ggplot2

Luis Moreno

2/1/2021

Getting Started

`ggplot2` has an underlying grammar, based on the **Grammar of Graphics**, that allows you to **compose graphs by combining independent components**.

`ggplot2` allows you to produce graphics using the same structured thinking that you would use to design an analysis.

The Grammar of Graphics

A graphic maps the data to the **aesthetic attributes** (color, shape, size) of **geometric objects** (points, lines, bars).

Mapping Components

- **Layer**: collection of geometric elements and statistical transformations.
  - **geom**: represents the figures in the plot (points, lines, polygons).
  - **stats**: summarize the data.
- **Scale**: map values in the data space to values in the aesthetic space.
- **Coordinate system**: describes how data coordinates are mapped to the plane of the graphic and provides axes and grid lines to help read the graph.
- **Facet**: specifies how to break up and display subsets of data as small multiples (also known as conditioning or *latticing/trellising*).
- **Theme**: controls the finer points of display (font size, background color, etc.).

NOTE: `ggplot2` only describes static graphs, for dynamic and interactive graphics, look for `ggvis`. **Dianne Cook** and **Deborah F. Swayne** provides an excellent introduction to the interactive graphics package `GGobi`. GGobi can be connected to R with the `rggobi package`.

Web Visualization Tools for R

htmlwidgets for R: use `JavaScript` visualization libraries at the R console, R Markdown or Shiny.

Packages built on top of `htmlwidgets`:

- leaflet: maps,
- dygraph: time series,
- networkD3: networks.

ggplot() Key Components

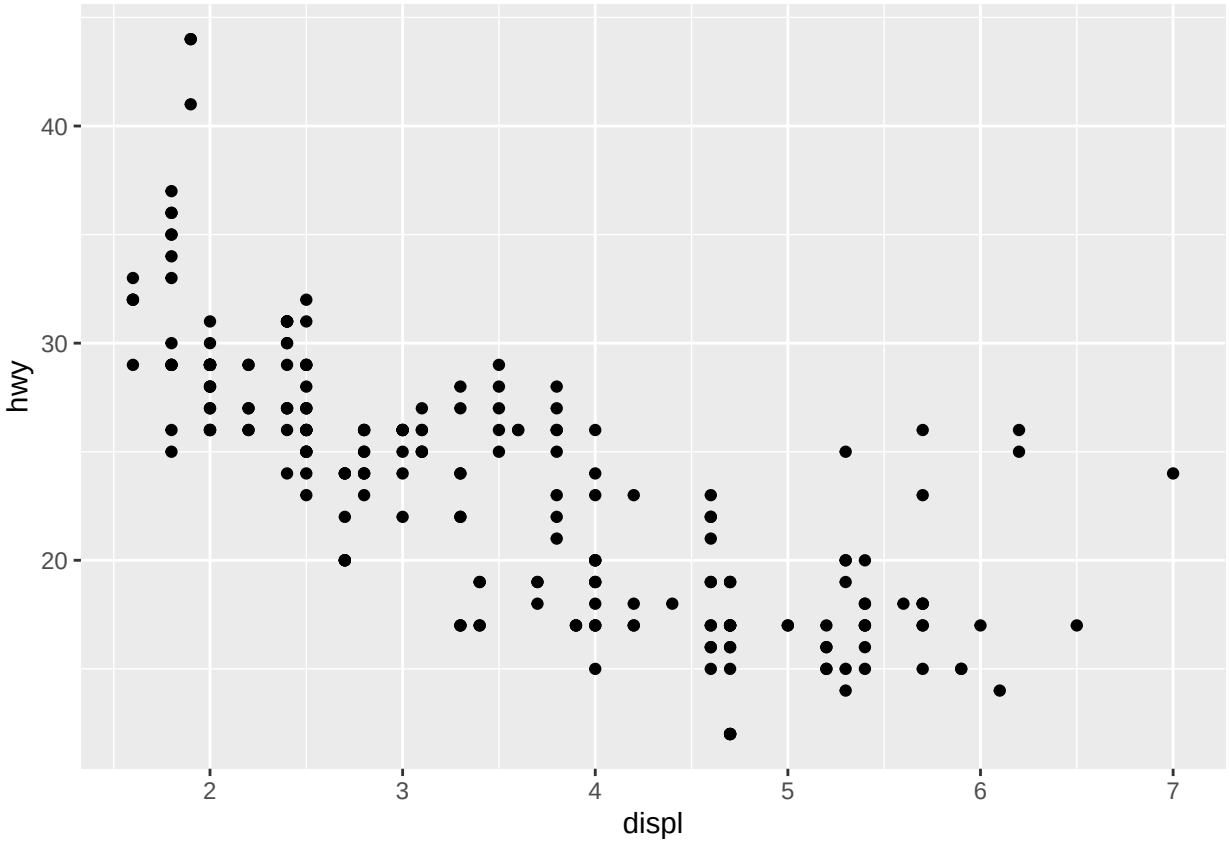
1. Data,
2. Aesthetic mappings between variables in the data and visual properties,
3. At least one layer describing how to render each observation. Layers are usually created with a geom function.

NOTE: **data** and **aesthetic mappings** are defined inside the `ggplot()` function, while layers are added with the `+` symbol.

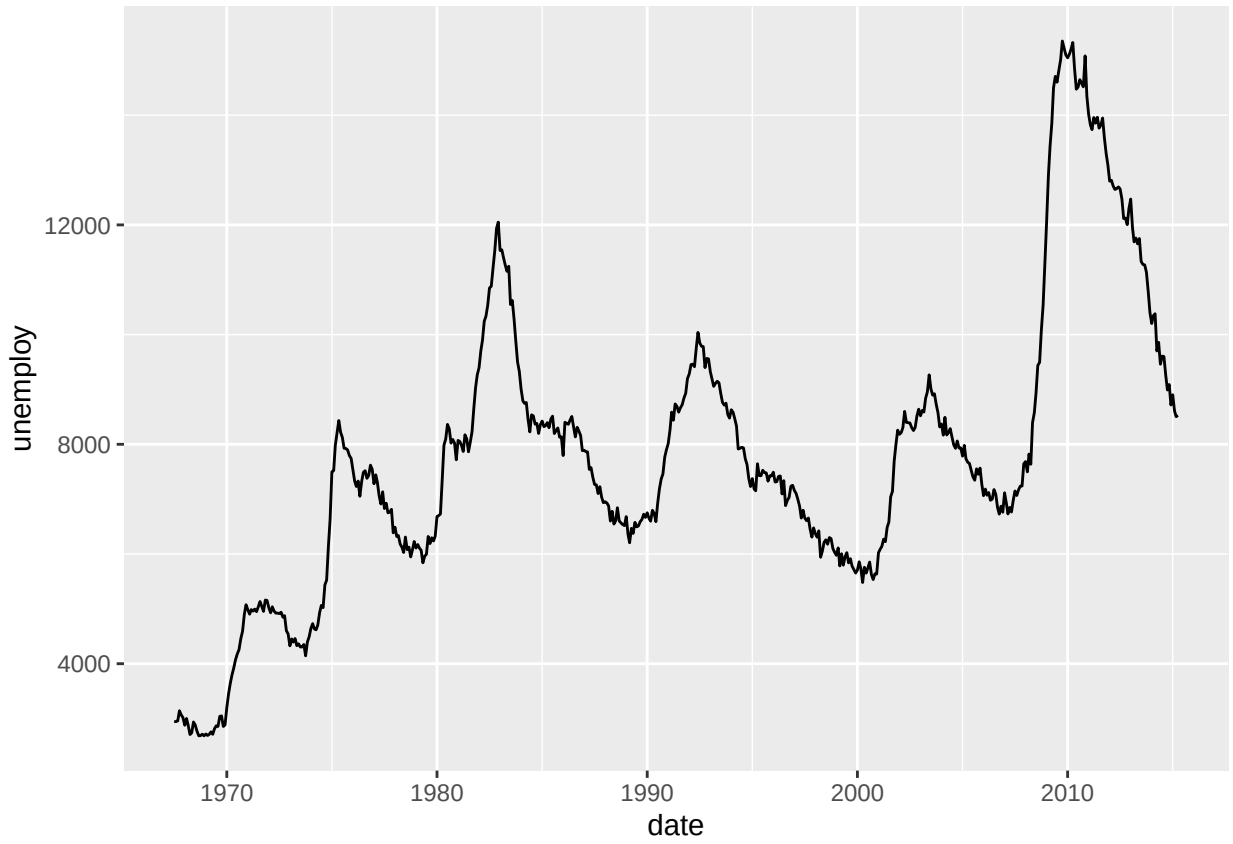
The order of the axis inside `aes()` is always `x` followed by `y`.

```
# Load required packages
library(ggplot2)
```

```
# Scatterplot
ggplot(mpg, aes(displ, hwy)) +
    geom_point()
```

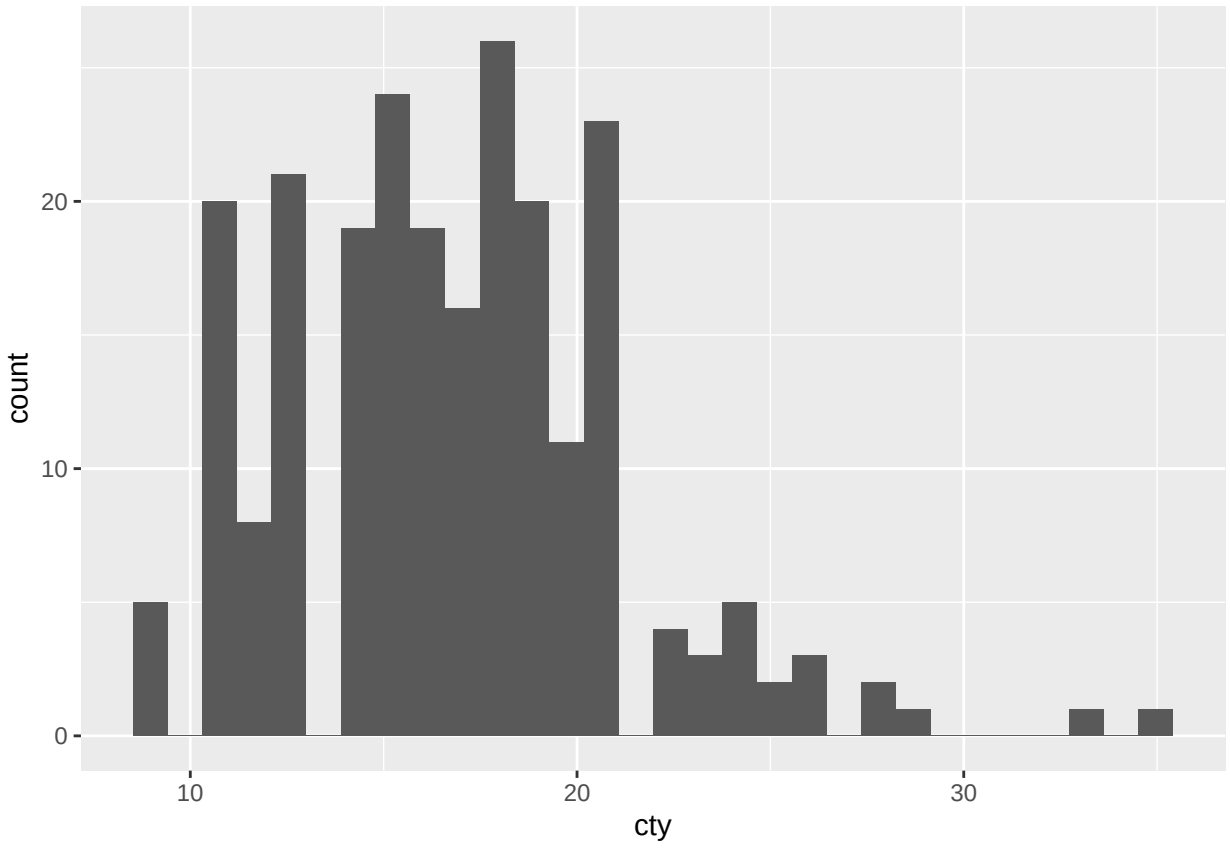



```
# Line plot
ggplot(economics, aes(date, unemploy)) +
    geom_line()
```




```
# Histogram
ggplot(mpg, aes(cty)) +
     geom_histogram()
```

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



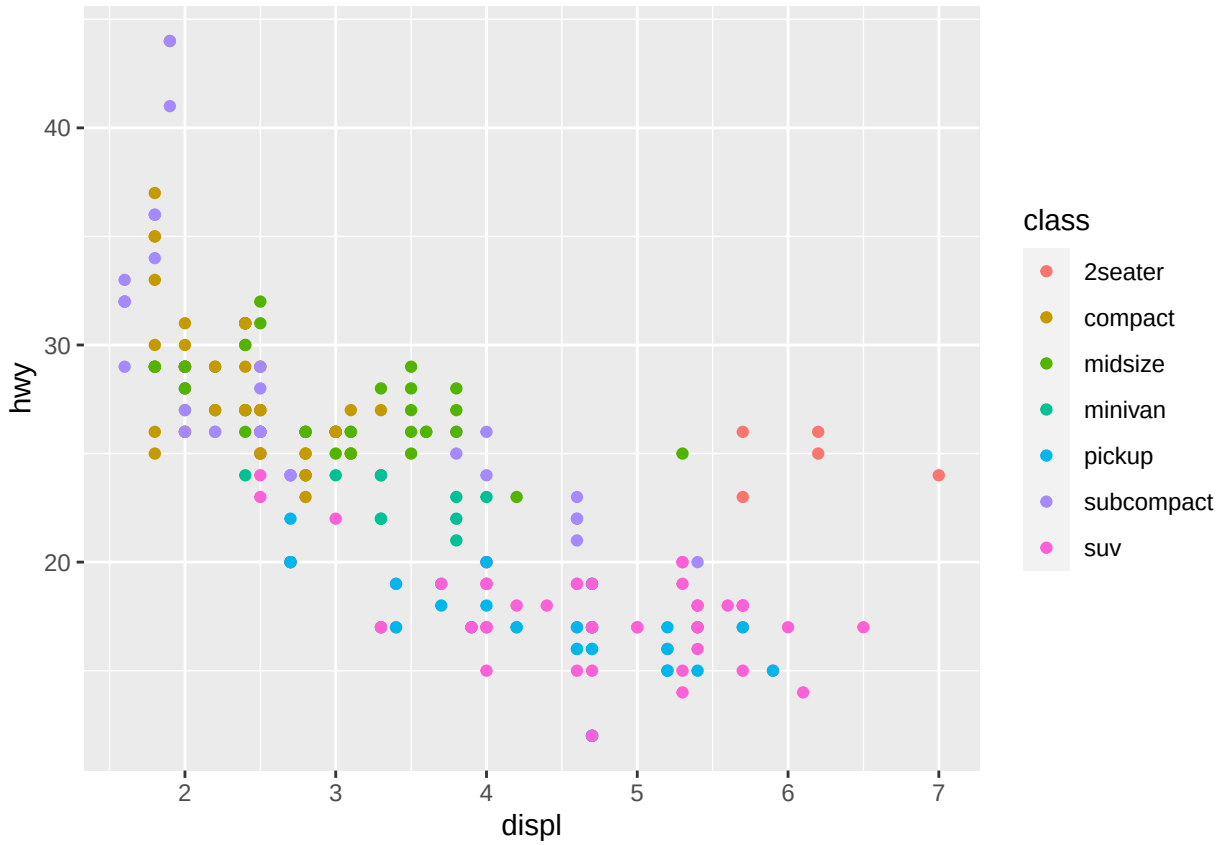
Main Aesthetic Attributes

Colors, shapes and size of a plot are defined inside the `aes()` function.

