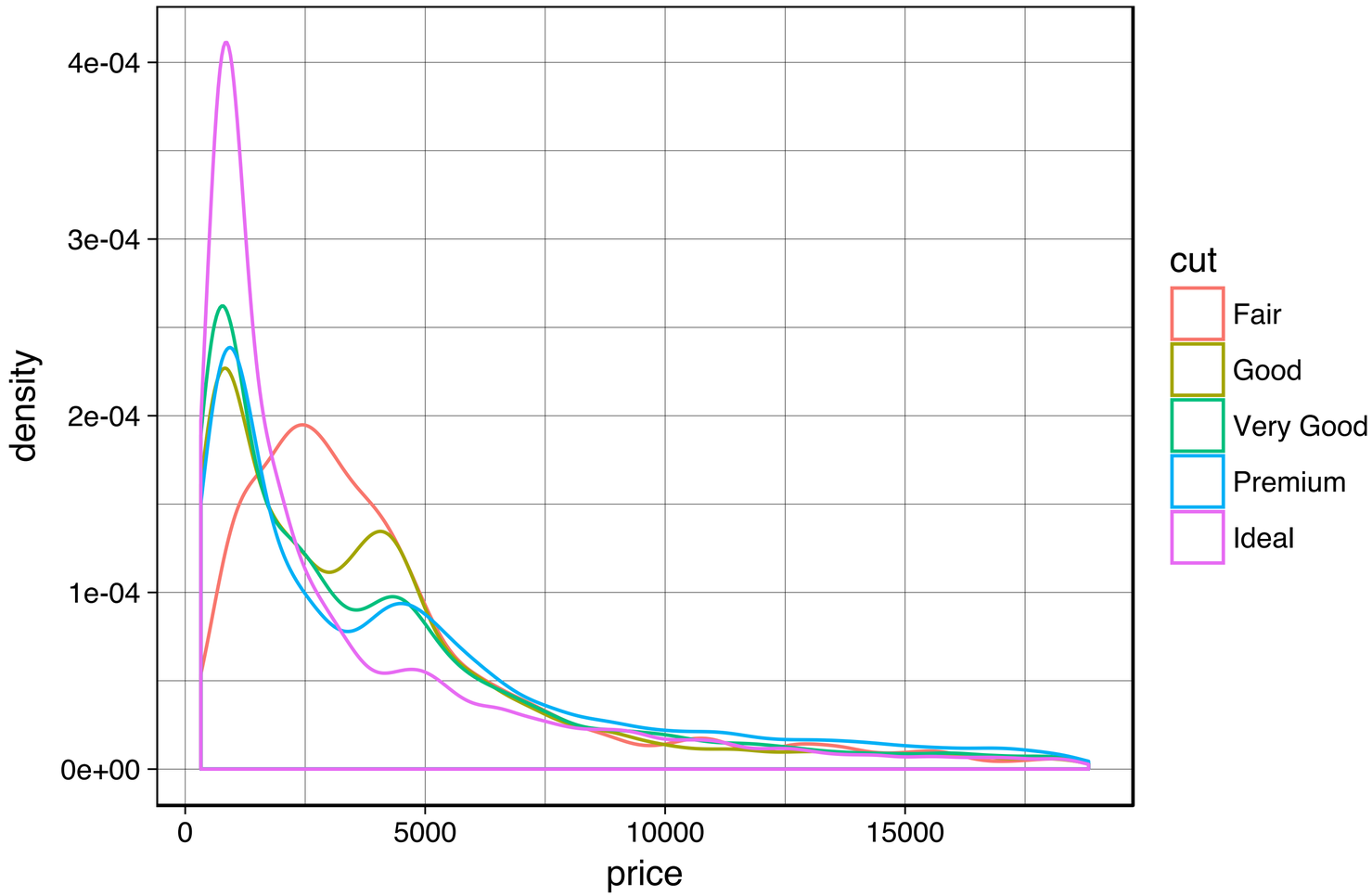

```
p1 + theme_dark()
```

theme_light()



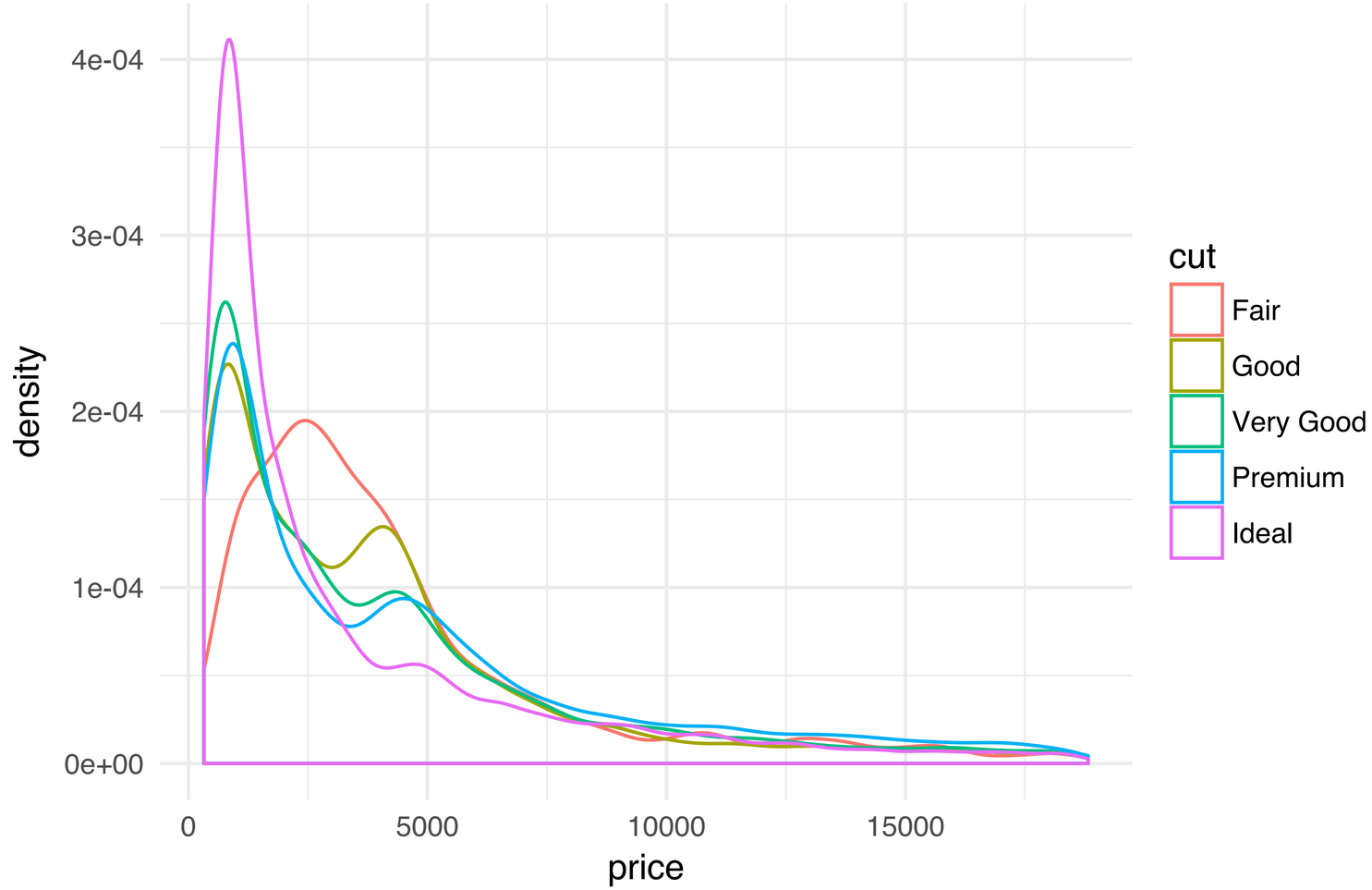

```
p1 + theme_light()
```

theme_linedraw()



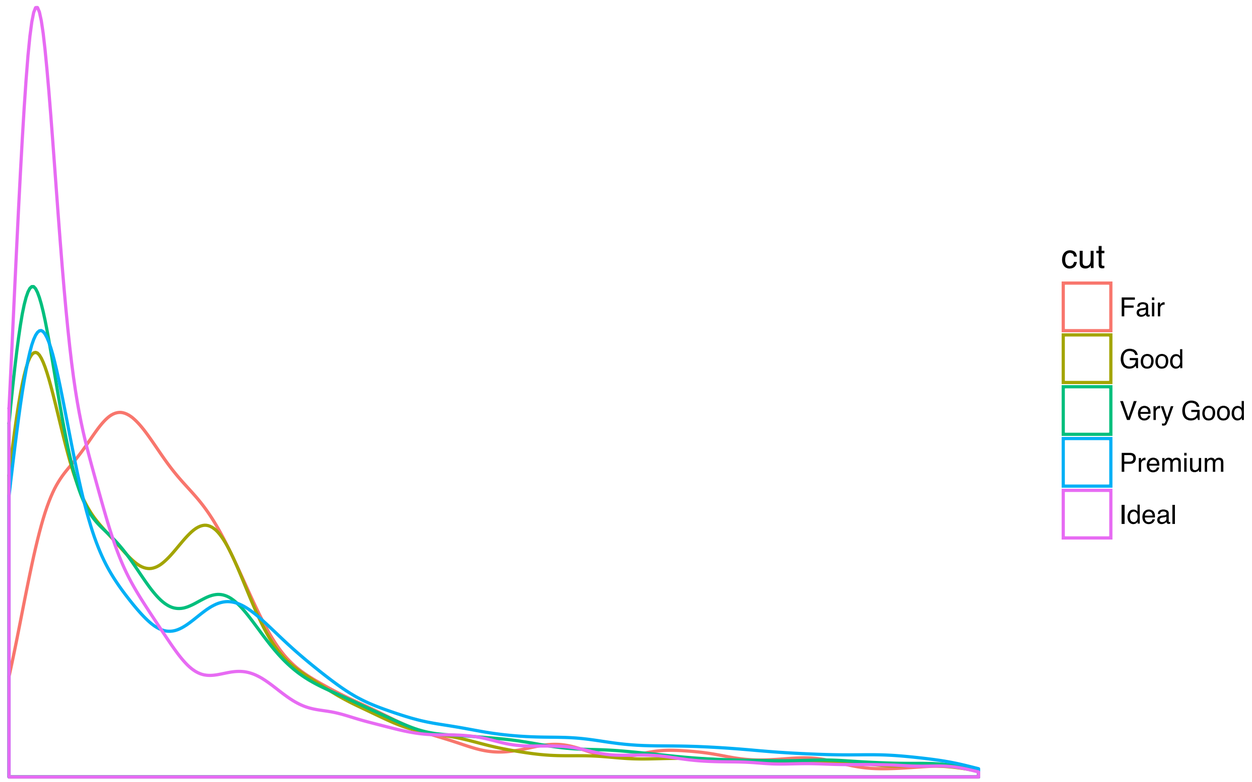

```
p1 + theme_linedraw()
```

theme_minimal()



