Advanced Graphics with the package ggplot2

J. Henningsen, T. Eiting, C.-Y. Kuo, M. Rosario

10.12.2012

1 Introduction

1.1 Welcome

As scientists, we know the importance of displaying data in an easy-to-read, attractive format. Furthermore, good figures can help us explore data and examine them from a unique perspective. R contains built in plot functions, but they can be limited in scope. The package ggplot2 contains alternative methods to quickly build attractive plots, with further options for extensive customization.

1.2 Getting started

Let us begin by installing the ggplot2 package. As always, make sure you have the latest version of R. Once installed, we load the package

```
# install and load the package ggplot2
install.packages("ggplot2")

## Error:  trying to use CRAN without setting a mirror

library(ggplot2)
# set work directory
setwd("User/Apple/Desktop")

## Error:  cannot change working directory
```

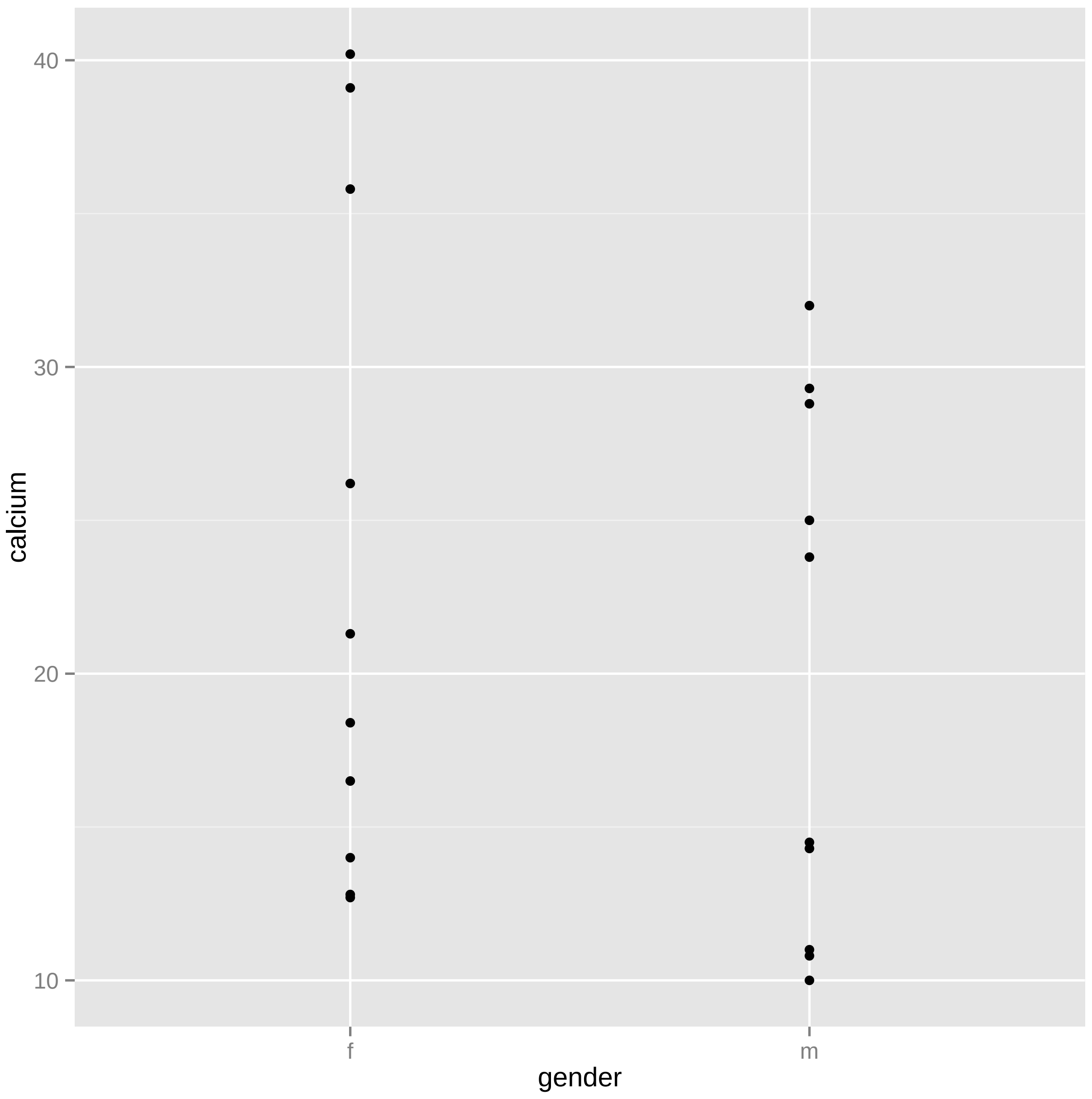
2 qplot

We'll begin with a function called qplot. This function is a quick (hence the q) and dirty way to visualize data with minimal code. Let's use the calcium dataset that we've examined in previous weeks.

```
# read in the calcium dataset
cal <- read.csv("calcium.csv", header = TRUE)
head(cal)

##   calcium gender hormone height
## 1    16.5      f       n   1.60
## 2    18.4      f       n   1.71
## 3    12.7      f       n   1.42
## 4    14.0      f       n   1.66
## 5    12.8      f       n   1.76
## 6    14.5      m       n   1.73
```

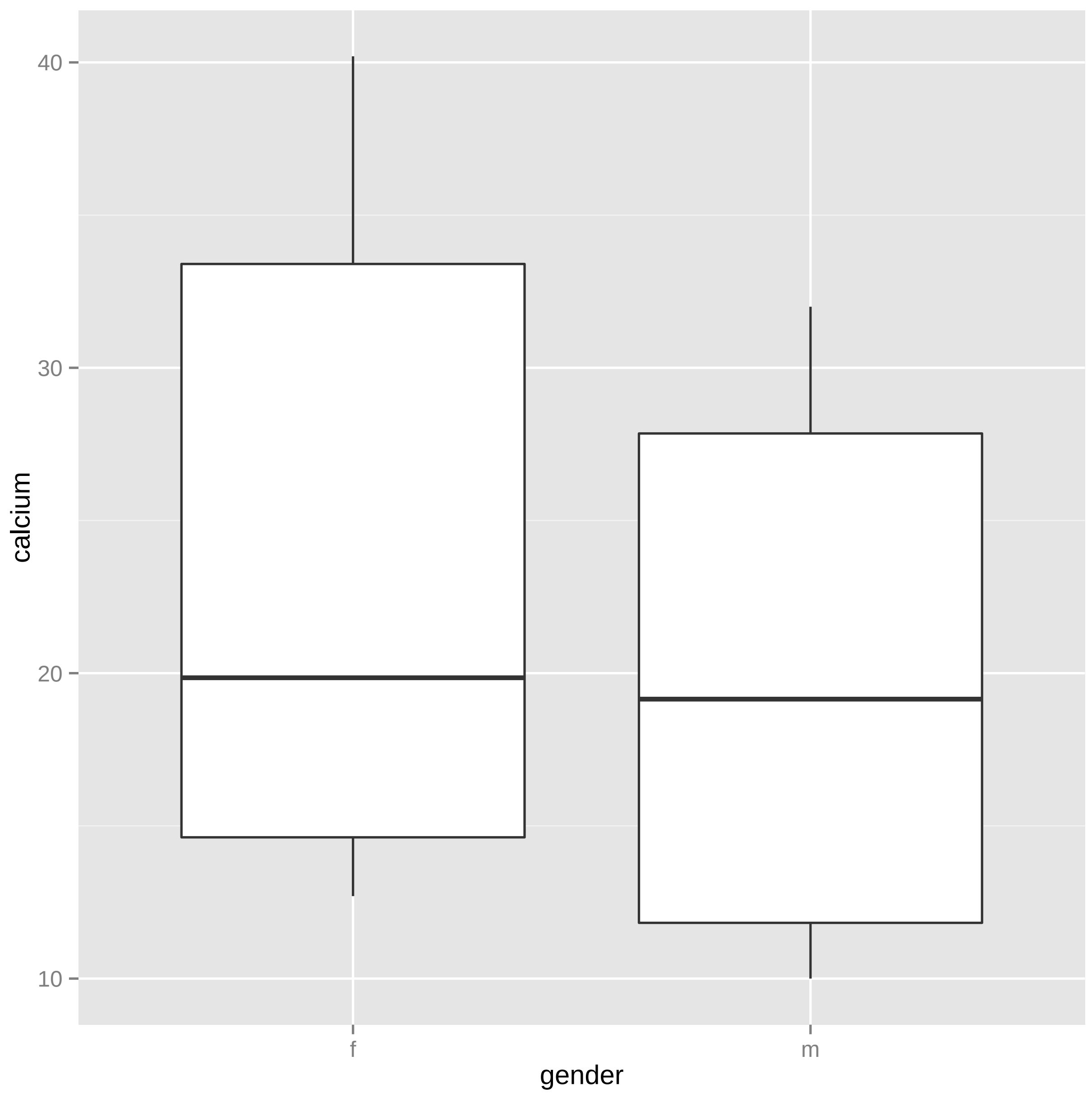
For example, we can look at calcium levels by gender with qplot.

```r
qplot(x = gender, y = calcium, data = cal)
```



