Hoban lab ggplot2 walk-through

Ash Hamilton

To reproduce the code throughout this tutorial you only need to load the `ggplot2` package

```
library(ggplot2)
```

Ggplot2 also comes with a number of built-in data sets! This tutorial will (mostly) use the ggplot2 provided Midwest dataset, which is a data frame that contains demographic information of Midwest counties from 2000 US census.

Grammar of Graphics:

The philosophy of ggplot2 is based on the Grammar of Graphics, which adapted by Hadley Wickham to describe the components of a plot. In this framework, a plot at its most basic must be composed of:

- the data being plotted
- the geometric objects (circles, lines, etc.) that appear on the plot
- a set of mappings from variables in the data to the aesthetics (appearance) of the geometric objects

The Grammar of Graphics also describes more complex (optional) plot components such as:

- a position adjustment for locating each geometric object on the plot
- a scale (e.g., range of values) for each aesthetic mapping used
- a coordinate system used to organize the geometric objects
- the facets or groups of data shown in different plots
- a statistical transformation used to calculate the data values used in the plot

Each of these components can be controlled with unique commands in ggplot2.

A basic plot:

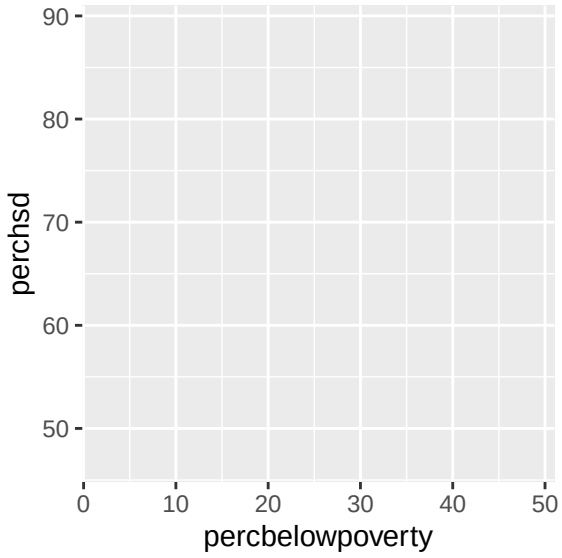
In order to create a plot, you:

1) Call the `ggplot()` function which creates a blank canvas

```
ggplot(data = midwest)
```

2) Specify aesthetic mappings, which specifies how you want to map variables to visual aspects. In this case we are simply mapping the `percbelowpoverty` (percent of residents of the county below the poverty line) and `perchsd` (percent of residents of the county with a highschool diploma) variables to the x- and y-axes respectively.

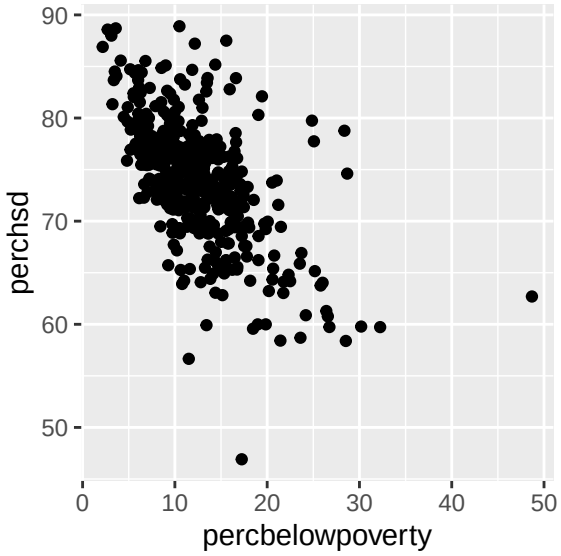
```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd))
```



Note that despite specifying the data and the axes, the plot itself is empty.

3) Add a layer that specifies which type of geometric object will show up on the plot

```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point()
```



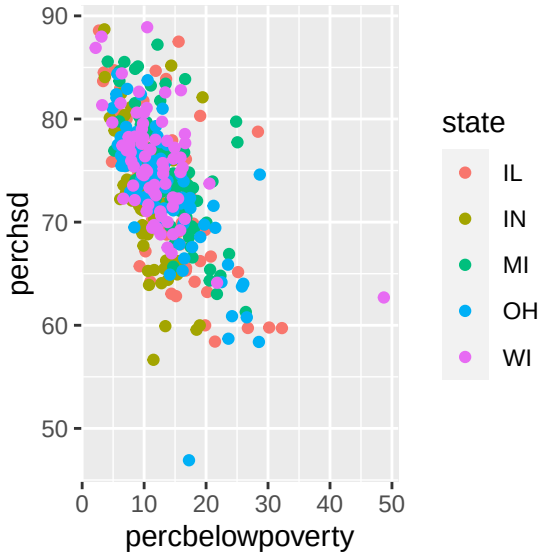
Notice the `+` in the above code block. You will always use the `+` operator to add new layers onto your visualization.

Aesthetics:

Aesthetic mappings (arguments within the `aes()` function) take properties of the data and use them to influence visual characteristics of the plotted data, such as `position`, `color`, `size`, `shape`, or transparency value (`alpha`)

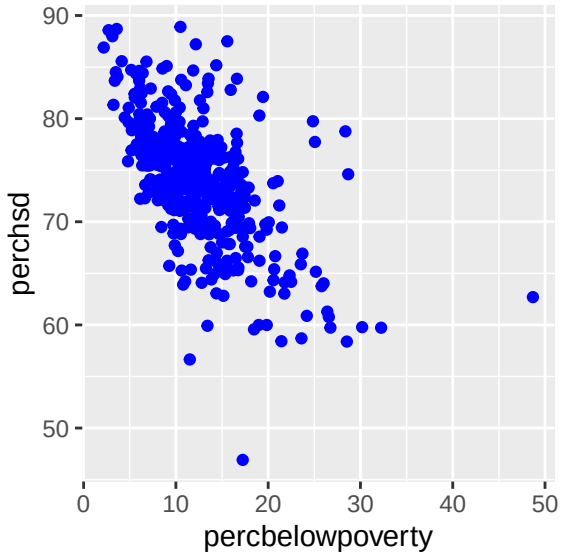
For example, we can add a mapping from the state of the counties to a color characteristic:

```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(color = state))
```