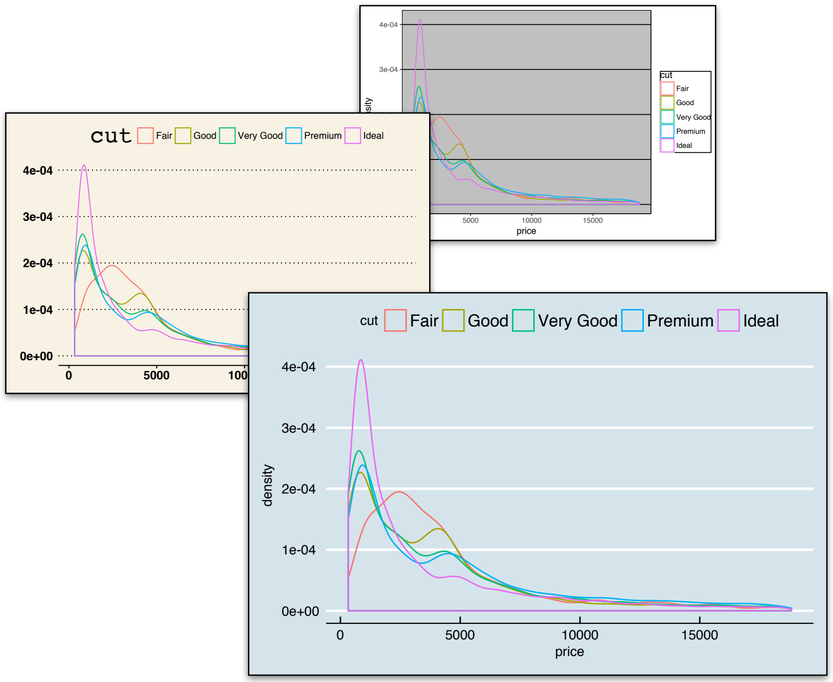

```
p1 + theme_minimal()
```

theme_void()




```
p1 + theme_void()
```

ggthemes



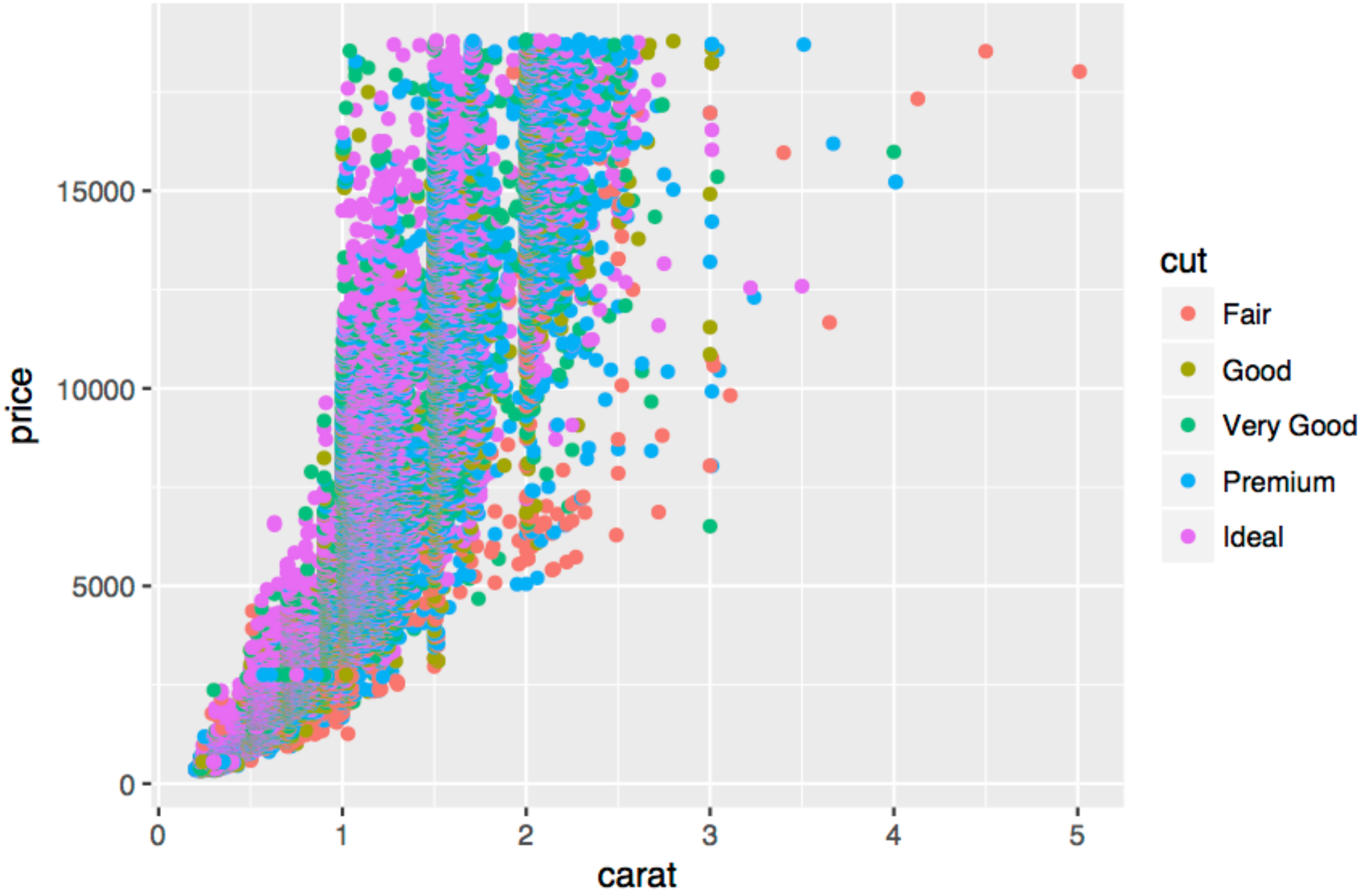
A package of additional ggplot2 themes

```
# 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)

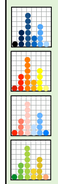
scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

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**)**

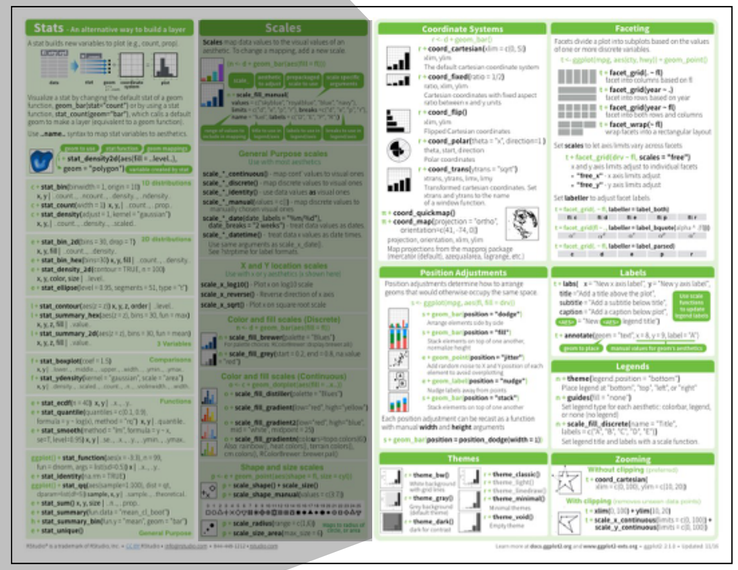
o + **scale_fill_gradientn(**colours=topo.colors(6)**)**
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

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)**)**

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

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



Scales control the appearance of data elements.

ggplot2

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

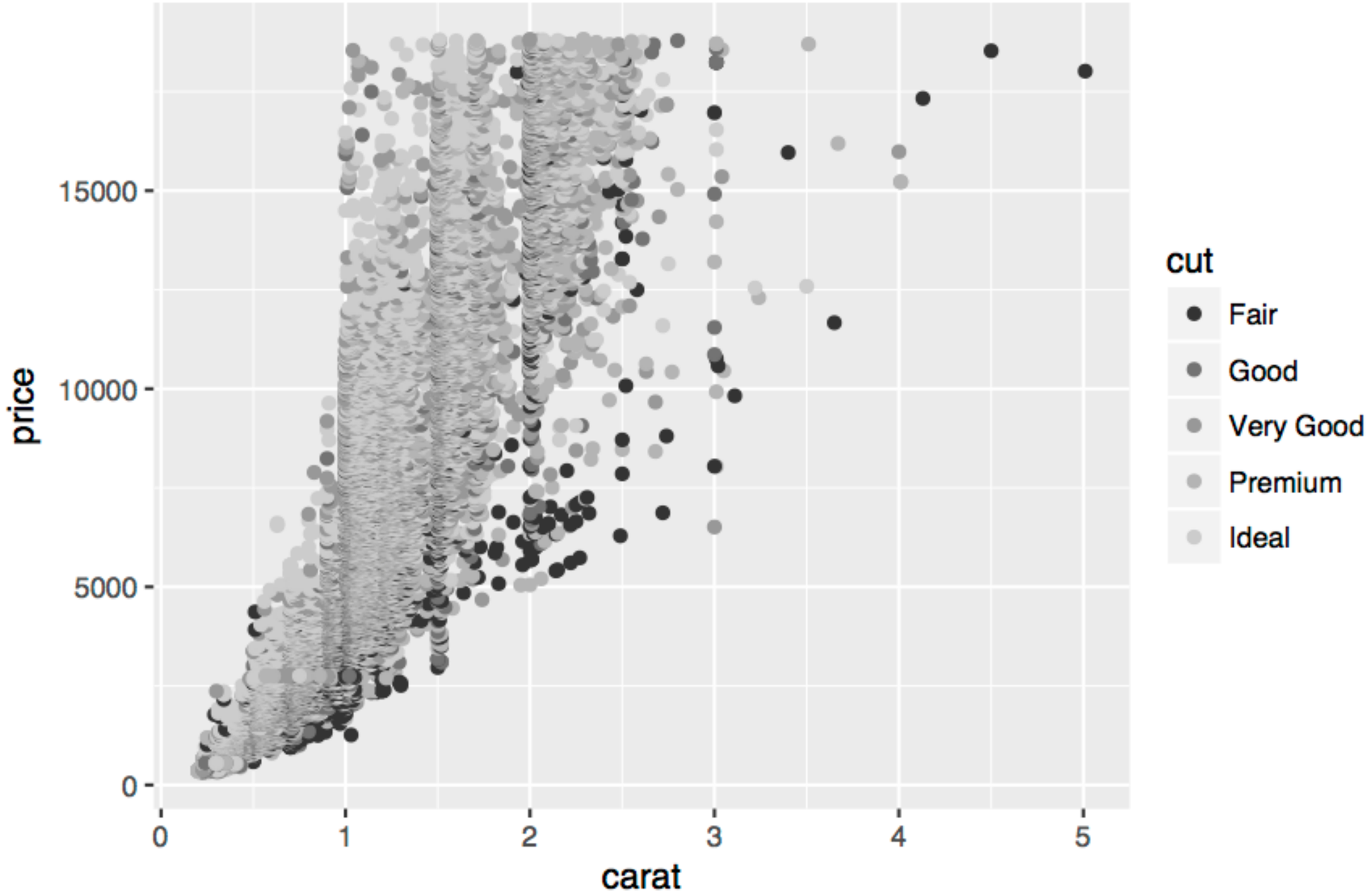
You can add a scale function for each aesthetic.