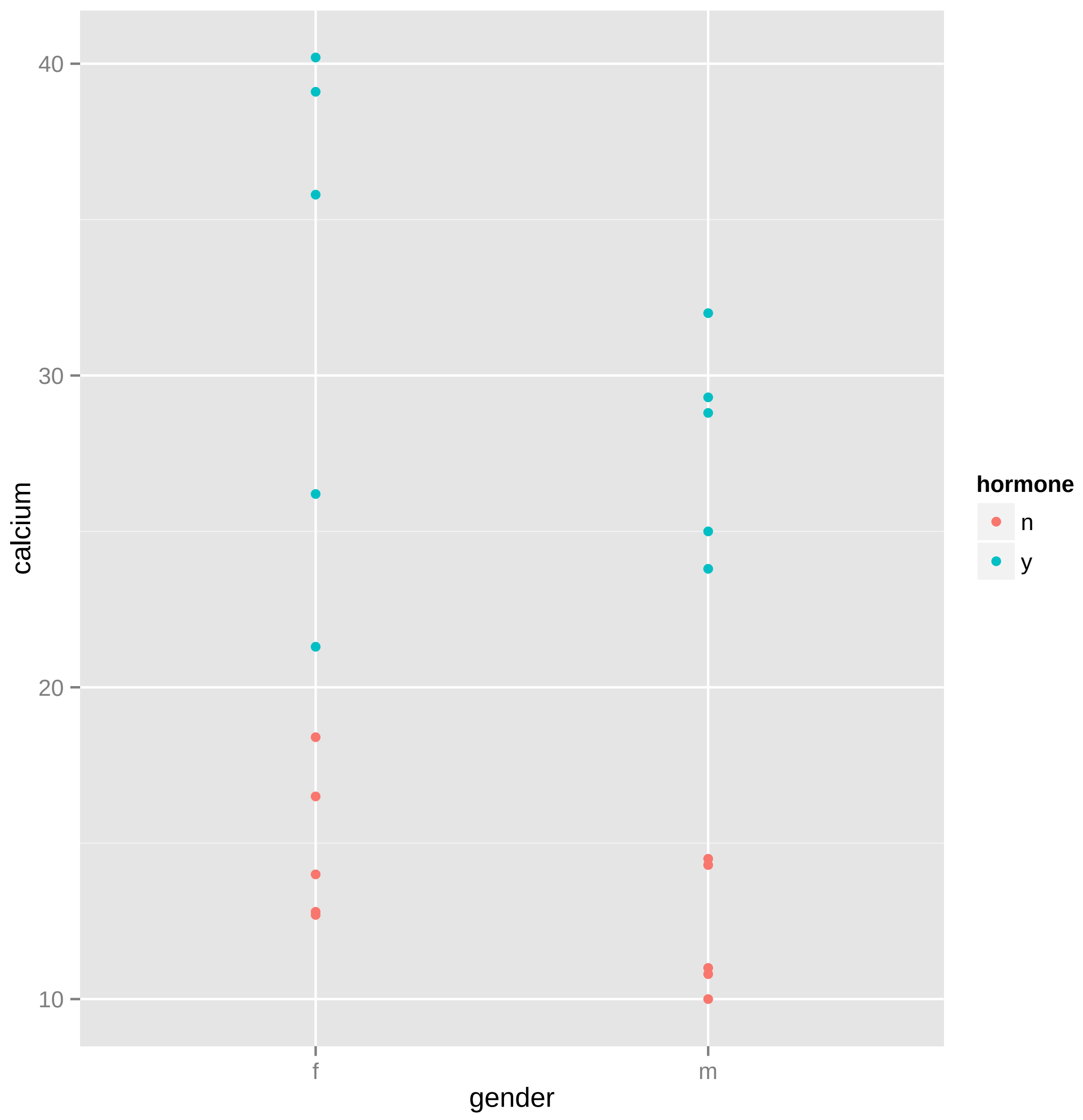
Note that we didn't specify the shape or type of figure. Given the characteristics of the data, qplot tries to come up with an appropriate figure. However we can override the default shape. ggplot refers to the shape of a figure as a geom, short for geometric object. By adding the geom argument to our code, we can get something different.

```r
qplot(x = gender, y = calcium, data = cal, geom = "boxplot")
```



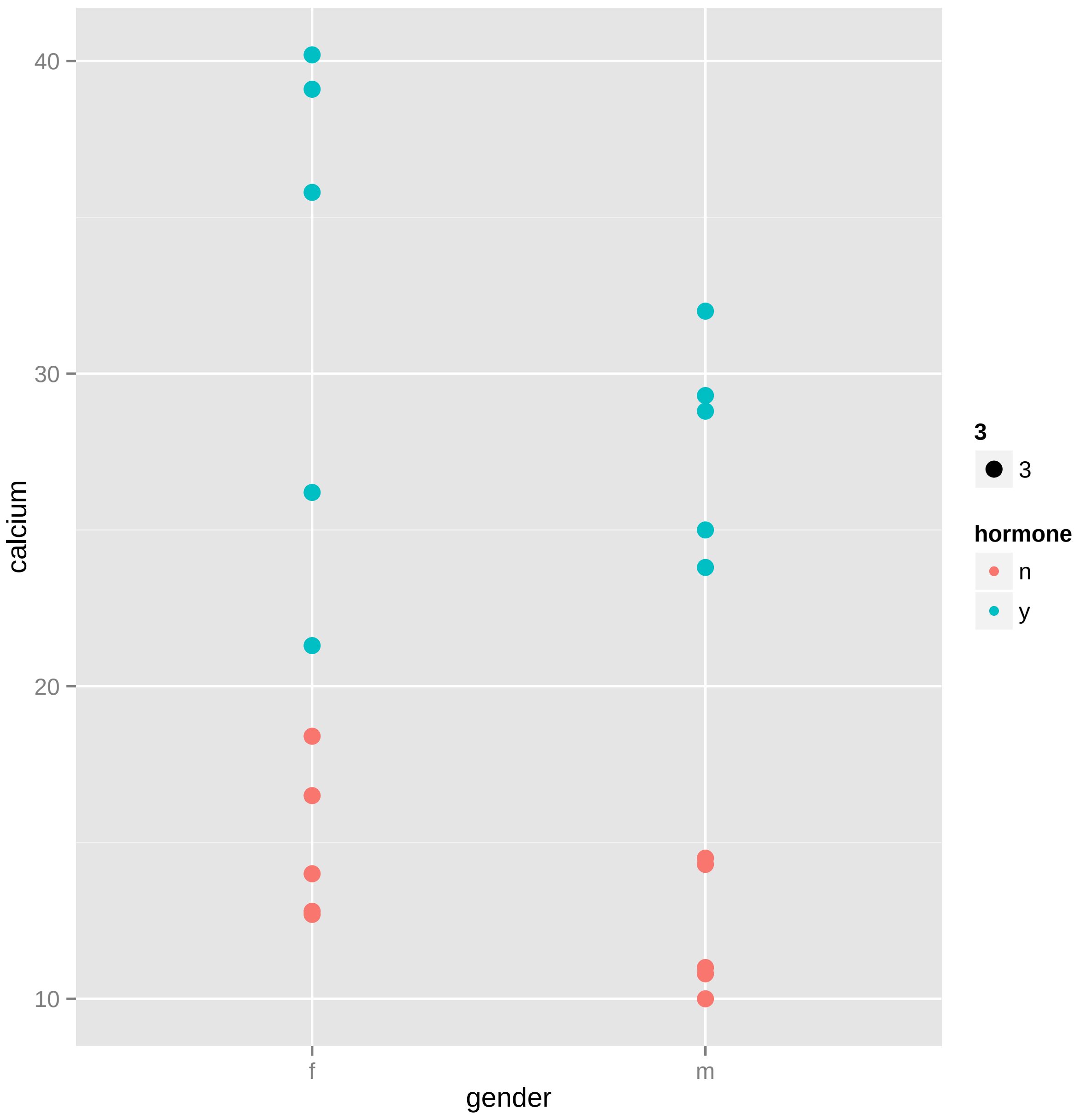
A great thing about the ggplot package is the ability to easily differentiate data points on multiple characteristics. For example, this data set contains males and females, which appear to differ in hormone levels. However, we've ignored the hormone treatments applied. We can highlight the treatment group with the argument color.

```
qplot(x = gender, y = calcium, data = cal, color = hormone)
```



We can further specify other characteristics of the plot. Its a simple thing to change the size of the points.

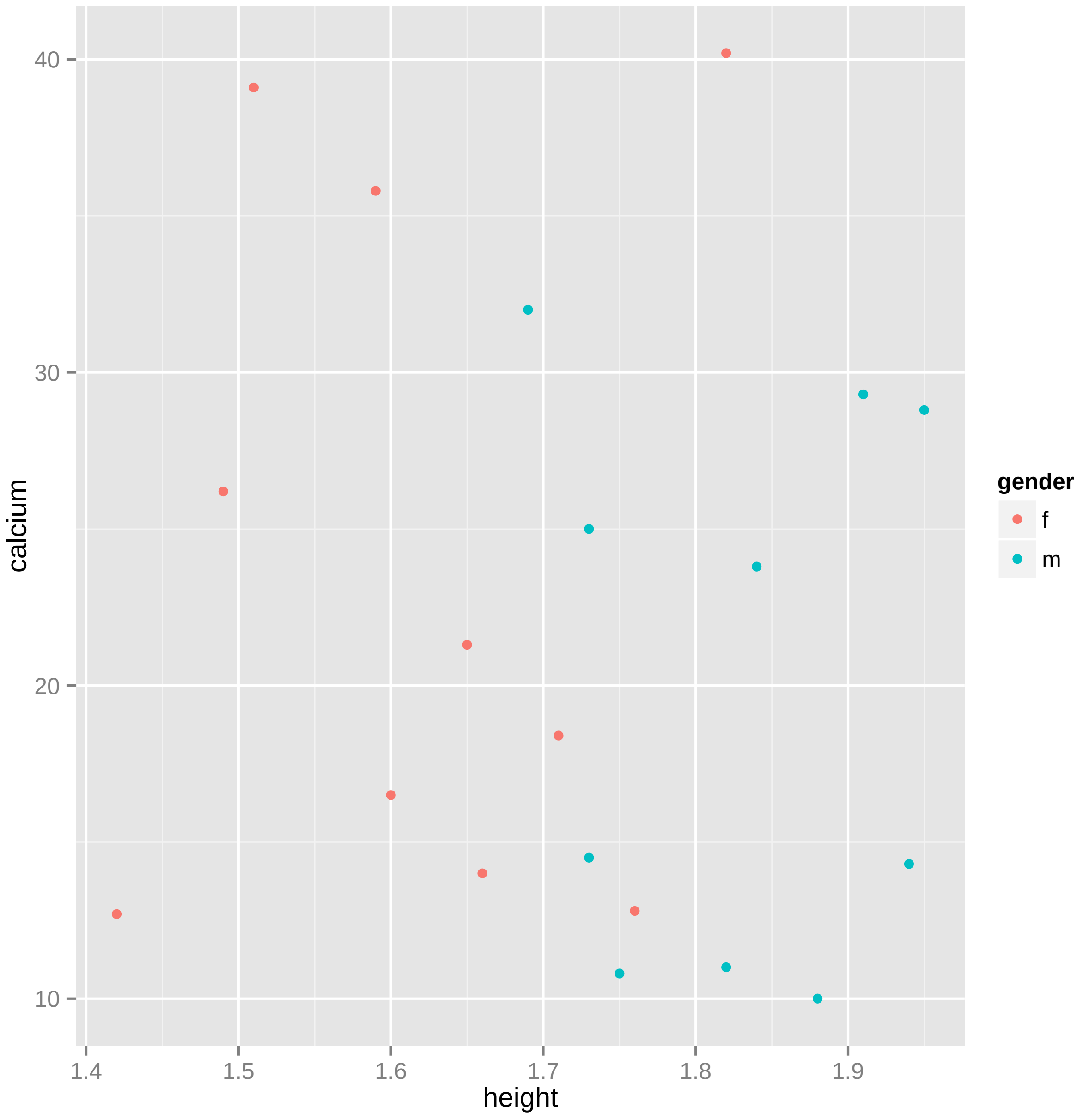
```r
qplot(x = gender, y = calcium, data = cal, color = hormone, size = 3)
```



The legend is unusual here because of the color method we used. We mapped the colors onto the variables, instead of setting the color. Later, we will see how to avoid this by using a more advanced function.

