Week 12 Module 3: qplot()

CSCI E-5a: Programming in R

Module Overview

Let's get started with `ggplot2` by exploring a feature called "quickplots", implmented by the `qplot()` function.

The `qplot()` function provides a convenient interface to the `ggplot2` machinery, and enables us to create conventional graphs with a familiar interface.

Some `ggplot2` purists object to using `qplot()`, but I think it's a good place to get started with the package.

Section 1: Getting Started

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

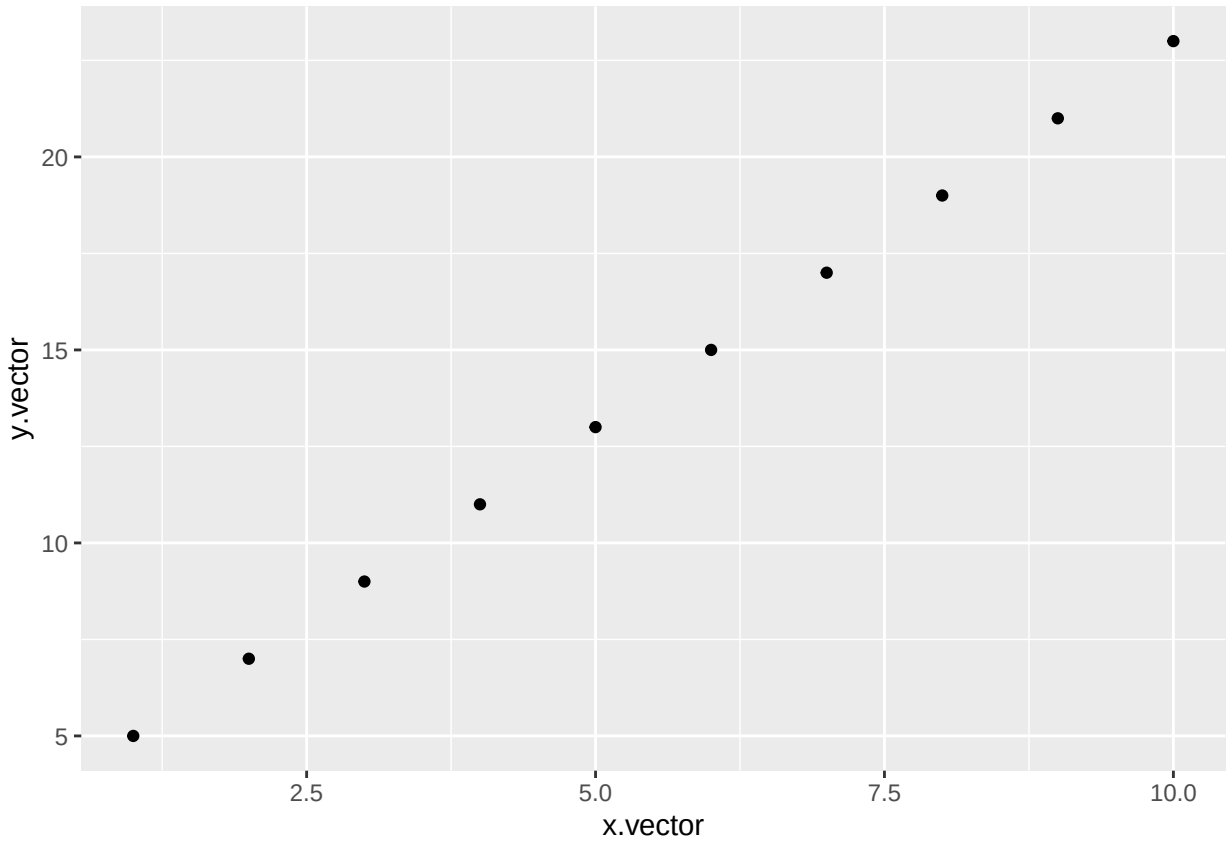
```
library( "ggplot2" )
```

Let's make a very simple plot: the x values will be the integers from 1 to 10, and the y values will be a simple linear function of these x values:

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

Now we'll make a scatterplot of `x.vector` and `y.vector`:

```
qplot( x.vector, y.vector, geom = "point" )
```



Let's try this again.

This time, we'll run `qplot()` to create a `ggplot2` object, but we'll store it in a variable:

```
basic.qplot.point.object <-
    qplot( x.vector, y.vector, geom = "point" )
```

Notice that now no graph is displayed.

The `qplot()` procedure doesn't actually draw anything, but rather creates an object of class `ggplot`:

```
class( basic.qplot.point.object )
```

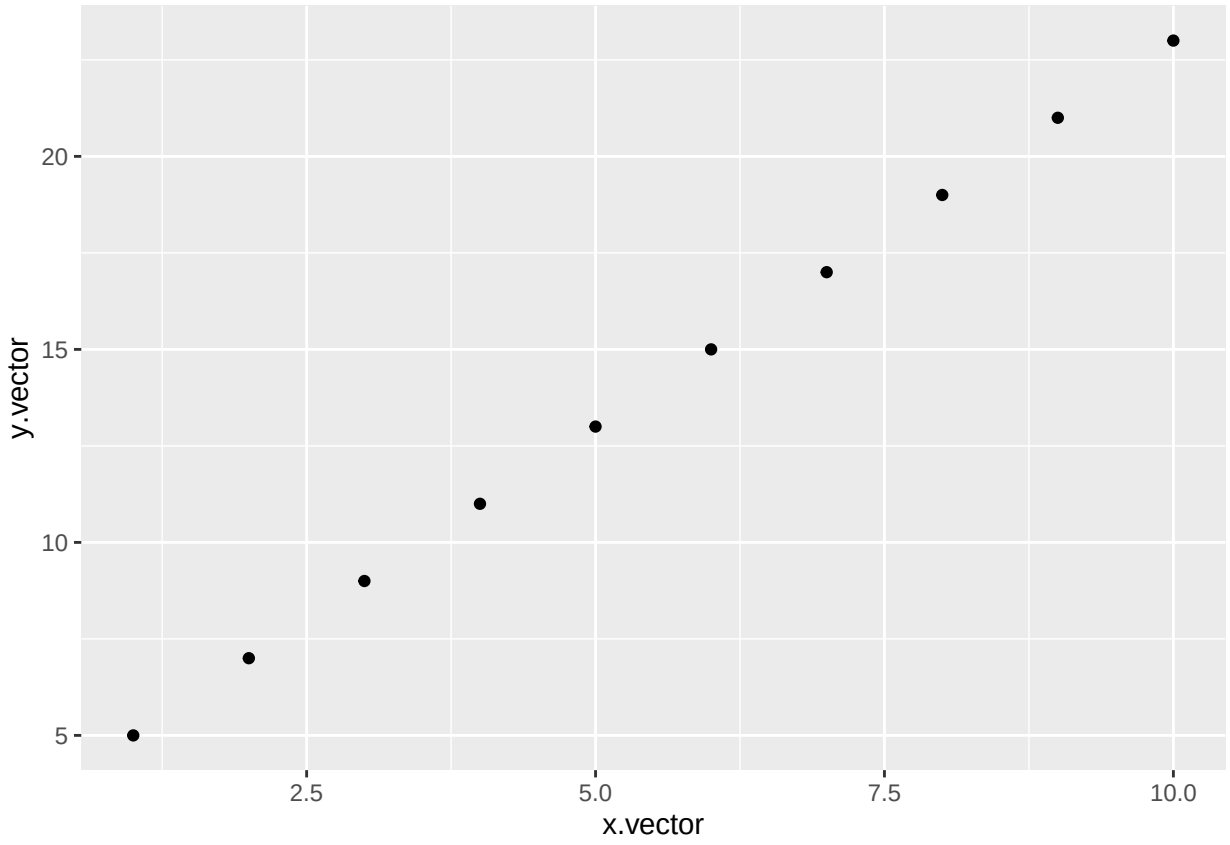
```
## [1] "gg"     "ggplot"
```

Let's take a look at the structure of this object:

```
str( basic.qplot.point.object )
```

This only draws a graph when R evaluates `basic.qplot.point.object`:

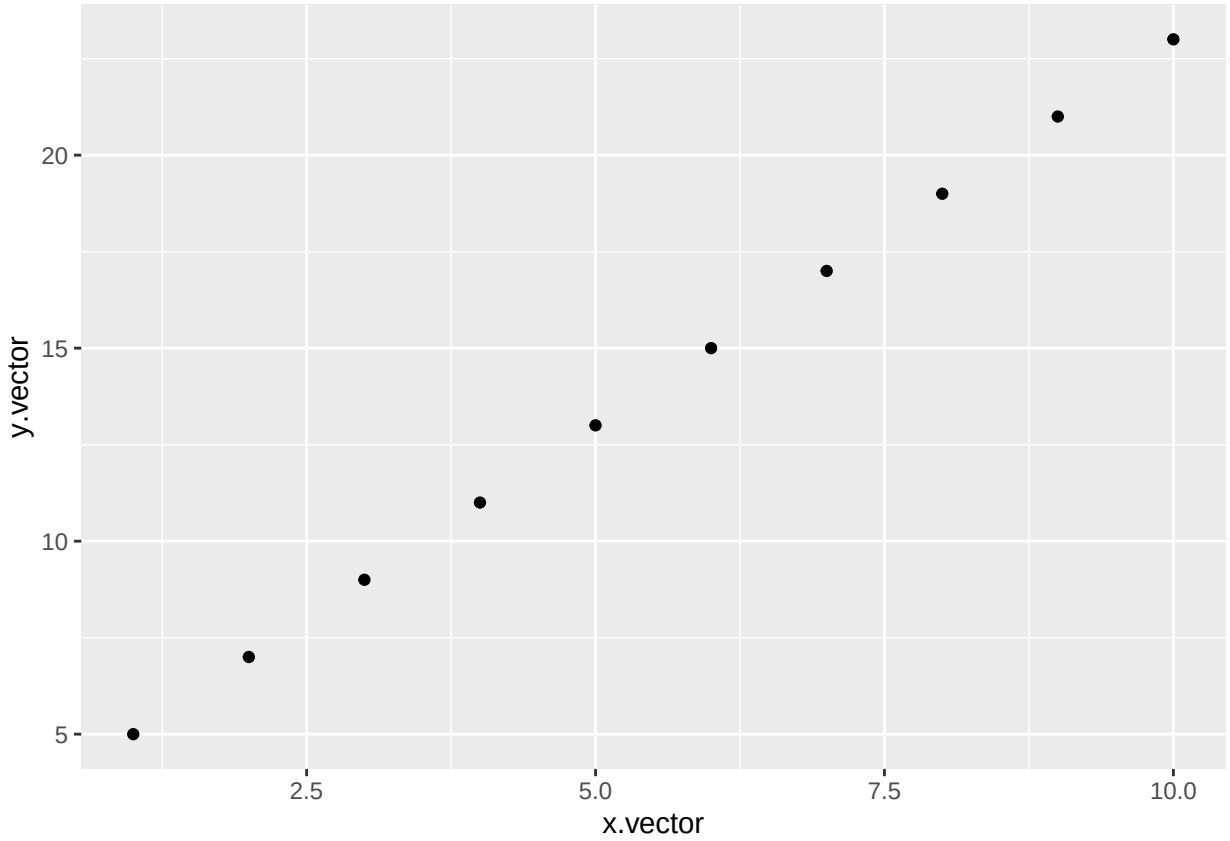
```
basic.qplot.point.object
```



Before we continue, let's check out the Data Visualization cheatsheet and find the `point` geom.

Actually, we didn't have to specify the `geom`, because `qplot` will use the `point` geom by default if there are two vectors of data:

```
qplot( x.vector, y.vector )
```



Section 2: Working with Data Frames

The `ggplot2` package makes it easy to work with data stored in data frames.

