Notes on Chapter 11: Communication

Norman Lim

2025-07-23

Prerquisites

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.1.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(scales)
```

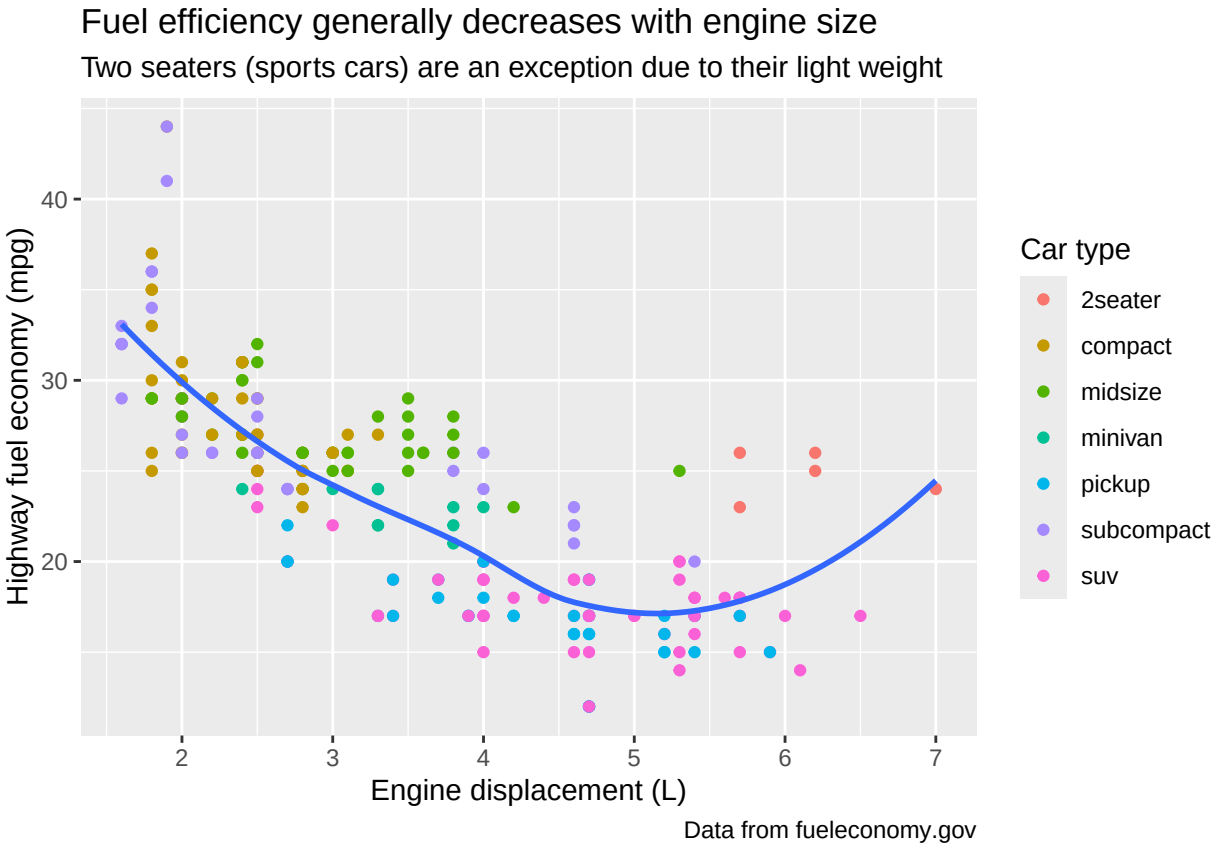
```
##
## Attaching package: 'scales'
##
## The following object is masked from 'package:purrr':
##
##     discard
##
## The following object is masked from 'package:readr':
##
##     col_factor
```

```
library(ggrepel)
library(patchwork)
```

Adding labels to a plot using the `labs()` function:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  labs(
    x = "Engine displacement (L)",
    y = "Highway fuel economy (mpg)",
    color = "Car type",
    title = "Fuel efficiency generally decreases with engine size",
    subtitle = "Two seaters (sports cars) are an exception due to their light weight",
    caption = "Data from fueleconomy.gov"
  )
```

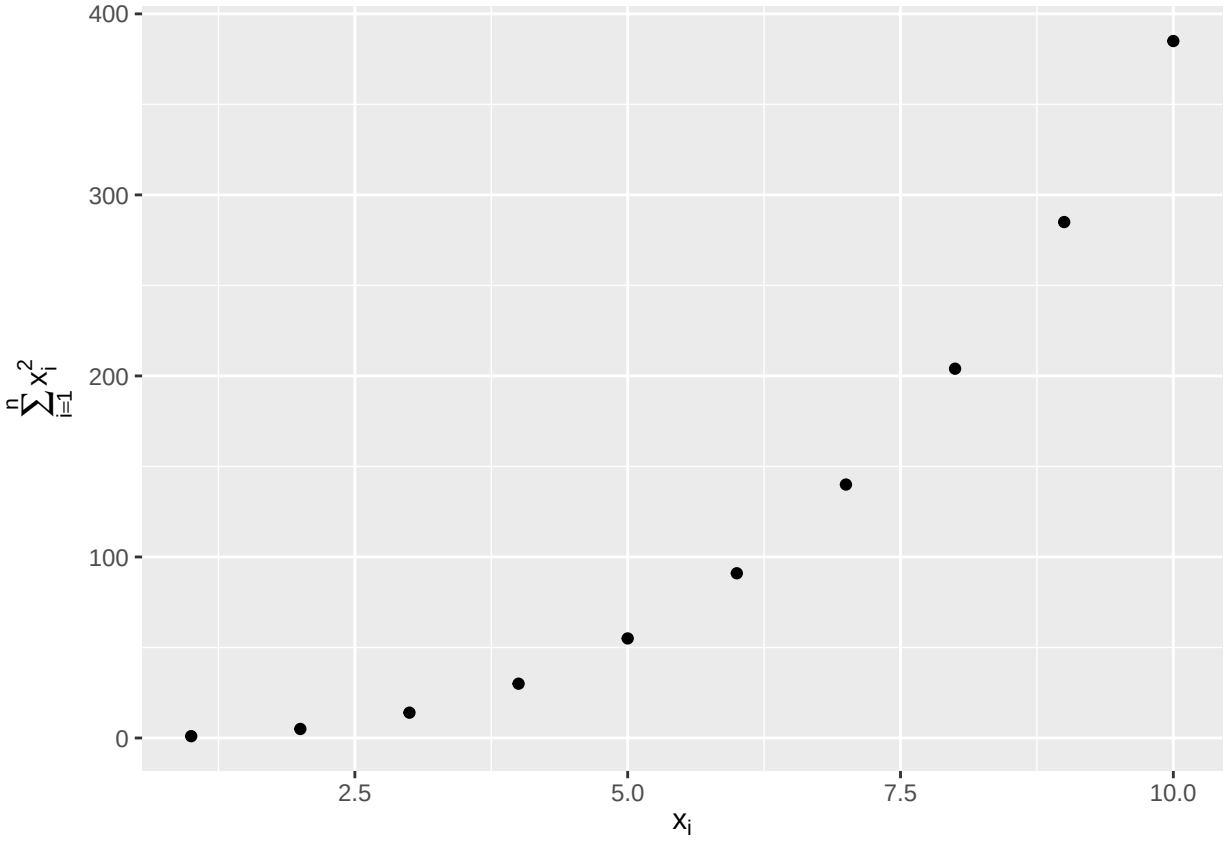
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



- plot title should summarize the main finding.
- titles that only describes what the plot is, should be avoided.

Using mathematical equations or symbols in the labels instead of text strings made possible by the `quote()` function:

```
df <- tibble(
  x = 1:10,
  y = cumsum(x^2)
)

ggplot(df, aes(x, y)) +
  geom_point() +
  labs(
    x = quote(x[i]),
    y = quote(sum(x[i] ^ 2, i == 1, n))
  )
```

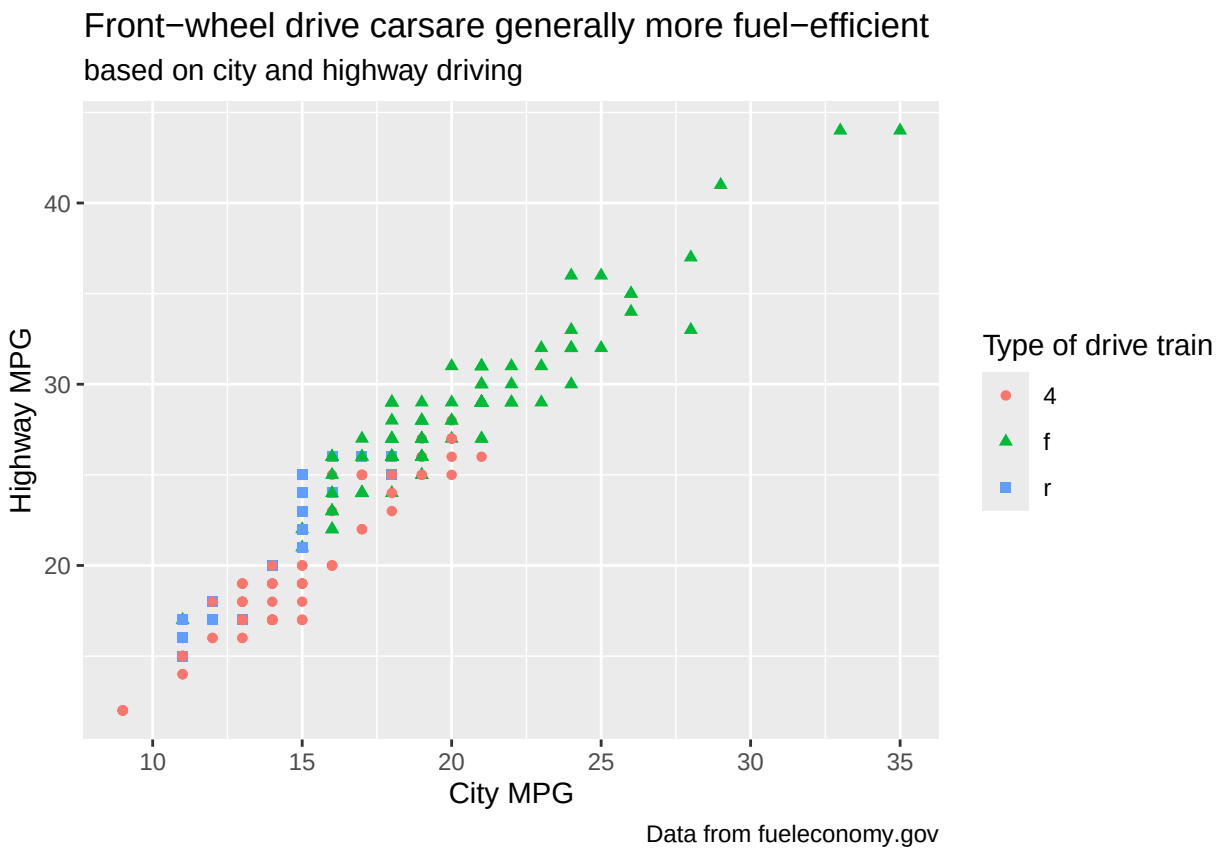


- learn more `quote()` in `?plotmath`

Exercises

1. Create one plot on the fuel economy data with customized title, subtitle, caption, x, y, and color labels.

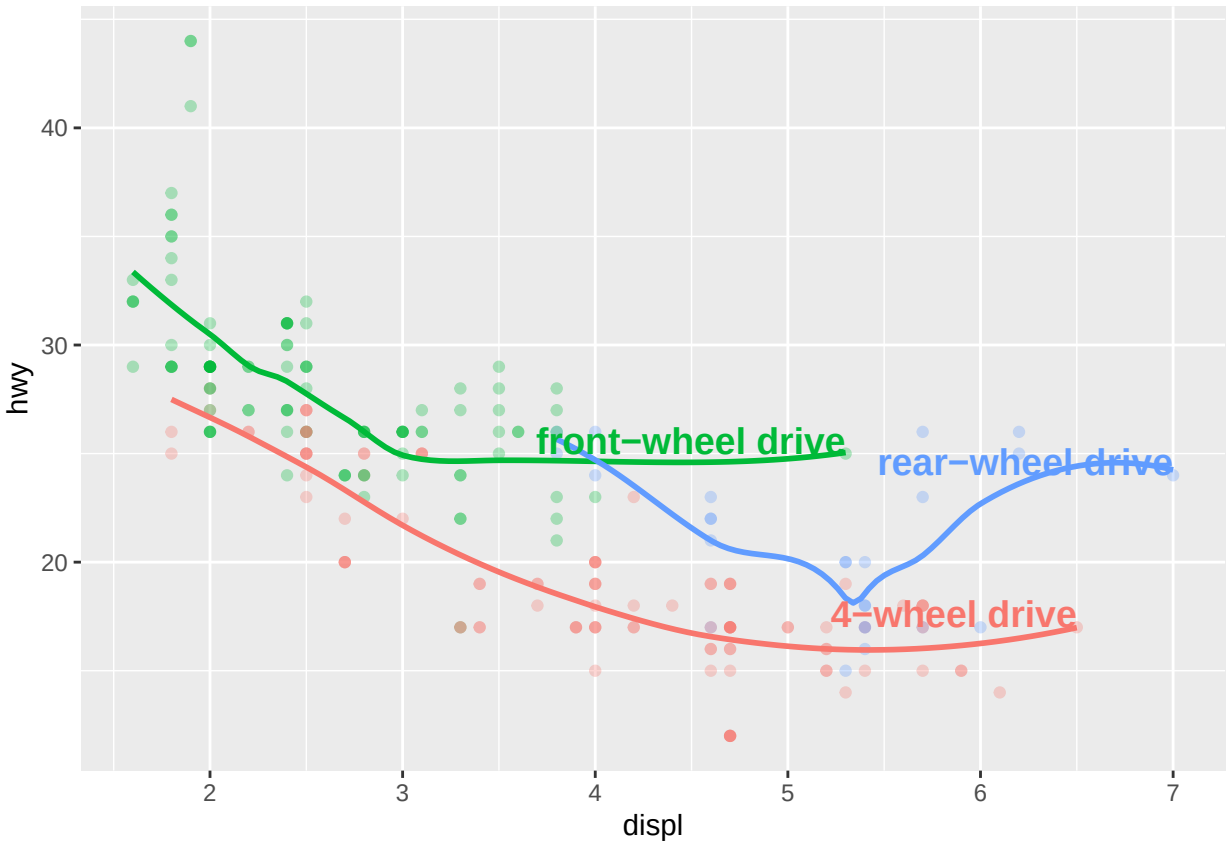
```
ggplot(mpg, aes(x = cty, y = hwy)) +
  geom_point(aes(color = drv, shape = drv)) +
  labs(
    title = "Front-wheel drive carsare generally more fuel-efficient",
    subtitle = "based on city and highway driving",
    color = "Type of drive train",
    shape = "Type of drive train",
    x = "City MPG",
    y = "Highway MPG",
    color = "Type of drive train",
    caption = "Data from fueleconomy.gov"
  )
```



Adding annotations to the plot using `geom_text()`

```
# create a new data frame containing the desired "labels"
label_info <- mpg |>
  group_by(drv) |>
  arrange(desc(displ)) |>
  slice_head(n = 1) |>
  mutate(
    drive_type = case_when(
      drv == "f" ~ "front-wheel drive",
      drv == "r" ~ "rear-wheel drive",
      drv == "4" ~ "4-wheel drive"
    )
  ) |>
  select(displ, hwy, drv, drive_type)

# use the new data frame to directly label the mpg data
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point(alpha = 0.3) +
  geom_smooth(se = FALSE) +
  geom_text(
    data = label_info,
    aes(x = displ, y = hwy, label = drive_type),
    fontface = "bold", size = 5, hjust = "right", vjust = "bottom"
  ) +
  theme(legend.position = "none")
```

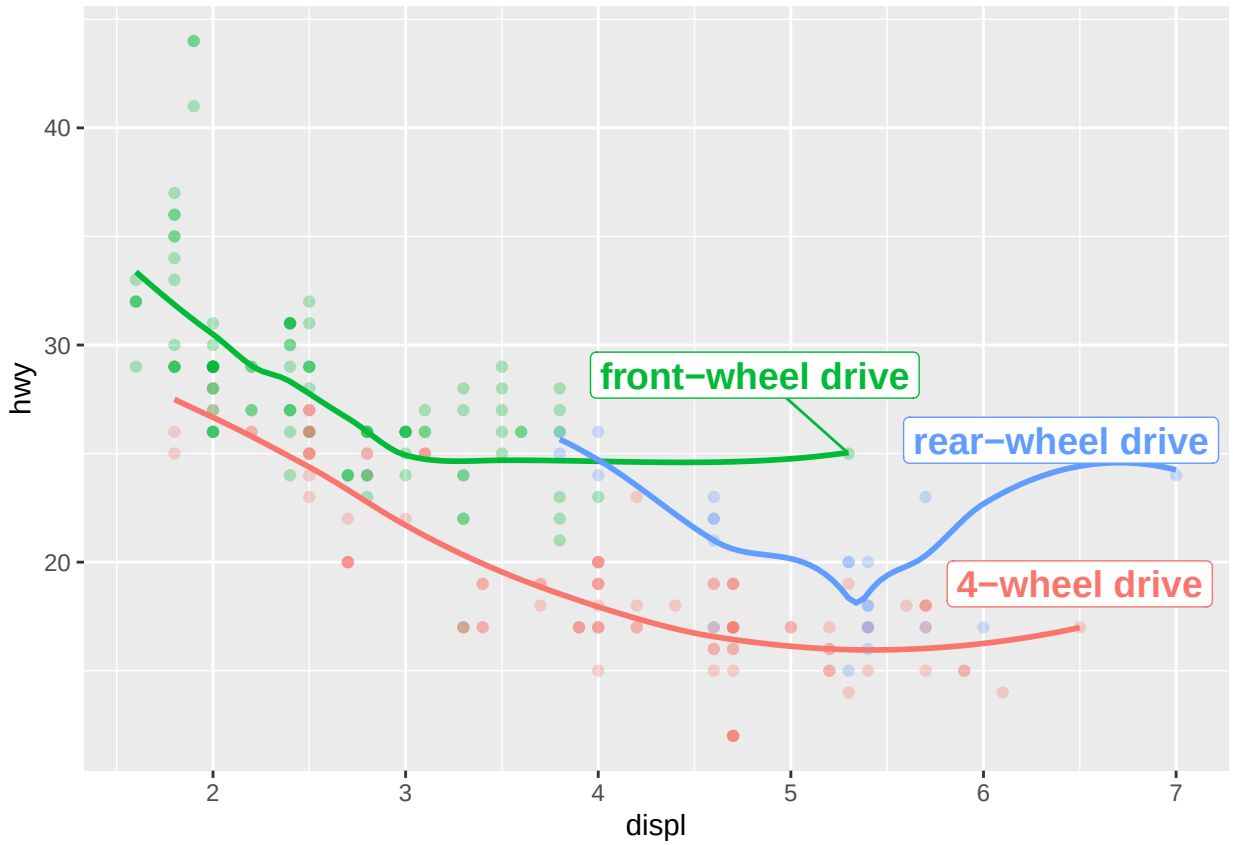
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Using `label_repel()` to prevent the label from overlapping with the lines (and other things) on the plot

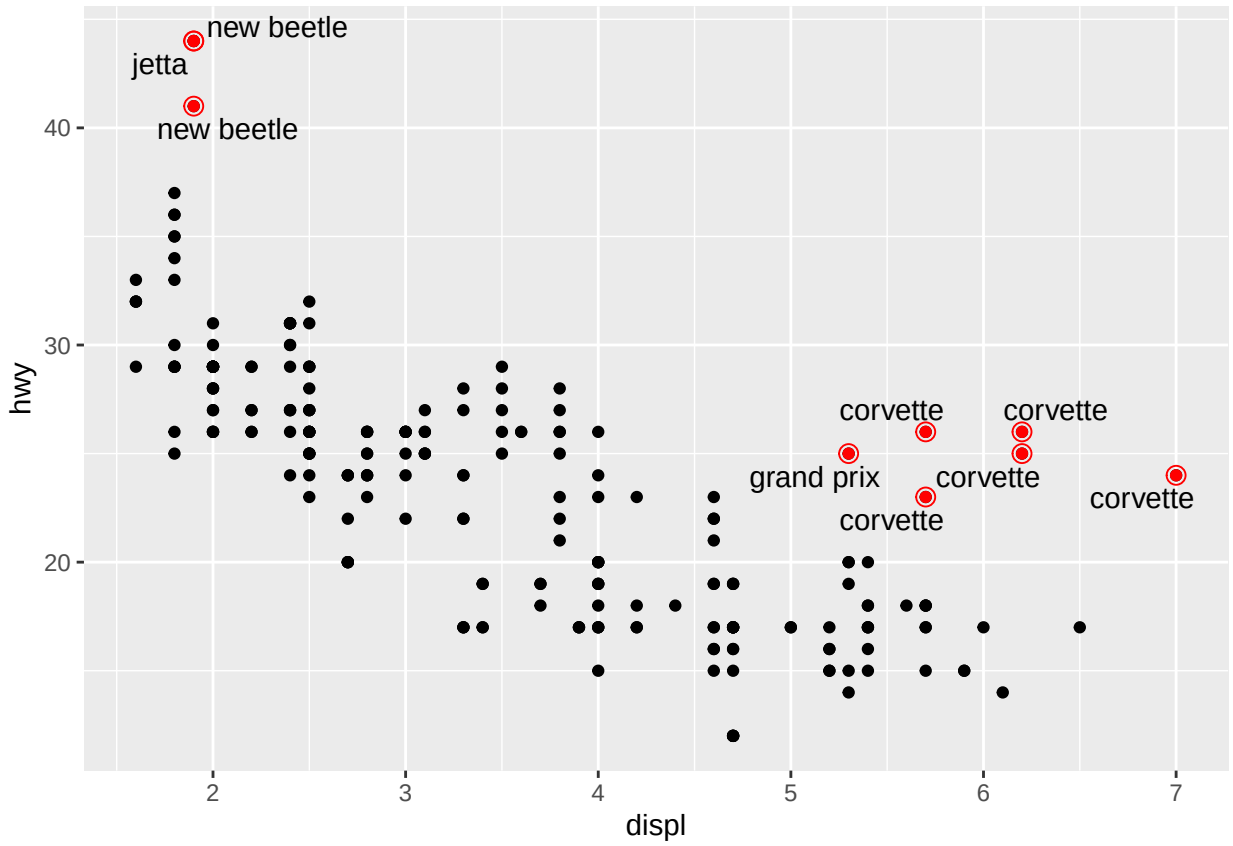
```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point(alpha = 0.3) +
  geom_smooth(se = FALSE) +
  geom_label_repel(
    data = label_info,
    aes(x = displ, y = hwy, label = drive_type),
    fontface = "bold", size = 5, nudge_y = 2
  ) +
  theme(legend.position = "none")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Using `geom_text_repel()`

```
potential_outliers <- mpg |>
  filter(hwy > 40 | (hwy > 20 & displ > 5))

ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  geom_text_repel(data = potential_outliers, aes(label = model)) +
  geom_point(data = potential_outliers, color = "red") +
  geom_point(
    data = potential_outliers,
    color = "red", size = 3, shape = "circle open"
  )
```



More ideas for annotating the plot: - use `geom_hline()` and/or `geom_vline()` to add reference lines. (it is also a good idea to use `linewidth = 2` and `color = "white"` to make them easy to see).
- use `geom_rect()` to draw a rectangle around points of inteerest.
- use `goem_segment()` with the `arrow` argument to "draw" arrows on the plot.
- use `annotate()` to directly write annotations on the plot.

Demonstration using `annotate()` with `stringr::str_wrap()`:

```
trend_text <- "Larger engine sizes tend to have lower fuel economy" |>
  str_wrap(width = 30)

trend_text
```

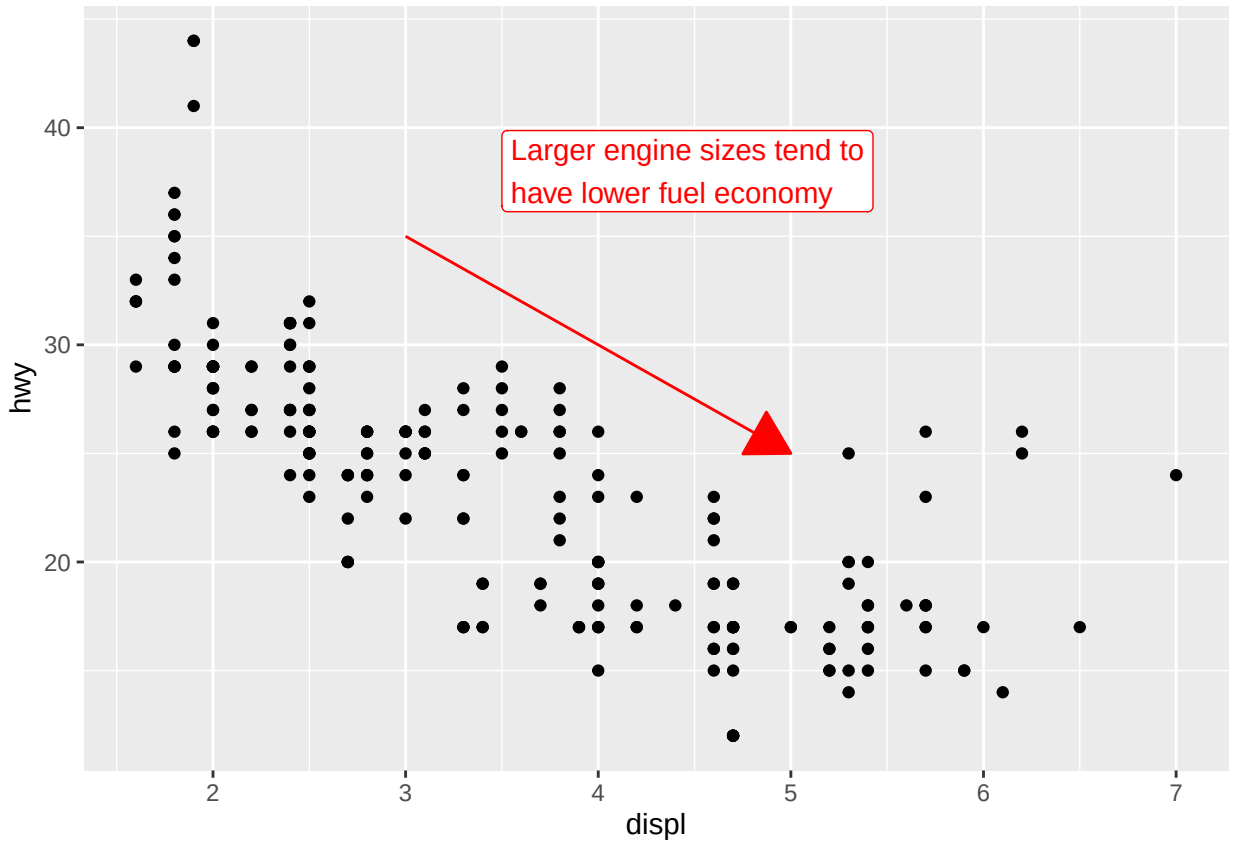
```
## [1] "Larger engine sizes tend to\nhave lower fuel economy"
```

As you can see, the `str_wrap()` function automatically added the line break (`\n`).

Another annotation demonstration

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  annotate(
    geom = "label", x = 3.5, y = 38,
    label = trend_text,
    hjust = "left", color = "red"
  ) +
  annotate(
    geom = "segment",
    x = 3, y = 35, xend = 5, yend = 25, color = "red",
    arrow = arrow(type = "closed")
```

```
)
```

