Lecture 7

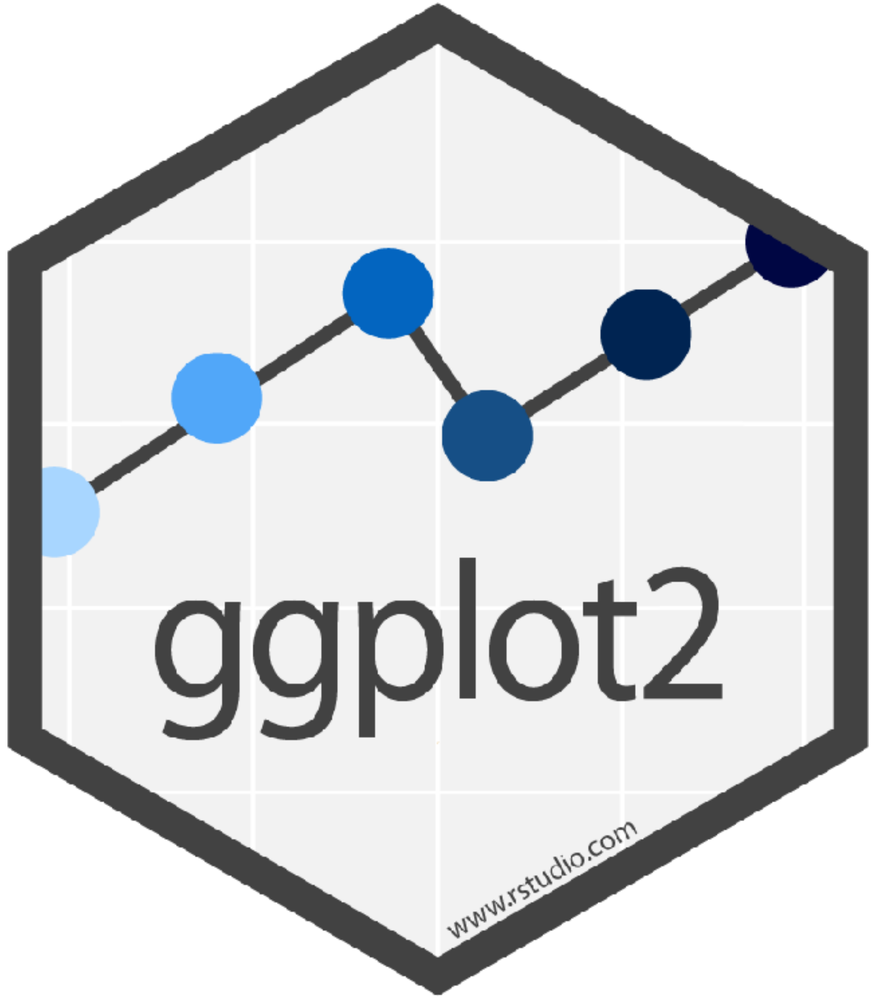
Week 3, Nov 12th 2020
(3.7)

Lecture 4 - Recap

- Introduction to practical sessions
- Group assignment
- Introduction to R Markdown
- Introduction to visual analytics in R
- Introduction to tidyverse

Lecture 7

- Introduction to ggplot2
  - geometric objects
  - layered grammar of graphics
  - statistical transformation
  - position adjustment

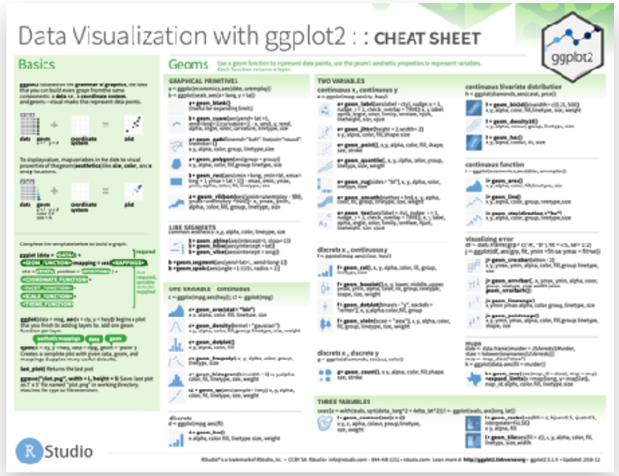


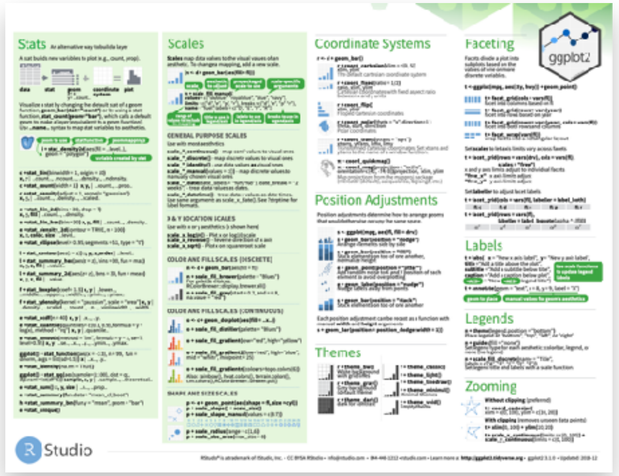
geoms

Geometric objects

geoms

- ggplot2 uses geometric shapes to encode data
- ~30 built-in geoms, and more offered by other packages
- Type `geom_` in console and hit the TAB key…
- Check out the ggplot2 cheatsheet from RStudio!



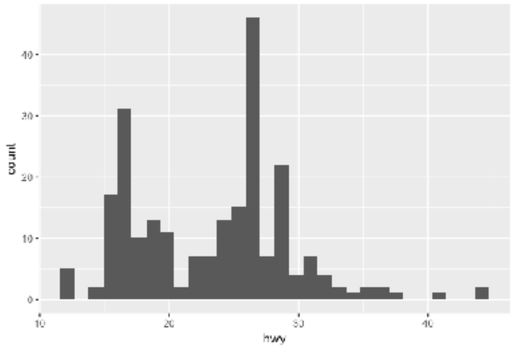


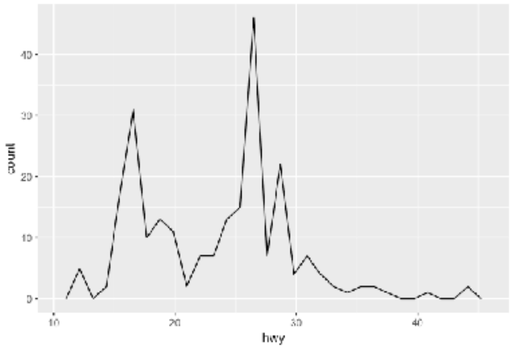

```
geom_abline
geom_bar
geom_bin2d
geom_blank
geom_boxplot
geom_contour
geom_count
geom_hex
geom_crossbar
geom_density
geom_density_2d
geom_dotplot
geom_errorbarh
geom_freqpoly
geom_histogram
geom_jitter
geom_label
geom_map
geom_path
geom_point
geom_polygon
geom_quantile
geom_raster
geom_ribbon
geom_rug
geom_segment
geom_smooth
geom_violin
```

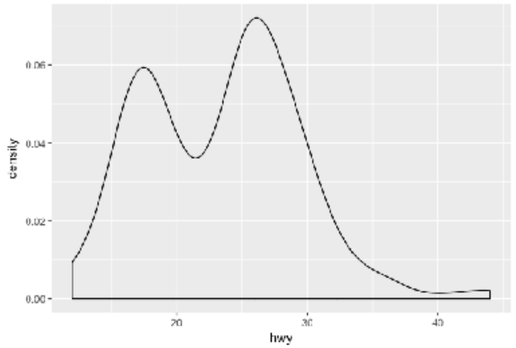
continuous univariate

aesthetic mapping argument

```
ggplot(data = mpg, mapping = aes(x = hwy)) +
  geom_histogram()

ggplot(data = mpg, mapping = aes(x = hwy)) +
  geom_freqpoly()

ggplot(data = mpg, mapping = aes(x = hwy)) +
  geom_density()
```



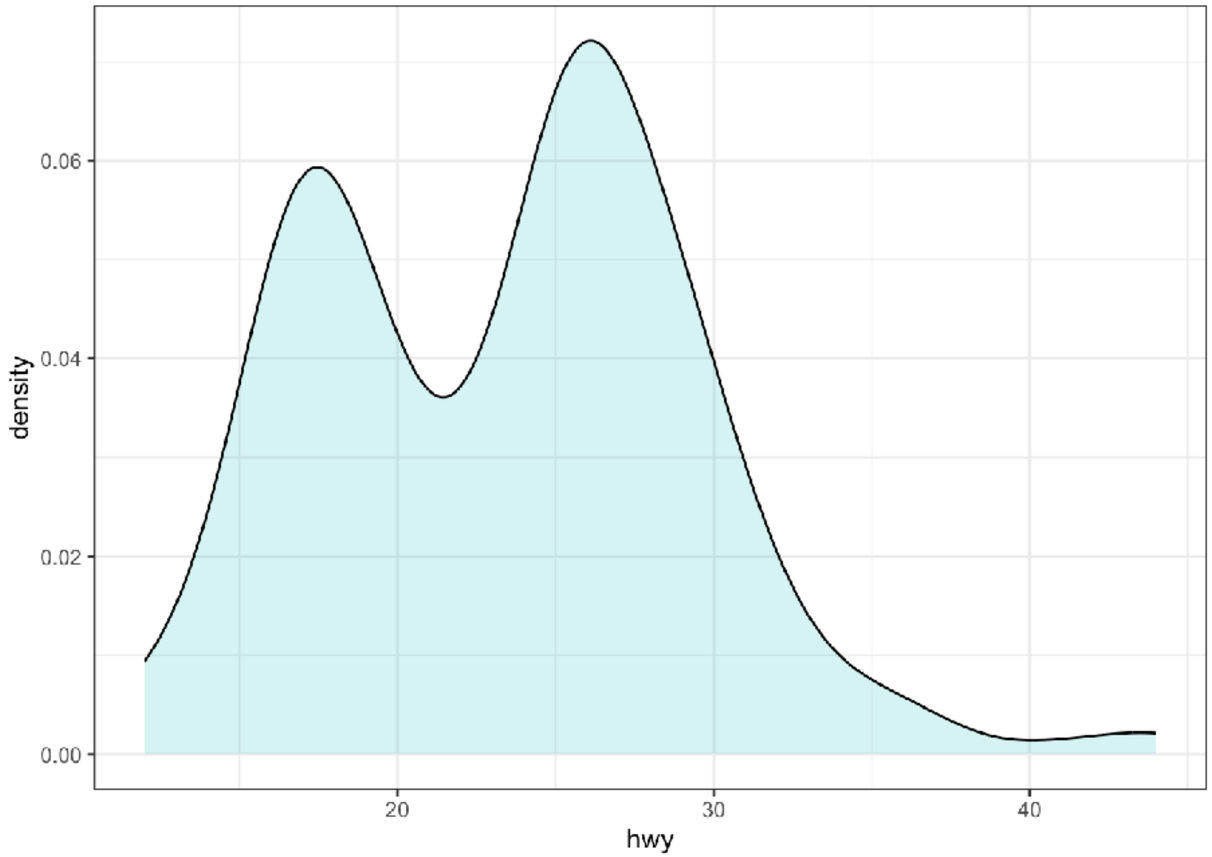




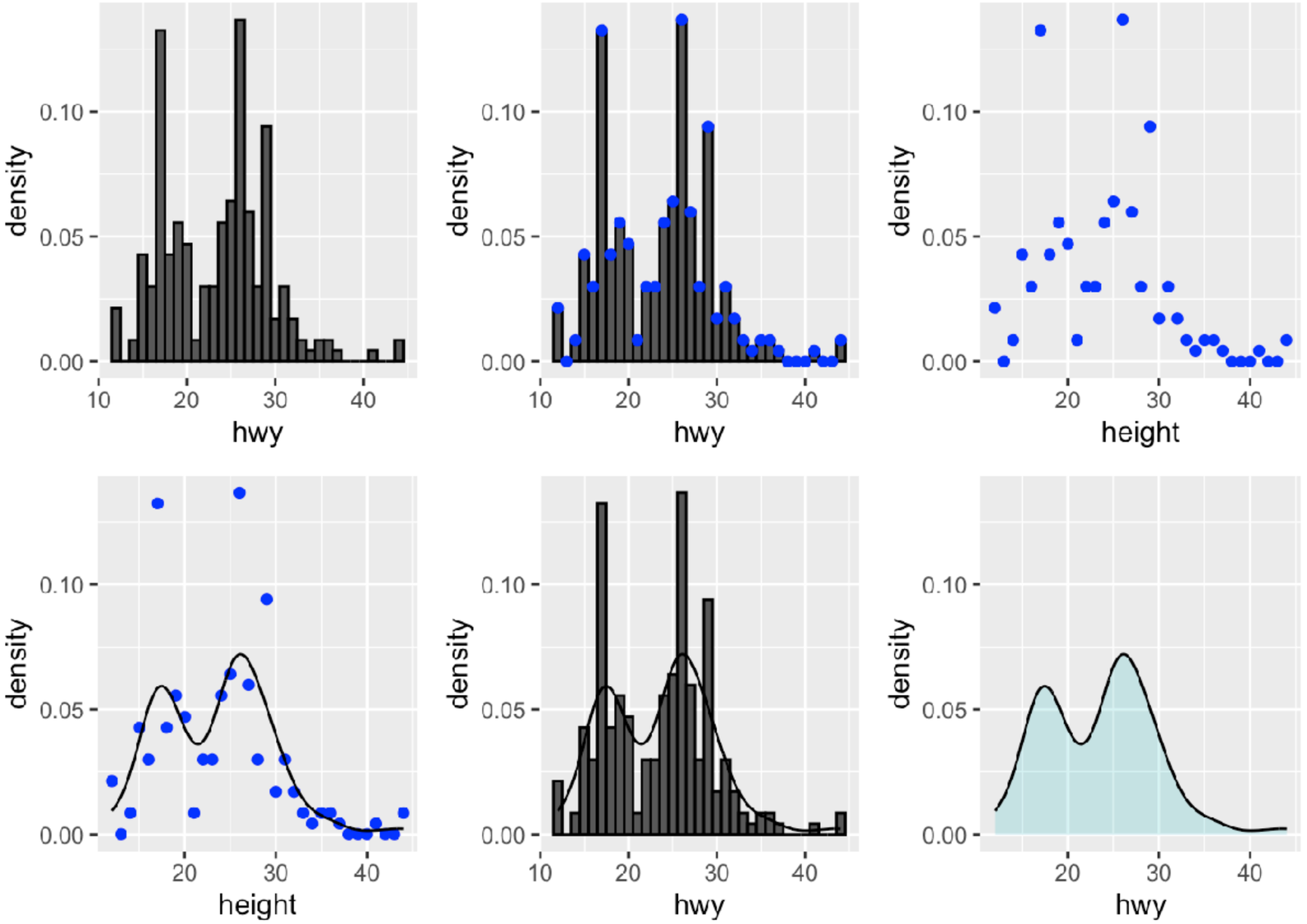
geom_abline
geom_bar
geom_bin2d
geom_blank
geom_boxplot
geom_contour
geom_count
geom_hex
geom_crossbar
geom_density
geom_density_2d
geom_dotplot
geom_errorbarh
geom_freqpoly
geom_histogram
geom_jitter
geom_label
geom_map
geom_path
geom_point
geom_polygon
geom_quantile
geom_raster
geom_ribbon
geom_rug
geom_segment
geom_smooth
geom_violin

