Topic 8: `ggplot2` Graphics System

ISOM3390: Business Programming in R

Base Graphics System in R

Base R contains functions that can be used to create scatter plots, boxplots, histograms, lines best approximating data, etc. It is robust and has served statisticians well.

We could feasibly do anything we want in base graphics, but…

- Awkward workflow for juxtaposition of many related plots.
- No built-in support for encoding additional information, especially categorical, via color or line type etc.

Complex graphs are time-consuming.

What is `ggplot2`?

The `ggplot2` graphics model is based on the **Grammar of Graphics**.

2 principles:

- Meaningful plots through aesthetic mapping (from data to aesthetic attributes (colour, shape, size) of geometric objects (points, lines, bars)).
- Graphics = distinct layers of grammatical elements.

It puts an organized framework to compose a graph with various independent components in a structured way.

Grammatical Elements of a Layer

Element	Description	Examples
Data	The data being plotted	
Aesthetics	Visual features onto which we map variables in the data	x position, y position, color, shape, size, linetype, pointtype, etc.
Geoms	Geometric objects used to draw each observation in the data	line, points, polygons, bars, etc.
Stats	Statistical transformations that summarise data in many useful ways	binning and counting observations to create a histogram, summarising a 2d relationship with a linear model, etc.
Positions	Position adjustments that deal with overlapping geometric objects	dodging objects side-to-side, jittering points, stacking objects on top of each another, etc.

The `mpg` Dataset

```
mpg  # fuel economy data from 1999 and 2008 for 38 popular models of cars.
```

```
## # A tibble: 234 x 11
##    manufacturer model displ  year   cyl trans drv     cty   hwy fl    cla~
##    <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <ch>
##  1 audi         a4      1.8  1999     4 auto~ f        18    29 p     com~
##  2 audi         a4      1.8  1999     4 manu~ f        21    29 p     com~
##  3 audi         a4      2    2008     4 manu~ f        20    31 p     com~
##  4 audi         a4      2    2008     4 auto~ f        21    30 p     com~
##  5 audi         a4      2.8  1999     6 auto~ f        16    26 p     com~
##  6 audi         a4      2.8  1999     6 manu~ f        18    26 p     com~
##  7 audi         a4      3.1  2008     6 auto~ f        18    27 p     com~
##  8 audi         a4 q~   1.8  1999     4 manu~ 4        18    26 p     com~
##  9 audi         a4 q~   1.8  1999     4 auto~ 4        16    25 p     com~
## 10 audi         a4 q~   2    2008     4 manu~ 4        20    28 p     com~
## # ... with 224 more rows
```

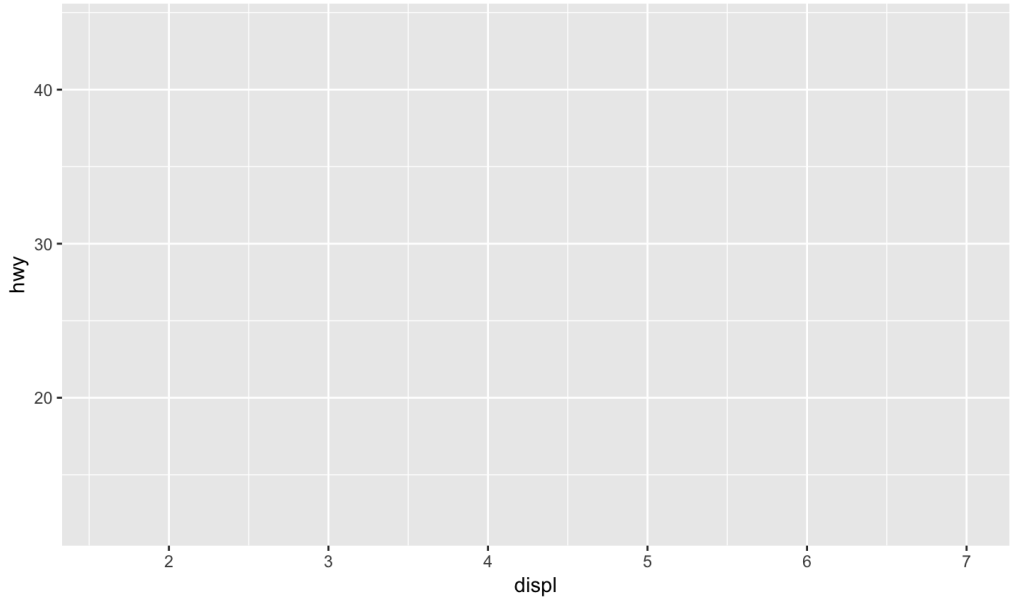
```
glimpse(mpg)
```

```
## Observations: 234
## Variables: 11
## $ manufacturer <chr> "audi", "audi", "audi", "audi", "audi", "audi", "...
## $ model        <chr> "a4", "a4", "a4", "a4", "a4", "a4", "a4", "a4 qua...
## $ displ        <dbl> 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8, 1.8, 2.0,...
## $ year         <int> 1999, 1999, 2008, 2008, 1999, 1999, 2008, 1999, 1...
## $ cyl          <int> 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6...
## $ trans        <chr> "auto(l5)", "manual(m5)", "manual(m6)", "auto(av)...
## $ drv          <chr> "f", "f", "f", "f", "f", "f", "f", "4", "4", "4",...
## $ cty          <int> 18, 21, 20, 21, 16, 18, 18, 18, 16, 20, 19, 15, 1...
## $ hwy          <int> 29, 29, 31, 30, 26, 26, 27, 26, 25, 28, 27, 25, 2...
## $ fl           <chr> "p", "p", "p", "p", "p", "p", "p", "p", "p", "p",...
## $ class        <chr> "compact", "compact", "compact", "compact", "comp...
```

The `ggplot()` Function

`ggplot()` takes data and **aesthetic mappings** and creates a new `ggplot` object.

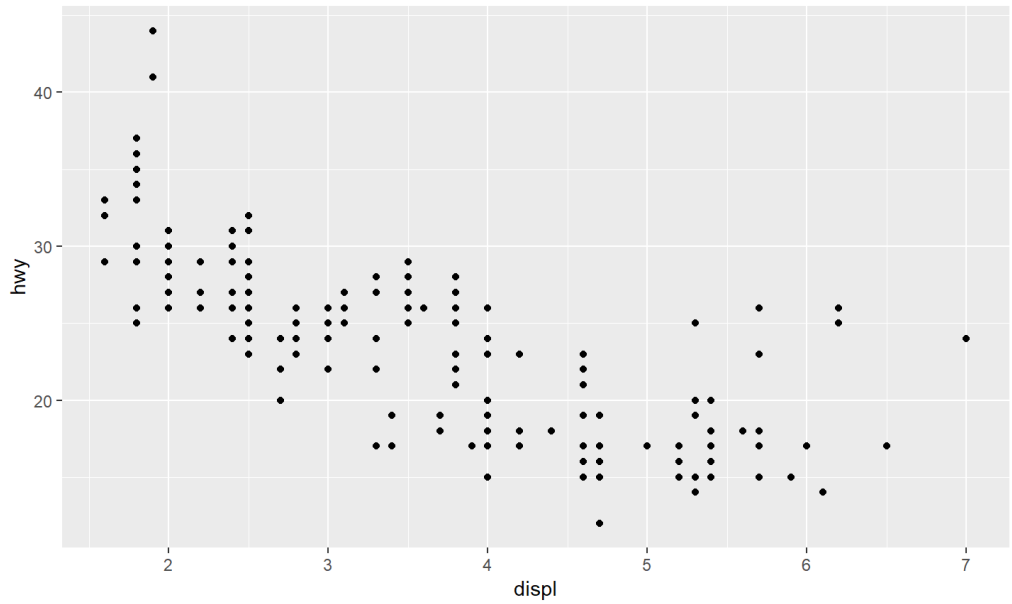
```
p <- ggplot(data = mpg, aes(x = displ, y = hwy))  # The `aes()` function defines the aesthetic mappings.
p  # There's nothing to see until a layer of geometric shapes is added.
```



`geom_()` Functions

Geoms perform the actual rendering of a layer (added on with +) and control the type of the plot to be created:

```
p + geom_point()
```

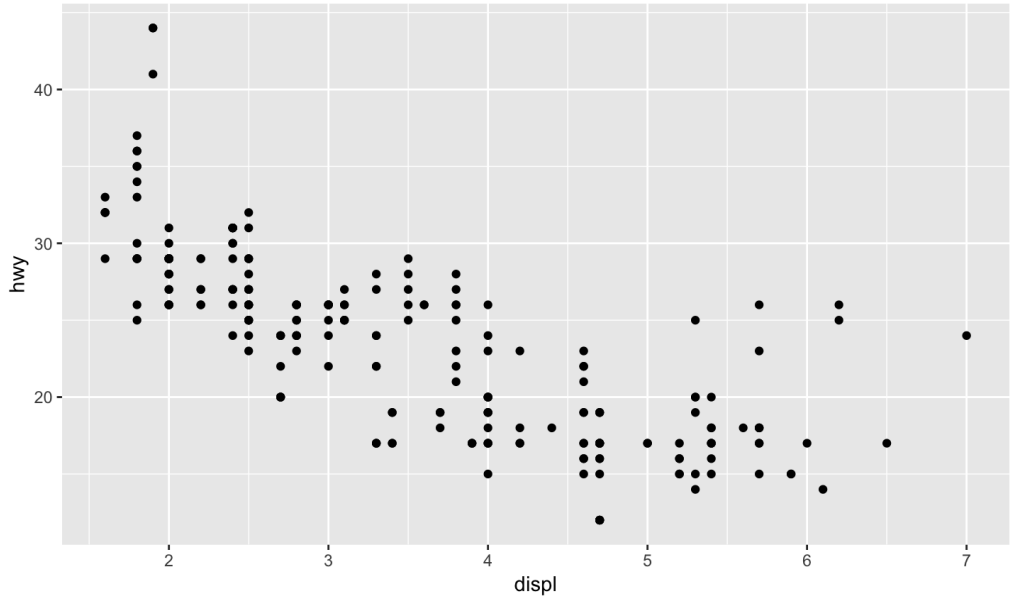


Important geoms include `geom_smooth()`, `geom_boxplot()`, `geom_bar()`, `geom_histogram()`, etc.

`geom_point()` is actually a shortcut.

Behind the scenes, it calls the `layer()` function fully specifying five components to create a new layer:

```
p + layer(mapping = NULL, data = NULL, geom = "point", stat = "identity", position = "identity")  # The default setting in ggplot() is used if NULL.
```

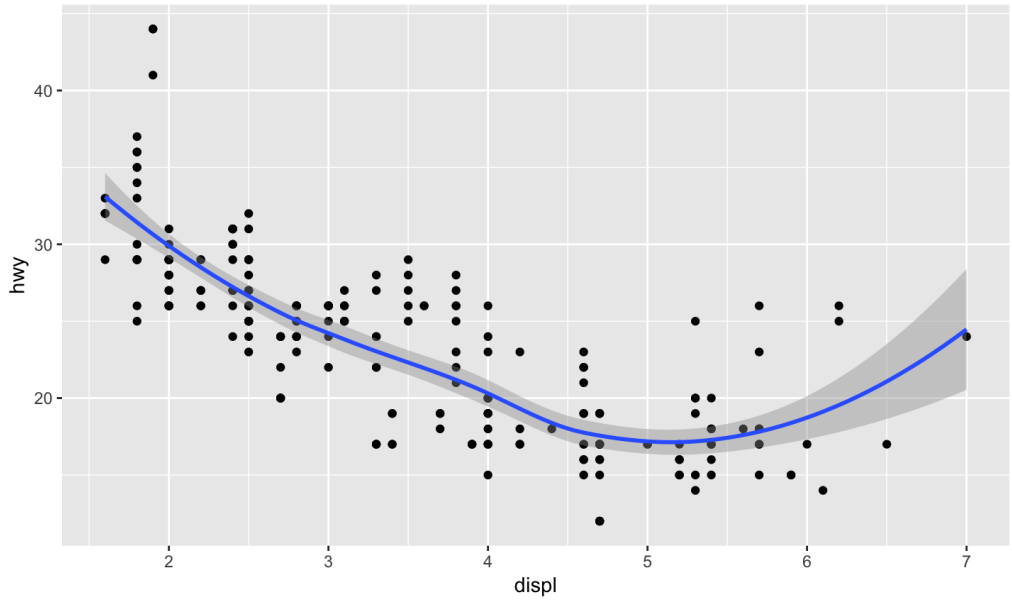



```
# 'identity' indicates no position adjustments.
```

Adding More Geoms

It could be useful to add a smoothed line to reveal the dominant pattern among the points:

```
p + geom_point() + geom_smooth()  # method = 'loess' and formula 'y ~ x' are used
```

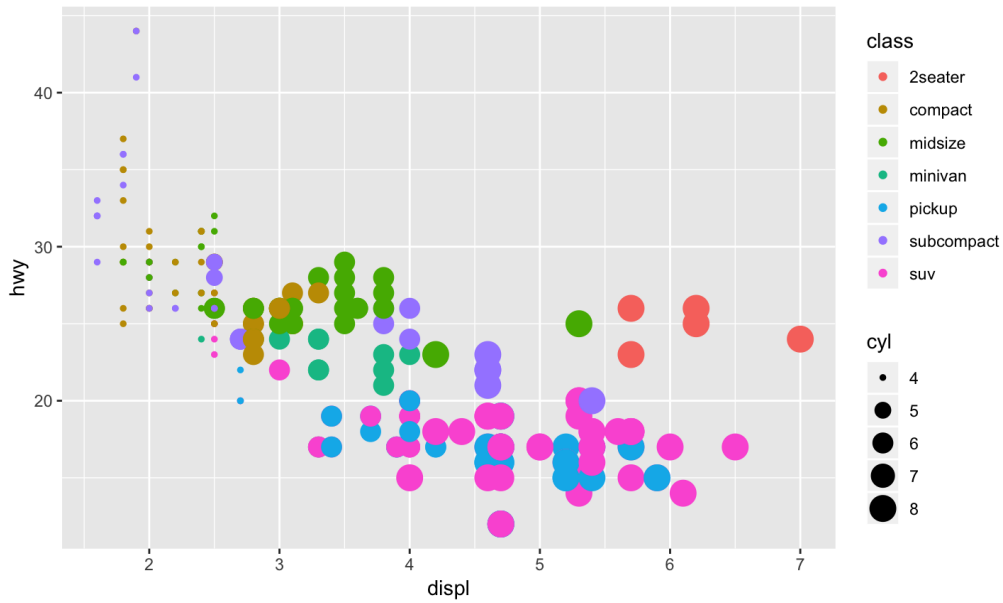


We can construct increasingly sophisticated plots by adding on more types of components.

Aesthetics

To represent additional variables in the plot, we can use other aesthetics like colour, shape, and size, which are added into the call to `aes()`:

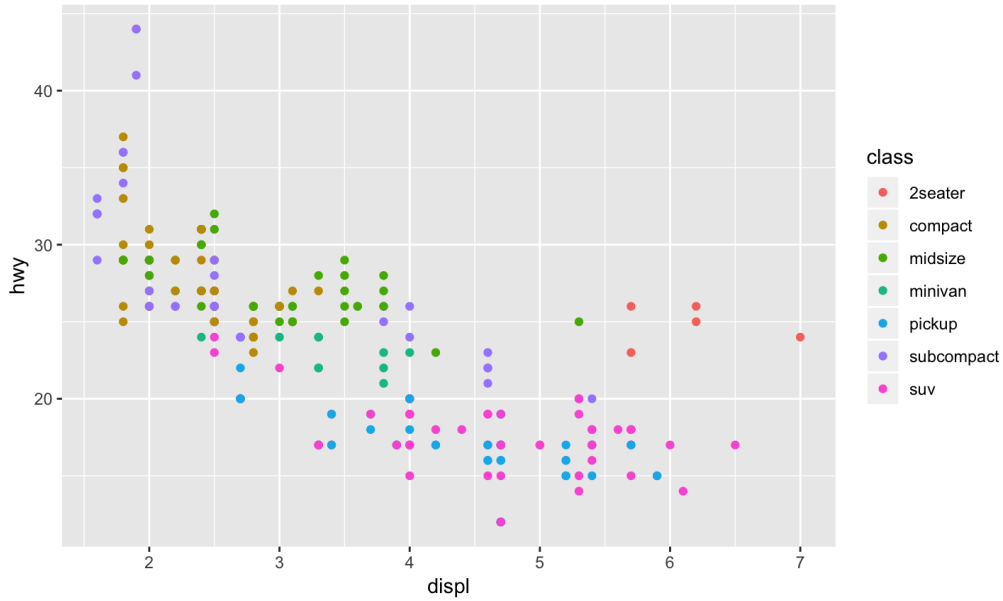
```
ggplot(mpg, aes(displ, hwy, colour = class, size = cyl)) + geom_point()
```



Note: Different types of aesthetic attributes work better with different types of variables.

