Chapter 4: Data Visualization

Mark Andrews

Contents

Plotting in R with `ggplot` **3**

Histograms. density plots, bar plots, etc **6**
- Histograms . . . 6
- Frequency polygons . . . 11
- Density plots . . . 14
- Barplots . . . 16

Tukey boxplots **21**

Scatterplots **27**
- Adding marginal distributions . . . 29
- Adding a smoothing function . . . 31
- Adding labels . . . 35
- Bubbleplots . . . 37

Facet plots **38**

References **41**

Data visualization is a major part of data analysis. Far from being just a means to add some eye-candy or ornamentation to otherwise dull reports or powerpoint slides, visualization allows us explore data and find patterns that would easily be missed were we to rely only on numerical summary statistics. A classic example that vividly illustrates this point is known as *Anscombe's quartet* (Anscombe 1973). In this example, there are 4 separate data sets, each with two variables labelled x and y. The means and standard deviations of both the x and y variables are identical across all four data sets. Likewise, the Pearson's correlation coefficient between the x and y is also identical across the data sets. These summary statistics are shown in the following table.

set	mean(x)	mean(y)	sd(x)	sd(y)	cor(x, y)
I	9	7.5	3.32	2.03	0.82
II	9	7.5	3.32	2.03	0.82
III	9	7.5	3.32	2.03	0.82
IV	9	7.5	3.32	2.03	0.82

On the basis of these numbers, the four data sets seem to be identical, or at least it seems likely that they will be highly similar. However, when we visualize the scatter plots of x and y variables for each case, it is evident that they differ from one another in substantial ways, see Figure 1. A key characteristic of data visualization, therefore, is that "it forces us to notice what we never expected to see" (Tukey 1977). In other words, data visualization is not simply a means to graphically illustrate what we already know, but to reveal patterns and structures in the data.

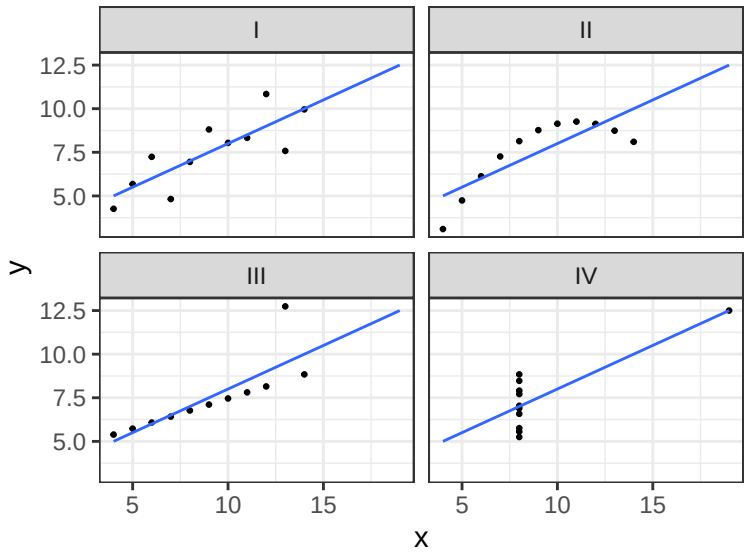


Figure 1: Anscombe's quartet. Across all of the four data sets shown here, the means and standard deviations of the x and y variables are identical, as is the correlation between x and y.

There is no strict formula or procedure that we must always follow when visualizing data. Any data set can be visualized is many ways; some of these will be informative and provide insight, others will not. Generally, we should aim to fully explore the data and maximize what can be learned from it. To do so, we should always try to look at the data in different ways, from different perspectives, and at coarser and finer levels. This process is iterative in that we may start with something very simple and then progressively refine it. Sometimes, we need to work by trial and error, starting with some visualization technique that then needs to be abandoned in favour of another. In general, we should keep in mind some guiding principles. For example, Hartwig and Dearing (1979), when describing exploratory data analysis generally, state we that we should be guided by principles of *scepticism* and *openness*. We ought to be sceptical to the possibility that any visualization may obscure or misrepresent our data, and we should be open to the possibility of patterns and structures that we were not expecting. Some guiding principles for visualization mentioned by Edward R. Tufte in his *Visual Display of Quantitative Information* (Tufte 1983) are the following.

- *Above all else show the data*
- *Avoid distorting what the data have to say*
- *Present many numbers in a small space*
- *Encourage the eye to compare different pieces of data*
- *Reveal the data at several levels of detail, from a broad overview to the fine structure*

Just as we have guiding principles, so too are there major visualization tools or techniques that should generally be known. The major tools that we will consider here primarily include the following.

- *Histograms, density plots, bar plots*: These are used to display the distribution of values of continuous and discrete variables.
- *Boxplots*: Like histograms and density plots, boxplots (or box-and-whisker plots) display the distribution of values of continuous variables. However, they are more closely tied to robust statistical descriptions and so deserve to be treated as a class onto themselves.
- *Scatterplots*: Scatterplots and their variants such as *bubbleplots* are used to display bivariate data, or the relationships between two variables. Usually, scatterplots are used in cases where both variables are continuous, but may also be used, though perhaps with additional modification, when one variable is discrete.

There are, however, many other important types of data visualization methods. For example, heatmaps are

used to display large rectangular grids of data, and geospatial maps display data superimposed on maps of physical space such as a country. As important as these methods are, we have not included them in order to keep this chapter of a manageable size. There are many goods books that delve much deeper into data visualization using `ggplot`. Highly recommended is Healy (2019).

Plotting in R with `ggplot`

In R, there are two major sets of tools for visualization. These are usually known as the *base R* and the *ggplot* plotting systems. The base R plotting system is part of the default, or base, installation of R. Its principal command is `plot`. The ggplot system is provided by the `ggplot2` package (see Wickham 2016), which is part of the *tidyverse*, and its principal command is `ggplot`. The *gg* in its name refers to the *grammar of graphics* (Wilkinson 2005; Wickham 2010), which is a system of rules for mapping variables in a data set to properties (e.g. shape, size, colour, position) of a plot. While the base R plotting system is powerful and not to be dismissed or seen as obsolete, here will exclusively use ggplot. The reason for this is that ggplot is a higher level plotting system, meaning that allows us to create complex visualizations with a minimal amount of code. The same visualizations can almost always be produced using the base plot as well, but doing so usually involves much more code and much more fine tuning.

To use the ggplot plotting system, we must first load the `ggplot2` package. This can be loaded direcly as follows.

```
library(ggplot2)
```

Alternatively, we can just load `tidyverse`, which then loads `ggplot2`.

```
library(tidyverse)
```

There are three key pieces of information that we specify with the `ggplot` command:

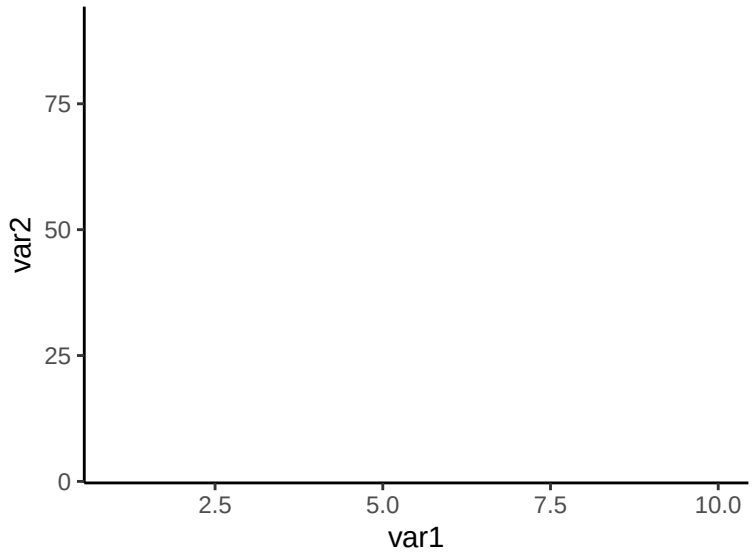
1. The data set containing all the data. This is almost always a single data frame, but sometimes it is multiple data frames.
2. A set of *aesthetic mappings*. This is where we map variables in the data frame to properties like shape, size, colour, position, etc., of the graphic.
3. A set of *layers* that render the aesthetic mappings, usually according to prespecified *geoms* or geometric objects like lines, points, bars, etc.

The best way to understand these components and how `ggplot` works is to work with some very simple examples. To begin, we'll create a very simple data frame.

```
simple_df <- tribble(
  ~var1, ~var2, ~var3,
  1,   4,  'a',
  3,   10, 'a',
  7,   40, 'b',
  10,  90, 'b'
)
```

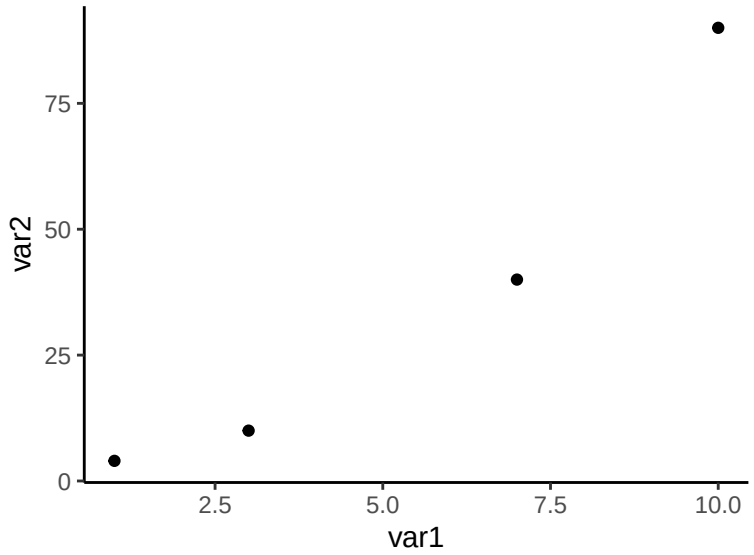
Let's say that we want to create a simple scatterplot where the values of `var1` specify the position of points along the x-axis and the values of `var2` specify the positions of points along with y-axis. To do so, we would start by specifying the data frame we're using as `simple_df` and the specify the aesthetic mapping we want as in the following command.

```
ggplot(simple_df,
       mapping = aes(x = var1, y = var2)
)
```



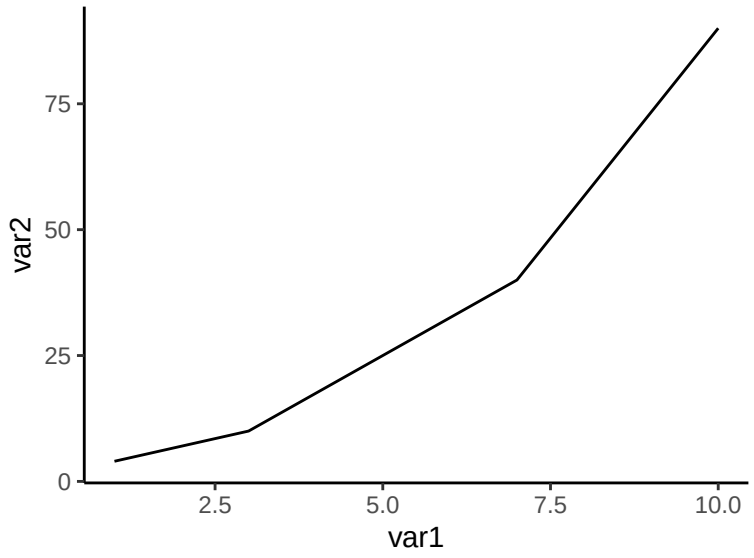
As can be seen, in itself, this did not produce much. It essentially created a blank canvas with the `var1` and `var2` mapped to the x-axis and y-axis, respectively. To see the points, we need to add a layer that tells `ggplot` how to render the graphic using the mapping we have set up. Because we want a scatterplot, we use the `geom_point` geom function as follows.

```
ggplot(simple_df,
       mapping = aes(x = var1, y = var2)
) + geom_point()
```



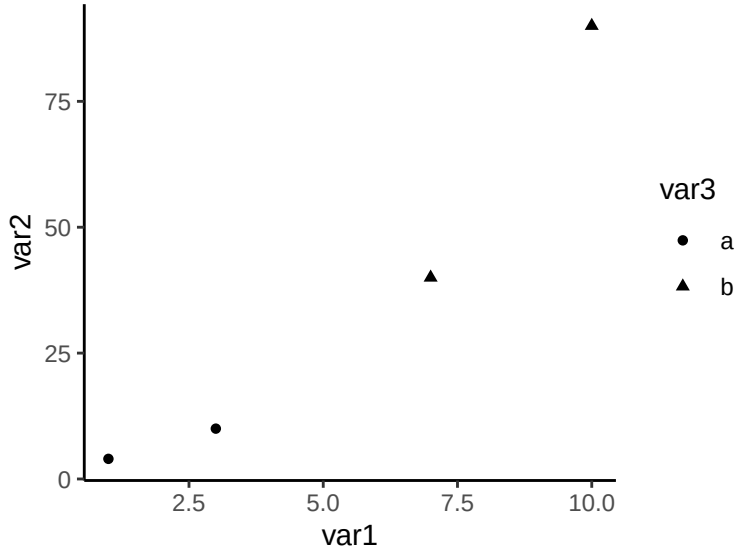
If we now prefer to display this same data using a line plot, we can exchange `geom_point()` for `geom_line()`.

```
ggplot(simple_df,
       mapping = aes(x = var1, y = var2)
) + geom_line()
```



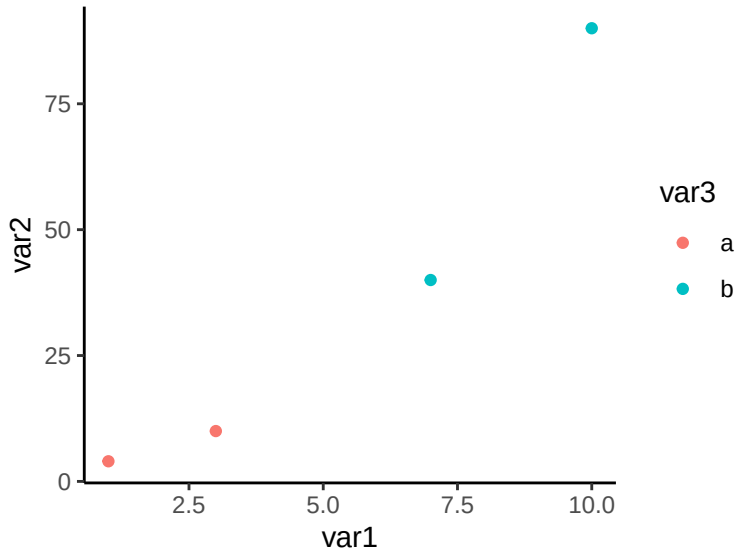
We could extend the original scatterplot by adding an additional aesthetic mapping by using `var3`. Because this is a discrete variable, it could be mapped to a discrete property of the points, such as their shape. This is very simple to do. We need only add `shape = var3` to the `aes` function, and then when we render with `geom_point`, the points will have shapes according to their values of `var3`.

```
ggplot(simple_df,
       mapping = aes(x = var1, y = var2, shape = var3)
) + geom_point()
```



Or if we prefer to represent `var3` with colours, we could use `colour = var3` (we could also use `color = var3`, if you prefer American English spelling, or even just `col = var3`).

```
ggplot(simple_df,
       mapping = aes(x = var1, y = var2, colour = var3)
) + geom_point()
```



These examples just scratch the surface of what `ggplot` is capable of, but they do illustrate its general functionality. In the following sections, we will provide much more depth and explore many more features of `ggplot`. Before we proceed, we should note that in all the above example, we were using somewhat verbose commands. For example, the second argument to `ggplot` is always assumed to be the `aes` mapping (the first argument is always the name of the data frame), so we could drop `mapping =`, as in the following example.

```
ggplot(simple_df,
       aes(x = var1, y = var2)
) + geom_point()
```

Likewise, because the first and second argument to `aes` are assumed to be the `x` and `y` axis mapping, we could drop the `x =` and `y =`, as in the following example.

```
ggplot(simple_df,
       aes(var1, var2)
) + geom_point()
```

In what follows, we will usually keep the more verbose style, but you may prefer to drop it as you get more used how `ggplot` works.

Histograms. density plots, bar plots, etc

Histograms and related visualization methods like frequency polygons, area plots, density plots, and bar plots, are simple but still highly effective tools to visualize the distribution of values of continuous variables.

Histograms

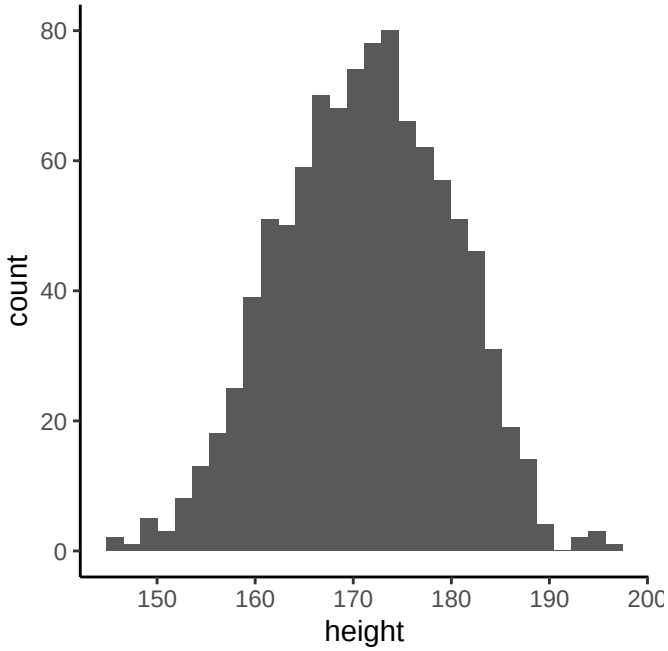
Histograms are one of the simplest and generally useful ways of visualizing distributions of the values of individual variables. To illustrate them, we'll use the `weight` data frame, from which will be downsample to 1000 points.

```
weight_df <- read_csv("data/weight.csv") %>%
  sample_n(down_sample)
```

If we want to display the distribution of the `height` variable, we would proceed as follows.