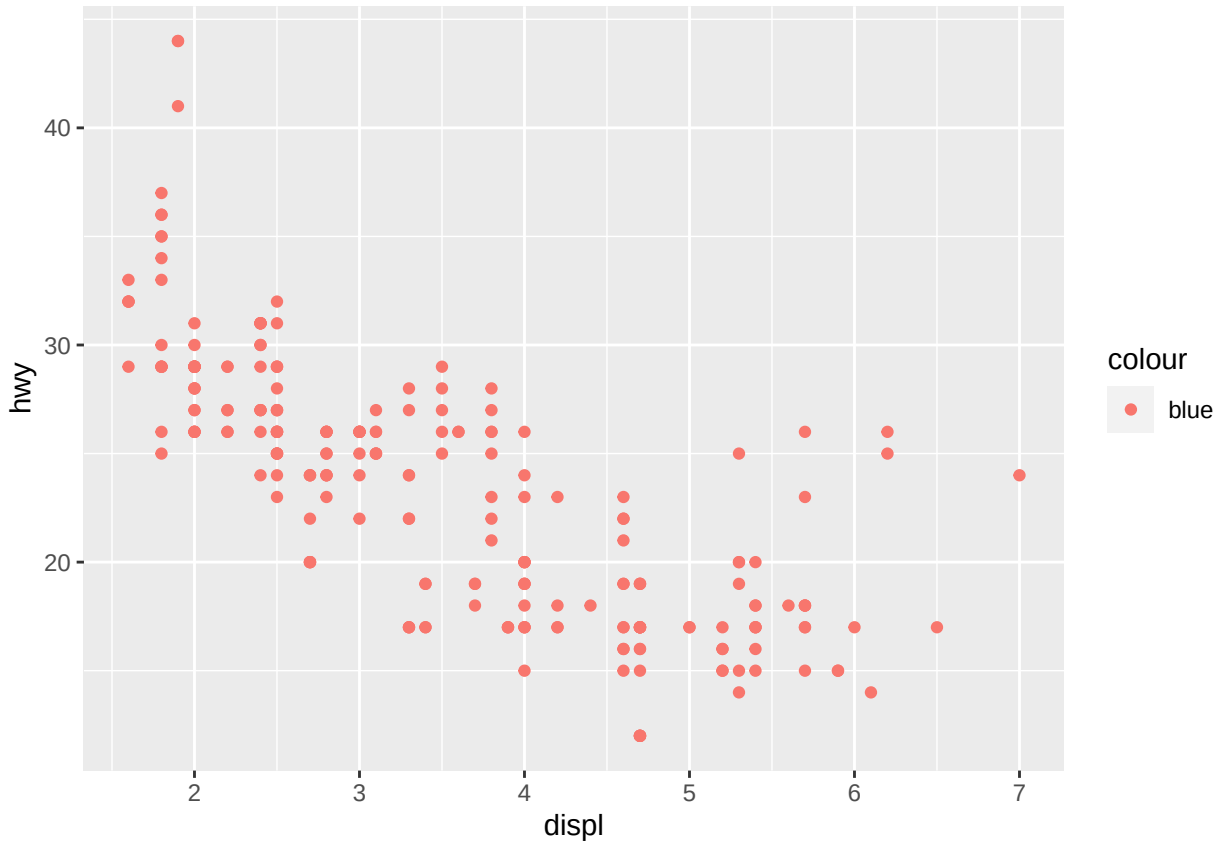
A **scale** is used when passing data as aesthetic attributes.
This scale creates an axis or legend to read the plot.

To set an aesthetic to a fixed value, without scaling it, do so in the individual layer outside of `aes()`.

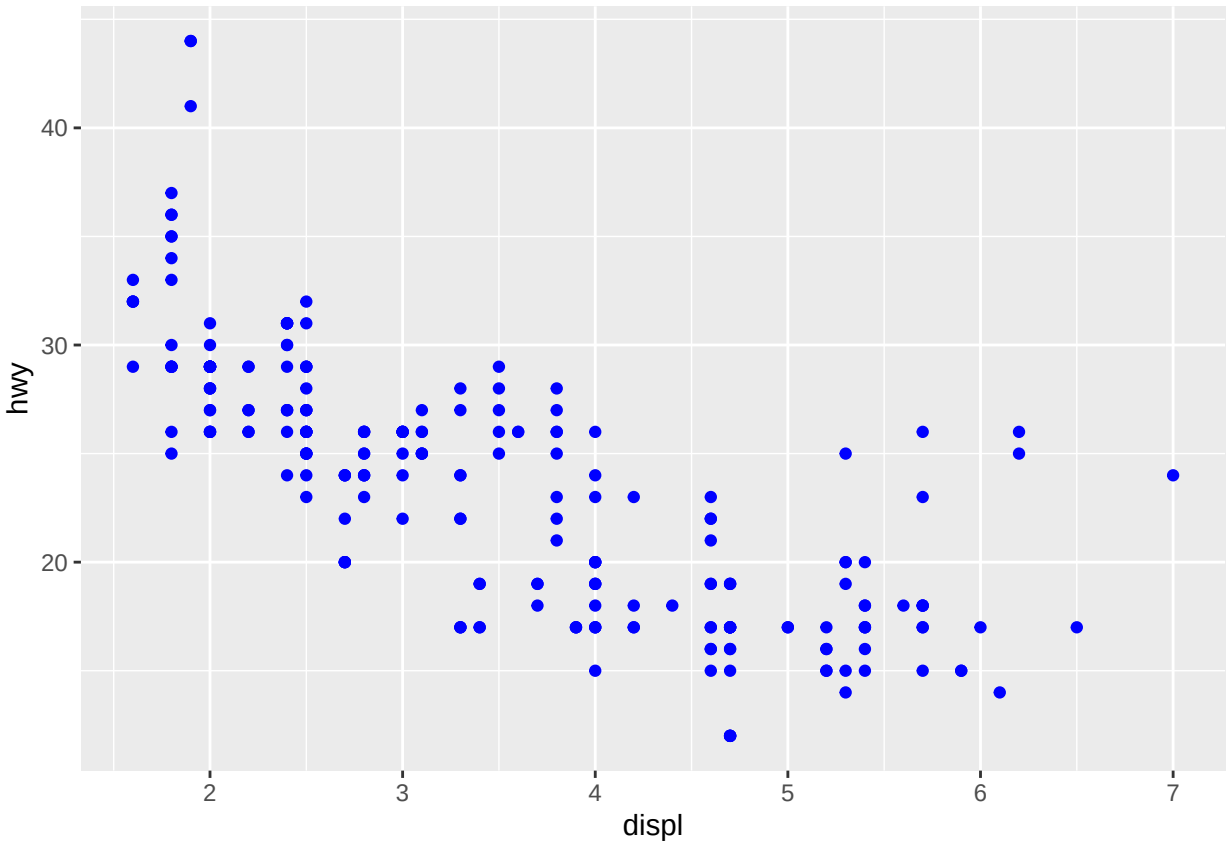
```
# Use a variable or data to set the plot colors
ggplot(mpg, aes(displ, hwy, color = class)) +
     geom_point()
```




```
# Set the name of the color legend for the plot
ggplot(mpg, aes(displ, hwy)) +
     geom_point(aes(color = "blue"))
```




```
# Set up a fixed color for the plot
ggplot(mpg, aes(displ, hwy)) +
     geom_point(color = "blue")
```



Color and shape work well with **categorical variables**, while size works well for **continuous variables**.

If there is too much data it can be hard to distinguish different groups. An alternative solution is to use **faceting**.

T&T: instead of trying to make one very complex plot that shows everything at once, see if you can create a series of simple plots that tell a story, leading the reader from ignorance to knowledge.

Faceting

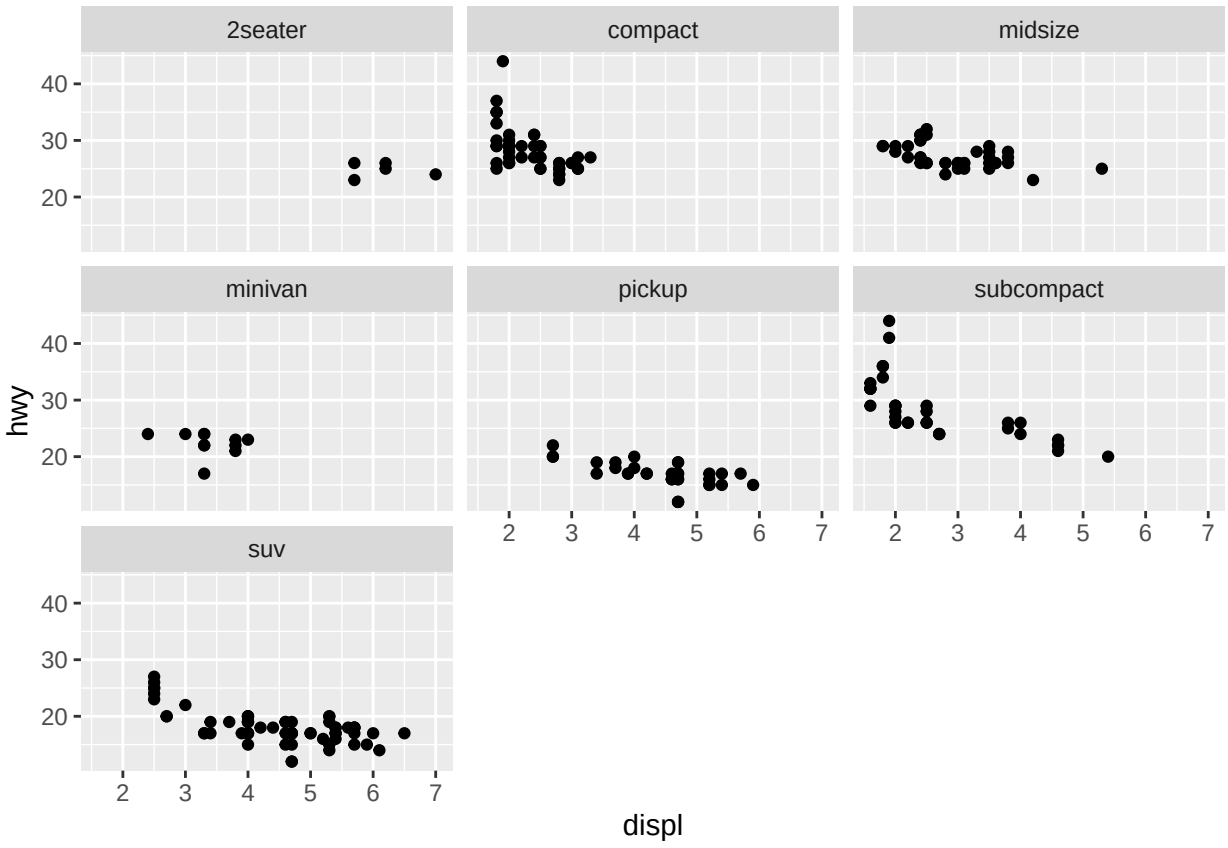
Faceting creates tables of graphics by **splitting the data into subsets** and displaying the **same graph for each subset**.

Types of faceting:

- grid,
- wrapped.

Add a faceting specification with `facet_wrap()` with the variable name as parameter, preceded by `~`.

```
# Plot a table graphic by variable
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  facet_wrap(~class)
```

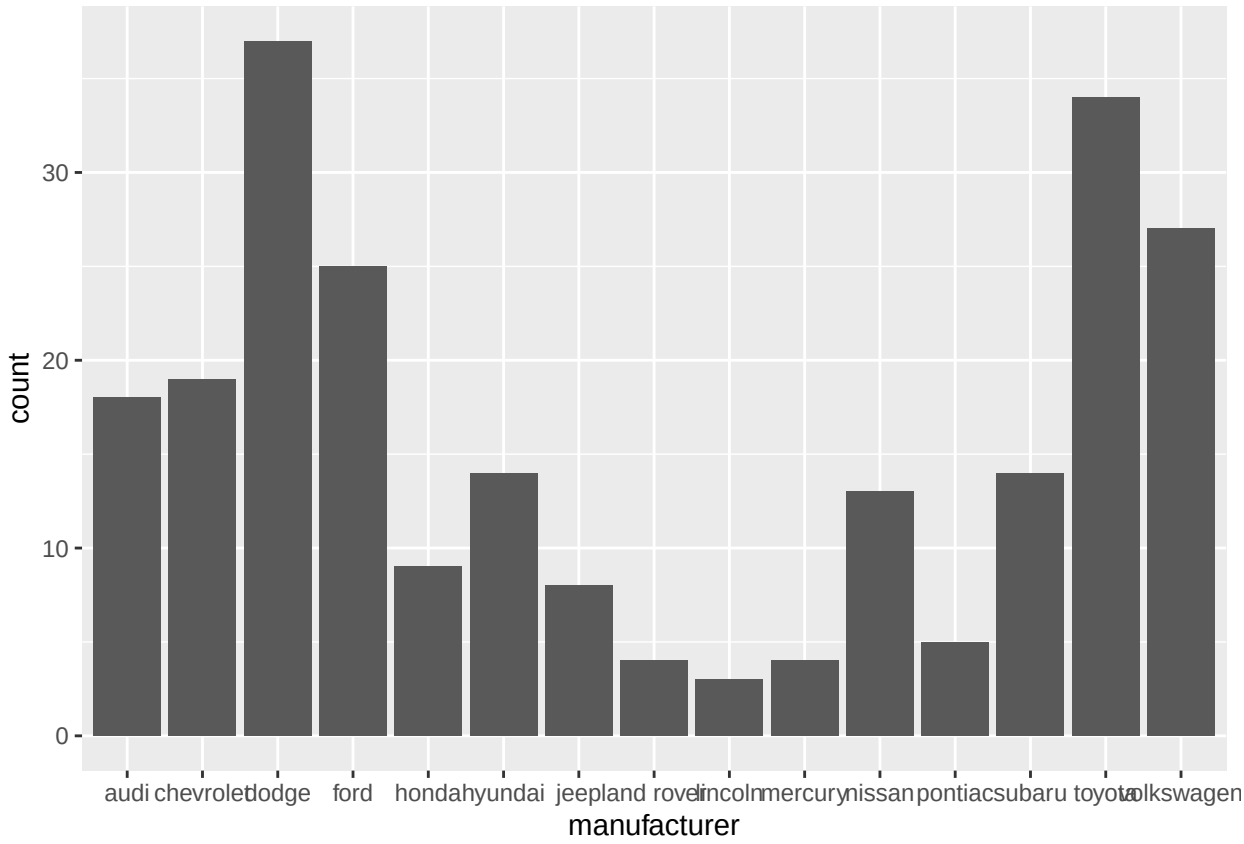


Plot Geoms

Main Plots

- `geom_point()`: it creates a scatterplot for visualizing the relationship between two continuous variables.
- `geom_boxplot()`: produces a box-and-whisker plot to summarize the distribution of a set of points.
- `geom_histogram()` and `geom_freqpoly()` show the distribution of continuous variables.
- `geom_bar()`: shows the distribution of categorical variables.
- `geom_path()` and `geom_line()` draw lines between the data points. A line plot produces lines that travel from left to right, while paths can go in any direction. Lines are typically used to explore how things change over time.

```
ggplot(mpg, aes(manufacturer)) +
  geom_bar()
```



Bar Plots

They're typically used to display numeric values (on the y-axis), for different categories (on the x-axis).

Sometimes the bar heights represent **counts of cases** in the data set, and sometimes they represent **values** in the data set.

The default behavior of `geom_bar()` is to count the rows for each x value. It doesn't expect a y-value.

In `geom_bar()`, how aggregation is to be performed is specified as an argument:

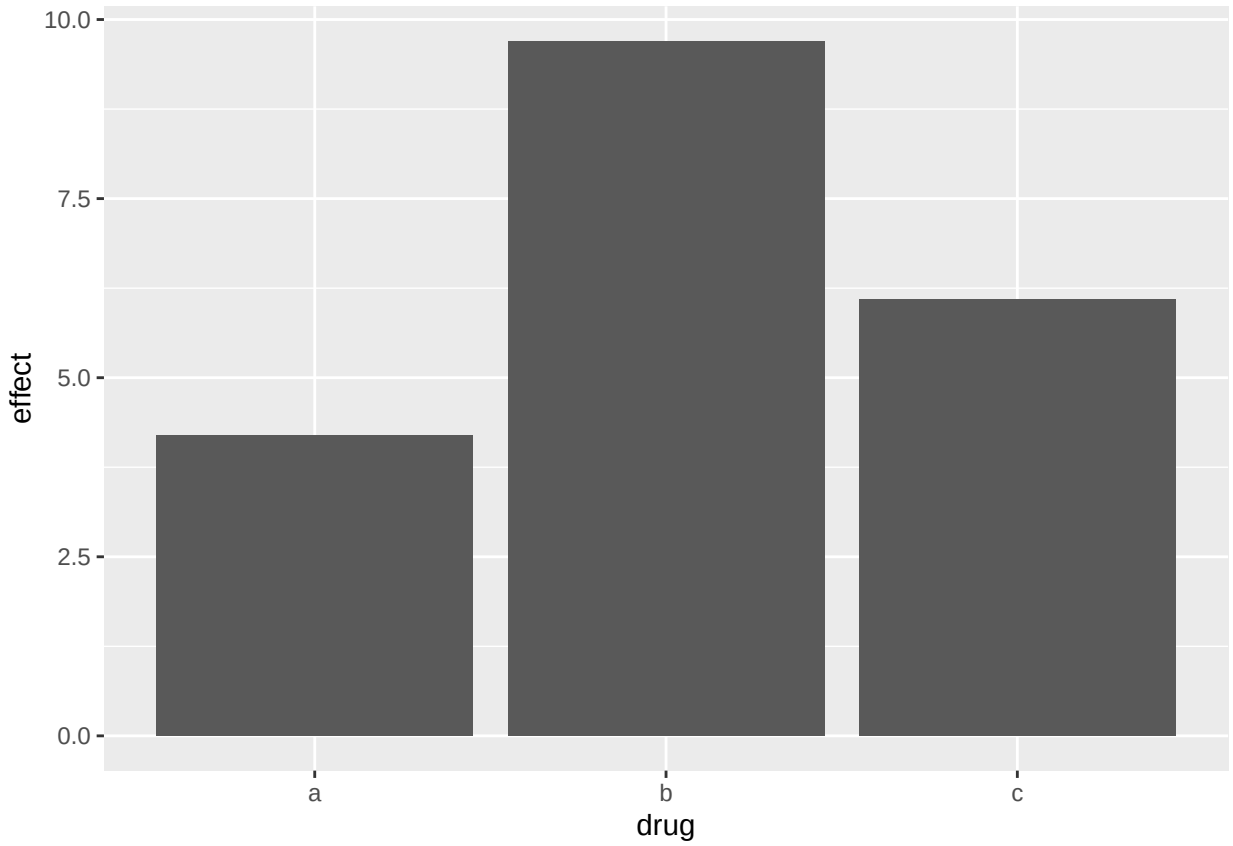
- `stat = "count"` is the default value,
- `stat = "identity"` will skip the aggregation and you'll need to provide the y values.

