Graphics(ggplot2)

Canhong Wen

The `ggplot2` package

- produces layered statistical graphics.
- uses an underlying "grammar" to build graphs component-by-component rather than providing premade graphs.
- is easy enough to use without any exposure to the underlying grammar, but even easier to use once you know the grammar.
- http://ggplot2.org
- Find more at https://www.r-graph-gallery.com/ggplot2-package.html

Installing `ggplot2`

1. Directly

```
install.packages("ggplot2")
```

2. From `tidyverse`

```
install.packages("tidyverse")
```

- `tidyverse` is a suite of packages consists of many useful packages such as
  - `dplyr`: data management (e.g. filtering, selecting, sorting, processing by group)
  - `tidyr`: tidying/restructuring data'
  - `haven`: reading data in other formats (Stata, SAS, SPSS)
  - `stringr`: string variable processing

See more in "R for Data Science" and "ggplot2: Elegant Graphics for Data Analysis" by Hadley Wickham.

Grammar of graphics in `ggplot2`

1. **Data**: variables mapped to aesthetic features of the graph.
2. **Layers**: (1). **Geoms**: objects/shapes on the graph. (2). **Stats**: stastical transformations that summarize data,(e.g mean, confidence intervals)
3. **Scales**: mappings of aesthetic values to data values. Legends and axes display these mappings.
4. **Coordiante systems**: the plane on which data are mapped on the graphic.
5. **Faceting**: splitting the data into subsets to create multiple variations of the same graph (paneling).
6. **Theme**: control the finers points of display like the font size and background color properties.

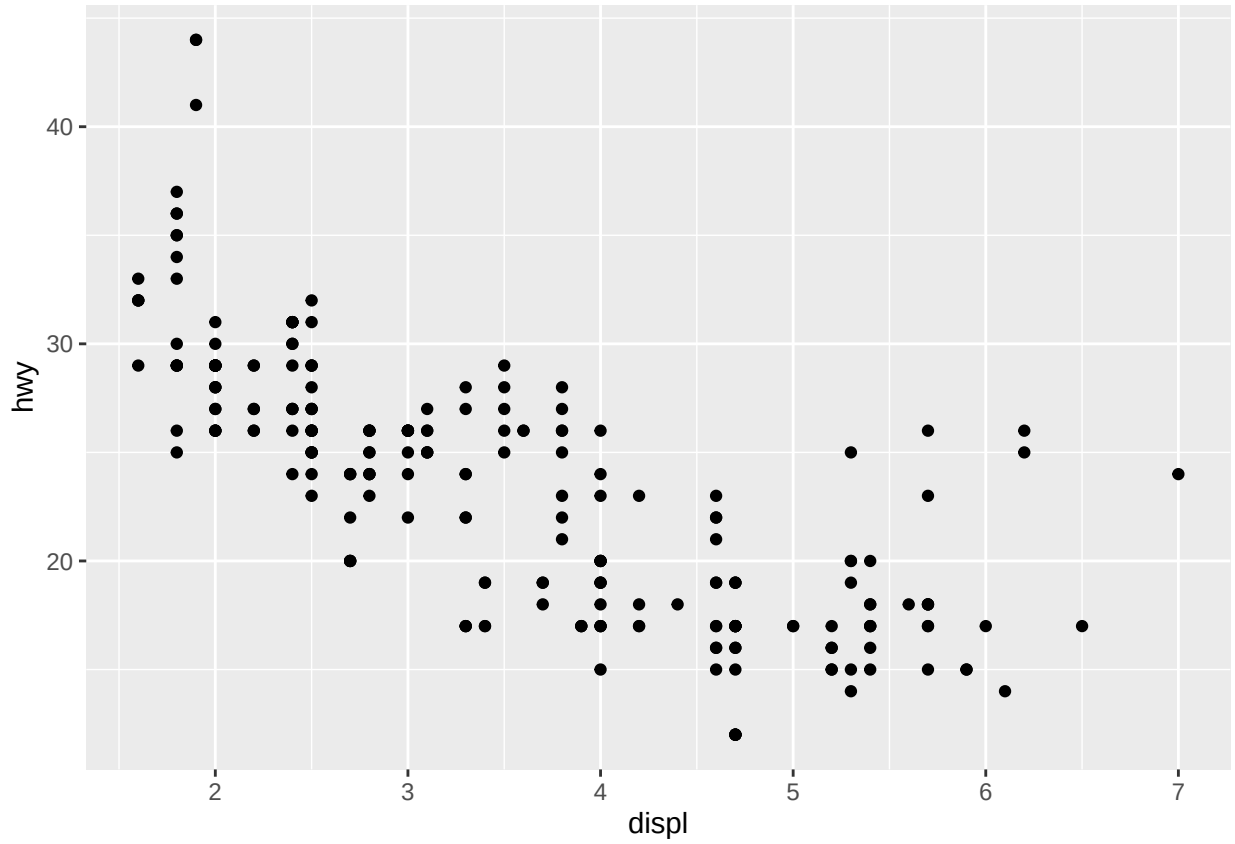
My first plot

```
library(ggplot2)
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(~ f        18    29 p     comp~
## 2 audi         a4      1.8  1999     4 manua~ f        21    29 p     comp~
```

```
## 3 audi         a4      2    2008     4 manua~ f        20    31 p     comp~
## 4 audi         a4      2    2008     4 auto(~ f        21    30 p     comp~
## 5 audi         a4      2.8  1999     6 auto(~ f        16    26 p     comp~
## 6 audi         a4      2.8  1999     6 manua~ f        18    26 p     comp~
```

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



In the above `ggplot()` function:

- data set is `mpg`
- Inside of the function `aes()`
  - `displ` mapped to x-axis
  - `hwy` mapped to y-axis
- `geom_point()`: scatterplot

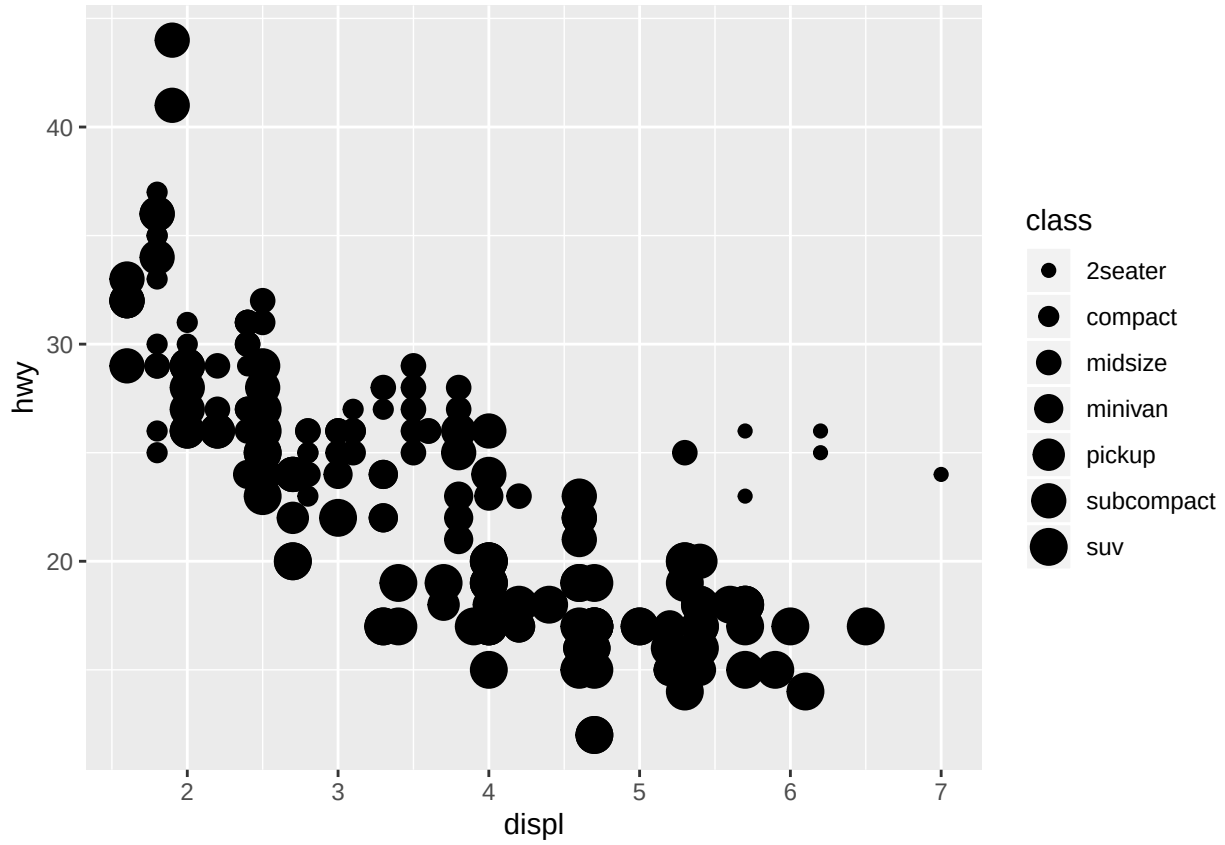
Data

Data

- Data for plotting with `ggplot2` tools must be stored in a `data.frame`.
- Map variables to aesthetics using the `aes()` function. For example,
  - `x`: positioning along x-axis
  - `y`: positioning along y-axis
  - `color`: color of objects; for 2-d objects, the color of the object's outline (compare to fill below)
  - `fill`: fill color of objects
  - `alpha`: transparency of objects (value between 0, transparent, and 1, opaque – inverse of how many stacked objects it will take to be opaque)
  - `linetype`: how lines should be drawn (`solid`, `dashed`, `dotted`, etc.)
  - `shape`: shape of markers in scatter plots
  - `size`: how large objects appear

```
ggplot(data = mpg, aes(x = displ, y = hwy, size = class)) + geom_point()
```

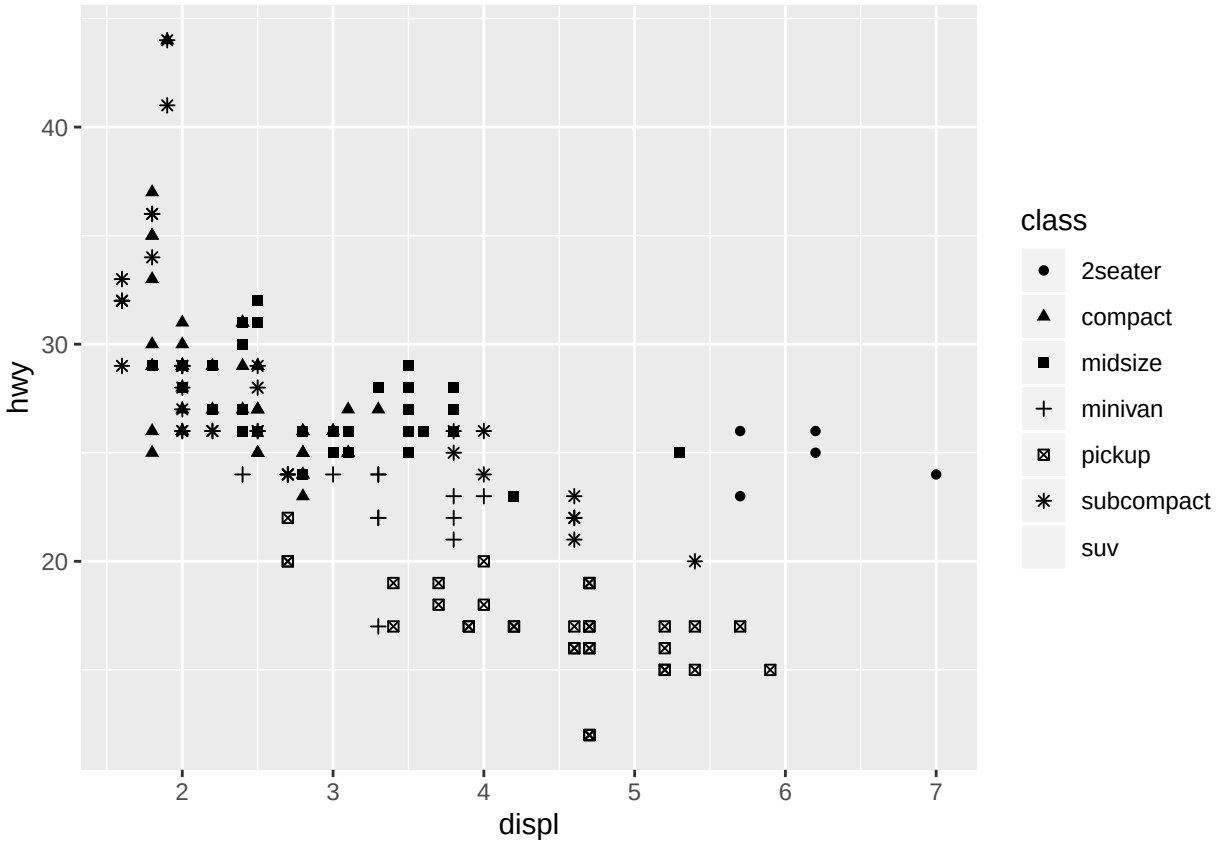
```
## Warning: Using size for a discrete variable is not advised.
```



```
ggplot(data = mpg, aes(x = displ, y = hwy, shape = class)) + geom_point()
```

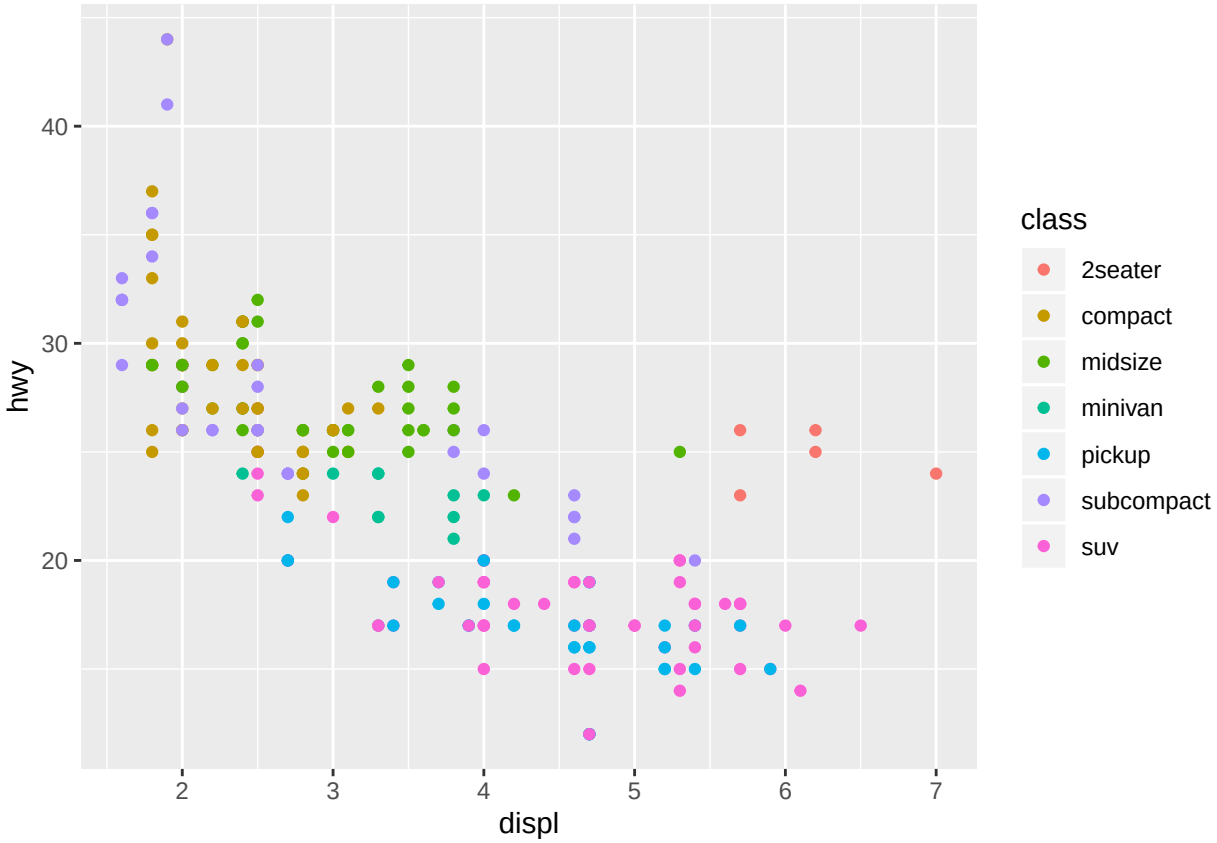
```
## Warning: The shape palette can deal with a maximum of 6 discrete values
## because more than 6 becomes difficult to discriminate; you have 7.
## Consider specifying shapes manually if you must have them.
```

```
## Warning: Removed 62 rows containing missing values (geom_point).
```



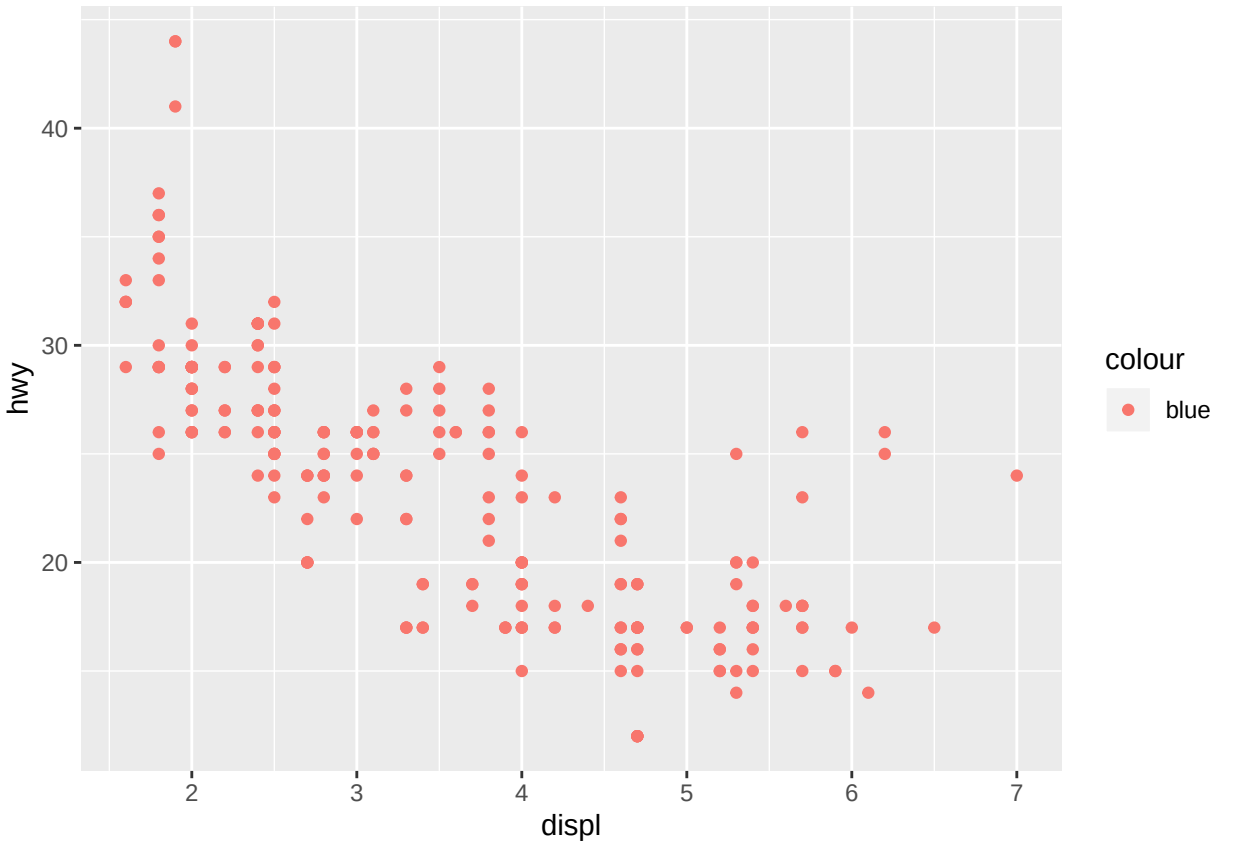
##

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



Whta's wrong?

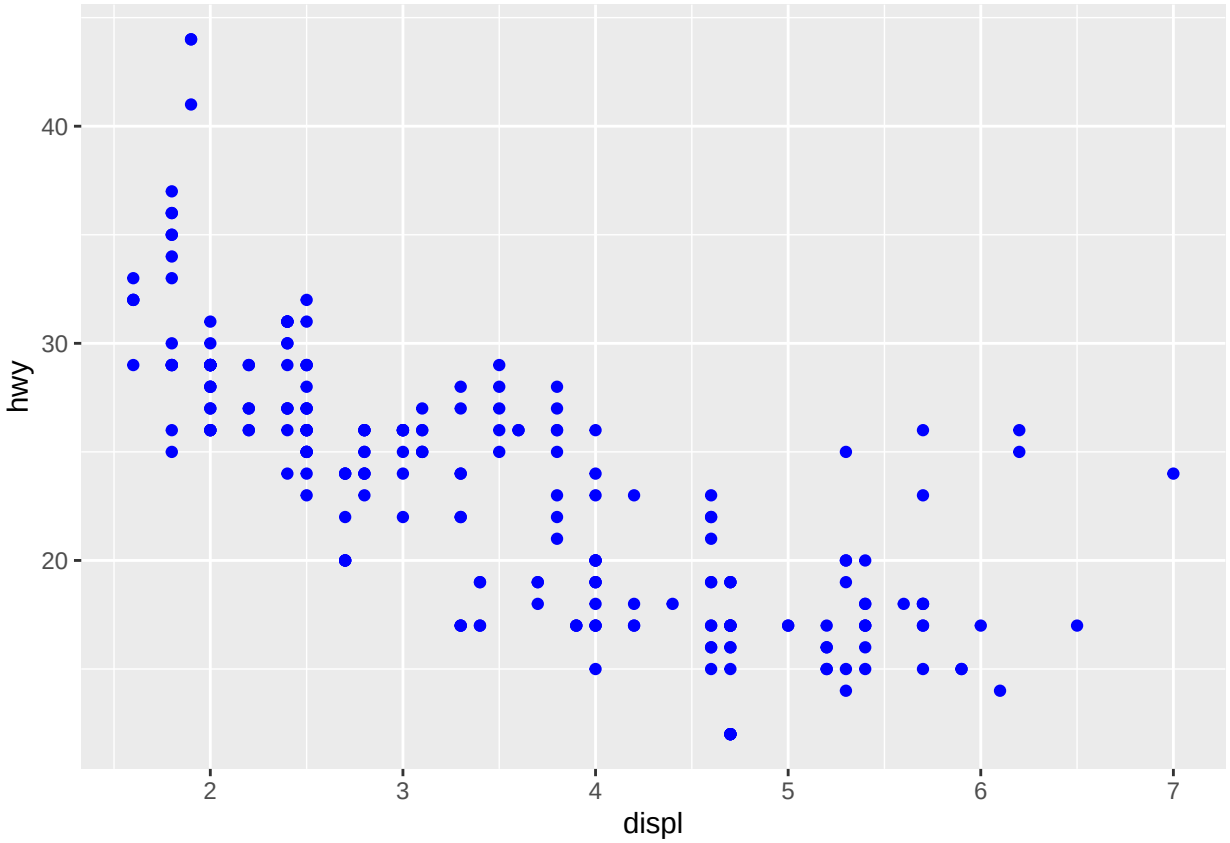
```
ggplot(data = mpg, aes(x = displ, y = hwy, color = "blue")) + geom_point()
```



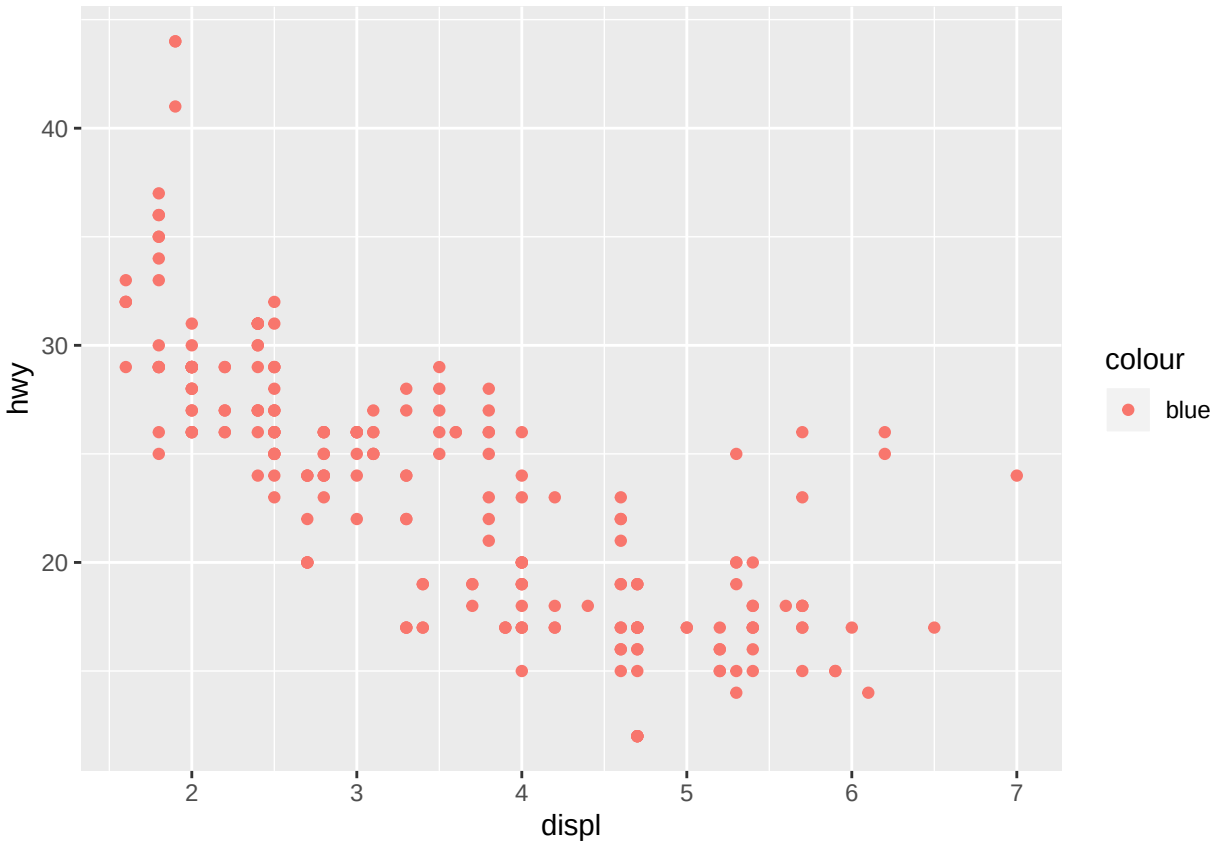
`Mapping` vs setting

- *Map* aesthetics to variables inside the `aes()` function. When mapped, the aesthetic will vary as the variable varies.
- *Set* aesthetics to a constant outside the `aes()` function.
- Setting an aesthetic to a constant within `aes()` can lead to unexpected results, as the aesthetic is then set to a default value rather than the specified value.

```
ggplot(data = mpg, aes(x = displ, y = hwy)) + geom_point(color = "blue")
```

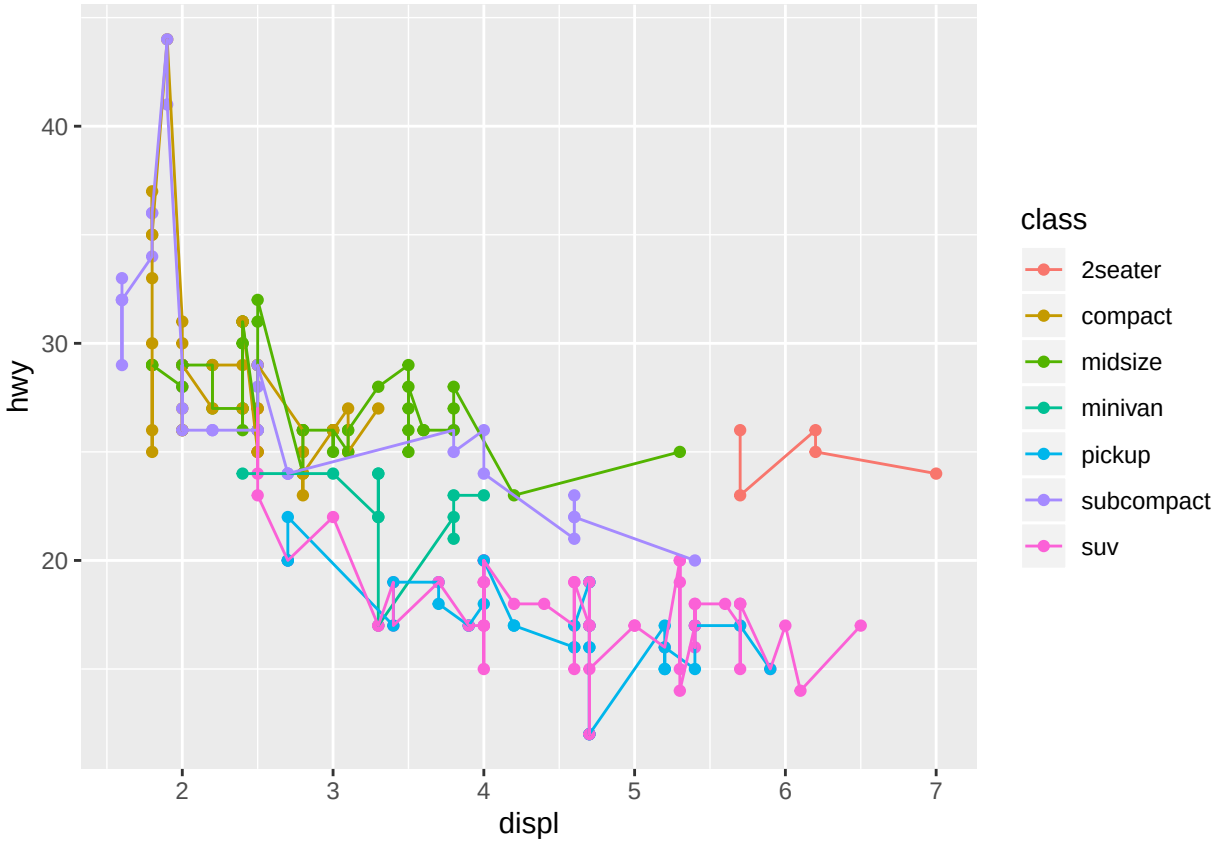



```
ggplot(data = mpg, aes(x = displ, y = hwy)) + geom_point(aes(color = "blue"))
```