Aesthetic mappings can be supplied in the initial `ggplot()` call, in individual layers, or in some combination of both.

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



Specifying Aesthetics in Layers

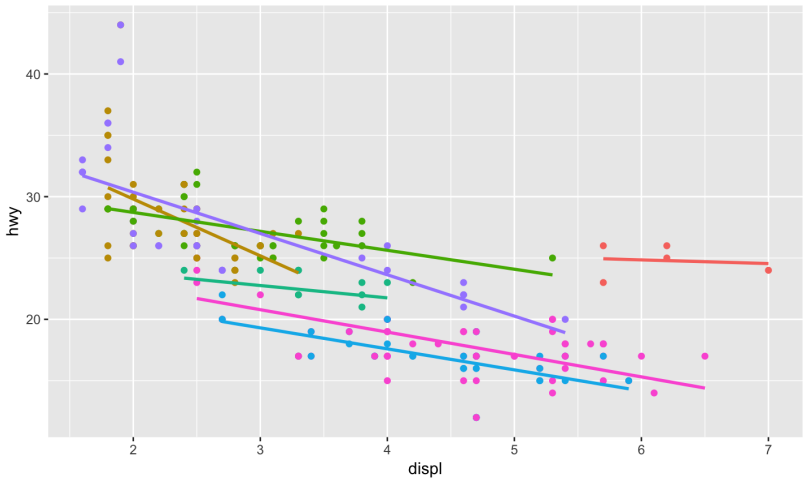
```
ggplot(mpg, aes(displ, hwy))
```

Operations	Layer Aeasthetics	Results
Add	`aes(colour = class)`	`aes(displ, hwy, colour = class)`
Override	`aes(x = hwy, y = disp)`	`aes(hwy, disp)`
Remove	`aes(y = NULL)`	`aes(displ)`

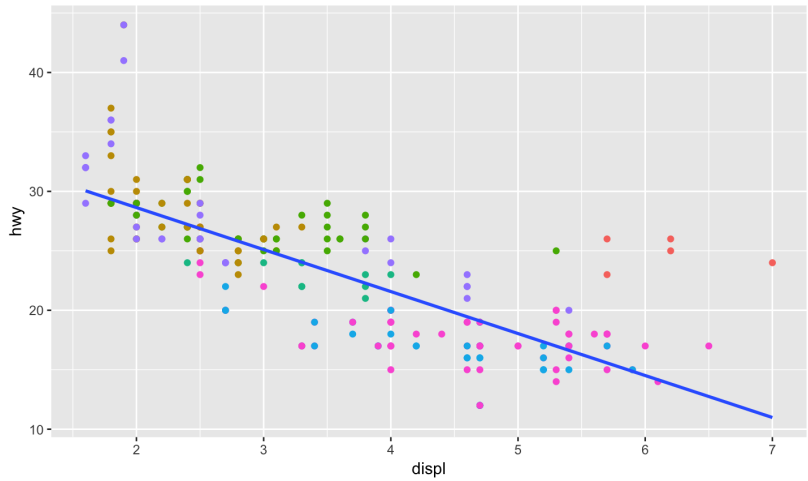
In the Plot vs. In the Layers

The scope of the aesthetic mapping for `colour` is different:

```
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  theme(legend.position = "none")
```




```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class)) +
  geom_smooth(method = "lm", se = FALSE) +
  theme(legend.position = "none")
```



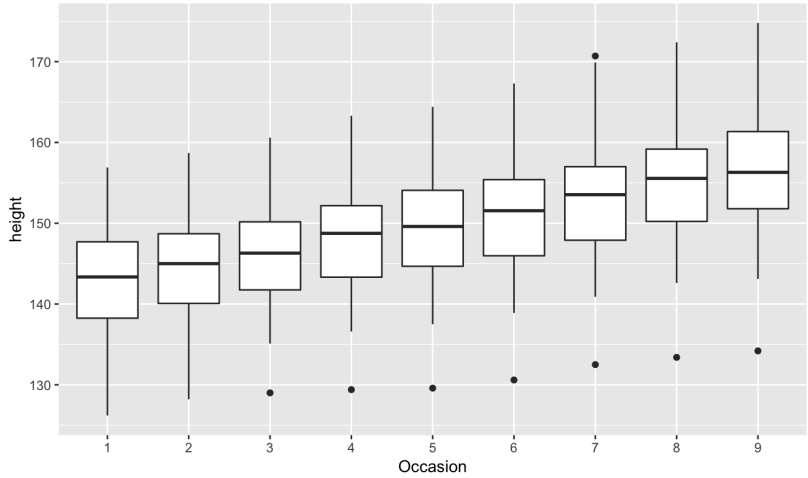
The `group` Aesthetic

By default, the group is set to the interaction of all discrete variables in the plot.

```
as_tibble(nlme::Oxboys)
```

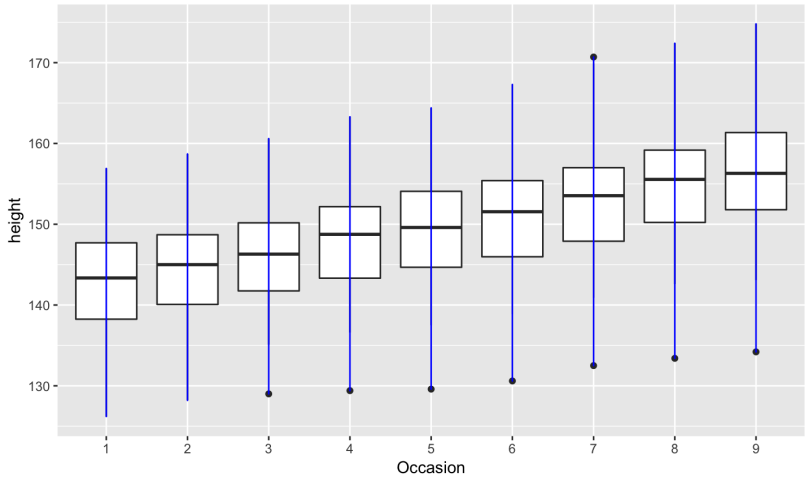
```
## # A tibble: 234 x 4
##    Subject     age height Occasion
##  * <ord>     <dbl>  <dbl> <ord>
##  1 1       -1        140. 1
##  2 1       -0.748    143. 2
##  3 1       -0.463    145. 3
##  4 1       -0.164    147. 4
##  5 1       -0.0027   148. 5
##  6 1        0.247    150. 6
##  7 1        0.556    152. 7
##  8 1        0.778    153. 8
##  9 1        0.994    156. 9
## 10 2       -1        137. 1
## # ... with 224 more rows
```

```
h <- ggplot(nlme::Oxboys, aes(Occasion, height))
h + geom_boxplot()
```

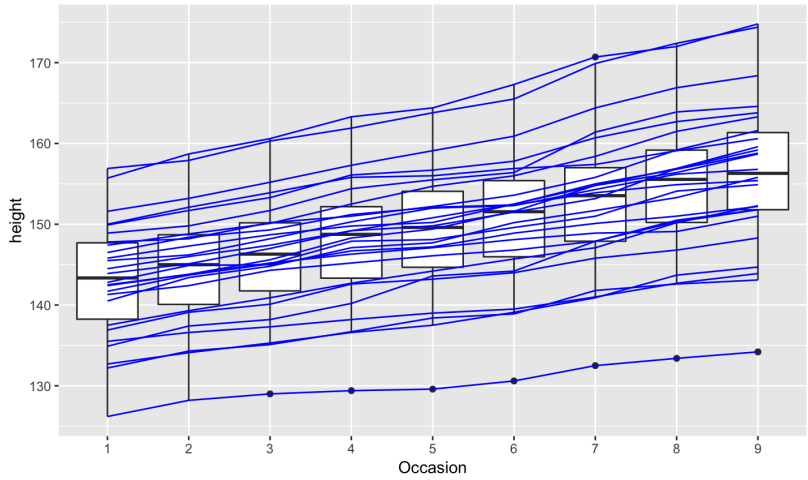


When the default way does not partition the data correctly, we need to explicitly define the grouping structure, using `group`.

```
h + geom_boxplot() + geom_line(colour = "blue")
```




```
h + geom_boxplot() + geom_line(aes(group = Subject), colour = "blue")
```

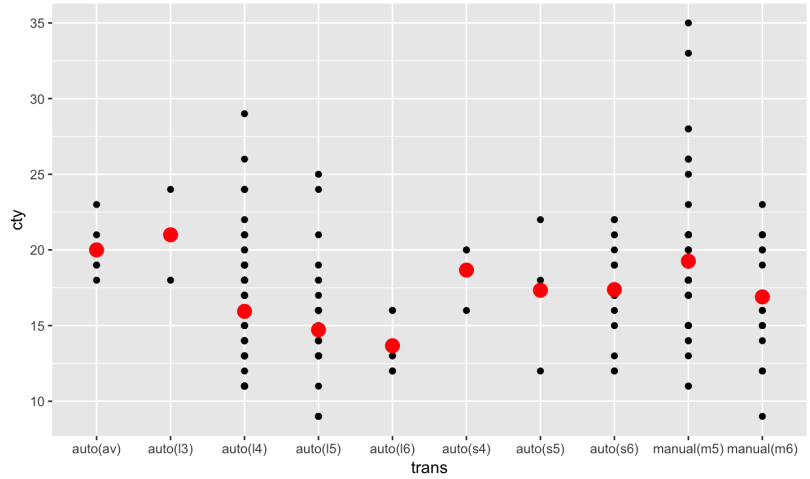


Stats

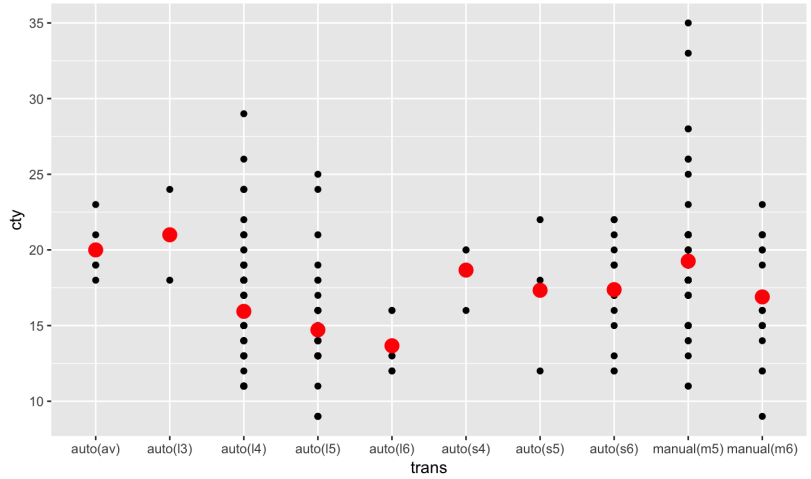
A statistical transformation, or **stat**, transforms the data to produce some quantities that are then mapped to aesthetics of a geom.

Two ways to add a layer for a **stat**:

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




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

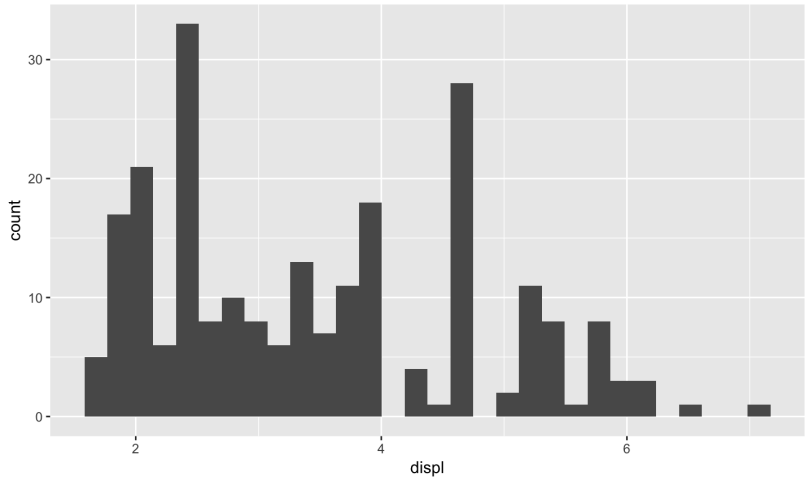


`geom_()` functions have default stats, while `stat_()` functions also have default geoms.

```
str(stat_bin)
```

```
## function (mapping = NULL, data = NULL, geom = "bar", position = "stack",
##     ..., binwidth = NULL, bins = NULL, center = NULL, boundary = NULL,
##     breaks = NULL, closed = c("right", "left"), pad = FALSE, na.rm = FALSE
##     show.legend = NA, inherit.aes = TRUE)
```

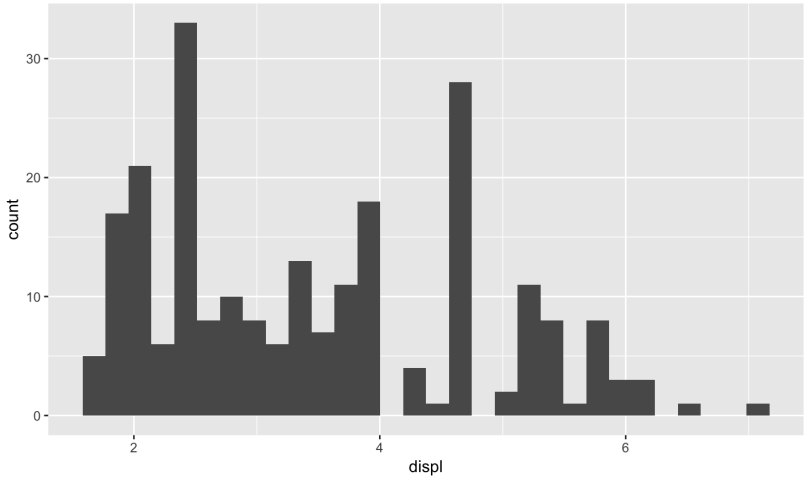
```
ggplot(mpg, aes(displ)) + stat_bin()
```




```
str(geom_histogram)
```

```
## function (mapping = NULL, data = NULL, stat = "bin", position = "stack",
##     ..., binwidth = NULL, bins = NULL, na.rm = FALSE, show.legend = NA,
##     inherit.aes = TRUE)
```