geom_density

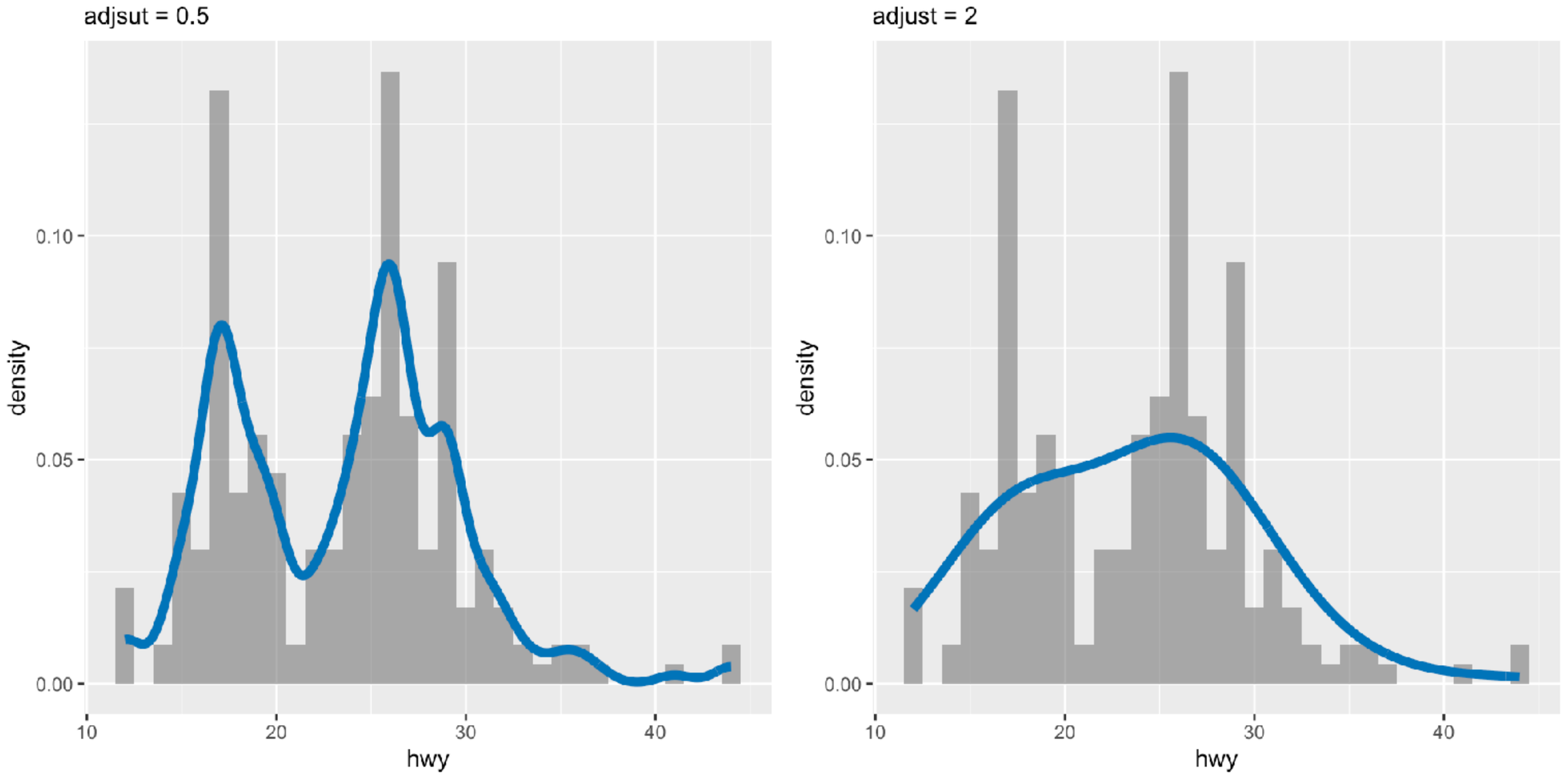
```
ggplot(data = mpg, aes(x = hwy)) +
  geom_density(alpha = .2, fill= "#00BFC4", color = 0) +
  geom_line(stat='density') +
  theme_bw()
```



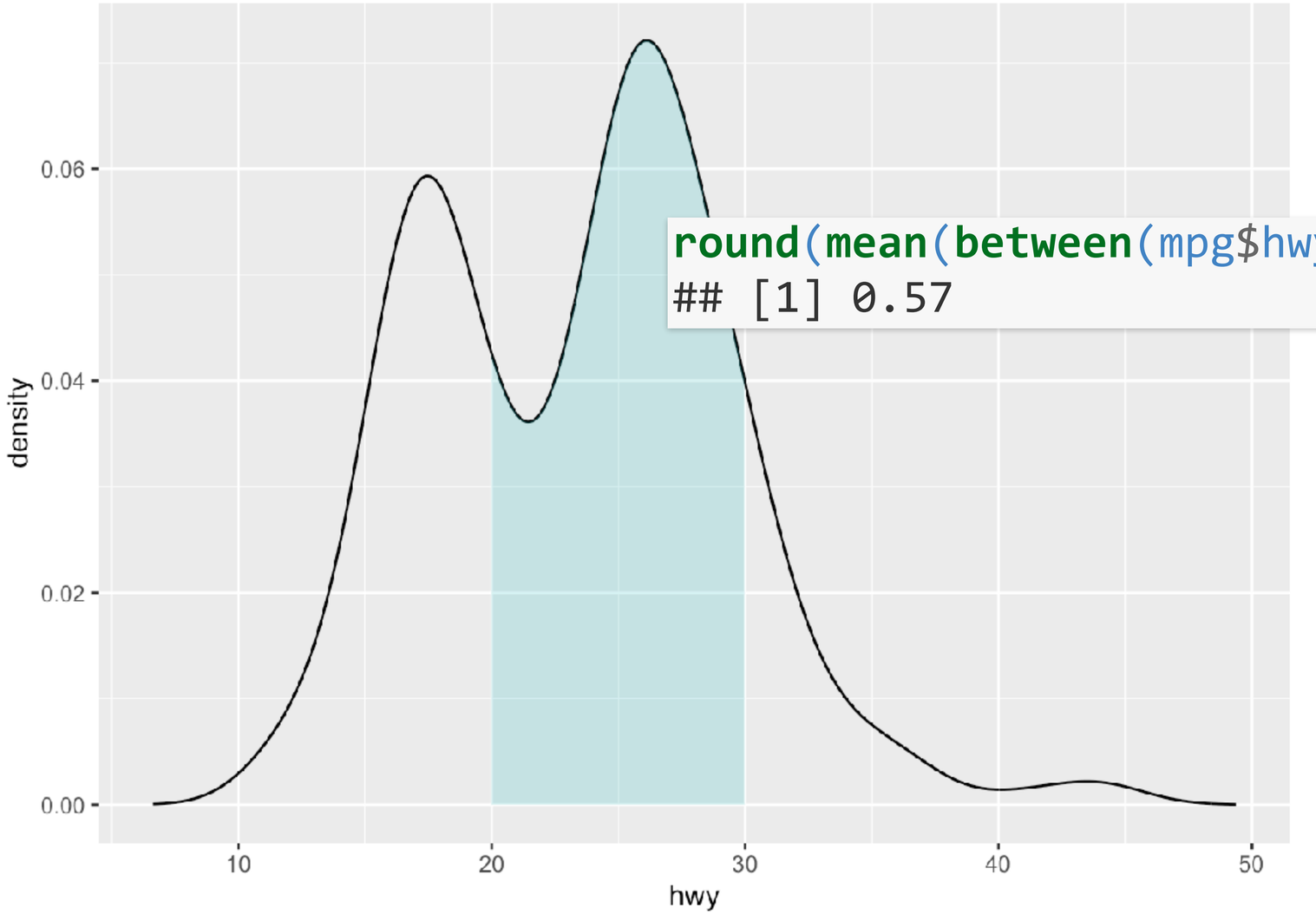
Smooth density



Smoothness



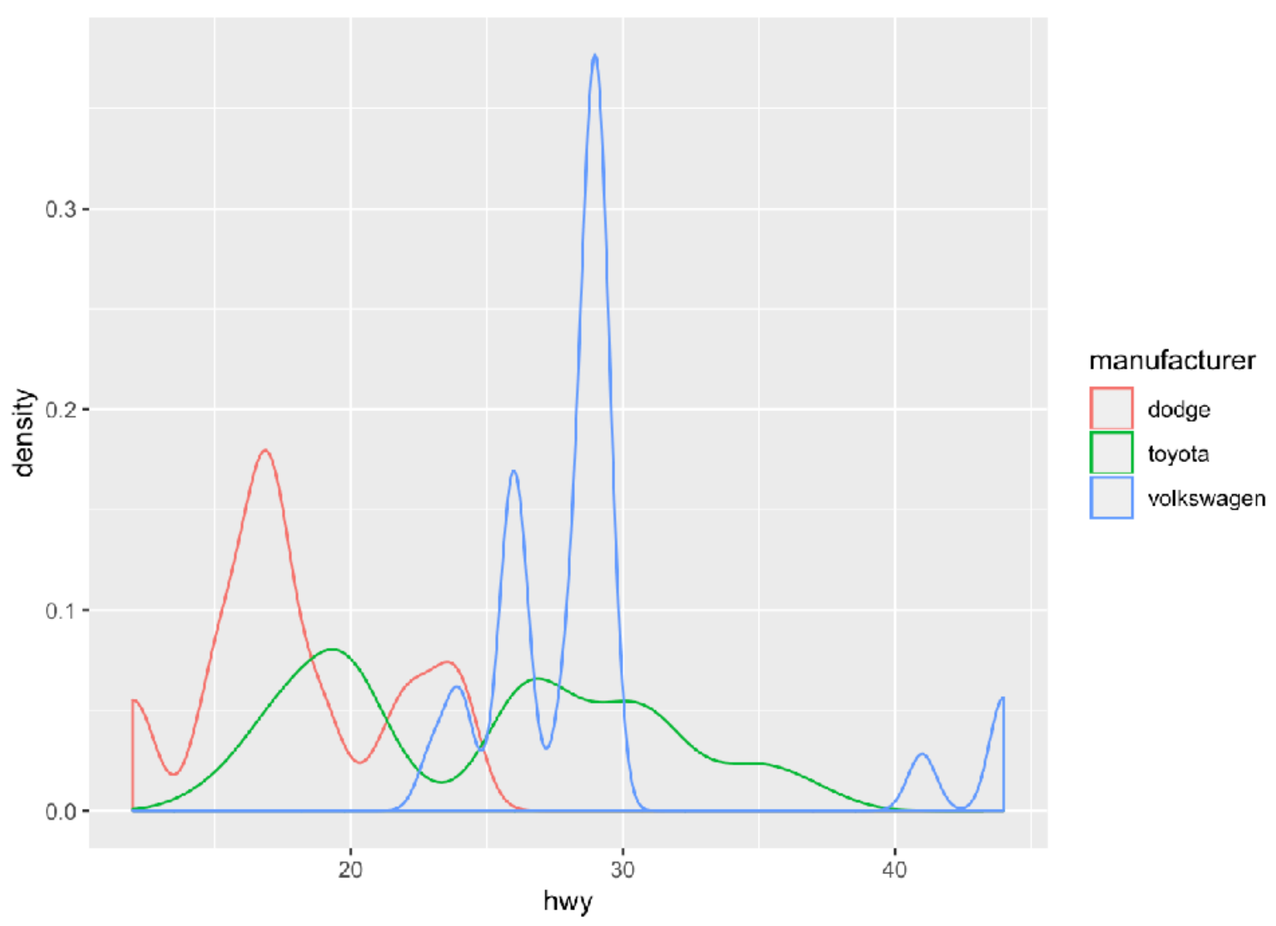
Interpreting y-scale




```
round(mean(between(mpg$hwy, 20, 30)), 2)
## [1] 0.57
```

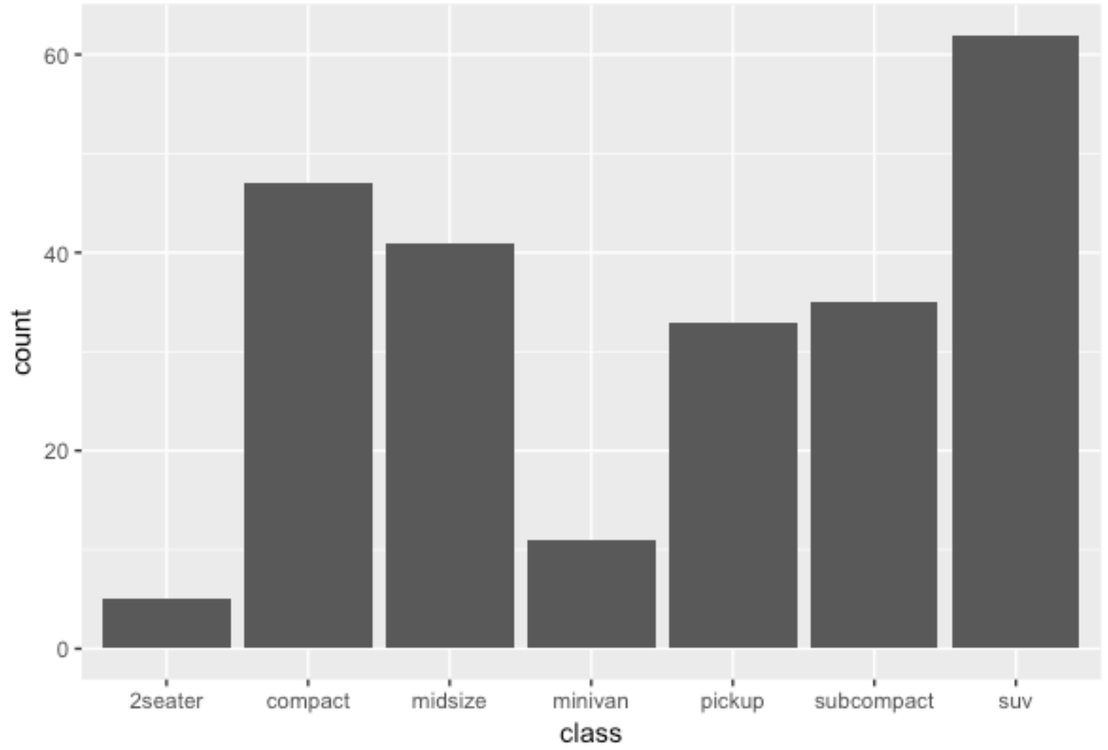
Comparing distribution

```
mpg %>%
  filter(manufacturer %in% c("dodge", "toyota", "volkswagen")) %>%
  ggplot(aes(x = hwy, color = manufacturer)) +
  geom_density(adjust = 0.5)
```



categorical univariate

```
ggplot(data = mpg, mapping = aes(x = class)) +
  geom_bar()
```



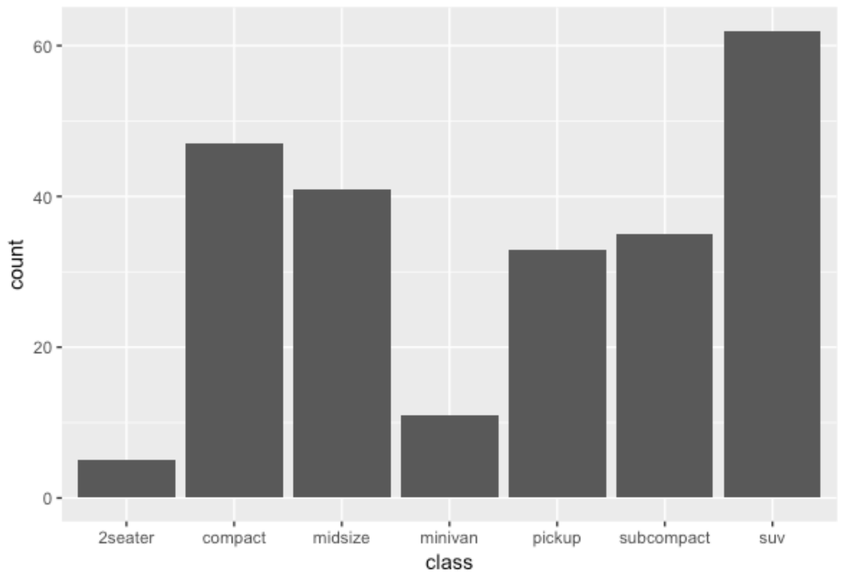
geom_abline
geom_bar
geom_bin2d
geom_blank
geom_boxplot
geom_contour
geom_count
geom_hex
geom_crossbar
geom_density
geom_density_2d
geom_dotplot
geom_errorbarh
geom_freqpoly
geom_histogram
geom_jitter
geom_label
geom_map
geom_path
geom_point
geom_polygon
geom_quantile
geom_raster
geom_ribbon
geom_rug
geom_segment
geom_smooth
geom_violin