```
s + scale_color_grey()
```

scale_

aesthetic name

specific scale name
(see cheatsheet or
docs.ggplot2.org)

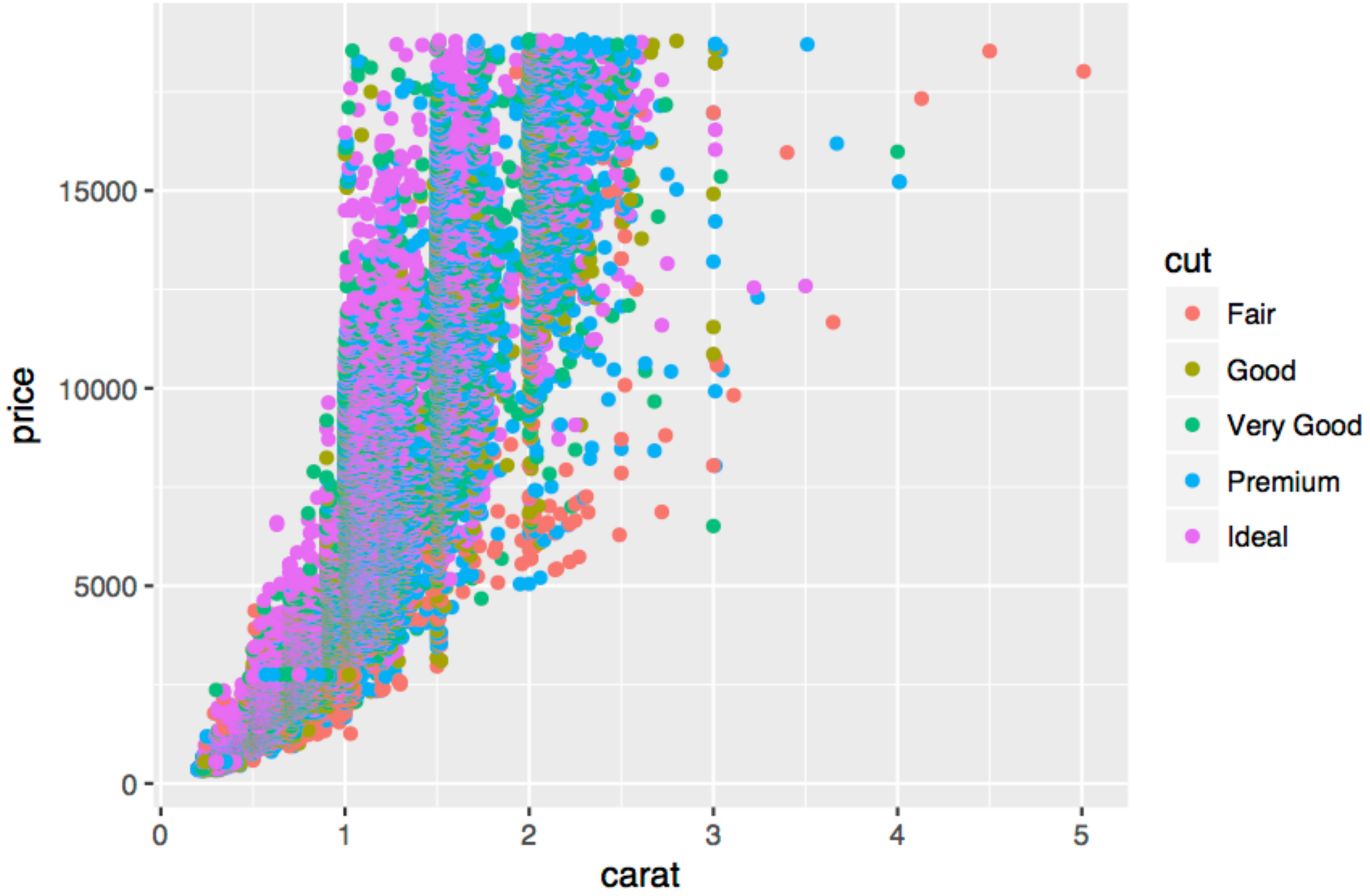


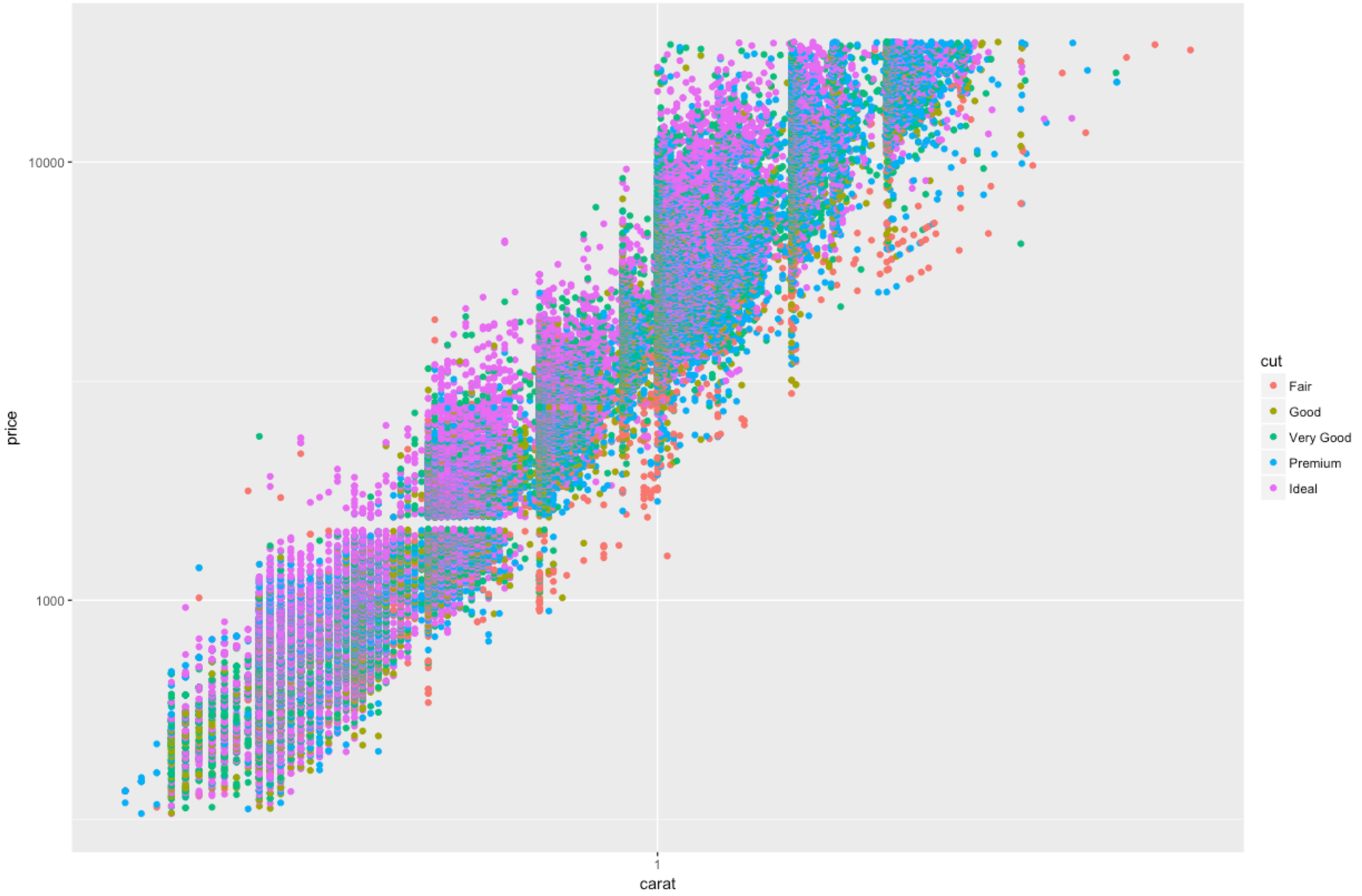

```
s + scale_color_grey()
```

common scales

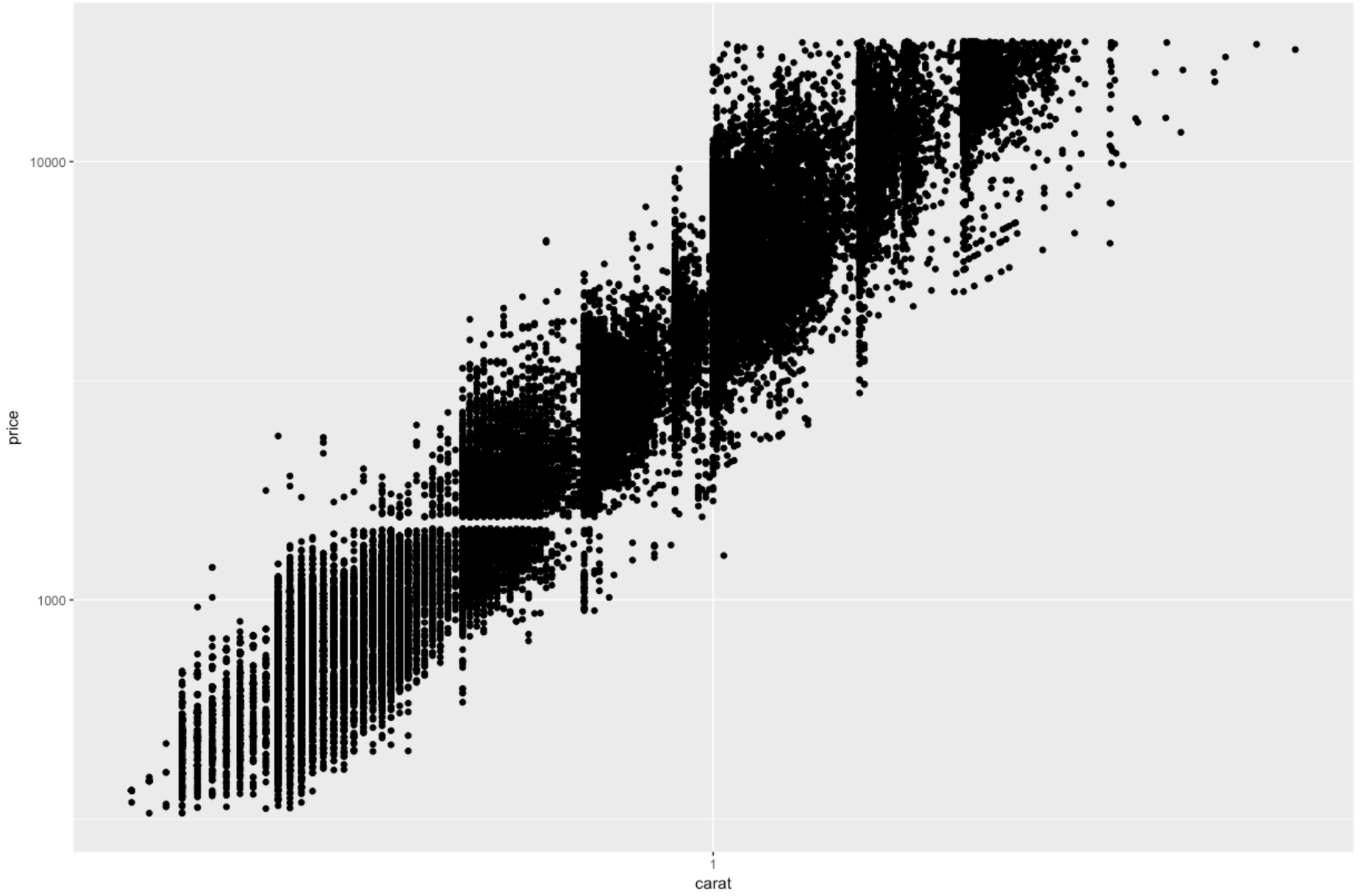
The two most common uses for scales are to

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


price
15000
10000
5000
0
0
1
2
3
4
5
carat
cut
Fair
Good
Very Good
Premium
Ideal

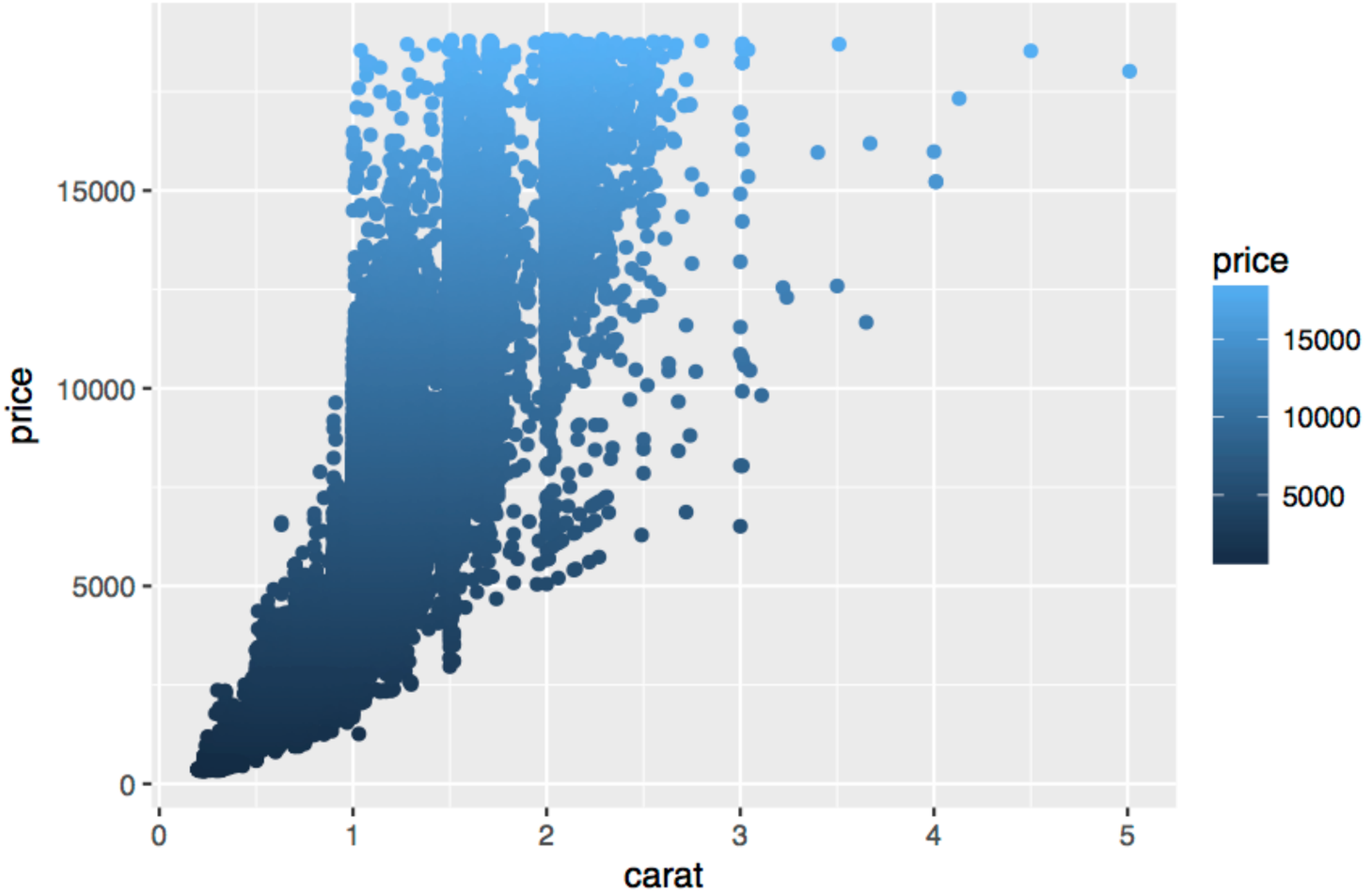