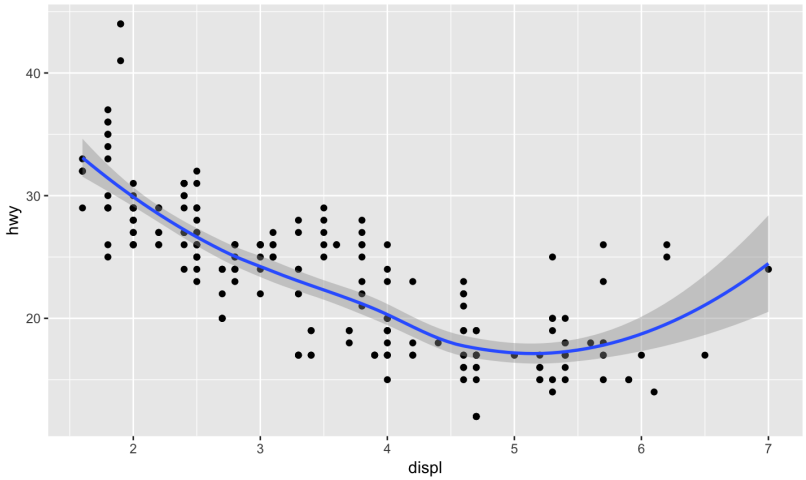
```
ggplot(mpg, aes(displ)) + geom_histogram()
```



```
str(stat_smooth)
```

```
## function (mapping = NULL, data = NULL, geom = "smooth", position = "identity",
##     ..., method = "auto", formula = y ~ x, se = TRUE, n = 80, span = 0.75,
##     fullrange = FALSE, level = 0.95, method.args = list(), na.rm = FALSE,
##     show.legend = NA, inherit.aes = TRUE)
```

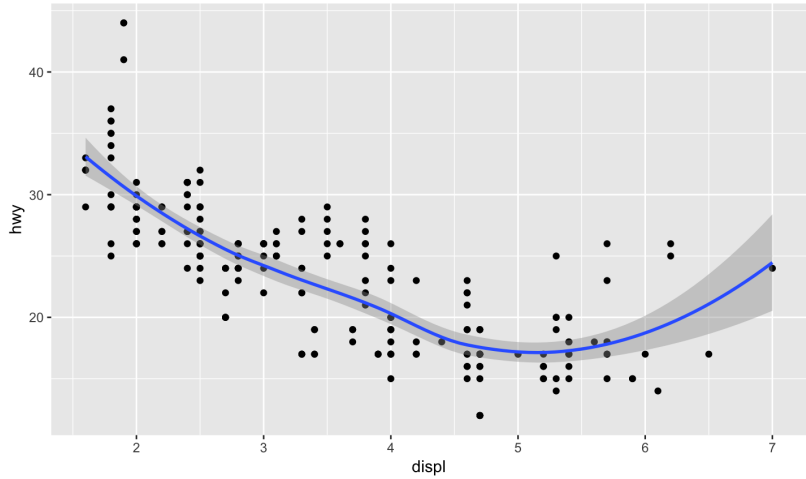
```
p + geom_point() + stat_smooth()
```



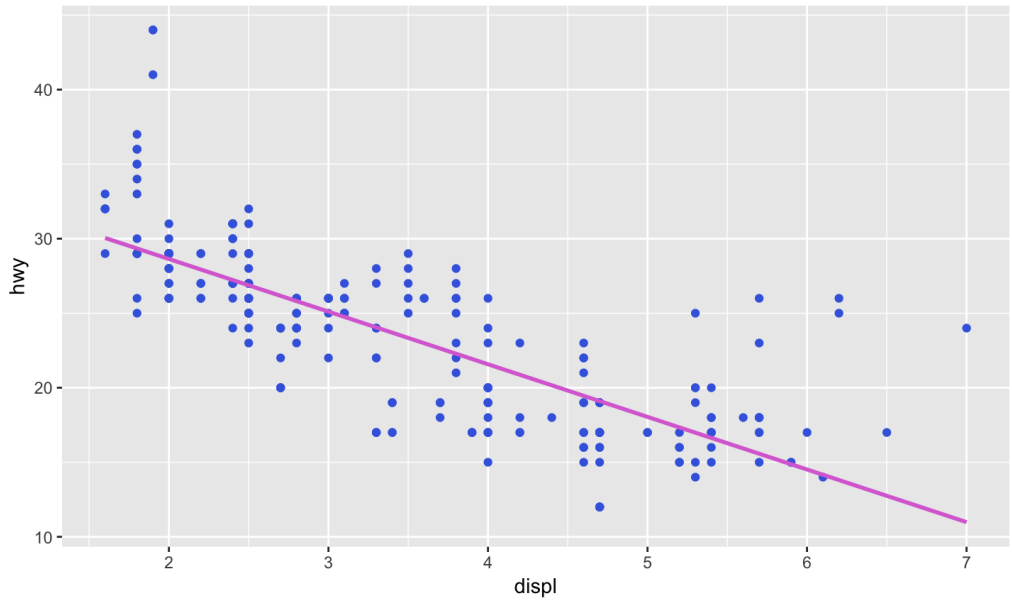

```
str(geom_smooth)
```

```
## function (mapping = NULL, data = NULL, stat = "smooth", position = "identity",
##     ..., method = "auto", formula = y ~ x, se = TRUE, na.rm = FALSE,
##     show.legend = NA, inherit.aes = TRUE)
```

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

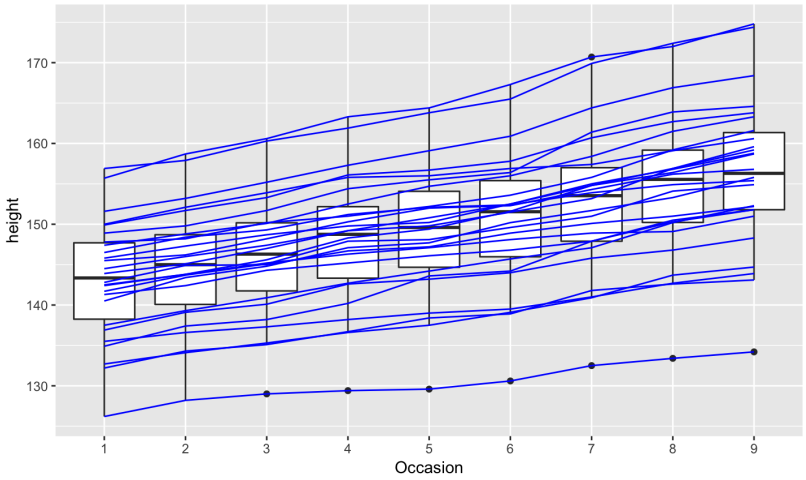


```
p + layer(geom = "point", stat = "identity", position = "identity", params = list(colour = "royalblue")) +
    layer(geom = "smooth", stat = "smooth", position = "identity", params = list(colour = "orchid",
        alpha = 0.5, method = "lm", se = FALSE))
```

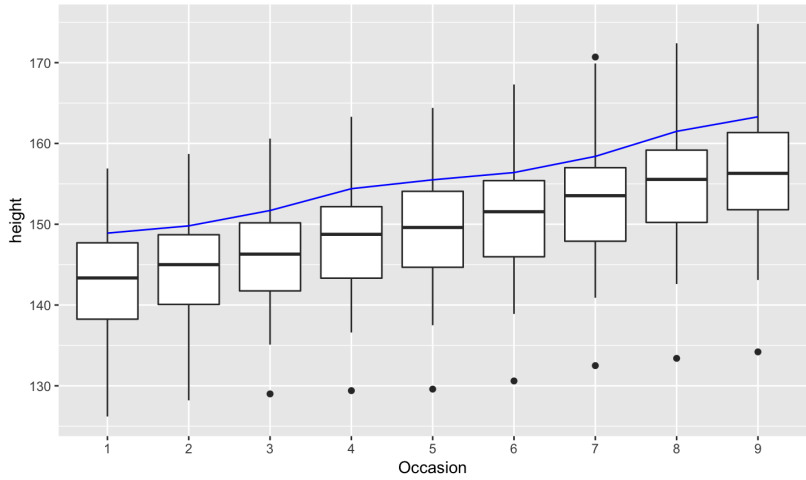


Additional arguments controlling the geom and the stat are given by `params`.

```
h + geom_boxplot() + stat_summary(aes(group = Subject),
                                  geom = "line", colour = "blue")
```




```
h + geom_boxplot() + stat_summary(aes(group = 1), geom = "line",
    fun.y = "quantile", fun.args = list(probs = 0.8), colour = "blue")
```

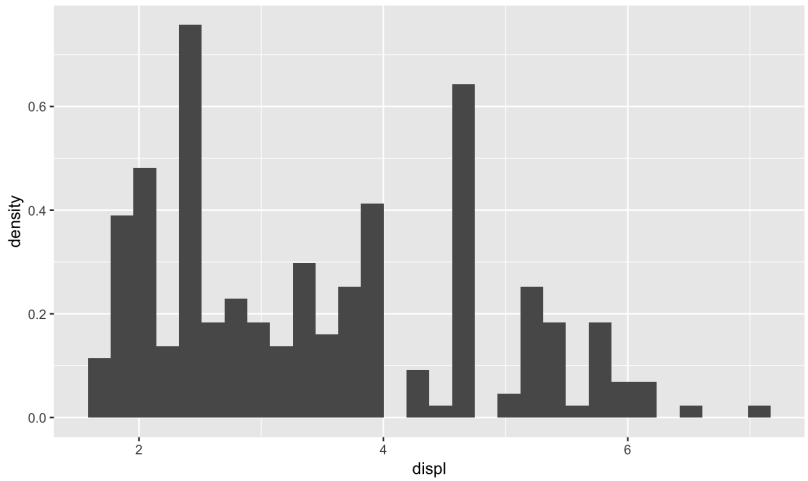


Changing Plotted Variables

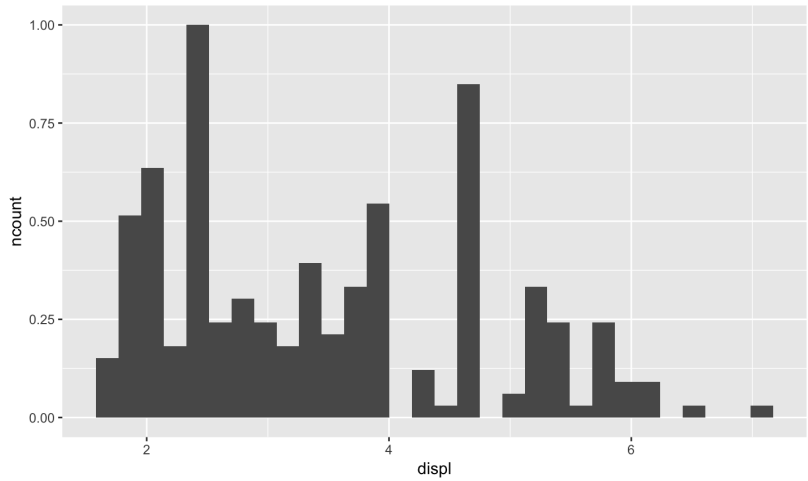
More than one variable can be generated by a **stat**.

Use `aes()` to change the default mapping with the variable name being surrounded by `..`:

```
ggplot(mpg, aes(displ)) + geom_histogram(aes(y = ..density..))
```



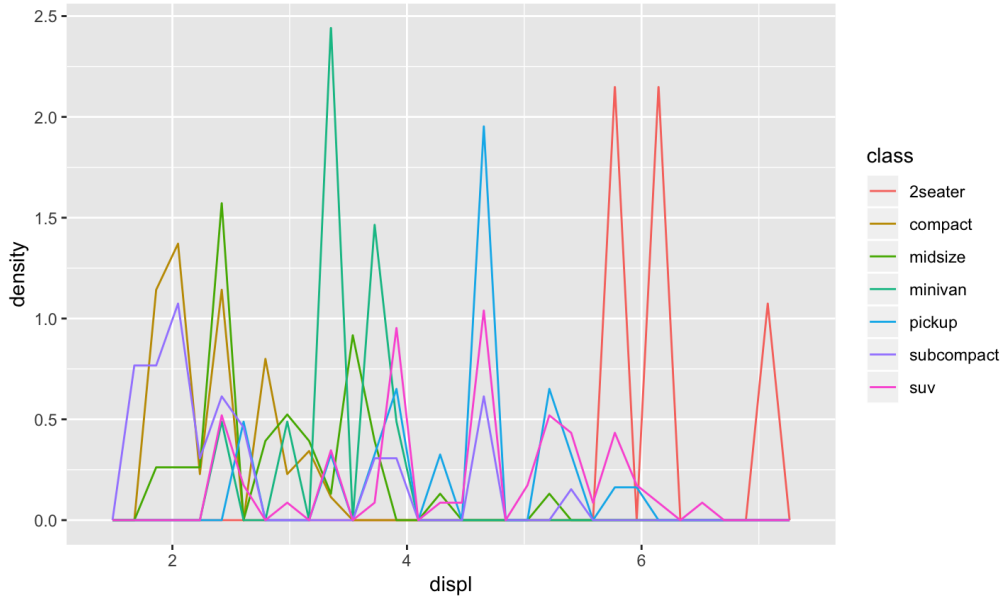

```
ggplot(mpg, aes(displ)) + geom_histogram(aes(y = ..ncount..))
```



```
str(geom_freqpoly)
```

```
## function (mapping = NULL, data = NULL, stat = "bin", position = "identity",
##     ..., na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

```
ggplot(mpg, aes(displ, colour = class)) + geom_freqpoly(aes(y = ..density..))
```

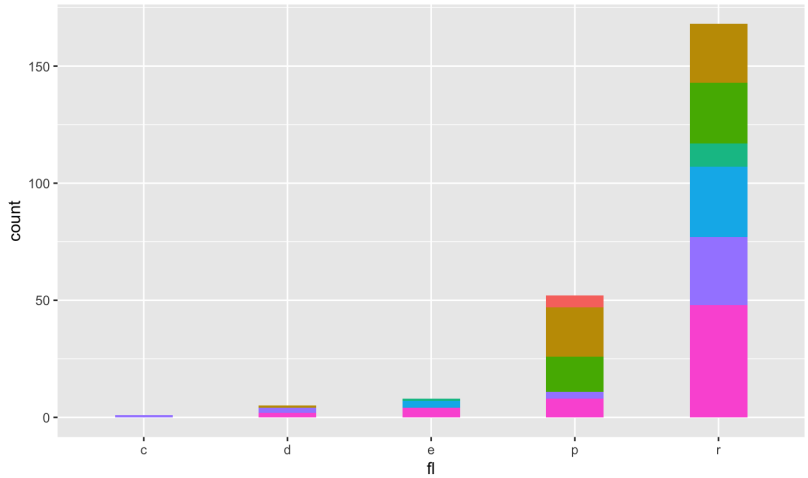


Position Adjustments

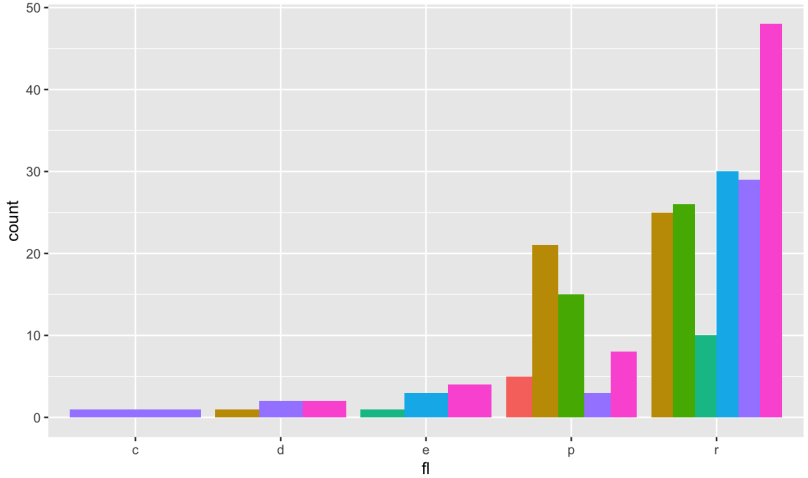
All layers have a position adjustment that resolves overlapping geoms within a layer.

Override the default by using the position argument to the `geom_()` or `stat_()` function.

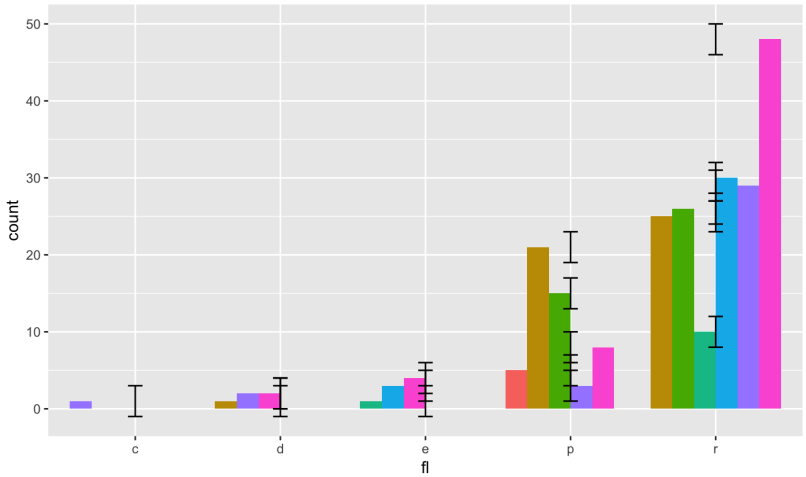
```
ggplot(mpg, aes(fl, fill = class)) +
  geom_bar(width = 0.4) +
  theme(legend.position = "none")
```



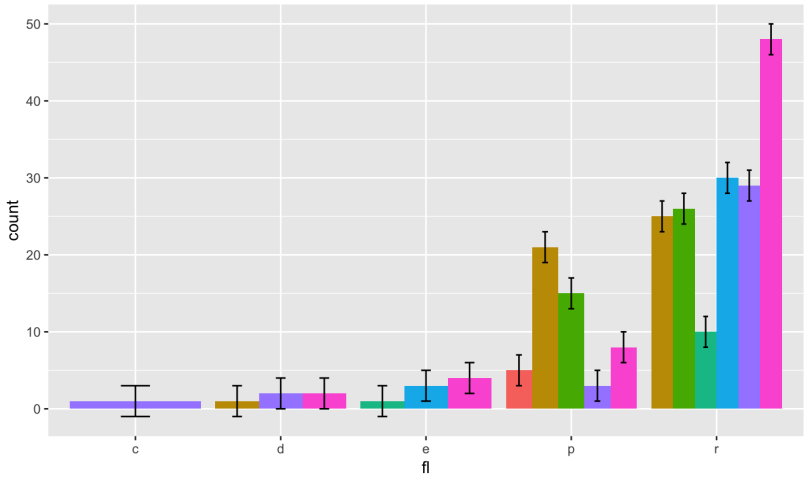

