Week 12 Module 4: ggplot2

Programming in R

Module Overview

In the previous module, we began our study of `ggplot2` by using the `qplot()` function.

In this module, we'll study the fundamental tools of `ggplot2`, without the `qplot()` function to serve as an interface.

We'll also consider some graphs from the popular textbook "R for Data Science", by Hadley Wickham and Garrett Grolemund.

I'll finish up by making some recommendations about where to find more material if you want to continue your study of this fascinating set of tools.

Remember, our goal here is not to cover every aspect of the `ggplot2` package, but simply to enable you to become comfortable with the framework.

Section 1: Basic Scatterplots

First, let's load in the `ggplot2` package:

```
library( "ggplot2" )
```

Now we'll construct a simple data frame:

```
x.vector <- 1:10
y.vector <- 2 * x.vector + 3

species.factor <-
    factor(
        c( "Hedgehog", "Aardvark", "Hedgehog", "Hedgehog",
           "Aardvark", "Hedgehog", "Hedgehog", "Aardvark",
           "Hedgehog", "Aardvark"
        )
    )

simple.data.frame <-
    data.frame(
        x.column = x.vector,
        y.column = y.vector,
        species.column = species.factor
    )
```

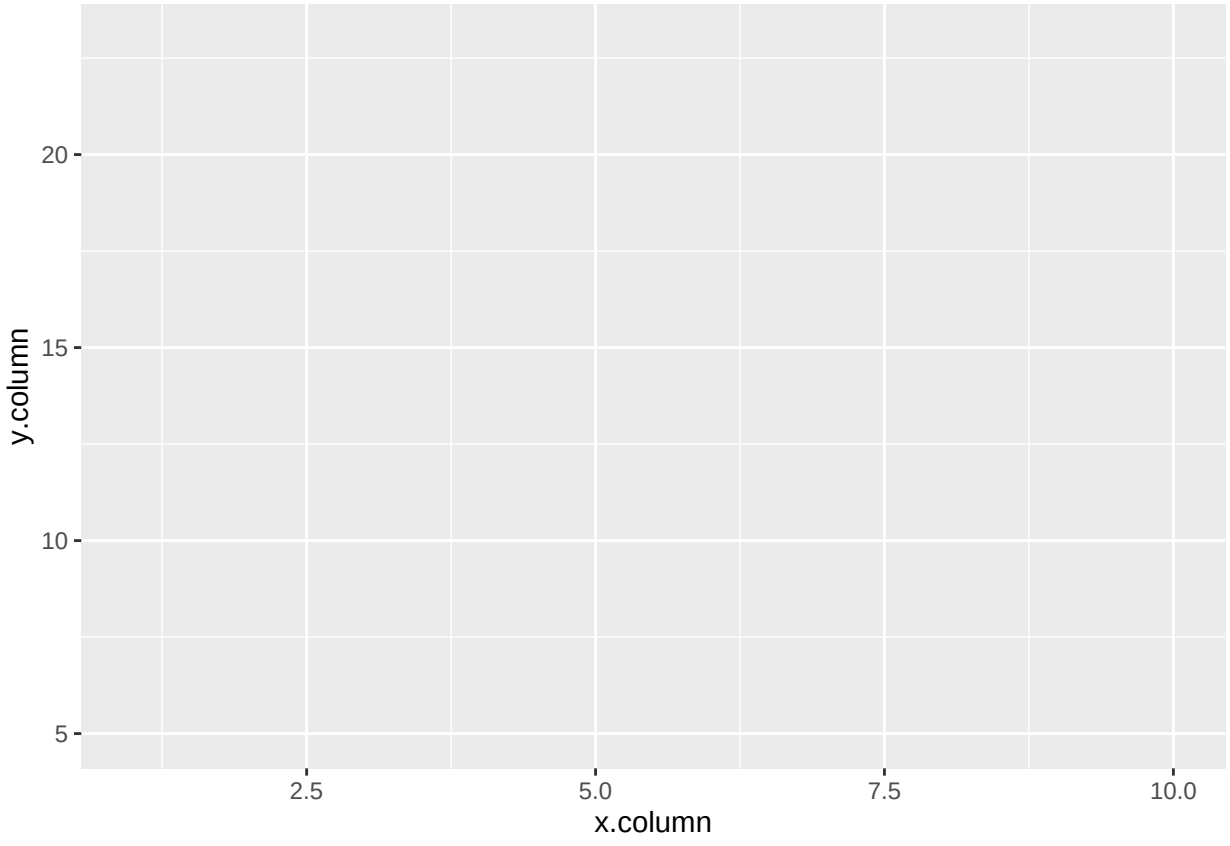
Now we'll construct a basic `ggplot2` object:

```
basic.ggplot2.object <-
    ggplot(
        data = simple.data.frame,
        aes(
            x = x.column,
            y = y.column
        )
    )
```

Note that when we use the `ggplot()` function to create this graphics object, we explicitly state the aesthetic mappings using the `aes()` function.

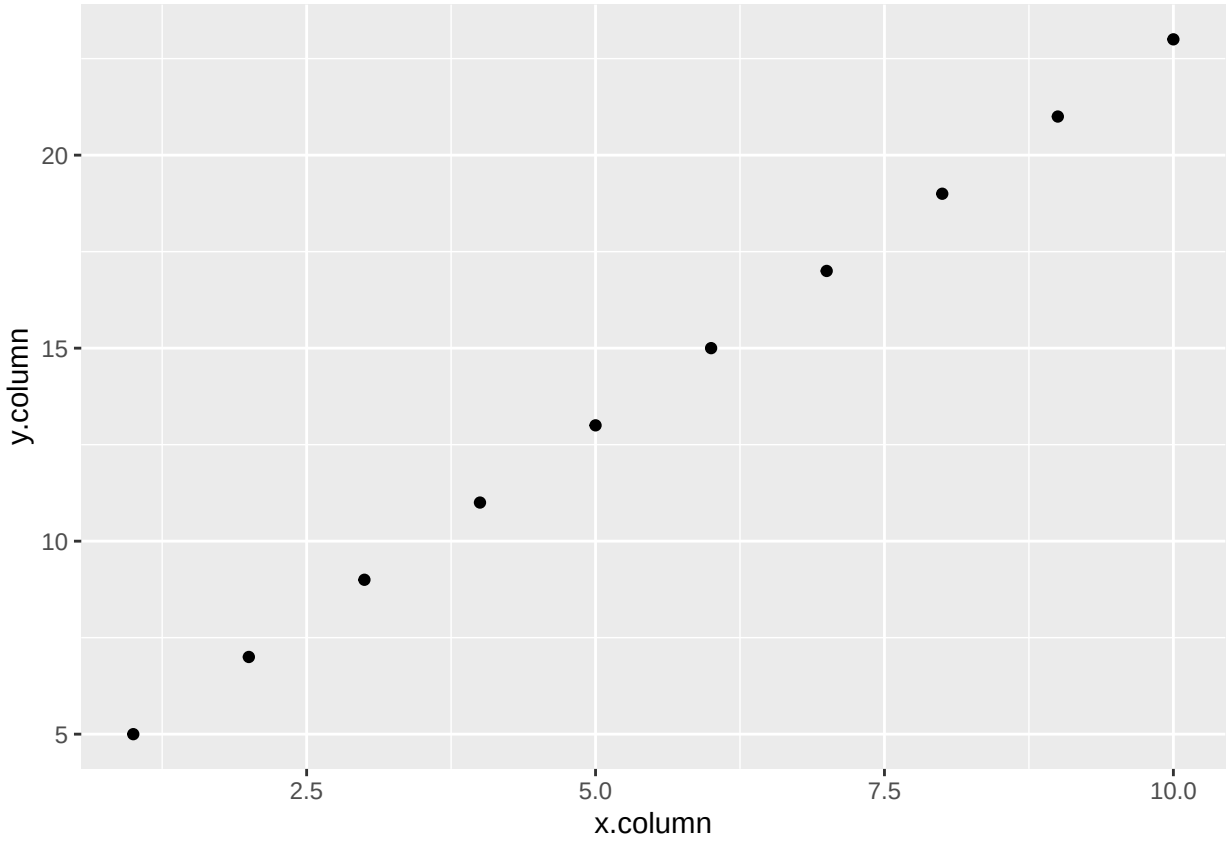
This `ggplot2` object isn't very interesting to look at, because we haven't specified any view using a `geom`.

```
basic.ggplot2.object
```



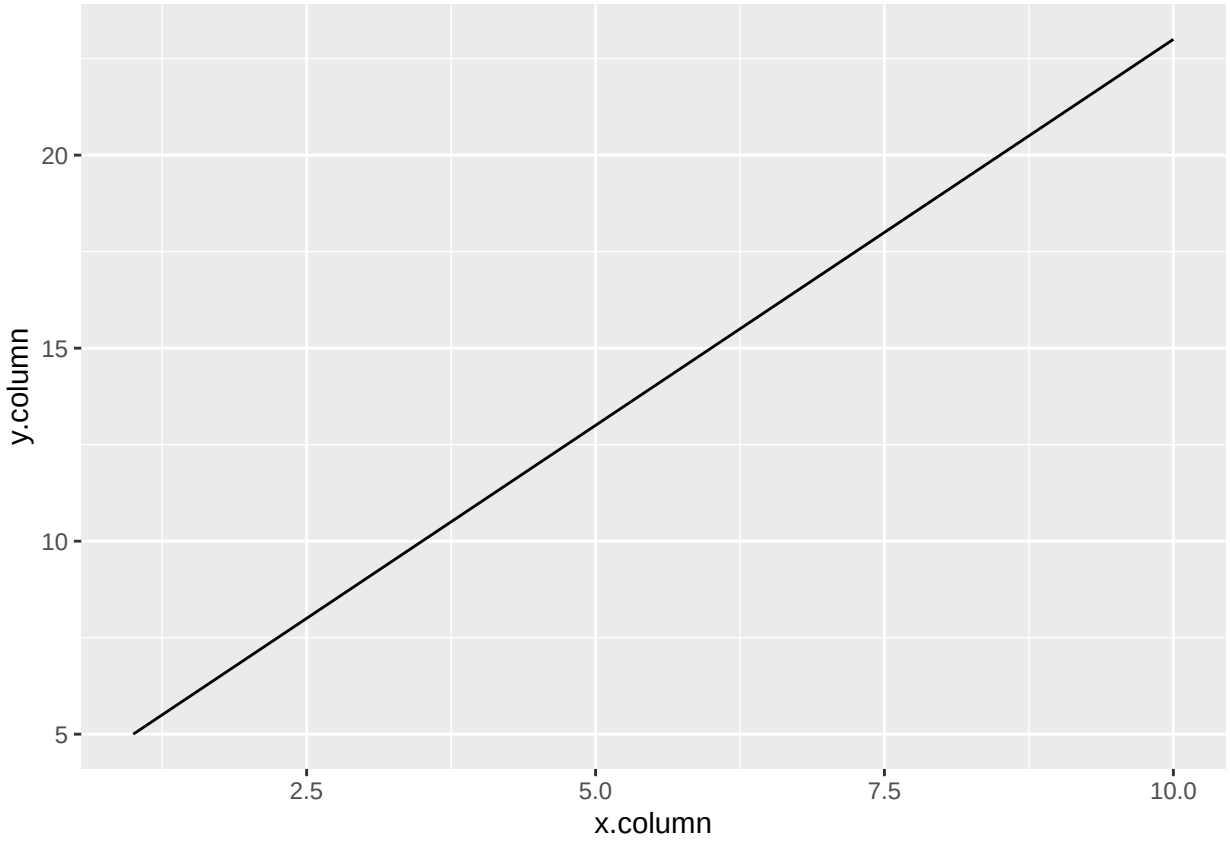
Now we can create a scatterplot by using the `geom_point()` function:

```
basic.ggplot2.object +
    geom_point()
```



