COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

Lecture 6: Data Visualization II

Data Science for Business Analytics

Thibault Vatter <thibault.vatter@unil.ch>

Department of Statistics, Columbia University and HEC Lausanne, UNIL

26.03.2018

Graphics for communications

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

Two weeks ago, you learned how to use plots as tools for *exploration*:

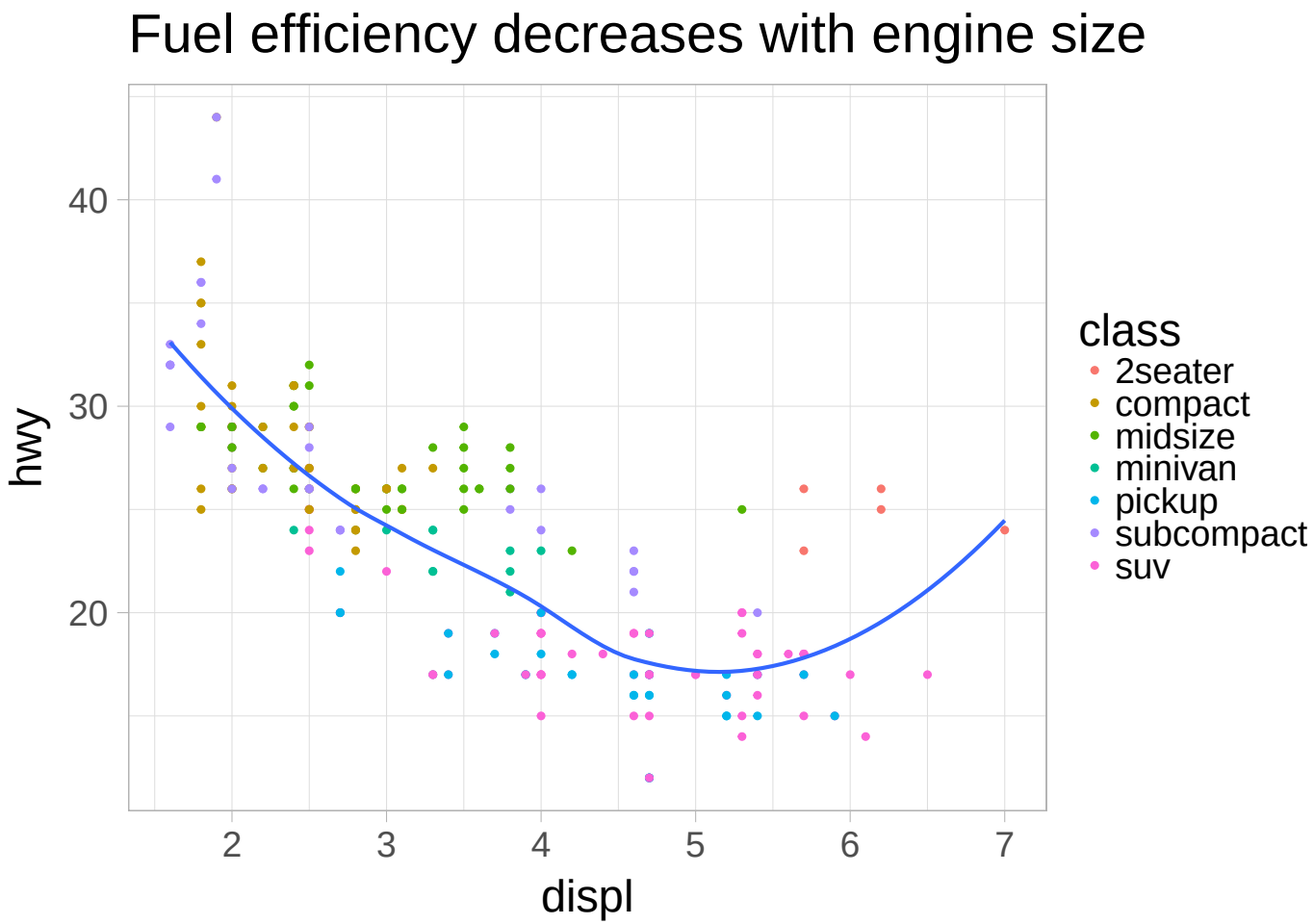
- With exploratory plots, you know—even before looking—which variables the plot will display.
- Each plot is made for a purpose, then is quickly looked at, and finally discarded the next plot.

Once you understand your data, you need to

- **communicate** your understanding to others,
- and to help do that quickly, make your plots self-explanatory.

Label

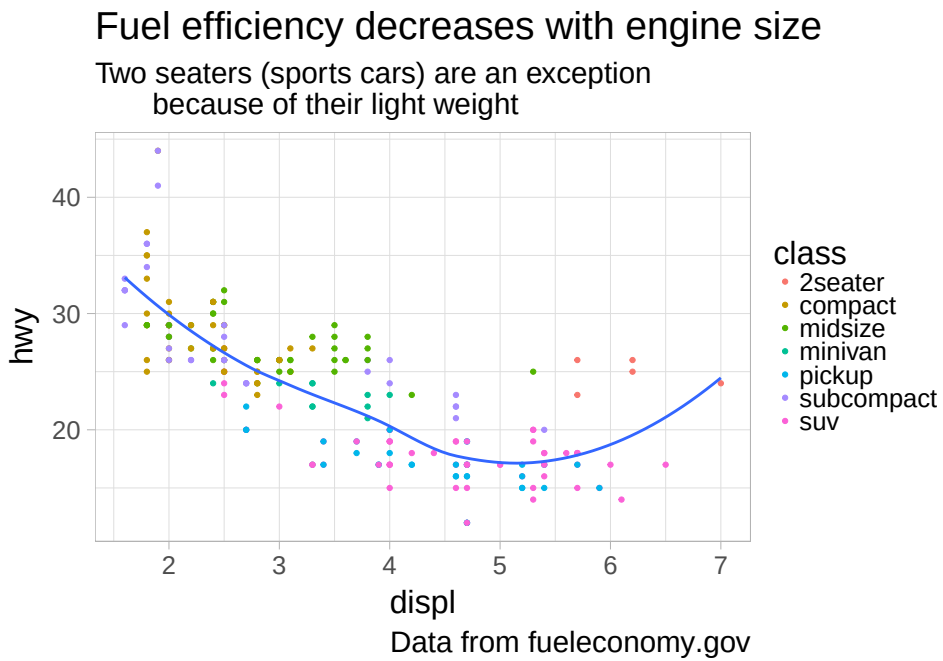

```
ggplot(mpg, aes(displ, hwy)) + geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  labs(title = "Fuel efficiency decreases with engine size")
```



More text

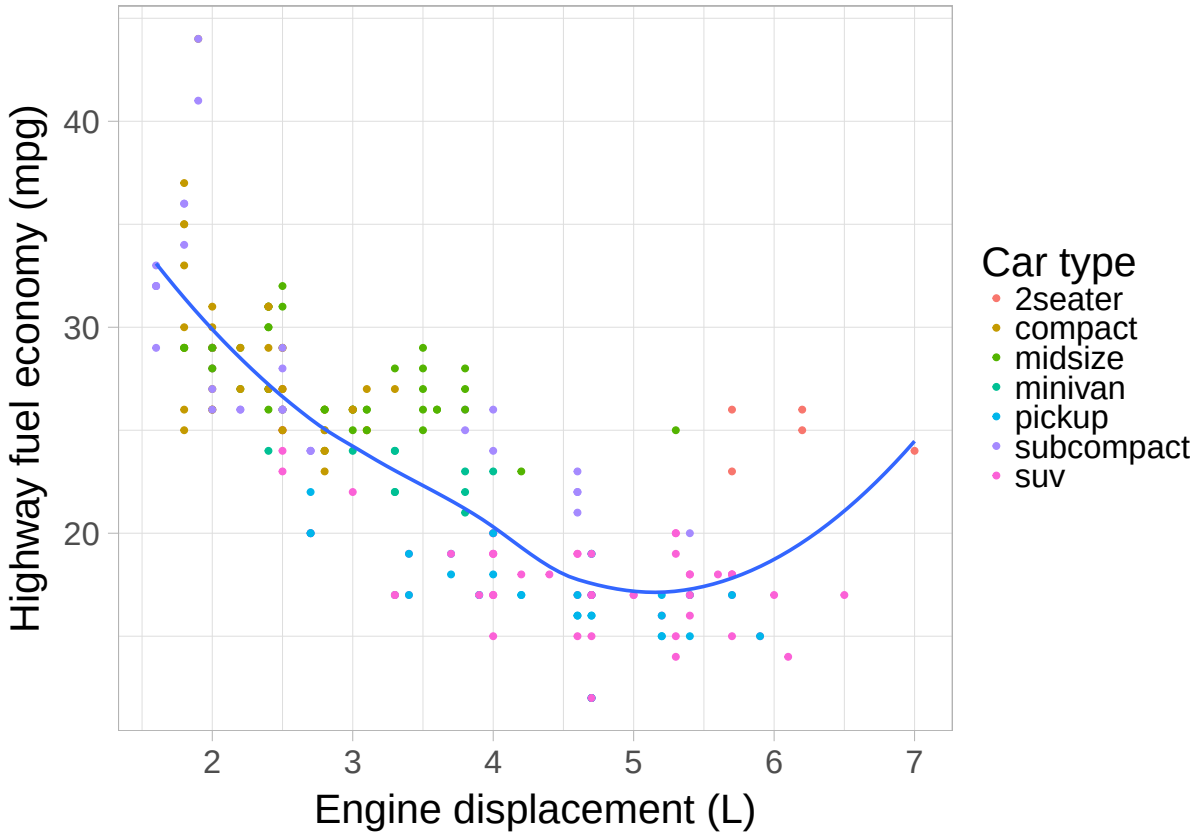
- `subtitle` adds additional detail beneath the title.
- `caption` adds text at the bottom right of the plot.

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) + geom_smooth(se = FALSE) +
  labs(title = "Fuel efficiency decreases with engine size",
       subtitle = "Two seaters (sports cars) are an exception
       because of their light weight",
       caption = "Data from fueleconomy.gov")
```



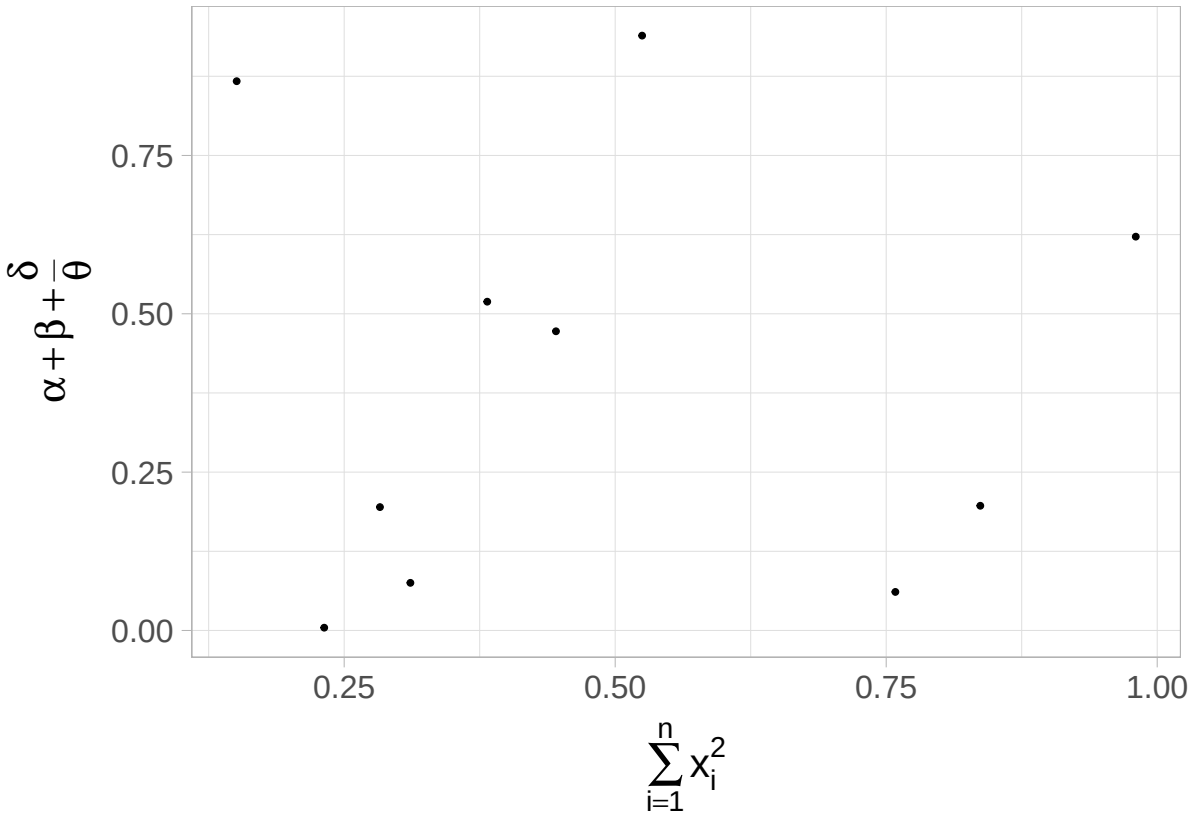
Axis