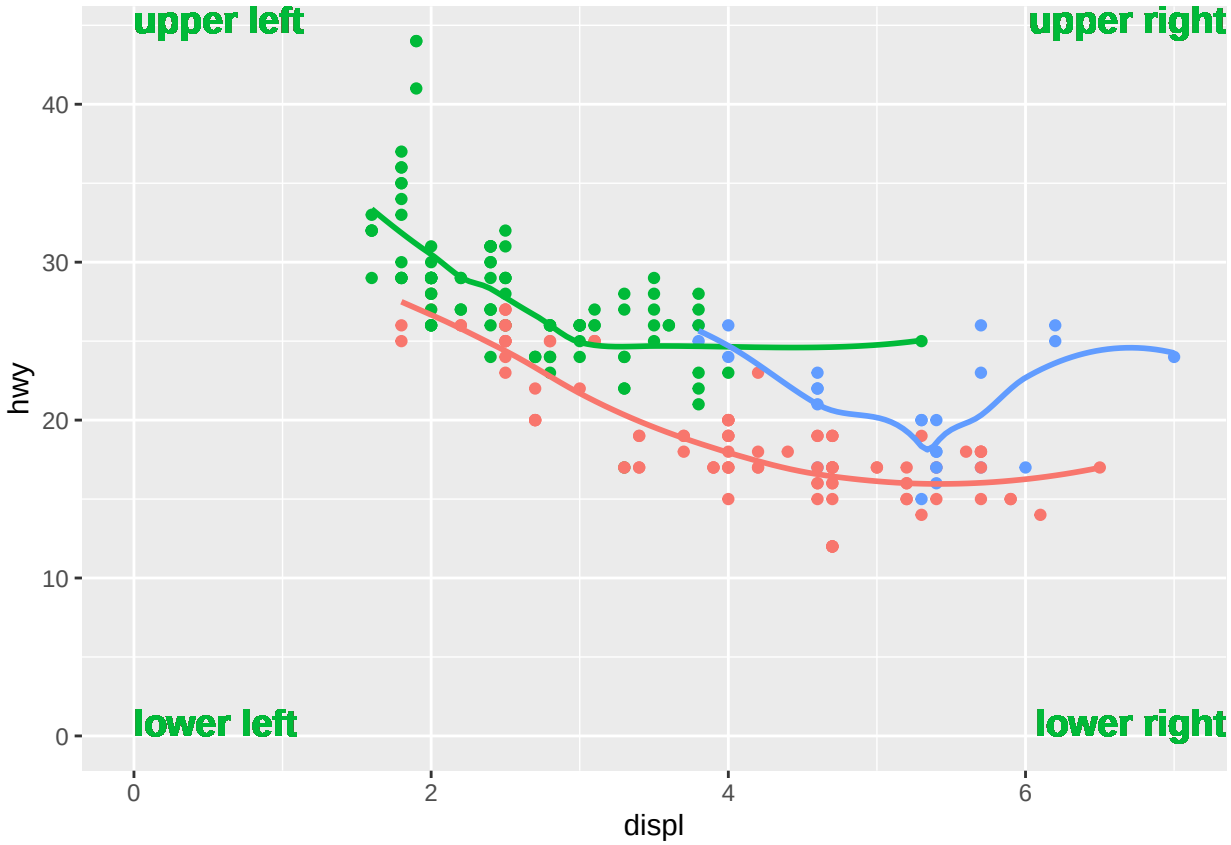


Exercises

1. Use `geom_text()` with infinite positions to place text at the four corners of the plot.

```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(se = FALSE) +
  geom_text(
    aes(x = 0, y = Inf, label = "upper left"),
    fontface = "bold", size = 5, hjust = "left", vjust="top"
  ) +
  geom_text(
    aes(x = Inf, y = Inf, label = "upper right"),
    fontface = "bold", size = 5, hjust = "right", vjust = "top"
  ) +
  geom_text(
    aes(x = 0, y = 0, label = "lower left"),
    fontface = "bold", size = 5, hjust = "left", vjust = "bottom"
  ) +
  geom_text(
    aes(x = Inf, y = 0, label = "lower right"),
    fontface = "bold", size = 5, hjust = "right", vjust = "bottom"
  ) +
  theme(legend.position = "none")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

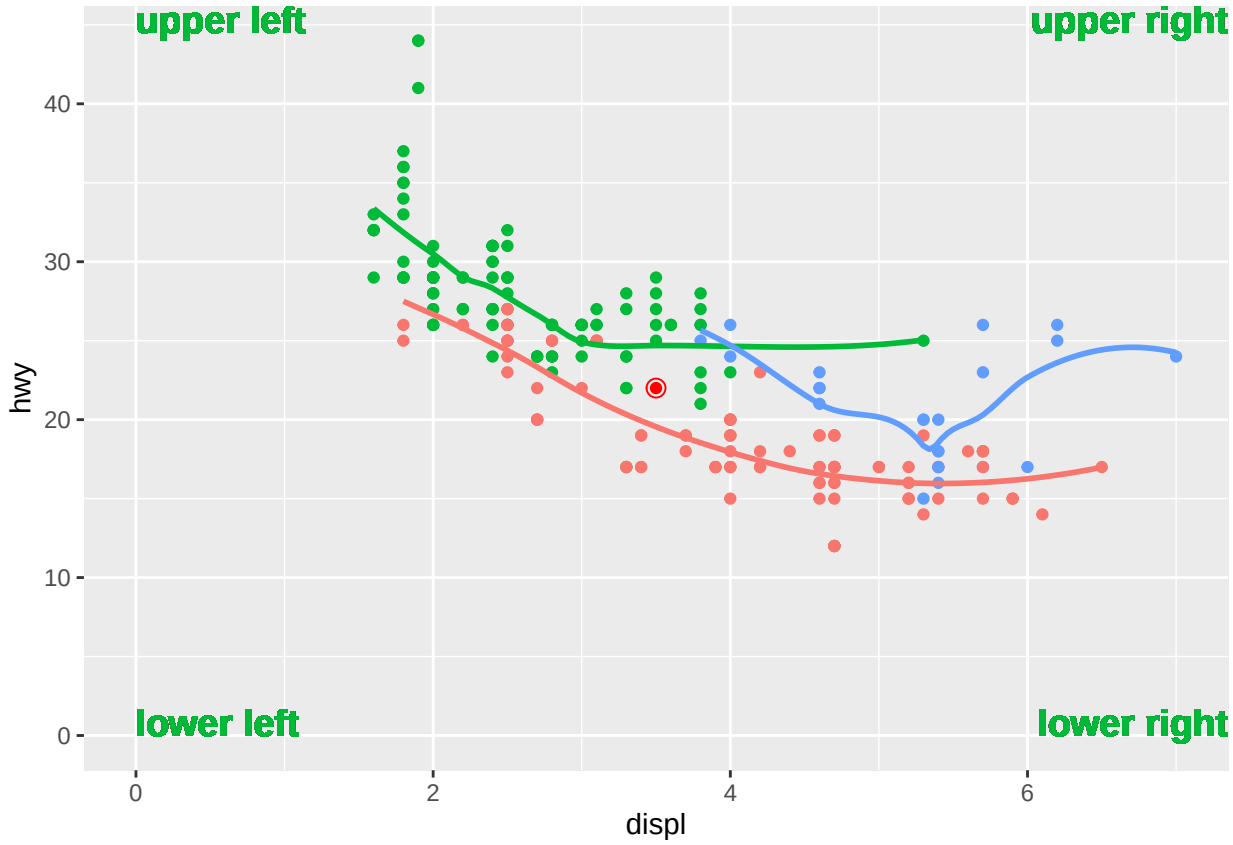


2. Use `annotate()` to add a point geom in the middle of your last plot without having to create a tibble. Customize the shape, size, or color of the point.

```
mid_x = (max(mpg$displ)) / 2
mid_y = (max(mpg$hwy)) / 2

ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(se = FALSE) +
  geom_text(
    aes(x = 0, y = Inf, label = "upper left"),
    fontface = "bold", size = 5, hjust = "left", vjust="top"
  ) +
  geom_text(
    aes(x = Inf, y = Inf, label = "upper right"),
    fontface = "bold", size = 5, hjust = "right", vjust = "top"
  ) +
  geom_text(
    aes(x = 0, y = 0, label = "lower left"),
    fontface = "bold", size = 5, hjust = "left", vjust = "bottom"
  ) +
  geom_text(
    aes(x = Inf, y = 0, label = "lower right"),
    fontface = "bold", size = 5, hjust = "right", vjust = "bottom"
  ) +
  annotate(
```

```
    geom = "point",
    x = mid_x, y = mid_y, color = "red"
  ) +
  annotate(
    geom = "point",
    x = mid_x, y = mid_y,
    color = "red", size = 3,
    shape = "circle open"
  ) +
  theme(legend.position = "none")
```

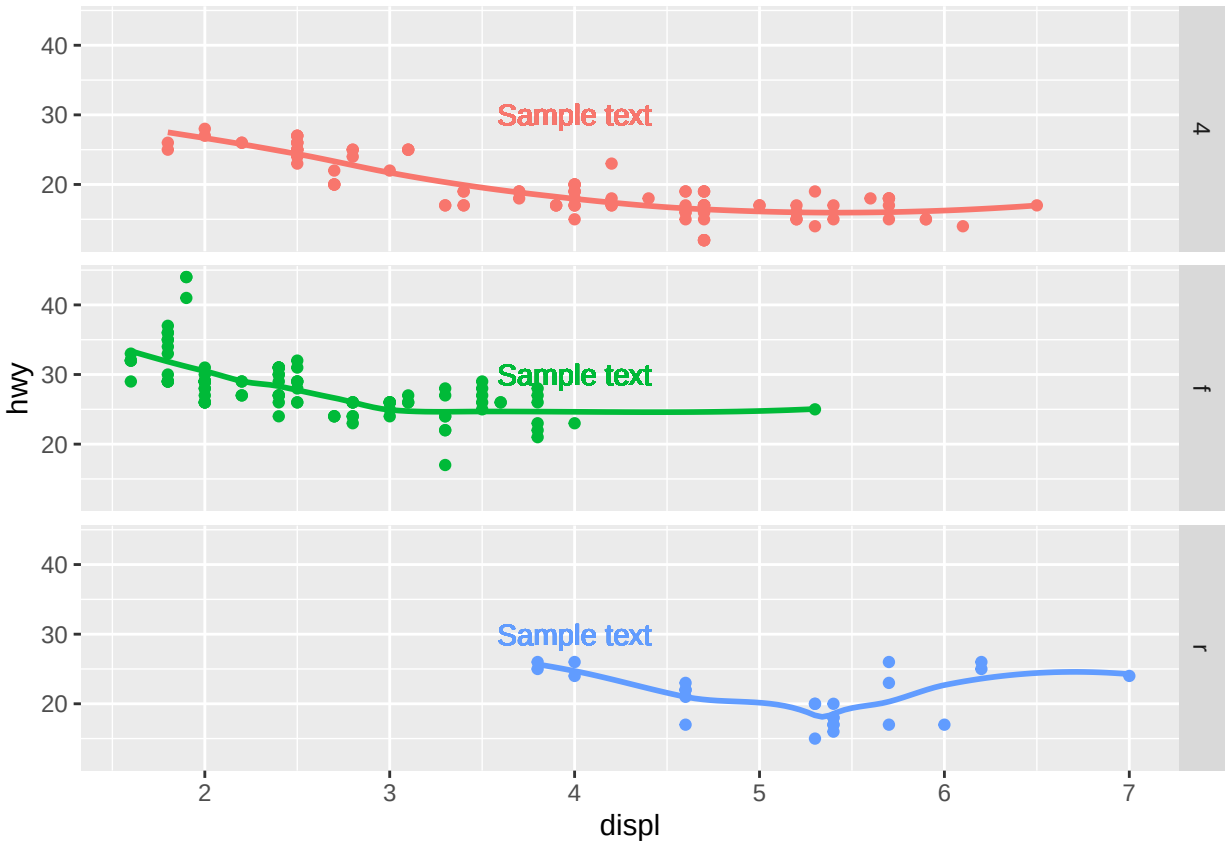
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



3. How do labels with `geom_text()` interact with faceting? How can you add a label to a single facet? How can you put a different label in each facet? (Hint: Think about the dataset that is being passed to `geom_text()`.)

```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(se = FALSE) +
  facet_grid(drv ~ .) +
  theme(legend.position = "none") +
  geom_text(
    x = 4, y = 30,
    label = "Sample text"
  )
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

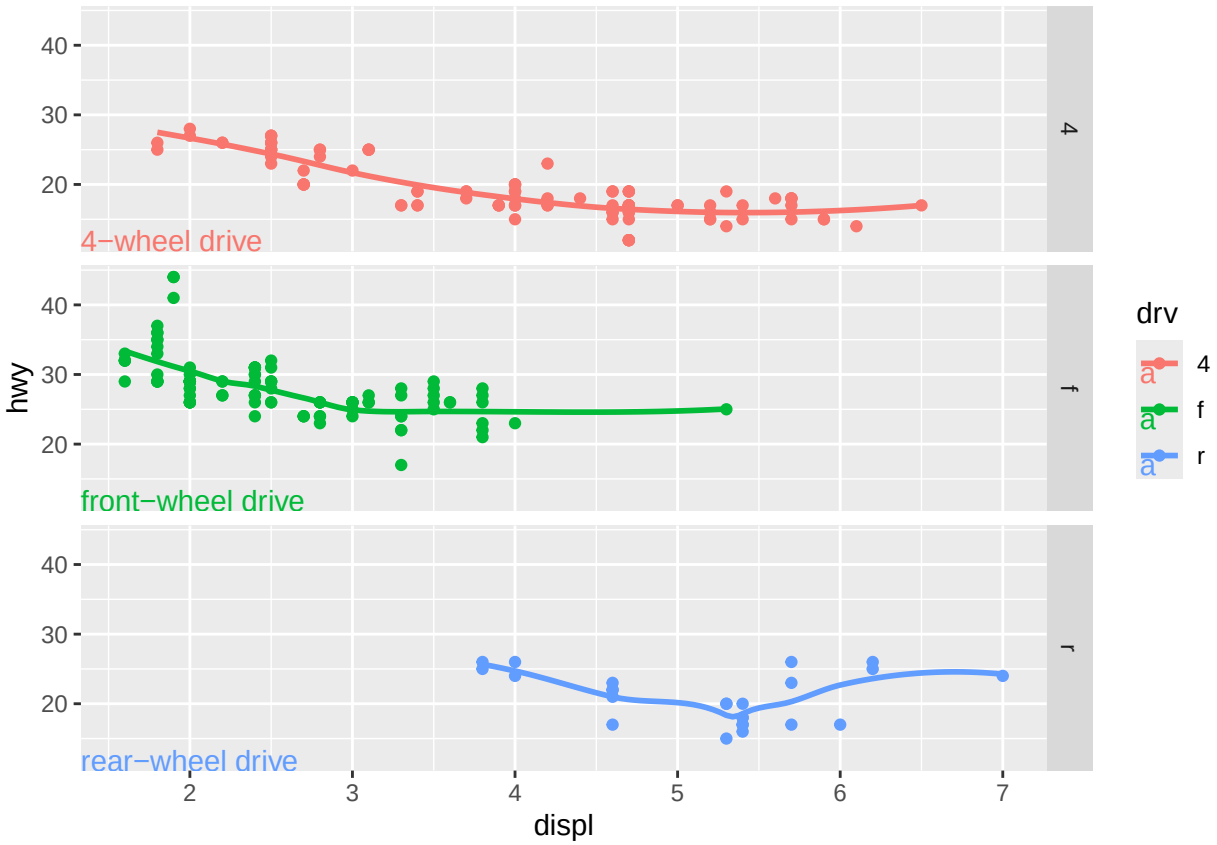


The text label gets "distributed" among the different facets as dictated by the specified coordinates.

To apply different text label to each facet, we could create a separate text data first then apply it to the plot like so:

```
text_data <- tibble(
  text_label = c("4-wheel drive", "front-wheel drive", "rear-wheel drive"),
  drv = c(4, "f", "r")
)

ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(se = FALSE) +
  facet_grid(drv ~ .) +
  geom_text(
    data = text_data,
    aes(x = -Inf, y = -Inf, label = text_label),
    hjust = "left", vjust = "bottom"
  )
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

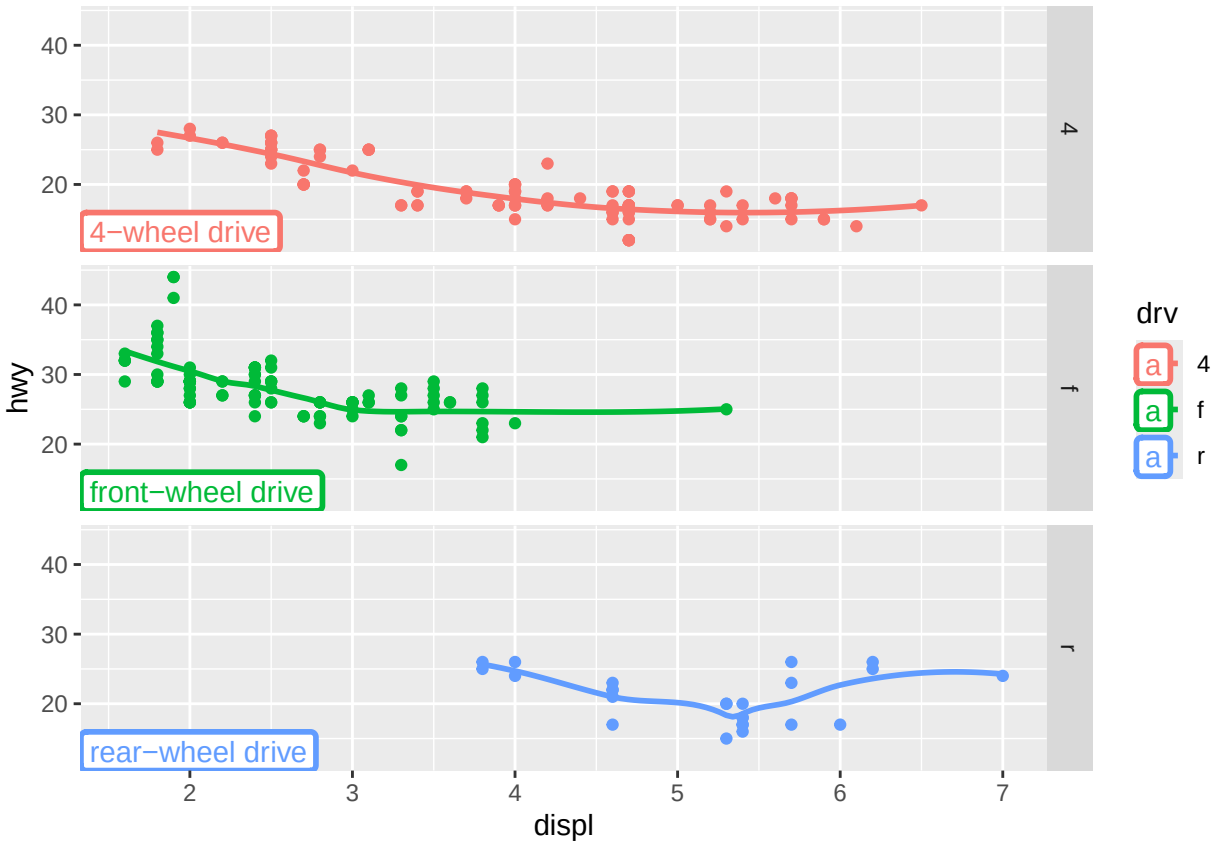


4. What arguments to `geom_label()` control the appearance of the background box?

Ans. The arguments in the `label.*` family control the appearance of the background box.

```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(se = FALSE) +
  facet_grid(drv ~ .) +
  geom_label(
    data = text_data,
    aes(x = -Inf, y = -Inf, label = text_label),
    hjust = "left", vjust = "bottom",
    label.size = 1
  )
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



5. What are the four arguments to `arrow()`? How do they work? Create a series of plots that demonstrate the most important options.

Ans. - `angle`: angle of the arrowhead in degrees
- `length`: length of the arrowhead - `ends`: [`"last"`, `"first"`, or `"both"`] indicates which endpoint to draw the arrowhead on
- `type`: [`"open"` or `"closed"`] indicates whether the arrowhead is a closed triangle or not (v-shaped)

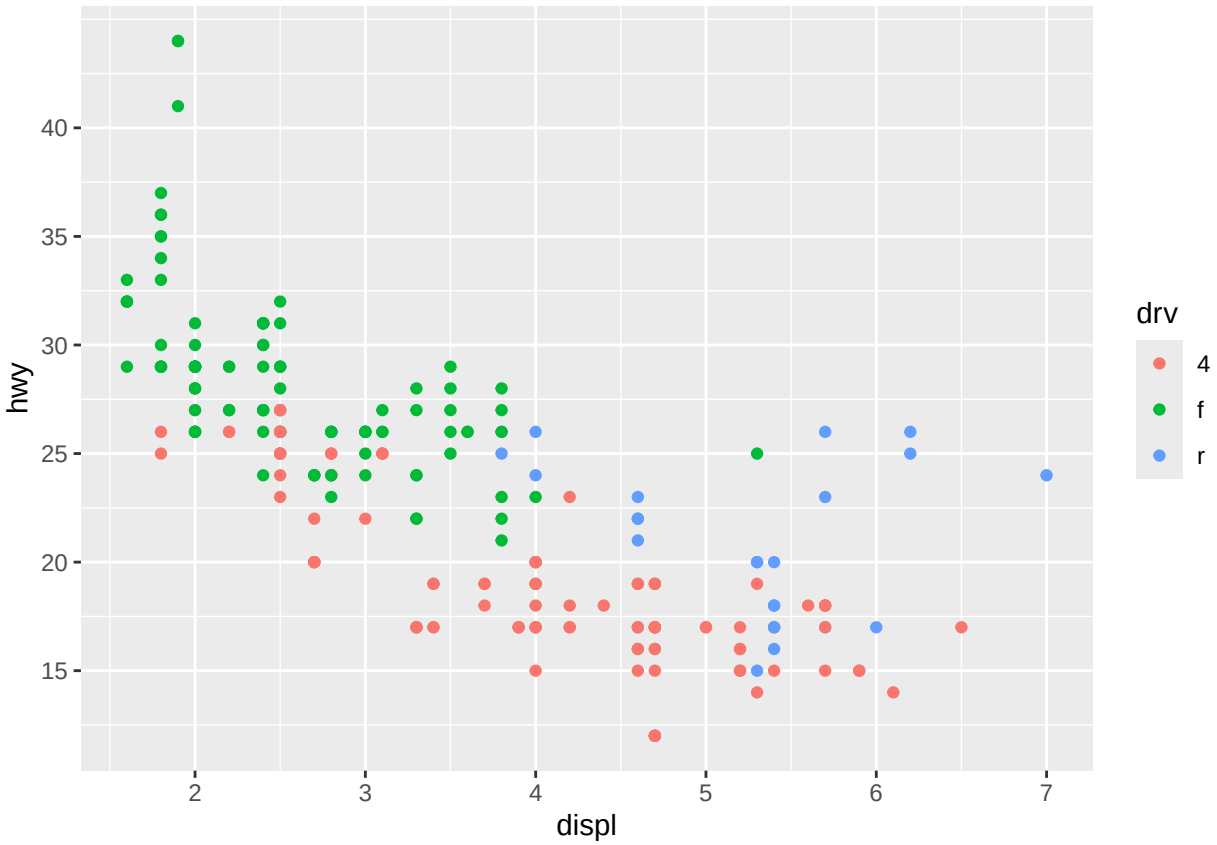
Scales

- ggplot2 automatically adds scales
- the automatic scaling can be overridden by specifying the `scale_*` argument

Axis ticks and legend keys

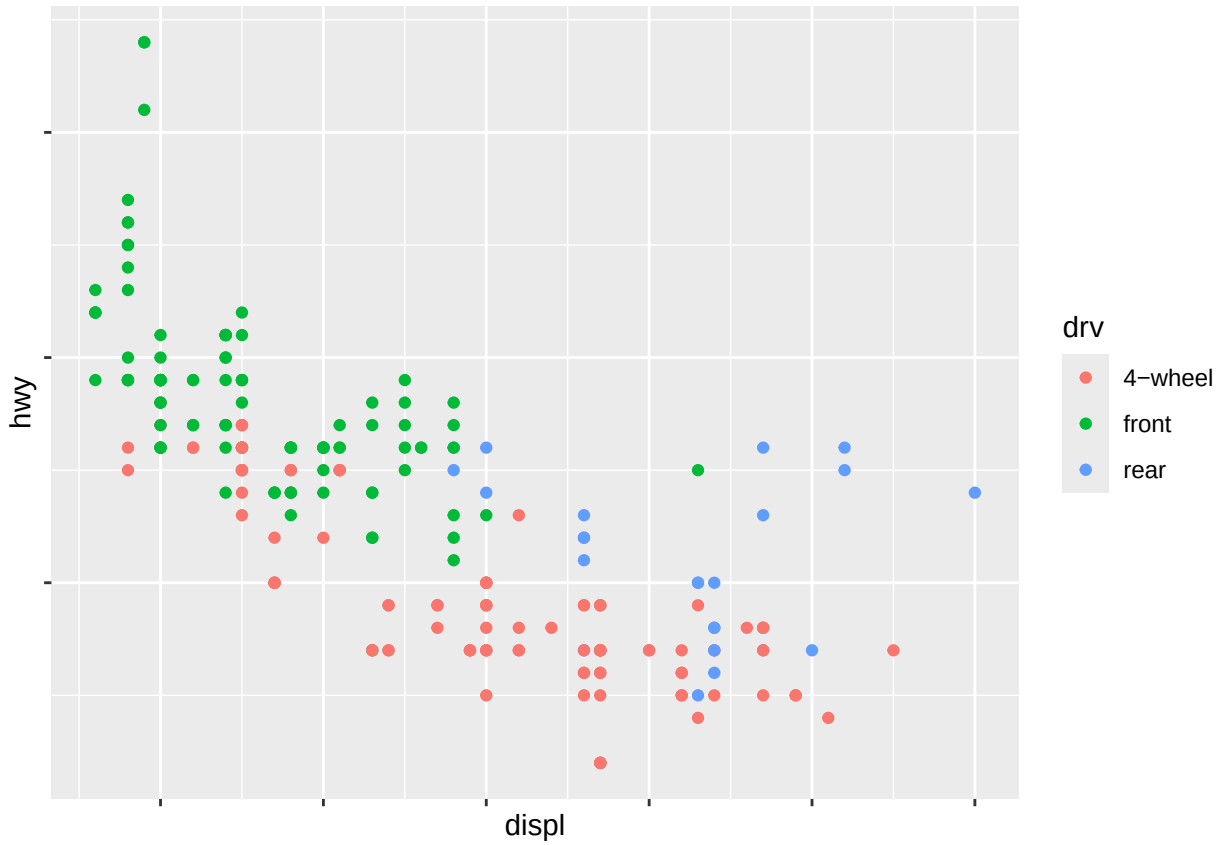
- axis ticks and legend keys are generally called **guides**
- axis ticks are controlled by `breaks` and `labels` arguments.