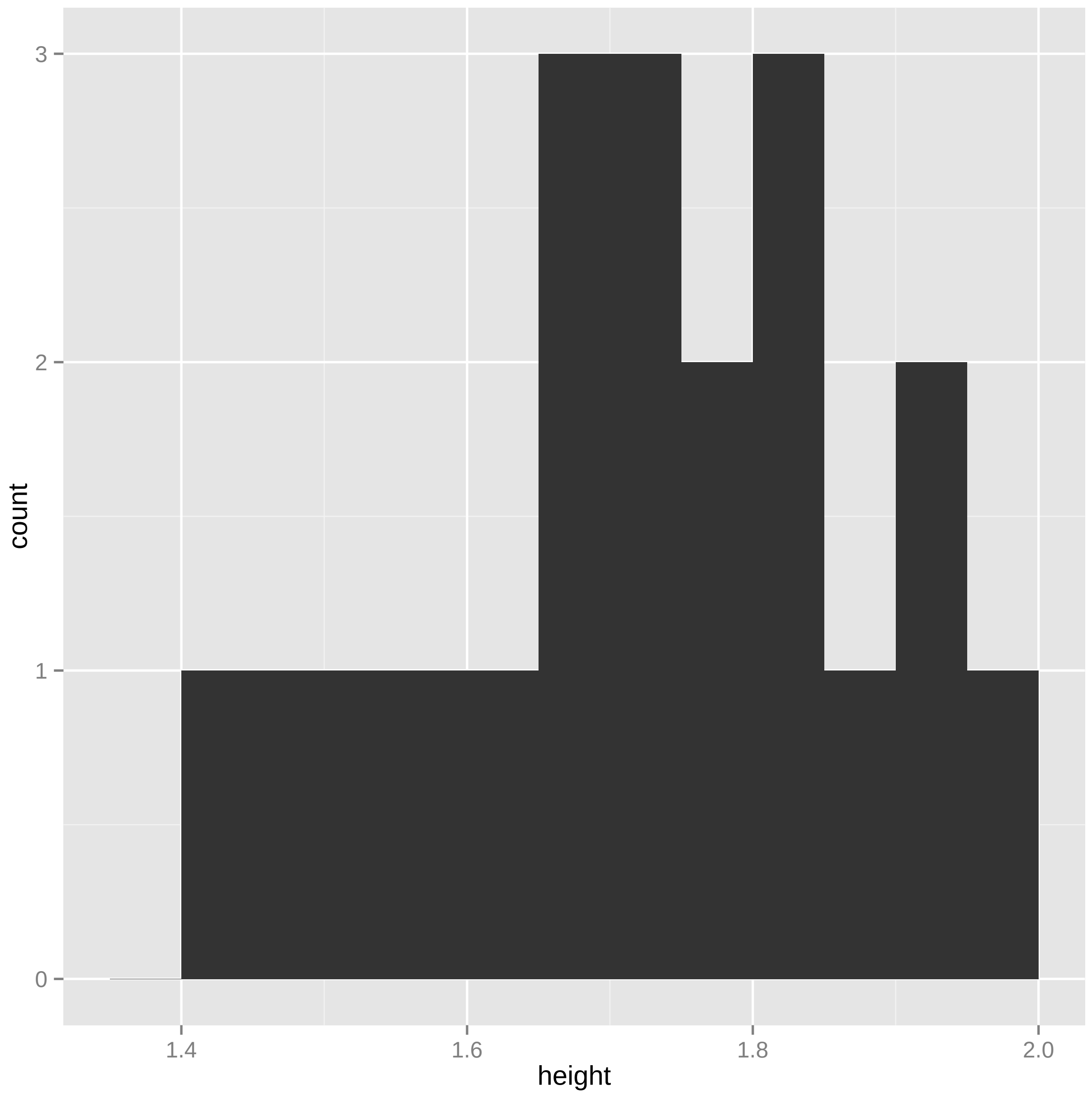
If we provide two continuous variables, qplot produces a scatter plot. Once again, we can identify points by another variable.

```
qplot(x = height, y = calcium, data = cal, color = gender)
```



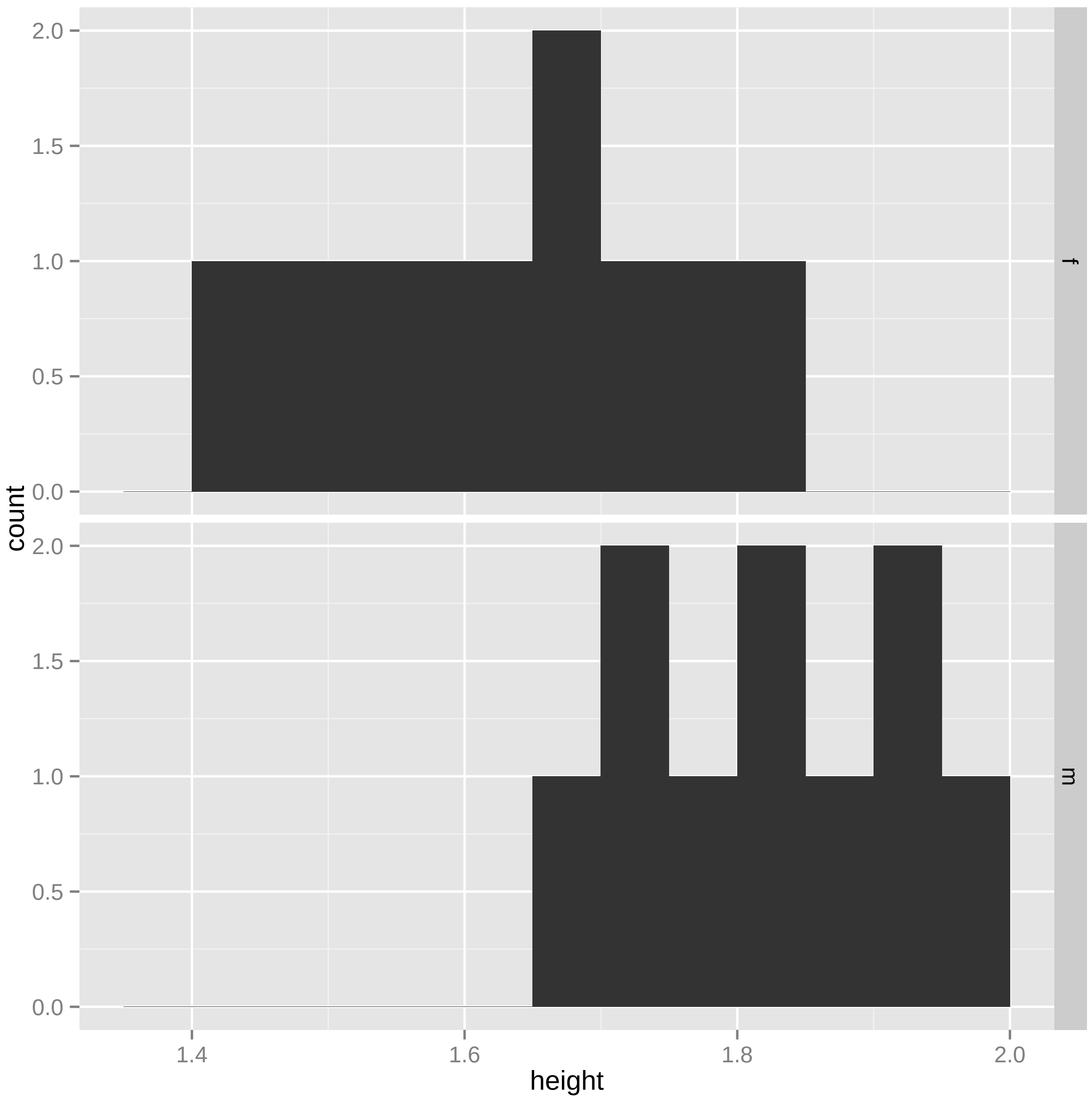
With a continuous x and no specified y, qplot defaults to a histogram. We can specify the width of the bins on the figure.

```
qplot(x = height, data = cal, geom = "histogram", binwidth = 0.05)
```



Sometimes, we may wish to plot two figures adjacent to one another. In ggplot, this is called faceting, and we can separate data points based on another category, much like we did above using color. To do this, we use the facets argument. We then specify our division using formulaic notation. Notice the equation to the right of the tilda can't be blank, so we use a period.

```
qplot(x = height, data = cal, geom = "histogram", binwidth = 0.05, facets = gender ~
    .)
```



3 ggplot

While qplot is very useful to have a quick look at your data, when making plots for a paper or a talk, you may want customization that is beyond the capabilities of this function. This is where ggplot comes to the rescue. The function ggplot is the workhorse of this package, and provides many more options. The language of ggplot is based on a grammatical structure. Most of the details are beyond the scope of this course, but we will provide enough example to get you started and hopefully, to whet your appetite for more. The language of ggplot is built around a grammar, and once a user is proficient with the grammar, she is able to apply her knowledge to create many types of plots and customizations. The details of the grammar are beyond the scope of this course, but we will introduce some of the terminology used in the ggplot function.

One of the most evident differences between ggplot and the base plotting functions in R is the ability to store figures as objects while maintaining the ability to manipulate the figures at a later time. Let's start by creating an object, p, which will serve as the basis for our plot.