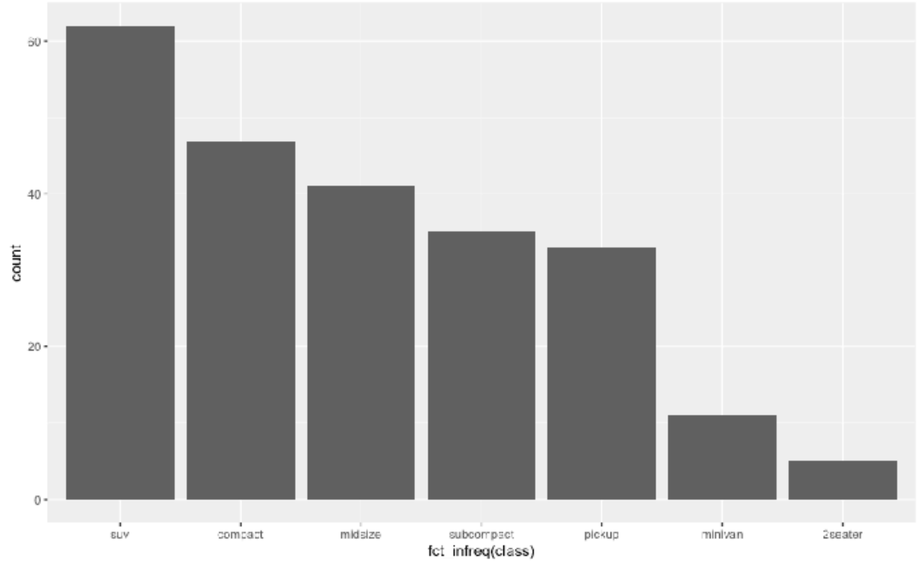
Sorting factors

tidyvere package: `forcats`

forcats functions

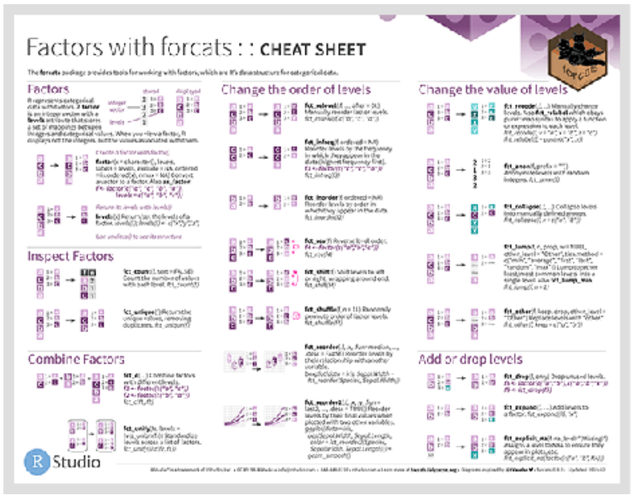
```
# before sorting
ggplot(data = mpg, mapping = aes(x = class)) +
  geom_bar()

# after sorting
ggplot(data = mpg, mapping = aes(x=fct_infreq(class)))+
  geom_bar()
```





forcats:: functions

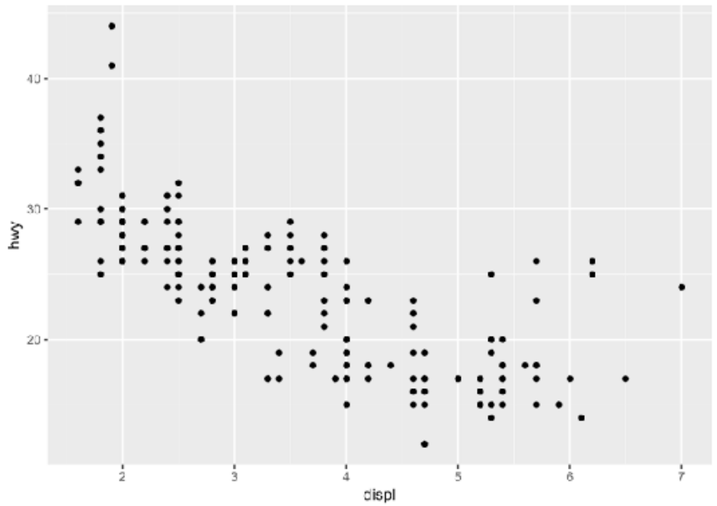
Function	Description
`fct_reorder()`	reordering a factor by another variable
`fct_infreq()`	reodering a factor by the frequency of value
`fct_relevel()`	changing the order of a factor by hand
`fct_lump()`	collapsing the least/most frequent values of a facotr into “other”
`fct_rev()`	reverse order of factor levels

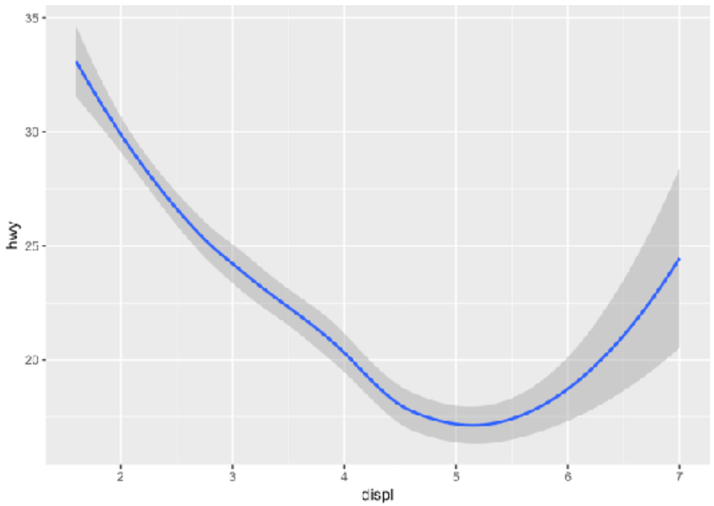


Check out the forcats cheatsheet for other functions!

continuous bivariate

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

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

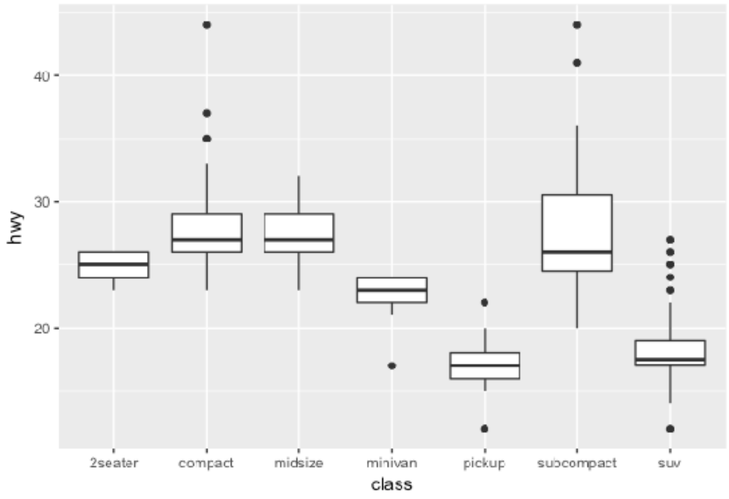


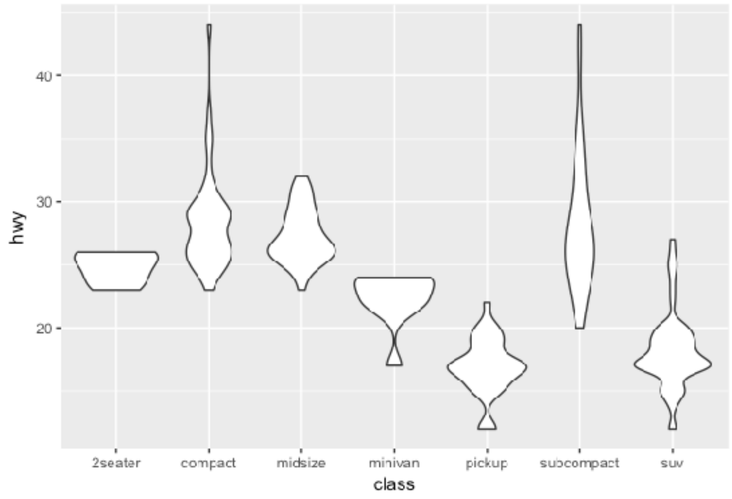


geom_abline
geom_bar
geom_bin2d
geom_blank
geom_boxplot
geom_contour
geom_count
geom_hex
geom_crossbar
geom_density
geom_density_2d
geom_dotplot
geom_errorbarh
geom_freqpoly
geom_histogram
geom_jitter
geom_label
geom_map
geom_path
geom_point
geom_polygon
geom_quantile
geom_raster
geom_ribbon
geom_rug
geom_segment
geom_smooth
geom_violin

bivariate

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

ggplot(data = mpg, aes(x = class, y = hwy)) +
  geom_violin()
```

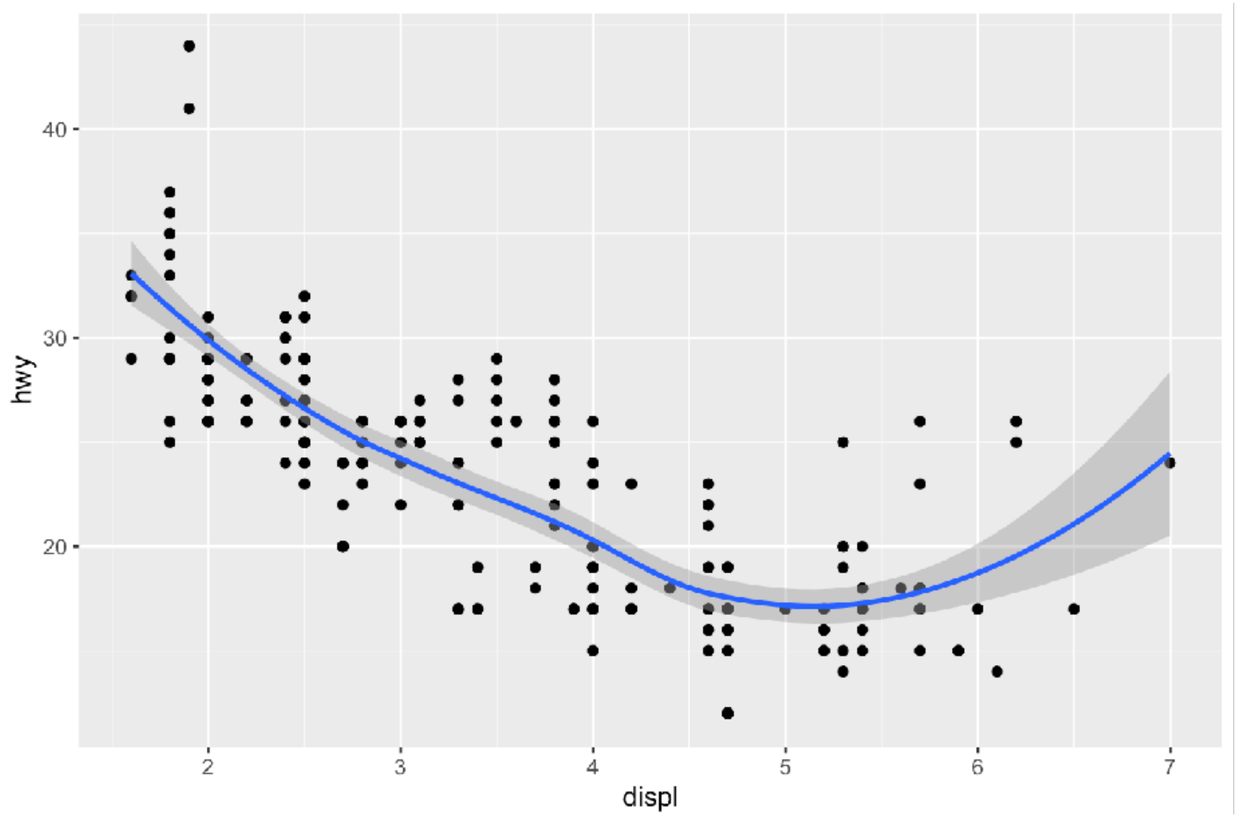




geom_abline
geom_bar
geom_bin2d
geom_blank
geom_boxplot
geom_contour
geom_count
geom_hex
geom_crossbar
geom_density
geom_density_2d
geom_dotplot
geom_errorbarh
geom_freqpoly
geom_histogram
geom_jitter
geom_label
geom_map
geom_path
geom_point
geom_polygon
geom_quantile
geom_raster
geom_ribbon
geom_rug
geom_segment
geom_smooth
geom_violin

layering multiple geoms

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



layering multiple geoms

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

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

- The 2nd code also produces the same graph. Why is it a better code than its above?

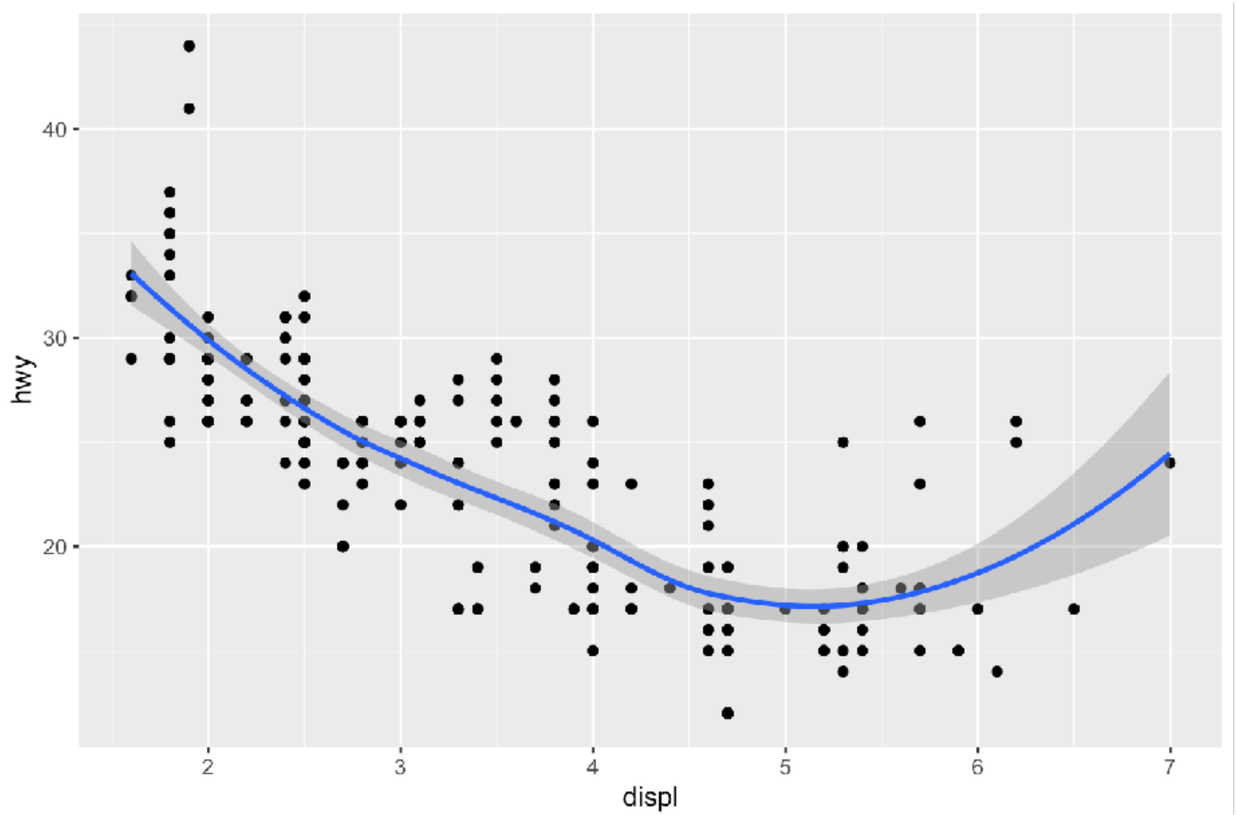
Your turn!

1. Think of what the output will look like, then run the following code to check your prediction.
    1. What does **`se`** arguement do?
    2. What does **`show.legend`** argument do?

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

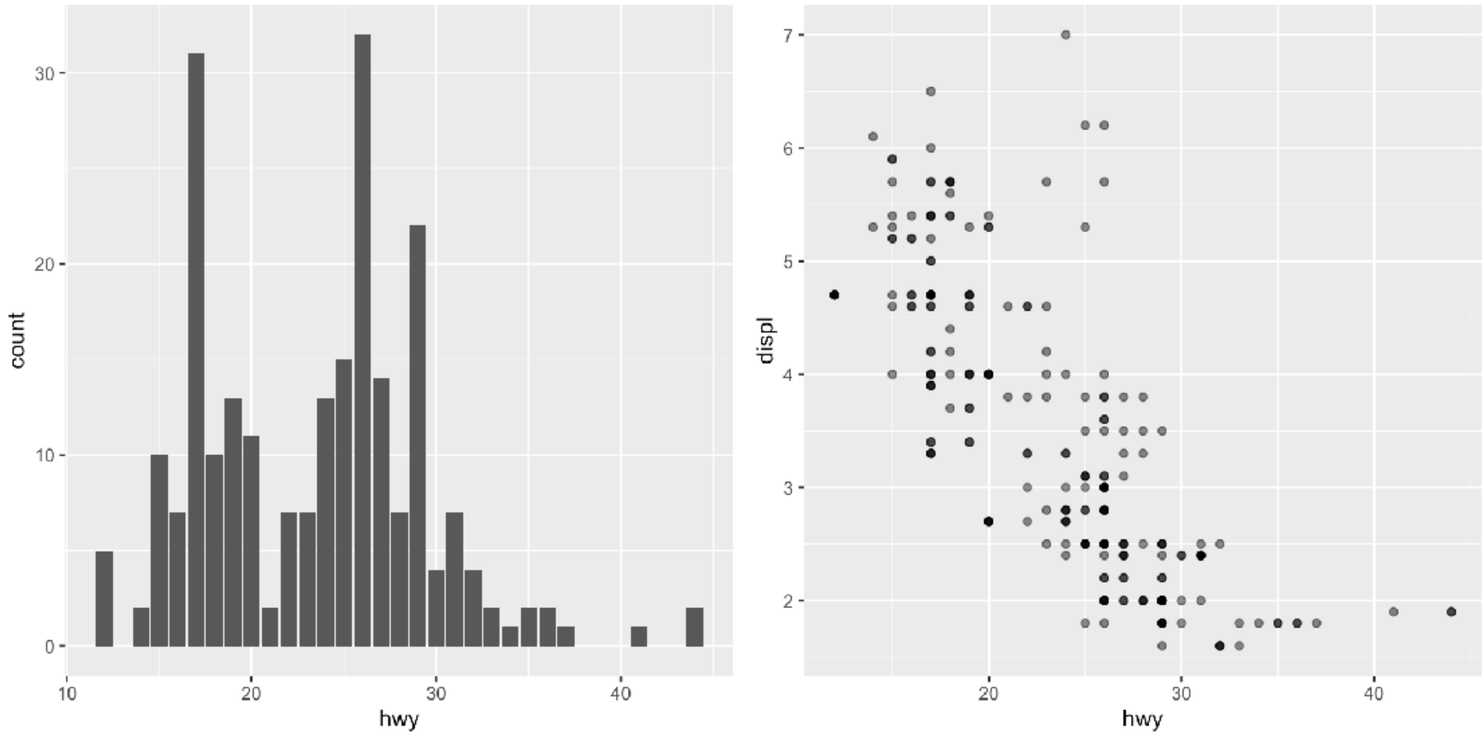
layering multiple geoms

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



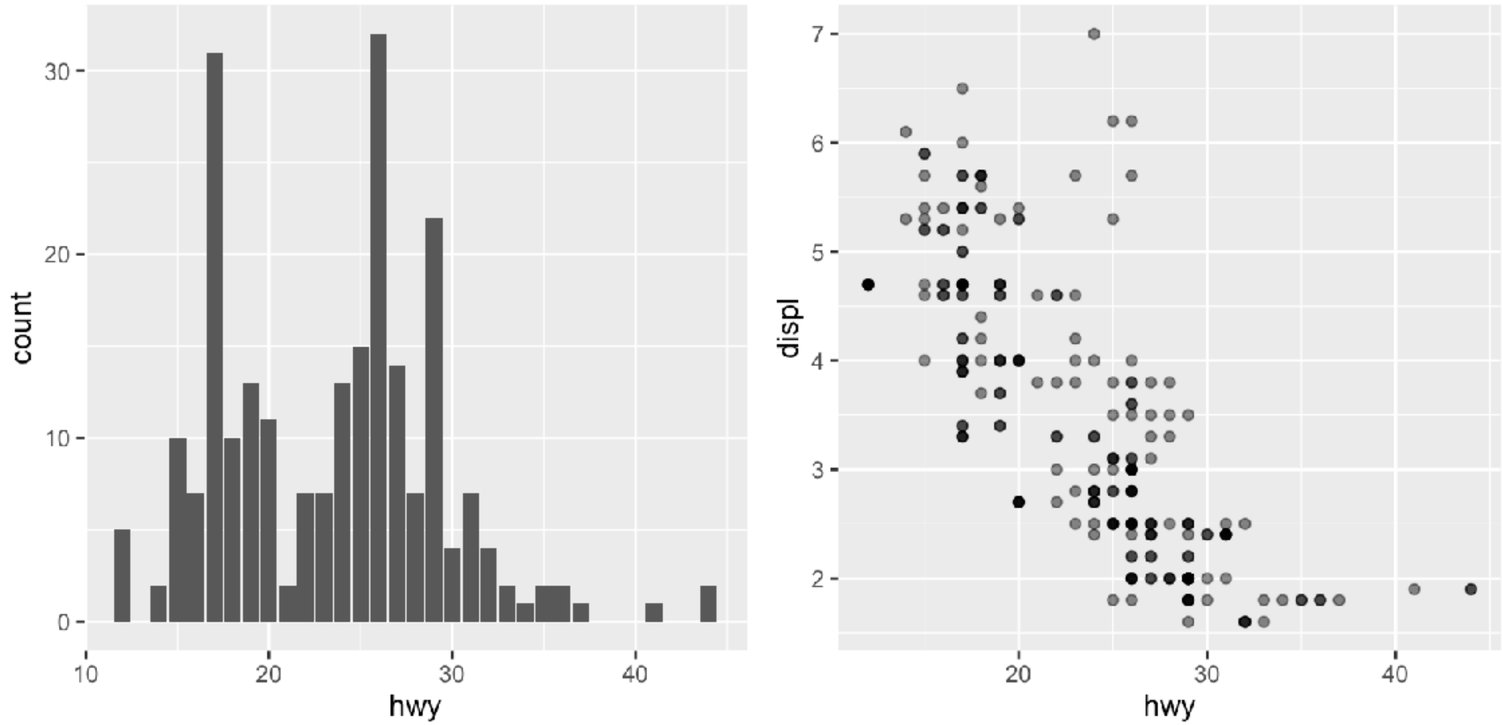
Grid of plots