Let's make a simple data frame:

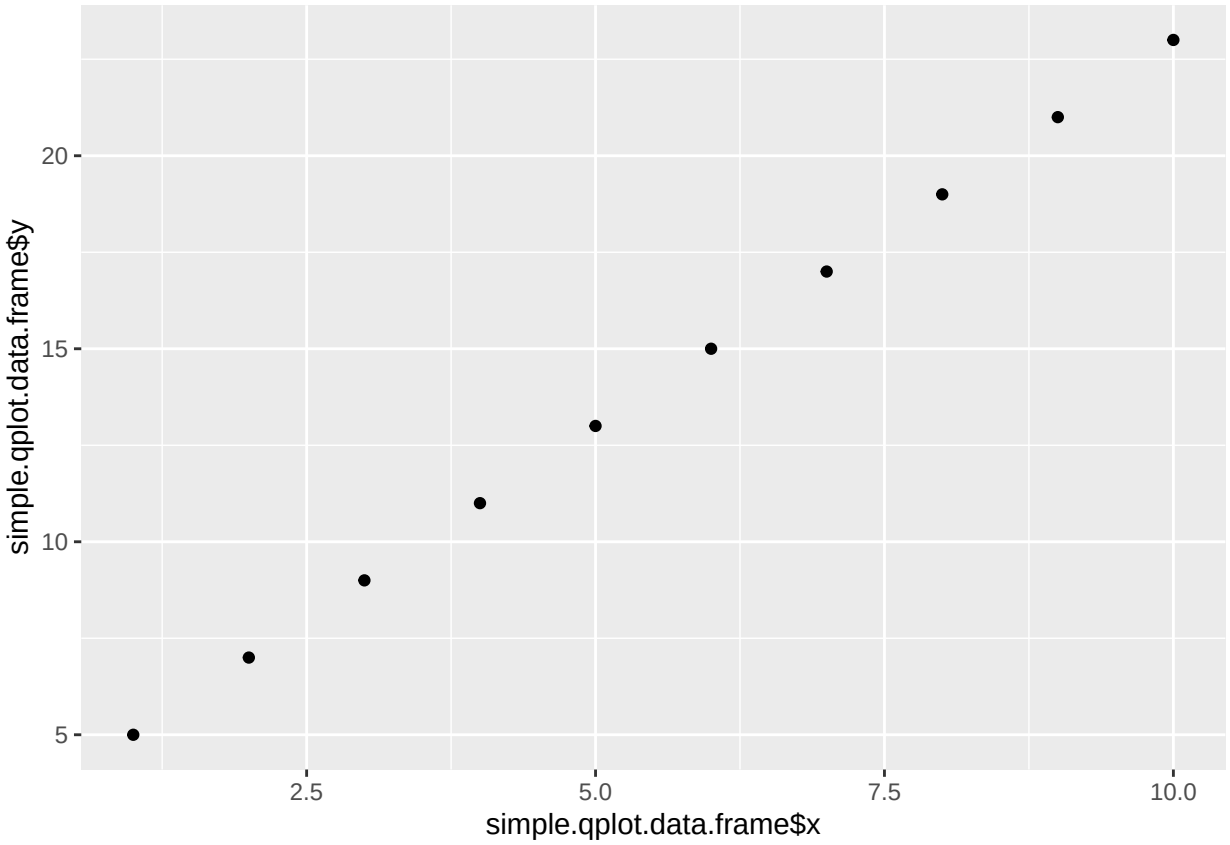
```
simple.qplot.data.frame <-
    data.frame(
        x = x.vector,
        y = y.vector
    )

head( simple.qplot.data.frame )
```

```
##   x  y
## 1 1  5
## 2 2  7
## 3 3  9
## 4 4 11
## 5 5 13
## 6 6 15
```

One way to plot the data in this data frame is to use the `$` operator to select the columns:

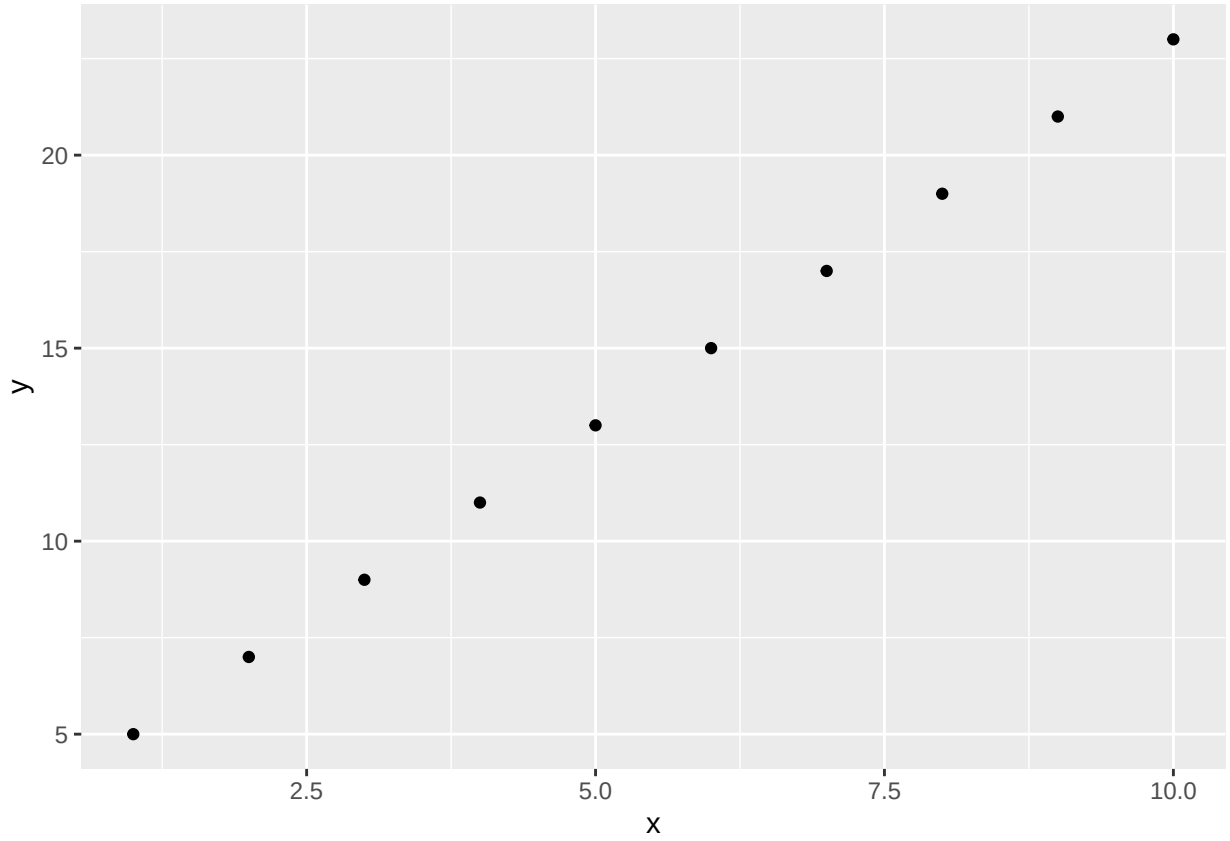
```
qplot(
    simple.qplot.data.frame$x,
    simple.qplot.data.frame$y,
    geom = "point"
)
```



That's perfectly fine, but the need to repeatedly specify the data frame when selecting the `x` and `y` columns clutters up the code.

In `ggplot2`, we can specify the data frame using the `data` argument:

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

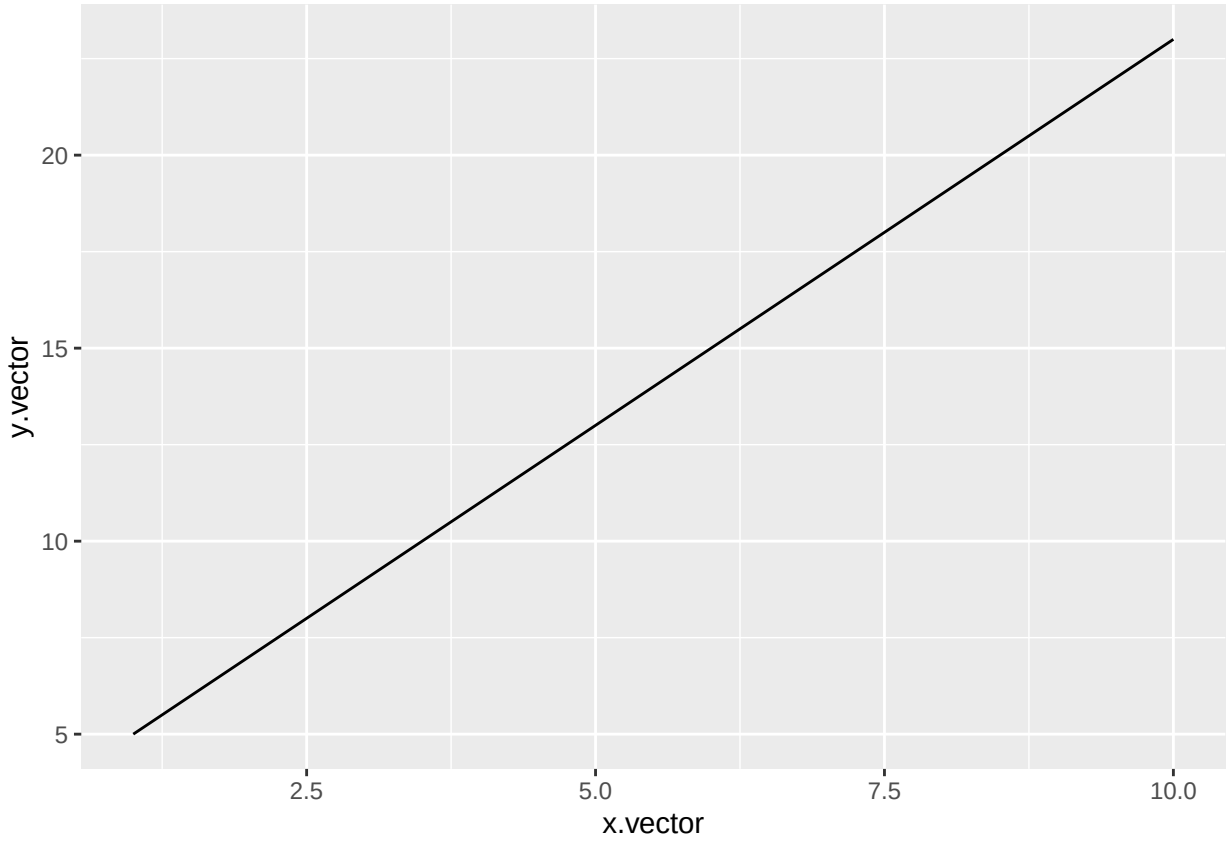


Section 3: Changing the `geom`

We can display the same data differently by changing the `geom` argument.