In `geom_col()` the data is not aggregated by default, it uses `stat_identity()` and leaves the data as is. It expects you to already have the y values calculated and to use them directly.

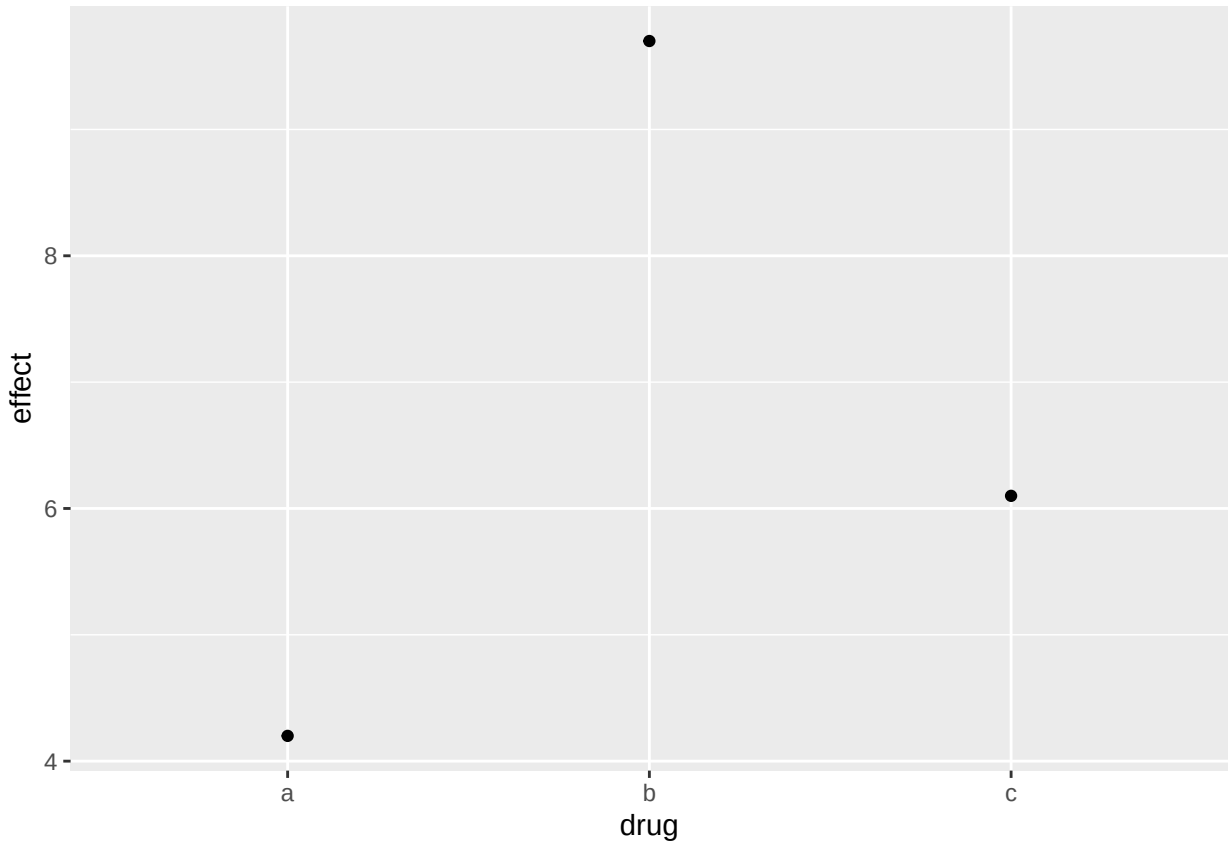
There are two different uses for bar charts:
- For un-summarized data, where each observation contributes one unit to the height of each bar, - For pre-summarized data.

T&T: for pre-summarized data do not run the default stat which bins and counts the data. It is better to use `geom_point()` because points take up less space than bars, and don't require that the y axis includes 0.

```
# Create pre-summarized data
drugs <- data.frame(
  drug = c("a", "b", "c"),
  effect = c(4.2, 9.7, 6.1)
)

# Create a Bar Plot and Point Plot
ggplot(drugs, aes(drug, effect)) + geom_bar(stat = "identity")
```




```
ggplot(drugs, aes(drug, effect)) + geom_point()
```



Scatterplot Scatterplots display the **relationship between two continuous variables**. To prevent overplotting with large datasets, **summarize** the data **before displaying** it.

`shape`: this aesthetic let us modify the shape that will be plotted.
- to set a shape we can assign a number (from 1 to 25) to `shape`
- Shapes from 1-14: only have an **outline**,
- Shapes from 15-20: have a **solid fill**,
- Shapes from 21-25: have an **outline and fill** that can be controlled separately with `color` and `fill`.

`size`: controls the size of the shapes.

```
# Load required packages
library(gcookbook)
library(dplyr)
```

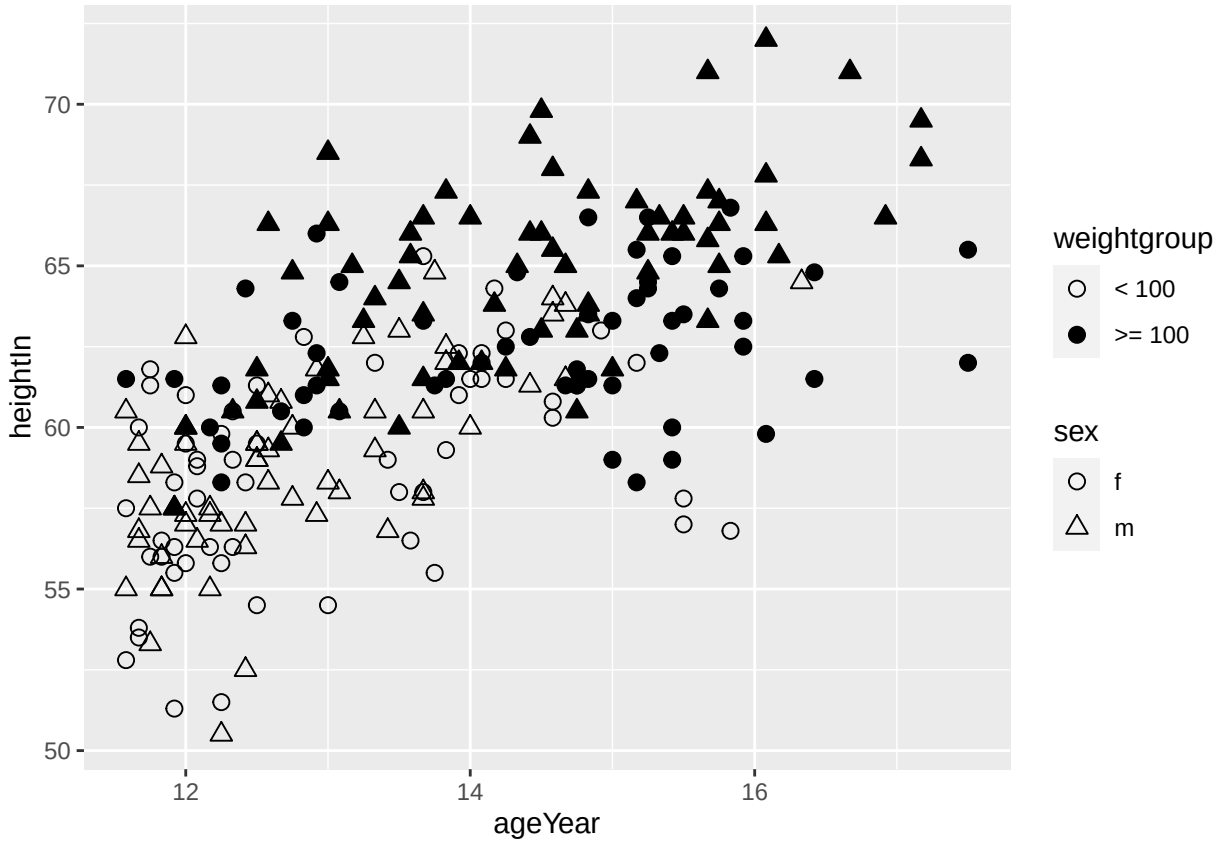
```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# Add a new column into the original dataset by using a condition to set the values
## Save the changes as a new dataset
hw <- heightweight %>%
  mutate(weightgroup = ifelse(weightLb < 100, "< 100", ">= 100"))
```

```
# Set the data and the values for each axis
ggplot(hw, aes(x = ageYear,
               y = heightIn,

               # Set the values of shape and fill with a variable
               shape = sex,
               fill = weightgroup)) +

  # Set the size of the points at 2.5
  geom_point(size = 2.5) +

  # Set the type of the shapes
  scale_shape_manual(values = c(21, 24)) +

  # Choose a fill palette that includes NA (hollow shape) and another color
  scale_fill_manual(
    values = c(NA, "black"),

    # Add a legend guide for each scale and specify aesthetic parameters of legend key
    # with a list
    guide = guide_legend(override.aes = list(shape = 21))
  )
```



Smoother

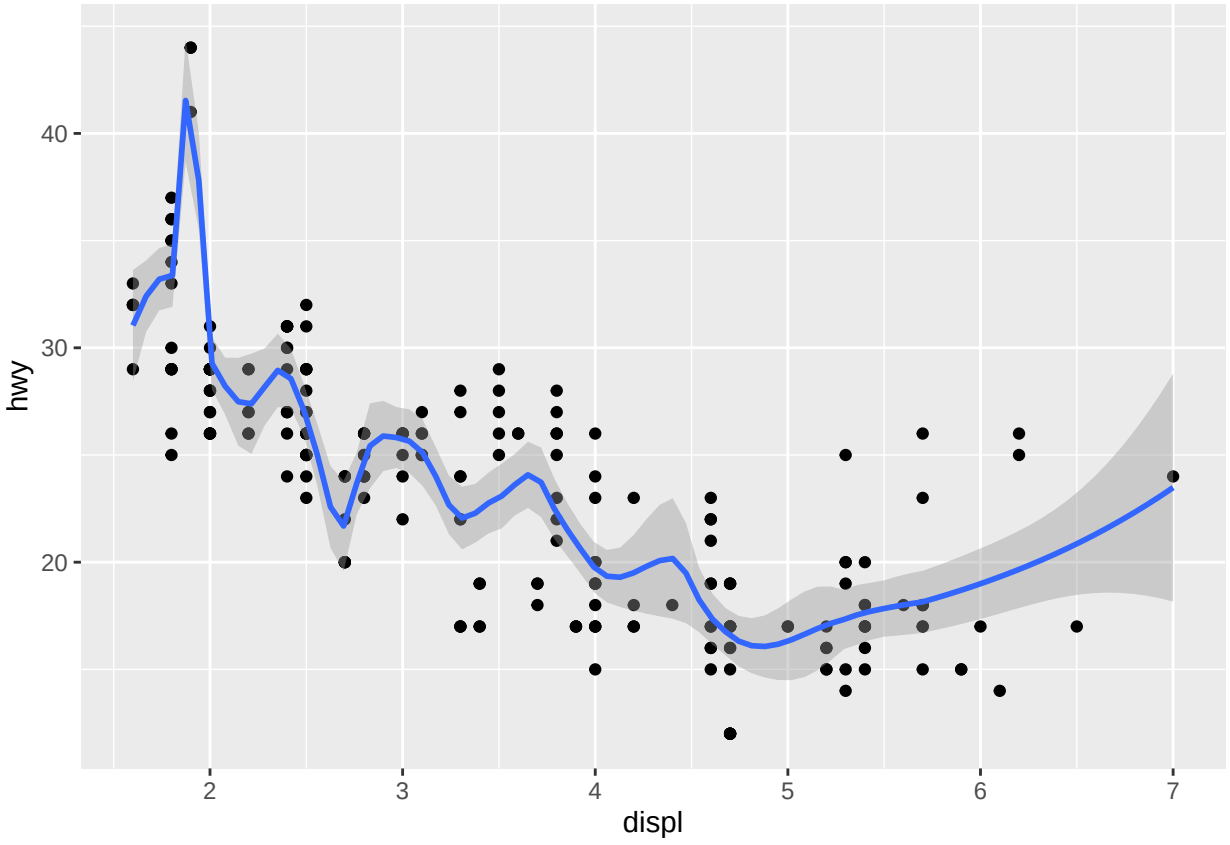
`geom_smooth()`: adds a smoother that shows the dominant pattern on noisy data.

Smoother parameters

- `geom_smooth(se = TRUE)`: displays the point-wise confidence interval around smooth.
- `geom_smooth(method =)` helps us choose which type of model is used to fit the smooth curve.
  - `method = "loess"`: is used as default for datasets fewer than 1,000 observations.
  - `method = "gam"`: fits a generalized additive model provided by the `mgcv` package. It is used when datasets are larger than 1,000 observations.
  - `method = "lm"`: fits a linear model, giving the line of best fit.
  - `method = "rlm"`: uses a robust fitting algorithm than `lm` so that outliers don't affect the fit as much. It's part of the MASS package, so remember to load that first.

```
# "loess" method with span = 0.2
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth(span = 0.2)
```

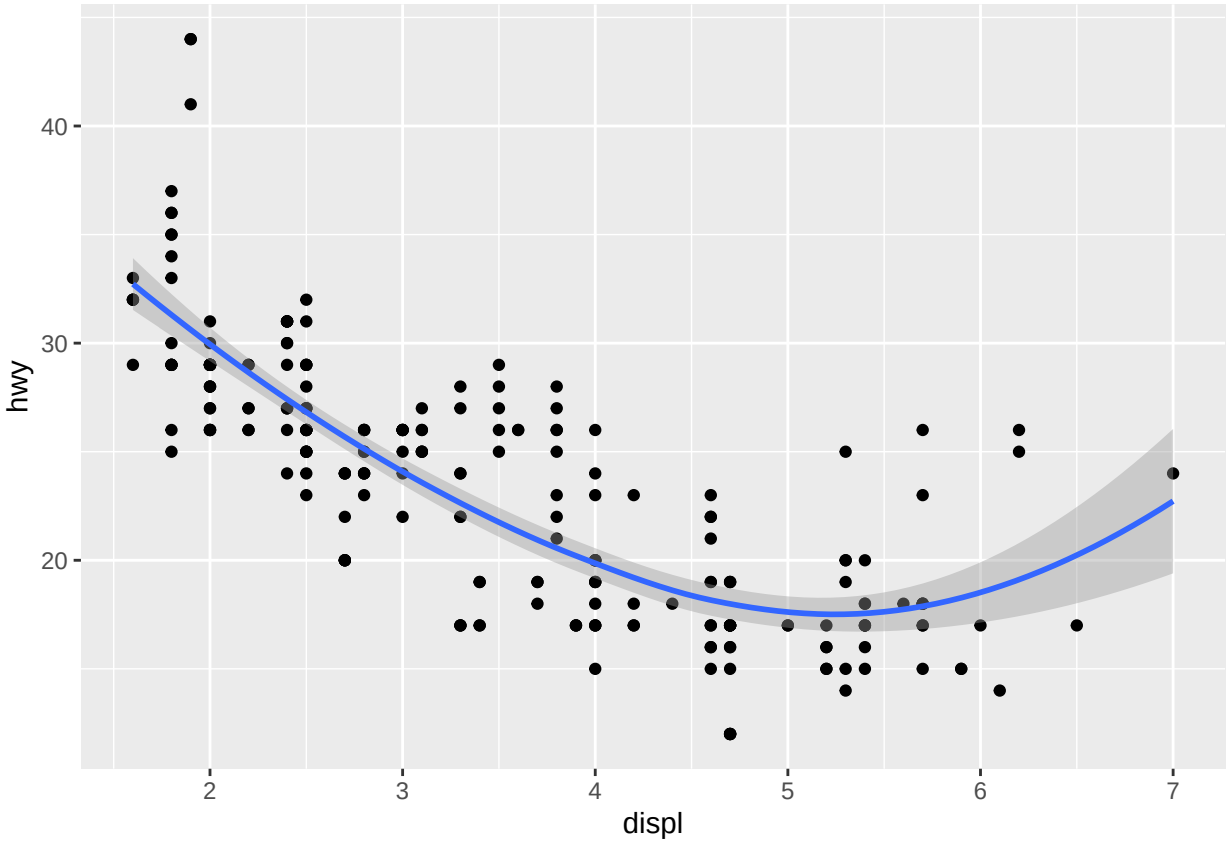
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
# "loess" method with span = 1
ggplot(mpg, aes(displ, hwy)) +
```