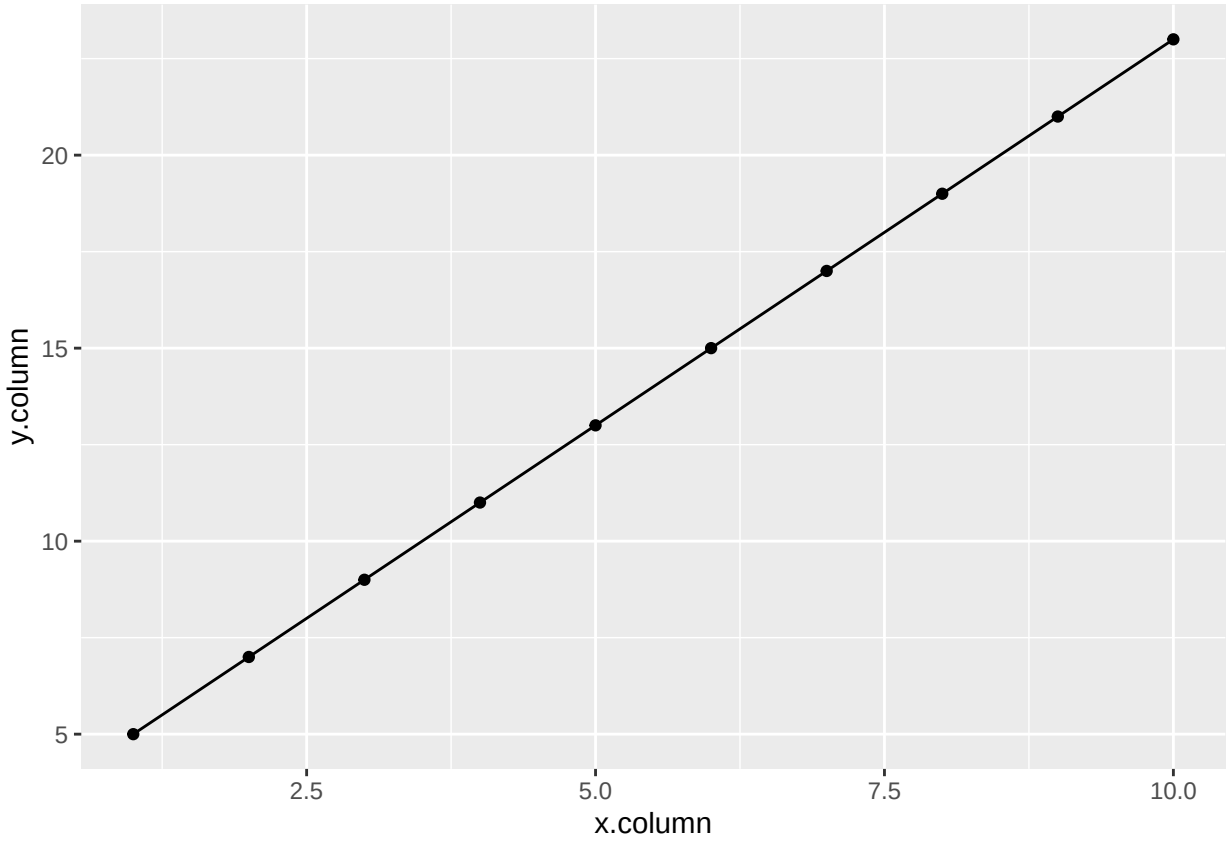
We can view this with a line using the `geom_line()` function:

```
basic.ggplot2.object +
    geom_line()
```



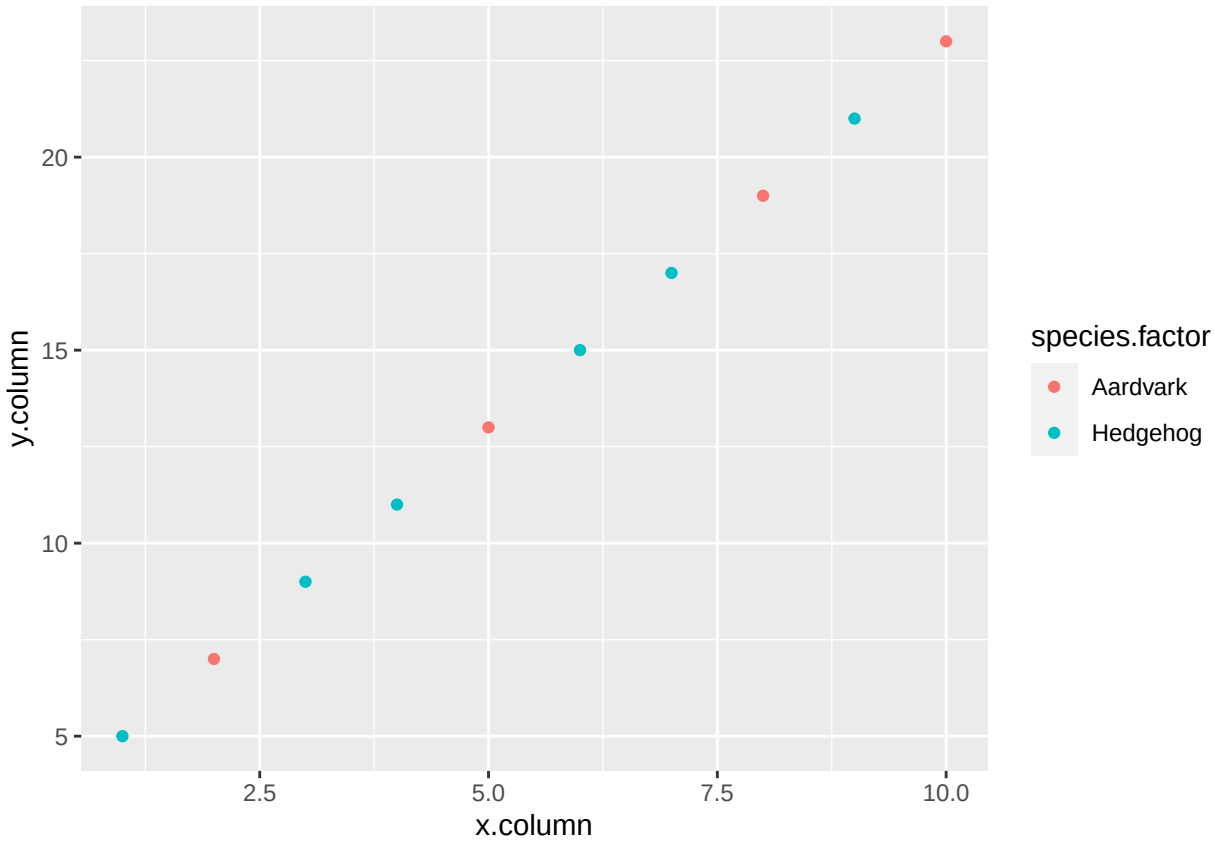
We can view both points and lines by adding both `geom_point()` and `geom_line()` to the graph:

```
basic.ggplot2.object +
    geom_point() +
    geom_line()
```



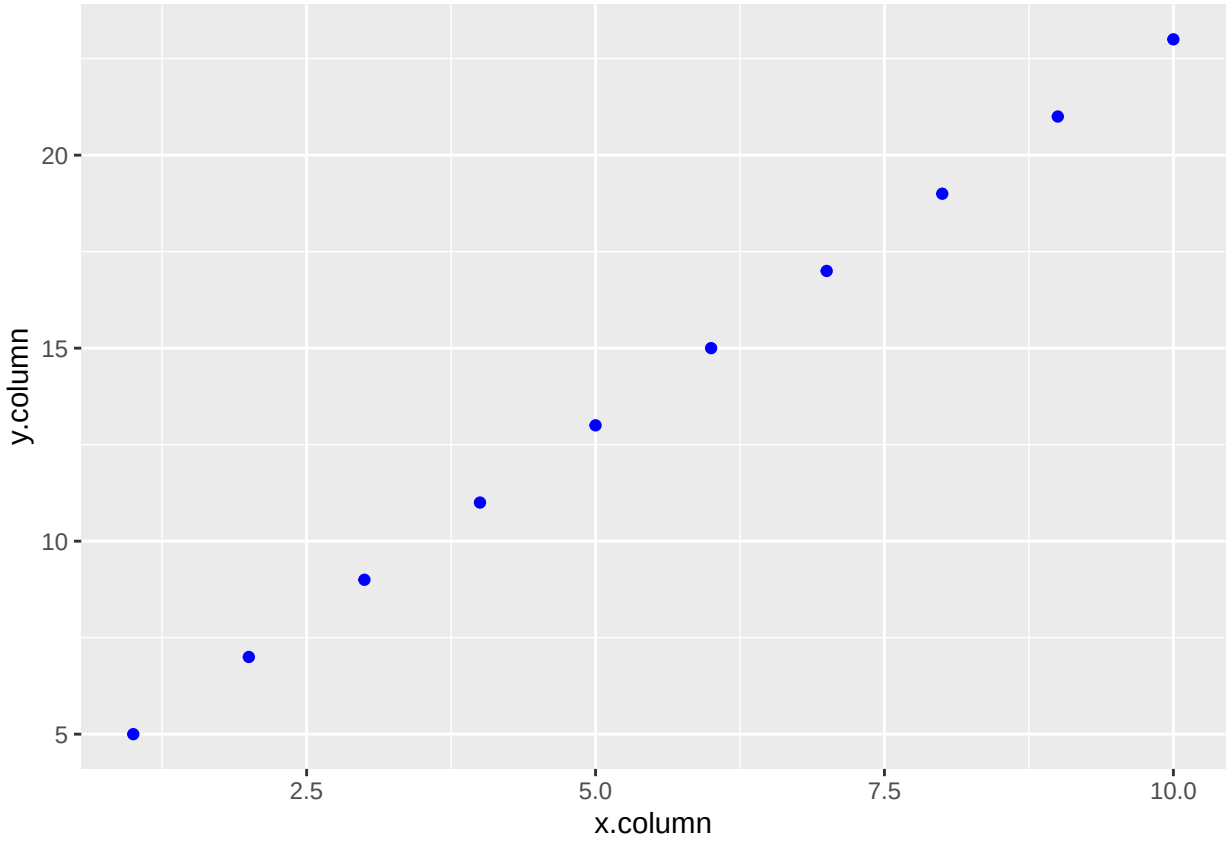
Let's change the color of the points to indicate the species:

```
basic.ggplot2.object +
    geom_point(
        aes( col = species.factor ),
        data = simple.data.frame
    )
```



Instead, if we just want to make all the points have the color "blue", we can write:

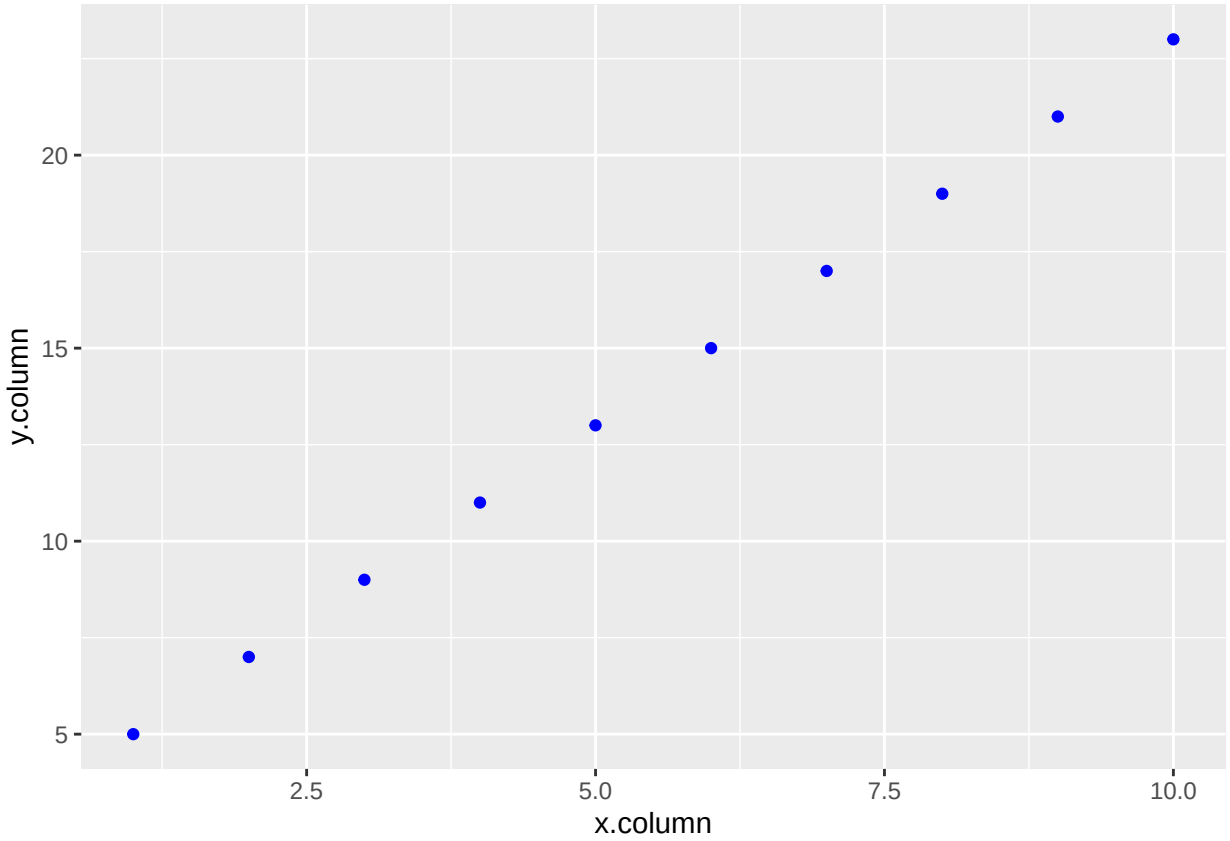
```
basic.ggplot2.object +
    geom_point( col = "blue" )
```



Hey!