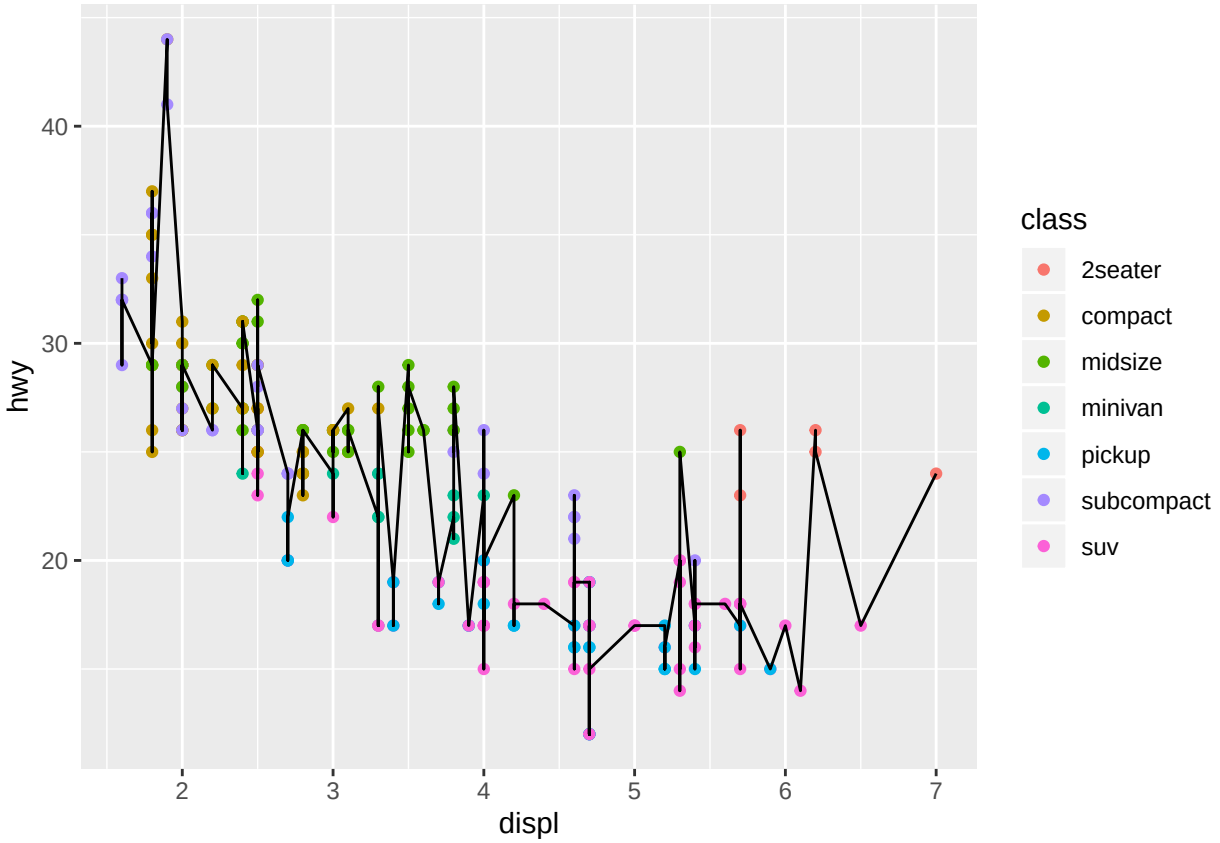


The position of `aes()`

```
ggplot(data = mpg, aes(x = displ, y = hwy,color = class)) +
  geom_point() + geom_line()
```




```
ggplot(data = mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) + geom_line()
```

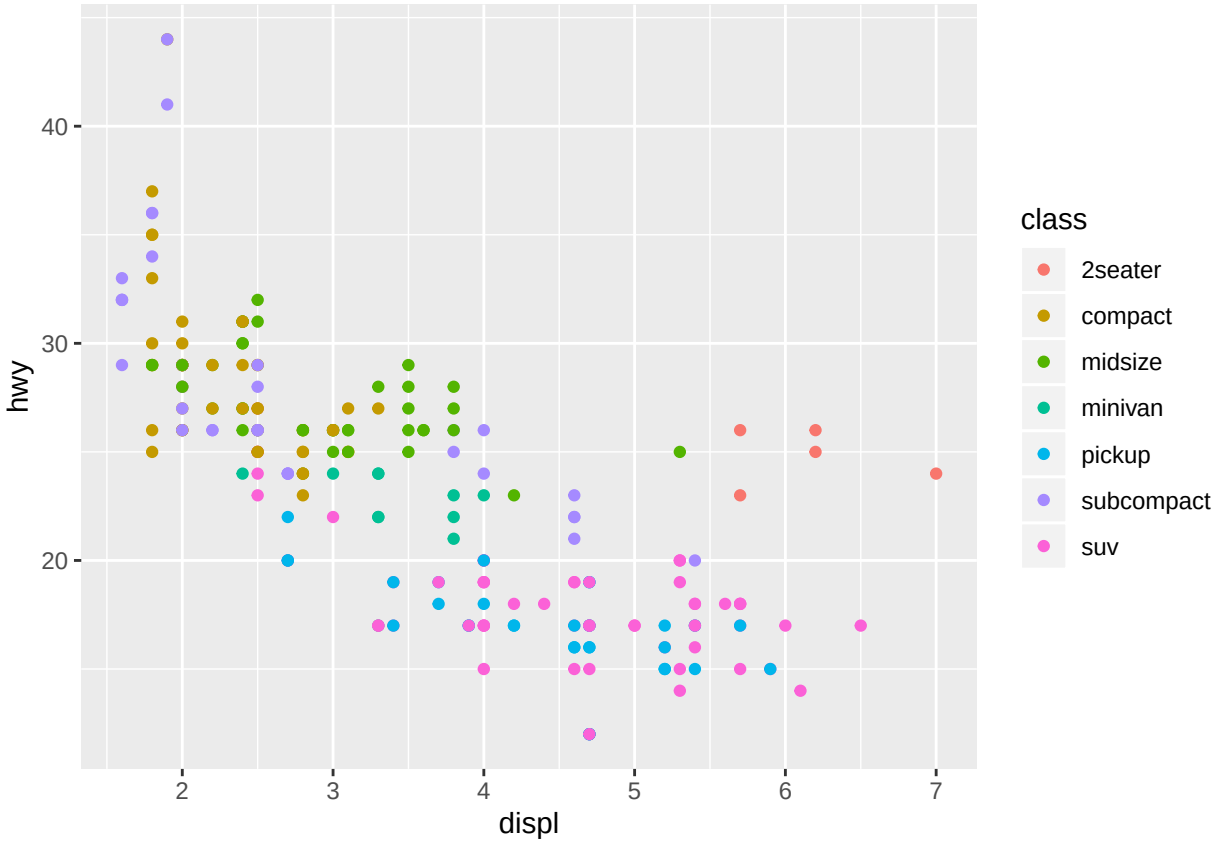


Layers

Layers

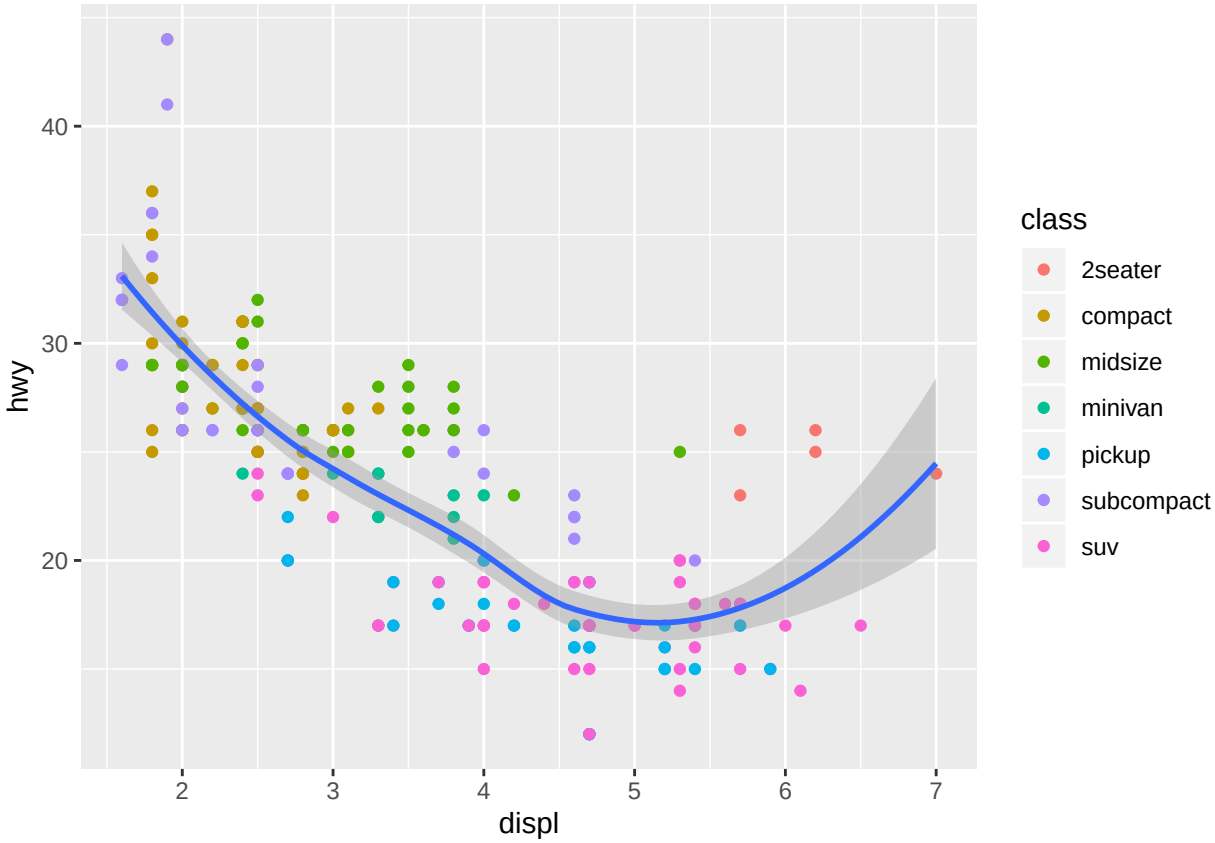
- Graphs in `ggplot2` are built layer-by-layer, rather than being premade. As we have already seen, we add more layers with the character `+`.
- A layer consists of graphics produced by either a `geom` or `stat` element. We can add layers in a virtually unrestricted way, allowing us to customize our graphs as we see fit.
- Remember that layers inherit the aesthetics from the `ggplot()` function, but they can be respecified for each layer.
- We can store ggplot specifications in R objects and then reuse the object to keep the code compact. When we add layers to a plot stored in an object, the graphic is rendered, with the layers inheriting aesthetics from the original ggplot.

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

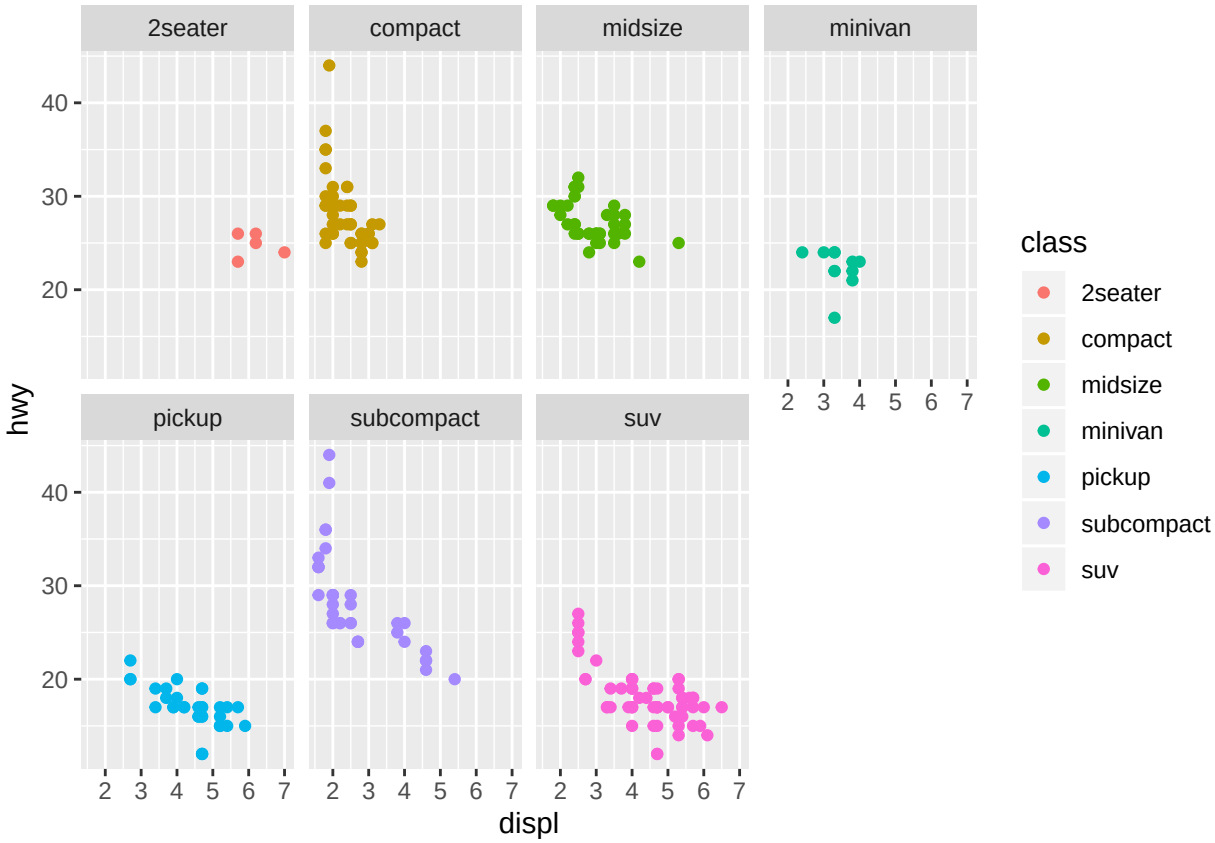



```
p + geom_smooth() #add a loess plot layer
```

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




```
p + facet_wrap(~ class, nrow = 2) # panel by class
```



Geoms

`Geom` functions differ in the geometric shapes produced for the plot:

- `geom_bar()`: bars with bases on the x-axis
- `geom_boxplot()`: boxes-and-whiskers
- `geom_errorbar()`: T-shaped error bars
- `geom_histogram()`: histogram
- `geom_line()`: lines
- `geom_point()`: points (scatterplot)
- `geom_ribbon()`: bands spanning y-values across a range of x-values
- `geom_smooth()`: smoothed conditional means (e.g. loess smooth)

Univariate

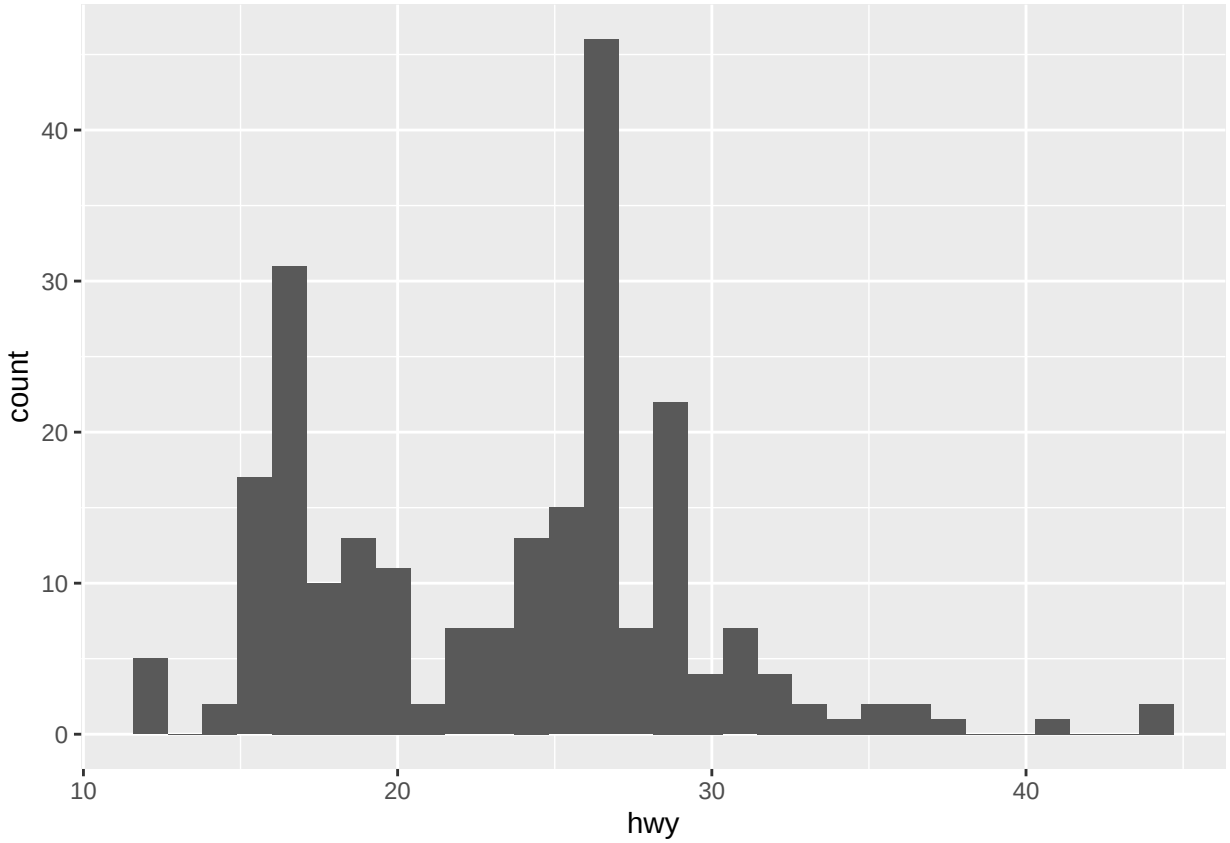
Geoms that require only the `x` aesthetic be specified are often used to plot the distribution of a single, continuous variable.

```
pro <- ggplot(data = mpg, aes(x = hwy))
```

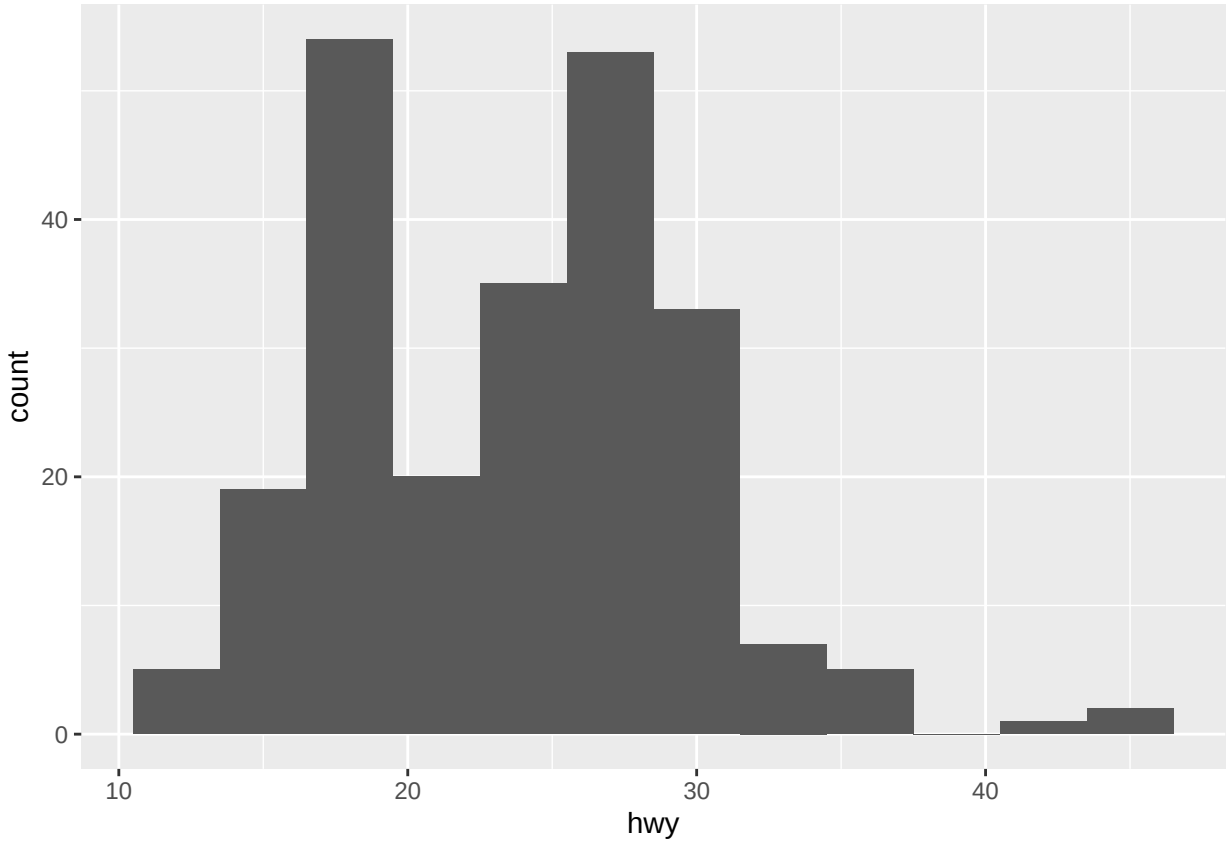
Histograms plot

```
pro + geom_histogram()
```

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

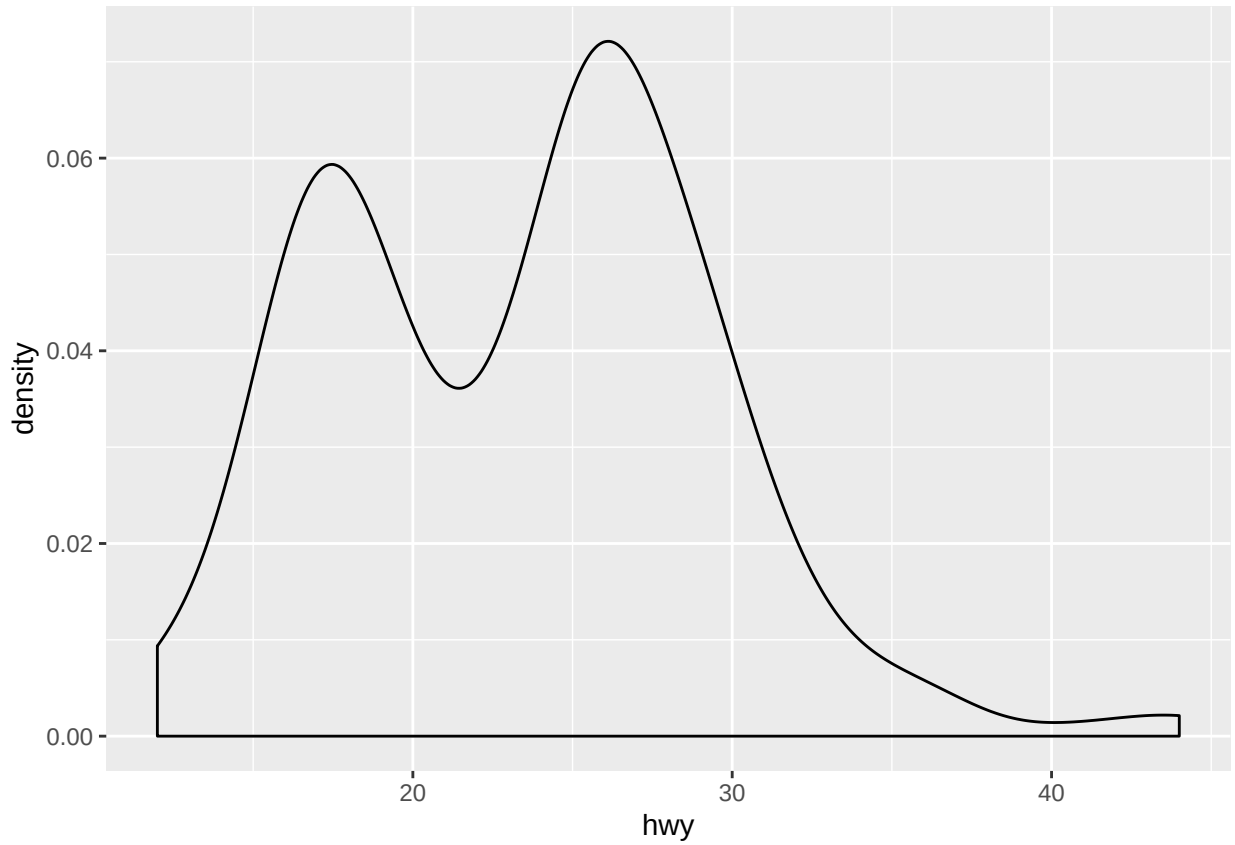



```
pro + geom_histogram(binwidth = 3)
```



Density plot

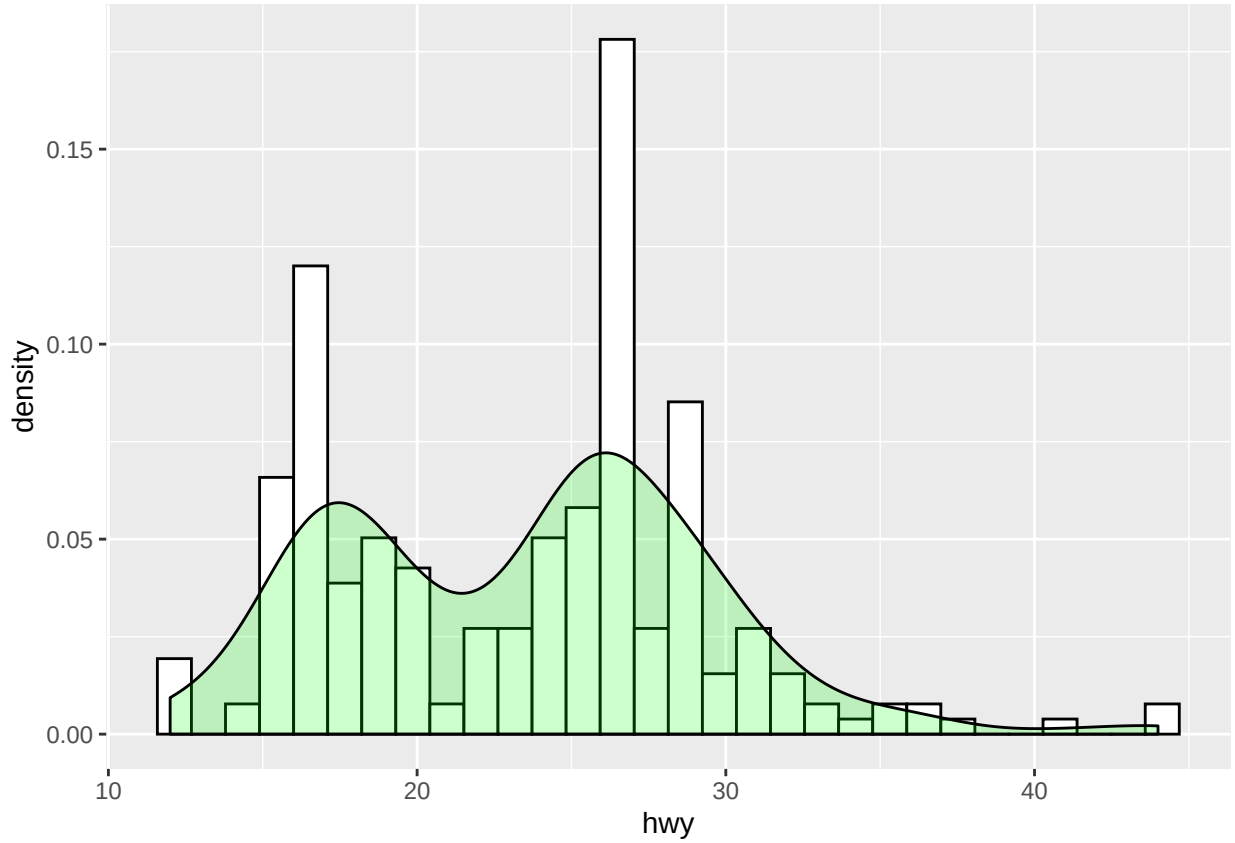
```
pro + geom_density()
```



Histogram overlaid with kde

```
pro + geom_histogram(aes(y=..density..), color = "black", fill="white") +
  geom_density(alpha=0.2, fill="green")
```

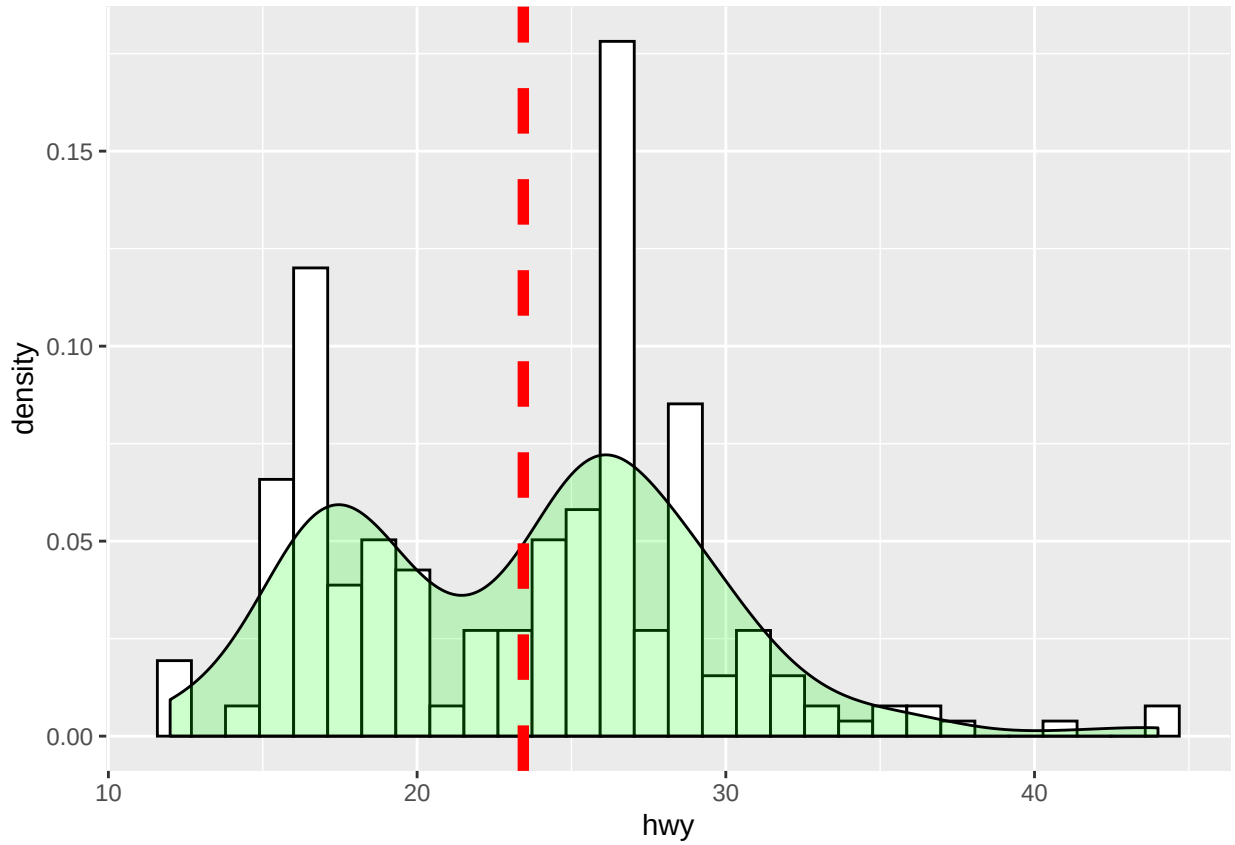
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



