

Data Visualization in R with **ggplot2**

The Structure of ggplot2 (Part 1)

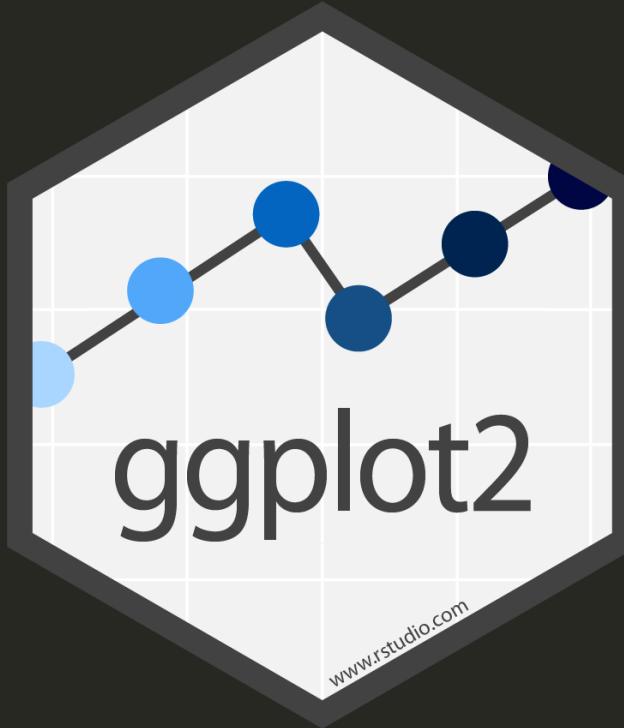
Cédric Scherer

Physalia Courses | March 2-6 2020

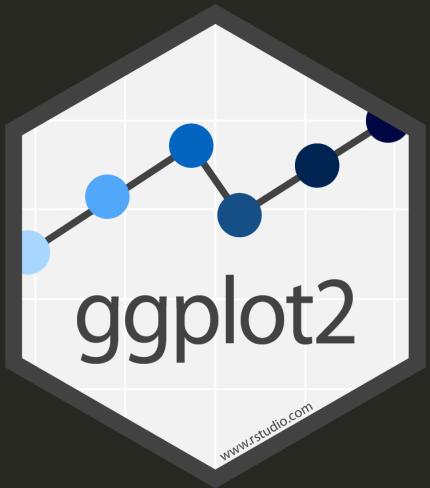
Photo by Richard Strozyński

Part 1

A Basic ggplot



The **ggplot2** Package



ggplot2 is a system for declaratively creating graphics,
based on "The Grammar of Graphics" (Wilkinson, 2005).

You provide the data, tell **ggplot2** how to map variables to aesthetics,
what graphical primitives to use, and it takes care of the details.

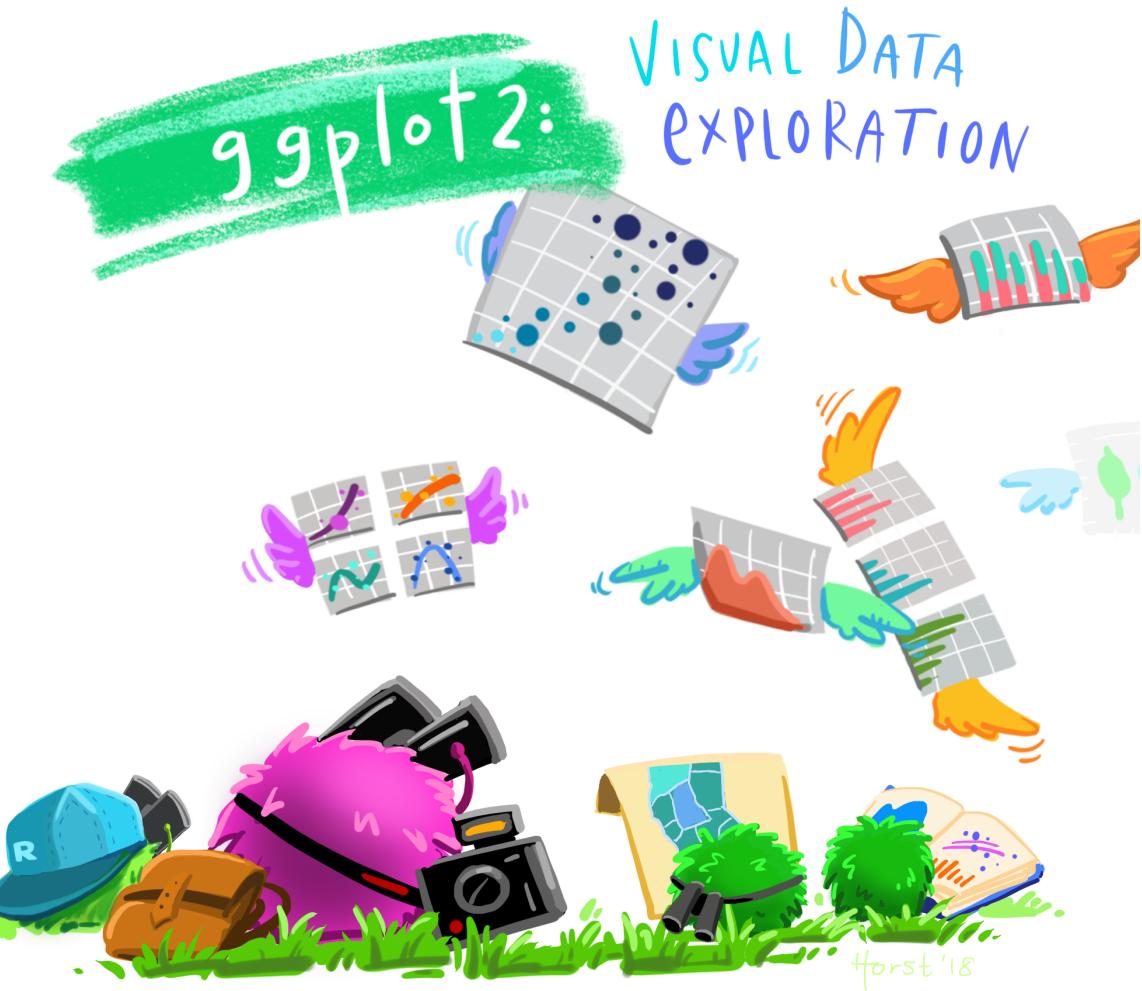


Illustration by Allison Horst (github.com/allisonhorst/stats-illustrations)

Advantages of ggplot2

- consistent underlying grammar of graphics (Wilkinson, 2005)
- very flexible, layered plot specification
- theme system for polishing plot appearance
- active and helpful community

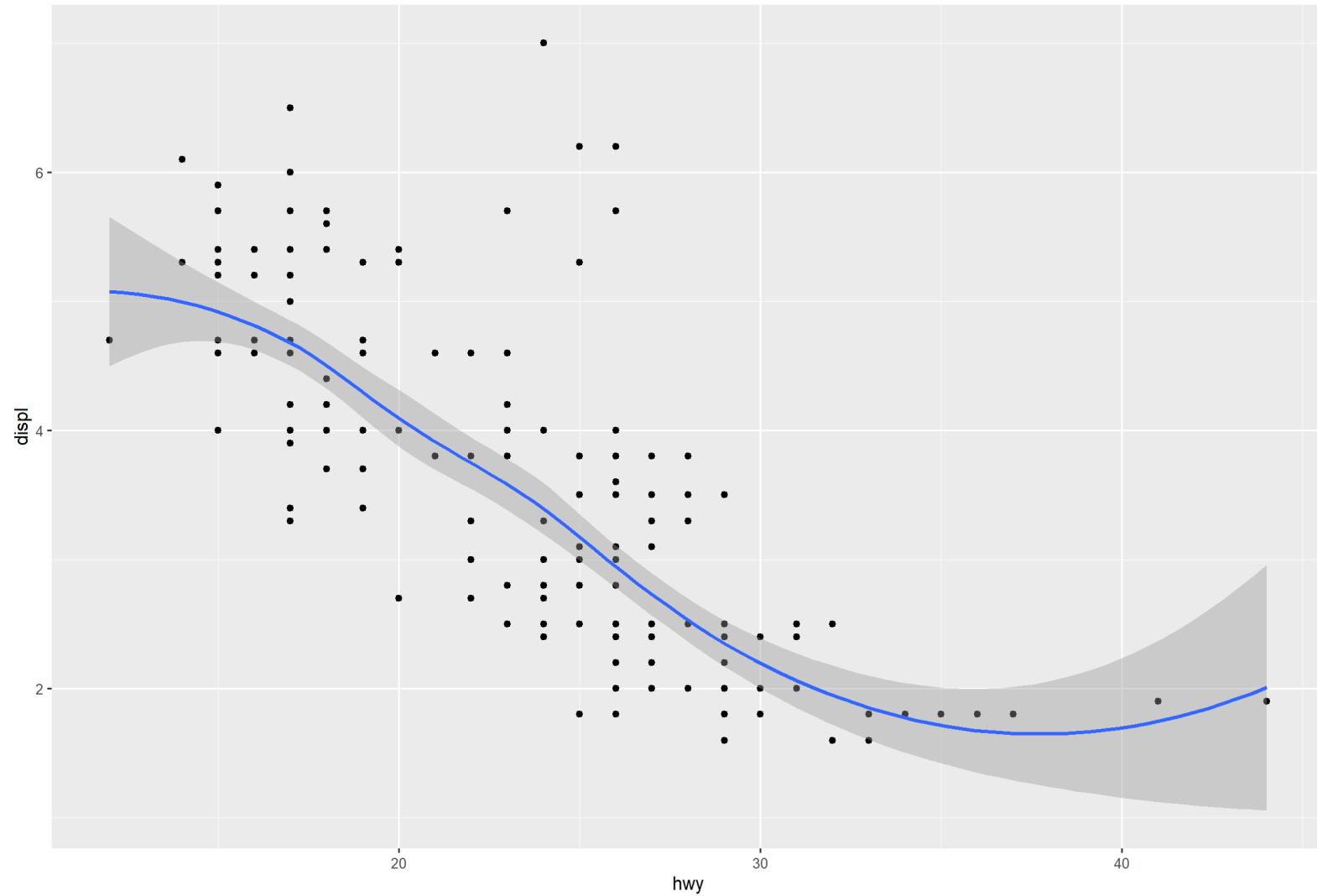
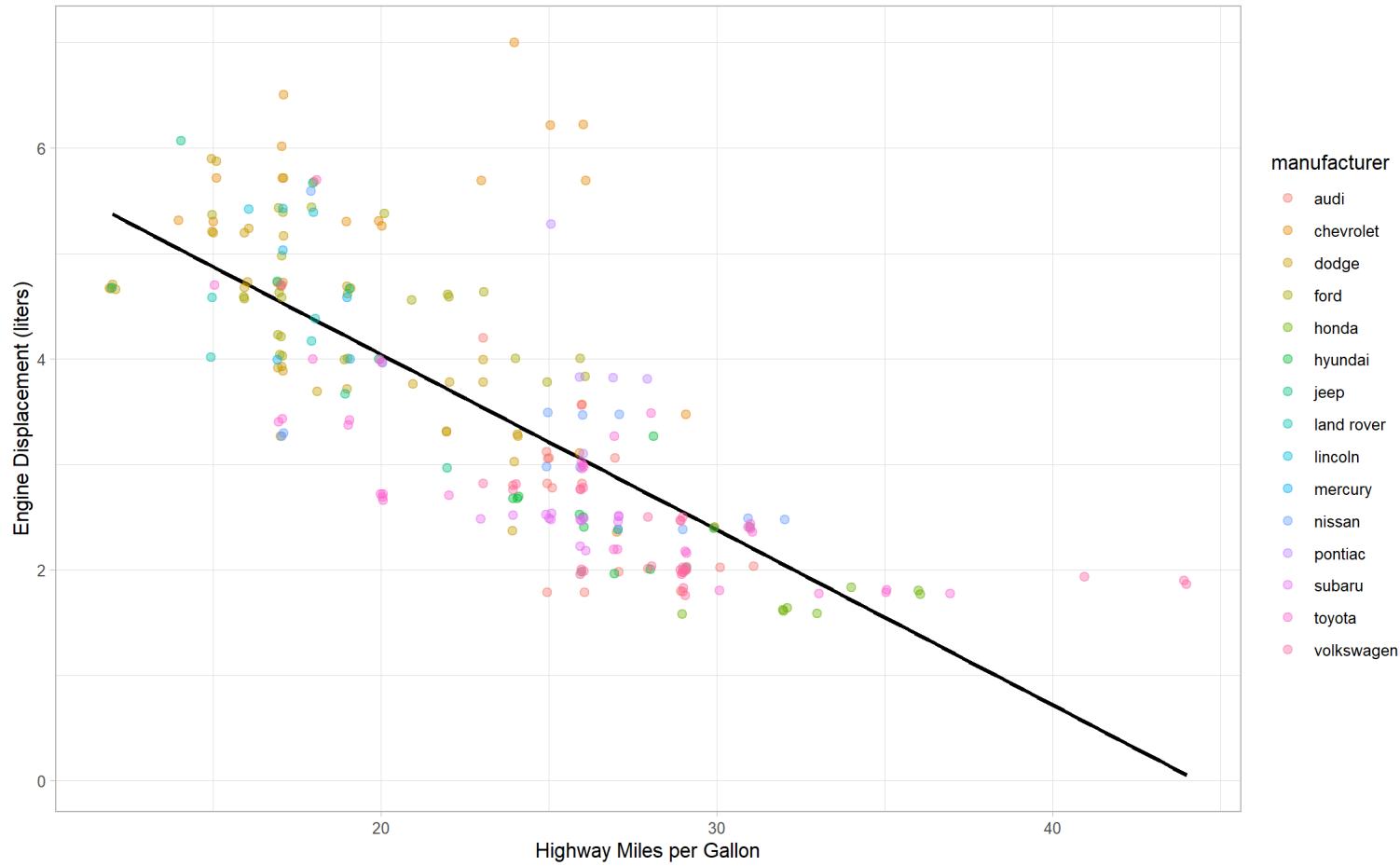




Fig 1:

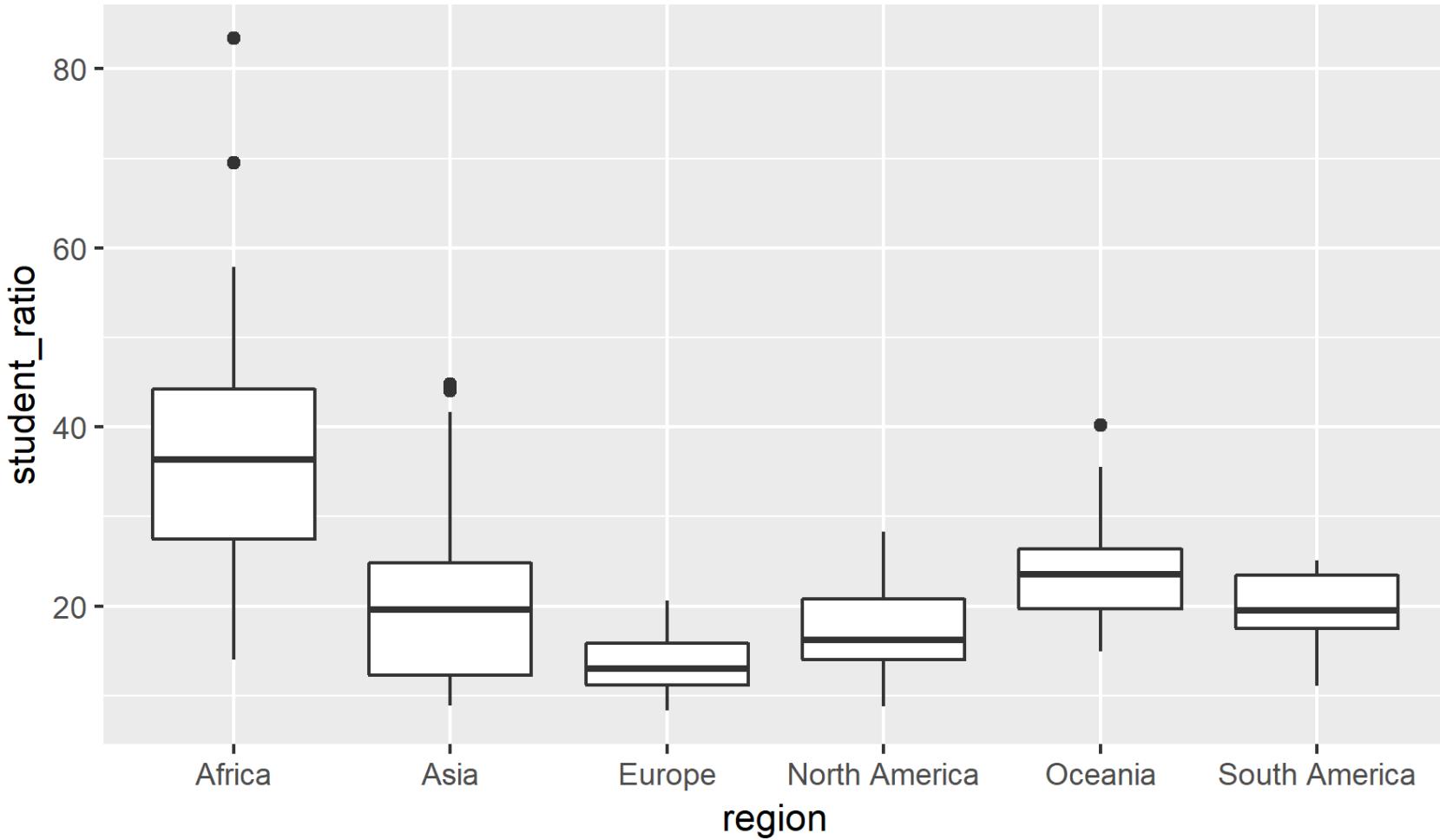
Weight and displacement by manufacturers of 38 popular models of cars

Simple linear regression of engine displacement (in liters) versus highway miles per gallon (MPG) for popular models of cars from 1999 to 2008.



Data: EPA (www.fueleconomy.gov)

The Evolution of a ggplot



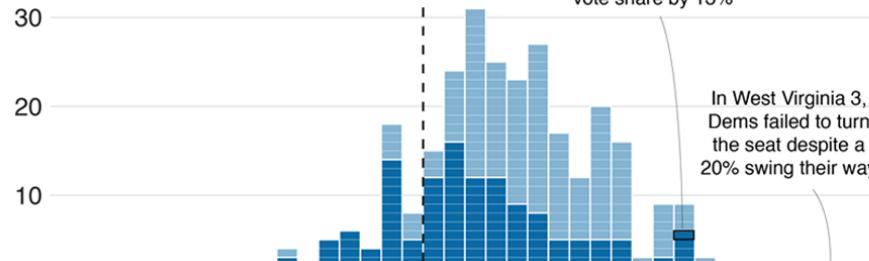
Data: UNESCO Institute for Statistics
Visualization by Cédric Scherer

The `ggplot2` Showcase

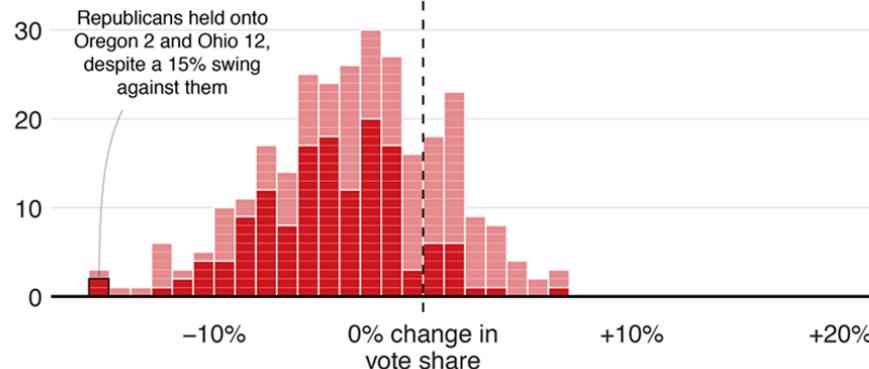
Blue wave

■ Won seat ■ Didn't win

Democrat candidates

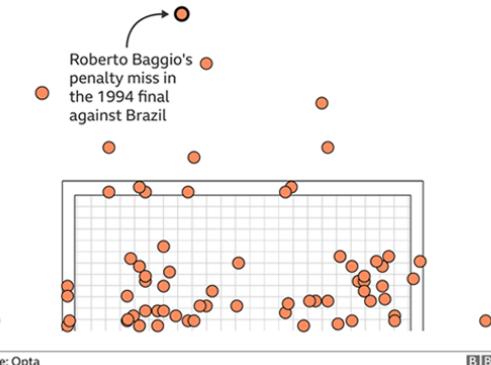


Republican candidates



Where penalties are saved

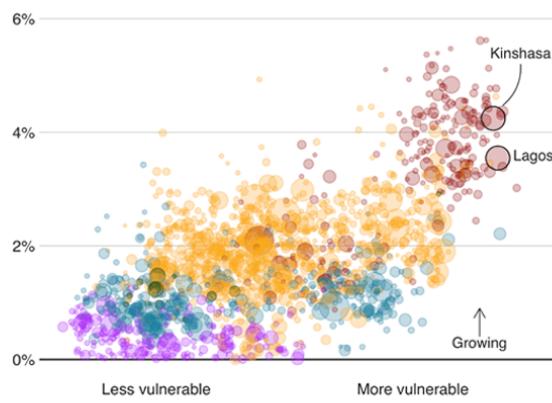
World Cup shootout misses and saves, 1982-2014



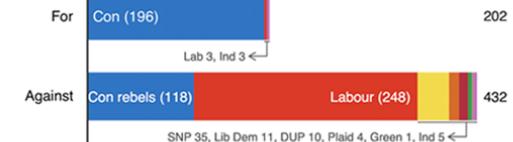
Fast-growing cities face worse climate risks

Population growth 2018-2035 over climate change vulnerability

■ Africa ■ Asia ■ Americas ■ Europe ■ Oceania



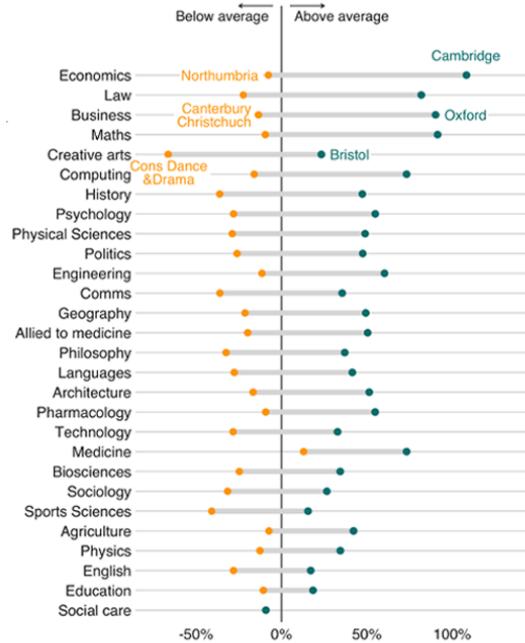
MPs rejected Theresa May's deal by 230 votes



Source: Commons Votes Services. Excludes 'tellers', the Speaker and deputies

Earnings vary across unis even within subjects

Impact on men's earnings relative to the average degree



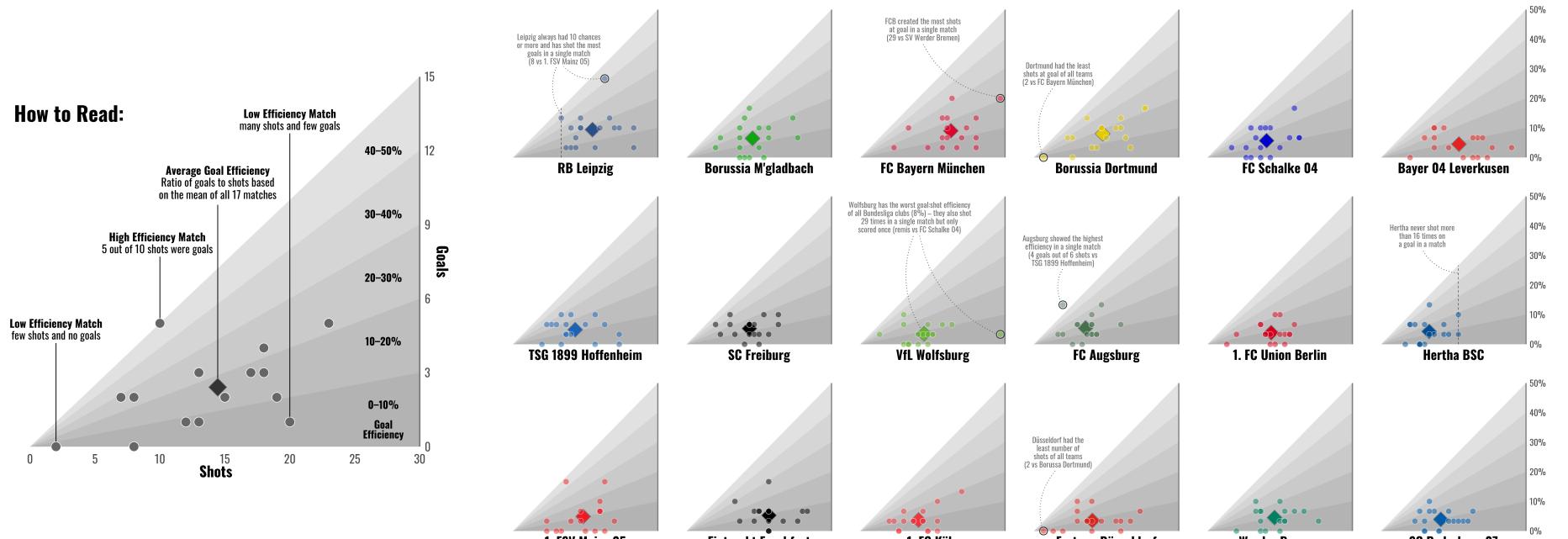
Collection of BBC Graphics

(modified from bbc.github.io/rcookbook)

RB Leipzig and Borussia Dortmund Make the Most of Their Opportunities

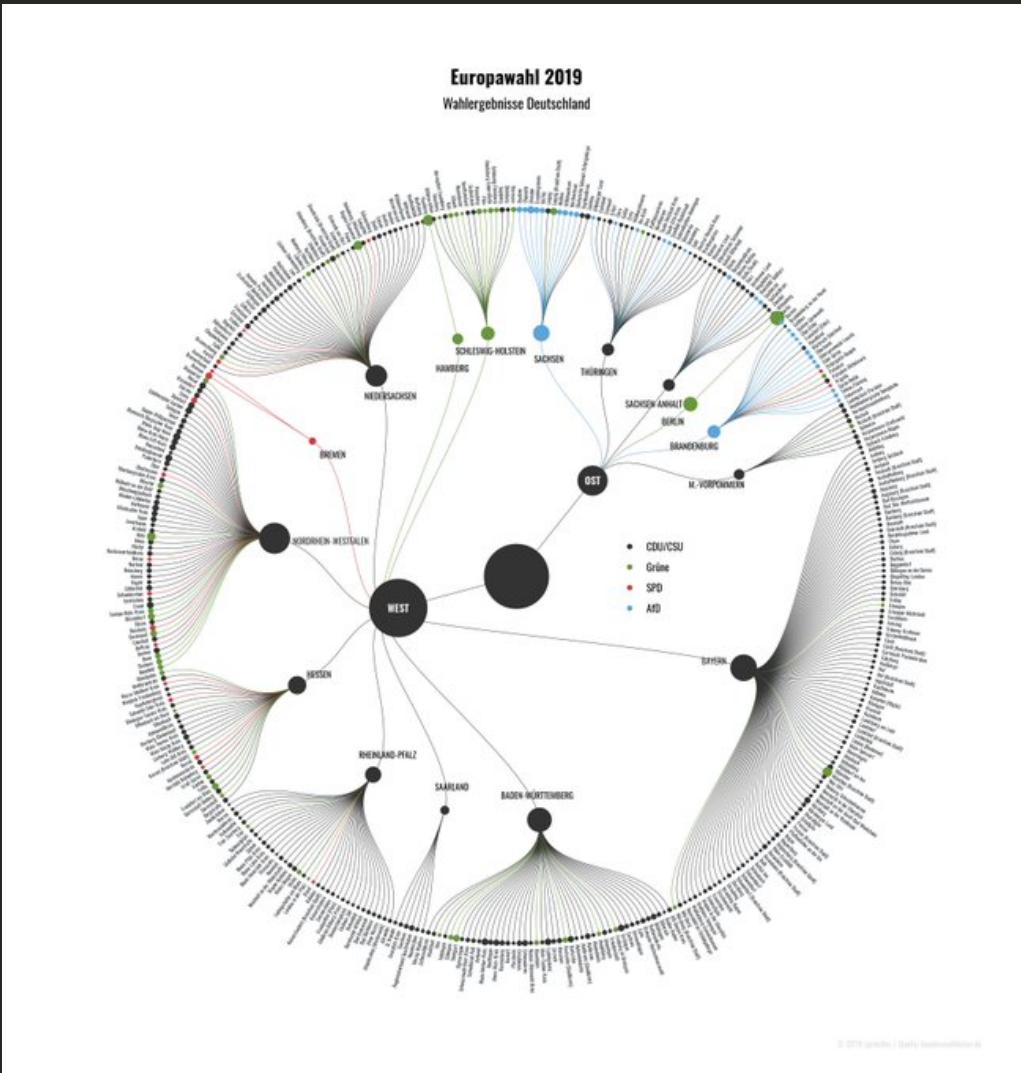
The small multiples show each club's goal:shot efficiency in the first season 2019/2020 of the 1. Bundesliga. However, while Borussia Dortmund also had very bad matches with only 2 chances, the Autumn champion RB Leipzig always scored minimum one goal and shot at least ten times on the opponent's goal in all 17 matches! Of all Bundesliga clubs, RB Leipzig also shot the most goals – 8 against Mainz.

How to Read:



Visualization by Cédric Scherer

Contribution to #SWDchallenge

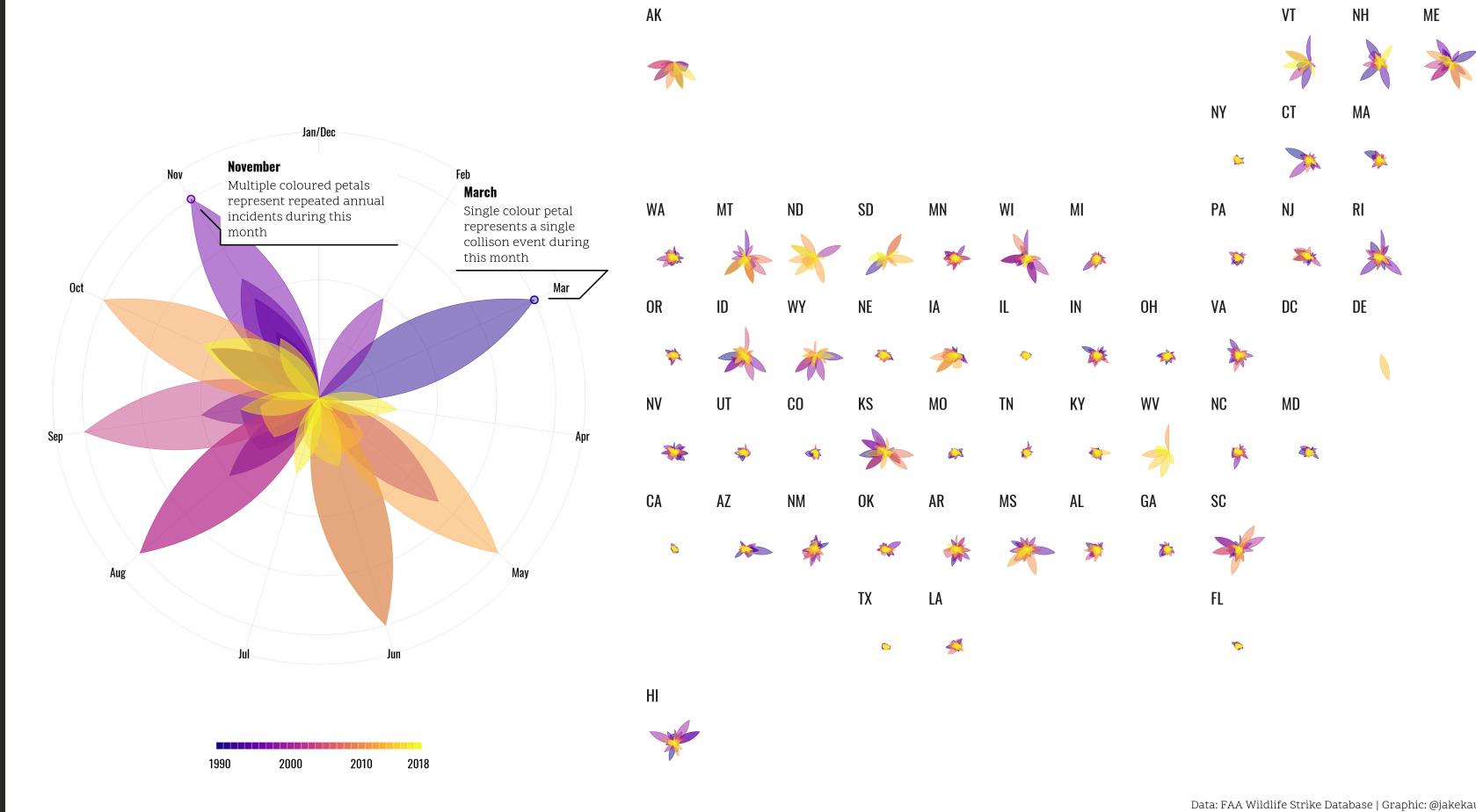


European Elections by Torsten Sprenger

(twitter.com/spren9er/status/1138000009306234880)

Seasonality of Wildlife-Aircraft Collisions by State

Presented below is a petal chart of wildlife collisions with aircraft, with an inset legend showing assisting interpretation. Wildlife collisions by state are presented as small multiples, geographically arranged. Smaller compact flowers illustrate states with collisions occurring year round, while the bigger flowers tend to see single or concentrated spikes of collision activity. Flowers with diverse colours indicate repeated annual collisions while the single-hued flowers illustrate more sparse or isolated annual events.



Data: FAA Wildlife Strike Database | Graphic: @jakekaupp

#TidyTuesday Contribution by Jake Kaupp

(github.com/jkaupp/tidytuesdays)

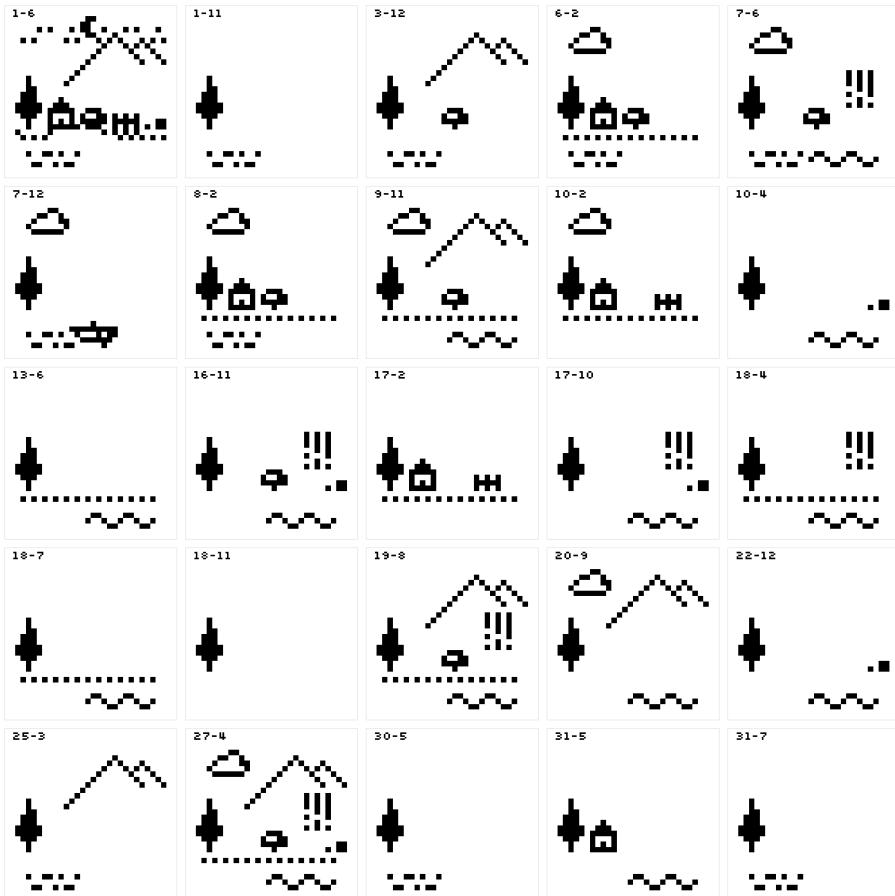
BOB ROSS - PAINTING BY THE ELEMENTS

GRAPHIC REPRESENTATIONS OF BOB ROSS' PAINTINGS WITH ELEMENTS IDENTIFIED IN THEM BY WALT HICKEY (FIVETHIRTYEIGHT). EACH ELEMENT REPRESENTS ONE OR MORE OCCURRENCES IN THE PAINTING. ONLY THE ELEMENTS IN THE LEGEND BELOW ARE DRAWN. TO THE RIGHT THERE ARE 25 RANDOM PAINTINGS THAT BOB PAINTED IN 'THE JOY OF PAINTING', WITH THE SEASON AND EPISODE NUMBER.



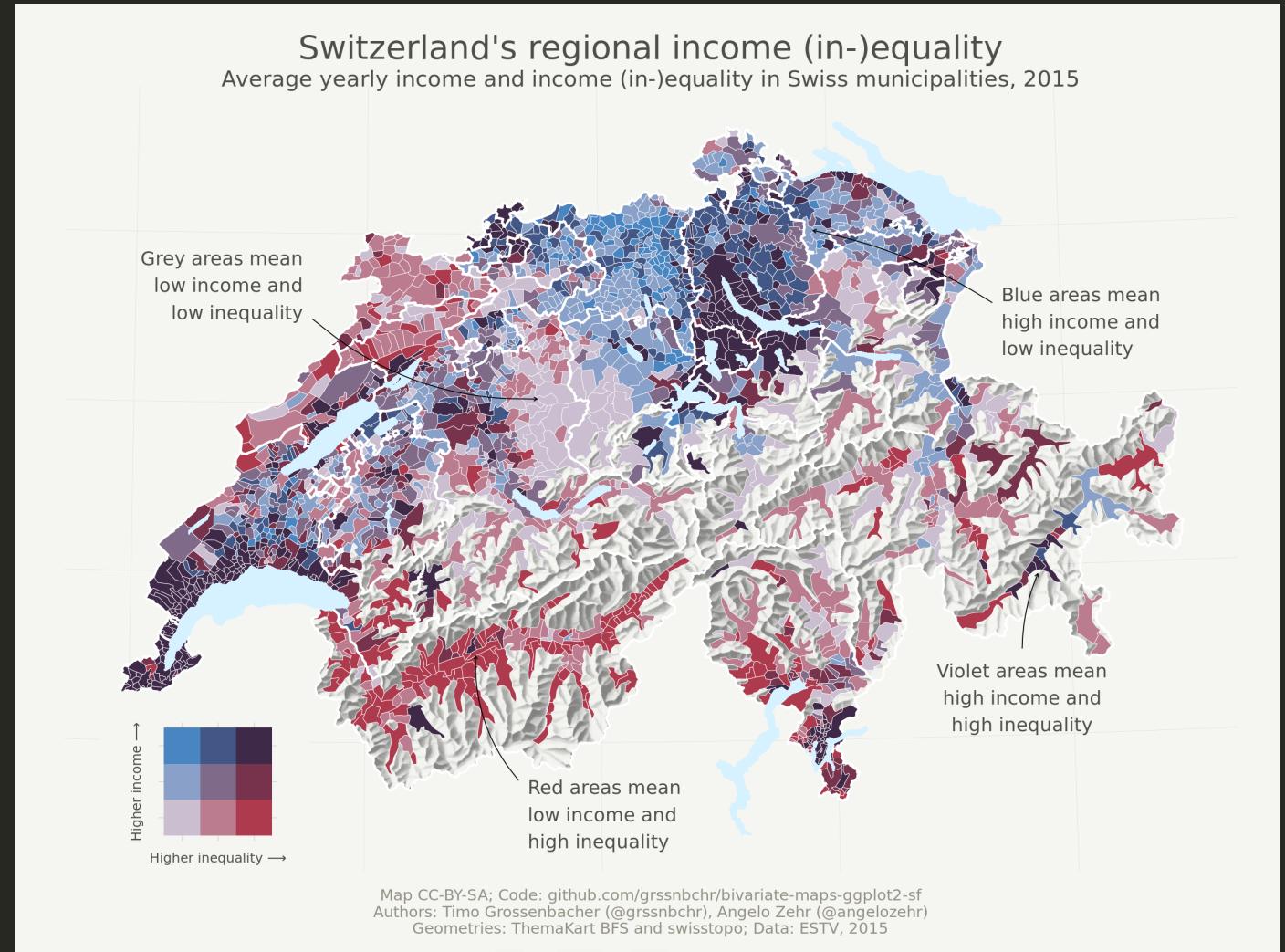
CLOUD (CIRRUS, CUMULUS), MOON, NIGHT, PALM TREE, SUN, MOUNTAIN (HILLS), CACTUS, LIGHTHOUSE, ROCKS, TREE (DECIDUOUS, CONIFER), BEACH, BOAT, CABIN (BARN, BUILDING, FARM), SEA (OCEAN, WAVES), BUSHES, FENCE, GRASS, WATERFALL, LAKE, RIVER

SOURCE: FIVETHIRTYEIGHT | PLOT: GEORGIOS KARAMANIS



#TidyTuesday Contribution by Georgios Karamanis

(github.com/gkaramanis/tidytuesday)

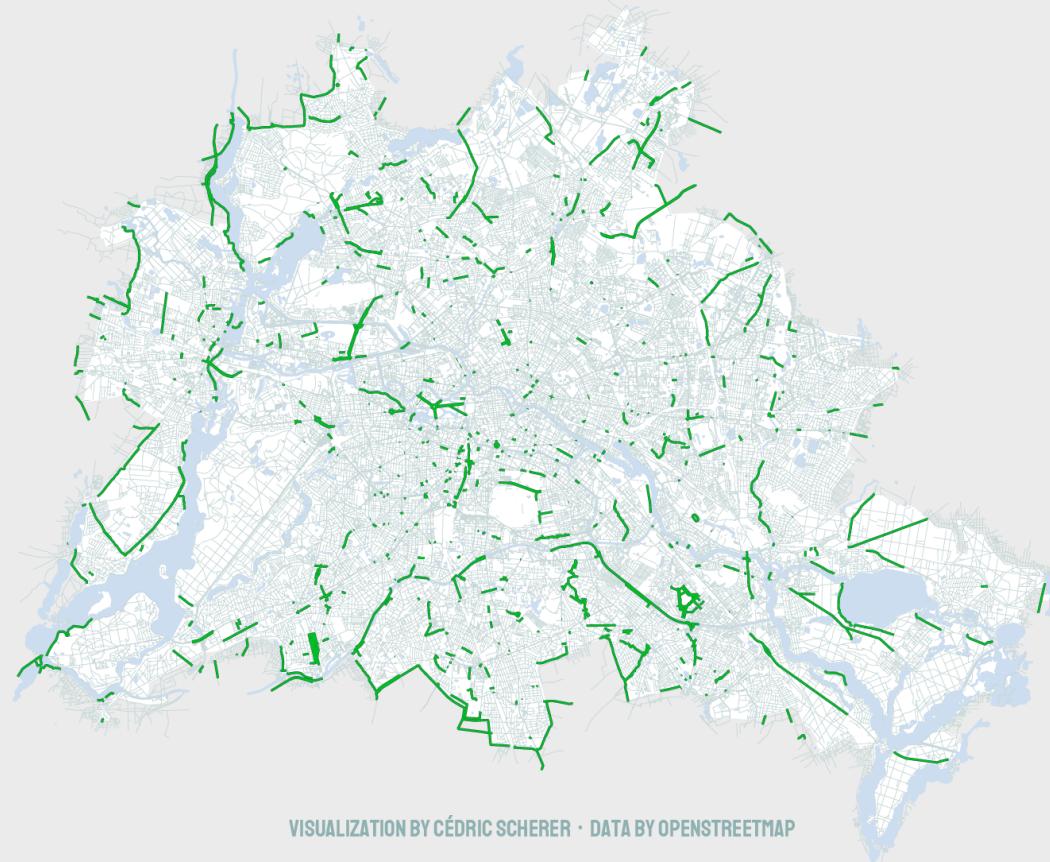


Bivariate Map by Timo Gossenbacher

(timogrossenbacher.ch/2019/04/bivariate-maps-with-ggplot2-and-sf)

MOVING THROUGH BERLIN BY BIKE

ROAD NETWORK OF BERLIN SHOWING STREETS DESIGNATED FOR CYCLISTS AND CAR DRIVERS



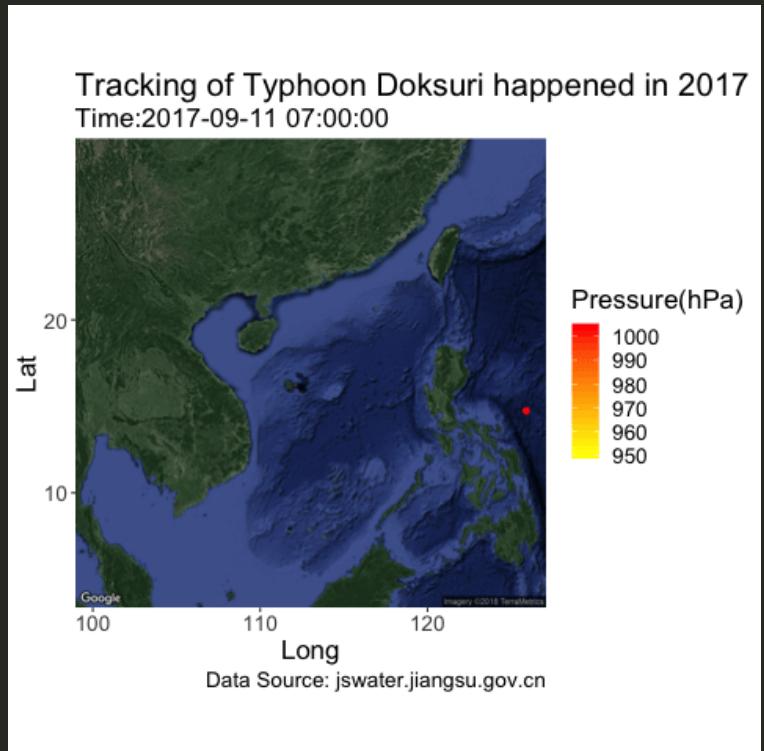
VISUALIZATION BY CÉDRIC SCHERER · DATA BY OPENSTREETMAP

Contribution to the #30DayMapChallenge

MOABIT

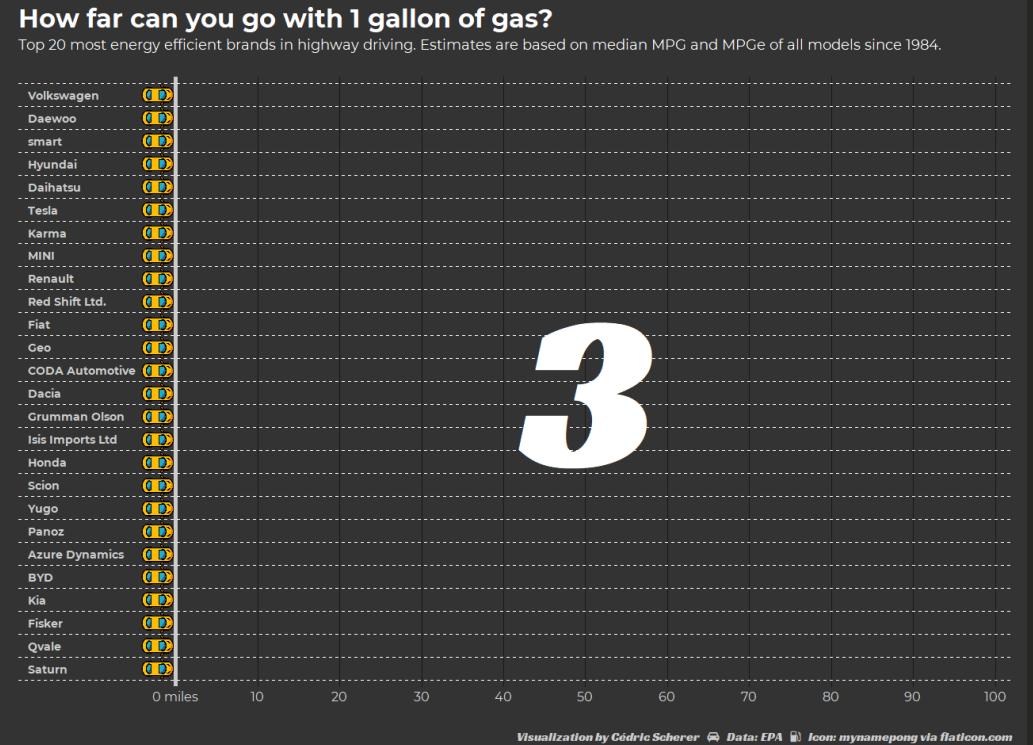


Contribution to the #30DayMapChallenge



Example of the {gganimate} Package

(github.com/thomasp85/gganimate/wiki)



Contribution to #TidyTuesday

The Setup

The package

ggplot2 is a data visualization package for the programming language **R** created by Hadley Wickham in 2005.

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

ggplot2 is part of the **tidyverse**, a set of packages that work in harmony to manipulate and explore data.

```
install.packages("tidyverse")
library(tidyverse)
```



Source: dcook.org/files/rstudio/#3

The package

We are going to use the development version of `ggplot2`.

You can install the most recent version on GitHub using the package `devtools`:

```
install.packages("devtools")
devtools::install_github("tidyverse/ggplot2")
```

The data

We use data from the *National Morbidity and Mortality Air Pollution Study* (NMMAPS), filtered for the city of Chicago and the timespan from January 1997 to December 2000.

```
chic <- read_csv(  
  "https://raw.githubusercontent.com/Z3tt/ggplot-courses/master/data/chicago-nmmaps.csv"  
  col_types = cols(season = col_factor(), year = col_factor()))  
)  
tibble::glimpse(chic)  
## Observations: 1,461  
## Variables: 10  
## $ city      <chr> "chic", "chic", "chic", "chic", "chic", "chic", "chic", "c...  
## $ date      <date> 1997-01-01, 1997-01-02, 1997-01-03, 1997-01-04, 1997-01-0...  
## $ death     <dbl> 137, 123, 127, 146, 102, 127, 116, 118, 148, 121, 110, 127...  
## $ temp      <dbl> 36.0, 45.0, 40.0, 51.5, 27.0, 17.0, 16.0, 19.0, 26.0, 16.0...  
## $ dewpoint  <dbl> 37.500, 47.250, 38.000, 45.500, 11.250, 5.750, 7.000, 17.7...  
## $ pm10      <dbl> 13.052268, 41.948600, 27.041751, 25.072573, 15.343121, 9.3...  
## $ o3        <dbl> 5.659256, 5.525417, 6.288548, 7.537758, 20.760798, 14.9408...  
## $ time      <dbl> 3654, 3655, 3656, 3657, 3658, 3659, 3660, 3661, 3662, 3663...  
## $ season    <fct> Winter, Winter, Winter, Winter, Winter, Winter, Winter, Wi...  
## $ year      <fct> 1997, 1997, 1997, 1997, 1997, 1997, 1997, 1997, 1997...
```

The Structure of `ggplot2`

"The Grammar of Graphics"

The Structure of `ggplot2`

Layer	Function	Explanation
Data	<code>ggplot(data)</code>	The raw data that you want to visualise.
Aesthetics	<code>aes()</code>	Aesthetic mappings of the geometric and statistical objects.
Layers	<code>geom_*</code> () and <code>stat_*</code> ()	The geometric shapes and statistical summaries representing the data.

The Structure of `ggplot2`

Layer	Function	Explanation
Data	<code>ggplot(data)</code>	The raw data that you want to visualise.
Aesthetics	<code>aes()</code>	Aesthetic mappings of the geometric and statistical objects.
Layers	<code>geom_*</code> () and <code>stat_*</code> ()	The geometric shapes and statistical summaries representing the data.
Scales	<code>scale_*</code> ()	Maps between the data and the aesthetic dimensions.
Coordinate System	<code>coord_*</code> ()	Maps data into the plane of the data rectangle.
Facets	<code>facet_*</code> ()	The arrangement of the data into a grid of plots.
Visual Themes	<code>theme()</code> and <code>theme_*</code> ()	The overall visual defaults of a plot.

The Structure of `ggplot2`

Layer	Function	Explanation
Data	<code>ggplot(data)</code>	The raw data that you want to visualise.
Aesthetics	<code>aes()</code>	Aesthetic mappings of the geometric and statistical objects.
Layers	<code>geom_*</code> () and <code>stat_*</code> ()	The geometric shapes and statistical summaries representing the data.
Scales	<code>scale_*</code> ()	Maps between the data and the aesthetic dimensions.
Coordinate System	<code>coord_*</code> ()	Maps data into the plane of the data rectangle.
Facets	<code>facet_*</code> ()	The arrangement of the data into a grid of plots.
Visual Themes	<code>theme()</code> and <code>theme_*</code> ()	The overall visual defaults of a plot.
Annotations	<code>annotate()</code>	Add additional labels, geometries or images to a plot.

Data

ggplot(data)

ggplot2::ggplot()

```
ggplot
## function (data = NULL, mapping = aes(), ..., environment = parent.frame())
## {
##     UseMethod("ggplot")
## }
## <bytecode: 0x0000000007fa6618>
## <environment: namespace:ggplot2>
```

ggplot(data)

We need to specify data in the `ggplot()` call:

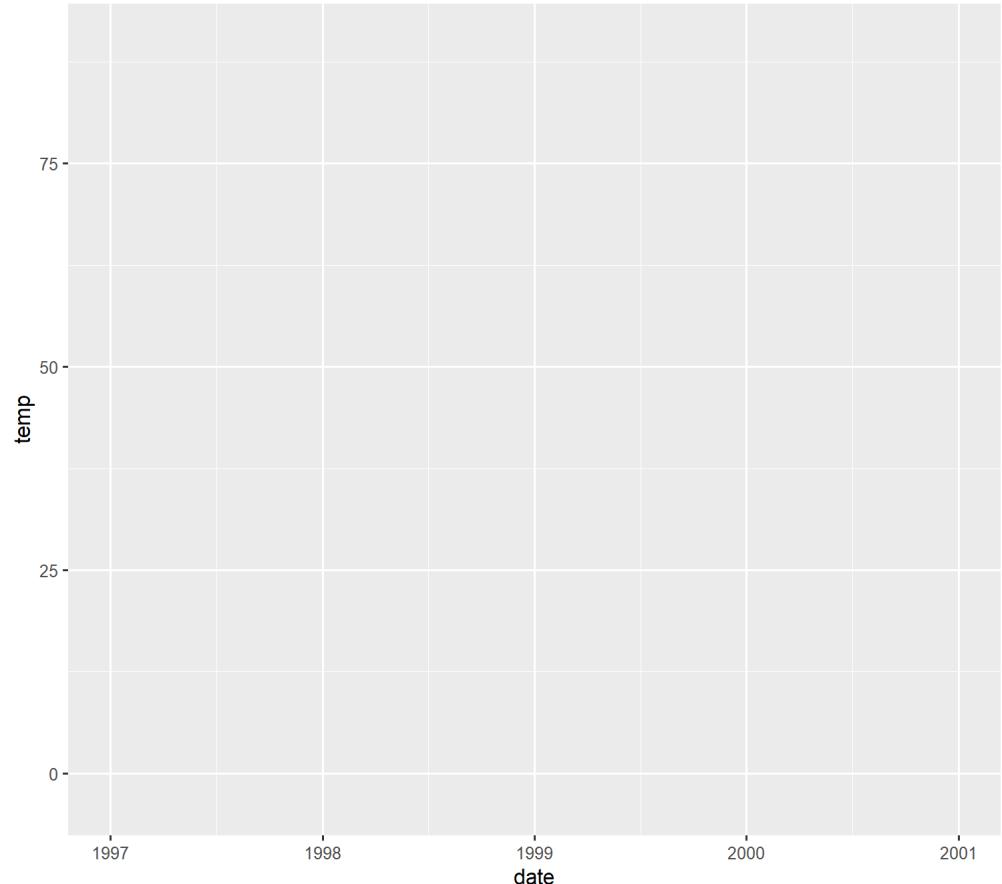
```
ggplot(data = chic)
```

There is only an empty panel because `ggplot2` doesn't know what of the data it should plot.

ggplot(data, aes(x, y))

We need to specify data and the two variables we want to plot as `aes`thetics of the `ggplot()` call:

```
ggplot(data = chic,  
       mapping =  
         aes(  
           x = date,  
           y = temp  
         )  
       )
```



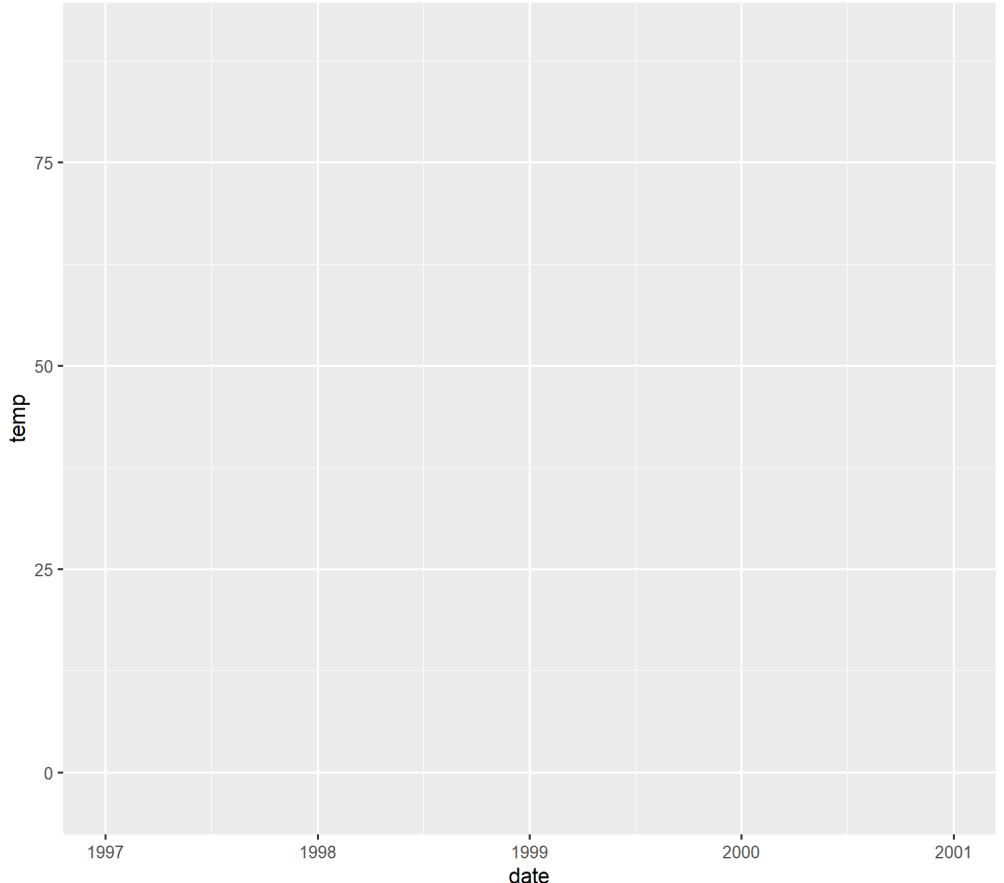
There is only an empty panel because `ggplot2` doesn't know how it should plot the data.

