Base Visualization in data science

Howard Nguyen

2023-05-06

Load libraries

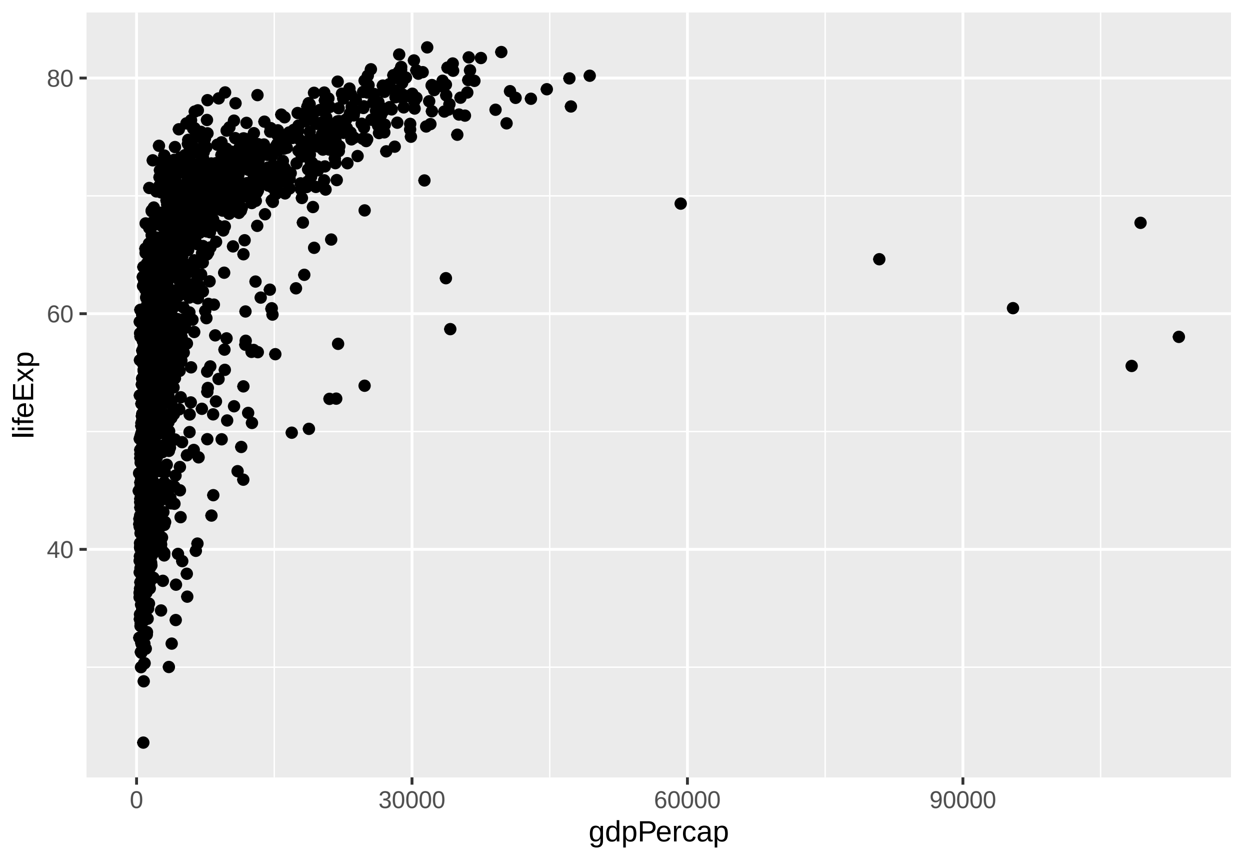
```
library(ggplot2)
library(gapminder)
```

```
gapminder
```

```
## # A tibble: 1,704 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
##  6 Afghanistan Asia       1977    38.4 14880372      786.
##  7 Afghanistan Asia       1982    39.9 12881816      978.
##  8 Afghanistan Asia       1987    40.8 13867957      852.
##  9 Afghanistan Asia       1992    41.7 16317921      649.
## 10 Afghanistan Asia       1997    41.8 22227415      635.
## # i 1,694 more rows
```

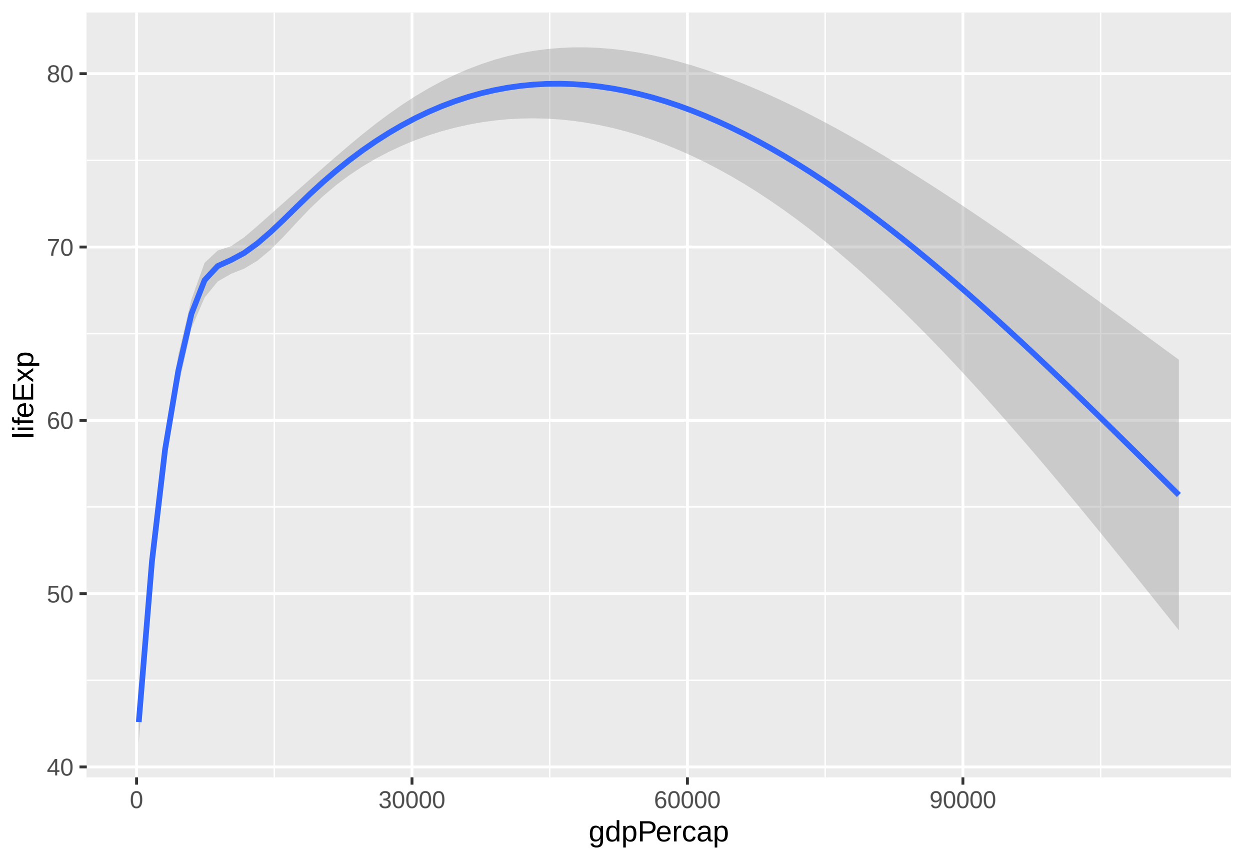
```
p <- ggplot(data=gapminder)
p <- ggplot(data = gapminder, mapping = aes(gdpPercap,lifeExp))
```

```
p + geom_point()
```



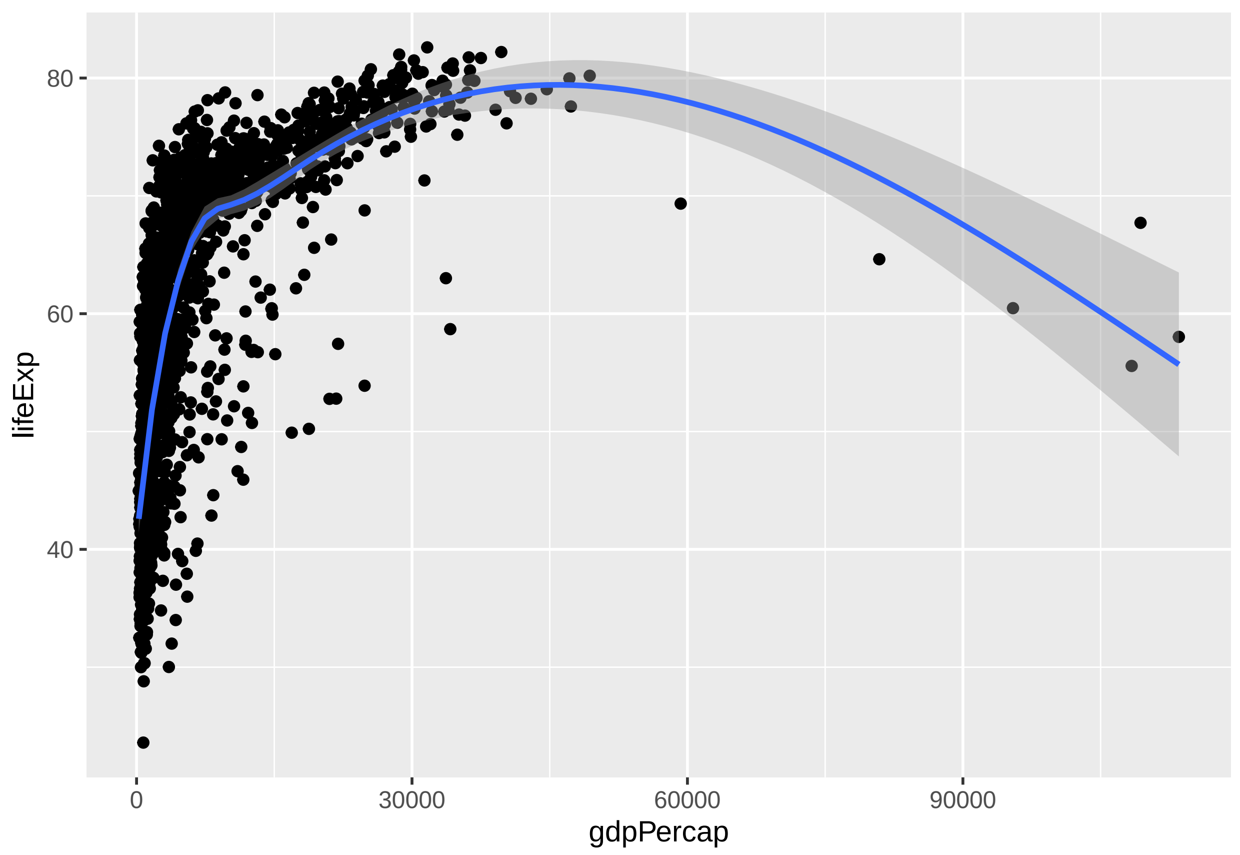
```
p + geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```



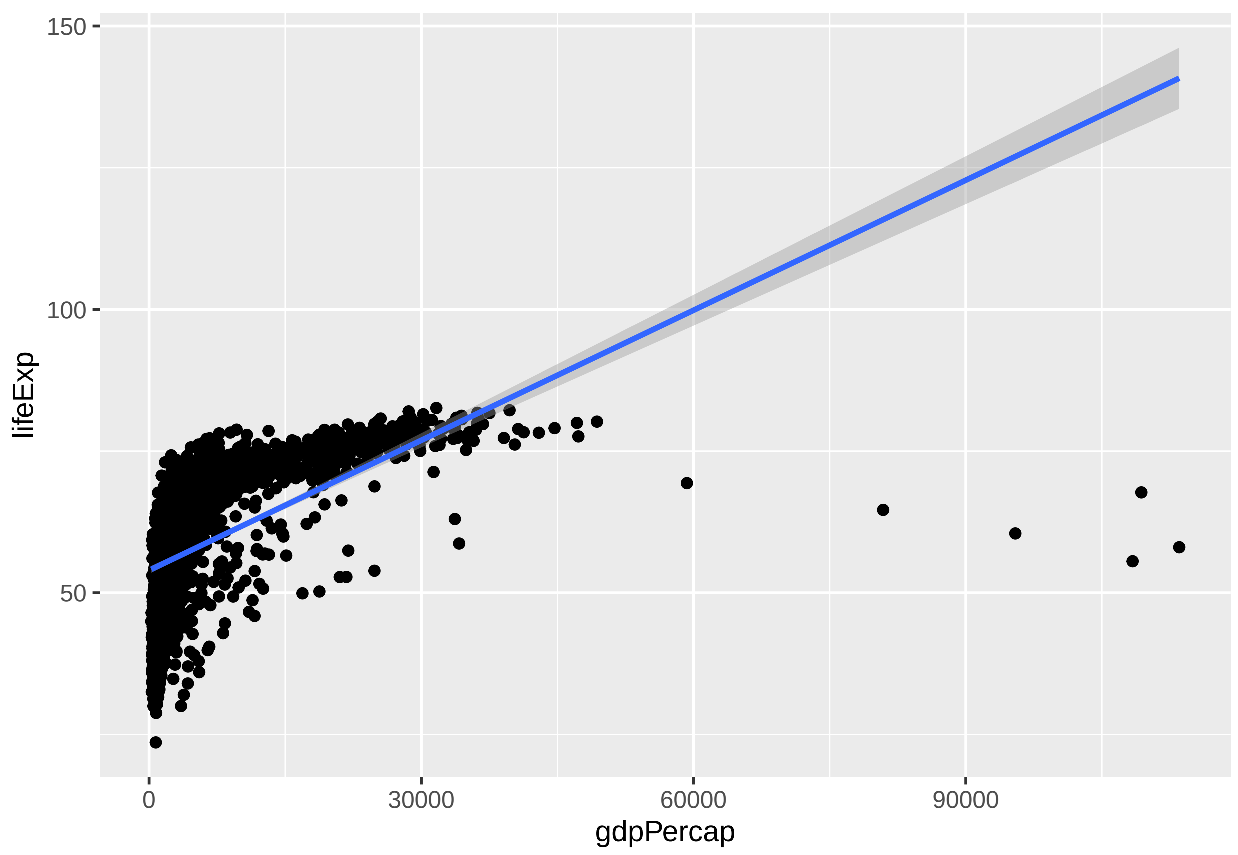
```
p + geom_point() + geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```



```r
p + geom_point() + geom_smooth(method = "lm")
```

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

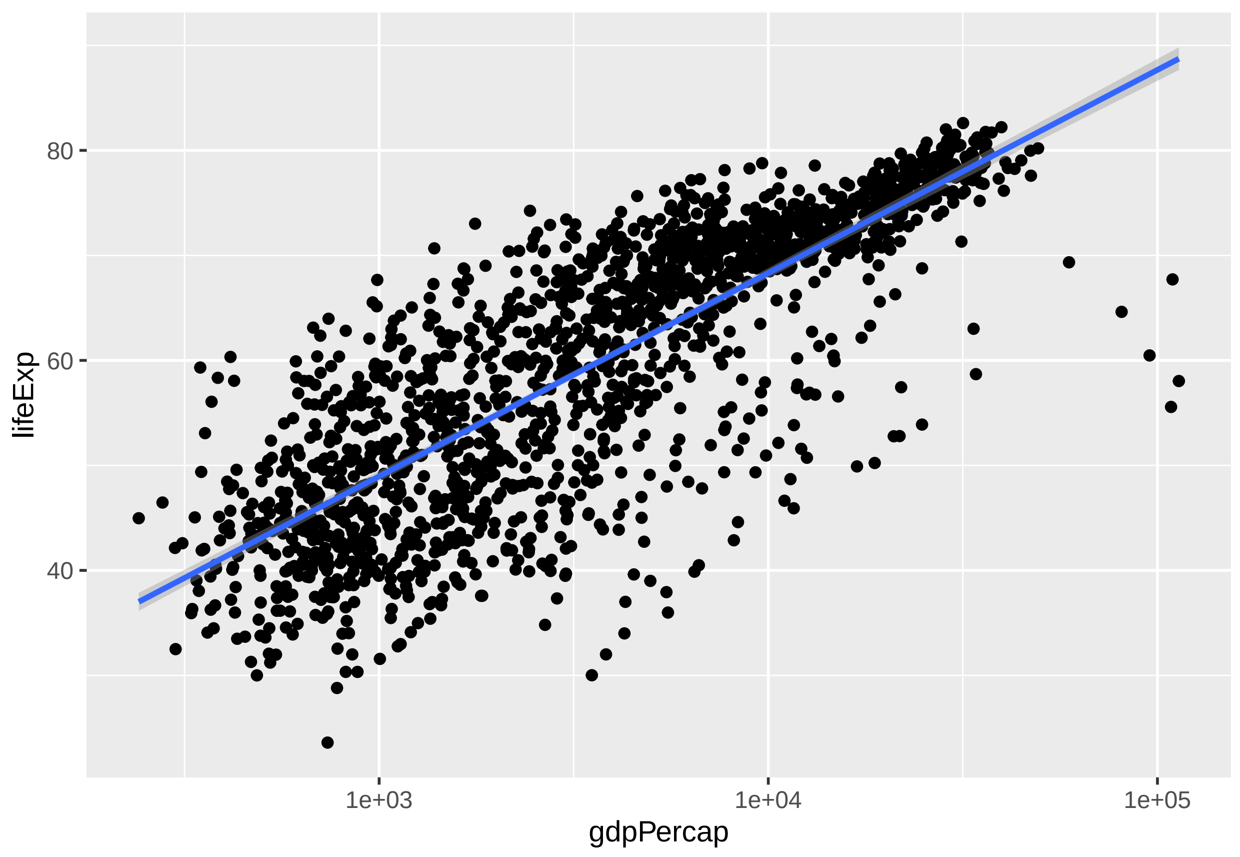


Scale Log 10

the data is quite bunched up against the left side. Gross domestic product per capita is not normally distributed across our country years. The x-axis scale would probably look better if it were transformed from a linear scale to a log scale. For this, we can use a function called scale_x_log10(). As you might expect, this function scales the x-axis of a plot to a log 10 basis. To use it we just add it to the plot:

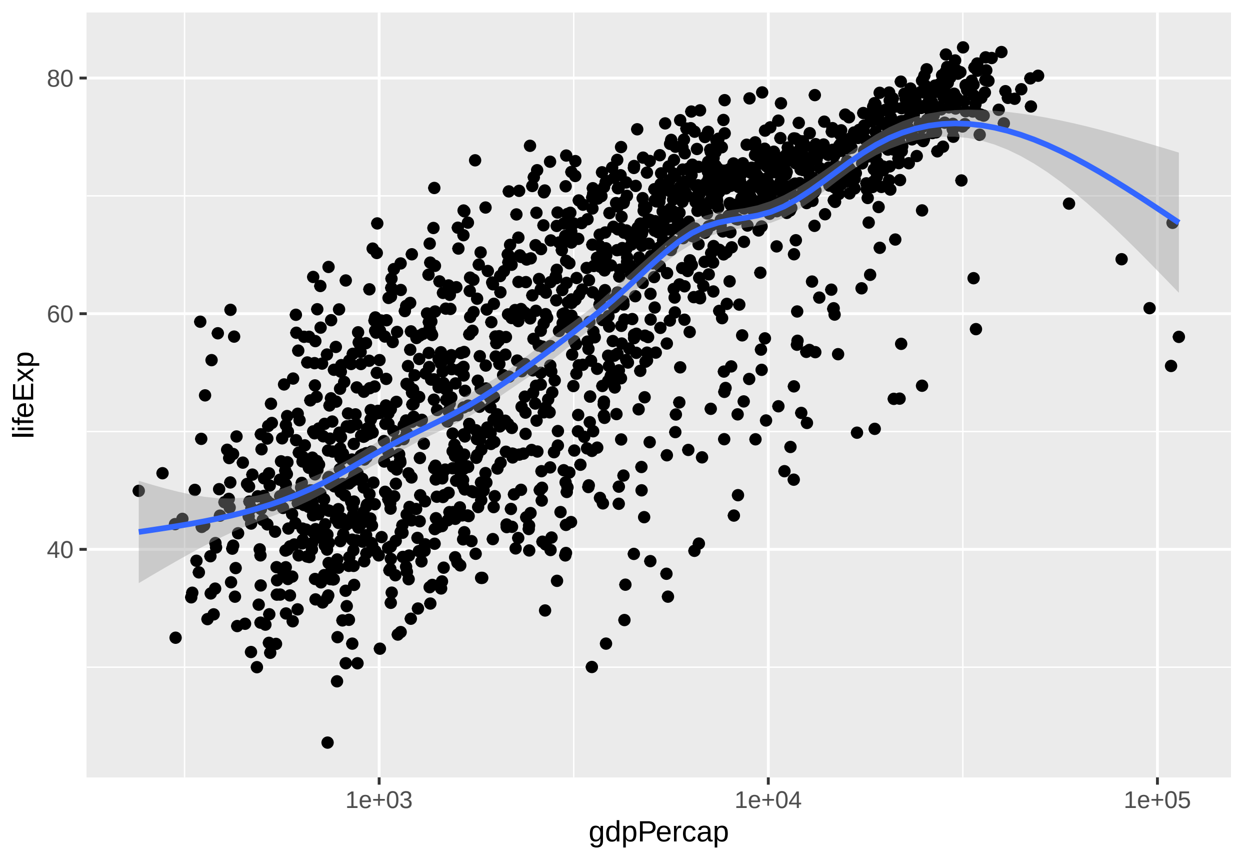
```
p + geom_point() +
  geom_smooth(method = "lm") +
  scale_x_log10()
```

```
## ‘geom_smooth()‘ using formula = ’y ~ x’
```