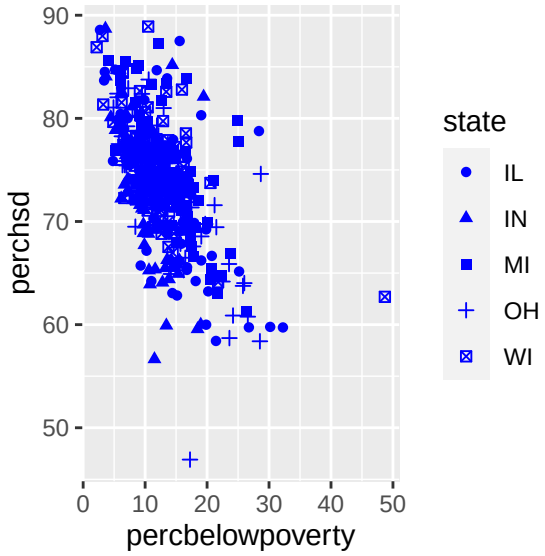
If you wish to apply an aesthetic property to an entire geometry (instead of mapping a characteristic of the data with the aesthetic), you can set that property as an argument to the geom method, outside of the `aes()` call

```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(color = "blue")
```



All geometry types will leverage the aesthetic mappings supplied but the specific visual properties that the data of a specific geometry type will map to will vary. For example, you can map data to the shape of a `geom_point` (e.g., if they should be circles or squares)

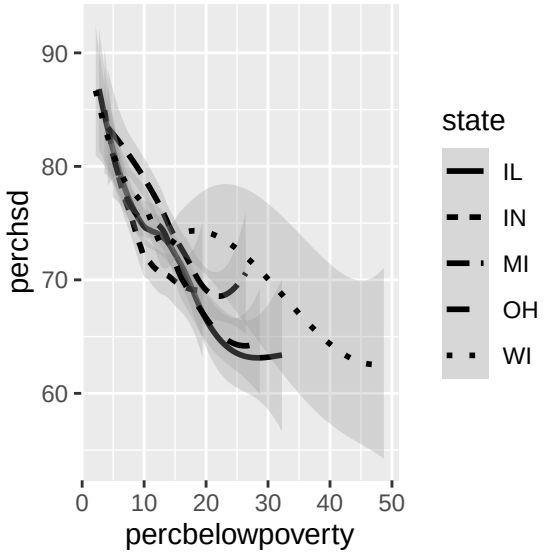
```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(shape = state), color = "blue") # Note the use of an aesthetic call that maps data to
```



or you can map data to the `linetype` of a `geom_line` or `geom_smooth` (e.g., if it is solid or dotted)

```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_smooth(aes(linetype = state), color = "black", alpha = .3) # Note like above, the specification
```

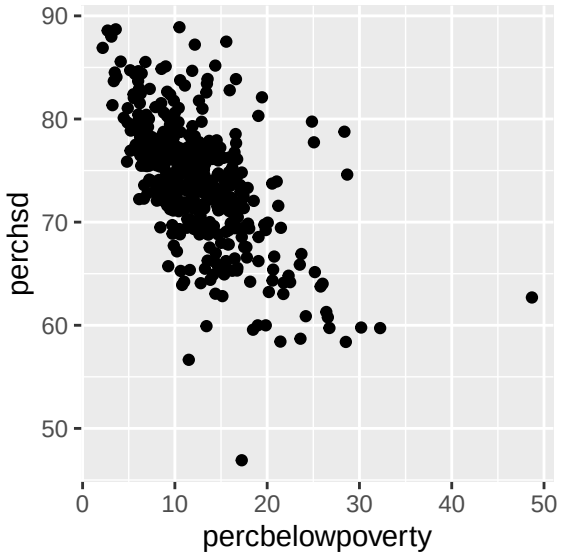
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



but not vice versa (as you can see below, ggplot will ouput a warning when you try to apply an impossible aesthetic to a geometry type)

```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(linetype = state))
```

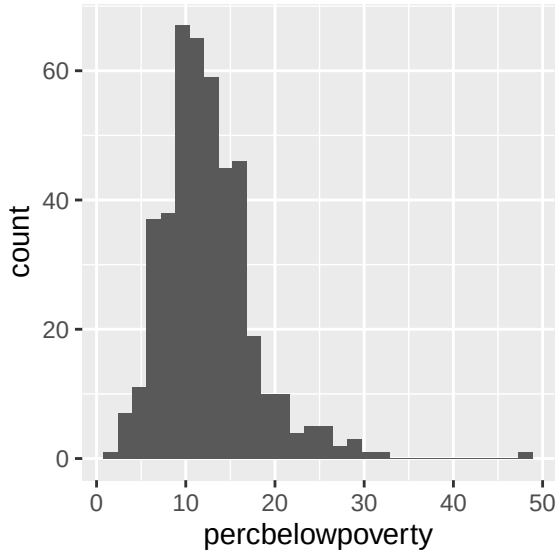
```
## Warning in geom_point(aes(linetype = state)): Ignoring unknown aesthetics:
## linetype
```



Most geoms require specifying an x and y mapping within `aes()` at the bare minimum, but some geoms (like `geom_histogram`, `geom_density`, and `geom_bar`) require either an x or a y

```
ggplot(data = midwest, aes(x = percbelowpoverty)) + #No y specified!
  geom_histogram()
```

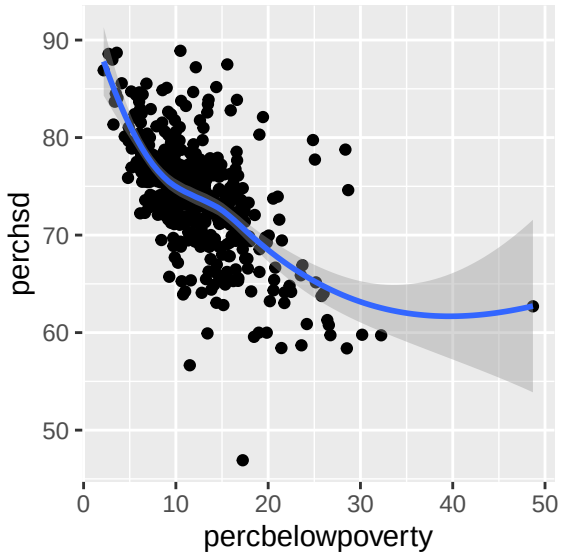
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



You can also easily add multiple geometries to a plot, allowing you to create complex graphics showing multiple aspects of your data at once!

```
ggplot(midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point() +
  geom_smooth() #geom_smooth is a function that calculates and then plots a simple linear model of your
```

