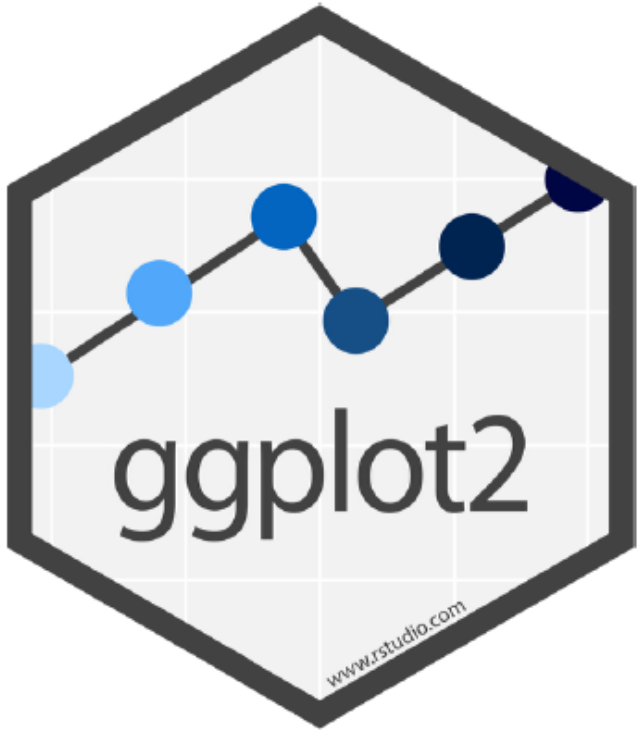
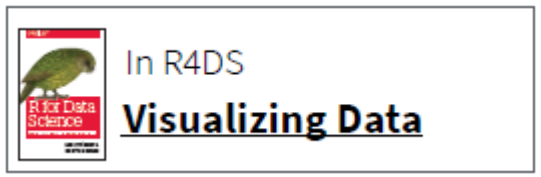
Visualize Data

ggplot2

One of the earliest packages in the tidyverse

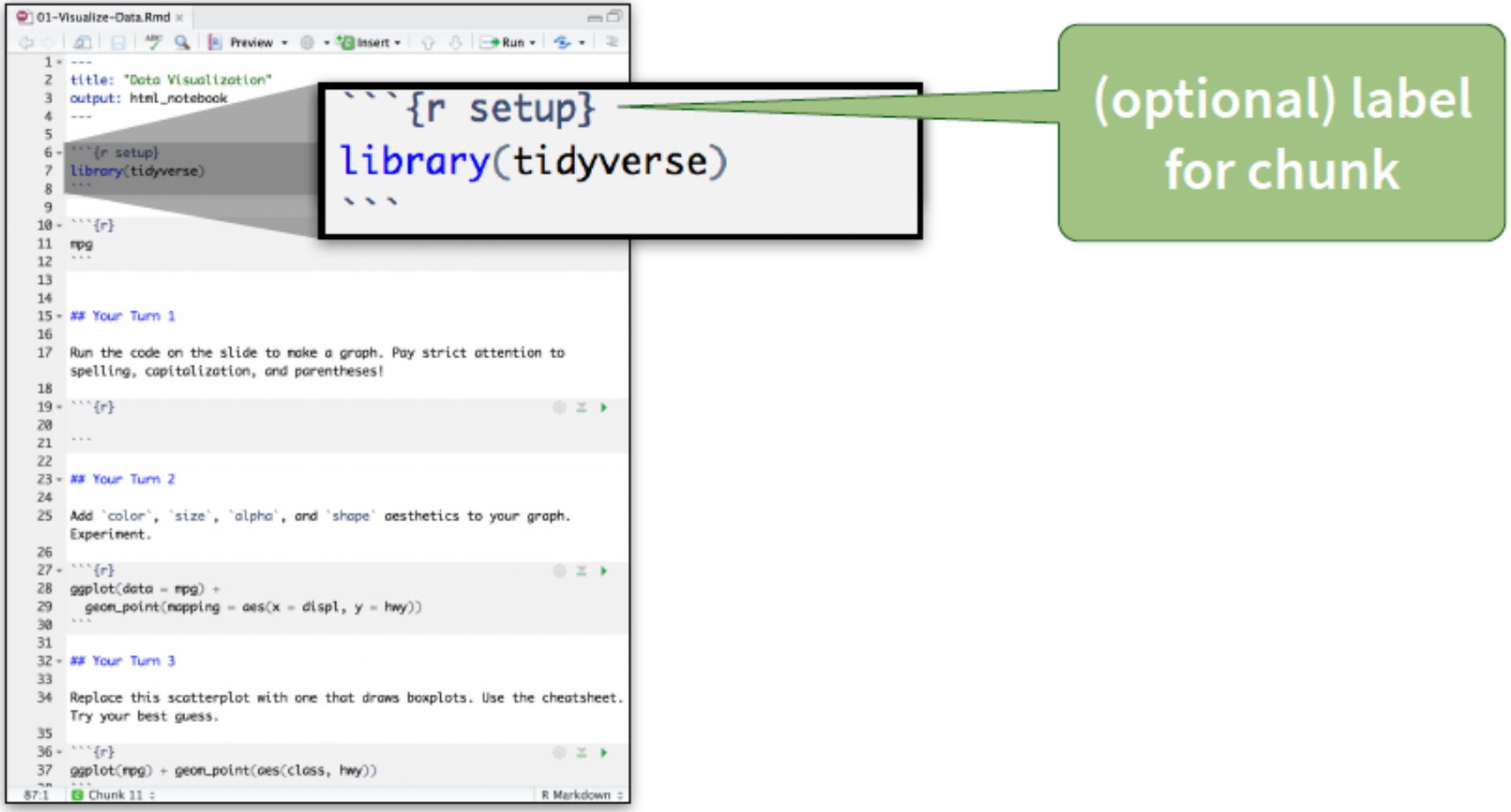
Implements the “grammar of graphics”





Setup

The setup chunk is always run once before anything else



About the data

Monthly averages of water quality data (e.g., temperature, salinity, dissolved oxygen, etc.) from 6 National Estuarine Research Reserves

```
```{r loaddata}
wq ← read.csv(here::here('data', 'monthly_wq.csv'))

```
```

Notice the use of the `here` package

Be sure to check out the data

- You can use:

`` `head()`, `tail()`, `str()`, ``

or

`` ‘dplyr::glimpse()` ``

QUIZ

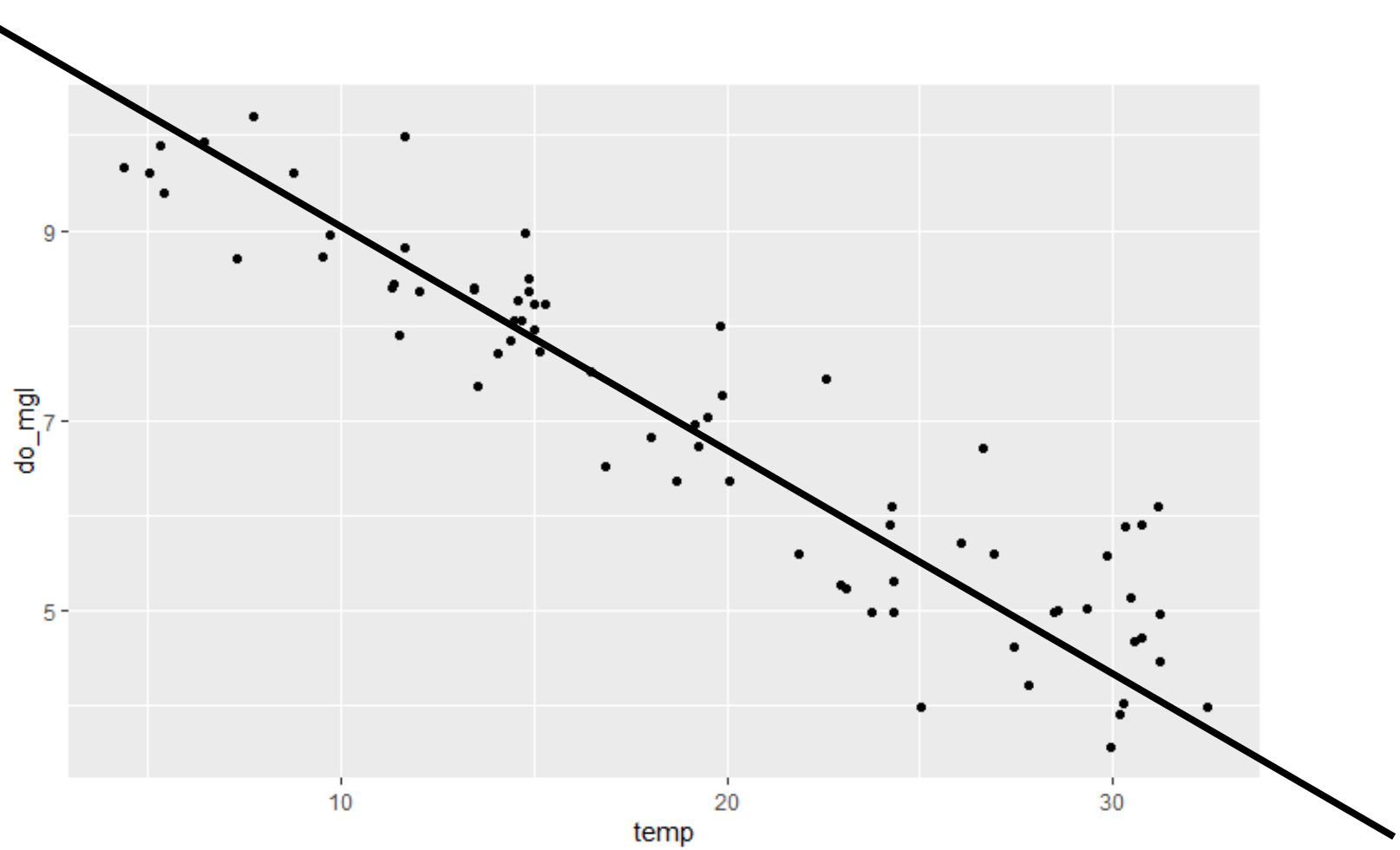
What relationship do you expect to see between temperature (`temp`) and dissolved oxygen (`do_mgl`)?

YOUR TURN 1

Run this code in your notebook to make a graph.

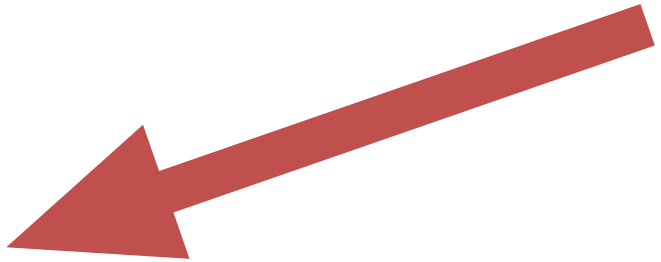
Pay careful attention to spelling, capitalization, and parentheses!

```
ggplot(data = wq) +
    geom_point(mapping = aes(x = temp, y = do_mgl))
```




```
ggplot(data = wq) +
    geom_point(mapping = aes(x = temp, y = do_mgl))
```

Be sure to always put the '+' at the end of a line, never at the start.



```
ggplot(data = wq) +
    geom_point(mapping = aes(x = temp, y = do_mgl))
```

1. Initialize the figure with `ggplot()`
2. Add layers with `geom_` functions

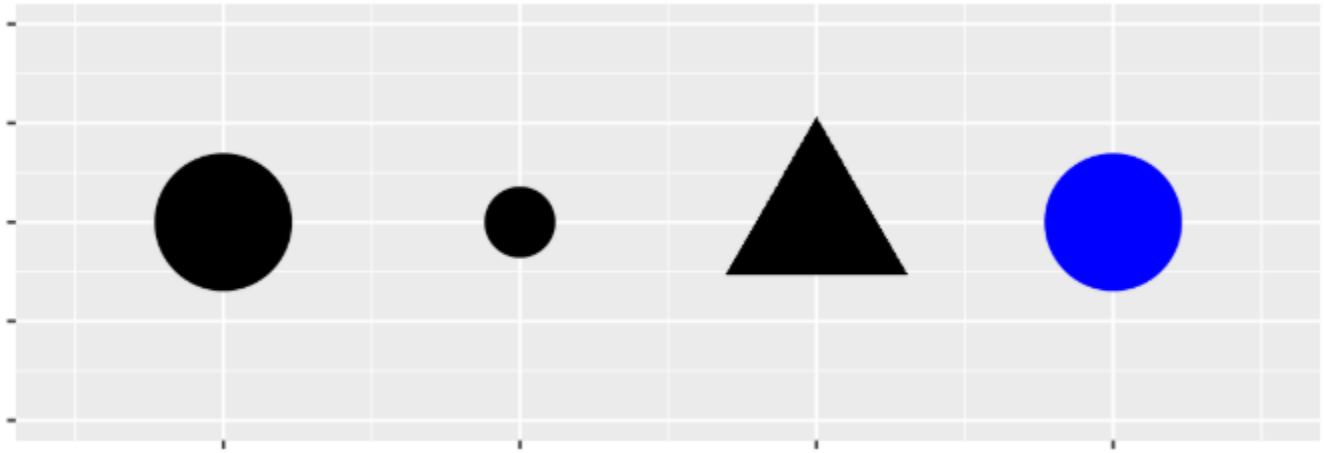
A template

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

```
geom_point(mapping = aes(x = temp, y = do_mgl))
```

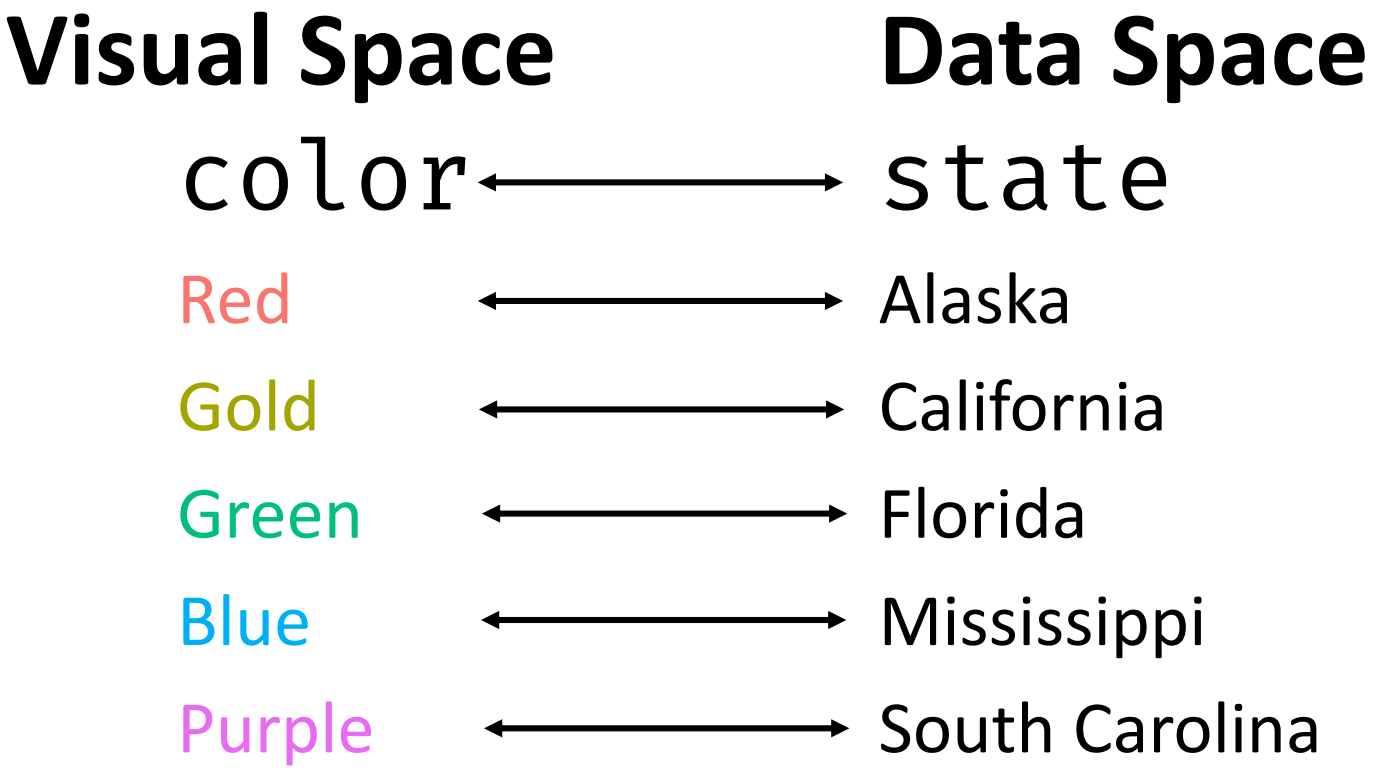
So, what are mappings?