```
library(gridExtra)

p1 <- ggplot(data = mpg) + geom_bar(aes(x = hwy))
p2 <- ggplot(data = mpg) + geom_point(aes(x = hwy, y = displ), alpha = 0.5)

grid.arrange(p1, p2, ncol = 2)
```



patchwork package

```
library(patchwork)
p1 <- ggplot(data = mpg) + geom_bar(aes(x = hwy))
p2 <- ggplot(data = mpg) + geom_point(aes(x = hwy, y = displ), alpha = 0.5)
p1 + p2
```



https://patchwork.data-imaginist.com/index.html

A layered grammer of graphics

Minard's map

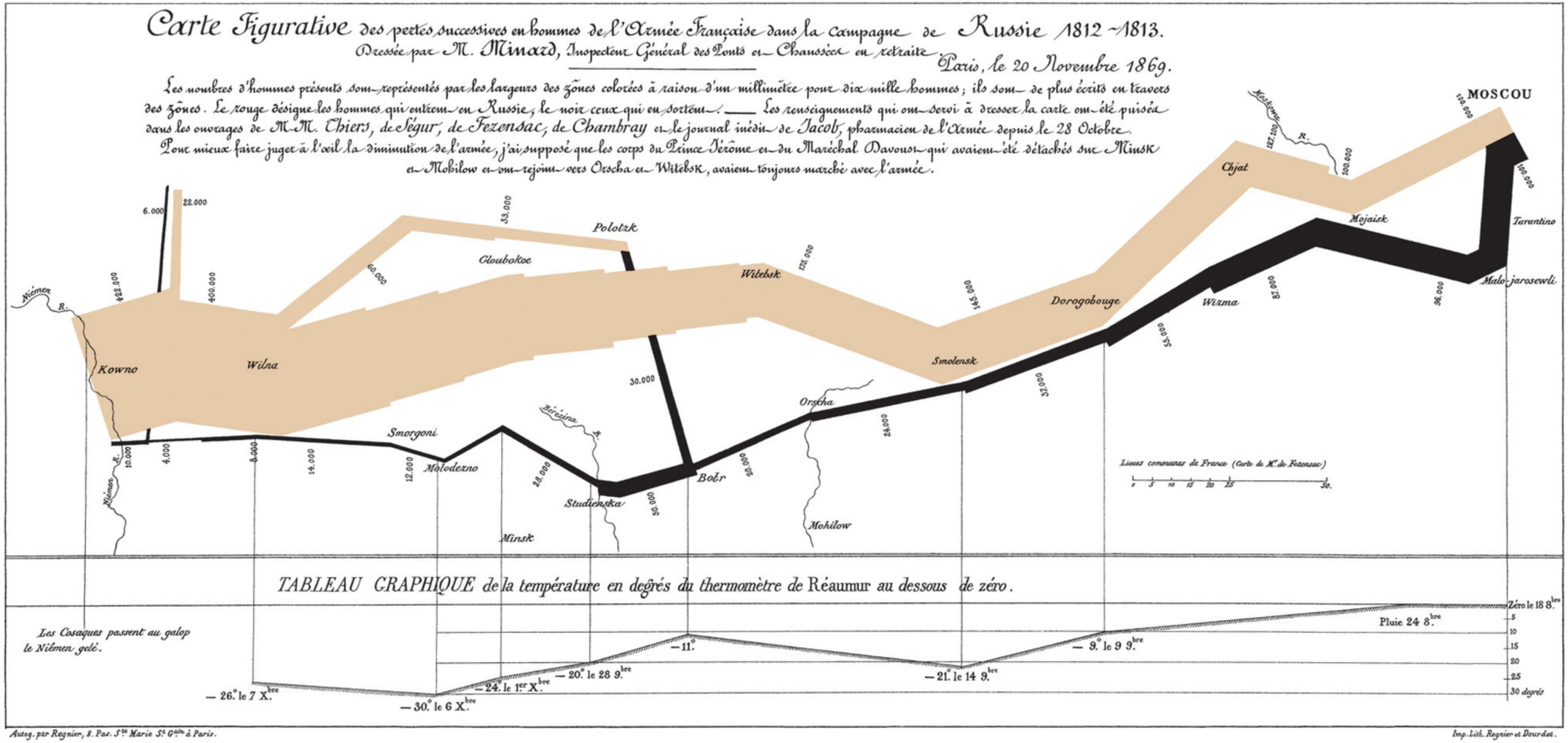


Image from:https://en.wikipedia.org/wiki/File:Minard.png

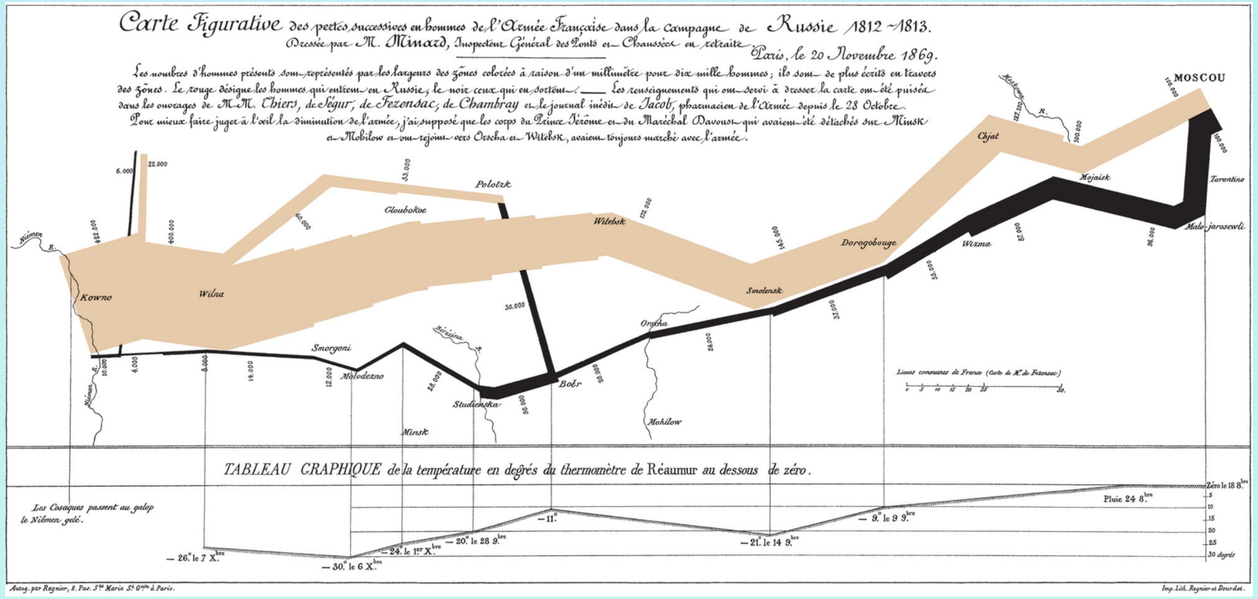
Datasets

- Download the datasets: http://bit.ly/minardDataset
- Load the data. More details of importing data in another lecture…

```
# tidyverse package
library(tidyverse)
# read table with function from readr
troops <- read_table2(“data/minard-troops.txt")

cities <- read_table2("data/minard-cities.txt")
```

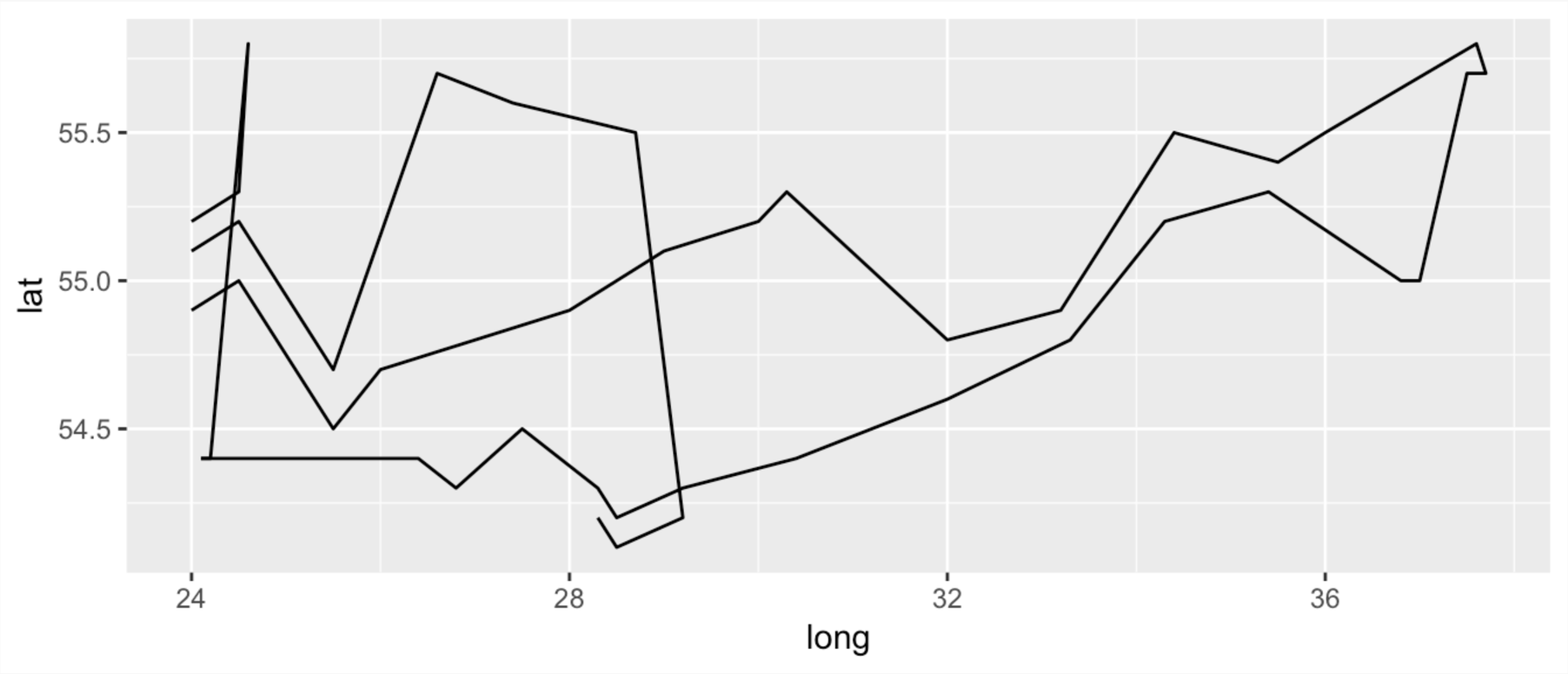
Your turn!

1. Examine the **troops** and **cities** data tables and consider how you would approach reproducing the Minard's map.



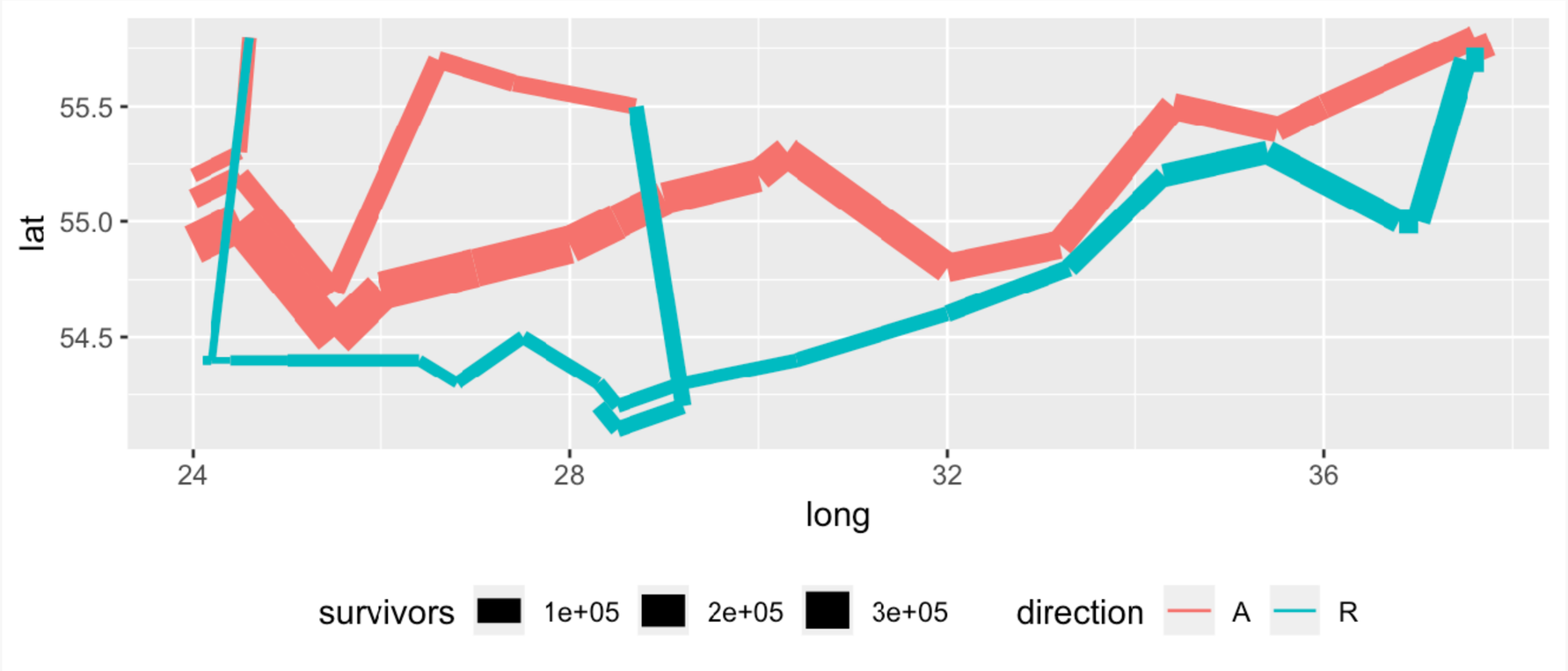
Solution