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



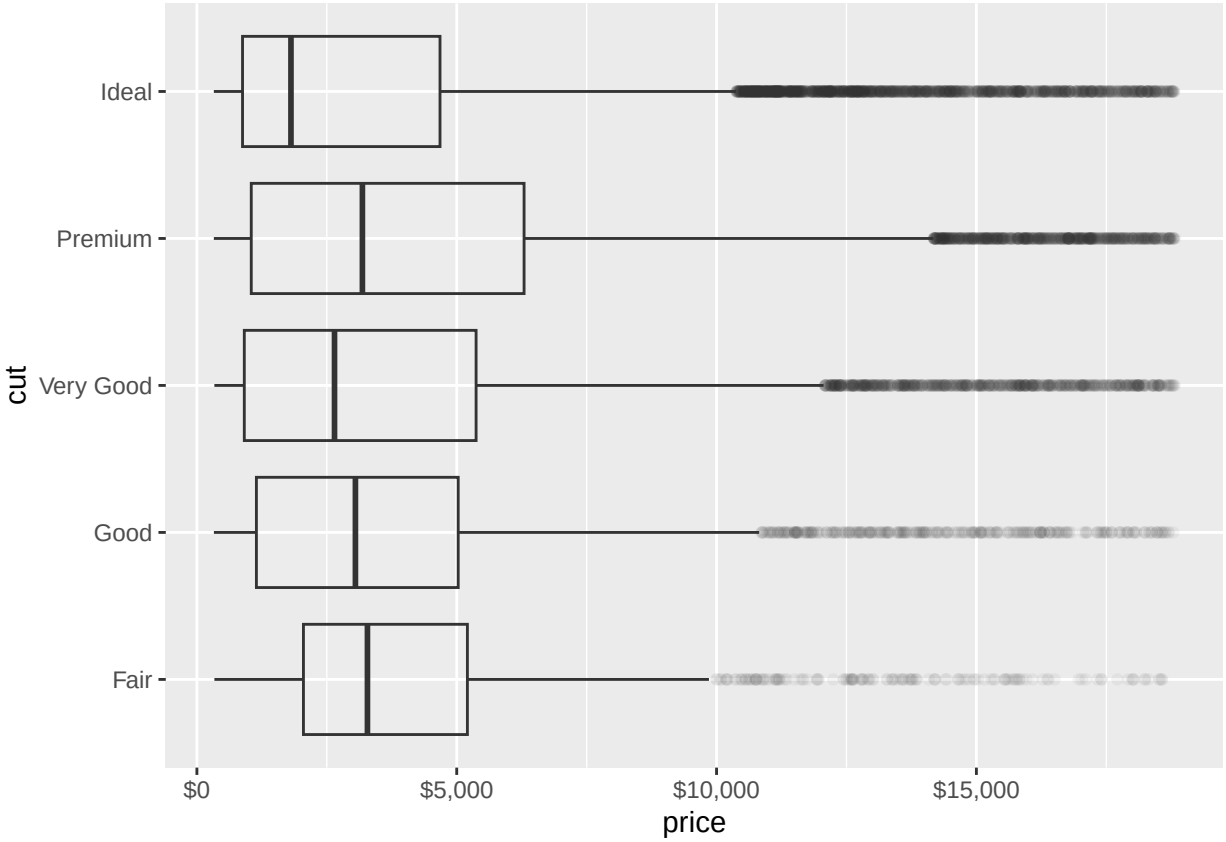
Using `labels` to suppress tick labels and modify legend keys:

```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  scale_x_continuous(labels = NULL) +
  scale_y_continuous(label = NULL) +
  scale_color_discrete(labels = c(
    "4" = "4-wheel",
    "f" = "front",
    "r" = "rear"
  ))
```



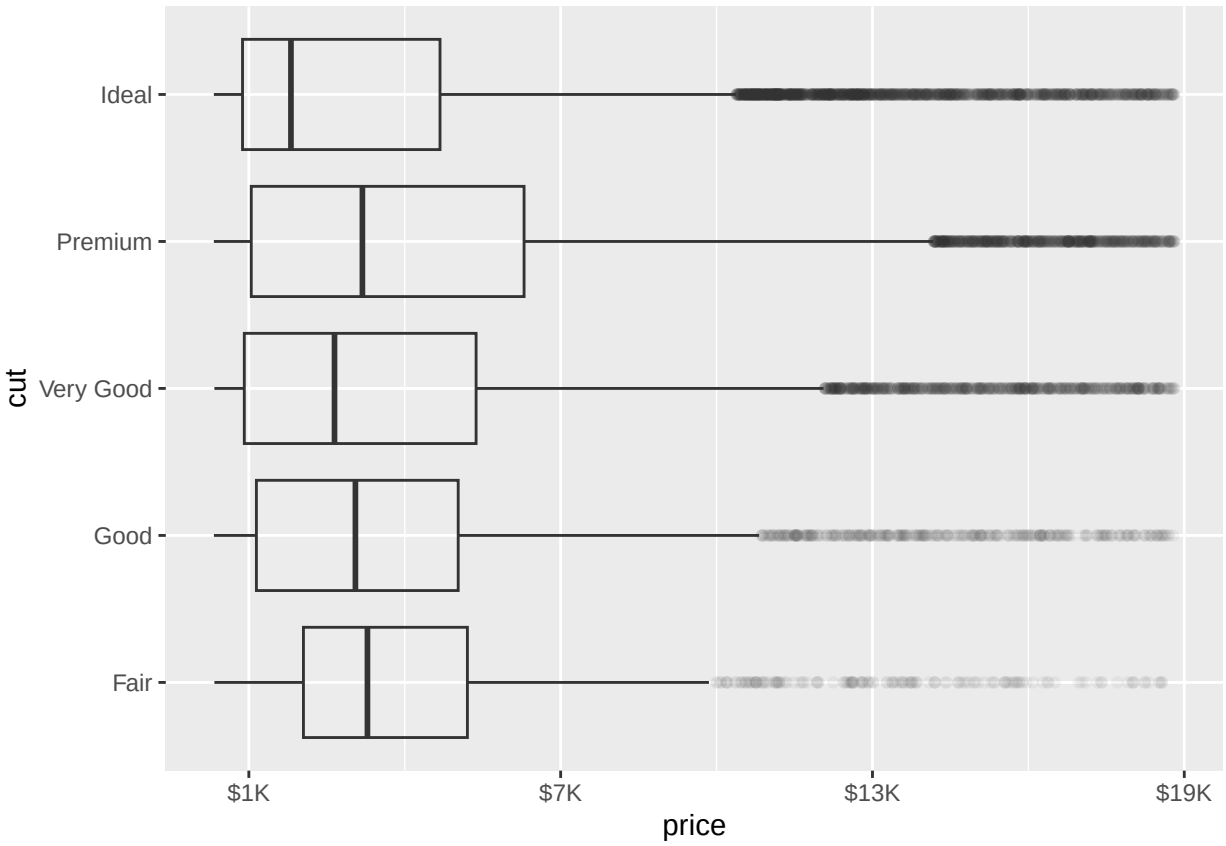
Using `label_dollar()` to customize number formatting to currency:

```
ggplot(diamonds, aes(x = price, y = cut)) +
  geom_boxplot(alpha = 0.05) +
  scale_x_continuous(labels = label_dollar())
```



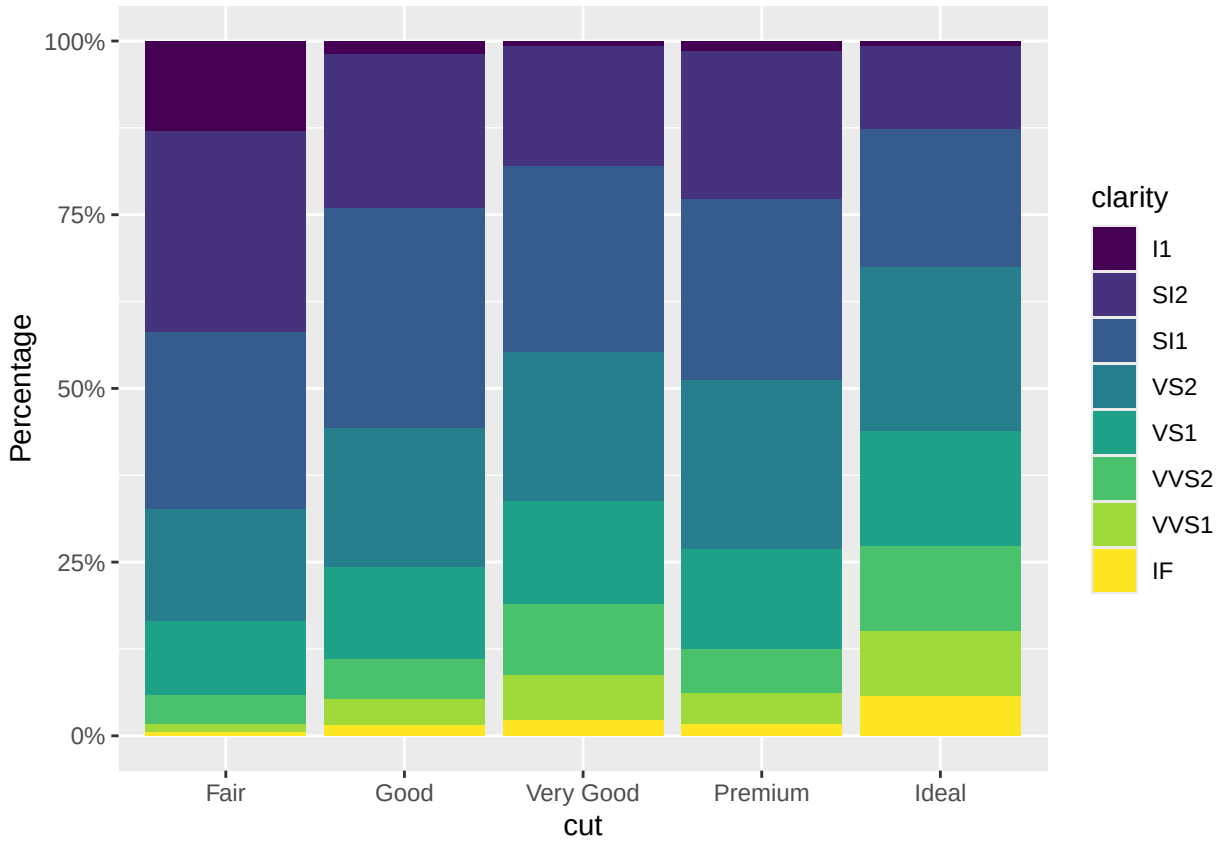
Using `suffix` to customize the units of the numerical data:

```
ggplot(diamonds, aes(x = price, y = cut)) +
  geom_boxplot(alpha = 0.05) +
  scale_x_continuous(
    labels = label_dollar(scale = 1/1000, suffix = "K"),
    breaks = seq(1000, 19000, by = 6000)
  )
```



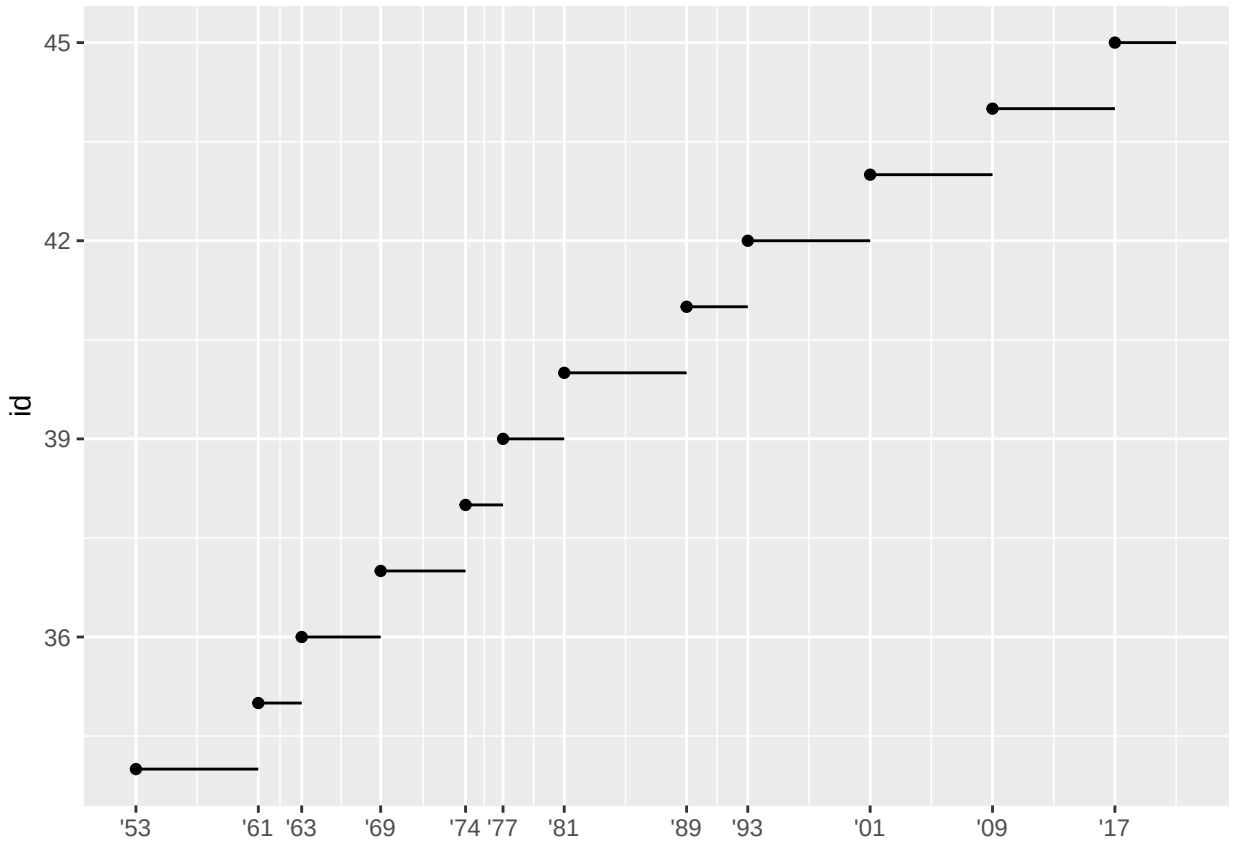
Using `label_percent()`:

```
ggplot(diamonds, aes(x = cut, fill = clarity)) +
  geom_bar(position = "fill") +
  scale_y_continuous(
    name = "Percentage",
    labels = label_percent()
  )
```



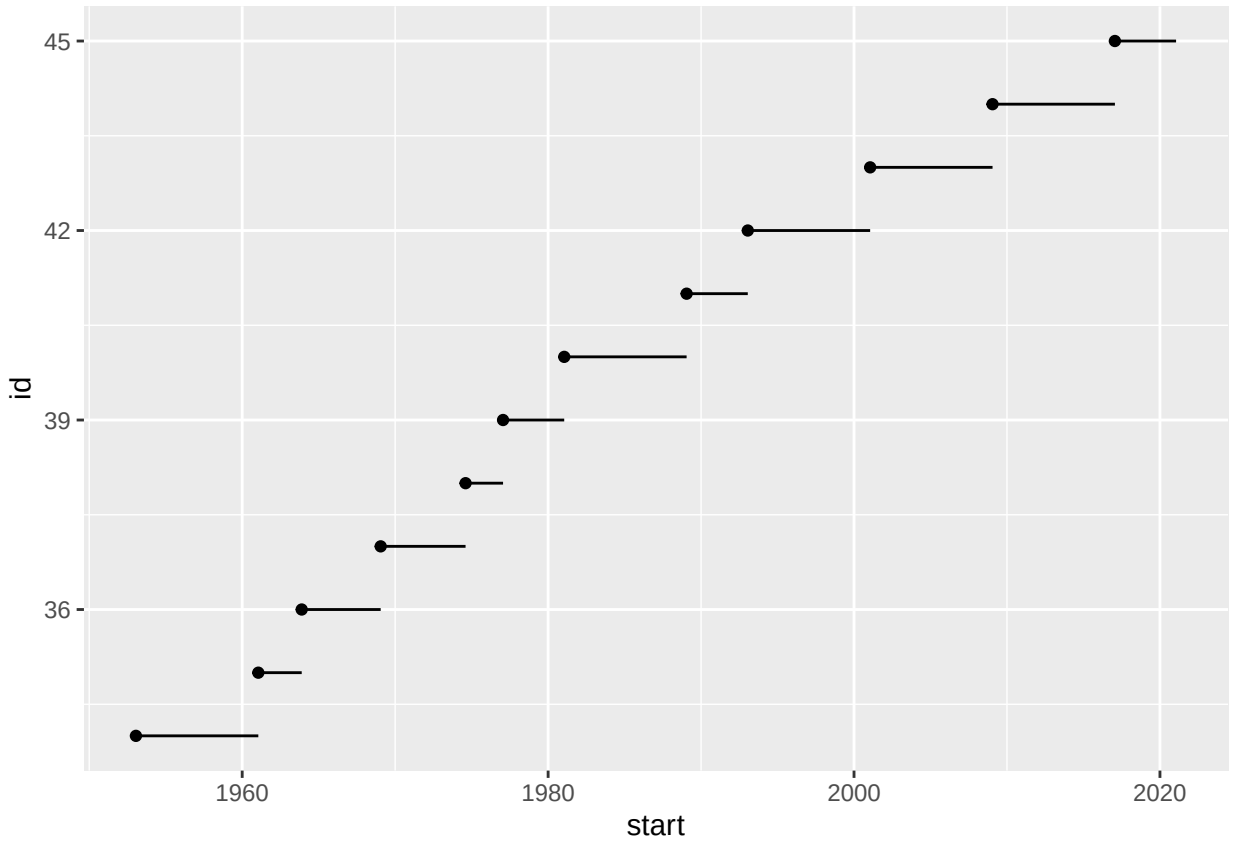
Using breaks to emphasize where exactly the observations occur:

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



Compare this plot with the "normal" plot:

```
presidential |>
  mutate(id = 33 + row_number()) |>
  ggplot(aes(x = start, y = id)) +
  geom_point() +
  geom_segment(aes(xend = end, yend = id))
```



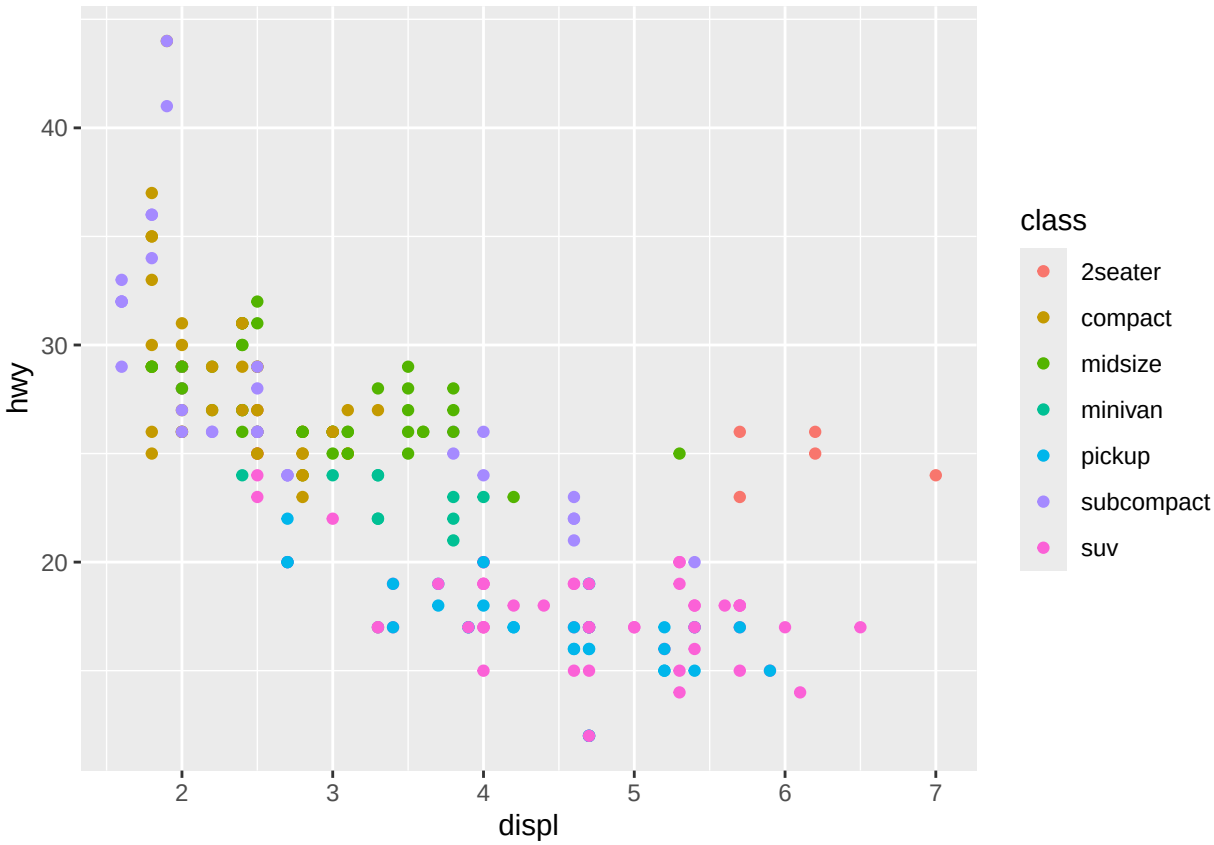
Legend layout

- legend layout can be controlled using a `theme()` setting:

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

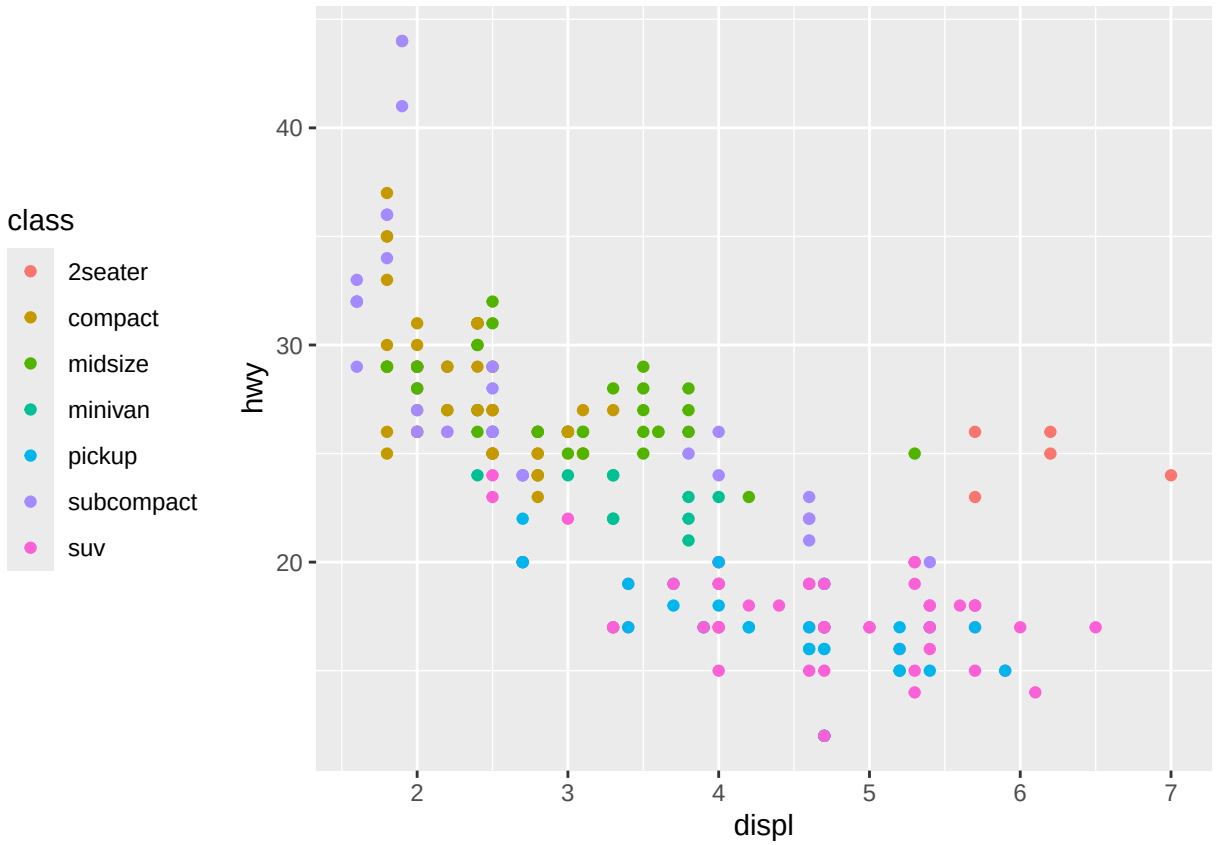
Positioning the legend on the right (the default):

```
base + theme(legend.position = "right")
```



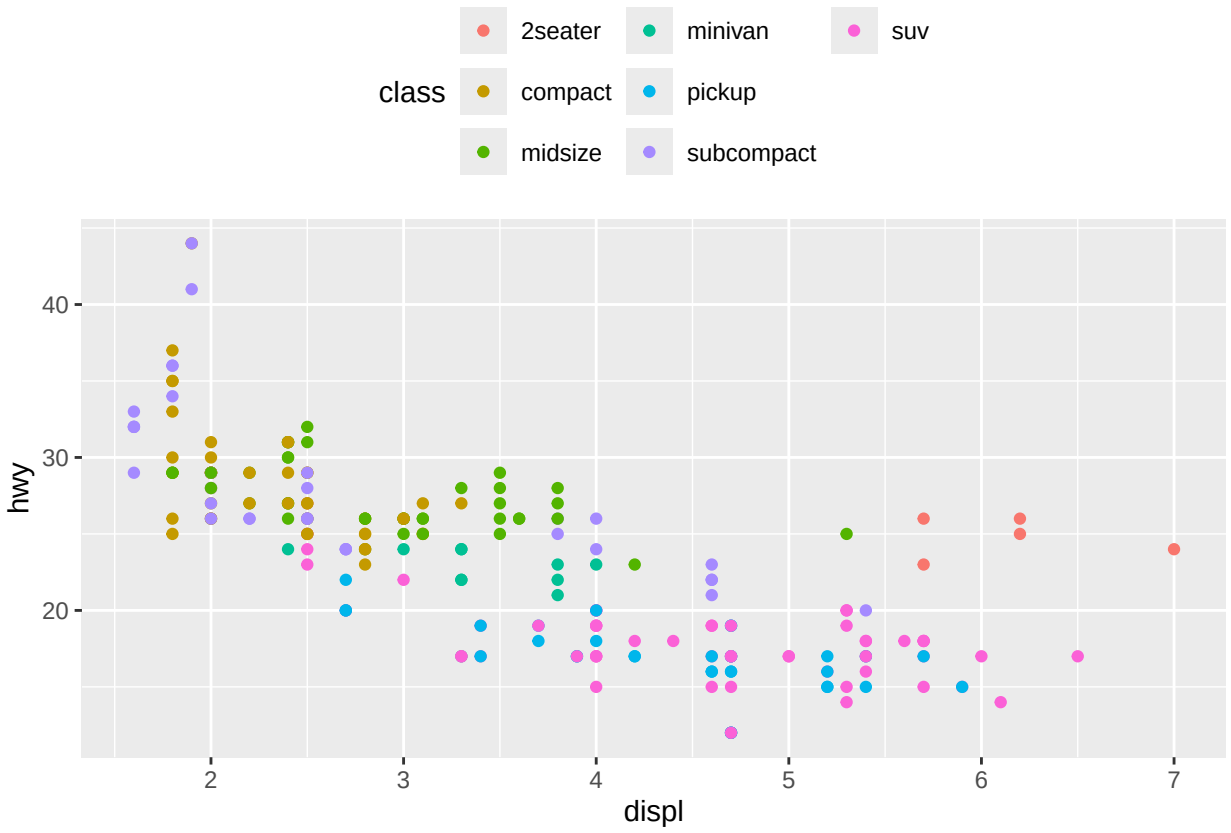
Positioning the legend on the left:

```
base + theme(legend.position = "left")
```