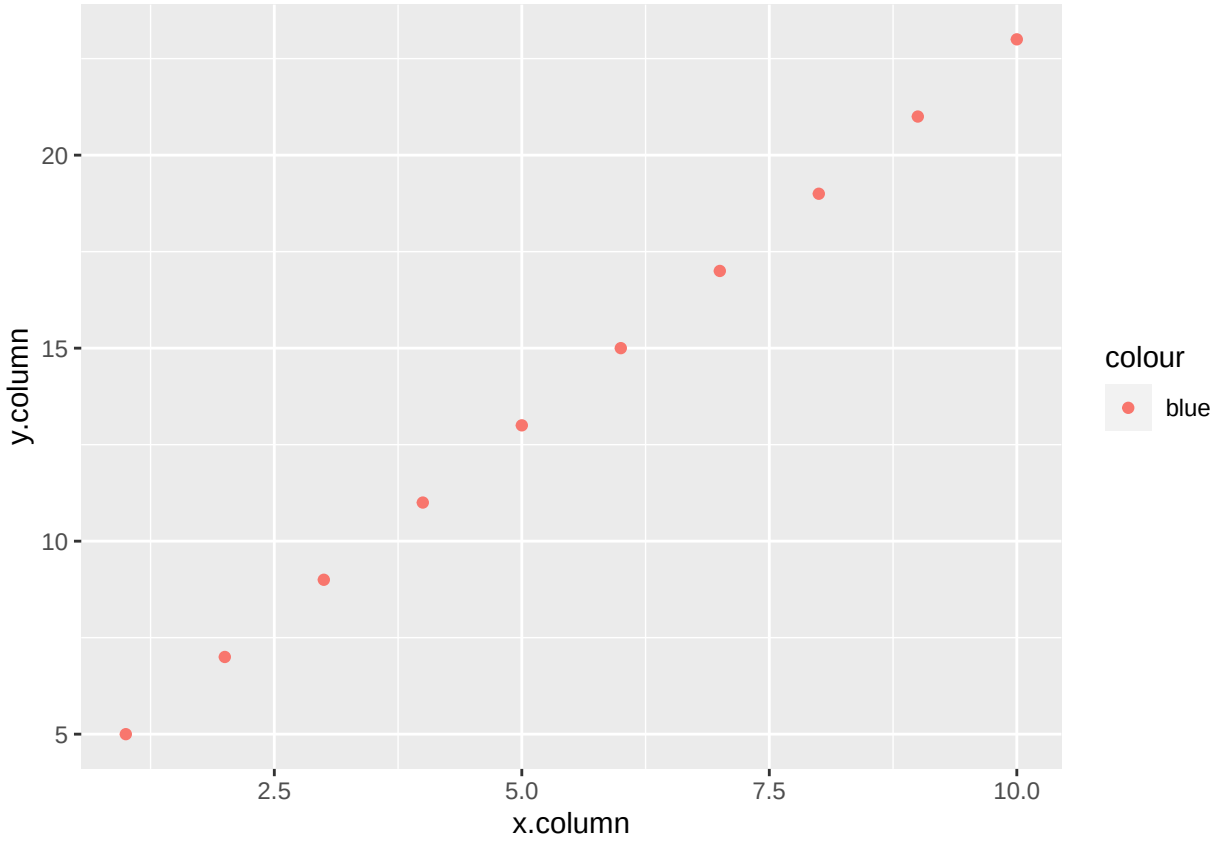
When we were using `qplot` and we wanted to color the points "blue", we had to use the `I()` function to suppress evaluation:

```
qplot(
    x.column,
    y.column,
    data = simple.data.frame,
    geom = "point",
    col = I( "blue" )
)
```



If we didn't use that `I()` function, then we ended up with this:

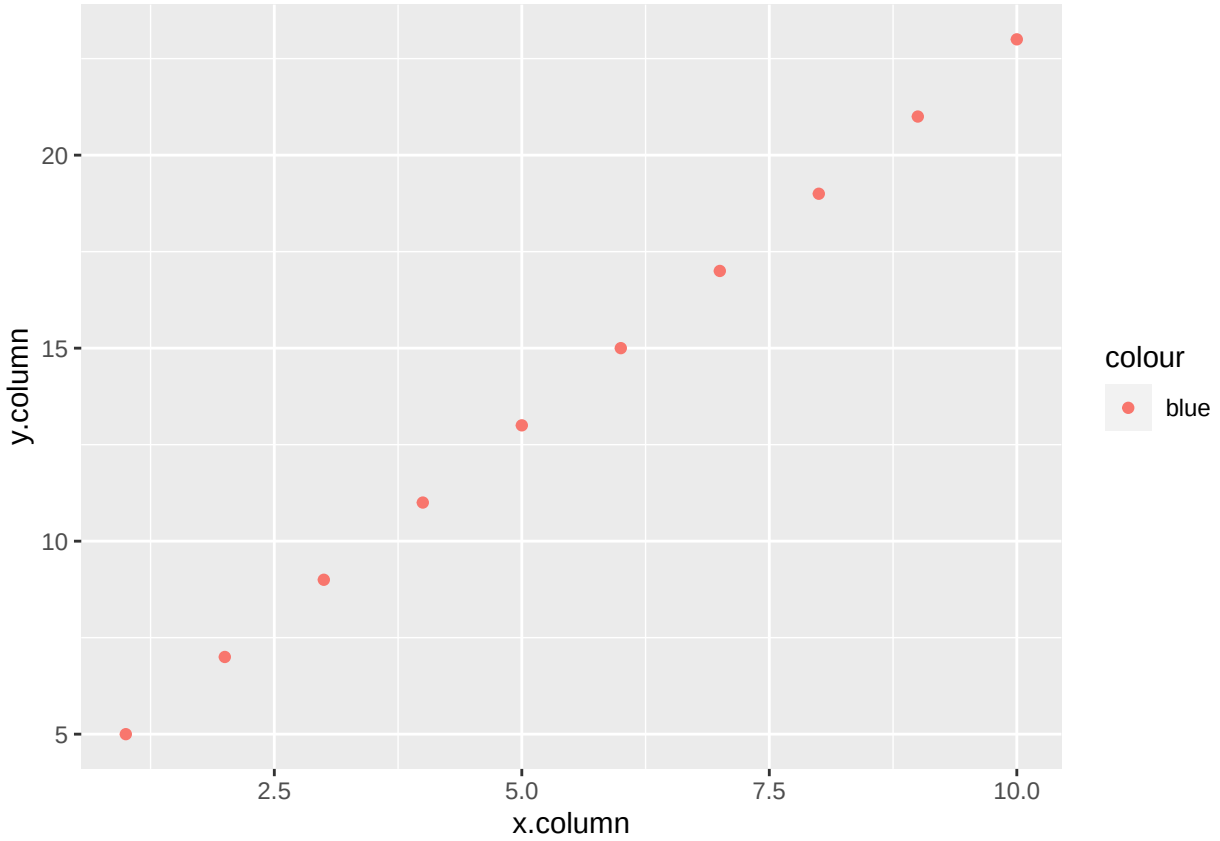
```
qplot(
    x.column,
    y.column,
    data = simple.data.frame,
    geom = "point",
    col = "blue"
)
```



That's because `qplot()` makes the default assumption that when you write `col = "blue"` that you are trying to associated the constant value "blue" with the `col` aesthetic.

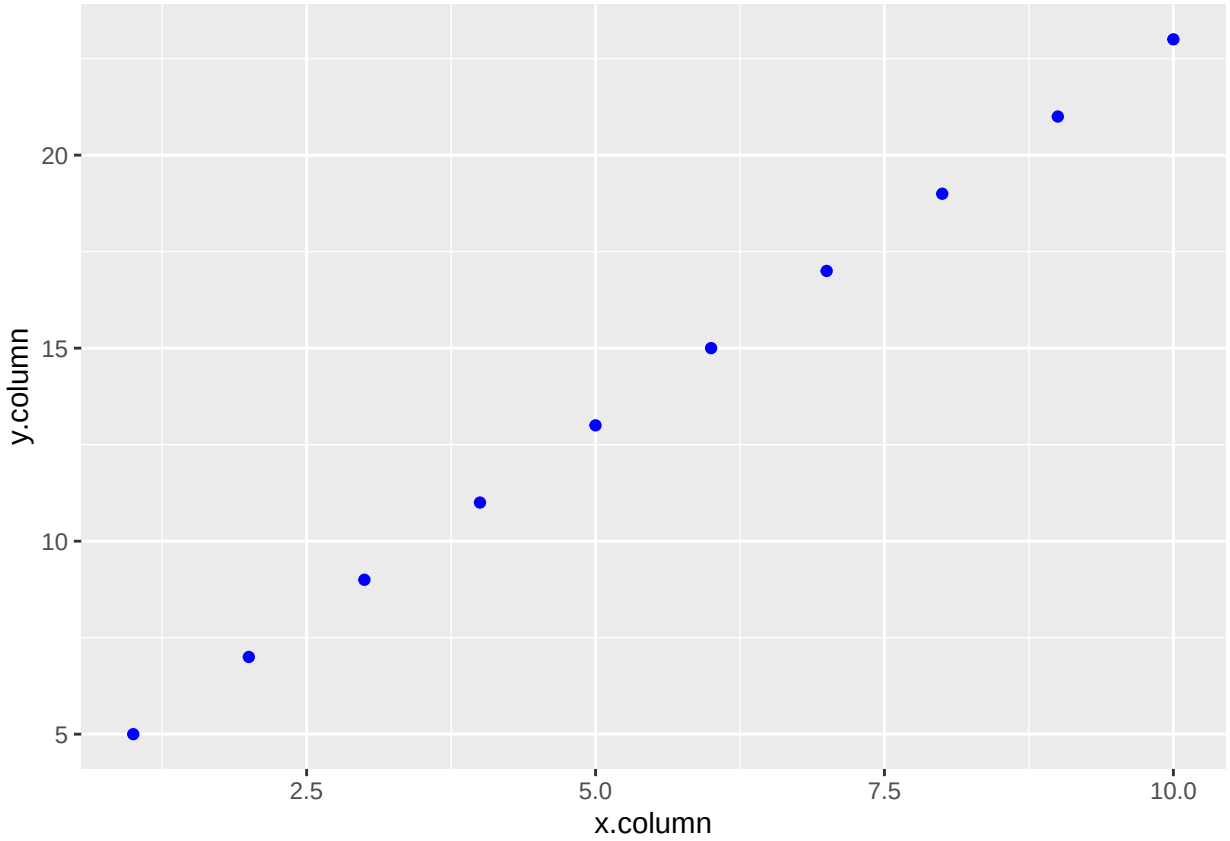
Essentially, `qplot()` is translating your code into this:

```
basic.ggplot2.object +
    geom_point(
        aes( col = "blue" )
    )
```



However, if we don't put `col = "blue"` inside a call to `aes()`, then it won't be treated as an aesthetic mapping, and instead it's just assigning the value "blue" to the color of the points: Essentially, `qplot()` is translating your code into this:

```
basic.ggplot2.object +
    geom_point(
        col = "blue"
    )
```