```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class)) +
  geom_smooth(se = FALSE) +
  labs(x = "Engine displacement (L)",
       y = "Highway fuel economy (mpg)",
       colour = "Car type")
```



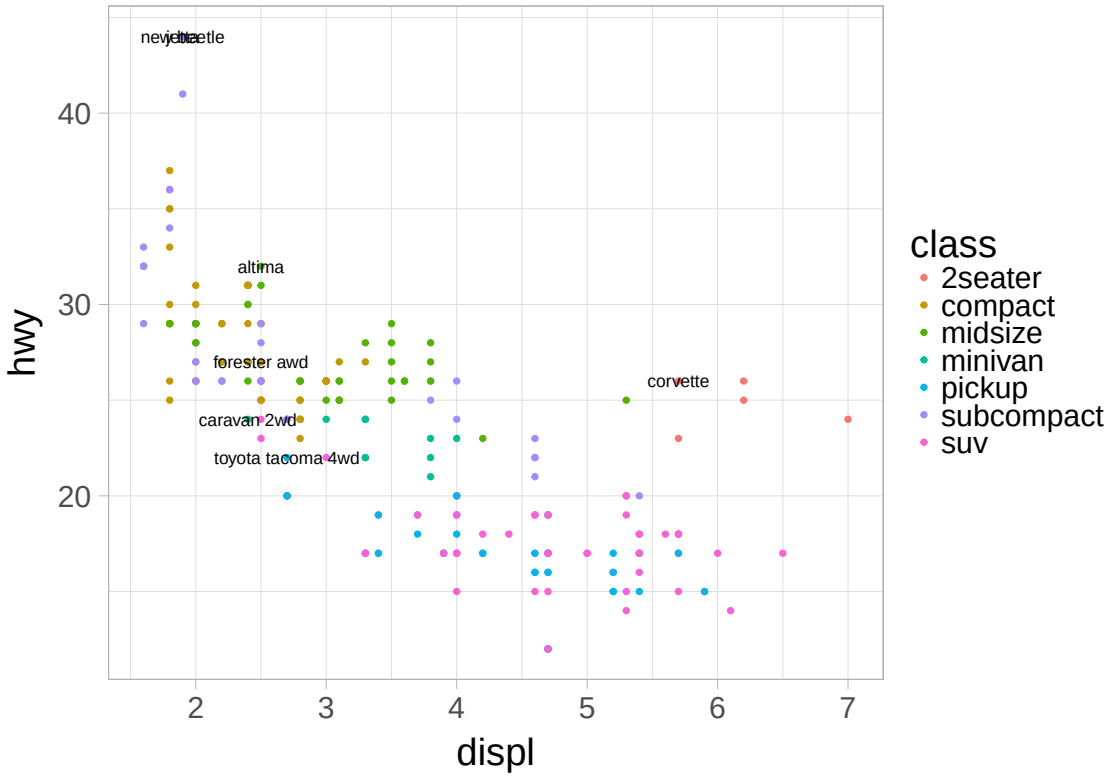
Mathematical expressions


```
df <- tibble(x = runif(10),
             y = runif(10))
ggplot(df, aes(x, y)) + geom_point() +
  labs(x = quote(sum(x[i] ^ 2, i == 1, n)),
       y = quote(alpha + beta + frac(delta, theta)))
```



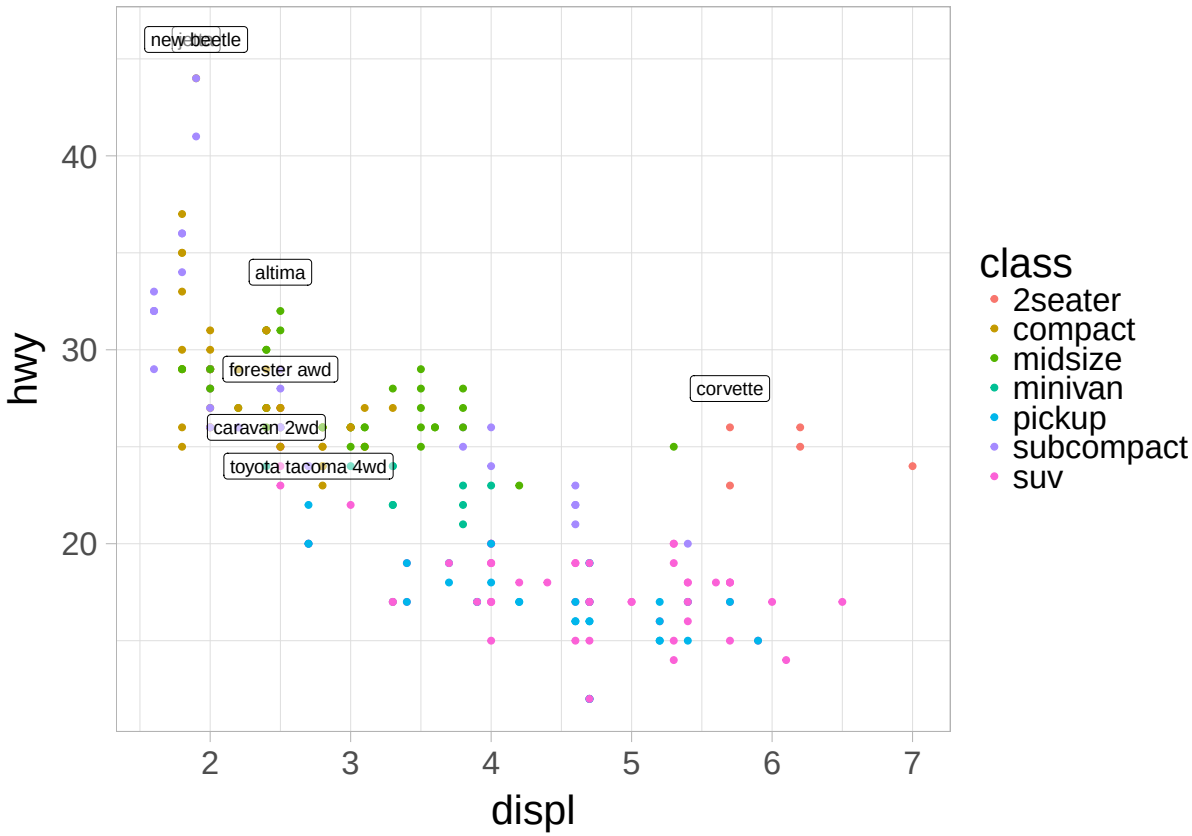
Annotations


```
best_in_class <- mpg %>%
  group_by(class) %>%
  filter(row_number(desc(hwy)) == 1)

ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class)) +
  geom_text(aes(label = model), data = best_in_class)
```



Nudge annotations

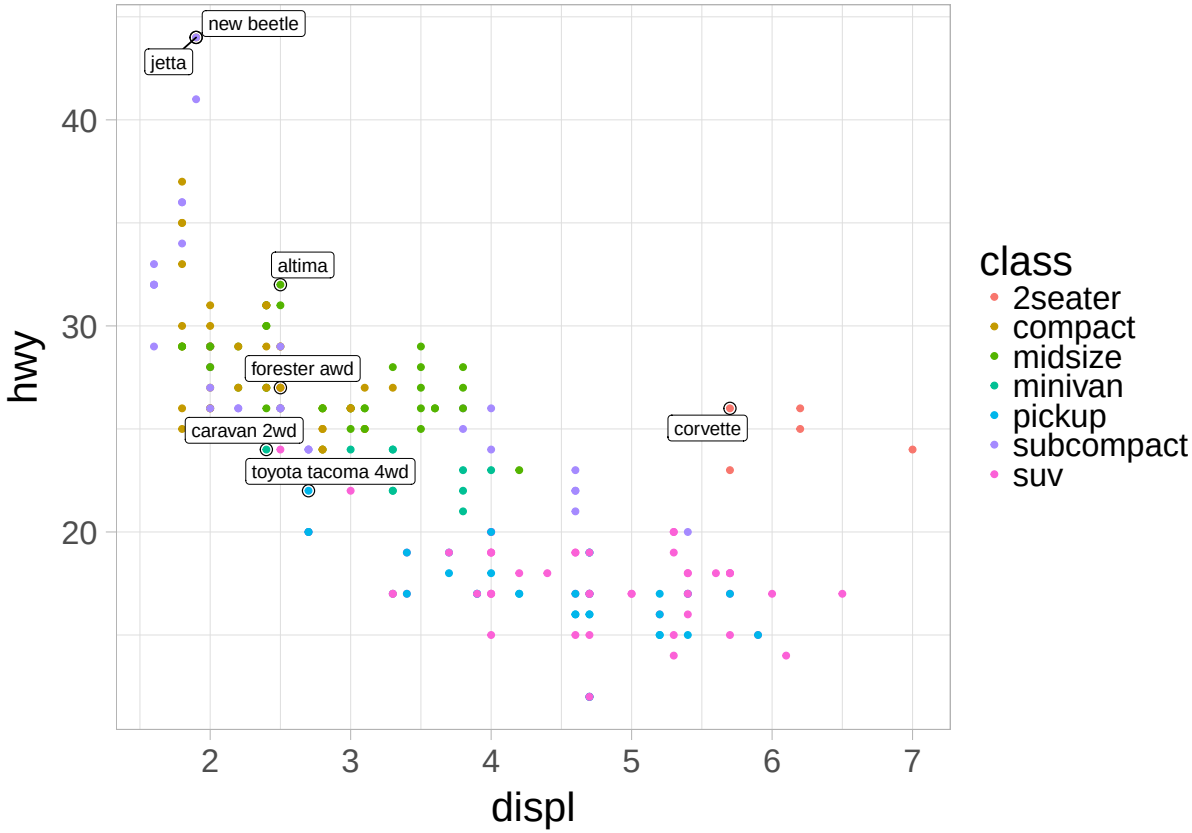

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class)) +
  geom_label(aes(label = model), data = best_in_class,
             nudge_y = 2, alpha = 0.5)
```



Or better

The **ggrepel** package by Kamil Slowikowski:

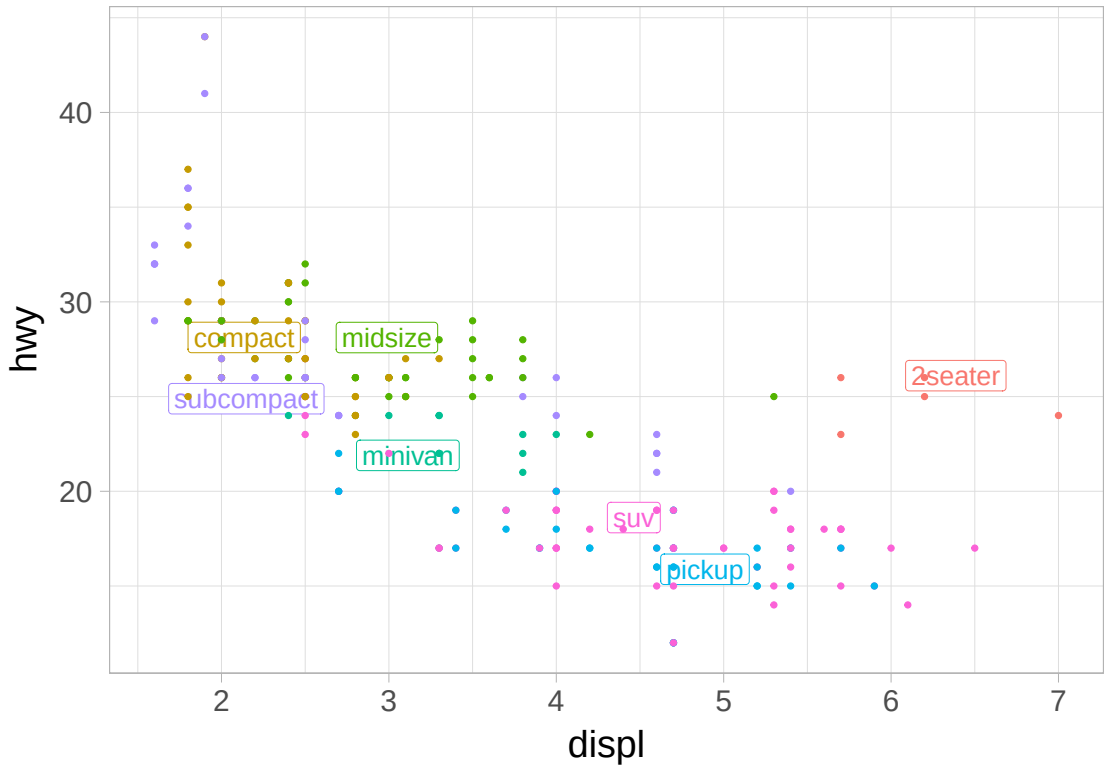
```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class)) +
  geom_point(size = 3, shape = 1, data = best_in_class) +
  ggrepel::geom_label_repel(aes(label = model), data = best_in_class)
```