Before we specified that the display should simply be the points, but we can change it to displaying a line by setting the `geom` argument to "line":

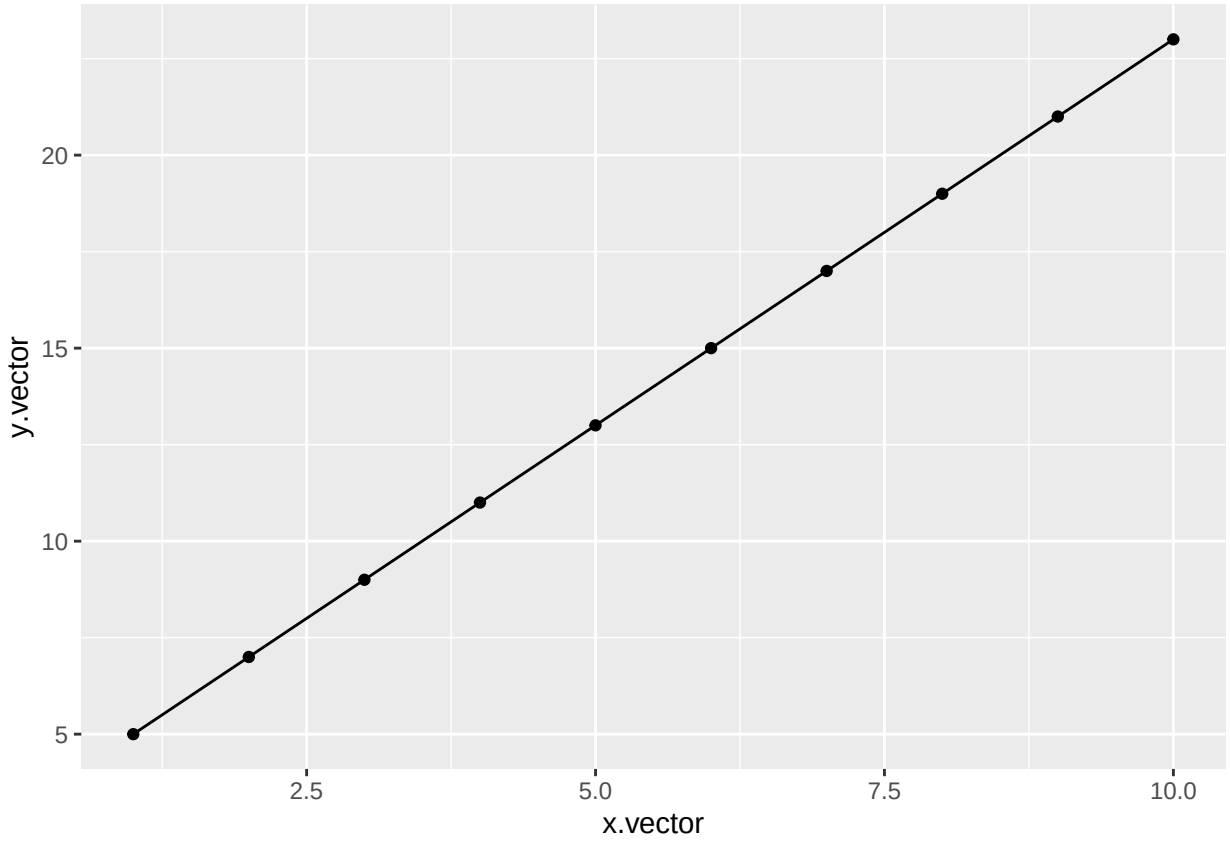
```
qplot( x.vector, y.vector, geom = "line" )
```



Before we move on, let's find the `line` geom on the Data Visualization cheatsheet.

We can display both points and lines together by assigning a vector to `geom`:

```
qplot( x.vector, y.vector, geom = c("point", "line") )
```

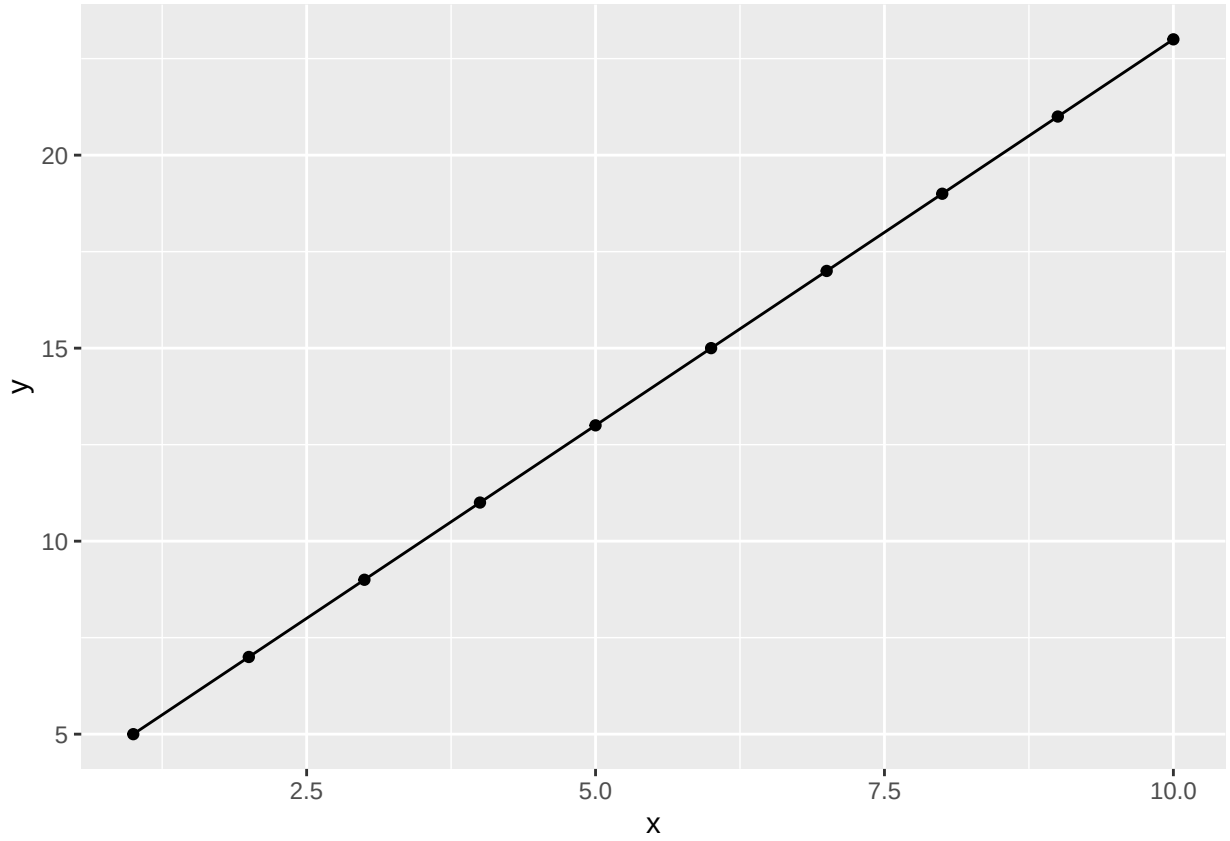


We can also accomplish this by adding the `geom` to a pre-existing `ggplot2` object:

```
basic.qplot.object <-
    qplot(
        x,
        y,
        data = simple.qplot.data.frame
    )

graphics.object.with.line <-
    basic.qplot.object + geom_line()
```

```
graphics.object.with.line
```



Section 4: Adding Color

It's nice to be able to control the colors in our plots.

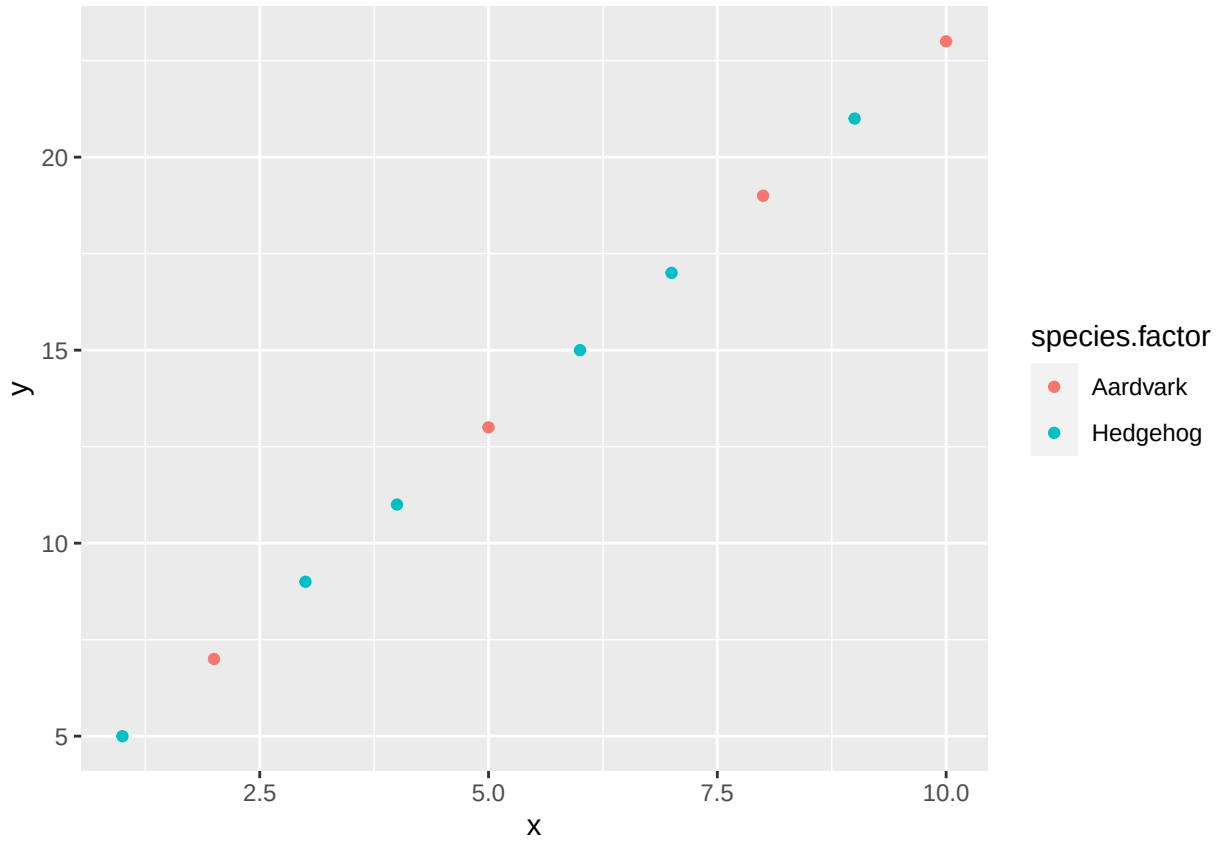
We'll learn something very profound about `ggplot2` when we try to do this!

First, let's create a factor variable:

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

Now we'll assign the factor variable to the `col` argument.

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

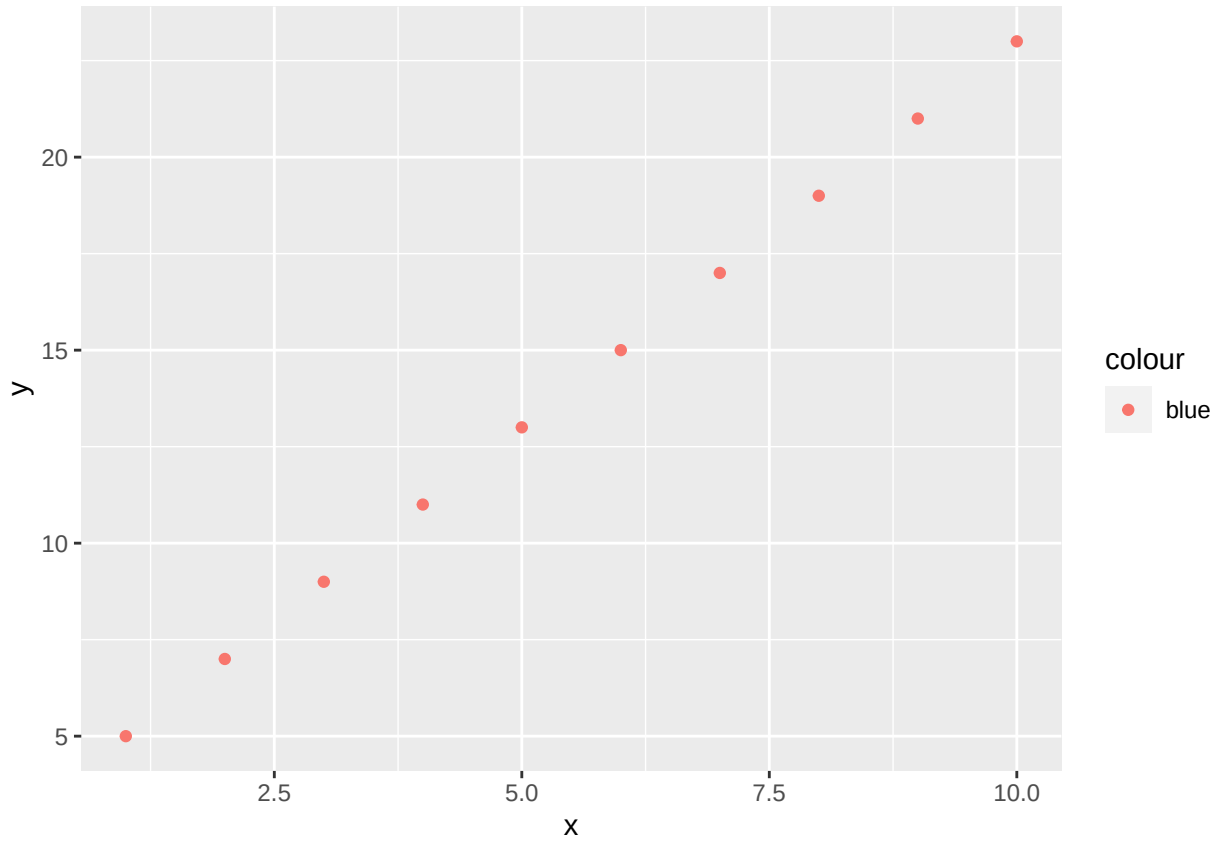


Notice what happened here: the factor variable was mapped to the aesthetic `col`, and so the colors of the points represent the different levels of the factor.