```
  geom_point() +
  geom_smooth(span = 1)
```

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




```
# Smoothing with "gam"
library(mgcv)
```

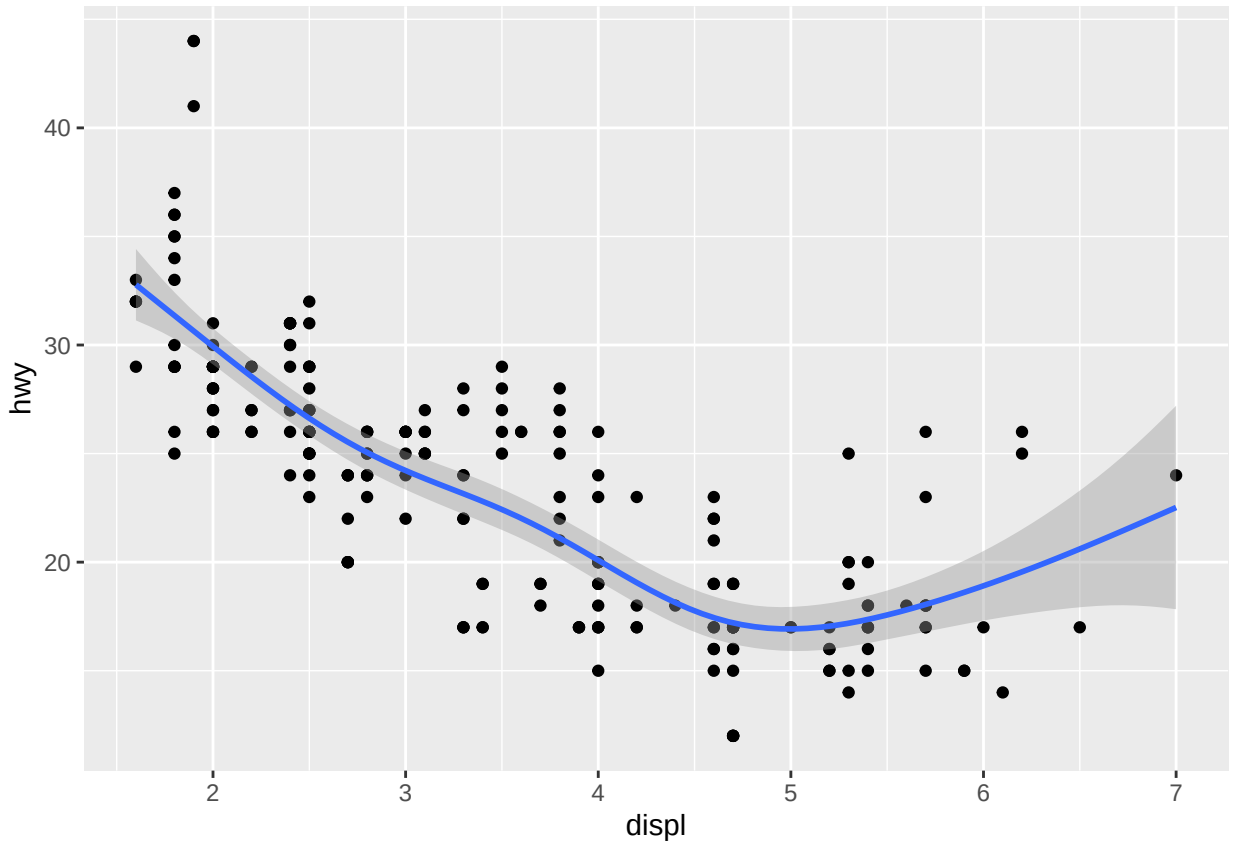
```
## Loading required package: nlme
```

```
##
## Attaching package: 'nlme'
```

```
## The following object is masked from 'package:dplyr':
##
##     collapse
```

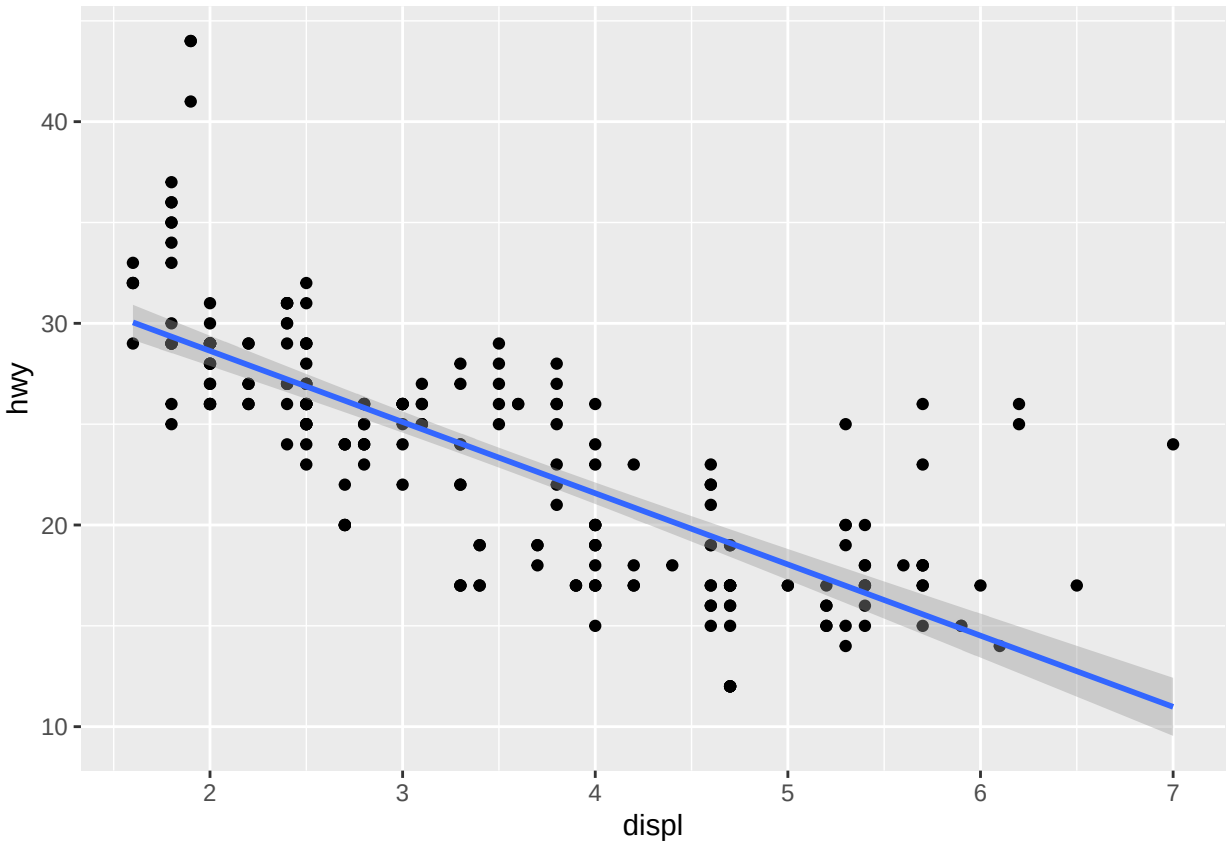
```
## This is mgcv 1.8-33. For overview type 'help("mgcv-package")'.
```

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth(method = "gam", formula = y ~ s(x))
```




```
# Smoothing with "lm"
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth(method = "lm")
```

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



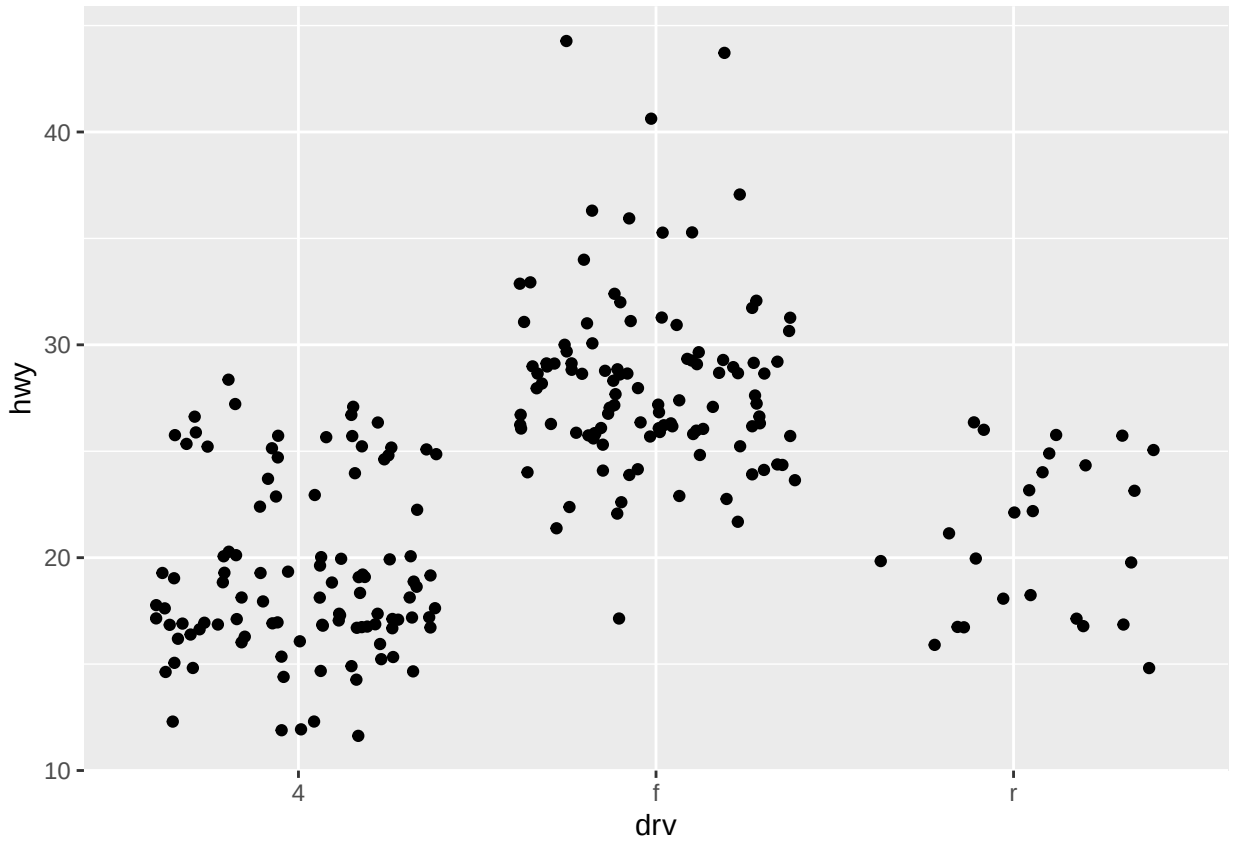
Boxplots and Jittered Points

Scatterplots helps us to visualize how the values of the **continuous variables** vary with the levels of the **categorical variable**.

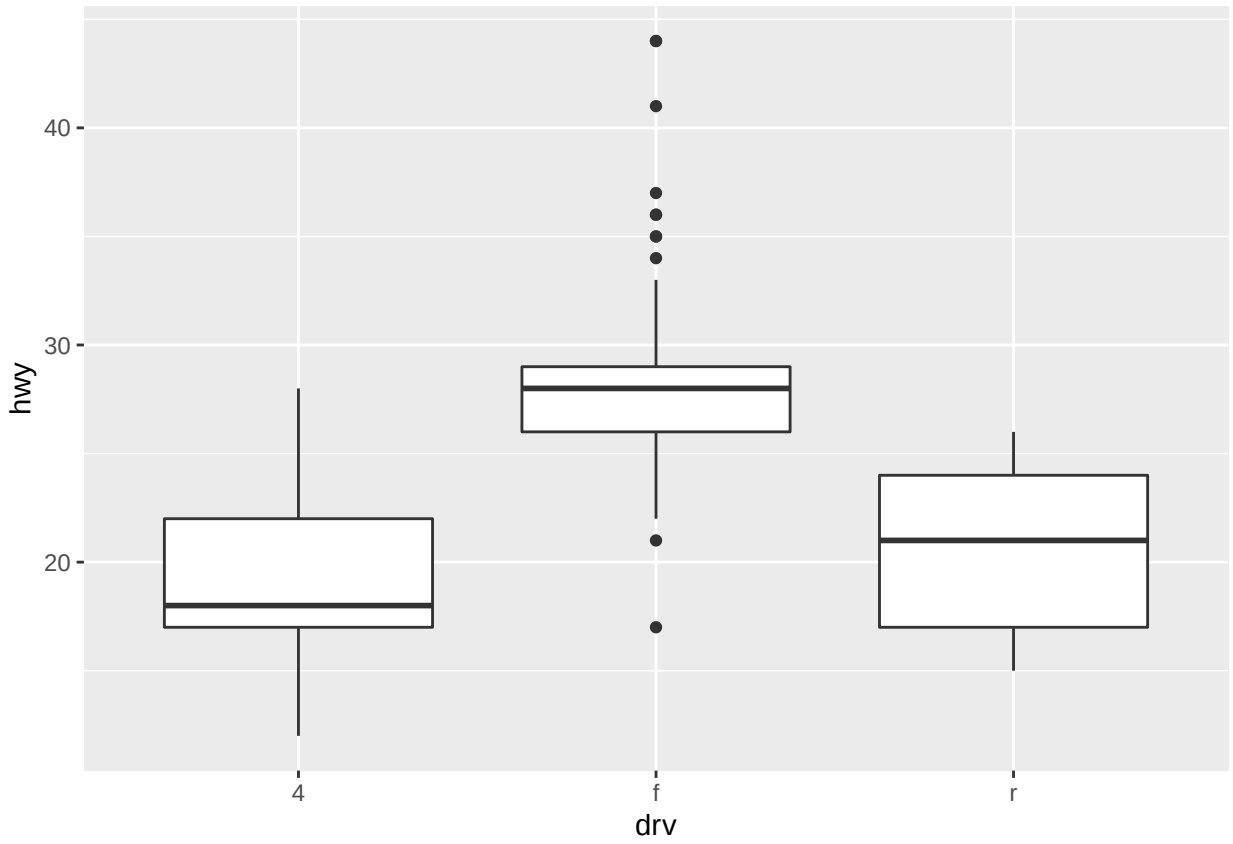
When there are few unique values, many points can be plotted in the same location. This can be solved by using theses techniques:

- `geom_jitter()`: adds a random noise to the data reducing overplotting.
- `geom_boxplot()`: summarize the shape of the distribution with a handful of summary statistics.
- `geom_violin()`: show a compact representation of the "density" of the distribution, highlighting the areas where more points are found.