Replace legend by labels on the plot

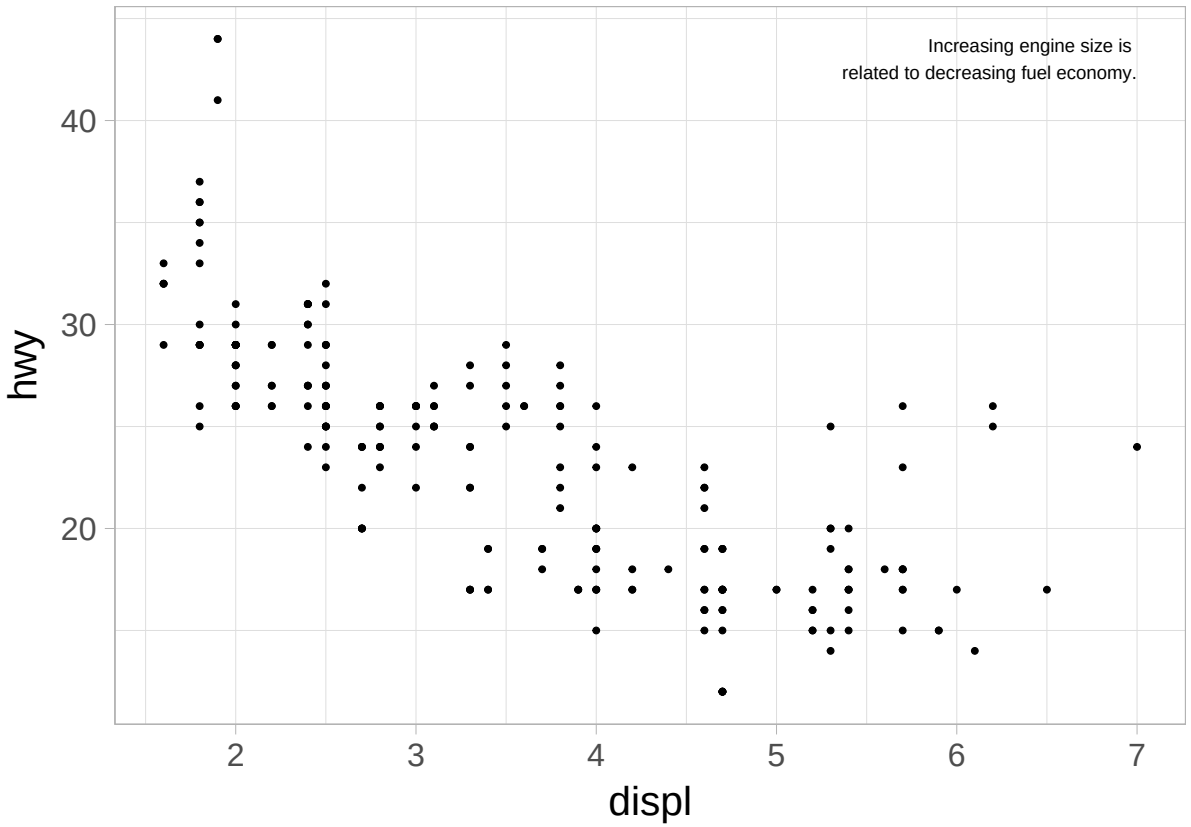
COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
class_avg <- mpg %>% group_by(class) %>%
  summarise(displ = median(displ), hwy = median(hwy))

ggplot(mpg, aes(displ, hwy, colour = class)) +
  ggrepel::geom_label_repel(aes(label = class), data = class_avg, size = 6,
                            label.size = 0, segment.color = NA) +
  geom_point() + theme(legend.position = "none")
```



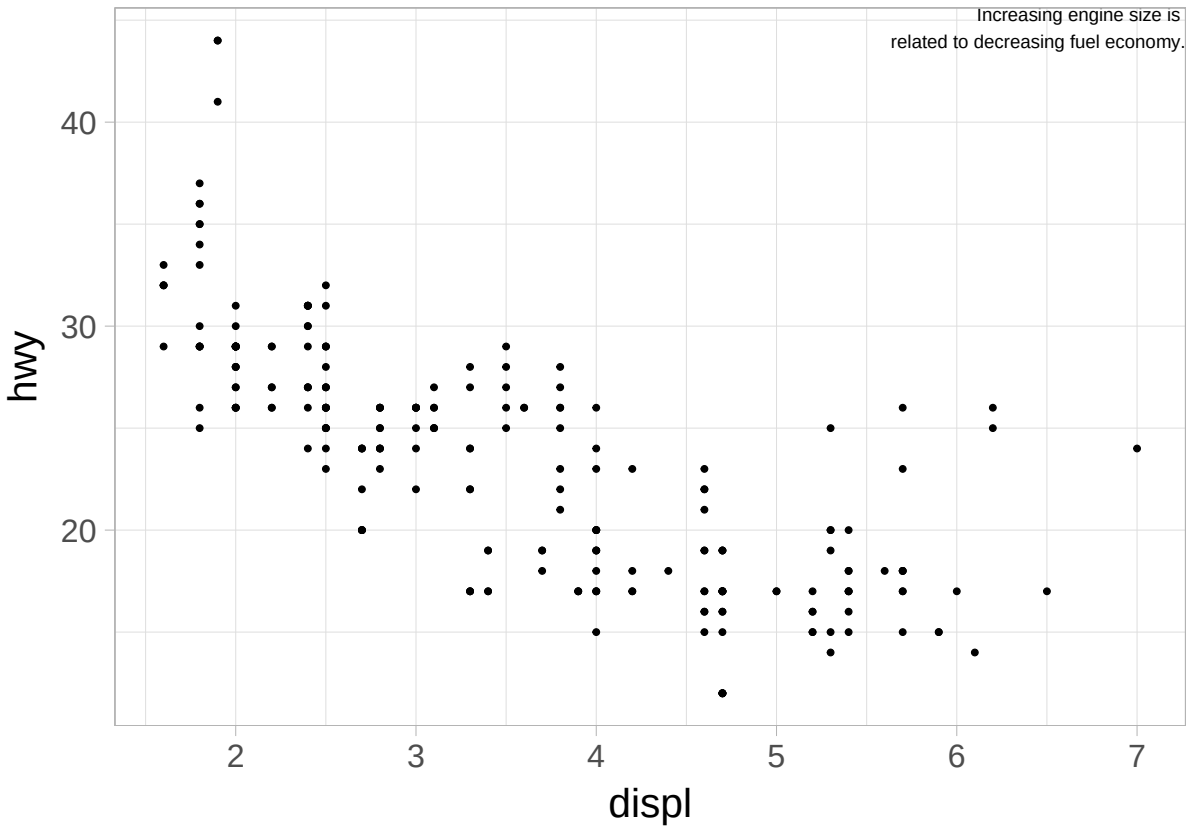
To add a single label to the plot


```
label <- mpg %>%
  summarise(displ = max(displ), hwy = max(hwy),
label = "Increasing engine size is \nrelated to decreasing fuel economy.")

ggplot(mpg, aes(displ, hwy)) + geom_point() +
  geom_text(aes(label = label), data = label, vjust = "top", hjust = "right")
```



An alternative


```
label <- tibble(displ = Inf, hwy = Inf,
label = "Increasing engine size is \nrelated to decreasing fuel economy.")

ggplot(mpg, aes(displ, hwy)) + geom_point() +
  geom_text(aes(label = label), data = label,
            vjust = "top", hjust = "right")
```



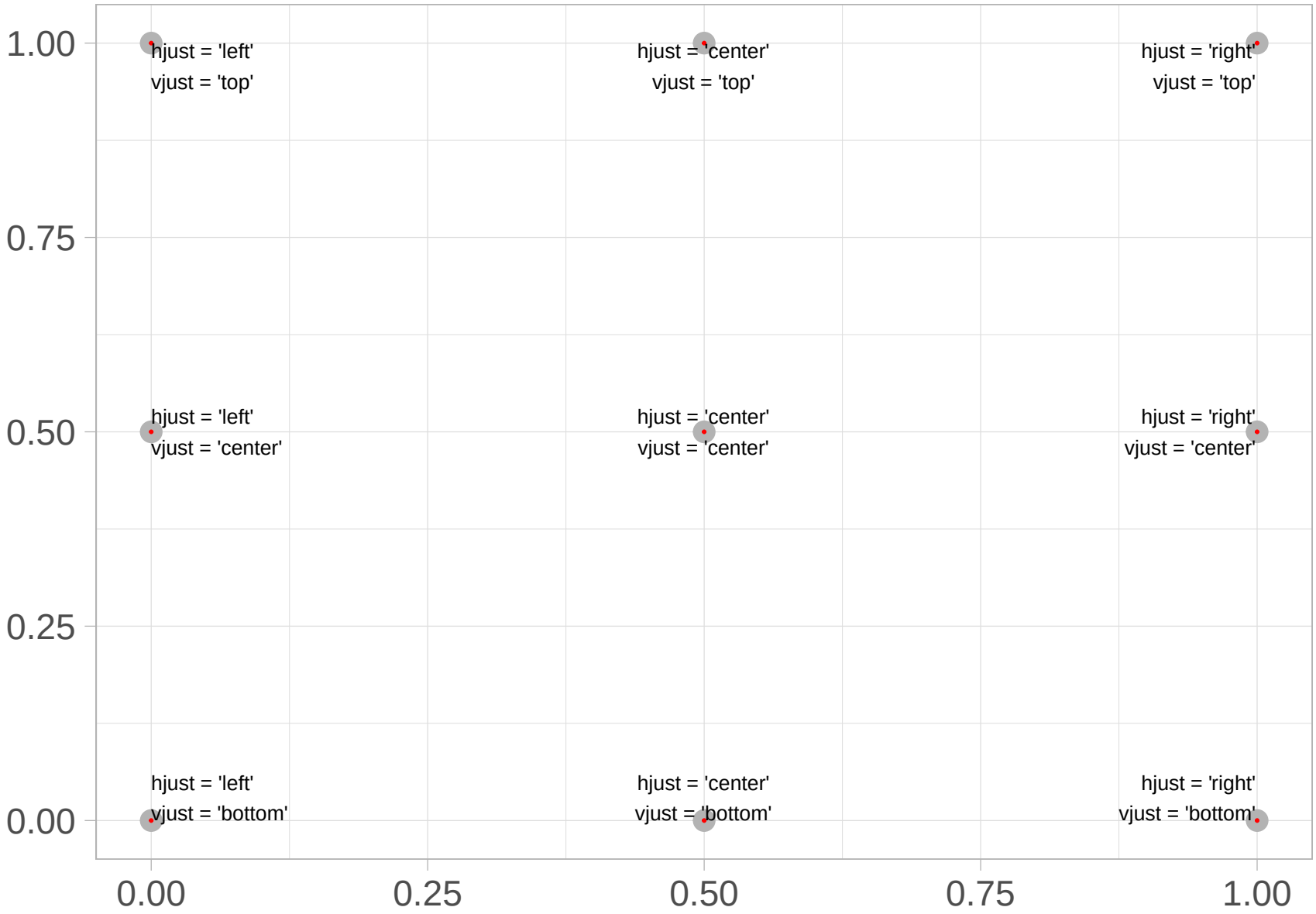
To automatically add line breaks


```
"Increasing engine size is related to decreasing fuel economy." %>%
  stringr::str_wrap(width = 40) %>%
  writeLines()
```