`ggplot(data, aes(x, y))`

We need to specify data and the two variables we want to plot as `aes`thetics of the `ggplot()` call:

Thanks to implicit matching of arguments in `ggplot(data, mapping)` and `aes(x, y)`, we can also write:

```
ggplot(chic, aes(date, temp))
```



`ggplot(data) + aes(x, y)`

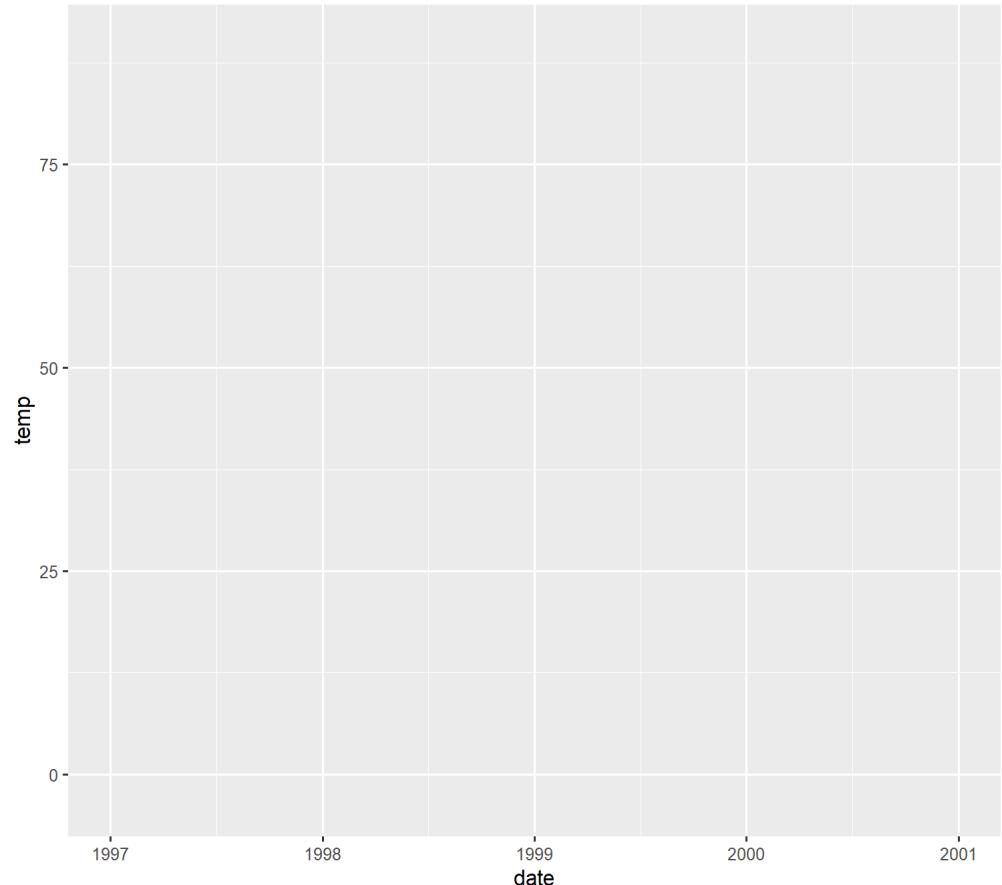
We need to specify data and the two variables we want to plot as `aes`thetics of the `ggplot()` call:

Since almost every `ggplot()` takes the same arguments
`(data, mapping = aes(x, y))`, we can also write:

```
ggplot(chic, aes(date, temp))
```

... or add the `aes`thetics outside the `ggplot()` function:

```
ggplot(chic) +  
  aes(date, temp)
```



Layers

geom_*() and **stat_***()

geom_*() and stat_*()

By adding one or multiple layers we can tell **ggplot2** *how* to represent the data.

There are lots of build-in geometric elements (**geom's**) and statistical transformations (**stat's**):

Layer: geoms

- geom_abline() geom_hline()
geom_vline()
- geom_bar() geom_col()
stat_count()
- geom_bin2d() stat_bin_2d()
- geom_blank()
- geom_boxplot() stat_boxplot()
- geom_contour() stat_contour()
- geom_count() stat_sum()
- geom_density() stat_density()
geom_density_2d()
stat_density_2d()
- geom_dotplot()
- geom_errorbar()
- geom_hex() stat_bin_hex()
- geom_freqpoly() geom_histogram()
stat_bin()
- geom_jitter()
- geom_crossbar() geom_errorbar()
geom_linerange()
geom_pointrange()

Layer: stats

- geom_map()
- geom_path() geom_line()
geom_step()
- geom_point()
- geom_polygon()
- geom_qq_line() stat_qq_line()
geom_qq() stat_qq()
- geom_quantile() stat_quantile()
- geom_ribbon() geom_area()
- geom_rug()
- geom_segment() geom_curve()
- geom_smooth() stat_smooth()
- geom_spoke()
- geom_label() geom_text()
- geom_raster() geom_rect()
geom_tile()
- geom_violin() stat_ydensity()
- coord_sf() geom_sf()
geom_sf_label() geom_sf_text()
stat_sf()

Adapted from ggplot2.tidyverse.org/reference

... and several more in extension packages, e.g. **{ggforce}**, **{ggalt}**, **{ggridges}**, **{ggrepel}**, **{ggcorrplot}**, **{ggraph}**, **{ggdendro}** & **{ggalluvial}**.

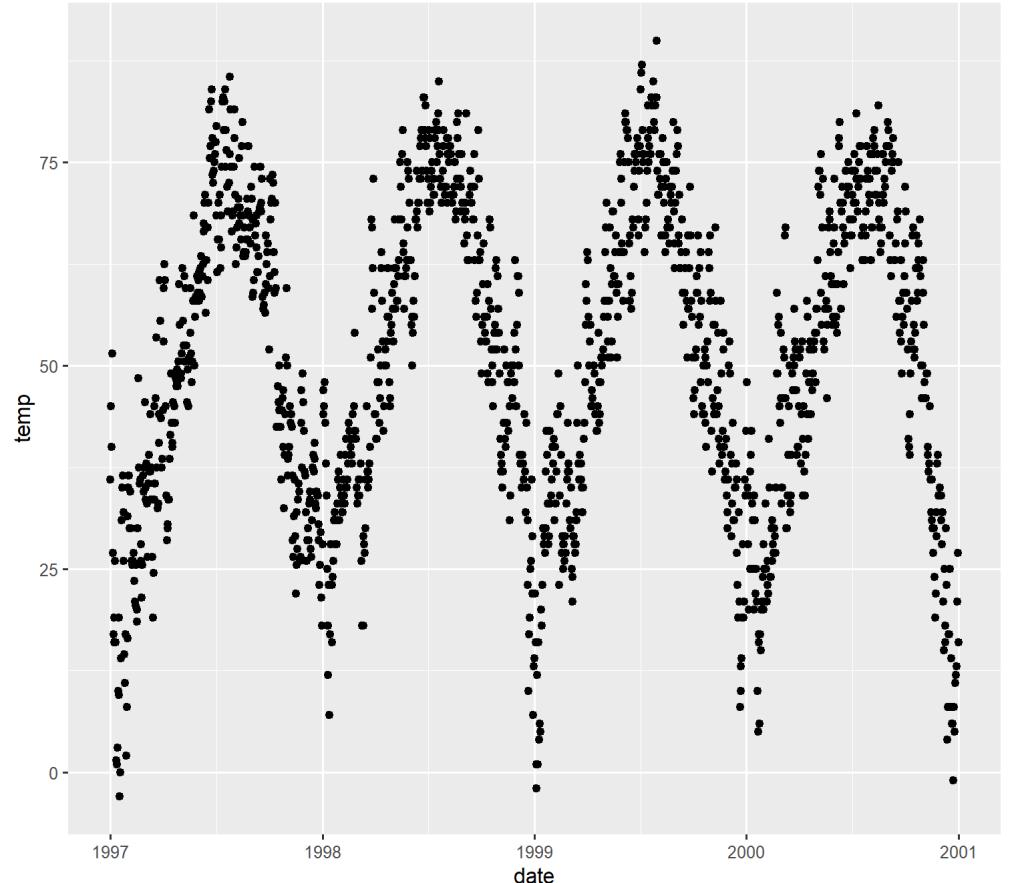
2. Layers

Geometric Layers: **geom_***()

geom_point()

We can tell `ggplot2` to represent the data for example as a scatter plot:

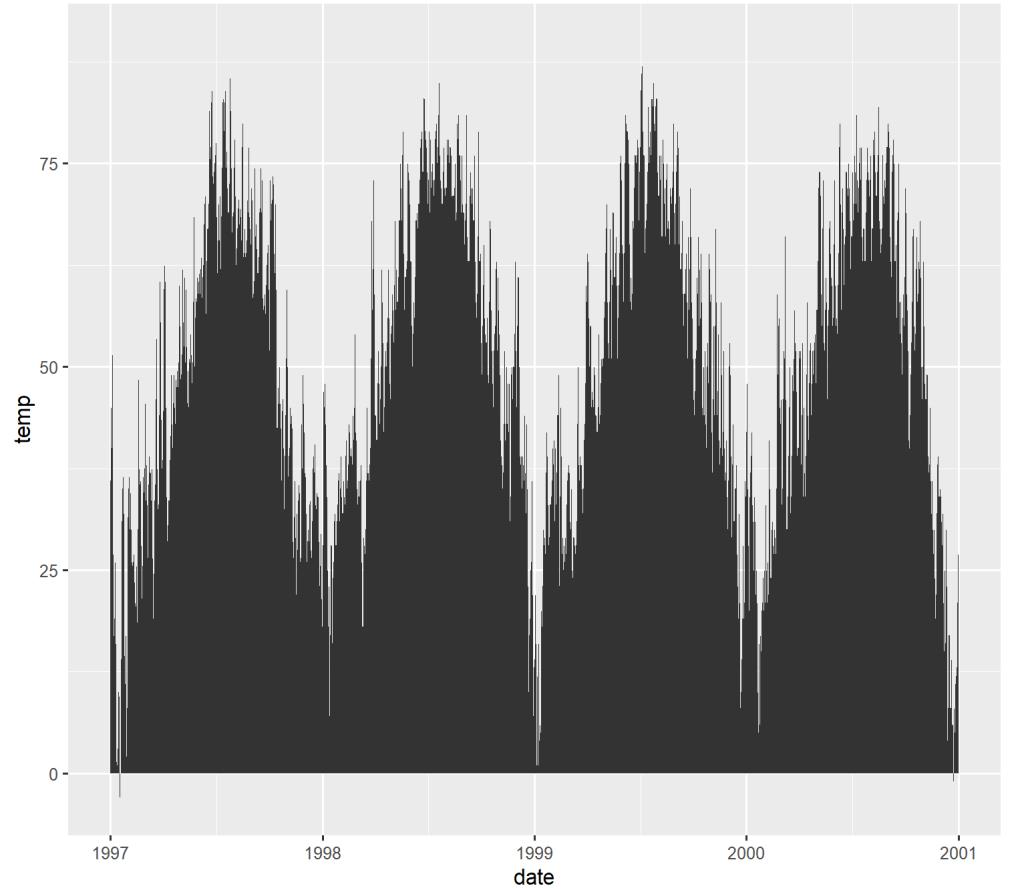
```
ggplot(chic, aes(date, temp)) +  
  geom_point()
```



geom_area()

... or turn it for example into an area plot:

```
ggplot(chic, aes(date, temp)) +  
  geom_area()
```



Spoiler: Themes!

The default theme of `ggplot2` is well known for it's beauty.

Even though Hadley ensures the design defaults were a

deliberate choice because it puts the data forward while still making grid lines visible

one of the first things I do is getting rid of it because I can't see it anymore.

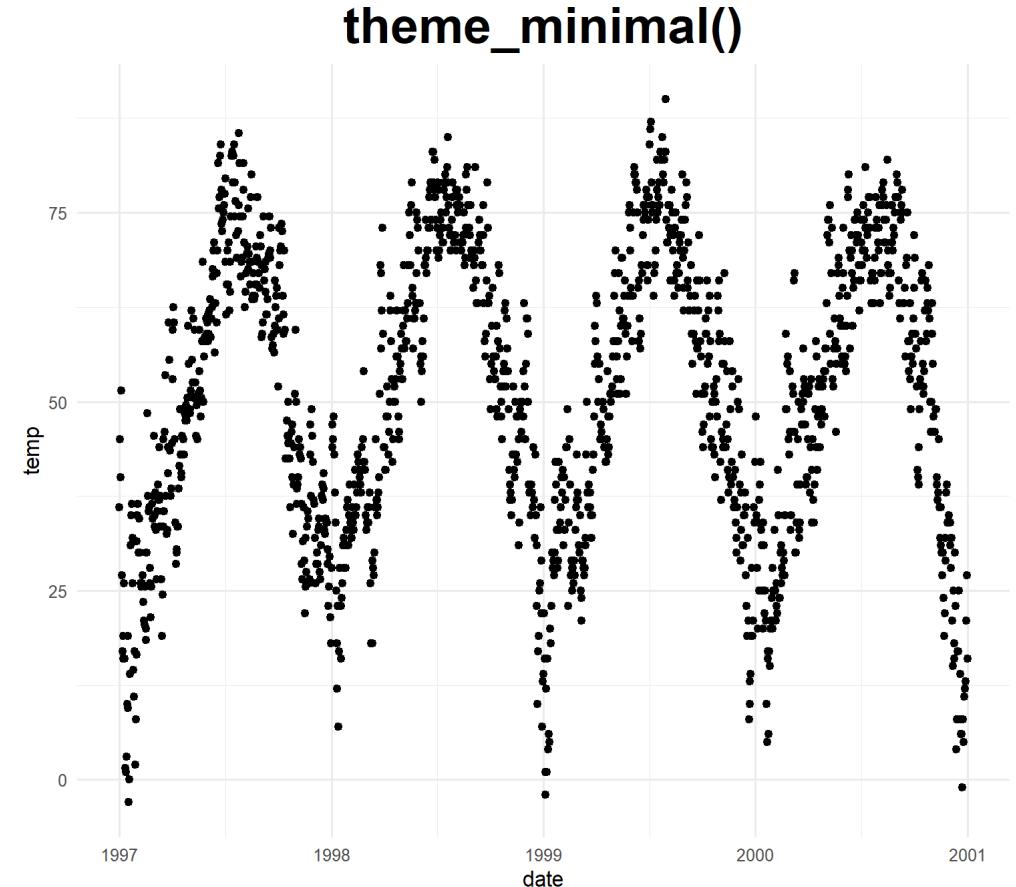
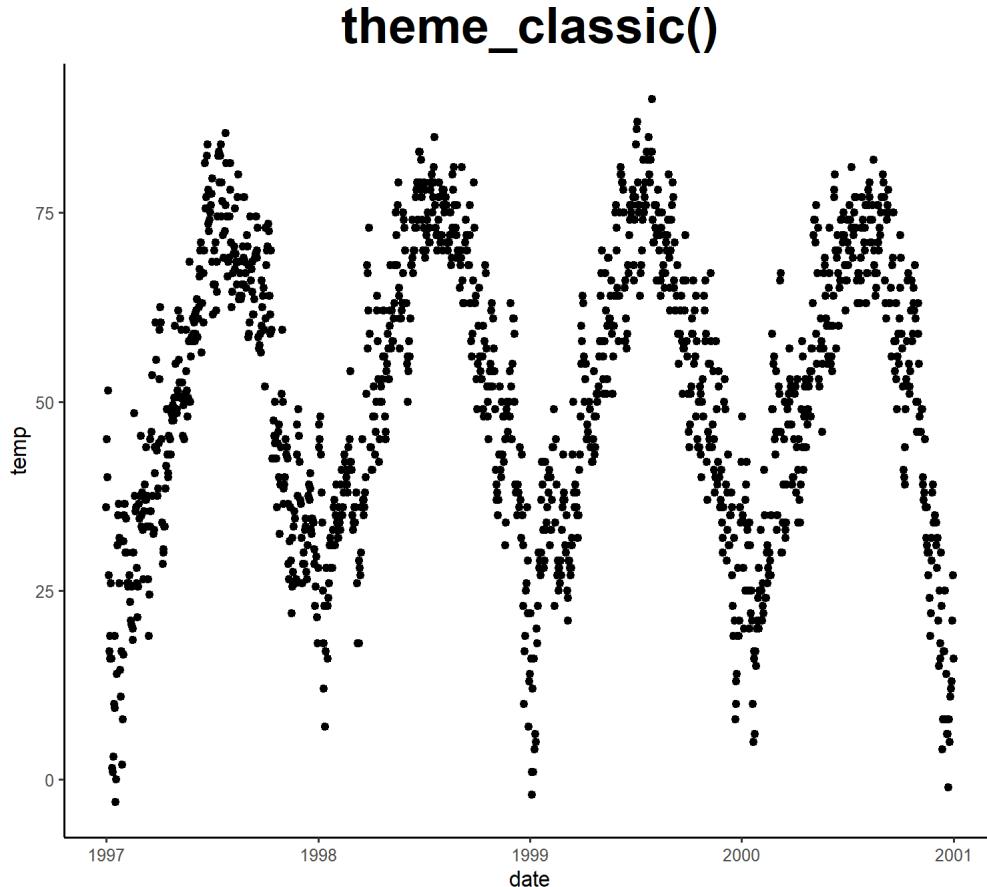
Also, I do not really agree with the reasoning:

The white grid lines are visible (which is important because they significantly aid position judgements), but they have little visual impact and we can easily tune them out.

The grey background gives the plot a similar typographic colour to the text, ensuring that the graphics fit in with the flow of a document without jumping out with a bright white background. Finally, the grey background creates a continuous field of colour which ensures that the plot is perceived as single visual entity.

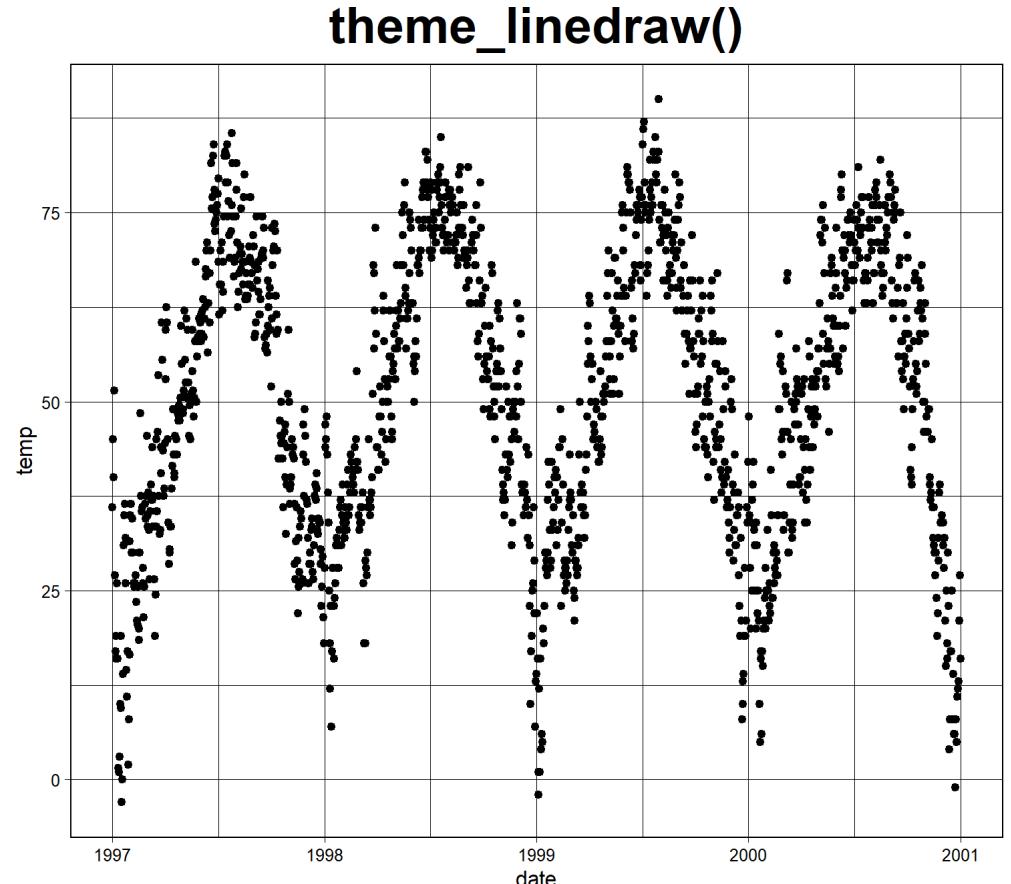
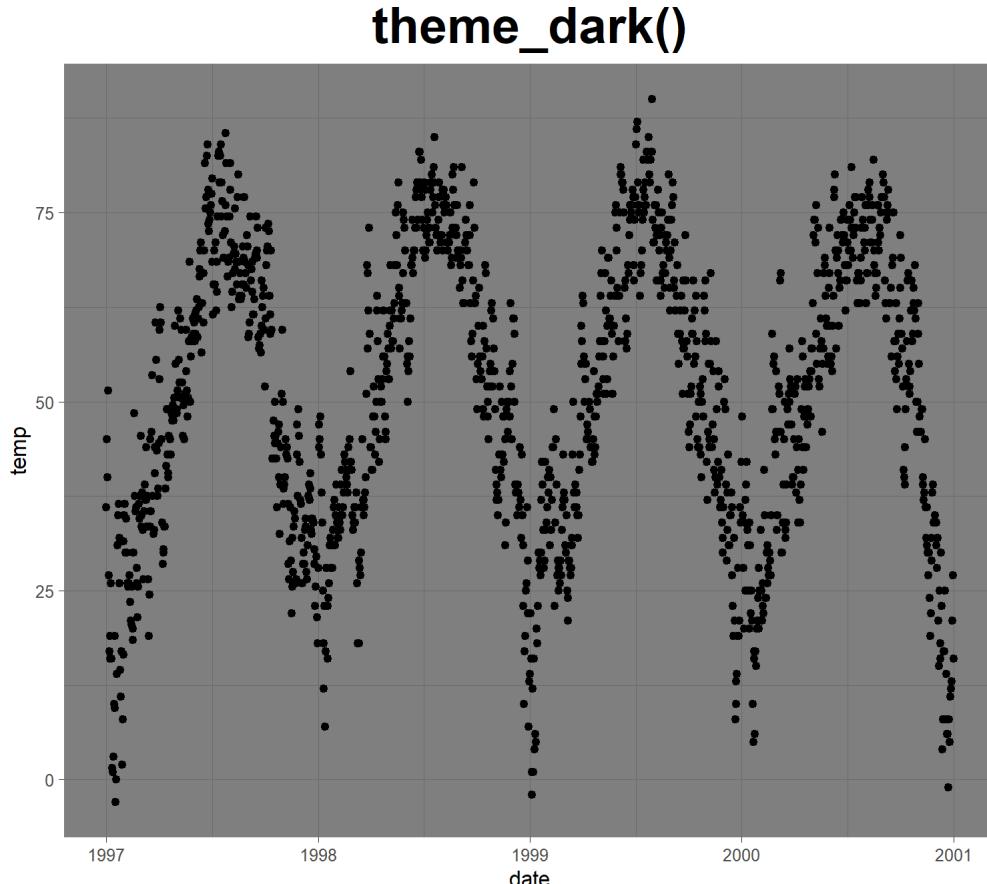
Spoiler: Themes!

Luckily, there are several built-in themes in `ggplot2` (and many many more in extension packages):



Spoiler: Themes!

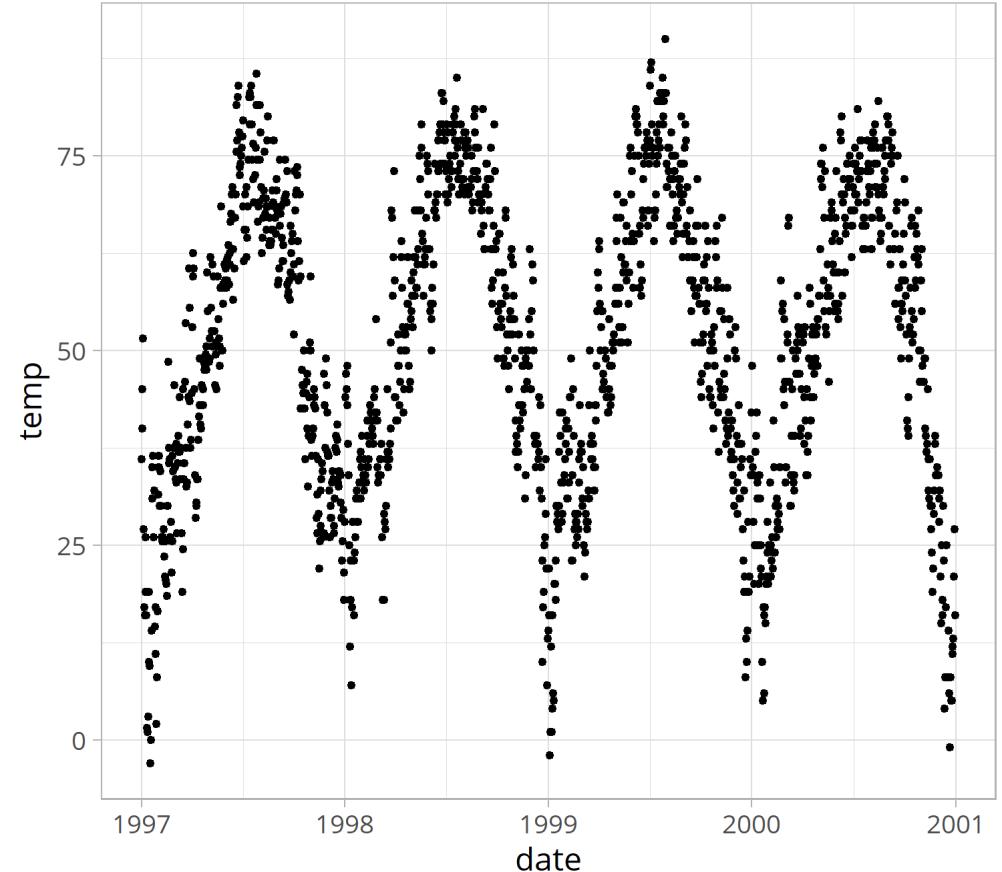
Luckily, there are several built-in themes in `ggplot2` (and many many more in extension packages):



Spoiler: Themes!

You can change the theme via `theme_set(theme_name())` - I am going to use `theme_light()` from now on:

```
theme_set(theme_light())  
  
ggplot(chic, aes(date, temp)) +  
  geom_point()
```



Your Turn!

- Turn our scatter plot into a line chart and into a bar chart!
- What's the difference between `geom_path()` and `geom_line()`?
- Create a box plot of temperature per date.
- What is the problem? How could you find out why this is happening?
- Try to create box plots as you would expect them.
- Bonus: Choose the built-in theme you like the most!

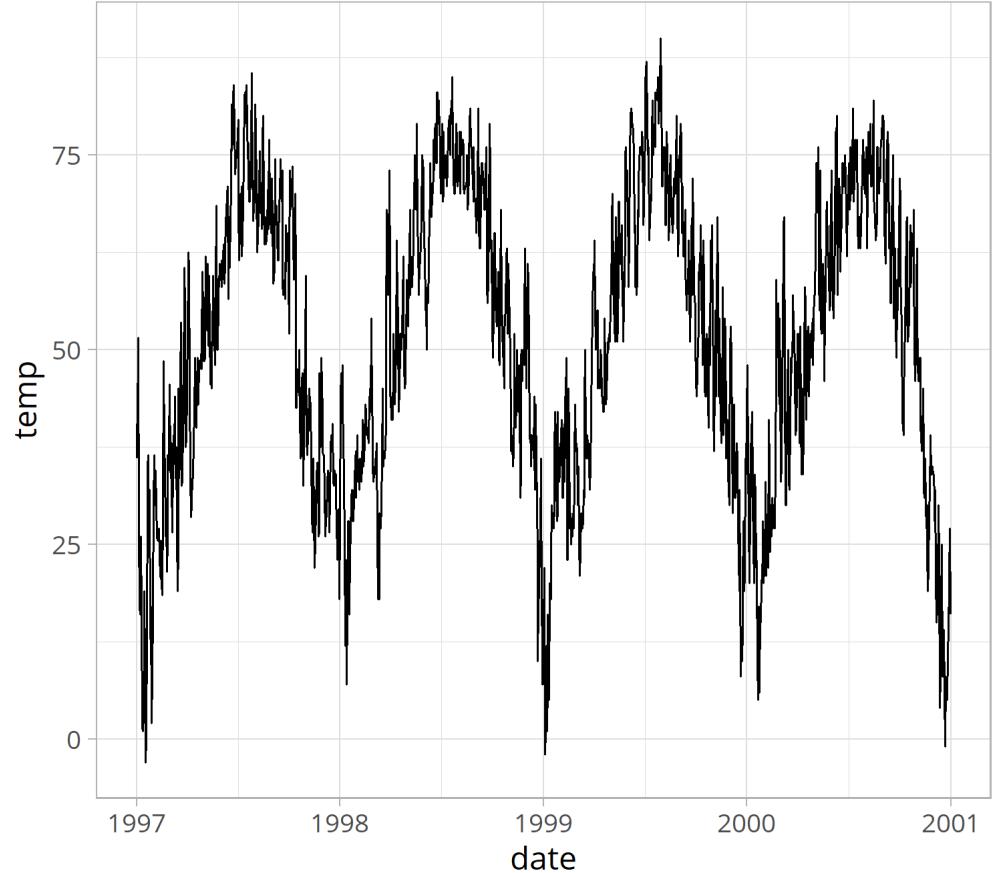
Your Turn!

- Turn our scatter plot into a line chart and into a bar chart!
- What's the difference between `geom_path()` and `geom_line()`?
- Create a box plot of temperature per date.
- What is the problem? How could you find out why this is happening?
- Try to create box plots as you would expect them.
- Hint: You need to specify one categorical variable!
- Bonus: Choose the built-in theme you like the most!

geom_line()

Our scatter plot as a line plot:

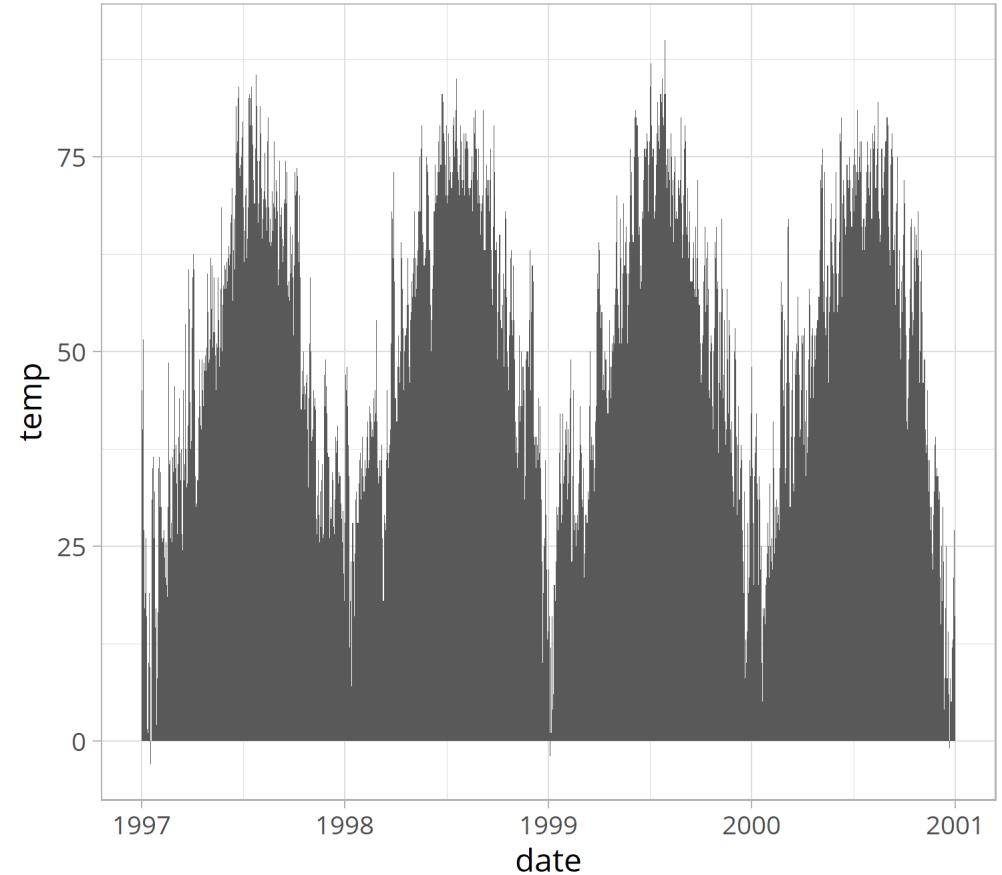
```
ggplot(chic, aes(date, temp)) +  
  geom_line()
```



geom_line()

... or as a bar plot:

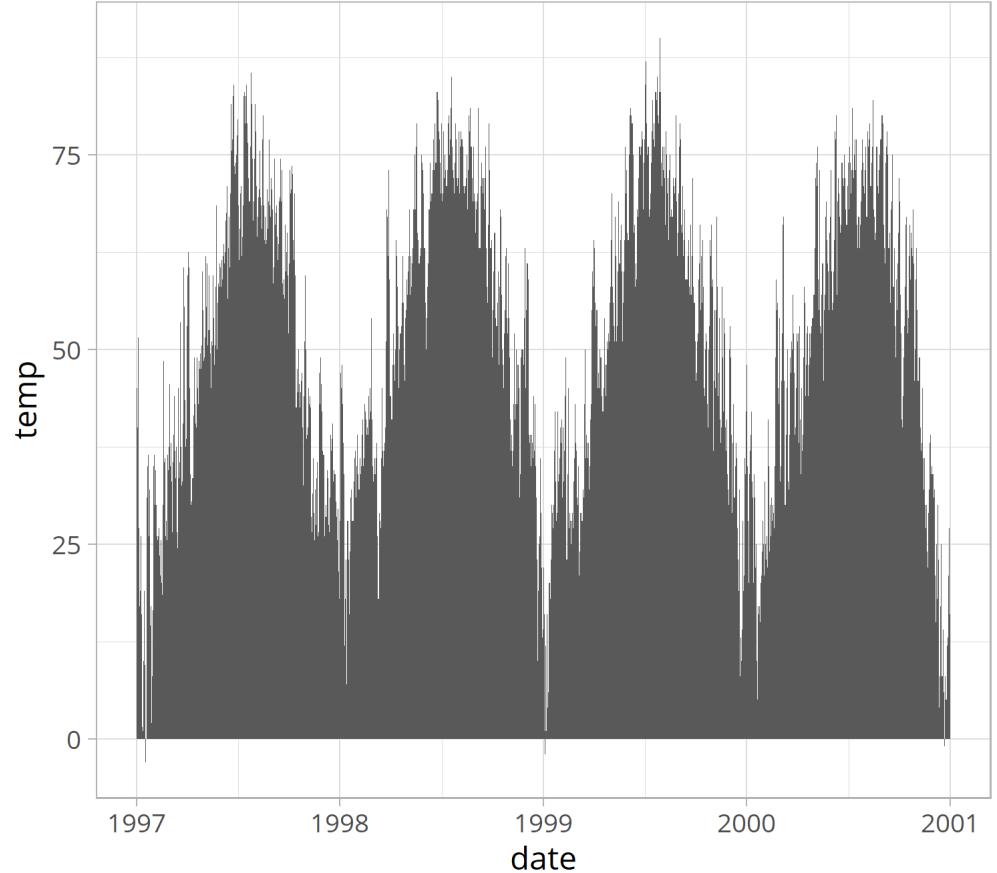
```
ggplot(chic, aes(date, temp)) +  
  geom_bar(stat = "identity")
```



geom_line()

... or as a bar plot:

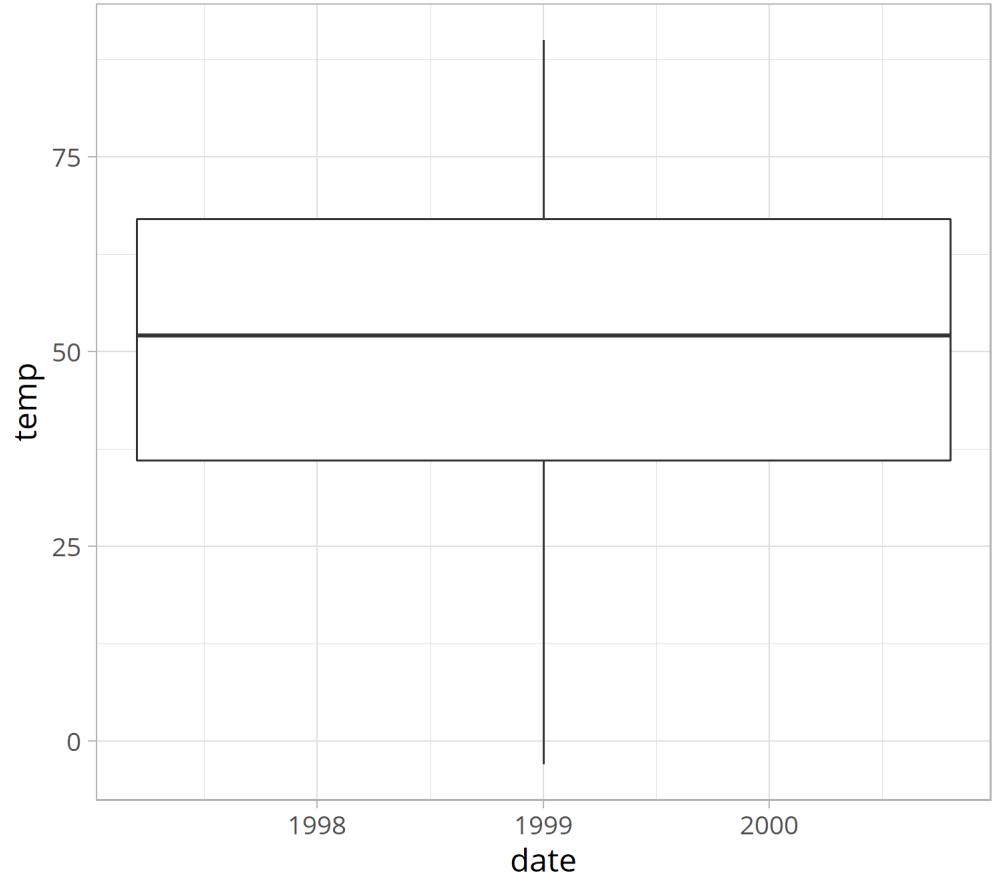
```
ggplot(chic, aes(date, temp)) +  
  geom_col()
```



geom_boxplot()

... or a box and whiskers plot:

```
ggplot(chic, aes(date, temp)) +  
  geom_boxplot()
```

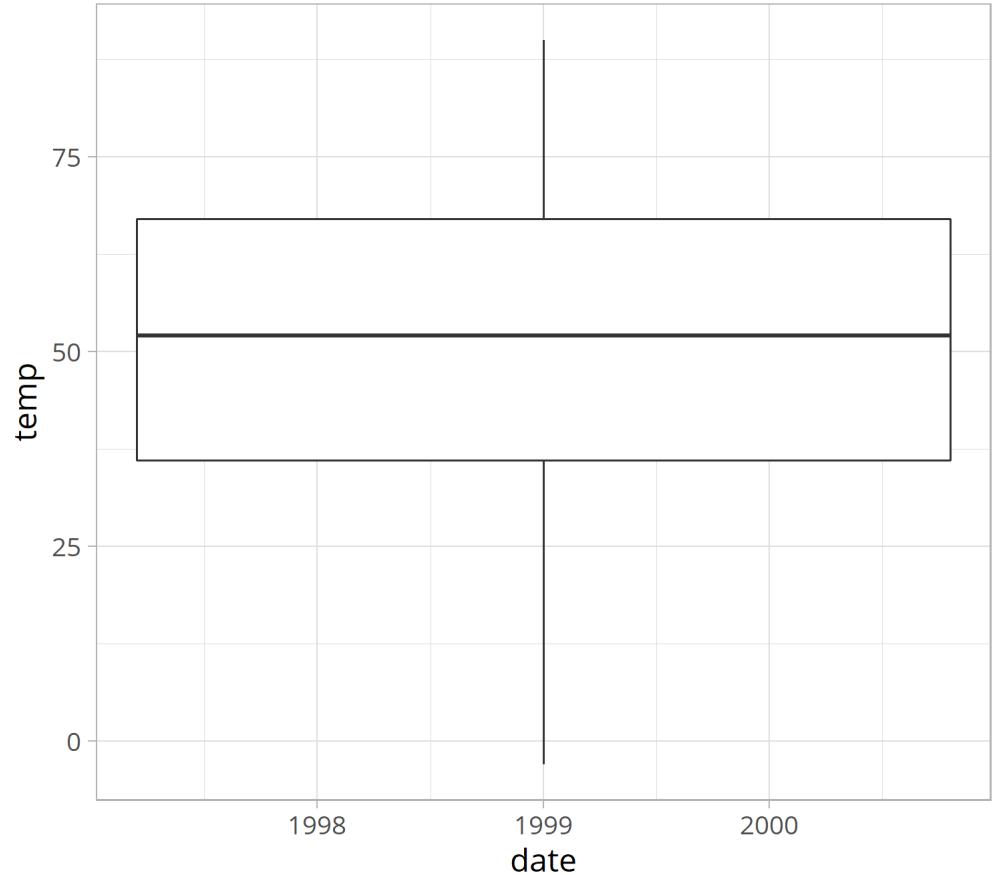


geom_boxplot()

... or a box and whiskers plot:

```
ggplot(chic, aes(date, temp)) +  
  geom_boxplot()
```

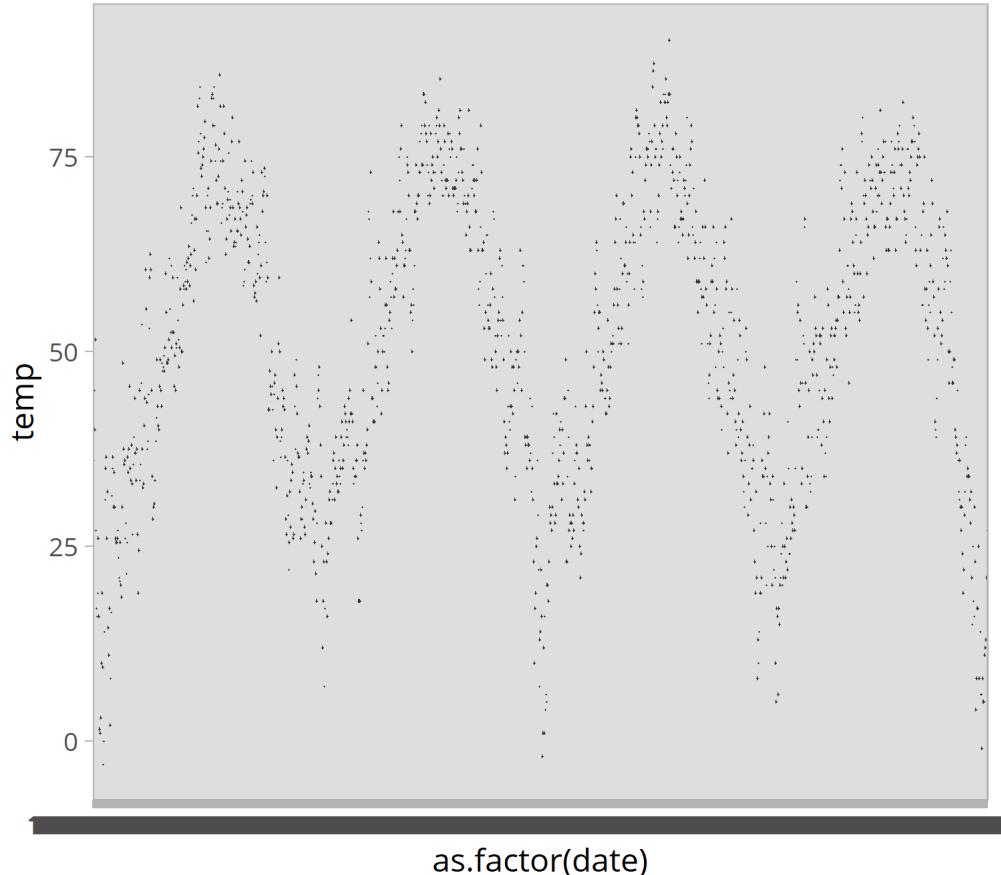
What's going on here?!



geom_boxplot()

We need to specify the variable as categorial (`as.factor(date)`), not as continuous (`date`):

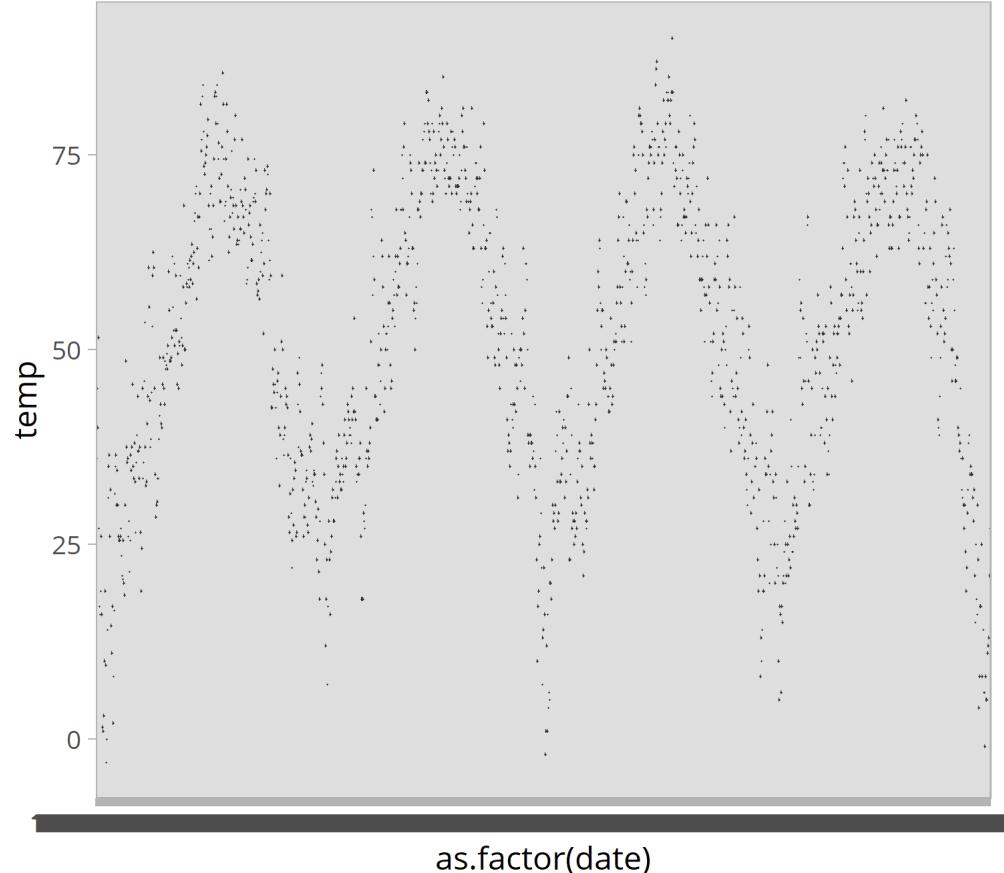
```
ggplot(chic, aes(as.factor(date), temp))  
  geom_boxplot()
```



geom_boxplot()

We need to specify the variable as categorial (`as.factor(date)`), not as continuous (`date`):

```
ggplot(chic, aes(as.factor(date), temp))  
  geom_boxplot()
```

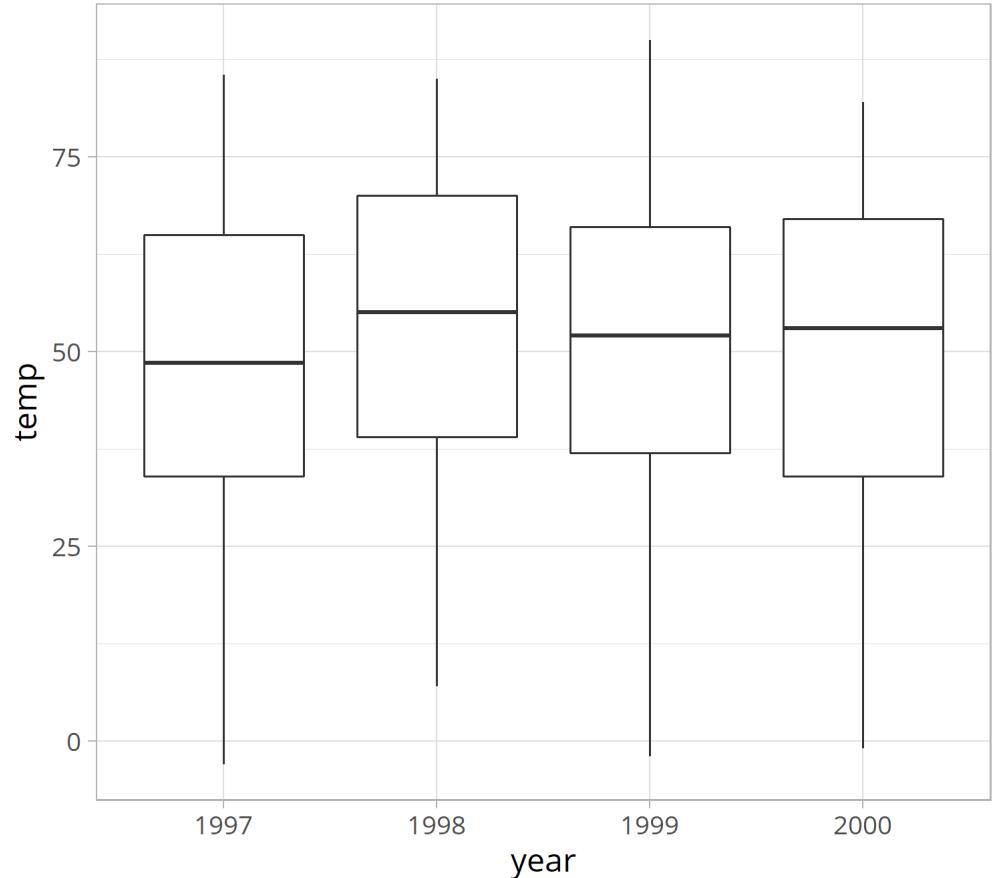


But - why are there no box plots?

geom_boxplot()

We need to specify the variable as categorial (e.g. `year` or `season`), not as continuous (`date`):

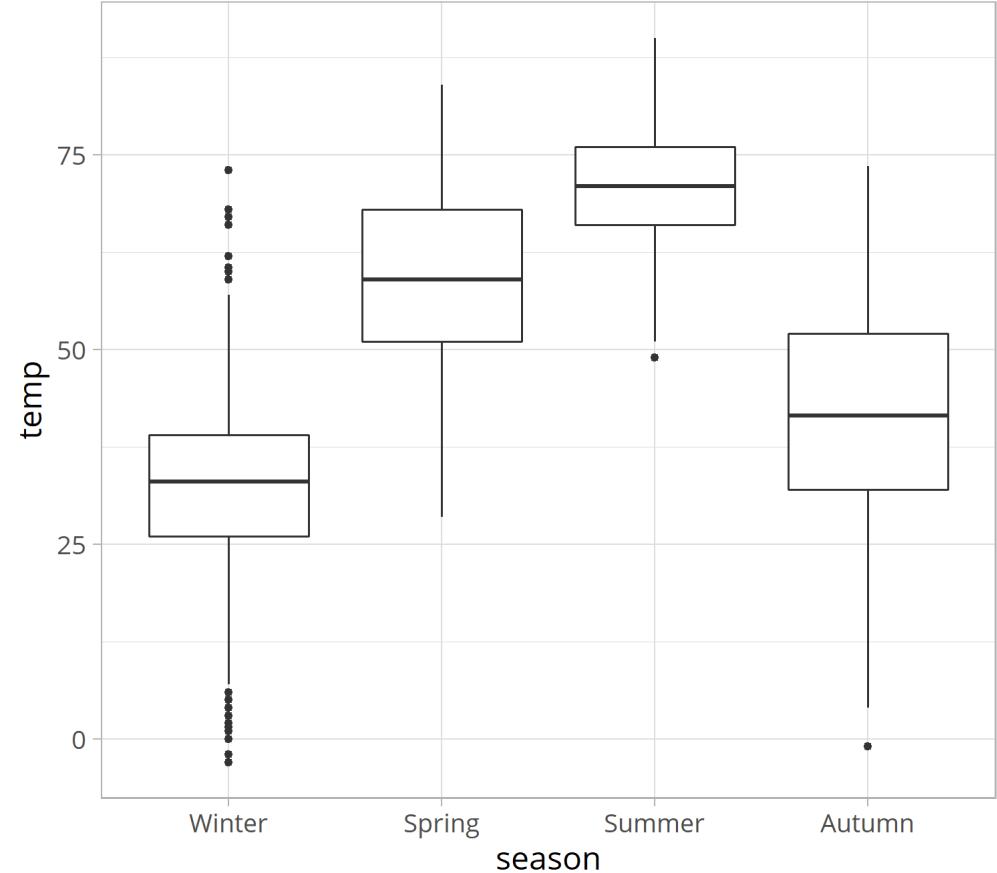
```
ggplot(chic, aes(year, temp)) +  
  geom_boxplot()
```



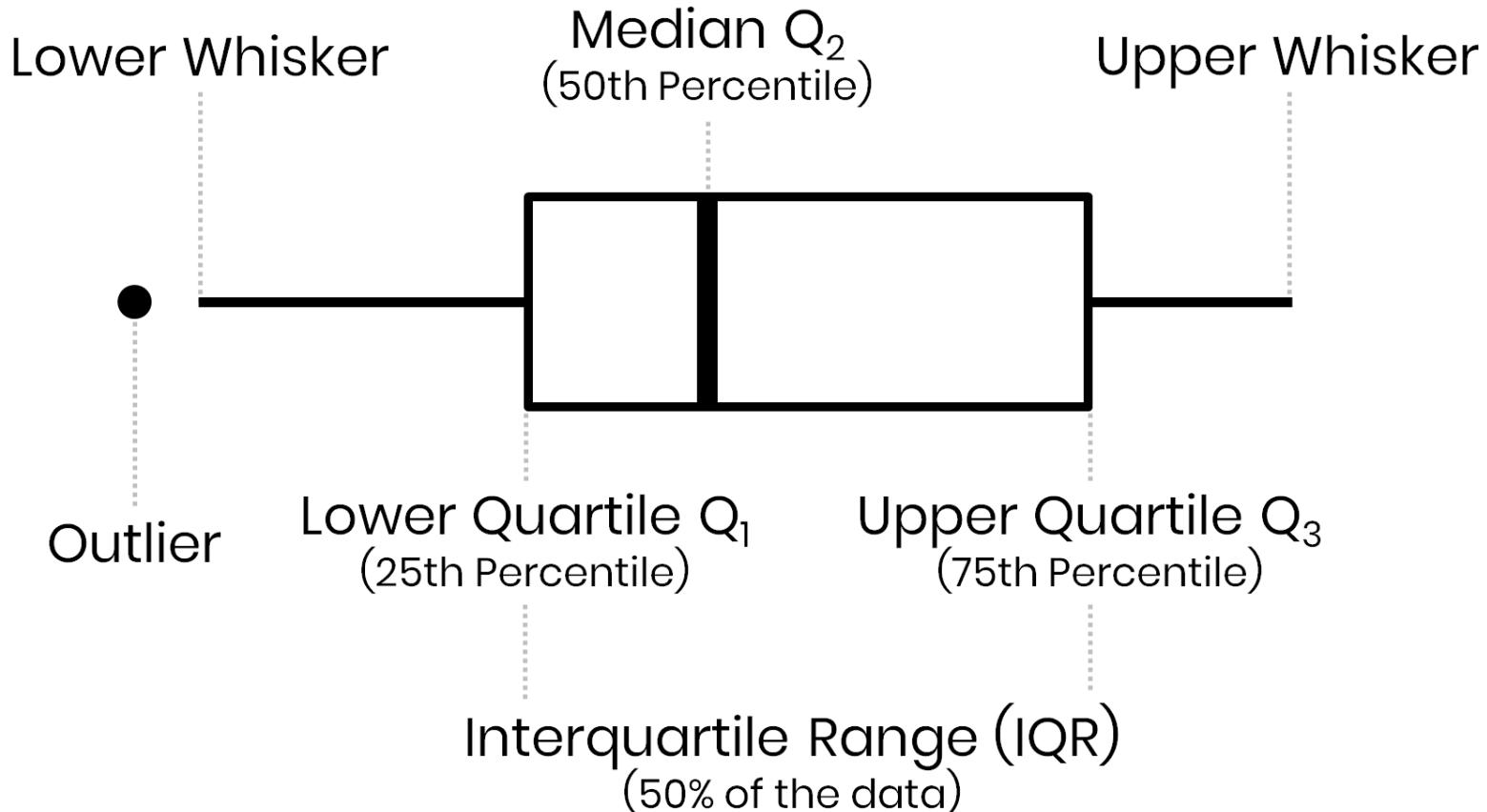
geom_boxplot()

We need to specify the variable as categorial (e.g. `year` or `season`), not as continuous (`date`):

```
ggplot(chic, aes(season, temp)) +  
  geom_boxplot()
```



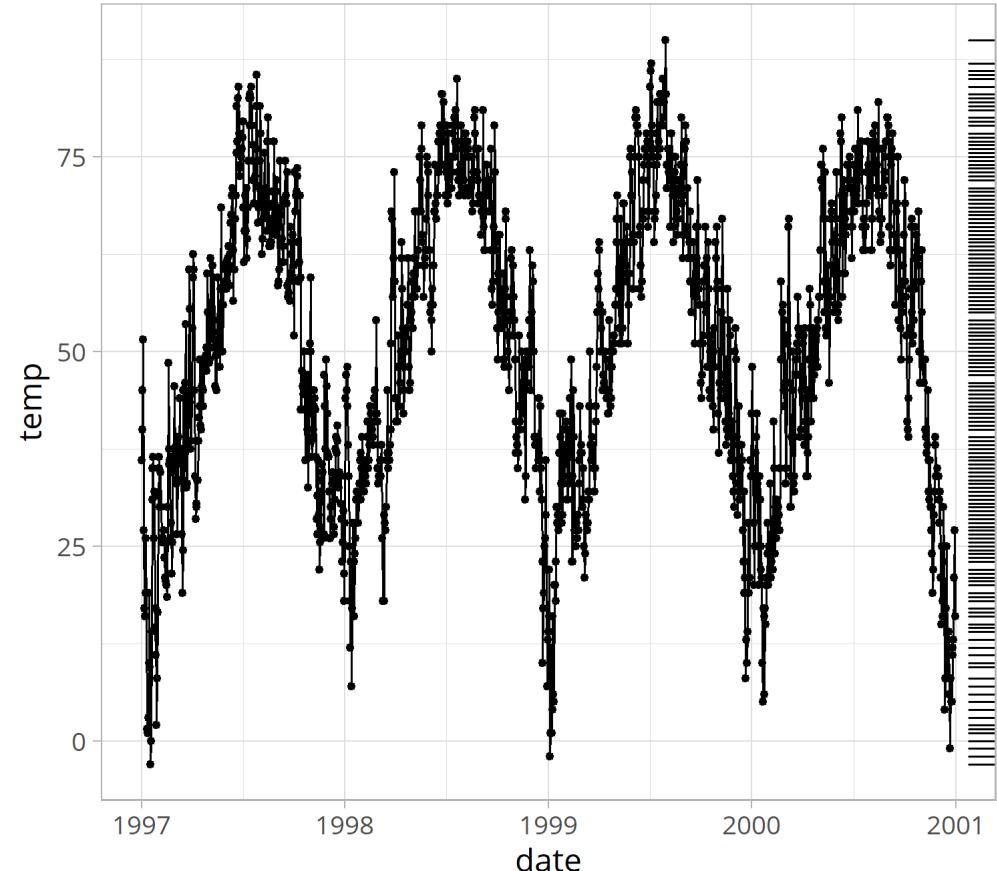
Box-and-Whisker Plots



Multiple `geom_*`'s

Other layers can be added to an existing plot - a line and a rug representation for example:

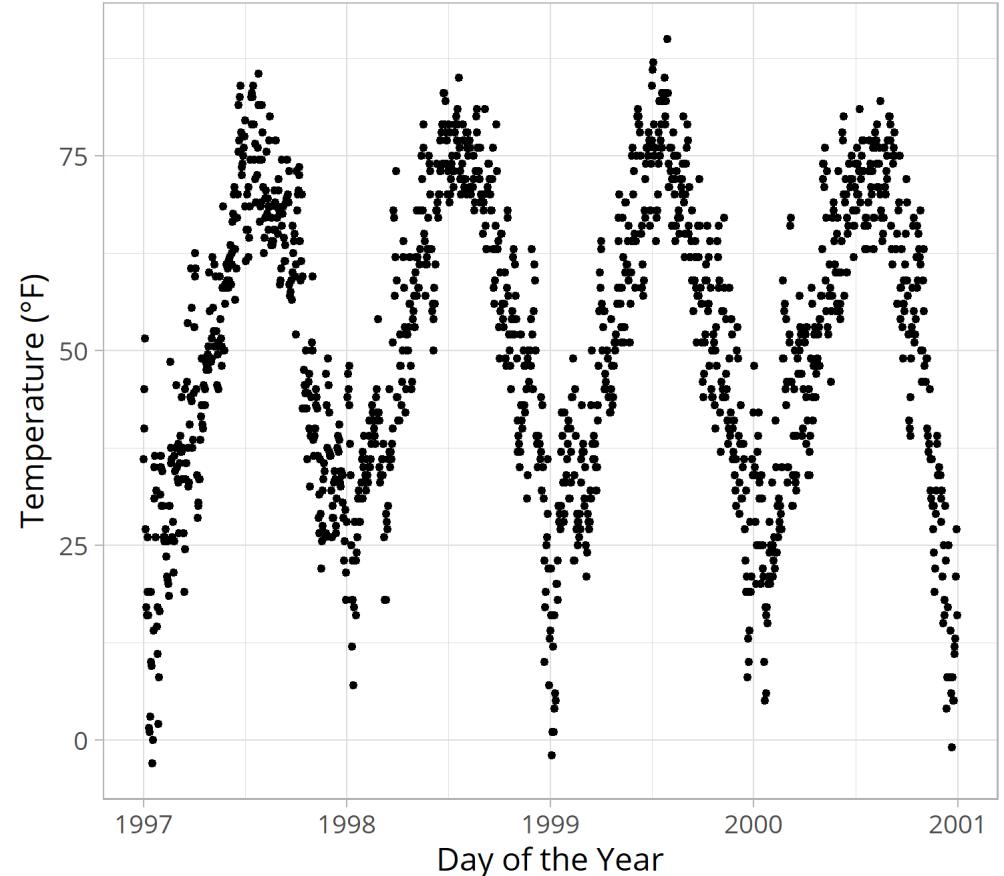
```
ggplot(chic, aes(date, temp)) +  
  geom_point() +  
  geom_line() +  
  geom_rug(sides = "r")
```