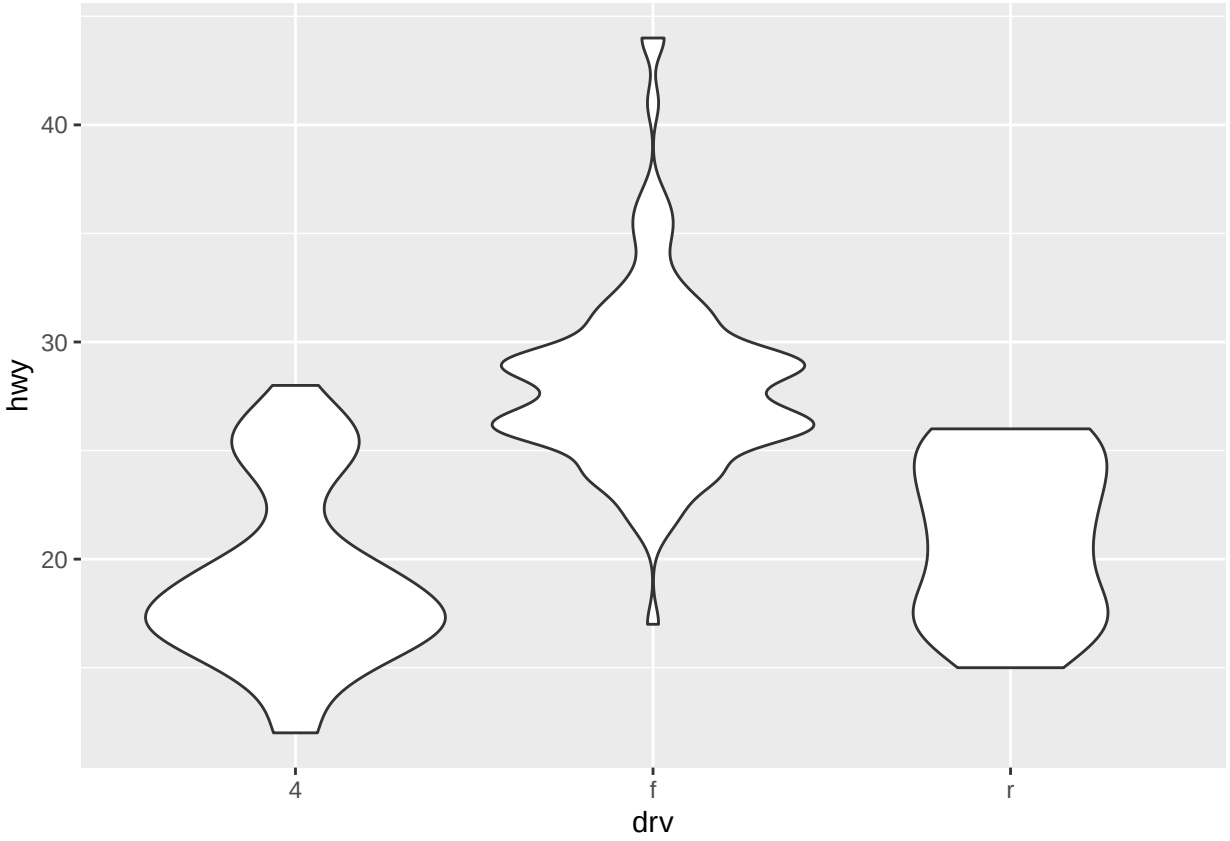
```
ggplot(mpg, aes(drv, hwy)) + geom_jitter()
```




```
ggplot(mpg, aes(drv, hwy)) + geom_boxplot()
```




```
ggplot(mpg, aes(drv, hwy)) + geom_violin()
```



Strenghts and Weaknesses

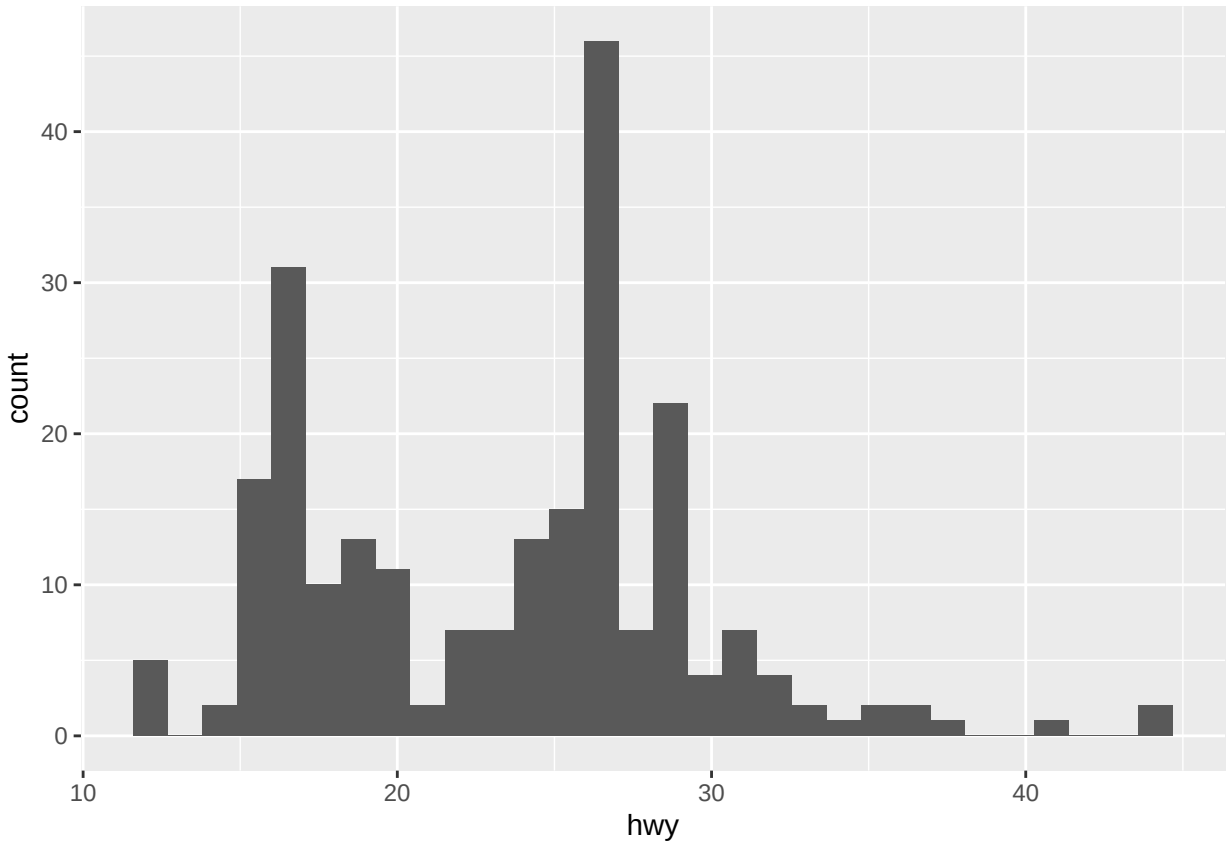
- **Boxplots** summarize the bulk of the distribution with only five numbers,
- **Jittered plots** show every point but only work with relatively small datasets,
- **Violin plots** give the richest display, but rely on the calculation of a density estimate, which can be hard to interpret.

Histograms and Frequency Polygons

They are useful for showing the distribution of a **single numeric variable**. They provide more information about the distribution of a single group than boxplots.

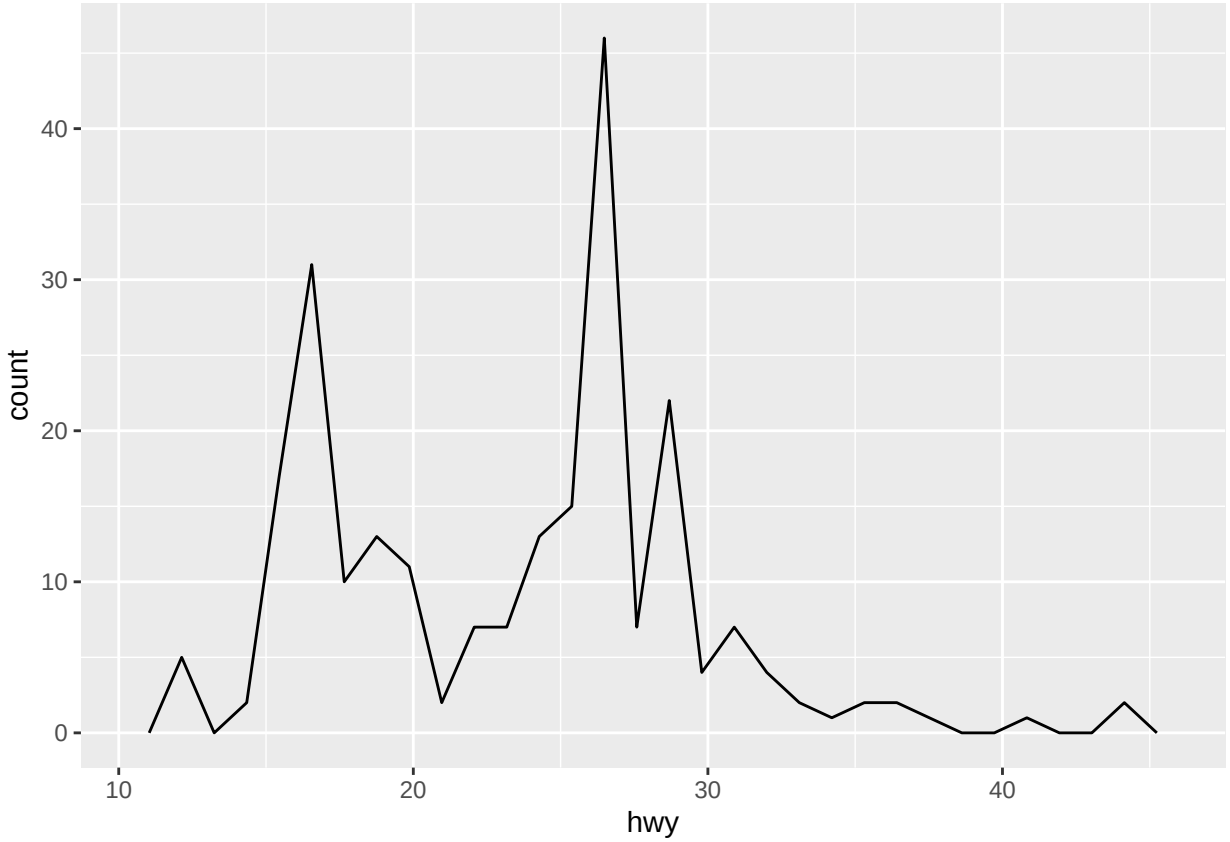
```
# Histogram
ggplot(mpg, aes(hwy)) + geom_histogram()
```

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




```
# Frequency Polygon
ggplot(mpg, aes(hwy)) + geom_freqpoly()
```

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



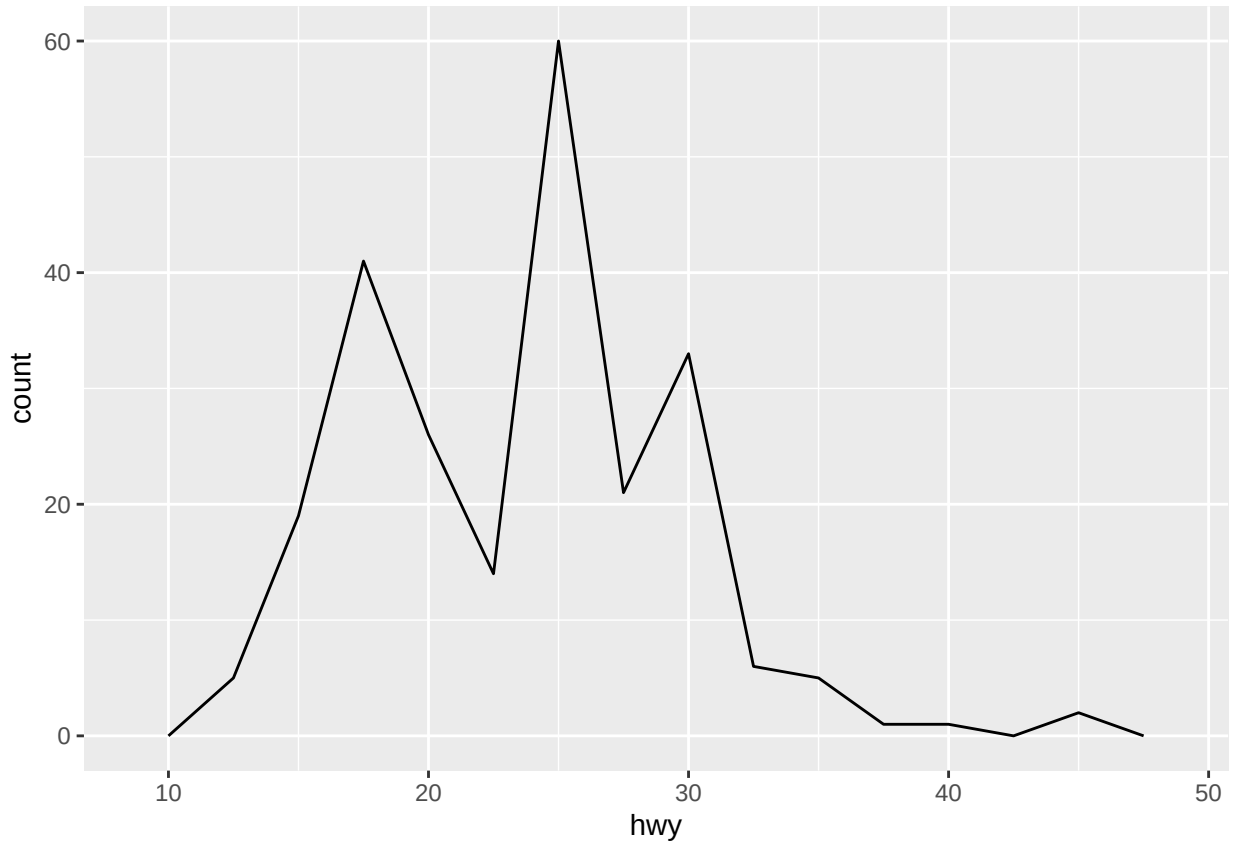
Both plots group individual data values into one instance of a graphic element (bin), then count the number of observations in each bin.

By default

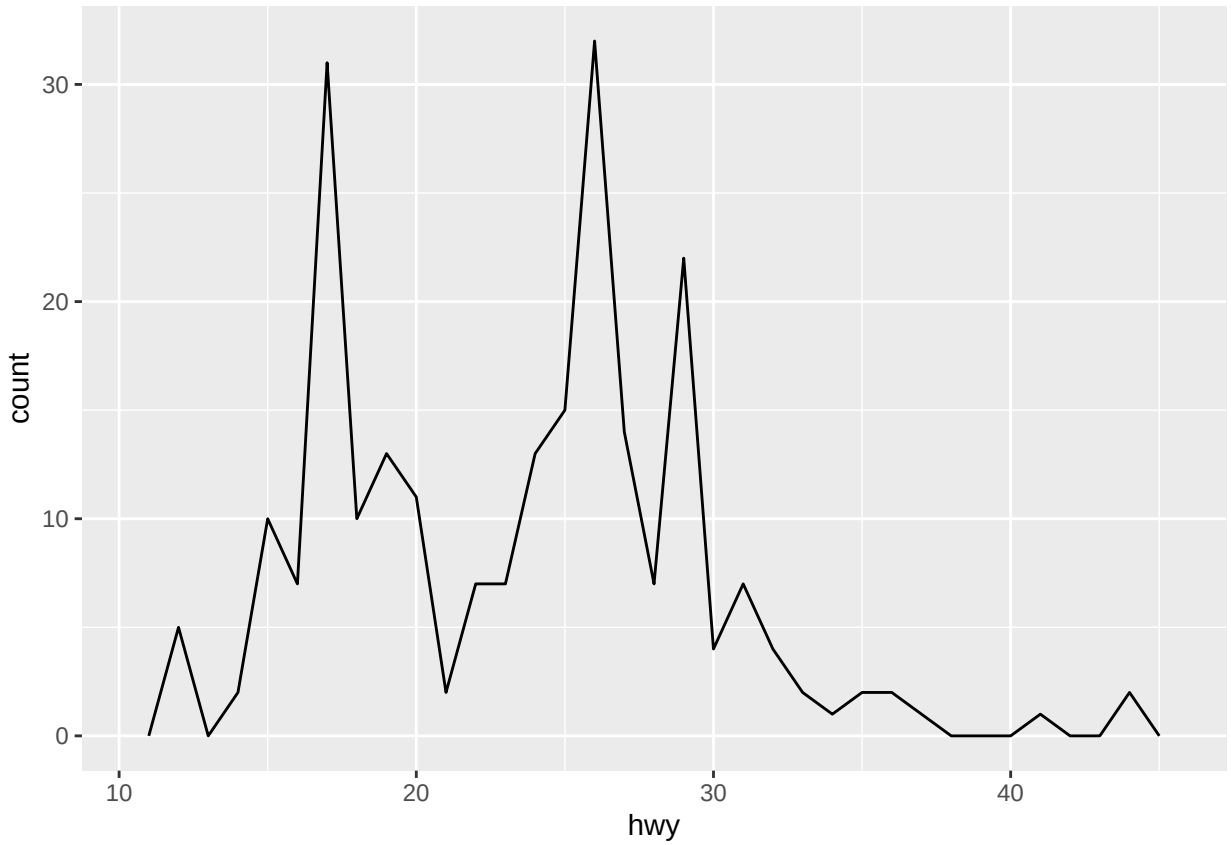
The only difference on both plots is the display:
- **histograms use bars**,
- **frequency polygons use lines**.