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

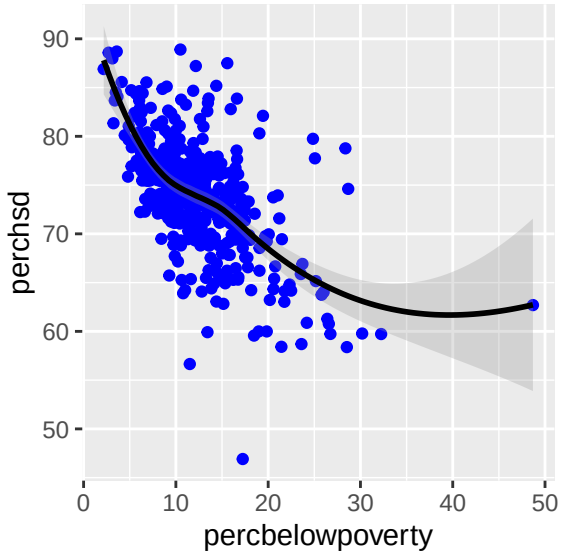


The aesthetics for each geom can be different, so you could show multiple lines on the same plot (or with different colors, styles, etc). It's also possible to give each geom a different data argument, so that you can show multiple data sets in the same plot.

For example, we can plot both points and a smoothed line for the same x and y variable but specify unique `color` and `alpha` values within each geom:

```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(color = "blue") +
    geom_smooth(color = "black", alpha = .3)
```

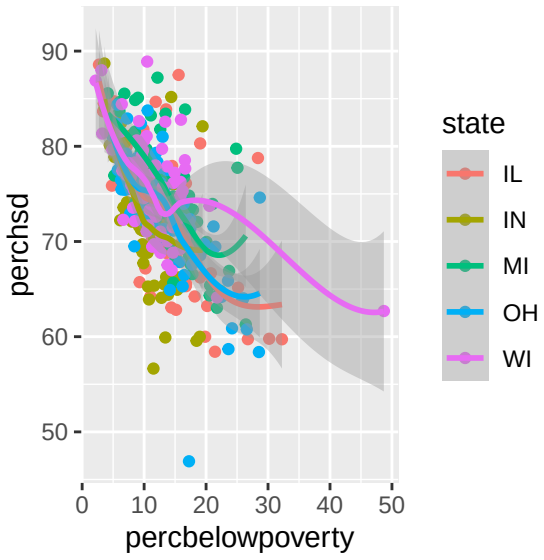
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



If we specify an aesthetic within `ggplot` it will be passed on to each geom that follows.

```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd, color = state)) +
  geom_point() +
  geom_smooth()
```

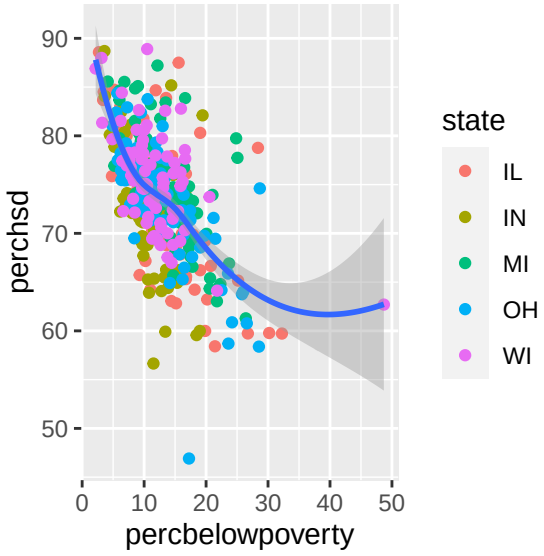
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Or we can specify certain aes within each geom, which allows us to only show certain characteristics for that specific layer (i.e. geom_point).

```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(color = state)) +
  geom_smooth()
```

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

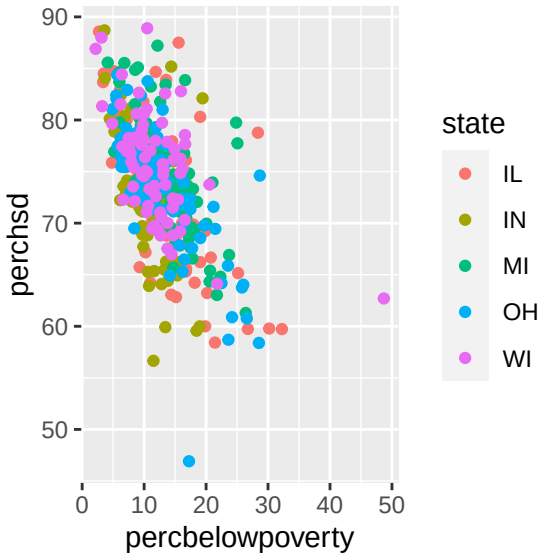


Other arguments:

These are all components of the grammar of graphics but will only be covered in brief. Extended discussions of them can be found online; see for example the Visualization chapter of Hadley's R for Data Science book.

Scales Whenever you specify an aesthetic mapping, ggplot uses a particular scale to determine the range of values that the data should map to. Thus when you specify:

```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(color = state))
```



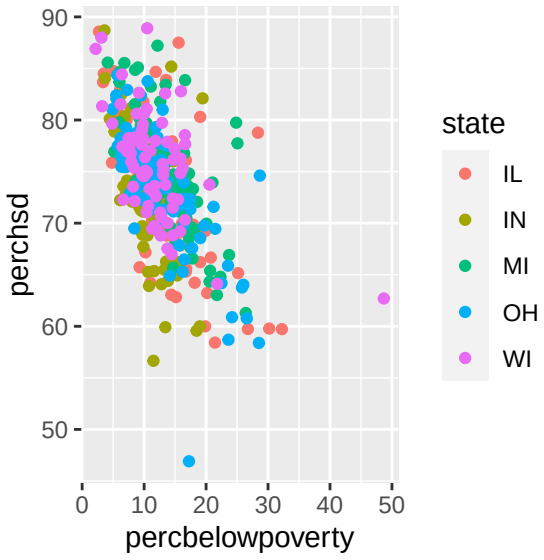
`ggplot` automatically adds a scale for each mapped aesthetic (percbelowpoverty, percbelowpoverty, state) to the plot

Each scale can be explicitly represented by a function with the following name: scale_, followed by the name of the aesthetic property, followed by an _ and the name of the scale.