```
## Increasing engine size is related to
## decreasing fuel economy.
```

To control the alignment of the label

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK



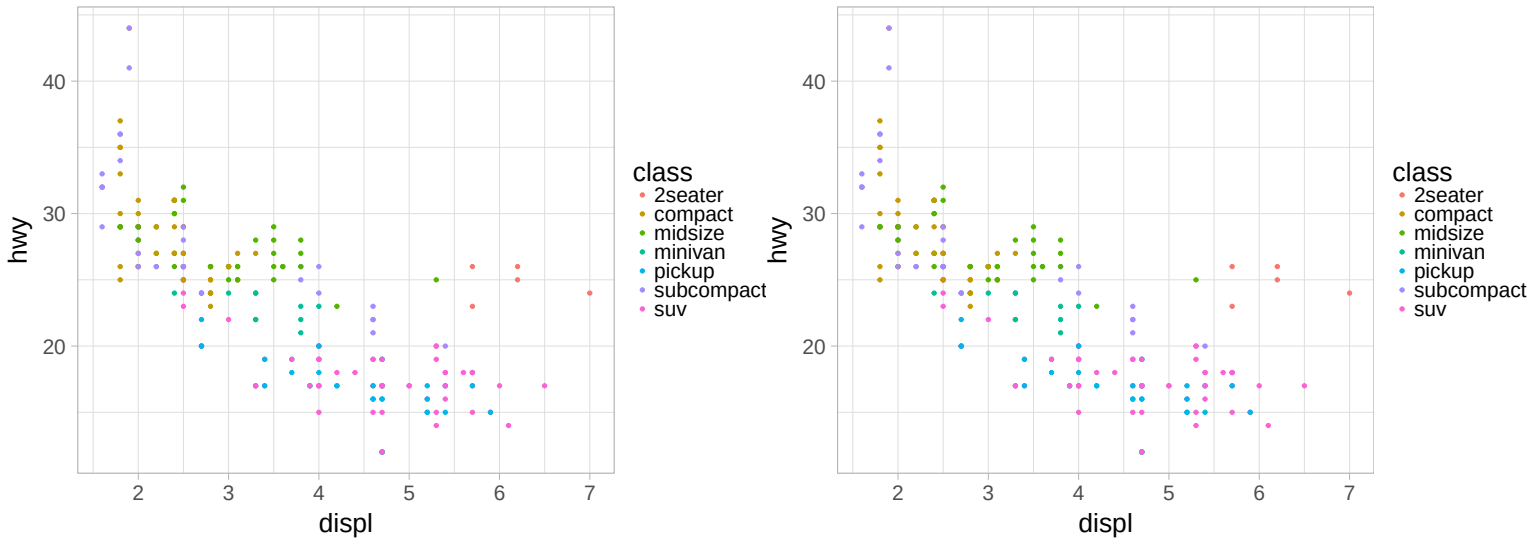
Geoms to help annotate your plot

- `geom_hline()` and `geom_vline()` to add reference lines (using e.g. `size = 2` is often a good idea)
- `geom_rect()` to draw a rectangle around points of interest (boundaries by `xmin`, `xmax`, `ymin`, `ymax`).
- `geom_segment()` with the `arrow` argument to draw attention to a point with an arrow (`x` and `y` define the starting location, and `xend` and `yend` define the end location).

The only limit is your imagination (and your patience)!

Scales


```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class))

ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class)) +
  scale_x_continuous() +
  scale_y_continuous() +
  scale_colour_discrete()
```

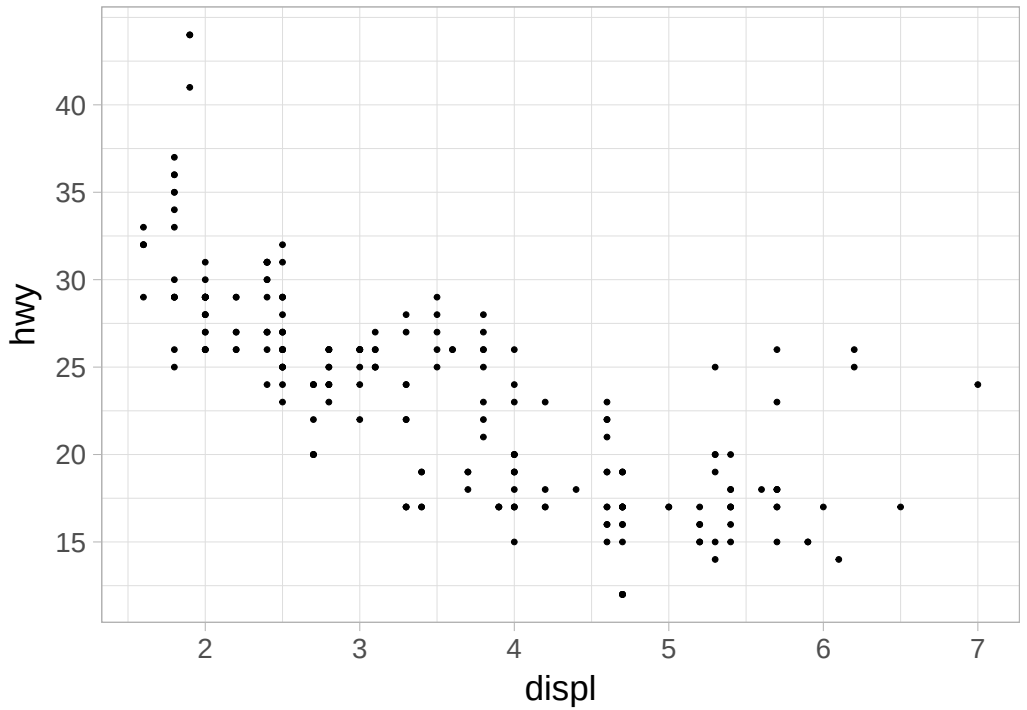


Axis ticks and legend keys

To control the ticks on the axes and the keys on the legend:

- `breaks` controls the position of the ticks, or the values associated with the keys.
- `labels` controls the text label associated with each tick/key.

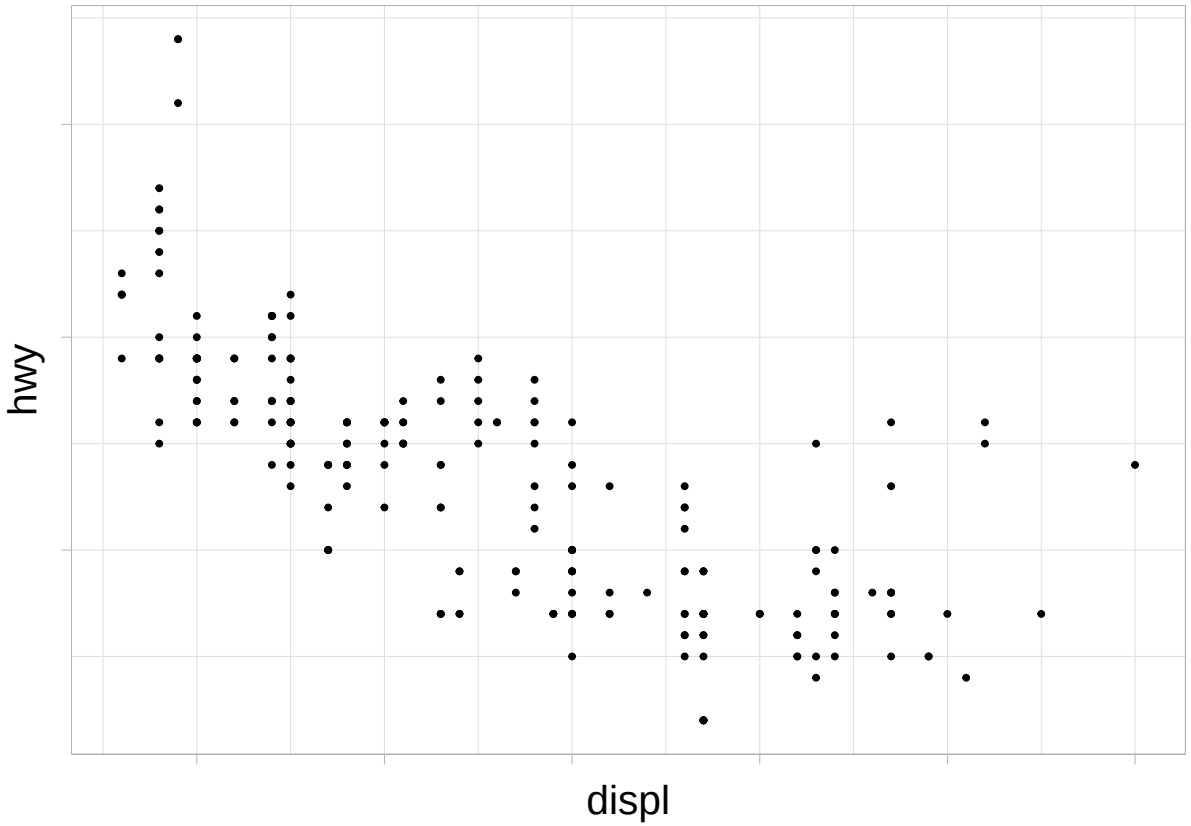
```
ggplot(mpg, aes(displ, hwy)) + geom_point() +
  scale_y_continuous(breaks = seq(15, 40, by = 5))
```



Axis ticks and legend keys II

A useful trick for maps, or for publishing plots where you can't share the absolute numbers:

```
ggplot(mpg, aes(displ, hwy)) + geom_point() +
  scale_x_continuous(labels = NULL) +
  scale_y_continuous(labels = NULL)
```



Two remarks

1. Collectively axes and legends are called **guides**:

   - Axes are used for x and y aesthetics;
   - legends are used for everything else.

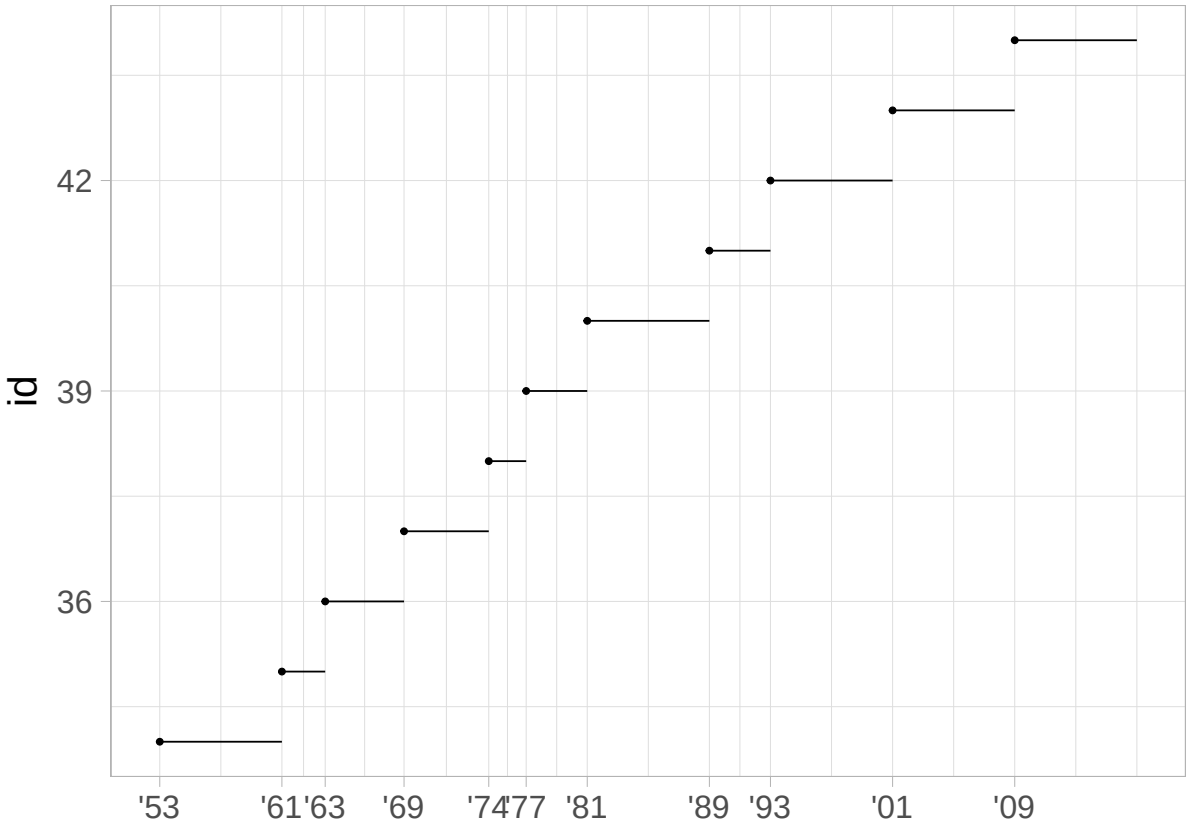
Hence, you can also use `breaks` and `labels` to control the appearance of legends.

2. Breaks and labels for date and datetime scales is a little different:

   - `date_labels` takes a format specification, in the same form as `parse_datetime()`.
   - `date_breaks` (not shown here), takes a string like "2 days" or "1 month".

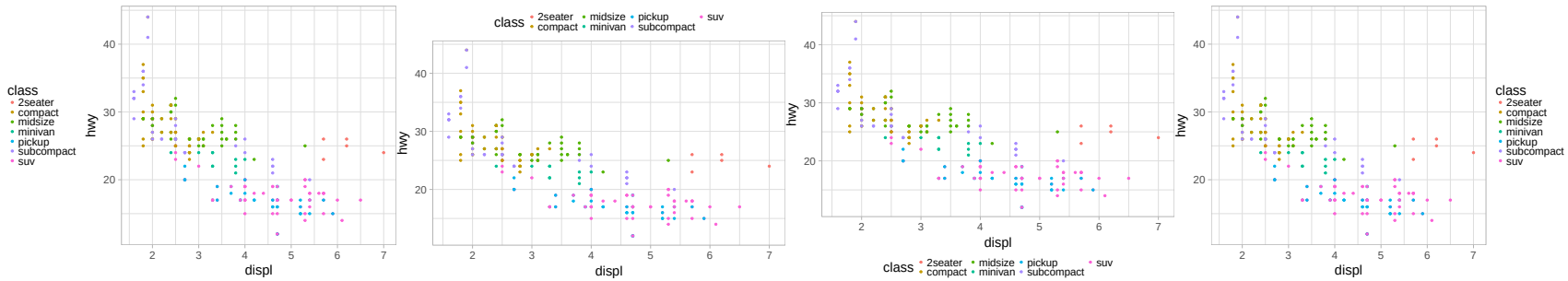
`breaks` highlighting exactly where the observations occur