```
ggplot(mpg, aes(fl, fill = class)) +
  geom_bar(position = "dodge") +
  theme(legend.position = "none")
```



```
ggplot(mpg, aes(fl, fill = class)) +
  geom_bar(position = position_dodge(preserve = "single")) +
  geom_errorbar(aes(ymin = ..count.. - 2, ymax = ..count.. + 2),
                stat = "count", width = 0.1) +
  theme(legend.position = "none")
```




```
ggplot(mpg, aes(fl, fill = class)) + geom_bar(position = "dodge") +
  geom_errorbar(aes(ymin = ..count.. - 2, ymax = ..count.. + 2),
                stat = "count", width = 0.2,
                position = position_dodge(width = 0.9)) +
  theme(legend.position = "none")
```

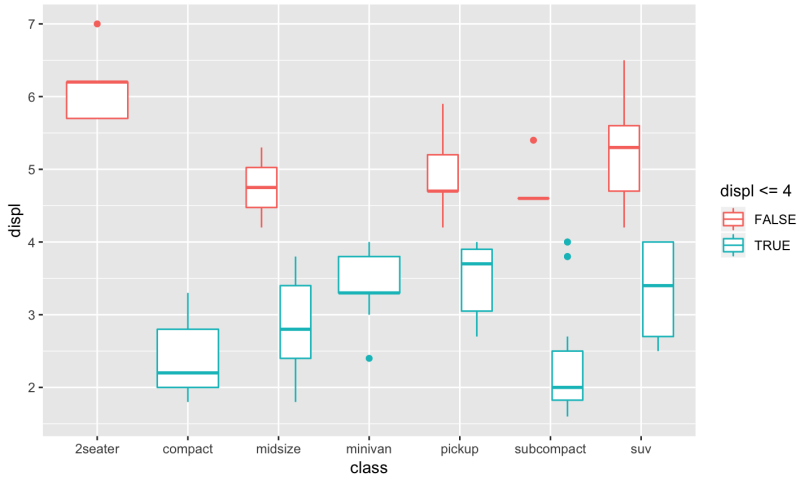


Position Functions

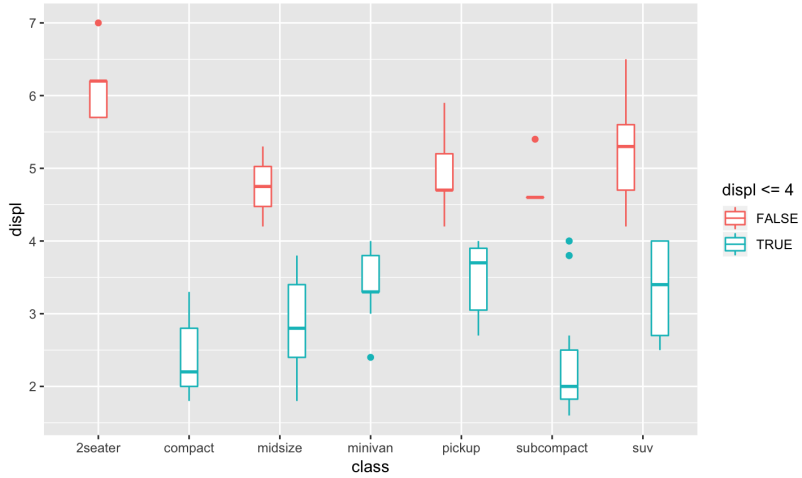
Functions prefixed with `position_` for position adjustments:

- The adjustments that apply primarily to bars:
  - `position_dodge()`, `position_dodge2()`: Dodge overlapping objects side-to-side
  - `position_stack()`, `position_fill()`: Stack overlapping objects on top of each another
- The primarily useful for points
  - `position_jitter()`: Jitter points to avoid overplotting
  - `position_jitterdodge()`: Simultaneously dodge and jitter
  - `position_nudge()`: Nudge points a fixed distance

```
ggplot(mpg, aes(class, displ)) +
  geom_boxplot(aes(colour = displ <= 4))
```




```
ggplot(mpg, aes(class, displ)) +
  geom_boxplot(aes(colour = displ <= 4), position =
                 position_dodge2(preserve = "single", padding = 0.5))
```



Other Grammatical Elements

Element	Description
Scales	Specify how values in the data space should be mapped to values in the aesthetic space
Facets	Define how to break up the data into subsets and display those subsets as small multiples
Coords	Describe how data coordinates are mapped to the plane of the graphic.
Themes	Control all non-data elements of a plot

Scales

Scales control the mapping from data to aesthetics and provide the tools (i.e., the axes and legends) that let us read the plot:

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

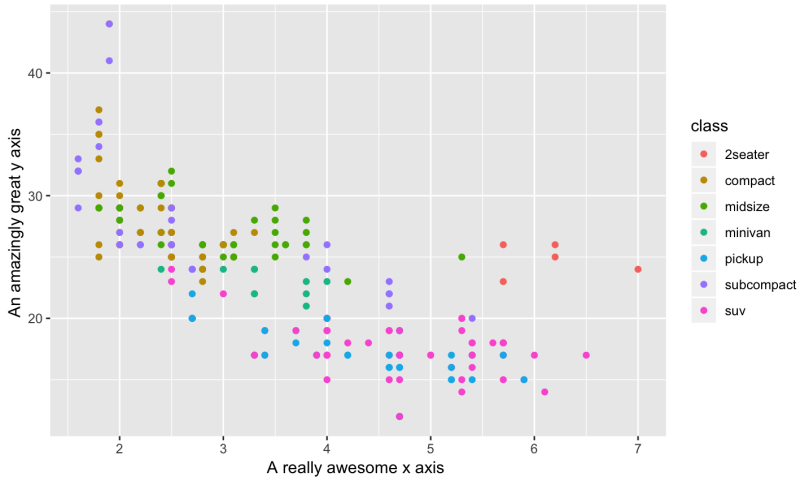
Behind the scene, it is:

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

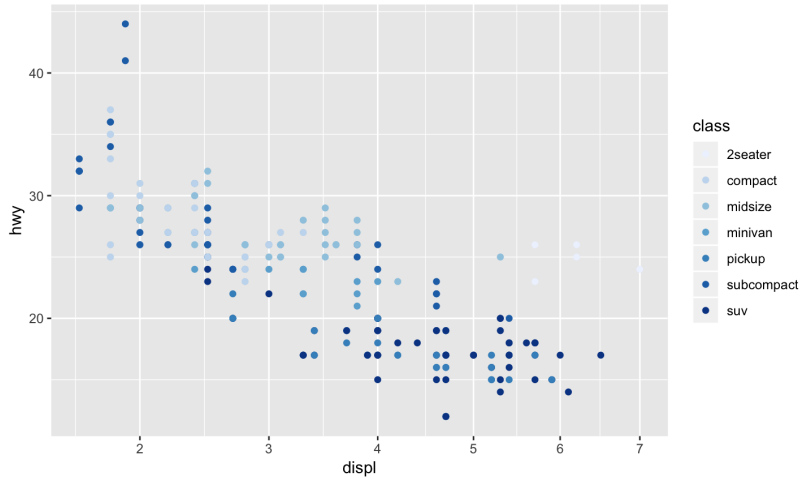
The naming convention for scales: a combination of the name of the aesthetic and the name of the scale, e.g., `scale_y_continuous()`, `scale_colour_hue()`, etc.

Explicitly call a `scale_()` function to override the defaults:

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



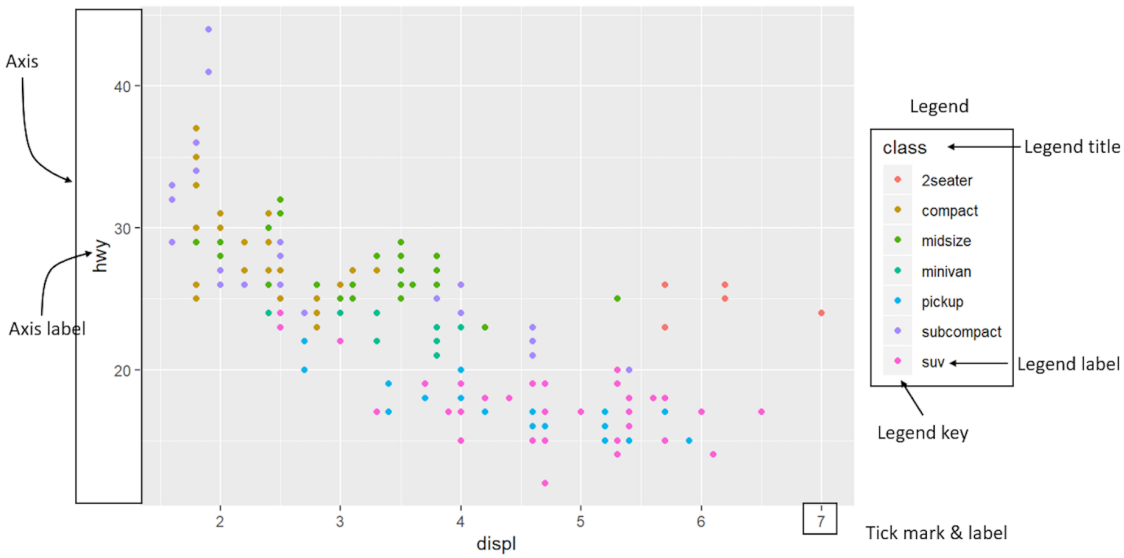

```
ggplot(mpg, aes(displ, hwy)) + geom_point(aes(colour = class)) +
  scale_x_sqrt() +
  scale_colour_brewer()
```



Guides: Legends and Axes

The axis or legend associated with a scale, collectively referred to as the **guide**, is the component of the scale that we're most likely to modify.

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

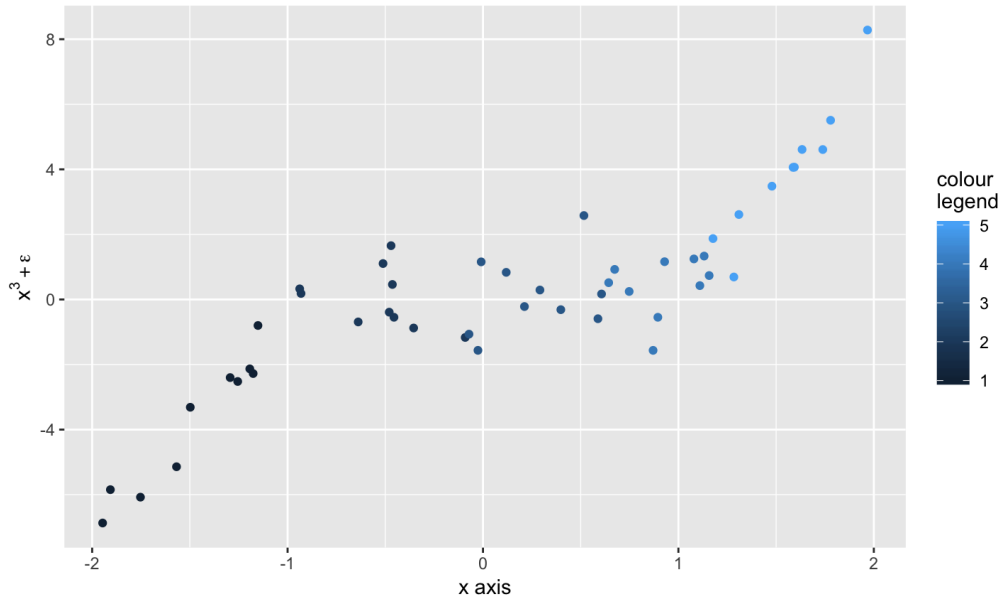


Note: Axes calibrate the scales of the position aesthetics, while legends calibrate the scales of non-position aesthetics.

Scale Title: `name`

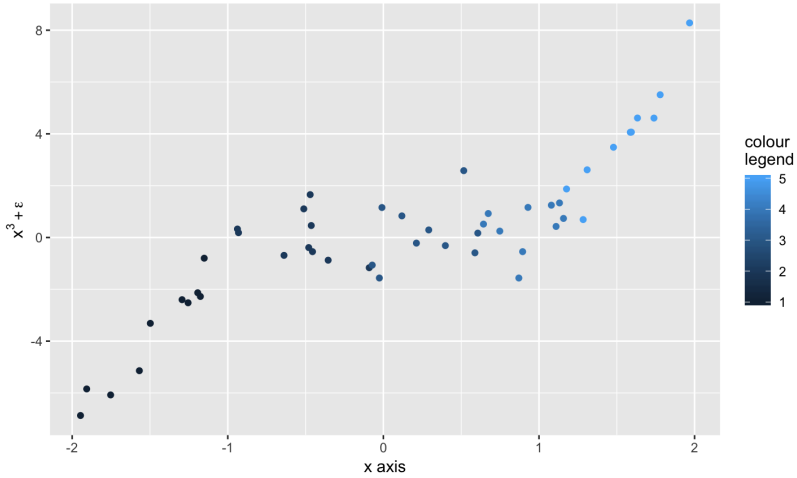
The first argument to the `scale_()` function, `name`, controls the axis label/legend title.