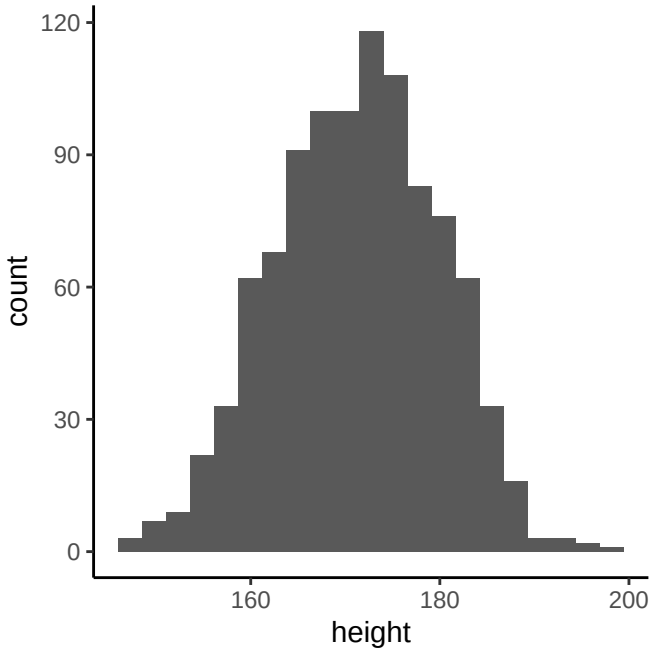
```
ggplot(weight_df,
       mapping = aes(x = height)
```

```
) + geom_histogram()
```



By default, the histogram will have 30 bins. It is usually good to override this either by specifying another value for `bins`, or by specifying the `binwidth`. Given that the `height` variable is measured in centimeters, we can specify that each bin should be 2.54 cm, or one inch, as follows.

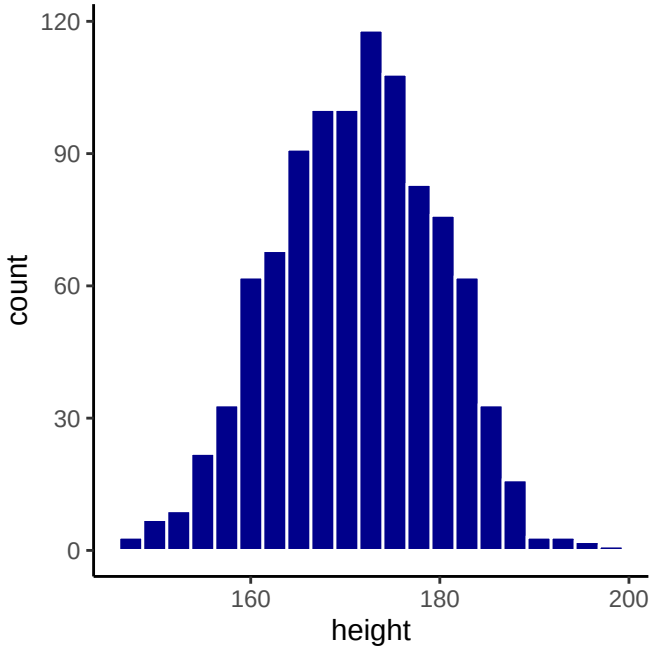
```
ggplot(weight_df,
       mapping = aes(x = height)
) + geom_histogram(binwidth = 2.54)
```



Any histogram consists of a set of bars, and each bar has a colour for its interior and another for its border. The interior colour is its `fill` colour, while `colour` specifies the colour of its border. If, for example, we wanted

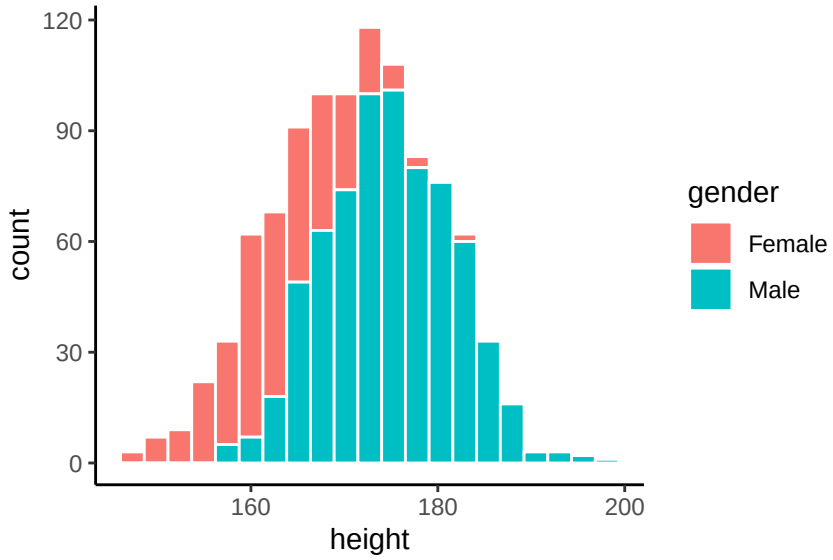
a histogram with dark blue interior and white border, which may help to distinguish between consecutive bars, we would do the following.

```
ggplot(weight_df,
       mapping = aes(x = height)
) + geom_histogram(binwidth = 2.54, colour = 'white', fill = 'darkblue')
```



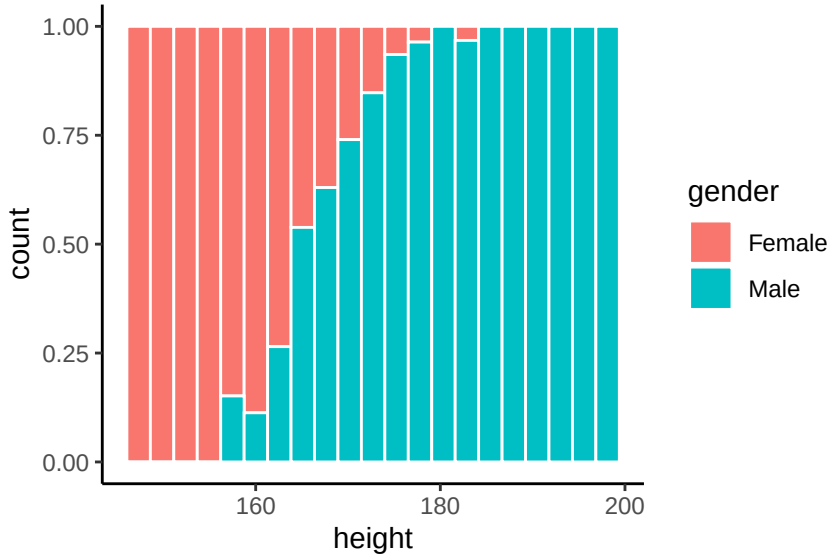
If, in the `aes` mapping, we specify that either `colour` or `fill`, or both, should be mapped some another variable with discrete values, we obtain a *stacked* histogram. In following example, we set the `fill` values to vary by the `gender` variable.

```
ggplot(weight_df,
       mapping = aes(x = height, fill = gender)
) + geom_histogram(binwidth = 2.54, colour = 'white')
```



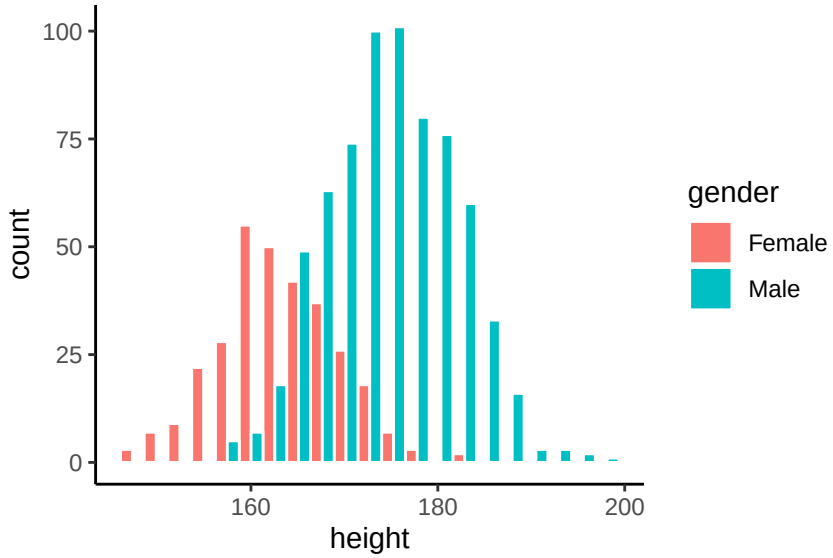
In a stacked histogram, the colour within each bin codes the proportion of that bin's values that correspond to each value of the discrete variable. A variant of the stacked histogram above is where each bar occupies 100% of the plot's height so that what is shown is the proportion of the bin's value corresponding to each value of the grouping variable. We can obtain this type of plot with by setting `position` to `fill` within `geom_histogram` (by default, `geom_histogram` has `position = stack`), as in the following example.

```
ggplot(weight_df,
       mapping = aes(x = height, fill = gender)
) + geom_histogram(binwidth = 2.54, colour = 'white', position = 'fill')
```



If, rather than a stacked histogram, we want two separate histograms, one for males and another for females, we can use other options. One option is to specify `position = 'dodge'` within `geom_histogram` as follows.

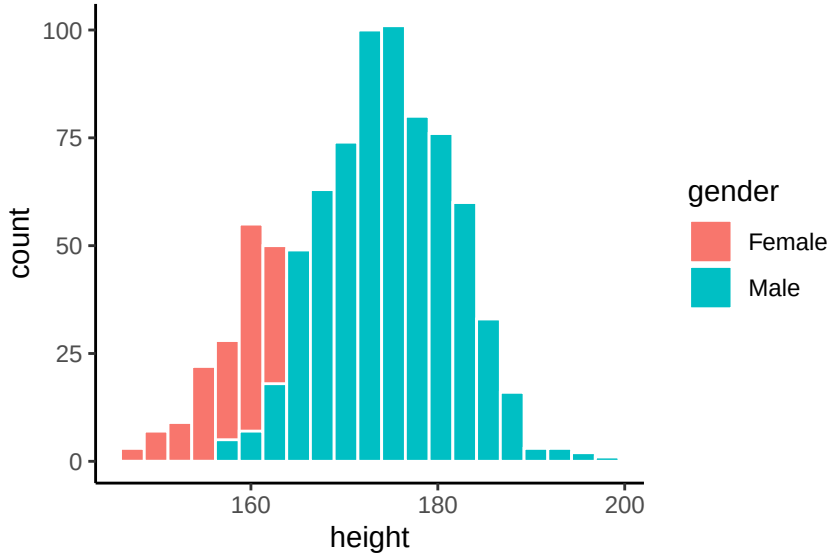
```
ggplot(weight_df,
       mapping = aes(x = height, fill = gender)
) + geom_histogram(binwidth = 2.54, colour = 'white', position = 'dodge')
```



This option uses the same bins as before but puts two non-overlapping bars at each bin, one for males and

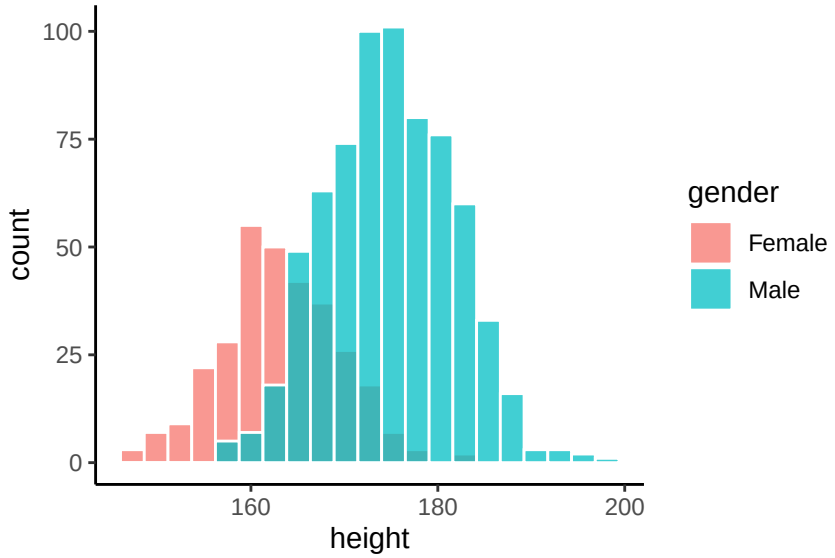
one for females. An alternative option is to place the bars corresponding to males and females at the exact same location by using `position = 'identity'` within `geom_histogram` as follows.

```
ggplot(weight_df,
       mapping = aes(x = height, fill = gender)
) + geom_histogram(binwidth = 2.54, colour = 'white', position = 'identity')
```



Notice that when using this `position = 'identity'` option, the second group, which in this case corresponds to the males, occludes the first one. We can avoid complete occlusion by setting the `alpha`, or opacity, level of the bars to be a value less than 1.0 as in the following example.

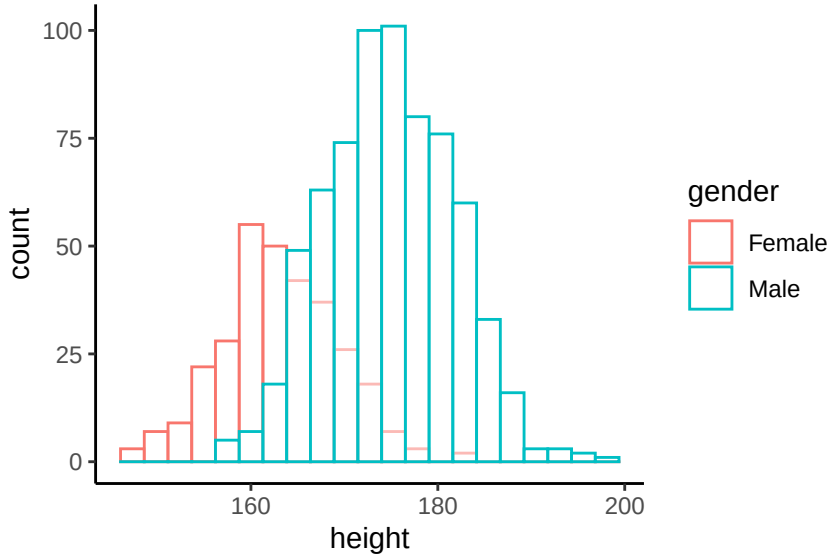
```
ggplot(weight_df,
       mapping = aes(x = height, fill = gender)
) + geom_histogram(binwidth = 2.54, colour = 'white', position = 'identity', alpha = 0.75)
```



Another option to deal with occlusion is to map `gender` to `colour` rather than `fill`, and set `fill` to be white.

```
ggplot(weight_df,
```

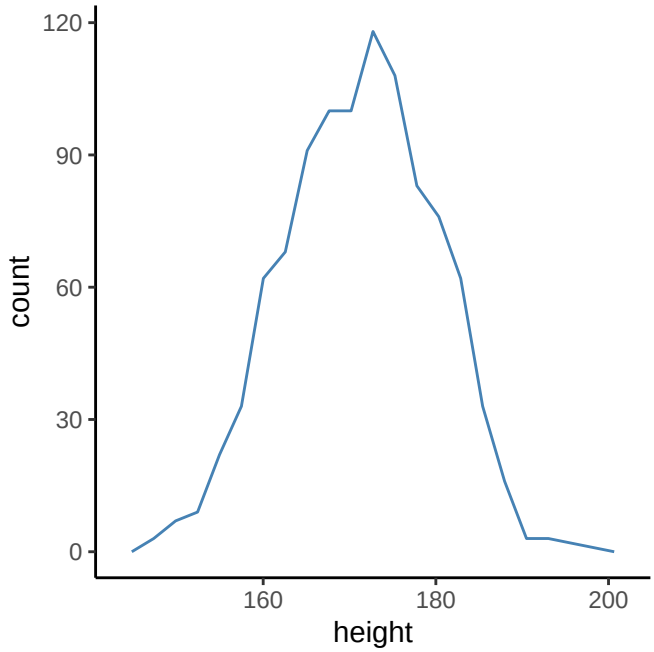
```
      mapping = aes(x = height, colour = gender)
) + geom_histogram(binwidth = 2.54, fill = 'white', position = 'identity', alpha= 0.5)
```



Frequency polygons

A frequency polygon is similar to a histogram but instead of using bars to display the number of values in each bin, it uses connected lines. The following plot displays the number of `height` values in each bin of width 2.54cm.