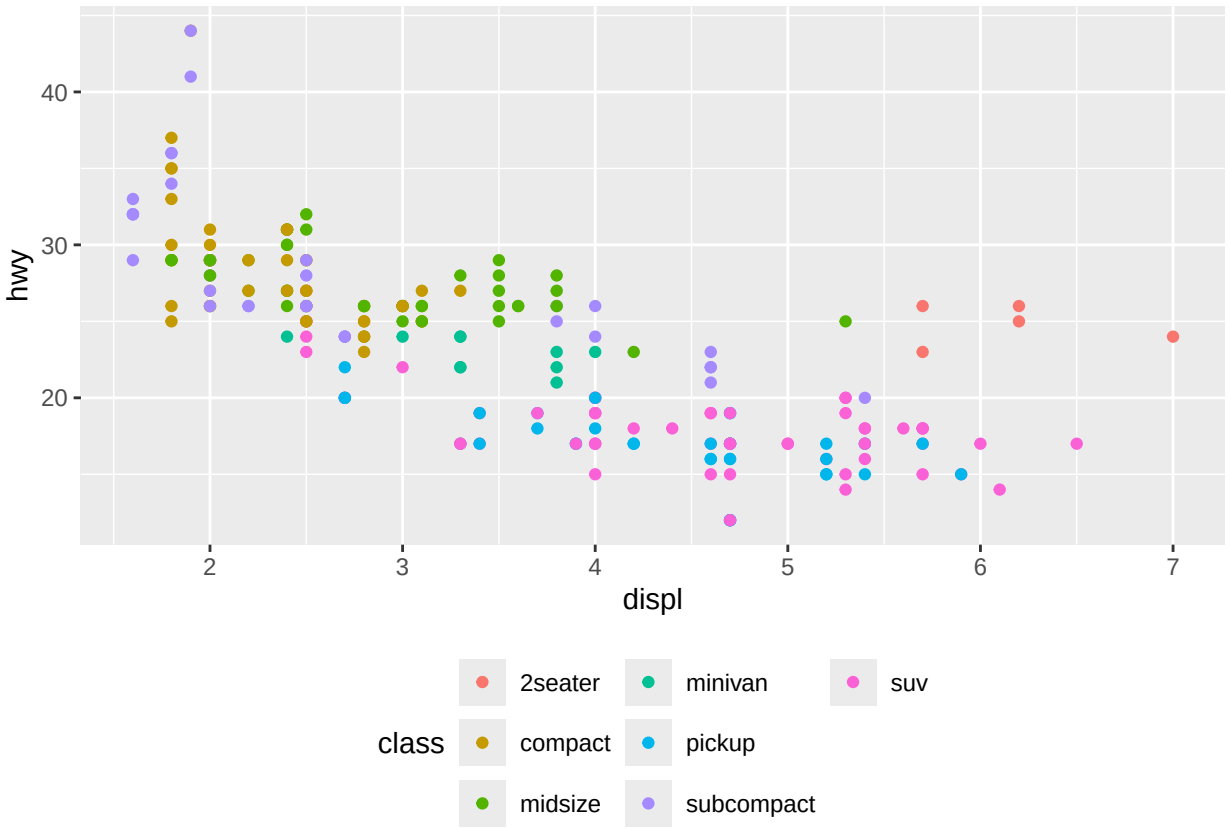
Positioning the legend at the top:

```
base +
  theme(legend.position = "top") +
  guides(color = guide_legend(nrow = 3))
```



Positioning the legend at the bottom:

```
base +
  theme(legend.position = "bottom") +
  guides(color = guide_legend(nrow = 3))
```



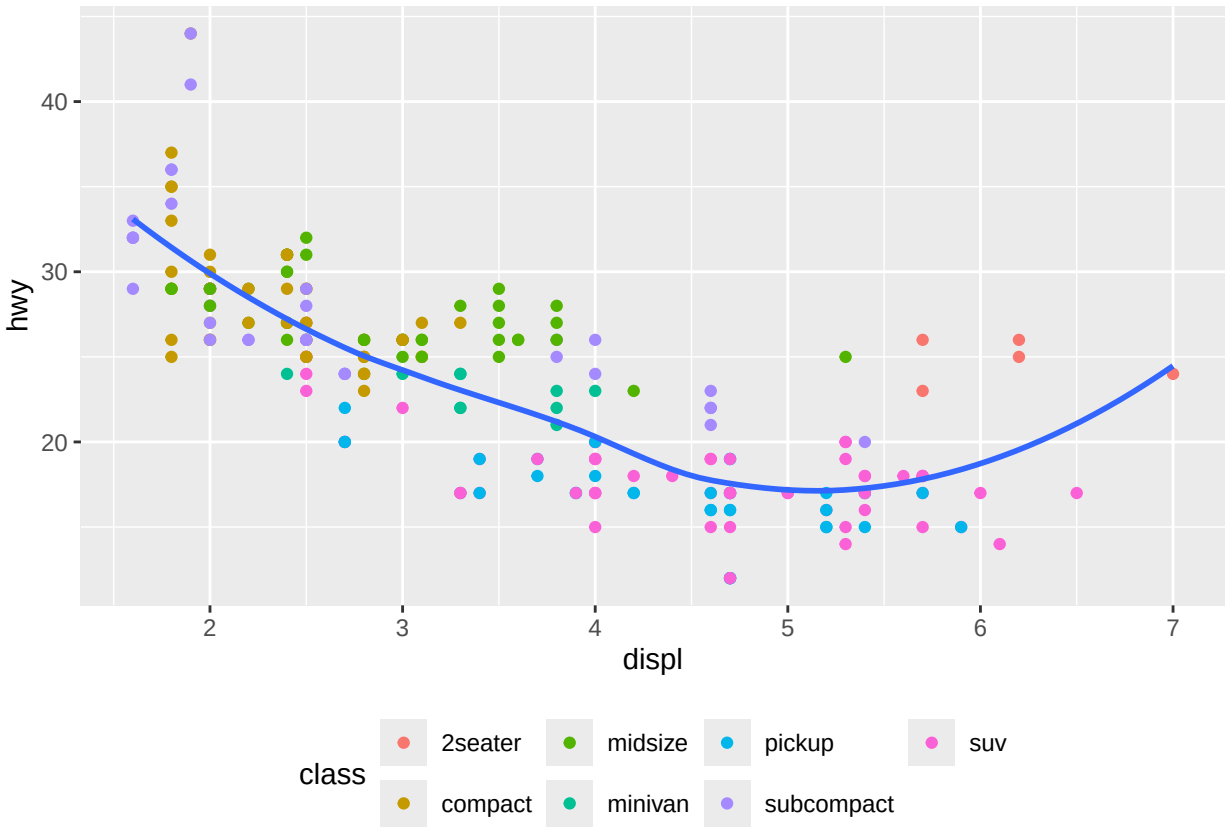
As shown in other examples above, we can use `legend.position = "none"` to suppress the appearance of the legend.

If we want finer control of the legend appearance, we can use `guides()` along with `guide_legend()` and/or `guide_colorbar()`:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  theme(legend.position = "bottom") +
  guides(
    guide_legend(nrow = 2, override.aes = list(size = 4))
  )
```

```
## Warning: Guides provided to `guides()` must be named.
## i All guides are unnamed.
```

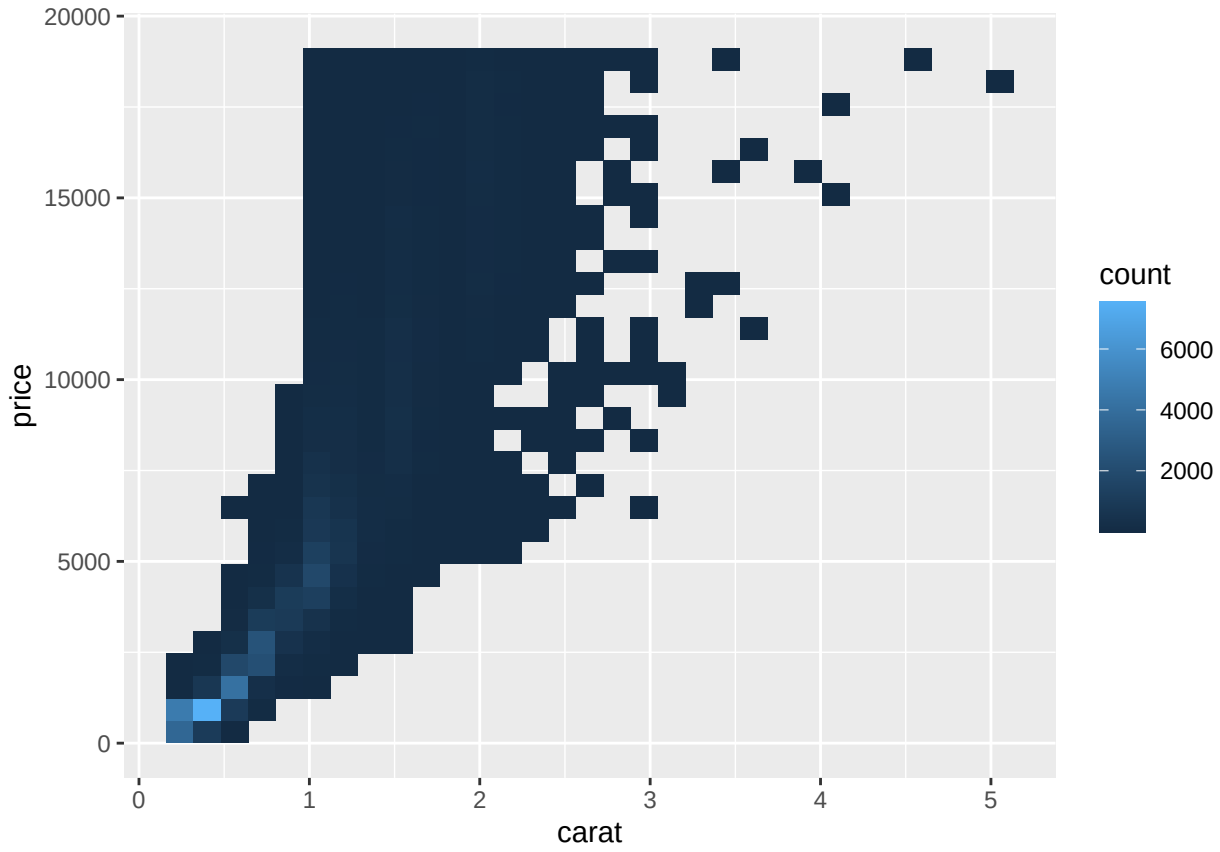
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Replacing a scale

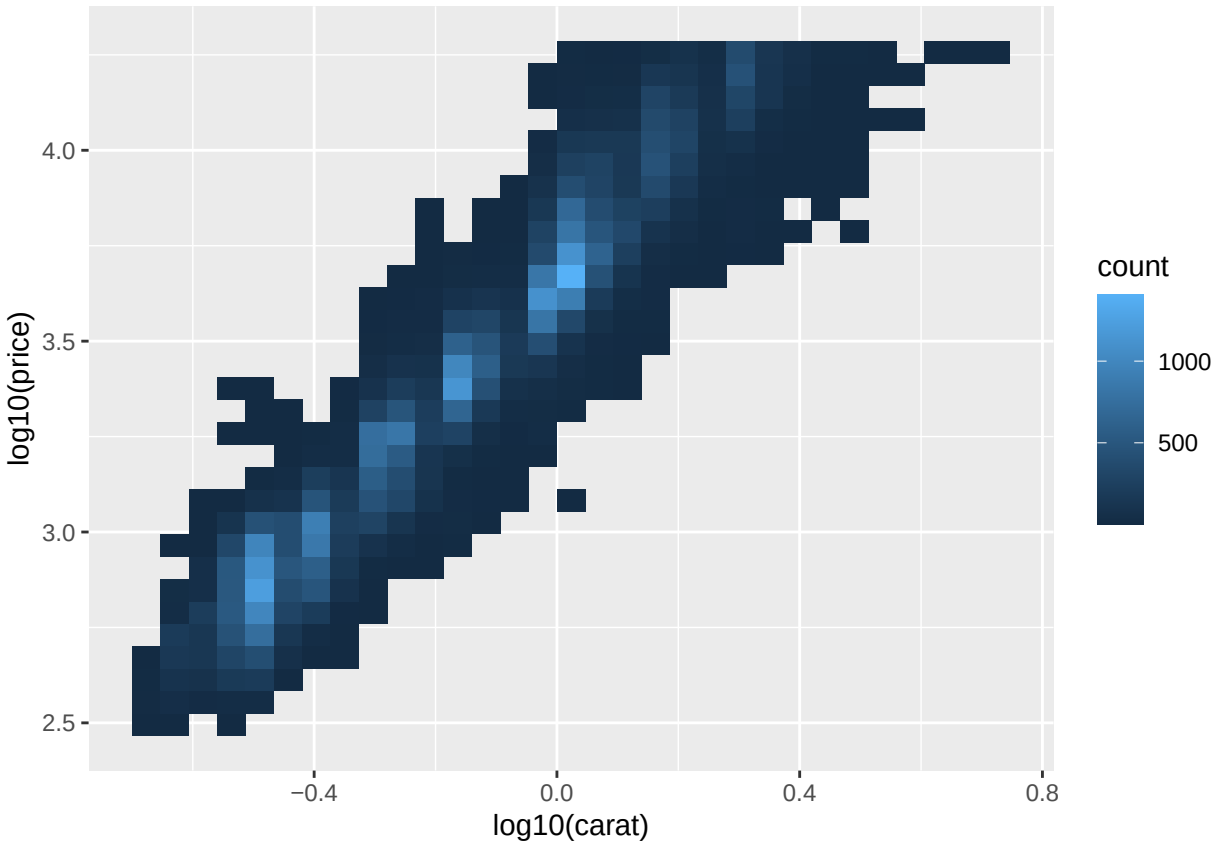
"Regular" plot:

```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_bin2d()
```



Log-transformed version of the plot above:

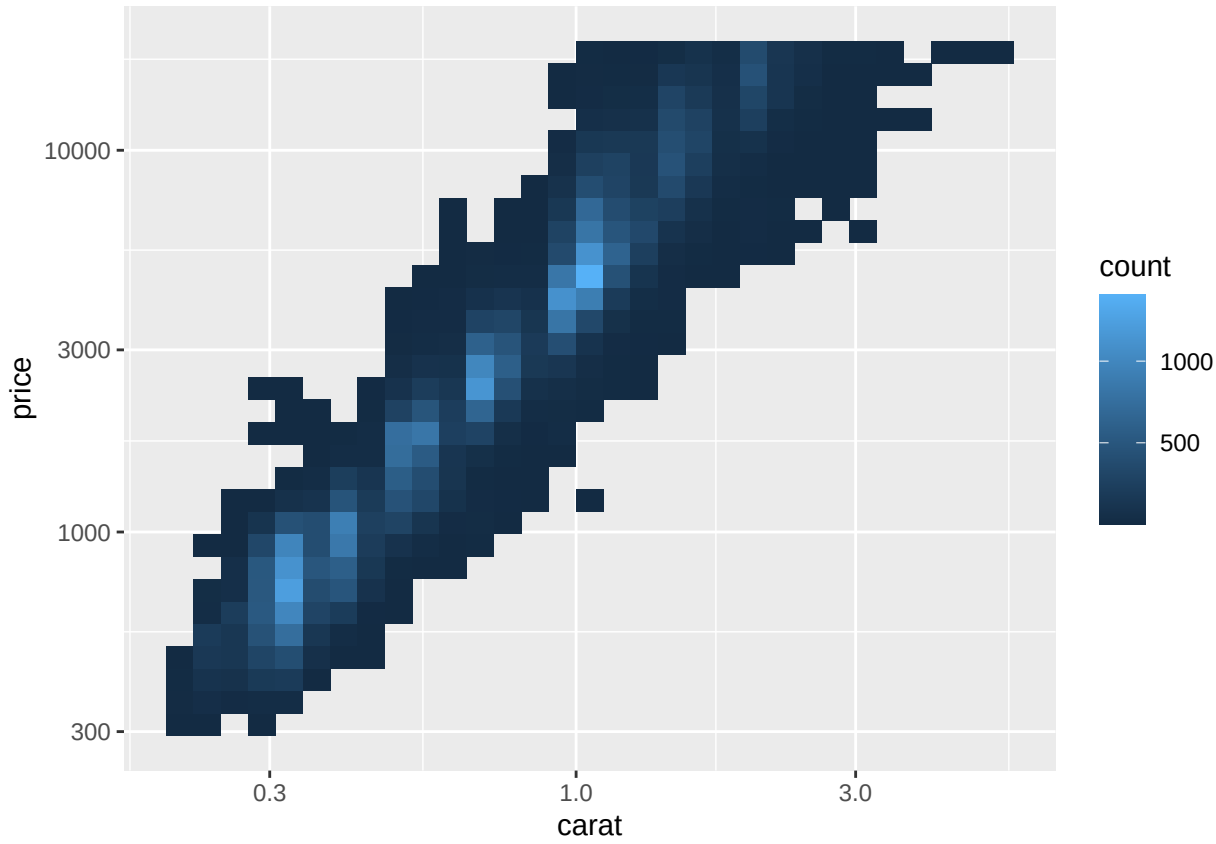
```
ggplot(diamonds, aes(x = log10(carat), y = log10(price))) +
  geom_bin2d()
```



- Doing it this way, the axes are now labeled with the transformed values.
- This makes it hard to interpret the plot sometimes.

Better approach by changing the scale of the axes:

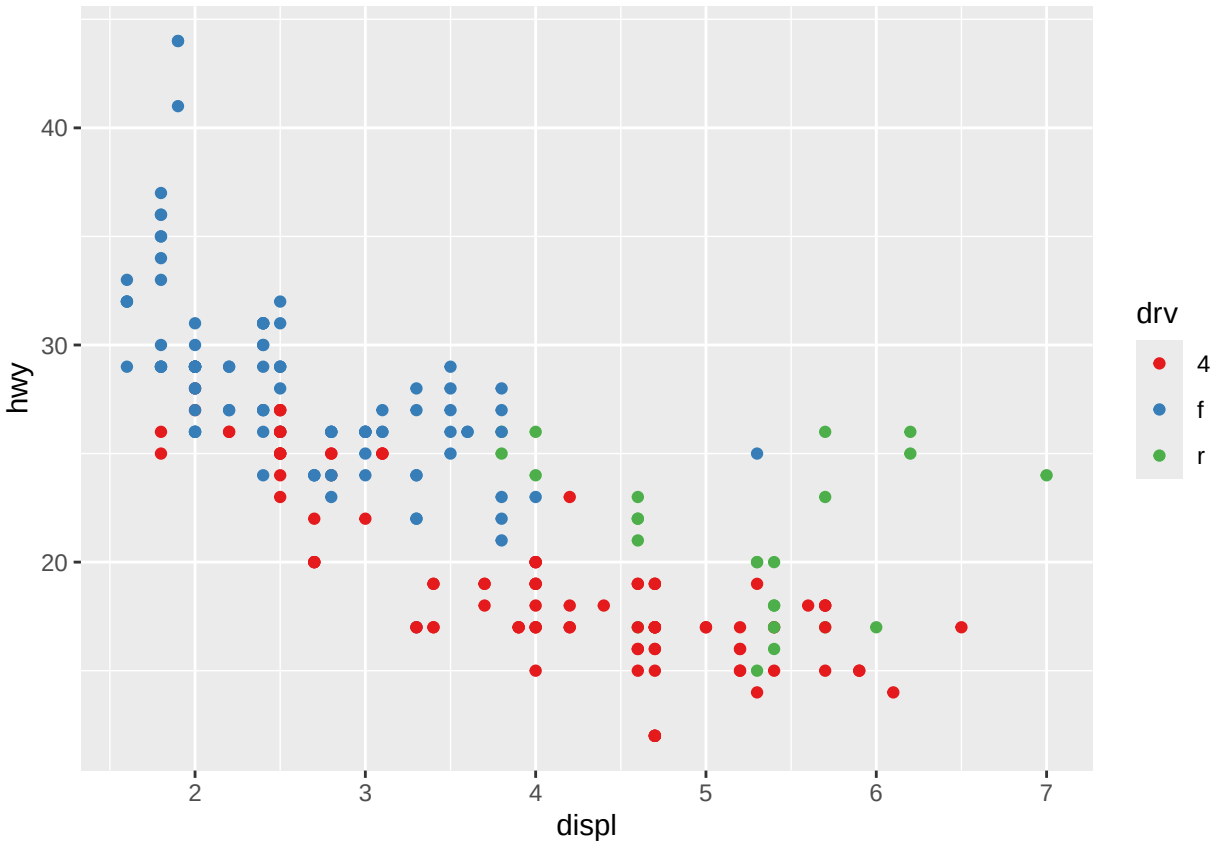
```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_bin2d() +
  scale_x_log10() +
  scale_y_log10()
```



The plot is visually identical to the previous one, but the axes are now labelled on the original data scale.

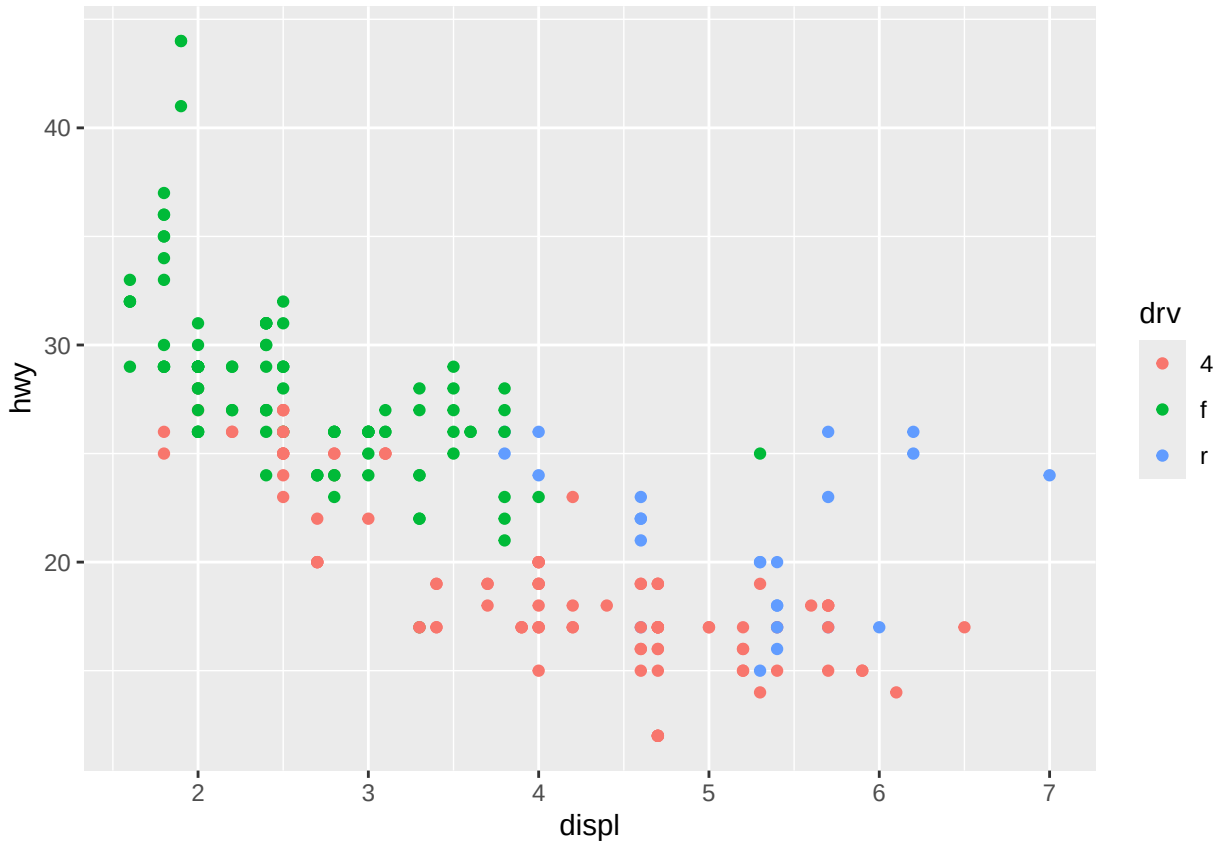
"Scaling" the colors –> picking colors that are evenly spaced around the color wheel:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = drv)) +
  scale_color_brewer(palette = "Set1")
```



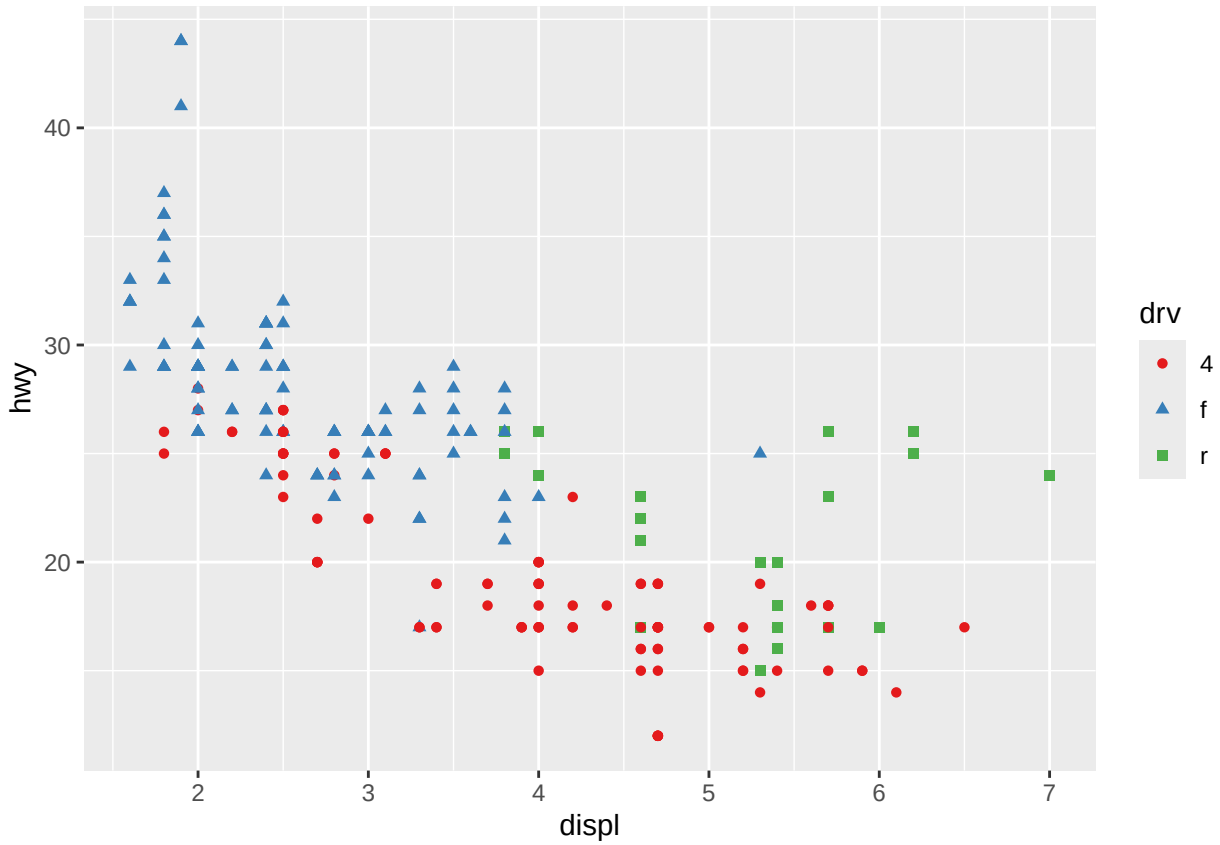
Compare this with the default color values:

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



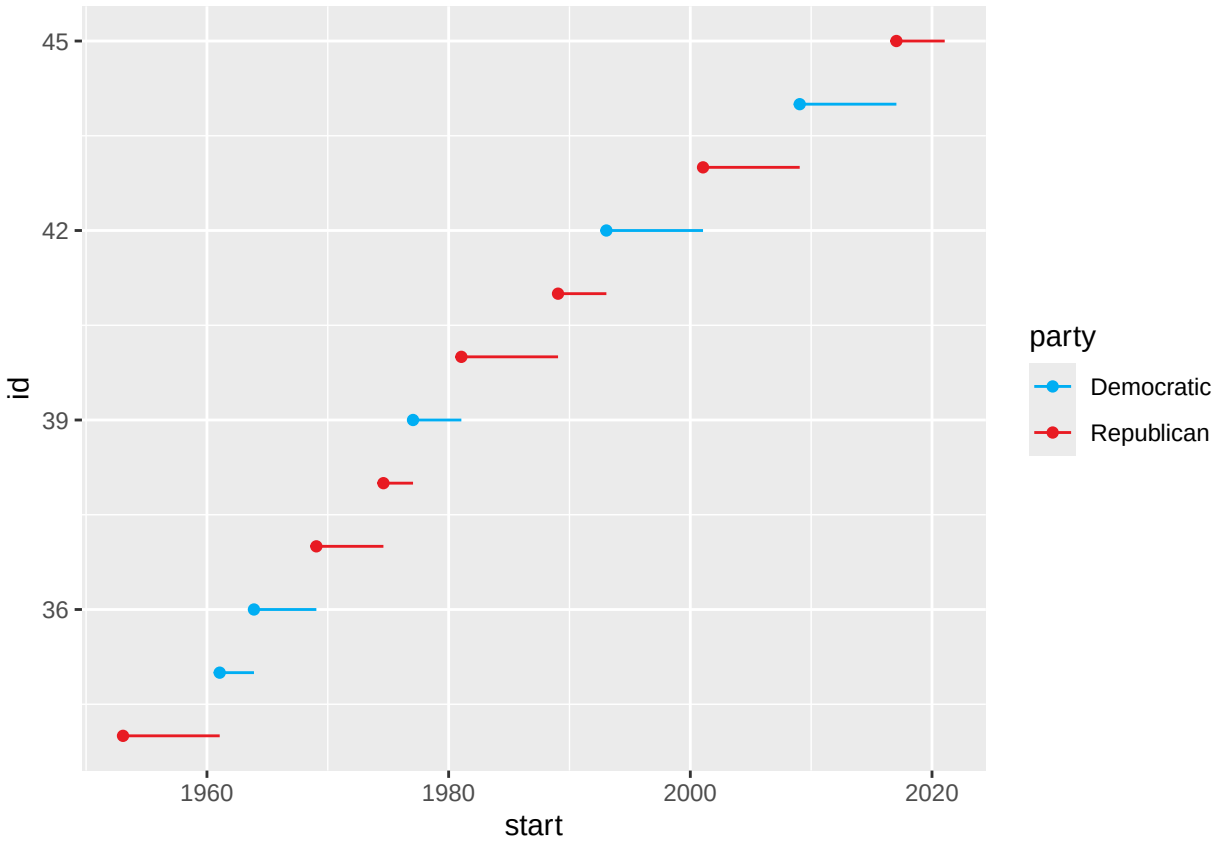
- The ColorBrewer scales have been tuned to work better for people with common types of color blindness.
- Adding a `shape` mapping can help ensure that the plot is interpretable even when rendered in black and white.

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = drv, shape = drv)) +
  scale_color_brewer(palette = "Set1")
```



Assigning a redefined color mapping using `color_scale_manual()`:

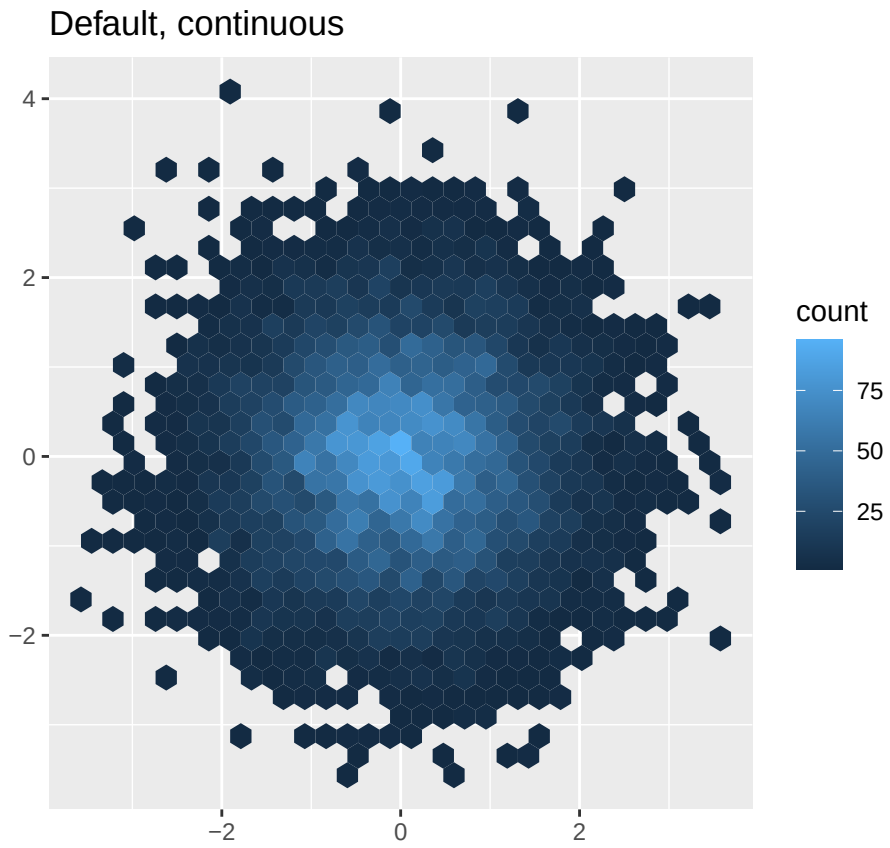
```
presidential |>
  mutate(id = 33 + row_number()) |>
  ggplot(aes(x = start, y = id, color = party)) +
  geom_point() +
  geom_segment(aes(xend = end, yend = id)) +
  scale_color_manual(
    values = c(Republican = "#E81B23", Democratic = "#00AEF3")
  )
```



- For continuous color, the functions `scale_color_gradient()` or `scale_fill_gradient()` can be used.
- for diverging scale, there is `scale_color_gradient2()`. - the viridis color scale was designed by Natheniel Smith and Stefan van der Walt to cater for people with various forms of color blindness.
- this (the viridis color scale) is also useful for making the plots interpretable even when rendered in black and white.

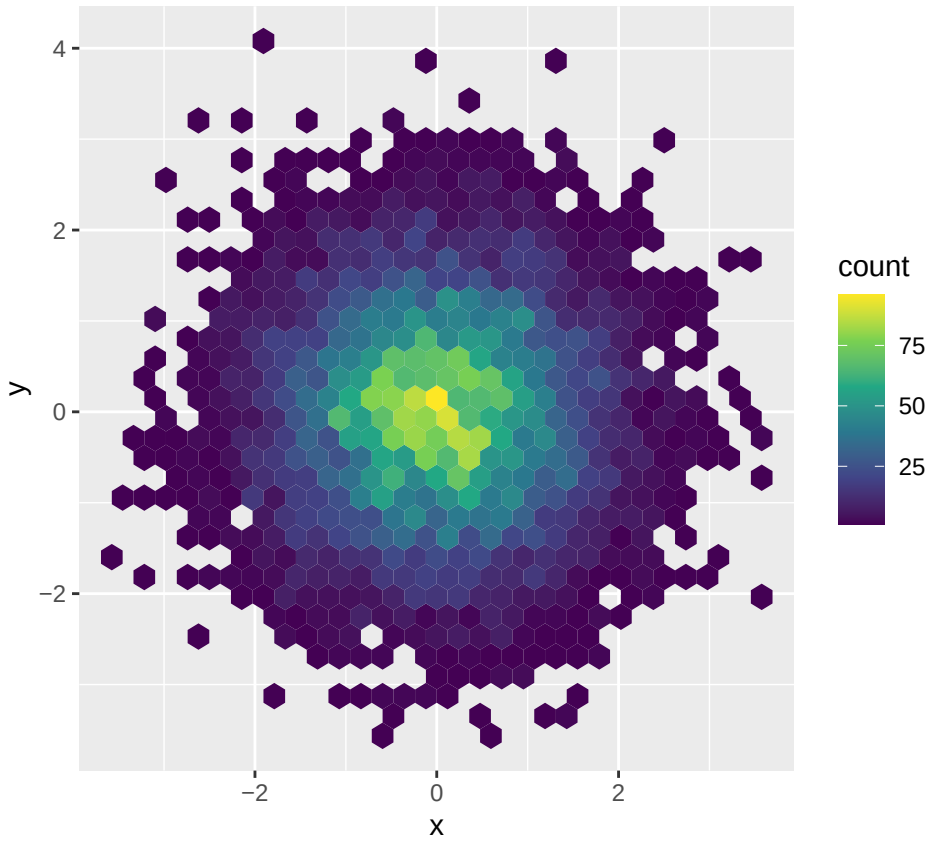
Plot using the default continuous color scheme:

```
df <- tibble(
  x = rnorm(10000),
  y = rnorm(10000)
)

# Default colors
ggplot(df, aes(x, y)) +
  geom_hex() +
  coord_fixed() +
  labs(title = "Default, continuous", x = NULL, y = NULL)
```



Same data, plotted using “viridis, continuous” color scale:

```
ggplot(df, aes(x, y)) +
  geom_hex() +
  coord_fixed() +
  scale_fill_viridis_c()
```

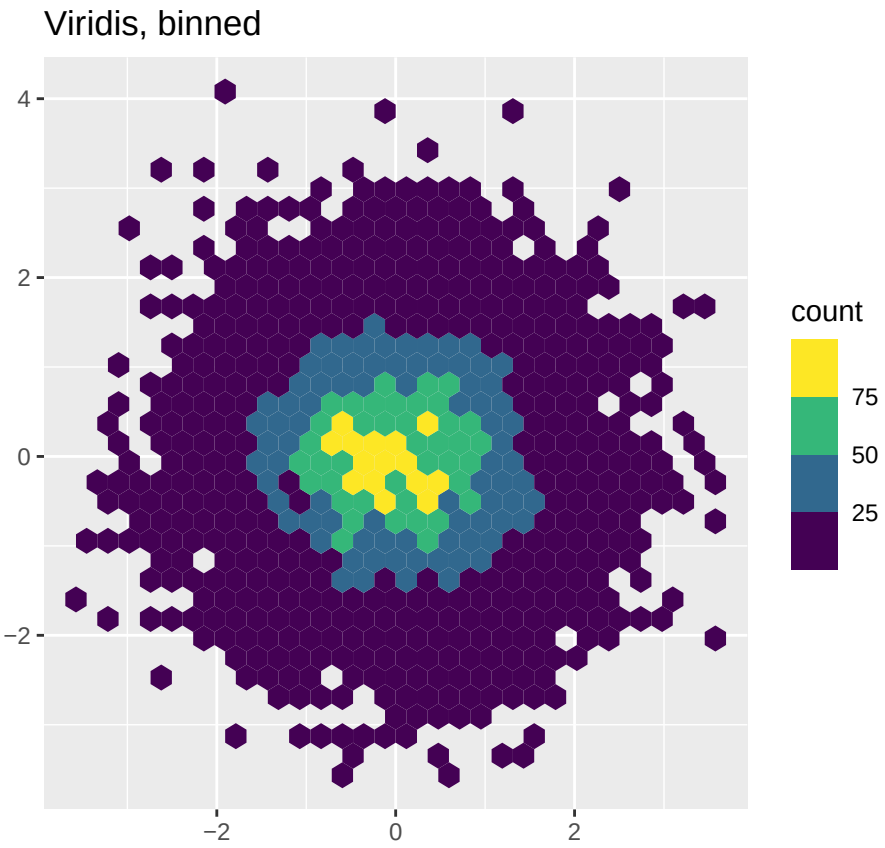



```
  labs(title = "Viridis, continuous", x = NULL, y = NULL)
```

```
## $x
## NULL
##
## $y
## NULL
##
## $title
## [1] "Viridis, continuous"
##
## attr(,"class")
## [1] "labels"
```

Same data, plotted using “viridis, continuous” color scale:

```
ggplot(df, aes(x, y)) +
  geom_hex() +
  coord_fixed() +
  scale_fill_viridis_b() +
  labs(title = "Viridis, binned", x = NULL, y = NULL)
```



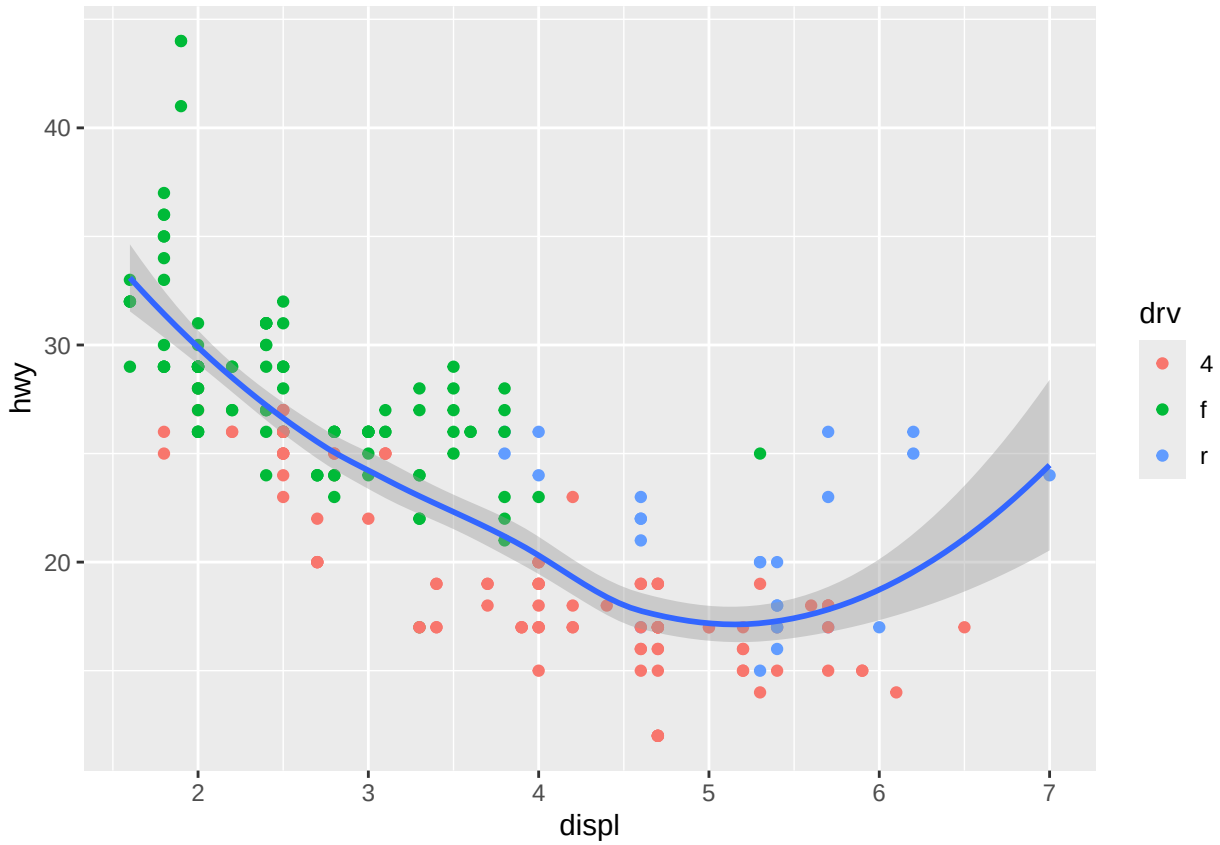
Zooming

Three ways of controlling the plot limits: 1. Adjusting what data are plotted. 2. Setting the limits in each scale. 3. Setting the `xlim` and `ylim` in `coord_cartesian()`.

Plot of entire data:

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

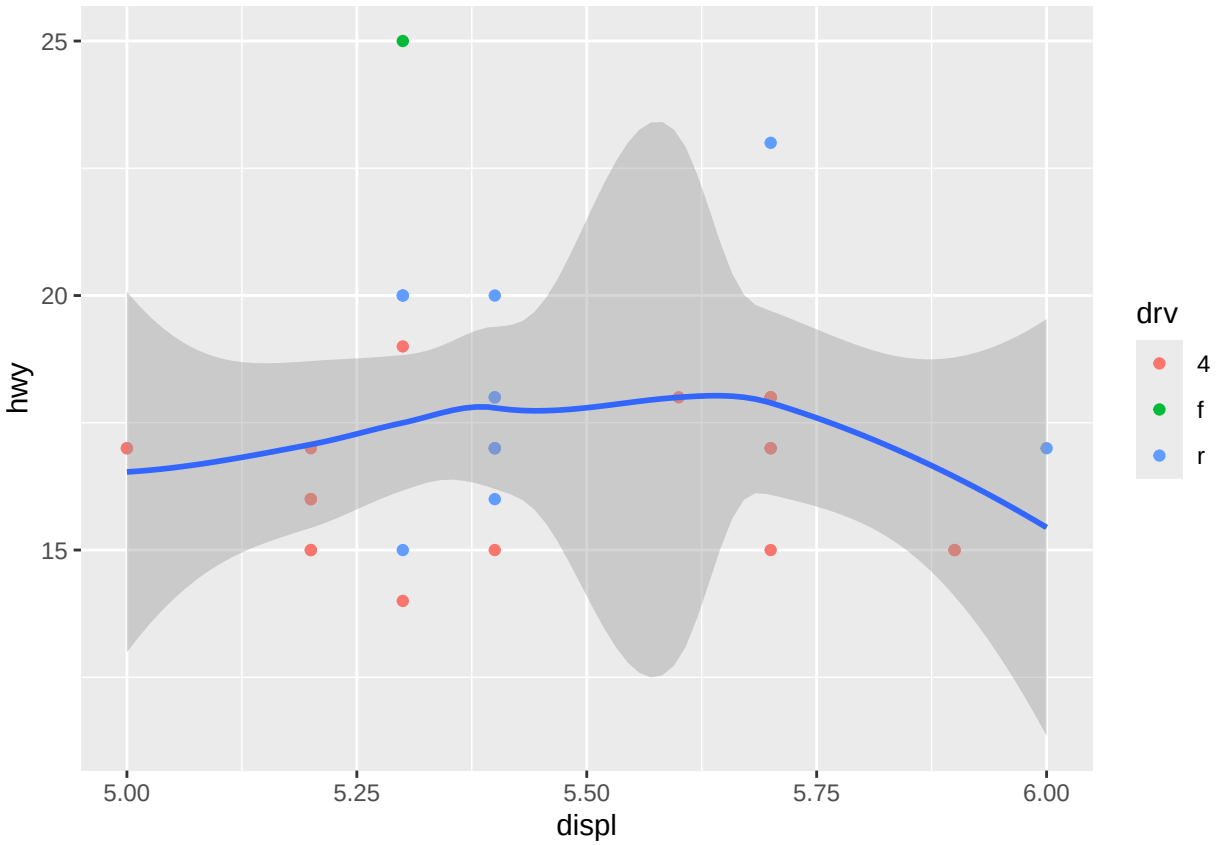
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Zoomed-in plot by filtering the data (No. 1)

```
mpg |>
  filter(displ >= 5 & displ <= 6 & hwy >= 10 & hwy <= 25) |>
  ggplot(aes(x = displ, y = hwy)) +
  geom_point(aes(color = drv)) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



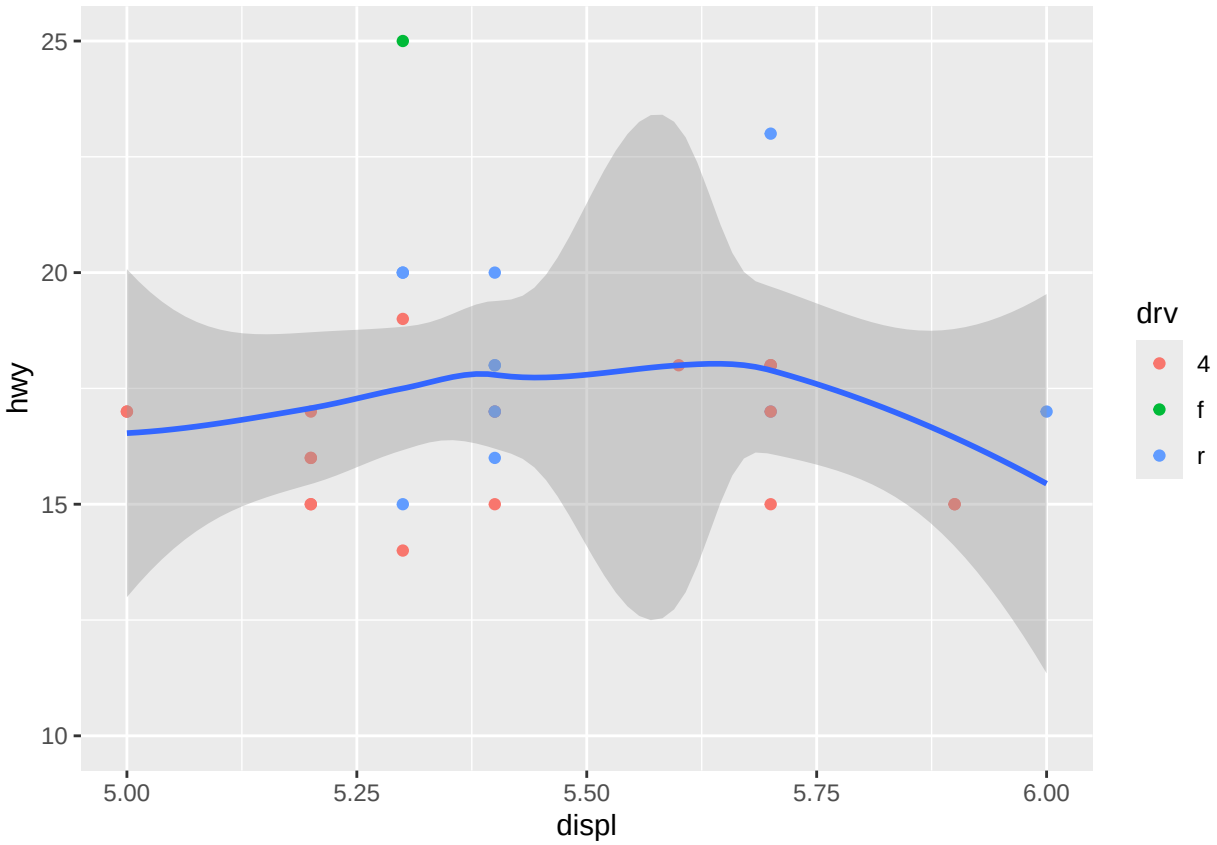
Zoomed-in plot by setting the limits in the scaling of the x and y coordinates (No. 2):

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = drv)) +
  geom_smooth() +
  scale_x_continuous(limits = c(5, 6)) +
  scale_y_continuous(limits = c(10, 25))
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
## Warning: Removed 202 rows containing non-finite outside the scale range
## (`stat_smooth()`).
```

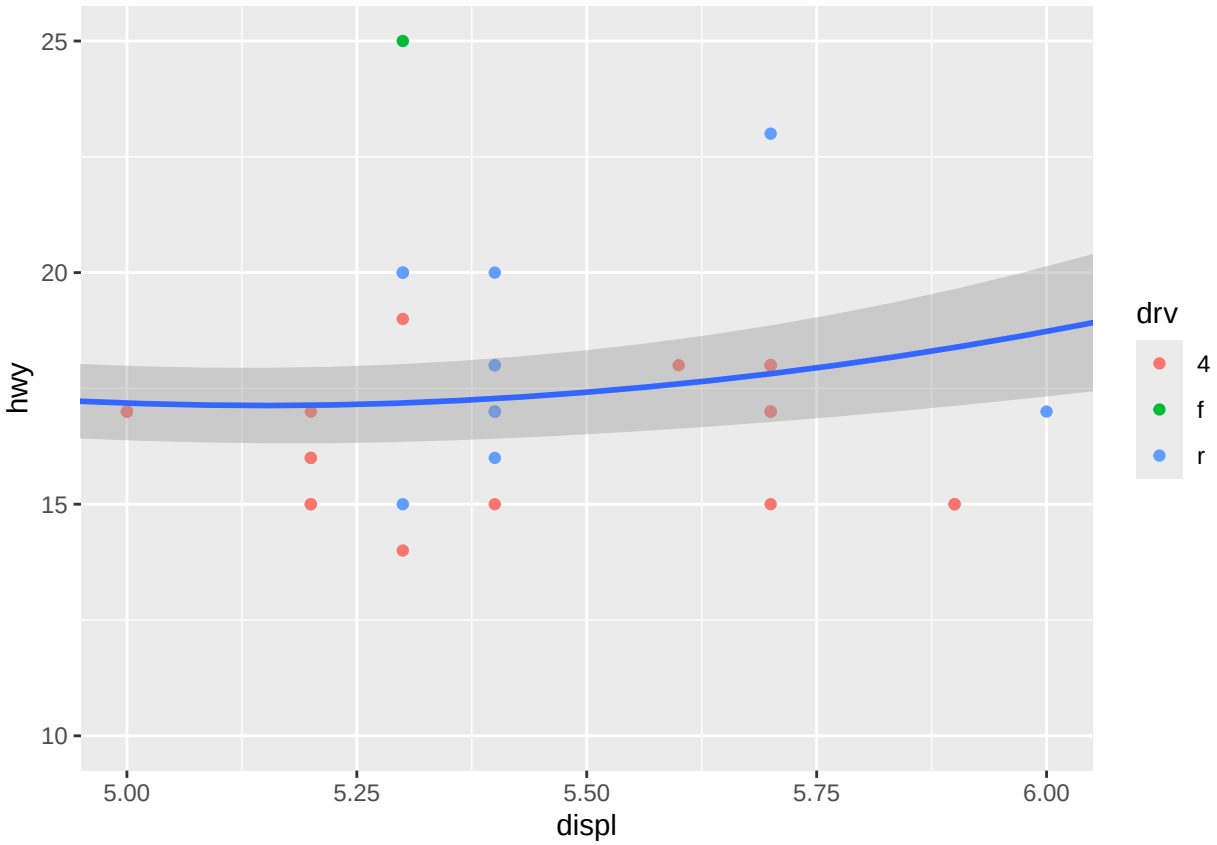
```
## Warning: Removed 202 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