```
# drawing the marching paths
ggplot(data = troops) +
  geom_path(mapping = aes(x = long, y = lat, group = group))
```



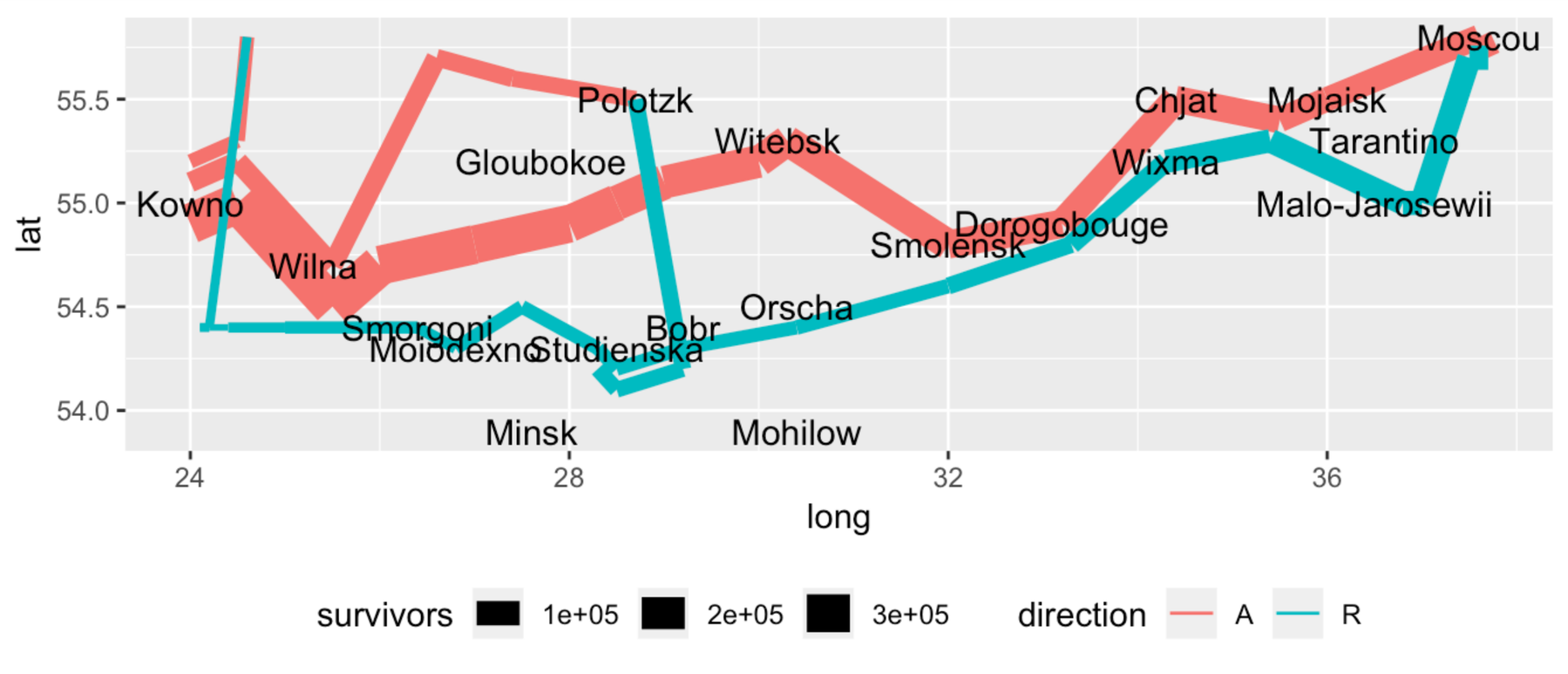
Solution

```
# drawing the marching paths
ggplot(data = troops) +
  geom_path(mapping = aes(x = long, y = lat, size = survivors,
colour = direction, group = group)) +
  theme(legend.position="bottom")
```



Solution

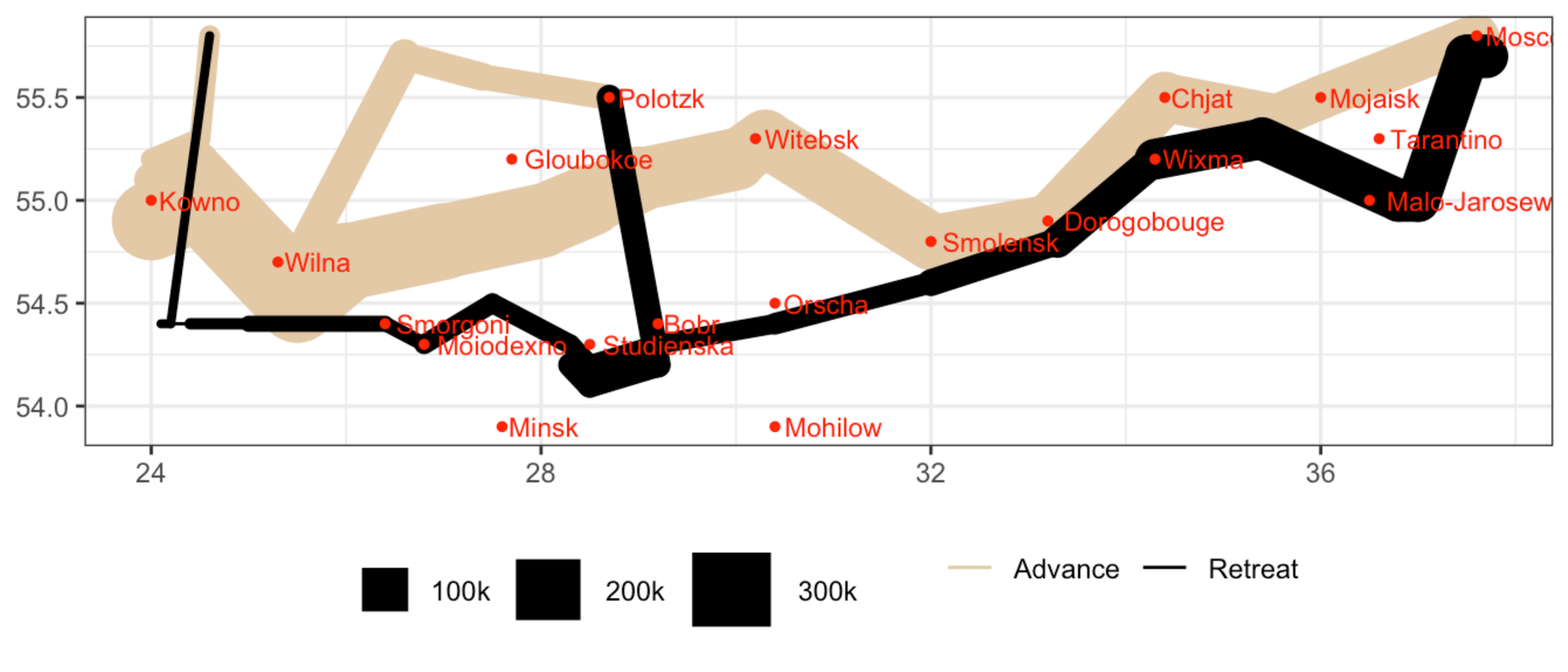
```
# Adding text labels
ggplot() +
  geom_path(data = troops, mapping = aes(x = long, y = lat, size = survivors, color =
direction, group = group)) +
  geom_text(data = cities, mapping = aes(x = long, y = lat, label = city), size = 4) +
  theme(legend.position="bottom")
```



Solution

```
ggplot() +
  geom_path(data = troops, mapping = aes(x = long, y = lat, size = survivors, color = direction, group = group),
lineend = "round", linejoin = "mitre") +
  scale_size(range = c(0.5, 12), limits = c(4000, 350000), trans = "identity", breaks = c(100000, 200000, 300000),
labels = c("100k", "200k", "300k"))+
  scale_color_manual(values = c("#E5CBAA","black"), labels = c("Advance", "Retreat")) +
  xlab(NULL) +
  ylab(NULL) +
  geom_point(data = cities, mapping = aes(x = long, y = lat, label = city), size = 1, color = "red")+
  geom_text(data = cities, mapping = aes(x = long, y = lat, label = city), size = 3, color = "red", hjust =-0.1) +
  theme_bw()+
  theme(legend.position="bottom")+
  guides(size = guide_legend(title = NULL), color = guide_legend(title =NULL))
```

Solution



Further reading link

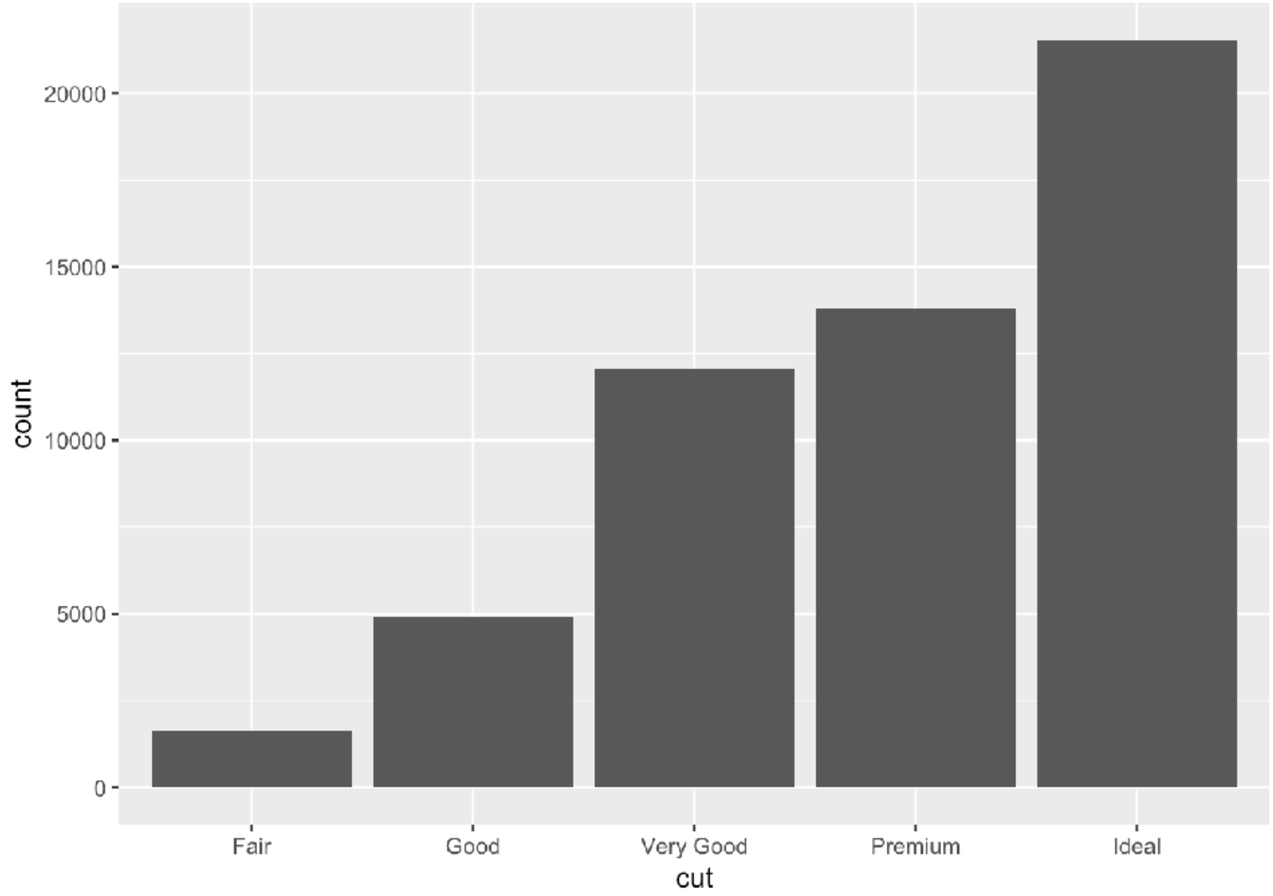
Statistical transformation

behind the scene

Statistical transformation

- Inner working of a seemingly simple bar chart
- **`diamonds`** datasets

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



Default stat

- look up in documentation: `?geom_bar()`

geom_bar {ggplot2} R Documentation

Bar charts

Description

There are two types of bar charts: `geom_bar()` and `geom_col()`. `geom_bar()` makes the height of the bar proportional to the number of cases in each group (or if the `weight` aesthetic is supplied, the sum of the weights). If you want the heights of the bars to represent values in the data, use `geom_col()` instead. `geom_bar()` uses `stat_count()` by default: it counts the number of cases at each x position. `geom_col()` uses `stat_identity()`: it leaves the data as is.

`stat_count()`

Usage

```
geom_bar(mapping = NULL, data = NULL, stat = "count",
  position = "stack", ..., width = NULL, binwidth = NULL,
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

geom_col(mapping = NULL, data = NULL, position = "stack", ...,
  width = NULL, na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE)

stat_count(mapping = NULL, data = NULL, geom = "bar",
  position = "stack", ..., width = NULL, na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE)
```

Arguments

Overriding case 1

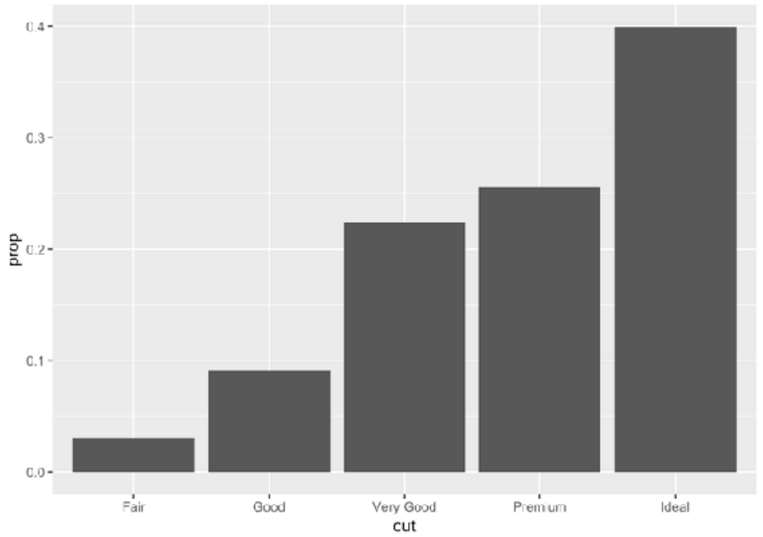
- WYSIWYG (what you see is what you get)
  - You may want to pre-calculate and supply the value exactly to create a bar chart
  - user **`stat = “identity”`** to define values for `x` and `y`.

```
# pre-calculate
cut <- c("Fair","Good", "Very Good", "Premium", "Ideal")
freq <- c(1610, 4906, 12082, 13791, 21551)
demo <- data_frame(cut, freq)

ggplot(data = demo) +
  geom_bar(mapping = aes(x = cut, y = freq), stat = "identity")
```

Overriding case 2

- Using computed variables
  - `geom_bar()` : `prop` returns groupwise proportion
  - access a computed variable by surrounding with two periods, e.g. **..prop..**

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, y = ..prop.., group = "demo"))
```



Computed variables

- look up in documentation: **`?geom_bar()`**

- `group`
- `linetype`
- `size`

Learn more about setting these aesthetics in `vignette("ggplot2-specs")`.

Computed variables

count

number of points in bin

prop

groupwise proportion

See Also

`geom_histogram()` for continuous data, `position_dodge()` and `position_dodge2()` for creating side-by-side bar charts.