Aesthetics are the visual properties of the objects in your plot.



Mappings describe how aesthetics should *relate to variables* in the data.

Mappings describe how aesthetics should *relate to variables* in the data.



Aesthetic property

Variable to map to

```
ggplot(data = wq) +
  geom_point(mapping = aes(x = temp, y = do_mgl, color = state))

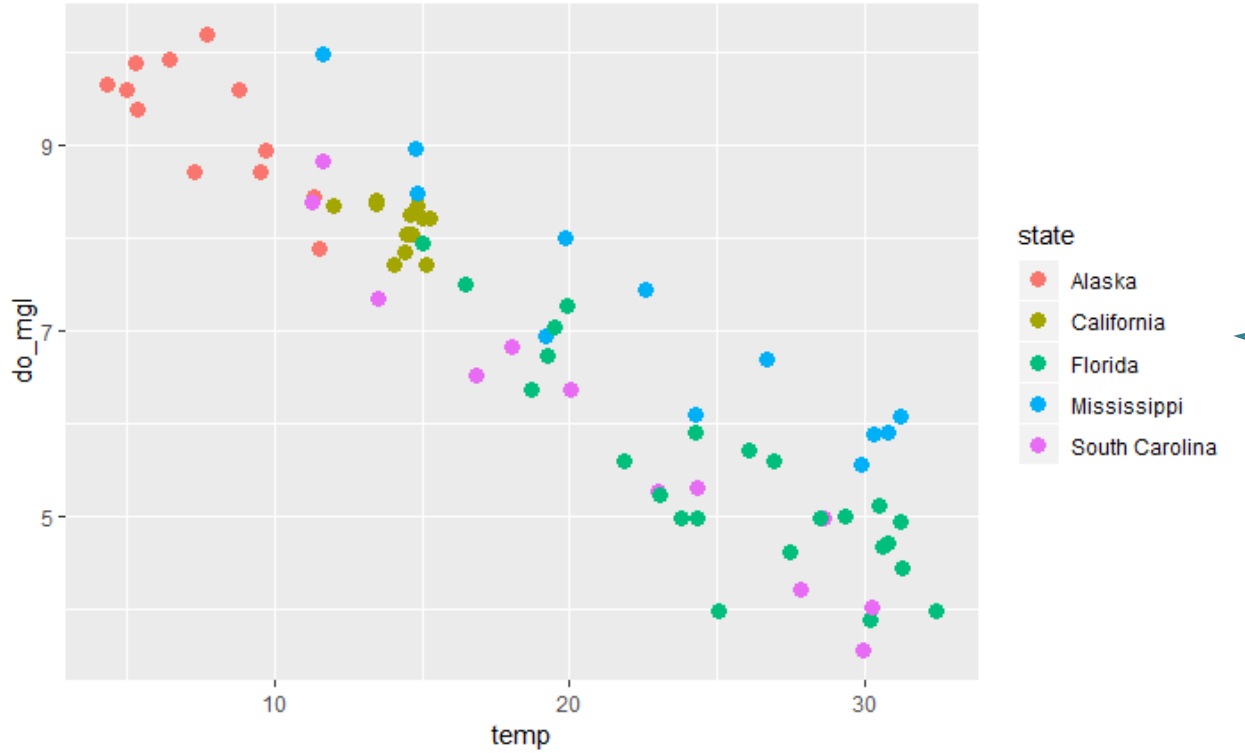
ggplot(data = wq) +
  geom_point(mapping = aes(x = temp, y = do_mgl, size = state))

ggplot(data = wq) +
  geom_point(mapping = aes(x = temp, y = do_mgl, shape = state))
```

Aesthetic property

Variable to map to

```
ggplot(data = wq) +
  geom_point(mapping = aes(x = temp, y = do_mgl, color = state))
```



Legends are automatically added

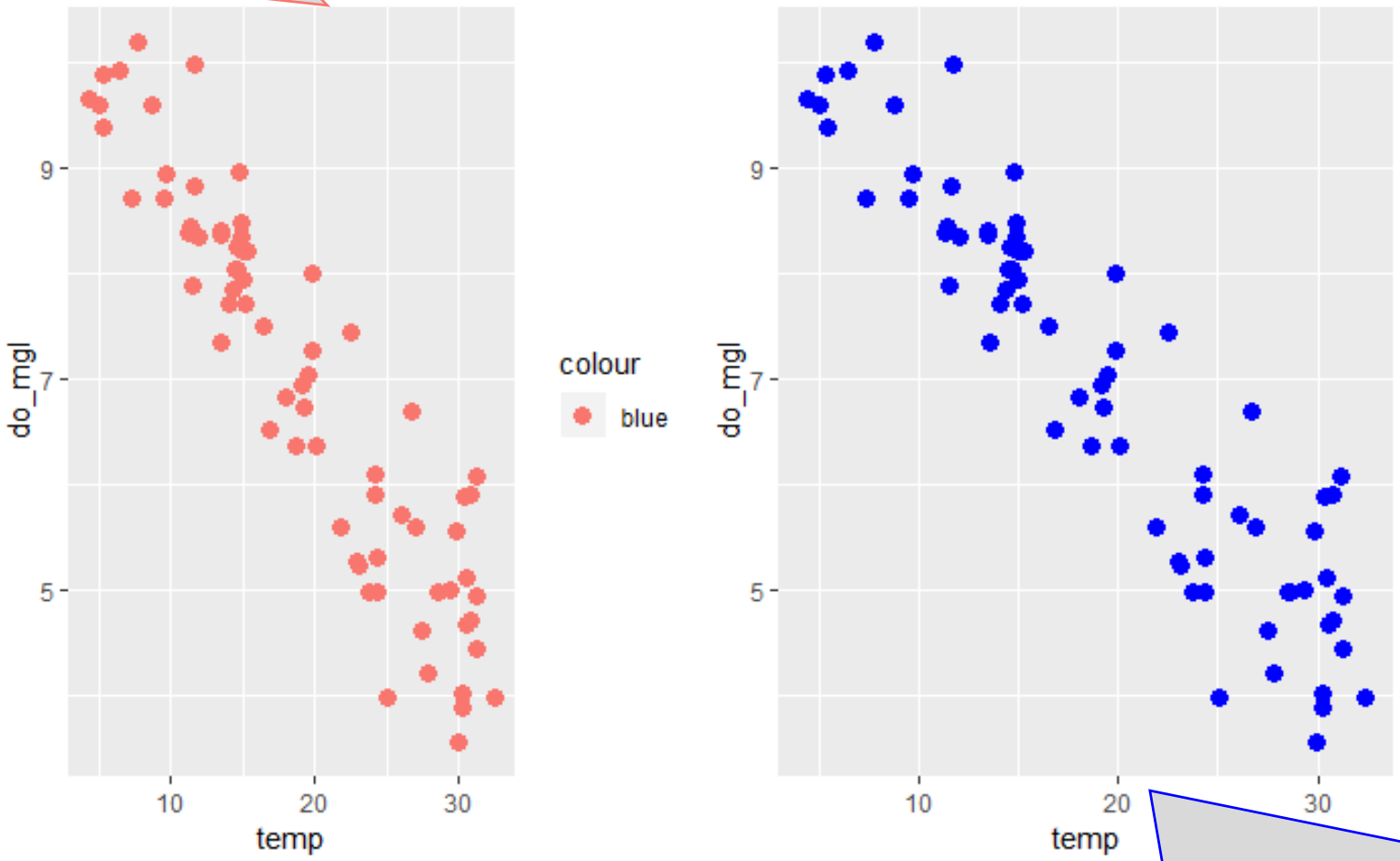
You can also ***set*** the aesthetic properties of your geom manually.

```
ggplot(wq) +
     geom_point(aes(x = temp, y = do_mgl, color = “blue”))
```

vs

```
ggplot(wq) +
     geom_point(aes(x = temp, y = do_mgl), color = “blue”)
```

```
ggplot(wq) +
       geom_point(aes(x = temp, y = do_mgl, color = “blue”))
```




```
ggplot(wq) +
       geom_point(aes(x = temp, y = do_mgl), color = “blue”)
```

TL;DR

- **Inside of aes():** maps an aesthetic to a variable

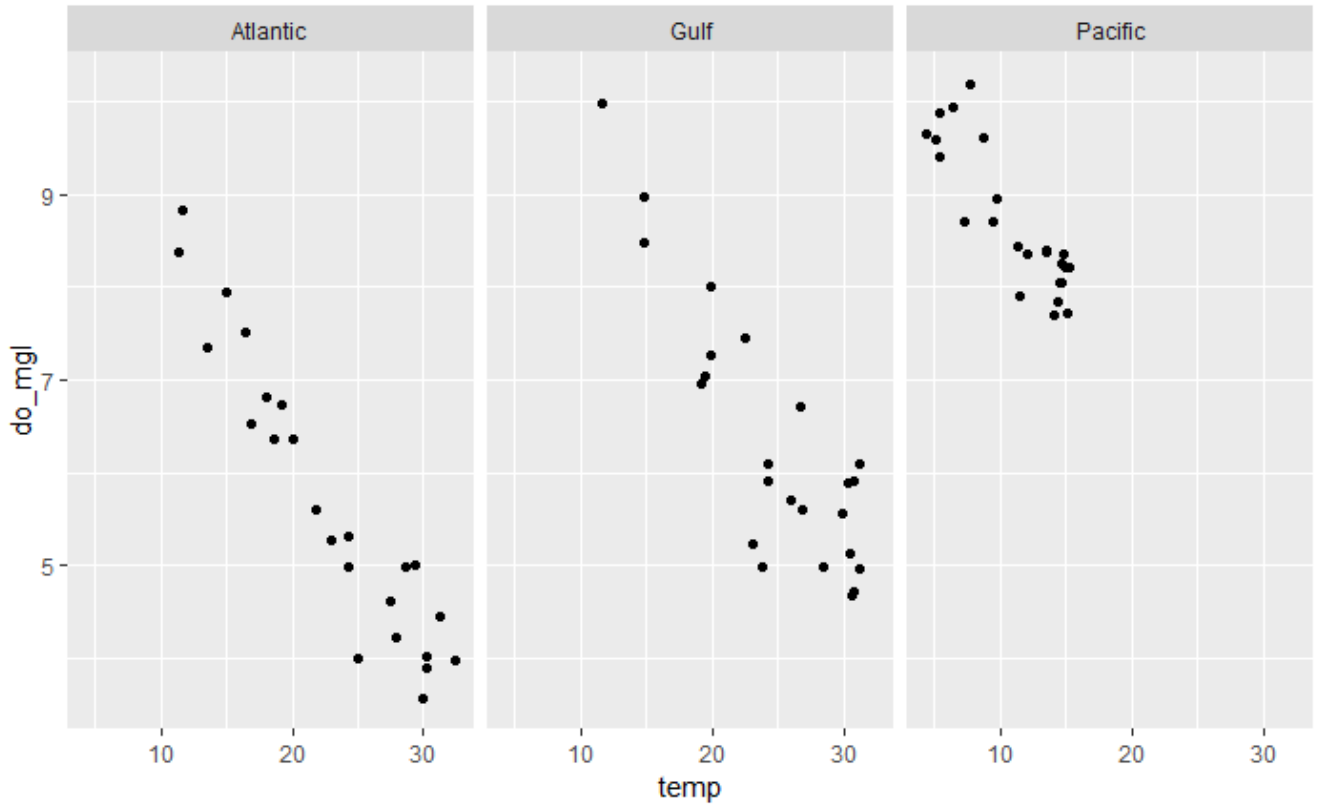
```
ggplot(wq) +
    geom_point(aes(x = temp, y = do_mgl, color = state))
```

- **Outside of aes():** sets an aesthetic to a value

```
ggplot(wq) +
    geom_point(aes(x = temp, y = do_mgl), color = “blue”)
```

What if I don't want to add additional variables as aesthetics?

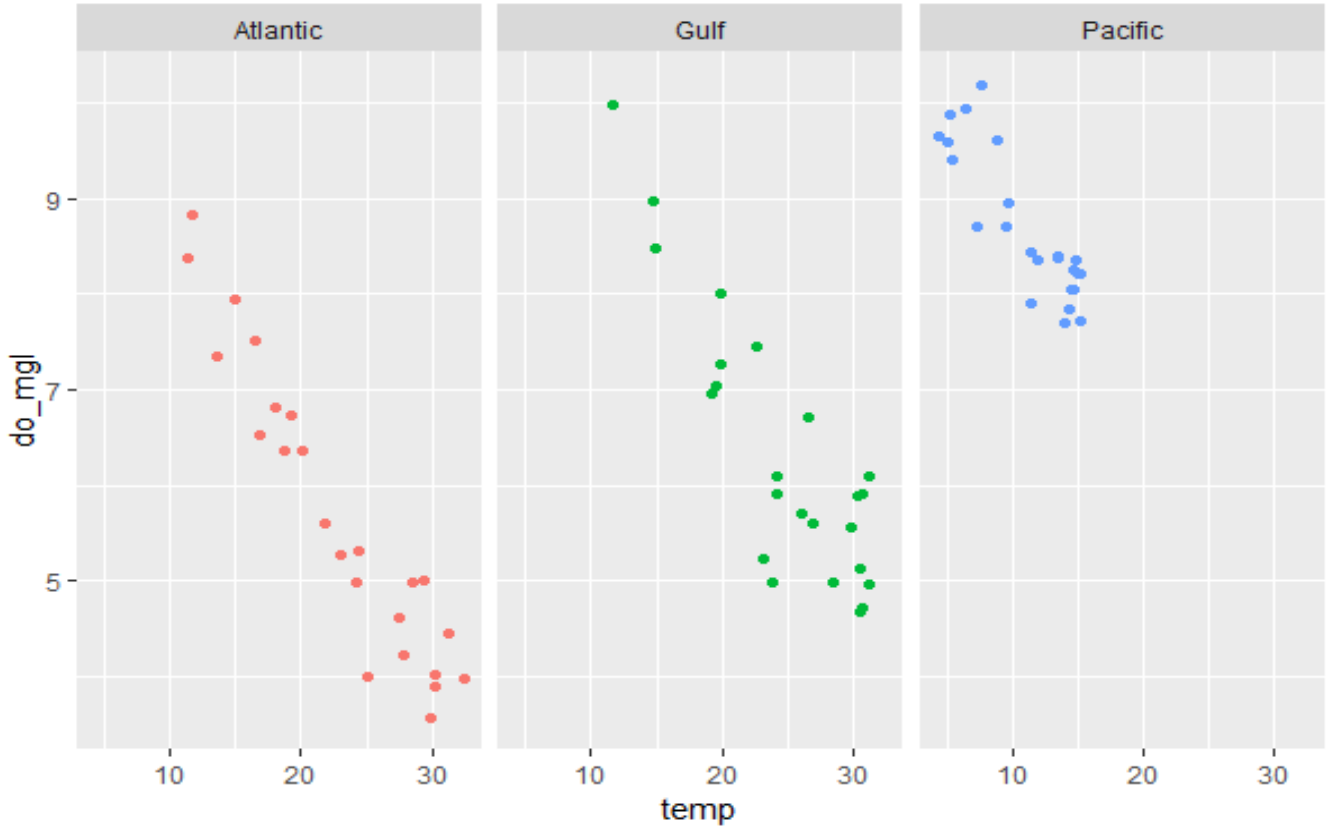
If they are categorical variables, you can use **facets**!



Subplots that each display one subset of the data

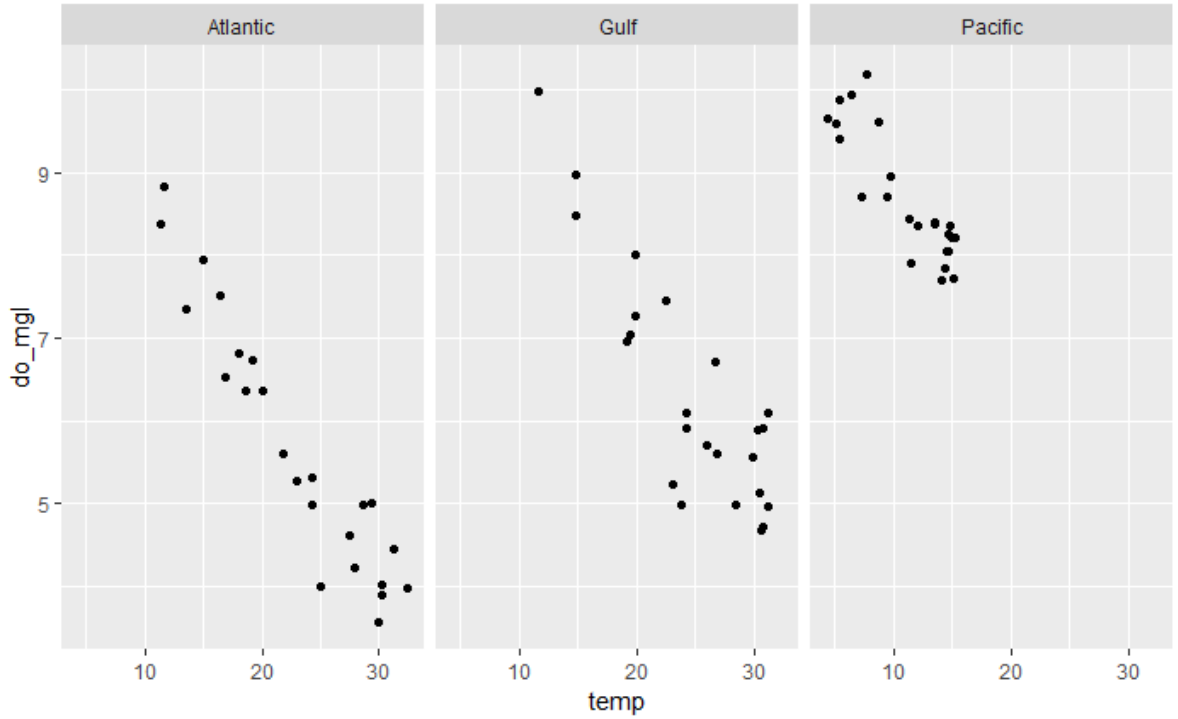
What if I don't want to add additional variables as aesthetics?