Add Labels

You can change the axes titles by adding `xlab()` and `ylab()` to your ggplot:

```
ggplot(chic, aes(date, temp)) +  
  geom_point() +  
  xlab("Day of the Year") +  
  ylab("Temperature (°F)")
```

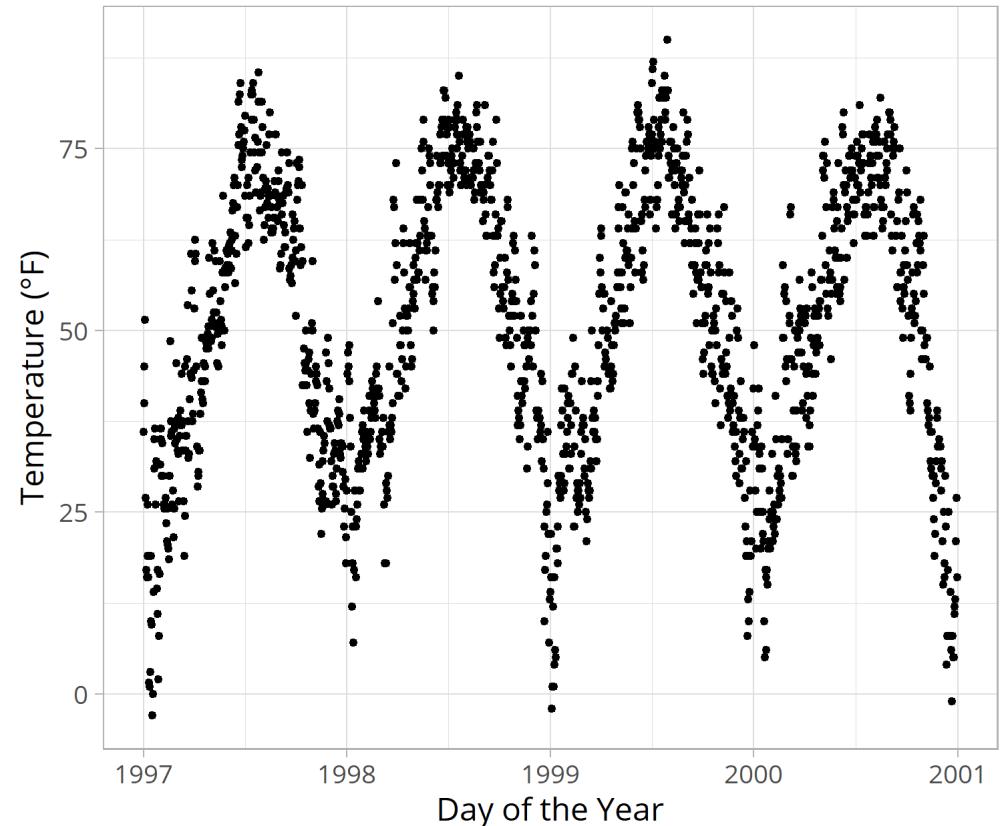


Add Labels

... and a title by adding `ggtitle()`:

```
ggplot(chic, aes(date, temp)) +  
  geom_point() +  
  xlab("Day of the Year") +  
  ylab("Temperature (°F)") +  
  ggtitle("Seasonal Change of Temperature")
```

Seasonal Change of Temperatures in Chicago



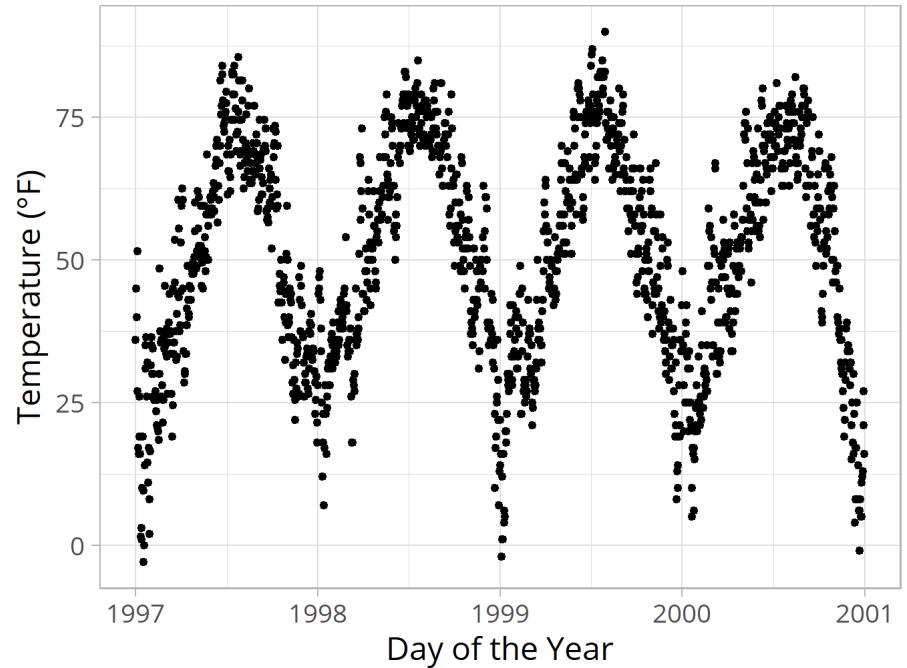
Add Labels

Using `labs()` you can also change all and more in one go:

```
ggplot(chic, aes(date, temp)) +  
  geom_point() +  
  labs(  
    x = "Day of the Year",  
    y = "Temperature (°F)",  
    title = "Seasonal Change of Temperature",  
    subtitle = "Daily temperatures (°F) in Chicago, IL",  
    caption = "Data: National Morbidity and Mortality Air Pollution Study (NMMAPS)",  
    tag = "Fig. 1"  
)
```

Fig. 1

Seasonal Change of Temperatures in Chicago
Daily temperatures (°F) in the city of Chicago, IL,
measured between 1997 and 2001



Data: National Morbidity and Mortality Air Pollution Study (NMMAPS)

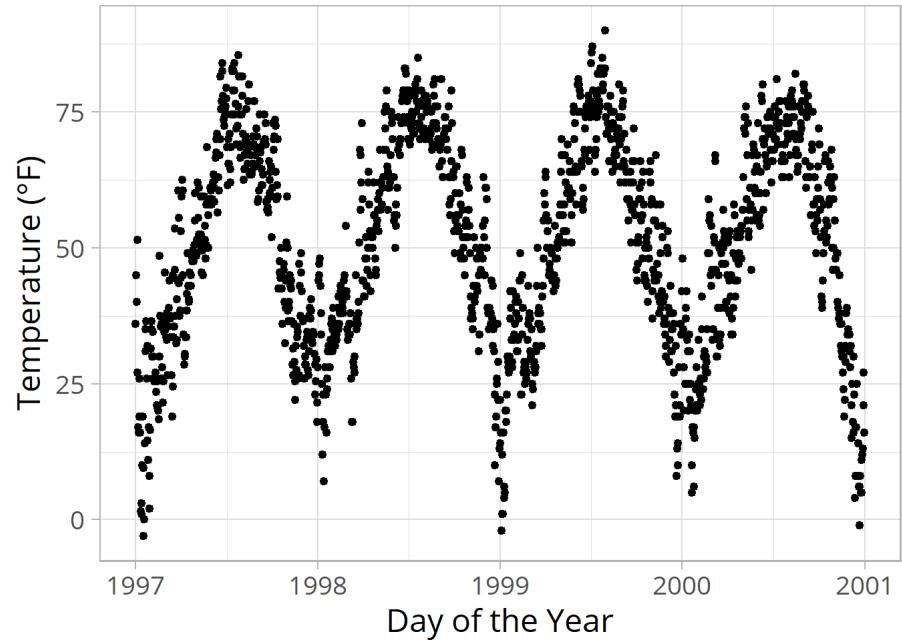
Add Labels

Using `labs()` you can also change all and more in one go:

```
ggplot(chic, aes(date, temp)) +  
  geom_point() +  
  labs(  
    x = "Day of the Year",  
    y = "Temperature (°F)",  
    title = "Seasonal Change of Temperatu  
    subtitle = "Daily temperatures (°F) i  
    caption = "\nData: National Morbidity  
    tag = "Fig. 1"  
)
```

Fig. 1

Seasonal Change of Temperatures in Chicago
Daily temperatures (°F) in the city of Chicago, IL,
measured between 1997 and 2001

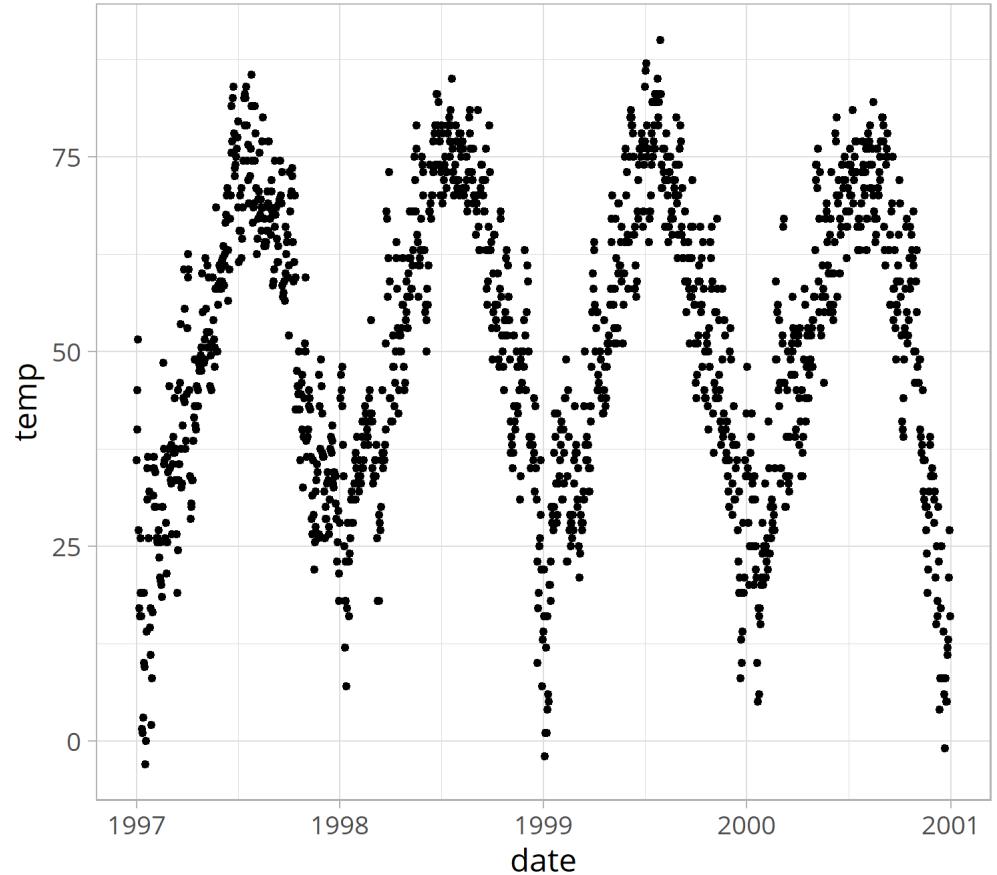


Data: National Morbidity and Mortality Air Pollution Study (NMMAPS)

ggplots as Object

You can assign ggplot to an object name:

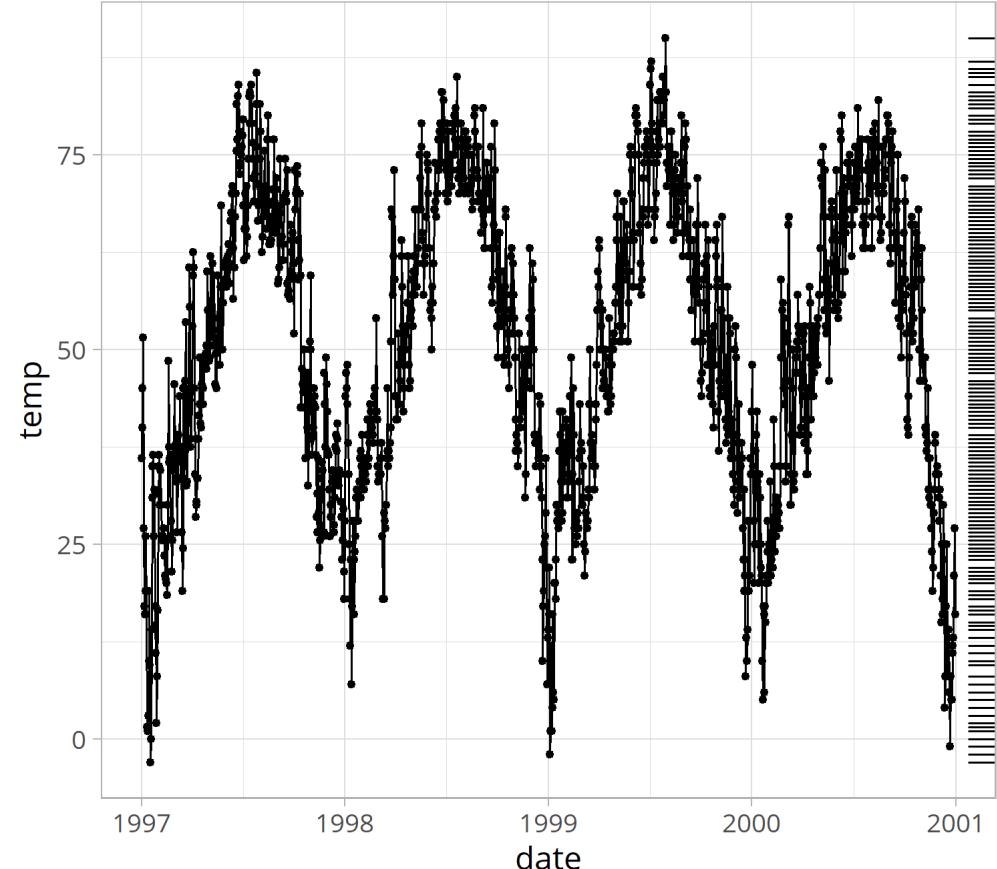
```
g <- ggplot(chic, aes(date, temp)) +  
  geom_point()  
  
gq
```



ggplots as Object

... and add layers afterwards:

```
g <- ggplot(chic, aes(date, temp)) +  
  geom_point()  
  
g +  
  geom_line() +  
  geom_rug(sides = "r")
```



Saving a ggplot

You can export your plot via the `ggsave()` function:

```
ggsave(filename = "my_ggplot.pdf",
        width = 10, height = 7,
        device = cairo_pdf)

ggsave(filename = "my_ggplot.png",
        width = 10, height = 7,
        dpi = 700)
```



Raster Graphic



Vector Graphic

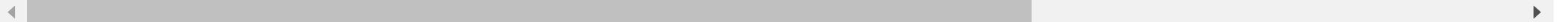
Source: canva.com

Your Turn!

- Import the data on password strength from *Information is Beautiful* from my GitHub repository:
<https://raw.githubusercontent.com/Z3tt/ggplot-courses/master/data/passwords.csv>
- Have a look at the raw data (both in the browser and R)
- Investigate the data set via summary functions.
- Create two plots with geom's of your choice to answer the following:
 - Which password category is ranked the lowest and which the highest?
 - Is there a password category that is more common than others?
- Export the plot with a nice aspect ratio.

Your Turn!

```
passwords <- read_csv("https://raw.githubusercontent.com/rfordatascience/tidyTuesday/m  
glimpse(passwords)  
## Observations: 507  
## Variables: 9  
## $ rank <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15...  
## $ password <chr> "password", "123456", "12345678", "1234", "qwerty...  
## $ category <chr> "password-related", "simple-alphanumeric", "simpl...  
## $ value <dbl> 6.91, 18.52, 1.29, 11.11, 3.72, 1.85, 3.72, 6.91,...  
## $ time_unit <chr> "years", "minutes", "days", "seconds", "days", "m...  
## $ offline_crack_sec <dbl> 2.170e+00, 1.110e-05, 1.110e-03, 1.110e-07, 3.210...  
## $ rank_alt <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15...  
## $ strength <dbl> 8, 4, 4, 4, 8, 4, 8, 4, 7, 8, 8, 1, 32, 9, 9, 8, ...  
## $ font_size <dbl> 11, 8, 8, 8, 11, 8, 11, 11, 11, 4, 23, 12, ...
```



Your Turn!

```
#View(passwords)

summary(passwords[,c(1:6, 8)])
##      rank          password           category        value
## Min.   : 1.0  Length:507           Length:507       Min.   : 1.290
## 1st Qu.:125.8 Class :character    Class :character  1st Qu.: 3.430
## Median :250.5 Mode  :character    Mode  :character  Median : 3.720
## Mean   :250.5
## 3rd Qu.:375.2
## Max.   :500.0
## NA's   :7
##      time_unit      offline_crack_sec      strength
## Length:507           Min.   : 0.00000   Min.   : 0.000
## Class :character     1st Qu.: 0.00321   1st Qu.: 6.000
## Mode  :character     Median : 0.00321   Median : 7.000
##                   Mean   : 0.50001   Mean   : 7.432
##                   3rd Qu.: 0.08350   3rd Qu.: 8.000
##                   Max.   :29.27000  Max.   :48.000
##                   NA's   :7         NA's   :7
```

Your Turn!

```
range(passwords$rank)
## [1] NA NA
```

Your Turn!

```
range(passwords$rank)
## [1] NA NA

range(passwords$rank, na.rm = T)
## [1] 1 500
```

Your Turn!

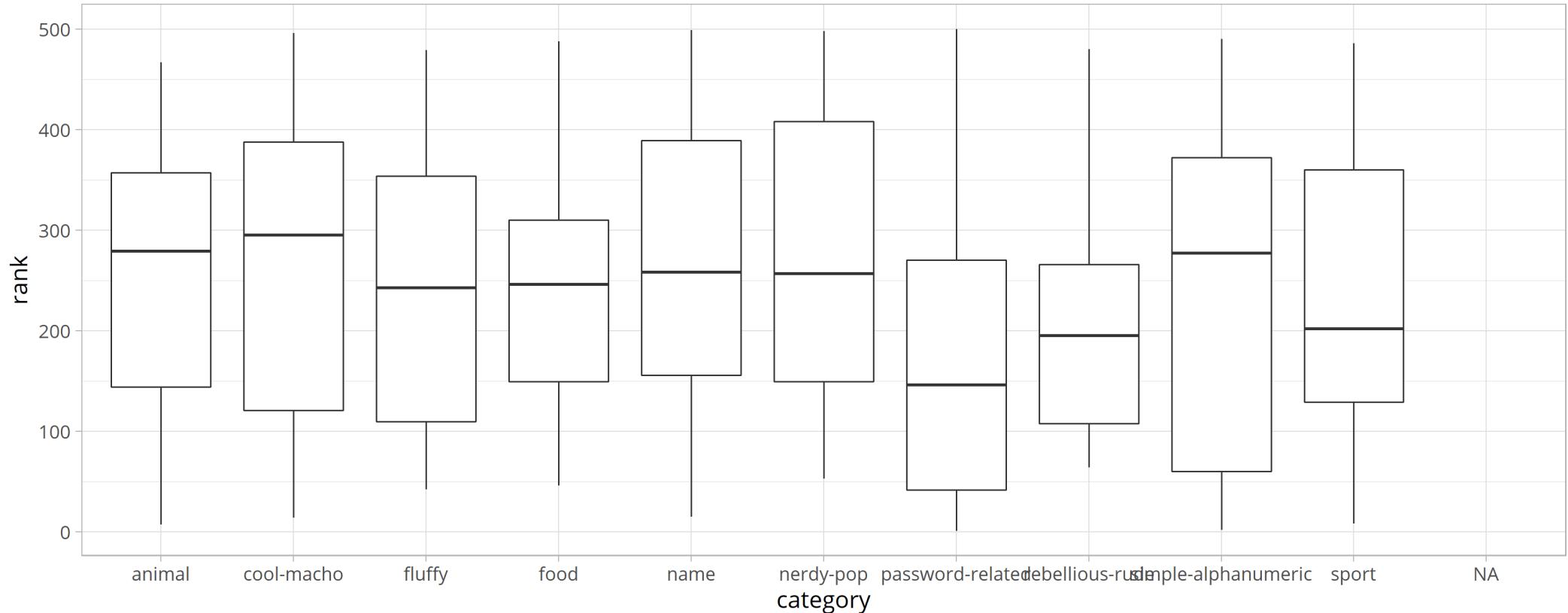
```
unique(passwords$category)
## [1] "password-related"      "simple-alphanumeric" "animal"
## [4] "sport"                  "cool-macho"          "name"
## [7] "fluffy"                 "food"                "nerdy-pop"
## [10] "rebellious-rude"        NA
```

Your Turn!

```
unique(passwords$password)
## [1] "password"    "123456"      "12345678"    "1234"        "qwerty"       "12345"
## [7] "dragon"       "baseball"     "football"     "letmein"      "monkey"       "696969"
## [13] "abc123"       "mustang"      "michael"      "shadow"       "master"       "jennifer"
## [19] "111111"       "2000"         "jordan"       "superman"    "harley"       "1234567"
## [25] "hunter"       "trustno1"     "ranger"       "buster"       "thomas"       "tigger"
## [31] "robert"        "soccer"       "batman"       "test"         "pass"         "killer"
## [37] "hockey"        "george"       "charlie"      "andrew"       "micelle"      "love"
## [43] "sunshine"      "jessica"      "6969"         "pepper"       "daniel"       "access"
## [49] "123456789"    "654321"       "joshua"       "maggie"       "starwars"     "silver"
## [55] "william"       "dallas"        "yankees"      "123123"      "ashley"       "666666"
## [61] "hello"          "amanda"        "orange"       "biteme"       "freedom"      "computer"
## [67] "sexy"           "thunder"       "nicole"       "ginger"       "heather"      "hammer"
## [73] "summer"         "corvette"     "taylor"       "austin"       "1111"         "merlin"
## [79] "matthew"        "121212"       "golfer"       "cheese"       "princess"     "martin"
## [85] "chelsea"        "patrick"      "richard"      "diamond"      "yellow"       "bigdog"
## [91] "secret"          "asdfgh"       "sparky"       "cowboy"       "camaro"       "anthony"
## [97] "matrix"          "falcon"        "iloveyou"     "bailey"       "guitar"       "jackson"
## [103] "purple"          "scooter"      "phoenix"      "aaaaaaa"      "morgan"       "tigers"
## [109] "porsche"        "mickey"        "maverick"     "cookie"       "nascar"       "peanut"
## [115] "justin"          "131313"       "money"        "samantha"    "parties"      "steelers"
## [121] "joseph"          "snoopy"        "boomer"       "whatever"    "iceman"       "smokey"
```

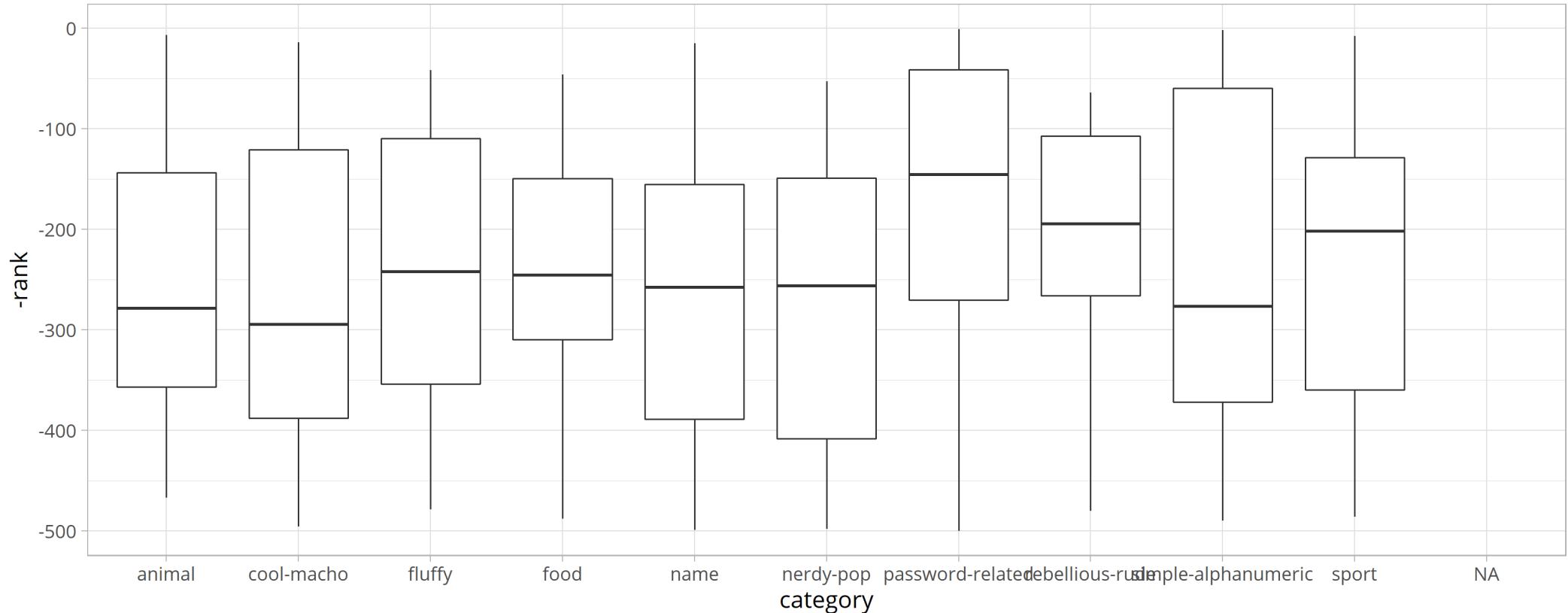
Your Turn!

```
ggplot(passwords, aes(category, rank)) +  
  geom_boxplot()
```



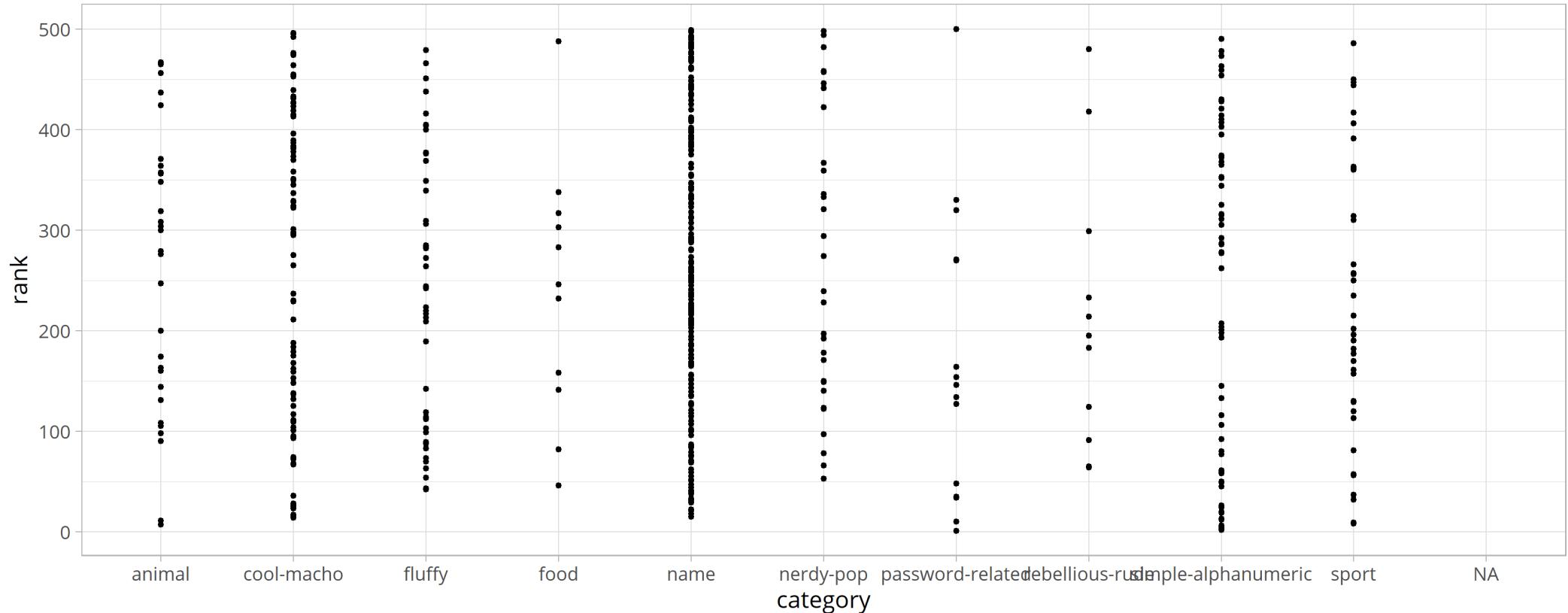
Your Turn!

```
ggplot(passwords, aes(category, -rank)) +  
  geom_boxplot()
```



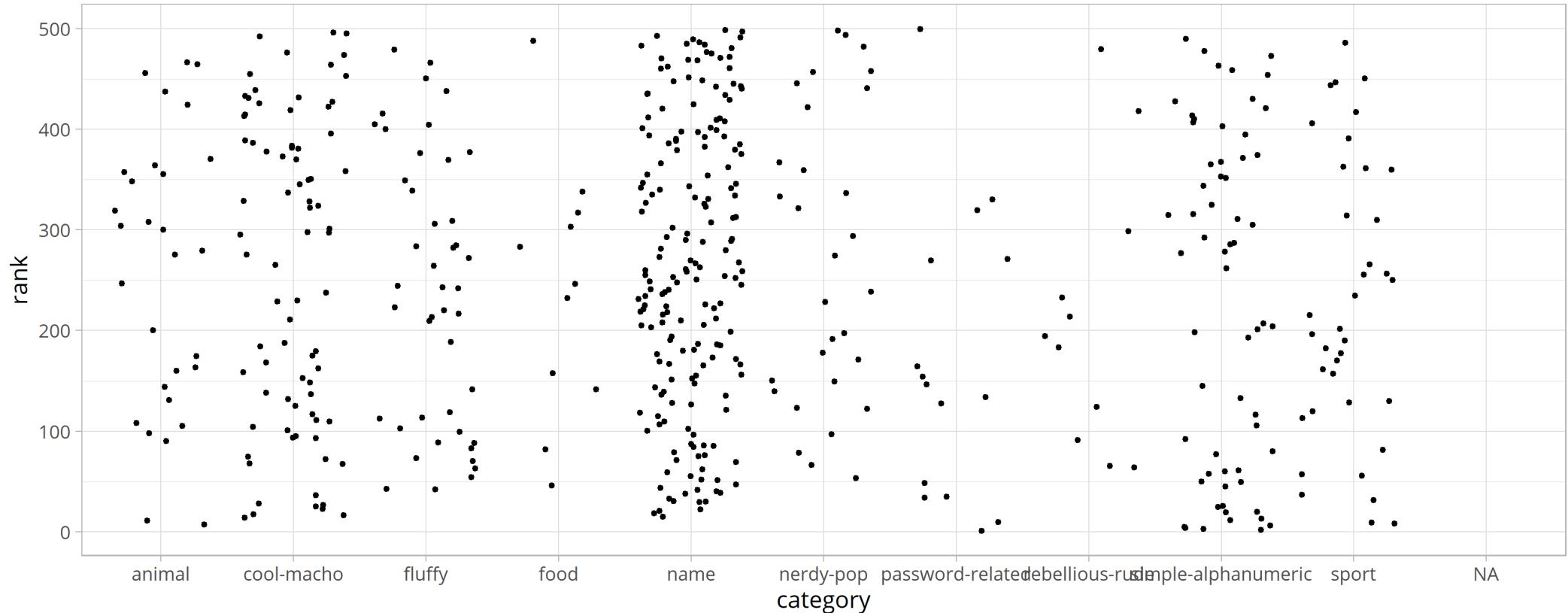
Your Turn!

```
ggplot(passwords, aes(category, rank)) +  
  geom_point()
```



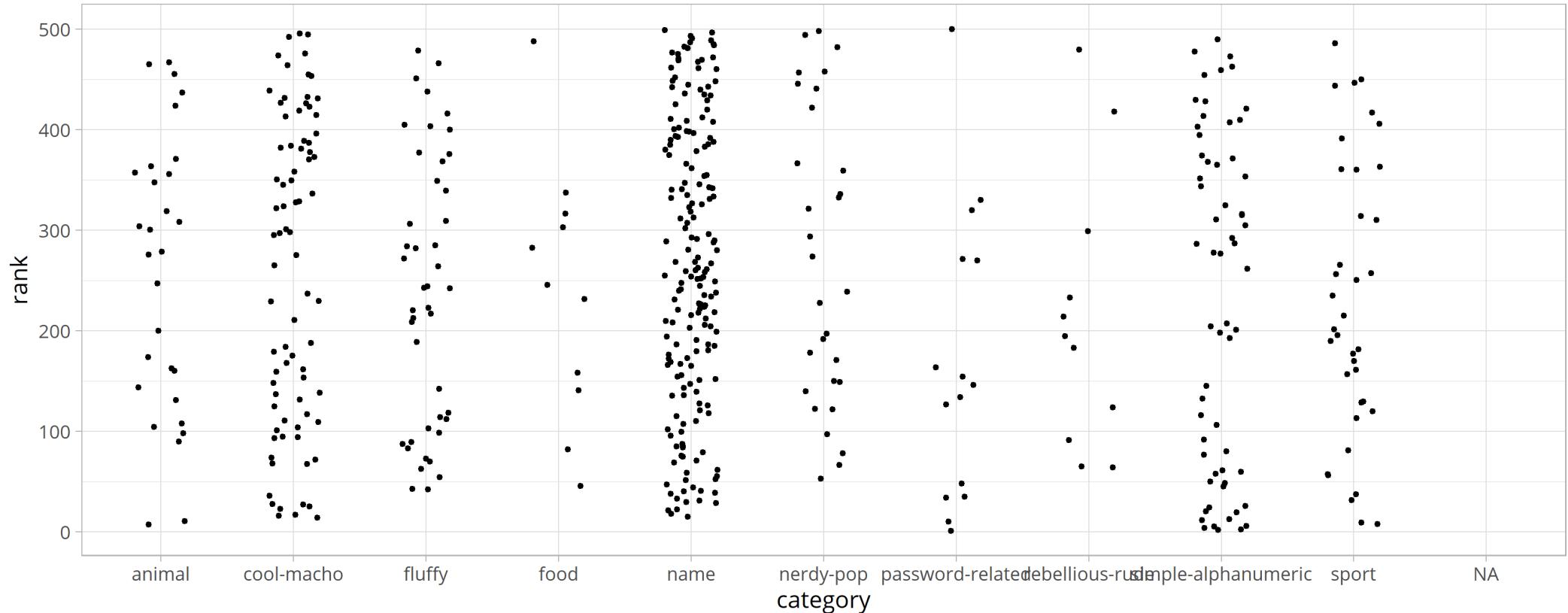
Your Turn!

```
ggplot(passwords, aes(category, rank)) +  
  geom_jitter()
```



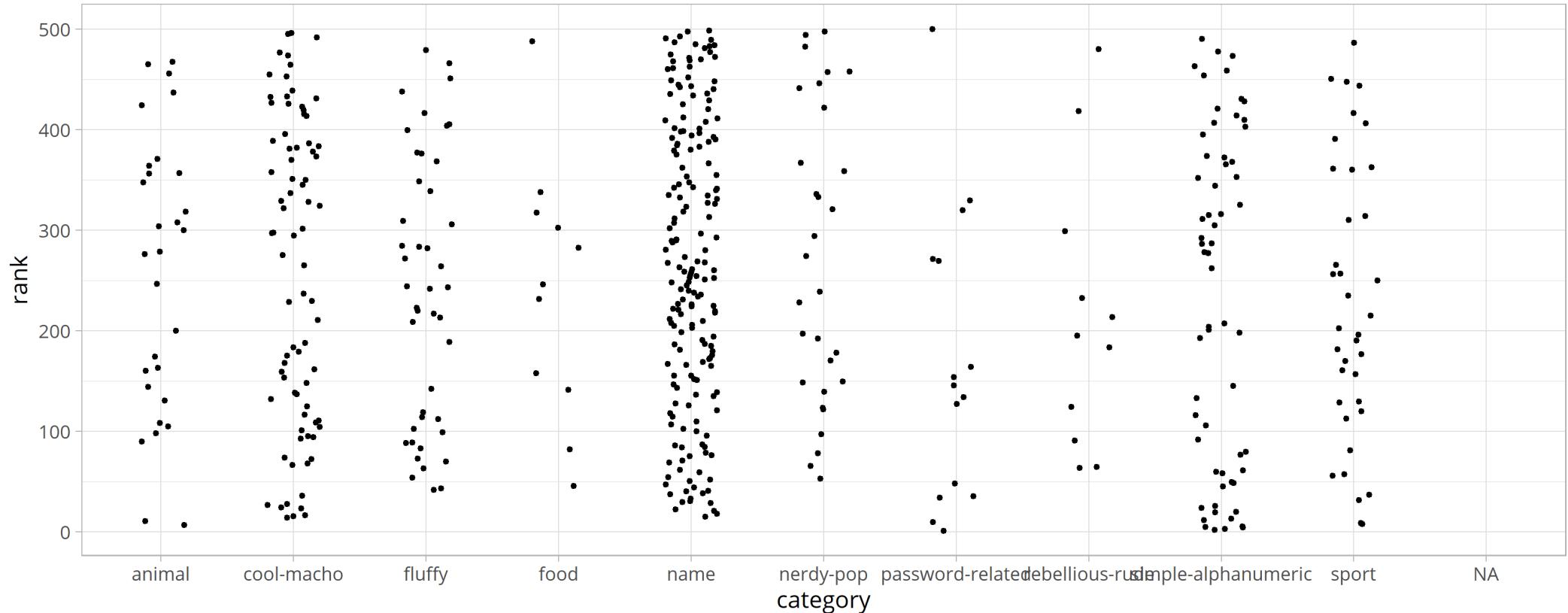
Your Turn!

```
ggplot(passwords, aes(category, rank)) +  
  geom_jitter(width = .2)
```



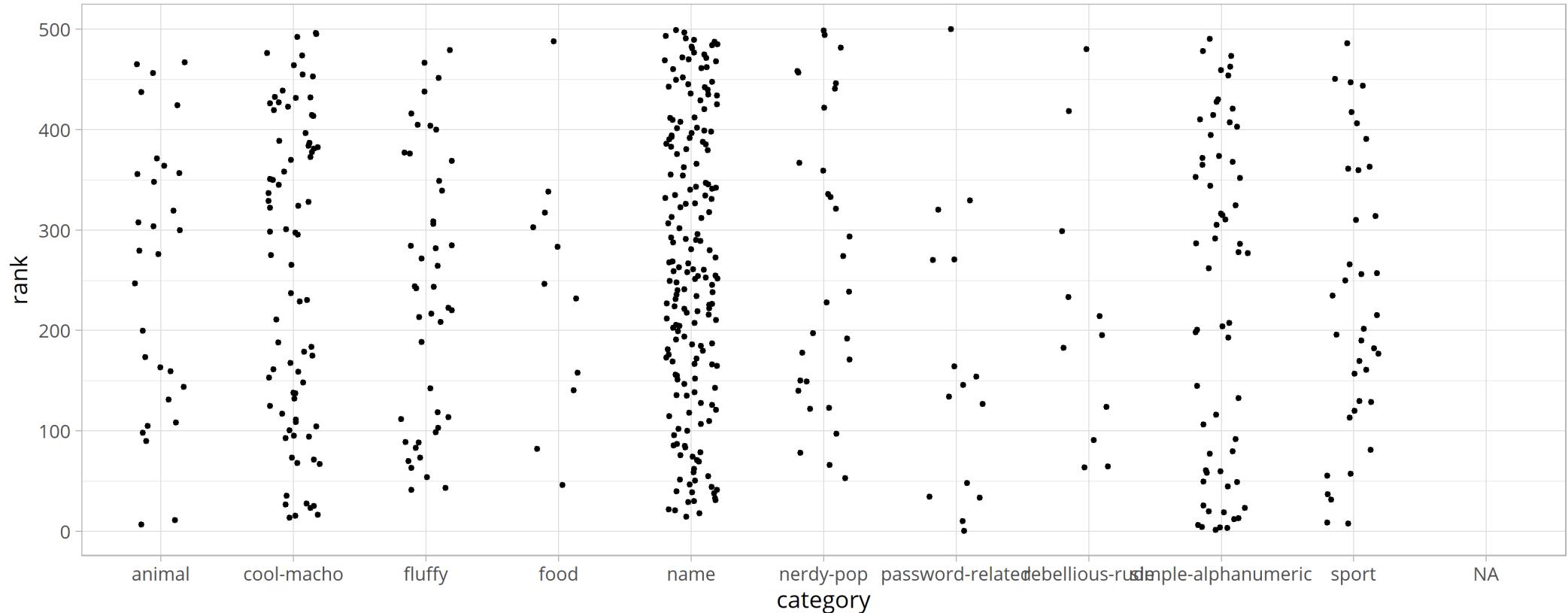
Your Turn!

```
ggplot(passwords, aes(category, rank)) +  
  geom_jitter(position = position_jitter(width = .2, seed = 1))
```



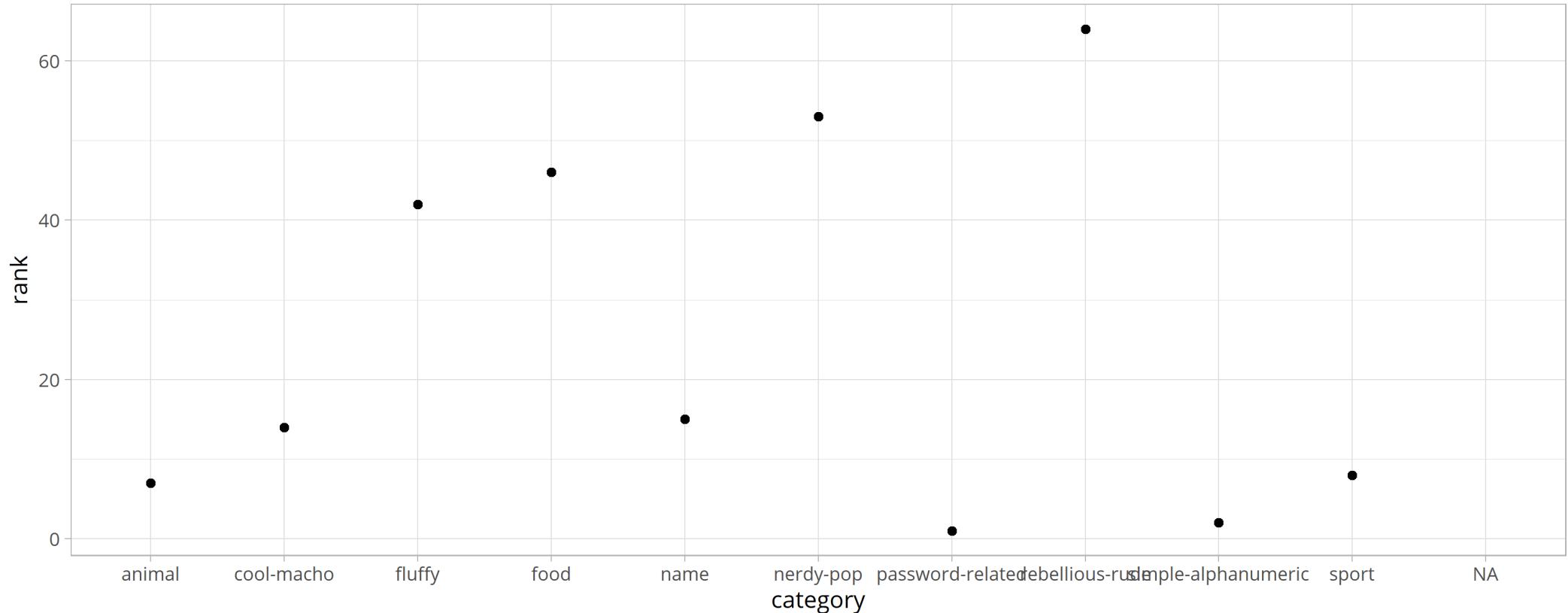
Your Turn!

```
ggplot(passwords, aes(category, rank)) +  
  geom_jitter(position = position_jitter(width = .2, seed = 2020))
```



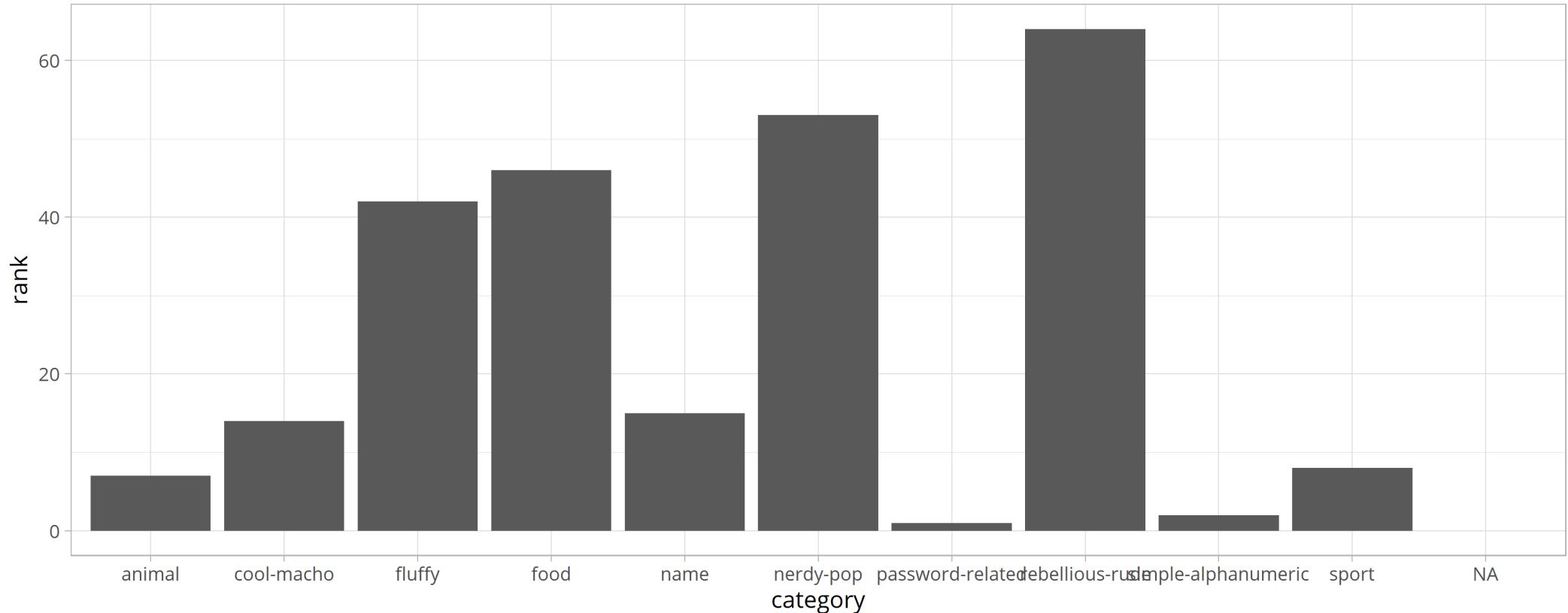
Your Turn!

```
ggplot(passwords, aes(category, rank)) +  
  stat_summary(fun = min)
```



Your Turn!

```
ggplot(passwords, aes(category, rank)) +  
  stat_summary(fun = min, geom = "col")
```



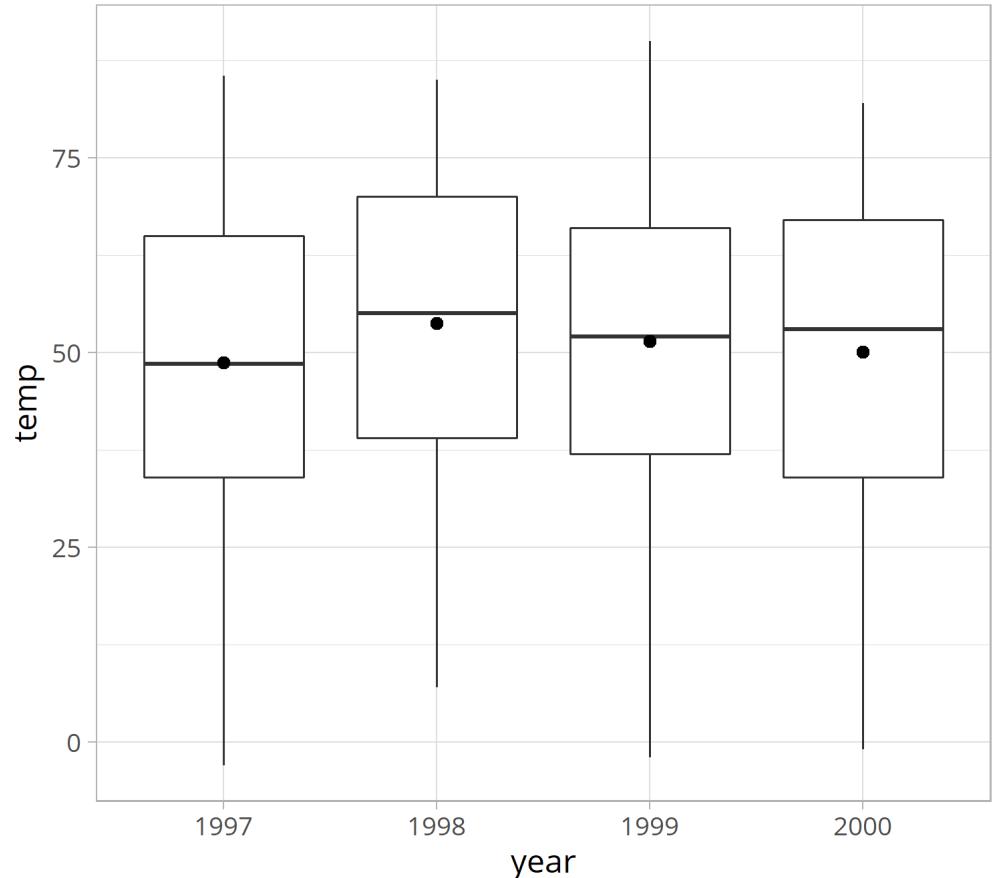
Layers

Statistical Layers: **stat_***()

stat_*

A handful of layers with attention to the statistical transformation rather than the visual appearance:

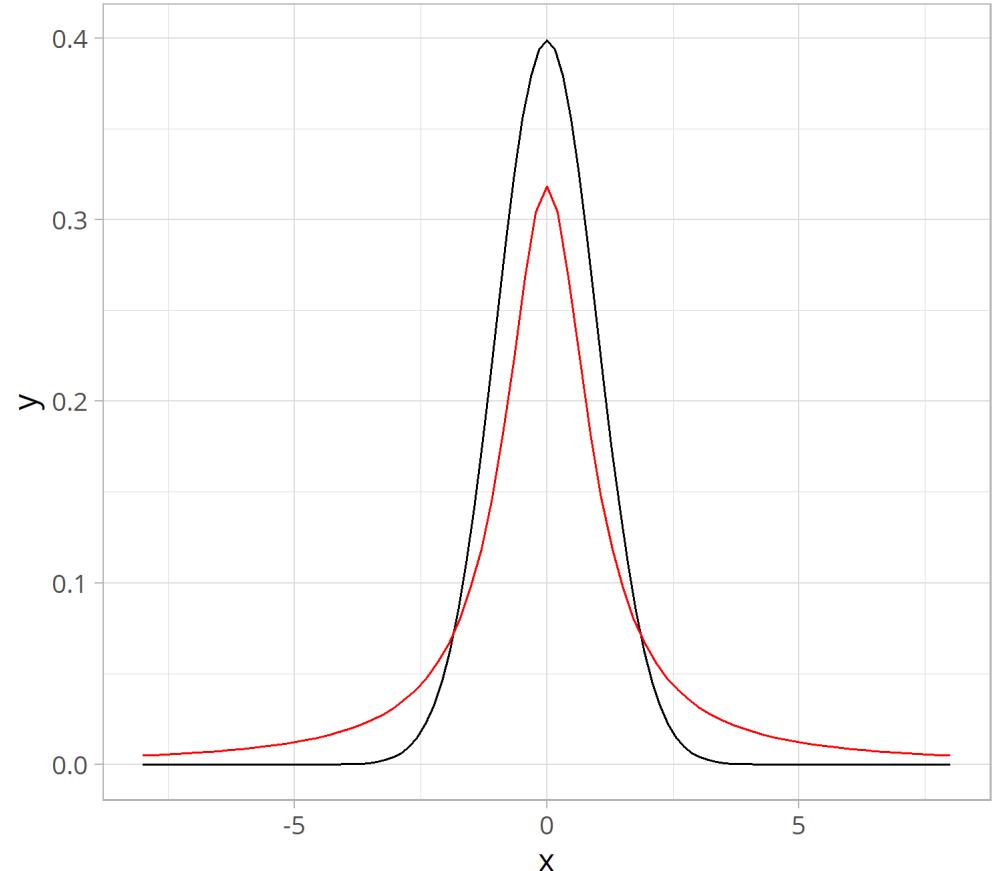
```
ggplot(chic, aes(year, temp)) +  
  geom_boxplot() +  
  stat_summary(fun = mean)  
## fun.y in the current CRAN version
```



stat_function()

`stat_function()` makes it easy to add a function to a plot, either continuous or discrete:

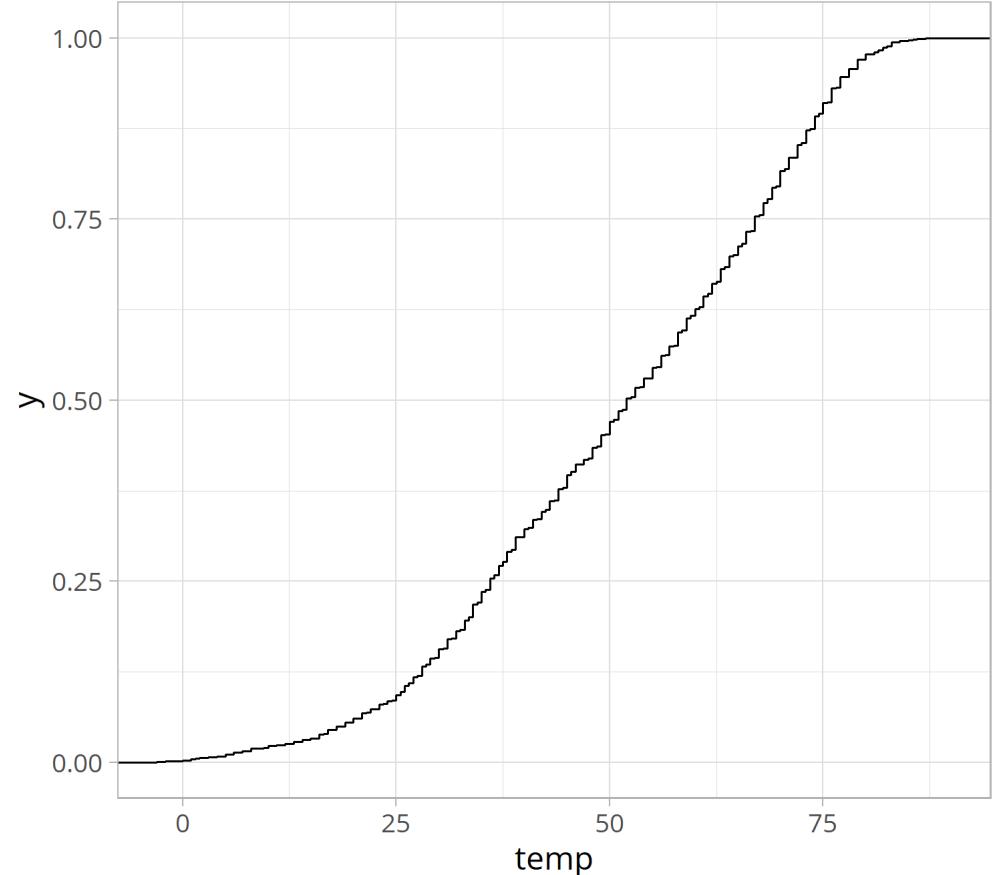
```
ggplot(tibble(x = c(-8, 8)), aes(x)) +  
  stat_function(fun = dnorm) +  
  stat_function(  
    fun = dcauchy,  
    color = "red",  
    n = 75  
)
```



stat_ecdf()

You can also easily plot the empirical cumulative distribution function (ECDF) of a variable:

```
ggplot(chic, aes(temp)) +  
  stat_ecdf()
```

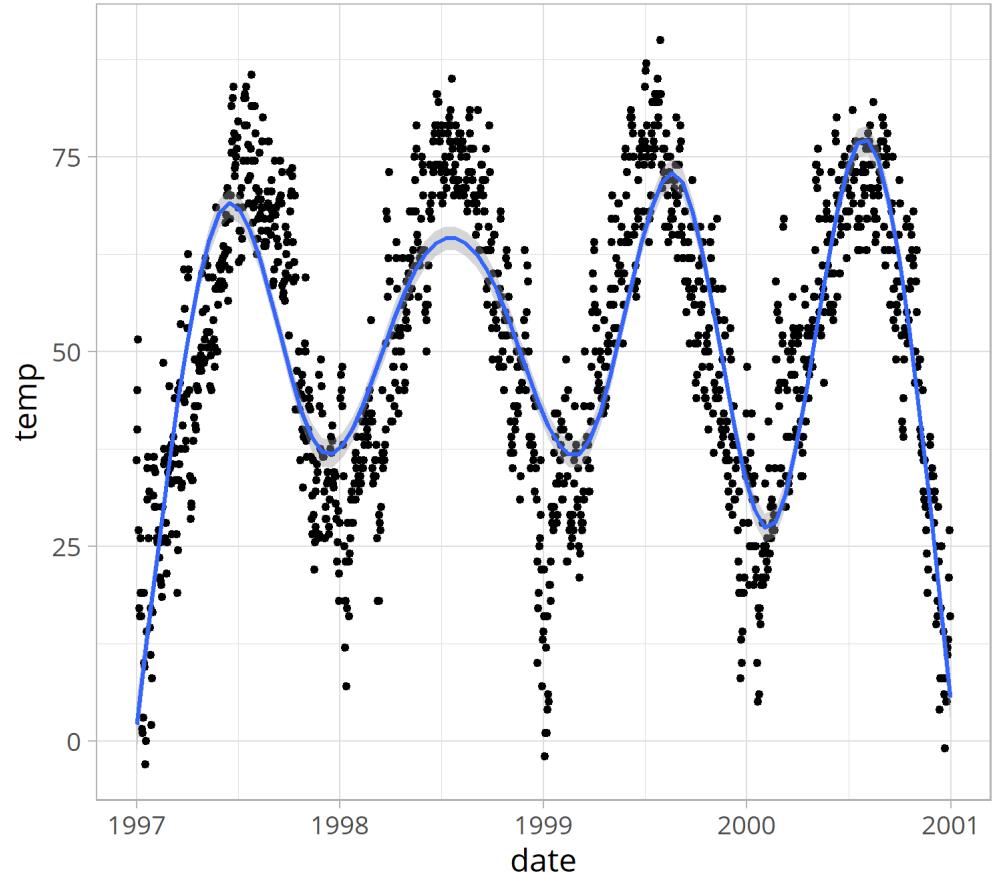


stat_smooth()

You can directly add smoothed conditional means:

```
ggplot(chic, aes(date, temp)) +  
  geom_point() +  
  stat_smooth()
```

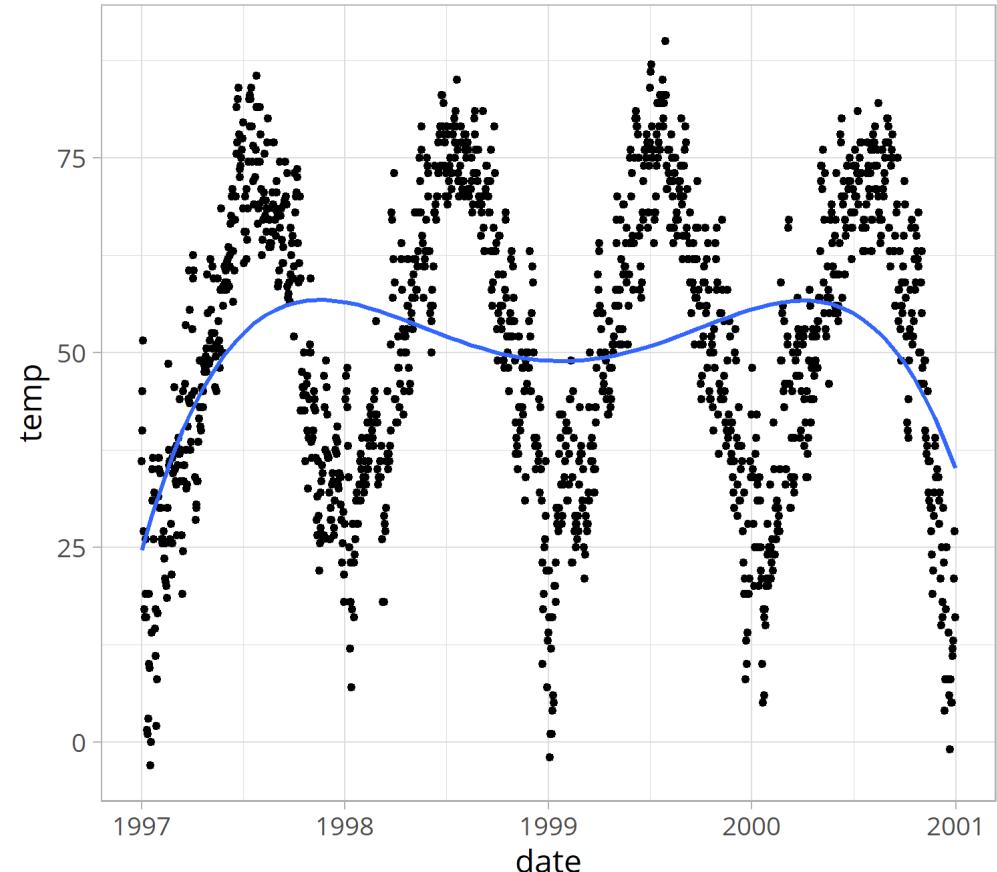
By default this adds a LOESS (locally weighted scatter plot smoothing, **method = "loess"**) or a GAM (generalized additive model, **method = "gam"**) depending on the number of data points (GAM in case of ≥ 1000 observations).



stat_smooth()

You can specify the fitting method and the formula:

```
ggplot(chic, aes(date, temp)) +  
  geom_point() +  
  stat_smooth(  
    method = "lm",  
    formula = y ~ x + I(x^2) +  
              I(x^3) + I(x^4),  
    se = F  
)
```

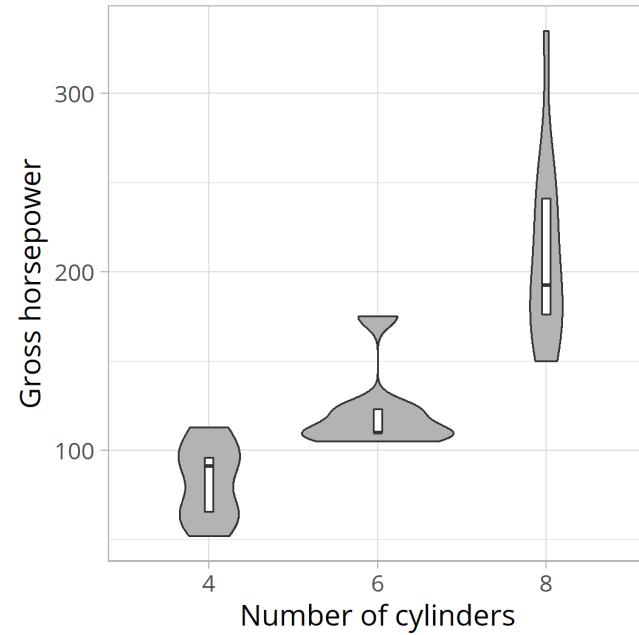
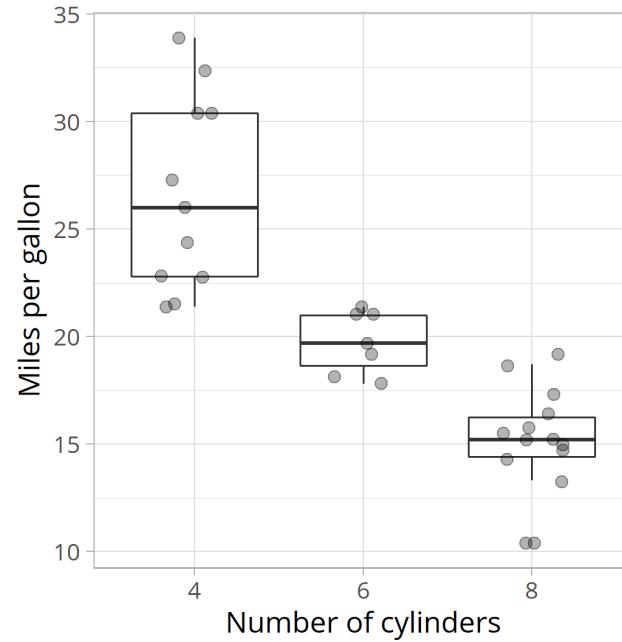


Other methods such as `method = "lm"` (without an explicit formula) for simple linear regressions and `method = "glm"` for generalized linear models are available as well.