A continuous scale will handle things like numeric data (where there is a continuous set of numbers), whereas a discrete scale will handle things like colors (since there is a small list of distinct colors).

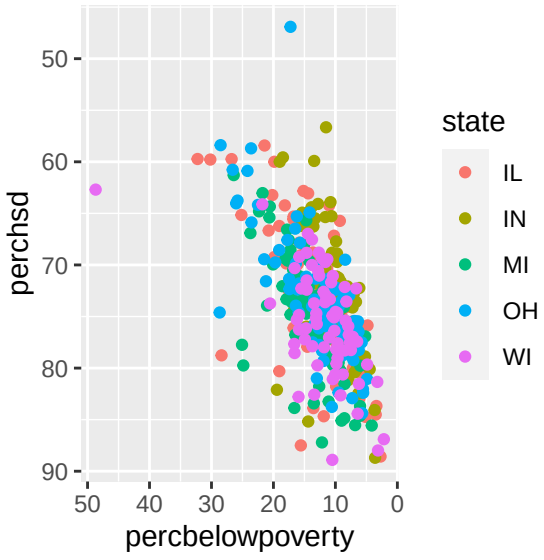
For example, the scale `ggplot` automatically applied to the above graph can be explicitly called and the same plot will be created:

```
# same as above, with explicit scales
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(color = state)) +
  scale_x_continuous() + # x is mapped from the percbelowpoverty variable which is continuous
  scale_y_continuous() + # y is mapped from the perchsd variable which is continuous
  scale_colour_discrete() # color is mapped from the state variable which is discrete
```



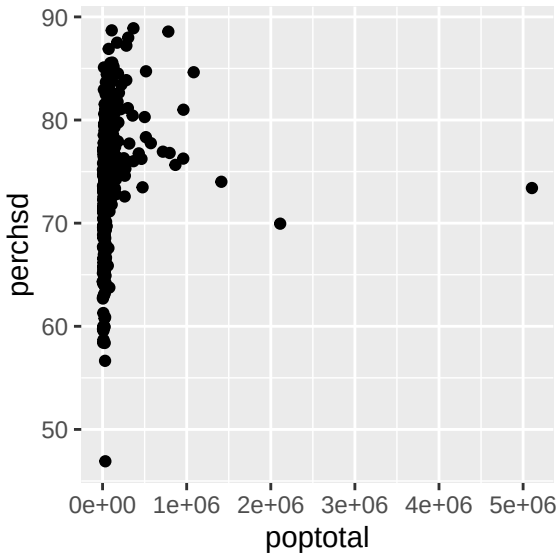
While the default scales will generally work fine, it is possible to explicitly add different scales to replace the defaults. For example, you can use a scale to change the direction of an axis:

```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(color = state)) +
  scale_x_reverse() +
  scale_y_reverse()
```

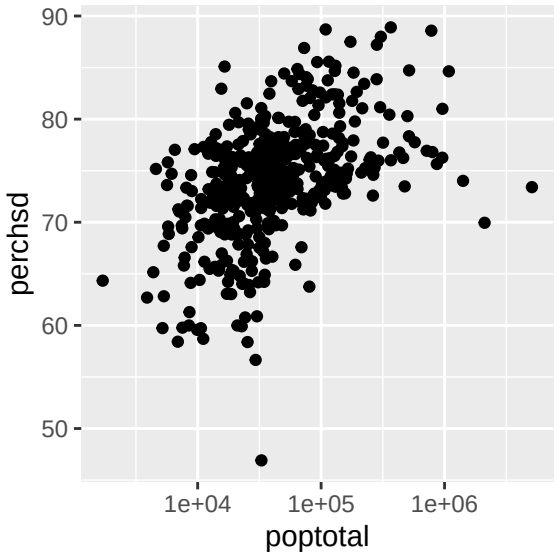


Similarly, you can use `scale_x_log10()` and `scale_x_sqrt()` to transform your scale or you can use the `trans=` argument within `scale_x_continuous` and `scale_y_continuous` to do the same thing:

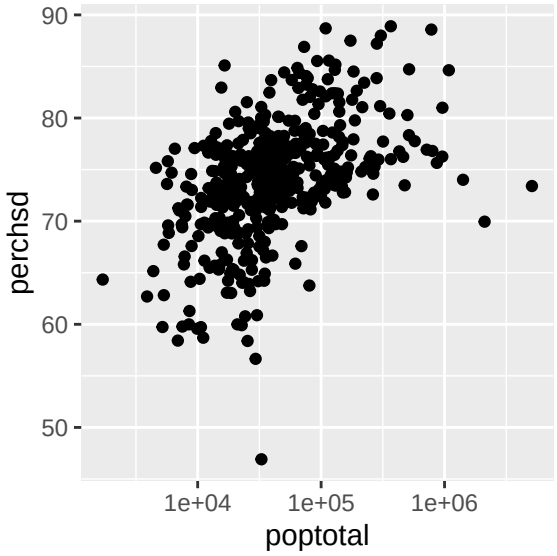
```
# initial plot, non-transformed axis
ggplot(data = midwest, aes(x = poptotal, y = perchsd)) +
  geom_point()
```




```
# transform x axis using scale_x_log10
ggplot(data = midwest, aes(x = poptotal, y = perchsd)) +
  geom_point() +
  scale_x_log10()
```




```
# transform x axis using scale_x_continuous
ggplot(data = midwest, aes(x = poptotal, y = perchsd)) +
  geom_point() +
  scale_x_continuous(trans = "log10")
```

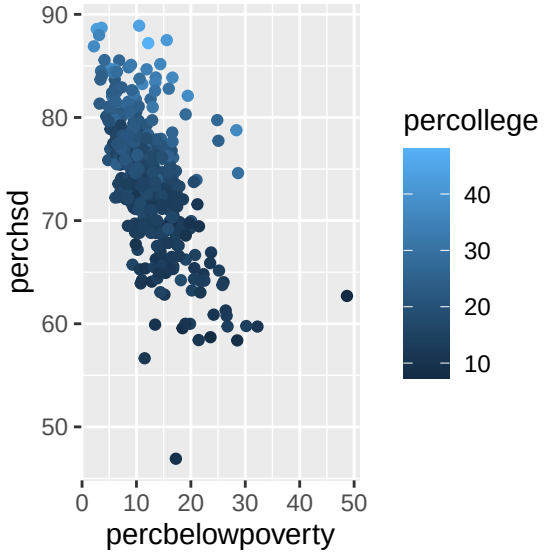


A common parameter to change is which set of colors to use in a plot. While you can use the default coloring, a more common option is to leverage the pre-defined palettes from viridis. These color sets have been carefully designed to look good and to be viewable to people with certain forms of color blindness. We can leverage these palettes by specifying the `scale_color_viridis()` function.