```
s + scale_x_log10() + scale_y_log10()
```



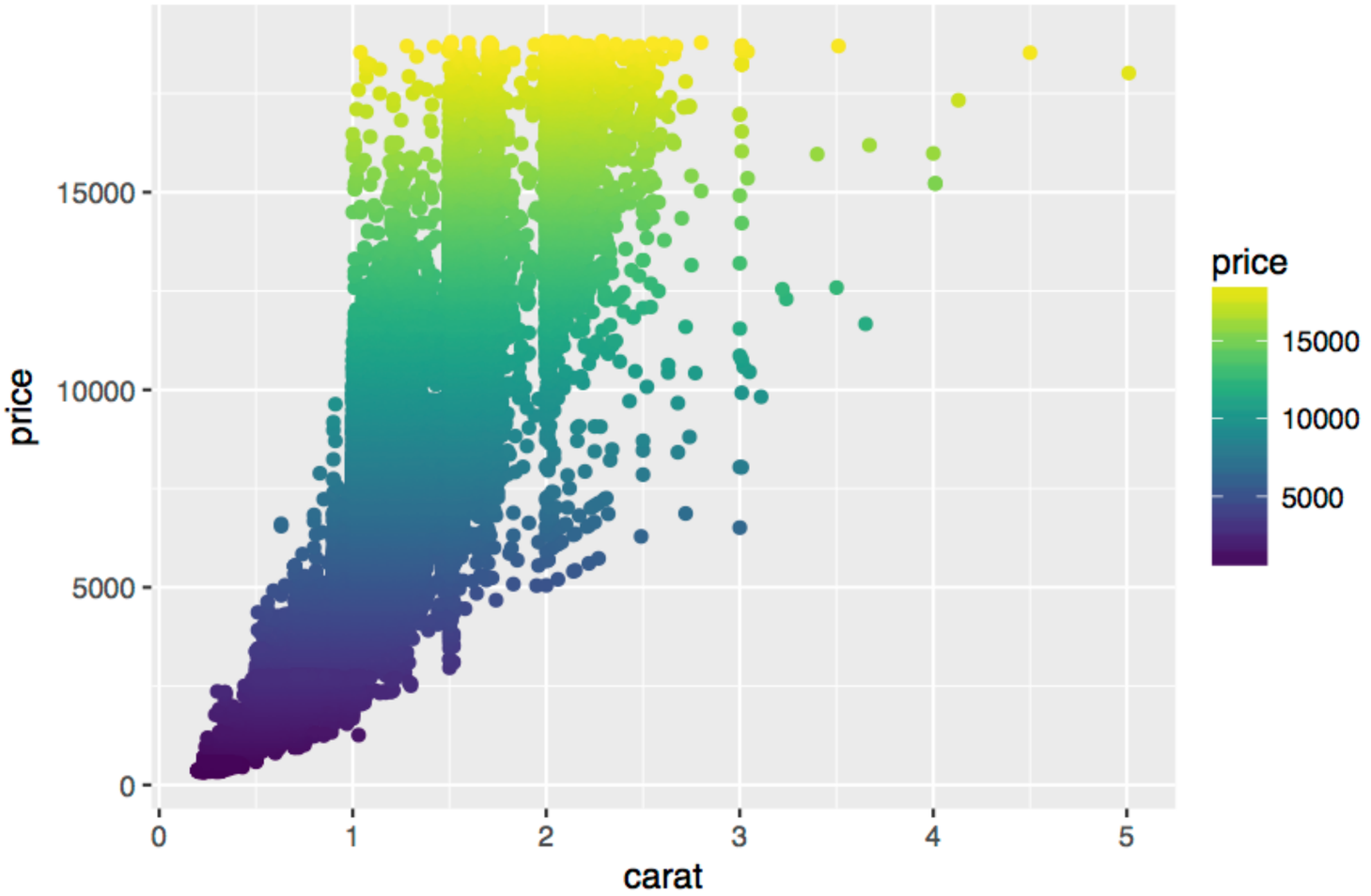

```
s + scale_x_log10() + scale_y_log10()
```

Continuous colors



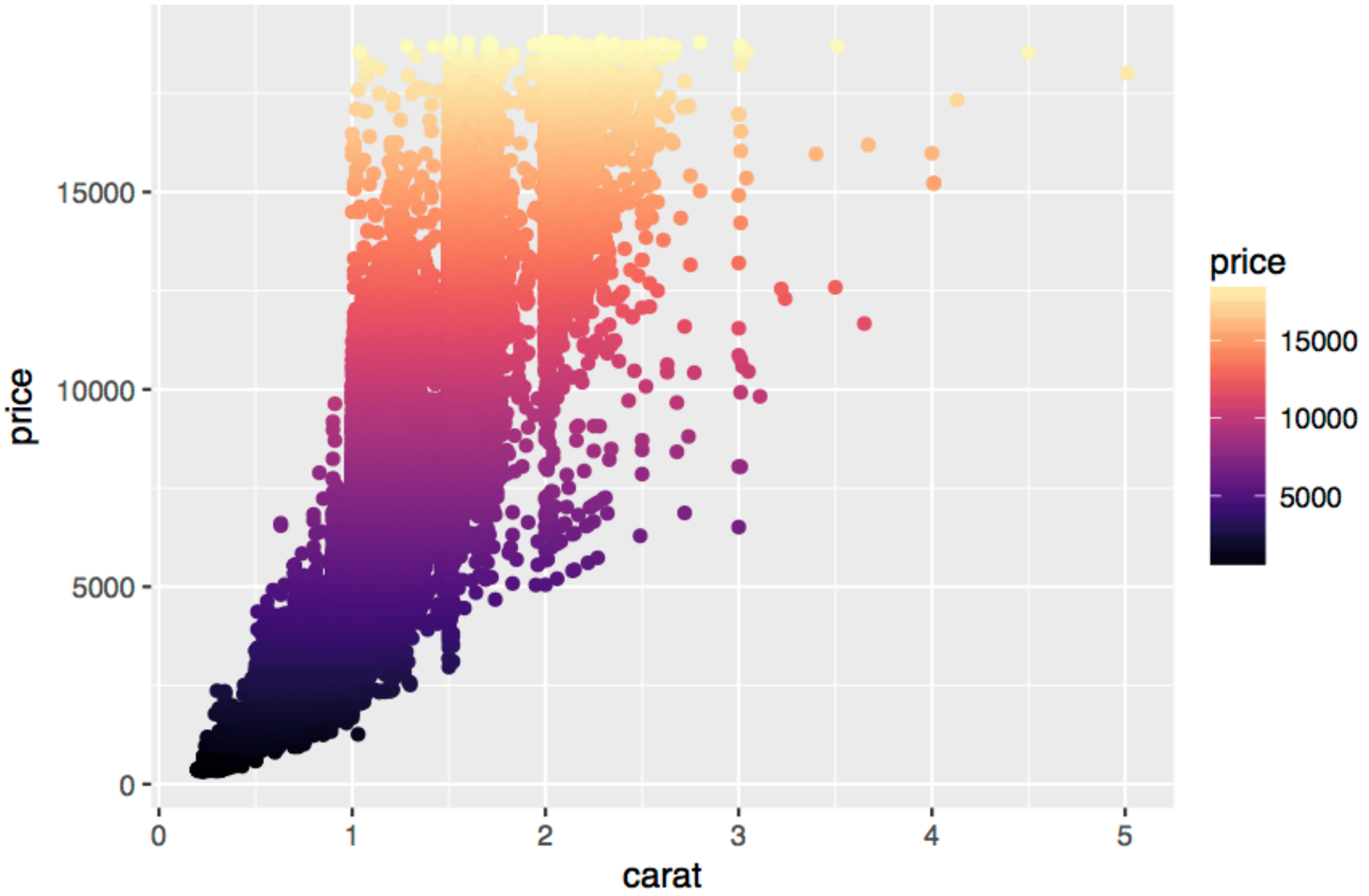

```
s1 <- ggplot(diamonds, aes(carat, price)) +
   geom_point(aes(color = price))
```

Viridis



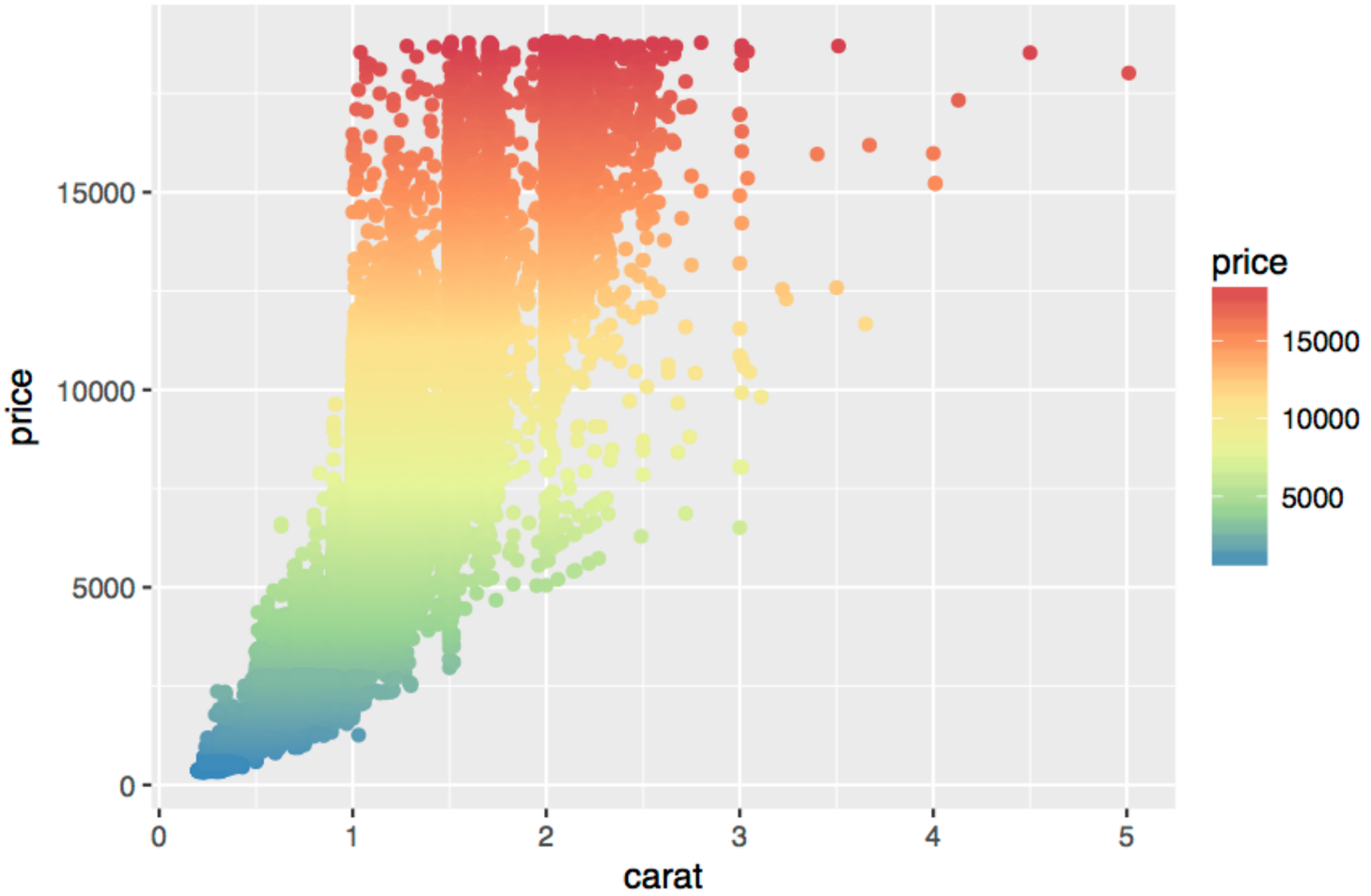