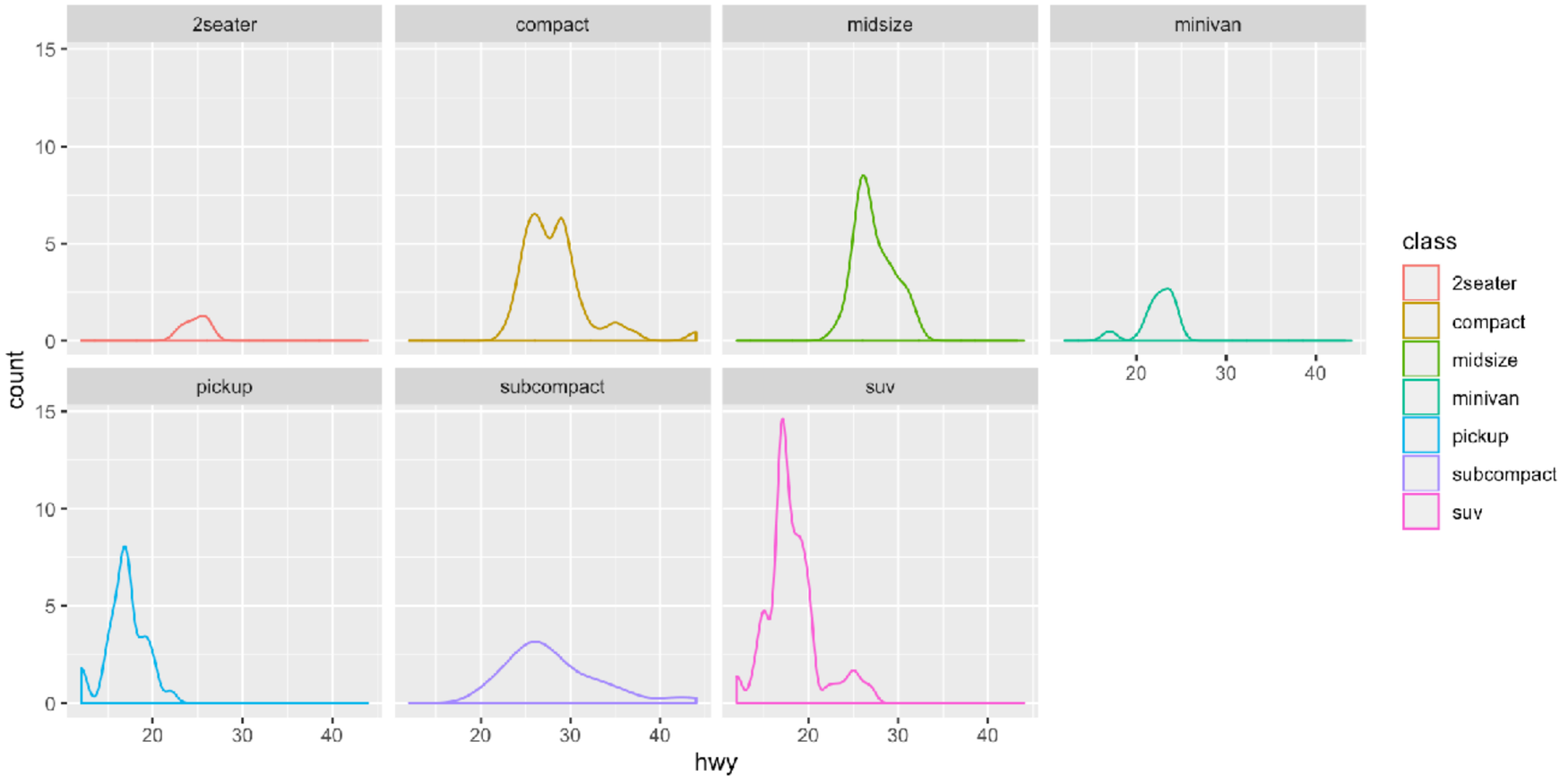
`stat_bin()`, which bins data in ranges and counts the cases in each range. It differs from `stat_count`, which counts the number of cases at each `x` position (without binning into ranges). `stat_bin()` requires continuous `x` data, whereas `stat_count` can be used for both discrete and continuous `x` data.

Examples

```
# geom_bar is designed to make it easy to create bar charts that show
```

Density plot

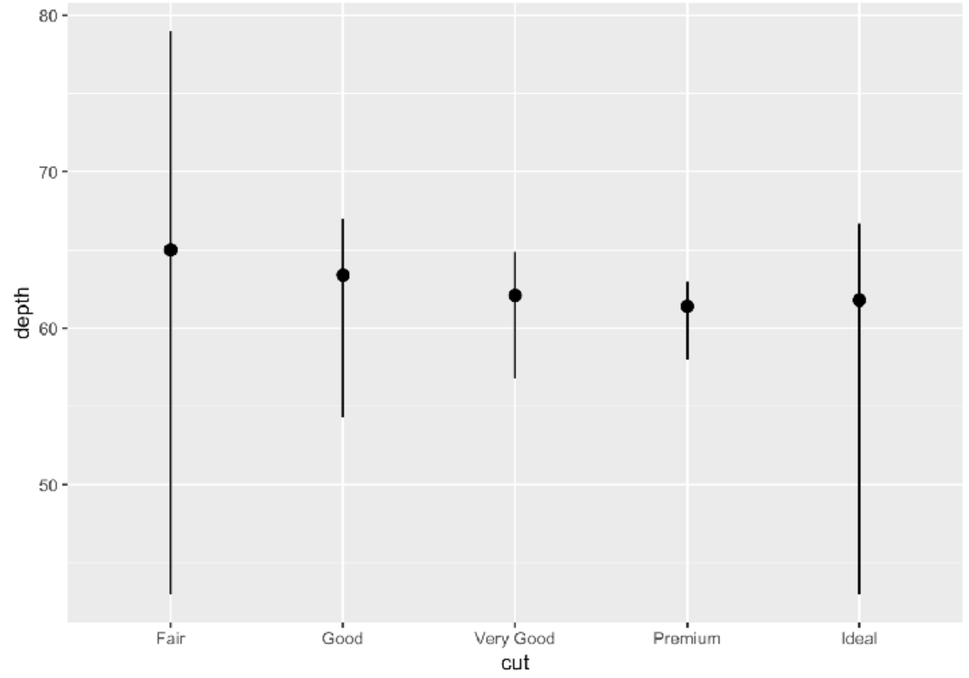
```
mpg %>%
  ggplot(aes(x = hwy, color = class)) +
  geom_density(aes(y = ..count..)) +
  facet_wrap(vars(class), nrow = 2)
```



Overriding case 3

- Using non-default statistical transformation
  - see the cheatsheet for other statistical transformations
  - example **`stat_summary()`**
  - **`cut`**: a categorical variable
  - **`depth`**: a quantitative variable for the total depth percentatge

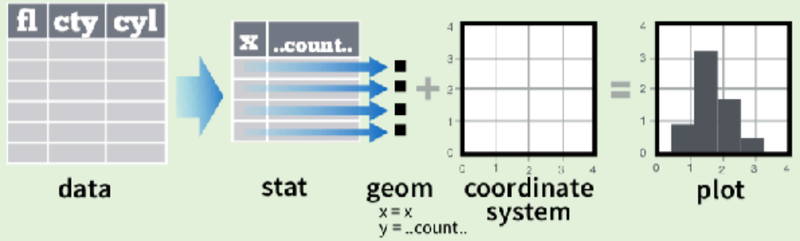
```
ggplot(data = diamonds) +
  stat_summary(mapping = aes(x = cut, y = depth),
fun.ymin = min, fun.ymax = max, fun.y = median)
```



ggplot2 cheatsheet

Stats - An alternative way to build a layer

Some plots visualize a **transformation** of the original data set. Use a **stat** to choose a common transformation to visualize, e.g. a + geom_bar(stat = "bin")



Each stat creates additional variables to map aesthetics to. These variables use a common **..name..** syntax.

stat functions and geom functions both combine a stat with a geom to make a layer, i.e. stat_bin(geom="bar") does the same as geom_bar(stat="bin")

stat function | layer specific mappings | variable created by transformation

```
i + stat_density2d(aes(fill = ..level..),
    geom = "polygon", n = 100)
```

geom for layer | parameters for stat

1D distributions

```
a + stat_bin(binwidth = 1, origin = 10)
    x, y | ..count.., ..ncount.., ..density.., ..ndensity..
a + stat_bindot(binwidth = 1, binaxis = "x")
    x, y, | ..count.., ..ncount..
a + stat_density(adjust = 1, kernel = "gaussian")
    x, y, | ..count.., ..density.., ..scaled..
```

2D distributions

```
f + stat_bin2d(bins = 30, drop = TRUE)
    x, y, fill | ..count.., ..density..
f + stat_binhex(bins = 30)
    x, y, fill | ..count.., ..density..
f + stat_density2d(contour = TRUE, n = 100)
    x, y, color, size | ..level..
```

3 Variables

```
m + stat_contour(aes(z = z))
    x, y, z, order | ..level..
m+ stat_spoke(aes(radius= z, angle =  z))
    angle, radius, x, xend, y, yend | ..x.., ..xend.., ..y.., ..yend..
m + stat_summary_hex(aes(z = z), bins = 30, fun = mean)
    x, y, z, fill | ..value..
m + stat_summary2d(aes(z = z), bins = 30, fun = mean)
    x, y, z, fill | ..value..
```

Comparisons

```
g + stat_boxplot(coef = 1.5)
    x, y | ..lower.., ..middle.., ..upper.., ..outliers..
g + stat_ydensity(adjust = 1, kernel = "gaussian", scale = "area")
    x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..
```

Your turn!

1. What is the default geom associated with **`stat_summary()`**? How can you rewrite the previous plot to use that geom function instead of the stat function? (Answer: **`geom_pointrange()`**)
2. What does **`geom_col()`** do? How is it different to **`geom_bar()`**? (Hint: Read the documentation)
3. What variables does **`stat_smooth()`** compute? What are parameters involved in controlling its behaviour?

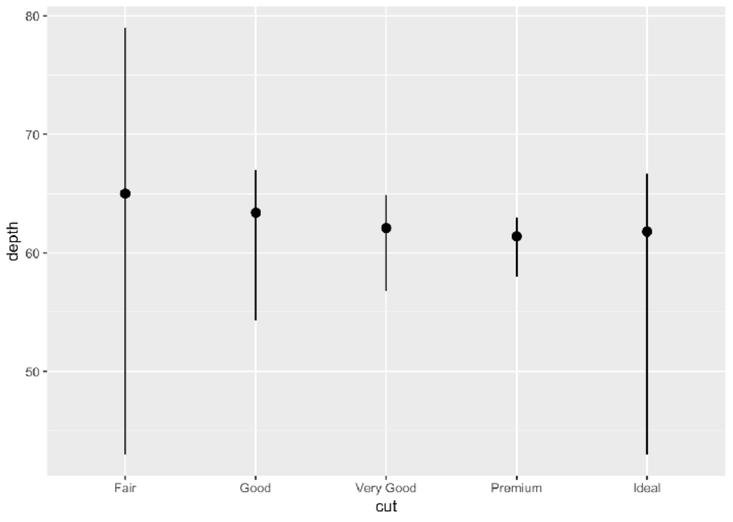
```
stat_summary(mapping = NULL, data = NULL, geom = "pointrange",
  position = "identity", ..., fun.data = NULL, fun.y = NULL,
  fun.ymax = NULL, fun.ymin = NULL, fun.args = list(),
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

Solution

- stat_summary():
  - default **geom** is **geom_pointrange()**
- geom_pointrange():
  - default **stat** is **stat_identity()**

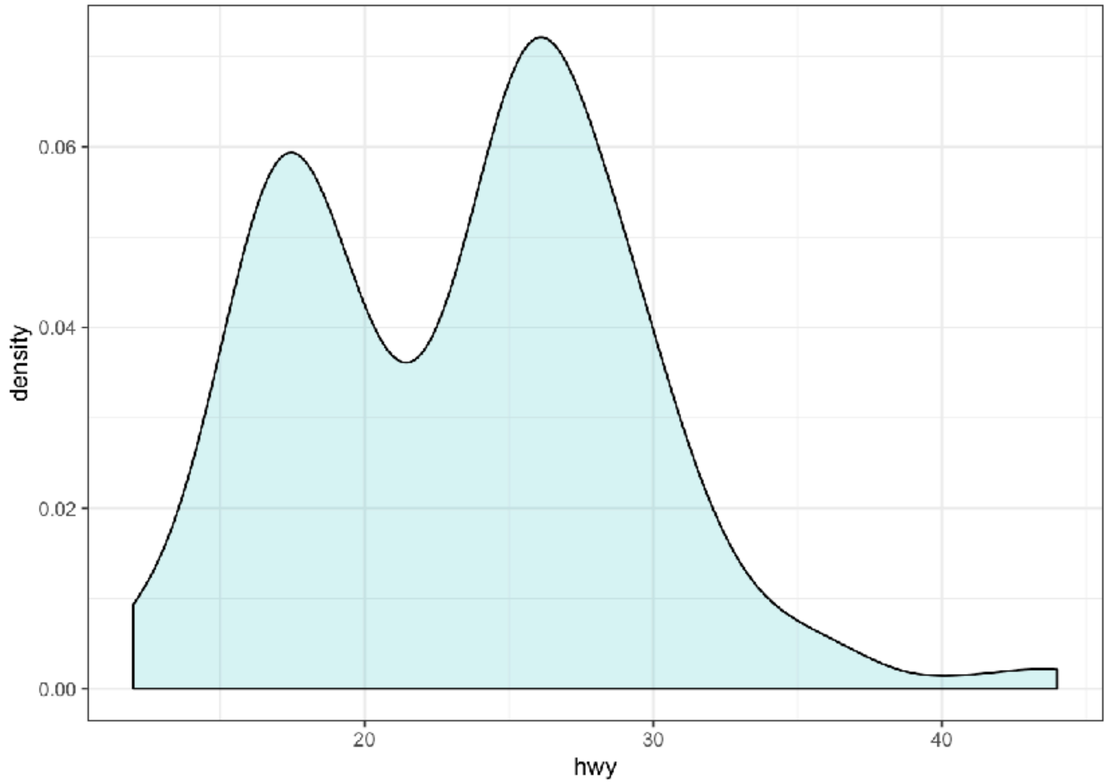
```
ggplot(data = diamonds) +
  stat_summary( mapping = aes(x = cut, y = depth),
                fun.ymin = min,
                fun.ymax = max,
                fun.y = median)
```

```
ggplot(data = diamonds) +
  geom_pointrange( mapping = aes(x = cut, y = depth),
                   stat = "summary",
                   fun.ymin = min,
                   fun.ymax = max,
                   fun.y = median)
```

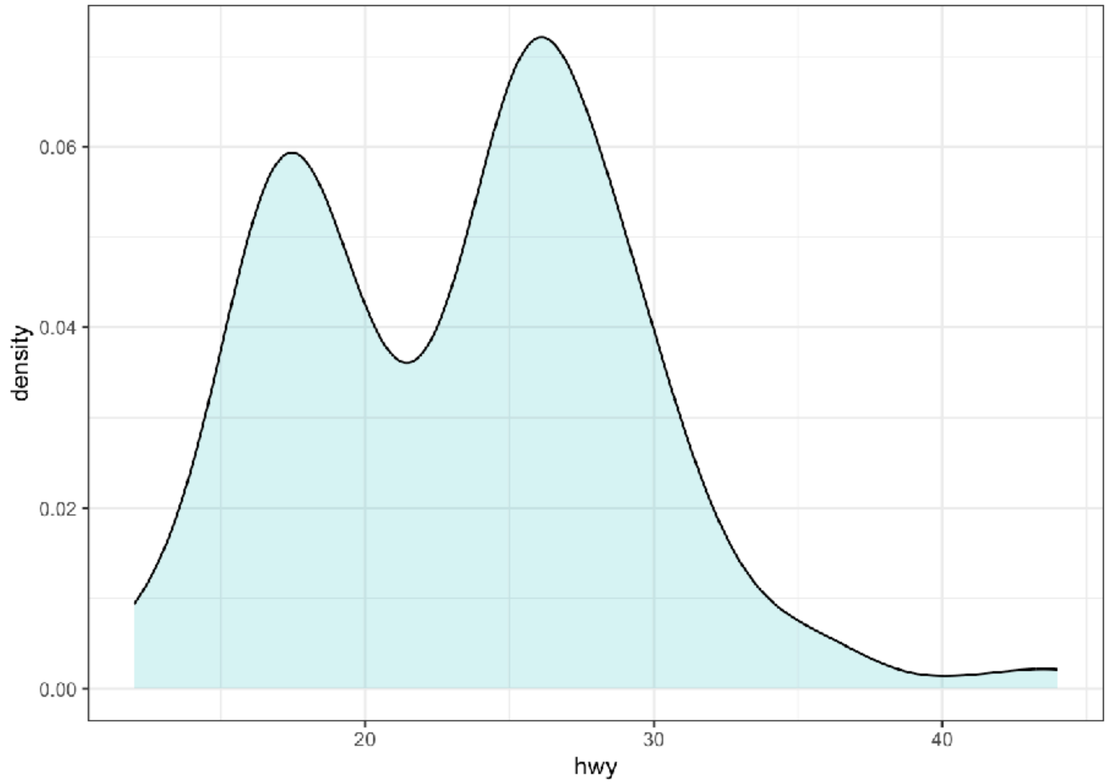


Styling tricks