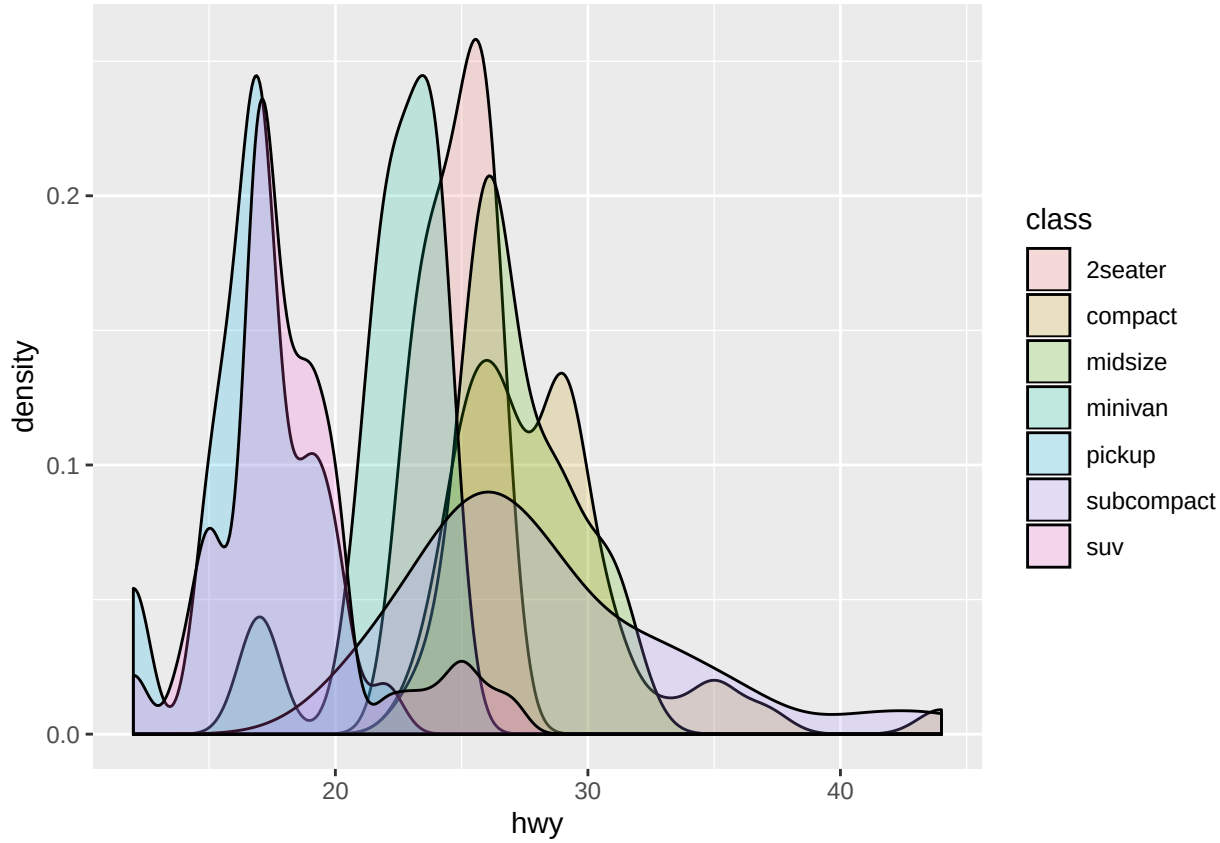
Add a line to histogram

```
pro + geom_histogram(aes(y=..density..), color = "black", fill="white") +
  geom_density(alpha=0.2, fill="green")+
  geom_vline(aes(xintercept=mean(hwy)), color="red", size=2, linetype="dashed")
```

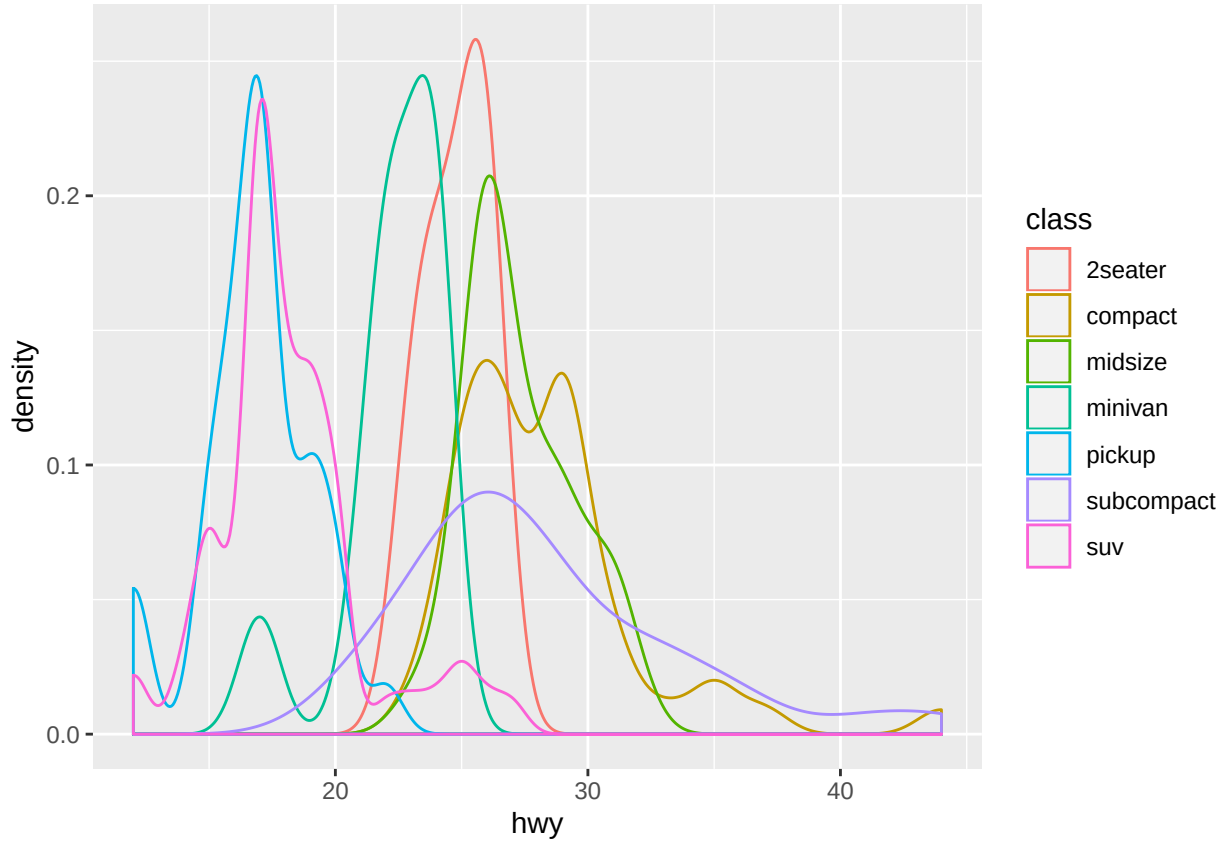
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
pro + geom_density(aes(fill=class), alpha=0.2)
```

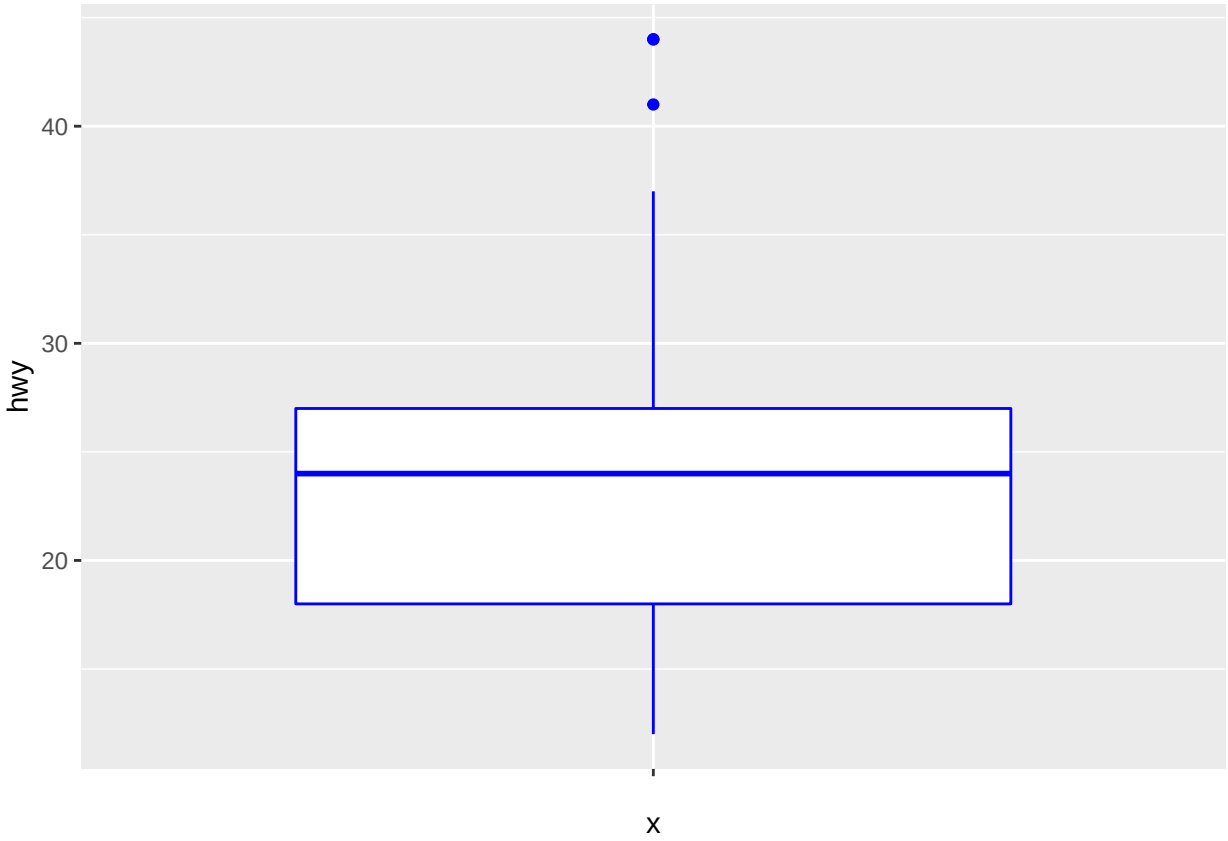



```
pro + geom_density(aes(color=class), alpha=0.2)
```



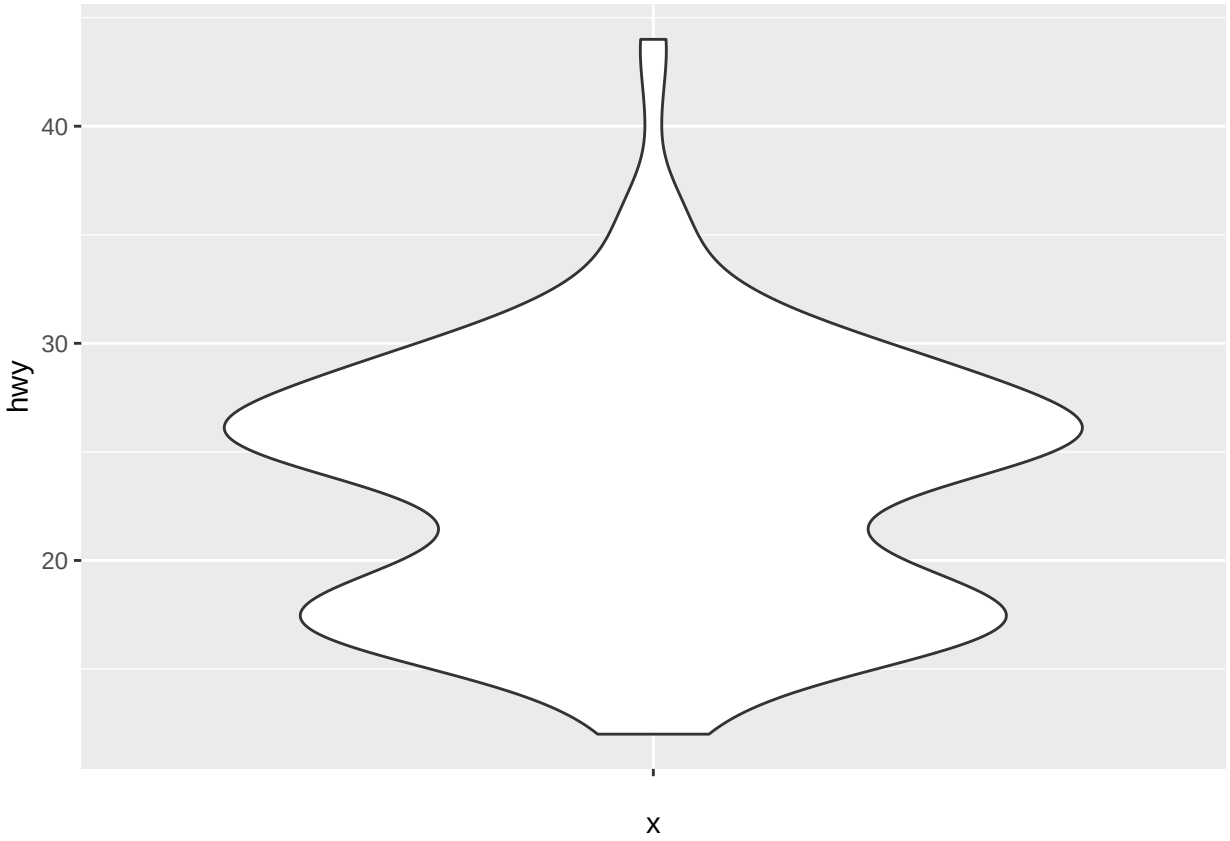
boxplot

```
ggplot(mpg, aes(x="",y=hwy)) +  geom_boxplot(color="blue")
```



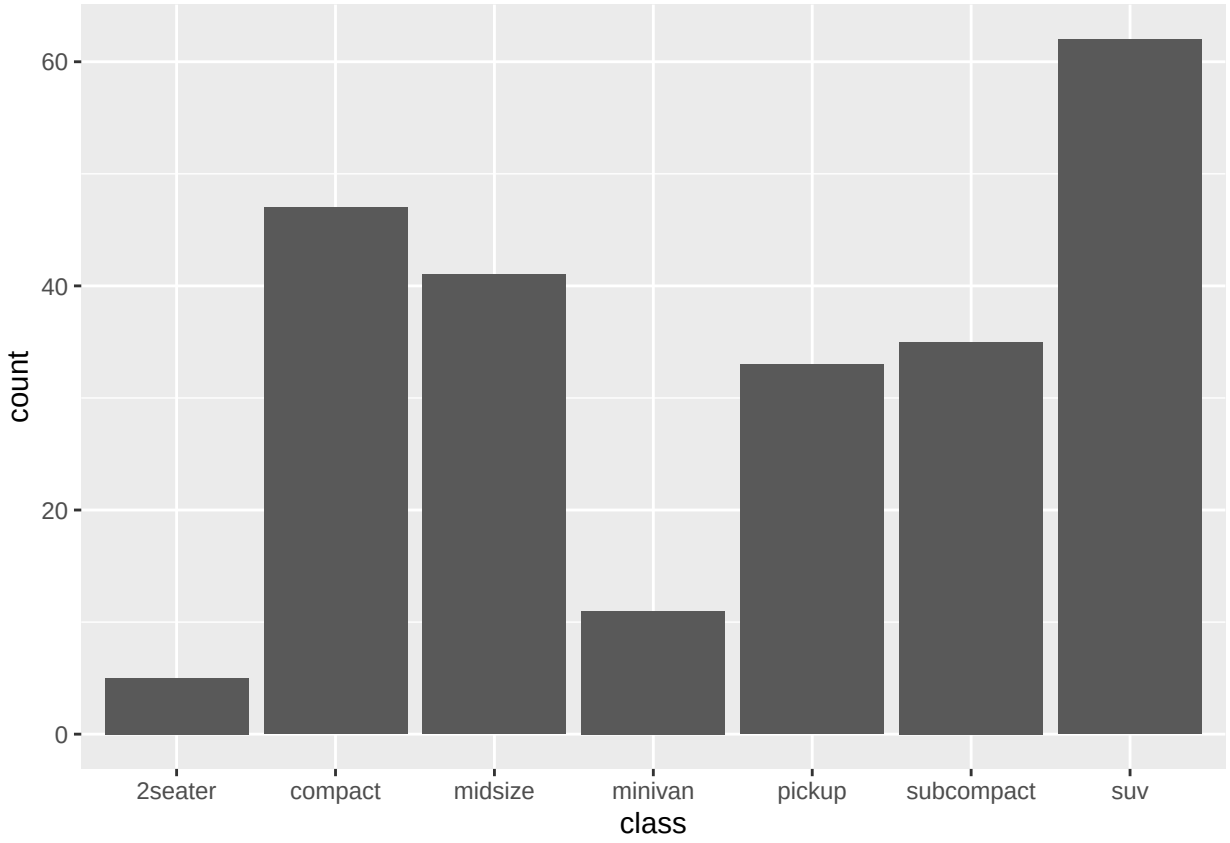
violin plot

```
ggplot(mpg, aes(x="",y=hwy)) +  geom_violin()
```

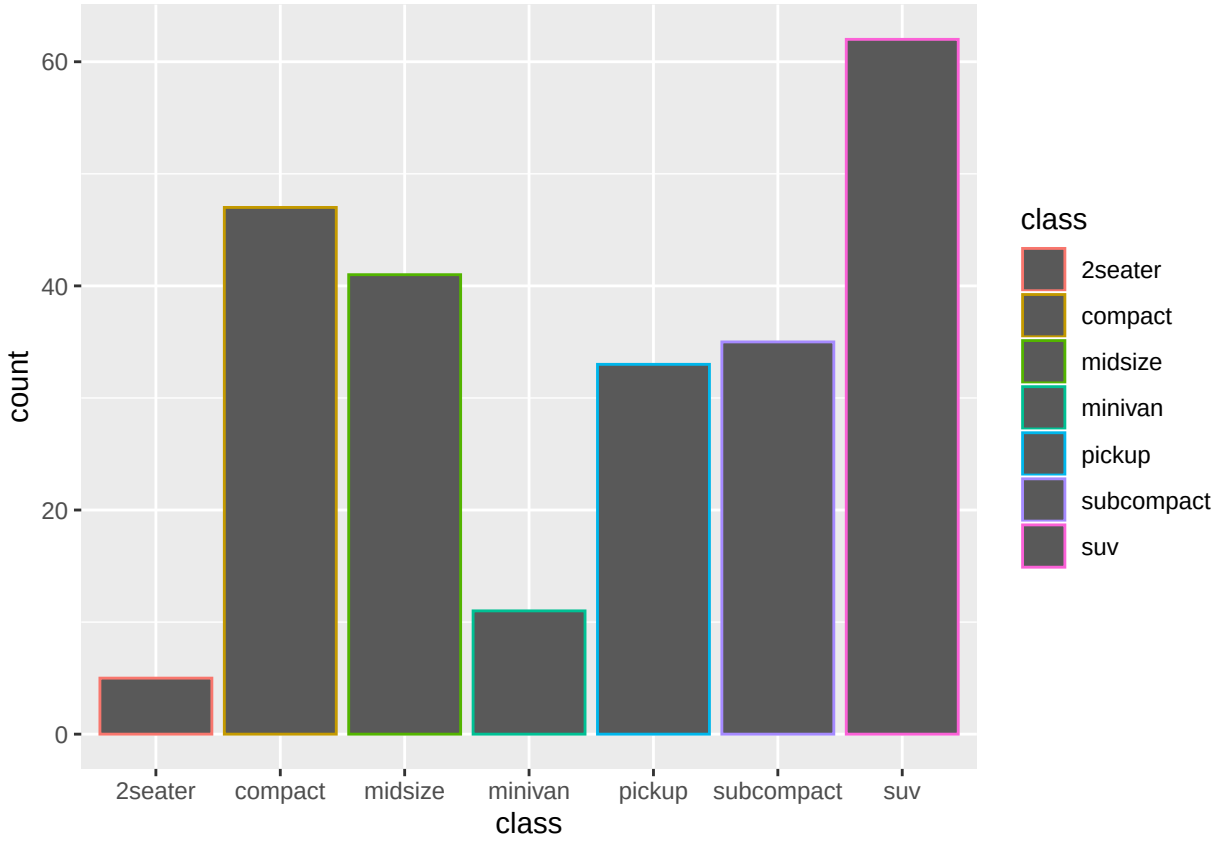


barplot

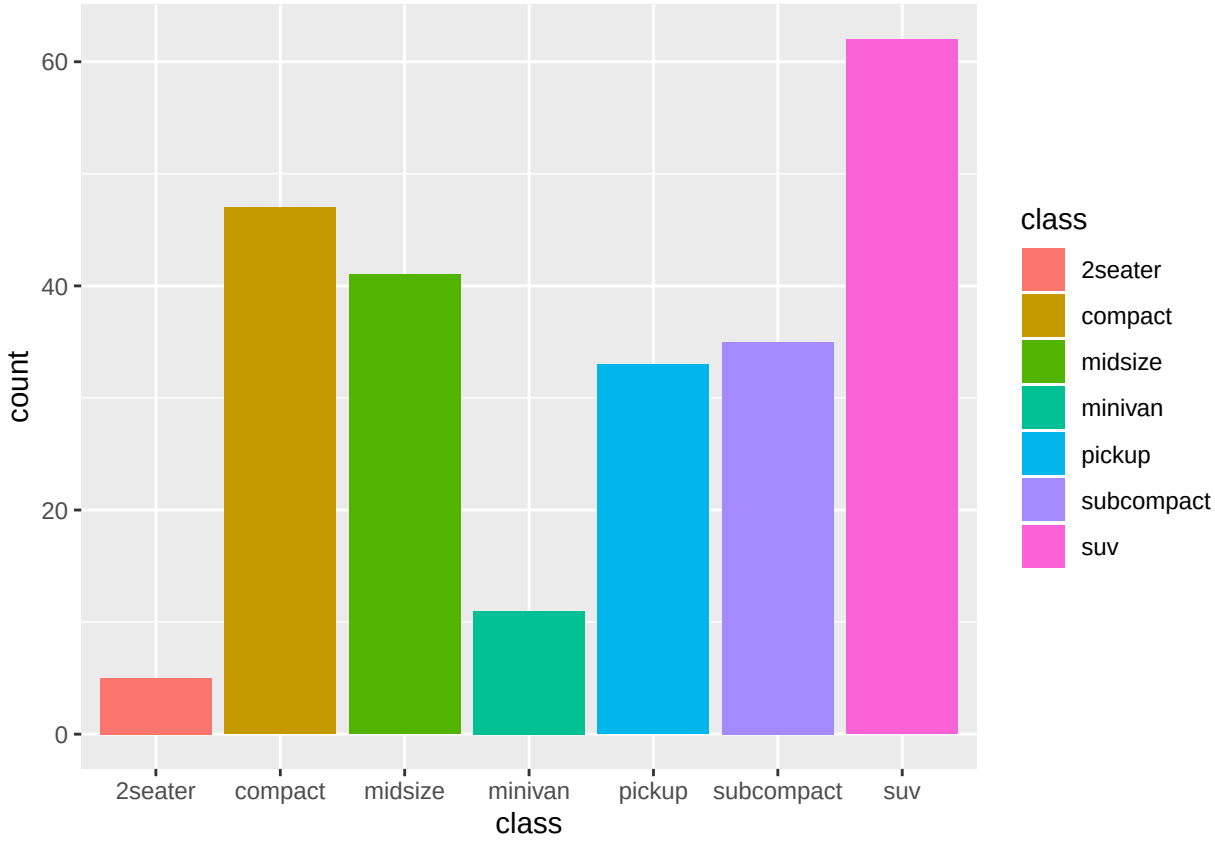
```
p <- ggplot(data = mpg, aes(x = class))
p + geom_bar()
```



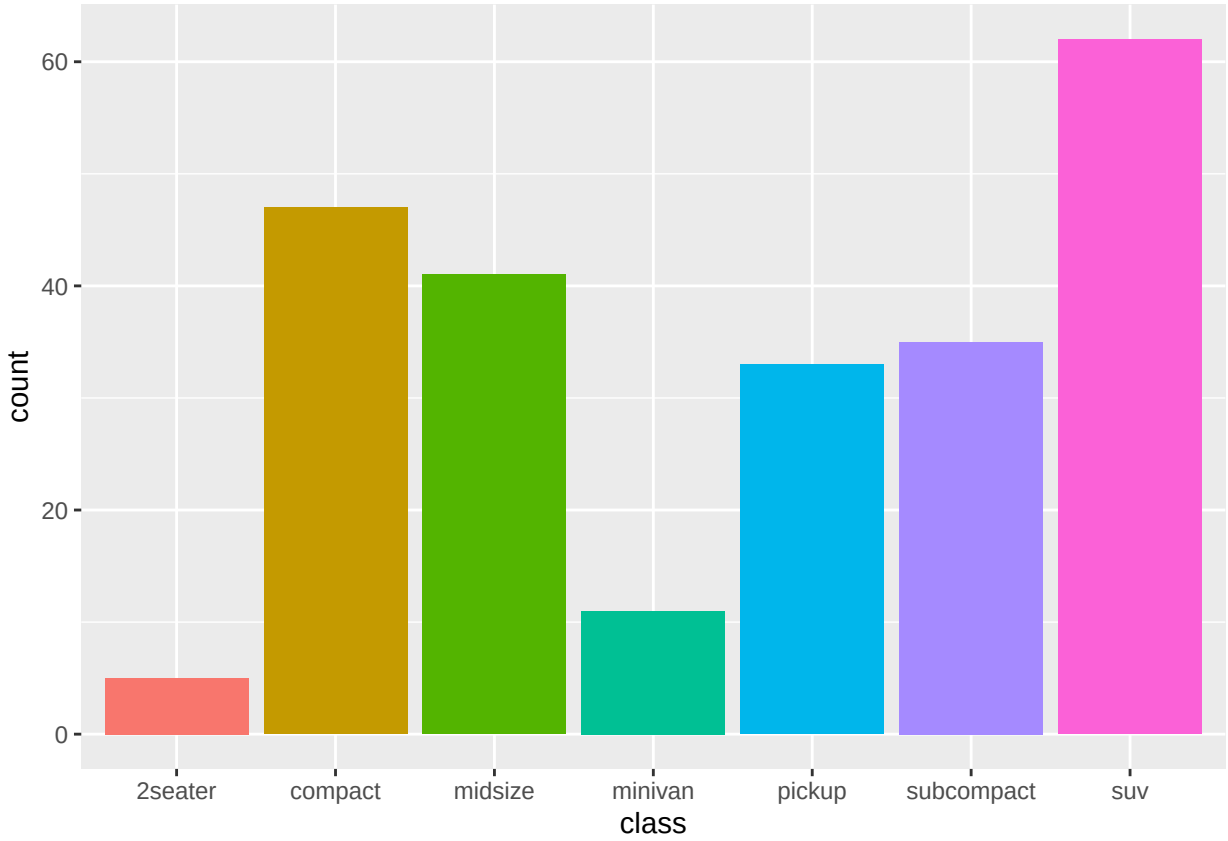

```
p + geom_bar(aes(color = class))
```




```
p + geom_bar(aes(fill = class))
```

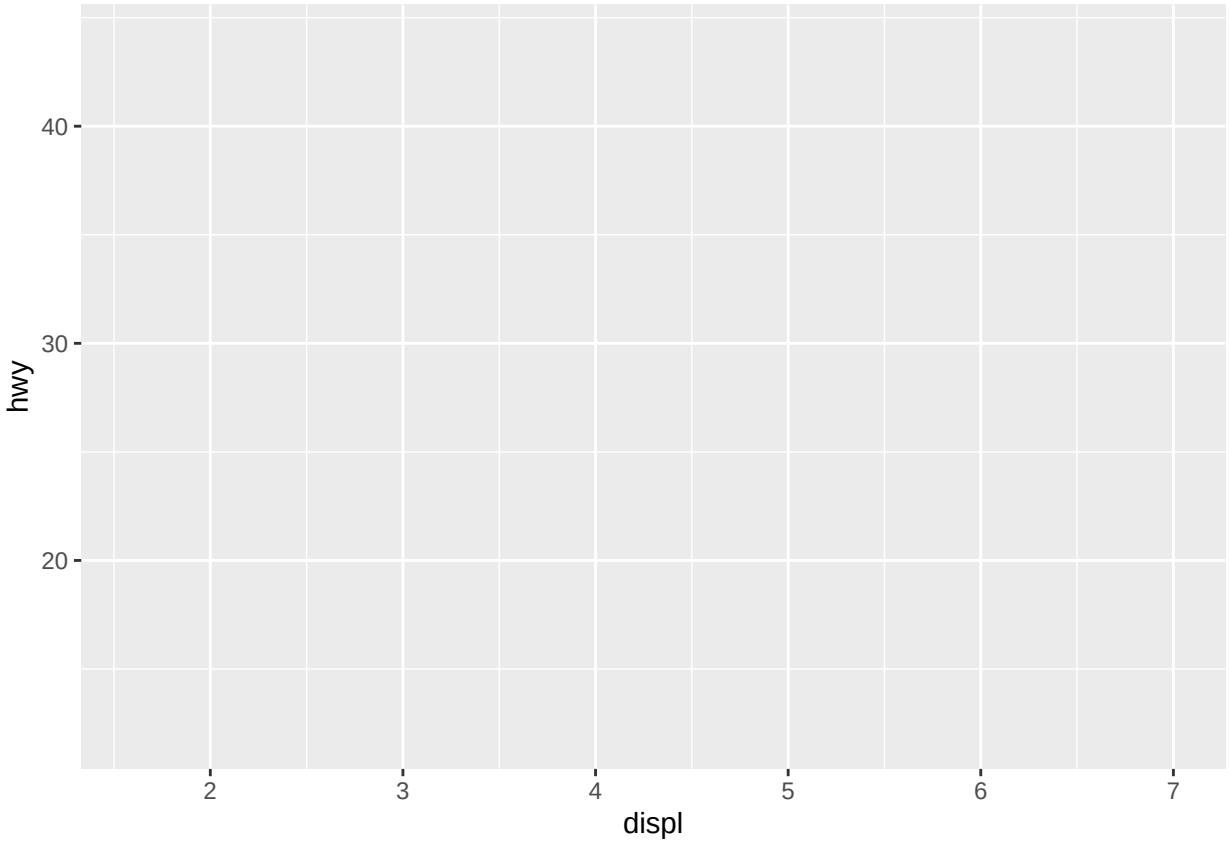