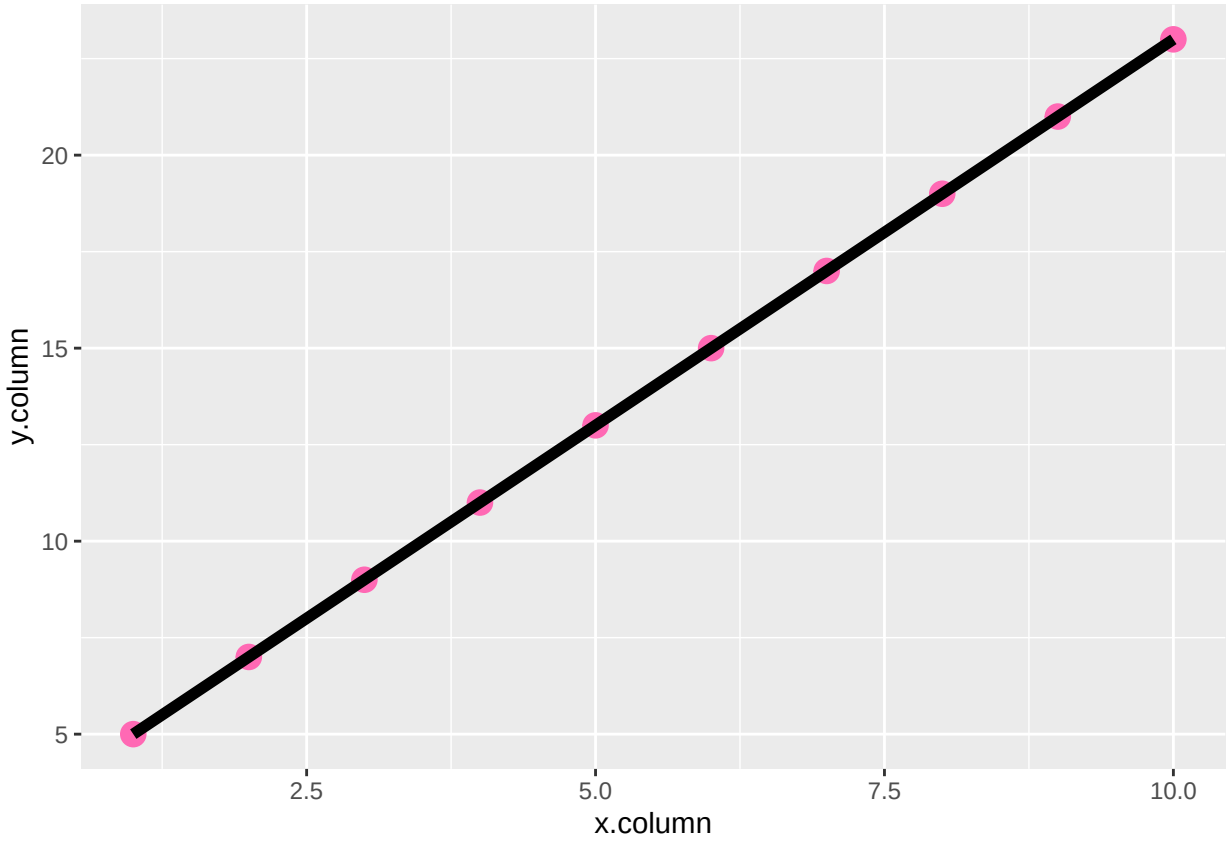
Simple!

Of course, I'm being facetious here.

It's actually somewhat subtle, and this example illustrates how you really have to be sensitive to the concept of an aesthetic mapping.

OK, let's make the line a little thicker, and the points larger with the color "hotpink":

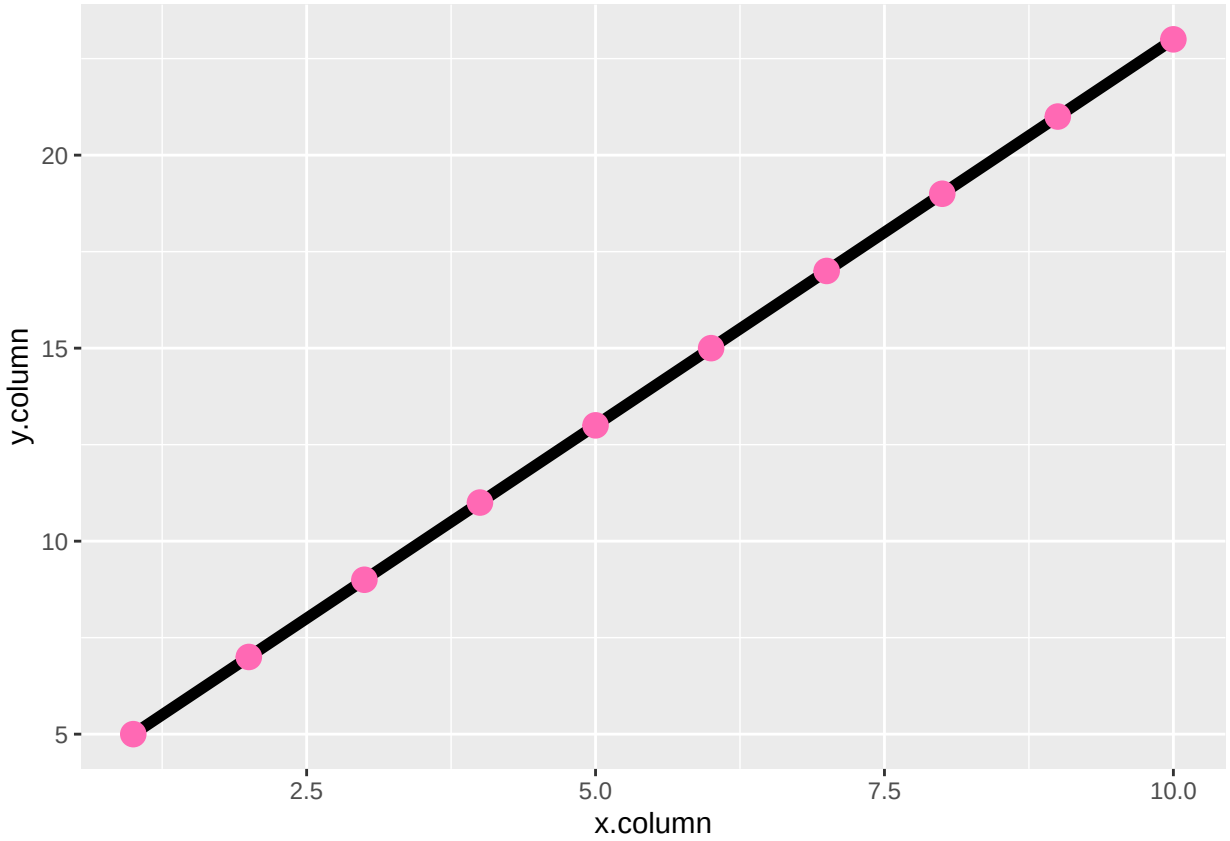
```
basic.ggplot2.object +
    geom_point( size = 4, col = "hotpink" ) +
    geom_line( size = 2 )
```



That doesn't look so great, because the thick black line is covering up the points.

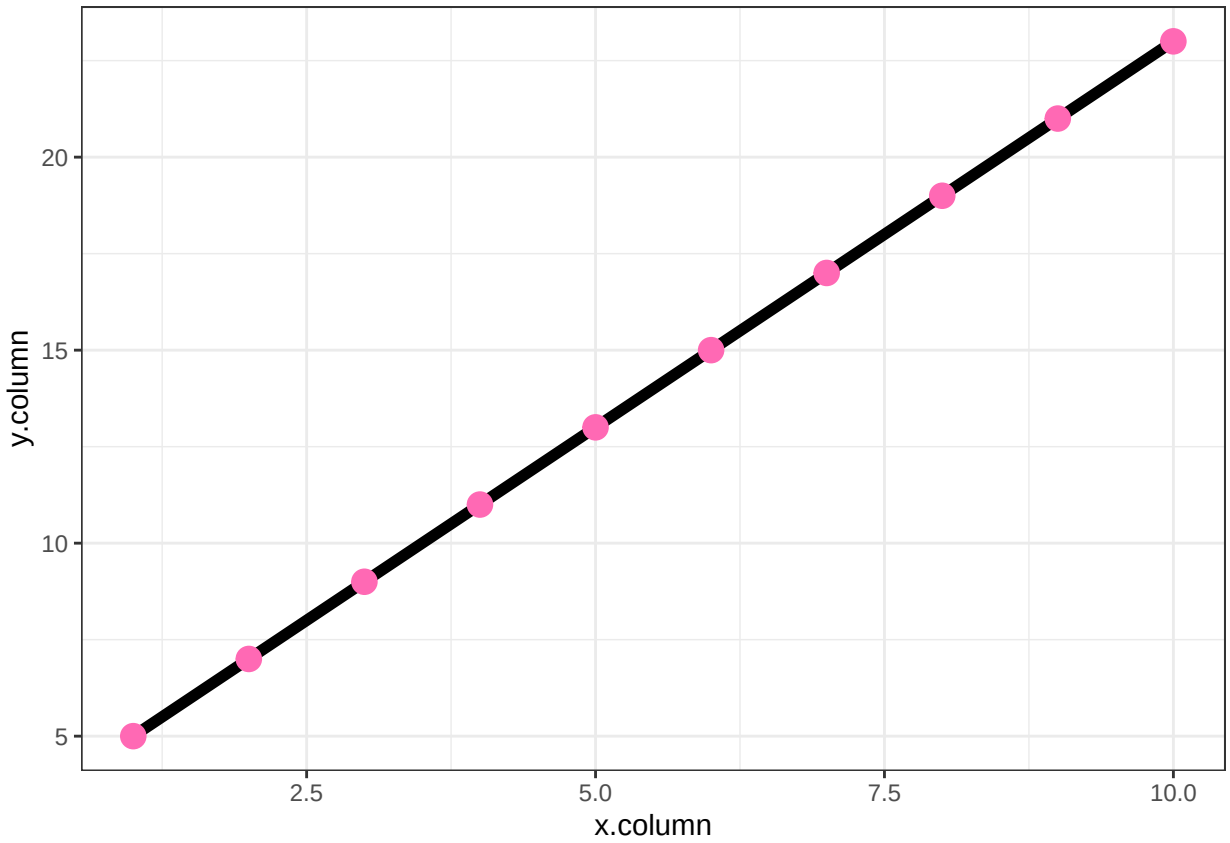
We can obtain a nicer graph by adding the `geom` for the line to the `ggplot2` object first, and then the `geom` for the points:

```
basic.ggplot2.object +
    geom_line( size = 2 ) +
    geom_point( size = 4, col = "hotpink" )
```



Finally, let's get rid of that gray background:

```
basic.ggplot2.object +
    geom_line( size = 2 ) +
    geom_point( size = 4, col = "hotpink" ) +
    theme_bw()
```



Section 2: Constructing a Histogram

Now let's see how to construct histograms using the basic `ggplot2` framework.

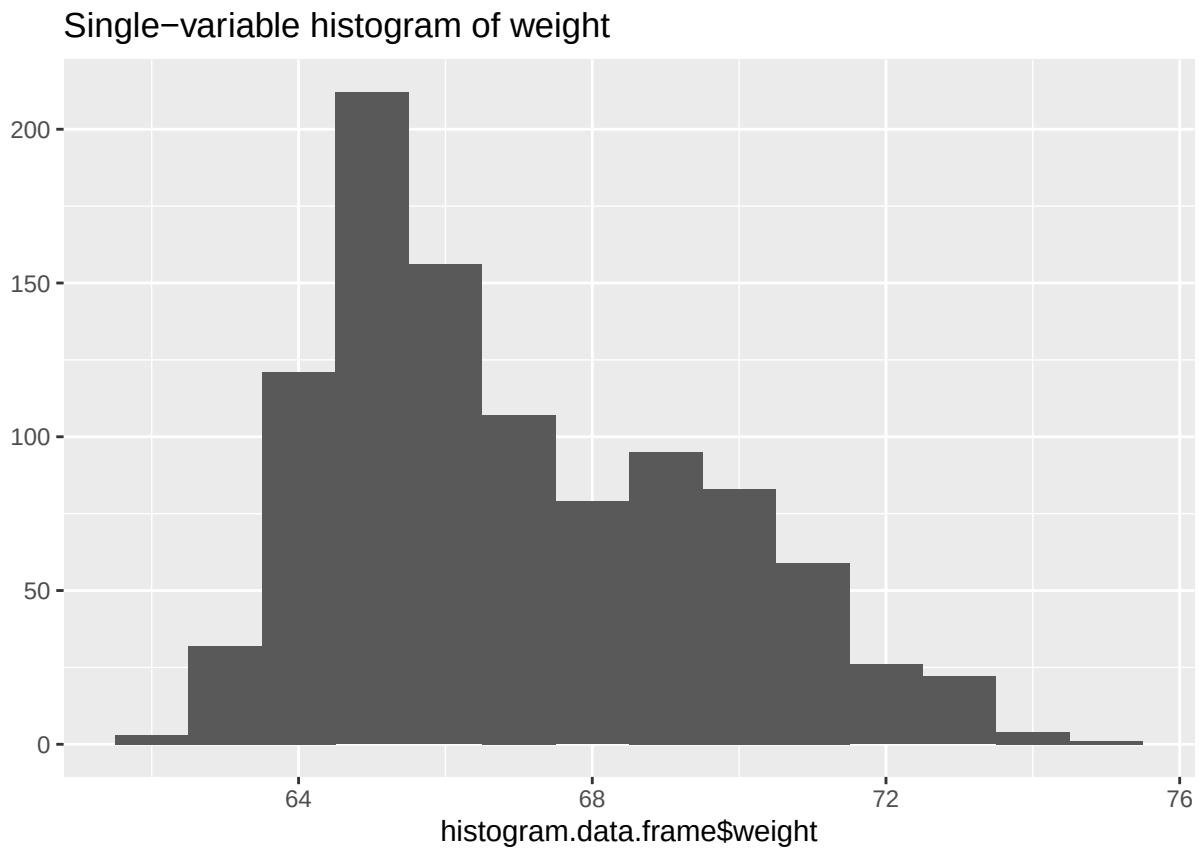
First, let's make some data to work with:

```
set.seed( 1 )

histogram.data.frame <-
    data.frame(
        weight =
            c(
                rnorm(500, mean = 65, sd = 1),
                rnorm(500, mean = 69, sd = 2)
            )
    )
```