If they are categorical variables, you can use **facets**!



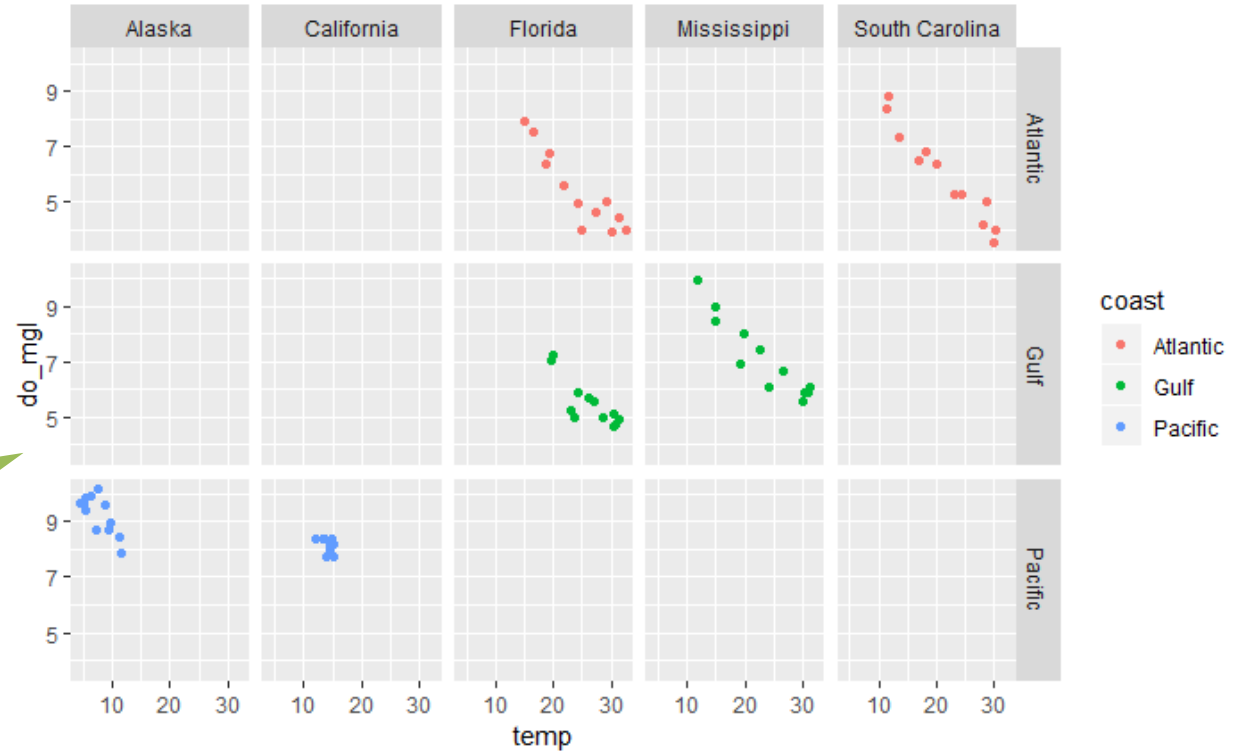
Subplots that each display one subset of the data

`facet_wrap()` ← for a single variable




```
ggplot(data = wq) +
  geom_point(mapping = aes(x = temp, y = do_mgl)) +
  facet_wrap(~ coast, nrow = 1)
```

`facet_grid()` ← for a combo of 2 variables



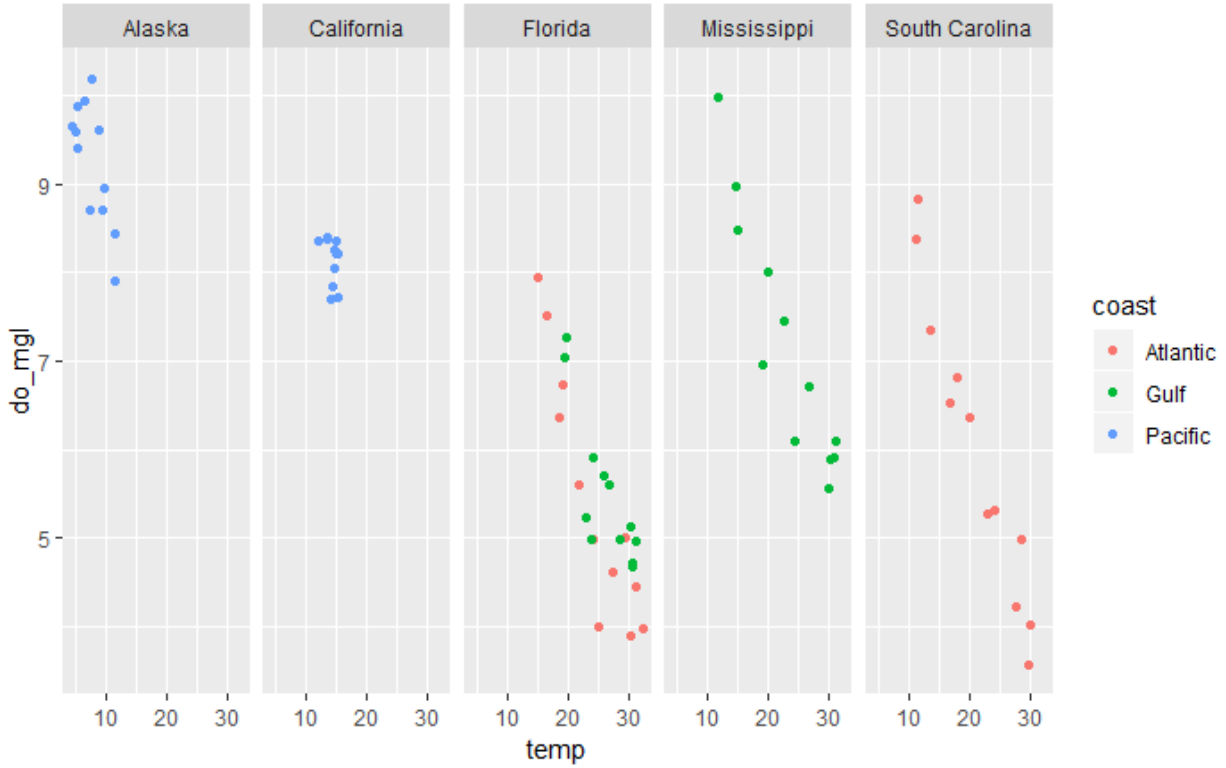
Sigh…it's a bad graph, I know!

```
ggplot(data = wq) +
  geom_point(mapping = aes(x = temp, y = do_mgl)) +
  facet_grid(coast ~ state)
```

`facet_grid()` ← for a combo of 2 variables

```
ggplot(data = wq) +
  geom_point(mapping = aes(x = temp, y = do_mgl)) +
  facet_grid(. ~ state)
```

Use this in place if you would prefer not to facet in the rows or columns dimension



YOUR TURN 2a and 2b

2a. In the next chunk, add color, size, alpha, and shape aesthetics to your graph. Play around with the data!

What happens when you map aesthetics to discrete and continuous variables?
What happens when you apply more than one aesthetic?

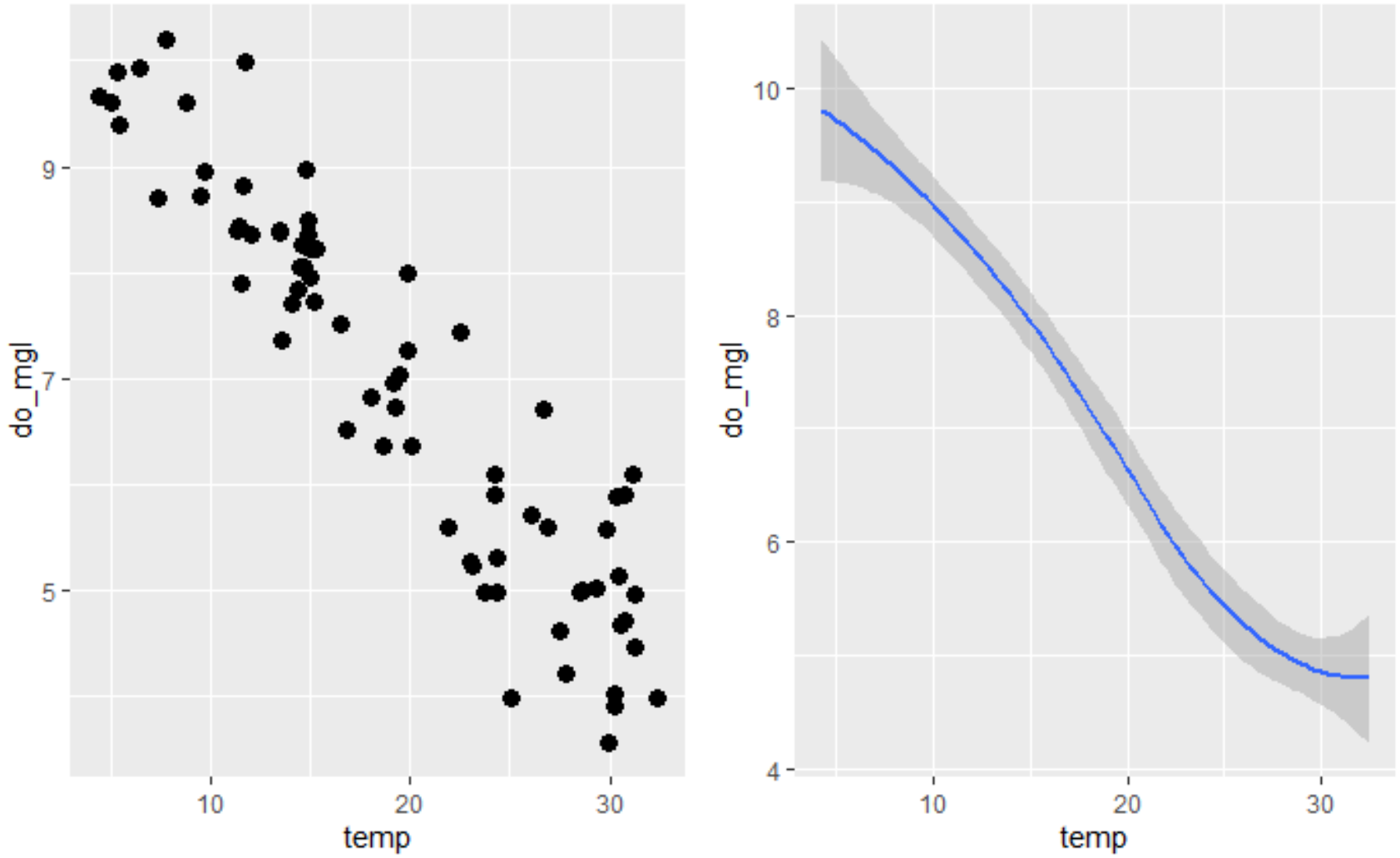
2b. Using the `wq` data, plot temperature (`temp`) by `month` as a scatterplot and facet by `state` and `coast`.

Play around with `facet_grid()` and `facet_wrap()`.

What kinds of things do you notice?

Geoms

How are these plots similar? Different?

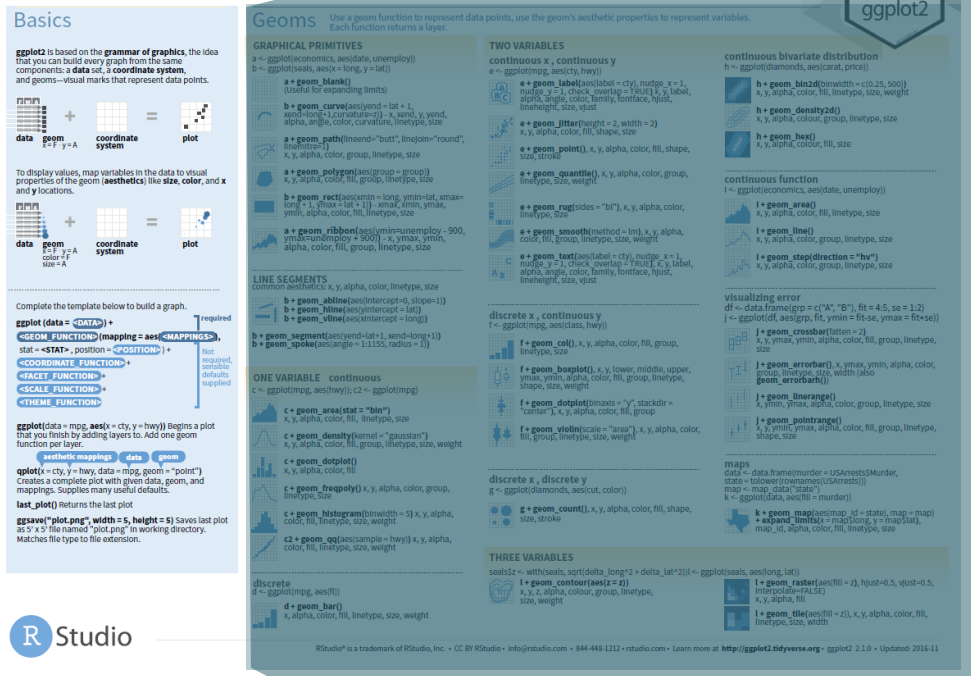


geoms

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

Every geom requires a mapping argument

Data Visualization with ggplot2 : : CHEAT SHEET



Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

data + **geom** (x = F, y = A) = **coordinate system** + **plot**

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

data + **geom** (x = F, y = A, color = F, size = A) = **coordinate system** + **plot**

Complete the template below to build a graph.

required

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

Not required, sensible defaults supplied

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings | data | geom

qplot(x = cty, y = hwy, data = mpg, geom = "point") Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", **width = 5, height = 5**) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

RStudio

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z)) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1)
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group))
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend=lat+1, xend=long+1))
b + geom_spoke(aes(angle = 1:1155, radius = 1))

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5) x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)) x, y, alpha, color, fill, linetype, size, weight

discrete

```
d <- ggplot(mpg, aes(fl))
```

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

```
e <- ggplot(mpg, aes(cty, hwy))
```

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"), x, y, alpha, color, linetype, size

e + geom_smooth(method = lm), x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

```
f <- ggplot(mpg, aes(class, hwy))
```

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"), x, y, alpha, color, fill, group

f + geom_violin(scale = "area"), x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

```
g <- ggplot(diamonds, aes(cut, color))
```