```
presidential %>%
  mutate(id = 33 + row_number()) %>%
  ggplot(aes(start, id)) + geom_point() +
    geom_segment(aes(xend = end, yend = id)) +
    scale_x_date(NULL, breaks = presidential$start, date_labels = "'%y")
```



Legend layout


```
base <- ggplot(mpg, aes(displ, hwy)) + geom_point(aes(colour = class))

base + theme(legend.position = "left")
base + theme(legend.position = "top")
base + theme(legend.position = "bottom")
base + theme(legend.position = "right") # the default
```

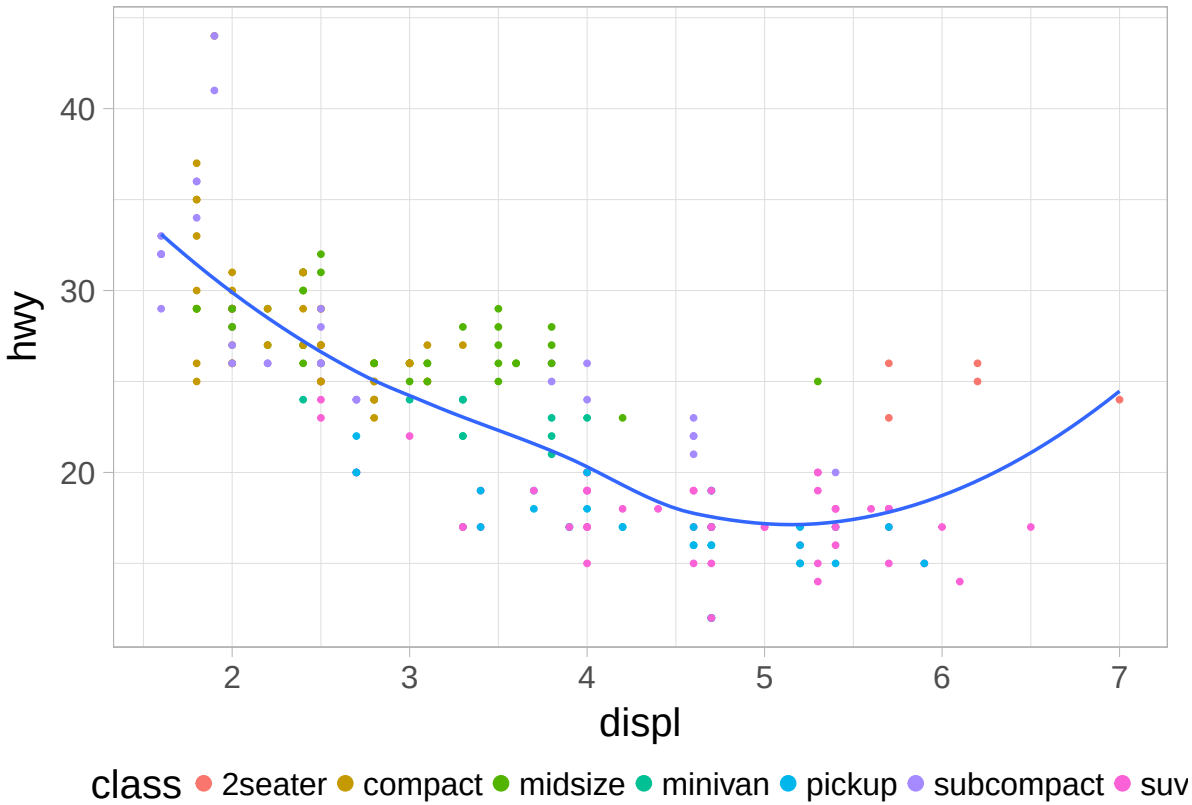


`legend.position = "none"` suppresses the display of the legend!

To control individual legends

Use guides(), guide_legend() or guide_colourbar():

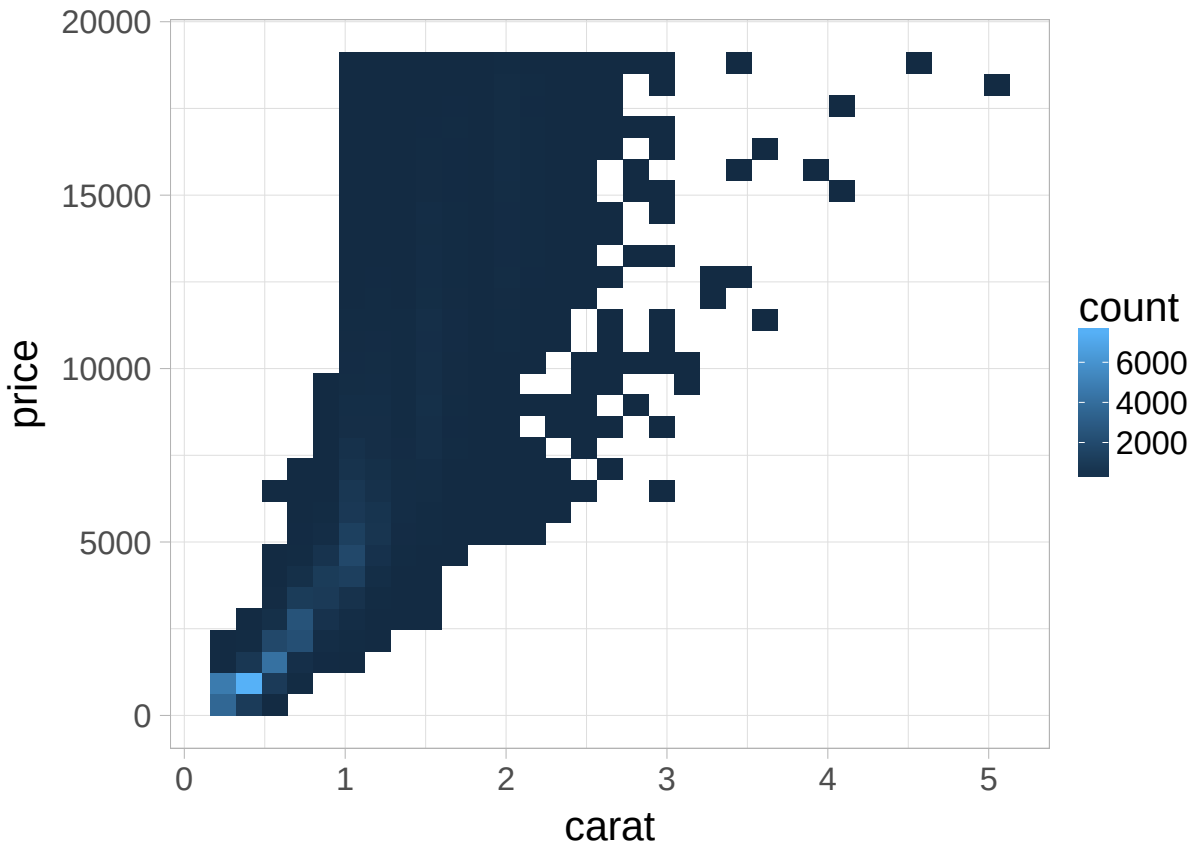
```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class)) +
  geom_smooth(se = FALSE) +
  theme(legend.position = "bottom") +
  guides(colour = guide_legend(nrow = 1, override.aes = list(size = 4)))
```



How could we improve the scale?

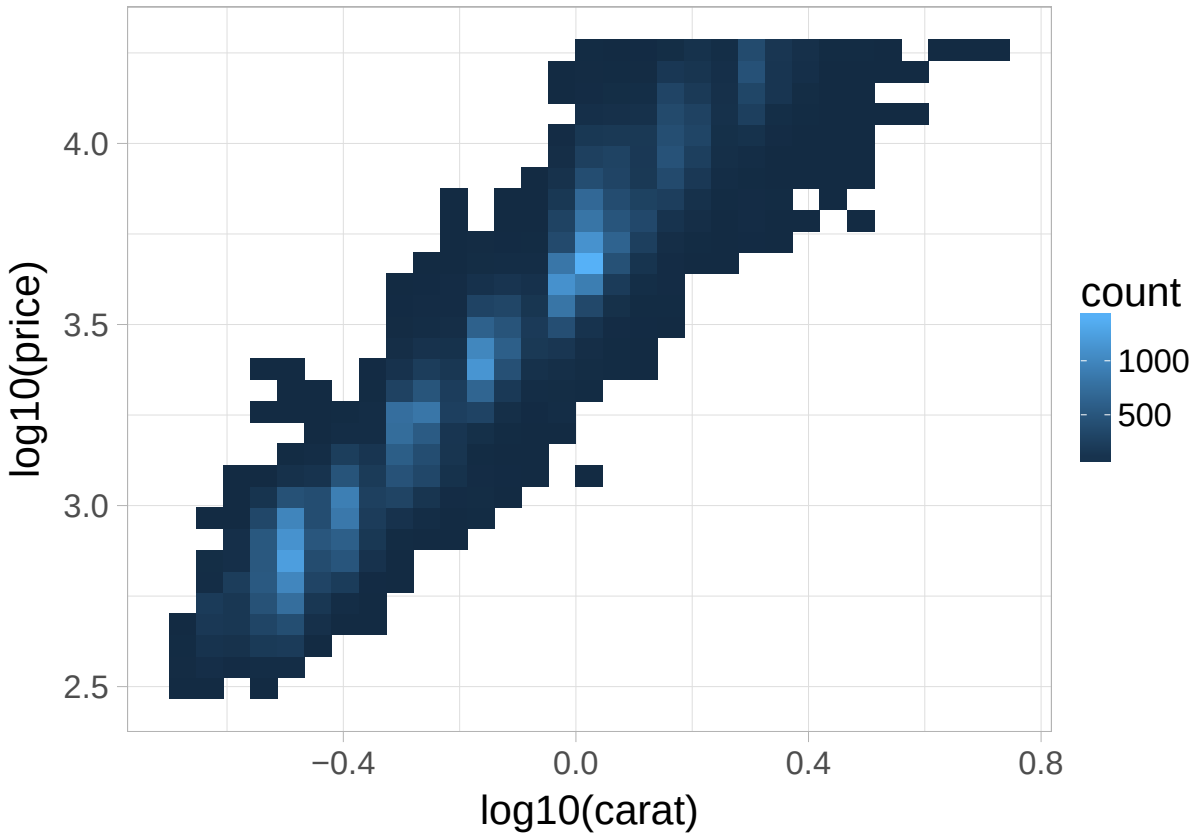
COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
ggplot(diamonds, aes(carat, price)) +
  geom_bin2d()
```