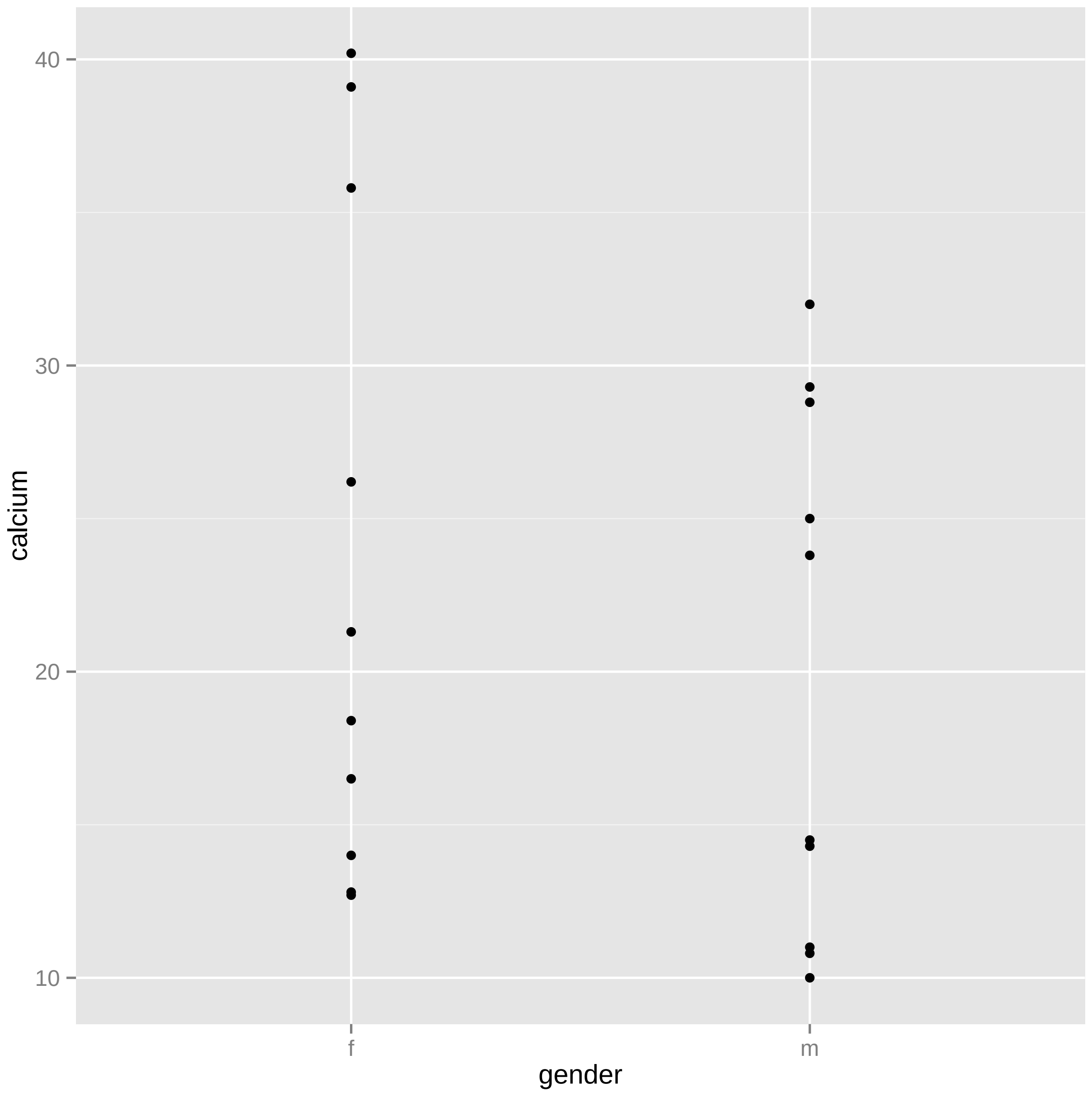
```r
p <- ggplot(data = cal, aes(x = gender, y = calcium))
```

Notice that if you call p, R returns an error. This is because we have not yet told ggplot what shape our figure should take.

3.1 changing plot types with geom functions

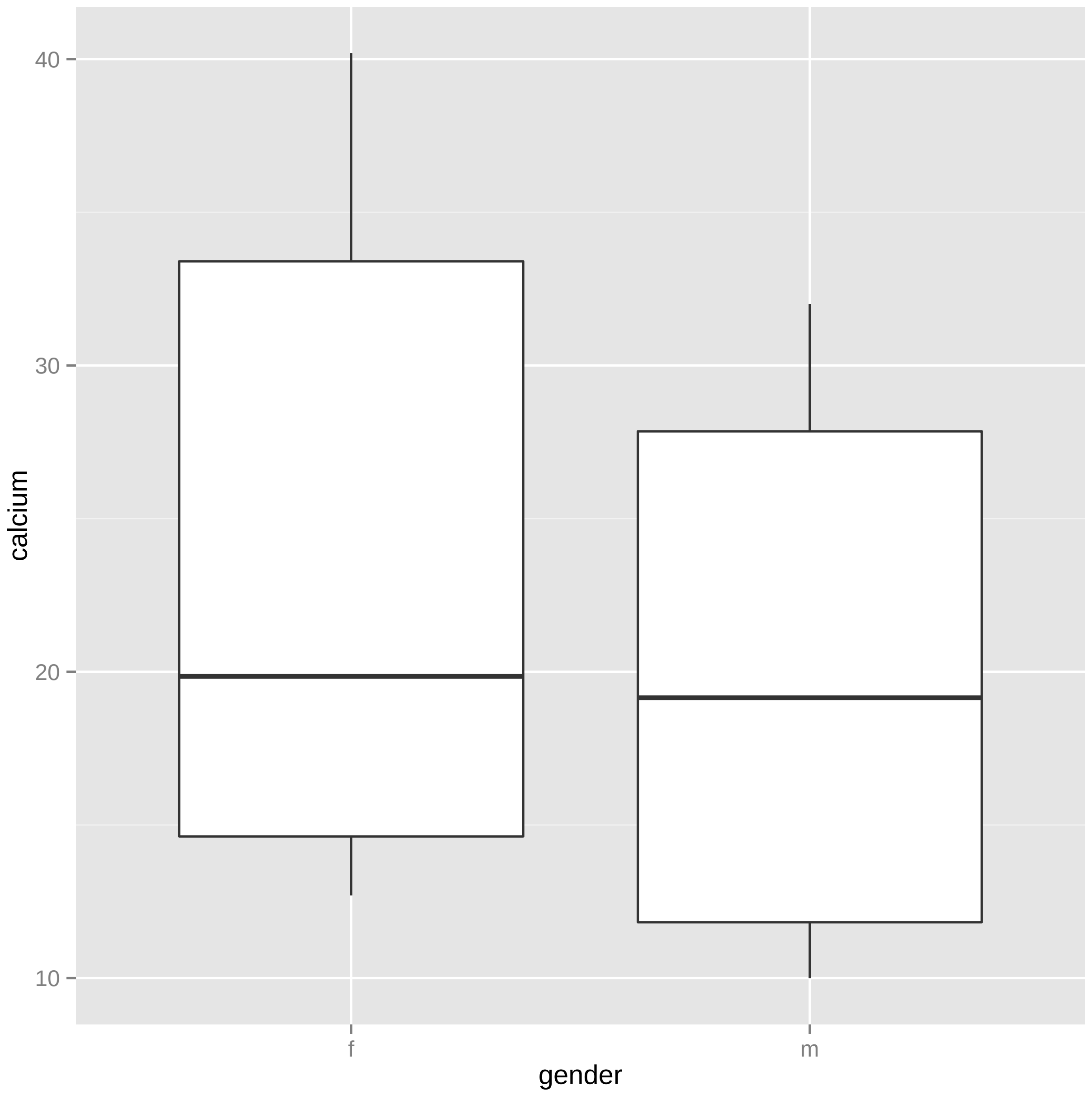
We've created the object p and defined some of its attributes. Specifically, we've told ggplot which data frame to use, and which columns contain our x and y variables. Recall the argument geom from our example above? Now, geom becomes part of a function, which specifies the shape when we add it to the object we created. For example, we can create a scatterplot:

```r
# create a scatterplot
p + geom_point()
```



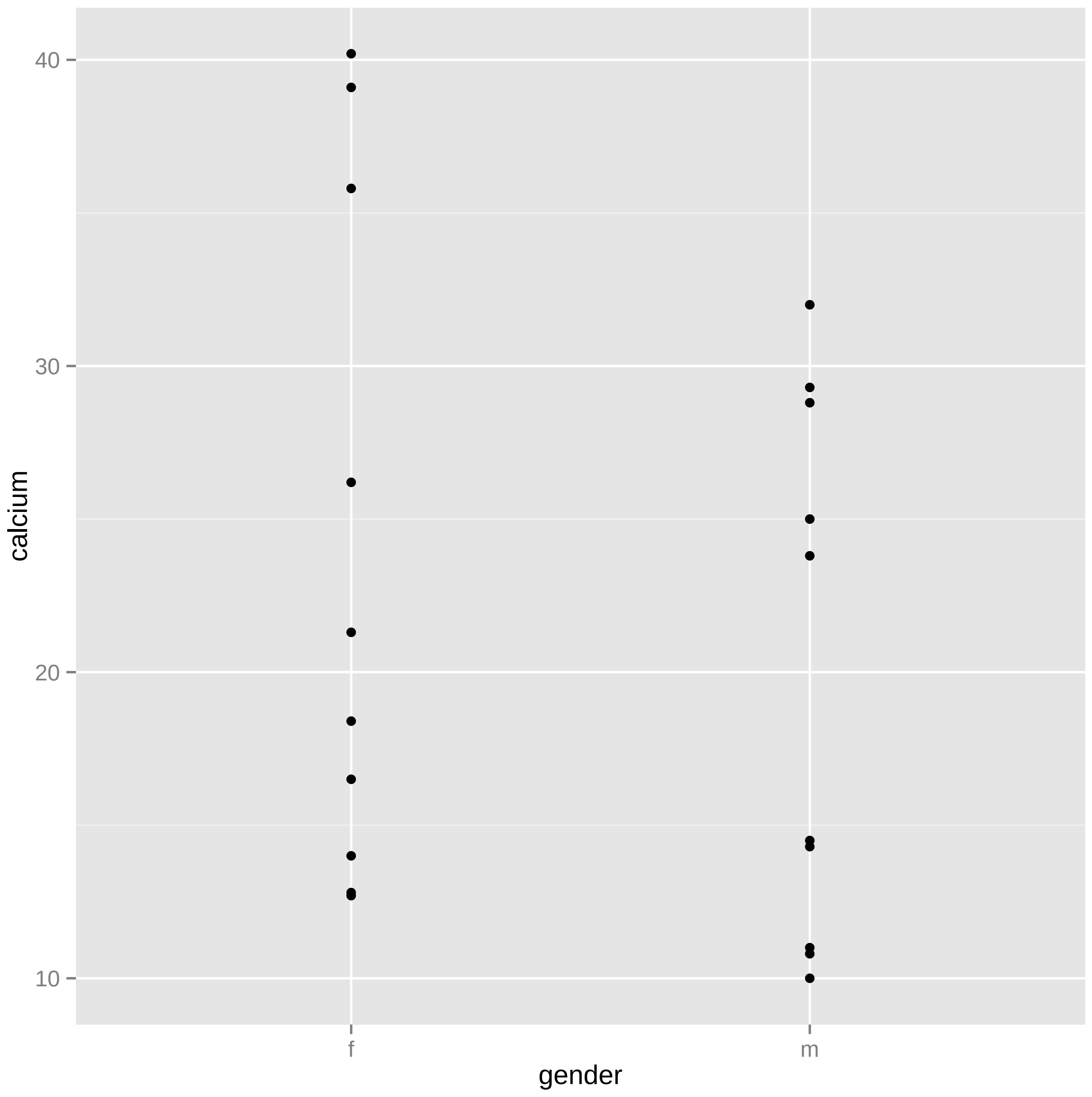
Or a boxplot:

```r
# create a boxplot
p + geom_boxplot()
```