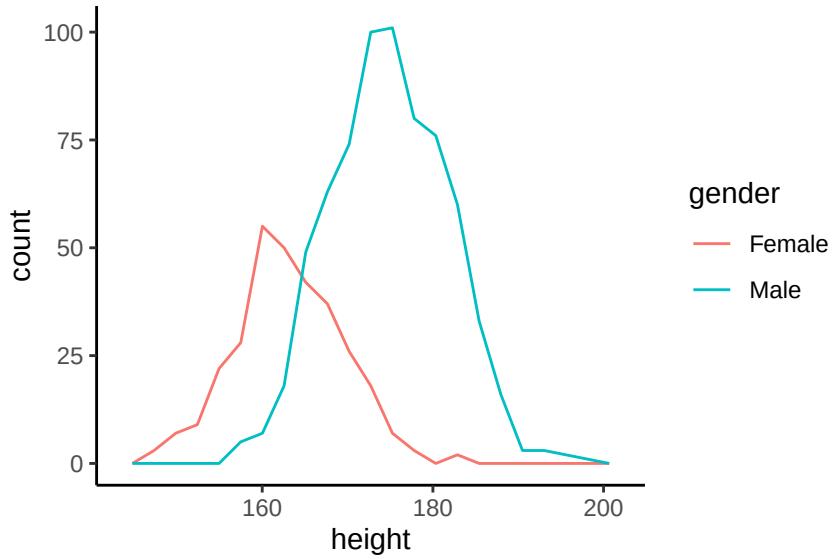
```
ggplot(weight_df,
      mapping = aes(x = height)
) + geom_freqpoly(binwidth = 2.54, colour = 'steelblue')
```



If we map `gender` to `colour`, we produce two overlaid lines. This is the frequency polygon equivalent of the

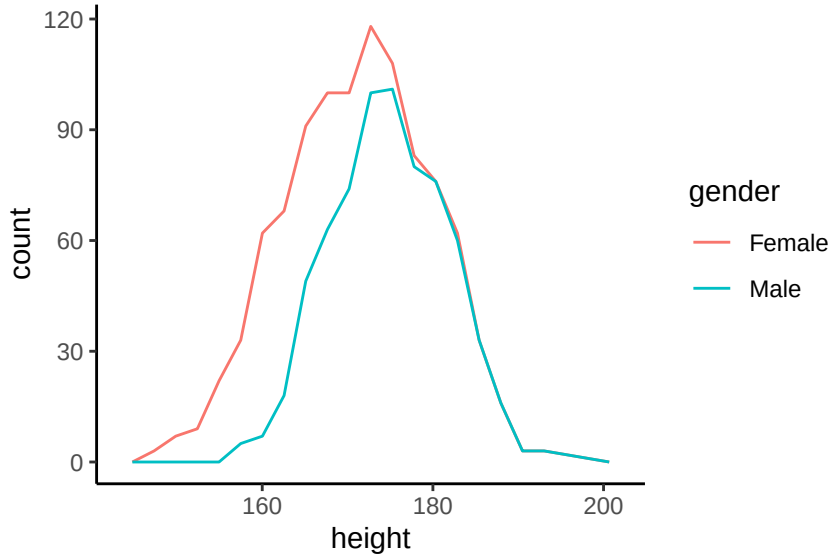
histogram using `position = 'identity'`.

```
ggplot(weight_df,
       mapping = aes(x = height, colour = gender)
) + geom_freqpoly(binwidth = 2.54) # `position = 'identity'` is the default
```



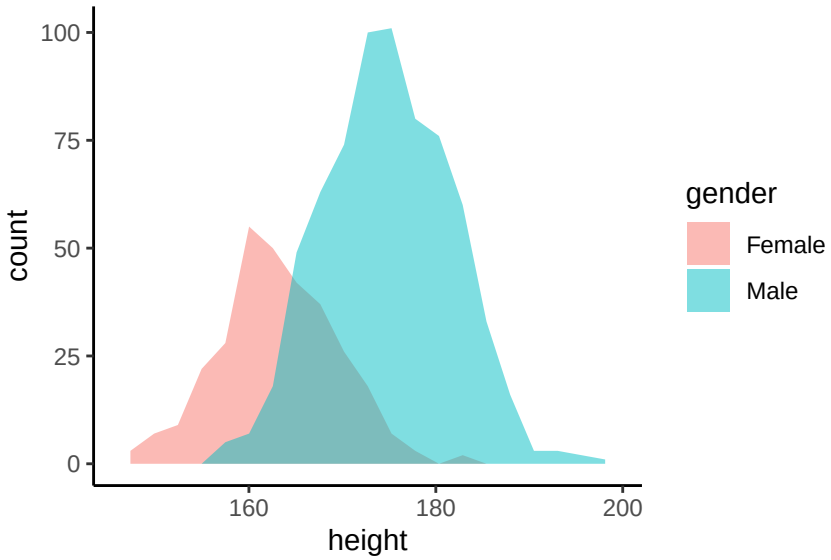
If we prefer a *stacked* frequency polygon, we use `position = 'stack'`.

```
ggplot(weight_df,
       mapping = aes(x = height, colour = gender)
) + geom_freqpoly(binwidth = 2.54, position = 'stack')
```



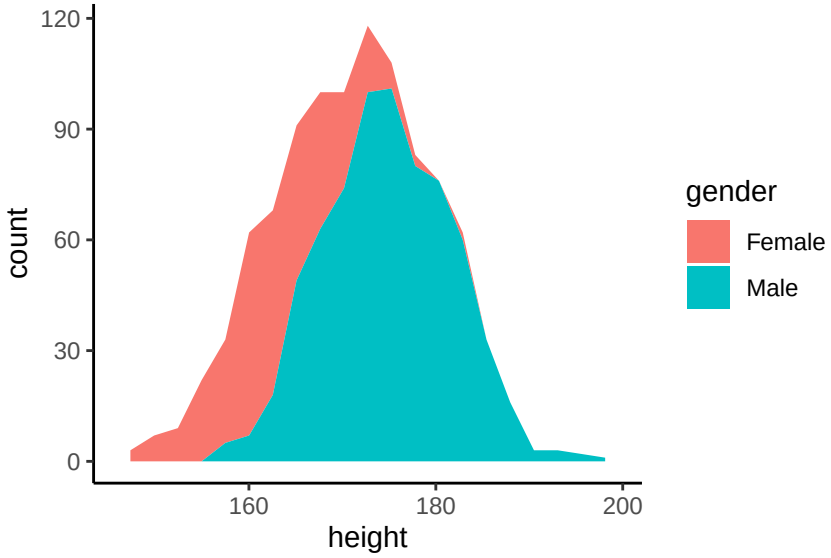
A related option is an *area plot*. To get the equivalent of the overlaid histograms, but will filled interiors, instead of using `geom_freqpoly`, we can use `geom_area`.

```
ggplot(weight_df,
       mapping = aes(x = height, fill = gender)
) + geom_area(binwidth = 2.54, stat = "bin", position = 'identity', alpha = 0.5)
```



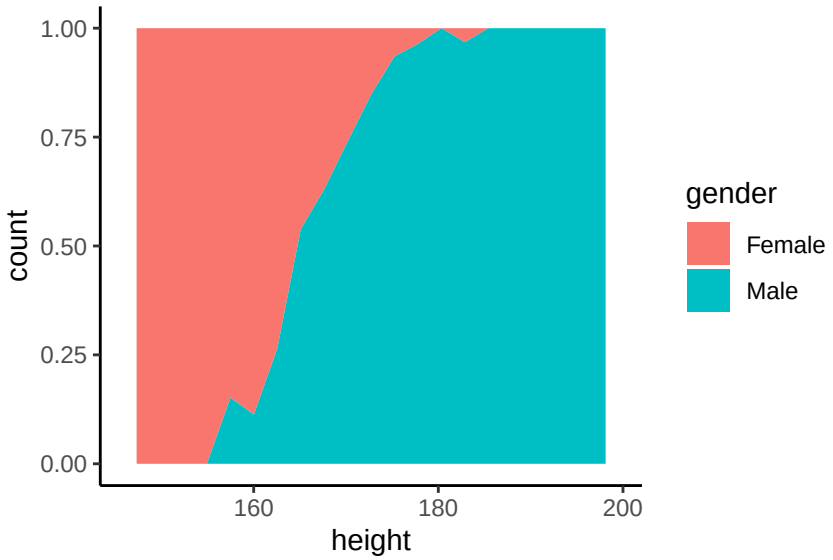
To obtained a stacked area plot, change `position` to have the value of `'stack'`.

```
ggplot(weight_df,
       mapping = aes(x = height, fill = gender)
) + geom_area(binwidth = 2.54, stat = "bin", position = 'stack')
```



We also have the option of setting `position` to have the value of `'fill'`.

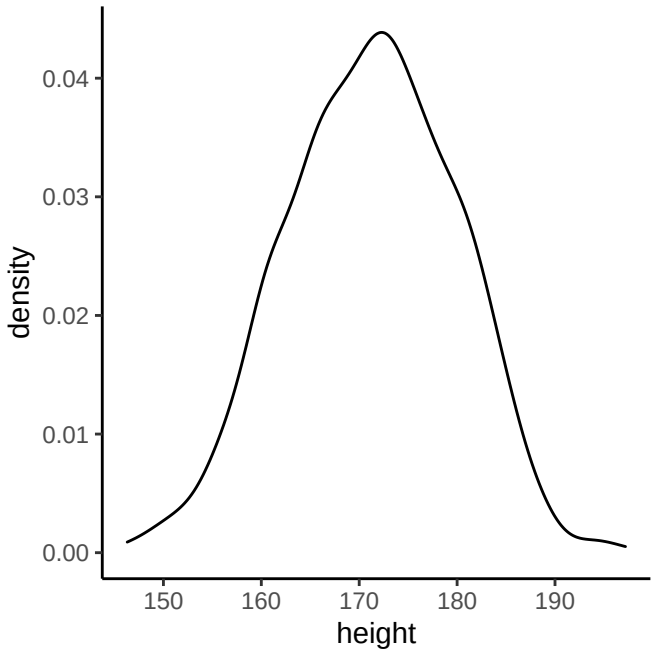
```
ggplot(weight_df,
       mapping = aes(x = height, fill = gender)
) + geom_area(binwidth = 2.54, stat = "bin", position = 'fill')
```



Density plots

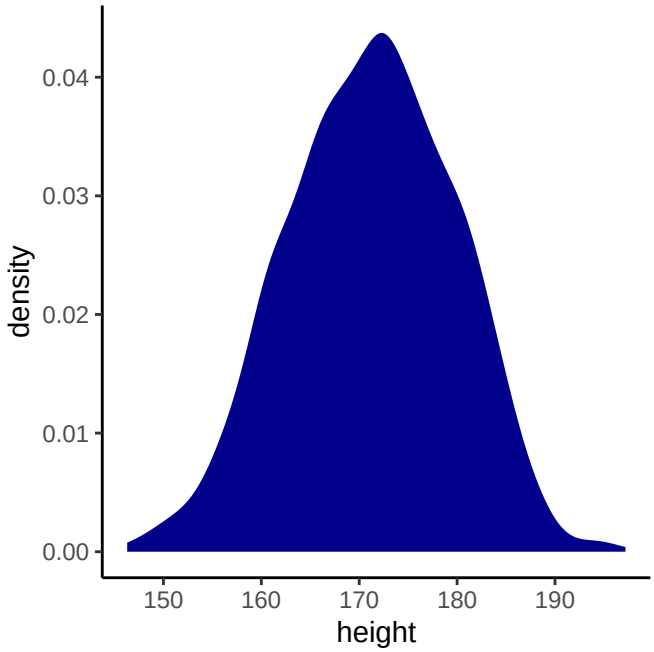
Histograms work by dividing the range of values of a variable into equal sized bins and simply counting the number of values in each bin. *Density plots*, on the other hand, uses *kernel density estimation* to estimate a probability density over the variable. In practical terms, we can see this as moving average smoothing of histograms. The default density plot of the `height` variable is obtained as follows.

```
ggplot(weight_df,
       mapping = aes(x = height)
) + geom_density()
```



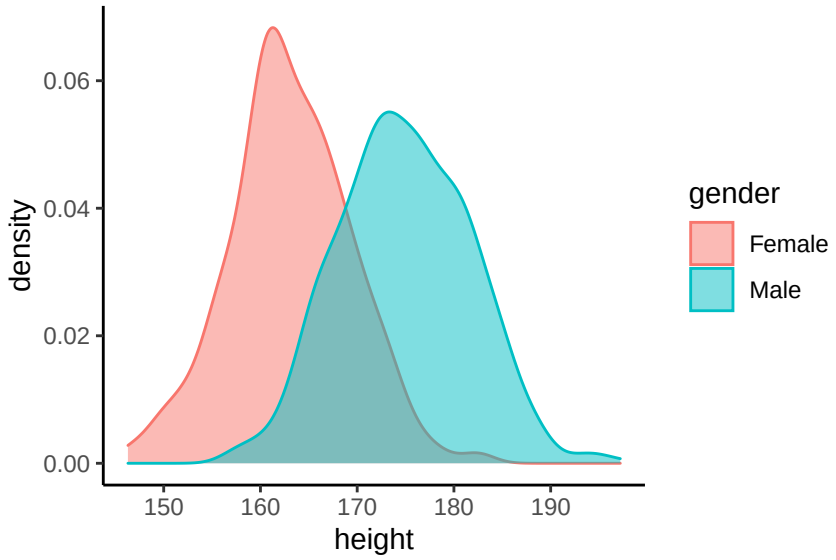
As with the histogram, we may change the colour of the border and the interior of the density plot with `colour` and `fill`, respectively.

```
ggplot(weight_df,
       mapping = aes(x = height)
) + geom_density(colour = 'white', fill = 'darkblue')
```



If we set `fill` or `colour`, or both, to map to `gender`, then the default result is the density plot equivalent of overlaid histograms above. As seen in the following example, we avoid complete occulusion of one density plot by the other by use of the `alpha` parameter.

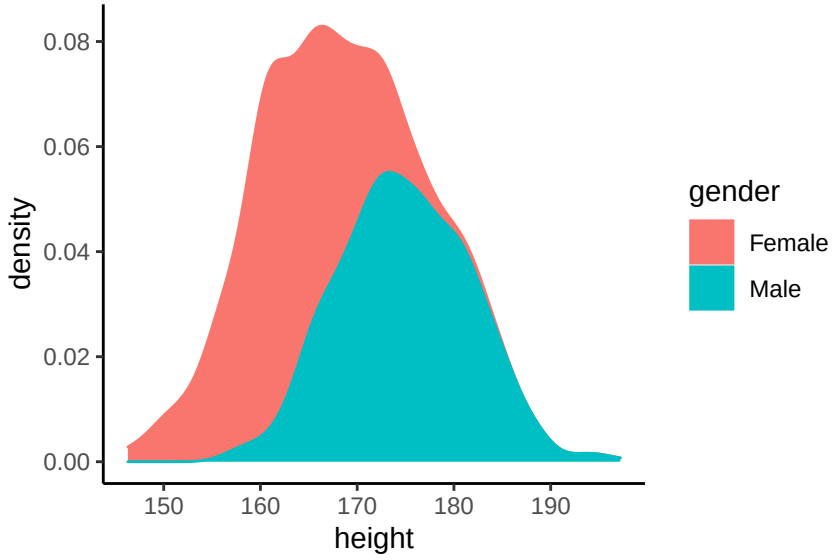
```
ggplot(weight_df,
       mapping = aes(x = height, fill = gender, colour = gender)
) + geom_density(alpha = 0.5)
```



To accomplish the density plot equivalent of the stacked histogram, we set `position = 'stack'`.