```
library(viridis)
s1 + scale_color_viridis()
```

Viridis




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

distiller




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

RColorBrewer

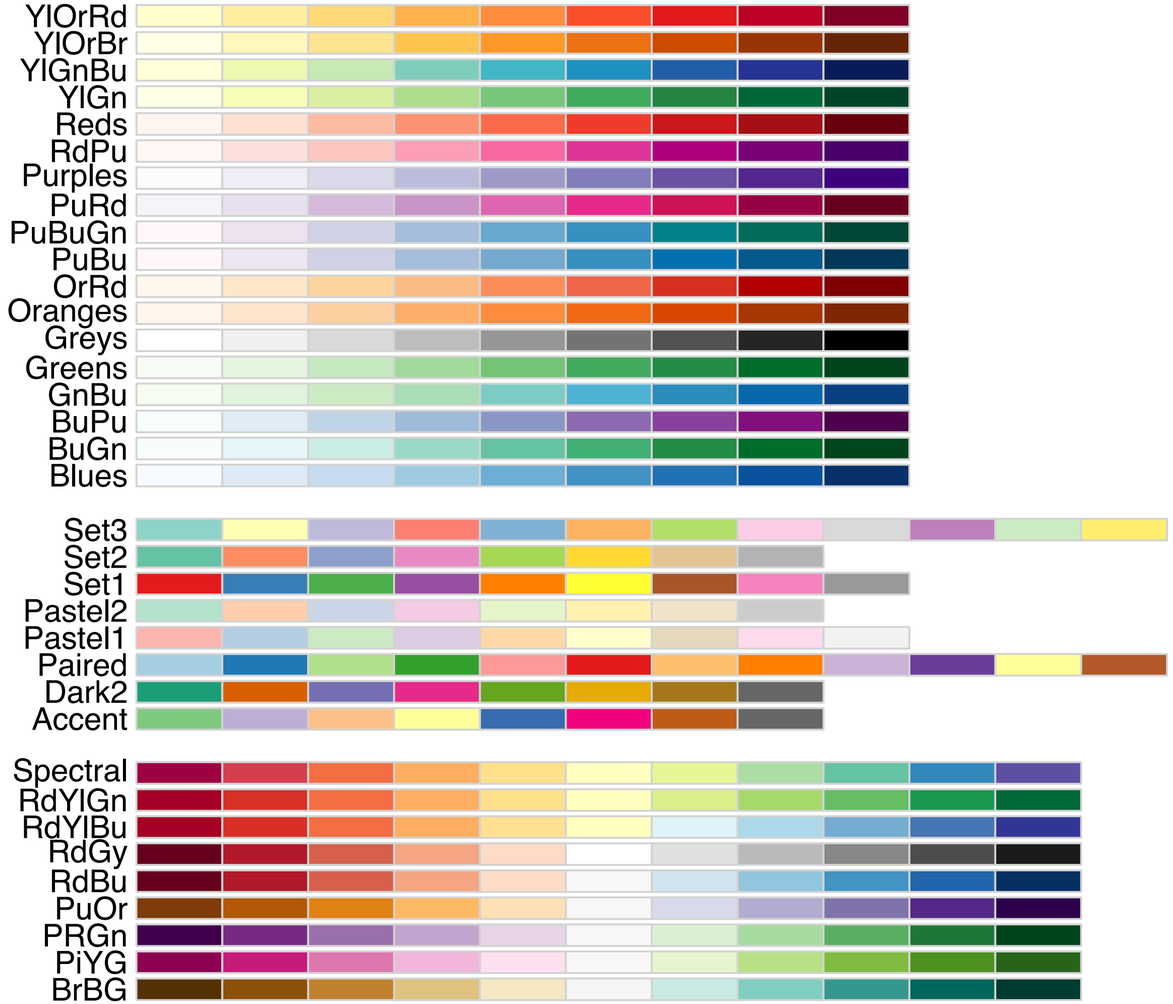
A package of color palettes

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

scale_color_distiller()* → continuous variables

scale_color_brewer()* → discrete variables

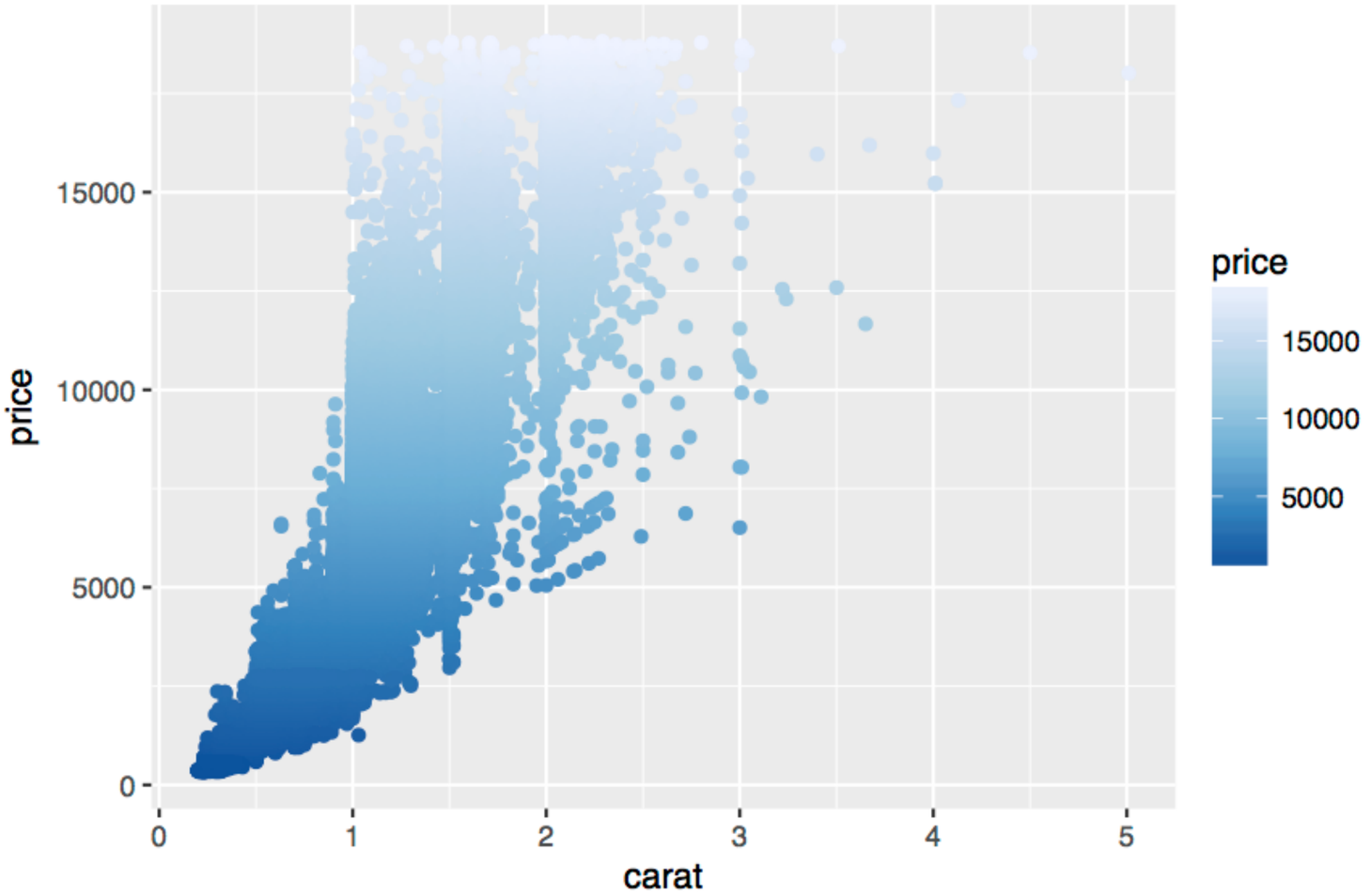
* Also scale_fill_distiller() and scale_fill_brewer()



To see available palettes

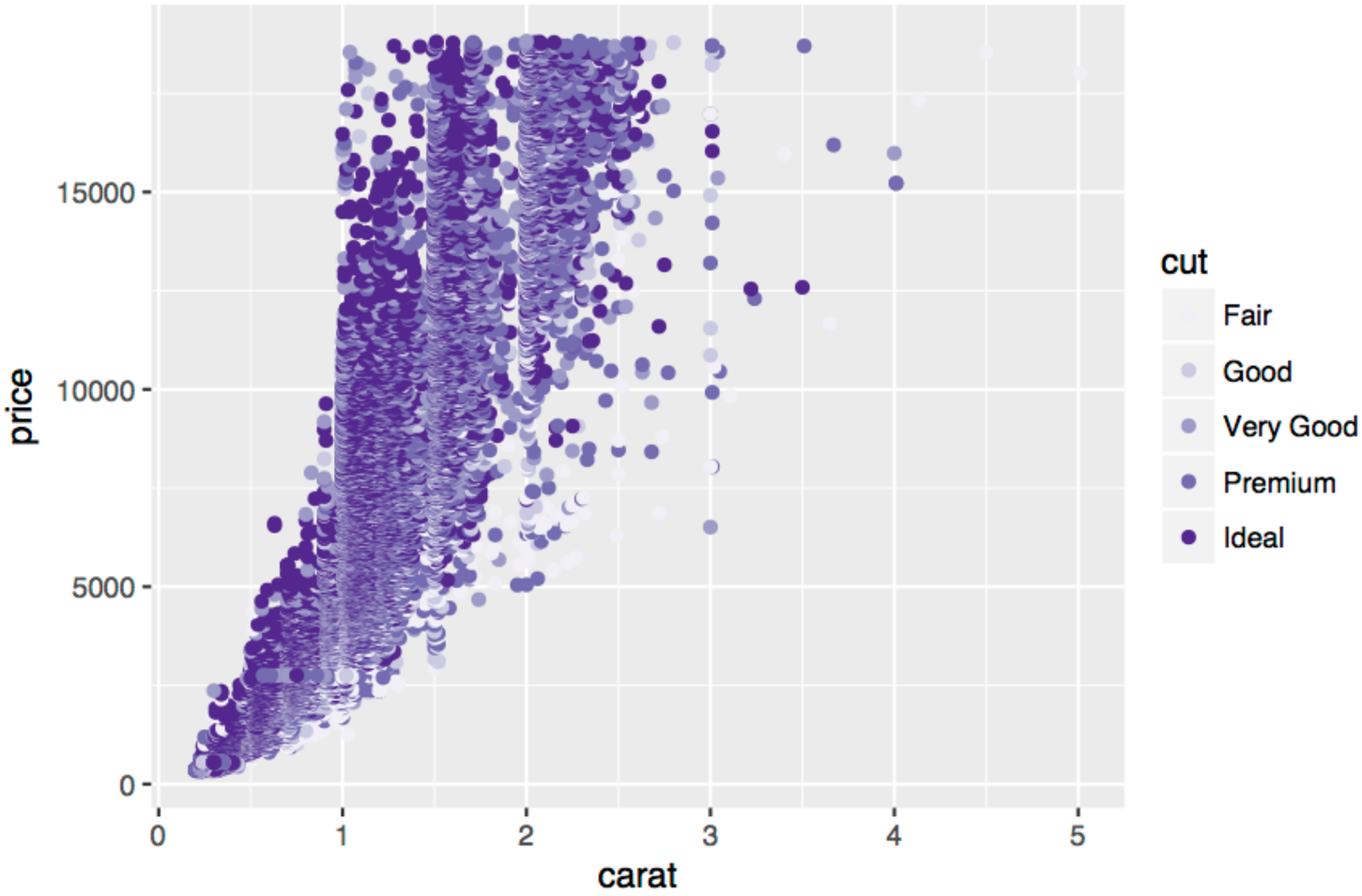
```
RColorBrewer::display.brewer.all()
```

distiller



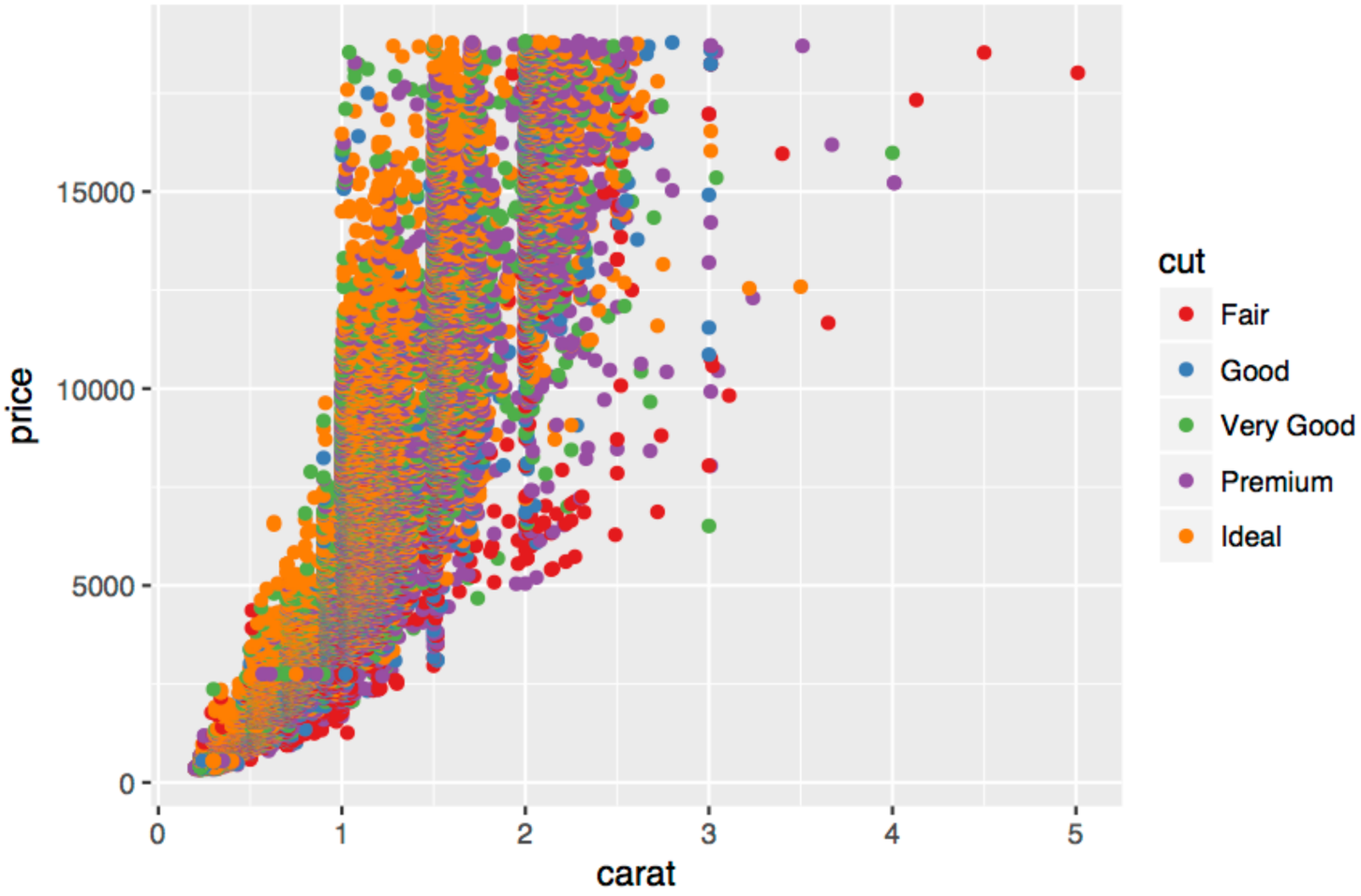