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

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

j + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

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

k + 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))l <- ggplot(seals, aes(long, lat))
```

l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

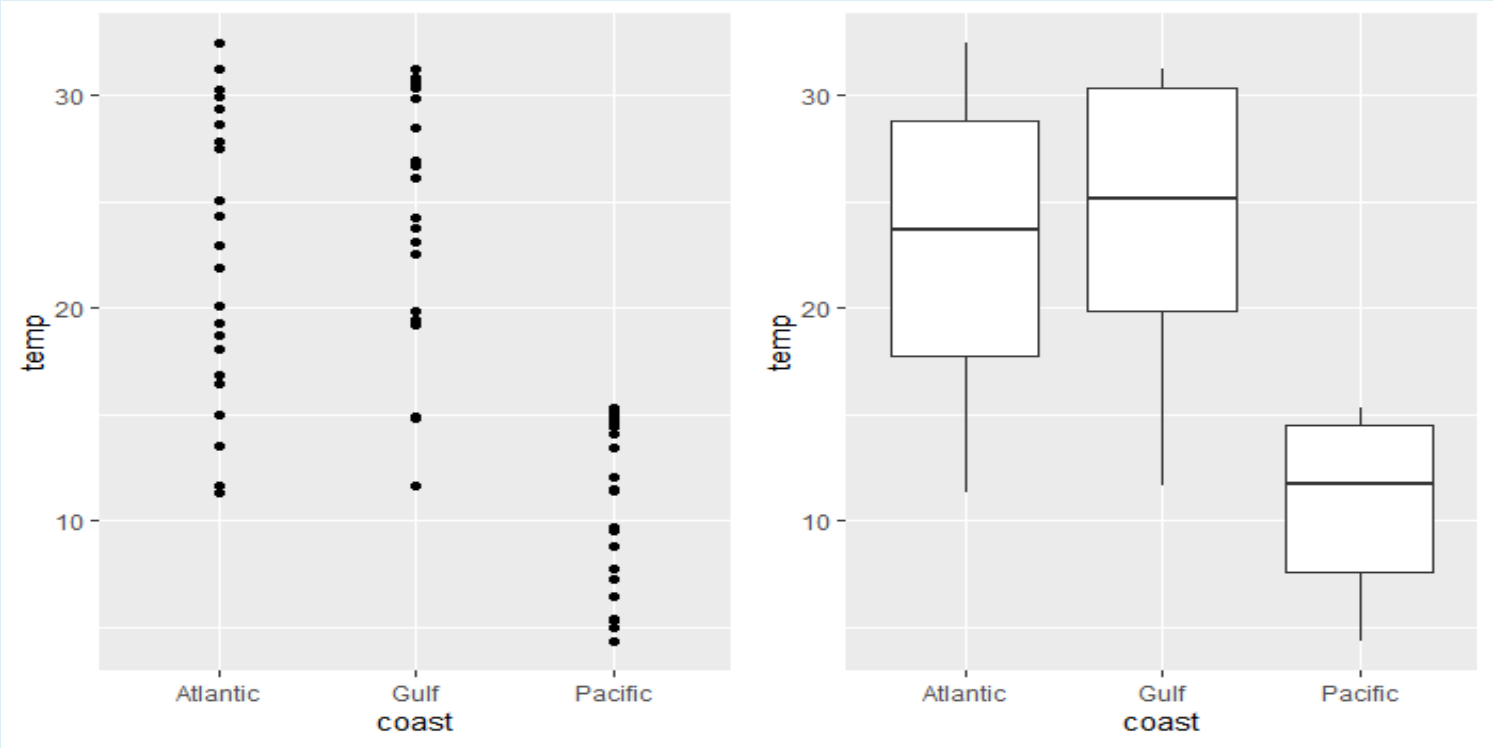
l + geom_tile(aes(fill = z)), x, y, alpha, color, fill, linetype, size, width

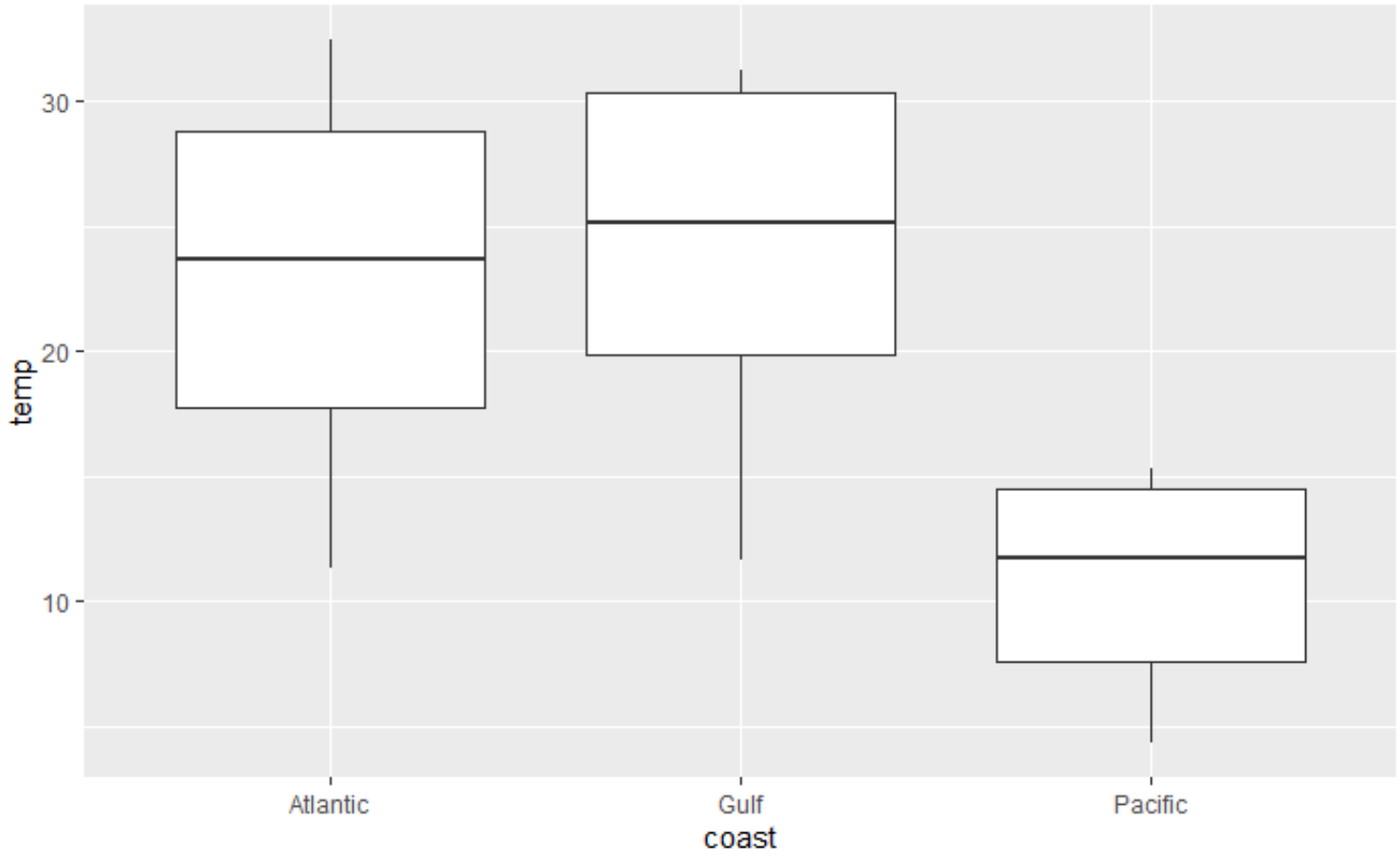
RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **http://ggplot2.tidyverse.org** • ggplot2 2.1.0 • Updated: 2016-11

You can also find the supported aesthetics by visiting the ?help for each geom!

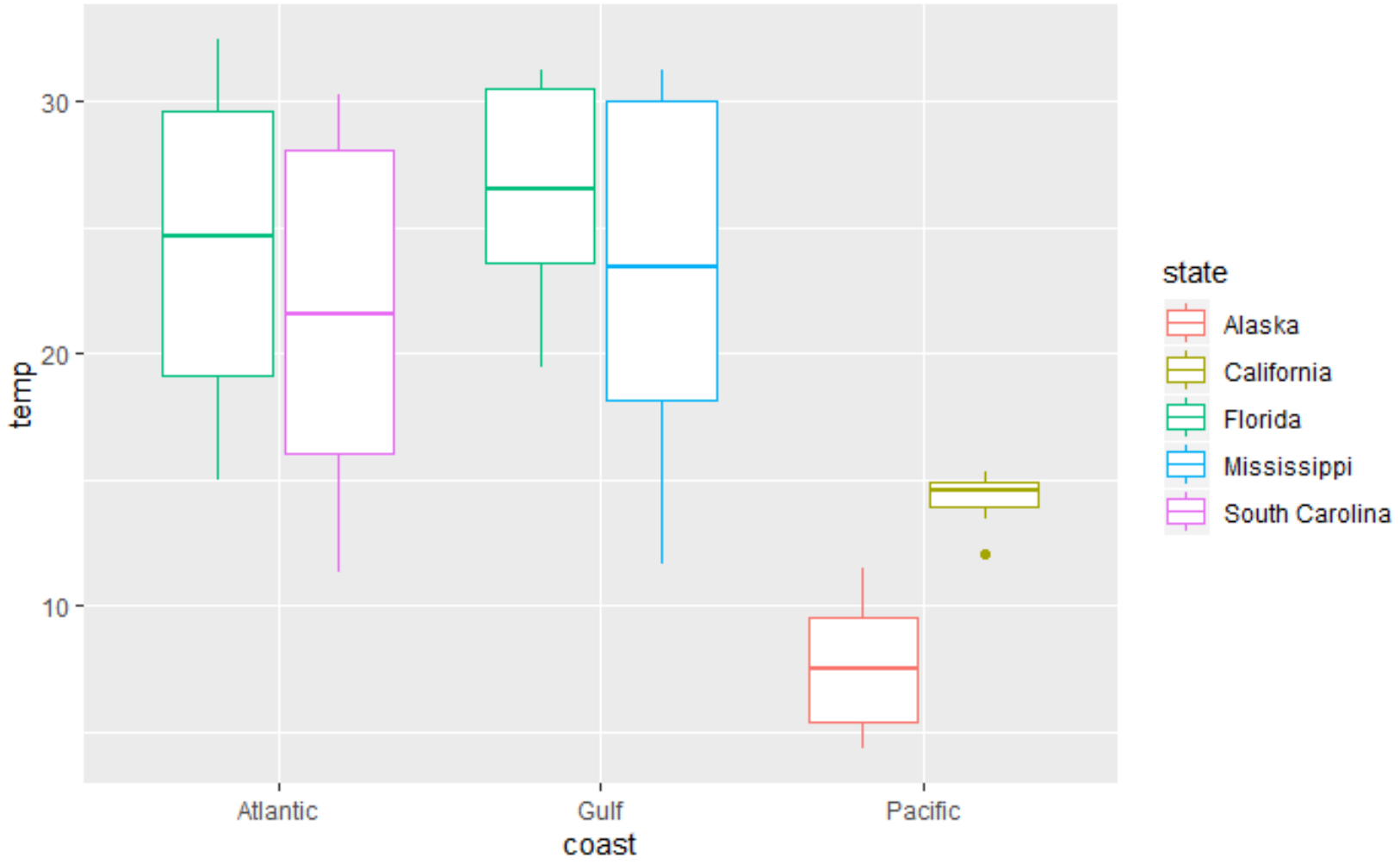
YOUR TURN 3

How do you think you would replace the scatterplot with one that draws boxplots? Use the cheatsheet and submit your best guess!






```
ggplot(data = wq) +
    geom_boxplot(mapping = aes(x = coast, y = temp))
```

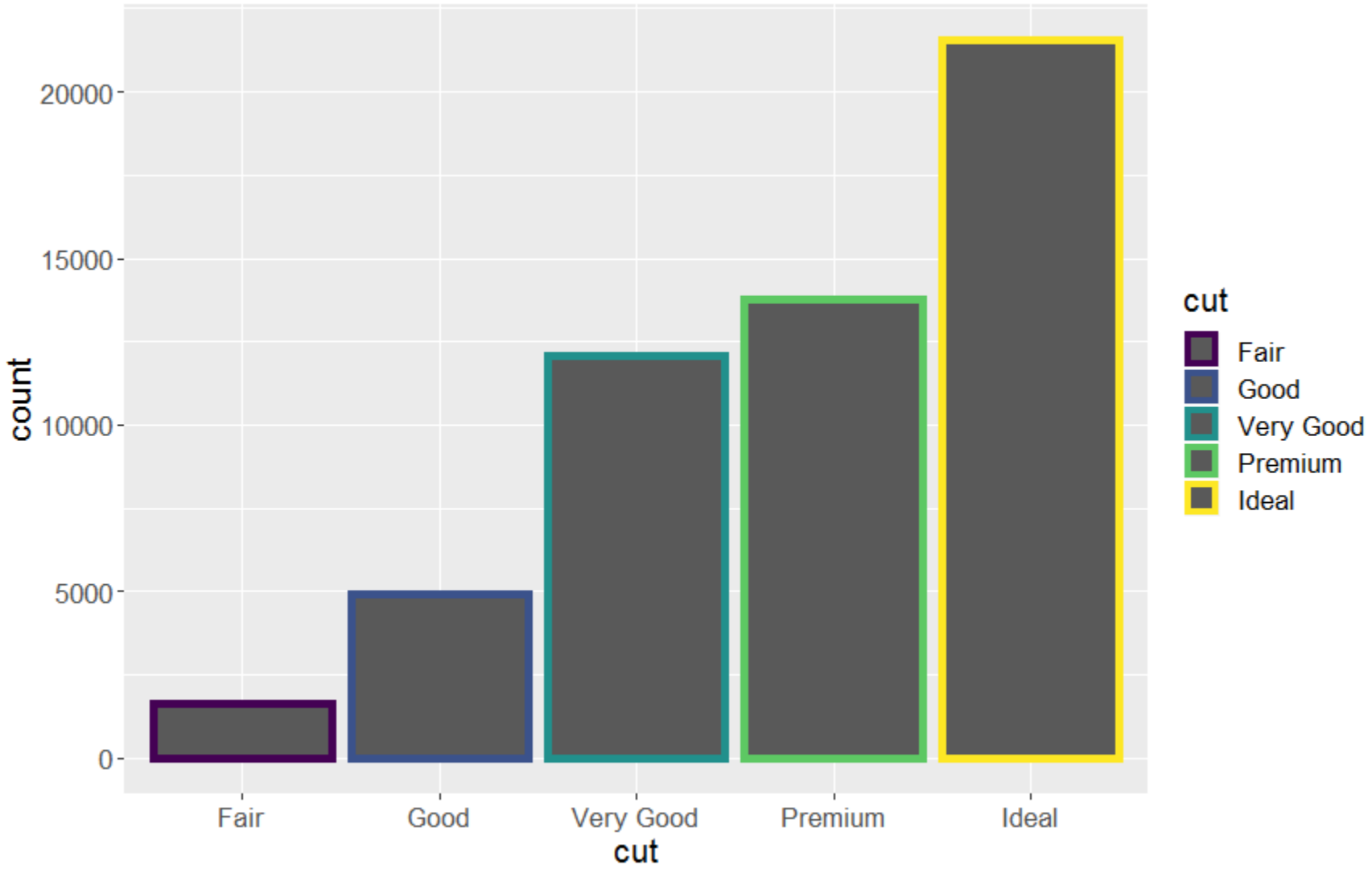



```
ggplot(data = wq) +
    geom_boxplot(aes(x = coast, y = temp, color = state))
```

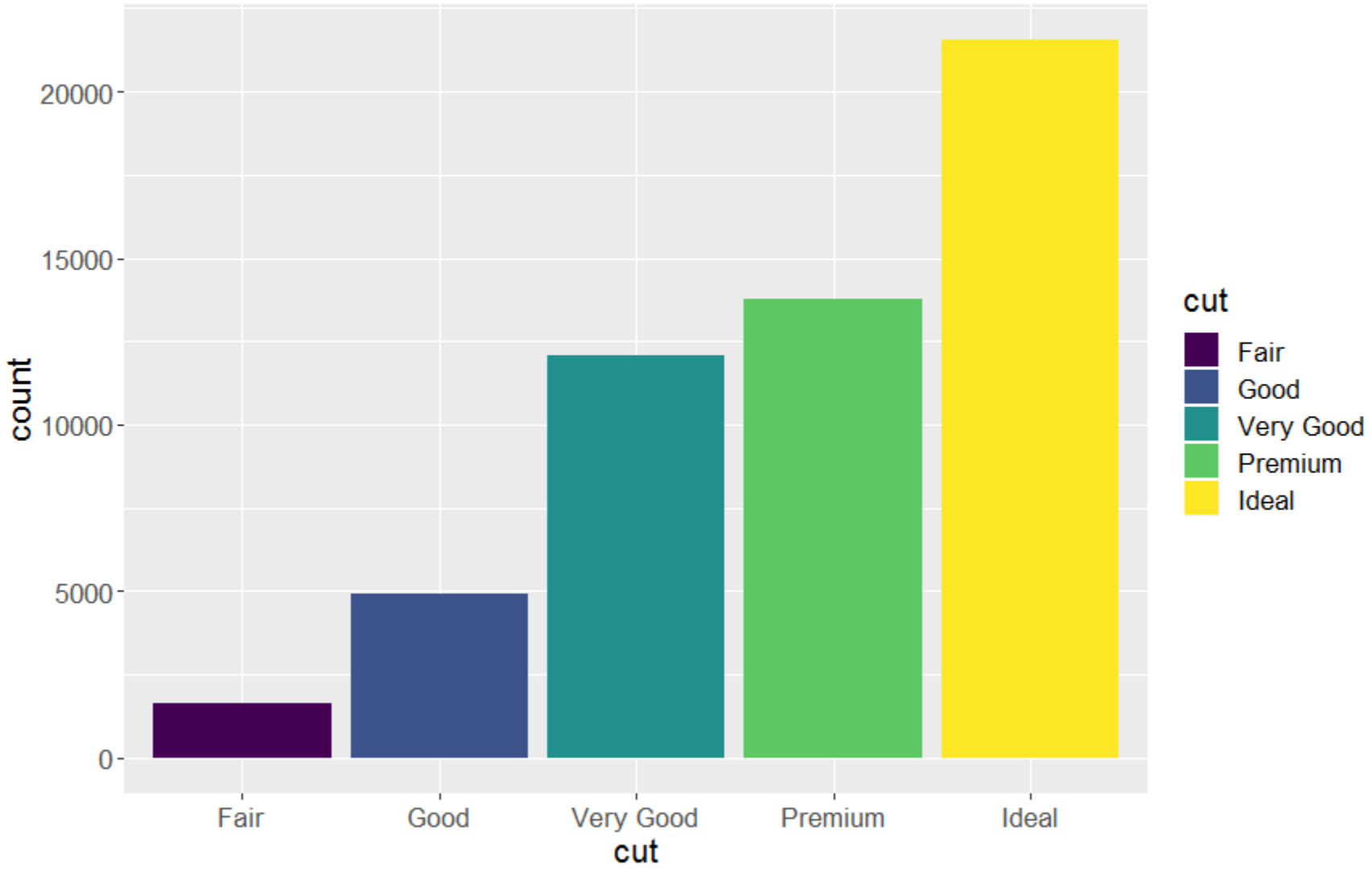
YOUR TURN 4

Load the `diamonds` dataset and examine it. This dataset is provided in the `ggplot2` package.

Create a bar chart of `cut` colored by `cut`. Use the cheatsheet.




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

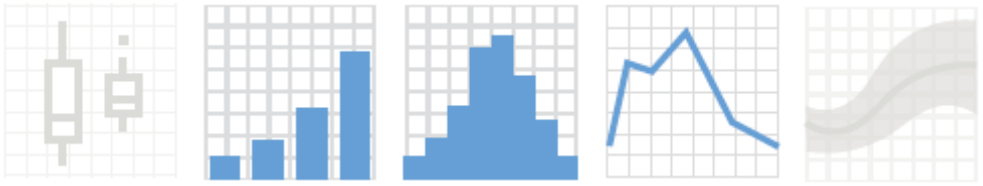