By default data is split into 30 bins. This can be changed by using `binwidth` argument.

```
ggplot(mpg, aes(hwy)) +
  geom_freqpoly(binwidth = 2.5)
```



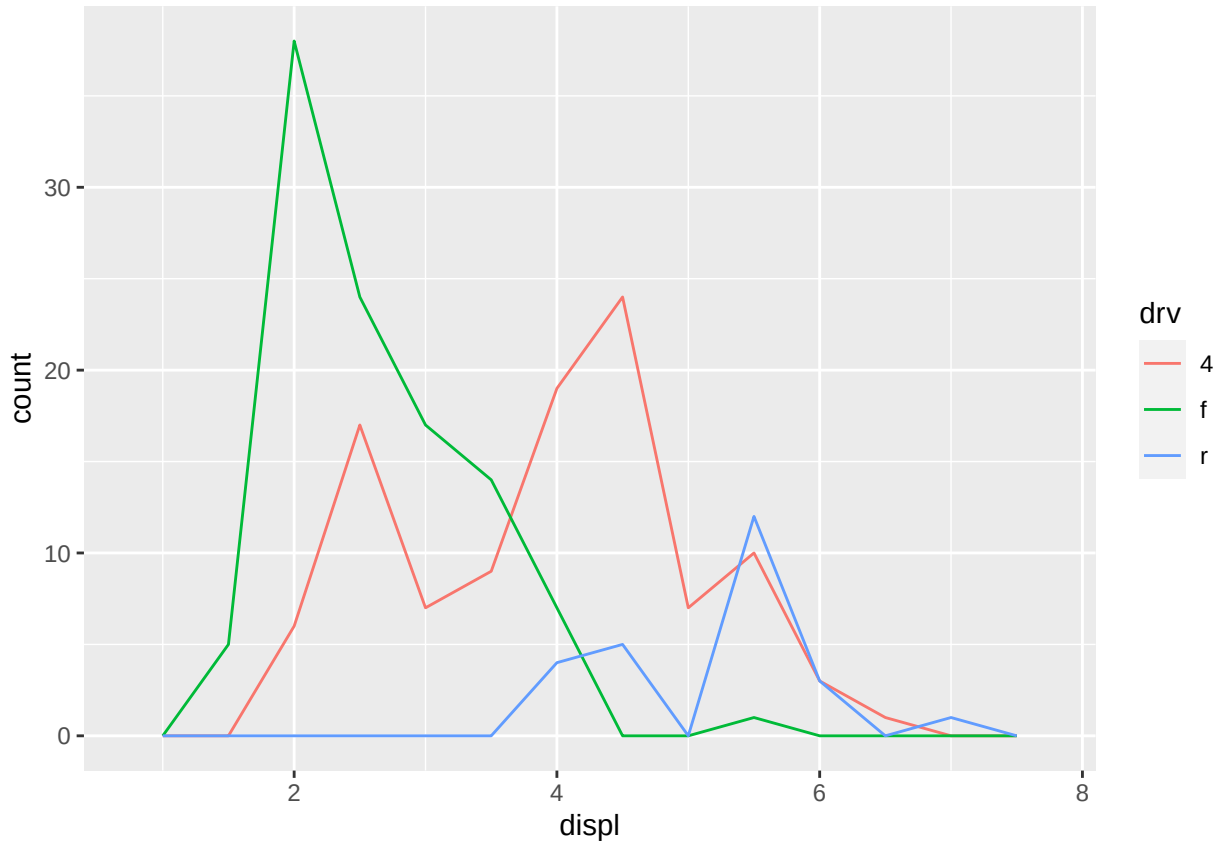

```
ggplot(mpg, aes(hwy)) +
  geom_freqpoly(binwidth = 1)
```



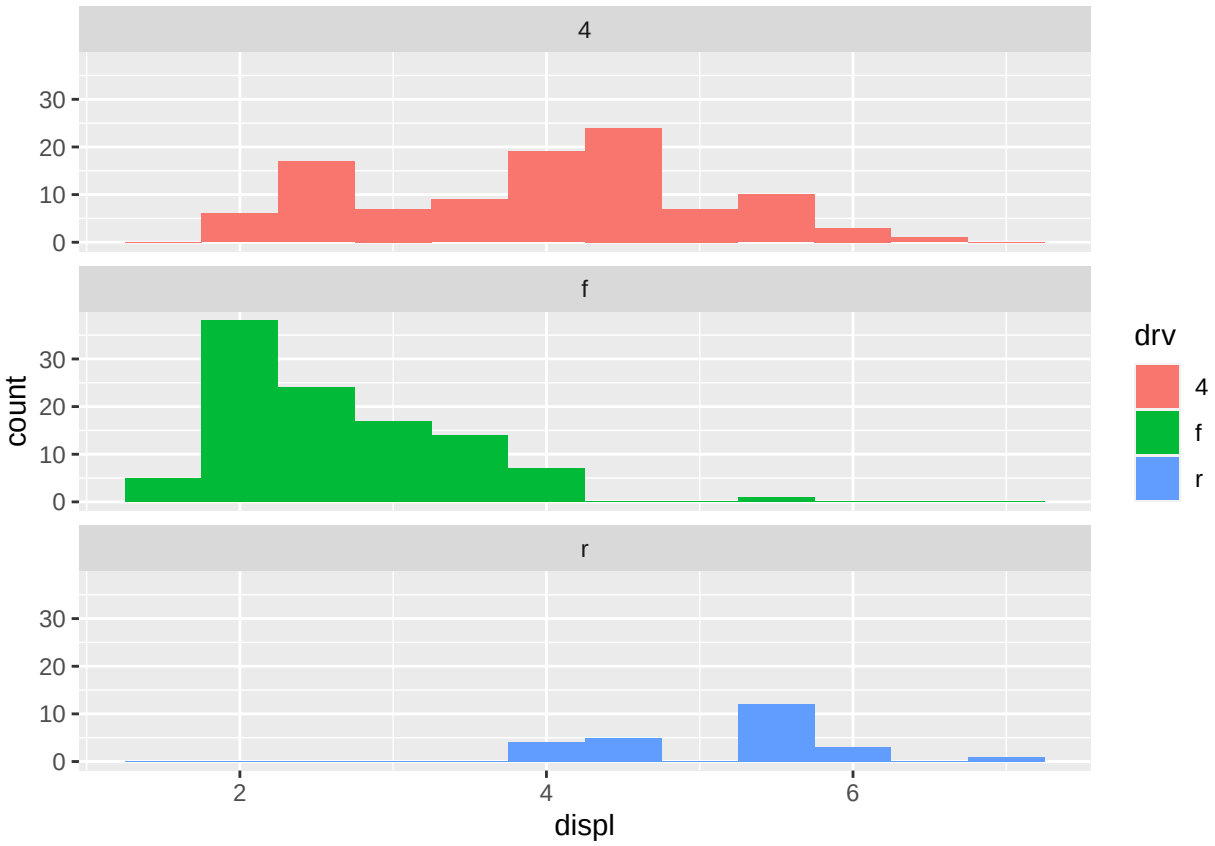
`geom_density()` is an alternative to the frequency polygons, but hey are harder to interpret and they assume that the underlying distribution is continuous, unbounded, and smooth.

It's easier to compare distributions using the frequency polygon, but faceting histograms helps us to see the distribution of each group.

```
# Frequency Polygon
ggplot(mpg, aes(displ, colour = drv)) +
  geom_freqpoly(binwidth = 0.5)
```




```
# Box Plot faceting
ggplot(mpg, aes(displ, fill = drv)) +
  geom_histogram(binwidth = 0.5) +
  facet_wrap(~drv, ncol = 1)
```

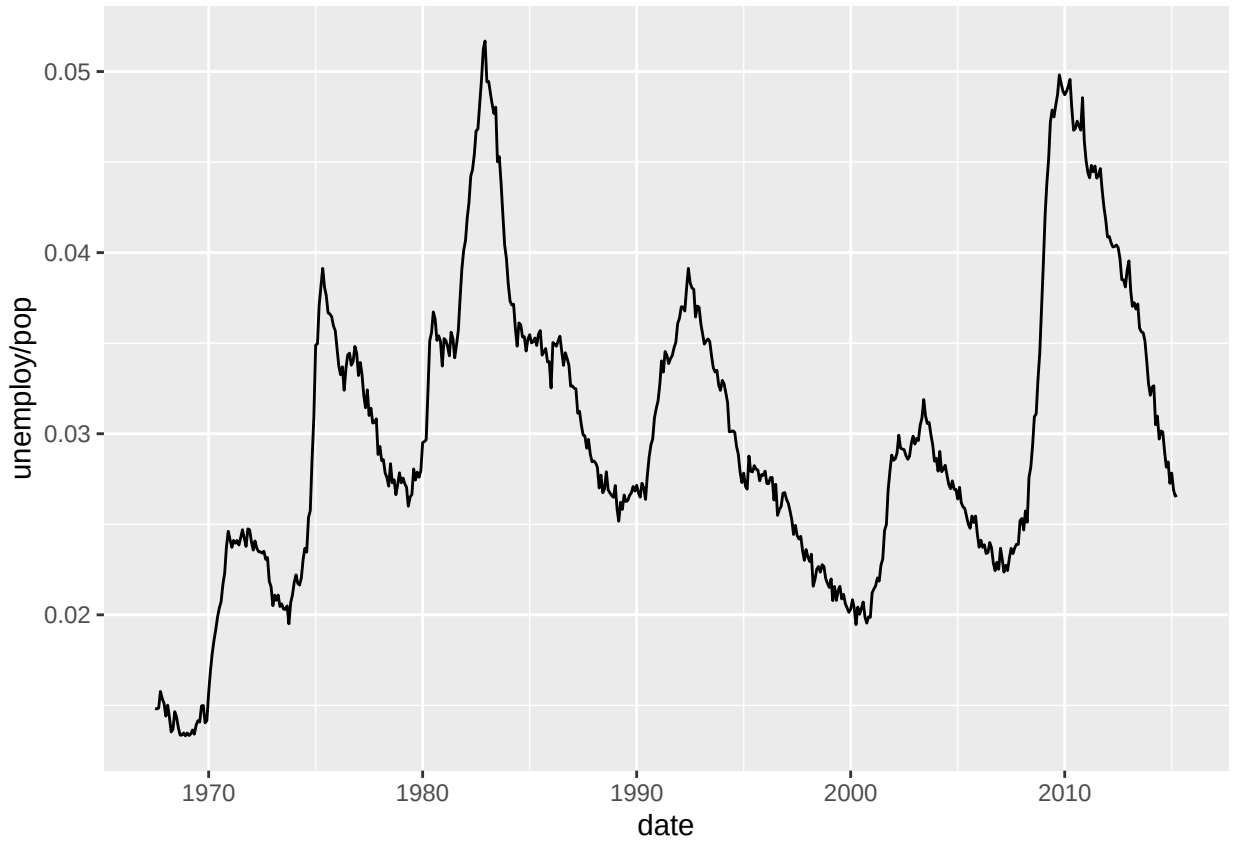


Time Series with Line and Path Plots

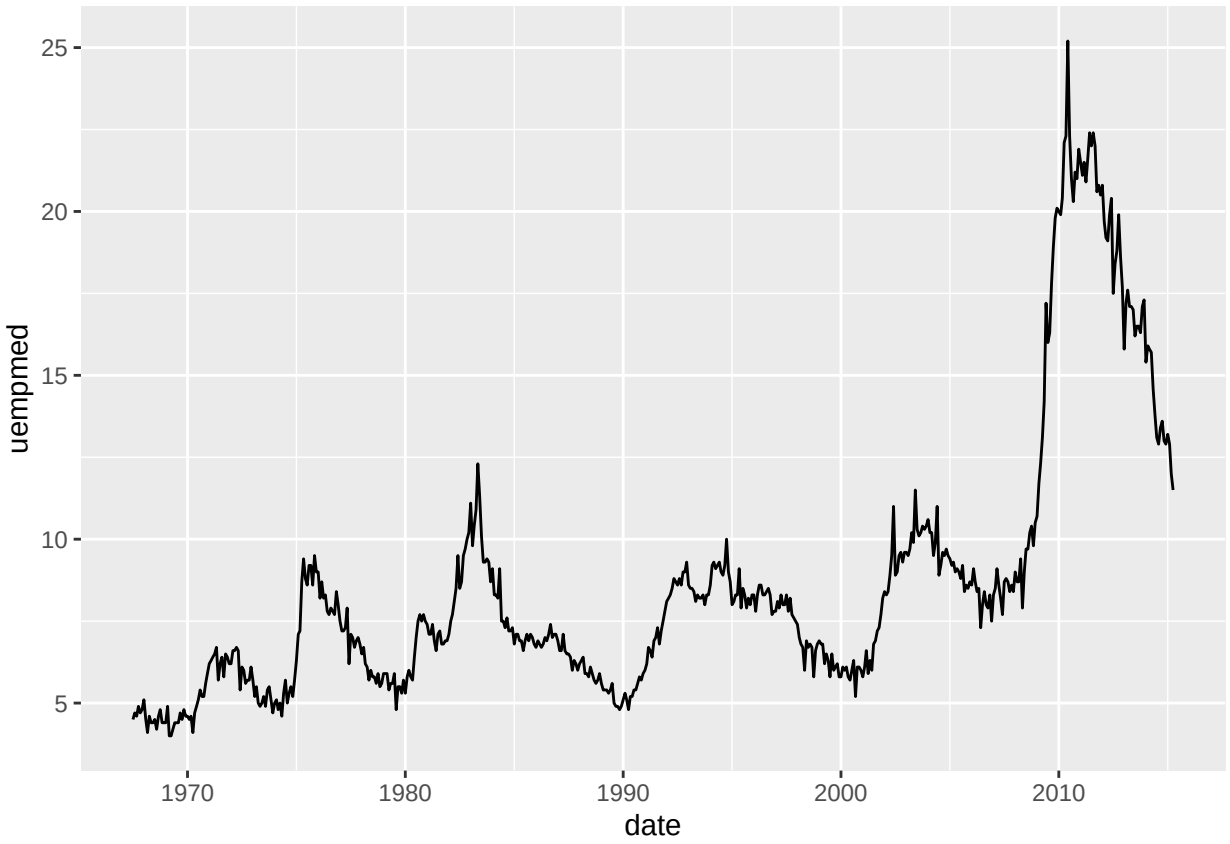
Line and path plots are typically used for time series data.