Zoomed-in plot bby setting the `xlim` and `ylim` parameters in `coord_cartesian()` (No. 3):

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = drv)) +
  geom_smooth() +
  coord_cartesian(xlim = c(5, 6), ylim = c(10, 25))
```

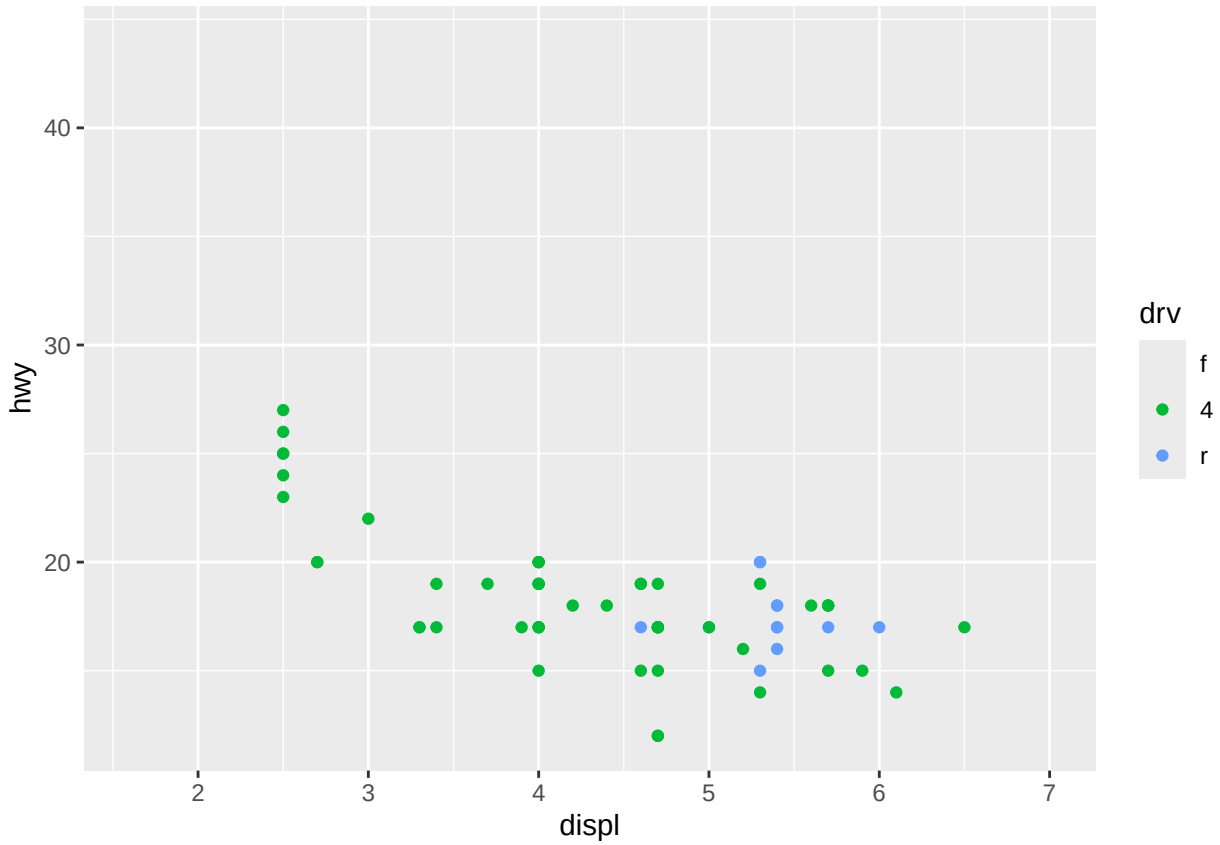
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



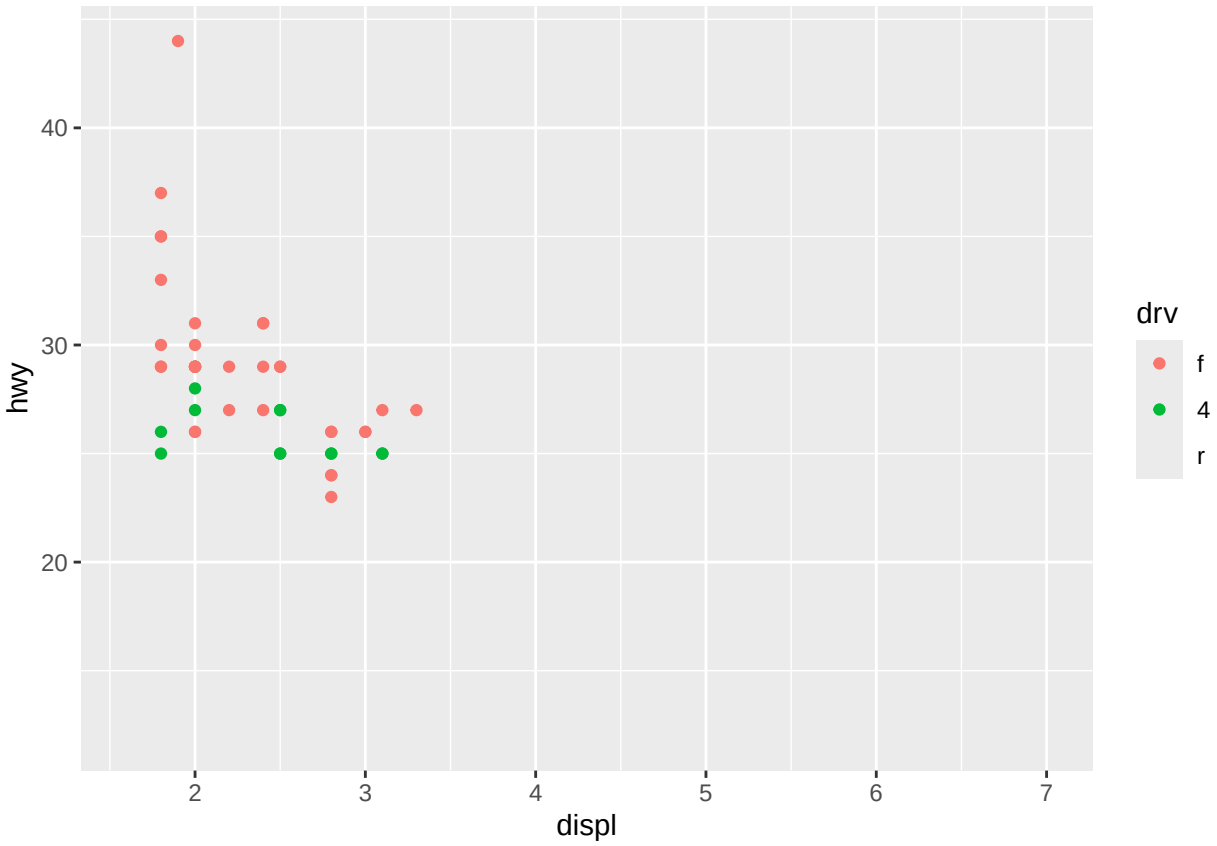
Which method to use? - generally, setting the limits on individual scales is more useful if you want to expand the limits to match the scales across different plots.
- however, setting the limits on individual scales is akin to subsetting the data. If you want to retain all the data, it is generally best to use `coord_cartesian()`

Example of plotting with expanded limits to match scales of different plots:

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

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

ggplot(suv, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  x_scale +
  y_scale +
  col_scale
```



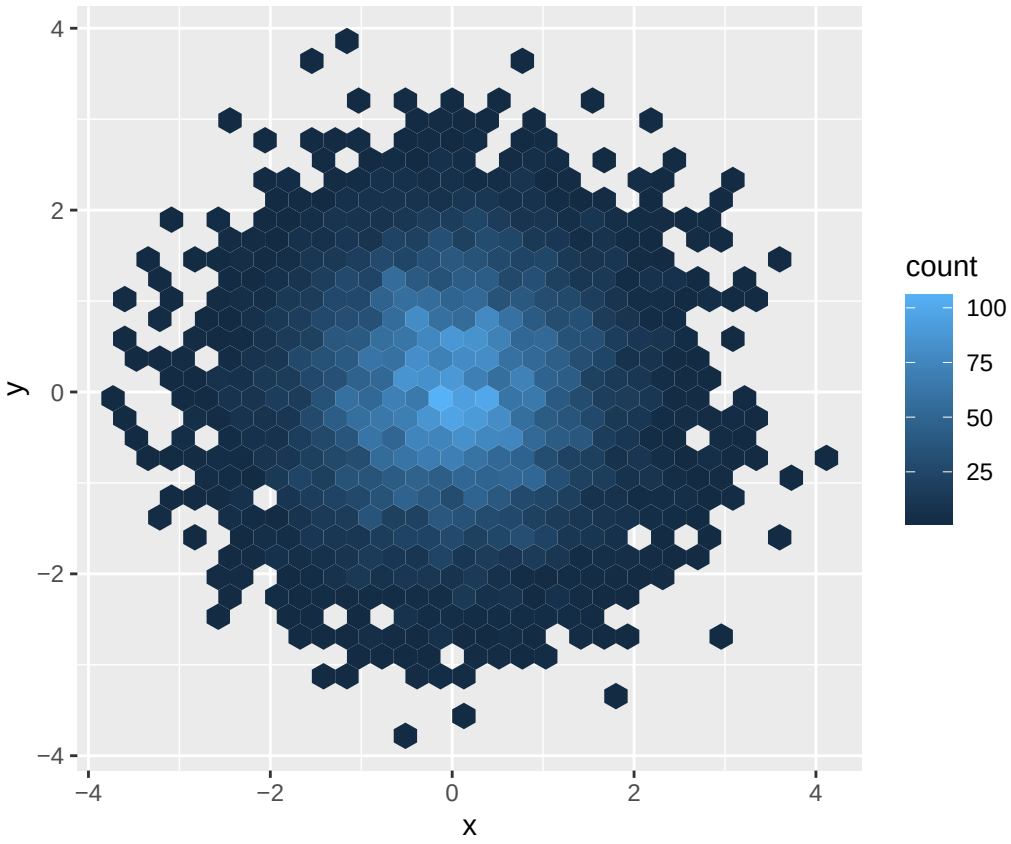

```
ggplot(compact, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  x_scale +
  y_scale +
  col_scale
```



Exercises

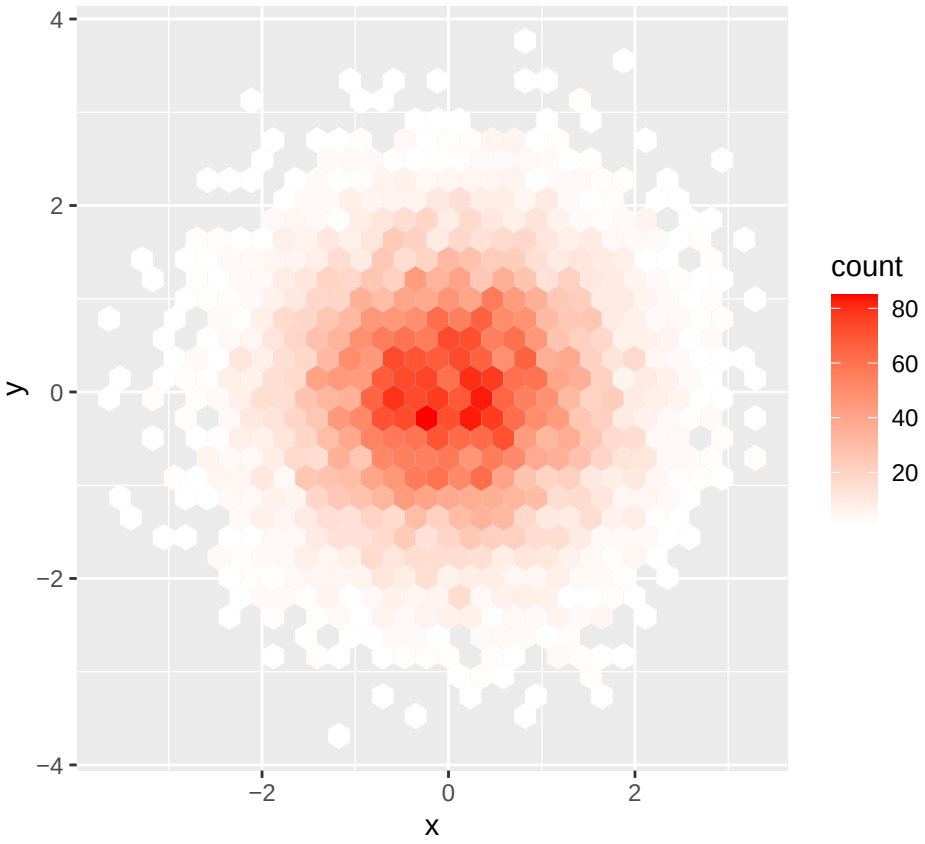
1. Why doesn't the following code override the default scale?

```
df <- tibble(
  x = rnorm(10000),
  y = rnorm(10000)
)

ggplot(df, aes(x, y)) +
  geom_hex() +
  scale_color_gradient(low = "white", high = "red") +
  coord_fixed()
```



Ans. Use`scale_fill_gradient()` instead of `scale_color_gradient()`

```
df <- tibble(
  x = rnorm(10000),
  y = rnorm(10000)
)

ggplot(df, aes(x, y)) +
  geom_hex() +
  scale_fill_gradient(low = "white", high = "red") +
  coord_fixed()
```



In `ggplot2`, `color` is mapped to line and point, and `fill` is mapped to something with area.

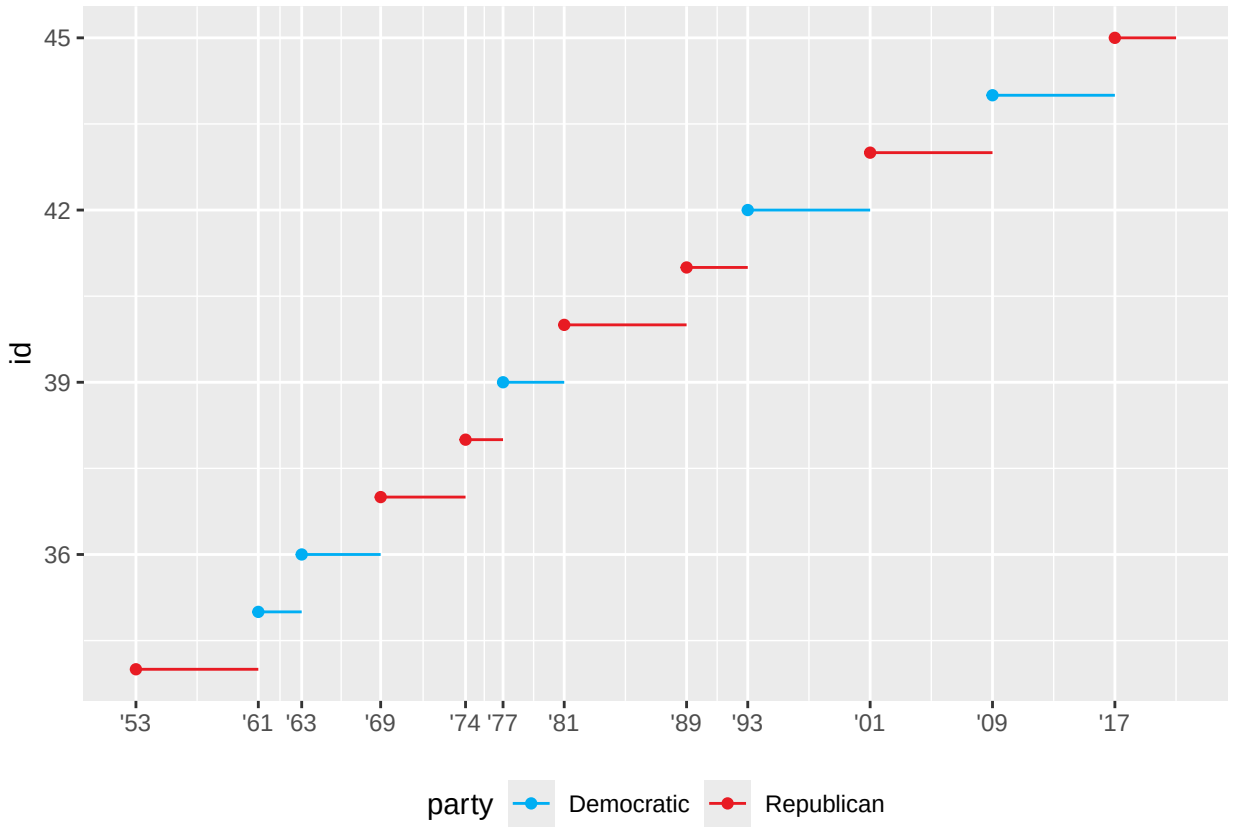
2. What is the first argument to every scale? how does it compare to `labs()`?

Ans. For the `scale_*` family of functions, the first argument is `name`. Meanwhile, `title` is the first argument to `labs()`. `title` refers to the title of the plot, while `name` refers to the axis name.

3. Change the display of the presidential terms by:

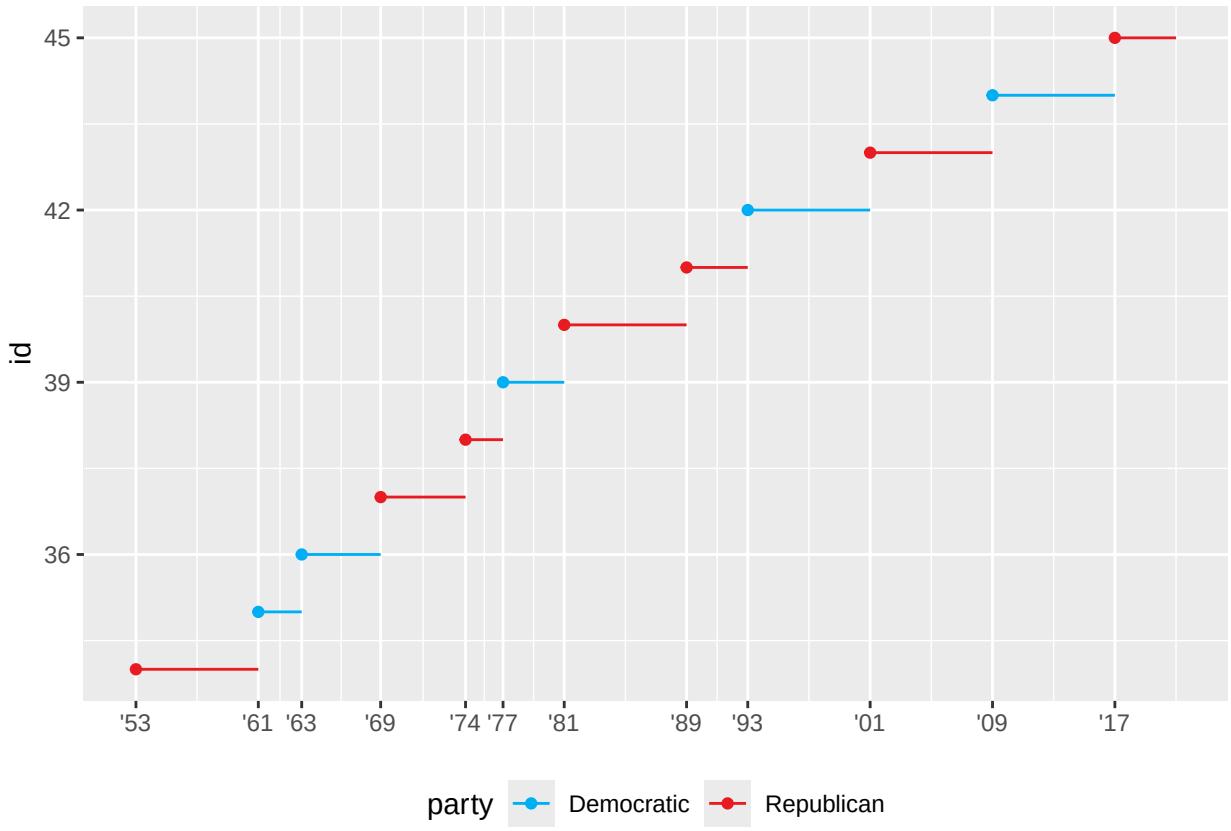
a. Combining the two variants that customize colors and x axis breaks.

```
presidential |>
  mutate(id = 33 + row_number()) |>
  ggplot(aes(x = start, y = id, color = party)) +
  geom_point() +
  geom_segment(aes(xend = end, yend = id)) +
  scale_color_manual(
    values = c(Republican = "#E81B23",
               Democratic = "#00AEF3")
  ) +
  scale_x_date(
    name = NULL,
    breaks = presidential$start,
    date_labels = "'%y"
  ) +
  theme(legend.position = "bottom")
```



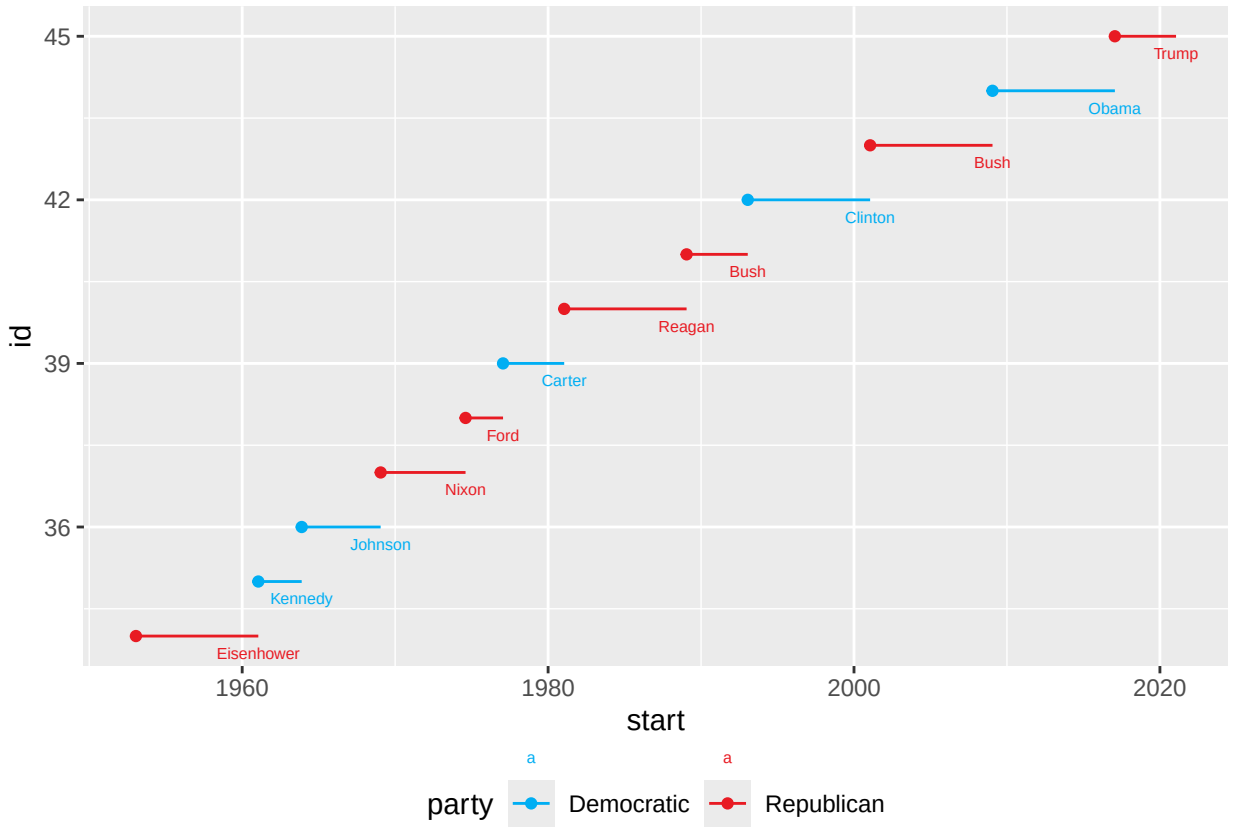
b. Improving the display of the y axis.

```
presidential |>
  mutate(id = 33 + row_number()) |>
  ggplot(aes(x = start, y = id, color = party)) +
  geom_point() +
  geom_segment(aes(xend = end, yend = id)) +
  scale_color_manual(
    values = c(Republican = "#E81B23",
               Democratic = "#00AEF3")
  ) +
  scale_x_date(
    name = NULL,
    breaks = presidential$start,
    date_labels = "'%y"
  ) +
  scale_y_continuous(breaks = seq(30, 50, by = 3)) +
  theme(legend.position = "bottom")
```



c. Labelling each term with the name of the president.

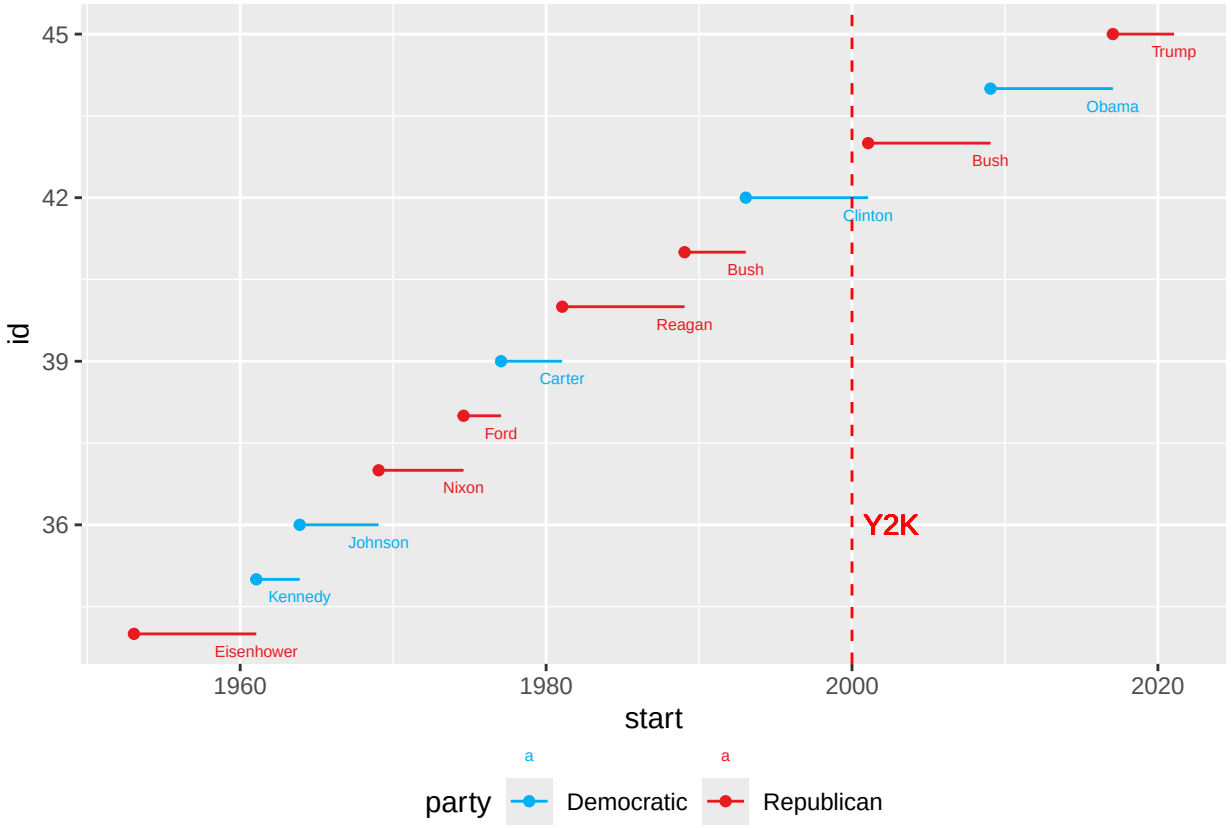
```
prez <- presidential |>
  mutate(id = 33 + row_number())

prez |>
  ggplot(aes(x = start, y = id, color = party)) +
  geom_point() +
  geom_segment(aes(xend = end, yend = id)) +
  scale_color_manual(
    values = c(Republican = "#E81B23",
               Democratic = "#00AEF3")
  ) +
  geom_text(aes(
    x = end, y = id, label = name),
    size = 2, hjust = "center", vjust = 2
    ) +
  theme(legend.position = "bottom")
```



d. Adding informative plot labels.