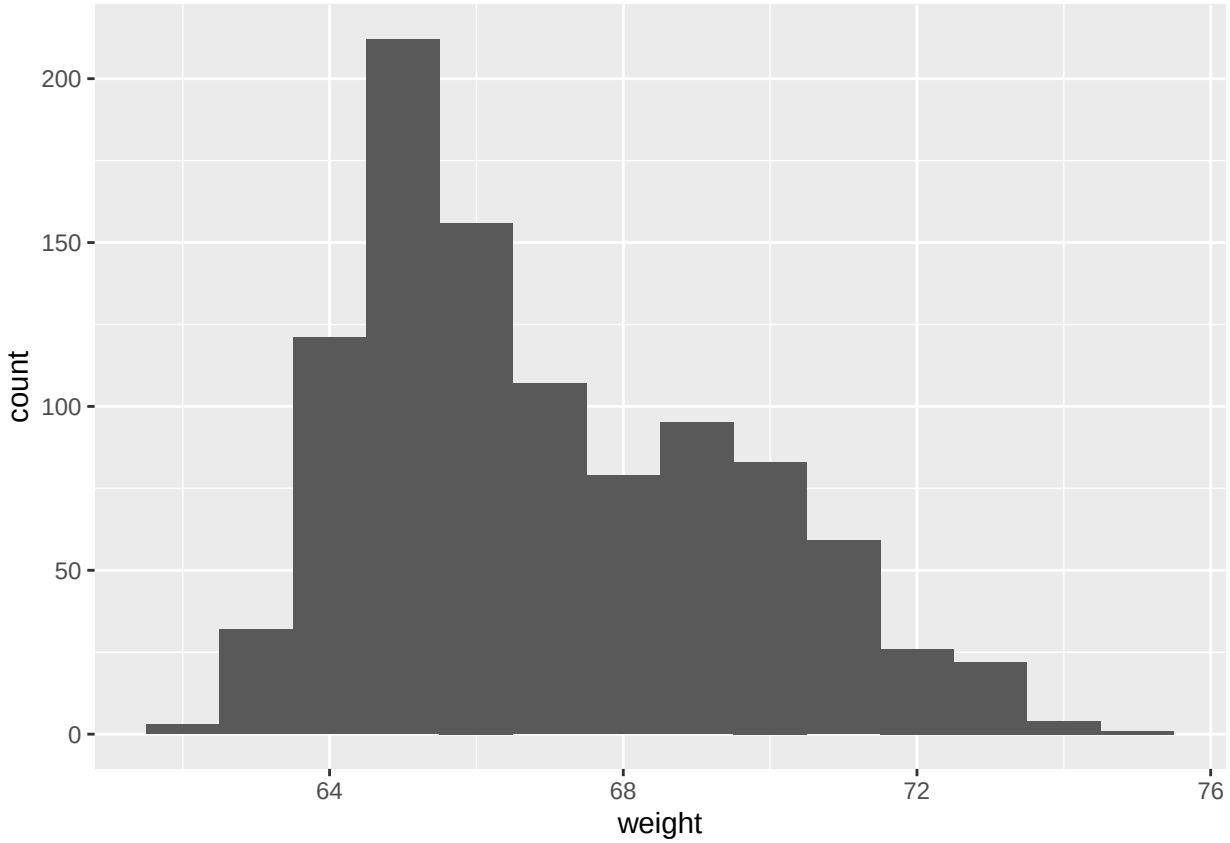
Here's the simple `qplot()` version of the histogram:

```
qplot(
    histogram.data.frame$weight,
    main = "Single-variable histogram of weight",
    binwidth = 1
)
```



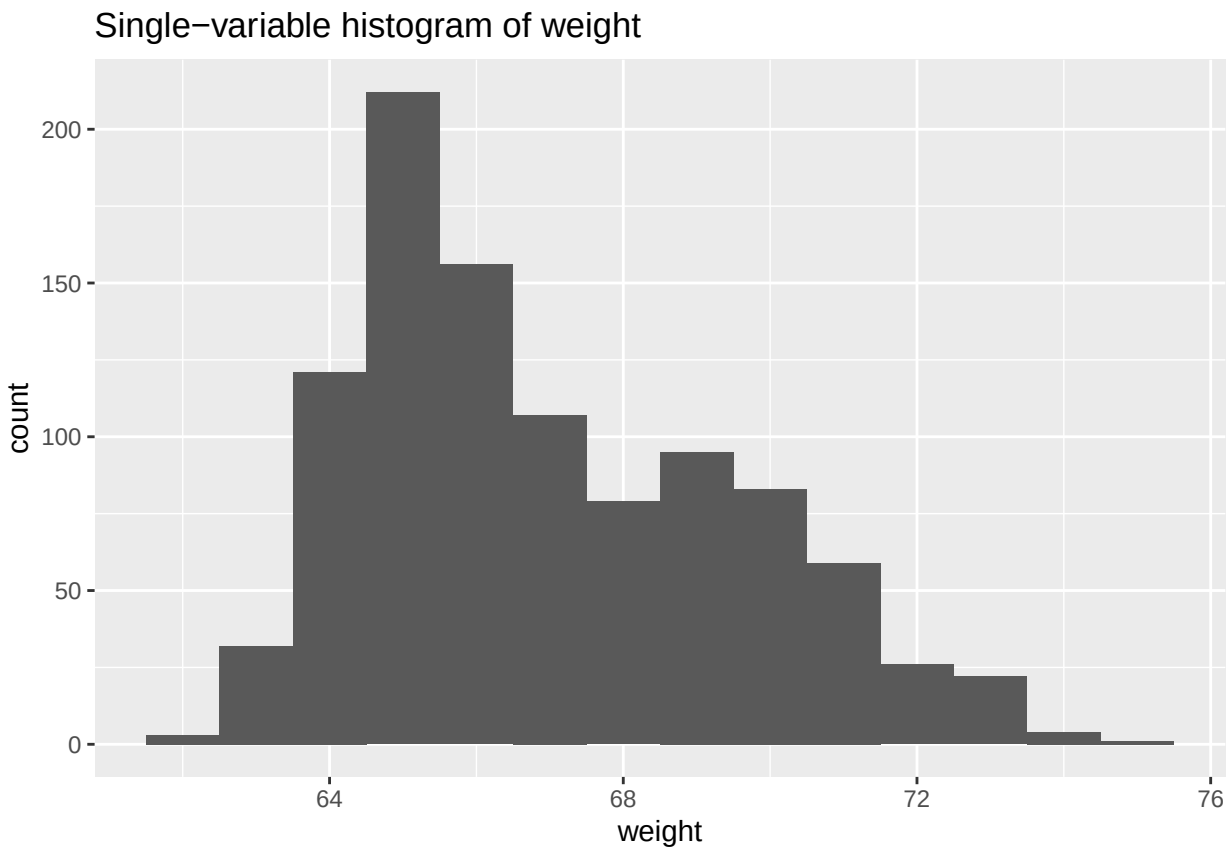
Now we'll do this using the basic `ggplot2` method:

```
ggplot(
    data = histogram.data.frame,
    aes(x = weight)
) +
    geom_histogram( binwidth = 1 )
```



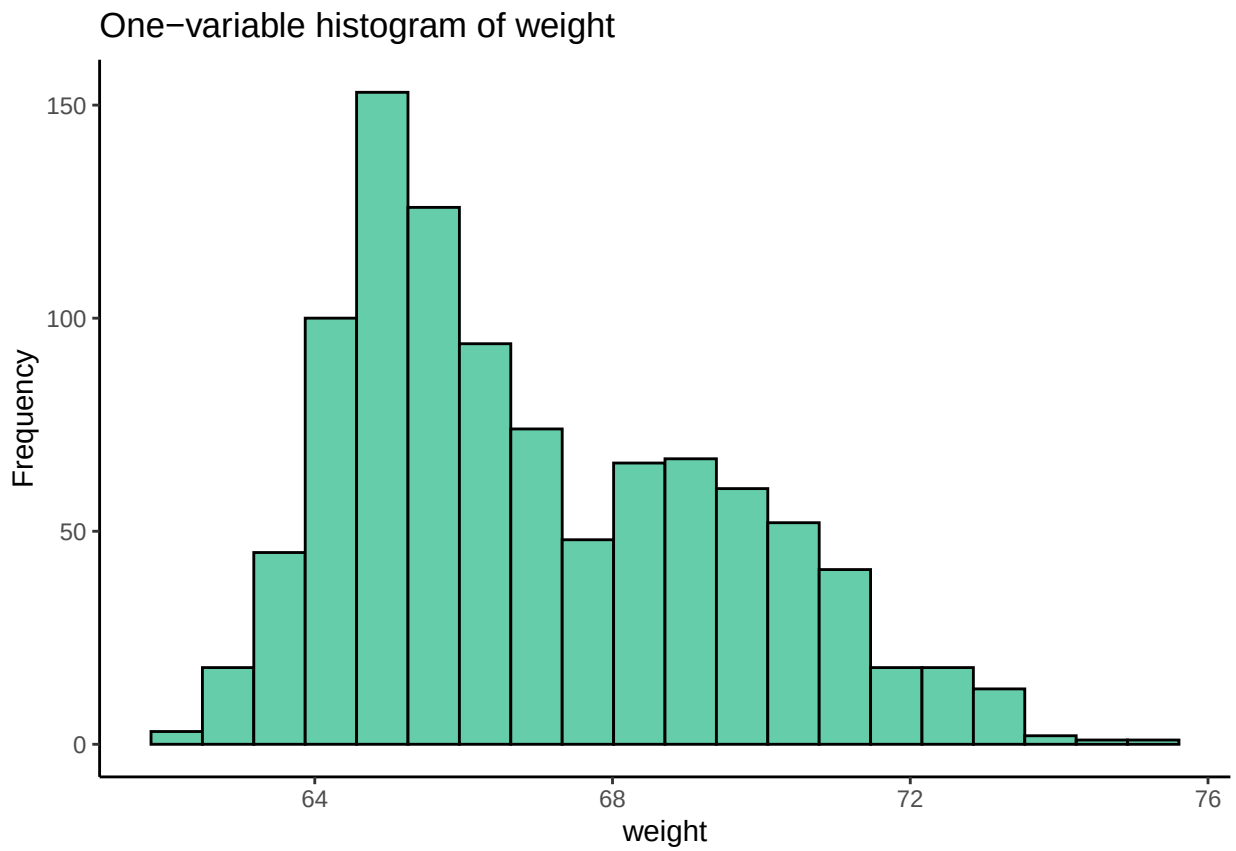
We can add a main title to this graph by using `labs()`:

```
ggplot(
    data = histogram.data.frame,
    aes(x = weight)
) +
    geom_histogram( binwidth = 1 ) +
    labs( title = "Single-variable histogram of weight" )
```