What happens if we just want all the points to be blue?

Let's try to assign the value "blue" to the `col` argument:

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



Whoops!

That was unexpected.

`qplot()` didn't make all the points blue – in fact, they're all *red*.

And it also created a weird legend entry on the right-hand side.

What happened?

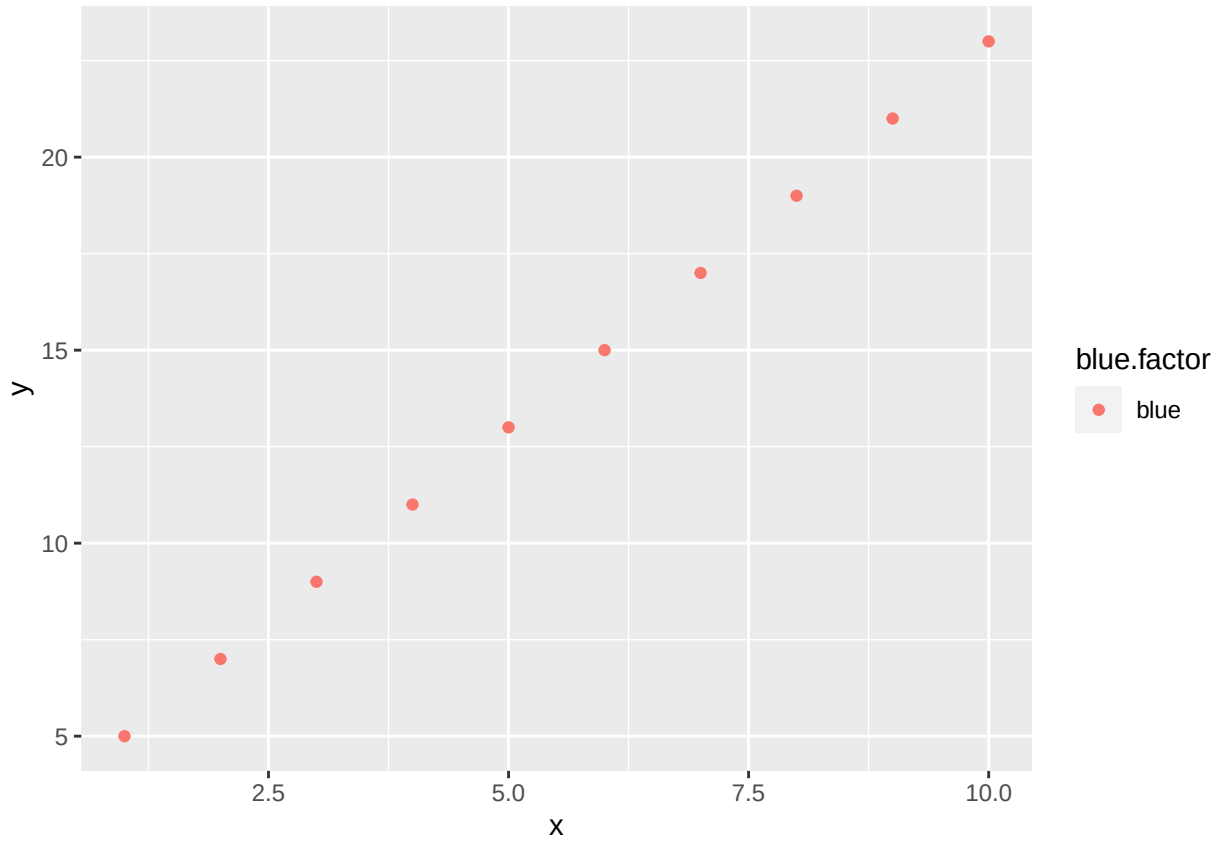
When we assigned the value "blue" to the `col` argument, `ggplot2` treats this as an aesthetic mapping, just like it did with the `species.factor` variable.

Since we used only a single value "blue", `ggplot2` will convert this to a factor variable with only one level, and every element in the dataset has this value.

Essentially, we did this:

```
blue.factor <-
    rep( "blue", 10 )

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



But why are the points red – didn't we ask for blue?

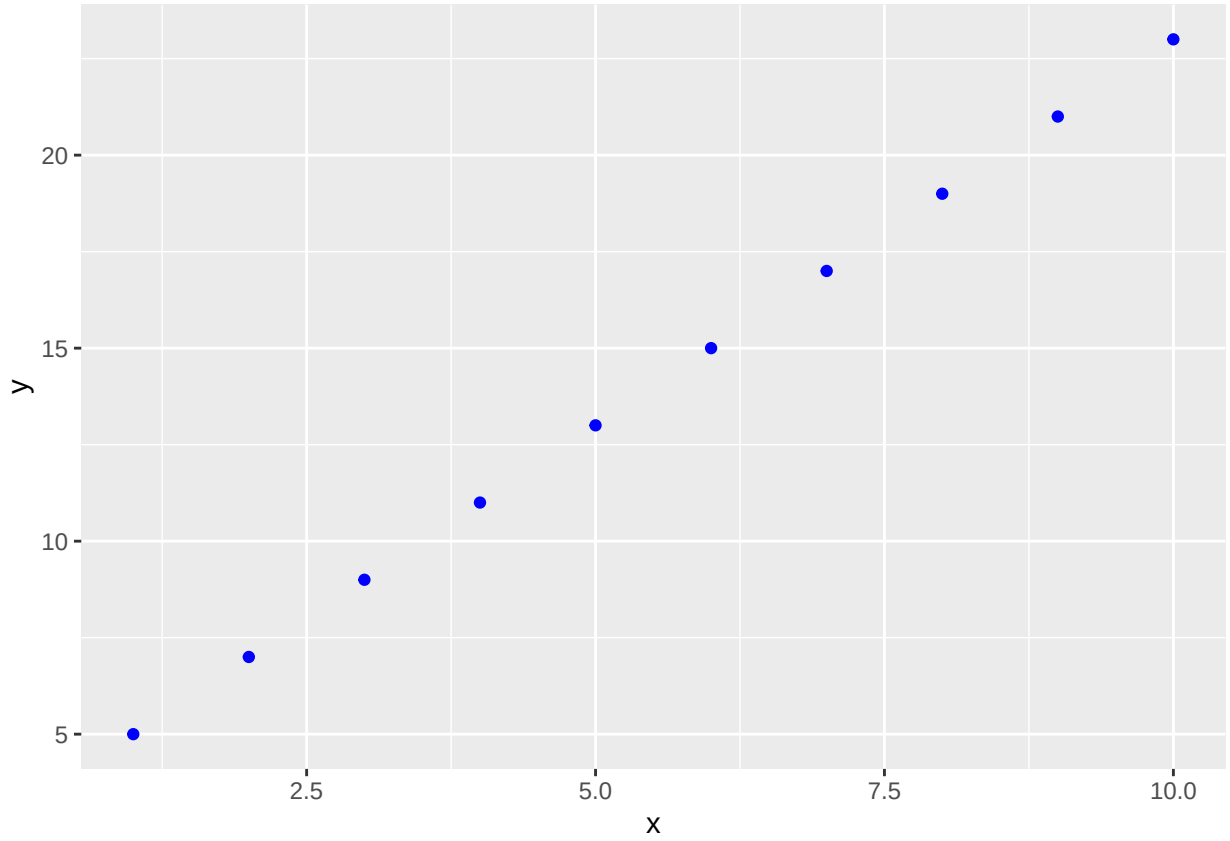
The answer here is that red is the first color in the default `ggplot2` color palette.

In other words, it was totally irrelevant that the thing that was assigned to `col` was in fact a valid R color name, because `ggplot2` just treated it as a factor level.

After all this, how do we change the color of the points to blue?

We assign the value "blue" to `col`, but we have to use the `I()` function to inhibit evaluation:

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

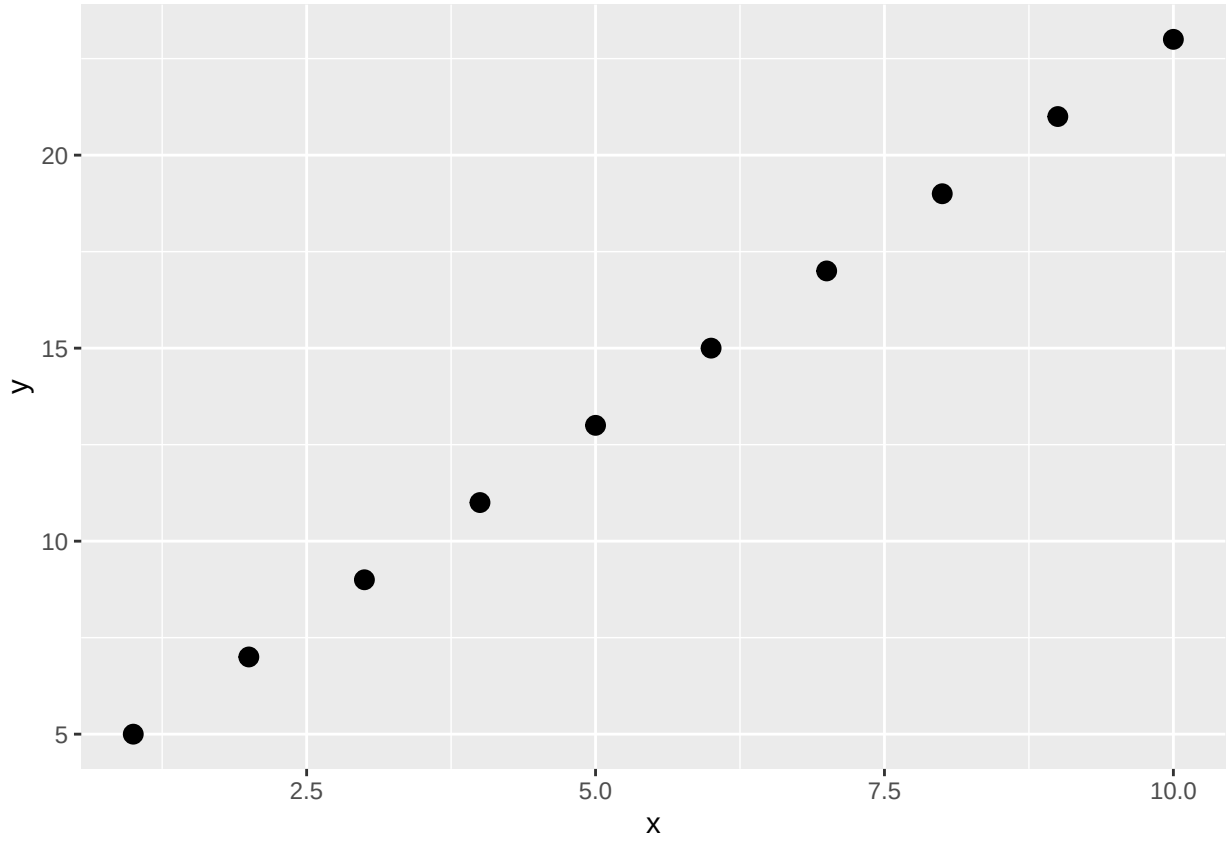


Now you might be wondering: why did I have to wrap "blue" with an `I()` function, but not "point"?

That's because the "point" value is specifying a **`geom`** method, and this isn't creating some sort of mapping of data to a visual component – it's just specifying the particular method for displaying the data.

You can also change the size of the graphics elements using **`size`** argument, but once again you have to inhibit evaluation by using the `I()` function:

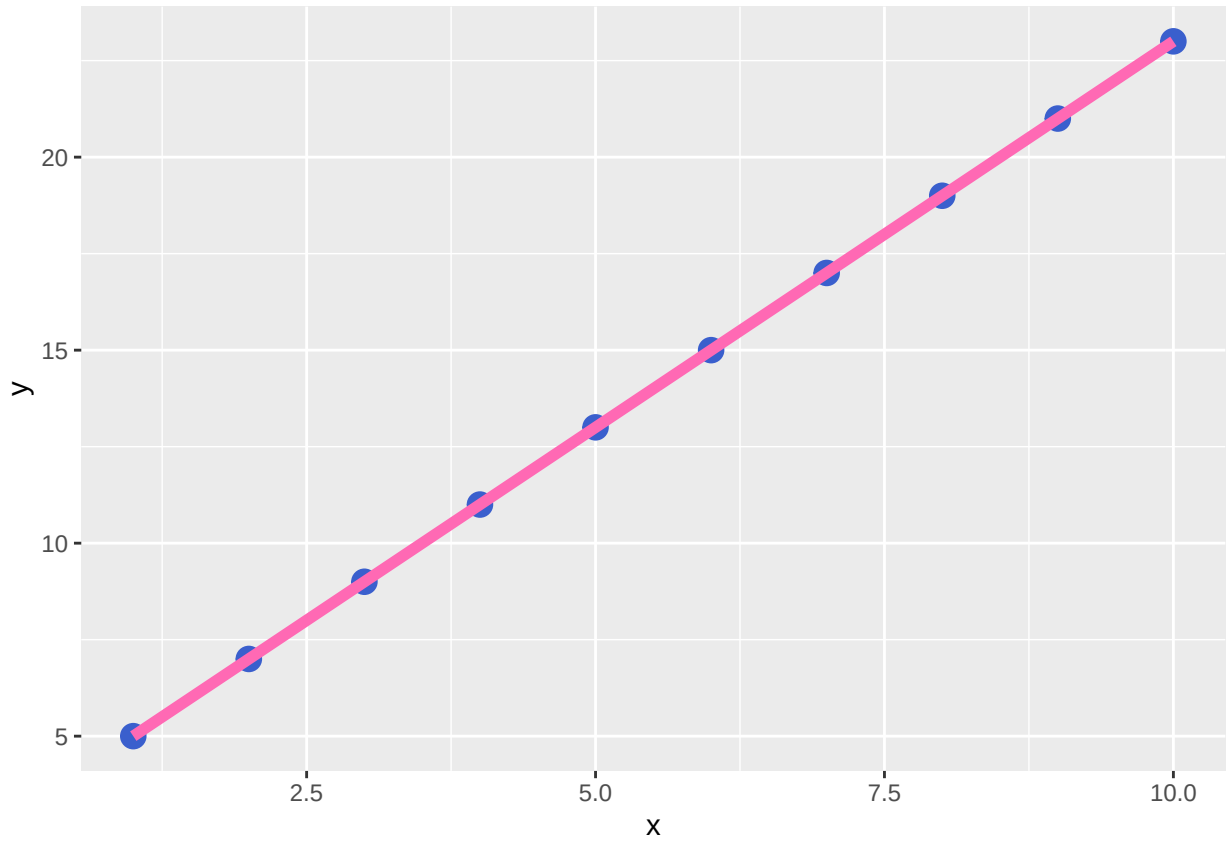
```
qplot(
        x,
        y,
        geom = "point",
        data = simple.qplot.data.frame,
        size = I( 3 )
    )
```



Here's how to change the size of the points and the lines:

```
qplot(
        x,
        y,
        geom = "point",
        data = simple.qplot.data.frame,
        size = I( 4 ),
        col = I( "royalblue3" )
    ) +

geom_line( size = I(2), color = I("hotpink" ))
```



You can clearly see that the points are being drawn first, and then the line.

These examples with adjusting color illustrate my point from the previous module – there's definitely a learning curve for `ggplot2`, and the graphics paradigm for the package is very different from pretty much any other graphics package.

That doesn't mean that it's impossible to learn, or that you shouldn't learn it.

It *does* mean that it requires an investment of time to achieve competence in it, and each person will have to make an individual decision as to whether or not that investment is worth it.

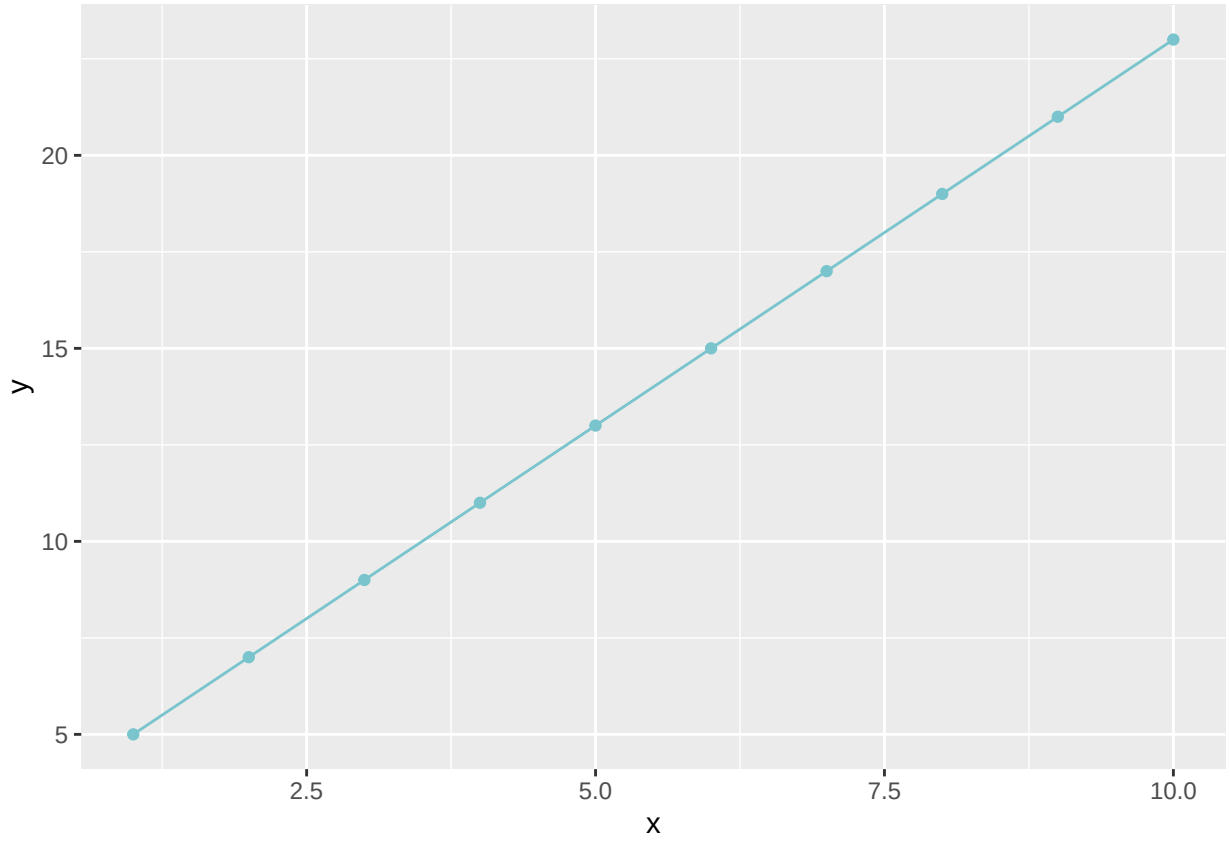
Section 5: Adjusting the Theme

You might have noticed that all of these graphs look different from the previous graphs we created using the base R graphics tools.

We can change the overall look and feel of this by using a different "theme", and we change the them by literally adding it to the graphics object created by `qplot()`:

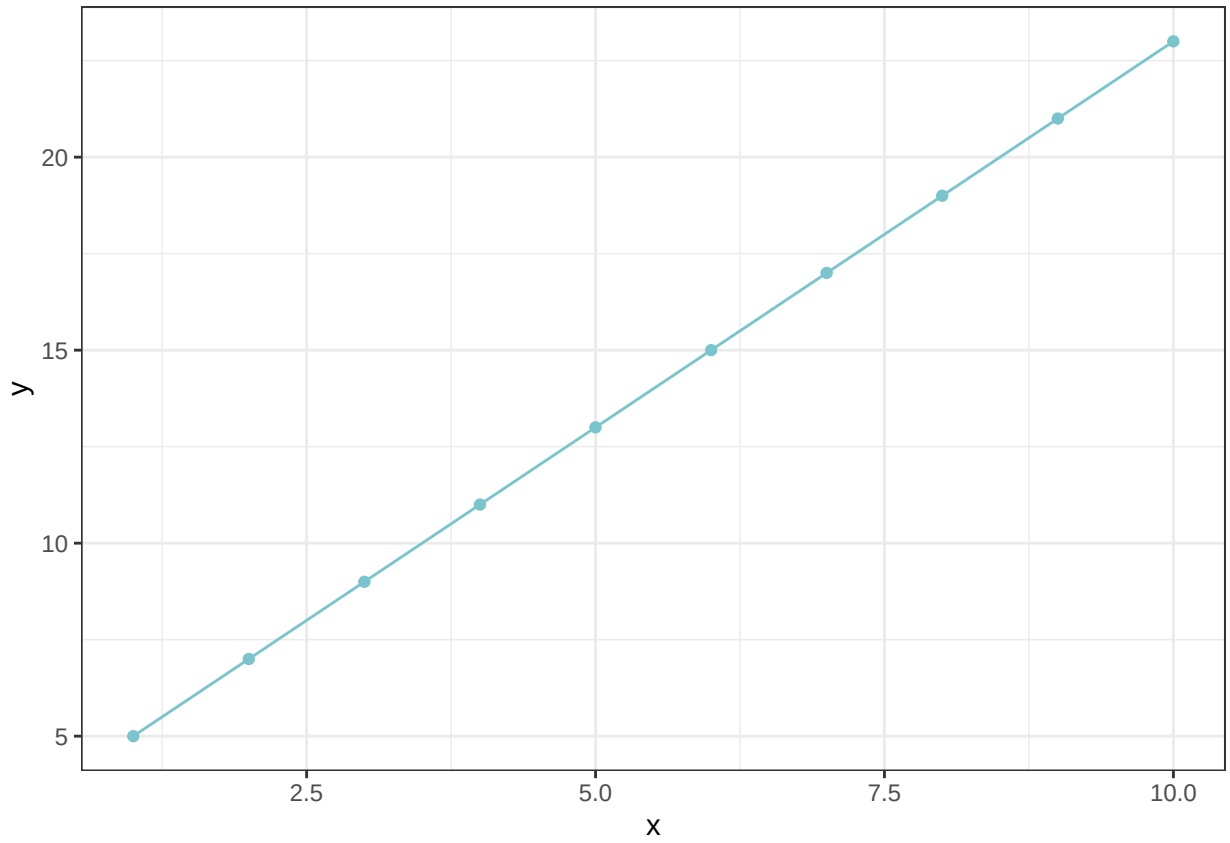
```
basic.qplot.object <-
    qplot(
        x,
        y,
        geom = c("point", "line"),
        data = simple.qplot.data.frame,
        col = I( "cadetblue3" )
    )
```