```
ggplot(data = mpg, aes(x = hwy)) +
  geom_density(alpha = .2, fill= "#00BFC4") +
  theme_bw()
```




```
ggplot(data = mpg, aes(x = hwy)) +
  geom_density(alpha = .2, fill= "#00BFC4", color = 0) +
  geom_line(stat='density') +
  theme_bw()
```

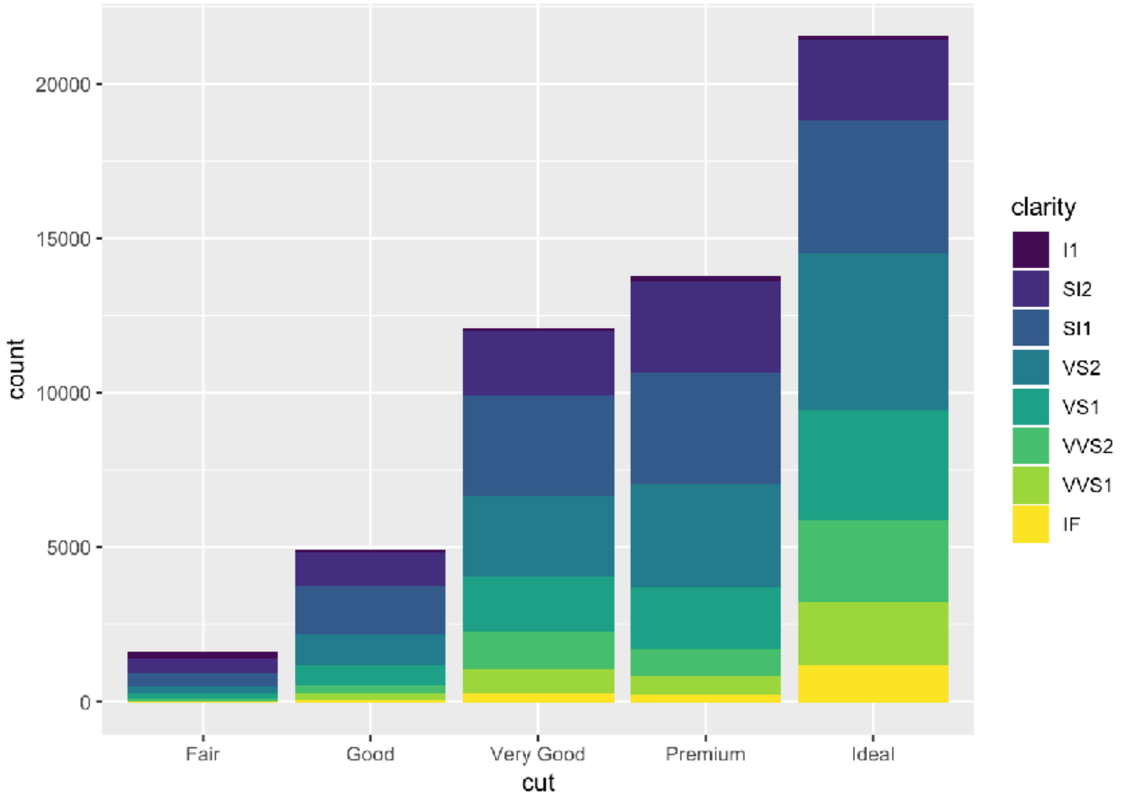


Position adjustments

Position adjustments

- The **`position`** argument specifies how the graphs are drawn, while **`stat`** defines the statistical transformation.

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

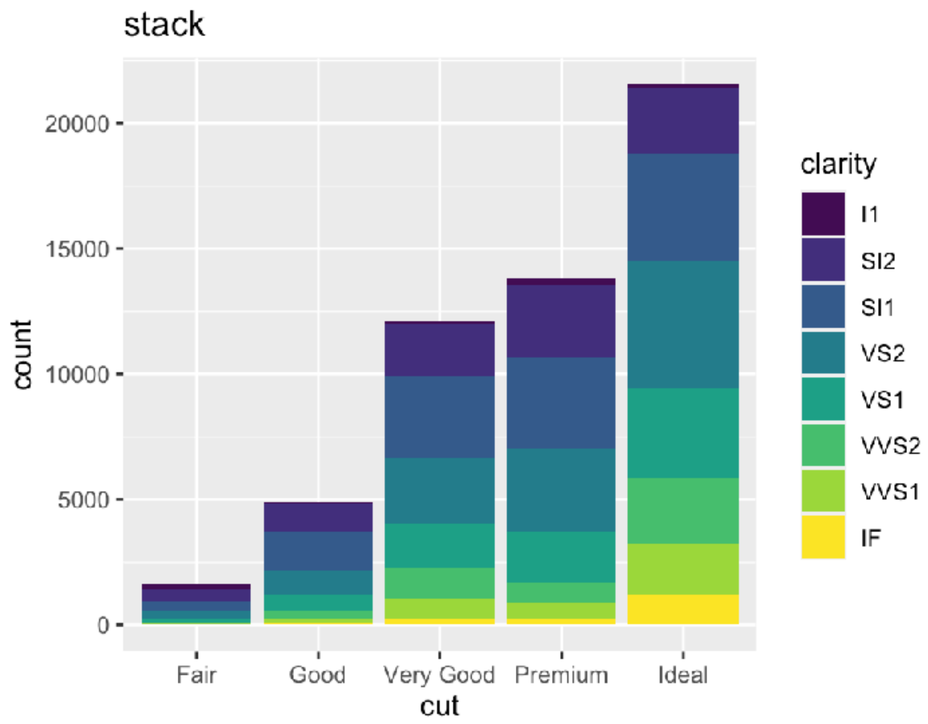


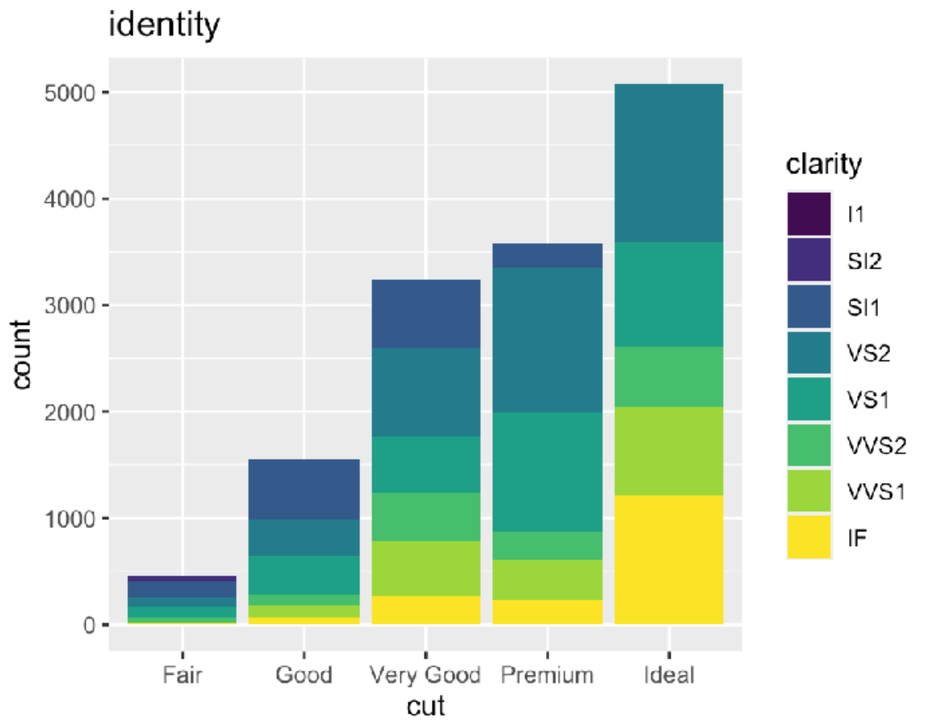
geom_bar

- 4 options:
    1. **`stack`** : default option to create a stacked barcharts
    2. **`identity`** : un-stacked, draws each object exactly where it falls in the context of the graph. This option is not very helpful as bars overlap
    3. **`dodge`** : avoids overlapping bars by placing beside one another
    4. **`fill`** : works like stack but visualise the proportions across groups

geom_bar

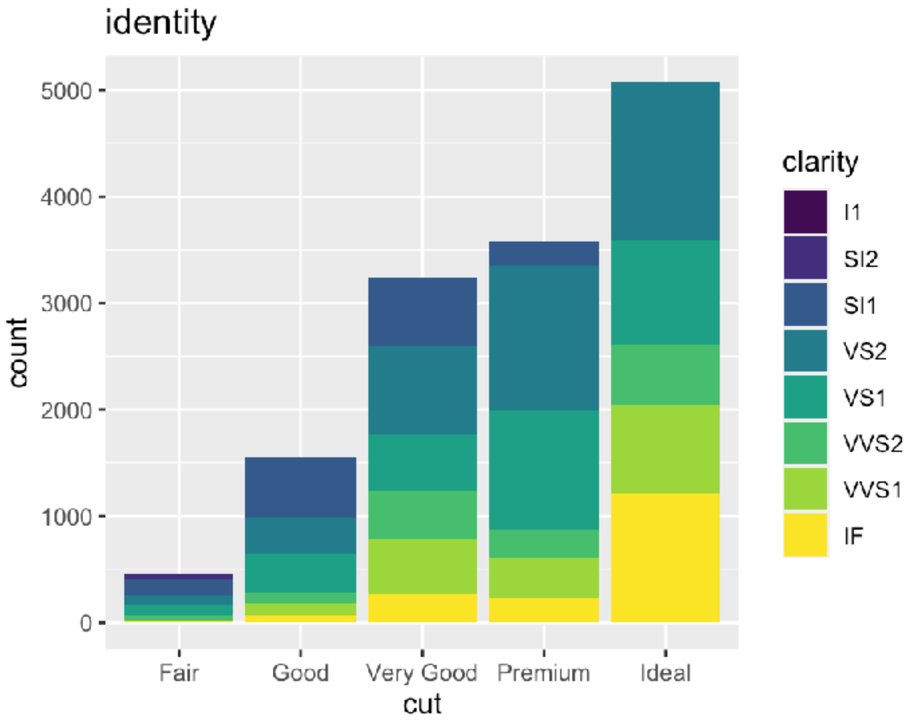
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "stack")

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

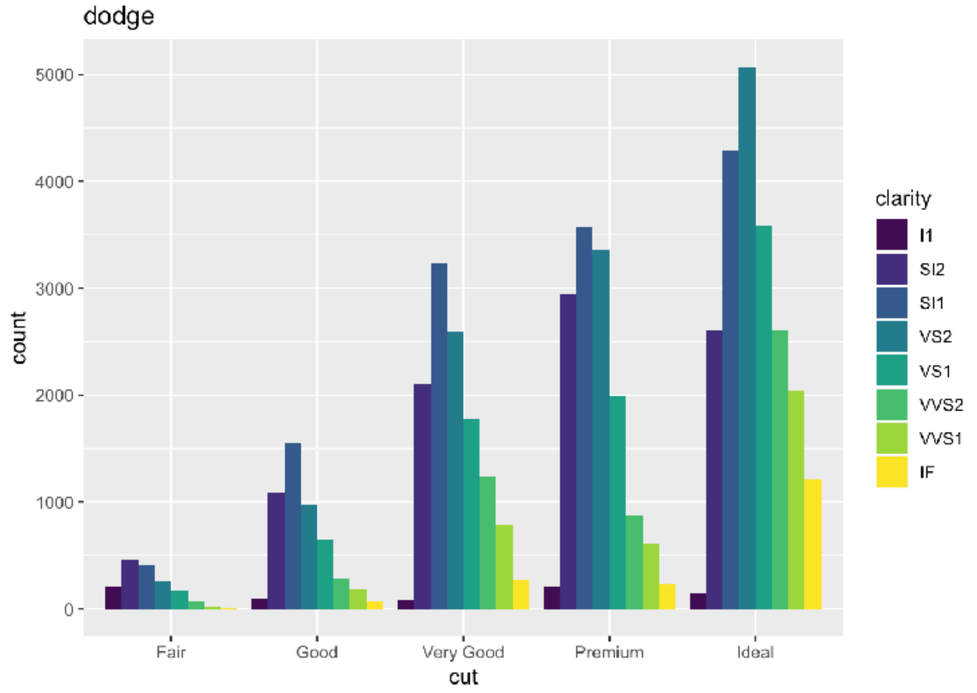




geom_bar

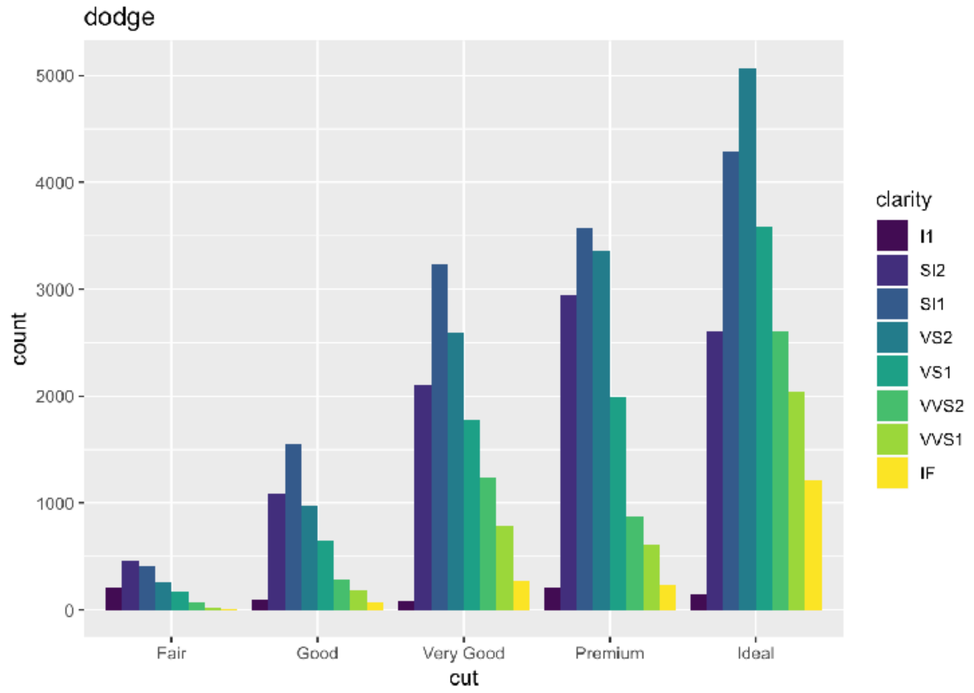
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "identity")

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

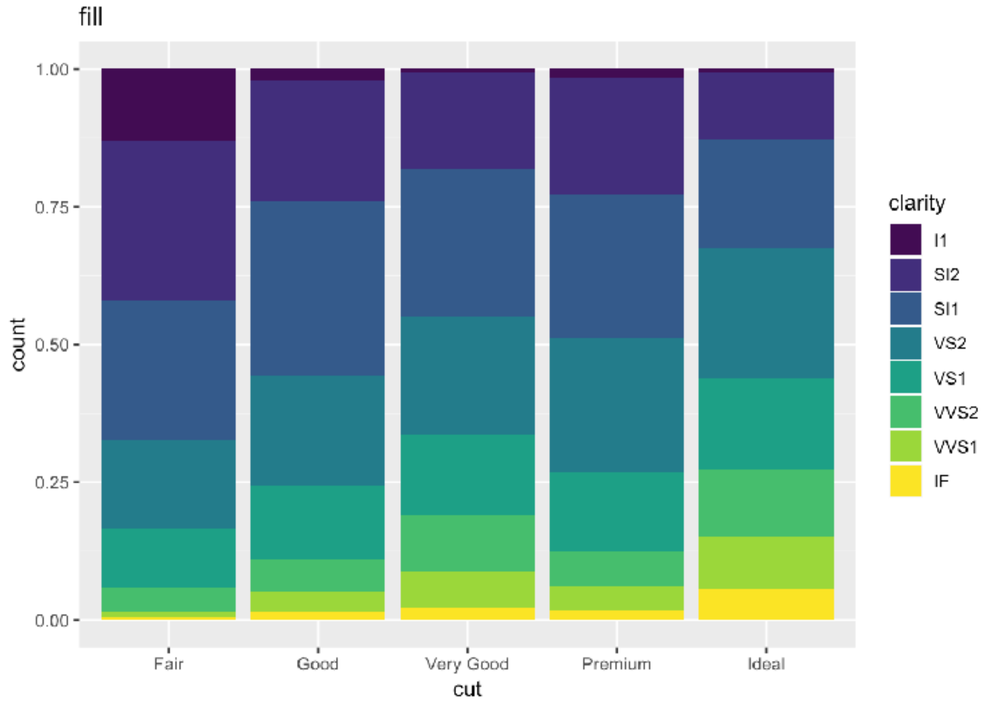




geom_bar

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

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

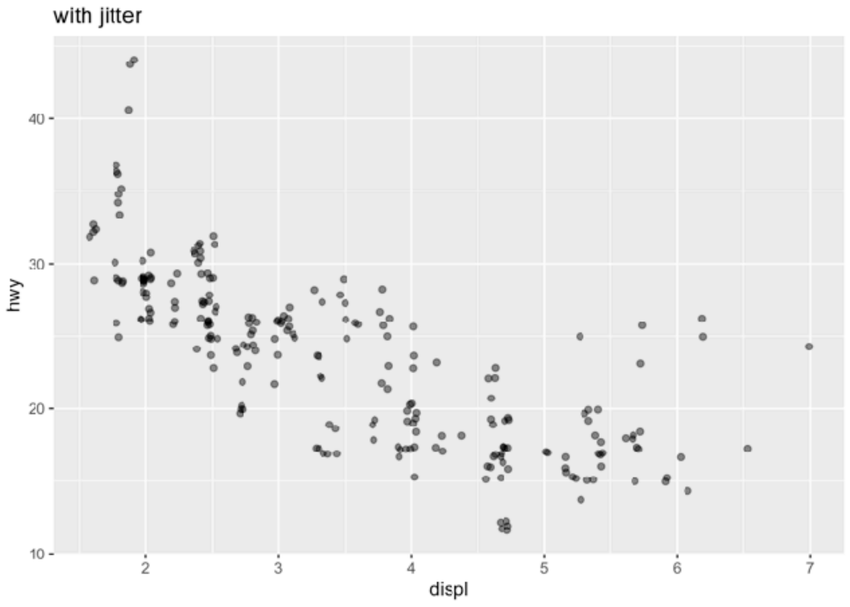


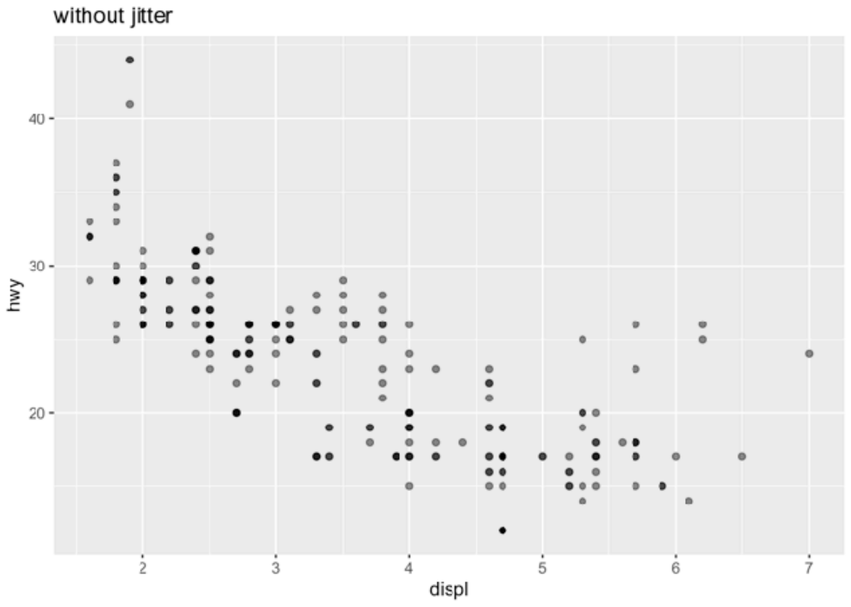


geom_jitter()

- avoid overplotting, e.g. scatter plot
  - adds small amount of random noise to each point to spread the overlapping points out

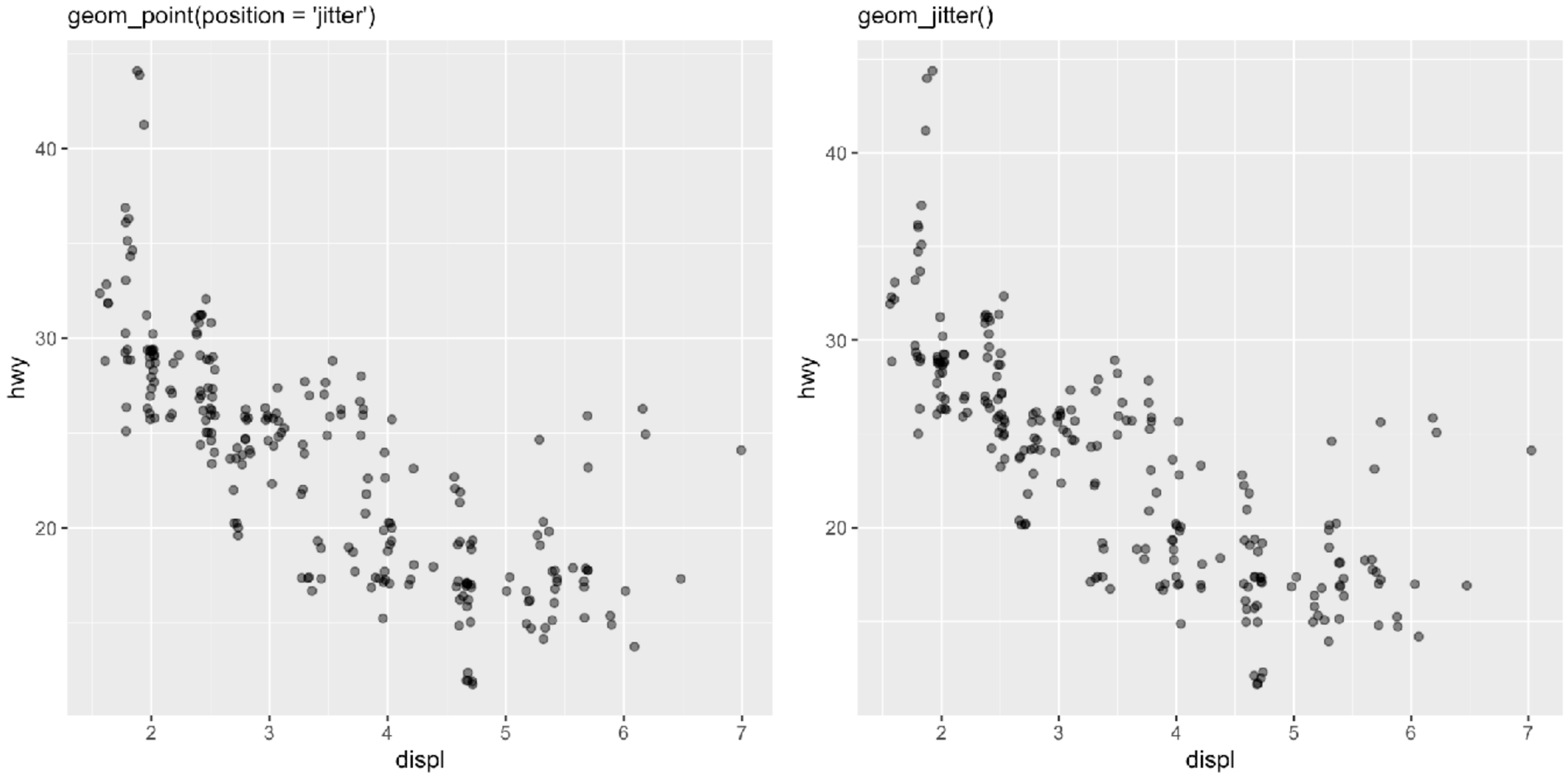
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), position = "jitter", alpha = 0.5)

ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), alpha = 0.5)
```