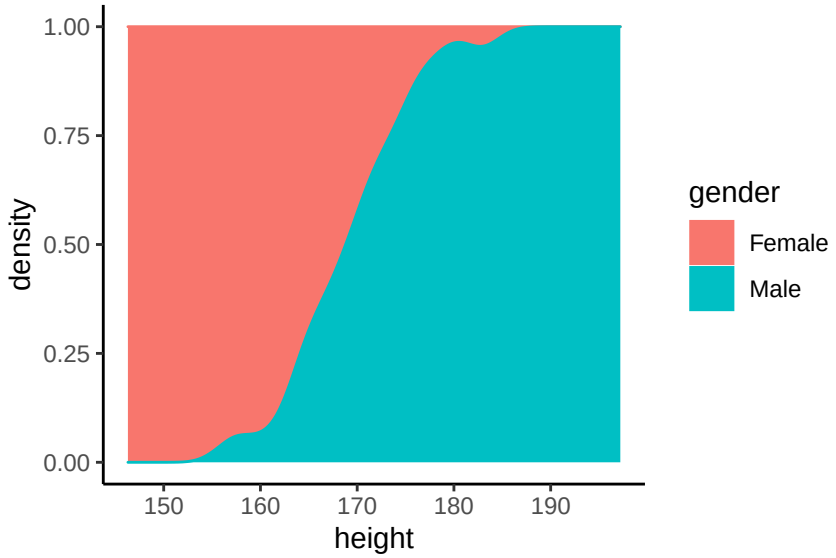
```
ggplot(weight_df,
```

```
       mapping = aes(x = height, fill = gender, colour = gender)
) + geom_density(position = 'stack')
```



The density plot equivalent of the filled stack histogram is accomplished with `position = 'fill'`.

```
ggplot(weight_df,
       mapping = aes(x = height, fill = gender, colour = gender)
) + geom_density(position = 'fill')
```



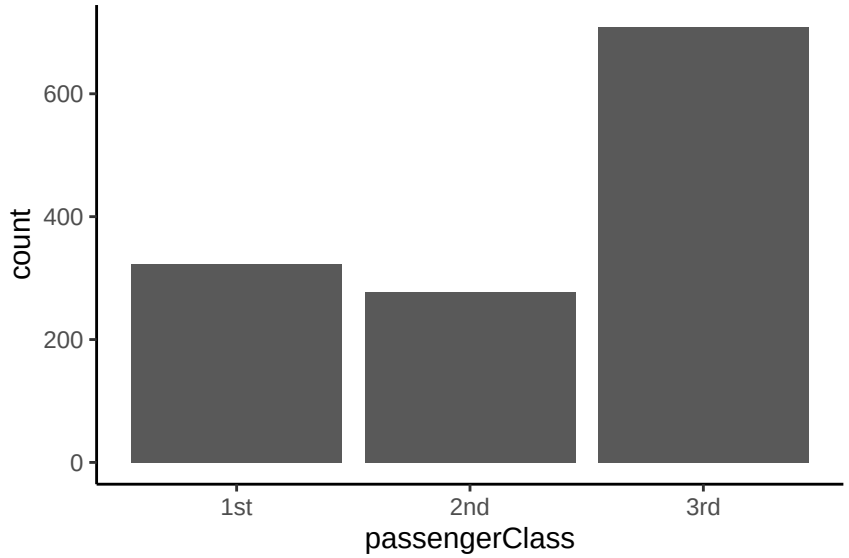
Barplots

Both histograms and density plots assume that the variables are continuous. For discrete variables, their counterpart is a *bar plot*. For each value of the discrete variable, the bar plot displays the number of observed instances of that value in the data. As example, the following `titanic` data frame tells us, for each of the 1309 passengers on the *RMS Titanic* on its fateful maiden voyage in 1912, their sex, age, and passenger class, and whether they survived or not.

```
titanic_df <- read_csv('data/TitanicSurvival.csv') %>% select(-X1)
glimpse(titanic_df)
#> Rows: 1,309
#> Columns: 4
#> $ survived       <chr> "yes", "yes", "no", "no", "no", "yes", "yes", "no", ...
#> $ sex            <chr> "female", "male", "female", "male", "female", "male"...
#> $ age            <dbl> 29.0000, 0.9167, 2.0000, 30.0000, 25.0000, 48.0000, ...
#> $ passengerClass <chr> "1st", "1st", "1st", "1st", "1st", "1st", "1st", "1s...
```

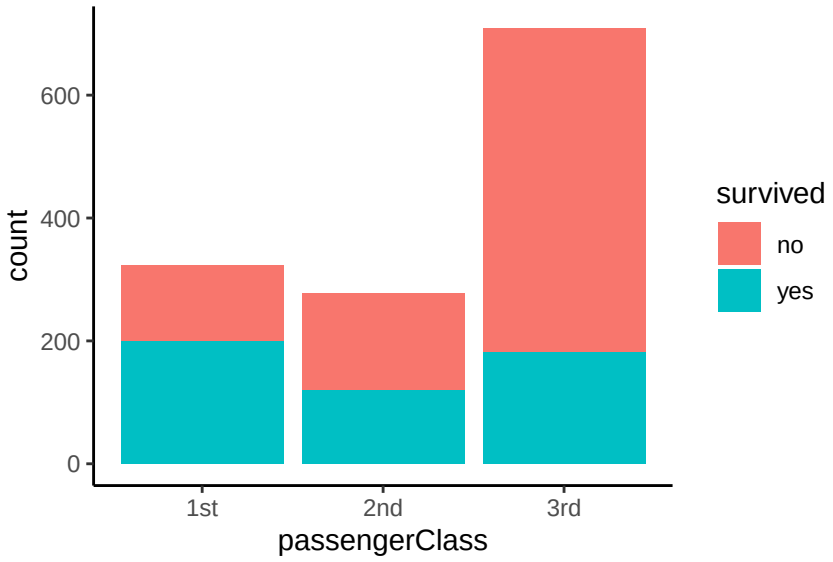
The number of passengers in each passenger class is given by the following bar plot.

```
ggplot(titanic_df,
       mapping = aes(x = passengerClass)
) + geom_bar()
```



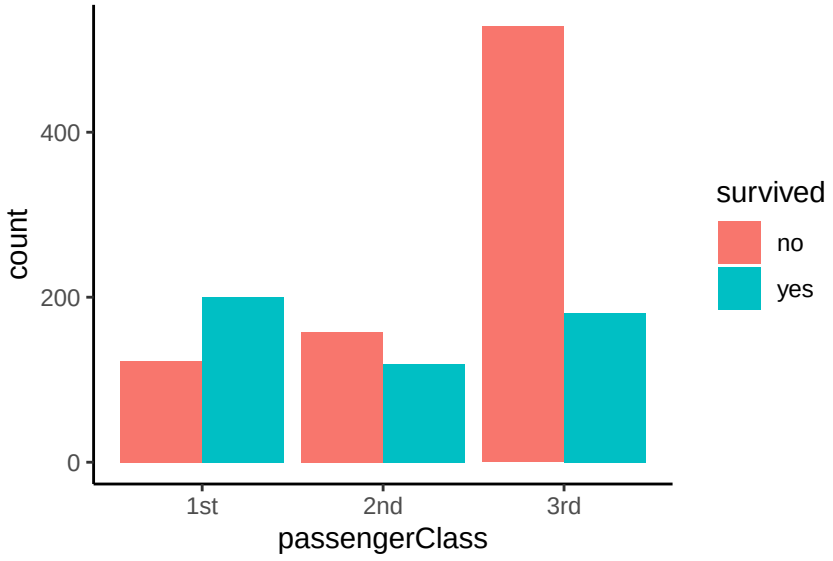
If we map the `fill` property to the variable `survived`, we may see the number of passengers in each class who survived or not.

```
ggplot(titanic_df,
       mapping = aes(x = passengerClass, fill = survived)
) + geom_bar()
```



We may display the same information using side by side bars, one for those who survived and other for those who did not, as follows.

```
ggplot(titanic_df,
       mapping = aes(x = passengerClass, fill = survived)
) + geom_bar(position = 'dodge')
```



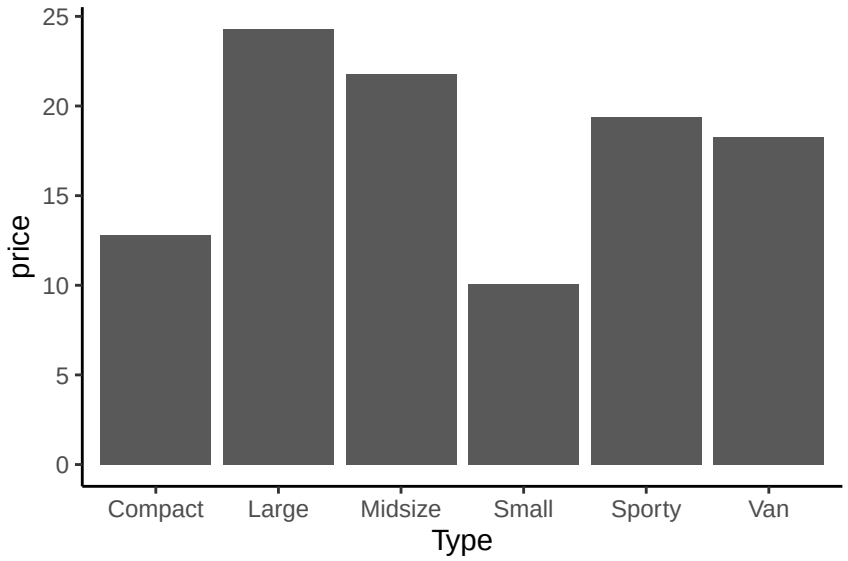
Sometimes, we wish to draw a barplot where the heights of the bars are given by the value of a variable in the data set. Consider the following small data set.

```
carprices <- read_csv('data/carprice.csv') %>%
  group_by(Type) %>%
  summarise(price = mean(Price))
carprices
#> # A tibble: 6 x 2
#>   Type    price
#>   <chr>   <dbl>
#> 1 Compact  12.8
```

```
#> 2 Large    24.3
#> 3 Midsize  21.8
#> 4 Small    10.0
#> 5 Sporty   19.4
#> 6 Van      18.3
```

In this, we have two variables `Type`, which indicates a type of vehicle, and `price`, which gives the average price of each vehicle type. We can plot this data by mapping the `y` attribute to `price` and indicated `stat = 'identity'` in `geom_bar`.

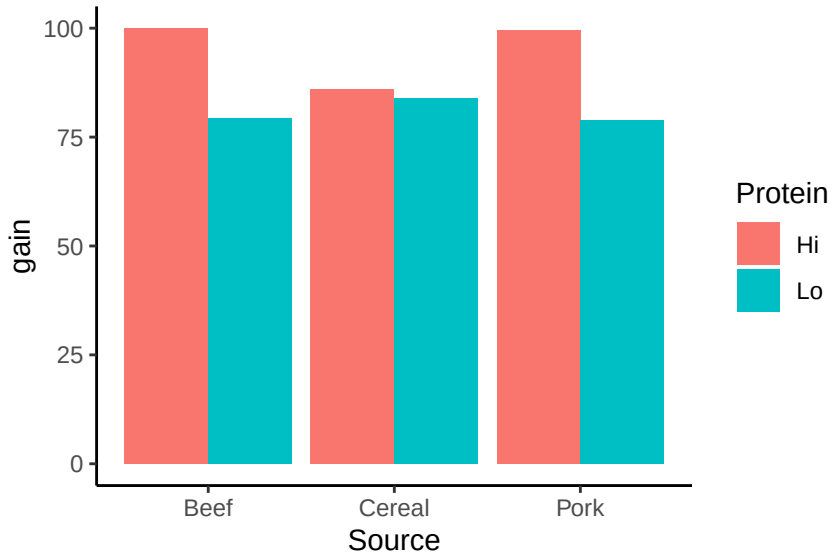
```
ggplot(carprices,
       mapping = aes(x = Type, y = price)
) + geom_bar(stat = 'identity')
```



Mapping the `fill` variable allows us to display values for two discrete variables.

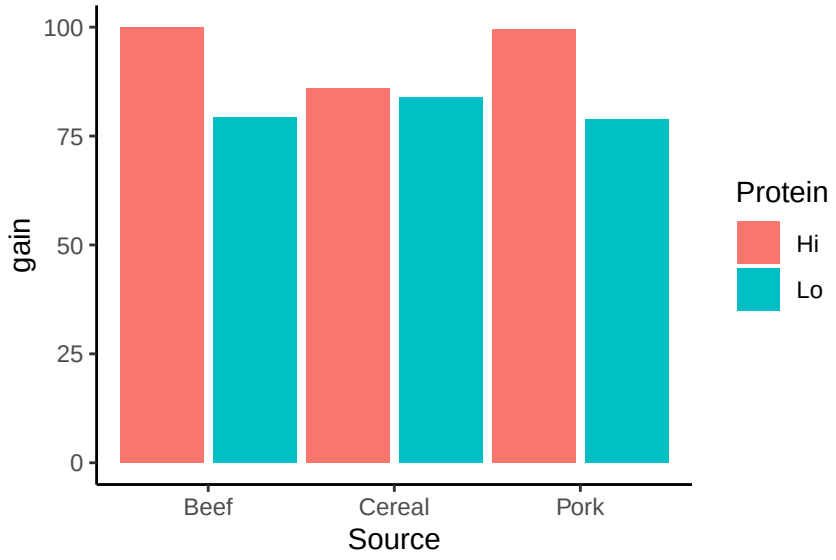
```
fat_rats <- read_csv('data/FatRats.csv') %>%
  group_by(Protein, Source) %>%
  summarize(gain = mean(Gain), se = sd(Gain)/sqrt(n()))

ggplot(fat_rats,
       mapping = aes(x = Source, y = gain, fill = Protein)
) + geom_bar(stat = 'identity', position = 'dodge')
```



In this example, we see that the bars corresponding `Hi` and `Lo` values of `Protein` are touching one another, while there is a gap between the pair of bars corresponding to the `Beef`, `Cereal`, `Pork` values of `Source`. How close the `Hi` and `Lo` bars are is determined by the `width` parameter of `dodge`, which defaults to 0.9 in this case. If we wish to change this parameter, for example, to a width of 1.0, we need to use `position = position_dodge(width = 1.0)` (the statement `position = 'dodge'` is a shortcut to `position = position_dodge()`).

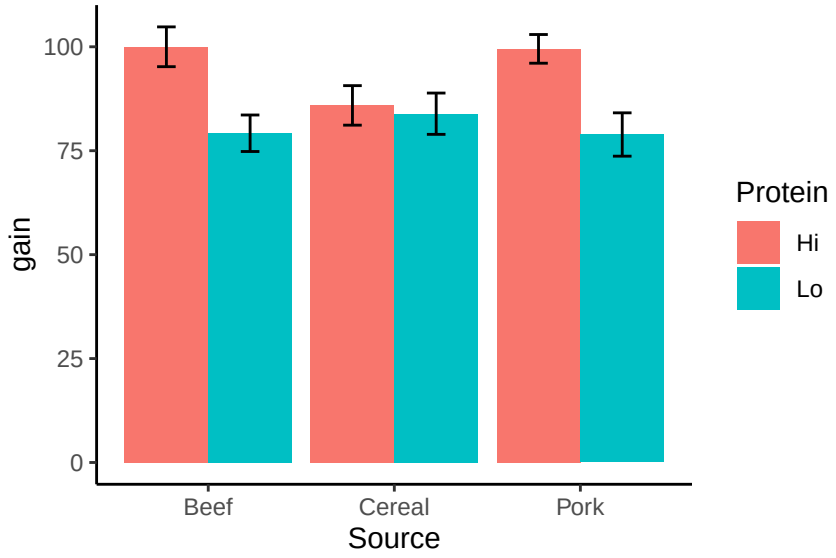
```
ggplot(fat_rats,
       mapping = aes(x = Source, y = gain, fill = Protein)
) + geom_bar(stat = 'identity', position = position_dodge(width = 1.0))
```



Notice that when the `width` of the position dodge is set to 1.0, the distance between all bars in plot is the same. Thus, by setting `width` lower than 1.0, we push the two bars in each pair closer together.

We may add error bars to these bar plots using `geom_errorbar`. With `geom_errorbar`, in order to position the errorbar at the center of each bar, we must set the `width` of the `position_dodge` of the `geom_errorbar` to match the default dodge width of the bars, which is 0.9.

```
ggplot(fat_rats,
       mapping = aes(x = Source, y = gain, fill = Protein, ymin = gain - se, ymax = gain + se)
) + geom_bar(stat = 'identity', position = 'dodge') +
  geom_errorbar(width = 0.2, position = position_dodge(width = 0.9))
```



As a word of warning, although the barplot with standard error error-bars has been and still is widely used in academic paper, it is highly *not* recommended as a means of displaying the distribution of values for different groups or categories. As has been made clear in, for example, Weissgerber et al. (2015), bars with standard errors hide the raw data and tend to conceal and obscure more than reveal, thus going against almost all the guiding principles for data visualization that we mentioned above.

Tukey boxplots

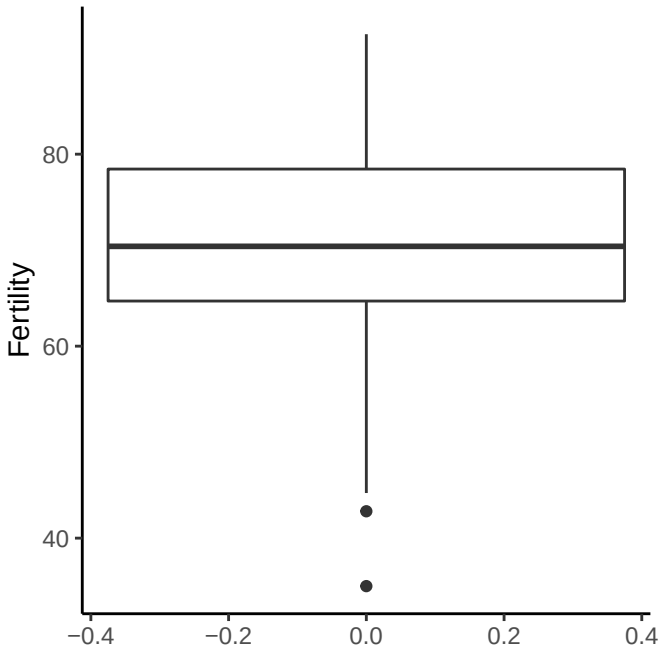
Boxplots, also known as box and whisker plots, are used to display the distribution of values of a variable. One subtype of boxplot is the *Tukey boxplot* (Tukey 1977). These are in fact most common subtype and are the default type implemented in `ggplot2` using the `geom_boxplot` function.