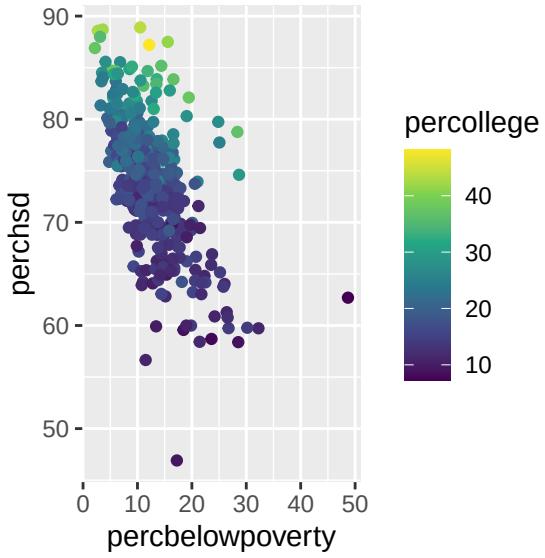
Note that you can get the possible palette names from this viridis website.

Also note that viridis has their own package and functions within that package that can be used within `ggplot`

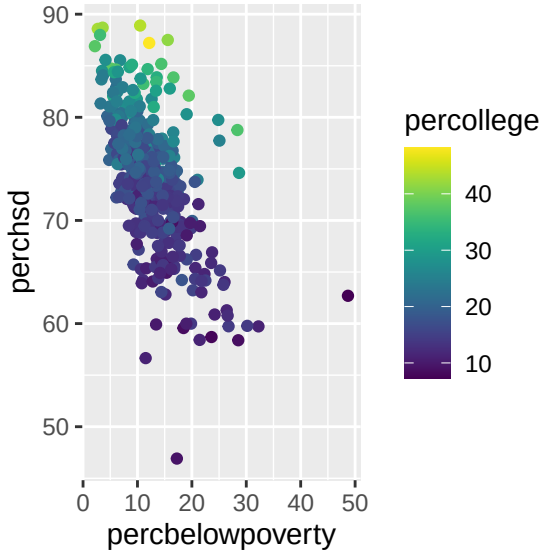
```
#default color palette
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(color = percollege))
```



```
#the default viridis color palette
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(color = percollege)) +
  scale_color_viridis_c() # call scale_color_viridis command from the viridis package
```

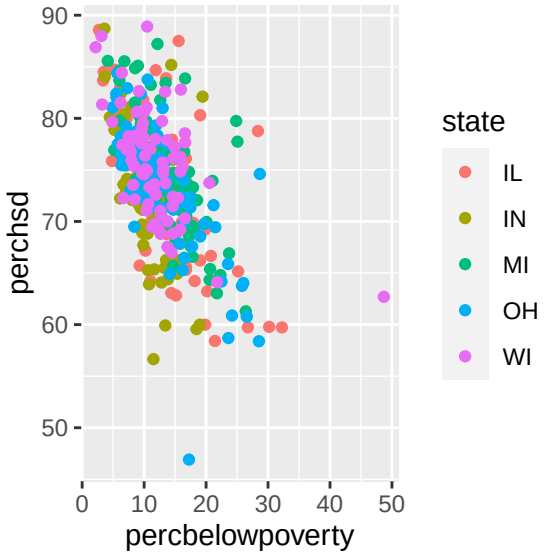



```
#calling viridis color palette from the viridis package
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(color = percollege)) +
  viridis::scale_color_viridis(discrete = F) # call scale_color_viridis command from the viridis packag
```

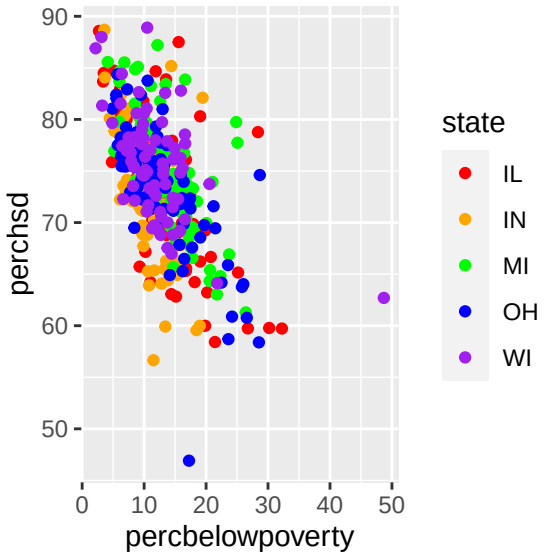


You can also manually specify the colors you want to use as a named vector (typically done for discrete data):

```
#default color palette
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(color = state))
```




```
#the default viridis color palette
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(color = state)) +
  scale_color_manual(values = c("red", "orange", "green", "blue", "purple")) # assign a vector of color
```



Here is a helpful image that I constantly refer to when trying to customize my colors:

brown4	darkorange4	gray	gray57	hotpink3	lightsalmon4	navajowhite1	plum3	slategray3	antiquewhite
brown3	darkorange3	goldenrod4	gray56	hotpink2	lightsalmon3	navajowhite	plum2	slategray2	aliceblue
brown2	darkorange2	goldenrod3	gray55	hotpink1	lightsalmon2	moccasin	plum1	slategray1	white
brown1	darkorange1	goldenrod2	gray54	hotpink	lightsalmon1	mistyrose4	plum	slategray	yellowgreen
brown	darkorange	goldenrod1	gray53	honeydew4	lightsalmon	mistyrose3	pink4	slateblue4	yellow4
blueviolet	darkolivegreen4	goldenrod	gray52	honeydew3	lightpink4	mistyrose2	pink3	slateblue3	yellow3
blue4	darkolivegreen3	gold4	gray51	honeydew2	lightpink3	mistyrose1	pink2	slateblue2	yellow2
blue3	darkolivegreen2	gold3	gray50	honeydew1	lightpink2	mistyrose	pink1	slateblue1	yellow1
blue2	darkolivegreen1	gold2	gray49	honeydew	lightpink1	mintcream	pink	slateblue	yellow
blue1	darkolivegreen	gold1	gray48	greenyellow	lightpink	midnightblue	peru	skyblue4	whitesmoke
blue	darkmagenta	gold	gray47	green4	lightgrey	mediumvioletred	peachpuff4	skyblue3	wheat4
blanchedalmond	darkkhaki	ghostwhite	gray46	green3	lightgreen	mediumturquoise	peachpuff3	skyblue2	wheat3
black	darkgrey	gainsboro	gray45	green2	lightgray	mediumspringgreen	peachpuff2	skyblue1	wheat2
bisque4	darkgreen	forestgreen	gray44	green1	lightgoldenrodyellow	mediumslateblue	peachpuff1	skyblue	wheat1
bisque3	darkgray	floralwhite	gray43	green	lightgoldenrod4	mediumseagreen	peachpuff	sienna4	wheat
bisque2	darkgoldenrod4	firebrick4	gray42	gray100	lightgoldenrod3	mediumpurple4	papayawhip	sienna3	violetred4
bisque1	darkgoldenrod3	firebrick3	gray41	gray99	lightgoldenrod2	mediumpurple3	palevioletred4	sienna2	violetred3
bisque	darkgoldenrod2	firebrick2	gray40	gray98	lightgoldenrod1	mediumpurple2	palevioletred3	sienna1	violetred2
beige	darkgoldenrod1	firebrick1	gray39	gray97	lightgoldenrod	mediumpurple1	palevioletred2	sienna	violetred1
azure4	darkgoldenrod	firebrick	gray38	gray96	lightcyan4	mediumpurple	palevioletred1	seashell4	violetred
azure3	darkcyan	dodgerblue4	gray37	gray95	lightcyan3	mediumorchid4	palevioletred	seashell3	violet
azure2	darkblue	dodgerblue3	gray36	gray94	lightcyan2	mediumorchid3	paleturquoise4	seashell2	turquoise4
azure1	cyan4	dodgerblue2	gray35	gray93	lightcyan1	mediumorchid2	paleturquoise3	seashell1	turquoise3
azure	cyan3	dodgerblue1	gray34	gray92	lightcyan	mediumorchid1	paleturquoise2	seashell	turquoise2
aquamarine4	cyan2	dodgerblue	gray33	gray91	lightcoral	mediumorchid	paleturquoise1	seagreen4	turquoise1
aquamarine3	cyan1	dimgrey	gray32	gray90	lightblue4	mediumblue	paleturquoise	seagreen3	turquoise
aquamarine2	cyan	dimgray	gray31	gray89	lightblue3	mediumaquamarine	palegreen4	seagreen2	tomato4
aquamarine1	cornsilk4	deepskyblue4	gray30	gray88	lightblue2	maroon4	palegreen3	seagreen1	tomato3
aquamarine	cornsilk3	deepskyblue3	gray29	gray87	lightblue1	maroon3	palegreen2	seagreen	tomato2
antiquewhite4	cornsilk2	deepskyblue2	gray28	gray86	lightblue	maroon2	palegreen1	sandybrown	tomato1
antiquewhite3	cornsilk1	deepskyblue1	gray27	gray85	lemonchiffon4	maroon1	palegreen	salmon4	tomato
antiquewhite2	cornsilk	deepskyblue	gray26	gray84	lemonchiffon3	maroon	palegoldenrod	salmon3	thistle4
antiquewhite1	cornflowerblue	deeppink4	gray25	gray83	lemonchiffon2	magenta4	orchid4	salmon2	thistle3
antiquewhite	coral4	deeppink3	gray24	gray82	lemonchiffon1	magenta3	orchid3	salmon1	thistle2
aliceblue	coral3	deeppink2	gray23	gray81	lemonchiffon	magenta2	orchid2	salmon	thistle1
white	coral2	deeppink1	gray22	gray80	lawngreen	magenta1	orchid1	saddlebrown	thistle
bisque3	coral1	deeppink	gray21	gray79	lavenderblush4	magenta	orchid	royalblue4	tan4
bisque2	coral	darkviolet	gray20	gray78	lavenderblush3	linen	orangered4	royalblue3	tan3
bisque1	chocolate4	darkturquoise	gray19	gray77	lavenderblush2	limegreen	orangered3	royalblue2	tan2
bisque	chocolate3	darkslategrey	gray18	gray76	lavenderblush1	lightyellow4	orangered2	royalblue1	tan1
beige	chocolate2	darkslategray4	gray17	gray75	lavenderblush	lightyellow3	orangered1	royalblue	tan
azure4	chocolate1	darkslategray3	gray16	gray74	lavender	lightyellow2	orangered	rosybrown4	steelblue4
azure3	chocolate	darkslategray2	gray15	gray73	khaki4	lightyellow1	orange4	rosybrown3	steelblue3
azure2	chartreuse4	darkslategray1	gray14	gray72	khaki3	lightyellow	orange3	rosybrown2	steelblue2
azure1	chartreuse3	darkslategray	gray13	gray71	khaki2	lightsteelblue4	orange2	rosybrown1	steelblue1
azure	chartreuse2	darkslateblue	gray12	gray70	khaki1	lightsteelblue3	orange1	rosybrown	steelblue
aquamarine4	chartreuse1	darkseagreen4	gray11	gray69	khaki	lightsteelblue2	orange	red4	springgreen4
aquamarine3	chartreuse	darkseagreen3	gray10	gray68	ivory4	lightsteelblue1	olivedrab4	red3	springgreen3
aquamarine2	cadetblue4	darkseagreen2	gray9	gray67	ivory3	lightsteelblue	olivedrab3	red2	springgreen2
aquamarine1	cadetblue3	darkseagreen1	gray8	gray66	ivory2	lightslategrey	olivedrab2	red1	springgreen1
aquamarine	cadetblue2	darkseagreen	gray7	gray65	ivory1	lightslategray	olivedrab1	red	springgreen
antiquewhite4	cadetblue1	darksalmon	gray6	gray64	ivory	lightslateblue	olivedrab	purple4	snow4
antiquewhite3	cadetblue	darkred	gray5	gray63	indianred4	lightskyblue4	oldlace	purple3	snow3
antiquewhite2	burlywood4	darkorchid4	gray4	gray62	indianred3	lightskyblue3	navyblue	purple2	snow2
antiquewhite1	burlywood3	darkorchid3	gray3	gray61	indianred2	lightskyblue2	navy	purple1	snow1
antiquewhite	burlywood2	darkorchid2	gray2	gray60	indianred1	lightskyblue1	navajowhite4	purple	snow
aliceblue	burlywood1	darkorchid1	gray1	gray59	indianred	lightskyblue	navajowhite3	powderblue	slategrey
white	burlywood	darkorchid	gray0	gray58	hotpink4	lightseagreen	navajowhite2	plum4	slategray4