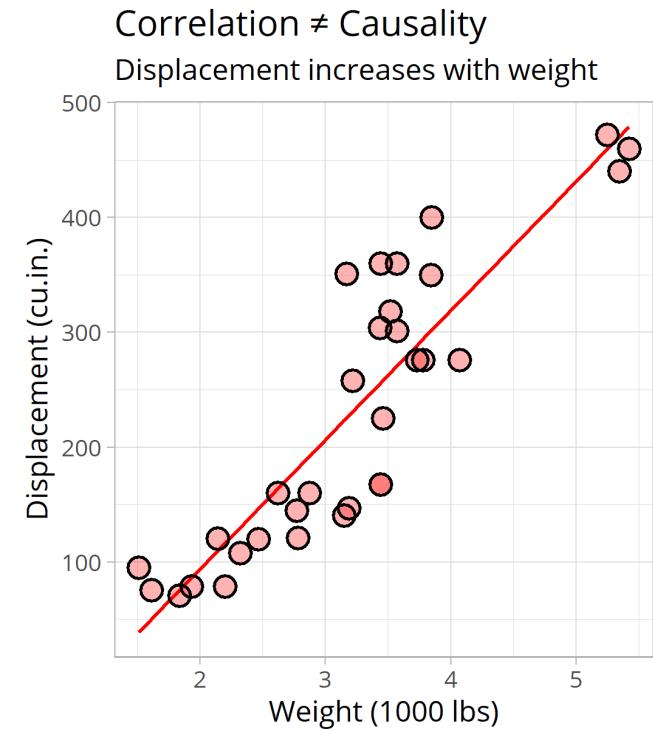
Your Turn!

A famous example datat set is `mtcars` which is pre-loaded in R.

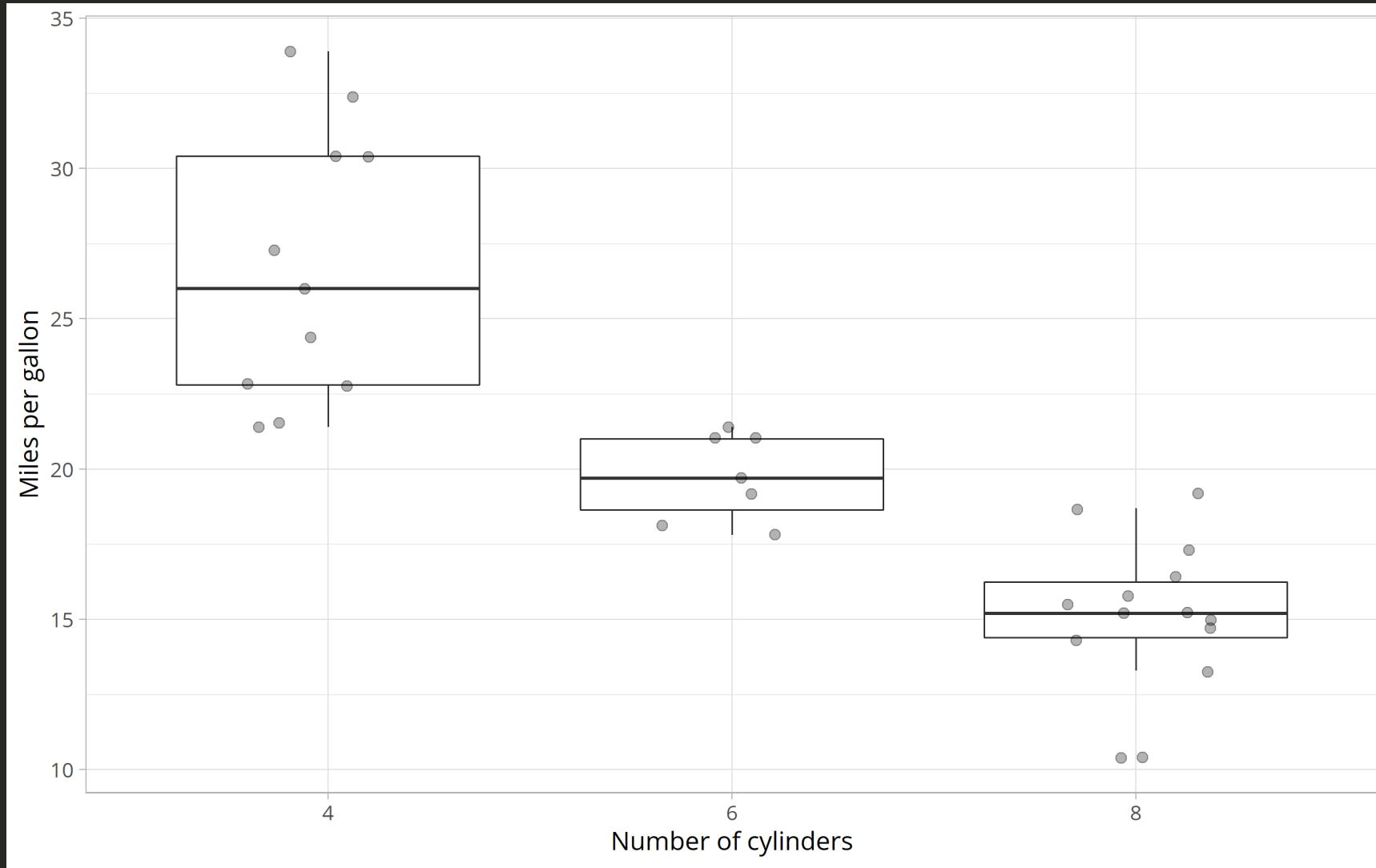
- Have a look at the data set and create the following three visualizations:



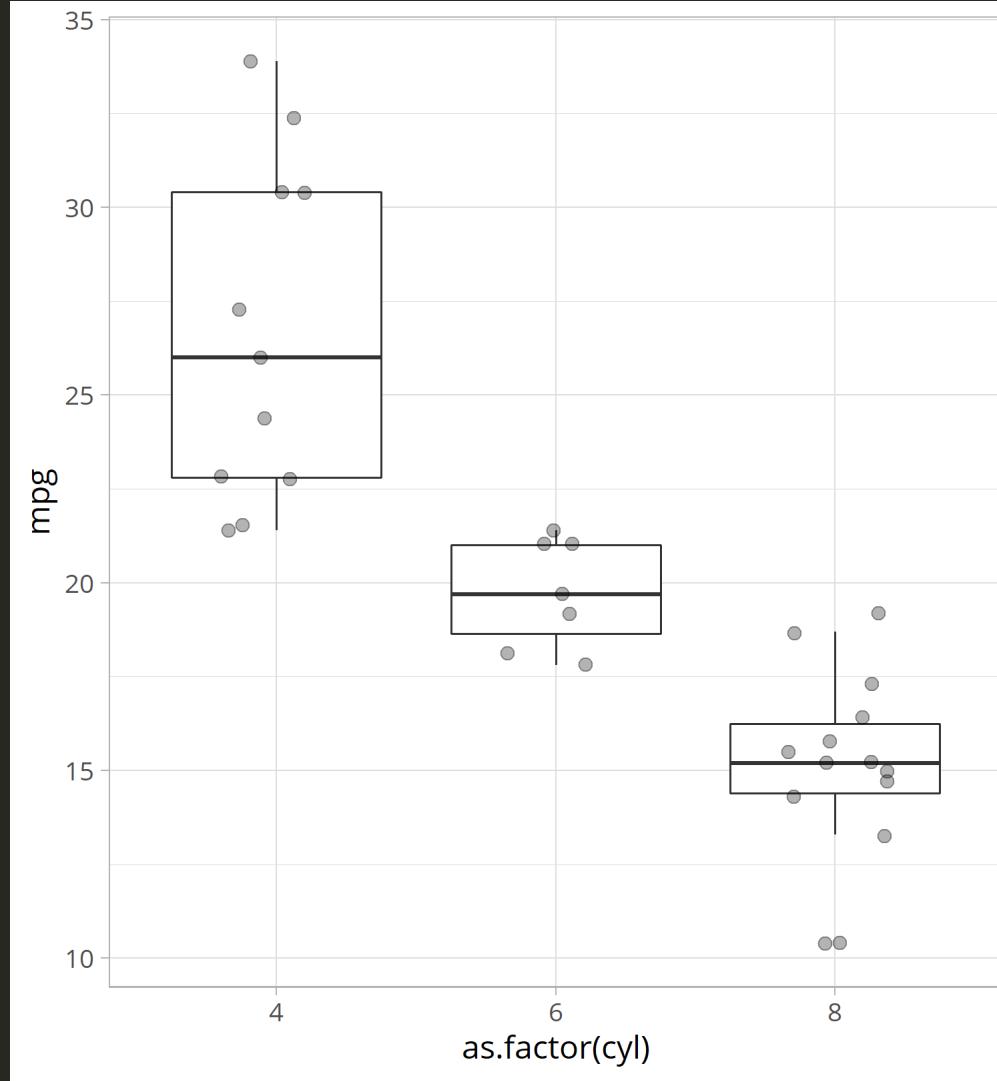
Data: 1974 "Motor Trend" US magazine



Your Turn: Create this Box Plot!

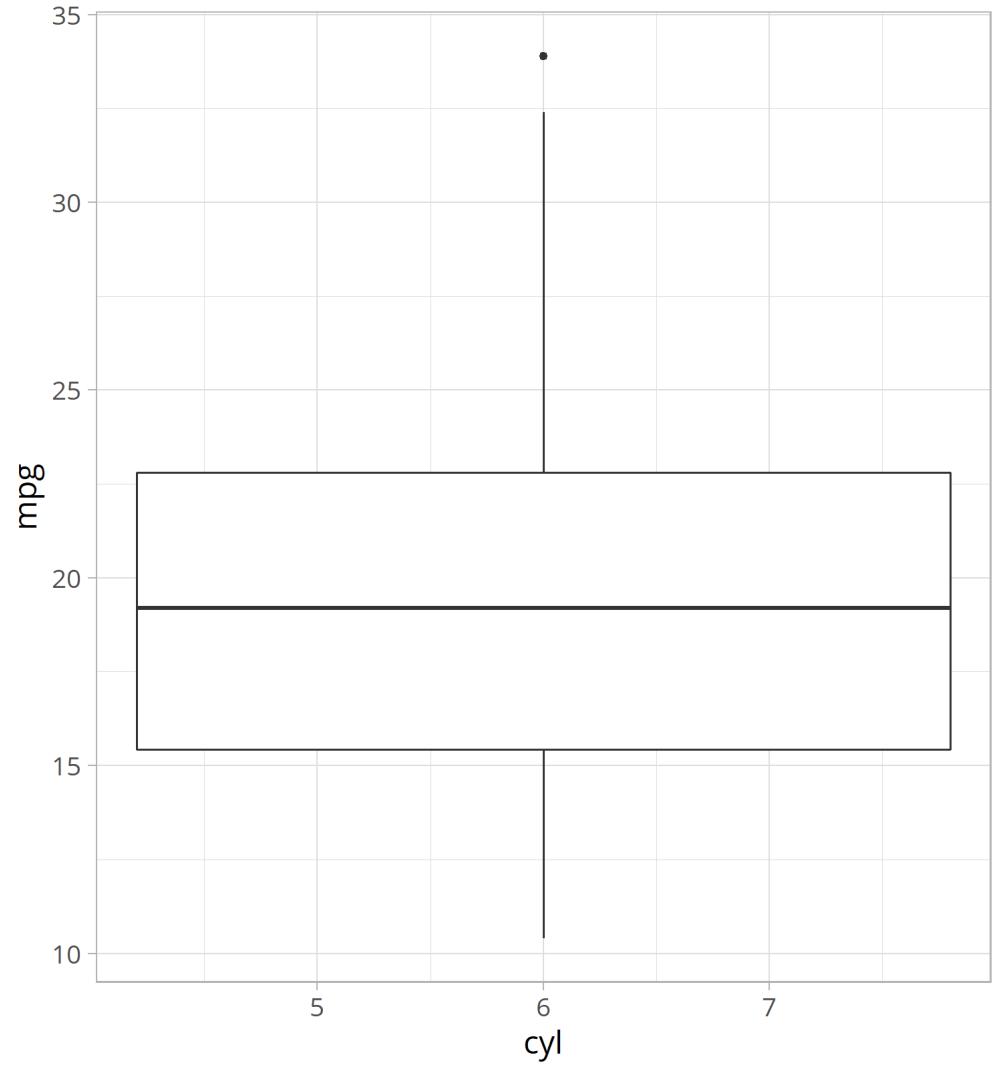


Your Turn: Create this Box Plot! (Less Mean)



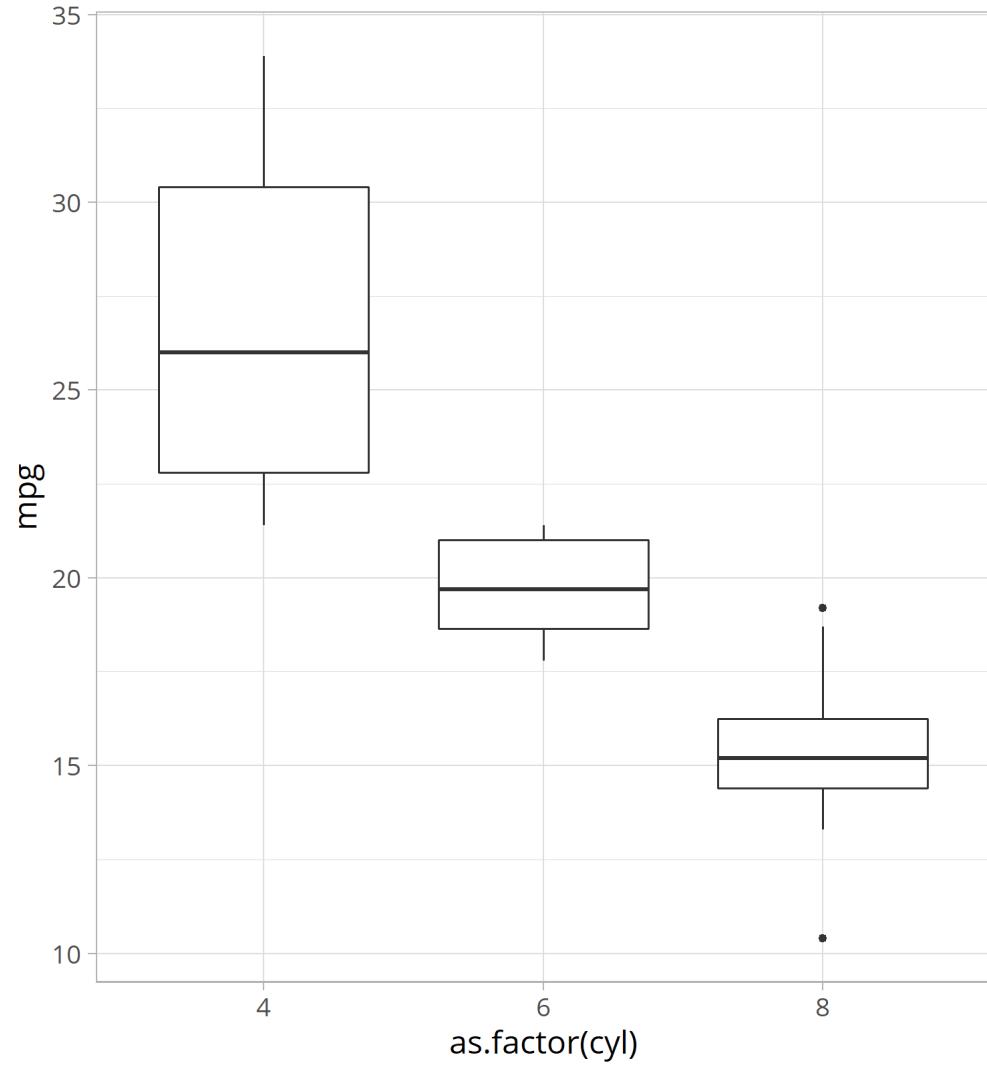
Your Turn: Box Plots with Raw Data

```
ggplot(  
  mtcars,  
  aes(cyl, mpg)  
) +  
  geom_boxplot()
```



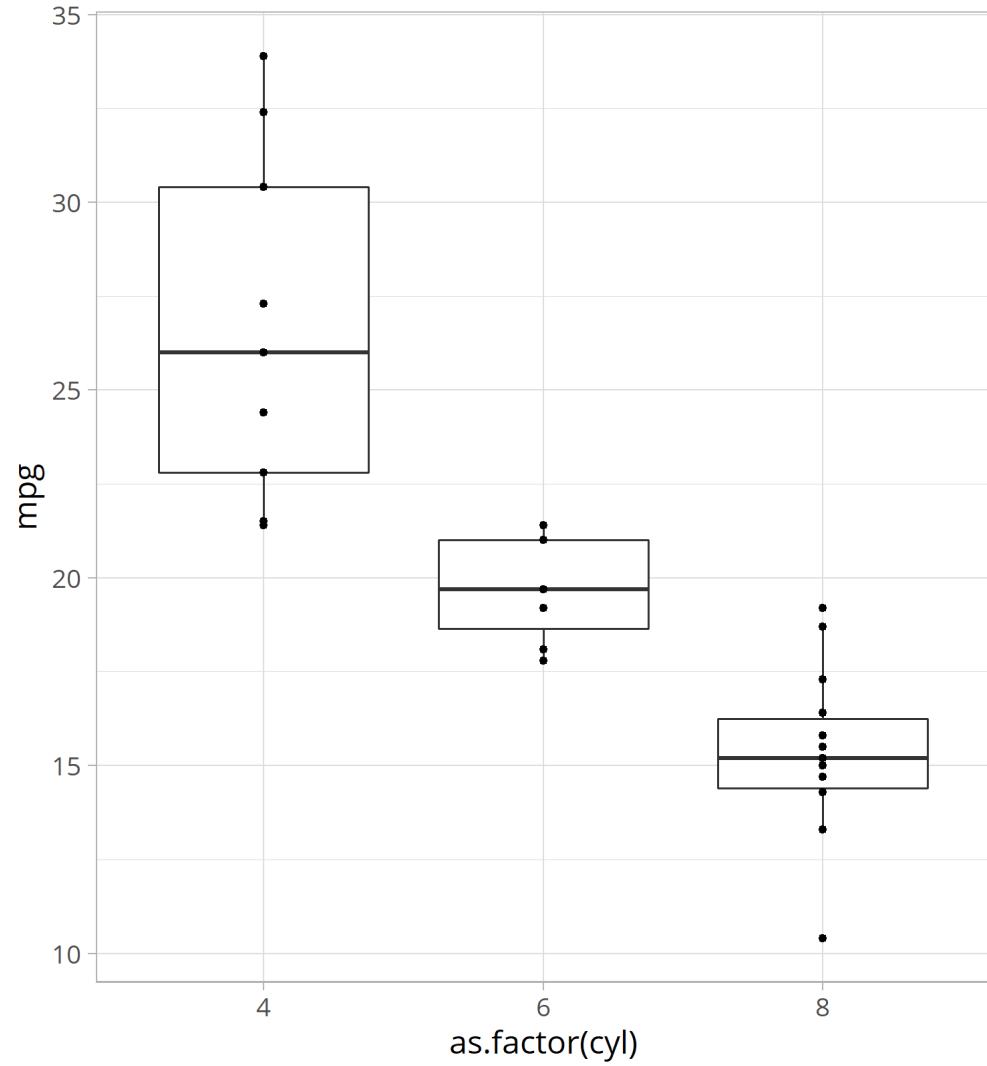
Your Turn: Box Plots with Raw Data

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), mpg)  
) +  
  geom_boxplot()
```



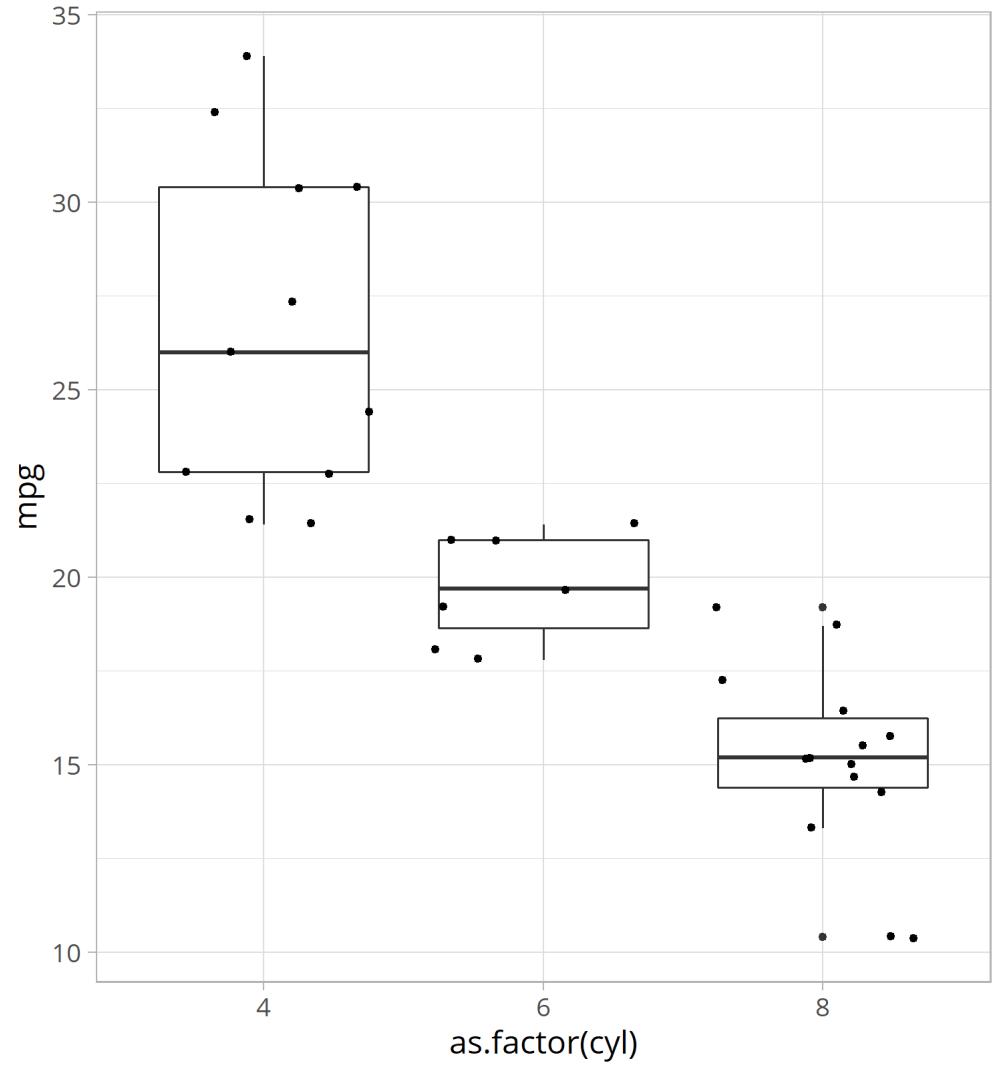
Your Turn: Box Plots with Raw Data

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), mpg)  
 ) +  
 geom_boxplot() +  
 geom_point()
```



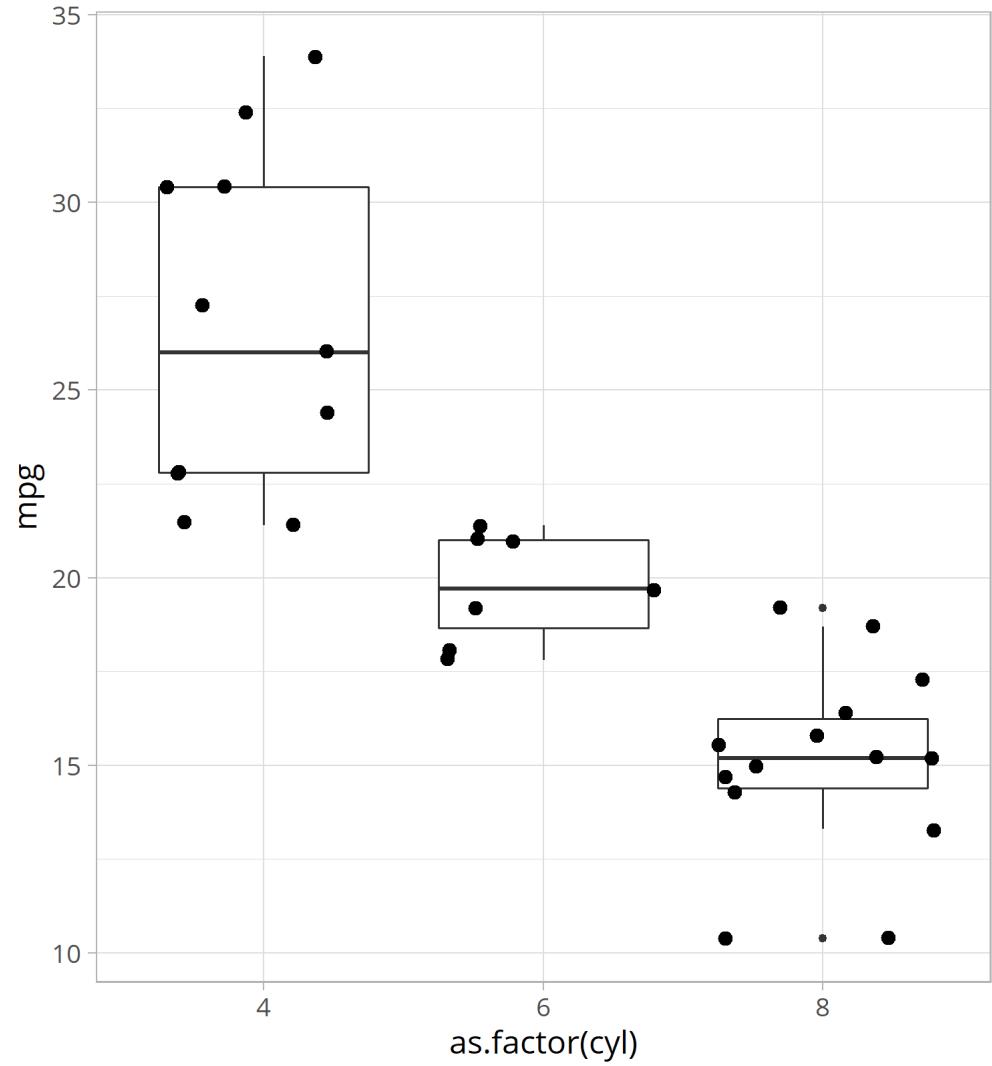
Your Turn: Box Plots with Raw Data

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), mpg)  
 ) +  
 geom_boxplot() +  
 geom_jitter()
```



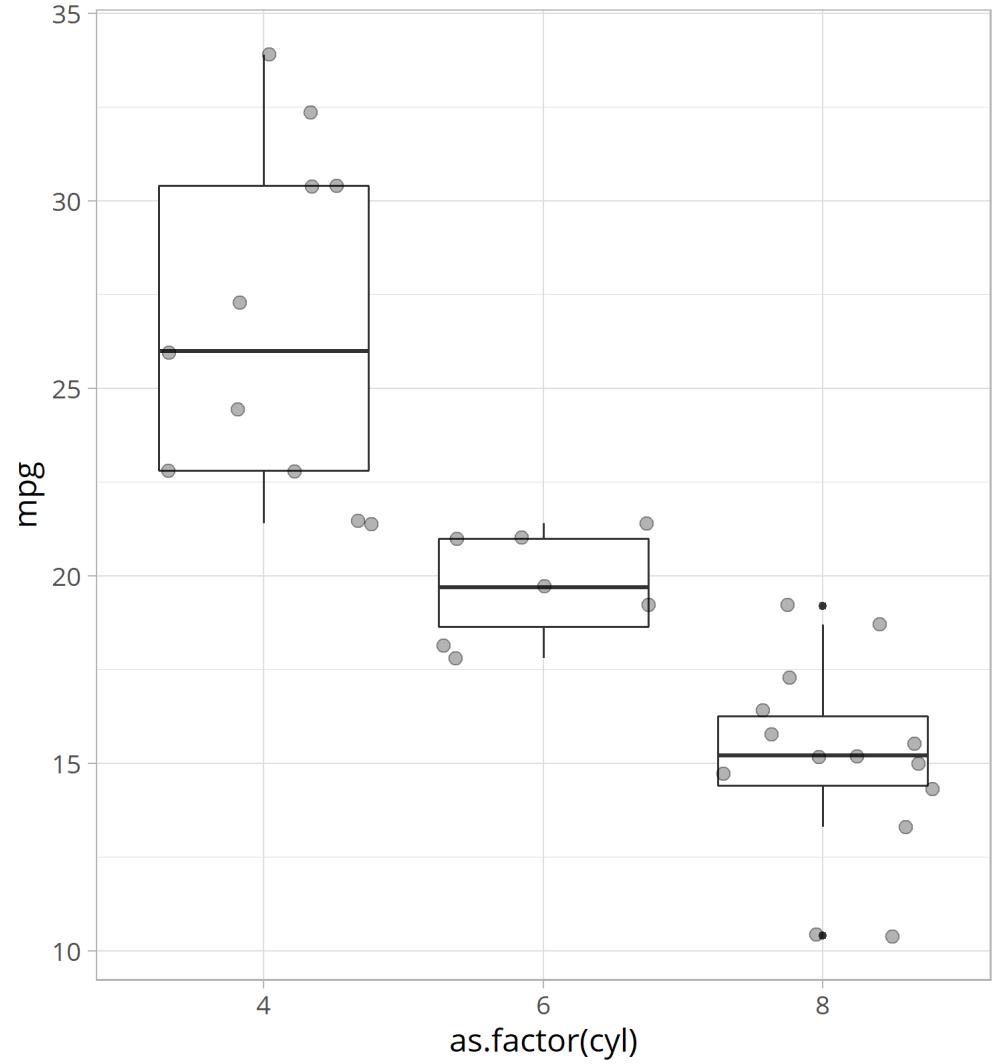
Your Turn: Box Plots with Raw Data

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), mpg)  
) +  
  geom_boxplot() +  
  geom_jitter(  
    size = 3  
)
```



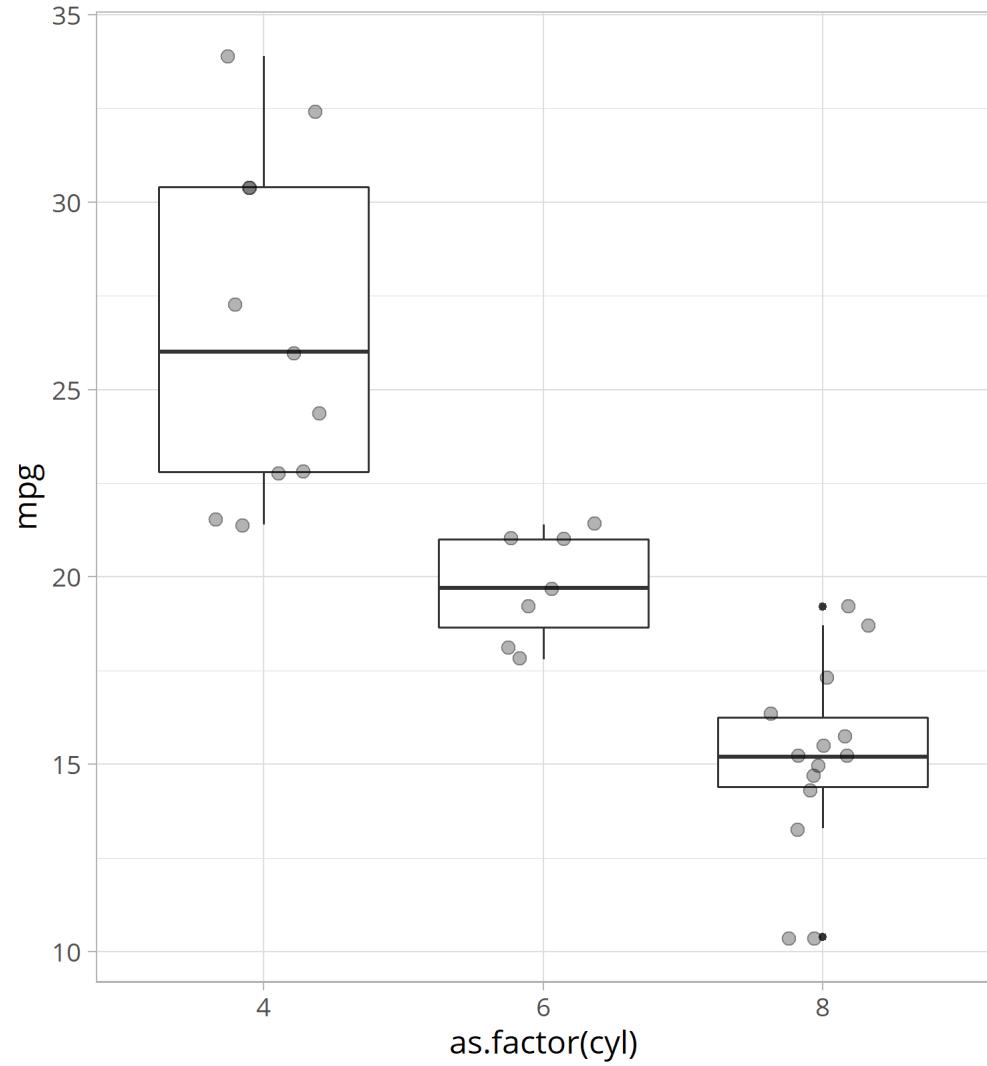
Your Turn: Box Plots with Raw Data

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), mpg)  
) +  
  geom_boxplot() +  
  geom_jitter(  
    size = 3,  
    alpha = .3  
)
```



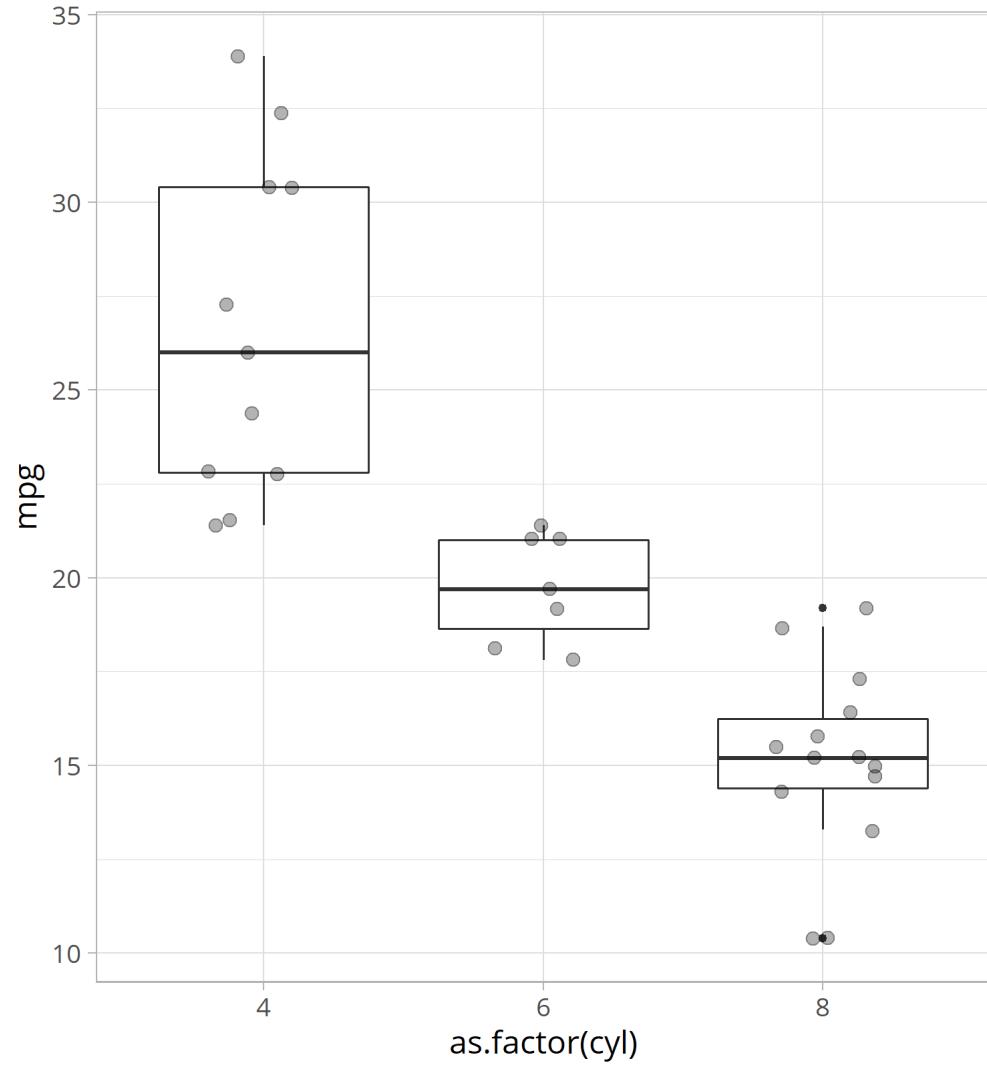
Your Turn: Box Plots with Raw Data

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), mpg)  
) +  
  geom_boxplot() +  
  geom_jitter(  
    size = 3,  
    alpha = .3,  
    width = .2  
)
```



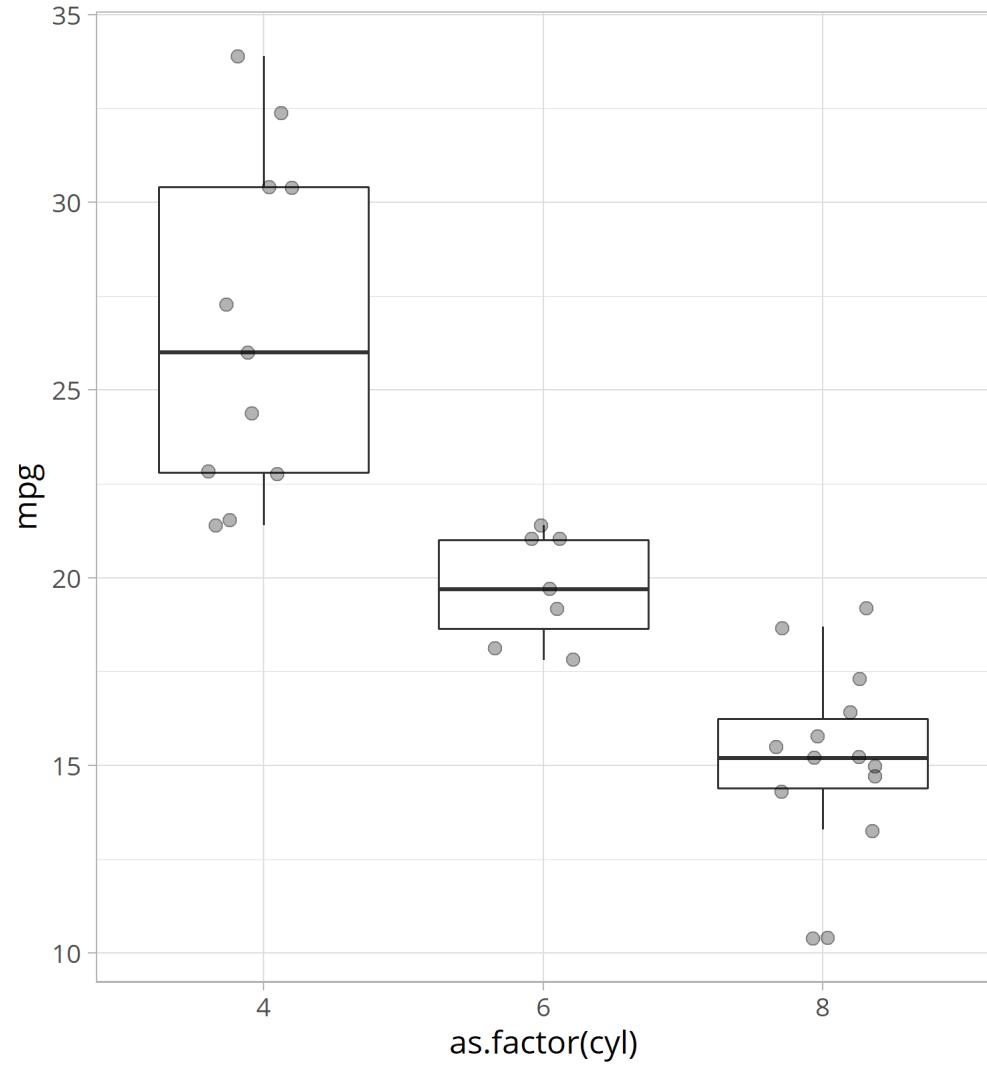
Your Turn: Box Plots with Raw Data

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), mpg)  
) +  
  geom_boxplot() +  
  geom_jitter(  
    size = 3,  
    alpha = .3,  
    position = position_jitter(  
      width = .2,  
      seed = 2020  
    )  
)
```



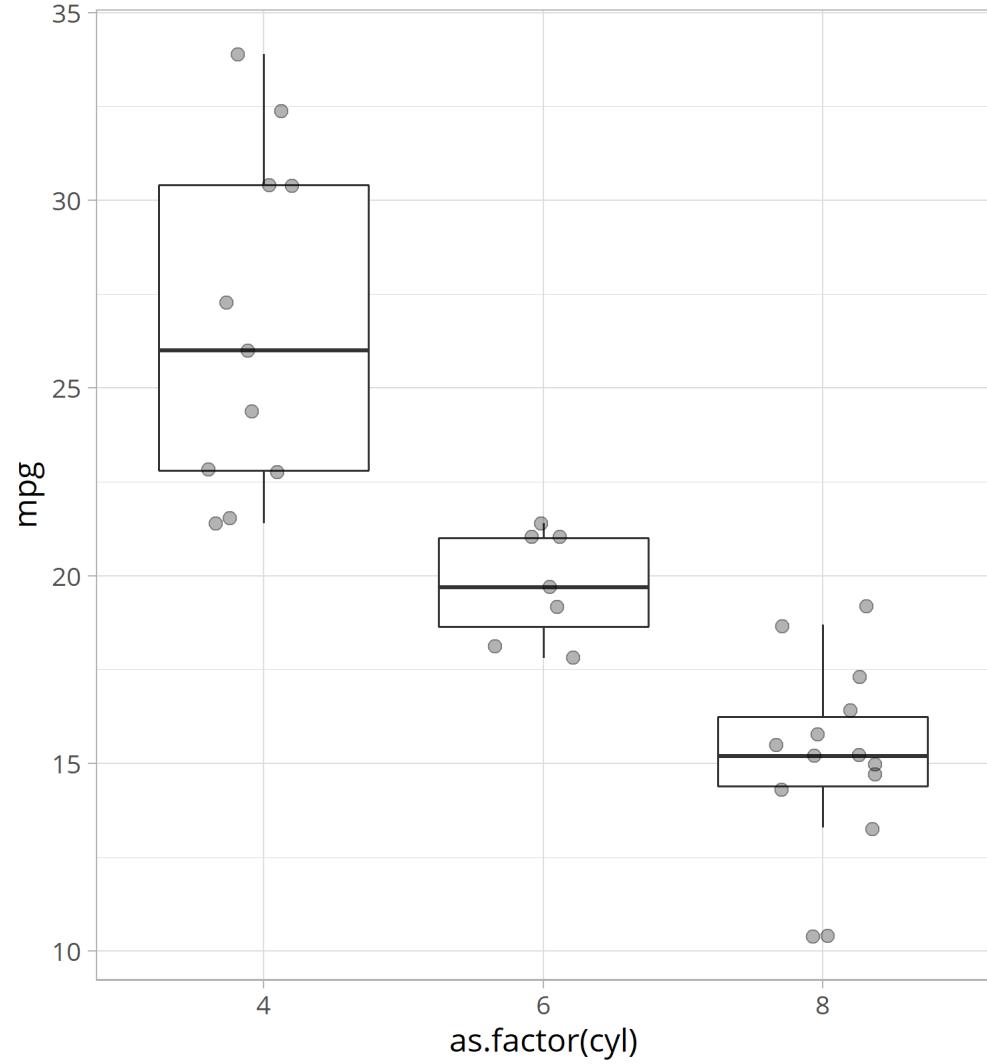
Your Turn: Box Plots with Raw Data

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), mpg)  
) +  
  geom_boxplot(  
    outlier.alpha = 0  
) +  
  geom_jitter(  
    size = 3,  
    alpha = .3,  
    position = position_jitter(  
      width = .2,  
      seed = 2020  
)  
)
```

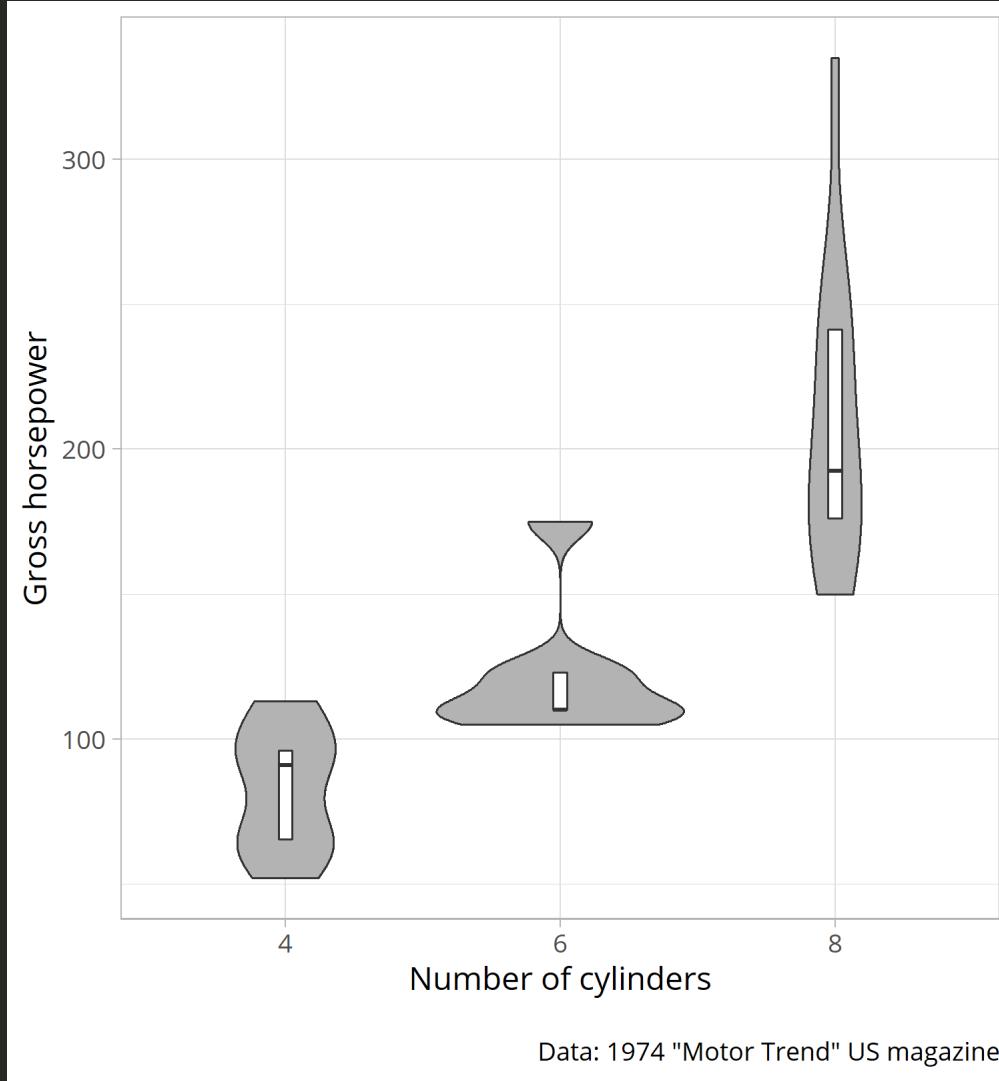


Your Turn: Box Plots with Raw Data

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), mpg  
) +  
  geom_boxplot(  
    #outlier.alpha = 0,  
    outlier.shape = NA  
) +  
  geom_jitter(  
    size = 3,  
    alpha = .3,  
    position = position_jitter(  
      width = .2,  
      seed = 2020  
)  
)
```

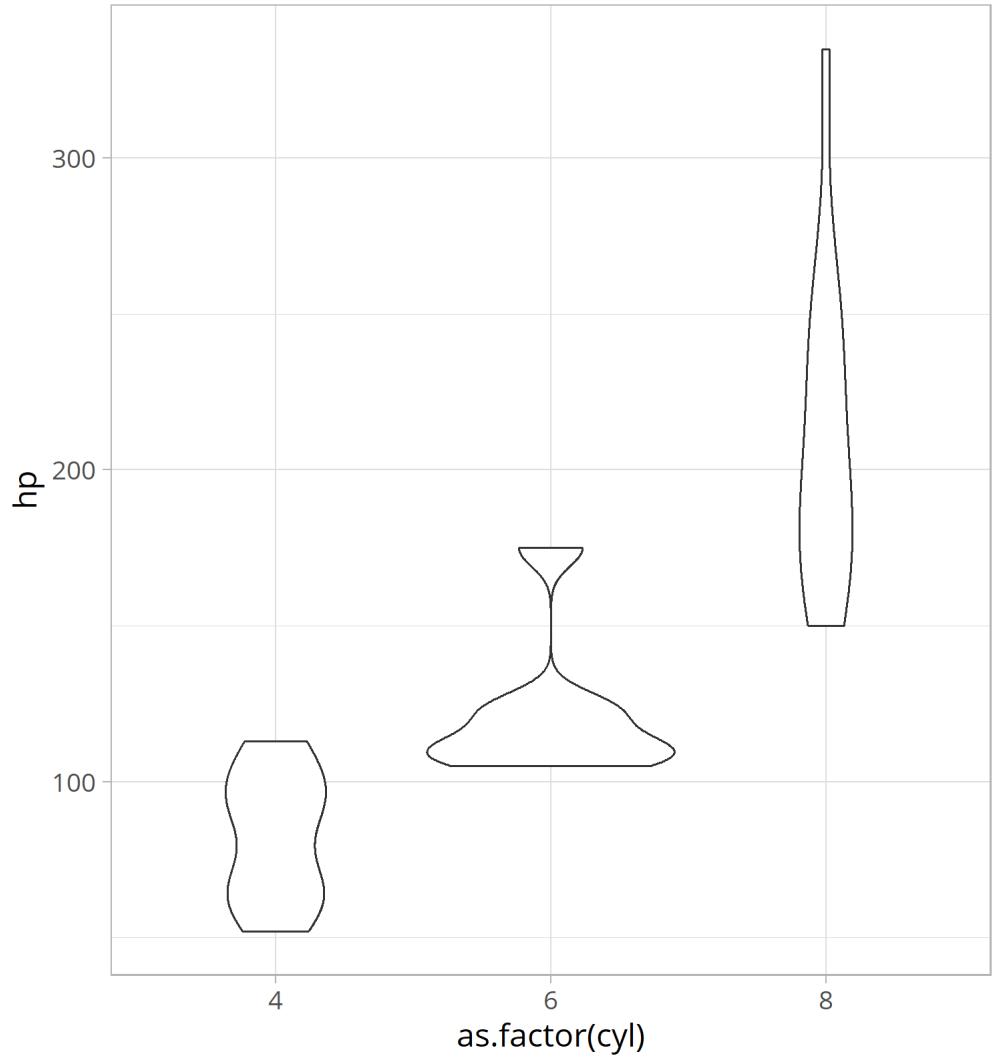


Your Turn: Create this Violin Plot!



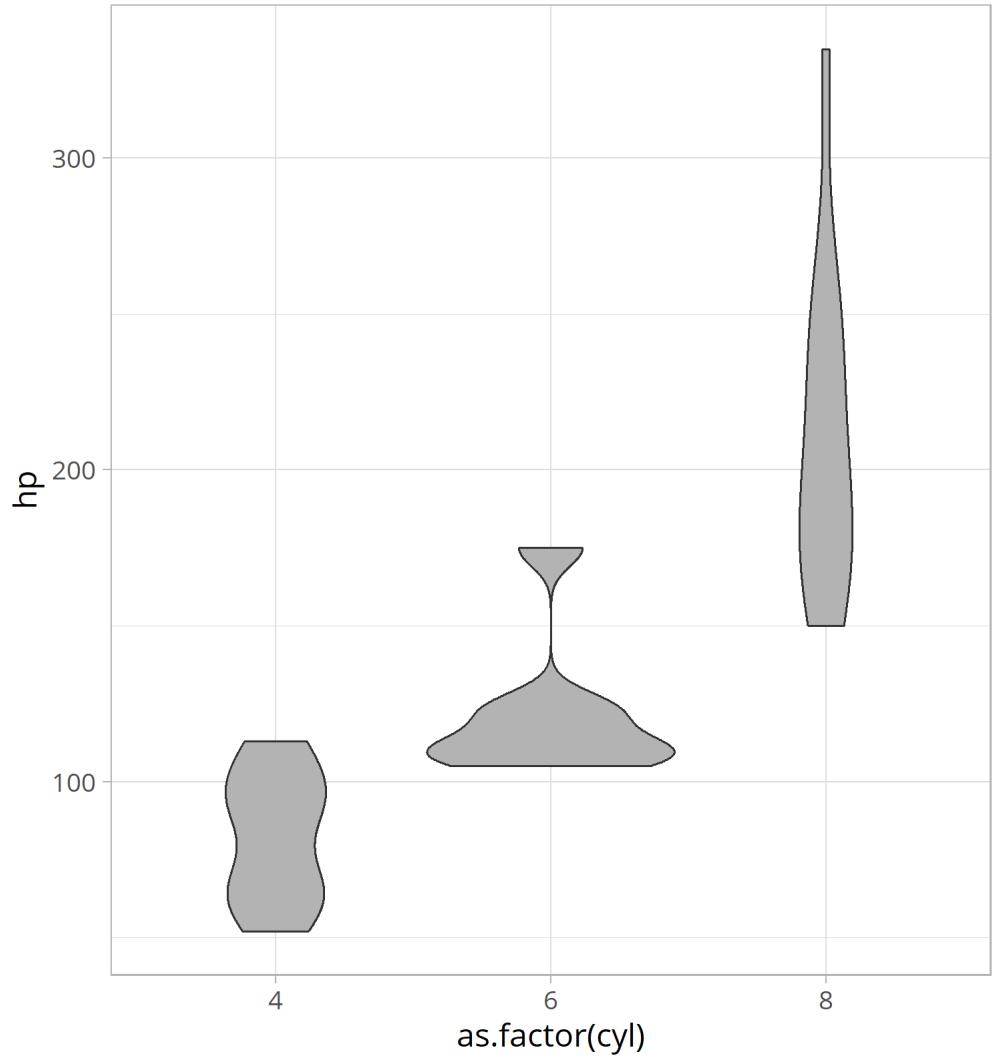
Your Turn: Violin Plot

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), hp)  
) +  
  geom_violin()
```