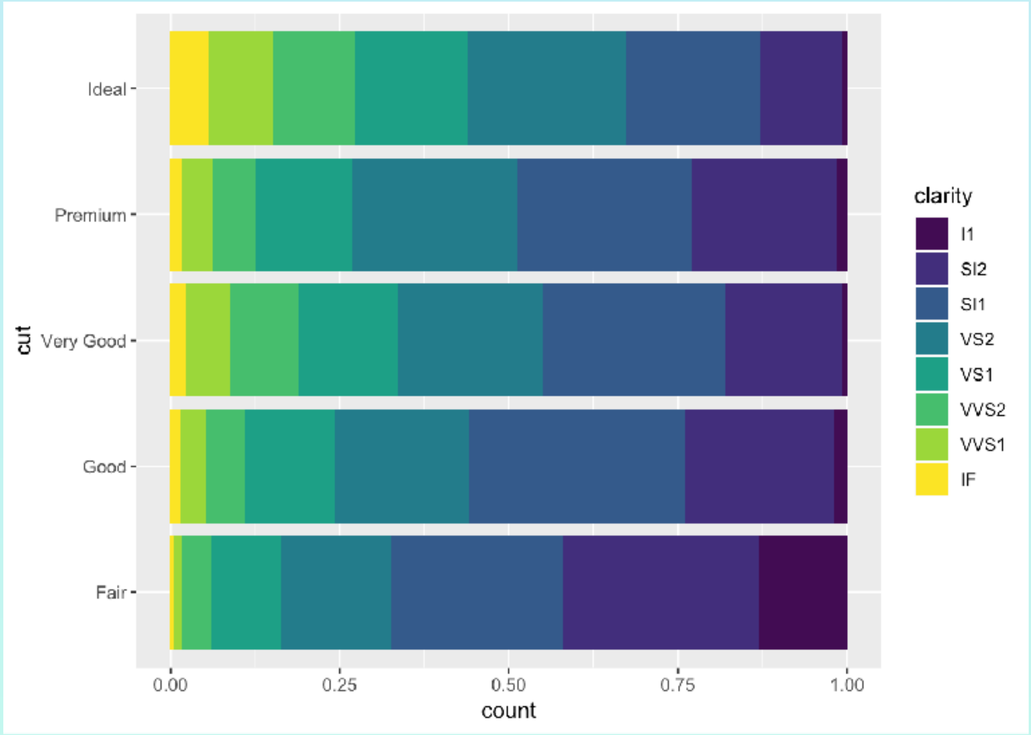
geom_jitter()

```
p1 <- ggplot(data = mpg) + geom_point(aes(x = displ, y = hwy), alpha = 0.5, position = "jitter")
p2 <- ggplot(data = mpg) + geom_jitter(aes(x = displ, y = hwy), alpha = 0.5)
grid.arrange(p1, p2, ncol = 2)
```



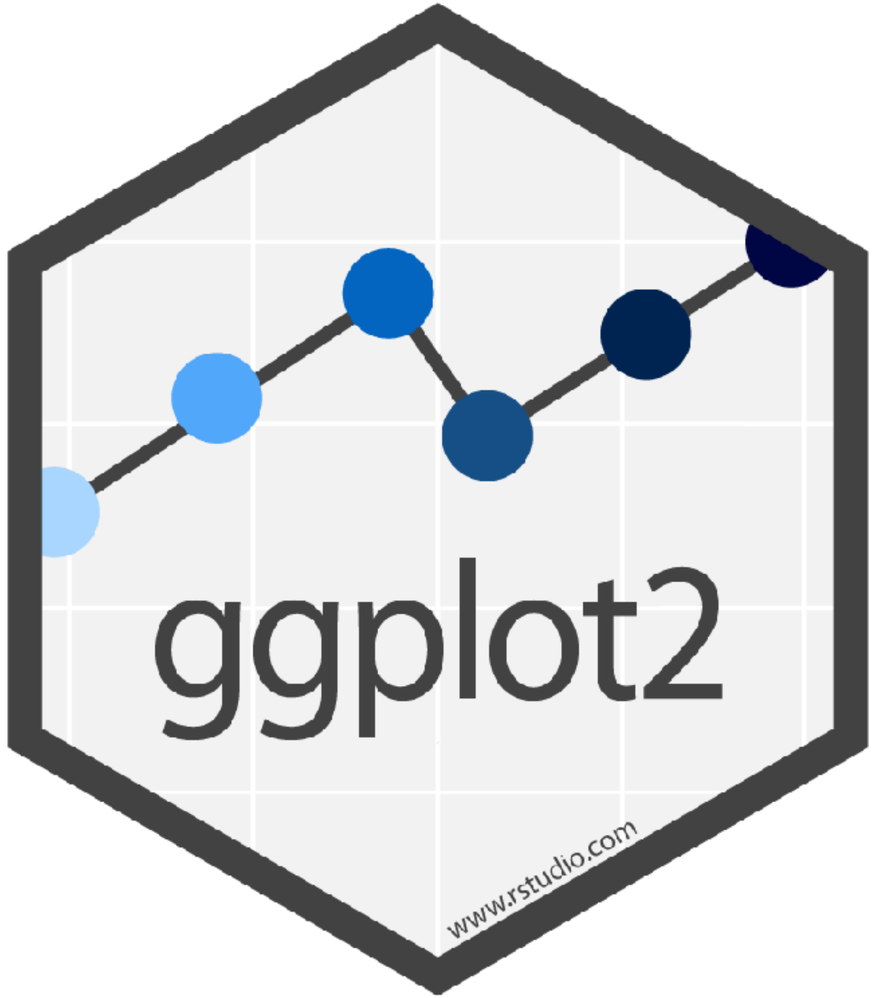
Your turn!

1. What parameters to **`geom_jitter()`** control the amount of displacement?
2. Try reproducing the following plot. (Hint: look up **`coord_flip()`**)



Lecture 7 - Summary

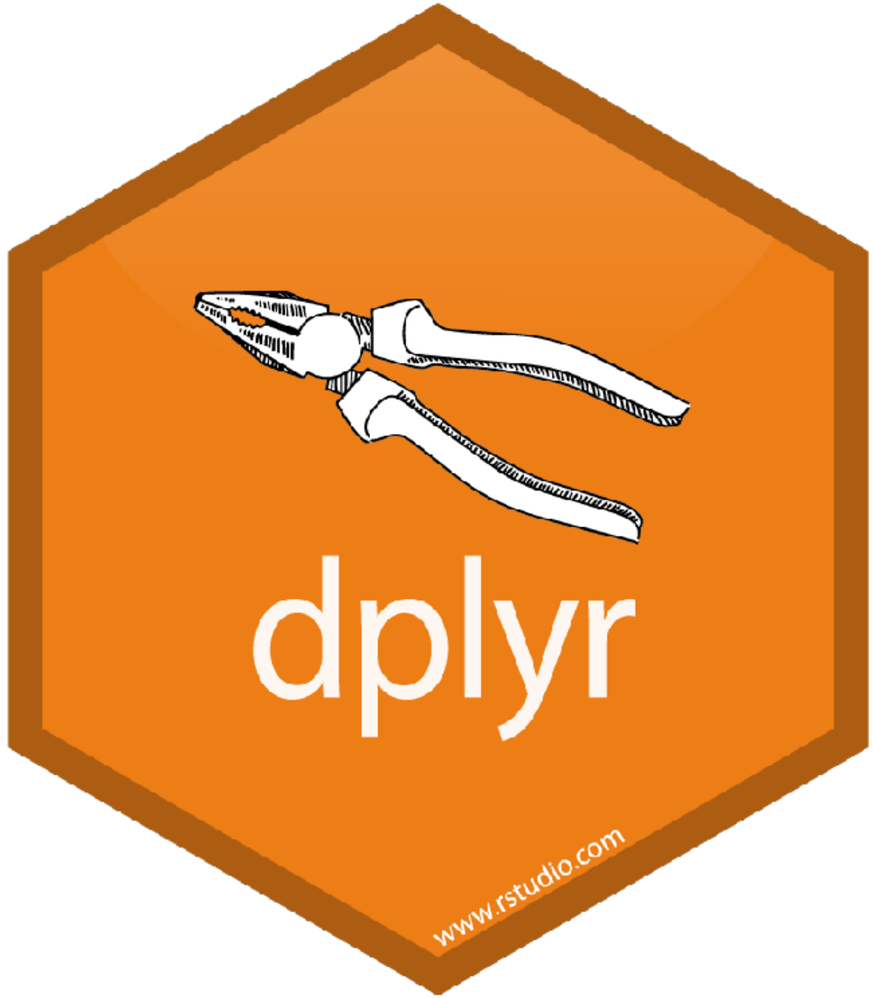
- Introduction to ggplot2
  - geometric objects
  - layered grammar of graphics
  - statistical transformation
  - position adjustment



Lecture 8 - Next lecture

- Data transformation
  - `dplyr` package
- Tidy data

- Scales
- Choropleth map



In-class exercise

- Instruction:
  - Go to Insendi and download the markdown:

  - Work together with your classmates in the breakout room
  - If you have a question, send a message to the instructor
    - You may be pulled out of breakout room if there is a common question
    - Also, check the forum to see answers to FAQs
  - Submit the HTML output indivually, via Insendi by **the next day 7am (UK time)!**
  - **You should now be able to resubmit within the deadline**