```
ggplot(data = diamonds) +
    geom_bar(mapping = aes(x = cut, fill = cut)
```

Hold up…count is not a variable in `diamonds`!

Many graphs plot the raw values of your dataset while others **calculate** new values to plot:

- Bar charts, histograms, and frequency polygons bin data and plot bin counts (# of points that fall into each bin)



Hold up…count is not a variable in `diamonds`!

Many graphs plot the raw values of your dataset while others **calculate** new values to plot:

- Smoothers, `geom_smooth()`, fit a model to your data and plot the predictions from the model

Hold up…count is not a variable in `diamonds`!

Many graphs plot the raw values of your dataset while others **calculate** new values to plot:

- Boxplots compute a summary of the distribution and display it in a formatted box

Hold up…count is not a variable in `diamonds`!

1. **geom_bar()** begins with the **diamonds** data set

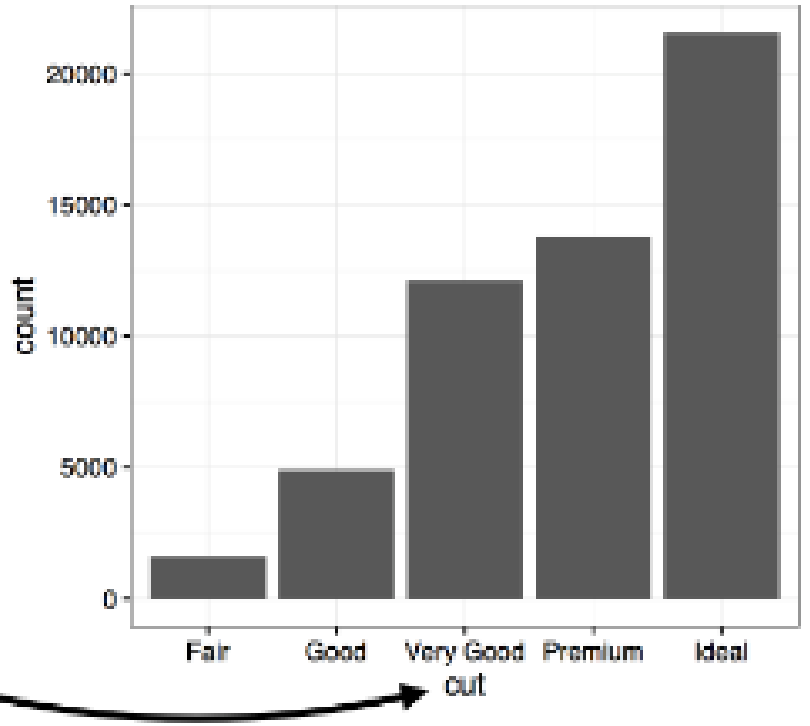
2. **geom_bar()** transforms the data with the "count" stat, which returns a data set of cut values and counts.

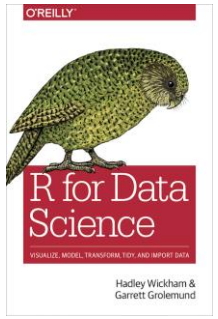
3. **geom_bar()** uses the transformed data to build the plot. cut is mapped to the x axis, count is mapped to the y axis.

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	Premium	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	Premium	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

Short for “statistical transformation” - the algorithm that is used to create new values for a graph.

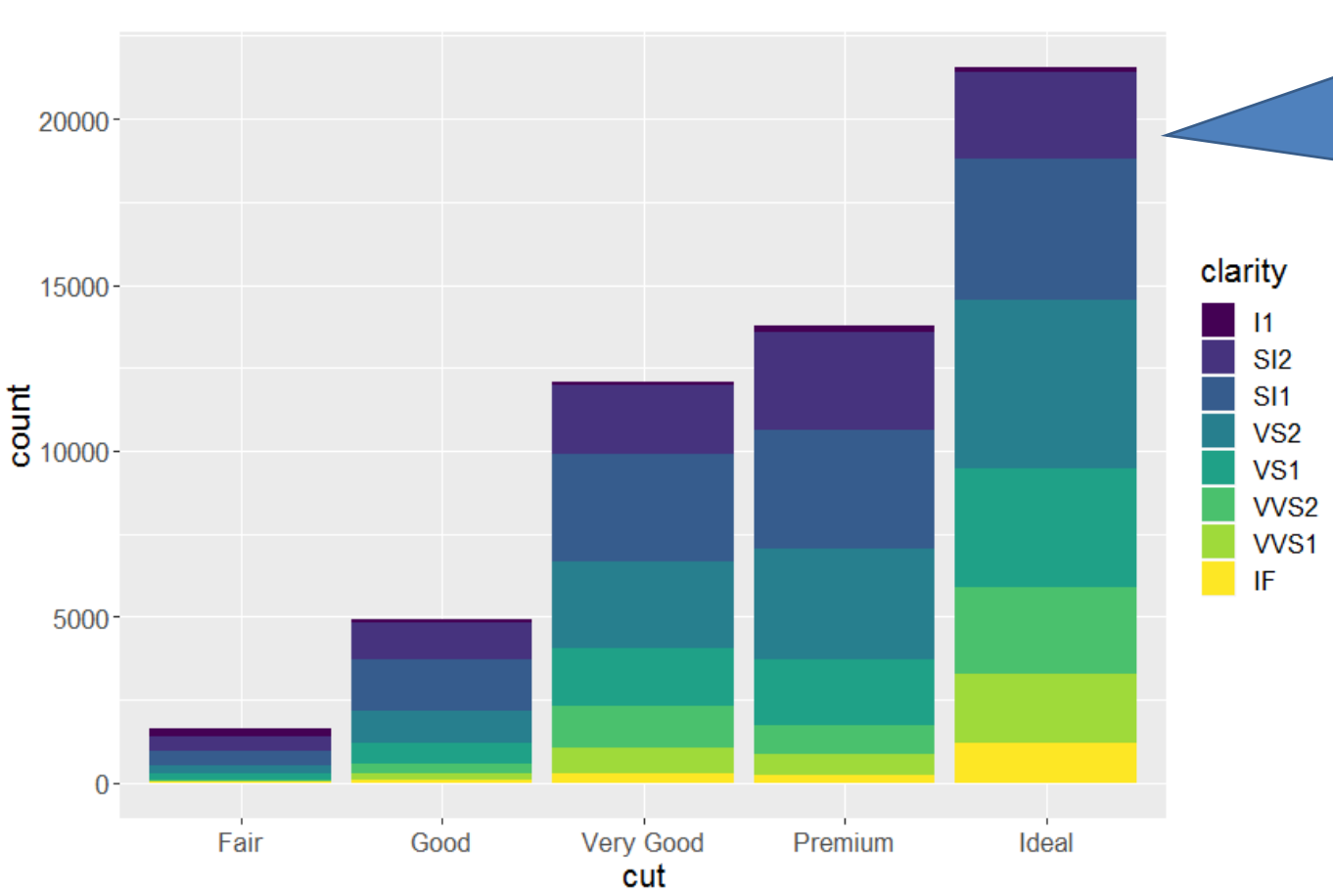
One of the five components of each layer!

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

Learn what stat a geom uses by inspecting the default value for the stat argument.
Ex: `?geom_bar`

Position: how overlapping objects are arranged

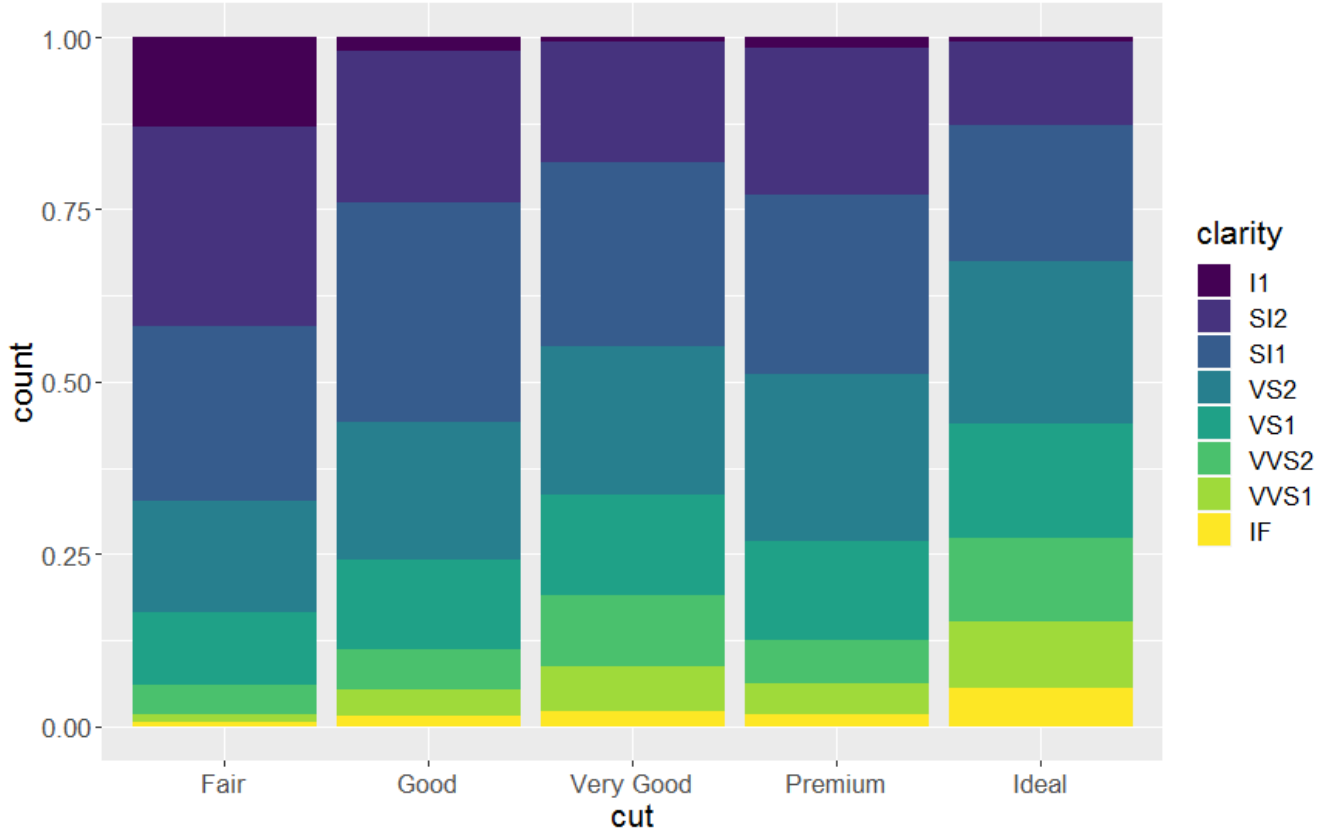
```
ggplot(data = diamonds) +
    geom_bar(mapping = aes (x = cut, fill = clarity))
```



Stacking is automatic and the default position with `geom_bar`

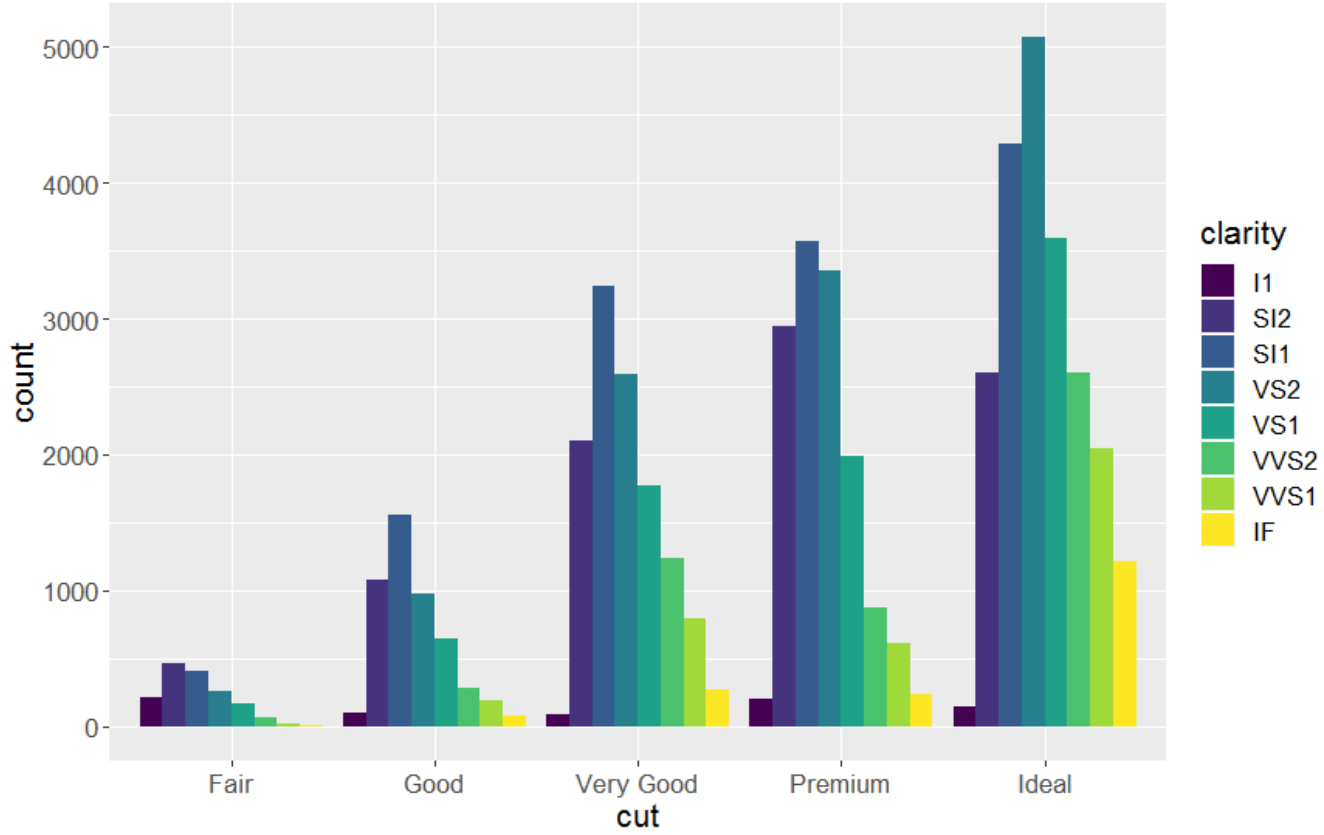
Position: how overlapping objects are arranged

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



Position: how overlapping objects are arranged

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



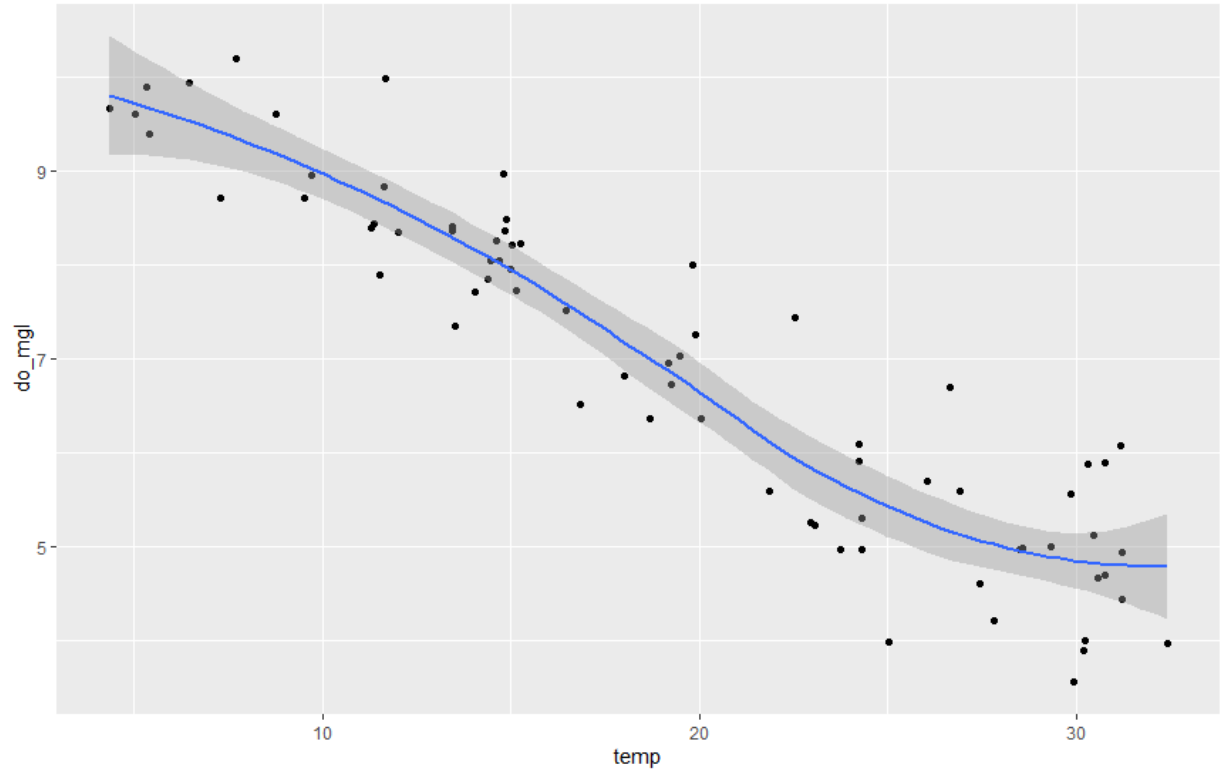
YOUR TURN 5

Predict what this code will do, then run it.