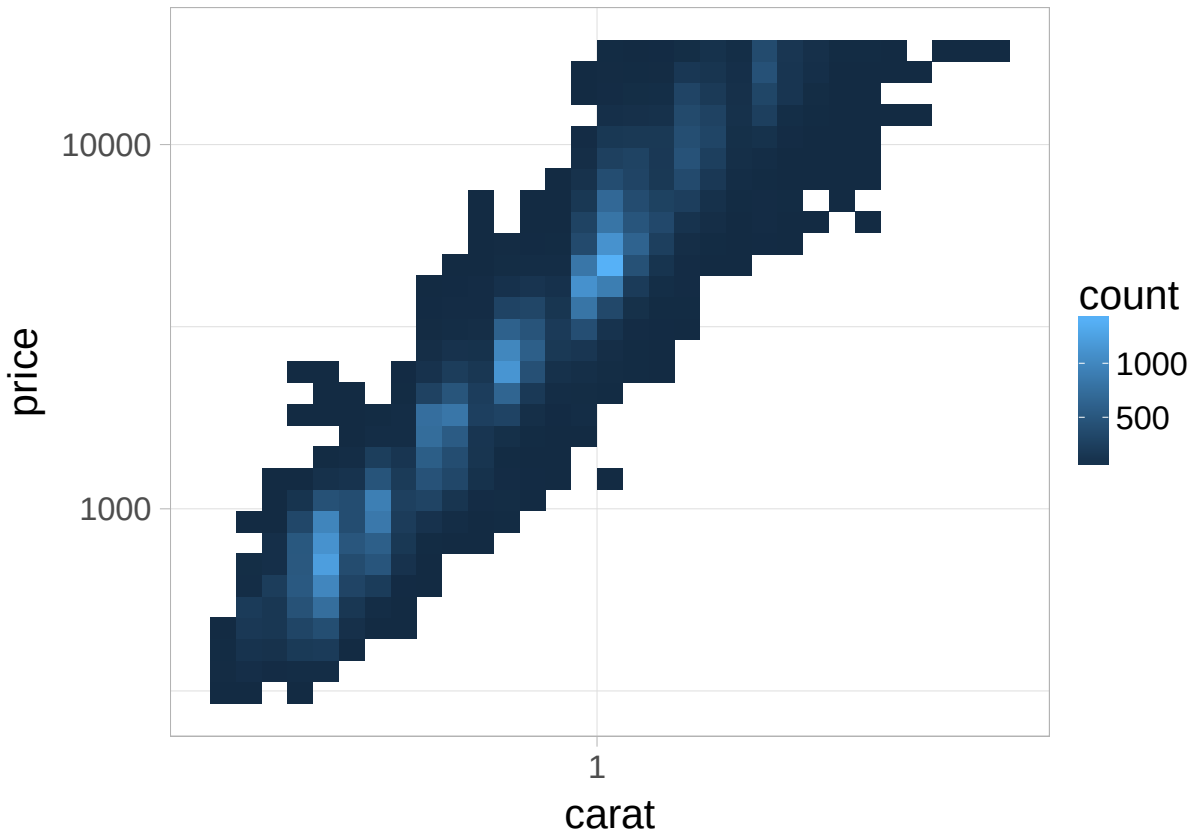
Log-transform the variables


```
ggplot(diamonds, aes(log10(carat), log10(price))) +
  geom_bin2d()
```



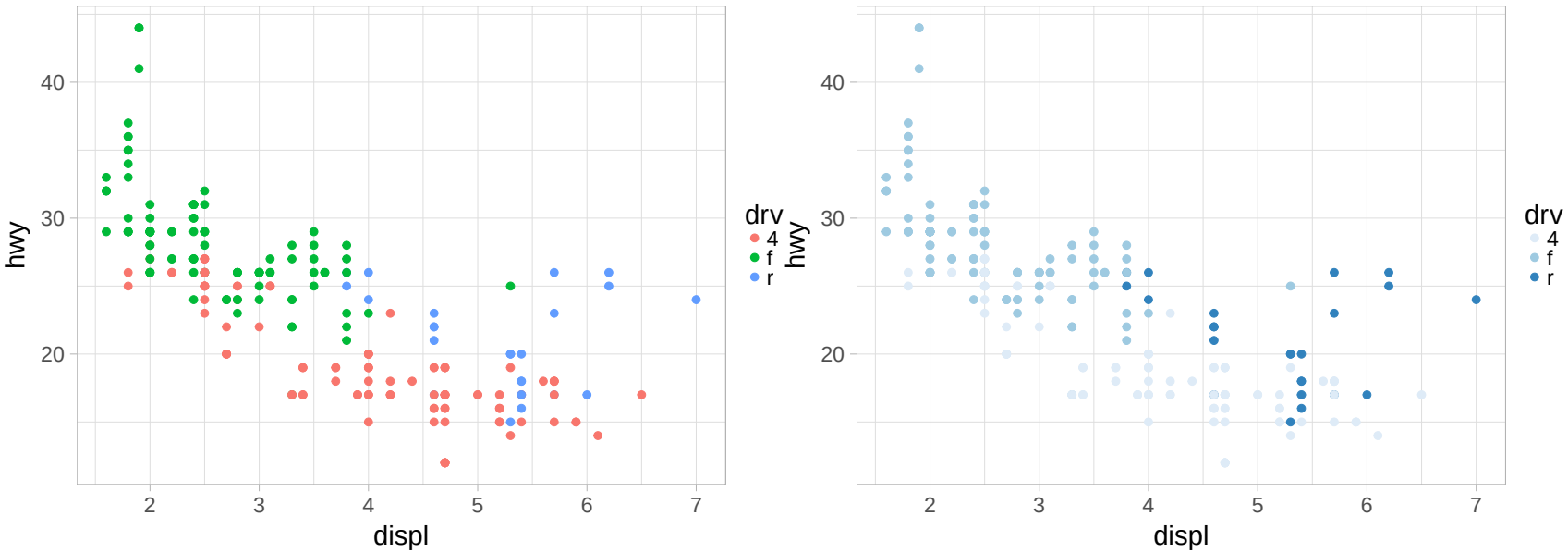
. . . or simply replace the scale


```
ggplot(diamonds, aes(carat, price)) + geom_bin2d() +
  scale_x_log10() + scale_y_log10()
```



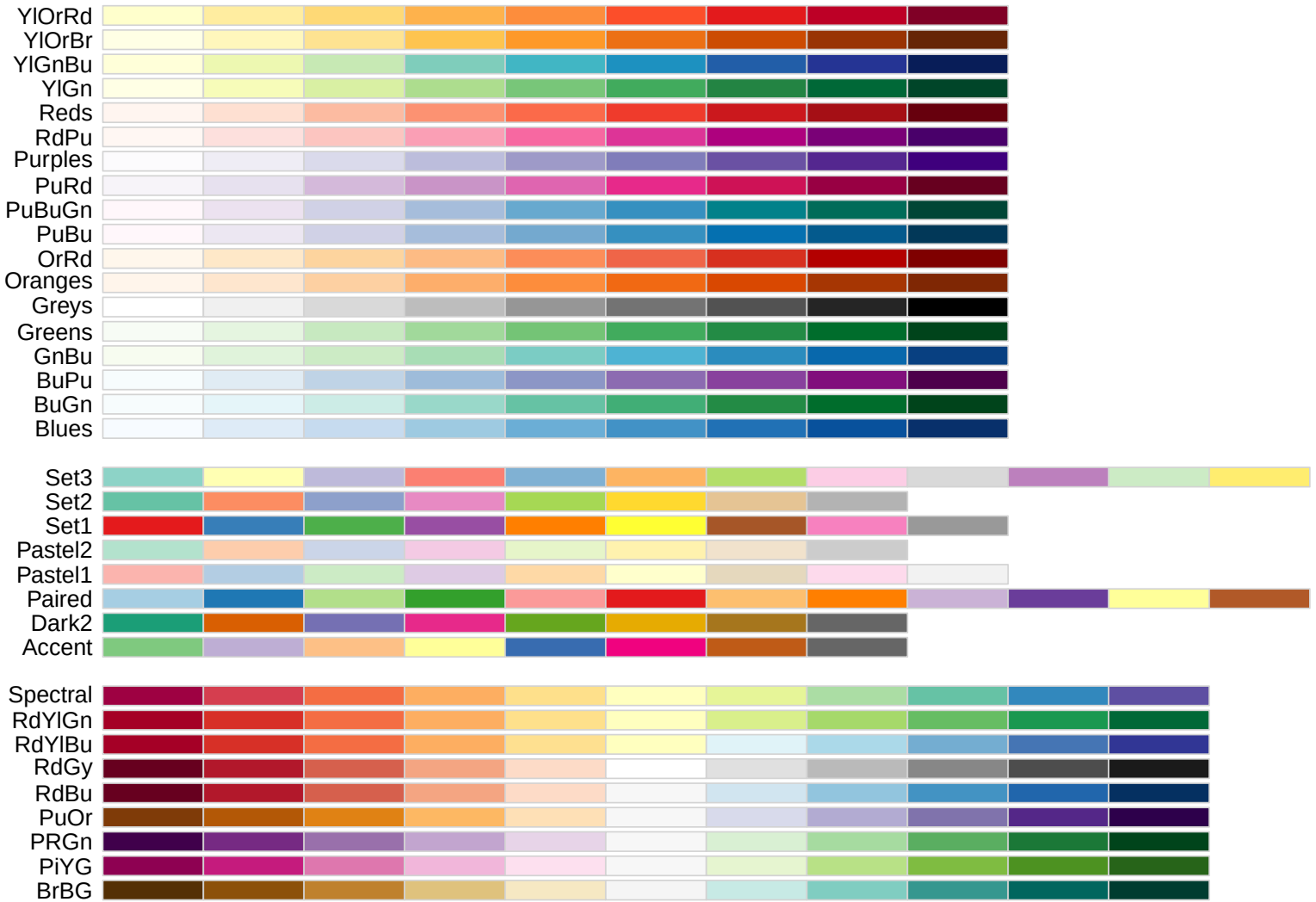
Replacing color scales


```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = drv), size = 3)

ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = drv), size = 3) +
  scale_color_brewer(palette = "Blues")
```



The ColorBrewer scales

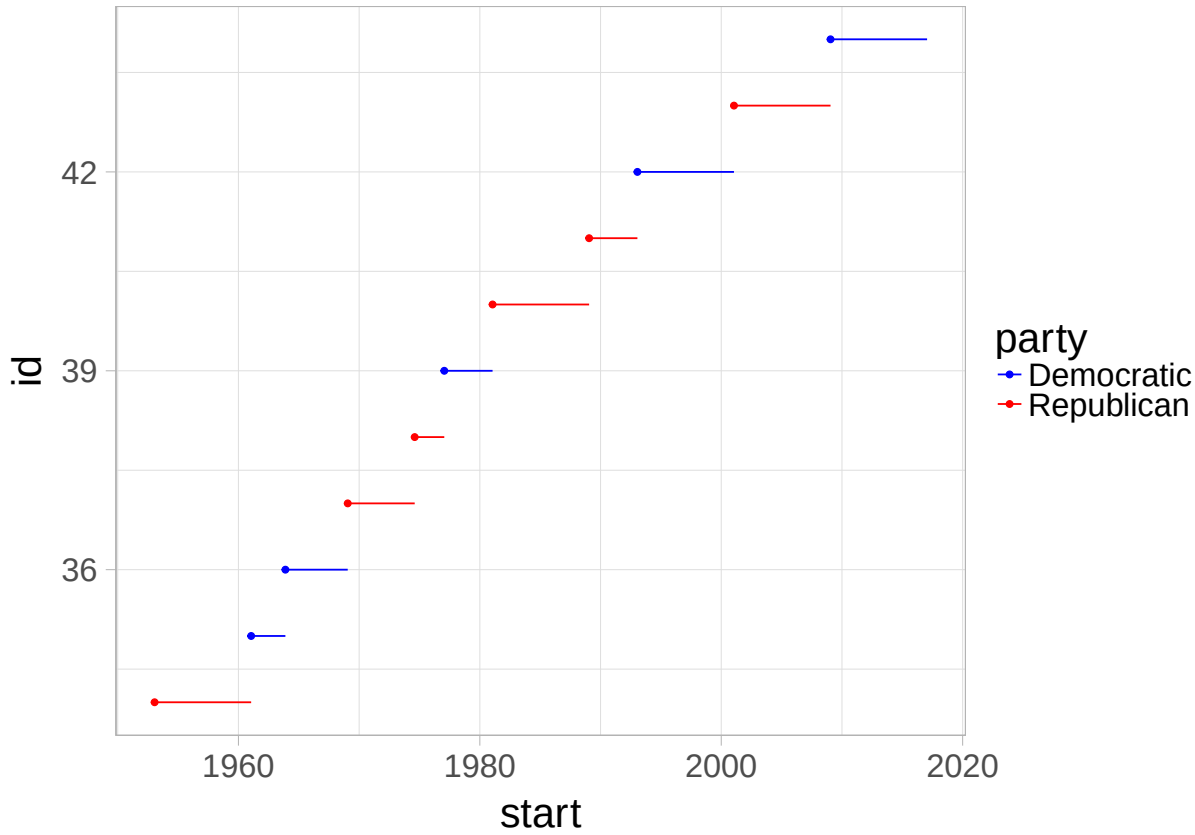
- Documented online at http://colorbrewer2.org/
- Available via the **RColorBrewer** package (Erich Neuwirth).