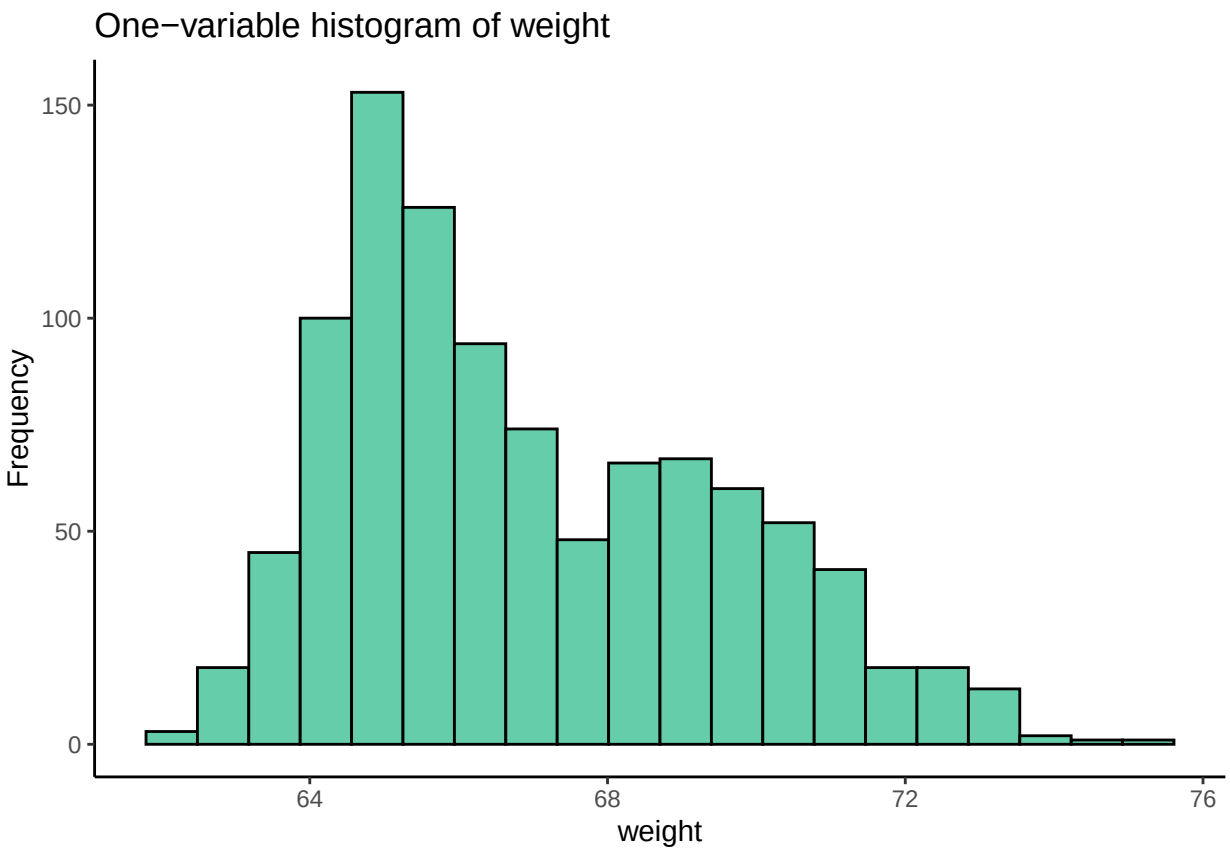
We also made a fancier version of this histogram using the `qplot()` function:

```
qplot( weight,
       data = histogram.data.frame,
       geom = "histogram",
       main = "One-variable histogram of weight",
       ylab = "Frequency",
       fill = I( "aquamarine3" ),
       color = I( "black"),
       bins = 20 ) +
   theme_classic()
```



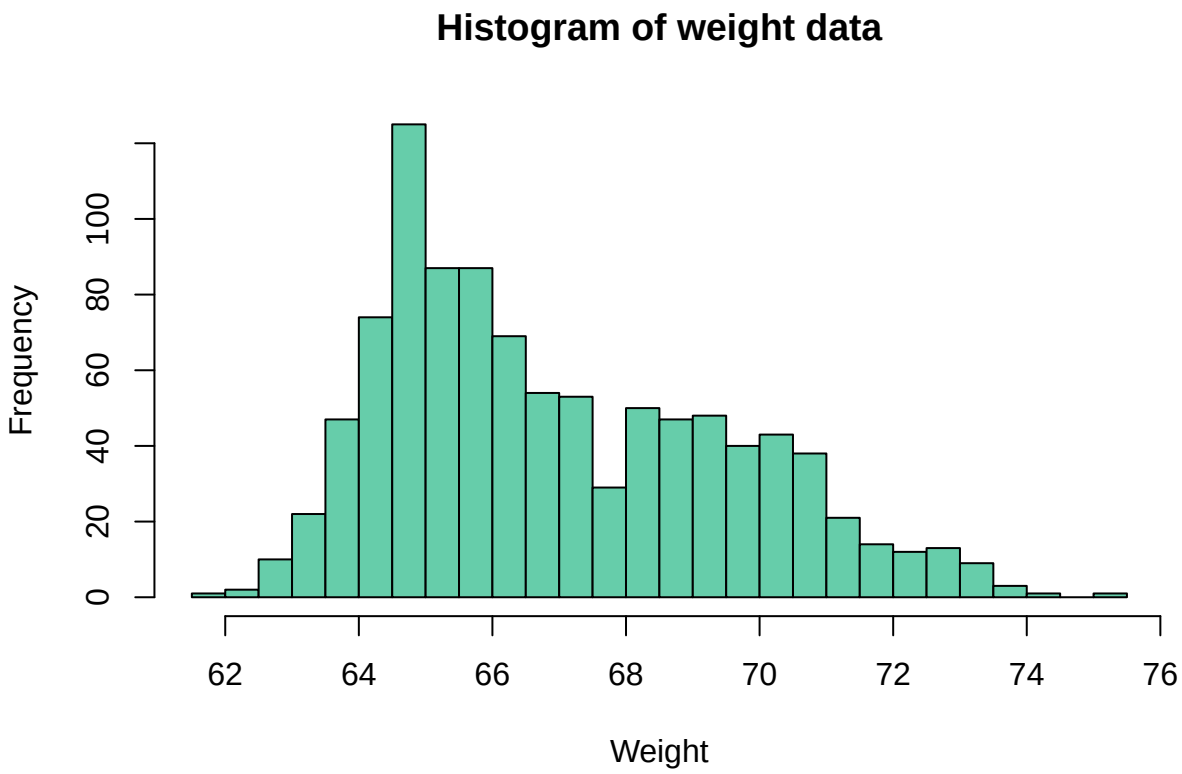
We can use the basic `ggplot2()` machinery to create this histogram:

```
ggplot(
    data = histogram.data.frame,
    aes(x = weight)
) +
    geom_histogram(
        fill = "aquamarine3",
        color = "black",
        bins = 20
    ) +
    labs( title = "One-variable histogram of weight",
          y = "Frequency" ) +
    theme_classic()
```



By contrast, here's a similar histogram constructed using the `hist()` graphics function from base R:

```
hist(
    histogram.data.frame$weight,
    main = "Histogram of weight data",
    xlab = "Weight",
    ylab = "Frequency",
    col = "aquamarine3",
    breaks = 20
)
```



One important difference between the two histograms is that we specified the number of bins as 20, and in the `ggplot2` graph we really did get 20 bins (count them!).

In base R graphics, as we've mentioned, R takes the request for 20 bins a simply a suggestion, and uses an internal algorithm to decide on the actual number of bins, so that the bins line up "nicely" with the integer values on the x-axis.

In this example, it seems to me that R is actually using 28 bins.

Notice that in the `ggplot2` graph the bins do *not* line up nicely with the integer ticks on the x-axis.

Section 3: Case Study

Let's take a look at some code from the book "R for Data Science", by Hadley Wickham and Garrett Grolemund.

The first chapter of the book is "Data Visualization with ggplot2", and the authors focus on the `mpg` dataset that is included with the `ggplot2` package.

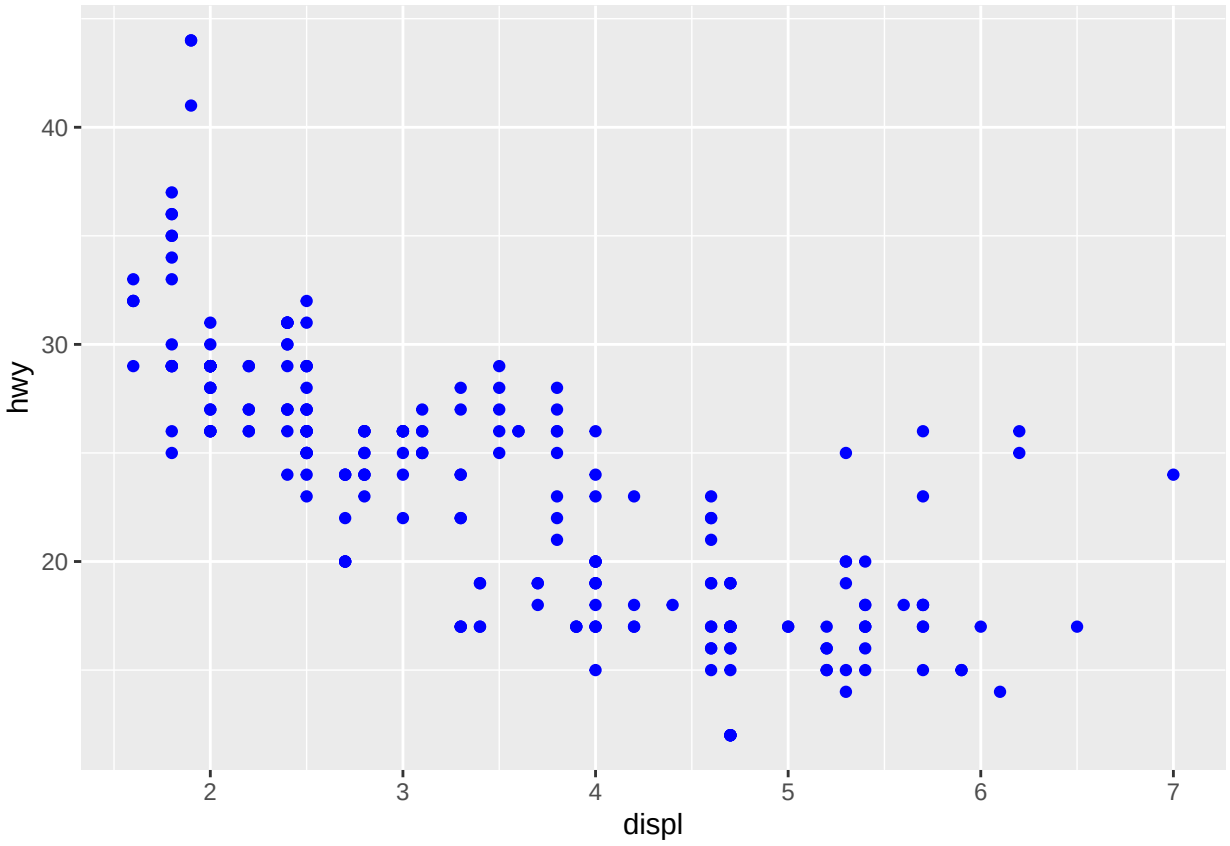
Let's take a look at some of the graphs from the section "Geometric Objects".

We'll start by creating a basic `ggplot2` graphics object for the data in `mpg`:

```
basic.mpg.object <-
    ggplot( mpg )
```

Now let's create a scatterplot of the highway mileage (denoted 'hwy') against the displacement (denoted `displ`):

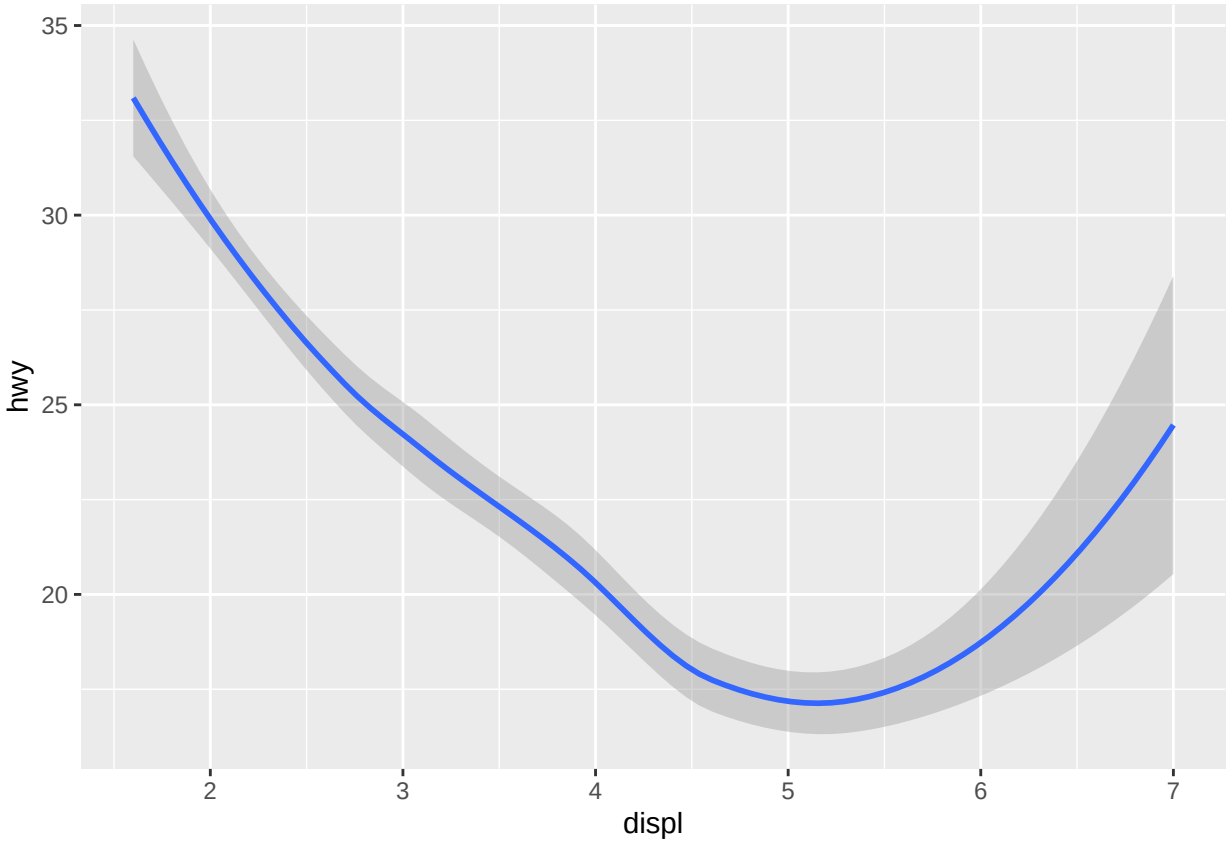
```
basic.mpg.object +
    geom_point(
        mapping = aes( x = displ, y = hwy),
        color = "blue"
    )
```