```
p + geom_bar(aes(fill = class)) + guides(fill=FALSE)
```



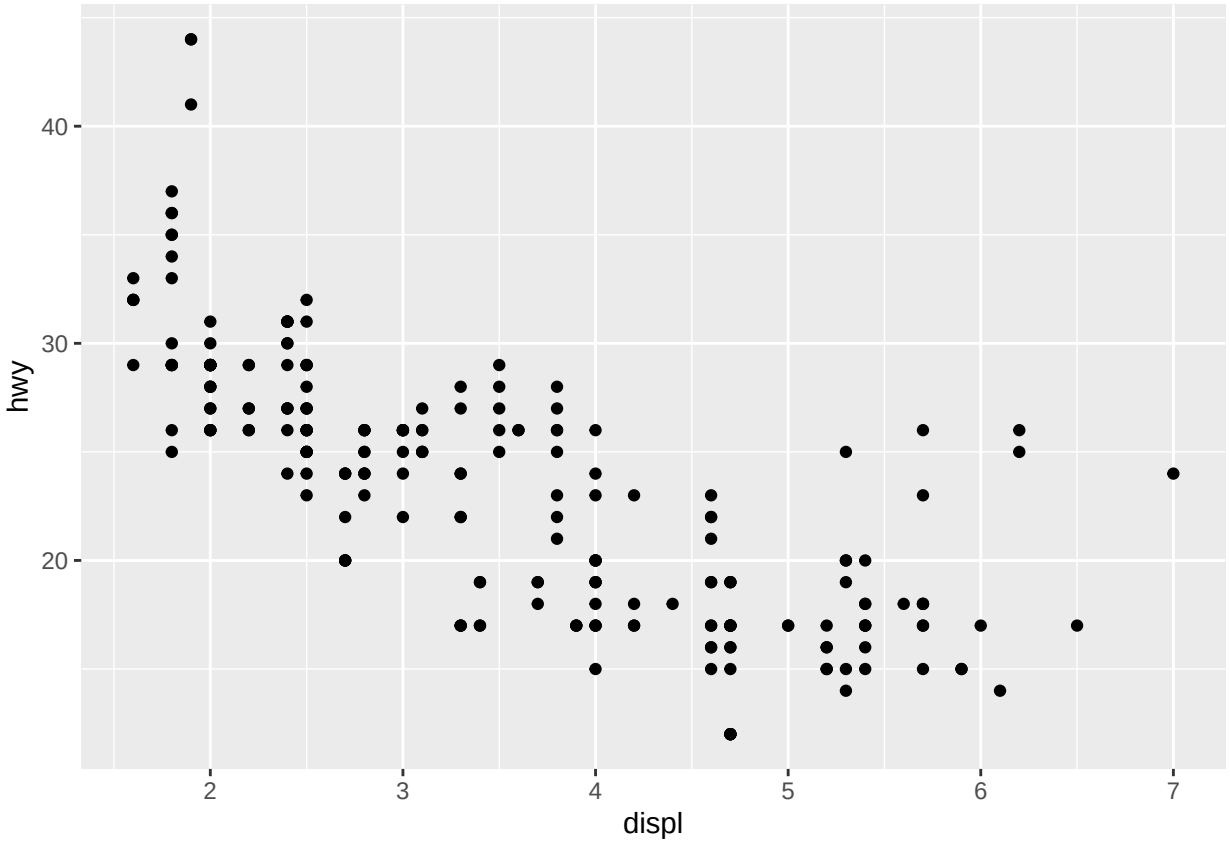
Bivariate plot

```
p <- ggplot(data = mpg, aes(x = displ, y = hwy))
p
```

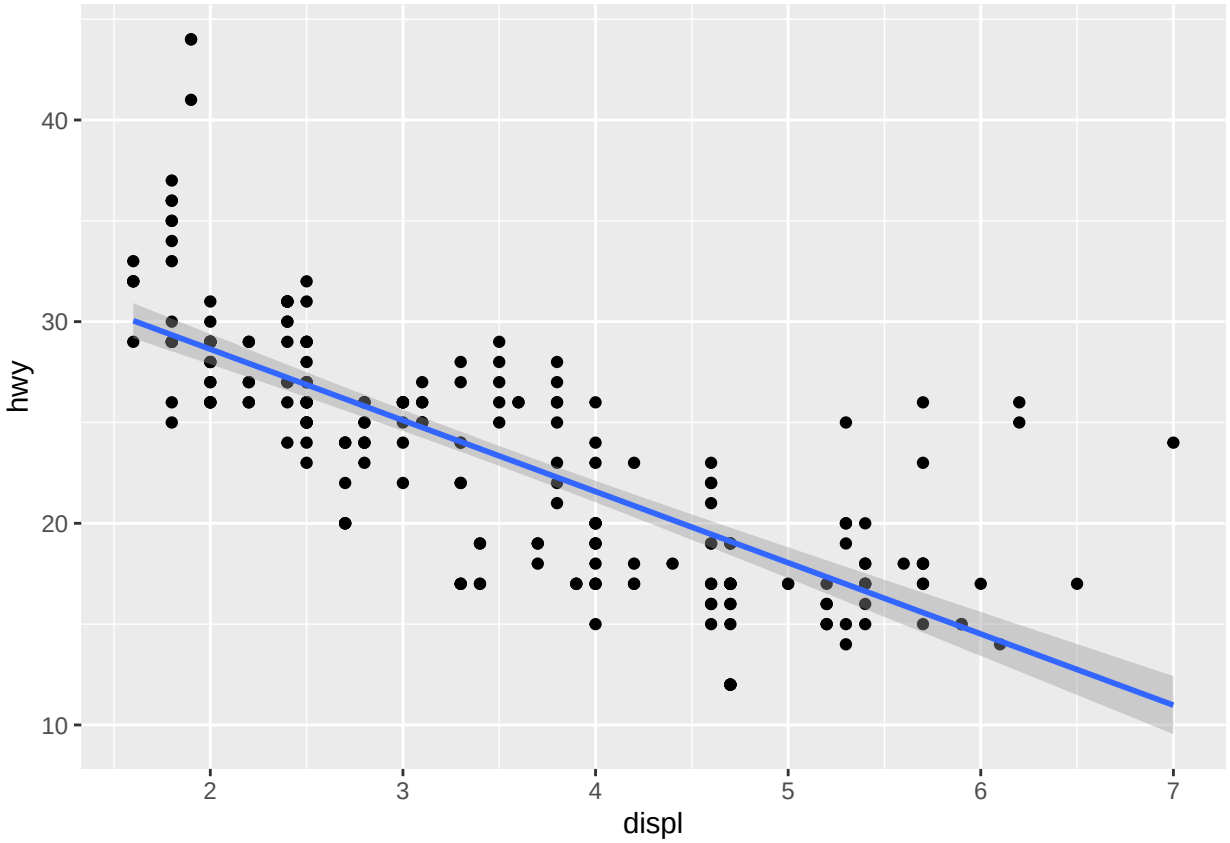


Scatterplot

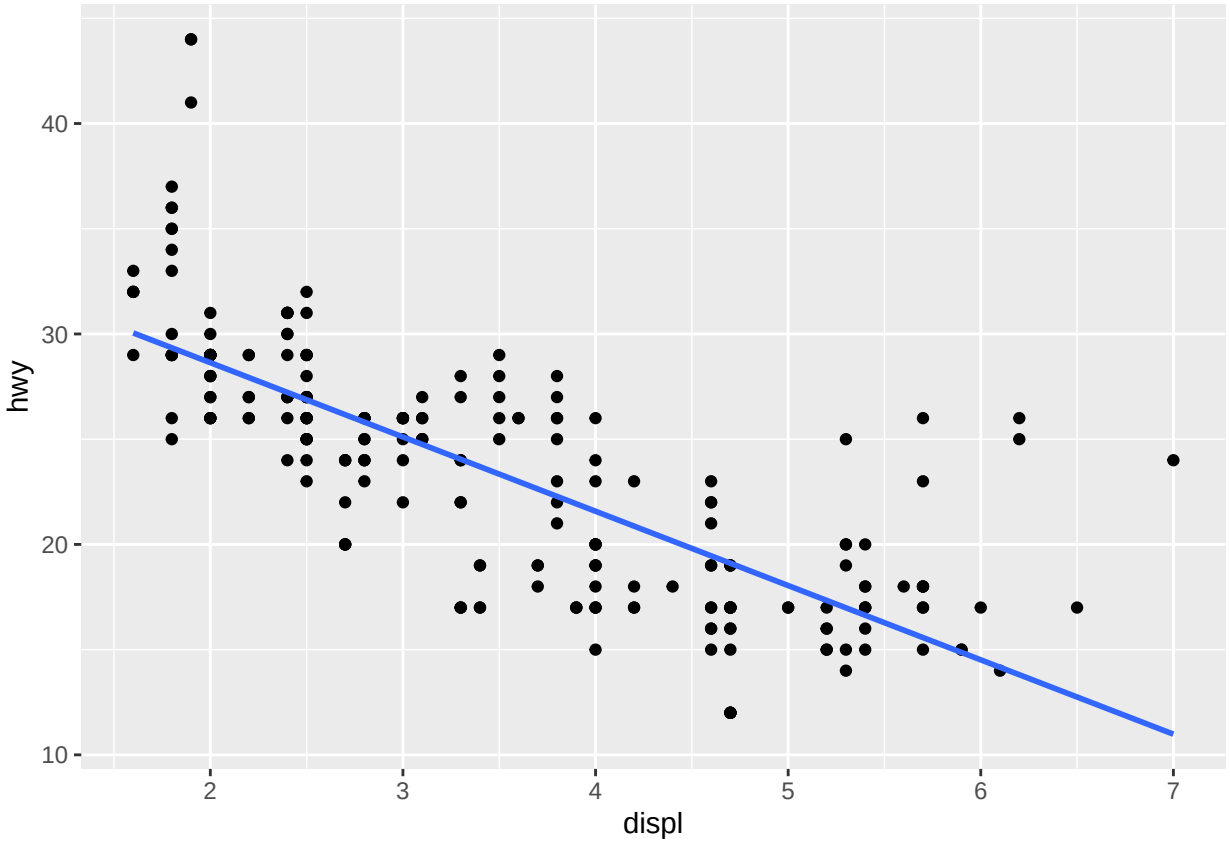
```
p <- ggplot(data = mpg, aes(x = displ, y = hwy))
p + geom_point()
```




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

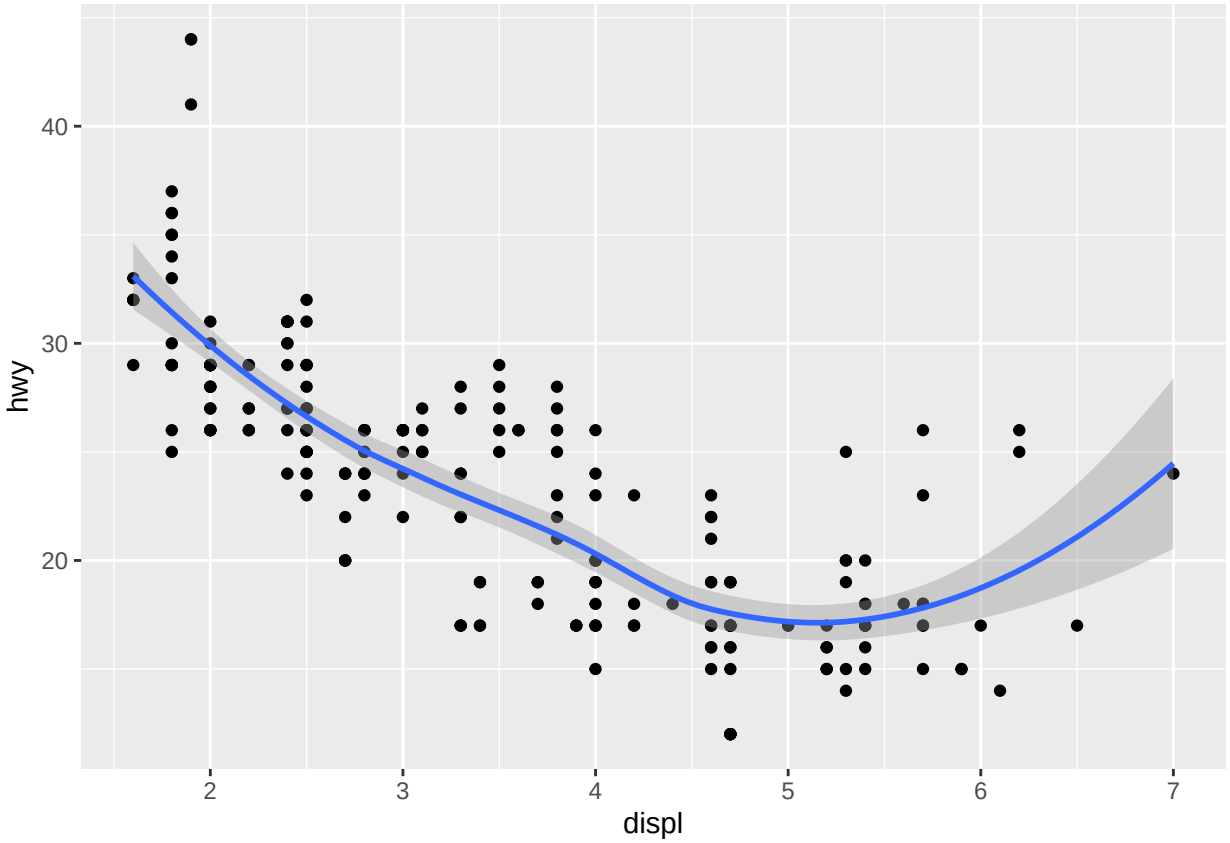



```
p + geom_point() + geom_smooth(method=lm, se = FALSE)
```

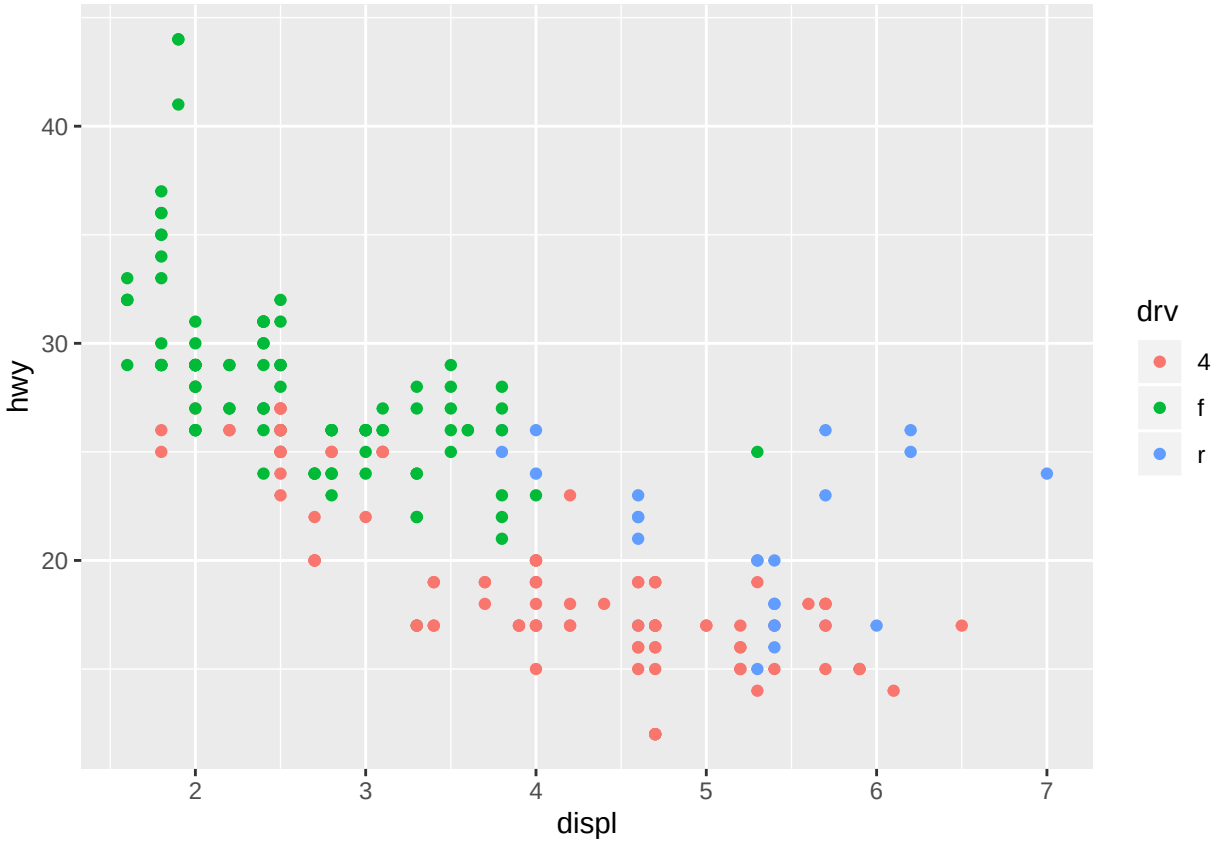



```
p + geom_point() + geom_smooth()
```

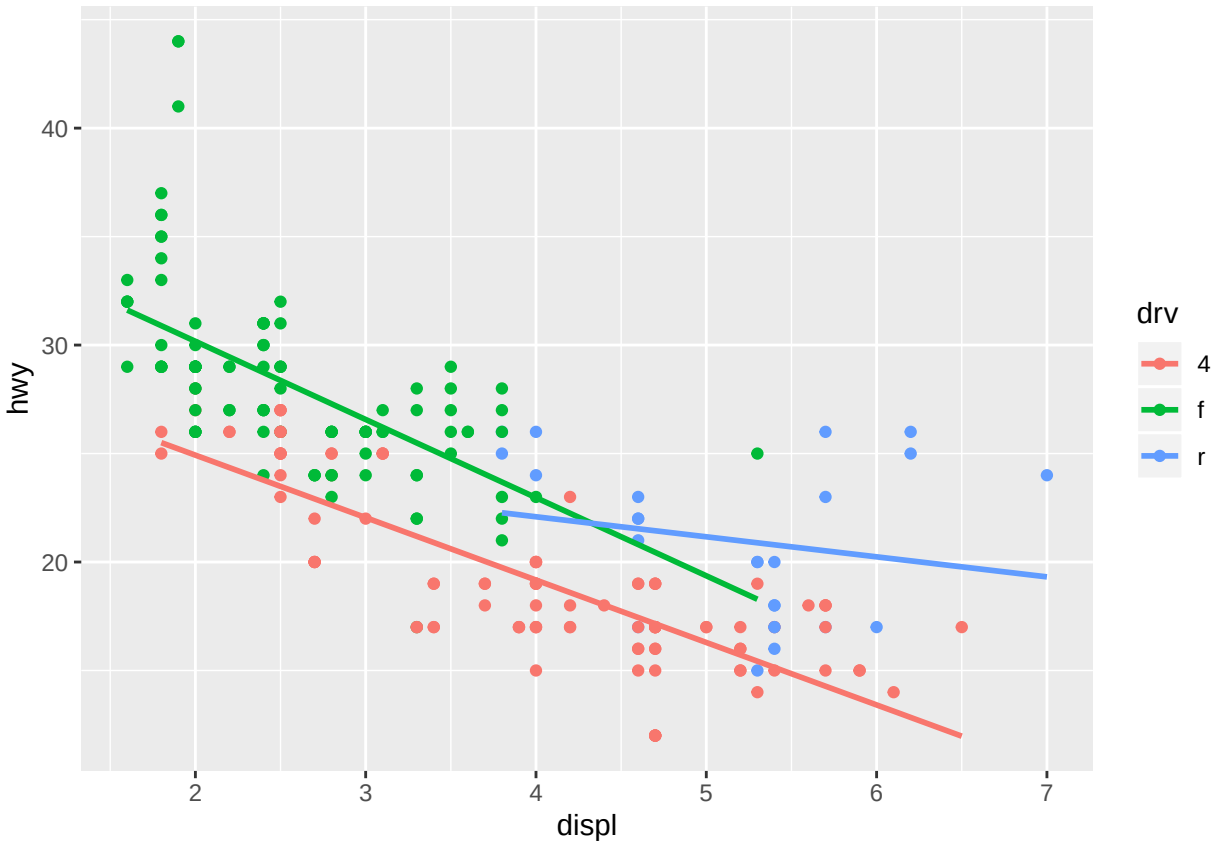
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



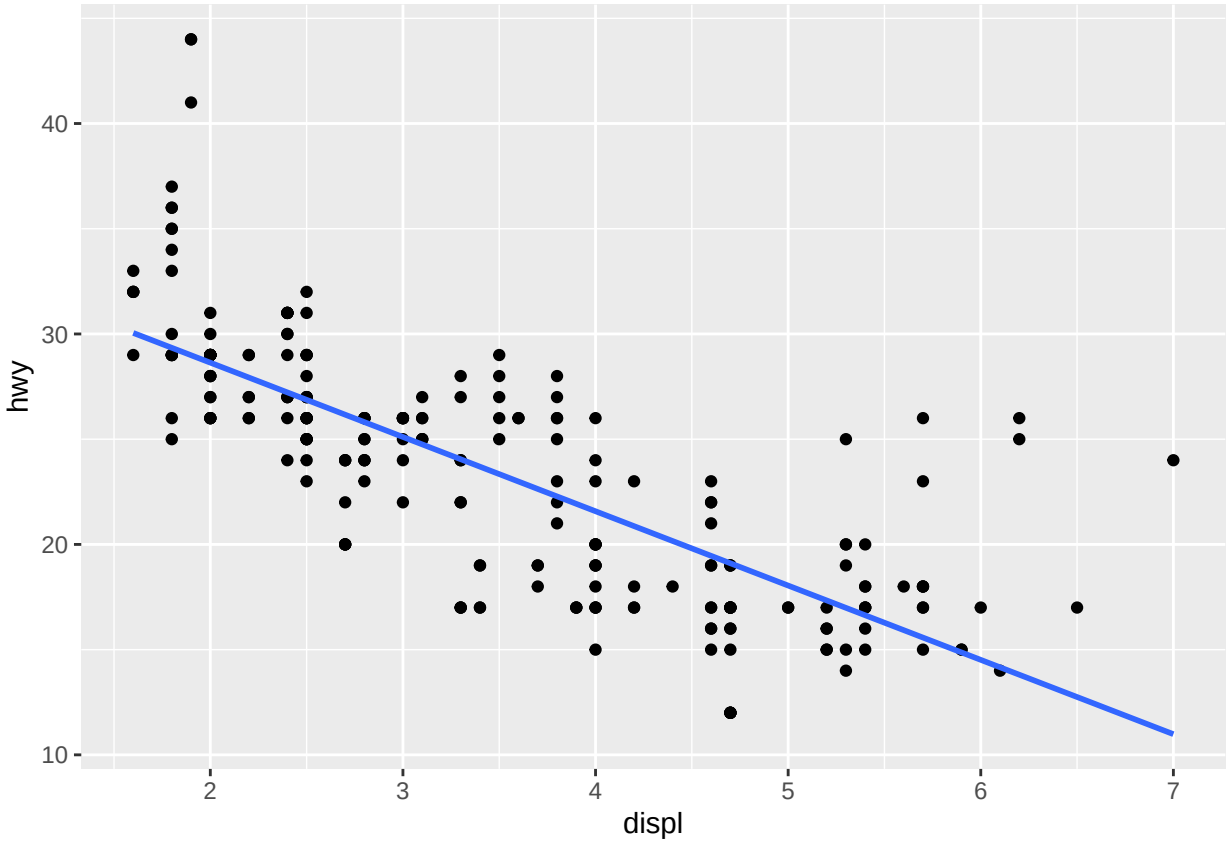

```
p + geom_point(aes(color=drv))
```




```
p1 <- ggplot(data = mpg, aes(x = displ, y = hwy,color=drv))
p1 + geom_point(aes(color=drv)) + geom_smooth(method=lm,se=FALSE)
```

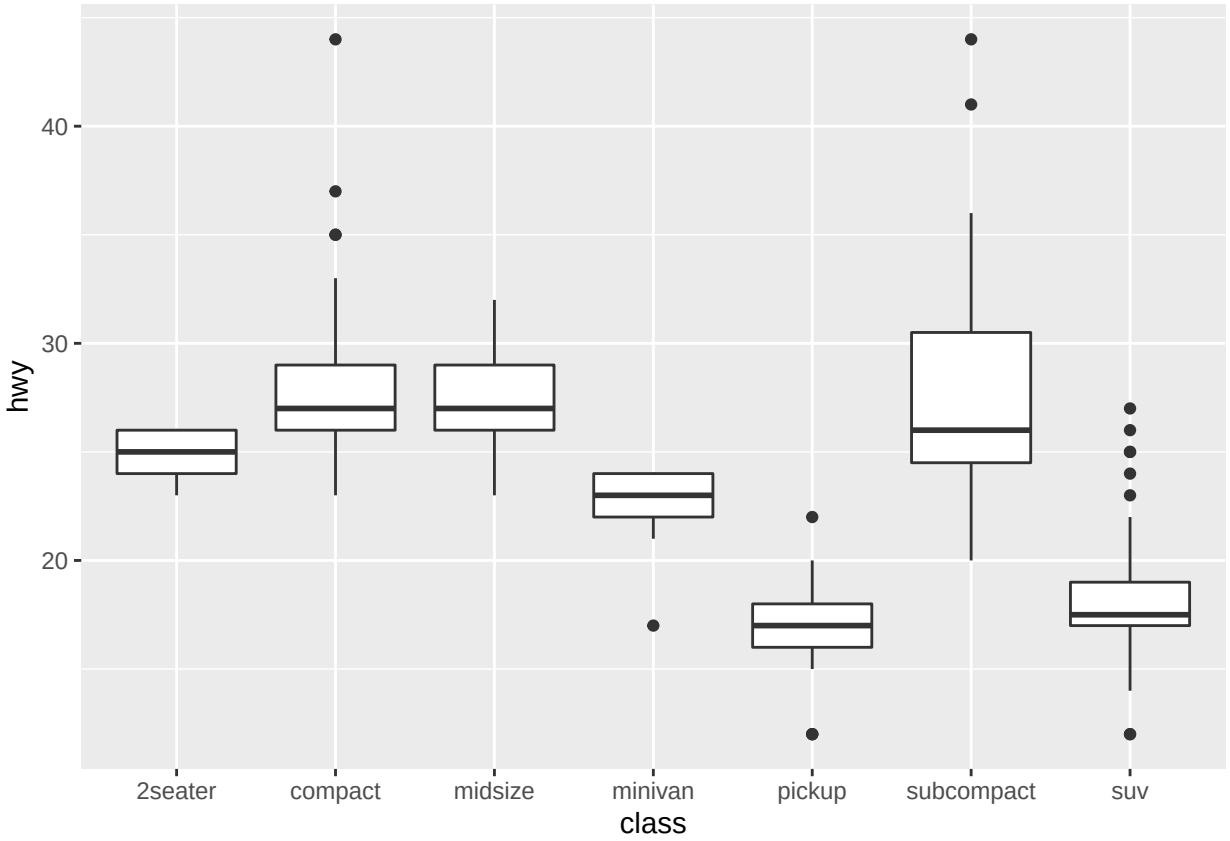