Using mannually defined mappings

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

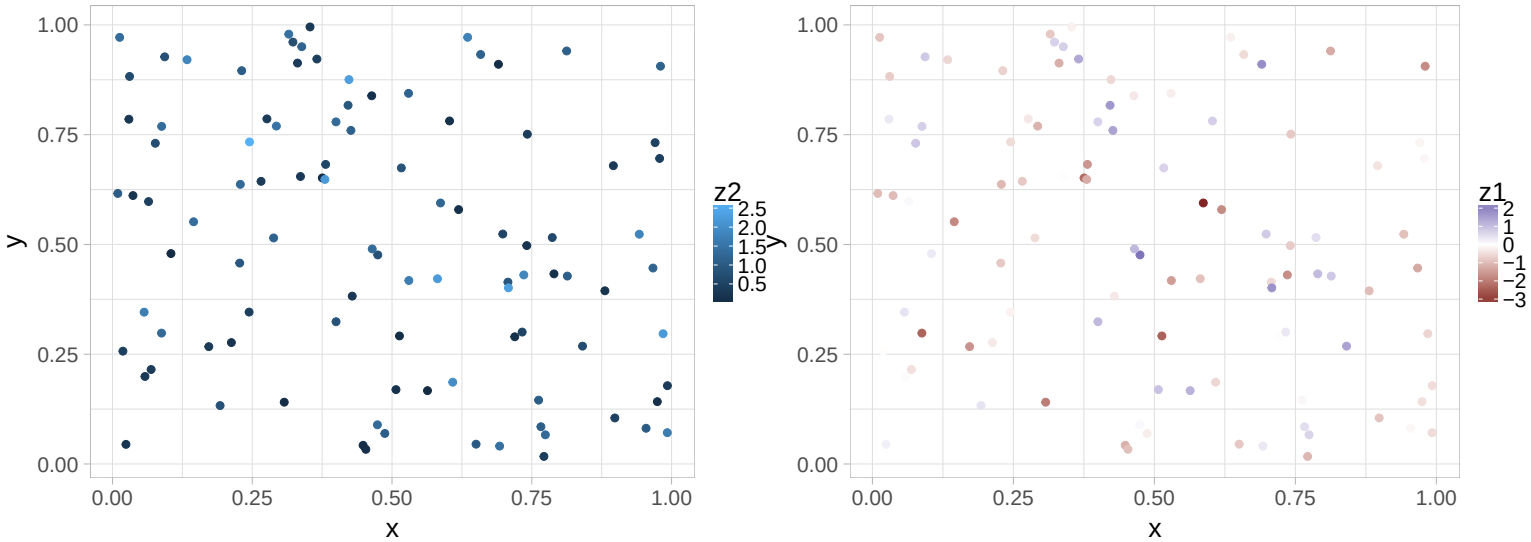
```
presidential %>%
  mutate(id = 33 + row_number()) %>%
  ggplot(aes(start, id, colour = party)) + geom_point() +
    geom_segment(aes(xend = end, yend = id)) +
    scale_colour_manual(values = c(Republican = "red", Democratic = "blue"))
```



Contiunous vs diverging color scales

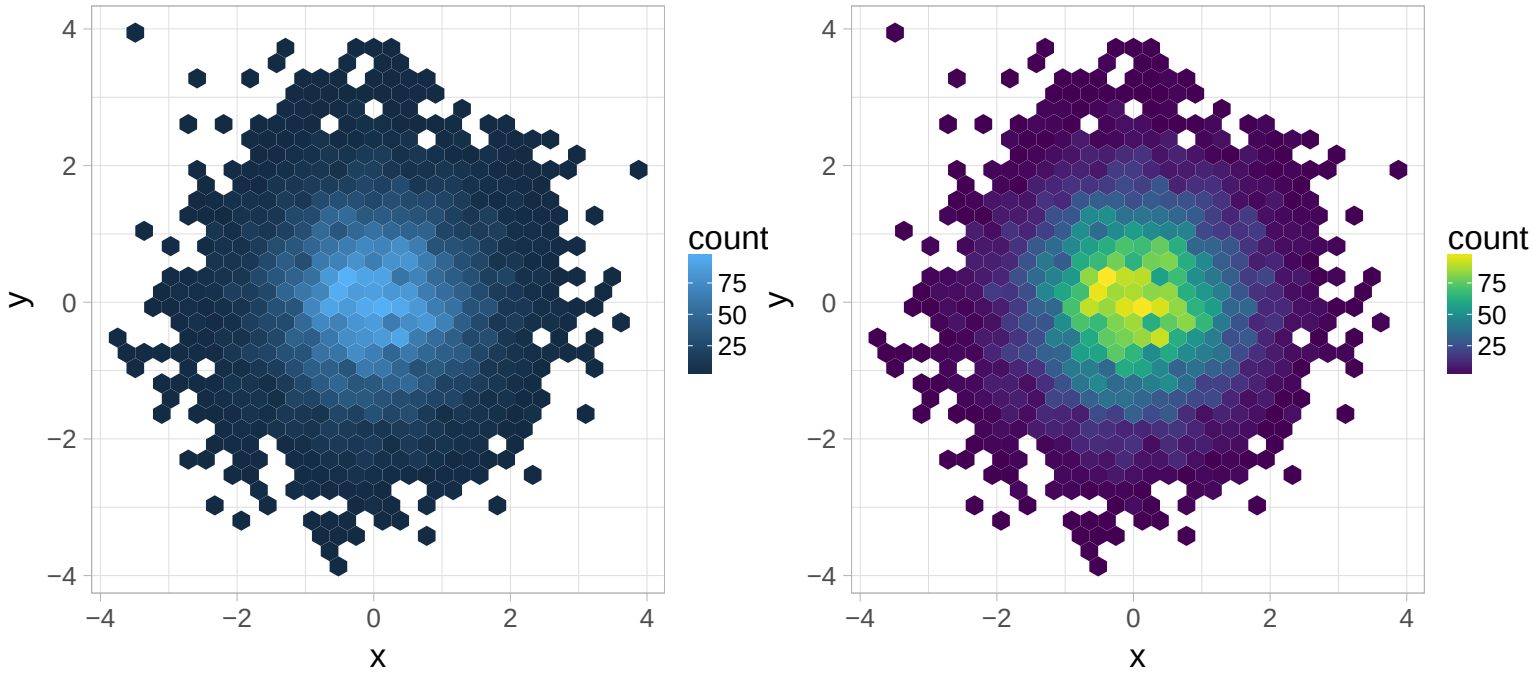
COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

```
df <- data.frame(x = runif(100), y = runif(100),
                 z1 = rnorm(100), z2 = abs(rnorm(100)))

ggplot(df, aes(x, y)) + geom_point(aes(colour = z2), size = 3)

ggplot(df, aes(x, y)) + geom_point(aes(colour = z1), size = 3) +
  scale_colour_gradient2()
```



A continuous analog of the categorical ColorBrewer


```
df <- tibble(x = rnorm(10000), y = rnorm(10000))
p <- ggplot(df, aes(x, y)) + geom_hex() + coord_fixed()
p
p + viridis::scale_fill_viridis()
```



Remark

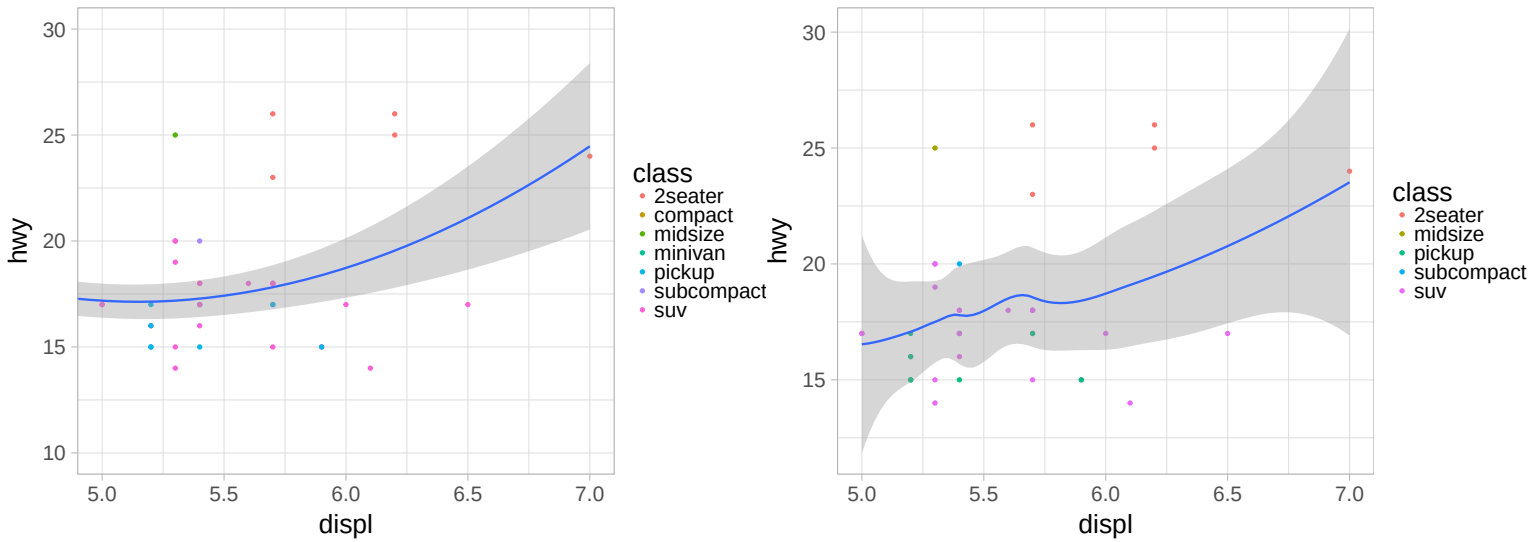
All color scales come in two variety:

- `scale_color_x()` for the `color` aesthetics (available in UK/US spellings)
- `scale_fill_x()` for the `fill` aesthetics.

Zooming

1. Adjusting what data are plotted
2. Setting the limits in each scale
3. Setting `xlim` and `ylim` in `coord_cartesian()`

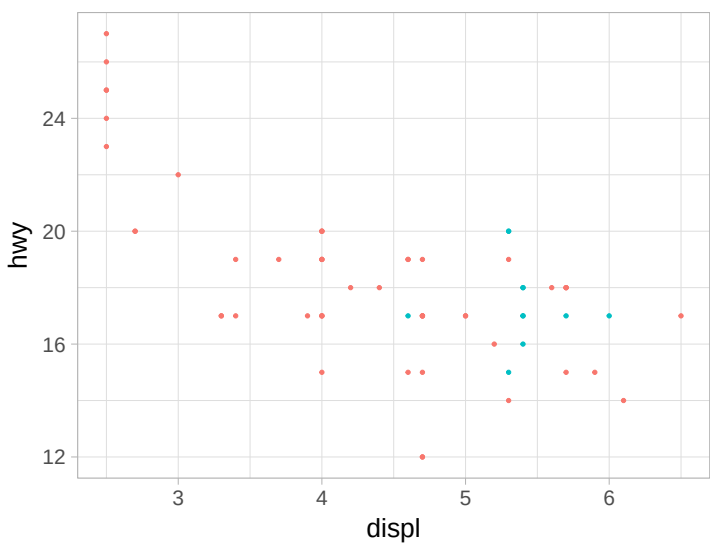
```
ggplot(mpg, mapping = aes(displ, hwy)) +
  geom_point(aes(color = class)) + geom_smooth() +
  coord_cartesian(xlim = c(5, 7), ylim = c(10, 30))

mpg %>% filter(displ >= 5, displ <= 7, hwy >= 10, hwy <= 30) %>%
  ggplot(aes(displ, hwy)) +
  geom_point(aes(color = class)) + geom_smooth()
```

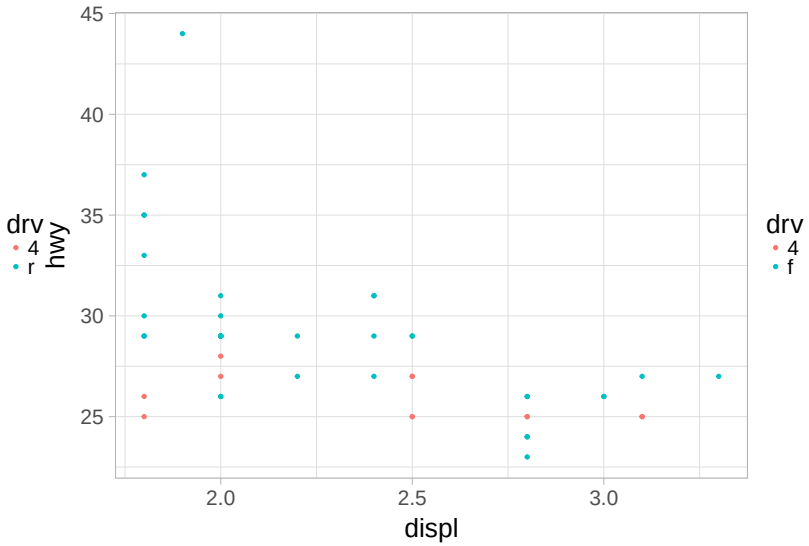


Set the `limits` on individual scales.