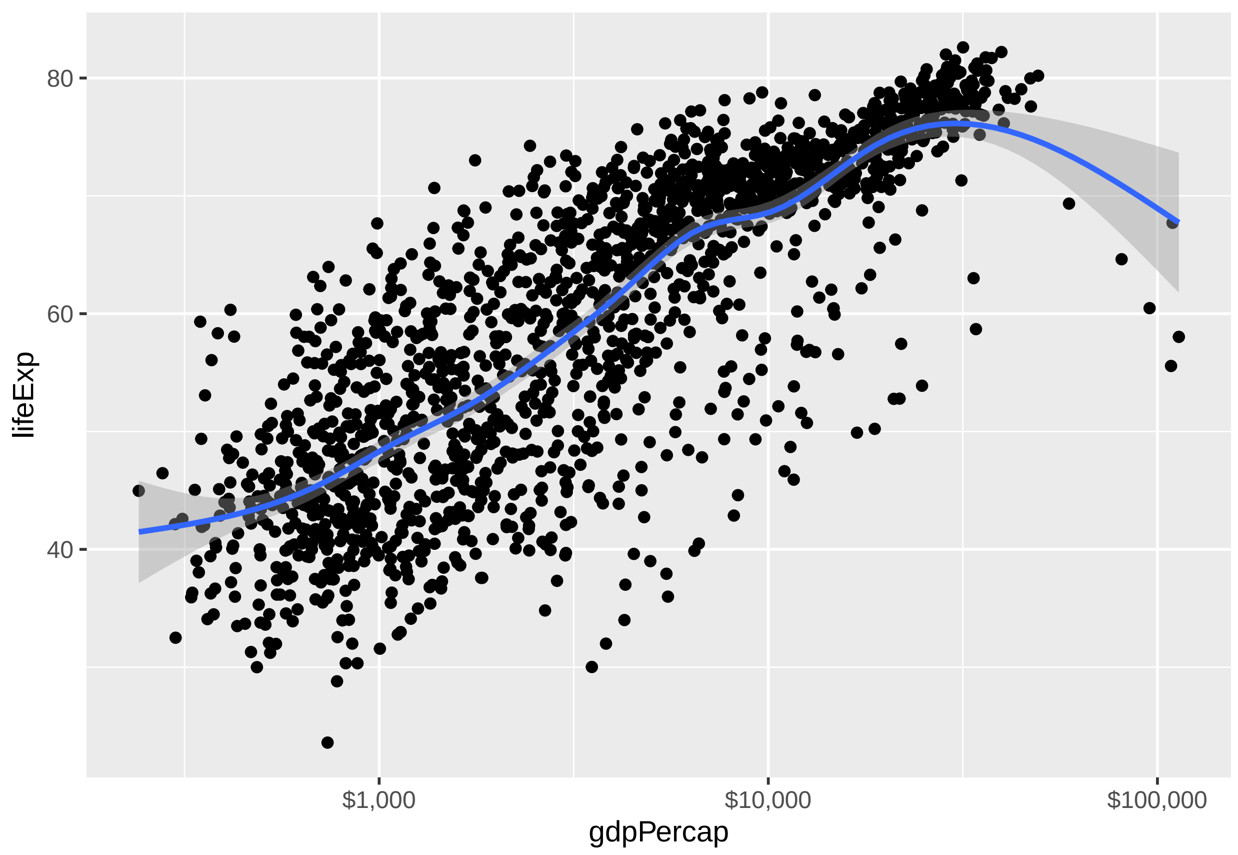
```r
p + geom_point() +
  geom_smooth(method = "gam") +
  scale_x_log10()
```

```
## `geom_smooth()` using formula = 'y ~ s(x, bs = "cs")'
```



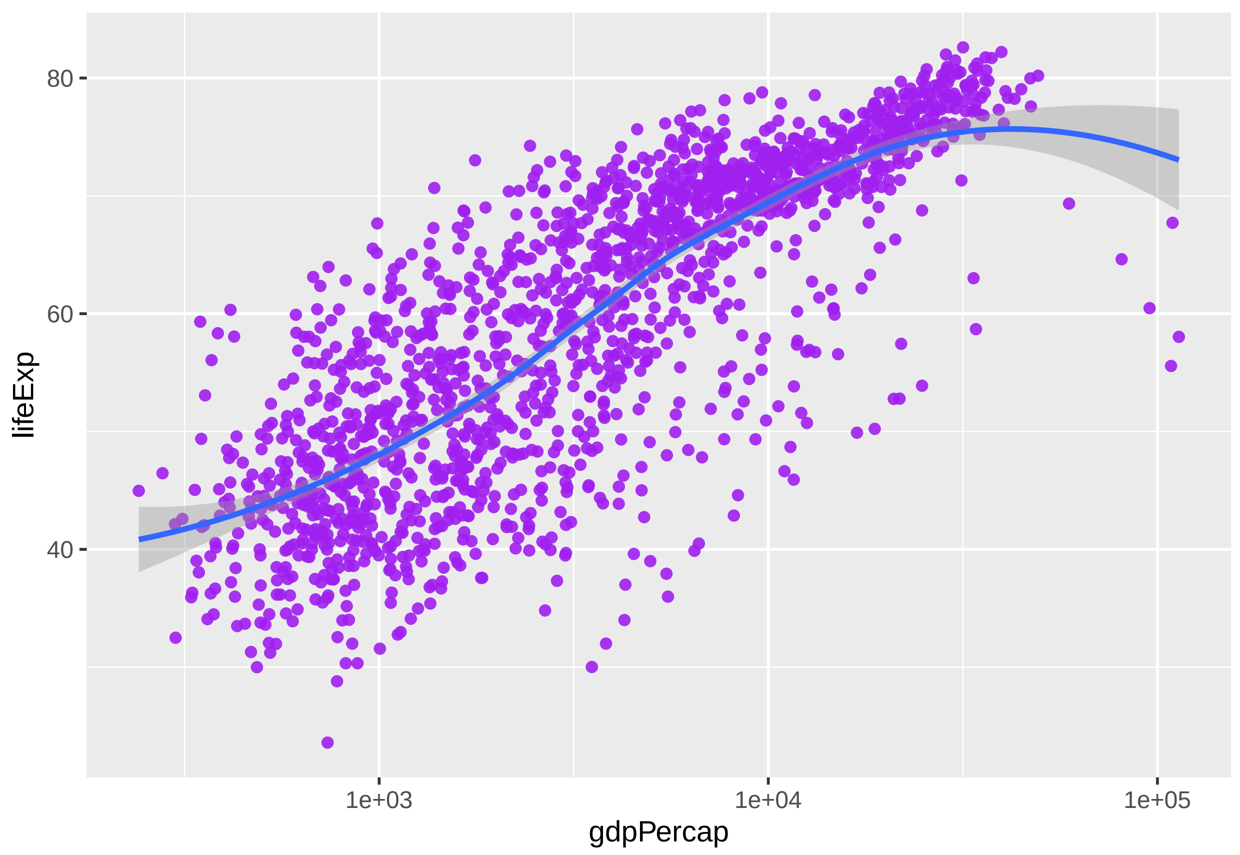
```
p + geom_point() +
  geom_smooth(method = "gam") +
  scale_x_log10(labels = scales::dollar)
```

```
## `geom_smooth()` using formula = 'y ~ s(x, bs = "cs")'
```



```
p + geom_point(color="purple", alpha=0.9) +
  geom_smooth(method = "loess") +
  scale_x_log10()
```

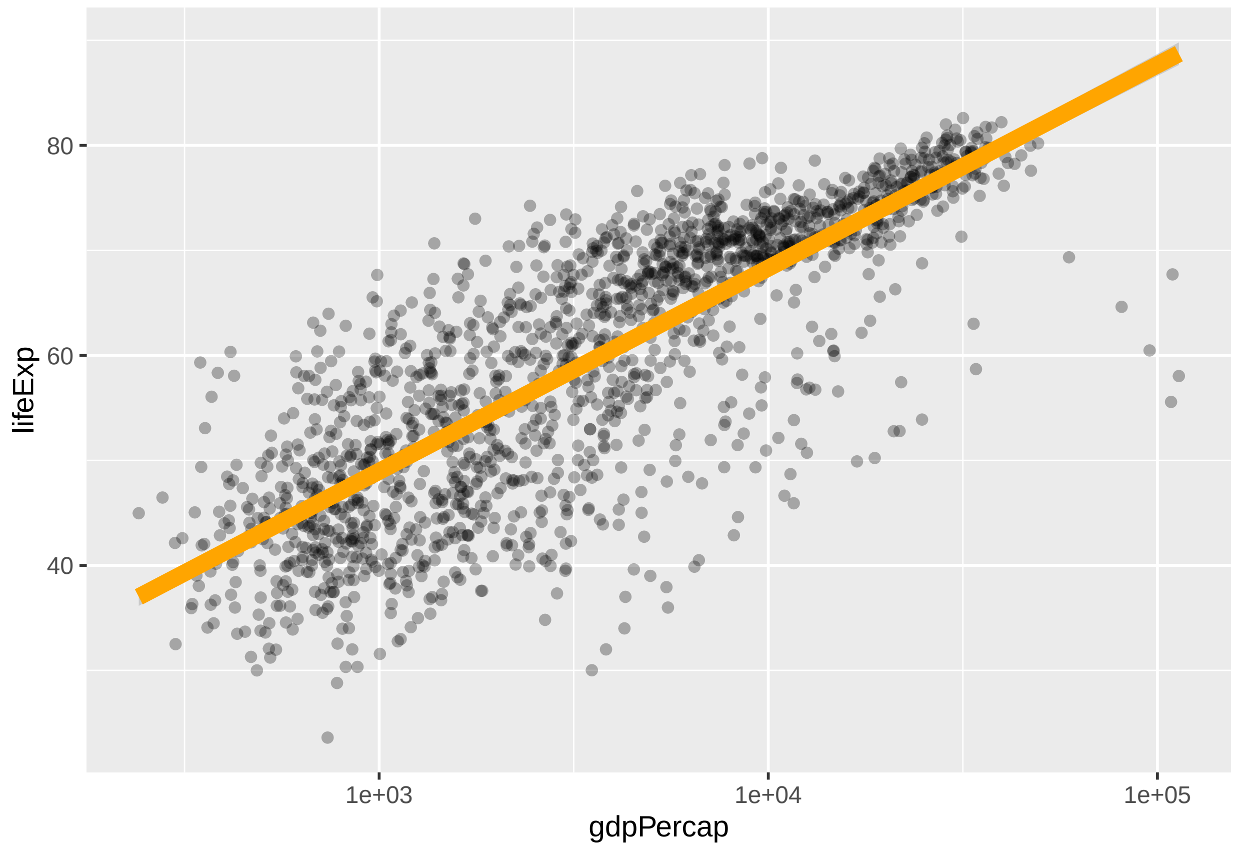
```
## ‘geom_smooth()‘ using formula = ’y ~ x’
```



```
p + geom_point(alpha=0.3) +
  geom_smooth(color="orange", se=TRUE, size=3, method="lm") +
  scale_x_log10()
```

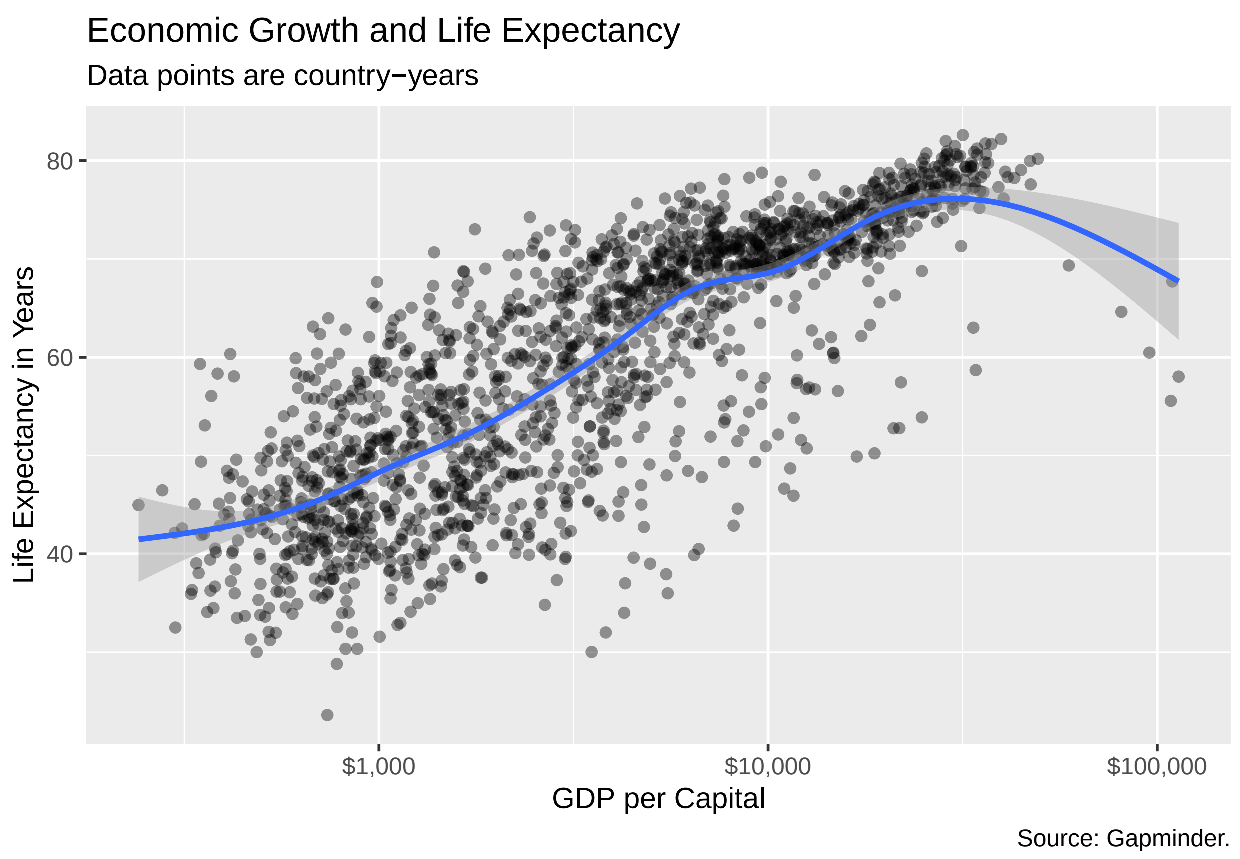
```
## Warning: Using 'size' aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use 'linewidth' instead.
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was
## generated.
```

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



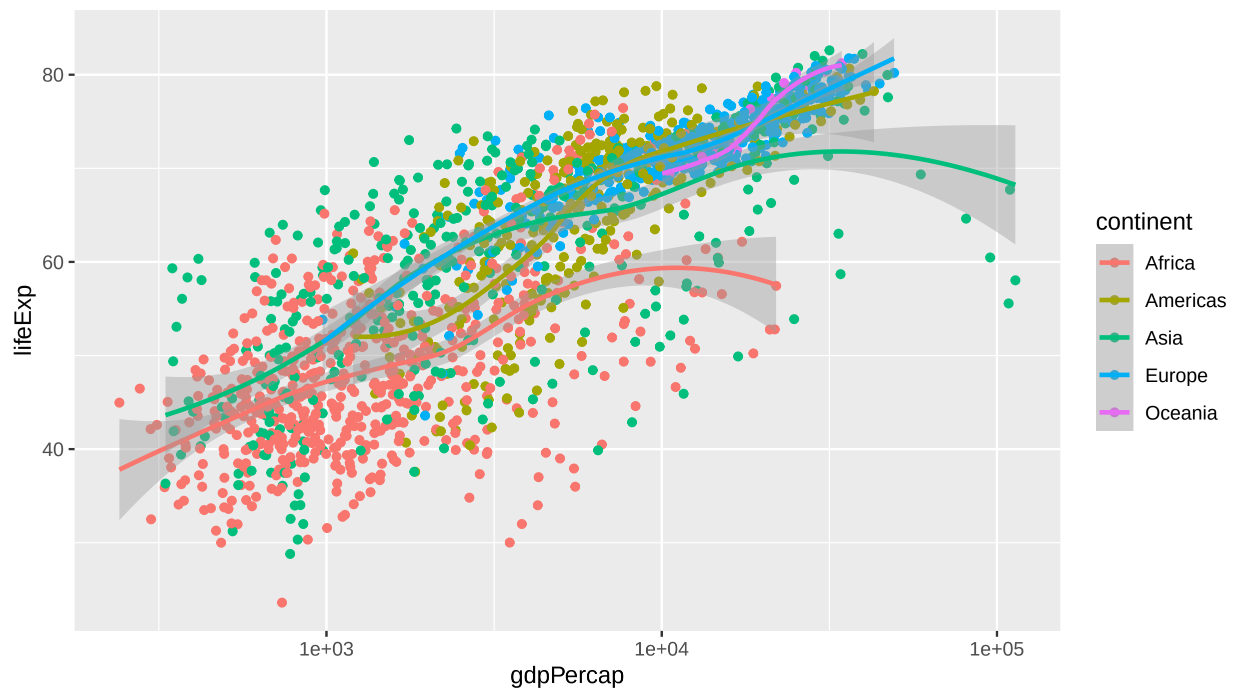
```
#final one :)
p + geom_point(alpha=0.4) +
  geom_smooth(method = "gam") +
  scale_x_log10(labels = scales::dollar) +
  labs(x="GDP per Capital", y="Life Expectancy in Years",
       title="Economic Growth and Life Expectancy",
       subtitle = "Data points are country-years",
       caption = "Source: Gapminder.")
```

```
## 'geom_smooth()' using formula = 'y ~ s(x, bs = "cs")'
```



```
p <- ggplot(data = gapminder, mapping = aes(gdpPercap,lifeExp,
                                             color=continent))
p+geom_point()+
  geom_smooth(method = "loess")+
  scale_x_log10()
```

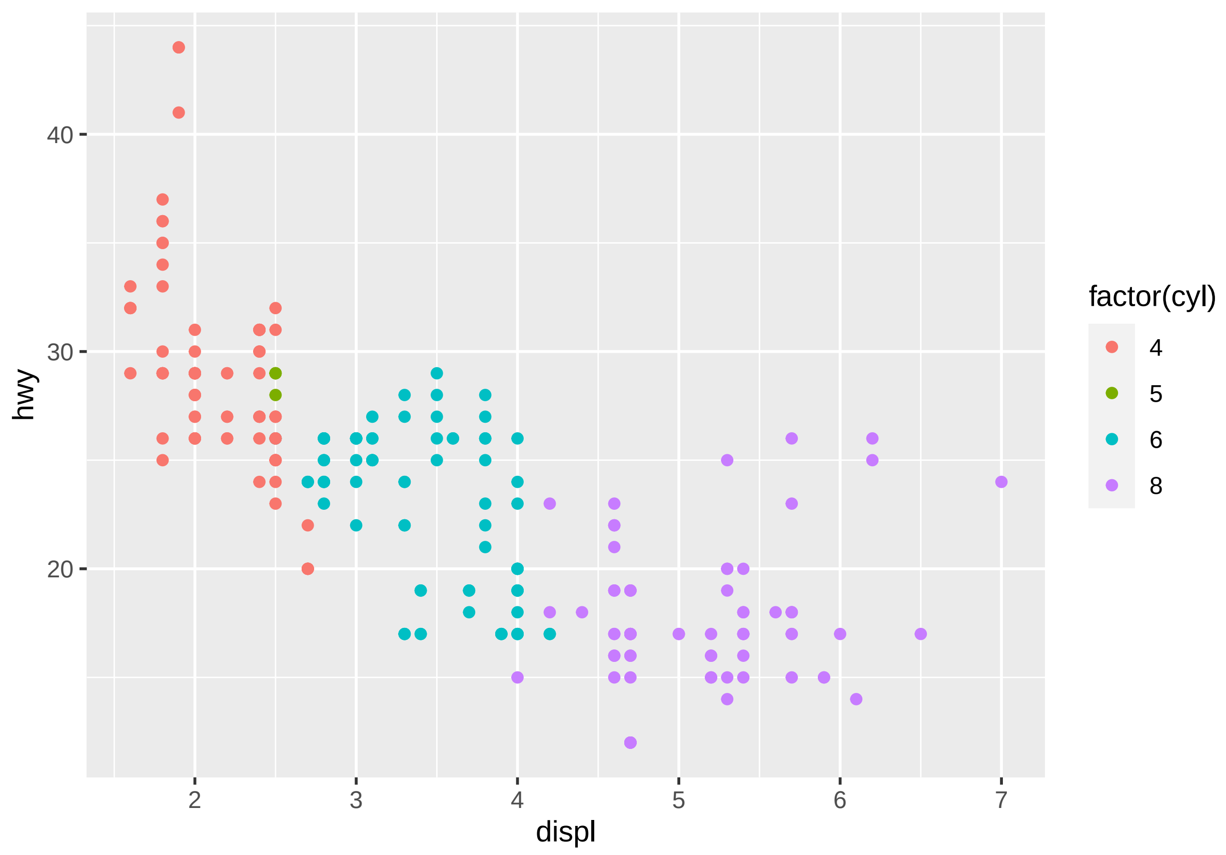
```
## `geom_smooth()` using formula = 'y ~ x'
```



Building a scatterplot

How are engine size and fuel economy related? We might create a scatterplot of engine displacement and highway mpg with points coloured by number of cylinders:

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

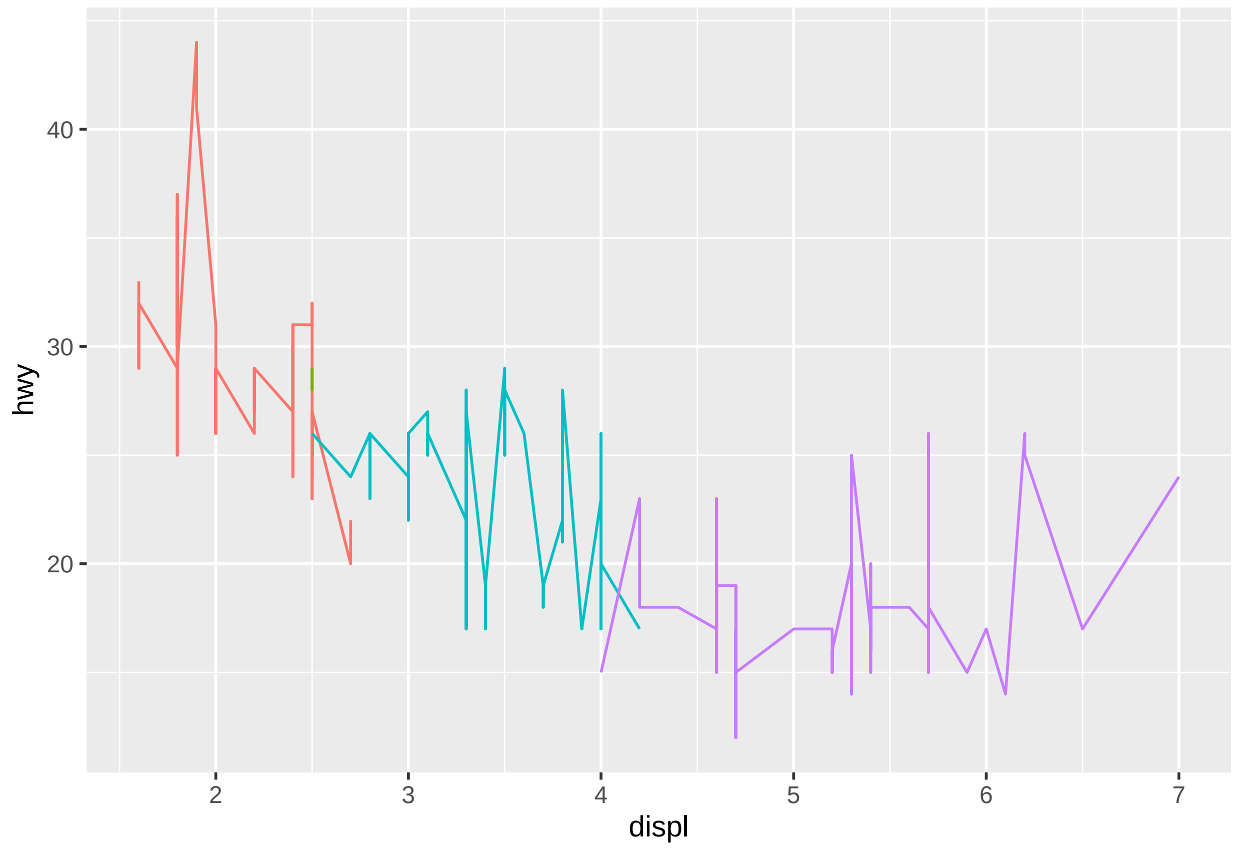


```r
str(mtcars)
```

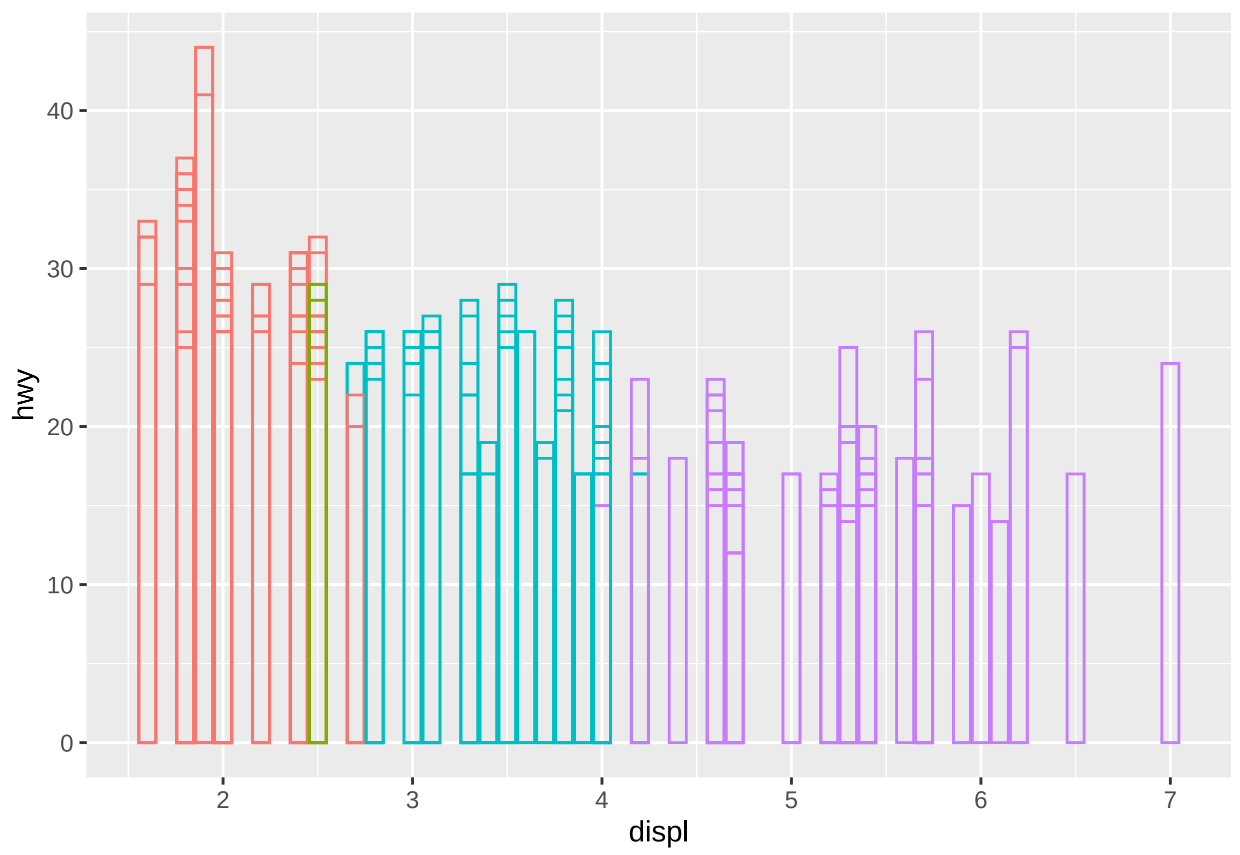
```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

The scatterplot uses points, but were we instead to draw lines we would get a line plot. If we used bars, we'd get a bar plot. Neither of those examples makes sense for this data, but we could still draw them:

```r
ggplot(mpg, aes(displ, hwy, colour = factor(cyl))) +
  geom_line() +
  theme(legend.position = "none")
```



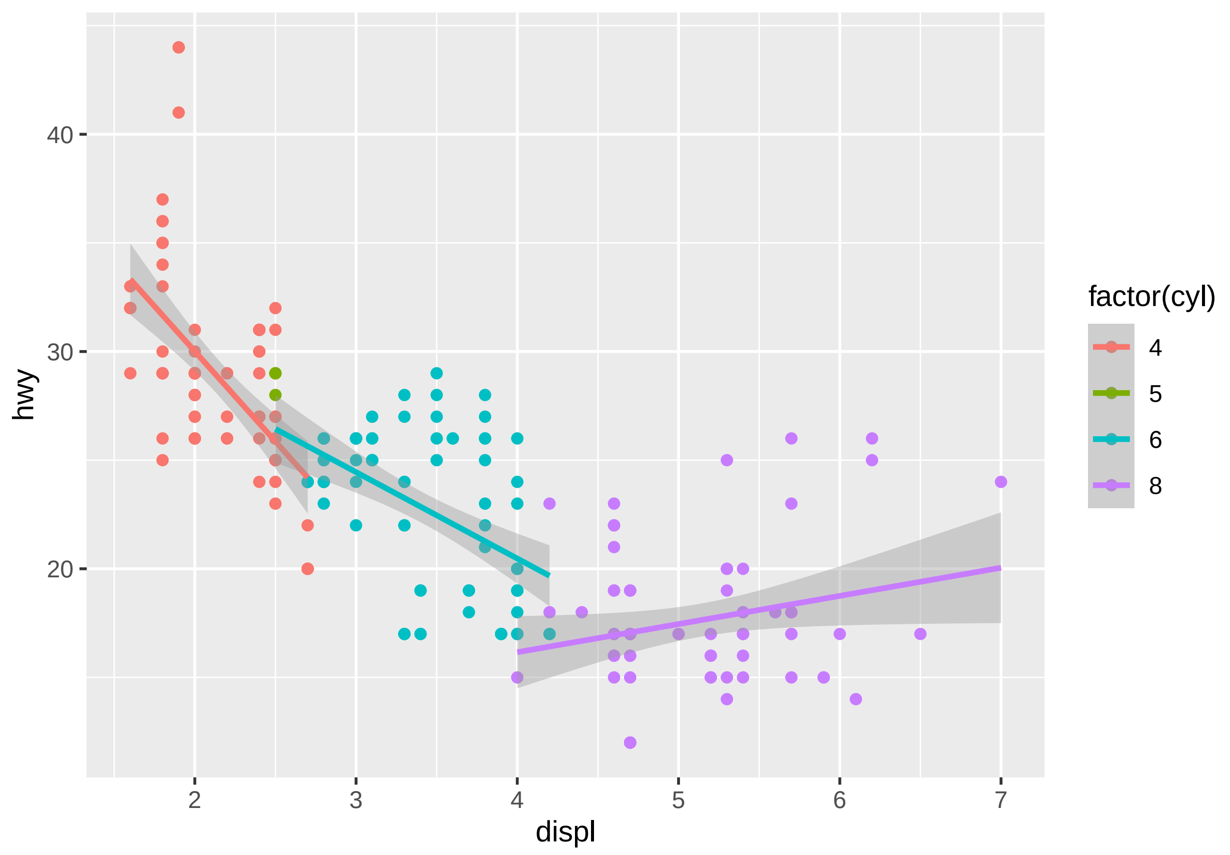
```
ggplot(mpg, aes(displ, hwy, colour = factor(cyl))) +
  geom_bar(stat = "identity", position = "identity", fill = NA) +
  theme(legend.position = "none")
```



More complex plots with combinations of multiple geoms don't have a special name, and we have to describe them by hand. For example, this plot overlays a per group regression line on top of a scatterplot:

```r
ggplot(mpg, aes(displ, hwy, colour = factor(cyl))) +
  geom_point() +
  geom_smooth(method = "lm")
```

```
## ‘geom_smooth()‘ using formula = ’y ~ x’
```



Adding complexity

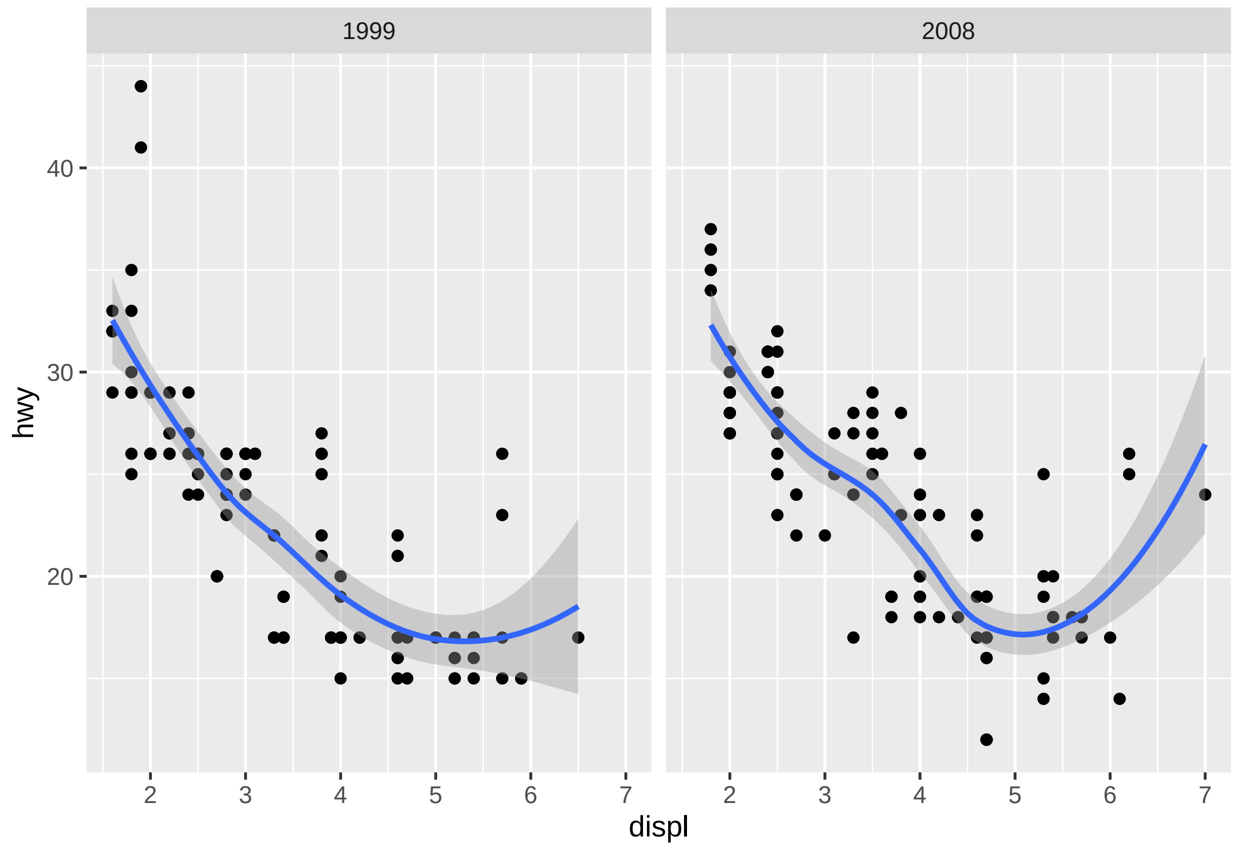
This plot adds three new components to the mix: facets, multiple layers and statistics. The facets and layers expand the data structure described below: each facet panel in each layer has its own dataset. You can think of this as a 3d array: the panels of the facets form a 2d grid, and the layers extend upwards in the 3rd dimension. In this case the data in the layers is the same, but in general we can plot different datasets on different layers.

```
names(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth() +
  facet_wrap(~year)
```

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



The data on each layer doesn't need to be the same, and it's often useful to combine multiple datasets in a single plot. To illustrate that idea I'm going to generate two new datasets related to the mpg dataset. First I'll fit a loess model and generate predictions from it. (This is what geom_smooth() does behind the scenes)

```
library(tidyverse)
mod <- loess(hwy ~ displ, data = mpg)
grid <- tibble(displ = seq(min(mpg$displ), max(mpg$displ), length = 50))
grid$hwy <- predict(mod, newdata = grid)
```

```
grid
```

```
## # A tibble: 50 x 2
##    displ   hwy
##    <dbl> <dbl>
##  1  1.6   33.1
##  2  1.71  32.2
##  3  1.82  31.3
##  4  1.93  30.4
##  5  2.04  29.6
##  6  2.15  28.8
##  7  2.26  28.1
##  8  2.37  27.4
##  9  2.48  26.7
## 10  2.59  26.1
## # i 40 more rows
```

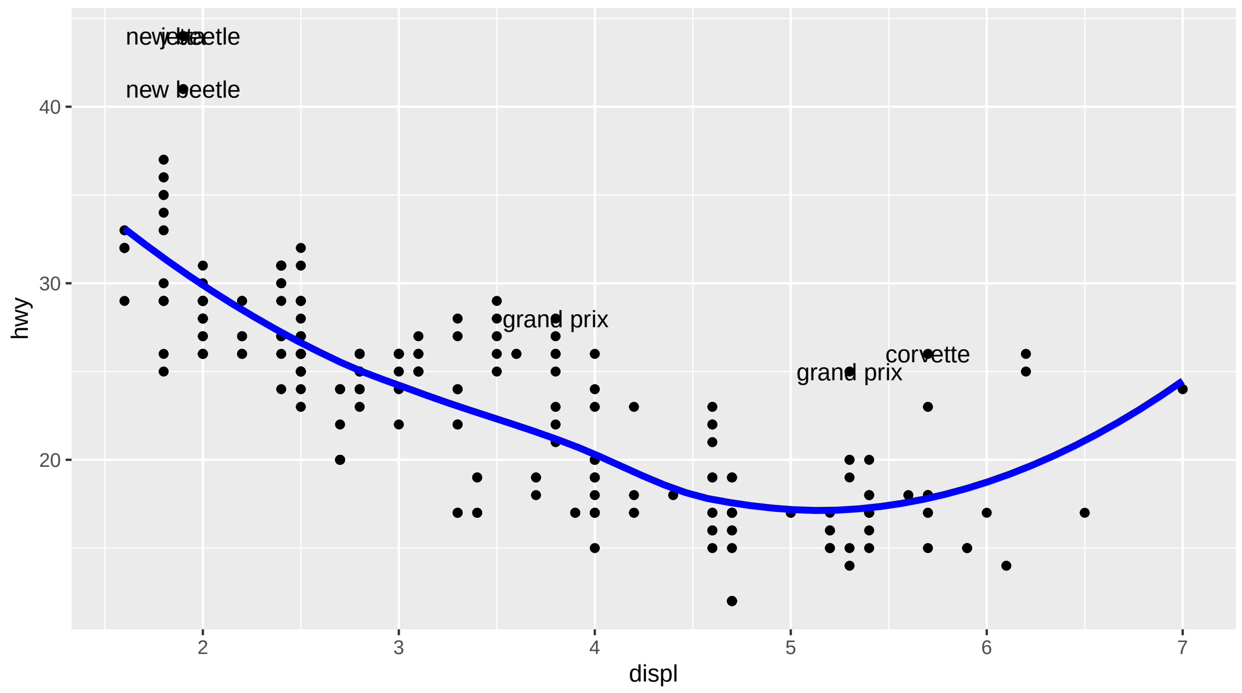
Next, we'll isolate observations that are particularly far away from their predicted values:

```r
std_resid <- resid(mod) / mod$s
outlier <- filter(mpg, abs(std_resid) > 2)
outlier
```

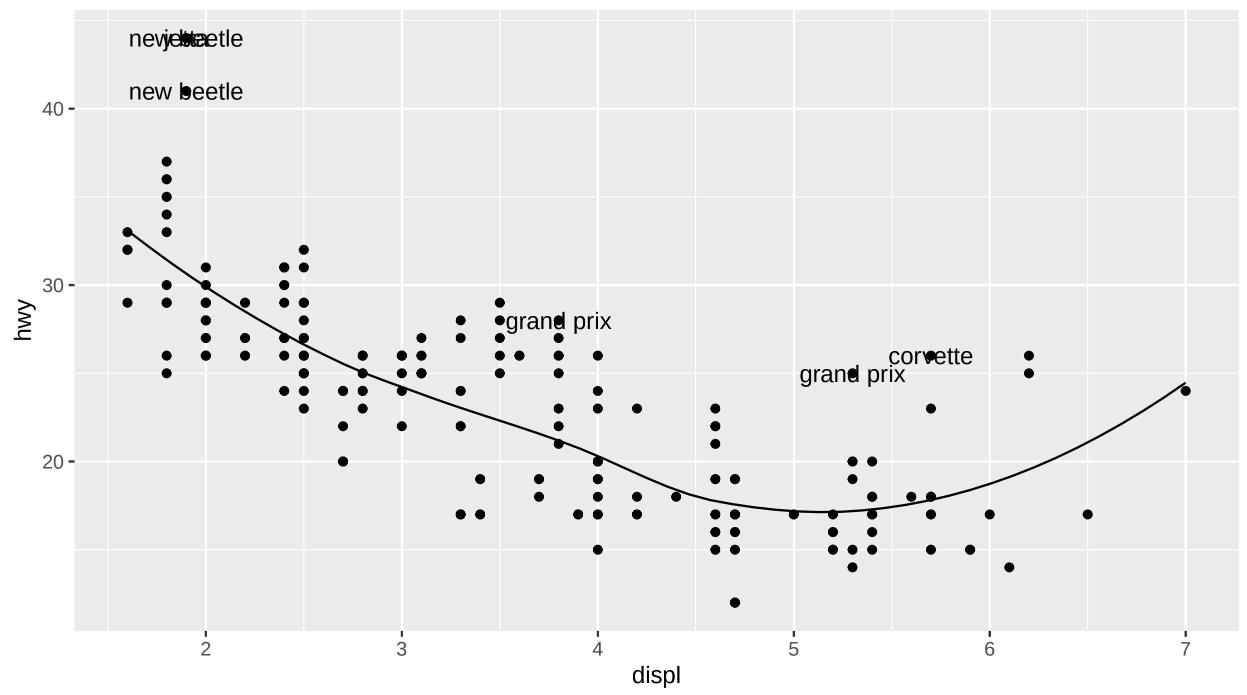
```
## # A tibble: 6 x 11
##   manufacturer model      displ  year   cyl trans  drv     cty   hwy fl    class
##   <chr>        <chr>      <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
## 1 chevrolet    corvette     5.7  1999     8 manua~ r        16    26 p     2sea~
## 2 pontiac      grand prix   3.8  2008     6 auto(~ f        18    28 r     mids~
## 3 pontiac      grand prix   5.3  2008     8 auto(~ f        16    25 p     mids~
## 4 volkswagen   jetta        1.9  1999     4 manua~ f        33    44 d     comp~
## 5 volkswagen   new beetle   1.9  1999     4 manua~ f        35    44 d     subc~
## 6 volkswagen   new beetle   1.9  1999     4 auto(~ f        29    41 d     subc~
```

I've generated these datasets because it's common to enhance the display of raw data with a statistical summary and some annotations. With these new datasets, I can improve our initial scatterplot by overlaying a smoothed line, and labelling the outlying points:

```r
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_line(data = grid, colour = "blue", size = 1.5) +
  geom_text(data = outlier, aes(label = model))
```