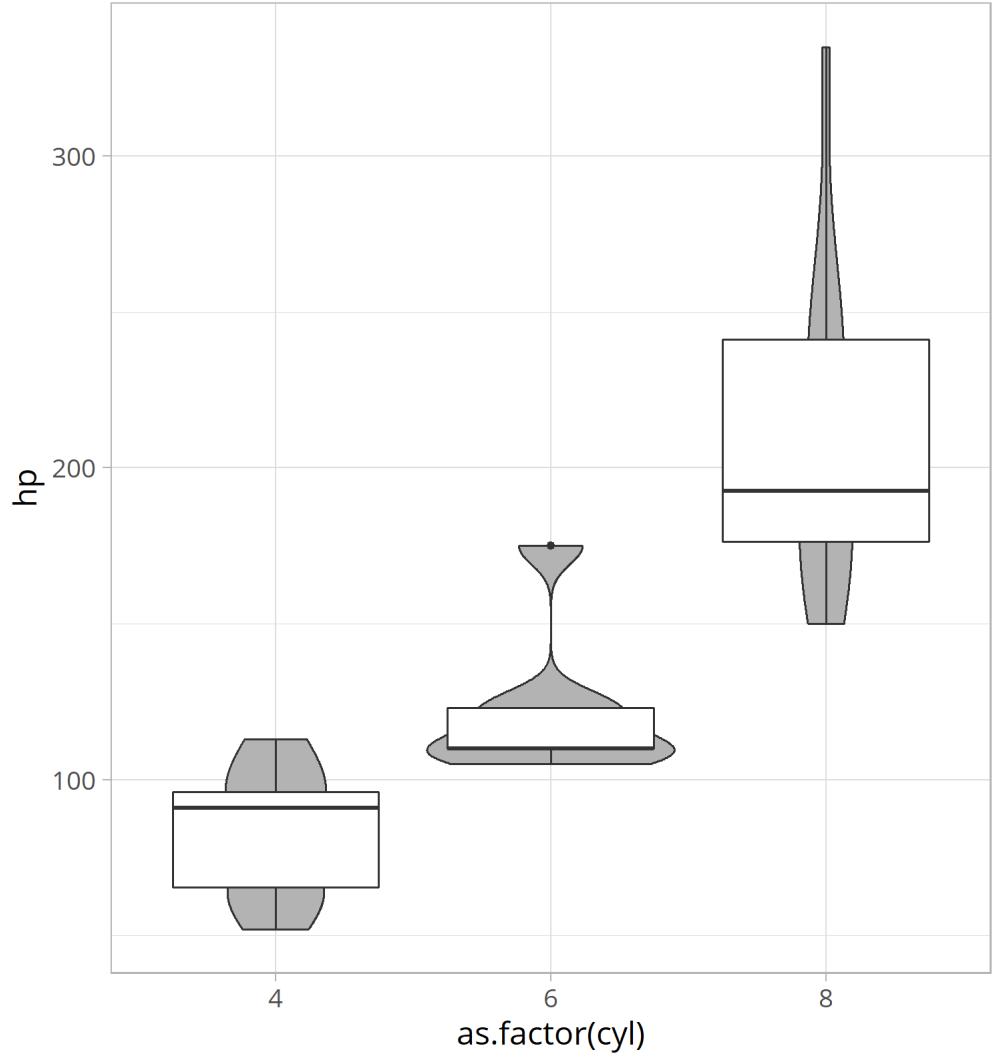
Your Turn: Violin Plot

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), hp)  
 ) +  
 geom_violin(  
   fill = "grey70"  
 )
```



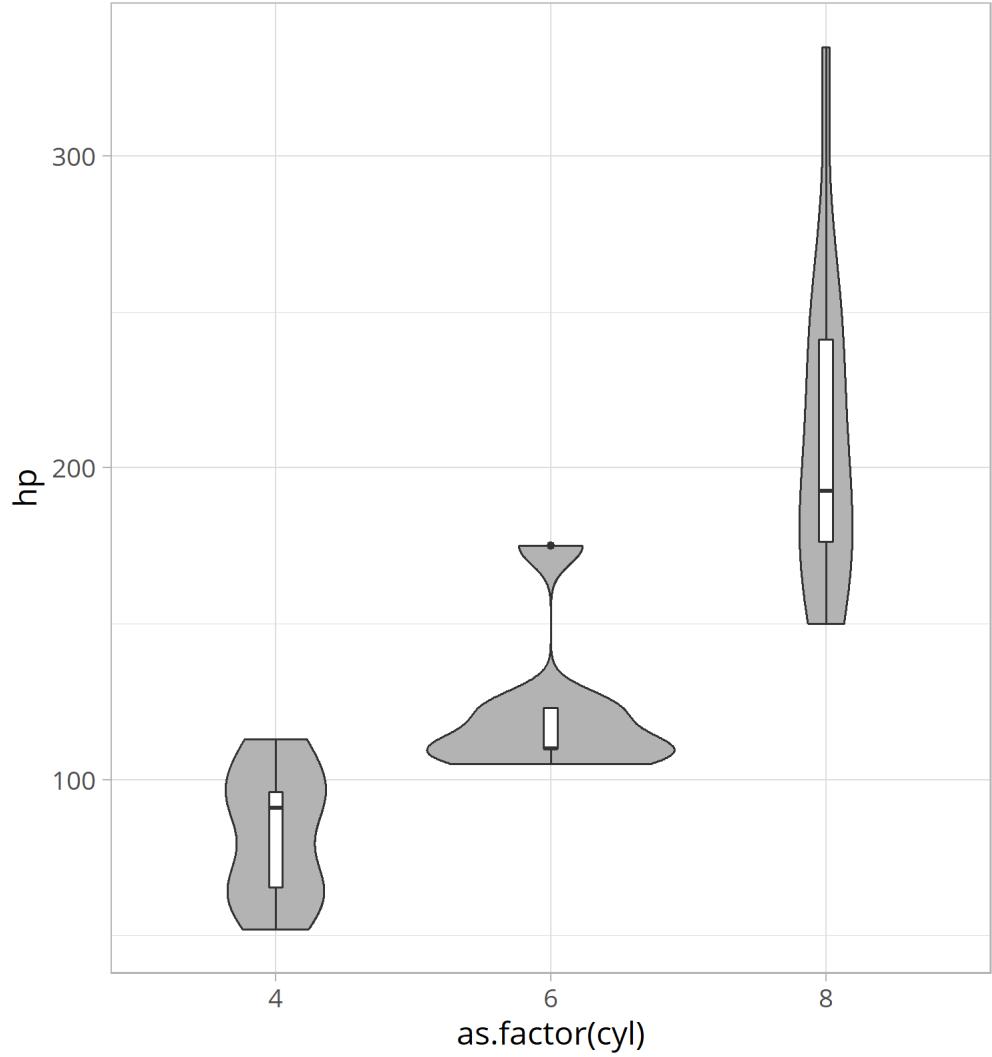
Your Turn: Violin Plot

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), hp)  
 ) +  
 geom_violin(  
   fill = "grey70"  
 ) +  
 geom_boxplot()
```



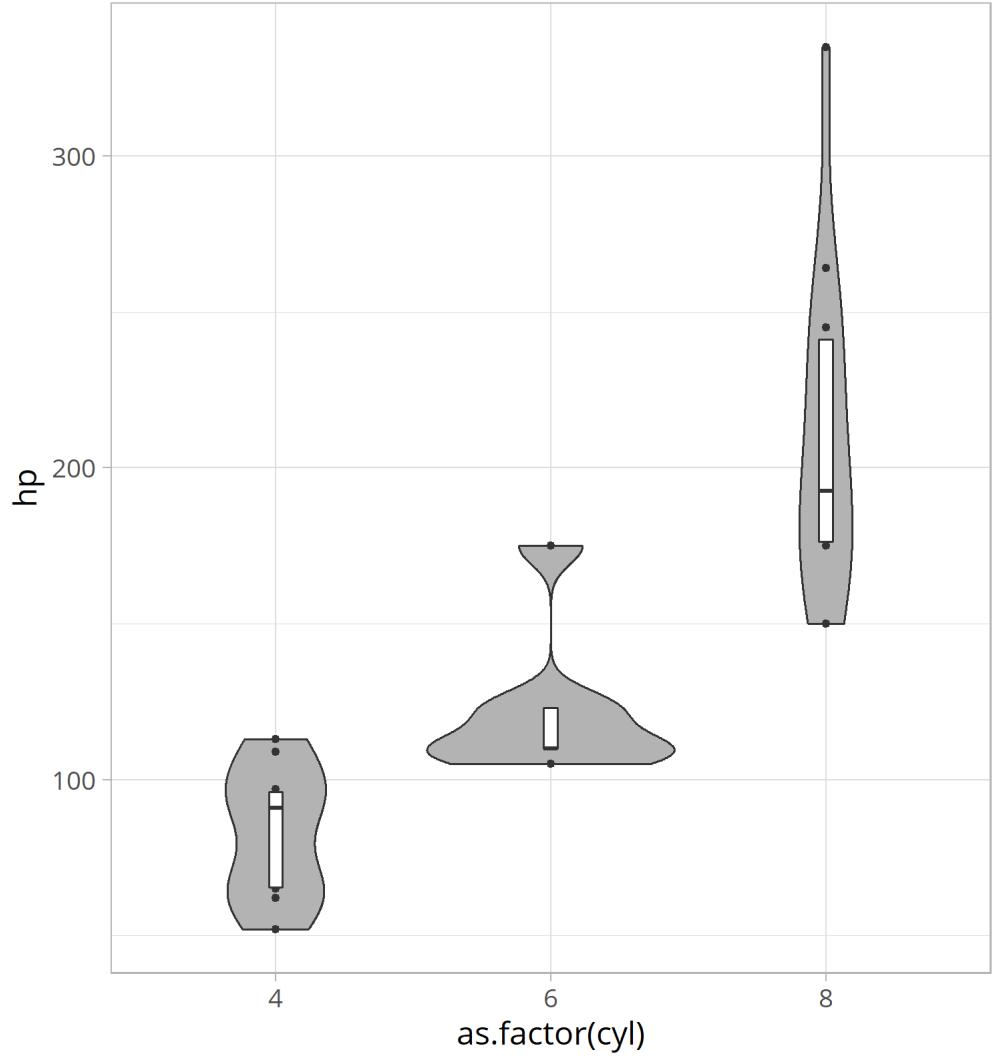
Your Turn: Violin Plot

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), hp)  
 ) +  
 geom_violin(  
   fill = "grey70"  
 ) +  
 geom_boxplot(width = .05)
```



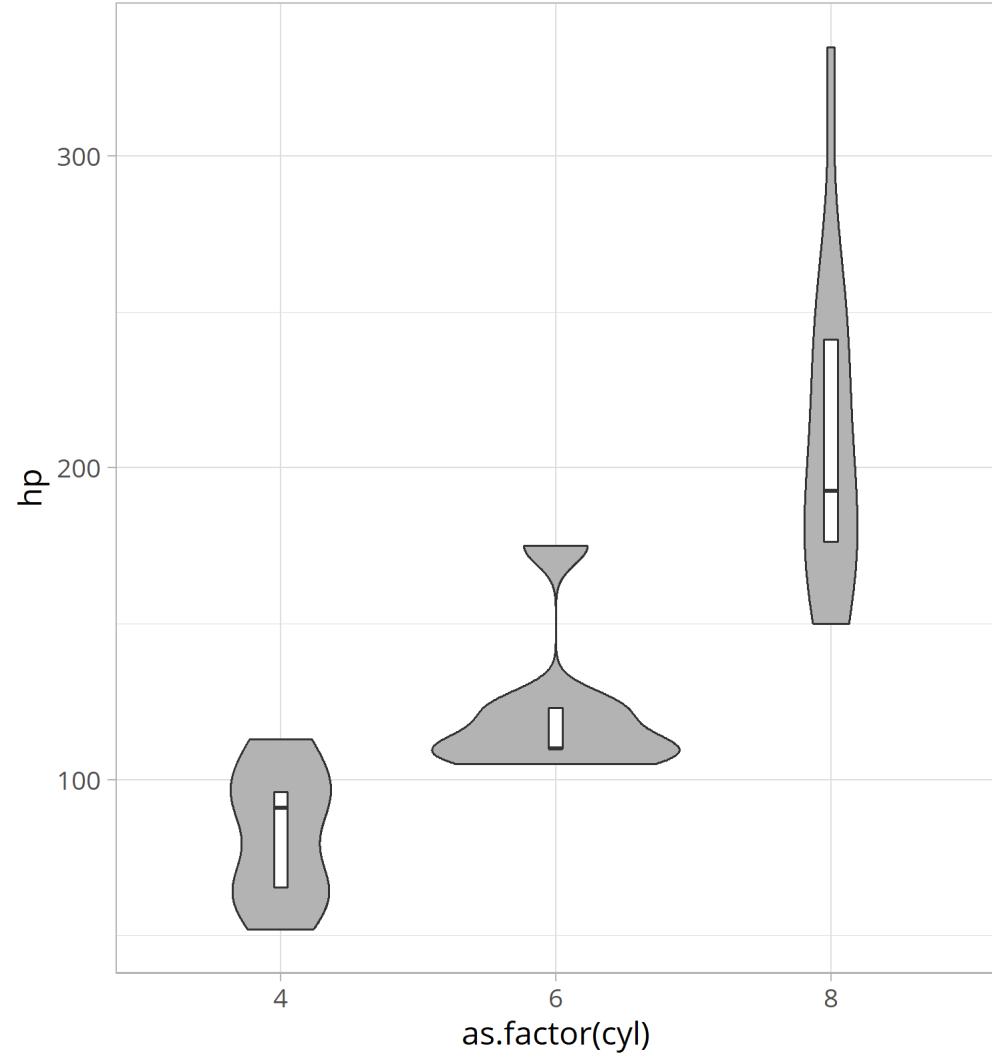
Your Turn: Violin Plot

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), hp)  
 ) +  
 geom_violin(  
   fill = "grey70"  
 ) +  
 geom_boxplot(  
   width = .05,  
   coef = 0  
 )
```



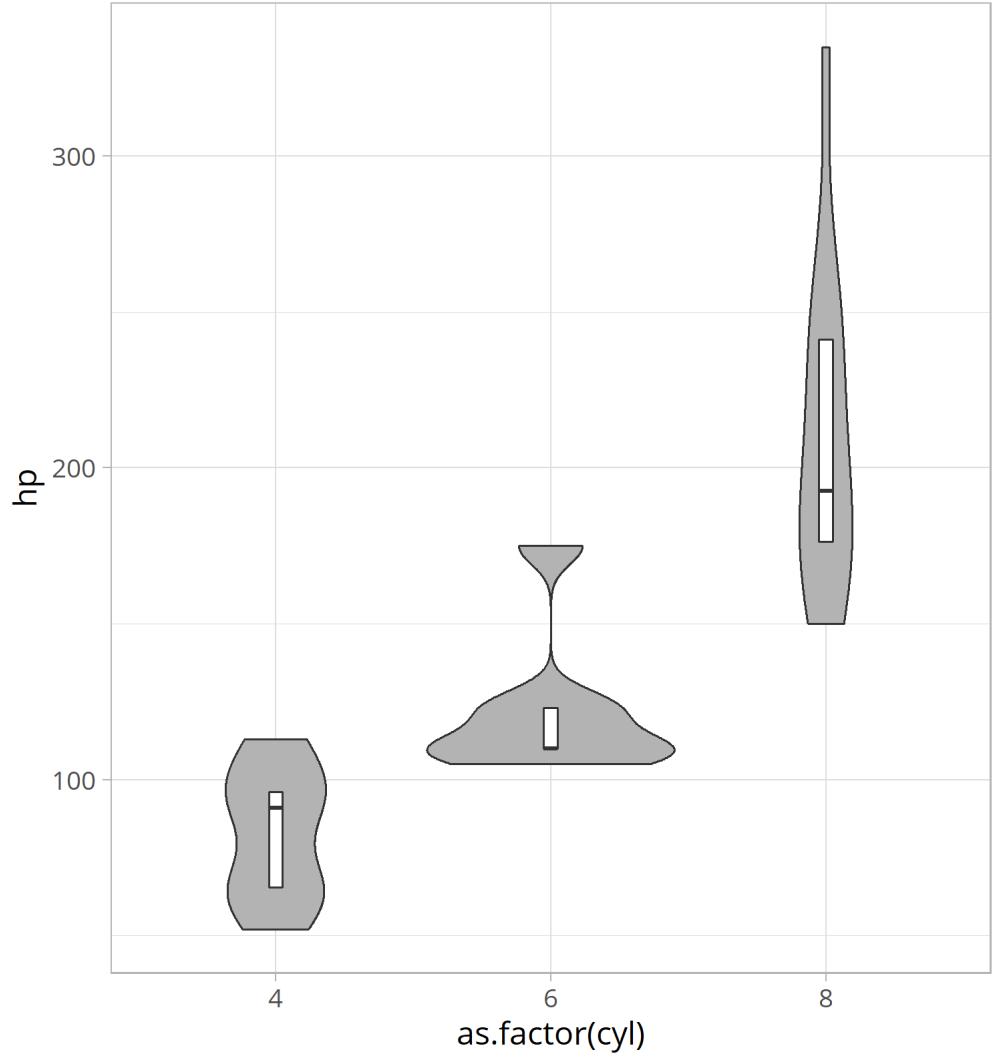
Your Turn: Violin Plot

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), hp)  
 ) +  
 geom_violin(  
   fill = "grey70"  
 ) +  
 geom_boxplot(  
   width = .05,  
   coef = 0,  
   outlier.shape = NA  
)
```



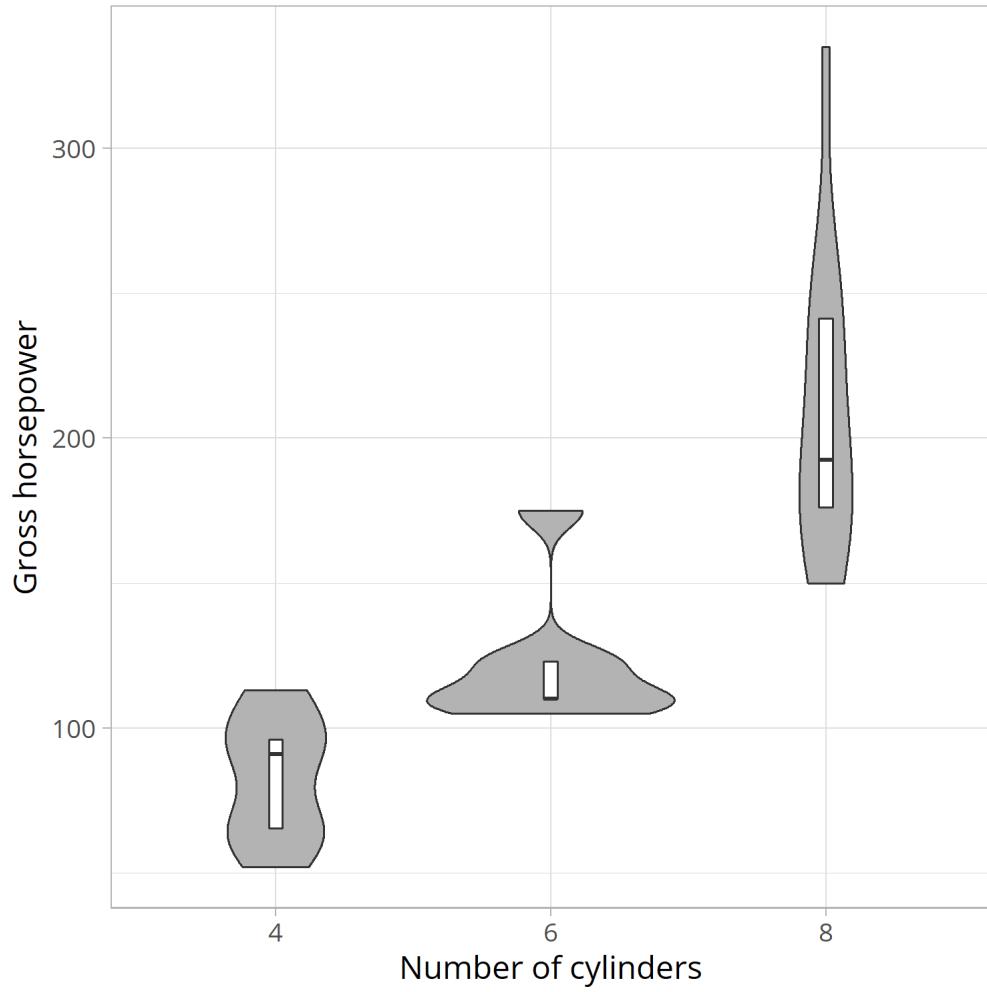
Your Turn: Violin Plot

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), hp)  
 ) +  
 geom_violin(  
   fill = "grey70"  
 ) +  
 geom_boxplot(  
   width = .05,  
   coef = 0,  
   #outlier.shape = NA  
   outlier.alpha = 0  
)
```



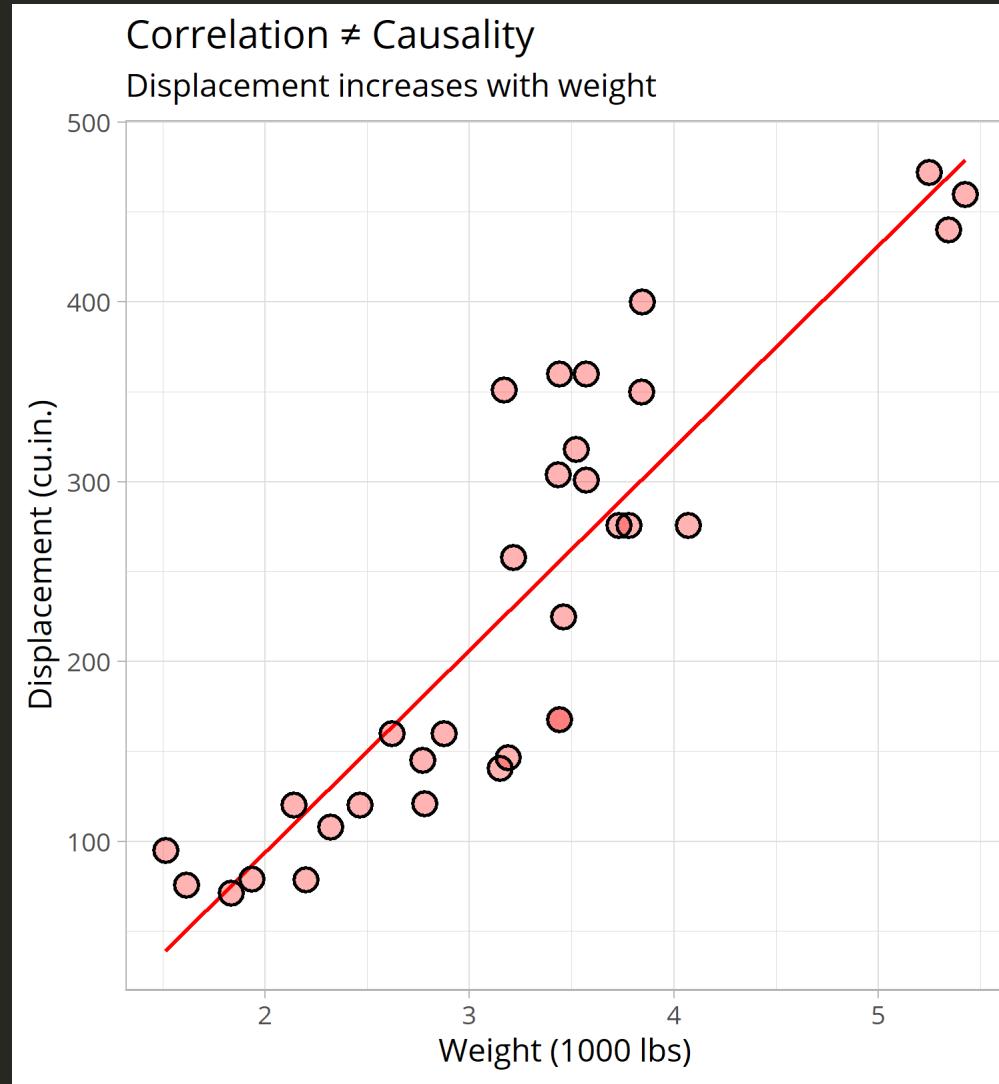
Your Turn: Violin Plot

```
ggplot(  
  mtcars,  
  aes(as.factor(cyl), hp)  
) +  
  geom_violin(  
    fill = "grey70"  
) +  
  geom_boxplot(  
    width = .05,  
    coef = 0,  
    #outlier.shape = NA  
    outlier.alpha = 0  
) +  
  labs(  
    x = "Number of cylinders",  
    y = "Gross horsepower",  
    caption = '\nData: 1974 "Motor Trend"  
)
```



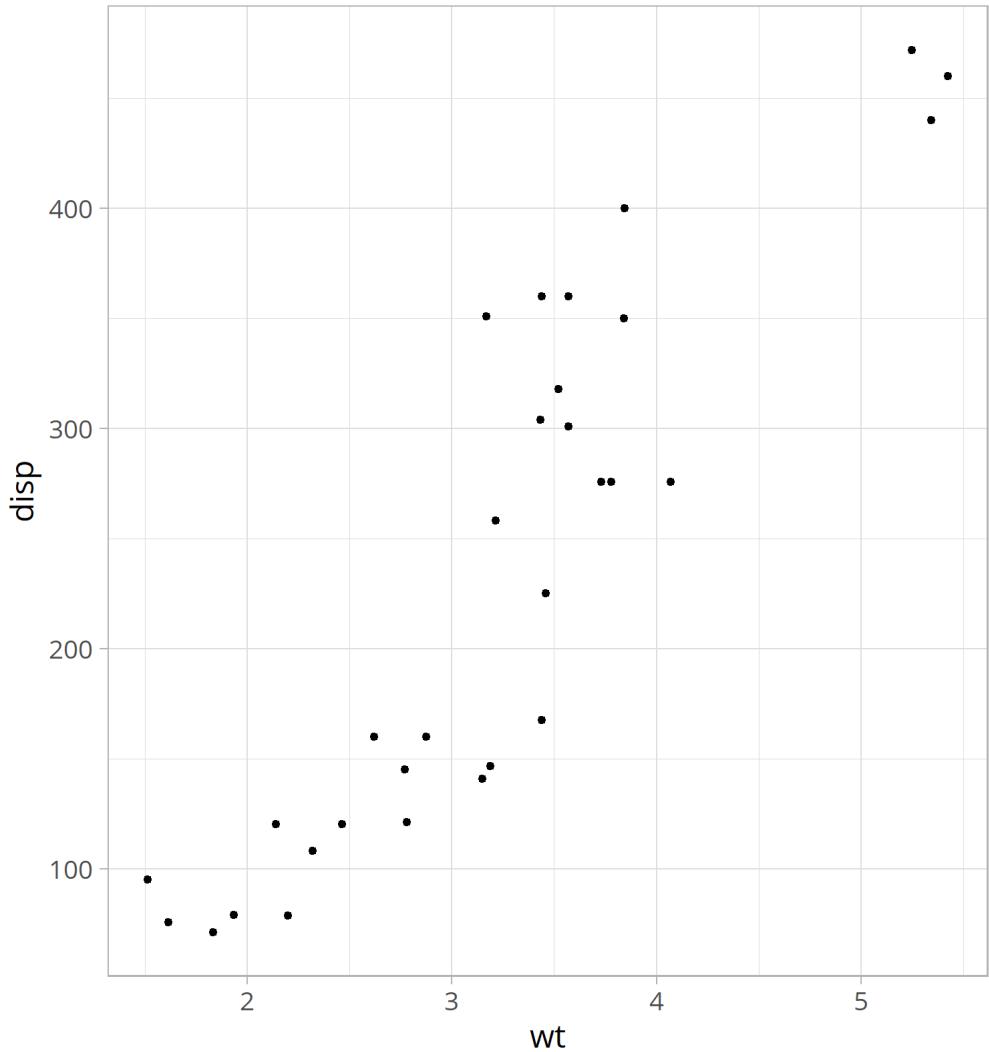
Data: 1974 "Motor Trend" US magazine

Your Turn: Create this Scatter Plot!



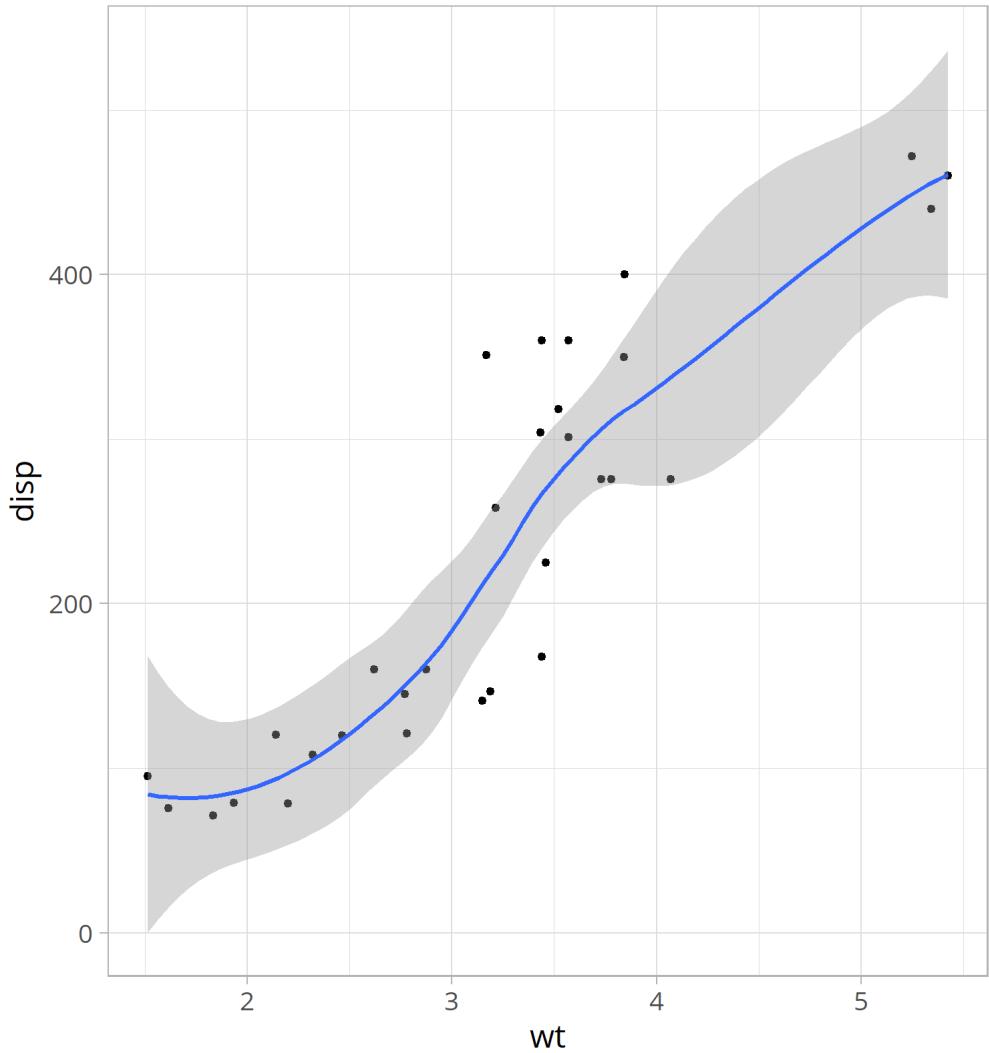
Your Turn: Scatter Plot with Linear Fitting

```
ggplot(mtcars, aes(wt, disp)) +  
  geom_point()
```



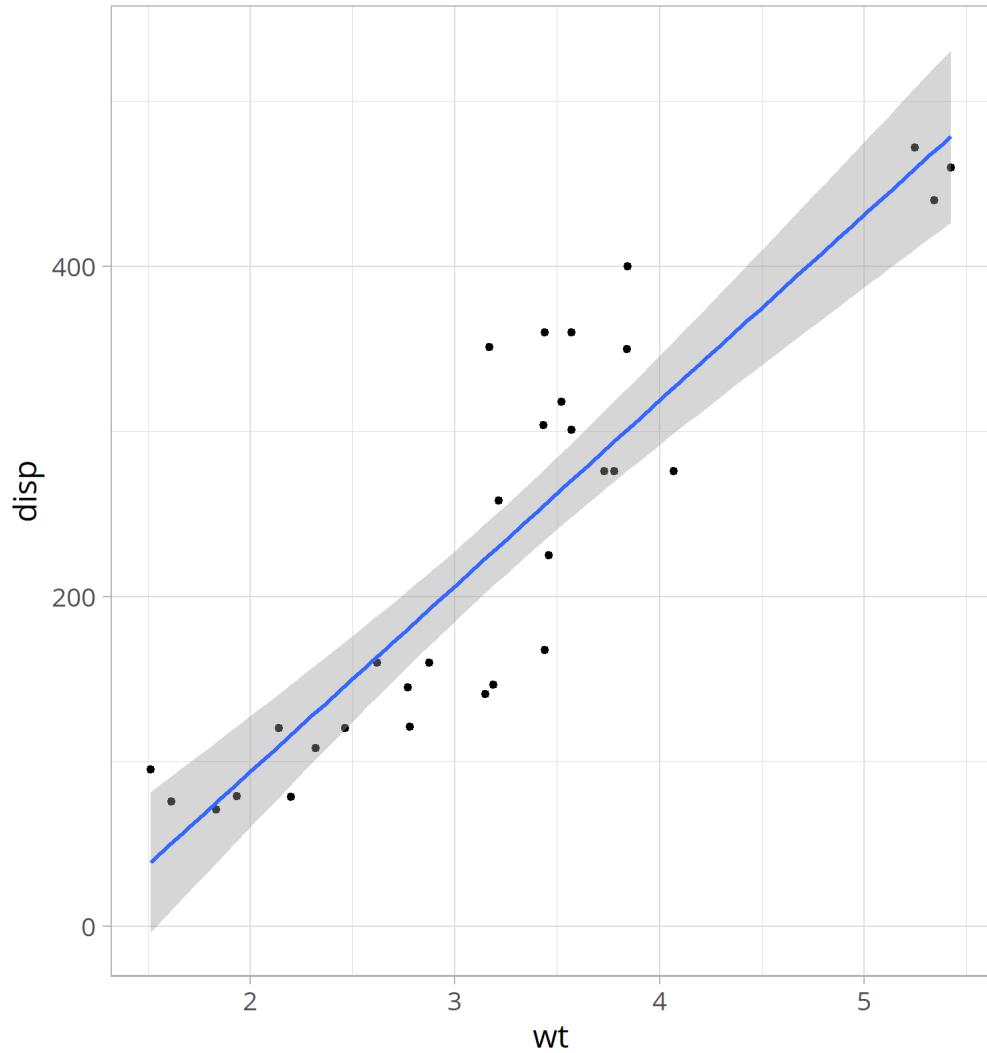
Your Turn: Scatter Plot with Linear Fitting

```
ggplot(mtcars, aes(wt, disp)) +  
  geom_point() +  
  stat_smooth()
```



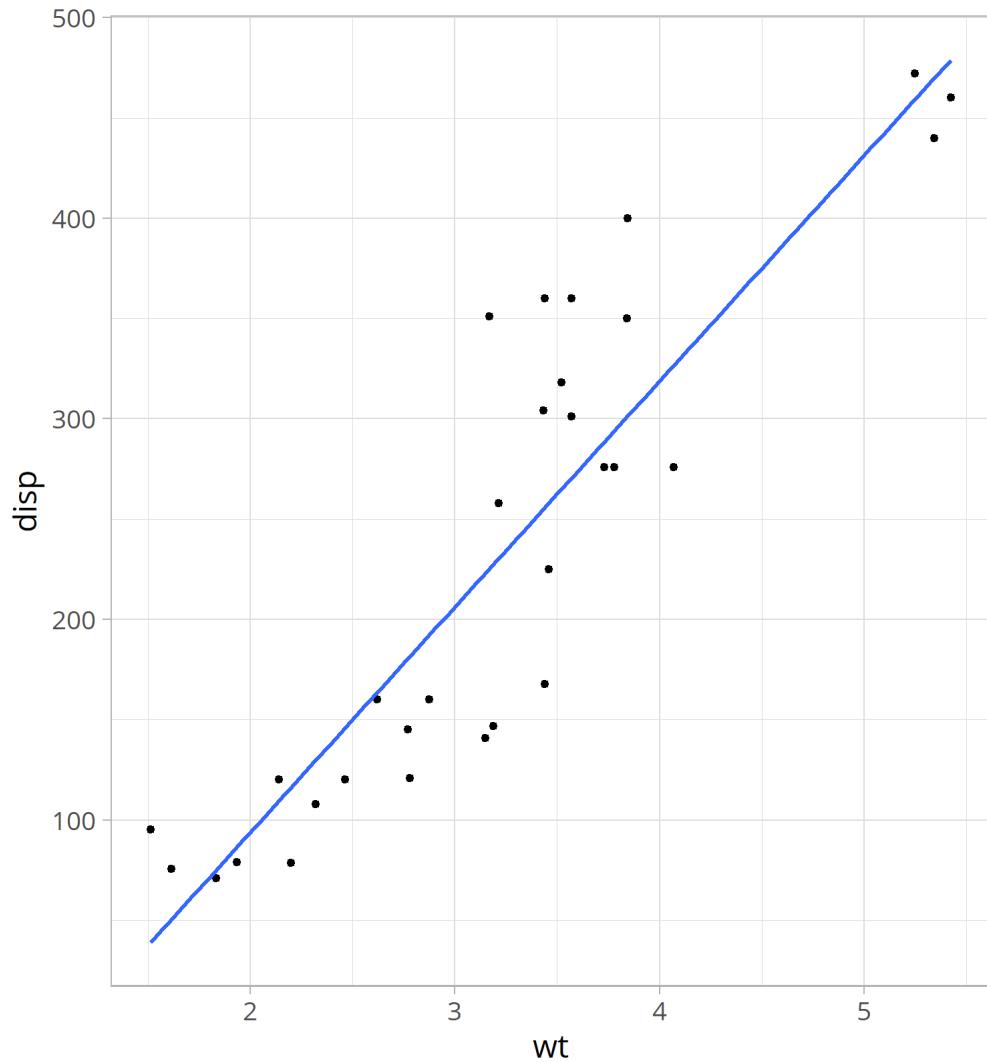
Your Turn: Scatter Plot with Linear Fitting

```
ggplot(mtcars, aes(wt, disp)) +  
  geom_point() +  
  stat_smooth(  
    method = "lm"  
)
```



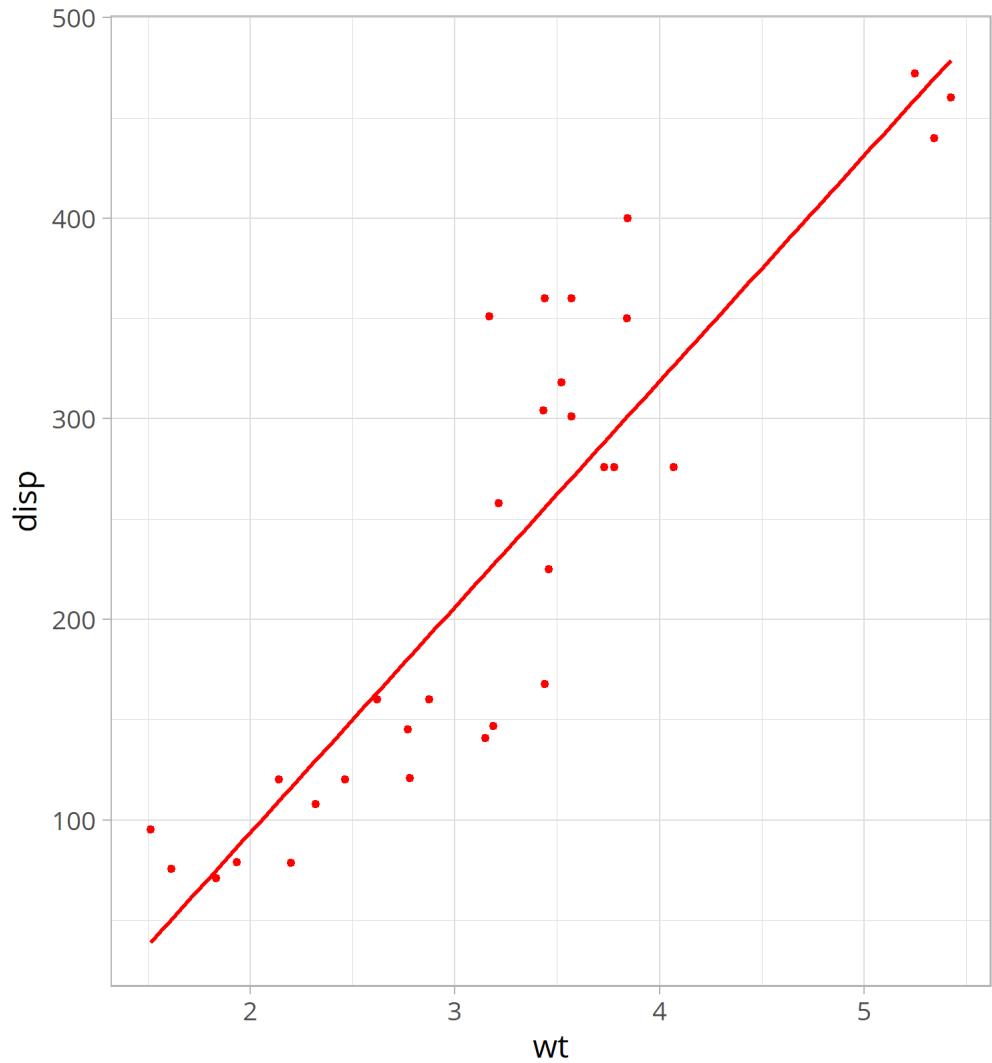
Your Turn: Scatter Plot with Linear Fitting

```
ggplot(mtcars, aes(wt, disp)) +  
  geom_point() +  
  stat_smooth(  
    method = "lm",  
    se = FALSE  
)
```



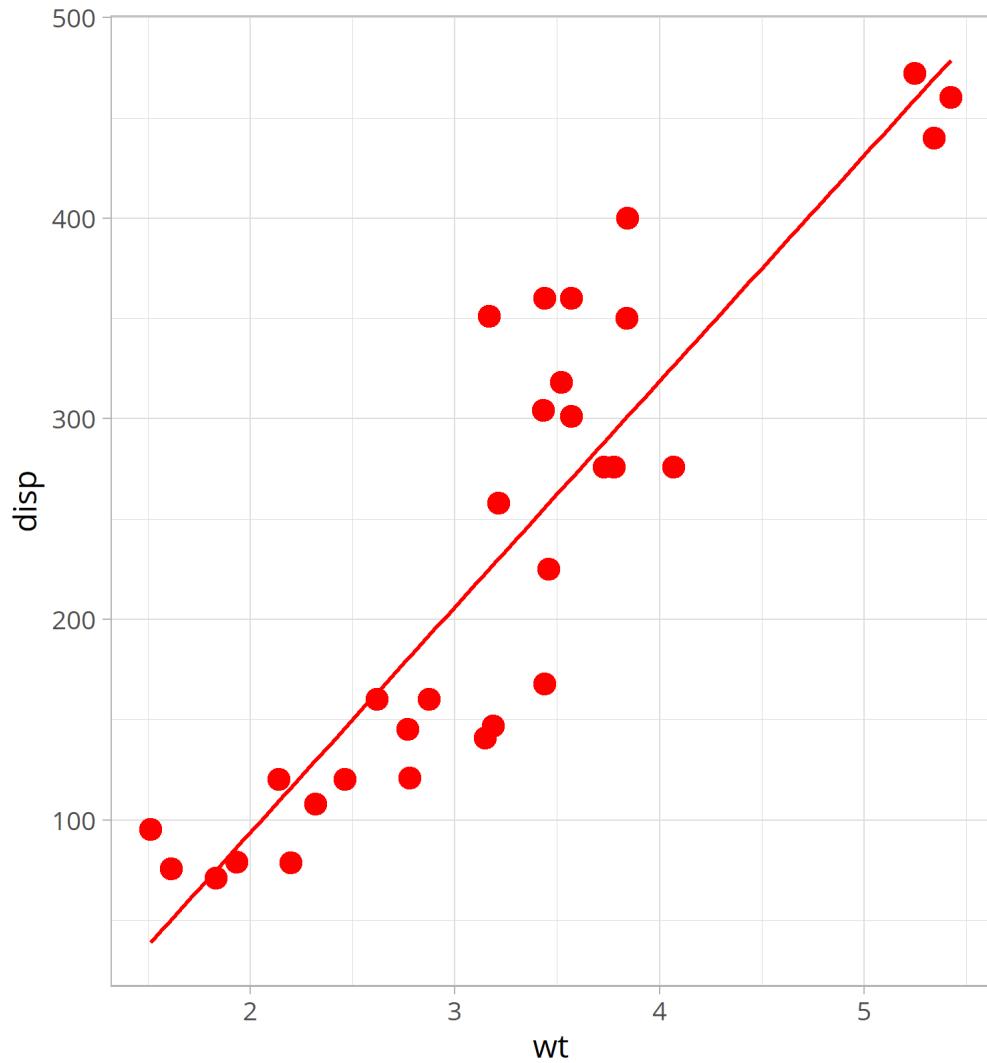
Your Turn: Scatter Plot with Linear Fitting

```
ggplot(mtcars, aes(wt, disp)) +  
  geom_point(  
    color = "red"  
) +  
  stat_smooth(  
    method = "lm",  
    se = FALSE,  
    color = "red"  
)
```



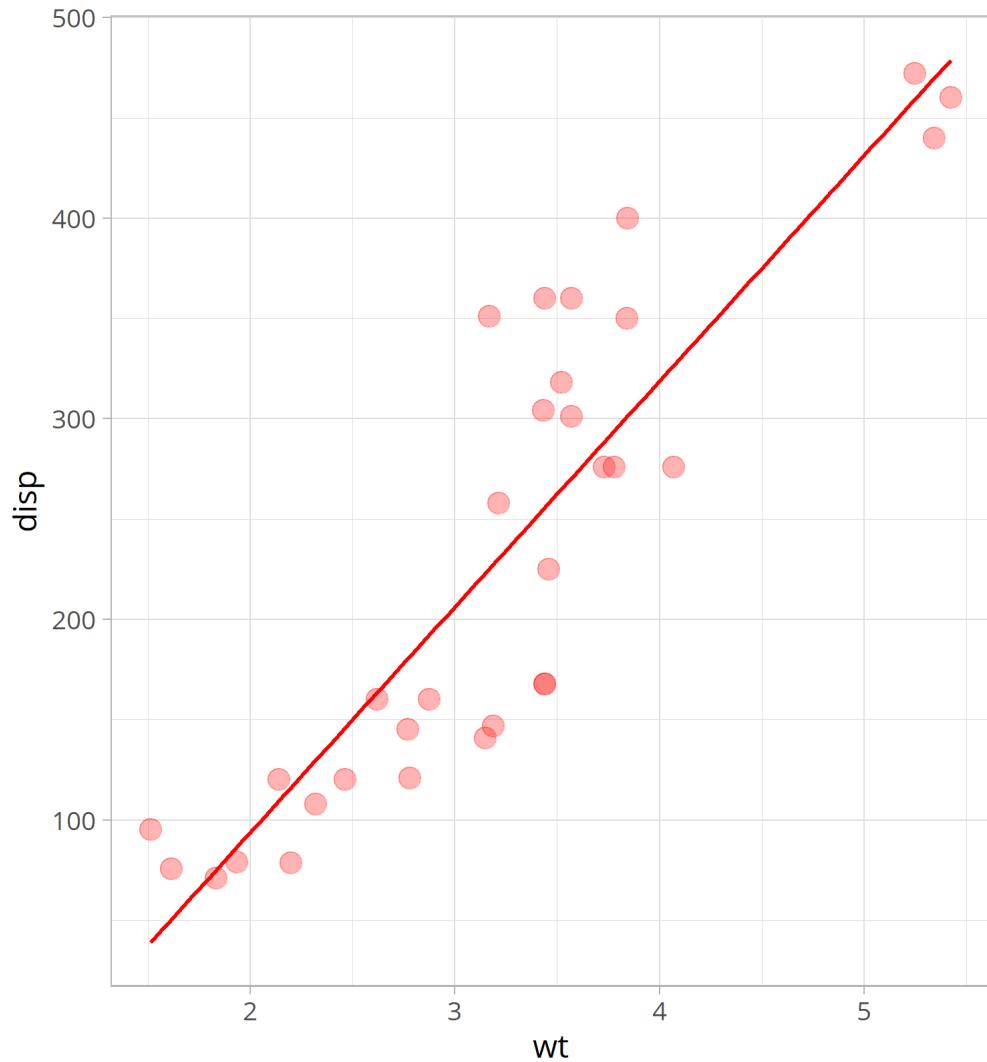
Your Turn: Scatter Plot with Linear Fitting

```
ggplot(mtcars, aes(wt, disp)) +  
  geom_point(  
    color = "red",  
    size = 5  
) +  
  stat_smooth(  
    method = "lm",  
    se = FALSE,  
    color = "red"  
)
```



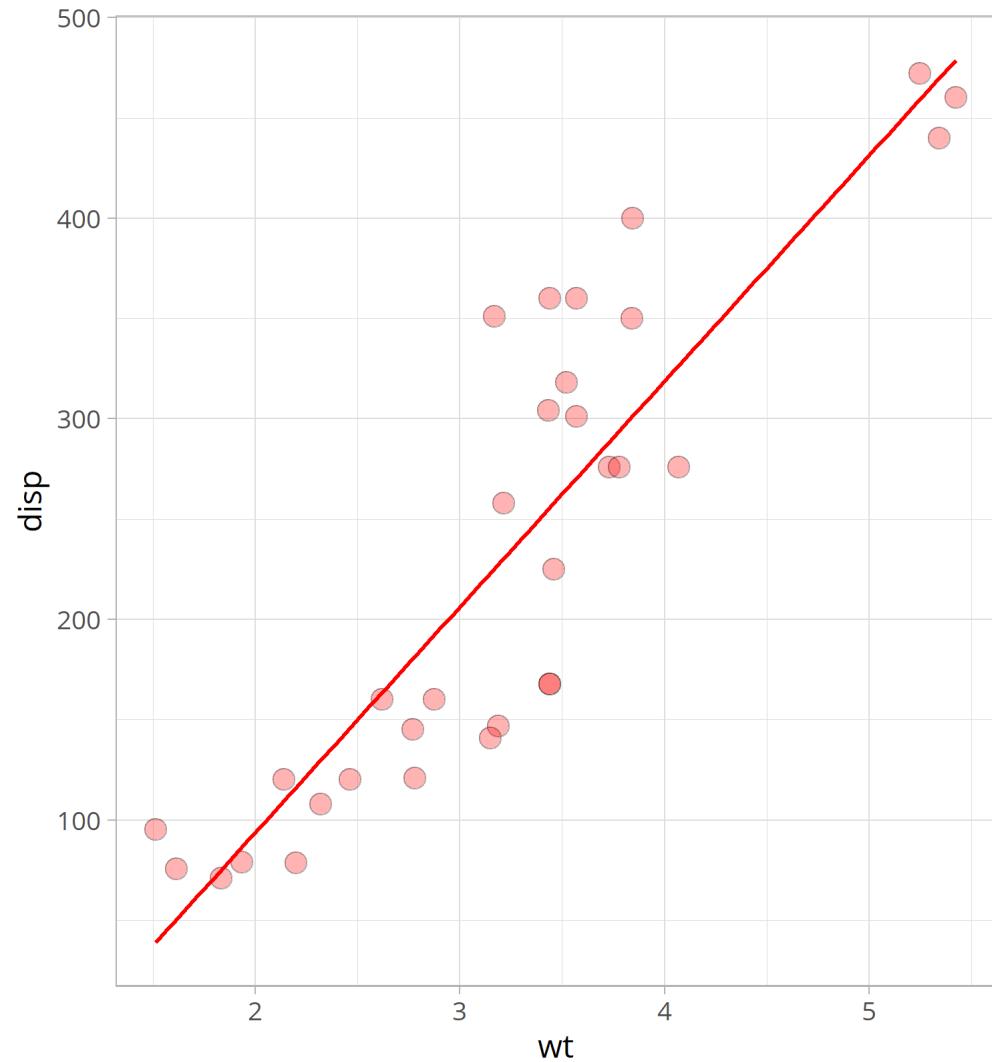
Your Turn: Scatter Plot with Linear Fitting

```
ggplot(mtcars, aes(wt, disp)) +  
  geom_point(  
    color = "red",  
    size = 5,  
    alpha = .3  
) +  
  stat_smooth(  
    method = "lm",  
    se = FALSE,  
    color = "red"  
)
```

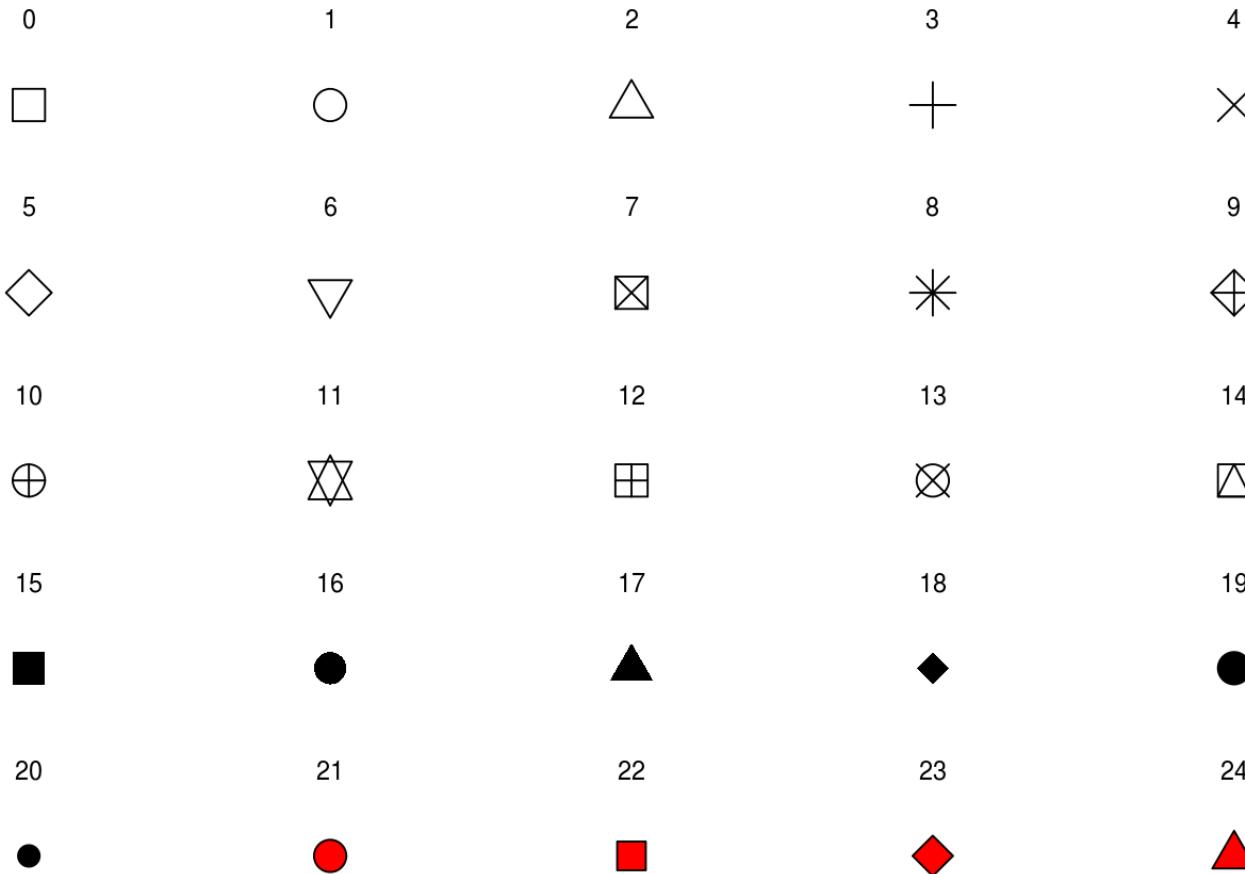


Your Turn: Scatter Plot with Linear Fitting

```
ggplot(mtcars, aes(wt, disp)) +  
  geom_point(  
    shape = 21,  
    fill = "red",  
    size = 5,  
    alpha = .3  
  ) +  
  stat_smooth(  
    method = "lm",  
    se = FALSE,  
    color = "red"  
  )
```



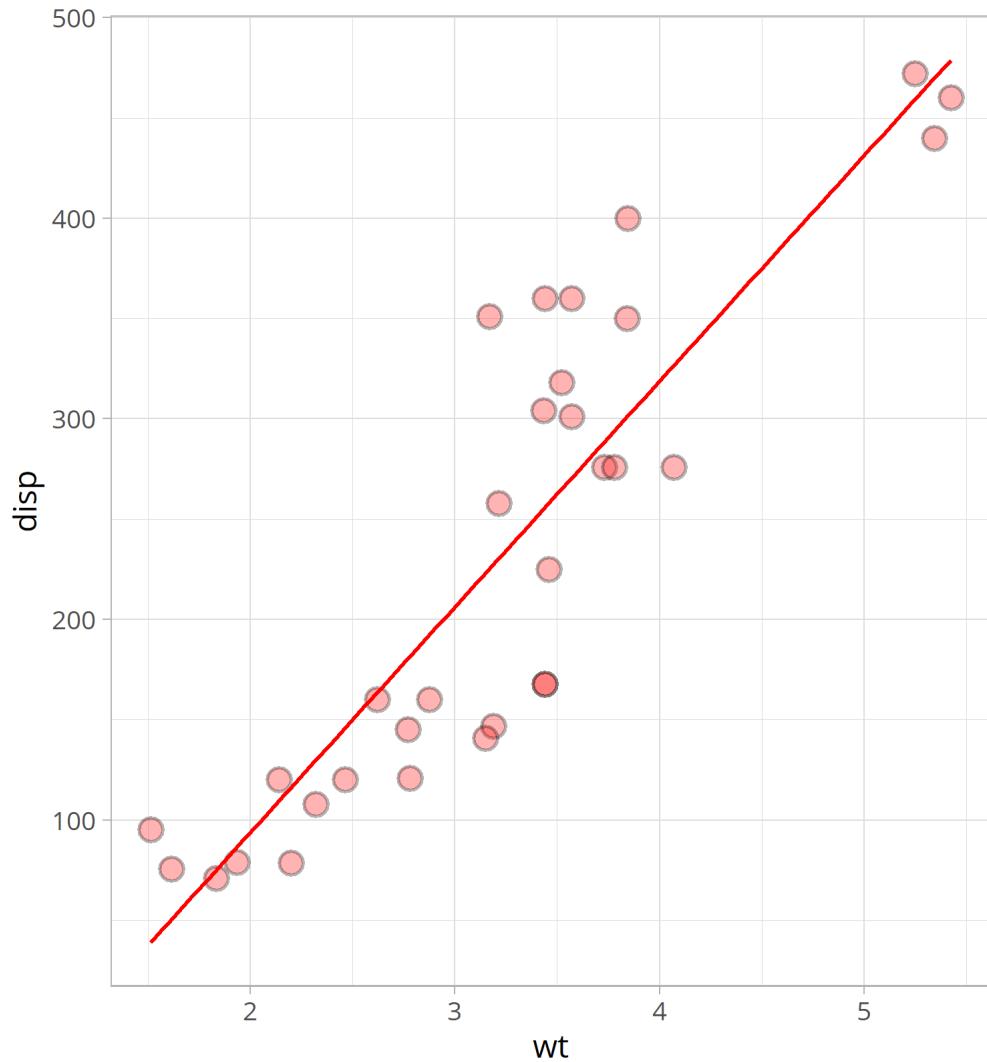
Your Turn: Scatter Plot with Linear Fitting



Source: ggplot2.tidyverse.org/articles/ggplot2-specs.html

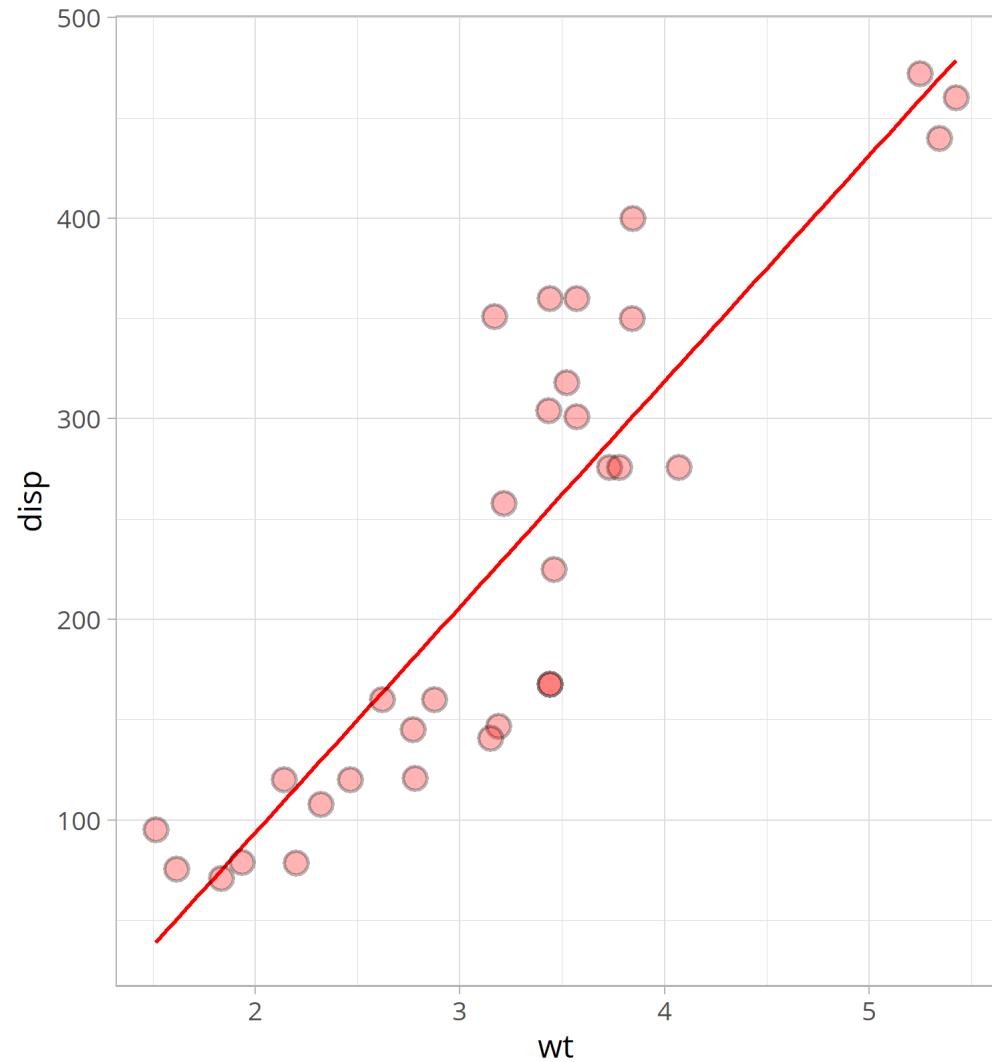
Your Turn: Scatter Plot with Linear Fitting

```
ggplot(mtcars, aes(wt, disp)) +  
  geom_point(  
    shape = 21,  
    fill = "red",  
    color = "black",  
    stroke = 1.25,  
    size = 5,  
    alpha = .3  
) +  
  stat_smooth(  
    method = "lm",  
    se = FALSE,  
    color = "red"  
)
```



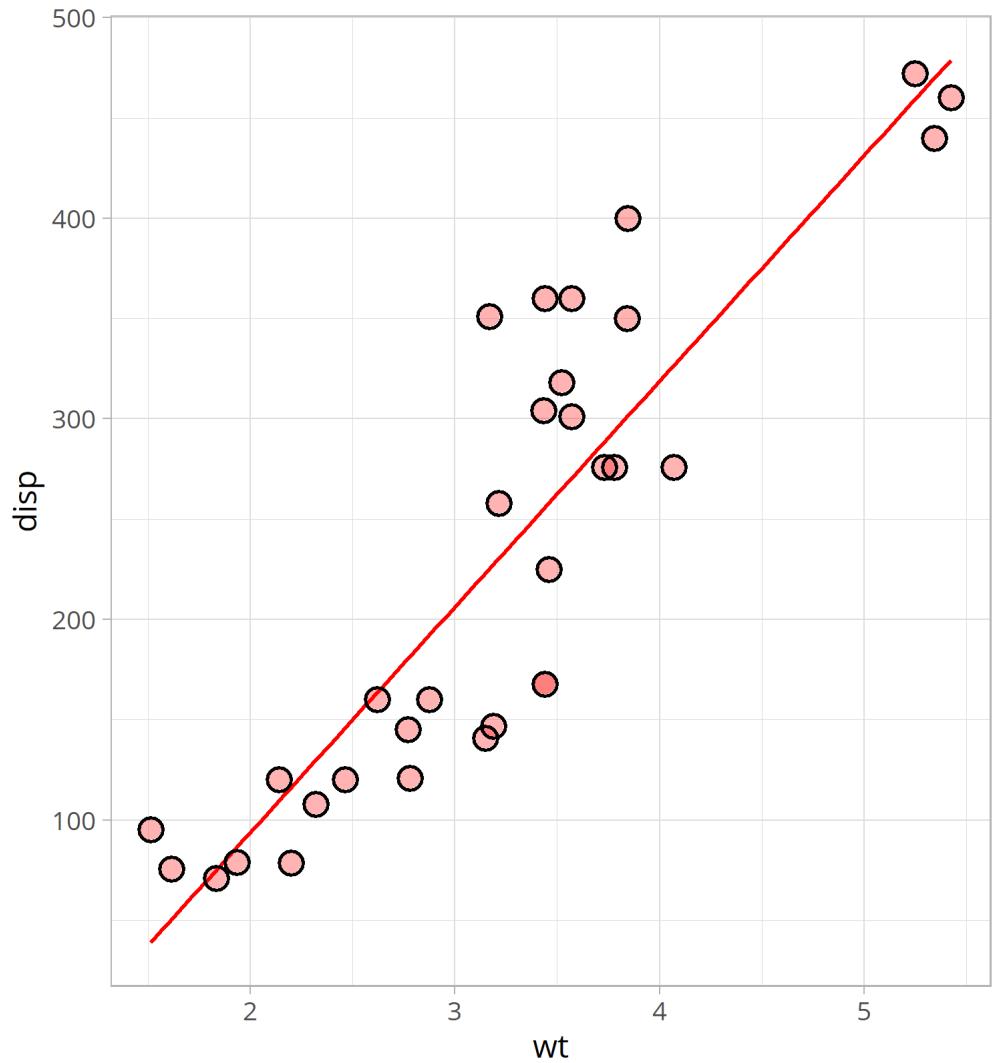
Your Turn: Scatter Plot with Linear Fitting

```
ggplot(mtcars, aes(wt, disp)) +  
  stat_smooth(  
    method = "lm",  
    se = FALSE,  
    color = "red"  
  ) +  
  geom_point(  
    shape = 21,  
    fill = "red",  
    color = "black",  
    stroke = 1.25,  
    size = 5,  
    alpha = .3  
  )
```



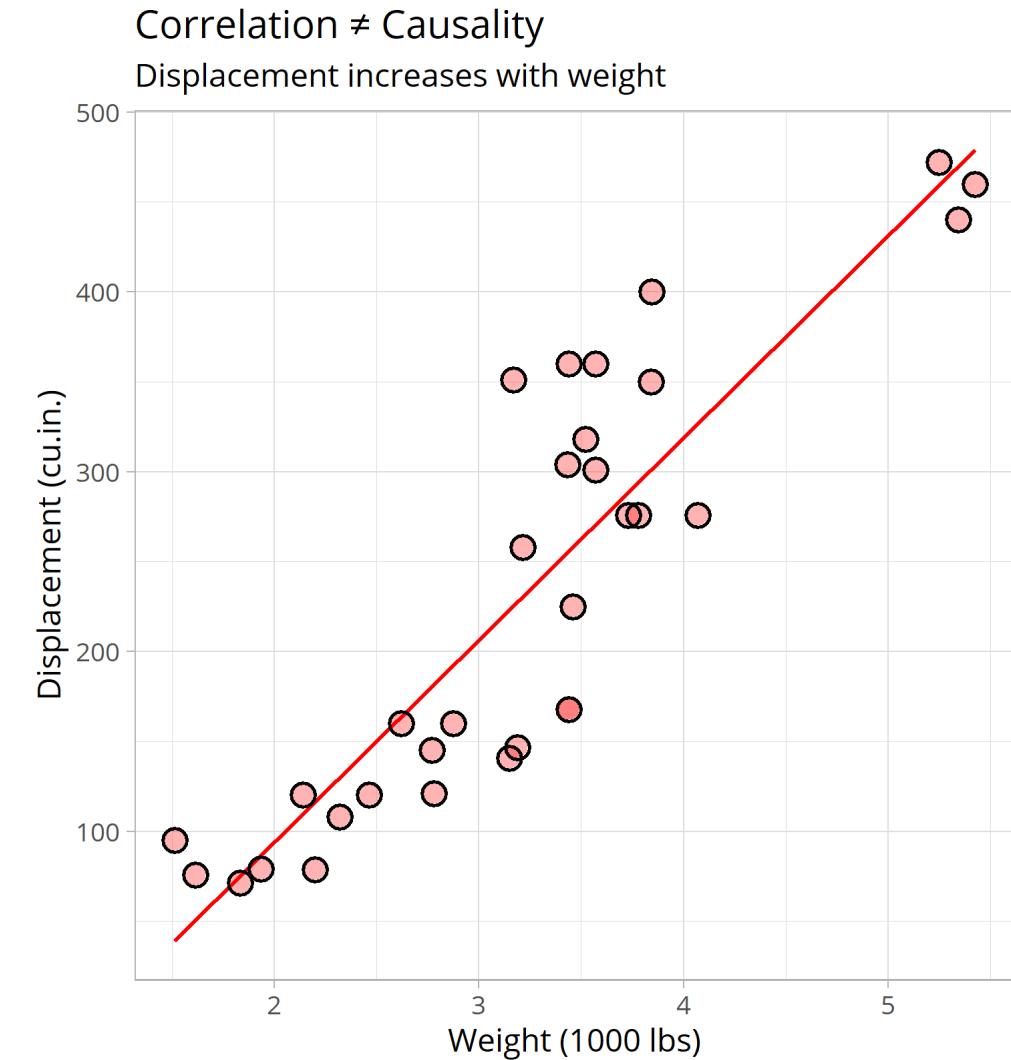
Your Turn: Scatter Plot with Linear Fitting

```
ggplot(mtcars, aes(wt, disp)) +  
  stat_smooth(  
    method = "lm",  
    se = FALSE,  
    color = "red"  
  ) +  
  geom_point(  
    shape = 21,  
    fill = "red",  
    color = "transparent",  
    size = 5,  
    alpha = .3  
  ) +  
  geom_point(  
    shape = 21,  
    fill = "transparent",  
    color = "black",  
    stroke = 1.25,  
    size = 5  
  )
```



Your Turn: Scatter Plot with Linear Fitting

```
ggplot(mtcars, aes(wt, disp)) +  
  stat_smooth(method = "lm",  
             se = FALSE,  
             color = "red") +  
  geom_point(shape = 21,  
             fill = "red",  
             color = "transparent",  
             size = 5,  
             alpha = .3) +  
  geom_point(shape = 21,  
             fill = "transparent",  
             color = "black",  
             stroke = 1.25,  
             size = 5) +  
  labs(x = "Weight (1000 lbs)",  
       y = "Displacement (cu.in.)",  
       title = "Correlation \u2260 Causality",  
       subtitle = "Displacement increases with weight")
```



Aesthetics

aes()

aes()

Aesthetics of the geometric and statistical objects, such as

- position via `x`, `y`, `xmin`, `xmax`, `ymin`, `ymax`, ...
- color via `color` and `fill`
- transparency via `alpha`
- size via `size` and `width`
- shape via `shape` and `linetype`

In general, everything which maps to the data needs to be wrapped in `aes()` (such as `x` and `y` in our previous examples) while static arguments are placed outside the `aes()` (such as `color` and `alpha` in our previous examples).

e.g.