```
basic.qplot.object
```



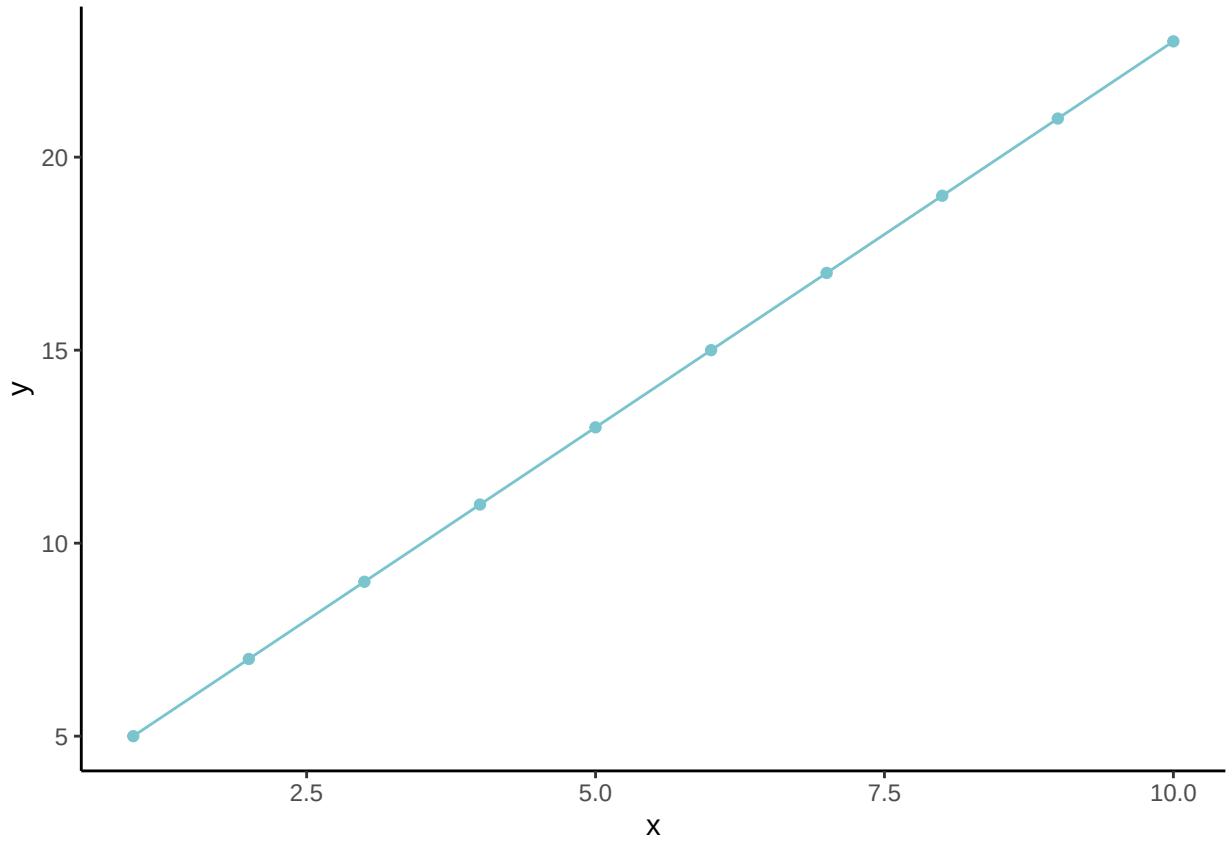

```
new.theme.qplot.object <-
    basic.qplot.object + theme_bw()

new.theme.qplot.object
```



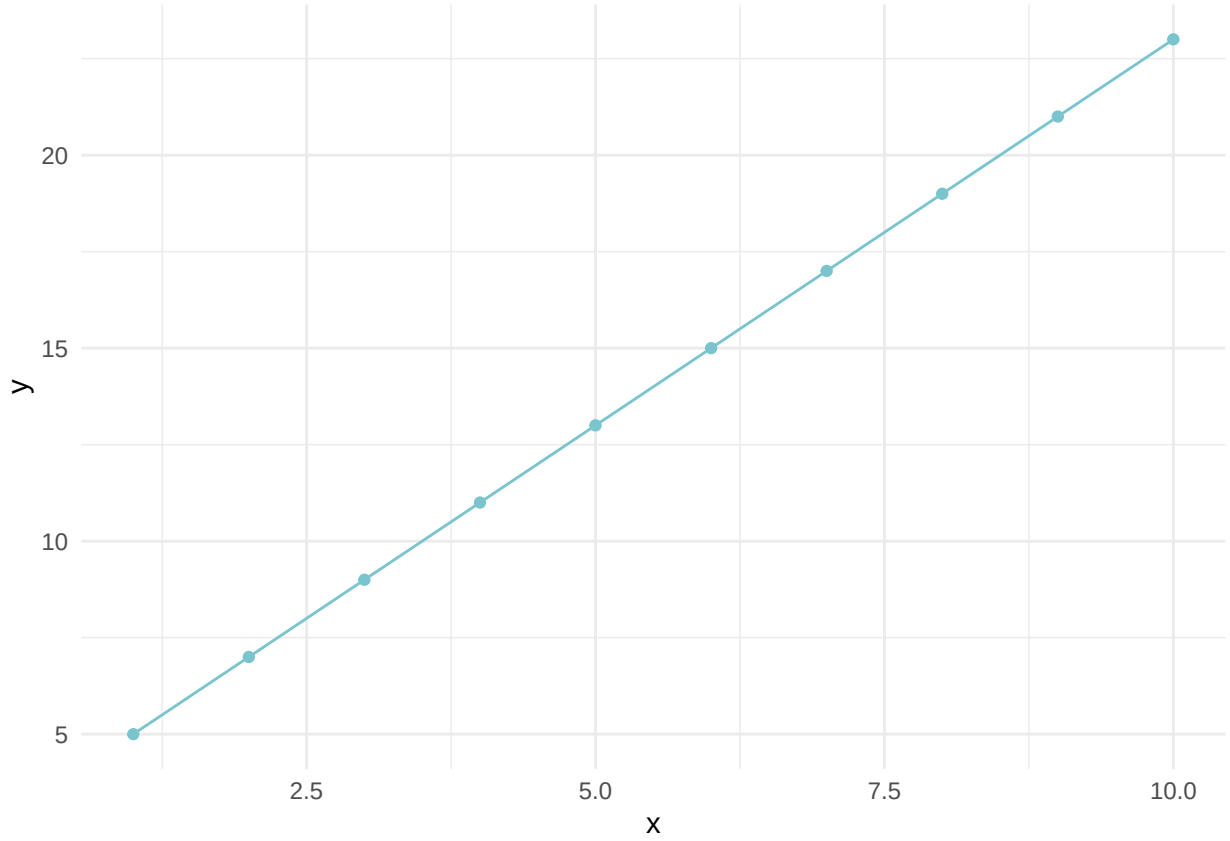
Here's the same plot using the `classic` theme:

```
basic.qplot.object + theme_classic()
```



Here's the same plot using the `minimal` theme:

```
basic.qplot.object + theme_minimal()
```



Personally, I find the default gray theme annoying, but of course that's purely a subjective preference.

We can globally reset the theme, so that now all the graphs will use the `bw` theme:

```
theme_set( theme_bw( base_size = 14 ) )
```

Section 6: More Graph Types

So far we've only made scatterplots.

In this section we'll make some other type of graphs.

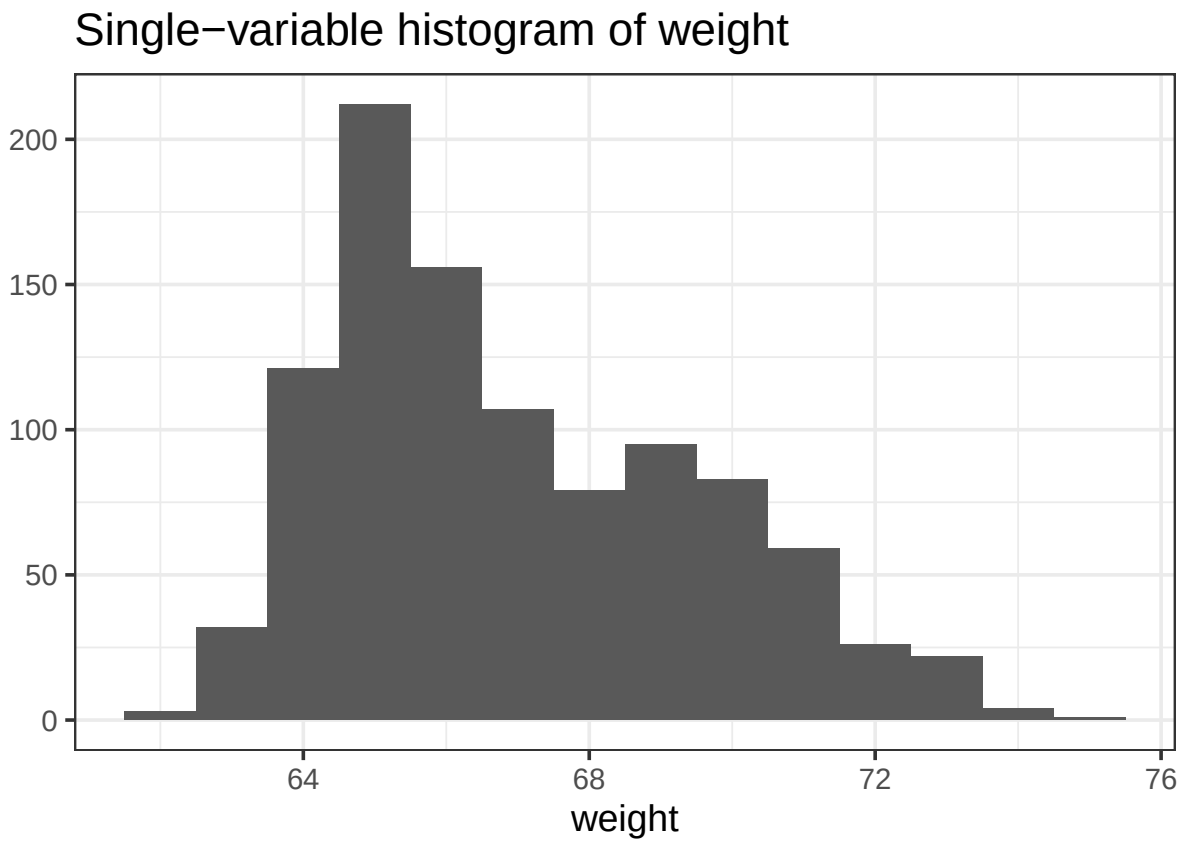
Let's start by generating some random data to play with:

```
set.seed( 1 )