```
p + geom_point() + geom_smooth(method=lm,se=FALSE)
```

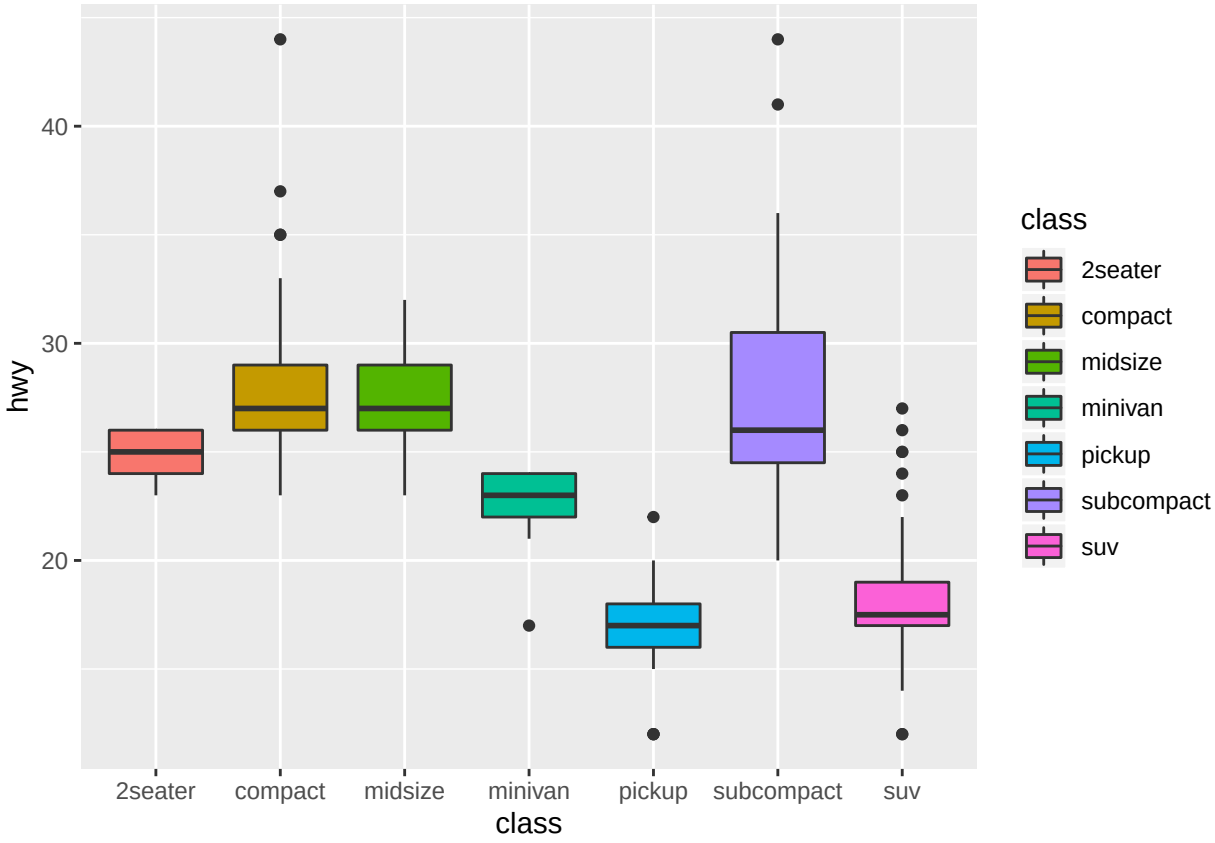


Boxplot

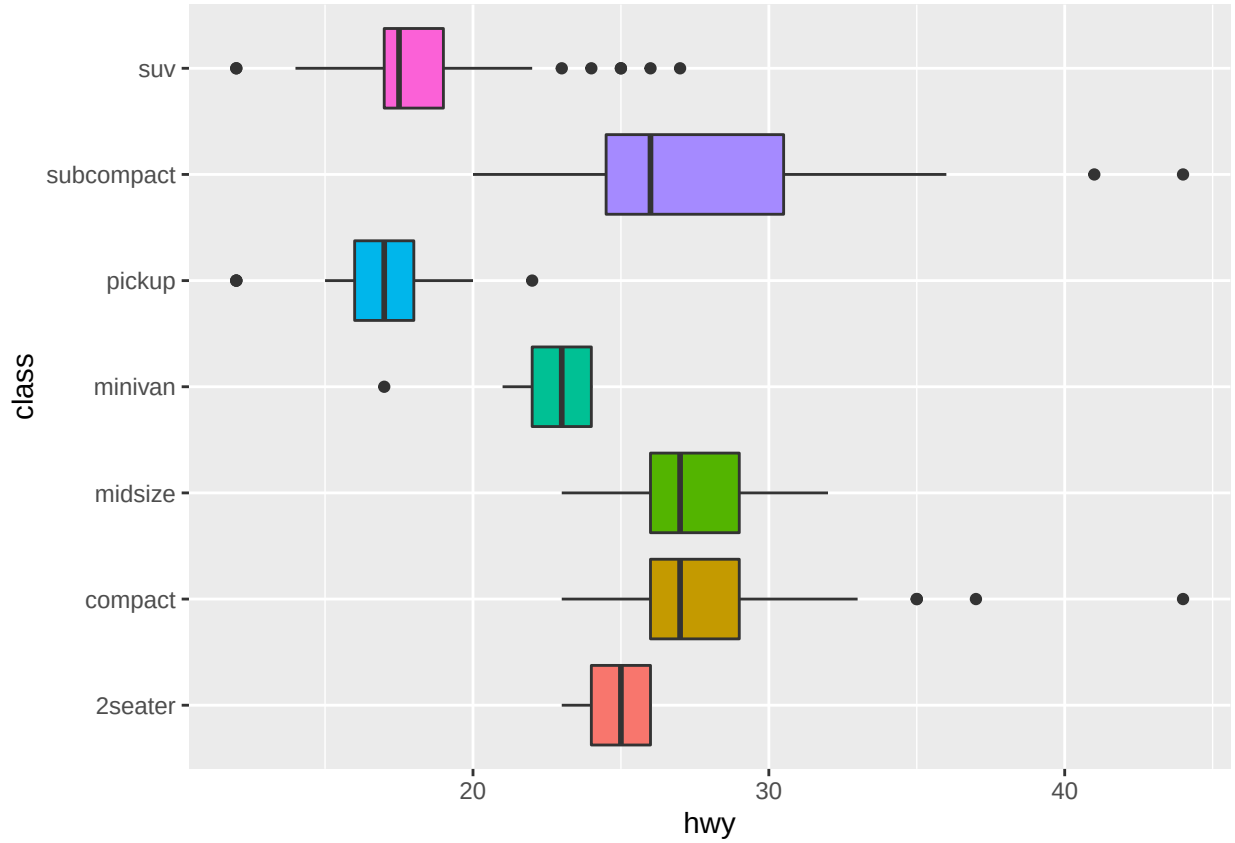
```
ggplot(mpg, aes(x=class,y=hwy)) + geom_boxplot()
```




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

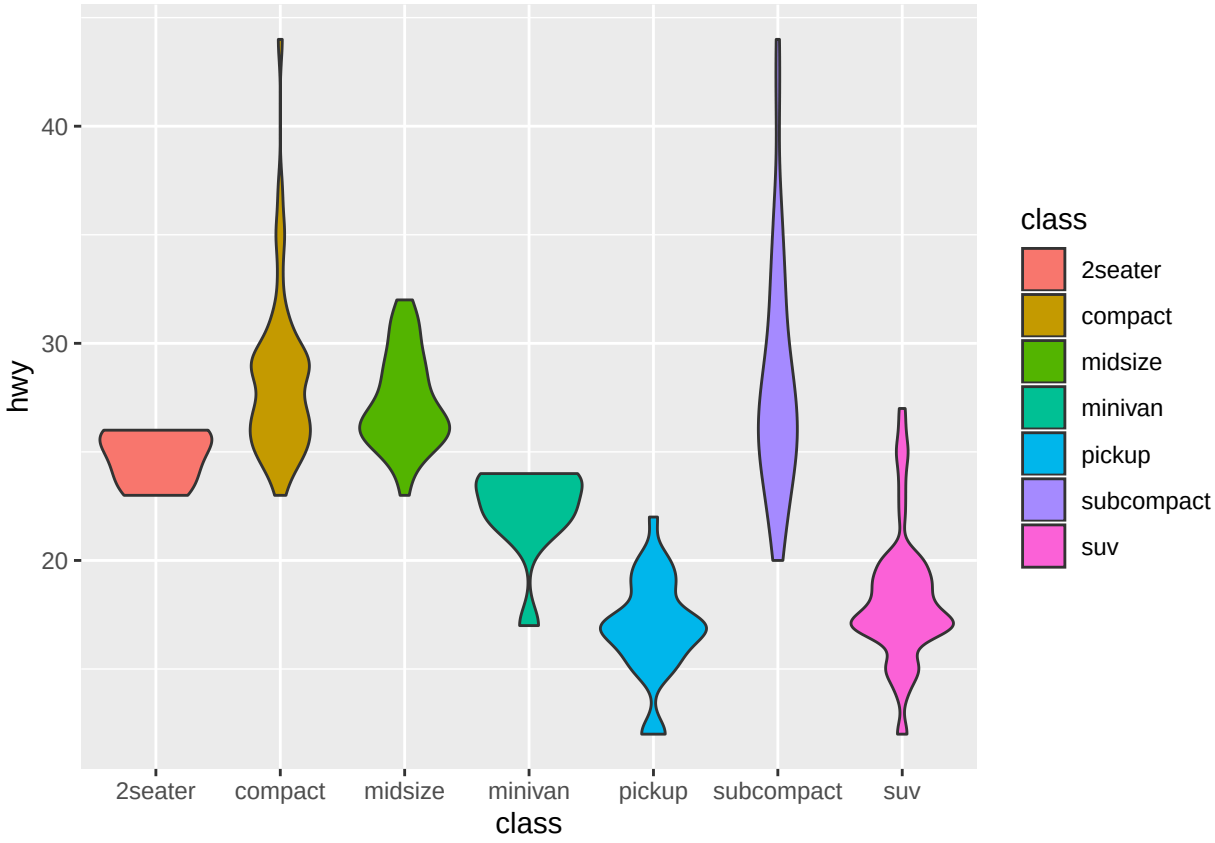



```
ggplot(mpg, aes(x=class,y=hwy, fill=class)) + geom_boxplot()  + coord_flip() + guides(fill=FALSE)
```



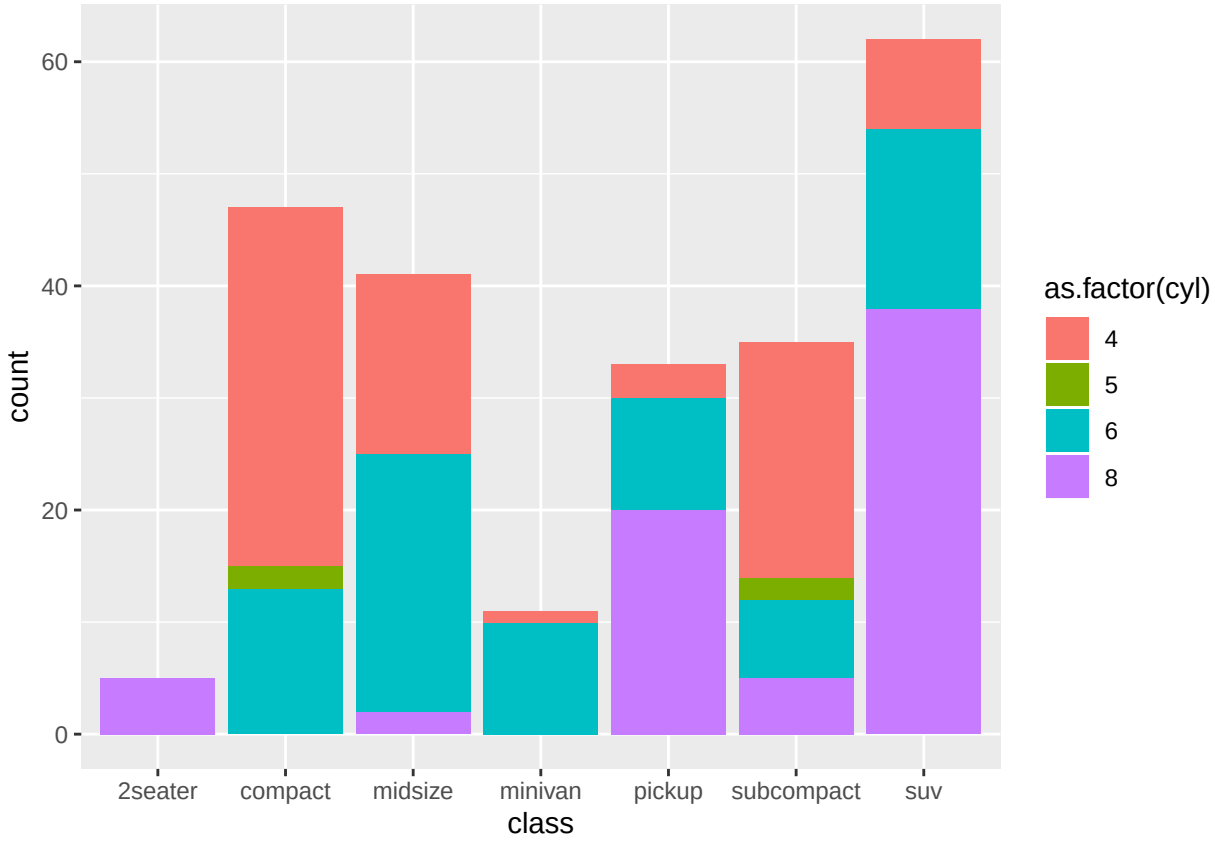
violin plot

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

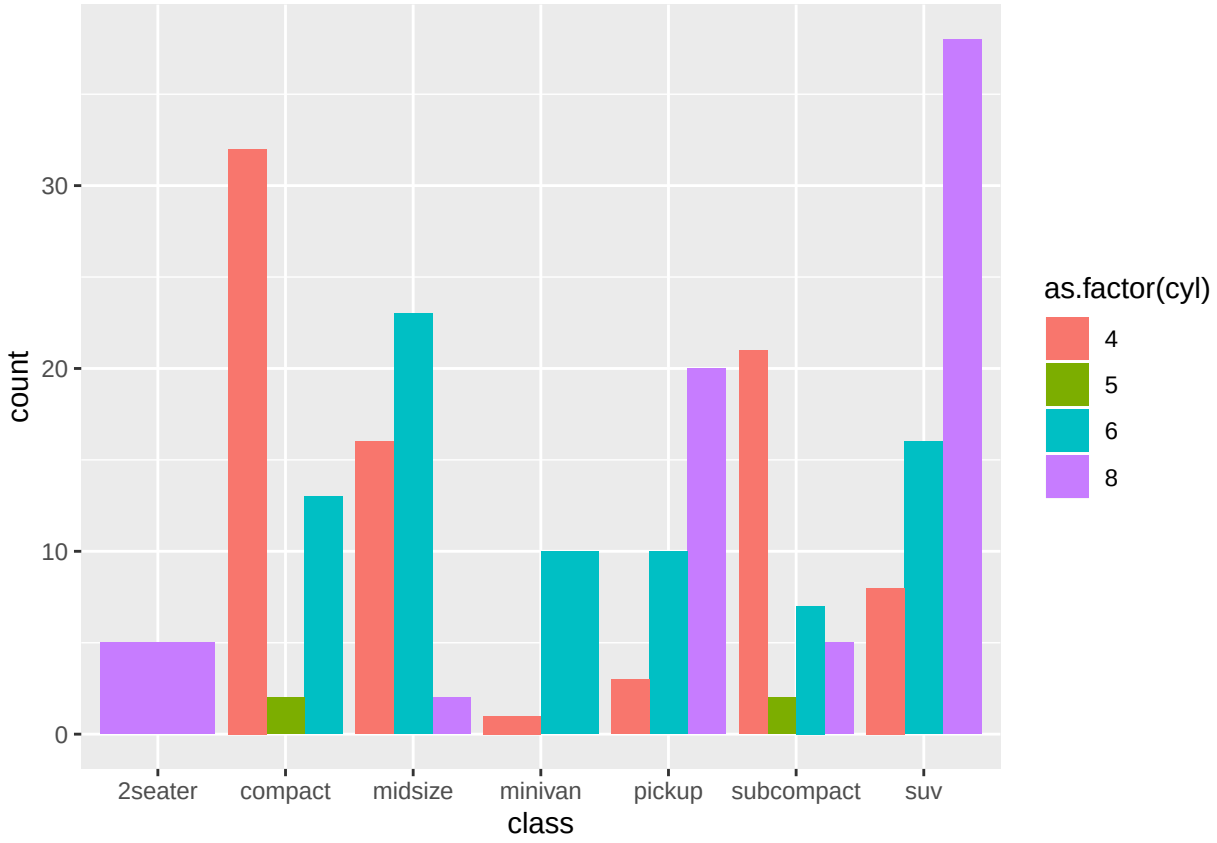


Barplot

```
p <- ggplot(mpg, aes(x=class, fill=as.factor(cyl)))
p + geom_bar()
```




```
p + geom_bar(position = position_dodge())
```



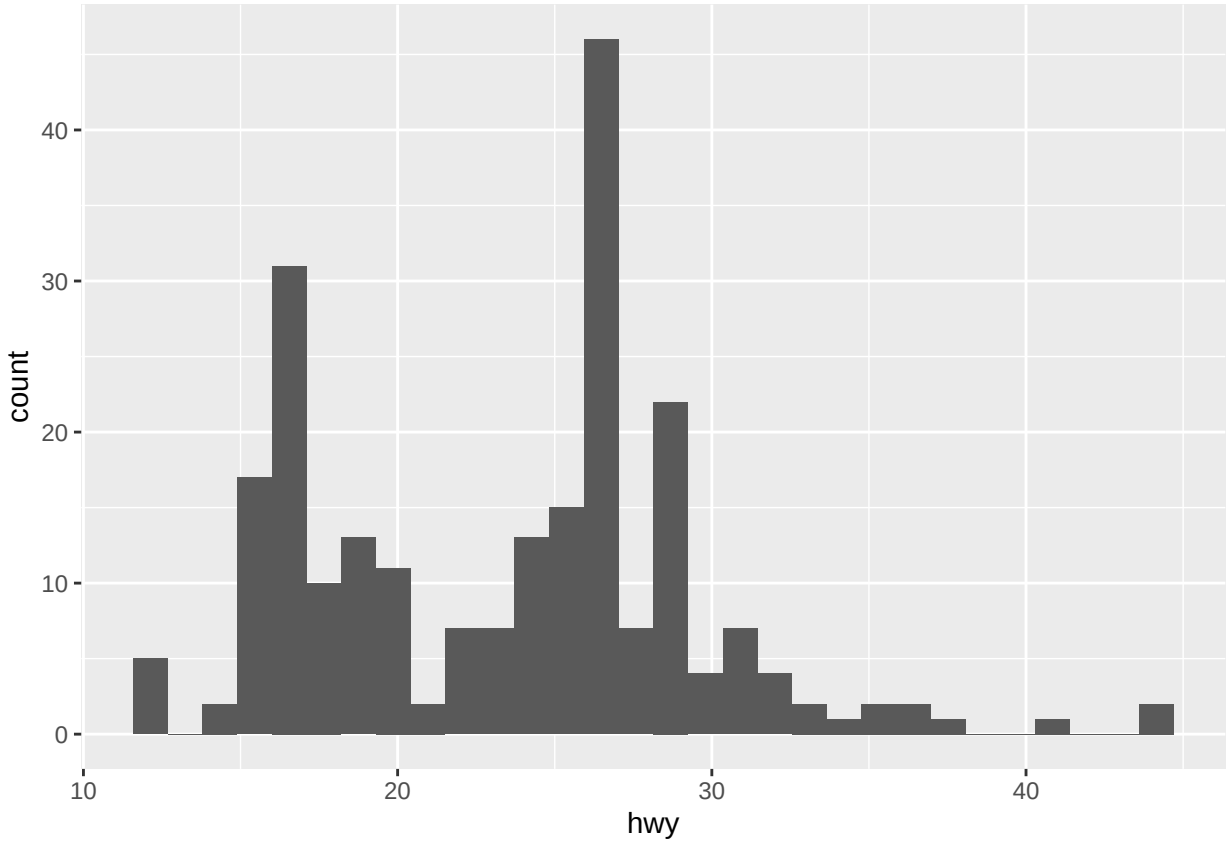
Stats

- The `stat` functions statistically transform data, usually as some form of summary. For example:
  - frequencies of values of a variable (histogram, bar graphs)
  - a mean
  - a confidence limit
- Each `stat` function is associated with a default geom, so no geom is required for shapes to be rendered. The geom can often be respecfied in the stat for different shapes.

`stat_bin()` produces a histogram

```
ggplot(data = mpg, aes(x = hwy)) + stat_bin()
```

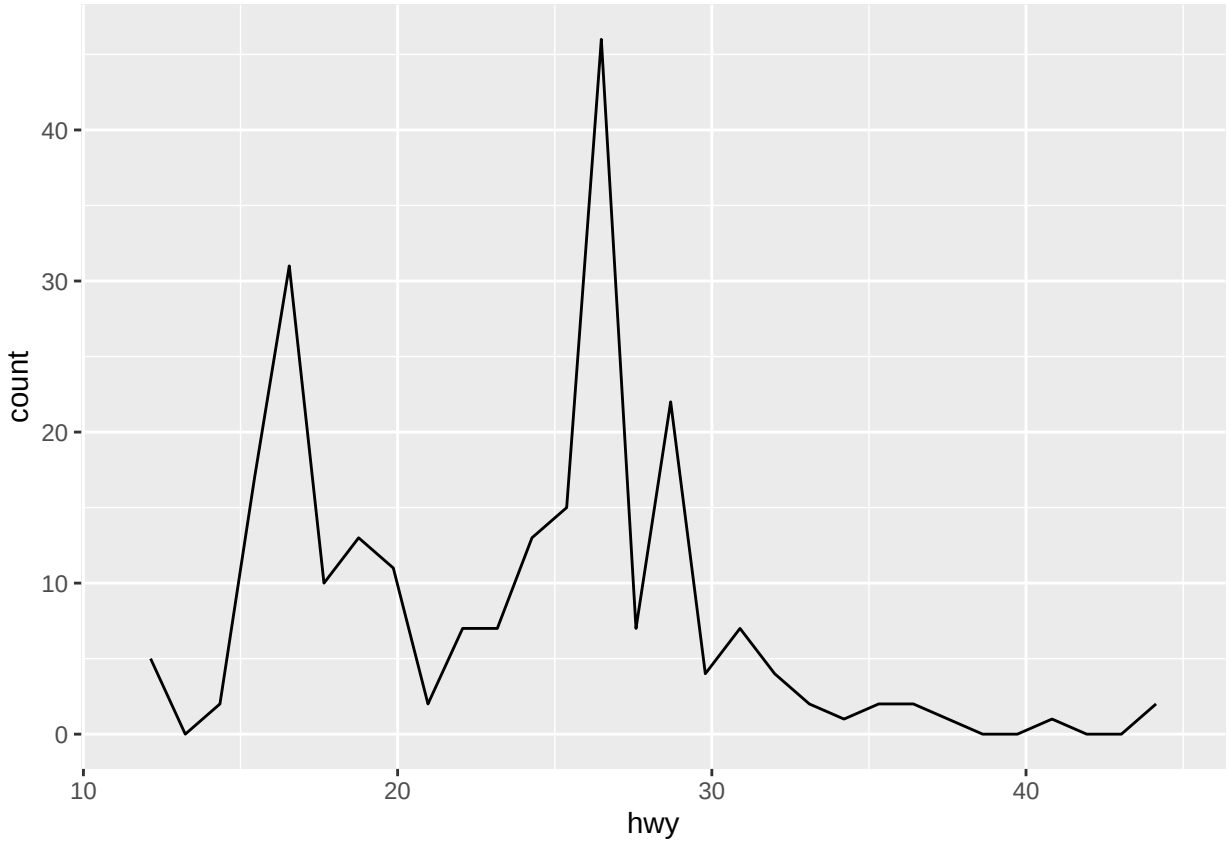
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Respecifying the geom to `geom=line`

```
ggplot(data = mpg, aes(x = hwy)) + stat_bin(geom = "line")
```

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

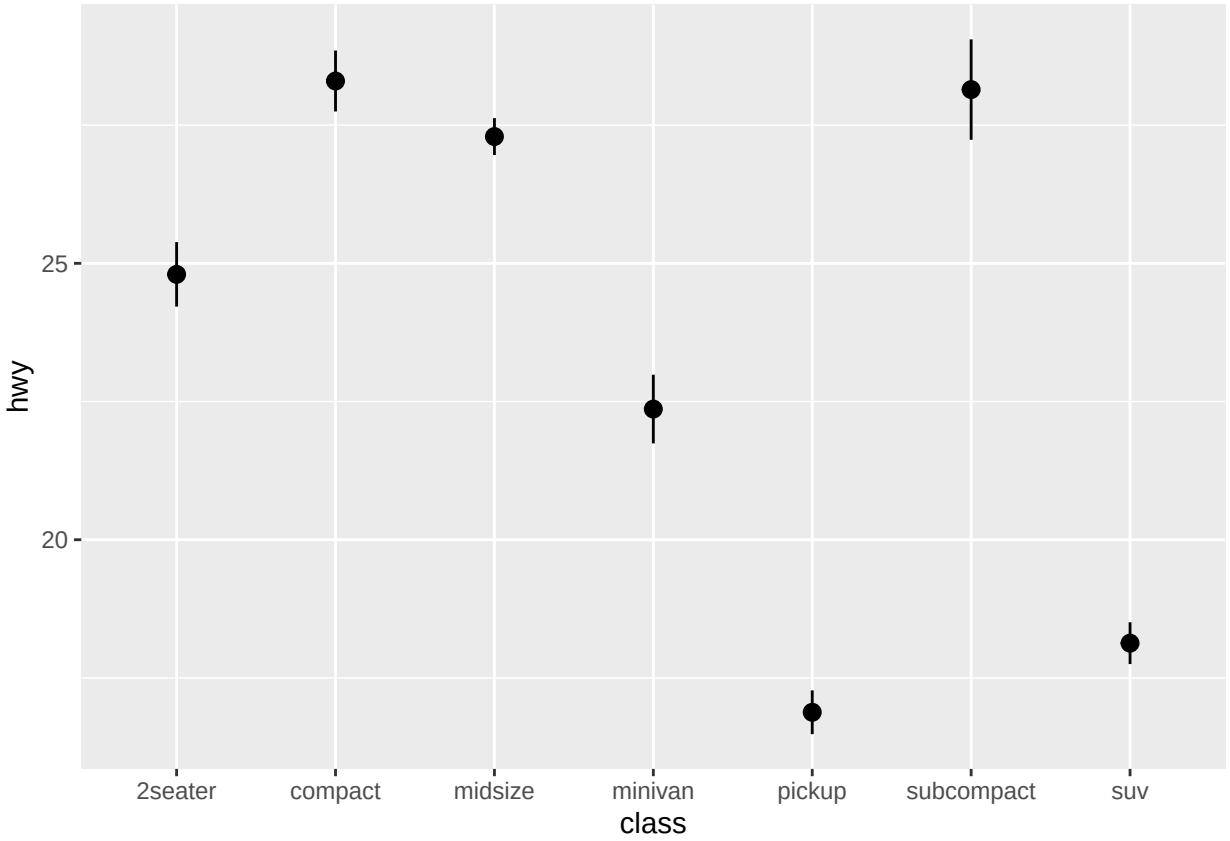


stat_summary() summarizes y at each x

- `stat_summary()` applies a summary function (e.g. mean) to y at each value or interval of x.
- The default summary function is `mean_se`, which returns the mean ± standard error. Thus, `stat_summary()` plots the mean and standard errors for y at each x value.
- The default geom is "pointrange", which, places a dot at the mean of y (for that x) and extends lines to mean ± s.e.