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

```
suv <- mpg %>% filter(class == "suv")
compact <- mpg %>% filter(class == "compact")

ggplot(suv, aes(displ, hwy, colour = drv)) + geom_point()
ggplot(compact, aes(displ, hwy, colour = drv)) + geom_point()
```

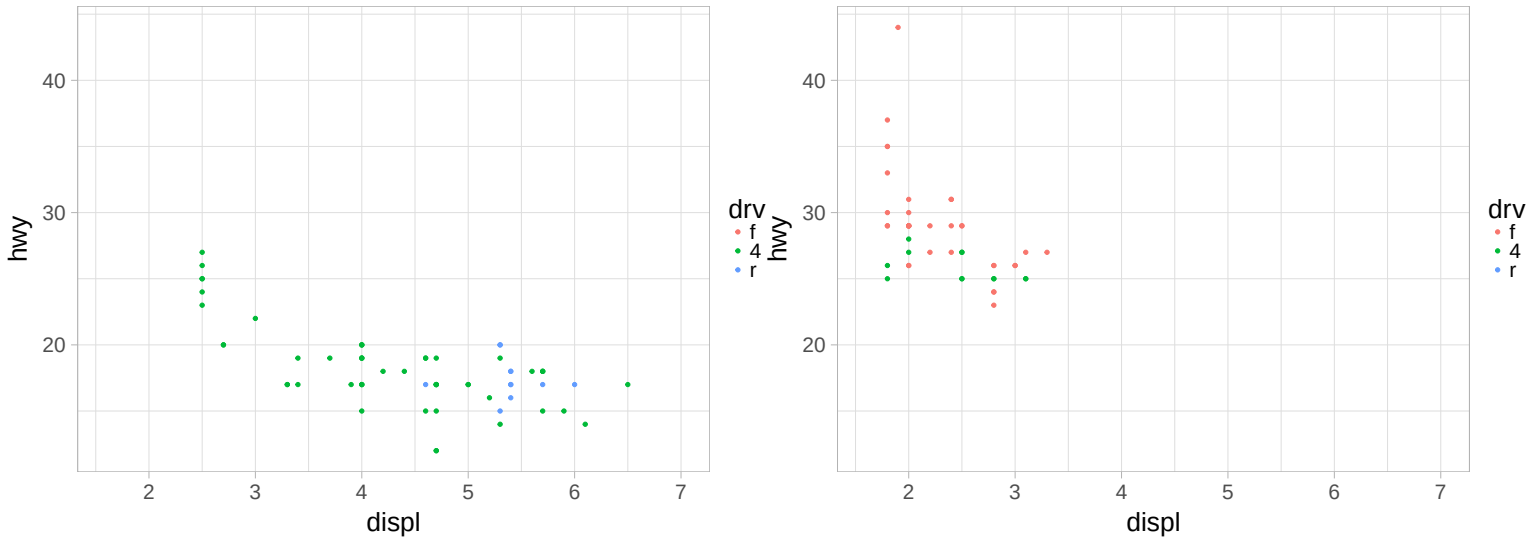




Share scales across multiple plots

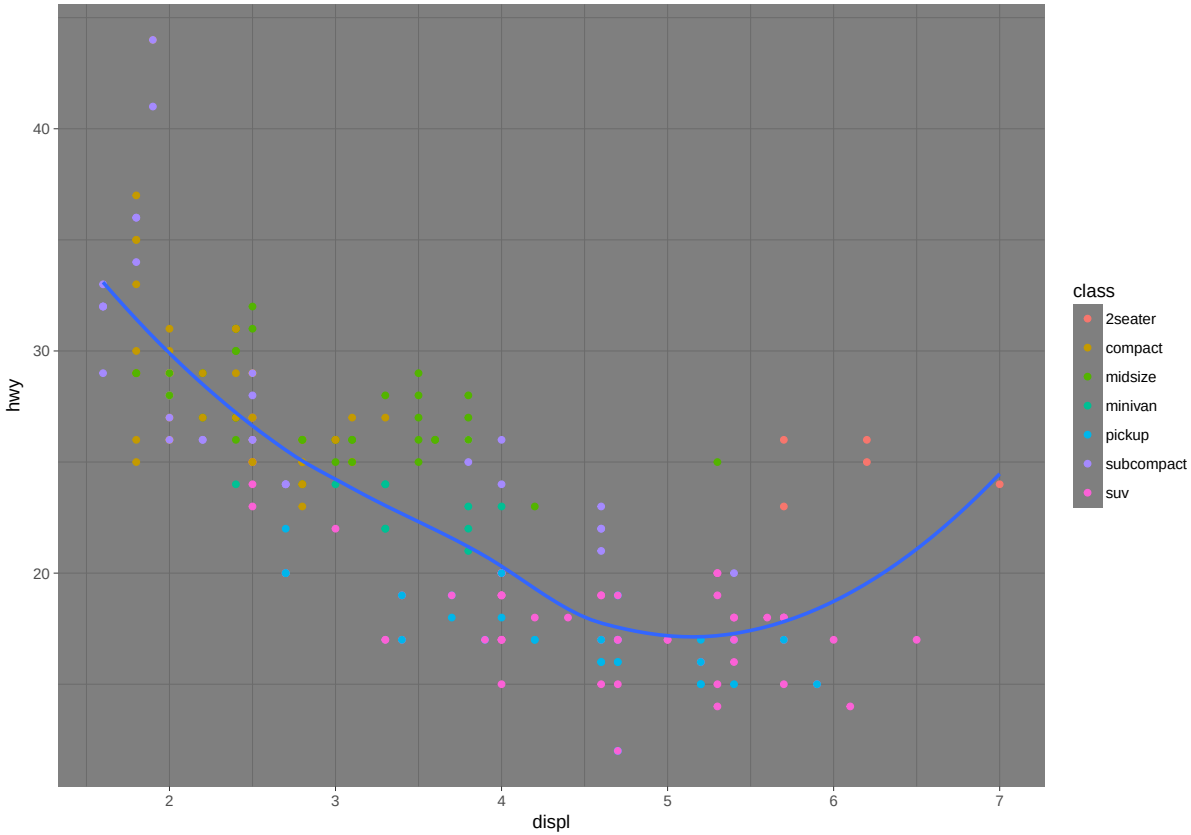
Training the scales with the `limits` of the full data:

```
x_scale <- scale_x_continuous(limits = range(mpg$displ))
y_scale <- scale_y_continuous(limits = range(mpg$hwy))
col_scale <- scale_colour_discrete(limits = unique(mpg$drv))

ggplot(suv, aes(displ, hwy, colour = drv)) + geom_point() +
  x_scale + y_scale + col_scale

ggplot(compact, aes(displ, hwy, colour = drv)) + geom_point() +
  x_scale + y_scale + col_scale
```

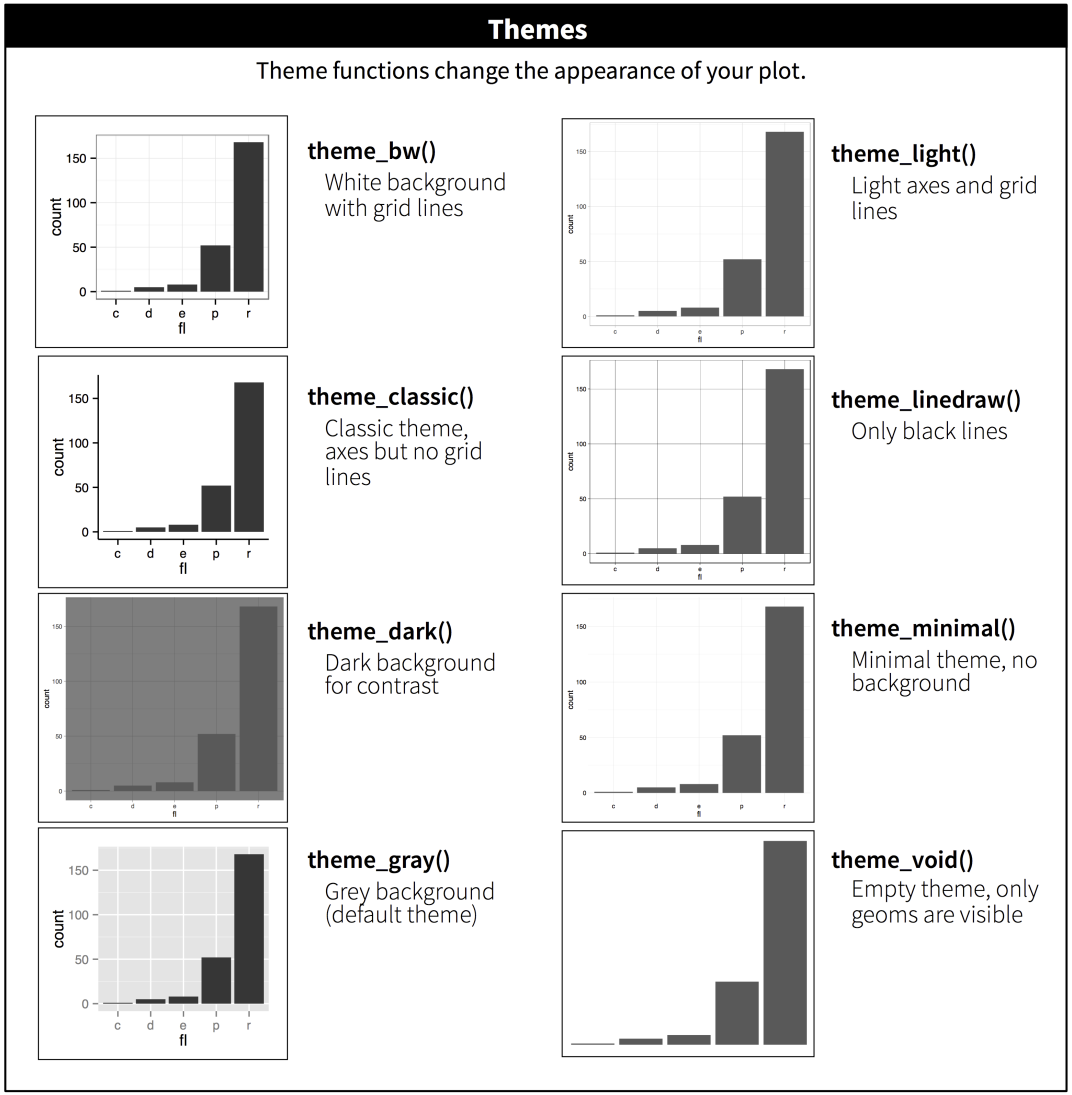


Themes


```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  theme_dark()
```



ggplot2 default themes



More in add-on packages like ggthemes

Saving your plots

Two main ways to get your plots out of R and into your final write-up:

- ggsave()
- knitr

```
ggplot(mpg, aes(displ, hwy)) + geom_point()
ggsave("my-plot.pdf")
```

Two remarks:

- Without specifying `width` and `height`, they are taken from the dimensions of the current plotting device. For reproducible code, you should **ALWAYS** specify them.
- You should be assembling your final reports using R Markdown, so rather focus on the important code chunk options.

Figure sizing

- Biggest challenge of graphics in R Markdown: getting your figures the right size and shape.
- Five main options: `fig.width`, `fig.height`, `fig.asp`, `out.width` and `out.height`.

Hadley's advice is to use three of the five options:

- Set `fig.width = 6` and `fig.asp = 0.618` as defaults. Then in individual chunks, adjust `fig.asp` if needed.
- Set `out.width` to a percentage of the line width (e.g., default to `out.width = "70%"` and `fig.align = "center"`)
- For multiple plots in a single row, set `out.width` to 50% for two plots, 33% for 3 plots, or 25% to 4 plots, and set `fig.align = "default"`.

More advices

On figure sizing:

- If you have to squint to read the text, tweak `fig.width`.
  - If `fig.width` is larger than the size the figure is rendered in the final doc, the text will be too small.
  - If `fig.width` is smaller, the text will be too big.
  - Experiment to figure out the right ratio between the `fig.width` and the eventual width in your document.

In general:

- Set `fig.show = "hold"`.
- Add captions to your plots, using `fig.cap = "something interesting"`.
- Use RStudio's data visualization cheatsheet.
- Inspire yourself from the R graph gallery.