```
set.seed(0)
df <- tibble(x = sort(runif(50, min = -2, max = 2)), y = x^3 + rnorm(50), z = rep(1:5, times = rep(10, times = 5)))
p1 <- ggplot(df, aes(x, y, colour = z))
p1 + geom_point() + scale_x_continuous("x axis") + scale_y_continuous(quote(x^3 + epsilon)) +
  scale_color_continuous("colour\nlegend")
```

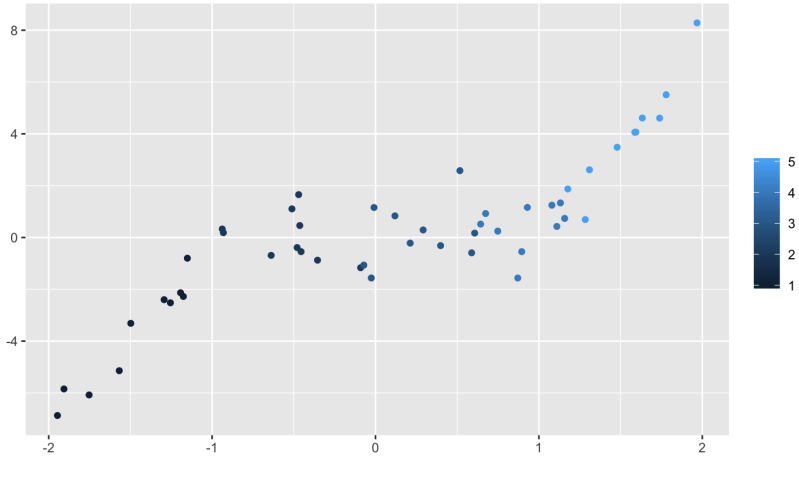


Because tweaking these labels is such a common task, there are three helpers: `xlab()`, `ylab()` and `labs()`:

```
p1 + geom_point() + xlab("x axis") +
  ylab(quote(x^3 + epsilon)) + labs(colour = "colour\nlegend")
```




```
p1 + geom_point() + labs(x = "", y = NULL,
        colour = NULL) #  two ways to remove the axis label
```

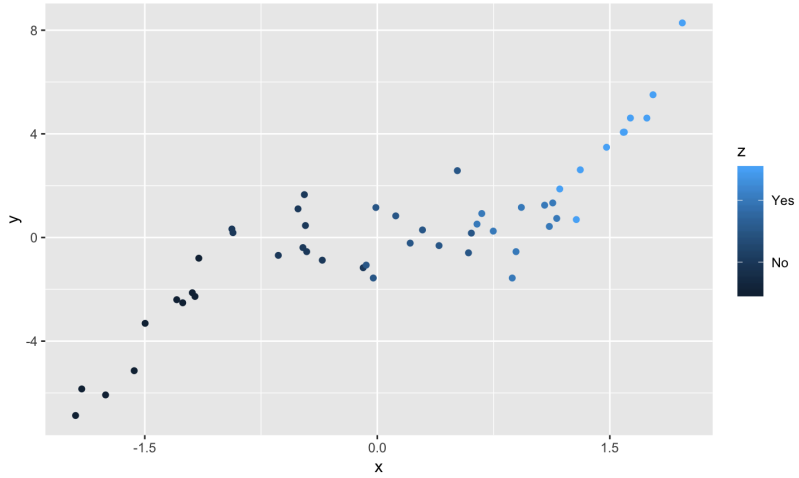


`breaks` and `labels`

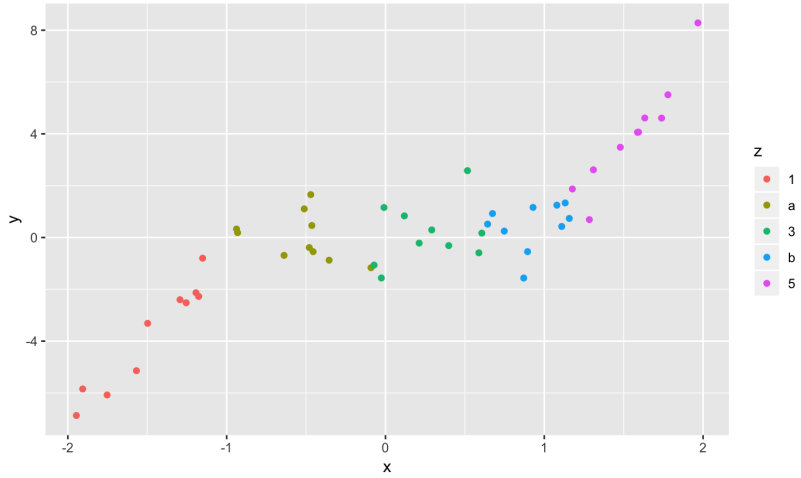
The `breaks` argument controls which values appear as tick marks on axes and keys on legends.

Each break has an associated label, controlled by the `labels` argument.

```
p1 + geom_point() + scale_x_continuous(breaks = c(-1.5, 0, 1.5)) +
  scale_color_continuous(breaks = c(2, 4), labels = c("No", "Yes"))
```




```
p1 + geom_point(aes(colour = as.factor(z))) +
  scale_colour_hue(labels = c("2" = "a", "4" = "b"))
```



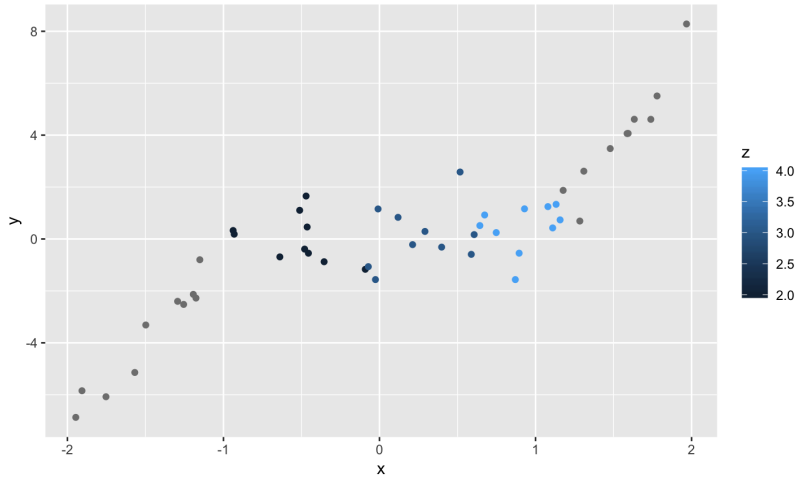
Setting `breaks` or `labels` to `NULL` can suppress them.

limits

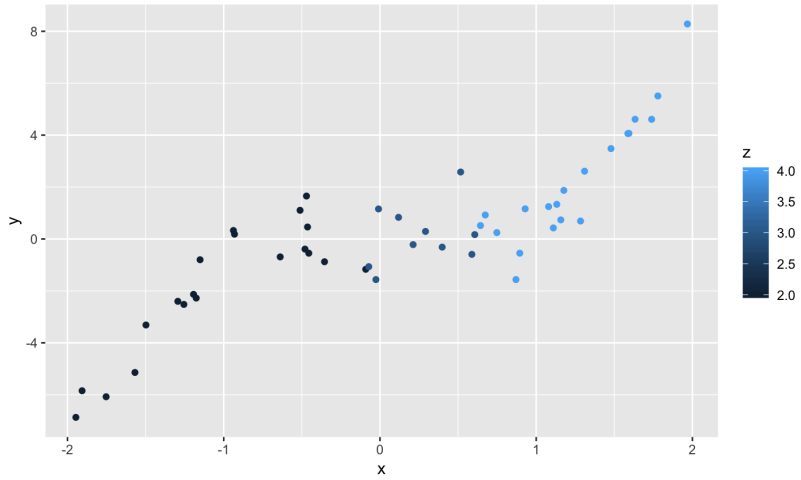
The `limits` of a scale controls the range of values to be mapped on the scale.

By default, any data outside the limits is converted to `NA`. This can be overridden by the `oob` (out of bounds) argument.

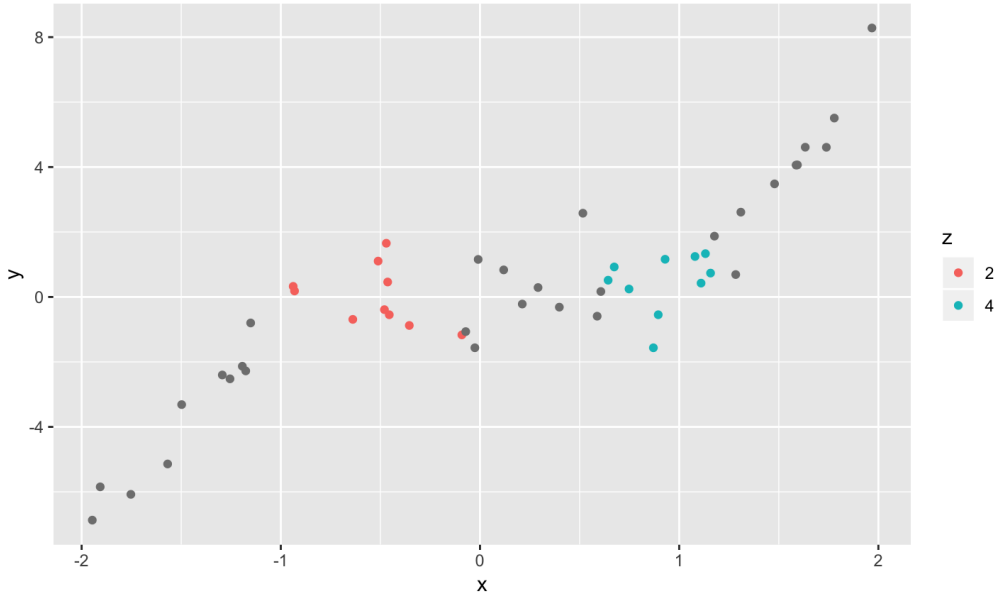
```
p1 + geom_point() +
  scale_colour_continuous(limits = c(2, 4))
```




```
p1 + geom_point() +
  scale_colour_continuous(limits = c(2, 4), oob = scales::squish)
```



```
p1 + geom_point(aes(colour = as.factor(z))) + scale_colour_hue(limits = c("2", "4"))
```



Some helpers: `xlim()`, `ylim()` and `lims()`.

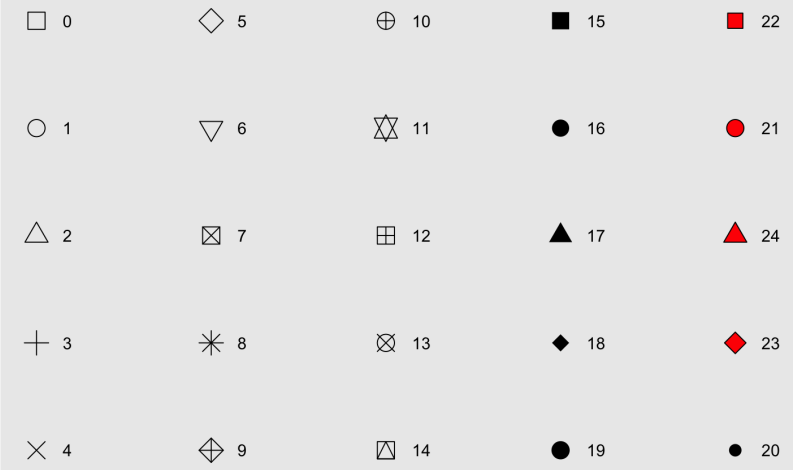
```
... + xlim("a", "b", "c")  # or equivalently lims(x = c('a', 'b', 'c'))
```

Postion Aesthetics	Variable Type	Scale	Special Arguments
x or y	continuous	continuous	`trans`: `"log10"`, `"sqrt"`, `"reverse"`, etc. shortcuts: `scale_x_log10()`, `scale_x_sqrt()`, `scale_x_reverse()`, etc.
		date	`date_breaks`: `"2 weeks"`, `"10 years"`, etc. ; `date_labels`: codes defined in `strftime()`, e.g., `"%b %d"`
		datetime	
		time	
	discrete	discrete	

Non-Position Aesthetics	Variable Type	Scale	Special Arguments
colour or fill		continuous	
	continuous	**gradient, gradient2, or gradientn**	`low`, `high`; `mid`; `colours`
		distiller	`palette`: `"Set1"`, `"Greens"`, `1`, etc.; `direction`: the order of colors in the scale
		hue	
	discrete	**brewer**	`palette`: `"Set1"`, `"Greens"`, `1`, etc.; `direction`: the order of colors in the scale
		grey	`start` and `end`: grey value at low and high ends of palette

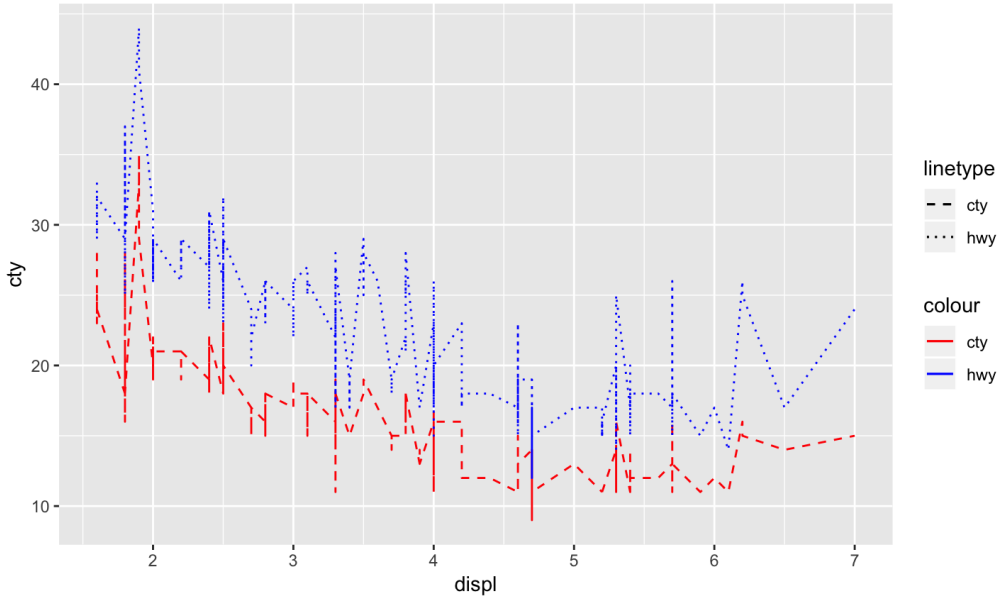
The `_manual()` Scale

Some discrete scales, `scale_linetype()`, `scale_shape()`, and `scale_size_discrete()` basically have no options, and are just a list of valid values that are mapped to the unique discrete values.



We need to create our own scale with the `_manual()` scale to customize them:

```
ggplot(mpg, aes(displ)) +
  geom_line(aes(y = cty, colour = "cty", linetype = "cty")) +
  geom_line(aes(y = hwy, colour = "hwy",  linetype = "hwy")) +
  scale_linetype_manual(values = c("cty" = 2, "hwy" = 3)) +
  scale_colour_manual(values = c("cty" = "red", "hwy" = "blue"))
```

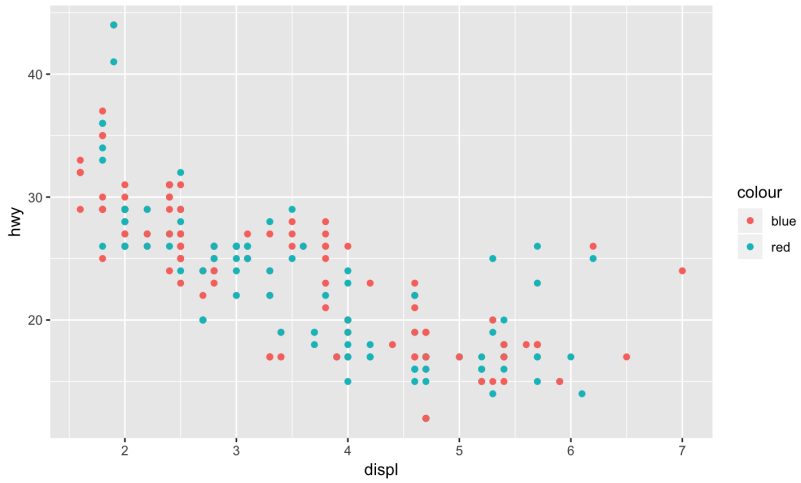


The `_identity()` Scale

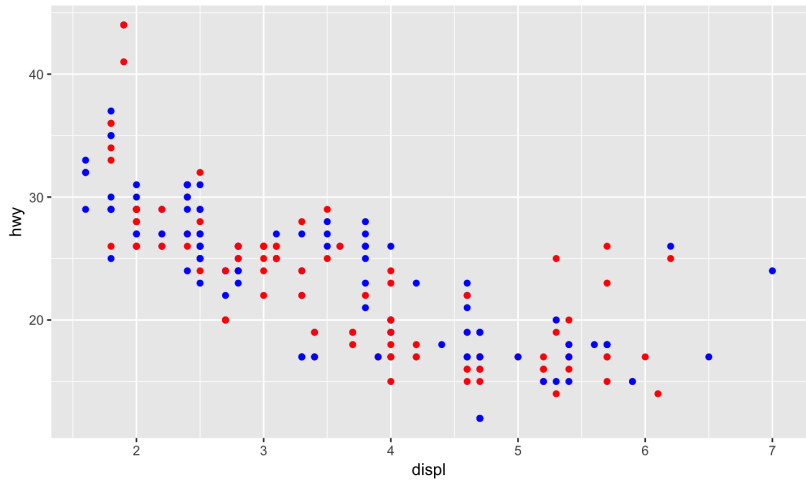
The **identity scale**, specified by functions suffixed with `_identity()`, is used when the data and aesthetic spaces are the same.