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

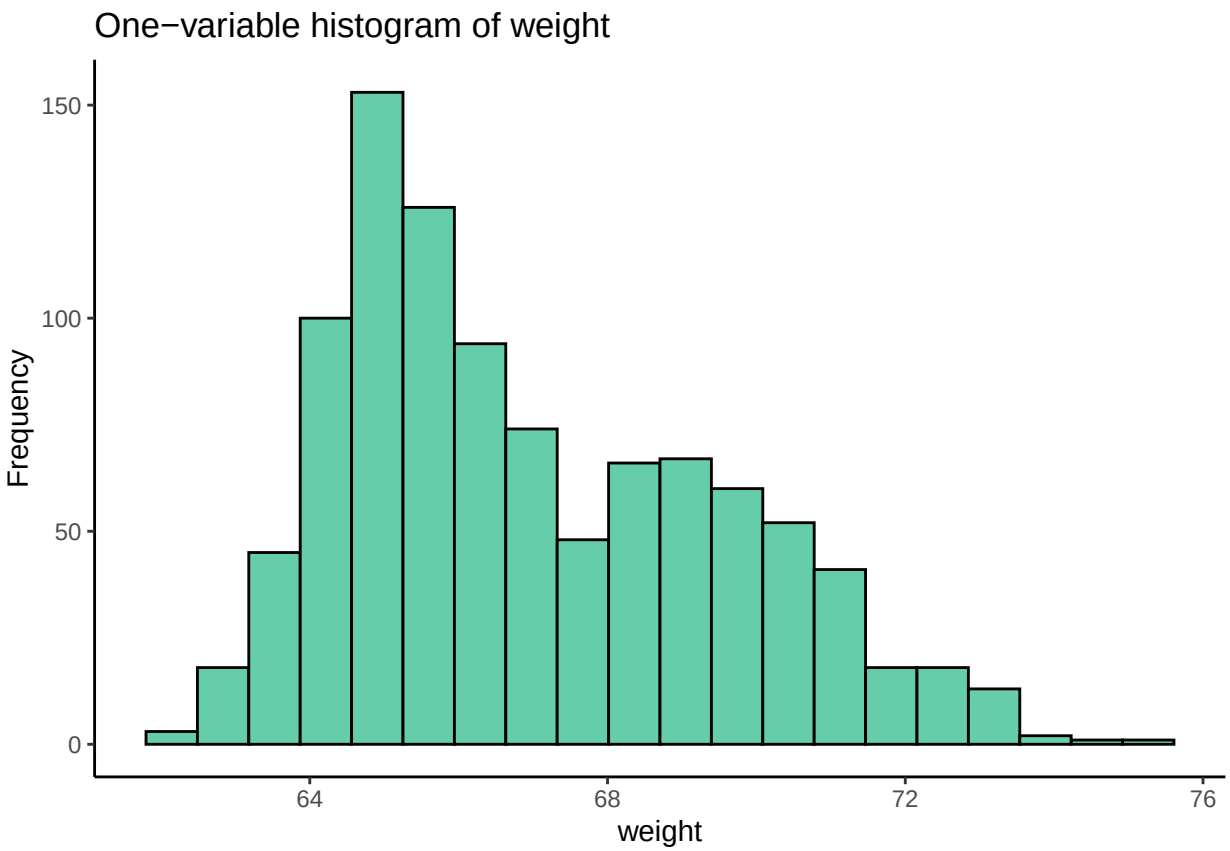
If we only pass one vector of data to `qplot()`, by default it will create a histogram:

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



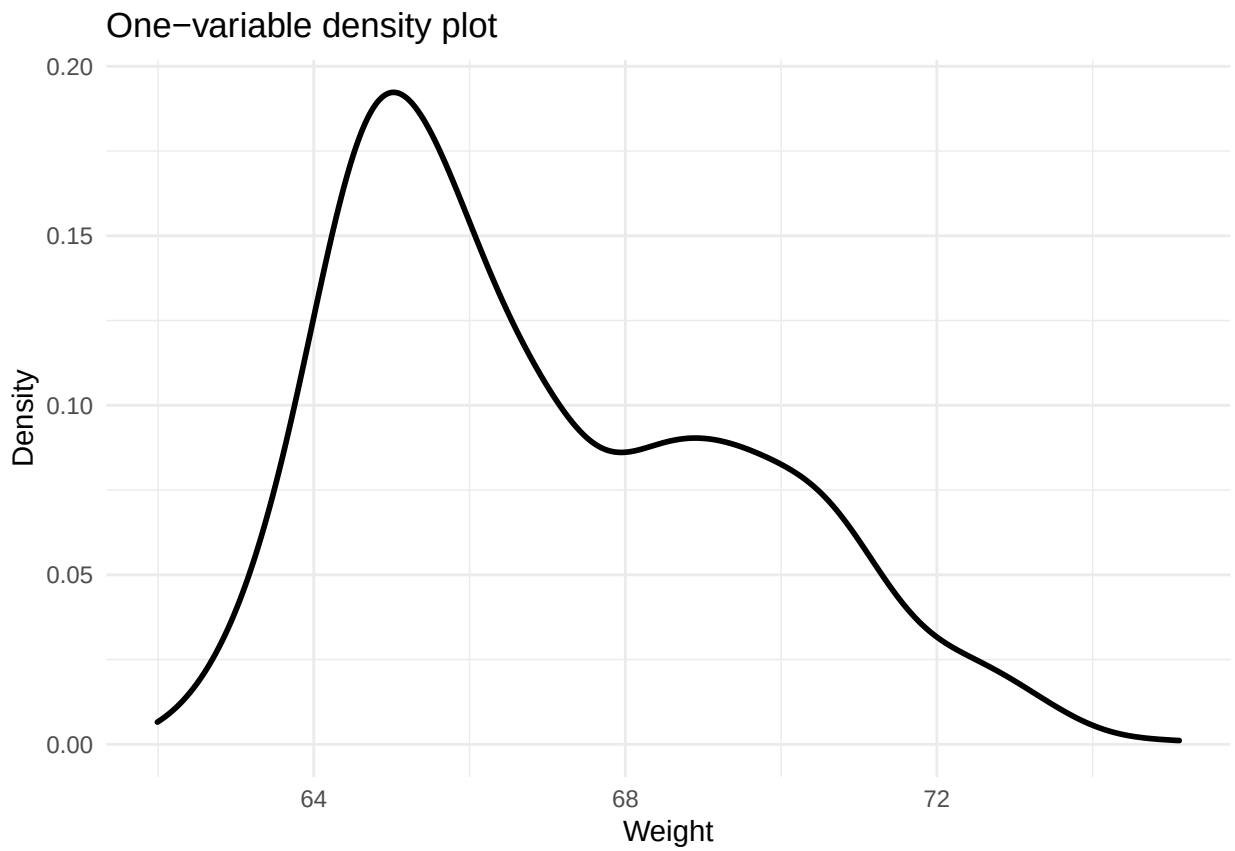
We can adjust the appearance of the graph by using options:

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



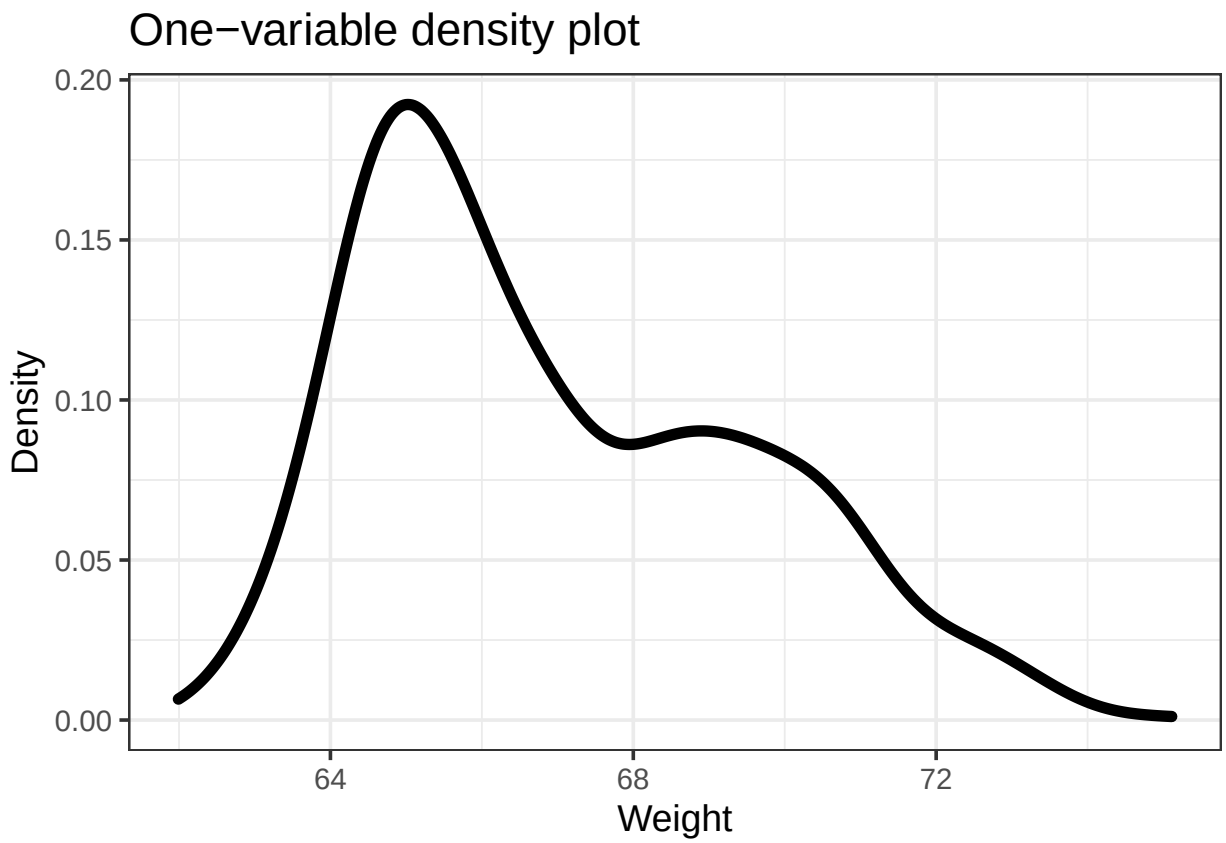
Density plot

```
qplot( weight,
       geom = "density",
       main = "One-variable density plot",
       size = I( 1 ),
       ylab = "Density",
       xlab = "Weight" ) +
   theme_minimal()
```



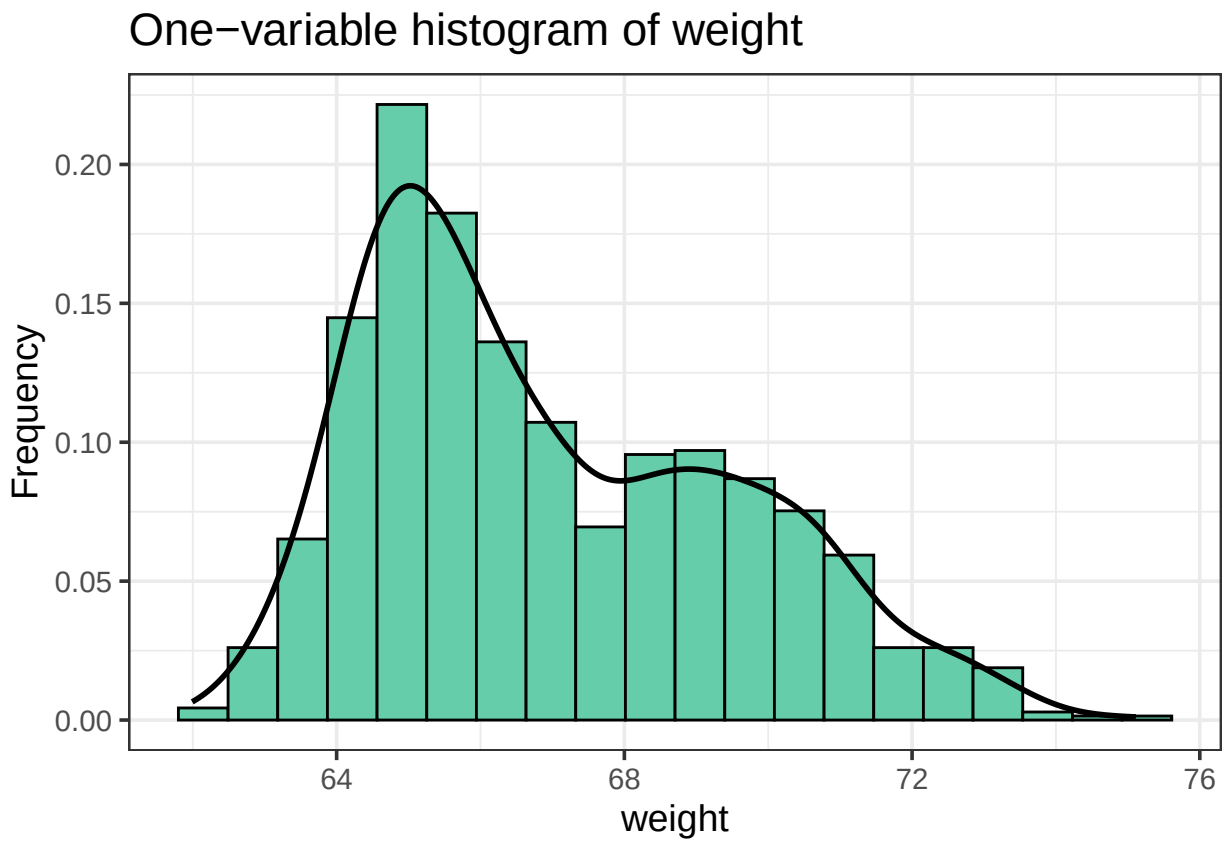
Let's make the density line thicker:

```
qplot( weight,
       geom = "density",
       main = "One-variable density plot",
       size = I( 2 ),
       ylab = "Density",
       xlab = "Weight" )
```