- **Line plots**: join the points from left to right,
  - They usually have time on the x-axis, showing how a single variable has changed over time.
- **Path plots**: join the points in the order that they appear in the dataset.
  - They show how two variables have simultaneously changed over time, with time encoded in the way that observations are connected.

```
# Unemployment rate over time
ggplot(economics, aes(date, unemploy / pop)) +
  geom_line()
```



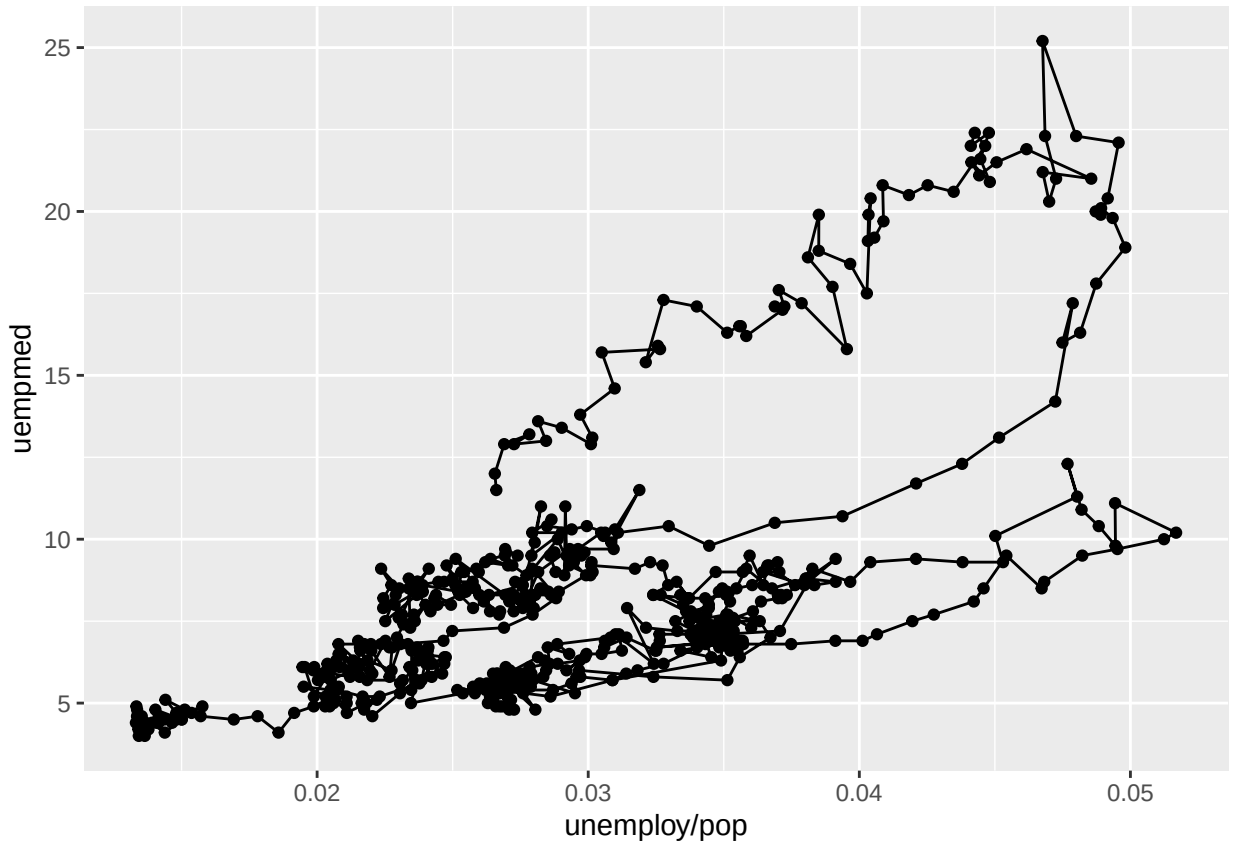

```
# Median number of weeks unemployed
ggplot(economics, aes(date, uempmed)) +
  geom_line()
```



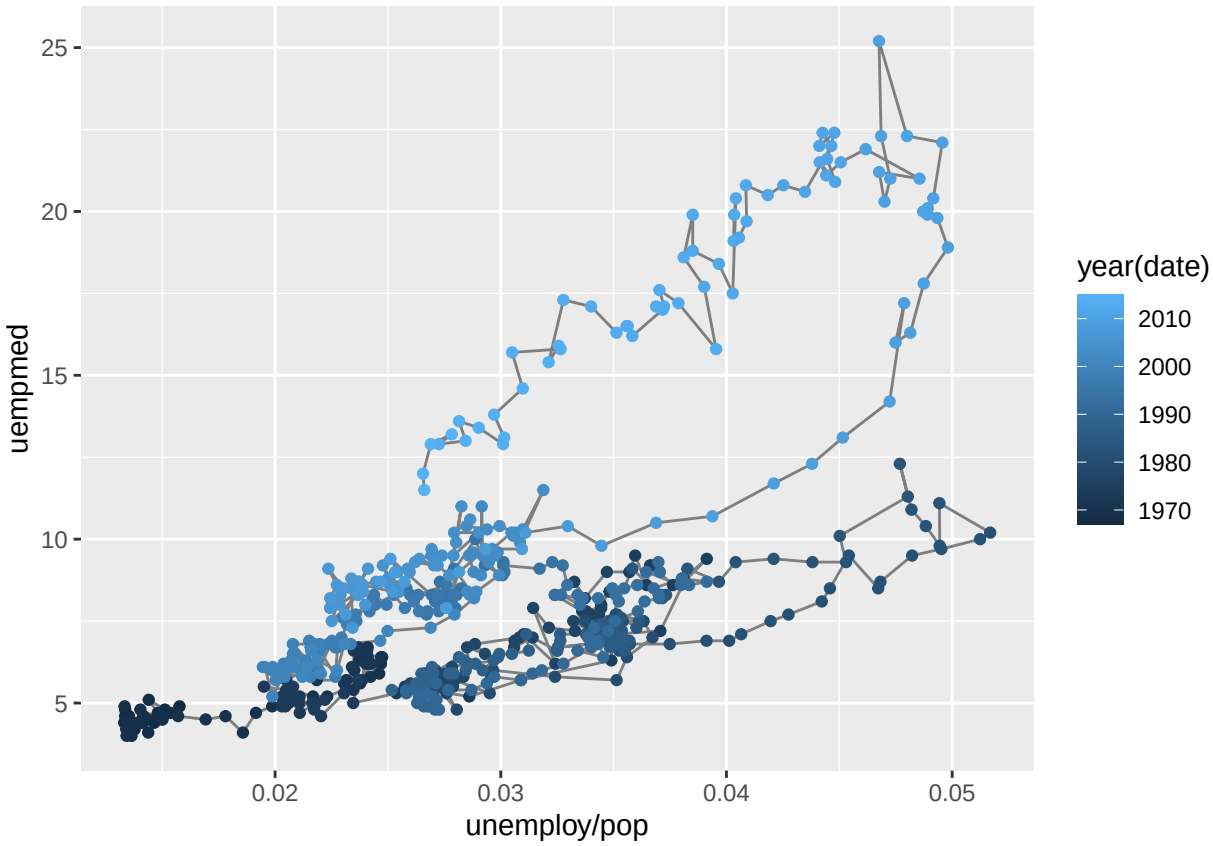
To examine relationships between two line plots while still be able to see the evolution over time we can join points adjacent in time with line segments, forming a path plot.

If our plots have many line crossings, we can color the points to track better the direction over time.

```
# Visualize the relationship between two continuous variables with a line path
ggplot(economics, aes(unemploy / pop, uempmed)) +
  geom_path() +
  geom_point()
```




```r
# Create a function to extract the year when passing a date
year <- function(x) as.POSIXlt(x)$year + 1900

# Visualize the relationship between two continuous variables
## Color each point to track better the change over time
ggplot(economics, aes(unemploy / pop, uempmed)) +
  geom_path(color = "grey50") +
  geom_point(aes(color = year(date)))
```



T&T: With longitudinal data, you often want to display multiple time series on each plot, each series representing one individual. To do this you need to map the group aesthetic to a variable encoding the group membership of each observation.

Resources

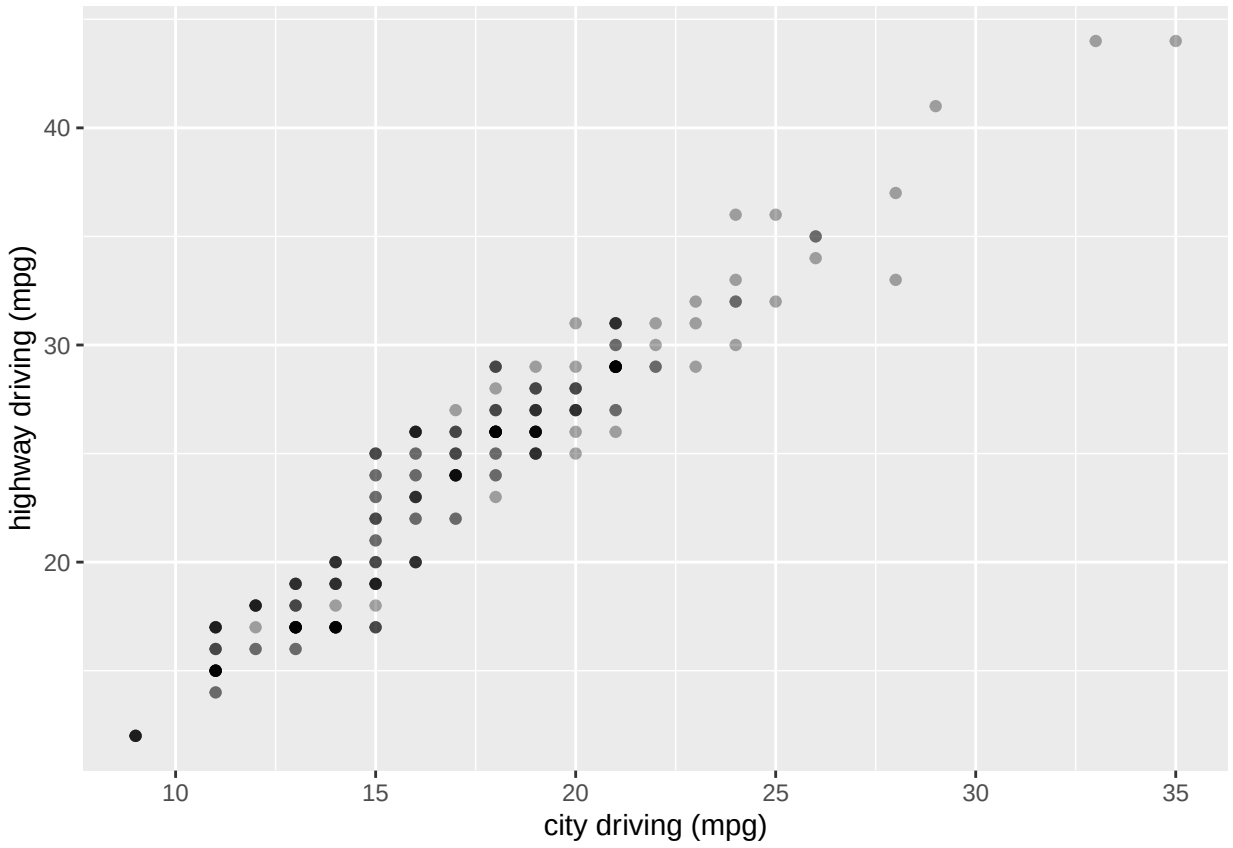
ggplot2 Overview

ggplot2 Reference

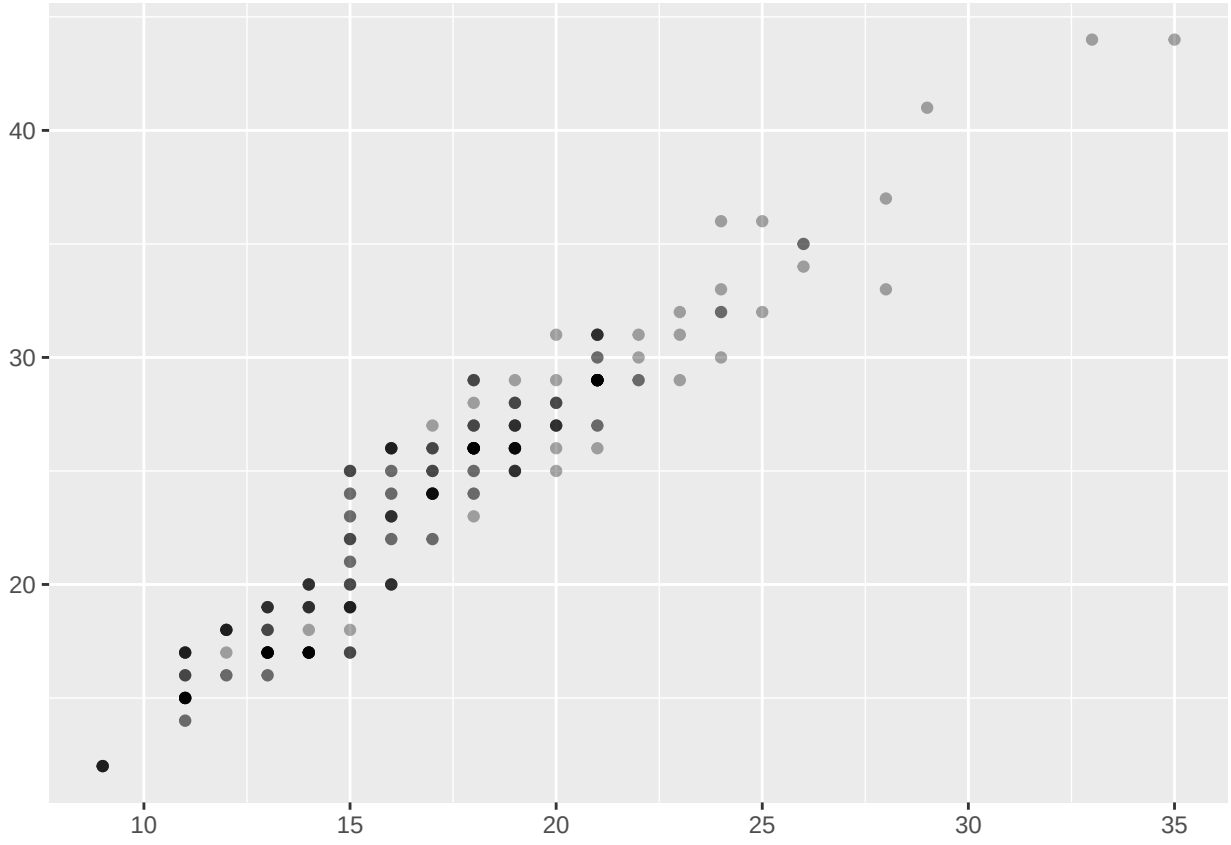
Modifying Axes

- `xlab()` and `ylab()` modify the x- and y-axis labels.
- `xlim()` and `ylim()` modify the limits of axes.

```
ggplot(mpg, aes(cty, hwy)) +
  geom_point(alpha = 1 / 3) +
  xlab("city driving (mpg)") +
  ylab("highway driving (mpg)")
```

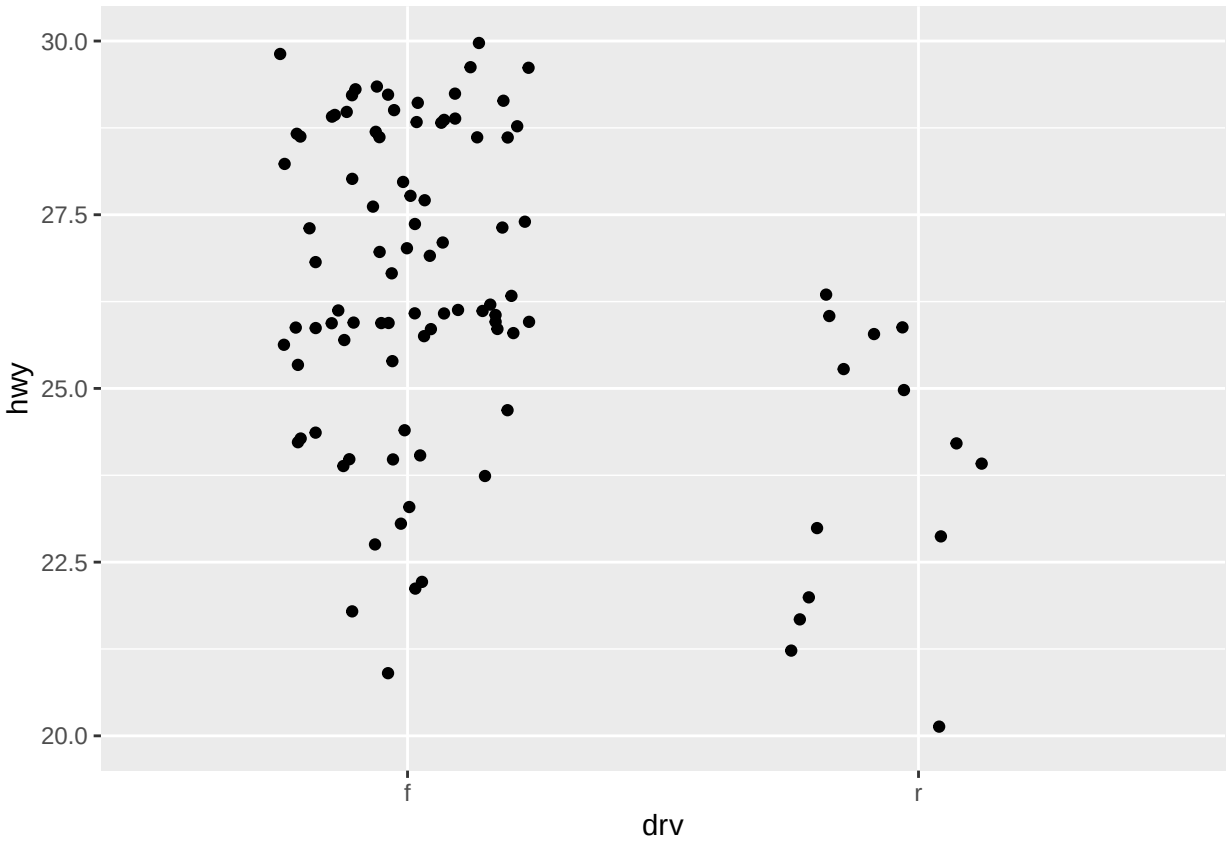



```
# Remove the axis labels with NULL
ggplot(mpg, aes(cty, hwy)) +
  geom_point(alpha = 1 / 3) +
  xlab(NULL) +
  ylab(NULL)
```