For some of following examples, we'll use the R built-in `swiss` data set used that provides data on fertility rates in 47 Swiss provinces in 1888. One predictor variable, `Examination`, gives the proportion of army draftees in that province who received the highest mark on an army examination. Another, `Catholic`, gives the proportion of province who are Catholic. Because each province either has a clear majority Catholic, or a clear majority of Protestant, we will create a new logical variable, `catholic`, that indicates if the province's Catholic proportion is greater than 0.5 or not. The data frame also has the name of the province as its row name, and we will create a new variable with these names.

```
swiss_df <- swiss %>% rownames_to_column('province') %>%
  mutate(catholic = Catholic > 50)
```

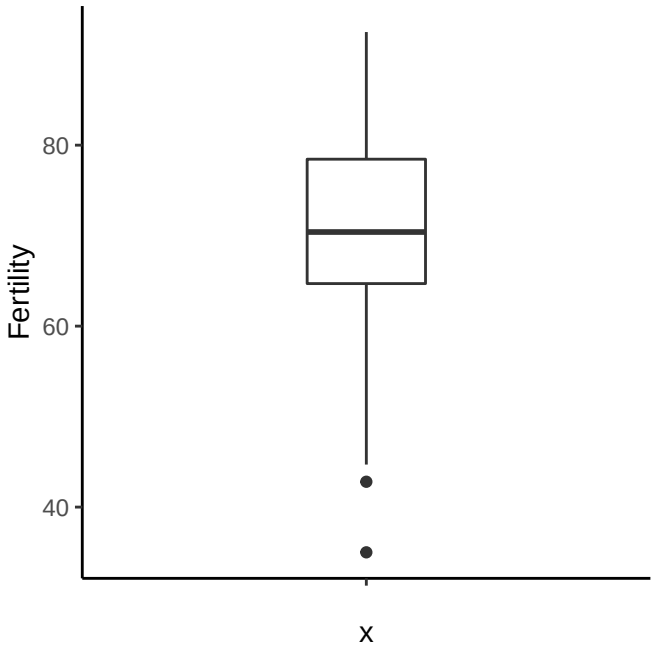
In the following plot, we use a Tukey boxplot to display the distribution of the `Fertility` variable in the `swiss` data set.

```
ggplot(swiss_df,
       mapping = aes(y = Fertility)
) + geom_boxplot()
```



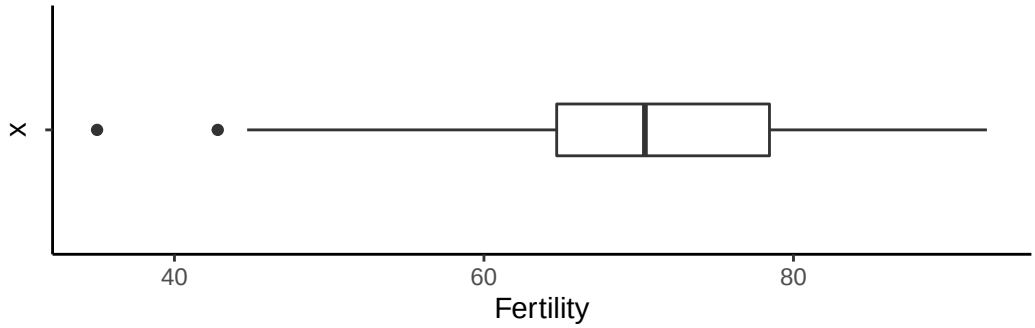
Notice that in this case, where we are displaying the distribution of a single variable only, we only set the `y` attribute in the `aes` mapping. The boxplot extends along the y axis, is centered at 0 on the x-axis and the left and right edges of the box are positioned at $x \approx -0.4$ and $x \approx 0.4$. This default style for a single boxplot can be improved by indicating that the x axis variable is discrete by setting `x = ''` within the `aes` mapping, and then changing the width of the boxplot.

```
ggplot(swiss_df,
       mapping = aes(x = '', y = Fertility)
) + geom_boxplot(width = 0.25)
```



We may convert this vertically extended boxplot to a horizontal one by a `coord_flip()`.

```
ggplot(swiss_df,
       mapping = aes(x = '', y = Fertility)
) + geom_boxplot(width = 0.25) +
  coord_flip()
```

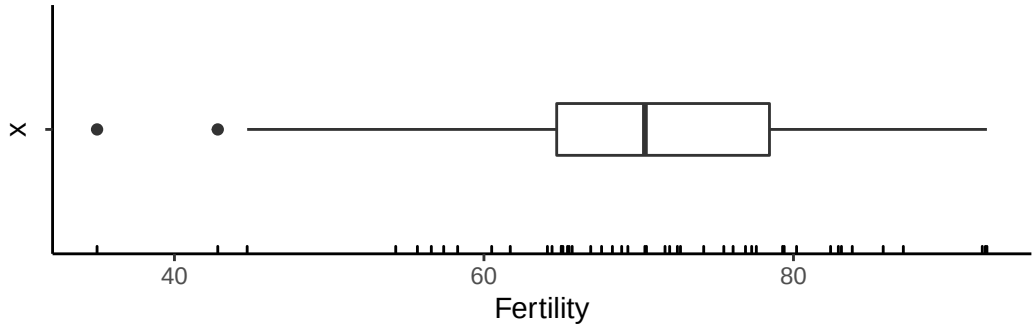


Tukey boxplots are defined as follows:

1. The *box* extends from the 25th to the 75th percentile.
2. The line or band within the box is the median value, which is also the 50th percentile.
3. The *whiskers* extend to the furthest points above the 75th percentile, or below the 25th percentile, that are within 1.5 times the inter-quartile range (the range from the 25th to the 75th percentile).
4. Any points beyond 1.5 times the inter-quartile range above the 75th percentile or below the 25th percentile is represented by a point and is classed as an *outlier*.

From a Tukey boxplot, we obtain robust measures of both central tendency, given by the band within the box, and of the the scale, given by the width of the box. We may also see whether and to what extent the distribution is skewed, both by the relative position of band within the box and the relative lengths of the two whiskers. We may also obtain a sense of kurtosis from the Tukey boxplot, which will describe in more detail below. However, clearly we are not showing all the available data, and visual summaries, however robust and efficient, can always conceal or obscure some important informative. It is generally a good idea, therefore, to supplement the boxplot with visualizations of the individual data points. One possibility is to provide a *rug plot*, which displays the location of every data point using a short line, as we see in the following example.

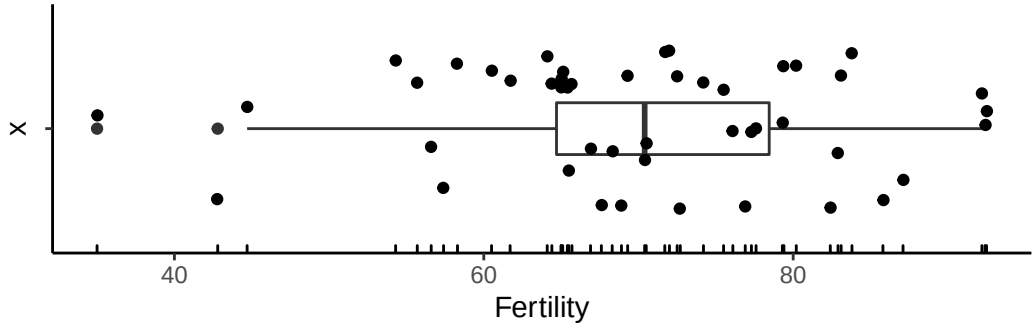
```
ggplot(swiss_df,
       mapping = aes(x = '', y = Fertility)
) + geom_boxplot(width = 0.25) +
  coord_flip() +
  geom_rug(sides ='l')
```



Note that we specify `sides = 'l'` in the `geom_rug` function to indicate that the rug should only be shown on the left axis, which is the bottom when the axis is flipped.

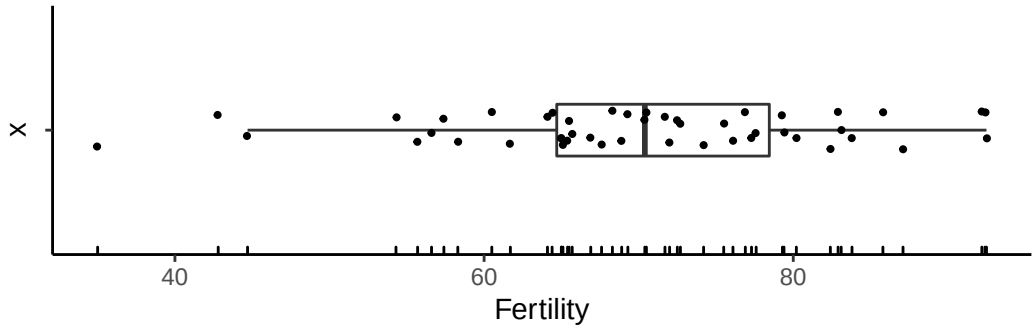
Another option for displaying all the day, which can also be used in combination with the rug plot, is to provide a *jitter* plot as follows.

```
ggplot(swiss_df,
       mapping = aes(x = '', y = Fertility)
) + geom_boxplot(width = 0.25) +
  geom_rug(sides = 'l') +
  geom_jitter() +
  coord_flip()
```



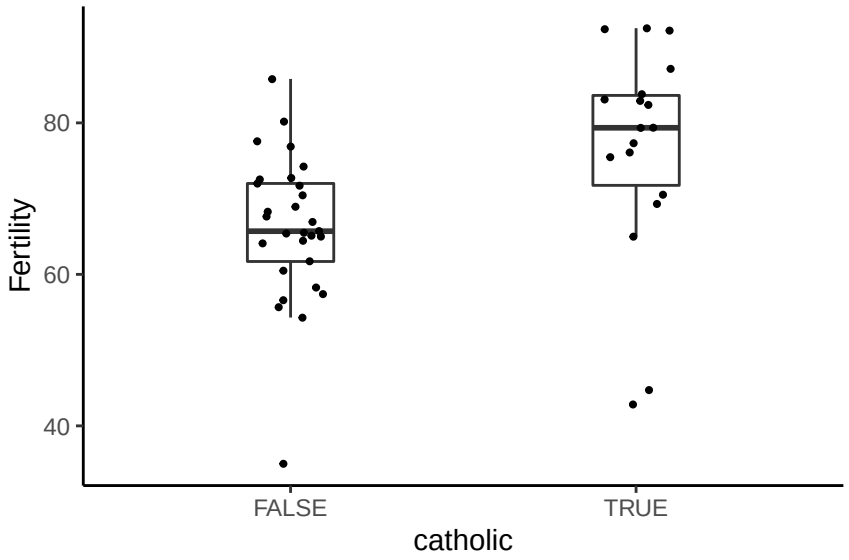
A jitter plot is like a scatterplot but where each point is randomly perturbed, which is useful if points tend to occur in identical or very close locations. The jitter plot just shown, however, may be tidied up by reducing the spread of the points along the vertical axis so that they are within the width of the box on that axis (using the `width` parameter of the `geom_jitter` function), and by reducing their size (using the `size` parameter). In addition, the outlier points no longer need to be displayed because all points will now be displayed by the jitter plot. We not display the outlier points by setting `outlier.shape = NA` within the `geom_boxplot` function.

```
ggplot(swiss_df,
       mapping = aes(x = '', y = Fertility)
) + geom_boxplot(width = 0.25, outlier.shape = NA) +
  geom_rug(sides = 'l') +
  geom_jitter(width = 0.1, size = 0.75) +
  coord_flip()
```



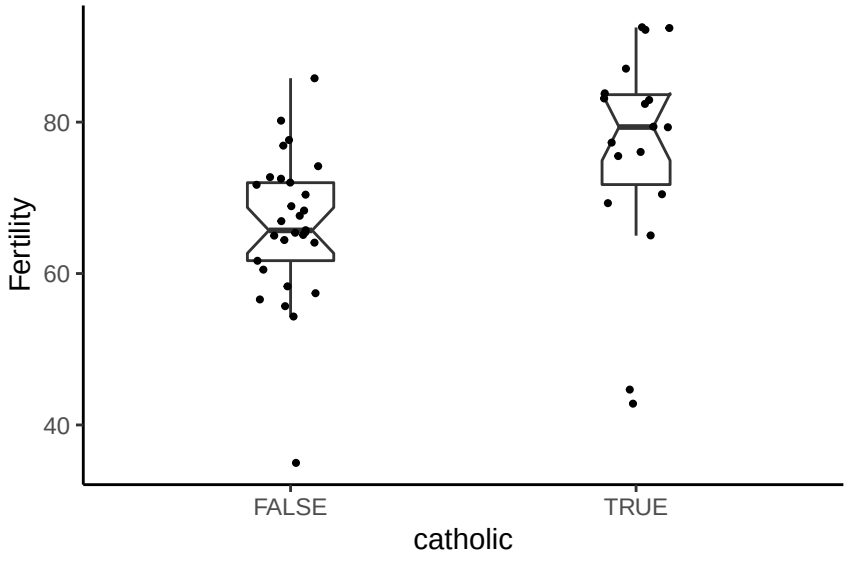
By mapping the `x` property to a third variable, we may display multiple box plots side by side.

```
ggplot(swiss_df,
       mapping = aes(x = catholic, y = Fertility)
) + geom_boxplot(width = 0.25, outlier.shape = NA) +
  geom_jitter(width = 0.1, size = 0.75)
```



When comparing boxplots, one convention is to scale the width of the box with the square root of the sample size, which we may doing by setting `varwidth` to `TRUE`. In addition, we may set `notch` to `TRUE` to provide a measure of the uncertainty concerning the true value of the median (calculated by 1.58 times the inter-quartile range divided by the square root of the sample size).

```
ggplot(swiss_df,
       mapping = aes(x = catholic, y = Fertility)
) + geom_boxplot(width = 0.25, outlier.shape = NA, varwidth = T, notch = T) +
  geom_jitter(width = 0.1, size = 0.75)
```

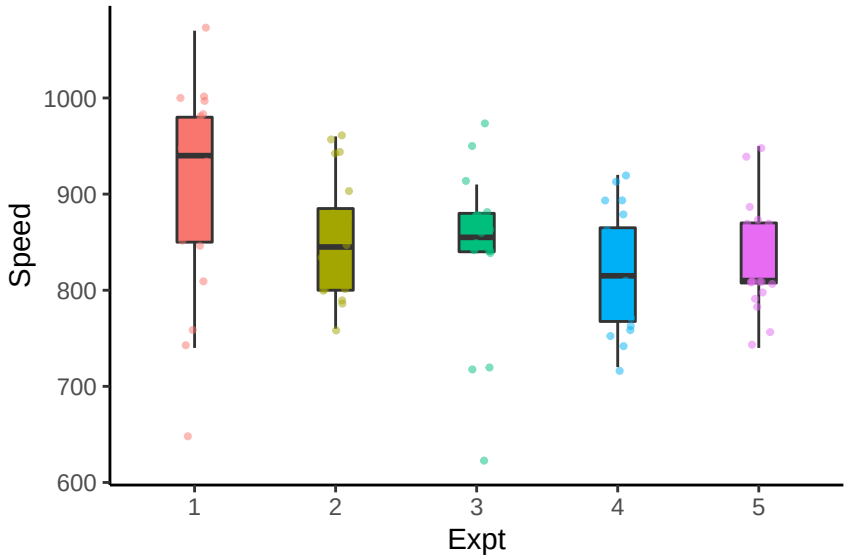


Note, however, that when the median band is close to either the 25th or 75th percentile, the notch may not display properly and so is not generally useful.

It may be helpful to colour code the different boxplots using `colour` or `fill`, or both. In this case, because the legend will be superfluous (given that the boxplots will be labelled by the x-axis), we may remove the legend by setting `legend.position = 'none'` in the `theme` function.

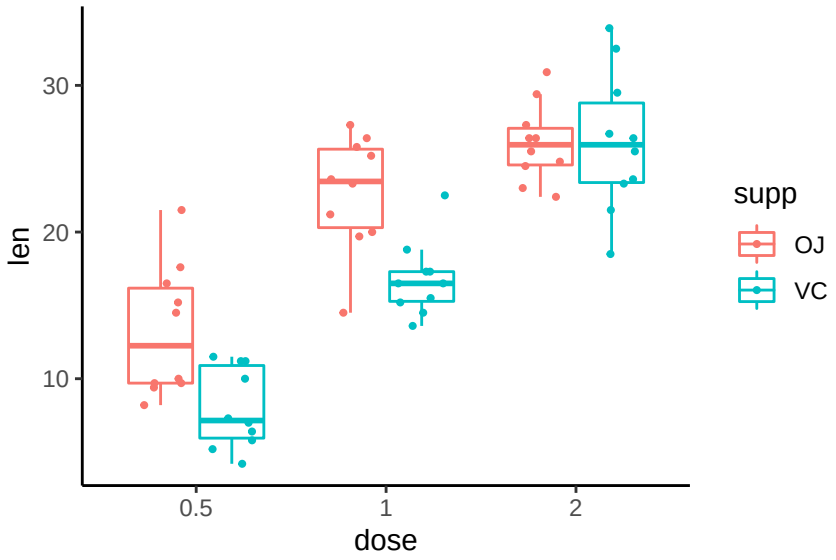
```
morley %>%
```

```
  mutate(Expt = as.factor(Expt)) %>%
  ggplot(
    mapping = aes(x = Expt, y = Speed, fill = Expt)
) + geom_boxplot(width = 0.25, outlier.shape = NA, varwidth = T) +
  geom_jitter(aes(colour = Expt), alpha = 0.5, width = 0.1, size = 0.75) +
  theme(legend.position = 'none')
```



Finally, we may use another variable, other than that used for the x axis, as the `colour` variable, thus displaying boxplots for each combination of the values of the two grouping variables. For example, in the following plot, we display one boxplot for the length (`len`) of tooth growth for each combination of the three levels of a `dose` variable and two different methods of supply `supp`. In this case, if we want to use jitters in addition to the boxplots, we need to position the jitters using `position_jitterdodge`.