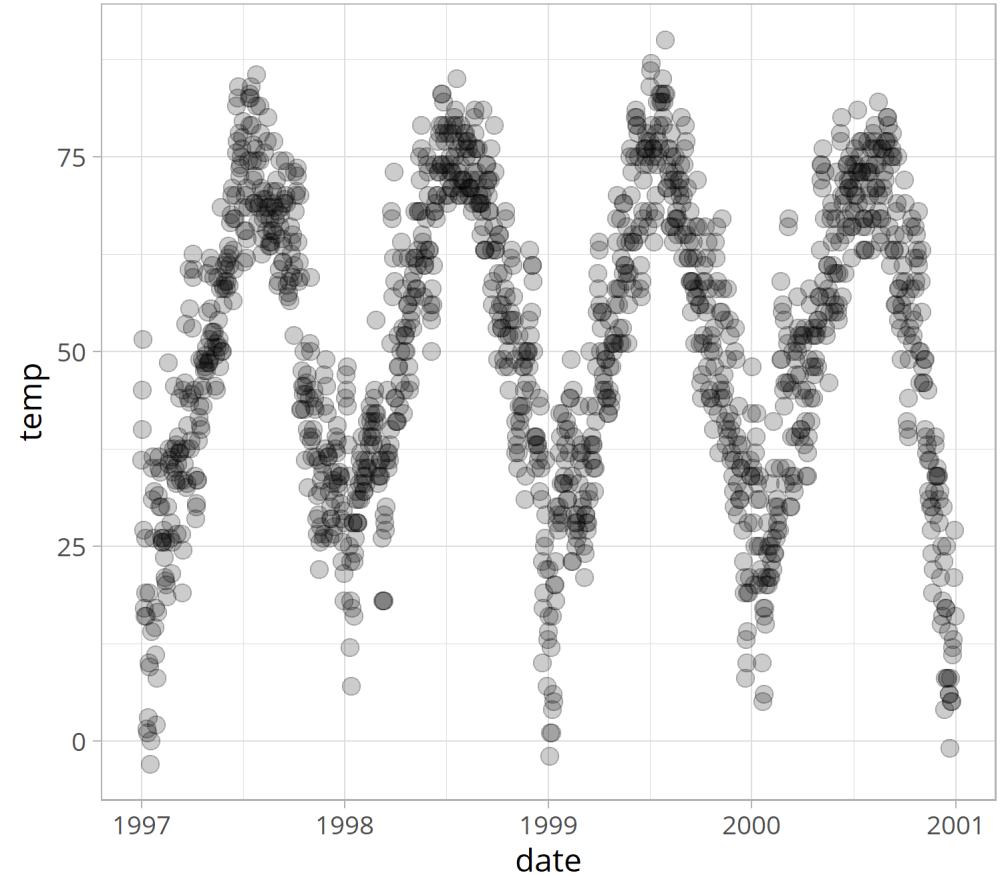
`geom_point(color = "red")` to color all points in the same color

`geom_point(aes(color = season))` to color points based on the variable `season`

aes(color/fill/alpha/size/shape)

This way, we can enlarge all points and add some transparency:

```
ggplot(chic, aes(date, temp)) +  
  geom_point(  
    size = 4,  
    alpha = .2  
  )
```

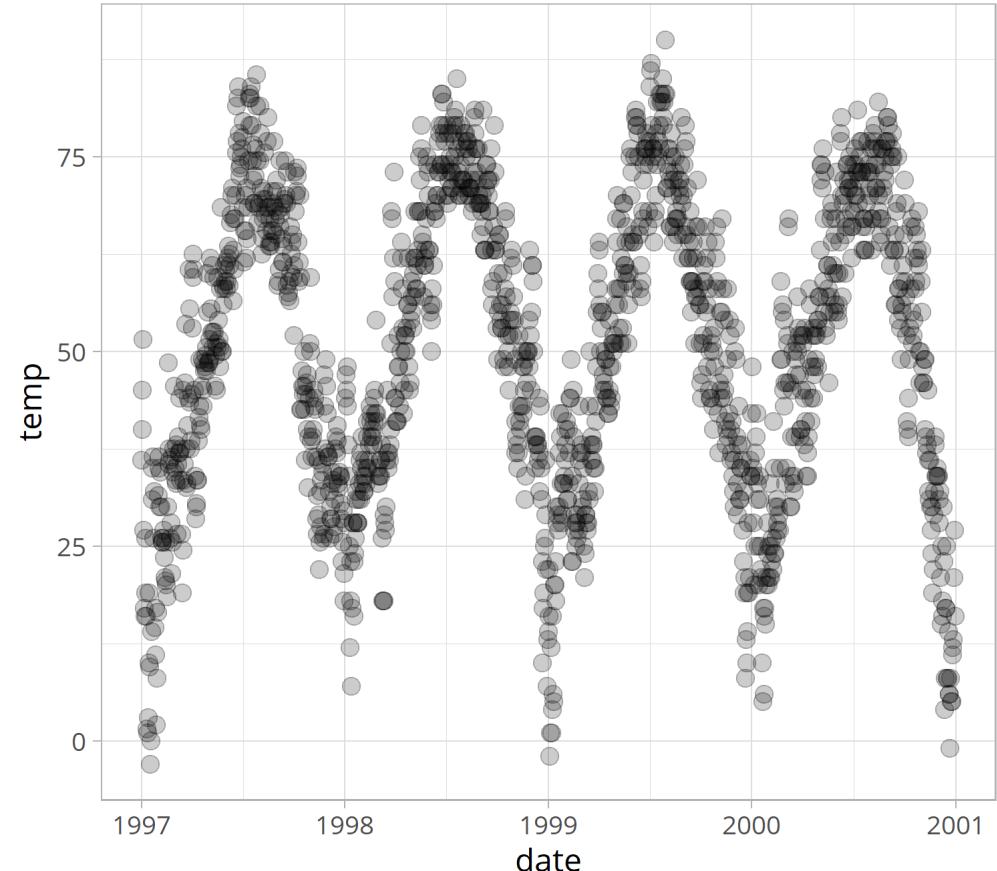


aes(color/fill/alpha/size/shape)

This way, we can enlarge all points and add some transparency:

```
ggplot(chic, aes(date, temp)) +  
  geom_point(  
    size = 4,  
    alpha = .2  
  )
```

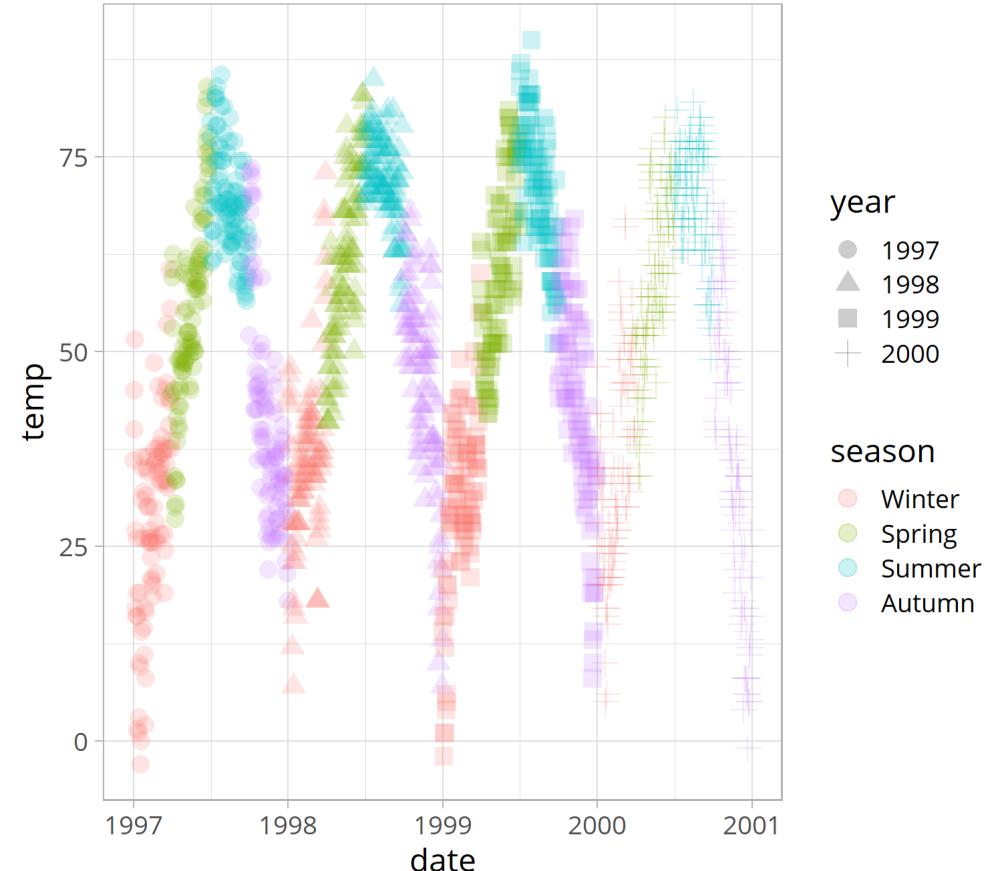
The changes in aesthetics are applied to all data points.



aes(color/fill/alpha/size/shape)

... and change the color and the shape based on **season** and **year**:

```
ggplot(chic, aes(date, temp)) +  
  geom_point(  
    aes(  
      color = season,  
      shape = year  
    ),  
    size = 4,  
    alpha = .2  
  )
```

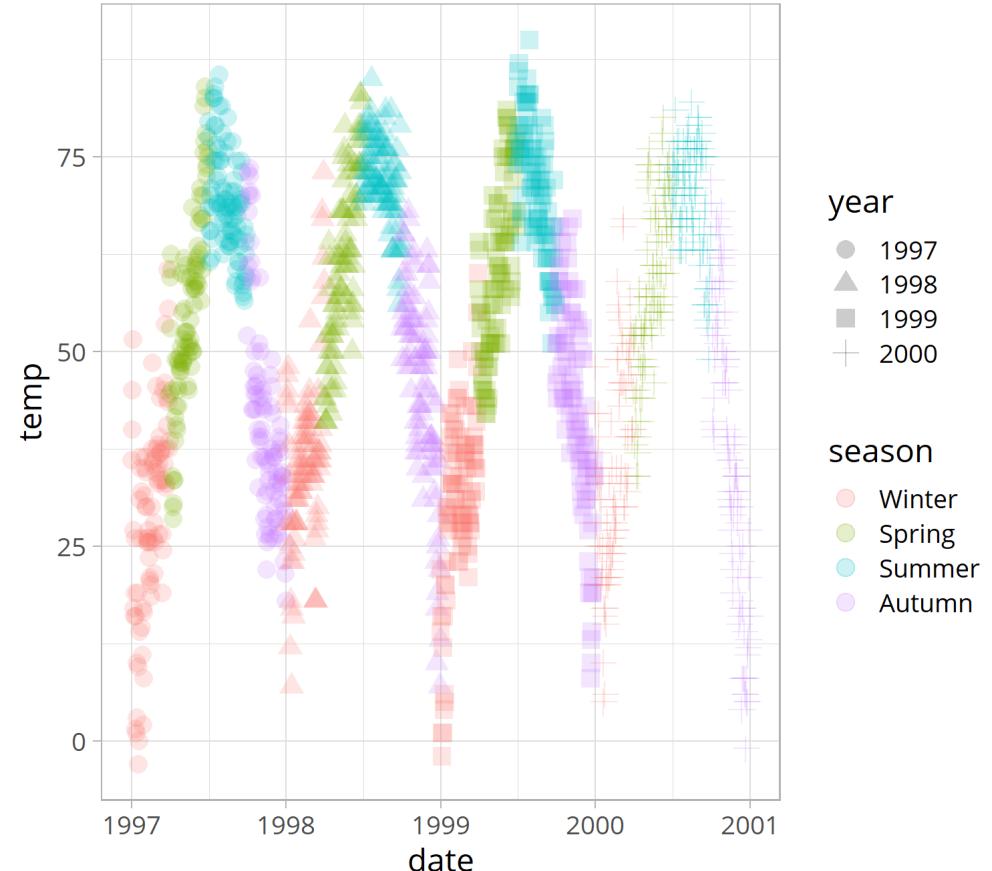


aes(color/fill/alpha/size/shape)

... and change the color and the shape based on **season** and **year**:

```
ggplot(chic, aes(date, temp)) +  
  geom_point(  
    aes(  
      color = season,  
      shape = year  
    ),  
    size = 4,  
    alpha = .2  
  )
```

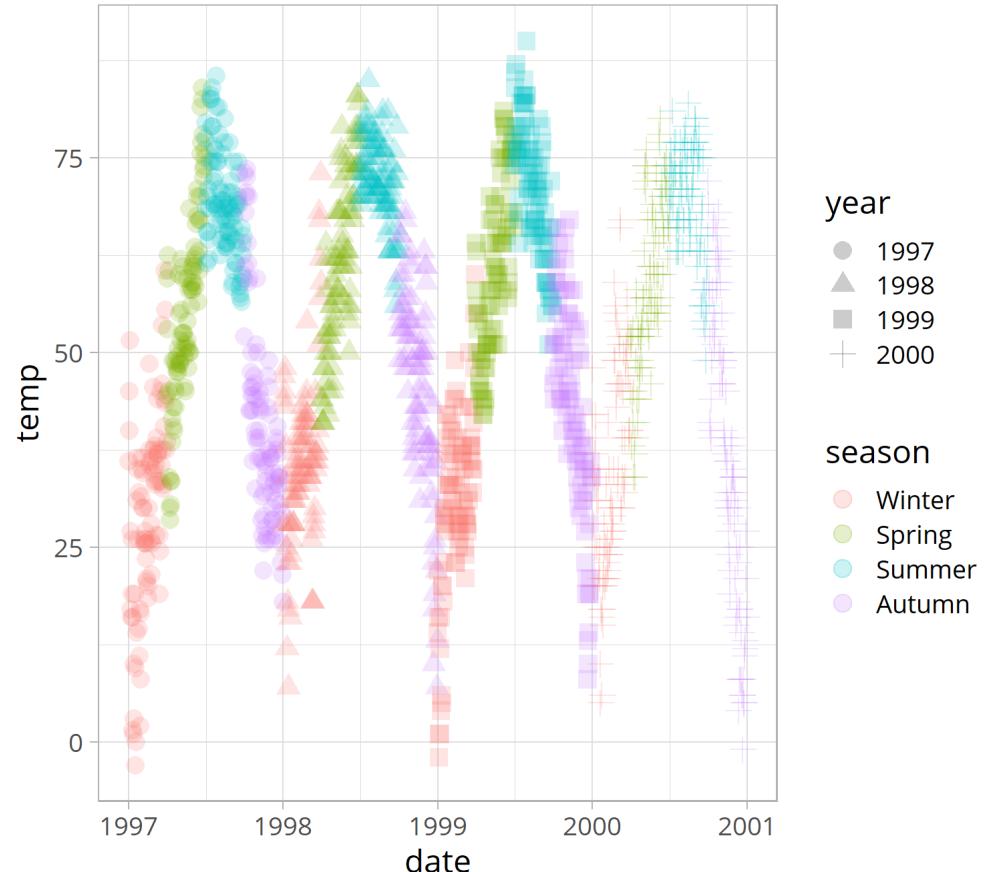
The changes in aesthetics are applied to each group.



Global aes()

Alternatively, all aesthetics can be grouped together (and are then applied to all `geom_*`() and `stats_*`()):

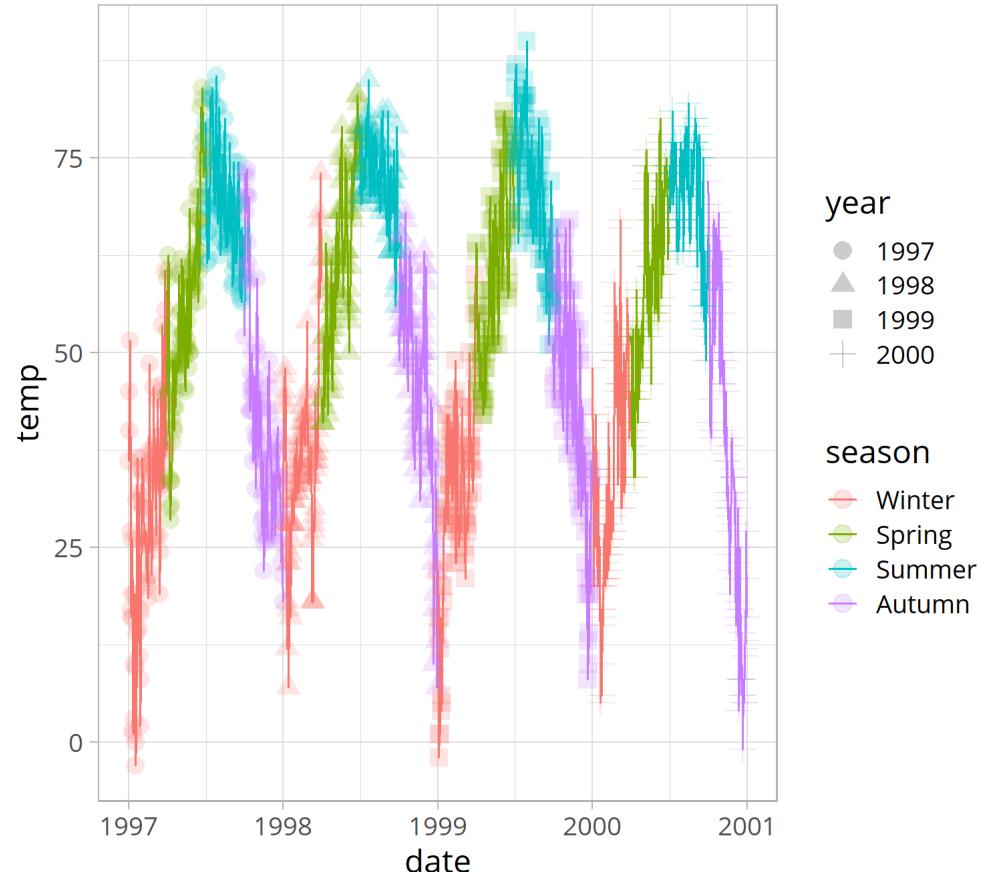
```
ggplot(  
  chic,  
  aes(date,  
      temp,  
      color = season,  
      shape = year)  
) +  
  geom_point(  
    size = 4,  
    alpha = .2  
)
```



Global aes()

Alternatively, all aesthetics can be grouped together (and are then applied to all `geom_*`() and `stats_*`()):

```
ggplot(  
  chic,  
  aes(date,  
      temp,  
      color = season,  
      shape = year)  
 ) +  
 geom_line() +  
 geom_point(  
   size = 4,  
   alpha = .2  
 )
```

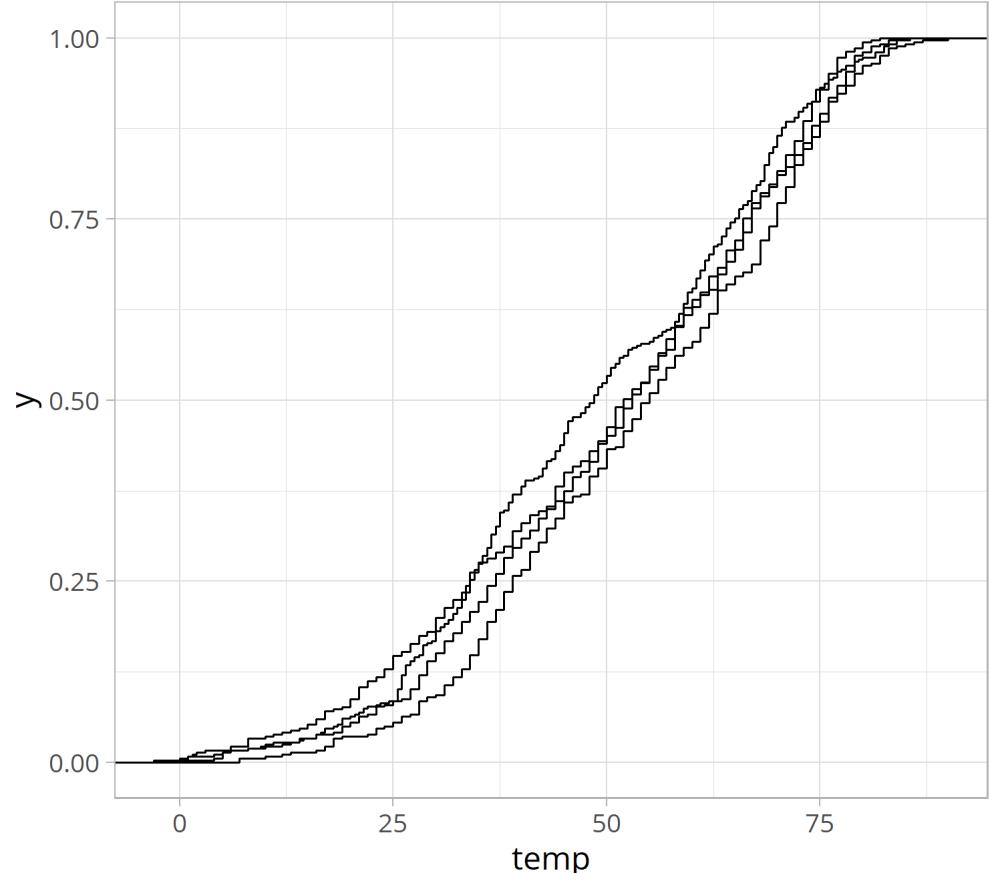


aes(group)

You can create subsets of the data by specifying a grouping variable via `group`:

```
ggplot(chic, aes(temp)) +  
  stat_ecdf(aes(group = year))
```

However, for most applications you can simply specify the grouping using visual aesthetics (`color`, `fill`, `size`, `alpha`, `shape`, `linetype`).



Your Turn!

- Run the same code without the grouping. Guess first what's going to happen.
- Replace the

`stat_ecdf(aes(group = year))` by `geom_boxplot(aes(group = year))`

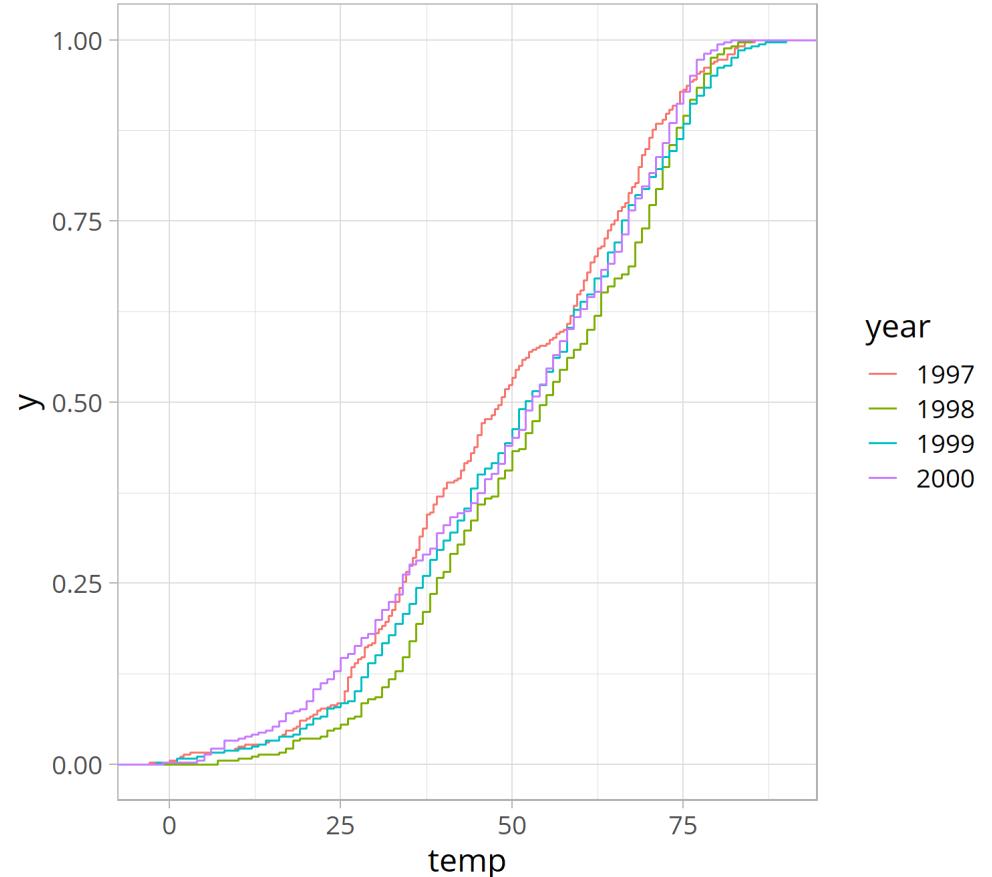
and run the code. How does this plot differ from previous ones?

- Replace the `group` argument by `color`, `linetype`, `shape` and `alpha`. What happens?

aes(color)

However, for most applications you can simply specify the grouping using visual aesthetics.

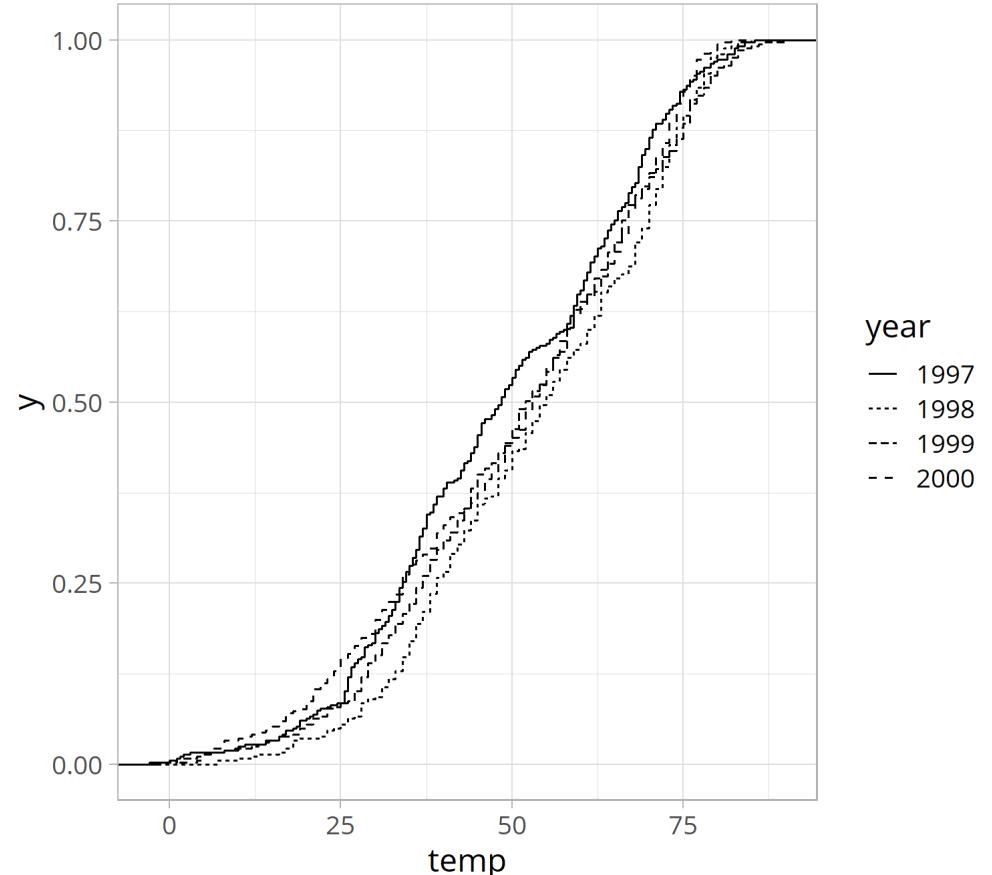
```
ggplot(chic, aes(temp)) +  
  stat_ecdf(aes(color = year))
```



aes(linetype)

However, for most applications you can simply specify the grouping using visual aesthetics.

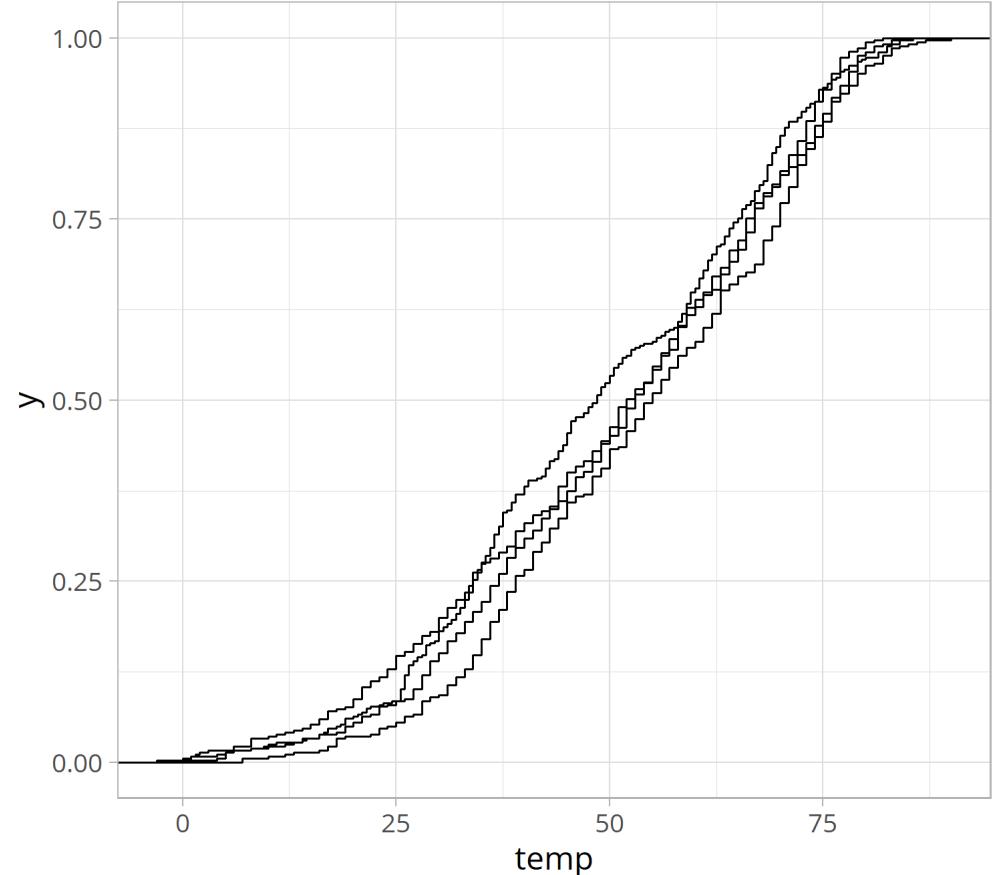
```
ggplot(chic, aes(temp)) +  
  stat_ecdf(aes(linetype = year))
```



aes(shape)

However, for most applications you can simply specify the grouping using visual aesthetics.

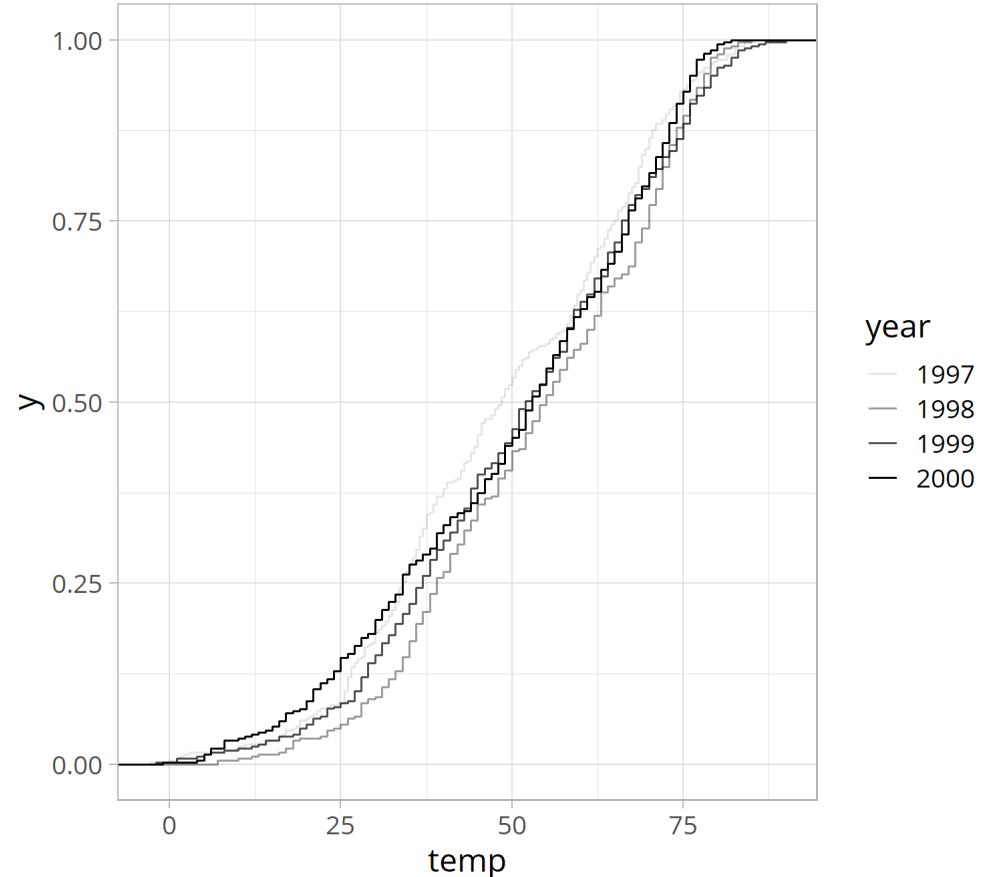
```
ggplot(chic, aes(temp)) +  
  stat_ecdf(aes(shape = year))
```



aes(alpha)

However, for most applications you can simply specify the grouping using visual aesthetics.

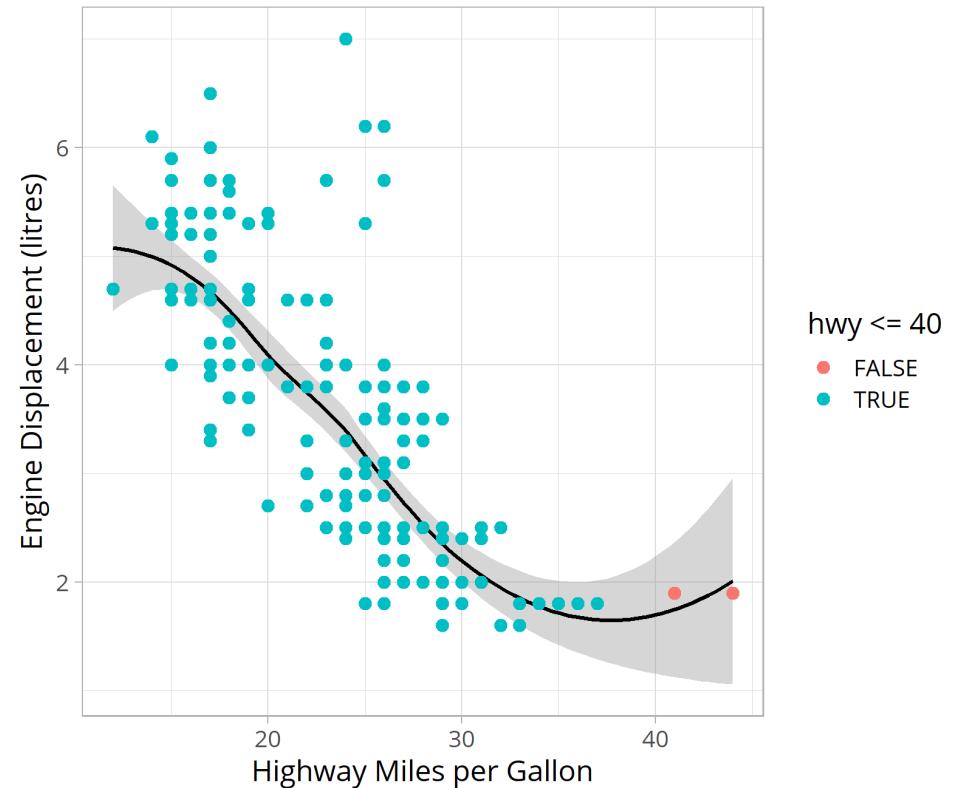
```
ggplot(chic, aes(temp)) +  
  stat_ecdf(aes(alpha = year))
```



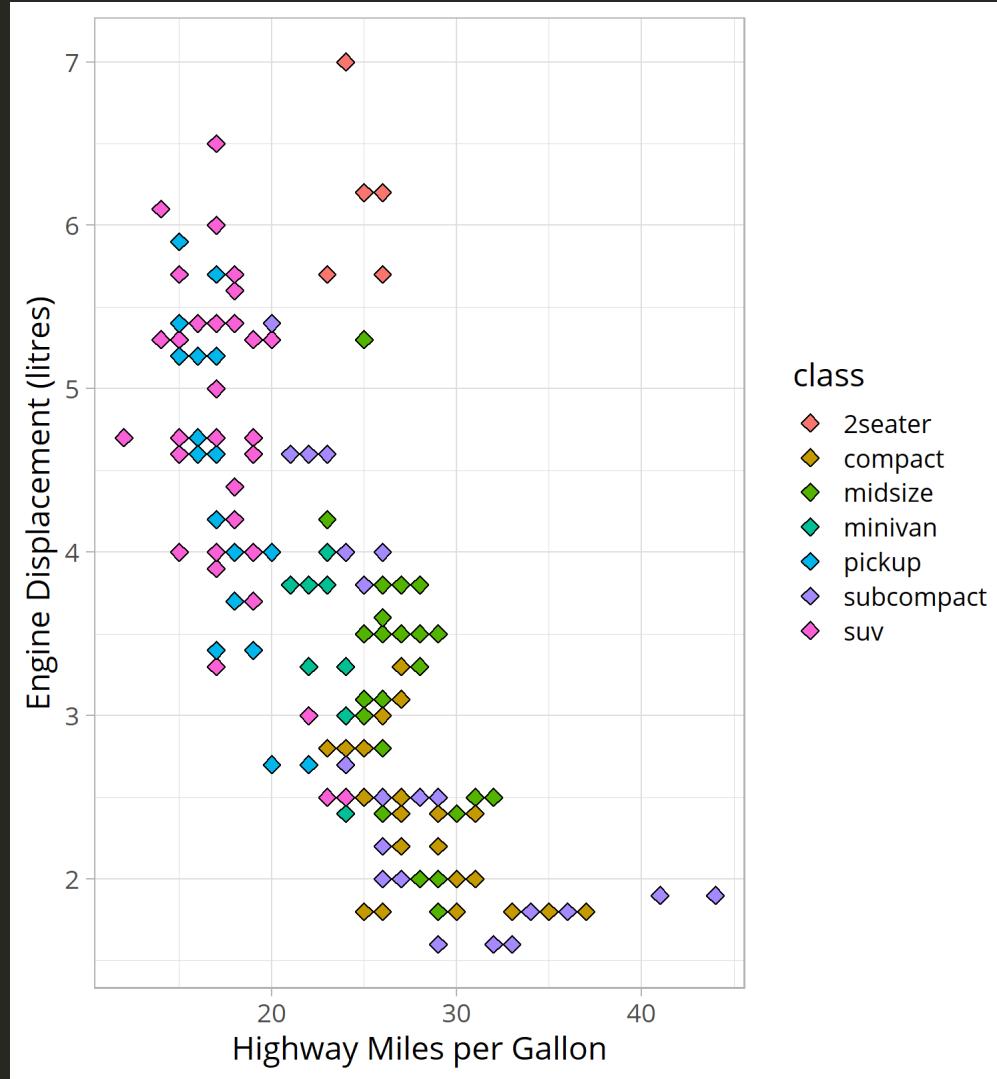
Your Turn!

Another training data set focussing on cars comes with `ggplot2` and is called `mpg`:

- Create the following visualizations:

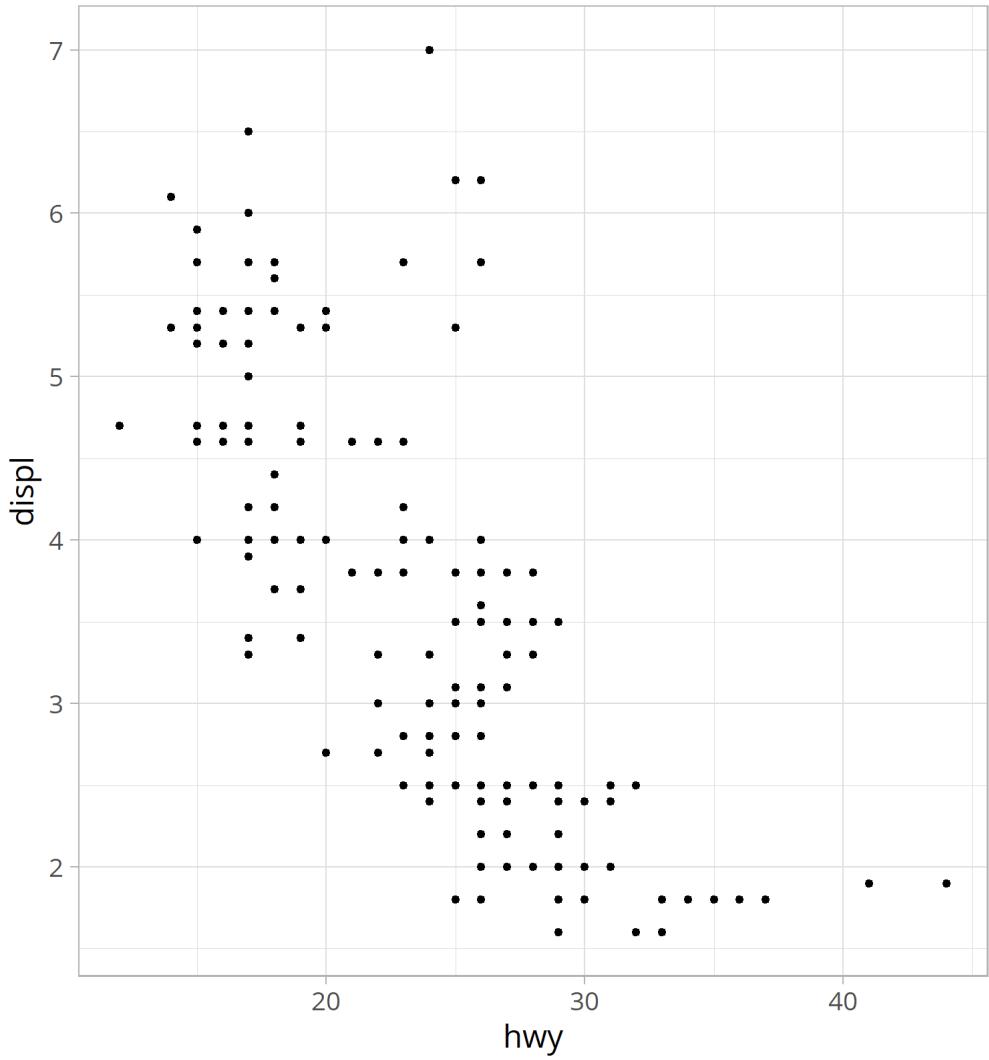


Your Turn: Create this Scatter Plot!



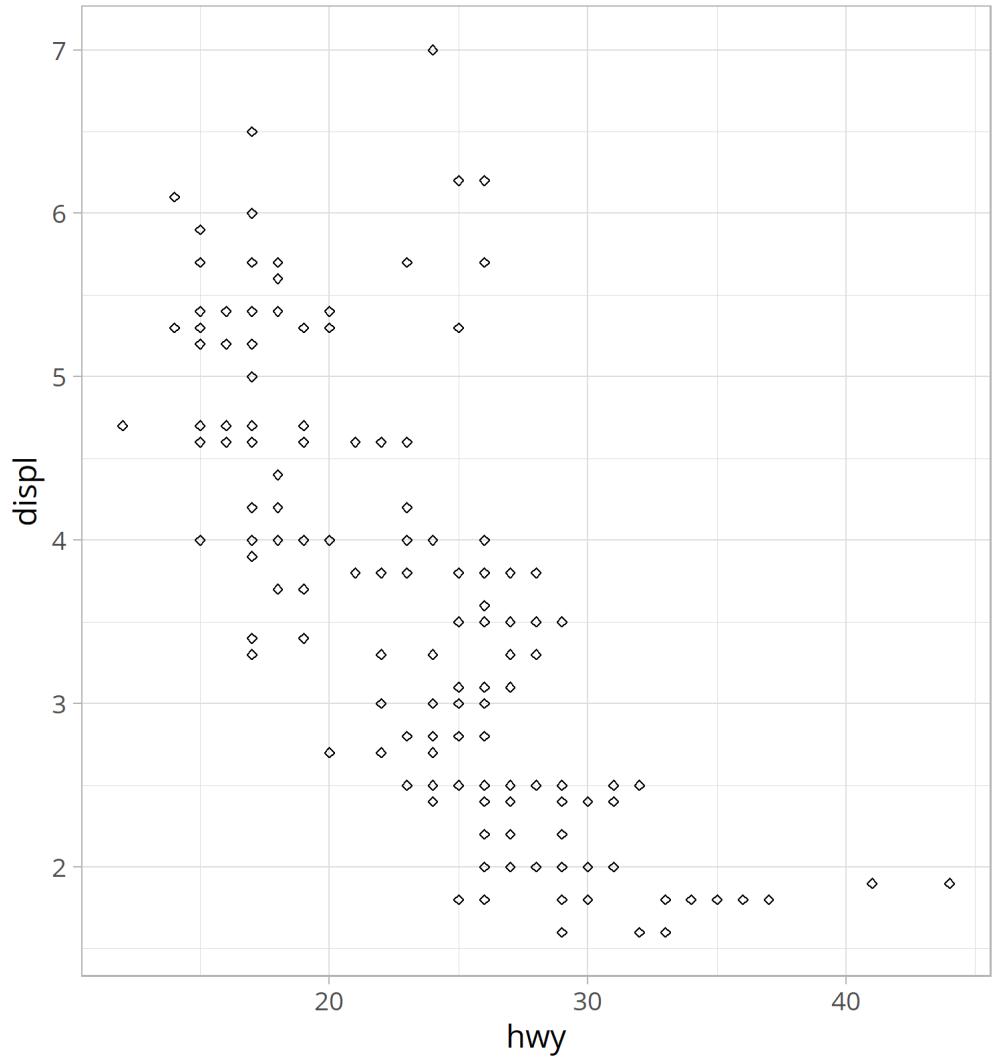
Your Turn: Scatter Plot with Diamonds

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



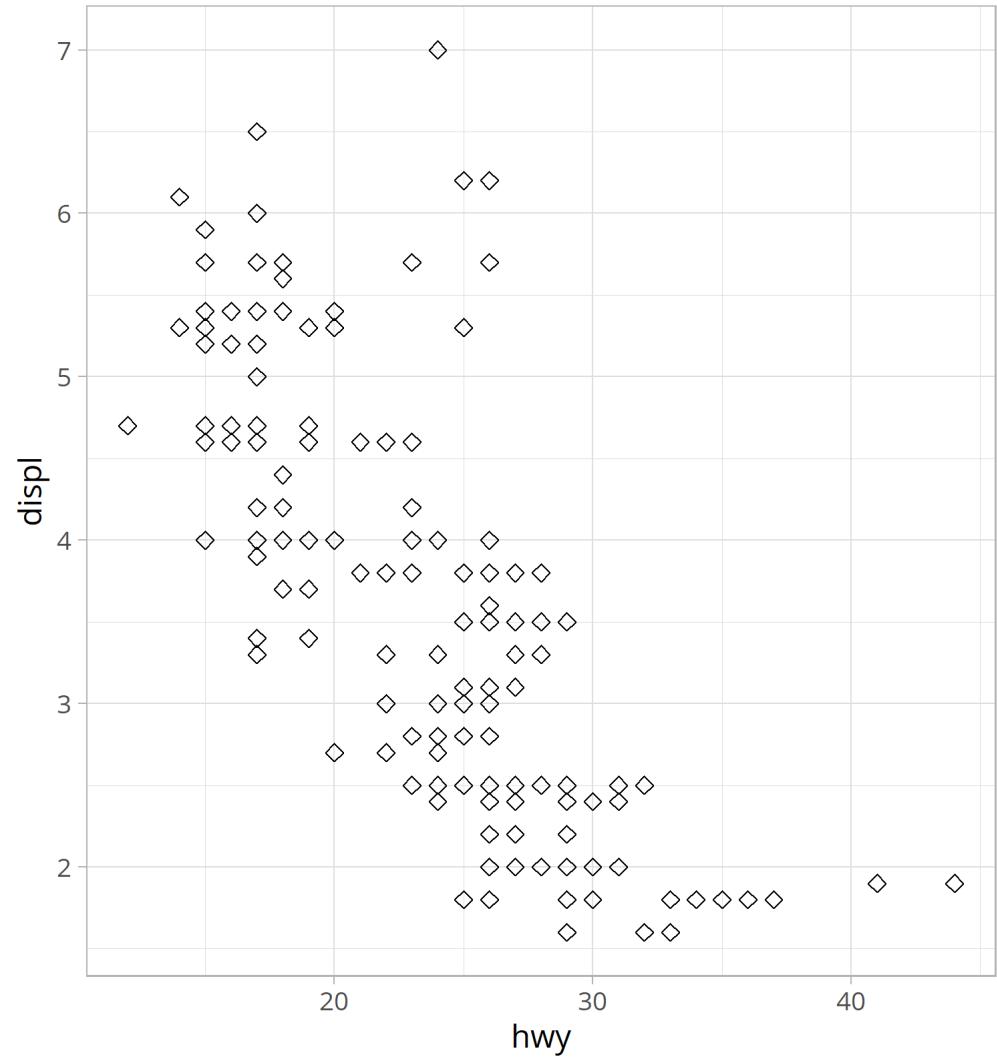
Your Turn: Scatter Plot with Diamonds

```
ggplot(mpg, aes(hwy, displ)) +  
  geom_point(  
    shape = 23  
)
```



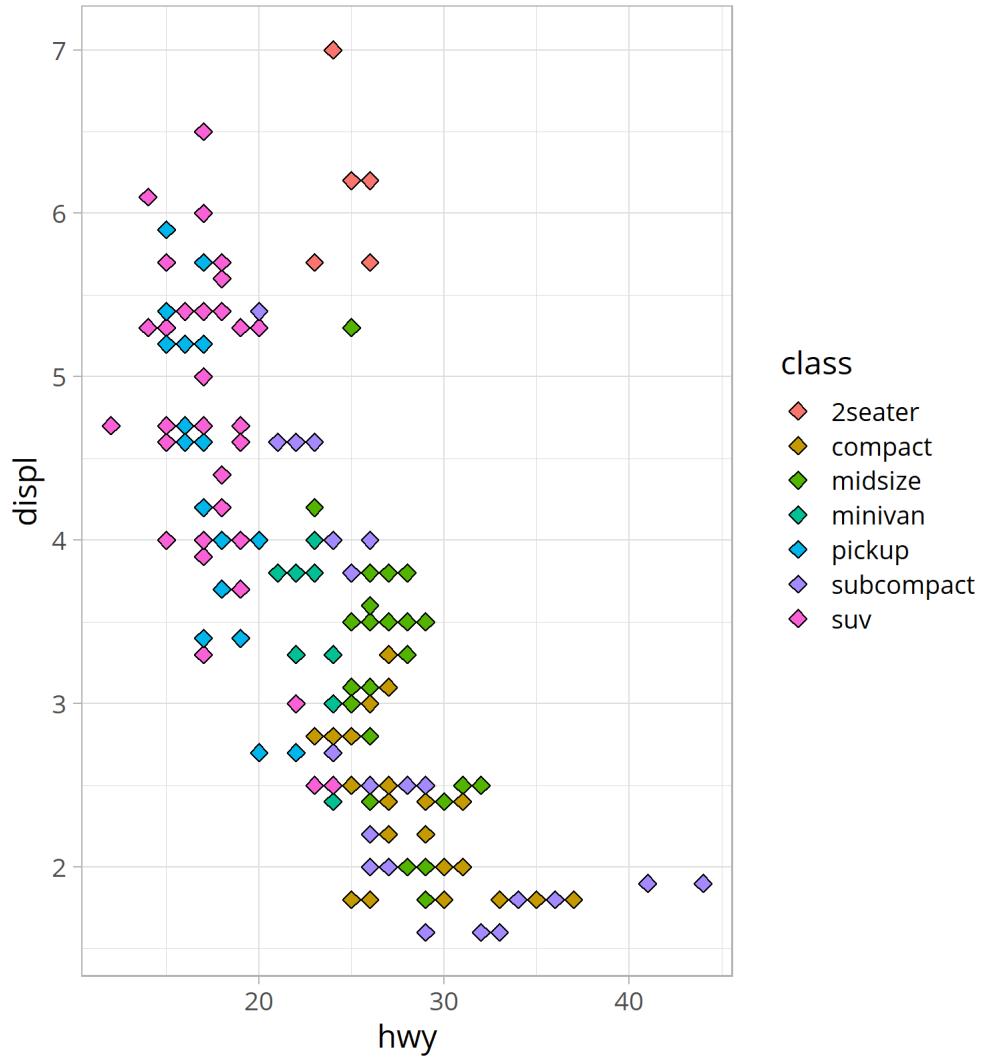
Your Turn: Scatter Plot with Diamonds

```
ggplot(mpg, aes(hwy, displ)) +  
  geom_point(  
    shape = 23,  
    size = 3  
)
```