```
ggplot(wq) +
  geom_point(aes(temp, do_mgl)) +
  geom_smooth(aes(temp, do_mgl))
```

Global vs. Local

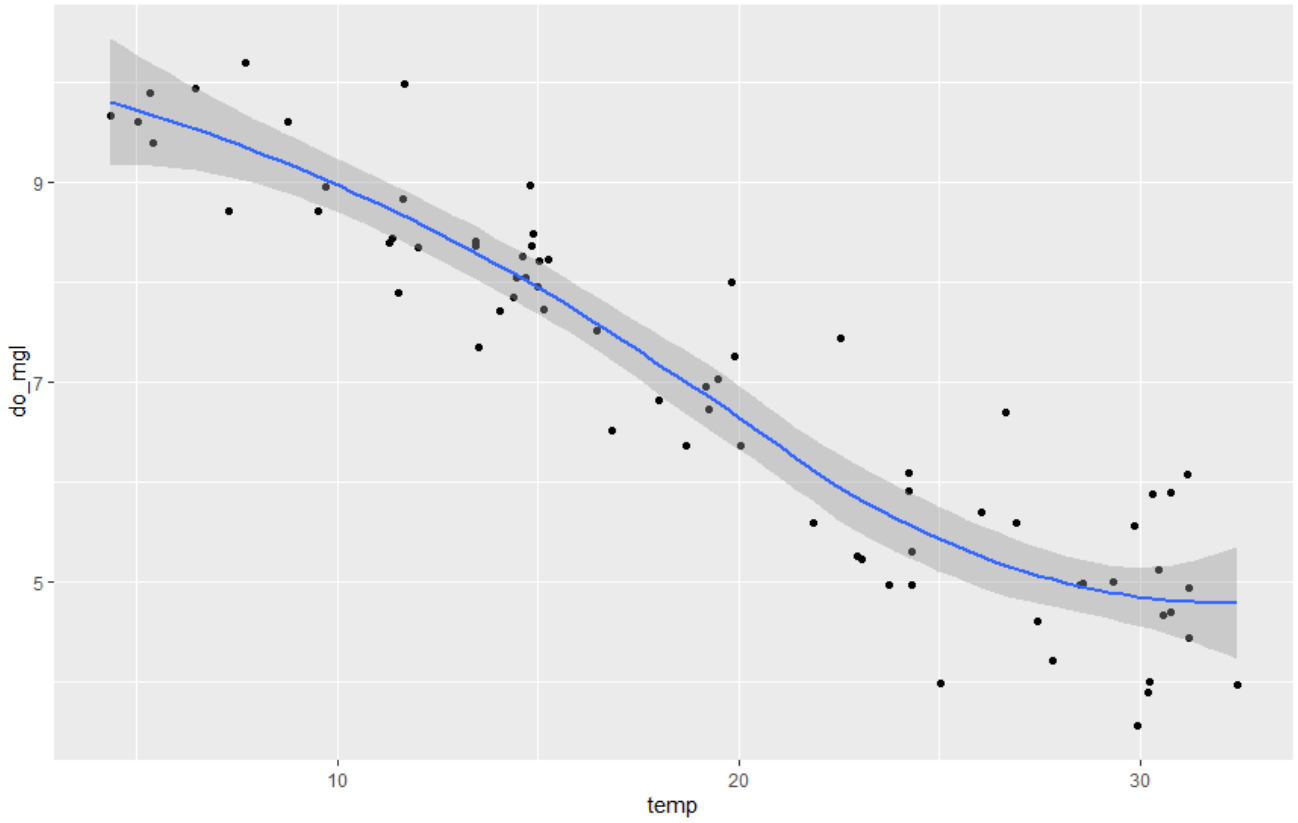
```
ggplot(wq) +
      geom_point(aes(x = temp, y = do_mgl)) +
      geom_smooth(aes(x = temp, y = do_mgl))
```



Global vs. Local

```
ggplot(data = wq, mapping = aes(x = temp, y = do_mgl)) +
      geom_point() +
      geom_smooth()
```

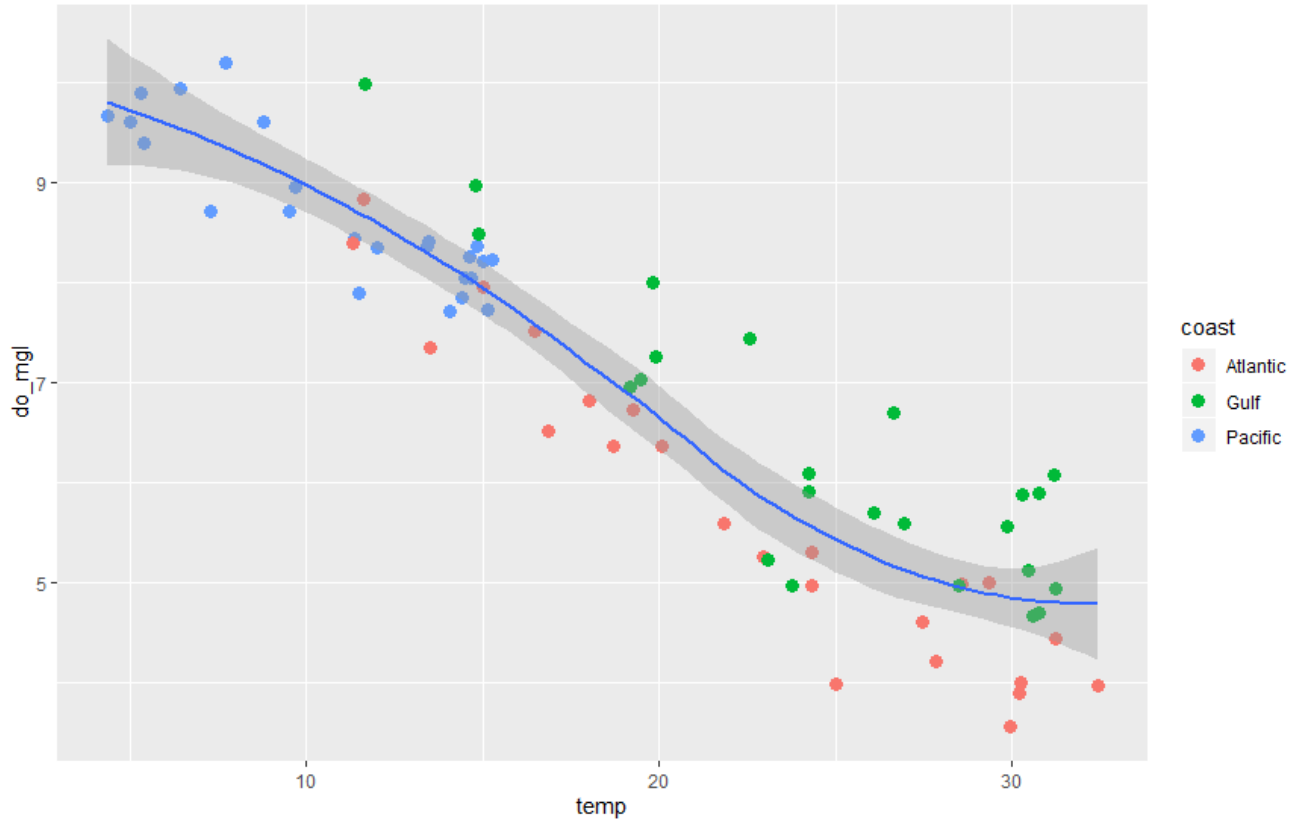
Mappings and data that appear in `ggplot()` will apply globally to each layer



Global vs. Local

```
ggplot(data = wq, mapping = aes(x = temp, y = do_mgl)) +
      geom_point(mapping = aes(color = coast)) +
      geom_smooth()
```

Mappings applied in geom_function will add or override the global mappings for that layer only



Let's recap. To make a graph…

1. Pick your **data** set

```
ggplot(data = <DATA>) +
```

Let's recap. To make a graph…

1. Pick your **data** set

```
ggplot(data = <DATA>) +
    <GEOM FUNCTION>()
```

2. Choose the best **geom** to display cases

Let's recap. To make a graph…

1. Pick your **data** set

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

2. Choose the best **geom** to display cases

3. **Map** aesthetic properties to variables.

Let's recap. To make a graph…

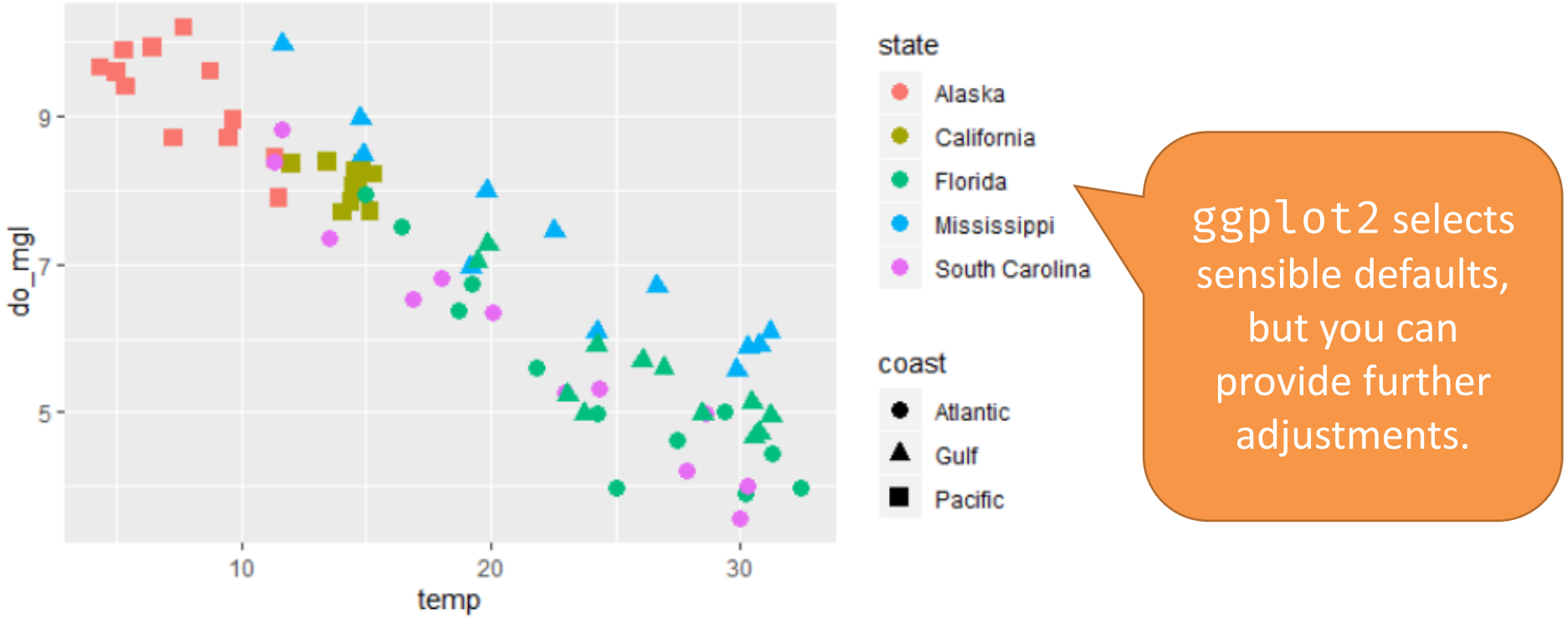
```
ggplot(data = <DATA>) +
    <GEOM FUNCTION>(mapping = aes(<MAPPINGS>),
                    stat = <STAT>,
                    position = <POSITION>)
```

Not required, but defaults are supplied by the specified geom

What else can I do?

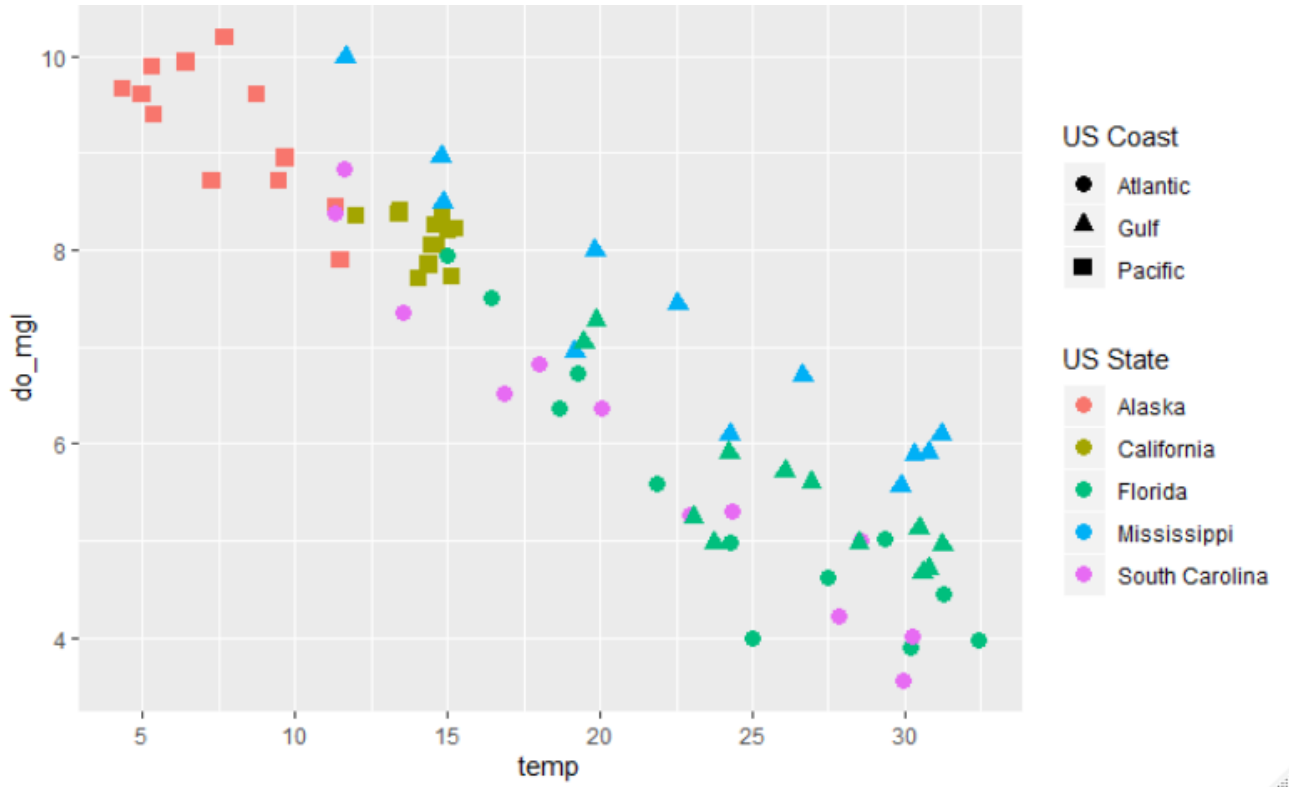
Scales, themes, and saving your plot.

Scales control the way your data is mapped to your geom and are required for every aesthetic.



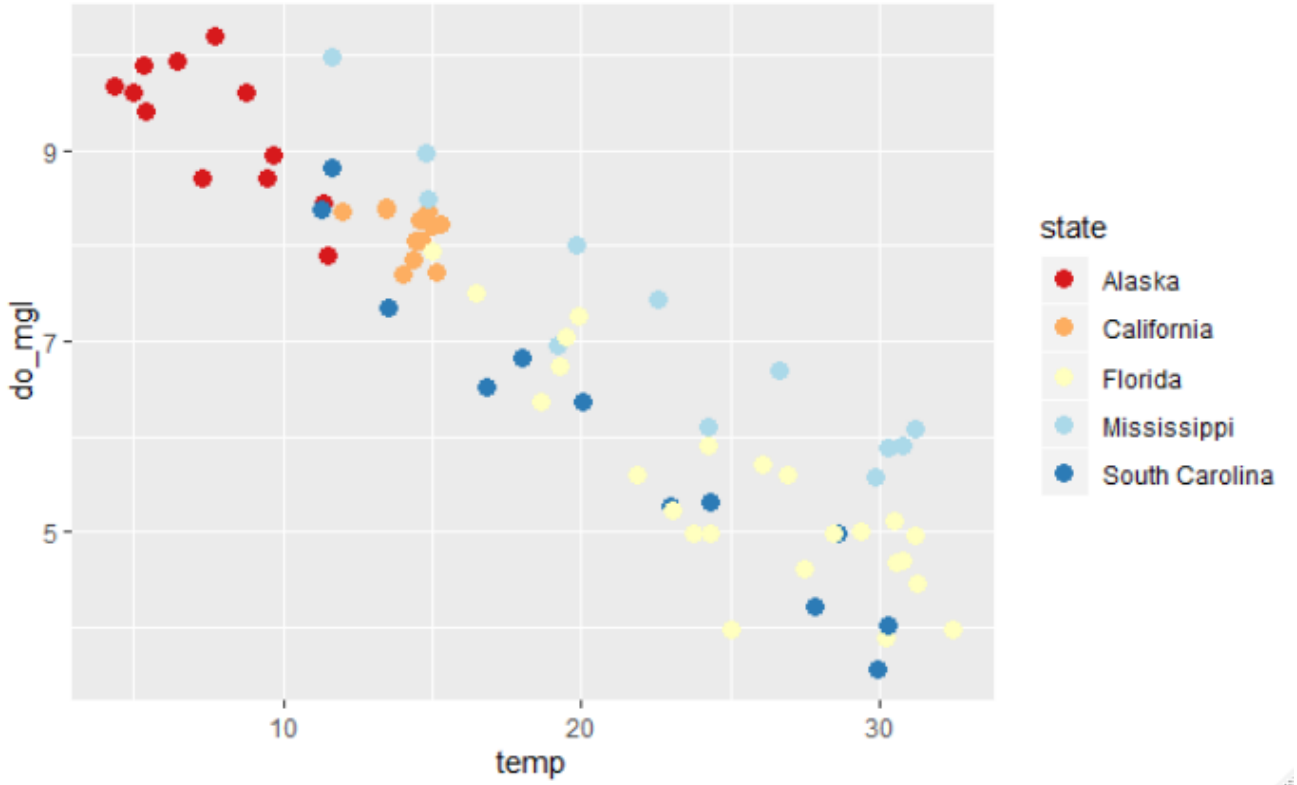