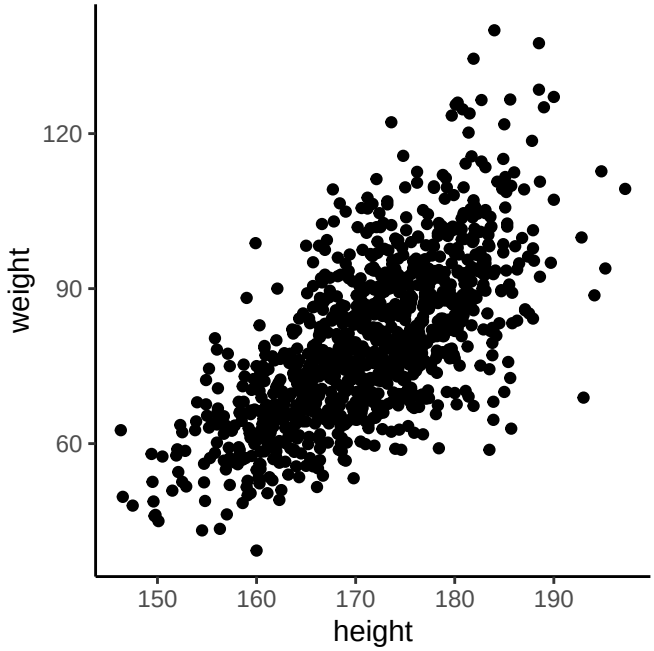
```
ToothGrowth %>%
  mutate(dose = as.factor(dose)) %>%
  ggplot(mapping = aes(x = dose, y = len, colour = supp)
) + geom_boxplot(outlier.shape = NA, varwidth = T) +
  geom_jitter(position = position_jitterdodge(0.5), size = 0.75)
```



Scatterplots

We've already seen some simple scatterplots. Here, we'll provide more in depth coverage using the `weight_df` data frame. The following code will display a scatterplot of `weight` as a function of `height`.

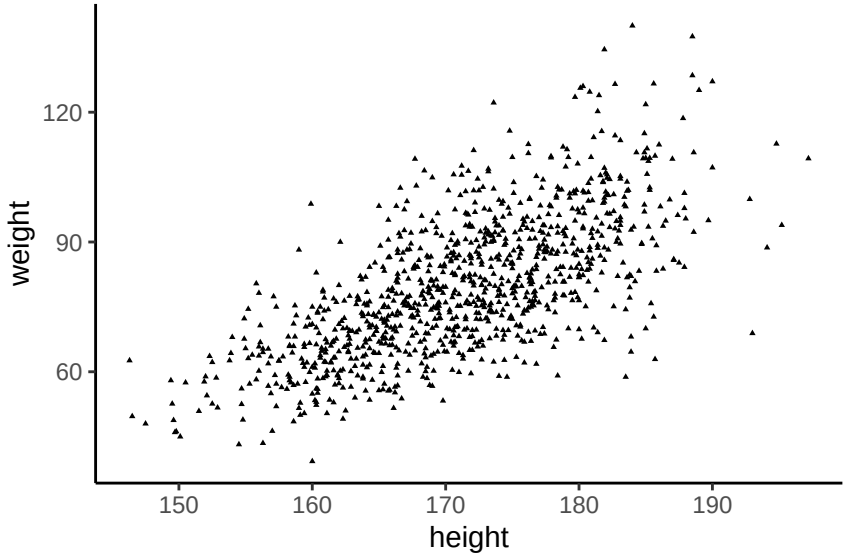
```
ggplot(weight_df,
       mapping = aes(x=height, y=weight)
) + geom_point()
```



In this example, the values of the `height`, and `weight` determine the `x` and `y` coordinates of the points. Other attributes of the points, like their shape, size, etc., are not determined by the data, and so are set to their default values, which in this case corresponds to filled dots of unit size. In general, however, we always change these attributes that are not determined by the data from their default values by specifying `size`, `shape`,

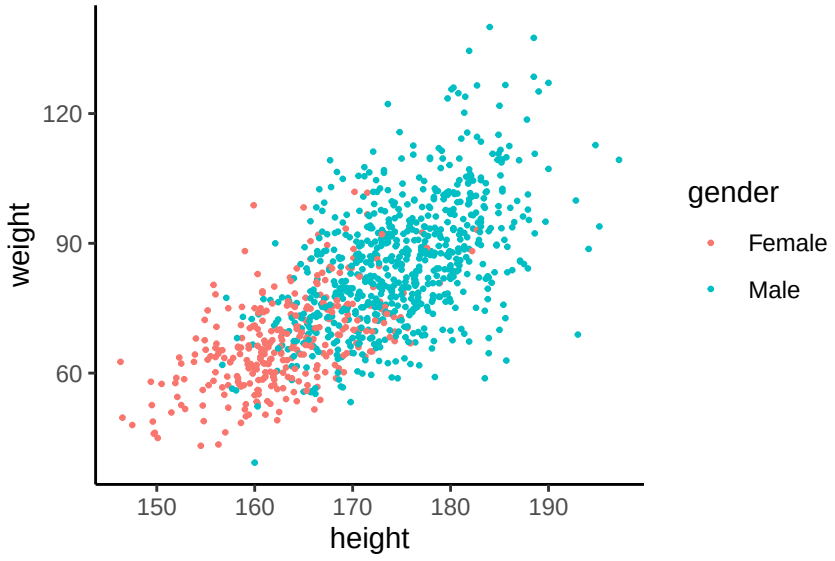
etc., values within the geoms. For example, we change the previous plots to have points that are 0.5 the size of the original and are triangles as follows.

```
ggplot(weight_df,
       mapping = aes(x=height, y=weight)
) + geom_point(size = 0.5, shape='triangle')
```



As in histograms, density plots, bar plots, etc., as we've seen, scatterplots can also have the colour of the points determined by another variable. The following example, we colour code the points according to whether the observation corresponds to a male or a female.

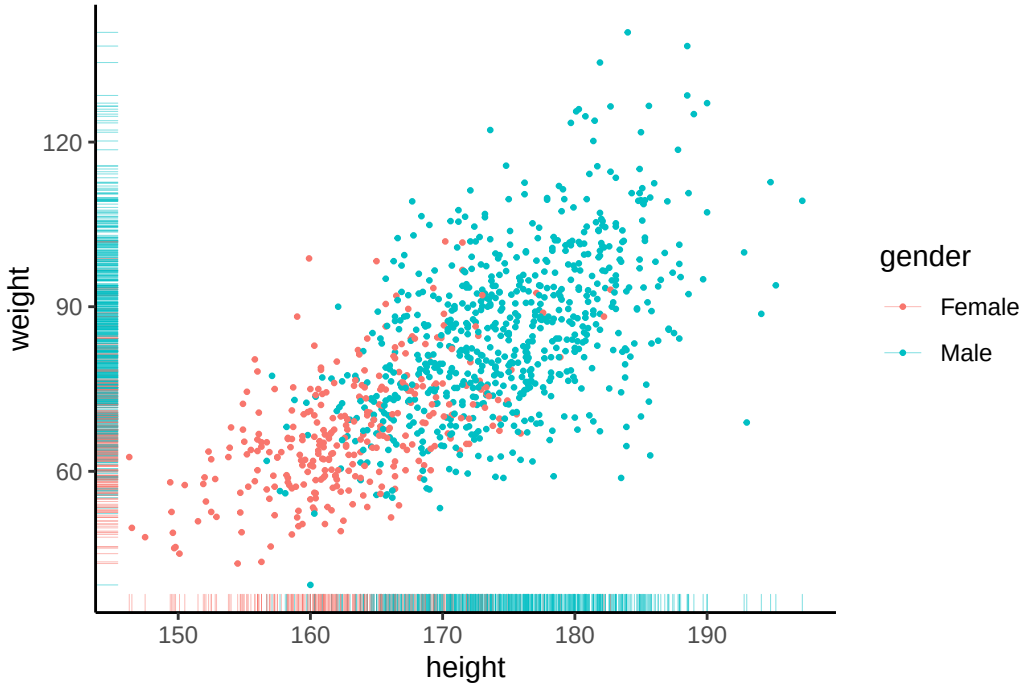
```
ggplot(weight_df,
       mapping = aes(x=height, y=weight, colour = gender)
) + geom_point(size = 0.5)
```



Adding marginal distributions

Scatterplots display pairs of values as points in two dimensional space. It is often useful to also simultaneously see the distribution of the values over the individual variables. There are a few different options to do so. A common choice is to use a rug plot, like we saw above in the case of boxplots, which displays values on each dimension as short lines. In the following example, we'll change the `alpha` transparency value and the line width (using `size`) to make the rug plot less cluttered.

```
ggplot(weight_df,
       mapping = aes(x=height, y=weight, colour=gender)
) +
  geom_point(size = 0.5) +
  geom_rug(alpha = 0.5, size = 1/10)
```



A wider set of options for marginal distributions in scatterplots is available with the `ggMarginal` function that is part of the `ggExtra` package.

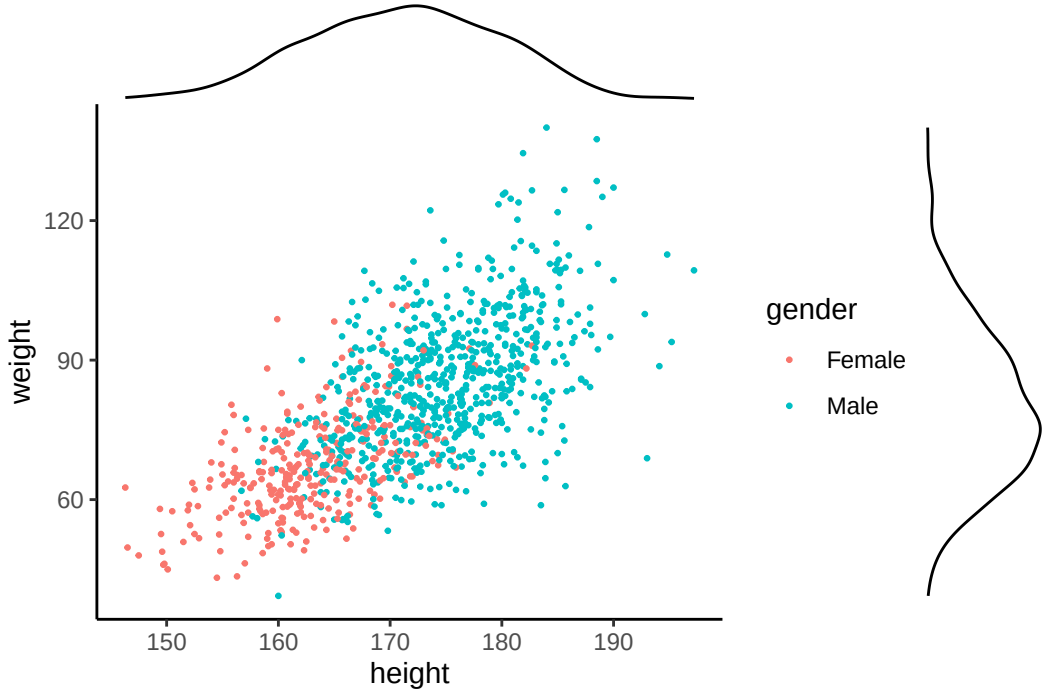
```
library("ggExtra")
```

To use `ggMarginal`, we need to first assign the plot object to a named variable such as `p`.

```
p <- ggplot(weight_df,
       mapping = aes(x=height, y=weight, colour=gender)
) + geom_point(size = 0.5)
```

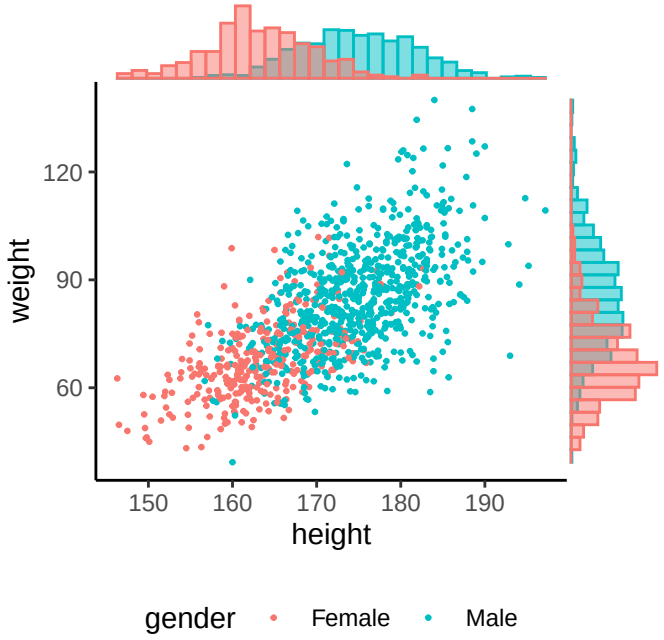
Notice that in this case nothing is displayed. The `ggplot` function creates the scatterplot as before, but assigns it to `p` rather than displaying it. We may now display the scatterplot with its marginal distributions as follows.

```
ggMarginal(p)
```



As can be seen, by default, a density function for the `height` and `weight` variables are displayed along the top and right sides, respectively. In addition, by default, the grouping colour is not shown. We can override these defaults. For example, to produce a marginal histogram, rather than a density plot, that uses the `aes` colour mapping, we may do the following.

```
p <- p + theme(legend.position = 'bottom')
ggMarginal(p, type = 'histogram', groupColour = T, groupFill = T,
           position = 'identity', alpha = 0.5)
```



Notice also that in this plot, we moved the legend that displayed the `aes` colour mapping to the botttom of

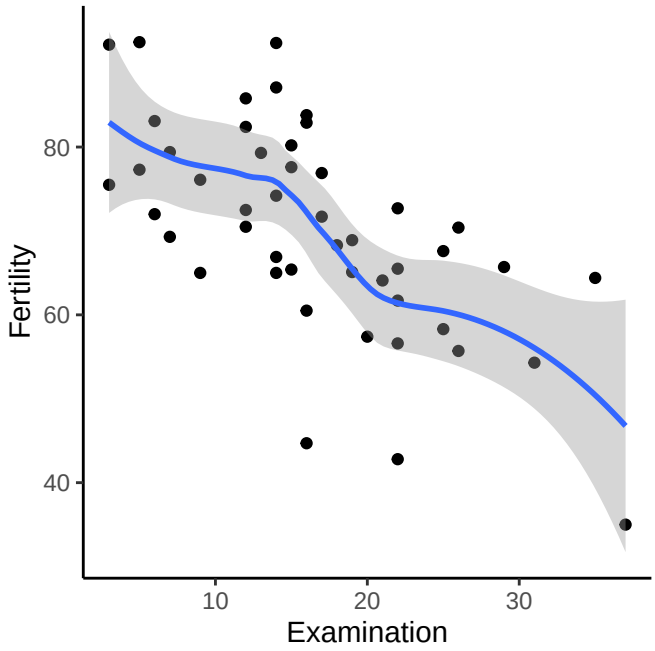
the plot using the `theme()` function.

Adding a smoothing function

A *loess* (locally estimated scatterplot smoothing) smoother aims to capture the trends in a scatterplot by fitting locally weighted regression function, usually a low degree polynomial, at evenly spaced values along the range of values of the predictor or explanatory variable. Its local weighting feature weights data points closer the central point more than those further away. The loess smoother is the default smoother for use with scatterplots and can be obtained using `geom_smooth()`.

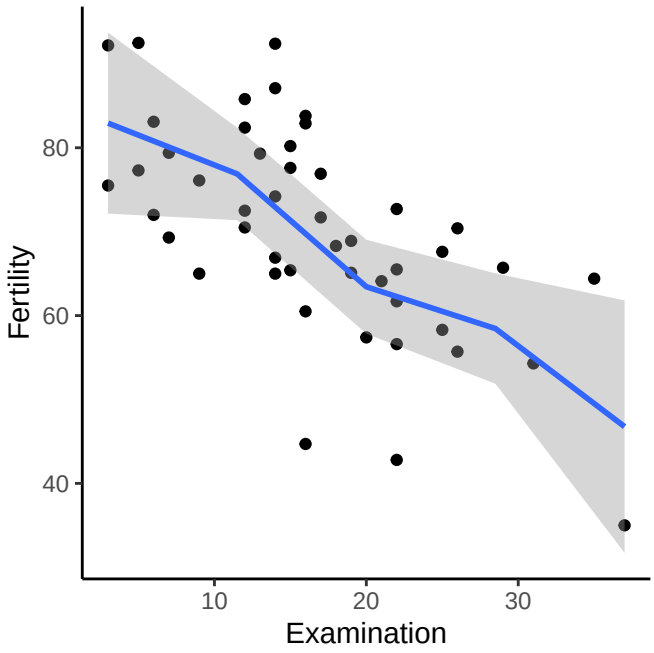
In the following plot, we display a scatter plot of the fertility rate against the `Examination` score of each province.

```
ggplot(swiss_df,
       mapping = aes(x = Examination, y = Fertility)
) + geom_point() + geom_smooth()
```



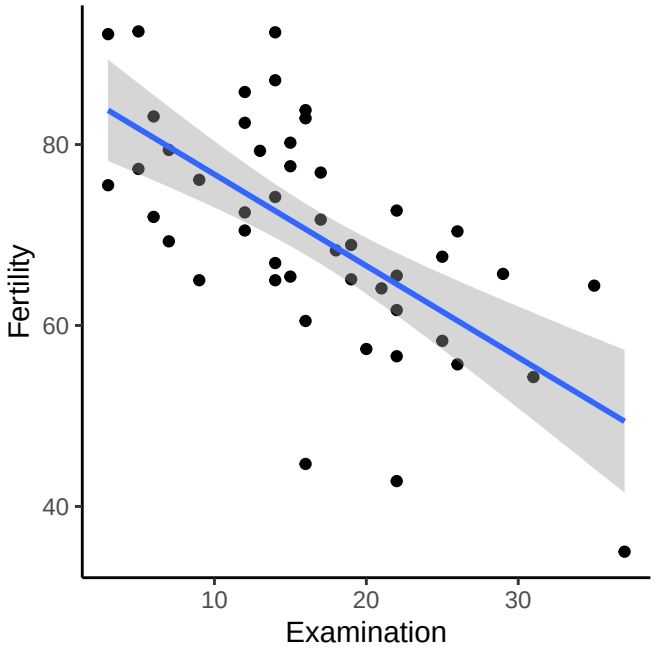
By default, the loess smoother uses 80 evenly spaced points in the range of the predictor variable, and uses a linear regression at each point. We can change the number of evaluation points and the regression function to use specifying the value of `n` and `formula`, respectively in `geom_smooth`. For example, in the following example, we evaluate linear regression at 5 points.

```
ggplot(swiss_df,
       mapping = aes(x = Examination, y = Fertility)
) + geom_point() + geom_smooth(n = 5, formula = y ~ x)
```



We can produce a linear fit to the scatterplot by setting `method = 'lm'` in `geom_smooth()` as in the following plot.