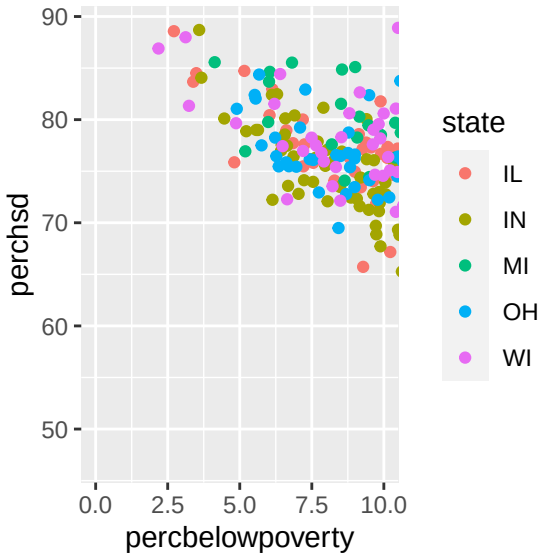
Coordinate systems The next term from the Grammar of Graphics that can be specified is the coordinate system. As with scales, coordinate systems are specified with functions that all start with coord_ and are added as a layer. There are a number of different possible coordinate systems to use, including:

- `coord_cartesian` the default cartesian coordinate system, where you specify x and y values (e.g. allows you to zoom in or out).
- `coord_flip` a cartesian system with the x and y flipped
- `coord_fixed` a cartesian system with a "fixed" aspect ratio (e.g., 1.78 for a "widescreen" plot)
- `coord_polar` a plot using polar coordinates
- `coord_quickmap` a coordinate system that approximates a good aspect ratio for maps. See documentation for more details.

Some examples:

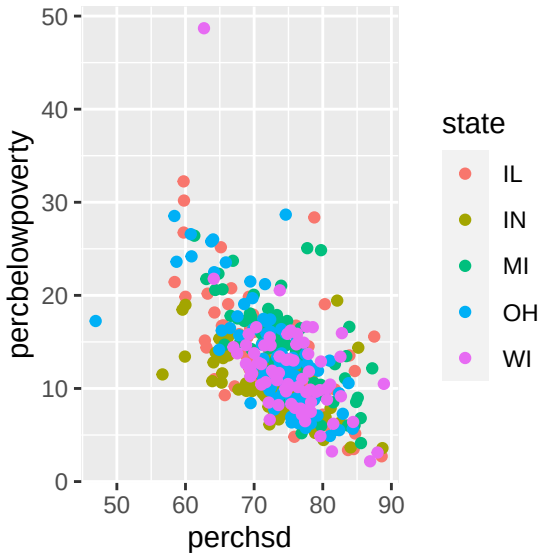
Zoom in with `coord_cartesian`

```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(color = state)) +
  coord_cartesian(xlim = c(0, 10)) # only show the x axis from value of 0 to value of 10 (but all point
```



Flip x and y axis with `coord_flip`

```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(color = state)) +
  coord_flip()
```



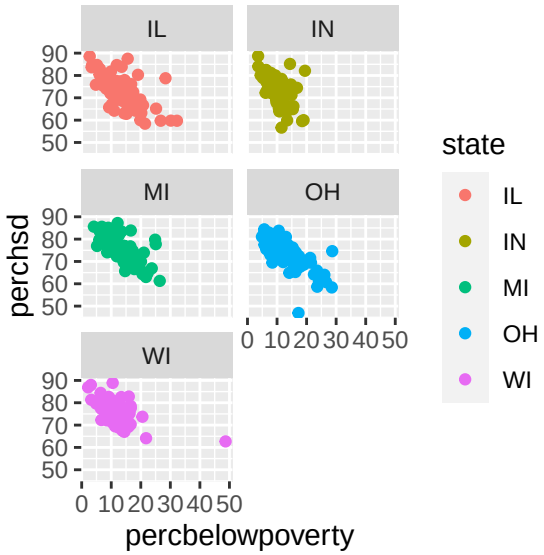
Facets Facets are ways of grouping a data plot into multiple different pieces (subplots). This allows you to view a separate plot for each value in a categorical variable.

Similar to scales and coordinate systems, different types of faceting are available and following the naming patter facet_ followed by the type of facet.

Within the argument of the facet_ function, you need to specify what variable to facet around with the tilde ~ followed by the variable name.

One of the possible functions to facet data is the `facet_wrap()` function. This will produce rows of subplots, one subplot for each categorical variable (the number of rows can be specified with the `nrow` argument:

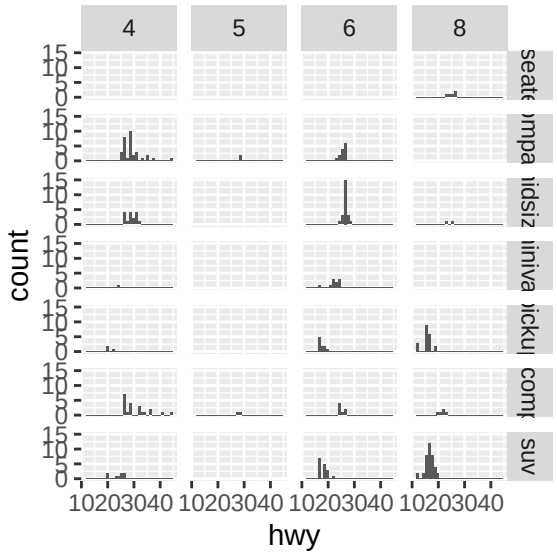
```
ggplot(data = midwest, aes(x = percbelowpoverty, y = perchsd)) +
  geom_point(aes(color = state)) +
  facet_wrap(~state, nrow = 3)
```



You can use `facet_grid` to facet your data into a grid rather than a row. This is most common when you want to facet by more than one categorical variable. With `facet_grid` the variable to the left of the tilde ("~") will be represented in the rows and the variable to the right will be represented across the columns.

```
ggplot(mpg) +
  geom_histogram(aes(x = hwy)) +
  facet_grid(class ~ cyl)
```