```
ggplot(wq, aes(x = temp, y = do_mgl)) +
  geom_point(aes(shape = coast, color = state), size = 3)
```

The specific scale function you use is dependent on the type of scale




```
ggplot(wq, aes(x = temp, y = do_mgl)) +
  geom_point(aes(shape = coast, color = state), size = 3) +
  scale_shape_discrete(name = "US Coast") +
  scale_color_discrete(name = "US State") +
  scale_y_continuous(breaks = c(2, 4, 6, 8, 10)) +
  scale_x_continuous(breaks = c(5, 10, 15, 20, 25, 30, 35))
```

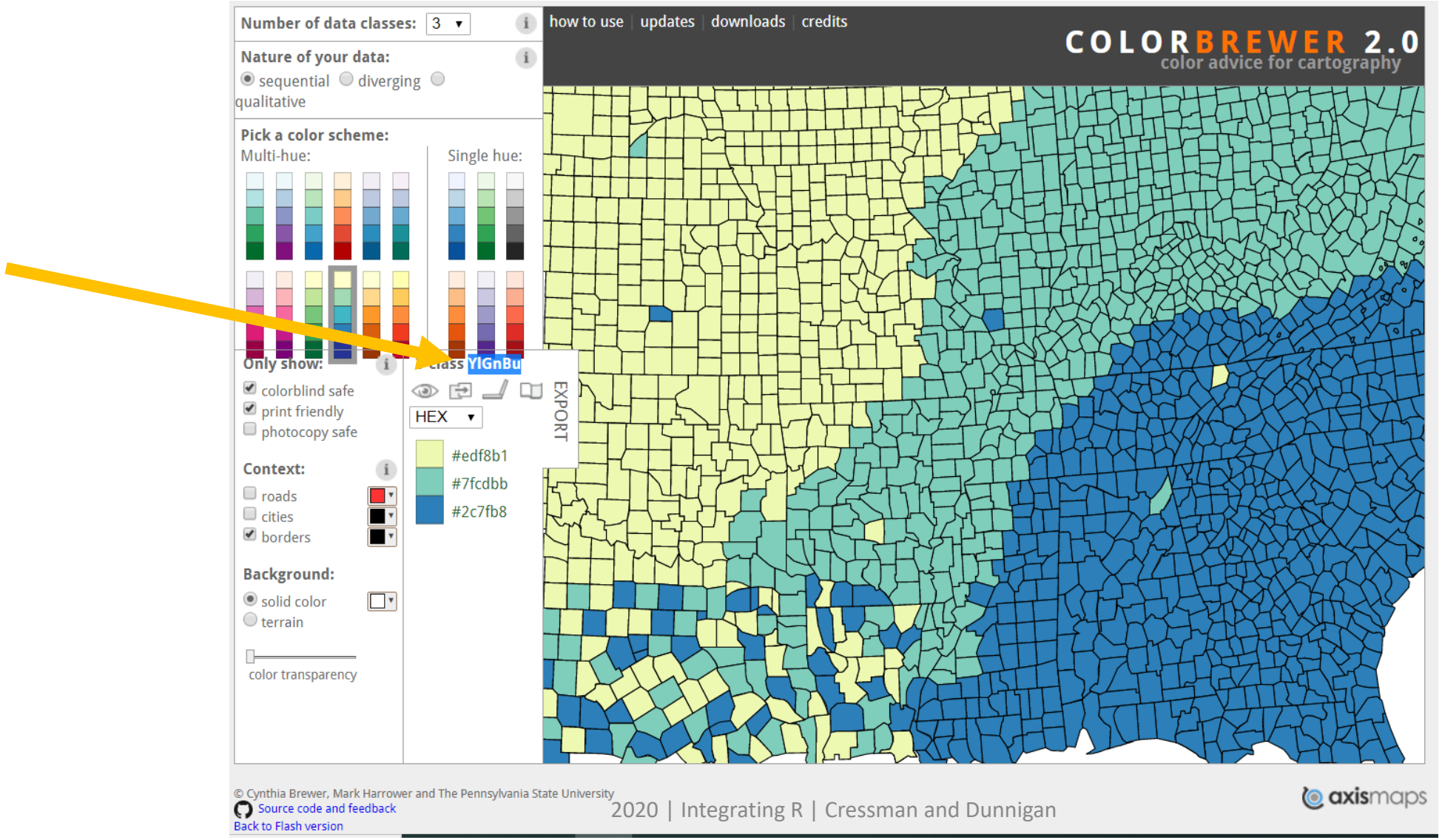
scale_color_brewer()

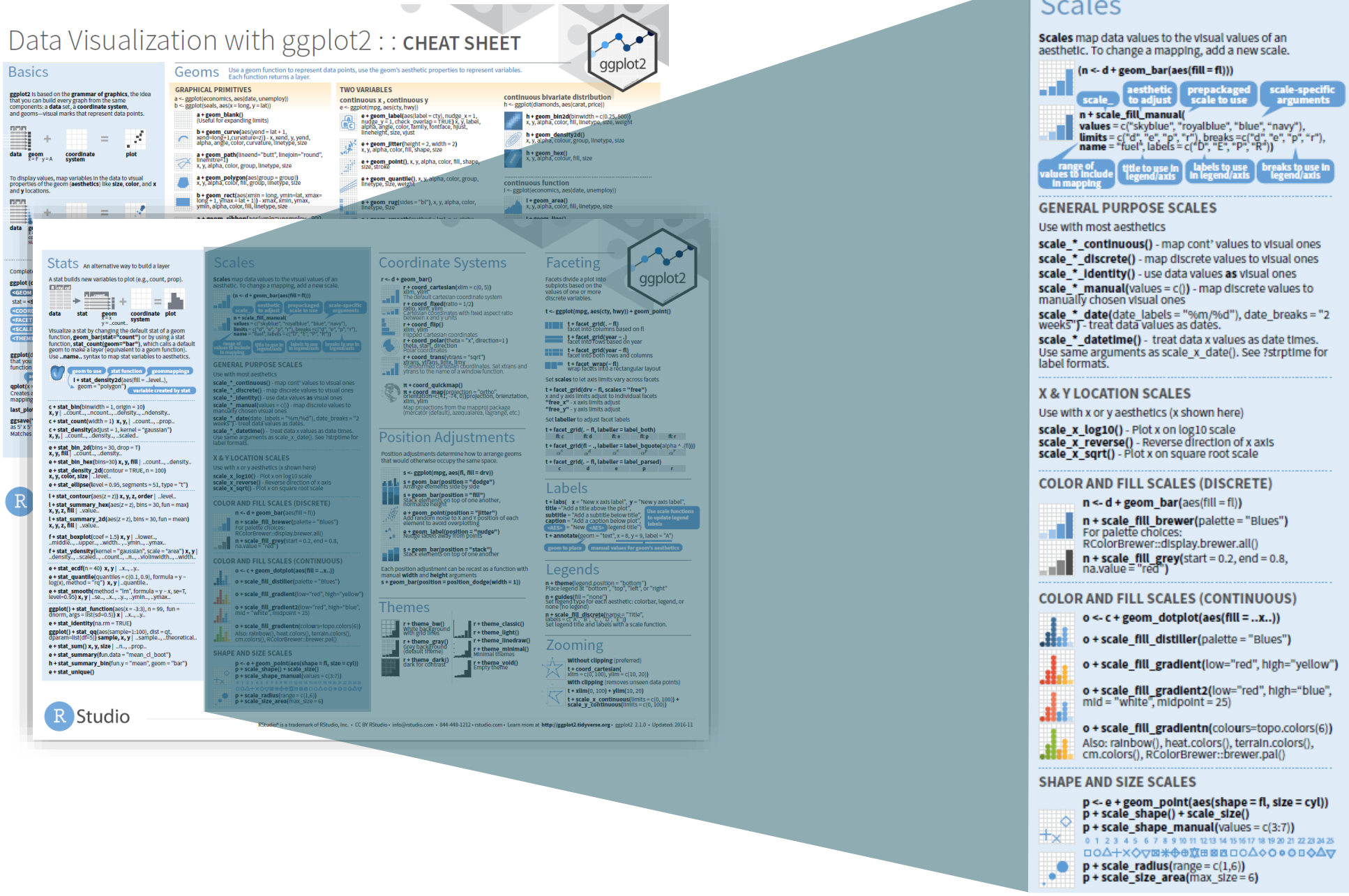


Visit colorbrewer2.org for more information!

```
ggplot(wq, aes(x = temp, y = do_mgl)) +
  geom_point(aes(color = state), size = 3) +
  scale_color_brewer(type = “div", palette = “RdYlBu")
```

www.colorbrewer2.org





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 & 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"**)**

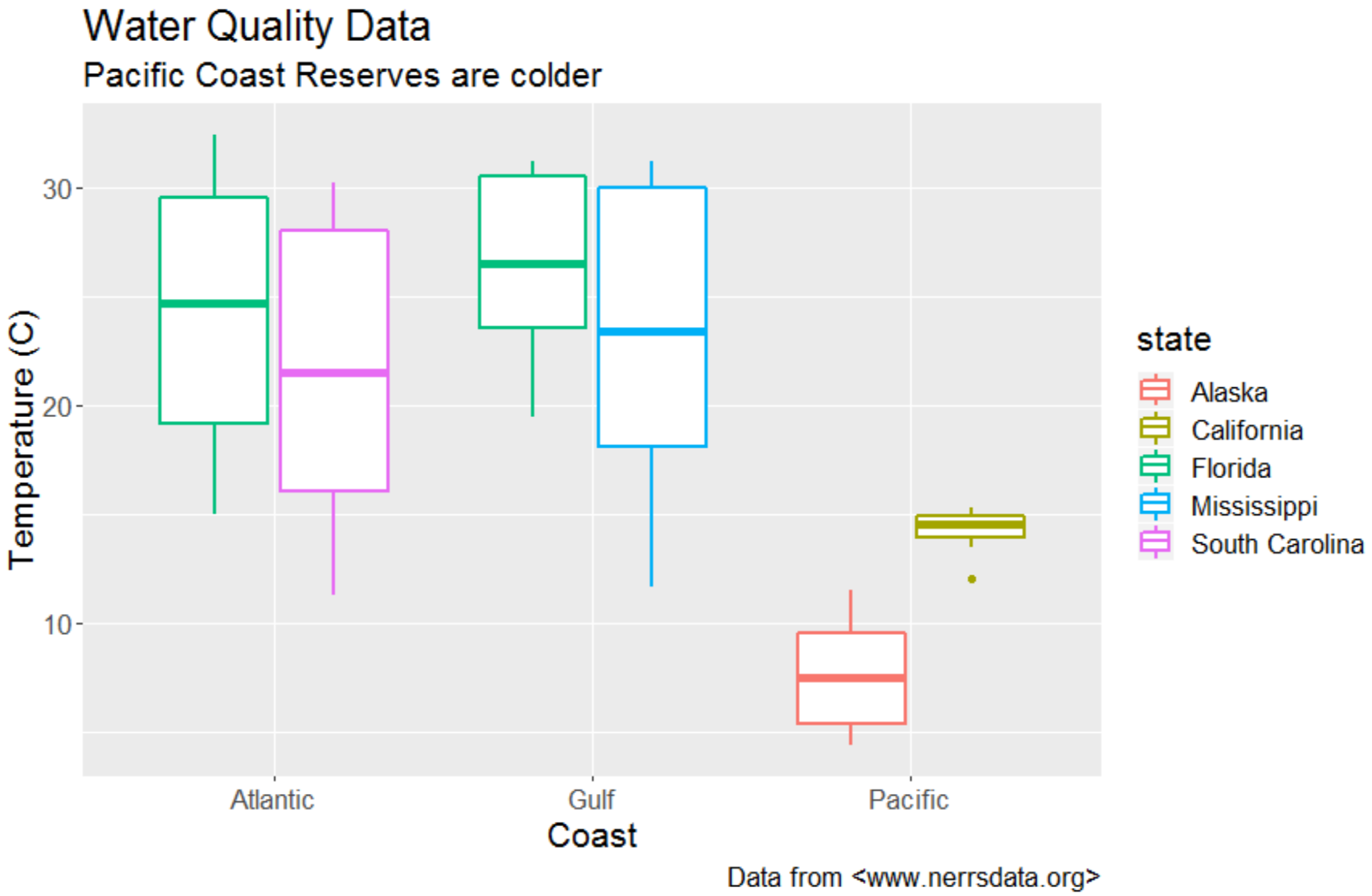
COLOR AND FILL SCALES (CONTINUOUS)

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**)**

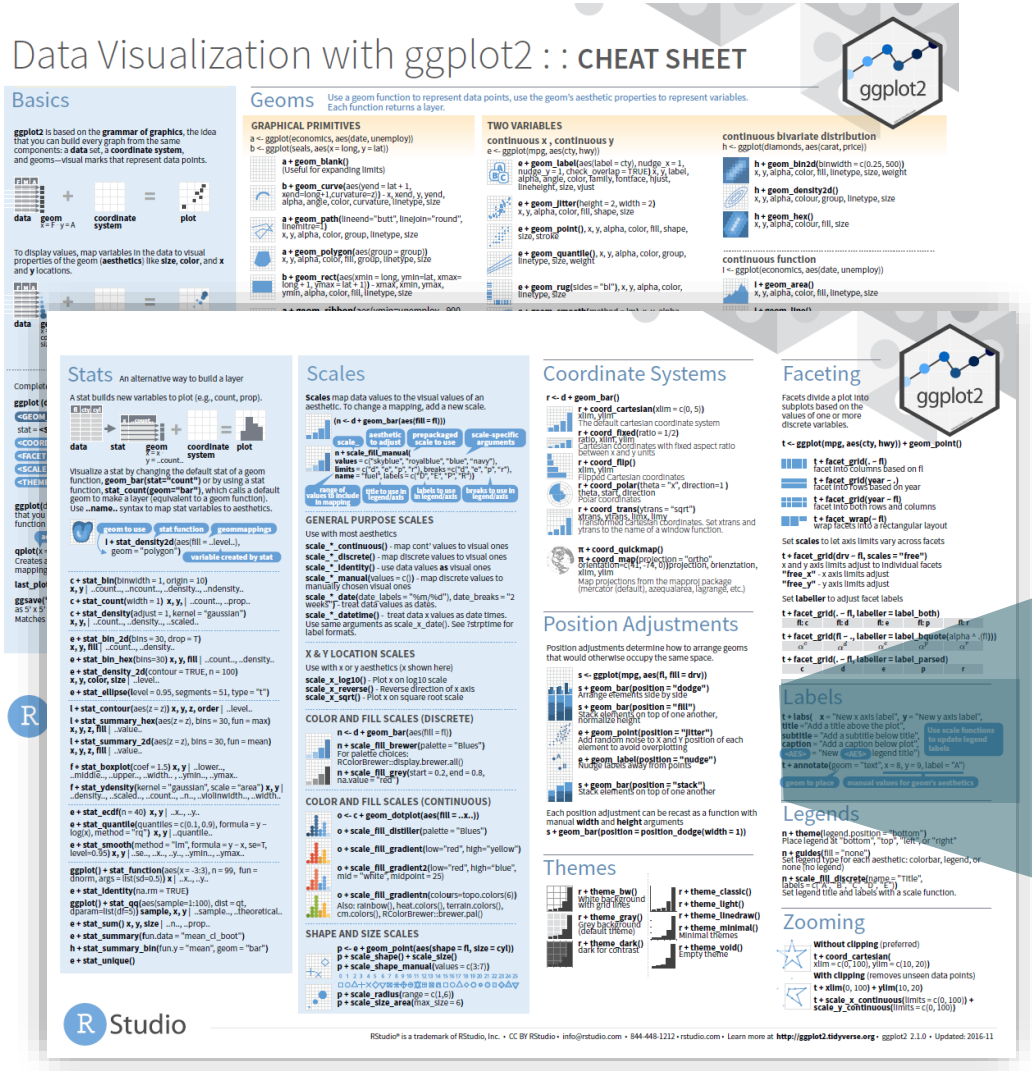
Titles and Captions



Titles and Captions