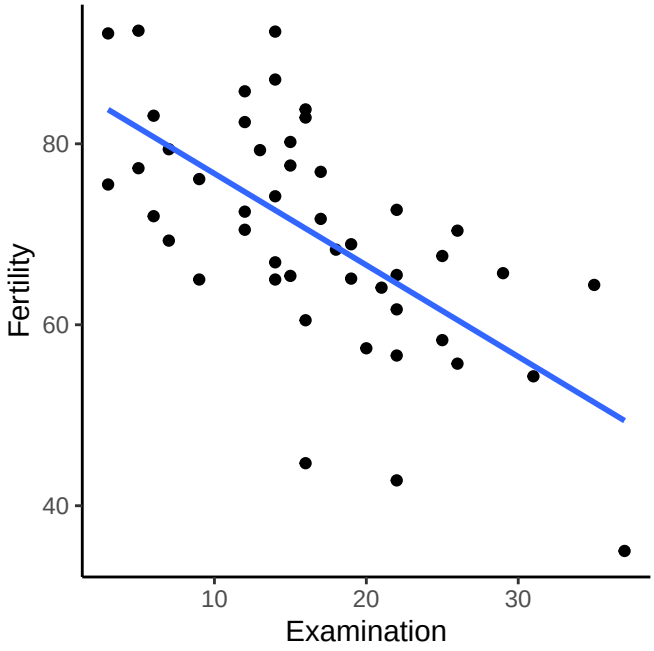
```
ggplot(swiss_df,
       mapping = aes(x = Examination, y = Fertility)
) + geom_point() + geom_smooth(method = 'lm')
```



This will display the least squares line of best fit that we would obtain from a standard linear regression analysis. Note that we may turn off the standard error shading by setting `se = F`.

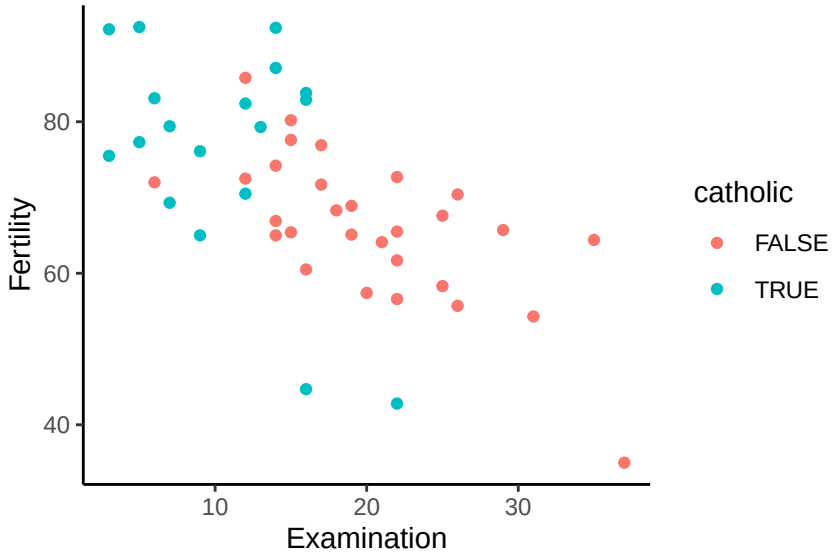
```
ggplot(swiss_df,
       mapping = aes(x = Examination, y = Fertility)
```

```
) + geom_point() + geom_smooth(method = 'lm', se = F)
```



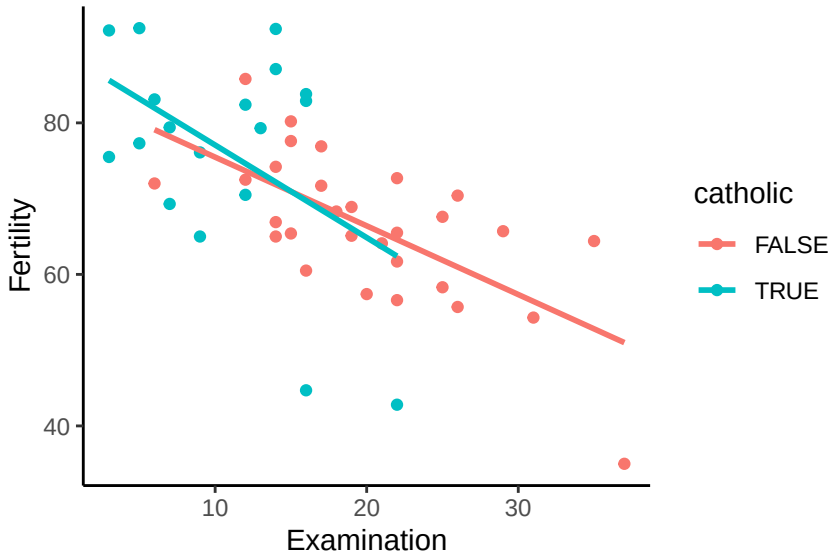
If we map `colour` to `catholic`, as we've seen, each point will colour code whether the corresponding province is Catholic or Protestant.

```
ggplot(swiss_df,
       mapping = aes(x = Examination, y = Fertility, colour = catholic)
) + geom_point()
```



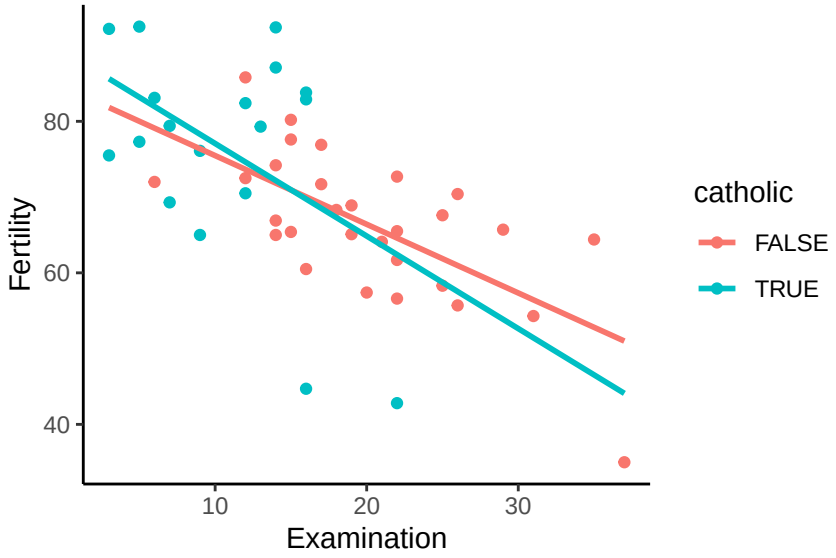
If we were to, in addition, add a smoothing curve or line of best fit, we would produce one for each of the two sets of points defined by the `catholic` variable.

```
ggplot(swiss_df,
       mapping = aes(x = Examination, y = Fertility, colour = catholic)
) + geom_point() + geom_smooth(method = 'lm', se = F)
```



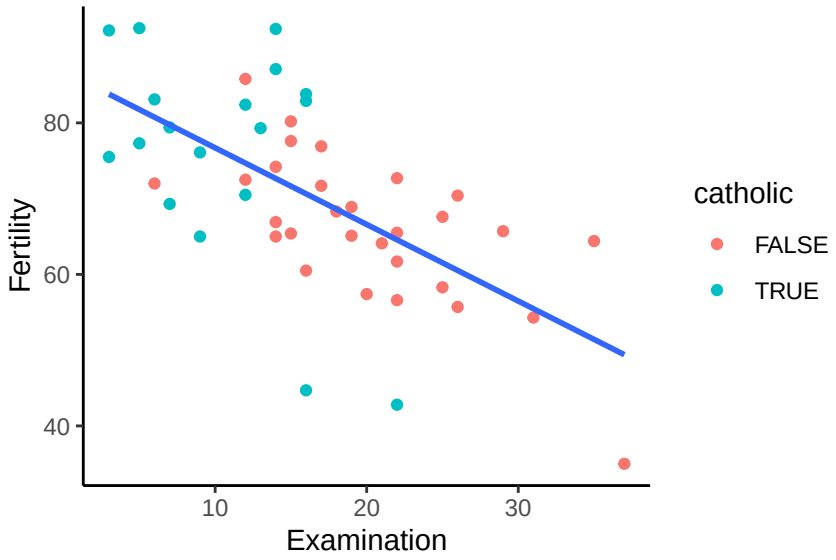
As can be seen, each line extends only over the range of its subset of points. However, we extend each line to full range of value by setting `fullrange` to `TRUE`.

```
ggplot(swiss_df,
       mapping = aes(x = Examination, y = Fertility, colour = catholic)
) + geom_point() + geom_smooth(method = 'lm', se = F, fullrange = T)
```



It may be that we would prefer to have a single line of best fit to all the points in the scatterplot while still keeping the points colour coded by a third variable. To do so, we need to restrict the `colour = catholic` aesthetic mapping to `geom_point()` as in the following example. As can be seen, each line extends only over the range of its subset of points. However, we extend each line to full range of value by setting `fullrange` to `TRUE`.

```
ggplot(swiss_df,
       mapping = aes(x = Examination, y = Fertility)
) + geom_point(mapping = aes(colour = catholic)) +
  geom_smooth(method = 'lm', se = F, fullrange = T)
```

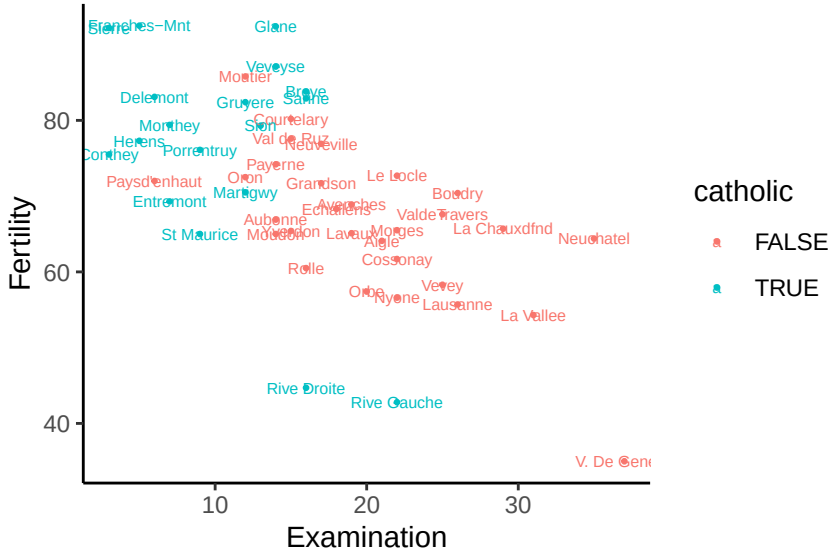


In the above example, both `geom_point` and `geom_smooth` inherited the global `aes` mapping. Then, `geom_point` added the additional mapping of `colour = catholic`.

Adding labels

We may use labels instead of, or in addition to, points in the scatterplot setting the `label` mapping and using `geom_text`.

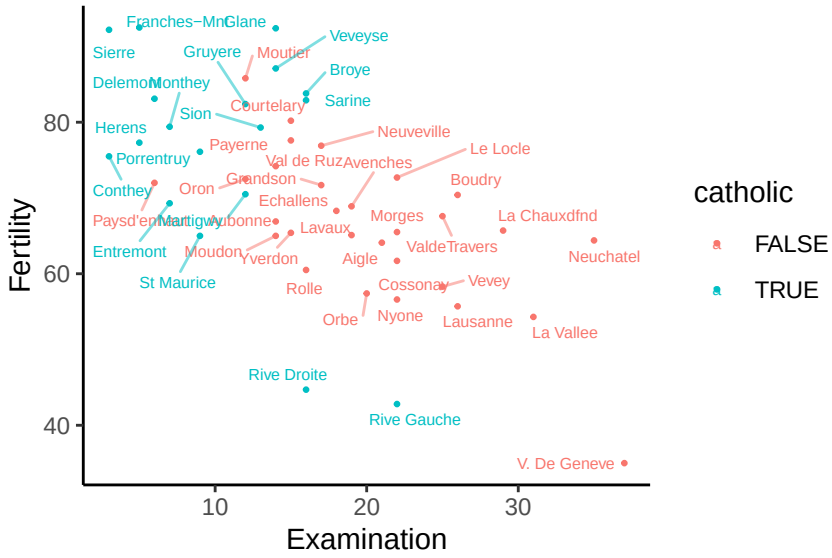
```
swiss_df %>%
  ggplot(mapping = aes(x = Examination, y = Fertility, label = province, colour = catholic)
         ) +
    geom_point(size = 0.5) +
  geom_text(size = 2)
```



In some cases, some labels overlap one another, and in all cases, the labels overlay the points. Using the `geom_text_repel` from the the `ggrepel` package in place of `geom_text` can overcome, or at least minimize, some of these problems.

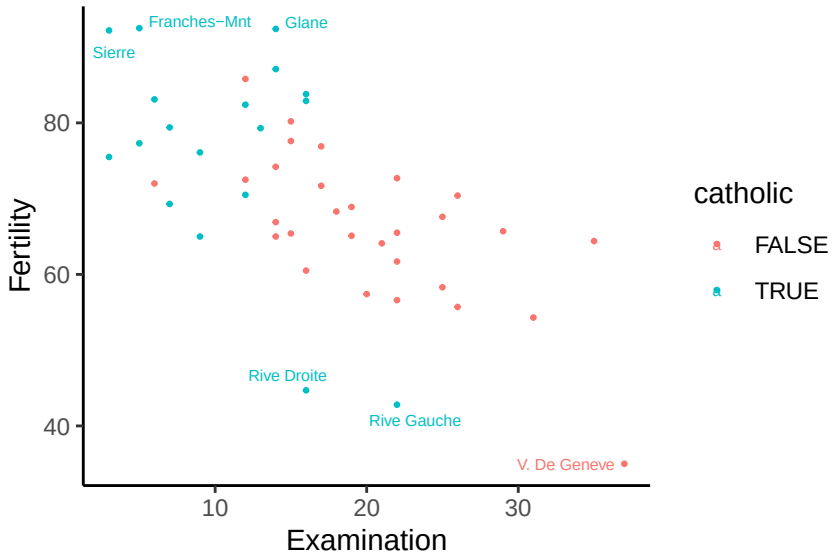
```
library(ggrepel)

swiss_df %>%
  ggplot(mapping = aes(x = Examination, y = Fertility, label = province, colour = catholic)
         ) +
    geom_point(size = 0.5) +
  geom_text_repel(size = 2, segment.alpha = 0.5)
```



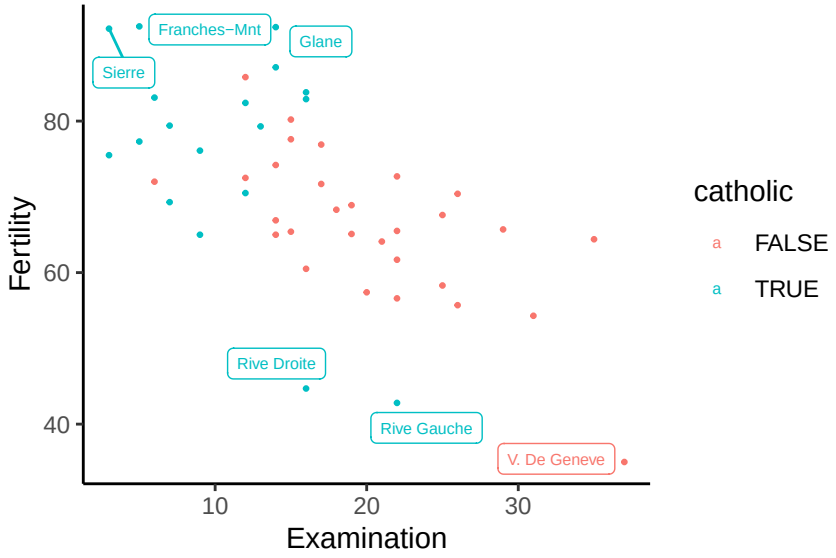
For scatterplots with a relatively large number of points, labeling all points may lead more to clutter than to clarity. However, if we are selective with which plots we label, then this may be very useful to draw attention to certain points, as in the following example, where we only label those provinces where fertility rates were very high or very low.

```
swiss_df %>%
  mutate(extreme = ifelse(Fertility < 50 | Fertility > 90, province, '')) %>%
  ggplot(mapping = aes(x = Examination, y = Fertility, label = extreme, colour = catholic)
         ) +
  geom_point(size = 0.5) +
  geom_text_repel(size = 2)
```