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

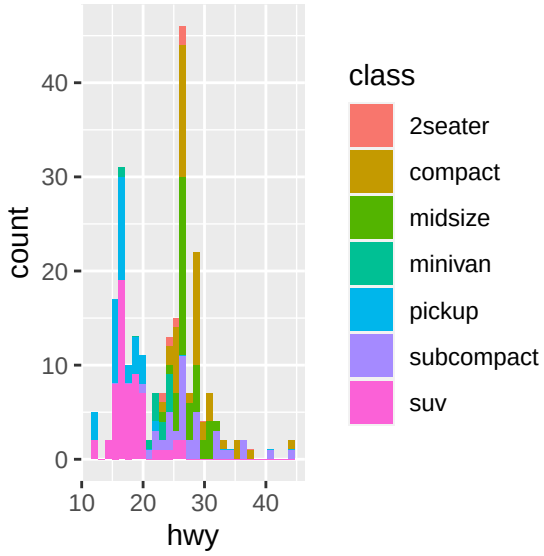


Position Position The position argument is used to tweak how certain aspects of the plots are displayed. Its use depends heavily on the type of plot. For each geom, some positions will work, some will do nothing, and some will produce nonsense. They are most commonly used when trying to create grouped plots.

For example, we can look at a histogram of car mpg (hwy).

```
ggplot(mpg, aes(x = hwy, fill = class)) +
  geom_histogram()
```

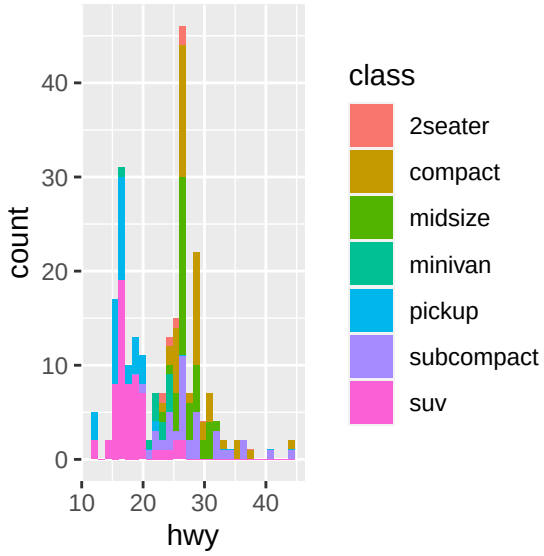
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



If we add `fill = class`, it will group by class. The default is `position = "stack"`; let's see what each does.

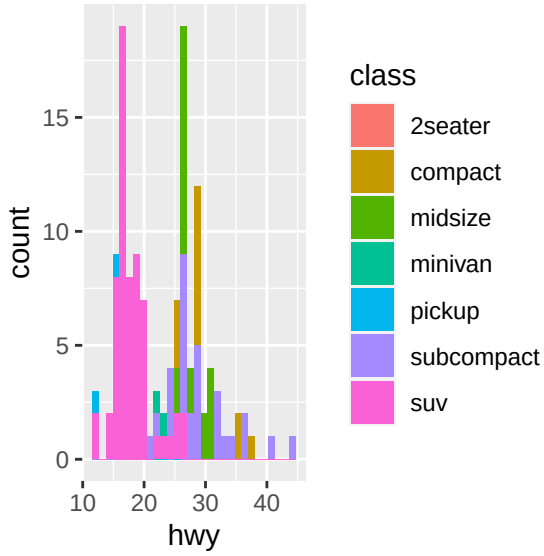
```
ggplot(mpg, aes(x = hwy, fill = class)) +
  geom_histogram(position = "stack")
```

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



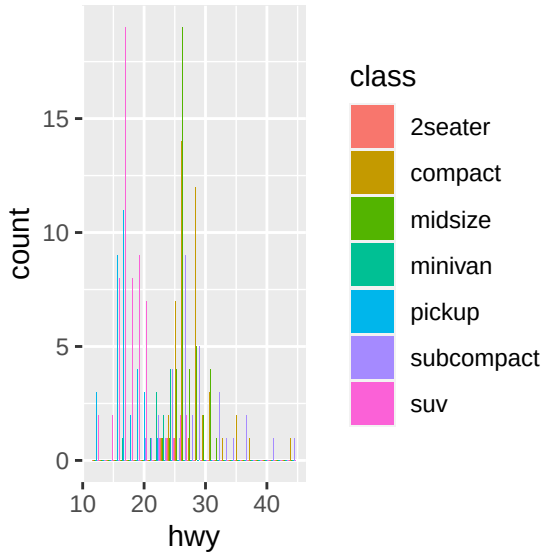

```
ggplot(mpg, aes(x = hwy, fill = class)) +
  geom_histogram(position = "identity")
```

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




```
ggplot(mpg, aes(x = hwy, fill = class)) +
  geom_histogram(position = "dodge")
```

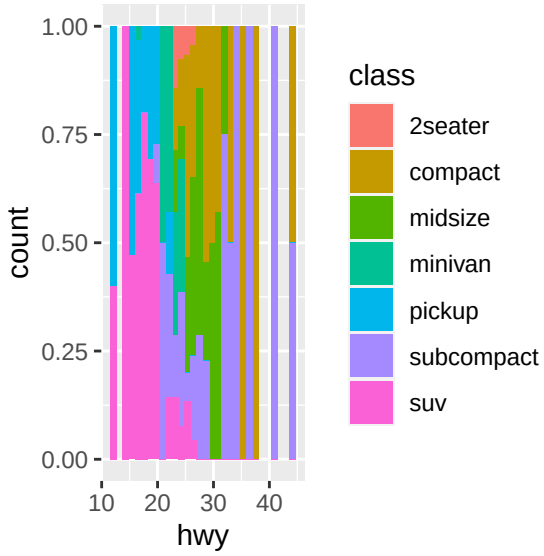
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ggplot(mpg, aes(x = hwy, fill = class)) +
  geom_histogram(position = "fill")
```

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

```
## Warning: Removed 35 rows containing missing values (`geom_bar()`).
```



In general:

- identity is useful when you want to plot things exactly as they are.
- stack is useful when you want to look both at overall values and per-group values.
- dodge is useful for group comparisons.
- fill is useful for considering percentages instead of counts.
- jitter is useful for scatter plots (and similar) when multiple values may be placed at the same point.

Positions are generally a trial-and-error procedure. If the default isn't sufficient, see if the others look/work better.

Stat We can work through this another day if you're interested!