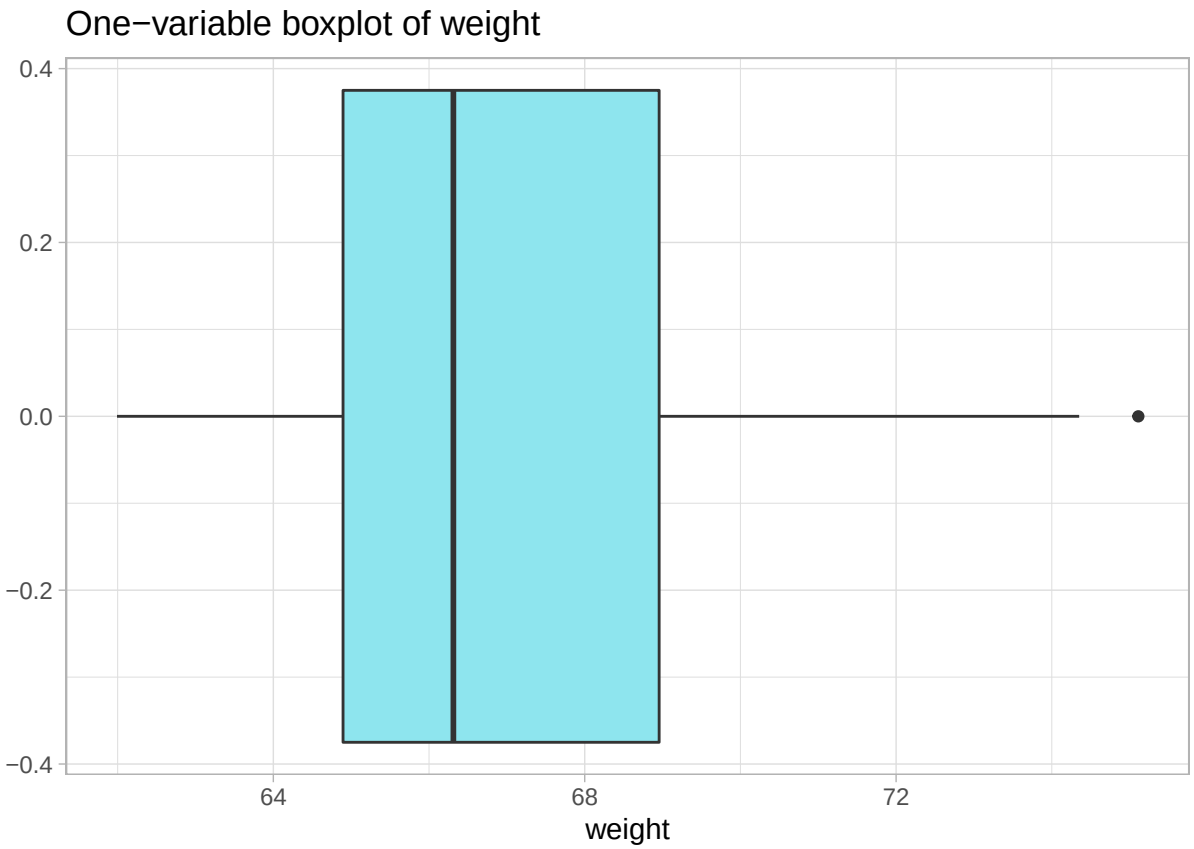
Histogram with superimposed density curve

```
qplot( weight,
       geom = "histogram",
       y = ..density..,
       main = "One-variable histogram of weight",
       ylab = "Frequency",
       fill = I( "aquamarine3" ),
       color = I( "black"),
       bins = 20 ) +
   geom_line( size = I( 1 ), stat = "density" )
```



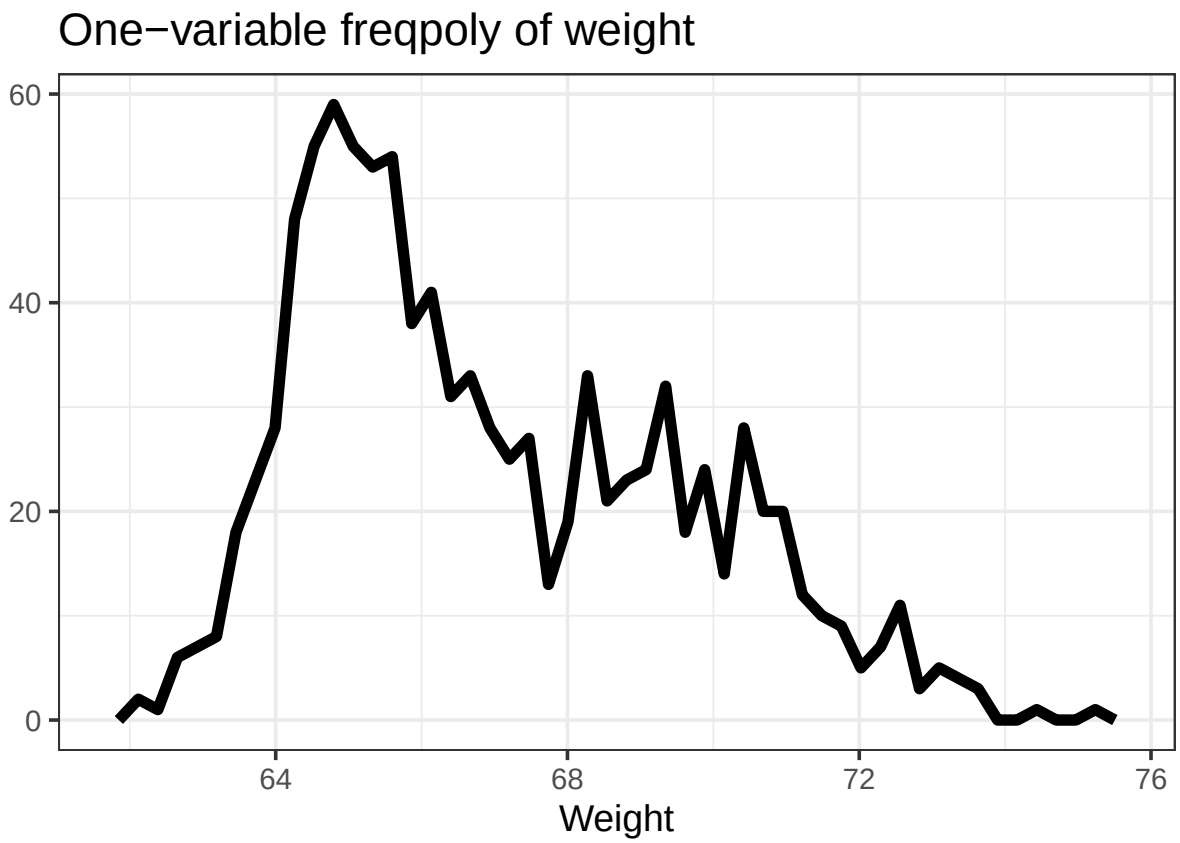
Boxplots

```
qplot( NULL,
       weight,
       geom = "boxplot",
       xlab = "",
       fill = I( "cadetblue2" ),
       main = "One-variable boxplot of weight" ) +
   coord_flip() +
   theme_light()
```



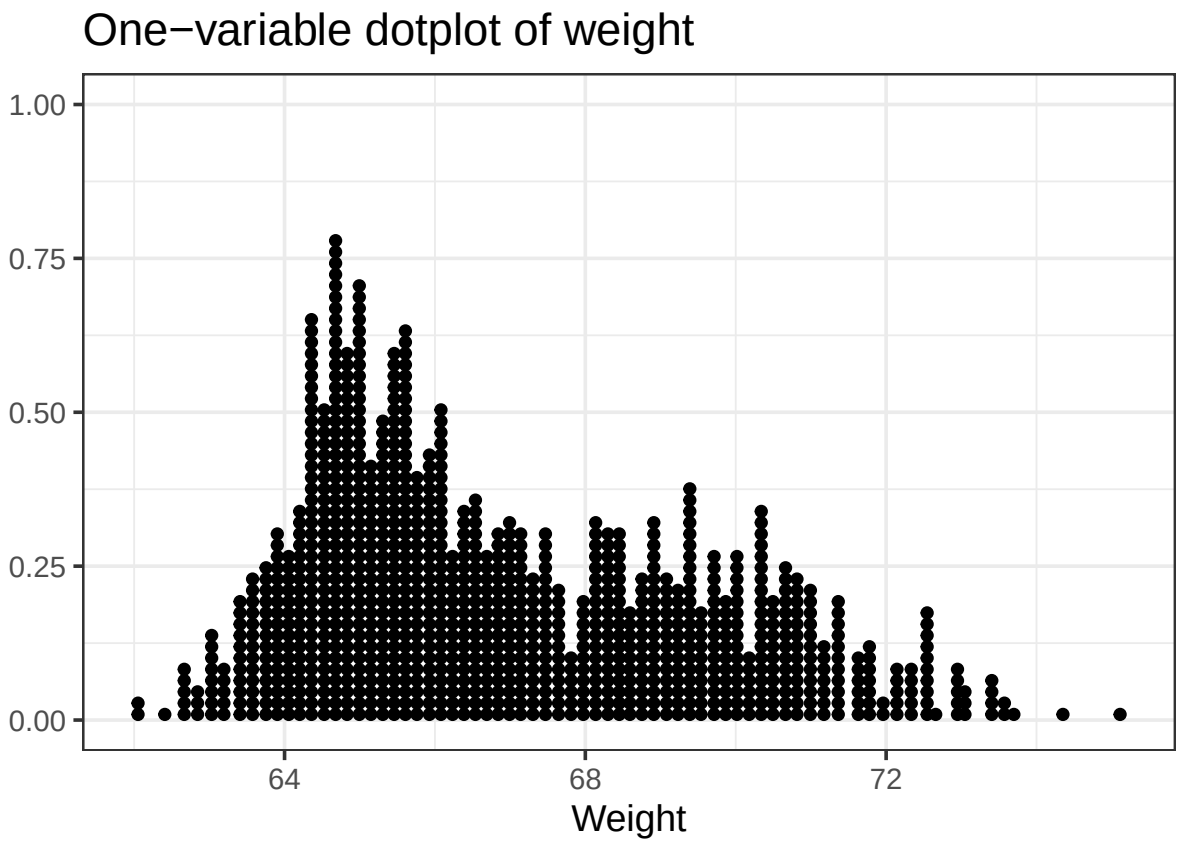
Freqpolys

```
qplot( weight,
       geom = "freqpoly",
       xlab = "Weight",
       bins = 50,
       size = I( 2 ),
       main = "One-variable freqpoly of weight" )
```



Dotplots

```
qplot( weight,
       xlab = "Weight",
       geom = "dotplot",
       binwidth = 0.15,
       main = "One-variable dotplot of weight"
)
```


One−variable dotplot of weight
1.00
0.75
0.50
0.25
0.00
64
68
72
Weight