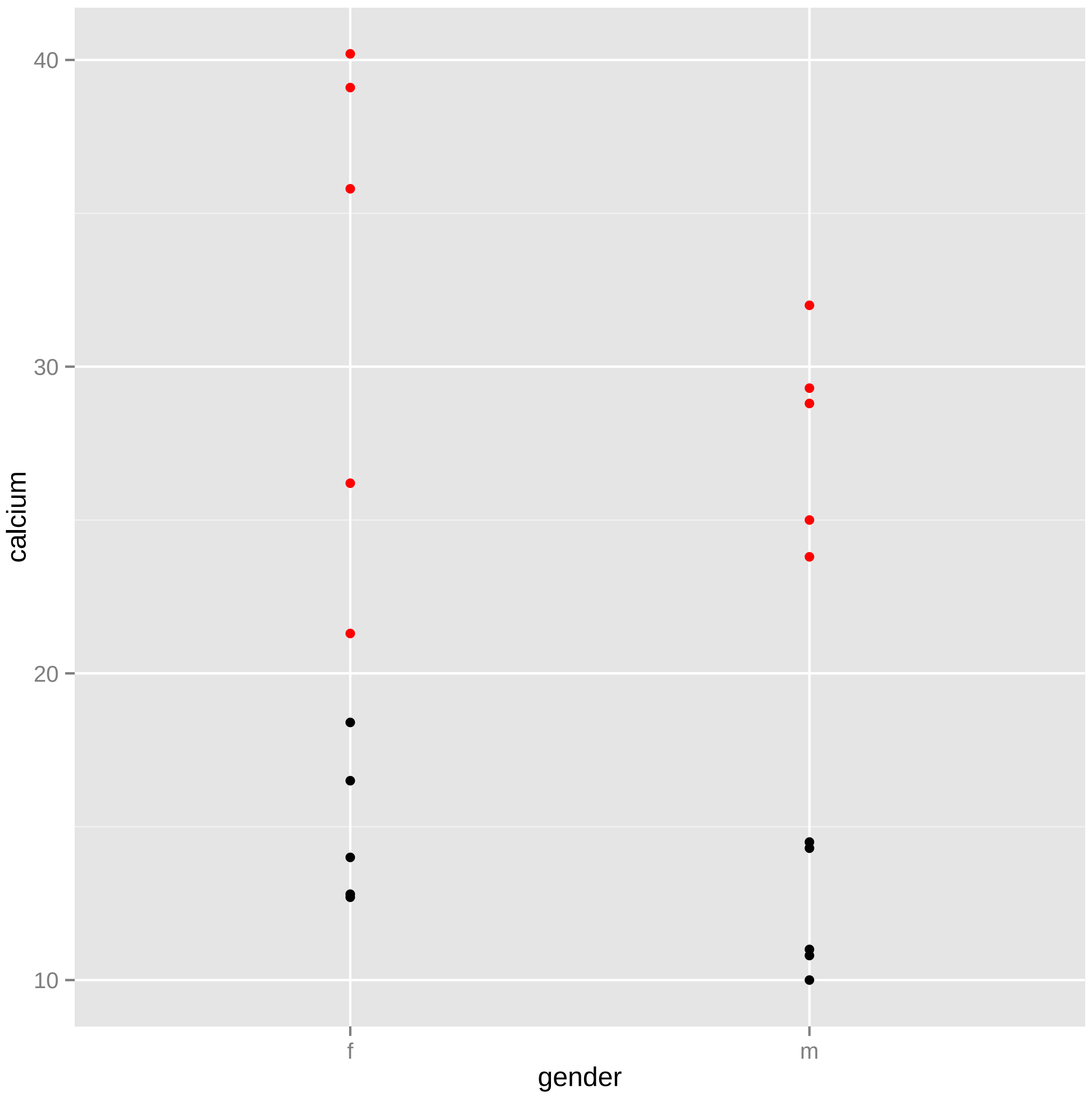
There is no reason we can't store our object with the geom specified. We just assign it a name and then we can call it.

```r
# assign a name to the object with geom specified
p.point <- p + geom_point()
p.point
```



Like we did with qplot, we can use the color argument to identify points based on another variable.

```
q <- p + geom_point(color = cal$hormone)
q
```



Let's look at a few other things we can do by using a larger data set. mtcars is included in the ggplot2 package and contains data from a Car and driver road test of various cars. We can learn about the data set and see the first few rows with these functions.

```
`?`(mtcars)
head(mtcars)

##                    mpg cyl disp  hp drat    wt  qsec vs am gear
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3
##                   carb
## Mazda RX4              4
## Mazda RX4 Wag          4
## Datsun 710             1
## Hornet 4 Drive         1
## Hornet Sportabout      2
## Valiant                1
```

We will begin by creating a simple scatter plot of miles per gallon by weight.

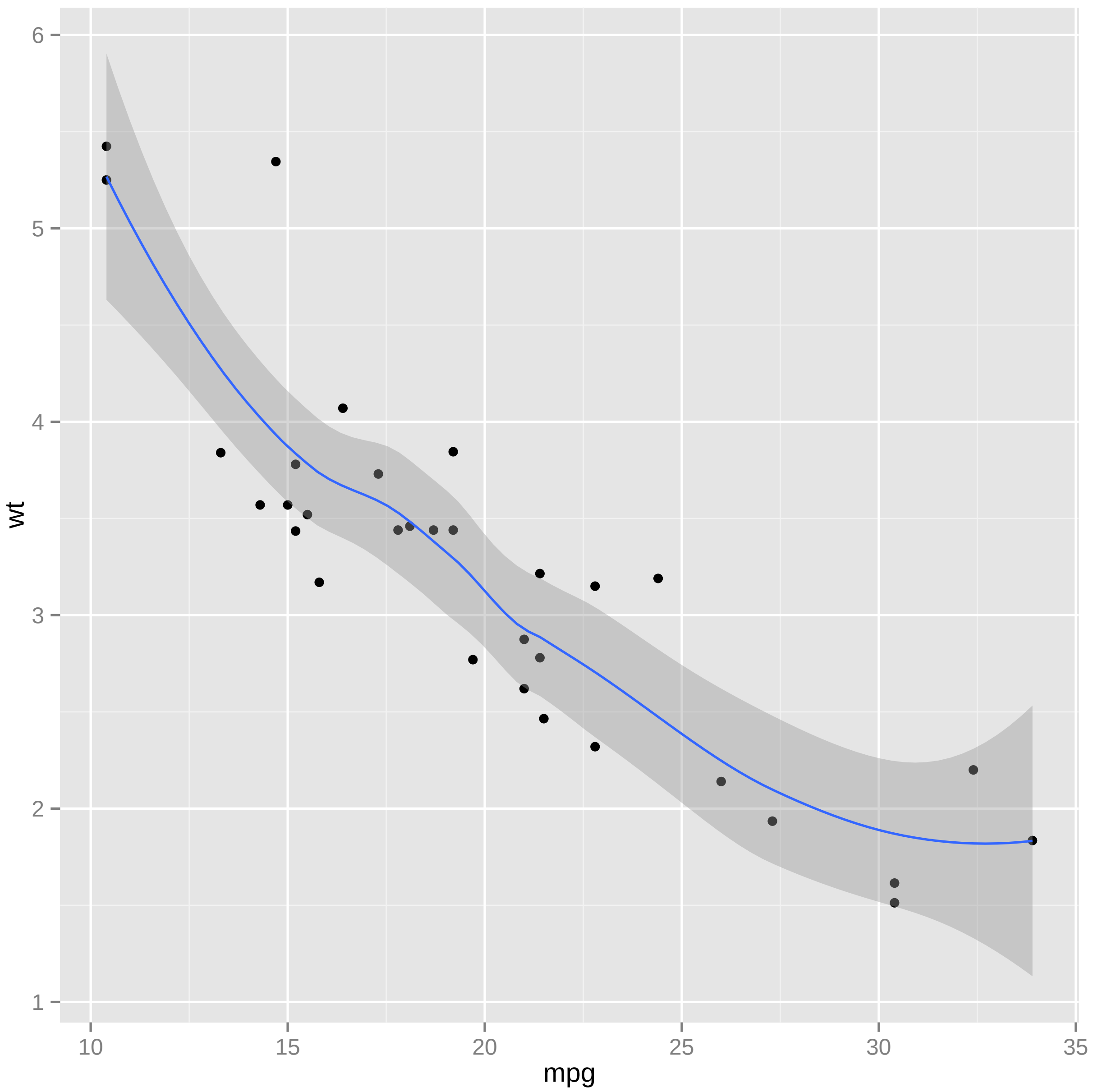
```
p <- ggplot(data = mtcars, aes(x = mpg, y = wt))
summary(p)

## data: mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb
##   [32x11]
## mapping:  x = mpg, y = wt
## faceting: facet_null()
```

The summary function lets us see the attributes of the plot. We have mapped the variable on the plot, but in order to create a layer and display the data, we need to specify the geom. Once we've added that, we can see that the geom is now included in the summary.

```
q <- p + geom_point()
summary(q)

## data: mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb
##   [32x11]
## mapping:  x = mpg, y = wt
## faceting: facet_null()
## -----------------------------------
## geom_point: na.rm = FALSE
## stat_identity:
## position_identity: (width = NULL, height = NULL)
```