```r
ggplot(mapping = aes(displ, hwy)) +
  geom_point(data = mpg) +
  geom_line(data = grid) +
  geom_text(data = outlier, aes(label = model))
```



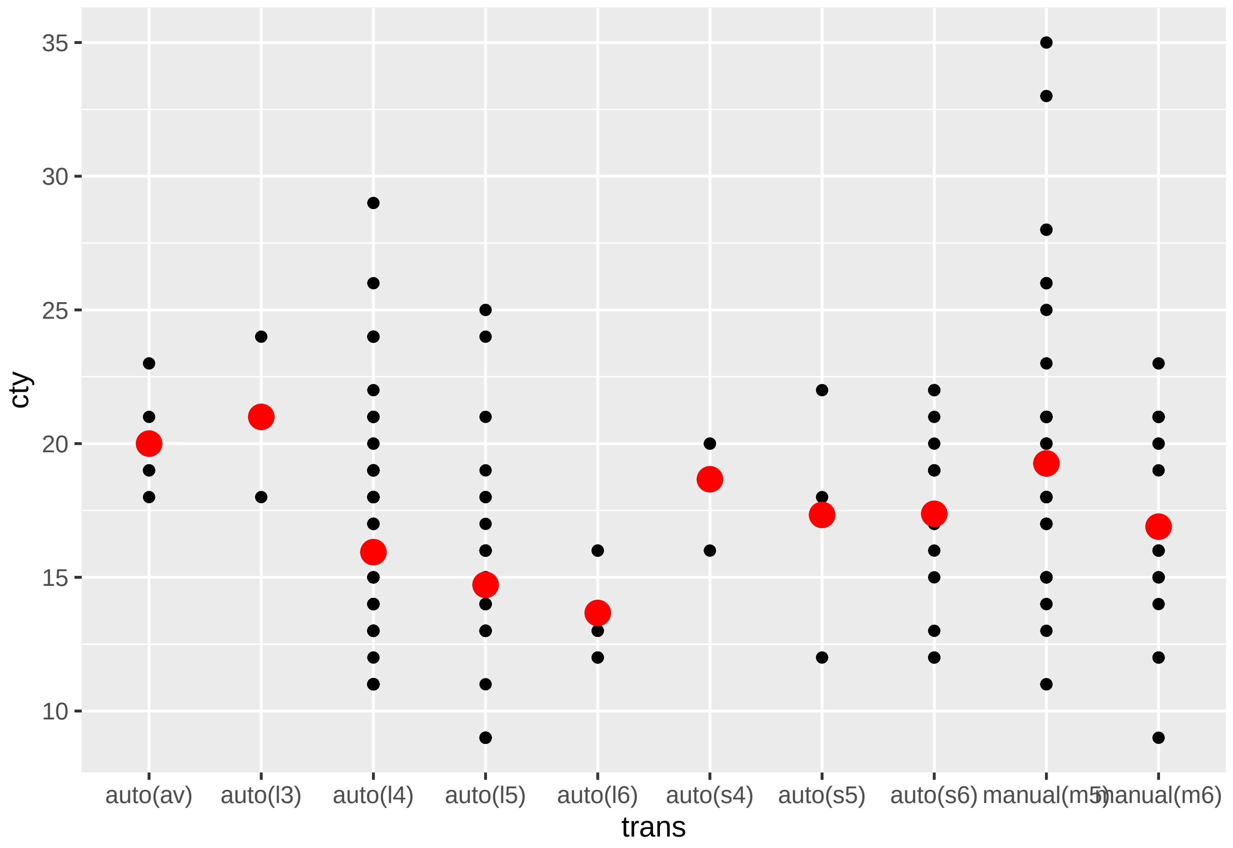
```r
head(mpg)
```

```
## # A tibble: 6 x 11
##   manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa~
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa~
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa~
## 4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa~
## 5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compa~
## 6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p     compa~
```

A statistical transformation, or stat, transforms the data, typically by summarising it in some manner. For example, a useful stat is the smoother, which calculates the smoothed mean of y, conditional on x. You've already used many of ggplot2's stats because they're used behind the scenes to generate many important geoms: • stat_bin(): geom_bar(), geom_freqpoly(), geom_histogram() • stat_bin2d(): geom_bin2d() • stat_bindot(): geom_dotplot() • stat_binhex(): geom_hex() • stat_boxplot(): geom_boxplot() • stat_contour(): geom_contour() • stat_quantile(): geom_quantile() • stat_smooth(): geom_smooth() • stat_sum(): geom_count() Other stats can't be created with a geom_ function: • stat_ecdf(): compute a empirical cumulative distribution plot. • stat_function(): compute y values from a function of x values. • stat_summary(): summarise y values at distinct x values. • stat_summary2d(), stat_summary_hex(): summarise binned values. • stat_qq(): perform calculations for a quantile-quantile plot. • stat_spoke(): convert angle and radius to position. • stat_unique(): remove duplicated rows. There are two ways to use these functions. You can either add a stat_() function and override the default geom, or add a geom_() function and override the default stat:

```r
ggplot(mpg, aes(trans, cty)) +
  geom_point() +
  stat_summary(geom = "point", fun.y = "mean", colour = "red", size = 4)
```

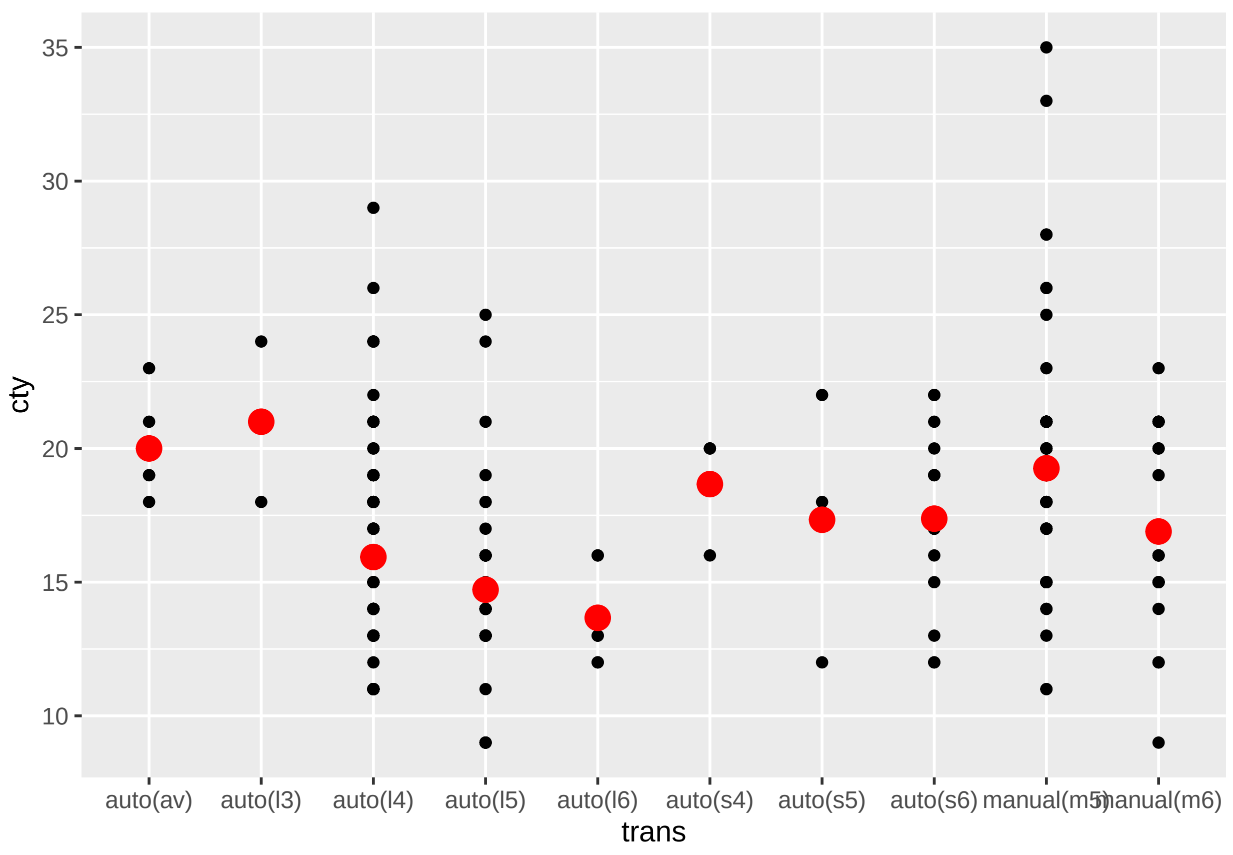
```
## Warning: The `fun.y` argument of `stat_summary()` is deprecated as of ggplot2 3.3.0.
## i Please use the `fun` argument instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



```
ggplot(mpg, aes(trans, cty)) +
  geom_point() +
  geom_point(stat = "summary", fun.y = "mean", colour = "red", size = 4)
```

```
## Warning in geom_point(stat = "summary", fun.y = "mean", colour = "red", :
## Ignoring unknown parameters: `fun.y`
```

```
## No summary function supplied, defaulting to `mean_se()`
```



I think it's best to use the second form because it makes it more clear that you're displaying a summary, not the raw data.

Generated Variables

Internally, a stat takes a data frame as input and returns a data frame as output, and so a stat can add new variables to the original dataset. It is possible to map aesthetics to these new variables. For example, stat_bin, the statistic used to make histograms, produces the following variables: • count, the number of observations in each bin • density, the density of observations in each bin (percentage of total / bar width) • x, the centre of the bin These generated variables can be used instead of the variables present in the original dataset. For example, the default histogram geom assigns the height of the bars to the number of observations (count), but if you'd prefer a more traditional histogram, you can use the density (density). To refer to a generated variable like density, ".." must surround the name. This prevents confusion in case the original dataset includes a variable with the same name as a generated variable, and it makes it clear to any later reader of the code that this variable was generated by a stat. Each statistic lists the variables that it creates in its documentation. Compare the y-axes on these two plots:

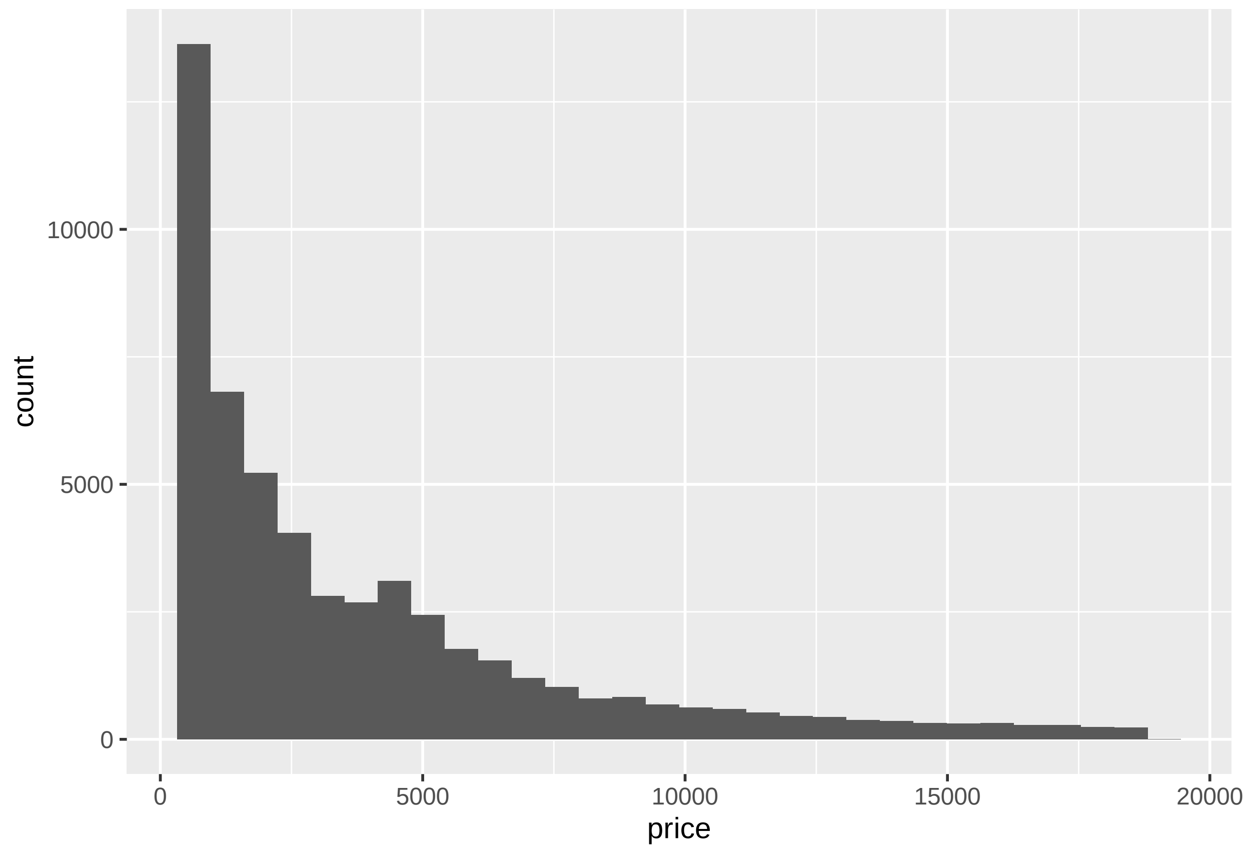
```
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4  0.29 Premium   I     VS2      62.4    58    334  4.2   4.23  2.63
## 5  0.31 Good      J     SI2      63.3    58    335  4.34  4.35  2.75
## 6  0.24 Very Good J     VVS2     62.8    57    336  3.94  3.96  2.48
```

```
ggplot(diamonds, aes(price)) +
  geom_histogram(bindwidth = 500)
```

```
## Warning in geom_histogram(bindwidth = 500): Ignoring unknown parameters:
## `bindwidth`
```

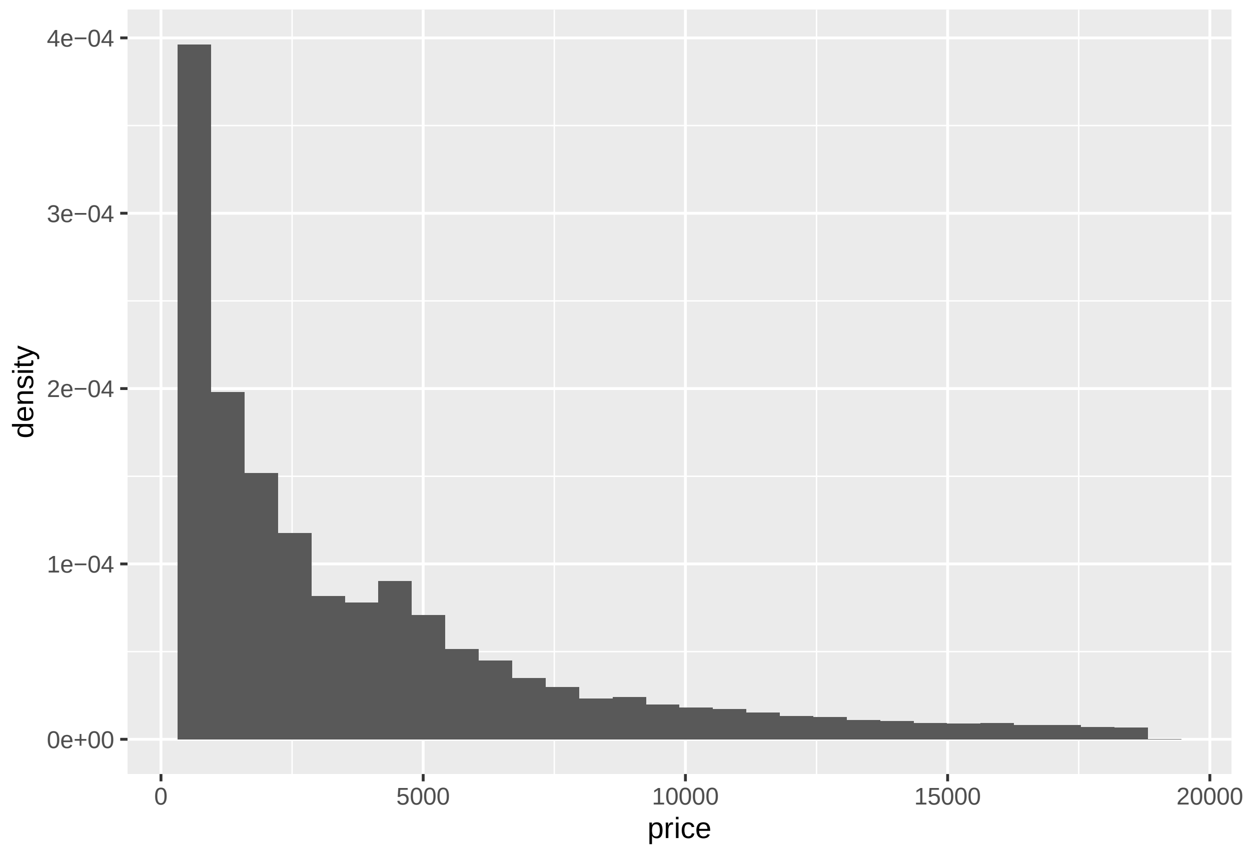
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
ggplot(diamonds, aes(price)) +
  geom_histogram(aes(y = ..density..), bindwidth = 500)
```

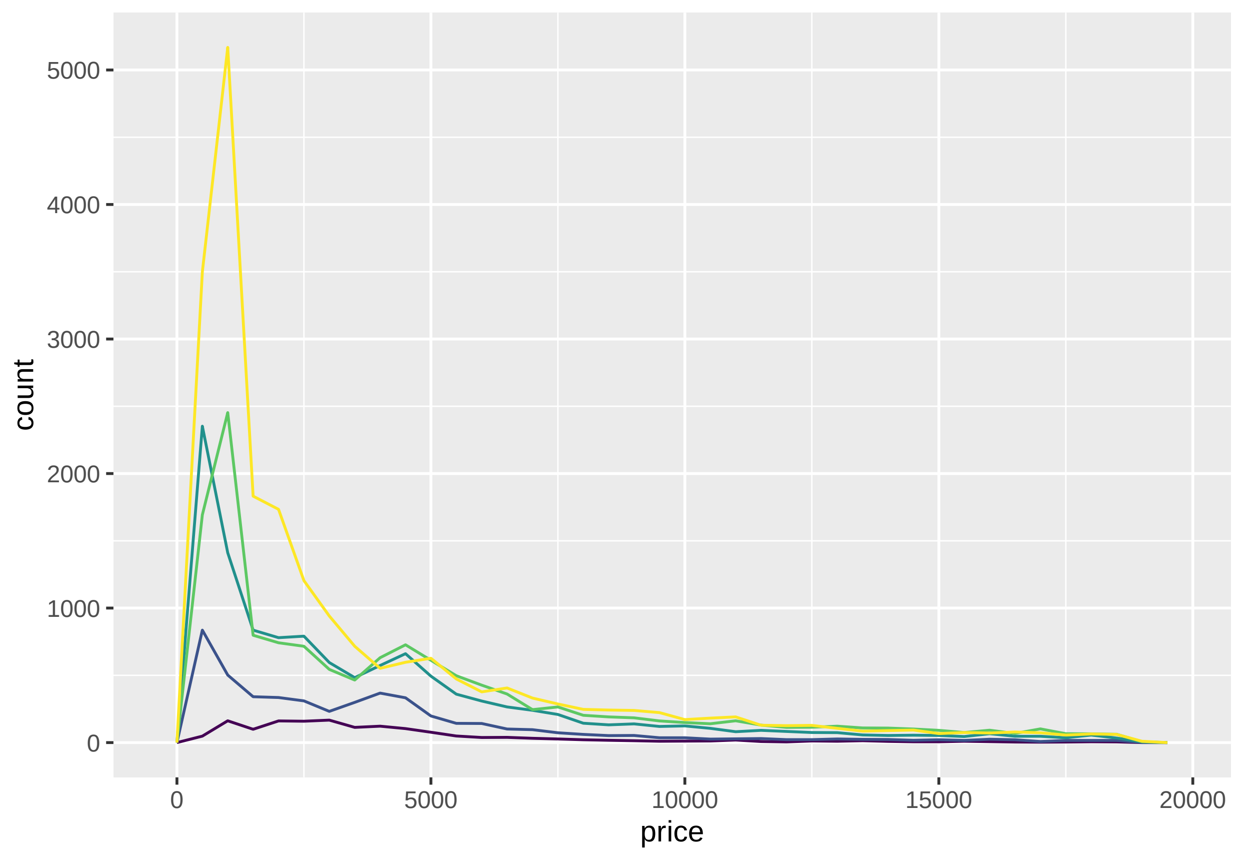
```
## Warning in geom_histogram(aes(y = ..density..), bindwidth = 500): Ignoring
## unknown parameters: `bindwidth`
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

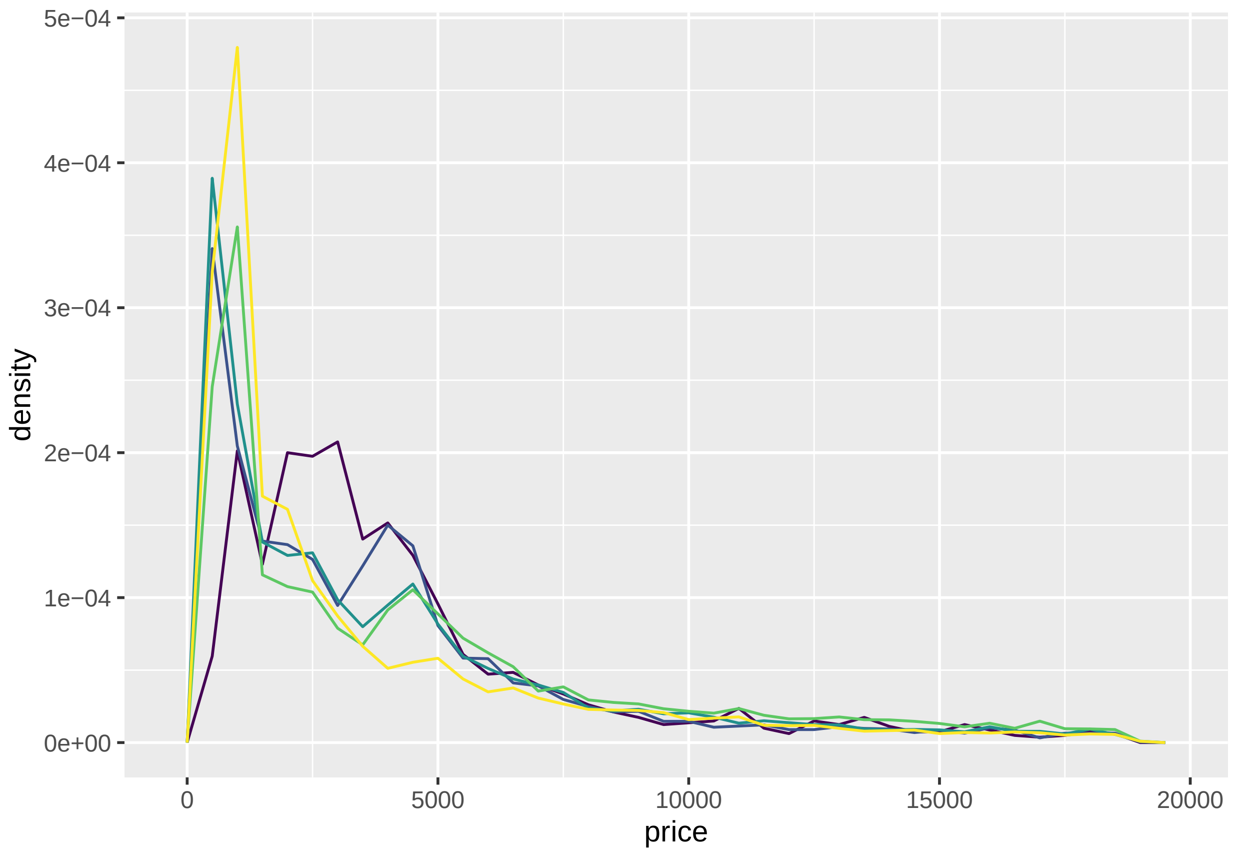


This technique is particularly useful when you want to compare the distribution of multiple groups that have very different sizes. For example, it's hard to compare the distribution of price within cut because some groups are quite small. It's easier to compare if we standardise each group to take up the same area:

```
ggplot(diamonds, aes(price, colour = cut)) +
  geom_freqpoly(binwidth = 500) +
  theme(legend.position = "none")
```