```
s1 + scale_color_distiller(palette = "Blues")
```

brewer




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

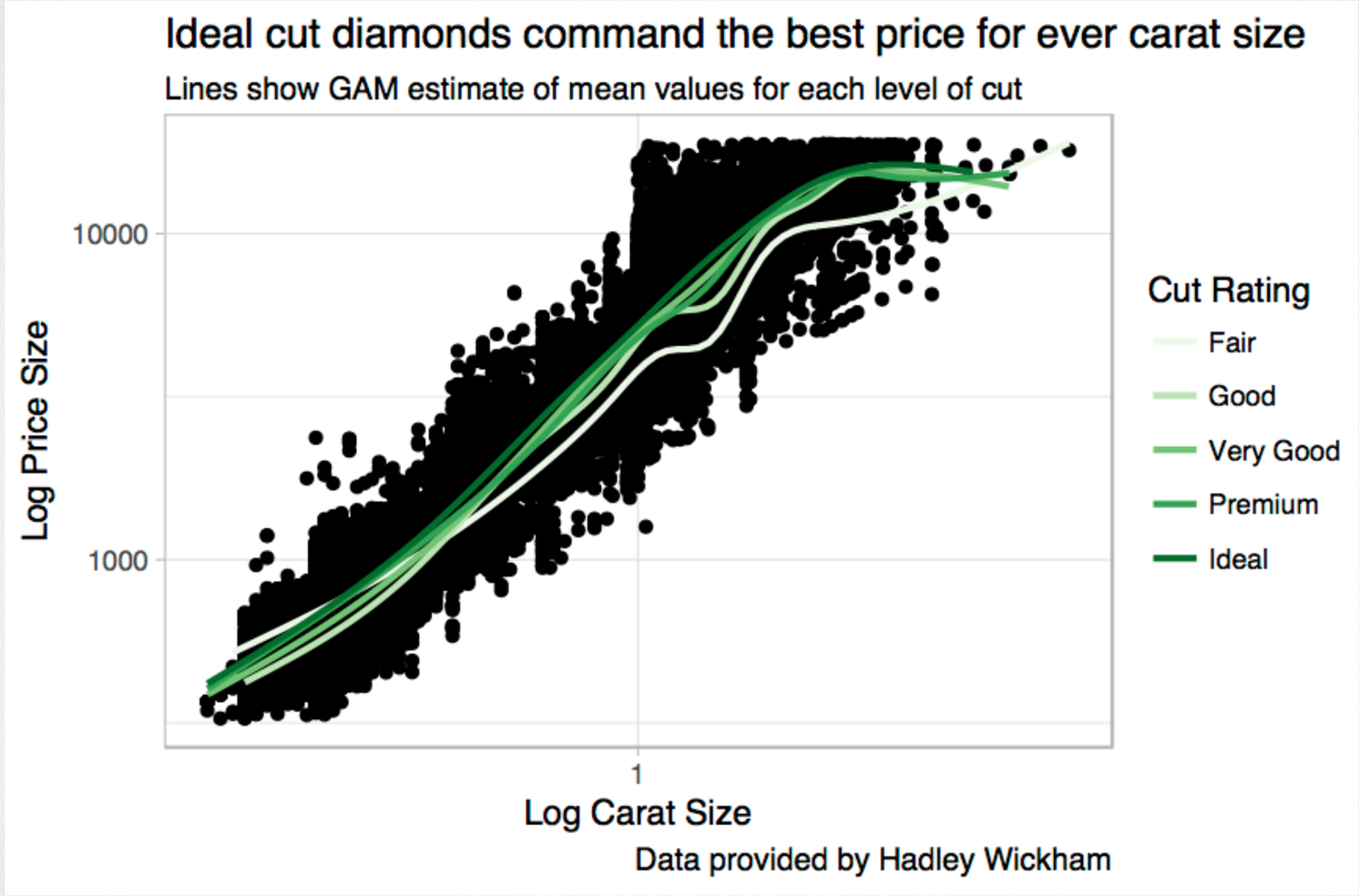
brewer




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

Your turn

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

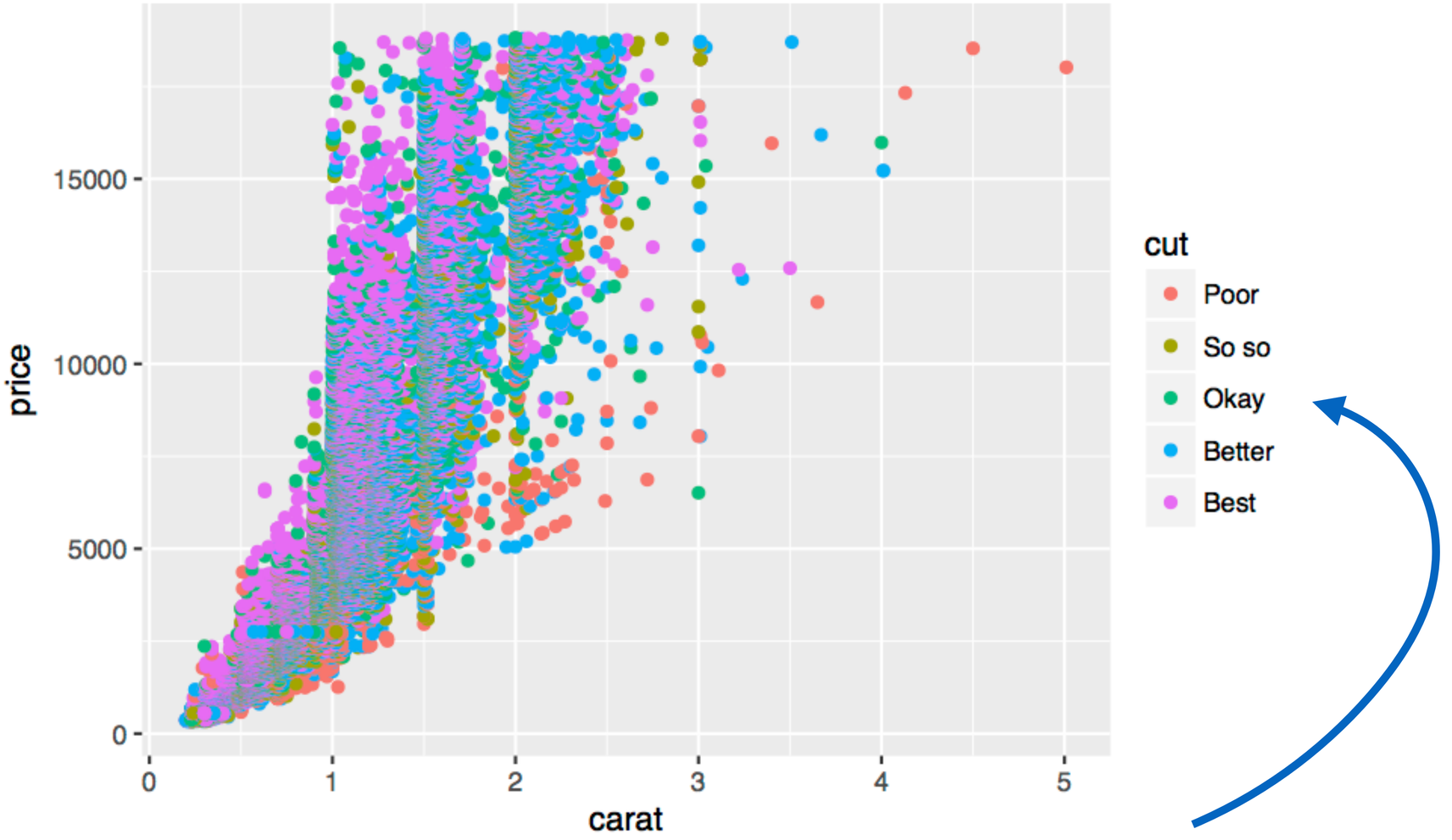


10:00