```
col <- sample(rep(c("red", "blue"), times = 117))
```

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = col)) + scale_colour_discrete("colour")
```




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

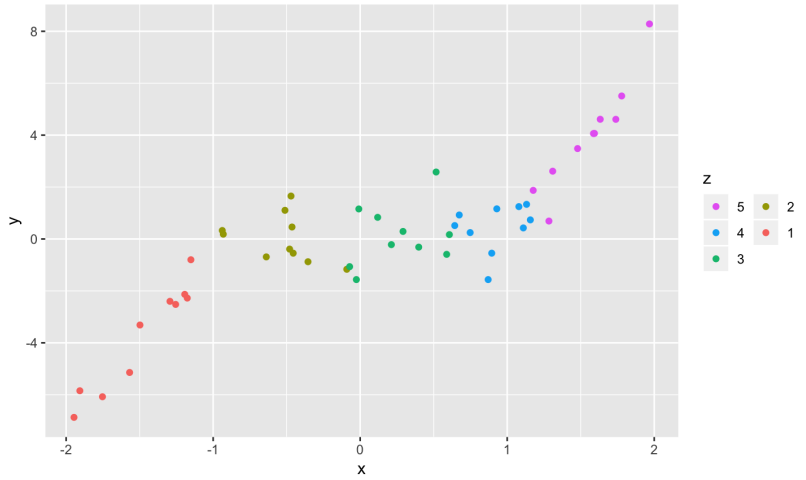


Guide Functions

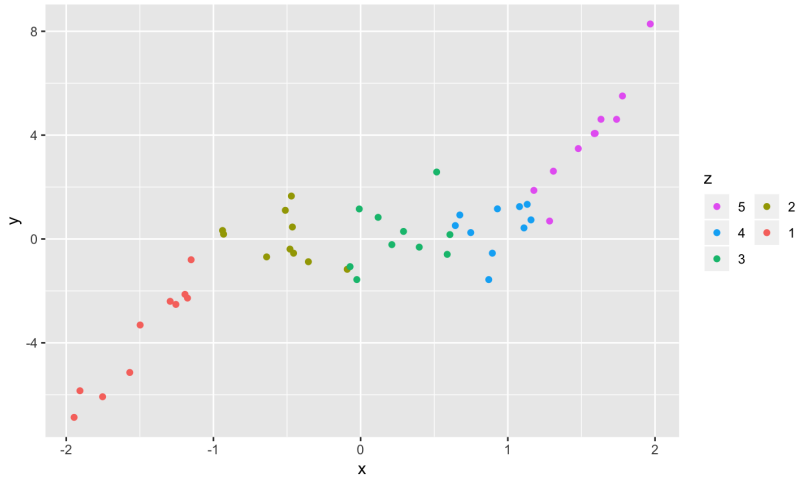
`guide_colourbar()` (only with continuous colour scales) and `guide_legend()` (with any discrete or continuous aesthetic) offer additional control over guide appearance.

The default guide can be overridden using the `guide` argument of the corresponding scale function or the `guides()` helper function:

```
p1 + geom_point(aes(colour = as.factor(z))) +
  scale_colour_hue(guide = guide_legend(ncol = 2, reverse = TRUE))
```




```
p1 + geom_point(aes(colour = as.factor(z))) +
  guides(colour = guide_legend(ncol = 2, reverse = TRUE))
```

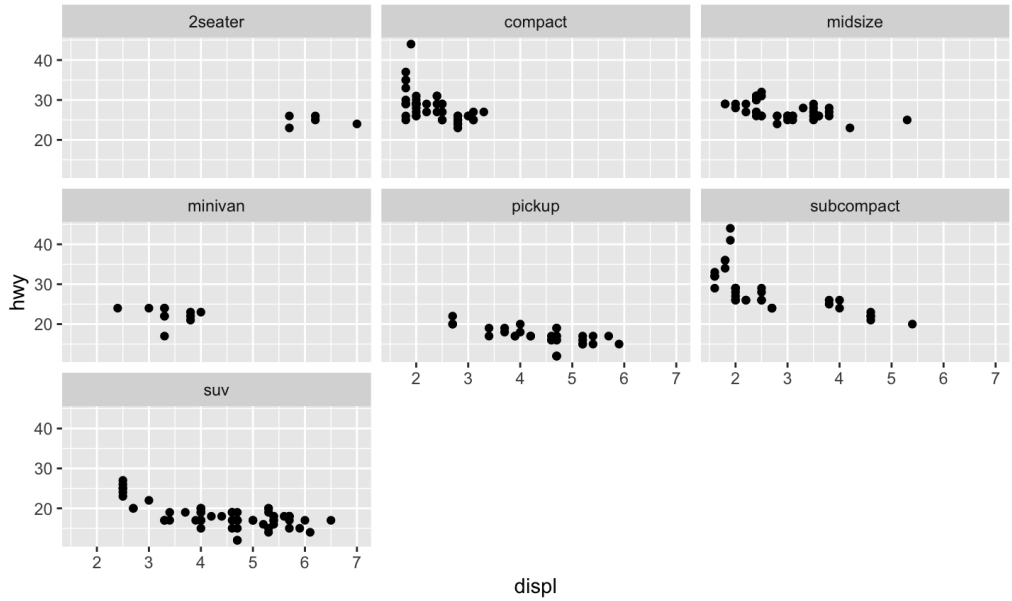


Facetting

Facetting is an alternative to displaying categorical variables on a plot

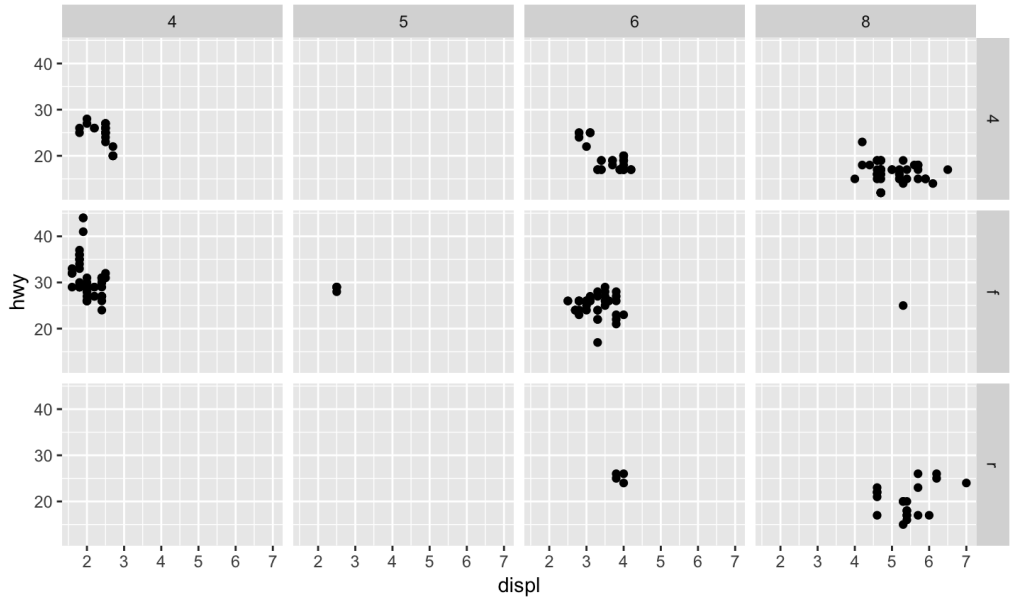
`facet_wrap()`: wraps a 1d ribbon of panels into 2d.

```
p + geom_point() + facet_wrap(~ class)
```



`facet_grid()`: produces a 2d grid of panels defined by variables which form the rows and columns.

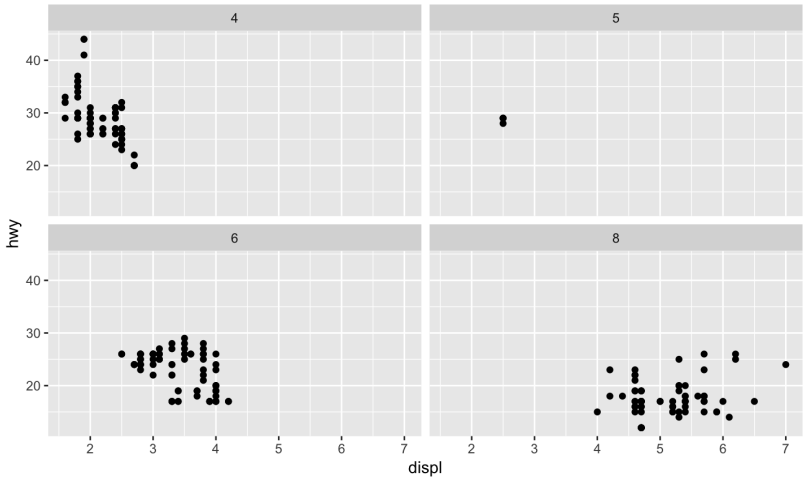
```
p + geom_point() + facet_grid(drv ~ cyl)
```



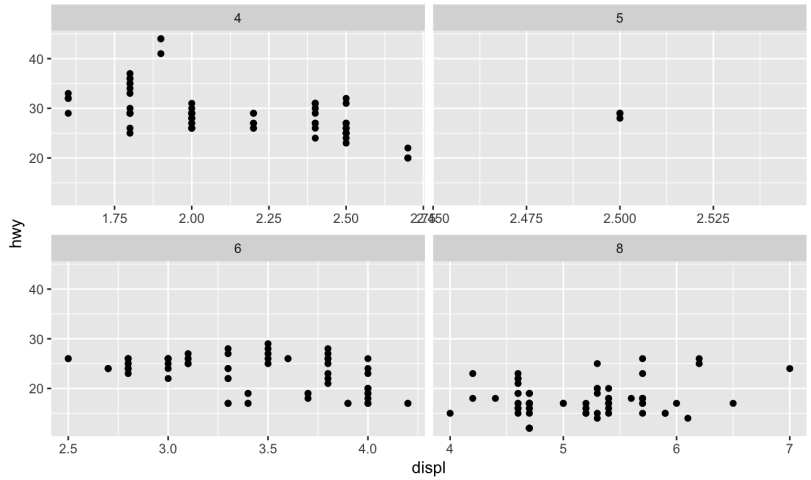
Controlling Scales

Both `facet_wrap()` and `facet_grid()` have the `scales` parameter that controls whether the position scales are the same in all panels (`fixed`) or allowed to vary between panels (`free`, `free_x`, `free_y`)

```
p + geom_point() + facet_wrap(~ cyl)
```




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

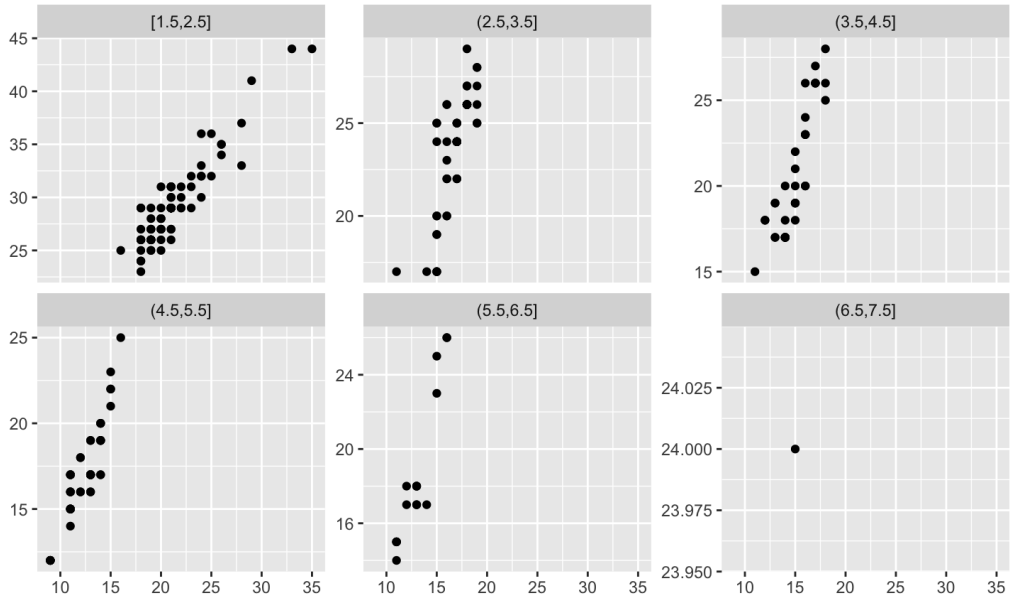


Facetting Continuous Variables

We must first discretize continuous variables so as to facet them.

`ggplot2` provides 3 helper functions: `cut_interval(x, n)`, `cut_width(x, width)`, and `cut_number(x, n = 10)`.

```
ggplot(mpg, aes(cty, hwy)) + geom_point() + labs(x = NULL, y = NULL) +
  facet_wrap(~ cut_width(displ, 1), nrow = 2, scales = "free_y")
```



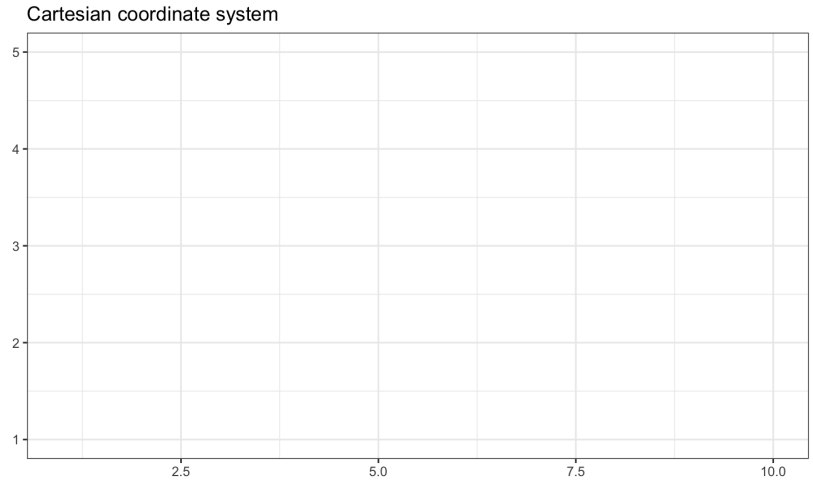
Coordinate Systems

A coordinate system (**coord**) maps the position of objects onto the plane of the plot.

While the **scales** control the values that appear on the axes and how they map from data to position, it is the coordinate system which actually draws the axes and grid lines.