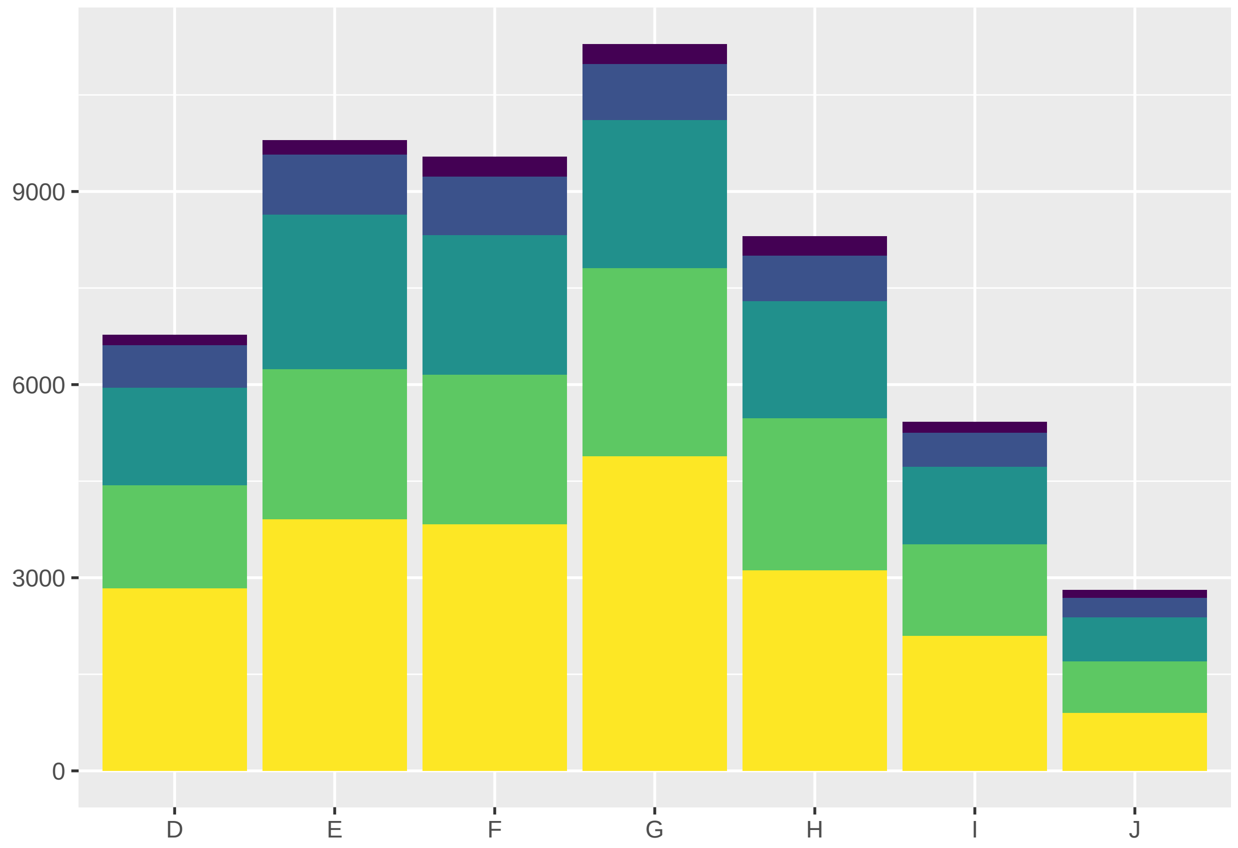
```
ggplot(diamonds, aes(price, colour = cut)) +
  geom_freqpoly(aes(y = ..density..), binwidth = 500) +
  theme(legend.position = "none")
```



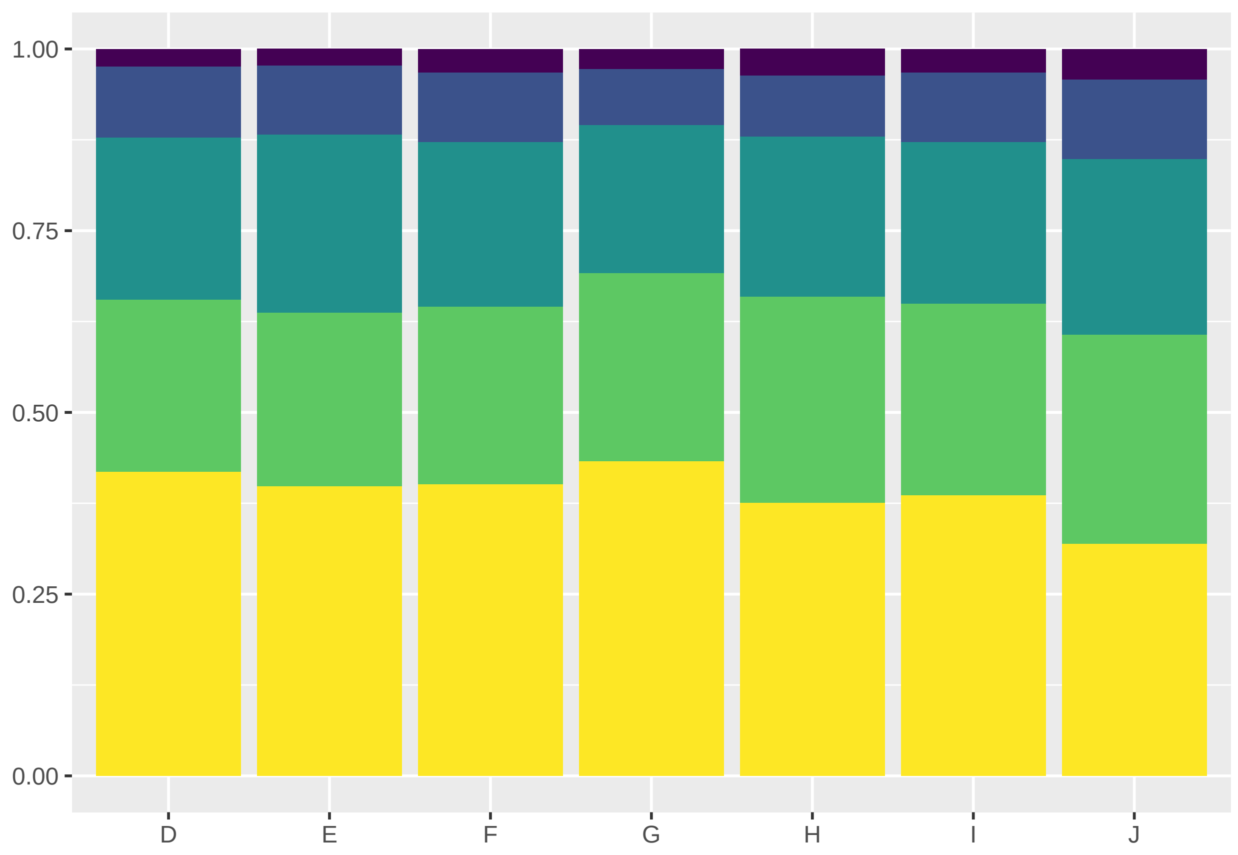
Position Adjustments

Position adjustments apply minor tweaks to the position of elements within a layer. Three adjustments apply primarily to bars: • position_stack(): stack overlapping bars (or areas) on top of each other. • position_fill(): stack overlapping bars, scaling so the top is always at 1. • position_dodge(): place overlapping bars (or boxplots) side-by-side.

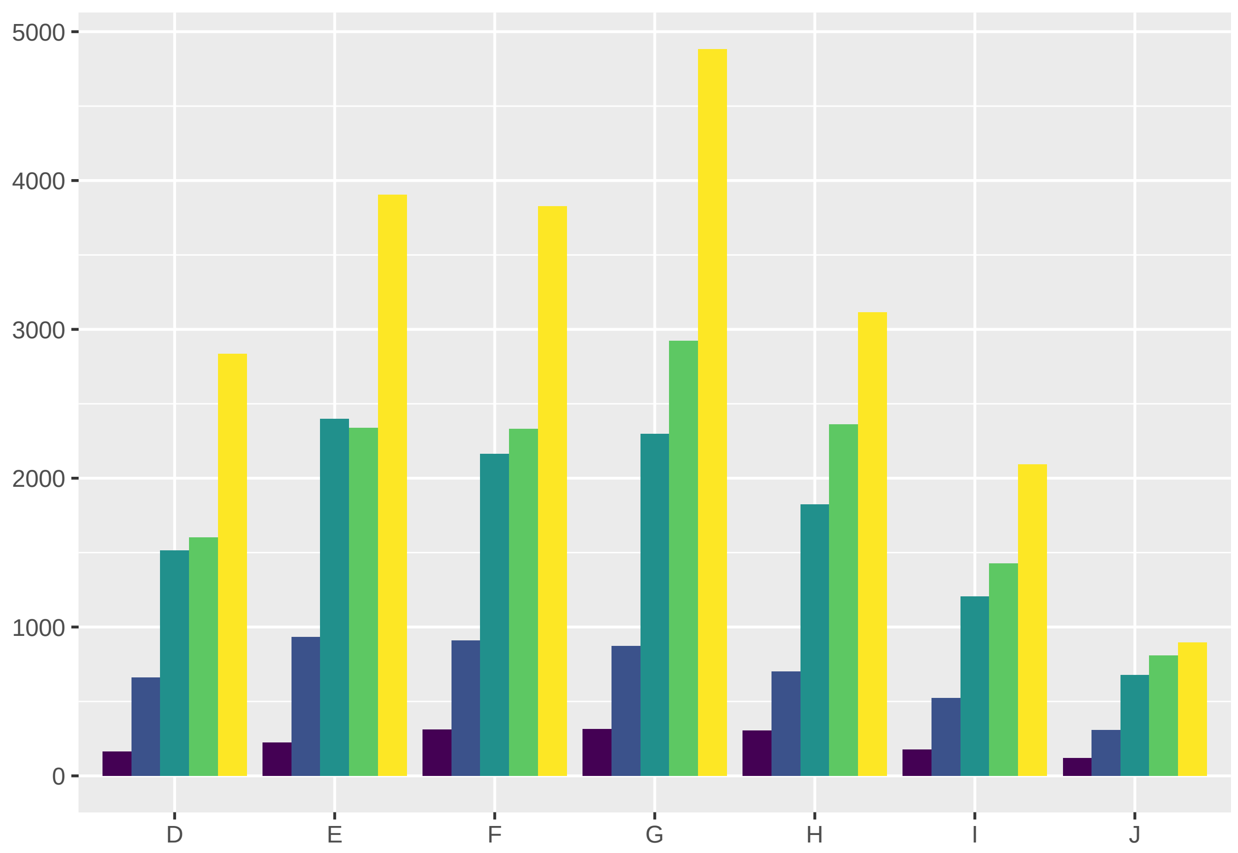
```
dplot <- ggplot(diamonds, aes(color, fill = cut)) +
  xlab(NULL) + ylab(NULL) + theme(legend.position = "none")
# position stack is the default for bars, so `geom_bar()`
# is equivalent to `geom_bar(position = "stack")`.
dplot + geom_bar()
```



```r
dplot + geom_bar(position = "fill")
```

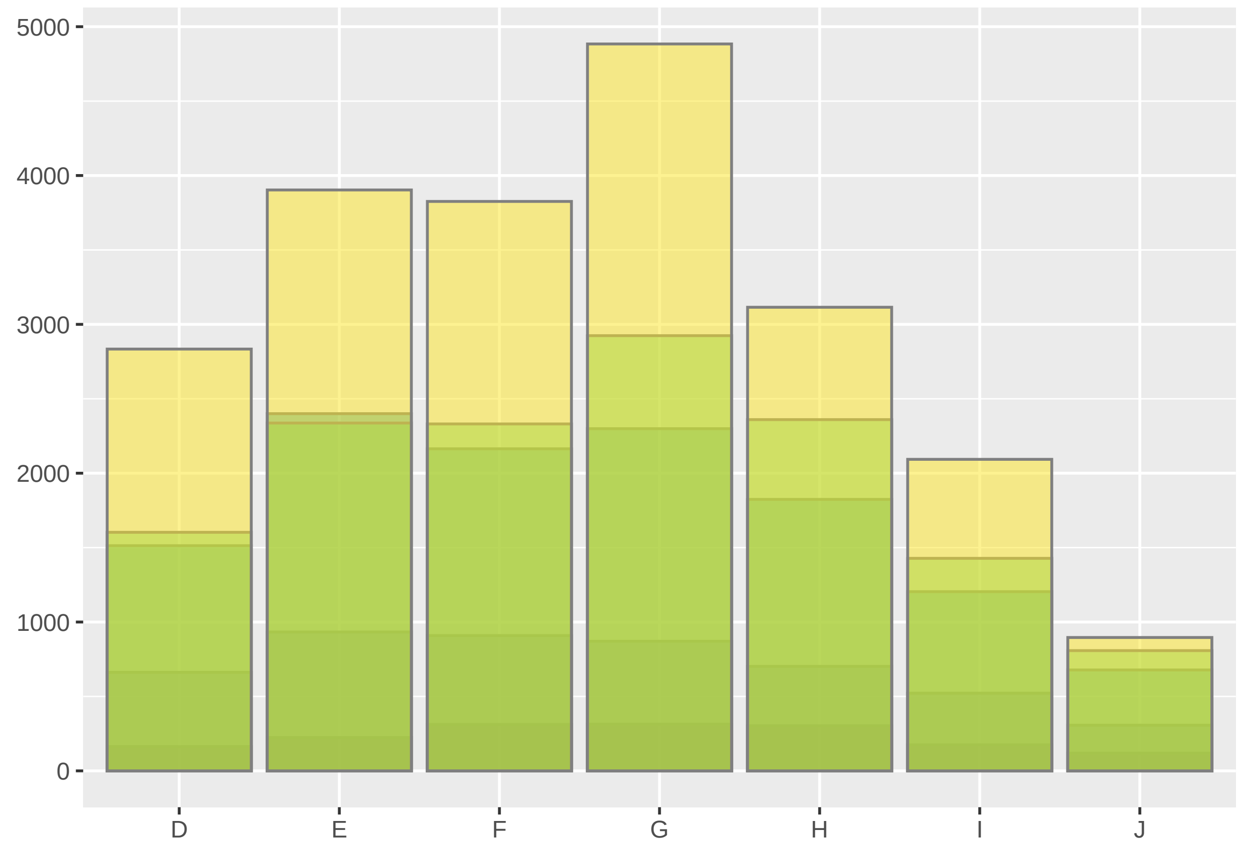


```
dplot + geom_bar(position = "dodge")
```

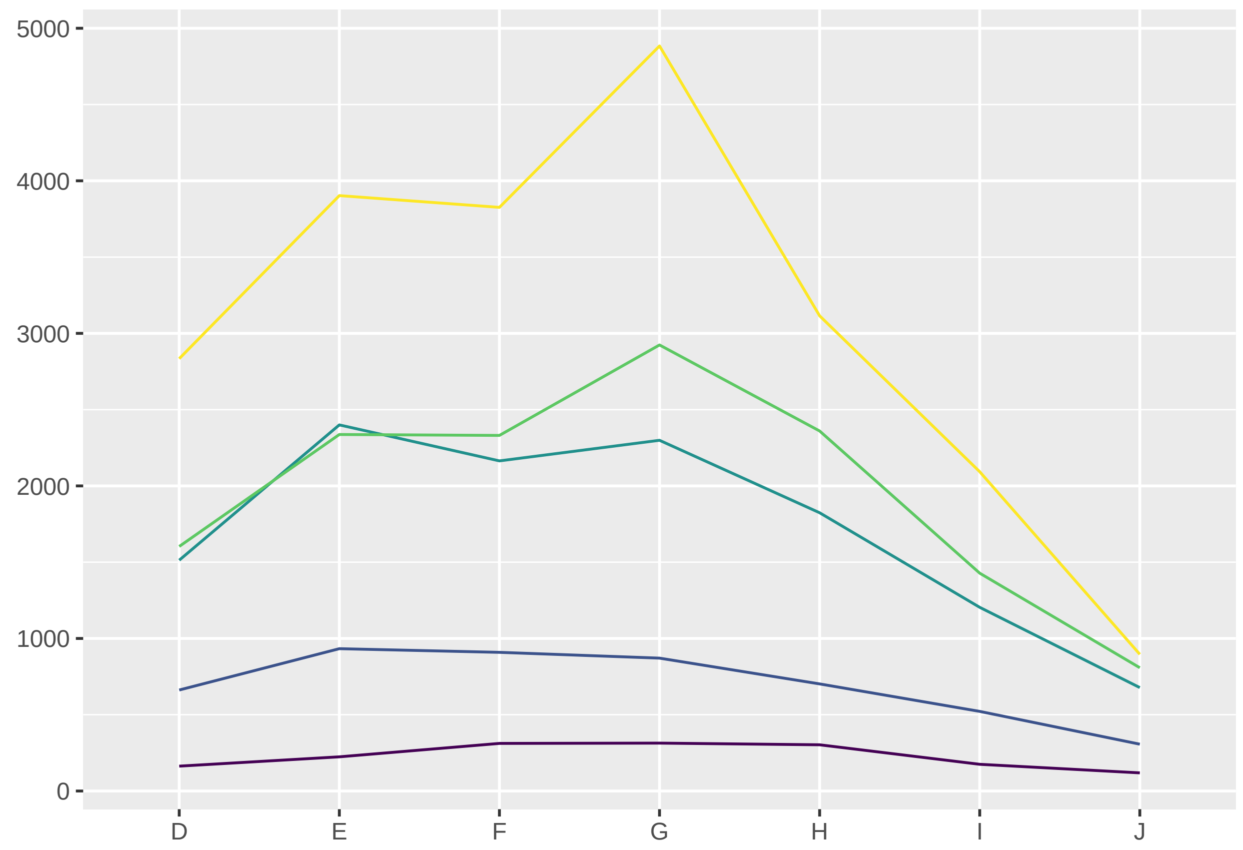


There's also a position adjustment that does nothing: position_identity(). The identity position adjustment is not useful for bars, because each bar obscures the bars behind, but there are many geoms that don't need adjusting, like the frequency polygon:

```
dplot + geom_bar(position = "identity", alpha = 1 / 2, colour = "grey50")
```



```
ggplot(diamonds, aes(color, colour = cut)) +
  geom_freqpoly(aes(group = cut), stat = "count") +
  xlab(NULL) + ylab(NULL) +
  theme(legend.position = "none")
```

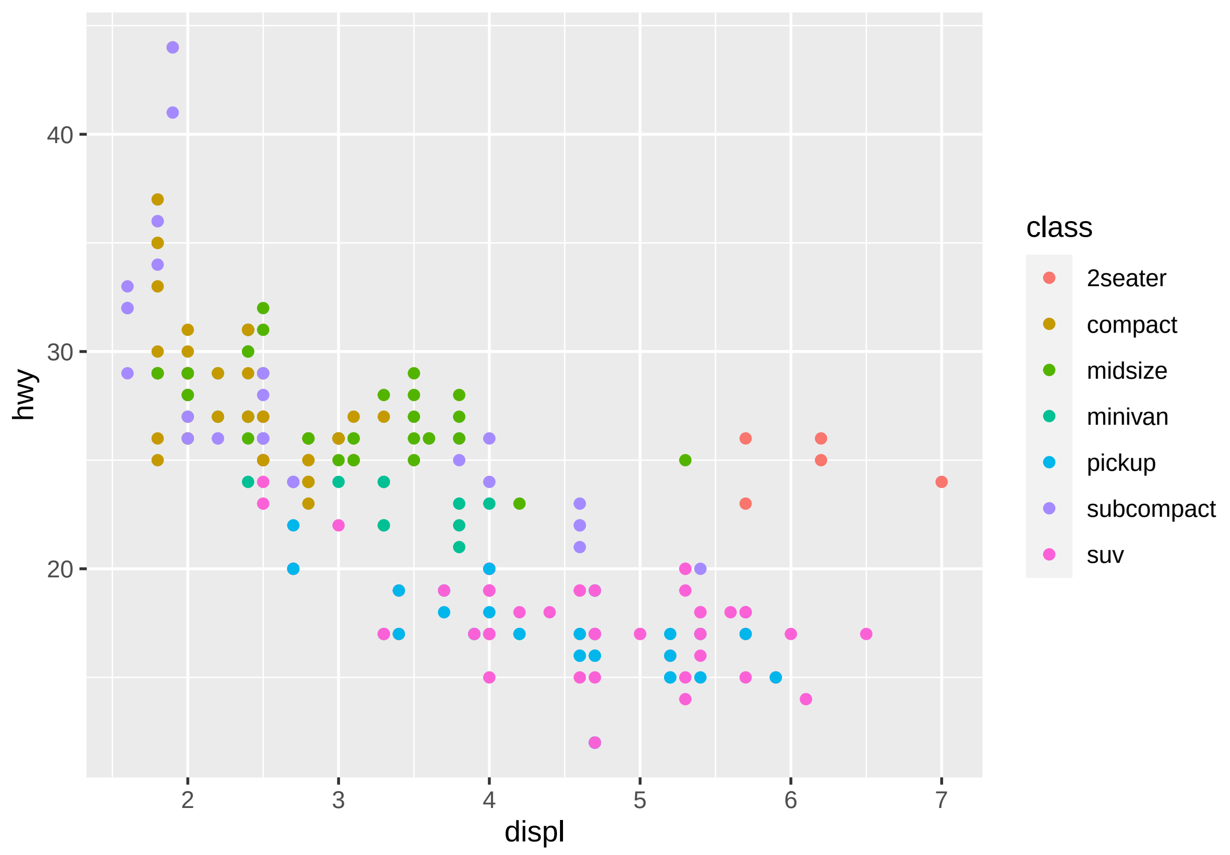


# Scales, axes, and legends Scales control the mapping from data to aesthetics. They take your data and turn it into something that you can see, like size, colour, position or shape. Scales also provide the tools that let you read the plot: the axes and legends. Formally, each scale is a function from a region in data space (the domain of the scale) to a region in aesthetic space (the range of the scale). The axis or legend is the inverse function: it allows you to convert visual properties back to data.