```
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"))
```

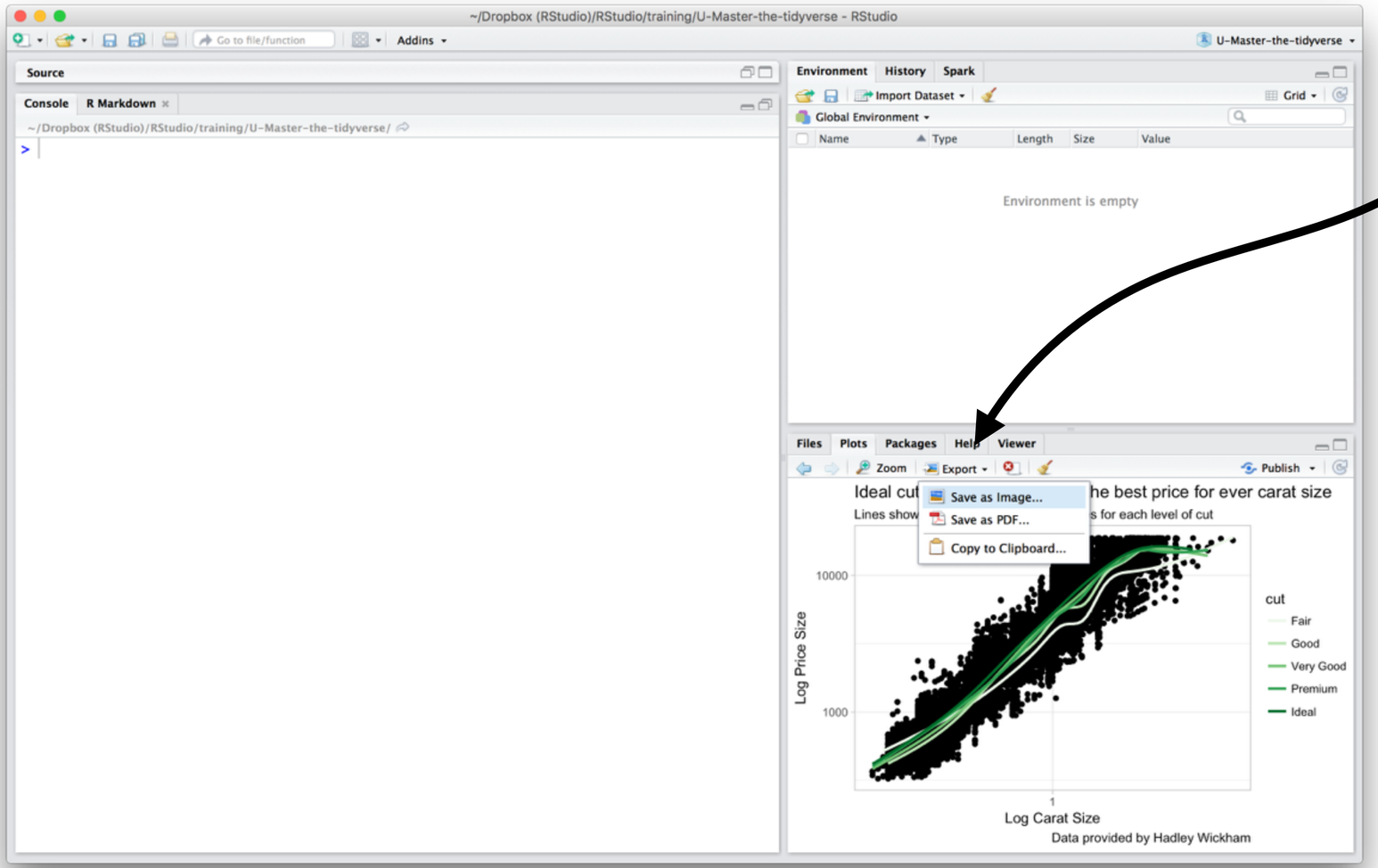
Default scales

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?

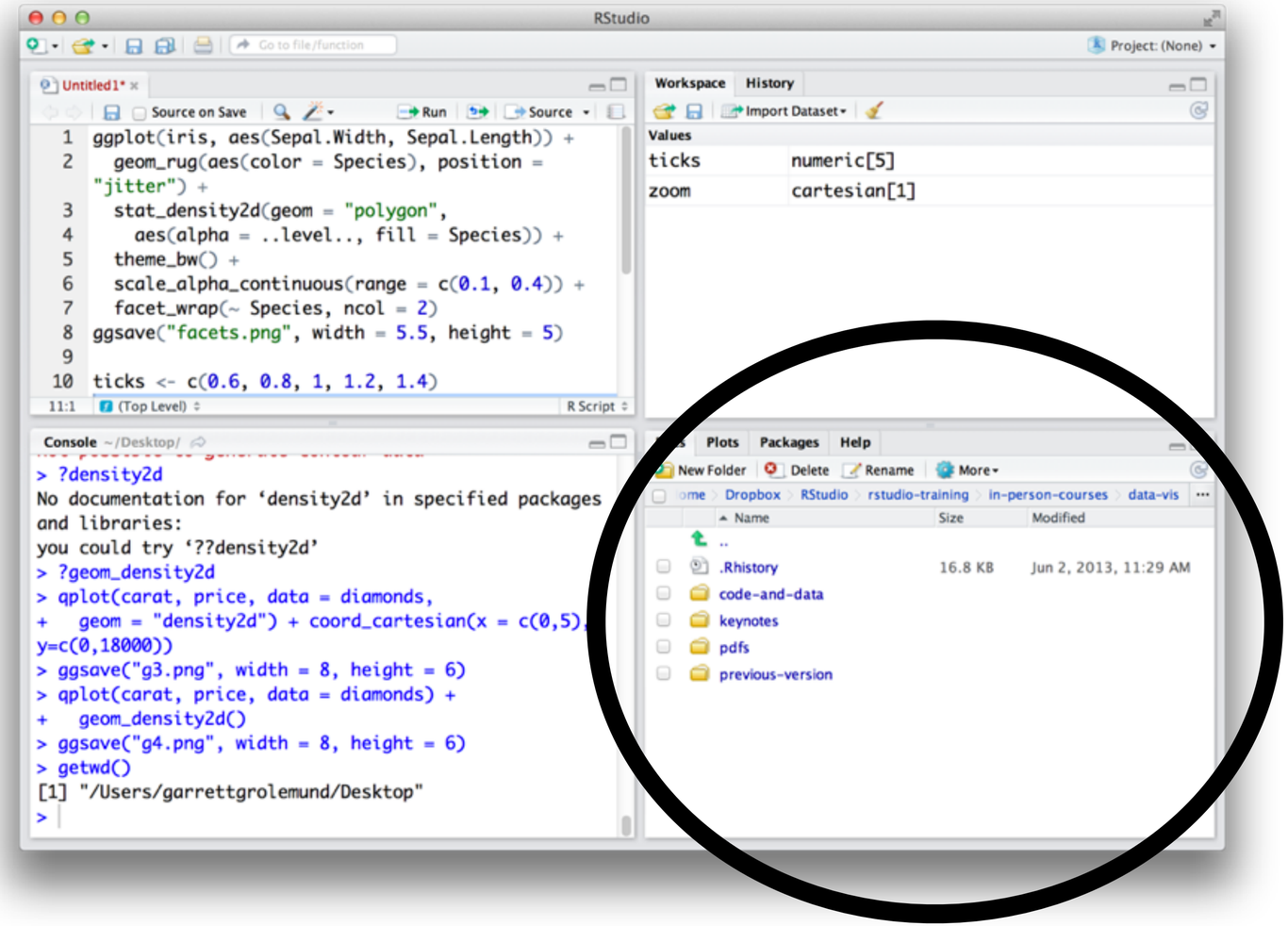
```
get_wd()
```

Working directory

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

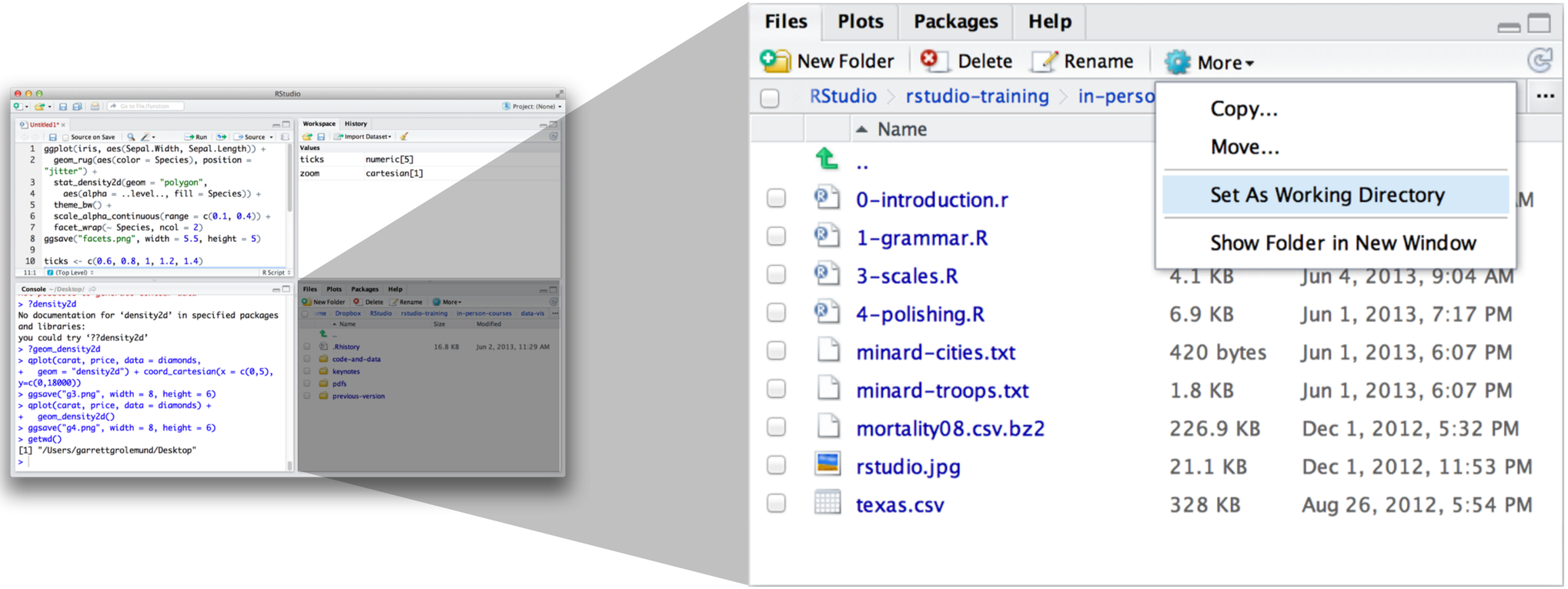
- This folder is known as your "working directory"
- When you save files, R will save them here
- When you load files, R will look for them here

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



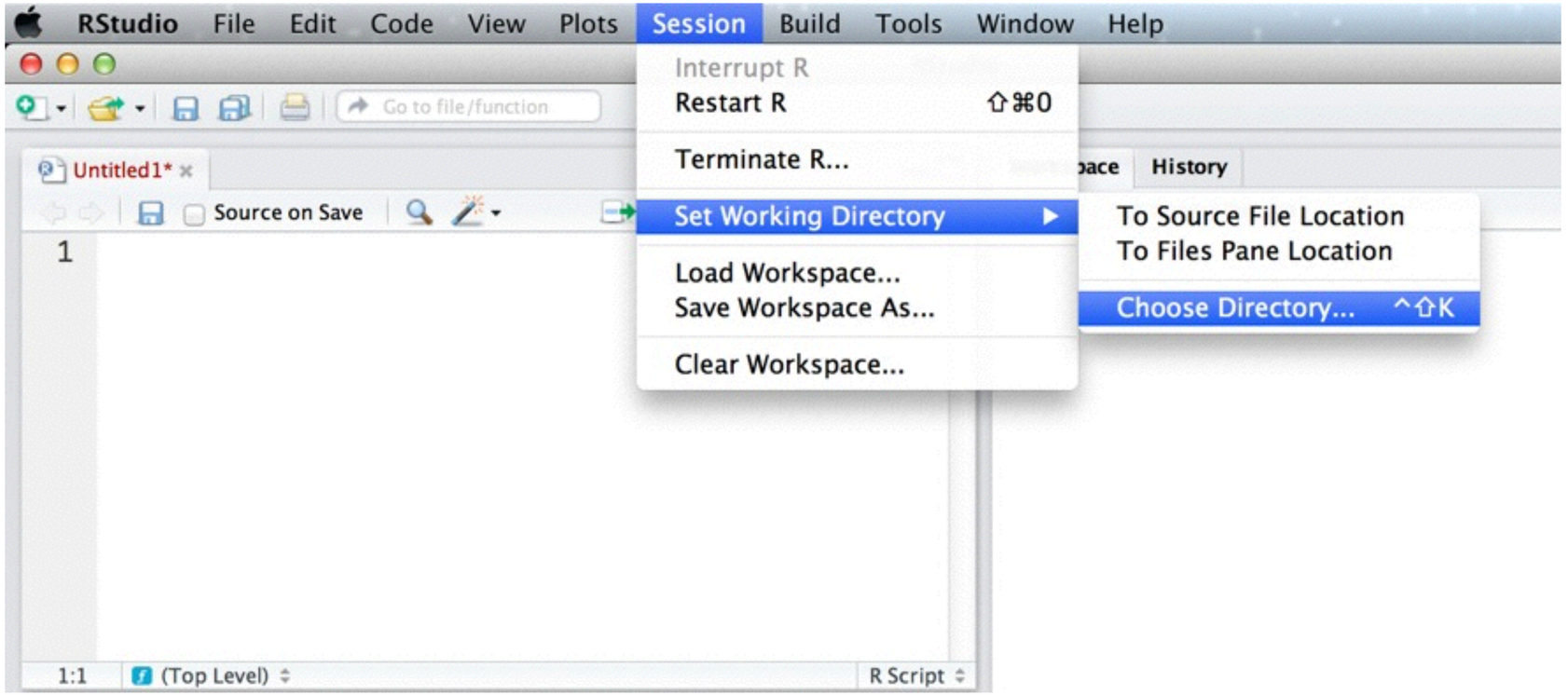