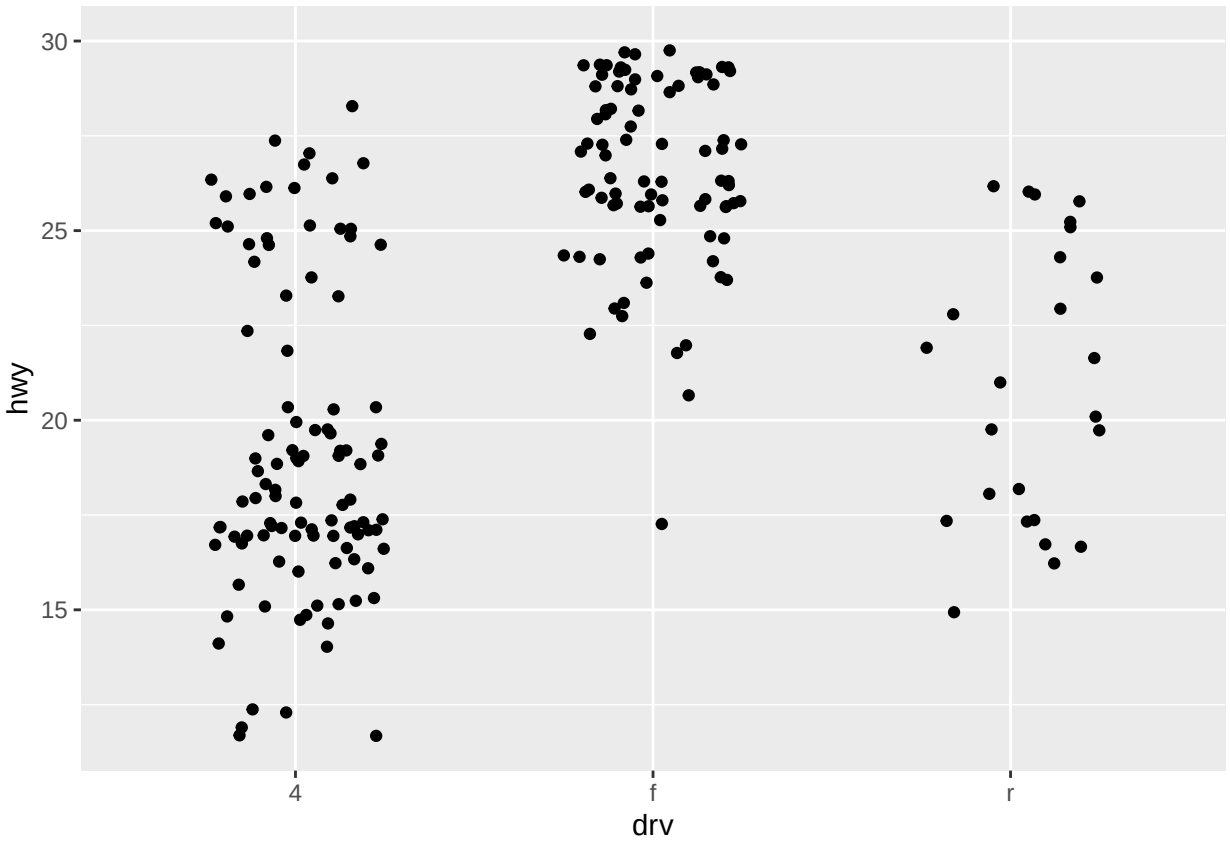



```
ggplot(mpg, aes(drv, hwy)) +
  geom_jitter(width = 0.25) +
  xlim("f", "r") +
  ylim(20, 30)
```

```
## Warning: Removed 137 rows containing missing values (geom_point).
```




```
# For continuous scales, use NA to set only one limit
ggplot(mpg, aes(drv, hwy)) +
  geom_jitter(width = 0.25, na.rm = TRUE) +
  ylim(NA, 30)
```

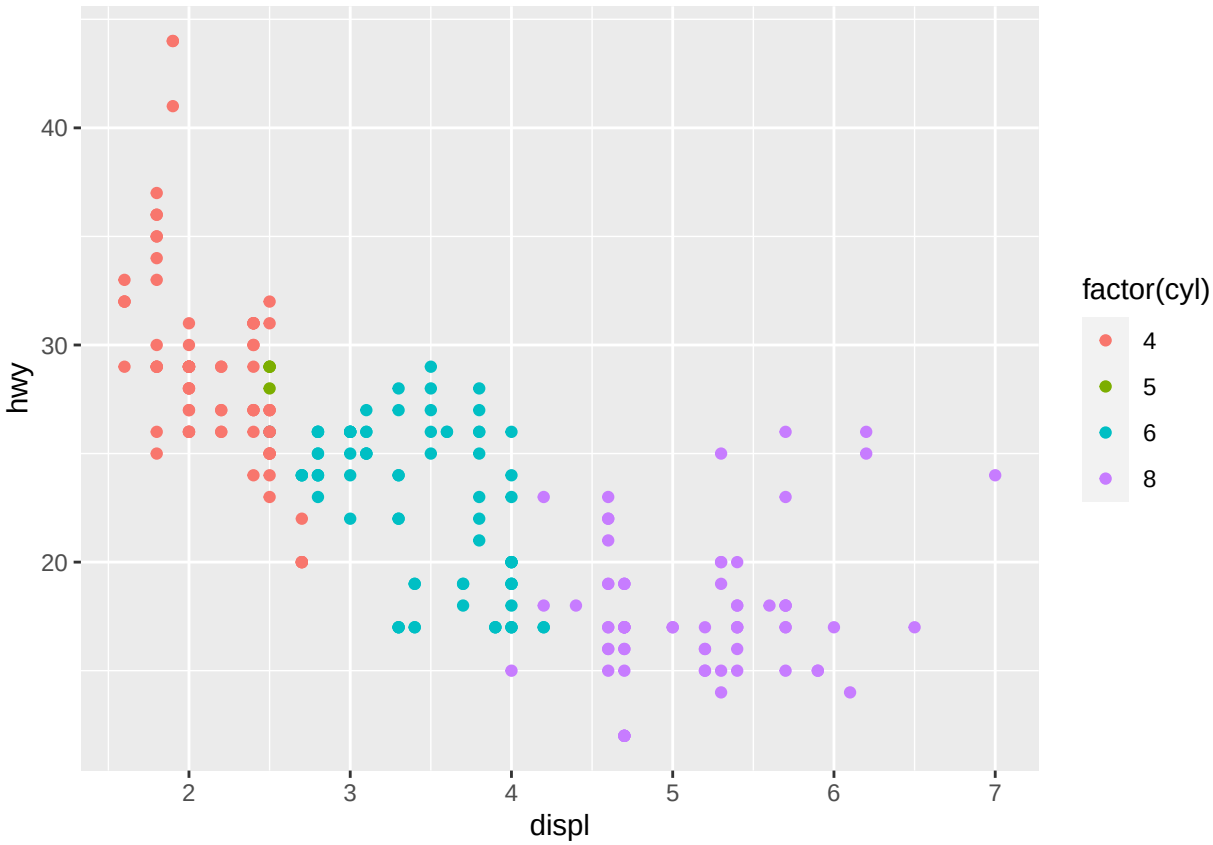


Storing Plots on Variables

A plot can be saved into a variable and manipulate it.
Operations that can be performed with the plot variable are:

- Render it on screen with `print()` when running inside a loop function,
- Save it to disk with `ggsave()`,
- Describe its structure with `summary()`,
- Save a cached copy of it to disk, with `saveRDS()`. This saves a complete copy of the plot object, so you can easily re-create it with `readRDS()`.

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

# Print the plot variable
print(p)
```




```
# Save the plot to disk
ggsave("plot.png", p, width = 5, height = 5)

# Describe the plot structure
summary(p)
```

```
## data: manufacturer, model, displ, year, cyl, trans, drv, cty, hwy, fl,
##   class [234x11]
## mapping:  x = ~displ, y = ~hwy, colour = ~factor(cyl)
## faceting: <ggproto object: Class FacetNull, Facet, gg>
##     compute_layout: function
##     draw_back: function
##     draw_front: function
##     draw_labels: function
##     draw_panels: function
##     finish_data: function
##     init_scales: function
##     map_data: function
##     params: list
##     setup_data: function
##     setup_params: function
##     shrink: TRUE
##     train_scales: function
##     vars: function
##     super:  <ggproto object: Class FacetNull, Facet, gg>
## -----------------------------------
## geom_point: na.rm = FALSE
```

```
## stat_identity: na.rm = FALSE
## position_identity
```

```
# Saved a cached copy of the plot
saveRDS(p, "plot.rds")
q <- readRDS("plot.rds")
```

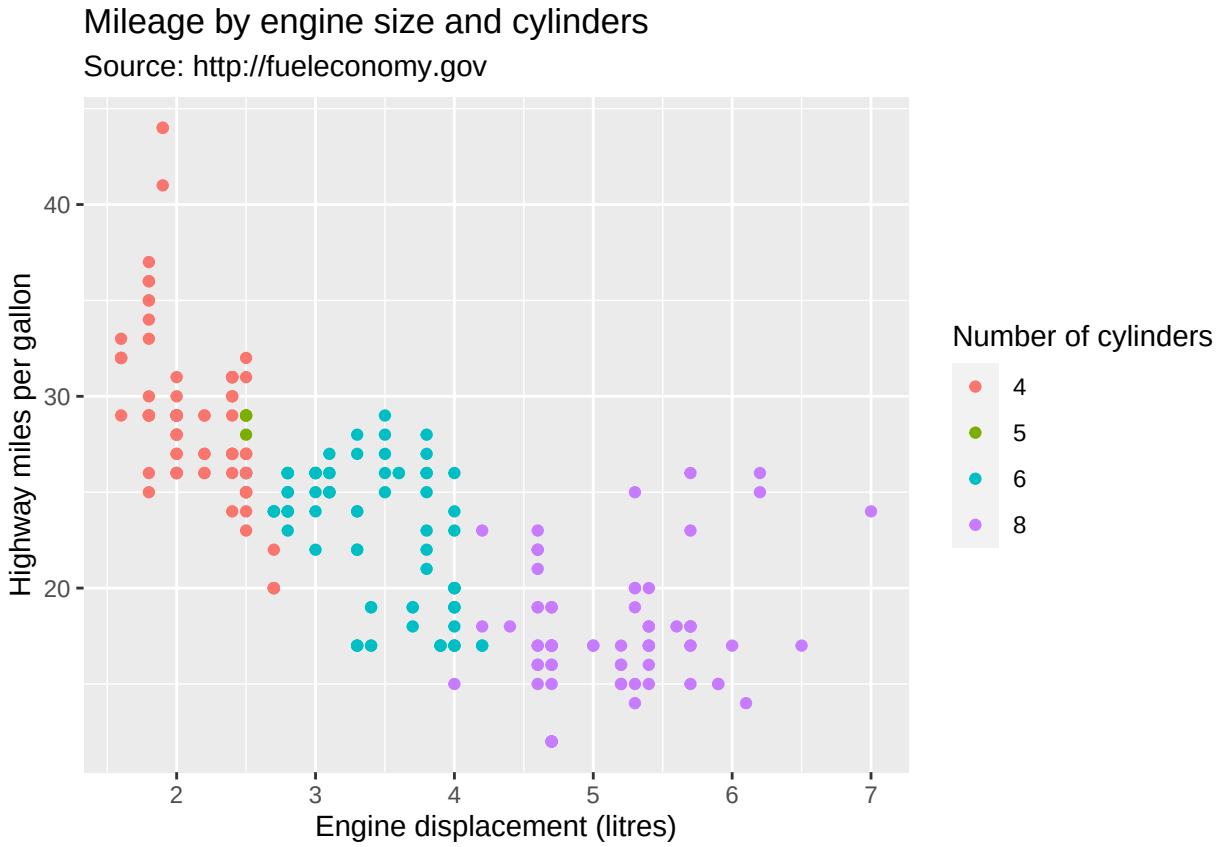
Annotations

An annotation supplies **metadata**, which provides additional information about the data being displayed.

Plot and Axis Titles

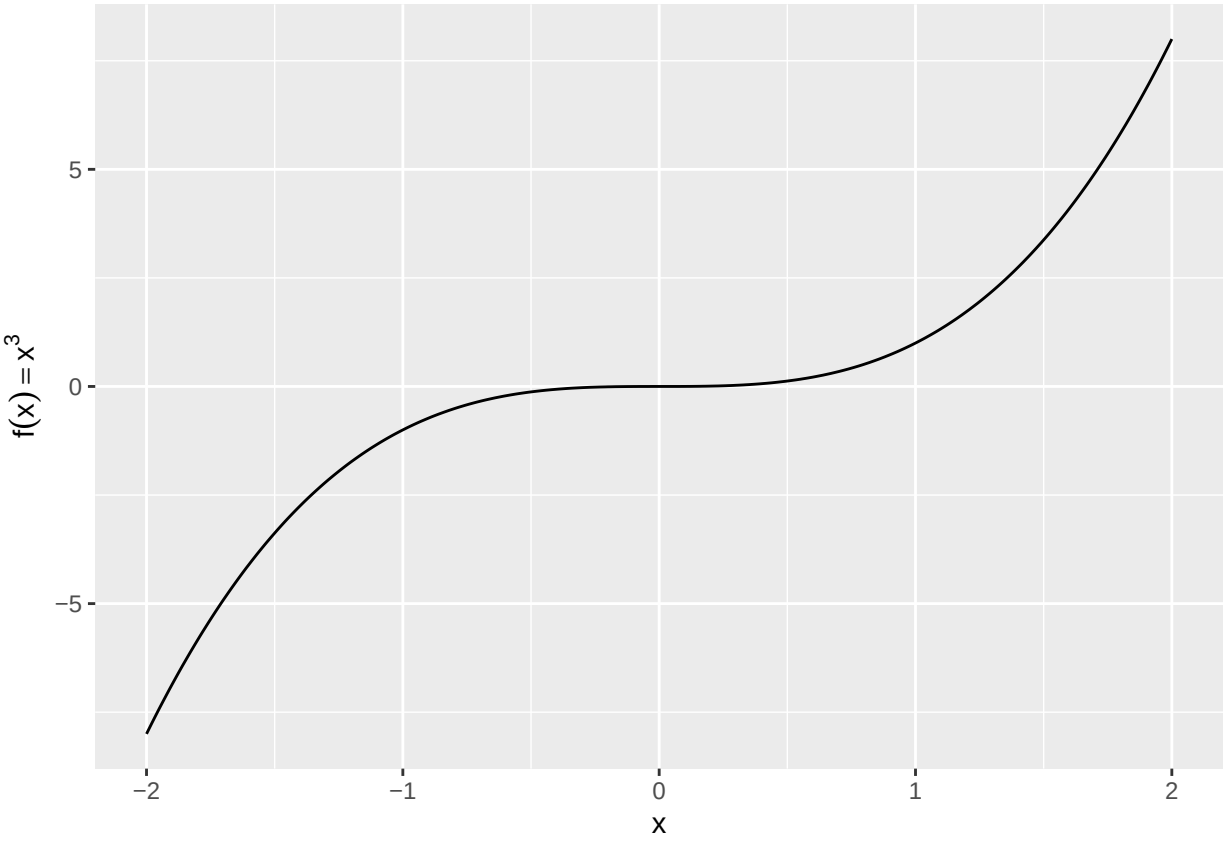
ggplot2 provides `labs()` to help us modify the titles, axes, and legends associated with the plot.

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = factor(cyl))) +
  labs(
    x = "Engine displacement (litres)",
    y = "Highway miles per gallon",
    colour = "Number of cylinders",
    title = "Mileage by engine size and cylinders",
    subtitle = "Source: http://fueleconomy.gov"
  )
```



The values supplied to labs() are typically **text strings**, with `\n` used to specify **line breaks**, but you can also supply **mathematical expressions** wrapped in `quote()`.

```r
values <- seq(from = -2, to = 2, by = .01)
df <- data.frame(x = values, y = values ^ 3)
ggplot(df, aes(x, y)) +
  geom_path() +
  labs(y = quote(f(x) == x^3))
```



Themes

ggthemes: Themes don't change the perceptual properties of the plot, but they do help you **make the plot aesthetically pleasing** or match an existing style guide. Themes give you control over things like fonts, ticks, panel strips, and backgrounds.

RColorBrewer: Creates nice looking **color palettes** especially for thematic maps.

ggplot2 Reference: Reference to all features of the ggplot2 package.

allYourFigureAreBelongToUs: Various plot configurations.