There are 2 types of coordinate system:

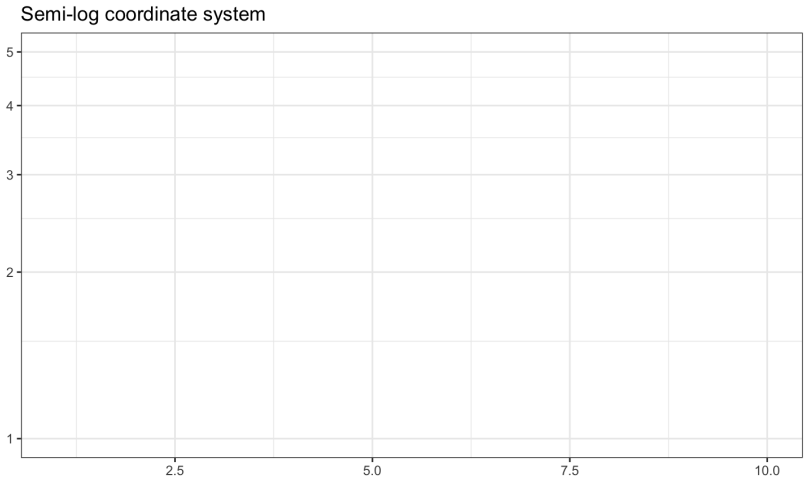
```
t <- ggplot(tibble(a = c(1, 10), b = c(1, 5)), mapping = aes(a, b)) +
  geom_blank() + labs(x = NULL, y = NULL) + theme_bw()
t + ggtitle(label = "Cartesian coordinate system")
```



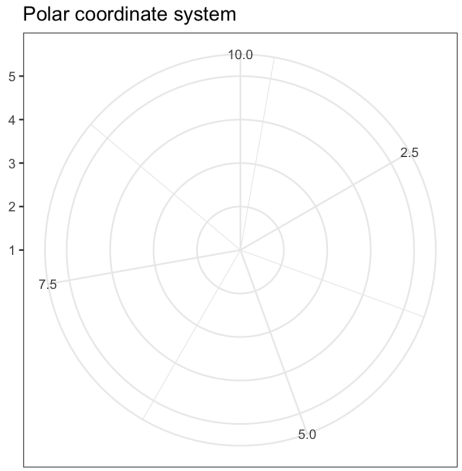
- Linear coordinate systems preserve the shape of geoms:
  - `coord_cartesian()`,
  - `coord_flip()` (with x and y axes flipped),
  - and `coord_fixed()` (with a fixed aspect ratio).

- On the contrary, non-linear coordinate systems can change the shapes: `coord_map()`/`coord_quickmap()`, `coord_polar()`, and `coord_trans()`.

```
t + coord_trans(y = "log10") +
  ggtitle(label = "Semi-log coordinate system")
```




```
t + coord_polar() +
  ggtitle(label = "Polar coordinate system")
```

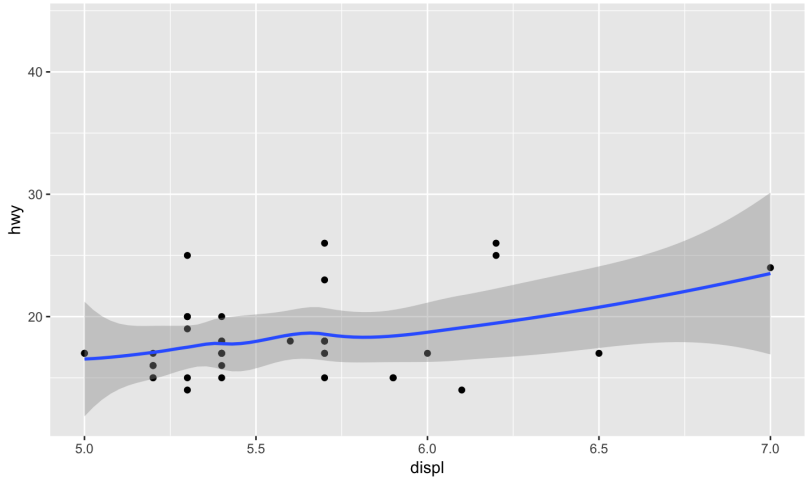


Zooming in with `xlim` and `ylim`

linear coordinate systems (`coord_cartesian()`, `coord_flip()`, and `coord_fixed()`) have arguments `xlim` and `ylim`.

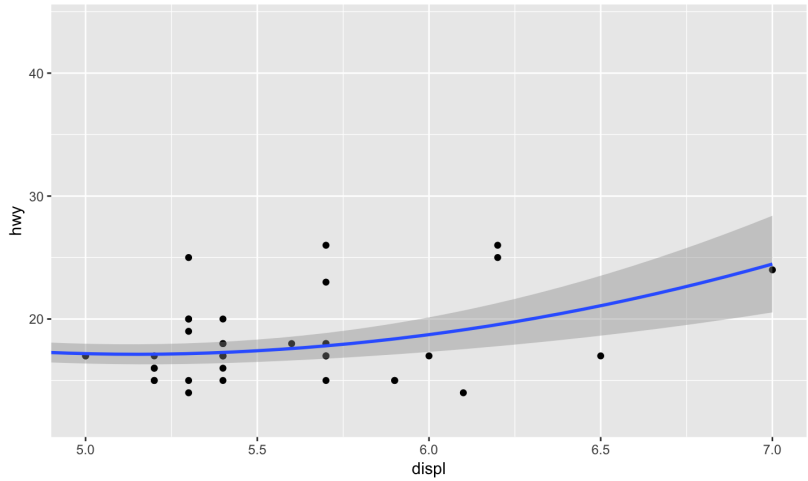
Setting scale limits throws any data outside the limits away:

```
p + geom_point() + geom_smooth() + scale_x_continuous(limits = c(5, 7))
```



Setting coordinate system limits keeps the data while displaying a small region of the plot:

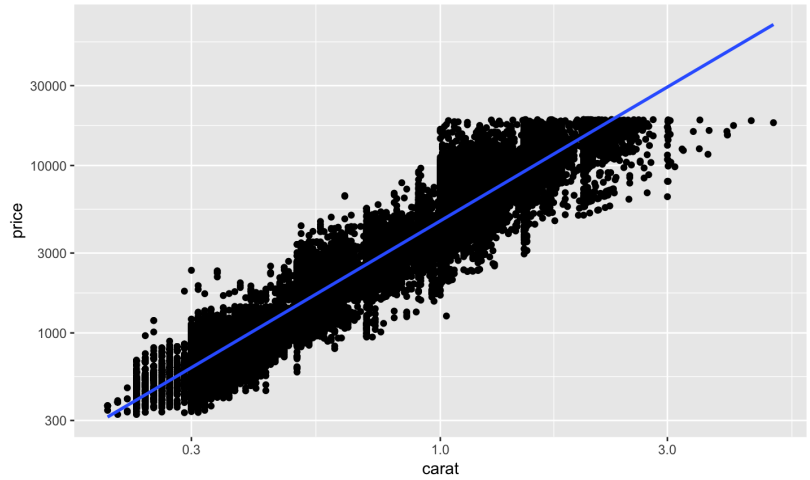
```
p + geom_point() + geom_smooth() + coord_cartesian(xlim = c(5, 7))
```



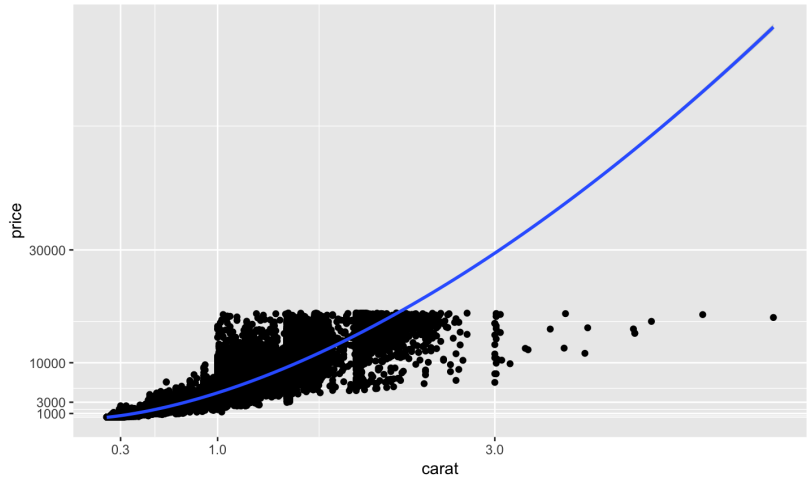
Transforming Data with `coord_trans()`

`coord_trans()` occurs after statistical transformation and will affect the visual appearance of geoms

```
ggplot(diamonds, aes(carat, price)) +
  geom_point() +  geom_smooth(method = "lm") +
  scale_x_log10() +
  scale_y_log10()
```




```
ggplot(diamonds, aes(carat, price)) +
  geom_point() +  geom_smooth(method = "lm") +
  scale_x_log10() + scale_y_log10() +
  coord_trans(x = scales::exp_trans(10), y = scales::exp_trans(10))
```



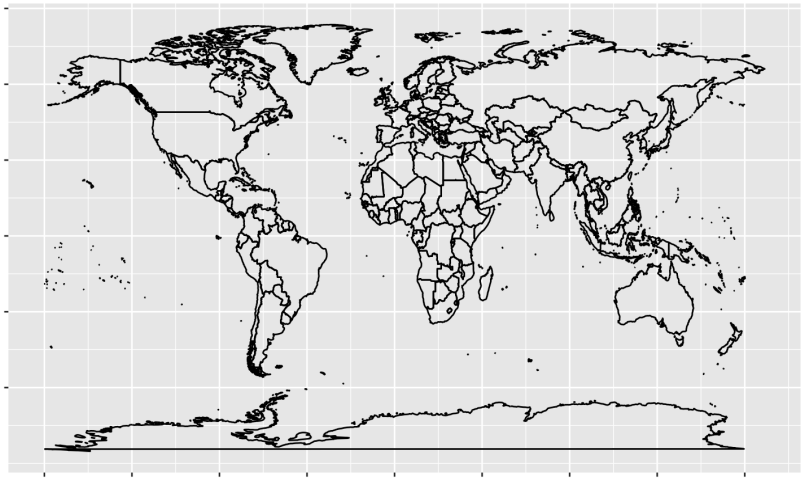
Map Projections with `coord_map()`

Maps are intrinsically displays of spherical data.

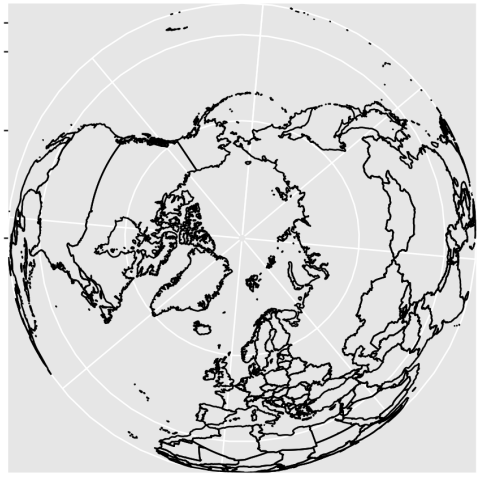
We can project the raw longitudes and latitudes data with `coord_map()`:

```
worldmap <- ggplot(map_data("world"), aes(long, lat, group = group)) +
  geom_path() + scale_y_continuous(NULL, breaks = (-2:3) * 30, labels = NULL) +
  scale_x_continuous(NULL, breaks = (-4:4) * 45, labels = NULL)
```

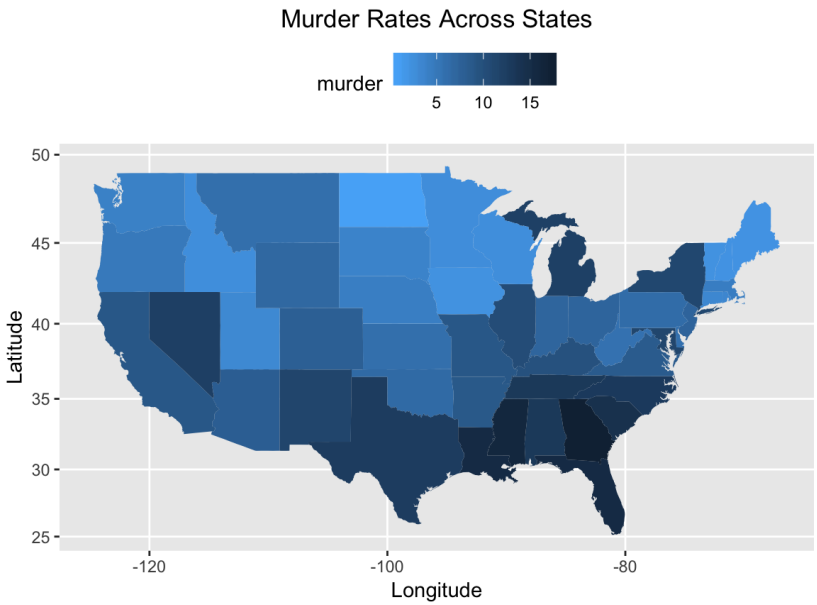
```
worldmap
```



```
worldmap + coord_map("ortho")
```



We can use `geom_polygon()` together with map data to plot a **heatmap** that gives a graphical representation of individual values of a variable as colors.



Themes: The Non-Data Part of a Plot

`ggplot2` separates the control over elements of a plot into data and non-data parts.

After the plot has been created, every detail of the rendering can be edited using the **theming system**.

The components of the theming system can be exemplified by a call to the `theme()` function:

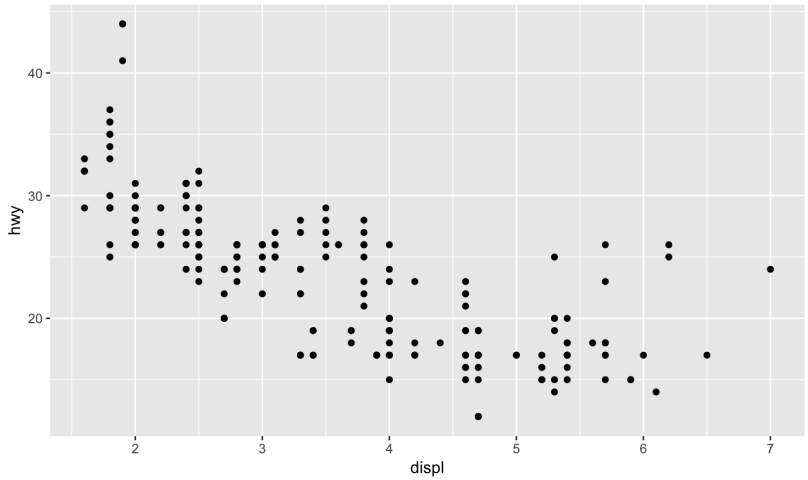
```
theme(plot.title = element_text(colour = "red"))  # overrides the default theme elements by calling element functions.
```

- Theme **elements**: the non-data elements that we can control; E.g., `plot.title`, `axis.ticks.x`, `legend.key.height`, etc.
  - These elements can be roughly grouped into five categories: plot, axis, legend, panel and facet.
- Each element is associated with an **element function**, describeing the visual properties of the element.
  - There are 4 basic types of built-in element functions: `element_text()`, `element_line()`, `element_rect()`, and `element_blank()`.