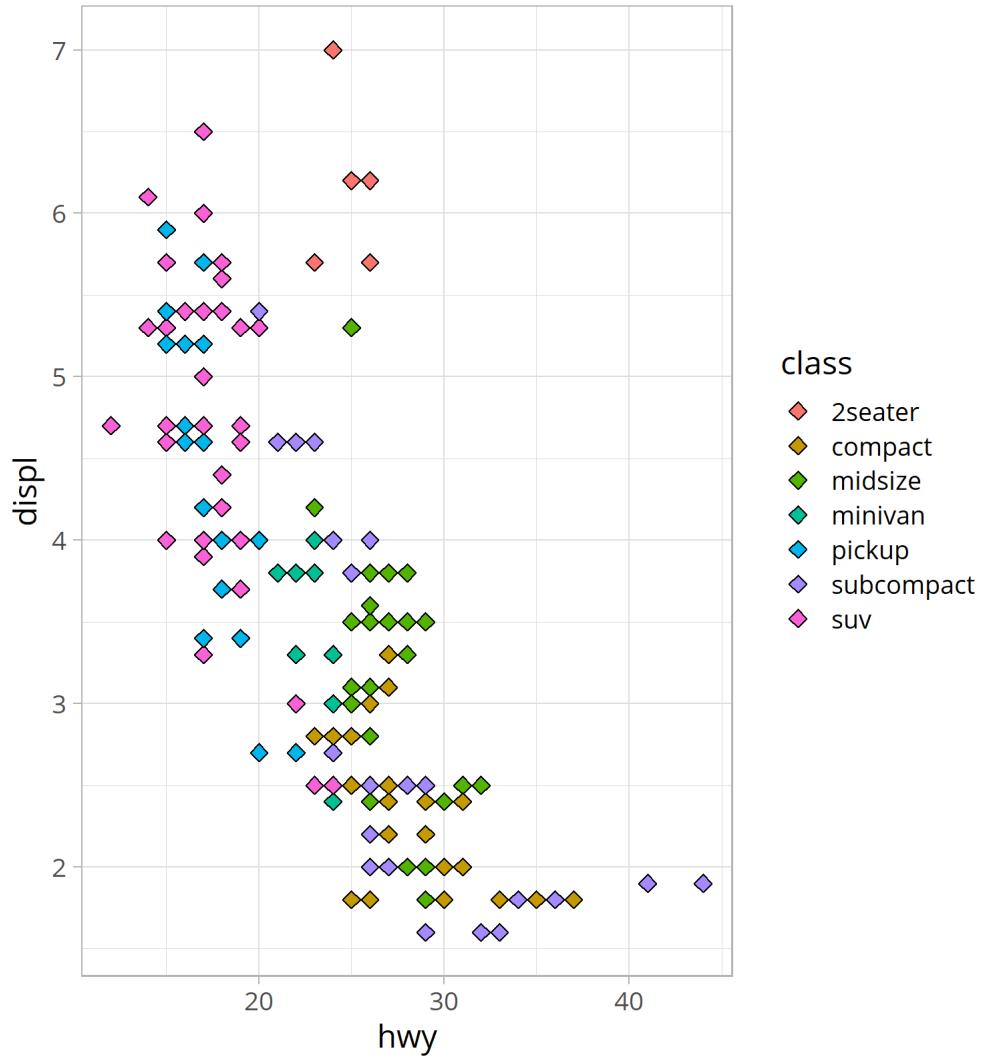
Your Turn: Scatter Plot with Diamonds

```
ggplot(mpg, aes(hwy, displ)) +  
  geom_point(  
    mapping = aes(fill = class),  
    shape = 23,  
    size = 3  
)
```



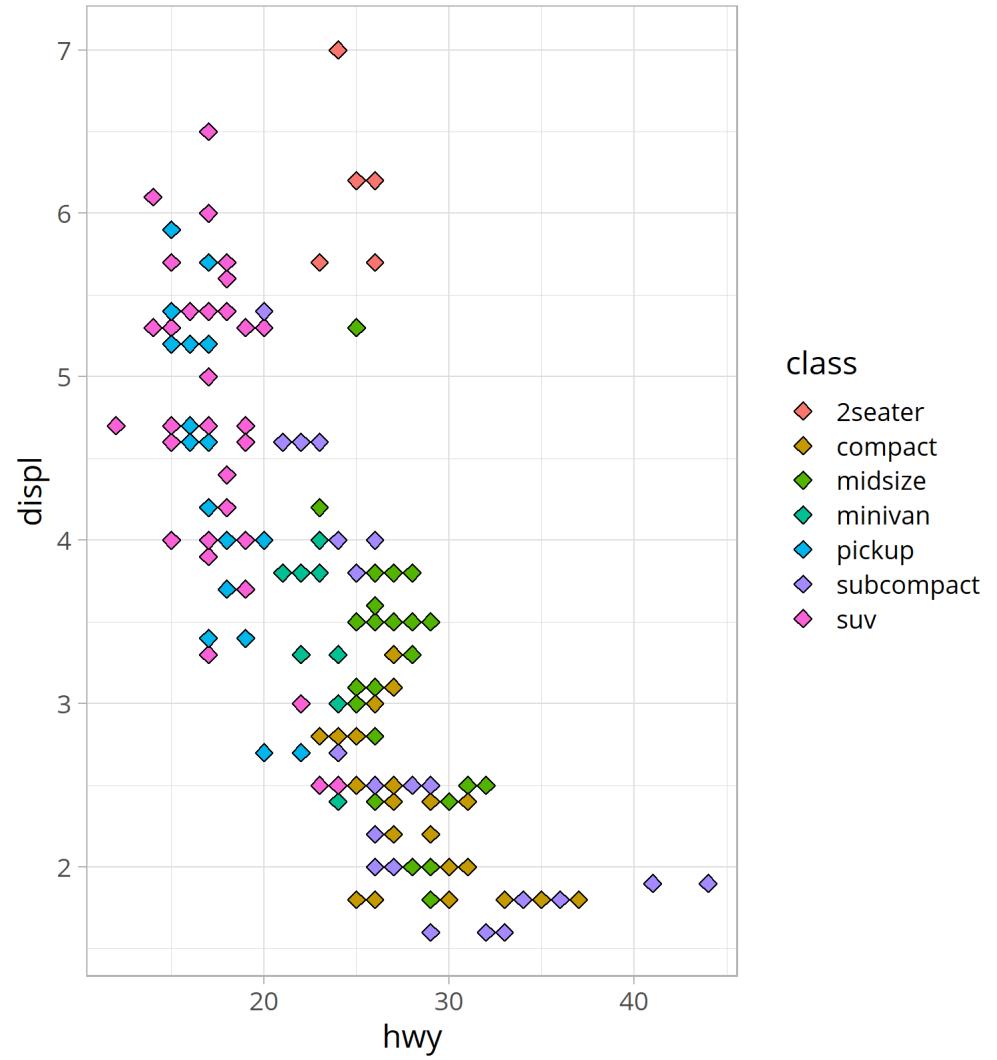
Your Turn: Scatter Plot with Diamonds

```
ggplot(mpg, aes(hwy, displ)) +  
  geom_point(  
    aes(fill = class),  
    shape = 23,  
    size = 3  
)
```



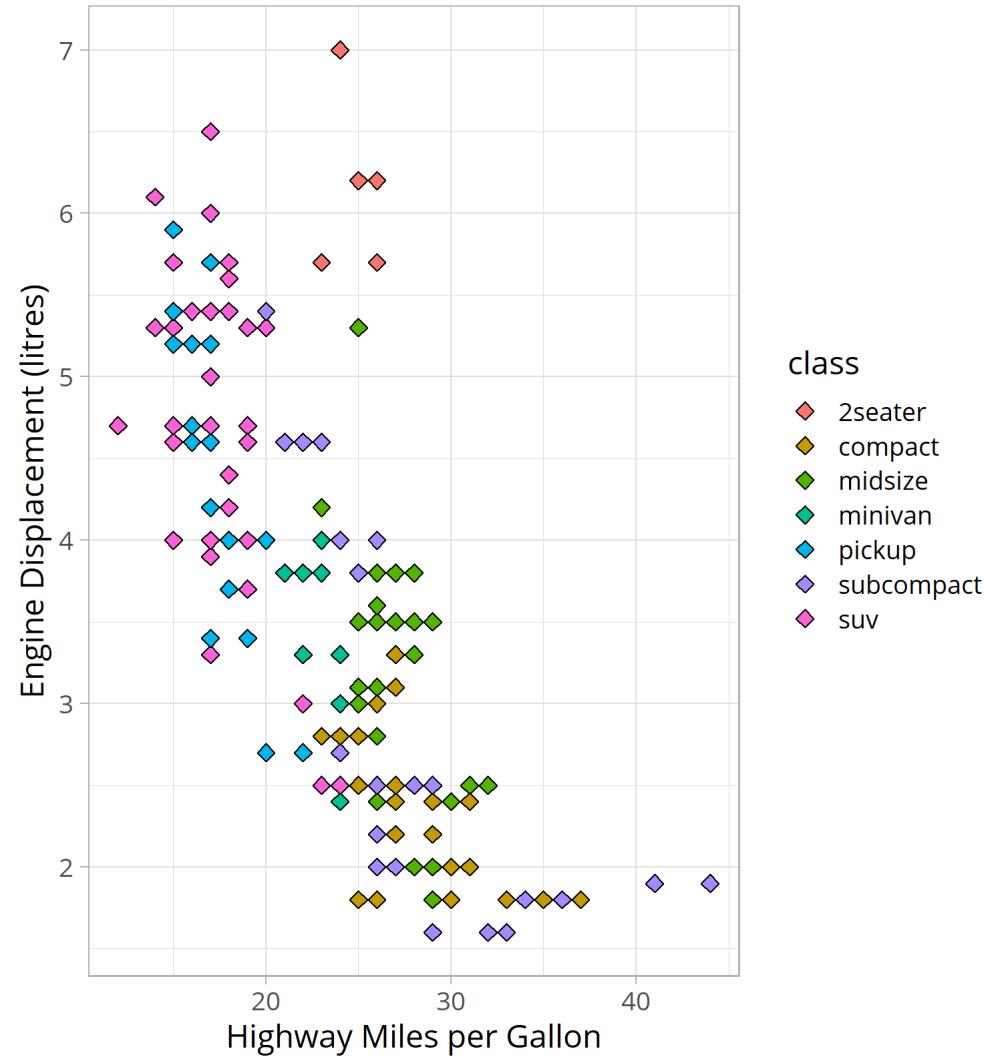
Your Turn: Scatter Plot with Diamonds

```
ggplot(  
  mpg,  
  aes(  
    hwy,  
    displ,  
    fill = class  
) +  
  geom_point(  
    shape = 23,  
    size = 3  
)
```

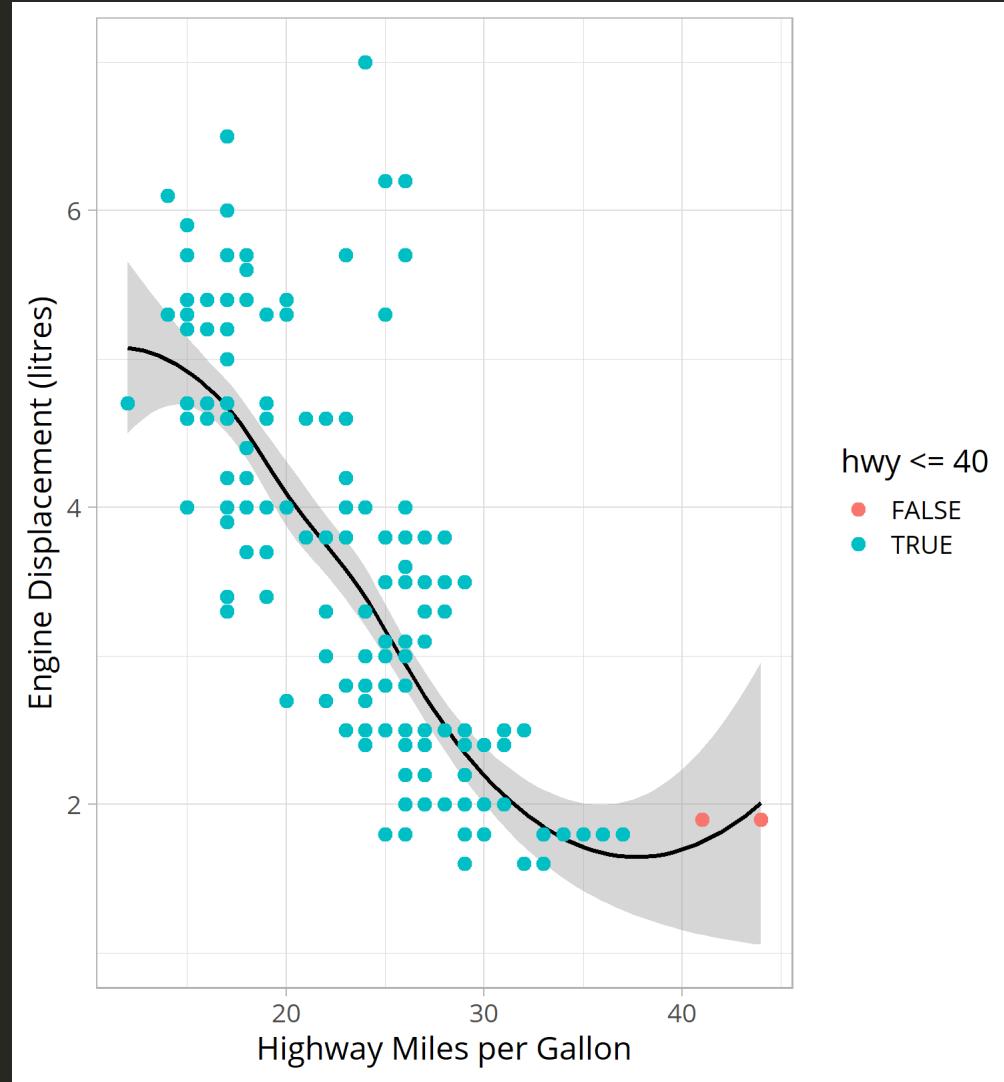


Your Turn: Scatter Plot with Diamonds

```
ggplot(  
  mpg,  
  aes(  
    hwy,  
    displ,  
    fill = class  
) +  
  geom_point(  
    shape = 23,  
    size = 3  
) +  
  labs(  
    x = "Highway Miles per Gallon",  
    y = "Engine Displacement (litres)"  
)
```

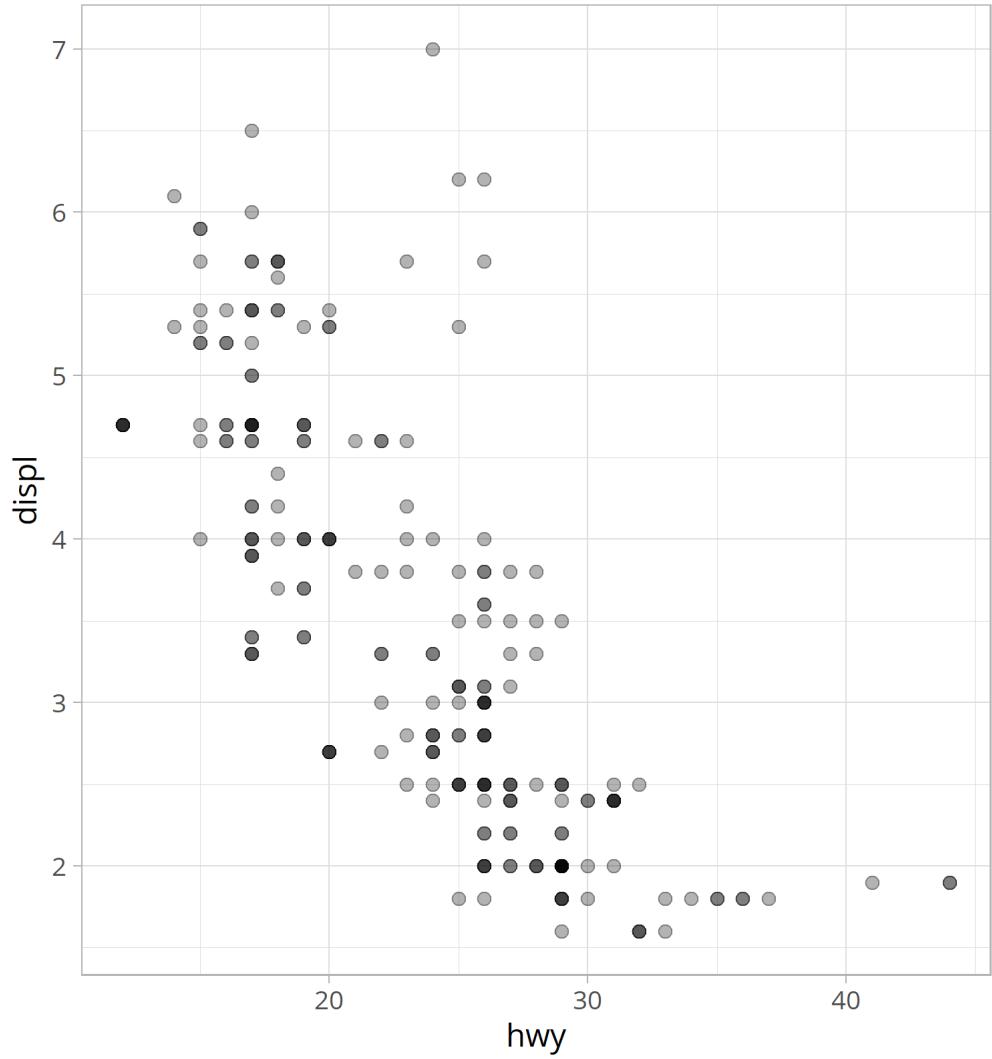


Your Turn: Create this Scatter Plot!



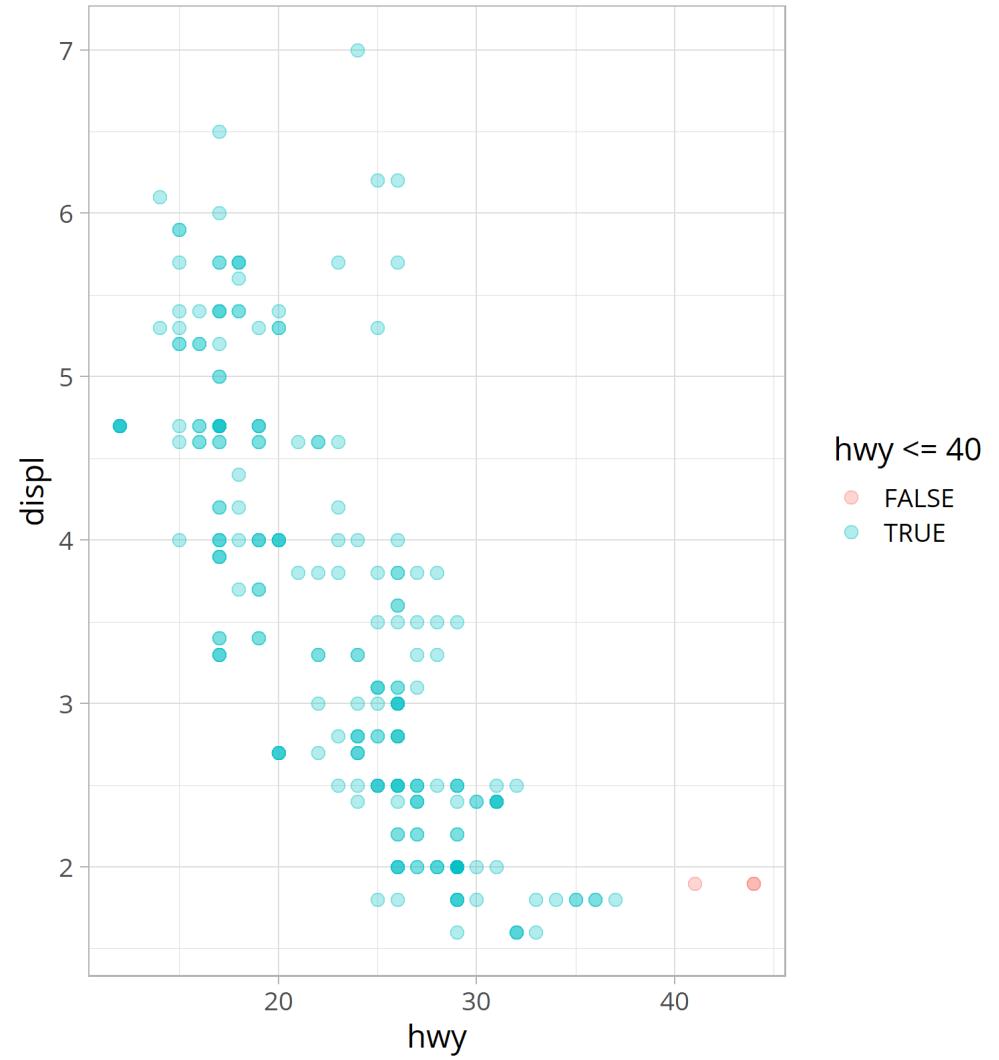
Your Turn: Scatter Plot with Highlights

```
ggplot(mpg, aes(hwy, displ)) +  
  geom_point(  
    size = 3,  
    alpha = .3  
)
```



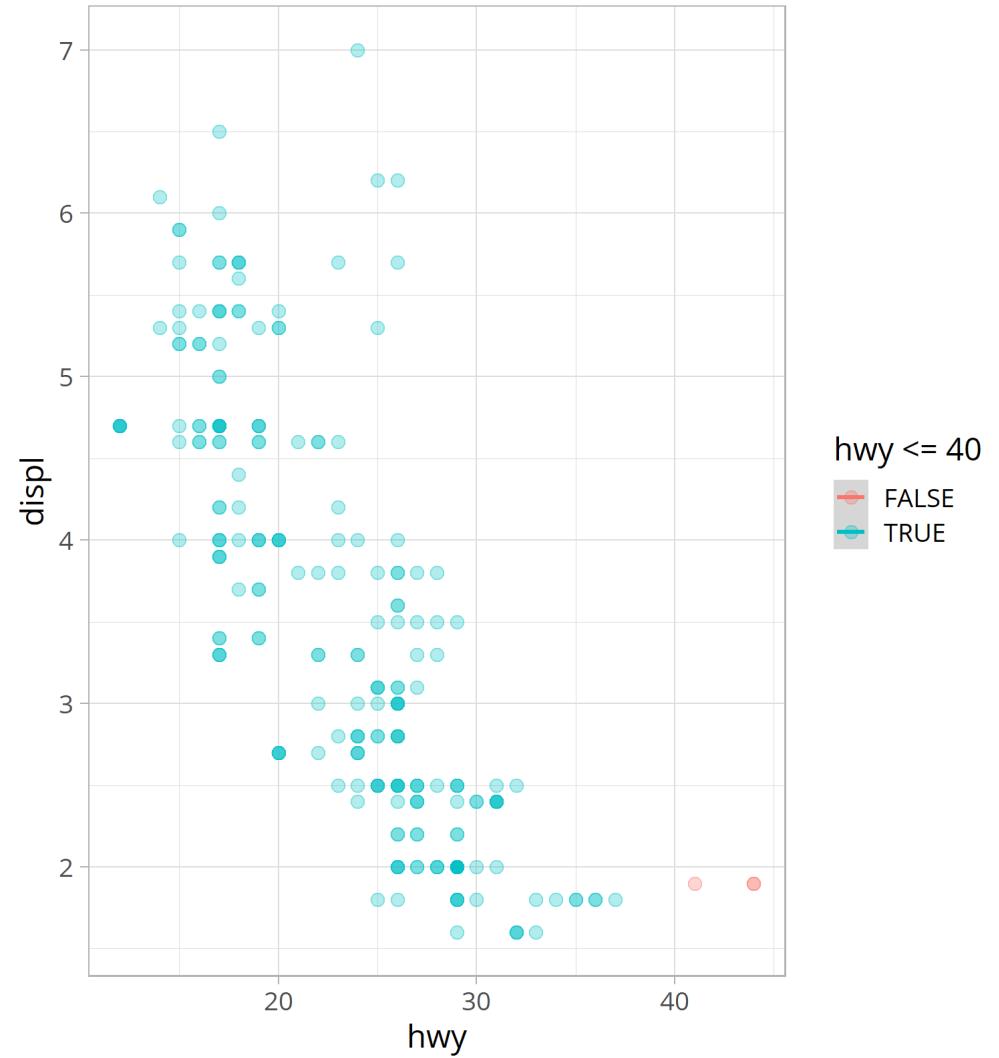
Your Turn: Scatter Plot with Highlights

```
ggplot(  
  mpg,  
  aes(  
    hwy,  
    displ,  
    color = hwy <= 40  
  )) +  
  geom_point(  
    size = 3,  
    alpha = .3  
)
```



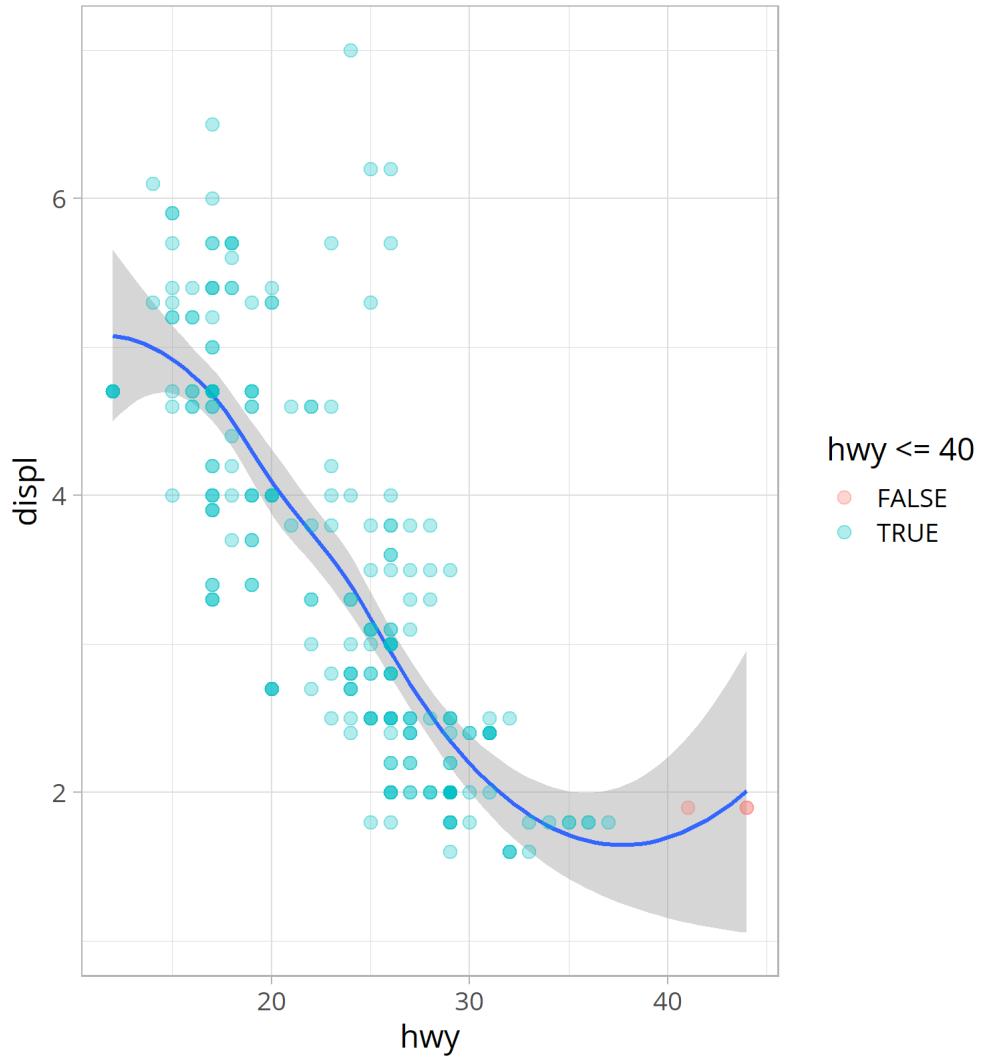
Your Turn: Scatter Plot with Highlights

```
ggplot(  
  mpg,  
  aes(  
    hwy,  
    displ,  
    color = hwy <= 40  
) +  
  stat_smooth() +  
  geom_point(  
    size = 3,  
    alpha = .3  
)
```



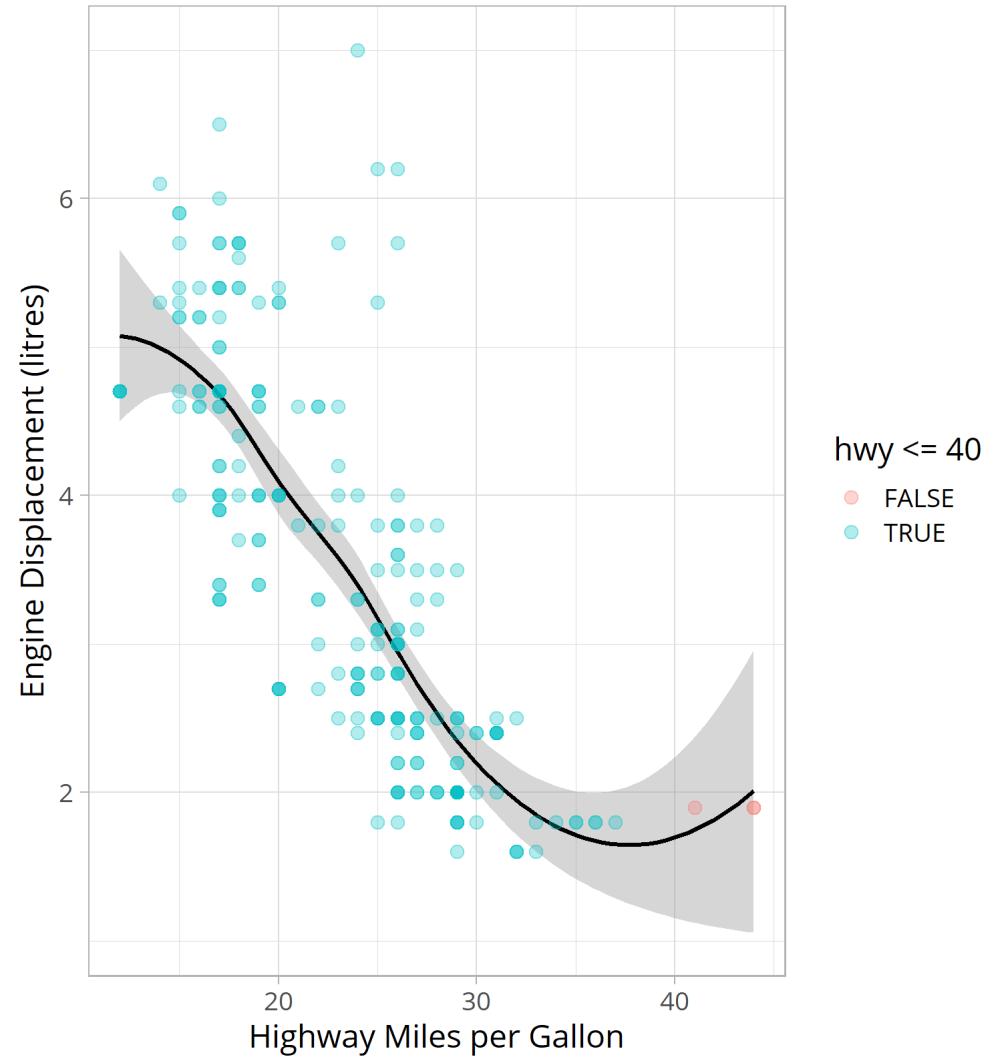
Your Turn: Scatter Plot with Highlights

```
ggplot(  
  mpg,  
  aes(  
    hwy,  
    displ  
  )) +  
  stat_smooth() +  
  geom_point(  
    aes(color = hwy <= 40),  
    size = 3,  
    alpha = .3  
)
```



Your Turn: Scatter Plot with Highlights

```
ggplot(  
  mpg,  
  aes(  
    hwy,  
    displ  
  )) +  
  stat_smooth(  
    color = "black"  
  ) +  
  geom_point(  
    aes(color = hwy <= 40),  
    size = 3,  
    alpha = .3  
  ) +  
  labs(  
    x = "Highway Miles per Gallon",  
    y = "Engine Displacement (litres)"  
  )
```

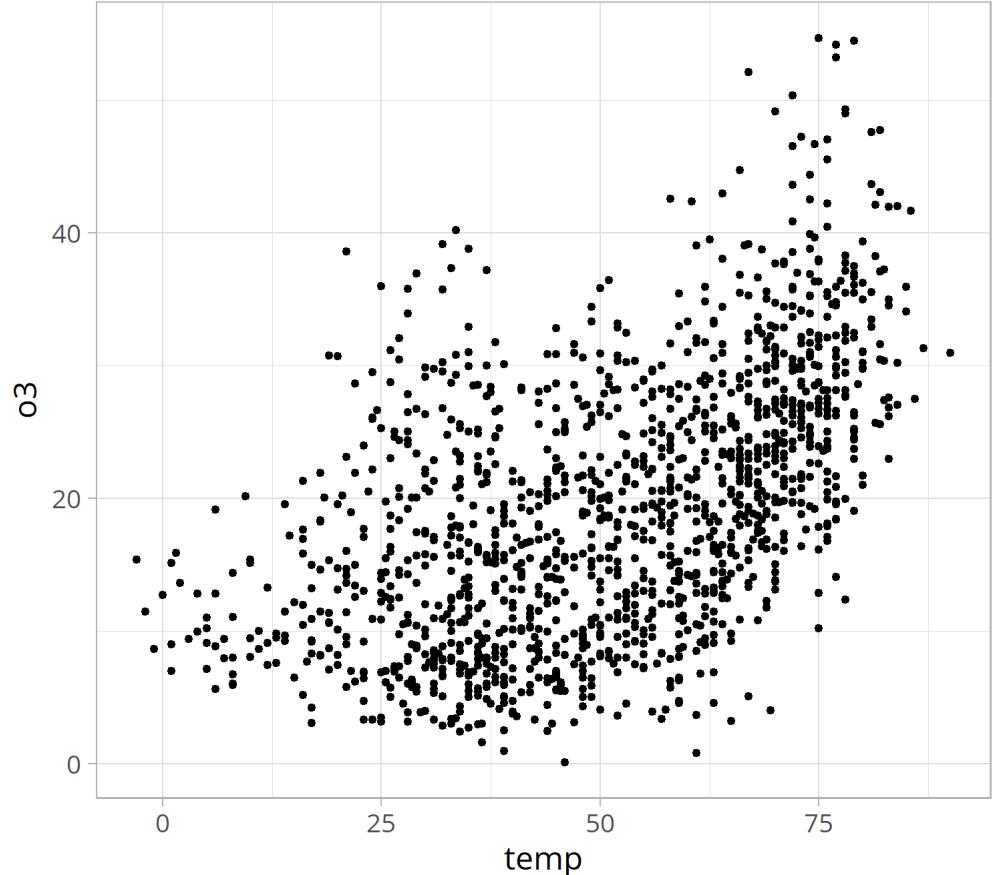


More geom's!

geom_*() and **stat_***()

You can display your data via **geom_point()** as scatter plot...

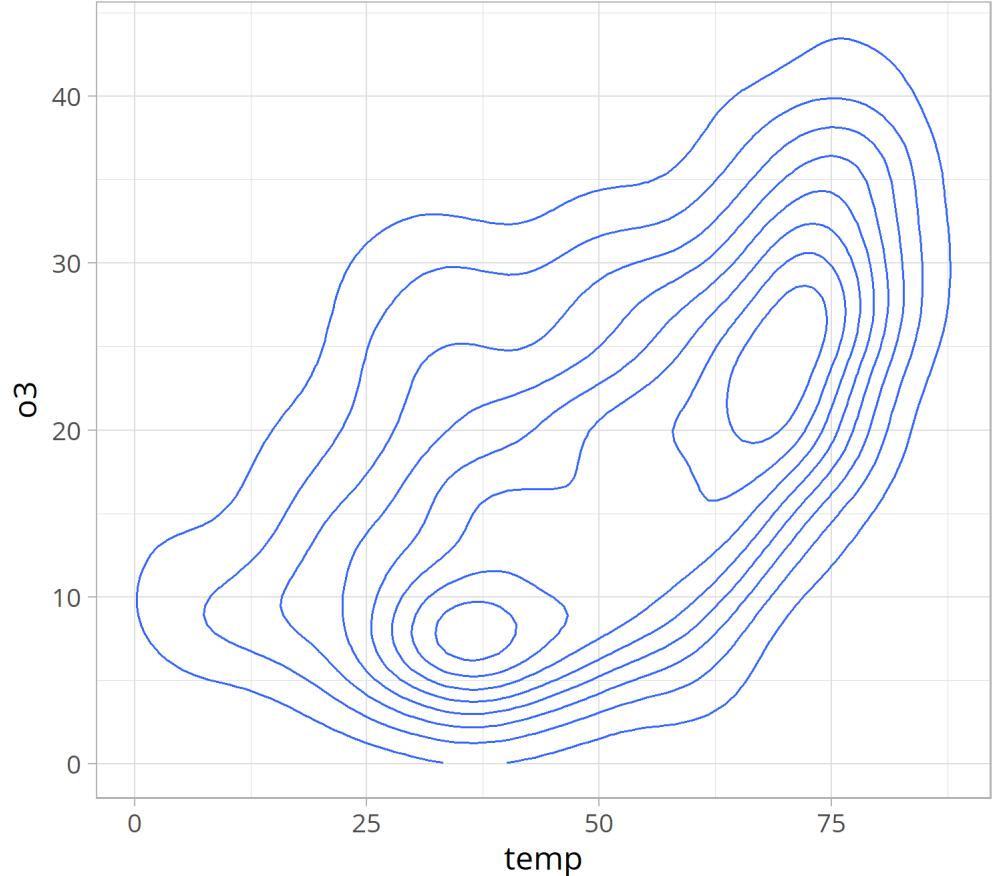
```
ggplot(chic, aes(temp, o3)) +  
  geom_point()
```



geom_*() and **stat_***()

You can display your data via **geom_point()** as scatter plot but there are also other built-in options:

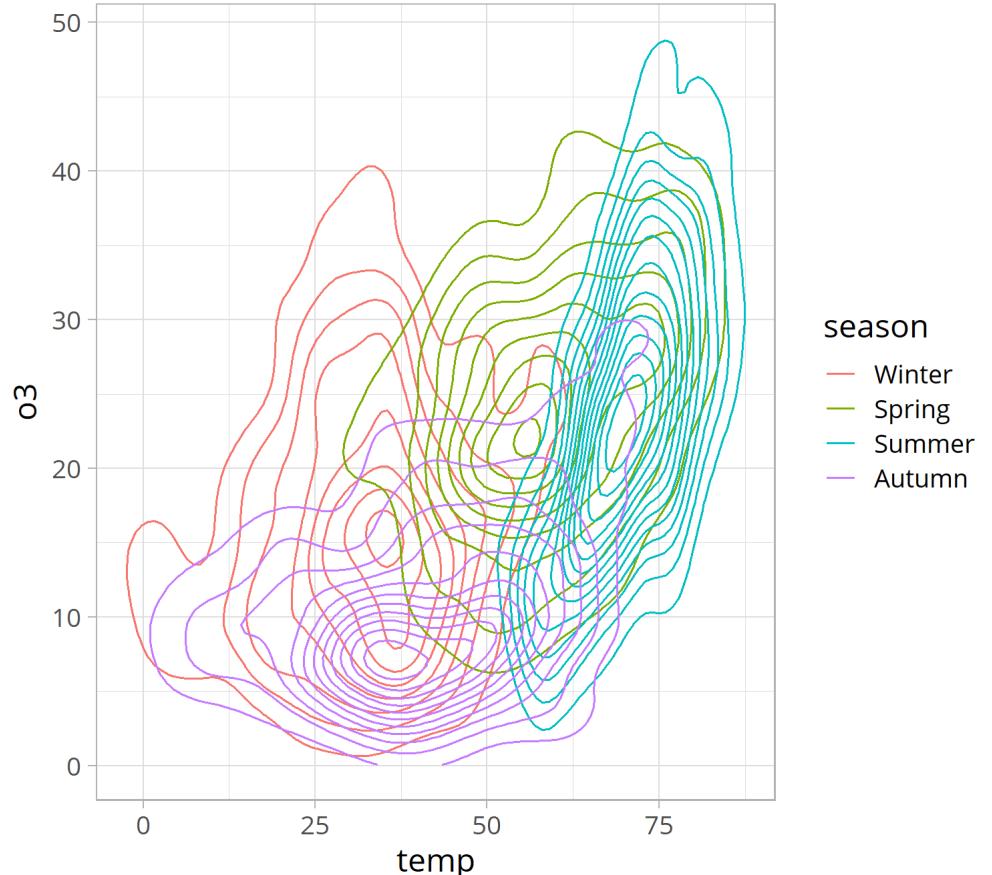
```
ggplot(chic, aes(temp, o3)) +  
  geom_density2d()
```



geom_*() and **stat_***()

You can display your data via **geom_point()** as scatter plot but there are also other built-in options:

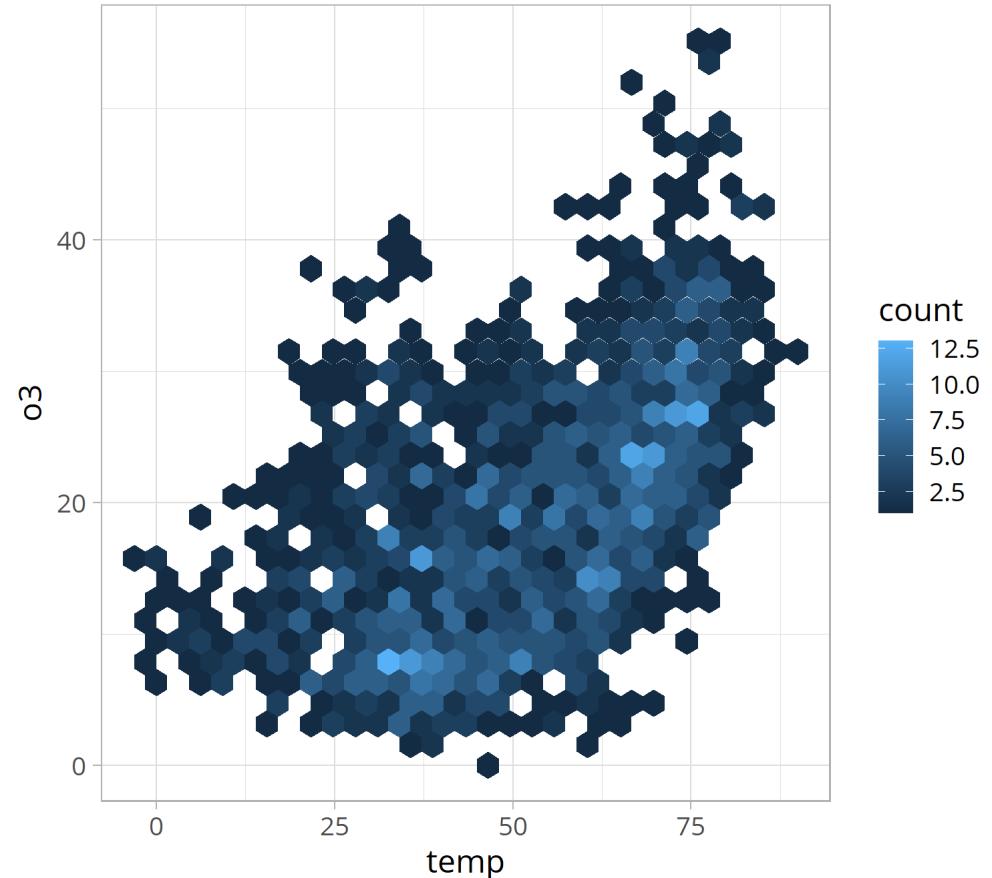
```
ggplot(chic, aes(temp, o3)) +  
  geom_density2d(  
    aes(color = season)  
)
```



geom_*() and **stat_***()

You can display your data via **geom_point()** as scatter plot but there are also other built-in options:

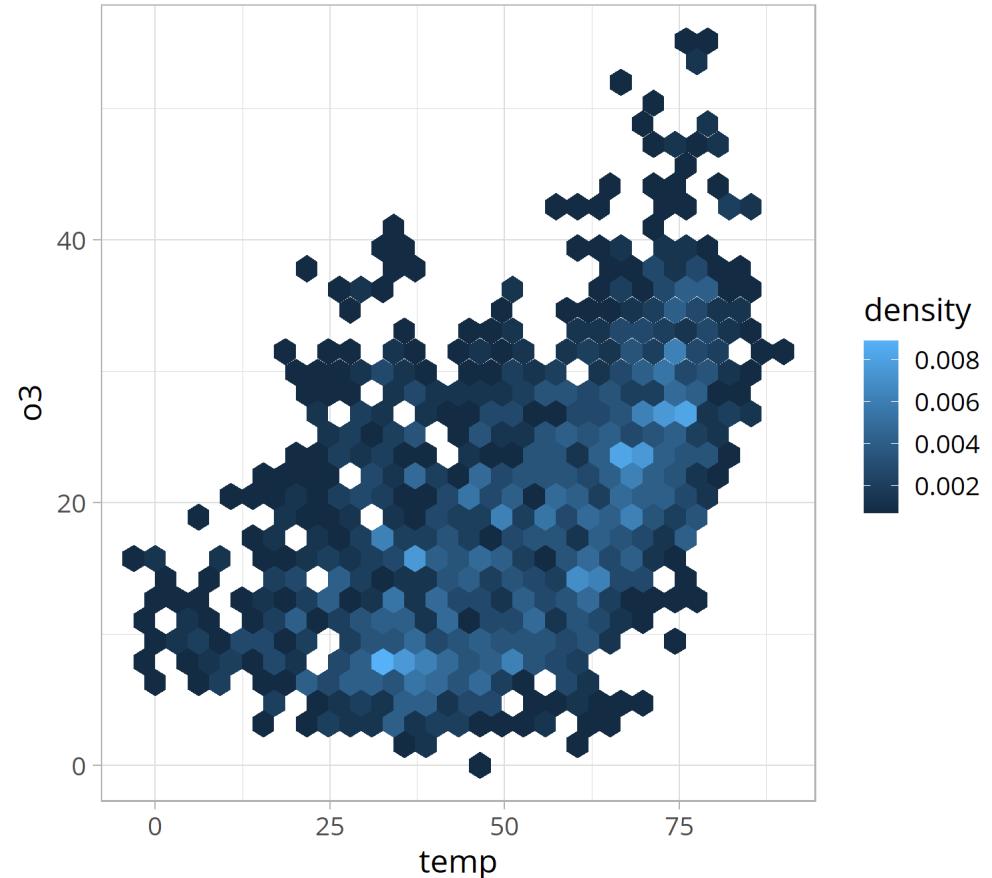
```
ggplot(chic, aes(temp, o3)) +  
  geom_hex()
```



geom_*() and **stat_***()

You can display your data via **geom_point()** as scatter plot but there are also other built-in options:

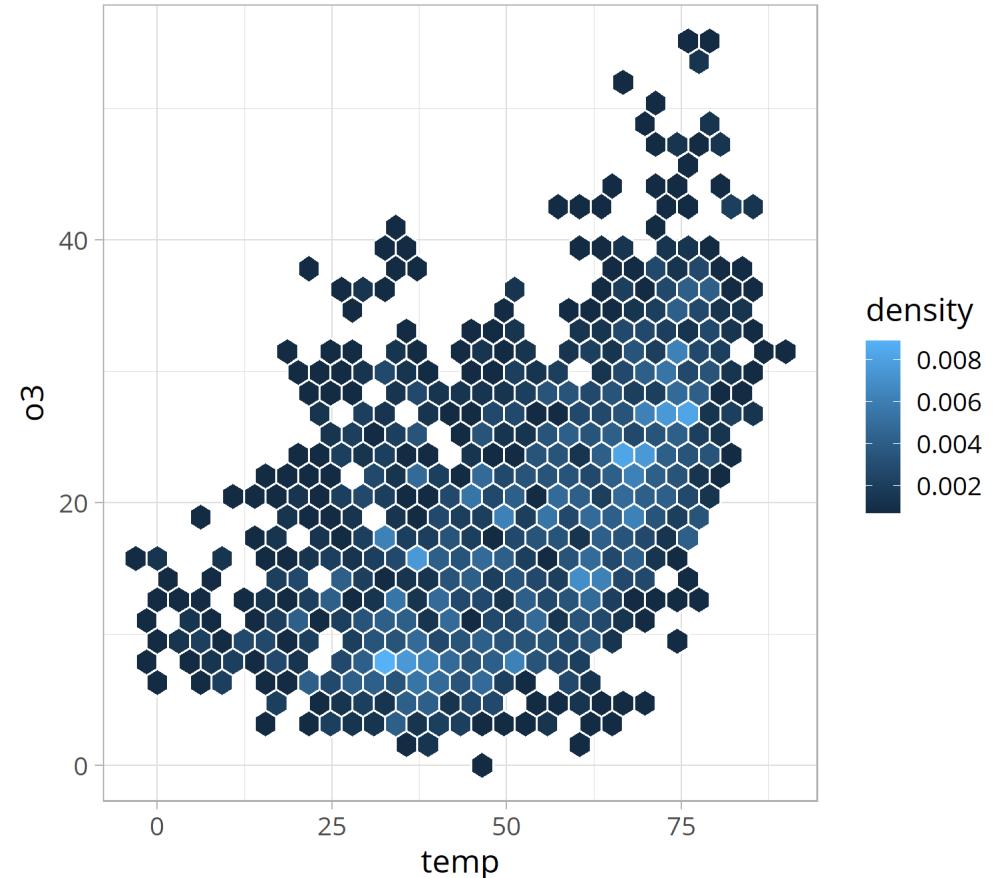
```
ggplot(chic, aes(temp, o3)) +  
  geom_hex(  
    aes(fill = ..density..)  
)
```



geom_*() and **stat_***()

You can display your data via **geom_point()** as scatter plot but there are also other built-in options:

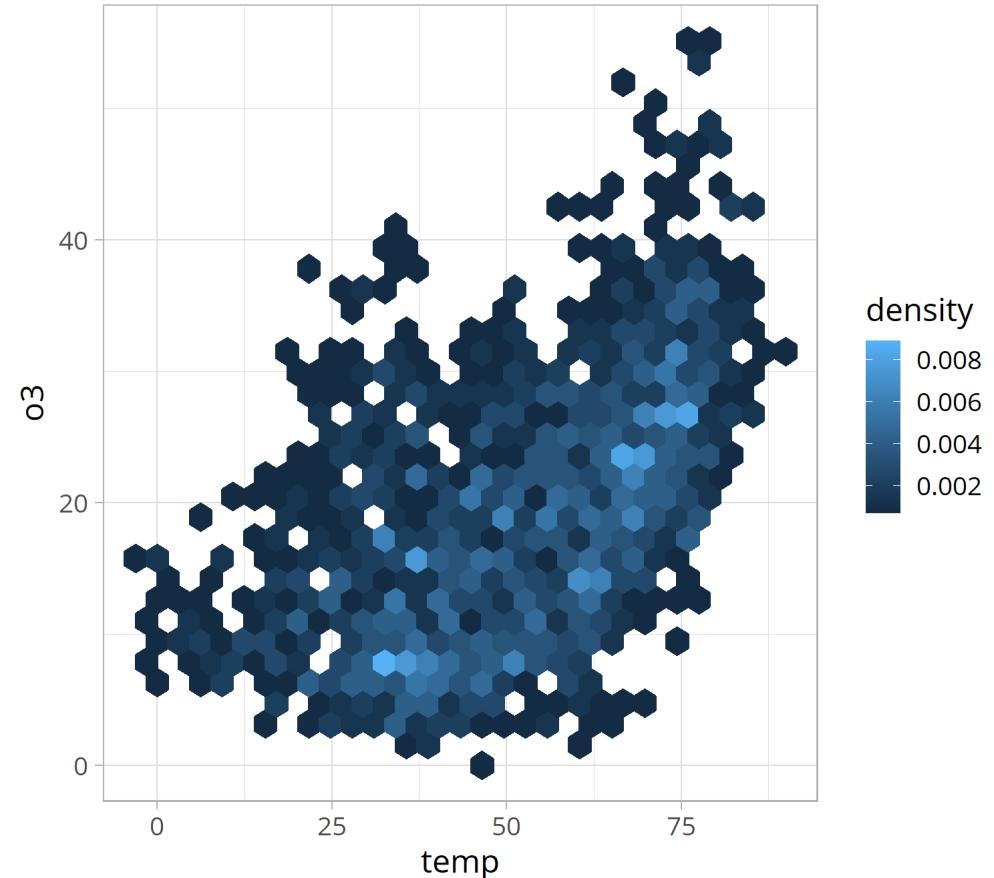
```
ggplot(chic, aes(temp, o3)) +  
  geom_hex(  
    aes(  
      fill = ..density..  
    ),  
    color = "white"  
  )
```



geom_*() and **stat_***()

You can display your data via **geom_point()** as scatter plot but there are also other built-in options:

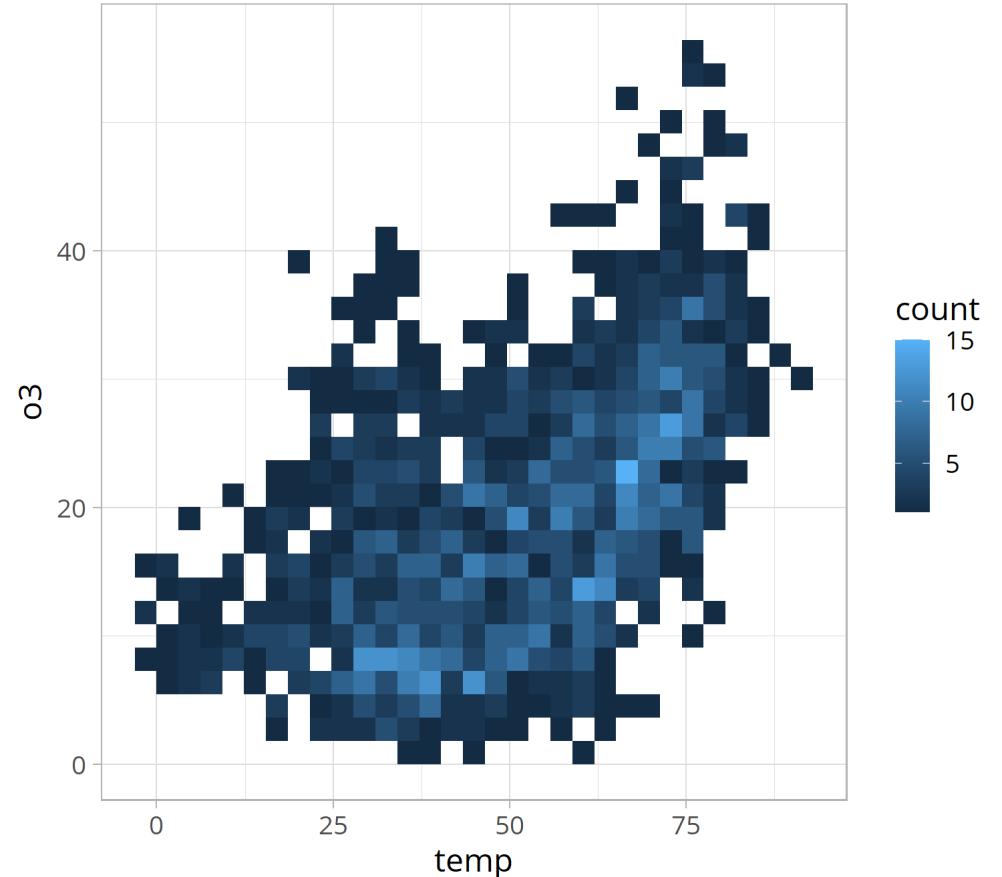
```
ggplot(chic, aes(temp, o3)) +  
  geom_hex(  
    aes(  
      fill = ..density..,  
      color = ..density..  
    )  
  )
```



geom_*() and **stat_***()

You can display your data via **geom_point()** as scatter plot but there are also other built-in options:

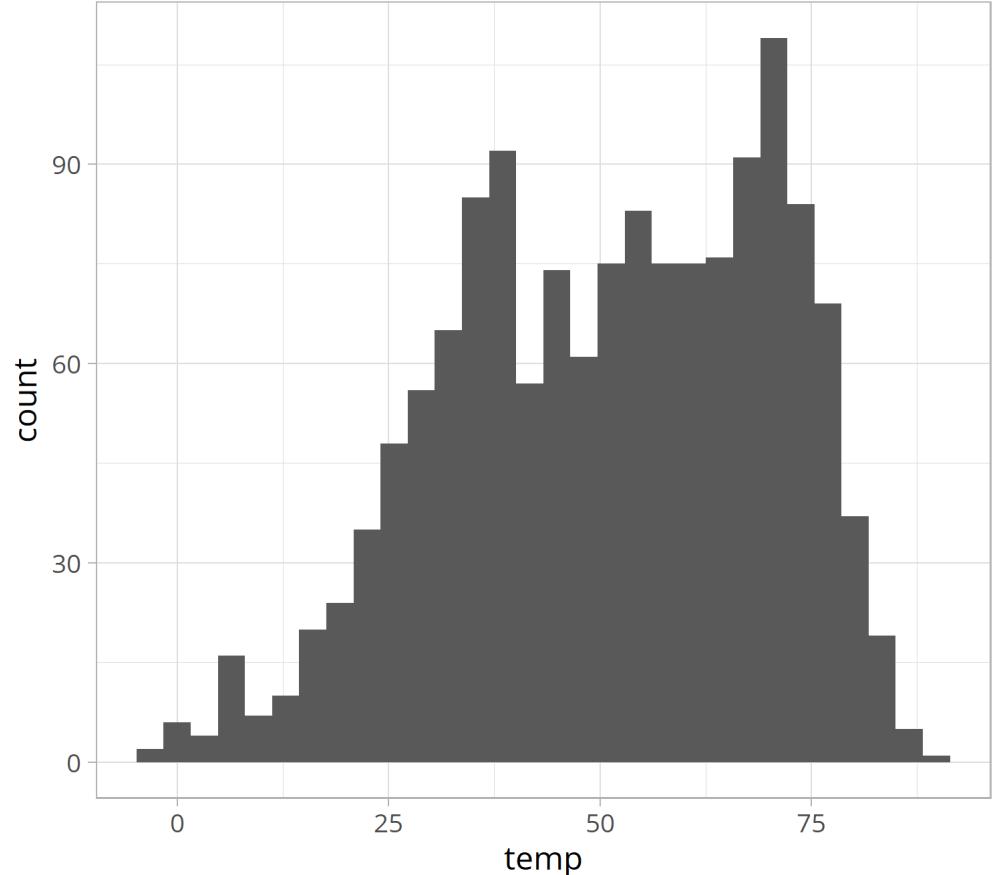
```
ggplot(chic, aes(temp, o3)) +  
  geom_bin2d()
```



geom_*() and **stat_***()

Until now, we have mostly focussed on geoms showing the relation between two variables - but some only need one:

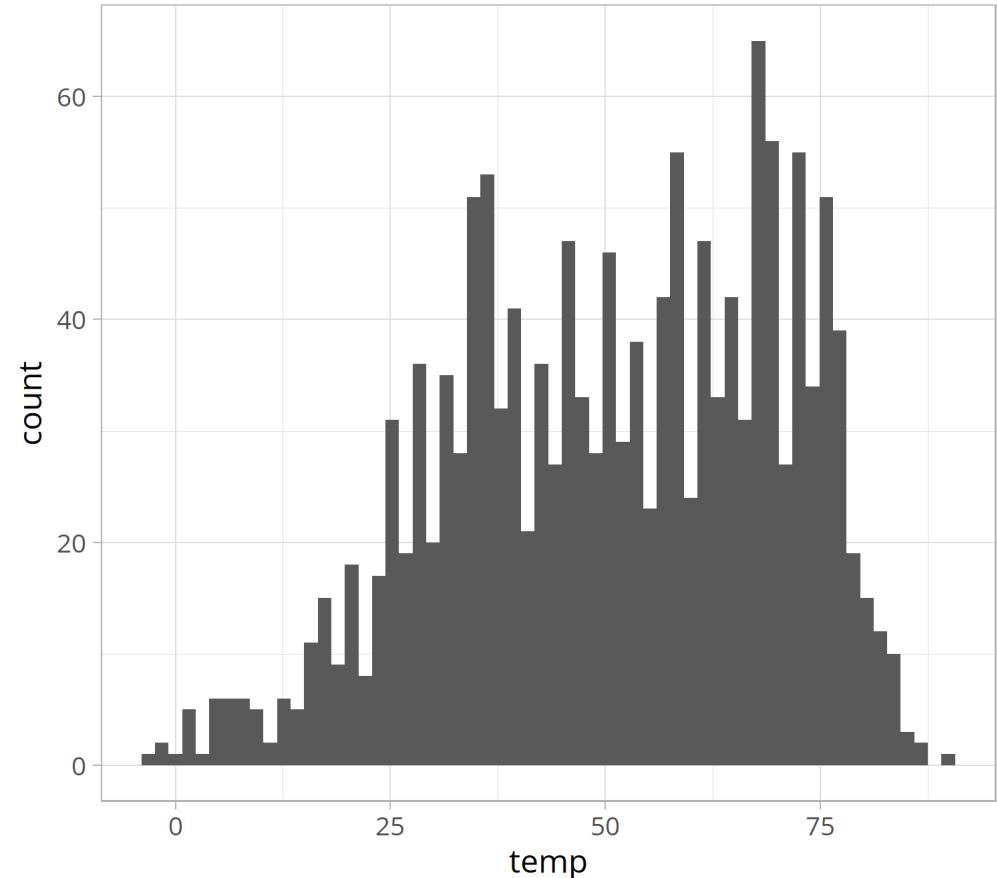
```
ggplot(chic, aes(temp)) +  
  geom_histogram()
```



geom_*() and **stat_***()