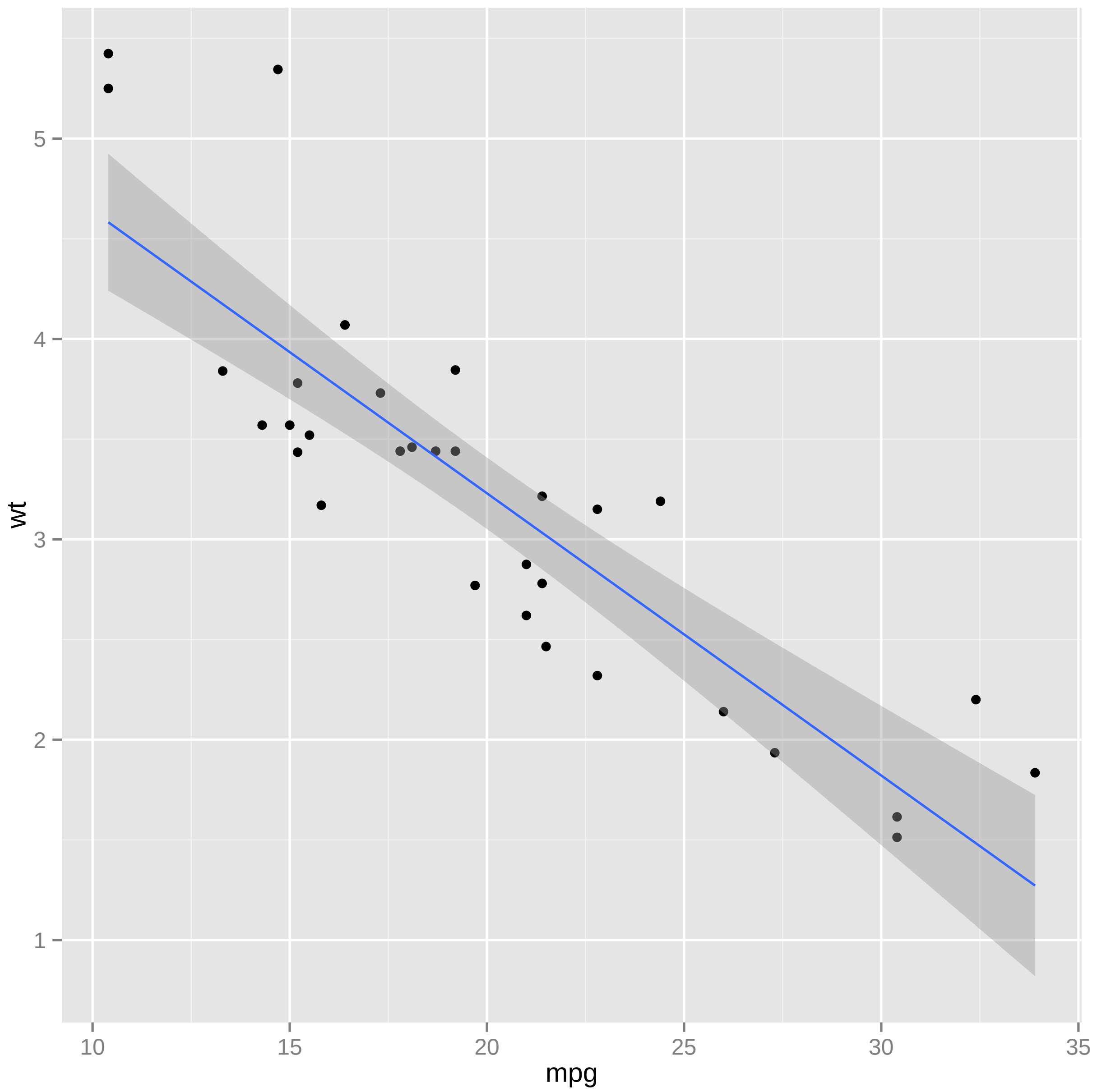
It is easy to add additional elements to a stored plot. For example, we can superimpose a smoothing curve over our existing scatter plot.

```
q + geom_smooth()

## geom_smooth:  method="auto" and size of largest group is <1000, so using loess.  Use 'method
= x' to change the smoothing method.
```

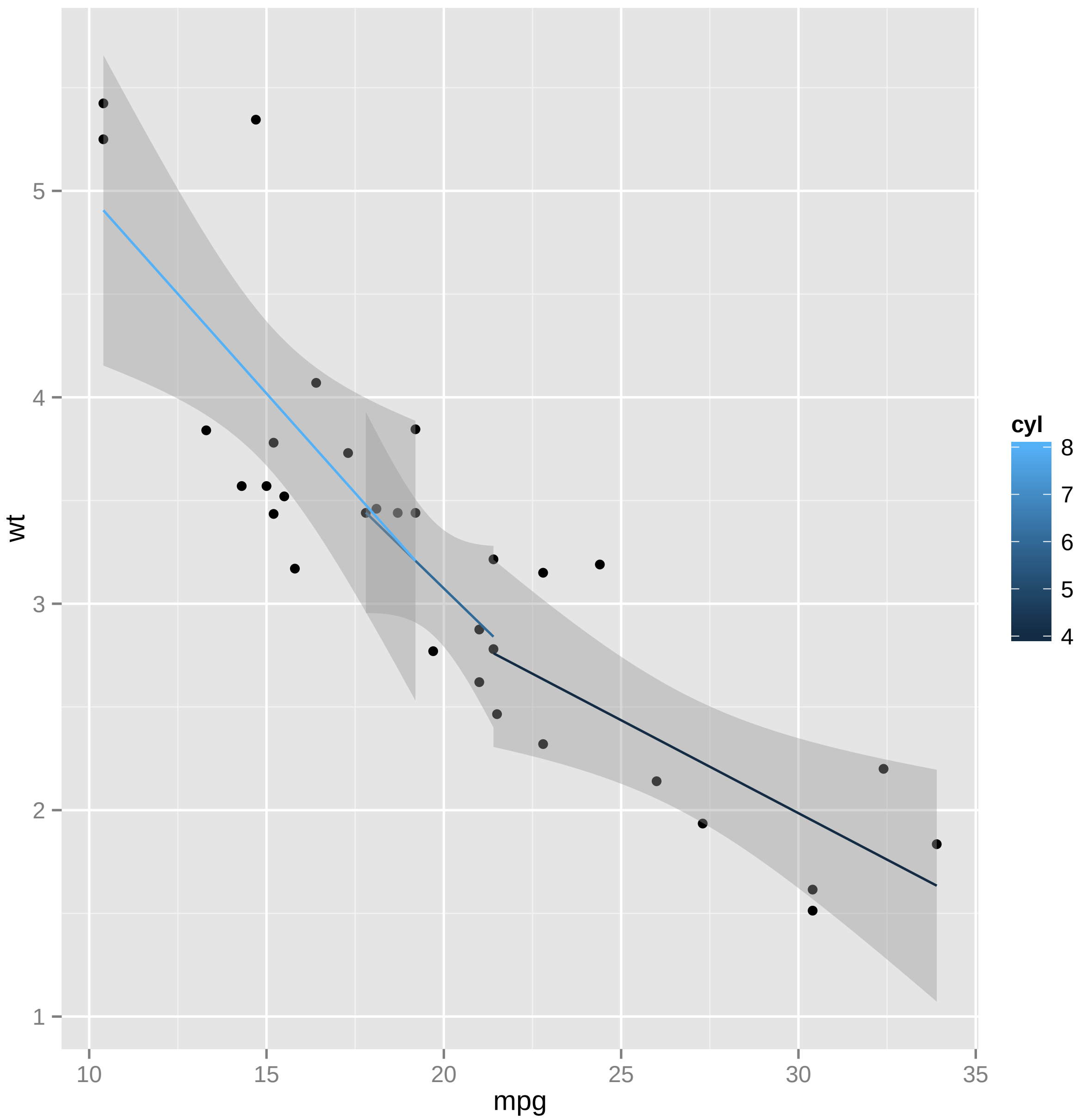


We can also specify that we prefer a regression line. Furthermore, we may like to see regression lines for different groups on one plot. In the second line of code, we specify that we want a line drawn separately for each number of cylinders in the engine (4,6,8), and we want them each to be a different color. We could even specify the color, as in the final line.

```
q + geom_smooth(method = lm)
q + geom_smooth(method = lm, aes(group = cyl, color = cyl))
```

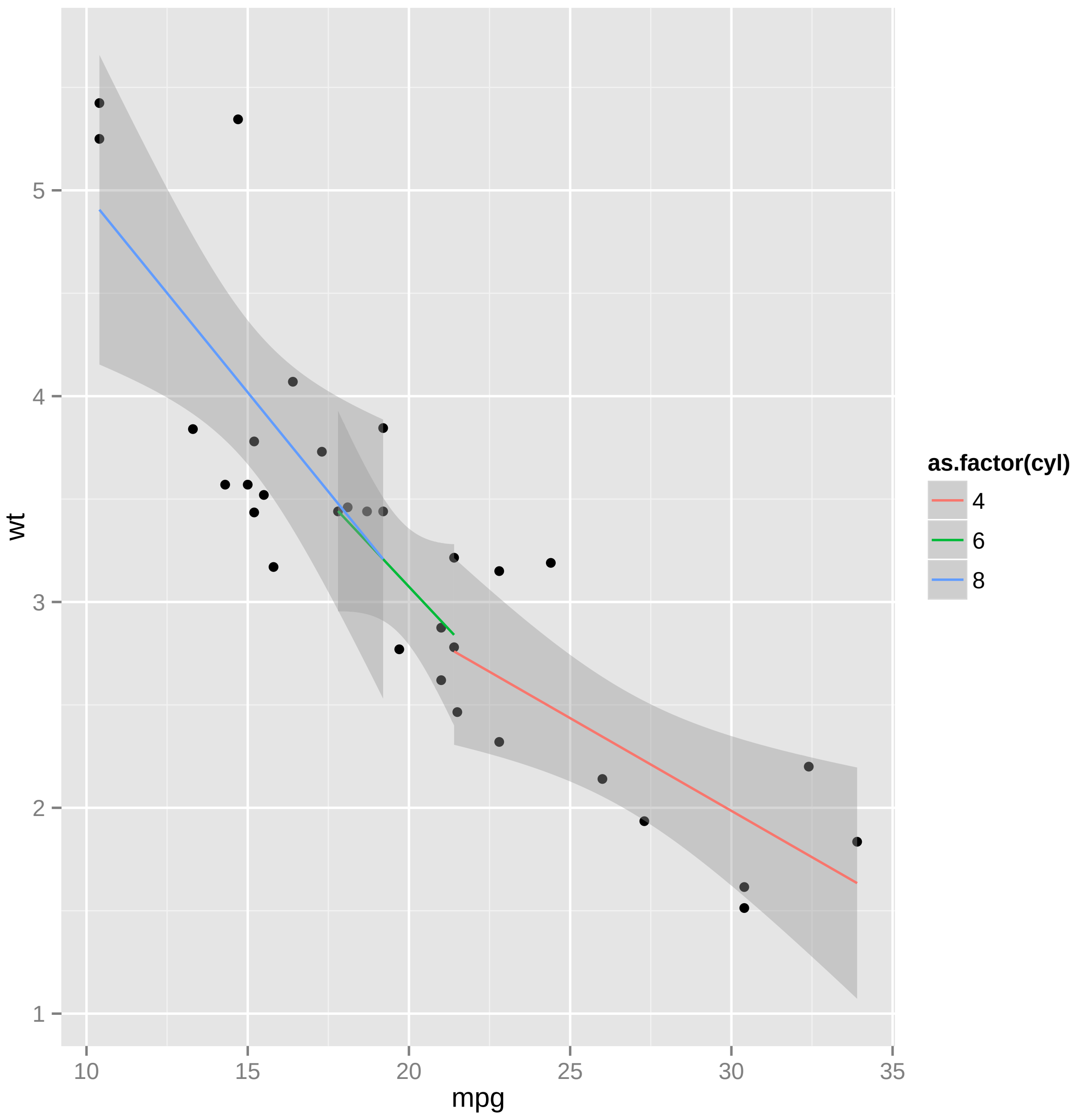


wt
mpg
5
4
3
2
1
10
15
20
25
30
35



ggplot interprets cylinder number as a continuous variable, and thus provides colors that are gradients of one color. We can change the variable to a factor and see how that changes the color.