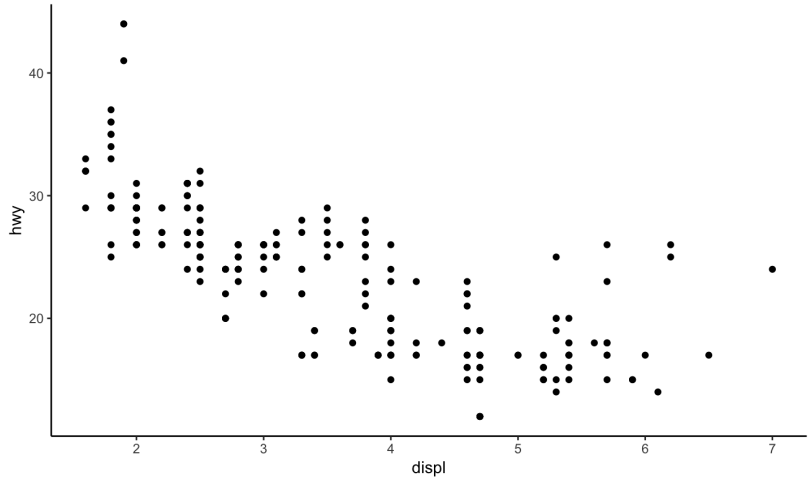
Complete Themes

There are several **complete themes** built in to `ggplot2`, setting all of the theme elements to values designed to work together harmoniously.

```
p + geom_point()  # theme_gray() is the default complete theme
```




```
p + geom_point() + theme_classic()
```

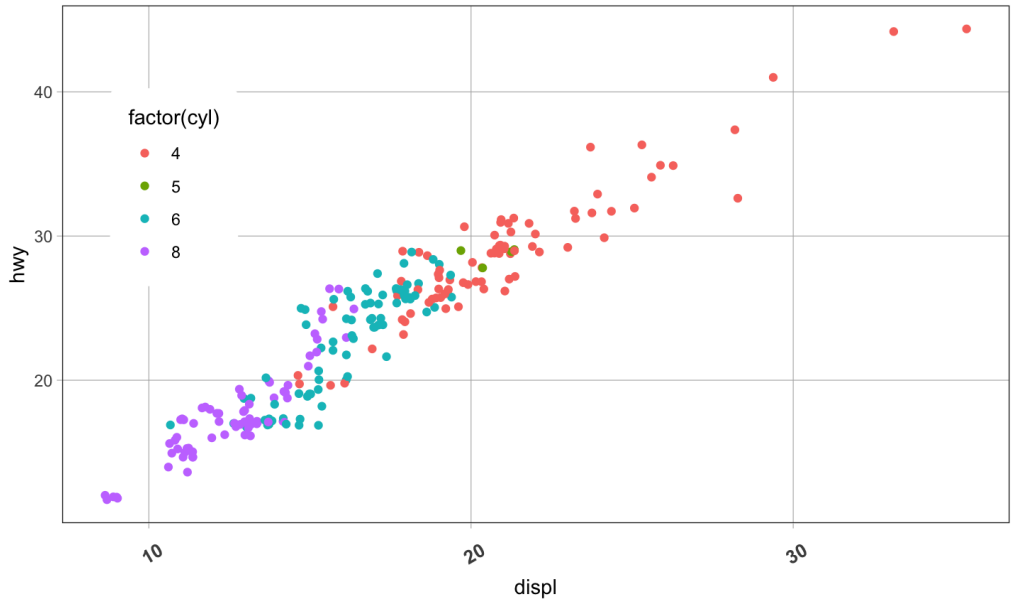


Packages like `ggthemes` provide more complete themes to use.

Modifying Theme Components

Use code of the form `plot + theme(element.name = element_function())` to modify an individual theme component.

```
p + geom_jitter(aes(cty, hwy, colour = factor(cyl))) + theme_bw() +
  theme(legend.background = element_rect(fill = "white", size = 4, colour = "white"),
        legend.justification = c(-0.5, 1.5), legend.position = c(0, 1),
        axis.ticks = element_line(colour = "grey70", size = 0.2),
        axis.text.x = element_text(margin = margin(t = 10), face = "bold", size = 10, angle = 30),
        panel.grid.major = element_line(colour = "grey70", size = 0.2), panel.grid.minor = element_blank())
```



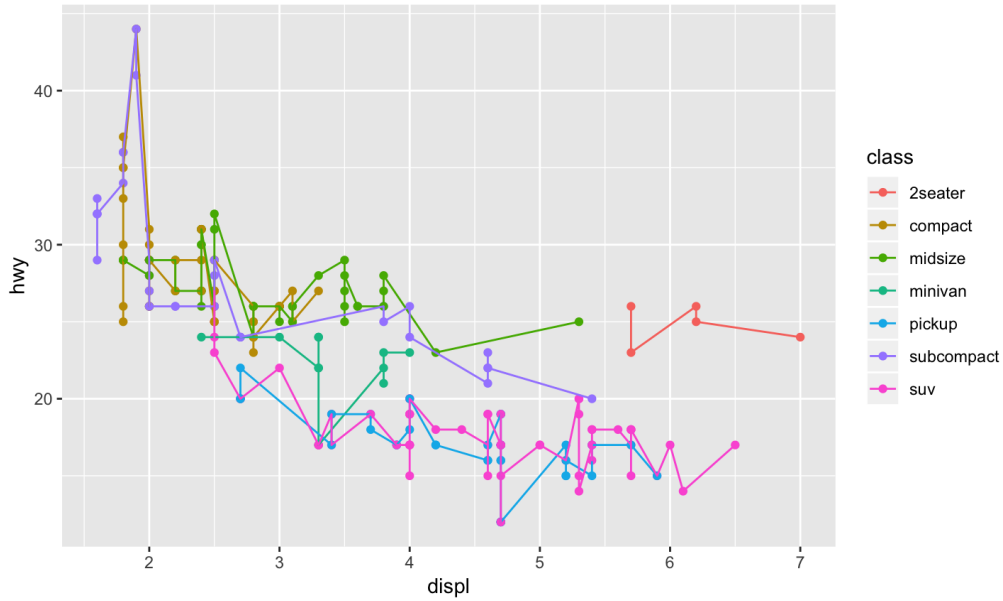
Quick Plot

`qplot()` or `quickplot()` provides a "quick-plot" routine, which essentially serves a similar purpose to R's base `plot()` function.

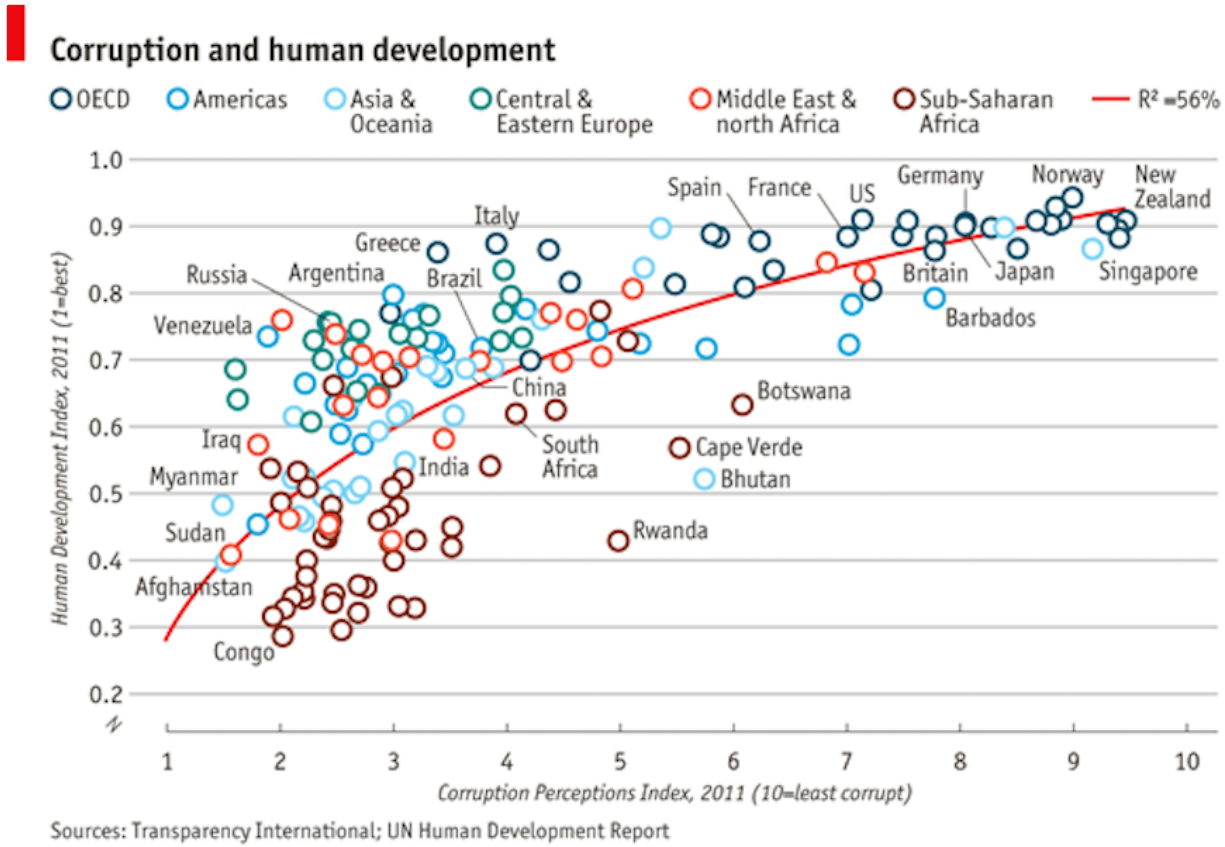
```
str(qplot)
```

```
## function (x, y, ..., data, facets = NULL, margins = FALSE, geom = "auto",
##     xlim = c(NA, NA), ylim = c(NA, NA), log = "", main = NULL, xlab = NULL,
##     ylab = NULL, asp = NA, stat = NULL, position = NULL)
```

```
qplot(x = displ, y = hwy, colour = class, data = mpg, geom = c("point", "line"))
```



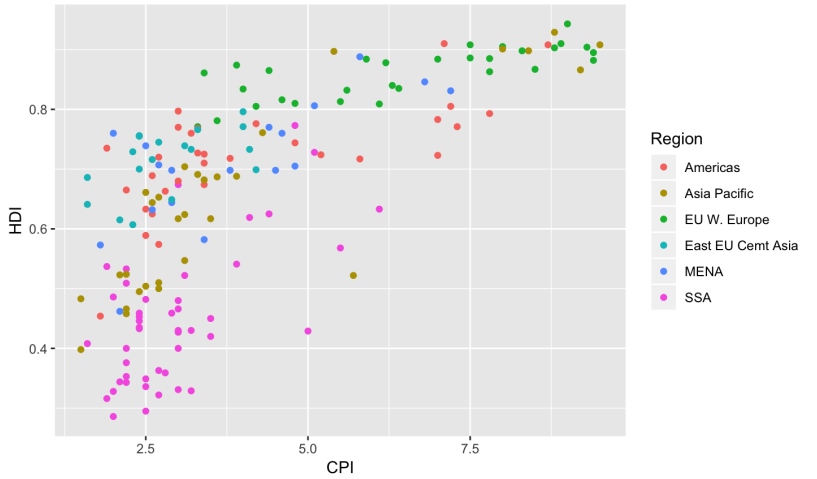
Challenge: Recreating An *Economist* Graph



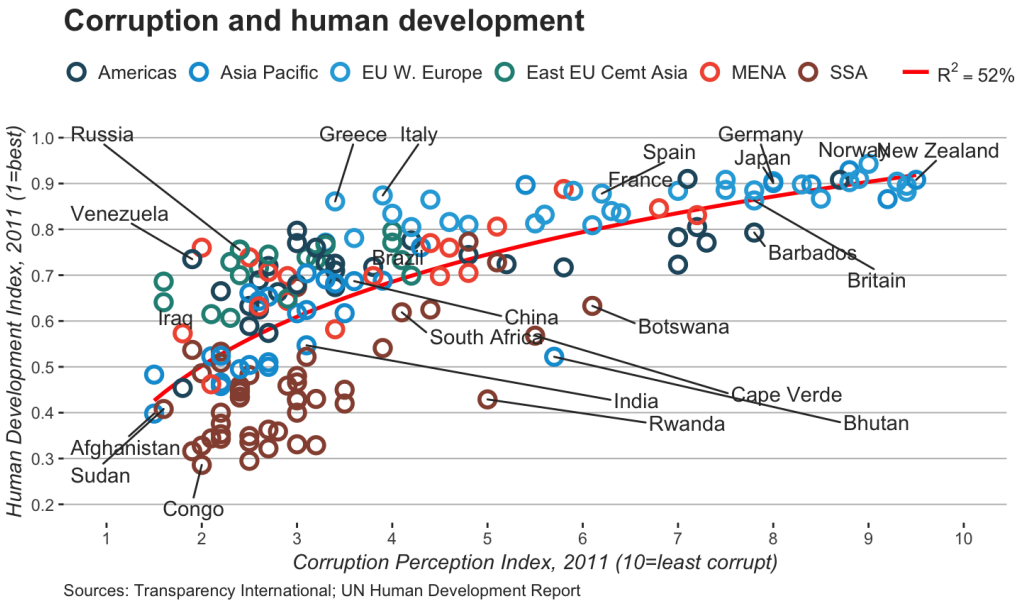
```
econData <- read_csv("EconomistData.csv")
econData
```

```
## # A tibble: 173 x 5
##    Country     HDI.Rank   HDI   CPI Region
##    <chr>          <int> <dbl> <dbl> <chr>
##  1 Afghanistan      172 0.398   1.5 Asia Pacific
##  2 Albania           70 0.739   3.1 East EU Cemt Asia
##  3 Algeria           96 0.698   2.9 MENA
##  4 Angola           148 0.486   2   SSA
##  5 Argentina         45 0.797   3   Americas
##  6 Armenia           86 0.716   2.6 East EU Cemt Asia
##  7 Australia          2 0.929   8.8 Asia Pacific
##  8 Austria           19 0.885   7.8 EU W. Europe
##  9 Azerbaijan        91 0.7     2.4 East EU Cemt Asia
## 10 Bahamas           53 0.771   7.3 Americas
## # ... with 163 more rows
```

```
econFig <- ggplot(econData, aes(x = CPI, y = HDI, color = Region))
econFig + geom_point()
```



```
econFig + geom_smooth(aes(linetype = "r2"), method = "lm",
              formula = y ~ x + log(x), se = FALSE, color = "red") +
  geom_point(shape = 1, size = 3, stroke = 1.5) +
  geom_text_repel(aes(label = Country), color = "gray20",
                  data = filter(econData, Country %in% pointsToLabel), force = 10) +
  scale_x_continuous(name = "Corruption Perception Index, 2011 (10=least corrupt)", limits = c(1.0, 10.0), breaks = 1:10) +
  scale_y_continuous(name = "Human Development Index, 2011 (1=best)", limits = c(0.2, 1.0), breaks = seq(0.2, 1.0, by = 0.1)) +
  scale_color_manual(name = "", values = c("#24576D", "#099DD7", "#28AADC", "#248E84", "#F2583F", "#96503F"),
                     guide = guide_legend(nrow = 1, order=1)) +
  scale_alpha_discrete(range = c(0, 1), guide = FALSE) +
  scale_linetype(name = "", breaks = "r2", labels = list(bquote(R^2==.(mR2)))) +
  ggtitle("Corruption and human development") +
  labs(caption="Sources: Transparency International; UN Human Development Report") +
  theme_bw() +
  theme(panel.border = element_blank(), panel.grid = element_blank(), panel.grid.major.y = element_line(color = "gray"),
        text = element_text(color = "gray20"),
        axis.title.x = element_text(face="italic"), axis.title.y = element_text(face="italic"),
        legend.position = "top", legend.direction = "horizontal", legend.box = "horizontal",
        legend.text = element_text(size = 12), legend.spacing.x = unit(0.2, "line"),
        plot.caption = element_text(hjust=0), plot.title = element_text(size = 16, face = "bold"))
```



`ggsave()` exports a ggplot to a pdf file of a specified size on disk.

```
ggsave(filename = "econFig.pdf", plot = econFig, width = 6, height = 7)
```

`ggplot2` Plotting System Summary

All together, the **layered grammar of graphics** defines a plot as the combination of:

- A default dataset and set of mappings from variables to aesthetics.
- One or more layers, each composed of a geometric object, a statistical transformation, a position adjustment, and optionally, a dataset and aesthetic mappings.
- One scale for each aesthetic mapping.
- A coordinate system.
- The facetting specification.

By thinking "verb", "noun", "adjective", etc. for graphics, `ggplot2` provides a "theory" of graphics on which to build new graphics and graphical objects and shortens the distance from mind to page.