```
ggplot(data = wq) +
 geom_boxplot(mapping = aes(x = coast, y = temp, color = state)) +
 labs(x = "Coast",
      y = "Temperature (C)",
      title = "Water Quality Data",
      subtitle = "Pacific Coast Reserves are colder",
      caption = "Data from <www.nerrsdata.org>")
```

Data Visualization with ggplot2 : : CHEAT SHEET



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",
<AES> = "New <AES> legend title")
t + annotate(geom = "text", x = 8, y = 9, label = "A")
```

Use scale functions to update legend labels

geom to place

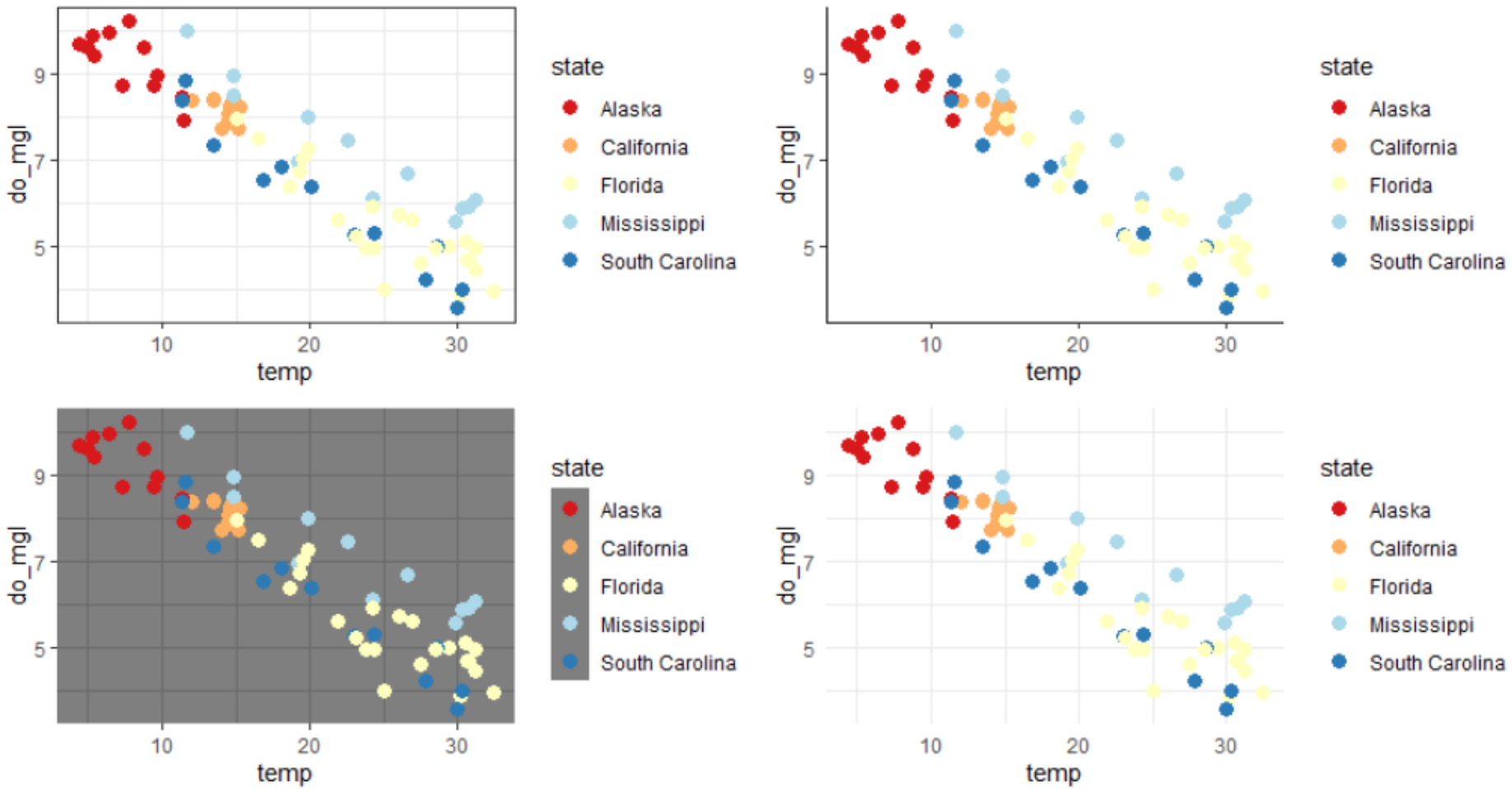
manual values for geom's aesthetics

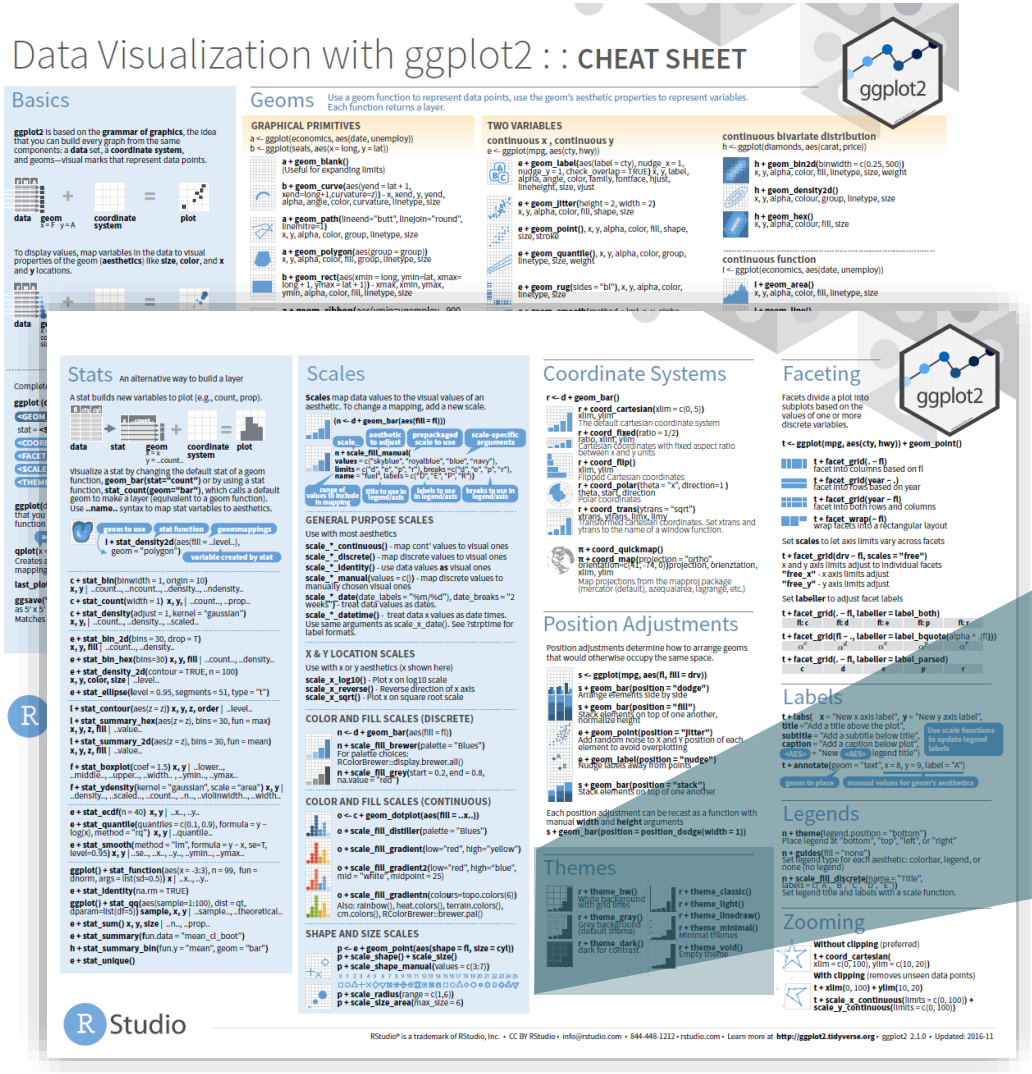
Themes: the non-data elements of your plot.

Do not affect how the data is:

- rendered by geoms, or
- transformed by scales.

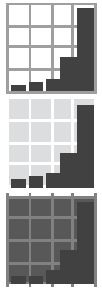
Help you make the plot aesthetically pleasing or match an existing style guide (fonts, ticks, panel strips, backgrounds, etc).


do_mgl
temp
state
Alaska
California
Florida
Mississippi
South Carolina



The ggthemes package by Jeffrey Arnold provides even more complete themes!

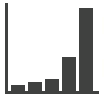
Themes



r + theme_bw()
White background with grid lines

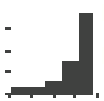
r + theme_gray()
Grey background (default theme)

r + theme_dark()
dark for contrast



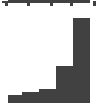
r + theme_classic()

r + theme_light()



r + theme_linedraw()

r + theme_minimal()
Minimal themes



r + theme_void()
Empty theme

Structure of the theme system

Three components:

1. `theme()` function
2. Theme elements (e.g., `plot.title` element which controls the appearance of the plot title)
3. Element function (of which there are four)
    1. `element_text()`
    2. `element_line()`
    3. `element_rectangle()`
    4. `element_blank()`

Structure of the theme system

```
p + theme(plot.title = element_text(size = 18),
          axis.y.title = element_text(size = 12),
          axis.x.title = element_text(size = 12),
          axis.text = element_text(vjust = 0.9),
          panel.background = element_rect(fill = “white”))
```

Explore all the different elements by going to the help page
`?theme`

ggThemeAssist

Saving your plot

What does this command return?

```
getwd()
```

The Working Directory

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

- This is the “working directory”
- R will look for files here
- R will save files here

The files pane of RStudio IDE will display your working directory

Saving plots

`ggsave()` will save the last plot.

```
ggsave("my-first-plot.png")
ggsave("my-first-plot.tiff")
```

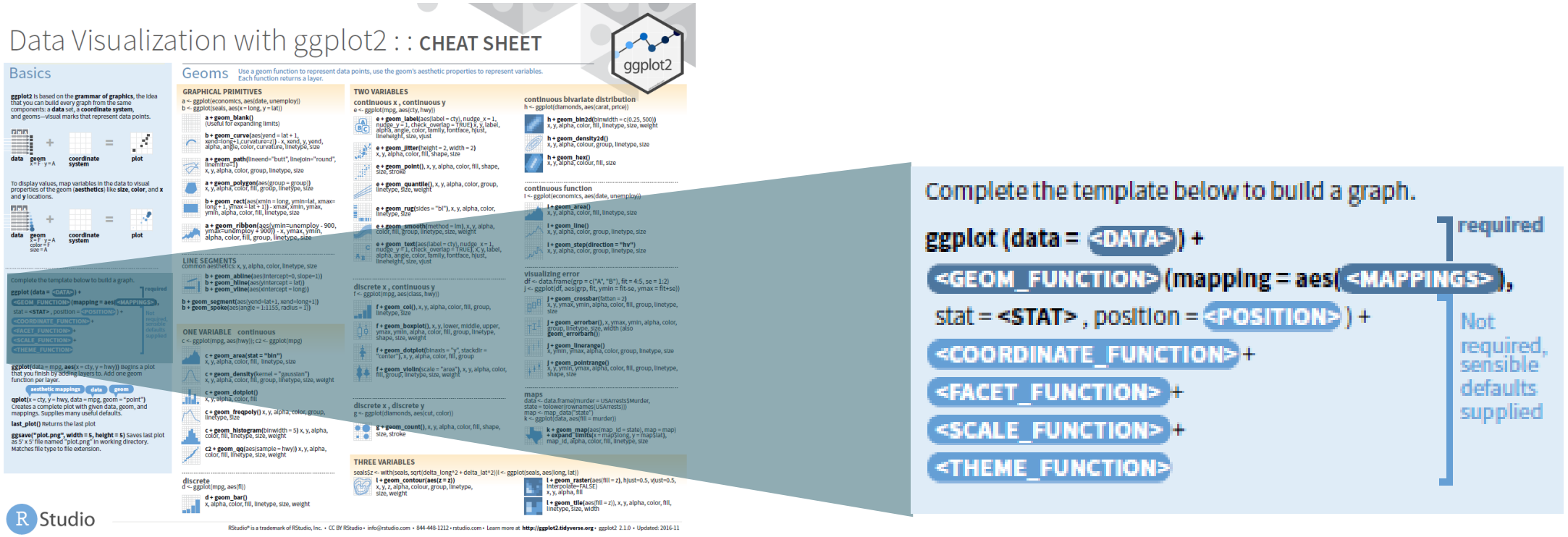
You can further specify size:

```
ggsave("my-first-plot.png", height = 6, width = 8,
       units = "cm")
```

Or even resolution:

```
ggsave("my-first-plot.png", dpi = 300)
```

ggplot2 template



CHALLENGE

Using the `diamonds` data make a scatterplot of `carat` by `price` and assign color by `clarity`.

Customize the themes and colors of the plot to make it the **ugliest** plot that you can.

Save this plot in your files pane.

OYO

Determine what the default stat is for each of the following geoms:

- `geom_line`
- `geom_density`
- `geom_smooth`

OYO

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

(You may have to refresh your files list).