```
ggplot(data = mpg, aes(x = class, y = hwy)) + stat_summary()
```

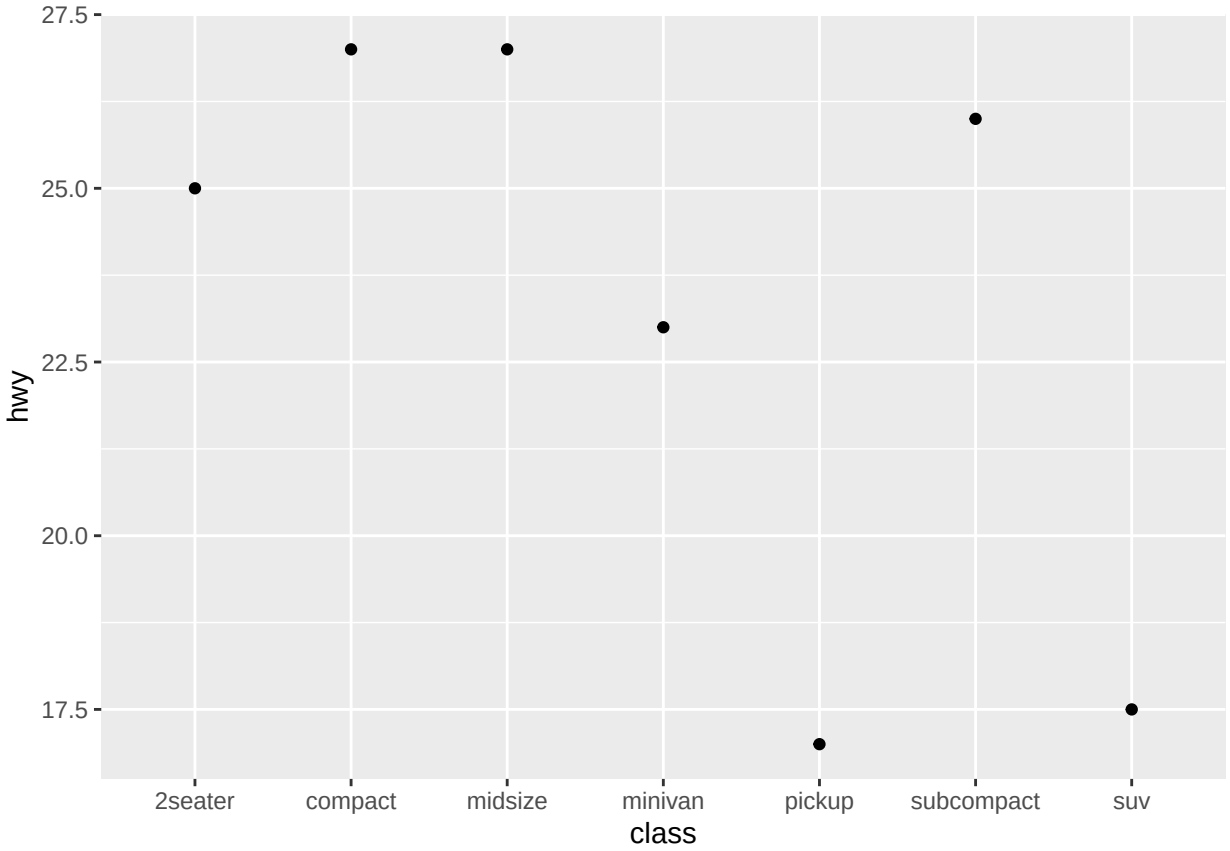
```
## No summary function supplied, defaulting to `mean_se()
```



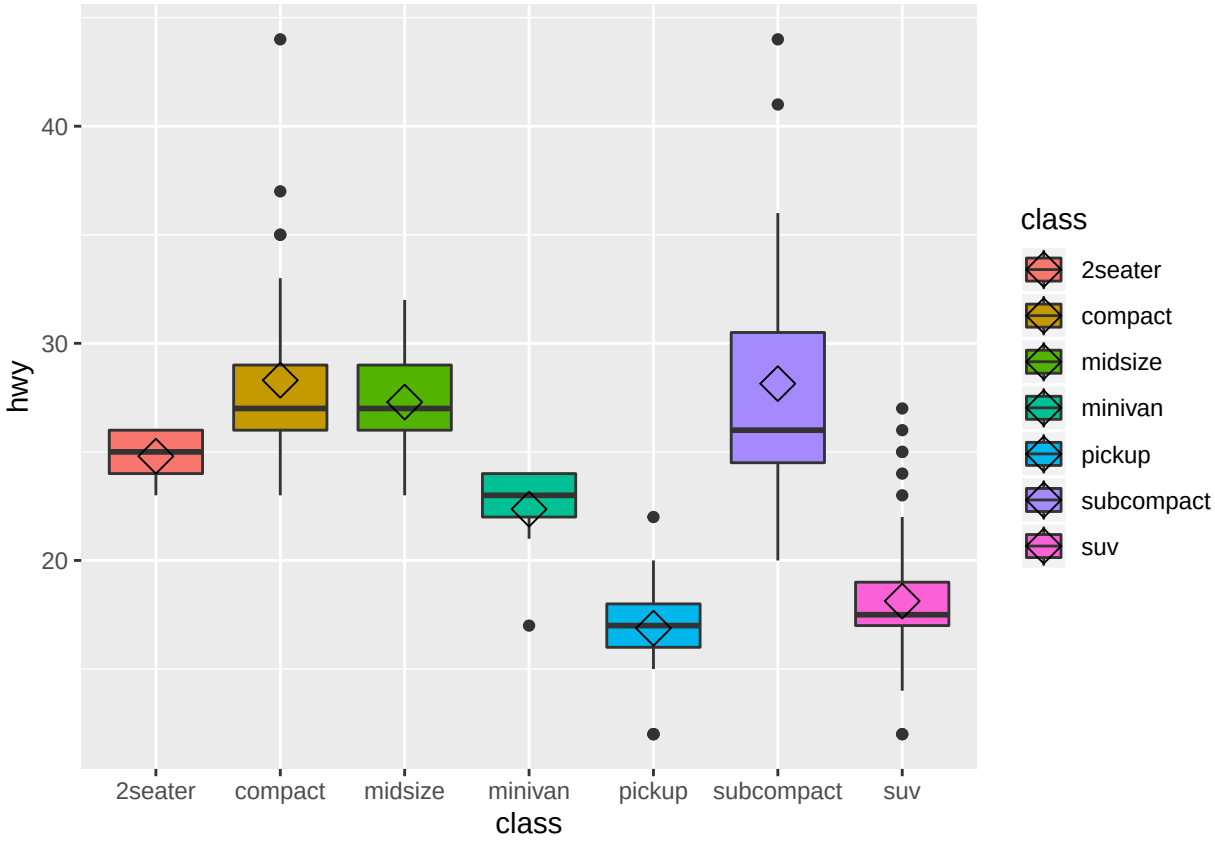
Other summary functions and geoms with `stat_summary()`

- mean_cl_boot: mean and bootstapped confidence interval (default 95%)
- mean_cl_normal: mean and Guassian (t-distribution based) confidence interval (default 95%)
- mean_dsl: mean plus or minus standard deviation times some constant (default constant=2)
- median_hilow: median and outer quantiles (default outer quantiles = 0.025 and 0.975)

```
ggplot(data = mpg, aes(x = class, y = hwy)) +
  stat_summary( fun.y = "median", geom = "point")
```

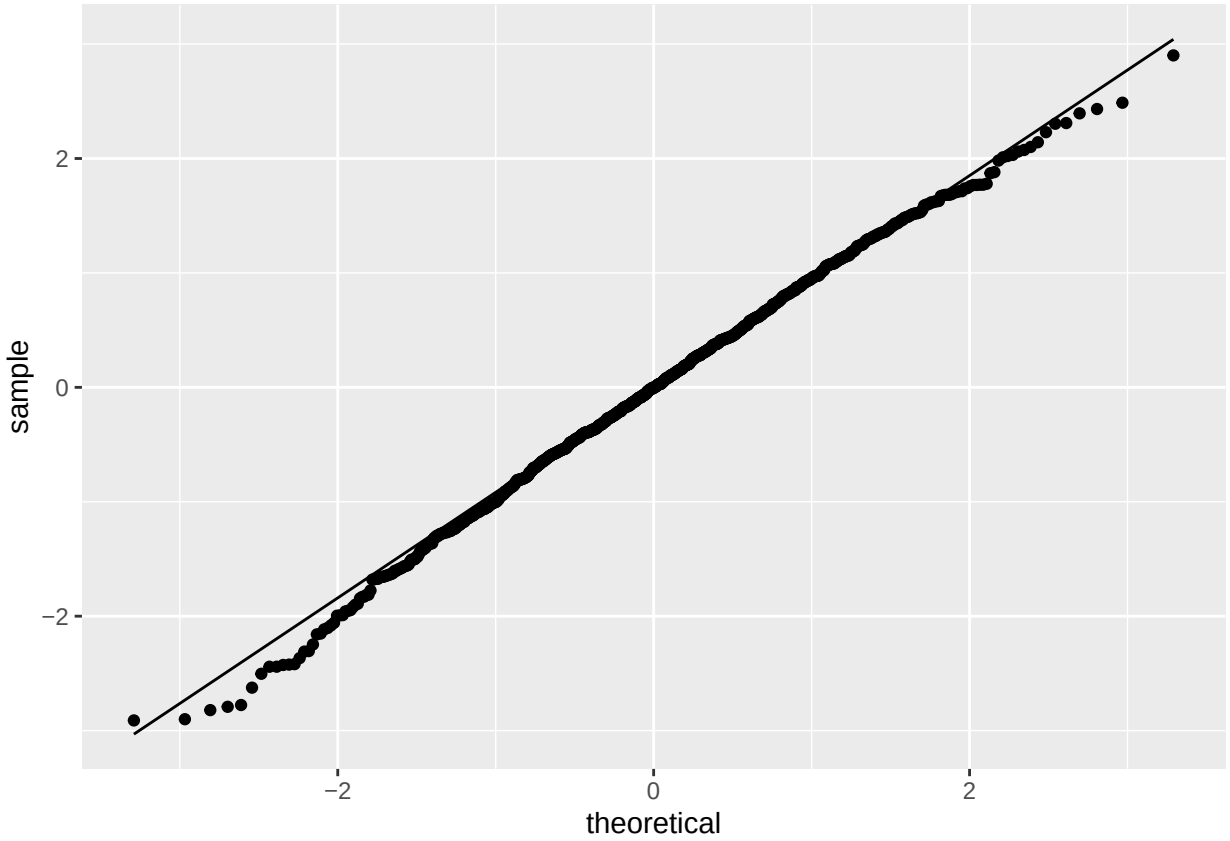



```
ggplot(mpg, aes(x=class,y=hwy, fill=class)) + geom_boxplot() +
  stat_summary(fun.y=mean, geom="point", shape=5, size=4)
```

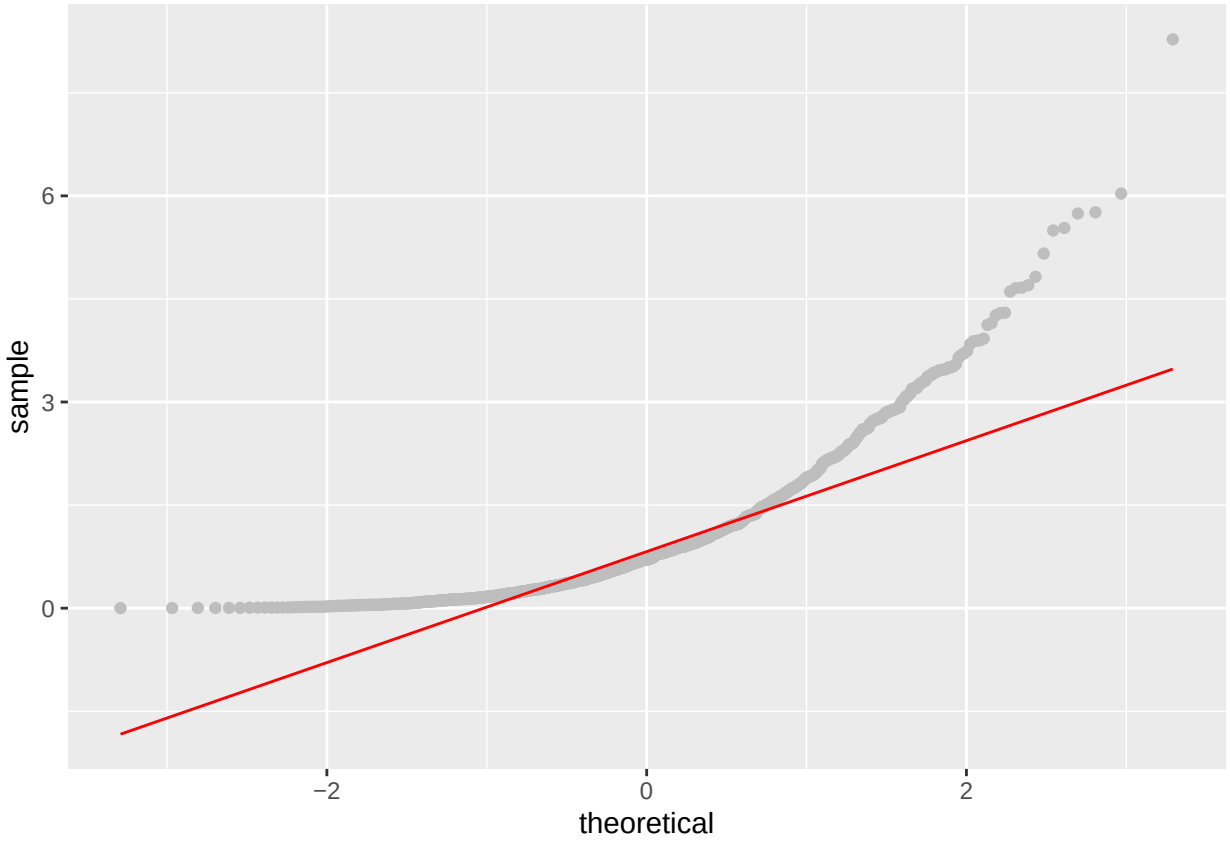


qq plot

```
data <- data.frame(x = rnorm(1000), y = rexp(1000))
ggplot(data, aes(sample=x))+stat_qq()+stat_qq_line()
```




```
data <- data.frame(x = rnorm(1000), y = rexp(1000))
ggplot(data, aes(sample=y))+stat_qq( col="grey") +stat_qq_line( col="red")
```



Scales

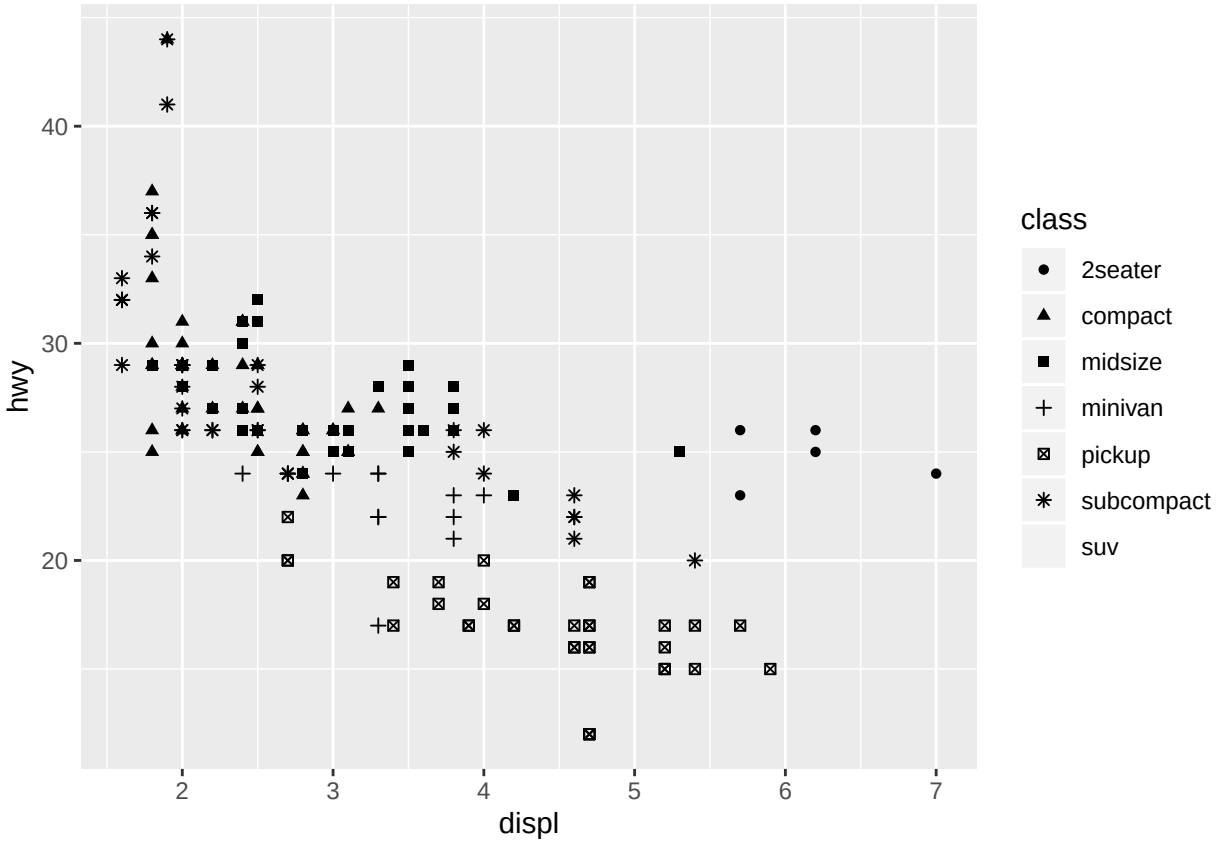
Scales

- Scales define which aesthetic values are mapped to data values.
- The `ggplot2` package usually allows the user to control the scales for each aesthetic. Look for scale functions called something like `scale_aesthetic_manual()` to specify exactly which visual elements will appear on the graph.
- The default `shape` scale

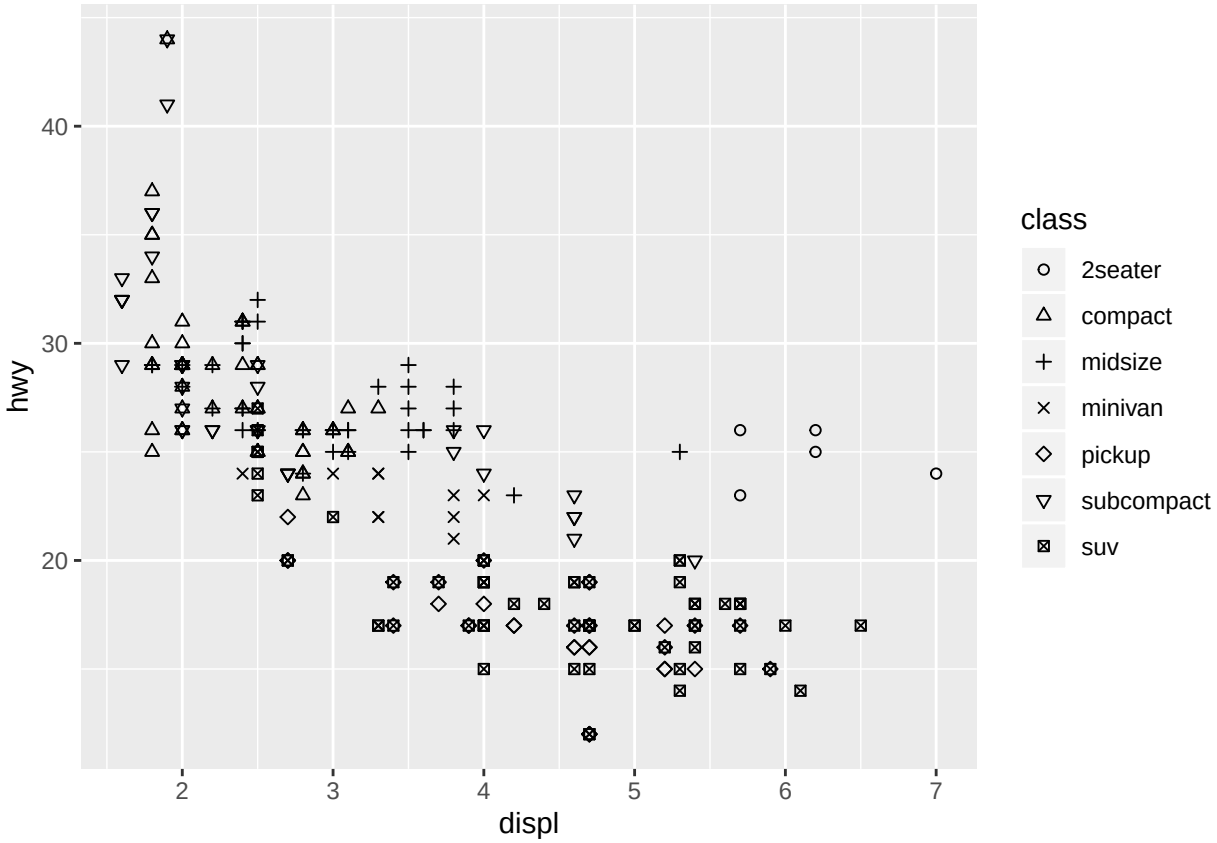
```
ggplot(data = mpg, aes(x = displ, y = hwy, shape = class)) + geom_point()
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values
## because more than 6 becomes difficult to discriminate; you have 7.
## Consider specifying shapes manually if you must have them.
```

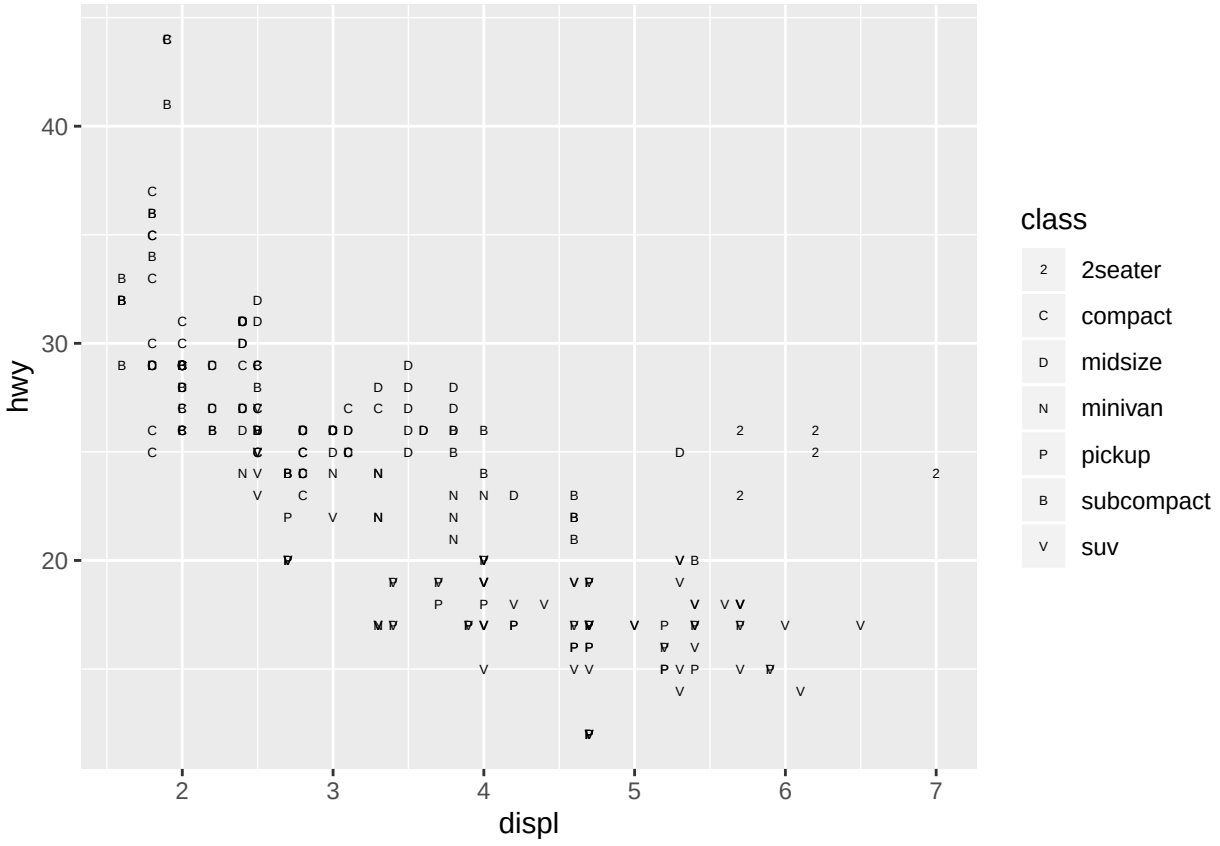
```
## Warning: Removed 62 rows containing missing values (geom_point).
```




```
ggplot(data = mpg, aes(x = displ, y = hwy, shape = class)) + geom_point() +
  scale_shape_manual(values = 1:7)
```



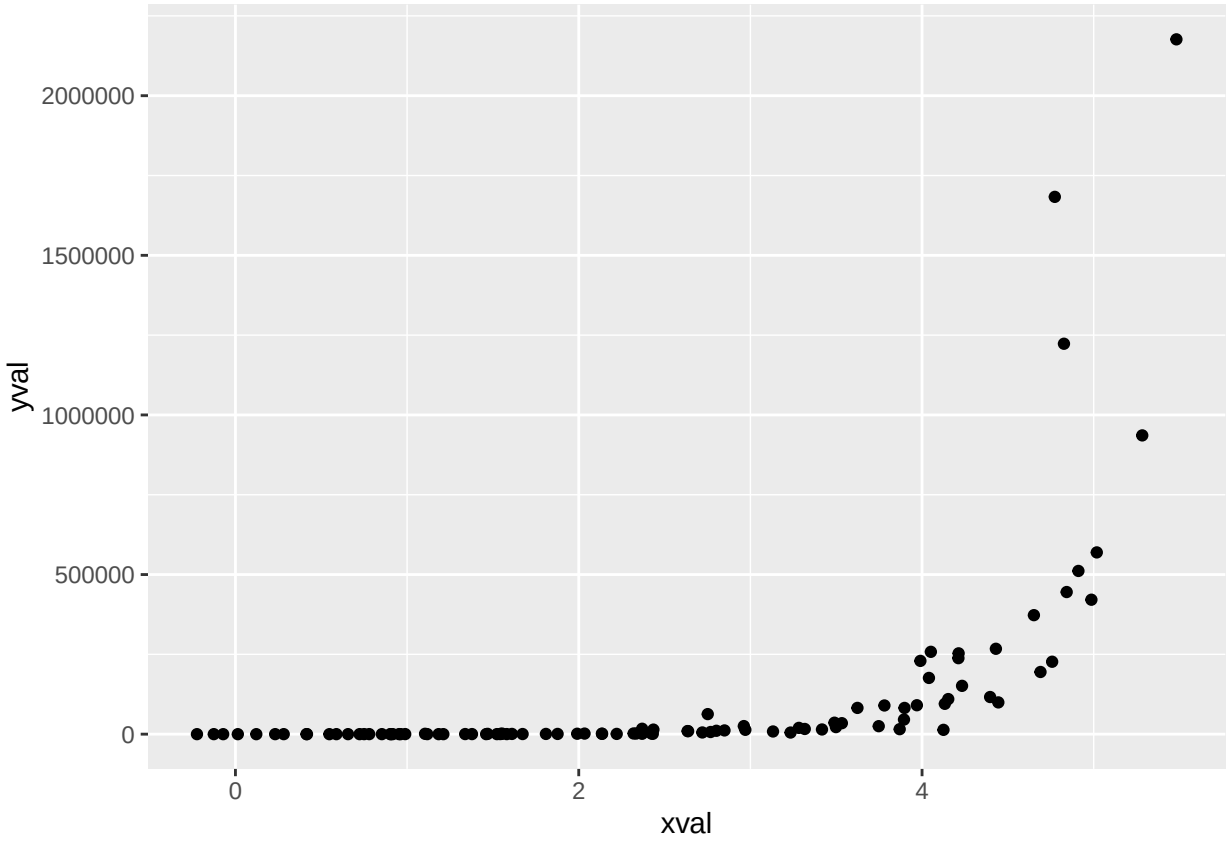

```
ggplot(data = mpg, aes(x = displ, y = hwy, shape = class)) + geom_point() +
  scale_shape_manual(values = c("2","C","D","N","P","B","V"))
```



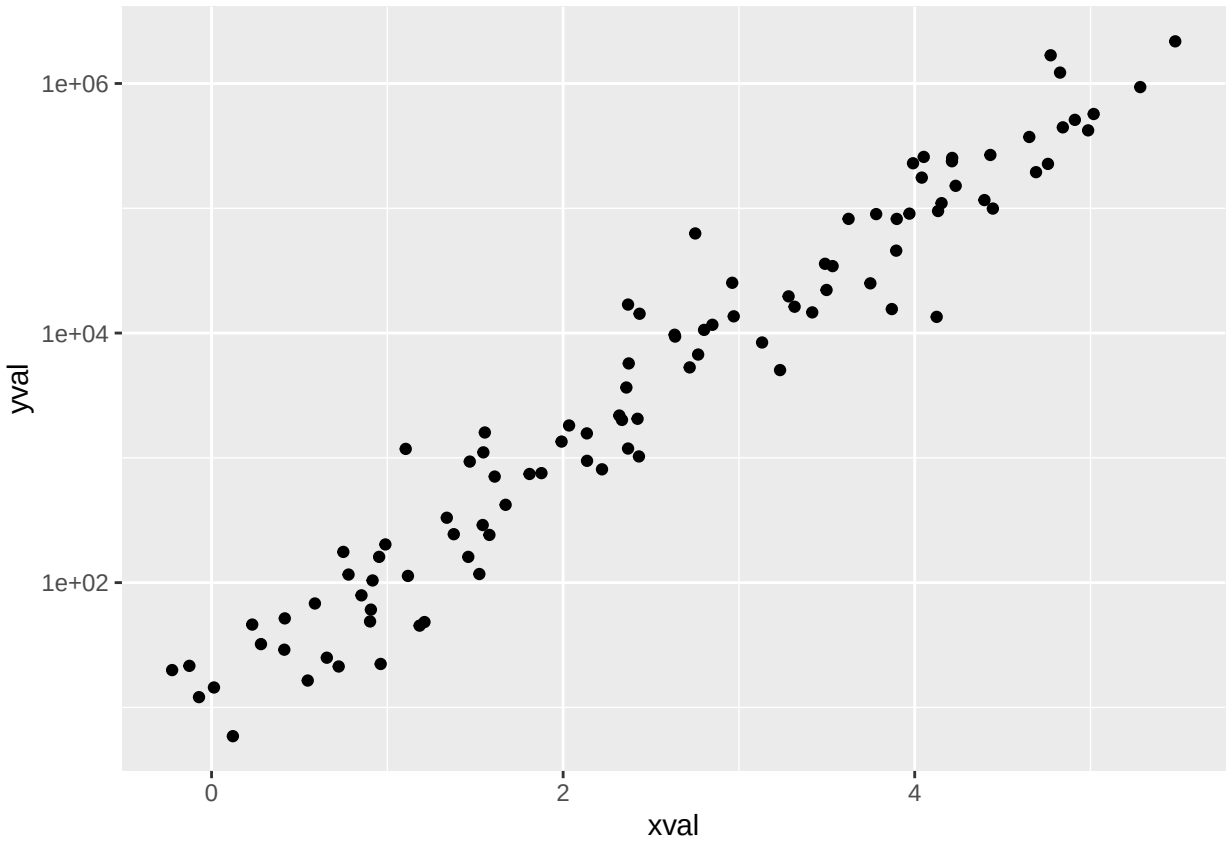
Scale the axis