Modifying scales

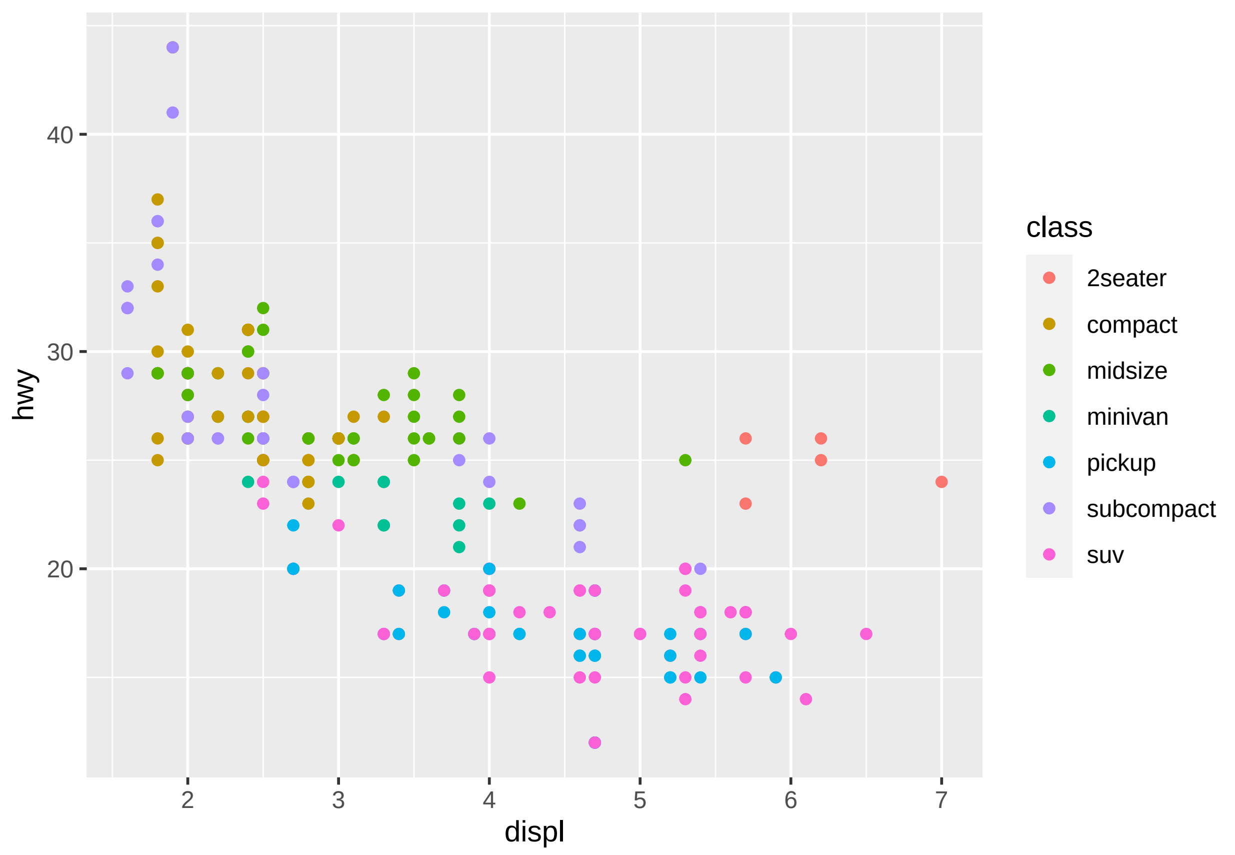
A scale is required for every aesthetic used on the plot. When you write:

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



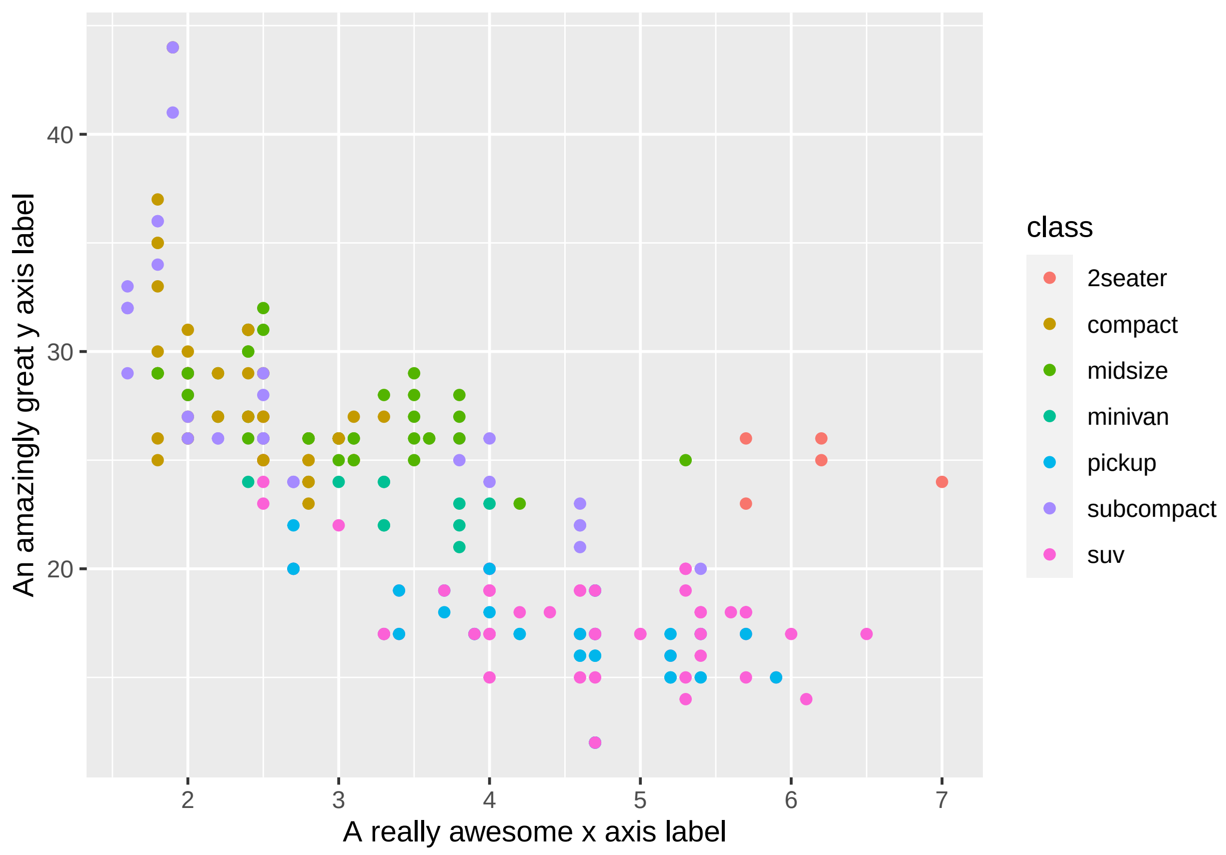
What actually happens is this:

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



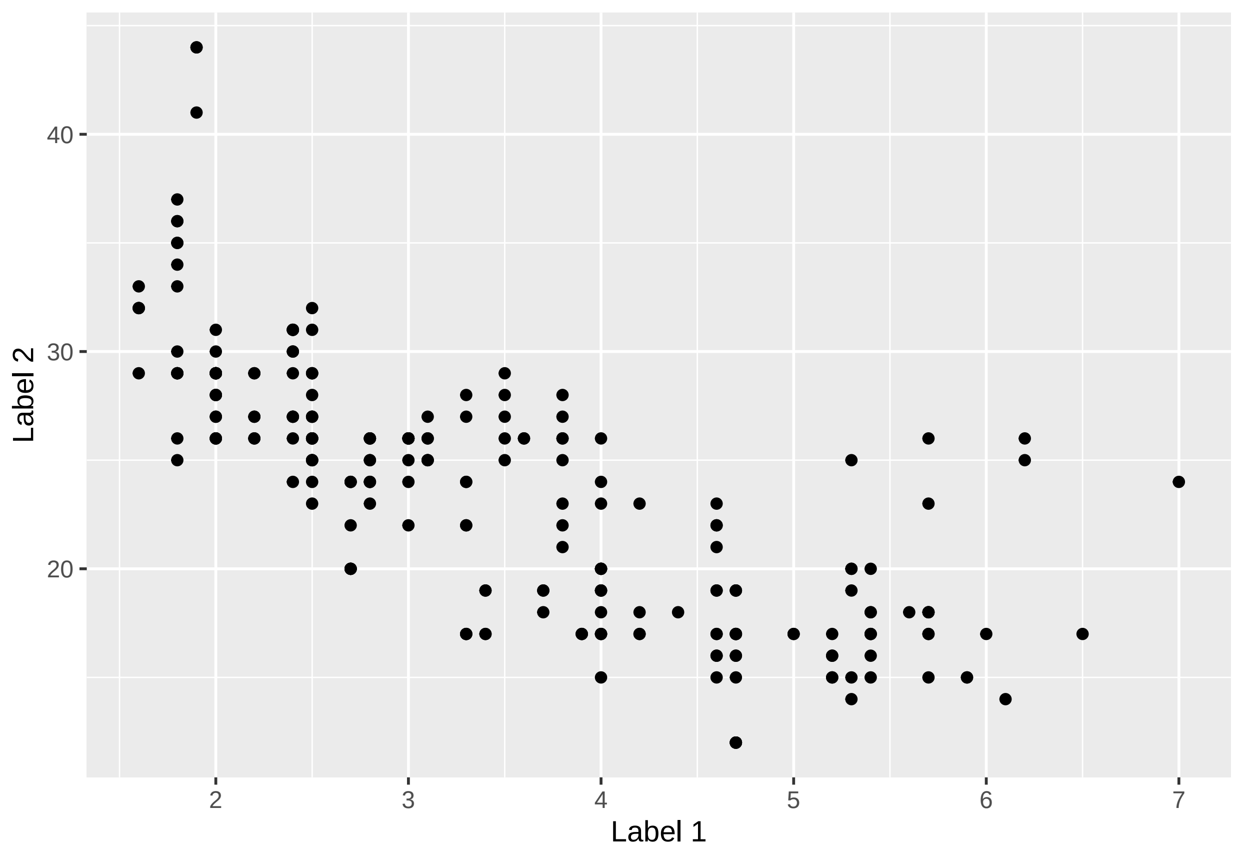
Default scales are named according to the aesthetic and the variable type: scale_y_continuous(), scale_colour_discrete(), etc. It would be tedious to manually add a scale every time you used a new aesthetic, so ggplot2 does it for you. But if you want to override the defaults, you'll need to add the scale yourself, like this:

```r
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class)) +
  scale_x_continuous("A really awesome x axis label") +
  scale_y_continuous("An amazingly great y axis label")
```



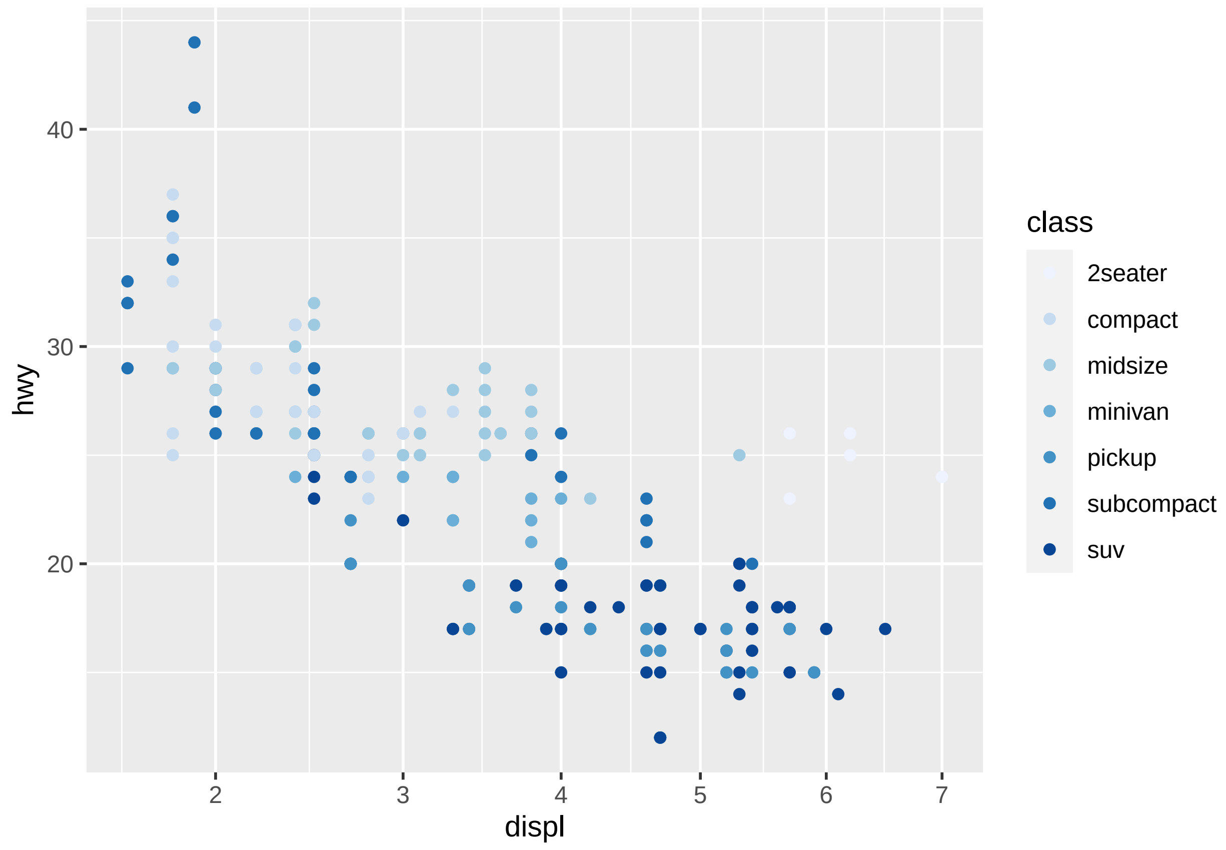
The use of + to "add" scales to a plot is a little misleading. When you + a scale, you're not actually adding it to the plot, but overriding the existing scale. This means that the following two specifications are equivalent:

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  scale_x_continuous("Label 1") +
  scale_y_continuous("Label 2")
```

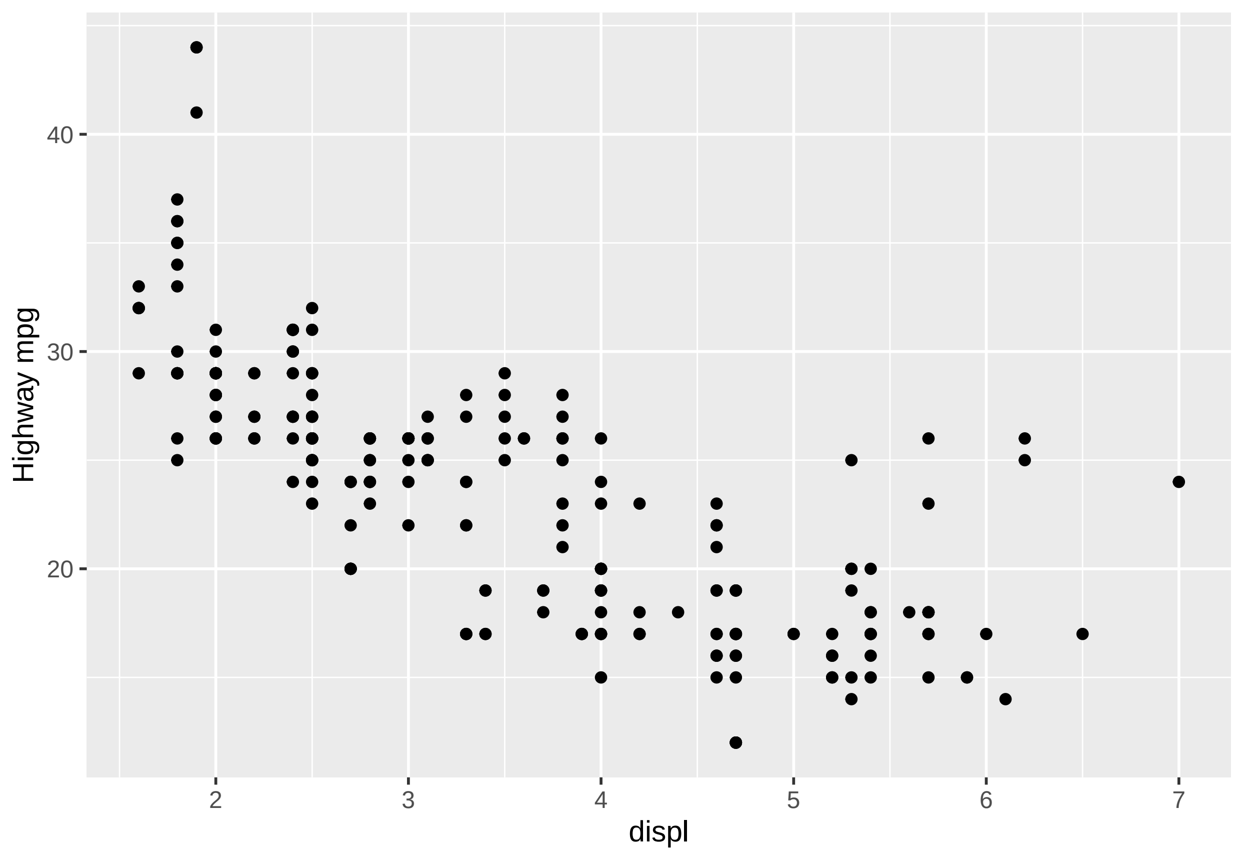


```
#> Scale for 'x' is already present. Adding another scale for 'x',
#> which will replace the existing scale.
```

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

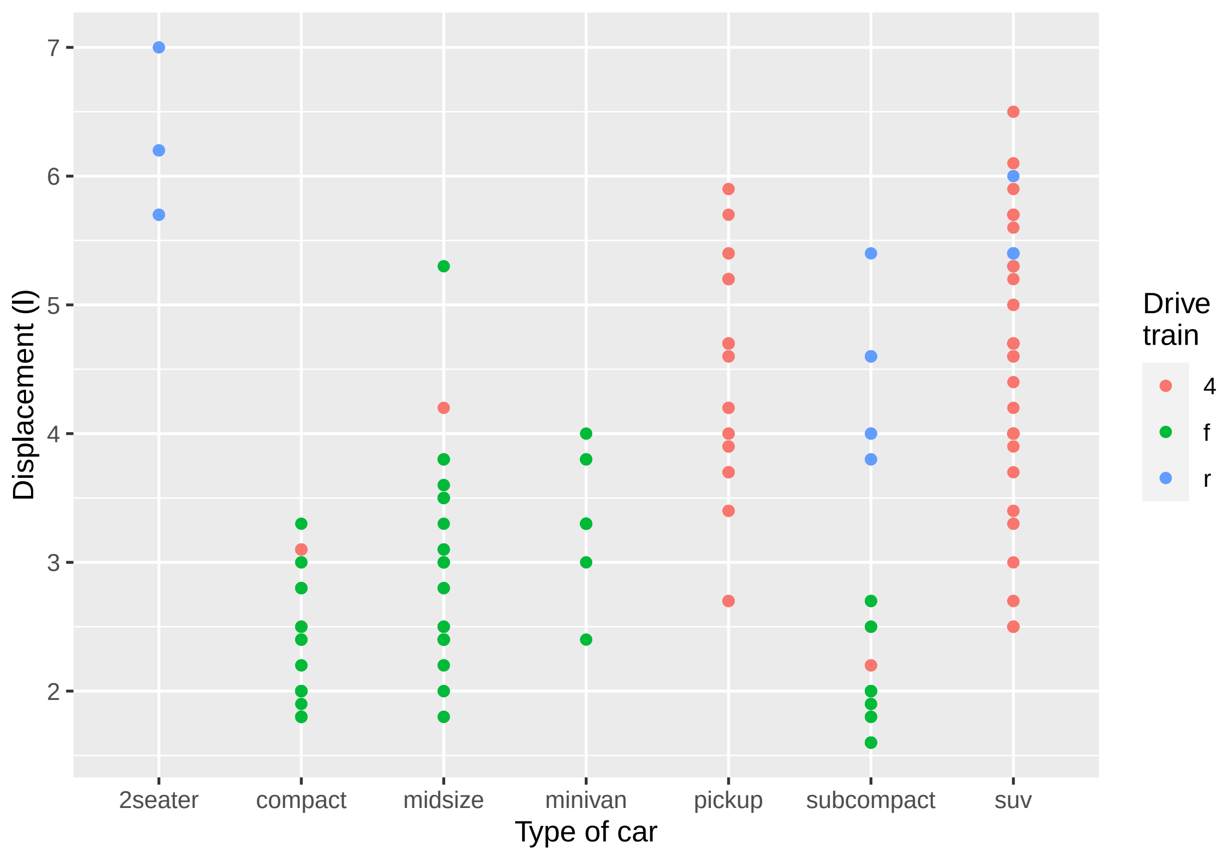