Instead of plotting the individual data points using a scatterplot, let's instead visualize this with a smoothed curve:

```
basic.mpg.object +
    geom_smooth( aes(x = displ, y = hwy) )
```

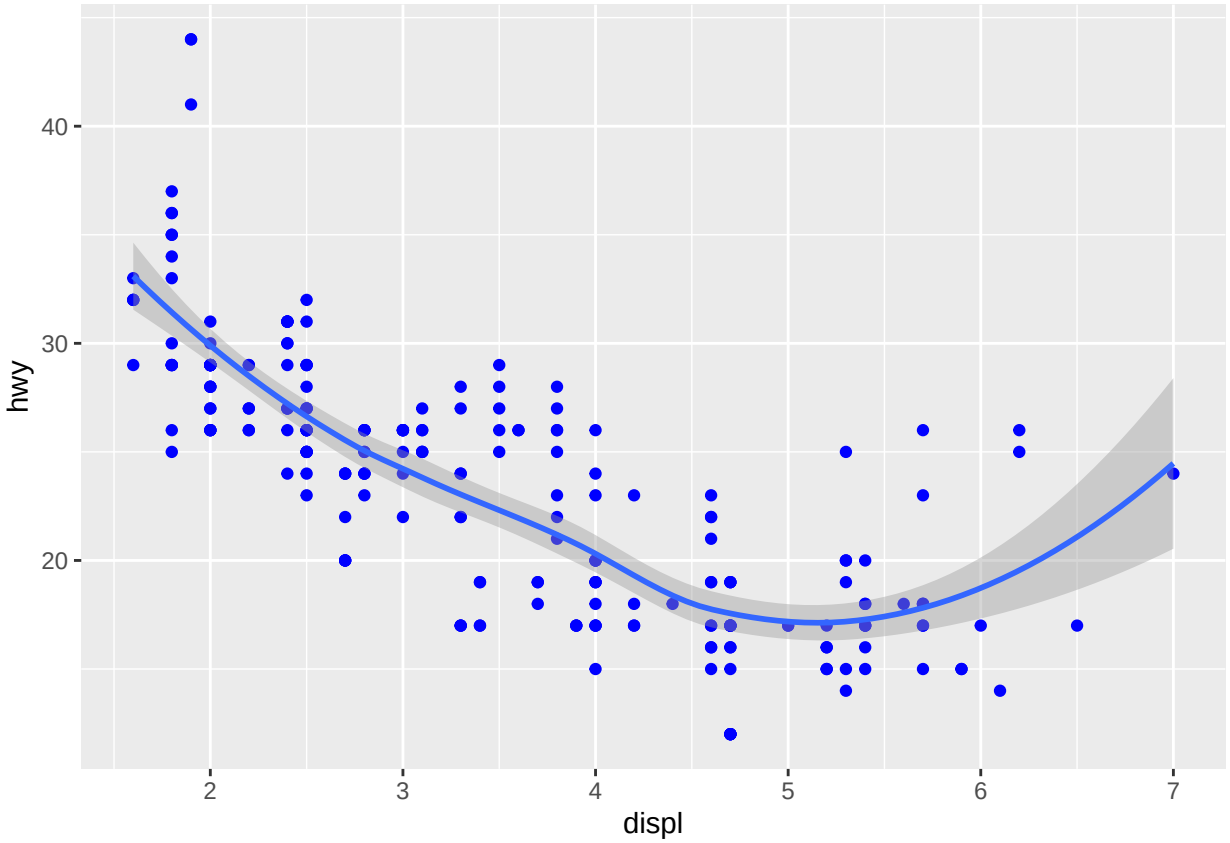
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



We can include both of these visualizations in the graph by adding the two `geom()` objects to the `basic.mpg.object`:

```
basic.mpg.object +
    geom_point(
        mapping = aes( x = displ, y = hwy),
        color = "blue"
    ) +
    geom_smooth( aes(x = displ, y = hwy) )
```

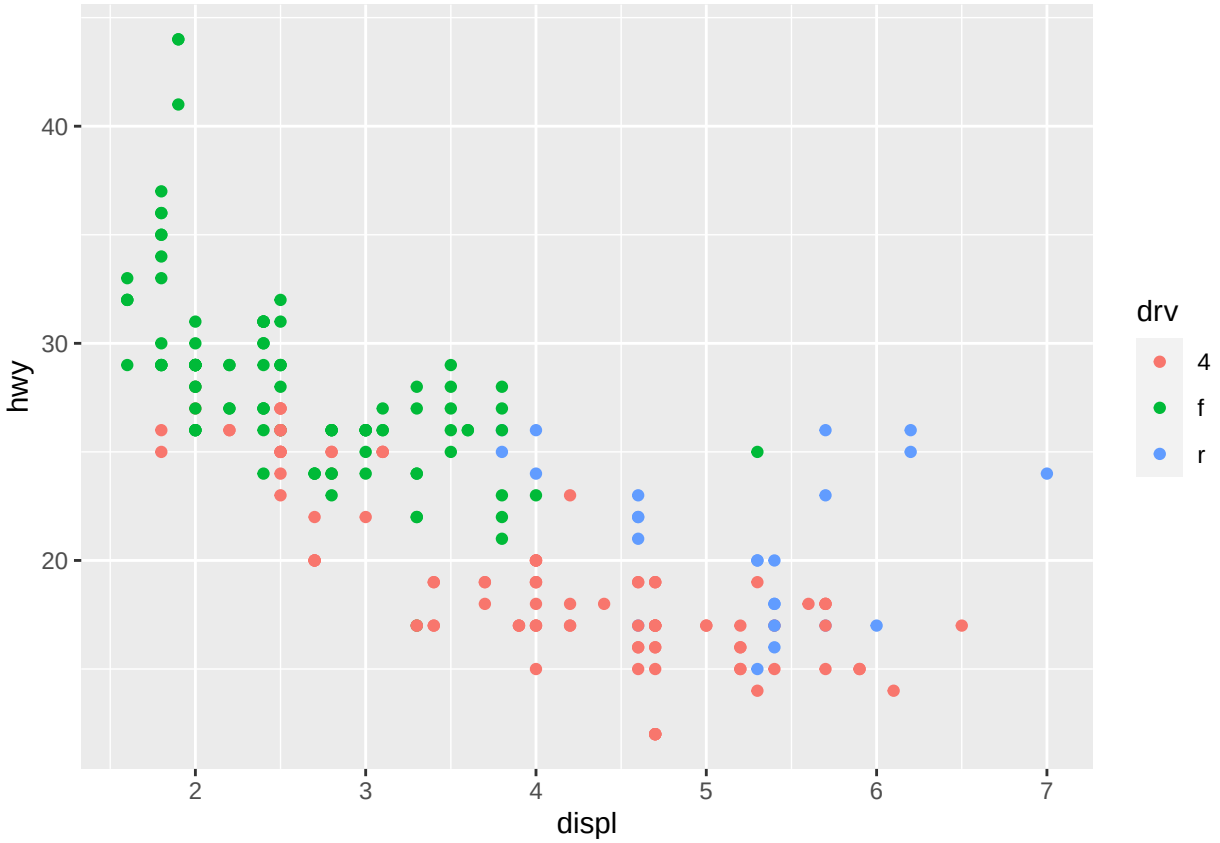
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Another variable in the `mpg` data frame is `drv`, which indicates whether or not the vehicle drive is front-wheel, rear-wheel, or 4-wheel.

Let's display the scatterplot, using the color of the points to indicate the drive category of the vehicle:

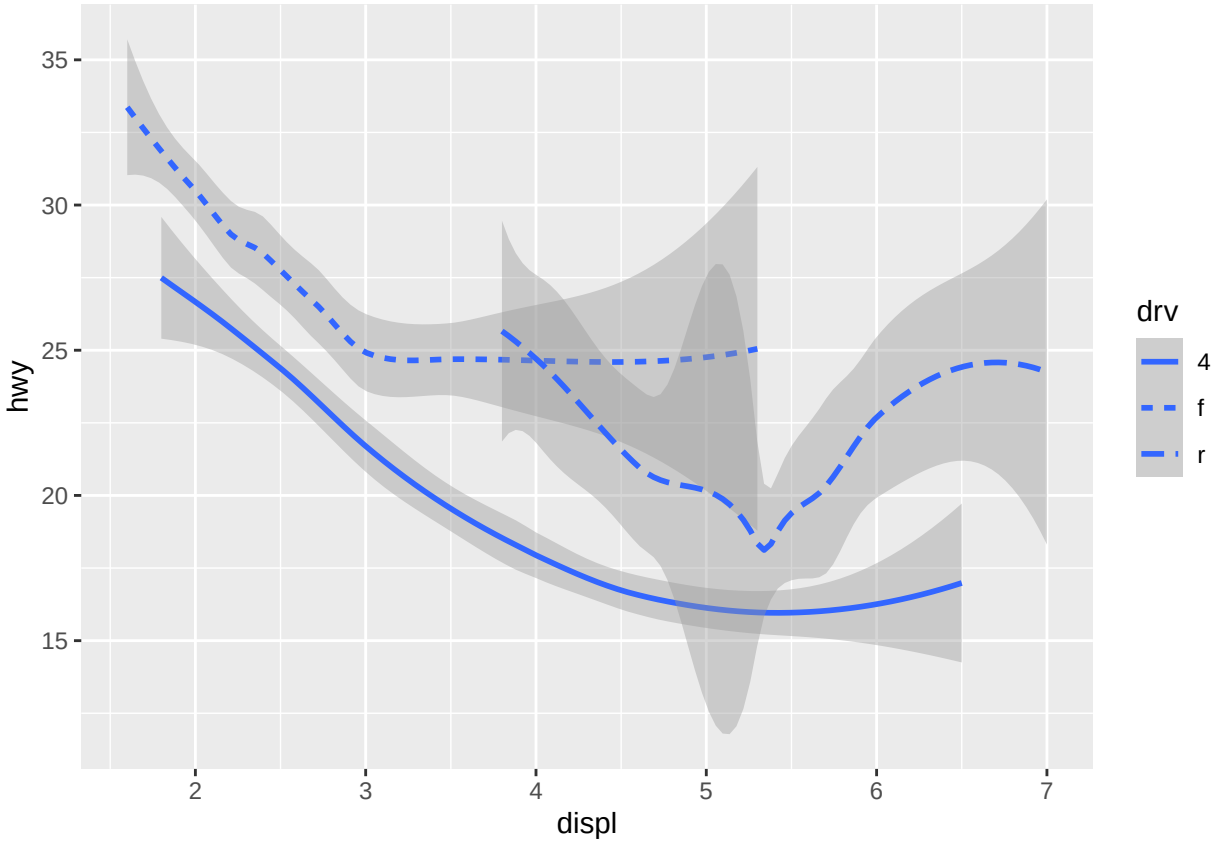
```
basic.mpg.object +
    geom_point(
        mapping =
            aes(
                x = displ,
                y = hwy,
                color = drv
            )
    )
```



Now let's visualize the smoothed regression curves for each level of `drive` by mapping this variable to the `linetype` aesthetic:

```
basic.mpg.object +
    geom_smooth(
        aes(x = displ, y = hwy, linetype = drv)
    )
```