```
# Create some noisy exponentially-distributed data
set.seed(201)
n <- 100
dat <- data.frame(
    xval = (1:n+rnorm(n,sd=5))/20,
    yval = 10*10^((1:n+rnorm(n,sd=5))/20)
)
```

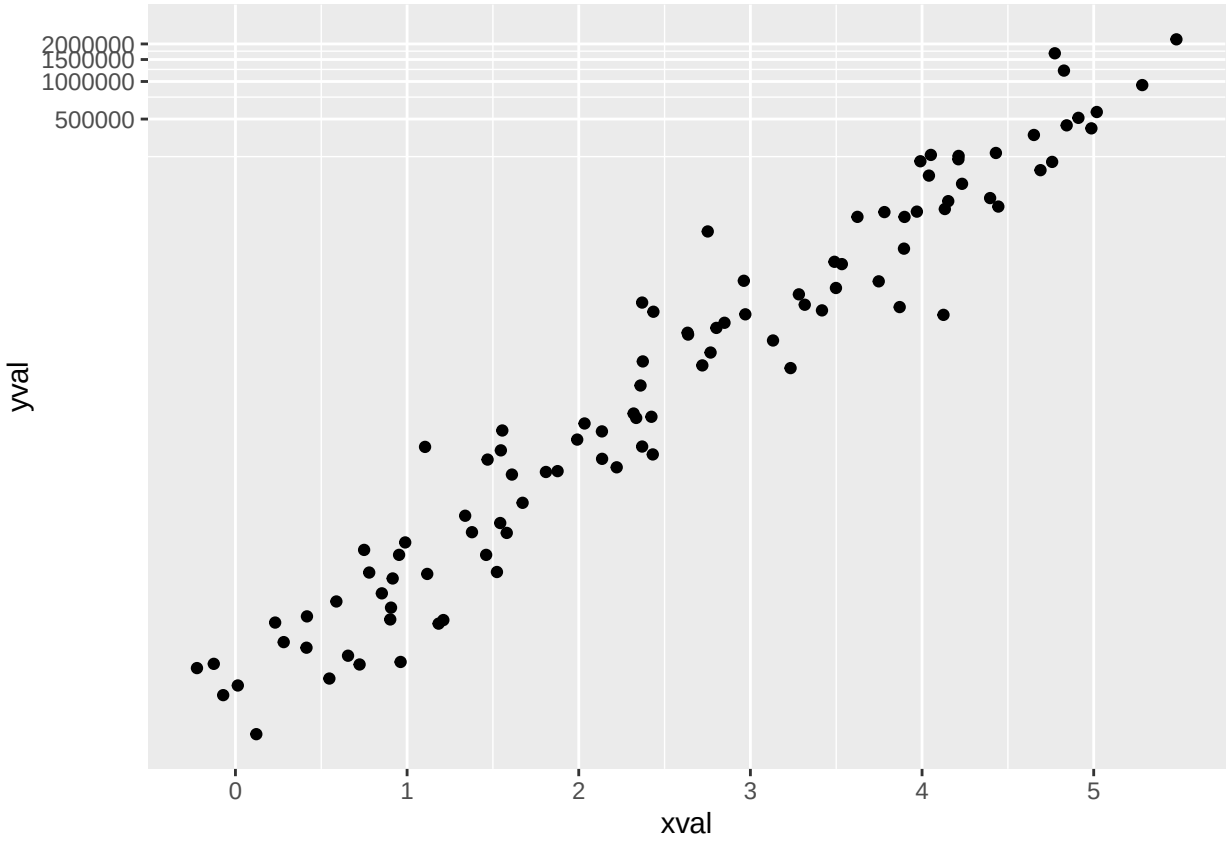
```
# A scatterplot with regular (linear) axis scaling
sp <- ggplot(dat, aes(xval, yval)) + geom_point()
sp
```



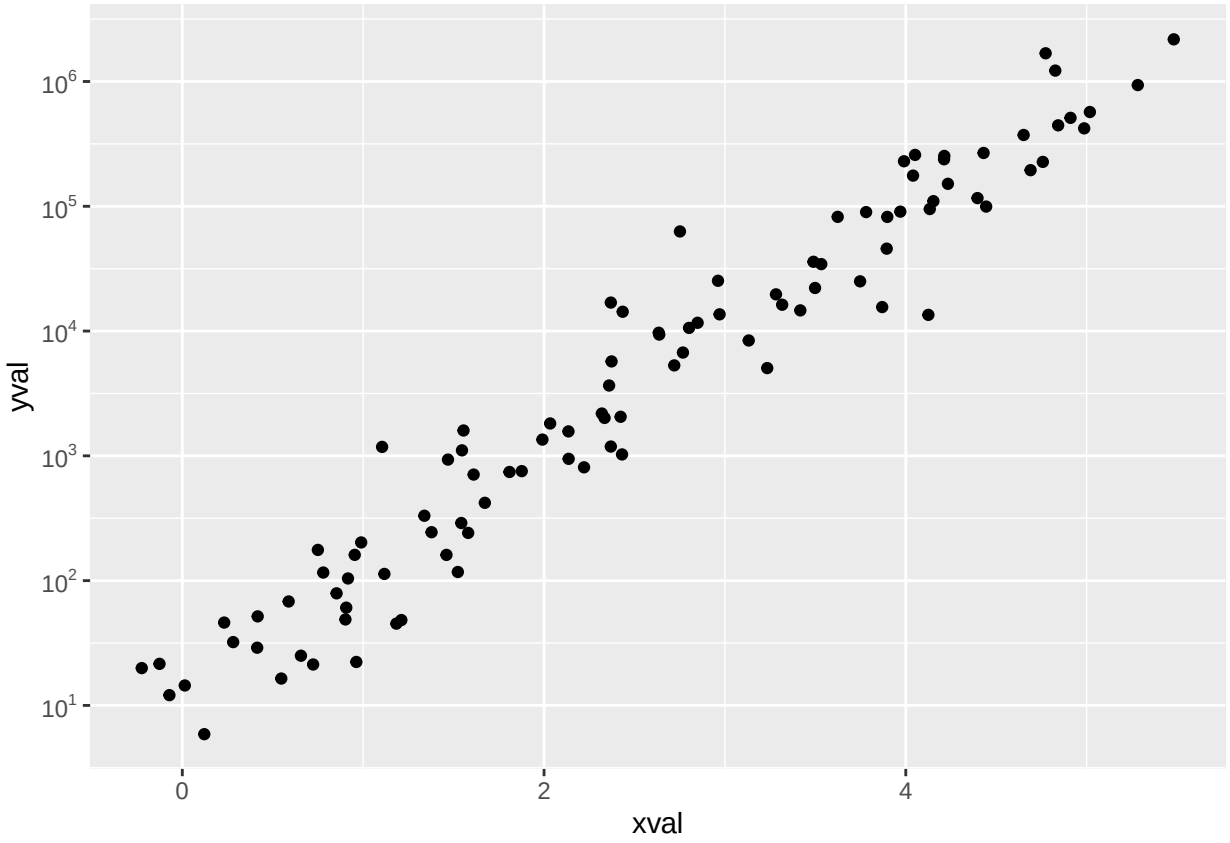

```
sp + scale_y_log10()
```




```
# log2 coordinate transformation (with visually-diminishing spacing)
sp + coord_trans(y="log10")
```




```
library(scales)      # Need the scales package
sp + scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                   labels = trans_format("log10", math_format(10^.x)))
```

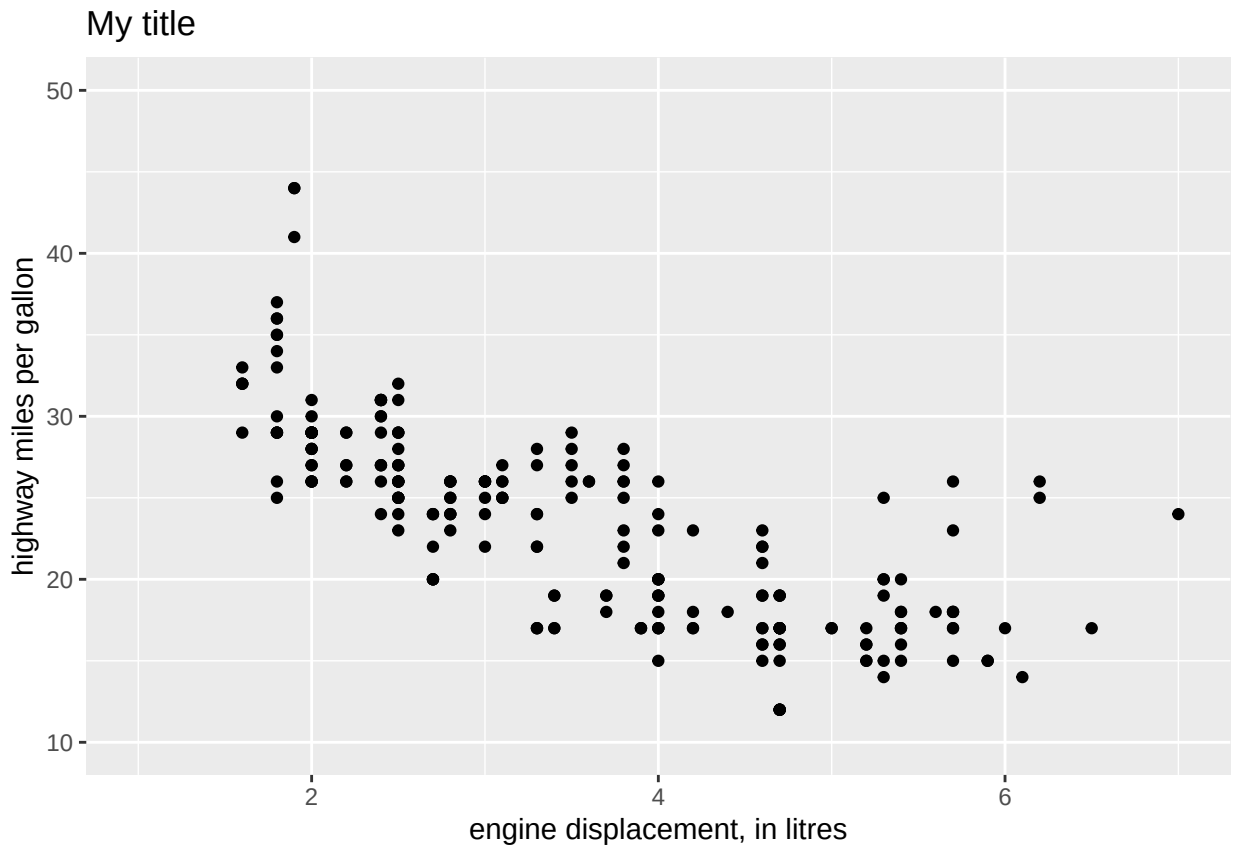


Setting axis limits and labeling scales

- Axes visualize the scales for the aesthetics x and y. To adjust the axes use:
  - `lims`, `xlim`, `ylim`: set axis limits
  - `expand_limits`: extend limits of scales for various aethetics
  - `xlab`, `ylab`, `ggtitle`, `labs`: give labels (titles) to x-axis, y-axis, or graph; labs can set labels for all aesthetics and title

```
p <- ggplot(data = mpg, aes(x = displ, y = hwy)) + geom_point()
```

```
p + lims(x=c(1,7), y=c(10,50)) +labs(x= "engine displacement, in litres", y = "highway miles per gallon
  ggtitle("My title")
```



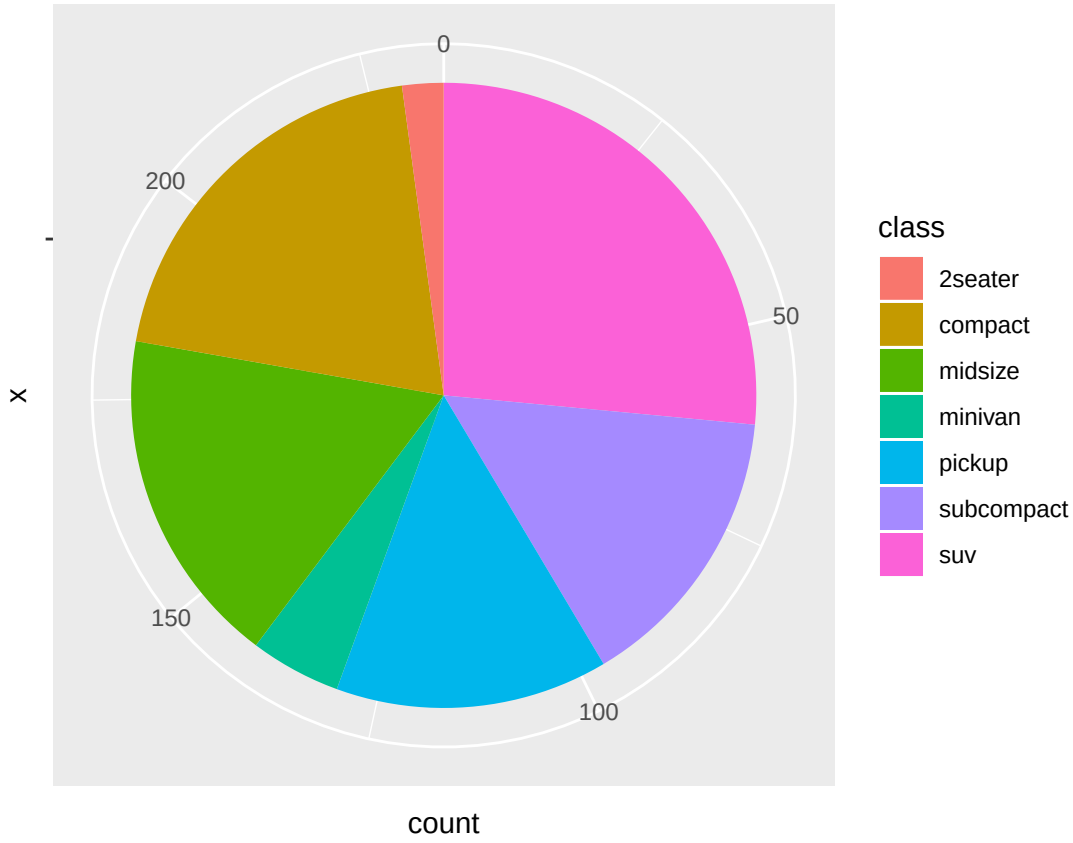
Coordinate systems

Coordinate systems

- Coordinate systems define the planes on which objects are positioned in space on the plot.
- Most plots use Cartesian coordinate systems, as do all the plots in the seminar.
- ggplot2 provides multiple coordinate systems, including polar, flipped Carteisan and map projections.

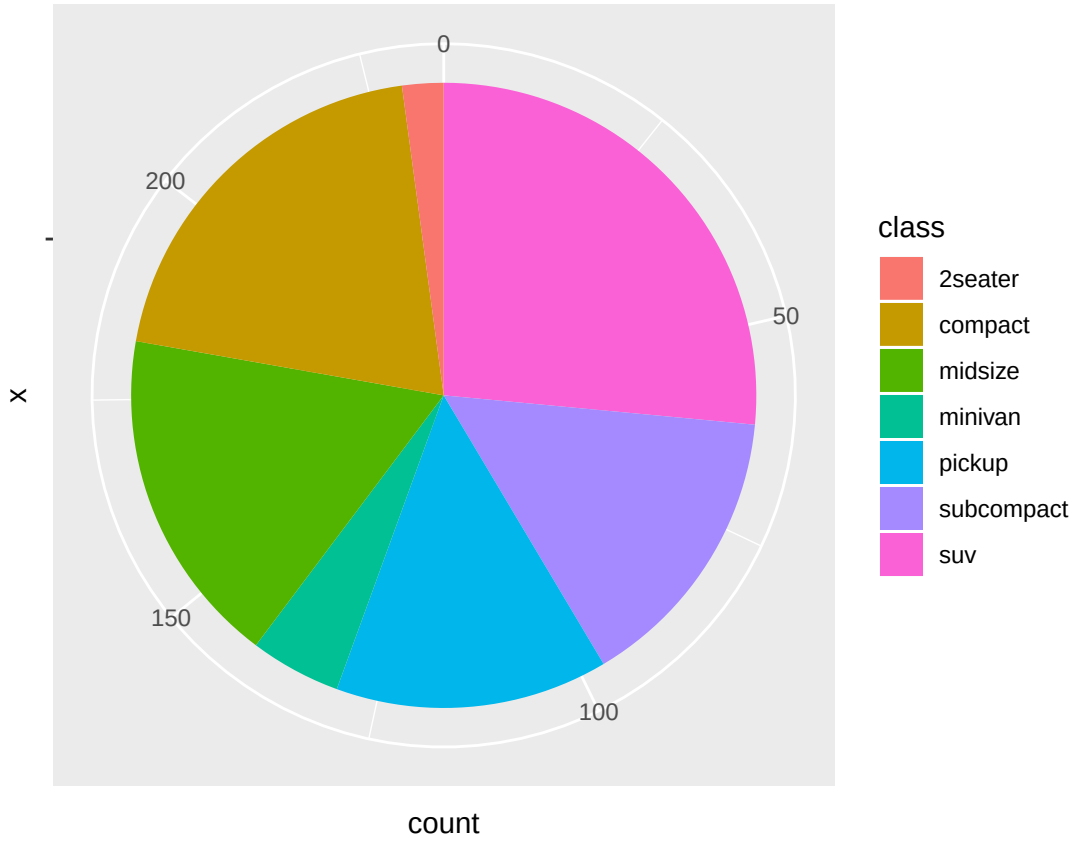
pie chart

```
p <- ggplot(mpg, aes(x="", fill=class))
p + geom_bar() + coord_polar(theta="y", start=0)
```

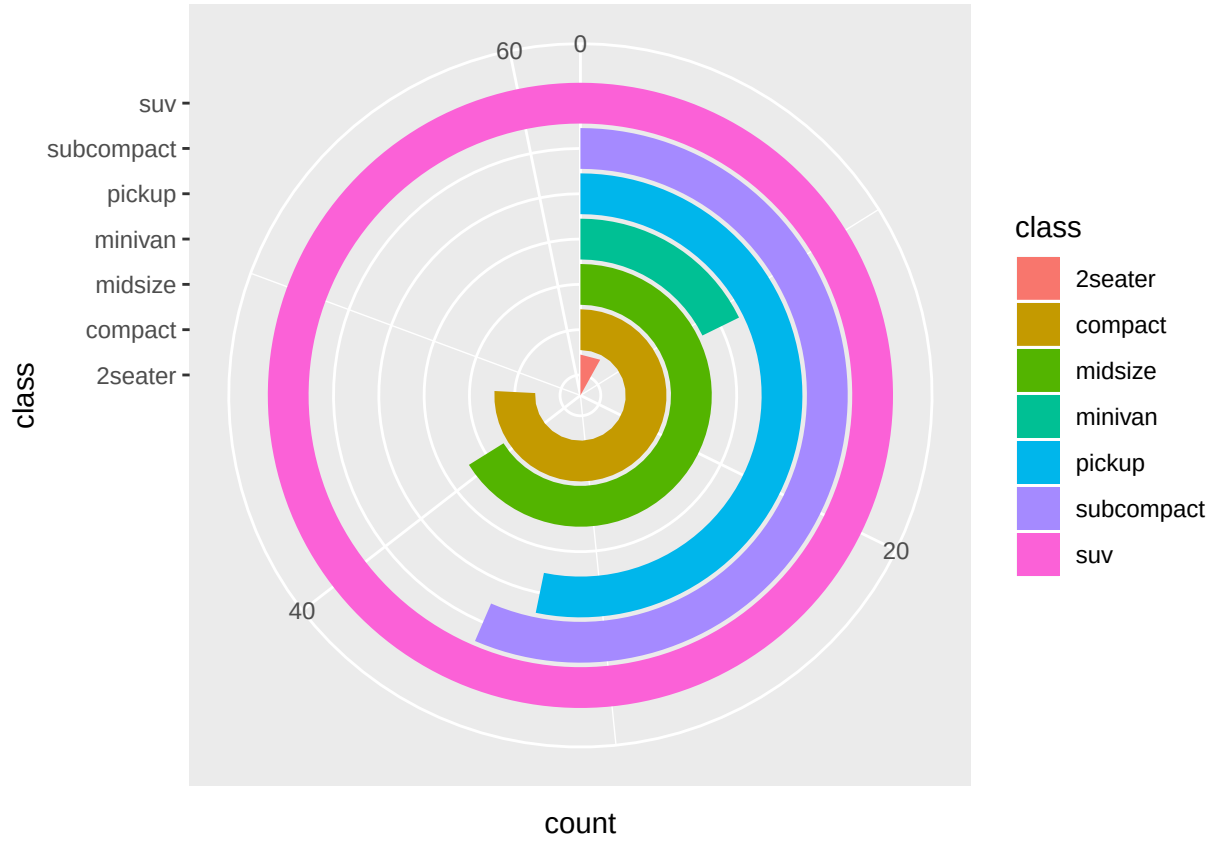


polar bar chart

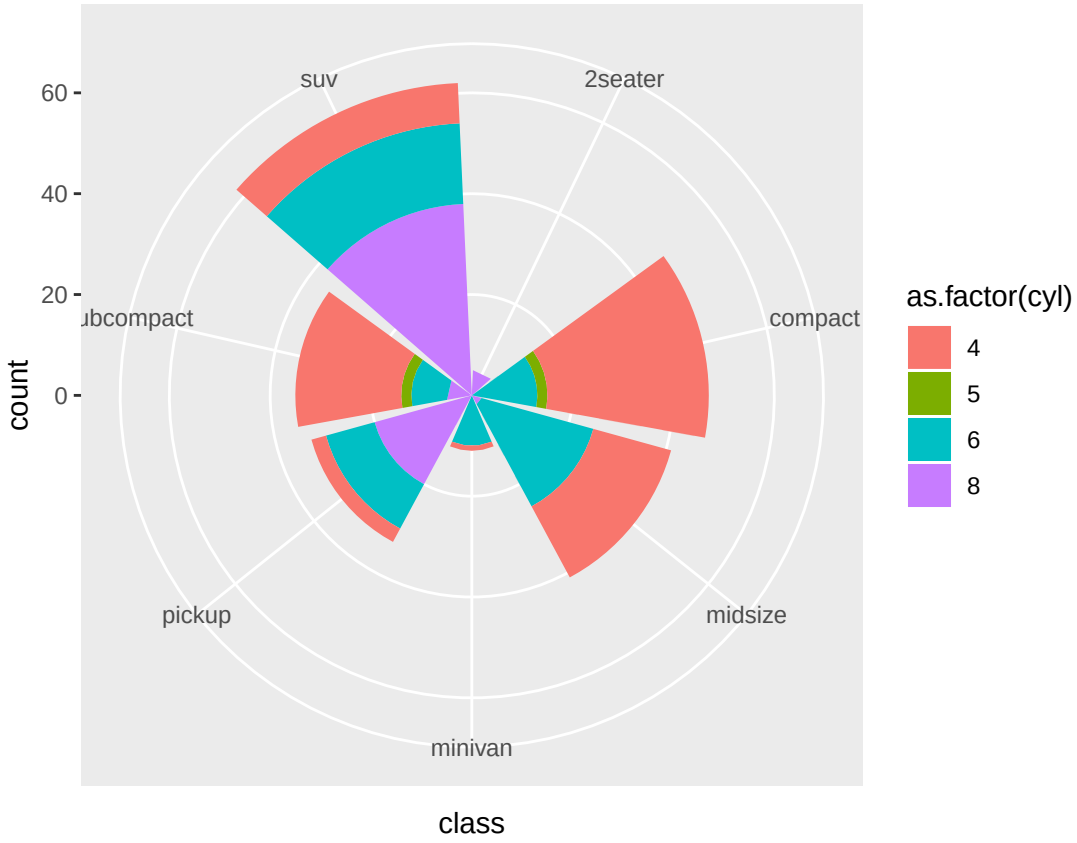
```
p <- ggplot(mpg, aes(x="", fill=class))
p + geom_bar() + coord_polar(theta="y", start=0)
```




```
p <- ggplot(mpg, aes(x=class, fill=class))
p + geom_bar() + coord_polar(theta="y", start=0)
```




```
p <- ggplot(mpg, aes(x=class, fill=as.factor(cyl)))
p + geom_bar() + coord_polar()
```

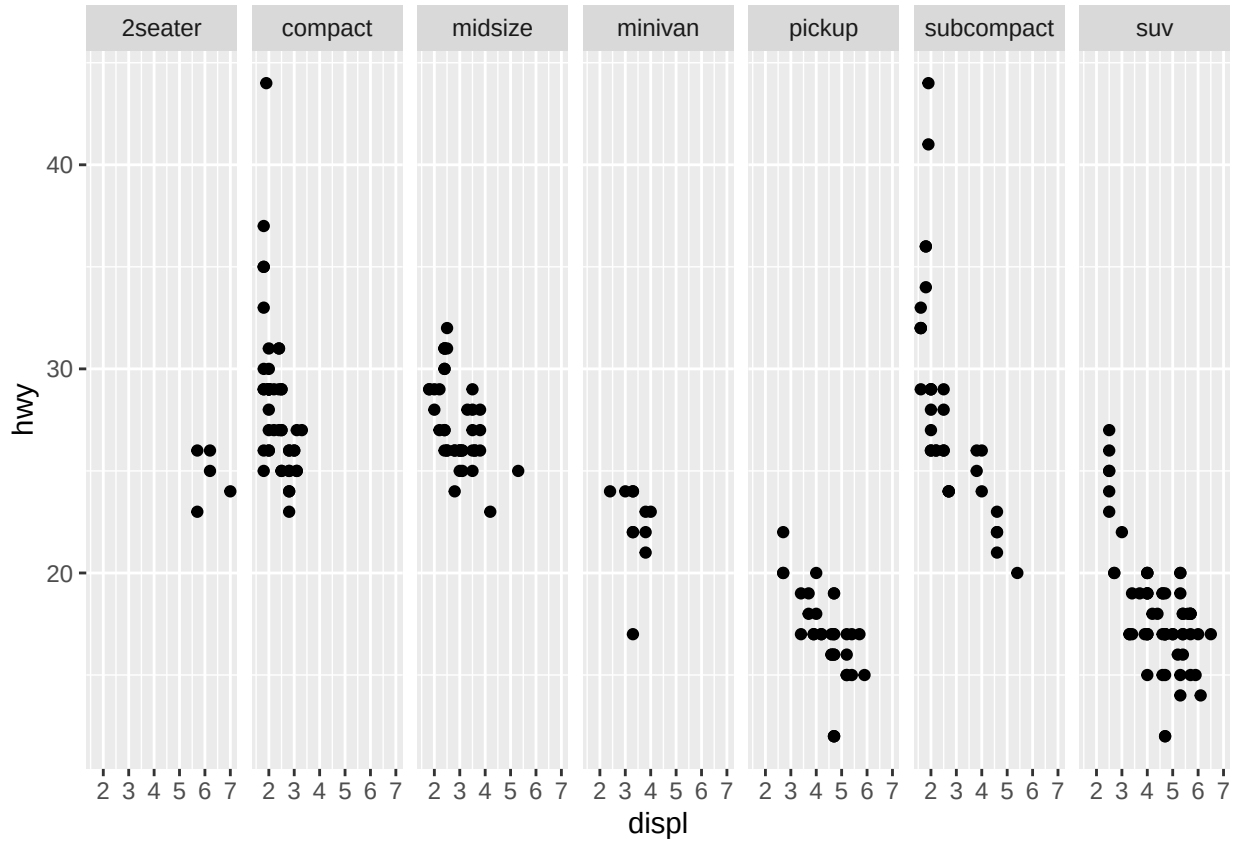


Facet

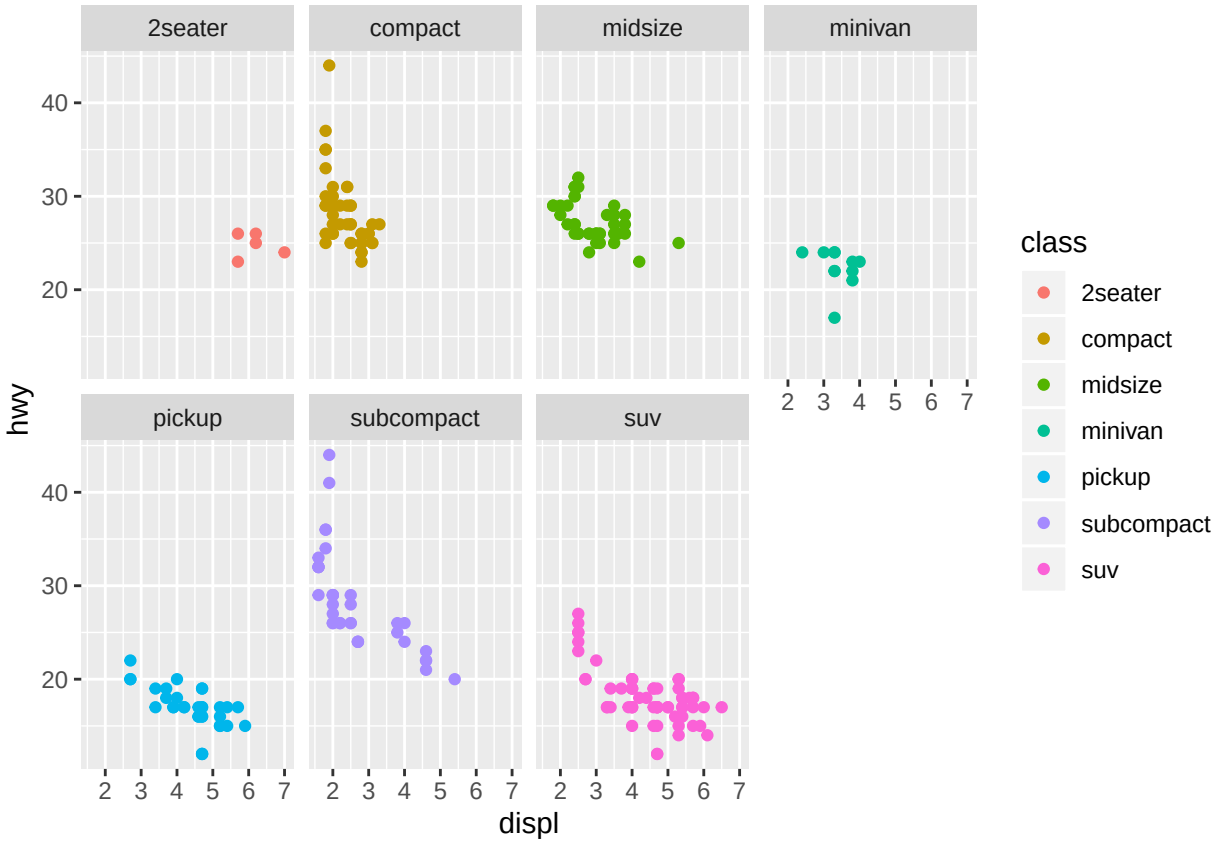
Faceting (paneling)

- Split plots into small multiples (panels) with the faceting functions, `facet_wrap()` and `facet_grid()`. The resulting graph shows how each plot varies along the faceting variable(s).
- `facet_wrap` wraps a ribbon of plots into a multirow panel of plots. The number of rows and columns can be specified. Multiple faceting variables can be separated by +.

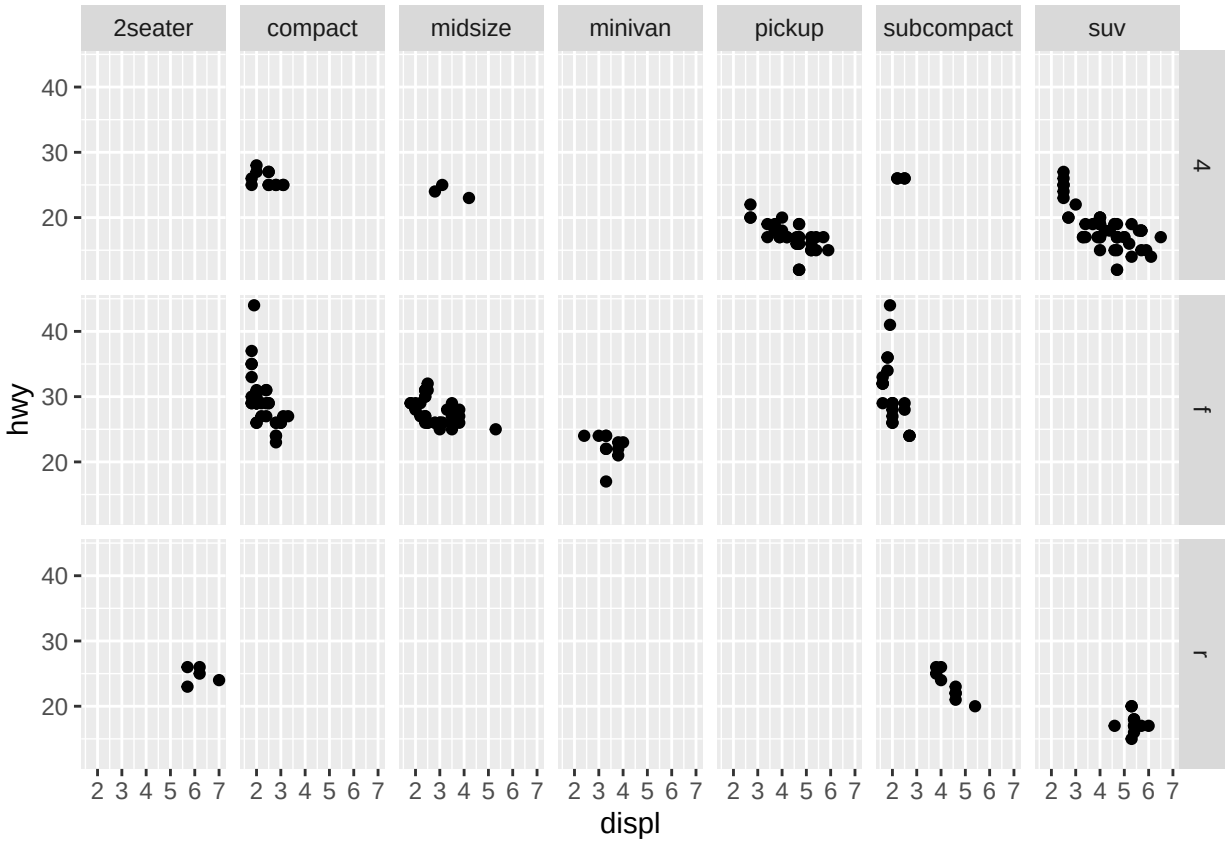
```
ggplot(data = mpg, aes(x = displ, y = hwy)) + geom_point() +
   facet_grid(~ class)
```



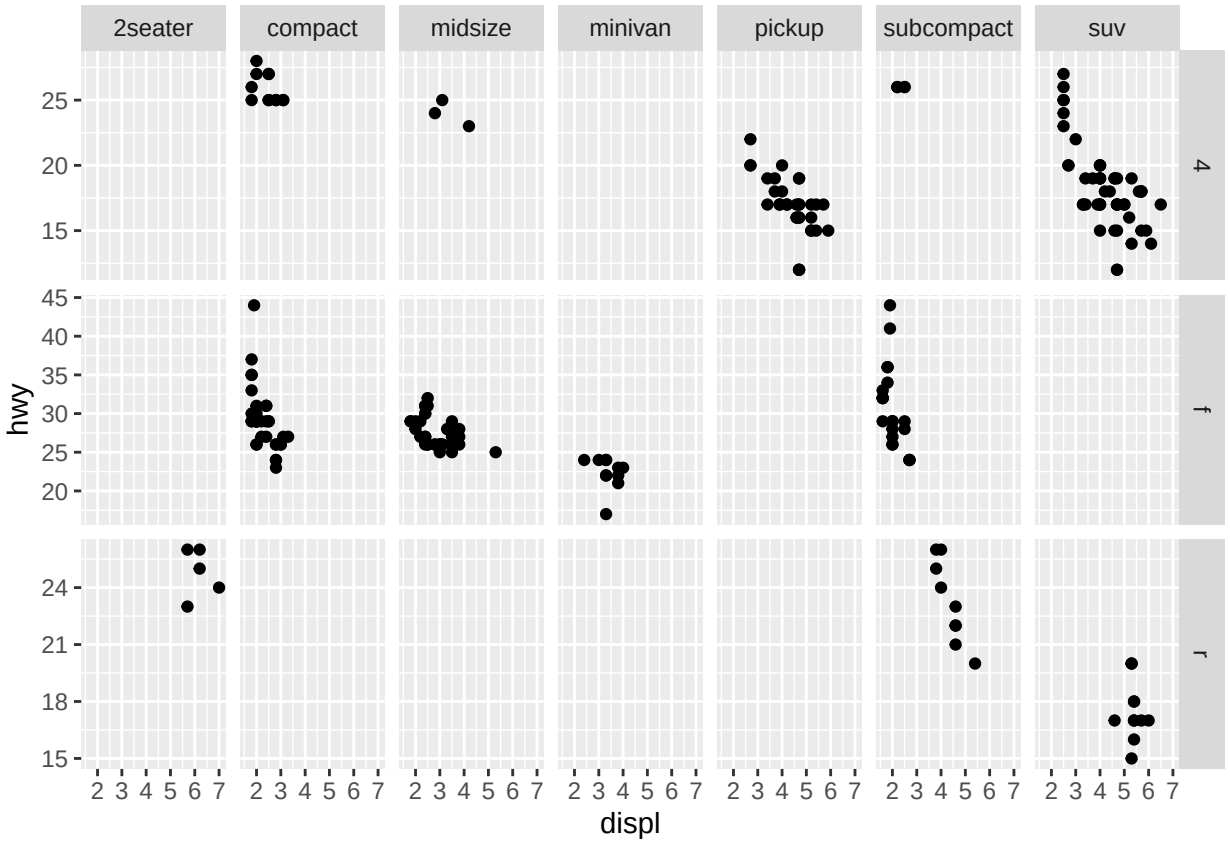