```r
ggplot(mpg, aes(displ)) +
scale_y_continuous("Highway mpg") +
scale_x_continuous() +
geom_point(aes(y = hwy))
```

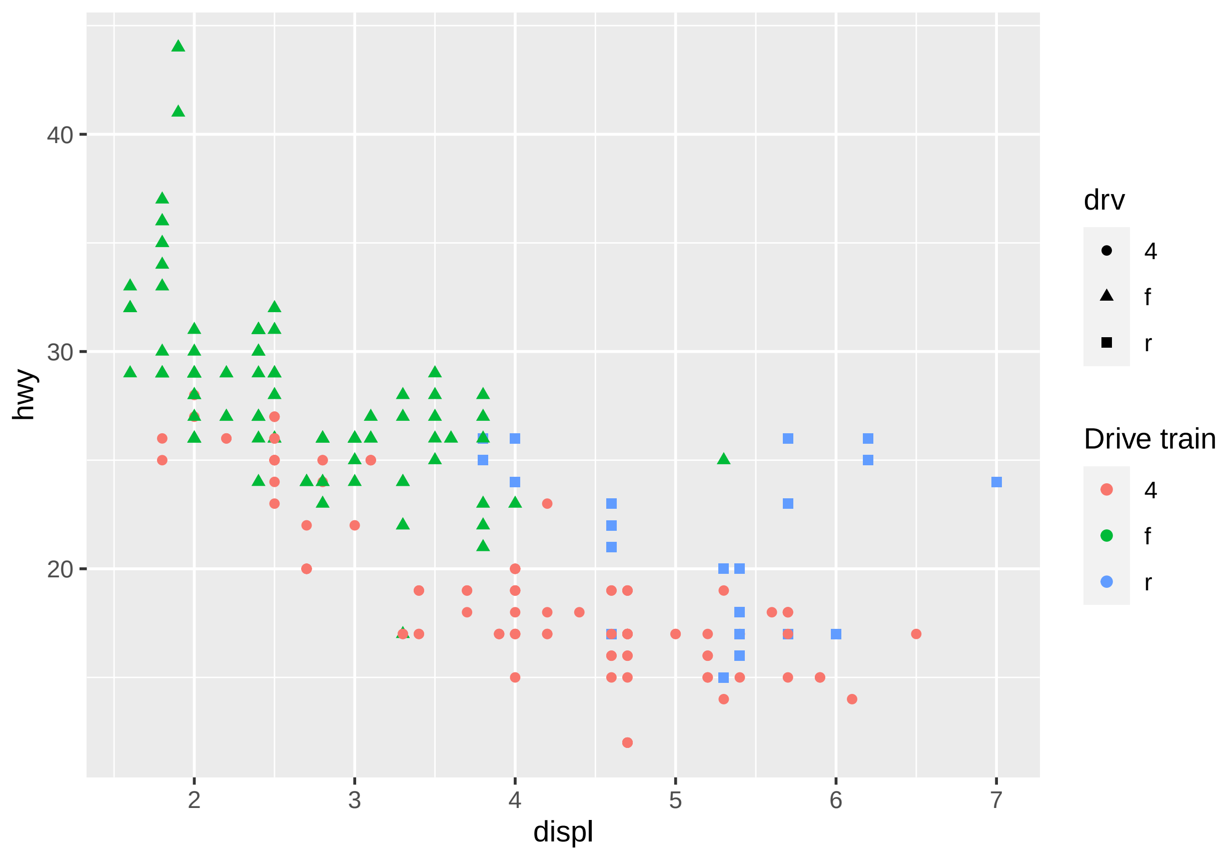


```
ggplot(mpg, aes(y = displ, x = class)) +
  scale_y_continuous("Displacement (l)") +
  scale_x_discrete("Car type") +
  scale_x_discrete("Type of car") +
  scale_colour_discrete() +
  geom_point(aes(colour = drv)) +
  scale_colour_discrete("Drive\ntrain")
```

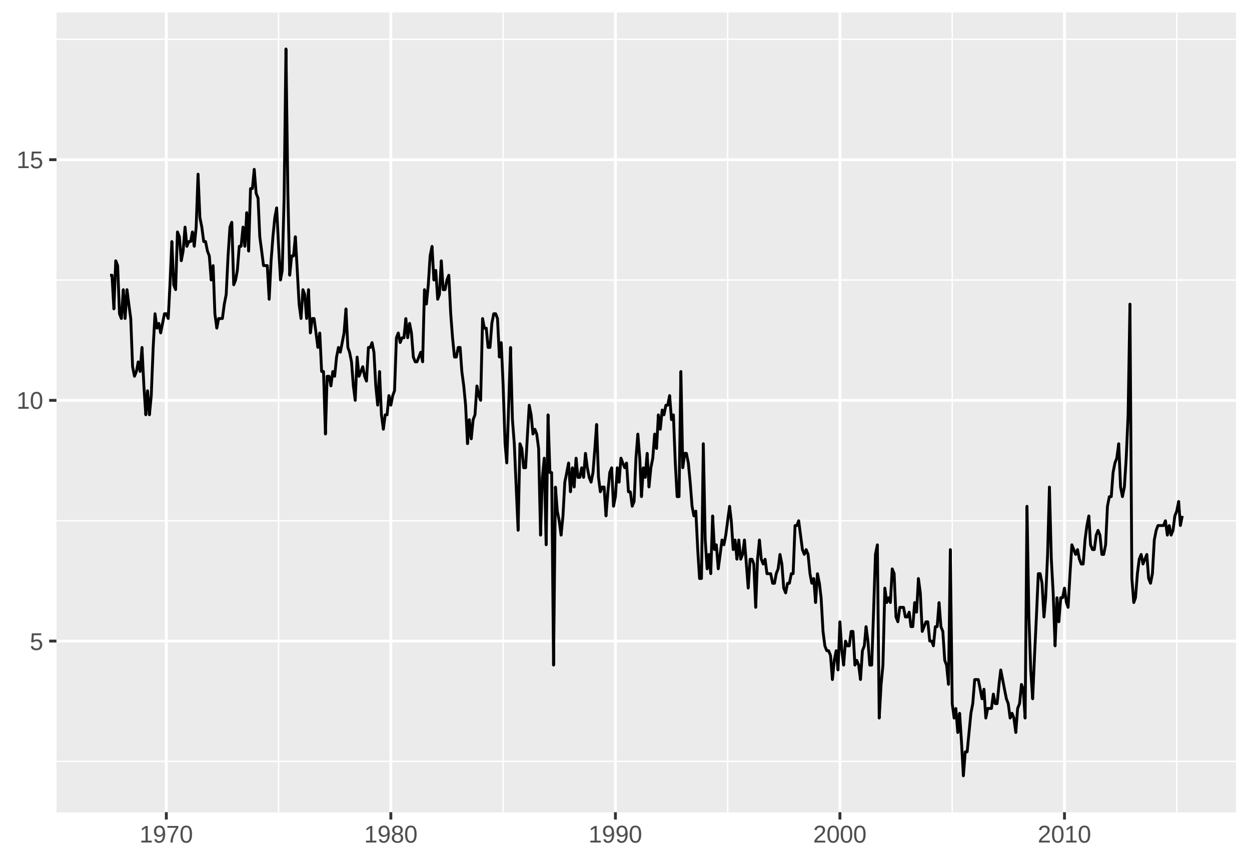
```
## Scale for x is already present.
## Adding another scale for x, which will replace the existing scale.
## Scale for colour is already present.
## Adding another scale for colour, which will replace the existing scale.
```



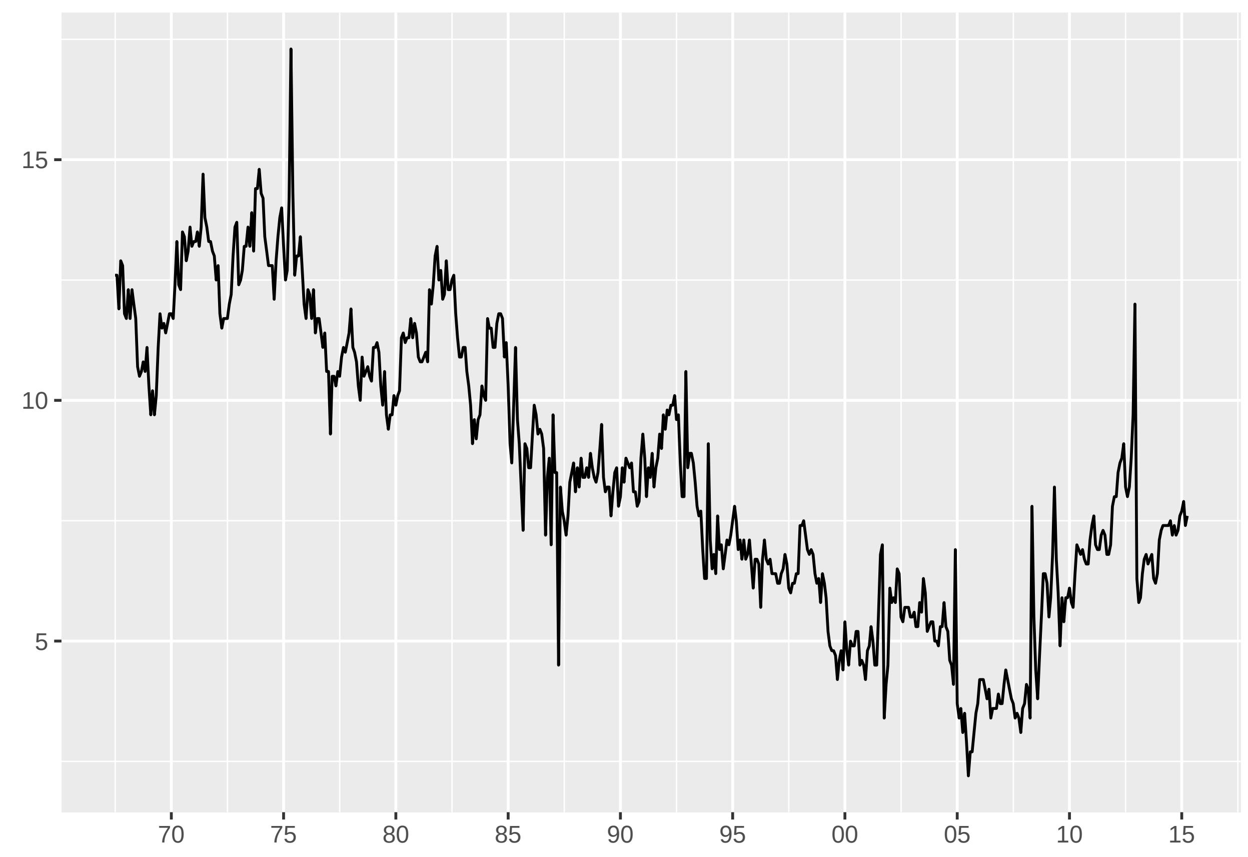
```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = drv, shape = drv)) +
  scale_colour_discrete("Drive train")
```



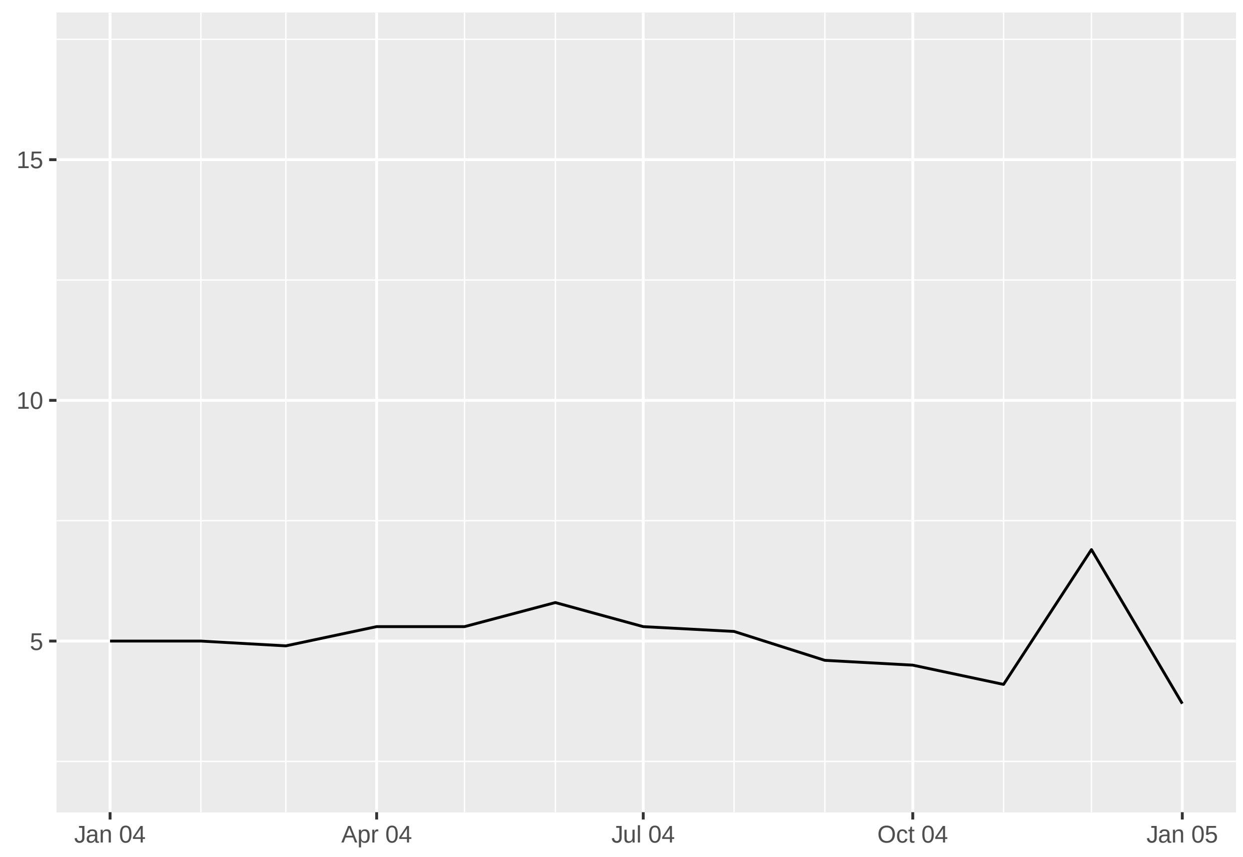
```r
base <- ggplot(economics, aes(date, psavert)) +
  geom_line(na.rm = TRUE) +
  labs(x = NULL, y = NULL)
base # Default breaks and labels
```



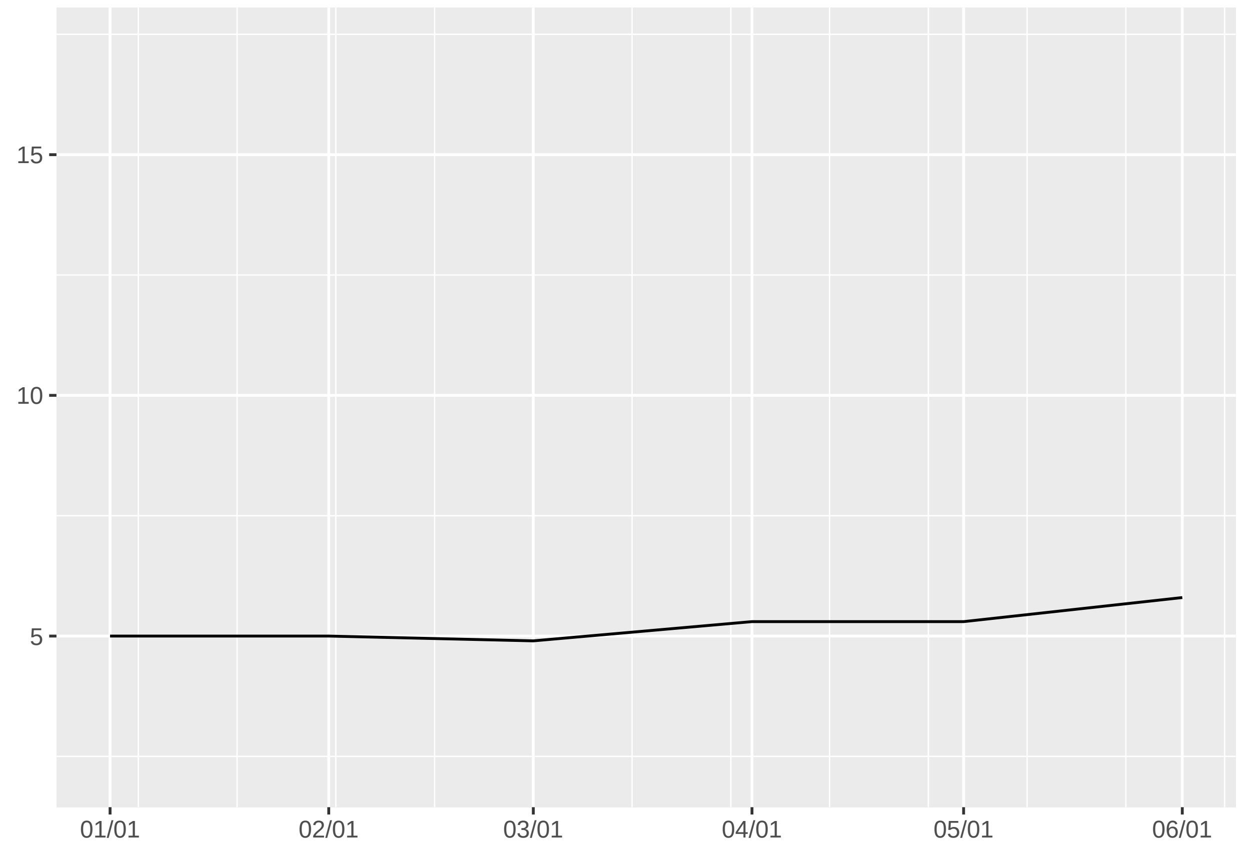
```r
base + scale_x_date(date_labels = "%y", date_breaks = "5 years")
```