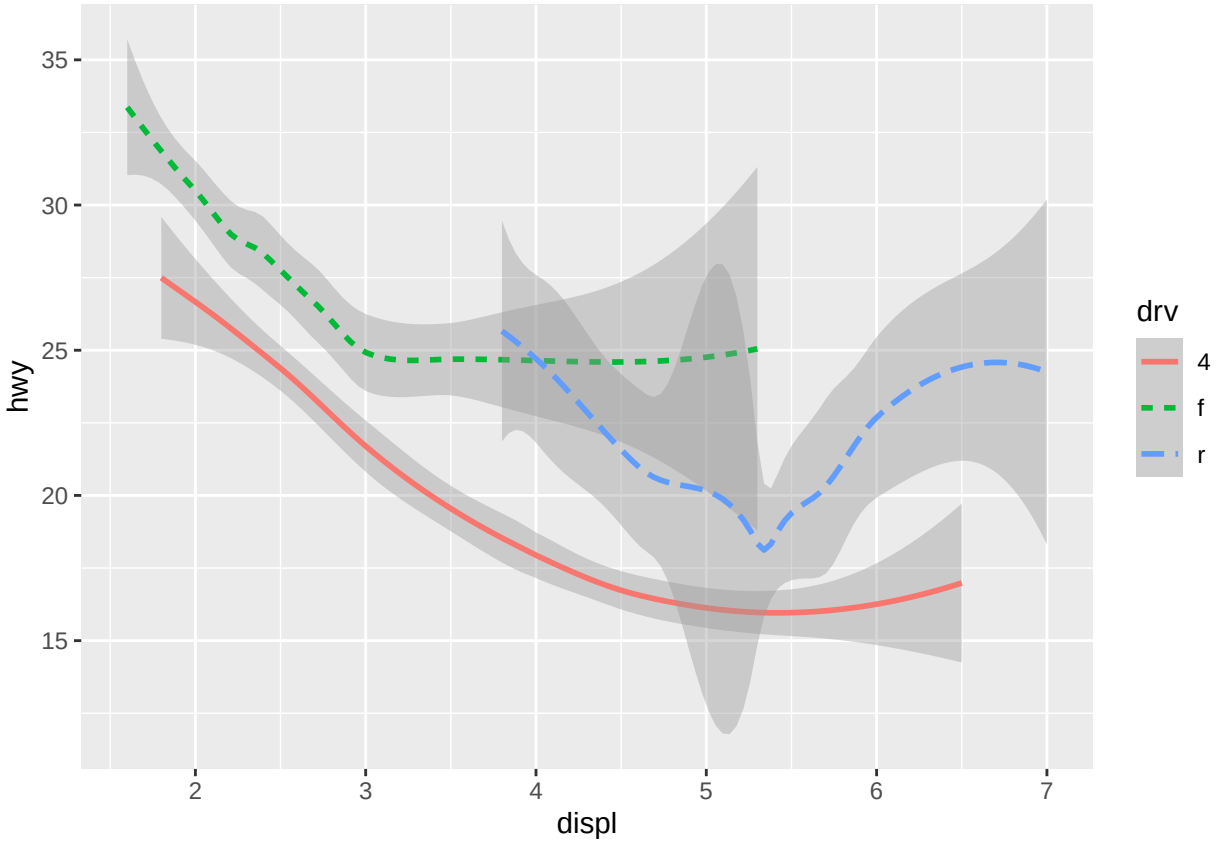
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Let's make this more dramatic by mapping the `drv` variable to both the `linetype` and the `color` aesthetics:

```
basic.mpg.object +
    geom_smooth(
        aes(
            x = displ,
            y = hwy,
            linetype = drv,
            color = drv
        )
    )
```

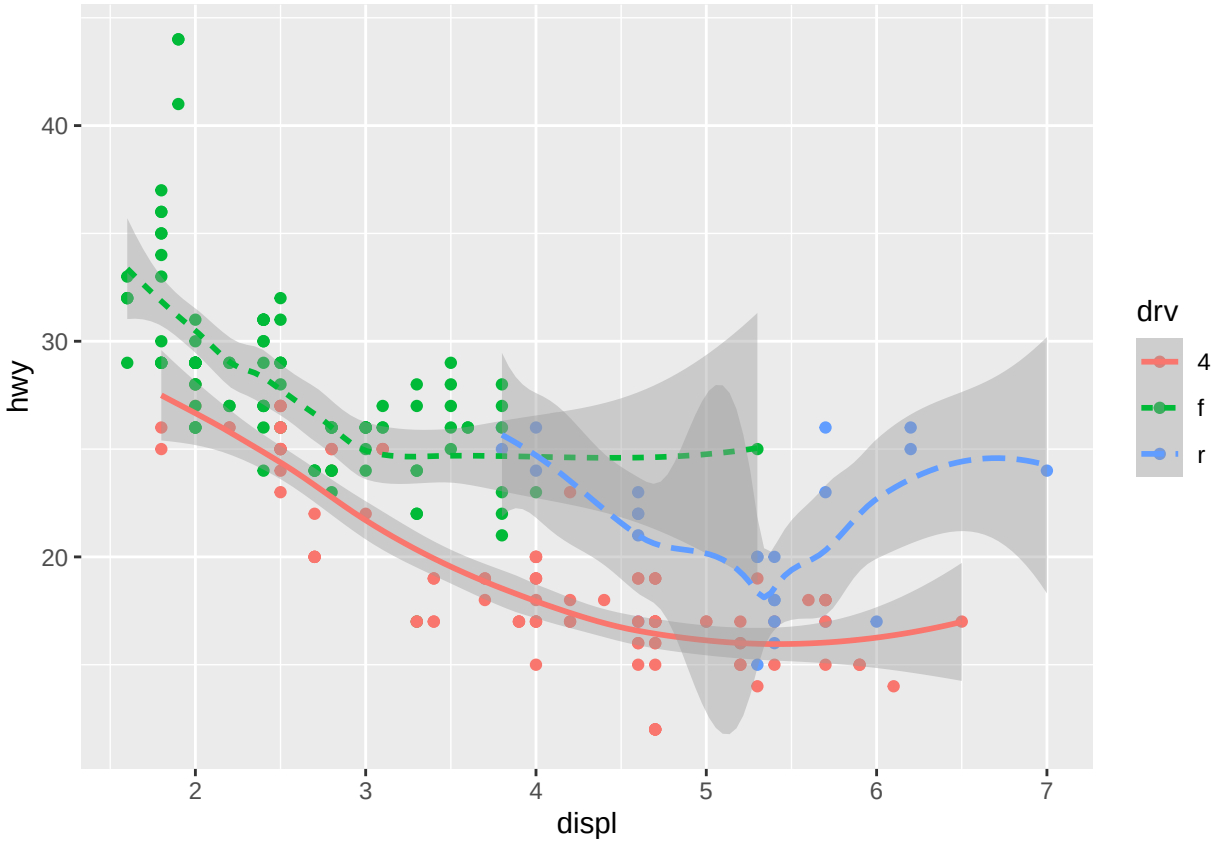
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Next, let's put all of this together by displaying the scatterplot and the smoothed curve together, stratifying by the `drv` variable:

```
basic.mpg.object +
    geom_point(
        mapping =
            aes(
                x = displ,
                y = hwy,
                color = drv
            )
    ) +
    geom_smooth(
        aes(
            x = displ,
            y = hwy,
            linetype = drv,
            color = drv
        )
    )
```

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



Far out!!

Finally, you might notice that there's a lot of redundancy to this graph, because we specified many of the same aesthetics for both the points and the smoothing curves.

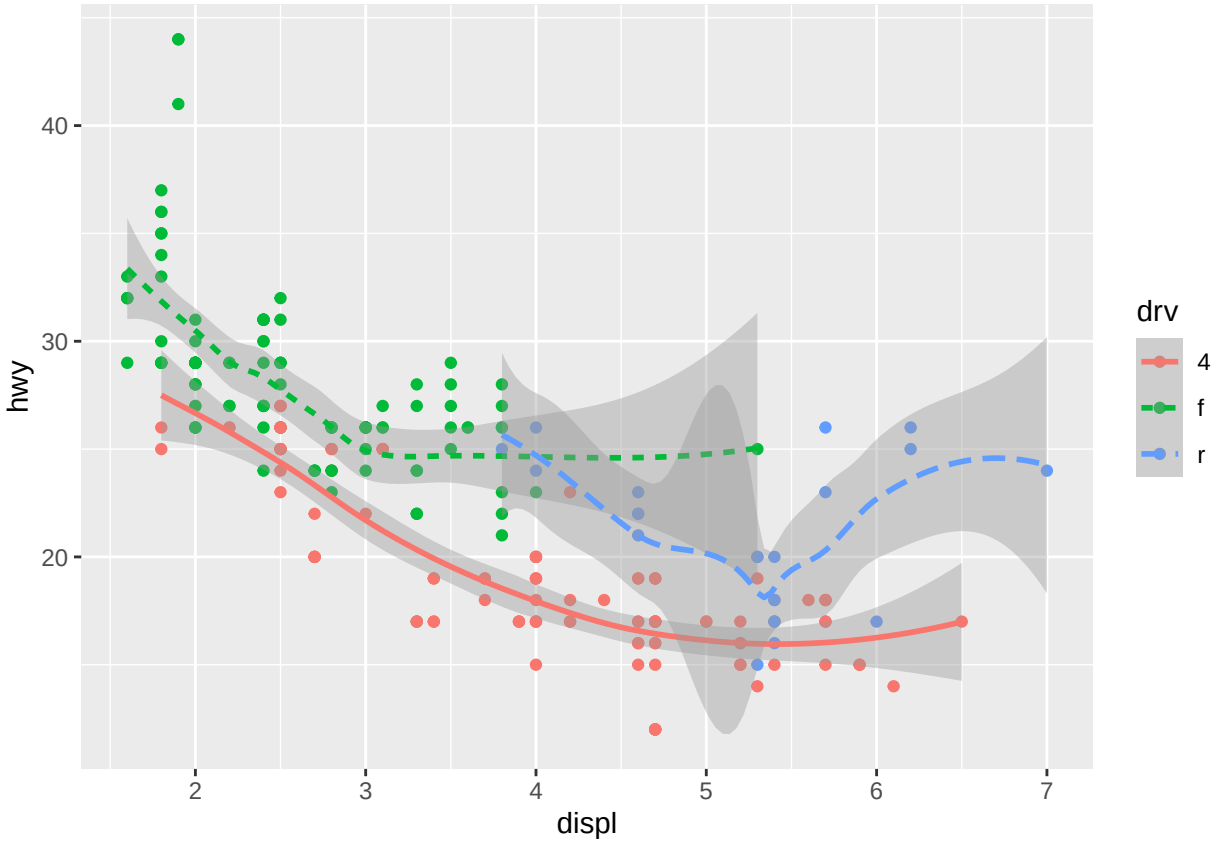
We can instead create a more complex `ggplot` object and include the aesthetics in this:

```
basic.mpg.object.2 <-
    ggplot(
        mpg,
        aes(
            x = displ,
            y = hwy,
            color = drv
        )
    )
```

Now we can make the same graph with less effort:

```
basic.mpg.object.2 +
    geom_point() +
    geom_smooth( aes( linetype = drv ) )
```

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



Section 4: Further Study

The goal in this lecture was to enable you to become familiar with the basic framework of `ggplot2`, especially the use of aesthetics and geometries.

The best way to learn more about `ggplot2` is to see lots of examples.

At the beginning of the course, I recommended the text:

- *R in 24 Hours*, by Andy Nicholls, Richard Pugh, and Aimee Gott (SAMS Publishing, ISBN: 978-0672338489).

This has a nice introductory treatment of `ggplot2`, focusing on `qplot()`.

Another recommended text for the course is:

- *R for Data Analysis in easy steps*, by Mike McGrath (In Easy Steps, ISBN: 978-1840787955).

The last 3 chapters of this text are devoted to `qplot()` and `ggplot2`.

You can purchase a full-color PDF of this book for just a few dollars from the publisher's website.

An excellent source of detailed information on `ggplot2` is:

- *R Graphics Cookbook*, by Winston Change (O'Reilly Media, ISBN: 978-1491978603)

This book is a little expensive, but there's nothing else like it for the sheer amount of practical code examples.

Another detailed treatment is:

- *Data Visualization: A Practical Introduction*, by Kieran Healy (Princeton University Press, ISBN: 978-0691181622)

This book is beautifully designed and produced, and is practically a coffee-table display book.

Of course, if you really want to get serious about `ggplot2`, then there is no substitute for the definitive reference written by the designer himself:

- *ggplot2: Elegant Graphics for Data Analysis*, by Hadley Wickham (Springer, ISBN: 978-3319242750)

Finally, there is lots of great material on the web, although it's usually just specific examples, and not a carefully constructed course.

Probably the best website for `ggplot2` examples is The R Graph Gallery:

https://www.r-graph-gallery.com/

Another great website is:

http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html

Good luck!