Until now, we have mostly focussed on geoms showing the relation between two variables - but some only need one:

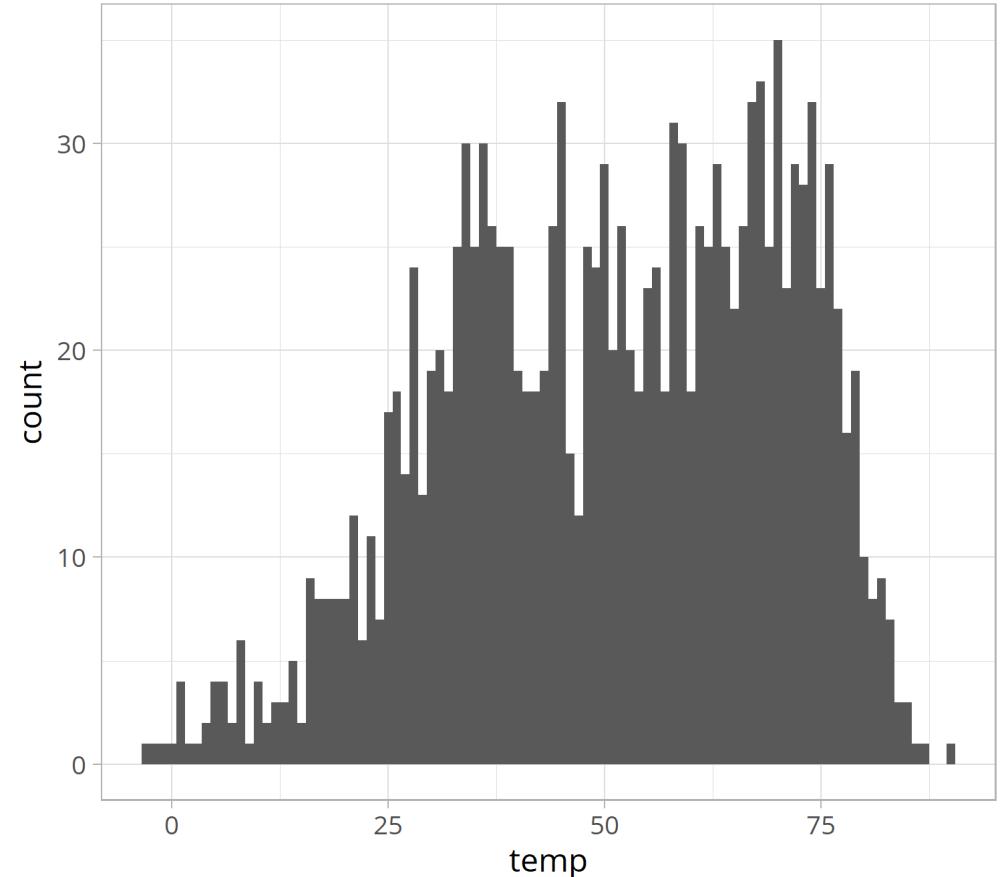
```
ggplot(chic, aes(temp)) +  
  geom_histogram(  
    bins = 60  
)
```



geom_*() and **stat_***()

Until now, we have mostly focussed on geoms showing the relation between two variables - but some only need one:

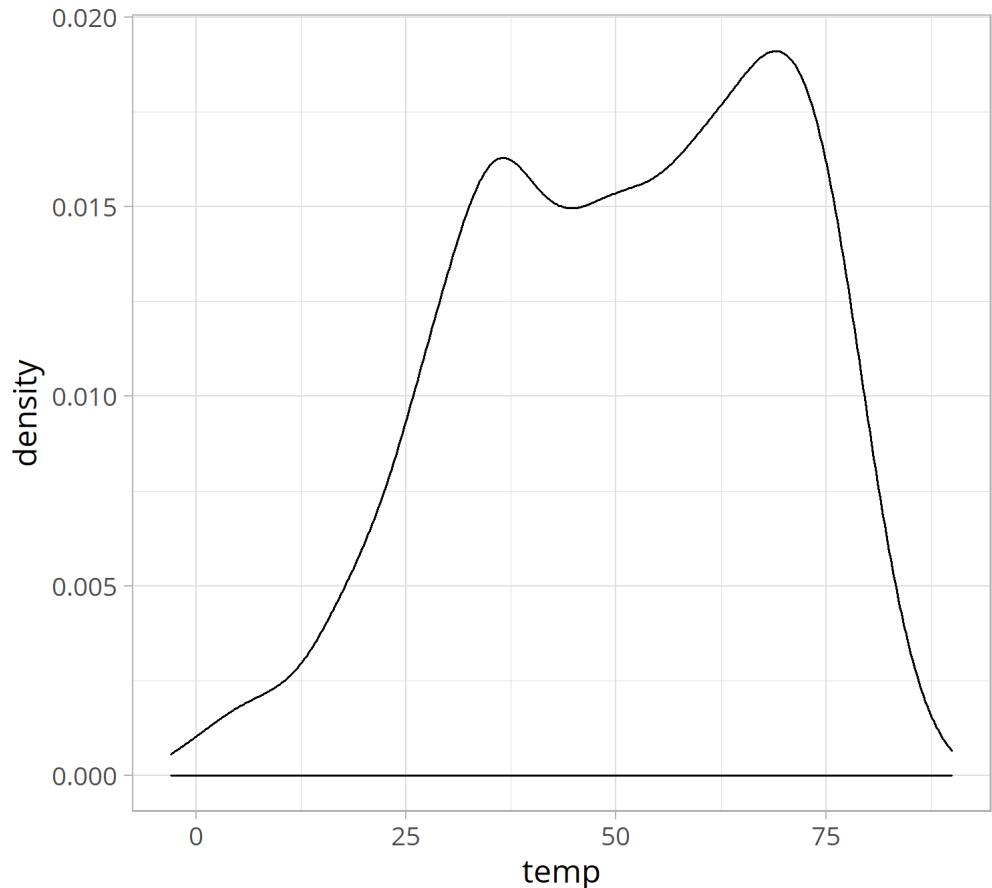
```
ggplot(chic, aes(temp)) +  
  geom_histogram(  
    binwidth = 1  
)
```



geom_*() and **stat_***()

Until now, we have mostly focussed on geoms showing the relation between two variables - but some only need one:

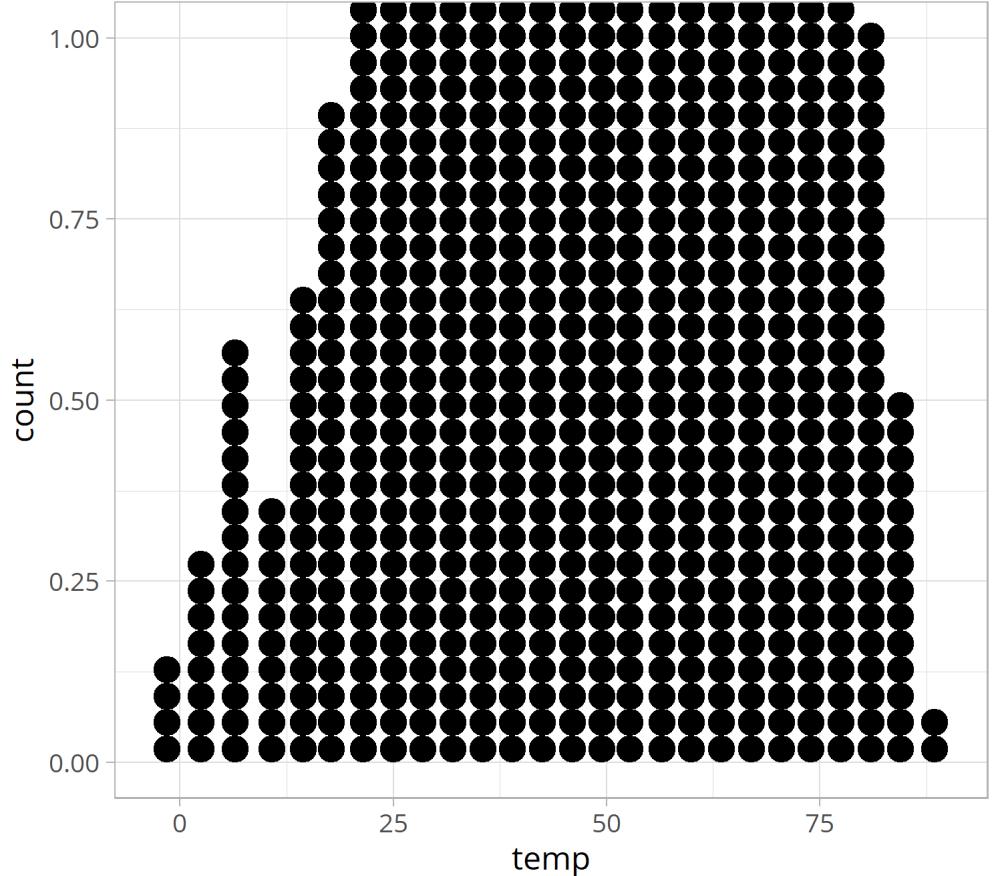
```
ggplot(chic, aes(temp)) +  
  geom_density()
```



geom_*() and **stat_***()

Until now, we have mostly focussed on geoms showing the relation between two variables - but some only need one:

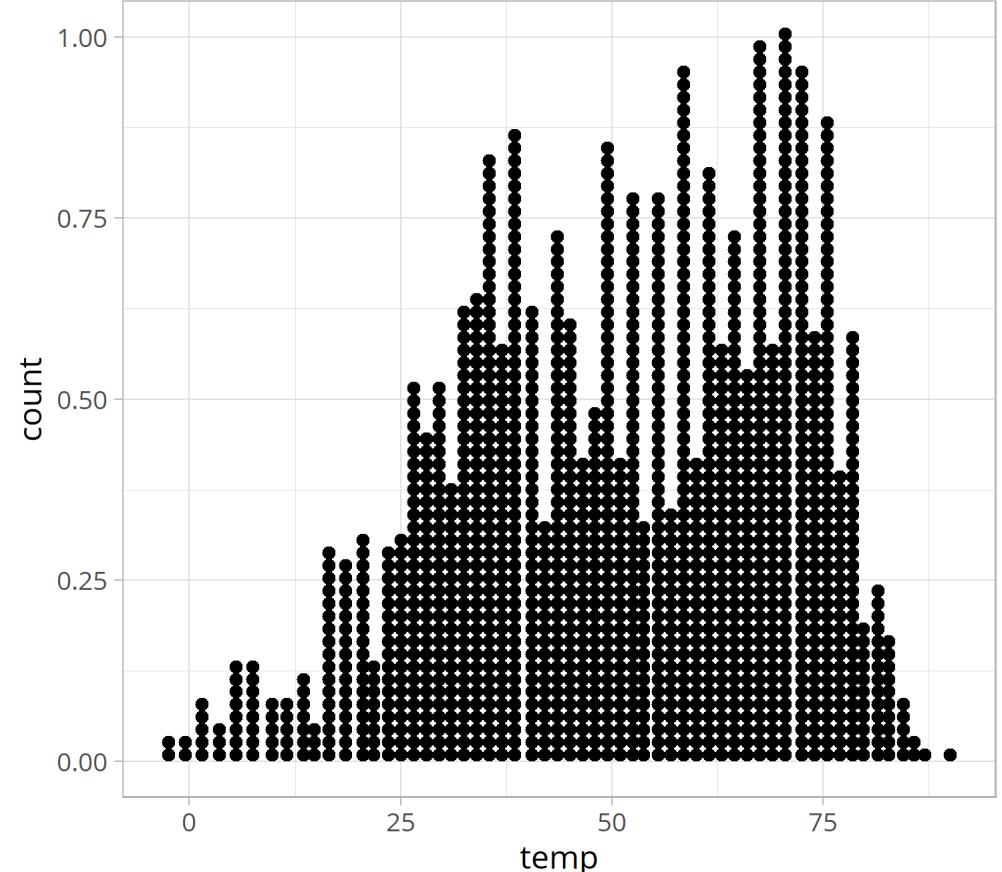
```
ggplot(chic, aes(temp)) +  
  geom_dotplot()
```



geom_*() and **stat_***()

Until now, we have mostly focussed on geoms showing the relation between two variables - but some only need one:

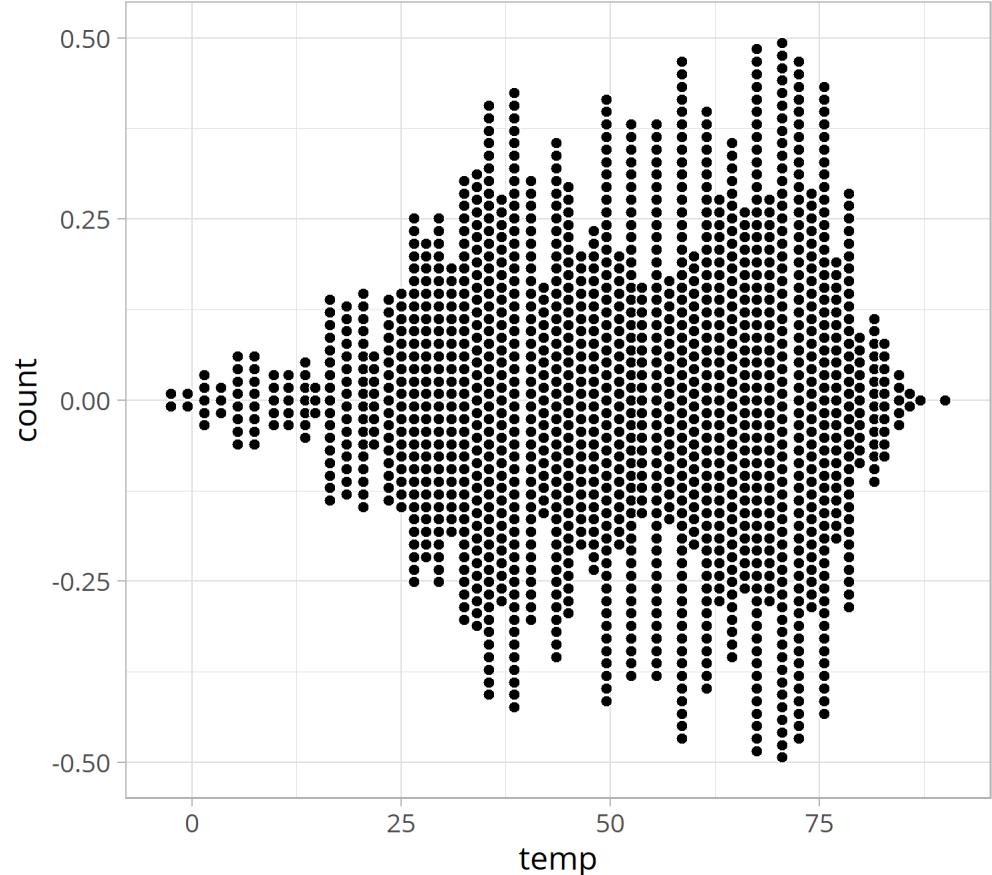
```
ggplot(chic, aes(temp)) +  
  geom_dotplot(  
    binwidth = 1.5  
  )
```



geom_*() and **stat_***()

Until now, we have mostly focussed on geoms showing the relation between two variables - but some only need one:

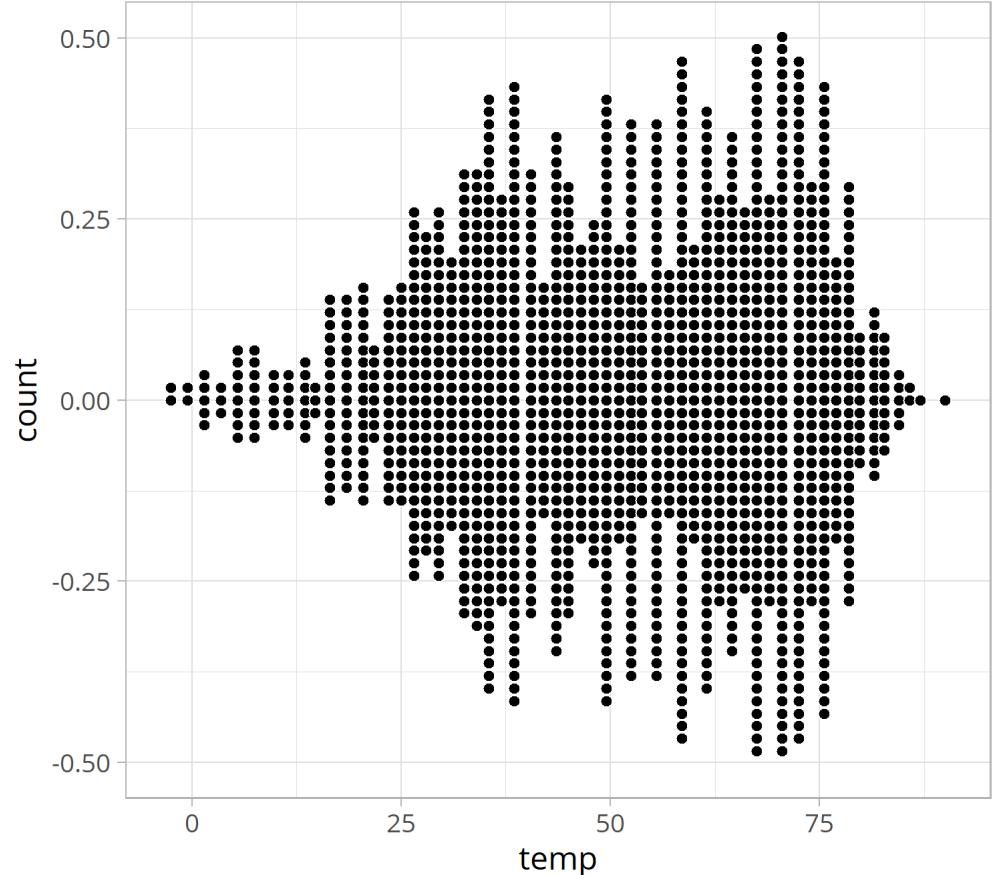
```
ggplot(chic, aes(temp)) +  
  geom_dotplot(  
    binwidth = 1.5,  
    color = "white",  
    stackdir = "center"  
)
```



geom_*() and **stat_***()

Until now, we have mostly focussed on geoms showing the relation between two variables - but some only need one:

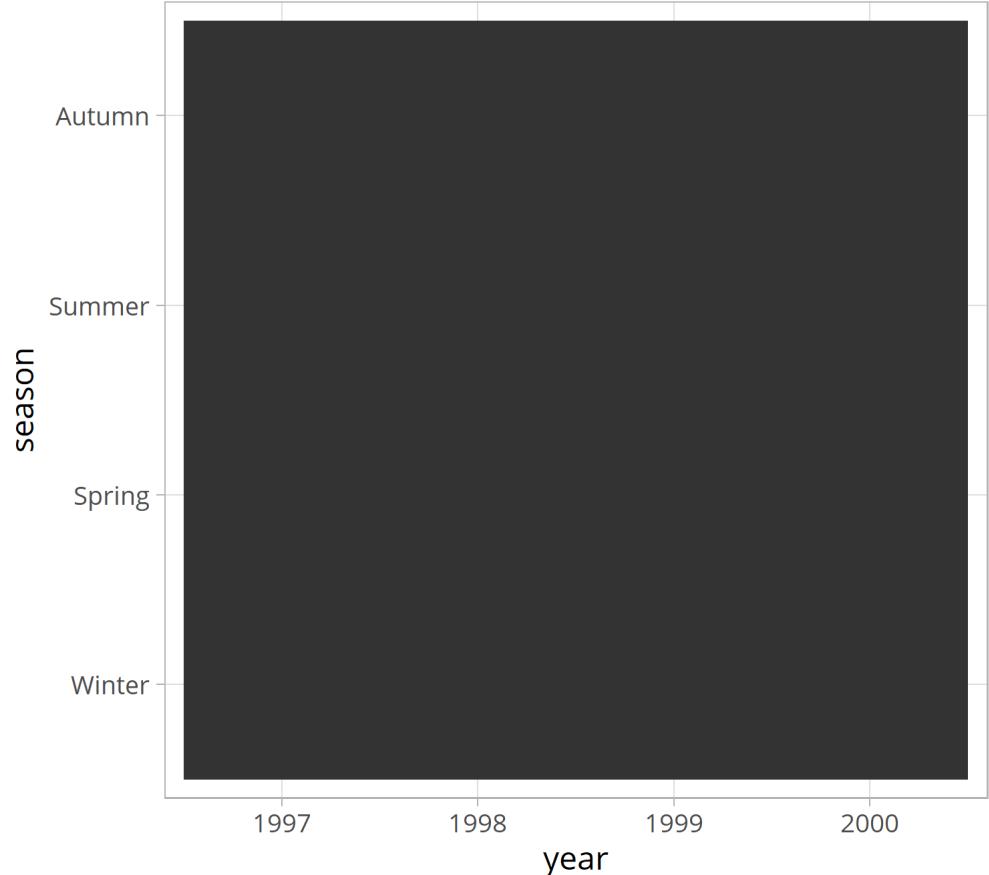
```
ggplot(chic, aes(temp)) +  
  geom_dotplot(  
    binwidth = 1.5,  
    color = "white",  
    stackdir = "centerwhole"  
)
```



geom_*() and **stat_***()

... and some which make only sense with three variables:

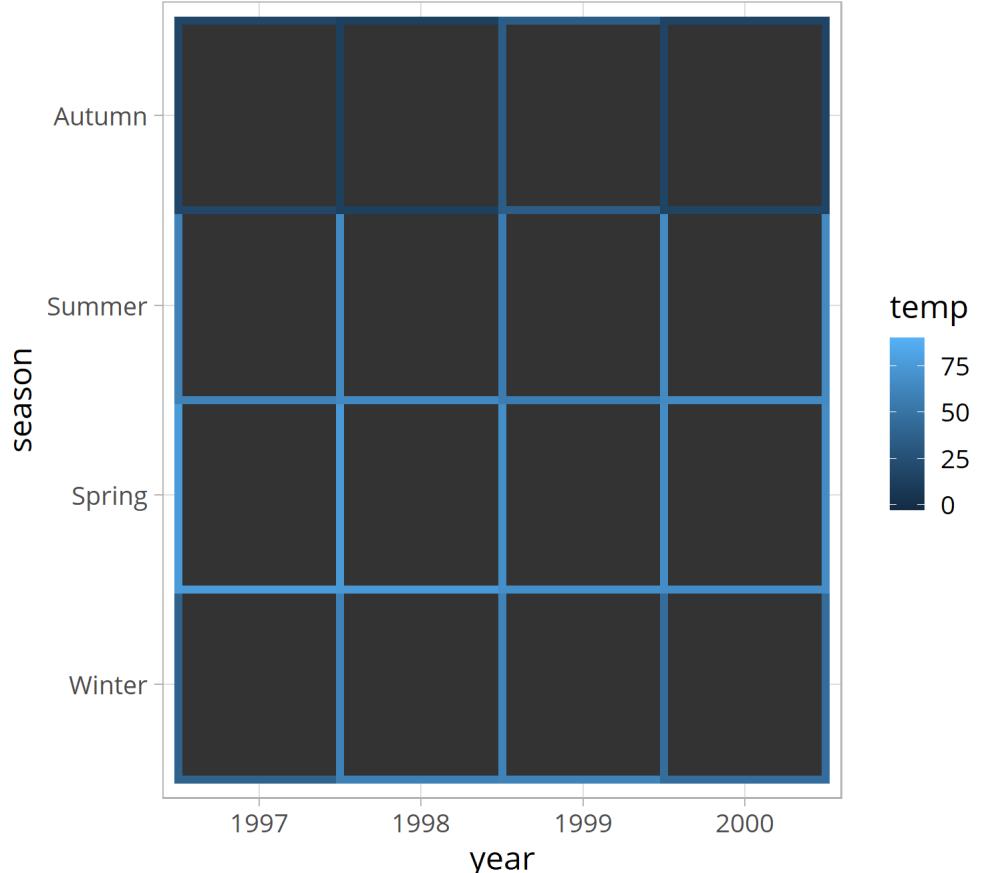
```
ggplot(chic, aes(year, season)) +  
  geom_tile()
```



geom_*() and **stat_***()

... and some which make only sense with three variables:

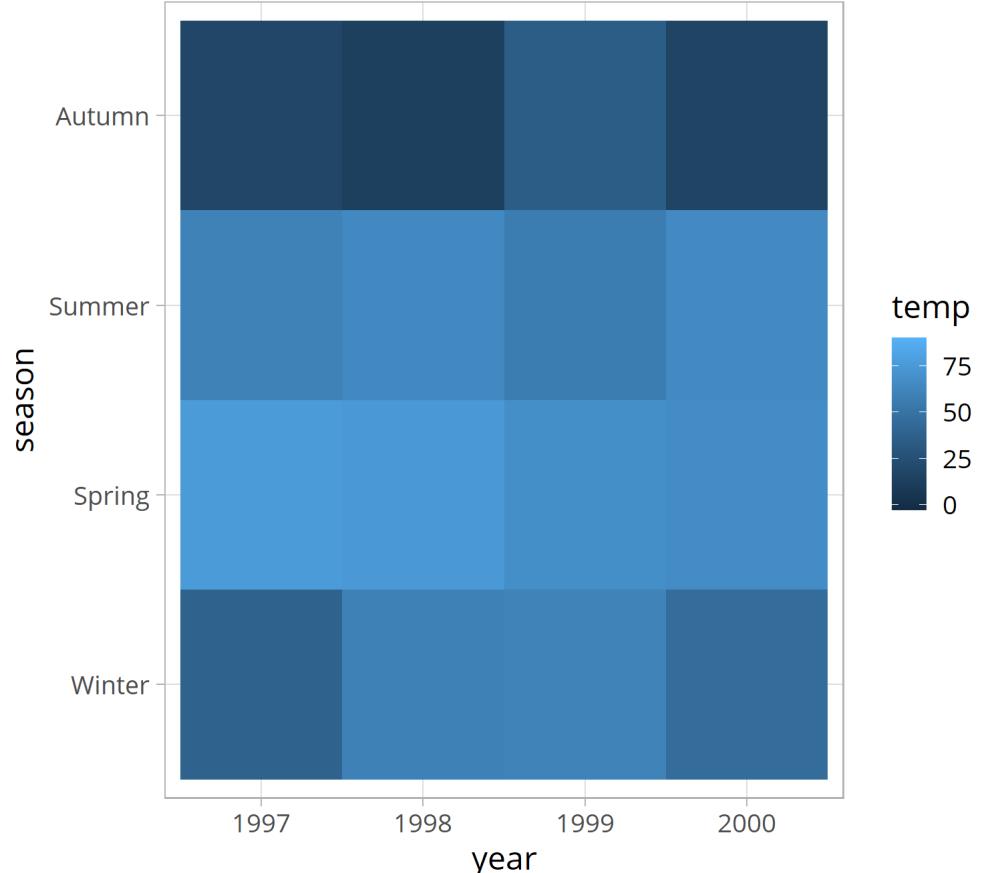
```
ggplot(chic, aes(year, season)) +  
  geom_tile(  
    aes(color = temp),  
    size = 2  
  )
```



geom_*() and **stat_***()

... and some which make only sense with three variables:

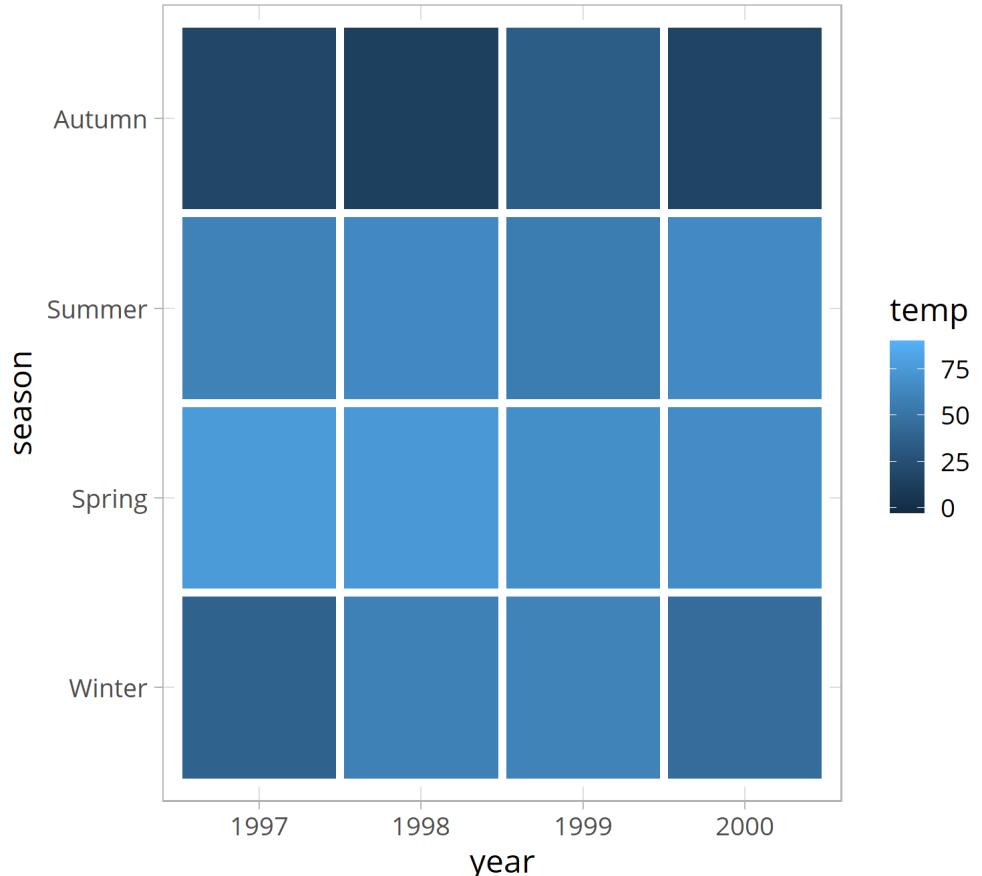
```
ggplot(chic, aes(year, season)) +  
  geom_tile(  
    aes(fill = temp)  
  )
```



geom_*() and **stat_***()

... and some which make only sense with three variables:

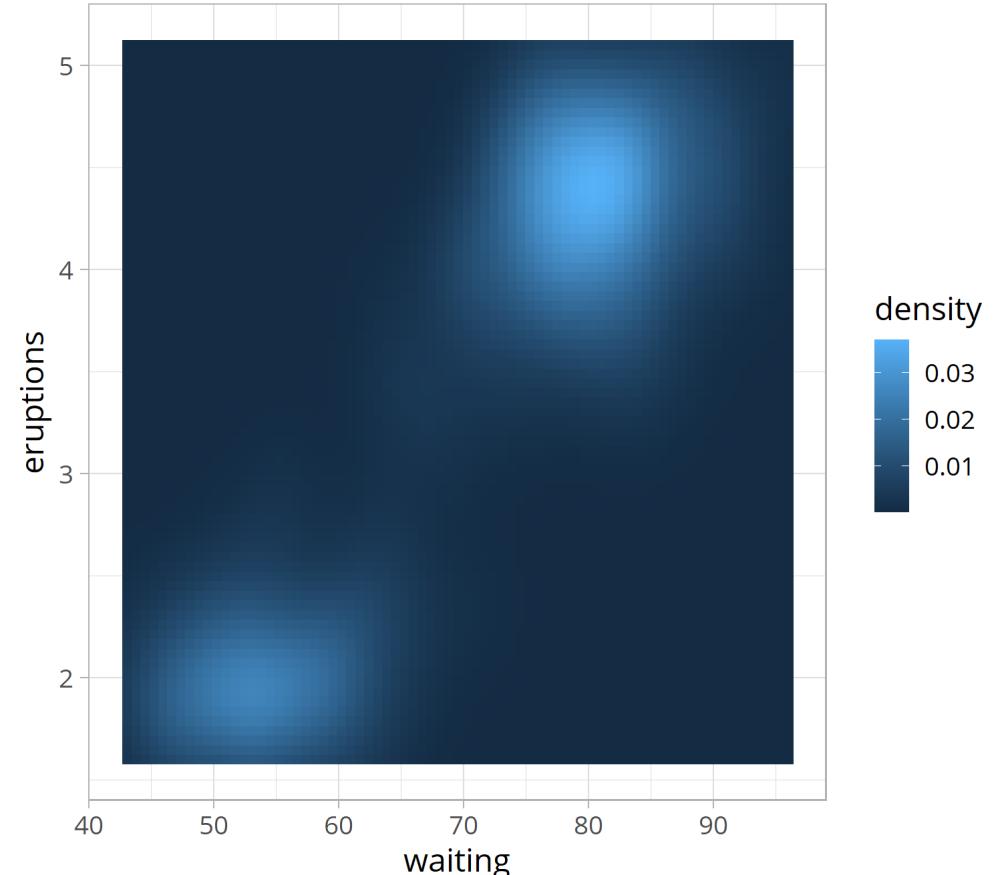
```
ggplot(chic, aes(year, season)) +  
  geom_tile(  
    aes(fill = temp),  
    color = "white",  
    size = 2  
)
```



geom_*() and **stat_***()

... and some which make only sense with three variables:

```
ggplot(  
  faithful,  
  aes(  
    waiting,  
    eruptions  
  )) +  
  geom_raster(  
    aes(fill = density)  
)
```

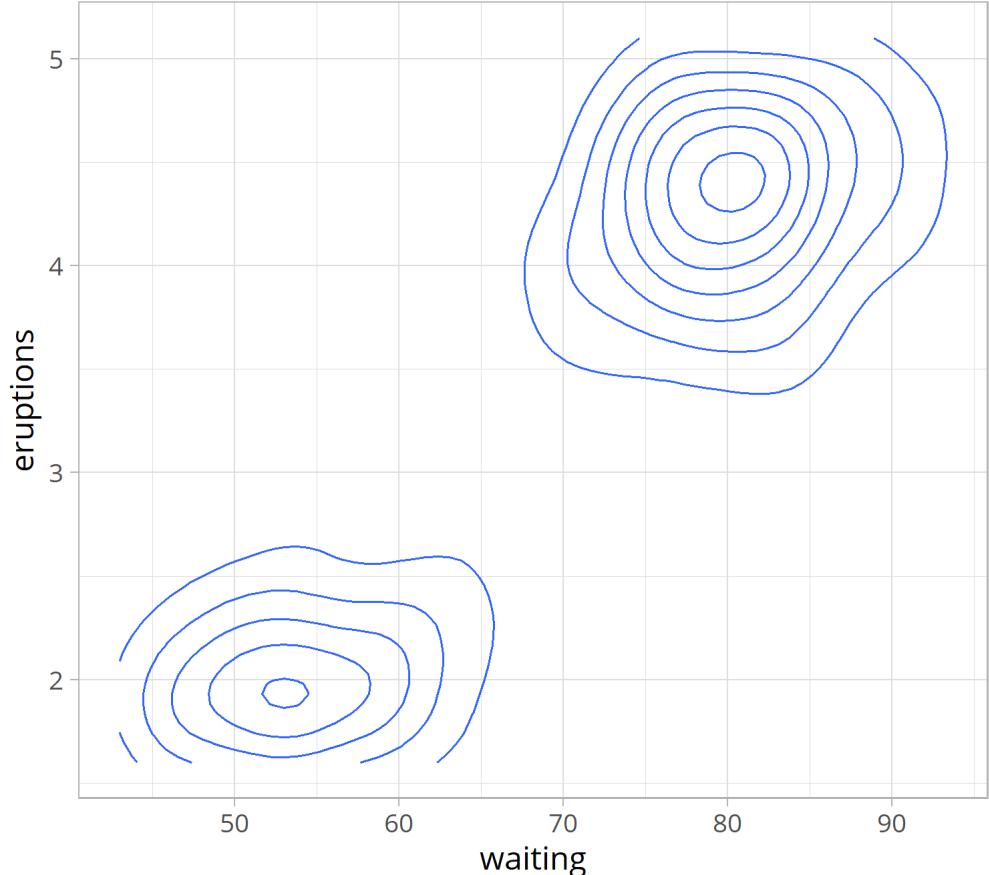


geom_raster() does more or less the same as
geom_tile() but is is a "high performance special case for
when\all the tiles are the same size".

geom_*() and **stat_***()

... and some which make only sense with three variables:

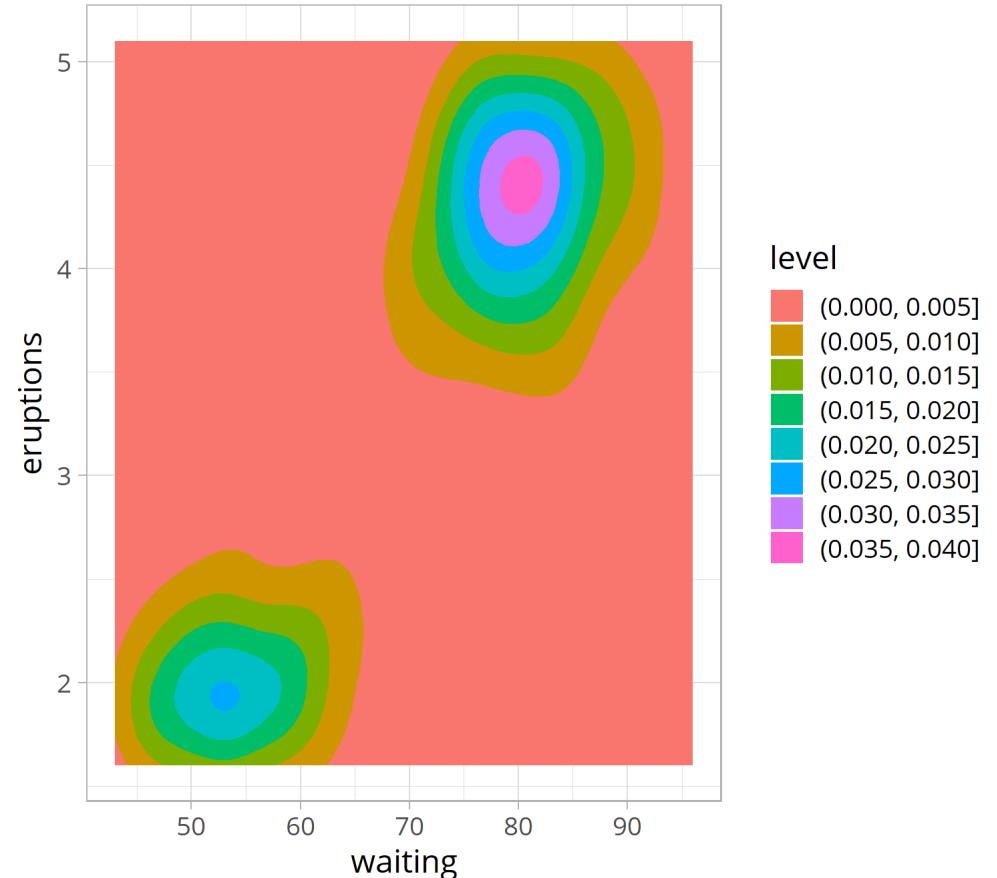
```
ggplot(  
  faithful,  
  aes(  
    waiting,  
    eruptions,  
    z = density  
  )) +  
  geom_contour()
```



geom_*() and **stat_***()

... and some which make only sense with three variables:

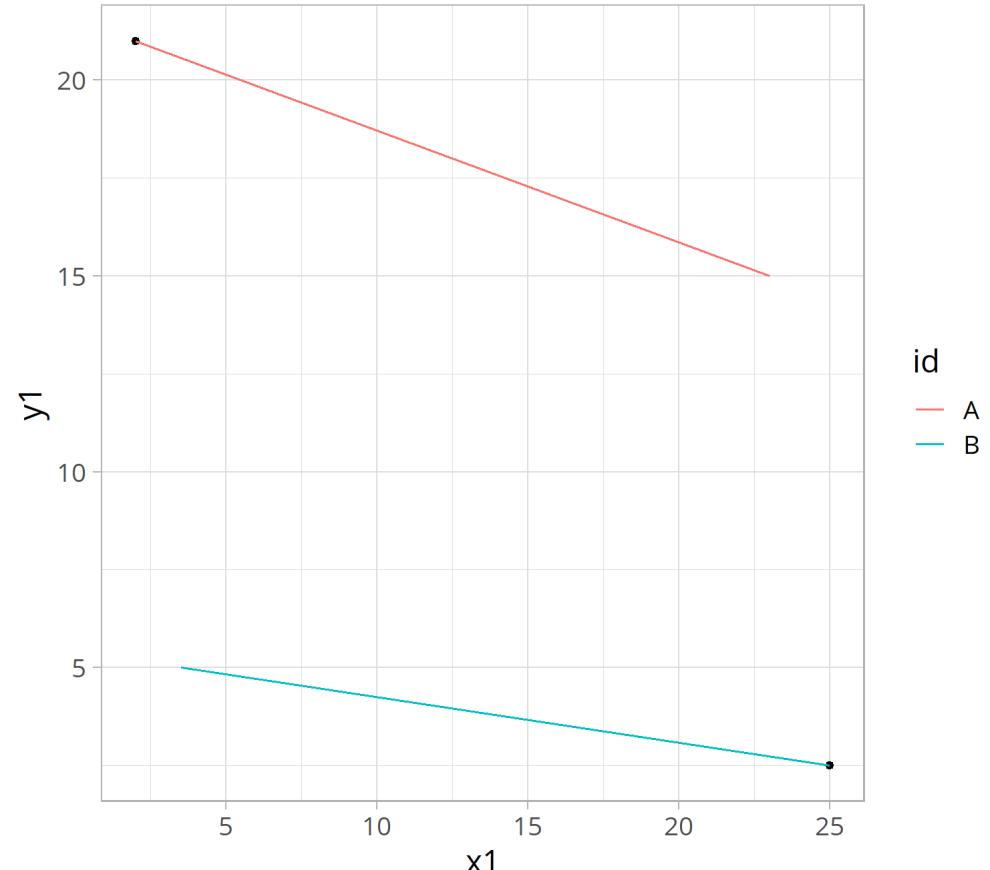
```
ggplot(  
  faithful,  
  aes(  
    waiting,  
    eruptions,  
    z = density  
  )) +  
  geom_contour_filled()
```



geom_*() and **stat_***()

Some need even more inputs:

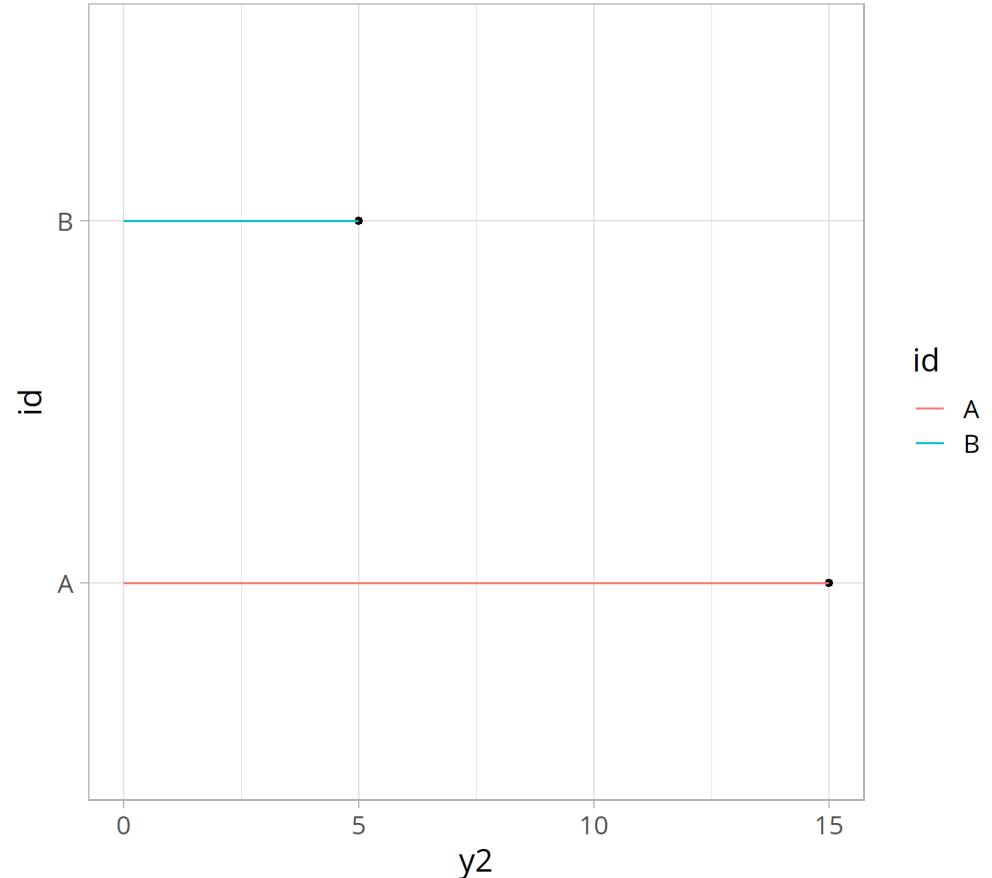
```
points <- tibble(  
  x1 = c(2, 25),  
  x2 = c(23, 3.5),  
  y1 = c(21, 2.5),  
  y2 = c(15, 5),  
  id = c("A", "B"))  
  
ggplot(points, aes(x1, y1)) +  
  geom_point() +  
  geom_segment(  
    aes(  
      xend = x2,  
      yend = y2,  
      color = id  
    ))
```



geom_*() and **stat_***()

Some need even more inputs:

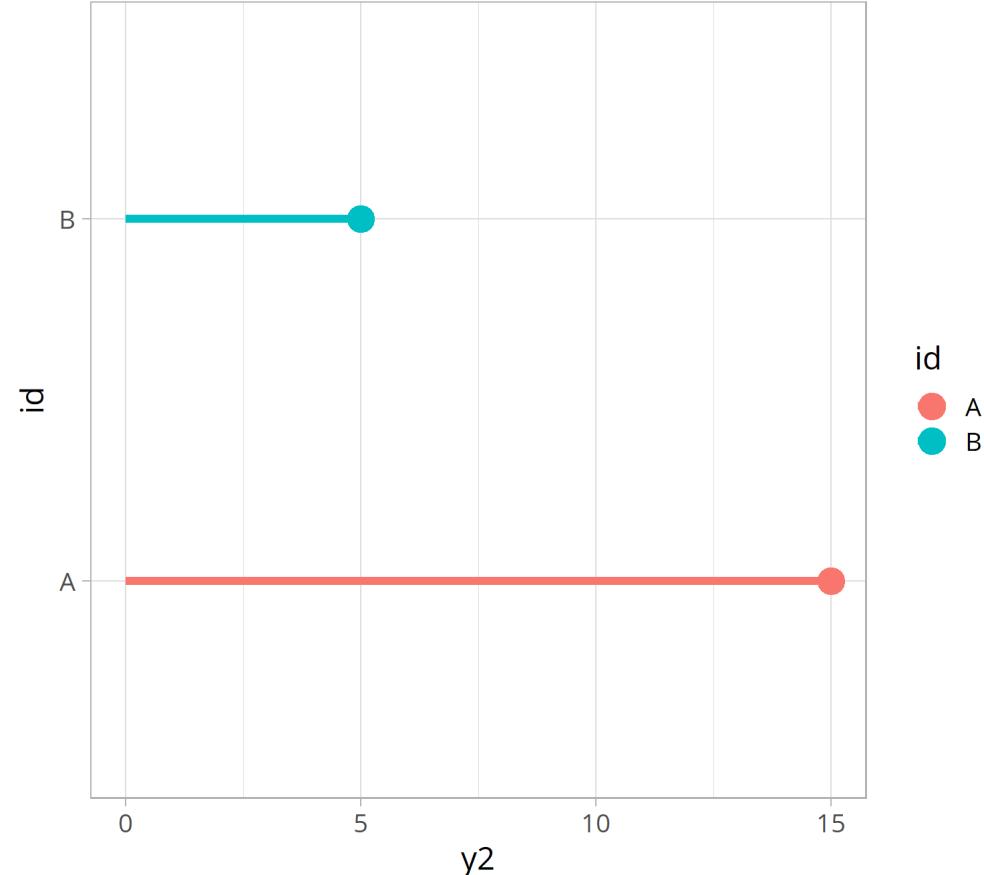
```
ggplot(points, aes(y2, id)) +  
  geom_point() +  
  geom_segment(  
    aes(  
      xend = 0,  
      yend = id,  
      color = id  
    )  
  )
```



geom_*() and **stat_***()

Some need even more inputs:

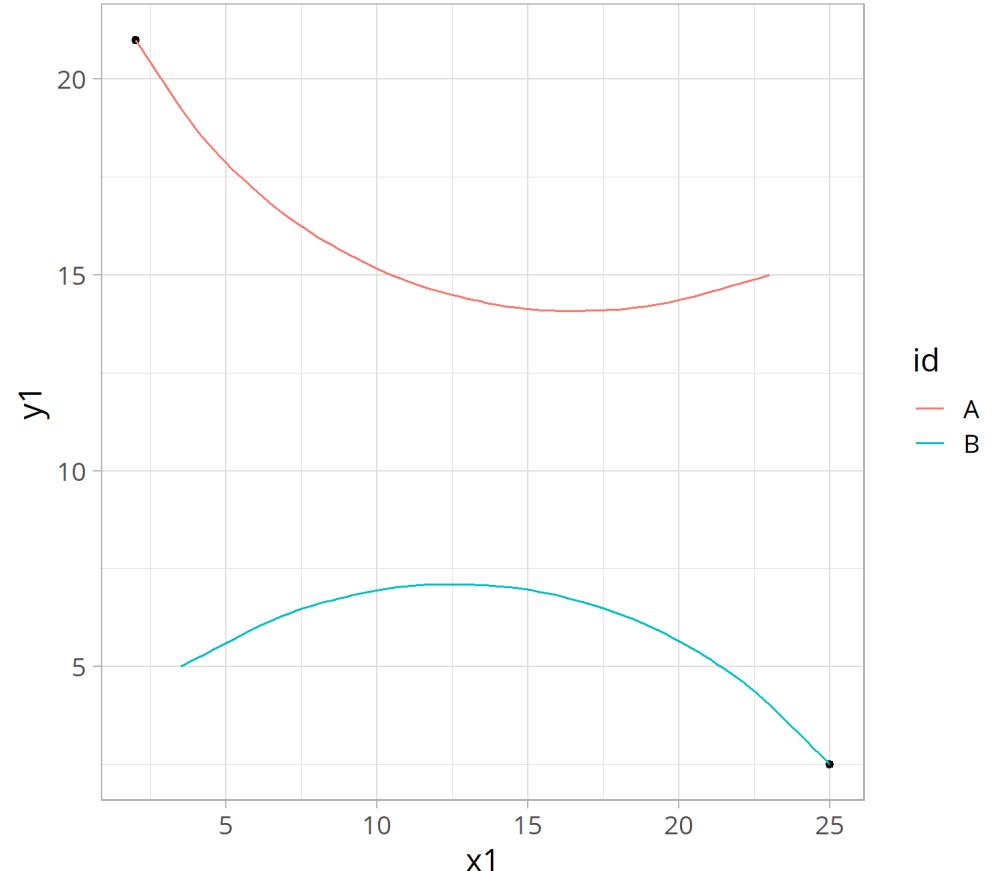
```
ggplot(  
  points,  
  aes(  
    y2,  
    id,  
    color = id  
  )) +  
  geom_point(size = 6) +  
  geom_segment(  
    aes(  
      xend = 0,  
      yend = id  
    ),  
    size = 2  
  )
```



geom_*() and **stat_***()

Some need even more inputs:

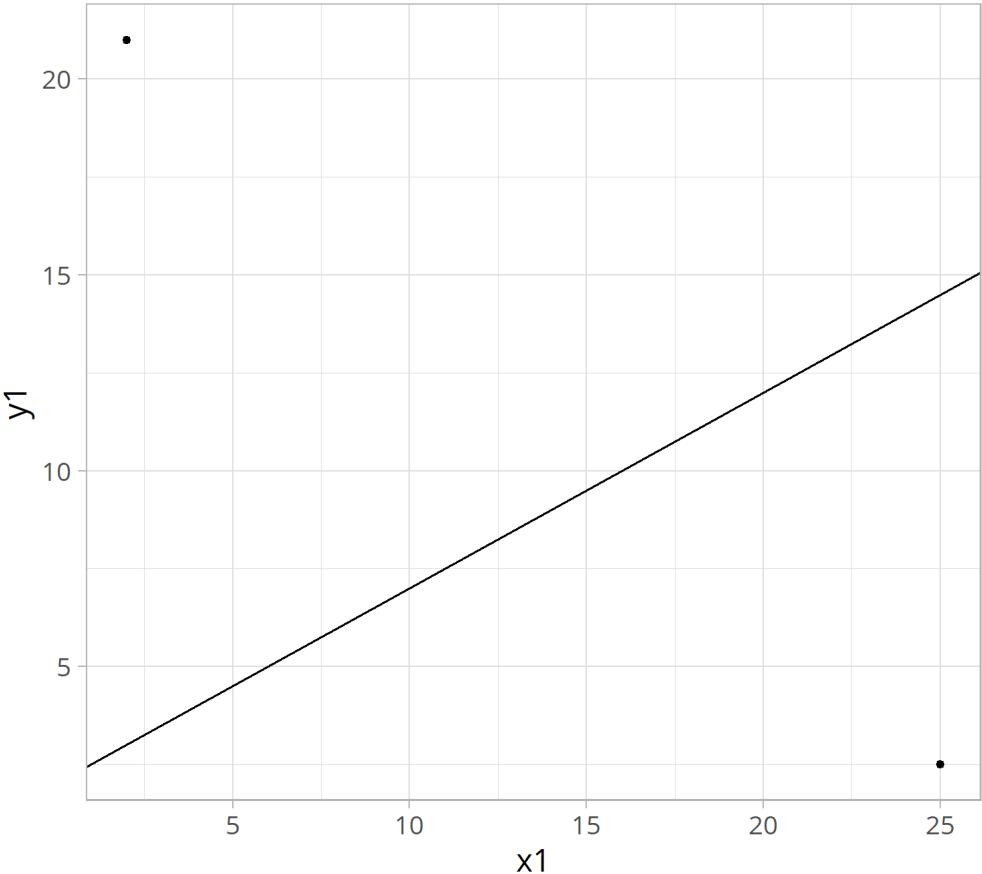
```
ggplot(points, aes(x1, y1)) +  
  geom_point() +  
  geom_curve(  
    aes(  
      xend = x2,  
      yend = y2,  
      color = id  
    ),  
    curvature = .4,  
    arrow = arrow(  
      length = unit(0.03, "pt")  
    )  
)
```



geom_*() and **stat_***()

... or very different input compared to the other **geom**'s:

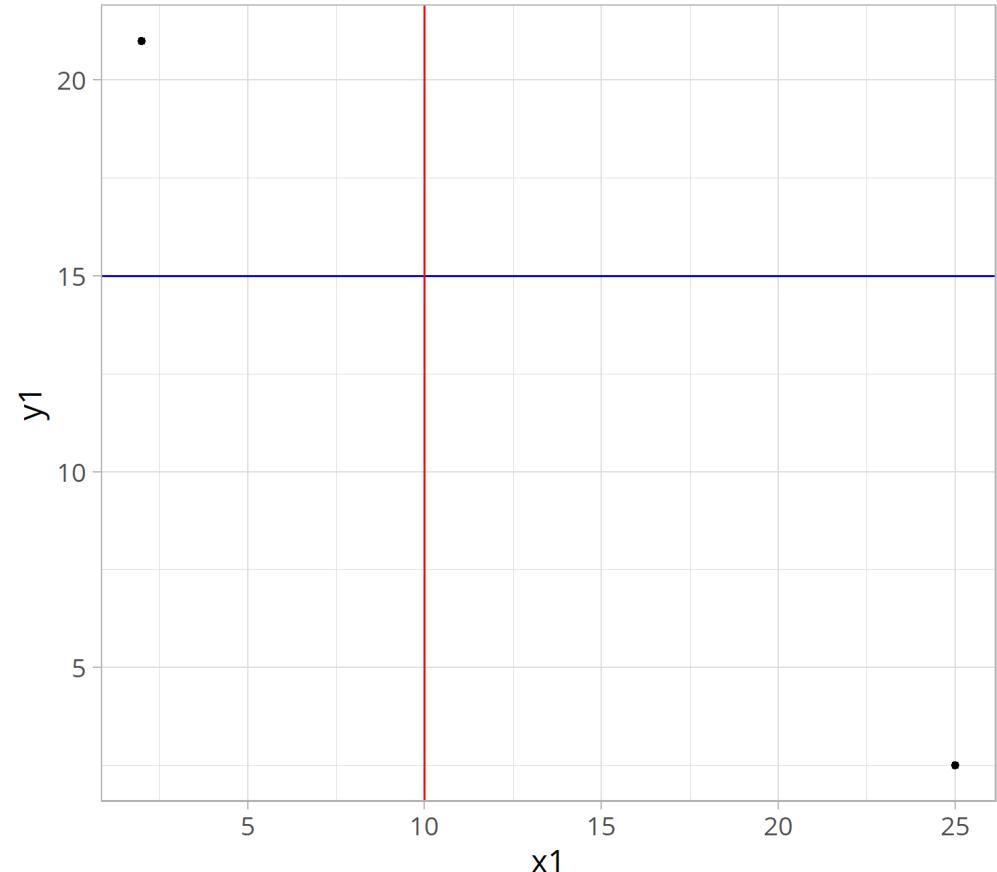
```
ggplot(points, aes(x1, y1)) +  
  geom_point() +  
  geom_abline(  
    slope = .5,  
    intercept = 2  
)
```



geom_*() and **stat_***()

... or very different input compared to the other **geom**'s:

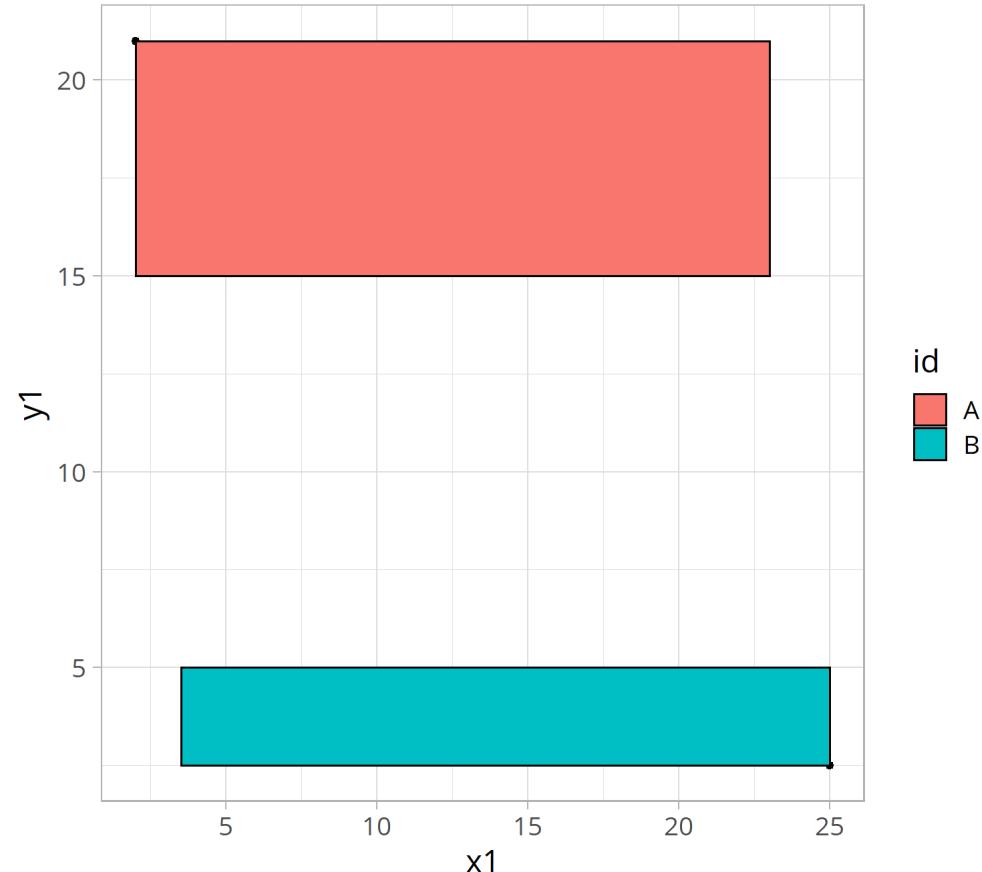
```
ggplot(points, aes(x1, y1)) +  
  geom_point() +  
  geom_hline(  
    yintercept = 15,  
    color = "blue"  
  ) +  
  geom_vline(  
    xintercept = 10,  
    color = "red"  
  )
```



geom_*() and **stat_***()

... or very different input compared to the other **geom**'s:

```
ggplot(points, aes(x1, y1)) +  
  geom_point() +  
  geom_rect(  
    aes(  
      xmin = x1,  
      xmax = x2,  
      ymin = y1,  
      ymax = y2,  
      fill = id  
    ),  
    color = "black"  
)
```



ggplot2: Build a data MASTERpiece



Illustration by Allison Horst (github.com/allisonhorst/stats-illustrations)