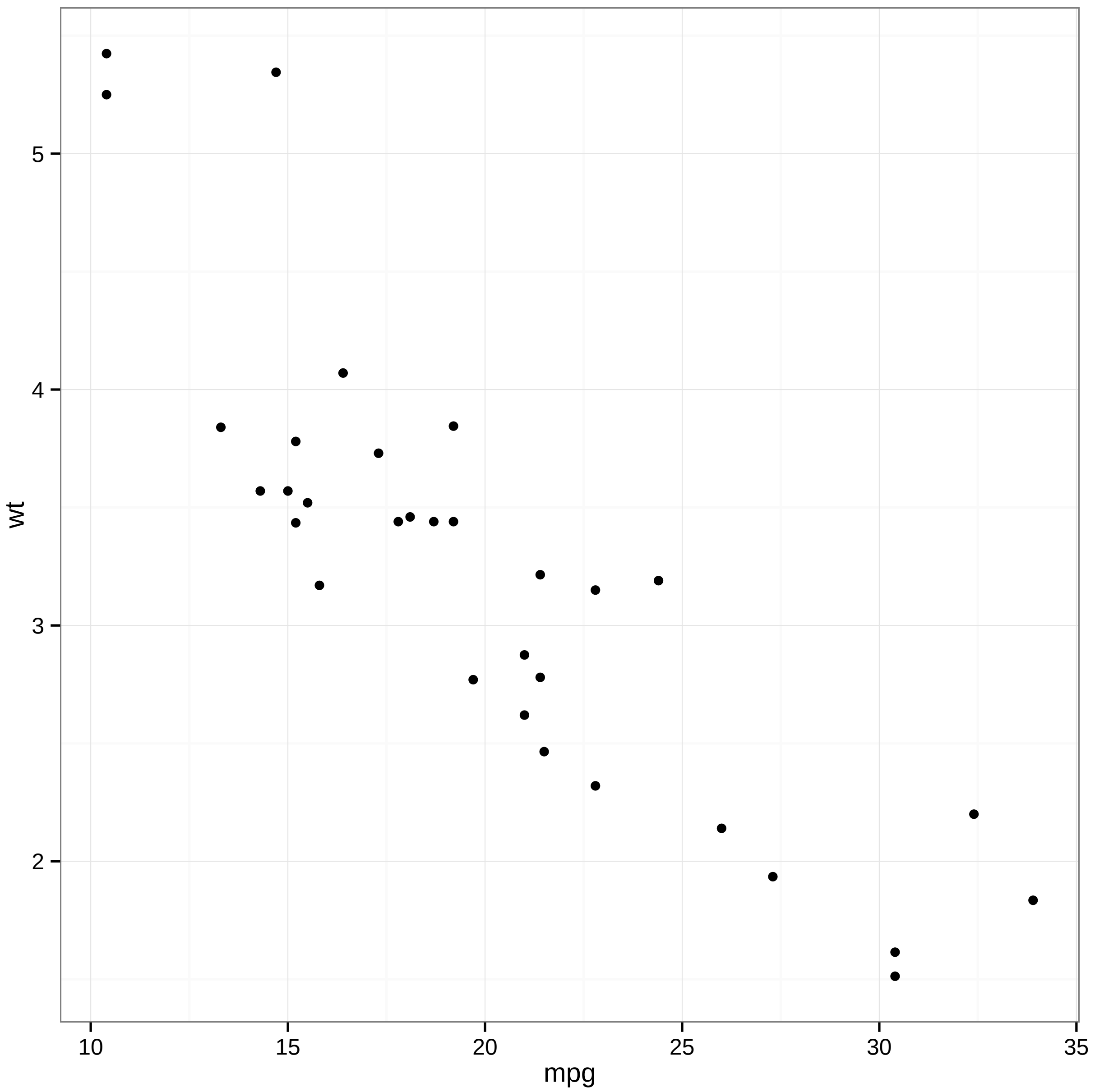
```
q + geom_smooth(method = lm, aes(group = as.factor(cyl), color = as.factor(cyl)))
```



3.2 changing plot background in ggplot

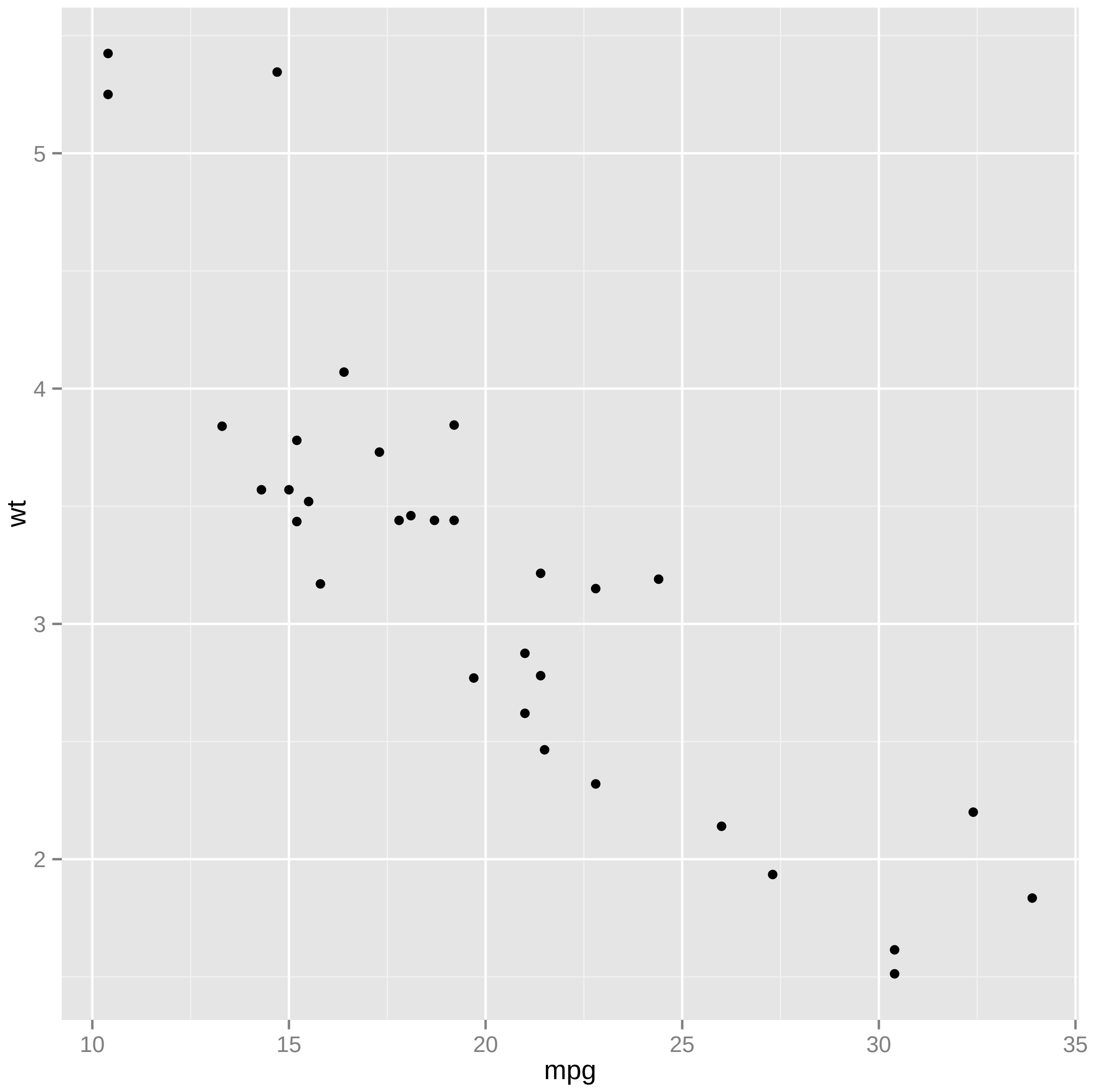
One of the features of ggplot is that figures are always presented with a unified theme. Every element of this theme is customizable, either on a per-need basis, or globally. If you dislike the default theme with the grey background, you can change it into black and white.

```
theme_set(theme_bw())
q
```



You can always change the background back to grey.

```
theme_set(theme_grey())
q
```



3.3 faceting in ggplot

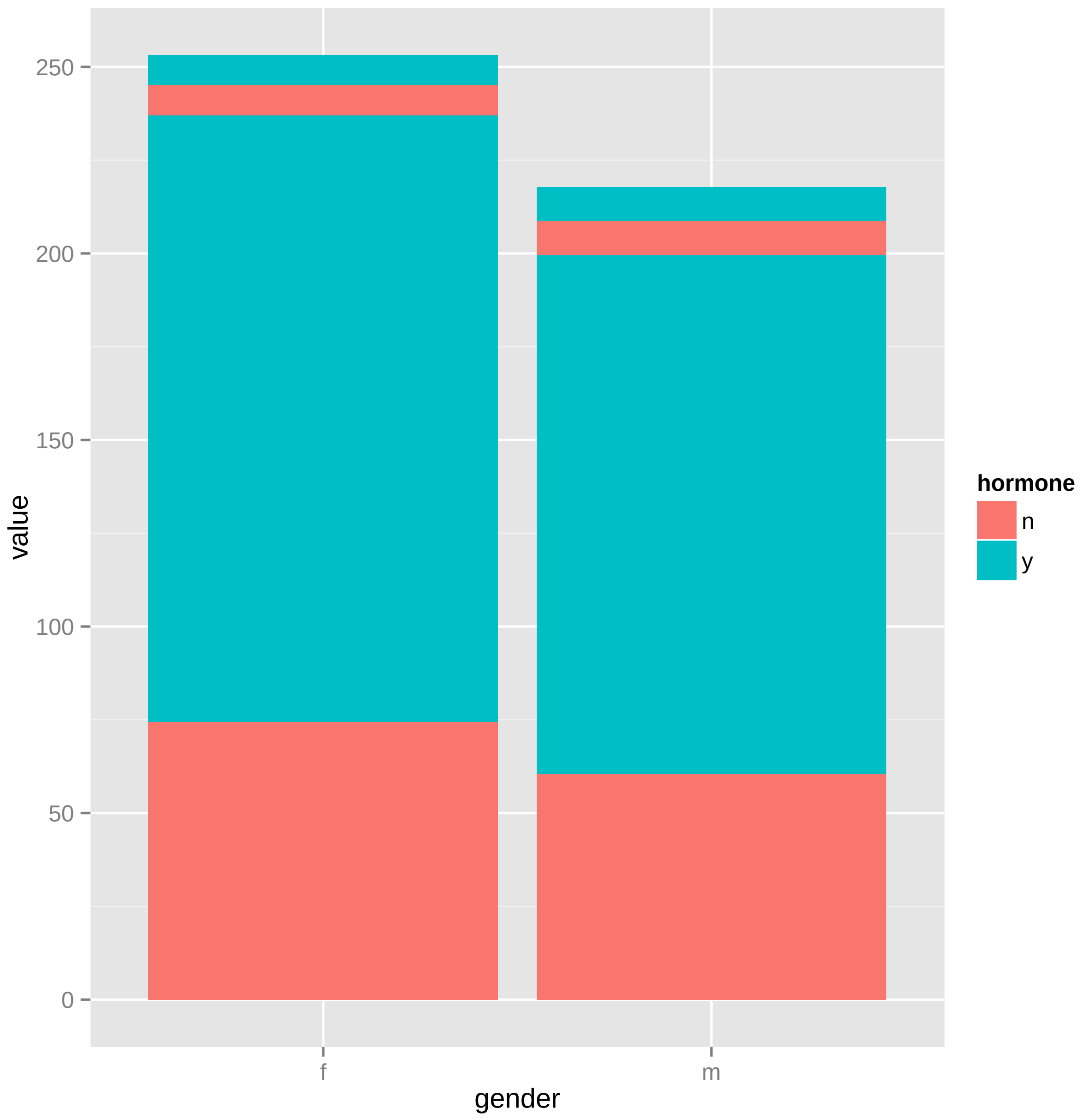
Now we will explore faceting with the ggplot function, and create plots similar to those we made in the previous exercises. We begin by rearranging our cal data with a handy little function called melt found in the reshape2 package. The melt function will reorganize the data based on which identifying variables (id.vars) we want to feature in the plot. In this case, those are gender and hormone.

```
install.packages("reshape2", dependencies = TRUE)

## Error:  trying to use CRAN without setting a mirror

library(reshape2)
cal.melt <- melt(cal, id.vars = c("gender", "hormone"))
head(cal.melt)
```

```
##   gender hormone variable value
## 1      f       n  calcium  16.5
## 2      f       n  calcium  18.4
## 3      f       n  calcium  12.7
## 4      f       n  calcium  14.0
## 5      f       n  calcium  12.8
## 6      m       n  calcium  14.5
```