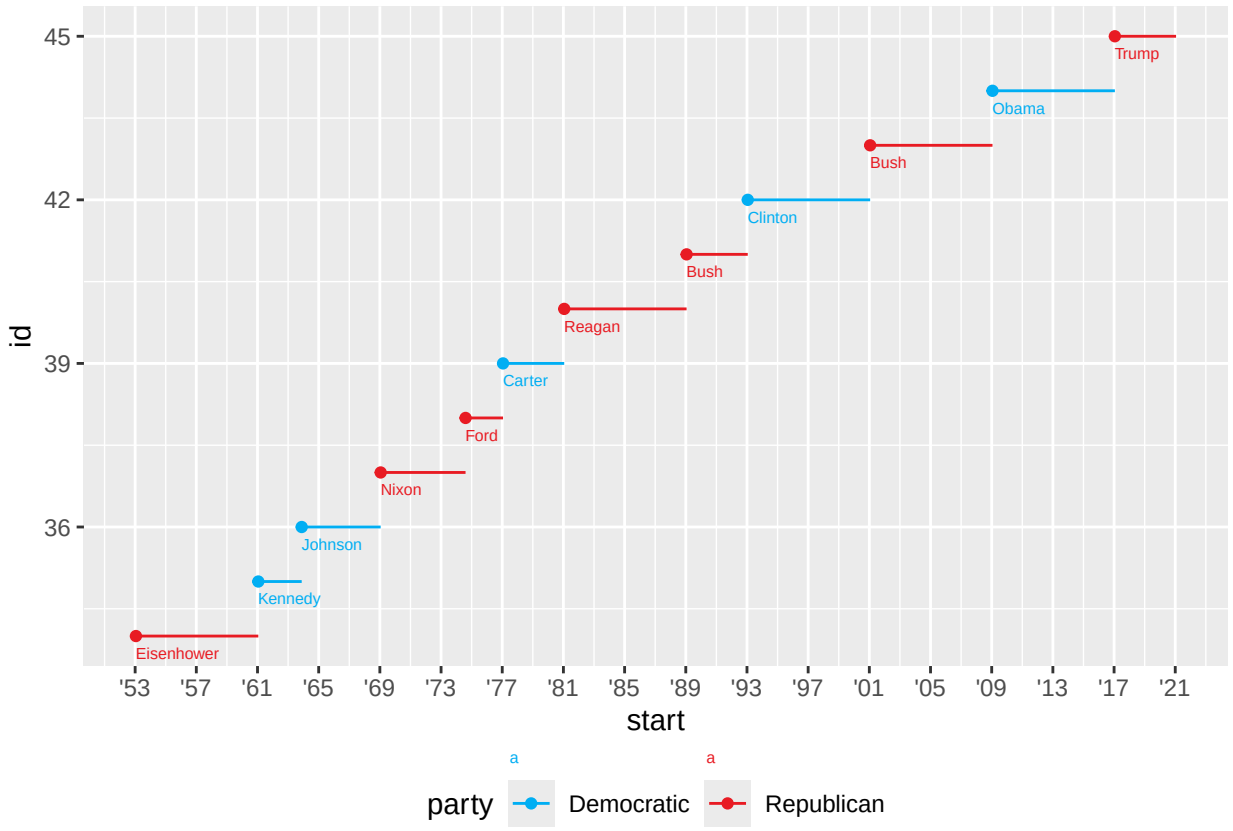
```
prez <- presidential |>
  mutate(id = 33 + row_number())

prez |>
  ggplot(aes(x = start, y = id, color = party)) +
  geom_point() +
  geom_segment(aes(xend = end, yend = id)) +
  scale_color_manual(
    values = c(Republican = "#E81B23",
               Democratic = "#00AEF3")
  ) +
  geom_text(
    aes(x = end, y = id, label = name),
    size = 2, hjust = "center", vjust = 2
    ) +
  geom_vline(
    aes(xintercept = as.Date("2000-01-01")),
    linewidth = 0.5,
    color = "red",
    linetype = 2
  ) +
  geom_text(
    aes(x = as.Date("2000-10-01"), y = 36, label = "Y2K"),
    color = "red", hjust = "left"
  ) +
  theme(legend.position = "bottom")
```

```
## Warning in geom_text(aes(x = as.Date("2000-10-01"), y = 36, label = "Y2K"), : All aesthetics have le
## i Please consider using `annotate()` or provide this layer with data containing
##   a single row.
```



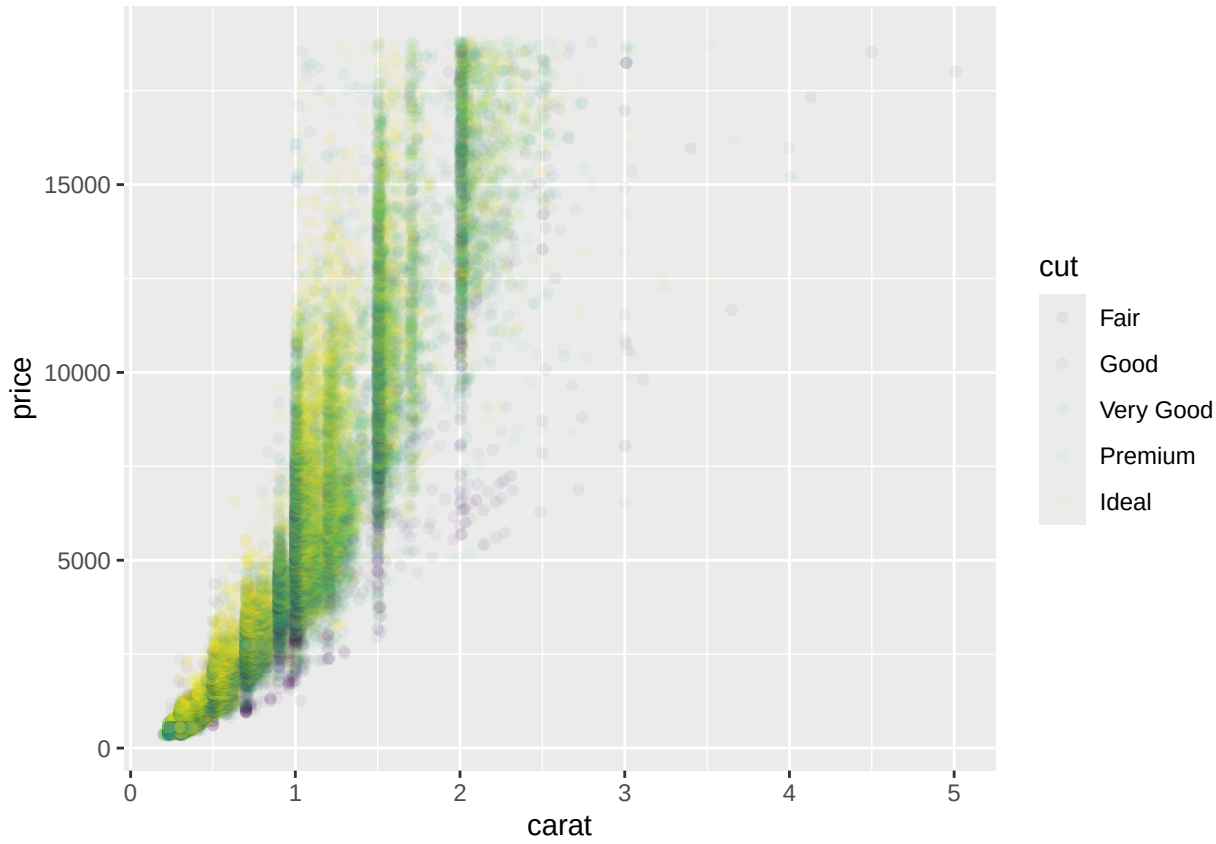
e. Placing breaks every 4 years (this is trickier than it seems!).

```
prez |>
  ggplot(aes(x = start, y = id, color = party)) +
  geom_point() +
  geom_segment(aes(xend = end, yend = id)) +
  scale_color_manual(
    values = c(Republican = "#E81B23",
               Democratic = "#00AEF3")
  ) +
  geom_text(aes(
    x = start, y = id, label = name),
    size = 2, hjust = "left", vjust = 2
    ) +
  scale_x_date(date_breaks = "4 years", date_labels = "'%y") +
  theme(legend.position = "bottom")
```



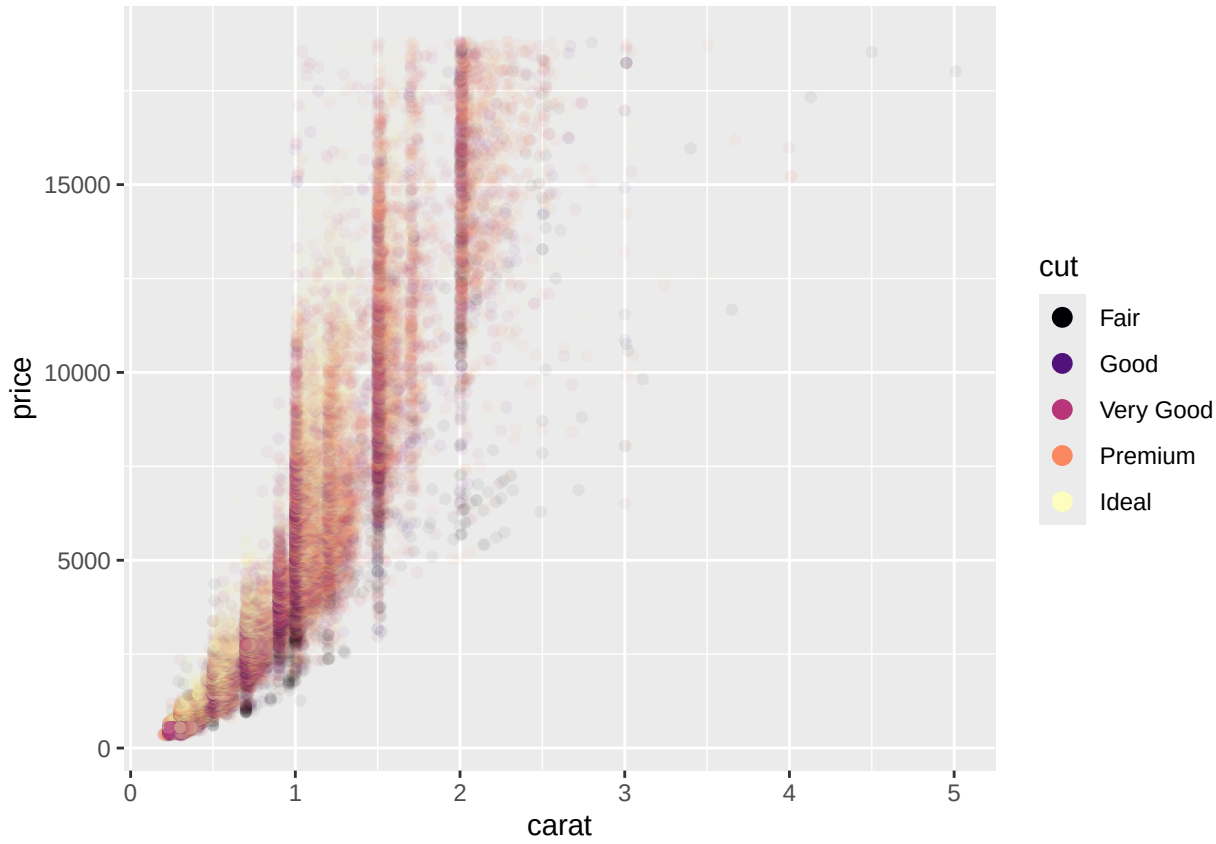
5. First, create the following plot. Then, modify the code using override.aes to make the legend easier to see.

```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point(aes(color = cut), alpha = 1/20)
```



Ans.

```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point(aes(color = cut), alpha = 1/20) +
  scale_color_viridis_d(
    option = "magma",
    guide = guide_legend(override.aes = list(
      size = 3,
      alpha = 1
    ))
  )
```



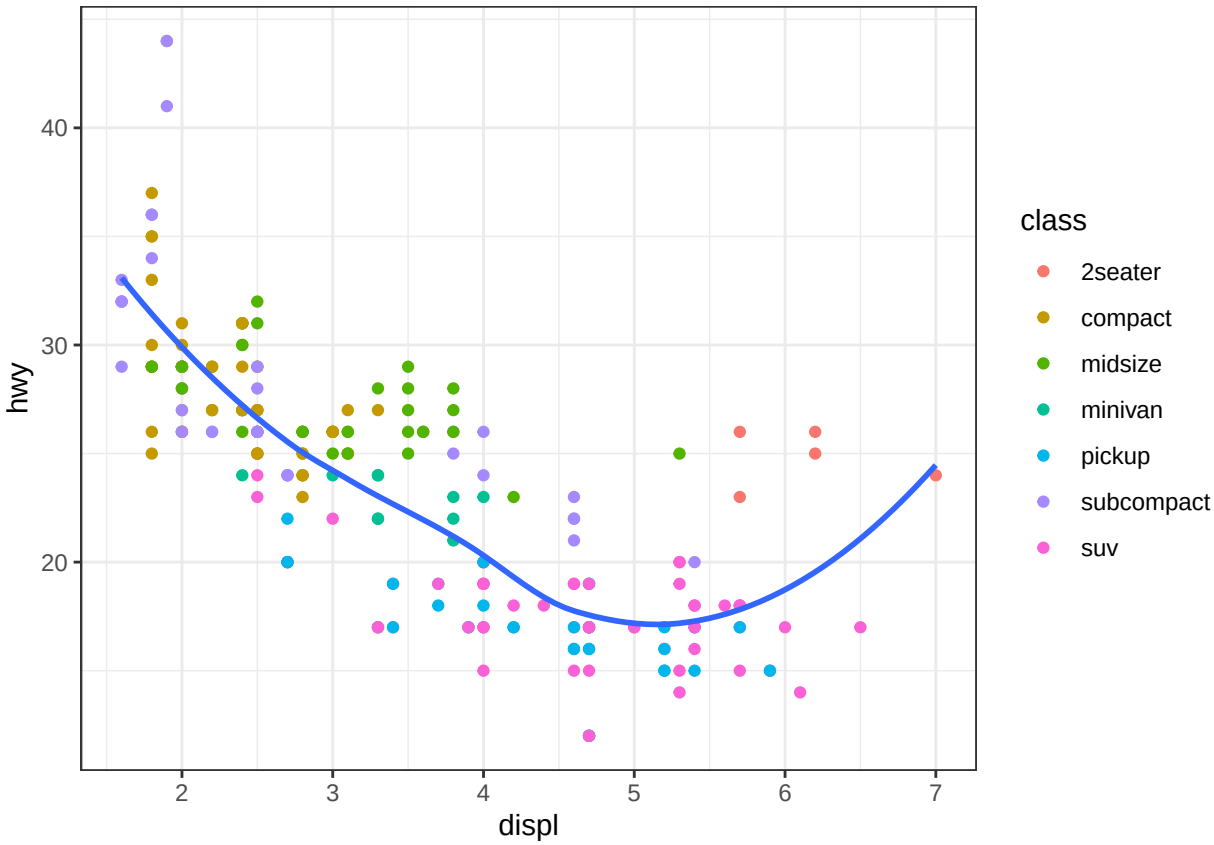
Themes

Customizing the non-data elements by applying a theme:

White background theme:

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

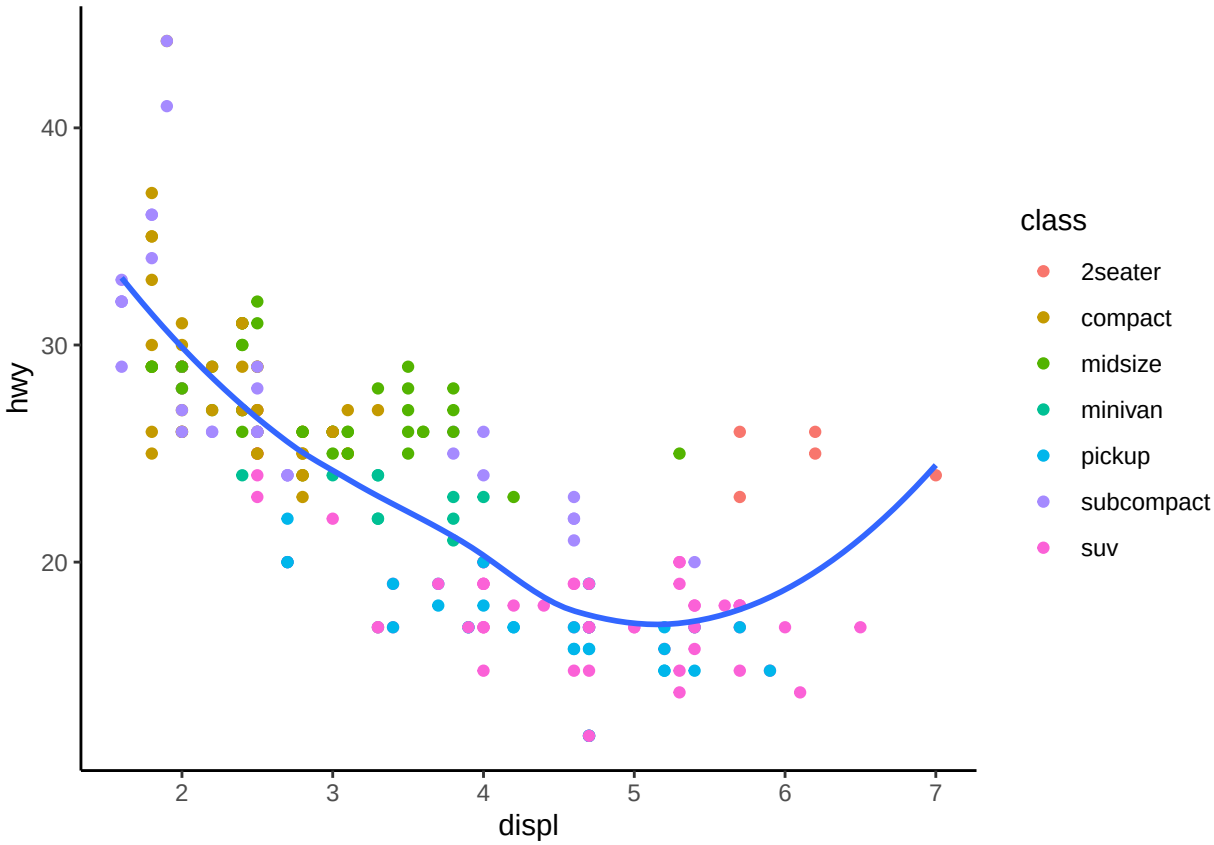
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Classic theme:

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

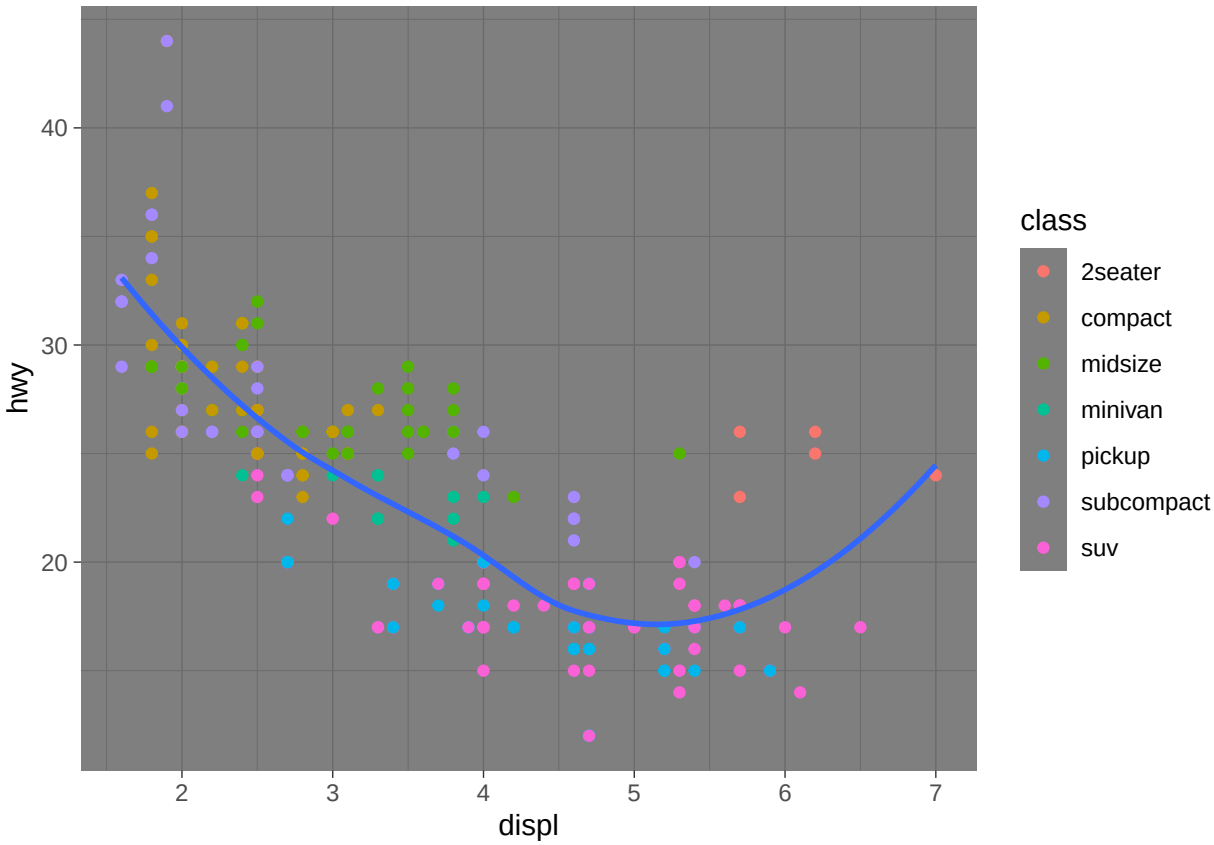
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Dark theme:

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

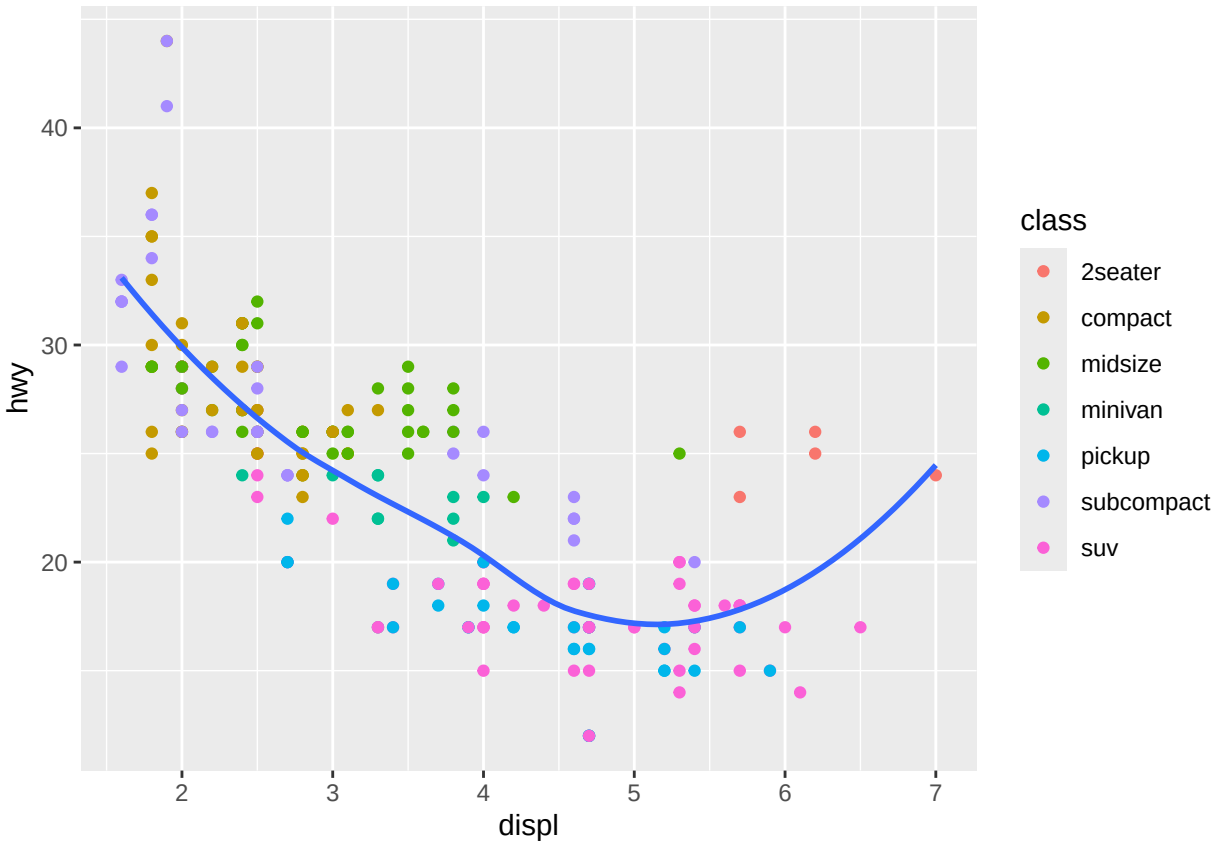
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Gray theme:

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

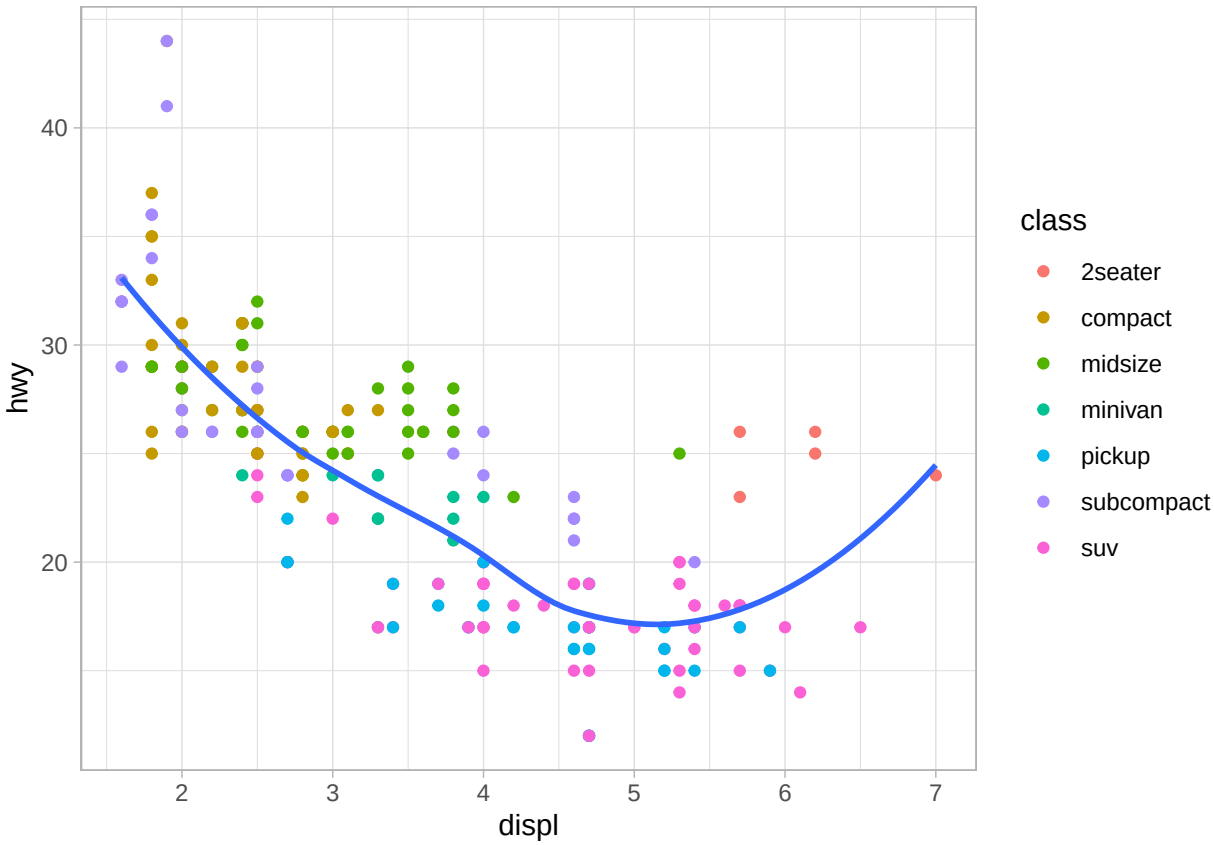
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Light theme:

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

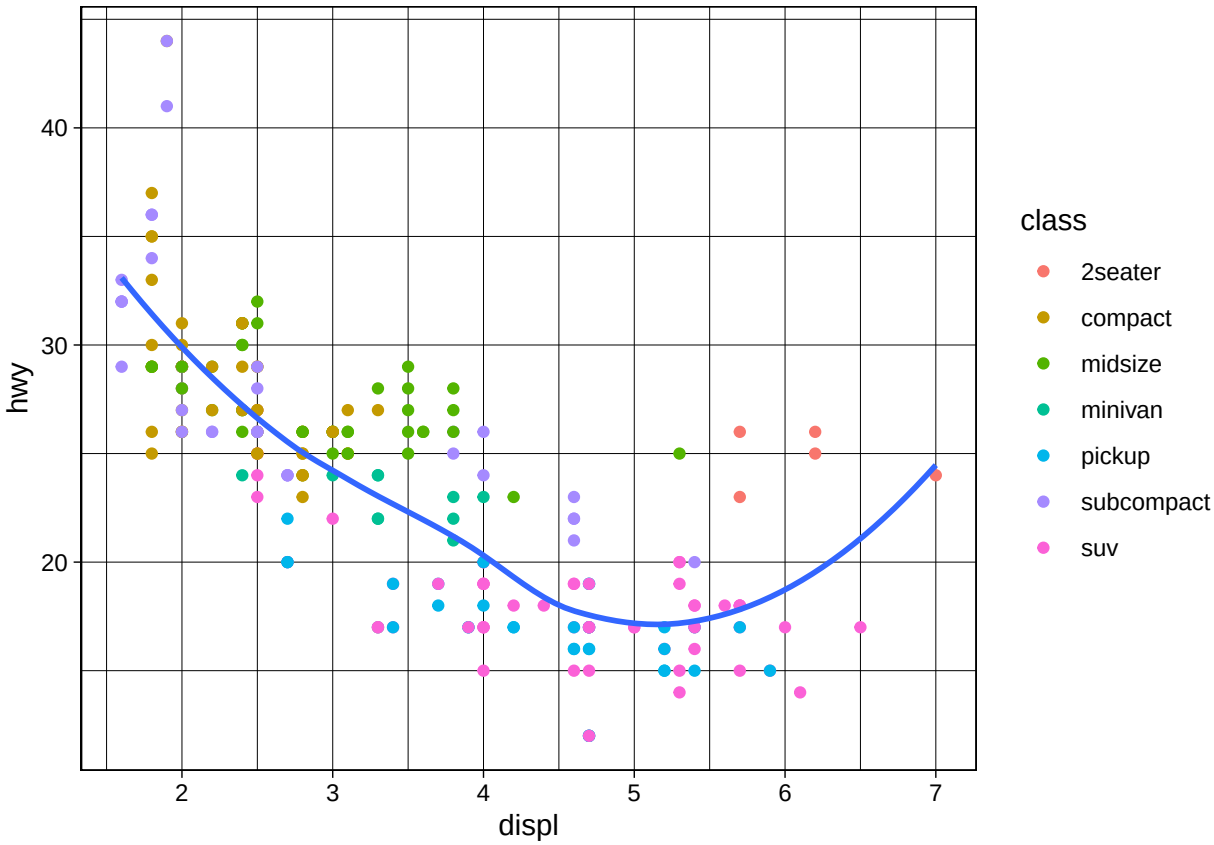
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Linedraw theme:

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

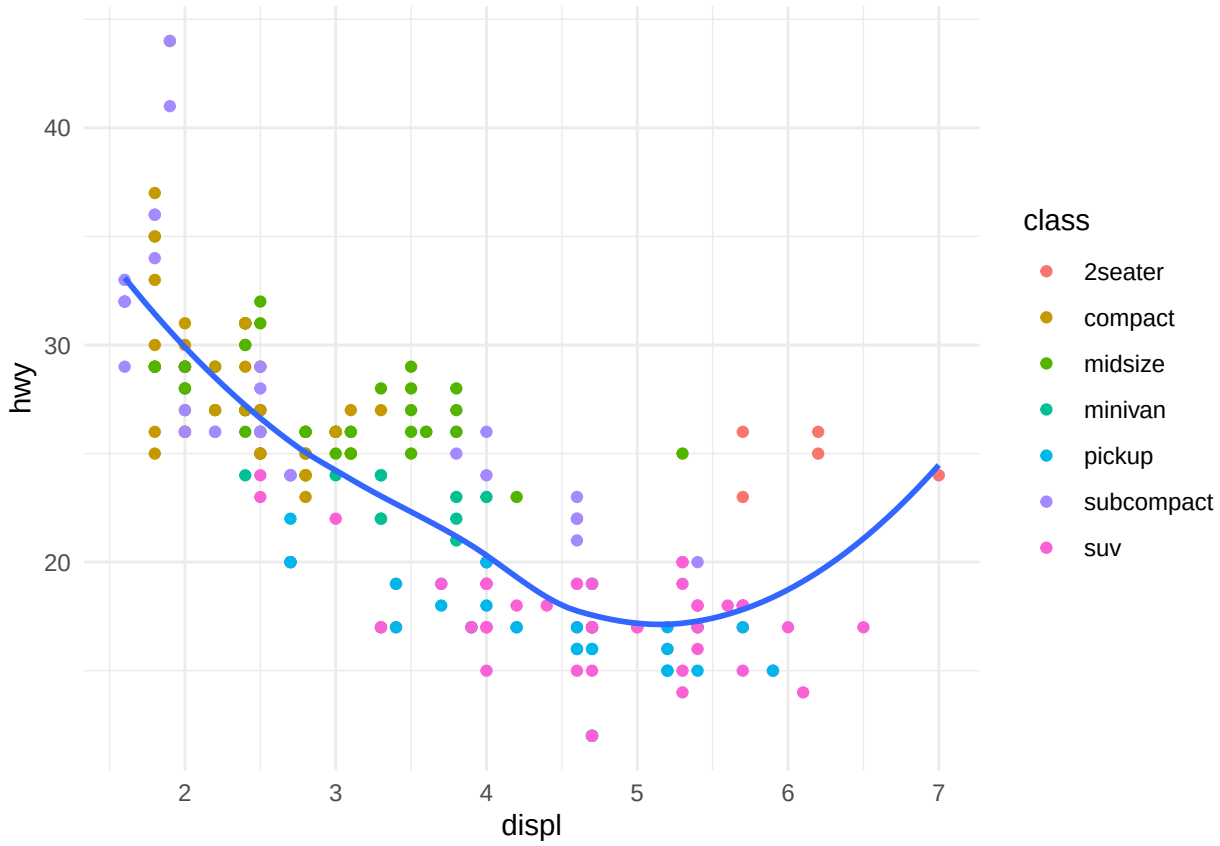
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Minimal theme:

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

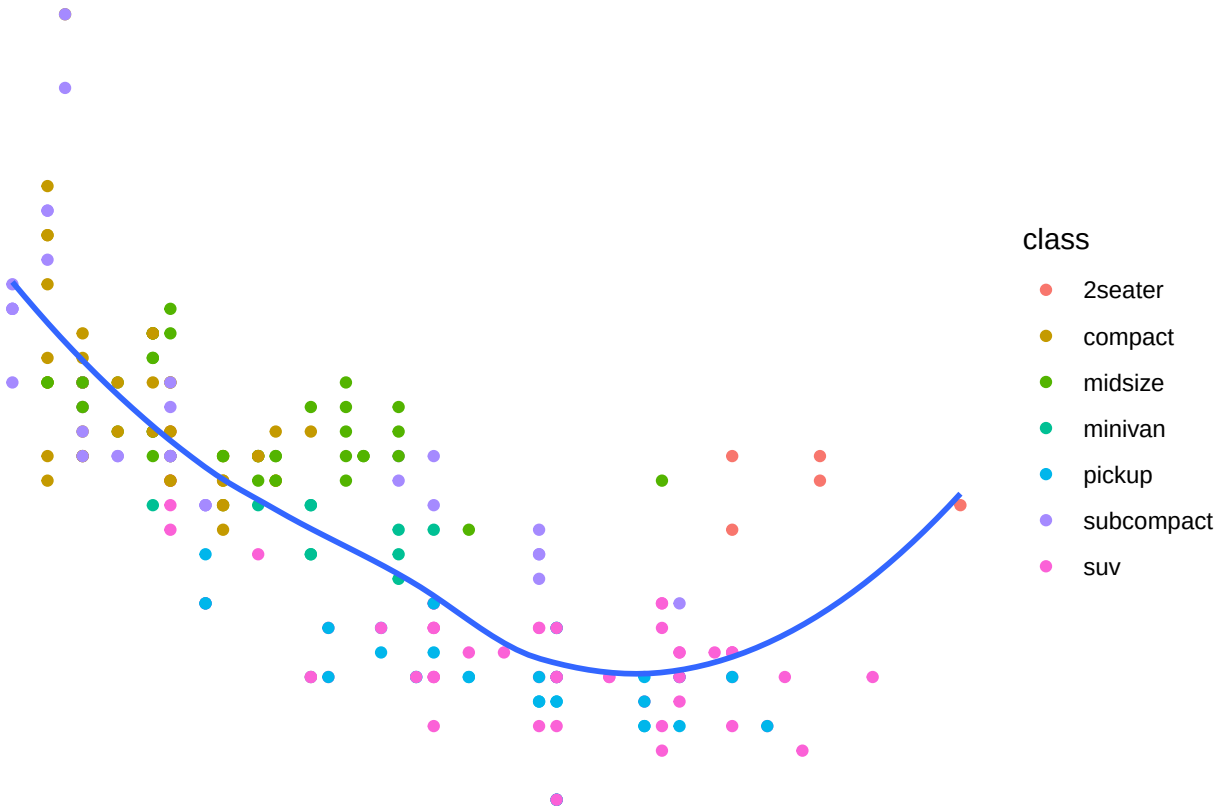
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Void theme:

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

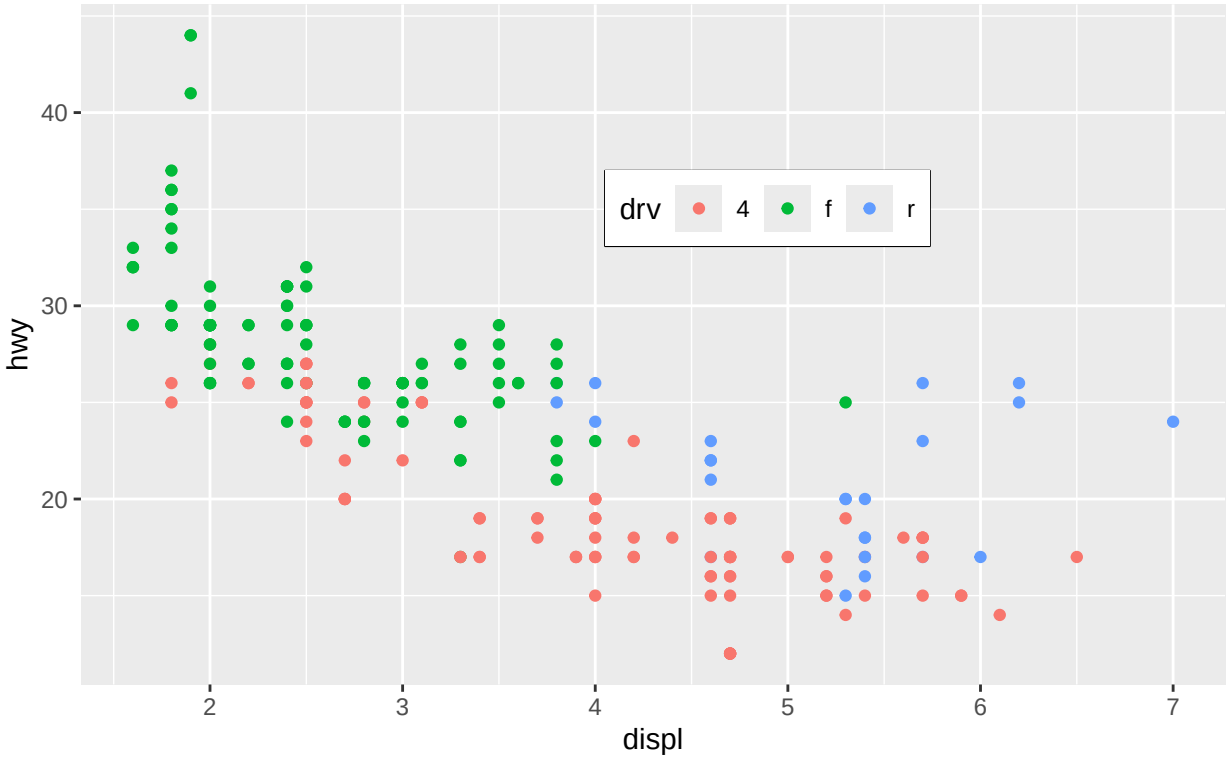
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Customized theme:

```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  labs(
    title = "Larger engine sizes tend to have lower fuel economy",
    caption = "Source: https://fueleconomy.gov."
  ) +
  theme(
    legend.position = c(0.6, 0.7),
    legend.direction = "horizontal",
    legend.box.background = element_rect(color = "black"),
    plot.title = element_text(face = "bold"),
    plot.title.position = "plot",
    plot.caption.position = "plot",
    plot.caption = element_text(hjust = 0)
  )
```

```
## Warning: A numeric `legend.position` argument in `theme()` was deprecated in ggplot2
## 3.5.0.
## i Please use the `legend.position.inside` argument of `theme()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

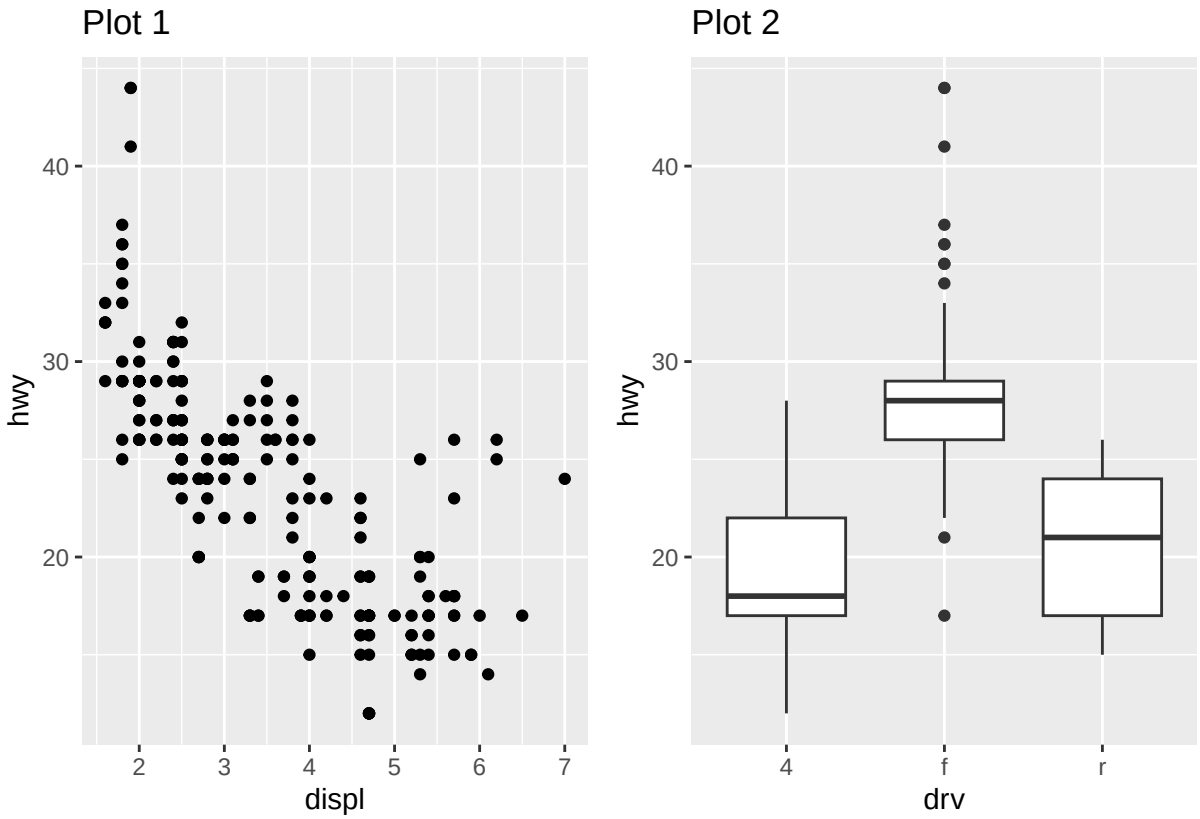


Source: https://fueleconomy.gov.

Layout

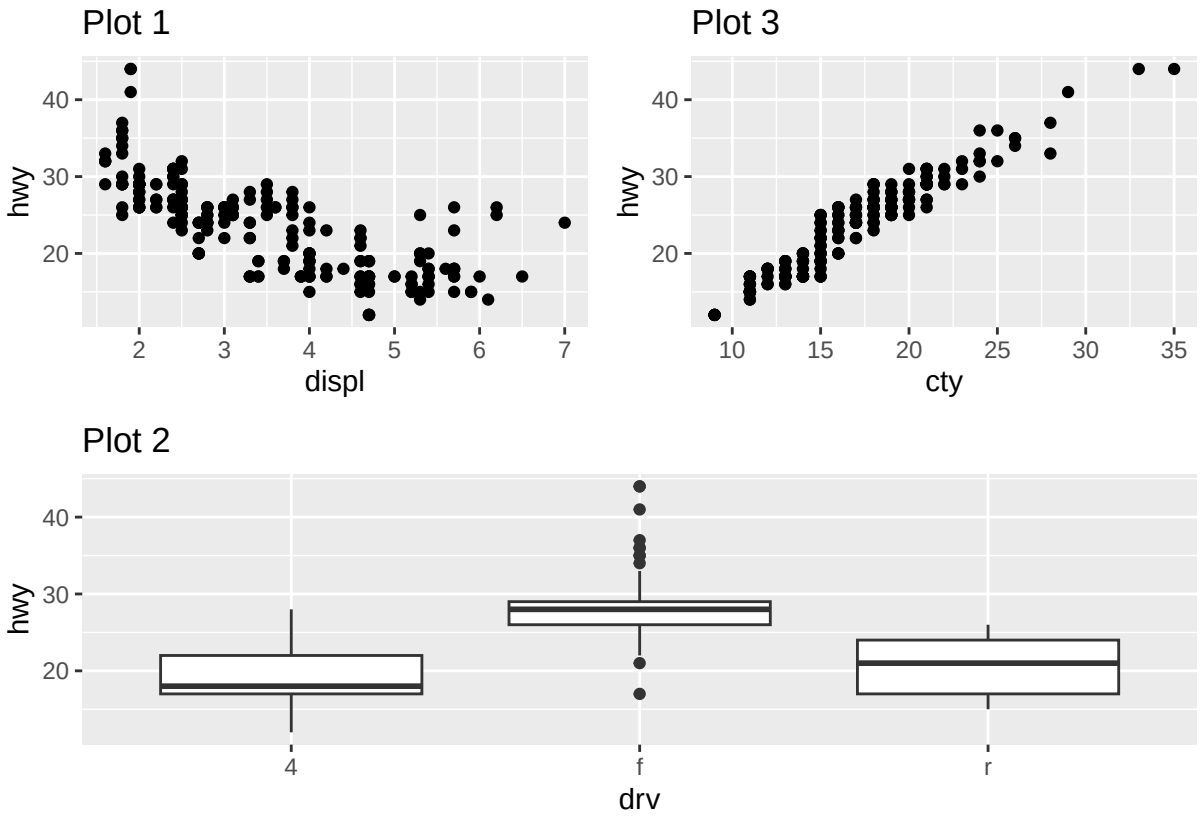
Side-by-side plot layout using `+` from the `patchwork` package:

```
p1 <- ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  labs(title = "Plot 1")

p2 <- ggplot(mpg, aes(x = drv, y = hwy)) +
  geom_boxplot() +
  labs(title = "Plot 2")

p1 + p2
```



Complex plot layout using | and / from the `patchwork` package:

```
p3 <- ggplot(mpg, aes(x = cty, y = hwy)) +
  geom_point() +
  labs(title = "Plot 3")

(p1 | p3) / p2
```

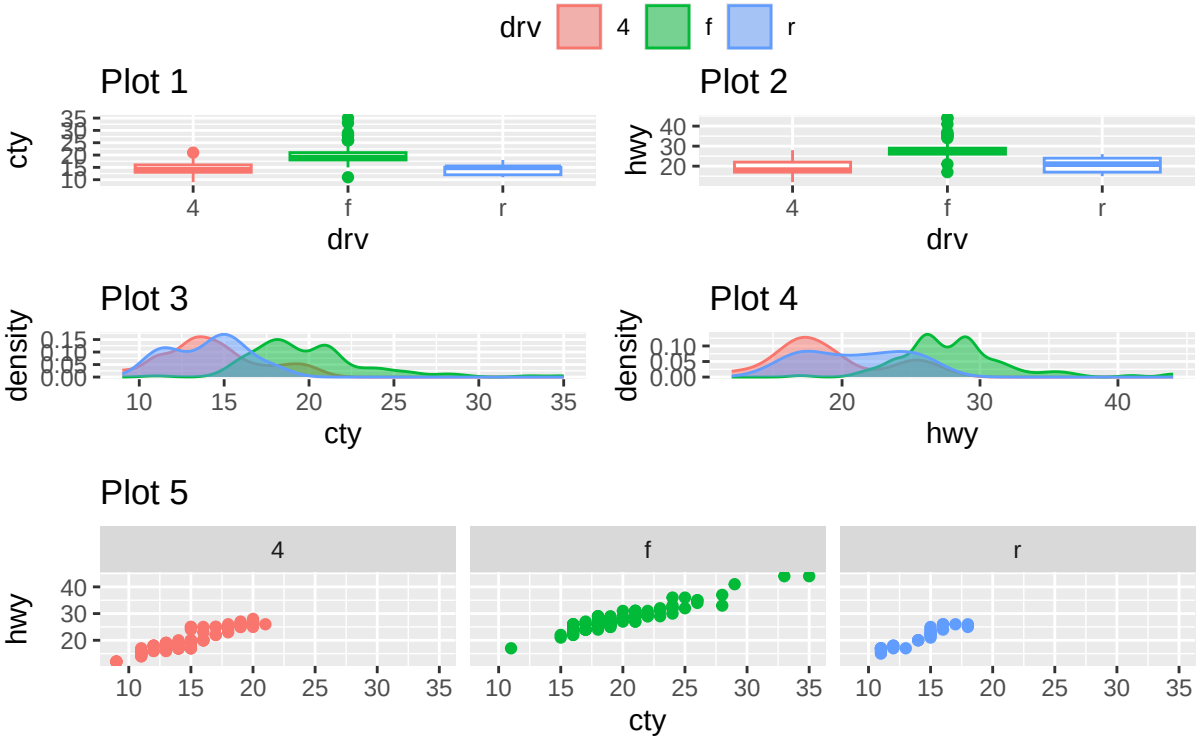


Other examples:

```
p1 <- ggplot(mpg, aes(x = drv, y = cty, color = drv)) +
  geom_boxplot(show.legend = FALSE) +
  labs(title = "Plot 1")

p2 <- ggplot(mpg, aes(x = drv, y = hwy, color = drv)) +
  geom_boxplot(show.legend = FALSE) +
  labs(title = "Plot 2")

p3 <- ggplot(mpg, aes(x = cty, color = drv, fill = drv)) +
  geom_density(alpha = 0.5) +
  labs(title = "Plot 3")

p4 <- ggplot(mpg, aes(x = hwy, color = drv, fill = drv)) +
  geom_density(alpha = 0.5) +
  labs(title = "Plot 4")

p5 <- ggplot(mpg, aes(x = cty, y = hwy, color = drv)) +
  geom_point(show.legend = FALSE) +
  facet_wrap(~drv) +
  labs(title = "Plot 5")

(guide_area() / (p1 + p2) / (p3 + p4) / p5) +
  plot_annotation(
    title = "City and highway mileage for cars with different drive trains",
    caption = "Source: https://fueleconomy.gov."
  ) +
```

```
plot_layout(
  guides = "collect",
  heights = c(1, 3, 2, 4)
  ) &
theme(legend.position = "top")
```

City and highway mileage for cars with different drive trains



Source: https://fueleconomy.gov.