Changing the Working directory

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

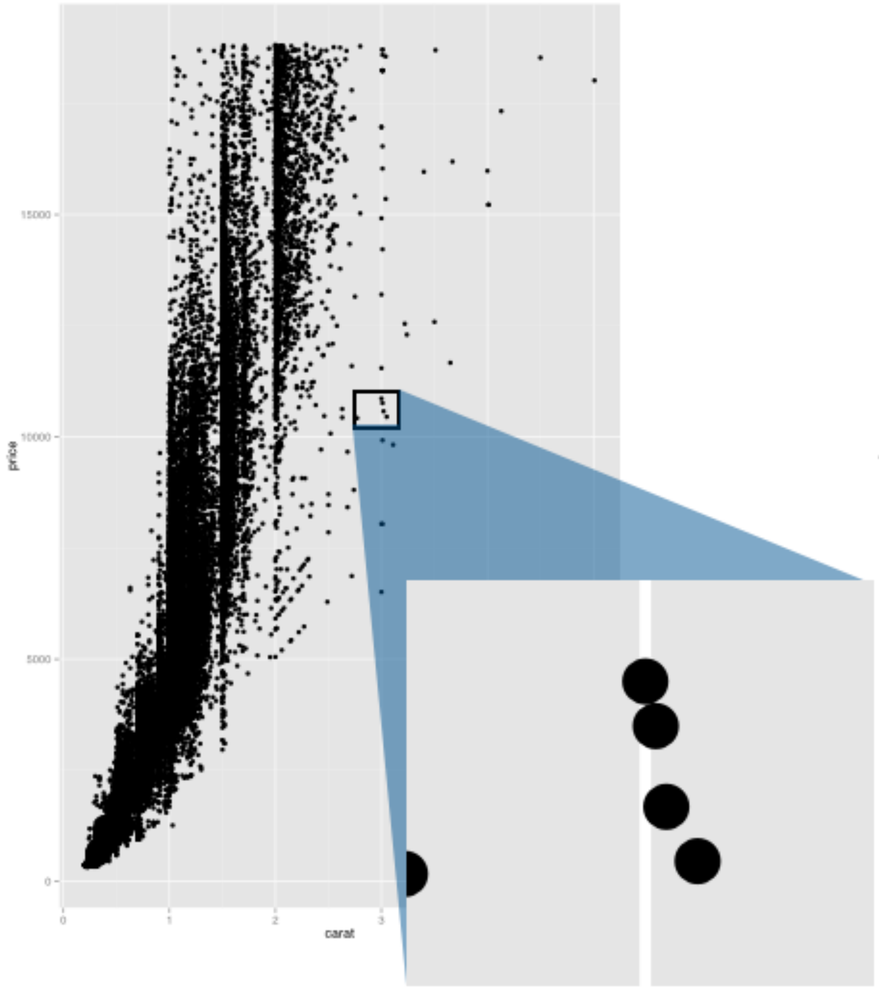
`ggsave()` saves the last plot.

Uses size on screen:

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

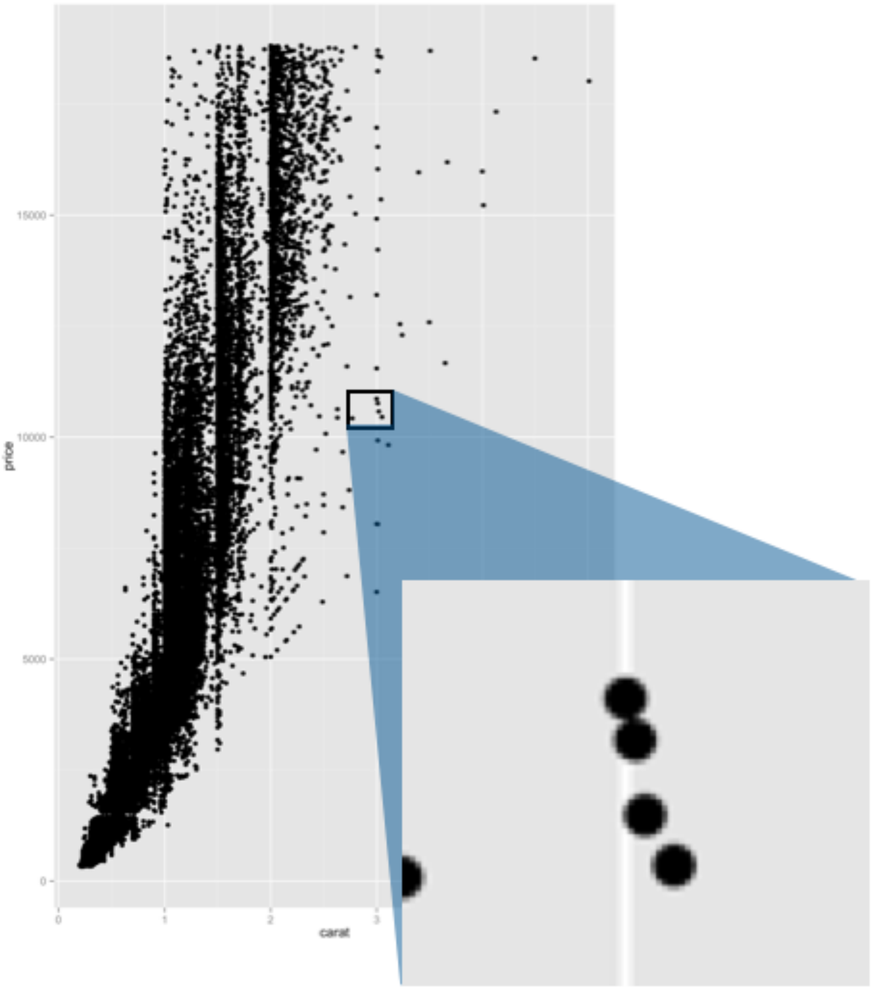
Specify size in inches

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



PDF

vector based
good for most plots

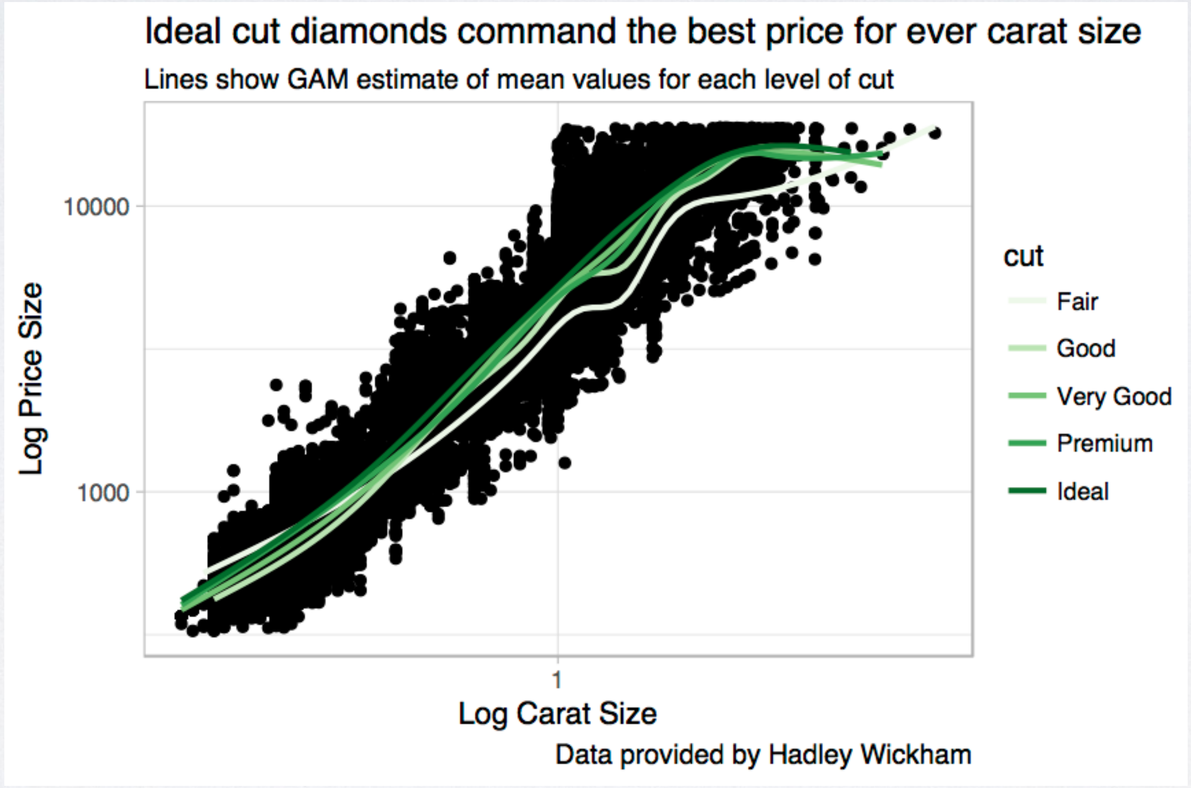


PNG

raster based
faster for many points

Your turn

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



Data Visualization with