As an alternative to `geom_text` or `geom_text_repel`, we could use `geom_label` or `geom_label_repel`.

```
swiss_df %>%
  mutate(extreme = ifelse(Fertility < 50 | Fertility > 90, province, '')) %>%
  ggplot(mapping = aes(x = Examination, y = Fertility, label = extreme, colour = catholic)
         ) +
  geom_point(size = 0.5) +
  geom_label_repel(size = 2)
```

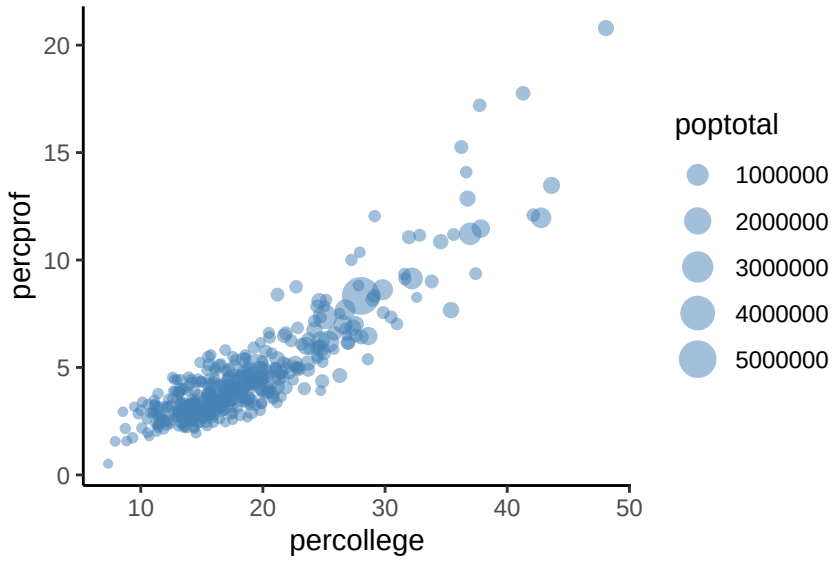


Bubbleplots

Bubbleplots are scatterplots where the size of the point is determined by the value of third variable. In the following, we plot the percentage of professional workers against the percentage of university educated people in different counties in the Mid West USA. We then scale the size of the point with the population.

```
midwest %>%
  ggplot(mapping = aes(x = percollege, y = percprof, size = poptotal)
```

```
        ) +
geom_point(alpha = 0.5, colour = 'steelblue')
```

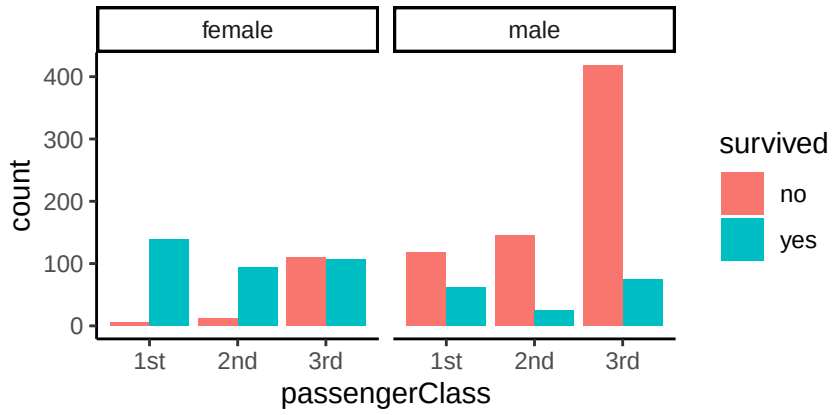


While any continuous variable could be mapped to `size`, because larger "bubbles" visually imply that the observation to which the point corresponds is also larger in some physical sense, it is perhaps generally advisable that `size` should map to a variable describing size in some sense, such as population or area, etc.

Facet plots

Facet plots allow us produce multiple related subplots, where each subplot displays some subset of the data. As an example, in the following, we plot the barplot of survival or not by both passenger class and by the sex of the passenger on the Titanic.

```
ggplot(titanic_df,
       mapping = aes(x = passengerClass, fill = survived)
) + geom_bar(position = 'dodge') +
  facet_wrap(~sex)
```



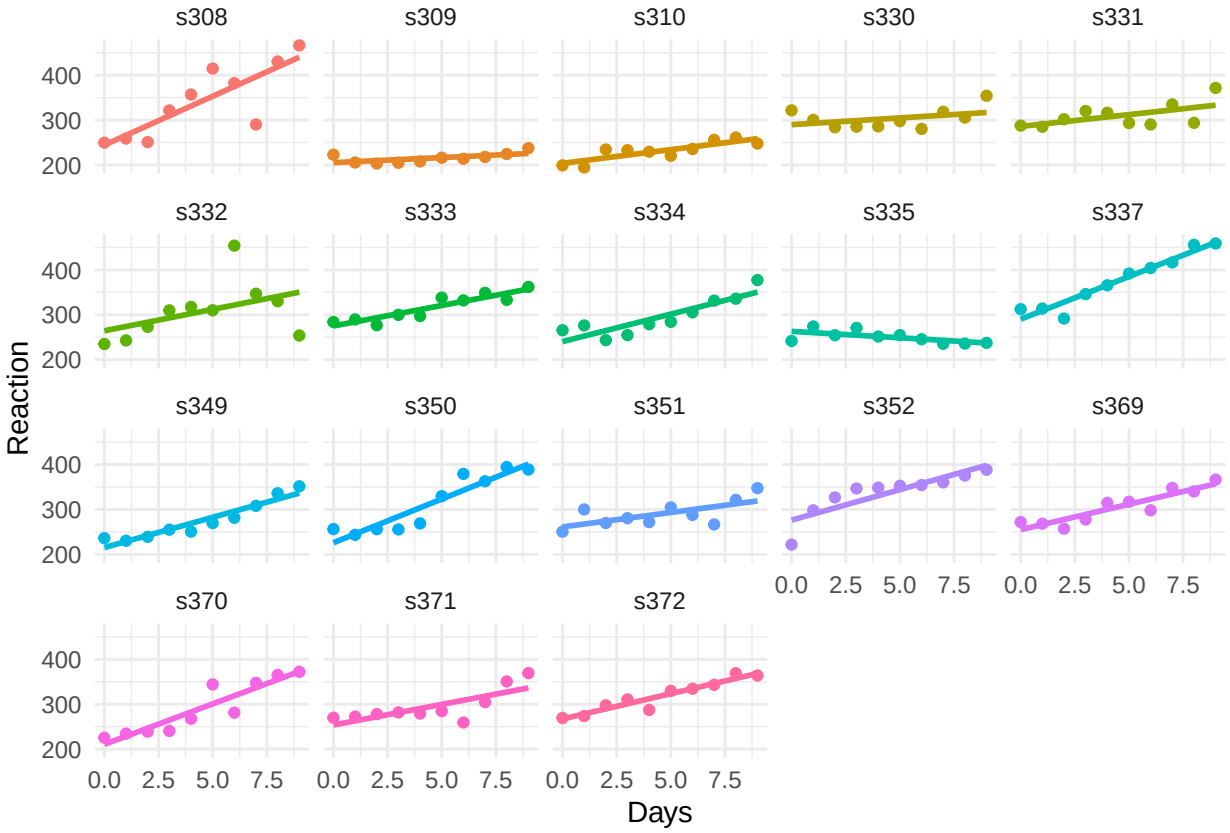
In this case, we simply added `facet_wrap(~sex)` as an additional statement at the end of the plot that produced the bar plot of survival rates by passenger class (see Page 18).

In cases where there are multiple subplots, `facet_wrap` will *wrap* the subplots. For example, in the following

plot, we produce one scatterplot with line of bestfit for each one of 18 subjects in an experiment. This data, available in `sleepstudy.csv`, was originally derived from a data set in the package `lme4`.

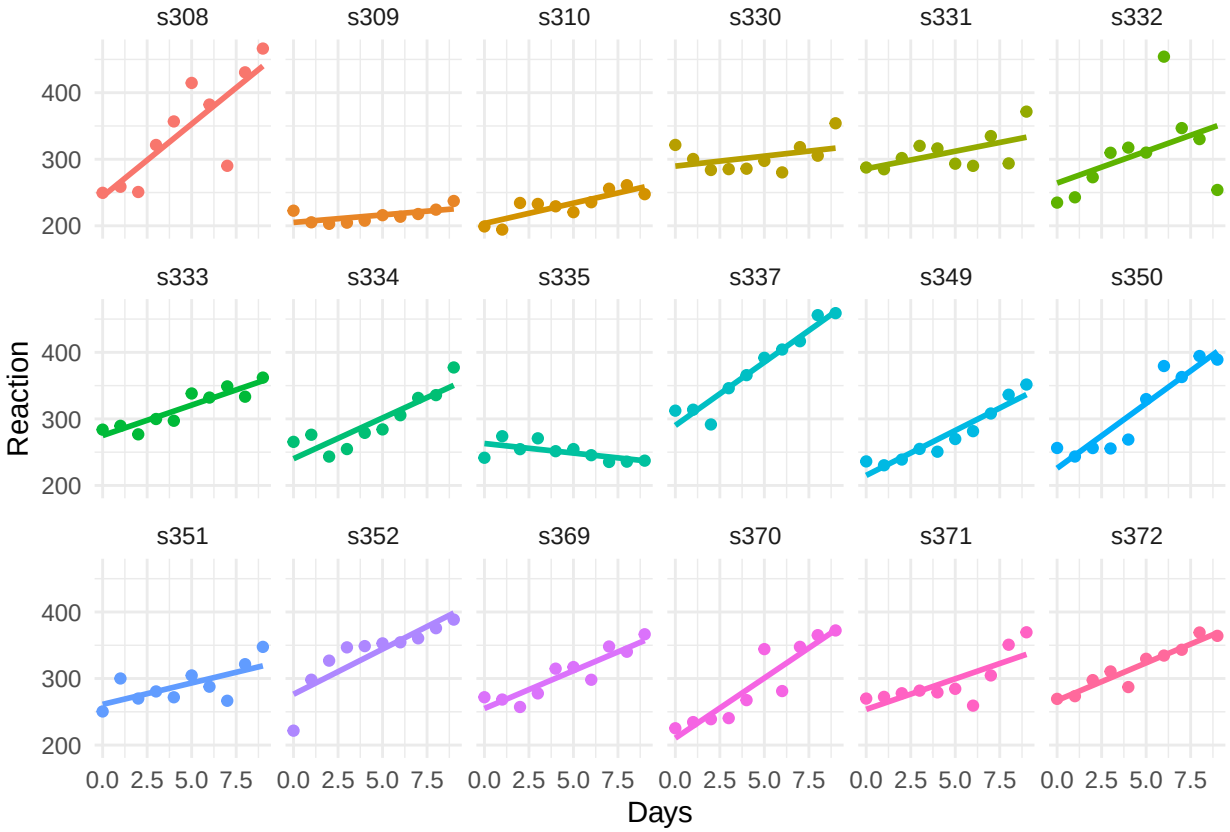
```
sleepstudy_df <- read_csv("data/sleepstudy.csv")

ggplot(sleepstudy_df,
       mapping = aes(x = Days, y = Reaction, colour = Subject)
) + geom_point() +
  geom_smooth(method = 'lm', se = F) +
  facet_wrap(~Subject) +
  theme_minimal() +
  theme(legend.position = 'none')
```



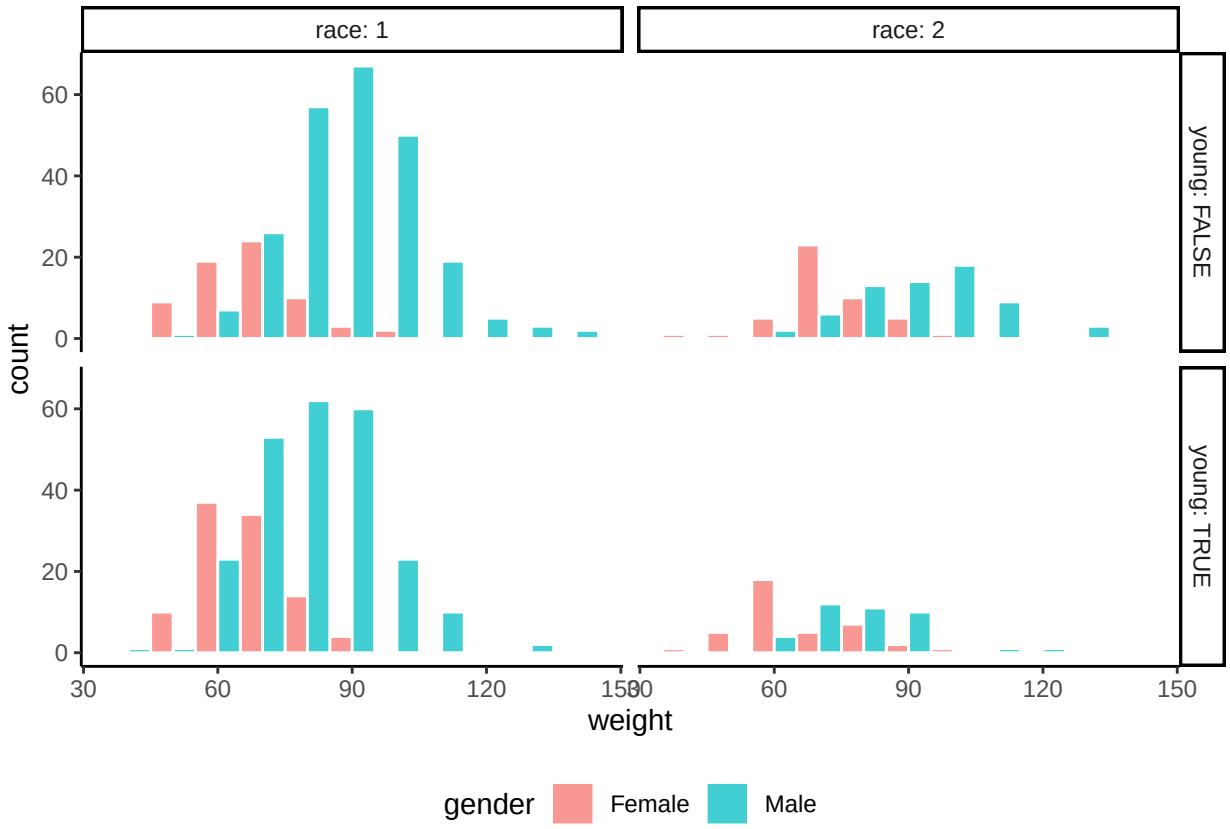
With `facet_wrap`, we may use `nrow` to specify the number of rows to use.

```
ggplot(sleepstudy_df,
       mapping = aes(x = Days, y = Reaction, colour = Subject)
) + geom_point() +
  geom_smooth(method = 'lm', se = F) +
  facet_wrap(~Subject, nrow = 3) +
  theme_minimal() +
  theme(legend.position = 'none')
```



We may facet by two, or more, discrete variables simultaneously using `facet_grid`, which will produce a grid with columns signifying values of one the faceting variables and the rows signifying the other. In the following example, we show histograms of weight for males and females who are either young (defined as below the median age) or not, and in one of two different racial categories.

```
weight_df %>%
  filter(race %in% c(1, 2)) %>%
  mutate(young = age < median(age),
         race = as.factor(race)) %>%
  ggplot(mapping = aes(x = weight, fill = gender)) +
  geom_histogram(binwidth = 10, colour = 'white', position = 'dodge', alpha = 0.75) +
  facet_grid(young ~ race, labeller = label_both) +
  theme(legend.position = 'bottom')
```



References

Anscombe, Francis J. 1973. "Graphs in Statistical Analysis." *The American Statistician* 27 (1): 17–21.

Hartwig, Frederick, and Brian E Dearing. 1979. *Exploratory Data Analysis*. Sage.

Healy, Kieran. 2019. *Data Visualization: A Practical Introduction*. Princeton University Press.

Tufte, E. R. 1983. *The Visual Display of Quantitative Information*. Graphics Press.

Tukey, J. W. 1977. *Exploratory Data Analysis*. Addison-Wesley Publishing Company.

Weissgerber, Tracey L., Natasa M. Milic, Stacey J. Winham, and Vesna D. Garovic. 2015. "Beyond Bar and Line Graphs: Time for a New Data Presentation Paradigm." *PLOS Biology* 13 (4): 1–10.

Wickham, Hadley. 2010. "A Layered Grammar of Graphics." *Journal of Computational and Graphical Statistics* 19 (1): 3–28. https://doi.org/10.1198/jcgs.2009.07098.

———. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

Wilkinson, Leland. 2005. *The Grammar of Graphics*. Berlin, Heidelberg: Springer-Verlag.