```
base + scale_x_date(
limits = as.Date(c("2004-01-01", "2005-01-01")),
date_labels = "%b %y",
date_minor_breaks = "1 month"
)
```



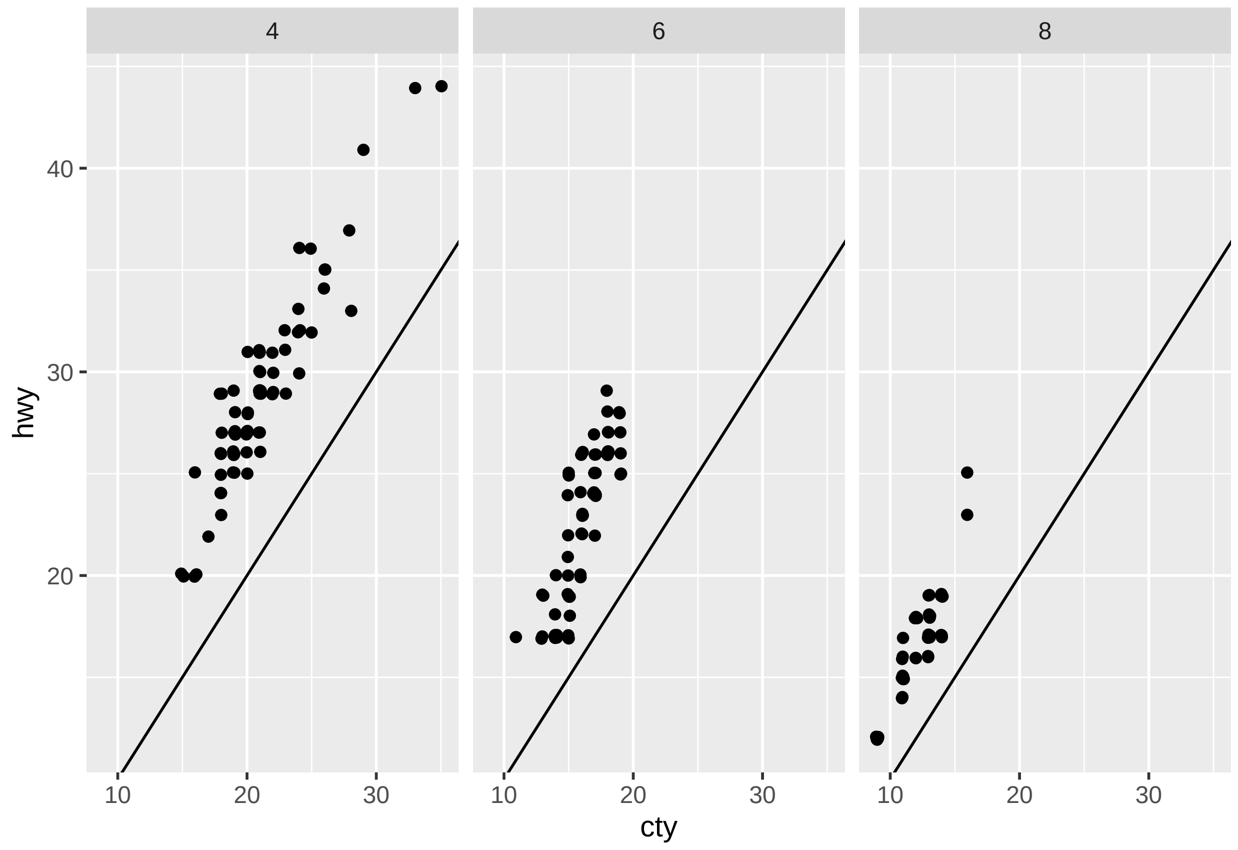
```
base + scale_x_date(
limits = as.Date(c("2004-01-01", "2004-06-01")),
date_labels = "%m/%d",
date_minor_breaks = "2 weeks"
)
```



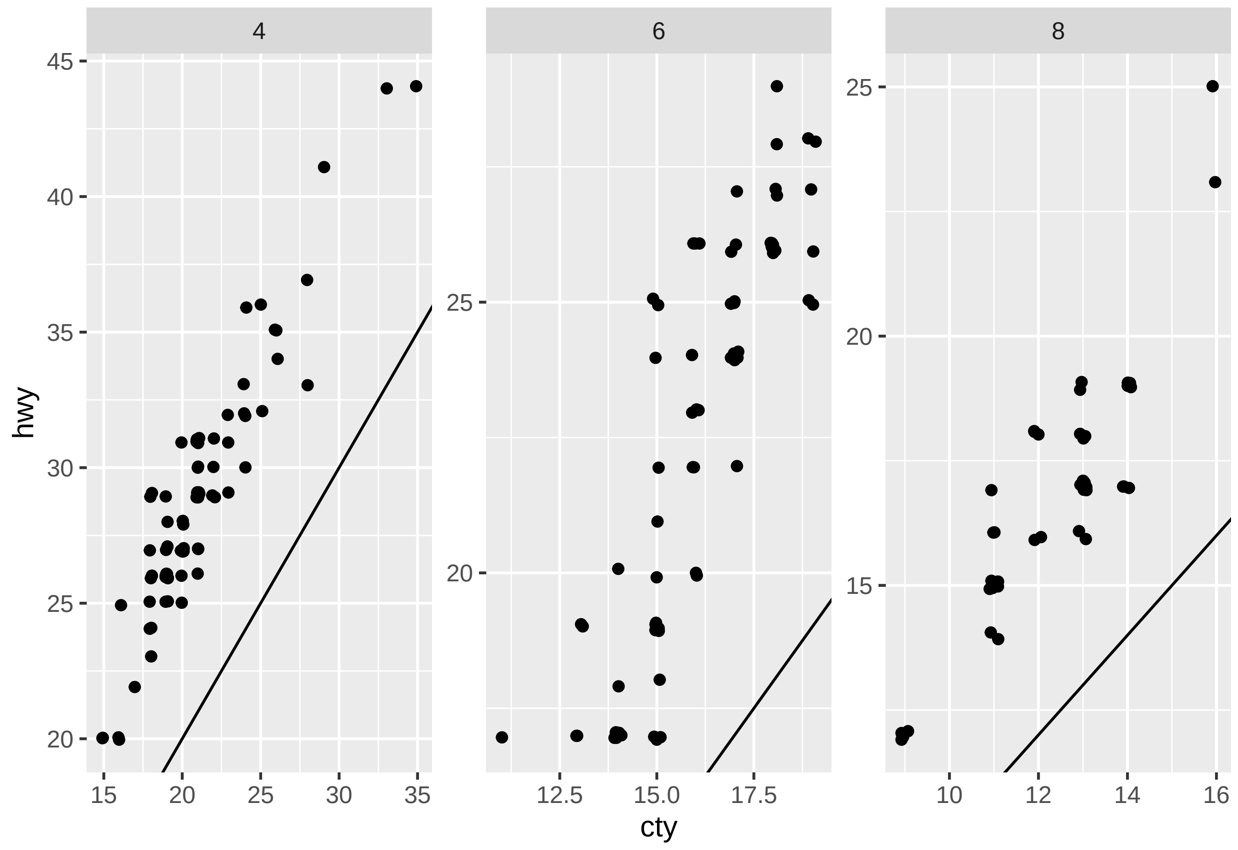
```r
mpg2 <- subset(mpg, cyl != 5 & drv %in% c("4", "f") & class != "2seater")
```

Fixed scales make it easier to see patterns across panels; free scales make it easier to see patterns within panels.

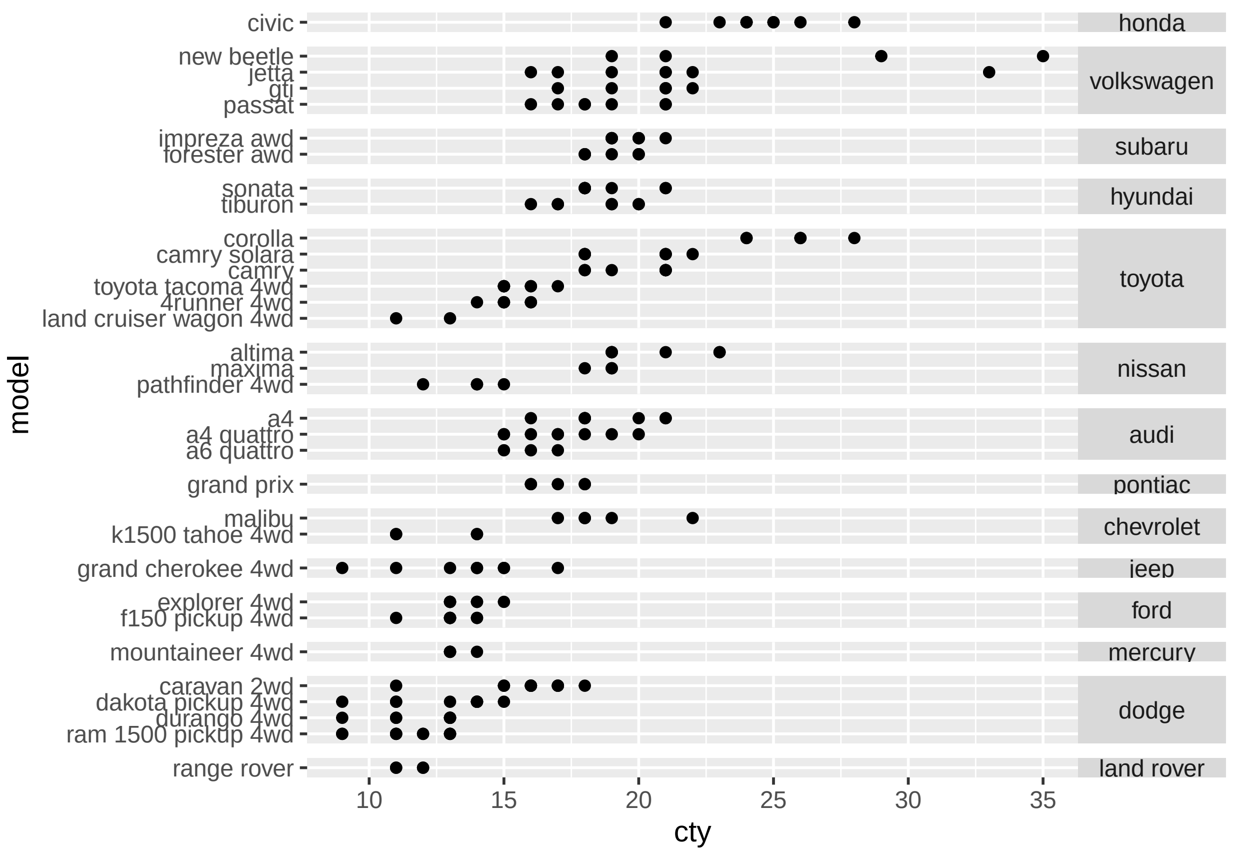
```r
p <- ggplot(mpg2, aes(cty, hwy)) +
geom_abline() +
geom_jitter(width = 0.1, height = 0.1)
p + facet_wrap(~cyl)
```



```
p + facet_wrap(~cyl, scales = "free")
```



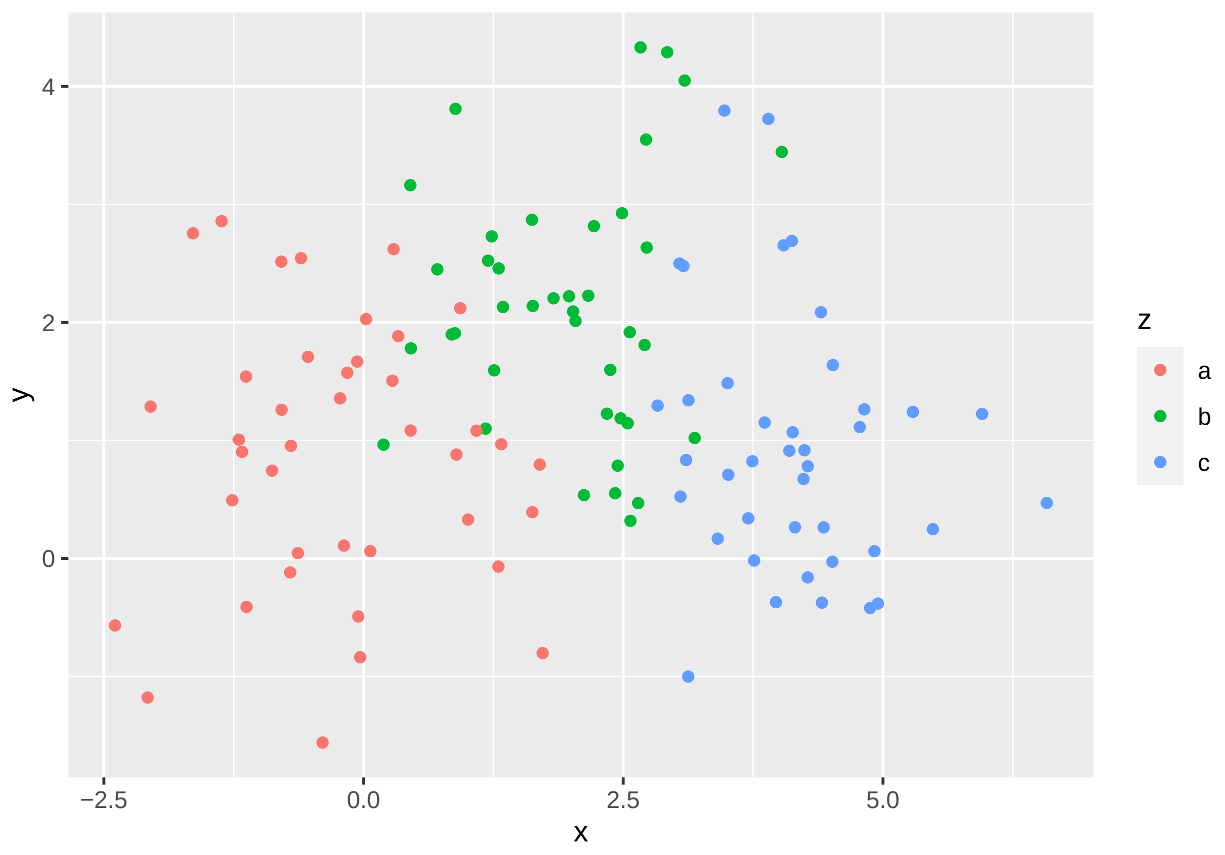
```
mpg2$model <- reorder(mpg2$model, mpg2$cty)
mpg2$manufacturer <- reorder(mpg2$manufacturer, -mpg2$cty)
ggplot(mpg2, aes(cty, model)) +
  geom_point() +
  facet_grid(manufacturer ~ ., scales = "free", space = "free") +
  theme(strip.text.y = element_text(angle = 0))
```



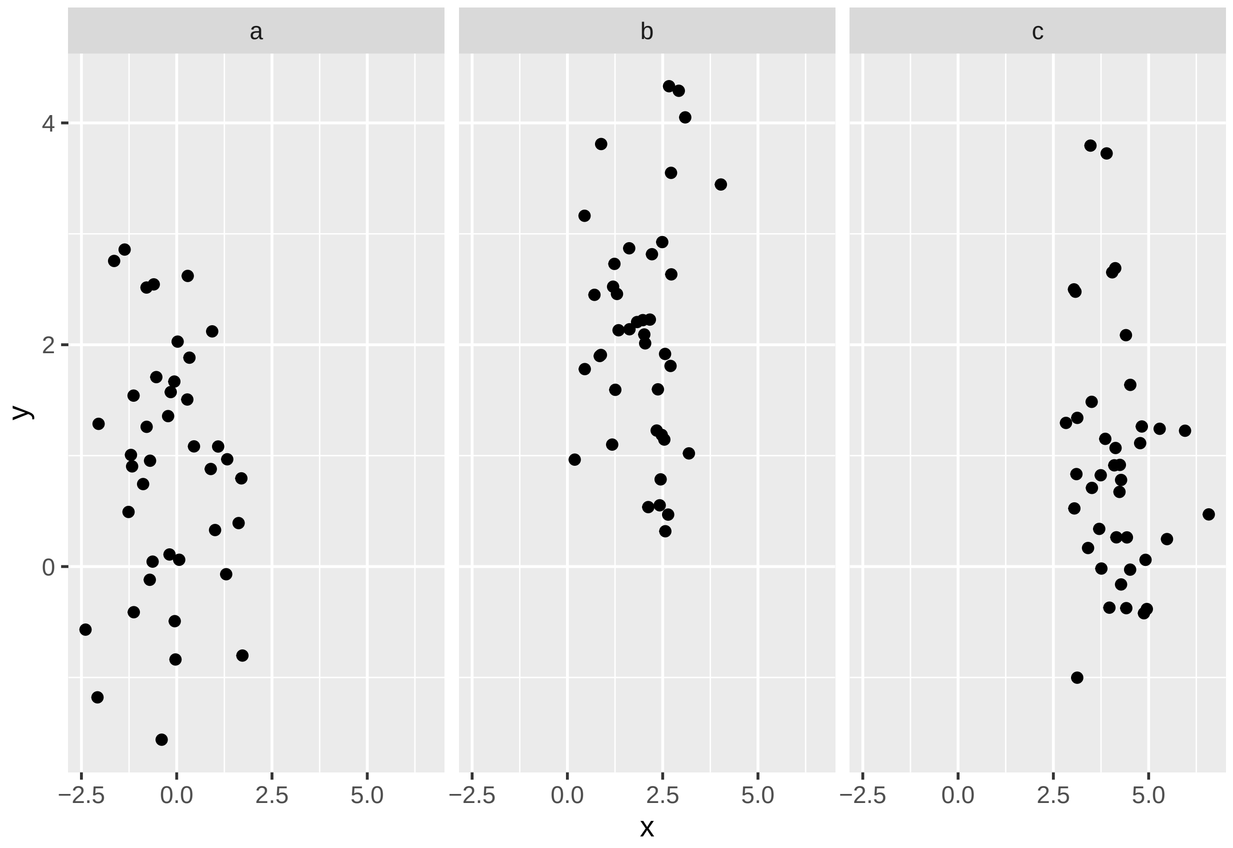
Grouping vs. facetting

Facetting is an alternative to using aesthetics (like colour, shape or size) to differentiate groups. Both techniques have strengths and weaknesses, based around the relative positions of the subsets. With facetting, each group is quite far apart in its own panel, and there is no overlap between the groups. This is good if the groups overlap a lot, but it does make small differences harder to see. When using aesthetics to differentiate groups, the groups are close together and may overlap, but small differences are easier to see.

```
df <- data.frame(
x = rnorm(120, c(0, 2, 4)),
y = rnorm(120, c(1, 2, 1)),
z = letters[1:3]
)
ggplot(df, aes(x, y)) +
  geom_point(aes(colour = z))
```

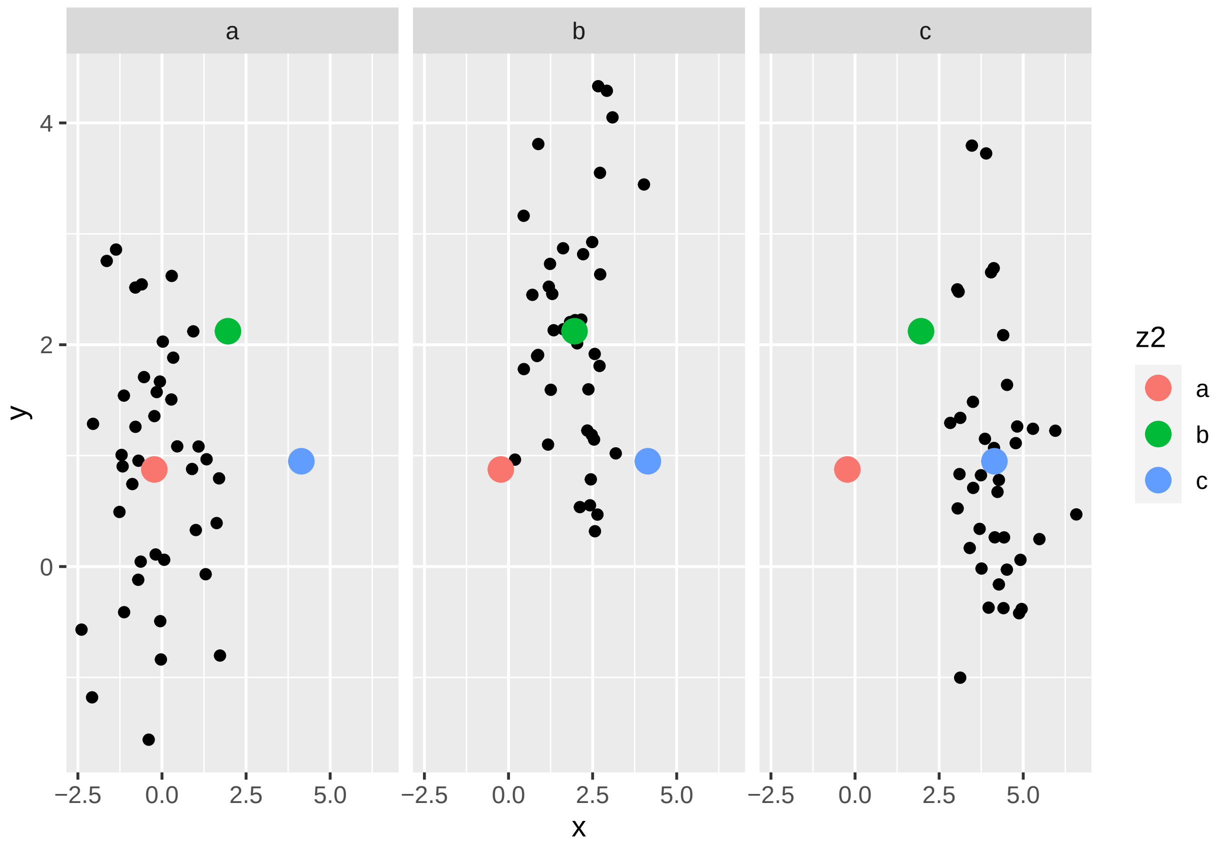


```
ggplot(df, aes(x, y)) +
  geom_point() +
  facet_wrap(~z)
```



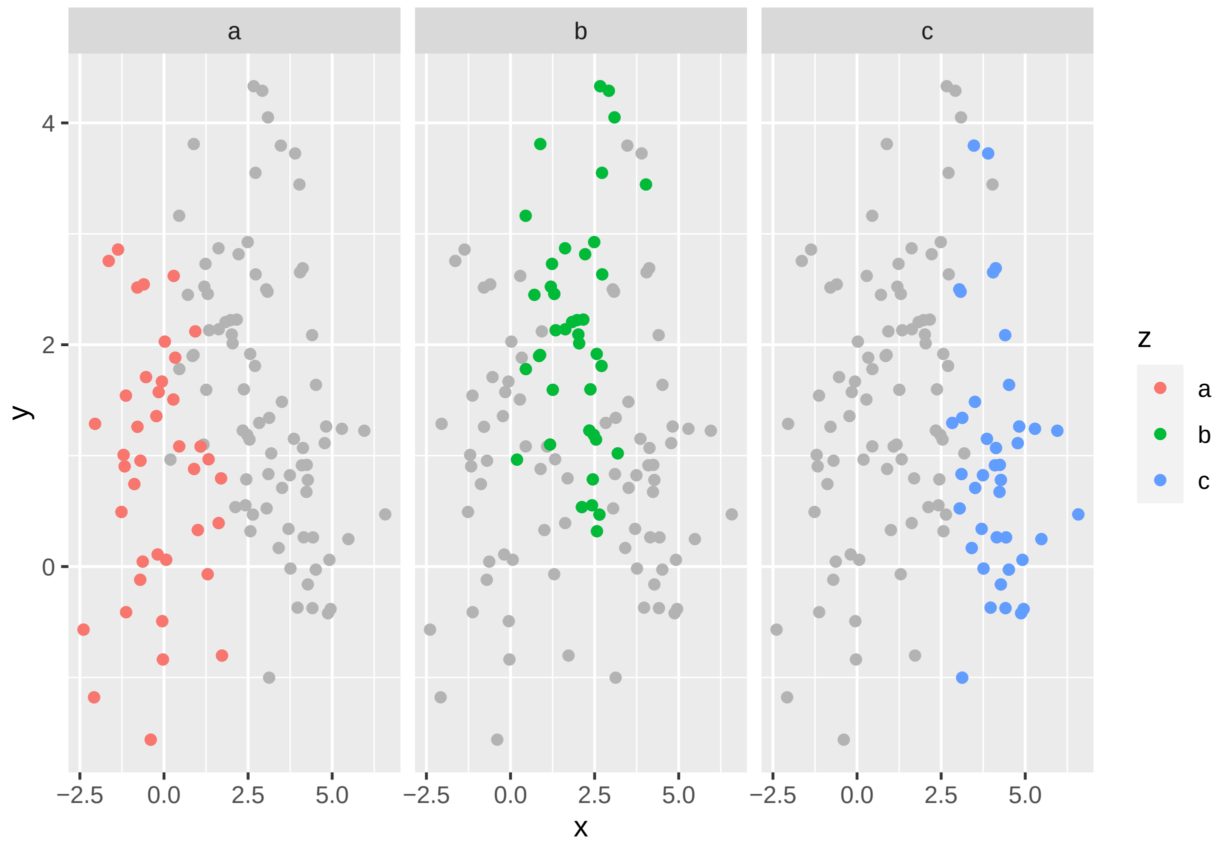
Comparisons between facets often benefit from some thoughtful annotation. For example, in this case we could show the mean of each group in every panel. Note that we need two “z” variables: one for the facets and one for the colours.

```
df_sum <- df %>%
  group_by(z) %>%
  summarise(x = mean(x), y = mean(y)) %>%
  rename(z2 = z)
ggplot(df, aes(x, y)) +
  geom_point() +
  geom_point(data = df_sum, aes(colour = z2), size = 4) +
  facet_wrap(~z)
```



Another useful technique is to put all the data in the background of each panel:

```
df2 <- dplyr::select(df, -z)
ggplot(df, aes(x, y)) +
  geom_point(data = df2, colour = "grey70") +
  geom_point(aes(colour = z)) +
  facet_wrap(~z)
```


a
b
c
y
x
4
2
0
−2.5
0.0
2.5
5.0
z
a
b
c