```
ggplot(data = mpg, aes(x = displ, y = hwy, col=class)) + geom_point() +
   facet_wrap(~ class, nrow = 2) # panel by class
```



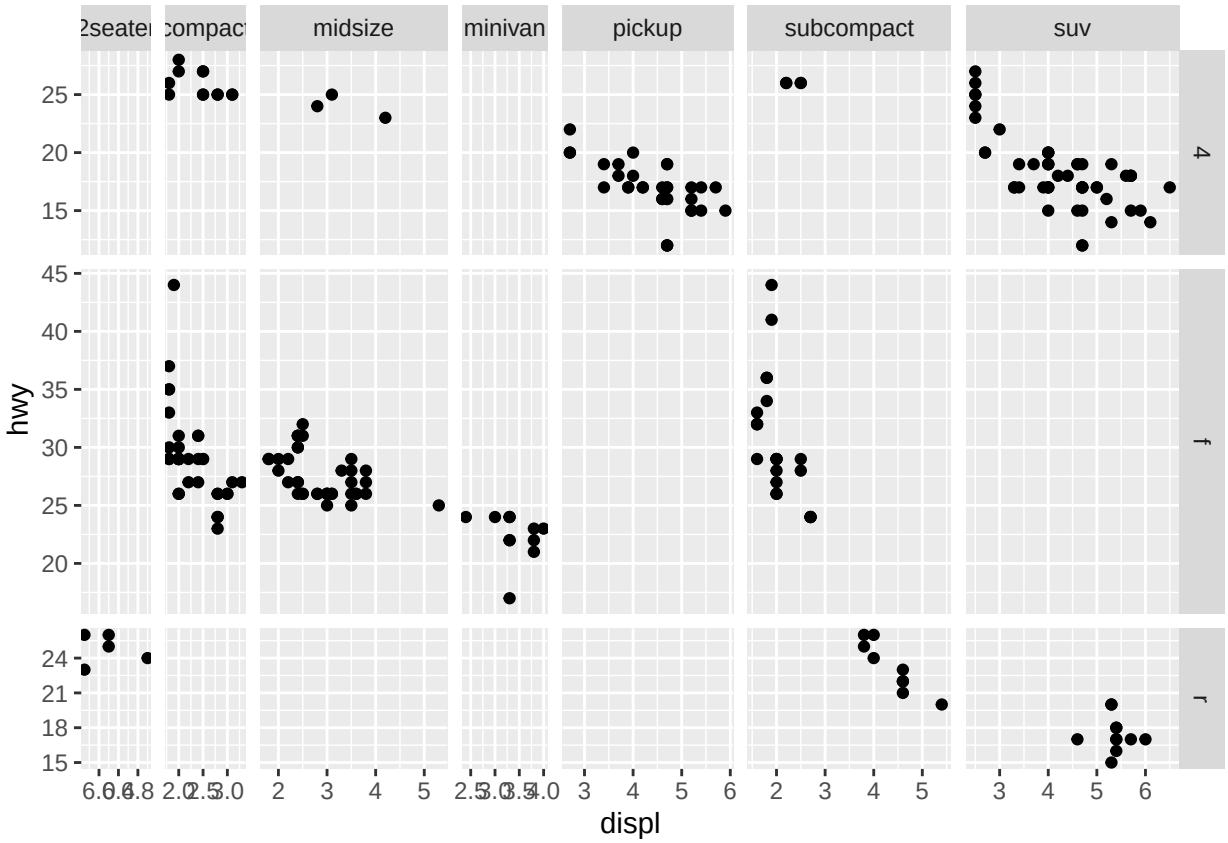

```
ggplot(data = mpg, aes(x = displ, y = hwy)) + geom_point() +
   facet_grid(drv ~ class)
```




```
ggplot(data = mpg, aes(x = displ, y = hwy)) + geom_point() +
   facet_grid(drv ~ class, scales = "free_y")
```




```
ggplot(data = mpg, aes(x = displ, y = hwy)) + geom_point() +
   facet_grid(drv ~ class, scales = "free", space="free")
```



Multiple plot

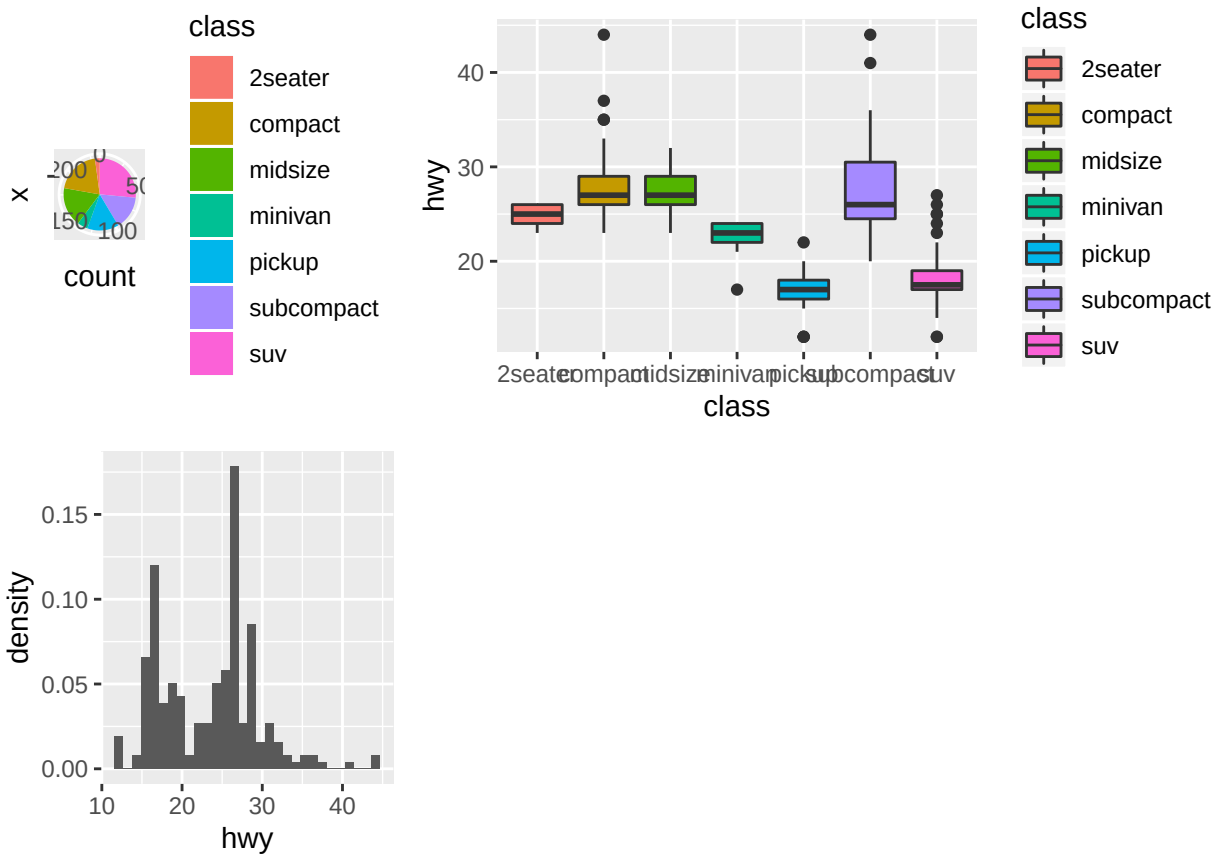
```
library("gridExtra")
```

```
## Warning: package 'gridExtra' was built under R version 3.6.3
```

```
p1 <- ggplot(mpg, aes(x="", fill=class))+geom_bar()+coord_polar(theta="y")
p2 <- ggplot(mpg, aes(x=class, y=hwy, fill=class))+geom_boxplot()
p3 <- ggplot(mpg, aes(x=hwy,y=..density..)) + geom_histogram()
```

```
grid.arrange(p1, p2, p3, ncol=2, widths=c(1,2))
```

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



Other packages for multiple plot

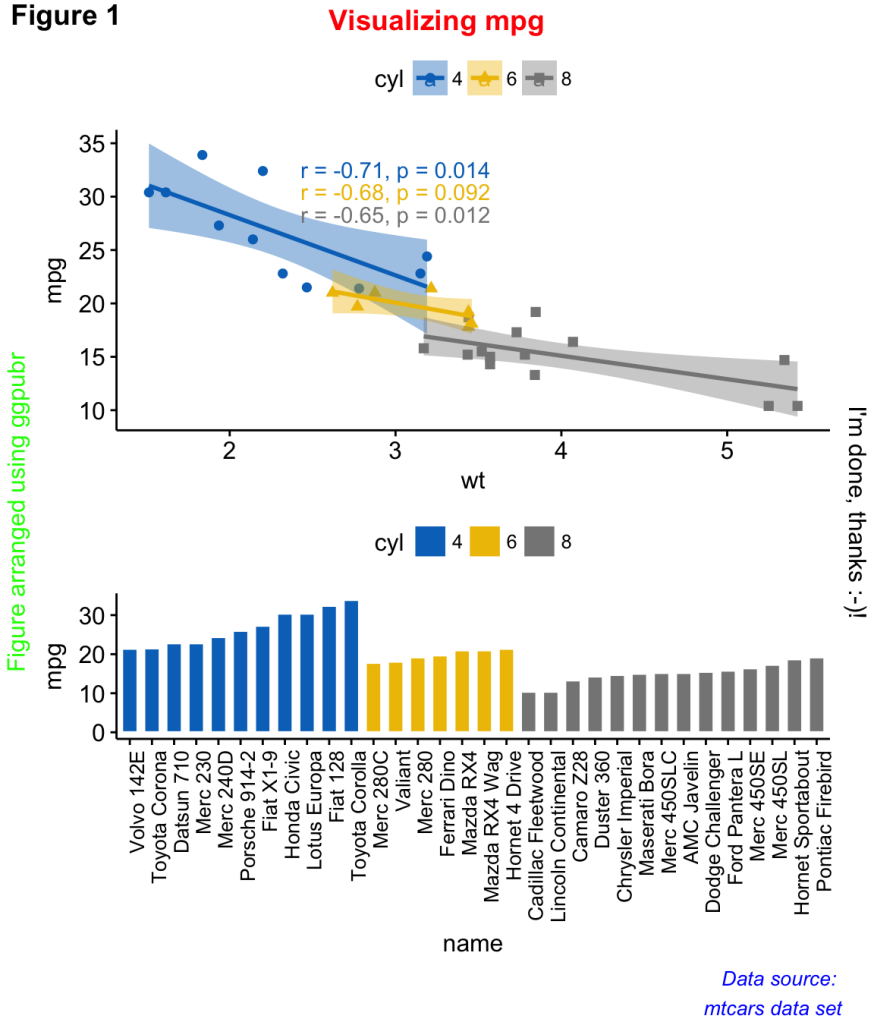
- `cowplot`
- `ggpubr`

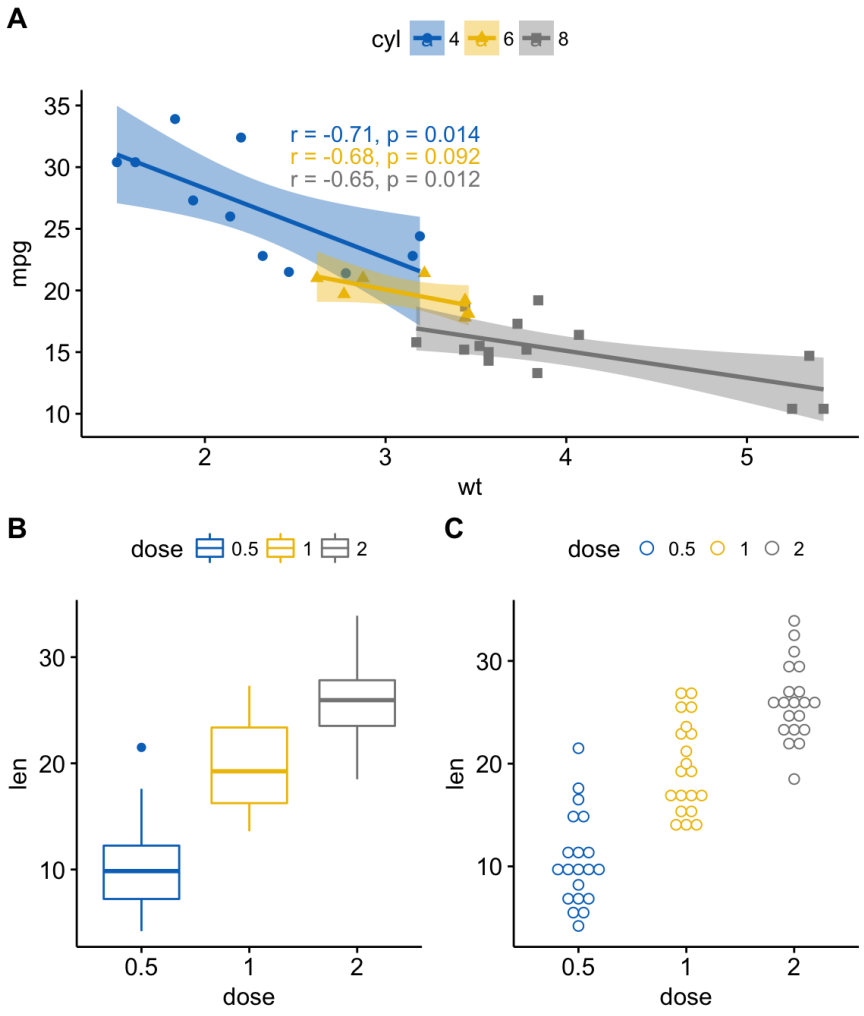
Themes

Themes

- **Themes** control elements of the graph not related to the data. For example:
  - background color
  - size of fonts
  - gridlines
  - color of labels
- These data-independent graph elements are known as **theme elements**.

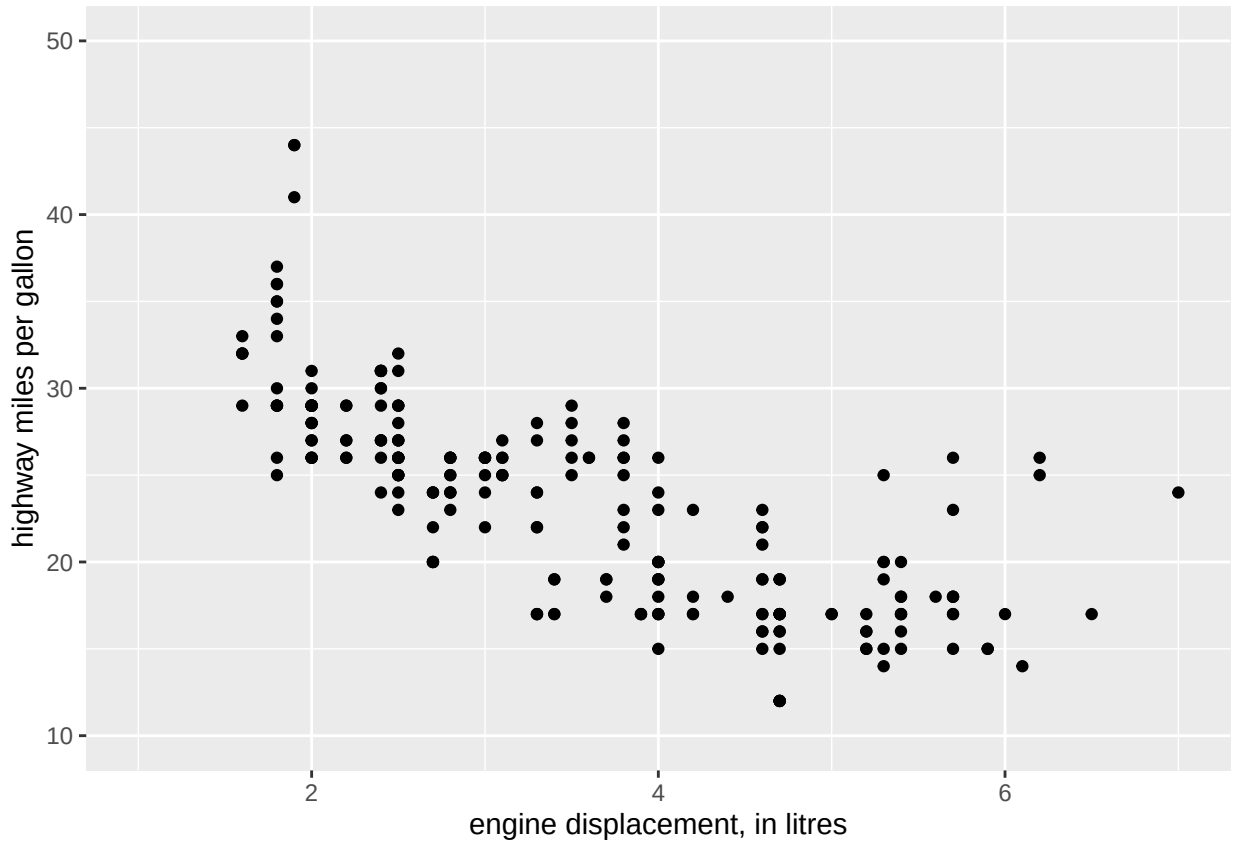
Figure 1


Visualizing mpg
cyl 4 6 8
r = -0.71, p = 0.014
r = -0.68, p = 0.092
r = -0.65, p = 0.012
mpg
wt
cyl 4 6 8
mpg
name
Volvo 142E
Toyota Corona
Datsun 710
Merc 230
Merc 240D
Porsche 914-2
Fiat X1-9
Honda Civic
Lotus Europa
Fiat 128
Toyota Corolla
Merc 280C
Valiant
Merc 280
Ferrari Dino
Mazda RX4
Mazda RX4 Wag
Hornet 4 Drive
Cadillac Fleetwood
Lincoln Continental
Camaro Z28
Duster 360
Chrysler Imperial
Maserati Bora
Merc 450SLC
AMC Javelin
Dodge Challenger
Ford Pantera L
Merc 450SE
Merc 450SL
Hornet Sportabout
Pontiac Firebird
Figure arranged using ggpubr
I'm done, thanks :-) !
Data source:
mtcars data set


A
cyl 4 6 8
r = -0.71, p = 0.014
r = -0.68, p = 0.092
r = -0.65, p = 0.012
mpg
wt
B
dose 0.5 1 2
len
dose
C
dose 0.5 1 2
len
dose

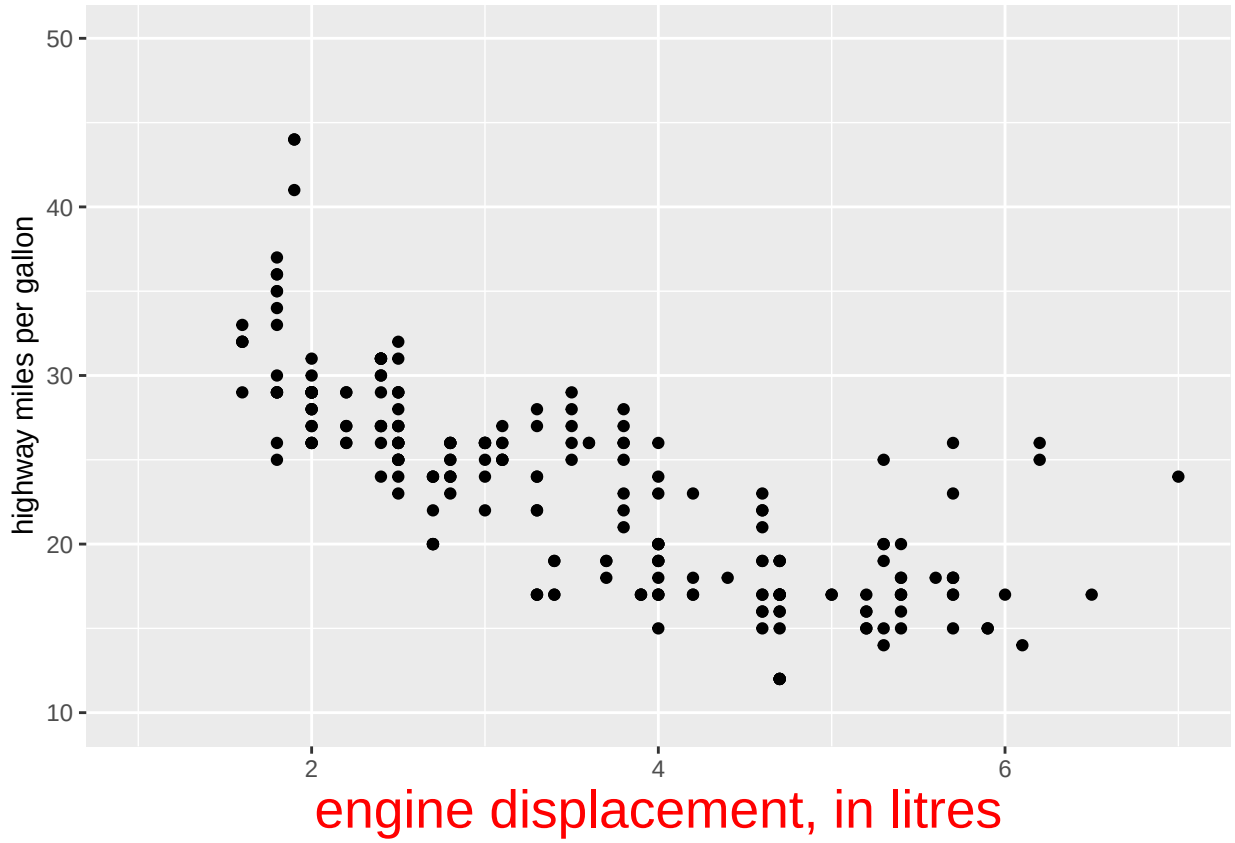
- Within the `theme()` function, adjust the properties of theme elements by specifying the arguments of the `element_` functions.
- Each theme element can be classified as one of three basic elements: a line, a rectangle, or text. Each basic element has a corresponding `element_` function, where graphical aspects of that element can be controlled:
  - `element_line()` - can specify `color`, `size`, `linetype`, etc.
  - `element_rect()` - can specify `fill`, `color`, `size`, etc.
  - `element_text()` - can specify `family`, `size`, `color`, etc.
  - `element_blank()` - removes theme elements from graph

```
pt <- ggplot(data = mpg, aes(x = displ, y = hwy)) + geom_point() +
  lims(x=c(1,7), y=c(10,50)) +
  labs(x= "engine displacement, in litres", y = "highway miles per gallon")
pt
```

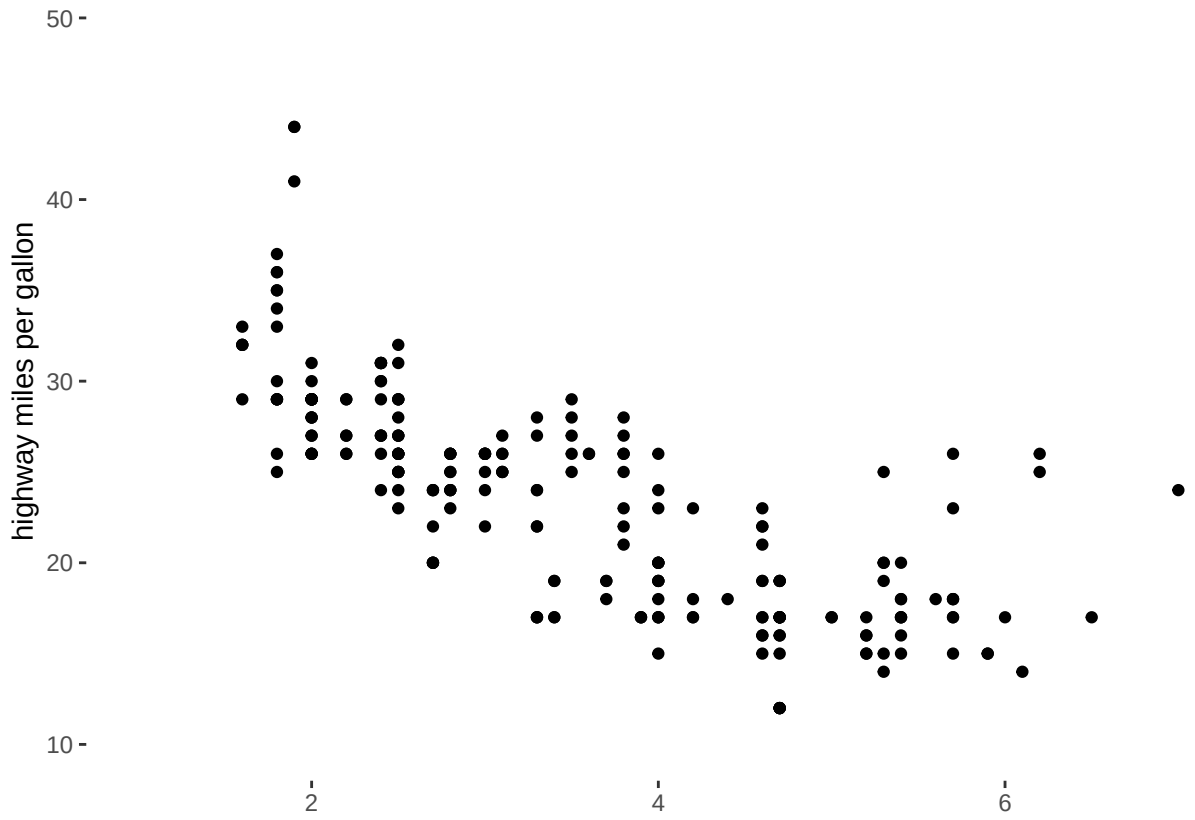



```
pt + theme(panel.background = element_rect(fill = "lightblue"))
```


```
pt + theme(axis.title.x = element_text(size = 20, color = "red"))
```




```
pt + theme(panel.background = element_blank(), axis.title.x = element_blank())
```



Sving plots to files

- `ggsave()` makes saving plots easy. The last plot displayed is saved by default, but we can also save a plot stored to an R object.
- `ggsave` attempts to guess the device to use to save the image from the file extension, so use a meaningful extension. Available devices include eps/ps, tex (pictex), pdf, jpeg, tiff, png, bmp, svg and wmf. The size of the image can be specified as well.

```
ggsave("plot.pdf")
```

```
## Saving 6.5 x 4.5 in image
```

```
#save plot stored in "p" as png and resize
ggsave("myplot.png", plot=p, width=7, height=5, units = "in")
```