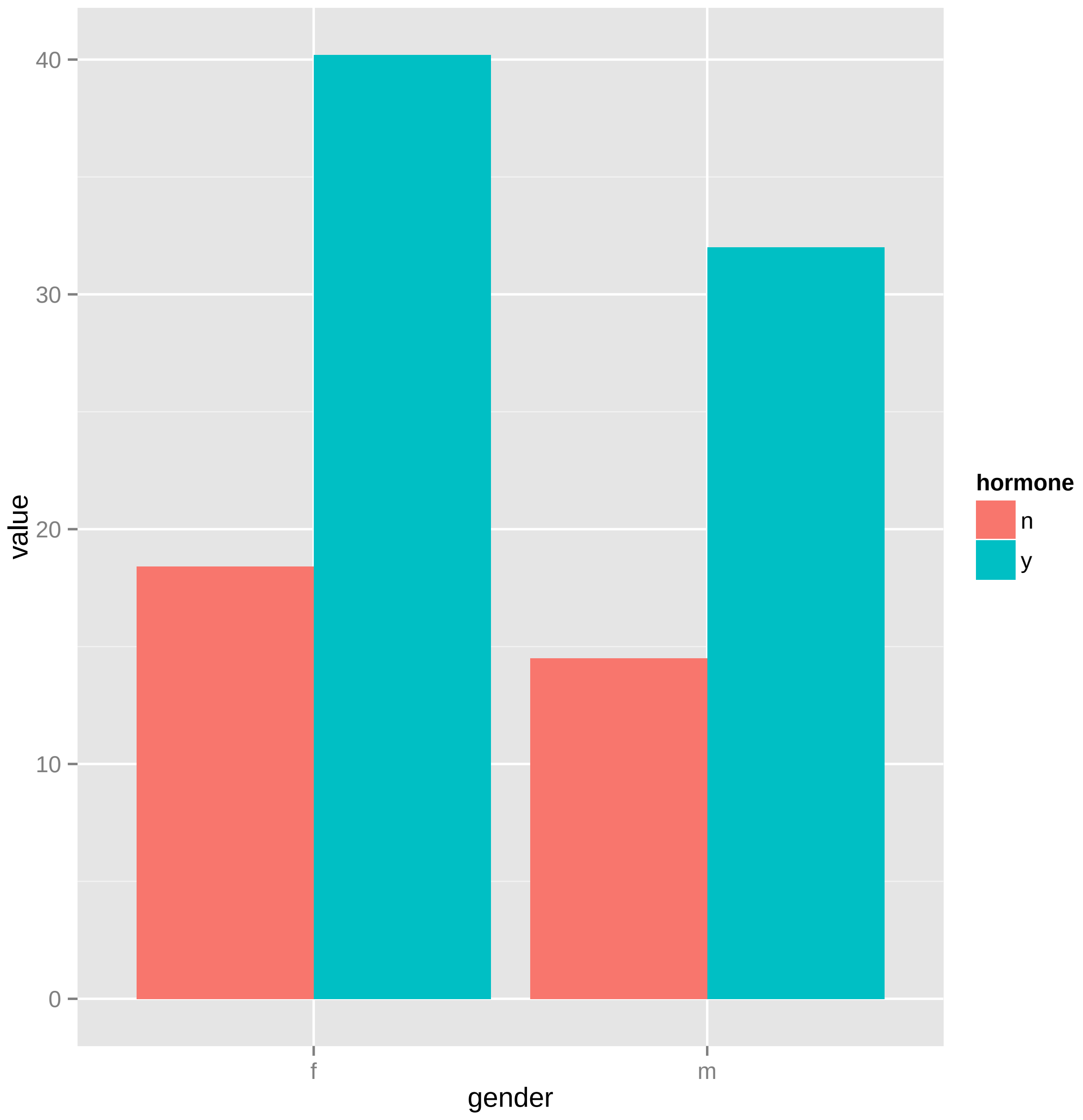
Now we set up our plot details and aesthetics. ”fill” is the argument to the aes function that will color our bars based on the factor specified. Then, we add the geom. The geom boxplot needs to be told which statistic (stat) to use in order to create the appropriate bars. We choose identity because we do not wish for any transformations or other changes to the data.

```
p <- ggplot(data = cal.melt, aes(x = gender, y = value, fill = hormone))
p + geom_bar(stat = "identity")
```

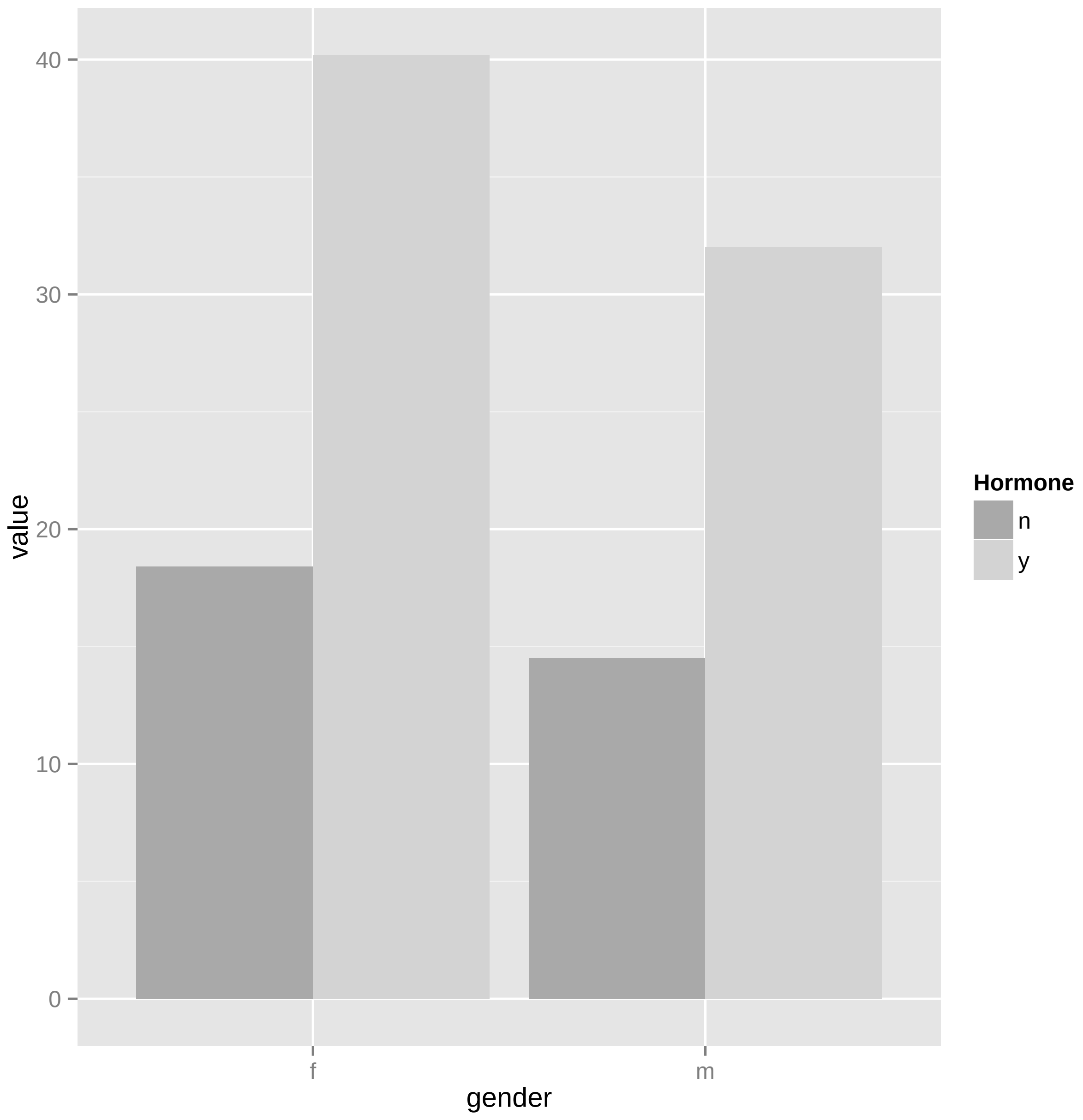


Oops. Notice that our bars are stacked and not adjacent to one another as we prefer. In order to change that, we use the position argument. The option dodge will place the bars side-by-side. Once we have that how we like it, we can change the color. We provide a label for the legend, and specify which colors we prefer.

```
q1 <- p + geom_bar(stat = "identity", position = "dodge")
q1
# scale_fill_manual allows you to specify the color in the barplot
q1 + scale_fill_manual("Hormone", values = c("darkgrey", "lightgrey"))
```



40
30
20
10
0
value
f
m
gender
hormone
n
y



4 Conclusion

The options for customization are endless. There are two very important resources for users of ggplot. The first is a book from the author of the package. In it, Wickham (2009) provides all of the framework for a user3 to begin with ggplot. Second, there is a very active online community of users. Typically, questions are answered quite soon after they are posted. The forums, including an online manual, can be found via http://ggplot2.org/.